Universidad de las Palamas de Gran Canaria

Escuela de Ingeniería Informática

Grado en Ingeniería Informática

COMPUTACIÓN EN LA NUBE

Práctica 7

Data Lake de Vehículos con AWS

Autor:
Miguel Ángel Rodríguez Ruano

Curso Académico:
2025/2026

Fecha de Entrega:
15/01/2026

Índice

1 Introducción

Esta práctica tiene como objetivo el diseño e implementación de un Data Lake completo en AWS para la gestión y análisis de datos. El proyecto integra servicios de ingesta en tiempo real, almacenamiento distribuido y procesamiento ETL, demostrando la capacidad de la nube para manejar grandes volúmenes de datos de forma escalable y eficiente.

En este caso en concreto la arquitectura implementada permite transformar datos en bruto de vehículos usados (marca, modelo, kilometraje, año, tipo de combustible, etc.) en información analítica estructurada y optimizada, lista para consultas SQL mediante Amazon Athena. El sistema desarrollado utiliza los siguientes servicios de AWS:

- **Amazon Kinesis Data Streams**: Para la ingesta de datos en tiempo real, permitiendo recibir y procesar aproximadamente 36.000 registros de vehículos de forma continua mediante streaming.
- **Amazon Kinesis Data Firehose**: Como consumidor del stream de Kinesis, aplicando transformaciones mediante Lambda y entregando los datos en S3 con particionamiento dinámico por fecha.
- **AWS Lambda**: Para enriquecer y transformar los registros en tiempo real antes de su almacenamiento, añadiendo metadatos de procesamiento y generando claves de partición dinámicas.
- **Amazon S3**: Como almacenamiento central del Data Lake, organizado en capas (*raw*, *processed*, *scripts*, *errors*, *logs* y *queries*) siguiendo las mejores prácticas de arquitectura de datos en la nube.
- **AWS Glue Data Catalog**: Para la catalogación automática de esquemas mediante crawlers, manteniendo un repositorio centralizado de metadatos accesible por múltiples servicios.
- **AWS Glue ETL Jobs**: Para transformación de datos mediante trabajos PySpark, generando agregaciones por marca y año de fabricación en formato Parquet optimizado.
- **Amazon Athena**: Para ejecutar consultas SQL interactivas sobre los datos procesados, permitiendo análisis ad-hoc sin necesidad de infraestructura de bases de datos.

El despliegue completo de la infraestructura se ha automatizado mediante scripts en PowerShell, integrando AWS CLI para la orquestación de servicios. Para el desarrollo local del productor de datos, se configuró un entorno virtual Python gestionado con `uv`, incluyendo la librería `boto3` para la interacción programática con los servicios de AWS, así como `loguru` para registro detallado de eventos durante la ingesta de datos.

2 Desarrollo de las Actividades

2.1 Configuración del Bucket S3

El bucket S3 constituye el almacenamiento central del Data Lake, siguiendo una arquitectura de múltiples capas que separa datos crudos, procesados, recursos auxiliares y áreas de trabajo para diferentes servicios AWS.

2.1.1 Creación del Bucket

El bucket se creó con la siguiente nomenclatura: `datalake-cars-[ACCOUNT_ID]`, donde `ACCOUNT_ID` es el identificador único de la cuenta AWS. Esta convención garantiza unicidad global del nombre del bucket en todo el ecosistema de AWS y facilita la trazabilidad de recursos por cuenta.

```
aws s3 mb s3://datalake-cars-$ACCOUNT_ID --region us-east-1
```

Listing 1: Comando de creación del bucket S3 mediante AWS CLI

El bucket se desplegó en la región `us-east-1` (Virginia del Norte), como se observa en la Figura 1.

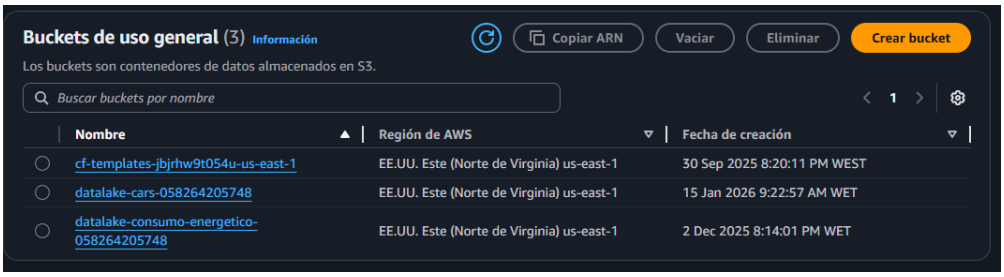

Figura 1: Bucket S3 creado en la consola de AWS

2.1.2 Estructura de Carpetas

Se implementó la siguiente jerarquía de carpetas mediante el comando `put-object` de AWS CLI:

- **raw/cars_data/**: Almacena los datos en bruto tal como llegan desde Kinesis Firehose, organizados con particionamiento dinámico por fecha de procesamiento (formato: `processing_date=YYYY-MM-DD`). Esta capa mantiene fidelidad total con los datos originales sin transformaciones.
- **processed/**: Contiene los resultados de los trabajos ETL de AWS Glue en formato Parquet:
  - `by_brand/`: Agregaciones estadísticas agrupadas por marca de vehículo, particionadas por el campo `brand`.

  - `by_year/`: Agregaciones estadísticas agrupadas por año de fabricación, particionadas por el campo `model_year`.

- **scripts/**: Almacena los scripts Python de los trabajos Glue ETL, como `cars_aggregation_by_brand.py` y `cars_aggregation_by_year.py`, accesibles por los workers de Glue durante la ejecución.
- **errors/**: Destinada a registros que Kinesis Firehose no pudo procesar correctamente, clasificados por tipo de error.
- **queries/**: Almacena los resultados de las consultas ejecutadas en Amazon Athena en formato CSV, incluyendo metadatos de ejecución.
- **logs/**: Contiene los logs de Spark generados por los trabajos Glue durante su ejecución, útiles para debugging y análisis de rendimiento del procesamiento ETL.

```
aws s3api put-object --bucket datalake-cars-$ACCOUNT_ID --key raw/
aws s3api put-object --bucket datalake-cars-$ACCOUNT_ID --key processed/
aws s3api put-object --bucket datalake-cars-$ACCOUNT_ID --key scripts/
aws s3api put-object --bucket datalake-cars-$ACCOUNT_ID --key errors/
aws s3api put-object --bucket datalake-cars-$ACCOUNT_ID --key queries/
aws s3api put-object --bucket datalake-cars-$ACCOUNT_ID --key logs/
```

Listing 2: Creación de la estructura de carpetas en S3

La estructura resultante se puede observar en la Figura 2.

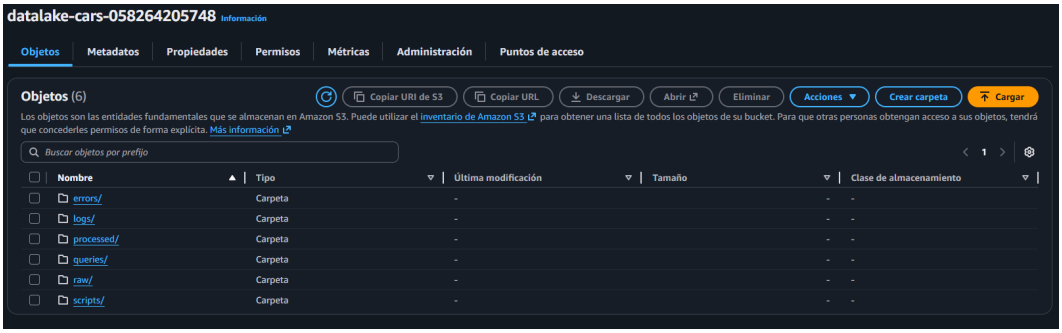

Figura 2: Estructura de carpetas del bucket S3

2.1.3 Justificación del Diseño

Esta arquitectura de Data Lake sigue el patrón *medallion* simplificado, ampliamente adoptado en arquitecturas modernas de datos:

- **Bronze (raw)**: Datos sin procesar e inmutables, manteniendo fidelidad absoluta con la fuente original. Durante la ingesta, Kinesis Firehose enriquece cada registro añadiendo

el campo `processing_date` mediante la función Lambda de transformación, permitiendo el particionamiento dinámico por fecha. Esto facilita la gestión del ciclo de vida y optimiza consultas temporales.

- **Silver (processed)**: Datos limpios, agregados y optimizados en formato Parquet columnar, reduciendo hasta un 80 % el tamaño de almacenamiento respecto a JSON y mejorando significativamente el rendimiento de consultas analíticas en Athena.

La separación de áreas funcionales (*scripts*, *errors*, *logs*, *queries*) mejora la organización, facilita el troubleshooting y permite aplicar políticas de ciclo de vida diferenciadas por tipo de contenido (por ejemplo, eliminar logs antiguos pero retener datos procesados indefinidamente).

2.2 Implementación del Productor de Datos

El productor de datos simula un sistema de gestión de inventario de vehículos usados que envía información en tiempo real a Kinesis Data Streams mediante la librería `boto3` de Python.

2.2.1 Tipo de Datos Generados

El dataset contiene aproximadamente 36.000 registros de vehículos con los siguientes atributos (Tabla 1):

Tabla 1: Estructura de datos de vehículos

Campo	**Tipo**	**Descripción**
id	string	Identificador único del vehículo
brand	string	Marca (Toyota, Honda, Ford, etc.)
model	string	Modelo del vehículo
model_year	integer	Año de fabricación
milage	integer	Kilometraje en millas
fuel_type	string	Tipo de combustible
engine	string	Especificaciones del motor
transmission	string	Tipo de transmisión
ext_col	string	Color exterior
int_col	string	Color interior
accident	string	Historial de accidentes
clean_title	string	Estado del título legal

2.2.2 Configuración del Entorno de Desarrollo

El entorno de desarrollo local se configuró utilizando `uv` como gestor de entornos virtuales Python, garantizando aislamiento de dependencias:

```
# Crear entorno virtual
uv venv

# Activar entorno (Windows)
.venv\Scripts\activate

# Instalar dependencias
uv pip install boto3 loguru
```

Listing 3: Configuración del entorno virtual con uv

Las dependencias principales incluyen:

- **boto3**: SDK oficial de AWS para Python, utilizado para interactuar con Kinesis Data Streams.
- **loguru**: Librería de logging avanzada para registro detallado del proceso de ingesta.

2.2.3 Código del Productor

El productor utiliza el método `put_record` de Kinesis para enviar registros individuales con control fino sobre el proceso:

```
import boto3
import json
import datetime
from loguru import logger

STREAM_NAME = 'cars-stream'
REGION = 'us-east-1'
DELAY_BETWEEN_RECORDS = 0.01  # 10ms entre registros

kinesis = boto3.client('kinesis', region_name=REGION)

def run_producer():
    """Envia registros de coches al stream de Kinesis"""
    data = load_data(INPUT_FILE)
    total_records = len(data)

    for i, car in enumerate(data, start=1):
        payload = {
            'id': car['id'],
            'brand': car['brand'],
            'model': car['model'],
```

```
            'model_year': car['model_year'],
            'milage': car['milage'],
            # ... resto de campos
            'timestamp_ingestion': datetime.datetime.now(
                datetime.timezone.utc
            ).isoformat()
        }

        response = kinesis.put_record(
            StreamName=STREAM_NAME,
            Data=json.dumps(payload),
            PartitionKey=car['brand']  # Distribucion por marca
        )

        time.sleep(DELAY_BETWEEN_RECORDS)

        # Log cada 100 registros
        if i % 100 == 0 or i == total_records:
            logger.info(f"Enviado: {i:,}/{total_records:,}")

# Codigo completo disponible en Anexo A
```

Listing 4: Fragmento principal del productor Kinesis (kinesis.py)

2.2.4 Estrategia de Particionamiento

La clave de partición (`PartitionKey`) utiliza el campo `brand` del vehículo, distribuyendo los registros entre los shards del stream según la marca. Esto garantiza:

- **Distribución equilibrada**: Las marcas populares se distribuyen uniformemente entre shards.
- **Orden garantizado**: Todos los registros de una misma marca se procesan secuencialmente en el mismo shard.
- **Escalabilidad**: Permite aumentar el número de shards si el volumen de una marca crece significativamente.

2.2.5 Parámetros de Configuración

Tabla 2: Configuración del productor Kinesis

Parámetro	**Valor**
Nombre del stream	cars-stream
Región AWS	us-east-1
Número de shards	1 shard (2 MB/s escritura)
Método de envío	put_record (individual)
Delay entre registros	10 ms (0.01 segundos)
Partition key	brand (marca del vehículo)
Formato de datos	JSON con timestamp UTC

El delay de 10 ms entre registros (como se observa en la Tabla 2) evita saturar el stream y permite observar el flujo de datos en tiempo real, simulando un escenario realista de ingesta continua. Durante la ejecución, se puede verificar la llegada de datos en el visor de datos del stream, como se muestra en la Figura 3.

