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

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.

```
$env:BUCKET_NAME = "datalake-laptops-$($env:ACCOUNT_ID)"
aws s3 mb s3://$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.

```
response = kinesis.put_record(
    StreamName=STREAM_NAME,
    Data=json.dumps(laptop),
    PartitionKey=str(company) # Agrupar shards por marca
)
```

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.

```
# Re-codificar para Firehose (Debe terminar en salto de linea)
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`.

```
# CORRECCION DE TIPO DE DATO (CHOICE)
# Esto fuerza a que Price_euros sea tratado siempre como double
try:
    dynamic_frame = dynamic_frame.resolveChoice(specs = [('Price_euros','
        cast:double')])
except:
    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.

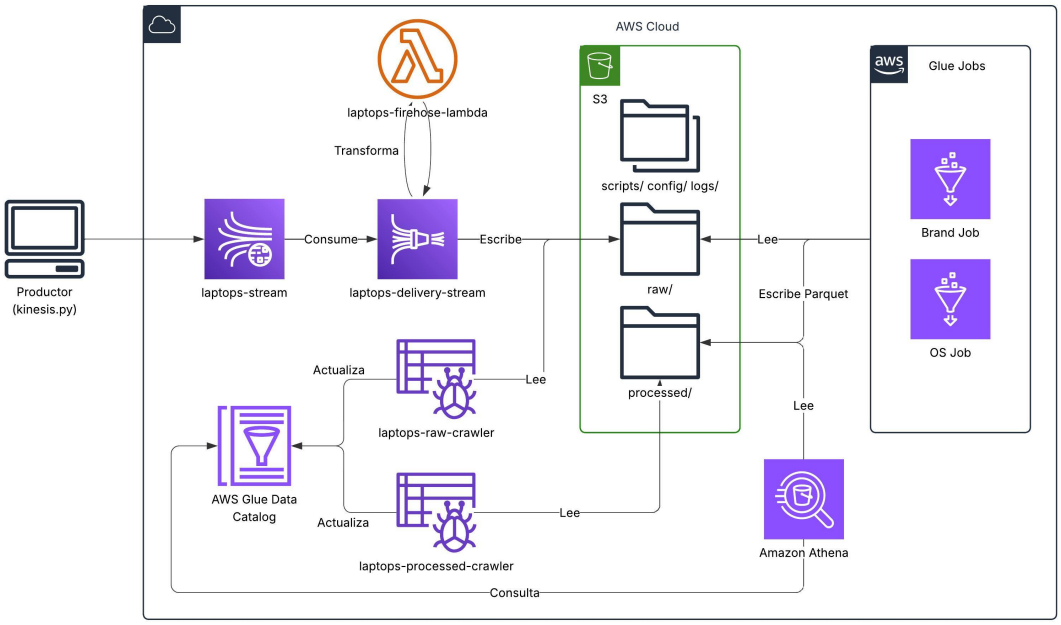

Figura 1: Diagrama de arquitectura del Data Lake Serverless en AWS.

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.

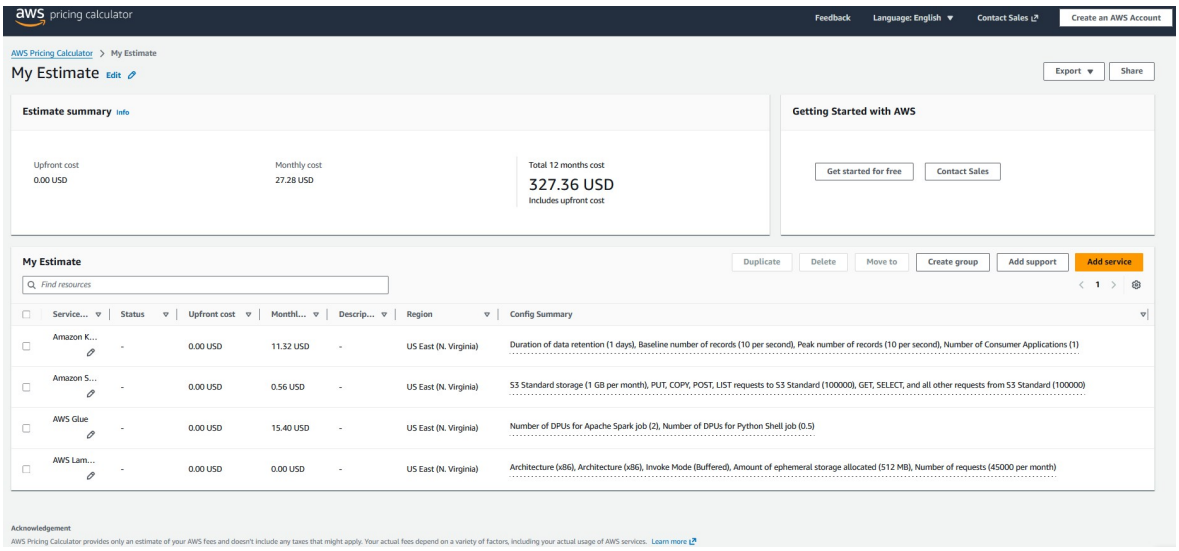

Figura 2: Captura de la calculadora oficial de AWS con el desglose de costes estimado.

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 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.

```
# EJECUTAR SOLO UNA VEZ SI NO SE TIENE EL ENVIRONMENT
# ====================================================
# uv init
# uv add boto3
# uv add loguru
# uv venv
# ====================================================

# Activar Environment
.venv\Scripts\activate

# RELLENAR ANTES DE EJECUTAR
# ====================================================
# $env:AWS_ACCESS_KEY_ID="PONER DATOS DE AWS"
# $env:AWS_SECRET_ACCESS_KEY="PONER DATOS DE AWS"
```

```
# $env:AWS_SESSION_TOKEN="PONER DATOS DE AWS"
# ====================================================

# Comprobar que AWS responde a la cuenta puesta con las variables, si devuelve un JSON con la cuenta y el Arn continuar
aws sts get-caller-identity

# PASO 1: Desplegar la infraestructura
# ====================================================
.\deploy.ps1

# PASO 2: Simular el envio de datos a Kinesis
# ====================================================
uv run kinesis.py

# PASO 3: Iniciar el Crawler de Glue para catalogar datos en bruto
# ====================================================
aws glue start-crawler --name laptops-raw-crawler
# Ejecutar para ver el estado (termina cuando READY)
aws glue get-crawler --name laptops-raw-crawler --query "Crawler.State"

# PASO 4: Ejecutar los JOBS de Glue para transformar y almacenar datos procesados
# ====================================================
aws glue start-job-run --job-name laptops-analytics-brand
aws glue start-job-run --job-name laptops-analytics-opsys
# Ejecutar para ver el estado (termina cuando READY)
aws glue get-job-runs --job-name laptops-analytics-brand --max-results 1 --query "JobRuns[0].JobRunState"
aws glue get-job-runs --job-name laptops-analytics-opsys --max-results 1 --query "JobRuns[0].JobRunState"

# PASO 5: Iniciar el Crawler de Glue para catalogar datos procesados
# ====================================================
.\aws_crawler.ps1

# PASO 6: Consultar AWS Athena y realizar varias sentencias SQL

# PASO 7: Limpiar la cuenta de recursos para evitar costes (comprobar en AWS que se han borrado correctamente)
# ====================================================
.\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.

