Microsoft

Azure Data Factory – Data Flows

https://aka.ms/dflinks

Krishna Golla
GinSiu Cheng
Patrick Gryczka

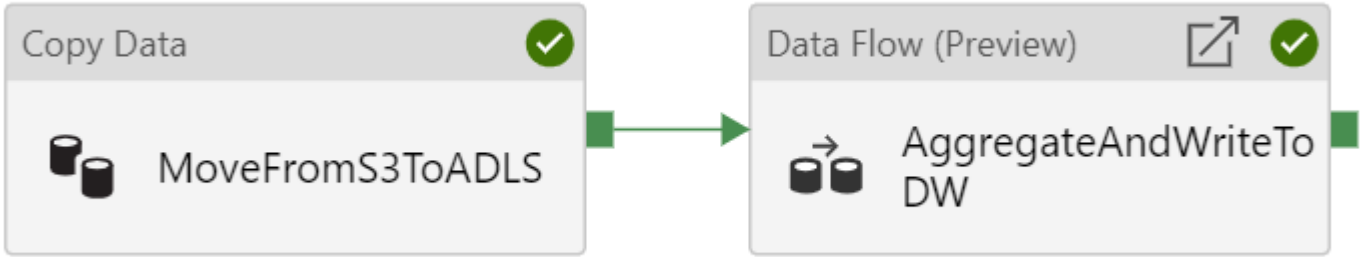
Agenda

- Overview
  - Data Factory Components
  - What is Mapping Data Flow
  - Modern data warehousing patterns
- Authoring and designing
  - Creation
  - Transformation overview
  - Expression language and visual expression builder
  - Debug mode and data preview
  - Schema drift and pattern matching
  - Data Flow parameterization
- Operationalizing
  - Data Flow activity
  - Data flow integration runtime
  - Pipeline Debug
  - Monitoring
  - Best practices
- Resources

Data Factory Components

Pipeline

- Logical grouping of activities that performs a unit of work
  - What gets run by a data factory
  - Activities can be run sequentially or in parallel
- Data factory can have one or more pipelines
- Ex: a pipeline can ingest data from S3 to ADLS (Copy activity), and then on success run a Data Flow transforming the data and writing it to a SQL DB (Data Flow activity)

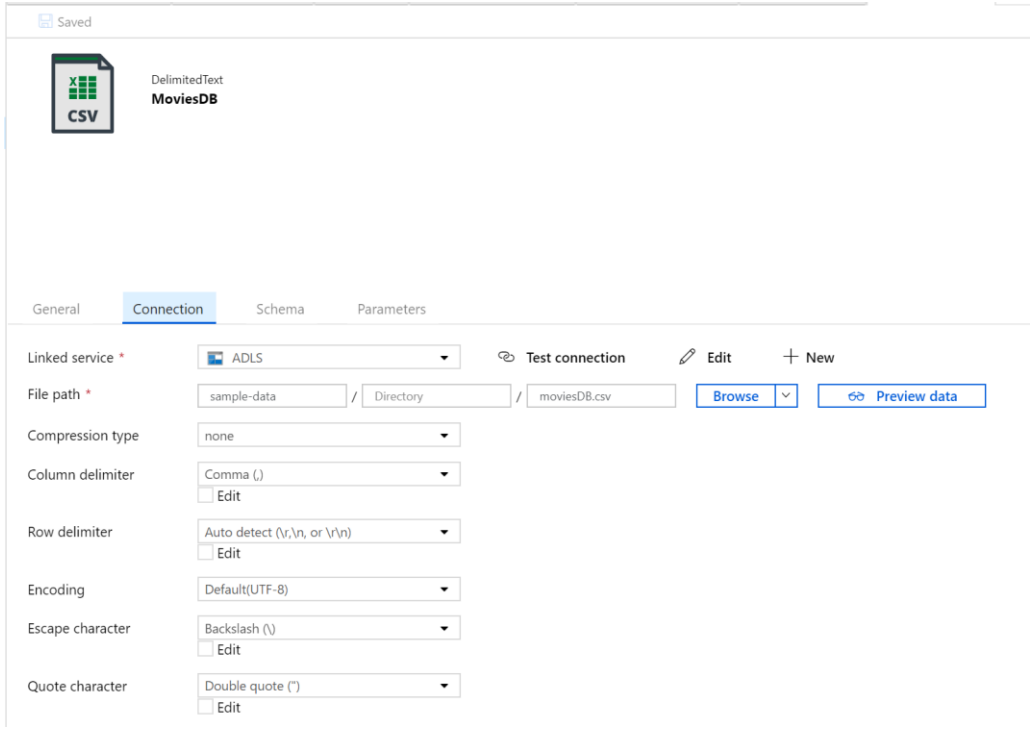


Activity

- A processing step in a pipeline
  - Ex: Copy activity to copy data from one data store to another data store
  - Ex: Hive activity to run a Hive query on an Azure HDInsight cluster, to transform or analyze your data.
- Three types of activities:
  - Data movement (Copy and Data Flow)
  - Data transformation (Data Flow, Notebook, HDInsight, etc)
  - Control flow (Lookup, For Each, Get Metadata, etc)

Datasets

- Data structures within the data stores (connectors)
- Point to or reference the data you want to use in your activities as inputs or outputs
  - Physical representation of the data in the store
- Associated with a linked service
  - Many datasets can point to single linked service
- Referenced by activities
  - Ex: Copy has source/sink datasets

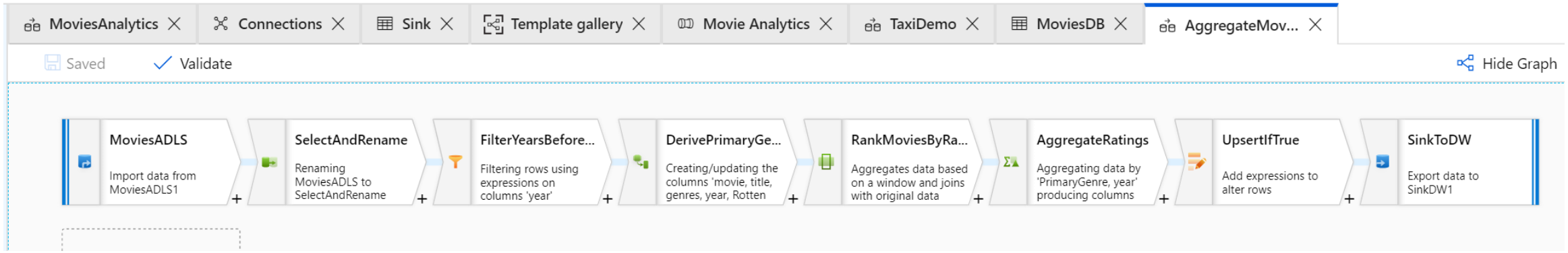


Linked Services

- Define the connection to a data source
- The connection information that's needed for Data Factory to connect to external resources
  - Ex: Azure Blob Storage linked service uses connection string stored in Key Vault (another linked service) to connect to the storage account
- 80+ connectors
- Two types:
  - Represent a **data store**
    - Ex: on-premises SQL Server database, Oracle database, file share, Azure blob storage account, etc
    - See list of connectors
  - Represent a **compute resource** that can host the execution of an activity.
    - Ex: the HDInsightHive activity runs on an HDInsight Hadoop cluster
    - See full list of compute environments data

Data Flows

- Visually-designed data transformation run on spark
- Ran from Data Flow activity in pipeline
- Comprised of transformations
- Source and Sink reference datasets

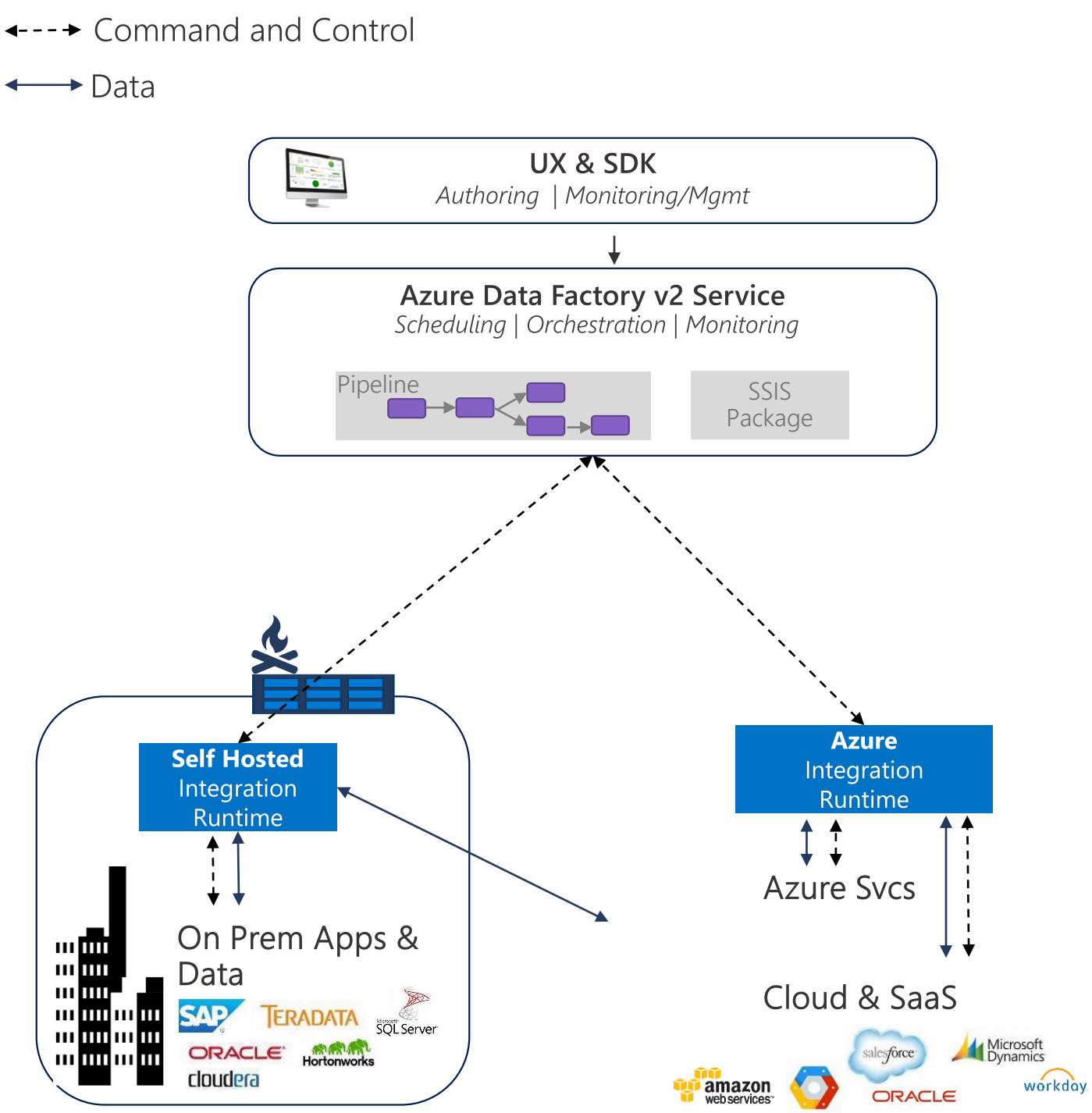


Triggers

- Unit of processing that determines when a pipeline execution needs to be kicked of
  - One trigger can have multiple pipelines associated
  - One pipeline can be associated with multiple triggers
- 3 types:
  - Schedule
  - Tumbling Window
  - Event-Based
- Ex: Schedule trigger runs a data ingest pipeline at 0:00 UTC everyday

Integration Runtime

- Compute infrastructure used by Azure Data Factory to provide the following data integration capabilities across different network environments:
  - **Data Flow**: Execute a Data Flow in managed Azure compute environment.
  - **Data movement**: Copy data across data stores in public network and data stores in private network (on-premises or virtual private network). It provides support for built-in connectors, format conversion, column mapping, and performant and scalable data transfer.
  - **Activity dispatch**: Dispatch and monitor transformation activities running on a variety of compute services such as Azure Databricks, Azure HDInsight, Azure Machine Learning, Azure SQL Database, SQL Server, and more.
  - **SSIS package execution**: Natively execute SQL Server Integration Services (SSIS) packages in a managed Azure compute environment.



Data Factory

A data integration account.

Location of orchestration, service metadata

Integration Runtime (IR)

ADF's execution engine

Three core capabilities:

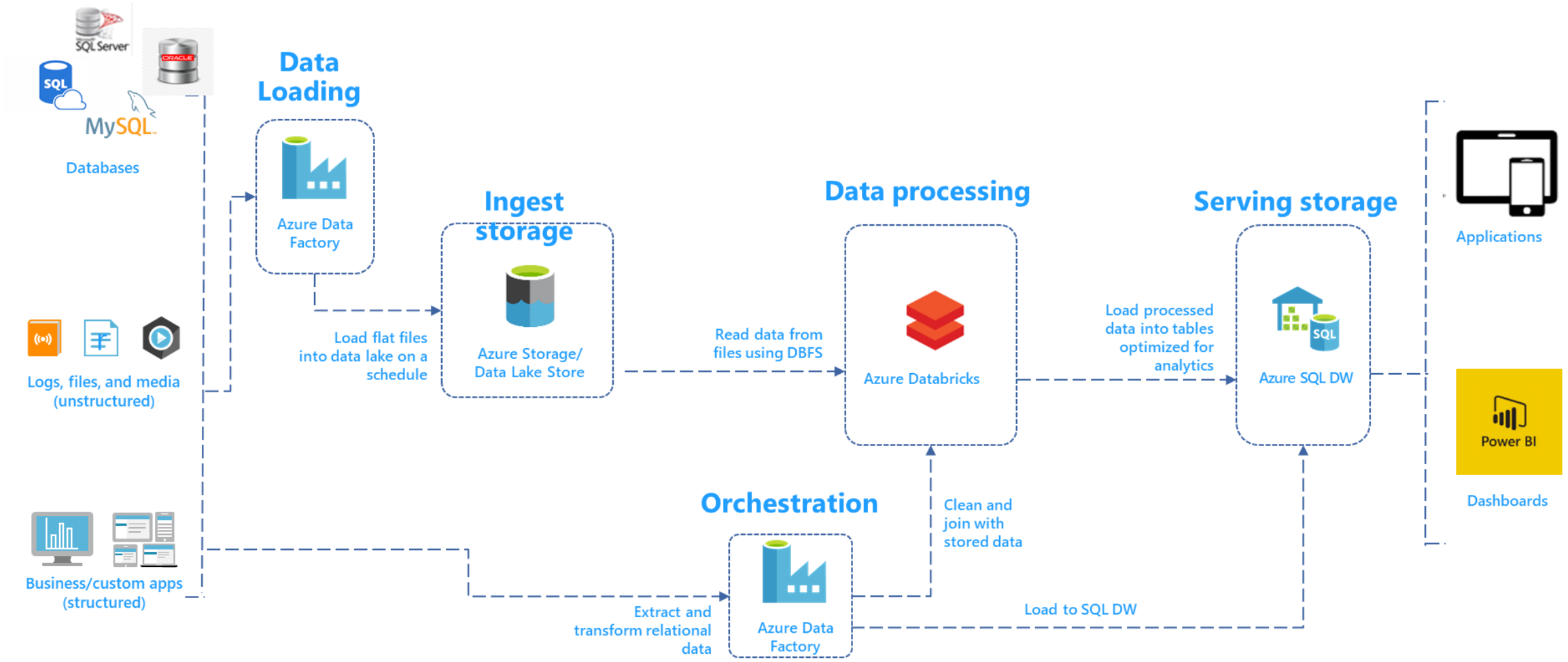
- data movement
- pipeline activity execution
- SSIS package execution

Mapping Data Flows Overview

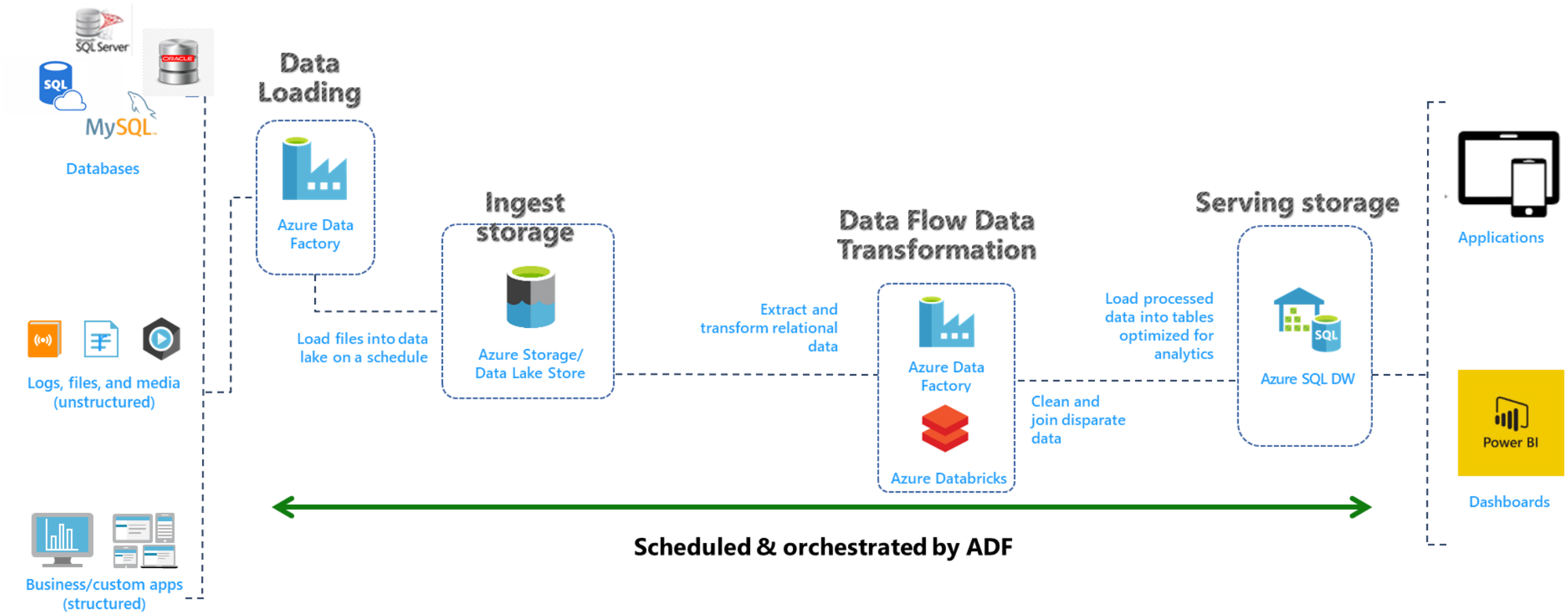
Code-free ETL/ELT @Scale

- Data cleansing, transformation, aggregation, conversion, etc.
- Cloud scale via Spark execution
- Guided experience to easily build resilient data flows
- Monitor and Manage dataflows from a single pane of glass

