

Lab overview

In this module, students will learn how Azure Data factory can be used to orchestrate the data movement from a wide range of data platform technologies. They will be able to explain the capabilities of the technology and be able to set up an end to end data pipeline that ingests data from SQL Database and load the data into Azure Synapse Analytics. The student will also demonstrate how to call a compute resource.

Lab objectives

After completing this lab, you will be able to:

1. Setup Azure Data Factory
2. Ingest data using the Copy Activity
3. Use the Mapping Data Flow task to perform transformation
4. Perform transformations using a compute resource

Scenario

You are assessing the tooling that can help with the extraction, load and transforming of data into the data warehouse, and have asked a Data Engineer within your team to show a proof of concept of Azure Data Factory to explore the transformation capabilities of the product. The proof of concept does not have to be related to AdventureWorks data, and you have given them freedom to pick a dataset of their choice to showcase the capabilities.

In addition, the Data Scientists have asked to confirm if Azure Databricks can be called from Azure Data Factory. To that end, you will create a simple proof of concept Data Factory pipeline that calls Azure Databricks as a compute resource.

At the end of this lab, you will have:

1. Setup Azure Data Factory
2. Ingested data using the Copy Activity
3. Used the Mapping Data Flow task to perform transformation
4. Performed transformations using a compute resource

Exercise 1: Setup Azure Data Factory

Individual exercise

The main task for this exercise are as follows:

1. Setup Azure Data Factory

Task 1: Setting up Azure Data Factory.

Create your data factory: Use the [Azure Portal](#) to create your Data Factory.

1. In Microsoft Edge, go to the Azure portal tab, click on the **+ Create a resource** icon, type **data factory**, and then click **Data Factory** from the resulting search, and then click **Create**.
2. In the New Data Factory screen, create a new Data Factory with the following options:
 - **Subscription:** Your subscription
 - **Resource group:** awrgstudxx
 - **Region:** select the location closest to you
 - **Name:** xx-data-factory, where xx are your initials
 - **Version:** V2
 - **Git configuration:** Configure Git Later check
 - Leave other options to their default settings

Create Data Factory ...

The screenshot shows the 'Create Data Factory' page in the Azure portal. The 'Basics' tab is selected, showing 'Project details' and 'Instance details' sections. The 'Project details' section includes a 'Subscription' dropdown and a 'Resource group' dropdown with a 'Create new' link. The 'Instance details' section includes a 'Region' dropdown, a 'Name' text box with a green checkmark, and a 'Version' dropdown. At the bottom, there are three buttons: 'Review + create' (blue), '< Previous' (grey), and 'Next : Git configuration >' (white with a grey border).

Section	Field	Value	Status
Project details	Subscription *	[Blurred]	
	Resource group *	awrgstudkdj	
Instance details	Region *	West Europe	
	Name *	kdj-data-factory	✓
	Version *	V2	
	Buttons: Review + create, < Previous, Next : Git configuration >		

3. **Note:** The creation of the Data Factory takes approximately 1 minute.
4. In the **git configuration** blade **check** Configure git later.
5. Click **review + create** and then select **create**.

Result: After you completed this exercise, you have created an instance of Azure Data Factory

Exercise 2: Ingest data using the Copy Activity

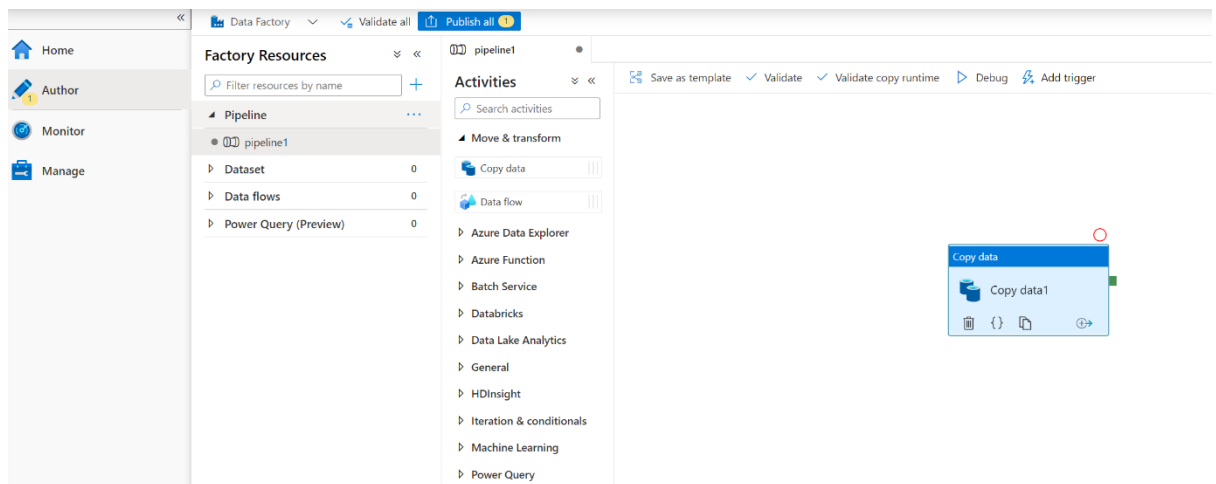
Individual exercise

The main tasks for this exercise are as follows:

1. Add the Copy Activity to the designer
2. Create a new HTTP dataset to use as a source
3. Create a new ADLS Gen2 sink
4. Test the Copy Activity

Task 1: Add the Copy Activity to the designer

1. On the deployment successful message, click on the button **Go to resource**.
2. In the xx-data-factory screen, in the middle of the screen, click on the button, **Author & Monitor**
3. **Open the authoring canvas** If coming from the ADF homepage, click on the **pencil icon** on the left sidebar and select the **+ pipeline button** to open the authoring canvas and create a pipeline.
4. **Add a copy activity** In the Activities pane, open the **Move and Transform** accordion and drag the **Copy data** activity onto the pipeline canvas.