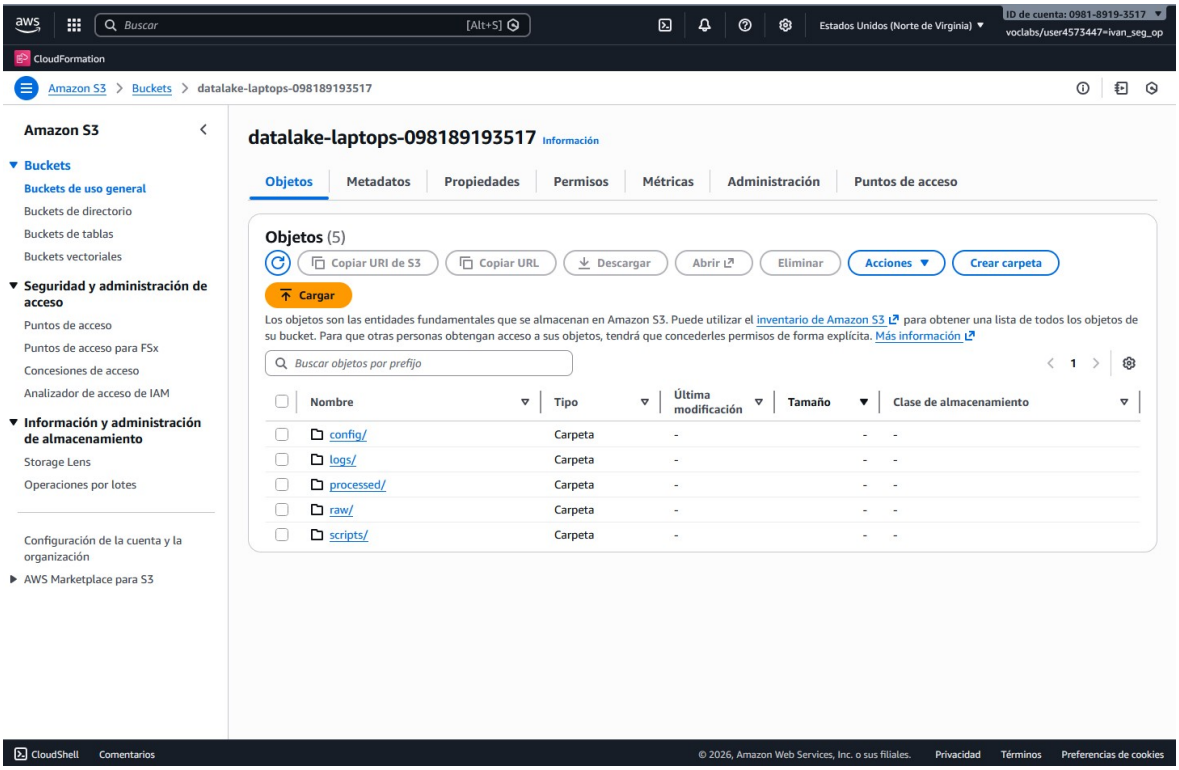

Figura 3: Creación del Bucket S3 y estructura de carpetas.

Paso 2: Ingesta de Datos

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

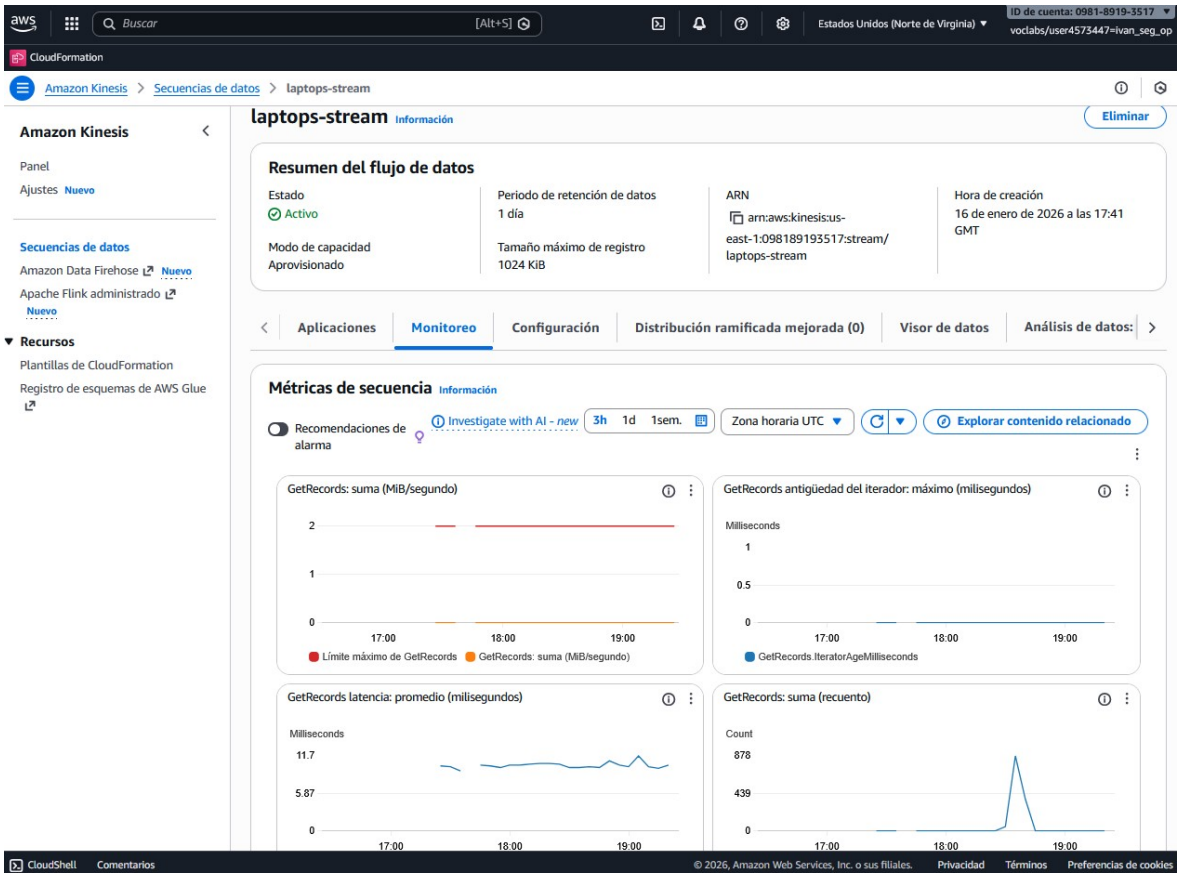

Figura 4: Métricas de ingesta en Kinesis Data Streams.