Modern data warehouse pattern today

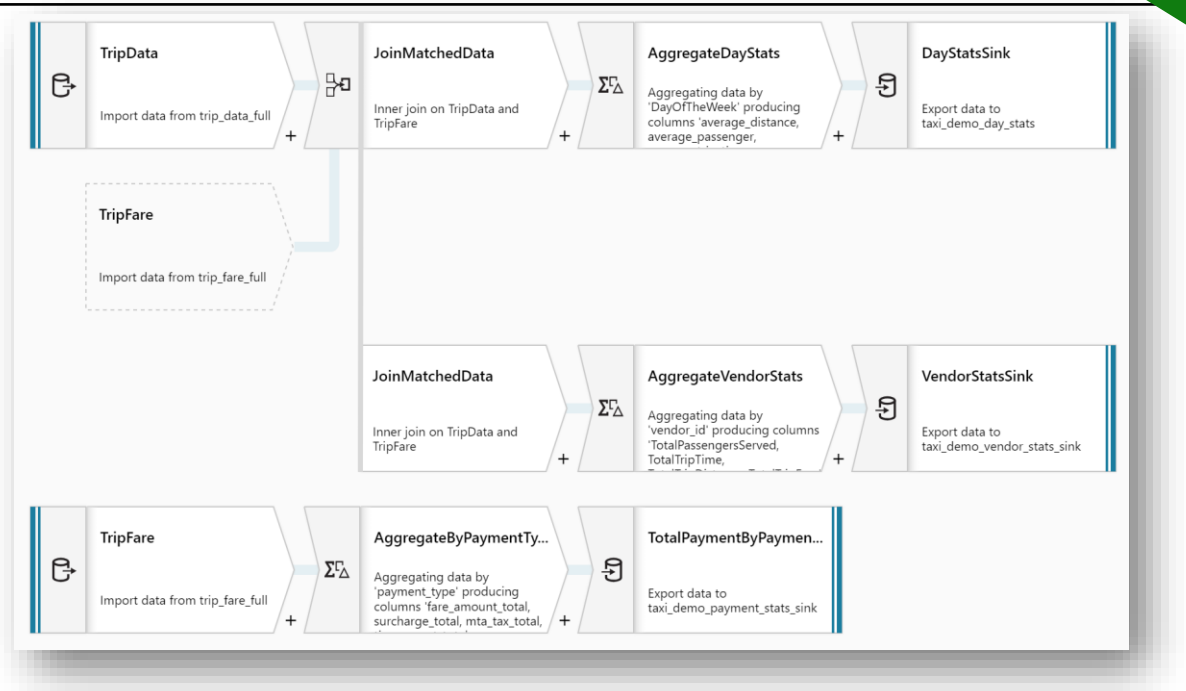


Modern data warehouse pattern with data flows

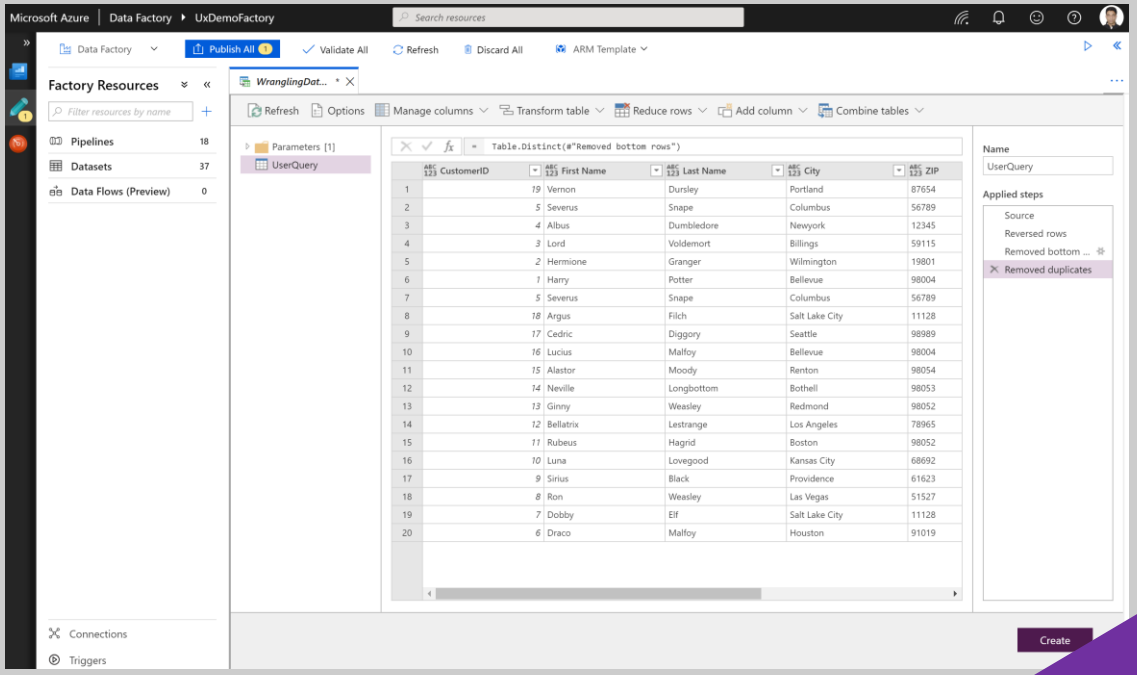


GENERALLY AVAILABLE

MAPPING DATAFLOW
Code-free data transformation @scale



WRANGLING DATAFLOW
Code-free data preparation @scale

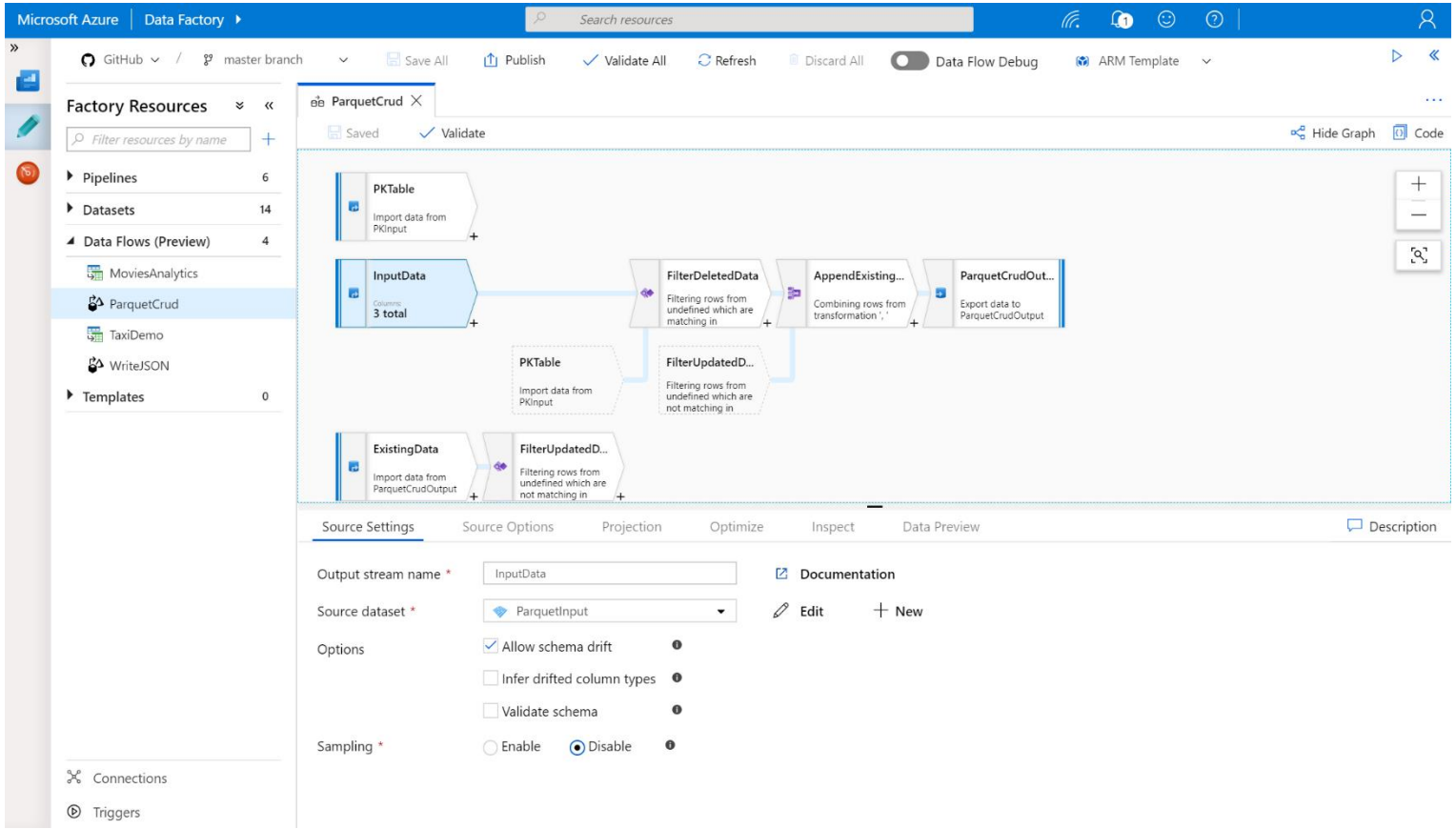


PUBLIC PREVIEW

What are Mapping Data Flows?

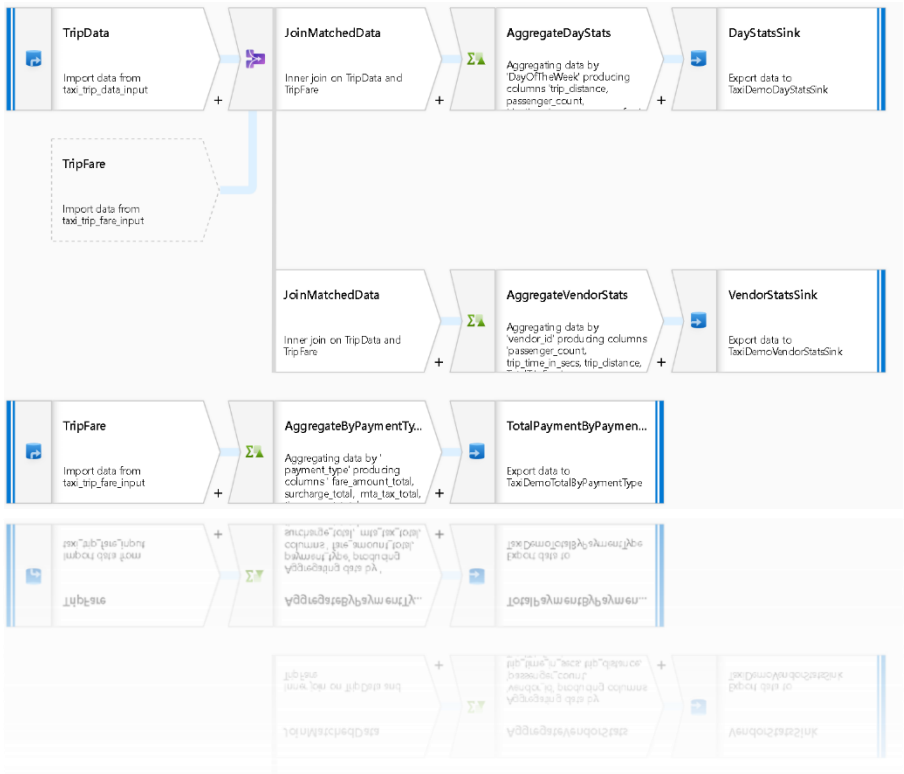
Mapping Data Flow is a new feature of Azure Data Factory that allows you to build data transformations code-free in a visual user interface

- Transform at scale, in the cloud
- Code-free pipelines do NOT require understanding of Spark / Scala / Python / Java
- Serverless scale-out transformation execution engine
- Resilient data transformation Flows built for big data scenarios with unstructured data requirements
- Operationalized with Data Factory scheduling, control flow and monitoring



Code-free data transformation at scale

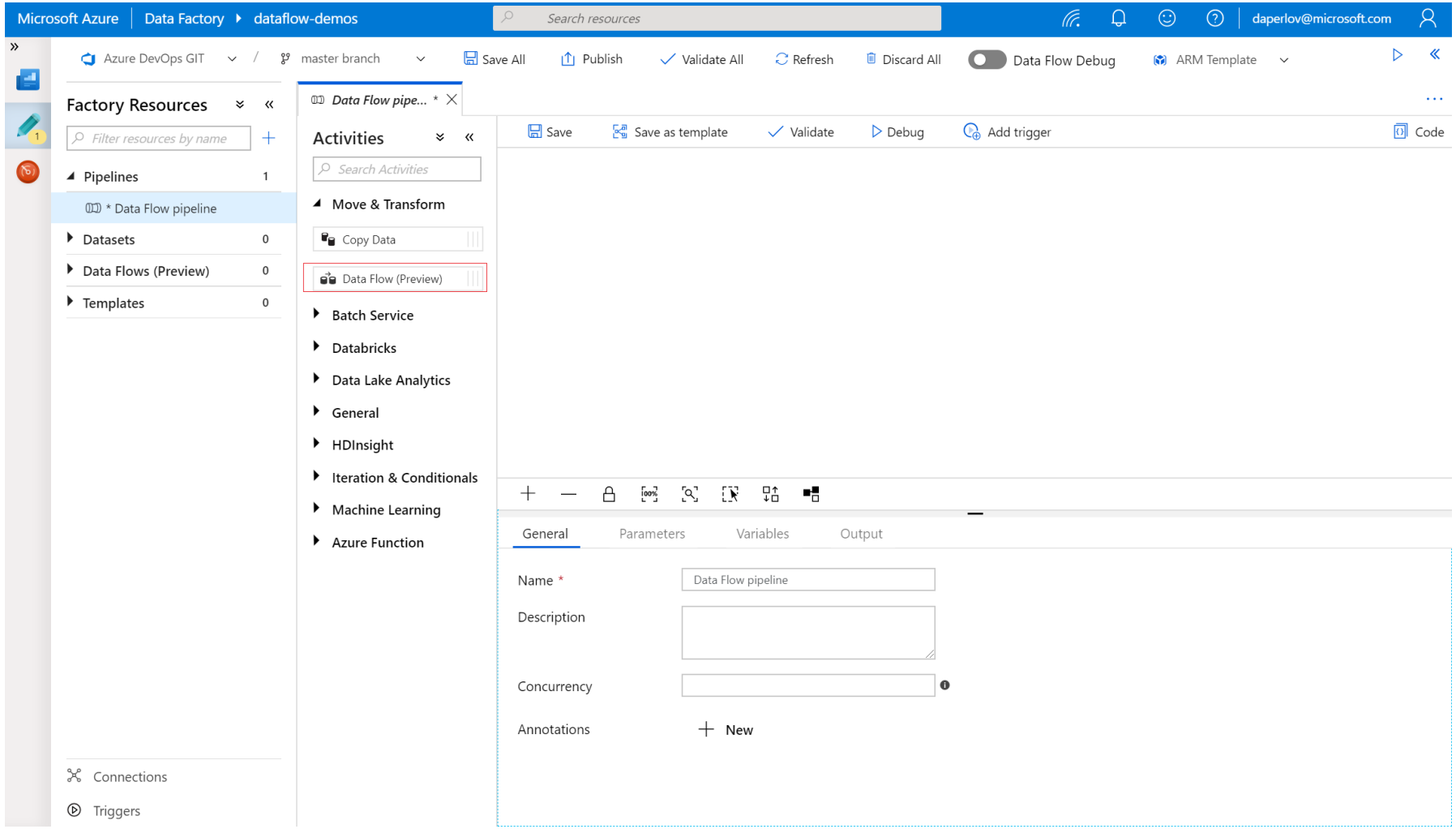
- Does not require understanding of Spark, big data execution engines, clusters, Scala, Python, etc
- Focus on building business logic and data transformation
  - Data cleansing
  - Data conversions
  - Data prep
  - Data exploration
  - Aggregation



