

Práctica 7

Computación en la Nube

Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería Informática

Grado en Ingeniería Informática

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	5
4. Presupuesto y estimación de costes	5
5. Conclusiones	6
6. Referencias y bibliografía	6
7. Anexos	7
7.1. Anexo A. Script de despliegue	7
7.2. Anexo B. Consumidor Firehose	13
7.3. Anexo C. Análisis por marca	14
7.4. Anexo D. Análisis por sistema operativo	16
7.5. Anexo E. Crawler de AWS Glue	18
7.6. Anexo F. Script de limpieza	19

1 Introducción

El objetivo principal de esta práctica es el diseño, despliegue y validación de una arquitectura completa 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 hasta su transformación final para la toma de decisiones.

La solución adopta un enfoque *serverless* (sin servidor), minimizando la carga operativa de administración de infraestructura. Los componentes clave seleccionados son:

- **Amazon S3:** Actúa como la capa de persistencia centralizada (Data Lake), proporcionando almacenamiento escalable y duradero.
- **Amazon Kinesis Data Streams:** Facilita la ingesta de datos de alta velocidad y baja latencia, desacoplando los productores de los consumidores.
- **Amazon Kinesis Data Firehose:** Gestiona la entrega fiable de los datos en streaming hacia el almacenamiento, gestionando buffers y transformaciones intermedias.
- **AWS Glue:** Proporciona un entorno unificado para el descubrimiento de metadatos (Data Catalog) y la ejecución de trabajos ETL (Extract, Transform, Load).

La arquitectura implementada procesa un conjunto de datos referente a un inventario de dispositivos portátiles. Cada registro contiene atributos heterogéneos como marca, modelo, resolución de pantalla, especificaciones de hardware (CPU, RAM, GPU), sistema operativo y precio (`deploy.ps1`). El flujo termina con la generación de datasets analíticos en formato columnar, listos para ser consultados mediante SQL.

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 `deploy.ps1`, aplicando principios de Infraestructura como Código (IaC). El nombre del bucket se genera de forma dinámica concatenando el prefijo `datalake-laptops-` con el identificador único de la cuenta AWS, asegurando así un espacio de nombres globalmente único.

La estructura de carpetas definida sigue las mejores prácticas de organización de datos por capas ("zonas"), lo que facilita 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/`: O "Curated Zone". Contiene los resultados refinados generados por los trabajos ETL. Los datos aquí están limpios, enriquecidos y convertidos a formato Parquet para optimizar el rendimiento de lectura y reducir costes de almacenamiento.
- `config/`: Repositorio para ficheros de configuración externos o parámetros que puedan requerir los procesos.
- `scripts/`: Almacenamiento de los códigos fuente Python/Spark utilizados por los trabajos de AWS Glue.
- `logs/`: Centralización de registros de eventos y errores para facilitar la depuración y auditoría del sistema.

Esta segmentación permite aplicar políticas de seguridad y retención diferenciadas según la criticidad y el uso de los datos en cada etapa.

2.2 Implementación del productor de datos

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

Para garantizar una simulación realista de un entorno de streaming, el productor introduce latencias artificiales entre envíos. Cada registro enviado incluye una clave de partición (*Partition Key*), en este caso basada en la marca del dispositivo (*Company*), lo que asegura que los datos relacionados se dirijan al mismo shard, manteniendo el orden secuencial si fuera necesario.

```
1 aws kinesis create-stream --stream-name energy-stream --shard-count 1
```

Listing 1: Comando de creación del Stream con capacidad provisionada

El uso de `shard-count 1` define la capacidad inicial de ingesta (1 MB/segundo o 1000 registros/segundo), suficiente para el volumen de datos de esta práctica, aunque escalable horizontalmente según la demanda. El código completo del productor, incluyendo el manejo de excepciones y logging, se incluye en el Anexo.

