<https://quickstarts.snowflake.com/guide/getting_started_with_azure_data_factory_and_snowflake/index.html?index=..%2F..index#0>

2. Use Case

As a retail analyst, imagine effortlessly unraveling your company's sales performance with Azure Data Factory and Snowflake. Your sales data, stored in Azure SQL, seamlessly flows into Snowflake, where you can conduct comprehensive analyses with unprecedented speed and scalability. A combination of GUI and scripting-based interfaces allows anyone, no matter their skillset or preferences, to easily orchestrate data pipelines. With pushdown compute capabilities, choose between SQL and Python to effortlessly sift through the data within Snowflake, revealing invaluable insights.

3. Set Up Snowflake Environment

The first thing to do is create a database and warehouse in your Snowflake environment. Run the below code in a Snowflake worksheet. You are using the AccountAdmin role here for demo purposes, but in production you will likely use a different role.

-- Create a new database (if not already created)

CREATE DATABASE IF NOT EXISTS ADFdemo;

USE DATABASE ADFdemo;

-- Create a new virtual warehouse (if not already created)

CREATE WAREHOUSE IF NOT EXISTS ADFdemo WITH WAREHOUSE\_SIZE='X-SMALL';

CREATE SCHEMA IF NOT EXISTS Raw;

CREATE SCHEMA IF NOT EXISTS Analytics;

CREATE OR REPLACE TABLE Raw.Customer(

CustomerID int,

NameStyle STRING ,

Title STRING NULL,

FirstName STRING ,

MiddleName STRING NULL,

LastName STRING ,

Suffix STRING NULL,

CompanyName STRING NULL,

SalesPerson STRING NULL,

EmailAddress STRING NULL,

Phone STRING NULL,

PasswordHash STRING ,

PasswordSalt STRING ,

rowguid STRING ,

ModifiedDate datetime

);

CREATE OR REPLACE TABLE Raw.Product(

ProductID int ,

Name STRING ,

ProductNumber STRING ,

Color STRING NULL,

StandardCost decimal ,

ListPrice decimal ,

Size STRING NULL,

Weight decimal(8, 2) NULL,

ProductCategoryID int NULL,

ProductModelID int NULL,

SellStartDate datetime ,

SellEndDate datetime NULL,

DiscontinuedDate datetime NULL,

ThumbNailPhoto STRING NULL,

ThumbnailPhotoFileName STRING NULL,

rowguid STRING ,

ModifiedDate datetime

);

CREATE OR REPLACE TABLE Raw.ProductCategory(

ProductCategoryID int ,

ParentProductCategoryID int NULL,

Name STRING ,

rowguid STRING ,

ModifiedDate datetime

);

CREATE OR REPLACE TABLE Raw.SalesOrder(

SalesOrderID int ,

SalesOrderDetailID int ,

OrderQty int ,

ProductID int ,

UnitPrice DECIMAL ,

UnitPriceDiscount DECIMAL ,

LineTotal DECIMAL,

rowguid STRING ,

ModifiedDate datetime ,

RevisionNumber int ,

OrderDate datetime ,

DueDate datetime ,

ShipDate datetime NULL,

Status int ,

SalesOrderNumber STRING,

PurchaseOrderNumber STRING NULL,

AccountNumber STRING NULL,

CustomerID int ,

ShipToAddressID int NULL,

BillToAddressID int NULL,

ShipMethod STRING ,

CreditCardApprovalCode STRING NULL,

SubTotal DECIMAL ,

TaxAmt DECIMAL ,

Freight DECIMAL ,

TotalDue DECIMAL,

Comment STRING NULL

);

CREATE TABLE IF NOT EXISTS Analytics.CustomerOrders (

SalesOrderID int,

orderdate datetime,

DueDate datetime,

ShipDate datetime,

PurchaseOrderNumber STRING,

AccountNumber STRING,

TotalDue DECIMAL,

rowguid STRING,

Title STRING,

FirstName STRING,

LastName STRING,

CompanyName STRING,

EmailAddress STRING,

LineTotal DECIMAL,

Productname STRING,

ProductNumber STRING,

ProductCategoryName STRING

);

CREATE OR REPLACE PROCEDURE Analytics.InsertCustomerOrders()

RETURNS VARCHAR

LANGUAGE SQL

AS

$$

BEGIN

INSERT INTO Analytics.CustomerOrders

SELECT

so.SalesOrderID,

so.orderdate,

so.DueDate,

so.ShipDate,

so.PurchaseOrderNumber,

so.AccountNumber,

so.TotalDue,

so.rowguid,

c.Title,

c.FirstName,

c.LastName,

c.CompanyName,

c.EmailAddress,

so.LineTotal,

p.name,

p.ProductNumber,

pc.Name

FROM raw.SalesOrder so

join raw.Customer c on c.CustomerID=so.CustomerID

join raw.Product p on p.ProductID=so.ProductID

join raw.ProductCategory pc on pc.ProductCategoryID=p.ProductCategoryID

LEFT JOIN Analytics.CustomerOrders co on co.SalesOrderID=so.SalesOrderID

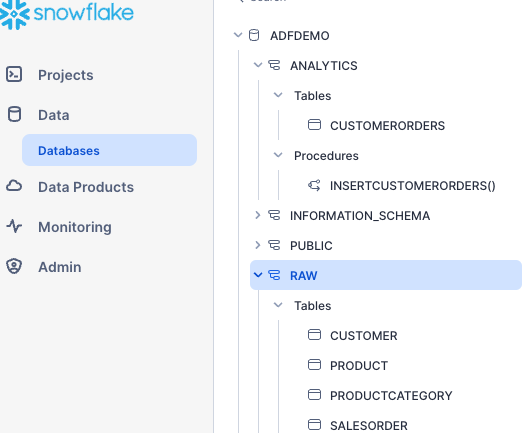
WHERE co.rowguid is null;

RETURN 'Success';

END;

$$;

This SQL has set up the raw tables for us to land data into, as well as some stored procedures you'll use later in the quickstart.

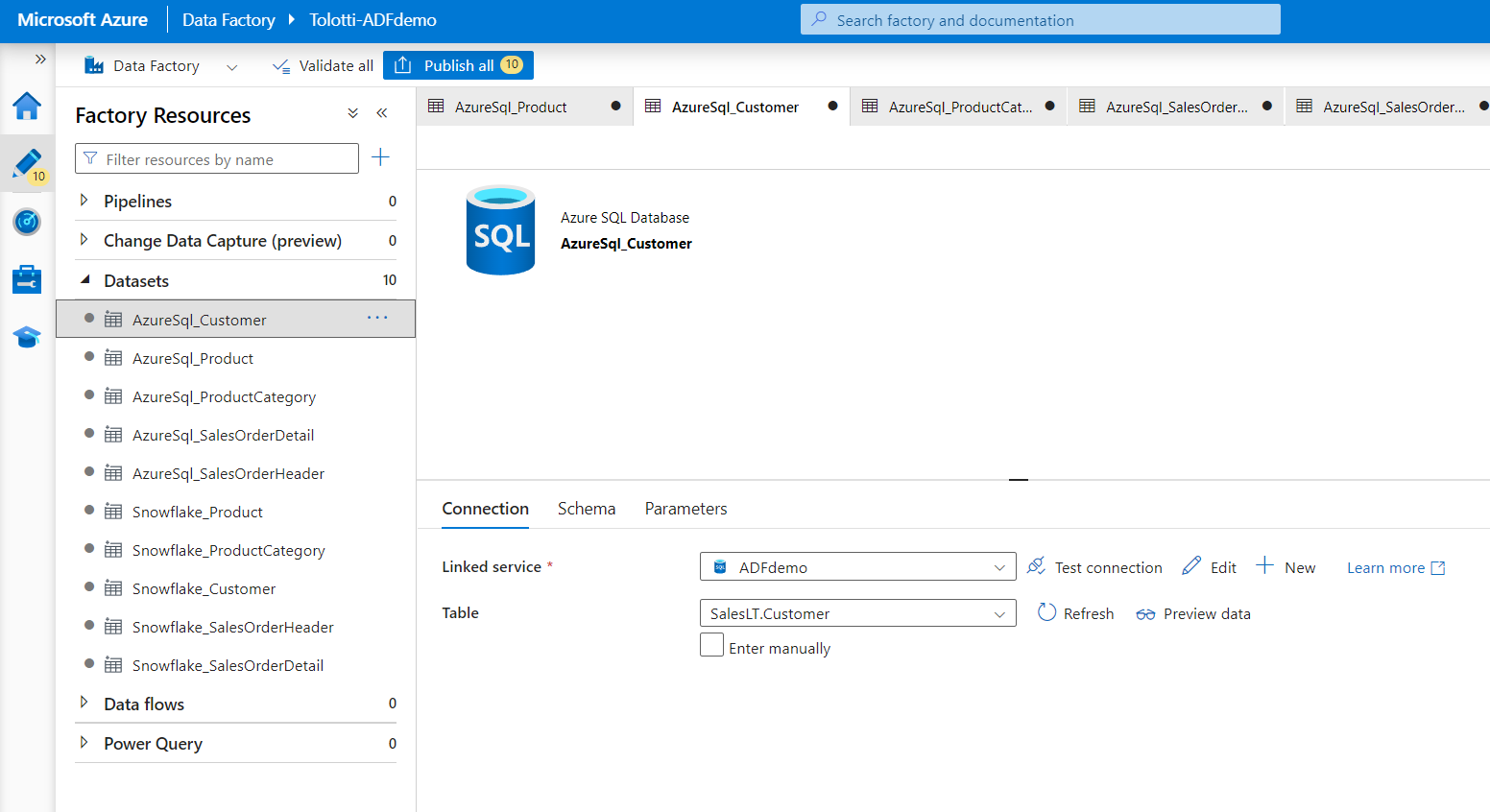
The result should look like this screenshot, with tables, stored procedure, and Schemas: 