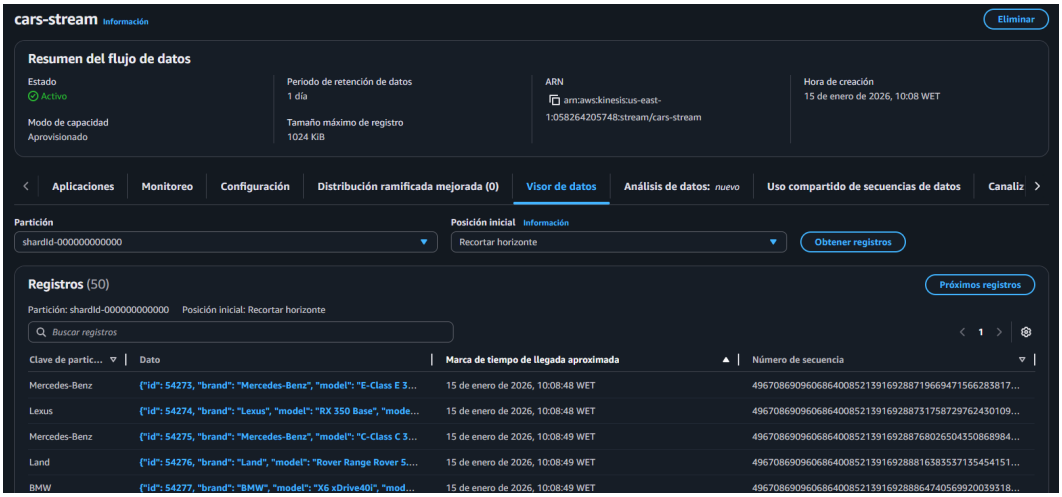

Figura 3: Visor de datos de Kinesis mostrando registros en tiempo real durante la ingesta

2.2.6 Ejecución del Productor

El productor se ejecuta desde la línea de comandos una vez configurado el entorno virtual:

```
# Activar entorno virtual
.venv\Scripts\activate

# Ejecutar productor
python kinesis.py
```

Listing 5: Ejecución del productor de datos

Durante la ejecución, `loguru` registra el progreso en tiempo real, mostrando el número de registros enviados, porcentaje de avance y shard asignado por Kinesis, como se observa en la Figura 4.

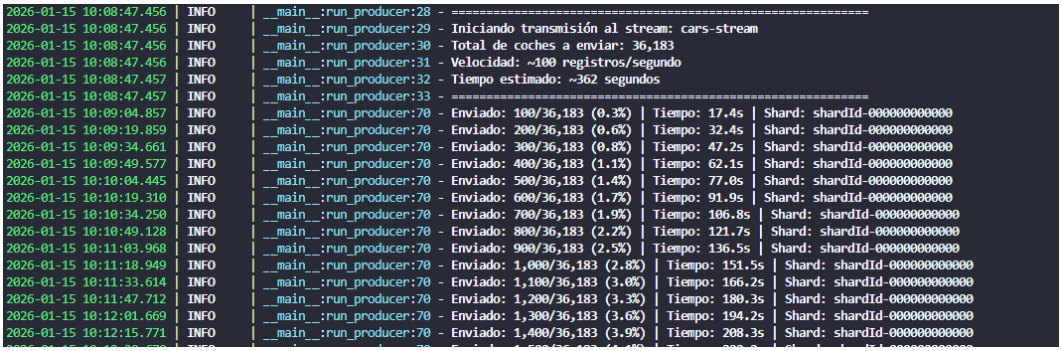

Figura 4: Logs del productor mostrando progreso de ingesta de datos

2.3 Configuración del Consumidor Kinesis Firehose

Kinesis Data Firehose actúa como consumidor del stream, aplicando transformaciones mediante AWS Lambda y entregando los datos en S3 con particionamiento dinámico por fecha de procesamiento.

2.3.1 Configuración de Kinesis Firehose

El delivery stream se configuró para consumir datos desde Kinesis Data Streams y entregarlos en el bucket S3 tras aplicar transformaciones (Tabla 3):

Tabla 3: Parámetros de configuración de Kinesis Firehose

Parámetro	**Valor**
Nombre del delivery stream	cars-delivery-stream
Tipo de fuente	Kinesis Data Streams
Stream de origen	cars-stream
Destino	Amazon S3
Bucket S3	datalake-cars-[ACCOUNT_ID]
Tamaño del buffer	64 MB
Intervalo del buffer	60 segundos
Compresión	Ninguna (datos JSON)
Particionamiento dinámico	Habilitado
Procesamiento Lambda	Habilitado
Prefijo de destino	raw/cars_data/processing_date=!.../
Prefijo de errores	errors/!firehose:error-output-type/
Rol IAM	LabRole

El buffer de Firehose acumula registros hasta alcanzar 64 MB o transcurrir 60 segundos (lo que ocurra primero) antes de escribir en S3, optimizando el número de archivos generados y reduciendo costos de operaciones PUT.

2.3.2 Función Lambda de Transformación

Se implementó una función Lambda en Python para enriquecer los datos y generar las claves de partición dinámicas necesarias para organizar los datos por fecha:

```
import json
import base64
import datetime

def lambda_handler(event, context):
    """Lambda para Kinesis Firehose: enriquecimiento y particionado"""
    output = []

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

            # Calcular fecha de procesamiento
            processing_time = datetime.datetime.now(
                datetime.timezone.utc
            )
            partition_date = processing_time.strftime('%Y-%m-%d')

            # Crear registro de salida con metadata para particionado
            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:
            # Marcar registro como fallido
            output.append({
```

```
                'recordId': record['recordId'],
                'result': 'ProcessingFailed',
                'data': record['data']
            })

    return {'records': output}

# Codigo completo disponible en Anexo B
```

Listing 6: Función Lambda de transformación para Firehose (firehose.py)

La función Lambda realiza tres operaciones principales:

- **Decodificación**: Convierte los datos de base64 a JSON legible.
- **Enriquecimiento**: Añade el campo `processing_date` calculado en formato ISO (YYYY-MM-DD).
- **Generación de metadata**: Crea las claves de partición que Firehose utilizará para organizar los archivos en S3.

2.3.3 Despliegue de la Función Lambda

La función Lambda se desplegó mediante AWS CLI, empaquetando el código Python en un archivo ZIP:

```
# Comprimir codigo
Compress-Archive -Path "firehose.py" -DestinationPath "firehose.zip"

# Crear funcion Lambda
aws lambda create-function \
  --function-name cars_firehose_processor \
  --runtime python3.9 \
  --role arn:aws:iam::ACCOUNT_ID:role/LabRole \
  --handler firehose.lambda_handler \
  --zip-file fileb://firehose.zip \
  --timeout 60 \
  --memory-size 128
```

Listing 7: Despliegue de la función Lambda

La configuración de Lambda incluye:

- **Runtime**: Python 3.9 (compatible con las librerías estándar utilizadas)
- **Timeout**: 60 segundos (suficiente para procesar lotes grandes)
- **Memoria**: 128 MB (mínimo necesario para transformaciones JSON simples)

Se ve la lambda en AWS en la Figura 5.

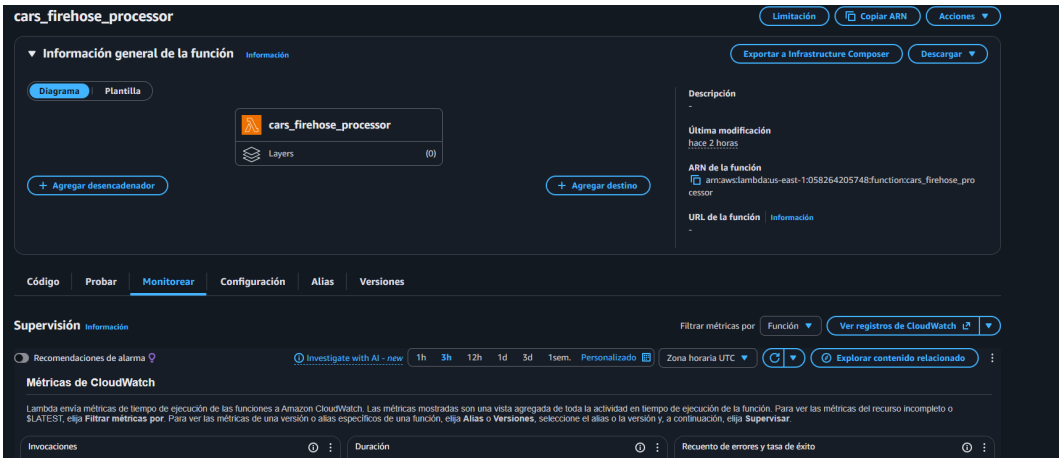

Figura 5: Función Lambda de transformación para Kinesis Firehose

2.3.4 Particionamiento Dinámico en S3

La configuración de particionamiento dinámico utiliza la clave `processing_date` generada por Lambda para organizar los archivos en S3:

- **Prefijo S3**: `raw/cars_data/processing_date=!{partitionKeyFromLambda:processing_date}/`
- **Resultado**: Los archivos se organizan en carpetas como:
  - `raw/cars_data/processing_date=2026-01-15/`
  - `raw/cars_data/processing_date=2026-01-16/`

Esta estrategia de particionamiento proporciona múltiples beneficios:

- **Consultas eficientes**: Athena puede filtrar por rango de fechas escaneando solo las particiones necesarias, reduciendo costos y tiempo de ejecución.
- **Gestión del ciclo de vida**: Facilita la aplicación de políticas S3 Lifecycle para archivar o eliminar datos antiguos automáticamente.
- **Organización lógica**: Los datos se agrupan naturalmente por día de procesamiento, simplificando auditorías y troubleshooting.

2.3.5 Configuración del Delivery Stream

El delivery stream se creó vinculando todos los componentes mediante AWS CLI:

```
aws firehose create-delivery-stream \
  --delivery-stream-name cars-delivery-stream \
  --delivery-stream-type KinesisStreamAsSource \
  --kinesis-stream-source-configuration \
    "KinesisStreamARN=arn:aws:kinesis:us-east-1:...:stream/cars-stream,\
     RoleARN=arn:aws:iam::...:role/LabRole" \
  --extended-s3-destination-configuration file://firehose_config.json
```

Listing 8: Fragmento de configuración del delivery stream

El archivo `firehose_config.json` contiene la configuración detallada del destino S3, incluyendo el procesador Lambda, los prefijos de particionamiento y la configuración del buffer, la configuración se observa en la Figura 6.

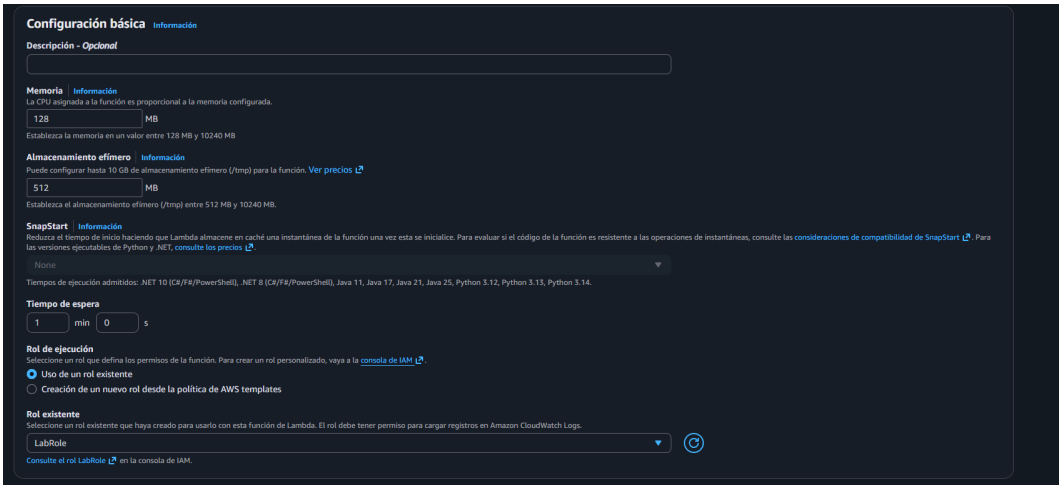

Figura 6: Configuración del delivery stream de Kinesis Firehose

2.3.6 Verificación de Datos en S3

Tras la ejecución del productor y transcurridos los 60 segundos del buffer de Firehose, los datos se almacenan en S3 organizados por fecha de procesamiento, como se observa en la Figura 7.

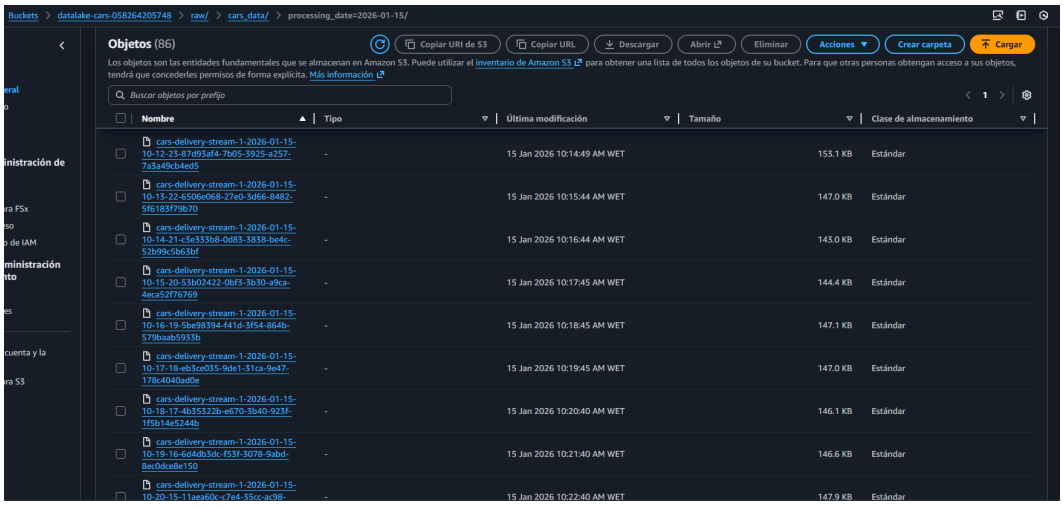

Figura 7: Datos almacenados en S3 con particionamiento dinámico por fecha

2.4 Configuración de AWS Glue

AWS Glue proporciona la capa de catalogación automática de esquemas y transformación ETL del Data Lake, permitiendo descubrir, catalogar y transformar los datos almacenados en S3.

