ISC	Infraestructuras y Servicios Cloud
25/26	Ingesta, almacenamiento y Serveless
GIIA	Práctica 5

Práctica 5 ISC: Data Lake Serverless para Smart City "Vigía"

Introducción: El Desafío de la Smart City "Vigía"

La ciudad de "Neo-Tech" está implementando el programa "Vigía" para optimizar el consumo energético de su red de **farolas inteligentes**. El desafío es que la información de consumo llega en **tiempo real** (IoT) y debe ser correlacionada con **datos históricos masivos** de movilidad y **datos de contexto** (tarifas energéticas) para tomar decisiones.

El Desafío de la Ciudad: La ciudad de "Neo-Tech" está implementando su programa "Vigía" para ser más sostenible y eficiente. El mayor reto es que el departamento de energía no puede correlacionar el consumo eléctrico en tiempo real con los factores que lo provocan (el clima, las horas pico, o la densidad de tráfico). Necesitan una plataforma que no solo almacene todos los datos, sino que los haga consultables y procesables automáticamente.

Su Misión como Ingeniero de IA: Usted debe diseñar y construir el backend de datos para la plataforma Vigía en la nube (AWS).

Ingesta Rápida: Debe configurar la recepción de datos de Farolas Inteligentes (dispositivos simulados) a través del punto de entrada de IoT.

Unificación: Debe configurar un Data Lake (S3) que almacene los datos de los sensores, los datos de tarifas externas (Web Scraping) y el histórico masivo de tráfico.

Procesamiento Inteligente: Usando servicios serverless (AWS Glue y Athena), debe transformar la data masiva y de streaming en un formato único y optimizado (Parquet) para que los analistas puedan consultar: "¿Cuál es el patrón de consumo energético en las zonas con alta densidad de tráfico y alta contaminación?"

Objetivo General

Diseñar e implementar una arquitectura de datos **Serverless** en AWS que gestione el ciclo de vida de los datos. El estudiante debe demostrar el uso de **AWS IoT Core** (Ingesta *Streaming*), **DynamoDB** (NoSQL *Serving Layer*), **S3** (Data Lake), **AWS Lambda** (Lógica Serverless) y **AWS Glue/Athena** (Procesamiento y Análisis).

Requisito	**Servicios Serverless de AWS**	**Justificación de la Elección**
Ingesta Streaming	AWS IoT Core	Pasarela segura para dispositivos IoT (Farolas).
Almacenamiento Dual (RA 1)	S3 y DynamoDBv2	S3 para histórico (**Big Data**) y DynamoDB para estado actual (**Baja Latencia**).
Computación (RA 2)	AWS Glue & Lambda	Glue para ETL masivo y Lambda para tareas programadas (Web Scraping).

I. Flujo IoT: Ingesta de Telemetría y Almacenamiento Dual

Esta fase conecta Node-RED (simulando las farolas inteligentes) a AWS IoT Core y enruta los datos al Data Lake y a la capa de Servido NoSQL.

Paso 1: Preparación de Destinos y Configuración IoT

1. **Creación de la Tabla DynamoDB (NoSQL)**
    - **Acción:** Cree la tabla vigia_farolas_estado con **farola_id** como Clave de Partición (String).
    - **Justificación:** Se utiliza DynamoDB para la **Capa de Servido Operacional**. Almacena el último estado de la farola, lo cual es vital para la monitorización en tiempo real con latencia de milisegundos.
2. **Configuración de AWS IoT Core**
    - En la lista de servicios, acceder a los servicios “Internet de las Cosas” en el menú superior de la izquierda o en el buscador.
    - Activar el servicio IoT Core para poder recibir los mensajes inyectados por los dispositivos.
    - En la opción Conectarse > Conectar un dispositivo seguiremos los pasos que se nos indican en las pantallas para crear un Thing (Objeto) y descargar el “kit” para poder conectar el dispositivo al servicio IoT.
        1. Durante los pasos, crearemos el Objeto.
        2. Crearemos el kit para la plataforma (SO + Lenguaje del cliente) que debemos almacenar en el cliente donde hayamos instalado Node-RED para su posterior configuración.
        3. Los certificados y las claves van en el kit, pero también podríamos descargarla en la sección "Security" del Thing.
            1. Localiza el Archivo del Script: En la carpeta que descargaste, busca el archivo ejecutable. Su contenido es la clave para la configuración.
            2. Extrae la Información de Conexión: Analiza la última línea del script para encontrar los datos que necesitas.
            3. Endpoint: El valor que sigue a --endpoint.
            4. Clave Privada (--key) y Certificado del Dispositivo (--cert): El nombre de los archivos que se encuentran en la misma carpeta.
            5. ID del Cliente (--client_id): Un identificador para tu conexión.
            6. Topic de Ejemplo (--topic): Un topic de prueba. En nuestro caso no usaremos ese y se creará dinámicamente desde el código.

7. Descarga el Certificado Raíz: Como el kit no lo incluye, descárgalo directamente desde el enlace oficial de Amazon. Es un archivo esencial para la seguridad.

   https://www.amazontrust.com/repository/AmazonRootCA1.pem

- **Ajuste de Política:** AWS crea una política por defecto que es muy restrictiva. Debes editarla para que tu flujo de Node-RED pueda publicar en tu topic personalizado.
  1. **Ir a la Política:** En la consola de AWS, navega a **AWS IoT Core > Secure > Policies**. Haz clic en la política con el nombre de tu Thing y edita las políticas.