4. Set Up Azure Environment

Set Up Azure SQL with AdventureWorks Data

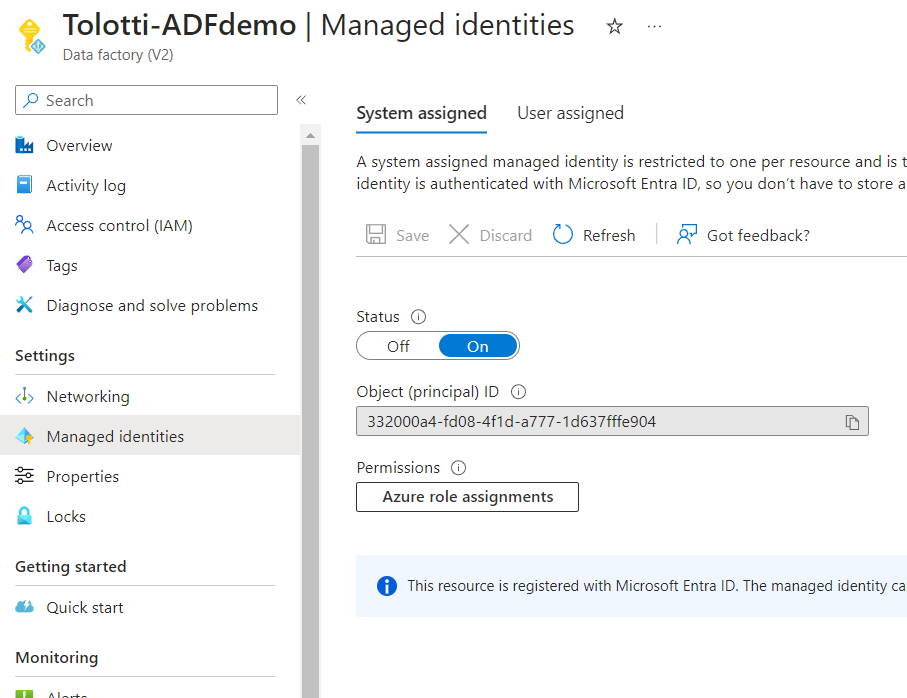
Now you need to set up your Azure SQL instance. When you create it you can include a sample Adventure Works dataset, this will represent your transactional sales database.

1. Select create > Search Martkerplace for Azure SQL > Click Create > Under SQL Databases select Single Database and click Create
   * Database Name: ADFdemo
2. Server (Select create new): Give a unique server name, enable SQL authentication for this quickstart and enter an admin login and password.
   * Go to Networking, for Connectivity method select Public Endpoint, select yes for both Allow Azure Services and Add Current client IP.
   * No changes in Networking or Security
3. Under Additional Settings, Use Existing Data: Select Sample
   * Select Create



Set Up ADF Workspace

Head back to the Azure Marketplace and search for Data Factory.  
Select Create > Enter a Name, your preferred region, and V2 for the Instance Details.  
Create with the default settings.



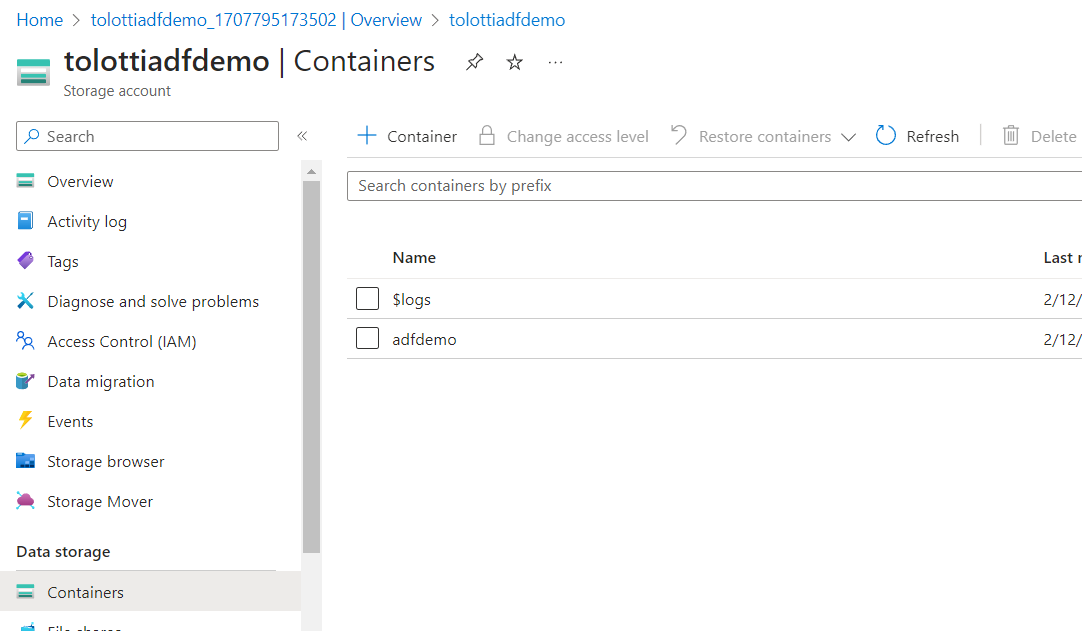
Set up Container for Staging

Behind the scenes, ADF will use Blob Storage to stage data as it is moved. So you need to have a Blob Storage container available.

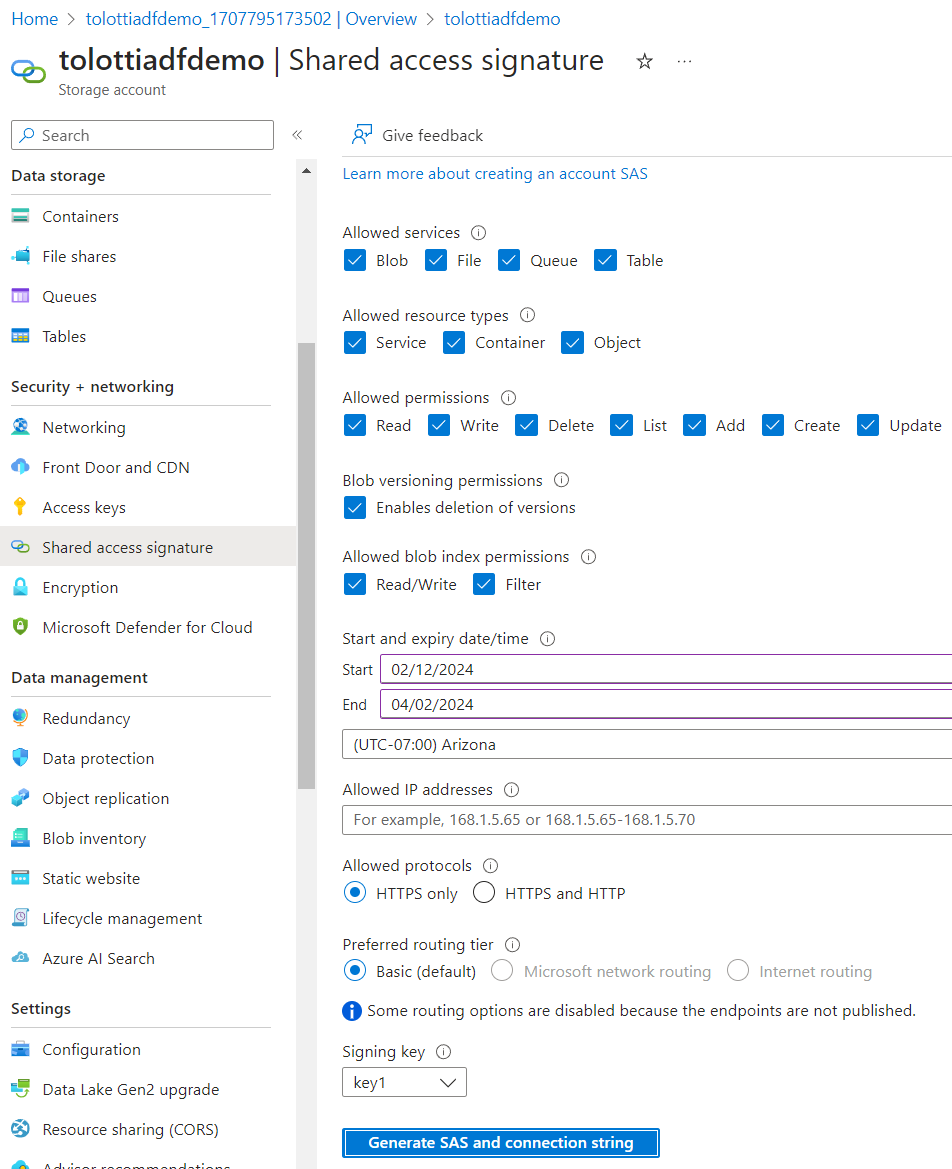
1. From the Marketplace, search for Storage Account and create.
   * Give the Account a unique name
   * Keep all the default settings the same and Create

