

Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería Informática

Grado en Ingeniería Informática

Práctica 7

Computación en la Nube

Curso académico: 2025–2026

Estudiante: Iván Pérez Díaz

Fecha de entrega: 16 de enero de 2026

Índice

1. Introducción	2
2. Desarrollo de las actividades	2
2.1. Configuración del bucket S3	2
2.2. Implementación del productor de datos	3
2.3. Configuración del consumidor (Kinesis Firehose)	3
2.4. Configuración de AWS Glue	4
3. Diagrama del flujo de datos	4
4. Presupuesto y estimación de costes	5
5. Conclusiones	6
6. Referencias y bibliografía	6
7. Anexos	7
7.1. Anexo A. Runmap de Ejecución	7
7.2. Anexo B. Script de despliegue	15
7.3. Anexo C. Productor Kinesis	21
7.4. Anexo D. Transformación Firehose	22
7.5. Anexo E. Análisis por Marca (Spark)	23
7.6. Anexo F. Análisis por Sistema Operativo (Spark)	25
7.7. Anexo G. Crawler de Resultados	27
7.8. Anexo H. Script de limpieza	27
7.9. Anexo I. Dataset de Origen	29
7.10. Anexo J. Uso de la Inteligencia Artificial Generativa	29

1 Introducción

El objetivo principal de esta práctica es el diseño, despliegue y validación de una arquitectura completa en la nube, utilizando el ecosistema de servicios gestionados de Amazon Web Services (AWS). Se busca implementar un pipeline de datos robusto que abarque desde la generación de la información en tiempo real hasta las validaciones finales con SQL.

La solución adopta un enfoque *serverless* (sin servidor), minimizando la carga operativa de administración de infraestructura y permitiendo un escalado automático. Los servicios clave seleccionados y su función en la arquitectura son:

- **Amazon S3 (Simple Storage Service):** Actúa como la capa de persistencia centralizada (Data Lake). Proporciona almacenamiento de objetos altamente escalable, duradero y seguro, permitiendo separar los datos en diferentes zonas según su estado de procesamiento (Raw vs Processed).
- **Amazon Kinesis Data Streams:** Facilita la ingesta de datos de alta velocidad y baja latencia. Desacopla los productores de los consumidores, permitiendo que múltiples aplicaciones procesen el flujo de datos simultáneamente.
- **Amazon Kinesis Data Firehose:** Gestiona la entrega fiable de los datos en streaming hacia el almacenamiento en S3. Incluye capacidades de transformación en vuelo (mediante AWS Lambda) y gestión de buffers para optimizar la escritura de ficheros.
- **AWS Glue:** Proporciona un entorno unificado para el descubrimiento de metadatos (Data Catalog) y la ejecución de trabajos ETL (Extract, Transform, Load) basados en Apache Spark, facilitando la limpieza y agregación de datos sin gestionar servidores.

La arquitectura implementada procesa un conjunto de datos simulado referente a un inventario transaccional de dispositivos portátiles. Cada registro contiene atributos heterogéneos como marca, modelo, resolución de pantalla, especificaciones de hardware (CPU, RAM, GPU), sistema operativo y precio (datos.json). El flujo termina con la generación de datasets analíticos en formato columnar (Parquet), listos para ser consultados mediante SQL en herramientas como Amazon Athena.

2 Desarrollo de las actividades

2.1 Configuración del bucket S3

La creación y configuración del bucket S3 se realiza de forma automatizada mediante el script de infraestructura como código (IaC) `deploy.ps1`. El nombre del bucket se genera de forma dinámica concatenando el prefijo `datalake-laptops-` con el identificador único de la cuenta AWS (Account ID), asegurando así un espacio de nombres globalmente único, requisito indispensable en S3.

```

1 $env:BUCKET_NAME = "datalake-laptops-${$env:ACCOUNT_ID}"
2 aws s3 mb s3://$env:BUCKET_NAME 2>$null

```

Listing 1: Generación dinámica del nombre del bucket en deploy.ps1

La estructura de carpetas definida sigue las de diseño de Data Lakes, organizando los datos por capas ("zonas") para facilitar la gobernanza y el ciclo de vida de la información:

- **raw/**: Conocida como "Landing Zone". Almacena los datos en bruto tal cual son recibidos desde Kinesis Firehose, en formato JSON. Esta capa es inmutable y sirve como fuente de verdad histórica.
- **processed/**: Contiene los resultados refinados generados por los trabajos ETL. Los datos aquí están limpios, enriquecidos y convertidos a formato Parquet.
- **config/, scripts/, logs/**: Carpetas de soporte para scripts, configuraciones y auditoría.

2.2 Implementación del productor de datos

El componente productor es una aplicación desarrollada en Python (`kinesis.py`) que simula la generación de eventos de negocio en tiempo real. Utilizando la librería `boto3`, el script lee un dataset base y envía registros individuales al servicio Amazon Kinesis Data Streams.

Un aspecto técnico crucial es el uso de la **Partition Key**. Cada registro enviado incluye una clave de partición basada en la marca del dispositivo (`Company`). Esto asegura que todos los datos de una misma marca (ej. `Apple` o `Dell`) se dirijan siempre al mismo shard, manteniendo el orden secuencial.

```

1 response = kinesis.put_record(
2     StreamName=STREAM_NAME,
3     Data=json.dumps(laptop),
4     PartitionKey=str(company) # Agrupar shards por marca
5 )

```

Listing 2: Envío de registros con Partition Key en kinesis.py

2.3 Configuración del consumidor (Kinesis Firehose)

Amazon Kinesis Data Firehose actúa como el puente entre el flujo de datos en tiempo real y el almacenamiento persistente. Antes de escribir en S3, Firehose invoca una función AWS Lambda (`firehose.py`) que realiza tres tareas fundamentales:

1. **Enriquecimiento temporal**: Añade un campo `processed_at`.

2. **Formato JSON Lines:** Convierte el objeto JSON estándar añadiendo un salto de línea, requisito para que Athena procese múltiples objetos.
3. **Particionamiento Dinámico:** Define la clave de partición basada en la fecha.

```

1 # Re-codificar para Firehose (Debe terminar en salto de línea)
2 output_payload = json.dumps(data_json) + '\n'

```

Listing 3: Transformación y formato JSON Lines en firehose.py

2.4 Configuración de AWS Glue

AWS Glue se utiliza para el descubrimiento de metadatos y el procesamiento ETL. Primero, el script `deploy.ps1` configura el crawler `laptops-raw-crawler` que infiere el esquema de los datos en bruto.

Posteriormente, se implementan dos trabajos (*Glue Jobs*) basados en Apache Spark. Un desafío técnico resuelto fue la inconsistencia de tipos de datos en el campo de precio (*Schema Evolution*), donde algunos valores eran enteros y otros decimales. Se utilizó `resolveChoice` para unificar el tipo a `double`.

```

1 # CORRECCION DE TIPO DE DATO (CHOICE)
2 # Esto fuerza a que Price_euros sea tratado siempre como double
3 try:
4     dynamic_frame = dynamic_frame.resolveChoice(specs = [('Price_euros', 'cast:double')])
5 except:
6     logger.warning("No se requirio resolveChoice o fallo...")

```

Listing 4: Resolución de ambigüedad de tipos en `laptops_analytics_brand.py`

3 Diagrama del flujo de datos

La arquitectura implementada establece un pipeline unidireccional que garantiza la integridad, trazabilidad y escalabilidad del dato. El flujo de la información atraviesa las siguientes etapas secuenciales:

1. **Generación:** El script productor inyecta registros JSON en el Stream de Kinesis simulando transacciones.
2. **Transporte y Buffer:** Kinesis Data Streams recibe los datos y Kinesis Firehose los consume, gestionando la memoria intermedia.
3. **Transformación en Vuelo:** Firehose invoca a la función Lambda para añadir metadatos temporales y formatear a JSON Lines.

4. **Almacenamiento Raw:** Firehose vuelca los bloques de datos en la zona raw de S3, particionados por fecha.
5. **Catalogación:** El Crawler de Glue escanea S3 y actualiza el esquema en el Catálogo de Datos.
6. **Procesamiento Batch:** Los Glue Jobs leen del catálogo, ejecutan transformaciones Spark en memoria distribuida y agregan la información.
7. **Almacenamiento Processed:** Los resultados finales se escriben en la zona processed de S3 en formato Parquet.
8. **Consumo:** Amazon Athena utiliza el catálogo actualizado para lanzar consultas SQL sobre los datos procesados.

La Figura 1 ilustra gráficamente la topología completa de la solución desplegada.

