Introduction to ETL jobs in Azure Synapse Analytics

In the previous chapter, we introduced you to Azure Synapse Analytics data pipelines as the code-free tool inside of your Synapse Studio used for building data integration and ETL jobs. In this chapter, we are going to focus on one of the key activities in the pipeline toolbox that we briefly touched on previously called “Data Flows”. Synapse pipelines are where you go to build workflow logic and data flows are where you go to build data transformation process for an ETL job. Both features originated in Azure Data Factory (ADF) and both are built for scale. Data Flows process and transform data in a code-free manner, making it super-easy for building ETL jobs. You can scale data flows easily because Synapse uses Synapse Spark under the hood for transforming TBs of data.

In Synapse pipelines, you’ll add data flows when you are building ETL jobs that are intended to transform and process data for analytics, ultimately landing the refined data in either a Synapse SQL pool or in the data lake. By using a visually-oriented scale-out tool like Synapse Data Flows, data engineers can provide business-ready data warehouses and lakehouses through Synapse pipelines as a way to facilitate business intelligence reporting tools for making data-driven business decisions.

Tour of Data Flows

In Synapse Studio, you’ll find data flows in the Develop section of the UI (see figure 3-1). From there, you can create new data flows using the + new button and you can manage existing data flows including organizing the artifacts in folders.

Graphical user interface, text, application

Description automatically generated

Figure 3-1. Data flows are in the “develop” section of the

You can get started quickly by clicking + new to create a new data flow, which will launch a new blank canvas where for designing your data flows. In the next section, we’ll examine the data flow design surface and walk through many different aspects of the data flows designer in Synapse.

The data flow design surface

When you first create a new data flow, you will notice that the “Add Source” tile is always present with a dotted outline. Clicking there will allow you to add as many sources as you need for our data flow job. Each of the additional tiles in your data flow that you will add as you build your ETL job represents a transformation “primitive”. These are the building blocks that you’ll use to define your job as a left-to-right process that you’ll construct from source, to transformation, to sink, which is what defines where your processed data will land (figure 3-2).

Diagram

Description automatically generated

Figure 3-2. The data flows design surface

To add transformations to your data flow, click the + next to each tile and you’ll see the transformation toolbox appear. As you construct your graph design, you can zoom in and out in the designer and search the graph (figure 3-3).

Icon

Description automatically generated

Figure 3-3. Zoom in / out and search on the data flow designer

As you build out your ETL processing logic, you’ll notice that the construction paradigm in the data flow designer UI is quite different than the workflow DAG format of the pipeline designer. This is intentional as pipelines are built for the purpose of automating workflows orchestrating processes, while data flow graphs construct a set of instructions that ADF will interpret as data transformation logic. The data flow designer is a “construction-oriented design” intended to build a complete end-to-end set of transformation semantics that will be interpreted by a Spark job that is managed by Synapse. Because the data flow designer is not a free-form graph, the mechanism you’ll use to move nodes is click on the tile you wish to move and change the “Incoming Stream” property to connect it to a different node in your graph.

The data flow graph is essentially a graphical representation of a transformation script (known as Data Flow Script) which is a DSL (domain-specific language) that provides a set of instructions for Synapse. There is a Spark executor job that will receive this script payload and execute your instructions on distributed Spark clusters. You can always view and edit the script behind the graph manually. You’ll find a script button on the top ribbon of the designer (figure 3-4) that launches a script editor (figure 3-5) with full Intellisense and auto-complete so that you can view and edit the script behind the graph.

Graphical user interface, application, table

Description automatically generated

Figure 3-4. Data flow script-behind button

Graphical user interface, text

Description automatically generated

Figure 3-5. Data Flow Script example

Data transformation primitives

Transformation primitives are the transformation blocks that appear in a toolbox (figure 3-6) when you click the previous tile’s plus sign. You’ll use these as building blocks to construct your data flow. They are the fundamental set of actions that will be taken against your data that you’ve defined as the source and the final result to be written in the sink. Each data flow can contain n-number of sources and sinks. The limits to the amount of data, sources, and sinks that a data flow contain is based on the scale limits that you’ve defined in your Azure Integration Runtime data flow settings, which we will discuss later in this chapter. For now, let’s talk about the different transformation categories available to you.

