# **Metadata driven ETL in Azure**

Modern Data Platforms are built to process data dynamically by creating metadata-driven data pipelines. A metadata-driven framework provides more scalability. It speeds up development, provides better maintainability, reusability and visibility. You can process thousands of tables and apply a variety of processing steps without designing all data flows manually.

When the data pipeline processing is designed by metadata-driven approach, data lineage won’t show up out of the box because all processing logic sits in a metadata repository. You can overcome this problem, by hooking up a data lineage registration component like Azure Purview etc.

## **Metadata Repository**

While designing a metadata-driven processing framework, you can setup a database for metadata configuration. A metadata-driven processes is a different approach than building transformations in an ETL tool. Instead, you store specifications in an external database, which fed that information to an engine, such as ADF or Spark.

In this blogpost I am using an SQL Server database as metadata repository. This initial framework consist few tables ,views and stored procedures for defining the components used in the Data platform. The source and destinations are SQL Server databases. AdventureWorksDW2019 is used as source. I have created destination database as AdventureWorksUAT. The metadata repository also exist in the destination database. You can find the code for tables, views and stored procedures in my [Git repository](https://github.com/ashok-poreddy/mdp-etl-azure/tree/main/metadata-repository) here.

### **1.1) Metadata – Tables**

* **ETL Systems:** Describes all data sources, destinations.
* **ETL Data Period:** Describes volume of live data and historical data based on a date.
* **ETL Tables:** Describes the tables to be loaded from a source.
* **ETL Columns:** Describes the columns of each tables to be loaded.
* **ETL Pipelines:** this is a placeholder for the data pipelines, including name and description.
* **ETL Log Tables:** Used for storing logging results for tables load status.

In a real time, comprehensive framework there are additional metadata tables for validations, transformations, scheduling, triggering pipelines, security on data usage etc. These are not in the scope of this blog. We will cover this in next blog post.

* **ETL Transformations:** Include references to source, destination, columns etc.
* **ETL Aggregations:** Describes aggregations on numeric columns group by subject oriented dimension keys.
* **ETL Log Pipelines:** Used for storing logging results for pipelines execution status.

### **1.2) Metadata – Views**

* **vLogSuccessTables:** Display table load success details from the [dbo].[ETL Log Tables] table.
* **vLogFailureTables:** Display table load failure details from the [dbo].[ETL Log Tables] table.

### **1.3) Metadata – Stored Procedures**

* **spGetSourceConnection:** Get the source connection details like server, database, username, password.
* **spGetDestinationConnection:** Get the destination connection details like server, database, username, password.
* **spGetTableList:** Get the list of tables to be loaded to the destination database. Table names are configured in [dbo].[ETL Tables] table whether to load or not.
* **spLoadTable:** Generate the load script (SELECT query) for a given source table based on the configurations specified in [dbo].[ETL Columns] table. This script is used in the ADF copy activity.
* **spTruncateTable:** Truncate the given table before loading data. This used in ADF copy activity’s pre-copy script property of sink section. Truncating a table before loading data is varied based on business requirements, in this blog I am trying to empty the table then load data.
* **spLogTableLoad:** Log the status of data loading for each table in [dbo].[ETL Log Tables] table. Logging details including Status (Success/Failure), TableName, StartTime, EndTime, Duration, RecordCount etc.

## **Data Landing Zones**

While loading data from source to destination, it is necessary to clean, validate and transform the data according business rules. You need to have data landing zones like RAW, STAGE, CURATED etc. These landing zones along with the transformations are part of the data life cycle i.e., from data capturing to data consuming.

* **RAW:** Data is in original format without any validations. It’s read-only and accessible to service accounts only.
* **STAGE:** This is general staging after applying cleaning, validation etc. Accessible to data engineers, data analysts, data scientists.
* **EDW/CURATED:** Data is processed after applying different transformations and ready to consume for dashboards. Accessible to data engineers, data analysts, data scientists.
* **SANDBOX:** This is laboratory purpose to do EDA (Explanatory Data Analysis) for AI-ML applications. Accessible to data engineers, data scientists.
* **TEMP:** Supports ingestion framework by creating and cleaning temporary data sets etc. Data is short lived here. Accessible to service accounts, data engineers.
* **MASTER:** It is reference data and archive data. Accessible to data engineers, data analysts, data scientists.

We can configure these landing zones in the metadata repository, but it is not in scope of this blog due to its complexity. We will cover this in next blog post.

## **ADF**

Once metadata repository is setup, you can configure our pipelines. ADF provides many features to develop metadata driven pipelines. Different activities are used to get the metadata configurations and make the pipelines as engines in the data platform.

For example, Lookup activity is used to fetch the details of components to be loaded and processed. Components like list of tables and columns from each table are already configured in the metadata repository tables. Then activities like For-each, Copy, Data Flow are used to load and perform data processing further.

### **3.1) SHIR – Selfhosted Integration Runtime**

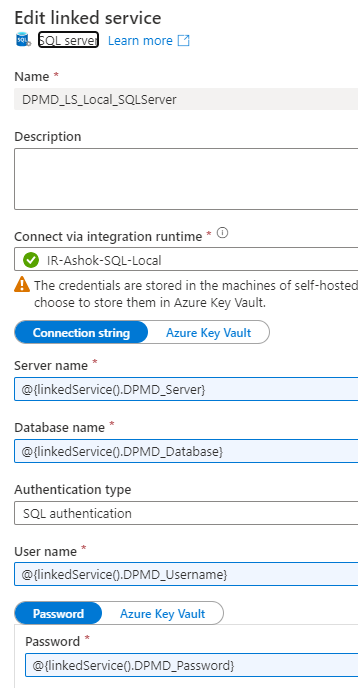
In this blog post I am using SQL Server on my local machine. SHIR need to be installed and configured in the local machine. Please follow below link to know how to install and configure SHIR.

