UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

GRADO EN INGENIERÍA INFORMÁTICA

COMPUTACIÓN EN LA NUBE - PRÁCTICA OBLIGATORIA 2

INGESTA Y PROCESAMIENTO DE DATOS EN AWS

Ayman Asbai Ghoudan
Curso académico 2025/26

Índice

1 Introducción

La segunda práctica entregable de la asignatura consiste en el correcto aprendizaje de la manera en la que recibir datos masivos de mano de AWS Kinesis, almacenarlos de forma segura con un bucket de S3 y someterlos a una transformación con AWS Glue para que puedan ser analizados posteriormente.

Para llevar esto a cabo, se ha ido desarrollando una arquitectura de procesamiento de datos escalable y desacoplada por medio de los servicios que se proporcionan en el laboratorio de AWS Academy. Entre otros aspectos, esta práctica contribuye a visualizar la relevancia de acciones como ingerir, almacenar y procesar flujos de datos en tiempo real, fundamental para cualquier organización hoy en día en el ámbito informático.

Además de ello, se ha simulado un escenario de streaming de datos utilizando un conjunto de datos que recoge los ganadores de premios Nobel desde 1901 hasta 2016. El flujo de trabajo implementado cubre el ciclo de vida completo del dato, tal que:

1. **Ingesta de Datos:** Se utiliza un script `kinesis.py` de Python para simular un productor que inyecta registros en tiempo real hacia *Amazon Kinesis Data Streams*, un servicio altamente escalable para el procesamiento de flujos de datos.
2. **Almacenamiento y Transformación Intermedia:** A través de *Amazon Kinesis Firehose*, los datos son recogidos del stream, transformados ligeramente mediante una función de *AWS Lambda* para permitir un particionado dinámico eficiente, y finalmente depositados en un datalake alojado en *Amazon S3*.
3. **Catalogación y ETL:** Una vez los datos residen en S3, se utiliza *AWS Glue*. En primer lugar, los *crawlers* de Glue descubren automáticamente el esquema de los datos. En segundo lugar, se ejecutan *jobs ETL* para agregar y analizar la información (uno de los casos utilizados en esta práctica ha sido obtener métricas a raíz del género), depositando los resultados procesados nuevamente en S3 en formato *parquet*.

De este modo, la arquitectura planteada permite separar adecuadamente las responsabilidades y lograr el objetivo de la práctica.

2 Desarrollo de las actividades

En este apartado de la memoria del trabajo, se detallarán cada uno de los pasos que se han seguido para cumplir con la implementación de cada uno de los componentes de la arquitectura.

2.1 Configuración del bucket S3

Amazon Simple Storage Service es considerado uno de los pilares fundamentales en el mundo de la computación en la nube dada su escalabilidad prácticamente ilimitada, su alta disponibilidad y su durabilidad. Al ser un servicio de almacenamiento de objetos, también conocido como "*el cubo para todo*", permite desacoplar el almacenamiento del cómputo, una característica esencial en los ecosistemas de datos modernos.

Su relevancia radica no sólo en la persistencia de los datos, sino en su capacidad para integrarse de forma nativa con servicios de análisis y procesamiento como AWS Glue y Amazon Athena.

El primer paso para consolidar el mencionado datalake es la creación de un bucket en Amazon S3, el cual servirá como repositorio central tanto para los datos crudos como para los datos procesados y los scripts de configuración.

La particularidad de este servicio de AWS es que debe disponer de un nombre único no sólo en términos de la cuenta de usuario, sino en toda la plataforma. De esta manera, cada bucket puede ser accedido mediante URLs como, por ejemplo:

`https://<nombre-del-bucket>.s3.amazonaws.com`

Es por ello que se ha seguido este patrón y se ha decidido configurar el mismo con un nombre característico como **datalake-laureates-<ID_CUENTA>**, en el que, al incluir el ID de cuenta, se garantiza que no pueda haber ningún conflicto. Dentro de este bucket, se ha definido una estructura de directorios jerárquica para mantener el orden y facilitar las políticas de ciclo de vida y acceso. La estructura implementada es la que se aprecia a continucación:

- **raw/**: Esta carpeta está destinada a almacenar los datos tal cual llegan desde el productor, sin modificaciones analíticas. Dentro de ésta, se utiliza la subcarpeta `laureates/` para los registros propios de los premios Nobel.
- **processed/**: Contendrá los resultados de los jobs ETL de Glue, almacenados típicamente en formatos columnares, tal como Parquet, para optimizar consultas posteriores.
- **config/** y **scripts/**: Almacenan ficheros de configuración y los scripts de Python utilizados por los jobs de Glue, asegurando que la infraestructura como código esté centralizada.

- **queries/** y **errors/**: Son carpetas auxiliares para guardar consultas de Athena o registros de errores generados por Kinesis Firehose si fallase la transformación o entrega.

Esta separación de carpetas, la cual puede apreciarse visualmente con la Figura 1, es fundamental para evitar mezclar datos de diferentes módulos de procesamiento y para simplificar la configuración de los crawlers de Glue, que pueden apuntar a carpetas específicas sin riesgo de leer archivos de configuración como si fueran datos.

Cabe destacar que no todas estas carpetas tienen uso actualmente, dado que están impuestas con visión a futuro para mejoras e integraciones, tales como puede ser AWS Athena, la cual permite ejecutar consultas/queries directamente sobre datos almacenados en el bucket de S3.

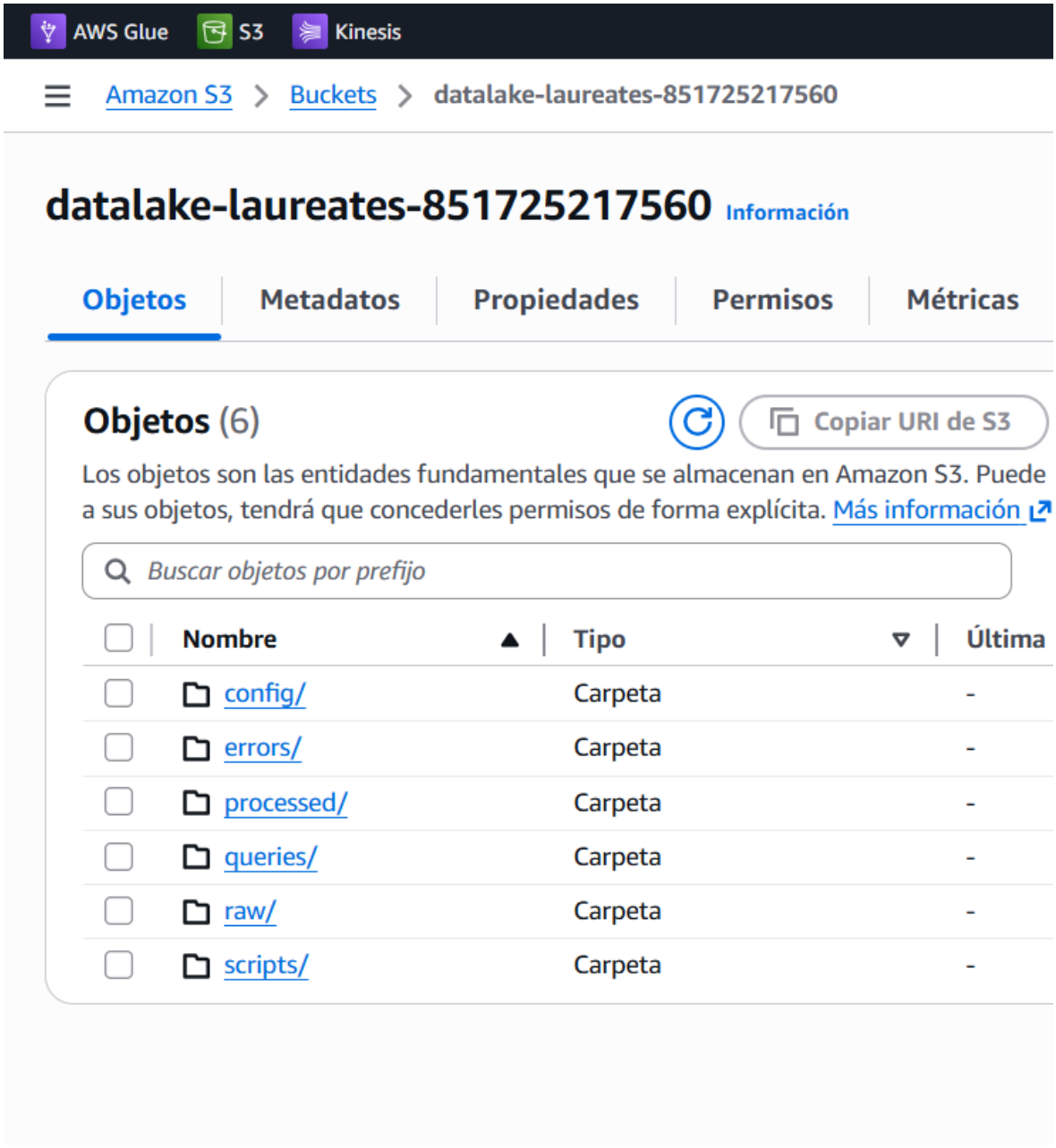

Figura 1: Estructura de carpetas creada en el bucket S3.

2.2 Implementación del productor de datos

El servicio de transmisión de datos del que se hará uso en esta sección es **Amazon Kinesis Data Streams** y, además de ser masivamente escalable y duradero, está diseñado para capturar y procesar grandes volúmenes de datos en tiempo real. Dentro de la arquitectura del datalake de *laureates*, Kinesis actúa como un búfer, permitiendo desacoplar a los productores de datos de los consumidores.