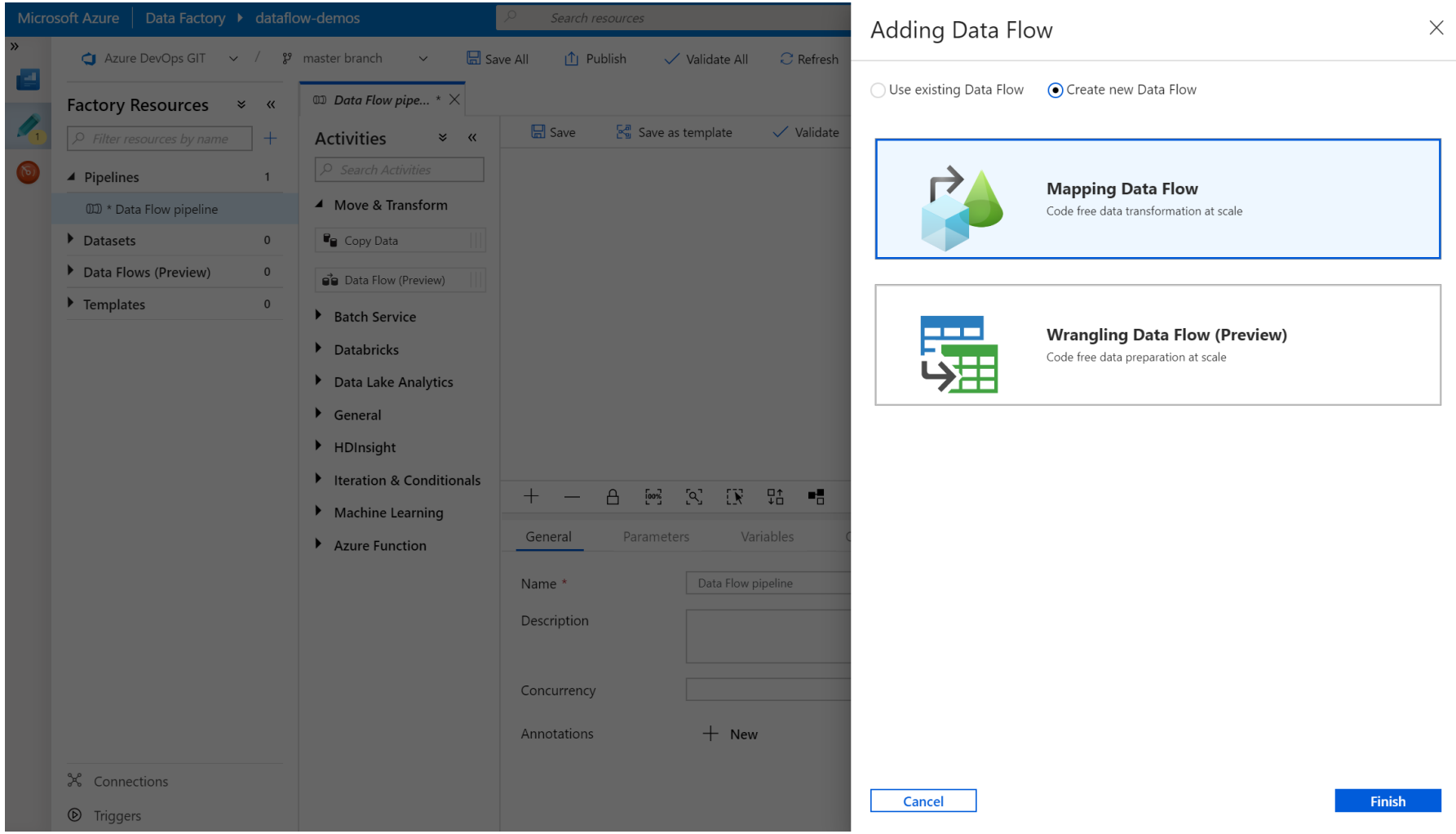
not …

Authoring and designing Mapping Data Flows

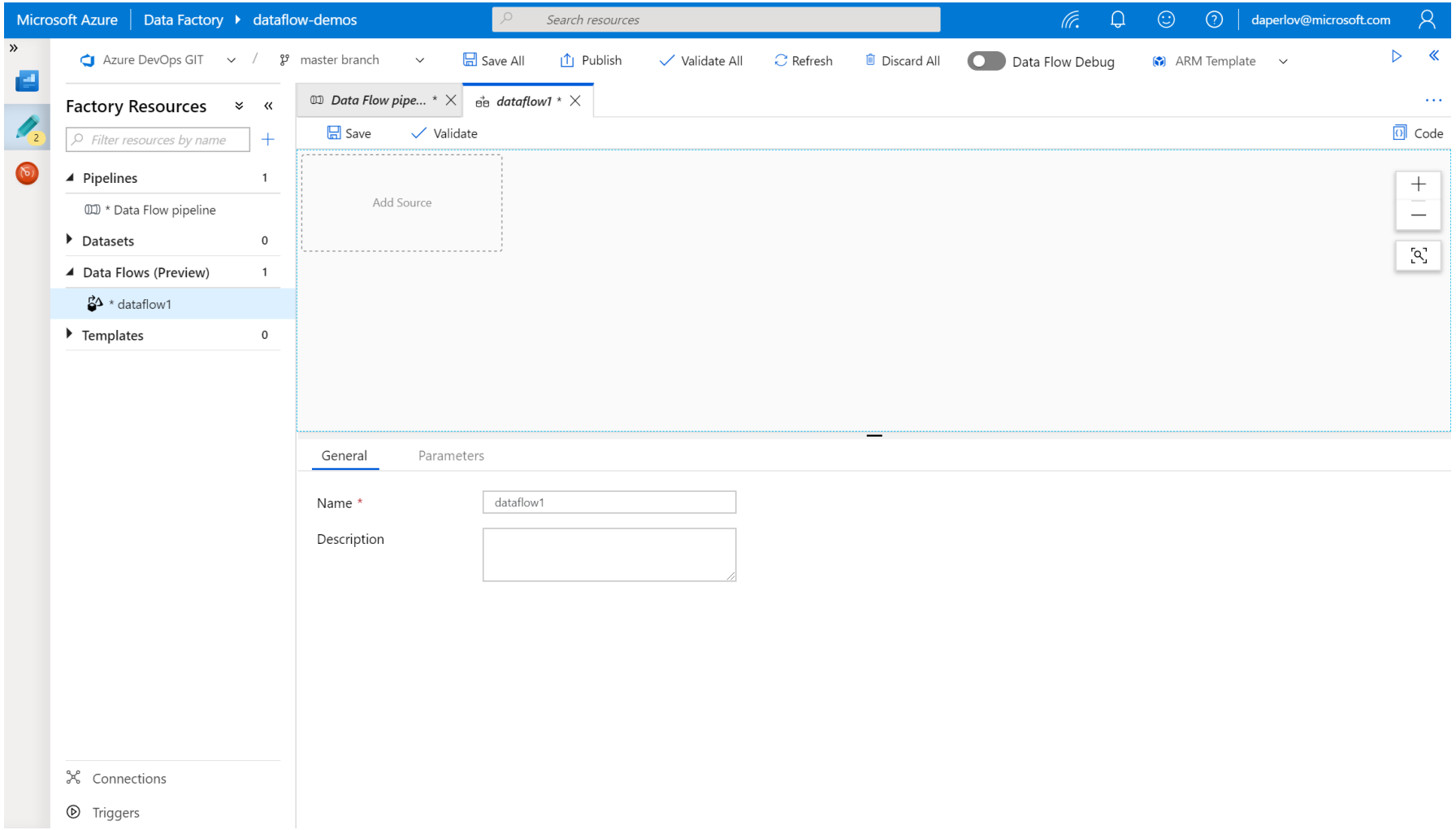
Creating a Mapping Data Flow



Creating a Mapping Data Flow



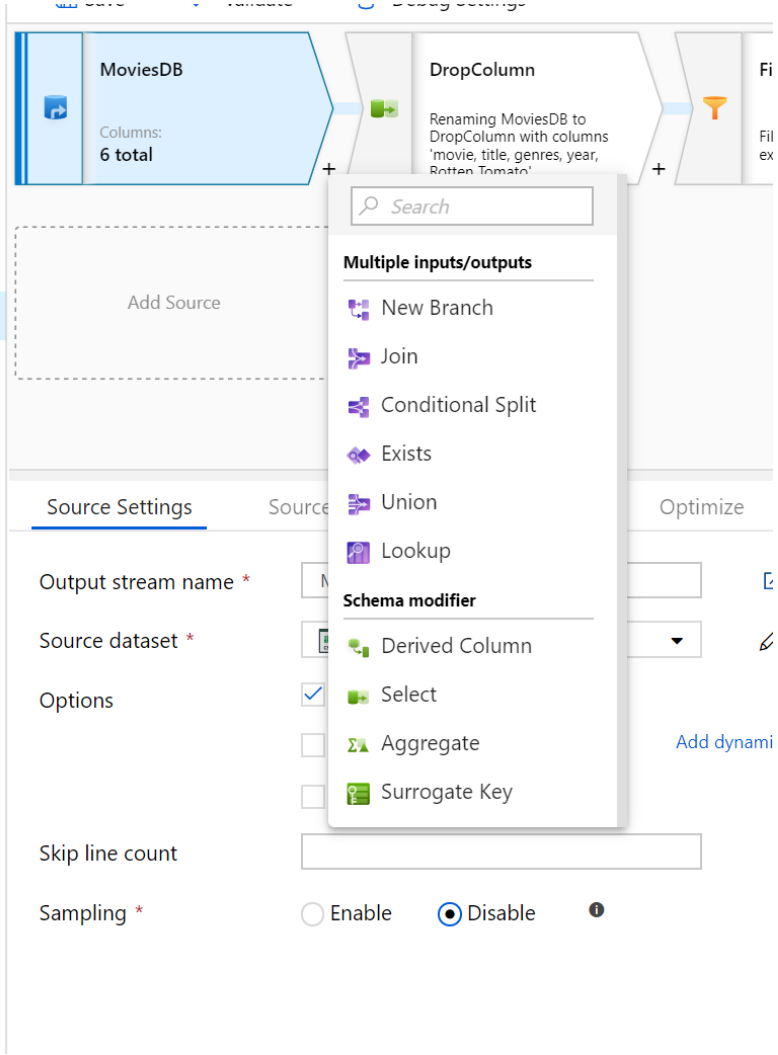
Creating a Mapping Data Flow



Mapping Data Flow Transformations

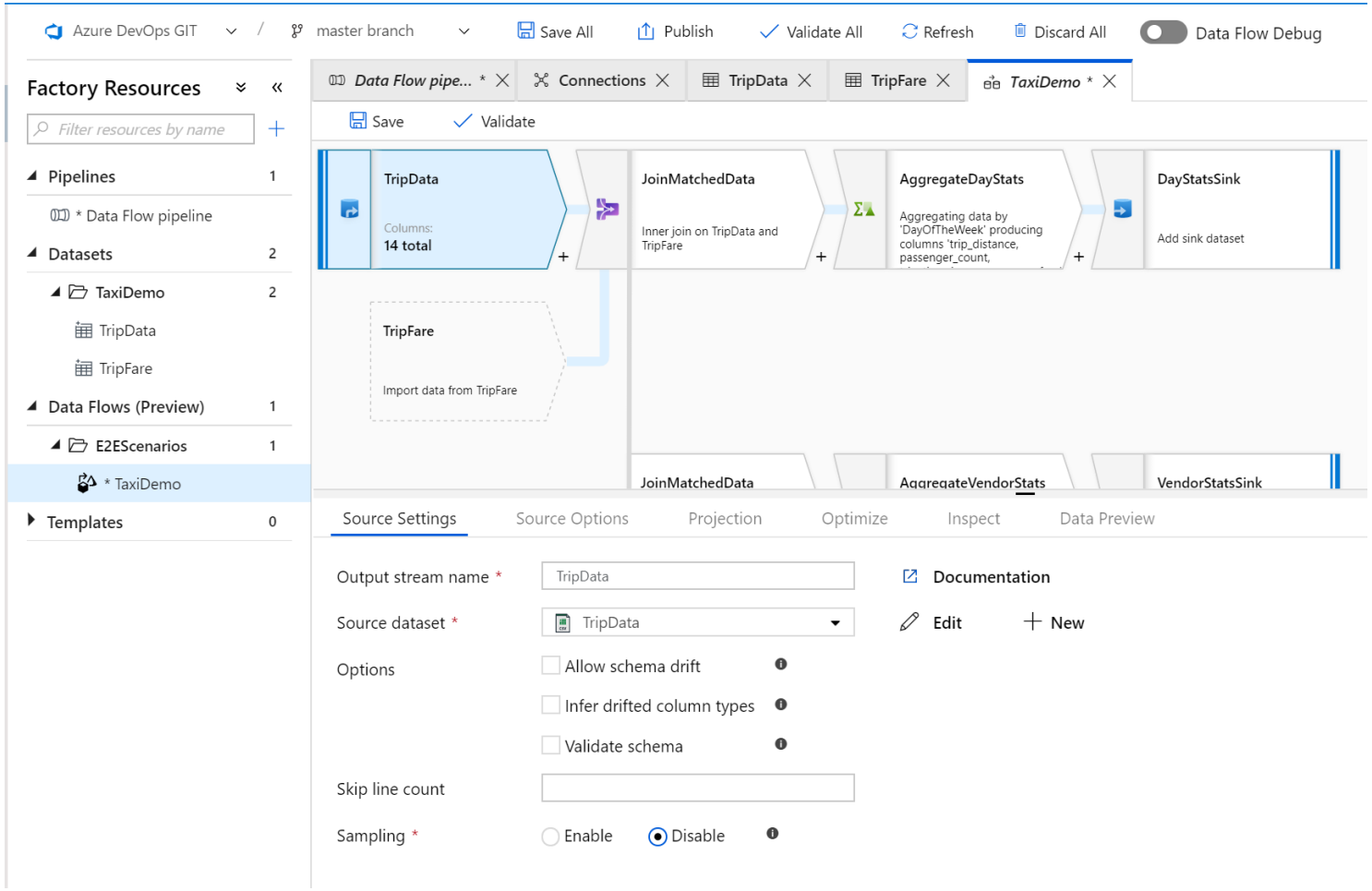
Transformations

- A ‘step’ in the data flow
  - Executor intelligently orders them at runtime
  - 18 currently available
- Core logic of data flow
  - Add/Remove/Alter Columns
  - Join or lookup data from datasets
  - Change number or order of rows
  - Aggregate data



Source Transformation

- Defines the data coming into your data flow
- Set projection, schema drift, data partitioning, actions after completion
- Min: 1, Max: ∞



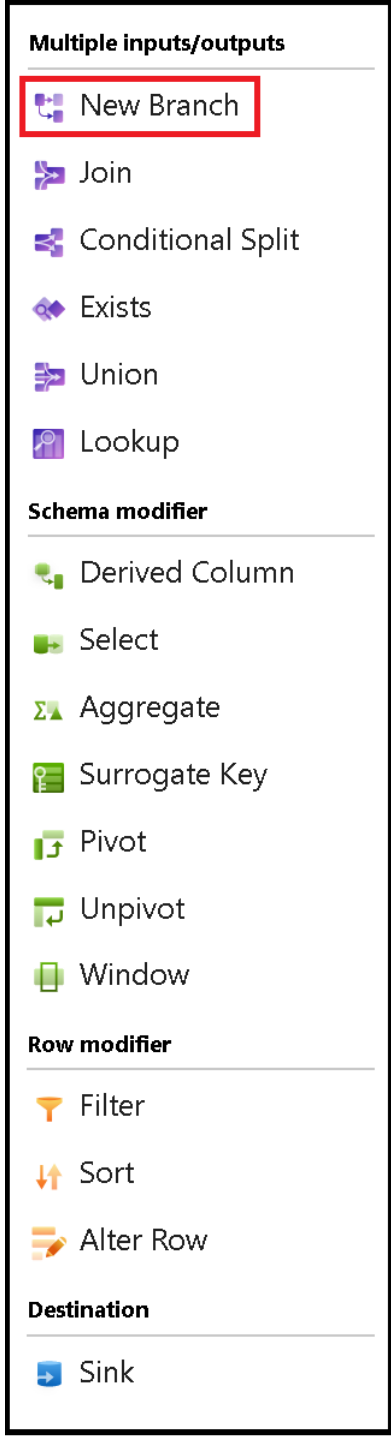
Supported Sources

- Azure Blob storage
- ADLS Gen1&2
- Azure SQL DB
- Azure SQL Datawarehouse
- CosmosDB – SQL API

- File Supported formats: CSV, JSON, Avro, Parquet

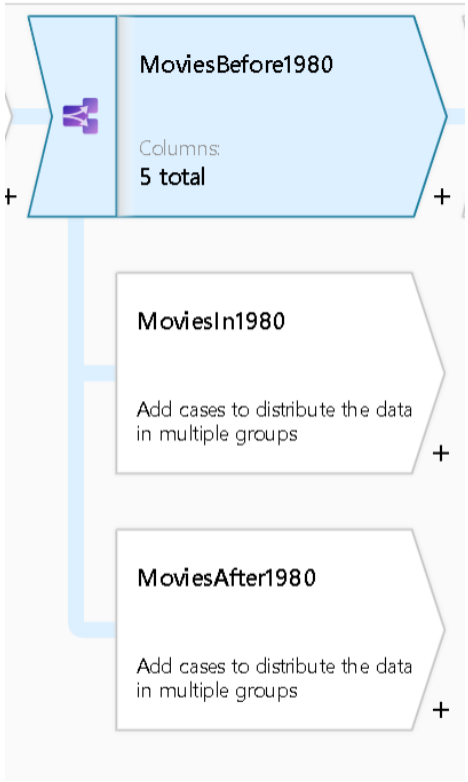
New Branch

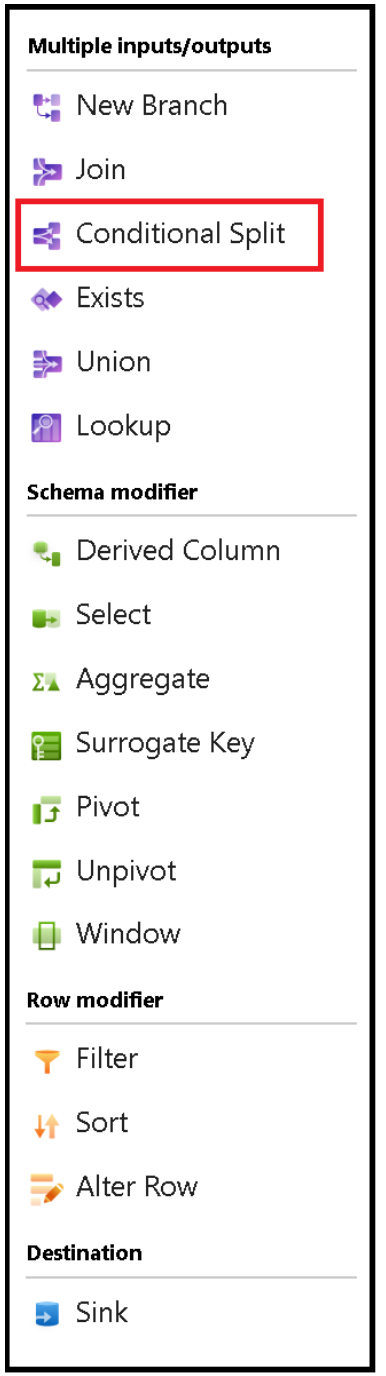
- Use to duplicate data streams from any stage in your data flow
- Useful for self-joins
- Useful for operating on the same data with different transformation requirements, i.e. Aggregate in one branch.



Conditional Split

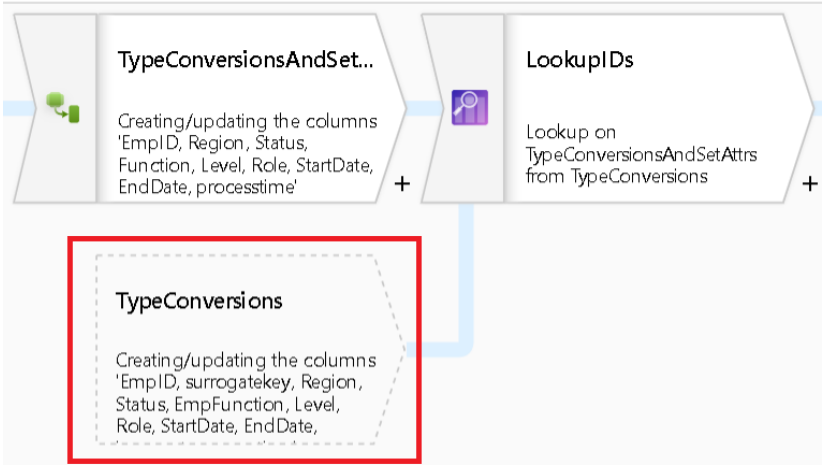
- Perform conditional transformation on your data
- Split data into separate streams based on conditions
- Useful when sinking data to different locations or different stores
- Useful when you need to perform different calculations on your data based upon a set of values

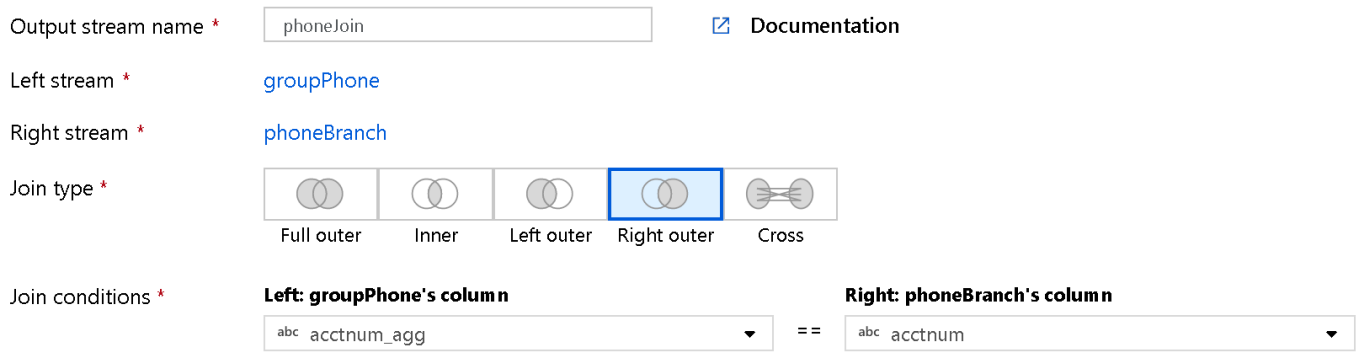


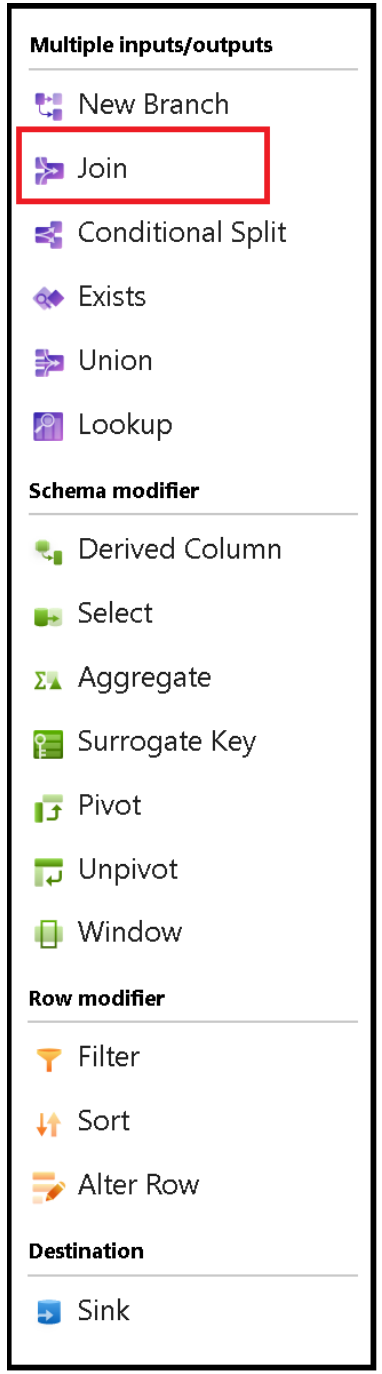


Join

- SQL Join equivalent
- Join incoming stream (left) to any other block in your data flow (right)
- ADF will draw a shadow node for you so you can see the right-side relation