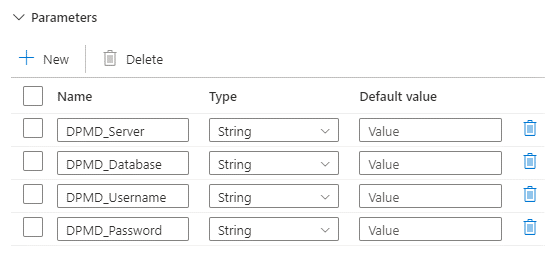
* [SHIR Setup](https://learn.microsoft.com/en-us/azure/data-factory/create-self-hosted-integration-runtime?tabs=data-factory)

### **3.2) Linked Services**

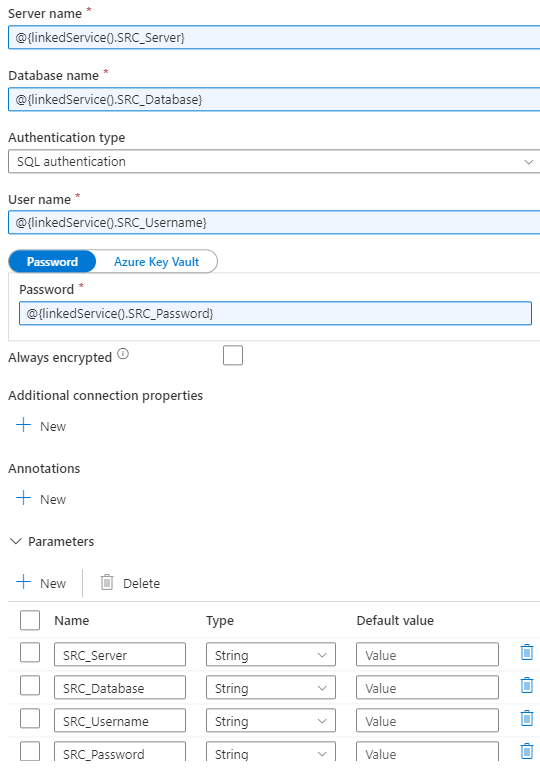
Below linked services and data sets are configured. They are utilized in various activities of the ADF pipelines.

**1. DPMD\_LS\_Local\_SQLServer:** It is used to connect to the on-prem SQL Server metadata repository and get connection details for source and destination.

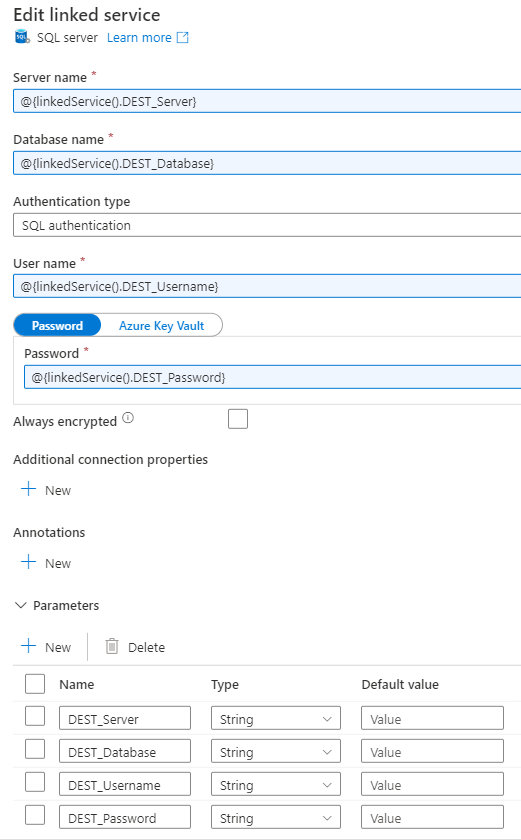




**2. DPMD\_LS\_SRC\_Local\_SQLServer:** It connect to the on-prem source SQL Server.

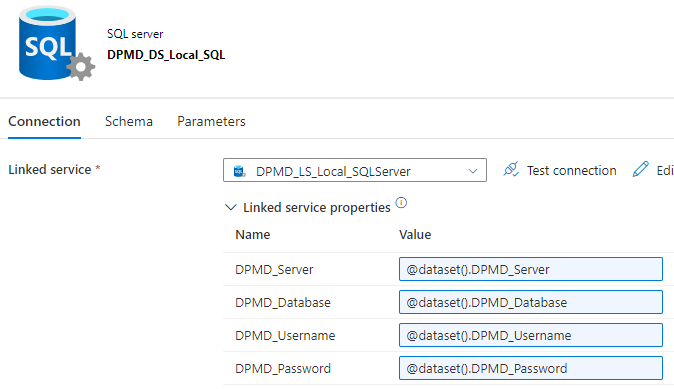


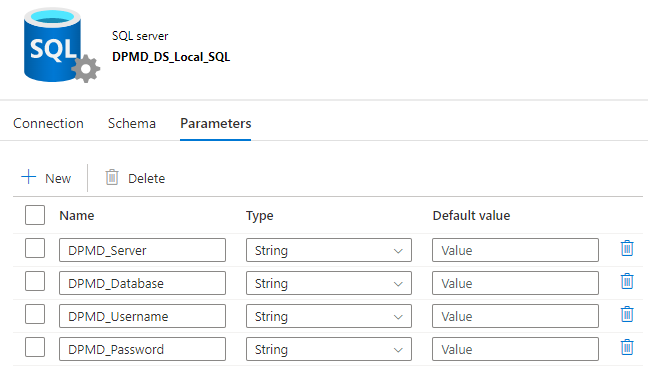
**3. DPMD\_LS\_DEST\_Local\_SQLServer:** It connect to the on-prem destination SQL Server.