Task 2: Create a new HTTP dataset to use as a source

1. In the Source tab of the Copy activity settings, click **+ New**
2. In the data store list, select the **HTTP** tile and click **continue**
3. In the file format list, select the **DelimitedText** format tile and click **continue**
4. In **Set properties** blade, give your dataset an understandable name such as **HTTPSource** and click on the **Linked Service** dropdown. If you have not created your HTTP Linked Service, select **New**.
5. In the **New Linked Service (HTTP)** screen, copy the URL of the moviesDB csv file below in the **Base URL** textbox. You can access the data with no authentication required using the following endpoint:

<https://github.com/parveenkraina/DE-Python-Data/raw/main/moviesDB.csv>
6. In the **Authentication type** drop down, select **Anonymous**. and click on **Create**.
 - Once you have created and selected the linked service, specify the rest of your dataset settings. These settings specify how and where in your connection we want to pull the data. As the url is pointed at the file already, no relative endpoint is required. As the data has a header in the first row, set **First row as header** to be true and select Import schema from **connection/store** to pull the schema from the file itself. Select **Get** as the request method. You will see the following screen

Set properties

Name

HTTPSource

Linked service *

HttpServer1



Relative URL

First row as header



Import schema



From connection/store



From sample file



None

Request method

GET



Additional headers

Request body

▸ Advanced

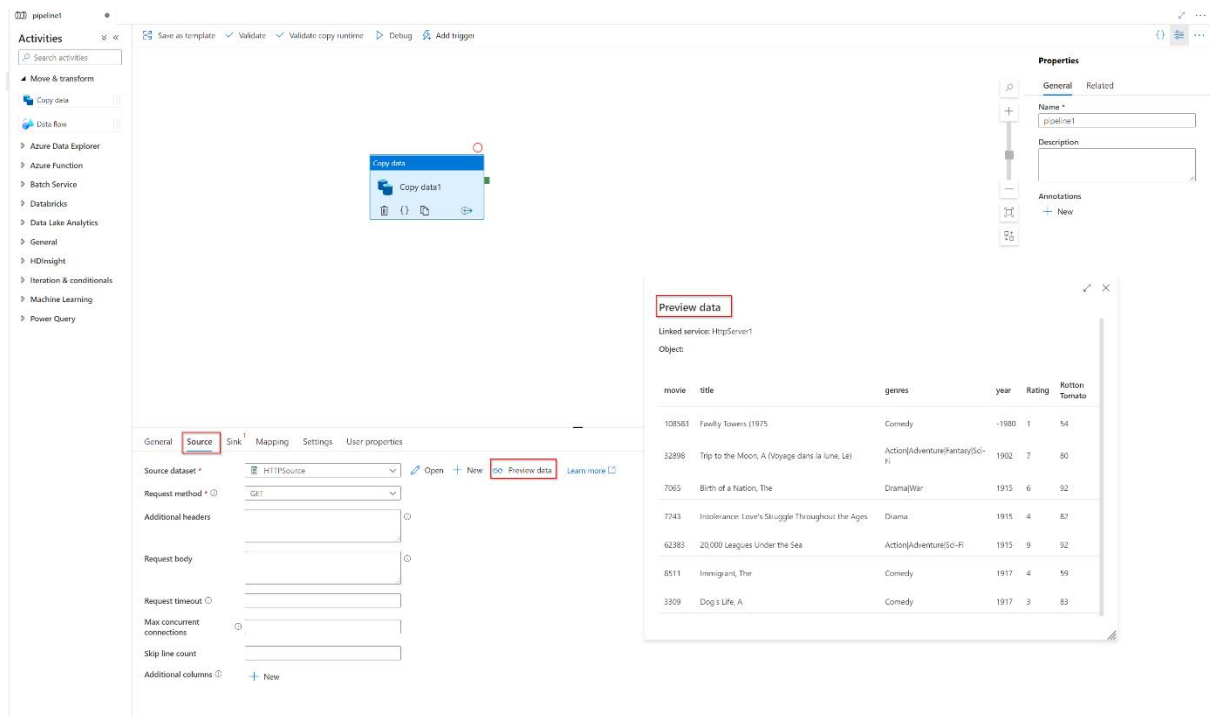
OK

Back

Cancel

- Click **OK** once completed.

a. To verify your dataset is configured correctly, click **Preview data** in the Source tab of the copy activity to get a small snapshot of your data.



Task 3: Create a new ADLS Gen2 dataset sink

1. Click on the **Sink tab**, and then click **+ New**
2. Select the **Azure Data Lake Storage Gen2** tile and click **Continue**.
3. Select the **DelimitedText** format tile and click **Continue**.
4. In the Set Properties blade, give your dataset an understandable name such as **ADLSG2** and click on the **Linked Service** dropdown. If you have not created your ADLS Linked Service, select **New**.
5. In the New linked service (Azure Data Lake Storage Gen2) blade, select your authentication method as **Account key**, select your **Azure Subscription** and select your Storage account name of **awdlstudxx**. You will see a screen as follows:

New linked service (Azure Data Lake Storage Gen2)

Name *
AzureDataLakeStorage1

Description

Connect via integration runtime * ⓘ
AutoResolveIntegrationRuntime

Authentication method
Account key

Account selection method ⓘ
☒ From Azure subscription ☐ Enter manually

Azure subscription ⓘ

Storage account name *
awdlssstudkdj

Test connection ⓘ
☒ To linked service ☐ To file path

Annotations
[+ New](#)

▸ Parameters

▸ Advanced ⓘ

[Create](#) [Test connection](#) [Cancel](#)

6. Click on **Create**
7. Once you have configured your linked service, you enter the set properties blade. As you are writing to this dataset, you want to point the folder where you want moviesDB.csv copied to. In the example below, I am writing to folder **output** in the file system **data**. While the folder can be dynamically created, the file system must exist prior to writing to it. Set **First row as header** to be true. You can either Import schema from **sample file** (use the moviesDB.csv file shared

Set properties

Name
ADLSG2

Linked service *
AzureDataLakeStorage1

File path
data / output / File

First row as header ☒

Import schema
☐ From connection/store
 ☒ From sample file
 ☐ None

Select file
 moviesDB.csv Browse

Advanced

OK Back Cancel

- Click **OK** once completed.