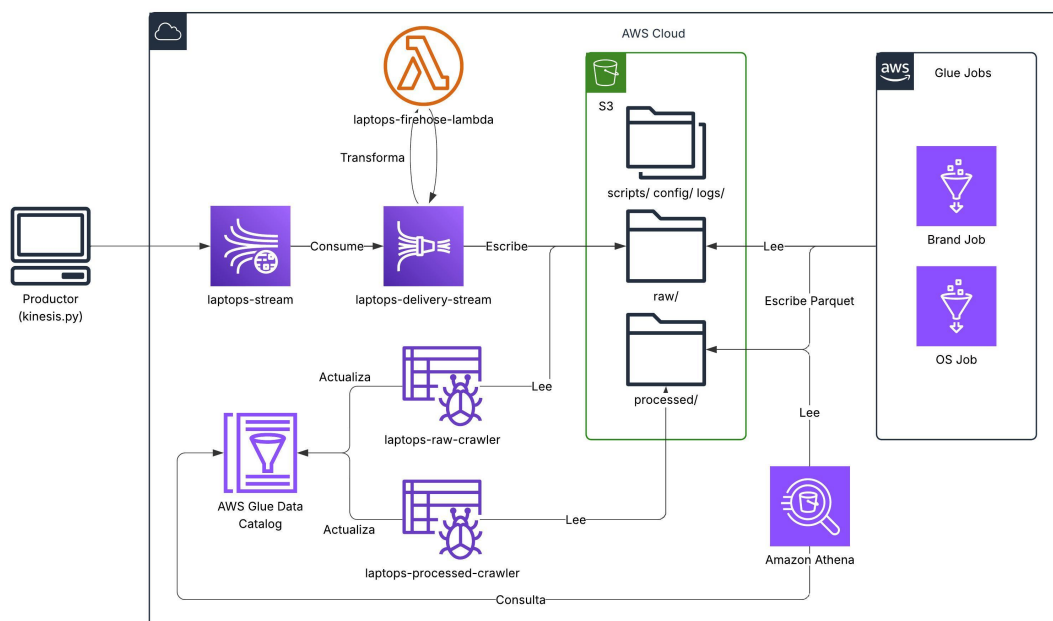


Figura 1: Diagrama de arquitectura del Data Lake Serverless en AWS.

4 Presupuesto y estimación de costes

La gestión de costes es fundamental en arquitectura Cloud. La estimación presentada a continuación se basa en la calculadora oficial de AWS (AWS Pricing Calculator) para la región `us-east-1`, considerando un escenario de operación mensual estándar con un flujo de datos moderado.

- **Kinesis:** 1 Shard activo durante 730 horas/mes (24x7).

- **Glue:** Ejecución diaria de Crawlers y Jobs ETL (Worker Type G.1X), consumiendo recursos bajo demanda.

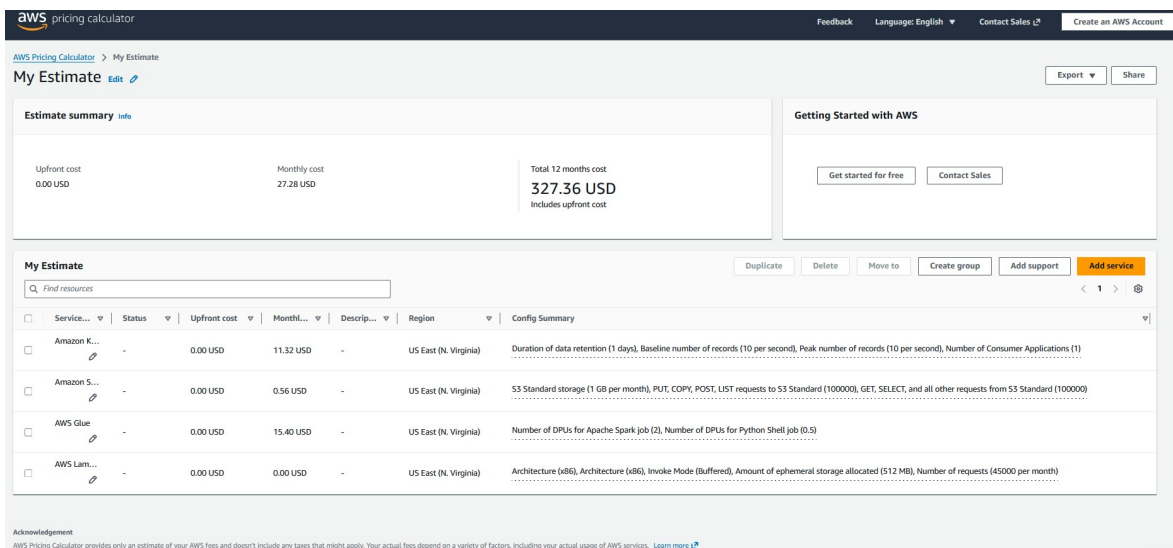


Figura 2: Captura de la calculadora oficial de AWS con el desglose de costes estimado.

5 Conclusiones

En esta práctica se han asentado conocimientos sobre el diseño y despliegue de una arquitectura para gestionar la incerción y procesamiento de datos dentro de la nube de AWS, adicionalmente, se ha aprendido a utilizar servicios como Kinesis, AWS Glue y S3.

Entre las principales lecciones aprendidas y dificultades superadas, destacan:

- **Schema Evolution:** La importancia de manejar la heterogeneidad de los datos numéricos (enteros vs floats) en Spark mediante `resolveChoice`, dado que el desconocimiento de las mismas ocasionaron muchos problemas y errores.
- **Infraestructura como Código:** El uso de scripts PowerShell (`deploy`, `clean` y `aws-crawler`) que ha permitido desplegar y destruir la infraestructura de forma repetible, evitando costes fantasma y de manera automatizada.
- **Particionamiento:** Se ha comprobado cómo el particionamiento dinámico en la ingesta mejora la organización del Data Lake.

6 Referencias y bibliografía

- Amazon Web Services. (2026). *Amazon Kinesis Data Streams Developer Guide*. Recuperado de la documentación oficial de AWS.
- Amazon Web Services. (2026). *AWS Glue Developer Guide*. Sección sobre ETL Jobs y DynamicFrames.

- Amazon Web Services. (2026). *Amazon S3 User Guide*. Best practices for Data Lakes.
- Apache Spark. (2026). *PySpark Documentation*. DataFrame API y tipos de datos.
- Amazon Web Services. (2026). *AWS Pricing Calculator* [Web application]. <https://calculator.aws/>
- Lucid Software Inc. (2026). *Lucidchart* [Web application]. <https://www.lucidchart.com/>
- Google. (2026). *Gemini* (Versión 1.5) [Large language model]. <https://gemini.google.com/>
- OpenAI. (2026). *ChatGPT* (Versión GPT-4) [Large language model]. <https://chat.openai.com/>
- Mohammad Anas. (2023). *Laptop Price Dataset* [Data set]. Kaggle. <https://www.kaggle.com/datasets>
- Pérez, I. (2026). *Cloud-Computing-AWS-2* [Software repository]. GitHub. <https://github.com/ivanpere/Cloud-Computing-AWS-2>

7 Anexos

El código fuente completo, así como el historial de versiones y la documentación adicional, se encuentra disponible en el siguiente repositorio público de GitHub:

<https://github.com/ivanperezdiaz829/Cloud-Computing-AWS-2>

A continuación se incluye una copia estática de los scripts principales utilizados para la realización de la práctica.