(Note: you are leaving the Networking open to public access for this Quickstart, be sure to follow your organization's security best practices for non-demo projects)

1. Once created select Go to Resource
   * On the left select Containers
   * Add a Container named adfdemo



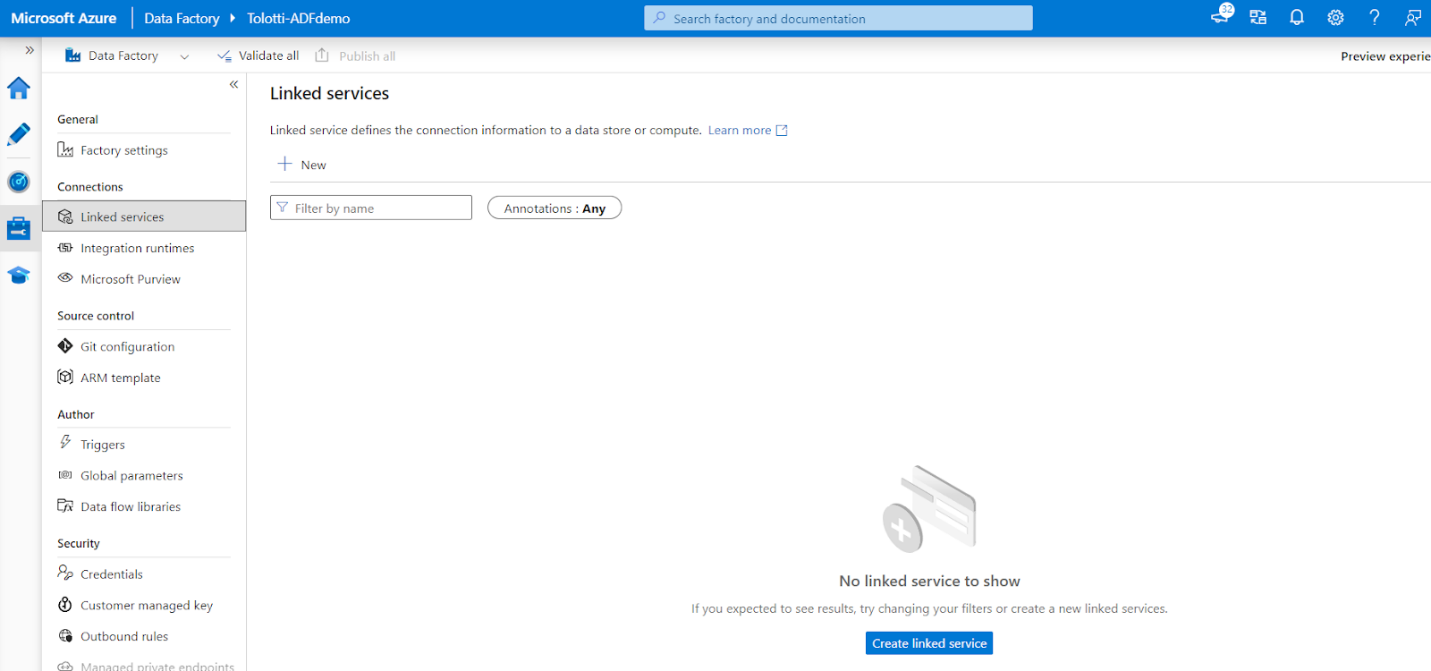
1. You'll need to generate a SAS token for ADF to access the container.
   * On the left panel select Shared Access Signature
   * Ensure all resource types are available
   * Make the expiration End Date long enough to finish the Quickstart
   * Generate SAS and connection strong, then copy out the SAS token (Note: Be sure to save it or leave this tab open, you won't be able to get back to the token)



5. Configure the Linked Services

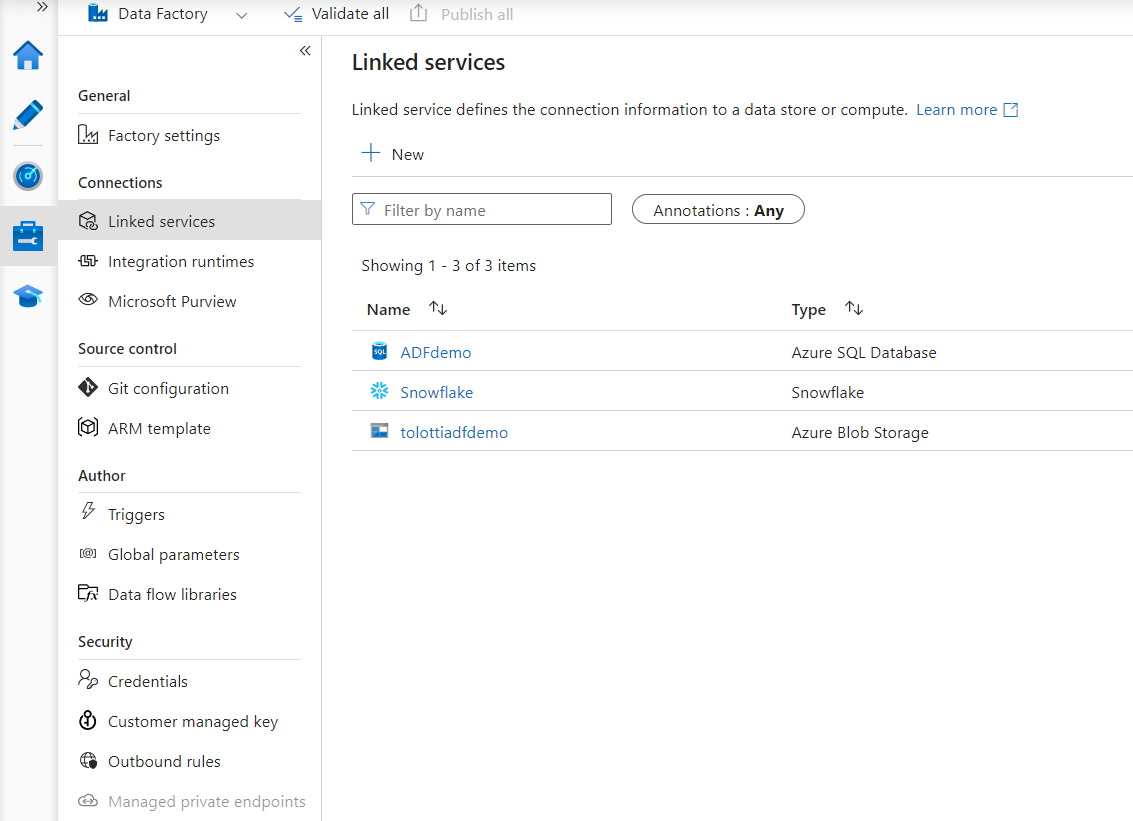
Now you want to get into the ADF Studio. Go to the ADF resource you created and click Launch Studio from the Overview tab. This will open up the Data Factory Studio.

On the Left panel, click on the Manage tab, and then linked services.



Linked Services act as the connection strings to any data sources or destinations you want to interact with. In this case you want to set up services for Azure SQL, Snowflake, and Blob Storage.

1. Click Create Linked Services > Select Azure SQL Database > Continue
   * Add a Name, select all the subscription information for your Azure SQL database
   * Enter your SQL authentication username and password setup previously
   * Test connection and create
2. Click Create Linked Services > Select Azure Blob Storage> Continue
   * Set Authentication type to SAS URI and enter the information created in previous steps
   * The URL should be just the base URL: such as AccountName.blob.core.windows.net
3. Click Create Linked Services > Select Snowflake > Continue
   * Fill out the details for your account, note the format of the account name
   * The database and warehouse are both ADFDEMO



6. Create Source & Sink Datasets

Next you need to create datasets that map the data inside the linked services. In this case you'll need to set up 5 source datasets for the Azure SQL tables, and 5 sink datasets for the Snowflake tables.

Navigate to Author on the side menu, under the three dot menu on Datasets select New Dataset.