Task 4: Test the Copy Activity

At this point, you have fully configured your copy activity. To test it out, click on the **Debug** button at the top of the pipeline canvas. This will start a pipeline debug run.

- To monitor the progress of a pipeline debug run, click on the **Output** tab of the pipeline
- To view a more detailed description of the activity output, click on the eyeglasses icon. This will open up the copy monitoring screen which provides useful metrics such as Data read/written, throughput and in-depth duration statistics.

Parameters Variables Settings Output							
Pipeline run ID: 338b90f-e17c-4861-b20b-a2b5cdc9546f 🔄 🔍 View debug run consumption							
Name	Type	Run start	Duration	Status	Integration runtime	Run ID	
Copy data1	Copy data	2021-06-14T13:48:49.8748741Z	00:00:05	Succeeded	DefaultIntegrationRuntime (West Europe)	eda2a8bc-1f52-4261-84f6-c08bbc03...	

- To verify the copy worked as expected, open up your ADLS gen2 storage account and check to see your file was written as expected

Exercise 3: Transforming Data with Mapping Data Flow

Individual exercise

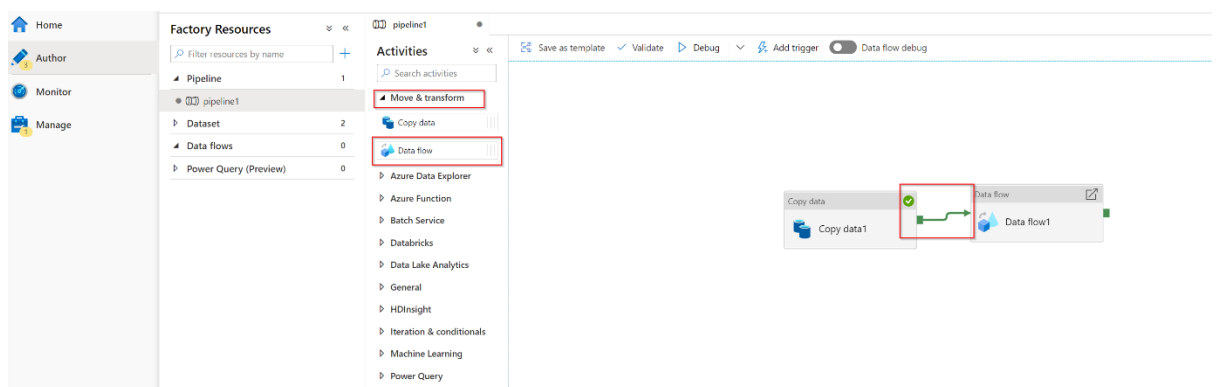
Now that you have moved the data into Azure Data Lake Store Gen2, you are ready to build a Mapping Data Flow which will transform your data at scale via a spark cluster and then load it into a Data Warehouse.

The main tasks for this exercise are as follows:

1. Preparing the environment
2. Adding a Data Source
3. Using Mapping Data Flow transformation
4. Writing to a Data Sink
5. Running the Pipeline

Task 1: Preparing the environment

1. **Add a Data Flow activity** In the Activities pane, open the **Move and Transform** accordion and drag the **Data Flow** activity onto the pipeline canvas and connect with the **Copy data** activity we created earlier.



