

# Prueba Técnica – Ingeniería de Datos

---



# DataKnow

**Agradecimientos:** Muchas gracias a los evaluadores por leer este contenido.

**Autor:** Juan Camilo Barrero Velásquez

**Correo:** jcbarrerov@unal.edu.co

**Fecha:** 01/12/2025

**Github de este repositorio** <https://github.com/jcbarrerov/pruebaTecnicaDK> **LinkedIn**

[www.linkedin.com/in/juan-camilo-barrero-velasquez-engineer](http://www.linkedin.com/in/juan-camilo-barrero-velasquez-engineer)

---

## Configuración del Entorno

1. Clona este repositorio en tu máquina local con el siguiente código:

```
git clone https://github.com/jcbarrerov/pruebaTecnicaDK
```

2. Navega al directorio del proyecto:

```
cd pruebaTecnicaDK
```

3. Crea un entorno virtual para este proyecto:

```
python -m venv venv
```

4. Activa el entorno virtual:

En Windows:

```
venv\Scripts\activate
```

En macOS y Linux:

```
source venv/bin/activate
```

5. Instala las dependencias del proyecto:

```
pip install -r requirements.txt
```

## 📁 Tabla de Contenido

- Prueba Técnica – Ingeniería de Datos
- Quest 1: Carga de Información
  - Solución
    - Extracción - Lectura del archivo
    - Transformación - Procesamiento de la información
    - Carga - Creación del DataFrame y exportación del documento CSV
    - Ejecución del ETL
    - Resultados
- Quest 2: Manipulación de datos
  - 1. Cargar el data set
  - 2. Filtrar por agente y tipo de central
  - 3. Cargar el archivo ddec1204txt
  - 4. Merge por CENTRAL
  - 5. Suma horizontal
  - 6. Filtrar por suma > 0
  - 7. Exportación del dataset
- Quest 3: Prueba de SQL
  - Parte 1: Tabla CLIMA
  - Parte 2: Mejoras para alto volumen
  - Parte 3: Tabla CLIMA\_DIA
  - Parte 4: Delta de temperaturas (ventanas)
- Quest 5: Prueba Azure
  - Arquitectura propuesta
  - Data Factory
  - KPIs para la base de datos de Adventure Works
    - KPI Ventas Totales
    - KPI Margen Bruto Porcentaje

- KPI 3. Valor Promedio Por Pedido
  - KPI 4. Tasa entregas a tiempo
  - KPI 5. Top 5 Productos Mes
  - DataFlows
  - Resultados
  - Quest 7: Arquitectura
    - Explicación resumida de la arquitectura
    - Gobierno de Datos y Modelos
  - ⚒ Herramientas Usadas
  - 📖 Bibliografía
- 

## Quest 1: Carga de Información

---

Cargar un data set, realizar el cargue y depuración del archivo OFEI1204.txt. Se debe entregar una tabla con las siguientes columnas:

Agente	Planta	Hora_1	Hora_2	Hora_3	...	Hora_24
--------	--------	--------	--------	--------	-----	---------

Solamente procesar los registros Tipo D. Enviar junto con la tabla resultante el código utilizado. Explicar el paso a paso en un archivo de texto (.doc o .pdf).

## Solución

Para la solución de este punto implementó de una filosofía de programación modular en la que se establecieron tres procesos principales: extracción, transformación y carga.

### Extracción - Lectura del archivo

Para este proceso de lectura se diseñó la función *read\_file* que se encarga de usar la ruta del archivo a leer, `filepath`, para luego abrir el archivo utilizando un modo de lectura "r" con codificación "utf-8", almacenar el contenido en la variable `content`, imprimir un mensaje de indicando el éxito de la operación y finalmente retornarlo

```
def read_file(filepath: str) -> str:  
    with io.open(filepath, "r", encoding="utf-8") as f:  
        content = f.read()  
    print("File read successfully!")  
    return content
```

### Transformación - Procesamiento de la información

En este módulo del código se establecen dos funciones, `extract_date` y `get_rows`.

En la función `extract_date` se recibe el parametro `file`, que será el string resultante del proceso de extracción sobre el cual se usa la expresión regular para buscar la fecha de formato "YYYY-MM-DD". En caso de encontrar la fecha imprime un mensaje de éxito y retorna el valor de la fecha en la variable `date`, en caso de no encontrarla, retorna `None`. Este dato será utilizado más adelante en el nombre del archivo generado.

```
def extract_date(file:str) -> str:
    date = re.search(r'\d{4}-\d{2}-\d{2}', file)
    print("Date extracted successfully!")
    return date.group() if date else None
```

En la función `get_rows` se recibe nuevamente el parametro `file`, que se le asignará de la misma manera que en el proceso anterior. Luego, divide el texto en bloques usando el patrón "\n\n\nAGENTE:" presente en el texto. Almacena el nombre del agente que estará presente en las filas venideras correspondientes al bloque e itera por las lineas del bloque.

En la iteración intenta dividir la información de la fila en las respectivas columnas usando la coma como patrón, si el segundo elemento es " D" entonces almanecena la fila en la variable `complete_line` teniendo como primer elemento el agente y lo agrega a la lista `rows` luego de dividir cada elemento de la cadena por comas. `rows` es por lo tanto una lista de listas donde cada elemento dentro de la lista principal es la lista correspondiente a la información de una fila con sus elementos (que son cadenas de texto) separados.

Todo lo descrito en el parrafo anterior se encuentra dentro de un `try`, que en caso de un `IndexError` se encargará de imprimir una advertencia en consola estableciendo la linea que ha de ser excluida y continuando la iteración. Todo esto, con el fin de poder hacer una revisión completa de la cadena.

Finalmente la función imprime un mensaje de éxito y retorna la variable `rows` con las filas que cumplen la condición establecida.

```
def get_rows(file:str) -> List[List[str]]:
    rows = []
    for block in file.split("\n\n\nAGENTE: "):
        agente = block.split("\n")[0]

        for line in block.split("\n")[1:]:
            try:
                if line.split(",")[1] == " D":
                    complete_line = [agente + ", " + line]
                    rows.append(complete_line[0].split(","))
            except IndexError:
                print(f"WARNING! line being excepted: '{line}'")
    print("Rows obtained successfully!")
    return rows
```

## Carga - Creación del DataFrame y exportación del documento CSV

Este módulo está compuesto de dos funciones `create_dataframe` y `load_csv`.

En la función `create_dataframe` recibe los parámetros `rows` (una lista de listas de strings) resultante del módulo de extracción y `columns` correspondiente a una lista que contiene los caracteres correspondientes a los nombres de las columnas.

Luego se crea un DataFrame de pandas con las filas y columnas establecidas en los parámetros, utiliza el método `map` que permite retirar los espacios extra al inicio y fin de cada valor si es un string y finalmente imprime un mensaje estableciendo el éxito de la ejecución y retorna el DataFrame.

```
def create_dataframe(rows:List[List[str]], columns:List[str]) -> pd.DataFrame:
    df = pd.DataFrame(rows, columns=columns)
    df = df.map(lambda x: x.strip() if isinstance(x, str) else x)
    print("DataFrame created successfully!")
    return df
```

La función `load_csv` tiene los parámetros `df` (el DataFrame retornado en la función anterior), `date` (la fecha obtenida en la función `extract_date`) y `path` (la ruta donde el archivo va a ser guardado). En este caso el `path` no es un parámetro obligatorio, de ser especificado lo usa para guardar el archivo dentro del directorio.

Esta función genera el archivo CSV sin índices con el nombre "`OFFEI_cleaned_{date}.csv`", donde `{date}` es reemplazado por la fecha. Finalmente imprime un mensaje indicando que el archivo se generó correctamente.

```
def load_csv(df:pd.DataFrame, date:str, path:str | None = None) -> None:
    if path == None:
        path_csv = f"OFFEI_cleaned_{date}.csv"
    else:
        path_csv = f"{path}/OFFEI_cleaned_{date}.csv"
    df.to_csv(path_csv, index=False)
    print("CSV file loaded successfully!")
```

## Ejecución del ETL

Finalmente se realiza la importación de las funciones de los módulos y se establece la ruta dónde se encuentra el archivo. Se llaman las funciones con los argumentos correspondientes y asignamos el retorno de las funciones a las variables necesarias para seguir el proceso de ETL

```
import os

from module.extract import read_file
from module.transform import extract_date, get_rows
from module.load import create_dataframe, load_csv

PATH = os.path.abspath("../data/OFEI1204.txt")

if __name__ == '__main__':
    file = read_file(PATH)
```

```
date = extract_date(file)
rows = get_rows(file)
columns = ["Agente", "Planta", "Tipo"] + ["Hora_{}".format(i) for i in
range(1, 25)]
df = create_dataframe(rows, columns)
load_csv(df, date)
```

## Resultados

El resultado final de esta prueba está en el siguiente archivo [OFFEI\\_cleansed\\_2017-12-04.csv](#)

---

## Quest 2: Manipulación de datos

---

1. Cargar un data set, del archivo Excel Master Data, únicamente las siguientes columnas:
  - Nombre visible Agente
  - AGENTE (OFEI)
  - CENTRAL (dDEC, dSEGDES, dPRU...)
  - Tipo de central (Hidro, Termo, Filo, Menor)
2. Seleccionar los registros que pertenecen al agente EMGESÁ ó EMGESÁ S.A. y adicionalmente que el Tipo de Central sea 'H' o 'T'.
3. Cargar el archivo dDEC1204.TXT que viene por Central.
4. Realizar el merge de los dos data sets por Central.
5. Calcular la suma horizontal de todas las horas para cada planta.
6. Seleccionar solamente los registros de las plantas cuya suma horizontal sea mayor que cero.
7. Los resultados deben ser entregados en un dataset.
8. Enviar junto con la tabla resultante el código utilizado.
9. Explicar el paso a paso en un archivo de texto (.doc o .pdf).