  2. **Editar el Documento de la Política:** Reemplaza el documento JSON de la política por el siguiente, más sencillo y funcional. Este documento da permiso para conectar y publicar en cualquier tema.

```
{
        "Version": "2012-10-17",
        "Statement": [
        {
                "Effect": "Allow",
                "Action": [
                "iot:Connect",
                "iot:Publish",
                "iot:Receive",
                "iot:Subscribe" ],
                "Resource": "*"
        }
        ]
}
```

  Verificar la Vinculación: En **AWS IoT Core > Secure > Certificates**, haz clic en el certificado de tu dispositivo y comprueba que tu política esté adjunta. Si no es así, adjúntala.

Paso 2: Creación de la Regla de IoT (El Enrutador Serverless)

Creamos una sola regla para el enrutamiento y la transformación, resolviendo el problema del mapeo y los permisos de IAM simultáneamente.

1. **Declaración de Consulta SQL (Enriquecimiento)**

- **Acción:** Ingrese la siguiente consulta en la Regla (en **Actuar → Reglas**):

```
SELECT
  home_id,
  consumption_kw,
  timestamp,
  timestamp() AS processed_time
FROM
  'smartcity/consumo/#'
```

- **Justificación:** El SQL enriquece el mensaje añadiendo el **processed_time** (marca de tiempo de ingestión) y selecciona explícitamente los atributos. Esto es un paso de **Transformación Serverless** y asegura que el JSON enviado a los destinos esté limpio.

2. **Acción 1: S3 (Data Lake)**
   - **Acción:** Configure la acción **Enviar a Bucket S3**.
   - **Clave S3:** farolas/year=${timestamp("yyyy")}/month=${timestamp("MM")}/day=${timestamp("dd")}/${timestamp()}-data.json
   - **Rol: Seleccione un Rol de Servicio preexistente** (RolLab).

3. **Acción 2: DynamoDBv2 (Servidor Operacional NoSQL)**
   - **Acción:** Configure la acción **Insertar en una tabla de DynamoDB (v2)**.
   - **Tabla:** vigia_farolas_estado.
   - **Clave de Partición:** **farola_id** (Nombre de la columna en DynamoDB).
   - **Valor de Clave de Partición:** **${home_id}** (El valor del identificador del dispositivo original).
   - **Plantilla JSON de Atributos: (Si la opción está visible)** Use esta plantilla para asegurar que no aparezca payload:

```
JSON
{
  "farola_id": "${home_id}",
  "consumption_kw": "${consumption_kw}",
  "processed_time": "${processed_time}",
  "timestamp": "${timestamp}"
}
```

- **Justificación:** Al usar el campo original ${home_id} para la clave de partición y especificar los atributos, se logra el **modelo de datos NoSQL** deseado sin la columna payload.

Paso 3: Ingesta (Node-RED)

Los estudiantes deberán instalar las siguientes herramientas:

- **Node.js y npm:** Es el entorno de ejecución para Node-RED.

  Id a la página oficial de Node.js y descarguen la versión recomendada (LTS). El instalador incluye npm (Node Package Manager).

- **Node-RED:** Abran una terminal o un símbolo del sistema. Ejecuten el comando: npm install -g --unsafe-perm node-red. Aquí está la dirección para los pasos de instalación según el sistema operativo utilizado.

  Running Node-RED locally : Node-RED

  Podéis también ayudaros de los diferentes nodos en el canal de ayuda con Node-RED:

  TUTORIAL completo de NODE RED: Aprende a utilizarlo de forma SENCILLA

- **Instalar la librería Cheerio:** Cuando accedáis a Node-Red a través del navegador, id al Menú > Manage Palette > Install y buscar cheerio, Instalar.

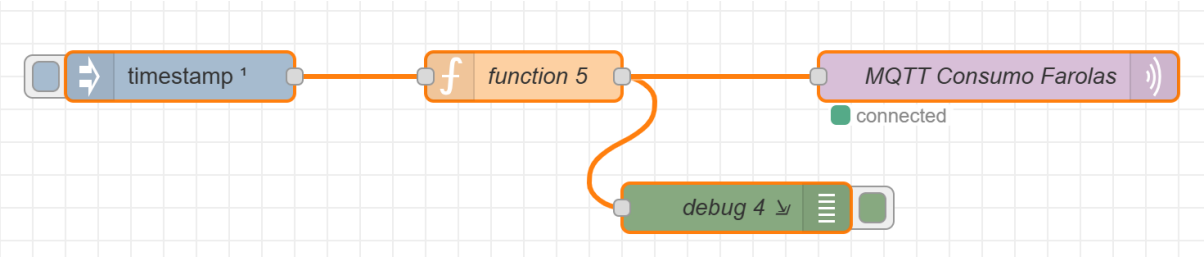
Creación de flujos

Los estudiantes crearán dos flujos en el editor de Node-RED (accesible en el navegador en http://localhost:1880).