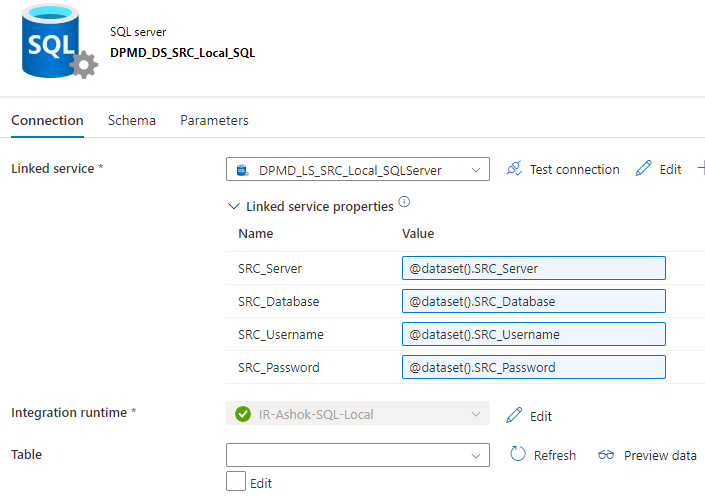
### **3.3) Data sets**

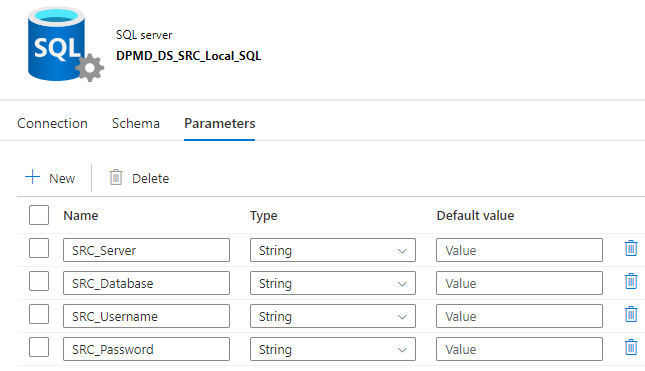
**1. DPMD\_DS\_Local\_SQL: C**onnect to the on-prem SQL Server metadata repository.



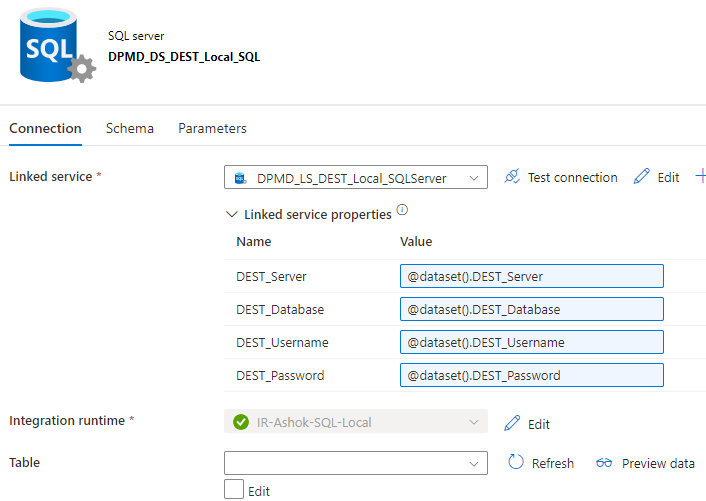


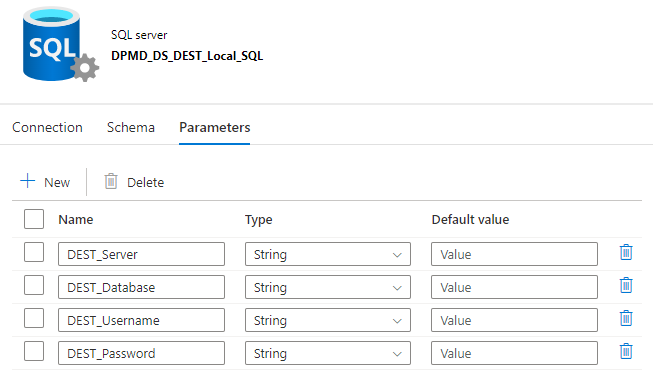
**2. DPMD\_DS\_SRC\_Local\_SQL:** Connect to the on-prem source SQL Server.



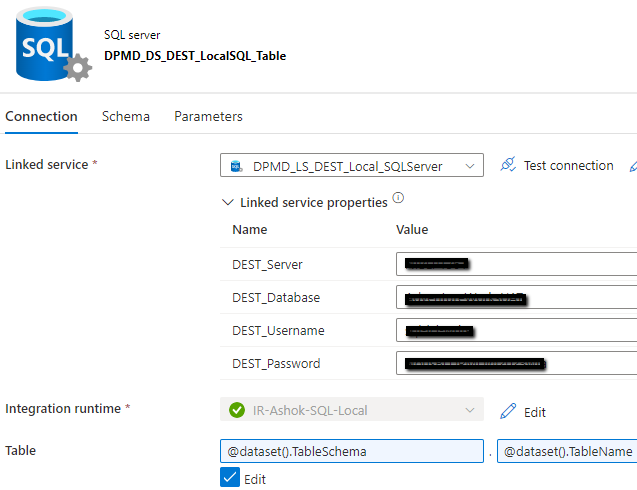


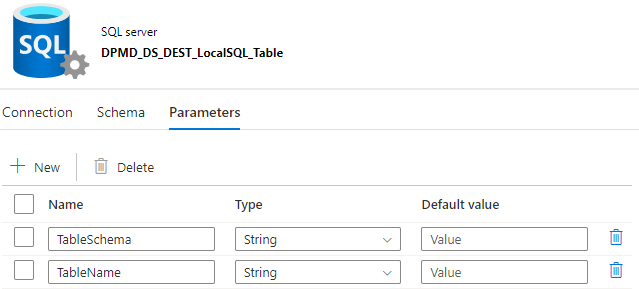
**3. DPMD\_DS\_DEST\_Local\_SQL:** Connect to the on-prem destination SQL Server.





