Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería Informática

Grado en Ingeniería Informática

Asignatura: Computación en la Nube

Curso Académico 2025/2026

Memoria de Práctica

Ingestión y Procesamiento de Datos de Consumo Energético

Autor: Sergio Acosta Quintana

Fecha: 7 de enero de 2026

Índice

1. Introducción

El objetivo principal de esta práctica es diseñar, desplegar y documentar una arquitectura completa de procesamiento de datos en streaming (tiempo real) utilizando los servicios de Amazon Web Services (AWS). El escenario planteado simula un entorno IoT donde múltiples dispositivos de control climático (HVAC) envían métricas de consumo energético, temperatura y voltaje de forma continua.

Para dar solución a este problema, hemos implementado un pipeline que conecta los siguientes servicios gestionados:

- **Amazon Kinesis Data Streams:** Actúa como punto de entrada de alta velocidad para la ingesta de datos.
- **Amazon Kinesis Data Firehose:** Se encarga de la entrega fiable de estos datos hacia el almacenamiento persistente.
- **AWS Lambda:** Realiza transformaciones ligeras "al vuelo" (ETL) antes del almacenamiento, específicamente para añadir particionado temporal.
- **Amazon S3:** Funciona como nuestro Data Lake, almacenando tanto los datos crudos como los procesados.
- **AWS Glue:** Permite catalogar los datos y ejecutar trabajos de análisis y agregación más complejos mediante Apache Spark.

Todo el despliegue de la infraestructura se ha automatizado mediante scripts de PowerShell, haciendo uso de la AWS CLI y la librería `boto3` de Python para la lógica de aplicación.

2. Desarrollo de las actividades

A continuación, se detalla el proceso técnico seguido para la implementación de cada componente de la arquitectura.

2.1. Configuración del bucket S3

El primer paso consistió en crear el almacenamiento centralizado (Data Lake). Se creó un bucket S3 con un nombre único basado en el ID de la cuenta para evitar colisiones de nombres globales en AWS.

Nombre del bucket generado: `datalake-energy-consumption-992382582640`

Dentro de este bucket, se estableció una estructura de carpetas jerárquica para mantener el orden y el ciclo de vida de los datos:

- `raw/`: Destino de los datos tal cual llegan de Firehose (particionados por fecha).
- `processed/`: Destino de los datos agregados y limpios generados por AWS Glue.
- `scripts/`: Repositorio de los scripts ETL de Python que utiliza Glue.
- `config/` y `errors/`: Carpetas para configuraciones temporales y registros de errores de procesamiento.

A continuación, se muestra la evidencia de la creación del bucket y su estructura en la consola de AWS:

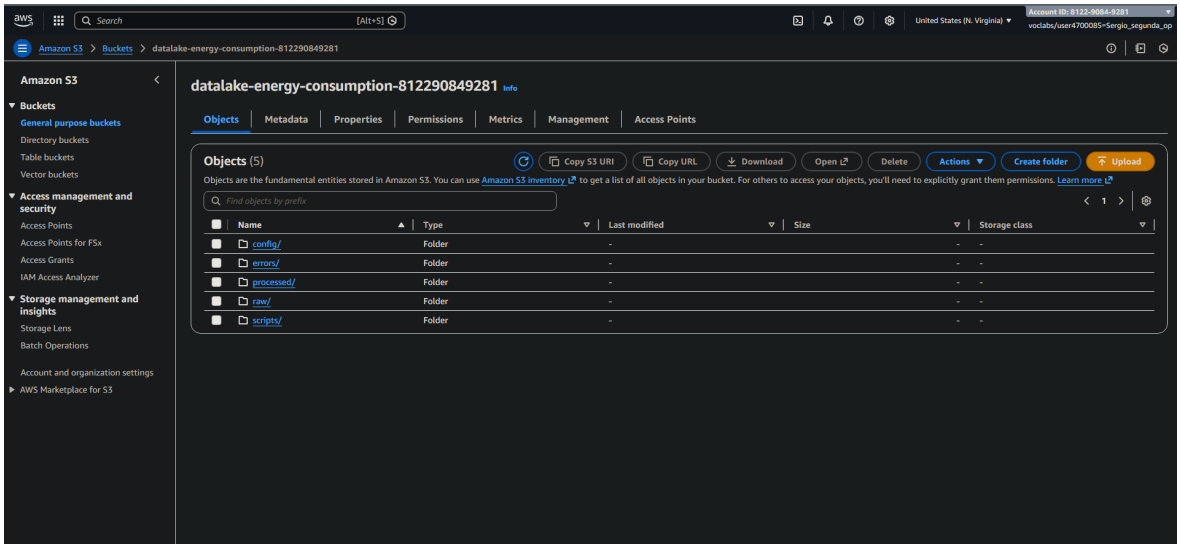

Figura 1: Consola de S3 mostrando la jerarquía de carpetas implementada para organizar el Data Lake.

Esta estructura nos permite separar claramente los datos "sucios" (raw) de los datos de valor "limpios" (processed), siguiendo las buenas prácticas de diseño de un Data Lake.

2.2. Implementación del productor de datos (Kinesis)

Para simular el comportamiento de los dispositivos IoT, utilicé un script en Python (`kinesis.py`). Este script lee un archivo JSON local (`mydata.json`) que contiene un dataset de lecturas simuladas de dispositivos HVAC.

El productor realiza las siguientes acciones lógicas:

1. Carga los datos estáticos del archivo JSON.
2. Itera sobre cada dispositivo y sus lecturas.
3. Aplana la estructura anidada del JSON para facilitar su análisis posterior (tabularización).
4. Envía cada registro individual al **Kinesis Data Stream** llamado `consumo-energetico-stream`.
5. Utiliza el ID del dispositivo como *Partition Key* para garantizar que los datos de un mismo sensor mantengan su orden secuencial.

A continuación se muestra el fragmento clave del código donde se realiza el envío a AWS:

```
# Fragmento representativo del código (ver Anexo A para código completo)
response = kinesis.put_record(
    StreamName=STREAM_NAME,
    Data=json.dumps(payload) + '\n',
    PartitionKey=dev_id
)
```

Listing 1: Fragmento del productor enviando datos a Kinesis

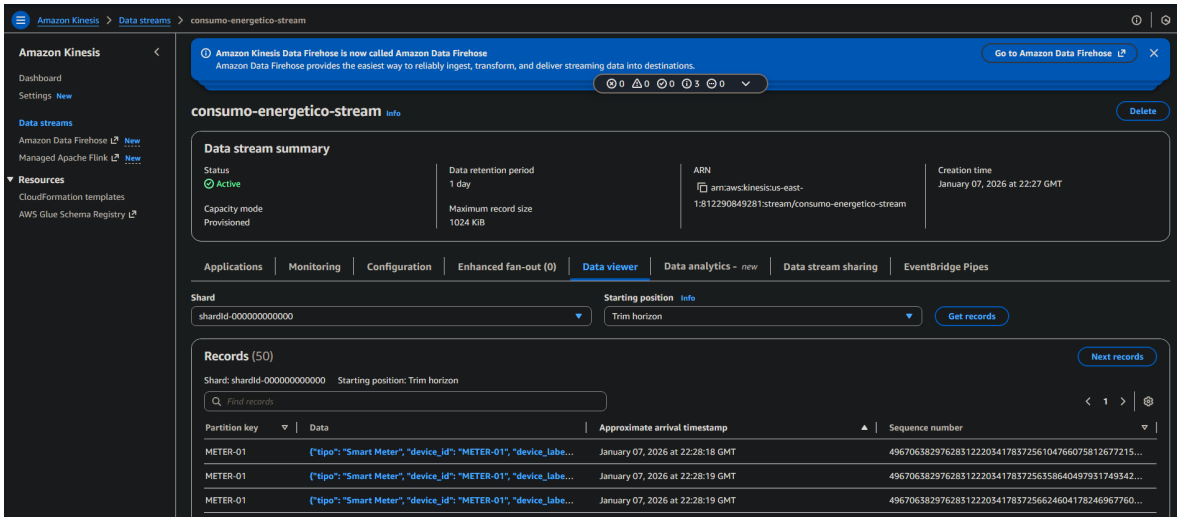

Figura 2: Visor de datos (Data Viewer) de Kinesis confirmando la recepción de registros JSON en tiempo real.