* Select Azure SQL database, name it AzureSQL\_Product and enter your linked server to azure SQL
* Then set the table name to SalesLT.Product

Repeat this process for the following:

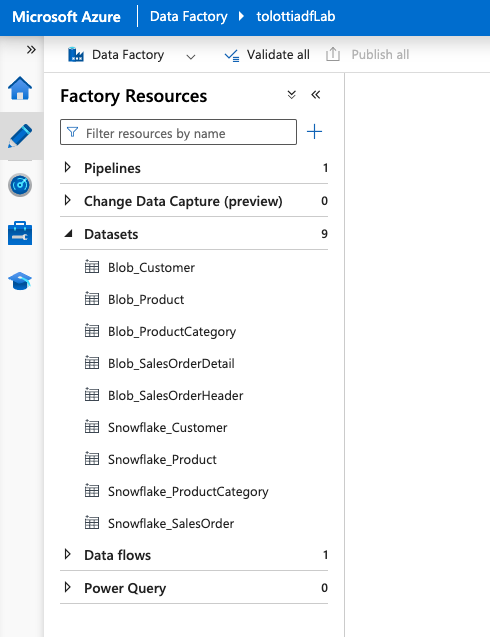
* AzureSQL\_ProductCategory
* AzureSQL\_Customer
* AzureSQL\_SalesOrderHeader
* AzureSQL\_SalesOrderDetail

Now you want to set up the Sink datasets in Snowflake.

* Create a new dataset, but this time select Snowflake
* Name the dataset Snowflake\_Product
* Set the linked service to the Snowflake one you created. And set the tablename to Raw.Product

Again repeat this process for the following:

* Snowflake\_ProductCategory
* Snowflake\_Customer
* Snowflake\_SalesOrder



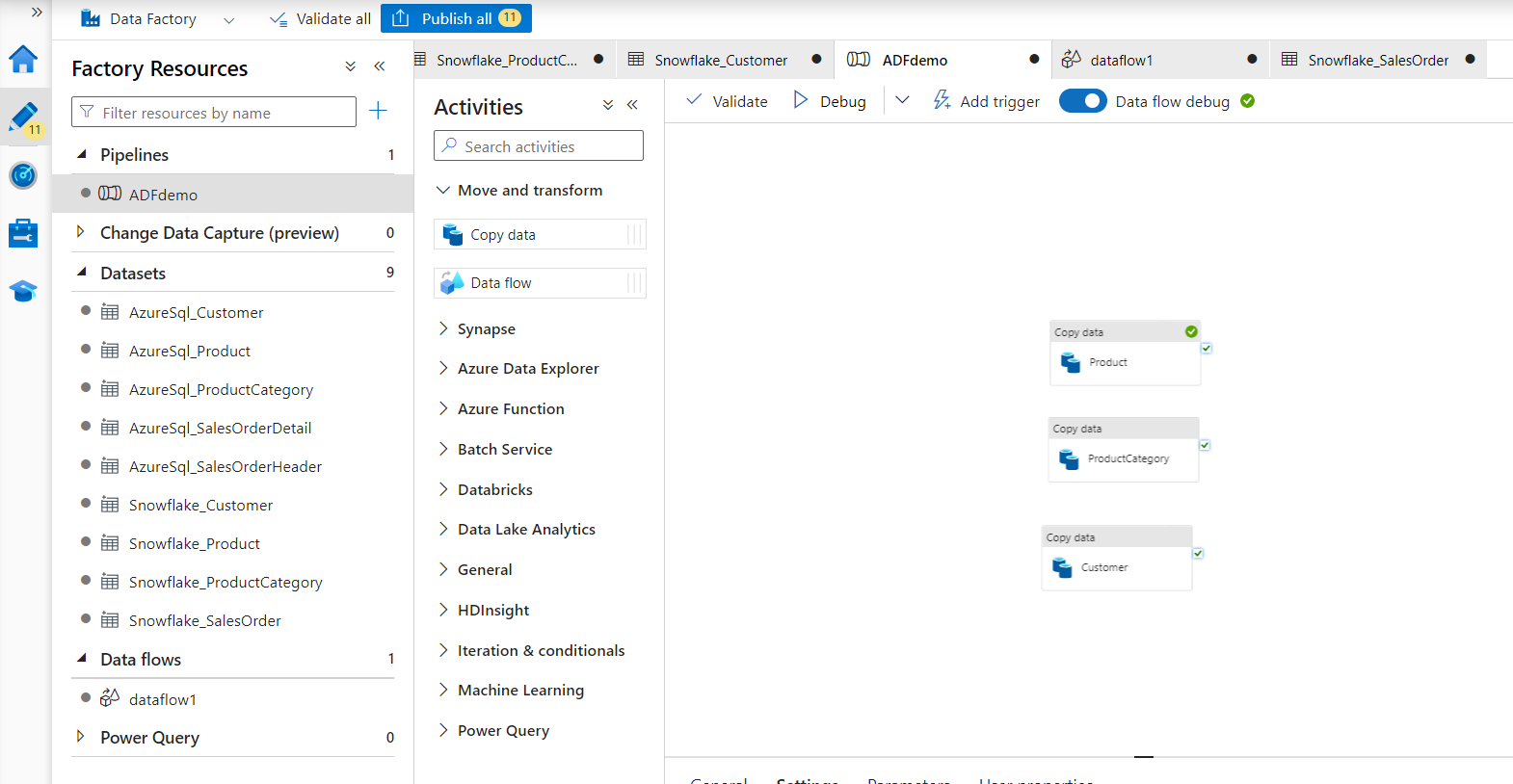
7. Create the Pipeline

Now it's time to set up the pipeline that will control the flow of the data.

Copy Data

You'll start with a Copy Data Activity to directly copy the data from source to sink, with no transformation. Transformations can be done later in Snowflake, which is an ELT process.

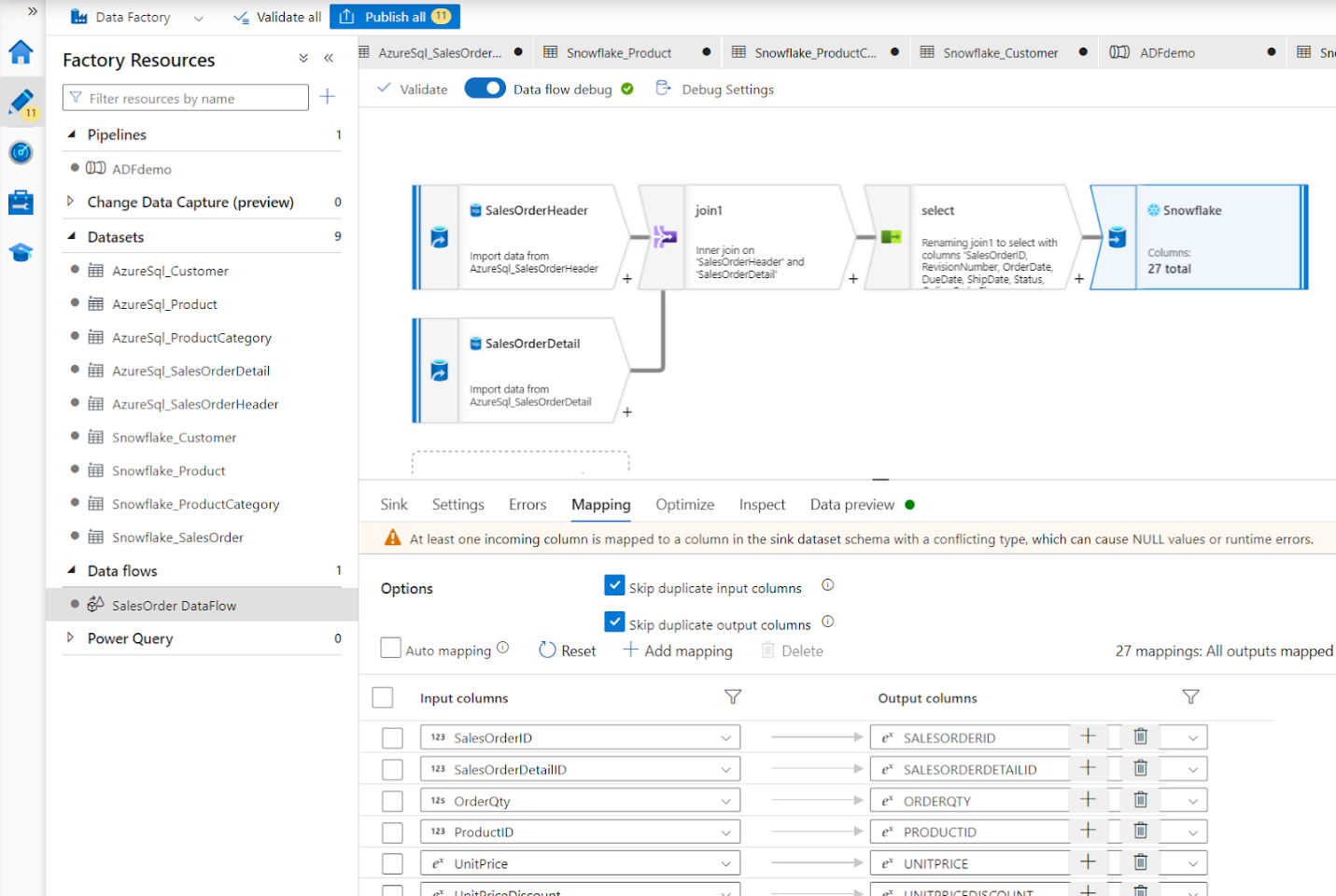
1. Under Move and Transform, select a CopyData activity.
   * Name the CopyData Activity as Product.
   * Under the Source tab, select AzureSql\_Product.
   * Under the Sink tab, select Snowflake\_Product. In the pre-copy script enter "Truncate table Raw.Product;". This SQL will clear what's in the raw staging in Snowflake each time data is loaded.
   * Under the Settings tab, toggle Enable Staging and set the Staging account to the Blob Storage setup previously.
2. You can copy and paste the Activity to create two more CopyData activities.
   * For the first, update the name to ProductCategory, and update the source and sink to the product category tables. Update the Sink truncate command to Raw.ProductCategory.
   * For the second, do the same with Customer.