2.4.1 Creación de la Base de Datos en Glue

Se creó una base de datos lógica en el Glue Data Catalog para organizar las tablas y metadatos del Data Lake:

```
aws glue create-database \
  --database-input '{"Name": "cars_db"}'
```

Listing 9: Creación de la base de datos en Glue

Esta base de datos actúa como un contenedor lógico para todas las tablas del proyecto, siendo accesible desde AWS Glue, Amazon Athena y otros servicios compatibles con el Glue Data Catalog. Se observa la correcta creación en la Figura 8.

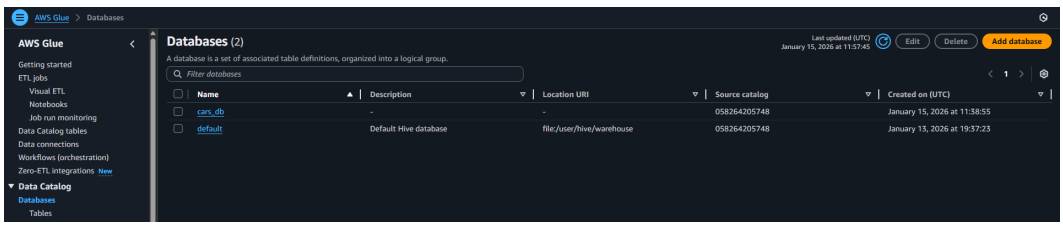

Figura 8: Base de datos cars_db creada en el Glue Data Catalog

2.4.2 Configuración del Crawler para Datos Raw

Se implementó un crawler para analizar automáticamente los datos en bruto almacenados en S3 y generar el esquema en el catálogo (Tabla 4):

Tabla 4: Configuración del crawler de datos raw

Parámetro	**Valor**
Nombre del crawler	cars_raw_crawler
Rol IAM	LabRole
Base de datos destino	cars_db
Ruta S3	s3://datalake-cars-[ACCOUNT_ID]/raw/cars_data/
Frecuencia de ejecución	Bajo demanda
Agrupación de tablas	Una tabla por ruta S3
Detectar particiones	Habilitado (automático)

El crawler se ejecuta bajo demanda tras la llegada de datos a S3, analizando la estructura de los archivos JSON y detectando automáticamente tanto el esquema como las particiones en formato Hive (`processing_date=YYYY-MM-DD`).

```
# Crear crawler
aws glue create-crawler \
  --name cars_raw_crawler \
  --role arn:aws:iam::ACCOUNT_ID:role/LabRole \
  --database-name cars_db \
  --targets '{"S3Targets": [{"Path": "s3://datalake-cars-ACCOUNT_ID/raw/
    cars_data/"}]}' \
  --configuration file://crawler_config.json

# Iniciar crawler
aws glue start-crawler --name cars_raw_crawler
```

Listing 10: Creación y ejecución del crawler raw

Durante la ejecución, el crawler escanea los archivos JSON organizados en carpetas con estructura de particiones (formato `processing_date=YYYY-MM-DD/`), infiere los tipos de datos y crea automáticamente la tabla `cars_data` en el catálogo incluyendo la definición de la columna de partición `processing_date`. El crawler registra todas las particiones encontradas sin necesidad de operaciones manuales adicionales, como se observa en la Figura 9.

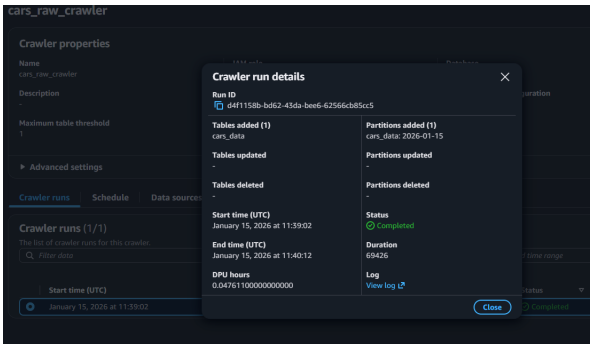

Figura 9: Ejecución del crawler mostrando estado completado

2.4.3 Verificación de la Tabla con Amazon Athena

Tras la ejecución del crawler, el script automatizado verifica que la tabla se haya creado correctamente y que sea accesible mediante Amazon Athena:

```
# Obtener nombre de la tabla creada por el crawler
$tableName = (aws glue get-tables --database-name cars_db \
  --query 'TableList[0].Name' --output text 2>$null)

# Configurar ubicacion de salida de Athena
$athenaOutputLocation = "s3://$env:BUCKET_NAME/queries/"

# Ejecutar consulta de prueba para contar registros
$countQuery = "SELECT COUNT(*) as total FROM cars_db.$tableName;"
$testQueryId = (aws athena start-query-execution \
  --query-string $countQuery \
  --result-configuration OutputLocation=$athenaOutputLocation \
  --query-execution-context Database=cars_db \
  --query QueryExecutionId --output text).Trim()

# Esperar y obtener resultado
Start-Sleep -Seconds 5
$result = aws athena get-query-results \
  --query-execution-id $testQueryId \
  --query 'ResultSet.Rows[1].Data[0].VarCharValue' --output text
```

Listing 11: Verificación automática de tabla y particiones

Esta verificación confirma que la tabla es consultable y que las particiones detectadas automáticamente por el crawler están correctamente registradas en el Data Catalog.

2.4.4 Trabajos ETL Implementados

Se desarrollaron dos trabajos ETL en PySpark para generar agregaciones analíticas sobre los datos en bruto. Los scripts se almacenaron en el bucket S3 bajo la carpeta `scripts/` y se referenciaron al crear los trabajos Glue.

Trabajo 1: Agregación por Marca Este trabajo genera estadísticas de vehículos agrupados por marca, leyendo directamente desde el Glue Data Catalog:

```
from awsglue.context import GlueContext
from pyspark.context import SparkContext
from pyspark.sql.functions import col, count, avg, min as spark_min, max
    as spark_max

args = getResolvedOptions(sys.argv, ['JOB_NAME', 'database',
                                     'table_name', 'output_path'])

glueContext = GlueContext(SparkContext())

# Leer desde Glue Catalog
dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
    database=args['database'],
    table_name=args['table_name']
)

df = dynamic_frame.toDF()

# Agregacion por marca
brand_agg = df.groupBy("brand").agg(
    count("id").alias("total_vehicles"),
    avg("milage").alias("avg_mileage"),
    spark_min("milage").alias("min_mileage"),
    spark_max("milage").alias("max_mileage")
).orderBy("brand")

# Escribir en Parquet particionado por brand
brand_agg.write.mode("overwrite") \
    .partitionBy("brand") \
    .parquet(args['output_path'])

# Codigo completo disponible en Anexo C
```

Listing 12: Fragmento del trabajo ETL por marca (cars_aggregation_by_brand.py)

Las métricas generadas incluyen:

- Total de vehículos por marca
- Kilometraje promedio, mínimo y máximo por marca

Trabajo 2: Agregación por Año de Fabricación Este trabajo genera estadísticas agrupadas por año de fabricación con métricas adicionales:

```
# Agregacion por ano de fabricacion
```

```
year_agg = df.groupBy("model_year").agg(
    count("id").alias("total_vehicles"),
    countDistinct("brand").alias("unique_brands"),
    avg("milage").alias("avg_mileage"),
    spark_min("milage").alias("min_mileage"),
    spark_max("milage").alias("max_mileage")
).orderBy(col("model_year").desc())

# Escribir en Parquet particionado por model_year
year_agg.write.mode("overwrite") \
    .partitionBy("model_year") \
    .parquet(args['output_path'])

# Codigo completo disponible en Anexo D
```

Listing 13: Fragmento del trabajo ETL por año (cars_aggregation_by_year.py)

Las métricas adicionales para esta agregación incluyen:

- Total de vehículos por año de fabricación
- Número de marcas únicas por año (`countDistinct`)
- Estadísticas de kilometraje por año

2.4.5 Configuración de los Trabajos Glue

Los trabajos ETL se configuraron mediante AWS CLI, pasando los argumentos necesarios como parámetros de línea de comandos (Tabla 5):

Tabla 5: Parámetros de configuración de los trabajos Glue ETL

Parámetro	**cars_brand_job**	**cars_year_job**
Versión de Glue	4.0	4.0
Tipo de worker	G.1X (4 vCPU, 16 GB)	G.1X (4 vCPU, 16 GB)
Número de workers	2	2
Script S3	scripts/cars_aggregation_by_brand.py	scripts/cars_aggregation_by_year.py
Argumento –database	cars_db	cars_db
Argumento –table_name	Detectado por crawler	Detectado por crawler
Argumento –output_path	s3://.../processed/by_brand/	s3://.../processed/by_year/
Logs Spark habilitados	CloudWatch + S3	CloudWatch + S3
Ruta logs Spark	s3://.../logs/	s3://.../logs/

```
# Subir script a S3
aws s3 cp "cars_aggregation_by_brand.py" \
```

```
  "s3://$env:BUCKET_NAME/scripts/" --quiet

# Crear trabajo Glue
aws glue create-job \
  --name cars_brand_job \
  --role arn:aws:iam::ACCOUNT_ID:role/LabRole \
  --command "Name=glueetl,ScriptLocation=s3://.../scripts/
   cars_aggregation_by_brand.py,PythonVersion=3" \
  --default-arguments '{"--database":"cars_db","--table_name":"cars_data
   ","--output_path":"s3://.../processed/by_brand/","--enable-continuous-
   cloudwatch-log":"true","--spark-event-logs-path":"s3://.../logs/"}' \
  --glue-version "4.0" \
  --number-of-workers 2 \
  --worker-type "G.1X"
```

Listing 14: Creación del trabajo ETL por marca

2.4.6 Ejecución Secuencial de los Trabajos ETL

El script de despliegue ejecuta los trabajos secuencialmente, esperando a que cada uno complete antes de iniciar el siguiente:

```
# Ejecutar trabajo por marca
Write-Host "Starting Brand Job..."
$BrandRunId = (aws glue start-job-run --job-name cars_brand_job \
  --query 'JobRunId' --output text).Trim()

# Esperar a que complete
do {
    Start-Sleep -Seconds 15
    $Status = (aws glue get-job-run --job-name cars_brand_job \
      --run-id $BrandRunId \
      --query 'JobRun.JobRunState' --output text).Trim()
    Write-Host "  -> $Status"
} while ($Status -in "STARTING", "RUNNING", "STOPPING")

if ($BrandStatus -eq "SUCCEEDED") {
    # Ejecutar trabajo por ano
    Write-Host "Starting Year Job..."
    $YearRunId = (aws glue start-job-run --job-name cars_year_job \
      --query 'JobRunId' --output text).Trim()
    # ... esperar completacion
}
```

Listing 15: Ejecución automatizada de los trabajos ETL

Ambos trabajos procesan los datos completos (aproximadamente 36.000 registros) y generan las tablas agregadas en formato Parquet particionado, como se observa en la Figura

10.

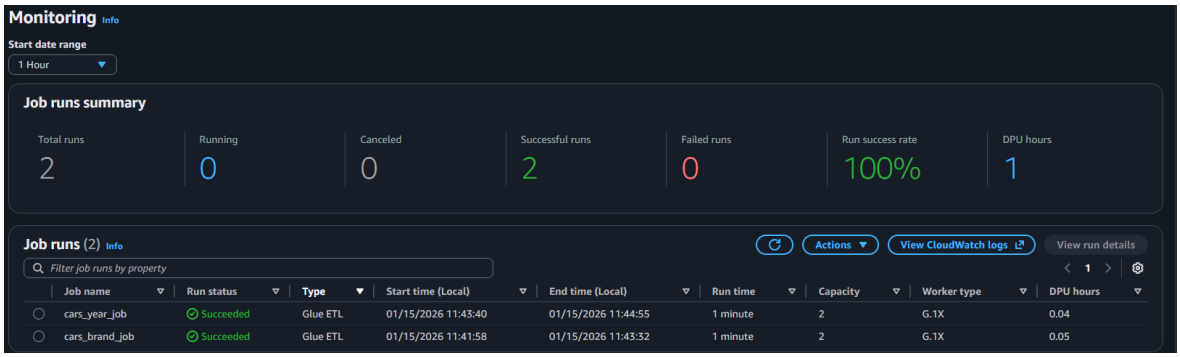

Figura 10: Historial de ejecución de los trabajos ETL mostrando estado completado

2.4.7 Crawler de Datos Procesados

Tras la ejecución exitosa de ambos trabajos ETL, el script crea automáticamente un segundo crawler para catalogar las tablas procesadas (Tabla 6):

Tabla 6: Configuración del crawler de datos procesados

Parámetro	**Valor**
Nombre del crawler	cars_processed_crawler
Rol IAM	LabRole
Base de datos destino	cars_db
Ruta S3	s3://datalake-cars-[ACCOUNT_ID]/processed/
Frecuencia de ejecución	Bajo demanda (ejecutado tras jobs ETL)

```
# Crear crawler
aws glue create-crawler --name cars_processed_crawler \
  --role $env:ROLE_ARN \
  --database-name cars_db \
  --targets '{"S3Targets": [{"Path": "s3://$env:BUCKET_NAME/processed
    /"}]}'

# Iniciar crawler
aws glue start-crawler --name cars_processed_crawler

# Esperar a que complete
do {
    Start-Sleep -Seconds 15
    $CrawlerState = (aws glue get-crawler \
      --name cars_processed_crawler \
      --query 'Crawler.State' --output text 2>$null)
    Write-Host "  Crawler: $CrawlerState"
} while ($CrawlerState -and $CrawlerState -ne "READY")
```

Listing 16: Creación y ejecución del crawler de datos procesados

Este crawler detecta automáticamente las dos tablas agregadas generadas por los trabajos ETL:

- `by_brand`: Agregaciones por marca (particionadas por `brand`)
- `by_year`: Agregaciones por año (particionadas por `model_year`)

En la Figura 11 se observan todas las tablas creadas.