7.1 Anexo A. Runmap de Ejecución

Es una guía paso a paso de las ejecuciones necesarias para desplegar, probar y destruir la arquitectura.

```

1 # EJECUTAR SOLO UNA VEZ SI NO SE TIENE EL ENVIRONMENT
2 # =====
3 # uv init
4 # uv add boto3
5 # uv add loguru
6 # uv venv
7 # =====
8
9 # Activar Environment
10 .venv\Scripts\activate
11
12 # RELLENAR ANTES DE EJECUTAR
13 # =====
14 # $env:AWS_ACCESS_KEY_ID="PONER DATOS DE AWS"
15 # $env:AWS_SECRET_ACCESS_KEY="PONER DATOS DE AWS"

```



```

16 # $env:AWS_SESSION_TOKEN="PONER DATOS DE AWS"
17 # =====
18
19 # Comprobar que AWS responde a la cuenta puesta con las variables, si
    devuelve un JSON con la cuenta y el Arn continuar
20 aws sts get-caller-identity
21
22 # PASO 1: Desplegar la infraestructura
23 # =====
24 .\deploy.ps1
25
26 # PASO 2: Simular el envio de datos a Kinesis
27 # =====
28 uv run kinesis.py
29
30 # PASO 3: Iniciar el Crawler de Glue para catalogar datos en bruto
31 # =====
32 aws glue start-crawler --name laptops-raw-crawler
33 # Ejecutar para ver el estado (termina cuando READY)
34 aws glue get-crawler --name laptops-raw-crawler --query "Crawler.State
    "
35
36 # PASO 4: Ejecutar los JOBS de Glue para transformar y almacenar datos
    procesados
37 # =====
38 aws glue start-job-run --job-name laptops-analytics-brand
39 aws glue start-job-run --job-name laptops-analytics-opsys
40 # Ejecutar para ver el estado (termina cuando READY)
41 aws glue get-job-runs --job-name laptops-analytics-brand --max-results
    1 --query "JobRuns[0].JobRunState"
42 aws glue get-job-runs --job-name laptops-analytics-opsys --max-results
    1 --query "JobRuns[0].JobRunState"
43
44 # PASO 5: Iniciar el Crawler de Glue para catalogar datos procesados
45 # =====
46 .\aws_crawler.ps1
47
48 # PASO 6: Consultar AWS Athena y realizar varias sentencias SQL
49
50 # PASO 7: Limpiar la cuenta de recursos para evitar costes (comprobar
    en AWS que se han borrado correctamente)
51 # =====
52 .\clean.ps1

```

Listing 5: Script runmap.ps1: Ejecución completa de la práctica

Guía Visual de Ejecución

Paso 1: Despliegue de Infraestructura

Ejecución de `deploy.ps1`. Se valida la creación del Bucket S3 que actuará como Data Lake.

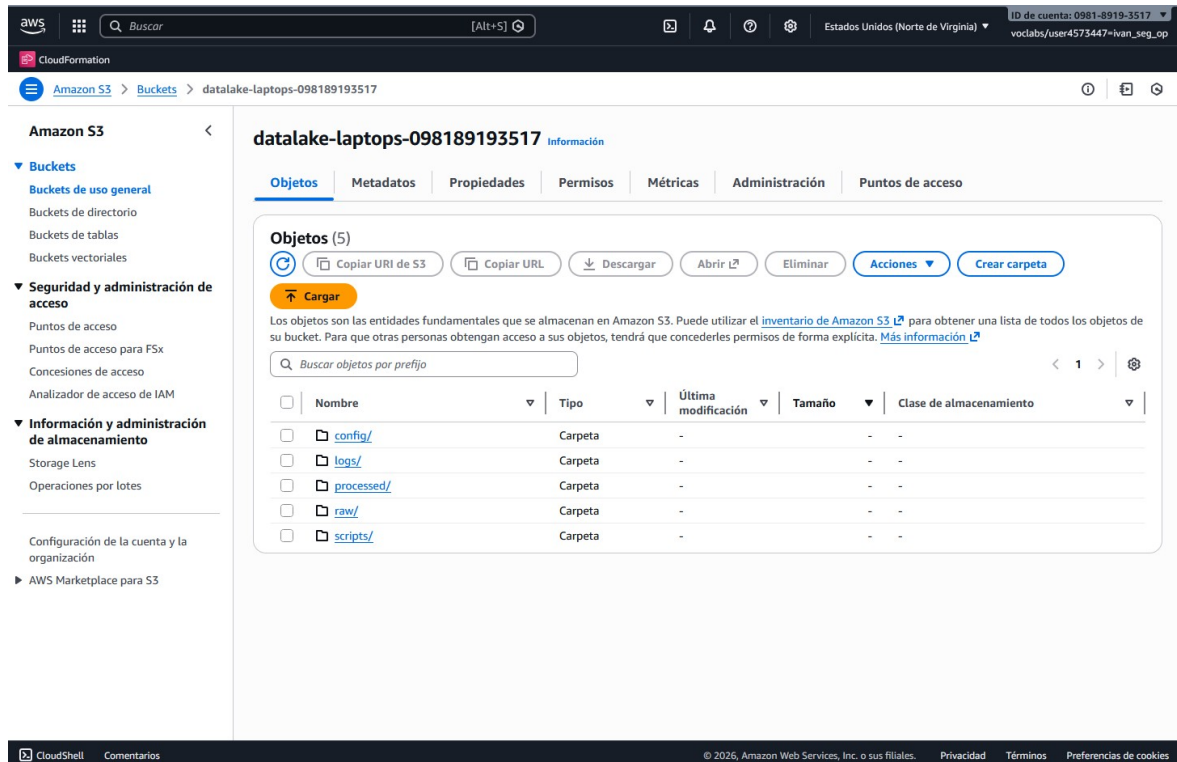


Figura 3: Creación del Bucket S3 y estructura de carpetas.

Paso 2: Ingesta de Datos

Monitorización del stream de Kinesis recibiendo los registros enviados por el productor Python.

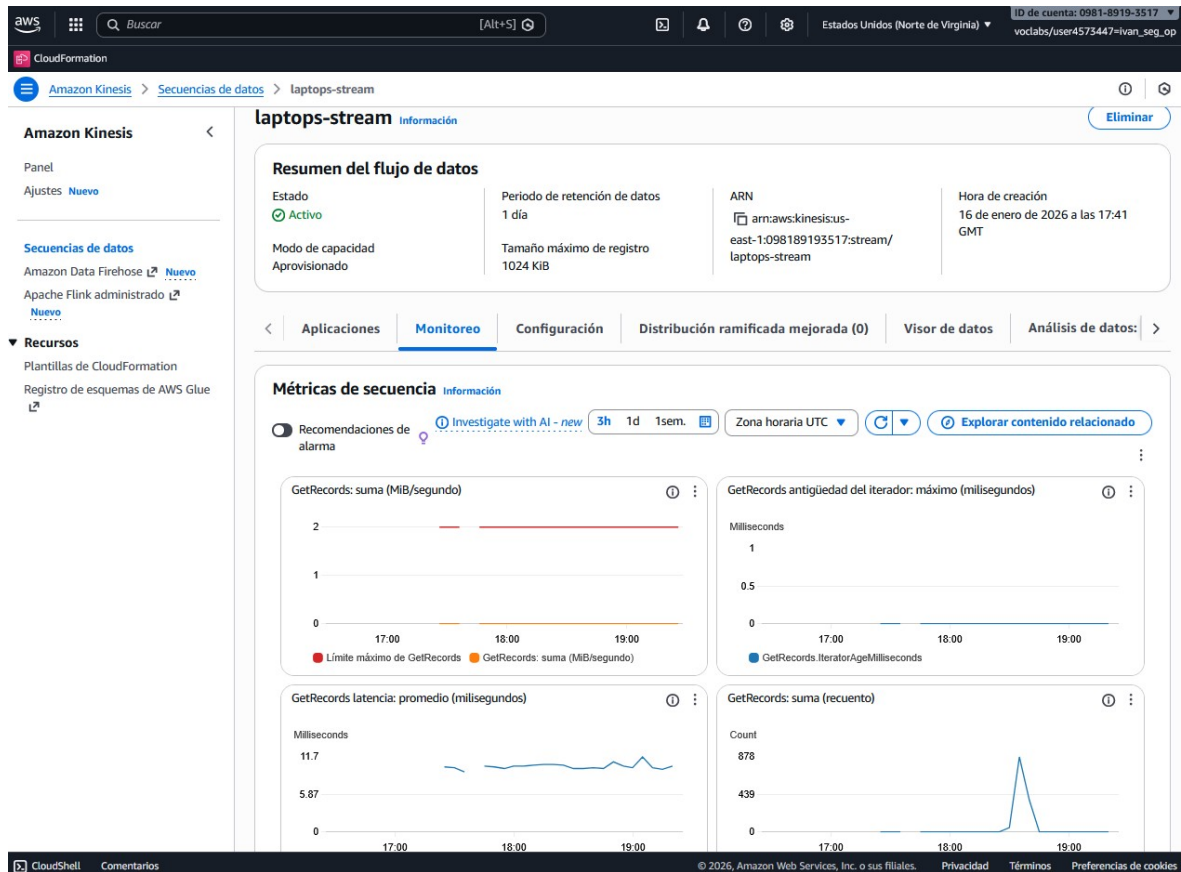


Figura 4: Métricas de ingesta en Kinesis Data Streams.

Paso 3: Catalogación Raw

Verificación de que los datos han aterrizado en la zona raw/ de S3 tras pasar por Firehose y antes de ser catalogados.

