Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería Informática

Grado en Ingeniería Informática

Memoria de la Práctica Obligatoria 2

Ingesta y procesamiento del Ranking Nacional de Vóley Playa

Asignatura: Computación en la Nube
Curso académico: 2025–2026
Alumno: Miguel Castellano Hernández
Fecha de entrega: 16 de enero de 2026

Índice

1 Introducción

Esta memoria describe la implementación de un flujo de datos en AWS para el análisis del **Ranking Nacional de Vóley Playa Masculino de España**. Se procesan registros únicos por jugador que representan su puntuación total acumulada. El objetivo es categorizar el nivel de los deportistas y agregar métricas por clubes mediante una arquitectura escalable que utiliza **S3, Kinesis y AWS Glue**.

2 Desarrollo de las actividades

2.1 Configuración de la Infraestructura de Almacenamiento

La base del proyecto reside en Amazon S3, configurado como un Data Lake estructurado. Para garantizar la organización y escalabilidad, se ha implementado una jerarquía de prefijos (carpetas virtuales) que segmentan los datos según su estado de procesamiento y propósito.

Como se observa en la **Figura 1**, la estructura incluye el directorio `raw/` para la ingesta masiva sin procesar y `processed/` para los resultados finales tras la ejecución de los Jobs de Glue.

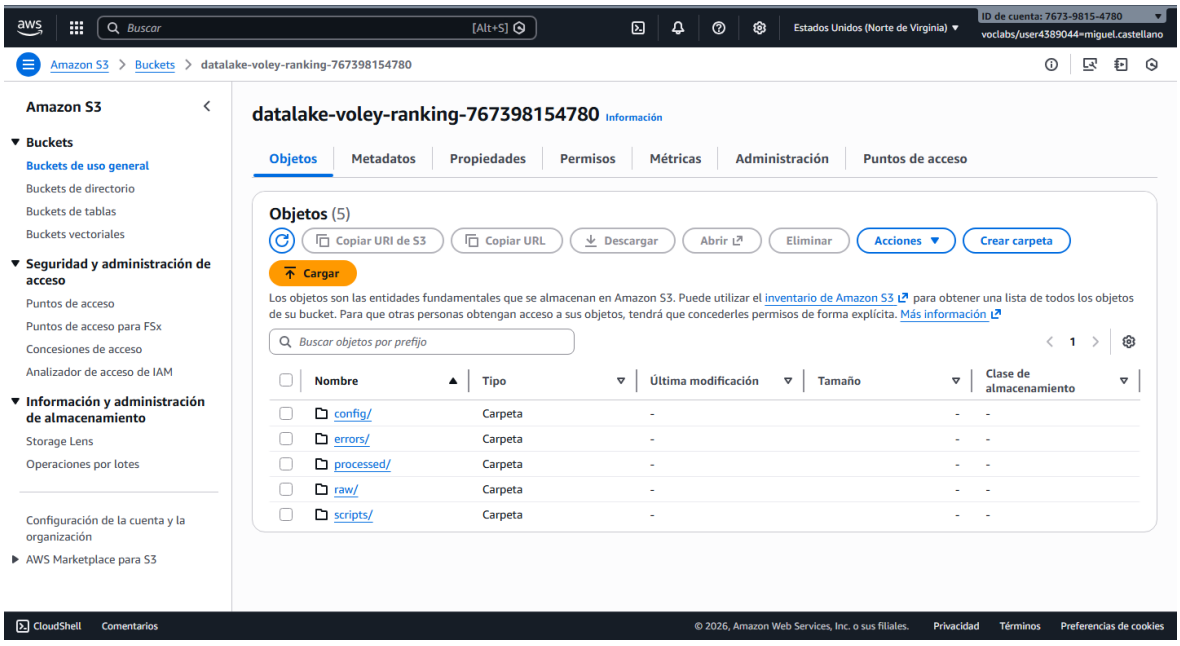

Figura 1: Estructura de directorios en Amazon S3 para la gestión del ciclo de vida de los datos

2.2 Ingesta de Datos en Tiempo Real (Kinesis)

Para la captura del ranking, se desplegó un Amazon Kinesis Data Stream denominado `beach-voley-national-ranking`. Este componente actúa como el punto de entrada de alta disponibilidad para los registros de los jugadores.

La **Figura 2** muestra el flujo en estado activo, operando en modo aprovisionado. En el visor de datos se confirma la llegada de registros JSON que contienen los campos críticos: `IdPersona`, `Puntos` y `EquipoVoleyPlaya`.

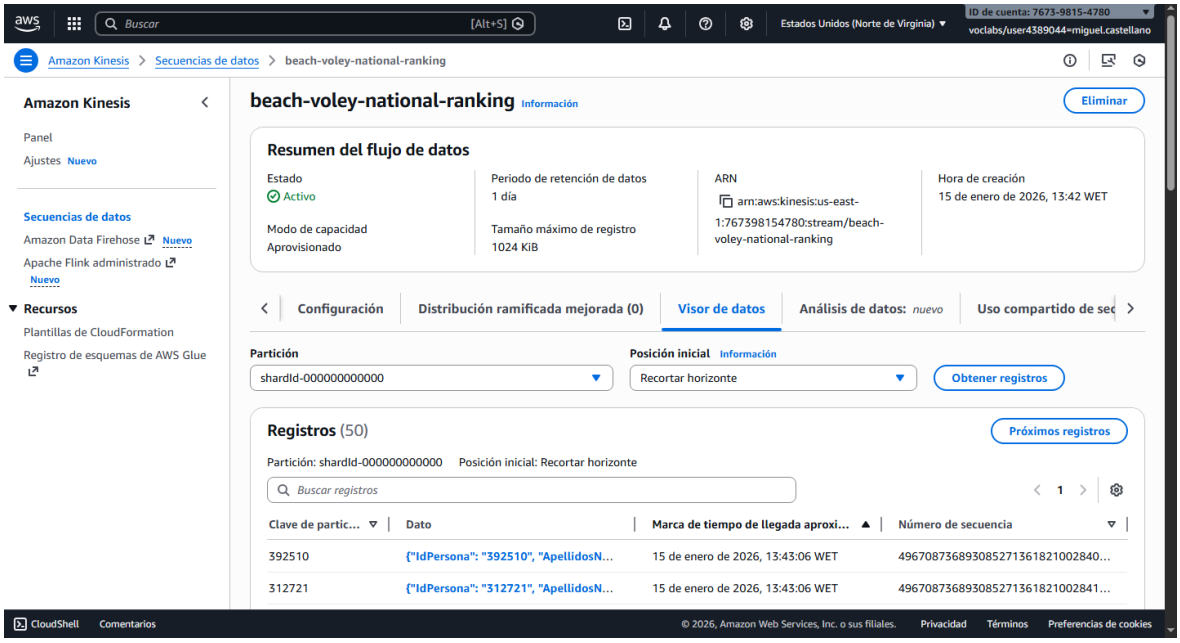

Figura 2: Monitorización de la secuencia de datos y validación de registros de entrada en Kinesis

2.3 Transformación y Transporte (Lambda y Firehose)

Se utilizó Amazon Data Firehose acoplado a una función AWS Lambda para la fase de transformación intermedia.