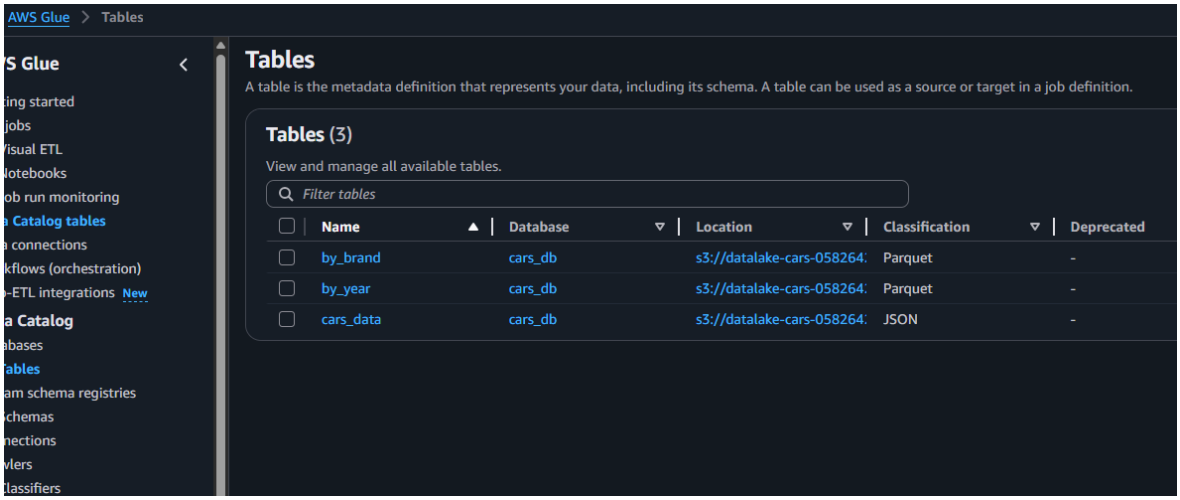

Figura 11: Tablas catalogadas en Glue Data Catalog (raw + procesadas)

Al finalizar, el script muestra un resumen completo del pipeline, confirmando la creación de las 3 tablas (1 raw + 2 procesadas) en el Glue Data Catalog. Gracias a la catalogación realizada por los crawlers, estas tablas son inmediatamente consultables mediante Amazon Athena, permitiendo ejecutar queries SQL estándar sobre los datos almacenados en S3 sin necesidad de cargarlos en una base de datos tradicional. La disponibilidad de los datos en S3 organizados en sus respectivas carpetas con particionamiento optimiza el rendimiento y reduce los costes de las consultas al permitir filtrado por particiones.

2.5 Diagrama del Flujo de Datos

El flujo completo de datos a través del Data Lake sigue una arquitectura de streaming con procesamiento ETL, desde la ingesta hasta las consultas analíticas.

2.5.1 Arquitectura del Pipeline

El diagrama de la 12 ilustra el flujo end-to-end de datos a través de todos los servicios AWS implementados:

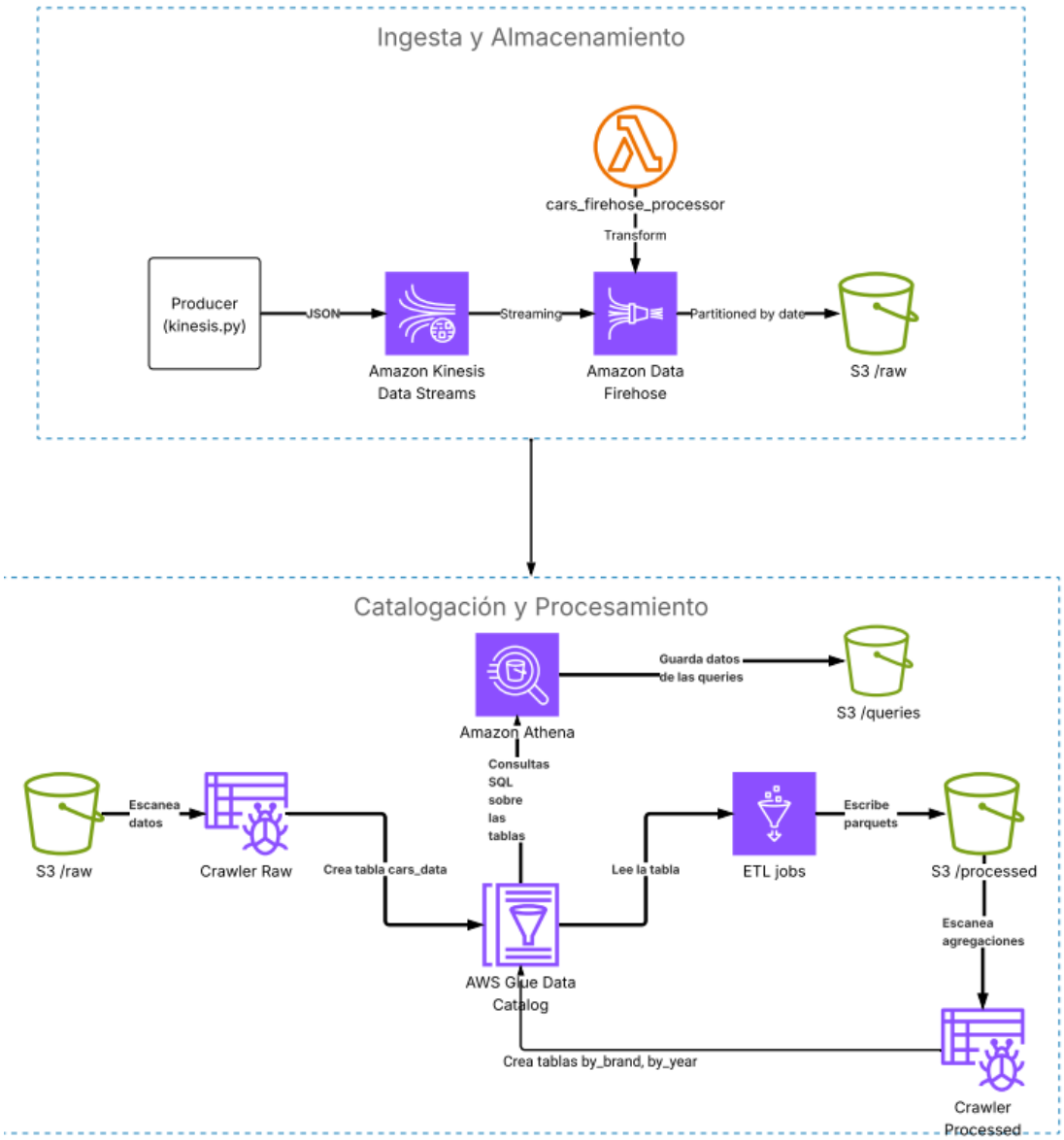

Figura 12: Diagrama del flujo de datos completo del Data Lake

2.5.2 Etapas del Flujo

El pipeline se compone de las siguientes etapas:

1. **Ingesta**: El productor Python (`kinesis.py`) envía registros JSON individuales a Kinesis Data Streams con particionamiento por marca.

2. **Transformación en tránsito**: Kinesis Firehose consume el stream y procesa cada registro mediante la función Lambda, añadiendo el campo `processing_date`.

3. **Almacenamiento raw**: Los datos transformados se escriben en S3 bajo `raw/cars_data/` con particionamiento dinámico por fecha.

4. **Catalogación**: El crawler de Glue escanea los datos raw, detecta el esquema y genera la tabla `cars_data` en el Data Catalog.

5. **Procesamiento ETL**: Los trabajos Glue leen desde el catálogo, ejecutan agregaciones PySpark y escriben resultados en formato Parquet bajo `processed/`.

6. **Catalogación procesada**: Un segundo crawler detecta las tablas agregadas (`by_brand`, `by_year`) y las registra en el catálogo.

7. **Consultas**: Amazon Athena permite ejecutar consultas SQL sobre cualquier tabla catalogada, tanto raw como procesadas.

2.6 Presupuesto y estimación de costes

A continuación se presenta una estimación de costes para un escenario de producción realista, asumiendo una ingesta continua de datos (24 horas al día, 30 días al mes) con un volumen moderado de aproximadamente 20 registros por minuto. 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 **AWS Pricing Calculator** como referencia principal, con la configuración detallada en la Tabla 7.

Tabla 7: Estimación de costes mensuales y anuales

Servicio	**Mes**	**Año**
Amazon Kinesis Data Streams	$10.97	$131.64
Amazon Data Firehose	$0.08	$0.96
AWS Lambda	$0.31	$3.72
Amazon S3	$0.24	$2.88
AWS Glue	$8.95	$107.40
Amazon Athena	$0.44	$5.28
SUBTOTAL	**$20.99**	**$251.88**
Contingencia (20 %)	**$4.20**	**$50.38**
TOTAL	**$25.19**	**$302.26**

2.6.1 Detalle de los cálculos

Amazon Kinesis Data Streams ($10.97/mes) 1 shard × 730 h × $0.015/h = $10.95. Unidades PUT: $\lceil 1\text{ KB} / 25\text{ KB} \rceil$ = 1 unidad × 867,240 registros × $0.000000014 = $0.01. Total: $10.97/mes.

Amazon Data Firehose ($0.08/mes) Ingesta de 4.18 GB/mes: 4.18 GB × $0.029/GB = $0.12. Particionamiento dinámico: 0.84 GB × $0.02/GB = $0.02. Objetos S3: $0.00007. Total efectivo: $0.08/mes.

AWS Lambda ($0.31/mes) 43,200 invocaciones/mes con 128 MB de memoria y 200 ms de duración. Cómputo: 1,080 GB-s × $0.0000166667 = $0.02. Solicitudes: 43,200 × $0.0000002 = $0.01. Total: $0.31/mes.

Amazon S3 ($0.24/mes) Almacenamiento: 1 GB × $0.023 = $0.023. Solicitudes PUT: 43,200 × $0.000005 = $0.216. Solicitudes GET: 10,000 × $0.0000004 = $0.004. Total: $0.24/mes.

AWS Glue ($8.95/mes) ETL Jobs: 2 DPU × 10 h/mes × $0.44/DPU-hora = $8.80. Crawlers: 2 × 0.17 h × $0.44 = $0.15. Total: $8.95/mes.

Amazon Athena ($0.44/mes) 900 consultas/mes × 0.00009765625 TB × $5/TB = $0.44/mes.

2.6.2 Distribución de costes

La distribución porcentual del presupuesto mensual (sin contingencia) es:

- **Amazon Kinesis Data Streams:** 52.3 % ($10.97) - Mayor componente debido al coste fijo del shard.
- **AWS Glue:** 42.6 % ($8.95) - ETL jobs diarios con procesamiento Spark.
- **Amazon Athena:** 2.1 % ($0.44) - Consultas SQL sobre datos catalogados.
- **AWS Lambda:** 1.5 % ($0.31) - Transformaciones en streaming.
- **Amazon S3:** 1.1 % ($0.24) - Almacenamiento y operaciones.
- **Amazon Data Firehose:** 0.4 % ($0.08) - Ingesta y particionamiento.

2.6.3 Colchón de contingencia

Se incluye un margen del 20 % ($4.20/mes) para absorber imprevistos como picos de tráfico, ejecuciones adicionales de ETL, consultas exploratorias, o costes de transferencia de datos entre servicios.

2.6.4 Optimizaciones recomendadas

Kinesis y Glue representan el 94.9 % del presupuesto total. Se recomienda:

- **Kinesis Data Streams:** Considerar modo On-Demand si el tráfico es variable, evitando el coste fijo del shard.
- **Glue ETL Jobs:** Reducir frecuencia (semanal en lugar de diaria) o usar Flex Execution (reducción del 34 %).
- **Glue Crawlers:** Ejecutar bajo demanda o semanalmente.
- **S3:** Implementar políticas de ciclo de vida para mover datos antiguos a S3 Glacier tras 90 días.

Si se aplica la **Capa Gratuita de AWS**, el coste mensual se reduciría aproximadamente a $20.96 (sin contingencia), ya que Lambda incluye 1M de invocaciones gratuitas y S3 ofrece 5GB de almacenamiento sin coste.

2.7 Conclusiones

La realización de esta práctica ha permitido adquirir experiencia práctica en el diseño e implementación de una arquitectura completa de ingesta, procesamiento y análisis de datos en la nube utilizando servicios gestionados de AWS.

2.7.1 Conocimientos adquiridos

A lo largo del desarrollo de la práctica se han consolidado los siguientes conocimientos:

- **Ingesta de datos en tiempo real:** Se ha implementado un flujo completo de streaming utilizando Amazon Kinesis Data Streams como punto de entrada y Kinesis Data Firehose como consumidor. Esto ha permitido comprender cómo gestionar flujos de datos continuos y aplicar transformaciones en tránsito mediante funciones Lambda.
- **Transformación y enriquecimiento de datos:** La función Lambda integrada con Firehose ha demostrado ser una solución eficiente para añadir metadatos (como fechas de ingesta) sin necesidad de modificar el productor original. Además, se ha aprendido a configurar el particionado dinámico en S3 para optimizar consultas posteriores.
- **Procesamiento ETL con AWS Glue:** Se ha trabajado con Crawlers para catalogación automática de esquemas y con Jobs ETL escritos en PySpark para realizar agregaciones y transformaciones complejas. La experiencia con el formato Parquet ha evidenciado sus ventajas en términos de compresión y rendimiento de consultas.