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.

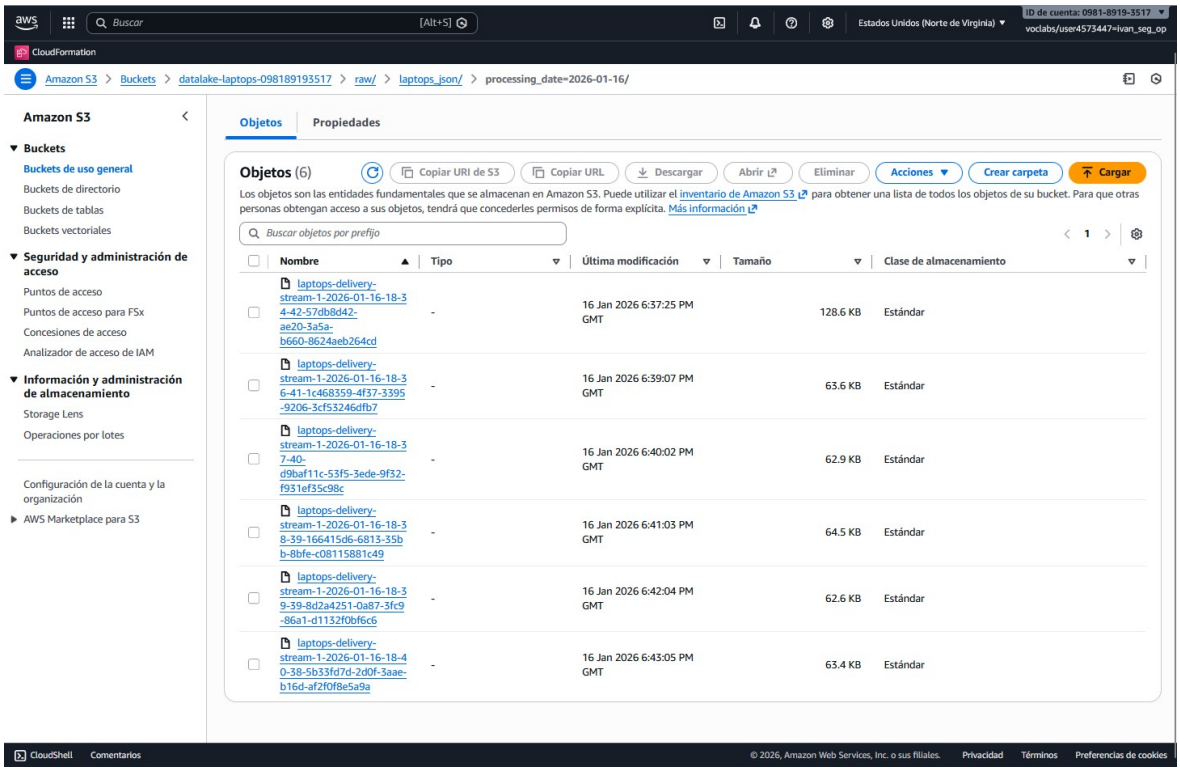

Figura 5: Datos JSON aterrizados en S3 (Zona Raw).

Paso 4: Procesamiento ETL

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

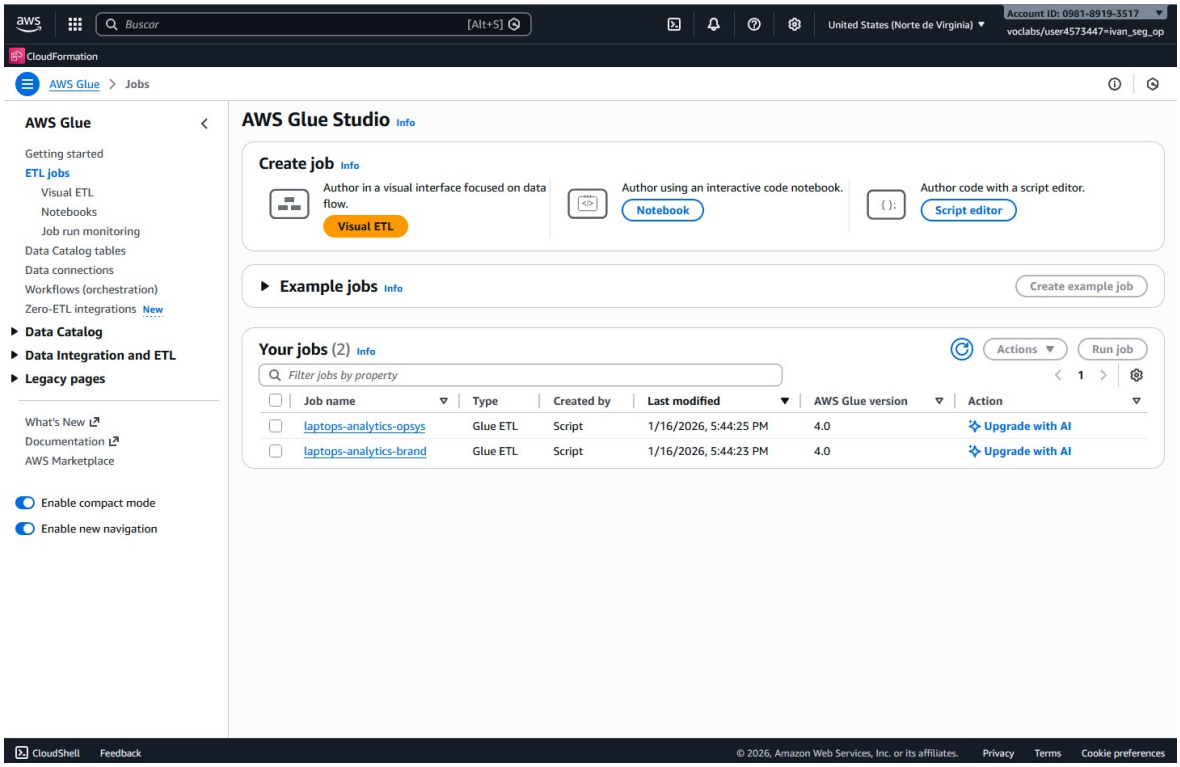

Figura 6: Estado de ejecución de los Glue Jobs.