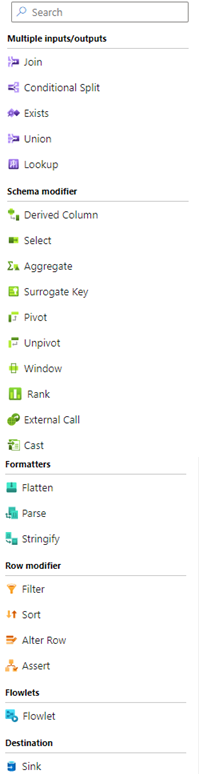


Figure 3-6. Transformation toolbox

Source transformation

Notice that in the toolbox there are not any source transformations. That is because the design surface will always show at least one new source that you can configure to add a new source to your data flow. Simply click on the dotted-lined box to configure sources for your data flow. All sources will appear on the far-left of your graph so that you can easily navigate to find all of your data sources.

There are 3 types of sources that you can configure in Synapse data flows:

1. Integration dataset is a shared dataset in Synapse similar to ADF datasets that we described in the previous pipeline chapter
2. Inline datasets are defined only within the context of the data flow. They are not shared across other artifacts inside your Synapse workspace. These are important for data flow-only connectors and can be used in Flowlets for creating reusable data flow pattern templates, which we will touch on later
3. Workspace DB is a unique data flow source only available in Synapse. You Synapse workspace data sources like Lake DBs will appear here without any dataset configuration required. Synapse already has the connection information for your data stored in the Synapse metastore, making this connection type possible only in Synapse.

Graphical user interface, application

Description automatically generated

Figure 3-7. Sample data flow source configuration

Multiple inputs / outputs

In this category of transformations, you’ll find transformations that will take one or more inputs and will then output one or more streams. In Synapse data flows, think of “streams” as data moving throughout your flow on each separate row of your graph.

New Branch

If you wish to duplicate an existing stream for the purpose of branching to apply separate transformations to the same data stream, use the New Branch transformation. You must add another subsequent transformation to the new branch in order to make this new duplicate stream valid. On the design surface, the graph UI will duplicate the node and shows a connector line between the original and new branch (figure 3-8).

Diagram

Description automatically generated

Figure 3-8. “SurrogateKey” was duplicated with a New Branch, indicated in the graph with the grey connector line

Join

One of the most common uses of data flows in Synapse is combining, or joining, data together (figure 3-8). Data flows will place all of your source data into Spark data frames, which surface inside your graph as “streams”. This means that you can essentially build complex and powerful data virtualization in Synapse using data flows by joining the data together in a single federated view. There is also an optional “fuzzy matching” feature that is very useful when joining data in the lake. The nature of lake data is that it will be quite varied and how low data quality. With fuzzy matching you can join data that is similar rather than exact matching in figure 3-9, you can see an example of setting the similarity threshold. The resulting match score is also visible in the results in your data sets in Synapse data flows.

Graphical user interface, text

Description automatically generated

Figure 3-8. Join transformation configuration panel

Graphical user interface, text, application

Description automatically generated

Figure 3-9. Fuzzy matching join

Conditional Split

Think of the conditional split as a case statement or if/then/else where you will split your data based on a boolean expression. You’ll set the condition using the data flows expression builder and expression language, which we will talk about a bit later in this chapter. You can use the transformation settings panel (figure 3-10) to set the different paths to take based on expressions.

Graphical user interface, text, application

Description automatically generated

Figure 3-10. Conditional split transformation

Exists

Use the exists transformation when you need to compare two streams of data for the existence or non-existence of values. This transformation acts very similar to a SQL Exists statement but can be used with any source types, making it a very powerful data lake construct.

Union

Similar to a SQL union operator, the union transformation can combine two or more streams together. You can union the streams based on either the column names or based on ordinal position (figure 3-11).

Graphical user interface, text, email

Description automatically generated

Figure 3-11. Union transformation

Lookup

Lookup provides the ability to enrich your data stream with reference data from another stream. It is a real-time lookup which is useful for data lookups where the reference data does not change frequently. Synapse data flows also support cached lookups which can be quicker than the Lookup transformation. While the lookup transformation is slower because it must perform the lookup against the source reference data for every row, it is more resilient to frequently changing data, whereas the cached lookup is static.