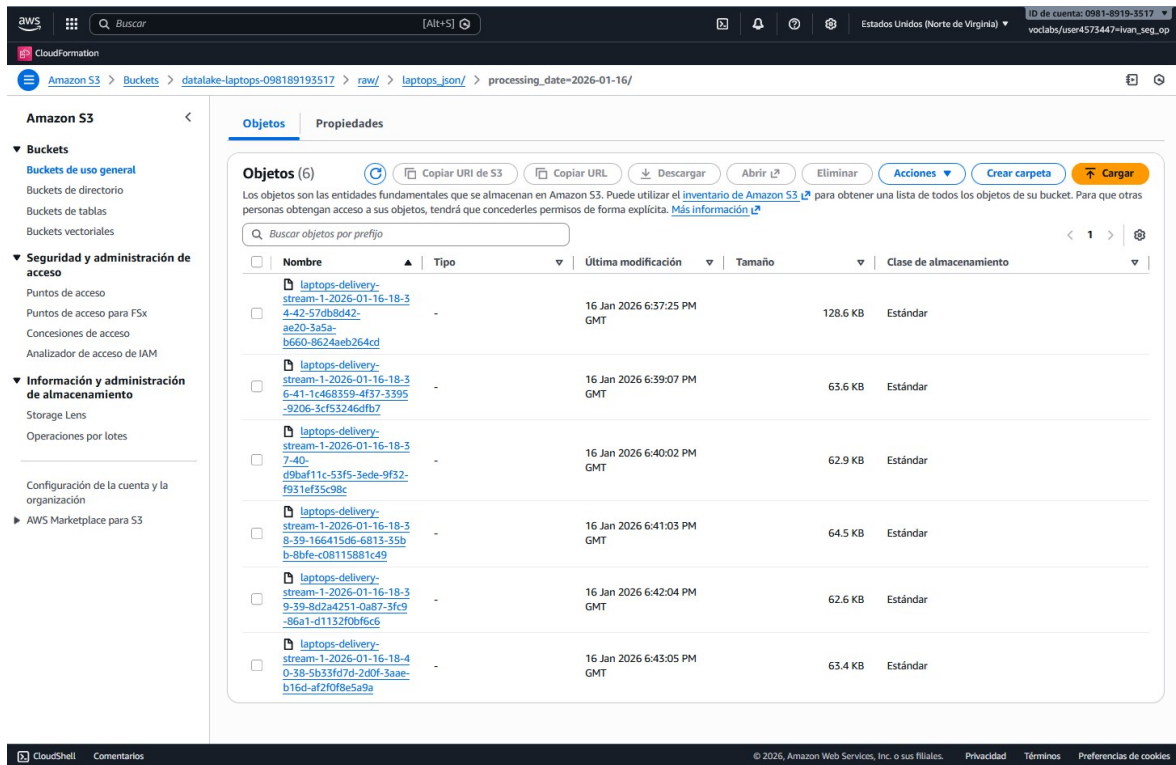


Figura 5: Datos JSON aterrizados en S3 (Zona Raw).

Paso 4: Procesamiento ETL

Panel de AWS Glue mostrando la ejecución exitosa (*Succeeded*) de los Jobs de Spark para Marcas y Sistemas Operativos.

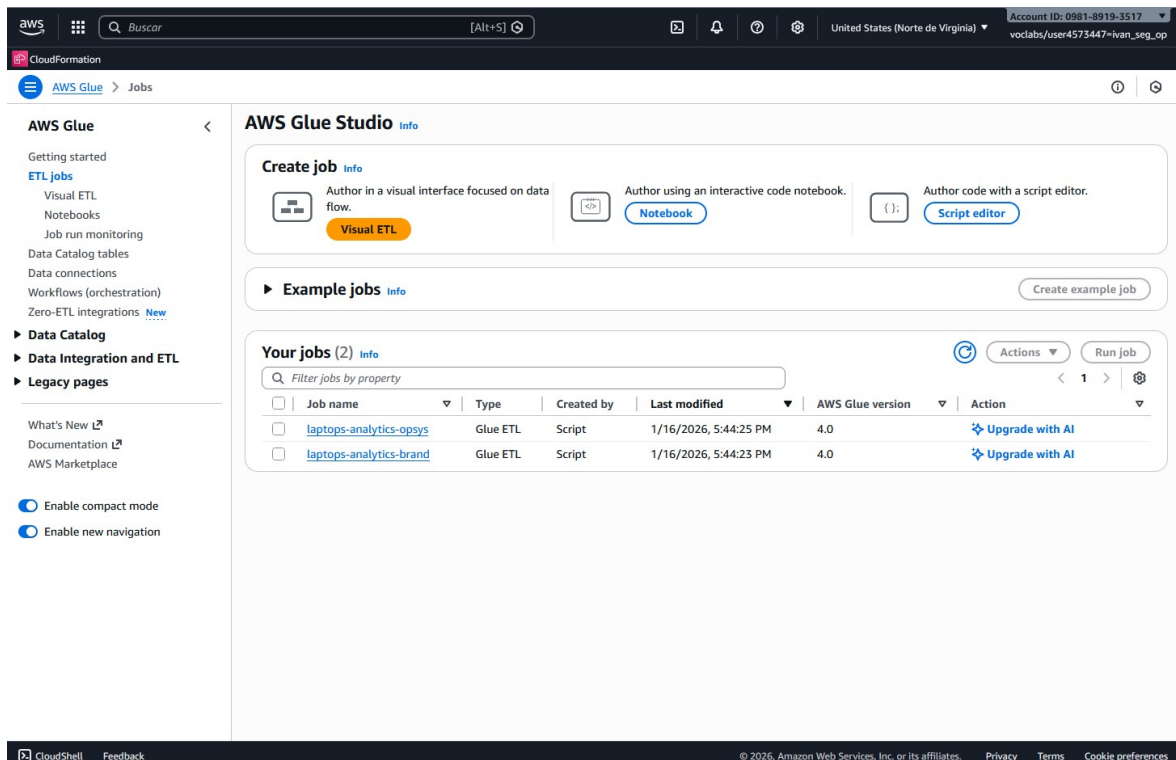


Figura 6: Estado de ejecución de los Glue Jobs.

Paso 5: Resultados Procesados

Tras la ejecución del crawler de resultados, se generan las carpetas particionadas en formato Parquet en la zona processed/.

The screenshot shows the Amazon S3 console interface. The top navigation bar includes the AWS logo, a search bar, and the account ID. The left sidebar shows the 'Amazon S3' menu with options like 'Buckets', 'Seguridad y administración de acceso', and 'Información y administración de almacenamiento'. The main content area displays the 'laptops_by_brand/' bucket. The 'Objetos' tab is selected, showing a list of 4 objects. Each object is a Parquet file with a name starting with 'part-00000-22b3894b-0f2' and a size of 1.7 KB or 1.6 KB. The 'laptops_by_opsys/' bucket is also visible below.

Nombre	Tipo	Última modificación	Tamaño	Clase de almacenamiento
part-00000-22b3894b-0f2-a-43ef-a37b-afd21bcfdb02-c000.snappy.parquet	parquet	16 Jan 2026 6:46:12 PM GMT	1.7 KB	Estándar
part-00001-22b3894b-0f2-a-43ef-a37b-afd21bcfdb02-c000.snappy.parquet	parquet	16 Jan 2026 6:46:12 PM GMT	1.7 KB	Estándar
part-00002-22b3894b-0f2-a-43ef-a37b-afd21bcfdb02-c000.snappy.parquet	parquet	16 Jan 2026 6:46:12 PM GMT	1.7 KB	Estándar
part-00003-22b3894b-0f2-a-43ef-a37b-afd21bcfdb02-c000.snappy.parquet	parquet	16 Jan 2026 6:46:12 PM GMT	1.6 KB	Estándar

Nombre	Tipo	Última modificación	Tamaño	Clase de almacenamiento
part-00000-98bbdda-bd61-4921-acd9-66f4c9ad42ee-c000.snappy.parquet	parquet	16 Jan 2026 6:46:03 PM GMT	1.6 KB	Estándar
part-00001-98bbdda-bd61-4921-acd9-66f4c9ad42ee-c000.snappy.parquet	parquet	16 Jan 2026 6:46:03 PM GMT	1.6 KB	Estándar
part-00002-98bbdda-bd61-4921-acd9-66f4c9ad42ee-c000.snappy.parquet	parquet	16 Jan 2026 6:46:03 PM GMT	1.6 KB	Estándar
part-00003-98bbdda-bd61-4921-acd9-66f4c9ad42ee-c000.snappy.parquet	parquet	16 Jan 2026 6:46:03 PM GMT	1.6 KB	Estándar

Figura 7: Estructura de salida en S3 para Marcas (up) y SO (down).