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. Se ha configurado un *Delivery Stream* que consume datos directamente del stream de Kinesis y los deposita en la ruta `raw/` del bucket S3.

Una característica crítica implementada en esta fase es la **Transformación de Datos mediante Lambda**. Antes de escribir en S3, Firehose invoca una función AWS Lambda (`laptops-firehose-lambda`) que realiza dos tareas fundamentales:

1. **Enriquecimiento temporal:** Añade un campo `processed_at` con la marca de tiempo UTC exacta del procesamiento.
2. **Formateo y Particionamiento:** Convierte el JSON estándar en JSON Lines (un registro por línea) y define las claves de partición dinámicas (`partitionKeyFromLambda`).

Esto habilita el "Particionamiento Dinámico" en S3, organizando los archivos físicamente en carpetas por fecha (e.g., `processing_date=2023-10-27/`), lo que mejora drásticamente la eficiencia de los crawlers y las consultas futuras.

```

1 # Agrega timestamp de procesamiento
2 data_json['processed_at'] = processing_time.isoformat()
3 # Define la estructura de carpetas en S3 basada en la fecha
4 partition_date = processing_time.strftime('%Y-%m-%d')
```

Listing 2: Lógica de particionado y time-stamping en Lambda

Además, se ha configurado un buffer en Firehose (por tamaño o por tiempo, 64MB o 60 segundos) para optimizar el tamaño de los archivos resultantes y reducir las llamadas a la API de S3.

2.4 Configuración de AWS Glue

AWS Glue se utiliza como componente central de integración de datos. La configuración se divide en dos fases: metadatos y procesamiento.

Primero, se define el **Data Catalog**. Se crea la base de datos lógica `laptops_db` y se despliega un Crawler (`laptops-raw-crawler`). Este crawler inspecciona periódicamente la ruta `raw/` en S3, infiere el esquema de los archivos JSON (detectando tipos de datos como string, double, integer) y actualiza la tabla `laptops_json` en el catálogo. Esto permite que los datos no estructurados sean consultables como si fueran una tabla relacional.

Posteriormente, se implementan dos trabajos (*Glue Jobs*) basados en Spark:

- **Analytics por Marca:** Agrega datos por fabricante, calculando métricas como precio medio, máximo y conteo de modelos.
- **Analytics por SO:** Realiza un análisis similar segmentado por sistema operativo.

Un desafío técnico resuelto en los scripts fue la inconsistencia de tipos de datos (Schema Evolution). Se utilizó la función `resolveChoice` de `DynamicFrame` para castear explícitamente el campo `Price_euros` a `double`, evitando errores cuando Spark encontraba enteros y decimales mezclados. Finalmente, los datos se escriben en formato **Parquet** con compresión **Snappy**, un estándar de la industria que ofrece un esquema columnar altamente eficiente para analítica.

3 Diagrama del flujo de datos

La arquitectura implementada establece un pipeline unidireccional que garantiza la integridad y trazabilidad 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.
2. **Buffer y Transformación:** Kinesis Firehose recoge los registros, invoca a la función Lambda para añadir metadatos y acumula los datos en memoria.
3. **Almacenamiento Raw:** Firehose vuelca los bloques de datos en la zona raw de S3, particionados por fecha.
4. **Descubrimiento:** El Crawler de Glue escanea S3 y actualiza el esquema en el Catálogo de Datos.
5. **Procesamiento Batch:** Los Glue Jobs leen del catálogo, ejecutan transformaciones Spark en memoria distribuida y agregan la información.
6. **Almacenamiento Processed:** Los resultados finales se escriben en la zona processed de S3 en formato Parquet.

La Figura ?? ilustra gráficamente esta topología, destacando la separación entre la capa de streaming (Kinesis) y la capa batch (Glue).

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.