2.3. Configuración del consumidor (Firehose y Lambda)

Una vez los datos están fluyendo en el Stream, necesitamos persistirlos en S3. Para ello configuré un **Kinesis Data Firehose**. Sin embargo, antes de guardar, utilizamos una función **AWS Lambda** para procesar los datos en tiempo real.

La función Lambda (`energy_firehose_processor`) realiza una tarea crítica de transformación:

- Decodifica el registro entrante (base64).
- Añade un campo calculado `processing_date` basado en la fecha actual UTC.
- Vuelve a codificar el registro y lo devuelve a Firehose.

Esto permite a Firehose utilizar el particionado dinámico, guardando los archivos en S3 con la estructura `processing_date=YYYY-MM-DD`, lo que optimiza enormemente las consultas posteriores.

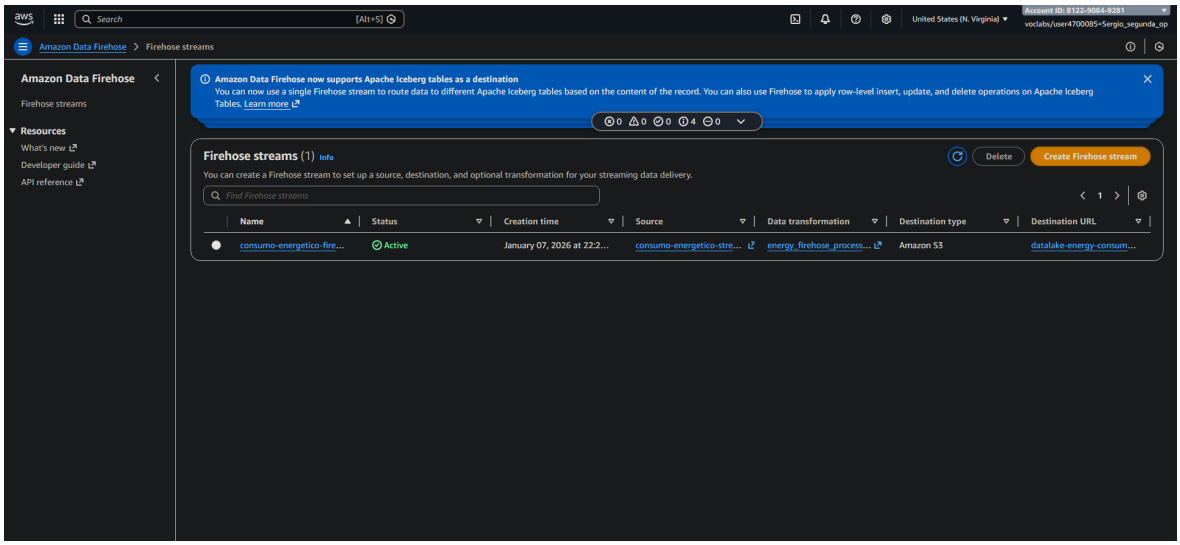

Figura 3: Detalle del Stream de entrega en Firehose mostrando la integración con Lambda y el destino S3.

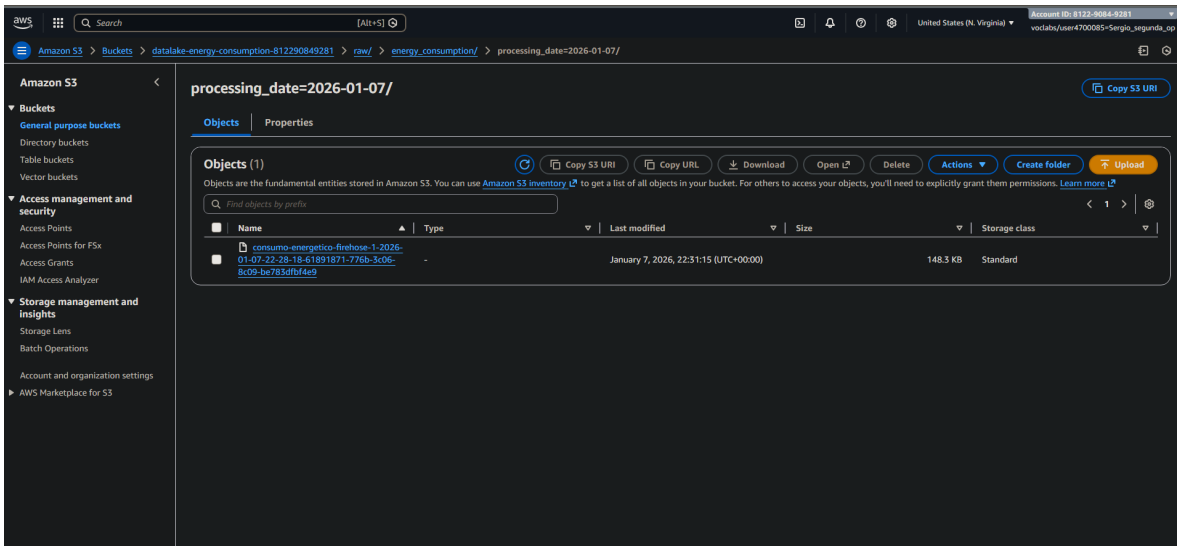

Figura 4: Verificación del almacenamiento en S3: los datos se han guardado respetando el particionado por fecha.

2.4. Configuración de AWS Glue

Con los datos ya almacenados en la carpeta `raw/`, el siguiente paso fue hacerlos consultables y agregar métricas de negocio.

Primero, configuré un **Crawler** de Glue (`energy_raw_crawler`). Este crawler escaneó el bucket S3 y creó automáticamente la tabla `energy_consumption` en la base de datos `energy_db` del Catálogo de Datos, infiriendo el esquema (columnas como voltaje, temperatura, etc.) y reconociendo las particiones.

Posteriormente, se crearon dos trabajos ETL (Jobs) en PySpark para procesar la información:

1. **Job Diario:** Agrupa los datos por día y tipo de dispositivo, calculando sumas de consumo y promedios de temperatura.

2. **Job Mensual:** Realiza una agregación similar pero a nivel mensual para informes de largo plazo.

Estos jobs leen del catálogo de datos, transforman los tipos de datos y escriben el resultado en formato **Parquet** (formato columnar comprimido optimizado para Big Data) en la carpeta `processed/`.

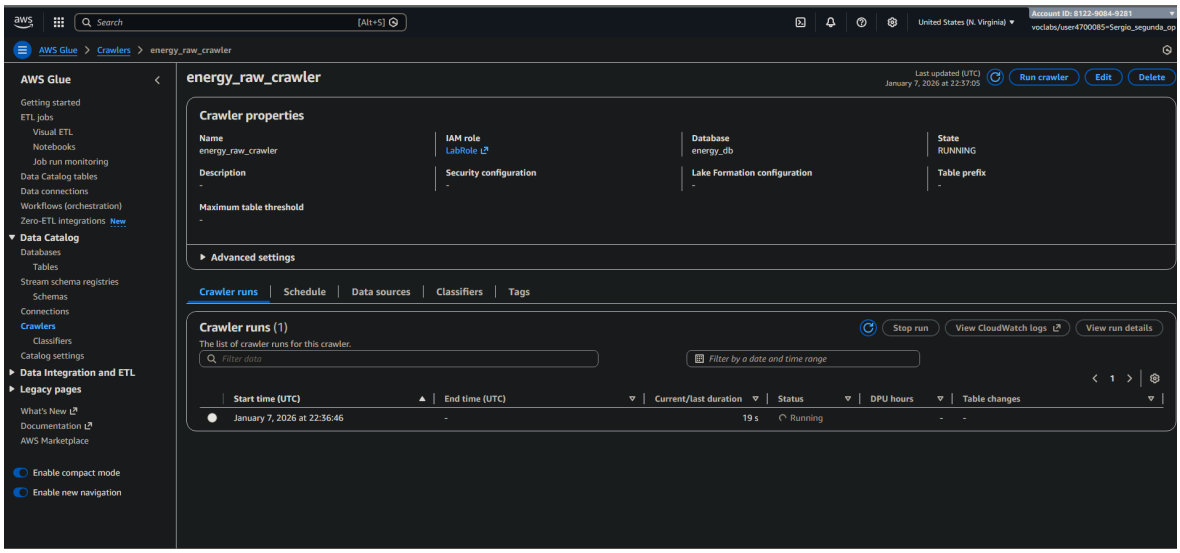

Figura 5: Propiedades del Crawler de Glue utilizado para inferir el esquema de los datos crudos.