Schema modifier

Schema modifier transformations are perhaps the most powerful transformations in the toolbox and will take the most compute power because they will affect the output and the final shape of your data. As you modify your schema using these transformations in your graph, you’ll be able to view the change in the shape of your data using the Inspect panel in each transformation.

Derived Column

You’ll use the derived column transformation very often. This transformation allows you to change existing values and add new columns using the expression builder and the Synapse transformation expression language that we will talk about shortly.

Select

Roughly equivalent to the column selection in a SQL select statement, the Select transformation will allow you to reorder, rename, and basically reshape your schemas.

Aggregate

When building ETL processes that load analytical databases and schema models, you will very often need to summarize and aggregate disparate data. With the aggregate transformation, you can collect values together using an aggregate function from the data flow expression language using the expression builder with an optional group-by clause also using columns or expression, similar to how SQL aggregate functions work. Note that if you do not group-by, data flows will have to aggregate across your entire dataset, which can be a blocking operation at runtime that you may result in long-running data flows when a lot of data is present.

Surrogate Key

The surrogate key transformation is used to create an incrementing unique key value that is very useful when managing dimension tables in a data warehouse.

Pivot

Use the pivot transformation to pivot row values into columns. In figure 3-12, you’ll see how the transformation configuration requires you to pick columns for group by and for aggregation of the pivoted values.

Graphical user interface, text, application, email

Description automatically generated

Figure 3-12. Pivot transformation

Unpivot

Unpivot can be thought of as the opposite of Pivot and transforms columns into row values. This is commonly used when normalizing data.

Window

You’ll find the Window transformation acts very similar to SQL windows functions. Basically, Window will allow you to utilize all of the window and aggregation functions in the data flow library and apply those functions across windows of data that you define in the “over” clause (figure 3-13).

Graphical user interface, text, application, email

Description automatically generated

Figure 3-13. Window transformation

Rank

The rank transformation is a scaled-out version of the rank() and denseRank() functions from the Window library above. It is built for optimal performance using large datasets and is recommended for use cases where you do not require a window “over” clause and instead are ranking all of your source data.

Graphical user interface, application

Description automatically generated

Figure 3-14. Rank transformation

External Call

The external call transformation is a great way to enrich your data or to call custom routines from web services. External call support Get, Post, Patch, Put, and Delete REST request methods.

Cast

In the derived column transformation described above, a very common action that you’ll perform is casting column types. There is also a stand-alone cast transformation that makes it very easy to visually cast data types and set an assertion if the casting fails. We’ll talk more about asserts later. For now, think of the cast transformation as a simple way to both cast and check for errors without needing to add separate derived column and assert transformations.

Graphical user interface, application

Description automatically generated

Figure 3-15. Cast transformation

Formatters

When dealing with complex data formats and structures, you will want to look into the formatters transformations.

Flatten

As a data engineer working with data lakes, you are going to often work with data that is hierarchical with structures and arrays. In Synapse data flows, you can unroll arrays using the Flatten transformation. This is very useful when you need to convert arrays of data into tabular relational data stores. For example, take arrays from JSON documents and flatten them into repeating rows for each value in the row index and then store the new resulting rows in a Synapse SQL pool.

Parse

Sometimes, hierarchical data can be embedded inside string fields in your relational source data. When your data contains embedded structures, use the Parse transformation to turn that plain text into the appropriate structures.

Stringify

Stringify is essentially the inverse of the parse transformation. This transformation will take complex data types like structures and turn them into strings so that they can be consumed downstream in your data flow by targets like delimited text files.

Row modifier

Row modifier transformations can change the order of rows, set flags on rows based on criteria that you specify, filter rows based on your criteria, and set rows as errors.

Filter

Use the Filter transformation similar to how you would use the where clause in a SQL select statement. This transformation will filter out rows from a data stream that do not meet the condition that you specify.