- **Consultas SQL sobre datos en S3:** Amazon Athena ha permitido ejecutar consultas SQL estándar directamente sobre los datos almacenados en S3, sin necesidad de cargarlos en una base de datos tradicional. Esto demuestra la flexibilidad de las arquitecturas de *data lake*.
- **Gestión de costes en la nube:** El análisis de presupuesto ha puesto de manifiesto la importancia de optimizar los recursos para controlar gastos, especialmente en servicios como AWS Glue que pueden representar la mayor parte del coste operativo.

En conclusión, esta práctica ha proporcionado una visión integral de cómo construir pipelines de datos escalables y resilientes en AWS, desde la ingesta hasta el análisis, aplicando buenas prácticas de arquitectura cloud y gestión de costes.

2.8 Referencias y bibliografía

2.8.1 Figuras

2.8.2 Tablas

2.8.3 Documentación técnica

Referencias

[1] Amazon Web Services. (2025). *Amazon Kinesis Data Streams Developer Guide*.

https://docs.aws.amazon.com/kinesis/latest/dev/

[2] Amazon Web Services. (2025). *AWS Glue Developer Guide*.

https://docs.aws.amazon.com/glue/latest/dg/

[3] Amazon Web Services. (2025). *Amazon S3 User Guide*.

https://docs.aws.amazon.com/s3/

[4] Amazon Web Services. (2026). *AWS Pricing Calculator*.

https://calculator.aws/

2.8.4 Referencias del presupuesto

El presupuesto completo y detallado, incluyendo todos los parámetros de configuración y cálculos intermedios, puede consultarse en la AWS Pricing Calculator mediante el siguiente enlace:

https://calculator.aws/#/estimate?id=255654d3929bc3177f8e2f9c54d667de25220641

Esta estimación fue generada el 15 de enero de 2026 y refleja los precios vigentes de AWS para la región US East (Norte de Virginia).

2.9 Repositorio del código fuente

El código fuente completo de este proyecto, incluyendo todos los scripts, funciones y archivos de configuración, está disponible en el siguiente repositorio de GitHub:

https://github.com/maikos08/CN-P2

3 Anexos

3.1 Anexo A: Código fuente completo

3.1.1 A.1. Script productor de datos (kinesis.py)

Este script se encarga de leer los datos del archivo JSON y enviarlos al stream de Kinesis de forma continua.

```
import boto3
import json
import time
from loguru import logger
import datetime

# CONFIGURACI N
STREAM_NAME = 'cars-stream'
REGION = 'us-east-1'
INPUT_FILE = 'cars_data.json'

# Configuraci n del delay entre registros
DELAY_BETWEEN_RECORDS = 0.01

kinesis = boto3.client('kinesis', region_name=REGION)

def load_data(file_path):
    """Carga el archivo JSON con los datos de coches"""
    with open(file_path, 'r', encoding='utf-8') as f:
        return json.load(f)

def run_producer():
    """Env a los registros de coches al stream de Kinesis """
    data = load_data(INPUT_FILE)
    total_records = len(data)
    records_sent = 0

    logger.info(f"{'='*60}")
    logger.info(f"Iniciando transmisi n al stream: {STREAM_NAME}")
    logger.info(f"Total de coches a enviar: {total_records:,}")
    logger.info(f"Velocidad: ~{1/DELAY_BETWEEN_RECORDS:.0f} registros/
    segundo")
    logger.info(f"Tiempo estimado: ~{total_records *
    DELAY_BETWEEN_RECORDS:.0f} segundos")
    logger.info(f"{'='*60}")

    start_time = time.time()

    # Enviar registro por registro al stream
```

```
    for i, car in enumerate(data, start=1):

        payload = {
            'id': car['id'],
            'brand': car['brand'],
            'model': car['model'],
            'model_year': car['model_year'],
            'milage': car['milage'],
            'fuel_type': car['fuel_type'],
            'engine': car['engine'],
            'transmission': car['transmission'],
            'ext_col': car['ext_col'],
            'int_col': car['int_col'],
            'accident': car['accident'],
            'clean_title': car['clean_title'],
            'timestamp_ingestion': datetime.datetime.now(datetime.
    timezone.utc).isoformat()
        }

        try:
            # Enviar a Kinesis
            response = kinesis.put_record(
                StreamName=STREAM_NAME,
                Data=json.dumps(payload),
                PartitionKey=car['brand']  # Usa marca como partition key
            )

            records_sent += 1

            # Log cada 100 registros o al final
            if i % 100 == 0 or i == total_records:
                progress_pct = (i / total_records) * 100
                elapsed = time.time() - start_time
                logger.info(
                    f"Enviado: {i:,}/{total_records:,} ({progress_pct:.1f
    }%) | "
                    f"Tiempo: {elapsed:.1f}s | "
                    f"Shard: {response['ShardId']}"
                )

            # Peque a pausa para simular streaming
            time.sleep(DELAY_BETWEEN_RECORDS)

        except Exception as e:
            logger.error(f"Error enviando registro {i}: {str(e)}")
            continue
```

```
    elapsed_total = time.time() - start_time

    logger.info(f"{'='*60}")
    logger.info(f"    Transmisi n completada")
    logger.info(f"    Total registros enviados: {records_sent:,}/{
    total_records:,}")
    logger.info(f"    Tiempo total: {elapsed_total:.1f}s ({elapsed_total
    /60:.1f} minutos)")
    logger.info(f"    Velocidad real: {records_sent/elapsed_total:.1f}
    registros/segundo")
    logger.info(f"{'='*60}")

if __name__ == '__main__':
    try:
        run_producer()
    except FileNotFoundError:
        logger.error(f"No se encuentra el archivo {INPUT_FILE}")
    except Exception as e:
        logger.error(f"Error inesperado: {str(e)}")
        import traceback
        traceback.print_exc()
```

Listing 17: Productor de datos - kinesis.py

3.1.2 A.2. Función Lambda de transformación (firehose.py)

Función Lambda que añade metadatos de fecha para el particionado dinámico en S3.

```
import json
import base64
import datetime

def lambda_handler(event, context):
    """
    Lambda para Kinesis Firehose
    """
    output = []

    for record in event['records']:
        try:
            # Decodificar el payload del registro
            payload = base64.b64decode(record['data']).decode('utf-8')
            data_json = json.loads(payload)

            # Calcular fecha de procesamiento para particionado
            processing_time = datetime.datetime.now(datetime.timezone.utc
    )
```

```
            # Crear clave de partici n en formato YYYY-MM-DD
            partition_date = processing_time.strftime('%Y-%m-%d')

            # Crear registro de salida con metadata para particionado
    din mico
            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:
            # Si hay error, marcar el registro como fallido
            print(f"Error procesando registro: {str(e)}")
            output.append({
                'recordId': record['recordId'],
                'result': 'ProcessingFailed',
                'data': record['data']
            })

    return {'records': output}
```

Listing 18: Lambda para Firehose - firehose.py

3.1.3 A.3. Jobs ETL de AWS Glue

A.3.1. Agregación por marca

Job de Glue que agrupa los datos por marca de vehículo y calcula estadísticas.

```
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, count, avg, min as spark_min, max
    as spark_max
```

```
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)


# Leer desde Glue Catalog
print(f"Reading from {args['database']}.{args['table_name']}...")
dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
    database=args['database'],
    table_name=args['table_name']
)


# Convertir a DataFrame
df = dynamic_frame.toDF()
print(f"Records read: {df.count()}")


# Agregaci n por marca
print("Aggregating by brand...")
brand_agg = df.groupBy("brand").agg(
    count("id").alias("total_vehicles"),
    avg("milage").alias("avg_mileage"),
    spark_min("milage").alias("min_mileage"),
    spark_max("milage").alias("max_mileage")
).orderBy("brand")


print(f"Aggregated records: {brand_agg.count()}")
brand_agg.show(10)


# Escribir resultado en Parquet PARTICIONADO por brand
print(f"Writing to {args['output_path']} partitioned by brand...")
brand_agg.write.mode("overwrite").partitionBy("brand").parquet(args['
    output_path'])


print("Job completed successfully.")
job.commit()
```

Listing 19: ETL Job - cars_aggregation_by_brand.py

A.3.2. Agregación por año

Job de Glue que agrupa los datos por año de fabricación.

```
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, count, avg, min as spark_min, max
    as spark_max, countDistinct


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)


# Leer desde Glue Catalog
print(f"Reading from {args['database']}.{args['table_name']}...")
dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
    database=args['database'],
    table_name=args['table_name']
)


# Convertir a DataFrame
df = dynamic_frame.toDF()
print(f"Records read: {df.count()}")


# Agregaci n por a o
print("Aggregating by year...")
year_agg = df.groupBy("model_year").agg(
    count("id").alias("total_vehicles"),
    countDistinct("brand").alias("unique_brands"),
    avg("milage").alias("avg_mileage"),
    spark_min("milage").alias("min_mileage"),
```

```
    spark_max("milage").alias("max_mileage")
).orderBy(col("model_year").desc())


print(f"Aggregated records: {year_agg.count()}")
year_agg.show(15)


# Escribir resultado en Parquet PARTICIONADO por model_year
print(f"Writing to {args['output_path']} partitioned by model_year...")
year_agg.write.mode("overwrite").partitionBy("model_year").parquet(args['
    output_path'])


print("Job completed successfully.")
job.commit()
```