Data Flow

Now add a Data Flow Activity. Data Flows are a traditional ETL task. You can perform transformations on the data in the middle of the data movement. If you are familiar with SSIS, data flows have similar functionality. In this case you are going to merge the Sales Order Header and Detail into a single table in Snowflake.

* Name the activity SalesOrders.
* Under Settings select New Data flow.
* Under the Properties set the Name to SalesOrder DataFlow.
* At the top, select Data Flow Debug, and OK. This will create a Spark cluster for debugging.
* Add a Source. Name the output stream SalesOrderHeader and Select the dataset as AzureSQL\_SalesOrderHeader. Under projection click Import projection to get the field metadata.
* Add a second source. Name the output stream SalesOrderDetail and Select the dataset as AzureSQL\_SalesOrderDetail. Under projection click Import projection to get the field names.
* Select the small + to the right of the SalesOrderHeader task, and add Join.
* Set the left stream to SalesOrderHeader, and the right to SalesOrderDetail.
* Under join conditions set both the left and right to SalesOrderID.
* Click the + and add in a select task. Since there are some columns in both tables with the same name, the skip duplicate input columns will just use the SalesOrderHeader columns.
* Select + and add a sink. Name it Snowflake, set the dataset to Snowflake\_SalesOrder. Under mapping uncheck Auto Mapping so it uses the fields from the select step. You may need to open the Dataset and Import schema to refresh the metadata if the mapping does not look correct.



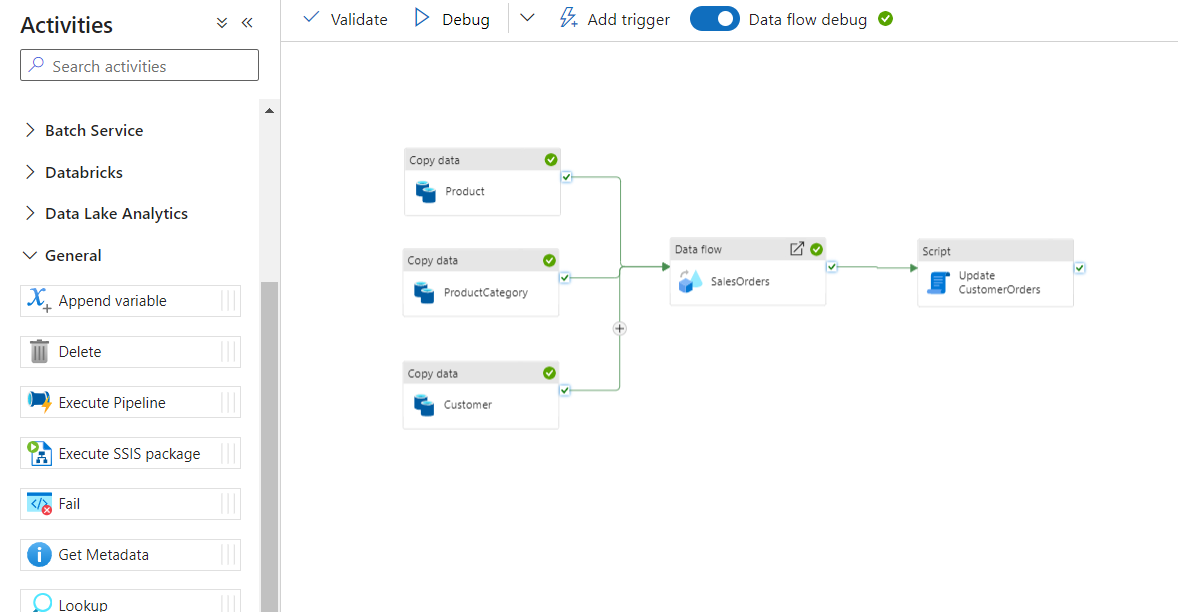
Script Tasks

In order to push compute to Snowflake, you can add in script activities to call queries or stored procedures. In this step you can transform the data however you need to, using SQL or python via Snowpark.

Back in the pipeline, under the General Activities, add in a Script Activity.

Name the activity Update CustomerOrders, under Settings add in a query for CALL Analytics.InsertCustomerOrders();

Now orchestrate the order of execution by connecting activities with the Success connector.



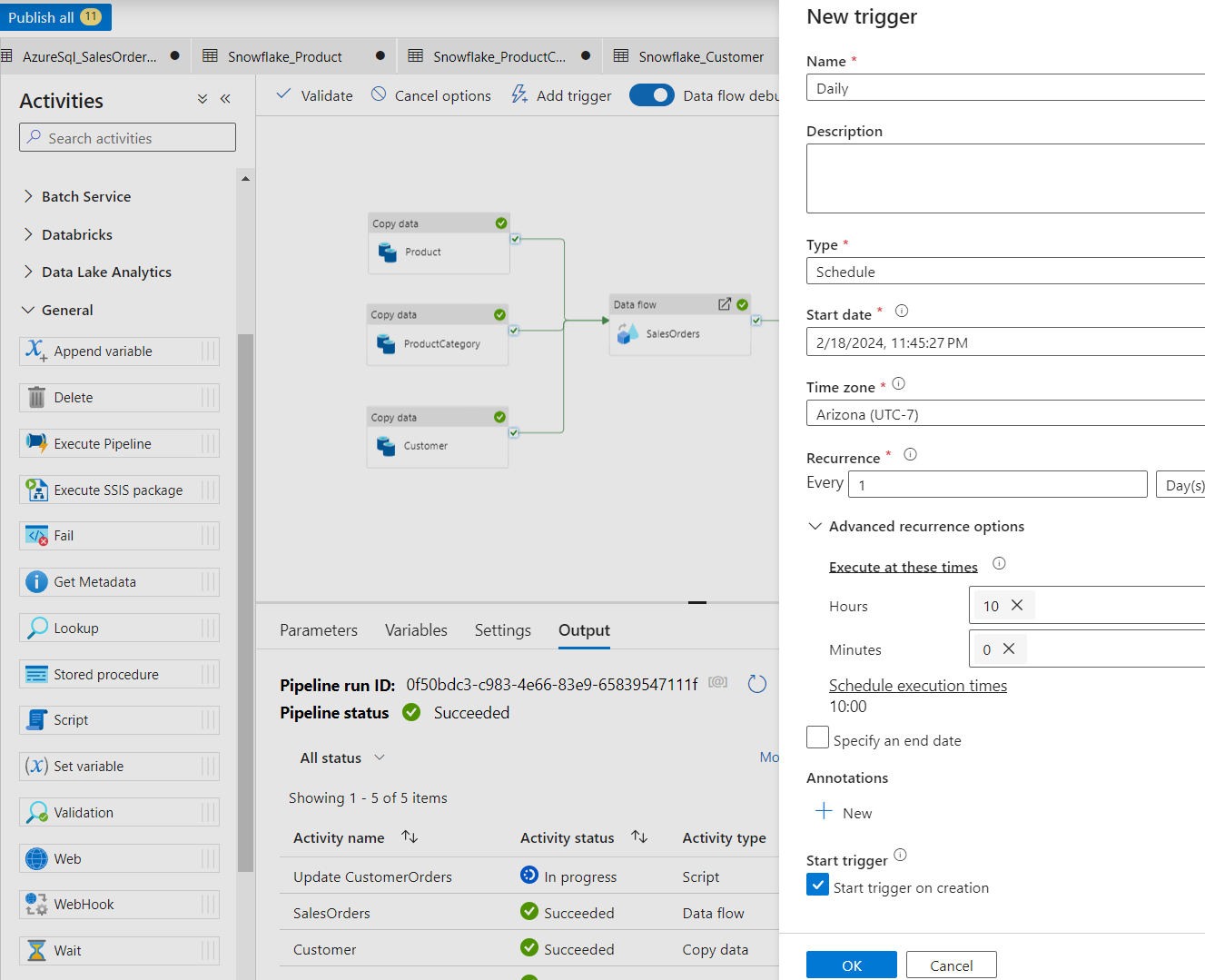
By clicking the Validate and Debug you can perform a test run of the entire pipeline.

Then go into your Snowflake instance and validate that the tables have data.