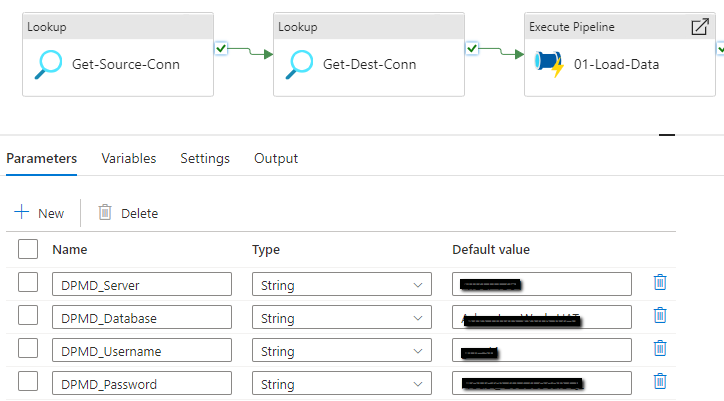
**4. DPMD\_DS\_DEST\_LocalSQL\_Table:** Connect to the on-prem destination SQL Server.





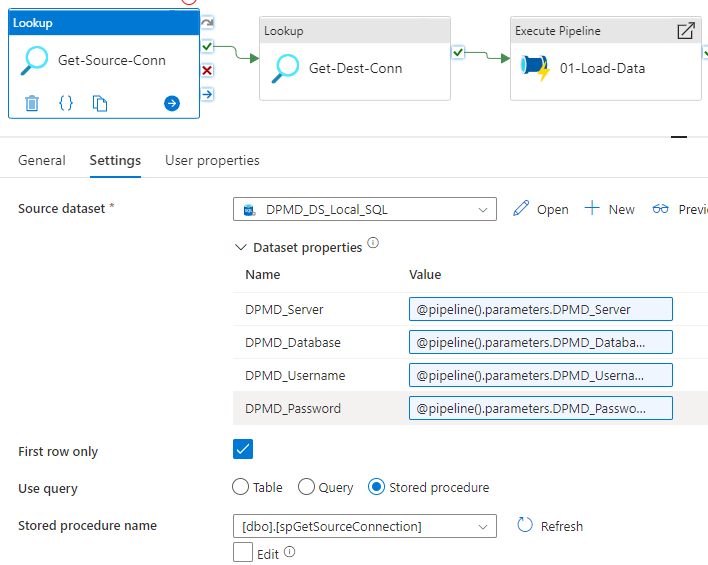
### **3.4) 00-Get-Connections**

This pipeline consist parameters for connecting to metadata repository and capture the connection details for source and destinations. Parameters can be passed from a scheduling trigger or configured manually at the time deployment.



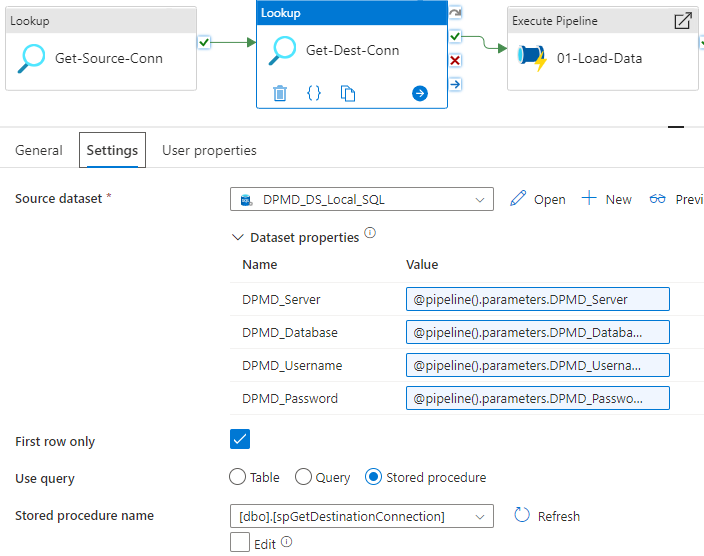
#### **Get-Source-Conn**

This lookup activity call the stored procedure [dbo].[spGetSourceConnection] and get the source connection details like server, database, username, password.



