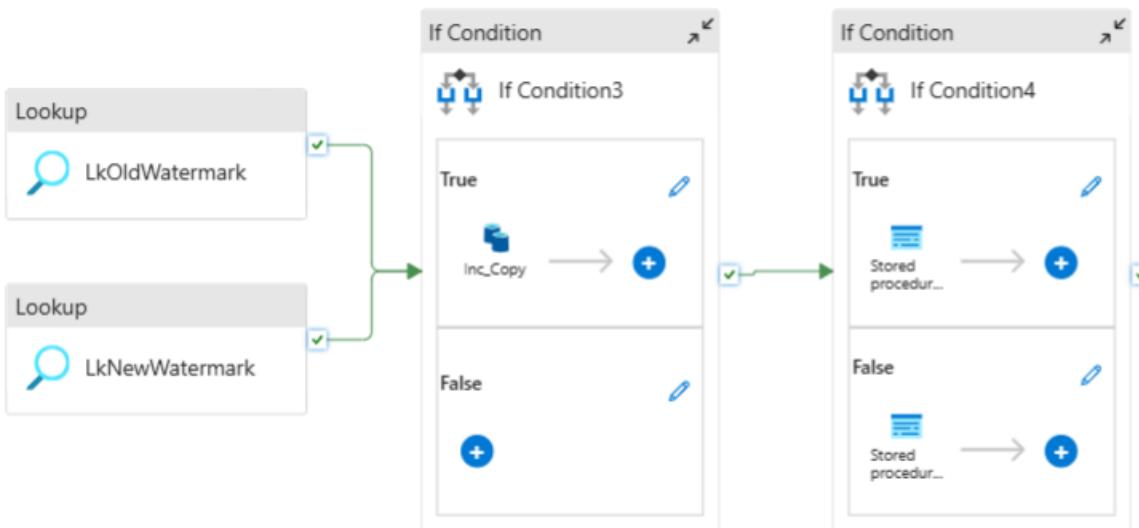
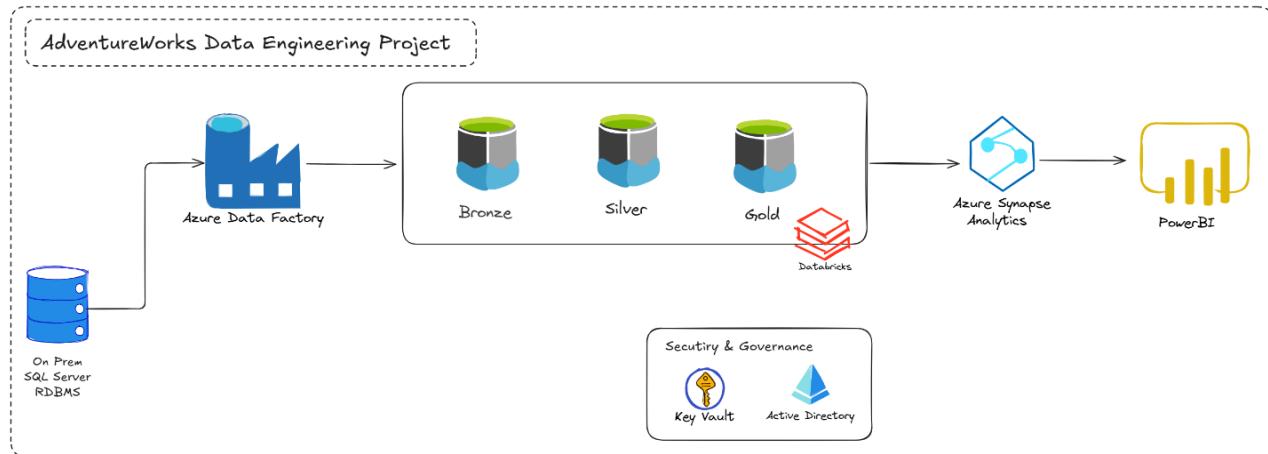


# Proyecto de ingeniería y análisis de datos para el sector e-commerce en el ecosistema de Azure

Ing. José Carlos Reyes



## Introducción

Se presenta el proyecto de Ingeniería y análisis de datos con el dataset de Microsoft AdventureWorks, la versión 2019. El objetivo es construir un pipeline de datos que permita la ingestión y automatice la limpieza y transformación hasta tener los datos listos para almacenarlos en el data warehouse y su posterior visualización en un tablero empleando Power-BI.

El cliente cuenta con información histórica en un período del 2010-2014, por lo que nos ha solicitado llevar a cabo un análisis de las ventas incluyendo métricas como el total de ventas, la inversión o presupuesto gastado, ganancias (ventas – inversión), así como también poder visualizar las ventas por producto, ventas por categoría de producto, ventas por ciudad lo anterior con la posibilidad de filtrar por año. A su vez el cliente desea actualizar o añadir de manera mensual los registros de ventas para contar con la información más reciente disponible, por lo que se propone un procesamiento tipo Batch.

## Arquitectura de datos

En la figura 1 podemos visualizar la arquitectura de datos a utilizar. Partimos de la base de datos almacenada en SQL Server. De la base tomaremos las tablas o información que vamos a requerir para nuestro análisis. Utilizaremos el servicio de Orquestación de datos, Azure Data Factory para realizar la ingestión de datos hacia la capa bronce en el Data Lake. Posteriormente, leemos las tablas de la base en la capa bronce con Azure Databricks, las convertimos a formato delta, realizamos su limpieza y transformación hacia la capa silver, así como también realizamos unos joins para obtener las dimension y fact tables del modelo de datos en la capa gold como archivos delta. Azure Synapse Analytics (Warehouse) puede leer las tablas que están dentro del data lake en la capa gold. Dentro del warehouse se construye un pipeline para crear vistas con cada una de las tablas al aplicar una “Lookup activity”, un “ForEach activity” y una “Store-procedure activity” dentro del foreach. Finalmente, desde Power-BI se cargan las tablas desde Azure Synapse y se construye un tablero con los KPI's y visualizaciones solicitadas. Se simula la carga de nuevos datos dentro de la base original al insertar datos para el mes de Febrero de 2014 y se comprueba al correr el pipeline que se encuentra operando de manera correcta. Se describen en seguida cada una de las etapas para un mayor detalle. Los archivos pueden encontrarse en el siguiente repositorio:

<https://github.com/jcreyesh/ADWorks-Azure-Data-Engineering-Project>

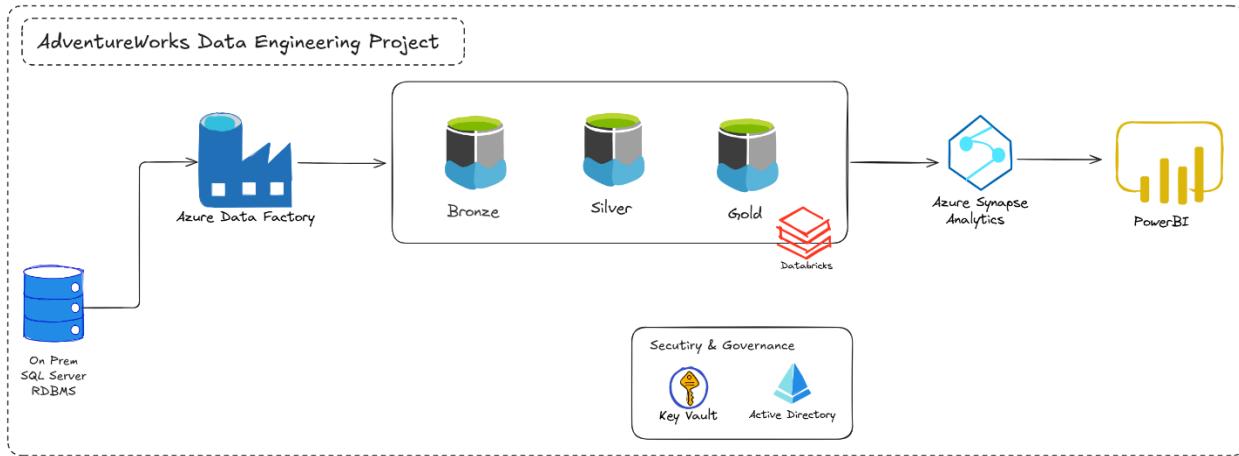


Figura 1.- Diagrama de la arquitectura para el proyecto AdventureWorks utilizando el ecosistema de Azure.

## Base de datos AventureWorks en SQL Server

La base de datos utilizada se restauró con el archivo “AdventureWorks2019.bak” la cual puede descargarse en el siguiente link:

<https://learn.microsoft.com/es-es/sql/samples/adventureworks-install-configure?view=sql-server-ver17&tabs=ssms>

De la base original, vamos a tomar 7 tablas las cuales son de interés para el análisis y creamos una segunda base de datos a la cual llamamos “AdventureWorksDM2019”. Dicha base contiene las tablas: DimCustomer, DimDate, DimGeography, DimProduct, DimProductCategory, DimProductSubcategory y FactInternetSales. En la figura 2 podemos observar las tablas dentro de la base en el IDE de SQL Server, Sql Server Management Studio.

La captura de pantalla muestra la interfaz de Sql Server Management Studio (SSMS) con la siguiente configuración:

- Object Explorer:** Muestra la estructura de la base de datos "AdventureWorksDM2019", incluyendo:
  - Databases: AdventureWorksDM2019
  - Tables: DimCustomer, DimDate, DimGeography, DimProduct, DimProductCategory, DimProductSubcategory, FactInternetSales, FactInternetSalesAdded, sales\_watermarktable
  - Views, External Resources, Synonyms, Programmability, Query Store, Service Broker, Storage, Security
  - AdventureWorksDW2019
- SQL Query Editor:** Muestra la consulta: `SELECT * FROM dbo.FactInternetSales`.
- Results:** Muestra los resultados de la consulta, que incluye 12 filas de datos.

ProductKey	OrderDateKey	DueDateKey	ShipDateKey	CustomerKey	PromotionKey	CurrencyKey	SalesTerritoryKey	SalesOrderNumber
1 222	20131002	20131014	20131009	13406	1	100	8	S0676
2 588	20131002	20131014	20131009	28705	1	100	8	S0676
3 214	20131002	20131014	20131009	28705	1	100	8	S0676
4 561	20131002	20131014	20131009	24700	1	100	7	S0676
5 477	20131002	20131014	20131009	24700	1	100	7	S0676
6 479	20131002	20131014	20131009	24700	1	100	7	S0676
7 490	20131002	20131014	20131009	24700	1	100	7	S0676
8 583	20131002	20131014	20131009	24305	1	6	9	S0676
9 225	20131002	20131014	20131009	24305	1	6	9	S0676
10 480	20131002	20131014	20131009	26022	1	6	9	S0676
11 388	20131002	20131014	20131009	26022	1	6	9	S0676
12 217	20131002	20131014	20131009	26022	1	6	9	S0676

Figura 2.- Base de datos dentro de SSMS para el proyecto de AdventureWorks.

## Creación del Key-vault service

En el azure portal seleccionamos azure key-vaults y creamos un servicio para la gestión de las contraseñas. En la siguiente figura se muestran los detalles para el despliegue del servicio.

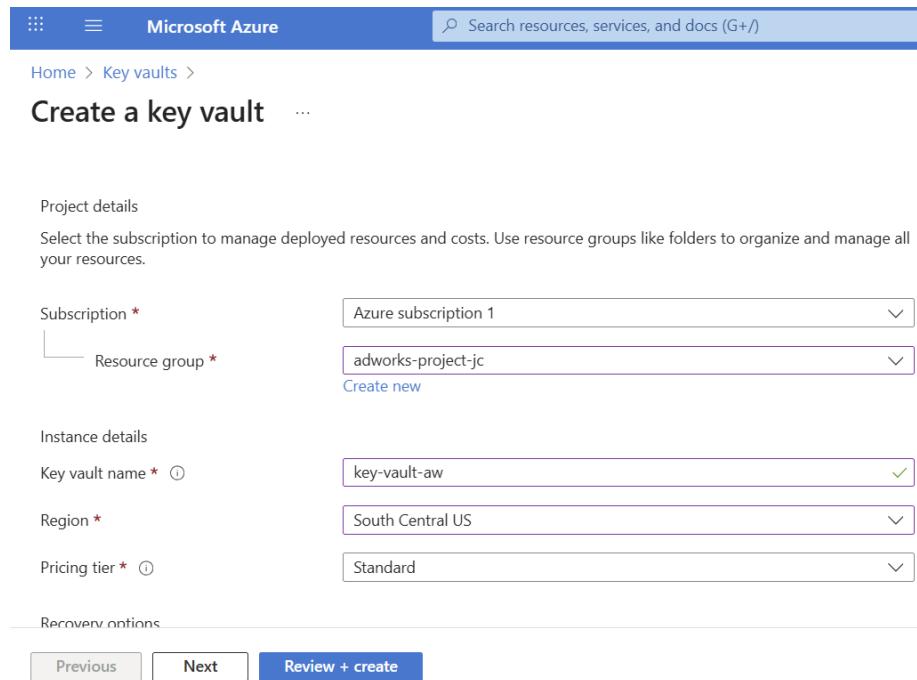


Figura 3.- Creación del servicio key-vault para la gestión de contraseñas en azure.

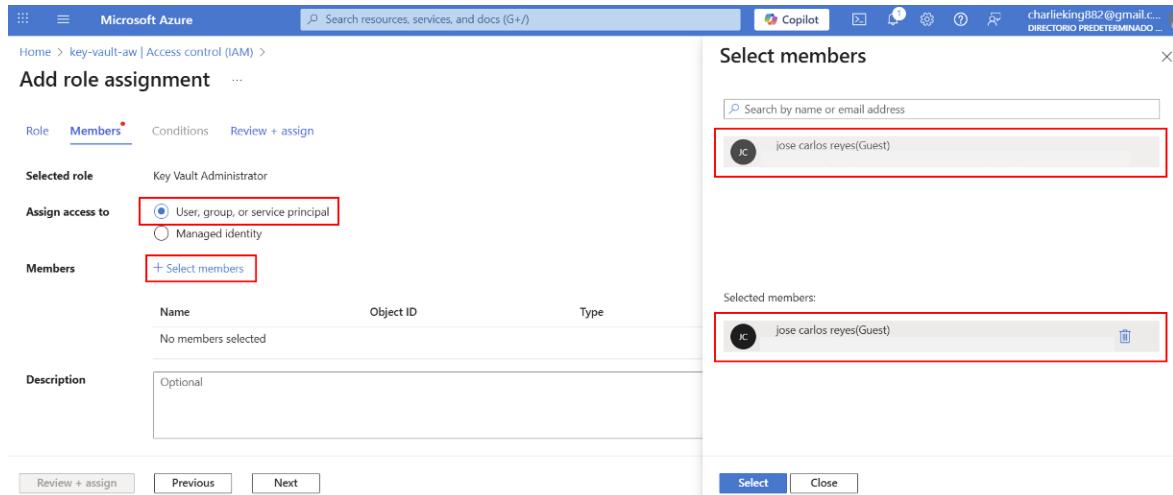
No obstante, se requiere contar con un rol de administrador de Key-vault para poder brindar los permisos necesarios. Para ello, dentro del key-vault, seleccionamos el apartado “Access control (IAM)”, Role assignments, Key vault Administrator, user, group or service principal, select members y damos click en el usuario(figuras 4 y 5). Lo anterior nos permite poder crear objetos dentro del key-vault como los secretos. Nos vamos al key-vault, seleccionamos el apartado de secretos, generate/import y generamos el secreto para el password de sql-server (figura 6).

This screenshot shows the "Access control (IAM)" blade for the "key-vault-aw" resource. The "Role assignments" tab is selected. A red box highlights the "+ Add" button. The table lists two role assignments:

Name	Type	Role	Scope	Condition
jose carlos reyes 97264dde-a4d1-450e-b5...	User	Owner	Subscription (Inherited)	None
jose carlos reyes 97264dde-a4d1-450e-b5...	User	Owner	Subscription (Inherited)	None

Figura 4.- Asignación del rol como administrador dentro del key-vault parte-I.

Ing. José Carlos Reyes



**Figura 5.- Asignación del rol como administrador dentro del key-vault parte-II.**

Upload options	Manual
Name *	password
Secret value *	*****
Content type (optional)	
Set activation date	<input type="checkbox"/>
Set expiration date	<input type="checkbox"/>
Enabled	Yes <input checked="" type="radio"/> No <input type="radio"/>
Tags	0 tags

**Figura 6.- Generación del secreto para el uso de la base de datos On-premise SQL-Server.**

Posteriormente nos vamos al apartado Access control-IAM en el key-vault, role assignments, add, Key vault secrets user, managed identity, select members, suscripción, Data factory, select (figura 7).

The screenshot shows the Microsoft Azure interface for managing access control (IAM). A modal window titled "Select managed identities" is open, prompting the user to choose a subscription and managed identity for the role assignment. The "Subscription" dropdown is set to "Azure subscription 1". The "Managed identity" dropdown is set to "Data factory (V2) (1)". Below these, a search bar allows for further filtering. On the left, the main page shows the "Members" tab selected for the "Key Vault Secrets User" role, with no members assigned yet. A "Description" field is also present. At the bottom right of the modal, a red box highlights the "Select" button.

**Figura 7.-** Asignación de rol como usuario del key-vault a data factory.

## Linked Services

### AzureKeyVault1 linked service

Para la conexión con el key-vault, creamos el linked service, indicamos el nombre, seleccionamos la suscripción, el key-vault creado y verificamos con una prueba de conexión, como se muestra en la figura 8.