8. Triggers and Monitors

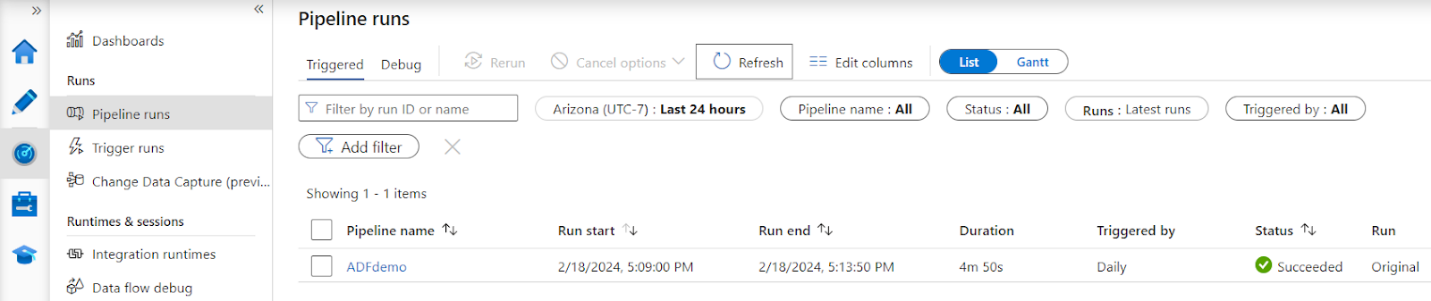
ADF is designed to be an automated pipeline orchestrator.

* At the top of the pipeline click Add Trigger, and New/Edit.
* Set Recurrence to Every 1 Day.
* Set the time to whatever works best for your testing.



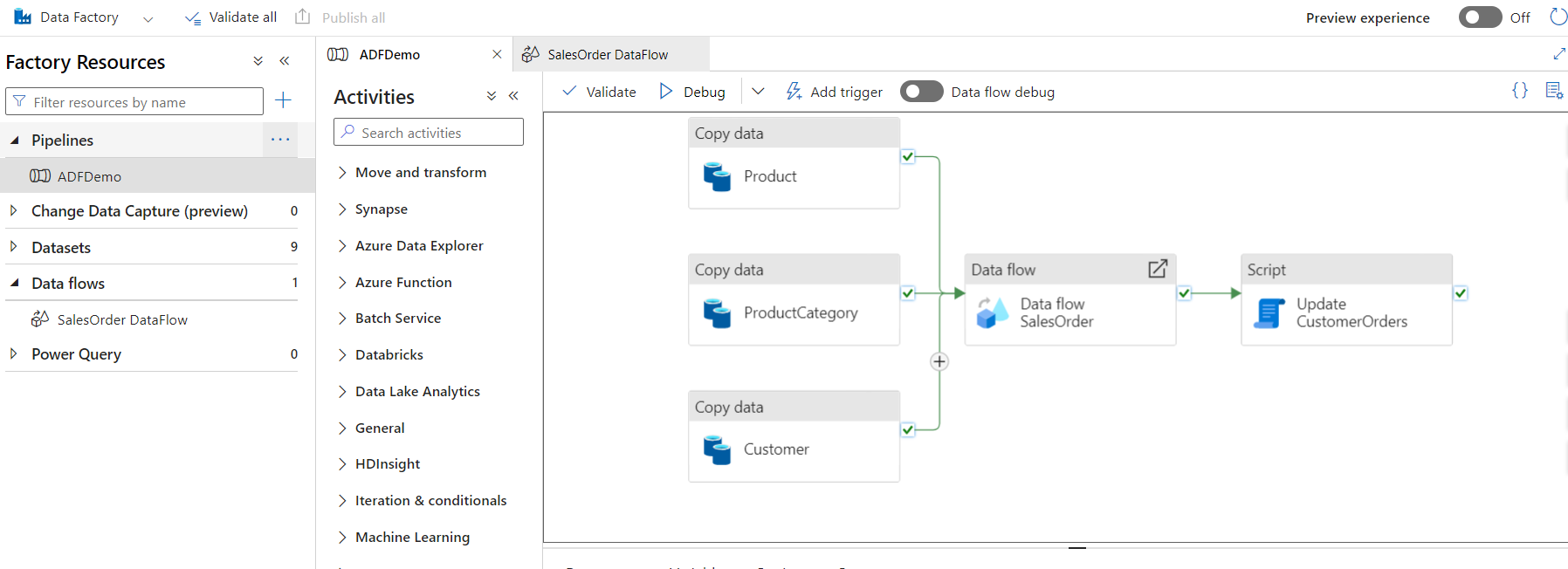
Now publish all objects created by clicking on Publish all at the top.

Once the trigger has activated and the pipeline is running, you can monitor the progress of all runs in the Monitor tab on the left menu.