Su funcionamiento se basa en **shards**, que se definen como fragmentos, los cuales proporcionan una capacidad de procesamiento dedicada, permitiendo una ingesta ordenada y segura. En este proyecto, el uso de Kinesis es fundamental para transformar una carga de datos estática en un flujo continuo de elementos, simulando un entorno de producción donde la información llega de manera continuada.

Para simular la entrada de datos en tiempo real, se ha desarrollado un script en Python, denominado `kinesis.py`. Dicho script actúa como un productor que lee un fichero estático JSON (`nobel_laureates.json`) y envía cada registro individualmente a un *Amazon Kinesis Data Stream* llamado `laureates-stream`.

El diseño de este productor de datos incluye las siguientes características clave:

- **Carga de datos:** Se procede a leer el archivo JSON completo que contiene información histórica de todos los premiados (comprendidos entre 1901-2016).
- **Envío continuo:** Recorre la lista de laureados y utiliza la librería `boto3` para realizar la llamada a `put_record`, la cual se encarga de enviar a Kinesis el registro completo (con cada uno de los campos del registro: 'prize', 'full name', entre otros).
- **Clave de partición** Se utiliza el campo `Category` como clave de partición. Esto asegura que los registros de la misma categoría se dirijan preferentemente al mismo *shard* (fragmento) de Kinesis, lo cual puede ser beneficioso si se quisiera mantener el orden dentro de una categoría. No obstante, en este caso se utiliza un sólo shard para la práctica.
- **Simulación de un periodo de latencia:** También, se ha añadido un `time.sleep(0.1)` en el script entre envíos para evitar saturar el ancho de banda de subida y simular un flujo continuo de elementos, en lugar de una carga por lotes instantánea.

A continuación, pese a que el proyecto completo se encuentre adjuntado al final del documento (ver anexo 7.7), se muestra un extracto relevante del código del productor donde se realiza el envío del registro:

```
# Se itera sobre todos los laureados
for laureate in laureates_list:
    category = laureate.get('Category', 'Unknown')

    # Se envia a Kinesis el registro completo
    kinesis.put_record(
        StreamName=STREAM_NAME, # 'laureates-stream'
        Data=json.dumps(laureate),
        PartitionKey=category
    )
    # ... Log y sleep ...
```

Listing 1: Extracto del script del productor de Kinesis (7.7)

Este enfoque permite verificar en tiempo real cómo la métrica de `IncomingRecords` aumenta en la consola de Kinesis.

2.3 Configuración del consumidor: Kinesis Firehose

Como consumidor del stream de datos, se ha configurado el recurso Amazon Kinesis Data Firehose. Firehose, es un servicio de entrega totalmente gestionado que, en la presente arquitectura, no hace más que capturar los datos del stream y los deposita en el bucket de S3.

En cuanto a este caso, la configuración del (`laureates-delivery-stream`) incluye una funcionalidad avanzada y crucial para la organización del datalake: *el particionado dinámico*.

2.3.1 Transformación con AWS Lambda

Para lograr este particionado, Firehose invoca una función `laureates-firehose-lambda`) de AWS Lambda antes de escribir en S3. El código de esta función, alojado en `firehose.py`, realiza secuencialmente lo siguiente:

1. Recibe un lote de registros codificados en Base64.
2. Decodifica cada registro y calcula la fecha actual de procesamiento.
3. Añade metadatos al registro de salida, específicamente una clave `partitionKeys` con el valor de la fecha (`processing_date`).
4. Devuelve el registro a Firehose.

Esta transformación es ligera pero esencial, ya que permite a Firehose saber en qué carpeta exacta de S3 debe colocar cada archivo.

2.3.2 Configuración del Destino S3

En la configuración del destino S3 dentro de Firehose, se ha definido el prefijo utilizando la variable extraída por la Lambda, de modo que el prefijo configurado queda de la siguiente forma:

```
raw/laureates/processing_date=!partitionKeyFromLambda:processing_date
```

Esto resulta en que los archivos se guarden en rutas como `laureates-firehose-lambda`) automáticamente, por ejemplo. Este esquema de particionado conformado por parejas clave-valor optimiza enormemente el rendimiento de los Crawlers de Glue y las consultas posteriores en AWS Athena, ya que permite podar o deshacerse de particiones innecesarias al leer datos.

Además, tal y como se refleja en el script de despliegue de la arquitectura, se configuraron opciones de *Buffering* (conformadas por 64 MB o 60 segundos) para agrupar múltiples registros pequeños en archivos más grandes, mejorando la eficiencia de la entrada/salida en S3.

2.4 Configuración de AWS Glue

El componente final que conformará esta memoria es AWS Glue, el cual se encarga de catalogar y procesar los datos almacenados en S3. Su implementación se divide en dos etapas, la del descubrimiento de esquema y la de la transformación de datos. Seguidamente, se exponen dichas etapas de manera respectiva:

2.4.1 Crawlers y catálogo de datos

Para este proyecto, se han configurado dos crawlers:

1. **laureates-raw-crawler**: Apunta a la ruta `s3://.../raw/laureates/`. Su función es leer los archivos JSON generados por Firehose, inferir el esquema (que presenta campos como *id, first_name, category*, entre otros) y crear una tabla en la base de datos `laureates_db` del Glue Data Catalog.
2. **laureates-processed-crawler**: A diferencia del anterior, este apunta a la ruta `processed/`. Este se ejecuta después de los jobs ETL para catalogar las nuevas tablas de datos agregados.

El uso de Crawlers automatiza la gestión de los metadatos, de modo que si el esquema de los datos JSON cambiara en el futuro, el Crawler podría actualizar la definición de la tabla

automáticamente en la siguiente ejecución. Un ejemplo de este caso podría ser una situación donde se añadiera un nuevo campo al JSON de origen.

2.4.2 ETL Jobs

Se han desarrollado tres jobs de ETL utilizando la librería de PySpark, los cuales aprovechan la capacidad de procesamiento distribuido de Glue. Los scripts que se observan a continuación, leen la tabla catalogada (`raw`), realizan agregaciones y escriben el resultado en formato Parquet. Cada una de estas acciones son las que conforman el nombre del recurso ETL, asociándose a: Extract, Transform y Load.

- **Agregación por país:** Agrupa los datos por `birth_country` y `category`, contando el número total de premiados en este contexto. El resultado se escribe particionado por país, lo que facilita consultas geográficas, tales como podrían ser las regiones con más personas premiadas. (Anexo 7.1)
- **Agregación por década:** Calcula la década de cada premio a partir del campo `year`, y agrupa los resultados para mostrar tendencias temporales. (Anexo 7.2)
- **Agregación por género:** Analiza la distribución de premios entre hombres, mujeres y organizaciones, agrupando por `gender` y `category`. (Anexo 7.3)

Todos estos jobs siguen un patrón común y secuencial, que se basa en lo siguiente:

1. Inicializar el `GlueContext` y crear un *DynamicFrame* desde el catálogo.
2. Convertir dicho DynamicFrame a un DataFrame de Spark para facilitar las transformaciones, que son el filtrado, "groupingz conteo.
3. Escribir el resultado de nuevo a S3 utilizando `glueContext.write_dynamic_frame`.

El uso del formato Parquet con compresión Snappy en la salida garantiza que los datos procesados ocupen menos espacio y, además, sean mucho más rápidos de consultar que los JSON originales.

```
# Agrupar por pais de nacimiento y cataegoria
aggregated_df = df.filter(col("birth_country") != "") \
    .groupBy("birth_country", "category") \
    .agg(count("*").alias("total_laureates")) \
    .orderBy("birth_country", "category")

# Escritura a S3 en formato Parquet
glueContext.write_dynamic_frame.from_options(
    frame=output_dynamic_frame,
    connection_type="s3",
    connection_options={
```

```
        "path": output_path,
        "partitionKeys": ["birth_country"]
    },
    format="parquet",
    format_options={"compression": "snappy"}
)
```

Listing 2: Lógica del job by_country en PySpark (7.1)

3 Diagrama del flujo de datos

En esta sección, se verá el prototipo de diagrama del flujo de datos diseñado para este caso.

Tal y como se puede ver en la Figura 2, se ha tratado de trasladar la arquitectura descrita a lo largo de este documento en este diagrama.

De manera resumida, se comienza con un productor que envía datos de premios Nobel en tiempo real a Amazon Kinesis Data Streams. Los datos pasan a Kinesis Firehose, que utiliza una función Lambda para añadir una fecha de procesamiento antes de guardarla y se depositan los datos crudos (es decir, los denotados como *raw*) en la carpeta `/raw` de Amazon S3. Con un crawler de Glue se catalogan estos datos para que, seguidamente, los jobs de ETL los procesen y agreguen según corresponda (ya sea por país, género o por décadas). Finalmente, los datos procesados y optimizados se guardan nuevamente en Amazon S3, pero esta vez en una nueva carpeta `/processed`, listos para ser procesados.