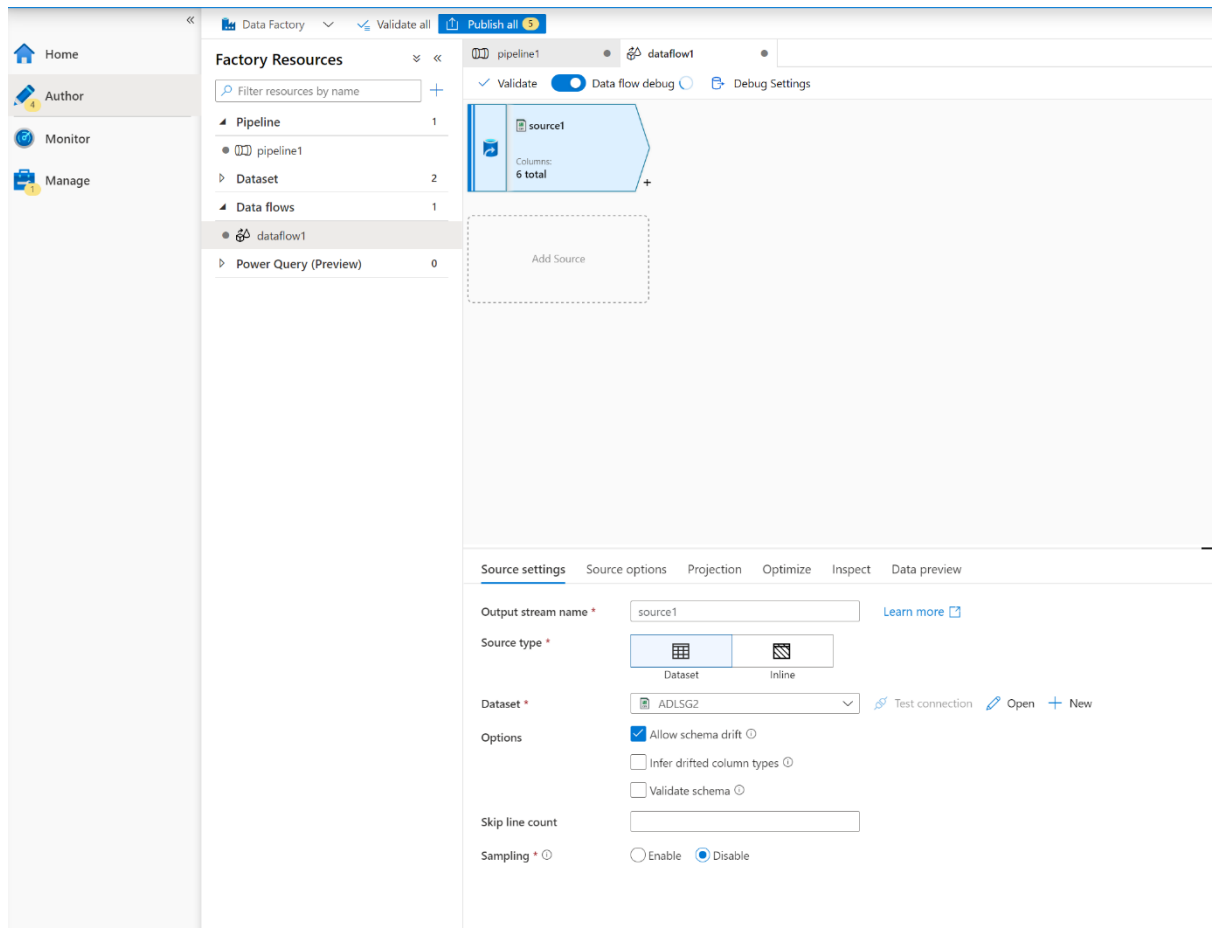
2. Turn the **Data Flow Debug** slider located at the top of the authoring module on, and click **OK** in the **Turn on data flow debug** screen that appears.

NOTE: Data Flow clusters take 5-7 minutes to warm up.

3. Select the data flow activity in the pipeline workspace. In the lower pane, select the settings tab, click **+ New** for the variable **Dataflow**

Task 2: Adding a Data Source

1. **Add an ADLS source:** Click on the Mapping Data Flow object in the canvas. Go to the source settings tab. In the **Dataset** dropdown, select your **ADLSG2** dataset used in your Copy activity

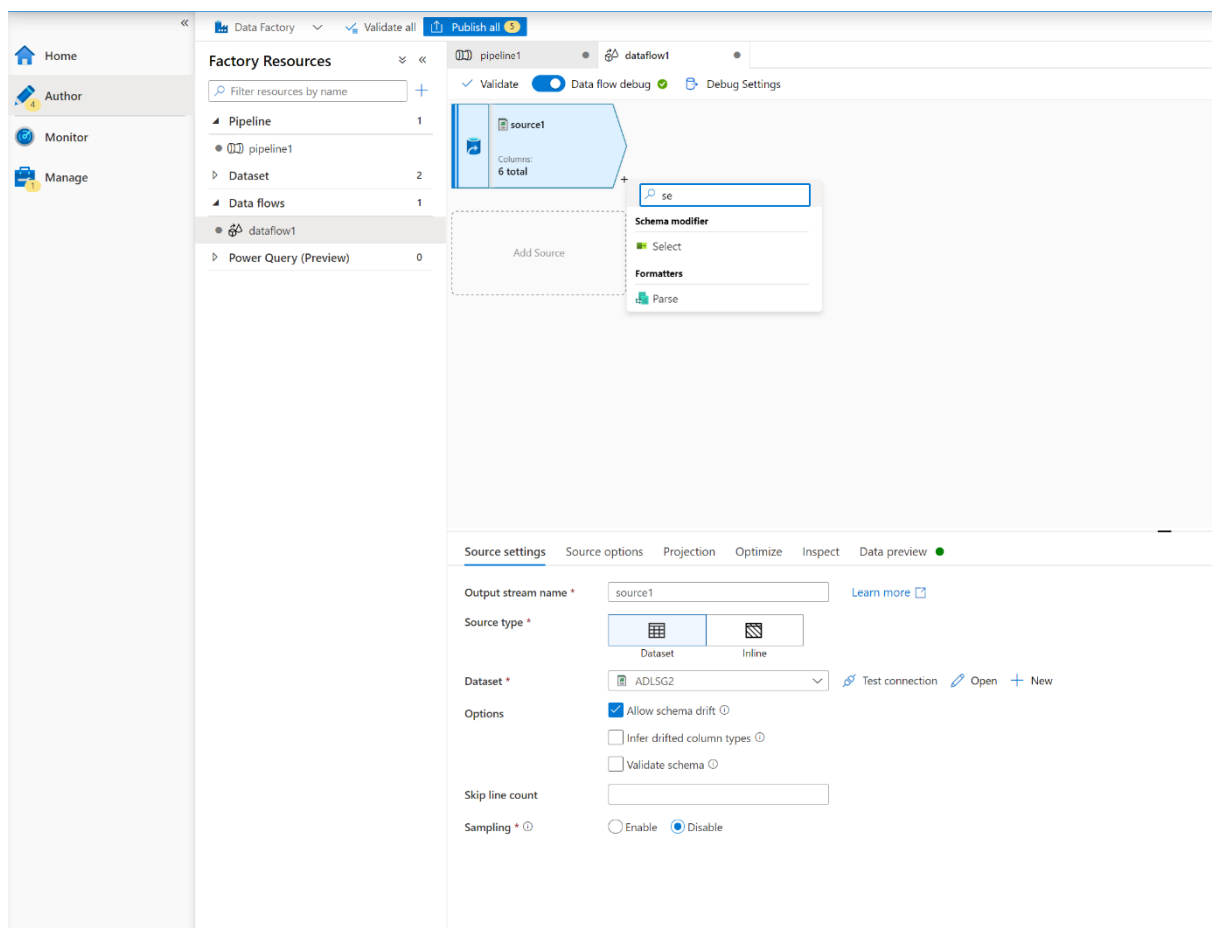


- If your dataset is pointing at a folder with other files, you may need to create another dataset or utilize parameterization to make sure only the moviesDB.csv file is read
- If you have not imported your schema in your ADLS, but have already ingested your data, go to the dataset's 'Schema' tab and click 'Import schema' so that your data flow knows the schema projection.