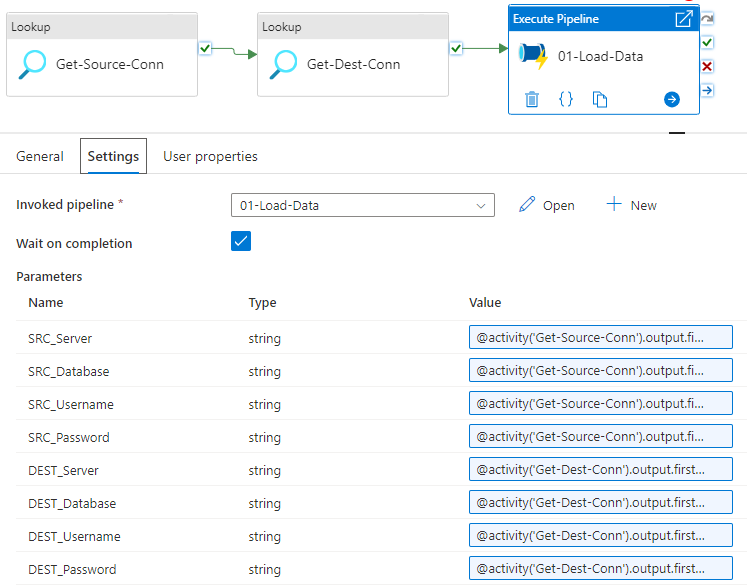
#### **Get-Dest-Conn**

This lookup activity call the stored procedure [dbo].[spGetDestinationConnection] and get the destination connection details like server, database, username, password.



#### **Load-Data**

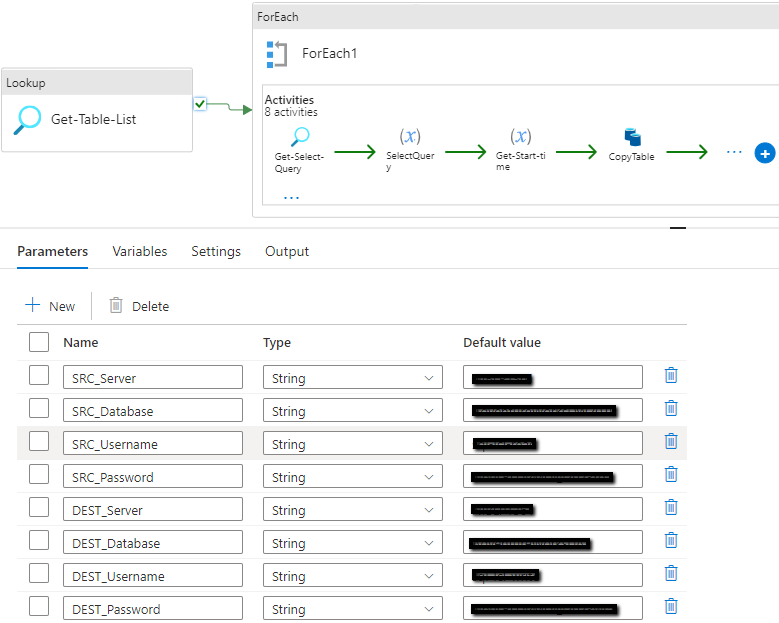
This execute pipeline activity execute other pipeline to load the data. It pass the parameters to the pipeline for source and destination connections, which are captured in the lookup activities.



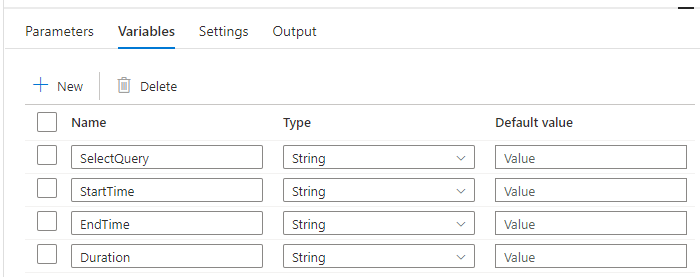
**Example:** @activity('Get-Source-Conn').output.firstRow['SourceServer']

### **3.5) 01-Load-Data**

This pipeline take the parameters from parent pipeline for source and destination connections and use them in various activities.



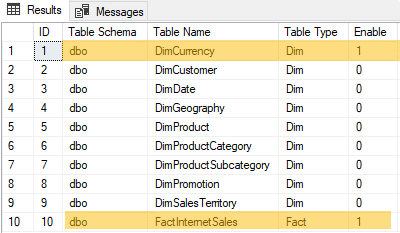
It also consist below variables which are assigned with values and used in the activities.



#### **Get-Table-List**

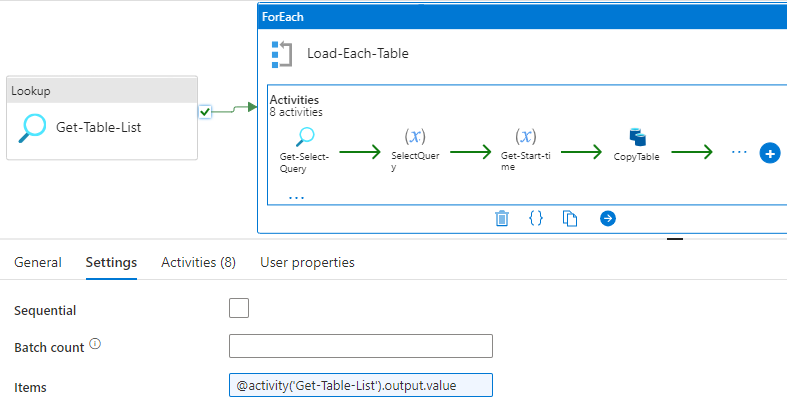
This lookup activity call the stored procedure [dbo].[spGetTableList] and get the list of tables to loaded based on the configurations in the metadata repository table [dbo].[ETL Tables]. Tables tagged with value 1 for column [Enable] are considered for loading.