Sort

Allows you to set the sort order for your data. Keep in mind that this is a “blocking” transformation, which means that Synapse must collect all of the data partitions to produce a final sorted view of your data, which will impact the time it takes to process your data. Sort is useful when testing your data, but is not generally used very often in big data scenarios. If you wish to sort data while debugging or building your data flows, use the built-in sorting from the data preview pane.

Alter Row

When writing data to a database sink, you will want to use an alter row transformation to set the appropriate database action flags. With the Alter Row transformation (figure 3-16), you’ll create policies using the expression builder for insert, update, delete, and upsert (aka merge) options for your data to define what will happen when Synapse writes data to your database. In the sink, you’ll choose the appropriate insert, update, delete, and upsert actions and then choose key columns to define matching rows.

Graphical user interface, application

Description automatically generated

Figure 3-16. Alter row transformation

Assert

Use asserts as a way to easily at data quality and data validation checks to your data flow. The assert transformation allows you to add your own custom set of expectations and rules to flag rows as errors should they fail your assertions. You can create your own assert failure ID and failure descriptions that you can then output in the sink as a separate error handler (figure 3-17).

Graphical user interface, text, application

Description automatically generated

Figure 3-17. Assert transformation

Flowlets

Flowets allow you to take portions of your logic data flow graph and turn it into a reusable component that can be used in other data flows. In figure 3-18 you’ll see that I’ve turned on “multi-select” on the design surface and then drew a bounding box around a portion of my graph. You can provide a name for this Flowlet and re-use it another data flow. To add a Flowlet to your data flow, select the Flowlet transformation and then select your saved custom Flowlets from the list.

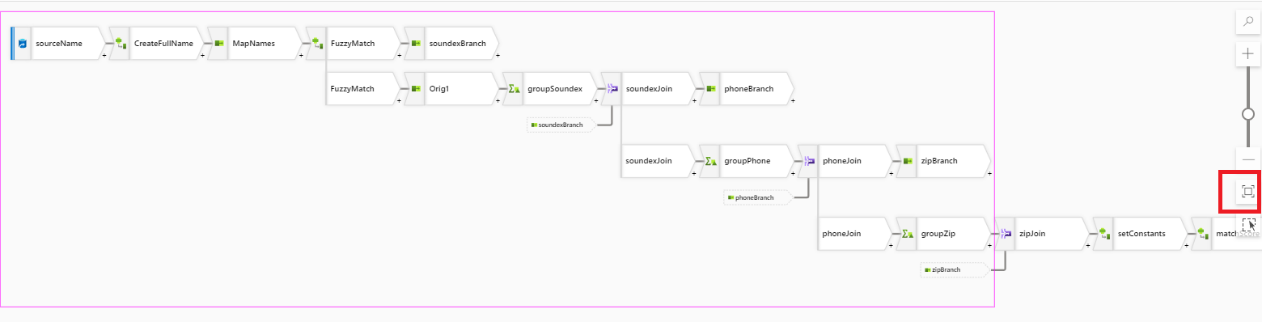


Figure 3-18. Select transformation to turn into a flowlet

Destination

Destinations are known as sinks in Synapse pipelines and data flows. Similar to the Source transformation, you can also select from 4 types of datasets (figure 3-19) that allow you to configure the data store where your processed data will land. The additional type that is in the sink but not in the source is the Cache sink. This destination type can store your results in memory in the Spark cluster. This is useful for cached lookups used throughout your data flow and can also be used to pass results back to the calling pipeline.

Graphical user interface, application

Description automatically generated

Figure 3-19. Sink destination data

You can also set error handling in the sink using the “Errors” tab to redirect rows that fail assertions or fail due to database driver errors.

Data flow expression builder

Similar to the pipeline builder experience, the Synapse data flows design surface includes an expression builder. The data flow expression builder provides full access to the data flow expression library as well as local variables, cached lookups, user-defined functions, parameters, and schema columns (figure 3-20).

Graphical user interface, text, application, table, Excel

Description automatically generated

Figure 3-20. Data flow expression builder

As you use the expression builder to build your transformation expressions, you’ll have access to the 100s of transformation functions available through Intellisense and auto-complete as well as from the “Functions” category under “Expression elements”. You can also create your own library of custom logic as “User defined functions”. You can manage your libraries from the Manage category in Synapse Studio (figure 3-21).