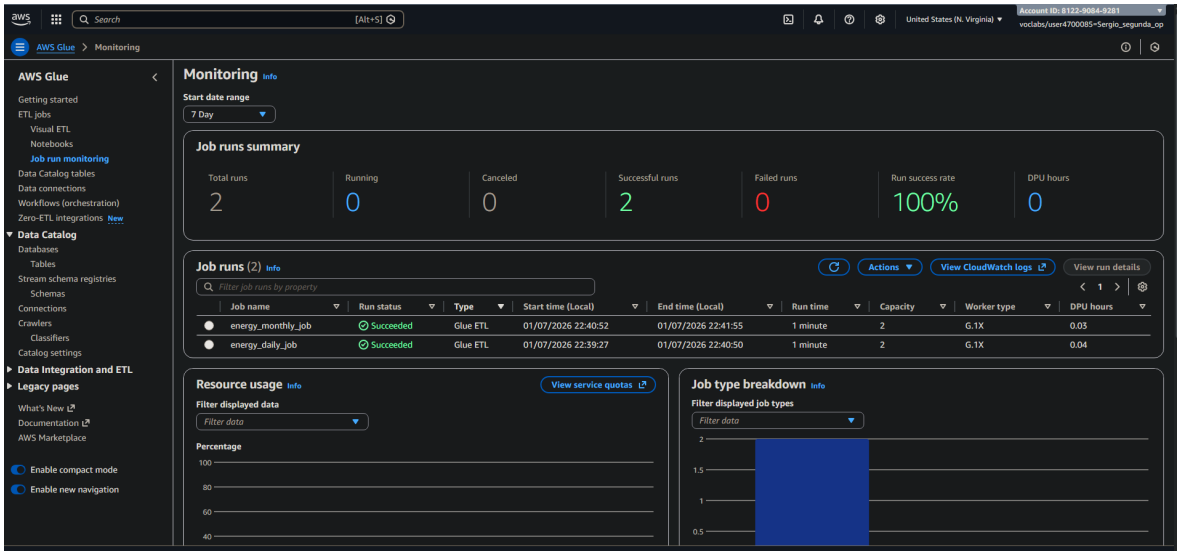

Figura 6: Panel de monitorización de Glue mostrando la ejecución exitosa (”Succeeded”) de los trabajos ETL.

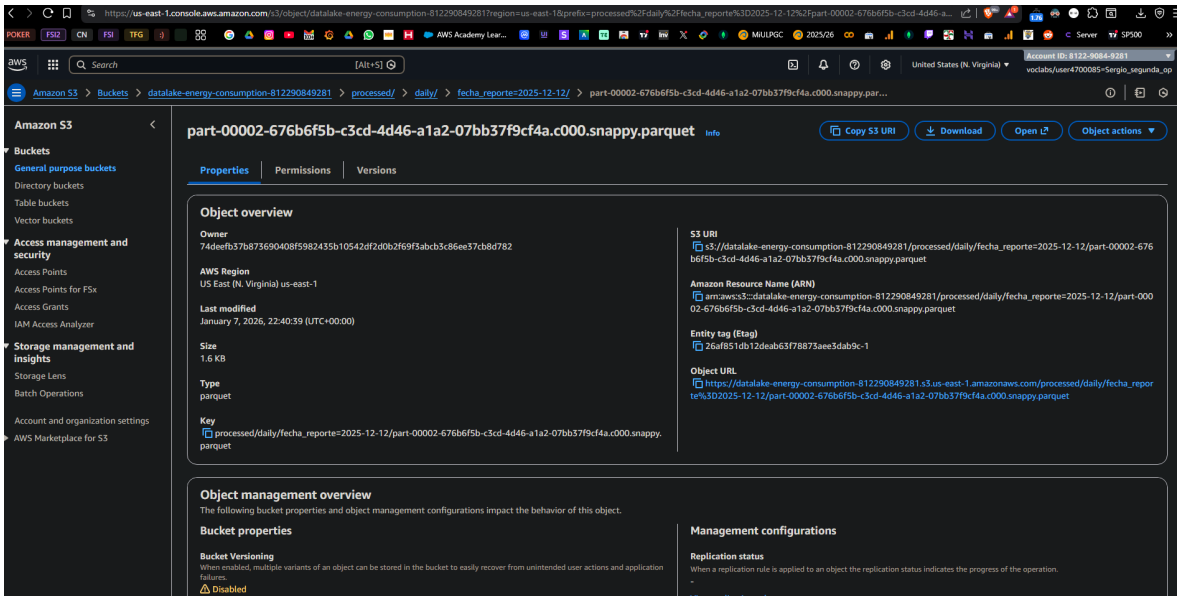

Figura 7: Resultado final: Archivo en formato Parquet generado en la carpeta ‘processed‘ tras la agregación.

2.5. Validación y Análisis de Resultados (Athena)

Una vez finalizados los trabajos ETL, verificamos la disponibilidad de los datos procesados en el Data Catalog. Como se observa en la siguiente figura, las tablas `daily` y `monthly` han sido creadas correctamente en el catálogo, apuntando a los datos en formato Parquet.

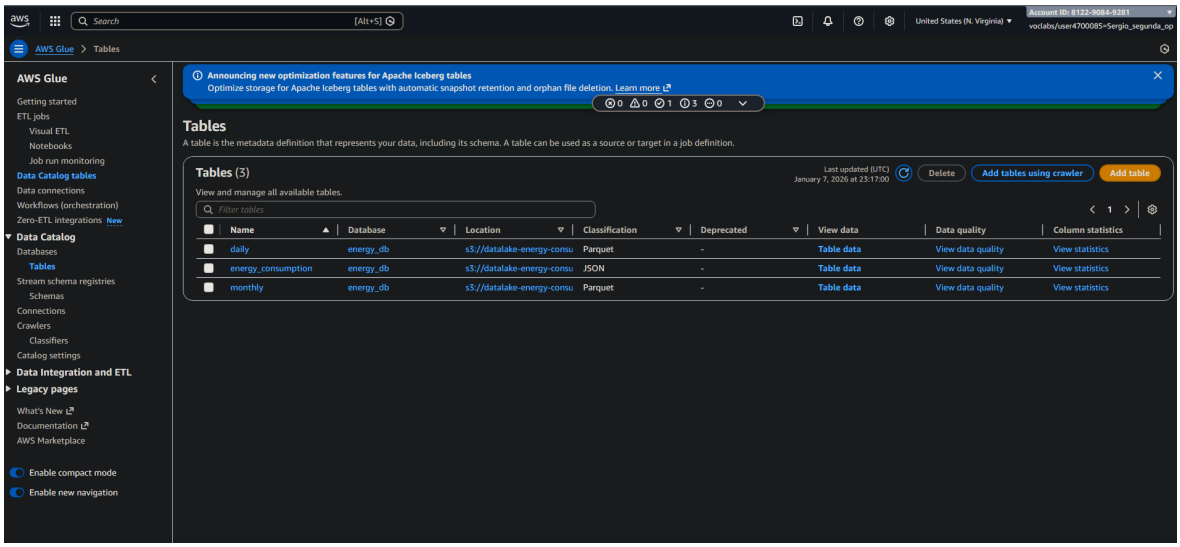

Figura 8: Tablas generadas en el AWS Glue Data Catalog (daily, monthly y energy_consumption).