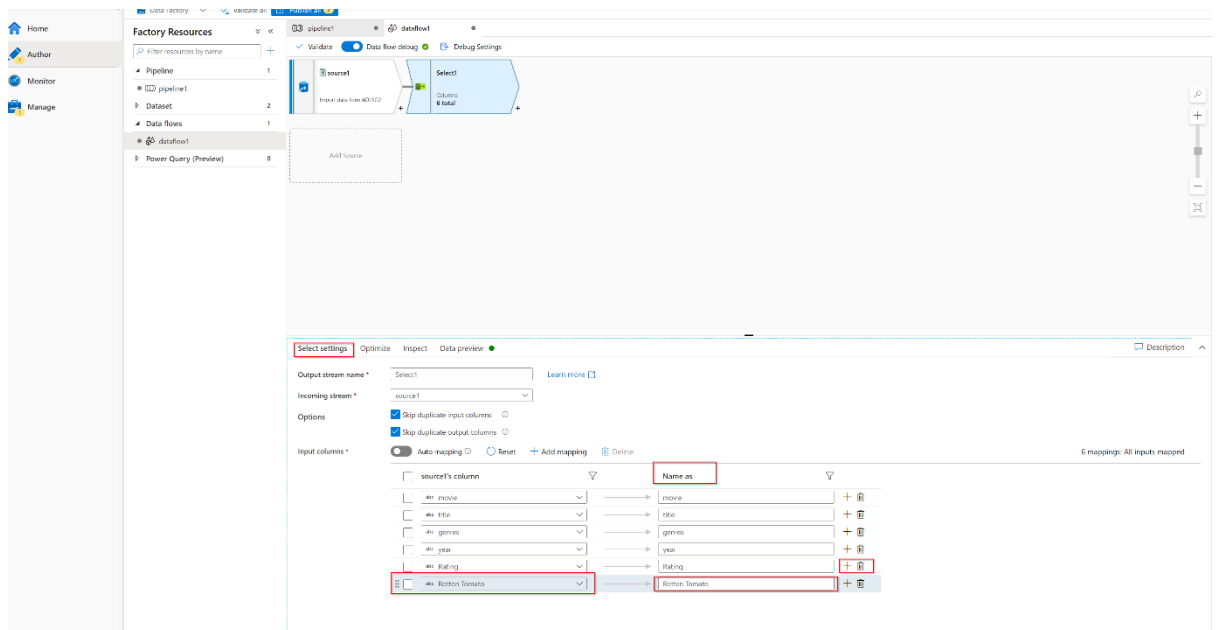
Once your debug cluster is warmed up, verify your data is loaded correctly via the **Data preview** tab. Once you click the refresh button, Mapping Data Flow will show calculate a snapshot of what your data looks like when it is at each transformation.

Task 3: Using Mapping Data Flow transformation

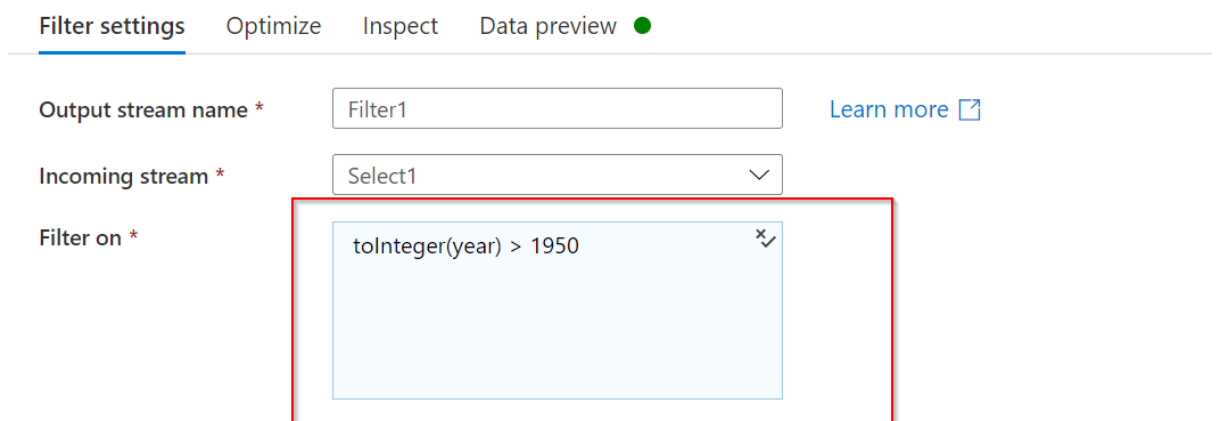
1. **Add a Select transformation to rename and drop a column:** In the preview of the data, you may have noticed that the "Rotten Tomatoes" column is misspelled. To correctly name it and drop the unused Rating column, you can add a [Select transformation](#) by clicking on the + icon next to your ADLS source node and choosing **Select** under Schema modifier.



In the **Name as** field, under the **Select settings** tab, change 'Rotton' to 'Rotten'. To drop the Rating column, hover over it and click on the trash can icon.