You can also manage parameters and local variables in the expression builder. All of these expression elements, including the metadata from your source data, are all part of the data flow metamodel, meaning that Intellisense will provide auto-complete for all of those elements as you type in your logic.

Graphical user interface, application

Description automatically generated

Figure 3-21. User defined functions are stored in Data flow libraries from the Manage section of Synapse Studio

Debug

While building your data flows, there are two primary ways in which you can debug and preview your work. In both of these scenarios, we’ll switch on the debug session from Synapse Studio, which will instantiate a Synapse Spark pool so that we can process our data flow graph and view the results in real time (figure 3-22).



Figure 3-22. Data flow debug session button

You do not need to have a debug cluster available while building your data flows. You can still view the metadata from the Inspect pane on each transformation and continue to build the rest of your logic. But switching on the debug session will provide the compute necessary to execute your data flow in real time, allowing you to debug and interactively view the data frames as you are working to validate and unit test your logic (figure 3-23). This preview data experience is also interactive, allowing you to sort the columns, move columns, and export your view to a CSV file for further exploration. You can also view descriptive statistics for each column by clicking on the “Statistics” button.

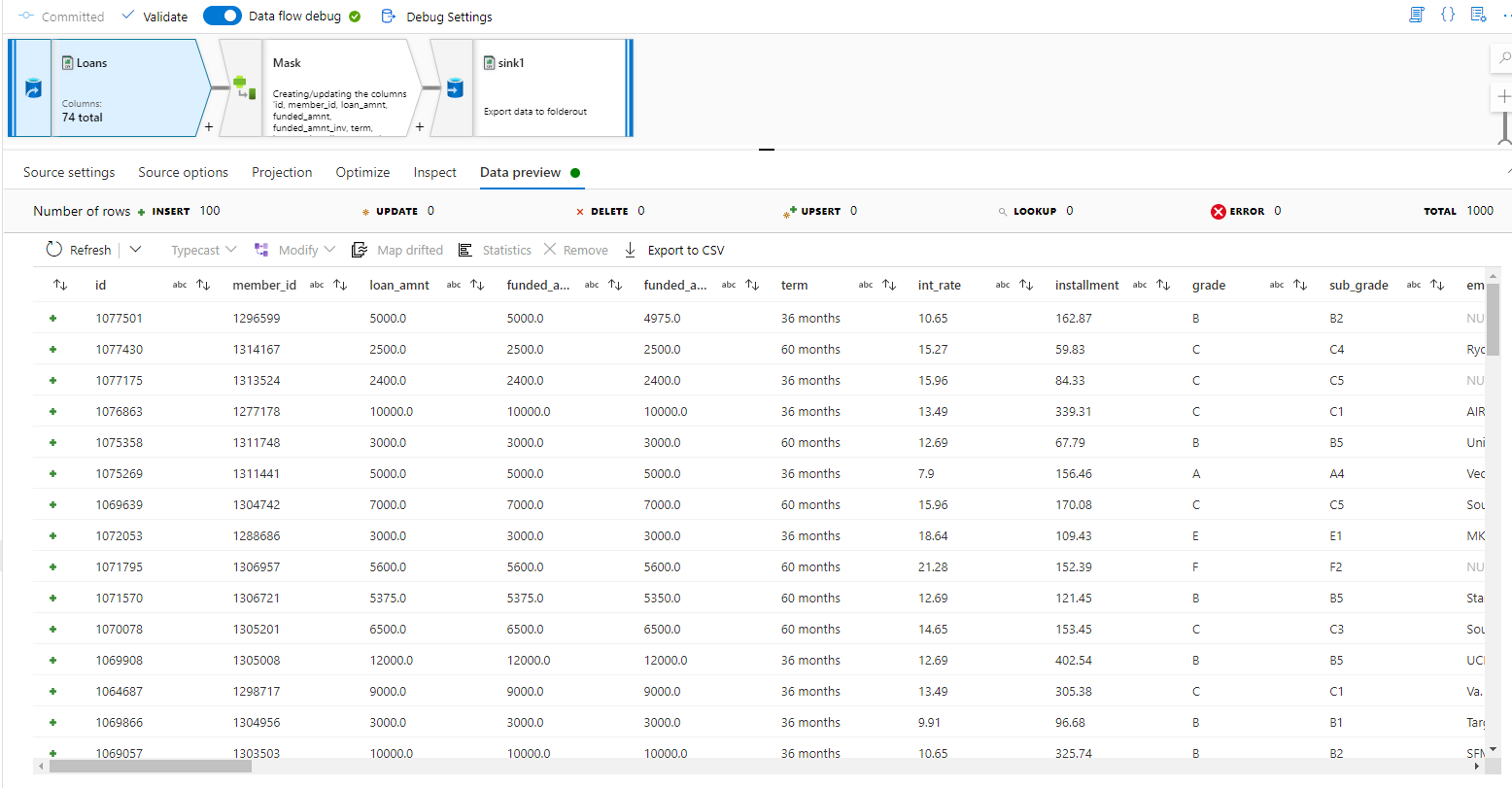


Figure 3-23. Data preview is available from every transformation

When you switch on the debug session, you will be prompted to choose an integration runtime and a time to live for your debug session (figure 3-24). For debugging and viewing preview data, the small auto resolve default IR is typically sufficient. Note that when you are viewing data preview in a debug session, no data is being written to the sink destinations. This is simply a peek at the results of each step in your logic being returned from the live Spark cluster via the Azure IR.

Graphical user interface

Description automatically generated

Figure 3-24. Select the Azure IR for your debug session

Once you are satisfied that you are seeing the expected results throughout your data flow, through to the sinks, you can now test the complete data flow end-to-end from a pipeline (figure 3-25). When you test your data flow from a pipeline, data will now get written to your sink destinations, which makes this a good time to test by viewing the final results in your target database, folder, file, etc. A good practice when testing your data flow is to set source sampling on from the source transformation so that you can test with smaller data for a quicker test. You can then switch off sampling at each source when you are done testing.