The screenshot shows the Microsoft Azure Data Factory interface. On the left, the navigation menu includes "General", "Connections", "Linked services", and "Source control". The "Linked services" option is selected. In the center, a "New linked service" dialog is open, specifically for "Azure Key Vault". The "Azure key vault selection method" section shows "From Azure subscription" selected. The "Azure subscription" dropdown is set to "Azure subscription 1 (1ca37624-0d3a-4f75-954d-c0d3f59f08fb)". The "Azure key vault name" dropdown is set to "key-vault-aw". The "Authentication method" dropdown is set to "System-assigned managed identity". Below this, connection details are shown: Managed identity name: "adworks-adf-jc", Managed identity object ID: "0f9d34c-acfc-4ce5-8b4b-9c1300ebda93", and a note about granting Data Factory service managed identity access to the Azure Key Vault. At the bottom of the dialog, a red box highlights the "Create" button.

**Figura 8.-** Creación del linked service para la conexión con azure key-vault y data factory.

## Pipeline para la ingestión de datos desde SQL-Server hacia la capa bronce del datalake

Una vez que se tiene la base de datos lista, podemos iniciar con el proceso de ingestión de datos hacia el data lake en Azure empleando un pipeline con Azure Data Factory. Vamos a considerar un escenario real en el que estaremos trabajando con 7 tablas, 6 dimension tables y 1 fact table. De las 7, 2 tablas van a tener una carga de datos incremental y 5 van a tener una carga completa o tipo Full, o en otras palabras vamos a construir un metadata-driven pipeline, en el cual leeremos un “load-config.txt” para realizar iteraciones por cada tabla e identificar sus parámetros. La figura 9 muestra el contenido del archivo .txt, el cual está almacenado en un container llamado “config-files” dentro data lake.

### load-config.txt

Blob

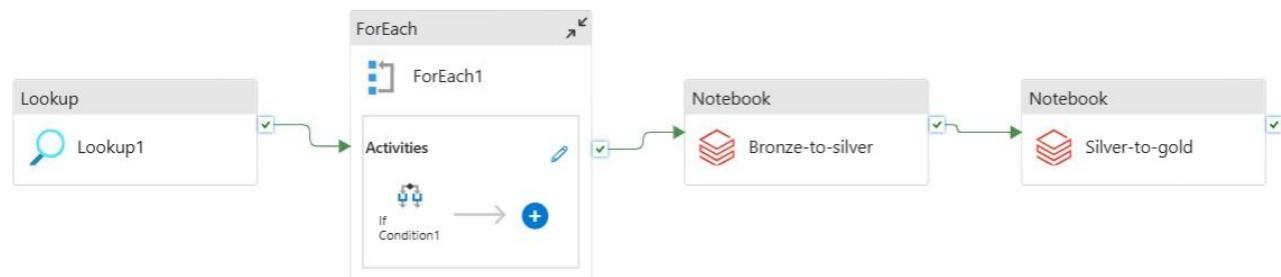
Save Discard Download Refresh Delete

Overview Versions Edit Generate SAS

```
1 database,tablename,loadtype,watermark_table,watermark_column
2 AdventureWorksDM2019,dbo.DimCustomer,Incremental,dbo.customer_watermarktable,CustomerKey
3 AdventureWorksDM2019,dbo.DimDate,Full,
4 AdventureWorksDM2019,dbo.DimGeography,Full,
5 AdventureWorksDM2019,dbo.DimProduct,Full.,
6 AdventureWorksDM2019,dbo.DimProductCategory,Full.,
7 AdventureWorksDM2019,dbo.DimProductSubCategory,Full.,
8 AdventureWorksDM2019,dbo.FactInternetSales,Incremental,dbo.sales_watermarktable,OrderDate
```

**Figura 9.-** Config-file para el copiado de las tablas desde SQL-Server, la columna loadtype define si se trata de una carga tipo “Full” o “Incremental”.

El pipeline consiste en una actividad “Lookup” para leer el archivo de config-file, posteriormente una actividad “ForEach” para iterar cada fila dentro del archivo config que representa cada tabla, ejecuta dentro del foreach el copiado ya sea tipo “Full” o “Incremental”. Posteriormente contiene una actividad tipo “Notebook” de databricks para el procesado de la capa bronce a silver y uno más para la capa silver a gold, con lo anterior tenemos los datos actualizados desde la ingestión y hasta la capa gold de la arquitectura medallón(figura 10).



**Figura 10.-** Pipeline completo para la ingestión, limpieza y transformación de datos desde la base On-Prem SQL Server hacia la capa Gold en el Azure Data Lake.

## Lookup Activity

El objetivo de la actividad es leer el archivo “load-config.txt” que se ubica dentro del contenedor config-files en el data lake. Se requiere conectarse al data lake, por lo que es necesario definir un linked service que apunte hacia el data lake, así como un dataset que permita la lectura de un archivo dellimited text el cual definimos en la pestaña de Author dentro de data factory (figuras 11 y 12). Utilizamos key-vault para la gestión de los secretos y la conexión segura entre los servicios.

**Edit linked service**

Azure Data Lake Storage Gen2 [Learn more](#)

Name \*  
AzureDataLakeStorage1

Description

Connect via integration runtime \* ⓘ  
 AutoResolveIntegrationRuntime

Authentication type  
 Account key

Account selection method ⓘ  
 From Azure subscription  Enter manually

URL \*  
 https://adworksadlsljc.dfs.core.windows.net/

[Storage account key](#) [Azure Key Vault](#)

AKV linked service \* ⓘ  
 AzureKeyVault1

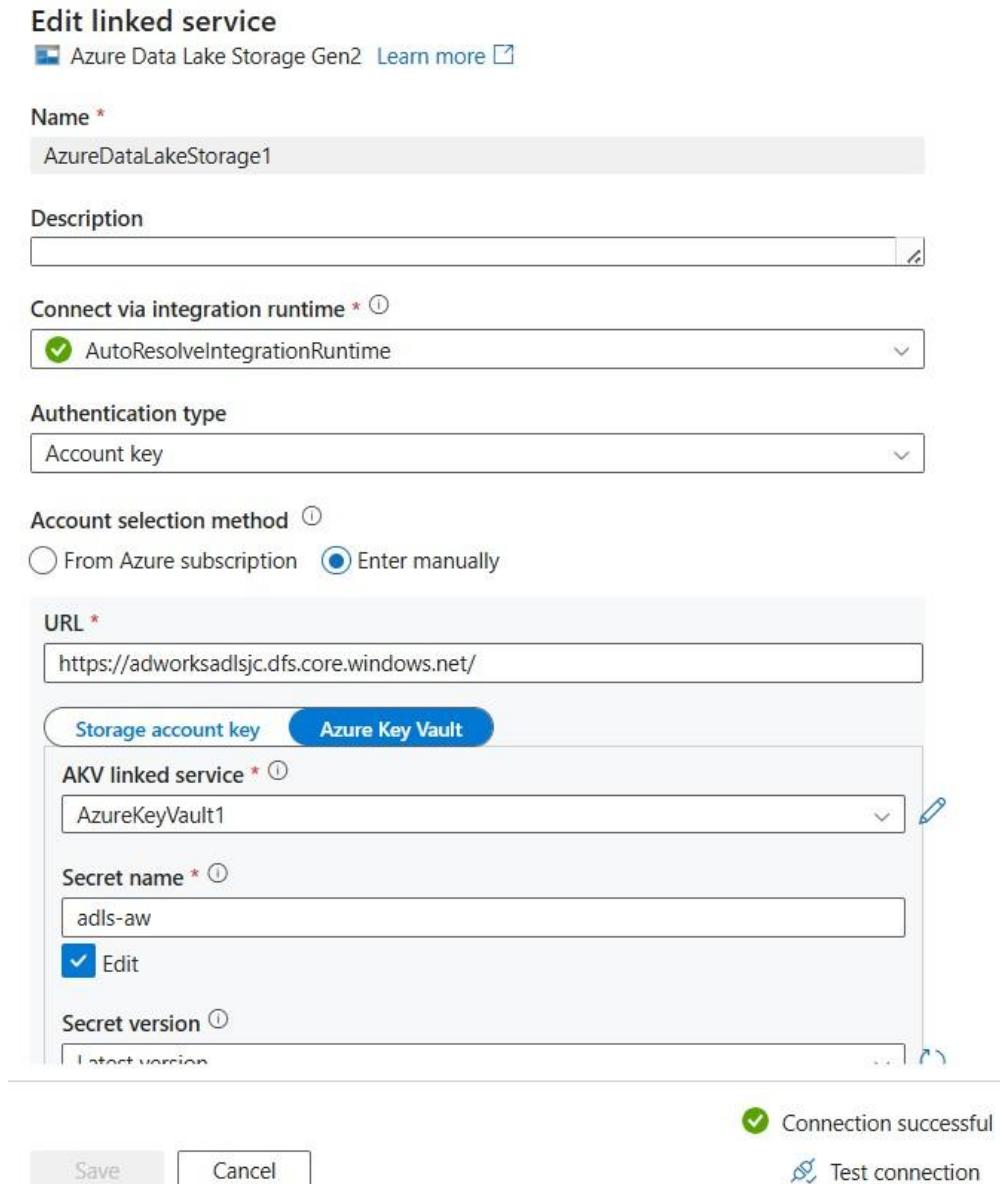
Secret name \* ⓘ  
 adls-aw  
 Edit

Secret version ⓘ  
 Latest version

Connection successful

Test connection

Save Cancel



**Figura 11.-** Linked service para la conexión hacia el data lake en la actividad Lookup1.

Al configurar el dataset, seleccionamos la opción preview data para confirmar la correcta lectura del archivo, el cual nos muestra un resultado como el de la figura 13.

En el canvas, seleccionamos la actividad Lookup1 y en Settings seleccionamos el dataset creado, en nuestro caso llamado “DelimitedText1”, como se muestra en la figura 12.

The screenshot shows the 'Connection' tab of the dataset configuration. The dataset is named 'DelimitedText1'. The 'File path' is set to 'config-files' / 'load-config.txt'. The 'Linked service' is 'AzureDataLakeStorage1'. Other settings include 'Compression type' (No compression), 'Column delimiter' (Comma (,)), 'Row delimiter' (Default (\r,\n, or \r\n)), 'Encoding' (Default(UTF-8)), 'Quote character' (Double quote (')), 'Escape character' (Backslash (\)), 'First row as header' (checked), and 'Null value' (empty).

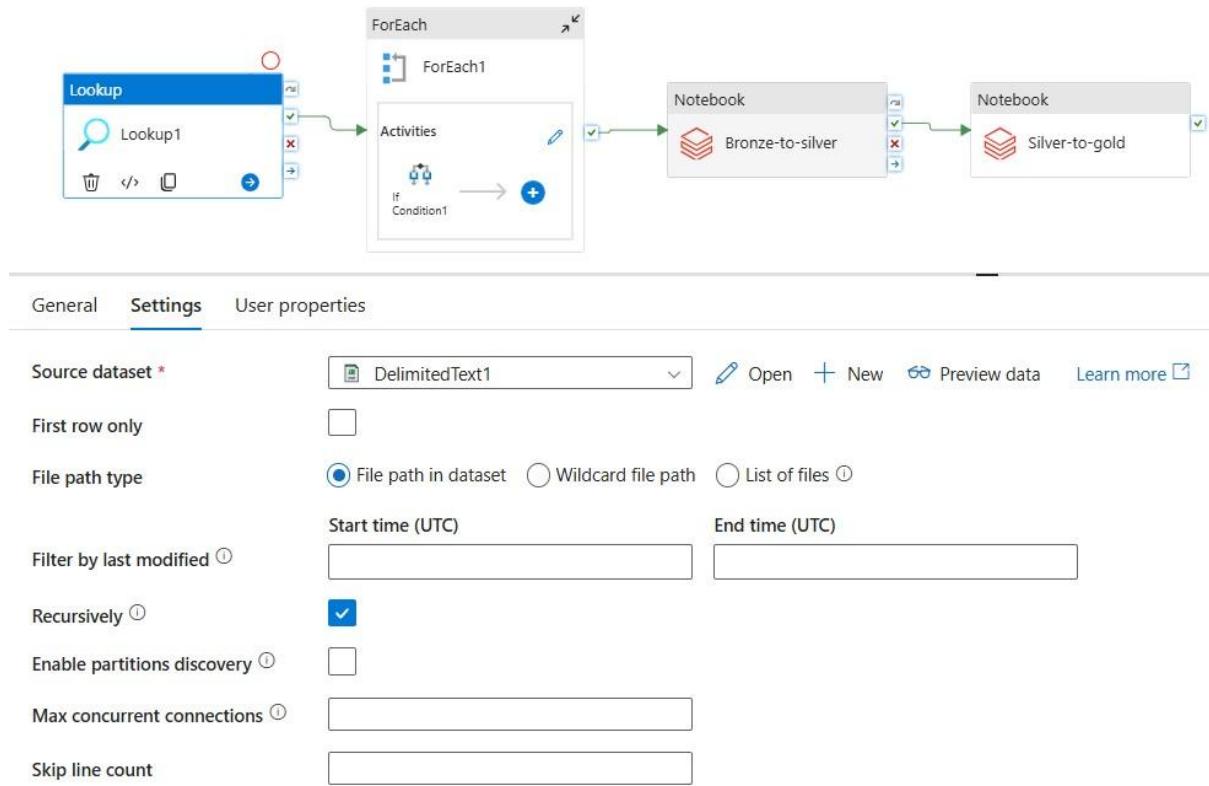
**Figura 12.-** Dataset para la actividad Lookup1, para la lectura del archivo load-config.txt dentro del data lake.

The screenshot shows the 'Preview data' section. The linked service is 'AzureDataLakeStorage1' and the object is 'load-config'. The preview table has columns: #, database, tablename, loadtype, watermark\_table, and watermark\_column. The data shows 7 rows of configurations:

#	database	tablename	loadtype	watermark_table	watermark_column
1	AdventureWorksDM2019	dbo.DimCustomer	Incremental	dbo.customer_watermarktable	CustomerKey
2	AdventureWorksDM2019	dbo.DimDate	Full		
3	AdventureWorksDM2019	dbo.DimGeography	Full		
4	AdventureWorksDM2019	dbo.DimProduct	Full		
5	AdventureWorksDM2019	dbo.DimProductCategory	Full		
6	AdventureWorksDM2019	dbo.DimProductSubCategory	Full		
7	AdventureWorksDM2019	dbo.FactInternetSales	Incremental	dbo.sales_watermarktable	OrderDate

**Figura 13.-** Resultado al seleccionar la opción “preview data” durante la configuración del dataset para la actividad Lookup1.

Con lo anterior hemos terminado de configurar la Lookup activity, procederemos ahora a la actividad ForEach, la cual ejecutará iteraciones para cada fila o registro dentro del archivo load-config.txt como se describe enseguida.



**Figura 14.-** Configuración de la Lookup activity en la pestaña Settings, seleccionando el Source dataset que apunta hacia el config file en el data lake.

## ForEach Activity

En el foreach leemos el resultado de la actividad lookup, para tener una lista con los datos del archivo de configuración. Seleccionamos la actividad en el canvas, seleccionamos Settings, Items, Add dynamic content y colocamos la expresión que se muestra en la figura 15 y en el siguiente snippet:

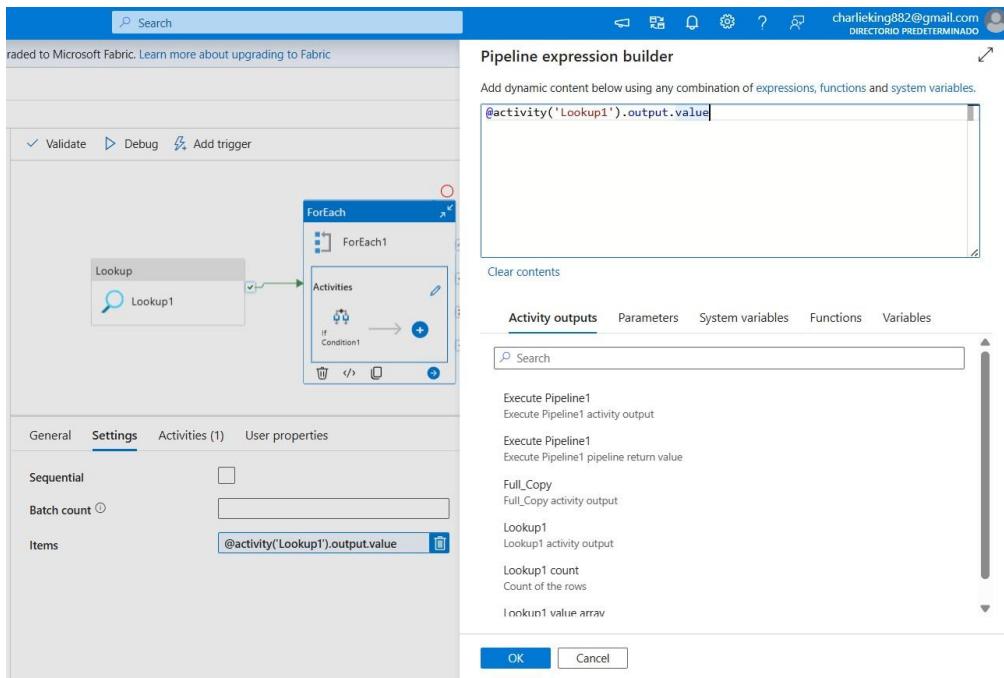
---

```
@activity('lookup1').output.value
```

---

En la figura 16 podemos visualizar el resultado que obtenemos al correr la actividad lookup1, en la que podemos notar una lista con cada registro y los componentes de cada registro como el database, tablename, loadtype, watermark\_table y watermark\_column, éstos valores nos serán de utilidad al construir las siguientes actividades del pipeline.