2. **Add a Filter Transformation to filter out unwanted years:** Say you are only interested in movies made after 1951. You can add a [Filter transformation](#) to specify a filter condition by clicking on the **+** icon next to your Select transformation and choosing **Filter** under Row Modifier. Click on the **expression box** to open up the [Expression builder](#) and enter in your filter condition. Using the syntax of the [Mapping Data Flow expression language](#), **toInteger(year) > 1950** will convert the string year value to an integer and filter rows if that value is above 1950.



When you clicked on **open expression builder** you can verify your condition is working properly. This will also show by a check mark in the **Filter on** textbox.

Visual expression builder

Filter1

Expression

`toInteger(year) > 1950`

+
-
*
/
||
&&
!

Expression elements

- All
- Functions
- Input schema
- Parameters
- Cached lookup

Expression values

+ Create new

- abc movie
- abc title
- abc genres
- abc year
- abc Rotten Tomato
- 123 `abs(123 numeric_value)`
- 123 `acos(123 numeric_value)`
- ANY `add(ANY first_expression, ANY second_expression)`
- `addDays(date/timestamp, ANY days to add)`
- `addMonths(date/timestamp, ANY months to add)`
- `and(first_condition, second_condition)`
- `[] array(ANY items)`

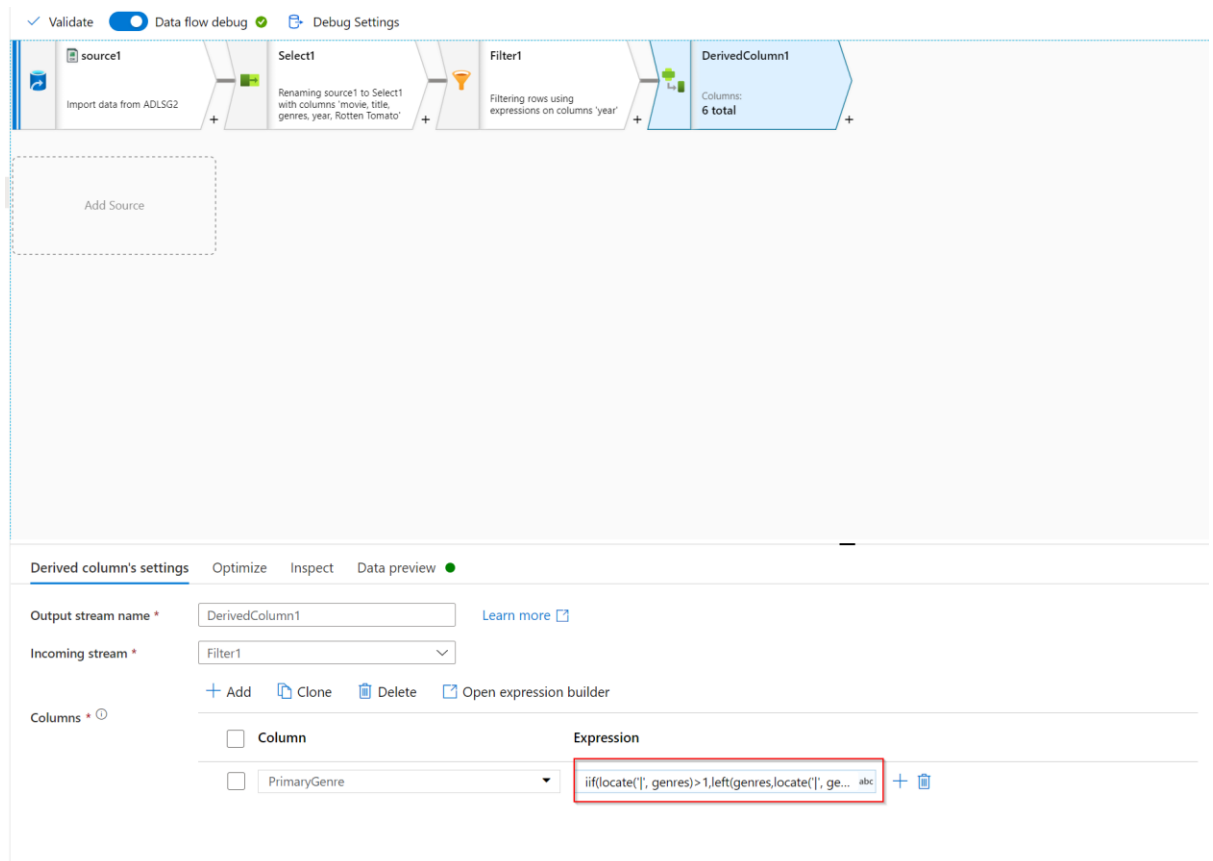
Data preview Refresh

Save and finish

Cancel

Clear contents

- Add a Derive Transformation to calculate primary genre:** As you may have noticed, the genres column is a string delimited by a '|' character. If you only care about the *first* genre in each column, you can derive a new column named **PrimaryGenre** via the **Derived Column** transformation by clicking on the **+** icon next to your Filter transformation and choosing **Derived Column** under Schema Modifier. Similar to the filter transformation, the derived column uses the Mapping Data Flow expression builder to specify the values of the new column.



In this scenario, you are trying to extract the first genre from the genres column which is formatted as 'genre1|genre2|...|genreN'. Use the **locate** function to get the first 1-based index of the '|' in the genres string. Using the **iif** function, if this index is greater than 1, the primary genre can be calculated via the **left** function which returns all characters in a string to the left of an index. Otherwise, the PrimaryGenre value is equal to the genres field. You can verify the output via the expression builder's Data preview pane.