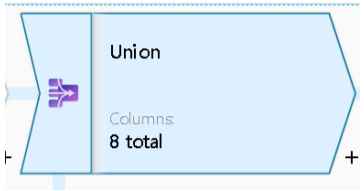


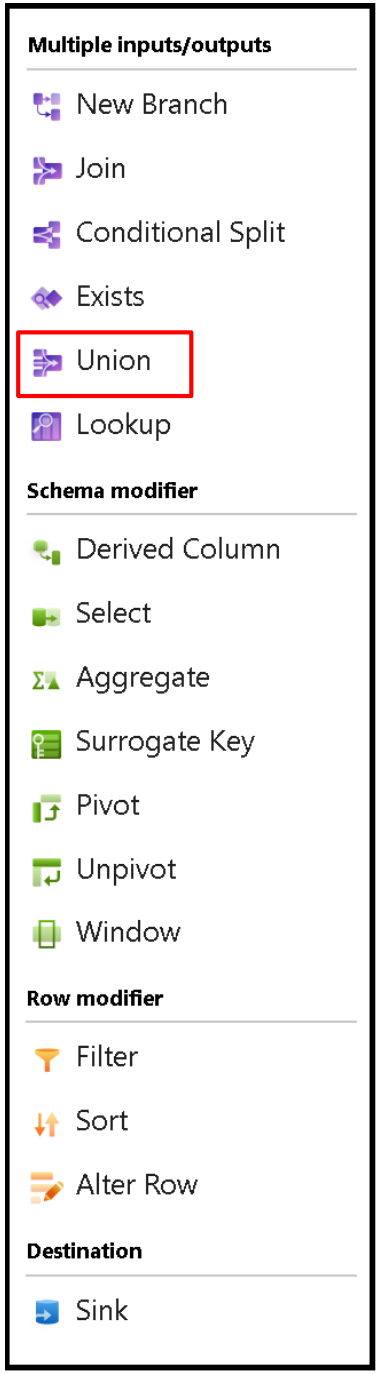




Union

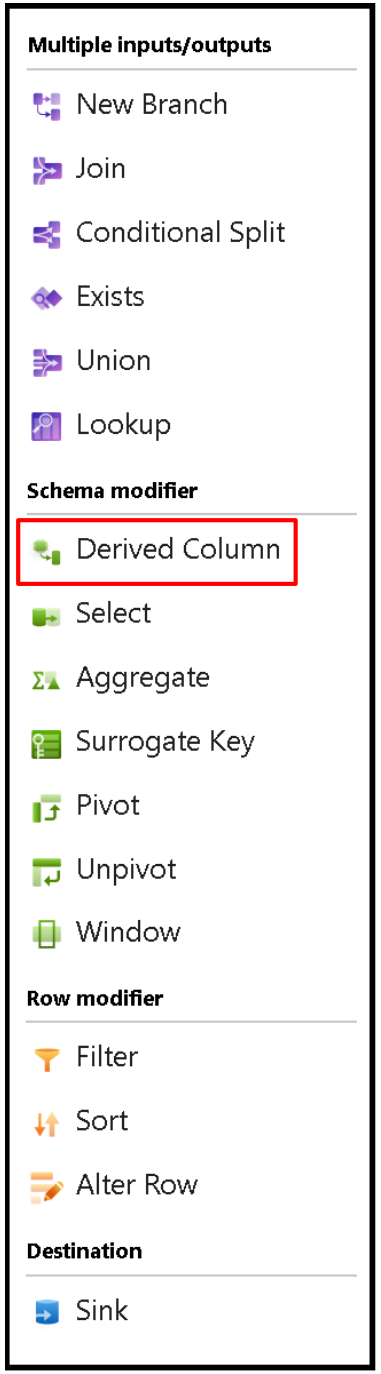
- Combine data from multiple streams
- Add as many streams as needed
- Combines data together either based by column name or column position
- Useful when you have a number of similar data sources that you wish to act upon with a series of transformations





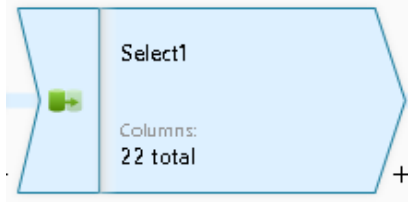
Derived Column

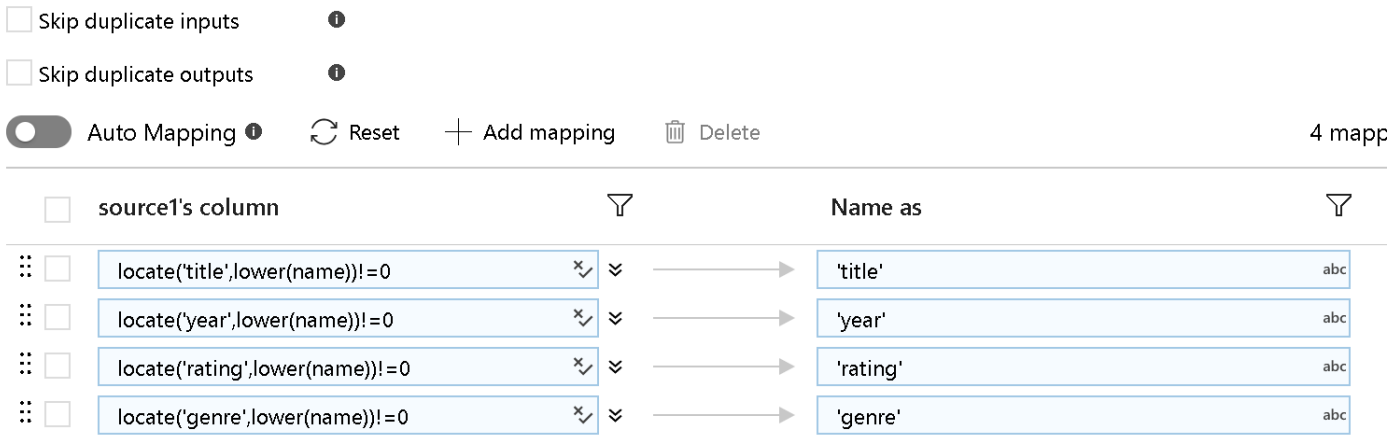
- Most heavily-used transformation
- Row transformations
- Used to generate new columns and new column values
- Modify existing columns and column values
- Transform data at row and column level using data flow expressions
- Support patterns and regular expressions for flexible transformations and schema drift

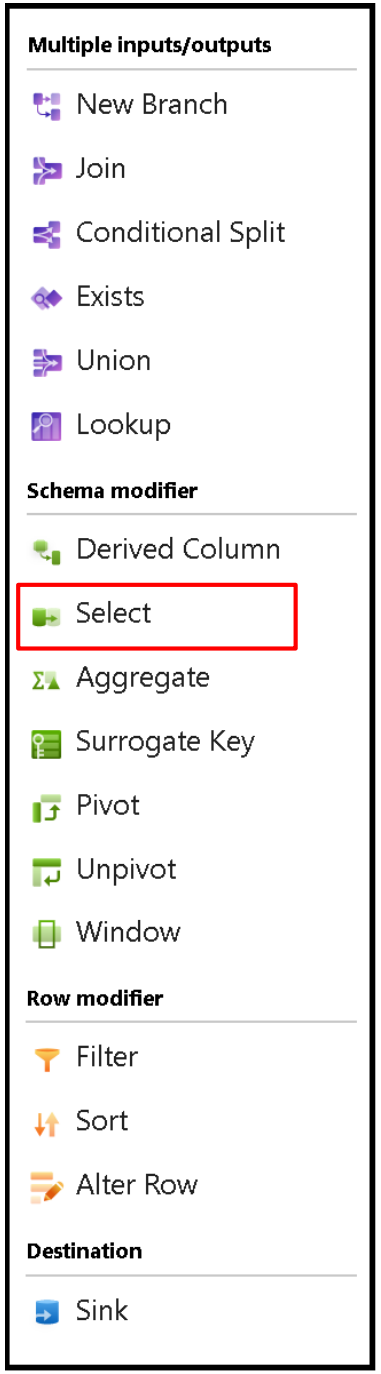


Select Transformation

- Use it for metadata / column maintenance
- Aliases your streams and columns
- Use it to prune your columns, keep only columns you're interested in
- Useful to generate canonical models for flexible schemas
- Use it to remove duplicate columns in your metadata (common after joins & lookups)

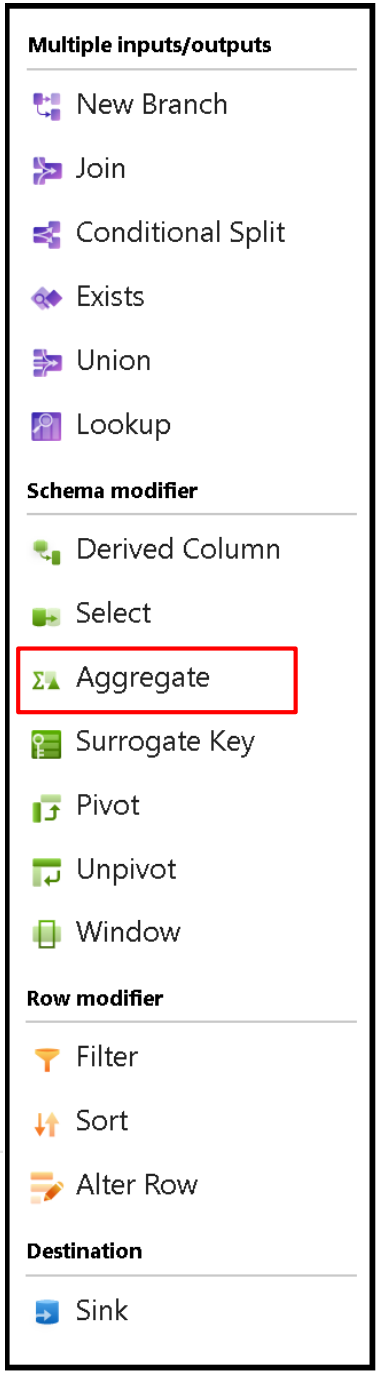


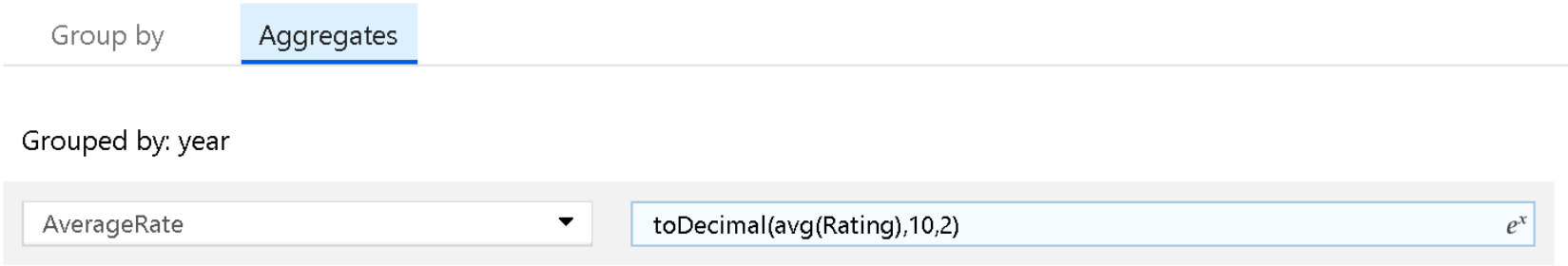




Aggregate

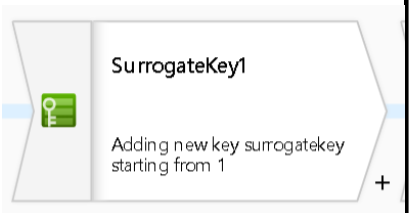
- Aggregates data based on the aggregate functions you choose in the expression language
- Similar to using SQL aggregation functions in a SELECT statement
- First, set your group-by columns
- Then, choose your aggregation functions
- Aggregate must read full data set to create aggregated output
- Only columns used in aggregate are returned via output

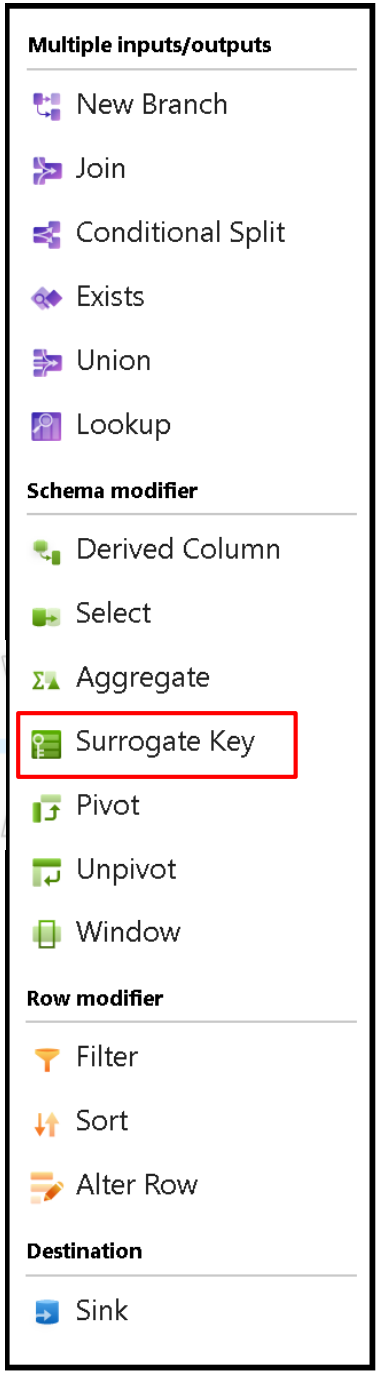




Surrogate Key

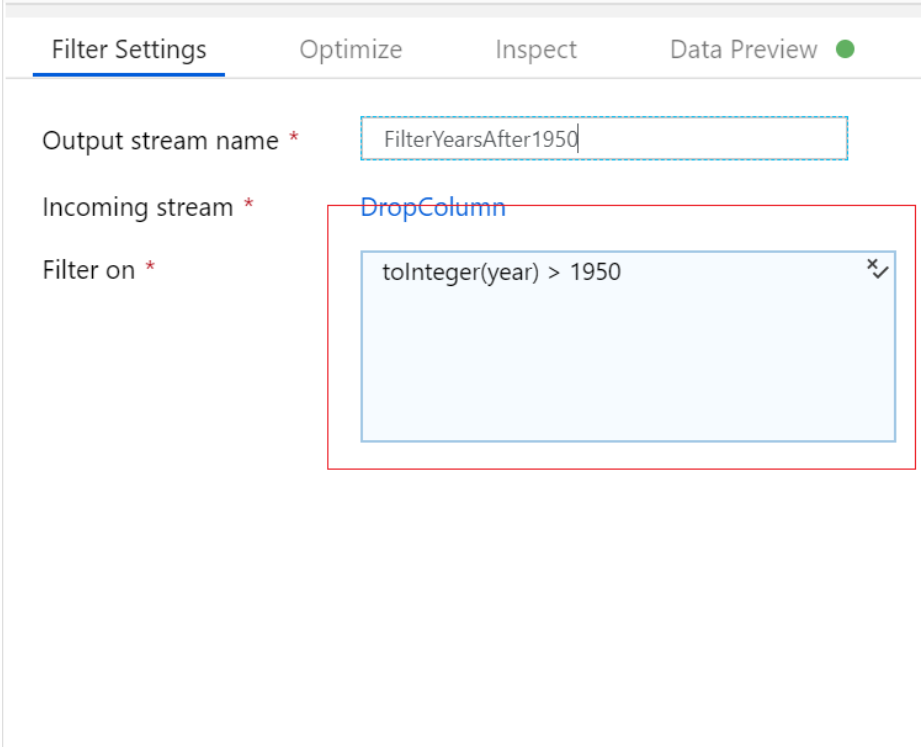
- Auto-generate incrementing key to use as a non-business (surrogate) key in your data
- To seed the starting value of your surrogate key, use a Derived Column and a lookup for MAX key value from your table
- Useful for generating keys for star schema dimension table members

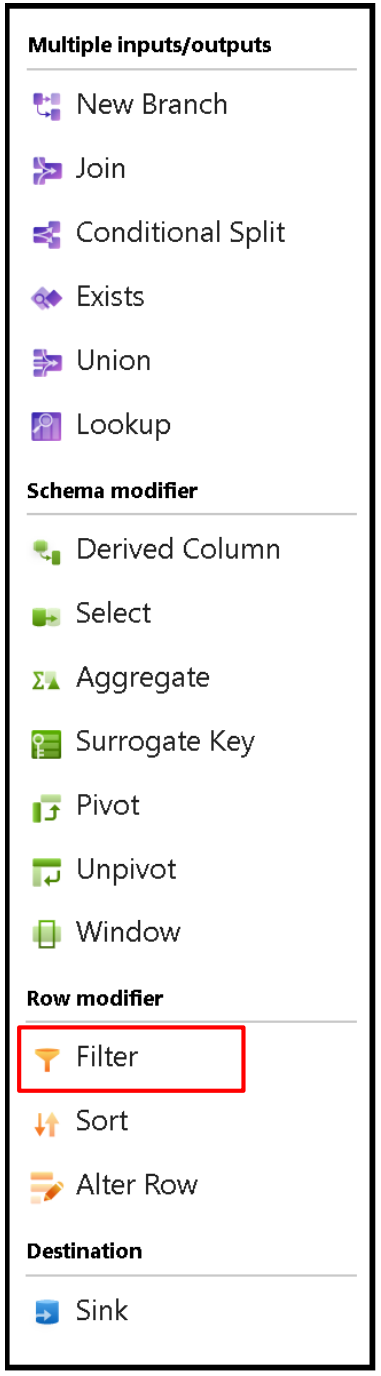




Filter Transformation

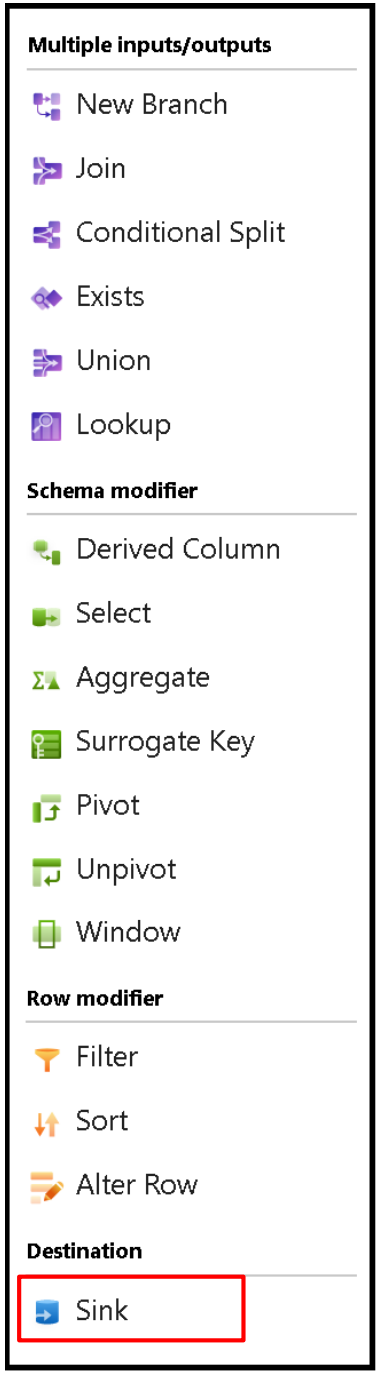
- Row filtering
- Acts like SQL WHERE clause
- Write expressions that return Boolean true/false
- Use data preview to view sample results