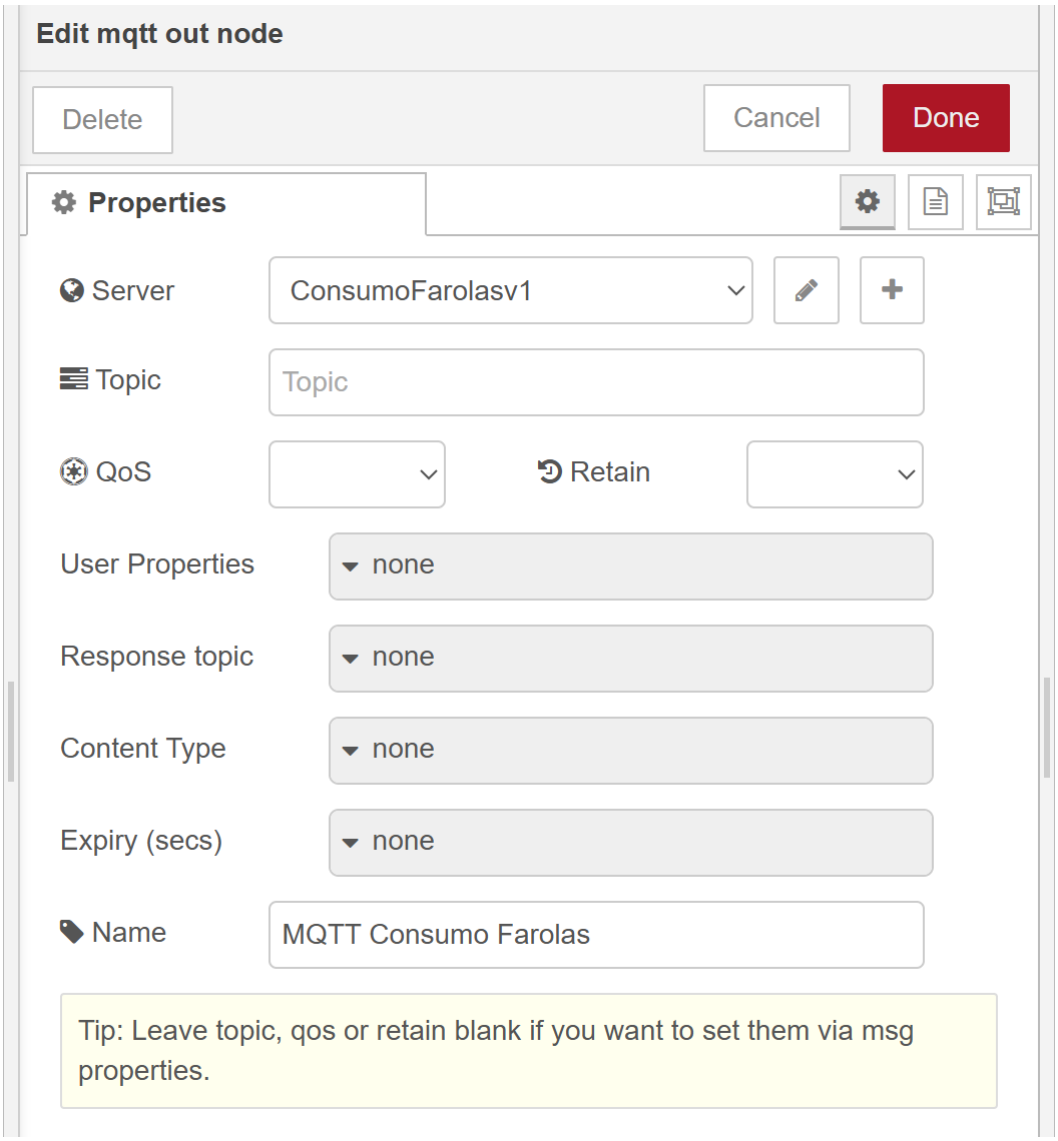
Flujo 1: Simulación de Consumo de la red de farolas Hogar

1. Arrastren un nodo **Inject** y configúrenlo para que se repita **cada 10 segundos** (inicialmente cuando se haga click sobre dicho nodo).
2. Conecten un nodo **Function** a su salida. Copien y peguen el siguiente código:

3. Conecten la salida del Function a un nodo **mqtt out**.

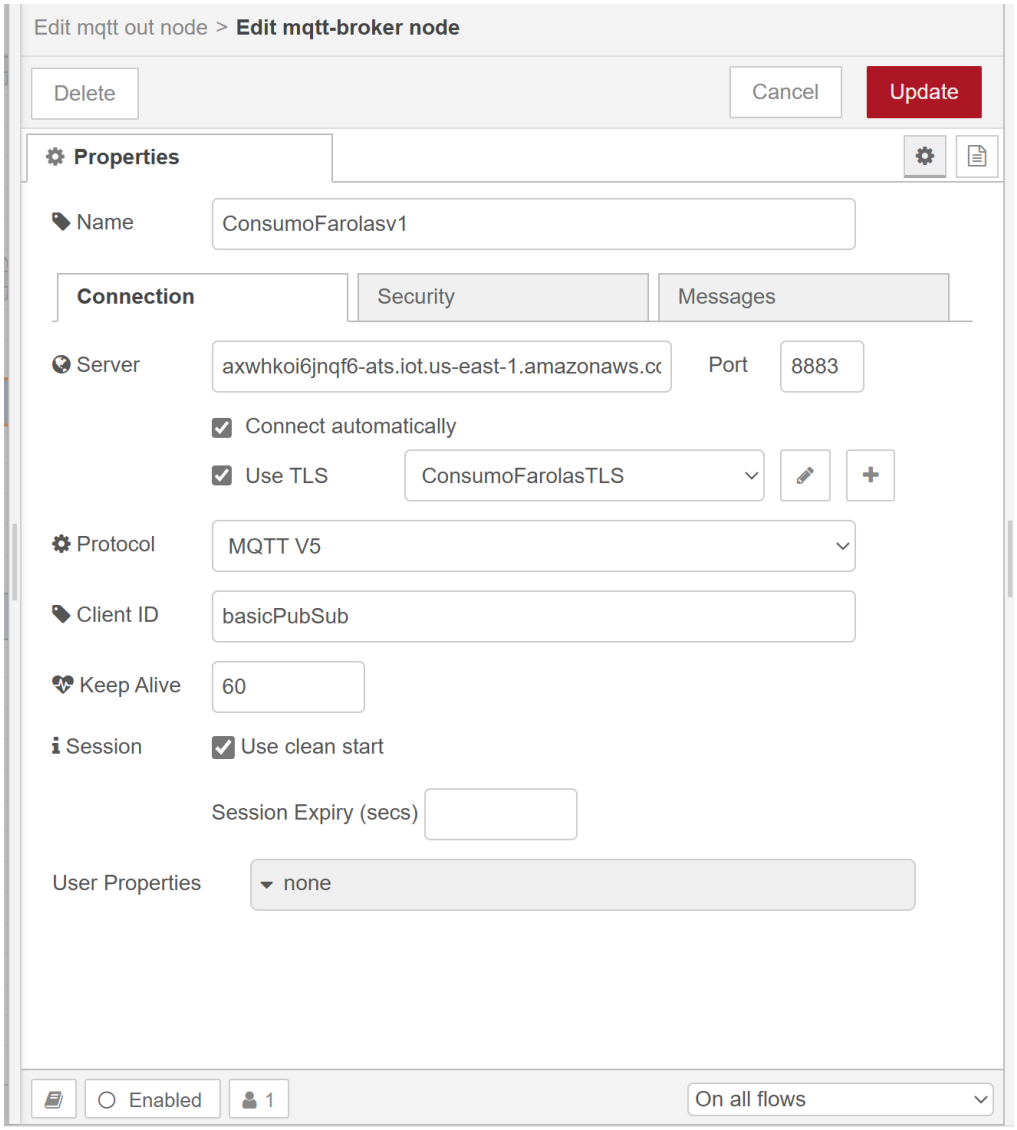


Configurar el nodo MQTT Consumo Farolas haciendo doble click sobre el nodo.