Para validar la integridad de los datos y su utilidad de negocio, utilizamos **Amazon Athena** para realizar consultas SQL estándar directamente sobre los archivos resultantes en S3, sin necesidad de cargar servidores de base de datos.

A continuación, se muestra una consulta a la tabla `daily`, donde se observan las métricas consolidadas (suma de consumo, promedios de temperatura) agrupadas por día:

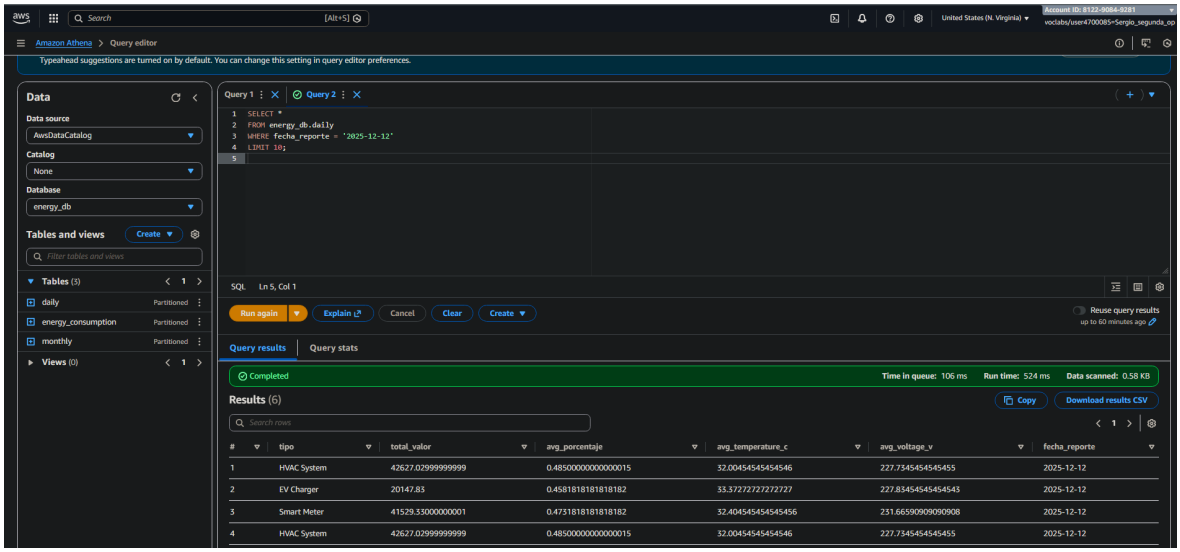

Figura 9: Consulta SQL en Athena validando los resultados de la agregación diaria.

Finalmente, validamos la agregación mensual, que proporciona una visión de alto nivel sobre el comportamiento de los dispositivos a largo plazo:

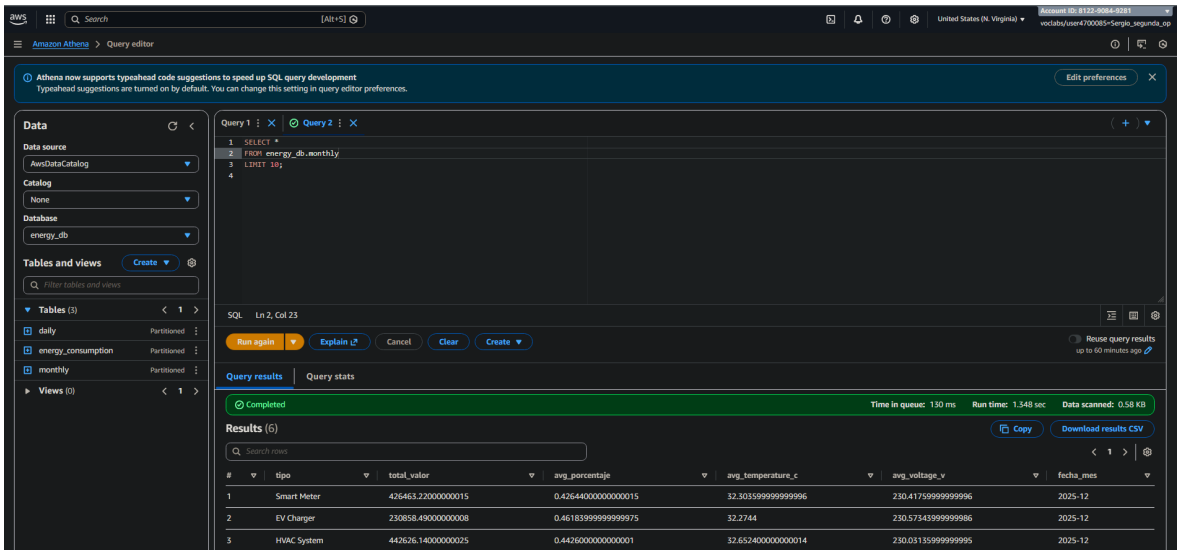

Figura 10: Consulta SQL en Athena visualizando las métricas mensuales finales.

3. Diagrama del flujo de datos

El diseño de la arquitectura sigue un flujo lineal y desacoplado, garantizando la escalabilidad. A continuación se detalla el flujo de la información, desde la generación en el productor Python hasta su análisis en el Data Catalog.

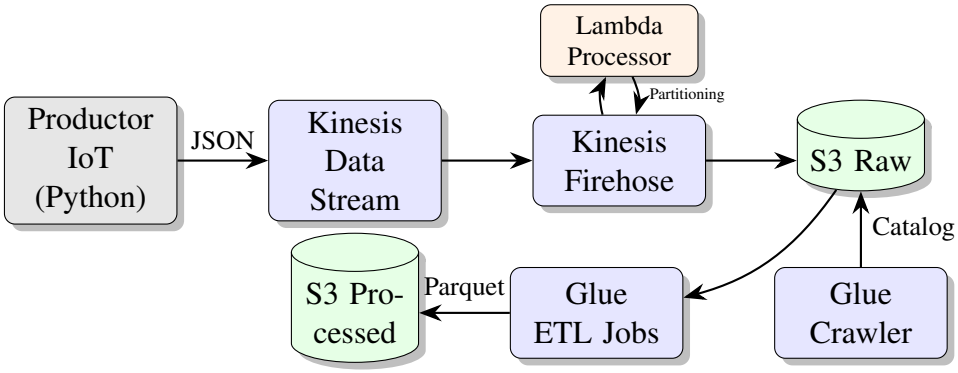

Figura 11: Arquitectura del Pipeline de Streaming Energético

4. Presupuesto y estimación de costes

A continuación se presenta una estimación de costes para un escenario de producción hipotético, asumiendo una ingestión continua de datos (24 horas al día, 30 días al mes) con un volumen moderado (aprox. 5 GB/mes). Los cálculos se realizan utilizando precios de lista (On-Demand) **sin aplicar la Capa Gratuita (Free Tier)** de AWS.

Para la elaboración de este presupuesto se ha utilizado la **Calculadora de Precios de AWS** como referencia principal.