Ahora vamos a incluir las actividades dentro del foreach. Seleccionamos en editar y abrimos un nuevo canvas. En el buscador de actividades, colocamos if condition y jalamos la activity. La actividad If Condition nos permitirá evaluar si se trata de una carga de datos de tipo "Full" o completa o una carga "Incremental", para nuestro ejemplo 2 tablas serán cargas incrementales y 5 serán "Full" (figura 17).

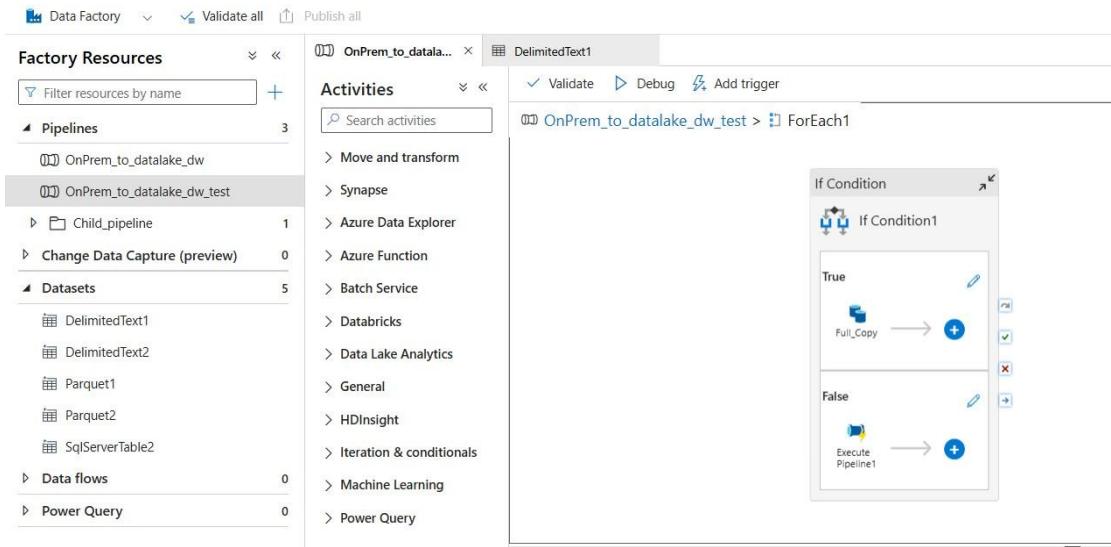


**Figura 15.-** Expresión para obtener la lista con los registros en el archivo load-config.

```
Output
Copy to clipboard

{
  "count": 7,
  "value": [
    {
      "database": "AdventureWorksDW2019",
      "tablename": "dbo.DimCustomer",
      "loadtype": "Incremental",
      "watermark_table": "dbo.customer_watermarktable",
      "watermark_column": "CustomerKey"
    },
    {
      "database": "AdventureWorksDW2019",
      "tablename": "dbo.DimDate",
      "loadtype": "Full",
      "watermark_table": null,
      "watermark_column": null
    },
    {
      "database": "AdventureWorksDW2019",
      "tablename": "dbo.DimGeography",
      "loadtype": "Full",
      "watermark_table": null,
      "watermark_column": null
    },
    {
      "database": "AdventureWorksDW2019",
      "tablename": "dbo.DimProduct",
      "loadtype": "Full",
      "watermark_table": null,
      "watermark_column": null
    },
    {
      "database": "AdventureWorksDW2019",
      "tablename": "dbo.DimProductCategory",
      "loadtype": "Full",
      "watermark_table": null,
      "watermark_column": null
    },
    ...
  ]
}
```

**Figura 16.-** Lectura del resultado de la actividad lookup.



**Figura 17.-** Actividad If-Condition para definir la actividad a seguir en función del tipo de carga de datos ya sea incremental o completa(full).

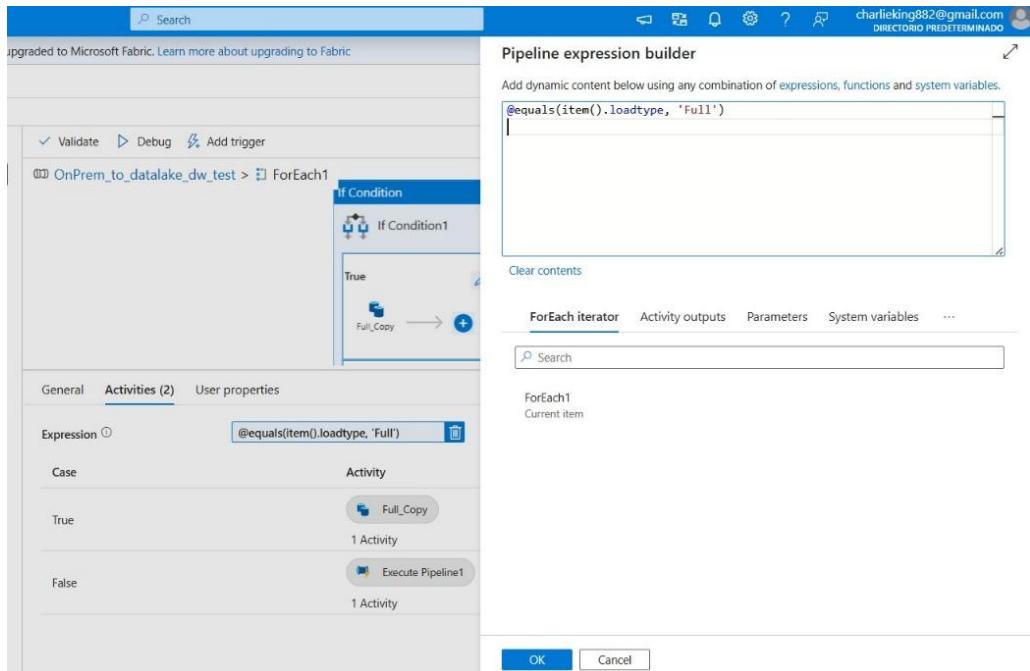
Seleccionamos la actividad, nos vamos a Activities, Expression, Add Dynamic content y colocamos la expresión siguiente:

---

```
@equals(item().loadtype, 'Full')
```

---

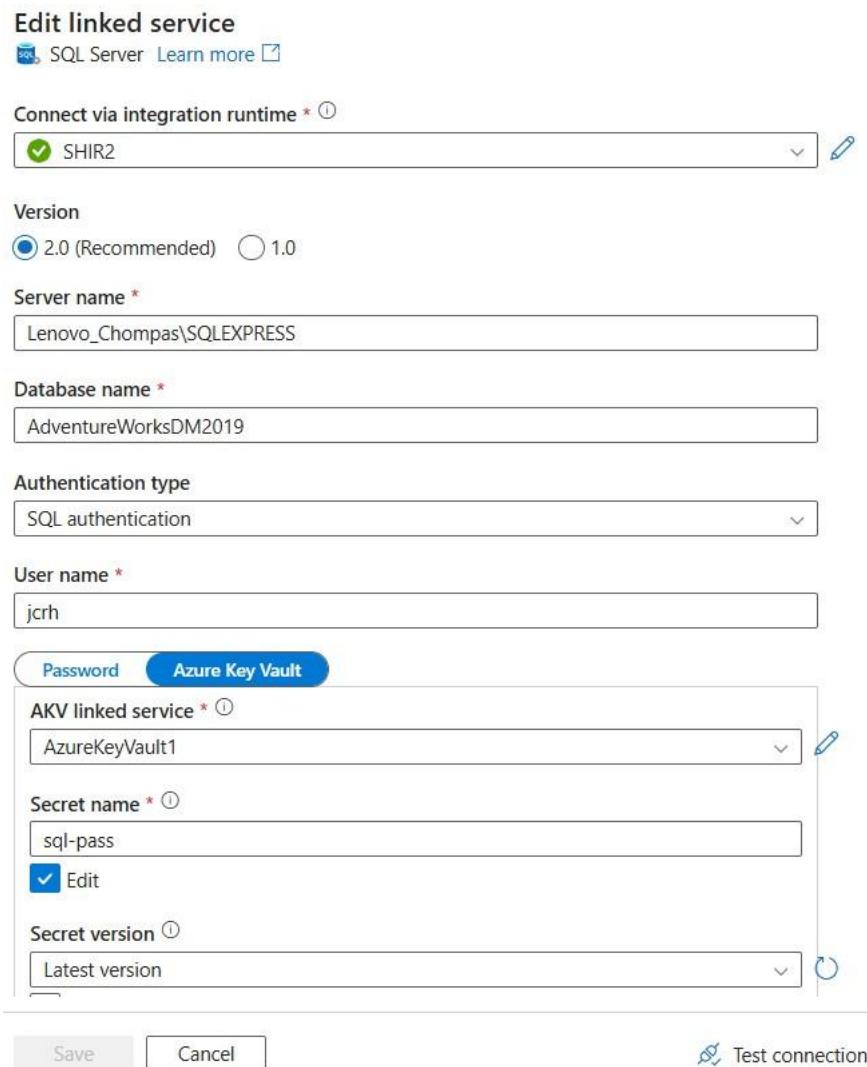
En la figura 18 podemos ver la condición a evaluar dentro de la If Condition.



**Figura 18.-** Definición de la condición if al leer la columna load type del archivo load-config.txt a través de la lookup activity.

## Full Copy Activity

Dentro del if Condition en la sección de True, es decir si la carga es tipo Full, generamos en el canvas una Copy activity. Para poder copiar los datos desde la base en SQL Server vamos a requerir definir para la source data, un linked service y un dataset. Nos vamos al apartado de manage, linked service y creamos uno nuevo para conectarnos con SQL Server, seleccionamos el nombre del servidor, la base de datos, tipo de autenticación, utilizamos el key-vault y seleccionamos el nombre del secreto como se muestra en la figura 19.



**Figura 19.-** Definición del linked service para la conexión hacia la base de datos dentro de SQL-Server.

Ya que tenemos el linked service, procedemos a crear el dataset. En la pestaña de Author, seleccionamos new dataset, colocamos el linked service creado y realizamos una prueba de conexión, como se muestra en la figura 20.

**Figura 20.-** Creación del dataset para la lectura de datos desde la fuente (SQL-Server).

En la sección Source, elegimos el dataset, en la parte de use query elegimos query y en el expresión builder colocamos la siguiente consulta para leer cada tabla:

---

```
@concat('select * from ', item().tablename)
```

---

En la figura 21 se muestra la configuración de la pestaña source en la Full\_copy activity.

**Figura 21.-** Configuración de la fuente de datos para la copy activity tipo Full.

Para la parte del sink, se requiere tener un linked service que apunte hacia el data lake y un dataset definido como parquet file. El linked service ya lo hemos definido al leer el archivo load-config.txt, por lo que solo es necesario configurar el dataset para guardar las tablas como archivos parquet. Al copiar las tablas vamos a utilizar la columna “tablename” para generar un folder y un archivo con el mismo nombre, para lo anterior creamos dos parámetros dentro del dataset, uno denominado “directory” y otro llamado “filename”(figura 22).

Name	Type	Default value
directory	String	<input type="button" value="Value"/>
fileName	String	<input type="button" value="Value"/>

**Figura 22.-** Definición de parámetros dentro del dataset para copiar los datos hacia la capa bronce del data lake.

En la pestaña “connection”, seleccionamos el linked service el data lake, realizamos una prueba de conexión, en la sección de File path, colocamos el nombre del contenedor (bronze), en directorio seleccionamos “add dynamic content” y colocamos la siguiente expresión, la cual nos permite leer de los resultados en la lista de la lookup activity.

---

```
@dataset().directory
```

---

Para el fileName colocamos la expresión siguiente:

---

```
@dataset().fileName
```

---

La figura 23 muestra la definición de los parámetros para la definición del dataset.

The screenshot shows the Azure Data Factory Studio interface. On the left, the 'Factory Resources' sidebar lists Pipelines, Datasets, Data flows, and Power Query. Under 'Datasets', 'Parquet1' is selected. The main pane displays the dataset properties. The 'Connection' tab shows 'Linked service' set to 'AzureDataLakeStorage1' with a 'Connection successful' status. The 'Schema' tab shows 'Parquet' selected. The 'Parameters' tab shows the file path expression: 'bronze/@dataset().directory/@dataset().fileName'. The 'Compression type' is set to 'snappy'.

**Figura 23.-** Definición del dataset para guardar los datos en formato parquet dentro de la capa bronce del data lake.

Procedemos ahora a configurar la pestaña de Sink en la copy activity. Seleccionamos el sink dataset recién configurado, en la parte de dataset properties colocamos para el directorio y filename la siguiente expresión:

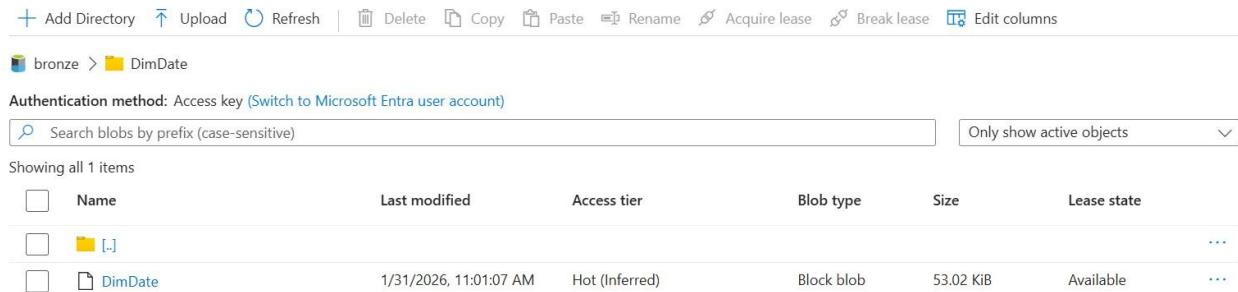
```
@split(item().tablename, '.')[1]
```

La figura 24 muestra la configuración de la pestaña Sink en la copy activity.

The screenshot shows the Azure Data Factory Studio interface. On the left, the 'Activities' sidebar lists various options like Move and transform, Synapse, Azure Data Explorer, etc. The main pane shows a pipeline structure: 'OnPrem\_to\_datalake\_dw\_test > ForEach1 > If Condition1 - True activities'. A 'Copy data' activity is selected. The 'Sink' tab is active. The 'Sink dataset' is set to 'Parquet1'. In the 'Dataset properties' section, there are two rows: 'directory' with the value '@split(item().tablename, \'.\')[1]' and 'filename' with the value '@split(item().tablename, \'.\')[1]'. Other tabs like General, Source, Mapping, Settings, and User properties are also visible.

**Figura 24.-** Definición de la pestaña Sink para la copy activity tipo Full.

Una vez que nosotros corremos el pipeline como resultado tenemos un archivo tipo parquet con el nombre de la tabla, por ejemplo en la figura 25 podemos ver la tabla llamada DimDate en el contenedor bronce del datalake, confirmando una correcta carga de los datos.



The screenshot shows the Azure Storage Explorer interface. At the top, there are navigation buttons: Add Directory, Upload, Refresh, Delete, Copy, Paste, Rename, Acquire lease, Break lease, and Edit columns. Below the buttons, the path is shown as bronze > DimDate. A note says "Authentication method: Access key (Switch to Microsoft Entra user account)". There is a search bar with placeholder "Search blobs by prefix (case-sensitive)" and a dropdown menu "Only show active objects". Below the search bar, it says "Showing all 1 items". A table lists one item:

<input type="checkbox"/>	Name	Last modified	Access tier	Blob type	Size	Lease state
<input type="checkbox"/>	DimDate	1/31/2026, 11:01:07 AM	Hot (Inferred)	Block blob	53.02 KiB	Available

**Figura 25.-** Confirmación de la carga de datos en la capa bronce donde se muestra la carpeta DimDate así como el archivo parquet DimDate.

## Incremental Copy Activity

Una vez que hemos finalizado con el copy activity para una carga de datos full, procedemos a construir la copy activity para la carga de datos incremental. Este tipo de carga de datos tiene una mayor dificultad en comparación a la carga tipo “Full”, dado que requiere identificar los nuevos registros que se han añadido desde la última corrida del pipeline. Para mayor detalle pueden consultar el tutorial que un servidor realizó basado en la documentación de Microsoft Azure, el link del repositorio y documentación son los siguientes:

Repositorio:

<https://github.com/jcreyesh/Carga-Incremental-de-datos-con-Azure>

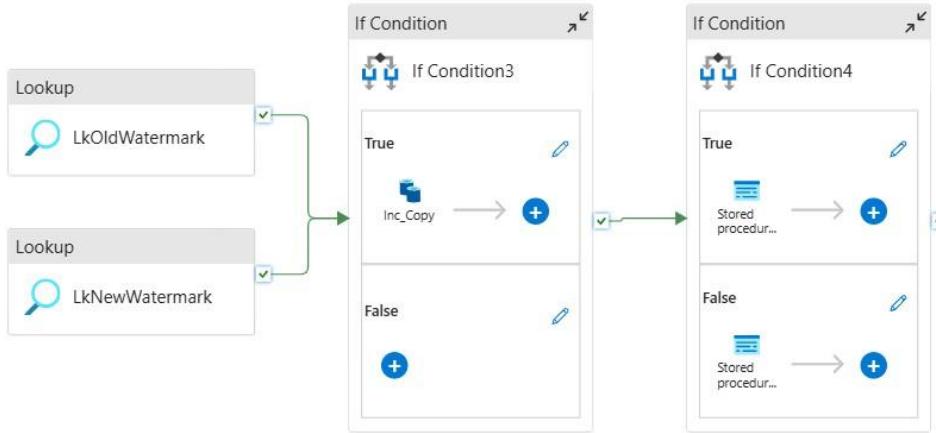
Documentación de Microsoft Azure:

<https://learn.microsoft.com/en-us/azure/data-factory/tutorial-incremental-copy-powershell>

En la figura 17 veíamos que el ifCondition nos evaluaba si se trataba de una Full Copy Activity o una Incremental Copy Activity, en caso de ser incremental nos enviaba a una actividad Execute pipeline, esto se refiere a que tenemos un child pipeline el cual podemos llamar desde nuestro pipeline principal o Parent pipeline. Creamos un nuevo pipeline con el que trabajaremos las cargar incrementales. El pipeline child se ve como se muestra en la figura 26.