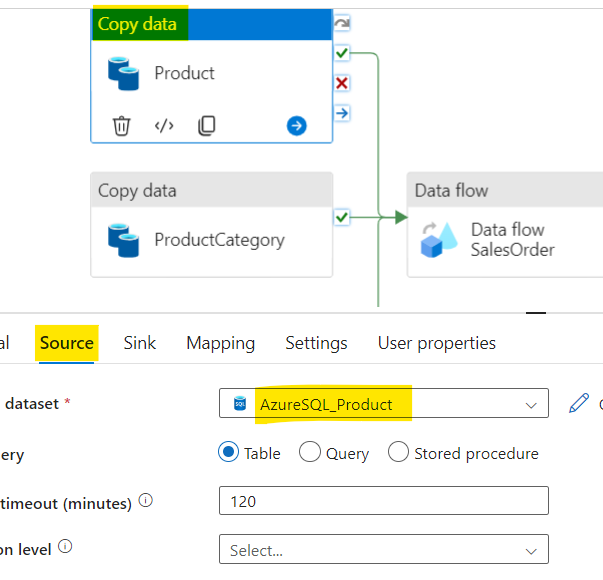
in adf

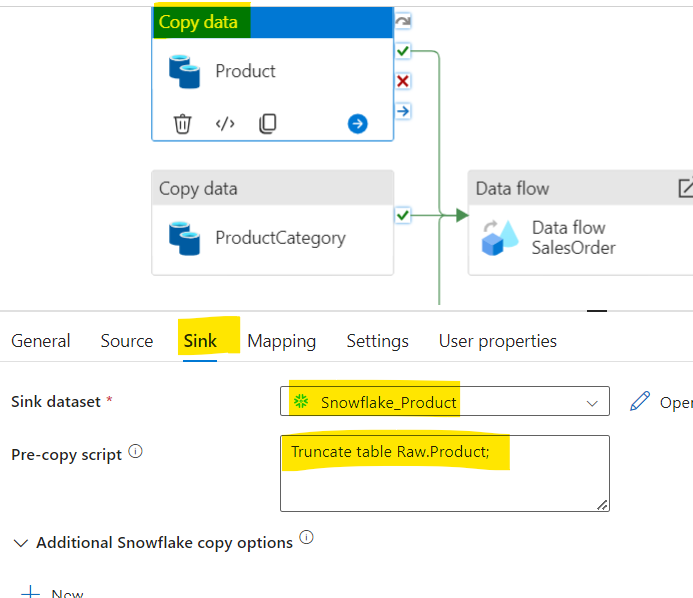
pipeline



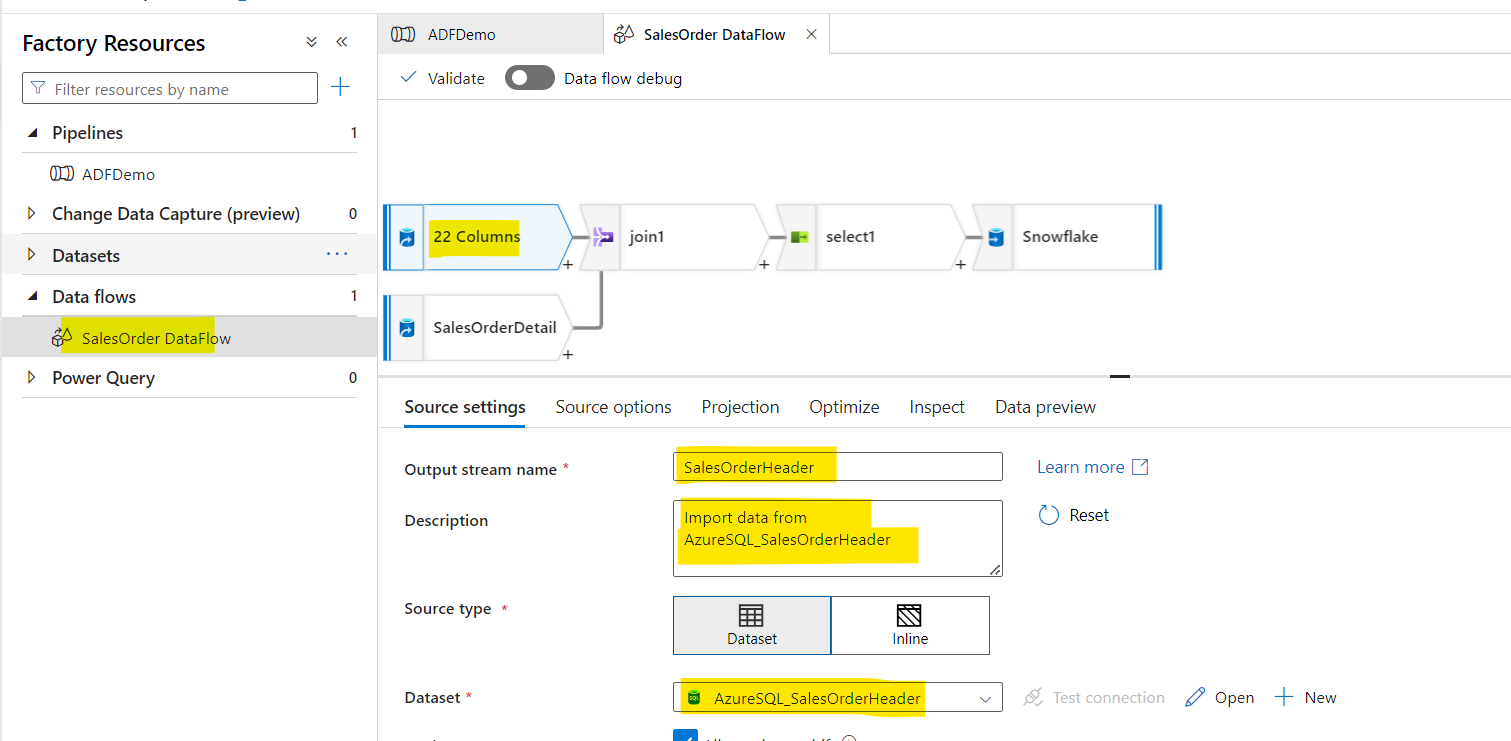
Copy

------

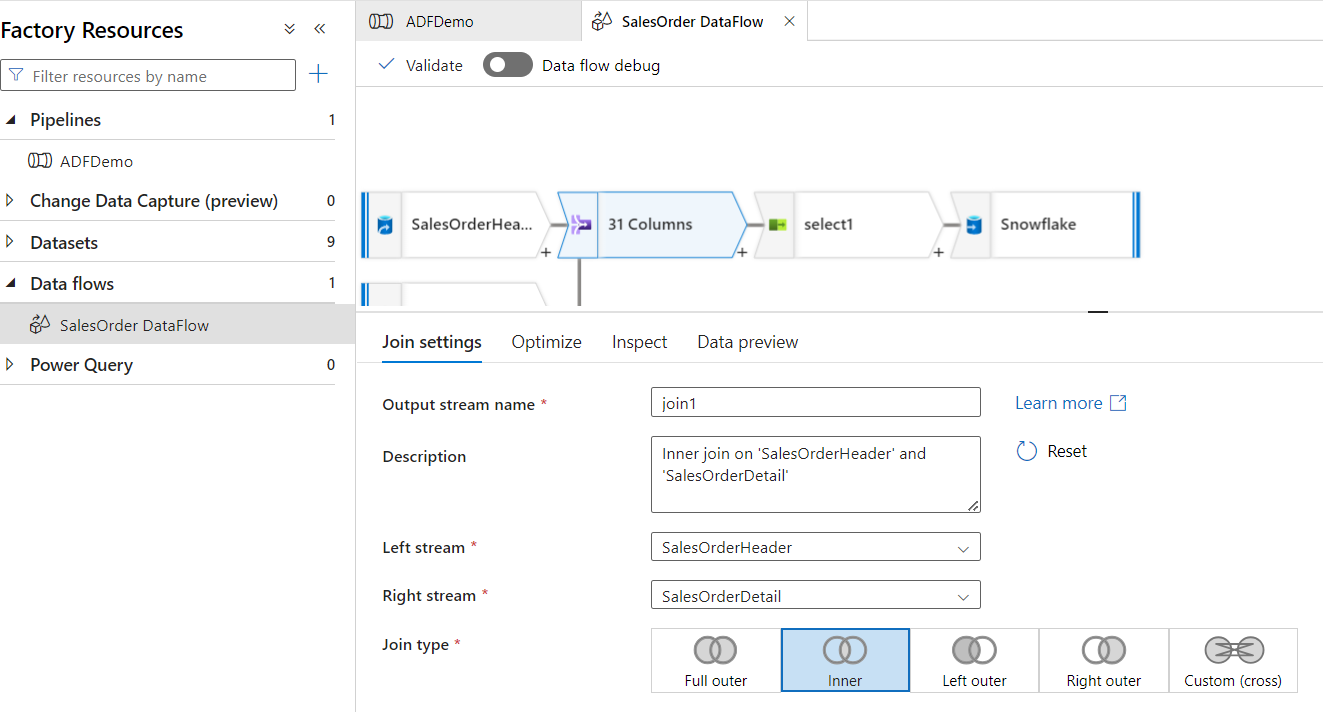




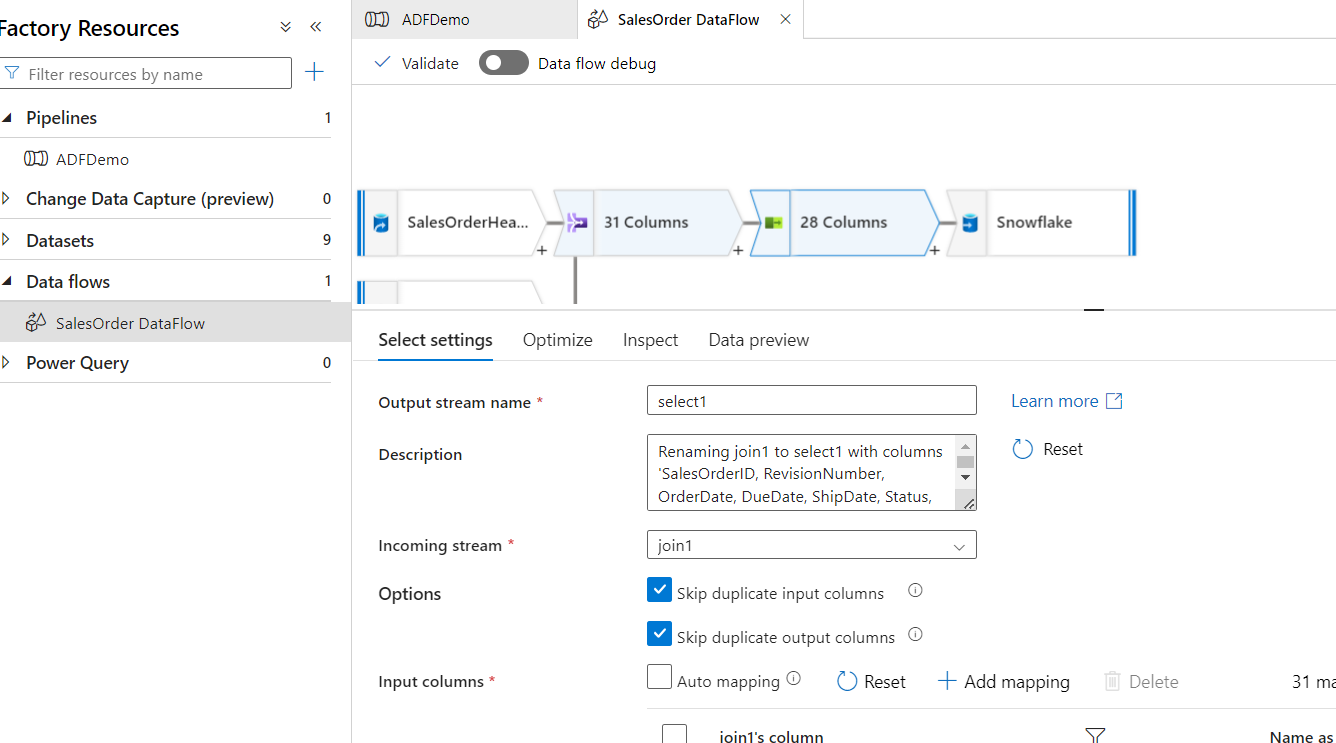
Dataflow



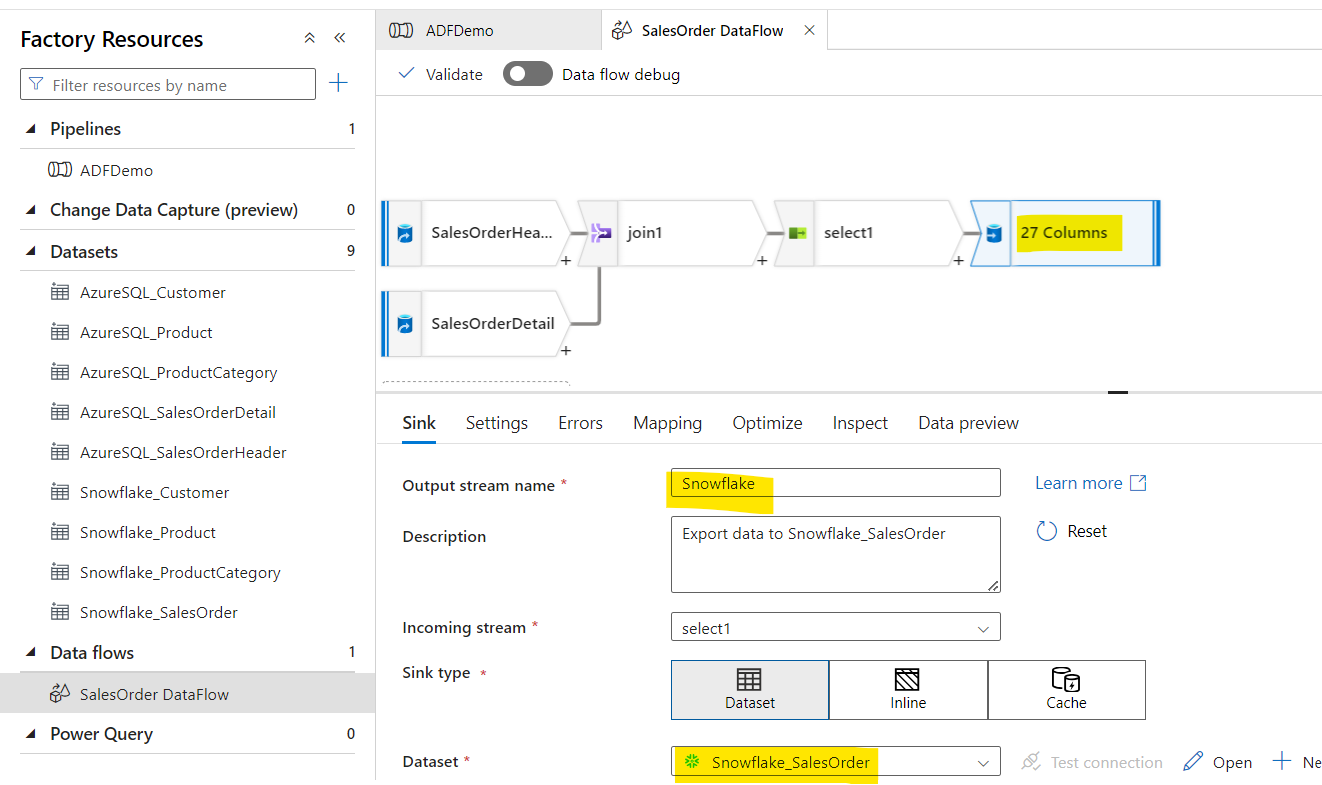
Join



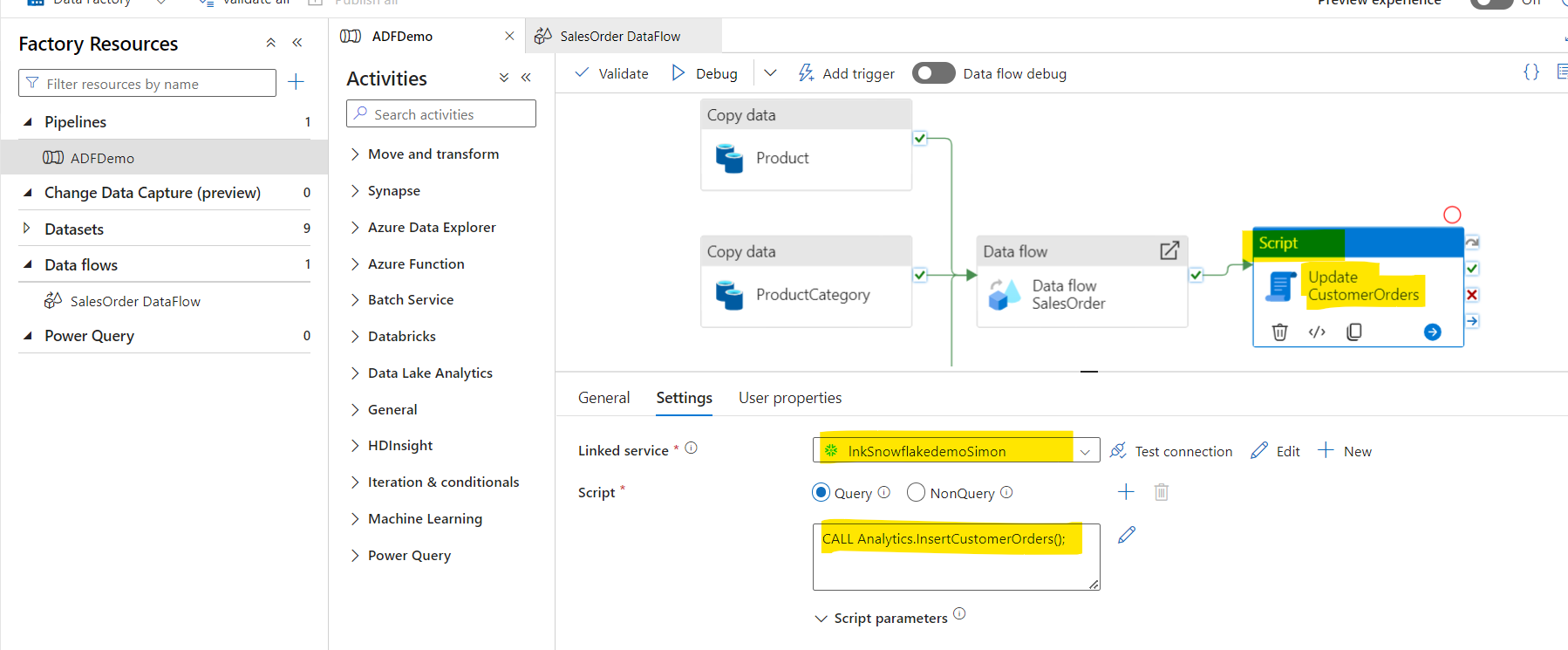
Output select



Export to snowflake



Final script



snwflk aws

usr: ctaflia2023

pwd: fli4316ERQ

Dedicated Login URL: https://dcywfks-uy97486.snowflakecomputing.com

account: DCYWFKS.UY97486

starburst

usr: ctaflia2024@gmail.com

pwd: fli4316ERQ

snwflk azure

gmail

usr: ctaflia2024@gmail.com

pwd: flia12345

usr: ctaflia2024

pwd: fli4316ERQ

Dedicated Login URL: https://lga45163.east-us-2.azure.snowflakecomputing.com

https://ufwatye-zza55073.snowflakecomputing.com

account: UFWATYE.ZZA55073

account for adf: ufwatye-zza55073

azure sql db

server name: srvadfdemo

db name:

login: ctaflia2023

pwd: fli4316ERQ

azure data factory

adf name: adfdemoSimonV2

Object (principal) ID: 26dd8672-f8e1-4baa-baa9-326c87f9f573

storage account

name: storagedemoSimon

container name: containeradfdemo

Connection string: BlobEndpoint=https://storagedemosimon.blob.core.windows.net/;QueueEndpoint=https://storagedemosimon.queue.core.windows.net/;FileEndpoint=https://storagedemosimon.file.core.windows.net/;TableEndpoint=https://storagedemosimon.table.core.windows.net/;SharedAccessSignature=sv=2022-11-02&ss=bfqt&srt=sco&sp=rwdlacupiytfx&se=2024-06-09T01:26:50Z&st=2024-05-11T17:26:50Z&spr=https&sig=6QQaz4SiXA1jI%2FeuKzAcCfGT3xOHkFYcJ0IDBsTSjf4%3D