Como se detalla en el código de la **Figura 3**, la función Lambda realiza una limpieza de los registros e inyecta metadatos de particionamiento (`partitionKeys`).

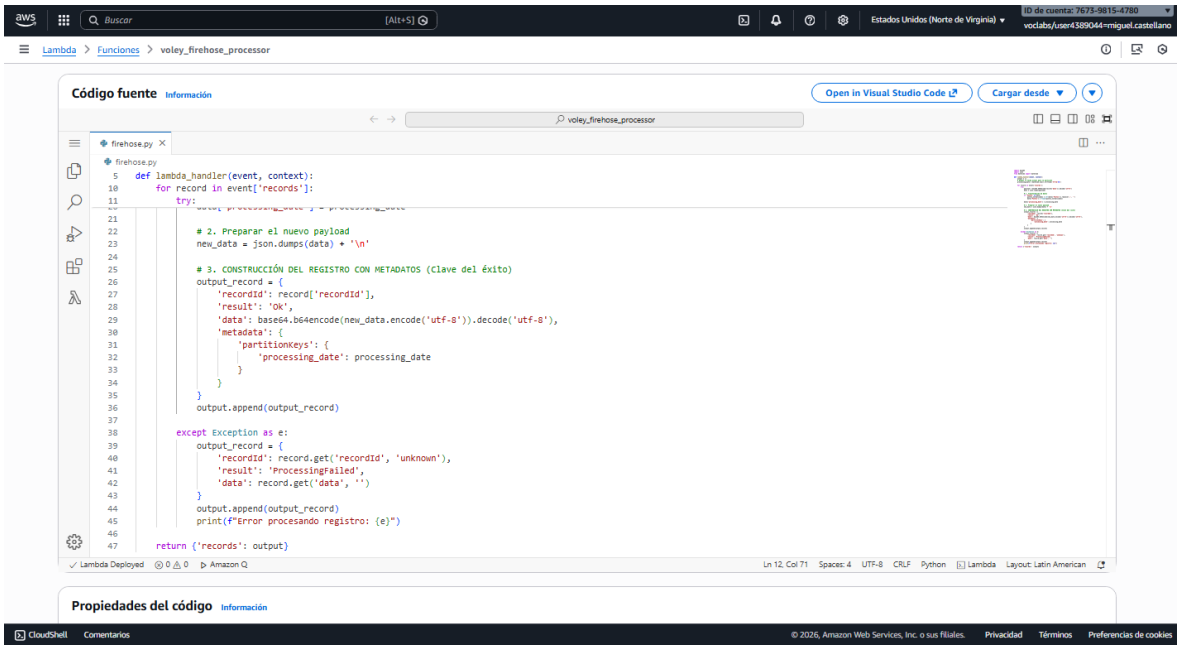

Figura 3: Lógica de transformación en AWS Lambda para el particionamiento dinámico de registros

El flujo final de entrega se visualiza en la **Figura 4**, donde el servicio Firehose actúa como puente entre Kinesis y S3. El resultado exitoso de esta integración se evidencia en la **Figura 5**, donde los datos aparecen correctamente particionados por prefijos temporales.

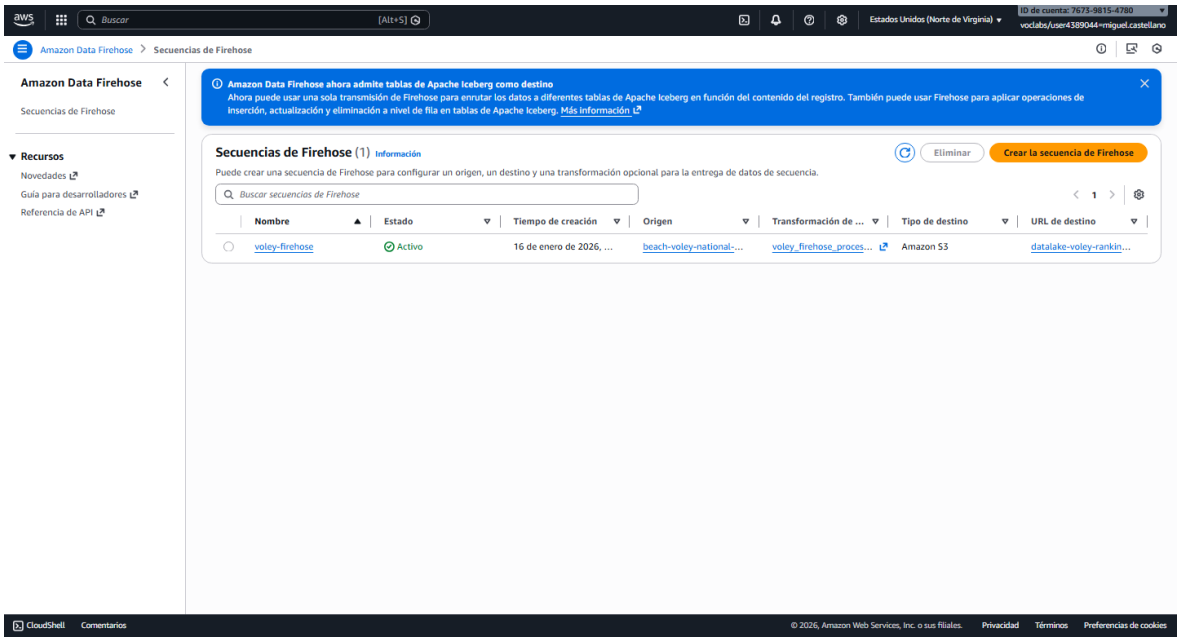

Figura 4: Configuración de la secuencia de Amazon Data Firehose para el transporte automatizado