## Lookup Activity – ‘LkOldWatermark’

Dentro de SQL-Server vamos a crear dos tablas para guardar la últimas fechas de actualización o el último identificador, una para nuestra tabla de ventas o FactInternetSales la cual utilizará la columna de OrderDate como watermark y otra tabla para la tabla DimCustomer la cual utilizará la columna CustomerKey como watermark.



**Figura 26.-** Child pipeline dentro de la execute pipeline activity para la carga de datos incremental.

Para la FactSales utilizamos el siguiente código en SSMS:

---

```
create table dbo.sales_watermarktable
(
TableName varchar(255),
WatermarkValue datetime,
);

insert into dbo.sales_watermarktable
values ('data_source_table', '1/1/2010 12:00:00 AM');
```

---

Para la DimCustomer creamos la tabla dentro de Sql-Server:

---

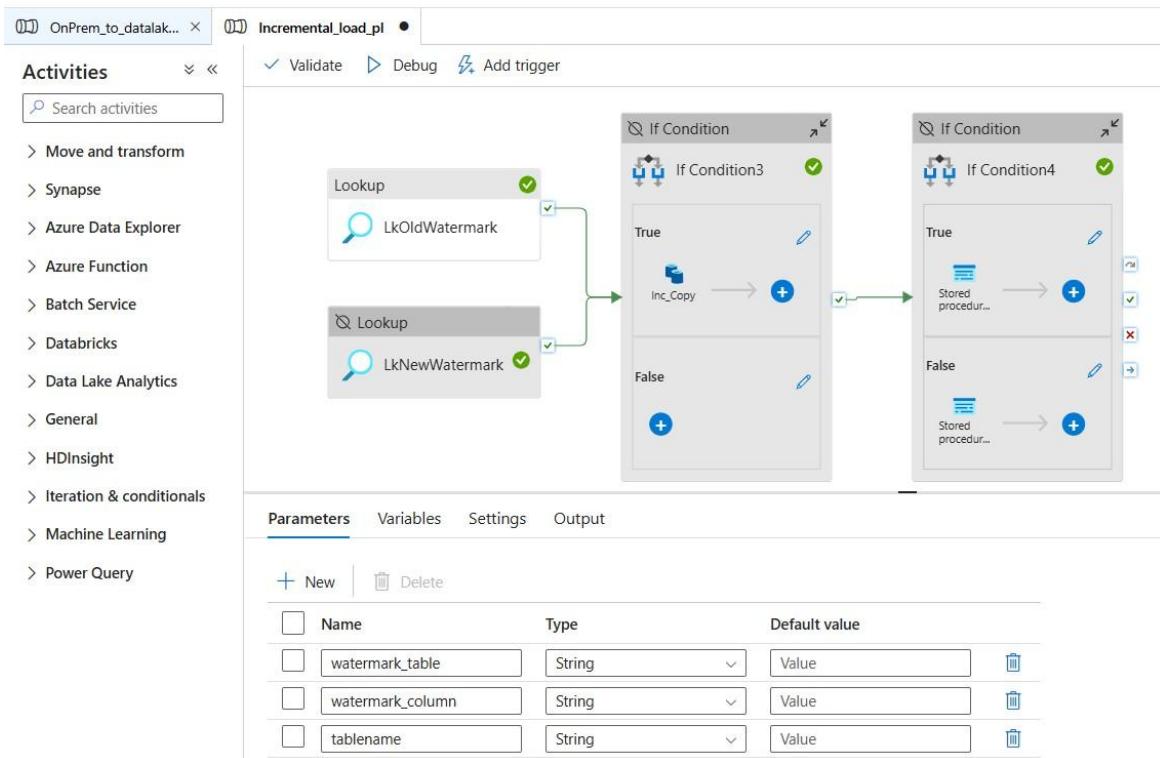
```
create table dbo.customer_watermarktable
(
TableName varchar(255),
WatermarkValue int,
);

insert into dbo.customer_watermarktable
values ('data_source_table', 1)
```

---

En estas dos watermark tables es en donde las actividades LookupOld y LookupNew van a obtener los valores del último registro actualizado en función de la fecha o del identificador id. Para lo anterior, como vamos a utilizar parámetros dinámicos en el pipeline, es decir el pipeline debe ejecutarse cada vez que identifique una carga de tipo incremental al realizar las iteraciones del archivo “load-config.txt”, no obstante esa lectura del archivo se realizó en el Parent pipeline, por lo que entra en juego un concepto que se conoce como parámetros de pipeline, es decir parámetros que podamos utilizarlos si es necesario en un pipeline anidado.

En el canvas en el child pipeline, seleccionamos el apartado de “Parameters” y declaramos 3 parámetros de tipo string, watermark\_table, watermark\_column y tablename como se muestra en la figura 27.



**Figura 27.- Declaración de parámetros dentro del child pipeline.**

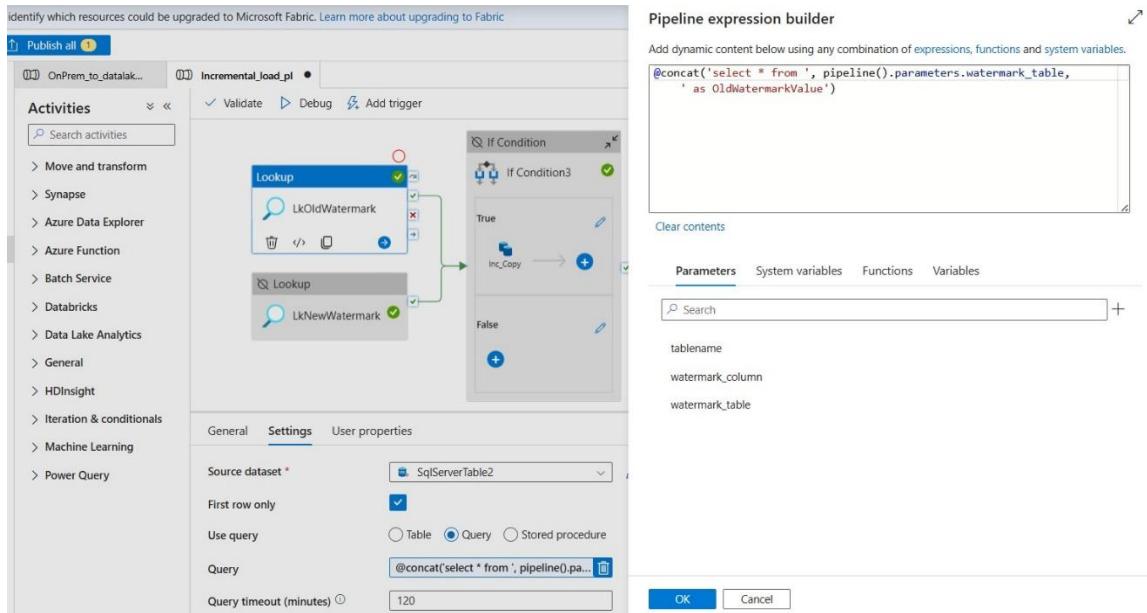
Una vez con lo anterior, podemos configurar la primera actividad Lookup, la seleccionamos dentro del canvas y nos vamos a la parte de Settings, como source dataset utilizamos el mismo vinculado a Sql-Server, en use Query colocamos Query, abrimos el expression builder y colocamos lo siguiente, en la cual utilizaremos los parámetros del pipeline que hemos declarado:

---

```
@concat('select * from ', pipeline().parameters.watermark_table, ' as OldWatermarkValue')
```

---

Al correr el pipeline solo con la primer lookup activity obtenemos como resultado los datos de la tabla como TableName y el WatermarkValue, como se puede ver en la figura 29 en el apartado de FirstRow.



**Figura 28.-** Configuración de la primer actividad lookup con la cual se obtiene el valor anterior de actualización para las cargas incrementales.

**Output**

```
{
  "firstRow": {
    "TableName": "data_source_table",
    "WatermarkValue": "2010-01-01T00:00:00Z"
  },
  "effectiveIntegrationRuntime": "SHIR2",
  "billingReference": {
    "activityType": "PipelineActivity",
    "billableDuration": [
      {
        "meterType": "SelfhostedIR",
        "duration": 0.01666666666666666,
        "unit": "Hours"
      }
    ],
    "totalBillableDuration": [
      {
        "meterType": "AzureIR",
        "duration": 0.06666666666666667,
        "unit": "Hours"
      }
    ]
  },
  "durationInQueue": {
    "integrationRuntimeQueue": 4
  }
}
```

**Figura 29.-** Resultado que la primer actividad lookup mostrando el valor inicial para la primera corrida del pipeline.

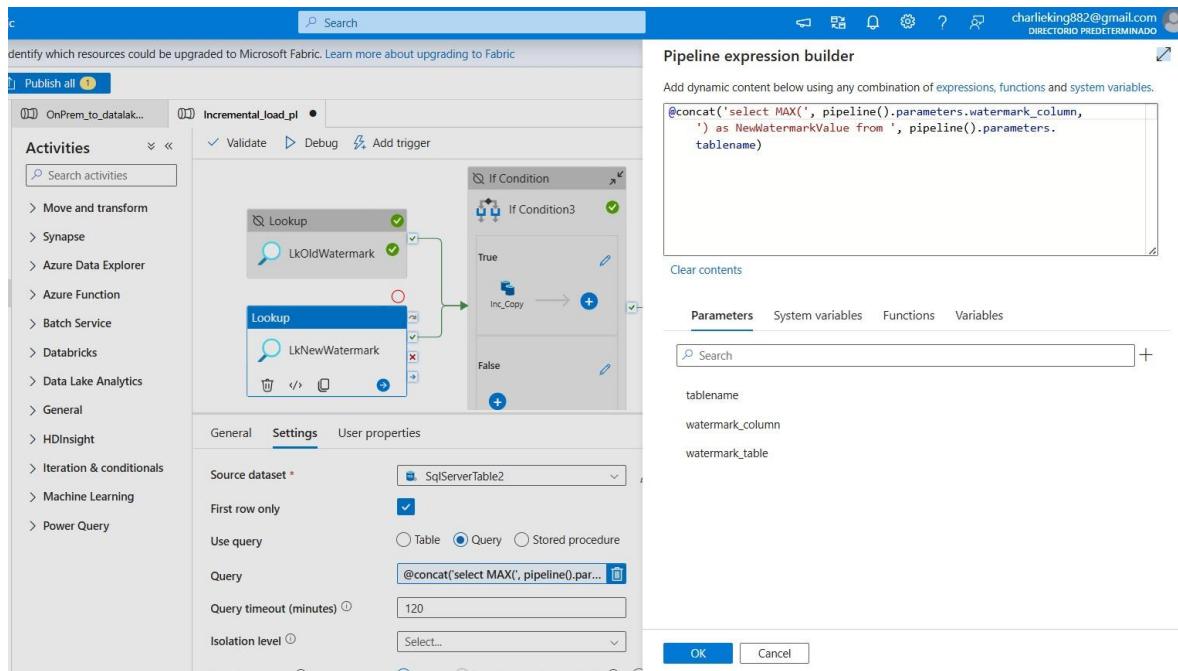
## Lookup Activity – ‘LkNewWatermark’

Para esta actividad lookup, vamos a obtener el máximo valor de una columna utilizada como watermark, puede ser una fecha o timestamp o algún identificador. Para el caso de la tabla FactSalesInternet utilizaremos la columna OrderDate y para la tabla DimCustomer utilizaremos la columna CustomerKey, como se define en el load-config file en la columna watermark\_column.

Seleccionamos la actividad, nos vamos al apartado Settings, use query y colocamos la query siguiente para obtener el máximo valor ya sea en fecha o en id:

```
@concat('select MAX(', pipeline().parameters.watermark_column, ') as NewWatermarkValue from ', pipeline().parameters.tablename)
```

En la figura 30 se muestra la configuración de la segunda Lookup activity con la query anterior.



**Figura 30.-** Configuración de la segunda actividad lookup con la cual se obtiene el máximo valor de la columna utilizada como watermark para identificar el delta o los registros añadidos.

Al correr la actividad podemos ver que el resultado que nos arroja tiene como máxima fecha el día 28-01-2014, la cual tomará como valor más reciente(figura 31). En otras palabras para la primera corrida del pipeline tomará el delta de datos con fechas  $01-01-2010 < \Delta t \leq 28-01-2014$ .

**Output**

**Copy to clipboard**

```
{
  "firstRow": {
    "NewWatermarkValue": "2014-01-28T00:00:00Z"
  },
  "effectiveIntegrationRuntime": "SHIR2",
  "billingReference": {
    "activityType": "PipelineActivity",
    "billableDuration": [
      {
        "meterType": "SelfhostedIR",
        "duration": 0.01666666666666666,
        "unit": "Hours"
      }
    ],
    "totalBillableDuration": [
      {
        "meterType": "AzureIR",
        "duration": 0.06666666666666667,
        "unit": "Hours"
      }
    ]
  },
  "durationInQueue": {
    "integrationRuntimeQueue": 1
  }
}
```

**Figura 31.-** Resultado que la segunda actividad lookup mostrando el máximo valor para la primer corrida del pipeline.

### Primera IfCondition Activity

En este primer if-condition evaluamos si el NewWatermarkValue es mayor que el OldWatermarkValue, es decir registros que no se hayan añadido aún a nuestros datos. En la sección de activites, en el expression builder utilizamos la función @greater de azure para realizar el comparativo entre fechas o watermarkvalues:

---

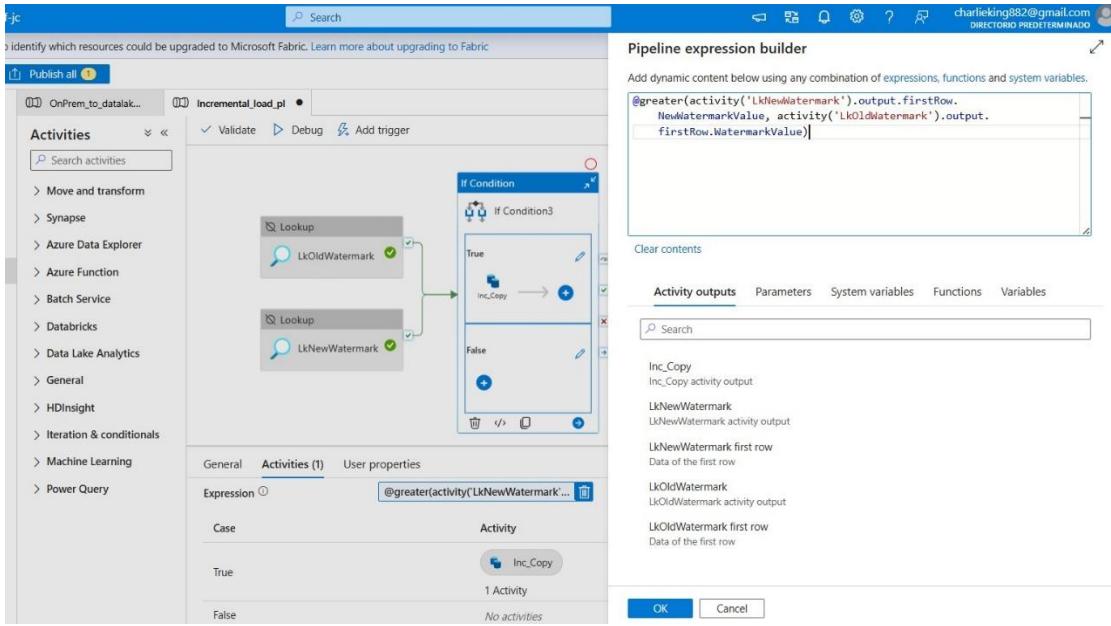
```
@greater(activity('LkNewWatermark').output.firstRow.NewWatermarkValue,
activity('LkOldWatermark').output.firstRow.WatermarkValue)
```

---

En la figura 32 podemos visualizar la configuración de la actividad ifCondition. En caso de que sea verdadera la condición se realizará un copiado incremental de los datos, mientras que si es falso no se ejecuta actividad alguna, por tal razón se ve vacía la parte de False. Dentro de la parte verdadera colocamos una Copy Activity, a la que llamamos “Inc\_Copy”.