Listing 20: ETL Job - cars_aggregation_by_year.py

3.1.4 A.4. Script de configuración automática (setup.ps1)

Script de PowerShell que automatiza la creación de todos los recursos AWS necesarios.

```
# Script de despliegue completo - Cars Data Lake
# Compatible con PowerShell 5.1+ y AWS CLI 2.2.0+


Write-Host "========================================" -ForegroundColor
    Cyan
Write-Host "  DATA LAKE COCHES - SETUP COMPLETO" -ForegroundColor Cyan
Write-Host "========================================" -ForegroundColor
    Cyan


# Configuracion
$env:AWS_REGION = "us-east-1"
$env:ACCOUNT_ID = (aws sts get-caller-identity --query Account --output
    text).Trim()
$env:BUCKET_NAME = "datalake-cars-$($env:ACCOUNT_ID)"
$env:ROLE_ARN = (aws iam get-role --role-name LabRole --query 'Role.Arn'
    --output text).Trim()


Write-Host "Bucket: $env:BUCKET_NAME" -ForegroundColor Cyan
Write-Host "Role: $env:ROLE_ARN" -ForegroundColor Cyan
```

```
# Helper functions
function New-TempJson {
    param($Content)
    $Path = [System.IO.Path]::GetTempFileName()
    $Content | ConvertTo-Json -Depth 10 -Compress | Out-File -FilePath
   $Path -Encoding ASCII
    return $Path
}


function Wait-ForJob {
    param($JobName, $RunId)
    Write-Host "Waiting for job $JobName..." -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 "  -> $Status" -ForegroundColor Gray
    } while ($Status -in "STARTING", "RUNNING", "STOPPING")
    return $Status
}


# ============================================
# 1. S3 BUCKET
# ============================================
Write-Host "`n[1/8] Setting up S3..." -ForegroundColor Yellow
aws s3 mb "s3://$env:BUCKET_NAME" 2>$null
aws s3api put-object --bucket $env:BUCKET_NAME --key raw/ 2>$null | Out-
   Null
aws s3api put-object --bucket $env:BUCKET_NAME --key processed/ 2>$null |
    Out-Null
aws s3api put-object --bucket $env:BUCKET_NAME --key scripts/ 2>$null |
   Out-Null
aws s3api put-object --bucket $env:BUCKET_NAME --key errors/ 2>$null |
   Out-Null
aws s3api put-object --bucket $env:BUCKET_NAME --key queries/ 2>$null |
   Out-Null
aws s3api put-object --bucket $env:BUCKET_NAME --key logs/ 2>$null | Out-
   Null
Write-Host "OK S3 configurado" -ForegroundColor Green


# ============================================
# 2. KINESIS STREAM
# ============================================
Write-Host "`n[2/8] Setting up Kinesis..." -ForegroundColor Yellow
```

```
aws kinesis create-stream --stream-name cars-stream --shard-count 1 2>
    $null


$streamActive = $false
$attempts = 0
while (-not $streamActive -and $attempts -lt 20) {
    Start-Sleep -Seconds 5
    $attempts++
    $streamStatus = aws kinesis describe-stream --stream-name cars-stream
    --query 'StreamDescription.StreamStatus' --output text 2>$null
    if ($streamStatus -eq "ACTIVE") { $streamActive = $true }
}
Write-Host "OK Kinesis Stream activo" -ForegroundColor Green


# ============================================
# 3. LAMBDA + FIREHOSE
# ============================================
Write-Host "`n[3/8] Setting up Firehose..." -ForegroundColor Yellow


$LAMBDA_NAME = "cars_firehose_processor"
$ZIP_PATH = "firehose.zip"


if (Test-Path $ZIP_PATH) { Remove-Item $ZIP_PATH }
Compress-Archive -Path "firehose.py" -DestinationPath $ZIP_PATH -Force


aws lambda delete-function --function-name $LAMBDA_NAME 2>&1 | Out-Null
aws lambda create-function `
    --function-name $LAMBDA_NAME `
    --runtime python3.9 `
    --role $env:ROLE_ARN `
    --handler firehose.lambda_handler `
    --zip-file "fileb://$ZIP_PATH" `
    --timeout 60 `
    --memory-size 128 | Out-Null


$LAMBDA_ARN = (aws lambda get-function --function-name $LAMBDA_NAME --
    query 'Configuration.FunctionArn' --output text).Trim()
Start-Sleep -Seconds 5


$DELIVERY_STREAM_NAME = "cars-delivery-stream"
```

```
aws firehose delete-delivery-stream --delivery-stream-name
    $DELIVERY_STREAM_NAME 2>&1 | Out-Null
Start-Sleep -Seconds 5


$FIREHOSE_CONFIG_JSON = @"
{
    "BucketARN": "arn:aws:s3:::$($env:BUCKET_NAME)",
    "RoleARN": "$env:ROLE_ARN",
    "Prefix": "raw/cars_data/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 = "firehose_config.json"
$FIREHOSE_CONFIG_JSON | Out-File -FilePath $CONFIG_FILE -Encoding ASCII


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

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


Remove-Item $CONFIG_FILE -ErrorAction SilentlyContinue
Write-Host "OK Firehose configurado" -ForegroundColor Green


# ============================================
# 4. PRODUCTOR
# ============================================
Write-Host "`n[4/8] Generating Data & Syncing..." -ForegroundColor Yellow
python kinesis.py


Write-Host "Esperando 180 segundos a que Firehose escriba en S3..." -
    ForegroundColor Magenta


$totalSeconds = 180
$interval = 10
$elapsed = 0


while ($elapsed -lt $totalSeconds) {
    Start-Sleep -Seconds $interval
    $elapsed += $interval
    $remaining = $totalSeconds - $elapsed
    Write-Host "  Tiempo transcurrido: $elapsed s (faltan $remaining s)"
    -ForegroundColor Gray
}


Write-Host "OK Espera completada" -ForegroundColor Green


# ============================================
# 5. GLUE DATABASE
# ============================================
Write-Host "`n[5/8] Setting up Glue..." -ForegroundColor Yellow


$DbInput = @{ Name = "cars_db" }
$DbFile = New-TempJson -Content $DbInput
aws glue create-database --database-input "file://$DbFile" 2>&1 | Out-
    Null
Remove-Item $DbFile
Write-Host "OK Database creada" -ForegroundColor Green
```

```
# ============================================
# 6. CRAWLER RAW
# ============================================
Write-Host "`n[6/8] Creating RAW Crawler..." -ForegroundColor Yellow

aws glue delete-crawler --name cars_raw_crawler 2>&1 | Out-Null
Start-Sleep -Seconds 2

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

$crawlerConfig = @"
{
    "Version": 1.0,
    "CrawlerOutput": {
        "Tables": {
            "TableThreshold": 1
        }
    }
}
"@

[System.IO.File]::WriteAllText("$PWD\crawler_config.json", $crawlerConfig
    , [System.Text.UTF8Encoding]::new($false))

aws glue create-crawler `
    --name cars_raw_crawler `
    --role $env:ROLE_ARN `
    --database-name cars_db `
    --targets "file://$CrawlerFile" `
    --configuration file://crawler_config.json

Remove-Item $CrawlerFile
Remove-Item crawler_config.json

Write-Host "Starting RAW Crawler..." -ForegroundColor Gray
aws glue start-crawler --name cars_raw_crawler

Write-Host "Waiting for crawler to finish..." -ForegroundColor Magenta
do {
    Start-Sleep -Seconds 15
```

```
    $CrawlerState = (aws glue get-crawler --name cars_raw_crawler --query
    'Crawler.State' --output text 2>$null)
    if ($CrawlerState) {
        Write-Host "  Crawler Status: $CrawlerState" -ForegroundColor
    Gray
    }
} while ($CrawlerState -and $CrawlerState -ne "READY")

$tableName = (aws glue get-tables --database-name cars_db --query '
    TableList[0].Name' --output text 2>$null)
if ($tableName -and $tableName -ne "None") {
    Write-Host "OK Table created: $tableName" -ForegroundColor Green

    Write-Host "  Testing with Athena..." -ForegroundColor Yellow

    $athenaOutputLocation = "s3://$env:BUCKET_NAME/queries/"
    $countQuery = "SELECT COUNT(*) as total FROM cars_db.$tableName;"
    $testQueryId = (aws athena start-query-execution --query-string
    $countQuery --result-configuration OutputLocation=
    $athenaOutputLocation --query-execution-context Database=cars_db --
    query QueryExecutionId --output text).Trim()

    Start-Sleep -Seconds 5

    $result = aws athena get-query-results --query-execution-id
    $testQueryId --query 'ResultSet.Rows[1].Data[0].VarCharValue' --output
    text 2>$null

    if ($result) {
        Write-Host "  OK Table readable: $result records" -
    ForegroundColor Green
    } else {
        Write-Host "  WARNING Cannot read table or no data available" -
    ForegroundColor Yellow
    }

} else {
    Write-Host "ERROR: Table not created" -ForegroundColor Red
    exit 1
}


# ============================================
# 7. ETL JOBS
# ============================================
Write-Host "`n[7/8] Setting up ETL Jobs..." -ForegroundColor Yellow
```

```
aws s3 cp "cars_aggregation_by_brand.py" "s3://$env:BUCKET_NAME/scripts/"
    --quiet
aws s3 cp "cars_aggregation_by_year.py" "s3://$env:BUCKET_NAME/scripts/"
   --quiet


aws glue delete-job --job-name cars_brand_job 2>&1 | Out-Null
$BrandCommand = @{
    Name = "glueetl"
    ScriptLocation = "s3://$env:BUCKET_NAME/scripts/
   cars_aggregation_by_brand.py"
    PythonVersion = "3"
}
$BrandArgs = @{
    "--database" = "cars_db"
    "--table_name" = $tableName
    "--output_path" = "s3://$env:BUCKET_NAME/processed/by_brand/"
    "--job-language" = "python"
    "--enable-continuous-cloudwatch-log" = "true"
    "--spark-event-logs-path" = "s3://$env:BUCKET_NAME/logs/"
}


$BrandCmdFile = New-TempJson -Content $BrandCommand
$BrandArgsFile = New-TempJson -Content $BrandArgs


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


Remove-Item $BrandCmdFile, $BrandArgsFile


aws glue delete-job --job-name cars_year_job 2>&1 | Out-Null
$YearCommand = @{
    Name = "glueetl"
    ScriptLocation = "s3://$env:BUCKET_NAME/scripts/
   cars_aggregation_by_year.py"
    PythonVersion = "3"
```

```
}
$YearArgs = @{
    "--database" = "cars_db"
    "--table_name" = $tableName
    "--output_path" = "s3://$env:BUCKET_NAME/processed/by_year/"
    "--job-language" = "python"
    "--enable-continuous-cloudwatch-log" = "true"
    "--spark-event-logs-path" = "s3://$env:BUCKET_NAME/logs/"
}


$YearCmdFile = New-TempJson -Content $YearCommand
$YearArgsFile = New-TempJson -Content $YearArgs


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


Remove-Item $YearCmdFile, $YearArgsFile


Write-Host "OK Jobs created" -ForegroundColor Green


# ============================================
# 8. EJECUTAR PIPELINE
# ============================================
Write-Host "`n[8/8] Running Pipeline..." -ForegroundColor Yellow


Write-Host "Starting Brand Job..." -ForegroundColor Yellow
$BrandRunId = (aws glue start-job-run --job-name cars_brand_job --query '
    JobRunId' --output text).Trim()
$BrandStatus = Wait-ForJob -JobName "cars_brand_job" -RunId $BrandRunId


if ($BrandStatus -eq "SUCCEEDED") {
    Write-Host "OK Brand Job SUCCEEDED" -ForegroundColor Green
```

```
Write-Host "`nStarting Year Job..." -ForegroundColor Yellow
$YearRunId = (aws glue start-job-run --job-name cars_year_job --query
'JobRunId' --output text).Trim()
$YearStatus = Wait-ForJob -JobName "cars_year_job" -RunId $YearRunId


if ($YearStatus -eq "SUCCEEDED") {
    Write-Host "OK Year Job SUCCEEDED" -ForegroundColor Green


    Write-Host "`nCreating processed crawler..." -ForegroundColor
Cyan


    aws glue delete-crawler --name cars_processed_crawler 2>$null
    Start-Sleep -Seconds 2


    $ProcessedTargets = @{ S3Targets = @( @{ Path = "s3://$env:
BUCKET_NAME/processed/" } ) }
    $ProcessedFile = New-TempJson -Content $ProcessedTargets
    aws glue create-crawler --name cars_processed_crawler --role $env
:ROLE_ARN --database-name cars_db --targets "file://$ProcessedFile"
    Remove-Item $ProcessedFile


    Write-Host "Starting processed crawler..." -ForegroundColor
Yellow
    aws glue start-crawler --name cars_processed_crawler


    Write-Host "Waiting for crawler..." -ForegroundColor Magenta
    do {
        Start-Sleep -Seconds 15
        $CrawlerState = (aws glue get-crawler --name
cars_processed_crawler --query 'Crawler.State' --output text 2>$null)
        if ($CrawlerState) {
            Write-Host "  Crawler: $CrawlerState" -ForegroundColor
Gray
        }
    } while ($CrawlerState -and $CrawlerState -ne "READY")


    $tables = aws glue get-tables --database-name cars_db --query '
TableList[].Name' --output json | ConvertFrom-Json
```

```
        Write-Host "`n====================================" -
    ForegroundColor Green
        Write-Host "PIPELINE COMPLETADO EXITOSAMENTE" -ForegroundColor
    Green
        Write-Host "====================================" -
    ForegroundColor Green


        Write-Host "`nTablas creadas:" -ForegroundColor Cyan
        $tables | ForEach-Object { Write-Host "  * $_" -ForegroundColor
    Green }


        Write-Host "`nEstructura en S3:" -ForegroundColor Cyan
        Write-Host "  raw/cars_data/:" -ForegroundColor Yellow
        aws s3 ls s3://$env:BUCKET_NAME/raw/cars_data/ | Select-Object -
    First 3


        Write-Host "`n  processed/by_brand/:" -ForegroundColor Yellow
        aws s3 ls s3://$env:BUCKET_NAME/processed/by_brand/ | Select-
    Object -First 5


        Write-Host "`n  processed/by_year/:" -ForegroundColor Yellow
        aws s3 ls s3://$env:BUCKET_NAME/processed/by_year/ | Select-
    Object -First 5

    } else {
        Write-Host "ERROR Year Job FAILED" -ForegroundColor Red
    }
} else {
    Write-Host "ERROR Brand Job FAILED" -ForegroundColor Red
}


Remove-Item "firehose.zip" -ErrorAction SilentlyContinue
Write-Host "`nSetup completado" -ForegroundColor Green
```

Listing 21: Script de setup - setup.ps1

3.1.5 A.5. Script de limpieza de AWS

Script en powershell que elimina todos los recursos creados con el script de setup.ps1, intenta eliminar mas cosas que se pusieron durante el desarrollo y pruebas en la práctica, pero que no se usan en la entrega (como 2 crawlers más que no se llegan a crear o un clasificador