El modelo de costes varía según el servicio:

- **Kinesis:** Se factura por "Shard Hour". Un shard activo 24/7 tiene un coste fijo, independientemente del tráfico (hasta su límite).
- **S3:** Modelo "Pay-as-you-go" basado en almacenamiento GB/mes y número de peticiones PUT/GET.
- **Glue:** Facturación por DPU-Hour (Data Processing Unit). Se cobra por el tiempo de ejecución de los jobs y crawlers, con un mínimo de facturación.
- **Lambda:** Coste basado en número de invocaciones y GB-segundo de memoria utilizada.

Cuadro 1: Estimación detallada de costes de servicios AWS (Escenario mensual)

Servicio	Coste mensual (€)	Coste anual (€)
Amazon S3 (Almacenamiento y Peticiones)	5	60
Amazon Kinesis Data Streams (1 Shard)	15	180
AWS Glue (Crawlers y Jobs ETL diarios)	10	120
AWS Lambda (Computación y Free Tier)	2	24
Total Estimado	32	384

Cabe destacar que, para cargas de trabajo de laboratorio o desarrollo, muchos de estos costes se mitigan gracias a la Capa Gratuita (Free Tier) de AWS, especialmente en Lambda y S3.

5 Conclusiones

La realización de esta práctica ha permitido validar experimentalmente una arquitectura moderna de Data Lake sobre AWS. Se ha demostrado cómo la combinación de servicios desacoplados (Kinesis para transporte, S3 para almacenamiento, Glue para cómputo) ofrece una solución escalable, flexible y mantenible.

Entre las principales lecciones aprendidas y dificultades superadas, destacan:

- **Gestión de Tipos de Datos:** La importancia de manejar la heterogeneidad de los datos (enteros vs floats) en Spark para evitar fallos en tiempo de ejecución.
- **Particionamiento:** Cómo el particionamiento dinámico en la ingesta mejora significativamente la organización del Data Lake.
- **Infraestructura como Código:** El valor de utilizar scripts (PowerShell/AWS CLI) para desplegar recursos de forma repetible frente a la configuración manual en consola.

Como líneas de trabajo futuro, la arquitectura podría evolucionar incorporando **Amazon Athena** para consultas SQL interactivas sobre los datos procesados, o **Amazon QuickSight** para la visualización de cuadros de mando (dashboards) de inteligencia de negocios.

6 Referencias y bibliografía

- Amazon Web Services. (2024). *Amazon Kinesis Data Streams Developer Guide*. Recuperado de la documentación oficial de AWS.
- Amazon Web Services. (2024). *AWS Glue Developer Guide*. Sección sobre ETL Jobs y DynamicFrames.

- Amazon Web Services. (2024). *Amazon S3 User Guide*. Best practices for Data Lakes.
- Apache Spark. (2024). *PySpark Documentation*. DataFrame API y tipos de datos.

7 Anexos

En este apartado se incluye el código fuente completo desarrollado para la orquestación, ingesta y transformación de datos. Estos scripts constituyen la base operativa de la práctica.