### Copy data Activity – Inc\_Copy

En la sección de source seleccionamos el dataset vinculado a Sql-Server, indicamos use query, y colocamos la siguiente consulta en el expression builder:



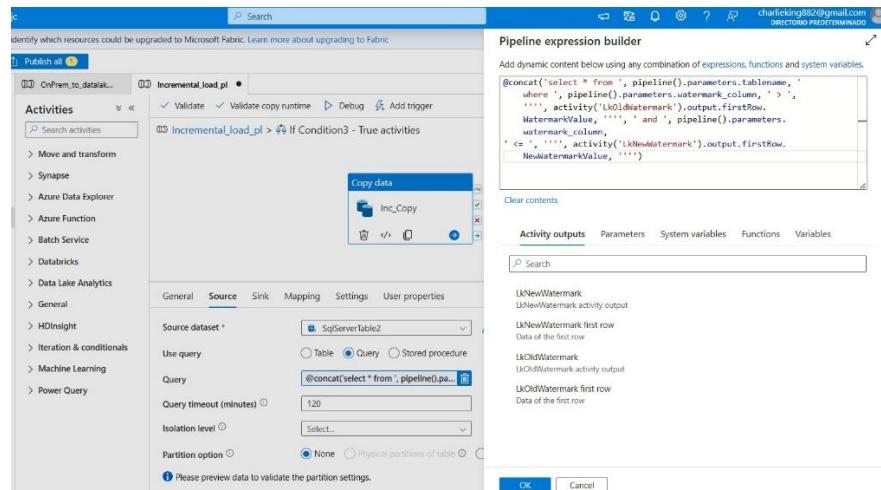
**Figura 32.-** Configuración de la actividad ifCondition para evaluar si existe un delta de datos a copiar.

---

```
@concat('select * from ', pipeline().parameters.tablename, ' where ',
pipeline().parameters.watermark_column, ' > ', '',
activity('LkOldWatermark').output.firstRow.WatermarkValue, '',
' and ',
pipeline().parameters.watermark_column,
' <= ', '',
activity('LkNewWatermark').output.firstRow.NewWatermarkValue, '')
```

---

En la figura 33 visualizamos la configuración de la pestaña Source en la Inc\_Copy. La query anterior no es más que filtrar en cada tabla los valores entre el OldWatermarValue y el NewWatermarkValue con sintaxis dentro de azure data factory. En Sql-Server escribiríamos la query como se muestra en la figura 34.



**Figura 33.-** Configuración de la pestaña source dentro de la Inc\_Copy Activity.

```

1 select * from dbo.FactInternetSales
2 where OrderDate > '2010-01-01 00:00:00.000'
3 and OrderDate <= '2014-02-28 00:00:00.000'

```

Results

ProductKey	OrderDateKey	DueDateKey	ShipDateKey	CustomerKey	PromotionKey	CurrencyKey	SalesTerritoryKey	SalesOrderNumber	SalesOrderLineNumber	RevisionNumber	OrderQuantity	UnitPrice	ExtendedAmount	UnitPrice
1 222	20131002	20131014	20131009	13406	1	100	8	S067621	2	1	1	34.99	34.99	0
2 586	20131002	20131014	20131009	29705	1	100	8	S067622	2	1	1	742.35	742.35	0
3 214	20131002	20131014	20131009	27025	1	100	8	S067623	1	1	1	54.99	54.99	0
4 561	20131002	20131014	20131009	24700	1	100	7	S067623	2	1	1	2384.07	2384.07	0
5 477	20131002	20131014	20131009	24700	1	100	7	S067623	3	1	1	4.99	4.99	0
6 479	20131002	20131014	20131009	24700	1	100	7	S067623	4	1	1	8.99	8.99	0
7 490	20131002	20131014	20131009	24700	1	100	7	S067623	1	1	1	53.99	53.99	0
8 583	20131002	20131014	20131009	24305	1	6	9	S067624	1	1	1	1700.99	1700.99	0
9 225	20131002	20131014	20131009	24305	1	6	9	S067624	2	1	1	8.99	8.99	0
10 488	20131002	20131014	20131009	24305	1	6	9	S067624	3	1	1	53.99	53.99	0
11 388	20131002	20131014	20131009	26022	1	6	9	S067625	1	1	1	1120.49	1120.49	0
12 217	20131002	20131014	20131009	26022	1	6	9	S067625	2	1	1	34.99	34.99	0
13 104	20131002	20131014	20131009	24777	1	6	9	S067626	1	1	1	539.99	539.99	0

Query executed successfully.

**Figura 34.-** Query dentro de SQL-Server similar a la que se coloca como contenido dinámico en Azure Data Factory. Para la primer corrida del pipeline nos arrojaría los 60,398 registros de ordenes en la Fact table.

En la sección de Sink, seleccionamos el dataset Parquet1, previamente definido. En la sección de dataset properties colocamos de manera dinámica los parámetros para el directory y el FileName, como sigue:

Directory:

---

```
@split(pipeline().parameters.tablename, '.')[1]
```

---

FileName:

---

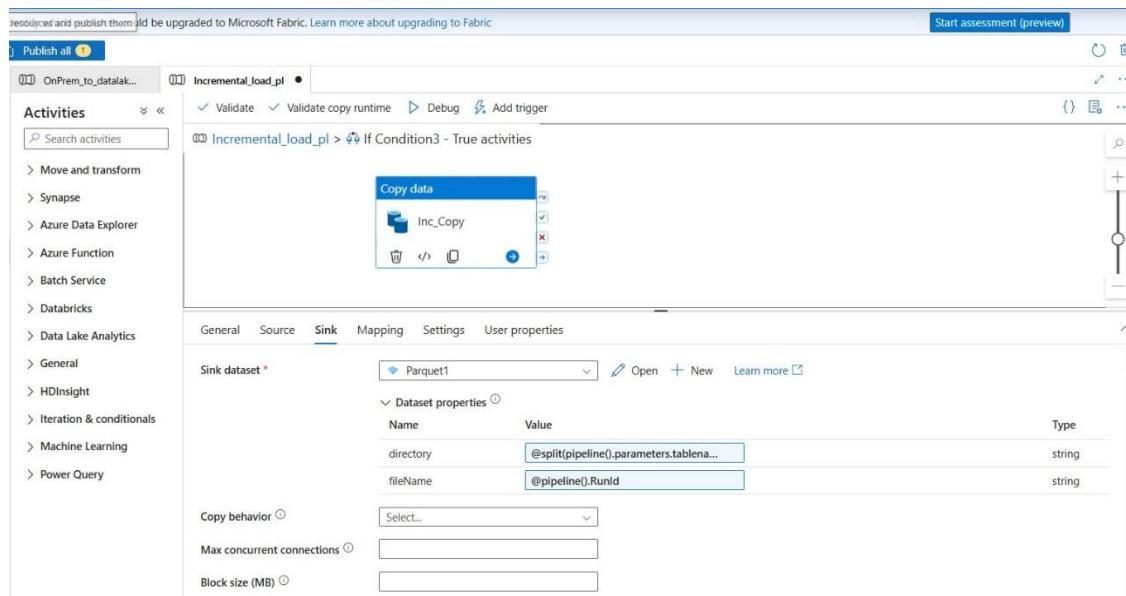
```
@pipeline().RunId
```

---

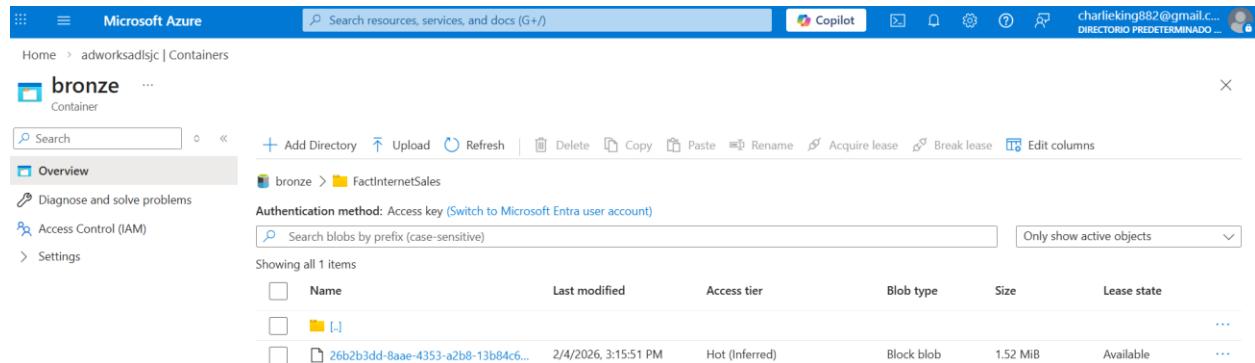
Es decir, al copiar los datos creará una carpeta con el nombre de la tabla, por ejemplo para la fact table creará la carpeta “FactInternetSales” y dentro de la carpeta irá añadiendo los archivos parquet con los datos incrementales, a cada archivo lo nombrará con el identificador de la corrida del pipeline. La figura 35 muestra la pestaña Sink de configuración para la Inc\_Copy. La figura 36 muestra el contenedor dentro de la carpeta bronce del data lake en la que podemos ver el resultado tras la primera corrida del pipeline, donde verificamos que se encuentra el archivo nombrado con el identificador de la corrida.

### Segunda IfCondition Activity

La Segunda ifCondition activity nos ayuda a definir el store procedure a aplicar en función de si se trata de la FactInternetSales table o la DimCustomer table, dado que cada una utiliza una columna distinta como valor Watermark o identificador para detectar los nuevos registros en la tabla. FactInternetSales, utiliza la columna OrderDate con un tipo de dato datetime, mientras que la DimCustomer utiliza el CustomerKey con un tipo de dato integer.



**Figura 35.-** Configuración de la actividad Inc\_Copy en la pestaña Sink.



**Figura 36.-** Carga incremental correcta de los datos en la Fact table dentro del conteneder bronce en el Azure Data Lake.

Para nuestra base de datos, solo tenemos 2 tablas distintas, así que solo existen 2 escenarios para el if condition, podríamos definir el true cuando se trata de la FactInternetSales, como se muestra en la figura 37, dentro del expresión builder colocamos lo siguiente:

---

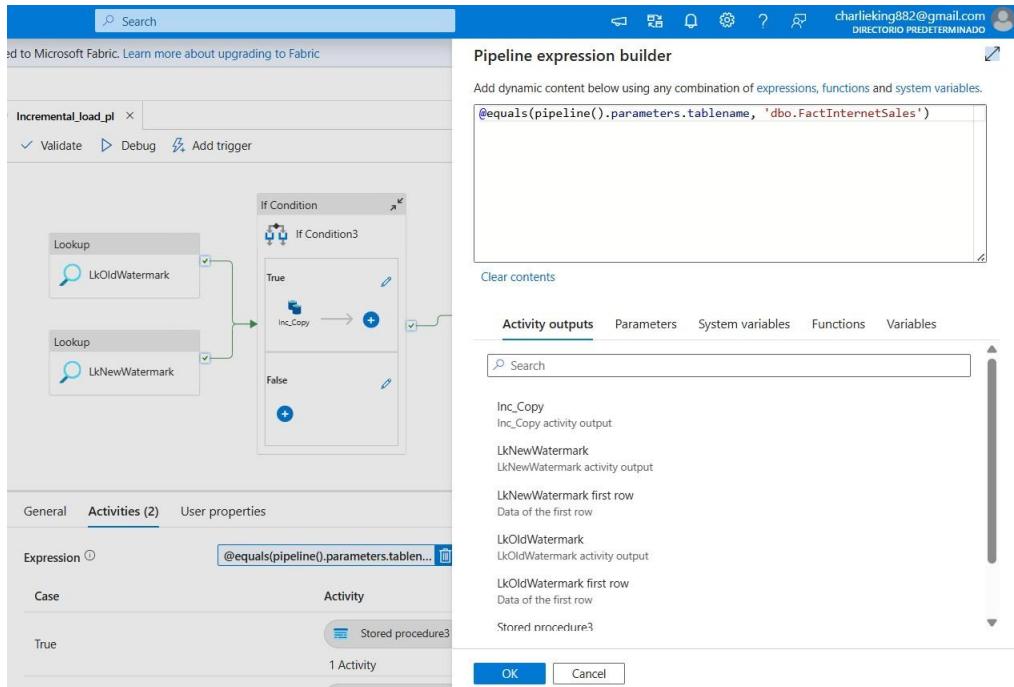
```
@equals(pipeline().parameters.tablename, 'dbo.FactInternetSales')
```

---

En caso de ser true o false, se ejecutará uno de los dos store procedures disponibles dentro de Sql-Server para actualizar las watermark tables.

### Store Procedure Activity – True

En caso de entrar en el true, se ejecutará la actividad Store procedure dentro de Sql-Server para actualizar el WatermarkValue en la tabla FactInternetSales. En la figura 38 podemos visualizar la configuración de la actividad store procedure.



**Figura 37.-** Configuración del expresión builder dentro de la seunda IfCondition.

Primero definimos el store procedure dentro de Sql-Server, como sigue:

---

```
create procedure usp_write_watermark @LastModifiedtime datetime, @TableName
varchar(50)
as
begin
update dbo.sales_watermarktable
set WatermarkValue = @LastModifiedtime
where TableName = @TableName
end
```

---

Ahora procedemos a configurar la actividad. Nos vamos al canvas, seleccionamos el linked service hacia Sql-Server, elegimos el store procedure e indicamos de manera dinámica los parámetros como sigue:

LastModifiedtime:

---

```
@activity('LkNewWatermark').output.firstRow.NewWatermarkValue
```

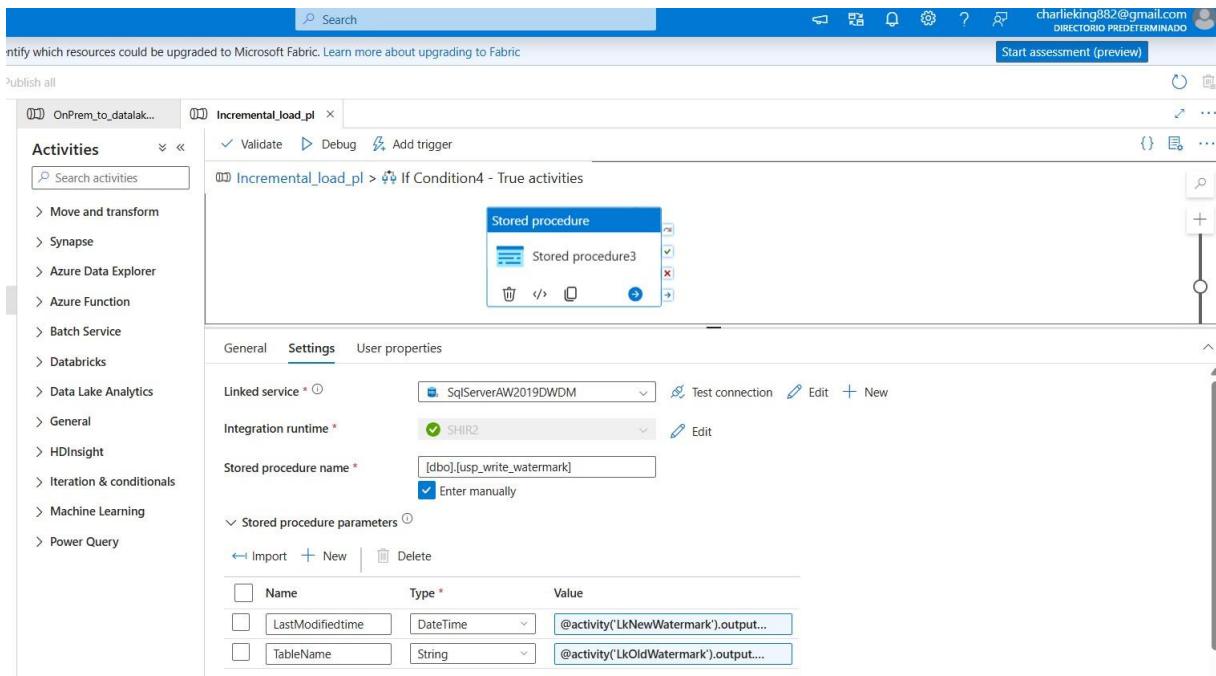
---

TableName:

---

```
@activity('LkOldWatermark').output.FirstRow.TableName
```

---



**Figura 38.-** Configuración de la actividad store procedure para la actualización de la tabla dbo.sales\_watermarktable dentro de Sql-Server.