Paso 6: Consultas SQL (Athena)

Validación de los datos mediante consultas SQL estándar sobre las tablas catalogadas.

Consulta 1: Consulta al coste promedio a partir del SO.

The screenshot shows the Amazon Athena console interface. On the left, the 'Datos' sidebar is visible with sections for 'Origen de datos' (set to 'AwsDataCatalog'), 'Catálogo' (set to 'Ningún elemento'), 'Base de datos' (set to 'laptops_db'), and 'Tablas y vistas' (listing 'laptops_by_brand', 'laptops_by_opsys', and 'laptops_json'). The main area displays 'Consulta 1' with the following SQL query:

```
1 SELECT opsys, cantidad_dispositivos,
2     ROUND(coste_promedio, 2) as coste_medio
3 FROM "laptops_db"."laptops_by_opsys"
4 ORDER BY cantidad_dispositivos DESC;
```

Below the query, the 'Ejecutar de nuevo' button is highlighted. The 'Resultados de la consulta' section shows the query is 'Completado' with a 'Tiempo en cola' of 109 ms, 'Tiempo de ejecución' of 526 ms, and 'Datos analizados' of 0.60 KB. The results are displayed in a table with 9 rows:

#	opsys	cantidad_dispositivos	coste_medio
1	Windows 10	1072	1168.14
2	No OS	66	587.97
3	Linux	62	617.07
4	Windows 7	45	1686.65
5	Chrome OS	27	553.59
6	macOS	13	1749.63
7	Mac OS X	8	1262.87
8	Windows 10 5	8	1286.48
9	Android	2	434.0

Figura 8: Consulta SQL de verificación inicial.

Consulta 2: Análisis de precios por Marca.

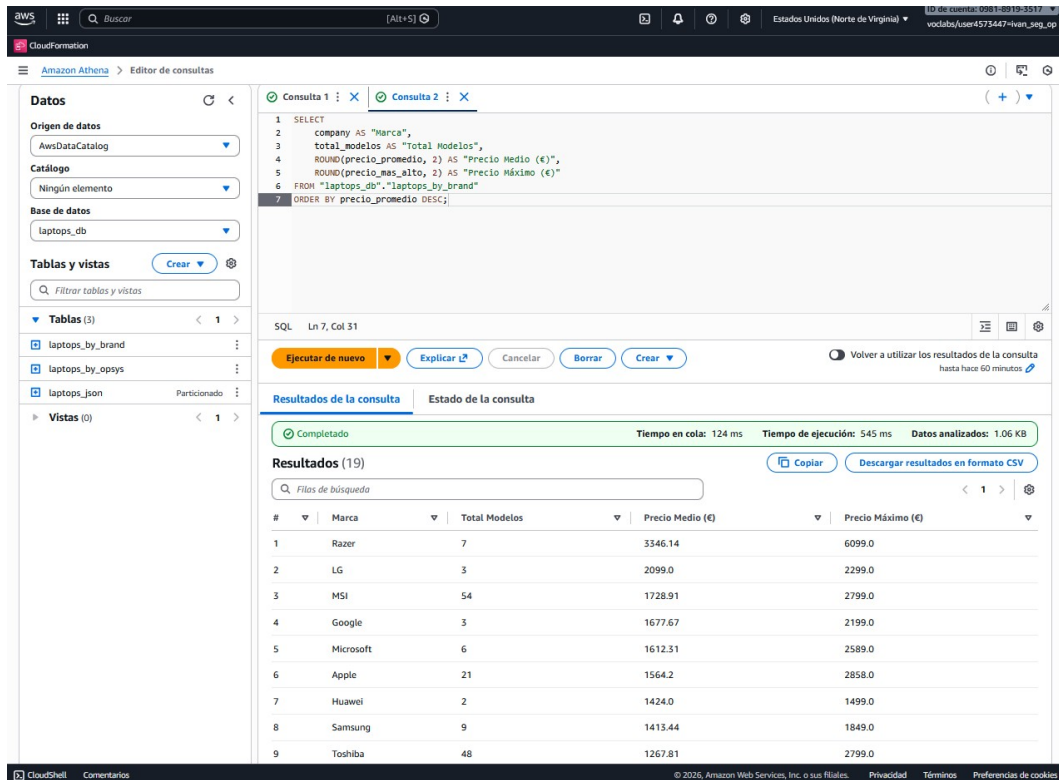


Figura 9: Ranking de marcas por precio promedio.

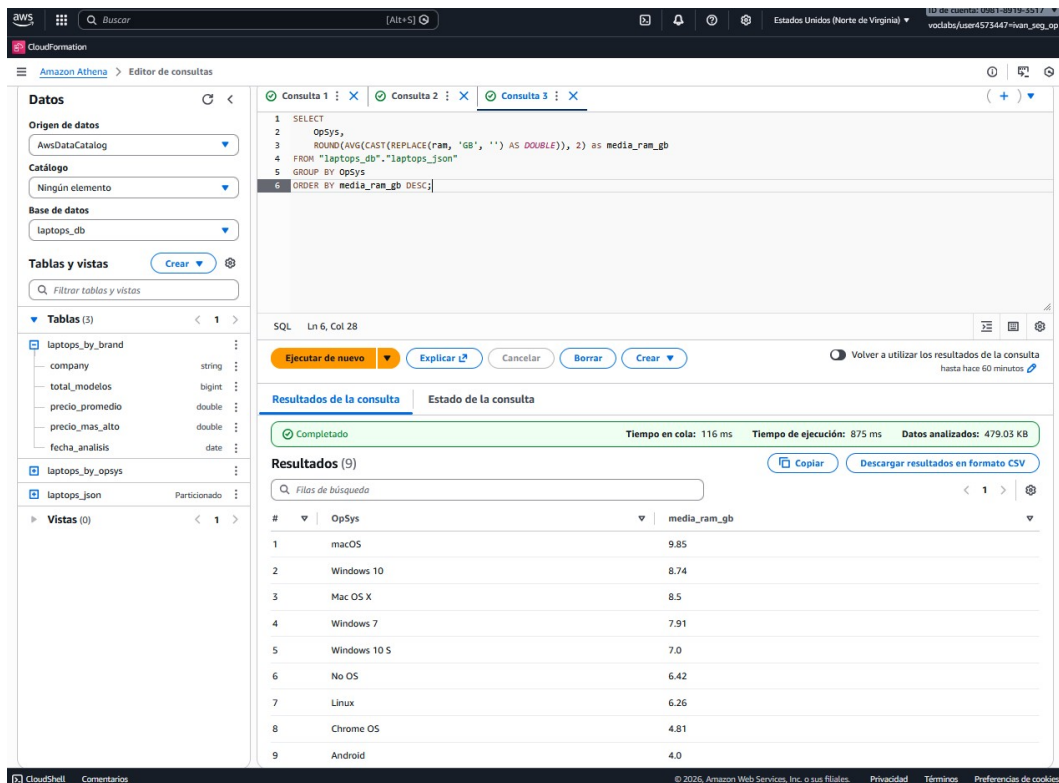
Consulta 3: Análisis de la cantidad media de RAM.

Figura 10: Agregación de dispositivos por Sistema Operativo.