Paso 5: Resultados Procesados

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

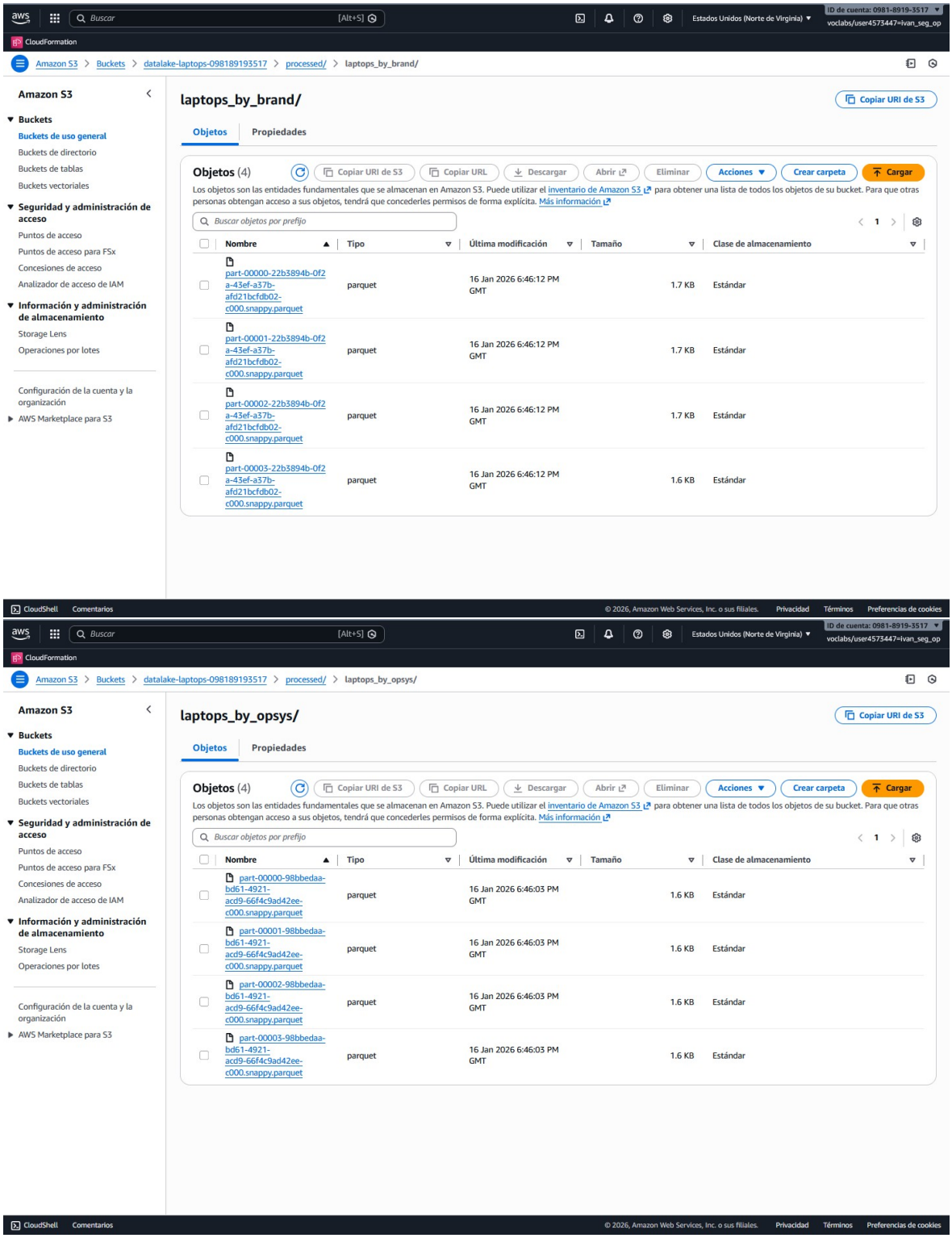

Figura 7: Estructura de salida en S3 para Marcas (up) y SO (down).

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.

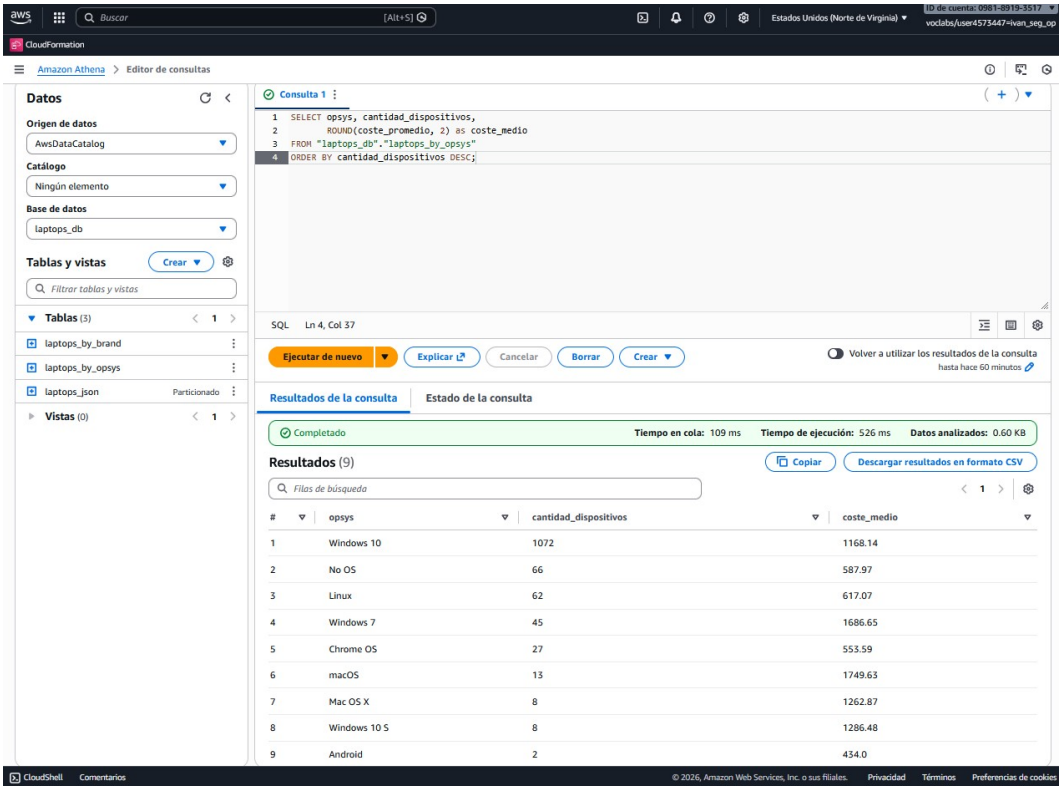

Figura 8: Consulta SQL de verificación inicial.