## Solución

Para trabajar este problema se utilizó la ayuda de *Jupyter notebooks* debido a su facilidad para seguir el flujo de las tareas solicitadas y visualizar los dataframes. Para este programa usaremos la librería *os*, para poder interpretar las rutas relativas de los archivos, y *pandas* para trabajar con los datos tabulares. Adicionalmente definimos las rutas relativas de los archivos necesarios para el código.

```
import os
import pandas as pd

PATH_EXCEL = os.path.abspath("../data/Datos Maestros VF.xlsx")
```

```
PATH_TXT = os.path.abspath("../data/dDEC1204.TXT")
PATH_TO_SAVE = os.path.abspath("./Dataset.csv")
```

## 1. Cargar el data set

Para cargar el dataset se utilizó el metodo `read_excel` de pandas en modo binario '`rb`' especificando en nombre de la hoja que contiene la información y el DataFrame resultante se asignó a la variable `df_excel_raw`.

Adicionalmente se creó un diccionario que contiene pares clave-valor con los nombres de las columnas del DataFrame y nombres simplificados sin espacio respectivamente para posteriormente renombrar las columnas.

Utilizando un `for` que recorre cada par del diccionario se renombran las columnas del DataFrame `df_excel_raw` sin necesidad de crear un nuevo DataFrame gracias al `inplace=True`.

Luego, se seleccionan los datos del DataFrame de las columnas:

- Nombre visible Agente que es `AGENTE_VISIBLE`
- AGENTE (OFEI) que es `AGENTE_OFEI`
- CENTRAL (dDEC, dSEGDES, dPRU...) que es `CENTRAL`
- Tipo de central (Hidro, Termo, Filo, Menor) que es `TIPO_CENTRAL`

utilizando la variable `select_columns_excel` que contiene los valores de las columnas a seleccionar del DataFrame se realiza la selección de los valores del DataFrame `df_excel_raw` creando una copia. El nuevo DataFrame es asignado a la variable `df_excel_selected`.

```
df_excel_raw = pd.read_excel(open(PATH_EXCEL, 'rb'),
                             sheet_name='Master Data Oficial')

dic_columns = { 'Nombre visible Agente':'AGENTE_VISIBLE',
                 'AGENTE (OFEI)':'AGENTE_OFEI',
                 'CENTRAL (dDEC, dSEGDES, dPRU...)':'CENTRAL',
                 'Tipo de central (Hidro, Termo, Filo, Menor)':'TIPO_CENTRAL' }

for name, rename in dic_columns.items():
    df_excel_raw.rename(columns={name: rename}, inplace=True)

select_columns_excel = list(dic_columns.values())
df_excel_selected = df_excel_raw[select_columns_excel].copy()
```

## 2. Filtrar por `AGENTE_VISIBLE` y `CENTRAL`

Para esta sección se filtró el DataFrame `df_excel_selected` por tres condiciones:

1. `['AGENTE_VISIBLE'] == "EMGESÁ"` Selecciona filas donde el agente tiene nombre visible "EMGESÁ".
2. `['AGENTE_OFEI'] == "EMGESÁ S.A."` Selecciona filas donde el `AGENTE_OFEI` coincide con "EMGESÁ S.A.".

3. `['TIPO_CENTRAL'].isin(['H', 'T'])` Filtra solo los registros cuyo tipo de central sea H o T utilizando `isin` y la lista con los valores deseados.

La condición 1 y 2 están vinculadas por una condición `or`, mientras que estas dos están ligadas por una condición `and` con la 3. Finalmente El DataFrame resultante se asigna a la variable `df_excel_filtered`.

```
df_excel_filtered = df_excel_selected[((df_excel_selected['AGENTE_VISIBLE'] == "EMGESAS.A.") & (df_excel_selected['AGENTE_OFEI'] == "EMGESAS.A.")) | (df_excel_selected['TIPO_CENTRAL'].isin(['H', 'T']))]
```

### 3. Cargar el archivo dDEC1204.TXT

Para cargar el archivo dDEC1204.TXT que contiene los datos por central se creó primero una lista con las columnas correspondientes a el archivo a cargar haciendo uso de un `for` para crear los 24 elementos correspondientes a las horas, luego, se utilizó el método `read_csv` de pandas para leer el archivo utilizando `encoding="latin1"` y se asignó al DataFrame `df_text`. Finalmente, se asignaron las columnas almacenadas en la lista `columns` al DataFrame.

```
columns = ["CENTRAL"] + ["Hora_{}".format(i) for i in range(1, 25)]
df_text = pd.read_csv(PATH_TXT, encoding="latin1")
df_text.columns = columns
```

### 4. Realizar el merge de los dos data sets por CENTRAL

Para realizar el merge de los DataFrames utilizamos el método de pandas `merge` que nos permite hacer una unión de dos DataFrames. Establecemos el parámetro `on="CENTRAL"` que indica que la unión se hace por la columna "CENTRAL" presente en ambos DataFrames y establecemos que el tipo de unión como `how="left"` que es similar al de un `left join` en el cual se toman todas las filas de `df_excel_filtered` y solamente se agregan los valores de las filas correspondientes de `df_text` si el valor de la columna "CENTRAL" coincide. De esta manera se asegura de que no se pierden valores de `df_excel_filtered` en caso de que no haya coincidencia. Finalmente se asigna el DataFrame resultante a `df_merged`.

```
df_merged = pd.merge(df_excel_filtered, df_text, on="CENTRAL", how="left")
```

### 5. Calcular la suma horizontal de todas las horas para cada planta

Para realizar la suma horizontal se creó la variable tipo lista `columns_to_sum` que almacena los valores de las columnas que queremos sumar (en este caso las correspondientes a las 24 horas). A continuación, utilizamos `df_merged[columns_to_sum]` para seleccionar las columnas y aplicamos el método `sum(axis=1)` para realizar la suma, donde `axis=1` establece que la suma debe ser realizada a lo largo de las filas (suma

horizontal). Finalmente, el vector resultante es almacenado en la columna "SUM\_OF\_HOURS" que se asigna dentro del mismo dataframe `df_merged`.

```
columns_to_sum = ["Hora_{}".format(i) for i in range(1, 25)]
df_merged["SUM_OF_HOURS"] = df_merged[columns_to_sum].sum(axis=1)
```

## 6. Seleccionar los registros de las plantas con suma horizontal mayor que cero

Para seleccionar los registros se utilizó la condición `df_merged["SUM_OF_HOURS"] > 0` que nos otorga un vector de booleanos que establecen si se cumple la condición, que la columna "SUM\_OF\_HOURS" sea mayor que cero. Luego, se inserta el vector dentro del DataFrame `df_merged` para seleccionar las filas que cumplen la condición. Por último, se asignó el DataFrame a la variable `df_final`.

```
df_final = df_merged[df_merged["SUM_OF_HOURS"] > 0]
```

## 7. Cargar los resultados en un Data set

Para finalizar este ejercicio se utilizó el método `to_csv` para cargar el DataFrame en un archivo de valores separados por comas utilizando la variable `PATH_TO_SAVE` establecida al inicio del código como ruta.

```
df_final.to_csv(PATH_TO_SAVE)
```

## Resultados

El resultado final de esta prueba está en el siguiente archivo [Dataset.csv](#)

# Quest 3: Prueba de SQL

Utiliza cualquier dialecto de SQL de tu elección para abordar estos desafíos, de preferencia genera los datos si lo ves necesario para simular y emplear las soluciones de diseño, la idea es explicar tu solución de tal forma que técnicamente el equipo pueda ser capaz de entender y visualizar usa las herramientas que deseas además de hacer los scripts de creación dependiendo de cada parte de la prueba.