json)

```
# Script de limpieza completo - Data Lake Coches
# Elimina TODOS los recursos creados en la practica
# SEGURO: Solo elimina recursos especificos de este proyecto

Write-Host "========================================" -ForegroundColor
    Red
Write-Host "  CLEANUP COMPLETO - DATA LAKE COCHES" -ForegroundColor Red
Write-Host "========================================" -ForegroundColor
    Red

Write-Host "`nEste script eliminara TODOS los recursos del proyecto:" -
    ForegroundColor Yellow
Write-Host "  - Kinesis Stream + Firehose" -ForegroundColor Gray
Write-Host "  - Lambda Function" -ForegroundColor Gray
Write-Host "  - S3 Bucket completo (con datos)" -ForegroundColor Gray
Write-Host "  - Glue Database, Crawlers, Jobs y Tablas" -ForegroundColor
    Gray
Write-Host "  - Resultados de Athena" -ForegroundColor Gray

$confirmation = Read-Host "`nEstas seguro? Escribe SI para continuar"

if ($confirmation -ne "SI") {
    Write-Host "`nCleanup cancelado" -ForegroundColor Green
    exit 0
}

Write-Host "`nIniciando cleanup..." -ForegroundColor Yellow

$env:ACCOUNT_ID = (aws sts get-caller-identity --query Account --output
    text).Trim()
$env:BUCKET_NAME = "datalake-cars-$($env:ACCOUNT_ID)"

$deleted = 0

# ============================================
# 1. GLUE JOBS
# ============================================
Write-Host "`n[1/9] Eliminando Glue Jobs..." -ForegroundColor Cyan

$jobs = @("cars_brand_job", "cars_year_job")
foreach ($job in $jobs) {
    $result = aws glue delete-job --job-name $job 2>&1
    if ($LASTEXITCODE -eq 0) {
        Write-Host "  OK Job eliminado: $job" -ForegroundColor Green
        $deleted++
    } else {
```

```
        Write-Host "  - Job no existe: $job" -ForegroundColor Gray
    }
}

# ============================================
# 2. GLUE CRAWLERS
# ============================================
Write-Host "`n[2/9] Eliminando Glue Crawlers..." -ForegroundColor Cyan

$crawlers = @("cars_raw_crawler", "cars_processed_crawler", "
    cars_brand_crawler", "cars_year_crawler")
foreach ($crawler in $crawlers) {
    $result = aws glue delete-crawler --name $crawler 2>&1
    if ($LASTEXITCODE -eq 0) {
        Write-Host "  OK Crawler eliminado: $crawler" -ForegroundColor
    Green
        $deleted++
    } else {
        Write-Host "  - Crawler no existe: $crawler" -ForegroundColor
    Gray
    }
}

# ============================================
# 3. GLUE TABLES
# ============================================
Write-Host "`n[3/9] Eliminando Glue Tables..." -ForegroundColor Cyan

$tables = @("cars_data", "by_brand", "by_year")
foreach ($table in $tables) {
    $result = aws glue delete-table --database-name cars_db --name $table
    2>&1
    if ($LASTEXITCODE -eq 0) {
        Write-Host "  OK Tabla eliminada: $table" -ForegroundColor Green
        $deleted++
    } else {
        Write-Host "  - Tabla no existe: $table" -ForegroundColor Gray
    }
}

# ============================================
# 4. GLUE DATABASE
# ============================================
Write-Host "`n[4/9] Eliminando Glue Database..." -ForegroundColor Cyan

$result = aws glue delete-database --name cars_db 2>&1
if ($LASTEXITCODE -eq 0) {
```

```
    Write-Host "  OK Database eliminada: cars_db" -ForegroundColor Green
    $deleted++
} else {
    Write-Host "  - Database no existe" -ForegroundColor Gray
}

# ============================================
# 5. GLUE CLASSIFIER
# ============================================
Write-Host "`n[5/9] Eliminando Glue Classifier..." -ForegroundColor Cyan

$result = aws glue delete-classifier --name cars-json-classifier 2>&1
if ($LASTEXITCODE -eq 0) {
    Write-Host "  OK Classifier eliminado" -ForegroundColor Green
    $deleted++
} else {
    Write-Host "  - Classifier no existe" -ForegroundColor Gray
}

# ============================================
# 6. FIREHOSE DELIVERY STREAM
# ============================================
Write-Host "`n[6/9] Eliminando Firehose..." -ForegroundColor Cyan

$firehoseStatus = aws firehose describe-delivery-stream --delivery-stream
    -name cars-delivery-stream 2>$null
if ($LASTEXITCODE -eq 0) {
    aws firehose delete-delivery-stream --delivery-stream-name cars-
    delivery-stream 2>&1 | Out-Null
    Write-Host "  OK Firehose eliminado (tardara 1 min en completarse)" -
    ForegroundColor Green
    $deleted++
} else {
    Write-Host "  - Firehose no existe" -ForegroundColor Gray
}

# ============================================
# 7. KINESIS STREAM
# ============================================
Write-Host "`n[7/9] Eliminando Kinesis Stream..." -ForegroundColor Cyan

$streamStatus = aws kinesis describe-stream --stream-name cars-stream 2>
    $null
if ($LASTEXITCODE -eq 0) {
    aws kinesis delete-stream --stream-name cars-stream --enforce-
    consumer-deletion 2>&1 | Out-Null
    Write-Host "  OK Kinesis Stream eliminado" -ForegroundColor Green
```

```
    $deleted++
} else {
    Write-Host "  - Stream no existe" -ForegroundColor Gray
}

# ============================================
# 8. LAMBDA FUNCTION
# ============================================
Write-Host "`n[8/9] Eliminando Lambda Function..." -ForegroundColor Cyan

$lambdaStatus = aws lambda get-function --function-name
    cars_firehose_processor 2>$null
if ($LASTEXITCODE -eq 0) {
    aws lambda delete-function --function-name cars_firehose_processor
    2>&1 | Out-Null
    Write-Host "  OK Lambda eliminada: cars_firehose_processor" -
    ForegroundColor Green
    $deleted++
} else {
    Write-Host "  - Lambda no existe" -ForegroundColor Gray
}

# ============================================
# 9. S3 BUCKET (CON TODO EL CONTENIDO)
# ============================================
Write-Host "`n[9/9] Eliminando S3 Bucket..." -ForegroundColor Cyan

$bucketExists = aws s3 ls s3://$env:BUCKET_NAME 2>$null
if ($LASTEXITCODE -eq 0) {
    Write-Host "  Vaciando bucket..." -ForegroundColor Yellow

    # Contar objetos
    $objectCount = (aws s3 ls s3://$env:BUCKET_NAME --recursive 2>$null |
    Measure-Object).Count
    Write-Host "    $objectCount objetos a eliminar" -ForegroundColor
    Gray

    # Vaciar bucket
    aws s3 rm s3://$env:BUCKET_NAME --recursive --quiet

    # Eliminar bucket
    aws s3 rb s3://$env:BUCKET_NAME --force 2>&1 | Out-Null

    Write-Host "  OK Bucket eliminado: $env:BUCKET_NAME" -ForegroundColor
    Green
    $deleted++
} else {
```

```
    Write-Host "  - Bucket no existe" -ForegroundColor Gray
}

# ============================================
# 10. ARCHIVOS TEMPORALES LOCALES
# ============================================
Write-Host "`n[10/10] Limpiando archivos temporales locales..." -
    ForegroundColor Cyan

$tempFiles = @("firehose.zip", "table_update.json", "crawler_config.json"
    , "fix_table.json")
$localDeleted = 0
foreach ($file in $tempFiles) {
    if (Test-Path $file) {
        Remove-Item $file -Force
        Write-Host "  OK Eliminado: $file" -ForegroundColor Green
        $localDeleted++
    }
}

if ($localDeleted -eq 0) {
    Write-Host "  - No hay archivos temporales" -ForegroundColor Gray
}

# ============================================
# RESUMEN
# ============================================
Write-Host "`n========================================" -ForegroundColor
    Green
Write-Host "CLEANUP COMPLETADO" -ForegroundColor Green
Write-Host "========================================" -ForegroundColor
    Green

Write-Host "`nRecursos eliminados: $deleted" -ForegroundColor White

Write-Host "`nRecursos limpiados:" -ForegroundColor Cyan
Write-Host "  - Glue Jobs (2)" -ForegroundColor Gray
Write-Host "  - Glue Crawlers (hasta 4)" -ForegroundColor Gray
Write-Host "  - Glue Tables (3)" -ForegroundColor Gray
Write-Host "  - Glue Database (1)" -ForegroundColor Gray
Write-Host "  - Glue Classifier (1)" -ForegroundColor Gray
Write-Host "  - Kinesis Firehose (1)" -ForegroundColor Gray
Write-Host "  - Kinesis Stream (1)" -ForegroundColor Gray
Write-Host "  - Lambda Function (1)" -ForegroundColor Gray
Write-Host "  - S3 Bucket completo (1)" -ForegroundColor Gray
Write-Host "  - Archivos temporales locales" -ForegroundColor Gray
```

```
Write-Host "`nNOTA:" -ForegroundColor Yellow
Write-Host "  - Firehose puede tardar 1 minuto en eliminarse
    completamente" -ForegroundColor Gray
Write-Host "  - Kinesis Stream se marca para eliminacion (24h)" -
    ForegroundColor Gray
Write-Host "  - Tu cuenta AWS no se ha visto comprometida" -
    ForegroundColor Gray
Write-Host "  - Solo se eliminaron recursos de este proyecto" -
    ForegroundColor Gray

Write-Host "`nVerificar limpieza:" -ForegroundColor Cyan
Write-Host "  aws glue get-databases" -ForegroundColor White
Write-Host "  aws s3 ls" -ForegroundColor White
Write-Host "  aws kinesis list-streams" -ForegroundColor White
Write-Host "  aws lambda list-functions" -ForegroundColor White

Write-Host "`nCleanup finalizado" -ForegroundColor Green
```