Tabla 1: Estimación de costes mensuales y anuales

Servicio	**Supuestos y Configuración**	**Mes**	**Año**
Amazon Kinesis	1 Shard activo (0.015 USD/hora x 730h). El coste por PUT payload es despreciable para este volumen.	$10.95	$131.40
Kinesis Firehose	Ingesta de 5GB mensuales. Precio por GB ingerido ($0.029/GB).	$0.15	$1.80
AWS Lambda	1M invocaciones ($0.20) + Cómputo est. (0.40 USD). Sin aplicar capa gratuita.	$0.60	$7.20
Amazon S3	Almacenamiento Standard (10GB/mes: $0.23) + Peticiones PUT/GET (40k archivos: $0.22).	$0.45	$5.40
AWS Glue	Crawler + 2 Jobs ETL diarios (2 workers G.1X, 10 min c/u). Coste más variable.	$15.00	$180.00
TOTAL		**$27.15**	**$325.80**

Nota: Los costes de AWS Glue representan la mayor parte del presupuesto debido al coste por DPU-Hora. Para optimizar, se podrían utilizar "Flex Execution" o reducir la frecuencia de los jobs.

5. Conclusiones

La realización de esta práctica me ha permitido comprender en profundidad cómo se integran los diferentes servicios de AWS para construir una solución "Serverless" y escalable.

He aprendido que:

- **Desacoplamiento:** Kinesis es fundamental para separar la lógica de producción de datos de la de consumo, evitando cuellos de botella.
- **Transformación en tránsito:** El patrón Firehose + Lambda es extremadamente potente y económico para tareas sencillas de limpieza y particionado, evitando tener que levantar servidores complejos.
- **Gestión de Metadatos:** AWS Glue elimina la necesidad de gestionar manualmente los esquemas de bases de datos, aunque he notado que los tiempos de arranque de los Jobs de Spark pueden ser lentos para volúmenes de datos muy pequeños (overhead de inicio).

El uso del rol `LabRole` proporcionado en el entorno de laboratorio simplificó enormemente la gestión de permisos, permitiendo centrar el esfuerzo en la lógica de la arquitectura y el flujo de datos.

6. Referencias

Amazon Web Services. (2026). *Amazon Kinesis Data Firehose Developer Guide*.
https://docs.aws.amazon.com/firehose/

Amazon Web Services. (2026). *AWS Glue Developer Guide*.
https://docs.aws.amazon.com/glue/

7. Anexos

En este apartado se incluye la totalidad del código fuente desarrollado y utilizado para la práctica.

7.1. Productor de Datos (Python)

Archivo: `kinesis.py`

```python
import boto3
from loguru import logger
import time
import json
import os

# Constants
STREAM_NAME = 'consumo-energetico-stream'
INPUT_FILE = '../data/mydata.json'

kinesis = boto3.client('kinesis')

def load_data(path: str):
    if not os.path.exists(path):
        logger.error(f"File not found: {path}")
        raise FileNotFoundError(f"File not found: {path}")

    with open(path, 'r') as f:
        return json.load(f)

def run_producer():
    try:
        data = load_data(INPUT_FILE)
    except Exception as e:
        logger.error(str(e))
        return

    records_sent = 0
    devices_list = data.get('devices', [])

    logger.info(f"Starting transmission to {STREAM_NAME} using {
    INPUT_FILE}")

    for device in devices_list:
        # Extract device level info
        dev_type = device.get('type', 'Unknown')
        dev_id = device.get('id', 'Unknown')
        dev_data = device.get('data', {})
```

```
        dev_label = dev_data.get('label', 'Unknown')

        # Get readings list
        readings = dev_data.get('readings', [])

        logger.info(f"Processing device: {dev_label} ({dev_id}) - {len(
    readings)} readings")

        for reading in readings:
            # Construct flattened payload including new metrics
            payload = {
                # Keys used for Partitioning/Grouping
                'tipo': dev_type,
                'device_id': dev_id,
                'device_label': dev_label,

                # Metrics
                'valor': reading.get('value'),
                'porcentaje': reading.get('percentage'),
                'voltage': reading.get('voltage_v'),
                'current': reading.get('current_a'),
                'temperature': reading.get('temperature_c'),
                'status': reading.get('status'),

                # Timestamp
                'timestamp_origen': reading.get('timestamp')
            }

            # Send to Kinesis
            try:
                response = kinesis.put_record(
                    StreamName=STREAM_NAME,
                    Data=json.dumps(payload) + '\n',
                    PartitionKey=dev_id
                )

                records_sent += 1
                if records_sent % 10 == 0:
                    logger.info(f"Sent total {records_sent} records...")

                # Rate limiting to simulate streaming
                time.sleep(0.05)

            except Exception as e:
                logger.error(f"Failed to send record: {e}")

    logger.success(f"Transmission complete. Total records sent: {
```

```
    records_sent}")

if __name__ == "__main__":
    run_producer()
```

Listing 2: Script del productor de datos

7.2. Datos de Ejemplo

Archivo: `mydata.json`


```
{
    "devices": [
        {
            "type": "HVAC System",
            "id": "IOT-HEAT-01",
            "data": {
                "label": "Living Room Heater",
                "readings": [
                    {
                        "timestamp": "2025-12-12T13:00:00",
                        "value": 2903.6,
                        "percentage": 0.73,
                        "voltage_v": 235.3,
                        "current_a": 12.34,
                        "temperature_c": 21.4,
                        "status": "active"
                    },
                    {
                        "timestamp": "2025-12-12T14:00:00",
                        "value": 159.13,
                        "percentage": 0.04,
                        "voltage_v": 227.33,
                        "current_a": 0.7,
                        "temperature_c": 40.9,
                        "status": "active"
                    },
                    {
                        "timestamp": "2025-12-12T15:00:00",
                        "value": 2280.54,
                        "percentage": 0.57,
                        "voltage_v": 226.02,
                        "current_a": 10.09,
                        "temperature_c": 34.9,
                        "status": "active"
                    },
                    {
                        "timestamp": "2025-12-12T16:00:00",
                        "value": 2604.11,
                        "percentage": 0.65,
                        "voltage_v": 232.51,
                        "current_a": 11.2,
                        "temperature_c": 22.6,
                        "status": "active"
                    },
```

```
                {
                    "timestamp": "2025-12-12T17:00:00",
                    "value": 3356.92,
                    "percentage": 0.84,
                    "voltage_v": 226.36,
                    "current_a": 14.83,
                    "temperature_c": 40.2,
                    "status": "error"
                },
                {
                    "timestamp": "2025-12-12T18:00:00",
                    "value": 1260.92,
                    "percentage": 0.32,
                    "voltage_v": 220.44,
                    "current_a": 5.72,
                    "temperature_c": 26.2,
                    "status": "active"
                }
            ]
        }
    },
    {
        "type": "HVAC System",
        "id": "IOT-COOL-01",
        "data": {
            "label": "Bedroom AC",
            "readings": [
                {
                    "timestamp": "2025-12-12T13:00:00",
                    "value": 1974.85,
                    "percentage": 0.49,
                    "voltage_v": 227.78,
                    "current_a": 8.67,
                    "temperature_c": 24.3,
                    "status": "active"
                },
                {
                    "timestamp": "2025-12-12T14:00:00",
                    "value": 428.43,
                    "percentage": 0.11,
                    "voltage_v": 223.14,
                    "current_a": 1.92,
                    "temperature_c": 25.8,
                    "status": "active"
                },
                {
                    "timestamp": "2025-12-12T15:00:00",
```

```
                        "value": 2640.28,
                        "percentage": 0.66,
                        "voltage_v": 239.59,
                        "current_a": 11.02,
                        "temperature_c": 40.2,
                        "status": "active"
                    },
                    {
                        "timestamp": "2025-12-12T16:00:00",
                        "value": 997.92,
                        "percentage": 0.25,
                        "voltage_v": 221.76,
                        "current_a": 4.5,
                        "temperature_c": 38.3,
                        "status": "active"
                    },
                    {
                        "timestamp": "2025-12-12T17:00:00",
                        "value": 1905.64,
                        "percentage": 0.48,
                        "voltage_v": 237.02,
                        "current_a": 8.04,
                        "temperature_c": 20.2,
                        "status": "active"
                    }
                ]
            }
        }
    ]
}
```