### Store Procedure Activity – False

En caso de que la condición sea false, ejecutara el store procedure sobre la tabla dbo.customer\_watermarktable. De igual manera, previo a la configuración dentro de Azure data factory, creamos el store procedure en Sql-Server, como sigue:

---

```
create procedure usp_write_watermark_cust @LastCustomer int, @TableName varchar(50)
as
begin
update dbo.customer_watermarktable
set WatermarkValue = @LastCustomer
where TableName = @TableName
end
```

---

Los parámetros los definimos como sigue:

LastModifiedtime:

---

```
@activity('LkNewWatermark').output.firstRow.NewWatermarkValue
```

---

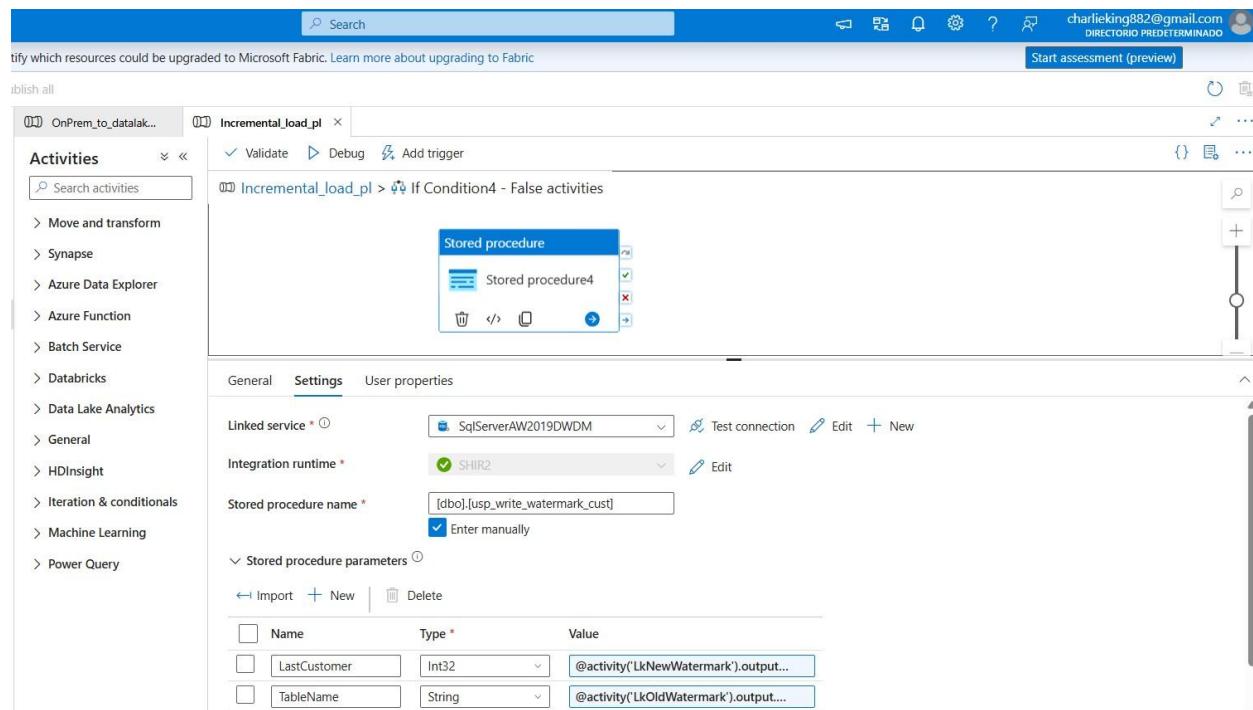
TableName:

---

```
@activity('LkOldWatermark').output.FirstRow.TableName
```

---

En la figura 39 podemos visualizar el canvas con la configuración del segundo store procedure para actualizar la segunda tabla de control dbo.customer\_watermarktable.



**Figura 39.-** Configuración de la actividad store procedure para la actualización de la tabla dbo.customer\_watermarktable dentro de Sql-Server.

Con lo anterior, hemos finalizado la construcción del pipeline hasta la etapa de ingestión de datos, es decir copiar los datos desde Sql-Server hasta la capa bronce en la arquitectura medallón, procederemos a configurar el espacio de trabajo dentro de databricks.

## Configuración para la conexión de databricks con el Azure Data Lake Gen 2

Existen distintos métodos para conectar databricks con el data lake, ahora nos conectaremos usando el key-vault y el servicio Microsoft Entra ID, conforme al siguiente tutorial:

<https://learn.microsoft.com/en-us/azure/databricks/connect/storage/tutorial-azure-storage#create-akv>.

El primer paso consiste en crear un servicio de Microsoft Entra ID, en el portal de azure, seleccionamos Entra ID, seleccionamos App registration, New registration y register (figura 40). Copiar y guardar los valores del client-ID y tenant-ID, los cuales vamos a guardar dentro del key-vault.

**\* Name**  
The user-facing display name for this application (this can be changed later).  
db-aw-app

**Supported account types**  
Who can use this application or access this API?  
 Accounts in this organizational directory only (Directorio predeterminado only - Single tenant)  
 Accounts in any organizational directory (Any Microsoft Entra ID tenant - Multitenant)  
 Accounts in any organizational directory (Any Microsoft Entra ID tenant - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)  
 Personal Microsoft accounts only

[Help me choose...](#)

By proceeding, you agree to the Microsoft Platform Policies [↗](#)

**Register**

**Figura 40.- Creación del Service principal en Microsoft Entra ID.**

Posteriormente, creamos el client secret para el service principal que ya tenemos. Seleccionamos manage, certificates & secrets, new client secret y añadimos una descripción como se muestra en la figura 41. Copiar y guardar el “value” del secret porque lo vamos a incluir dentro del key-vault.

db-aw-app | Certificates & secrets

Certificates & secrets

Add a client secret

Description	Expires
cl_secret	90 days (3 months)

New client secret

Description	Expires	Value

Add Cancel

**Figura 41.- Creación del secreto para el service principal.**

Ahora daremos acceso al service principal al data lake storage. Nos vamos a la cuenta del data lake, seleccionamos Access Control (IAM), Add, Add Role Assignment (figura 42).

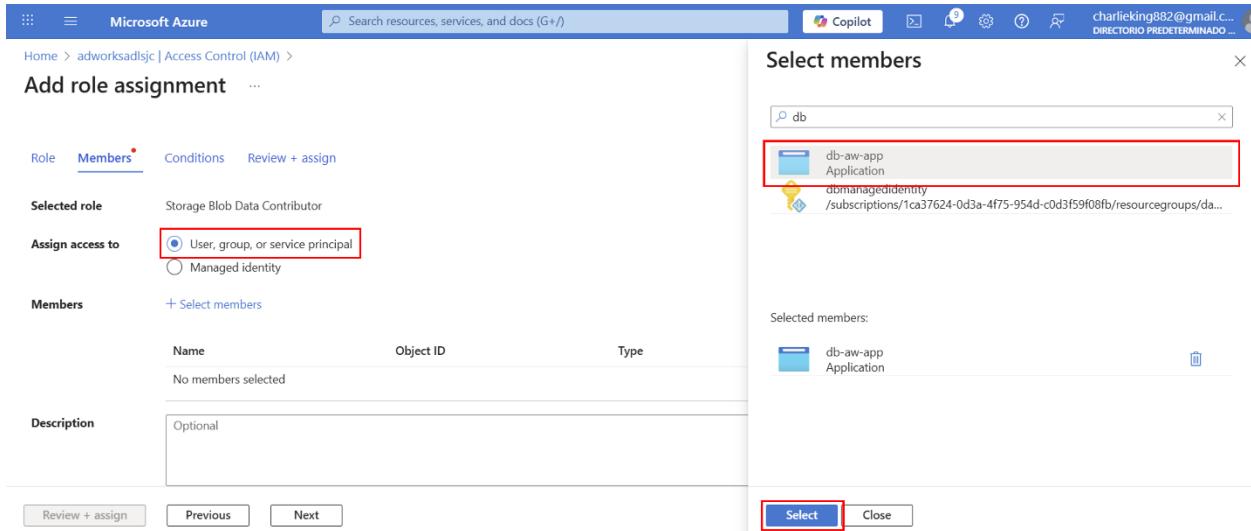
The screenshot shows the Microsoft Azure Storage account access control interface. On the left sidebar, 'Access Control (IAM)' is selected and highlighted with a red box. At the top center, there is a button labeled '+ Add < > Download role assignments' with a red box around it. Below this, another button labeled 'Add role assignment' is also highlighted with a red box. The main content area contains sections for 'Check access' and three buttons: 'Grant access to this resource', 'View access to this resource', and 'View deny assignments'. Each of these buttons has a 'View' button below it.

**Figura 42.-** Configuración para dar acceso al service principal al data lake.

Seleccionamos la opción storage blob data contributor y damos click en next. Seleccionamos user, group or service principal, select member, colocamos las iniciales del service principal, cuando aparezca le damos en select, y review + assign(figuras 43 y 44).

The screenshot shows the 'Add role assignment' wizard. At the top, the title 'Add role assignment' is displayed. Below it, there are tabs: 'Role' (selected), 'Members' (with a red asterisk), 'Conditions', and 'Review + assign'. A note states: 'A role definition is a collection of permissions. You can use the built-in roles or you can create your own custom roles.' Below this is a 'Copilot can help pick a role' button. Under 'Job function roles', there is a list of roles, with 'Storage Blob Data Contributor' selected. The details for this role are shown: 'Allows for read, write and delete access to Azure Storage blob containers and data'. At the bottom of the list, it says 'Showing 1 - 1 of 1 results.' At the very bottom of the page, there are buttons for 'Review + assign', 'Previous', and 'Next' (which is highlighted with a red box). There is also a 'Feedback' link at the bottom right.

**Figura 43.-** Asignación de rol para el service principal dentro del data lake.



**Figura 44.-** Selección del service principal con el rol de blob data contributor.

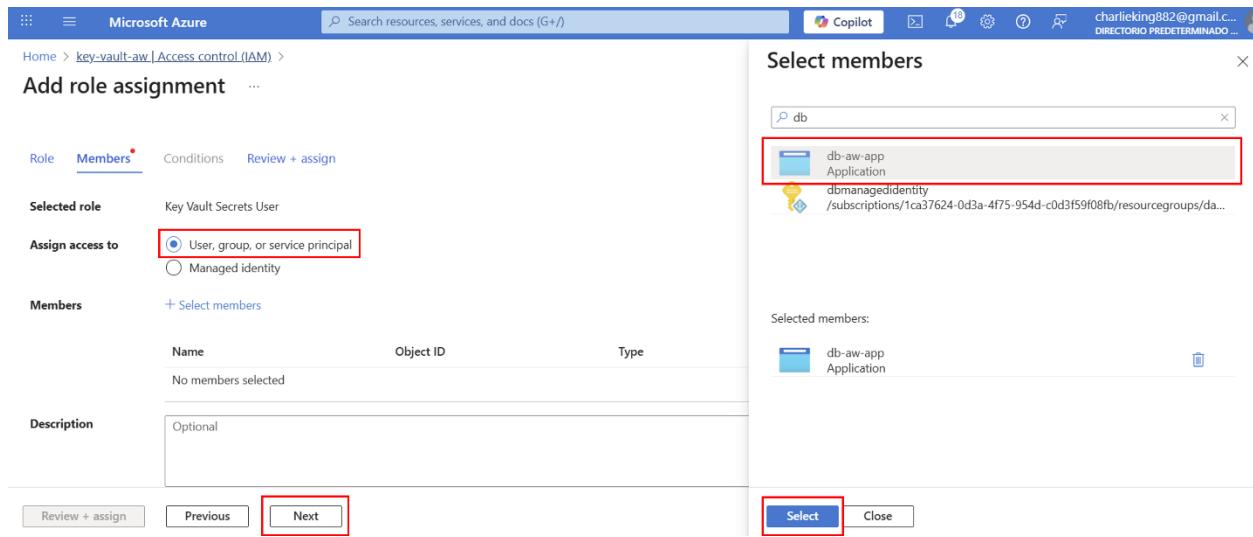
Ahora vamos al key-vault para crear los secretos para el service principal, crearemos uno para el client-id, tenant-id y secret-id.

Seleccionamos objects, secrets, generate/import, manual, colocamos el nombre del secreto, la clave y le damos en create (figura 45). Repetimos la operación para el tenant-id y el secret-id.

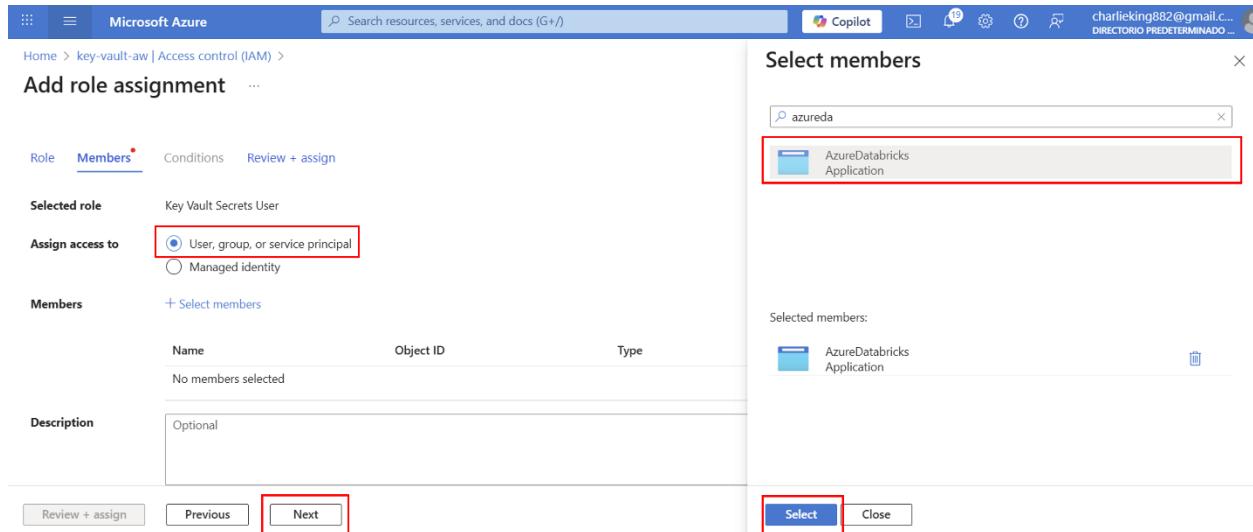
Upload options	Manual
Name *	db-client-id-app
Secret value *	*****
Content type (optional)	
Set activation date	<input type="checkbox"/>
Set expiration date	<input type="checkbox"/>
Enabled	<input checked="" type="radio"/> Yes <input type="radio"/> No
Tags	0 tags

**Figura 45.-** Secreto dentro del key-vault para el client-id del service principal.

Dentro del key-vault, damos acceso al service principal y a azure databricks para hacer uso de los secretos. Nos vamos a Access Control (IAM), Add Role Assignment, Key Vault Secrets User, Next, user, group, service principal y seleccionamos ya sea databricks o el service(figuras 46 y 47).



**Figura 46.- Asignación de key vault user al service principal.**

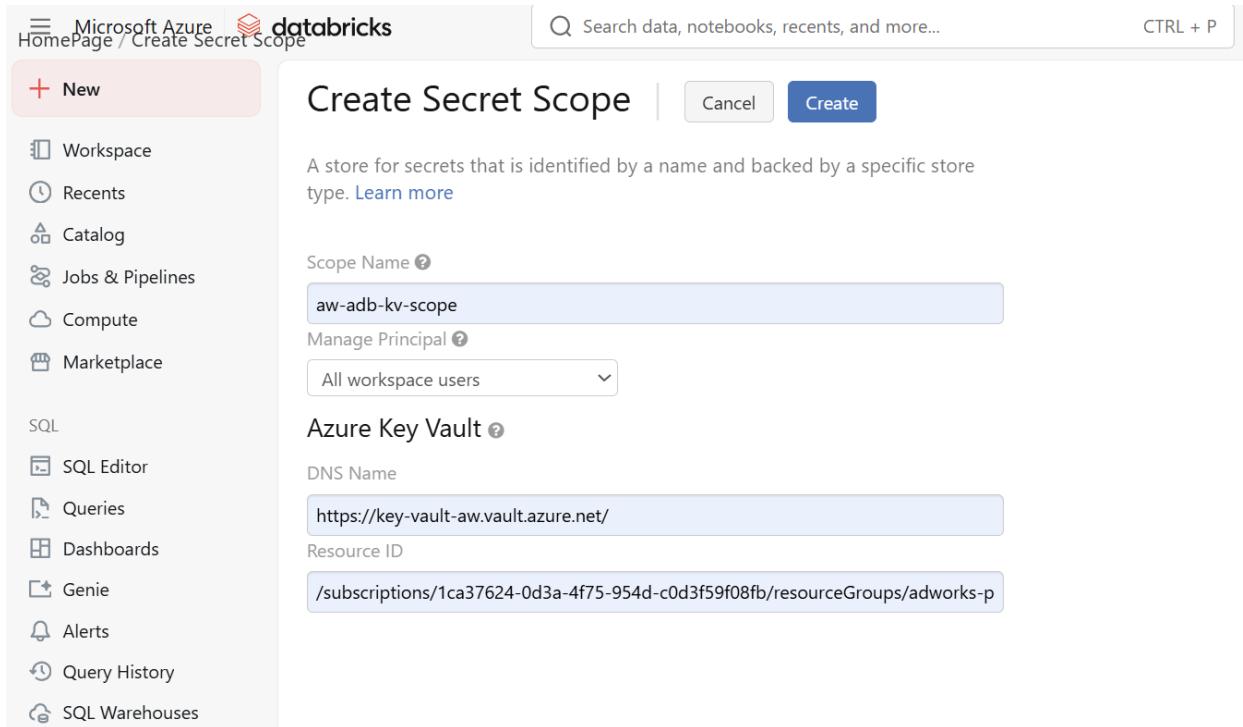


**Figura 47.- Asignación de key vault user a azure databricks.**

Procedemos a crear el secret scope dentro del workspace de databricks. Abrimos el siguiente link dependiendo de nuestro workspace:

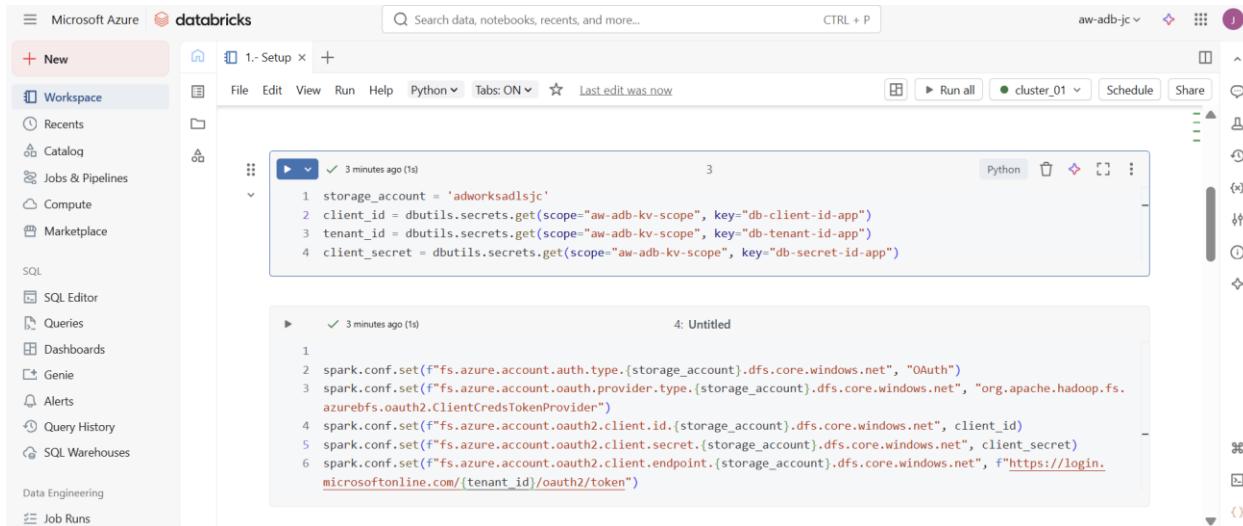
<https://adb- <instancia>.azuredatabricks.net/#secrets/createScope>

Asignamos un nombre al scope, seleccionamos all workspace users, colocamos el DNS name el cual corresponde al Vault URI, así como el Resource ID, éstos ultimo podemos encontrarlos dentro del key-vault en settings y properties, por último seleccionamos crear scope (figura 48).