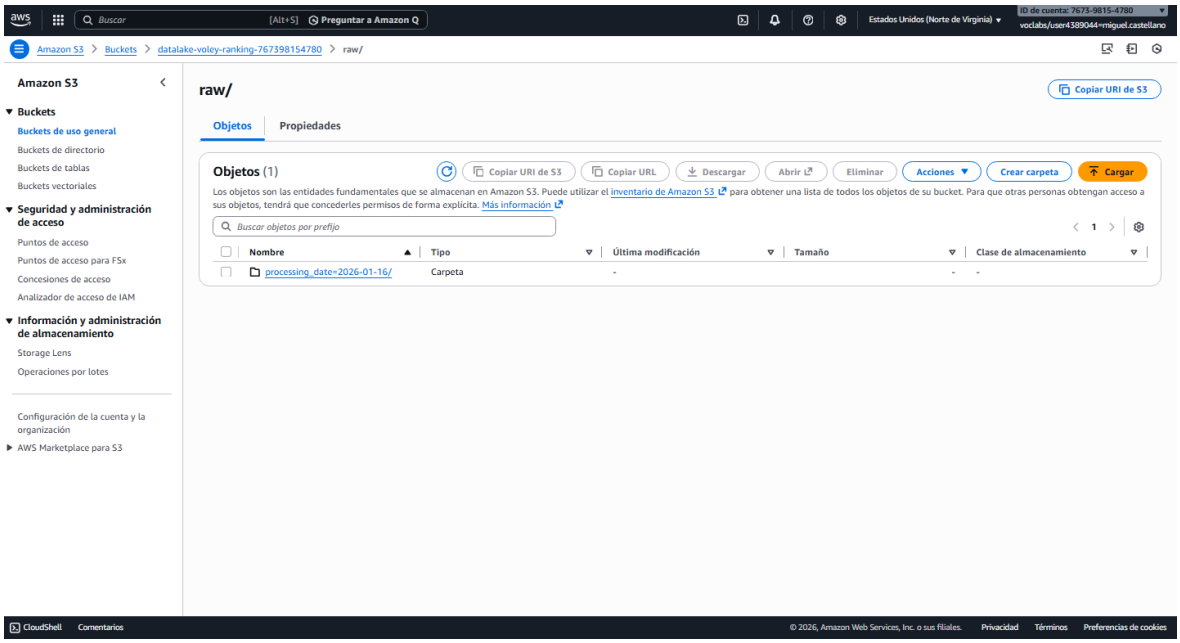

Figura 5: Resultados de la ingesta en S3: persistencia de datos organizada por prefijos temporales

2.4 Procesamiento Batch y Catalogación (AWS Glue)

2.4.1 Catalogación de Datos

Se configuró un Crawler que inspecciona los esquemas de los ficheros JSON en S3 y crea automáticamente definiciones de tablas en el Data Catalog.

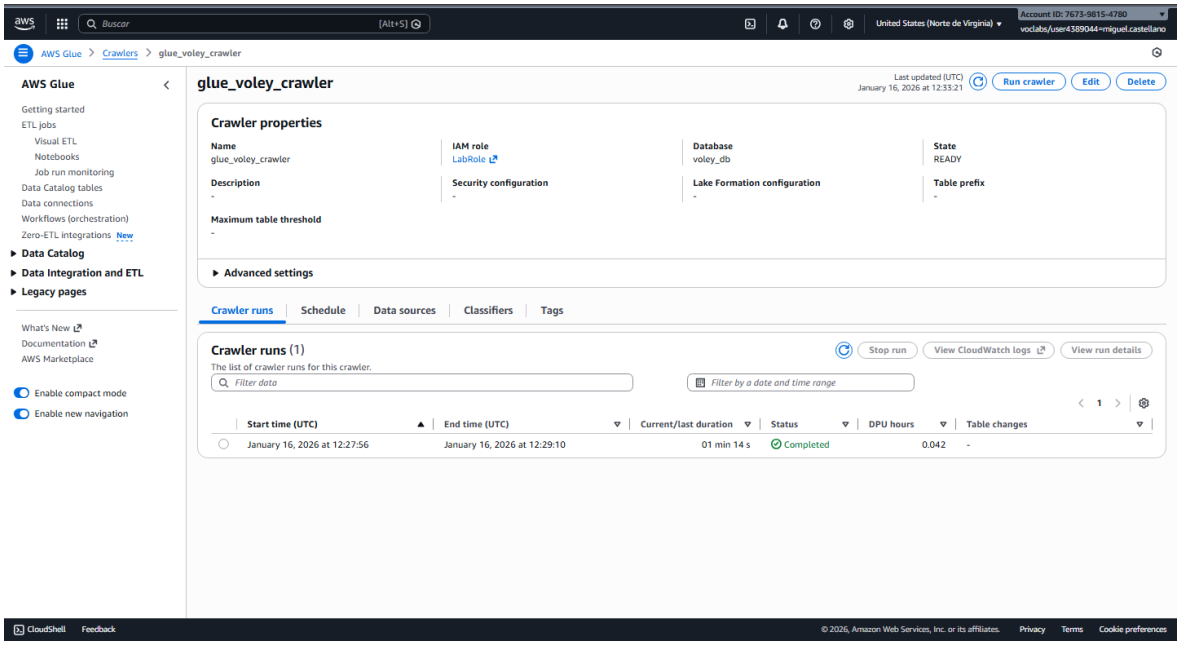

Figura 6: Configuración del Crawler encargado de la actualización del esquema en la base de datos

2.4.2 Ejecución de Jobs ETL

El panel de monitorización refleja el historial de ejecuciones, donde se identificaron y solventaron fallos iniciales de tipos de datos hasta lograr el éxito de los procesos.

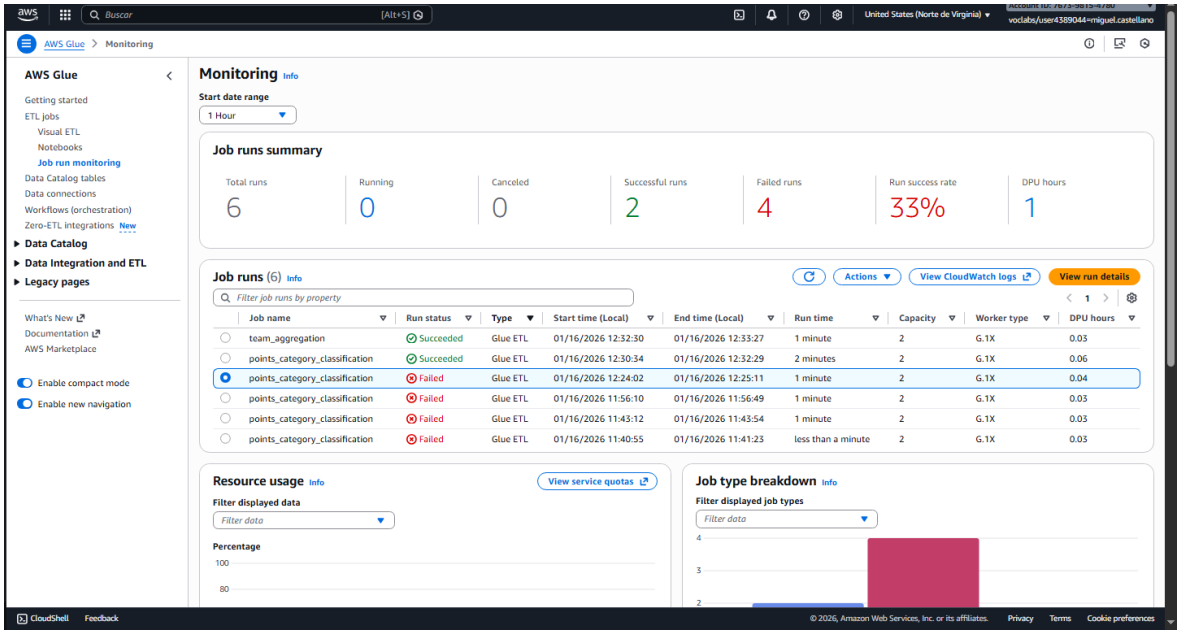

Figura 7: Panel de monitorización de AWS Glue mostrando el historial de ejecuciones y estados de los Jobs ETL.