Listing 3: Dataset de ejemplo

7.3. Procesador Lambda

Archivo: `firehose_processor.py`

```
import json
import base64
import datetime

def lambda_handler(event, context):
    """
    Decodes Firehose records, adds 'processing_date' for partitioning,
    and re-encodes data.
    """
    output = []

    for record in event['records']:
        try:
            # 1. Decode (base64 -> json)
            payload = base64.b64decode(record['data']).decode('utf-8')
            data_json = json.loads(payload)

            # 2. Logic: Extract date for partition key
            processing_time = datetime.datetime.now(datetime.timezone.utc
    )
            partition_date = processing_time.strftime('%Y-%m-%d')

            # 3. Prepare output
            # Firehose expects data re-encoded in base64
            output_record = {
                'recordId': record['recordId'],
                'result': 'Ok',
                'data': base64.b64encode((json.dumps(data_json) + '\n').
    encode('utf-8')).decode('utf-8'),
                'metadata': {
                    'partitionKeys': {
                        'processing_date': partition_date
                    }
                }
            }
            output.append(output_record)

        except Exception as e:
            # Send to error bucket if failed
            print(f"Error processing record: {e}")
            output_record = {
                'recordId': record['recordId'],
                'result': 'ProcessingFailed',
                'data': record['data']
```

```
            }
            output.append(output_record)

    return {'records': output}
```

Listing 4: Función Lambda para transformación

7.4. Scripts ETL de AWS Glue

Archivo: `energy_aggregation_daily.py`

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.functions import col, sum as spark_sum, avg, substring

# 1. Init
args = getResolvedOptions(sys.argv, ['JOB_NAME', 'database', 'table_name'
    , 'output_path'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

database = args['database']
table_name = args['table_name']
output_path = args['output_path']

# 2. Read from Catalog (Data from Crawler)
datasource = glueContext.create_dynamic_frame.from_catalog(
    database=database,
    table_name=table_name,
    transformation_ctx="datasource"
)

df = datasource.toDF()

# 3. Transform
# Extract date (YYYY-MM-DD) from timestamp
df_transformed = df.withColumn("fecha_reporte", substring(col("
    timestamp_origen"), 1, 10))

# Aggregate: Sum value, Avg percentage, Avg Temp, Avg Voltage by Date/
    Type
daily_agg = df_transformed.groupBy("fecha_reporte", "tipo") \
    .agg(
        spark_sum("valor").alias("total_valor"),
        avg("porcentaje").alias("avg_porcentaje"),
        avg("temperature").alias("avg_temperature_c"),
        avg("voltage").alias("avg_voltage_v")
    )
```

```
# Repartition for optimal writes
daily_agg = daily_agg.repartition("fecha_reporte")

# 4. Write to S3 (Parquet)
output_dyf = glueContext.create_dynamic_frame.from_catalog(
    database=database,
    table_name=table_name
).fromDF(daily_agg, glueContext, "output_dyf")

glueContext.write_dynamic_frame.from_options(
    frame=output_dyf,
    connection_type="s3",
    connection_options={
        "path": output_path,
        "partitionKeys": ["fecha_reporte"]
    },
    format="parquet",
    format_options={"compression": "snappy"},
    transformation_ctx="datasink"
)

job.commit()
```

Listing 5: Script ETL para agregación diaria

Archivo: `energy_aggregation_monthly.py`

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.functions import col, sum as spark_sum, avg, substring

# 1. Init
args = getResolvedOptions(sys.argv, ['JOB_NAME', 'database', 'table_name'
    , 'output_path'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

database = args['database']
table_name = args['table_name']
```

```
output_path = args['output_path']

# 2. Read from Catalog
datasource = glueContext.create_dynamic_frame.from_catalog(
    database=database,
    table_name=table_name,
    transformation_ctx="datasource"
)

df = datasource.toDF()

# 3. Transform (Monthly)
# Extract YYYY-MM
df_transformed = df.withColumn("fecha_mes", substring(col("
    timestamp_origen"), 1, 7))

# Aggregate by Month/Type
monthly_agg = df_transformed.groupBy("fecha_mes", "tipo") \
    .agg(
        spark_sum("valor").alias("total_valor"),
        avg("porcentaje").alias("avg_porcentaje"),
        avg("temperature").alias("avg_temperature_c"),
        avg("voltage").alias("avg_voltage_v")
    )

monthly_agg = monthly_agg.repartition("fecha_mes")

# 4. Write to S3
output_dyf = glueContext.create_dynamic_frame.from_catalog(
    database=database,
    table_name=table_name
).fromDF(monthly_agg, glueContext, "output_dyf")

glueContext.write_dynamic_frame.from_options(
    frame=output_dyf,
    connection_type="s3",
    connection_options={
        "path": output_path,
        "partitionKeys": ["fecha_mes"]
    },
    format="parquet",
    format_options={"compression": "snappy"},
    transformation_ctx="datasink"
)

job.commit()
```

Listing 6: Script ETL para agregación mensual

7.5. Scripts de Despliegue (PowerShell)

Archivo: `00_run_all.ps1`

```
# --- LOGGING SETUP ---
$LogDir = "$PSScriptRoot/../logs"
if (-not (Test-Path $LogDir)) { New-Item -ItemType Directory -Path
    $LogDir | Out-Null }
$LogFile = "$LogDir/deploy_$(Get-Date -Format 'yyyyMMdd_HHmm').log"

# Start recording all output (PS + Python) to file
Start-Transcript -Path $LogFile

Write-Host "=== PIPELINE DEPLOYMENT ===" -ForegroundColor Cyan

# 1. Load config
. "$PSScriptRoot/config/variables.ps1"

# 2. Setup S3
Write-Host "`n[1/5] Setting up S3..." -ForegroundColor Yellow
. "$PSScriptRoot/01_s3.ps1"

# 3. Setup Kinesis
Write-Host "`n[2/5] Setting up Kinesis..." -ForegroundColor Yellow
. "$PSScriptRoot/02_kinesis.ps1"

# 4. Setup Firehose
Write-Host "`n[3/5] Setting up Firehose..." -ForegroundColor Yellow
. "$PSScriptRoot/03_firehose.ps1"

# 5. Run Producer
Write-Host "`n[4/5] Generating Data & Syncing..." -ForegroundColor Yellow

# Move to python producer directory
Push-Location "$PSScriptRoot/../src/producer"

try {
    # Check for mydata.json
    if (-not (Test-Path "../data/mydata.json")) {
        throw "Data file '../data/mydata.json' not found. Please ensure
   your custom data file exists."
    }

    Write-Host "Installing requirements..."
    python -m pip install -r requirements.txt | Out-Null

    Write-Host "Running Python Producer with mydata.json..."
    python kinesis.py
```

```
}
catch {
    Write-Host "Error running Python script: $_" -ForegroundColor Red
    Pop-Location
    Stop-Transcript
    exit 1
}

# Return to scripts dir
Pop-Location

# Wait for Firehose buffer (60s buffer + margin)
Write-Host "Waiting 70s for Firehose buffer to flush to S3..." -
   ForegroundColor Magenta
Start-Sleep -Seconds 70

# 6. Setup Glue
Write-Host "`n[5/5] Setting up Glue..." -ForegroundColor Yellow
. "$PSScriptRoot/04_glue.ps1"

Write-Host "`nPipeline deployed successfully." -ForegroundColor Green
Write-Host "Logs saved to: $LogFile" -ForegroundColor Gray

# Stop logging
Stop-Transcript
```

Listing 7: 00_run_all.ps1 - Orquestador principal

Archivo: `01_s3.ps1`

```
. "$PSScriptRoot/config/variables.ps1"

Write-Host "Creating S3 Bucket and folders..."

# 1. Create Bucket
aws s3 mb "s3://$env:BUCKET_NAME"

# 2. Create folders (keys)
aws s3api put-object --bucket $env:BUCKET_NAME --key raw/
aws s3api put-object --bucket $env:BUCKET_NAME --key processed/
aws s3api put-object --bucket $env:BUCKET_NAME --key config/
aws s3api put-object --bucket $env:BUCKET_NAME --key scripts/
aws s3api put-object --bucket $env:BUCKET_NAME --key errors/
```