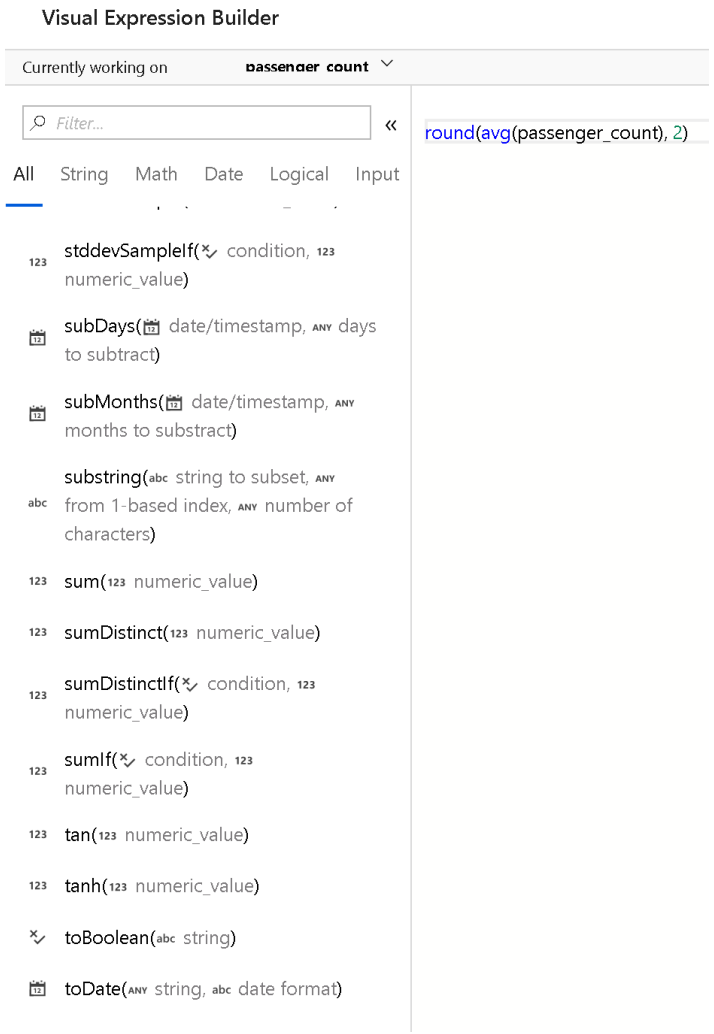
Sink Transformation

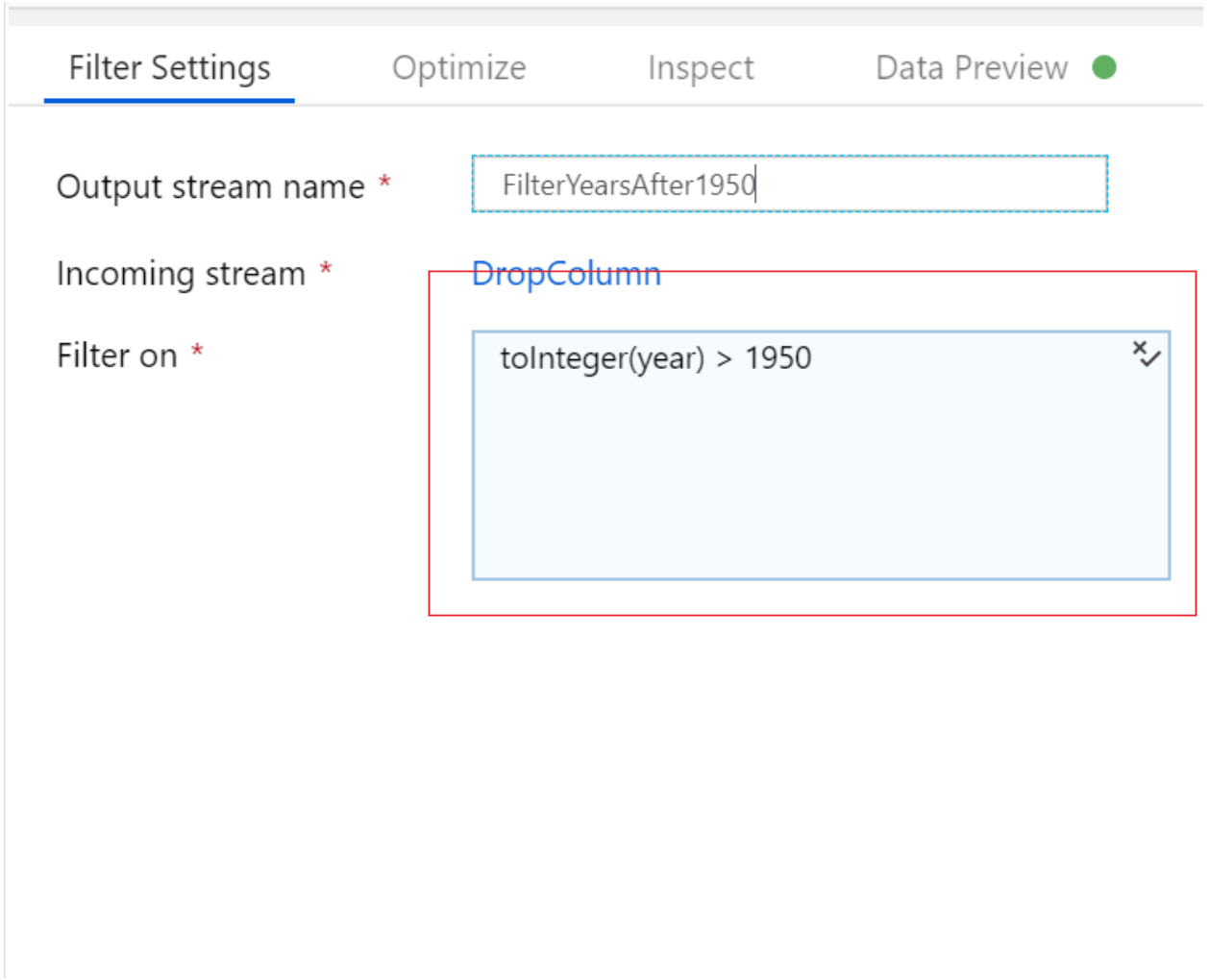
- This is where you will define the properties for landing your data in your destination target data store
- Pick the dataset to use that defines the shape of your data
- Use the Mapping tab to set the mapping automatically or manually
- You can use pattern matching rules to define your mappings here
- Choose a partitioning scheme to optimize data loading
- Set actions on your destinations, i.e. truncate table, clear folder, create table …
- Set pre/post actions for your SQL sinks
- Turn on / off schema drift
  - Schema drift on the Sink will instruct ADF to pass through all new fields to your destination



Mapping Data Flows Expressions

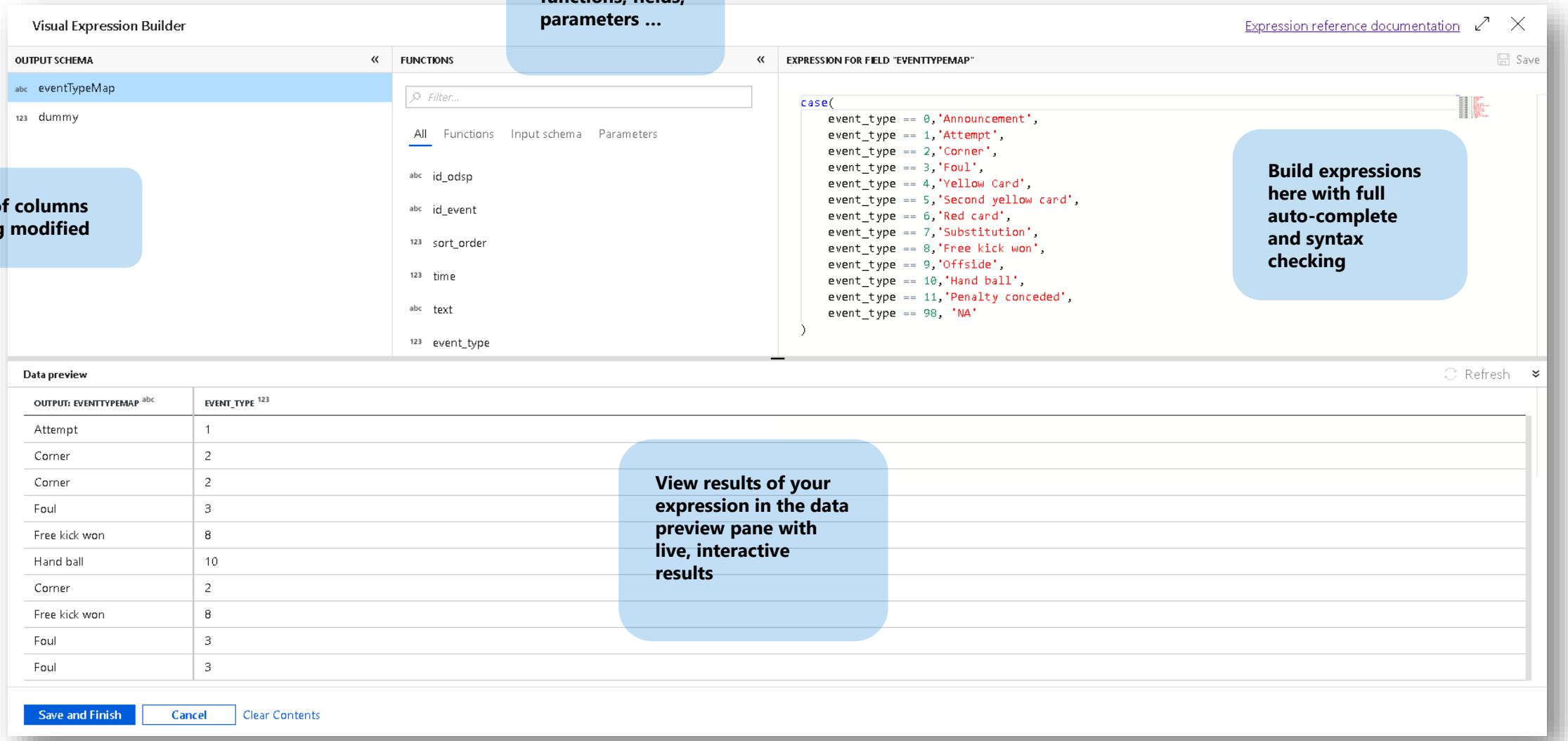
Visual Expression builder





Expression builder

All available functions, fields, parameters …



List of columns being modified

Build expressions here with full auto-complete and syntax checking

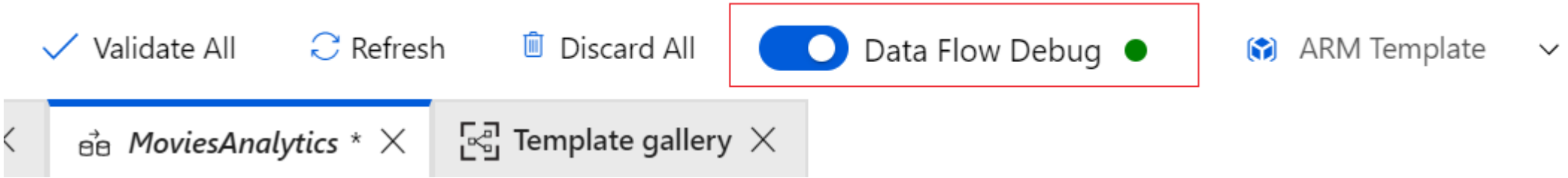
View results of your expression in the data preview pane with live, interactive results

Expression language

- Expressions can reference:
  - Expression functions listed here
  - Input schema columns
  - Drifted columns via byName function
  - Dataflow parameters
  - Literals
- Some transformations have unique functions
  - Aggregate (count, sum, etc), Window (rank, denserank), etc
- Evaluates to a spark data type

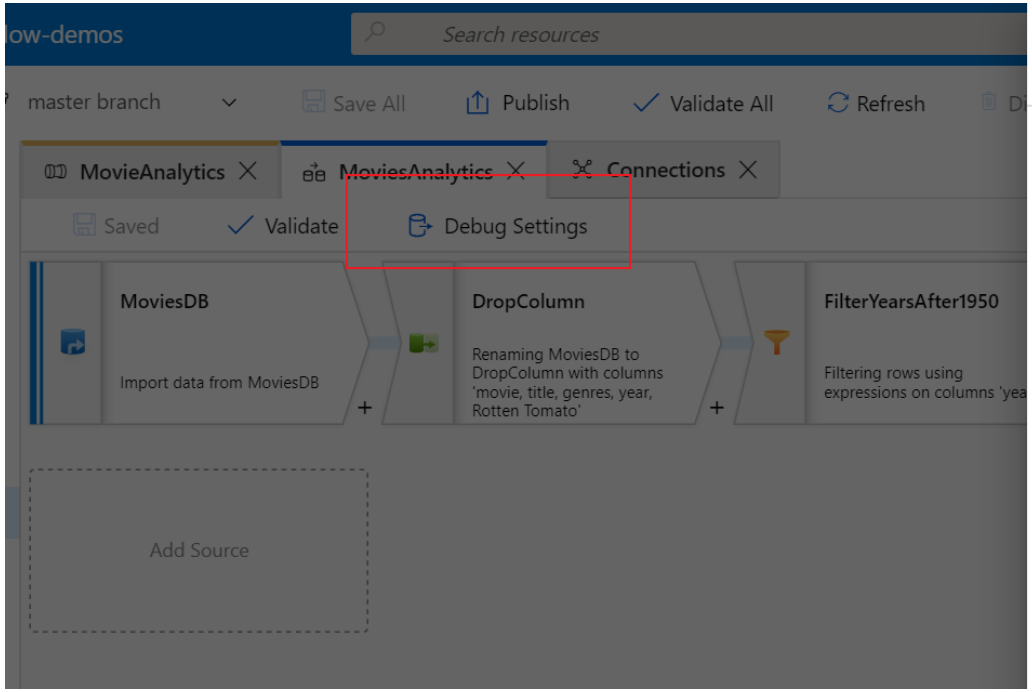
Debug mode

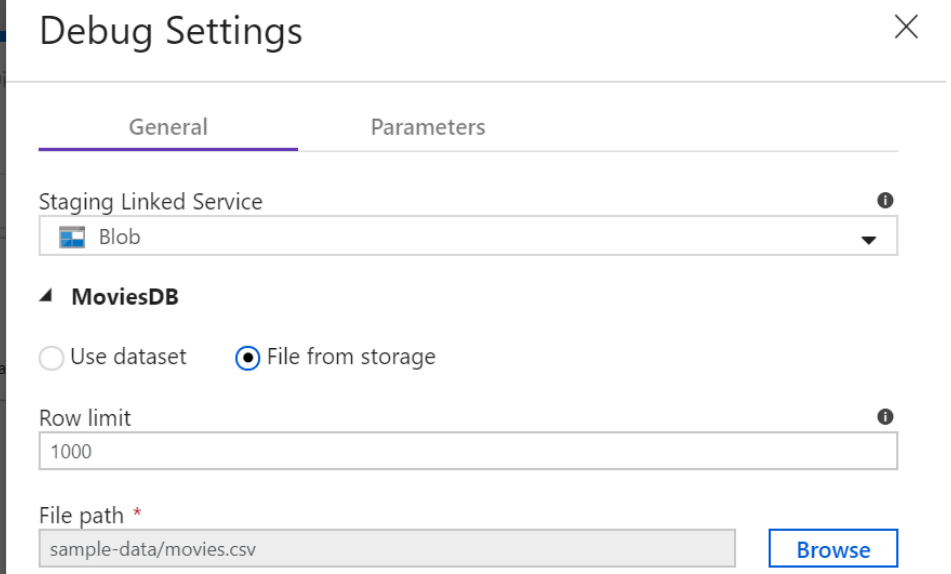
- Quickly verify logic during development on small interactive cluster
  - 4 core, 60-minute time to live
  - Get data preview snapshot at each transformation
- Preview output of expression in expression builder
- Run debug pipeline with no spin up
- Detect data types of DelimitedText source
- **Rule of thumb**: If developing Data Flows, turn on right away
  - Initial 5-minute start up time



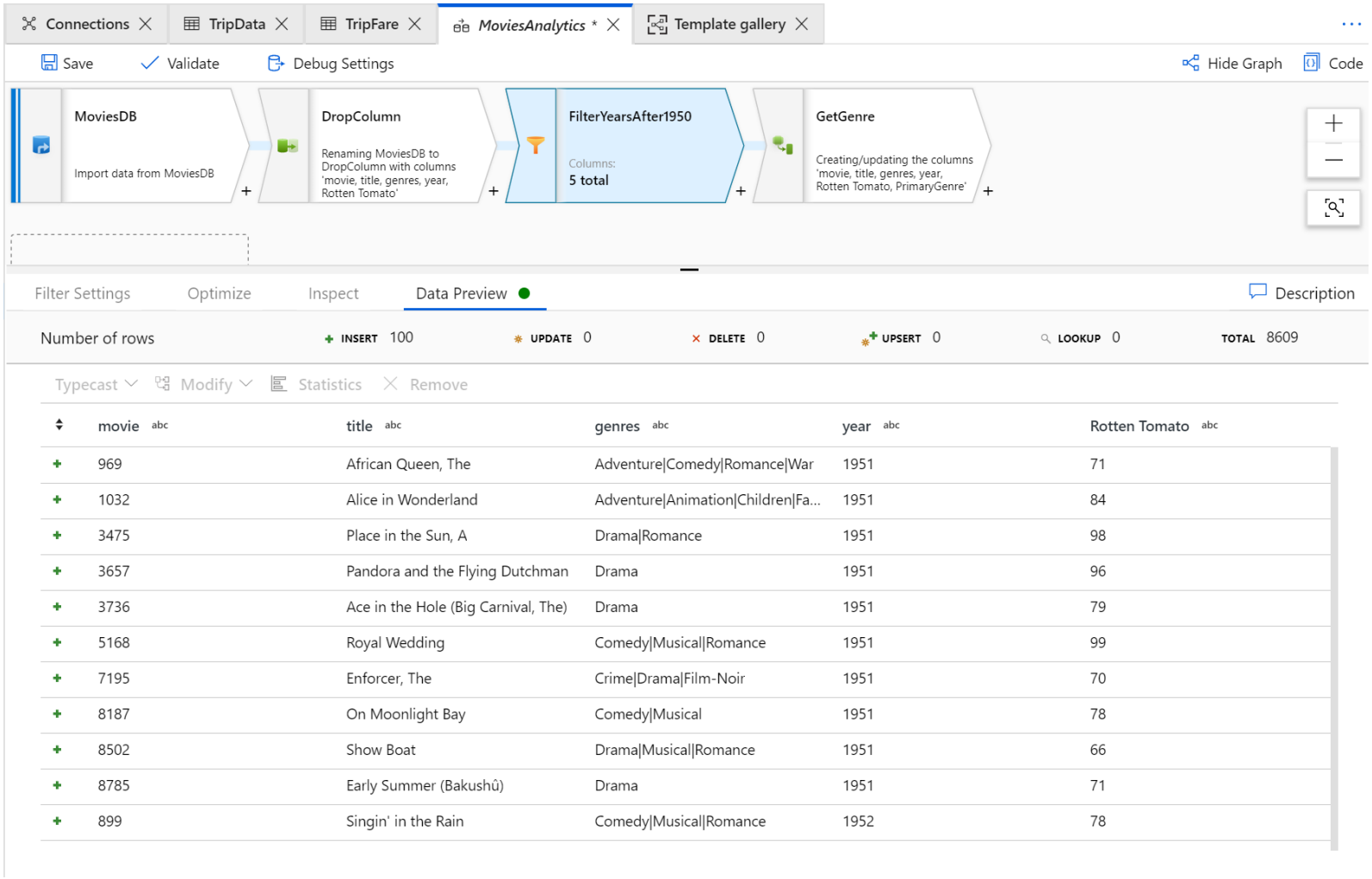
Debug settings

- Set Parameter values and sample data in debug settings
  - Change # of rows used per source
  - Replace source with debug dataset
  - Assign debug parameter values