**Parte 1** Un interesado nos solicita prepararnos para una nueva fuente de datos dentro de nuestro entorno de almacenamiento de datos. La tabla recogerá información meteorológica de forma horaria para diferentes regiones. Las dimensiones y métricas de la tabla deben crearse con los tipos de datos apropiados. La tabla debe diseñarse de manera que se pueda identificar de forma única cada registro dentro de ella. Las siguientes columnas deben estar presentes en la definición de la tabla:

- Localidad (Poblados en Medellín, Envigado, Sabaneta, etc.)
- País (Colombia)
- Temperatura (Grados Celsius)
- Fecha y hora del registro (horario)

- Cobertura de nubes (Mínima, Parcial, Total)
- Índice U/V
- Presión atmosférica
- Velocidad del viento (Nudos)

**Parte 2** La tabla definida en la Parte 1 se implementa y comienza a recopilar datos. La tabla se vuelve considerablemente grande, con millones de registros. Proporciona tres maneras en que la tabla actual puede mejorarse para manejar un conjunto de datos más grande y mantener una óptima legibilidad de los datos.

**Parte 3** El mismo interesado llega con nuevos requerimientos. Además de la tabla ya existente, se requiere una nueva tabla completamente separada que recopile la misma información, pero registre la temperatura en Fahrenheit (en lugar de Grados Celsius). Además, la nueva tabla contendrá los registros de temperatura distribuidos por día, en lugar de por hora. La nueva tabla debe contener todos los datos ya recopilados de la tabla definida en la Parte 1.

**Parte 4** Se recibe un nuevo requerimiento por parte del interesado. Ambas tablas definidas en la Parte 1 y la Parte 3 deben ahora capturar la diferencia de temperatura (delta) entre un registro y el anterior. En el caso horario, la nueva métrica contendrá la diferencia entre el momento actual y una hora antes. Para el caso diario, la nueva métrica contendrá la diferencia entre el momento actual y el día anterior. La nueva columna debe ser completada retroactivamente para todas las temperaturas ya existentes.

## Solución

Para la realización de este punto se utilizó SQL Server como dialecto y desplegamos la base de datos mediante el uso de docker desktop con una imagen de SQL Server 2022. Se realizó una conexión de la base de datos con el programa Azure Data Studio y se creó la base de datos "WEATHER".

### Parte 1. Creación de la tabla CLIMA y carga

Para la creación de la tabla **CLIMA** se decidió utilizar un **ID** como **PRIMARY KEY** que es un entero que se inserta automáticamente asignando un valor que incrementa en 1 con cada **INSERT** que se realiza gracias a **IDENTITY(1,1)**, esto permite identificar de manera única cada dato.

Para los datos relacionados con nombres de ubicaciones, y características climáticas (**LOCALIDAD**, **PAÍS** y **COVERTURA**) se utilizó el tipo de dato **VARCHAR** ya que ninguno tiene una longitud definida de caracteres y se ha definido el máximo de caracteres basado en qué tan largo podrían ser los nombres a insertar.

Para las mediciones meteorológicas (**TEMP\_CELCIUS**, **COVERTURA**, **INDICE\_UV**, **PRESIÓN\_ATM** y **VEL\_VIENTO\_NUDOS**) se establecieron como **DECIMAL** con precisiones y escalas adecuadas para los tipos de mediciones. Por ejemplo, el **INDICE\_UV** mayor a 11 es muy alto y en ocasiones puede ser registrado como un número decimal, en este caso podría ser almacenado con una precisión de dos décimas.

Para el dato **FECHA\_HORA** se estableció el tipo de dato **DATETIME2** que puede almacenar la fecha en formato **YYYY-MM-DD hh:mm:ss[.fracción]** que puede almacenar perfectamente datos como **2023-02-27 13:45:20** sin necesidad de especificar la fracción.

Para que cada valor tenga significado se realizó una restricción en la cual la pareja **LOCALIDAD** y **FECHA\_HORA** son únicas ya que no puede haber dos climas y condiciones meteorológicas iguales al mismo momento en el mismo lugar. Ninguno de los valores a ingresar podrá ser nulo.

```
CREATE TABLE CLIMA (
    ID INT IDENTITY(1,1) PRIMARY KEY,
    LOCALIDAD VARCHAR(150) NOT NULL,
    PAIS VARCHAR(150) NOT NULL,
    TEMP_CELCIUS DECIMAL(5,2) NOT NULL,
    FECHA_HORA DATETIME2 NOT NULL,
    COVERTURA VARCHAR(50) NOT NULL,
    INDICE_UV DECIMAL(4,2) NOT NULL,
    PRESION_ATM DECIMAL(6,2) NOT NULL,
    VEL_VIENTO_NUDOS DECIMAL(6,2) NOT NULL,
    CONSTRAINT LOCALIDAD_FECHA UNIQUE (LOCALIDAD, FECHA_HORA)
);
```

Debido al corto tiempo y para poder tener valores en la base de datos se pidió a una chat de texto predictivo generar un código en python para poder generar valores aleatorios automáticamente. Se realizó una conexión con la base de datos y se insertaron utilizando la librería `sqlalchemy` de python. Por favor revisar `generate_data.py`.

## Parte 2. Tres mejoras para mejorar la lectura con tablas grandes

### Implementación de Particionamiento

Las tablas grandes pueden ser particionadas para mejorar la consulta, en este caso, la tabla podría particionarse por un rango de fechas. Si la tabla `CLIMA` se partitiona cada mes o año, por ejemplo, se generaría bloques de datos mucho más pequeños. De esta manera las consultas que incluyen `WHERE FECHA_HORA BETWEEN` se convierten en consultas más eficientes ya que no tienen que leer todos los millones de datos y se hace un uso de los índices más eficiente.

### Implementación de índices más elaborados

La creación de indices adecuados para la consulta puede mejorar considerablemente la velocidad de la misma, de igual manera, los indices mal diseñados pueden no apoyar la consulta de la información o incluso empeorar el proceso al demandar más almacenamiento y procesamiento. Un índice que se podría implementar es el de la columna `FECHA_HORA`, inclusive, si las consultas son recurrentes para un lugar y tiempo específico se podría crear un índice compuesto (`LOCALIDAD, FECHA_HORA`). La creación de los índices necesarios para mejorar la consulta dependerá del propósito de las consultas y las consultas que se realicen de manera frecuente en la base de datos.

### Normalización de los datos en un modelado estrella

Existen muchos datos que pueden llegar a ser redundantes cuando la información se almacena a gran escala. Por ejemplo, si la información recopilada del país siempre es la misma o varía muy poco al igual que las localidades puede crearse una tabla de dimensiones y de hechos para mejorar el almacenamiento y consulta. Por ejemplo, se podrían crear tablas de dimensión de localidad, tiempo y clima(refiriéndonos a la variable de covertura) en torno a una tabla de hechos que contiene las métricas del clima.

## Parte 3. Creación de la tabla `CLIMA_DIA` y cargue de los datos

Para la creación de la tabla `CLIMA_DIA` se utilizó un formato de datos similar al utilizado para la tabla `CLIMA` para las métricas ya existentes se crearon las columnas con el prefijo `AVG average` en inglés para la palabra promedio, ya que los datos insertados a esta tabla tendrán el promedio de los valores registrados durante el día en la fecha especificada. Ya que el formato de la columna `FECHA` solo necesita almacenar `YYYY-MM-DD` podemos cambiar el tipo de dato a almacenar a `DATE`.

La columna `AVG_TEMP_FAHRENHEIT` almacenará el promedio de las temperaturas registradas en el día en escala Fahrenheit. Parar la columna `SET_COVERTURA` se almacenará una cadena de texto con los climas registrados durante el día, debido a que no se conoce cuantos caracteres contendrá se estableció como tipo de dato `VARCHAR(MAX)`. El `PRIMARY KEY` se estableció de la misma manera en como se hizo para la tabla `CLIMA` y nuevamente se creó un `CONSTRAINT` para que el par `LOCALIDAD, FECHA` fueran únicos, nuevamente ninguno de los valores a ingresar podrá ser nulo.

```
CREATE TABLE CLIMA_DIA (
    ID INT IDENTITY(1,1) PRIMARY KEY,
    LOCALIDAD VARCHAR(150) NOT NULL,
    PAIS VARCHAR(150) NOT NULL,
    AVG_TEMP_FAHRENHEIT DECIMAL(5,2) NOT NULL,
    FECHA DATE NOT NULL,
    SET_COVERTURA VARCHAR(MAX) NOT NULL,
    AVG_INDICE_UV DECIMAL(4,2) NOT NULL,
    AVG_PRESION_ATM DECIMAL(6,2) NOT NULL,
    AVG_VEL_VIENTO_NUDOS DECIMAL(6,2) NOT NULL,
    CONSTRAINT LOCALIDAD_FECHA_DIA UNIQUE (LOCALIDAD, FECHA)
);
```

Para la inserción de los datos se utilizó el siguiente código podemos explicar el código por secciones:

### **CTE para consultar los valores agregados**

En esta sección de código se utiliza la consulta temporal `CTE1`, las funciones de agregación y agrupación para obtener los datos solicitados que serán insertados en la tabla `CLIMA_DIA`.