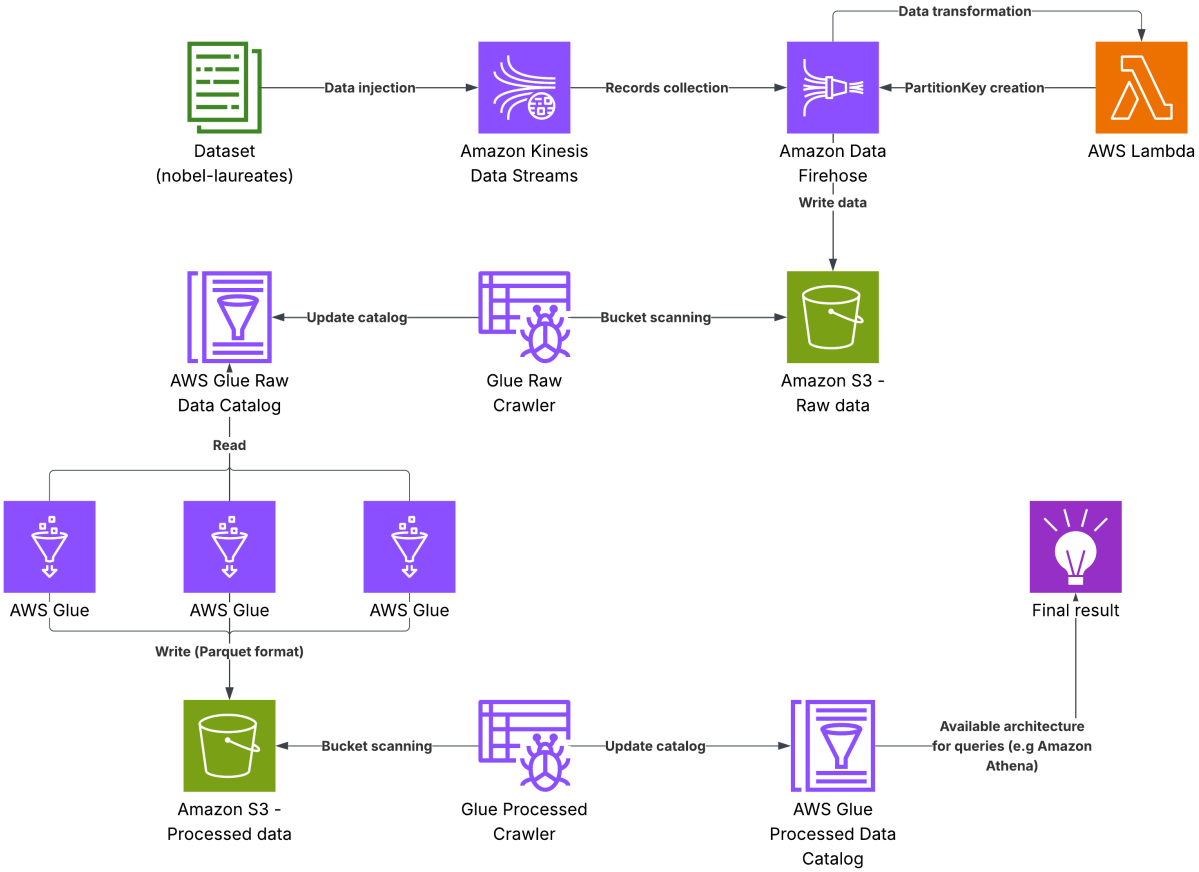

Figura 2: Diseño de la arquitectura de Nobel Laureates.

4 Presupuesto y estimación de costes

En relación a este apartado de la memoria, se detallará todo el planteamiento desarrollado para llevar a cabo el presupuesto y la estimación de los costes. Esta faceta es muy relevante en entornos de producción, puesto que no es suficiente disponer de una arquitectura escalable y funcional, sino que también es necesario contar con una buena estrategia que minimice los costes para llegar a un objetivo final con el máximo beneficio posible.

Según se ha especificado al comienzo de la introducción de la memoria, este proyecto está basado en la ingesta y procesamiento de un conjunto de datos sobre premios Nobel. Dado que se trata de un dominio de aplicación con un volumen de datos bastante moderado, ya que dispone de una frecuencia de actualización relativamente baja (anual, o excepcionalmente por eventos históricos), no requiere una infraestructura sobredimensionada propia de aplicaciones de alta carga de procesamiento.

Por tanto, la estimación se ha realizado con vista a la eficiencia de la arquitectura, ajustand la capacidad de los recursos a la carga real esperada del dataset seleccionado.

4.1 Bucket de S3

En este recurso, se ha decidido seleccionar 1GB/mes dado el formato del dataset en uso, siendo el mínimo permitido. Dicho dataset, tiene un peso menor a 1MB, por lo que todo el procesamiento de datos y generación de particiones, no deberían de exceder este límite de almacenamiento.

Además, se reservan 50.000 peticiones del tipo PUT, COPY, POST, LIST y 100.000 de GET y SELECT, en caso de que en ciertas ocasiones la arquitectura presencie más tráfico del esperado.

Con ello, se tendría un presupuesto estimado de 0.31 USD aproximadamente al mes, visible en la Figura 3.

Detailed Estimate

Name	Group	Region	Upfront cost	Monthly cost
Amazon Simple Storage Service (S3)	-	US East (N. Virginia)	0.00 USD	0.31 USD
Status	-			
Description:	-			
Config summary	S3 Standard storage (1 GB per month), PUT, COPY, POST, LIST requests to S3 Standard (50000), GET, SELECT, and all other requests from S3 Standard (100000)			

Figura 3: Estimación del bucket de S3.

4.2 Productor de datos

Este servicio está gestionado en modo "Provisioned", dado que al ser un tráfico predecible y bajo, no se necesita el modo .ºn-Demand"que escala sólo y es más caro para cargas bajas y constantes.

Dado que Firehose consume los datos casi al instante, no se necesita guardar los datos en el stream más que 24 horas. Además, se selecciona un tamaño promedio de registro de 1KB acorde al dataset en uso, y se especifica la transmisión de 10 registros por segundo. Esta decisión fue tomada ya que el modo Provisioned cuenta con un shard, lo cual proporciona una capacidad de hasta 1000 registros/segundo. En cambio, el código del productor (`kinesis.py`) incluye la directiva:

```
# Peque a pausa para simular streaming y no saturar de golpe
time.sleep(0.1)
```

Listing 3: Tiempo de espera para simular streaming. (7.7)

Esto genera que, en consecuencia, se transmitan 10 registros por segundo. De este modo, de este servicio de AWS se estima un precio que ronda los 10.95 USD, tal y como se ve en la Figura 4.

Name	Group	Region	Upfront cost	Monthly cost
Amazon Kinesis Data Streams	-	US East (N. Virginia)	0.00 USD	10.95 USD
Status	-			
Description:	-			
Config summary	Duration of data retention (1 days), Peak number of records (10 per second), Baseline number of records (per second)			

Figura 4: Estimación del productor de datos.

4.3 Consumidor del productor de datos

La estimación de este servicio debe ser coherente con la de S3, donde se especificaron 50000 peticiones. Como Firehose actúa de puente entre el stream (denominado *laureates-stream*) y S3, su volumen de datos será el mismo.

Como se comentó, si se tienen 50000 peticiones donde cada registro tiene un peso de 1KB, se debe especificar que el total de "Number of records for data ingestion"será 50, puesto que "Data records units"se ha fijado en miles (thousands).

Firehose cobra por GB procesado y por cada 1000 registros que deposita en carpetas y, dado que el valor mínimo del tamaño del objeto en el particionado dinámico es de 64 MB, se fija dicha cantidad ya que sobrepasa ligeramente la necesaria para este caso.

Esta configuración genera un costo mensual 0.01 USD, por lo que se piensa que puede haber un error de cálculo dado el precio tan bajo que se ofrece, apreciable en la Figura 5.

Name	Group	Region	Upfront cost	Monthly cost
Amazon Data firehose	-	US East (N. Virginia)	0.00 USD	0.01 USD
Status	-			
Description:	-			
Config summary	Source Type (Direct PUT or Kinesis Data Stream), Dynamic Partitioning (Add On) (Enabled), Data records units (thousands), Record size (1 KB), Data format conversion (optional) (Disabled), Number of records for data ingestion (50 per month), Average size objects delivered (64 MB), JQ Processing (optional) (Disabled), Number of subnets for VPC delivery (0), Average ratio of data processed to VPC vs data ingested (1.3)			

Figura 5: Estimación del consumidor del productor de datos.

4.3.1 Función Lambda

Al ser una función relativamente ligera ya que sólo añade una fecha de procesamiento al registro, no se deben especificar muchos registros. También, cabe destacar que el número de peticiones que espera tener la arquitectura es inferior al número máximo que permite el Free Tier de AWS, por lo que este servicio prácticamente saldría gratuito. Esta información se corrobora con la aportación de la Figura 6:

Name	Group	Region	Upfront cost	Monthly cost
AWS Lambda	-	US East (N. Virginia)	0.00 USD	0.00 USD
Status	-			
Description:	-			
Config summary	Invoke Mode (Buffered), Architecture (x86), Architecture (x86), Number of requests (50000 per month), Amount of ephemeral storage allocated (512 MB)			

Figura 6: Estimación de la función Lambda.

4.4 AWS Glue

Por último, se debe configurar la estimación tanto de los jobs de ETL, como los crawlers.

Por una parte, cada job generado en este proyecto, que son tres, trabajarán con la unidad mínima de workers estándar para un job de Spark en Glue. Deben trabajar con esta medida por el simple hecho de la simplicidad de sus tareas.