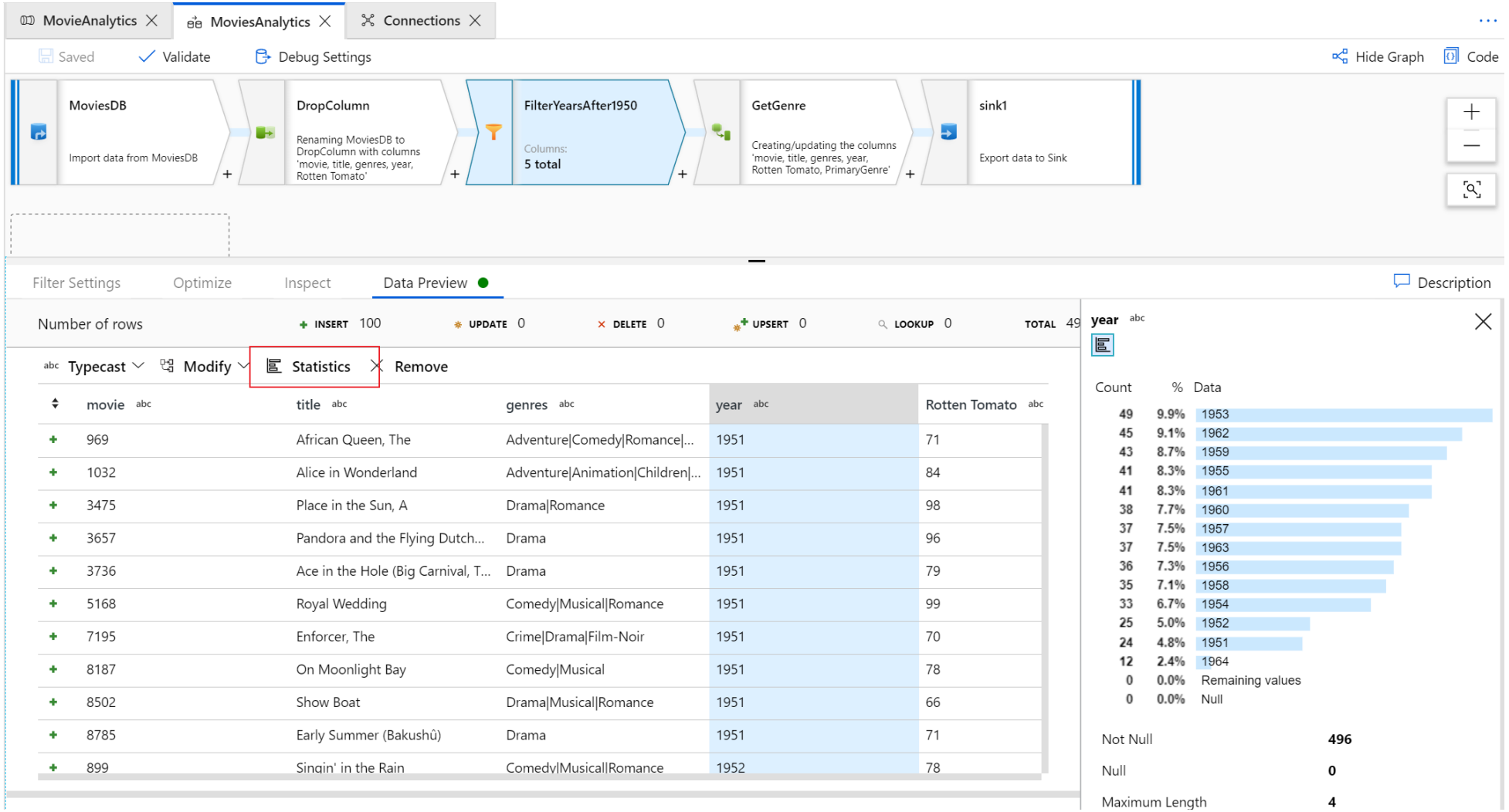


Data preview

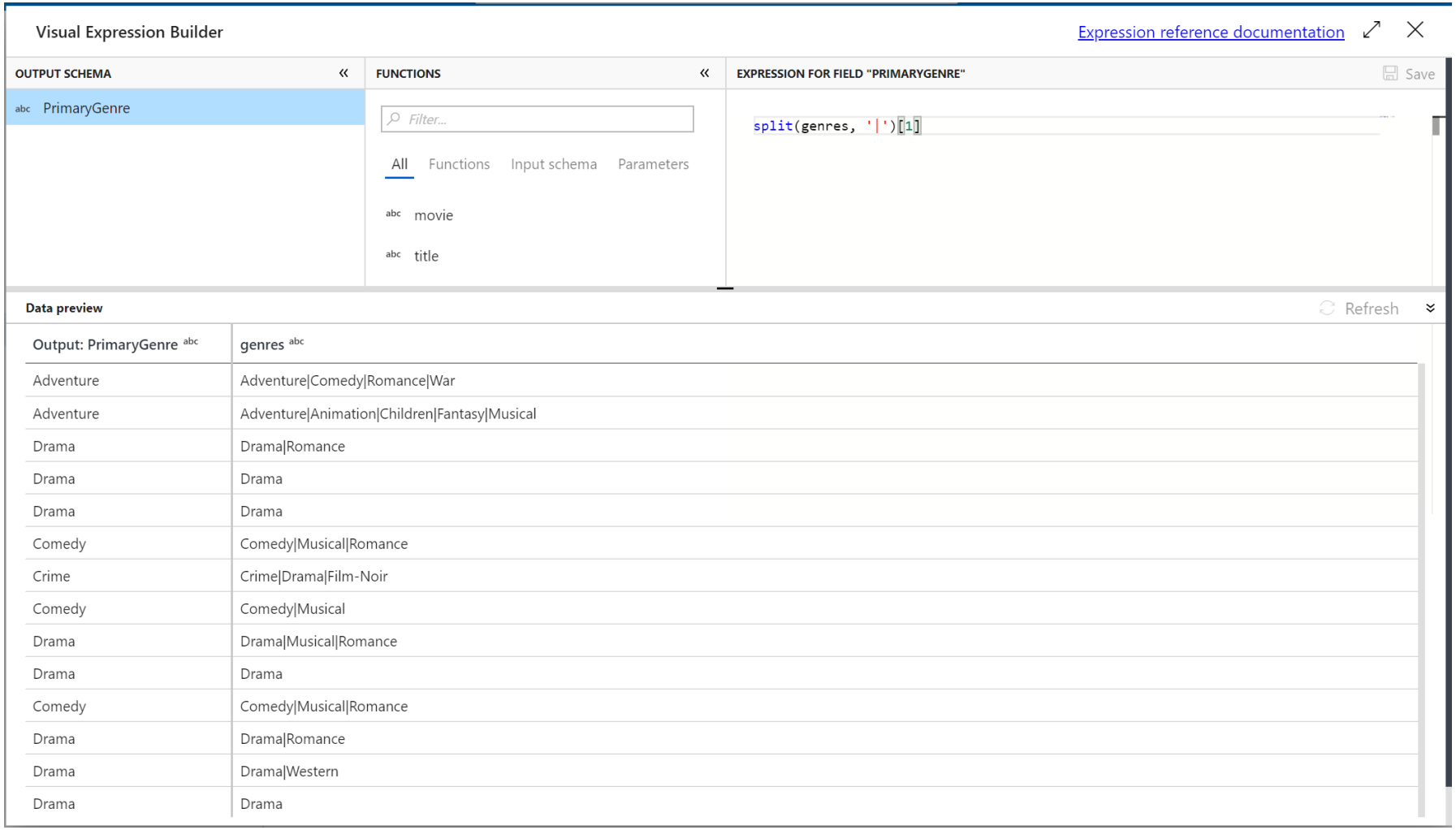


movie	title	genres	year	Rotten Tomato
969	African Queen, The	Adventure|Comedy|Romance|War	1951	71
1032	Alice in Wonderland	Adventure|Animation|Children|Fa...	1951	84
3475	Place in the Sun, A	Drama|Romance	1951	98
3657	Pandora and the Flying Dutchman	Drama	1951	96
3736	Ace in the Hole (Big Carnival, The)	Drama	1951	79
5168	Royal Wedding	Comedy|Musical|Romance	1951	99
7195	Enforcer, The	Crime|Drama|Film-Noir	1951	70
8187	On Moonlight Bay	Comedy|Musical	1951	78
8502	Show Boat	Drama|Musical|Romance	1951	66
8785	Early Summer (Bakushû)	Drama	1951	71
899	Singin' in the Rain	Comedy|Musical|Romance	1952	78

Data profiling



Expression output preview



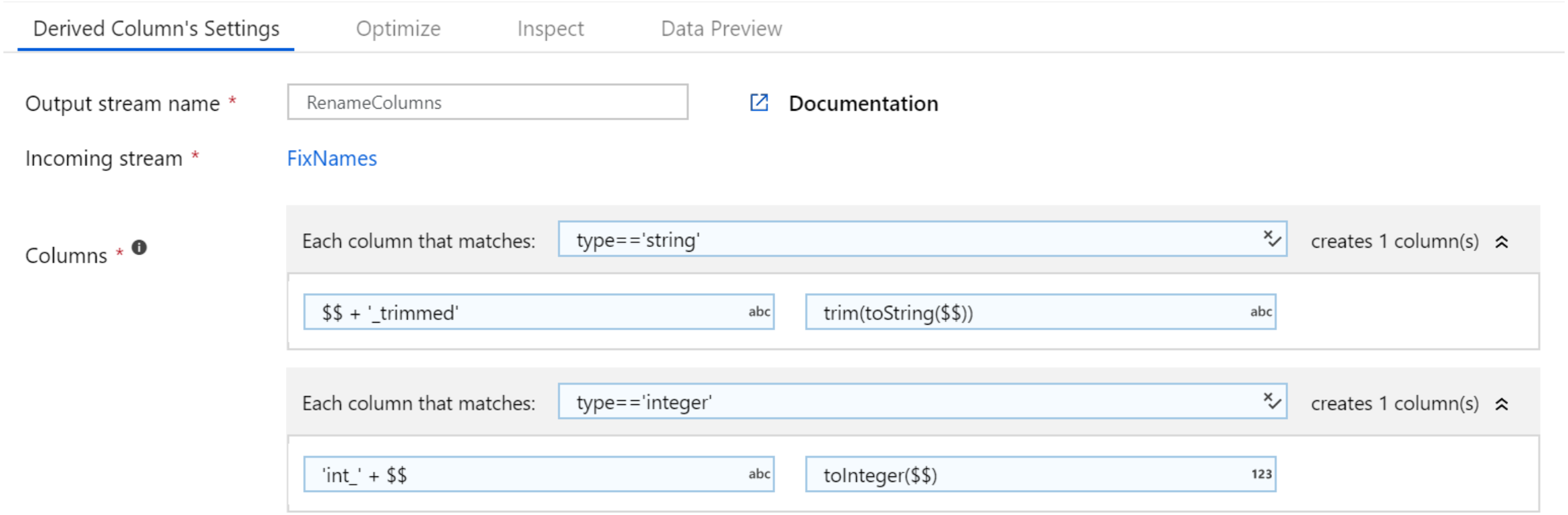
Output: PrimaryGenre abc	genres abc
Adventure	Adventure|Comedy|Romance|War
Adventure	Adventure|Animation|Children|Fantasy|Musical
Drama	Drama|Romance
Drama	Drama
Drama	Drama
Comedy	Comedy|Musical|Romance
Crime	Crime|Drama|Film-Noir
Comedy	Comedy|Musical
Drama	Drama|Musical|Romance
Drama	Drama
Comedy	Comedy|Musical|Romance
Drama	Drama|Romance
Drama	Drama|Western
Drama	Drama

Schema drift

- In most real-world data integration solutions, source and target data stores will change shape
  - Source data fields will change name
  - Number of columns will change over time
- Traditional ETL processes break when schemas drift
- Mapping Data Flow has built-in facilities for flexible schemas to handle schema drift
  - Patterns, rule-based mapping, byName function
  - Source: Read additional columns on top of what is defined in the dataset source
  - Sink: Write additional columns on top of what is defined in the dataset sink

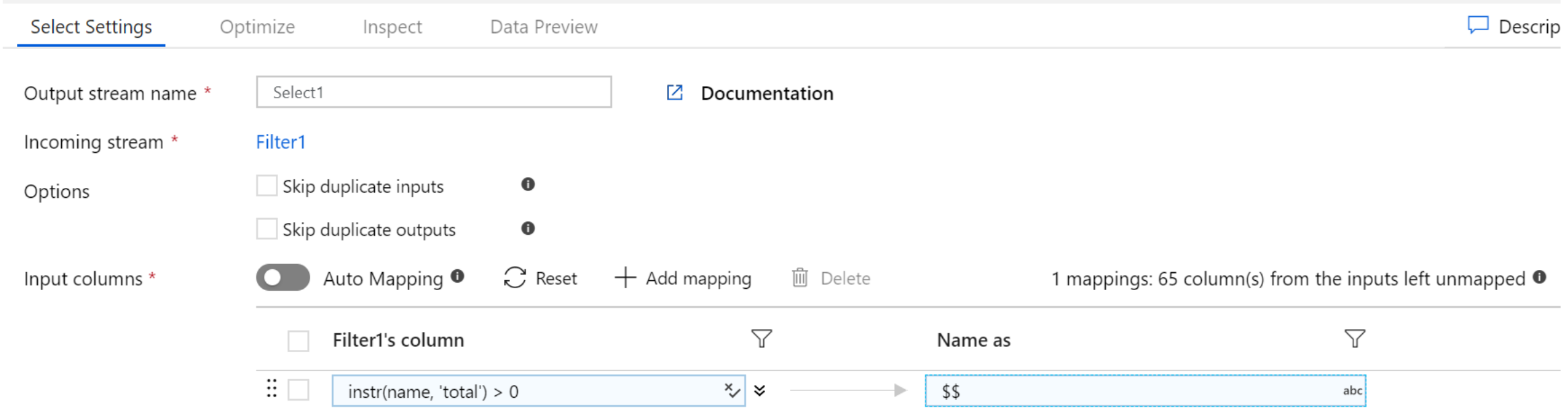
Pattern matching

- Match by name, type, stream, position



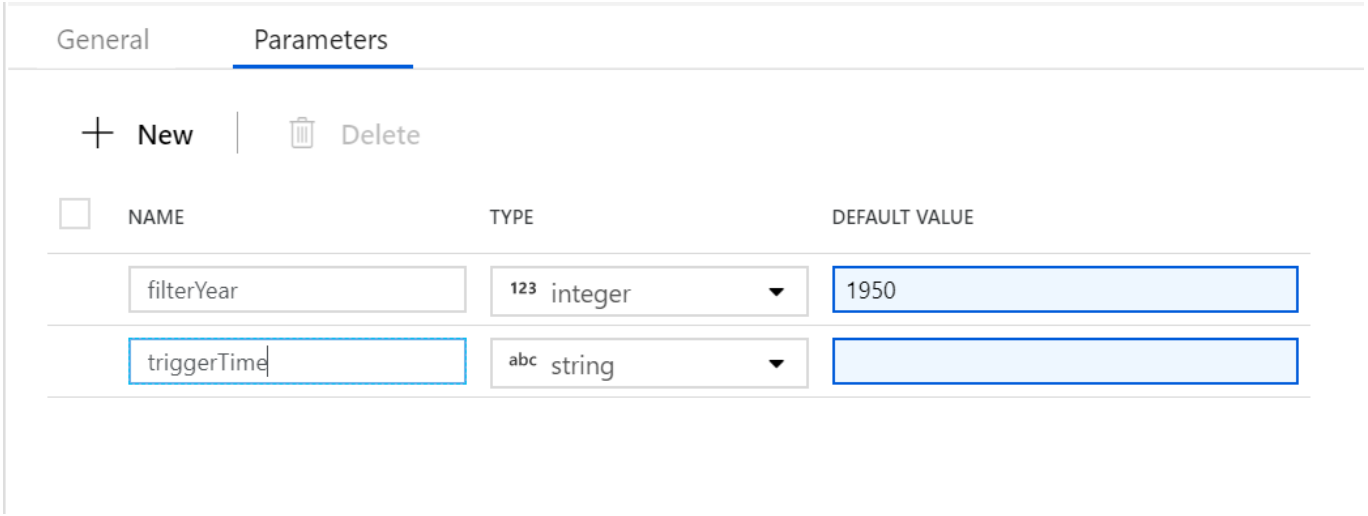
Rule-based mapping

Rather than pick and choose columns for transformations one-by-one, build policies that collect columns based on matching rules.

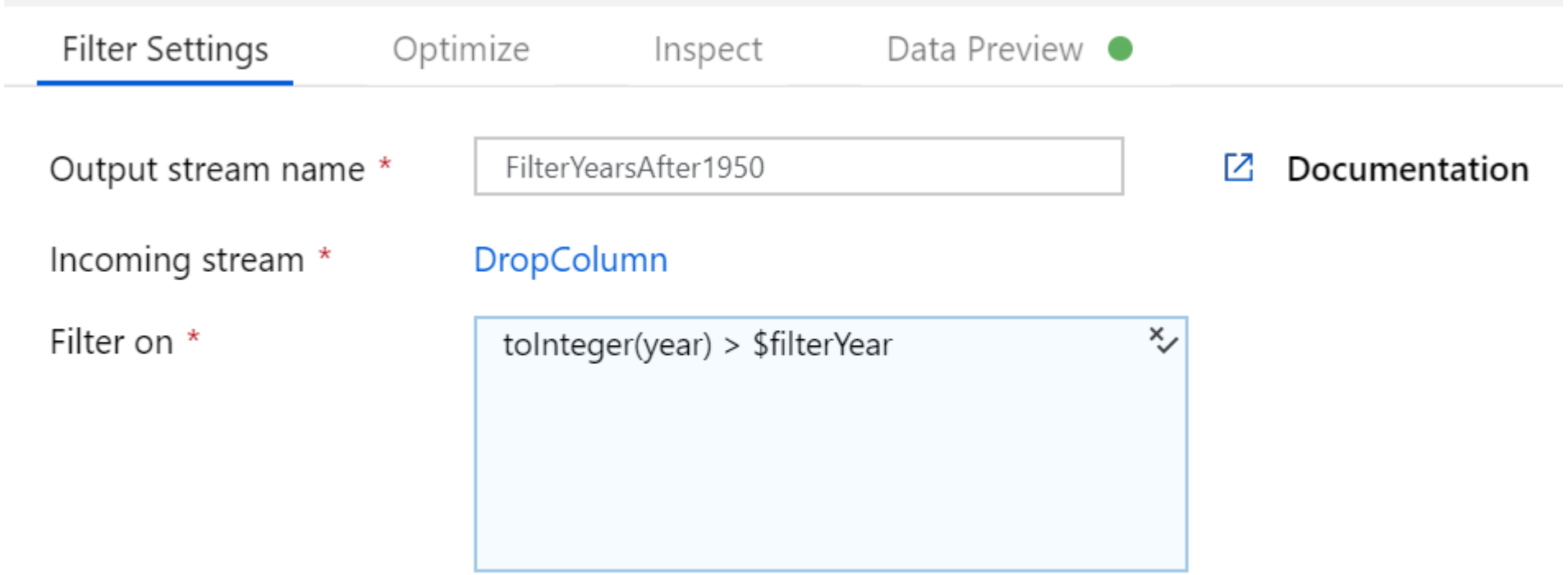


Data flow parameters

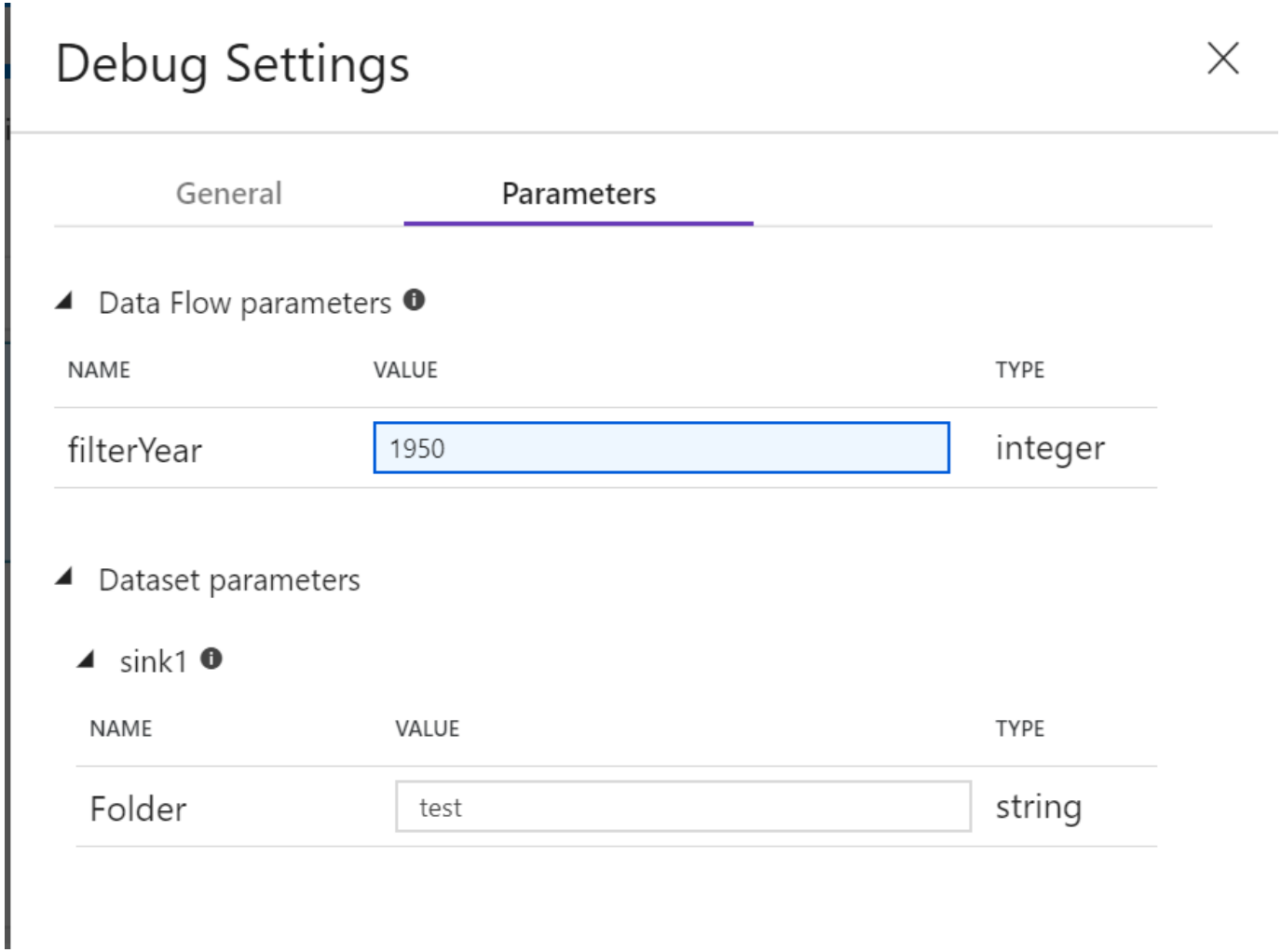
- Any expression value or computed column can reference Data Flow parameters
  - Set in general Data Flow panel
  - Reference in expression builder
    - $parameterName
- Can be assigned to any data flow types



