

Taller: Procesamiento y Análisis de Datos para una Aerolínea

Consigna para el Taller de Alumnos

Título: Análisis de Retrasos de Vuelos y Satisfacción de Clientes

Descripción:

Imagina que trabajas en el área de análisis de datos de una aerolínea. Se te asigna la tarea de integrar datos operativos de vuelos y datos de pasajeros para generar un informe sobre el nivel de ocupación de los vuelos. Este informe se utilizará para optimizar la asignación de recursos y mejorar las operaciones de la aerolínea.

Datos de Entrada

1. **`flights.csv`**: Contiene información sobre los vuelos realizados, como el número de vuelo, origen, destino, y retrasos en minutos.
2. **`passengers.csv`**: Contiene opiniones de los pasajeros sobre su experiencia, incluyendo una calificación de 1 a 5.

Objetivo:

Desarrollar un proceso automatizado para combinar los datos de los archivos proporcionados donde:

- Combine los dos archivos utilizando el número de vuelo (``flight_number``) como clave.
- Calcula la cantidad de pasajeros por vuelo (``passenger_count``).
- Calcula el porcentaje de ocupación de cada vuelo (``occupancy_rate``) utilizando la fórmula: $\text{occupancy_rate} = (\text{passenger_count} / \text{capacity}) * 100$

Así mismo, el dataframe resultante lo deberá de ingestar en Redshift en la tabla **flights_analysis** y conectar la metadata creada con Glue.

1. Ejecución:

- Crear el bucket S3 s3-training-activity-01.
- La inserción de datos en Redshift se va a realizar desde la consola, con un evento de testing; el objetivo es analizar la construcción de la Lambda y la lógica del código implantado.

2. Generación y escritura de datos procesados en Redshift:

- El DataFrame resultante (`merged_df`), producto de la combinación de los datos, debe ser escrito en Redshfit en la tabla **flights_analysis**.

3. Conectar nueva tabla Redshift con Glue

En esta sección, configuraremos una integración entre Glue y Redshift para gestionar la metadata de manera eficiente. Sigue los pasos a continuación:

Paso 1: Crear una nueva base de datos en Glue

- Crea una nueva base de datos en Glue y asígnale el nombre: training-metadata-redshift.
- Esta base de datos actuará como la fuente de metadata conectada con Redshift.

Paso 2: Crear un conector de tipo Redshift

1. Proporciona el nombre del clúster de Redshift.
2. Especifica la base de datos ya creada en Redshift, en este caso, dev.
3. Utiliza las credenciales de Redshift gestionadas a través de **AWS Secrets Manager**.
4. Asigna el rol IAM predeterminado, **LabRole**.
5. Define un nombre para el conector, por ejemplo: tr-redshift-connection.
6. Una vez creado, prueba la conexión desde Glue para garantizar que todo esté configurado correctamente.

Paso 3: Crear un crawler de tipo Redshift

1. Asigna un nombre al crawler, por ejemplo: tr-redshift-crawler.
2. Como es la primera vez, no mapearemos datos; este paso puede ser omitido.
3. Añade un nuevo origen de datos del tipo **JDBC Redshift**, utilizando el conector creado en el paso anterior.
4. En el **path**, define la ruta como: dev/%.
5. Asigna el rol IAM predeterminado, **LabRole**.
6. Como base de datos de destino, selecciona training-metadata-redshift, creada inicialmente en Glue.
7. Configura la frecuencia del crawler como **On-demand** (bajo demanda).
8. Después de crear el crawler, ejecútalo desde la consola de Glue.

Resultado esperado:

Una vez finalizado el proceso de ejecución del crawler tr-redshift-crawler, deberías poder visualizar la metadata de Redshift (específicamente de la base de datos dev) dentro de la base de datos training-metadata-redshift en Glue.

Código:

Este código debes copiarlo en tu lambda:

```
import os
import json
```

```

import pandas as pd
import awswrangler as wr

# Variables de entorno
S3_BUCKET = os.environ["S3_BUCKET"]
FLIGHTS_KEY = os.environ["FLIGHTS_KEY"]
PASSENGERS_KEY = os.environ["PASSENGERS_KEY"]
REDSHIFT_CLUSTER = os.environ["REDSHIFT_CLUSTER"]
REDSHIFT_DATABASE = os.environ["REDSHIFT_DATABASE"]
REDSHIFT_USER = os.environ["REDSHIFT_USER"]
REDSHIFT_PASSWORD = os.environ["REDSHIFT_PASSWORD"]
REDSHIFT_TABLE = os.environ["REDSHIFT_TABLE"]
REDSHIFT_SCHEMA = os.environ.get("REDSHIFT_SCHEMA", "public")

def lambda_handler(event, context):
    try:
        flights_path = f"s3://{S3_BUCKET}/{FLIGHTS_KEY}"
        passengers_path = f"s3://{S3_BUCKET}/{PASSENGERS_KEY}"

        flights_df = wr.s3.read_csv(flights_path)
        passengers_df = wr.s3.read_csv(passengers_path)

        merged_df = pd.merge(
            flights_df,
            passengers_df.groupby("flight_number")
                .size()
                .reset_index(name="passenger_count"),
            on="flight_number",
        )
        merged_df["occupancy_rate"] = (
            merged_df["passenger_count"] / merged_df["capacity"]
        ).fillna(0) * 100

        print(merged_df)

        con = wr.redshift.connect_temp(
            cluster_identifier=REDSHIFT_CLUSTER,
            database=REDSHIFT_DATABASE,
            user=REDSHIFT_USER,
        )

        wr.redshift.copy(
            df=merged_df,
            path=f"s3://{S3_BUCKET}/processed/temp",
            con=con,
            schema=REDSHIFT_SCHEMA,
            table=REDSHIFT_TABLE,
            iam_role=os.environ["IAM_ROLE"],
            mode="append",

```

```

    )

    return {
        "statusCode": 200,
        "body": json.dumps(
            {"message": "Proceso completado", "rows_inserted": len(merged_df)}
        ),
    }

except Exception as e:
    return {
        "statusCode": 500,
        "body": json.dumps({"message": "Error", "error": str(e)}),
    }

```

Variables:

Variable	Valor
S3_BUCKET	s3-training-activity-01
FLIGHTS_KEY	flights-01.csv
PASSENGERS_KEY	passengers-02
REDSHIFT_CLUSTER	redshift-cluster-training
REDSHIFT_DATABASE	dev
REDSHIFT_USER	awsuser
REDSHIFT_PASSWORD	
REDSHIFT_TABLE	flights_analysis
REDSHIFT_SCHEMA	public
IAM_ROLE	arn:aws:iam::933263644347:role/myRedshiftRole