Por otra parte, con una ejecución semanal de tres jobs, se tendrían 12 ejecuciones mensuales. A pesar de tardar segundos, Glue cobra un mínimo por unidad de tiempo, siendo ésta de 10 minutos. Por ello, se ha hecho un cálcculo para estimar la duración de ejecución de cada job:

```
3 jobs * 4 ejecuciones/mes * 10 minutos = 120 minutos
```

Además, al disponer de dos crawlers que se ejecutan tras cada ingesta, se ha estimado una duración de 40 minutos por crawler, lo que genera un precio estimado final de 2.35 USD mensuales, tal y como se puede observar en la Figura 7.

Name	Group	Region	Upfront cost	Monthly cost
AWS Glue	-	US East (N. Virginia)	0.00 USD	2.35 USD
Status	-			
Description:	-			
Config summary	Number of DPUs for Apache Spark job (2), Number of DPUs for Python Shell job (0.0625) Number of crawlers (2)			

Figura 7: Estimación de AWS Glue.

Con todo este planteamiento, se obtiene una estimación total como la que se aprecia en la Figura 8, con USD mensuales y USD anuales a pagar.

aws

Contact your AWS representative: Contact Sales

Export Date: 01/16/2026 **Language:** English

Estimate url

Estimate summary

Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	13.62 USD	163.44 USD Includes upfront cost

Figura 8: Estimación total de la arquitectura.

Cabe destacar el archivo .PDF generado con toda la estimación y presupuesto se encuentra adjuntado tanto en la entrega del proyecto, como en el repositorio de GitHub.

5 Conclusiones

Si bien la primera práctica obligatoria de la asignatura consistía en aprender a diseñar y desplegar una aplicación escalable por medio de servicios de AWS, llevar a cabo la realización de esta práctica, colabora de manera directa con el correcto aprendizaje acerca de la ingesta y el procesamiento de datos a gran escala. Tanto, que se han adquirido múltiples conocimientos del ecosistema de la computación en la nube a través de la implementación del ciclo de vida del dato, que va desde su propia ingesta hasta su transformación y almacenamiento final.

Entre los principales recursos con los que se ha trabajado y aprendido durante este proyecto, se manifiesta `Amazon Kinesis` como la herramienta encargada del streaming de datos, permitiendo simular escenarios propios del mercado laboral actual gracias a su alta concurrencia.

Asimismo, se tiene el uso de `AWS Glue`, puesto que se ha demostrado la eficiencia que gana una arquitectura a través de jobs de ETL, procesos capaces de mover y transformar datos desde una o más fuentes hasta un destino final. Y lo más llamativo es que no requiere del aprovisionamiento de una infraestructura física.

También, a pesar de haber conocido este recurso anteriormente, se ha profundizado en el potencial que ofrece Amazon S3, comprobando cómo el particionado dinámico que realiza la función Lambda optimiza enormemente el rendimiento de las consultas posteriores.

Con respecto a las dificultades manifestadas en la realización del proyecto, únicamente mencionar los largos tiempos de espera entre el despliegue de recursos y su disponibilidad efectiva; ya que, por ejemplo, esperar a que un crawler termine conllevaba demasiado tiempo. En especial, cuando por cuestiones de testeo de ciertas funcionalidades de la práctica, se mandaba a ejecutar numerosas veces el código para la detección y tratamiento de errores.

En cuanto a posibles futuras extensiones del trabajo realizado, tal y como se comentó previamente en la memoria, una de las principales metas sería la integración de Amazon Athena para permitir consultas SQL directas sobre los datos procesados, dada su utilidad y potencial exclusivamente en entornos de alta complejidad donde la automatización juega un papel fundamental. Cabe destacar que este servicio no es automatizado por sí mismo, pero su integración de consultas, por ejemplo, con AWS Lambda puede desembocar en una arquitectura bastante eficiente.

6 Referencias y bibliografía

En cuanto a las referencias que han sido utilizadas para llevar a cabo el desarrollo de la práctica, además del uso de la IA registrado en el punto 7.12, se encuentran las siguientes:

- Amazon Web Services. What is Amazon Kinesis Data Streams?
- Amazon Web Services. Develop consumers using Amazon Data Firehose
- Amazon Web Services. Working with jobs in AWS Glue
- Amazon Web Services. AWS Pricing calculator

7 Anexos

A lo largo de la redacción de la presente memoria, se ha detectado que LaTeX genera ciertos problemas con algunos caracteres, como pueden ser las tildes y los emoticonos, impidiendo su correcta presentación. Por tanto, es posible que algunos de los ficheros que se listen a continuación pierdan formato.

No obstante, antes de proceder con los scripts programados, también se adjunta un enlace al repositorio de GitHub en el que se ha estado implementando dicho proyecto, subsanando así cualquier inconveniente o molestia a la hora de revisar el código. También, es importante tener en cuenta que el archivo que conforma el dataset puede ser localizado

`https://github.com/24aymann/CN-P2`

Además de ello, se debe mencionar que los scripts implementados (tanto para el despliegue como la eliminación de los recursos) cubren todo el trabajo necesario para poder ejecutar la práctica. Esto implica que para corregir y testear el funcionamiento de la misma no hace falta más que:

1. Ingresar las credenciales de su cuenta de AWS cuando manda a ejecutar el comando `aws configure`
2. Seguir las indicaciones que encuentra en el fichero README.md (7.10) para configurar su entorno virtual, si lo que se desea es lanzar la arquitectura.
3. Por último, ejecutar el script correspondiente.

Sin más indicaciones que adjuntar, toda la implementación desarrollada en este proyecto se encuentra en la siguiente página, siguiendo el orden de aparición de la carpeta en la que se aloja.

7.1 ETL Job - Nobel Aggregation by Country (nobel_aggregation_by_country.py)

```
"""Job that shows the number of laureates per country and category."""
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, count
from awsglue.dynamicframe import DynamicFrame

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

def main():
    # Get arguments that define the job
    args = getResolvedOptions(sys.argv, ['database', 'table', '
    output_path'])
    database = args['database']
    table = args['table']
    output_path = args['output_path']

    logger.info(f"Database: {database}, Table: {table}, Output: {
    output_path}")

    # Initialize Contexts
    sc = SparkContext()
    glueContext = GlueContext(sc)

    # Read from Glue Catalog through GlueContext
    dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
        database=database,
        table_name=table
    )

    df = dynamic_frame.toDF()

    # Normalize column names
    for col_name in df.columns:
        new_col_name = col_name.lower().replace(" ", "_").strip()
        df = df.withColumnRenamed(col_name, new_col_name)

    # Group by birth country and category
    aggregated_df = df.filter(col("birth_country") != "") \
        .groupBy("birth_country", "category") \
```

```
        .agg(count("*").alias("total_laureates")) \
        .orderBy("birth_country", "category")

    output_dynamic_frame = DynamicFrame.fromDF(aggregated_df,
   glueContext, "output")

    logger.info(f"  Records   added: {aggregated_df.count()}")

    # Write to S3 in Parquet format
    glueContext.write_dynamic_frame.from_options(
        frame=output_dynamic_frame,
        connection_type="s3",
        connection_options={
            "path": output_path,
            "partitionKeys": ["birth_country"]
        },
        format="parquet",
        format_options={"compression": "snappy"}
    )

    logger.info("  Job   completed successfully.")


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

Listing 4: Job 1 - Aggregation by country

7.2 ETL Job - Nobel Aggregation Decadal (`nobel_aggregation_decadal.py`)

```
"""Job that shows the number of laureates per decade and category."""
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, floor, count
from awsglue.dynamicframe import DynamicFrame

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

def main():
    # Get arguments that define the job
    args = getResolvedOptions(sys.argv, ['database', 'table', '
    output_path'])
    database = args['database']
    table = args['table']
    output_path = args['output_path']

    logger.info(f"Database: {database}, Table: {table}, Output: {
    output_path}")

    # Initialize Contexts
    sc = SparkContext()
    glueContext = GlueContext(sc)

    # Read from Glue Catalog through GlueContext
    dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
        database=database,
        table_name=table
    )

    df = dynamic_frame.toDF()
    df_with_decade = df.withColumn("decade", (floor(col("year").cast("
    int") / 10) * 10).cast("string"))

    # Group by decade and category
    aggregated_df = df_with_decade.groupBy("decade", "category") \
        .agg(count("*").alias("total_laureates")) \
        .orderBy("decade", "category")
```

```
    output_dynamic_frame = DynamicFrame.fromDF(aggregated_df,
    glueContext, "output")

    logger.info(f"  Records   added: {aggregated_df.count()}")

    # Write to S3 in Parquet format
    glueContext.write_dynamic_frame.from_options(
        frame=output_dynamic_frame,
        connection_type="s3",
        connection_options={
            "path": output_path,
            "partitionKeys": ["decade"]
        },
        format="parquet",
        format_options={"compression": "snappy"}
    )

    logger.info("  Job   completed successfully.")