Data flow parameters



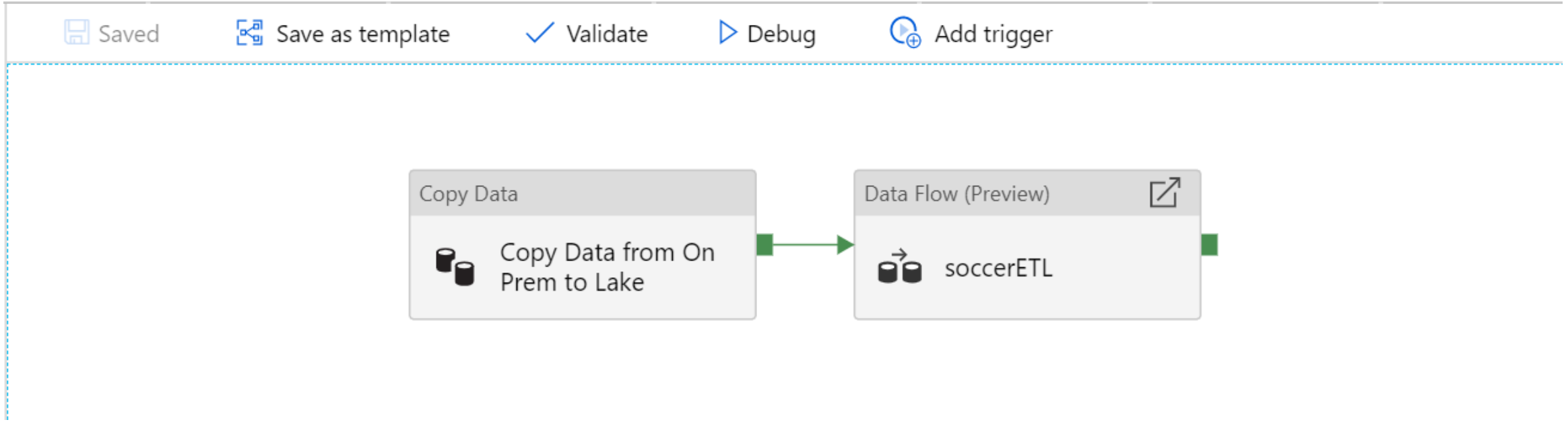
Debugging with parameters



Operationalizing Mapping Data Flows

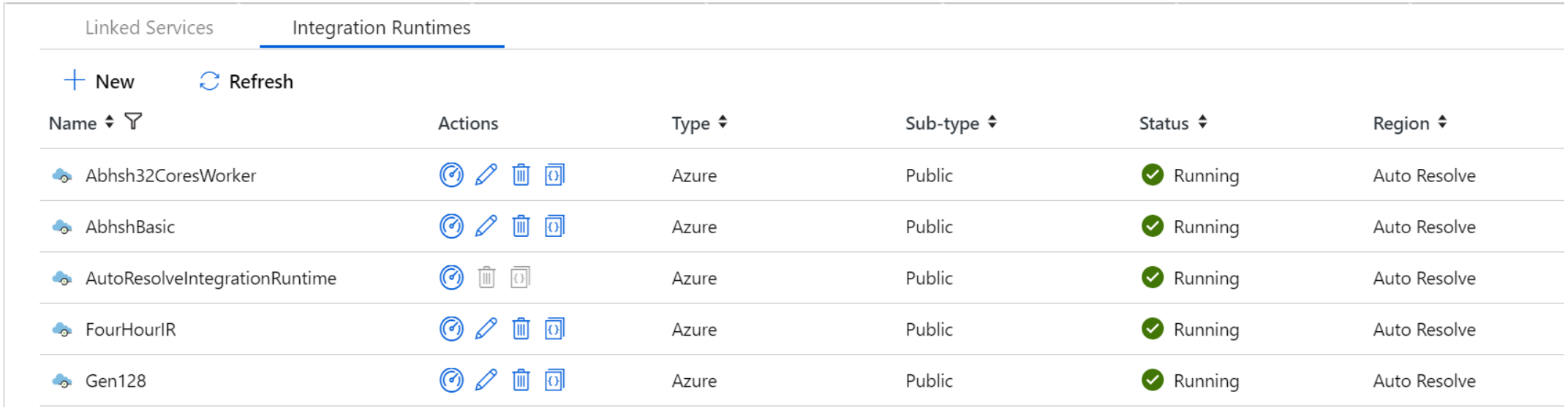
Data flow activity

- Ran as an activity in pipeline
  - Integrated with existing ADF control flow, scheduling, orchestration
- Choose which IR to run on
- Assign parameters



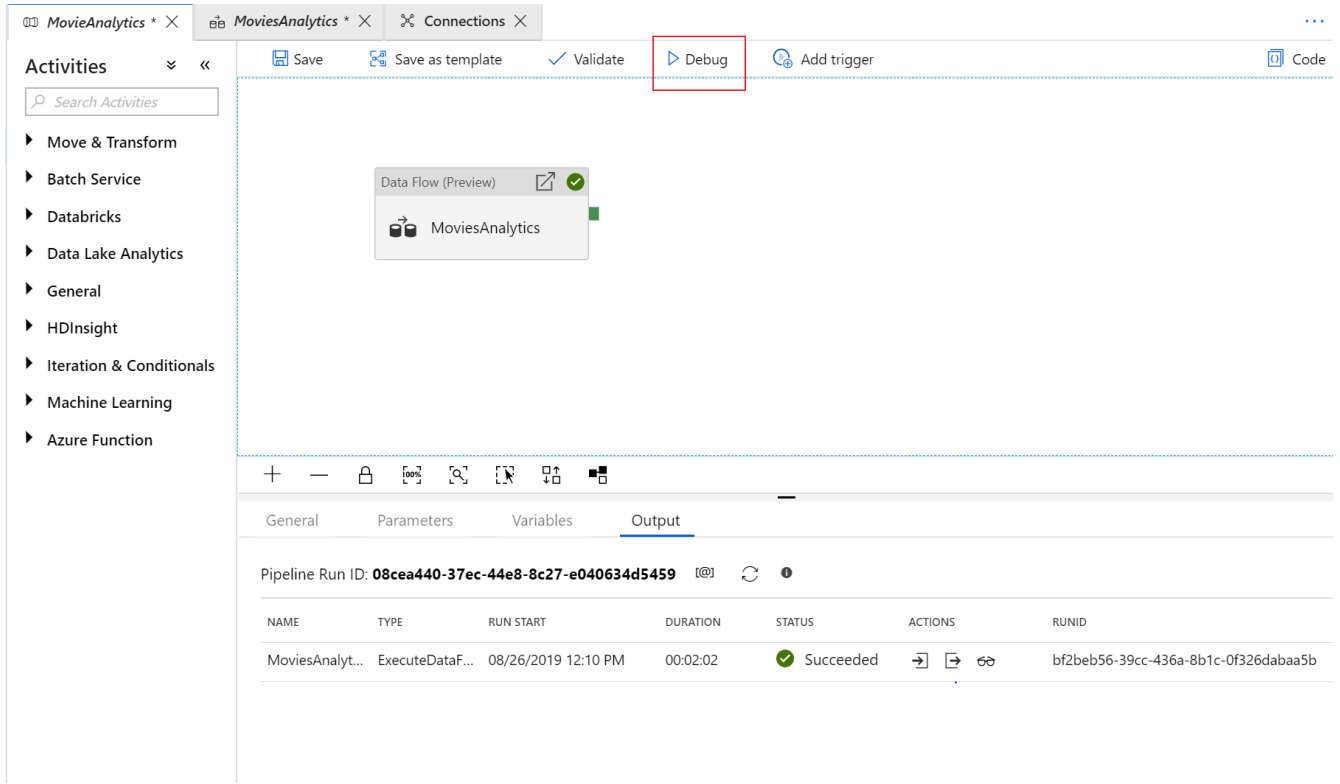
Data flow integration runtime

- Existing Azure IR
  - Choose compute type, # of cores, time to live
- **Time to live**: time a cluster is alive after last execution has concluded
  - Allows for chaining of sequential data flows without additional start up time

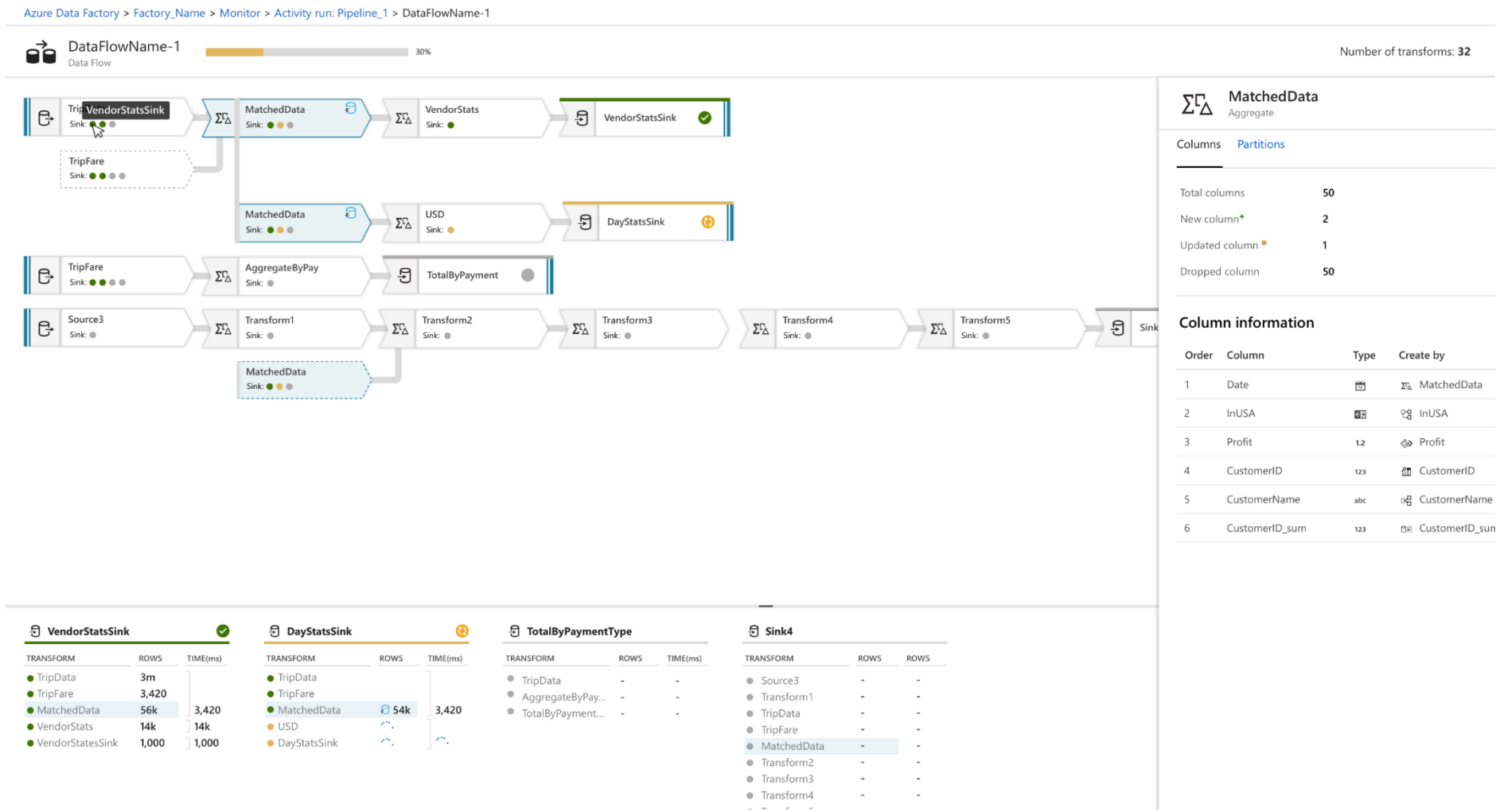


Pipeline debug run

- Runs against debug cluster
  - No 5 minute start up time
  - **Note:** Debug cluster is small and cannot handle most production workloads



Monitoring data flows



Wrangling Data Flows – Public Preview*

Wrangling Data flows

- Fast Interactive Data exploration and preparation for downstream analytics
- Code free agile data preparation at Cloud scale
- Leverages Power Query Online
- Data Validation – Remove Outliers, Anomalies
- Known schema to Unknown schema
- Ideal for data engineers\data scientists or heavy Excel users

Mapping vs Wrangling Data Flows-When to use what?

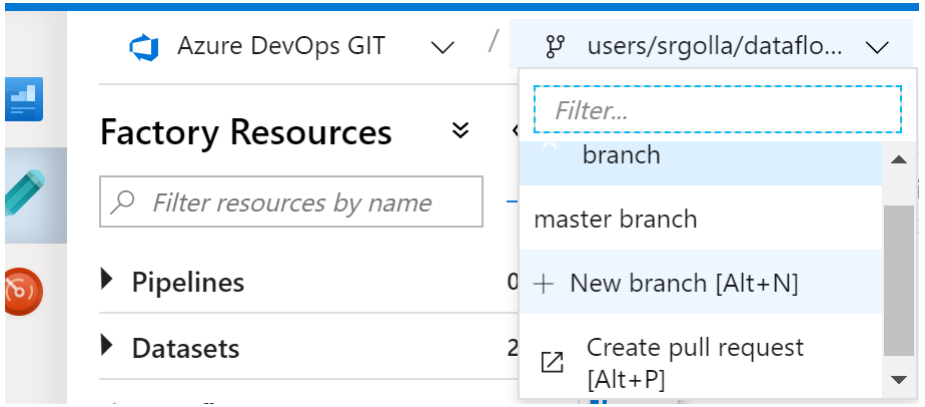
Mapping

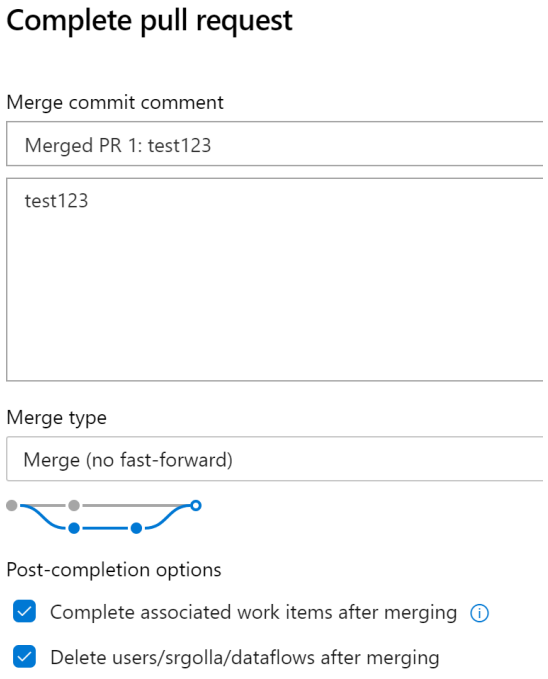
- Visually Transform data
- Known schema to known schema
- Translates visual logic into Spark
- Handles Schema Drift
- File and Table handling

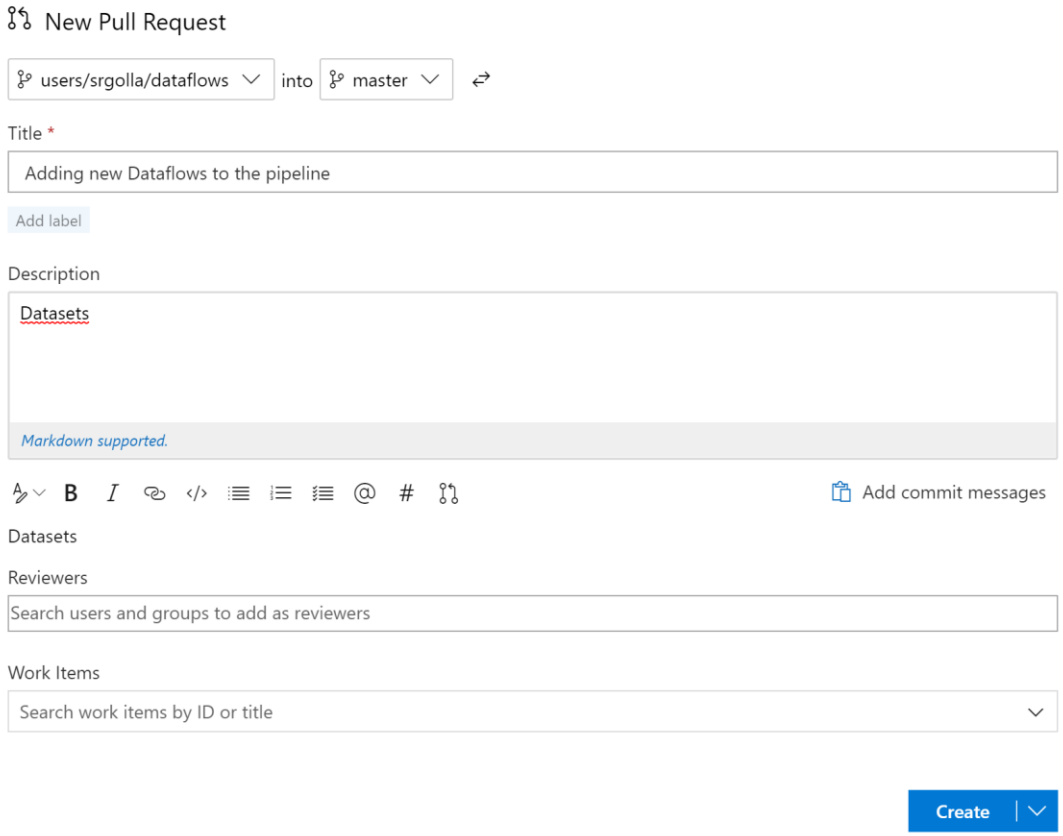
Wrangling

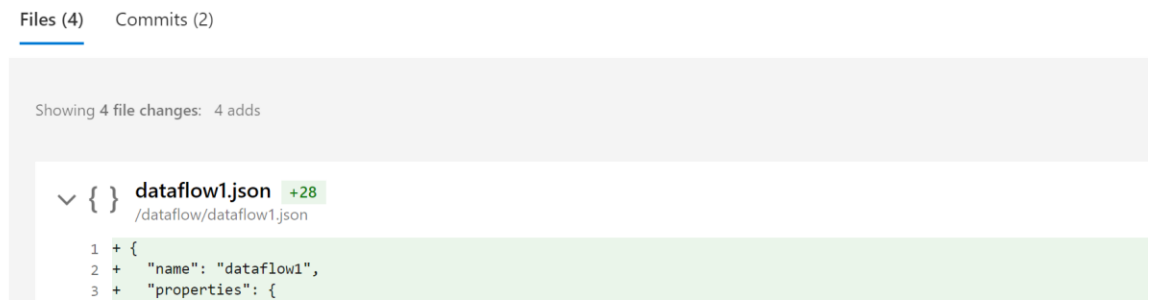
- Visually Explore and Prepare data
- Unknown schema exploration
- Translates M code to Spark
- No schema drift
- No File and table handling