7.2 Anexo B. Script de despliegue

Este script automatiza la creación de la infraestructura (Buckets, Roles, Streams, Lambdas y Jobs).

```

1 # =====
2 # SCRIPT DE DESPLIEGUE AWS - POWERSHELL
3 # =====
4
5 # --- 1. CONFIGURACION ---
6 $env:AWS_REGION = "us-east-1"
7
8 # Asegurar cuenta de LabRole
9 $CustomRoleName = "LabRole"
10
11 Write-Host "Iniciando validacion de credenciales..." -ForegroundColor
    Cyan
12
13 # 1.1 Validar Credenciales AWS CLI
14 try {
15     $identity = aws sts get-caller-identity --output json |
        ConvertFrom-Json
16     $env:ACCOUNT_ID = $identity.Account
17     Write-Host "Credenciales detectadas. Account ID: $($env:ACCOUNT_ID
        )" -ForegroundColor Green
18 } catch {
19     Write-Error "ERROR FATAL: No se detectan credenciales de AWS.
        Ejecuta 'aws configure' o refresca tu sesion."
20     exit 1
21 }
22
23 $env:BUCKET_NAME = "datalake-laptops-$($env:ACCOUNT_ID)"
24
25 # 1.2 Obtener ARN del Rol
26 try {
27     $env:ROLE_ARN = (aws iam get-role --role-name $CustomRoleName --
        query 'Role.Arn' --output text).Trim()
28     Write-Host "Rol encontrado: $($env:ROLE_ARN)" -ForegroundColor
        Green
29 } catch {
30     Write-Error "ERROR CRITICO: No se encuentra el rol '
        $CustomRoleName'. Si estas en una cuenta personal, crea un rol
        con permisos de Admin o cambia la variable '$CustomRoleName'
        en el script."
31     exit 1
32 }
33
34 Write-Host "-----" -

```



```

    ForegroundColor Cyan
35 Write-Host "RESUMEN_DE_DESPLIEGUE:"
36 Write-Host "Bucket: $($env:BUCKET_NAME)"
37 Write-Host "Region: $($env:AWS_REGION)"
38 Write-Host "-----" -
    ForegroundColor Cyan
39
40 # --- 2. S3 & KINESIS ---
41 Write-Host "Build: Creando Bucket S3..." -ForegroundColor Yellow
42 try {
43     aws s3 mb s3://$env:BUCKET_NAME 2>$null
44 } catch {
45     Write-Host "INFO: El bucket ya existe o no se pudo crear (
        verificar permisos)." -ForegroundColor Gray
46 }
47
48 Write-Host "Build: Creando estructura de carpetas..." -ForegroundColor
    Yellow
49 # Redirigir stderr a null para limpiar la salida si ya existen
50 aws s3api put-object --bucket $env:BUCKET_NAME --key raw/ 2>$null
51 aws s3api put-object --bucket $env:BUCKET_NAME --key raw/laptops_json/
    2>$null
52 aws s3api put-object --bucket $env:BUCKET_NAME --key processed/ 2>
    $null
53 aws s3api put-object --bucket $env:BUCKET_NAME --key config/ 2>$null
54 aws s3api put-object --bucket $env:BUCKET_NAME --key scripts/ 2>$null
55 aws s3api put-object --bucket $env:BUCKET_NAME --key logs/ 2>$null
56
57 Write-Host "Build: Creando Kinesis Stream..." -ForegroundColor Yellow
58 try {
59     aws kinesis create-stream --stream-name laptops-stream --shard-
        count 1 2>$null
60 } catch {
61     Write-Host "INFO: El stream probablemente ya existe." -
        ForegroundColor Gray
62 }
63
64 # --- 3. LAMBDA ---
65 Write-Host "Build: Empaquetando y desplegando Lambda..." -
    ForegroundColor Yellow
66
67 if (Test-Path "firehose.zip") { Remove-Item "firehose.zip" }
68 # Verificar que exista el archivo python
69 if (-not (Test-Path "firehose.py")) {
70     Write-Error "Falta el archivo 'firehose.py' en el directorio
        actual."
71     exit 1

```

```

72 }
73 Compress-Archive -Path "firehose.py" -DestinationPath "firehose.zip"
74
75 # Intentar crear.
76 aws lambda create-function '
77     --function-name laptops-firehose-lambda '
78     --runtime python3.12 '
79     --role $env:ROLE_ARN '
80     --handler firehose.lambda_handler '
81     --zip-file fileb://firehose.zip '
82     --timeout 60 '
83     --memory-size 128 2>$null
84
85 if (-not $?) {
86     Write-Host "La Lambda ya existe, actualizando código..." -
87         ForegroundColor DarkGray
88     aws lambda update-function-code --function-name laptops-firehose-
89         lambda --zip-file fileb://firehose.zip >$null
90 }
91
92 # Esperar propagacion
93 Start-Sleep -Seconds 5
94
95 $env:LAMBDA_ARN = (aws lambda get-function --function-name laptops-
96     firehose-lambda --query 'Configuration.FunctionArn' --output text)
97     .Trim()
98
99 # --- 4. FIREHOSE ---
100 Write-Host "Build: Creando Firehose Delivery Stream..." -
101     ForegroundColor Yellow
102
103 # Objeto de configuracion para JSON
104 $firehoseConfig = @{
105     BucketARN = "arn:aws:s3:::$env:BUCKET_NAME"
106     RoleARN = "$env:ROLE_ARN"
107     Prefix = "raw/laptops_json/processing_date=!{
108         partitionKeyFromLambda:processing_date}/"
109     ErrorOutputPrefix = "errors/{firehose:error-output-type}/"
110     BufferingHints = @{ SizeInMBs = 64; IntervalInSeconds = 60 }
111     DynamicPartitioningConfiguration = @{ Enabled = $true;
112         RetryOptions = @{ DurationInSeconds = 300 } }
113     ProcessingConfiguration = @{
114         Enabled = $true
115         Processors = @(
116             @{
117                 Type = "Lambda"
118                 Parameters = @(

```

```

112         @{ ParameterName = "LambdaArn"; ParameterValue = "
113             $env:LAMBDA_ARN" },
114         @{ ParameterName = "BufferSizeInMBs";
115             ParameterValue = "1" },
116         @{ ParameterName = "BufferIntervalInSeconds";
117             ParameterValue = "60" }
118     )
119 }
120
121 $firehoseConfig | ConvertTo-Json -Depth 10 | Out-File "firehose_config
122     .json" -Encoding ASCII
123
124 aws firehose create-delivery-stream '
125     --delivery-stream-name laptops-delivery-stream '
126     --delivery-stream-type KinesisStreamAsSource '
127     --kinesis-stream-source-configuration "KinesisStreamARN=arn:aws:
128         kinesis:$( $env:AWS_REGION ):$( $env:ACCOUNT_ID ):stream/laptops-
129         stream,RoleARN=$env:ROLE_ARN" '
130     --extended-s3-destination-configuration file://firehose_config.
131         json 2>$null
132
133 Remove-Item "firehose_config.json"
134
135 # --- 5. GLUE DATABASE & CRAWLER ---
136 Write-Host "Build: Configurando Glue..." -ForegroundColor Yellow
137
138 Set-Content -Path "glue_db.json" -Value '{"Name":"laptops_db"}'
139 aws glue create-database --database-input file://glue_db.json 2>$null
140 Remove-Item "glue_db.json"
141
142 $crawlerConfig = @{
143     Name = "laptops-raw-crawler"
144     Role = "$env:ROLE_ARN"
145     DatabaseName = "laptops_db"
146     Targets = @{ S3Targets = @( @{ Path = "s3://$env:BUCKET_NAME/raw/
147         laptops_json" } ) }
148 }
149
150 $crawlerConfig | ConvertTo-Json -Depth 5 | Out-File "glue_crawler.json
151     " -Encoding ASCII
152
153 aws glue create-crawler --cli-input-json file://glue_crawler.json 2>
154     $null
155 Remove-Item "glue_crawler.json"
156

```

```

149 # Iniciar crawler si no esta corriendo
150 try {
151     aws glue start-crawler --name laptops-raw-crawler 2>$null
152 } catch {}
153
154 # --- 6. GLUE JOBS ---
155 Write-Host "Build: Subiendo scripts y creando Glue Jobs..." -
    ForegroundColor Yellow
156
157 # Subida de scripts (Validando que existan)
158 if (-not (Test-Path "laptops_analytics_brand.py")) { Write-Error "
    Falta laptops_analytics_brand.py"; exit 1 }
159 if (-not (Test-Path "laptops_analytics_so.py")) { Write-Error "Falta
    laptops_analytics_so.py"; exit 1 }
160
161 aws s3 cp laptops_analytics_brand.py s3://$env:BUCKET_NAME/scripts/
162 aws s3 cp laptops_analytics_so.py s3://$env:BUCKET_NAME/scripts/
163
164 # --- JOB 1: MARCAS ---
165 $jobBrandConfig = @{
166     Name = "laptops-analytics-brand"
167     Role = "$env:ROLE_ARN"
168     Command = @{
169         Name = "glueetl"
170         ScriptLocation = "s3://$env:BUCKET_NAME/scripts/
            laptops_analytics_brand.py"
171         PythonVersion = "3"
172     }
173     DefaultArguments = @{
174         "--database" = "laptops_db"
175         "--table" = "laptops_json"
176         "--output_path" = "s3://$env:BUCKET_NAME/processed/
            laptops_by_brand/"
177         "--enable-continuous-cloudwatch-log" = "true"
178         "--spark-event-logs-path" = "s3://$env:BUCKET_NAME/logs/"
179     }
180     GlueVersion = "4.0"
181     NumberOfWorkers = 2
182     WorkerType = "G.1X"
183 }
184 $jobBrandConfig | ConvertTo-Json -Depth 5 | Out-File "job_brand.json"
    -Encoding ASCII
185
186 aws glue create-job --cli-input-json file://job_brand.json 2>$null
187 if (-not $?) { aws glue update-job --job-name laptops-analytics-brand
    --job-update file://job_brand.json >$null }
188 Remove-Item "job_brand.json"