La función de agregación por excelencia en esta consulta es `AVG` que se encarga de calcular el promedio de los valores seleccionados por el GROUP BY. Un aspecto relevante en la función de agregación usada en la métrica de temperacuta es que se utiliza la función sobre `((a.TEMP_CELCIUS * 9.0/5.0) + 32)` ya que la temperatura de la tabla `CLIMA` está expresada en grados celcius.

En el caso de la `FECHA` se utiliza la función `CAST`, que convierte el tipo de dato que se tenía en la tabla `CLIMA` (`DATETIME2`) a `DATE` para que únicamente seleccione la fecha con el formato establecido en la tabla `CLIMA_DIA`. Para la columna `SET_COVERTURA` se utilizó la función `STRING_AGG` que se encarga de agrupar todos los valores seleccionados por el `GROUP BY` y separarlos por `' , '`. Adicionalmente, a cada columna se le asigna el alias correspondiente al nombre de la columna de la tabla `CLIMA DÍA`.

Por último, se utiliza el `GROUP BY` por `LOCALIDAD, PAIS` y `CAST(a.FECHA_HORA AS DATE)`, de esta manera las agregaciones serán aplicadas diariamente para cada localidad independientemente.

## Inserción a la tabla CLIMA\_DIA

Luego se realiza la inserción de los datos resultantes de la CTE1, insertando así cada fila como un registro diario y utilizando la estructura de las columnas de la tabla CLIMA\_DIA.

```

WITH CTE1 AS (
    SELECT
        a.LOCALIDAD,
        a.PAIS,
        CAST(a.FECHA_HORA AS DATE),
        AVG((a.TEMP_CELCIUS * 9.0/5.0) + 32),
        STRING_AGG(a.COVERTURA, ', '),
        AVG(a.INDICE_UV),
        AVG(a.PRESION_ATM),
        AVG(a.VEL_VIENTO_NUDOS)
    FROM WEATHER.dbo.CLIMA AS a
    GROUP BY
        a.LOCALIDAD,
        a.PAIS,
        CAST(a.FECHA_HORA AS DATE)
)

INSERT INTO WEATHER.dbo.CLIMA_DIA (
    LOCALIDAD,
    PAIS,
    FECHA,
    AVG_TEMP_FAHRENHEIT,
    SET_COVERTURA,
    AVG_INDICE_UV,
    AVG_PRESION_ATM,
    AVG_VEL_VIENTO_NUDOS
)
SELECT
    LOCALIDAD,
    PAIS,
    FECHA,
    AVG_TEMP_FAHRENHEIT,
    SET_COVERTURA,
    AVG_INDICE_UV,
    AVG_PRESION_ATM,
    AVG_VEL_VIENTO_NUDOS
FROM CTE1;

```

Como adición a esta parte del punto en el código se añadió el código referente a un procedimiento almacenado, que permitiría realizar la carga de datos a la tabla CLIMA\_DIA estableciendo la variable @FECHA para un día específico deseado. Por favor revisar el código.

## Parte 4. Uso de funciones de ventana para calcular las diferencias de temperatura

Como primer paso para resolver este punto se añadió una nueva columna a la tabla CLIMA llamada DELTA\_TEMP\_C. Esta columna tiene el mismo tipo de dato que TEMP\_CELCUIUS y no podrá ser nulo.

Paso seguido se utilizó una consulta temporal **CTE** para obtener los datos con los que posteriormente se calculará la diferencia de temperatura. En esta consulta utilizamos la función **LAG(TEMP\_CELCIUS, 1)** que devuelve el valor previo a **TEMP\_CELCIUS** en la fila inmediatamente anterior sobre la partición seleccionada. En la partición se seleccionan bloques por **LOCALIDAD** y **PAIS** y se ordenan por **FECHA\_HORA** de manera ascendente. De esta manera la partición será para la misma **LOCALIDAD**, perteneciente al mismo **PAIS** ordenada por **FECHA\_HORA** en la cual los datos están ordenados en orden de recolección, así la función **LAG** devolverá el dato de la hora inmediatamente anterior a la fila en la que se encuentra, cabe aclarar que para el primer valor sobre la partición el resultado será **NULL** (ya que no existe un valor anterior a este). A esta función **LAG** se le asigna el nombre de **TEMP\_PREVIA**.

Por último, la sección de **UPDATE** se encarga de realizar un **INNER JOIN** por **ID** de la **CTE** con la tabla **CLIMA** para posteriormente calcular la diferencia de temperaturas y hacer el **UPDATE** de la columna **DELTA\_TEMP\_C** con el cálculo.

```
ALTER TABLE WEATHER.dbo.CLIMA ADD
    DELTA_TEMP_C DECIMAL(5,2) NULL;

WITH CTE AS (
    SELECT
        ID,
        TEMP_CELCIUS,
        LAG(TEMP_CELCIUS, 1) OVER (
            PARTITION BY LOCALIDAD, PAIS
            ORDER BY FECHA_HORA
        ) AS TEMP_PREVIA
    FROM WEATHER.dbo.CLIMA
)
UPDATE C
SET C.DELTA_TEMP_C = C.TEMP_CELCIUS - T.TEMP_PREVIA
FROM WEATHER.dbo.CLIMA AS C
INNER JOIN CTE AS T
ON C.ID = T.ID;
```

Para la tabla **CLIMA\_DIA** se realiza el mismo procedimiento que con la anterior, la columna añadida se llamó **DELTA\_TEMP\_F**.

```
ALTER TABLE WEATHER.dbo.CLIMA_DIA ADD
    DELTA_TEMP_F DECIMAL(5,2) NULL;

WITH CTE AS (
    SELECT
        ID,
        AVG_TEMP_FAHRENHEIT,
        LAG(AVG_TEMP_FAHRENHEIT, 1) OVER (
            PARTITION BY LOCALIDAD, PAIS
            ORDER BY FECHA
        ) AS TEMP_PREVIA
    FROM WEATHER.dbo.CLIMA_DIA
```

```
)
UPDATE C
SET C.DELTA_TEMP_F = C.AVG_TEMP_FAHRENHEIT - T.TEMP_PREVIA
FROM WEATHER.dbo.CLIMA_DIA AS C
INNER JOIN CTE AS T
ON C.ID = T.ID;
```

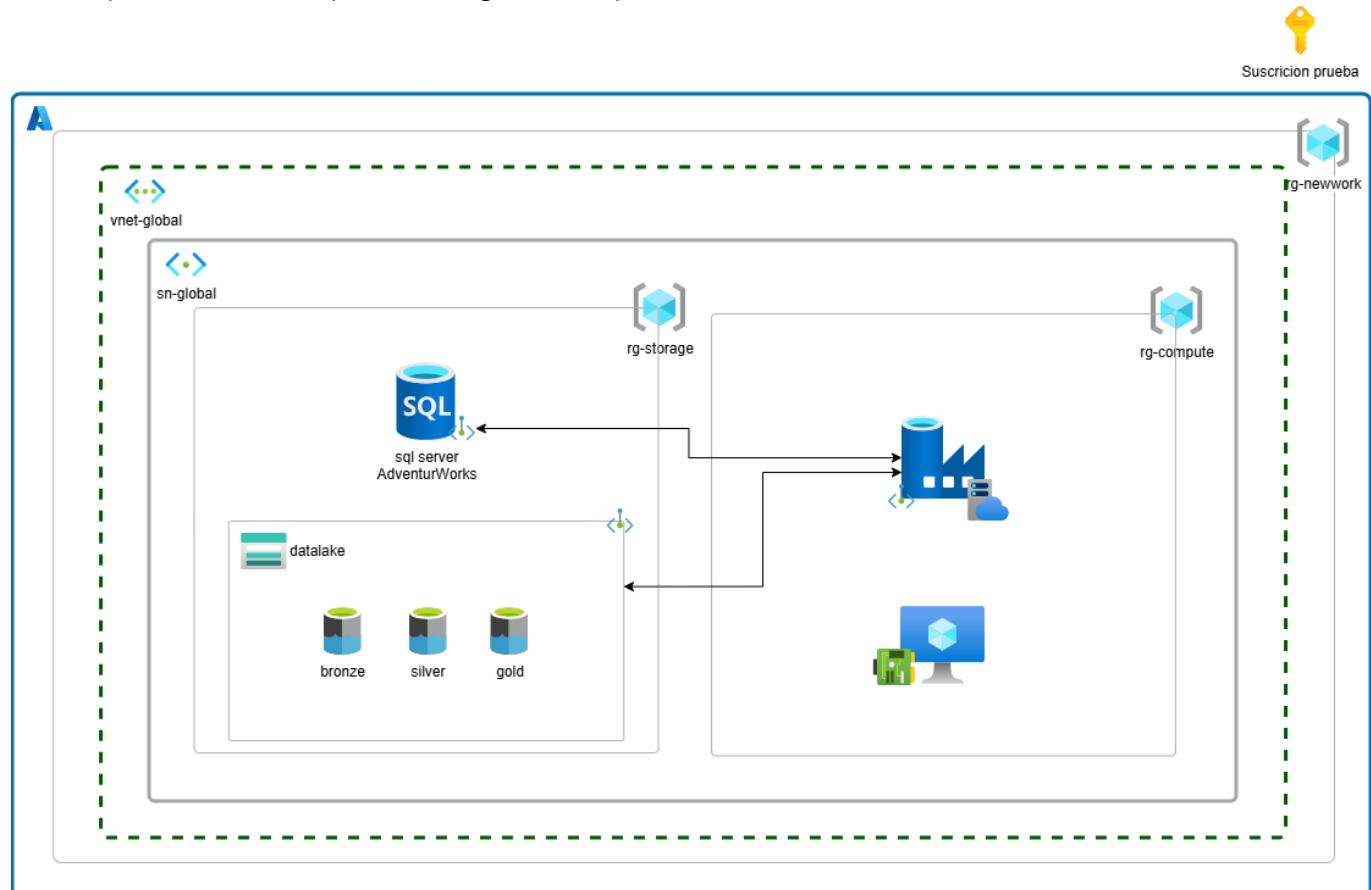
## Quest 5: Prueba Azure

Construya una solución completa en la nube de Azure que usando todas la base, de pruebas adventure Works, permita crear una etl, para la realización de un trabajo de reporteria dentro de la organización.