7.1 Anexo A. Script de despliegue

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

```

1 # =====
2 # SCRIPT DE DESPLIEGUE AWS (Corregido y Actualizado)
3 # =====
4
5 # --- 1. CONFIGURACION ---
6 $env:AWS_REGION = "us-east-1"
7
8 # NOTA: Si usas tu propia cuenta personal (no Academy), cambia "
9   LabRole" por el nombre de tu rol (ej. "AdminRole")
10 $CustomRoleName = "LabRole"
11
12 Write-Host "Iniciando validacion de credenciales..." -ForegroundColor
13   Cyan
14
15 # 1.1 Validar Credenciales AWS CLI
16 try {
17     $identity = aws sts get-caller-identity --output json |
18       ConvertFrom-Json
19     $env:ACCOUNT_ID = $identity.Account
20     Write-Host "Credenciales detectadas. Account ID: $($env:ACCOUNT_ID
21       )" -ForegroundColor Green
22 } catch {
23     Write-Error "ERROR FATAL: No se detectan credenciales de AWS.
24       Ejecuta 'aws configure' o refresca tu sesion."
25     exit 1
26 }
27
28 $env:BUCKET_NAME = "datalake-laptops-$($env:ACCOUNT_ID)"
29
30 # 1.2 Obtener ARN del Rol
31 try {
32     $env:ROLE_ARN = (aws iam get-role --role-name $CustomRoleName --
33       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
42 Write-Host "Build: Creando Bucket S3..." -ForegroundColor Yellow
43 try {
44     aws s3 mb s3://$env:BUCKET_NAME 2>$null
45 } catch {
46     Write-Host "■■■INFO: El bucket ya existe o no se pudo crear (
        verificar permisos)." -ForegroundColor Gray
47 }
48
49 Write-Host "Build: Creando estructura de carpetas..." -ForegroundColor
        Yellow
50 # Redirigimos stderr a null para limpiar la salida si ya existen
51 aws s3api put-object --bucket $env:BUCKET_NAME --key raw/ 2>$null
52 aws s3api put-object --bucket $env:BUCKET_NAME --key raw/laptops_json/
        2>$null
53 aws s3api put-object --bucket $env:BUCKET_NAME --key processed/ 2>
        $null
54 aws s3api put-object --bucket $env:BUCKET_NAME --key config/ 2>$null
55 aws s3api put-object --bucket $env:BUCKET_NAME --key scripts/ 2>$null
56 aws s3api put-object --bucket $env:BUCKET_NAME --key logs/ 2>$null
57
58 Write-Host "Build: Creando Kinesis Stream..." -ForegroundColor Yellow
59 try {
60     aws kinesis create-stream --stream-name energy-stream --shard-
        count 1 2>$null
61 } catch {
62     Write-Host "■■■INFO: El stream probablemente ya existe." -
        ForegroundColor Gray

```

```

63 }
64
65 # --- 3. LAMBDA ---
66
67 Write-Host "Build: Empaquetando y desplegando Lambda..." -
    ForegroundColor Yellow
68
69 if (Test-Path "firehose.zip") { Remove-Item "firehose.zip" }
70 # Verificamos que exista el archivo python
71 if (-not (Test-Path "firehose.py")) {
72     Write-Error "Falta el archivo 'firehose.py' en el directorio
        actual."
73     exit 1
74 }
75 Compress-Archive -Path "firehose.py" -DestinationPath "firehose.zip"
76
77 # Intentar crear.
78 aws lambda create-function '
79     --function-name laptops-firehose-lambda '
80     --runtime python3.12 '
81     --role $env:ROLE_ARN '
82     --handler firehose.lambda_handler '
83     --zip-file fileb://firehose.zip '
84     --timeout 60 '
85     --memory-size 128 2>$null
86
87 if (-not $?) {
88     Write-Host "La Lambda ya existe, actualizando código..." -
        ForegroundColor DarkGray
89     aws lambda update-function-code --function-name laptops-firehose-
        lambda --zip-file fileb://firehose.zip >$null
90 }
91
92 # Esperar propagacion
93 Start-Sleep -Seconds 5
94
95 $env:LAMBDA_ARN = (aws lambda get-function --function-name laptops-
    firehose-lambda --query 'Configuration.FunctionArn' --output text)
    .Trim()
96
97 # --- 4. FIREHOSE ---
98
99 Write-Host "Build: Creando Firehose Delivery Stream..." -
    ForegroundColor Yellow
100
101 # Objeto de configuracion para JSON
102 $firehoseConfig = @{

