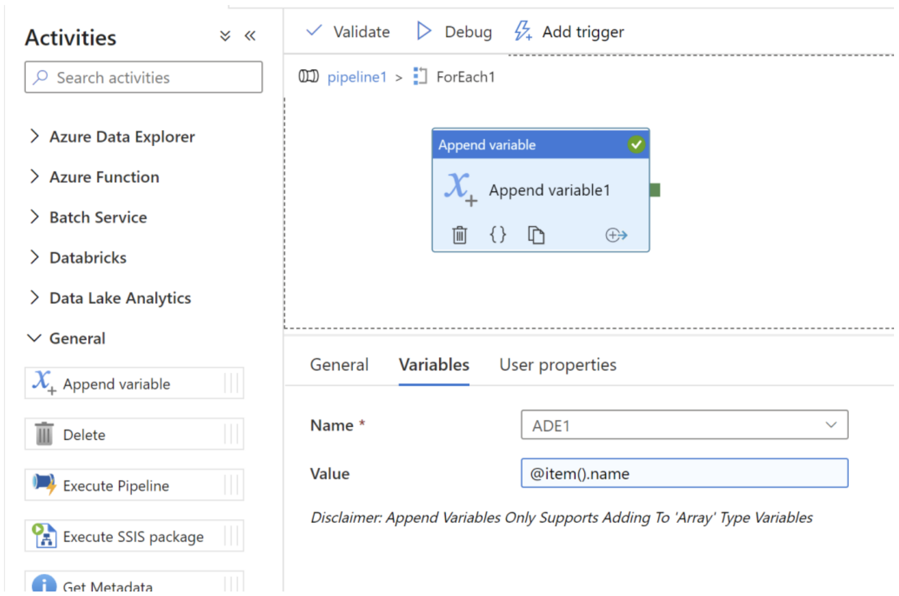
Append Variable Activity : Azure Data Factory



Name * ADE1

Value @item().name

Disclaimer: Append Variables Only Supports Adding To 'Array' Type Variables

We have looked at the *Set Variable* activity and an example of how use Set Variable to increment a variable in a previous post. There is another activity which interacts with variables in *Data Factory*, the *Append Variable* activity. In this post, we will discuss the *Append Variable* activity and see how it is different from the *Set Variable* activity.

To understand how the *Append Variable* activity works in a better way, we must have a basic understanding of variables in *Data Factory.* Data Factory supports three variable types:

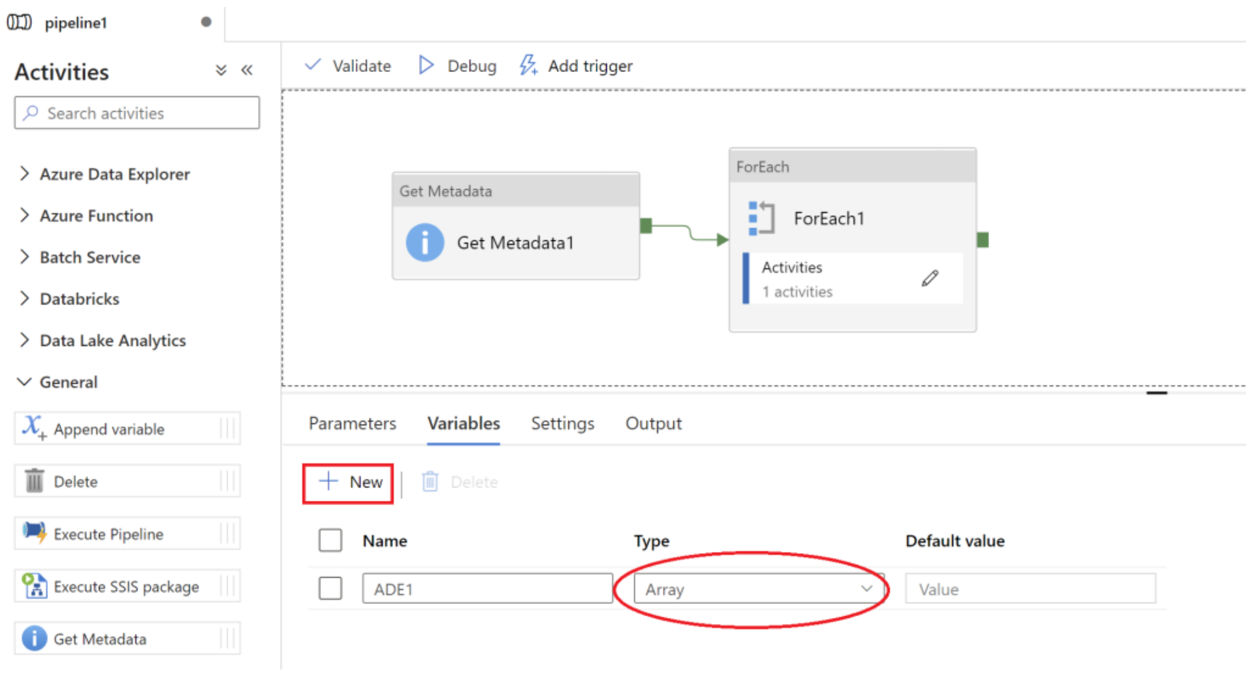
1. **String (Text)**
2. **Boolean (Binary e.g. Y/N)**
3. **Array (An Array Object)**

If you know data structure basics, you must know that an Array is a collection of elements of similar data type and each position in the array can be addressed using a number.

Append Variable activity can only be used with an Array type variable. As the name suggests, *Append variable* activity appends new items to the next positions in the Array.

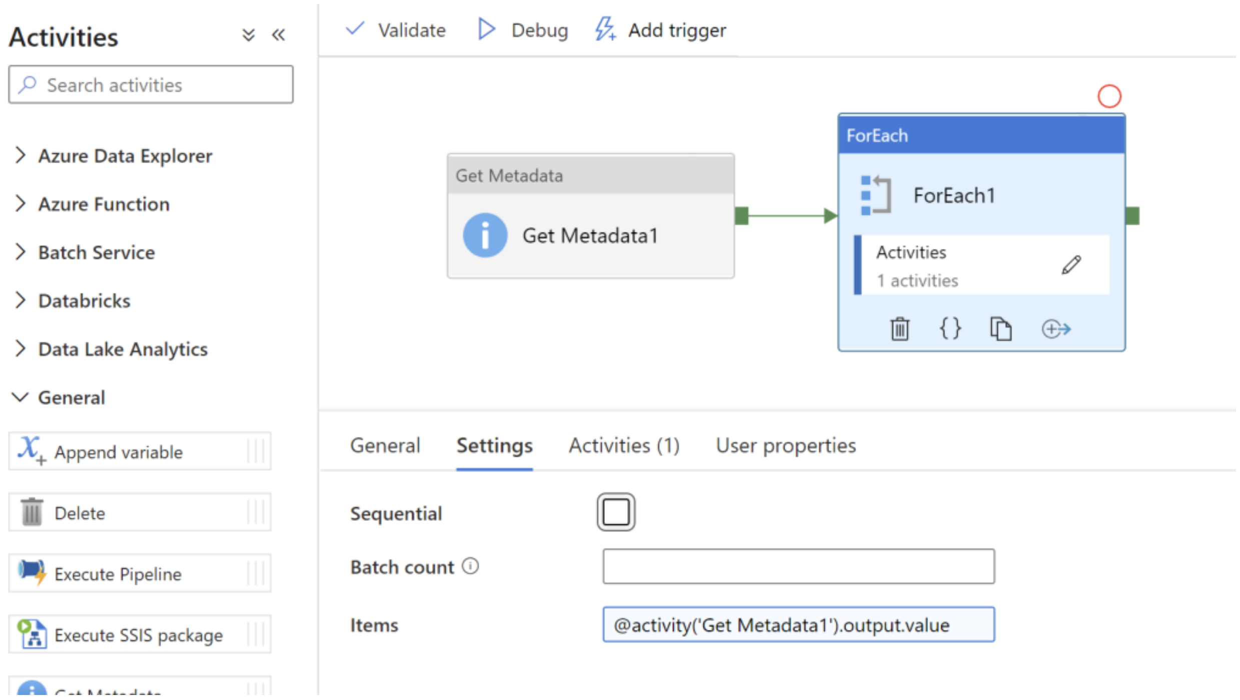
Let's consider a scenario where we would like to get a list of filenames from a folder and load it into an array type variable for further processing.

Firstly, we must create an array type variable, that we will be using to load the File name list. This can be done in the Variables tab and selecting the type as *Array.*



Name	Type	Default value
ADE1	Array	Value

Like the example in the previous post about loading filenames into an SQL table, we will use a *Get Metadata* activity to get the list of files from the folder. But instead of using a *Stored Procedure* Activity, we will use a *For Each* and *Append Variable* activities to process the file names.



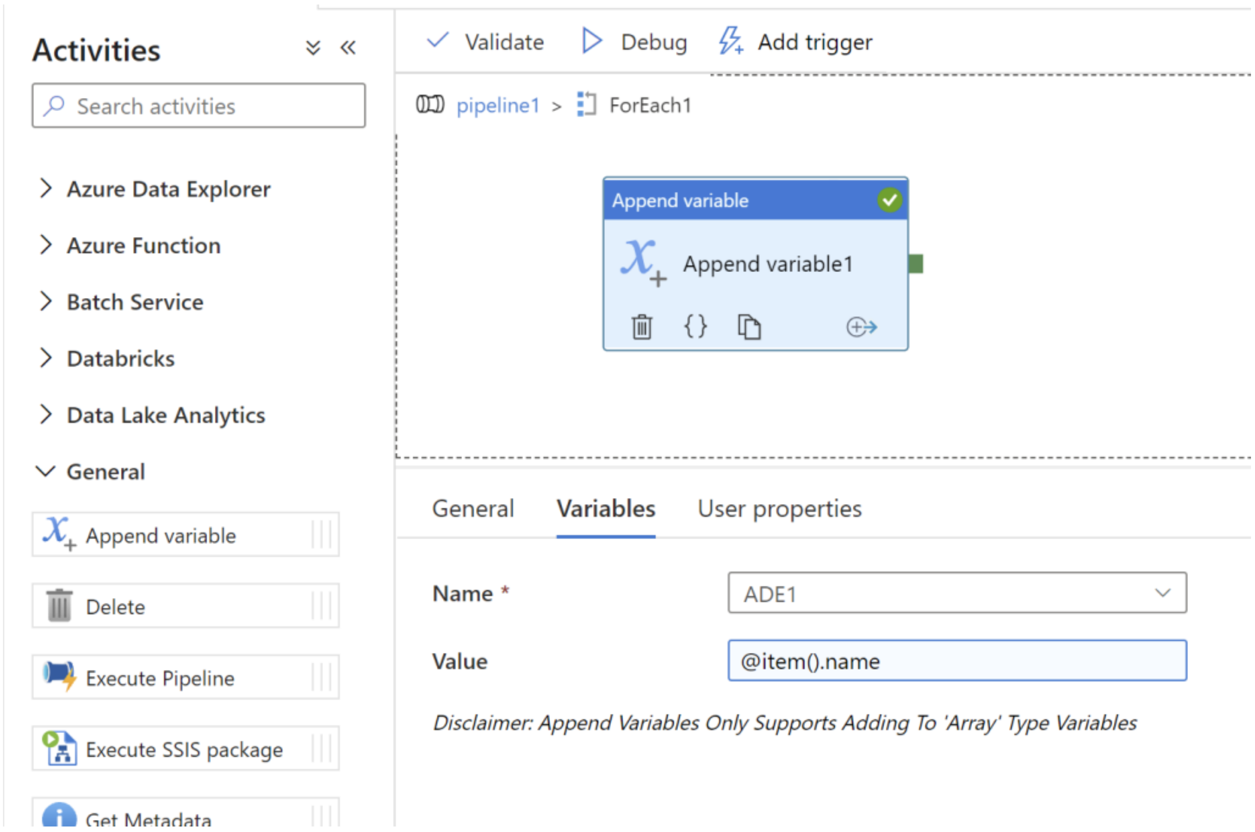
Sequential: ☐
Batch count:
Items: @activity('Get Metadata1').output.value

In the Items setting for the For Each activity, use the following code to access the Get Metadata activity output.

@activity('Get Metadata1').output.value

Next, inside the *For Each* activity, we create an *Append Variable* activity and select the ADE1 array variable that we created above. Use the code below in the value field to append the Filename to the array variable.

@item().name

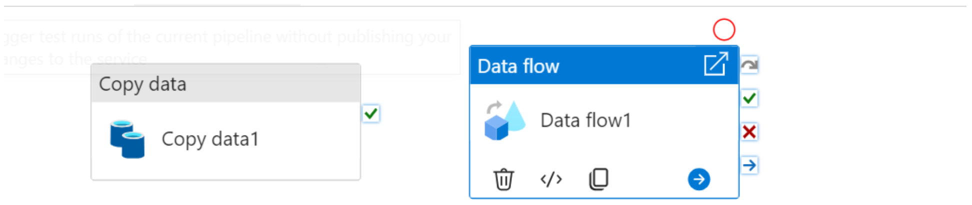


Name * ADE1

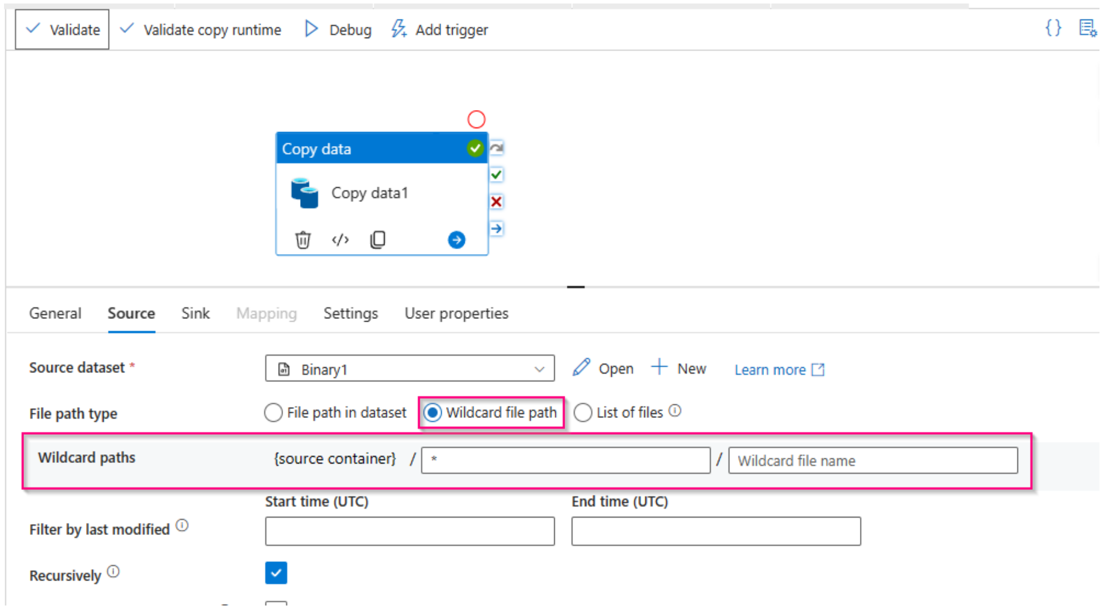
Value @item().name

Disclaimer: Append Variables Only Supports Adding To 'Array' Type Variables

Please note, this example will just load the variable ADE1 with the Filenames from the *Get Metadata* activity. More activities can be added to the control flow to access the filenames using the array variable.



COPY Activity



Source dataset * Binary1

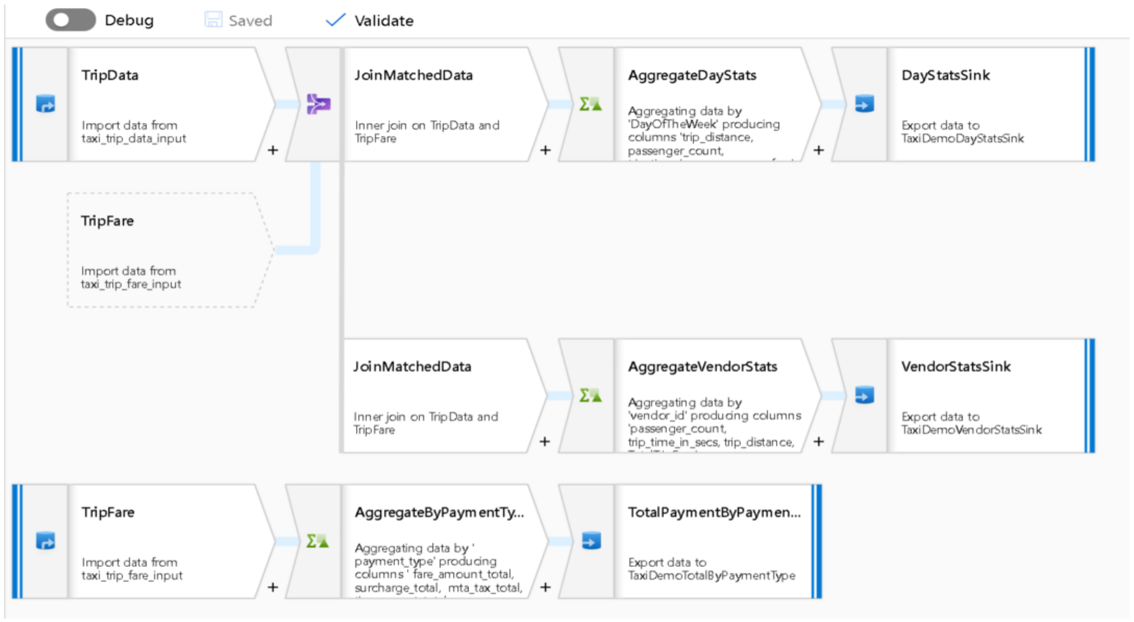
File path type ( ) File path in dataset (●) Wildcard file path ( ) List of files

Wildcard paths {source container} / * / Wildcard file name

Filter by last modified — Start time (UTC) — End time (UTC)

Recursively ✓

DATAFLOW ACTIVITY



TripData — Import data from taxi_trip_data_input

JoinMatchedData — Inner join on TripData and TripFare

AggregateDayStats — Aggregating data by 'DayOfTheWeek' producing columns 'trip_distance, passenger_count,

DayStatsSink — Export data to TaxiDemoDayStatsSink

TripFare — Import data from taxi_trip_fare_input

JoinMatchedData — Inner join on TripData and TripFare

AggregateVendorStats — Aggregating data by 'vendor_id' producing columns 'passenger_count, trip_time_in_secs, trip_distance,

VendorStatsSink — Export data to TaxiDemoVendorStatsSink

TripFare — Import data from taxi_trip_fare_input

AggregateByPaymentTy... — Aggregating data by 'payment_type' producing columns ' fare_amount_total, surcharge_total, mta_tax_total,

TotalPaymentByPaymen... — Export data to TaxiDemoTotalByPaymentType

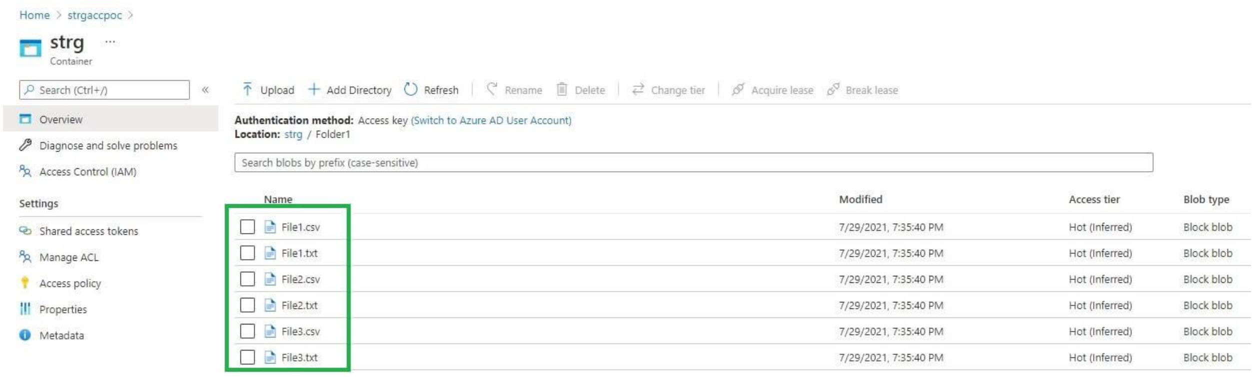
Working with the Delete Activity in Azure Data Factory

We can use the delete activity in Azure Data Factory to delete files from both on-premises and cloud storage. In this article, we will discuss the delete activity with the various options available for the file deletion. I will show the following operations with the Delete Activity:

- Delete files from a folder.
- Delete contents in the folder and folder itself.
- Delete specific file types in a folder.
- Delete a single file using a wildcard file name.
- Filter files using the last modified date.
- Delete a set of files in a folder.
- Deleting files from the subfolder.
- Logging in Delete Activity.

Delete files from a folder

In this example, We will delete all the files from Folder1. I have six files in the data lake folder. There are 3 csv and 3 text files:



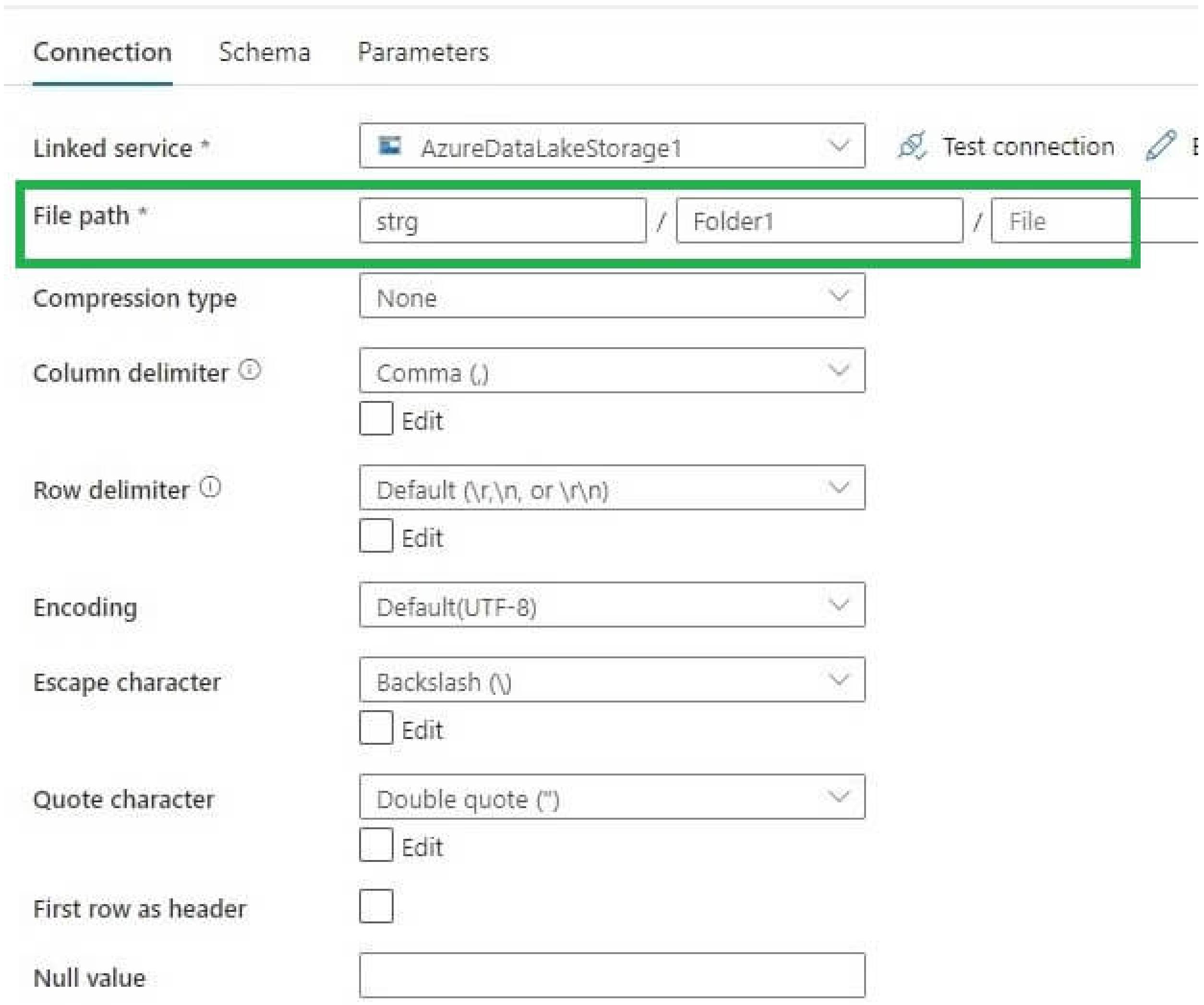
Authentication method: Access key (Switch to Azure AD User Account)
Location: strg / Folder1

Name	Modified	Access tier	Blob type
File1.csv	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File1.txt	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File2.csv	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File2.txt	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File3.csv	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File3.txt	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob

Let's take a look at the dataset properties below:

DelimitedText

DS_DELETE

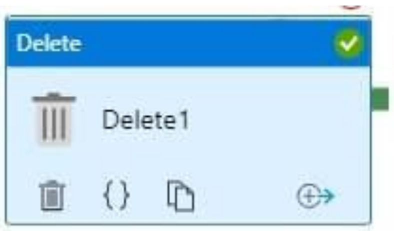


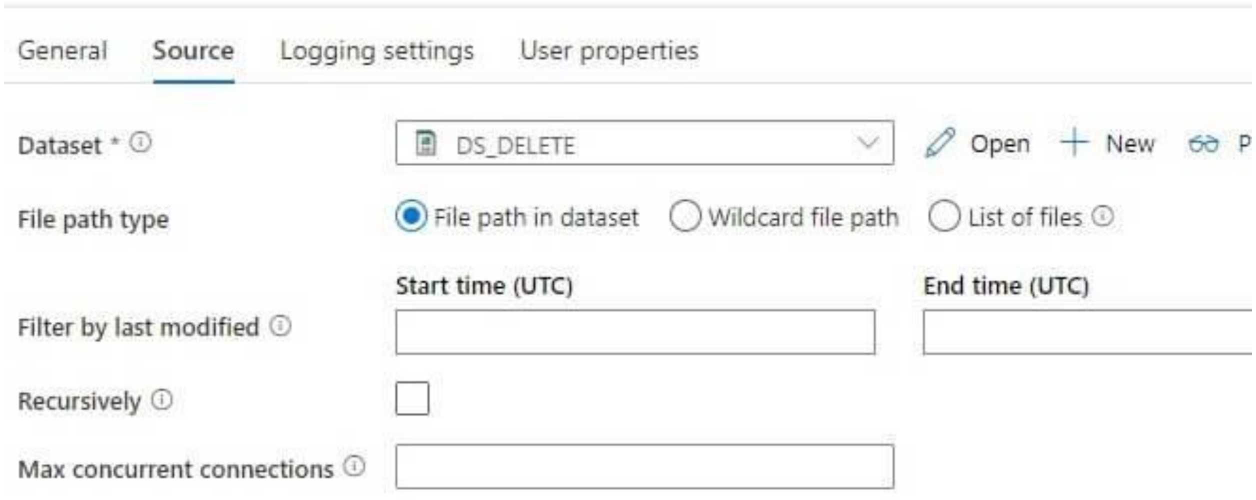
Linked service *	AzureDataLakeStorage1
File path *	strg / Folder1 / File
Compression type	None
Column delimiter	Comma (,) ☐ Edit
Row delimiter	Default (\r,\n, or \r\n) ☐ Edit
Encoding	Default(UTF-8)
Escape character	Backslash (\) ☐ Edit
Quote character	Double quote (") ☐ Edit
First row as header	☐
Null value	

The delete activity has these options in the source tab:

- **Dataset** - We need to provide a dataset that points to a file or a folder.
- **File Pathtype -** It has three options:
  - **Filepath in dataset** - With this option source file will be selected from the dataset.
  - **Wildcard file path** - We need to select this option when we want to delete source files with wildcard file names.
  - **List of files** - We can use a list of filenames to delete from the source folder.
- **Filter by last modified** - We can optionally files can be filtered by start time and end time.
- **Recursively** - This option determines whether files need to delete from the current folder or the subfolder.
- **Max concurrent connections** - Here we can specify a maximum number of the connections to delete files or a folder.

Now in the file path type option below we will use the file path in dataset option so that it will delete all the files in folder1.