Como resultado de estos Jobs, los datos se almacenan de forma particionada en Amazon S3, optimizando el rendimiento de las consultas posteriores (ver Figuras 8 y 9).

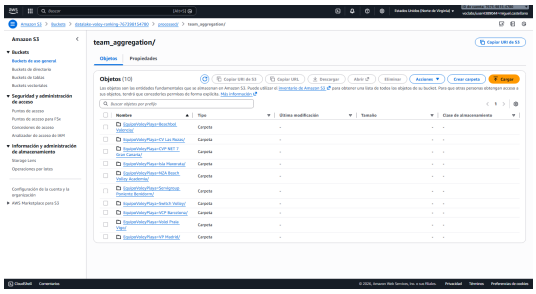

Figura 8: Estructura de salida particionada por equipo en S3.

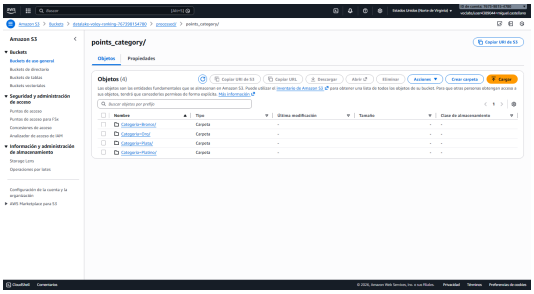

Figura 9: Estructura de salida particionada por categoría en S3.

2.5 Análisis y Generación del Ranking (Amazon Athena)

Finalmente, la capa de consumo se implementó en Amazon Athena, permitiendo ejecutar consultas SQL directamente sobre los ficheros Parquet almacenados en S3.

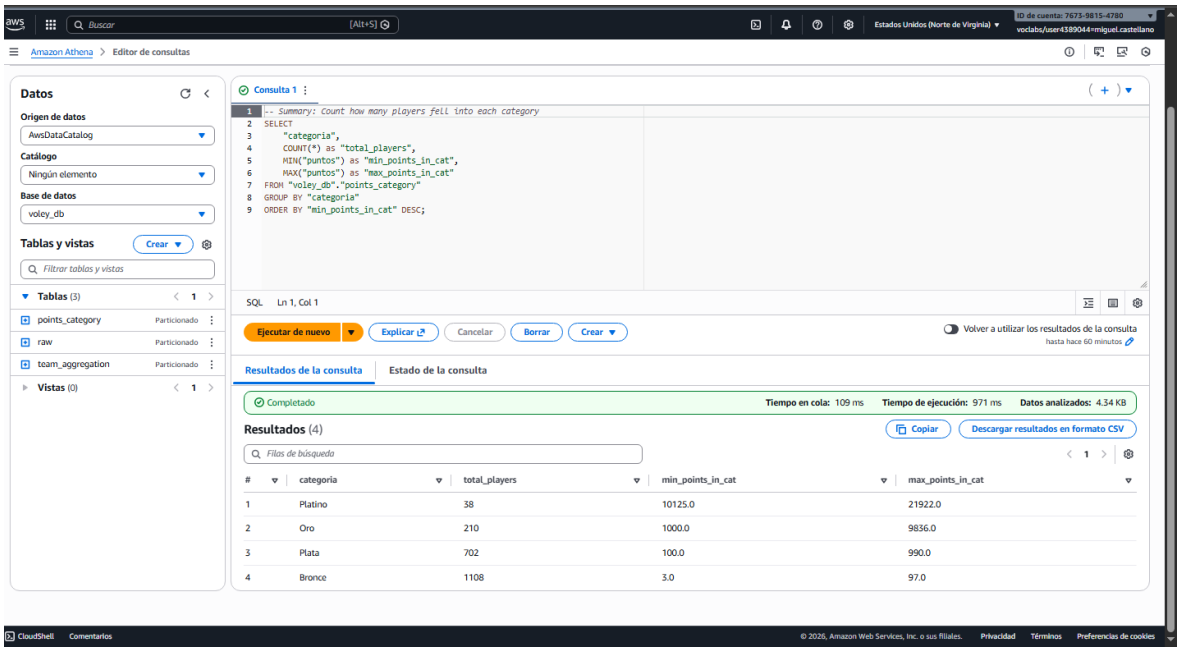

Figura 10: Análisis estadístico de la distribución de jugadores por categoría en Amazon Athena.

La **Figura 11** muestra la generación del ranking final de clubes, ordenando las entidades por el total de puntos acumulados.

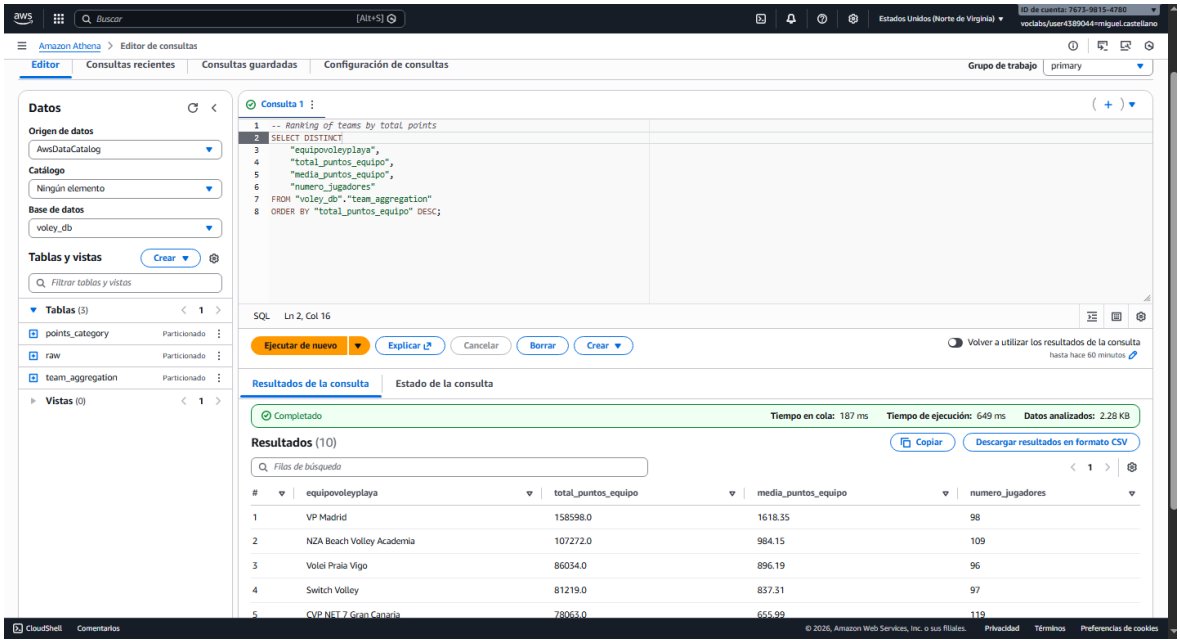

Figura 11: Generación del ranking final de equipos mediante consultas SQL en Amazon Athena.