```

```

189
190 # --- JOB 2: OPSYS ---
191 $jobOsConfig = @{
192     Name = "laptops-analytics-opsys"
193     Role = "$env:ROLE_ARN"
194     Command = @{
195         Name = "glueetl"
196         ScriptLocation = "s3://$env:BUCKET_NAME/scripts/
197             laptops_analytics_so.py"
198         PythonVersion = "3"
199     }
200     DefaultArguments = @{
201         "--database" = "laptops_db"
202         "--table" = "laptops_json"
203         "--output_path" = "s3://$env:BUCKET_NAME/processed/
204             laptops_by_opsys/"
205         "--enable-continuous-cloudwatch-log" = "true"
206         "--spark-event-logs-path" = "s3://$env:BUCKET_NAME/logs/"
207     }
208     GlueVersion = "4.0"
209     NumberOfWorkers = 2
210     WorkerType = "G.1X"
211 }
212 $jobOsConfig | ConvertTo-Json -Depth 5 | Out-File "job_os.json" -
213     Encoding ASCII
214
215
216 aws glue create-job --cli-input-json file://job_os.json 2>$null
217 if (-not $?) { aws glue update-job --job-name laptops-analytics-opsys
218     --job-update file://job_os.json >$null }
219 Remove-Item "job_os.json"
220
221 # --- 7. FINALIZACION ---
222
223 Write-Host "Infraestructura desplegada correctamente" -ForegroundColor
224     Green
225
226 Write-Host "-----"
227 Write-Host "Pasos siguientes:"
228 Write-Host "1. Ejecutar 'uv run kinesis.py' para enviar datos."
229 Write-Host "2. Espera 2-5 minutos."
230 Write-Host "3. Comprobar dentro de AWS Glue."
231 Write-Host "4. Ejecutar los Jobs 'laptops-analytics-brand'y'laptops -
232     analytics-opsys'."

```

Listing 6: Script deploy.ps1: Automatización de infraestructura AWS

7.3 Anexo C. Productor Kinesis

Script desarrollado en Python encargado de la simulación y envío de datos transaccionales al servicio Kinesis Data Streams.

```

1 import boto3
2 import json
3 import time
4 from loguru import logger
5
6 # CONFIGURACION
7 STREAM_NAME = 'laptops-stream'
8 REGION = 'us-east-1'
9 INPUT_FILE = 'datos.json'
10
11 kinesis = boto3.client('kinesis', region_name=REGION)
12
13 def load_data(file_path):
14     with open(file_path, 'r', encoding='utf-8') as f:
15         return json.load(f)
16
17 def run_producer():
18     laptops = load_data(INPUT_FILE)
19     records_sent = 0
20
21     logger.info(f"Iniciando transmisión de {len(laptops)} laptops al stream: {STREAM_NAME}...")
22
23     for laptop in laptops:
24         # Extraer campos clave para el log
25         laptop_id = laptop.get('laptop_ID')
26         company = laptop.get('Company')
27         price = laptop.get('Price_euros')
28
29         # Enviar a Kinesis
30         try:
31             response = kinesis.put_record(
32                 StreamName=STREAM_NAME,
33                 Data=json.dumps(laptop), # Enviar el objeto laptop entero
34                 PartitionKey=str(company) # Agrupar shards por marca (Apple, Dell, etc.)
35             )
36
37             records_sent += 1
38             logger.info(f"Enviado ID:{laptop_id} ({company}) - {price} euros Shard: {response['ShardId'][-5:]}")
39             # Pequeña pausa para ver el efecto streaming

```

```

40         time.sleep(0.2)
41
42     except Exception as e:
43         logger.error(f"Error enviando laptop {laptop_id}: {e}")
44
45     logger.info(f"Fin de la transmisión. Total registros enviados: {
46         records_sent}")
47
48 if __name__ == '__main__':
49     run_producer()

```

Listing 7: Script kinesis.py: Productor de datos a Kinesis

7.4 Anexo D. Transformación Firehose

Función Lambda encargada de la transformación y particionado temporal de los datos antes de su persistencia en S3.

```

1  import json
2  import base64
3  import datetime
4
5  def lambda_handler(event, context):
6      output = []
7
8      for record in event['records']:
9          try:
10             # 1. Decodificar entrada de Kinesis
11             payload = base64.b64decode(record['data']).decode('utf-8')
12             data_json = json.loads(payload)
13
14             # 2. Anadir metadatos de procesamiento (Timestamp)
15             processing_time = datetime.datetime.now(datetime.timezone.
16                 utc)
17             data_json['processed_at'] = processing_time.isoformat()
18
19             # 3. Preparar Partition Key para S3 (YYYY-MM-DD)
20             partition_date = processing_time.strftime('%Y-%m-%d')
21
22             # 4. Re-codificar para Firehose (Debe terminar en salto de
23                 linea para JSON Lines)
24             output_payload = json.dumps(data_json) + '\n'
25
26             output_record = {
27                 'recordId': record['recordId'],
28                 'result': 'Ok',

```

```

27         'data': base64.b64encode(output_payload.encode('utf
28             -8')).decode('utf-8'),
29         'metadata': {
30             'partitionKeys': {
31                 'processing_date': partition_date
32             }
33         }
34         output.append(output_record)
35
36     except Exception as e:
37         # Si falla un registro, se marca como failed pero no se
38         # detiene todo el lote
39         print(f"Error processing record: {e}")
40         output.append({
41             'recordId': record['recordId'],
42             'result': 'ProcessingFailed',
43             'data': record['data']
44         })
45     return {'records': output}

```

Listing 8: Script firehose.py: Lógica de transformación Lambda

7.5 Anexo E. Análisis por Marca (Spark)

Job de Spark ETL para la agregación de métricas de negocio basadas en el fabricante.

```

1  import sys
2  import logging
3  from pyspark.context import SparkContext
4  from awsglue.context import GlueContext
5  from awsglue.utils import getResolvedOptions
6  from pyspark.sql.functions import col, avg, count, max as spark_max,
7  current_date
8  from awsglue.dynamicframe import DynamicFrame
9
10 # Configuración de Logging
11 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(
12     levelname)s - %(message)s')
13 logger = logging.getLogger(__name__)
14
15 def main():
16     args = getResolvedOptions(sys.argv, ['database', 'table', '
17         output_path'])
18     database = args['database']
19     table = args['table']

```



```

17     output_path = args['output_path']
18
19     logger.info(f"Iniciando Analytics por Marca. DB: {database}, Table
20                 : {table}")
21
22     sc = SparkContext()
23     glueContext = GlueContext(sc)
24     spark = glueContext.spark_session
25
26     # 1. Leer datos
27     dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
28         database=database,
29         table_name=table
30     )
31
32     # --- CORRECCION DE TIPO DE DATO (CHOICE) ---
33     # Esto fuerza a que Price_euros sea tratado siempre como double,
34     # resolviendo la ambigüedad entre int y double.
35     try:
36         dynamic_frame = dynamic_frame.resolveChoice(specs = [(
37             'Price_euros', 'cast:double')])
38     except:
39         logger.warning("No se requirio resolveChoice o fallo,
40                         continuando...")
41
42     df = dynamic_frame.toDF()
43
44     if df.count() == 0:
45         logger.warning("No se encontraron datos de laptops.
46                         Finalizando.")
47         return
48
49     # Asegurar cast final
50     df = df.withColumn("Price_euros", col("Price_euros").cast("double"
51     ))
52
53     # 2. LOGICA: Agregacion por MARCA
54     agg_df = df.groupBy("Company") \
55         .agg(
56             count("laptop_ID").alias("total_modelos"),
57             avg("Price_euros").alias("precio_promedio"),
58             spark_max("Price_euros").alias("precio_mas_alto")
59         ) \
60         .withColumn("fecha_analisis", current_date()) \
61         .orderBy("precio_promedio", ascending=False)
62
63     # 3. Escribir resultados

```

```

59     output_dynamic_frame = DynamicFrame.fromDF(agg_df, glueContext, "
        output")
60
61     glueContext.write_dynamic_frame.from_options(
62         frame=output_dynamic_frame,
63         connection_type="s3",
64         connection_options={ "path": output_path },
65         format="parquet",
66         format_options={"compression": "snappy"}
67     )
68
69     logger.info(f"Job completado.")
70
71 if __name__ == "__main__":
72     main()