Consulta 2: Análisis de precios por Marca.

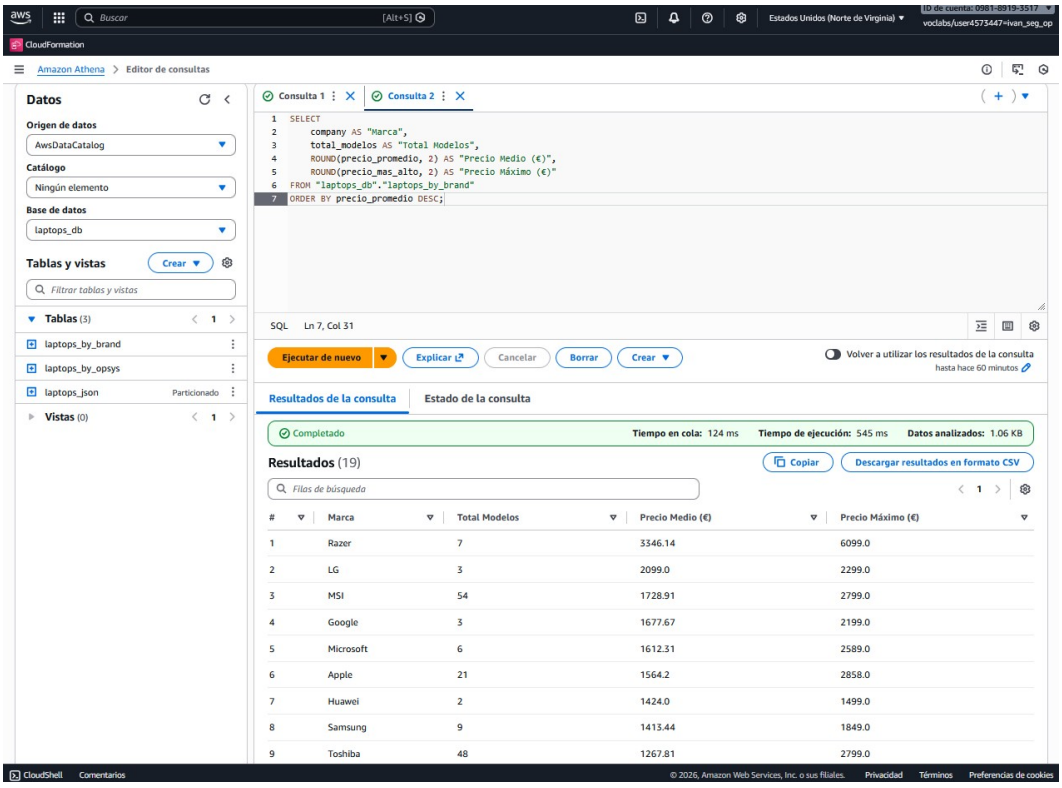

Figura 9: Ranking de marcas por precio promedio.

Consulta 3: Análisis de la cantidad media de RAM.

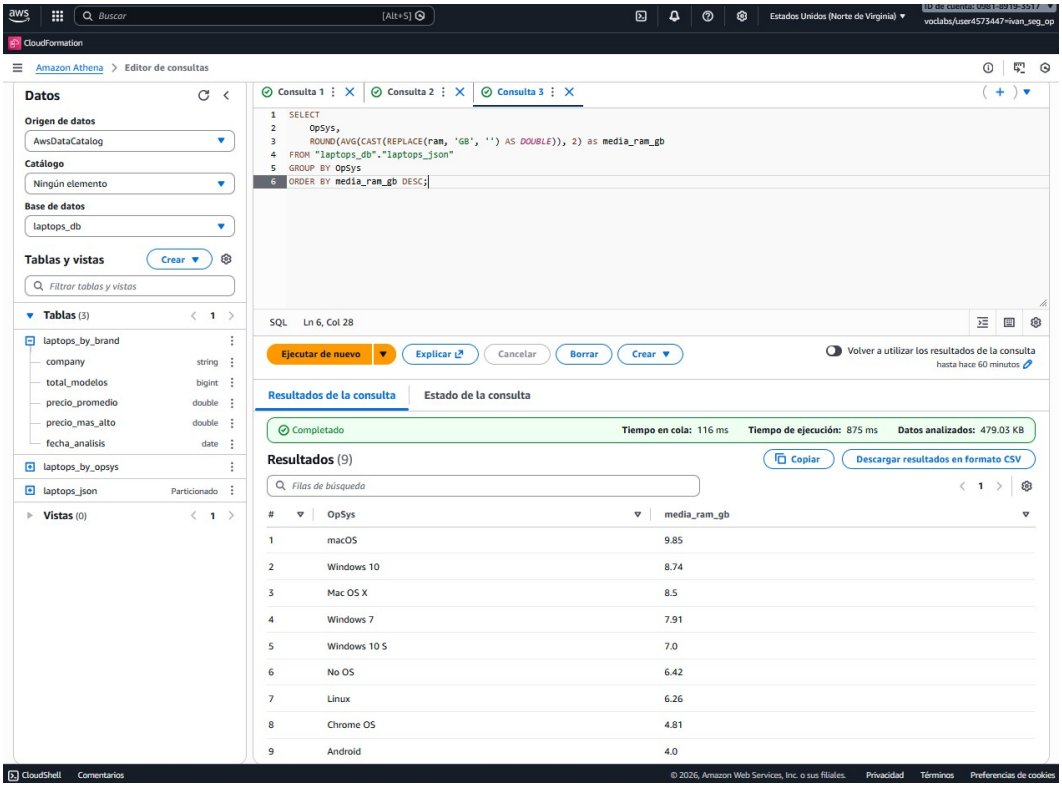

Figura 10: Agregación de dispositivos por Sistema Operativo.