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

Listing 5: Job 2 - Aggregation by decade

7.3 ETL Job - Nobel Aggregation Gender (nobel_aggregation_gender.py)

```
"""Job that shows the number of laureates per gender and category."""
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, count
from awsglue.dynamicframe import DynamicFrame

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

def main():
    # Get arguments that define the job
    args = getResolvedOptions(sys.argv, ['database', 'table', '
    output_path'])
    database = args['database']
    table = args['table']
    output_path = args['output_path']

    logger.info(f"Database: {database}, Table: {table}, Output: {
    output_path}")

    # Initialize Contexts
    sc = SparkContext()
    glueContext = GlueContext(sc)

    # Read from Glue Catalog through GlueContext
    dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
        database=database,
        table_name=table
    )

    df = dynamic_frame.toDF()

    # Group by year, category, and sex
    aggregated_df = df.groupBy("year", "category", "sex") \
        .agg(count("*").alias("total_laureates")) \
        .orderBy("year", "category", "sex")

    output_dynamic_frame = DynamicFrame.fromDF(aggregated_df,
    glueContext, "output")
```

```
    logger.info(f"  Records   added: {aggregated_df.count()}")

    # Write to S3 in Parquet format
    glueContext.write_dynamic_frame.from_options(
        frame=output_dynamic_frame,
        connection_type="s3",
        connection_options={
            "path": output_path,
            "partitionKeys": ["year"]
        },
        format="parquet",
        format_options={"compression": "snappy"}
    )

    logger.info("  Job   completed successfully.")


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

Listing 6: Job 3 - Aggregation by gender

7.4 Script de borrado de la arquitectura (delete_resources.sh)

```
#!/bin/bash
# Declaraci n de variables de entorno
AWS_REGION="us-east-1"
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text
    )
BUCKET_NAME="datalake-laureates-${ACCOUNT_ID}"

echo "==========          Iniciando limpieza de recursos...
    =========="
echo "Cuenta AWS:        $ACCOUNT_ID"
echo "Bucket Objetivo:   $BUCKET_NAME"

add_blankspaces() {
    echo ""
    echo "========================================="
    echo ""
}

execute_command() {
    local informative_message="$1"
    local success_message="$2"
    local error_message="$3"
    shift 3

    add_blankspaces
    echo "$informative_message"
    "$@" > /dev/null 2>&1

    if [ $? -eq 0 ]; then
        echo "    $success_message"
    else
        echo "       $error_message"
    fi
}


# ======================================
# ========== SECCI N: GLUE ETL ==========
# ======================================

execute_command "Eliminando el Job de Glue: Nobel Gender Aggregation
    ..." \
                "Job eliminado correctamente (o ya no exist a)." \
                "AVISO: El Job no existe o ya fue eliminado
    previamente." \
```

```
                aws glue delete-job --job-name nobel-gender-
    aggregation

execute_command "Eliminando el Job de Glue: Nobel Decadal Aggregation
    ..." \
                "Job eliminado correctamente (o ya no exist a)." \
                "AVISO: El Job no existe o ya fue eliminado
    previamente." \
                aws glue delete-job --job-name nobel-decadal-
    aggregation

execute_command "Eliminando el Job de Glue: Nobel Country Aggregation
    ..." \
                "Job eliminado correctamente (o ya no exist a)." \
                "AVISO: El Job no existe o ya fue eliminado
    previamente." \
                aws glue delete-job --job-name nobel-country-
    aggregation


# =======================================
# ===== SECCI N: GLUE CRAWLER & DB ======
# =======================================

execute_command "Eliminando el Crawler de Glue..." \
                "El crawler fue eliminado correctamente (o ya no
    exist a)." \
                "AVISO: El Crawler no existe o ya fue eliminado
    previamente." \
                aws glue delete-crawler --name laureates-raw-crawler

execute_command "Eliminando el Crawler de Glue de procesados..." \
                "El crawler de procesados fue eliminado correctamente
    (o ya no exist a)." \
                "AVISO: El Crawler de procesados no existe o ya fue
    eliminado previamente." \
                aws glue delete-crawler --name laureates-processed-
    crawler

execute_command "Eliminando la Glue Database..." \
                "La base de datos fue eliminada correctamente (o ya no
     exist a)." \
                "AVISO: La Base de Datos no existe o ya fue eliminada
    previamente." \
                aws glue delete-database --name laureates_db
```

```
# ======================================
# ========== SECCI N: FIREHOSE ==========
# ======================================

execute_command "Eliminando el Firehose Delivery Stream..." \
                "El Stream de Firehose fue eliminado correctamente (o
    ya no exist a)." \
                "AVISO: El Stream de Firehose no existe o ya fue
    eliminado previamente." \
                aws firehose delete-delivery-stream --delivery-stream-
    name laureates-delivery-stream

execute_command "Eliminando la Lambda Function..." \
                "La Lambda Function fue eliminada correctamente (o ya
    no exist a)." \
                "AVISO: La Lambda Function no existe o ya fue
    eliminada previamente." \
                aws lambda delete-function --function-name laureates-
    firehose-lambda


# ======================================
# ======== SECCI N: Kinesis & S3 ========
# ======================================

execute_command "Eliminando el Kinesis Stream..." \
                "El Stream de Kinesis fue eliminado correctamente (o
    ya no exist a)." \
                "AVISO: El Stream de Kinesis no existe o ya fue
    eliminado previamente." \
                aws kinesis delete-stream --stream-name laureates-
    stream

execute_command "Eliminando el S3 Bucket y todo su contenido..." \
                "El Bucket de S3 fue eliminado correctamente (o ya no
    exist a)." \
                "AVISO: El Bucket de S3 no existe o ya fue eliminado
    previamente." \
                aws s3 rb "s3://$BUCKET_NAME" --force


# ======================================
# ======== SECCI N: LOCAL FILES =========
# ======================================

add_blankspaces
echo "Eliminando los archivos temporales locales..."
```

```
rm -f firehose.zip
echo "  Hecho ! Limpieza completada exitosamente."

add_blankspaces
echo "      L I M P I E Z A   C O M P L E T A D A      "
```

Listing 7: Eliminación de recursos de la arquitectura al completo

7.5 Script de despliegue de la arquitectura (`deploy_resources.sh`)

```
#!/bin/bash
# Declaraci n de variables de entorno
AWS_REGION="us-east-1"
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text
    )
BUCKET_NAME="datalake-laureates-${ACCOUNT_ID}"
ROLE_ARN=$(aws iam get-role --role-name LabRole --query 'Role.Arn' --
    output text)

echo "==========      CONFIGURACI N INICIAL      =========="
echo "1         Regi n de AWS:     $AWS_REGION"
echo "2         Bucket:            $BUCKET_NAME"
echo "3         Role:              $ROLE_ARN"

add_blankspaces() {
    echo ""
    echo "============================================="
    echo ""
}

execute_command() {
    local informative_message="$1"
    local success_message="$2"
    local error_message="$3"
    shift 3

    add_blankspaces
    echo "$informative_message"
    "$@" > /dev/null 2>&1

    if [ $? -eq 0 ]; then
        echo "    $success_message"
    else
        echo "       $error_message"
        exit 1
    fi
}
```

```
# ======================================
# ======== SECCI N: Kinesis & S3 ========
# ======================================

# Crear el bucket de S3
execute_command "Creando el bucket de S3 $BUCKET_NAME..." \
                " Bucket  de S3 creado correctamente!" \
                "AVISO: Ha ocurrido un error al crear el bucket de S3.
    " \
                aws s3 mb s3://$BUCKET_NAME

# Crear carpetas del bucket (objetos vac os con / al final)
execute_command "Creando estructura de carpetas en S3..." \
                "  Estructura  de carpetas creada correctamente!" \
                "AVISO: Ha ocurrido un error al crear la estructura de
     carpetas." \
                bash -c "aws s3api put-object --bucket $BUCKET_NAME --
    key raw/ && \
                 aws s3api put-object --bucket $BUCKET_NAME --key raw/
    laureates/ && \
                 aws s3api put-object --bucket $BUCKET_NAME --key
    processed/ && \
                 aws s3api put-object --bucket $BUCKET_NAME --key
    config/ && \
                 aws s3api put-object --bucket $BUCKET_NAME --key
    scripts/ && \
                 aws s3api put-object --bucket $BUCKET_NAME --key
    queries/ && \
                 aws s3api put-object --bucket $BUCKET_NAME --key
    errors/"

# Crear el stream de Kinesis
execute_command "Creando el stream de Kinesis: laureates-stream..." \
                " Stream  de Kinesis creado correctamente!" \
                "AVISO: Ha ocurrido un error al crear el stream de
    Kinesis." \
                aws kinesis create-stream --stream-name laureates-
    stream --shard-count 1


# ======================================
# ========== SECCI N: FIREHOSE ==========
# ======================================

# Crear el zip de la lambda
```

```
execute_command "Empaquetando la funci n Lambda..." \
                "  Paquete   creado correctamente!" \
                "AVISO: Ha ocurrido un error al empaquetar la funci n
    Lambda." \
                python -c "import zipfile, sys; z=zipfile.ZipFile('
   firehose.zip', 'w'); z.write(sys.argv[1]); z.close()" "firehose.py
   "

# Crear la funci n lambda
execute_command "Creando la funci n Lambda: laureates-firehose-lambda
   ..." \
                "   Funcin    Lambda creada correctamente!" \
                "AVISO: Ha ocurrido un error al crear la funci n
   Lambda." \
                aws lambda create-function \
                    --function-name laureates-firehose-lambda \
                    --runtime python3.12 \
                    --role "$ROLE_ARN" \
                    --handler firehose.lambda_handler \
                    --zip-file fileb://firehose.zip \
                    --timeout 60 \
                    --memory-size 128

# Actualizar la funci n lambda
execute_command "Actualizando el c digo de la funci n Lambda..." \
                "   Cdigo    de Lambda actualizado correctamente!" \
                "AVISO: Ha ocurrido un error al actualizar el c digo
   de Lambda." \
                aws lambda update-function-code \
                    --function-name laureates-firehose-lambda \
                    --zip-file fileb://firehose.zip