Listing 8: 01_s3.ps1 - Creación de Bucket S3

Archivo: `02_kinesis.ps1`

```
. "$PSScriptRoot/config/variables.ps1"

Write-Host "Creating Kinesis Data Stream..."

# 1. Create Stream
aws kinesis create-stream --stream-name "consumo-energetico-stream" --
    shard-count 1
```

Listing 9: 02_kinesis.ps1 - Creación del Data Stream

Archivo: `03_firehose.ps1`

```
. "$PSScriptRoot/config/variables.ps1"

Write-Host "=== CONFIGURING FIREHOSE + LAMBDA ===" -ForegroundColor Cyan

# --- 1. PREPARE LAMBDA ---
Write-Host "`n[1/3] Deploying Processor Lambda..." -ForegroundColor
    Yellow

$LAMBDA_NAME = "energy_firehose_processor"
$ZIP_PATH = "$PSScriptRoot/../src/lambda/function.zip"

# Zip Python code
Write-Host "Zipping code..."
if (Test-Path $ZIP_PATH) { Remove-Item $ZIP_PATH }
Push-Location "$PSScriptRoot/../src/lambda/"
Compress-Archive -Path "firehose_processor.py" -DestinationPath "function
    .zip" -Force
Move-Item "function.zip" $ZIP_PATH -Force
Pop-Location

# Delete old function if exists
aws lambda delete-function --function-name $LAMBDA_NAME 2>&1 | Out-Null

# Create Function
Write-Host "Creating Lambda..."
aws lambda create-function `
    --function-name $LAMBDA_NAME `
    --runtime python3.9 `
    --role $env:ROLE_ARN `
    --handler firehose_processor.lambda_handler `
    --zip-file "fileb://$ZIP_PATH" `
    --timeout 60 `
    --memory-size 128 | Out-Null

# Get Lambda ARN
$LAMBDA_ARN = (aws lambda get-function --function-name $LAMBDA_NAME --
    query 'Configuration.FunctionArn' --output text).Trim()
Write-Host "Lambda ARN: $LAMBDA_ARN" -ForegroundColor Green

# Wait for propagation
Start-Sleep -Seconds 5

# --- 2. CREATE FIREHOSE ---
Write-Host "`n[2/3] Creating Firehose..." -ForegroundColor Yellow
$DELIVERY_STREAM_NAME = "consumo-energetico-firehose"
```

```
# Cleanup old stream (Silence error if not exists)
aws firehose delete-delivery-stream --delivery-stream-name
    $DELIVERY_STREAM_NAME 2>&1 | Out-Null
Start-Sleep -Seconds 5

# Create Temp Config JSON
$FIREHOSE_CONFIG_JSON = @"
{
    "BucketARN": "arn:aws:s3:::$($env:BUCKET_NAME)",
    "RoleARN": "$env:ROLE_ARN",
    "Prefix": "raw/energy_consumption/processing_date=!{
    partitionKeyFromLambda:processing_date}/",
    "ErrorOutputPrefix": "errors/!{firehose:error-output-type}/",
    "BufferingHints": { "SizeInMBs": 64, "IntervalInSeconds": 60 },
    "ProcessingConfiguration": {
        "Enabled": true,
        "Processors": [
            {
                "Type": "Lambda",
                "Parameters": [
                    { "ParameterName": "LambdaArn", "ParameterValue": "
    $LAMBDA_ARN" },
                    { "ParameterName": "BufferSizeInMBs", "ParameterValue
    ": "1" },
                    { "ParameterName": "BufferIntervalInSeconds", "
    ParameterValue": "60" }
                ]
            }
        ]
    },
    "DynamicPartitioningConfiguration": {
        "Enabled": true,
        "RetryOptions": { "DurationInSeconds": 300 }
    }
}
"@

$CONFIG_FILE = "$PSScriptRoot/firehose_config.json"
$FIREHOSE_CONFIG_JSON | Out-File -FilePath $CONFIG_FILE -Encoding ASCII

# Create Stream
aws firehose create-delivery-stream `
    --delivery-stream-name $DELIVERY_STREAM_NAME `
    --delivery-stream-type KinesisStreamAsSource `
    --kinesis-stream-source-configuration "KinesisStreamARN=arn:aws:
    kinesis:$($env:AWS_REGION):$($env:ACCOUNT_ID):stream/consumo-
    energetico-stream,RoleARN=$env:ROLE_ARN" `
```

```
    --extended-s3-destination-configuration "file://$CONFIG_FILE"

if ($?) { Write-Host "Firehose created." -ForegroundColor Green }
else { Write-Host "Firehose creation failed." -ForegroundColor Red }

# Cleanup temp file
Remove-Item $CONFIG_FILE -ErrorAction SilentlyContinue
```