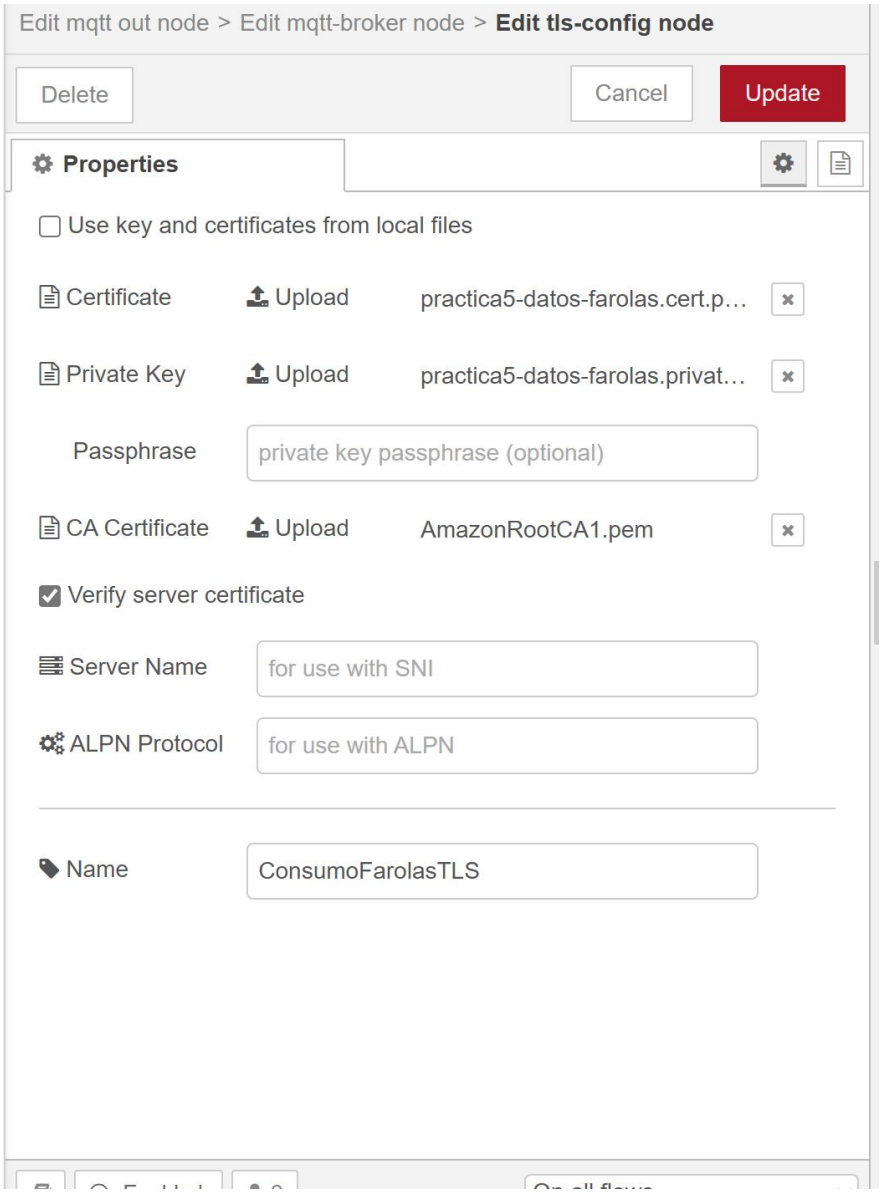


Rellenaremos los datos del Server a partir del kit que descargamos al crear el objeto IoT. Para ello pulsaremos en el icono del lápiz.

El campo Topic lo dejamos vacío porque lo generaremos de forma dinámica desde el código del nodo función.



Iremos cogiendo los datos del archivo Start del kit, revisando la última línea del script.



- **Verificación:** Confirme en la consola de DynamoDB que solo hay 5 registros que se actualizan constantemente, demostrando la **monitorización operacional** de las farolas. Verifcar que S3 incluye los históricos de consumo de las farolas.

II. Flujo de Contexto: Web Scraping con Lambda Serverless (RA 2)

Este flujo simula la recolección de **datos de contexto** (tarifas de energía) que no provienen de un *stream* continuo.

Paso 1: Creación de la Función Lambda (Código Mínimo)

1. **Creación de la Función:** Cree la función **`scraper-tarifas-luz`**
    - **Variables de Entorno:** Añada **`S3_BUCKET_NAME`** con el valor de su *bucket* S3 (*vigia-smartcity-raw-xx/*).
    - **Justificación:** Lambda es la forma **Serverless** de ejecutar tareas programadas (*Web Scraping*).
2. **Código Python (para simular y escribir a S3):**

```
import json
import datetime
import boto3
import os

S3_BUCKET = os.environ.get('S3_BUCKET_NAME')
s3_client = boto3.client('s3')

def lambda_handler(event, context):
    # 1. Simular la obtención del dato (Precio del kWh)
    precio_simulado = 0.15 + (datetime.datetime.now().minute / 600)

    data_json = {
        'timestamp_scraper': datetime.datetime.now().isoformat(),
        'tarifa_kwh': round(precio_simulado, 4),
        'ciudad': 'Neo-Tech'
    }

    # 2. Escritura en S3 (Destino del Web Scraping)
    file_key = f'tarifas/{datetime.date.today().isoformat()}/data.json'
```

```
    s3_client.put_object(
        Bucket=S3_BUCKET,
        Key=file_key,
        Body=json.dumps(data_json)
    )

    return {'statusCode': 200, 'body': 'Tarifas guardadas en S3'}
```

Paso 2: Orquestación Serverless (EventBridge)

1. **Amazon EventBridge:** Configure una regla de **EventBridge** (Programación) para invocar a la función **scraper-tarifas-luz** (ej. cada 1 minuto para probar).
    - Se puede crear desde la interfaz del servicio Lambda o desde el servicio Amazon EventBrdige
    - **Justificación: EventBridge** es el servicio *cron serverless* para automatizar la ejecución de tareas periódicas.

III. Flujo Masivo: Transformación y Análisis Serverless

Esta fase cumple el requisito de correlacionar con los **datos históricos de movilidad**.

Paso 1: Carga y Catalogación (RA 1)

1. **Carga Masiva:** Cargue el *dataset* histórico masivo de **Movilidad/Tráfico** (CSV/JSON) en S3: vigia-smartcity-raw-xx/historico_movilidad/.
   Usaremos un DataSet de la movilidad de taxis de NY descargando el archivo desde Kaggle. En la práctica se deja una muestra con menos datos para evitar el consumo excesivo de tarifa de AWS (1000 registros en lugar de 10.000.000).
   NYC Taxi Trip Records from JAN 2023 to JUN 2023

2. **AWS Glue Crawler:** Ejecute un **Crawler** sobre esa ruta.
   - **AWS Glue Crawler** es un paso fundamental en la práctica porque establece el **esquema y metadatos** de tu Data Lake, haciéndolo consultable. Es un servicio completamente *serverless* y automatiza lo que antes se hacía manualmente con Hive o Spark.
     El objetivo de este paso es que el **Crawler** examine los archivos CSV del *dataset* de Movilidad que subiste a S3, infiera automáticamente la estructura de los datos (nombres de columnas y tipos de datos), y registre ese esquema en el **AWS Glue Data Catalog**. Sin este catálogo, Amazon Athena no sabría cómo leer ni consultar los archivos en tu Data Lake.
     Aquí se muestra el proceso detallado paso a paso para ejecutar el Crawler sobre el *dataset* de Movilidad/Tráfico de S3:

     Creación del Crawler

     1. **Acceso al Servicio:** Vaya a la consola de AWS y busque **AWS Glue**. En el menú de navegación de la izquierda, expanda la sección **Catálogo de datos** (Data Catalog) y vaya a los Rastreadores (Crawlers) y haga clic en **Crear rastreador** (Create crawler).
     2. **Configuración del Origen de Datos:**

- **Nombre:** Asigne un nombre descriptivo, como vigia_movilidad_crawler.
- **Tipo de origen:** Seleccione **Datos que se encuentran en S3**.
- **Ruta de Inclusión:** Haga clic en **Agregar una fuente de datos** y especifique la ruta exacta de su *dataset* en S3: s3://vigia-smartcity-raw-xx/historico_movilidad/ (cuidado con las tildes).

3. **Configuración de IAM (Rol de Servicio):**
   - **Acción:** Seleccione un Rol de IAM existente o cree uno nuevo si el laboratorio lo permite (el rol del lab).
   - **Justificación:** AWS Glue necesita un **Rol de Servicio** para obtener permisos de lectura de S3 y permisos de escritura en el Data Catalog.

Configuración del Destino (Data Catalog)

1. **Configuración de la Salida:** En la sección **Propiedades de configuración del rastreador** (Crawler configuration properties), defina dónde se guardarán los metadatos.
   - **Base de datos:** Cree una nueva base de datos para la práctica (ej: vigia_datalake_db).
   - **Prefijo de la Tabla:** Opcional. Puede usar pr5_movilidad_ o movilidad_ para que las tablas comiencen con ese prefijo.

Ejecución del Crawler

1. **Ejecución:** Una vez que la configuración esté completa, haga clic en **Finalizar** (Finish) y luego haga clic en el Crawler recién creado y seleccione **Ejecutar rastreador** (Run crawler).
2. **Monitoreo:** El proceso puede tardar unos minutos, ya que Glue está inspeccionando el archivo CSV de gran tamaño, infiriendo el esquema y determinando si el formato es correcto.
3. **Verificación:** Una vez que el estado cambie a **"Listo"** (Ready), vaya a **Bases de Datos** (Databases) → **vigia_datalake_db** → **Tablas** (Tables). Debería ver una nueva tabla (ej: movilidad_historico_movilidad) con todas las columnas detectadas (como tpep_pickup_datetime, PULocationID, etc.).

Nota: Debería crear una tabla con las columnas de la cabecera. Debéis detectar qué error existe en la cabecera del CSV y ver por qué no ha podido crearla. Manualmente podemos editar el schema de la tabla y cambiar el nombre de las columnas
Optativo para subir nota:

- Crear un clasificador para resolverlo de forma automática, borrar la tabla, editar el Crawler y asociarle el clasificador en el paso 2.

Creación del Clasificador Personalizado

1. **Acceso a Glue:** Vaya a la consola de **AWS Glue**.
2. **Ruta:** En el menú de la izquierda, expanda **Catálogo de datos** (Data Catalog) → **Clasificadores** (Classifiers).
3. **Crear Clasificador:** Haga clic en **Crear clasificador**.
    - **Nombre:** CSV_Header_Movilidad
    - **Tipo de Clasificador: CSV**.
    - **Delimitador:** Generalmente, la coma (,).
    - **Comillas:** (Dejar por defecto, generalmente comillas dobles ").
    - **CRÍTICO: La Primera Fila es el Encabezado: Marque la casilla** que indica que la primera fila del archivo CSV contiene los nombres de las columnas.

      Investigar qué listado de cabeceras deberíais incluir en base a la localización del error.

    - **Guardar:** Haga clic en **Crear**.

Ejecutar el Crawler con el Clasificador Personalizado

Ahora, el Crawler debe usar tu nuevo clasificador, el cual le indica cómo interpretar la primera fila del archivo CSV de movilidad.

1. **Edición del Crawler:** Vaya a **Catálogo de datos** $\rightarrow$ **Rampas de rastreo** (Crawlers). Seleccione y edite el Crawler **vigia_movilidad_crawler**.

2. **Añadir Clasificador:** En la sección de configuración del Crawler, busque **Clasificadores personalizados** (Custom classifiers) y **seleccione** CSV_Header_Movilidad de la lista.
3. **Restablecer Estado:** Para que el Crawler procese el esquema de nuevo, marque la opción **"Volver a rastrear todos los directorios"** (si está disponible) o elimine la tabla creada anteriormente (si el *lab* lo permite).
4. **Ejecutar:** Ejecute el Crawler (vigia_movilidad_crawler) de nuevo.

Paso 2: Optimización del Data Lake (AWS Glue ETL) (RA 2)

1. **Creación y Propiedades del Job**
   - **Acceso:** Vaya a AWS Glue → **Trabajos** (Jobs) → **Crear trabajo** (Create job).
   - **Tipo de Creación:** Selecciona **Editor de scripts de Python Shell/PySpark** (Script editor).
   - **Configuración del Tipo de Job (Job Details):**
     - **Tipo de Job:** Selecciona **Spark**.
     - **Lenguaje: Python (PySpark)**.
     - **Versión de Glue:** Elige la versión más reciente compatible (ej., Glue 4.0 o 3.0).
     - **Rol de IAM:** Asigna el **Rol de Servicio** preexistente para Glue (ej., AWSGlueServiceRole-Vigia).