3 Diagrama del flujo de datos

El pipeline de datos diseñado sigue una arquitectura *serverless* dividida en tres etapas clave (ver Figura 12):

- **Ingesta:** Captura de datos en tiempo real mediante **Kinesis Data Streams** y transporte procesado con **Firehose**, apoyado en una **Lambda** para el particionamiento temporal.
- **Procesamiento:** Almacenamiento en **S3** (capa *raw*) y catalogación mediante **Glue Crawler**. Los **Jobs de Glue** ejecutan la lógica de negocio y transforman los datos a formato Parquet (capa *processed*).
- **Consumo:** Análisis final y generación del ranking mediante consultas SQL directas en **Amazon Athena**.

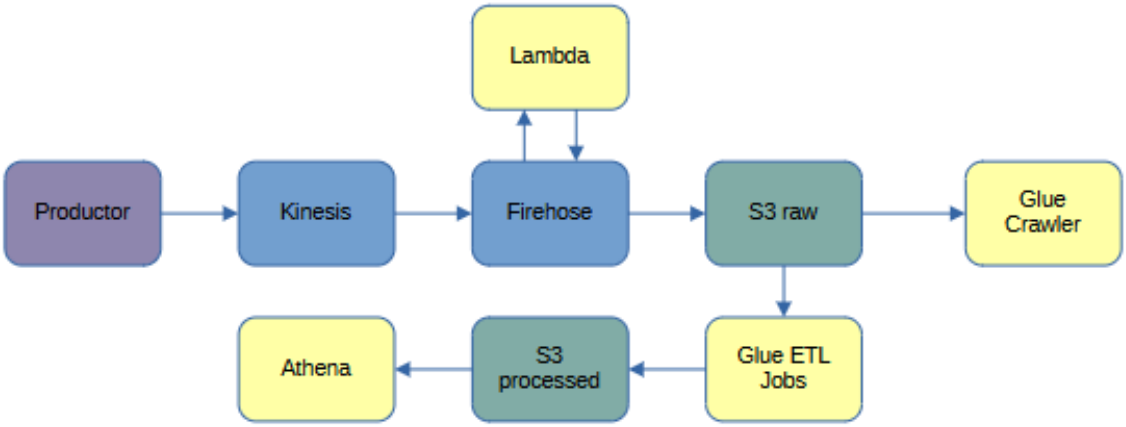

Figura 12: Arquitectura del pipeline de datos: flujo desde la ingesta hasta el análisis final.

4 Presupuesto y estimación de costes

La estimación de costes se ha realizado utilizando la **AWS Pricing Calculator** para la región de `us-east-1` (Norte de Virginia). El modelo se basa en un escenario real de 1.000 registros diarios ($\approx$ 156 KB por ingesta) y una retención de datos de un año.

5 Presupuesto y estimación de costes

La estimación de costes se ha realizado utilizando la **AWS Pricing Calculator** para la región de `us-east-1`. Los cálculos originales en USD se han convertido a euros utilizando una tasa de cambio aproximada de $1\ USD = 0,95\ EUR$. El modelo contempla 1.000 registros diarios y una retención de datos anual.

5.1 Desglose de costes (Mensual y Anual)

A continuación, se presenta el desglose detallado en euros. La mayoría de los servicios de almacenamiento y consulta operan en niveles mínimos de facturación debido al tamaño reducido de los archivos de ranking.

Cuadro 1: Estimación de costes: Ingesta y Procesamiento del Ranking (en €)

Servicio	**Concepto / Cálculo**	**Mensual (€)**	**Anual (€)**
Amazon Kinesis	1 Shard Provisionado (24/7)	10,26	123,12
Data Firehose	Ingesta <1 GB/mes (Mínimo)	0,01	0,11
AWS Lambda	Transformación (Capa gratuita)	0,00	0,00
Amazon S3	Almacenamiento y peticiones	0,05	0,57
AWS Glue Jobs	2 Jobs × 2 DPUs (10 min/día)	8,36	100,32
AWS Glue Crawler	1 Crawler × 2 DPUs (5 min/día)	2,09	25,08
Amazon Athena	Consultas SQL (<10 MB/mes)	0,01	0,11
Total Estimado		**20,78 €**	**249,31 €**

5.2 Análisis de eficiencia y optimización

Tras analizar el presupuesto, se identifican oportunidades de optimización para reducir el gasto innecesario derivado de la infrautilización de recursos:

- **Modo de Ingesta:** El coste de Kinesis representa casi el 50 % del total. Dado que la ingesta no es continua sino puntual (una vez al día), el uso de un *Shard* provisionado 24/7 es ineficiente. Cambiar a un modelo **On-Demand** o realizar una carga por lotes directamente a S3 reduciría este coste prácticamente a cero.
- **Motor de Procesamiento:** AWS Glue Spark está diseñado para grandes volúmenes de datos (*Big Data*). Para un ranking de 156 KB, el uso de **AWS Glue Python Shell** sería suficiente y mucho más económico, ya que permite fracciones de DPU (0,0625), lo que minimizaría el coste de procesamiento a escasos céntimos al mes.

6 Conclusiones

Tras la realización de esta práctica, se han alcanzado los objetivos propuestos mediante el diseño y despliegue de una arquitectura de datos *serverless* robusta. Las principales conclusiones extraídas son:

- **Eficacia del ecosistema AWS:** La integración de servicios como Kinesis, Firehose, Glue y Athena permite construir pipelines de datos complejos con una configuración mínima de infraestructura física. La capacidad de automatizar el descubrimiento de esquemas mediante *Crawlers* simplifica drásticamente el mantenimiento evolutivo del sistema.

- **Escalabilidad vs. Eficiencia económica:** Si bien la arquitectura diseñada es capaz de escalar horizontalmente para gestionar millones de jugadores sin cambios en el código, el análisis de costes revela que, para volúmenes pequeños de datos, el modelo de aprovisionamiento de Kinesis y Spark (Glue) resulta infrautilizado. Como se analizó en la sección anterior, la selección del ”sabor”de computación (Python Shell vs. Spark) es crítica para la rentabilidad del proyecto.