Example:

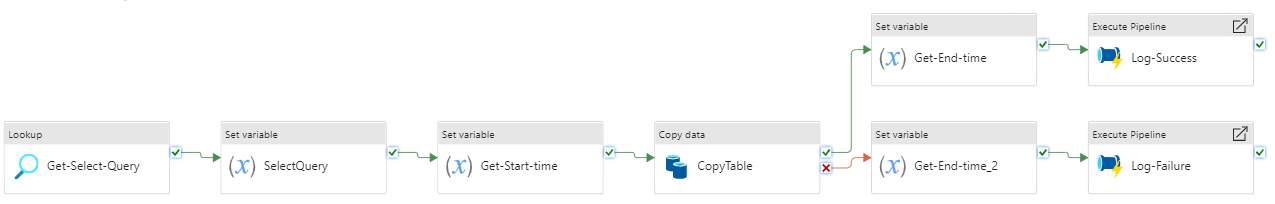


#### **Load-Each-Table**

This for-each activity load each table captured in the above lookup activity.

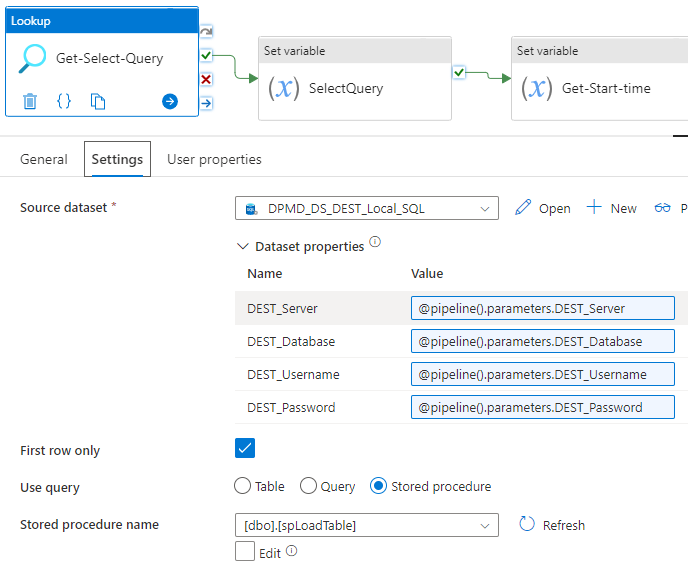


It consist below activities to accomplish the task.



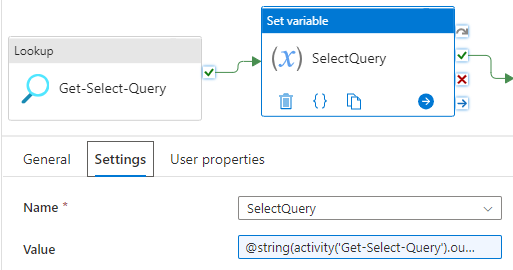
##### **Get-Select-Query**

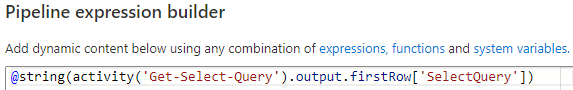
This lookup activity call the stored procedure [dbo].[spLoadTable] and get the generated select query based on the configurations in the metadata repository table [dbo].[ETL Columns].



##### **SelectQuery**

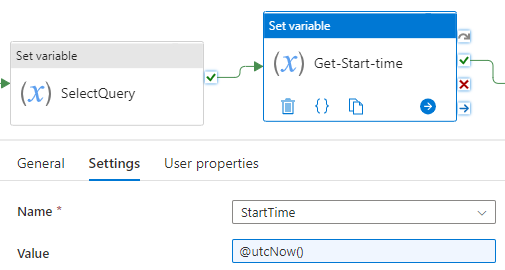
This set variable activity assign the generated select query to a variable. This query is then used in the copy activity source section.





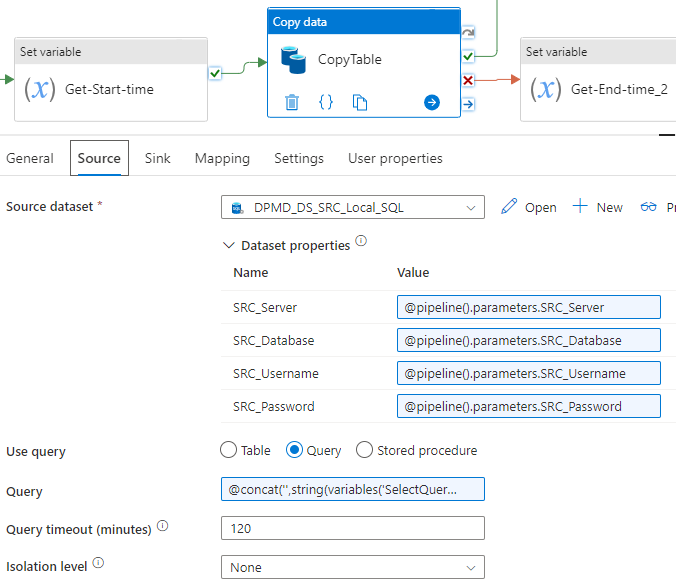
##### **Get-Start-time**

This set variable activity assign the start time of a table load to a variable. This value is then used in the logging part.



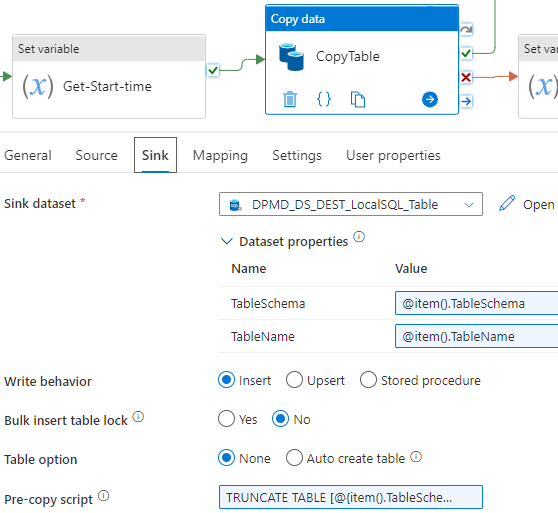
##### **CopyTable**

This copy activity execute the select query at source database and load the data to destination database. To simplify the task, I used same table name at the destination database also.