7.2 Anexo B. Script de despliegue

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

```
# ======================================
# SCRIPT DE DESPLIEGUE AWS - POWERSHELL
# ======================================

# --- 1. CONFIGURACION ---
$env:AWS_REGION = "us-east-1"

# Asegurar cuenta de LabRole
$CustomRoleName = "LabRole"

Write-Host "Iniciando validacion de credenciales..." -ForegroundColor Cyan

# 1.1 Validar Credenciales AWS CLI
try {
    $identity = aws sts get-caller-identity --output json | ConvertFrom-Json
    $env:ACCOUNT_ID = $identity.Account
```

```
    Write-Host "Credenciales detectadas. Account ID: $($env:ACCOUNT_ID)"
        -ForegroundColor Green
} catch {
    Write-Error "ERROR FATAL: No se detectan credenciales de AWS. Ejecuta
        'aws configure' o refresca tu sesion."
    exit 1
}

$env:BUCKET_NAME = "datalake-laptops-$($env:ACCOUNT_ID)"

# 1.2 Obtener ARN del Rol
try {
    $env:ROLE_ARN = (aws iam get-role --role-name $CustomRoleName --query
        'Role.Arn' --output text).Trim()
    Write-Host "Rol encontrado: $($env:ROLE_ARN)" -ForegroundColor Green
} catch {
    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."
    exit 1
}

Write-Host "------------------------------------------------" -
    ForegroundColor Cyan
Write-Host "RESUMEN DE DESPLIEGUE:"
Write-Host "Bucket: $($env:BUCKET_NAME)"
Write-Host "Region: $($env:AWS_REGION)"
Write-Host "------------------------------------------------" -
    ForegroundColor Cyan

# --- 2. S3 & KINESIS ---
Write-Host "Build: Creando Bucket S3..." -ForegroundColor Yellow
try {
    aws s3 mb s3://$env:BUCKET_NAME 2>$null
} catch {
    Write-Host "   INFO: El bucket ya existe o no se pudo crear (
        verificar permisos)." -ForegroundColor Gray
}

Write-Host "Build: Creando estructura de carpetas..." -ForegroundColor
    Yellow
# Redirigir stderr a null para limpiar la salida si ya existen
aws s3api put-object --bucket $env:BUCKET_NAME --key raw/ 2>$null
aws s3api put-object --bucket $env:BUCKET_NAME --key raw/laptops_json/ 2>
    $null
aws s3api put-object --bucket $env:BUCKET_NAME --key processed/ 2>$null
aws s3api put-object --bucket $env:BUCKET_NAME --key config/ 2>$null
```

```
aws s3api put-object --bucket $env:BUCKET_NAME --key scripts/ 2>$null
aws s3api put-object --bucket $env:BUCKET_NAME --key logs/ 2>$null

Write-Host "Build: Creando Kinesis Stream..." -ForegroundColor Yellow
try {
    aws kinesis create-stream --stream-name laptops-stream --shard-count
        1 2>$null
} catch {
    Write-Host "   INFO: El stream probablemente ya existe." -
        ForegroundColor Gray
}

# --- 3. LAMBDA ---
Write-Host "Build: Empaquetando y desplegando Lambda..." -ForegroundColor
    Yellow

if (Test-Path "firehose.zip") { Remove-Item "firehose.zip" }
# Verificar que exista el archivo python
if (-not (Test-Path "firehose.py")) {
    Write-Error "Falta el archivo 'firehose.py' en el directorio actual."
    exit 1
}
Compress-Archive -Path "firehose.py" -DestinationPath "firehose.zip"

# Intentar crear.
aws lambda create-function `
    --function-name laptops-firehose-lambda `
    --runtime python3.12 `
    --role $env:ROLE_ARN `
    --handler firehose.lambda_handler `
    --zip-file fileb://firehose.zip `
    --timeout 60 `
    --memory-size 128 2>$null

if (-not $?) {
    Write-Host "   -> La Lambda ya existe, actualizando codigo..." -
        ForegroundColor DarkGray
    aws lambda update-function-code --function-name laptops-firehose-
        lambda --zip-file fileb://firehose.zip >$null
}

# Esperar propagacion
Start-Sleep -Seconds 5

$env:LAMBDA_ARN = (aws lambda get-function --function-name laptops-
    firehose-lambda --query 'Configuration.FunctionArn' --output text).
    Trim()
```

```
94
95  # --- 4. FIREHOSE ---
96  Write-Host "Build: Creando Firehose Delivery Stream..." -ForegroundColor
        Yellow
97
98  # Objeto de configuracion para JSON
99  $firehoseConfig = @{
100     BucketARN = "arn:aws:s3:::$env:BUCKET_NAME"
101     RoleARN = "$env:ROLE_ARN"
102     Prefix = "raw/laptops_json/processing_date=!{partitionKeyFromLambda:
            processing_date}/"
103     ErrorOutputPrefix = "errors/!{firehose:error-output-type}/"
104     BufferingHints = @{ SizeInMBs = 64; IntervalInSeconds = 60 }
105     DynamicPartitioningConfiguration = @{ Enabled = $true; RetryOptions =
            @{ DurationInSeconds = 300 } }
106     ProcessingConfiguration = @{
107         Enabled = $true
108         Processors = @(
109             @{
110                 Type = "Lambda"
111                 Parameters = @(
112                     @{ ParameterName = "LambdaArn"; ParameterValue = "
                            $env:LAMBDA_ARN" },
113                     @{ ParameterName = "BufferSizeInMBs"; ParameterValue
                            = "1" },
114                     @{ ParameterName = "BufferIntervalInSeconds";
                            ParameterValue = "60" }
115                 )
116             }
117         )
118     }
119 }
120
121 $firehoseConfig | ConvertTo-Json -Depth 10 | Out-File "firehose_config.
        json" -Encoding ASCII
122
123 aws firehose create-delivery-stream `
124     --delivery-stream-name laptops-delivery-stream `
125     --delivery-stream-type KinesisStreamAsSource `
126     --kinesis-stream-source-configuration "KinesisStreamARN=arn:aws:
            kinesis:$($env:AWS_REGION):$($env:ACCOUNT_ID):stream/laptops-
            stream,RoleARN=$env:ROLE_ARN" `
127     --extended-s3-destination-configuration file://firehose_config.json
            2>$null
128
129 Remove-Item "firehose_config.json"
130
```

```
# --- 5. GLUE DATABASE & CRAWLER ---
Write-Host "Build: Configurando Glue..." -ForegroundColor Yellow

Set-Content -Path "glue_db.json" -Value '{"Name":"laptops_db"}'
aws glue create-database --database-input file://glue_db.json 2>$null
Remove-Item "glue_db.json"

$crawlerConfig = @{
    Name = "laptops-raw-crawler"
    Role = "$env:ROLE_ARN"
    DatabaseName = "laptops_db"
    Targets = @{ S3Targets = @( @{ Path = "s3://$env:BUCKET_NAME/raw/
        laptops_json" } ) }
}
$crawlerConfig | ConvertTo-Json -Depth 5 | Out-File "glue_crawler.json" -
    Encoding ASCII

aws glue create-crawler --cli-input-json file://glue_crawler.json 2>$null
Remove-Item "glue_crawler.json"

# Iniciar crawler si no esta corriendo
try {
    aws glue start-crawler --name laptops-raw-crawler 2>$null
} catch {}

# --- 6. GLUE JOBS ---
Write-Host "Build: Subiendo scripts y creando Glue Jobs..." -
    ForegroundColor Yellow

# Subida de scripts (Validando que existan)
if (-not (Test-Path "laptops_analytics_brand.py")) { Write-Error "Falta
    laptops_analytics_brand.py"; exit 1 }
if (-not (Test-Path "laptops_analytics_so.py")) { Write-Error "Falta
    laptops_analytics_so.py"; exit 1 }

aws s3 cp laptops_analytics_brand.py s3://$env:BUCKET_NAME/scripts/
aws s3 cp laptops_analytics_so.py s3://$env:BUCKET_NAME/scripts/

# --- JOB 1: MARCAS ---
$jobBrandConfig = @{
    Name = "laptops-analytics-brand"
    Role = "$env:ROLE_ARN"
    Command = @{
        Name = "glueetl"
        ScriptLocation = "s3://$env:BUCKET_NAME/scripts/
            laptops_analytics_brand.py"
        PythonVersion = "3"
```

```
    }
    DefaultArguments = @{
        "--database" = "laptops_db"
        "--table" = "laptops_json"
        "--output_path" = "s3://$env:BUCKET_NAME/processed/
            laptops_by_brand/"
        "--enable-continuous-cloudwatch-log" = "true"
        "--spark-event-logs-path" = "s3://$env:BUCKET_NAME/logs/"
    }
    GlueVersion = "4.0"
    NumberOfWorkers = 2
    WorkerType = "G.1X"
}
$jobBrandConfig | ConvertTo-Json -Depth 5 | Out-File "job_brand.json" -
    Encoding ASCII

aws glue create-job --cli-input-json file://job_brand.json 2>$null
if (-not $?) { aws glue update-job --job-name laptops-analytics-brand --
    job-update file://job_brand.json >$null }
Remove-Item "job_brand.json"

# --- JOB 2: OPSYS ---
$jobOsConfig = @{
    Name = "laptops-analytics-opsys"
    Role = "$env:ROLE_ARN"
    Command = @{
        Name = "glueetl"
        ScriptLocation = "s3://$env:BUCKET_NAME/scripts/
            laptops_analytics_so.py"
        PythonVersion = "3"
    }
    DefaultArguments = @{
        "--database" = "laptops_db"
        "--table" = "laptops_json"
        "--output_path" = "s3://$env:BUCKET_NAME/processed/
            laptops_by_opsys/"
        "--enable-continuous-cloudwatch-log" = "true"
        "--spark-event-logs-path" = "s3://$env:BUCKET_NAME/logs/"
    }
    GlueVersion = "4.0"
    NumberOfWorkers = 2
    WorkerType = "G.1X"
}
$jobOsConfig | ConvertTo-Json -Depth 5 | Out-File "job_os.json" -Encoding
    ASCII

aws glue create-job --cli-input-json file://job_os.json 2>$null
```

```powershell
if (-not $?) { aws glue update-job --job-name laptops-analytics-opsys --job-update file://job_os.json >$null }
Remove-Item "job_os.json"

# --- 7. FINALIZACION ---

Write-Host "Infraestructura desplegada correctamente" -ForegroundColor Green
Write-Host "------------------------------------------------"
Write-Host "Pasos siguientes:"
Write-Host "1. Ejecutar 'uv run kinesis.py' para enviar datos."
Write-Host "2. Espera 2-5 minutos."
Write-Host "3. Comprobar dentro de AWS Glue."
Write-Host "4. Ejecutar los Jobs 'laptops-analytics-brand' y 'laptops-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.

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

# CONFIGURACION
STREAM_NAME = 'laptops-stream'
REGION = 'us-east-1'
INPUT_FILE = 'datos.json'

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

def load_data(file_path):
    with open(file_path, 'r', encoding='utf-8') as f:
        return json.load(f)

def run_producer():
    laptops = load_data(INPUT_FILE)
    records_sent = 0

    logger.info(f"Iniciando transmision de {len(laptops)} laptops al stream: {STREAM_NAME}...")

    for laptop in laptops:
```

```
        # Extraer campos clave para el log
        laptop_id = laptop.get('laptop_ID')
        company = laptop.get('Company')
        price = laptop.get('Price_euros')

        # Enviar a Kinesis
        try:
            response = kinesis.put_record(
                StreamName=STREAM_NAME,
                Data=json.dumps(laptop),  # Enviar el objeto laptop entero
                PartitionKey=str(company) # Agrupar shards por marca (Apple, Dell, etc.)
            )

            records_sent += 1
            logger.info(f"Enviado ID:{laptop_id} ({company}) - {price} euros  Shard: {response['ShardId'][-5:]}")
            # Pequenia pausa para ver el efecto streaming
            time.sleep(0.2)

        except Exception as e:
            logger.error(f"Error enviando laptop {laptop_id}: {e}")

    logger.info(f"Fin de la transmision. Total registros enviados: {records_sent}")

if __name__ == '__main__':
    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.