- **Importancia del formato Parquet:** El uso de formatos de almacenamiento columnar en la capa *processed* ha demostrado ser fundamental. Al ejecutar consultas en Amazon Athena, el escaneo de datos es significativamente menor en comparación con el formato JSON original, lo que reduce tanto la latencia de respuesta como el coste por consulta.

- **Aprendizaje técnico:** El desarrollo ha permitido profundizar en conceptos de *Data Engineering* como el particionamiento dinámico, la transformación en vuelo mediante AWS Lambda y la catalogación de metadatos, habilidades esenciales en el paradigma actual de la Computación en la Nube.

7 Referencias y bibliografía

8 Referencias y bibliografía

1. Amazon Web Services (2024-2026). *Documentación técnica oficial de AWS: Amazon S3, Kinesis Data Streams, Data Firehose, AWS Glue y Amazon Athena*. Recuperado de: `https://docs.aws.amazon.com/`

2. Real Federación Española de Voleibol (RFEVB). *Normativa de Ranking Nacional de Vóley Playa*. Recuperado de: `https://www.rfevb.com`

3. Axel Cabrera. *Repositorio de base para la práctica de Computación en la Nube (p5_aula)*. GitHub. Recuperado de: `https://github.com/Axelcab/p5_aula`

9 Anexos

En esta sección se adjunta el código fuente desarrollado para la implementación de la arquitectura.

9.1 Productor de Datos (Ingesta)

```
import time
import boto3
import json

kinesis_client = boto3.client('kinesis', region_name='us-east-1')

STREAM_NAME = 'beach-voley-national-ranking'

def enviar_ranking_completo(ruta_archivo_json):
    try:
        with open(ruta_archivo_json, 'r', encoding='utf-8') as f:
            jugadores = json.load(f)

        print(f"Iniciando carga de {len(jugadores)} jugadores...")

        for jugador in jugadores:
            response = kinesis_client.put_record(
                StreamName=STREAM_NAME,
                Data=json.dumps(jugador).encode('utf-8'),
                PartitionKey=str(jugador['IdPersona'])
            )
            time.sleep(0.05)  # pequena pausa para no saturar
            print(f"Enviado: {jugador['ApellidosNombre']} -
                Sequence: {response['SequenceNumber']}")

    except FileNotFoundError:
        print("Error: No se encontro el archivo JSON con los datos
            del ranking.")
    except Exception as e:
        print(f"Error inesperado: {e}")

if __name__ == "__main__":
    enviar_ranking_completo('src/data/beach_voley_national_ranking.
        json')
```

Listing 1: Script del Productor (kinesis.py)

9.2 Lógica de Transformación (AWS Lambda)

```
import base64
import json
from datetime import datetime

def lambda_handler(event, context):
    output = []
    processing_date = datetime.now().strftime('%Y-%m-%d')

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

            # Normalizacion de puntos (decimal con coma a punto)
            if 'Puntos' in data:
                puntos_normalizados = str(data['Puntos']).replace('
                    ,', '.')
                data['Puntos'] = float(puntos_normalizados)

            data['processing_date'] = processing_date

            new_data = json.dumps(data) + '\n'

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

```
        except Exception as e:
            output_record = {
                'recordId': record.get('recordId', 'unknown'),
                'result': 'ProcessingFailed',
                'data': record.get('data', '')
            }
            output.append(output_record)
            print(f"Error procesando registro: {e}")

    return {'records': output}
```

Listing 2: Función Lambda para Firehose (firehose.py)

9.3 Trabajos ETL (AWS Glue)

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.dynamicframe import DynamicFrame
from awsglue.job import Job
from pyspark.sql.functions import col, when
from pyspark.sql.types import DoubleType

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

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

# Lectura desde Catalogo
datasource = glueContext.create_dynamic_frame.from_catalog(
    database=args['database'],
    table_name=args['table_name'],
    transformation_ctx="datasource"
)
```

```
df = datasource.toDF()
df = df.withColumn("Puntos", col("Puntos").cast(DoubleType()))

# Transformacion: Categorizacion
df_categoria = df.withColumn(
    "Categoria",
    when(col("Puntos") >= 10000, "Platino")
    .when(col("Puntos") >= 1000, "Oro")
    .when(col("Puntos") >= 100, "Plata")
    .otherwise("Bronce")
)

df_categoria = df_categoria.repartition("Categoria")

# Escritura en Parquet particionado
output_dyf = DynamicFrame.fromDF(df_categoria, glueContext, "
   output_dyf")

glueContext.write_dynamic_frame.from_options(
    frame=output_dyf,
    connection_type="s3",
    connection_options={
        "path": args['output_path'],
        "partitionKeys": ["Categoria"]
    },
    format="parquet",
    format_options={"compression": "snappy"},
    transformation_ctx="datasink"
)

job.commit()
```

Listing 3: Job 1: Clasificación por Categorías (points_category_classification.py)

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.dynamicframe import DynamicFrame
from awsglue.job import Job
from pyspark.sql.functions import col, sum as spark_sum, avg, count
```

```
    , round
from pyspark.sql.types import DoubleType

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

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

datasource = glueContext.create_dynamic_frame.from_catalog(
    database=args['database'],
    table_name=args['table_name'],
    transformation_ctx="datasource"
)

df = datasource.toDF()
df = df.withColumn("Puntos", col("Puntos").cast(DoubleType()))

# Agregacion por Equipo
df_equipo = df.groupBy("EquipoVoleyPlaya").agg(
    spark_sum("Puntos").alias("total_puntos_equipo"),
    avg("Puntos").alias("media_puntos_equipo"),
    count("*").alias("numero_jugadores")
)

df_equipo = df_equipo.withColumn("media_puntos_equipo", round(col("
    media_puntos_equipo"), 2))
df_equipo = df_equipo.repartition("EquipoVoleyPlaya")

output_dyf = DynamicFrame.fromDF(df_equipo, glueContext, "
    output_dyf")

glueContext.write_dynamic_frame.from_options(
    frame=output_dyf,
    connection_type="s3",
    connection_options={
        "path": args['output_path'],
        "partitionKeys": ["EquipoVoleyPlaya"]
```

```
    },
    format="parquet",
    format_options={"compression": "snappy"},
    transformation_ctx="datasink"
)

job.commit()
```

Listing 4: Job 2: Agregación por Equipos (team_aggregation.py)

9.4 Scripts de Despliegue (Infraestructura)

A continuación se incluyen los scripts de PowerShell utilizados para automatizar la creación de recursos en AWS.