Listing 22: Cleanup Data Lake - cleanup_cars.ps1

3.1.6 A.6. Script de prueba para consultas a Athena

```
# Script de consultas simplificado - Data Lake Coches


Write-Host "========================================" -ForegroundColor
    Cyan
Write-Host "  CONSULTAS SIMPLIFICADAS" -ForegroundColor Cyan
Write-Host "========================================" -ForegroundColor
    Cyan


$env:ACCOUNT_ID = (aws sts get-caller-identity --query Account --output
    text).Trim()
$env:BUCKET_NAME = "datalake-cars-$($env:ACCOUNT_ID)"
$athenaOutput = "s3://$env:BUCKET_NAME/queries/"


# ============================================
# 1. ESTADISTICAS BASICAS
# ============================================
Write-Host "`n[1] ESTADISTICAS BASICAS" -ForegroundColor Yellow
Write-Host "====================================" -ForegroundColor Gray


# Total registros
Write-Host "`nTotal de vehiculos en el dataset:" -ForegroundColor Cyan
```

```
$q1 = (aws athena start-query-execution --query-string "SELECT COUNT(*)
    as total FROM cars_db.cars_data;" --result-configuration
    OutputLocation=$athenaOutput --query-execution-context Database=
    cars_db --query QueryExecutionId --output text).Trim()
Start-Sleep -Seconds 5
$total = aws athena get-query-results --query-execution-id $q1 --query '
    ResultSet.Rows[1].Data[0].VarCharValue' --output text
Write-Host "  $total vehiculos" -ForegroundColor White


# Total de marcas
Write-Host "`nTotal de marcas diferentes:" -ForegroundColor Cyan
$q2 = (aws athena start-query-execution --query-string "SELECT COUNT(*)
    as total_brands FROM cars_db.by_brand;" --result-configuration
    OutputLocation=$athenaOutput --query-execution-context Database=
    cars_db --query QueryExecutionId --output text).Trim()
Start-Sleep -Seconds 5
$brands = aws athena get-query-results --query-execution-id $q2 --query '
    ResultSet.Rows[1].Data[0].VarCharValue' --output text
Write-Host "  $brands marcas" -ForegroundColor White


# ============================================
# 2. TOP RANKINGS
# ============================================
Write-Host "`n[2] TOP RANKINGS" -ForegroundColor Yellow
Write-Host "====================================" -ForegroundColor Gray


# Top 5 marcas por cantidad
Write-Host "`nTop 5 marcas con mas vehiculos:" -ForegroundColor Cyan
$q3 = (aws athena start-query-execution --query-string "SELECT brand,
    total_vehicles FROM cars_db.by_brand ORDER BY total_vehicles DESC
    LIMIT 5;" --result-configuration OutputLocation=$athenaOutput --query-
    execution-context Database=cars_db --query QueryExecutionId --output
    text).Trim()
Start-Sleep -Seconds 5
$top5brands = aws athena get-query-results --query-execution-id $q3 --
    query 'ResultSet.Rows[1:]' --output json | ConvertFrom-Json
foreach ($row in $top5brands) {
    $brand = $row.Data[0].VarCharValue
    $count = $row.Data[1].VarCharValue
    Write-Host "  $brand : $count vehiculos" -ForegroundColor White
}


# Top 3 anos con mas vehiculos
```

```
Write-Host "`nTop 3 anos con mas vehiculos:" -ForegroundColor Cyan
$q4 = (aws athena start-query-execution --query-string "SELECT model_year
    , total_vehicles FROM cars_db.by_year ORDER BY total_vehicles DESC
    LIMIT 3;" --result-configuration OutputLocation=$athenaOutput --query-
    execution-context Database=cars_db --query QueryExecutionId --output
    text).Trim()
Start-Sleep -Seconds 5
$top3years = aws athena get-query-results --query-execution-id $q4 --
    query 'ResultSet.Rows[1:]' --output json | ConvertFrom-Json
foreach ($row in $top3years) {
    $year = $row.Data[0].VarCharValue
    $count = $row.Data[1].VarCharValue
    Write-Host "  Ano $year : $count vehiculos" -ForegroundColor White
}


# ============================================
# 3. CONSULTAS CON FILTROS (USA PARTICIONES)
# ============================================
Write-Host "`n[3] CONSULTAS CON FILTROS" -ForegroundColor Yellow
Write-Host "====================================" -ForegroundColor Gray


# Filtro por marca especifica - BMW
Write-Host "`nEstadisticas de BMW:" -ForegroundColor Cyan
$q5 = (aws athena start-query-execution --query-string "SELECT brand,
    total_vehicles, CAST(avg_mileage AS INT) as avg_km FROM cars_db.
    by_brand WHERE brand = 'BMW';" --result-configuration OutputLocation=
    $athenaOutput --query-execution-context Database=cars_db --query
    QueryExecutionId --output text).Trim()
Start-Sleep -Seconds 5
$bmw = aws athena get-query-results --query-execution-id $q5 --query '
    ResultSet.Rows[1]' --output json | ConvertFrom-Json
$bmw_count = $bmw.Data[1].VarCharValue
$bmw_avg = $bmw.Data[2].VarCharValue
Write-Host "  Vehiculos: $bmw_count" -ForegroundColor White
Write-Host "  Kilometraje promedio: $bmw_avg km" -ForegroundColor White


# Filtro por marca especifica - Toyota
Write-Host "`nEstadisticas de Toyota:" -ForegroundColor Cyan
$q6 = (aws athena start-query-execution --query-string "SELECT brand,
    total_vehicles, CAST(avg_mileage AS INT) as avg_km FROM cars_db.
    by_brand WHERE brand = 'Toyota';" --result-configuration
    OutputLocation=$athenaOutput --query-execution-context Database=
    cars_db --query QueryExecutionId --output text).Trim()
Start-Sleep -Seconds 5
```

```
$toyota = aws athena get-query-results --query-execution-id $q6 --query '
    ResultSet.Rows[1]' --output json | ConvertFrom-Json
$toyota_count = $toyota.Data[1].VarCharValue
$toyota_avg = $toyota.Data[2].VarCharValue
Write-Host "  Vehiculos: $toyota_count" -ForegroundColor White
Write-Host "  Kilometraje promedio: $toyota_avg km" -ForegroundColor
    White


# ============================================
# 4. CONSULTAS COMPLEJAS
# ============================================
Write-Host "`n[4] CONSULTAS COMPLEJAS" -ForegroundColor Yellow
Write-Host "====================================" -ForegroundColor Gray


# Marcas con mayor kilometraje promedio
Write-Host "`nTop 3 marcas con MAYOR kilometraje promedio:" -
    ForegroundColor Cyan
$q7 = (aws athena start-query-execution --query-string "SELECT brand,
    CAST(avg_mileage AS INT) as avg_km, total_vehicles FROM cars_db.
    by_brand WHERE total_vehicles > 50 ORDER BY avg_mileage DESC LIMIT 3;"
     --result-configuration OutputLocation=$athenaOutput --query-execution
    -context Database=cars_db --query QueryExecutionId --output text).Trim
    ()
Start-Sleep -Seconds 5
$topkm = aws athena get-query-results --query-execution-id $q7 --query '
    ResultSet.Rows[1:]' --output json | ConvertFrom-Json
foreach ($row in $topkm) {
    $brand = $row.Data[0].VarCharValue
    $km = $row.Data[1].VarCharValue
    $count = $row.Data[2].VarCharValue
    Write-Host "  $brand : $km km promedio ($count vehiculos)" -
    ForegroundColor White
}


# Marcas con menor kilometraje promedio
Write-Host "`nTop 3 marcas con MENOR kilometraje promedio:" -
    ForegroundColor Cyan
$q8 = (aws athena start-query-execution --query-string "SELECT brand,
    CAST(avg_mileage AS INT) as avg_km, total_vehicles FROM cars_db.
    by_brand WHERE total_vehicles > 50 ORDER BY avg_mileage ASC LIMIT 3;"
    --result-configuration OutputLocation=$athenaOutput --query-execution-
    context Database=cars_db --query QueryExecutionId --output text).Trim
    ()
Start-Sleep -Seconds 5
```

```
$lowkm = aws athena get-query-results --query-execution-id $q8 --query '
    ResultSet.Rows[1:]' --output json | ConvertFrom-Json
foreach ($row in $lowkm) {
    $brand = $row.Data[0].VarCharValue
    $km = $row.Data[1].VarCharValue
    $count = $row.Data[2].VarCharValue
    Write-Host "  $brand : $km km promedio ($count vehiculos)" -
    ForegroundColor White
}


# Anos con mas diversidad de marcas
Write-Host "`nTop 3 anos con MAS diversidad de marcas:" -ForegroundColor
    Cyan
$q9 = (aws athena start-query-execution --query-string "SELECT model_year
    , unique_brands, total_vehicles FROM cars_db.by_year ORDER BY
    unique_brands DESC LIMIT 3;" --result-configuration OutputLocation=
    $athenaOutput --query-execution-context Database=cars_db --query
    QueryExecutionId --output text).Trim()
Start-Sleep -Seconds 5
$diversity = aws athena get-query-results --query-execution-id $q9 --
    query 'ResultSet.Rows[1:]' --output json | ConvertFrom-Json
foreach ($row in $diversity) {
    $year = $row.Data[0].VarCharValue
    $brands = $row.Data[1].VarCharValue
    $count = $row.Data[2].VarCharValue
    Write-Host "  Ano $year : $brands marcas diferentes ($count vehiculos
    )" -ForegroundColor White
}


# ============================================
# 5. CONSULTA AGREGADA COMPLEJA
# ============================================
Write-Host "`n[5] CONSULTA AGREGADA COMPLEJA" -ForegroundColor Yellow
Write-Host "======================================" -ForegroundColor Gray


# Marcas premium (mas de 60 vehiculos Y kilometraje > 120k)
Write-Host "`nMarcas premium (>60 vehiculos Y >120k km promedio):" -
    ForegroundColor Cyan
$q10 = (aws athena start-query-execution --query-string "SELECT brand,
    total_vehicles, CAST(avg_mileage AS INT) as avg_km FROM cars_db.
    by_brand WHERE total_vehicles > 60 AND avg_mileage > 120000 ORDER BY
    total_vehicles DESC;" --result-configuration OutputLocation=
    $athenaOutput --query-execution-context Database=cars_db --query
    QueryExecutionId --output text).Trim()
```

```
Start-Sleep -Seconds 5
$premium = aws athena get-query-results --query-execution-id $q10 --query
    'ResultSet.Rows[1:]' --output json | ConvertFrom-Json
if ($premium.Count -gt 0) {
    foreach ($row in $premium) {
        $brand = $row.Data[0].VarCharValue
        $count = $row.Data[1].VarCharValue
        $km = $row.Data[2].VarCharValue
        Write-Host "  $brand : $count vehiculos, $km km promedio" -
    ForegroundColor White
    }
} else {
    Write-Host "  No hay marcas que cumplan estos criterios" -
    ForegroundColor Gray
}


# Marcas con menos de 50 vehiculos
Write-Host "`nMarcas exclusivas (<50 vehiculos):" -ForegroundColor Cyan
$q11 = (aws athena start-query-execution --query-string "SELECT brand,
    total_vehicles, CAST(avg_mileage AS INT) as avg_km FROM cars_db.
    by_brand WHERE total_vehicles < 50 ORDER BY total_vehicles DESC LIMIT
    5;" --result-configuration OutputLocation=$athenaOutput --query-
    execution-context Database=cars_db --query QueryExecutionId --output
    text).Trim()
Start-Sleep -Seconds 5
$exclusive = aws athena get-query-results --query-execution-id $q11 --
    query 'ResultSet.Rows[1:]' --output json | ConvertFrom-Json
foreach ($row in $exclusive) {
    $brand = $row.Data[0].VarCharValue
    $count = $row.Data[1].VarCharValue
    $km = $row.Data[2].VarCharValue
    Write-Host "  $brand : $count vehiculos, $km km promedio" -
    ForegroundColor White
}


Write-Host "`n======================================" -ForegroundColor
    Green
Write-Host "CONSULTAS COMPLETADAS" -ForegroundColor Green
Write-Host "======================================" -ForegroundColor
    Green
```

Listing 23: Queries a Athena - query_simple.ps1

3.1.7 A.7. Ejemplo de datos JSON

A continuación se muestra una muestra de los registros JSON que se envían a Kinesis Data Streams. Cada registro representa un vehículo con sus características principales:


```
[
  {
    "id": 55000,
    "brand": "INFINITI",
    "model": "Model 5",
    "model_year": 2020,
    "milage": 63075,
    "fuel_type": "Gasoline",
    "engine": "477.0HP 3L 4 Cylinder Engine Gasoline Fuel",
    "transmission": "5-Speed A/T",
    "ext_col": "Purple",
    "int_col": "Gray",
    "accident": "At least 1 accident or damage reported",
    "clean_title": "Yes"
  },
  {
    "id": 55001,
    "brand": "Subaru",
    "model": "Model 34",
    "model_year": 2001,
    "milage": 85701,
    "fuel_type": "Hybrid",
    "engine": "195.0HP 6L V6 Engine Gasoline Fuel",
    "transmission": "Transmission w/Dual Shift Mode",
    "ext_col": "White",
    "int_col": "Blue",
    "accident": "At least 1 accident or damage reported",
    "clean_title": "Yes"
  },
  {
    "id": 55002,
    "brand": "Honda",
    "model": "Model 35",
    "model_year": 2008,
    "milage": 179052,
    "fuel_type": "Diesel",
    "engine": "206.0HP 3L V8 Engine Gasoline Fuel",
    "transmission": "5-Speed A/T",
    "ext_col": "Green",
    "int_col": "Black",
    "accident": "At least 1 accident or damage reported",
    "clean_title": "Yes"
  },
```

```
{
  "id": 55003,
  "brand": "GMC",
  "model": "Model 93",
  "model_year": 2000,
  "milage": 191237,
  "fuel_type": "E85 Flex Fuel",
  "engine": "261.0HP 4L 4 Cylinder Engine Gasoline Fuel",
  "transmission": "6-Speed M/T",
  "ext_col": "Red",
  "int_col": "Blue",
  "accident": "None reported",
  "clean_title": "Yes"
},
{
  "id": 55004,
  "brand": "BMW",
  "model": "335i",
  "model_year": 2008,
  "milage": 202721,
  "fuel_type": "Diesel",
  "engine": "251.0HP 4L V8 Engine Gasoline Fuel",
  "transmission": "8-Speed A/T",
  "ext_col": "Gray",
  "int_col": "Beige",
  "accident": "At least 1 accident or damage reported",
  "clean_title": "Yes"
},
{
  "id": 55005,
  "brand": "BMW",
  "model": "335i",
  "model_year": 2020,
  "milage": 197935,
  "fuel_type": "Gasoline",
  "engine": "506.0HP 6L V6 Engine Gasoline Fuel",
  "transmission": "10-Speed A/T",
  "ext_col": "Purple",
  "int_col": "Beige",
  "accident": "At least 1 accident or damage reported",
  "clean_title": "Yes"
},
{
  "id": 55006,
  "brand": "Volkswagen",
  "model": "Model 76",
  "model_year": 2010,
```

```
    "milage": 150880,
    "fuel_type": "Hybrid",
    "engine": "241.0HP 6L V8 Engine Gasoline Fuel",
    "transmission": "6-Speed M/T",
    "ext_col": "Blue",
    "int_col": "Red",
    "accident": "None reported",
    "clean_title": "Yes"
  },
  {
    "id": 55007,
    "brand": "MINI",
    "model": "Model 71",
    "model_year": 2001,
    "milage": 124338,
    "fuel_type": "Diesel",
    "engine": "357.0HP 5L V8 Engine Gasoline Fuel",
    "transmission": "A/T",
    "ext_col": "Gray",
    "int_col": "Orange",
    "accident": "At least 1 accident or damage reported",
    "clean_title": "Yes"
  },
  {
    "id": 55008,
    "brand": "Maserati",
    "model": "Model 41",
    "model_year": 2001,
    "milage": 245836,
    "fuel_type": "Hybrid",
    "engine": "353.0HP 5L V6 Engine Gasoline Fuel",
    "transmission": "A/T",
    "ext_col": "Green",
    "int_col": "-",
    "accident": "At least 1 accident or damage reported",
    "clean_title": "Yes"
  }
]
```


Listing 24: Ejemplo de datos - sample_cars.json

Descripción de los campos Cada registro JSON contiene los siguientes campos:

- **id**: Identificador único del vehículo
- **brand**: Marca del vehículo (ej: BMW, Honda, Toyota)

- **model**: Modelo específico del vehículo
- **model_year**: Año de fabricación (2000-2024)
- **milage**: Kilometraje total del vehículo
- **fuel_type**: Tipo de combustible (Gasoline, Diesel, Hybrid, E85 Flex Fuel)
- **engine**: Especificaciones del motor (potencia, cilindrada, configuración)
- **transmission**: Tipo de transmisión (automática/manual y número de velocidades)
- **ext_col**: Color exterior del vehículo
- **int_col**: Color interior del vehículo
- **accident**: Historial de accidentes reportados
- **clean_title**: Indica si el título está limpio (Yes/No)

Estos registros son procesados por el script `kinesis.py`, que los envía en lotes a Kinesis Data Streams para su posterior transformación y almacenamiento en el Data Lake.

3.2 Anexo B: Uso de Inteligencia Artificial

Durante la realización de esta práctica se ha utilizado inteligencia artificial (IA) como herramienta de apoyo en diversas fases del desarrollo. A continuación se detalla el uso específico de cada herramienta:

3.2.1 Herramientas utilizadas

- **ChatGPT (OpenAI)**
- **Google Gemini**
- **Perplexity AI**

3.2.2 Tareas asistidas por IA

Desarrollo de código

- Debugging de errores de codificación y configuración en Lambda y Firehose.
- Generación de la estructura inicial del script de automatización `setup.ps1`.
- Optimización de queries PySpark en los Jobs ETL de Glue.
- Generación de código para manejo de excepciones y logging en el productor de datos.
- Implementación de la lógica de particionado dinámico en Kinesis Firehose.

Estudio y comprensión de tecnologías

- Consulta sobre conceptos y arquitecturas de servicios cloud para procesamiento de datos en streaming.
- Investigación sobre buenas prácticas en diseño de pipelines ETL, optimización de costes y rendimiento.
- Estudio de bibliotecas y herramientas para desarrollo e integración con plataformas cloud.

Documentación y análisis

- Redacción y estructura de secciones de la memoria (Introducción, Desarrollo, Conclusiones).
- Generación de la tabla de estimación de costes y cálculos asociados.
- Elaboración de descripciones técnicas del flujo de datos y arquitectura.
- Formato y corrección de código LaTeX para tablas y listados.

3.2.3 Validación y trabajo independiente

A pesar del uso extensivo de herramientas de IA, todo el código generado ha sido:

- **Revisado y comprendido**: Se ha analizado cada línea de código para entender su funcionamiento y asegurar que cumple con los requisitos de la práctica.
- **Probado exhaustivamente**: Todos los componentes han sido ejecutados en AWS, depurados y ajustados hasta lograr un funcionamiento correcto.
- **Adaptado al contexto**: El código generado por IA fue modificado para ajustarse a los datos específicos del dataset de vehículos y a la arquitectura diseñada.

Las decisiones arquitectónicas principales (elección de servicios, diseño del flujo de datos, estrategia de particionado, frecuencia de ETL) fueron tomadas de forma completamente independiente, utilizando la IA únicamente como herramienta de implementación y consulta técnica.