Azure Devops integration - Build

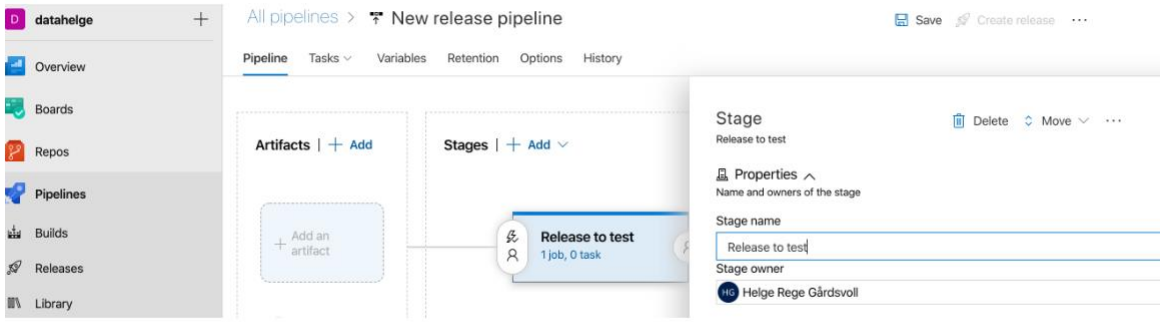




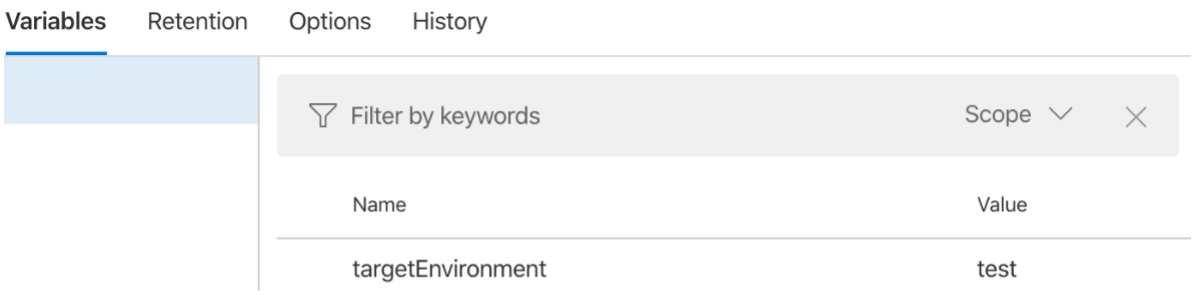




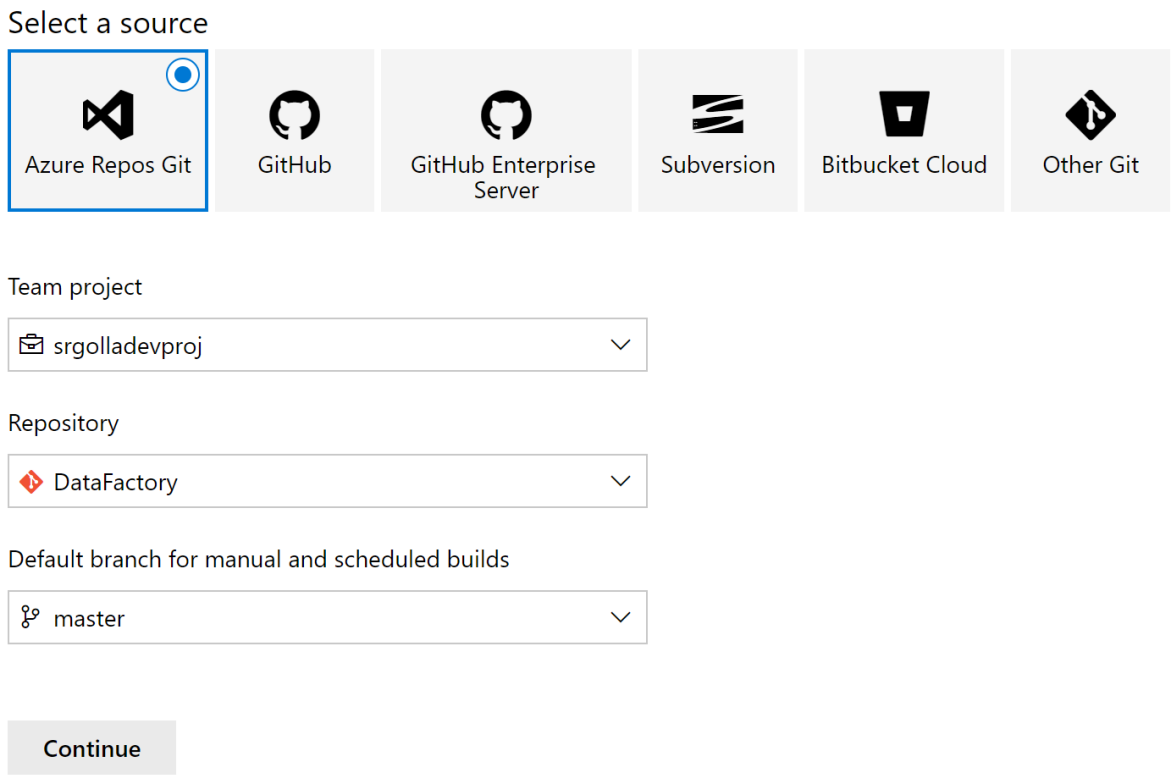
Release Pipelines



Creating a new release pipeline



Add a variable for the target environment



Mapping Data Flows Best Practices

Best Practices - Lifecycle

- Data Flow Performance Guide: https://aka.ms/dfperf
- Data Flow Debug – Data Preview
  - Limit source sizes
  - Use sample files
    - Upload local files
  - Increase size of debug clusters
- Test in Pipeline Debug before operationalizing
  - Set proper size of debug cluster
  - This is the right time to test your partitioning strategies
  - Consider this a smoke test
- Final pre-prod test: Trigger Now
- Check the output from your Sink in pipeline runs. Use this step as a sanity check.
- Now you're ready to set scheduled triggers for your data flow pipeline

Best Practices - Partitions

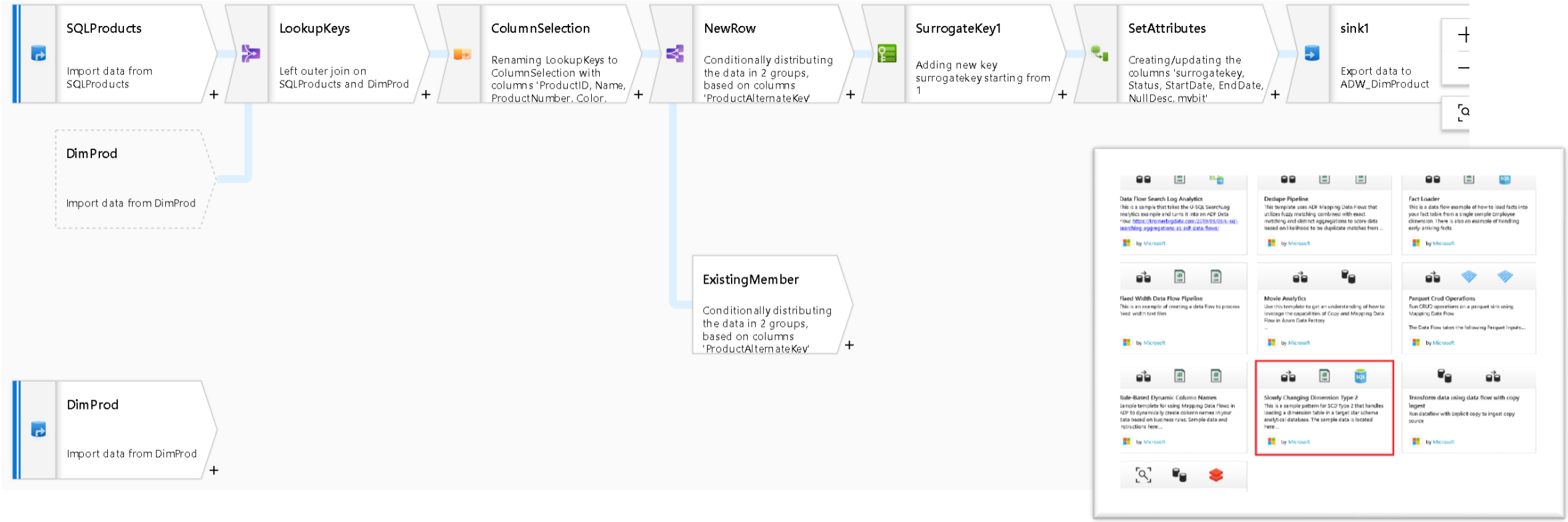
- Sources
  - File
    - Start with Round Robin, use Hash or Key if you understand your data and have high cardinality
  - Databases
    - Choose isolation level to match your performance requirements
    - Partition using “Source” partitioning
    - Use Batch size for large reads
- Sinks
  - File
    - Only use “Single File” output for testing or small files. Reduces output to single coalesced partition.
    - Allow part files to be written naturally, then collect them with a subsequent Copy Activity
  - Databases
    - Use pre/post processing scripts to disable/enable indexes, identity columns and other constraints that slow down ETL writing process
    - Set batch size for large writes
- Transformations
  - Use Broadcast for all join operations unless sources are very large
  - “Blocking” transformations like Sort that require visibility to all data before completing will reset your partitioning scheme

Best Practices - General

- Aggregations, Window, Pivot, Unpivot
  - These are “collection” transformations that will require you to reconnect your metadata after the Tx
  - Either use the self-join pattern or include all needed columns inside the Tx
    - https://mssqldude.wordpress.com/2018/12/20/adf-data-flows-self-join/
- Select Transformation
  - Very important transformation to prune unwanted columns, alias stream and columns for rule-based mapping
  - Use Select to define canonical models
- Set Data Types in Derived Columns
  - All delimited text file column values are read in as string data types because there is no data type stored in the file
  - Setting data types in Source Projections is volatile
  - Changing your source dataset removes your data type conversions

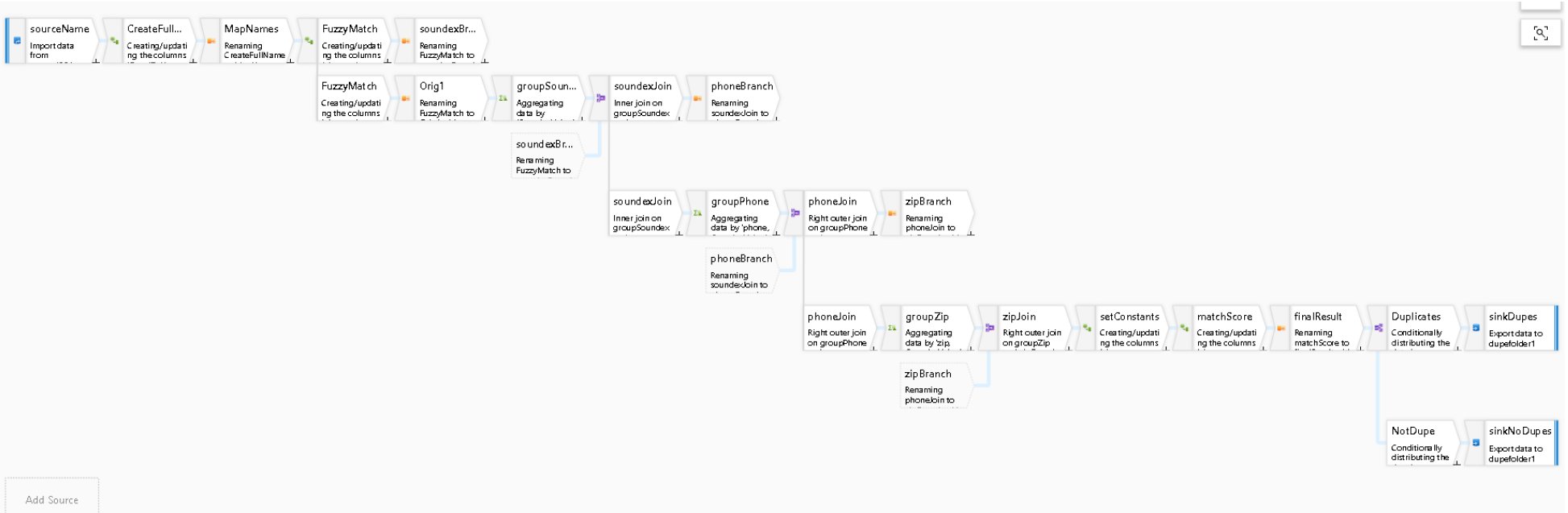
Mapping Data Flow common scenarios

Slowly Changing Dimension Scenario

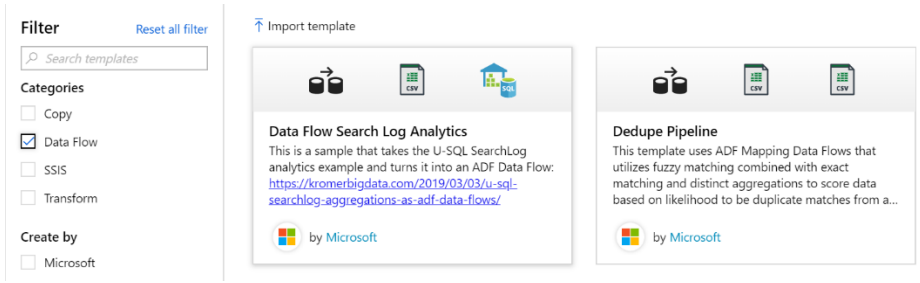


- Common DW pattern to manage changing attributes to dimension members
- Graphically build code-free SCD ETL pattern to load your data warehouse
- Connect directly to Azure SQL DB and Azure SQL DW
- Use Lookup, Surrogate Key, Derived Column and Select transforms

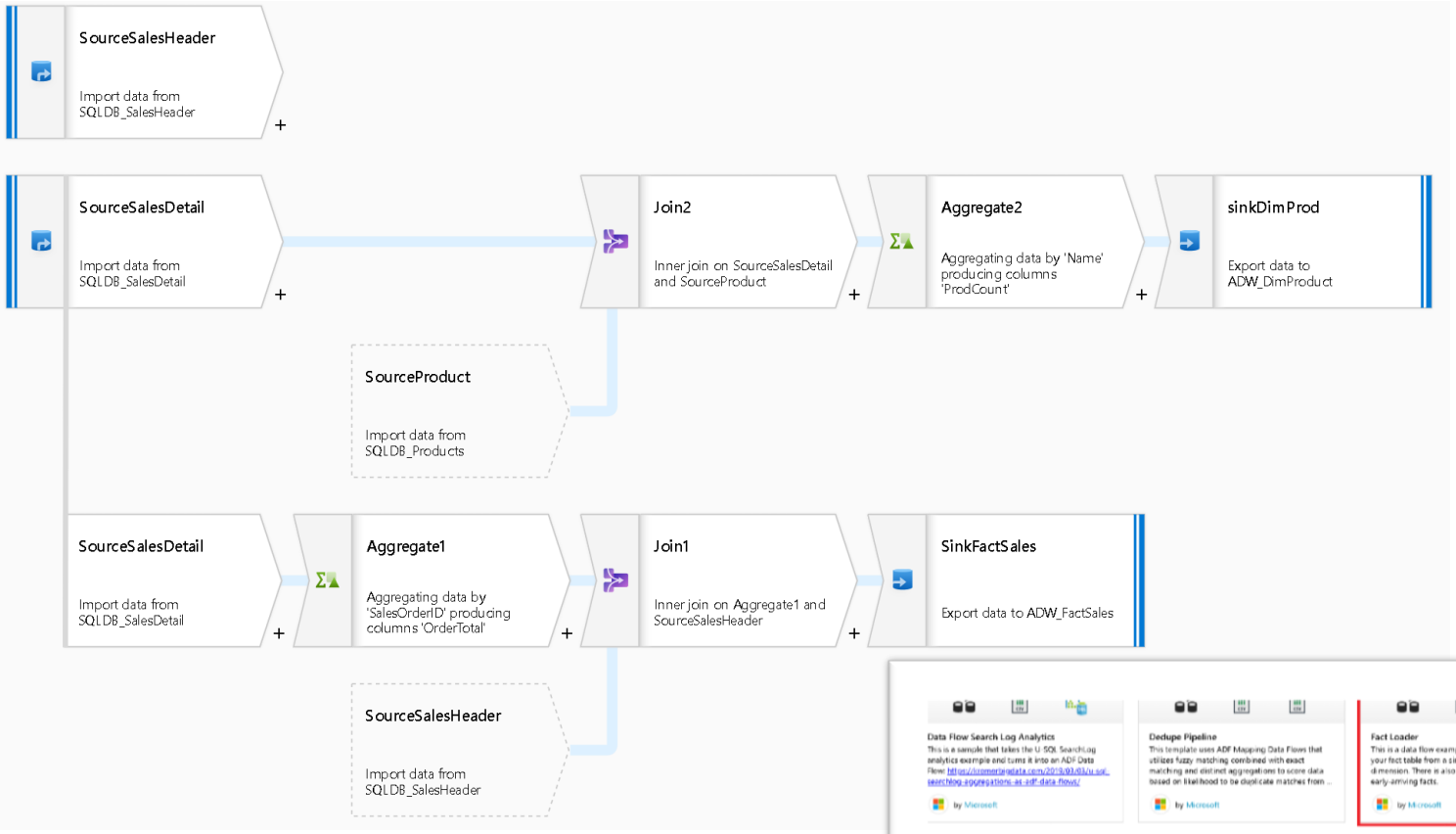
Data De-Duplication

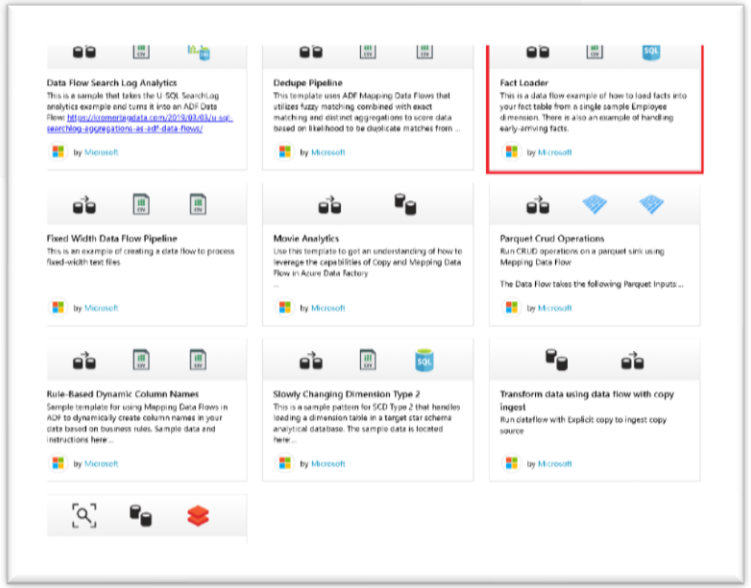


- Use this pattern to eliminate common rows from your data
- You pick a heuristic to use during duplicate matching
- You can tag rows and/or remove duplicate rows
- Use exact matching and/or fuzzy matching
- Available as pipeline template *Dedupe Pipeline*



Load Star Schema DW Scenario





- Classic ETL pattern is easy to build in ADF's code-free Data Flow visual data transformation environment
- Add Aggregate transforms to produce calculations that you store in your analytical database schema
- Use Join transform to combine data from multiple data sources and data streams inside your data flow

  Land your data in your Lake folders or direct to Azure SQL DW

Conclusion

Next Steps

- This was an Intro session to DataFlows.
- Please let us know if you are interested in a deep dive session
- Please reach out to us for any migrations from SSIS\Other ETL tools to Dataflows
- Partners – Varigence etc – Migrate from SSIS to Dataflows

- Krishna Golla – srgolla@Microsoft.com
- GinSiu Cheng - Gin.Cheng@microsoft.com
- Patrick Gryckza - Patrick.Gryczka@microsoft.com

Labs

- Aka.ms/dflabnyc