Listing 10: 03_firehose.ps1 - Configuración de Firehose y Lambda

Archivo: `04_glue.ps1`

```
. "$PSScriptRoot/config/variables.ps1"

Write-Host "=== CONFIGURING AWS GLUE (SEQUENTIAL MODE) ===" -
    ForegroundColor Cyan

# Helper function to create temp JSON files
function New-TempJson {
    param($Content)
    $Path = [System.IO.Path]::GetTempFileName()
    $Content | ConvertTo-Json -Depth 5 -Compress | Out-File -FilePath
    $Path -Encoding ASCII
    return $Path
}

# Función para esperar a que un Job termine
function Wait-ForJob {
    param($JobName, $RunId)

    Write-Host "Waiting for job $JobName (Run ID: $RunId) to finish..." -
    ForegroundColor Magenta

    do {
        Start-Sleep -Seconds 15
        $Status = (aws glue get-job-run --job-name $JobName --run-id
    $RunId --query 'JobRun.JobRunState' --output text).Trim()
        Write-Host "  -> $JobName Status: $Status" -ForegroundColor Gray
    } while ($Status -in "STARTING", "RUNNING", "STOPPING")

    return $Status
}

# 1. Upload ETL Scripts
Write-Host "`n[1/5] Uploading Spark scripts..."
aws s3 cp "$PSScriptRoot/energy_aggregation_daily.py" "s3://$env:
    BUCKET_NAME/scripts/energy_aggregation_daily.py"
```

```
aws s3 cp "$PSScriptRoot/energy_aggregation_monthly.py" "s3://$env:
    BUCKET_NAME/scripts/energy_aggregation_monthly.py"

# 2. Database
Write-Host "`n[2/5] Creating Database..."
# Use temp file to avoid PowerShell quoting issues
$DbInput = @{ Name = "energy_db" }
$DbFile = New-TempJson -Content $DbInput
aws glue create-database --database-input "file://$DbFile" 2>&1 | Out-
    Null
Remove-Item $DbFile

# 3. Raw Data Crawler
Write-Host "Configuring Raw Data Crawler..."
aws glue delete-crawler --name energy_raw_crawler 2>&1 | Out-Null

$CrawlerTargets = @{
    S3Targets = @(
        @{ Path = "s3://$env:BUCKET_NAME/raw/energy_consumption/" }
    )
}
$CrawlerFile = New-TempJson -Content $CrawlerTargets

aws glue create-crawler `
    --name energy_raw_crawler `
    --role $env:ROLE_ARN `
    --database-name energy_db `
    --targets "file://$CrawlerFile"

Remove-Item $CrawlerFile

# 4. ETL Jobs Creation (Daily & Monthly)
Write-Host "`n[3/5] Creating ETL Jobs..."

# --- Daily Job Setup ---
aws glue delete-job --job-name energy_daily_job 2>&1 | Out-Null
$DailyScript = "s3://$env:BUCKET_NAME/scripts/energy_aggregation_daily.py
    "
$DailyOut = "s3://$env:BUCKET_NAME/processed/daily/"

$DailyCommand = @{
    Name           = "glueetl"
    ScriptLocation = $DailyScript
    PythonVersion  = "3"
}
$DailyArgs = @{
    "--database"     = "energy_db"
```

```
    "--table_name"   = "energy_consumption"
    "--output_path"  = $DailyOut
    "--job-language" = "python"
}

$DailyCmdFile = New-TempJson -Content $DailyCommand
$DailyArgsFile = New-TempJson -Content $DailyArgs

aws glue create-job `
    --name energy_daily_job `
    --role $env:ROLE_ARN `
    --command "file://$DailyCmdFile" `
    --default-arguments "file://$DailyArgsFile" `
    --glue-version "4.0" `
    --number-of-workers 2 `
    --worker-type "G.1X"

Remove-Item $DailyCmdFile, $DailyArgsFile

# --- Monthly Job Setup ---
aws glue delete-job --job-name energy_monthly_job 2>&1 | Out-Null
$MonthlyScript = "s3://$env:BUCKET_NAME/scripts/
   energy_aggregation_monthly.py"
$MonthlyOut = "s3://$env:BUCKET_NAME/processed/monthly/"

$MonthlyCommand = @{
    Name           = "glueetl"
    ScriptLocation = $MonthlyScript
    PythonVersion  = "3"
}
$MonthlyArgs = @{
    "--database"     = "energy_db"
    "--table_name"   = "energy_consumption"
    "--output_path"  = $MonthlyOut
    "--job-language" = "python"
}

$MonthlyCmdFile = New-TempJson -Content $MonthlyCommand
$MonthlyArgsFile = New-TempJson -Content $MonthlyArgs

aws glue create-job `
    --name energy_monthly_job `
    --role $env:ROLE_ARN `
    --command "file://$MonthlyCmdFile" `
    --default-arguments "file://$MonthlyArgsFile" `
    --glue-version "4.0" `
    --number-of-workers 2 `
```

```
    --worker-type "G.1X"

Remove-Item $MonthlyCmdFile, $MonthlyArgsFile

# 5. Run Crawler & Trigger Jobs SEQUENTIALLY
Write-Host "`n[4/5] Starting Raw Crawler..."
aws glue start-crawler --name energy_raw_crawler

Write-Host "Waiting for Raw Crawler to finish cataloging..." -
    ForegroundColor Magenta
do {
    Start-Sleep -Seconds 15
    $CrawlerStatus = (aws glue get-crawler --name energy_raw_crawler --
    query 'Crawler.State' --output text).Trim()
    Write-Host "Crawler Status: $CrawlerStatus" -ForegroundColor Gray
} while ($CrawlerStatus -ne "READY")

# --- SEQUENTIAL EXECUTION START ---
Write-Host "`n[SEQUENTIAL EXECUTION] Starting Daily Job first..." -
    ForegroundColor Yellow
$DailyRunId = (aws glue start-job-run --job-name energy_daily_job --query
    'JobRunId' --output text).Trim()

# Wait for Daily Job
$DailyStatus = Wait-ForJob -JobName "energy_daily_job" -RunId $DailyRunId

if ($DailyStatus -eq "SUCCEEDED") {
    Write-Host "Daily Job SUCCEEDED. Starting Monthly Job..." -
    ForegroundColor Green

    $MonthlyRunId = (aws glue start-job-run --job-name energy_monthly_job
    --query 'JobRunId' --output text).Trim()

    # Wait for Monthly Job
    $MonthlyStatus = Wait-ForJob -JobName "energy_monthly_job" -RunId
    $MonthlyRunId

    if ($MonthlyStatus -eq "SUCCEEDED") {
        Write-Host "All ETL Jobs completed successfully." -
    ForegroundColor Green

        # --- NEW SECTION: PROCESSED DATA CRAWLER ---
        Write-Host "`n[5/5] Configuring & Running Final Crawler for
    Processed Data..." -ForegroundColor Cyan

        # Delete if exists to be clean
        aws glue delete-crawler --name energy_processed_crawler 2>&1 |
```

```
    Out-Null

        $ProcessedCrawlerTargets = @{
            S3Targets = @(
                @{ Path = "s3://$env:BUCKET_NAME/processed/" }
            )
        }
        $ProcCrawlerFile = New-TempJson -Content $ProcessedCrawlerTargets

        aws glue create-crawler `
            --name energy_processed_crawler `
            --role $env:ROLE_ARN `
            --database-name energy_db `
            --targets "file://$ProcCrawlerFile"

        Remove-Item $ProcCrawlerFile

        Write-Host "Starting Processed Data Crawler..."
        aws glue start-crawler --name energy_processed_crawler

        Write-Host "Waiting for Processed Crawler..." -ForegroundColor
    Magenta
        do {
            Start-Sleep -Seconds 15
            $ProcCrawlerStatus = (aws glue get-crawler --name
    energy_processed_crawler --query 'Crawler.State' --output text).Trim()
            Write-Host "Processed Crawler Status: $ProcCrawlerStatus" -
    ForegroundColor Gray
        } while ($ProcCrawlerStatus -ne "READY")

        Write-Host "`nPIPELINE FINISHED. Tables 'daily' and 'monthly'
    should now exist in Athena/Glue." -ForegroundColor Green
    }
    else {
        Write-Host "Monthly Job FAILED or STOPPED." -ForegroundColor Red
    }
}
else {
    Write-Host "Daily Job FAILED. Skipping Monthly Job to prevent errors/
    costs." -ForegroundColor Red
}
```

Listing 11: 04_glue.ps1 - Configuración de Glue y Crawler

Archivo: `99_cleanup.ps1`

```
. "$PSScriptRoot/config/variables.ps1"
```

```
Write-Host "CLEANING UP RESOURCES FOR ACCOUNT $env:ACCOUNT_ID..." -
    ForegroundColor Red

# 1. Glue
Write-Host "1. Deleting Glue Resources..."
aws glue delete-job --job-name energy_daily_job 2>$null
aws glue delete-job --job-name energy_monthly_job 2>$null
aws glue delete-crawler --name energy_raw_crawler 2>$null
aws glue delete-database --name energy_db 2>$null

# 2. Ingestion
Write-Host "2. Deleting Ingestion Resources..."
aws firehose delete-delivery-stream --delivery-stream-name "consumo-
    energetico-firehose" 2>$null
aws kinesis delete-stream --stream-name "consumo-energetico-stream" 2>
    $null
aws lambda delete-function --function-name "energy_firehose_processor" 2>
    $null

# 3. S3
Write-Host "3. Deleting Bucket..."
# Force delete removes all files inside first
aws s3 rb "s3://$env:BUCKET_NAME" --force 2>$null

Write-Host "Cleanup complete." -ForegroundColor Green
```

Listing 12: 99_cleanup.ps1 - Limpieza de recursos

Archivo: `config/variables.ps1`

```
# --- GLOBAL CONFIGURATION ---
$env:AWS_REGION = "us-east-1"
# Get AWS Account ID
$env:ACCOUNT_ID = (aws sts get-caller-identity --query Account --output
    text).Trim()
# Get LabRole ARN (Common in AWS Academy)
$env:ROLE_ARN = (aws iam get-role --role-name LabRole --query 'Role.Arn'
    --output text).Trim()
# Define unique bucket name
$env:BUCKET_NAME = "datalake-energy-consumption-$($env:ACCOUNT_ID)"

Write-Host "Config: Account $env:ACCOUNT_ID | Bucket $env:BUCKET_NAME"
```

Listing 13: config/variables.ps1 - Variables globales

7.6. Declaración de uso de IA

Para la realización de esta práctica, se han utilizado herramientas de Inteligencia Artificial Generativa (ChatGPT/Gemini) como apoyo para:

- La corrección sintáctica de los scripts.
- Tareas de debugging.
- La consulta de documentación técnica y librerías.
- Plantillas iniciales.