Dataset * DS_DELETE

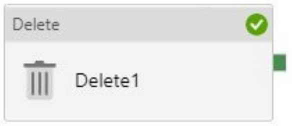
File path type [✓ File path in dataset] Wildcard file path List of files

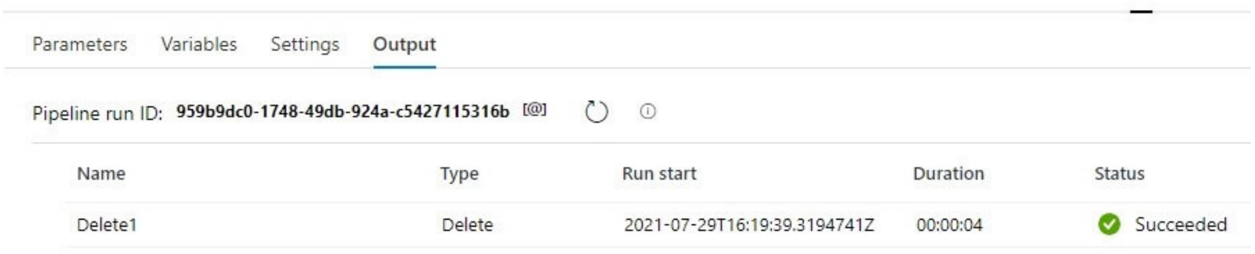
Filter by last modified Start time (UTC) End time (UTC)

Recursively ☐

Max concurrent connections

Let's run the pipeline:

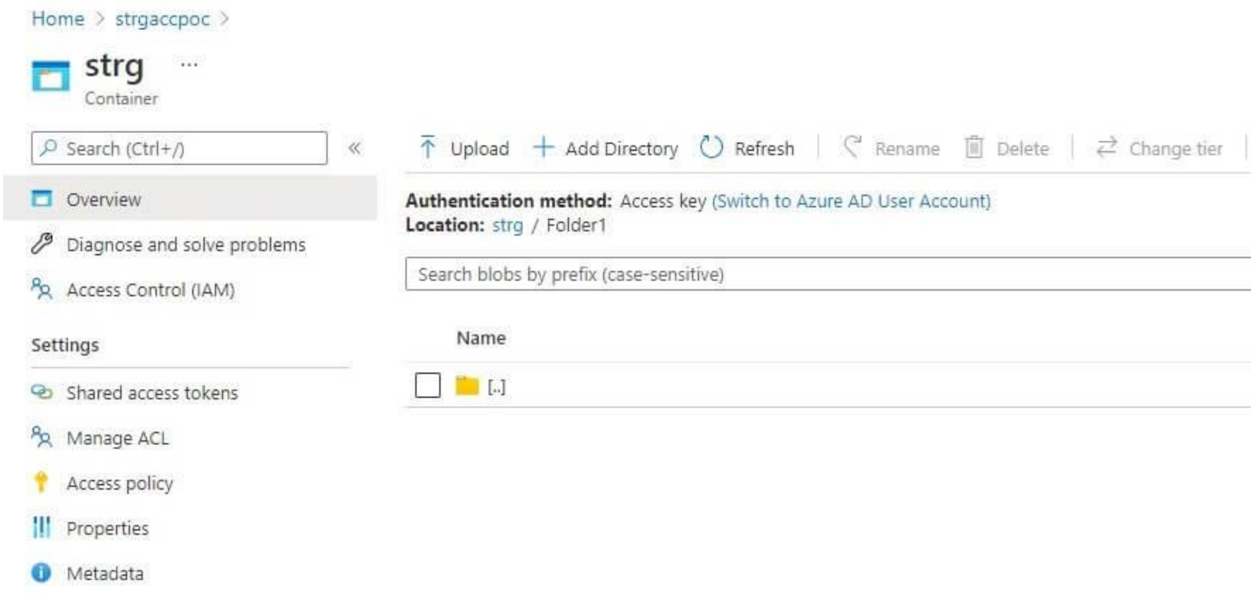




Pipeline run ID: 959b9dc0-1748-49db-924a-c5427115316b

Name	Type	Run start	Duration	Status
Delete1	Delete	2021-07-29T16:19:39.3194741Z	00:00:04	Succeeded

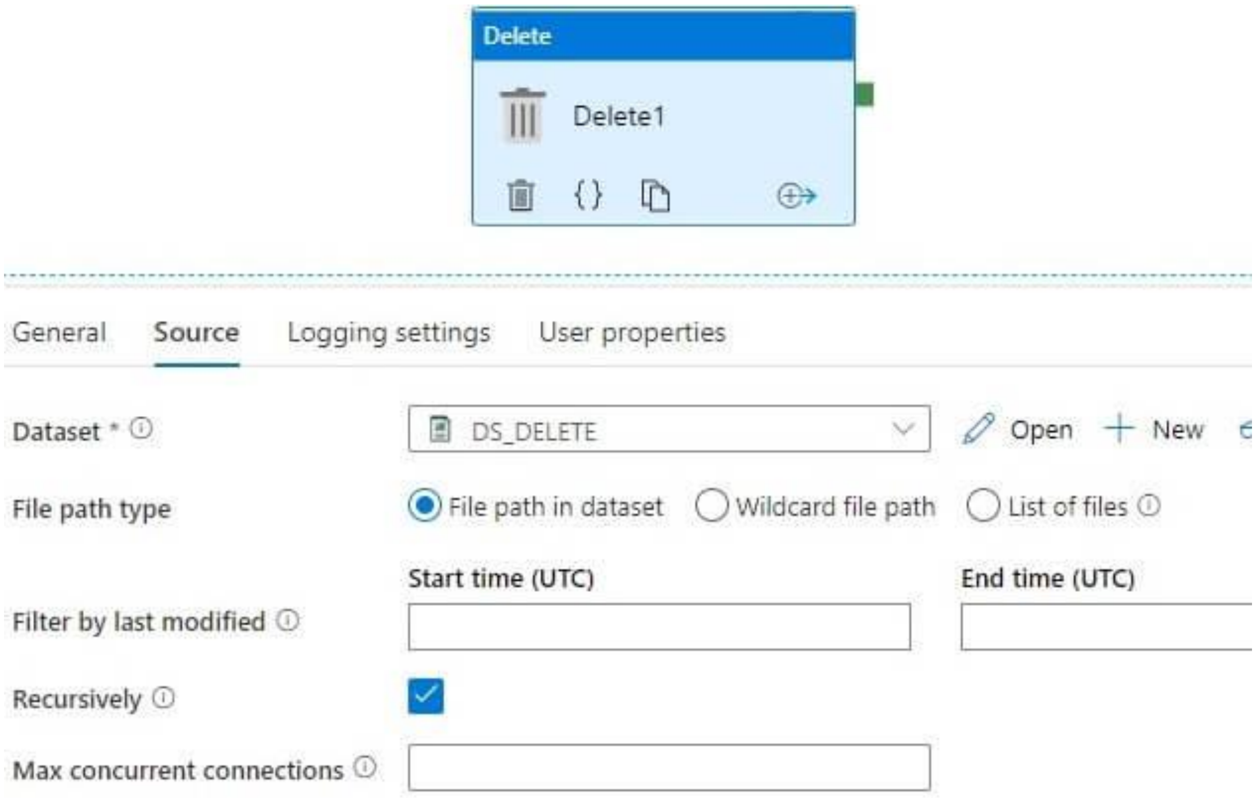
As expected all files are deleted from folder1 but not the folder, Folder1, itself.



Authentication method: Access key (Switch to Azure AD User Account)
Location: strg / Folder1

Delete contents in folder and folder itself

In this example, we will delete all contents in the folder and the folder itself. To delete the folder and its content, we need to check the recursively option, as shown below:



Dataset *: DS_DELETE

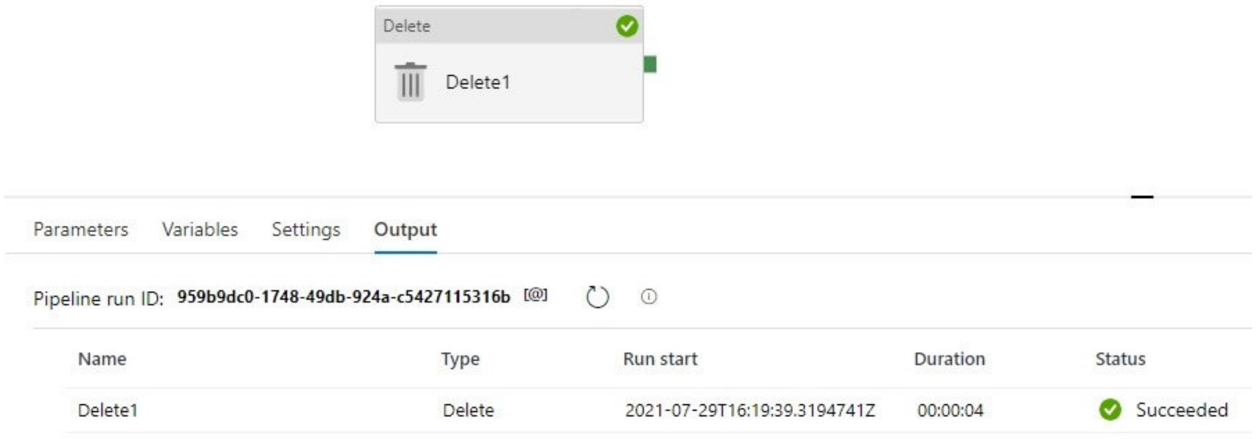
File path type: [✓ File path in dataset] Wildcard file path List of files

Filter by last modified: Start time (UTC) End time (UTC)

Recursively: ✓

Max concurrent connections:

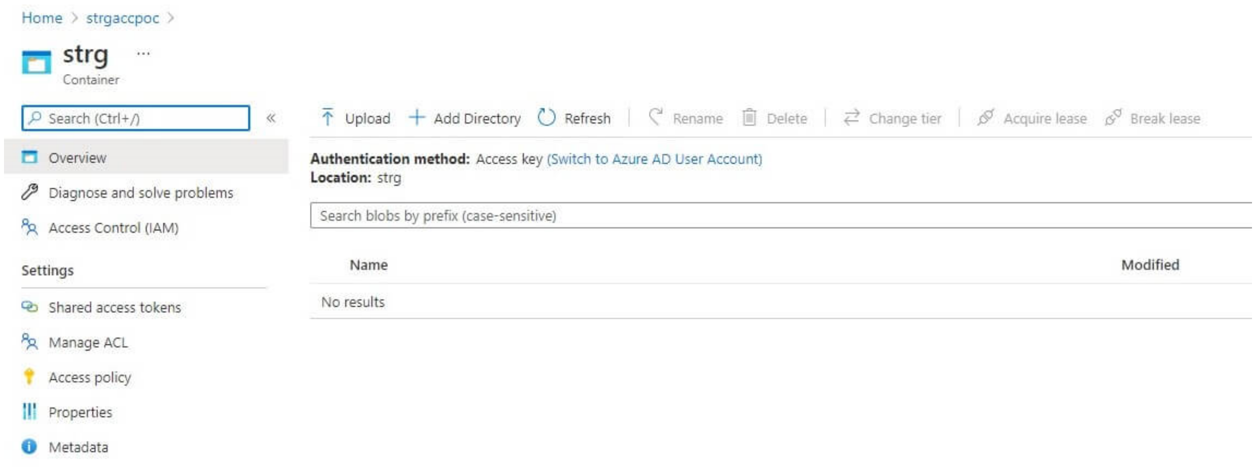
Let's run the pipeline:



Pipeline run ID: 959b9dc0-1748-49db-924a-c5427115316b

Name	Type	Run start	Duration	Status
Delete1	Delete	2021-07-29T16:19:39.3194741Z	00:00:04	✓ Succeeded

This time Folder1 is deleted along with all its content:

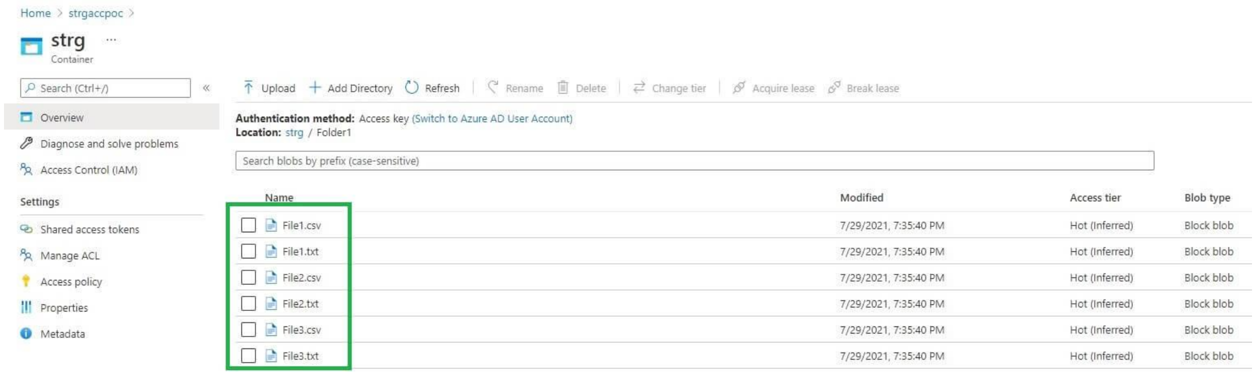


Authentication method: Access key (Switch to Azure AD User Account)
Location: strg

Name	Modified
No results	

Delete specific file types in a folder

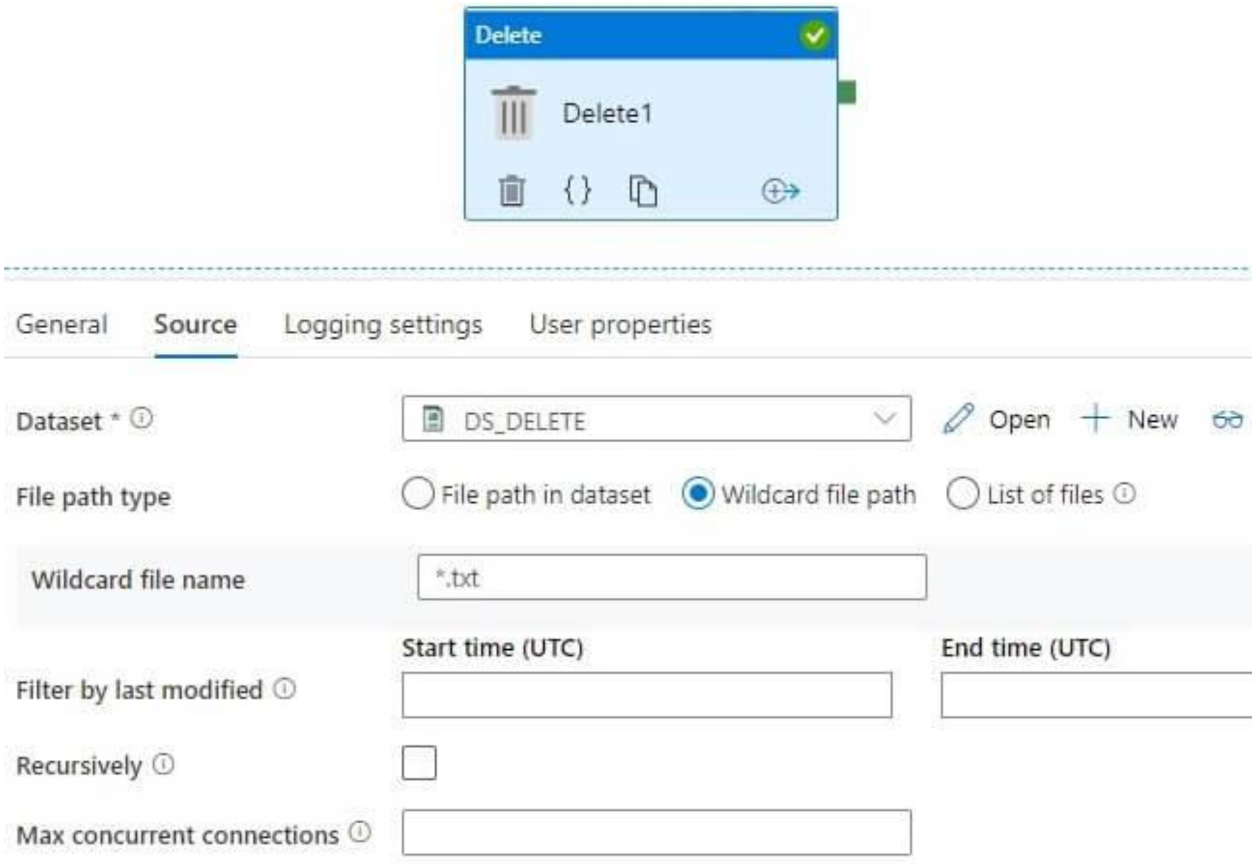
In this example, we will delete only text files from the source folder. I reset the source files in the data set. Now I have 6 files,3 csv files, and 3 text files.



Authentication method: Access key (Switch to Azure AD User Account)
Location: strg / Folder1

Name	Modified	Access tier	Blob type
File1.csv	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File1.txt	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File2.csv	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File2.txt	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File3.csv	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File3.txt	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob

Let's change the delete activity source settings, using a wildcard. In the wildcard file name, we are using *.txt to delete all text files:



Dataset * DS_DELETE

File path type: File path in dataset [✓ Wildcard file path] List of files

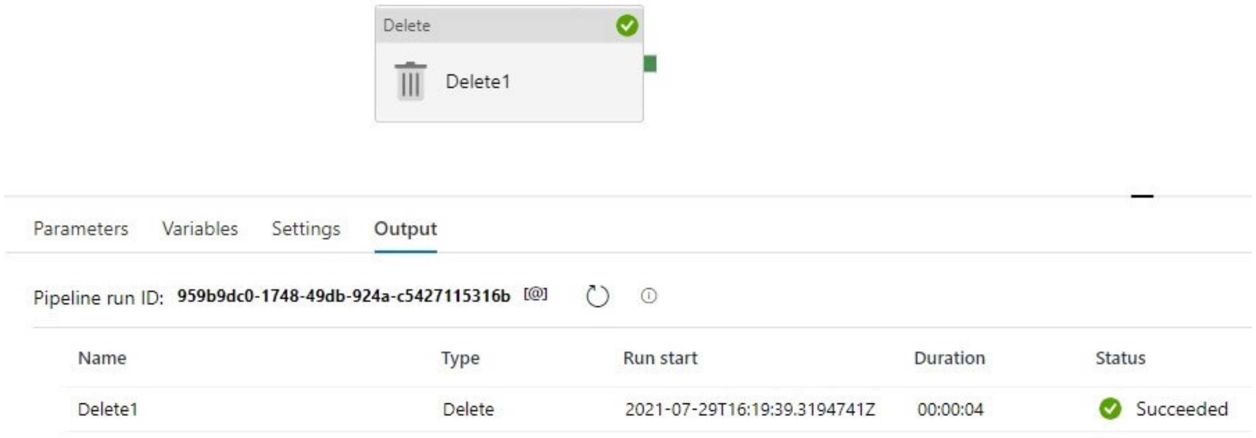
Wildcard file name: *.txt

Filter by last modified: Start time (UTC) / End time (UTC)

Recursively

Max concurrent connections

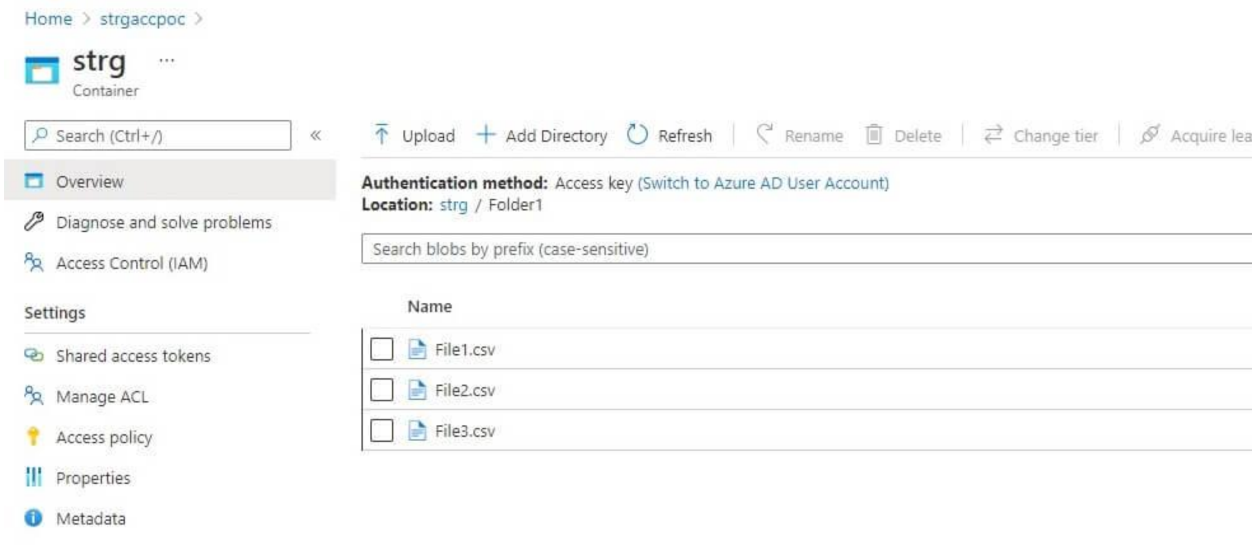
Let's run the pipeline:



Pipeline run ID: 959b9dc0-1748-49db-924a-c5427115316b

Name	Type	Run start	Duration	Status
Delete1	Delete	2021-07-29T16:19:39.3194741Z	00:00:04	Succeeded

Now all three text files are deleted from the folder Folder1 and the other 3 files are still there :



Home > strgaccpoc >

strg ···
Container

Search (Ctrl+/) «

Upload | Add Directory | Refresh | Rename | Delete | Change tier | Acquire lea

Overview
Diagnose and solve problems
Access Control (IAM)

Settings
Shared access tokens
Manage ACL
Access policy
Properties
Metadata

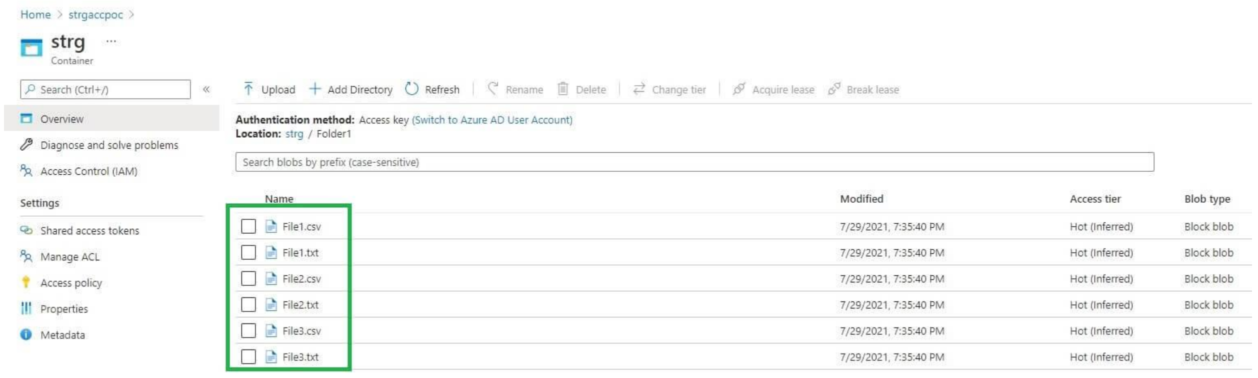
Authentication method: Access key (Switch to Azure AD User Account)
Location: strg / Folder1

Search blobs by prefix (case-sensitive)

Name
File1.csv
File2.csv
File3.csv

Delete a single file using a wildcard file name

In this example, we will delete a single file using a wildcard file name. I reset the source files in the data set. Now I have 6 files,3 csv files, and 3 text files.



Home > strgaccpoc >

strg ···
Container

Search (Ctrl+/) «

Upload | Add Directory | Refresh | Rename | Delete | Change tier | Acquire lease | Break lease

Overview
Diagnose and solve problems
Access Control (IAM)

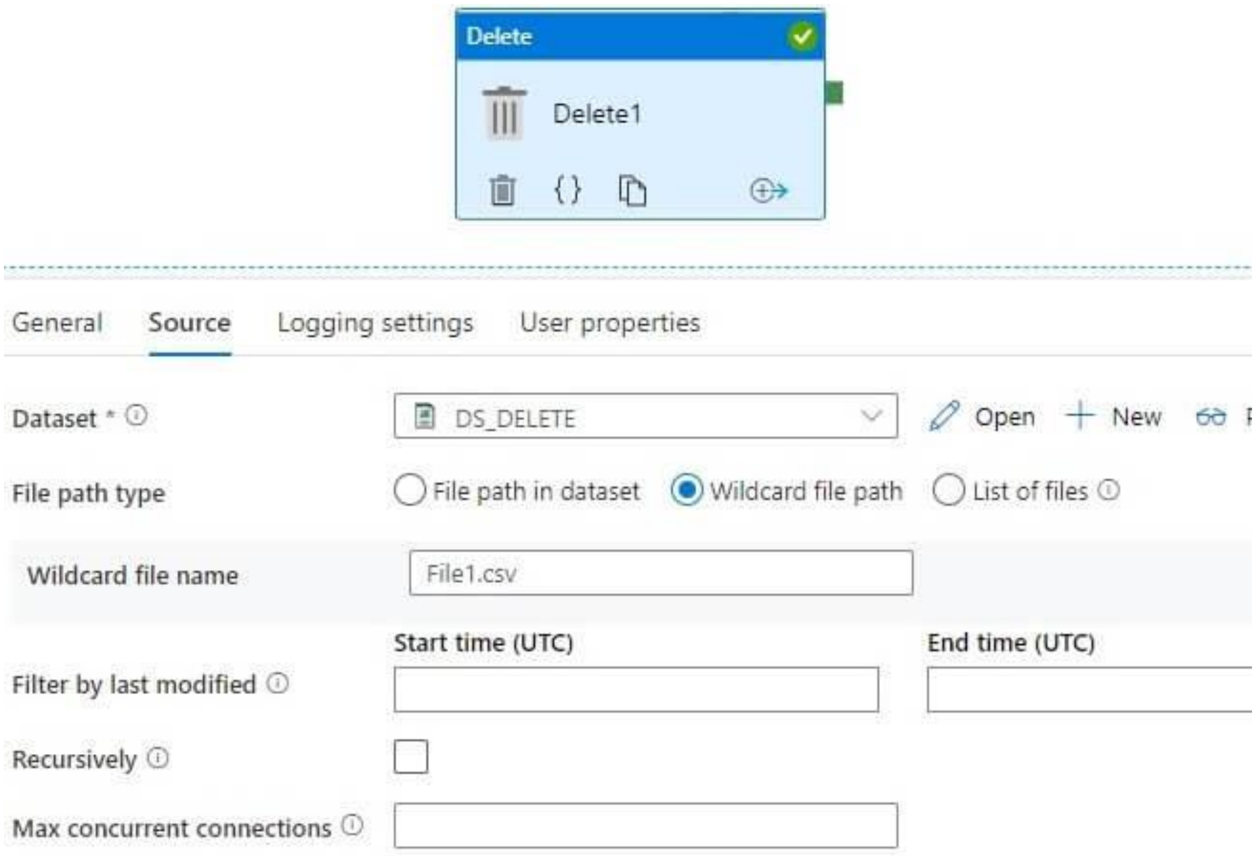
Settings
Shared access tokens
Manage ACL
Access policy
Properties
Metadata

Authentication method: Access key (Switch to Azure AD User Account)
Location: strg / Folder1

Search blobs by prefix (case-sensitive)

Name	Modified	Access tier	Blob type
File1.csv	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File1.txt	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File2.csv	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File2.txt	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File3.csv	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob
File3.txt	7/29/2021, 7:35:40 PM	Hot (Inferred)	Block blob

I enter Filename File1.csv in the Wildcard file path:



Dataset * DS_DELETE

File path type: ( ) File path in dataset [✓ Wildcard file path] ( ) List of files

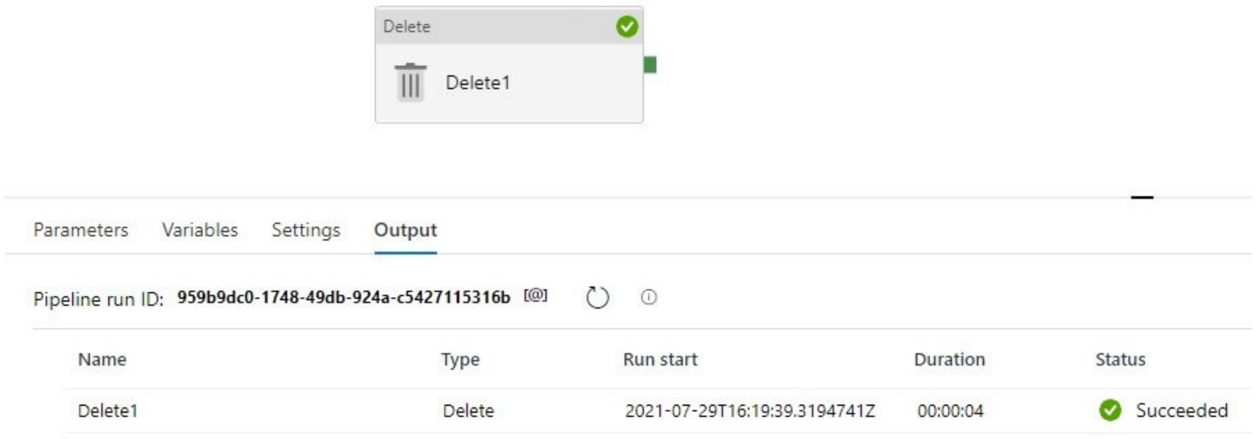
Wildcard file name: File1.csv

Filter by last modified: Start time (UTC) End time (UTC)