**Figura 48.- Creación del secret scope dentro del workspace de databricks.**

Por último, abrimos un notebook en databricks y definimos las variables como el storage account, client\_id, tenant\_id y client\_secret. Definimos la configuración de spark con éstas variables que se obtienen de los secretos del key-vault (figura 49).



**Figura 49.- Configuración de databricks para conectarse con el data lake gen 2.**

Podemos verificar el contenido del contendor bronze con el comando que se muestra en la figura 50.

$\Delta$ c path	$\Delta$ c name	$\Delta$ <sup>2</sup> size	$\Delta$ <sup>2</sup> modificationTime
1 abfss://bronze@adworksadlsjc.dfs.core.windows.net/DimCustomer/	DimCustomer/	0	1770315844000
2 abfss://bronze@adworksadlsjc.dfs.core.windows.net/DimDate/	DimDate/	0	1770315716000
3 abfss://bronze@adworksadlsjc.dfs.core.windows.net/DimGeography/	DimGeography/	0	1770315716000
4 abfss://bronze@adworksadlsjc.dfs.core.windows.net/DimProduct/	DimProduct/	0	1770315772000
5 abfss://bronze@adworksadlsjc.dfs.core.windows.net/DimProductCategory/	DimProductCategory/	0	1770315729000
6 abfss://bronze@adworksadlsjc.dfs.core.windows.net/DimProductSubCategory/	DimProductSubCategory/	0	1770315717000
7 abfss://bronze@adworksadlsjc.dfs.core.windows.net/FactInternetSales/	FactInternetSales/	0	1770316093000

↓ 7 rows | 10.61s runtime      Refreshed now

**Figura 50.-** Listado de archivos dentro del contenedor bronce en el data lake.

Podemos leer el archivo en la subcarpeta DimProduct para verificar que se ha cargado correctamente como se muestra en la siguiente figura, confirmando que la conexión con el data lake ha sido satisfactoria.

$\Delta$ <sup>2</sup> ProductKey	$\Delta$ c ProductAlternateKey	$\Delta$ <sup>2</sup> ProductSubcategoryKey	$\Delta$ c WeightUnitMeasureCode	$\Delta$ c SizeUnitMeasureCode
1	347	BK-M82S-48	1	LB
2	348	BK-M82B-38	1	LB
3	349	BK-M82B-42	1	LB
4	350	BK-M82B-44	1	LB
5	351	BK-M82B-48	1	LB

↓ 5 rows | 1.71s runtime      Refreshed now

**Figura 51.-** Visualización del archivo DimCustomer en formato parquet almacenado en el data lake.

### Transformación de datos de la capa bronce a silver en databricks

Para no tener mucha complejidad en las transformaciones a realizar y a manera de ejemplo, vamos a crear un notebook para traernos los datos de la capa bronce a silver, convirtiéndolos de formato parquet a formato delta, dado que el formato delta presenta ciertas ventajas sobre el

El siguiente snippet nos proporciona una lista con los nombres de las tablas o carpetas en la capa bronce:

---

```
table_names = []
for i in dbutils.fs.ls("abfss://bronze@adworksadlsjc.dfs.core.windows.net/"):
    table_names.append(i.name.split('/')[0])
table_names
```

---

Como resultado nos arroja una lista con los nombres de cada tabla.

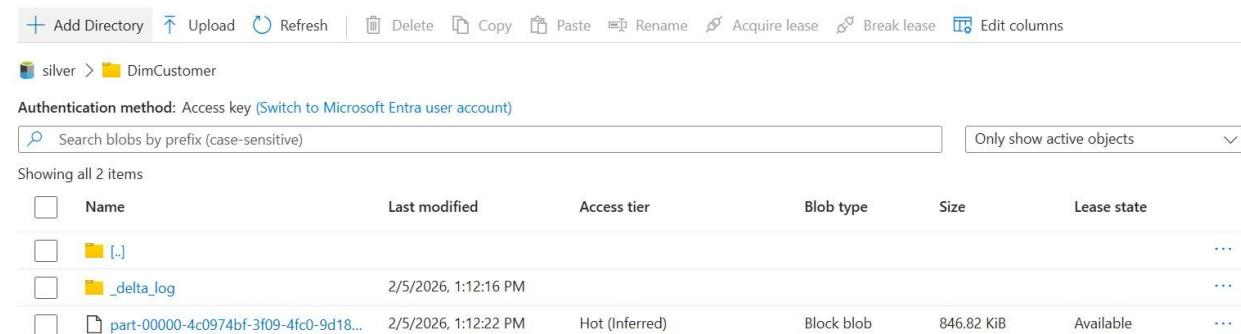
```
['DimCustomer',
'DimDate',
'DimGeography',
'DimProduct',
'DimProductCategory',
'DimProductSubCategory',
'FactInternetSales']
```

---

```
for table in table_names:
    path = "abfss://bronze@adworksadlsjc.dfs.core.windows.net/" + table + "/"
    df = spark.read.format("parquet").load(path)
    output_path = "abfss://silver@adworksadlsjc.dfs.core.windows.net/" + table + "/"
    # to datalake
    df.write.format("delta").mode("overwrite").save(output_path)
```

---

Como resultado, al correr el pipeline podemos observar que se han guardado las tablas en formato delta en la capa silver del datalake(figura 52).



The screenshot shows the Azure Storage Explorer interface. At the top, there are navigation buttons for 'Add Directory', 'Upload', 'Refresh', 'Delete', 'Copy', 'Paste', 'Rename', 'Acquire lease', 'Break lease', and 'Edit columns'. Below the header, it shows the path 'silver > DimCustomer'. A search bar says 'Search blobs by prefix (case-sensitive)' and a dropdown says 'Only show active objects'. A message at the bottom left says 'Authentication method: Access key (Switch to Microsoft Entra user account)'. The main area displays a table with the following data:

	Name	Last modified	Access tier	Blob type	Size	Lease state
<input type="checkbox"/>	...					...
<input type="checkbox"/>	_delta_log	2/5/2026, 1:12:16 PM				...
<input type="checkbox"/>	part-00000-4c0974bf-3f09-4fc0-9d18...	2/5/2026, 1:12:22 PM	Hot (Inferred)	Block blob	846.82 KiB	Available

**Figura 52.-** Resultado de la transformación de las fechas en las tabla de la base de datos.

El código anterior se guarda en un notebook llamado Bronze to silver, el cual vamos a ejecutar mediante una actividad de Databricks Notebook con Azure Data Factory, los notebooks se encuentran en el repositorio de github para su consulta.

## Transformación de datos de la capa silver a gold en databricks

La transformación de datos hacia la capa oro consiste en crear vistas con las tablas delta de la capa silver y aplicar una serie de joins con estas vistas para general el modelo de datos. Estas Dimension y Fact table van a ser de utilidad porque posteriormente se van a cargar en el Data Warehouse (Azure Synapse Analytics) y posteriormente serán leídas por la herramienta de visualización para la generación del tablero, estaremos usando en el proyecto Power-Bi. Mostraremos a manera de ejemplo el caso de las transformaciones en la DimProduct la cual mediante joins integra datos de las tablas ProductCategory y ProdutSubcategory.

Leemos y convertimos las tablas a vistas:

---

```
path = "abfss://silver@adworksadlsjc.dfs.core.windows.net/"  
df_customer = spark.read.format("delta").load(path + "DimCustomer/")  
df_date = spark.read.format("delta").load(path + "DimDate/")  
df_geography = spark.read.format("delta").load(path + "DimGeography/")  
df_product = spark.read.format("delta").load(path + "DimProduct/")  
df_productcategory = spark.read.format("delta").load(path + "DimProductCategory/")  
df_productsubcategory = spark.read.format("delta").load(path + "DimProductSubCategory/")  
df_sales = spark.read.format("delta").load(path + "FactInternetSales/")  
  
# Creating temporary views  
df_customer.createOrReplaceTempView("DimCustomer")  
df_date.createOrReplaceTempView("DimDate")  
df_geography.createOrReplaceTempView("DimGeography")  
df_product.createOrReplaceTempView("DimProduct")  
df_productcategory.createOrReplaceTempView("DimProductCategory")  
df_productsubcategory.createOrReplaceTempView("DimProductSubCategory")  
df_sales.createOrReplaceTempView("FactInternetSales")
```

---

Posteriormente utilizamos spark-sql para aplicar unos joins entre la DimProduct, DimProductCategory y DimProductSubcategory, como sigue:

---

```
df_dim_product = spark.sql("""
    SELECT
        p.ProductKey,
        p.ProductAlternateKey AS ProductItemCode,
        p.EnglishProductName AS ProductName,
        ps.EnglishProductSubcategoryName AS SubCategory,
        pc.EnglishProductCategoryName AS ProductCategory,
        p.Color AS ProductColor,
        p.Size AS ProductSize,
        p.ProductLine AS ProductLine,
        p.ModelName AS ProductModelName,
        p.EnglishDescription AS ProductDescription,
        COALESCE(p.Status, 'Outdated') AS ProductStatus
    FROM DimProduct as p
    LEFT JOIN DimProductSubcategory AS ps
        ON ps.ProductSubcategoryKey = p.ProductSubcategoryKey
    LEFT JOIN DimProductCategory AS pc
        ON pc.ProductCategoryKey = ps.ProductCategoryKey
    ORDER BY p.ProductKey ASC
""")  
  
path_gold_product = "abfss://gold@adworksadlsjc.dfs.core.windows.net/DimProduct/"
df_dim_product.write.format("delta").mode("overwrite").save(path_gold_product)
```

---

Como resultado tenemos 4 tablas en la capa gold, en la figura 53 podemos visualizar la creación de las tablas delta en la carpeta gold.

(1) Spark Jobs

df: pyspark.sql.connect.dataframe.DataFrame = [address\_id: integer, address\_line1: string ... 7 more fields]

Table +

	address_id	address_line1	address_line2	city	state_province	country_region
1	9	8713 Yosemite Ct.	null	Bothell	Washington	United States
2	11	1318 Lasalle Street	null	Bothell	Washington	United States
3	25	9178 Jumping St.	null	Dallas	Texas	United States
4	28	9228 Via Del Sol	null	Phoenix	Arizona	United States
5	32	26910 Indela Road	null	Montreal	Quebec	Canada

5 rows | 1m 44s runtime Refreshed now

**Figura 53.-** Tablas creadas para el data model a utilizar al construir el dasborad en Power-Bi.

## Integración de los databricks notebooks al pipeline de datos

Para poder añadir la ejecución de los notebooks al pipeline, previo es necesario agregar un linked service para poder conectarnos con Azure Databricks y a su vez contar con un token para guardarla en el key-vault.

Dentro del workspace databricks, nos vamos a la parte del nombre de usuario, seleccionamos settings, User, developer, Access tokens, Generate new token y creamos un nuevo token, el cual vamos a guardar dentro del key-vault (figura 54).

Comment	Creation	Expiration
adf	2026-01-12 16:17:44 CST	2026-04-12 17:17:44 CDT

**Figura 54.-** Token a guardar en el key-vault para la conexión de databricks con data factory.

Ahora nos vamos al key-vault, objetos, secretos, generate/import y pegamos el Access token recién creado. Nos vamos a la parte de manage, linked service, new, compute, Azure Databricks, next, from azure subscription, databricks workspace, Existing interactive cluster, access token,

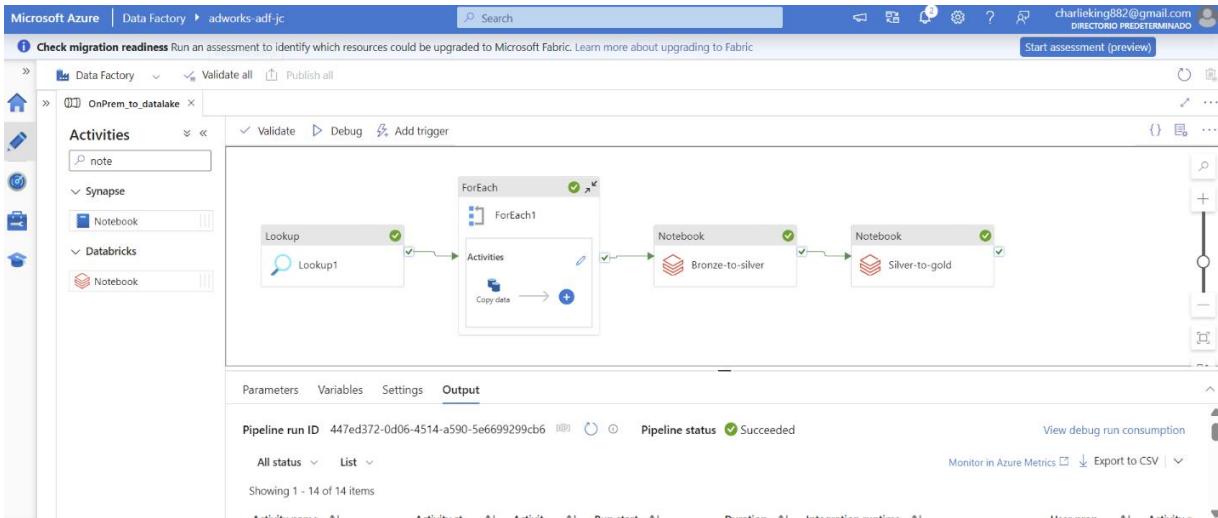
Azure key vault, secret name, seleccionamos el cluster y realizamos prueba de conexión (figura 55).

**Figura 55.-** Configuración del linked service para la conexión con databricks utilizando el Access token y key-vault.

Regresamos a la sección de Author, en la parte de activities, le damos en buscar, tecleamos notebooks y nos aparecen opciones de synapse y de databricks, arrastramos el de databricks hacia el canvas, posterior al ForEach. Seleccionamos el linked service e indicamos el notebook path (figura 56). Aplicamos el mismo procedimiento para el notebook de la capa silver a gold.

**Figura 56.-** Configuración del notebook de databricks para la transformación de datos de la capa bronce a silver.

En la figura 57 podemos visualizar el pipeline completo hasta la capa gold el cual corrió de manera satisfactoria. Las figuras 58 y 59 muestran los contenedores silver y gold dentro del data lake, confirmando que se han cargado de manera correcta las tablas en formato delta.



**Figura 57.-** Visualización del pipeline de datos tras añadir dos notebook activities de databricks para las capas silver y gold.

Name	Last modified	Access tier	Blob type	Size	Lease state
DimCustomer	2/5/2026, 1:12:16 PM				...
DimDate	2/5/2026, 1:12:48 PM				...
DimGeography	2/5/2026, 1:13:09 PM				...
DimProduct	2/5/2026, 1:13:19 PM				...
DimProductCategory	2/5/2026, 1:13:26 PM				...
DimProductSubCategory	2/5/2026, 1:13:29 PM				...
FactInternetSales	2/5/2026, 1:13:34 PM				...

**Figura 58.-** Contenedor silver dentro del data lake con las tablas delta.

Name	Last modified	Access tier	Blob type	Size	Lease state
DimCustomer	2/5/2026, 4:19:30 PM				...
DimDate	2/5/2026, 4:25:12 PM				...
DimProduct	2/5/2026, 4:27:56 PM				...
FactInternetSales	2/5/2026, 4:31:56 PM				...