# Obtener el ARN de la funci n lambda
LAMBDA_ARN=$(aws lambda get-function --function-name laureates-
   firehose-lambda --query 'Configuration.FunctionArn' --output text)
if [ -z "$LAMBDA_ARN" ]; then
    echo "AVISO: Ha ocurrido un error al obtener el ARN de la funci n
    Lambda."
    exit 1
fi

# Crear el delivery stream
execute_command "Creando Firehose Delivery Stream..." \
                "  Firehose   Delivery Stream creado correctamente!" \
                "AVISO: Ha ocurrido un error al crear el Firehose
   Delivery Stream." \
                aws firehose create-delivery-stream \
```

```
                    --delivery-stream-name laureates-delivery-stream \
                    --delivery-stream-type KinesisStreamAsSource \
                    --kinesis-stream-source-configuration "
    KinesisStreamARN=arn:aws:kinesis:$AWS_REGION:$ACCOUNT_ID"":stream/
    laureates-stream,RoleARN=$ROLE_ARN" \
                    --extended-s3-destination-configuration "{
                \"BucketARN\": \"arn:aws:s3:::$BUCKET_NAME\",
                \"RoleARN\": \"$ROLE_ARN\",
                \"Prefix\": \"raw/laureates/processing_date=!{
    partitionKeyFromLambda:processing_date}/\",
                \"ErrorOutputPrefix\": \"errors/!{firehose:error-
    output-type}/\",
                \"BufferingHints\": { \"SizeInMBs\": 64, \"
    IntervalInSeconds\": 60 },
                \"DynamicPartitioningConfiguration\": { \"Enabled\":
    true, \"RetryOptions\": { \"DurationInSeconds\": 300 } },
                \"ProcessingConfiguration\": { \"Enabled\": true, \"
    Processors\": [
                        { \"Type\": \"Lambda\", \"Parameters\": [
                                { \"ParameterName\": \"LambdaArn\", \"
    ParameterValue\": \"$LAMBDA_ARN\" },
                                { \"ParameterName\": \"BufferSizeInMBs
    \", \"ParameterValue\": \"1\" },
                                { \"ParameterName\": \"
    BufferIntervalInSeconds\", \"ParameterValue\": \"60\" }
                            ]
                        }
                    ]
                }
            }"


# =======================================
# ============ SECCI N: GLUE ============
# =======================================

# Crear la base de datos de Glue
execute_command "Creando la base de datos de Glue: laureates_db..." \
                " Base  de datos de Glue creada correctamente!" \
                "AVISO: Ha ocurrido un error al crear la base de datos
     de Glue." \
                aws glue create-database --database-input "{\"Name\":\
    "laureates_db\"}"

# Crear el crawler de Glue
execute_command "Creando el crawler de Glue: laureates-raw-crawler..."
    \
```

```
                " Glue  Crawler creado correctamente!" \
                "AVISO: Ha ocurrido un error al crear el crawler de
    Glue." \
                aws glue create-crawler \
                    --name laureates-raw-crawler \
                    --role "$ROLE_ARN" \
                    --database-name laureates_db \
                    --targets "{\"S3Targets\": [{\"Path\": \"s3://
    $BUCKET_NAME/raw/laureates\"}]}"

# Crear el crawler de Glue para datos procesados
execute_command "Creando el crawler de Glue para procesados: laureates
    -processed-crawler..." \
                " Glue  Crawler de procesados creado correctamente!" \
                "AVISO: Ha ocurrido un error al crear el crawler de
    Glue para procesados." \
                aws glue create-crawler \
                    --name laureates-processed-crawler \
                    --role "$ROLE_ARN" \
                    --database-name laureates_db \
                    --targets "{\"S3Targets\": [{\"Path\": \"s3://
    $BUCKET_NAME/processed/\"}]}"

add_blankspaces
echo "Ejecutando el producer de Kinesis (tardar  unos momentos)..."
python kinesis.py
if [[ $? -eq 0 ]]; then
    echo "     Datos  enviados a Kinesis!"
else
    echo "       AVISO: Ha ocurrido un error al ejecutar el producer
    de Kinesis."
    exit 1
fi
sleep 60

execute_command "Iniciando el crawler 'laureates-raw-crawler'..." \
                " Glue  Crawler iniciado!" \
                "AVISO: Ha ocurrido un error al iniciar el crawler de
    Glue." \
                aws glue start-crawler --name laureates-raw-crawler


# ========================================
# ========== SECCI N: GLUE ETL ==========
# ========================================

# Subir los scripts de ETL a S3
```

```
execute_command "Subiendo scripts ETL a S3..." \
                "  Scripts   ETL subidos correctamente!" \
                "AVISO: Ha ocurrido un error al subir scripts ETL." \
                bash -c "aws s3 cp jobs/nobel_aggregation_gender.py s3
    ://$BUCKET_NAME/scripts/ && \
                 aws s3 cp jobs/nobel_aggregation_decadal.py s3://
    $BUCKET_NAME/scripts/ && \
                 aws s3 cp jobs/nobel_aggregation_by_country.py s3://
    $BUCKET_NAME/scripts/"

# Variables de entorno
DATABASE="laureates_db"
TABLE="laureates"
GENDER_OUTPUT="s3://$BUCKET_NAME/processed/laureates_gender/"
DECADAL_OUTPUT="s3://$BUCKET_NAME/processed/laureates_decadal/"
COUNTRY_OUTPUT="s3://$BUCKET_NAME/processed/laureates_country/"

# Creaci n de los Jobs de Glue
execute_command "  Creando   primer Job Glue! ---> 1         Nobel Gender
     Aggregation..." \
                " Job  Nobel Gender Aggregation creado correctamente!"
    \
                "AVISO: Ha ocurrido un error al crear el Job nobel-
    gender-aggregation." \
                aws glue create-job \
                    --name nobel-gender-aggregation \
                    --role "$ROLE_ARN" \
                    --command "{
                        \"Name\": \"glueetl\",
                        \"ScriptLocation\": \"s3://$BUCKET_NAME/
    scripts/nobel_aggregation_gender.py\",
                        \"PythonVersion\": \"3\"
                    }" \
                    --default-arguments "{
                        \"--database\": \"$DATABASE\",
                        \"--table\": \"$TABLE\",
                        \"--output_path\": \"s3://$BUCKET_NAME/
    processed/laureates_by_gender/\",
                        \"--enable-continuous-cloudwatch-log\": \"true
    \",
                        \"--spark-event-logs-path\": \"s3://
    $BUCKET_NAME/logs/\"
                    }" \
                    --glue-version "4.0" \
                    --number-of-workers 2 \
                    --worker-type "G.1X"
```

```
execute_command "  Creando   segundo  Job  Glue! ---> 2         Nobel
    Decadal  Aggregation..." \
                "  Job   Nobel  Decadal  Aggregation  creado  correctamente!
    " \
                "AVISO: Ha ocurrido un error al crear el Job nobel-
    decadal-aggregation." \
                aws glue create-job \
                    --name nobel-decadal-aggregation \
                    --role "$ROLE_ARN" \
                    --command "{
                        \"Name\": \"glueetl\",
                        \"ScriptLocation\": \"s3://$BUCKET_NAME/
    scripts/nobel_aggregation_decadal.py\",
                        \"PythonVersion\": \"3\"
                    }" \
                    --default-arguments "{
                        \"--database\": \"$DATABASE\",
                        \"--table\": \"$TABLE\",
                        \"--output_path\": \"s3://$BUCKET_NAME/
    processed/laureates_by_decadal/\",
                        \"--enable-continuous-cloudwatch-log\": \"true
    \",
                        \"--spark-event-logs-path\": \"s3://
    $BUCKET_NAME/logs/\"
                    }" \
                    --glue-version "4.0" \
                    --number-of-workers 2 \
                    --worker-type "G.1X"

execute_command "  Creando   tercer  Job  Glue! ---> 3         Nobel
    Country  Aggregation..." \
                "  Job   Nobel  Country  Aggregation  creado  correctamente!
    " \
                "AVISO: Ha ocurrido un error al crear el Job nobel-
    country-aggregation." \
                aws glue create-job \
                    --name nobel-country-aggregation \
                    --role "$ROLE_ARN" \
                    --command "{
                        \"Name\": \"glueetl\",
                        \"ScriptLocation\": \"s3://$BUCKET_NAME/
    scripts/nobel_aggregation_by_country.py\",
                        \"PythonVersion\": \"3\"
                    }" \
                    --default-arguments "{
                        \"--database\": \"$DATABASE\",
                        \"--table\": \"$TABLE\",
```

```
                        \"--output_path\": \"s3://$BUCKET_NAME/
    processed/laureates_by_country/\",
                        \"--enable-continuous-cloudwatch-log\": \"true
    \",
                        \"--spark-event-logs-path\": \"s3://
    $BUCKET_NAME/logs/\"
                    }" \
                    --glue-version "4.0" \
                    --number-of-workers 2 \
                    --worker-type "G.1X"

add_blankspaces
echo "Esperando a que el crawler 'laureates-raw-crawler' termine..."
while true; do
    CRAWLER_STATE=$(aws glue get-crawler --name laureates-raw-crawler
    --query "Crawler.State" --output text)
    echo "Estado del Crawler: $CRAWLER_STATE"
    echo ""

    if [ "$CRAWLER_STATE" == "READY" ]; then
        echo "Crawler finalizado."
        break
    fi

    sleep 30
done

# Ejecuci n secuencial de los Jobs de Glue
for JOB_NAME in "nobel-gender-aggregation" "nobel-decadal-aggregation"
    "nobel-country-aggregation"; do
    execute_command "Iniciando el job de Glue $JOB_NAME..." \
                    " El  job de Glue $JOB_NAME fue iniciado
    correctamente!" \
                    "AVISO: Ha ocurrido un error al iniciar el job de
    Glue $JOB_NAME." \
                    aws glue start-job-run --job-name "$JOB_NAME"

    echo "Esperando a que el job $JOB_NAME finalice..."
    echo ""
    while true; do
        STATUS=$(aws glue get-job-runs --job-name "$JOB_NAME" --max-
    items 1 --query "JobRuns[0].JobRunState" --output text)
        echo "Estado de $JOB_NAME ---> $STATUS"

        if [ "$STATUS" == "SUCCEEDED" ]; then
            echo "    Job $JOB_NAME finalizado con  xito ."
            echo ""
```