```
$env:AWS_REGION = "us-east-1"
$env:ACCOUNT_ID = (aws sts get-caller-identity --query Account --
   output text)
$env:BUCKET_NAME = "datalake-voley-ranking-$($env:ACCOUNT_ID)"

Write-Host "Creando␣Bucket␣S3:␣$($env:BUCKET_NAME)..."
aws s3 mb "s3://$env:BUCKET_NAME"

# Creacion de carpetas logicas
aws s3api put-object --bucket $env:BUCKET_NAME --key "raw/"
aws s3api put-object --bucket $env:BUCKET_NAME --key "processed/"
aws s3api put-object --bucket $env:BUCKET_NAME --key "scripts/"
aws s3api put-object --bucket $env:BUCKET_NAME --key "errors/"
```

Listing 5: Creación del Bucket y Estructura S3 (create_bucket.ps1)

```
$env:AWS_REGION = "us-east-1"
$env:ACCOUNT_ID = (aws sts get-caller-identity --query Account --
   output text)

$BASE_DIR = Split-Path -Parent $PSScriptRoot
$JOBS_DIR = "$BASE_DIR\jobs"

$BUCKET_NAME = "datalake-voley-ranking-$($env:ACCOUNT_ID)"
$JOB1_SCRIPT = "points_category_classification.py"
$JOB2_SCRIPT = "team_aggregation.py"

$JOB1_OUTPUT = "s3://$BUCKET_NAME/processed/points_category/"
```

```
$JOB2_OUTPUT = "s3://$BUCKET_NAME/processed/team_aggregation/"

$ROLE_NAME = "LabRole"
$GLUE_DB = "voley_db"

$GLUE_JOB_1 = "points_category_classification"
$GLUE_JOB_2 = "team_aggregation"

$S3_RAW_DATA = "s3://$BUCKET_NAME/raw/"
$CRAWLER_NAME = "glue_voley_crawler"

$S3_PROCESSED_DATA = "s3://$BUCKET_NAME/processed/"
$CRAWLER_PROCESSED_NAME = "glue_voley_processed_crawler"

Write-Host "=== CONFIGURATION LOADED ===" -ForegroundColor Cyan

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

function Wait-ForJob {
    param($JobName, $RunId)
    Write-Host "Waiting for Glue Job: $JobName..." -ForegroundColor
        Magenta
    do {
        Start-Sleep -Seconds 15
        $Status = (aws glue get-job-run --job-name $JobName --run-
           id $RunId --query 'JobRun.JobRunState' --output text).
           Trim()
        Write-Host "  -> Status: $Status" -ForegroundColor Gray
    } while ($Status -in @("STARTING", "RUNNING", "STOPPING"))
    return $Status
}

function Wait-ForCrawler {
    param($Name)
    Write-Host "Waiting for Crawler: $Name..." -ForegroundColor
```

```
        Magenta
    do {
        Start-Sleep -Seconds 10
        $Status = (aws glue get-crawler --name $Name --query '
            Crawler.State' --output text)
        Write-Host "␣␣->␣State:␣$Status" -ForegroundColor Gray
    } while ($Status -ne "READY")
}

function Wait-ForDeletion {
    param($Name)
    while ($true) {
        $null = aws glue get-crawler --name $Name 2>$null
        if ($LASTEXITCODE -ne 0) { break }
        Start-Sleep -Seconds 5
    }
}

Write-Host "`n[1/4]␣Uploading␣Spark␣scripts␣to␣S3..." -
    ForegroundColor Cyan
aws s3 cp "$JOBS_DIR\$JOB1_SCRIPT" "s3://$BUCKET_NAME/scripts/
    $JOB1_SCRIPT"
aws s3 cp "$JOBS_DIR\$JOB2_SCRIPT" "s3://$BUCKET_NAME/scripts/
    $JOB2_SCRIPT"

Write-Host "`n[2/4]␣Configuring␣Data␣Catalog..." -ForegroundColor
    Cyan

$null = aws glue create-database --database-input "{\"Name\":\"
    $GLUE_DB\"}" 2>$null

$null = aws glue get-crawler --name $CRAWLER_NAME 2>$null
if ($LASTEXITCODE -eq 0) {
    aws glue delete-crawler --name $CRAWLER_NAME
    Wait-ForDeletion $CRAWLER_NAME
}

$CRAWLER_TARGETS = "{\`"S3Targets\`":␣[{\`"Path\`":␣\`"$S3_RAW_DATA
    \`"}]}"
aws glue create-crawler --name $CRAWLER_NAME --role $ROLE_NAME --
    database-name $GLUE_DB --targets $CRAWLER_TARGETS
```

```
Write-Host "Running␣RAW␣Crawler..." -ForegroundColor Yellow
$null = aws glue start-crawler --name $CRAWLER_NAME
Wait-ForCrawler $CRAWLER_NAME

$TABLE_NAME = (aws glue get-tables --database-name $GLUE_DB --query
    "TableList[0].Name" --output text).Trim().Replace('"','')
Write-Host "Table␣detected:␣$TABLE_NAME" -ForegroundColor Green

Write-Host "`n[3/4]␣Creating␣Glue␣ETL␣Jobs..." -ForegroundColor
   Cyan

function Create-GlueJob($JobName, $ScriptName, $OutputPath) {
    $null = aws glue delete-job --job-name $JobName 2>$null

    $JobCommand = @{ Name = "glueetl"; ScriptLocation = "s3://
       $BUCKET_NAME/scripts/$ScriptName"; PythonVersion = "3" }
    $JobArgs = @{ "--job-language" = "python"; "--output_path" =
       $OutputPath; "--database" = $GLUE_DB; "--table_name" =
       $TABLE_NAME }

    $CmdFile  = New-TempJson $JobCommand
    $ArgsFile = New-TempJson $JobArgs

    aws glue create-job --name $JobName --role $ROLE_NAME --command
        "file://$CmdFile" --default-arguments "file://$ArgsFile" --
       glue-version "4.0" --worker-type "G.1X" --number-of-workers
       2
    Remove-Item $CmdFile, $ArgsFile
}

Create-GlueJob $GLUE_JOB_1 $JOB1_SCRIPT $JOB1_OUTPUT
Create-GlueJob $GLUE_JOB_2 $JOB2_SCRIPT $JOB2_OUTPUT

Write-Host "`n[4/4]␣Executing␣Pipeline..." -ForegroundColor Cyan

$jobRun1 = aws glue start-job-run --job-name $GLUE_JOB_1 --query '
   JobRunId' --output text
$status1 = Wait-ForJob $GLUE_JOB_1 $jobRun1

if ($status1 -eq "SUCCEEDED") {
```

```
    $jobRun2 = aws glue start-job-run --job-name $GLUE_JOB_2 --
       query 'JobRunId' --output text
    $status2 = Wait-ForJob $GLUE_JOB_2 $jobRun2
} else {
    Write-Host "Job 1 Failed. Skipping Job 2." -ForegroundColor Red
    $status2 = "SKIPPED"
}

Write-Host "`n=== REGISTERING PROCESSED TABLES ===" -
   ForegroundColor Cyan

$null = aws glue get-crawler --name $CRAWLER_PROCESSED_NAME 2>$null
if ($LASTEXITCODE -eq 0) {
    aws glue delete-crawler --name $CRAWLER_PROCESSED_NAME
    Wait-ForDeletion $CRAWLER_PROCESSED_NAME
}