```

```

103 BucketARN = "arn:aws:s3:::$env:BUCKET_NAME"
104 RoleARN = "$env:ROLE_ARN"
105 Prefix = "raw/laptops_json/processing_date=!{
      partitionKeyFromLambda:processing_date}/"
106 ErrorOutputPrefix = "errors/{firehose:error-output-type}/"
107 BufferingHints = @{ SizeInMBs = 64; IntervalInSeconds = 60 }
108 DynamicPartitioningConfiguration = @{ Enabled = $true;
      RetryOptions = @{ DurationInSeconds = 300 } }
109 ProcessingConfiguration = @{
110     Enabled = $true
111     Processors = @(
112         @{
113             Type = "Lambda"
114             Parameters = @(
115                 @{ ParameterName = "LambdaArn"; ParameterValue = "
                      $env:LAMBDA_ARN" },
116                 @{ ParameterName = "BufferSizeInMBs";
                      ParameterValue = "1" },
117                 @{ ParameterName = "BufferIntervalInSeconds";
                      ParameterValue = "60" }
118             )
119         }
120     )
121 }
122 }
123
124 $firehoseConfig | ConvertTo-Json -Depth 10 | Out-File "firehose_config
      .json" -Encoding ASCII
125
126 aws firehose create-delivery-stream '
127     --delivery-stream-name laptops-delivery-stream '
128     --delivery-stream-type KinesisStreamAsSource '
129     --kinesis-stream-source-configuration "KinesisStreamARN=arn:aws:
      kinesis:$($env:AWS_REGION):$($env:ACCOUNT_ID):stream/energy-
      stream,RoleARN=$env:ROLE_ARN" '
130     --extended-s3-destination-configuration file://firehose_config.
      json 2>$null
131
132 Remove-Item "firehose_config.json"
133
134 # --- 5. GLUE DATABASE & CRAWLER ---
135
136 Write-Host "Build: Configurando Glue..." -ForegroundColor Yellow
137
138 Set-Content -Path "glue_db.json" -Value '{"Name":"laptops_db"}'
139 aws glue create-database --database-input file://glue_db.json 2>$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/
        laptops_json" } ) }
147 }
148 $crawlerConfig | ConvertTo-Json -Depth 5 | Out-File "glue_crawler.json
    " -Encoding ASCII
149
150 aws glue create-crawler --cli-input-json file://glue_crawler.json 2>
    $null
151 Remove-Item "glue_crawler.json"
152
153 # Iniciar crawler si no esta corriendo
154 try {
155     aws glue start-crawler --name laptops-raw-crawler 2>$null
156 } catch {}
157
158 # --- 6. GLUE JOBS ---
159
160 Write-Host "Build: Subiendo scripts y creando Glue Jobs..." -
    ForegroundColor Yellow
161
162 # Subida de scripts (Validando que existan)
163 if (-not (Test-Path "laptops_analytics_brand.py")) { Write-Error "
    Falta laptops_analytics_brand.py"; exit 1 }
164 # Aqui corregimos el nombre del archivo OS
165 if (-not (Test-Path "laptops_analytics_so.py")) { Write-Error "Falta
    laptops_analytics_so.py"; exit 1 }
166
167 aws s3 cp laptops_analytics_brand.py s3://$env:BUCKET_NAME/scripts/
168 aws s3 cp laptops_analytics_so.py s3://$env:BUCKET_NAME/scripts/
169
170 # --- JOB 1: MARCAS ---
171 $jobBrandConfig = @{
172     Name = "laptops-analytics-brand"
173     Role = "$env:ROLE_ARN"
174     Command = @{
175         Name = "glueetl"
176         ScriptLocation = "s3://$env:BUCKET_NAME/scripts/
            laptops_analytics_brand.py"
177         PythonVersion = "3"
178     }
179     DefaultArguments = @{
180         "--database" = "laptops_db"