- In the **Derived column's** settings tab, click **+Add** then, **Add column**, to add a column named **PrimaryGenre**.
 - Under **Expression** open the **Expression builder**.
 - Write `iif(locate('|', genres)>1, left(genres, locate('|', genres)-1), genres)`
 - Select **Save and finish**
4. **Rank movies via a Window Transformation** Say you are interested in how a movie ranks within its year for its specific genre. You can add a [Window transformation](#) to define window-based aggregations by clicking on the **+** icon next to your Derived Column transformation and clicking **Window** under Schema modifier. To accomplish this, specify what you are windowing over, what you are sorting by, what the range is, and how to calculate your new window columns. In this example, we will window over PrimaryGenre and year with an unbounded range, sort by Rotten Tomato descending, a calculate a

new column called RatingsRank which is equal to the rank each movie has within its specific genre-year.

- In the **Window settings** pane under the **Over** tab, select **PrimaryGenre** and add **year** by clicking on + and selecting **year** from the dropdown.

Window settings

Optimize

Inspect

Data preview

Output stream name *

Window1

?

Help

Learn more

Incoming stream *

DerivedColumn1

1. Over

2. Sort

3. Range by

4. Window columns

DerivedColumn1's column

Name as

abc PrimaryGenre

PrimaryGenre

+

abc year

year

+

- In the **Sort settings** pane, select the **Rotten Tomato** column, select **Descending** under **Order** and check **Nulls first**

Window settings

Optimize

Inspect

Data preview

Output stream name *

Window1

?

Help

Learn more

Incoming stream *

DerivedColumn1

1. Over

2. Sort

3. Range by

4. Window columns

DerivedColumn1's column

Order

Nulls first

abc Rotten Tomato

Descending

☒

+

- In the **Range by settings** pane, leave all settings per default.

Window settings Optimize Inspect Data preview ●

Output stream name * ? Help [Learn more](#)

Incoming stream * ▼

1. Over 2. Sort **3. Range by** 4. Window columns

Option * ⓘ ☒ Range by current row offset ☐ Range by column value

Unbounded ☒

- In the **Window columns settings** pane, rename the blank column to **RatingsRank** and enter as expression **rank()**

Window settings Optimize Inspect Data preview ●

Output stream name * ? Help [Learn more](#)

Incoming stream * ▼

1. Over 2. Sort 3. Range by **4. Window columns**

+ Add Clone Delete Open expression builder

<input type="checkbox"/>	Column	Expression
<input type="checkbox"/>	RatingsRank ▼	rank() 123 +

5. **Aggregate ratings with an Aggregate Transformation:** Now that you have gathered and derived all your required data, we can add an [Aggregate transformation](#) to calculate metrics based on a desired group by clicking on the **+** icon next to your Window transformation and clicking **Aggregate** under Schema modifier. As you did in the window transformation, let's group movies by PrimaryGenre and year

- Under the **Aggregate settings** tab, select **Group by**.
- Using the dropdown select the column **Primary Genre** and add the **year** column by clicking **+**, and dropdown.

Aggregate settings
Optimize
Inspect
Data preview

Output stream name *
Aggregate1
Learn more

Incoming stream *
Window1

Group by
Aggregates

Columns
Name as

abc PrimaryGenre	PrimaryGenre	+	
abc year	year	+	

In the Aggregates tab, you can aggregations calculated over the specified group by columns. For every genre and year, lets get the average Rotten Tomatoes rating, the highest and lowest rated movie (utilizing the windowing function) and the number of movies that are in each group. Aggregation significantly reduces the amount of rows in your transformation stream and only propagates the group by and aggregate columns specified in the transformation.

- Under the **Aggregate settings** tab, now select **Aggregates**. Add the following columns by clicking + and then **Add column**, with their respective expressions:
 - AverageRating: avg(toInteger({Rotten Tomato}))
 - HighestRead: first(title)
 - LowestRead: last(title)
 - NumberOfMovies: count()

Aggregate settings
Optimize
Inspect
Data preview

Output stream name *
Aggregate1
Learn more

Incoming stream *
Window1

Group by
Aggregates

Grouped by: PrimaryGenre, year

+ Add
Clone
Delete
Open expression builder

Column	Expression
AverageRating	avg(toInteger({Rotten Tomato})) 1.2
HighestRead	first(title) abc
LowestRead	last(title) abc
NumberOfMovies	count() 12L

- To see how the aggregate transformation changes your data, use the Data Preview tab
6. **Specify Upsert condition via an Alter Row Transformation** If you are writing to a tabular sink, you can specify insert, delete, update and upsert policies on rows using the [Alter Row transformation](#) by clicking on the **+** **icon** next to your Aggregate transformation and clicking **Alter Row** under Row modifier. Since you are always inserting and updating, you can specify that all rows will always be upserted.
- From the dropdown next to **Alter row conditions** in the **Alter row settings** tab, please select **Upsert if**. In the expression write **true()**

The screenshot shows the 'Alter row settings' tab with the following configuration:

- Output stream name ***: AlterRow1
- Incoming stream ***: Aggregate1
- Alter row conditions ***: Upsert if (dropdown), true() (expression field)

Task 4: Writing to a Data Sink

1. **Write to a Azure Synapse Analytics Sink:** Now that you have finished all your transformation logic, you are ready to write to a Sink.
 - i. Add a **Sink** by clicking on the **+** **icon** next to your Alter row transformation and clicking **Sink** under Destination.
 - ii. In the Sink tab, create a new data warehouse dataset via the **+** **New button** next to **Dataset**.
 - iii. Select **Azure Synapse Analytics** from the tile list and click **Continue**
 - iv. Select **+New** under **Linked service**. Configure your Azure Synapse Analytics connection to connect to the DWDB database created in Module 5.
 - v. **Account selection method: From Azure subscription**
 - vi. **Azure subscription:** select the subscription used for this lab.
 - vii. **Server name:** select your **wrkspcxx** server.
 - viii. **Database name:** **DWDB**

viii. **Authentication type: SQL authentication**

ix. For **username** use your server admin username, for **Password** use the corresponding password you provided, when setting up the service.

x. Click **Create** when finished.