2. **Paso 2: El Script PySpark (Transformación)**
   Una vez que el Job se crea, AWS te dirigirá al editor de **Script** donde debes pegar el código de transformación Serverless:

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
```

```
from pyspark.sql.functions import year, month, col, to_timestamp, lit
from pyspark.sql.types import DoubleType, StringType
from awsglue.dynamicframe import DynamicFrame # NECESARIO para la conversión

# --- 1. Inicialización y Contexto ---
glueContext = GlueContext(SparkContext.getOrCreate())

# --- 2. Configuración de Rutas y Nombres ---
# ATENCIÓN: Reemplaza "VIGIA-SMARTCITY-RAW-XX" y "VIGIA-SMARTCITY-PROCESSED-XX" con tus nombres de bucket reales.
DATABASE_NAME = "vigia_datalake_db"
TABLE_NAME = "historico_movilidadhist_rico_de_movilidad"
S3_PROCESSED_PATH = "s3://vigia-smartcity-processed-xx/analitica/"

# --- 3. Extracción: Leer la tabla RAW del Glue Data Catalog ---
datasource = glueContext.create_dynamic_frame.from_catalog(
    database=DATABASE_NAME,
    table_name=TABLE_NAME
)

# Convertir a DataFrame (Necesario para usar funciones SQL y manipulación de columnas)
df = datasource.toDF()

# --- 4. Transformación (T): Limpieza, Conversión de Tipos y Partición ---

# 4.1. Conversión de Fecha (CRÍTICO)
# El formato de fecha estándar de este dataset es YYYY-MM-DD HH:MM:SS
df_transform = df.withColumn(
    "pickup_time_ts",
```

```
    to_timestamp(col("tpep_pickup_datetime"), "yyyy-MM-dd HH:mm:ss")
)

# 4.2. Selección y Casting: Prepara los campos clave para el análisis
df_final = df_transform.select(
    # --- CAMPOS DE PARTICIÓN ---
    year(col("pickup_time_ts")).alias("year"),
    month(col("pickup_time_ts")).alias("month"),

    # --- CAMPOS CRÍTICOS PARA LA CORRELACIÓN Y ANÁLISIS ---
    col("PULocationID").cast(StringType()).alias("PULocationID"),  # ID de Zona (Correlación con farolas)
    col("DOLocationID").cast(StringType()).alias("DOLocationID"),
    col("trip_distance").cast(DoubleType()).alias("trip_distance"),  # Métrica de Distancia
    col("fare_amount").cast(DoubleType()).alias("fare_amount"), # Monto de Tarifa
    col("total_amount").cast(DoubleType()).alias("total_amount"),  # Monto Total

    # --- OTROS CAMPOS DE VALOR ---
    col("VendorID").alias("VendorID"),

col("passenger_count").cast(DoubleType()).alias("passenger_count")
)

# 4.3. Conversión de vuelta a DynamicFrame (Soluciona TypeError)
dynamic_frame_output = DynamicFrame.fromDF(
    df_final,
    glueContext,
    "dynamic_frame_output"
)
```