```
            break
        elif [[ "$STATUS" =~ (FAILED|TIMEOUT|STOPPED) ]]; then
            echo "    El Job $JOB_NAME ha fallado o se ha detenido.
    Estado: $STATUS"
            exit 1
        fi

        sleep 30
    done
done

execute_command "Iniciando el crawler de procesados 'laureates-
    processed-crawler'..." \
                " Glue  Crawler de procesados iniciado!" \
                "AVISO: Ha ocurrido un error al iniciar el crawler de
    procesados." \
                aws glue start-crawler --name laureates-processed-
    crawler

echo "Esperando a que el crawler 'laureates-processed-crawler' termine
    ..."
sleep 25
while true; do
    CRAWLER_STATE=$(aws glue get-crawler --name laureates-processed-
    crawler --query "Crawler.State" --output text)
    echo "Estado del Crawler de procesados: $CRAWLER_STATE"
    echo ""

    if [ "$CRAWLER_STATE" == "READY" ]; then
        echo "    Crawler de procesados finalizado."
        break
    fi

    sleep 30
done

add_blankspaces
echo "      F I N   D E L   D E S P L I E G U E      "
echo ""
```

Listing 8: Despliegue de la arquitectura completa

7.6 Consumidor del stream de datos (`firehose.py`)

```
import json
import base64
import datetime

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

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

        # Add processing timestamp
        processing_time = datetime.datetime.now(datetime.timezone.utc)

        # Create the partition key (YYYY-MM-DD format)
        partition_date = processing_time.strftime('%Y-%m-%d')

        output_record = {
            'recordId': record['recordId'],
            'result': 'Ok',
            'data': base64.b64encode((json.dumps(data_json) + '\n').
   encode('utf-8')).decode('utf-8'),
            'metadata': {
                'partitionKeys': {
                    'processing_date': partition_date
                }
            }
        }

        output.append(output_record)

    return {'records': output}
```

Listing 9: Consumidor del stream de datos

7.7 Productor de datos (kinesis.py)

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

# VARIABLES DE CONFIGURACI N
INPUT_FILE =    'nobel_laureates.json'
STREAM_NAME =   'laureates-stream'
REGION_NAME =   'us-east-1'

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

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

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

    logger.info(f"    Iniciando transmisi n al stream: {STREAM_NAME
   }...")

    # Se itera sobre todos los laureados
    for laureate in laureates_list:
        category = laureate.get('Category', 'Unknown')

        # Se env a a Kinesis el registro completo
        kinesis.put_record(
            StreamName=STREAM_NAME,
            Data=json.dumps(laureate),
            PartitionKey=category
        )

        records_sent += 1

        # Informaci n para el log
        laureate_name = laureate.get('Full Name', 'Unknown')
        year = laureate.get('Year', 'N/A')

        # Indicador de registro enviado, con salida formateada
        logger.info(f"( {year} - {category} )".ljust(23) + f"===>
        {laureate_name}     ")

        # Peque a pausa para simular streaming y no saturar de golpe
```

```
        time.sleep(0.1)

    logger.info("")
    logger.info("   La    transmisi n ha finalizado!   ")
    logger.info(f"Total de registros enviados: {records_sent}")


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

Listing 10: Productor de datos

7.8 Función main (`main.py`)

```
def main():
    print("Hello from cn-p2!")


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

Listing 11: El main de la práctica

7.9 Versión y dependencias necesarias del proyecto (`pyproject.toml`)

```
[project]
name = "cn-p2"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.13"
dependencies = [
    "boto3>=1.42.6",
    "loguru>=0.7.3",
]
```

Listing 12: Fichero con versión y dependencias necesarias

7.10 README.md elaborado para la preparación del entorno

```
<div align="center">
  <img src="https://www.eii.ulpgc.es/sites/default/files/eii-acron-mod
    .png"
       alt="Logo ULPGC"
       width="400"
       style="margin-bottom: 10px;"
   >
</div>

<h1 align="center">Pr ctica Obligatoria 2 - Ingesta y procesamiento
   de datos en AWS</h1>

<div align="center" style="font-family: 'Segoe UI', sans-serif; line-
   height: 1.6; margin-top: 30px;">
  <h2 style="font-size: 28px; margin-bottom: 10px;">
    Grado en Ingenier a Inform tica
  </h2>
  <h3 style="font-size: 24px; margin-bottom: 10px;">
    Computaci n en la Nube
  </h3>
</div>

**Autor:** Ayman Asbai Ghoudan

**Curso acad mico:** 2025/26

---

### Preparaci n del Entorno con `uv`

Los comandos que se observan a continuaci n crear n el proyecto
   correspondiente, instalar n las dependencias necesarias para su
   ejecuci n y activar n el entorno virtual.

**Crear proyecto e instalar dependencias:**

```bash
# 1. inicializa el proyecto
uv init

# 2. Instalar boto3 (la librer a de AWS para Python)
uv add boto3
uv add loguru
```
```

```
# 3. Crear el entorno virtual (si no se cre  autom ticamente con
    init)
uv venv
‘‘‘

**Activar el entorno virtual:**
En este caso, se decidi  utilizar la versi n de *Linux*, pero
    tambi n se adjuntan los procedimientos para las posibles
    alternativas.

- **Linux / macOS:**

  ‘‘‘bash
  source .venv/bin/activate
  ‘‘‘

- **Windows (PowerShell):**

  ‘‘‘powershell
  .venv\Scripts\activate
  ‘‘‘

- **Windows (CMD):**

  ‘‘‘cmd
  .venv\Scripts\activate.bat
  ‘‘‘

---
```