$PROCESSED_TARGETS = "{\`"S3Targets\`": [{\`"Path\`": \`"
   $S3_PROCESSED_DATA\`"}]}"
aws glue create-crawler --name $CRAWLER_PROCESSED_NAME --role
   $ROLE_NAME --database-name $GLUE_DB --targets $PROCESSED_TARGETS

Write-Host "Running Processed Data Crawler..." -ForegroundColor
   Yellow
$null = aws glue start-crawler --name $CRAWLER_PROCESSED_NAME
Wait-ForCrawler $CRAWLER_PROCESSED_NAME

Write-Host "`n=== PIPELINE FINISHED ===" -ForegroundColor Cyan
Write-Host "$GLUE_JOB_1 : $status1"
Write-Host "$GLUE_JOB_2 : $status2"
Write-Host "Processed tables are now available in Glue Catalog." -
   ForegroundColor Green
```

Listing 6: Despliegue y Orquestación Glue (glue.ps1)

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

```
Write-Host "`nCreando␣Firehose␣con␣Particionamiento␣Dinámico..." -
    ForegroundColor Yellow

$DELIVERY_STREAM_NAME = "voley-firehose"
$KINESIS_STREAM_NAME = "beach-voley-national-ranking"

aws firehose delete-delivery-stream --delivery-stream-name
    $DELIVERY_STREAM_NAME 2>$null
Write-Host "Esperando␣a␣que␣el␣stream␣anterior␣se␣elimine..."
Start-Sleep -Seconds 30

$FIREHOSE_CONFIG_TEMPLATE = @'
{
␣␣"BucketARN":␣"arn:aws:s3:::REPLACE_BUCKET",
␣␣"RoleARN":␣"REPLACE_ROLE",
␣␣"Prefix":␣"raw/processing_date=!{partitionKeyFromQuery:
    processing_date}/",
␣␣"ErrorOutputPrefix":␣"errors/!{firehose:error-output-type}/",
␣␣"BufferingHints":␣{
␣␣␣␣"SizeInMBs":␣64,
␣␣␣␣"IntervalInSeconds":␣60
␣␣},
␣␣"ProcessingConfiguration":␣{
␣␣␣␣"Enabled":␣true,
␣␣␣␣"Processors":␣[
␣␣␣␣␣␣{
␣␣␣␣␣␣␣␣"Type":␣"MetadataExtraction",
␣␣␣␣␣␣␣␣"Parameters":␣[
␣␣␣␣␣␣␣␣␣␣{
␣␣␣␣␣␣␣␣␣␣␣␣"ParameterName":␣"MetadataExtractionQuery",
␣␣␣␣␣␣␣␣␣␣␣␣"ParameterValue":␣"{processing_date:␣.processing_date}"
␣␣␣␣␣␣␣␣␣␣},
␣␣␣␣␣␣␣␣␣␣{
␣␣␣␣␣␣␣␣␣␣␣␣"ParameterName":␣"JsonParsingEngine",
␣␣␣␣␣␣␣␣␣␣␣␣"ParameterValue":␣"JQ-1.6"
␣␣␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣]
␣␣␣␣␣␣},
␣␣␣␣␣␣{
␣␣␣␣␣␣␣␣"Type":␣"Lambda",
␣␣␣␣␣␣␣␣"Parameters":␣[
```

```
          {
            "ParameterName": "LambdaArn",
            "ParameterValue": "REPLACE_LAMBDA"
          },
          {
            "ParameterName": "NumberOfRetries",
            "ParameterValue": "3"
          }
        ]
      }
    ]
  },
  "DynamicPartitioningConfiguration": {
    "Enabled": true,
    "RetryOptions": {
      "DurationInSeconds": 300
    }
  }
}
'@

$FIREHOSE_CONFIG_JSON = $FIREHOSE_CONFIG_TEMPLATE `
  -replace "REPLACE_BUCKET", $env:BUCKET_NAME `
  -replace "REPLACE_ROLE", $env:ROLE_ARN `
  -replace "REPLACE_LAMBDA", $LAMBDA_ARN

$CONFIG_FILE = "$PSScriptRoot/firehose_config.json"
$utf8NoBom = New-Object System.Text.UTF8Encoding($false)
[System.IO.File]::WriteAllText($CONFIG_FILE, $FIREHOSE_CONFIG_JSON,
    $utf8NoBom)

aws firehose create-delivery-stream `
  --delivery-stream-name $DELIVERY_STREAM_NAME `
  --delivery-stream-type KinesisStreamAsSource `
  --kinesis-stream-source-configuration "KinesisStreamARN=arn:aws:
     kinesis:$($env:AWS_REGION):$($env:ACCOUNT_ID):stream/
     $KINESIS_STREAM_NAME,RoleARN=$env:ROLE_ARN" `
  --extended-s3-destination-configuration file://$CONFIG_FILE

if ($?) {
  Write-Host "=== Firehose creado y configurado correctamente ==="
```

```
    -ForegroundColor Green
} else {
  Write-Host "=== Error en la creación de Firehose ===" -
    ForegroundColor Red
}

Remove-Item $CONFIG_FILE -ErrorAction SilentlyContinue
```

Listing 7: Configuración de Firehose (firehose_setup.ps1)

9.5 Muestra de Datos (JSON)

Ejemplo truncado de la estructura de datos utilizada (Ranking Nacional):


```
[
  {
    "IdPersona": "392510",
    "ApellidosNombre": "VIERA IGLESIAS, ALVARO",
    "Puntos": "21,922",
    "EquipoVoleyPlaya": "VP Madrid"
  },
  {
    "IdPersona": "312721",
    "ApellidosNombre": "SAUCEDO AMODEO, ANTONIO MANUEL",
    "Puntos": "21,922",
    "EquipoVoleyPlaya": "Switch Volley"
  },
  {
    "IdPersona": "299171",
    "ApellidosNombre": "HUERTA PASTOR, ALEJANDRO",
    "Puntos": "21,269",
    "EquipoVoleyPlaya": "NZA Beach Volley Academia"
  },
  {
    "IdPersona": "299170",
    "ApellidosNombre": "HUERTA PASTOR, JAVIER",
    "Puntos": "18,448",
    "EquipoVoleyPlaya": "CVP NET 7 Gran Canaria"
  },
  {
    "IdPersona": "309787",
    "ApellidosNombre": "MORENO FERRER, DANIEL",
    "Puntos": "16,786",
```

```
    "EquipoVoleyPlaya": "VCP Barcelona"
  }
  ... (resto de registros omitidos por brevedad)
]
```

Listing 8: Muestra del archivo de entrada (beach_voley_national_ranking.json)

9.6 Uso de Inteligencia Artificial

Se han utilizado **Gemini** y **ChatGPT** para la depuración de código, la generación de los datos ficticios a tratar y la estructuración de la memoria en LaTeX.