New linked service (Azure Synapse Analytics)

Name *
AzureSynapseAnalytics1

Description

Connect via integration runtime * ⓘ
AutoResolveIntegrationRuntime

Connection string **Azure Key Vault**

Account selection method ⓘ
☒ From Azure subscription ☐ Enter manually

Azure subscription
[Redacted] ▼

Server name *
wrkspckdj (Synapse workspace) ▼ ↻

Database name *
DWDB ▼ ↻

Authentication type *
SQL authentication ▼

User name *
sqladminuser

Password **Azure Key Vault**


Password *

Additional connection properties
+ New

Annotations
+ New

▸ Parameters

▸ Advanced ⓘ

Create  Test connection **Cancel**

- xi. In the **Set properties** page, select **Create new table** and enter in the schema of **dbo** and the table name of **Ratings**. Click **OK** once completed.

Set properties

Name
AzureSynapseAnalyticsTable1

Linked service *
AzureSynapseAnalytics1

☐ Select from existing table ☒ Create new table

Schema and table name
dbo . Ratings

Advanced

OK Back Cancel

- xii. Since an upsert condition was specified, you need to go to the Settings tab and select **Allow upsert**.
- xiii. For **Key columns** select **List of Columns** and add through + the two columns PrimaryGenre and year. based on key columns PrimaryGenre and year.

Sink Settings Mapping Optimize Inspect Data preview ●

Update method ⓘ
☐ Allow insert
☐ Allow delete
☒ Allow upsert
☐ Allow update

Key columns * ⓘ
☒ List of columns ☐ Custom expression ⓘ

abc PrimaryGenre +

abc year +

Skip writing key columns ☐

Table action ⓘ
☒ None ☐ Recreate table ⓘ ☐ Truncate table ⓘ

Enable staging ☒

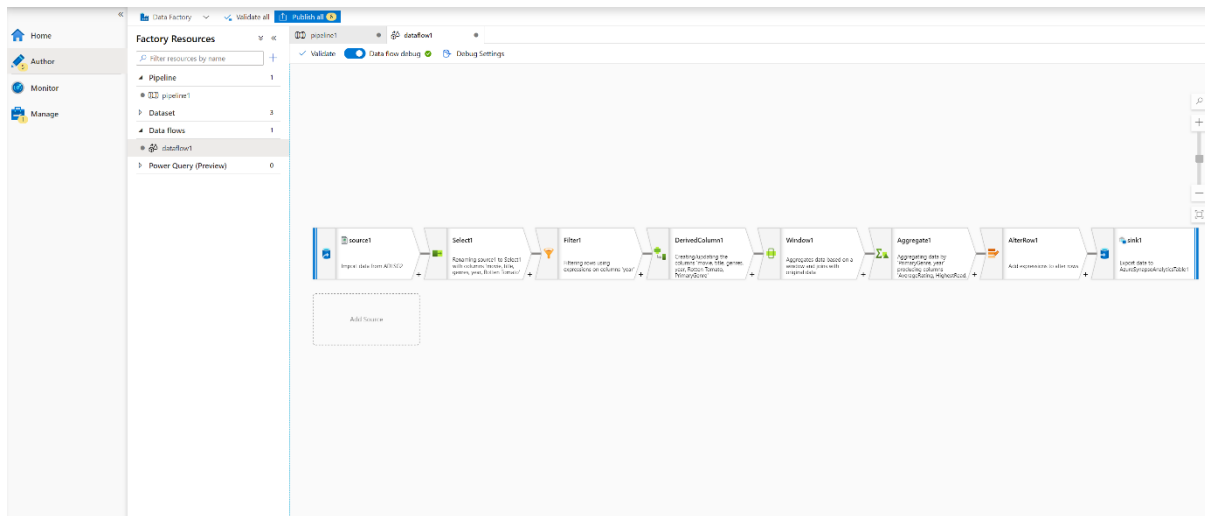
Batch size ⓘ

Pre SQL scripts ⓘ
☒ List of scripts ☐ Custom expression ⓘ

Post SQL scripts ⓘ
☒ List of scripts ☐ Custom expression ⓘ

- xiv. In the **Mapping** pane make sure you untick **Auto mapping**.

At this point, You have finished building your 8 transformation Mapping Data Flow. It's time to run the pipeline and see the results!



Task 5: Running the Pipeline

1. Go to the pipeline1 tab in the canvas. Because Azure Synapse Analytics in Data Flow uses [PolyBase](#), you must specify a blob or ADLS staging folder. In the **Settings** tab of the data flow, open up the **Staging** accordion and select your ADLS linked service and specify a staging folder path such as **data/dw-staging**.

General Settings Parameters¹ User properties

Data flow * dataflow1 [Open](#) [New](#)

Run on (Azure IR) * ① AutoResolveIntegrationRuntime

Compute type * General purpose
[Add dynamic content \[Alt+Shift+D\]](#)

Core count * 4 (+ 4 Driver cores)

Logging level * ① ☒ Verbose ☐ Basic ☐ None

▶ Sink properties

▲ Staging ①

Staging linked service ① AzureDataLakeStorage1 [Test connection](#) [Edit](#) [New](#)

Staging storage folder data / dw-staging [Browse](#) ▼

2. Before you publish your pipeline, run another debug run to confirm it's working as expected. Looking at the **Output** tab, you can monitor the status of both activities as they are running.

3. Once both activities succeeded, you can click on the eyeglasses icon next to the Data Flow activity to get a more in depth look at the Data Flow run.
4. If you used the same logic described in this lab, your Data Flow should have written 737 rows to your SQL DW.

```
1 Select count(*) as TotalCount from dbo.Ratings
2 |
```

Results Messages

View

Table

Chart

Export results

Search

TotalCount

737