```
# --- 5. Carga (L): Escritura Optimizada a S3 ---

glueContext.write_dynamic_frame.from_options(
    frame=dynamic_frame_output, # Usa el DynamicFrame convertido
    connection_type="s3",
    connection_options={
        # ¡CORRECCIÓN! Ruta al destino PROCESSED
        "path": S3_PROCESSED_PATH,
        "partitionKeys": ["year", "month"] # Particionamiento CRÍTICO
    },
    format="parquet"
)
```

Campos Clave del *Dataset* de Movilidad para la Práctica

El *dataset* de viajes en taxi (TLC Trip Record Data) suele incluir docenas de columnas. Nos enfocaremos en cuatro para el ETL.

Campo (Nombre Original)	Propósito en el Data Lake	Justificación para Glue/Athena
tpep_pickup_datetime o **lpep_pickup_datetime**	**Marca de Tiempo de Inicio del Viaje.**	**CRÍTICO para la Partición:** Glue debe extraer el **año, mes y día** de este campo para crear las carpetas de particionamiento en S3, optimizando las consultas.
PULocationID (Pickup Location ID) o **DOLocationID** (Dropoff Location ID)	**Ubicación (Geoespacial)**	**CRÍTICO para la Correlación:** Estos IDs representan zonas. Permiten a Athena correlacionar la **densidad de tráfico** de una zona con el consumo energético de las farolas en esa misma zona.
trip_distance	Distancia total recorrida.	**Métrica de Análisis:** Útil para que Athena calcule métricas básicas (ej. distancia total recorrida por día/zona) para simular una consulta *Big Data*.

Proceso de Transformación (AWS Glue ETL)

El **Job de AWS Glue** debe realizar la siguiente transformación en el *dataset* de movilidad:

3. **Ingesta (RAW):** El CSV completo se carga en s3://vigia-smartcity-raw-xx/historico_movilidad/.
4. **Transformación (Glue):**
    - Leer el archivo CSV masivo.
    - Convertir el campo de fecha (tpep_pickup_datetime) a un formato de tiempo estándar.
    - Extraer las columnas de partición: year, month, day.
    - **Escribir en Parquet:** Escribir el *dataset* optimizado a la capa *Processed* (s3://vigia-smartcity-processed-xx/analitica/) usando year, month, y day como claves de partición.

Por último, debemos generar un Crawler que lea los archivos **Parquet optimizados** que tu Job ETL ya escribió en S3 y crear la tabla analitica_movilidad en el Glue Data Catalog.

1. Creación del Crawler PROCESADO

- **Acceso a Glue:** Vaya a **AWS Glue** $\rightarrow$ **Catálogo de datos** $\rightarrow$ **Rampas de rastreo** (Crawlers).
- **Crear Crawler:** Haga clic en **Crear rastreador**.
- **Nombre:** Asigne el nombre **vigia_analitica_crawler**.
- **Ruta de Inclusión (DataSource):** Especifique la ruta de destino de su Job ETL (donde se guardaron los archivos Parquet):
- s3://vigia-smartcity-processed-XX/analitica_movilidad/
- **Rol de IAM:** Seleccione el Rol de Servicio preexistente para Glue.
- **Base de Datos de Salida:** Seleccione la base de datos **vigia_datalake_db**.
- **Tabla: CRÍTICO:** En la configuración de salida, asegúrese de que la tabla que se creará se llame **analitica_movilidad**.

2. Ejecución y Finalización del Flujo Serverless

- **Ejecutar Crawler:** Ejecute el Crawler **vigia_analitica_crawler**. Este proceso leerá los archivos Parquet particionados por /year=YYYY/month=MM/ y creará el esquema en el Catálogo.

Paso 3: Análisis y Verificación de la Optimización (RA 2)

1. **Configuración Inicial de Amazon Athena (Ruta Actualizada)**
    - **Acceso:** Vaya a la consola de **Amazon Athena**.
    - **Abrir Configuración:** En la parte superior de la pantalla, busca y haz clic en el menú desplegable que dice **"Workgroup: primary"** (Grupo de trabajo: principal) y selecciona **"View details"** (Ver detalles).
    - **Ir a Configuración de Consultas:** Se abrirá una nueva ventana. En la pestaña **"Query results configuration"** (Configuración de resultados de la consulta).
    - **Especificar Ubicación de Resultados:**

        Acción: Ingresa la ruta completa a una carpeta vacía en uno de tus *buckets* de S3.

        Ejemplo: s3://vigia-smartcity-raw-XX/athena-results/
    - **Guardar:** Haz clic en **"Guardar"** (Save).

    Resultado: Al guardar, Athena ya sabe dónde almacenar los resultados de tus consultas y está lista para que consultes el **Glue Data Catalog** y realices la comparación final.

2. Seleccionar el Catálogo de Datos

    Athena lee la estructura de tus tablas de AWS Glue Data Catalog.

Paso	Acción en la Interfaz de Athena
A. Volver al Editor	Vuelve al **Editor de consultas** (Query editor).

Paso	Acción en la Interfaz de Athena
B. Seleccionar Fuente de Datos	En el panel de la izquierda, asegúrate de que el **Origen de datos** (Data source) esté en **AwsDataCatalog**.
C. Seleccionar Base de Datos	En el menú desplegable **Base de datos** (Database), selecciona la base de datos que creaste: **vigia_datalake_db**.

Resultado: Verás dos tablas en el panel lateral: historico_movilidad (CSV RAW) y analitica_movilidad (Parquet PROCESSED).

3. **Amazon Athena (Consulta Serverless), ejecución comparativa rendimiento:**

Ahora ejecutarás las dos consultas para medir la eficiencia Serverless.

A. Consulta A: La Línea Base (CSV RAW)

Esta consulta escanea el archivo CSV completo. Observa el valor de "Datos Escaneados" (Data Scanned) al finalizar.

SQL

```
-- CONSULTA A: Sobre la tabla RAW (CSV sin optimizar)
SELECT
    COUNT(trip_distance)
FROM
    "vigia_datalake_db"."historico_movilidad"
WHERE
    tpep_pickup_datetime LIKE '2023-06%';
```

Ejecución: Pega la consulta, haz clic en Ejecutar (Run).

B. Consulta B: La Optimización Serverless (Parquet PROCESSED)

Esta consulta utiliza las columnas de partición (year y month) y el formato Parquet.

SQL

```
-- CONSULTA B: Sobre la tabla PROCESADA (PARQUET particionado)
SELECT
    COUNT(trip_distance)
FROM
    "vigia_datalake_db"."analitica_movilidad"
```

```
WHERE
    year = 2023
    AND month = 6;
```

Ejecución: Pega la consulta, haz clic en Ejecutar (Run).

Conclusión Final

Compara los resultados de Datos Escaneados para la Consulta A y la Consulta B. La Consulta B (Parquet) debería mostrar un valor significativamente menor (ej., 90% menos).

Esto demuestra que:

- AWS Glue hizo su trabajo Serverless (ETL).
- Athena (RA 2) puede ejecutar consultas de Big Data de manera eficiente y a bajo costo, justificando el diseño de tu Data Lake.