**Query:**

@concat('',string(variables('SelectQuery')),'')



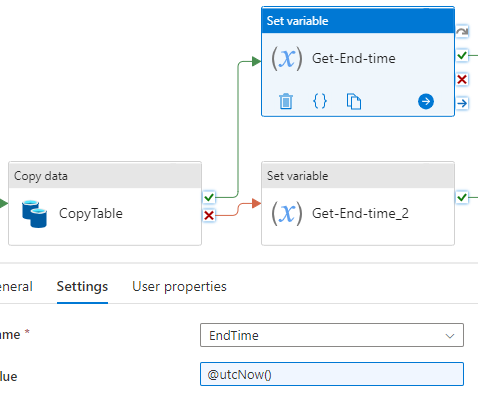
TRUNCATE TABLE [@{item().TableSchema}].[@{item().TableName}]

##### **Get-End-time**

This set variable activity assign the end time of a table load to a variable. This value is then used in the logging part in case of success.

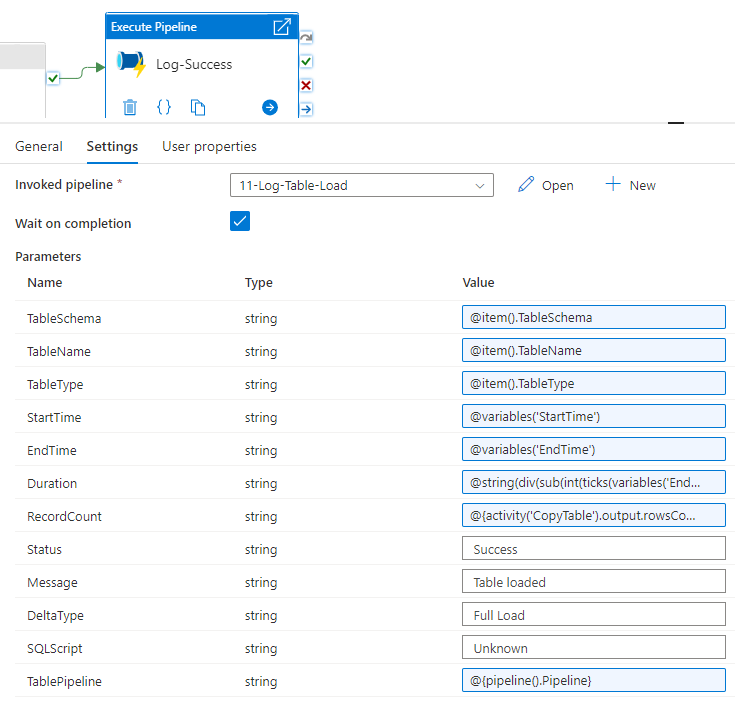
##### **Get-End-time\_2**

This set variable activity assign the end time of a table load to a variable. This value is then used in the logging part in case of failure.



##### **Log-Success**

This execute pipeline activity run the pipeline 11-Log-Table-Load and log the success details to [dbo].[ ETL Log Tables] table. It pass the below parameters to the pipeline.

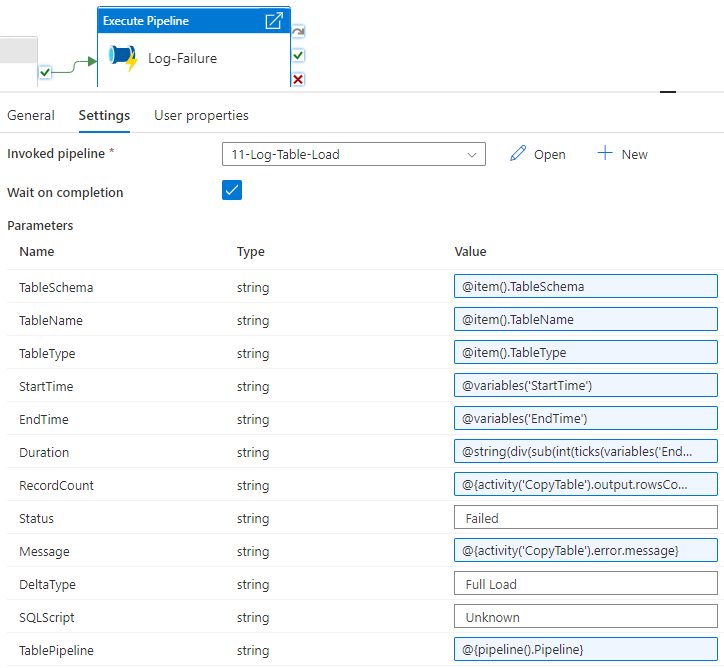


@string(div(sub(int(ticks(variables('EndTime'))),int(ticks(variables('StartTime')))),600000000))

@{activity('CopyTable').output.rowsCopied}

##### **Log-Failure**

This execute pipeline activity run the pipeline 11-Log-Table-Load and log the success details to [dbo].[ ETL Log Tables] table. It pass the below parameters to the pipeline.