```

```

181     "--table" = "laptops_json"
182     "--output_path" = "s3://$env:BUCKET_NAME/processed/
        laptops_by_brand/"
183     "--enable-continuous-cloudwatch-log" = "true"
184     "--spark-event-logs-path" = "s3://$env:BUCKET_NAME/logs/"
185 }
186 GlueVersion = "4.0"
187 NumberOfWorkers = 2
188 WorkerType = "G.1X"
189 }
190 $jobBrandConfig | ConvertTo-Json -Depth 5 | Out-File "job_brand.json"
        -Encoding ASCII
191
192 aws glue create-job --cli-input-json file://job_brand.json 2>$null
193 if (-not $?) { aws glue update-job --job-name laptops-analytics-brand
        --job-update file://job_brand.json >$null }
194 Remove-Item "job_brand.json"
195
196 # --- JOB 2: OPSYS (Nombre de archivo corregido) ---
197 $jobOsConfig = @{
198     Name = "laptops-analytics-opsys"
199     Role = "$env:ROLE_ARN"
200     Command = @{
201         Name = "glueetl"
202         # NOTA: Aquí referenciamos el archivo correcto "
                laptops_analytics_so.py"
203         ScriptLocation = "s3://$env:BUCKET_NAME/scripts/
                laptops_analytics_so.py"
204         PythonVersion = "3"
205     }
206     DefaultArguments = @{
207         "--database" = "laptops_db"
208         "--table" = "laptops_json"
209         "--output_path" = "s3://$env:BUCKET_NAME/processed/
                laptops_by_opsys/"
210         "--enable-continuous-cloudwatch-log" = "true"
211         "--spark-event-logs-path" = "s3://$env:BUCKET_NAME/logs/"
212     }
213     GlueVersion = "4.0"
214     NumberOfWorkers = 2
215     WorkerType = "G.1X"
216 }
217 $jobOsConfig | ConvertTo-Json -Depth 5 | Out-File "job_os.json" -
        Encoding ASCII
218
219 aws glue create-job --cli-input-json file://job_os.json 2>$null
220 if (-not $?) { aws glue update-job --job-name laptops-analytics-opsys

```

```

--job-update file://job_os.json >$null }
221 Remove-Item "job_os.json"
222
223 # --- 7. FINALIZACION ---
224
225 Write-Host "Infraestructura desplegada correctamente" -ForegroundColor
    Green
226 Write-Host "-----"
227 Write-Host "Pasos siguientes:"
228 Write-Host "1. Instalar librerías: uv add boto3 loguru"
229 Write-Host "2. Ejecutar 'uv run kinesis.py' para enviar datos."
230 Write-Host "3. Esperar 2 minutos."
231 Write-Host "4. Comprobar dentro de AWS Glue."
232 Write-Host "5. Ejecutar los Jobs 'laptops-analytics-brand', 'laptops-
    analytics-opsys'."

```

Listing 3: Script deploy.ps1: Automatización de infraestructura AWS

7.2 Anexo B. Consumidor Firehose

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

```

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             # Esto nos servira para saber cuando se proceso el
16             registro
17             processing_time = datetime.datetime.now(datetime.timezone.
18                 utc)
19             data_json['processed_at'] = processing_time.isoformat()
20
21             # 3. Preparar Partition Key para S3 (YYYY-MM-DD)
22             partition_date = processing_time.strftime('%Y-%m-%d')

```

```

22         # 4. Re-codificar para Firehose (Debe terminar en salto de
           linea para JSON Lines)
23         output_payload = json.dumps(data_json) + '\n'
24
25         output_record = {
26             'recordId': record['recordId'],
27             'result': 'Ok',
28             'data': base64.b64encode(output_payload.encode('utf
                -8')).decode('utf-8'),
29             'metadata': {
30                 'partitionKeys': {
31                     'processing_date': partition_date
32                 }
33             }
34         }
35         output.append(output_record)
36
37     except Exception as e:
38         # Si falla un registro, lo marcamos como error pero no
           paramos todo
39         print(f"Error processing record: {e}")
40         output.append({
41             'recordId': record['recordId'],
42             'result': 'ProcessingFailed',
43             'data': record['data']
44         })
45
46     return {'records': output}