**Figura 59.-** Contenedor gold dentro del data lake con las tablas delta.

## Carga de datos en el data warehouse (Synapse Analytics)

En Synapse Analytics vamos a crear un pipeline para crear las vistas con las tablas para el modelo de datos. Nos vamos a la sección de Integrate, creamos un nuevo pipeline y añadimos una actividad Get Metadata. En la sección de Settings seleccionamos un dataset tipo Binary que apunte hacia el contenedor en la capa Gold, añadimos un argumento y seleccionamos Child items

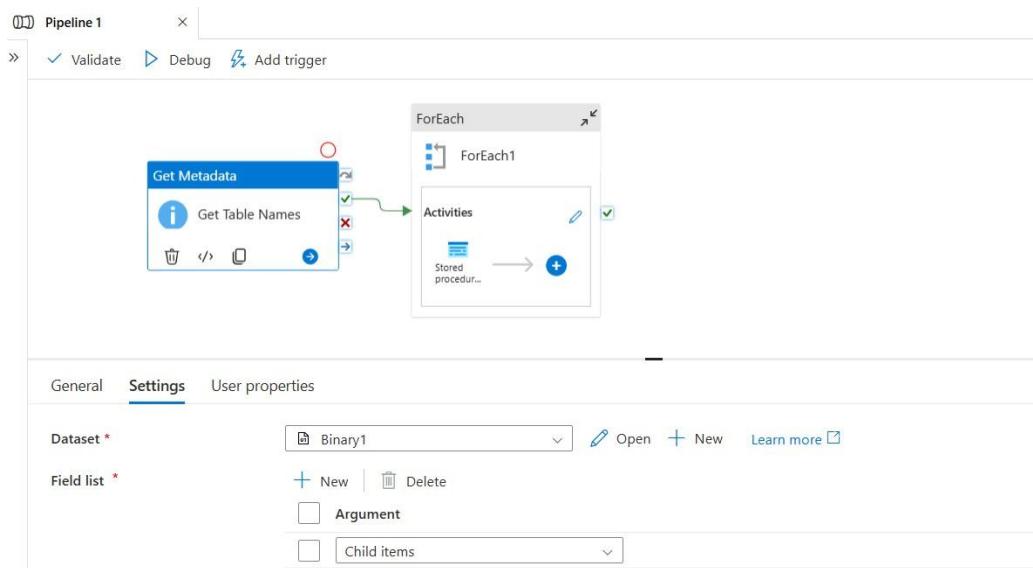


Figura 60.- Configuración de la Get Metadata Activity dentro de Synapse Analytics.

Añadimos un forEach el cual estará iterando a través de lista con los resultados de la activity Get Metadata. En la Sección de Setting, Items, añadimos el contenido en el expresión builder que se muestra en la figura 61.

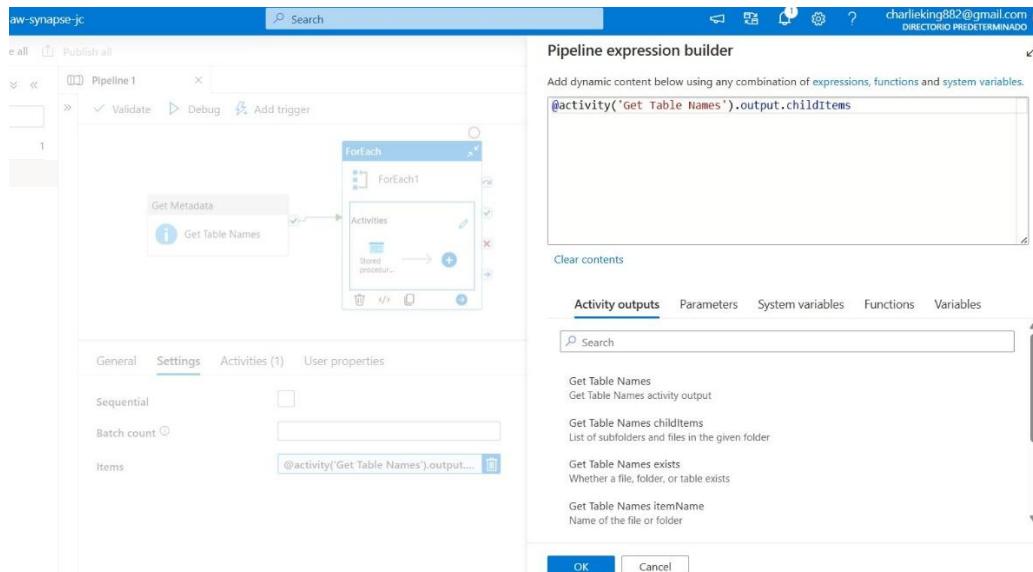


Figura 61.- Configuración de la actividad ForEach para iterar sobre los elementos resultado de actividad Get Metadata.

Dentro del ForEach, colocamos una actividad store procedure, el cual nos ejecutará para cada tabla el siguiente store procedure:

---

```
use gold_db
go

create or alter proc CreateSQLServerView_gold @ViewName nvarchar(100)
as
begin
    declare @statement varchar(MAX)
    set @statement = N'create or alter view ' + @ViewName + ' as
        select
        *
        from
            openrowset (
                bulk ''https://adworksadlsjc.dfs.core.windows.net/gold/' + @ViewName
+ '/'',
                format = ''DELTA''
            ) AS [result]'
    exec (@statement)
end
go
```

---

Una vez declarado el store procedure, podemos configurar la actividad. Creamos un linked service hacia la database dentro de synapse y seleccionamos el store procedure, como se muestra en la figura 62.

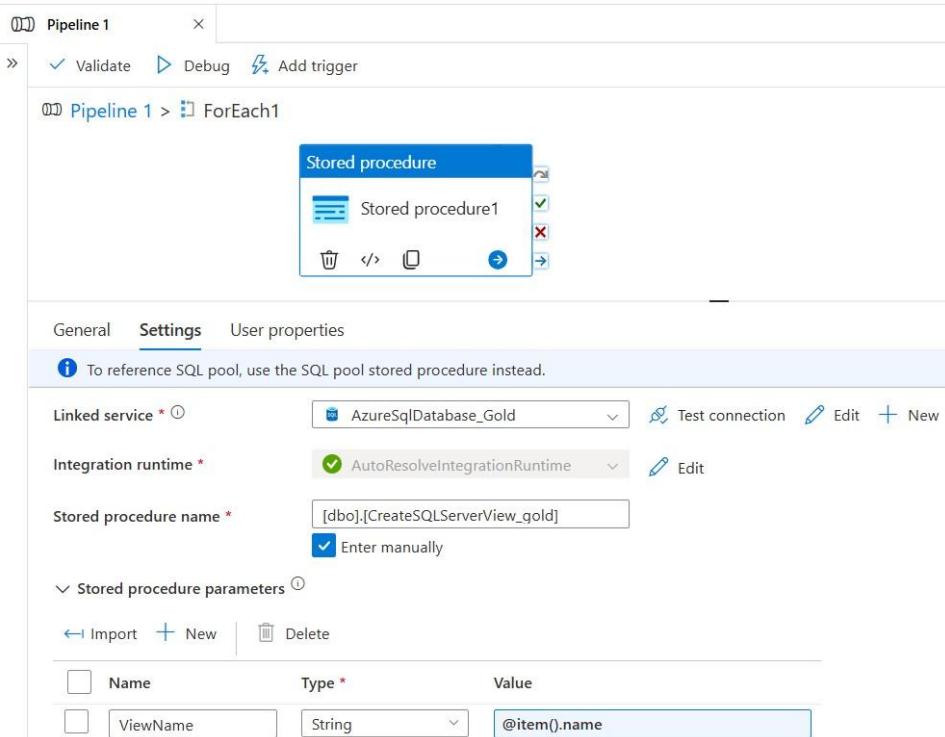


Figura 62.- Configuración de la actividad store procedure dentro para la creación de vistas.

Con la anterior procedemos a correr el pipeline y verificamos en la sección de Data, Workspace, gold\_db, views que se tienen creadas las vistas de manera correcta. En la figura 63 se muestra una query sobre la dbo.CustomerView, de manera satisfactoria.

The screenshot shows the Microsoft Synapse Studio interface. On the left, the 'Data' workspace is selected, showing a tree view of resources under 'gold\_db (SQL)'. Under 'Views', 'dbo.CustomerView' is listed. In the center, 'Pipeline 1' is active, with 'SQL script 1' containing the following query:

```

1 SELECT TOP (100) [CustomerKey]
2 ,[First_Name]
3 ,[Last_Name]
4 ,[Full_Name]
5 ,[Gender]
6 ,[DateFirstPurchase]
7 ,[Customer_City]
8 FROM [dbo].[CustomerView]

```

The 'Results' tab is selected, displaying the query results in a table format:

CustomerKey	First_Name	Last_Name	Full_Name	Gender	DateFirstPurch...	Customer_City
11000	Jon	Yang	Jon Yang	Male	2011-01-19	Rockhampton
11001	Eugene	Huang	Eugene Huang	Male	2011-01-15	Seaford
11002	Ruben	Torres	Ruben Torres	Male	2011-01-07	Hobart
11003	Christy	Zhu	Christy Zhu	Female	2010-12-29	North Ryde
11004	Elizabeth	Johnson	Elizabeth Johnson	Female	2011-01-23	Wollongong

At the bottom, a message indicates: '000004 Query executed successfully.'

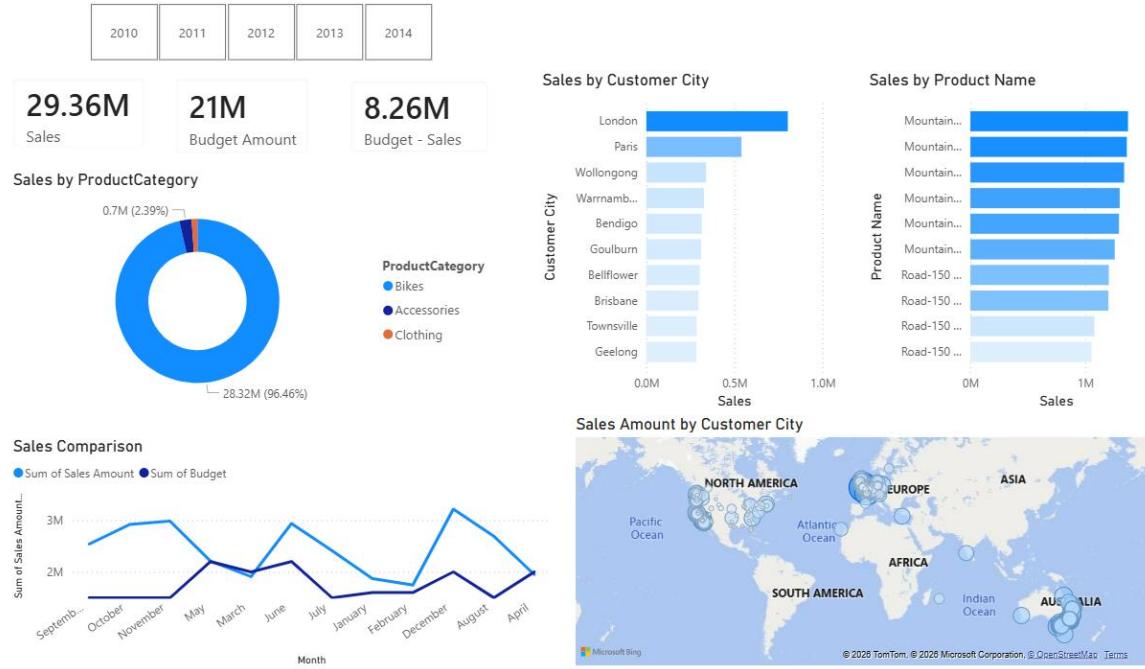
**Figura 63.-** Confirmación de la creación de vistas en la base dentro de Synapse y resultado de la query sobre la CustomerView.

Las vistas creadas nos servirán para la última etapa del proyecto, la cual consiste en la creación de un dashboard en Power-Bi.

## Visualización de datos en Power-Bi (Dashboard)

Por último, a manera de poder visualizar los datos y de verificar una carga y transformación correcta de los datos se construyó un dashboard sencillo que nos permita tener a la mano las métricas planteadas el principio, como como el total de ventas, la inversión o presupuesto gastado, ganancias (ventas – inversión), así como también poder visualizar las ventas por producto, ventas por categoría de producto, ventas por ciudad lo anterior con la posibilidad de filtrar por año. Sin embargo al no ser completamente un tutorial de Power-Bi, solo se muestra el tablero construido.

La figura 64 muestra un tablero de ventas en el que se pueden observar las métricas calculadas, así como ventas por producto, por categoría de producto y por ciudad de cliente. Realizaremos una carga posterior de 1 mes de datos para corroborar que existen cambios en las ventas totales al modificar los datos.



**Figura 64.-** Dashboard construido en Power-Bi para la visualización de las métricas.

### Adición de datos de 1 mes de ventas para una segunda corrida del pipeline

Vamos a realizar una segunda corrida del pipeline, para éste escenario vamos añadir 1 mes de ventas correspondiente al mes de febrero de 2014. En la primer corrida, las ventas llegaban hasta el día 28 de enero de 2014, al realizar una query sobre nuestra watermarktable de ventas podemos observar que es nuestro nuevo valor actualizado de OldWatermarktable (figura 65).

```
1 | select * from dbo.sales_watermarktable
```

TableName	WatermarkValue
1   data_source_table	2014-01-28 00:00:00.000

**Figura 65.-** Query que muestra la fecha más reciente de la FactInternetSales y que se utilizará como OldWatermarktable en la segunda corrida del pipeline.

Hemos añadido 1 mes más a la tabla dbo.FactInternetSales dentro de Sql-Server, teniendo ahora como fecha máxima el 28 de febrero de 2014, como se muestra en la query de la figura 66, ésta será la NewWatermarktable.

```

updateFactInt...nvitado (59)          SQLQuery2.sq...nvitado (83)*          SQLQuery1.s...itado (64)*
1   select max(OrderDate) as max_date
2   from dbo.FactInternetSales

```

Results

max_date
2014-02-28 00:00:00.000

**Figura 66.-** Adición de datos a la tabla FactInternetSales para evaluar la segunda corrida del pipeline.

Por último, realizamos una query para obtener el total de ventas y verificar que se ha incrementado de 26.36M, mostrado como métrica en el tablero de Power-Bi a 29.4M. Una vez que corramos el pipeline éste valor debe verse reflejado al actualizar el tablero.

```

updateFactInt...nvitado (59)          SQLQuery2.sq...nvitado (83)*          SQLQuery1.s...itado (64)*
1   select sum(SalesAmount)/1e6 as total
2   from dbo.FactInternetSales

```

No issues found

Results

total
29.4043719407

**Figura 67.-** Query dentro de SQL-Server para obtener el nuevo total de ventas al añadir el mes de febrero.

Tras aplicar un refresh en Power-Bi, podemos notar en la figura 68, que se ha actualizado el tablero con el nuevo total de ventas, confirmando que se ha corrido de manera correcta el pipeline.



**Figura 68.-** Actualización del tablero en Power-Bi, con el último mes, se observa el nuevo valor de ventas a 29.4 M.

Sí consultamos el data lake, en el contenedor bronce, en la FactInternetSales podemos notar que se encuentran los dos archivos parquet, por cada corrida, el segundo con los datos delta del mes de febrero.

Name	Last modified	Access tier	Blob type	Size	Lease state
[..]					...
291d21a1-32a8-41dc-a9de-721babb731b6.parquet	2/5/2026, 8:30:05 PM	Hot (Inferred)	Block blob	54.75 KiB	Available
3b78db26-c288-4a46-94ad-4bbab15.parquet	2/5/2026, 4:38:35 PM	Hot (Inferred)	Block blob	1.52 MiB	Available

**Figura 69.-** Datos cargados dentro del datalake capa bronce tras la segunda corrida del pipeline.

Al leer los datos desde databricks, si solo cargamos el segundo archivo, notamos que efectivamente solo se generó el archivo con los datos del delta, como se observa en la figura 70.

```

1 df = spark.read.format('parquet').load('abfss://bronze@adworksadlsjc.dfs.core.windows.net/FactInternetSales/
291d21a1-32a8-41dc-a9de-721babb731b6')
2 display(df)

```

(2) Spark Jobs

df: pyspark.sql.connect.DataFrame = [ProductKey: integer, OrderDateKey: integer ... 24 more fields]

CarrierTrackingNumber	CustomerPONumber	OrderDate	DueDate	ShipDate
1966	null	2014-02-28T00:00:00.000+00...	2014-03-09T00:00:00.000+00...	2014-03-04T00:00:00.000+00...
1967	null	2014-02-28T00:00:00.000+00...	2014-03-09T00:00:00.000+00...	2014-03-04T00:00:00.000+00...
1968	null	2014-02-28T00:00:00.000+00...	2014-03-09T00:00:00.000+00...	2014-03-04T00:00:00.000+00...
1969	null	2014-02-28T00:00:00.000+00...	2014-03-09T00:00:00.000+00...	2014-03-04T00:00:00.000+00...
1970	null	2014-02-28T00:00:00.000+00...	2014-03-09T00:00:00.000+00...	2014-03-04T00:00:00.000+00...

1,970 rows | 5.59s runtime      Refreshed 2 minutes ago

**Figura 70.-** Lectura del archivo con el delta de datos añadidos en la FactInternetSales.

## Conclusiones:

- Se desarrolló un proyecto end-to-end desde la ingesta, transformación, carga y visualización de datos en El ecosistema de Azure para una base de datos E-commerce.
- La ingesta de datos se realizan desde una base de datos On-prem (sql server) y considerando Cargas Full así como incrementales con un enfoque metadata-driven en Azure data factory. Se realizó la conexión a Sql-Server mediante el SelfHosted Integration Runtime de Azure. Se utilizaron dos pipeline(Parent y Child) pipeline para poder aplicar carga Full en incremental, por lo que fue necesario definir parámetros a nivel pipeline(pipeline().parameter).
- El proyecto está basado en la arquitectura medallón, utilizando azure data lake storage gen 2 como almacenamiento.
- Se realizaron las transformaciones como la conversión de parquet a formato delta, limpieza y agregaciones en un espacio de trabajo de databricks, con notebooks orquestados desde azure data factory.
- Se utilizó el servicio de Synapse Analytics como data warehouse, donde se construyó un pipeline para la ejecución de un store procedure dinámico para la generación de vistas para el Modelo de datos.
- Se desarrolló un tablero en Power-bi para la visualización y análisis de datos de ventas.
- Se simuló una carga adicional de datos para corroborar la operación del pipeline de datos dando resultados satisfactorios al comparar los tableros generados.

Espero el proyecto sea de utilidad y lo disfruten tanto como yo. Los archivos para replicarlo se encuentran en el repositorio.

Happy coding, jc.