```
import json
import base64
import datetime

def lambda_handler(event, context):
    output = []

    for record in event['records']:
        try:
            # 1. Decodificar entrada de Kinesis
```

```
            payload = base64.b64decode(record['data']).decode('utf-8')
            data_json = json.loads(payload)

            # 2. Aniadir metadatos de procesamiento (Timestamp)
            processing_time = datetime.datetime.now(datetime.timezone.utc
                )
            data_json['processed_at'] = processing_time.isoformat()

            # 3. Preparar Partition Key para S3 (YYYY-MM-DD)
            partition_date = processing_time.strftime('%Y-%m-%d')

            # 4. Re-codificar para Firehose (Debe terminar en salto de
                linea para JSON Lines)
            output_payload = json.dumps(data_json) + '\n'

            output_record = {
                'recordId': record['recordId'],
                'result': 'Ok',
                'data': base64.b64encode(output_payload.encode('utf-8')).
                    decode('utf-8'),
                'metadata': {
                    'partitionKeys': {
                        'processing_date': partition_date
                    }
                }
            }
            output.append(output_record)

        except Exception as e:
            # Si falla un registro, se marca como failed pero no se
                detiene todo el lote
            print(f"Error processing record: {e}")
            output.append({
                'recordId': record['recordId'],
                'result': 'ProcessingFailed',
                'data': record['data']
            })