```

Listing 4: Script firehose.py: Lógica de transformación Lambda

7.3 Anexo C. Análisis por marca

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,
   current_date
7 from awsglue.dynamicframe import DynamicFrame
8
9 # Configuración de Logging
10 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(
   levelname)s - %(message)s')

```

```

11 logger = logging.getLogger(__name__)
12
13 def main():
14     args = getResolvedOptions(sys.argv, ['database', 'table', '
        output_path'])
15     database = args['database']
16     table = args['table']
17     output_path = args['output_path']
18
19     logger.info(f"Iniciando Analytics por Marca. DB: {database}, Table
        : {table}")
20
21     sc = SparkContext()
22     glueContext = GlueContext(sc)
23     spark = glueContext.spark_session
24
25     # 1. Leer datos
26     dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
27         database=database,
28         table_name=table
29     )
30
31     # --- CORRECCION DE TIPO DE DATO (CHOICE) ---
32     # Esto fuerza a que Price_euros sea tratado siempre como double,
33     # resolviendo la ambigüedad entre int y double.
34     try:
35         dynamic_frame = dynamic_frame.resolveChoice(specs = [(
            'Price_euros', 'cast:double')])
36     except:
37         logger.warning("No se requirio resolveChoice o fallo,
            continuando...")
38     # -----
39
40     df = dynamic_frame.toDF()
41
42     if df.count() == 0:
43         logger.warning("No se encontraron datos de laptops.
            Finalizando.")
44         return
45
46     # Asegurar cast final (por si acaso)
47     df = df.withColumn("Price_euros", col("Price_euros").cast("double"
        ))
48
49     # 2. LOGICA: Agregacion por MARCA
50     agg_df = df.groupBy("Company") \
51         .agg(

```

```

52         count("laptop_ID").alias("total_modelos"),
53         avg("Price_euros").alias("precio_promedio"),
54         spark_max("Price_euros").alias("precio_mas_alto")
55     ) \
56     .withColumn("fecha_analisis", current_date()) \
57     .orderBy("precio_promedio", ascending=False)
58
59     # 3. Escribir resultados
60     output_dynamic_frame = DynamicFrame.fromDF(agg_df, glueContext, "
        output")
61
62     glueContext.write_dynamic_frame.from_options(
63         frame=output_dynamic_frame,
64         connection_type="s3",
65         connection_options={ "path": output_path },
66         format="parquet",
67         format_options={"compression": "snappy"}
68     )
69
70     logger.info(f"Job completado.")
71
72 if __name__ == "__main__":
73     main()

```

Listing 5: Script laptops_analytics_brand.py: ETL de marcas

7.4 Anexo D. Análisis por sistema operativo

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 = [('Price_euros', 'cast:double')])
32     except:
33         pass
34     # -----
35
36     df = dynamic_frame.toDF()
37
38     if df.count() == 0:
39         logger.warning("Sin datos.")
40         return
41
42     df = df.withColumn("Price_euros", col("Price_euros").cast("double"))
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, "output")
55
56     glueContext.write_dynamic_frame.from_options(
57         frame=output_dynamic_frame,

```

```

58         connection_type="s3",
59         connection_options={ "path": output_path },
60         format="parquet",
61         format_options={"compression": "snappy"}
62     )
63
64     logger.info(f"Job completado.")
65
66 if __name__ == "__main__":
67     main()

```

Listing 6: Script laptops_analytics_so.py: ETL de Sistemas Operativos

7.5 Anexo E. Crawler de AWS Glue

Configuración JSON utilizada para desplegar el Crawler de descubrimiento de metadatos.

```

1 # 1. Asegurarnos de que tenemos las variables (por si acaso)
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 "
   glue_results_crawler.json" -Encoding ASCII
20 aws glue create-crawler --cli-input-json file://glue_results_crawler.
   json
21
22 # 4. Arrancar el Crawler
23 Write-Host "Iniciando crawler..." -ForegroundColor Green
24 aws glue start-crawler --name laptops-processed-crawler
25
26 # 5. Limpieza