Recursively: ☐

Max concurrent connections

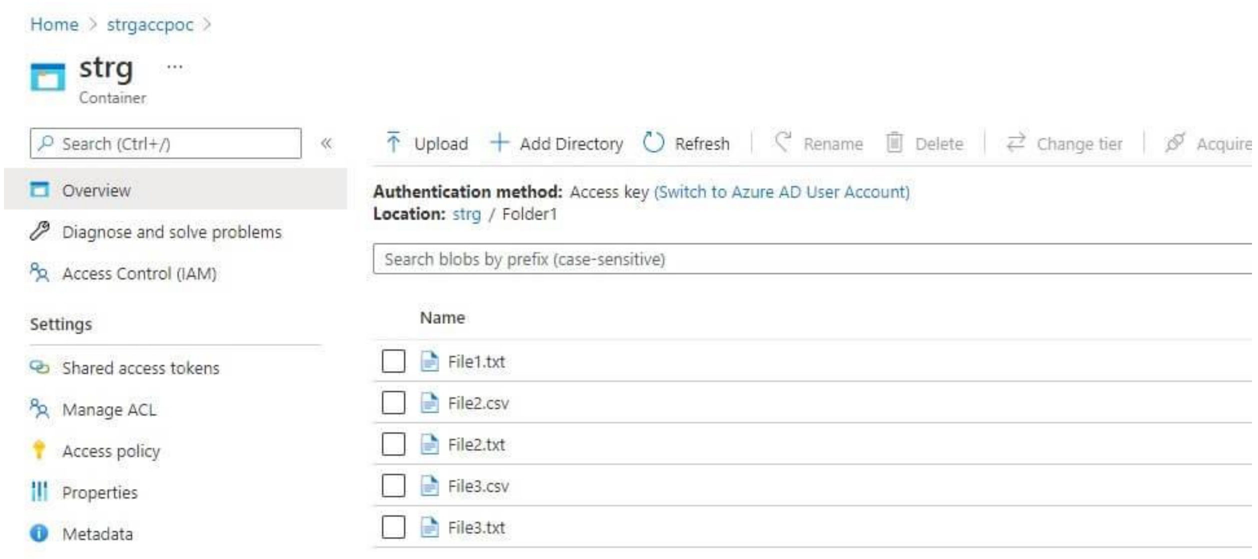
Now, I run the pipeline :



Pipeline run ID: 959b9dc0-1748-49db-924a-c5427115316b

Name	Type	Run start	Duration	Status
Delete1	Delete	2021-07-29T16:19:39.3194741Z	00:00:04	✓ Succeeded

As expected, File1.csv has been deleted and the five files are still available in the data lake folder:


Home > strgaccpoc >

strg ···
Container

Search (Ctrl+/)

Overview
Diagnose and solve problems
Access Control (IAM)

Settings
Shared access tokens
Manage ACL
Access policy
Properties
Metadata

Upload | Add Directory | Refresh | Rename | Delete | Change tier | Acquire

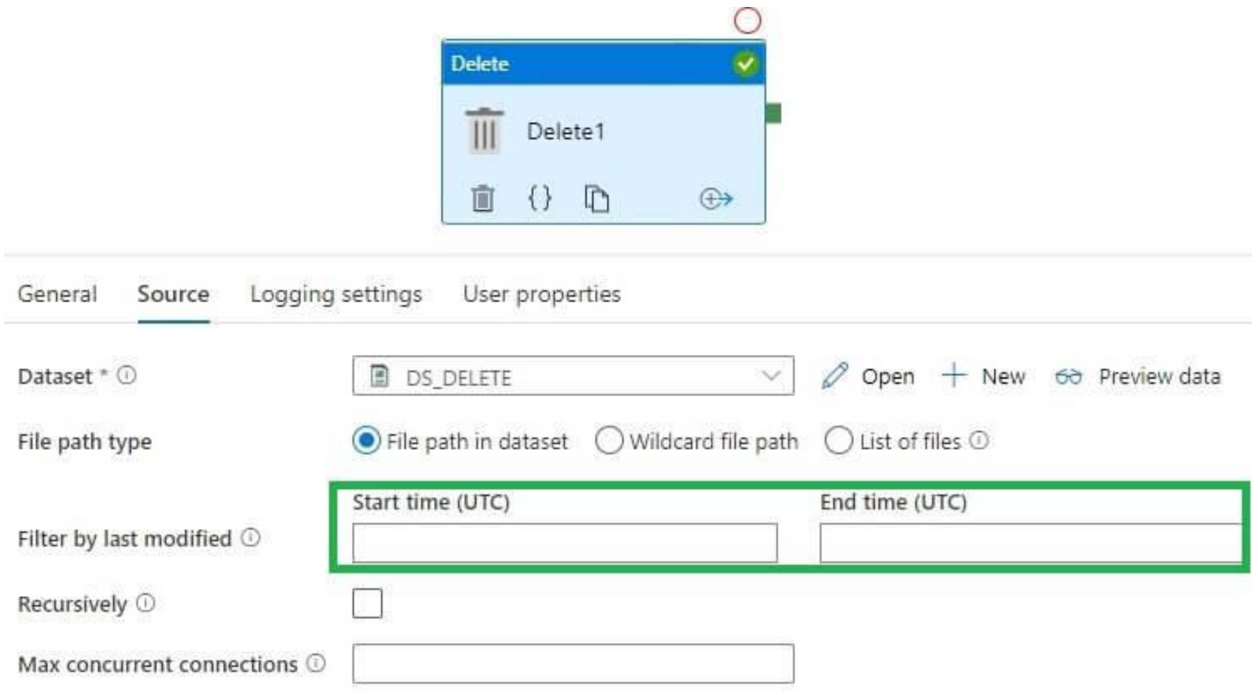
Authentication method: Access key (Switch to Azure AD User Account)
Location: strg / Folder1

Search blobs by prefix (case-sensitive)

Name
File1.txt
File2.csv
File2.txt
File3.csv
File3.txt

Filter files using the last modified date

There is an option available to filter files by start time and end time :

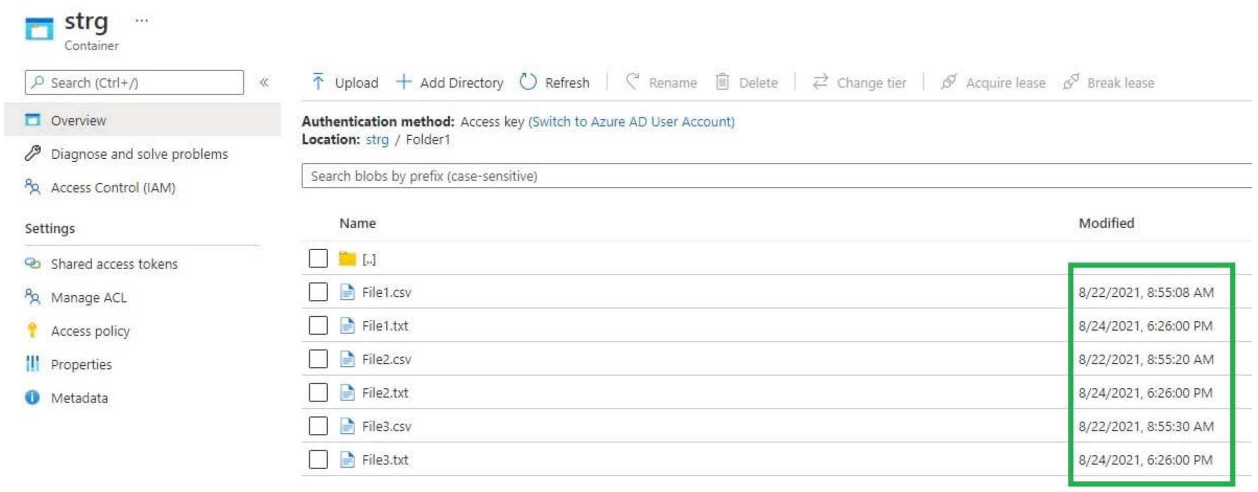

Delete

Delete1

General | Source | Logging settings | User properties

Dataset *	DS_DELETE
File path type	[✓ File path in dataset] Wildcard file path List of files
Filter by last modified	Start time (UTC): ___ End time (UTC): ___
Recursively	☐
Max concurrent connections	

In folder, Folder1, three of the files are from 22nd August and three of the files are from 24th August.



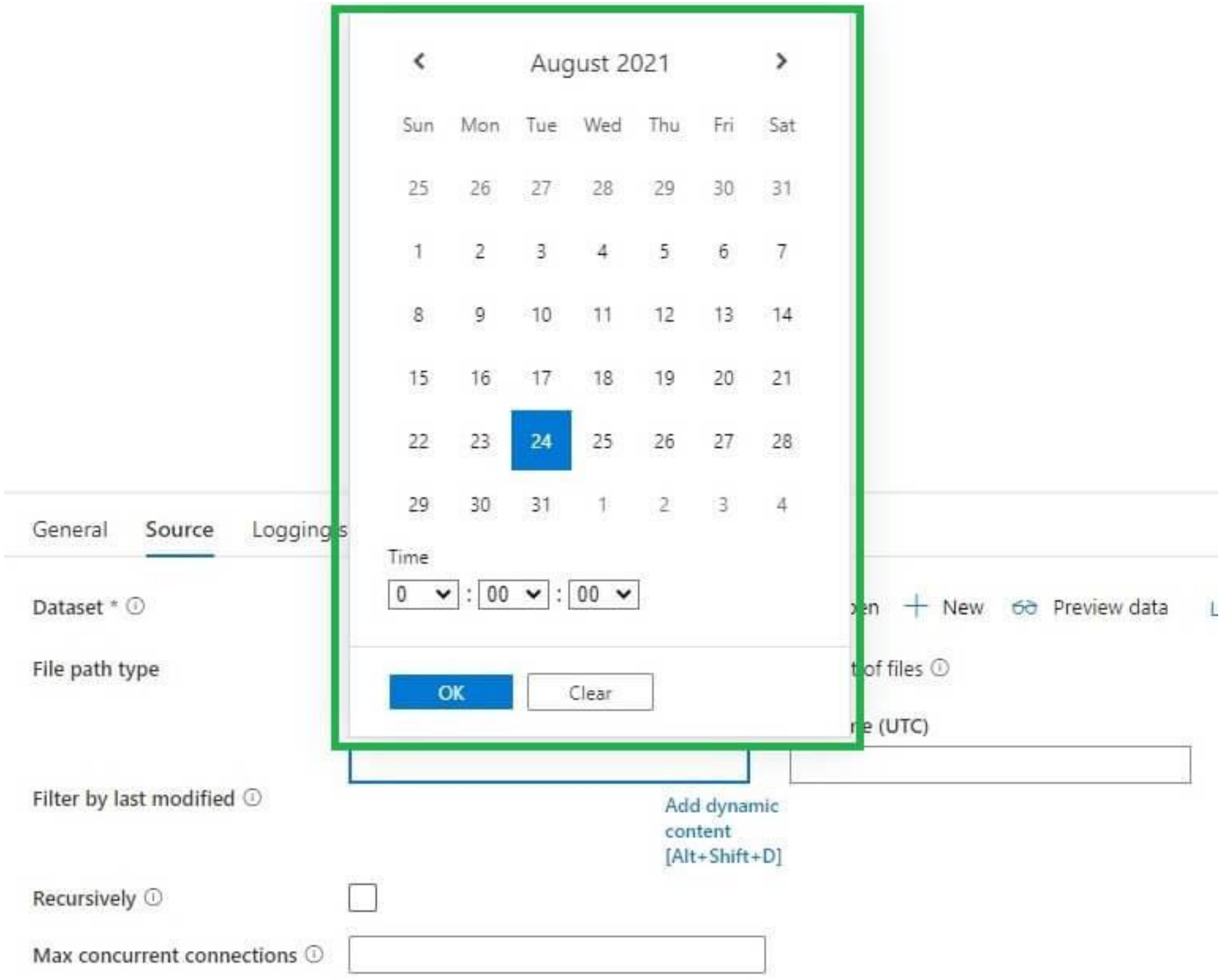
strg ···
Container

Upload | Add Directory | Refresh | Rename | Delete | Change tier | Acquire lease | Break lease

Authentication method: Access key (Switch to Azure AD User Account)
Location: strg / Folder1

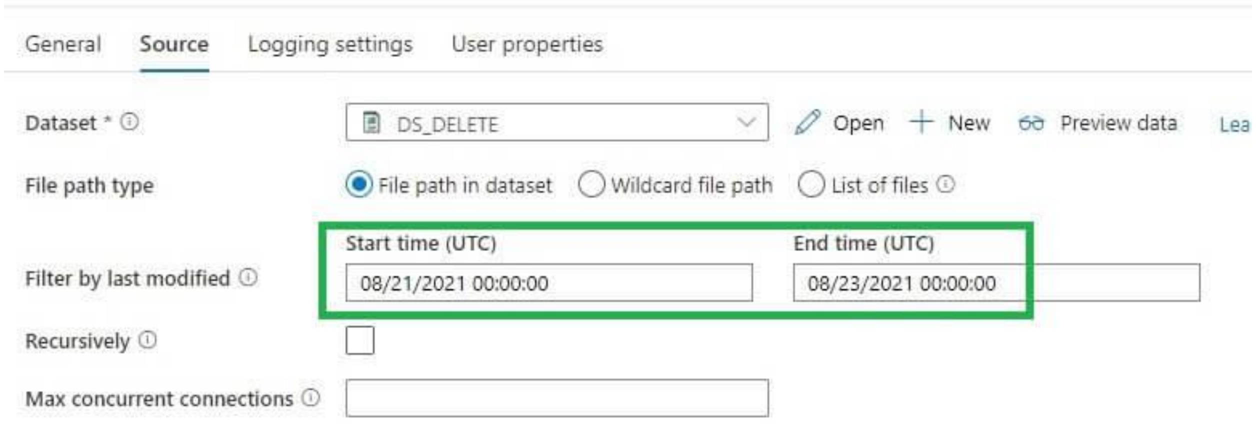
Name	Modified
[..]	
File1.csv	8/22/2021, 8:55:08 AM
File1.txt	8/24/2021, 6:26:00 PM
File2.csv	8/22/2021, 8:55:20 AM
File2.txt	8/24/2021, 6:26:00 PM
File3.csv	8/22/2021, 8:55:30 AM
File3.txt	8/24/2021, 6:26:00 PM

Let's use the filter by last modified date in the dataset, When we click on the box, it will open a calendar to select date and time.



Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

I have selected 08/21/2021 as the start date and 08/23/2021 as the end date. So, three files will be deleted as per this date filter.



Dataset * DS_DELETE

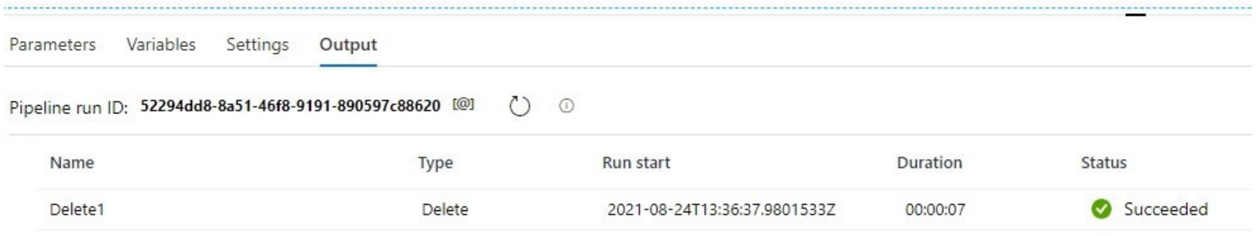
File path type: [✓ File path in dataset] Wildcard file path List of files

Filter by last modified: Start time (UTC) 08/21/2021 00:00:00 End time (UTC) 08/23/2021 00:00:00

Recursively ☐

Max concurrent connections

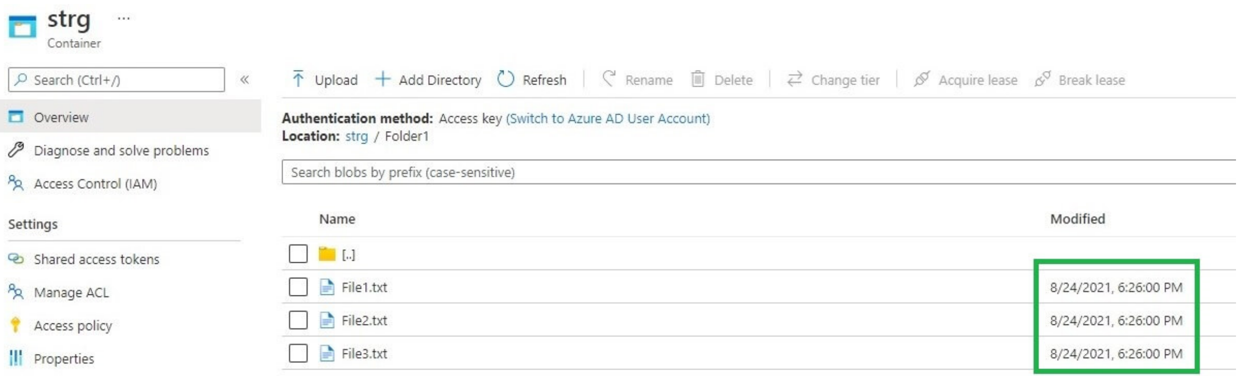
Now running the pipeline:



Pipeline run ID: 52294dd8-8a51-46f8-9191-890597c88620

Name	Type	Run start	Duration	Status
Delete1	Delete	2021-08-24T13:36:37.9801533Z	00:00:07	Succeeded

As expected three files are deleted from the folder:



Authentication method: Access key (Switch to Azure AD User Account)
Location: strg / Folder1

Name	Modified
[..]	
File1.txt	8/24/2021, 6:26:00 PM
File2.txt	8/24/2021, 6:26:00 PM
File3.txt	8/24/2021, 6:26:00 PM

ForEach activity in Azure Data Factory

The ForEach activity defines a repeating control flow in your pipeline. This activity could be used to iterate over a collection of items and execute specified activities in a loop.

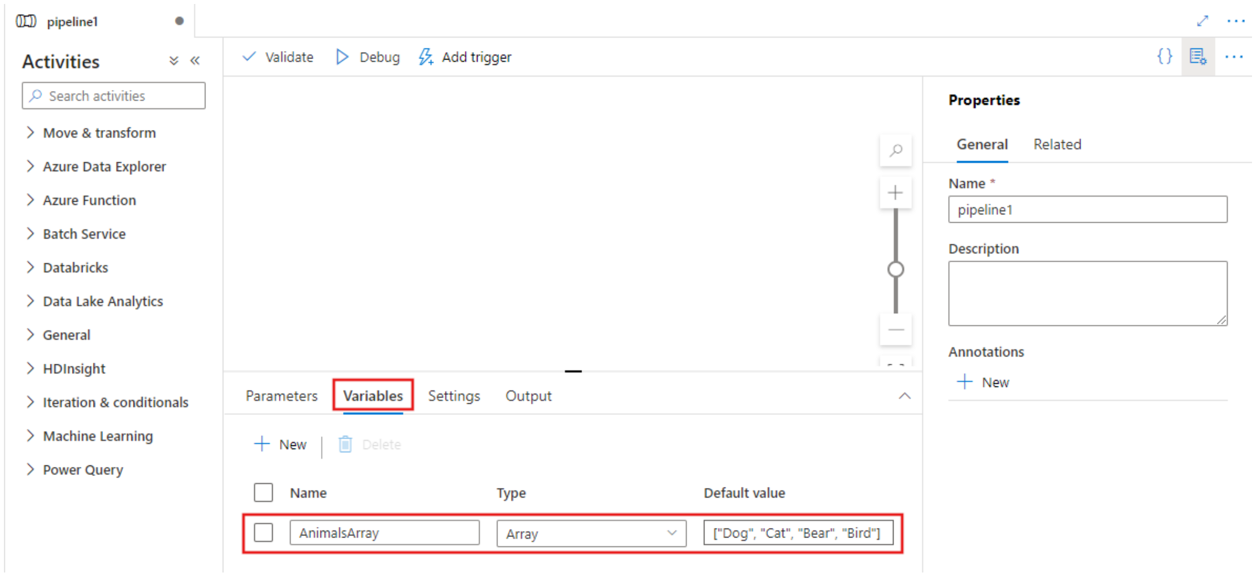
ForEach activity's item collection can include outputs of other activities, pipeline parameters or variables of array type. This activity is a compound activity- in other words, it can include more than one activity.

Foreach activity to ADF would be kind of similar to what for loop or while loop in various programming languages like C, C++, Python, Java, Scala and many others.

Create a ForEach activity with UI

To use a ForEach activity in a pipeline, complete the following steps:

1. You can use any array type variable or outputs from other activities as the input for your ForEach activity. To create an array variable, select the background of the pipeline canvas and then select the **Variables** tab to add an array type variable as shown below.

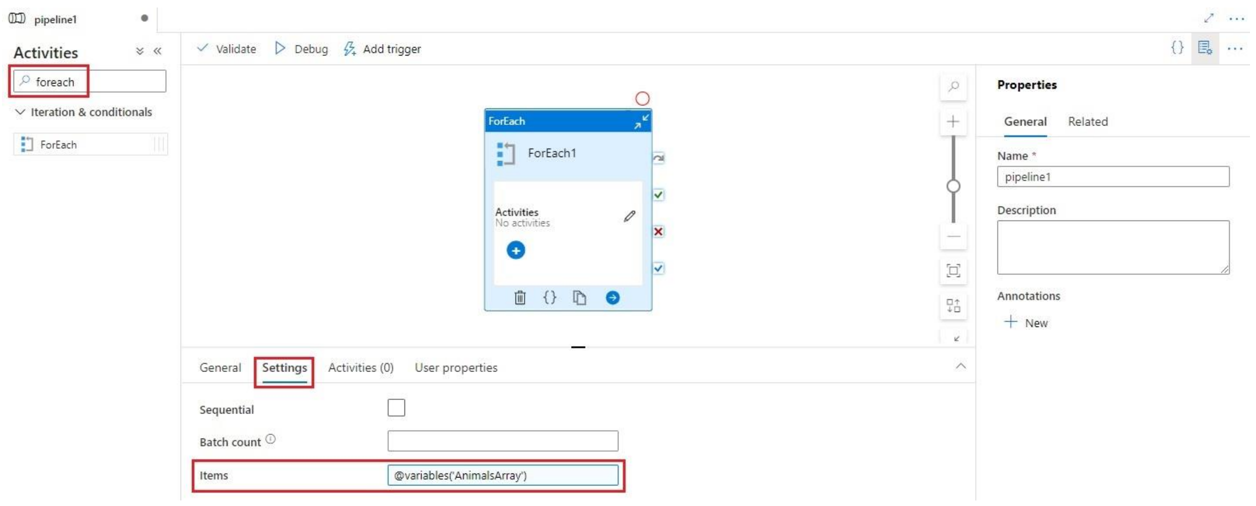


Name: pipeline1

Name	Type	Default value
AnimalsArray	Array	["Dog", "Cat", "Bear", "Bird"]

2. Search for *ForEach* in the pipeline Activities pane, and drag a ForEach activity to the pipeline canvas.

3. Select the new ForEach activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



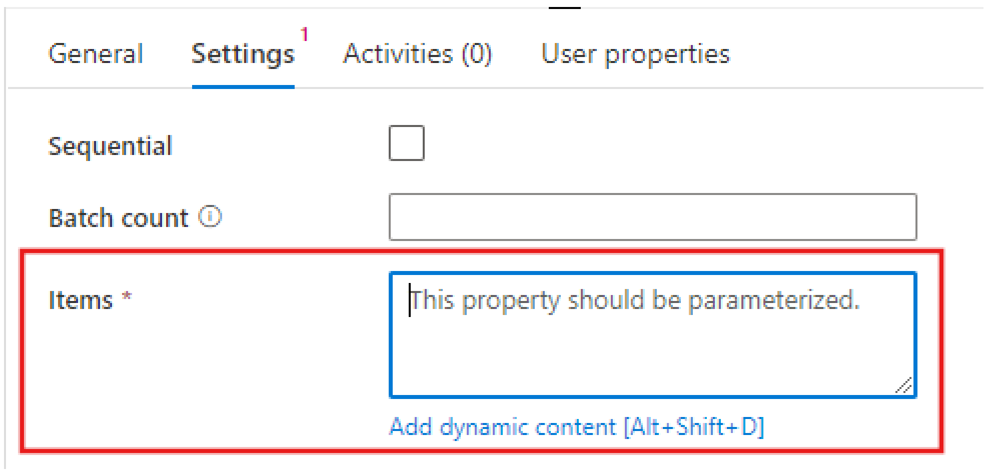
General | Settings | Activities (0) | User properties

Sequential

Batch count

Items @variables('AnimalsArray')

4. Select the **Items** field and then select the **Add dynamic content** link to open the dynamic content editor pane.



General | Settings | Activities (0) | User properties

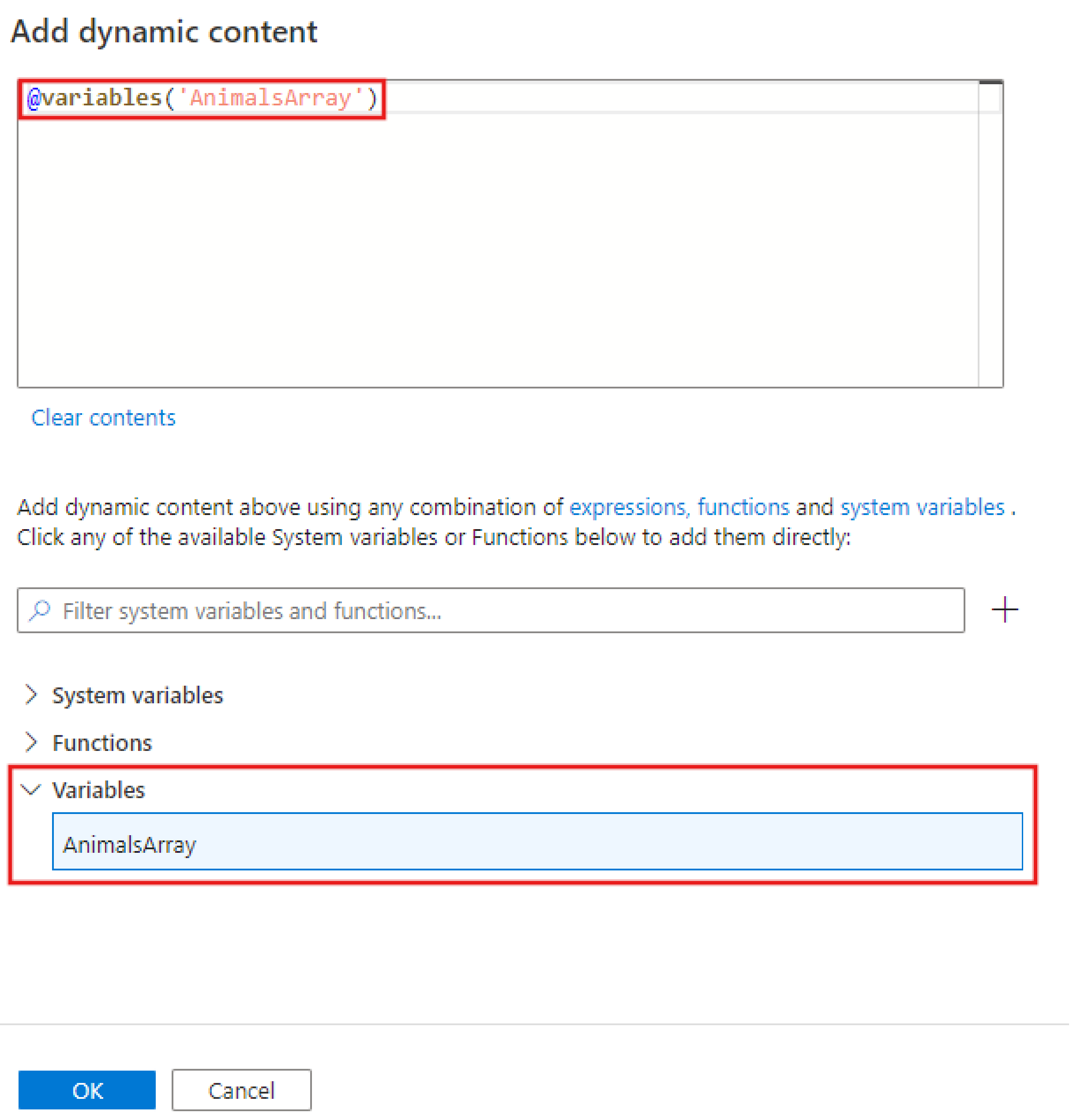
Sequential

Batch count

Items * This property should be parameterized.

Add dynamic content [Alt+Shift+D]

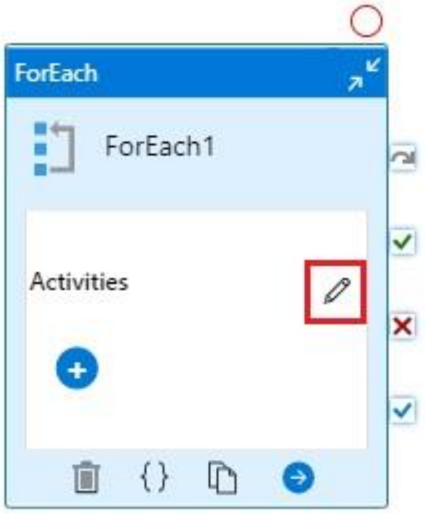
5. Select your input array to be filtered in the dynamic content editor. In this example, we select the variable created in the first step.