SAS token: sv=2022-11-02&ss=bfqt&srt=sco&sp=rwdlacupiytfx&se=2024-06-09T01:26:50Z&st=2024-05-11T17:26:50Z&spr=https&sig=6QQaz4SiXA1jI%2FeuKzAcCfGT3xOHkFYcJ0IDBsTSjf4%3D

Blob service SAS URL: https://storagedemosimon.blob.core.windows.net/?sv=2022-11-02&ss=bfqt&srt=sco&sp=rwdlacupiytfx&se=2024-06-09T01:26:50Z&st=2024-05-11T17:26:50Z&spr=https&sig=6QQaz4SiXA1jI%2FeuKzAcCfGT3xOHkFYcJ0IDBsTSjf4%3D

File service SAS URL: https://storagedemosimon.file.core.windows.net/?sv=2022-11-02&ss=bfqt&srt=sco&sp=rwdlacupiytfx&se=2024-06-09T01:26:50Z&st=2024-05-11T17:26:50Z&spr=https&sig=6QQaz4SiXA1jI%2FeuKzAcCfGT3xOHkFYcJ0IDBsTSjf4%3D

Queue service SAS URL: https://storagedemosimon.queue.core.windows.net/?sv=2022-11-02&ss=bfqt&srt=sco&sp=rwdlacupiytfx&se=2024-06-09T01:26:50Z&st=2024-05-11T17:26:50Z&spr=https&sig=6QQaz4SiXA1jI%2FeuKzAcCfGT3xOHkFYcJ0IDBsTSjf4%3D

Table service SAS URL: https://storagedemosimon.table.core.windows.net/?sv=2022-11-02&ss=bfqt&srt=sco&sp=rwdlacupiytfx&se=2024-06-09T01:26:50Z&st=2024-05-11T17:26:50Z&spr=https&sig=6QQaz4SiXA1jI%2FeuKzAcCfGT3xOHkFYcJ0IDBsTSjf4%3D

storagedemosimon.blob.core.windows.net

Linked Services

azure sql db: name: lnkSQLdemoSimon

azure blob storage: name: lnkStoragedemoSimon

Snowflake: name: lnkSnowflakedemoSimon