    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.

```
import sys
```

```
import logging
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.utils import getResolvedOptions
from pyspark.sql.functions import col, avg, count, max as spark_max,
    current_date
from awsglue.dynamicframe import DynamicFrame

# Configuracion de Logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname
    )s - %(message)s')
logger = logging.getLogger(__name__)

def main():
    args = getResolvedOptions(sys.argv, ['database', 'table', '
        output_path'])
    database = args['database']
    table = args['table']
    output_path = args['output_path']

    logger.info(f"Iniciando Analytics por Marca. DB: {database}, Table: {
        table}")

    sc = SparkContext()
    glueContext = GlueContext(sc)
    spark = glueContext.spark_session

    # 1. Leer datos
    dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
        database=database,
        table_name=table
    )

    # --- CORRECCION DE TIPO DE DATO (CHOICE) ---
    # Esto fuerza a que Price_euros sea tratado siempre como double,
    # resolviendo la ambiguedad entre int y double.
    try:
        dynamic_frame = dynamic_frame.resolveChoice(specs = [('
            Price_euros','cast:double')])
    except:
        logger.warning("No se requirio resolveChoice o fallo, continuando
            ...")

    df = dynamic_frame.toDF()

    if df.count() == 0:
        logger.warning("No se encontraron datos de laptops. Finalizando."
```

```
            )
        return

    # Asegurar cast final
    df = df.withColumn("Price_euros", col("Price_euros").cast("double"))

    # 2. LOGICA: Agregacion por MARCA
    agg_df = df.groupBy("Company") \
        .agg(
            count("laptop_ID").alias("total_modelos"),
            avg("Price_euros").alias("precio_promedio"),
            spark_max("Price_euros").alias("precio_mas_alto")
        ) \
        .withColumn("fecha_analisis", current_date()) \
        .orderBy("precio_promedio", ascending=False)

    # 3. Escribir resultados
    output_dynamic_frame = DynamicFrame.fromDF(agg_df, glueContext, "
        output")

    glueContext.write_dynamic_frame.from_options(
        frame=output_dynamic_frame,
        connection_type="s3",
        connection_options={ "path": output_path },
        format="parquet",
        format_options={"compression": "snappy"}
    )

    logger.info(f"Job completado.")

if __name__ == "__main__":
    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.

```
import sys
import logging
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.utils import getResolvedOptions
from pyspark.sql.functions import col, avg, count, min as spark_min, max
    as spark_max
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 = [('
               Price_euros','cast:double')])
32     except:
33         pass
34
35     df = dynamic_frame.toDF()
36     if df.count() == 0:
37         logger.warning("Sin datos.")
38         return
39
40     df = df.withColumn("Price_euros", col("Price_euros").cast("double"))
41
42     # 2. LOGICA: Agregacion por OS
43     agg_df = df.groupBy("OpSys") \
44         .agg(
45             count("laptop_ID").alias("cantidad_dispositivos"),
46             avg("Price_euros").alias("coste_promedio"),
47             spark_min("Price_euros").alias("coste_minimo"),
48             spark_max("Price_euros").alias("coste_maximo")
49         ) \
50         .orderBy("coste_promedio", ascending=False)
```

```
51
52      output_dynamic_frame = DynamicFrame.fromDF(agg_df, glueContext, "
            output")
53
54      glueContext.write_dynamic_frame.from_options(
55          frame=output_dynamic_frame,
56          connection_type="s3",
57          connection_options={ "path": output_path },
58          format="parquet",
59          format_options={"compression": "snappy"}
60      )
61      logger.info(f"Job completado.")
62  if __name__ == "__main__":
63      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://$env:BUCKET_NAME/processed/laptops_by_brand/" },
12         @{ Path = "s3://$env:BUCKET_NAME/processed/laptops_by_opsys/" }
13     )}
14 }
15
16 # 3. Guardar JSON temporal y crear Crawler
17 $resultsCrawler | ConvertTo-Json -Depth 5 | Out-File "
       glue_results_crawler.json" -Encoding ASCII
18 aws glue create-crawler --cli-input-json file://glue_results_crawler.json
19
20 # 4. Arrancar el Crawler
21 Write-Host "Iniciando crawler..." -ForegroundColor Green
22 aws glue start-crawler --name laptops-processed-crawler
23
```

```powershell
# 5. Limpieza
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.

```powershell
Write-Host "INICIANDO LIMPIEZA TOTAL (V2)..." -ForegroundColor Red

# Evita fallos si se ha cerrado la terminal
try {
    $env:ACCOUNT_ID = (aws sts get-caller-identity --query Account --
        output text).Trim()
    $env:BUCKET_NAME = "datalake-laptops-$($env:ACCOUNT_ID)"
    Write-Host "Cuenta detectada: $env:ACCOUNT_ID" -ForegroundColor Gray
    Write-Host "Bucket objetivo: $env:BUCKET_NAME" -ForegroundColor Gray
} catch {
    Write-Error "No se detectan credenciales AWS. Ejecuta 'aws configure'
        primero."
    exit 1
}

# --- 1. S3 (ELIMINACION FORZADA) ---
Write-Host "1. Eliminando Bucket S3..."
# El 2>$null oculta errores si el bucket ya no existe
aws s3 rb s3://$env:BUCKET_NAME --force 2>$null

# --- 2. STREAMS (LO MAS CARO) ---
Write-Host "2. Eliminando Streams..."
aws firehose delete-delivery-stream --delivery-stream-name laptops-
    delivery-stream 2>$null
aws kinesis delete-stream --stream-name laptops-stream 2>$null

# --- 3. GLUE (CRAWLERS, JOBS, DB) ---
Write-Host "3. Eliminando recursos Glue..."
aws glue delete-crawler --name laptops-raw-crawler 2>$null
aws glue delete-crawler --name laptops-processed-crawler 2>$null
aws glue delete-job --job-name laptops-analytics-brand 2>$null
aws glue delete-job --job-name laptops-analytics-opsys 2>$null

# Nota: Glue no deja borrar la DB si tiene tablas dentro.
# Intentar borrar las tablas primero (brute force simple)
aws glue delete-table --database-name laptops_db --name laptops_json 2>
    $null
```

```
aws glue delete-table --database-name laptops_db --name laptops_by_brand 2>$null
aws glue delete-table --database-name laptops_db --name laptops_by_opsys 2>$null
aws glue delete-database --name laptops_db 2>$null

# --- 4. LAMBDA ---
Write-Host "4. Eliminando Lambda..."
aws lambda delete-function --function-name laptops-firehose-lambda 2>$null

# --- 5. CLOUDWATCH LOGS (LIMPIEZA DE RASTROS) ---
Write-Host "5. Limpiando Logs residuales..."
aws logs delete-log-group --log-group-name "/aws/lambda/laptops-firehose-lambda" 2>$null
aws logs delete-log-group --log-group-name "/aws-glue/crawlers" 2>$null
aws logs delete-log-group --log-group-name "/aws-glue/jobs/laptops-analytics-brand" 2>$null
aws logs delete-log-group --log-group-name "/aws-glue/jobs/laptops-analytics-opsys" 2>$null

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 herramenientas 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.