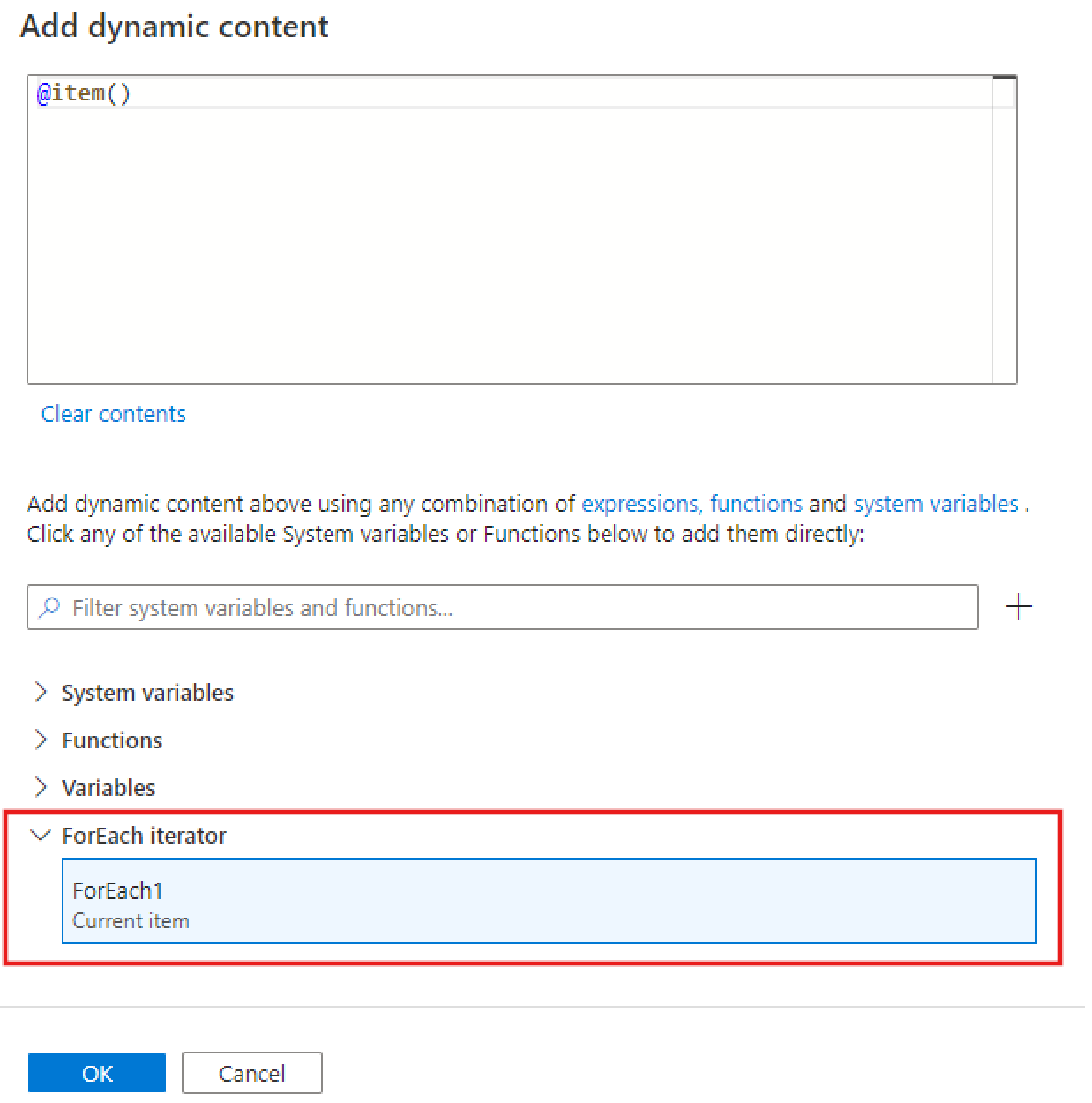
```
@variables('AnimalsArray')
```

Clear contents
Add dynamic content above using any combination of expressions, functions and system variables .
Click any of the available System variables or Functions below to add them directly:
Filter system variables and functions...
System variables
Functions
Variables
AnimalsArray
OK
Cancel

6. Select the Activities editor on the ForEach activity to add one or more activities to be executed for each item in the input **Items** array.



7. In any activities you create within the ForEach activity, you can reference the current item the ForEach activity is iterating through from the **Items** list. You can reference the current item anywhere you can use a dynamic expression to specify a property value. In the dynamic content editor, select the ForEach iterator to return the current item.



Add dynamic content

```
@item()
```

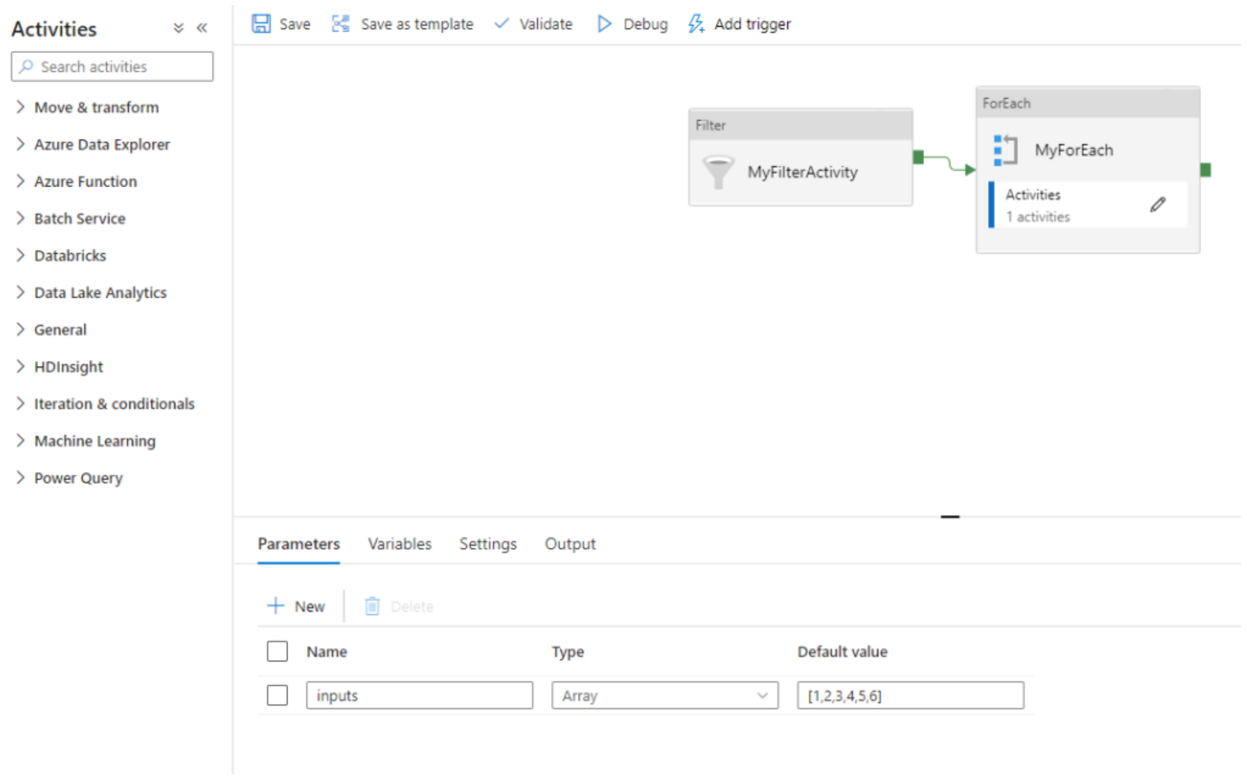
Clear contents

Add dynamic content above using any combination of expressions, functions and system variables .
Click any of the available System variables or Functions below to add them directly:

Azure Data Factory: Filter Activity

The purpose of *Filter Activity* is to process array items based on some condition. Consider a scenario where we would like to set the value of a variable to the current array item that satisfies some business rule or condition. We can use *Filter activity* to design the control flow.

Let’s have a look at the example below:

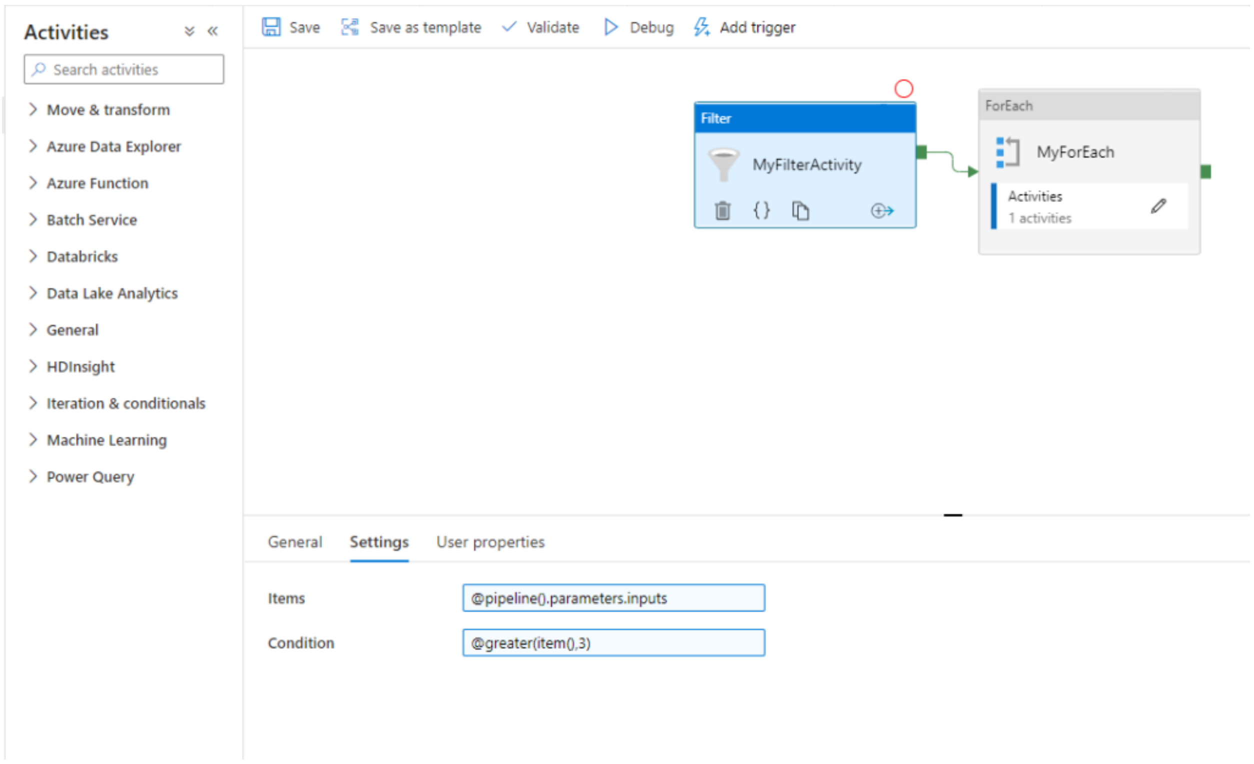


Name	Type	Default value
inputs	Array	[1,2,3,4,5,6]

Azure Data Factory: Filter Activity Example

As we can see in the screenshot above, the *pipeline* contains two activities (*Filter* and *ForEach*) with an input array which is being passed on as the *inputs pipeline parameter*.

Based on *condition* under the *settings* tab, the *Filter Activity* only selects the array items which are greater than 3:

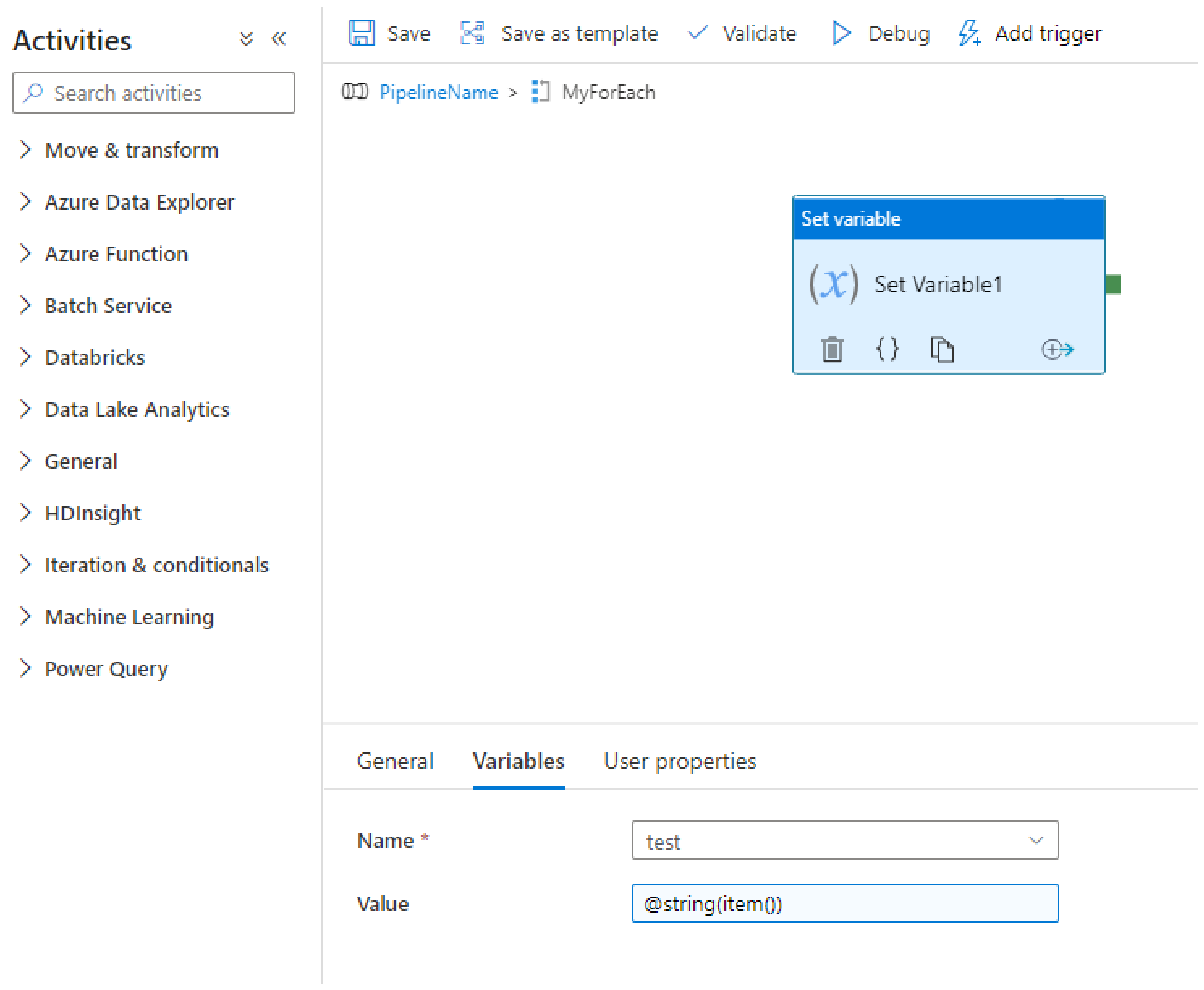


Items: @pipeline().parameters.inputs

Condition: @greater(item(),3)

Azure Data Factory: Filter Activity Settings

For every array item greater than 3, the *ForEach* activity runs the *Set Variable* activity which converts the current filtered item value to *string* datatype and assigns it to the variable *test*.

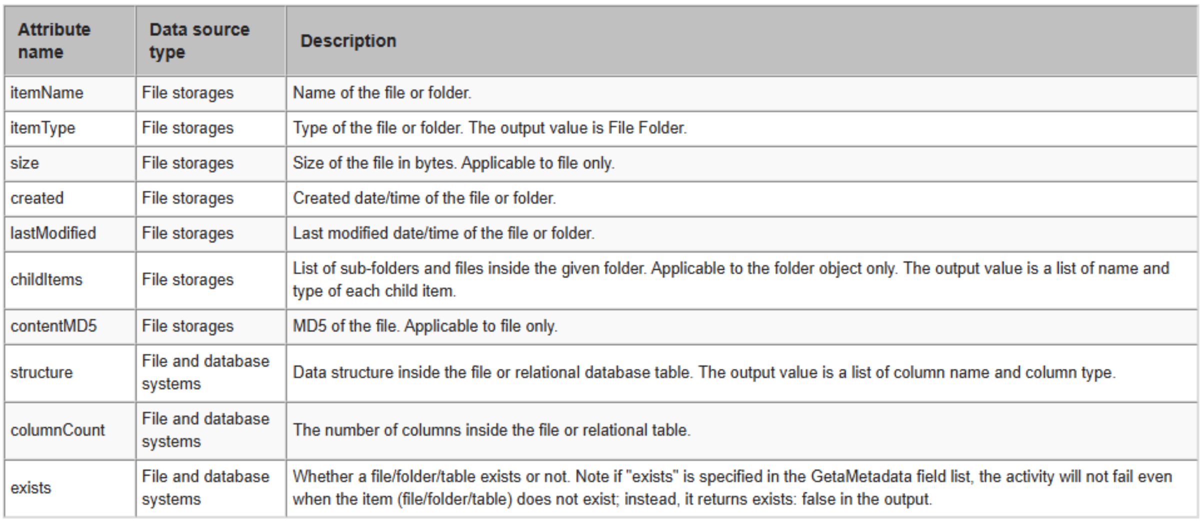


Name *	test
Value	@string(item())

GET METADATA ACTIVITY IN ADF

The **Get Metadata** activity allows reading metadata information of its sources.

The list of attributes returned by this activity is dependent on its source type, some attributes are available only for file-based sources, others available for database tables and there are few attributes applicable for both types. Here is the full list of attributes, borrowed from Microsoft's site:



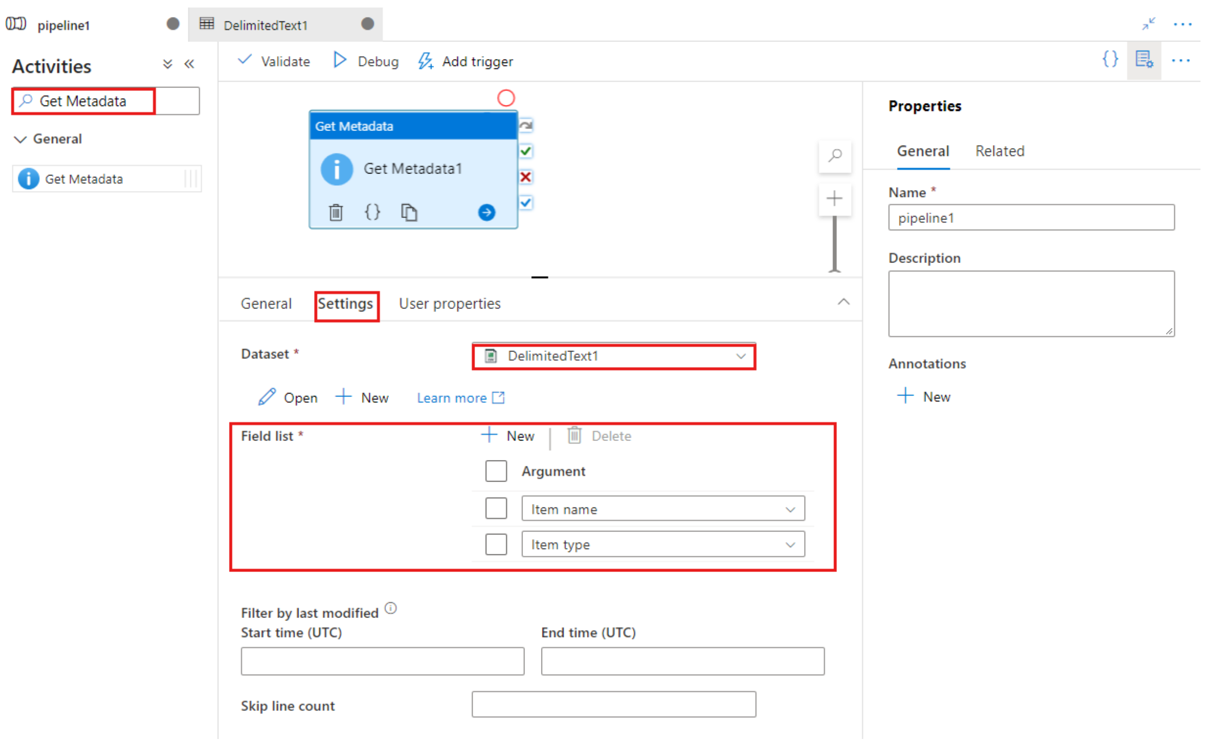
Attribute name	Data source type	Description
itemName	File storages	Name of the file or folder.
itemType	File storages	Type of the file or folder. The output value is File Folder.
size	File storages	Size of the file in bytes. Applicable to file only.
created	File storages	Created date/time of the file or folder.
lastModified	File storages	Last modified date/time of the file or folder.
childItems	File storages	List of sub-folders and files inside the given folder. Applicable to the folder object only. The output value is a list of name and type of each child item.
contentMD5	File storages	MD5 of the file. Applicable to file only.
structure	File and database systems	Data structure inside the file or relational database table. The output value is a list of column name and column type.
columnCount	File and database systems	The number of columns inside the file or relational table.
exists	File and database systems	Whether a file/folder/table exists or not. Note if "exists" is specified in the GetaMetadata field list, the activity will not fail even when the item (file/folder/table) does not exist; instead, it returns exists: false in the output.

Please note that the **childItems** attribute from this list is applicable to folders only and is designed to provide list of files and folders nested within the source folder.

The data obtained by **Get Metadata** activity can be used by subsequent iterative activities, to perform copy or transformation activities on a dynamic basis.

Use a Get Metadata activity in a pipeline, complete the following steps:

1. Search for *Get Metadata* in the pipeline Activities pane, and drag a Fail activity to the pipeline canvas.
2. Select the new Get Metadata activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.
3. Choose a dataset, or create a new one with the New button. Then you can specify filter options and add columns from the available metadata for the dataset.



Dataset * DelimitedText1

Field list *

☐	Argument
☐	Item name
☐	Item type

Filter by last modified

Start time (UTC)

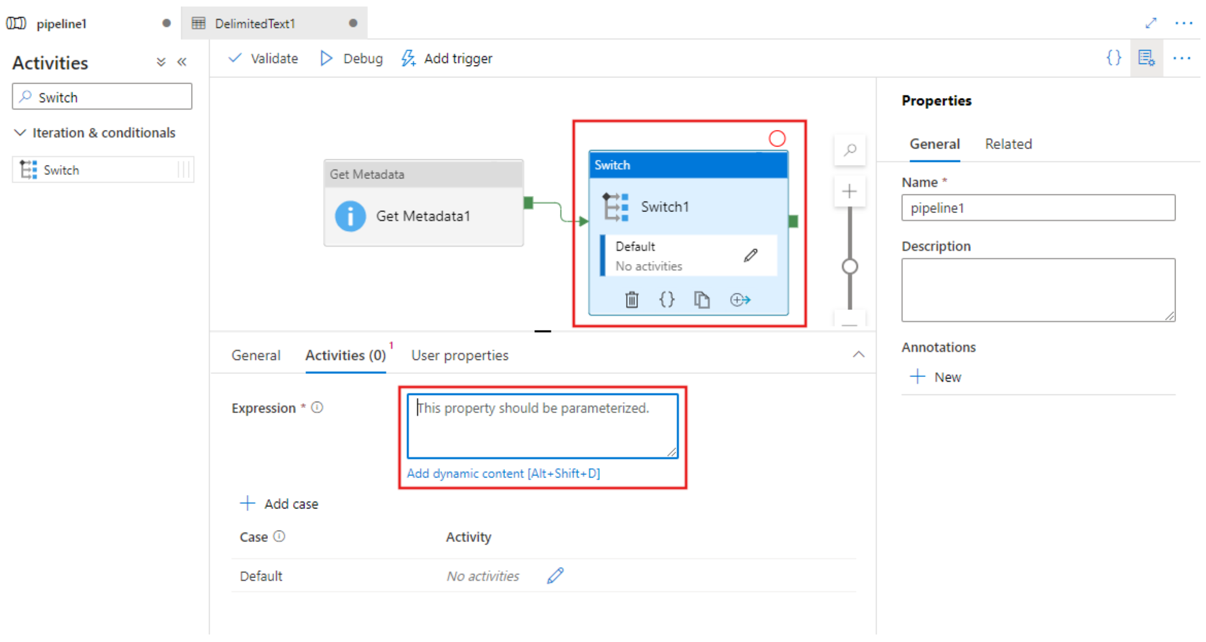
End time (UTC)

Skip line count

Name * pipeline1

Description

4. Use the output of the activity as an input to another activity, like a Switch activity in this example. You can reference the output of the Metadata Activity anywhere dynamic content is supported in the other activity.



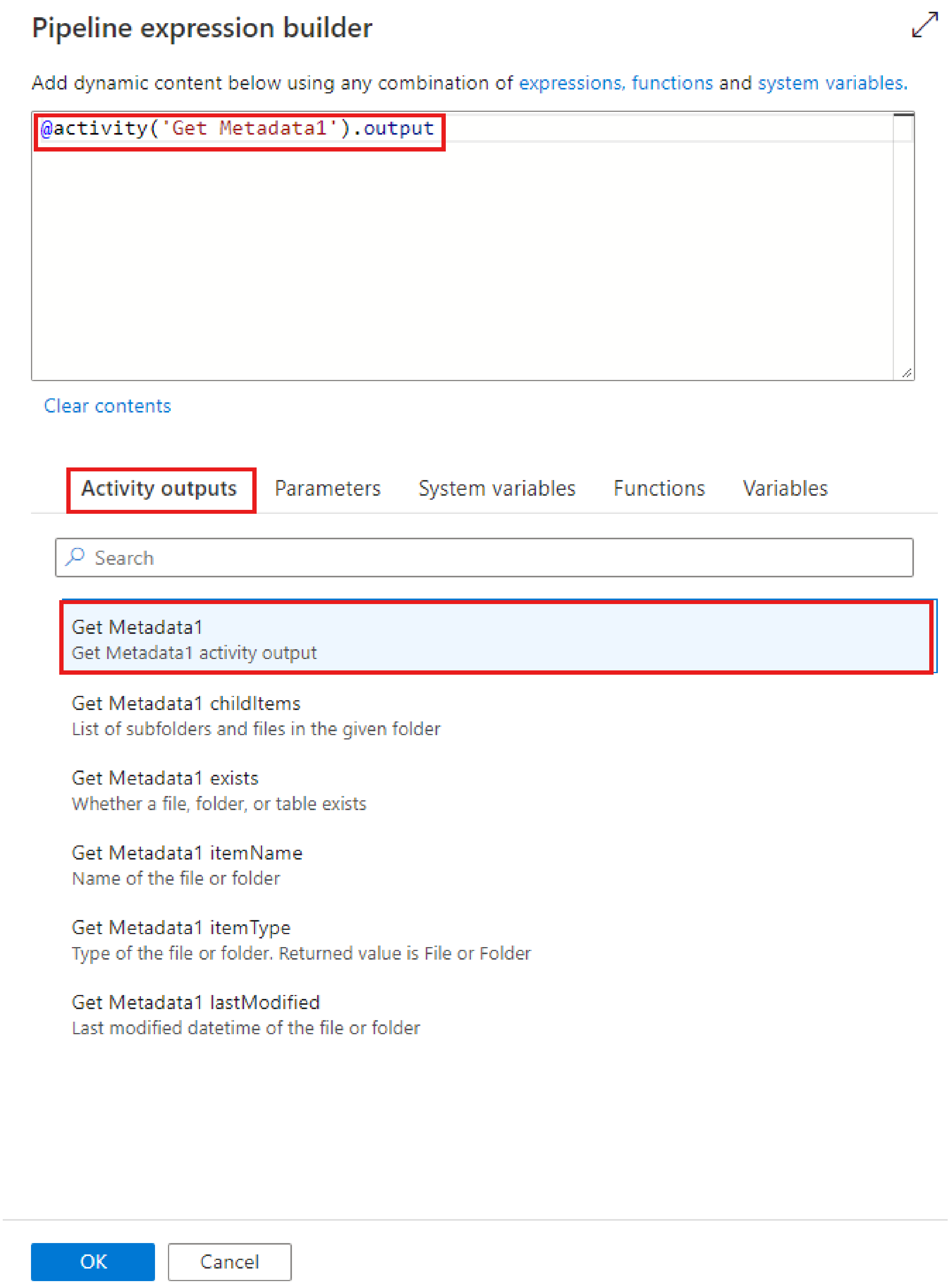
Expression * This property should be parameterized.