Listing 13: Presentación de la práctica y configuración del entorno

7.11 Lockfile de las dependencias en uso

```
version = 1
revision = 3
requires-python = ">=3.13"

[[package]]
name = "boto3"
version = "1.42.6"
source = { registry = "https://pypi.org/simple" }
dependencies = [
    { name = "botocore" },
    { name = "jmespath" },
    { name = "s3transfer" },
]
```

```
sdist = { url = "https://files.pythonhosted.org/packages/06/a8/
    e7a408127d61569df097d6c128775ffa3e609a023d4461686fd85fe5eef4/boto3
    -1.42.6.tar.gz", hash = "sha256:11
    dab889a24f378af6c93afd4aa06d7cace3866cbf02e78c7a77e9a7fb41967a",
    size = 112859, upload-time = "2025-12-09T23:00:33.685Z" }
wheels = [
    { url = "https://files.pythonhosted.org/packages/ee/dd/
    af36b27e9fc004cd8ae2290d6d76a8de34d098d17a3718ae875e5e856ab1/boto3
    -1.42.6-py3-none-any.whl", hash = "sha256:69
    ff5cf6431fe7870da009f23aceabb20d56b4c9852ba9a808eaf6cc30ae02a5",
    size = 140574, upload-time = "2025-12-09T23:00:31.355Z" },
]

[[package]]
name = "botocore"
version = "1.42.6"
source = { registry = "https://pypi.org/simple" }
dependencies = [
    { name = "jmespath" },
    { name = "python-dateutil" },
    { name = "urllib3" },
]
sdist = { url = "https://files.pythonhosted.org/packages/a6/28/
    e1ad336dd409e3cde0644cbba644056248e873b77129502f81581f5a222f/
    botocore-1.42.6.tar.gz", hash = "sha256:
    ab389c6874dfbdc4c18de9b4a02d300cb6c7f6f2d4622c73e5965aeef80e570d",
    size = 14851572, upload-time = "2025-12-09T23:00:21.993Z" }
wheels = [
    { url = "https://files.pythonhosted.org/packages/6c/1e/545
    d3ca599aea7a7aaad23735ae2d1c7280557e115086208d68af1621f93/botocore
    -1.42.6-py3-none-any.whl", hash = "sha256:
    c4aebdc391f3542270ebea8b8f0060fde514f6441de207dce862ed759887607e",
    size = 14527177, upload-time = "2025-12-09T23:00:17.197Z" },
]

[[package]]
name = "cn-p2"
version = "0.1.0"
source = { virtual = "." }
dependencies = [
    { name = "boto3" },
    { name = "loguru" },
]

[package.metadata]
requires-dist = [
    { name = "boto3", specifier = ">=1.42.6" },
```

```
    { name = "loguru", specifier = ">=0.7.3" },
]

[[package]]
name = "colorama"
version = "0.4.6"
source = { registry = "https://pypi.org/simple" }
sdist = { url = "https://files.pythonhosted.org/packages/d8/53/6
   f443c9a4a8358a93a6792e2acffb9d9d5cb0a5cfd8802644b7b1c9a02e4/
   colorama-0.4.6.tar.gz", hash = "sha256:08695
   f5cb7ed6e0531a20572697297273c47b8cae5a63ffc6d6ed5c201be6e44", size
    = 27697, upload-time = "2022-10-25T02:36:22.414Z" }
wheels = [
    { url = "https://files.pythonhosted.org/packages/d1/d6/3965
   ed04c63042e047cb6a3e6ed1a63a35087b6a609aa3a15ed8ac56c221/colorama
   -0.4.6-py2.py3-none-any.whl", hash = "sha256:4
   f1d9991f5acc0ca119f9d443620b77f9d6b33703e51011c16baf57afb285fc6",
   size = 25335, upload-time = "2022-10-25T02:36:20.889Z" },
]

[[package]]
name = "jmespath"
version = "1.0.1"
source = { registry = "https://pypi.org/simple" }
sdist = { url = "https://files.pythonhosted.org/packages/00/2a/
   e867e8531cf3e36b41201936b7fa7ba7b5702dbef42922193f05c8976cd6/
   jmespath-1.0.1.tar.gz", hash = "sha256:90261
   b206d6defd58fdd5e85f478bf633a2901798906be2ad389150c5c60edbe", size
    = 25843, upload-time = "2022-06-17T18:00:12.224Z" }
wheels = [
    { url = "https://files.pythonhosted.org/packages/31/b4/
   b9b800c45527aadd64d5b442f9b932b00648617eb5d63d2c7a6587b7cafc/
   jmespath-1.0.1-py3-none-any.whl", hash = "sha256:02
   e2e4cc71b5bcab88332eebf907519190dd9e6e82107fa7f83b1003a6252980",
   size = 20256, upload-time = "2022-06-17T18:00:10.251Z" },
]

[[package]]
name = "loguru"
version = "0.7.3"
source = { registry = "https://pypi.org/simple" }
dependencies = [
    { name = "colorama", marker = "sys_platform == 'win32'" },
    { name = "win32-setctime", marker = "sys_platform == 'win32'" },
]
sdist = { url = "https://files.pythonhosted.org/packages/3a/05/
   a1dae3dffd1116099471c643b8924f5aa6524411dc6c63fdae648c4f1aca/
```

```
    loguru-0.7.3.tar.gz", hash = "sha256:19480589
    e77d47b8d85b2c827ad95d49bf31b0dcde16593892eb51dd18706eb6", size =
    63559, upload-time = "2024-12-06T11:20:56.608Z" }
wheels = [
    { url = "https://files.pythonhosted.org/packages/0c/29/0348
    de65b8cc732daa3e33e67806420b2ae89bdce2b04af740289c5c6c8c/loguru
    -0.7.3-py3-none-any.whl", hash = "sha256:31
    a33c10c8e1e10422bfd431aeb5d351c7cf7fa671e3c4df004162264b28220c",
    size = 61595, upload-time = "2024-12-06T11:20:54.538Z" },
]

[[package]]
name = "python-dateutil"
version = "2.9.0.post0"
source = { registry = "https://pypi.org/simple" }
dependencies = [
    { name = "six" },
]
sdist = { url = "https://files.pythonhosted.org/packages/66/c0/0
    c8b6ad9f17a802ee498c46e004a0eb49bc148f2fd230864601a86dcf6db/python
    -dateutil-2.9.0.post0.tar.gz", hash = "sha256:37
    dd54208da7e1cd875388217d5e00ebd4179249f90fb72437e91a35459a0ad3",
    size = 342432, upload-time = "2024-03-01T18:36:20.211Z" }
wheels = [
    { url = "https://files.pythonhosted.org/packages/ec/57/56
    b9bcc3c9c6a792fcbaf139543cee77261f3651ca9da0c93f5c1221264b/
    python_dateutil-2.9.0.post0-py2.py3-none-any.whl", hash = "sha256:
    a8b2bc7bffae282281c8140a97d3aa9c14da0b136dfe83f850eea9a5f7470427",
    size = 229892, upload-time = "2024-03-01T18:36:18.57Z" },
]

[[package]]
name = "s3transfer"
version = "0.16.0"
source = { registry = "https://pypi.org/simple" }
dependencies = [
    { name = "botocore" },
]
sdist = { url = "https://files.pythonhosted.org/packages/05/04/74127
    fc843314818edfa81b5540e26dd537353b123a4edc563109d8f17dd/s3transfer
    -0.16.0.tar.gz", hash = "sha256:8
    e990f13268025792229cd52fa10cb7163744bf56e719e0b9cb925ab79abf920",
    size = 153827, upload-time = "2025-12-01T02:30:59.114Z" }
wheels = [
    { url = "https://files.pythonhosted.org/packages/fc/51/727
    abb13f44c1fcf6d145979e1535a35794db0f6e450a0cb46aa24732fe2/
    s3transfer-0.16.0-py3-none-any.whl", hash = "sha256:18
```

```
    e25d66fed509e3868dc1572b3f427ff947dd2c56f844a5bf09481ad3f3b2fe",
    size = 86830, upload-time = "2025-12-01T02:30:57.729Z" },
]

[[package]]
name = "six"
version = "1.17.0"
source = { registry = "https://pypi.org/simple" }
sdist = { url = "https://files.pythonhosted.org/packages/94/e7/
    b2c673351809dca68a0e064b6af791aa332cf192da575fd474ed7d6f16a2/six
    -1.17.0.tar.gz", hash = "sha256:
    ff70335d468e7eb6ec65b95b99d3a2836546063f63acc5171de367e834932a81",
     size = 34031, upload-time = "2024-12-04T17:35:28.174Z" }
wheels = [
     { url = "https://files.pythonhosted.org/packages/b7/ce/149
    a00dd41f10bc29e5921b496af8b574d8413afcd5e30dfa0ed46c2cc5e/six
    -1.17.0-py2.py3-none-any.whl", hash = "sha256:4721
    f391ed90541fddacab5acf947aa0d3dc7d27b2e1e8eda2be8970586c3274",
    size = 11050, upload-time = "2024-12-04T17:35:26.475Z" },
]

[[package]]
name = "urllib3"
version = "2.6.1"
source = { registry = "https://pypi.org/simple" }
sdist = { url = "https://files.pythonhosted.org/packages/5e/1d/0
    f3a93cca1ac5e8287842ed4eebbd0f7a991315089b1a0b01c7788aa7b63/
    urllib3-2.6.1.tar.gz", hash = "sha256:5379
    eb6e1aba4088bae84f8242960017ec8d8e3decf30480b3a1abdaa9671a3f",
    size = 432678, upload-time = "2025-12-08T15:25:26.773Z" }
wheels = [
     { url = "https://files.pythonhosted.org/packages/bc/56/190
    ceb8cb10511b730b564fb1e0293fa468363dbad26145c34928a60cb0c/urllib3
    -2.6.1-py3-none-any.whl", hash = "sha256:
    e67d06fe947c36a7ca39f4994b08d73922d40e6cca949907be05efa6fd75110b",
     size = 131138, upload-time = "2025-12-08T15:25:25.51Z" },
]

[[package]]
name = "win32-setctime"
version = "1.2.0"
source = { registry = "https://pypi.org/simple" }
sdist = { url = "https://files.pythonhosted.org/packages/b3/8f/705086
    c9d734d3b663af0e9bb3d4de6578d08f46b1b101c2442fd9aecaa2/
    win32_setctime-1.2.0.tar.gz", hash = "sha256:
    ae1fdf948f5640aae05c511ade119313fb6a30d7eabe25fef9764dca5873c4c0",
     size = 4867, upload-time = "2024-12-07T15:28:28.314Z" }
```

```
wheels = [
    { url = "https://files.pythonhosted.org/packages/e1/07/
    c6fe3ad3e685340704d314d765b7912993bcb8dc198f0e7a89382d37974b/
    win32_setctime-1.2.0-py3-none-any.whl", hash = "sha256:95
    d644c4e708aba81dc3704a116d8cbc974d70b3bdb8be1d150e36be6e9d1390",
    size = 4083, upload-time = "2024-12-07T15:28:26.465Z" },
]
```

Listing 14: Fichero que congela todas las dependencias del proyecto

7.12 Uso de la Inteligencia Artificial

Este apartado está planteado para definir aquellas facetas de la práctica en las que se haya beneficiado en mayor o menor medida de la Inteligencia Artificial.

En primer lugar, en los comienzos de realización del proyecto, se hizo uso de la misma para una explicación breve del funcionamiento de cada uno de los recursos de AWS que se manifiestan en este documento. Esto se debe a la necesidad de tener referencias teóricas que explicasen los servicios utilizados, puesto que tras la última sesión de teoría, éstos no fueron publicados.

En segundo lugar, los scripts de despliegue y eliminación de recursos de la arquitectura fueron ligeramente apoyados por la IA, especialmente en análisis de estado de los crawlers y jobs de Glue.

Además de ello, el dataset seleccionado para este proyecto surge de una consulta a esta herramienta. En dicha consulta, se adjuntó el conjunto de datos de consumo energético proporcionado en el proyecto base y se solicitó un dataset con condiciones similares, respetando el límite de tener más de 864 registros.

Por último, también se solicitó formatear la salida que se genera en la terminal con los registros que han sido enviados a Kinesis. Dicho formateo se presenta de la siguiente manera:

```
# Indicador de registro enviado, con salida formateada
logger.info(f"( {year} - {category} )".ljust(23) + f"===>          {
    laureate_name}        ")
```

Listing 15: Mensaje de log generado en el envío de datos