```

Listing 9: Script laptops_analytics_brand.py: ETL de marcas

7.6 Anexo F. Análisis por Sistema Operativo (Spark)

Job de Spark ETL para la agregación de métricas basadas en el Sistema Operativo.

```

1  import sys
2  import logging
3  from pyspark.context import SparkContext
4  from awsglue.context import GlueContext
5  from awsglue.utils import getResolvedOptions
6  from pyspark.sql.functions import col, avg, count, min as spark_min,
    max as spark_max
7  from awsglue.dynamicframe import DynamicFrame
8
9  logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(
    levelname)s - %(message)s')
10 logger = logging.getLogger(__name__)
11
12 def main():
13     args = getResolvedOptions(sys.argv, ['database', 'table', '
        output_path'])
14     database = args['database']
15     table = args['table']
16     output_path = args['output_path']
17
18     logger.info(f"Iniciando Analytics por OS. DB: {database}, Table: {
        table}")
19
20     sc = SparkContext()
21     glueContext = GlueContext(sc)

```

```

22     spark = glueContext.spark_session
23
24     dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
25         database=database,
26         table_name=table
27     )
28
29     # --- CORRECCION DE TIPO DE DATO ---
30     try:
31         dynamic_frame = dynamic_frame.resolveChoice(specs = [('
32             Price_euros', 'cast:double')])
33     except:
34         pass
35
36     df = dynamic_frame.toDF()
37     if df.count() == 0:
38         logger.warning("Sin_datos.")
39         return
40
41     df = df.withColumn("Price_euros", col("Price_euros").cast("double"
42         ))
43
44     # 2. LOGICA: Agregacion por OS
45     agg_df = df.groupBy("OpSys") \
46         .agg(
47             count("laptop_ID").alias("cantidad_dispositivos"),
48             avg("Price_euros").alias("coste_promedio"),
49             spark_min("Price_euros").alias("coste_minimo"),
50             spark_max("Price_euros").alias("coste_maximo")
51         ) \
52         .orderBy("coste_promedio", ascending=False)
53
54     output_dynamic_frame = DynamicFrame.fromDF(agg_df, glueContext, "
55         output")
56
57     glueContext.write_dynamic_frame.from_options(
58         frame=output_dynamic_frame,
59         connection_type="s3",
60         connection_options={ "path": output_path },
61         format="parquet",
62         format_options={"compression": "snappy"}
63     )
64
65     logger.info(f"Job_completado.")
66 if __name__ == "__main__":
67     main()

```

Listing 10: Script laptops_analytics_so.py: ETL de Sistemas Operativos

7.7 Anexo G. Crawler de Resultados

Script para catalogar los resultados finales procesados en formato Parquet.

```

1 # 1. Asegurar variables de entorno
2 if (-not $env:BUCKET_NAME) { $env:BUCKET_NAME = "datalake-laptops
   -098189193517" }
3 if (-not $env:ROLE_ARN) { $env:ROLE_ARN = "arn:aws:iam::098189193517:
   role/LabRole" }
4
5 # 2. Definir la configuracion del Crawler
6 $resultsCrawler = @{
7     Name = "laptops-processed-crawler"
8     Role = "$env:ROLE_ARN"
9     DatabaseName = "laptops_db"
10    Targets = @{ S3Targets = @(
11        @{ Path = "s3://$env:BUCKET_NAME/processed/laptops_by_brand/"
12            },
13        @{ Path = "s3://$env:BUCKET_NAME/processed/laptops_by_opsys/"
14            }
15    )}
16 }
17
18 # 3. Guardar JSON temporal y crear Crawler
19 $resultsCrawler | ConvertTo-Json -Depth 5 | Out-File "
20   glue_results_crawler.json" -Encoding ASCII
21 aws glue create-crawler --cli-input-json file://glue_results_crawler.
22   json
23
24 # 4. Arrancar el Crawler
25 Write-Host "Iniciando crawler..." -ForegroundColor Green
26 aws glue start-crawler --name laptops-processed-crawler
27
28 # 5. Limpieza
29 Remove-Item "glue_results_crawler.json"

```

Listing 11: Script aws_crawler.ps1: Definición del Crawler

7.8 Anexo H. Script de limpieza

Script crítico para la eliminación de recursos y prevención de costes tras la finalización del laboratorio.

```

1 Write-Host "INICIANDO LIMPIEZA TOTAL (V2)..." -ForegroundColor Red
2
3 # Evita fallos si se ha cerrado la terminal
4 try {

```

```

5     $env:ACCOUNT_ID = (aws sts get-caller-identity --query Account --
        output text).Trim()
6     $env:BUCKET_NAME = "datalake-laptops-${$env:ACCOUNT_ID}"
7     Write-Host "Cuenta detectada: $env:ACCOUNT_ID" -ForegroundColor
        Gray
8     Write-Host "Bucket objetivo: $env:BUCKET_NAME" -ForegroundColor
        Gray
9 } catch {
10     Write-Error "No se detectan credenciales AWS. Ejecuta 'aws
        configure' primero."
11     exit 1
12 }
13
14 # --- 1. S3 (ELIMINACION FORZADA) ---
15 Write-Host "1. Eliminando Bucket S3..."
16 # El 2>$null oculta errores si el bucket ya no existe
17 aws s3 rb s3://$env:BUCKET_NAME --force 2>$null
18
19 # --- 2. STREAMS (LO MAS CARO) ---
20 Write-Host "2. Eliminando Streams..."
21 aws firehose delete-delivery-stream --delivery-stream-name laptops-
    delivery-stream 2>$null
22 aws kinesys delete-stream --stream-name laptops-stream 2>$null
23
24 # --- 3. GLUE (CRAWLERS, JOBS, DB) ---
25 Write-Host "3. Eliminando recursos Glue..."
26 aws glue delete-crawler --name laptops-raw-crawler 2>$null
27 aws glue delete-crawler --name laptops-processed-crawler 2>$null
28 aws glue delete-job --job-name laptops-analytics-brand 2>$null
29 aws glue delete-job --job-name laptops-analytics-opsys 2>$null
30
31 # Nota: Glue no deja borrar la DB si tiene tablas dentro.
32 # Intentar borrar las tablas primero (brute force simple)
33 aws glue delete-table --database-name laptops_db --name laptops_json
    2>$null
34 aws glue delete-table --database-name laptops_db --name
    laptops_by_brand 2>$null
35 aws glue delete-table --database-name laptops_db --name
    laptops_by_opsys 2>$null
36 aws glue delete-database --name laptops_db 2>$null
37
38 # --- 4. LAMBDA ---
39 Write-Host "4. Eliminando Lambda..."
40 aws lambda delete-function --function-name laptops-firehose-lambda 2>
    $null
41
42 # --- 5. CLOUDWATCH LOGS (LIMPIEZA DE RASTROS) ---

```

```
43 Write-Host "5. Limpiando Logs residuales..."
44 aws logs delete-log-group --log-group-name "/aws/lambda/laptops-
    firehose-lambda" 2>$null
45 aws logs delete-log-group --log-group-name "/aws-glue/crawlers" 2>
    $null
46 aws logs delete-log-group --log-group-name "/aws-glue/jobs/laptops-
    analytics-brand" 2>$null
47 aws logs delete-log-group --log-group-name "/aws-glue/jobs/laptops-
    analytics-opsys" 2>$null
48
49 Write-Host "Limpieza completada - Cuenta limpia." -ForegroundColor
    Green
```

Listing 12: Script clean.ps1: Destrucción de recursos

7.9 Anexo I. Dataset de Origen

Archivo JSON que contiene el inventario de dispositivos utilizado por el productor de datos.

Ver archivo datos.json en GitHub

7.10 Anexo J. Uso de la Inteligencia Artificial Generativa

Se han utilizado principalmente 2 herramientas de IA generativa para asistir en la creación del formato del documento con el objetivo de que siga las restricciones impuestas, así como para la revisión ortográfica y el ordenamiento lógico de los contenidos de la memoria (siempre dentro del formato y estructura requerida).

- **ChatGPT-4 (OpenAI):** Se ha empleado para generar la estructura inicial del documento con el formato básico.
- **Gemini (Google):** Se ha utilizado como asistente a lo largo de la realización del proyecto, para obtener definiciones de algunas de las tecnologías explicadas en el presente documento y para la resolución de errores como los surgidos por el problema de tipos en el precio de los ordenadores del dataset.