Case	Activity
Default	No activities

Name * pipeline1

Description

5. In the dynamic content editor, select the Get Metadata activity output to reference it in the other activity.


Pipeline expression builder

Add dynamic content below using any combination of expressions, functions and system variables.

```
@activity('Get Metadata1').output
```

Clear contents

Get Metadata1
Get Metadata1 activity output

Get Metadata1 childItems
List of subfolders and files in the given folder

Get Metadata1 exists
Whether a file, folder, or table exists

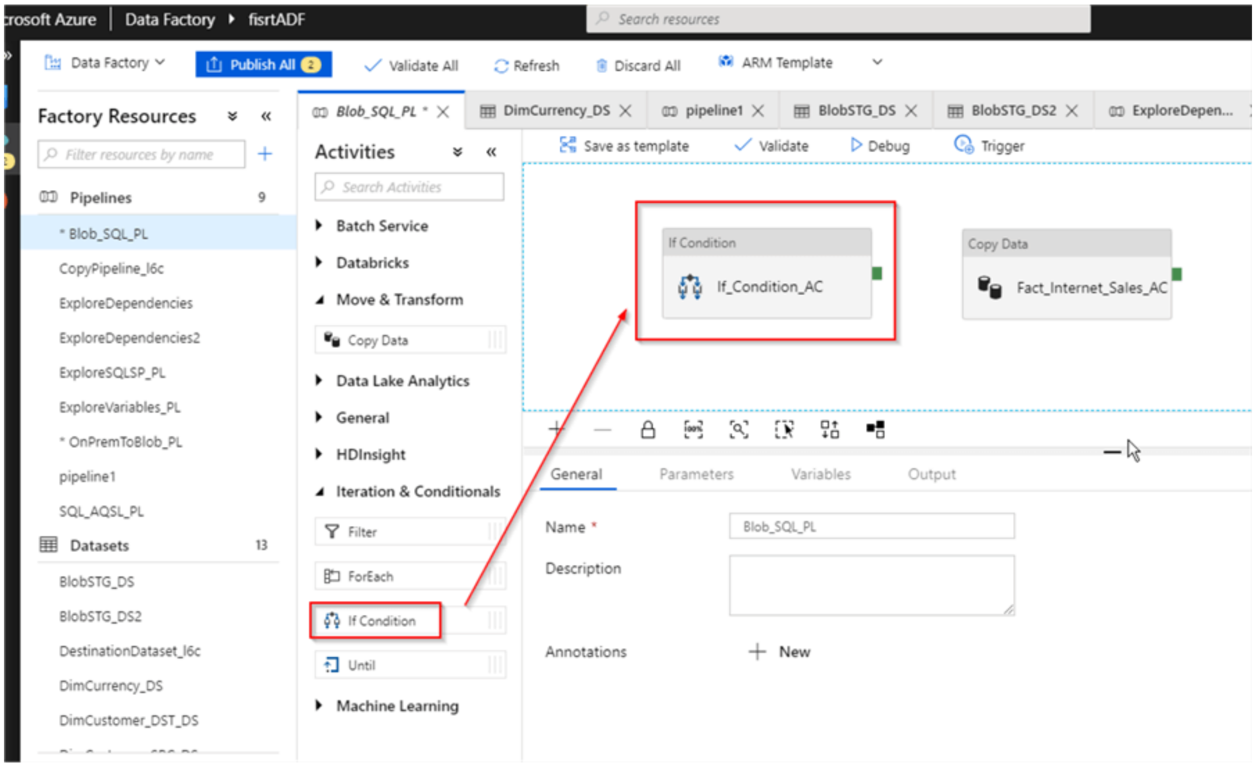
Get Metadata1 itemName
Name of the file or folder

Get Metadata1 itemType
Type of the file or folder. Returned value is File or Folder

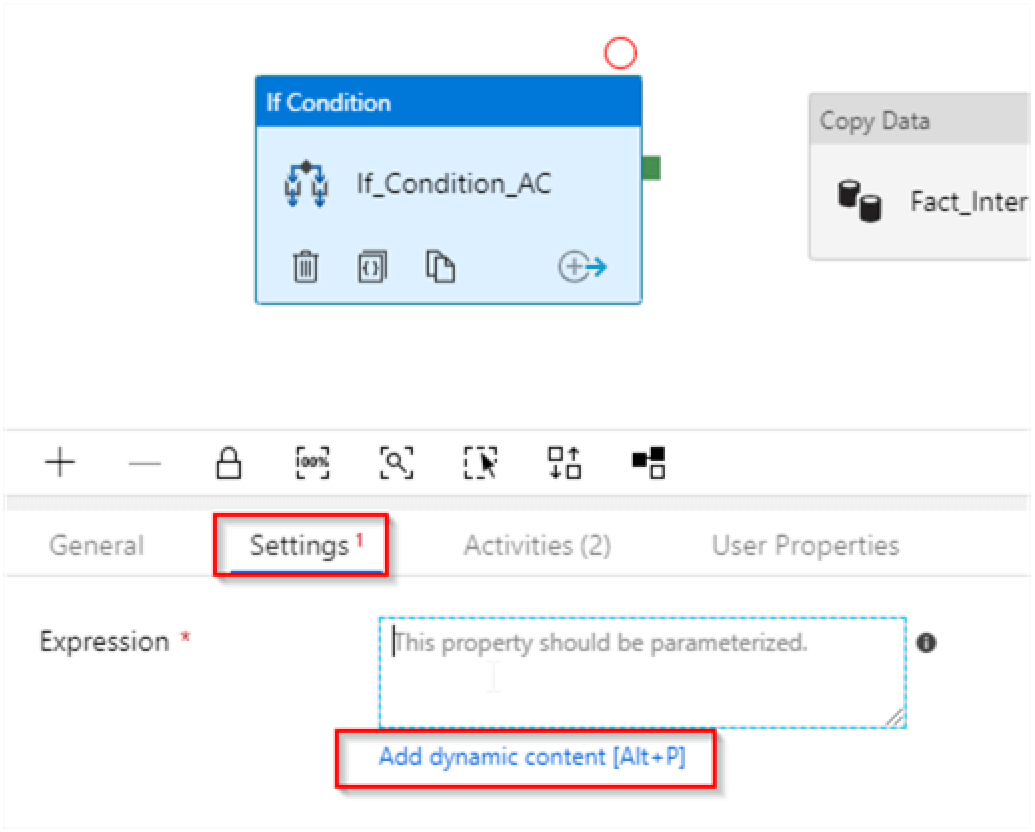
Get Metadata1 lastModified
Last modified datetime of the file or folder

Creating Azure Data Factory If Condition Activity

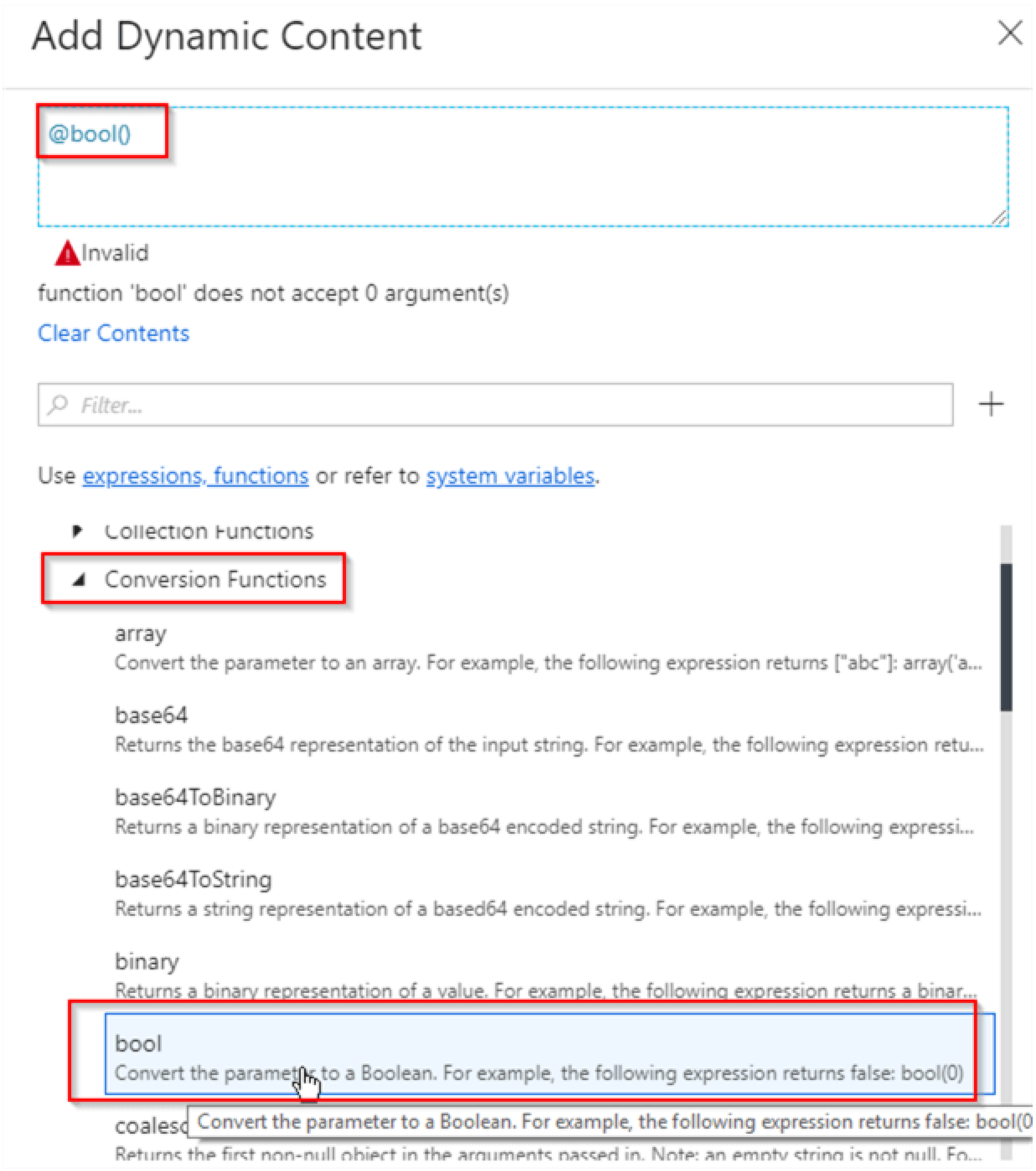
Select pipeline Blob_SQL_PL, expand 'Iterations and Conditionals' group on Activities panel, drag-drop an If Condition activity into the central panel and assign the name (I've named it If_Condition_AC):



Switch to the Settings tab, place the cursor in the Expression text box and click the 'Add dynamic content' link under that text box, to start building an evaluation expression:



Expand Functions/Conversion Functions group and select the bool function:


Add Dynamic Content

@bool()

Invalid

function 'bool' does not accept 0 argument(s)

Clear Contents

Filter...

Use expressions, functions or refer to system variables.

Collection Functions

Conversion Functions

array
Convert the parameter to an array. For example, the following expression returns ["abc"]: array('a...

base64
Returns the base64 representation of the input string. For example, the following expression retu...

base64ToBinary
Returns a binary representation of a base64 encoded string. For example, the following expressi...

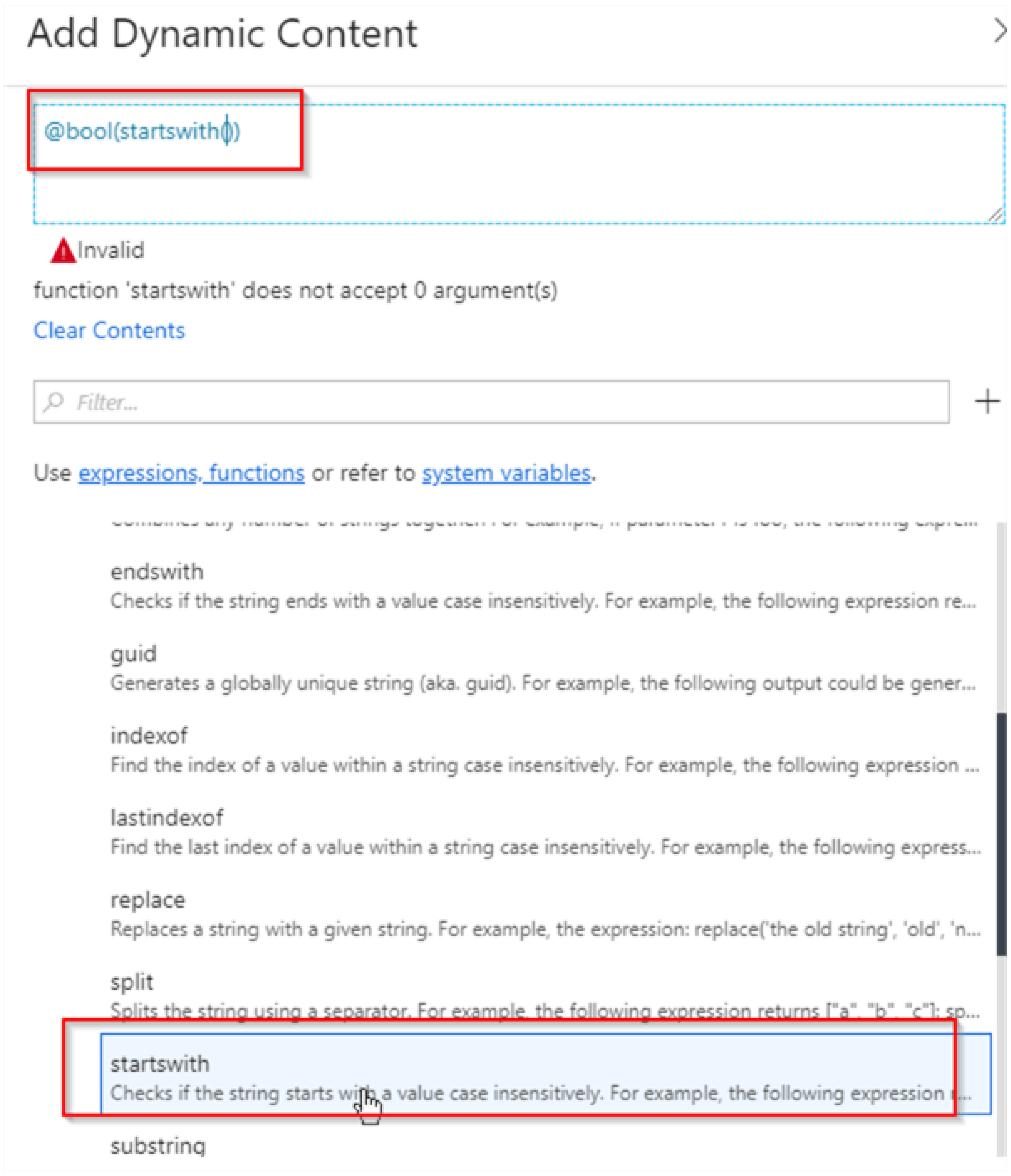
base64ToString
Returns a string representation of a based64 encoded string. For example, the following expressi...

binary
Returns a binary representation of a value. For example, the following expression returns a binar...

bool
Convert the parameter to a Boolean. For example, the following expression returns false: bool(0)

coalesc Convert the parameter to a Boolean. For example, the following expression returns false: bool(0
Returns the first non-null object in the arguments passed in. Note: an empty string is not null. Fo...

Place the cursor inside the bool function brackets, expand Functions/String Functions group and select the startswith function:


Add Dynamic Content

@bool(startswith())

Invalid

function 'startswith' does not accept 0 argument(s)

Clear Contents

Use expressions, functions or refer to system variables.

endswith
Checks if the string ends with a value case insensitively. For example, the following expression re...

guid
Generates a globally unique string (aka. guid). For example, the following output could be gener...

indexof
Find the index of a value within a string case insensitively. For example, the following expression ...

lastindexof
Find the last index of a value within a string case insensitively. For example, the following express...

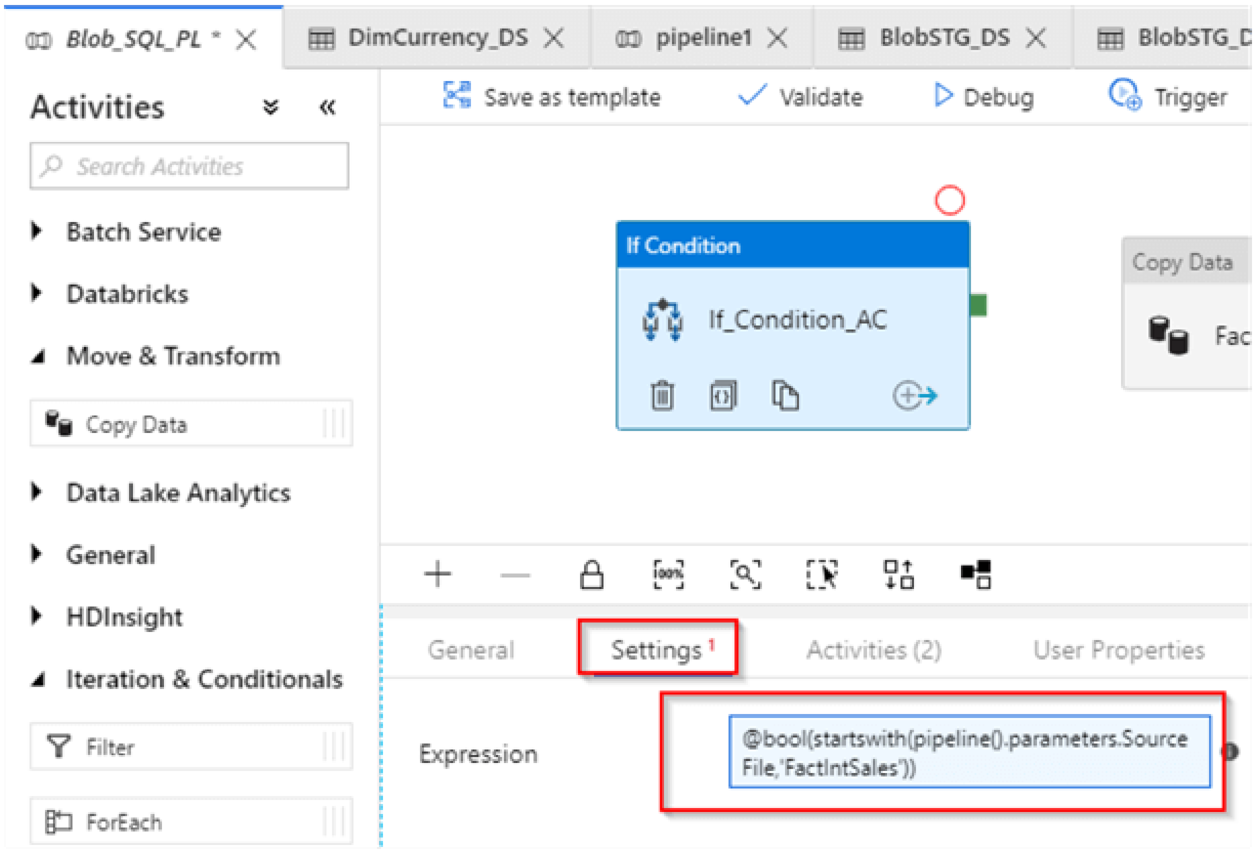
replace
Replaces a string with a given string. For example, the expression: replace('the old string', 'old', 'n...

split
Splits the string using a separator. For example, the following expression returns ["a", "b", "c"]: sp...

startswith
Checks if the string starts with a value case insensitively. For example, the following expression ...

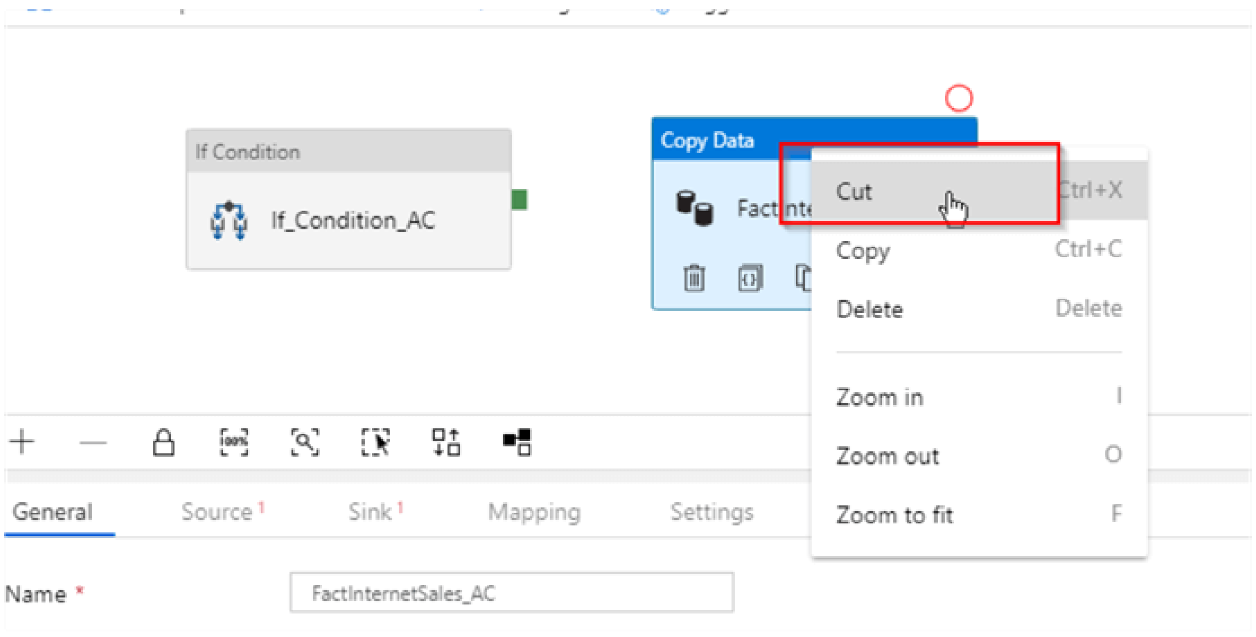
substring

Place the cursor inside the startswith function brackets and select SourceFile pipeline parameter we created earlier, followed by a comma and 'FactIntSales' string and then confirm to close the Add Dynamic Content window. Here's the final expression- @bool(startswith(pipeline().parameters.SourceFile,'FactIntSales')) , which evaluates whether or not the input file's name starts with 'FactIntSales' string. Here's a screenshot for the activity with the evaluation condition:



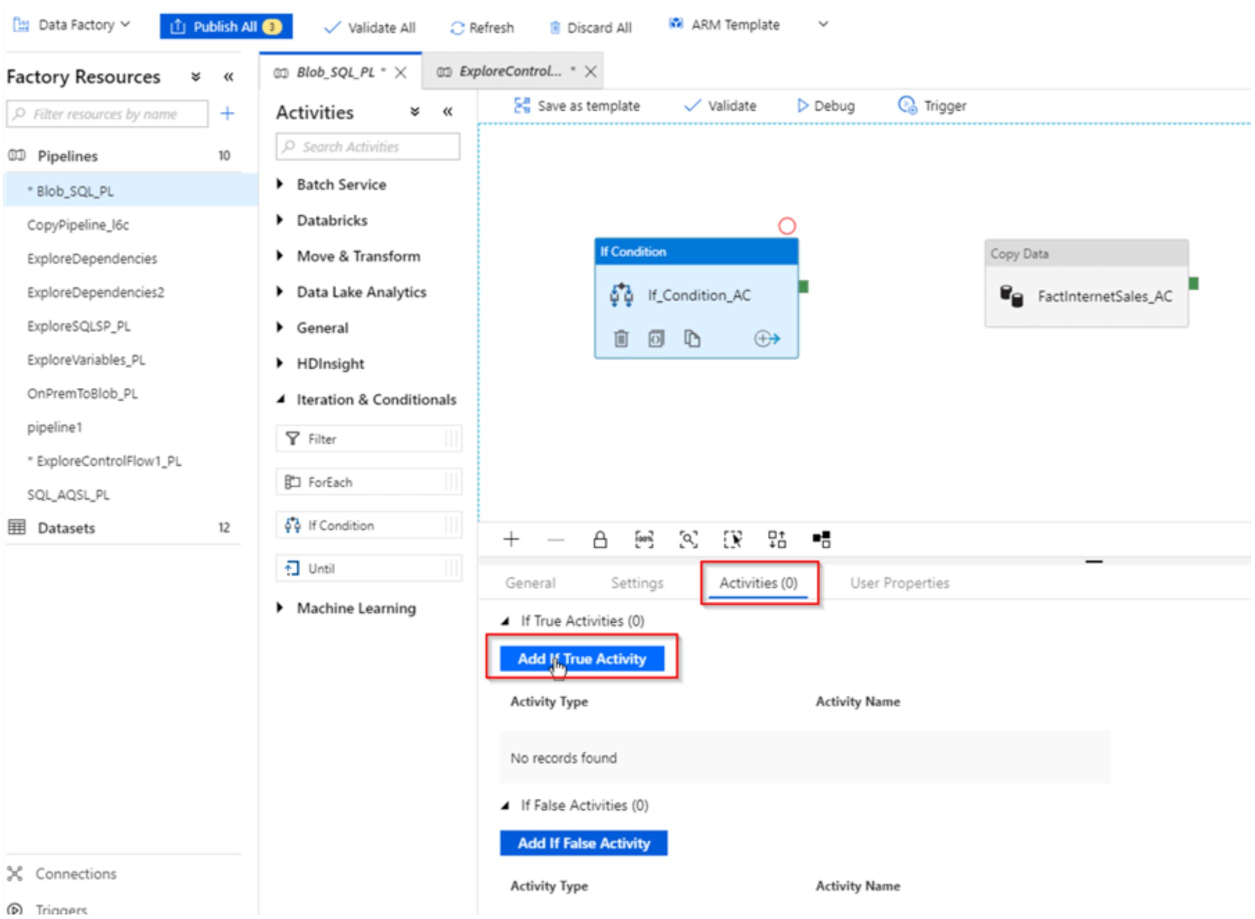
Expression: @bool(startswith(pipeline().parameters.SourceFile,'FactIntSales'))

Next, let's copy FactInternetSales_AC activity into the buffer, using right click and Cut command:

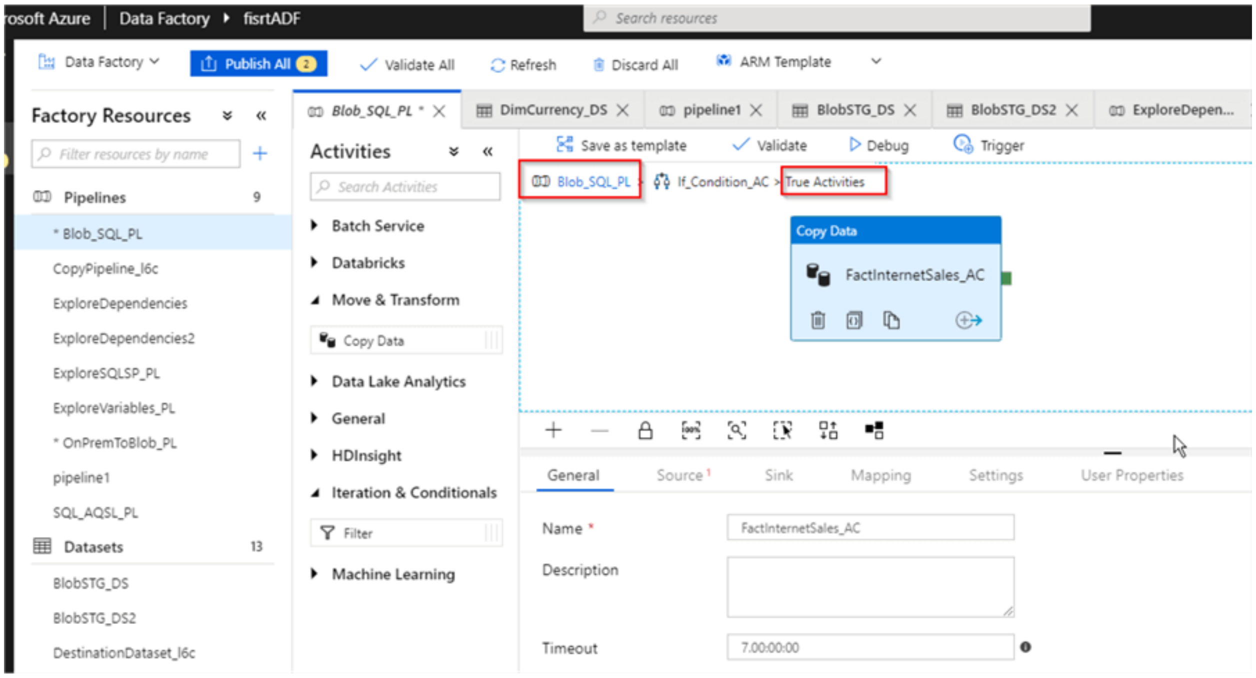


Name * FactInternetSales_AC

Now, we need to add activities to True and False evaluation groups. Select If_Condition_AC activity, switch to the Activities tab and click Add If True Activity button:

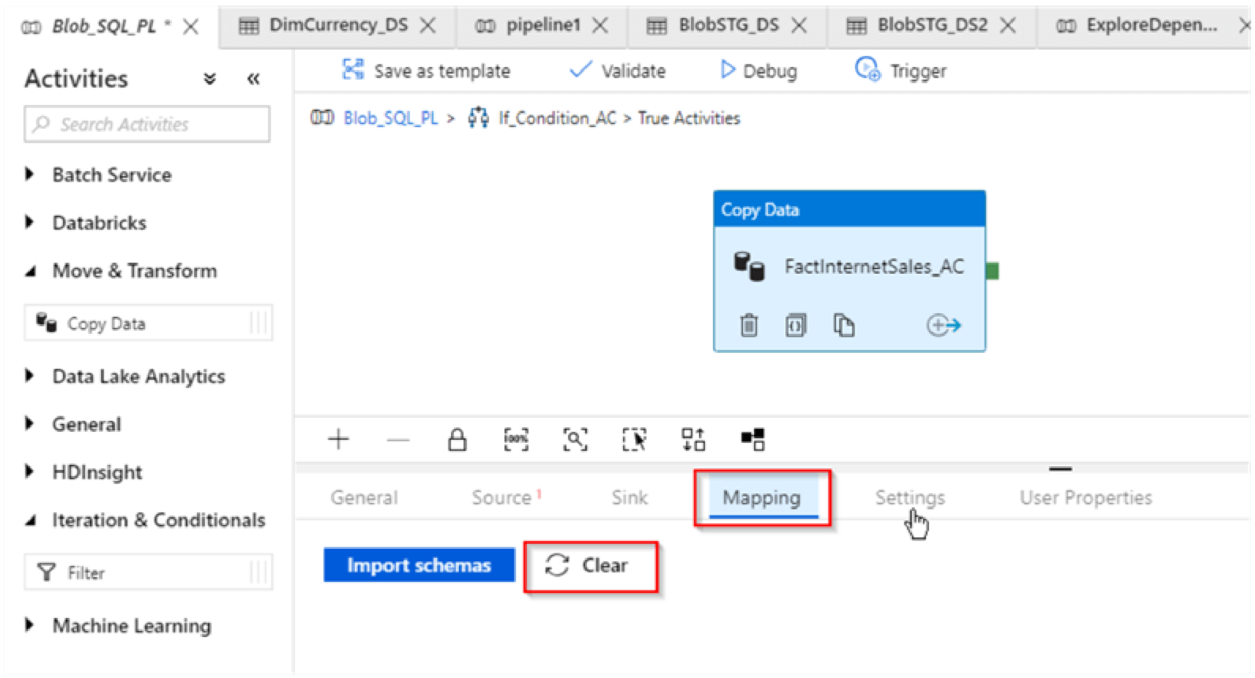


Right click in the design surface and select the Paste command, to paste the activity we copied earlier into the buffer and assign a name (I have named it FactInternetSales_AC):

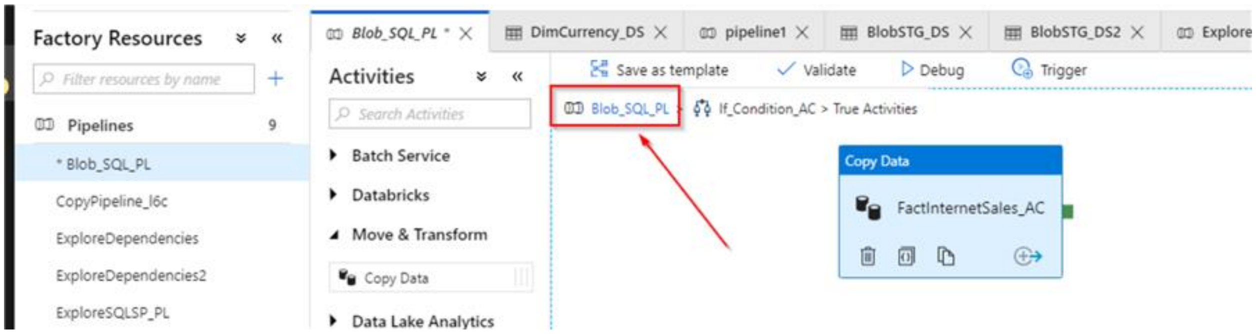


Name *	FactInternetSales_AC
Description	
Timeout	7.00:00:00

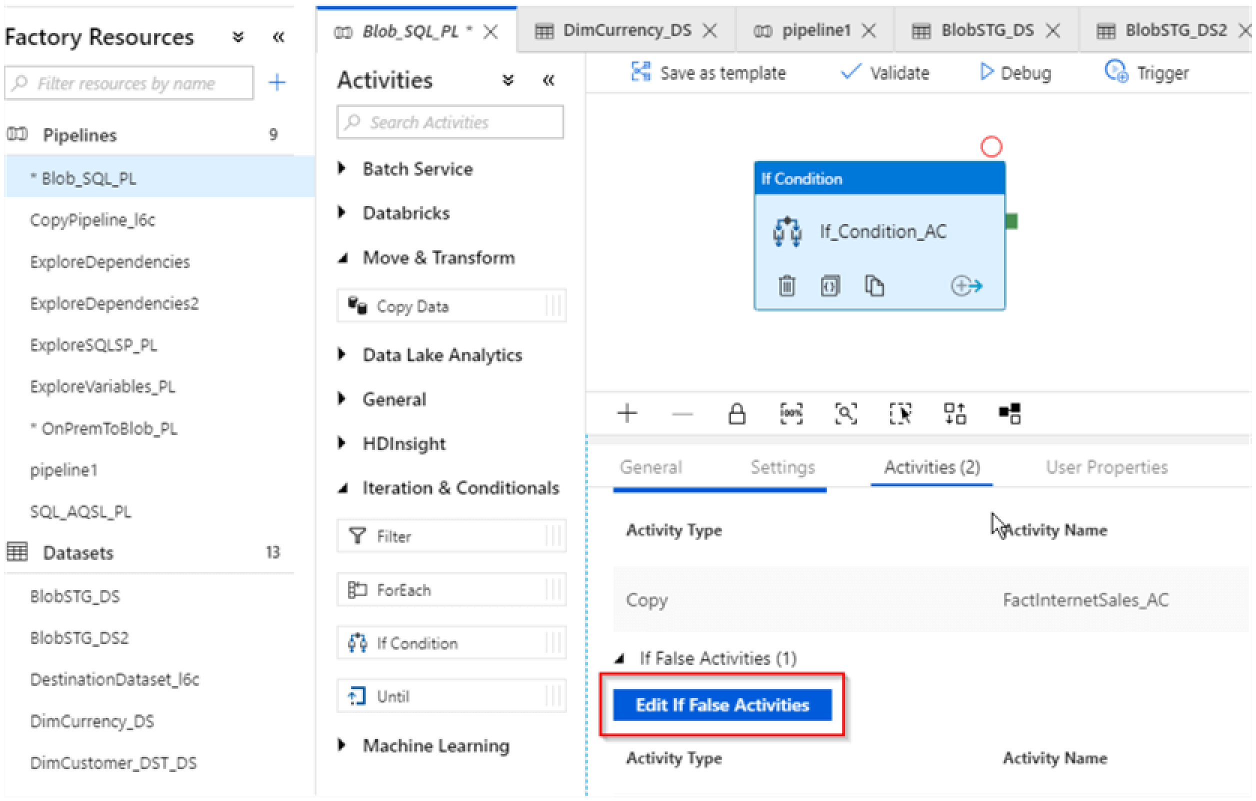
The activity FactInternetSales_AC originally has been created with the explicit field mapping (see Transfer On-Premises Files to Azure SQL Database for more details). However, because this pipeline is going to transfer files with different structures, we no longer need to have explicit mapping, so let's switch to the Mapping tab and click the Clear button, to remove mapping:



Please note the pipeline hierarchy link at the top of design surface, which allows you to navigate to the parent pipeline's design screen. We could add more activities into True Activities group, however that's not required for the purpose of this exercise, so let's click Blob_SQL_PL navigation link to return to the parent pipeline's design screen:

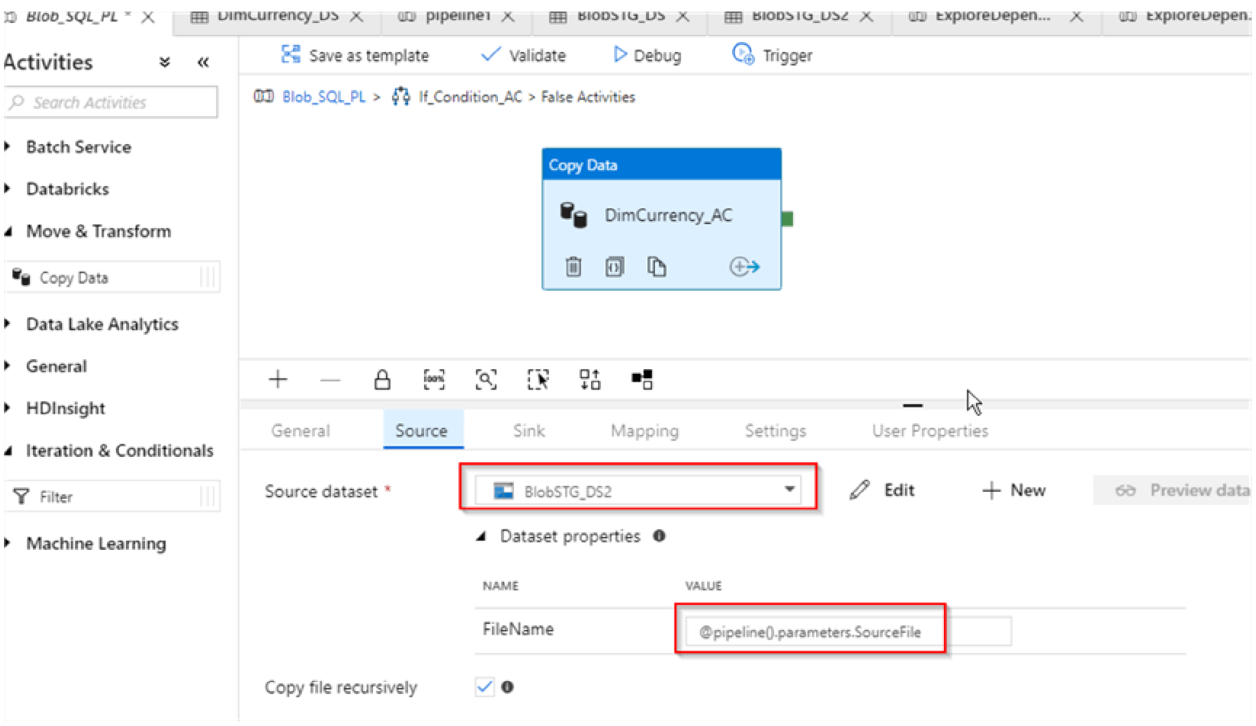


We'll follow similar steps to add activity into False group:



Activity Type	Activity Name
Copy	FactInternetSales_AC

Let's add a Copy activity to copy files from the blob storage container into the DimCurrency table in Azure SQL DB (I've named it DimCurrency_AC). This activity's source dataset screen will be identical to the FactInternetSales_AC activity's source screen:



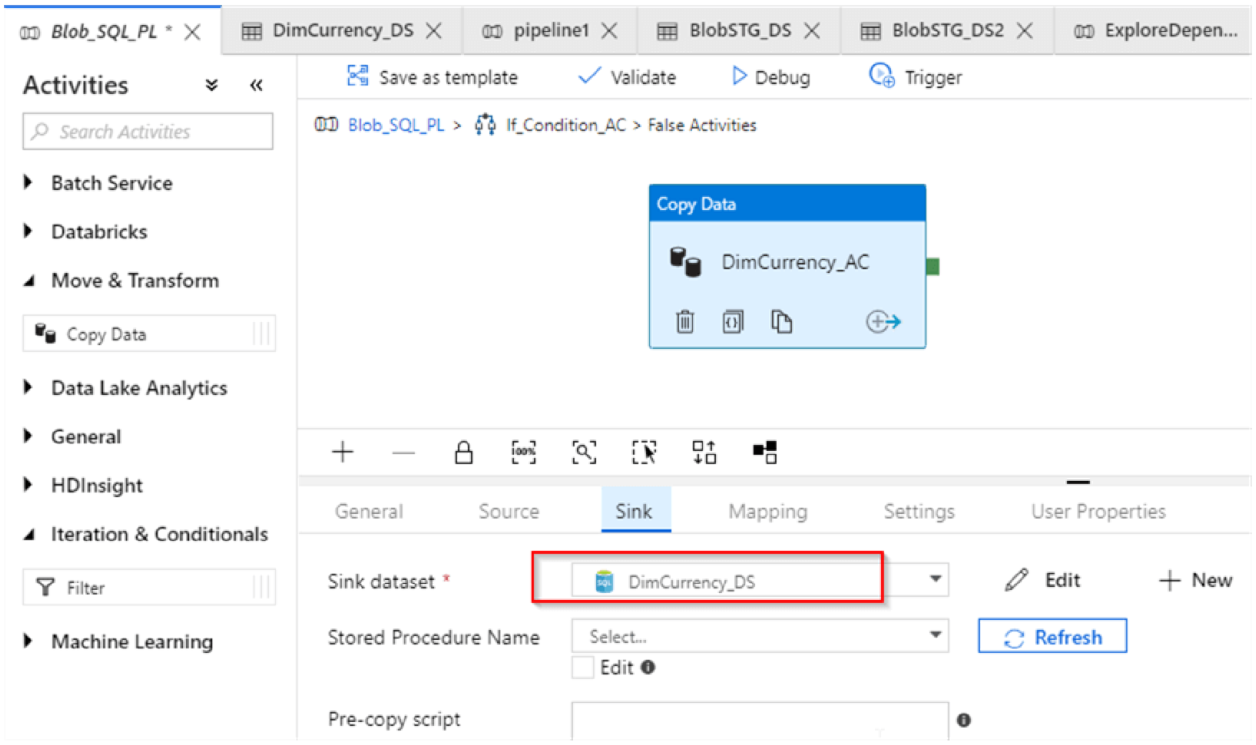
Source dataset * BlobSTG_DS2

Dataset properties

NAME	VALUE
FileName	@pipeline().parameters.SourceFile

Copy file recursively ✓

As for the sink dataset, we will need to create Azure SQL DB dataset, pointing to the DimCurrency table:



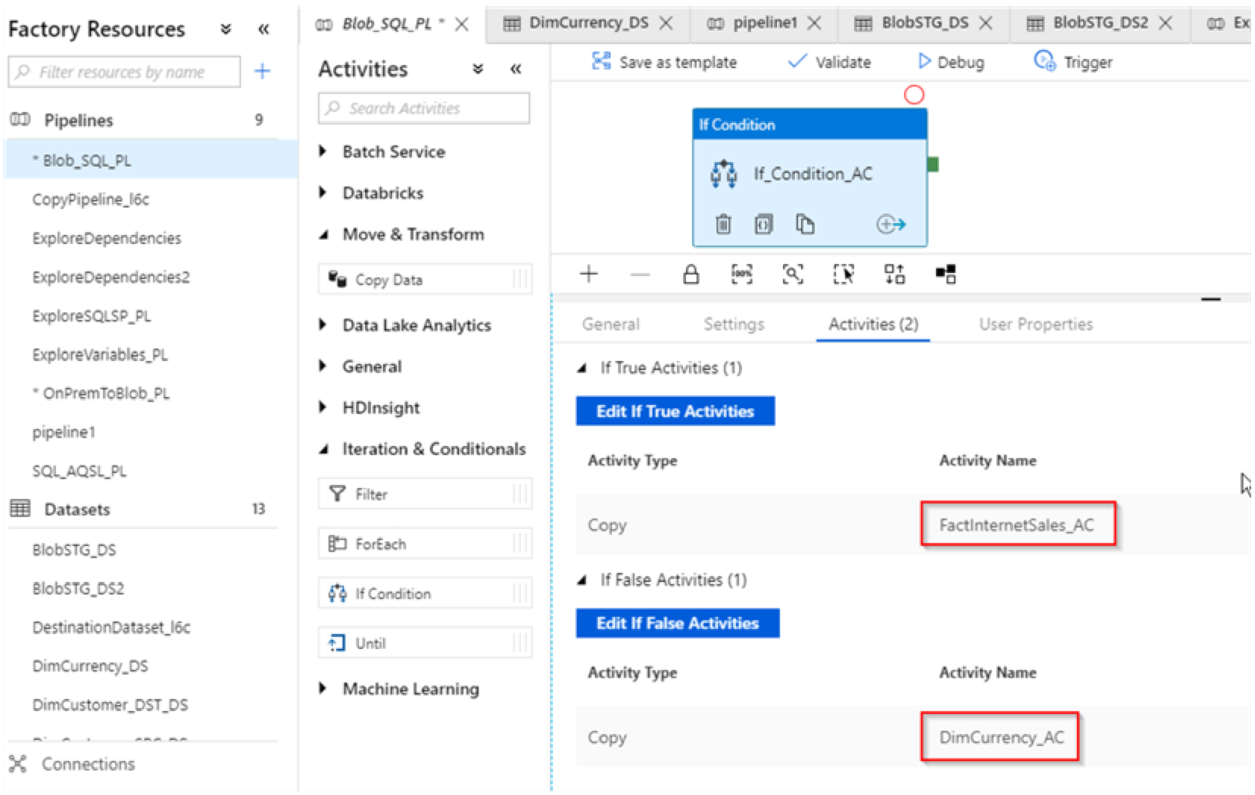
Sink dataset * DimCurrency_DS

Stored Procedure Name Select...

Edit

Pre-copy script

Now that we are done with the configuration of DimCurrency_AC activity, we can return to the parent screen, using the parent navigation link and publish changes. Here is how your final screen should look at this point:



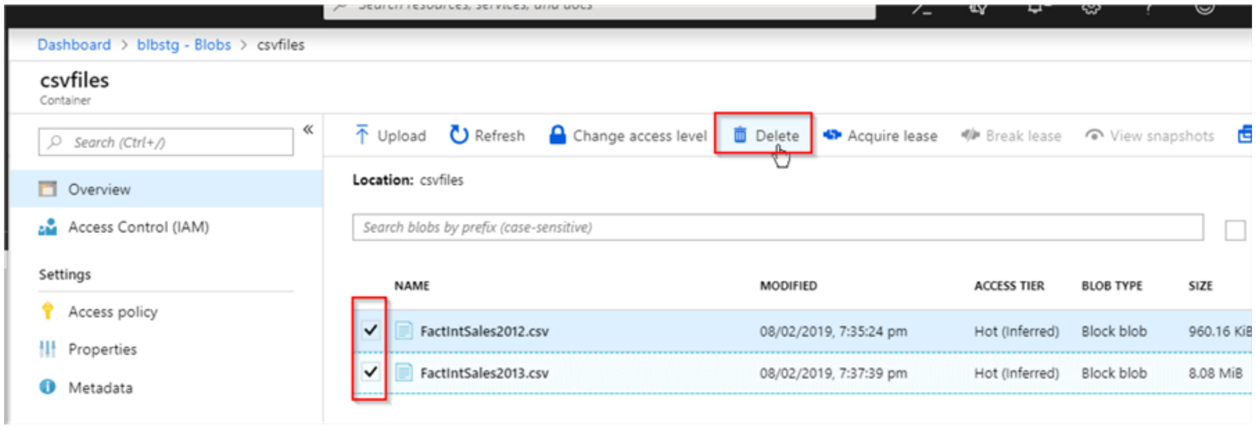
Activity Type	Activity Name
Copy	FactInternetSales_AC

Activity Type	Activity Name
Copy	DimCurrency_AC

For those, who want to see the JSON script for the pipeline we just created, I have attached the script here.

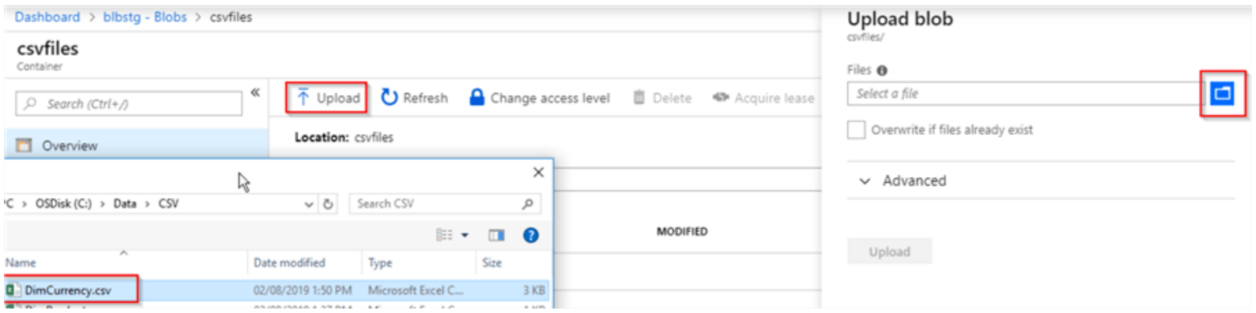
Validating Azure Data Factory Pipeline Execution

Because this pipeline has an event-based trigger associated with it, all we need to initiate it is to drop files into the source container. We can use Azure Portal to manage files in the blob storage, so let's open the Blob Storage screen and remove existing files from the csvfiles container:



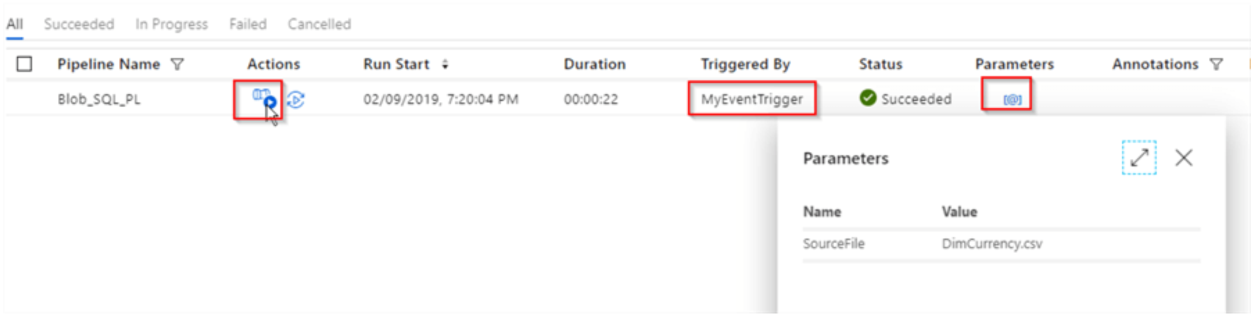
	NAME	MODIFIED	ACCESS TIER	BLOB TYPE	SIZE
✓	FactIntSales2012.csv	08/02/2019, 7:35:24 pm	Hot (Inferred)	Block blob	960.16 KiB
✓	FactIntSales2013.csv	08/02/2019, 7:37:39 pm	Hot (Inferred)	Block blob	8.08 MiB

Now, use the Upload button to select DimCurrency.csv file from the local folder:



Name	Date modified	Type	Size
DimCurrency.csv	02/08/2019 1:50 PM	Microsoft Excel C...	3 KB

Let's wait few minutes for this pipeline to finish and switch to the Monitor screen, to examine the execution results. As expected, MyEventTrigger has started the pipeline in response to DimCurrency.csv file's upload event:

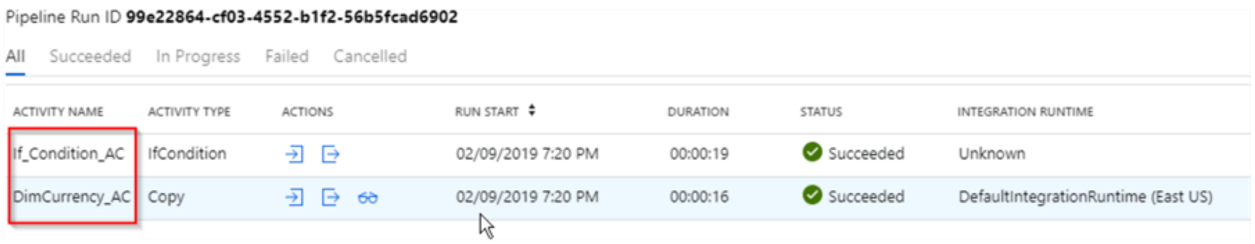


Pipeline Name	Actions	Run Start	Duration	Triggered By	Status	Parameters	Annotations
Blob_SQL_PL		02/09/2019, 7:20:04 PM	00:00:22	MyEventTrigger	Succeeded		

Parameters

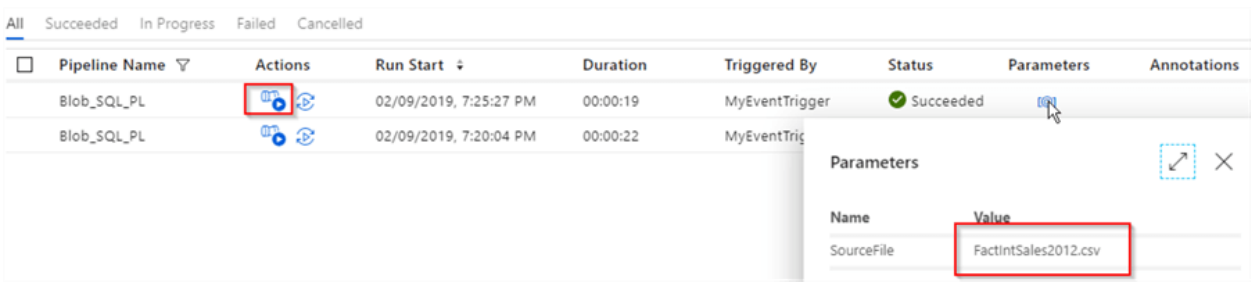
Name	Value
SourceFile	DimCurrency.csv

Upon further examination of execution details, we can see that DimCurrency_AC activity ran after conditional validation:


Pipeline Run ID **99e22864-cf03-4552-b1f2-56b5fcad6902**

ACTIVITY NAME	ACTIVITY TYPE	ACTIONS	RUN START	DURATION	STATUS	INTEGRATION RUNTIME
If_Condition_AC	IfCondition		02/09/2019 7:20 PM	00:00:19	Succeeded	Unknown
DimCurrency_AC	Copy		02/09/2019 7:20 PM	00:00:16	Succeeded	DefaultIntegrationRuntime (East US)

Now, let's upload FactIntSales2012.csv file and see the execution results:

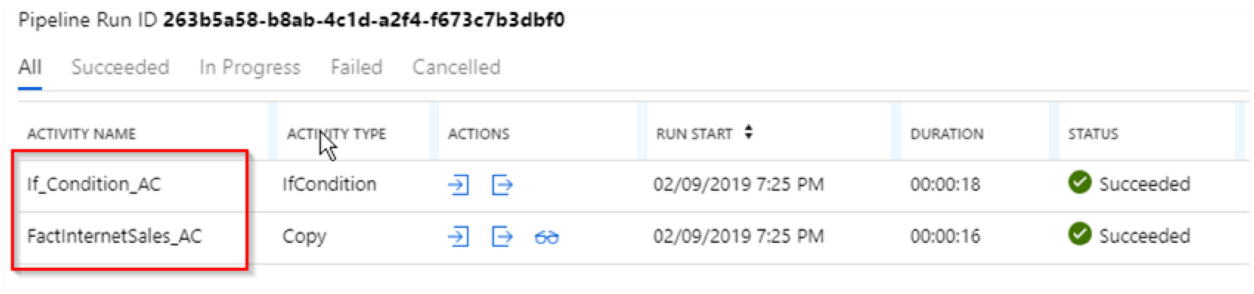


Pipeline Name	Actions	Run Start	Duration	Triggered By	Status	Parameters	Annotations
Blob_SQL_PL		02/09/2019, 7:25:27 PM	00:00:19	MyEventTrigger	Succeeded		
Blob_SQL_PL		02/09/2019, 7:20:04 PM	00:00:22	MyEventTrig			

Parameters

Name	Value
SourceFile	FactIntSales2012.csv

Activity Runs screen confirms that conditional activity worked as expected:



Pipeline Run ID **263b5a58-b8ab-4c1d-a2f4-f673c7b3dbf0**

All Succeeded In Progress Failed Cancelled

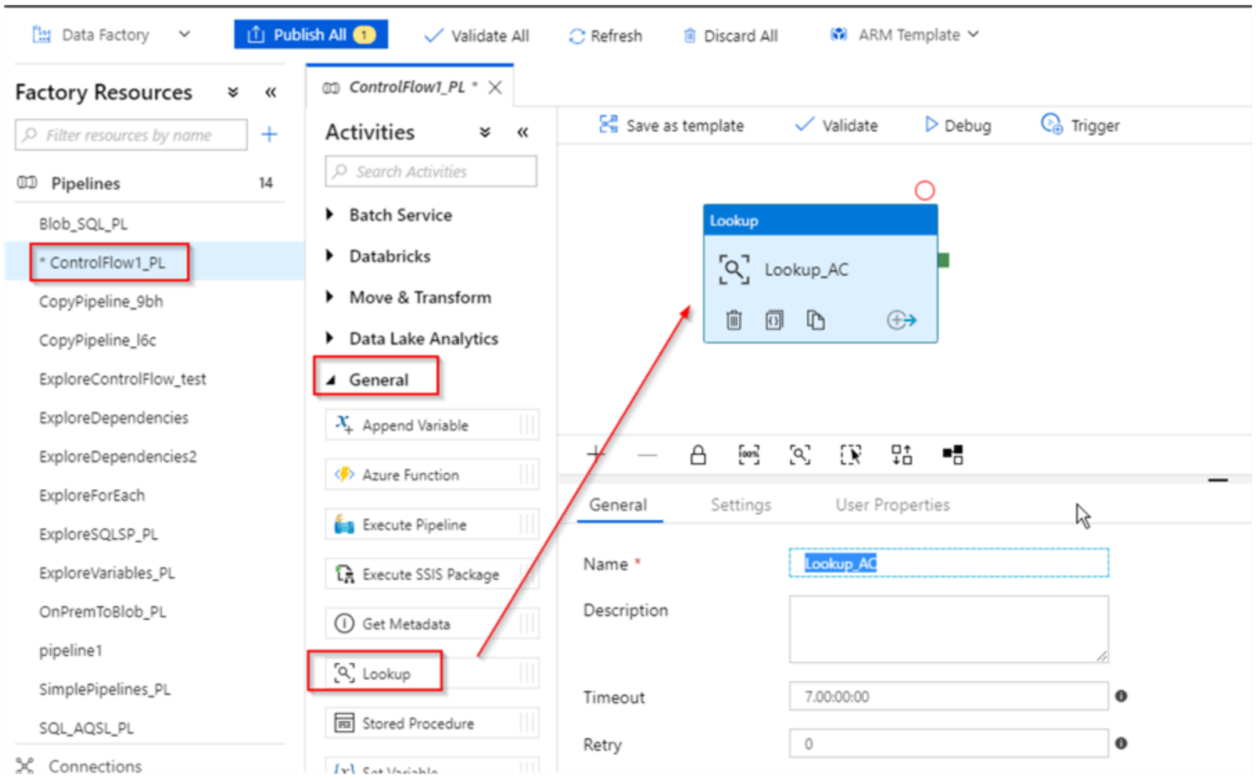
ACTIVITY NAME	ACTIVITY TYPE	ACTIONS	RUN START	DURATION	STATUS
If_Condition_AC	IfCondition		02/09/2019 7:25 PM	00:00:18	✓ Succeeded
FactInternetSales_AC	Copy		02/09/2019 7:25 PM	00:00:16	✓ Succeeded

Lookup activity - Azure Data Factory

In Azure Data Factory, a **Lookup activity** is used to retrieve a single row or a set of rows from a data source. This activity is particularly useful when you want to access metadata or configuration information that can be used later in your data processing workflows.