- desplegar base de datos en sql, con la base de pruebas adventure works
- realizar un pipeline con Azure Datafactory, utilizando data flow, para realizar la carga de una base de datos, crear 5 indicadores.
- realizar una etl, que poble un datalake.

### Solución

En esta prueba técnica se planteó la siguiente arquitectura:



El propósito de este esquema es plantear una arquitectura sencilla para desplegar los servicios básicos, se omitieron prácticas detalladas para la asignación de nombres y recursos de red. Pero se hizo el ejercicio de repartir grupos de recurso por contexto. En este caso, cómputo, almacenamiento y network.

## Data Factory

Se desplegó el recurso de Data Factory, se hicieron las correspondientes conexiones entre los servicios vinculados

![Data Factory](quest5\img)

## KPIs para la base de datos de Adventure Works

A continuación, se presentan las siguientes consultas que representan la lógica de los KPIs, estos KPIs fueron extraídos desde Github:

### 1. KPI Ventas Totales

Este KPI representa la suma del importe facturado por las líneas de pedido o *line totals* por periodo.

```
SELECT
    DATEPART(YEAR, h.OrderDate) AS Year,
    DATEPART(MONTH, h.OrderDate) AS Month,
    SUM(d.LineTotal) AS TotalSales
FROM SalesLT.SalesOrderHeader h
JOIN SalesLT.SalesOrderDetail d ON h.SalesOrderID = d.SalesOrderID
WHERE h.OrderDate BETWEEN '2000-01-01' AND '2025-12-31'
GROUP BY DATEPART(YEAR,h.OrderDate), DATEPART(MONTH,h.OrderDate);
```

### 2. KPI Margen Bruto Porcentaje

Representa la rentabilidad de los gastos operativos por periodo.

```
SELECT
    YEAR(h.OrderDate) AS Year,
    MONTH(h.OrderDate) AS Month,
    SUM(d.LineTotal) AS Revenue,
    SUM(p.StandardCost * d.OrderQty) AS COGS,
    CASE WHEN SUM(d.LineTotal) = 0 THEN NULL
        ELSE (SUM(d.LineTotal) - SUM(p.StandardCost * d.OrderQty)) * 100.0 /
    SUM(d.LineTotal)
    END AS GrossMarginPct
FROM SalesLT.SalesOrderHeader h
JOIN SalesLT.SalesOrderDetail d ON h.SalesOrderID = d.SalesOrderID
JOIN SalesLT.Product p ON d.ProductID = p.ProductID
WHERE h.OrderDate BETWEEN '2000-01-01' AND '2025-12-31'
GROUP BY YEAR(h.OrderDate), MONTH(h.OrderDate);
```

### KPI 3. Valor Promedio Por Pedido

Promedio del importe por órdenes de un periodo.

```

WITH OrderTotals AS (
    SELECT d.SalesOrderID, SUM(d.LineTotal) AS OrderTotal, MIN(h.OrderDate) AS OrderDate
    FROM SalesLT.SalesOrderHeader h
    JOIN SalesLT.SalesOrderDetail d ON h.SalesOrderID = d.SalesOrderID
    WHERE h.OrderDate BETWEEN '2000-01-01' AND '2025-12-31'
    GROUP BY d.SalesOrderID
)
SELECT YEAR(OrderDate) AS Year, MONTH(OrderDate) AS Month,
    AVG(OrderTotal) AS AOV, COUNT(*) AS OrdersCount
FROM OrderTotals
GROUP BY YEAR(OrderDate), MONTH(OrderDate);

```

#### KPI 4. Tasa entregas a tiempo

Porcentaje de órdenes por linea enviadas en la fecha o antes de la fecha estipulada.

```

SELECT YEAR(h.ShipDate) AS Year, MONTH(h.ShipDate) AS Month,
    SUM(CASE WHEN h.ShipDate <= h.DueDate THEN 1 ELSE 0 END) * 100.0 / COUNT(*)
AS OnTimePct,
    COUNT(*) AS TotalShipments
FROM SalesLT.SalesOrderHeader h
WHERE h.ShipDate IS NOT NULL AND h.ShipDate BETWEEN '2000-01-01' AND '2025-12-31'
GROUP BY YEAR(h.ShipDate), MONTH(h.ShipDate);

```

#### KPI 5. Top 5 Productos Mes

. Representa el top 5 de los productos más vendidos por mes

```

WITH MonthlyProductSales AS (
    SELECT
        DATEPART(YEAR, h.OrderDate) AS Año,
        DATEPART(MONTH, h.OrderDate) AS Mes,
        p.Name AS Producto,
        SUM(d.LineTotal) AS VentasProducto,
        RANK() OVER (
            PARTITION BY DATEPART(YEAR, h.OrderDate),
            DATEPART(MONTH, h.OrderDate)
            ORDER BY SUM(d.LineTotal) DESC
        ) AS RankingMensual
    FROM SalesLT.SalesOrderHeader h
    JOIN SalesLT.SalesOrderDetail d
        ON h.SalesOrderID = d.SalesOrderID
    JOIN SalesLT.Product p
        ON d.ProductID = p.ProductID
    GROUP BY
        DATEPART(YEAR, h.OrderDate),
        DATEPART(MONTH, h.OrderDate),

```

```

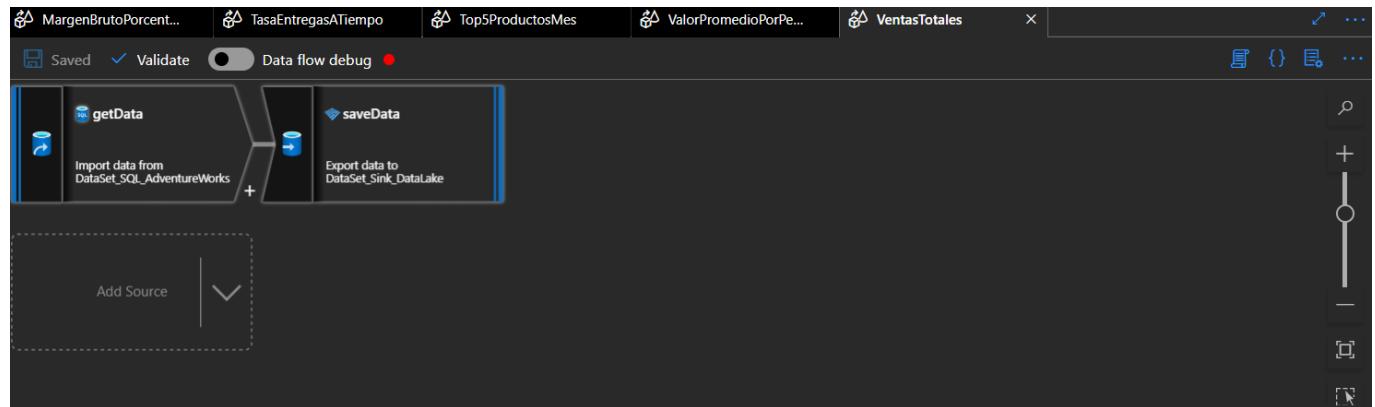
    p.Name
)
SELECT *
FROM MonthlyProductSales
WHERE RankingMensual <= 5
ORDER BY Año, Mes, RankingMensual;