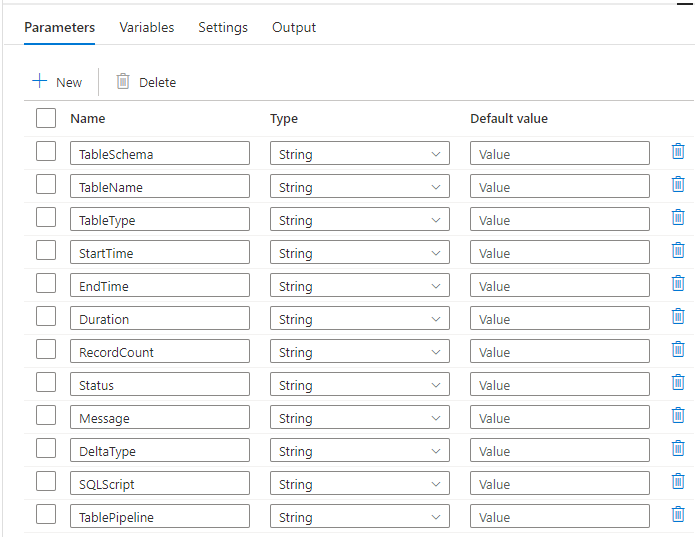


@string(div(sub(int(ticks(variables('EndTime'))) ,int(ticks(variables('StartTime')))),600000000))

@{activity('CopyTable').output.rowsCopied}

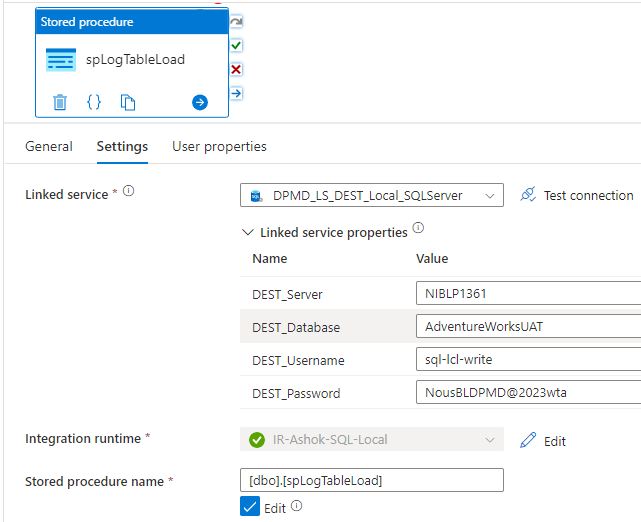
### **3.6) 11-Log-Table-Load**

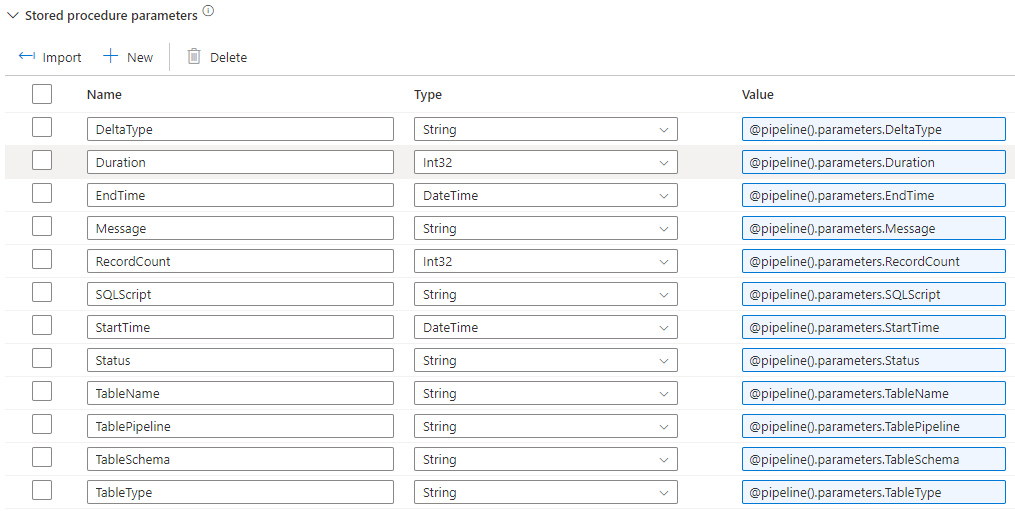
This pipeline consist the parameters for logging details. It execute a stored procedure to log the details in case of success and failure.



##### **spLogTableLoad**

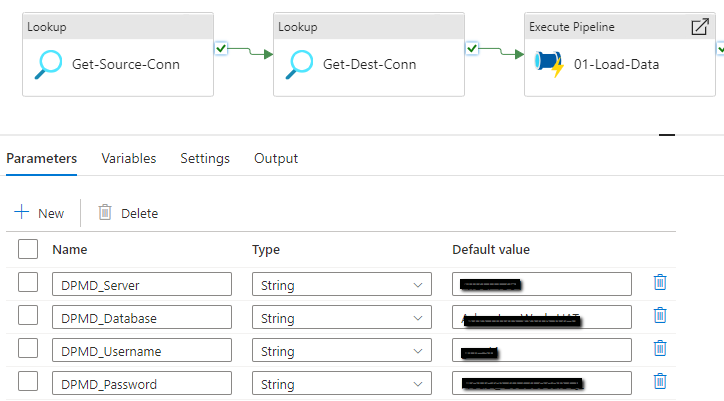
This activity call the stored procedure [dbo].[spLogTableLoad] and log the details based on the passes parameter values.





## **Executing the pipeline**

You can run the parent pipeline 00-Get-Connections by providing metadata connection details like server, database, username, password through pipeline parameters. Then it will capture source and destination connection details, table list then load the data.



## **Conclusion**

In this blog I have explained a metadata driven approach to develop a Data Platform for SQL Server based data stores. It possible to extend this to include different tools like Data Lake for staging, Databricks notebooks for data processing. We can even configure trained ML Models and map the data set to those models for next level Advanced Analytics utilizing AI-ML services, libraries with a programming language like Python.