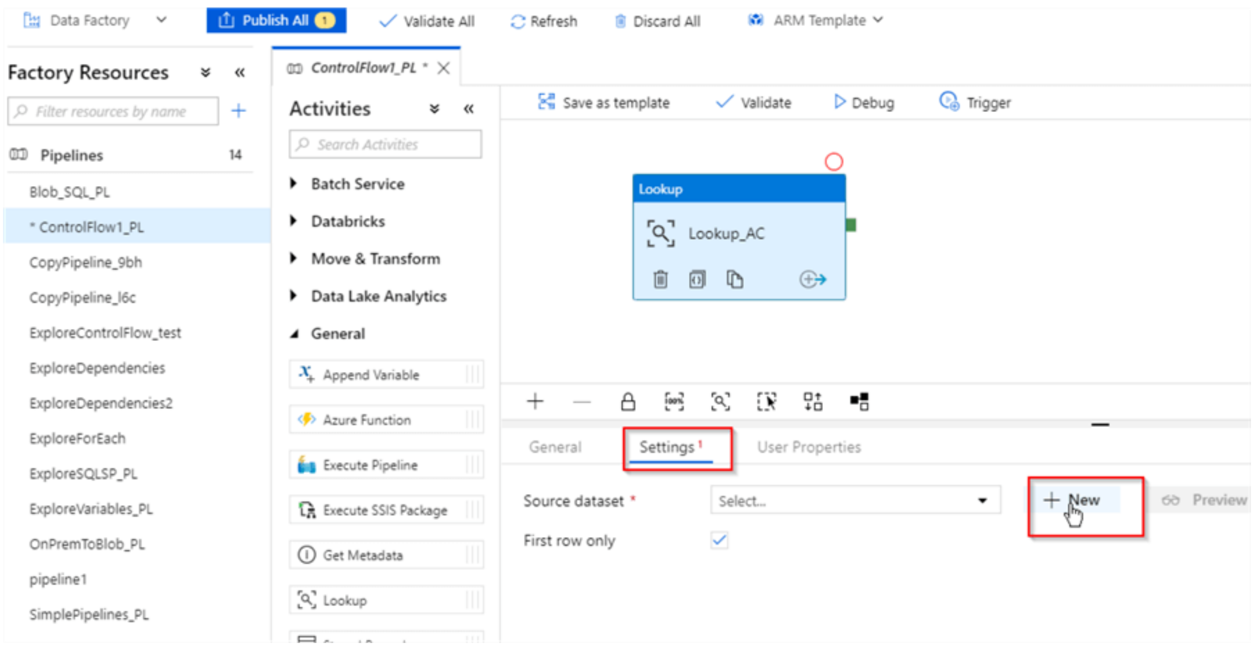
Note the following:

- The Lookup activity can return up to **5000 rows**; if the result set contains more records, the first 5000 rows will be returned.
- The Lookup activity output supports up to **4 MB** in size, activity will fail if the size exceeds the limit.
- The longest duration for Lookup activity before timeout is **24 hours**.



Name * Lookup_AC
Description
Timeout 7.00:00:00
Retry 0

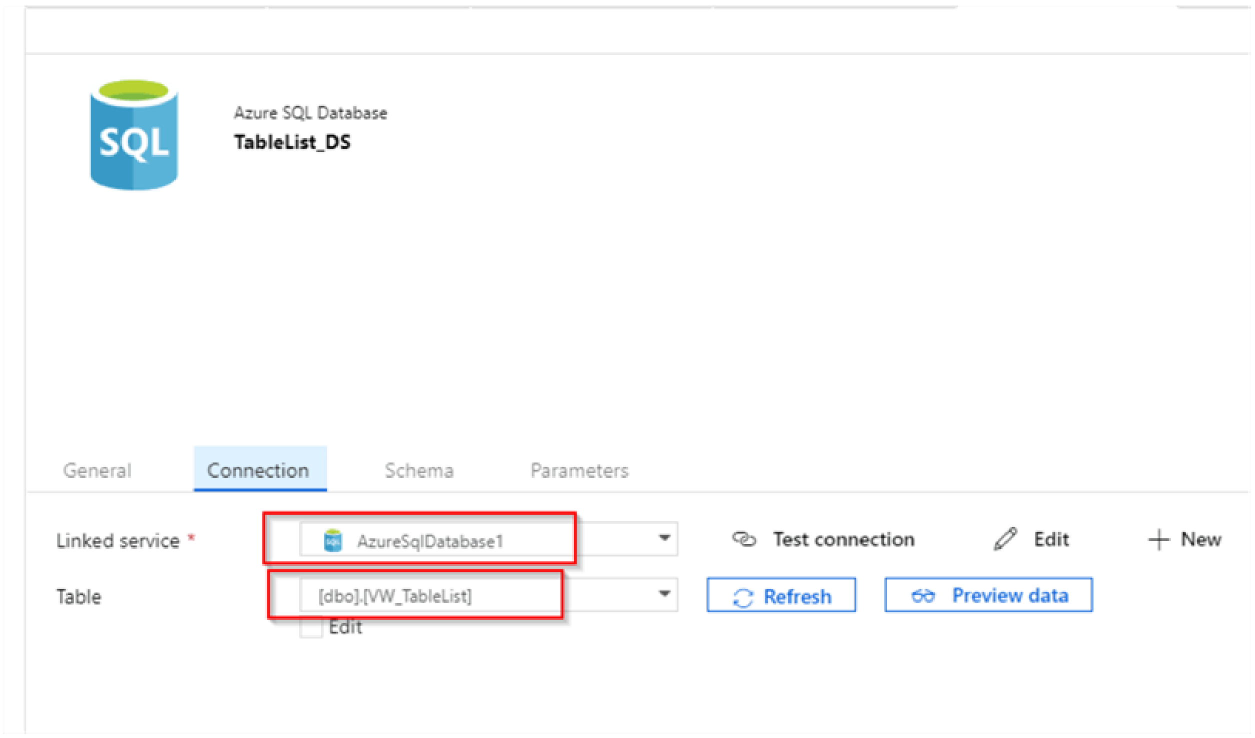
Switch to the Settings tab, click '+New' button to create a dataset, linked to the VW_TableList view in the SrcDb database:



Source dataset * Select...

First row only ✓

I've named the new dataset TableList_DS, see the below properties:

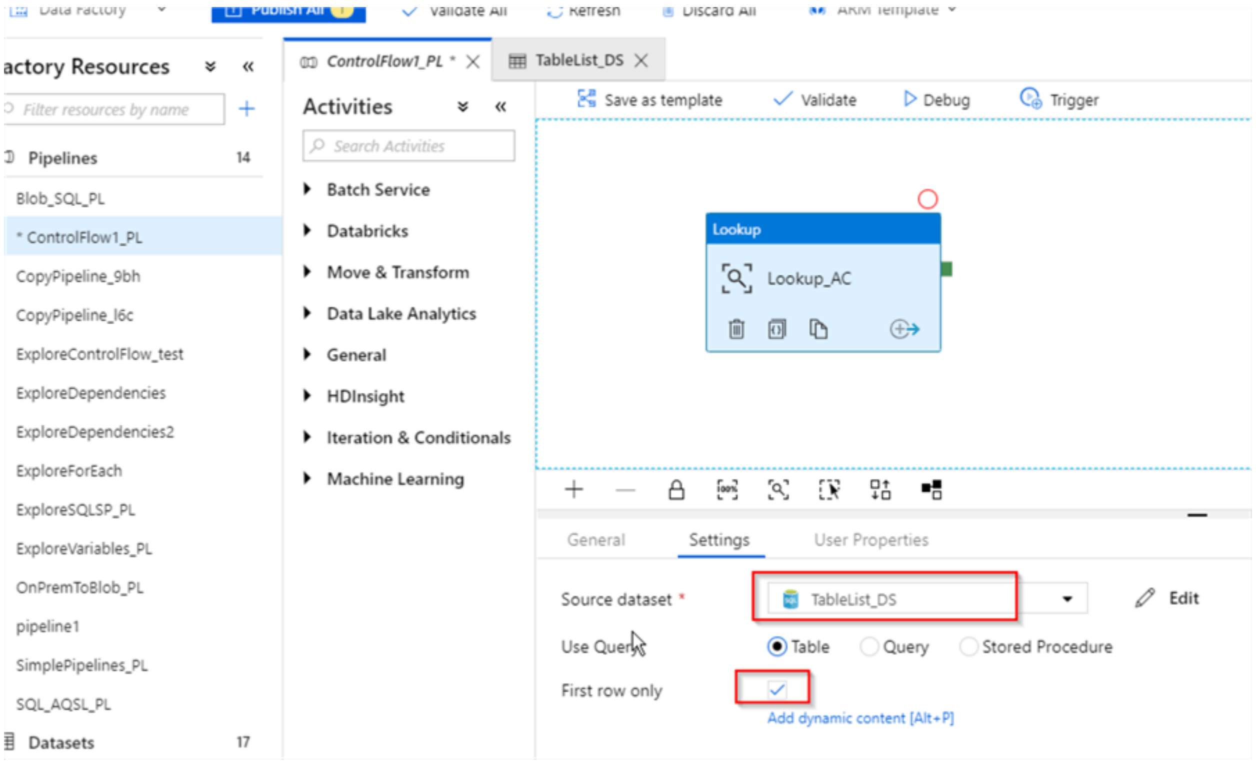


Linked service * AzureSqlDatabase1

Table [dbo].[VW_TableList]

Edit

The below screenshot shows the properties of the Lookup activity, with the new dataset configured. Please note that 'First row only' checkbox is checked, which will ensure that this activity produces only the first row from its data source:

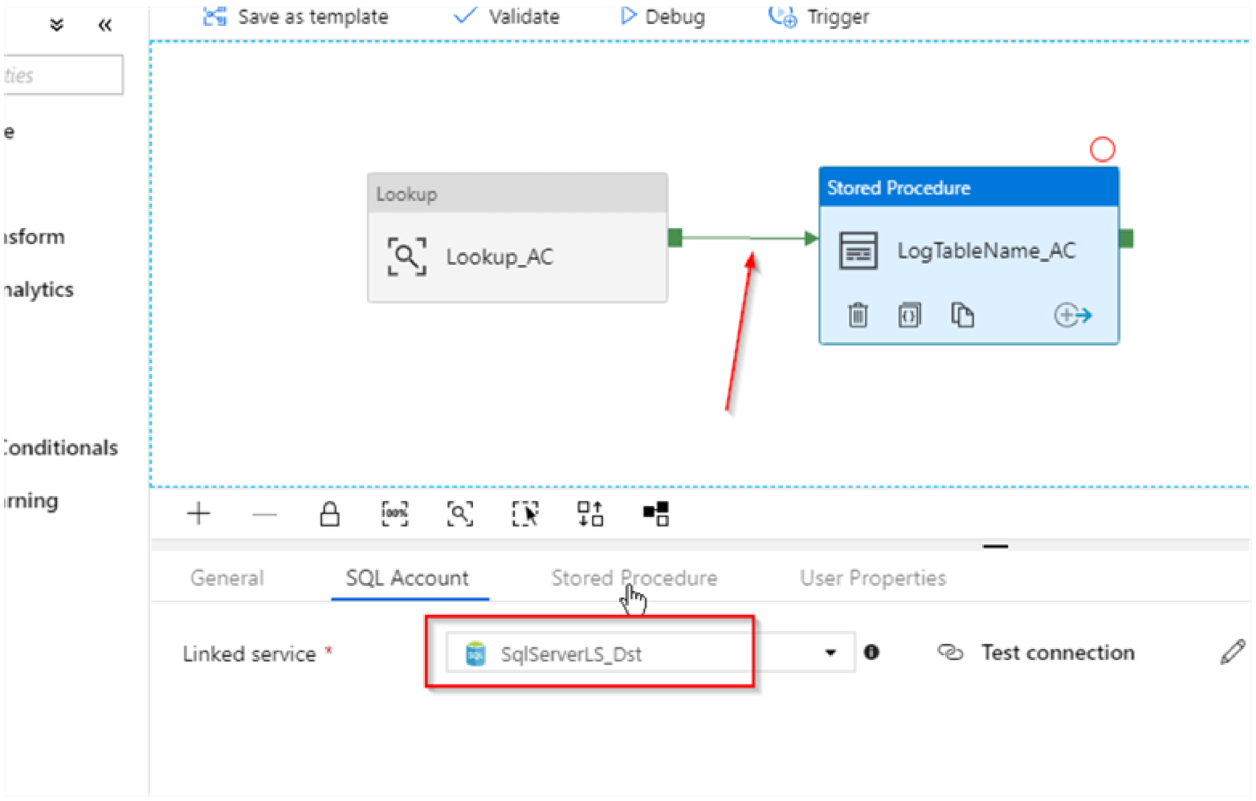


Source dataset * TableList_DS

Use Query [✓ Table] Query Stored Procedure

First row only ✓

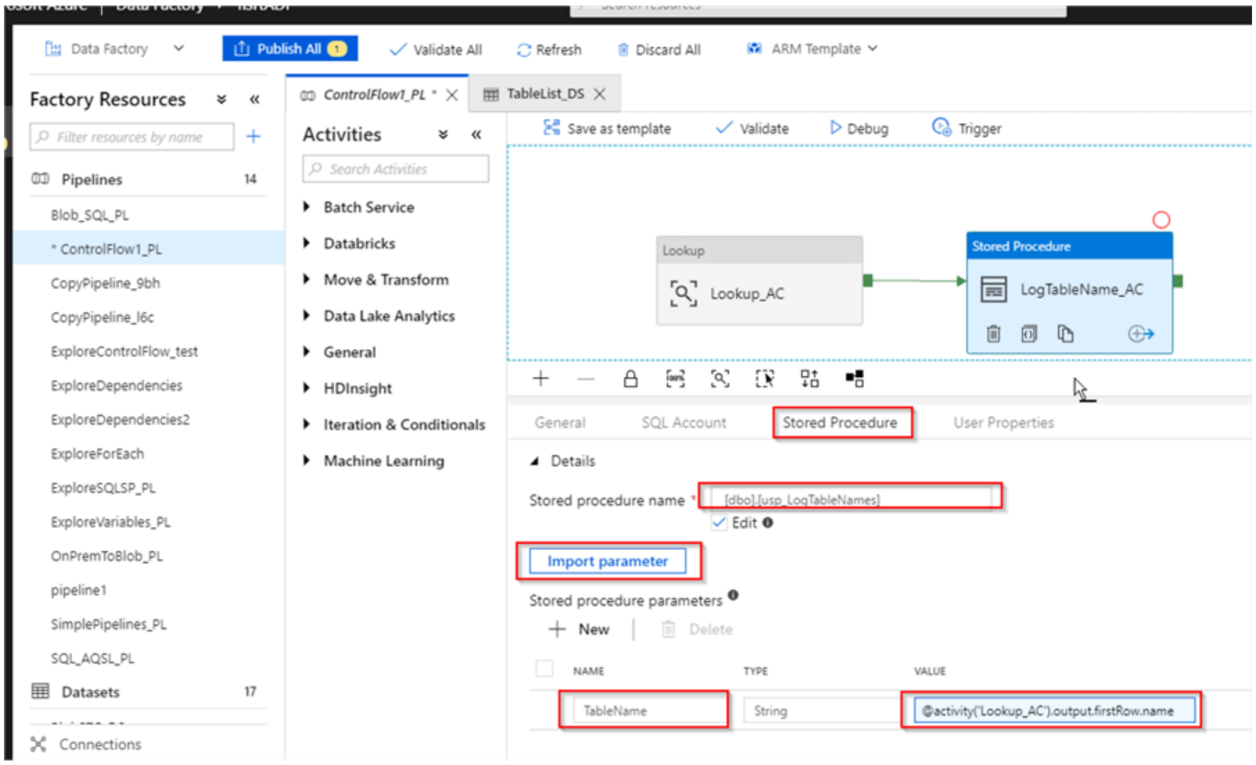
Next, let's add Stored Procedure activity,



Linked service * SqlServerLS_Dst

Next, switch to Stored Procedure tab, enter [dbo].[usp_LogTableNames] as the procedure's name, fetch the procedure's parameter, using the Import parameter button and enter the dynamic expression

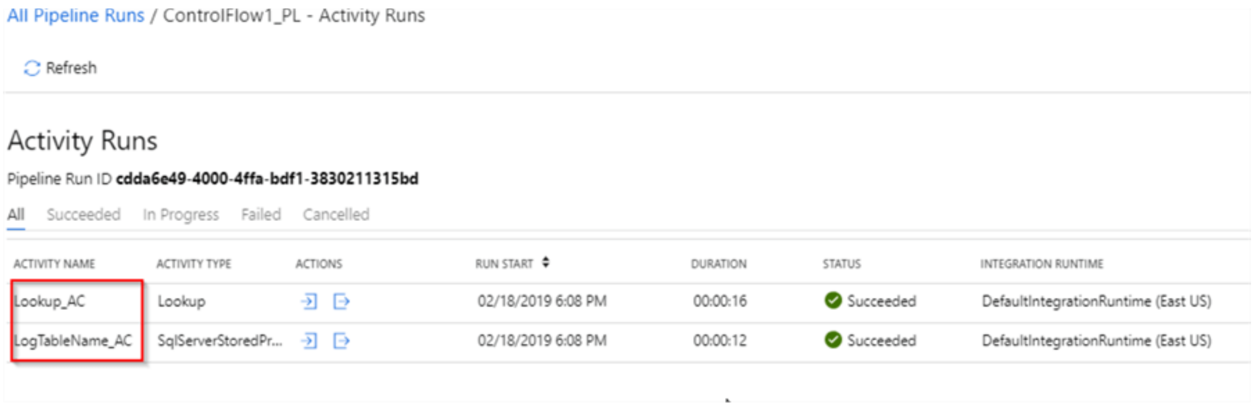
@activity('Lookup_AC').output.firstRow.name as its value. This expression reflects the data output from the Lookup activity:



Stored procedure name * [dbo].[usp_LogTableNames]
Edit
Stored procedure parameters

NAME	TYPE	VALUE
TableName	String	@activity('Lookup_AC').output.firstRow.name

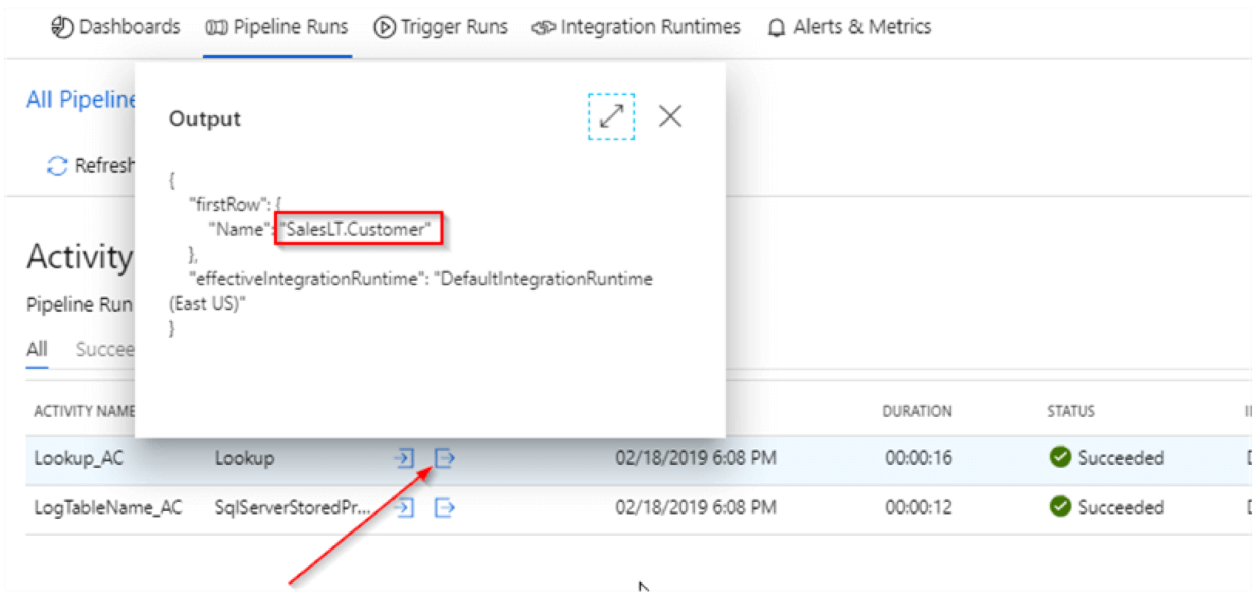
Finally, let's publish the changes, trigger it manually, switch to the Monitor page and open the Activity Runs window to examine the detailed execution logs:



Pipeline Run ID **cdda6e49-4000-4ffa-bdf1-3830211315bd**

ACTIVITY NAME	ACTIVITY TYPE	ACTIONS	RUN START	DURATION	STATUS	INTEGRATION RUNTIME
Lookup_AC	Lookup		02/18/2019 6:08 PM	00:00:16	Succeeded	DefaultIntegrationRuntime (East US)
LogTableName_AC	SqlServerStoredPr...		02/18/2019 6:08 PM	00:00:12	Succeeded	DefaultIntegrationRuntime (East US)

Using the Output button, we can examine the output of the lookup activity and see the value it produced:



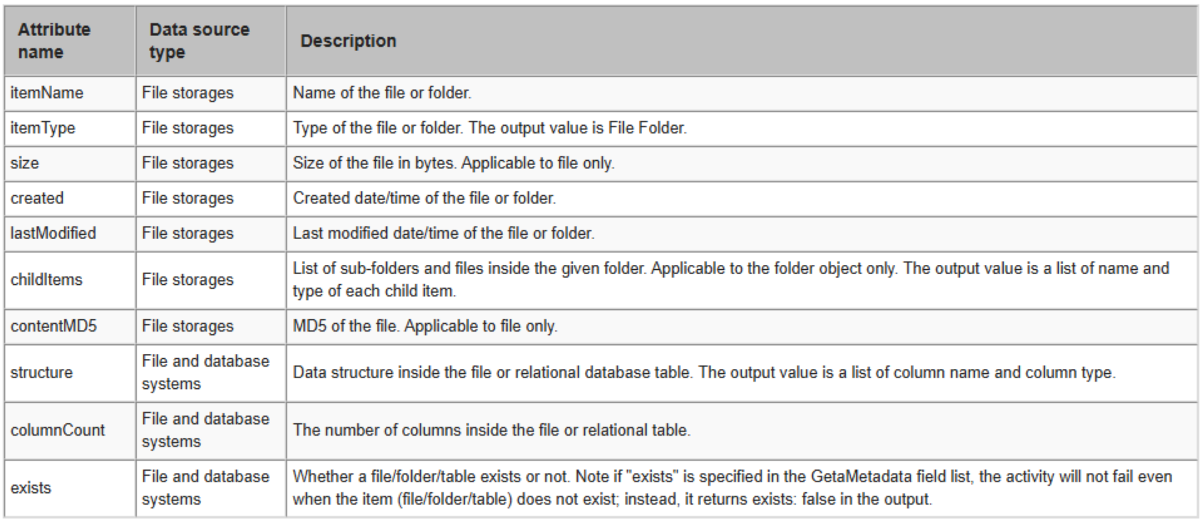
```
{
    "firstRow": {
        "Name": "SalesLT.Customer"
    },
    "effectiveIntegrationRuntime": "DefaultIntegrationRuntime
(East US)"
}
```

ACTIVITY NAME			DURATION	STATUS
Lookup_AC	Lookup	02/18/2019 6:08 PM	00:00:16	Succeeded
LogTableName_AC	SqlServerStoredPr...	02/18/2019 6:08 PM	00:00:12	Succeeded

Get Metadata Activity in ADF

The **Get Metadata** activity allows reading metadata information of its sources.

The list of attributes returned by this activity is dependent on its source type, some attributes are available only for file-based sources, others available for database tables and there are few attributes applicable for both types. Here is the full list of attributes, borrowed from Microsoft's site:



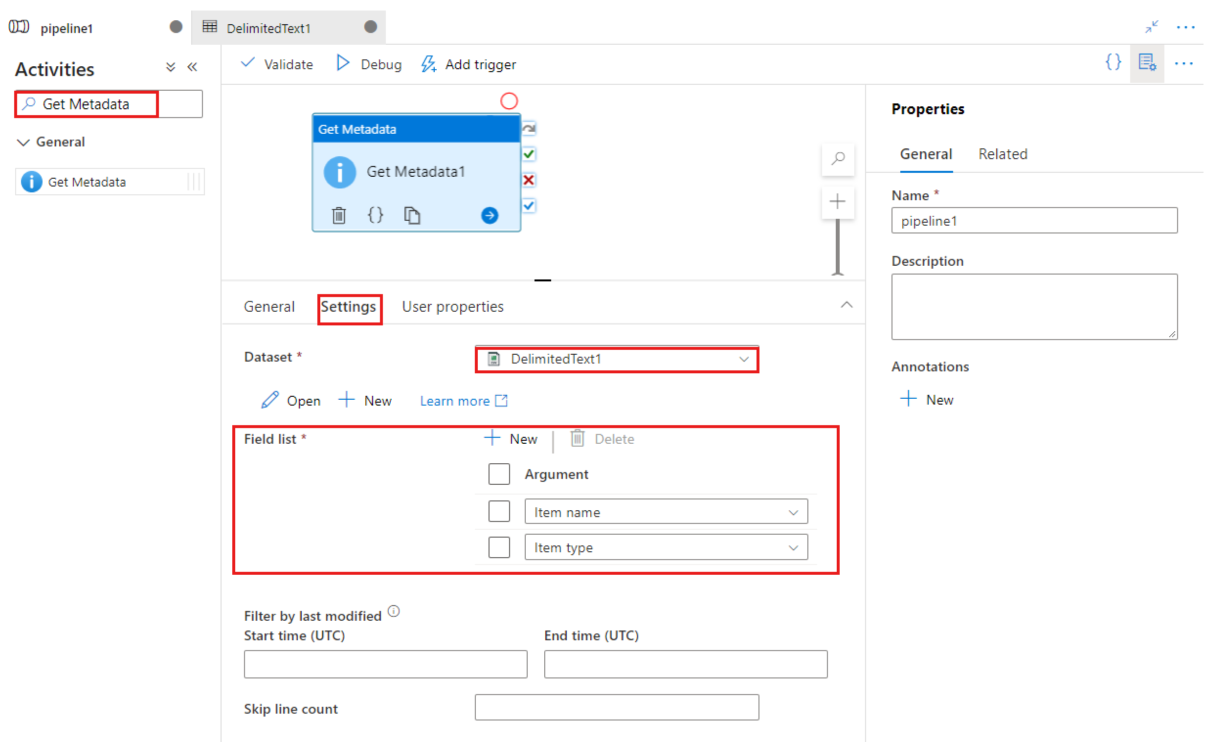
Attribute name	Data source type	Description
itemName	File storages	Name of the file or folder.
itemType	File storages	Type of the file or folder. The output value is File Folder.
size	File storages	Size of the file in bytes. Applicable to file only.
created	File storages	Created date/time of the file or folder.
lastModified	File storages	Last modified date/time of the file or folder.
childItems	File storages	List of sub-folders and files inside the given folder. Applicable to the folder object only. The output value is a list of name and type of each child item.
contentMD5	File storages	MD5 of the file. Applicable to file only.
structure	File and database systems	Data structure inside the file or relational database table. The output value is a list of column name and column type.
columnCount	File and database systems	The number of columns inside the file or relational table.
exists	File and database systems	Whether a file/folder/table exists or not. Note if "exists" is specified in the GetaMetadata field list, the activity will not fail even when the item (file/folder/table) does not exist; instead, it returns exists: false in the output.