```

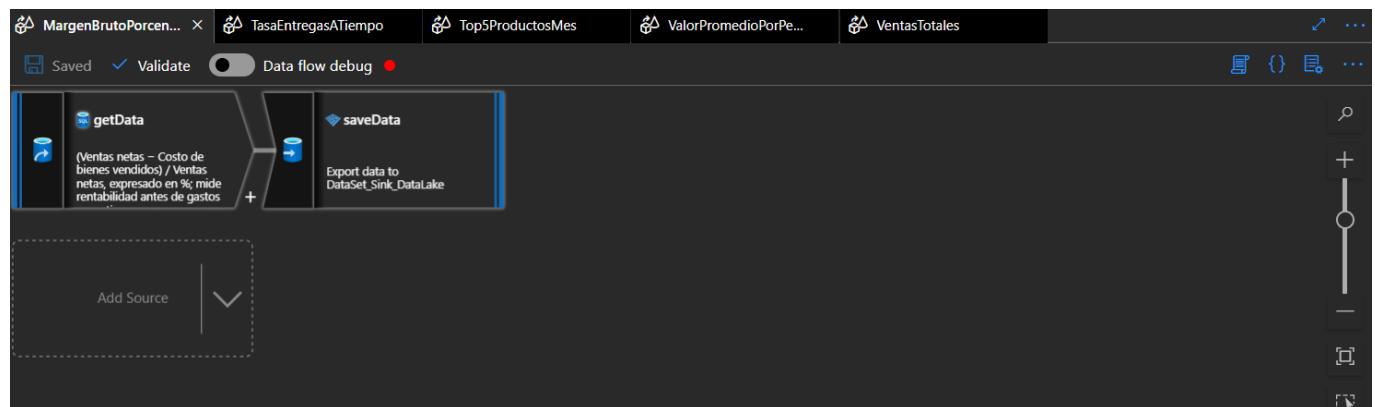
## DataFlows

A continuación se presentan los DataFlows creados para cada KPI:

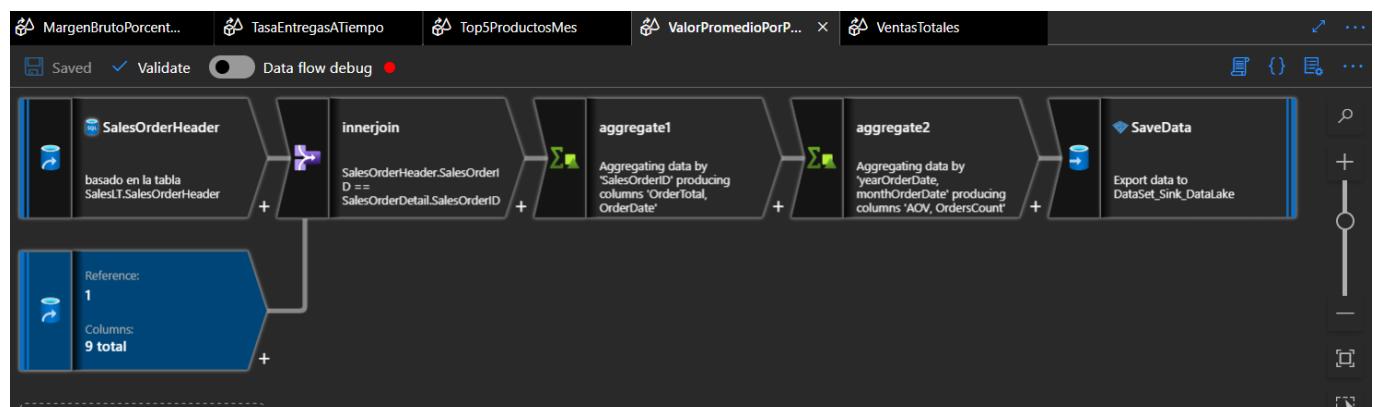
### 1. KPI Ventas Totales



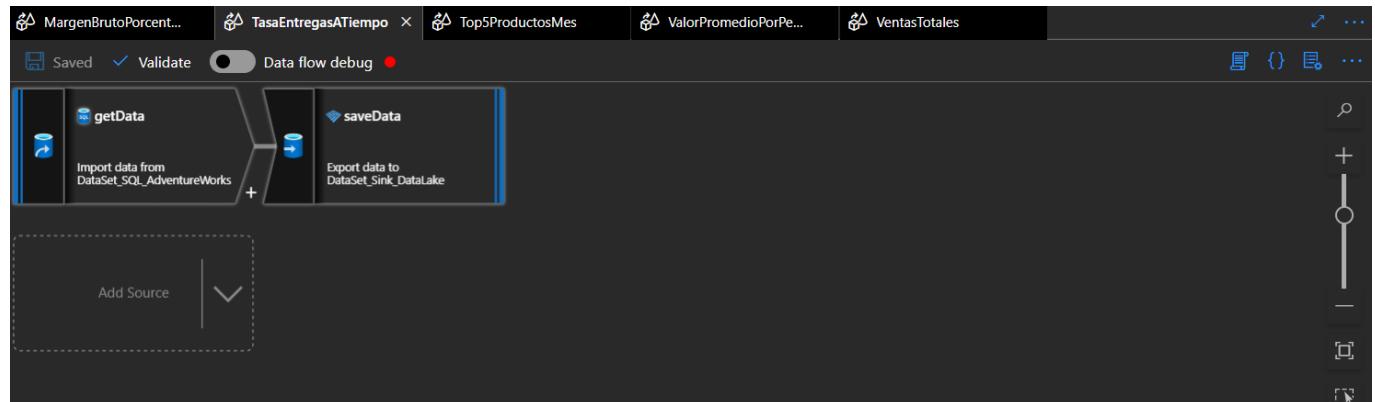
### 2. KPI Margen Bruto Porcentaje



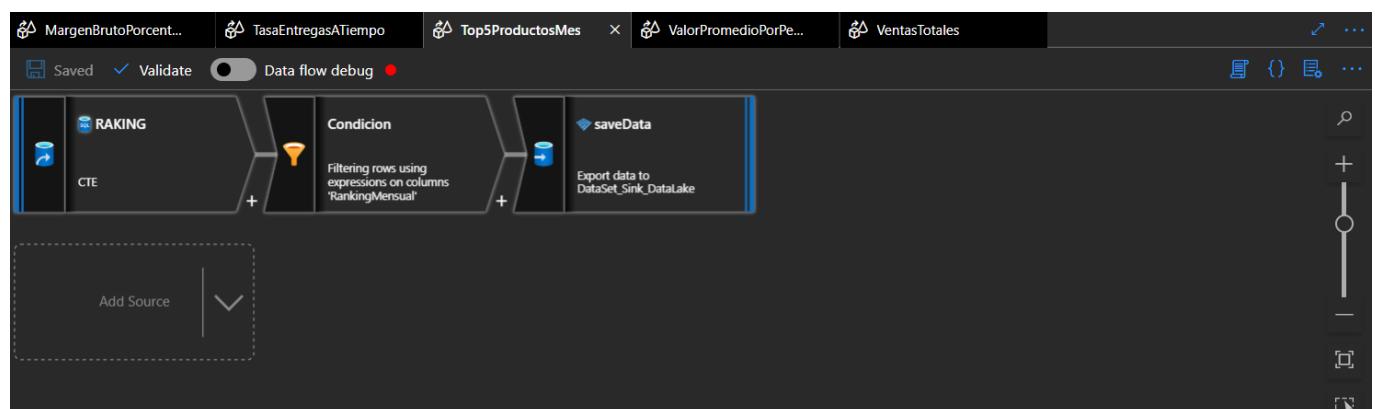
### KPI 3. Valor Promedio Por Pedido



## KPI 4. Tasa entregas a tiempo

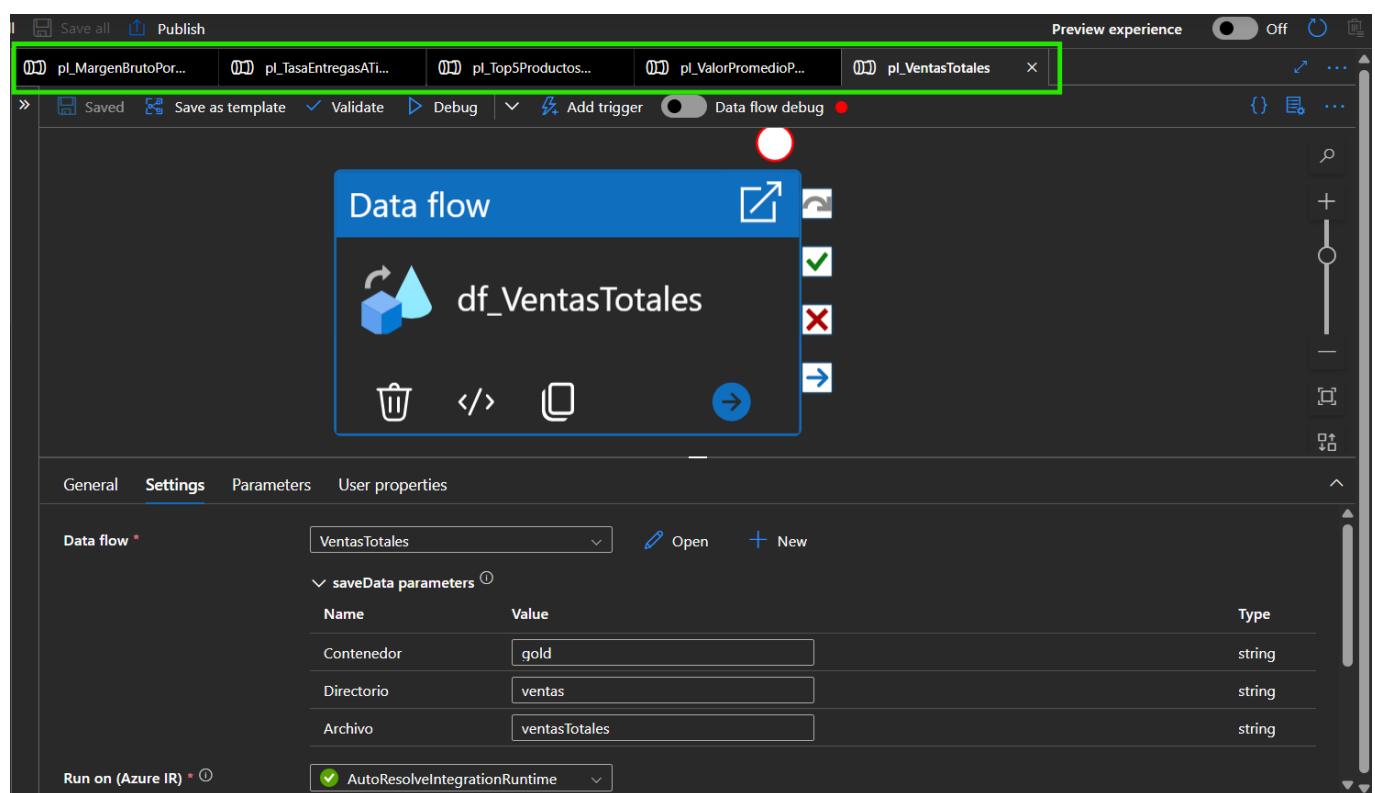


## KPI 5. Top 5 Productos Mes



## Resultados

Con base a los DataFlows desarrollados se construyeron los pipelines respectivos que ven en la siguiente imagen



A continuación, se muestra la ejecución de los pipelines

Pipeline runs																	
Triggered		Debug	Run	Cancel options	Refresh	Edit columns	List	Gantt									
<input type="button" value="Filter by run ID or name"/>		Bogota, Lima, Quito ... : Last 24 hours		Pipeline name : All		Status : All		Runs : Latest runs									
Triggered by : All		<input type="button" value="Add filter"/>		<input type="button" value="X"/>													
Showing 1 - 11 items																	
Last refreshed 0 minutes ago																	
<input type="checkbox"/>	Pipeline name	Run start ↑	Run end ↓	Duration	Triggered by	Status	Run	Parameters									
<input type="checkbox"/>	pl_VentasTotales	11/30/2025, 5:21:51 PM	11/30/2025, 5:25:02 PM	3m 11s	Manual trigger	<span style="color: green;">✓ Succeeded</span>	Original										
<input type="checkbox"/>	pl_ValorPromedioPorPedido	11/30/2025, 5:21:48 PM	11/30/2025, 5:25:13 PM	3m 26s	Manual trigger	<span style="color: green;">✓ Succeeded</span>	Original										
<input type="checkbox"/>	pl_Top5ProductosMes	11/30/2025, 5:21:44 PM	11/30/2025, 5:27:04 PM	5m 21s	Manual trigger	<span style="color: green;">✓ Succeeded</span>	Original										
<input type="checkbox"/>	pl_TasaEntregasATiempo	11/30/2025, 5:21:40 PM	11/30/2025, 5:24:57 PM	3m 18s	Manual trigger	<span style="color: green;">✓ Succeeded</span>	Original										
<input type="checkbox"/>	pl_MargenBrutoPorcentaje	11/30/2025, 5:21:36 PM	11/30/2025, 5:24:56 PM	3m 21s	Manual trigger	<span style="color: green;">✓ Succeeded</span>	Original										
<input type="checkbox"/>	pl_ValorPromedioPorPedido	11/30/2025, 5:06:56 PM	11/30/2025, 5:10:36 PM	3m 40s	Manual trigger	<span style="color: green;">✓ Succeeded</span>	Original										
<input type="checkbox"/>	pipeline1	11/30/2025, 4:59:32 PM	11/30/2025, 5:05:07 PM	5m 36s	Manual trigger	<span style="color: green;">✓ Succeeded</span>	Original										
<input type="checkbox"/>	pl_ValorPromedioPorPedido	11/30/2025, 4:40:14 PM	11/30/2025, 4:45:02 PM	4m 48s	Manual trigger	<span style="color: red;">✗ Failed</span> <span style="color: blue;">⌚</span>	Original										
<input type="checkbox"/>	pl_ValorPromedioPorPedido	11/30/2025, 4:30:59 PM	11/30/2025, 4:33:51 PM	2m 52s	Manual trigger	<span style="color: red;">✗ Failed</span> <span style="color: blue;">⌚</span>	Original										
<input type="checkbox"/>	pl_ValorPromedioPorPedido	11/30/2025, 4:23:23 PM	11/30/2025, 4:26:34 PM	3m 11s	Manual trigger	<span style="color: red;">✗ Failed</span> <span style="color: blue;">⌚</span>	Original										
<input type="checkbox"/>	pl_ValorPromedioPorPedido	11/30/2025, 3:57:34 PM	11/30/2025, 4:01:08 PM	3m 34s	Manual trigger	<span style="color: red;">✗ Failed</span> <span style="color: blue;">⌚</span>	Original										

Y por último se puede ver los datos almacenados en el DataLake

Add Directory							Upload	Refresh	Delete	Copy	Paste	Rename	Acquire lease	Break lease	Edit columns
gold >  ventas															
Authentication method: Access key <a href="#">Switch to Microsoft Entra user account</a>															
<input type="text"/> Search blobs by prefix (case-sensitive)															
Showing all 5 items															
<input type="checkbox"/>	Name	Last modified	Access tier	Blob type	Size	Lease state									
<input type="checkbox"/>	[..]						...								
<input type="checkbox"/>	ValorPromedioPorPedido	30/11/2025, 17:25:03					...								
<input type="checkbox"/>	margenBrutoPorcentaje	30/11/2025, 17:24:31					...								
<input type="checkbox"/>	tasaEntregasATiempo	30/11/2025, 17:24:35					...								
<input type="checkbox"/>	top5ProductosMes	30/11/2025, 17:26:36					...								
<input type="checkbox"/>	ventasTotales	30/11/2025, 17:24:46					...								

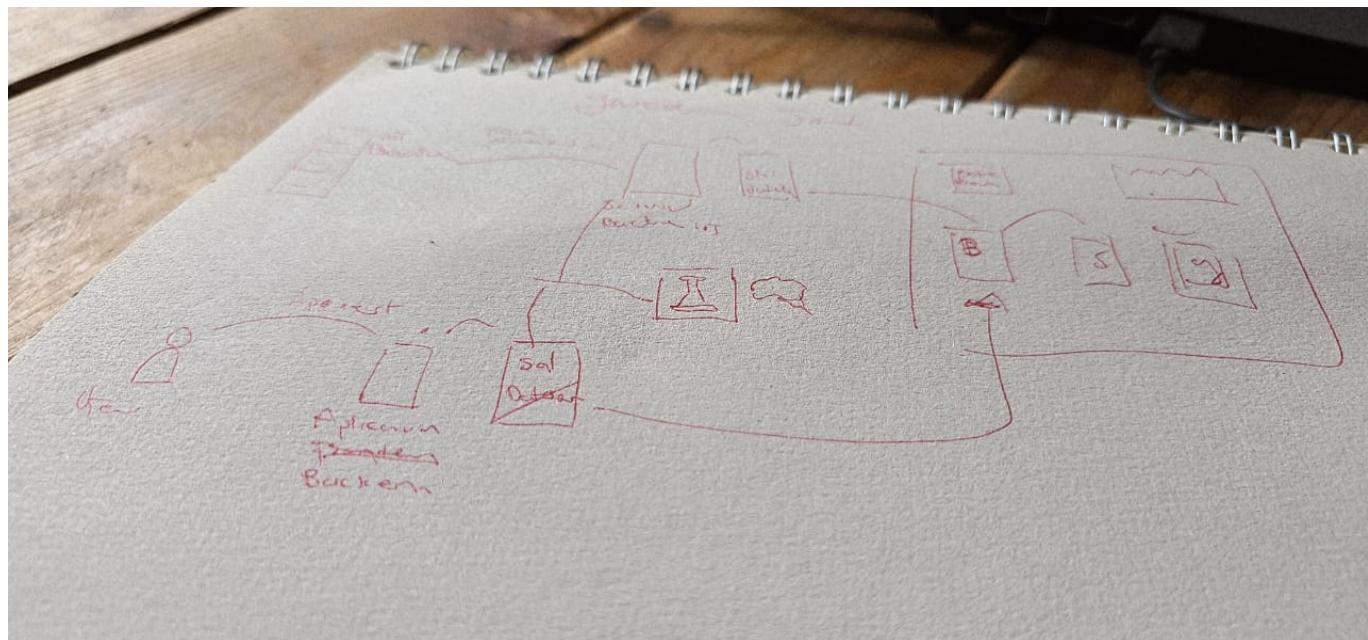
## Quest 7: Arquitectura

En la empresa gaseosas SA están trabajando en una solución analítica que sea capaz de procesar miles de datos de las ventas donde se describen comportamiento de compra y análisis previos hechos por vendedores a clientes con gran volumen de compra, de forma rápida y confiable mediante el uso de tecnologías Big Data de analítica, para entrenar un modelo que sea capaz de identificar los patrones de estas ventas y compararlos en tiempo real con los patrones de datos capturados de manera streaming por dispositivos implantados puntos de venta, para controlar tempranamente y evitar el desabastecimiento.