```

```
25 Remove-Item "glue_results_crawler.json"
```

Listing 7: Script aws_crawler.ps1: Definición del Crawler

7.6 Anexo F. 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 # --- PASO 0: RECUPERAR NOMBRE DEL BUCKET AUTOMATICAMENTE ---
4 # Esto evita fallos si cerraste la terminal y perdiste la variable
   $env:BUCKET_NAME
5 try {
6     $env:ACCOUNT_ID = (aws sts get-caller-identity --query Account --
       output text).Trim()
7     $env:BUCKET_NAME = "datalake-laptops-$($env:ACCOUNT_ID)"
8     Write-Host "Cuenta detectada: $env:ACCOUNT_ID" -ForegroundColor
       Gray
9     Write-Host "Bucket objetivo: $env:BUCKET_NAME" -ForegroundColor
       Gray
10 } catch {
11     Write-Error "No se detectan credenciales AWS. Ejecuta 'aws
       configure' primero."
12     exit 1
13 }
14
15 # --- 1. S3 (ELIMINACION FORZADA) ---
16 Write-Host "1. Eliminando Bucket S3..."
17 # El 2>$null oculta errores si el bucket ya no existe
18 aws s3 rb s3://$env:BUCKET_NAME --force 2>$null
19
20 # --- 2. STREAMS (LO MAS CARO) ---
21 Write-Host "2. Eliminando Streams..."
22 aws firehose delete-delivery-stream --delivery-stream-name laptops-
   delivery-stream 2>$null
23 aws kinessis delete-stream --stream-name energy-stream 2>$null
24
25 # --- 3. GLUE (CRAWLERS, JOBS, DB) ---
26 Write-Host "3. Eliminando recursos Glue..."
27 aws glue delete-crawler --name laptops-raw-crawler 2>$null
28 aws glue delete-crawler --name laptops-processed-crawler 2>$null
29 aws glue delete-job --job-name laptops-analytics-brand 2>$null
30 aws glue delete-job --job-name laptops-analytics-opsys 2>$null
31
32 # Nota: Glue no deja borrar la DB si tiene tablas dentro.
```

```
33 # Intentamos borrar las tablas primero (brute force simple)
34 aws glue delete-table --database-name laptops_db --name laptops_json
    2>$null
35 aws glue delete-table --database-name laptops_db --name
    laptops_by_brand 2>$null
36 aws glue delete-table --database-name laptops_db --name
    laptops_by_opsys 2>$null
37 aws glue delete-database --name laptops_db 2>$null
38
39 # --- 4. LAMBDA ---
40 Write-Host "4. Eliminando Lambda..."
41 aws lambda delete-function --function-name laptops-firehose-lambda 2>
    $null
42
43 # --- 5. CLOUDWATCH LOGS (LIMPIEZA DE RASTROS) ---
44 Write-Host "5. Limpiando Logs residuales..."
45 # Esto borra los grupos de logs generados por tu Lambda y tus Jobs
46 aws logs delete-log-group --log-group-name "/aws/lambda/laptops-
    firehose-lambda" 2>$null
47 aws logs delete-log-group --log-group-name "/aws-glue/crawlers" 2>
    $null
48 aws logs delete-log-group --log-group-name "/aws-glue/jobs/laptops-
    analytics-brand" 2>$null
49 aws logs delete-log-group --log-group-name "/aws-glue/jobs/laptops-
    analytics-opsys" 2>$null
50
51 Write-Host "Limpieza completada - Cuenta limpia." -ForegroundColor
    Green
```

Listing 8: Script clean.ps1: Destrucción de recursos

Se ha utilizado inteligencia artificial únicamente como herramienta de apoyo para la redacción y corrección de estilo de la memoria técnica, garantizando que todo el código y la arquitectura han sido implementados y validados por el estudiante.