Please note that the **childItems** attribute from this list is applicable to folders only and is designed to provide list of files and folders nested within the source folder.

The data obtained by **Get Metadata** activity can be used by subsequent iterative activities, to perform copy or transformation activities on a dynamic basis.

Use a Get Metadata activity in a pipeline, complete the following steps:

1. Search for *Get Metadata* in the pipeline Activities pane, and drag a Fail activity to the pipeline canvas.
2. Select the new Get Metadata activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.
3. Choose a dataset, or create a new one with the New button. Then you can specify filter options and add columns from the available metadata for the dataset.



Name * pipeline1

Description

Annotations

Dataset * DelimitedText1

Field list * Item name, Item type

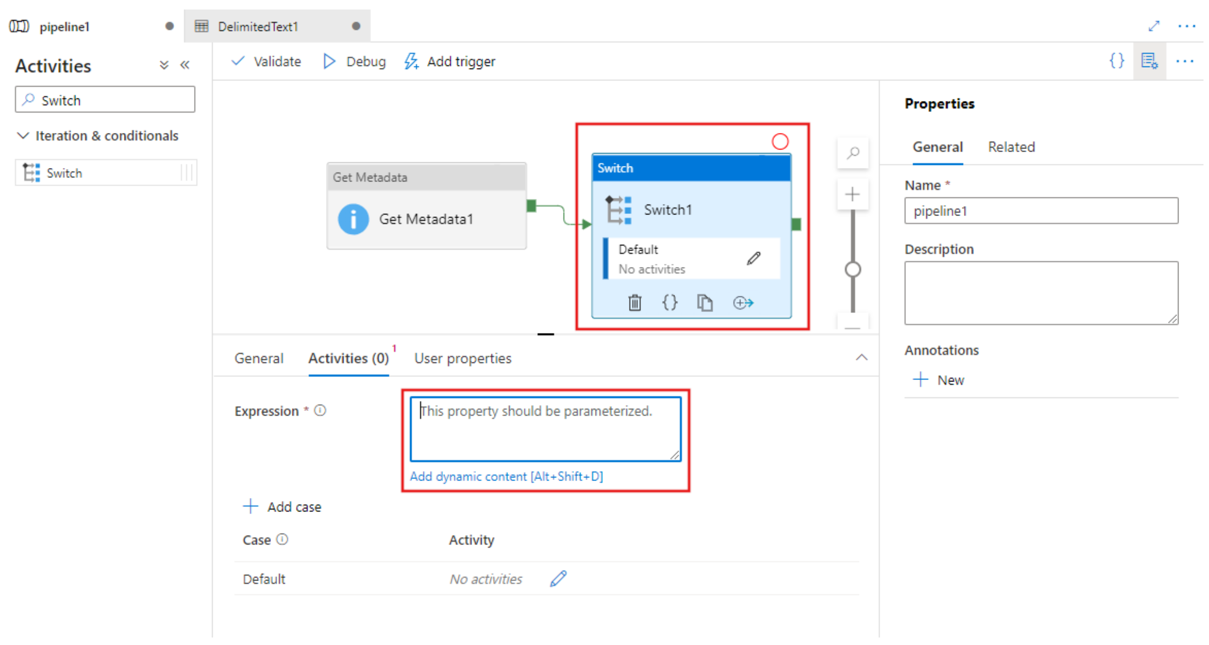
Filter by last modified

Start time (UTC)

End time (UTC)

Skip line count

4. Use the output of the activity as an input to another activity, like a Switch activity in this example. You can reference the output of the Metadata Activity anywhere dynamic content is supported in the other activity.



Name * pipeline1

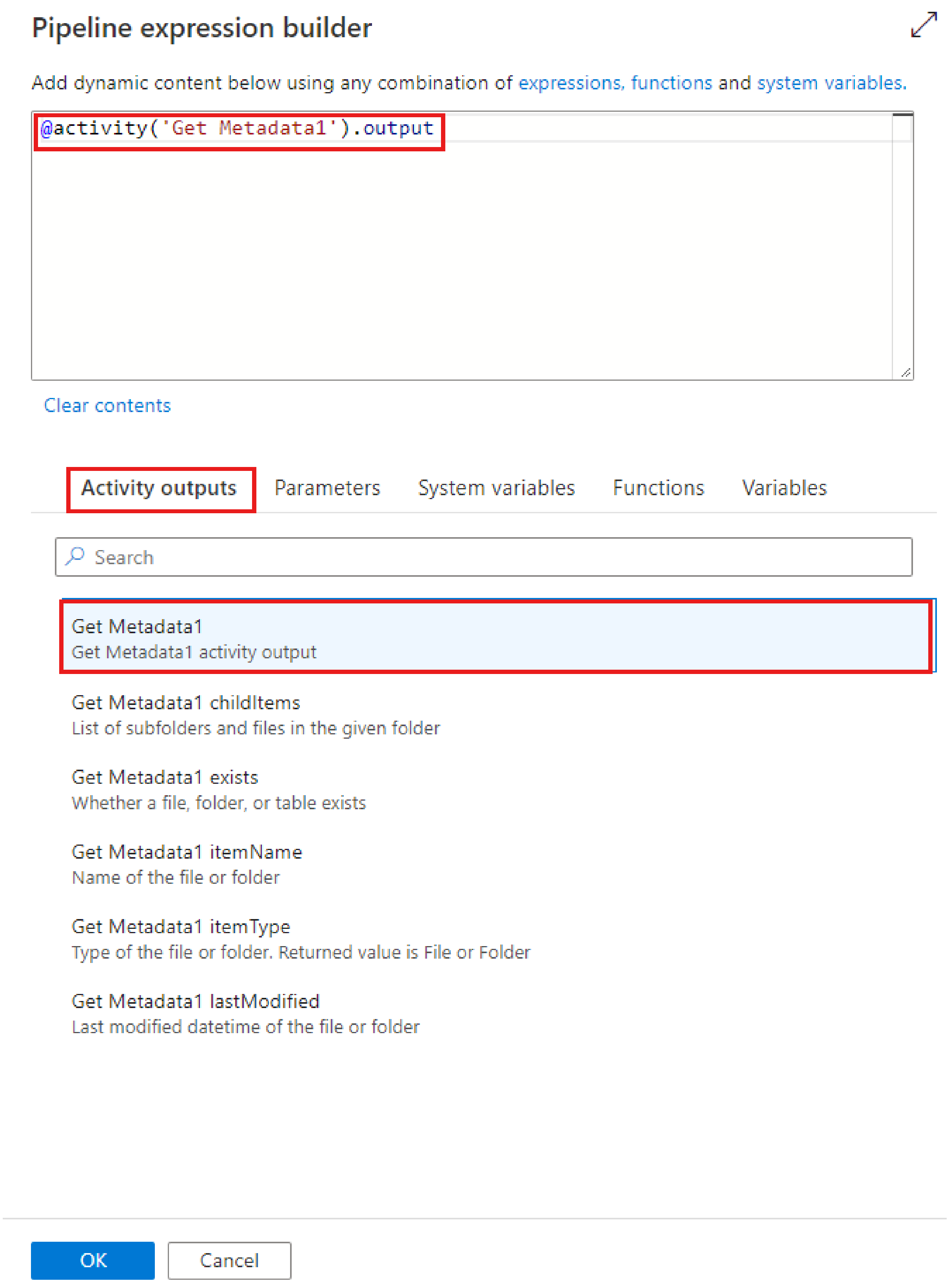
Description

Annotations

Expression * This property should be parameterized.

Case	Activity
Default	No activities

5. In the dynamic content editor, select the Get Metadata activity output to reference it in the other activity.



Pipeline expression builder

Add dynamic content below using any combination of expressions, functions and system variables.

```
@activity('Get Metadata1').output
```

Clear contents

Get Metadata1
Get Metadata1 activity output

Get Metadata1 childItems
List of subfolders and files in the given folder

Get Metadata1 exists
Whether a file, folder, or table exists

Get Metadata1 itemName
Name of the file or folder

Get Metadata1 itemType
Type of the file or folder. Returned value is File or Folder

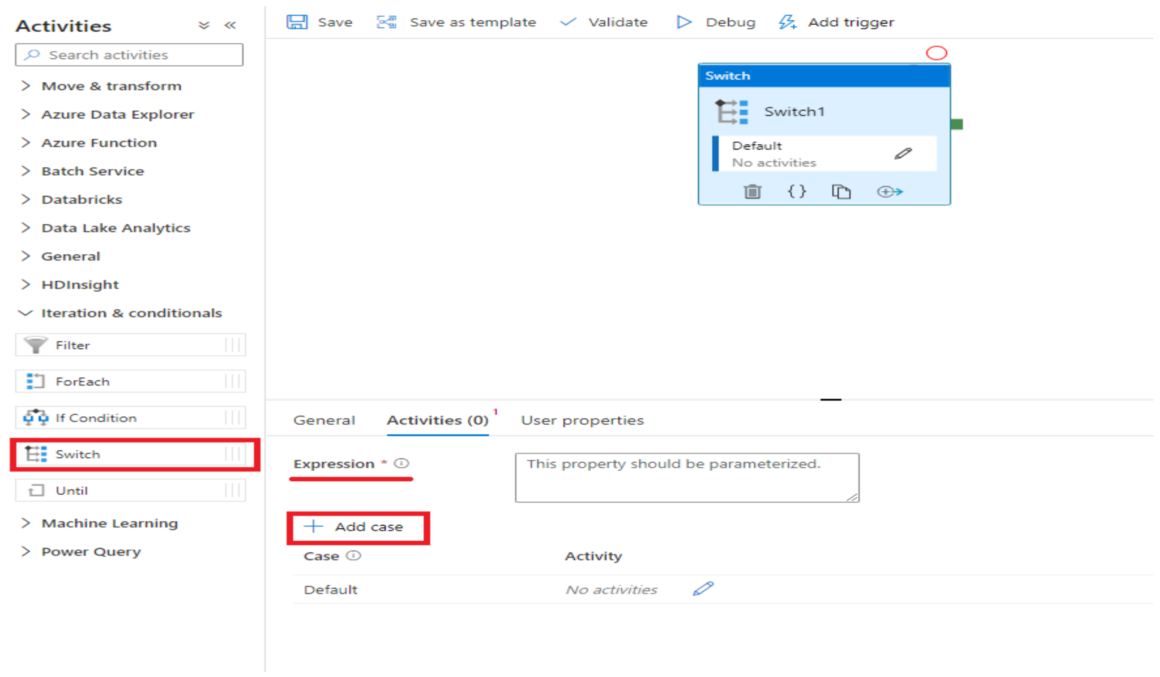
Get Metadata1 lastModified
Last modified datetime of the file or folder

Azure Data Factory: Switch Activity

While designing ETL data flows, we will come across some scenarios where we must branch the *control flow* based on a business rule or conditional expression. The easiest way to implement conditional branching, in *Data Factory* is by using the *Switch Activity.* It is very easy to understand and implement if you are familiar with the SQL *Case* Statement.

Using the *switch activity*, the control flow of the *pipeline* can be branched into multiple *Cases* based on a conditional expression. Each *case* in the *switch activity* can be used to pass the control flow to a new *activity* (or group of *activities*).

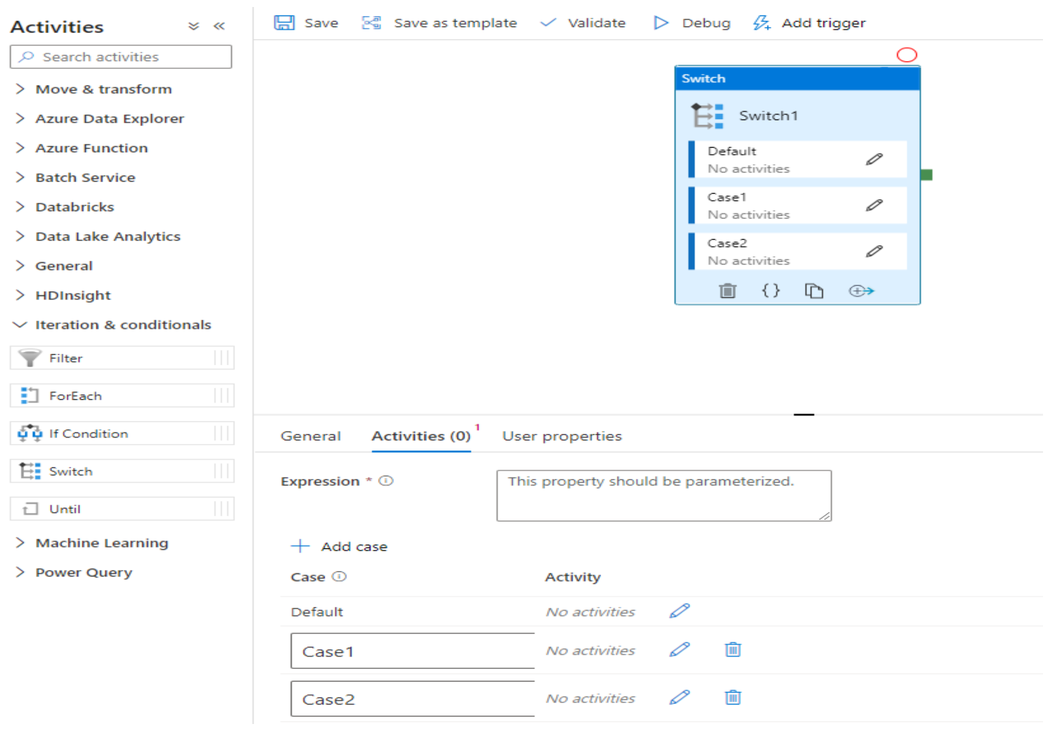
When we create a new *Switch activity*, it already has one case- the *default* case. This is the equivalent of the *ELSE* clause of the SQL *Case* statement.



Azure Data Factory: Switch Activity

The *Expression* property under *Activities* tab is a mandatory property and the branching is performed based on the expression evaluation.

Users can add more *Cases* using the *Add case* button.

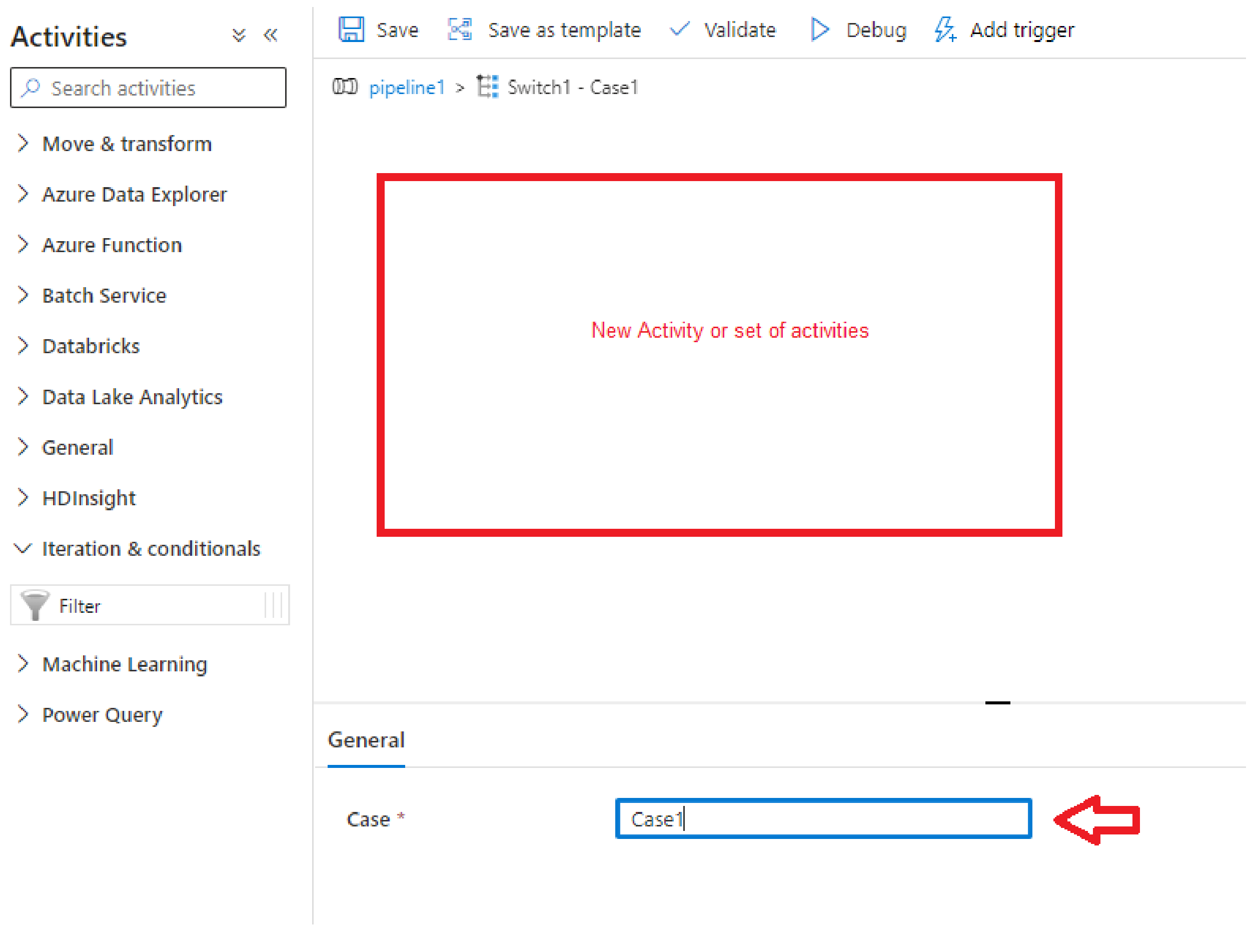


Expression * This property should be parameterized.

Case	Activity
Default	*No activities*
Case1	*No activities*
Case2	*No activities*

Azure Data Factory: Switch Activity User Defined Cases

Once the new *case* is added, by clicking on the pencil icon, users can provide the *expression evaluation value* for that *case* and design a new *activity* or set of *activities* to be executed when the *expression value* matches the *case value.*



Case * Case1

Azure Data Factory: Switch Activity Case Value

Like the SQL *Case* statement, if the *expression value* matches none of the user created *case values,* the control flow will be passed to the *Default* case.

Until activity in Azure Data Factory

The Until activity in Azure Data Factory (ADF) is a control flow activity that allows you to repeatedly execute a set of activities until a specified condition is met. It is like a "do-while" loop in traditional programming languages. The Until activity is particularly useful for scenarios where you need to perform operations such as polling an external system for a status update, retrying operations until they succeed, or processing data in chunks until all data has been processed.

ADF- What is Until activity in ADF?

Definition:

This is an activity in Azure Data Factory, using which you can run a set of activities in a loop until a specific condition is met.

- This is like a **Do While** loop we write in programming.
- This activity guarantees that the loop will execute at least once and check the condition at the end.

ADF – Until activity implementation

Requirement:

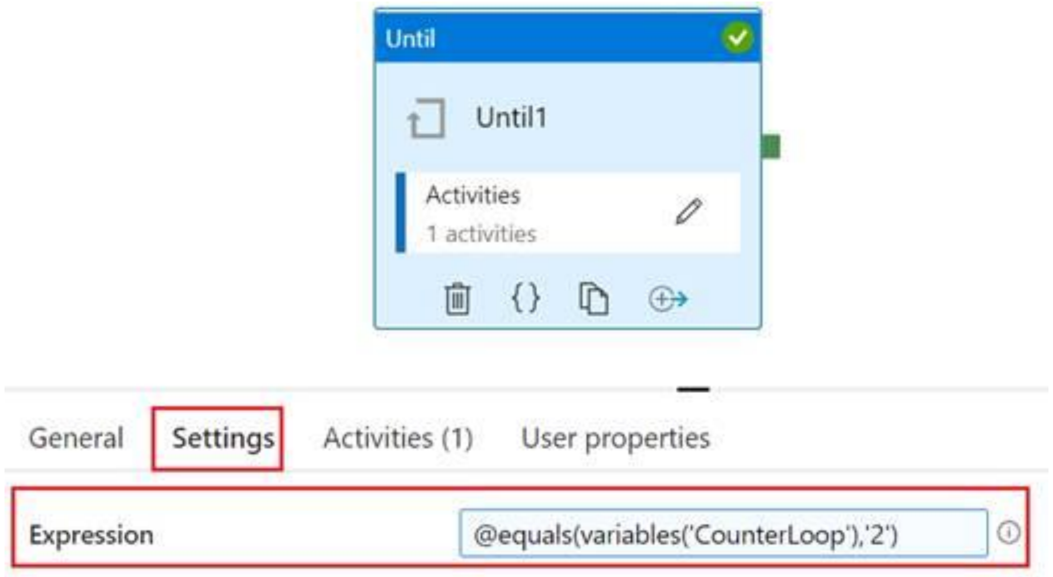
- The requirement is quite simple and for explanatory purposes. It will obviously be more challenging scenarios when working on live projects.
- The variable value is 1 and we will run activity until the variable's value is 2.

Implementation:

- Create a variable with value = 1.

- Add an Until Activity.
    - Click on the activity.
    - In the **Settings** tab, there is a property called **Expression**.
    - **Expression**: Specify the condition that determines when the loop should stop. The condition must evaluate to a Boolean value (**true** or **false**). The loop continues until the expression evaluates to **true**.

        - Example: **@equals(activity('CheckStatus').output.status, 'Completed')**
    - Until the Expression output value is not true, the Activity within this Until will keep running.



Expression @equals(variables('CounterLoop'),'2')

- As you can see in the screenshot, the expression says that do this activity until the value of the Variable becomes 2.
- If we go within the Until activity, we have used a Set activity, which sets the variable to value 2 (Keeping it simple).

    - **Activities**: Inside the Until activity, add one or more activities that should be executed in each iteration of the loop. This can include any control flow or data movement activities, such as a Web Activity to check the status of an external process or a Copy Activity to move data in chunks.

- Now, as the variable value becomes 2, it will not execute the next time, and it will just end the activity.

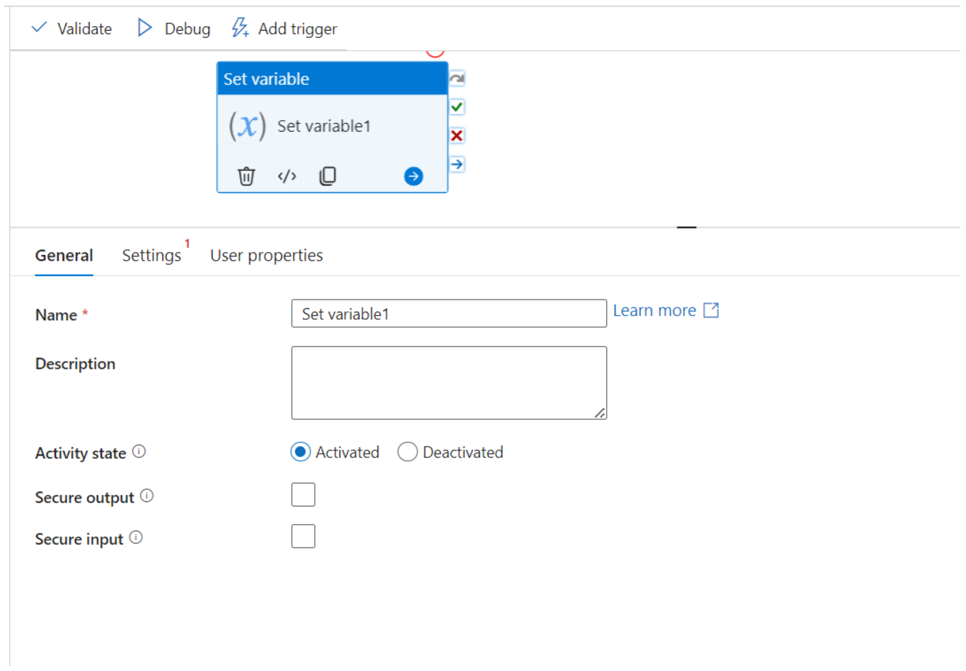
- **Timeout**: Optionally, you can specify a timeout for the Until activity. If the specified time elapses before the condition evaluates to **true**, the Until activity will fail.
- **Settings**:

    - **Batch Count**: The number of iterations to run in parallel, if applicable.
    - **Delay**: The time to wait between iterations, specified in seconds. This is useful for scenarios like polling, where you want to wait for a certain amount of time before checking a condition again.

Usage Scenarios

- **Polling an External Process:** One common use case for the Until activity is to poll an external process or service until a specific condition is met, such as waiting for a file to be available in a storage account or waiting for a long-running process to complete.
- **Retry Mechanisms:** The Until activity can be used to implement retry logic for operations that may fail due to transient errors. By enclosing the operation in an Until activity with a condition that checks for success, you can ensure the operation is retried until it succeeds, or a maximum number of attempts is reached.
- **Processing Data in Chunks:** In scenarios where you need to process large volumes of data in chunks, the Until activity can be used to iteratively process each chunk until all data has been processed. This is particularly useful when dealing with APIs that have rate limits or when processing large files that need to be split into smaller, more manageable pieces.

Set Variable Activity in ADF



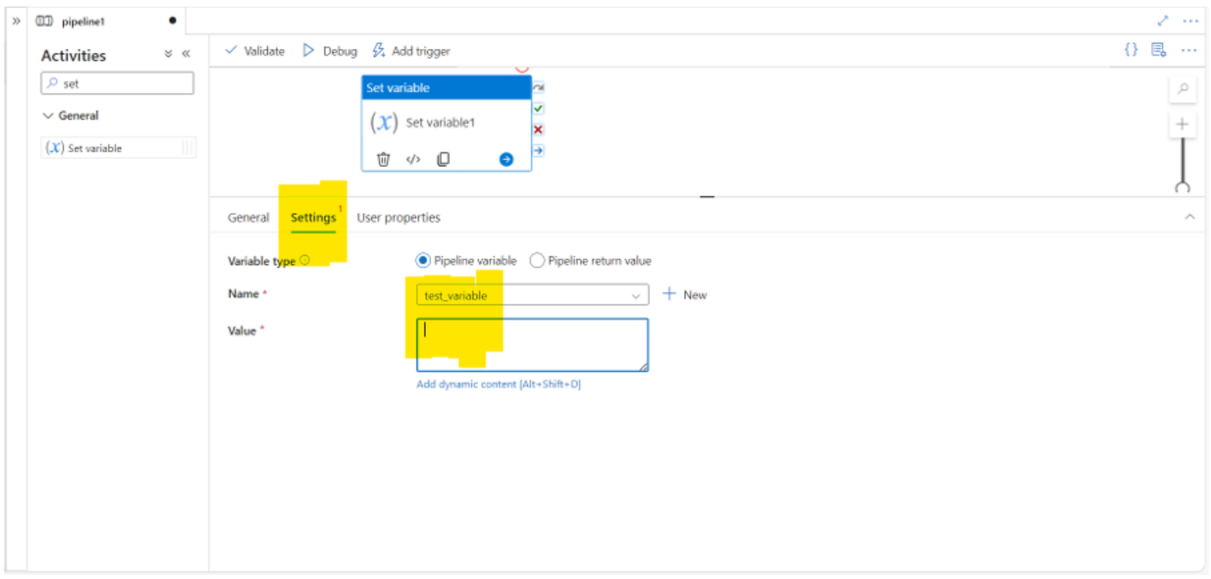
Name * Set variable1

Description

Activity state [✓ Activated] Deactivated

Secure output ☐

Secure input ☐



Variable type [✓ Pipeline variable] Pipeline return value

Name * test_variable

Value *

The **Set Variable** activity in Azure Data Factory (ADF) is used to assign a value to a variable during the execution of a pipeline. This is particularly useful for managing temporary data or for controlling the flow of your pipeline based on computed values.

How to Use the Set Variable Activity

Here's a step-by-step guide on how to implement the Set Variable activity:

1. Open Azure Data Factory

- Navigate to your Azure Data Factory instance in the Azure portal.

2. Create or Open a Pipeline

- Go to the **Author** tab.
- Create a new pipeline by clicking the "+" icon or open an existing one.

3. Add a Variable

- In the pipeline canvas, go to the **Variables** tab.
- Click on "New" to create a new variable.
- Define the variable name, type (e.g., String, Int, Bool), and an initial/default value.

4. Add the Set Variable Activity

- In the Activities pane, search for **Set Variable**.
- Drag and drop the **Set Variable** activity onto the pipeline canvas.

5. Configure the Set Variable Activity

- Select the Set Variable activity on the canvas.
- In the **Settings** tab, choose the variable you created from the dropdown.
- Assign a value to the variable:
  - You can set a static value directly.
  - Use dynamic expressions to derive the value from other activities. Click on "Add dynamic content" to input expressions.

6. Link Activities

- Connect the Set Variable activity to other activities as necessary. This ensures the variable is set before it is used in subsequent activities.

7. Debug and Test

- Use the **Debug** feature to run your pipeline and monitor the value of the variable. Check the output to ensure it has been set correctly.

8. Publish Changes

- After testing, click on "Publish" to save and deploy your changes.