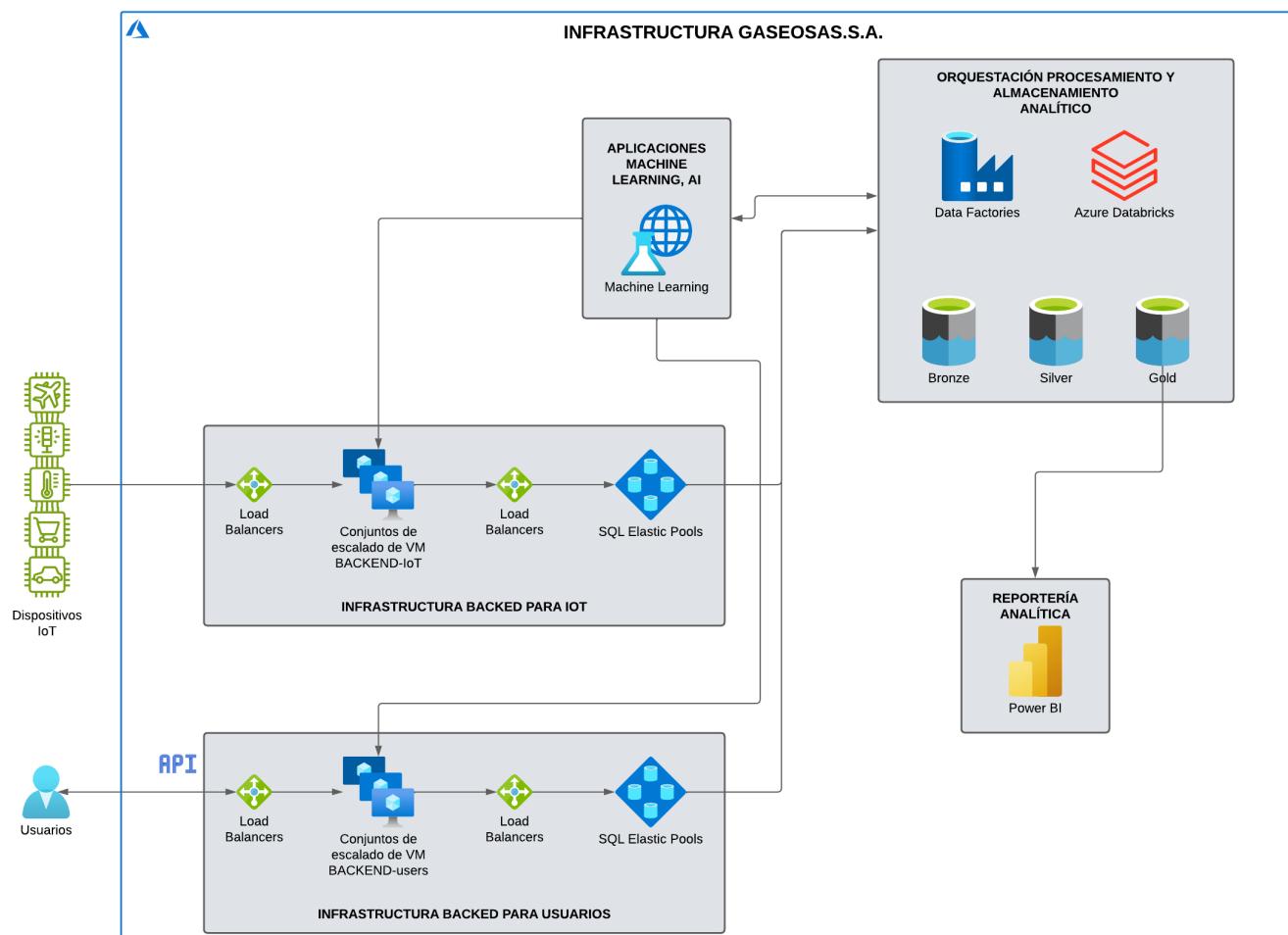
Tu tarea es realizar un correcto diseño de la arquitectura para la solución analítica que podría soportar estos requerimientos. (Ilustra tu diseño y da una breve explicación de su funcionamiento), es importa definir el gobierno de datos y modelos de acuerdo a los perfiles

## Solución

En una etapa inicial de exploración contemplé la utilización de productos SaaS y PaaS tanto de AWS como Azure, sin embargo, debido a la facilidad de conocimiento de los productos de Azure, finalmente se optó por esta propuesta.



Finalmente se decidió presentar la siguiente arquitectura final a muy alto nivel (poco detalle), en esta arquitectura no se contemplan arquitecturas de red, administración de recursos, monitoreo, etc.



## Explicación resumida de la arquitectura

La arquitectura propuesta para Gaseosas S.A. permite procesar miles de datos de ventas y señales en tiempo real provenientes de dispositivos instalados en los puntos de venta. El objetivo es anticipar comportamientos de consumo y evitar el desabastecimiento.

Para lograrlo, se separan dos flujos principales:

- **Dispositivos IoT**, que llegan desde dispositivos en tiendas, camiones, fábricas etc.
- **Uuarios**, como ventas históricas y análisis hechos por los vendedores.

Ambos flujos entran a una infraestructura flexible con balanceadores de carga y máquinas que pueden escalar según la demanda, lo que asegura que el sistema se mantenga disponible aun cuando aumenta la cantidad de datos.

Toda la información llega a un almacenamiento central donde luego es procesada mediante herramientas de orquestación y análisis. Aquí se organizan los datos en capas (Bronze, Silver y Gold), lo que permite tenerlos primero en bruto, luego limpios y finalmente listos para análisis y creación de modelos de inteligencia artificial. Sobre esta base se entrena modelo predictivo capaces de identificar patrones de compra y compararlos con los datos que llegan en tiempo real desde los dispositivos.

Finalmente, las capas procesadas alimentan reportes en Power BI, donde los equipos de ventas, logística y gerencia pueden visualizar alertas, tendencias y comportamientos relevantes.

## Gobierno de Datos y Modelos

El gobierno de datos en esta arquitectura básicamente se encarga de que toda la información que entra y sale del sistema esté bien organizada, sea segura y pueda usarse sin problemas por las áreas que la necesitan.

Primero, cuando los datos llegan desde los dispositivos IoT y desde los sistemas de ventas, pasan por diferentes capas (Bronze, Silver y Gold). Esto ayuda a que los datos se limpien, se ordenen y queden en un formato estándar para que los equipos de analítica y machine learning puedan trabajar sin errores.

También se manejan permisos por perfiles. Por ejemplo, los equipos técnicos pueden ver datos más detallados, mientras que quienes solo usan reportes en Power BI acceden solo a información final. Esto evita riesgos y asegura que cada persona vea solo lo que le corresponde.

Además, se registra de dónde viene cada dato y cómo se transforma. Eso es útil para auditorías y para saber qué está pasando si un reporte o un modelo muestra resultados extraños.

Finalmente, los modelos de machine learning también se administran: se guarda qué versión está en uso, con qué datos se entrenó y cuándo debe actualizarse. Así se evita que el modelo se vuelva obsoleto o dé predicciones equivocadas.

## Herramientas Usadas

---

Componente	Descripción
Python	Procesamiento del CSV

Componente	Descripción
Pandas	Transformaciones
VS Code / Jupyter / Azure Data Studio	Entorno de desarrollo
Git	Control de versiones
Azure	Plataforma en la nube
Draw.io	Plataforma sketching
LucidChart	Plataforma sketching

## Bibliografía

- Docker. *Docker Desktop Documentation*. Disponible en: <https://docs.docker.com/desktop/>
- Microsoft Learn. *Azure Data Factory – Control Flow Expression Language Functions*. Disponible en: <https://learn.microsoft.com/en-us/azure/data-factory/control-flow-expression-language-functions>
- Microsoft Learn. *Quickstart: Create a Data Factory*. Disponible en: <https://learn.microsoft.com/es-es/azure/data-factory/quickstart-create-data-factory>