Graphical user interface, text, application

Description automatically generated

Figure 3-25. Execute data flows from a Synapse pipeline using the Data Flow activity

Operationalize data flows

Now that you have completed design, development, and testing of your data flow, you are ready to move out of design mode and set your data flow to run unattended inside of a Synapse pipeline. In chapter 2, we talked about adding triggers to your pipeline. You can now take the pipeline with the data flow activity that you built for testing above and add a trigger so that it can run unattended. Note that scheduling your pipeline will require you to publish the pipeline, which means that all of your Synapse resources must be validated first. The suggested mechanism for moving from development to testing and then to production is to integrate your workspace with Git so that you can implement a CI/CD pattern. We will discuss the details of the Git and deployment practices later in this book.

Azure Integration Runtime data flow settings

An important aspect of executing you data flow ETL jobs in a pipeline in an effective and efficient manner is to set the right data flow runtime using the Azure Integration Runtime (IR). We spoke about IRs in the previous pipelines-focused chapter. In the case of data flow activities in your pipeline, you can assign Azure IRs to your data flows that can provide varying levels of scale (figure 3-26). So far in this chapter, as we got started using data flows and designing our graphs, we used the out-of-the-box standard “auto-resolve” IR that uses a small size compute. That translates to a Spark cluster with 1 driver node of 4 cores and 1 worker node of 4 cores. If you have larger workloads, you can also choose from the preset VM sizes for your cluster of medium or large. Alternatively, choose “custom” and set your own VM node sizes from the available options. When starting out with Synapse data flows, the best approach is to start with the small IR size and then increase gradually from there if you are not meeting your performance expectations.

Graphical user interface, text, application, email

Description automatically generated

Figure 3-26. Integration runtime configuration for data flows

Summary

In this chapter, we dove into Synapse data flows from the Synapse Studio. In the previous chapter, we introduced data pipelines and we used pipelines in this chapter to execute our data flows. In both cases, we only talked about the primary features that you’ll mainly interact with and were not able to cover all features. But as you get started building big data analytics projects in Synapse, you should now have the fundamental understanding you need. For the rest of this book, we’re going to dive deeper into these data integration and ETL features to build sample scenarios and introduce you to other important areas of Synapse Analytics.