

# Azure Data Engineering Project

Mohammed Ismail Shaikh

---

## Data Ingestion using Azure Data Factory

### Introduction

In this project, I designed dynamic pipelines in Azure Data Factory (ADF) to ingest multiple datasets from GitHub into Azure Data Lake Storage Gen2 (ADLS Gen2), enabling scalable and automated data ingestion.

### Tools and Concepts

For this project, I used Azure Data Factory (ADF) and Azure Data Lake Storage Gen2 (ADLS Gen2) to implement key concepts such as data ingestion, dynamic pipelines, and efficient data organization within the storage account.

### Understanding Dataset

I have four files on my GitHub page: `netflix_cast.csv`, `netflix_category.csv`, `netflix_countries.csv`, and `netflix_directors.csv`. I will ingest these files into Azure Data Lake using Azure Data Factory. To manage this process, I am using `netflix_titles.csv` as a validation file. I uploaded this file directly to the raw container in ADLS, and it will be used to ensure that the pipeline only runs when this file is available.

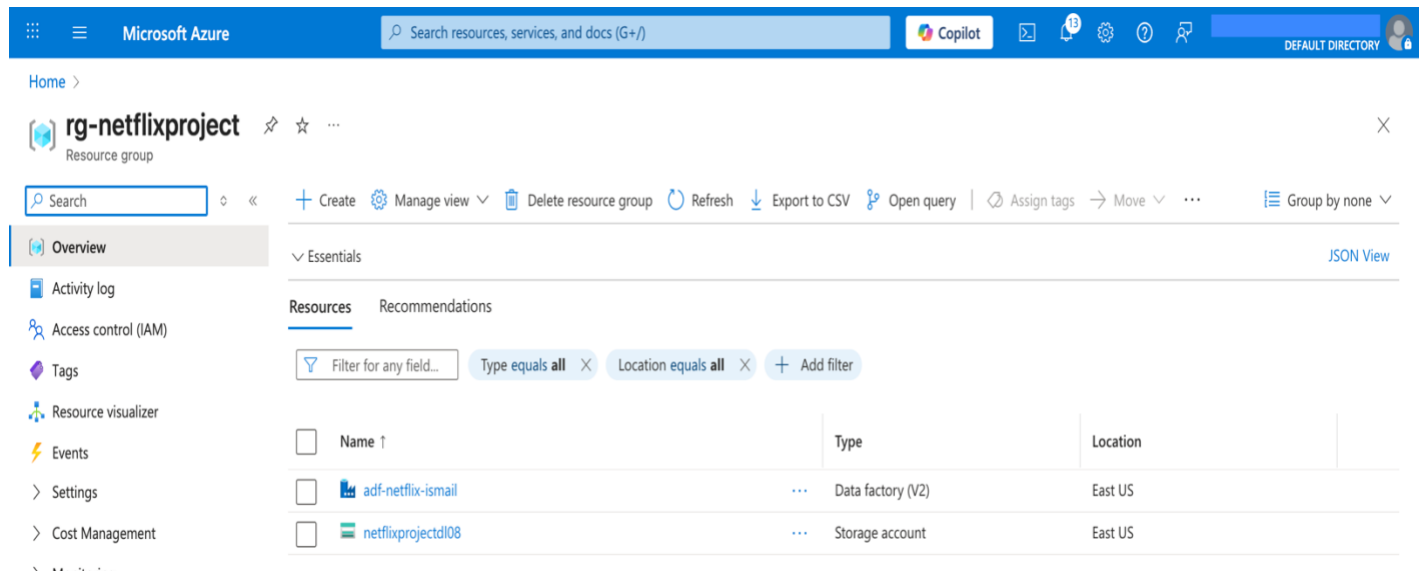
### Data Source

This [repository](#) contains the raw Netflix datasets used for ingestion in the Azure Data Factory pipeline.

# How I Set up ADLS and ADF

First, I created a resource group in the East US region to organize related Azure resources.

Inside this resource group, I set up Azure Data Factory (ADF) to handle data orchestration and an Azure Storage Account with Data Lake Storage Gen2 enabled to store and manage the ingested data."

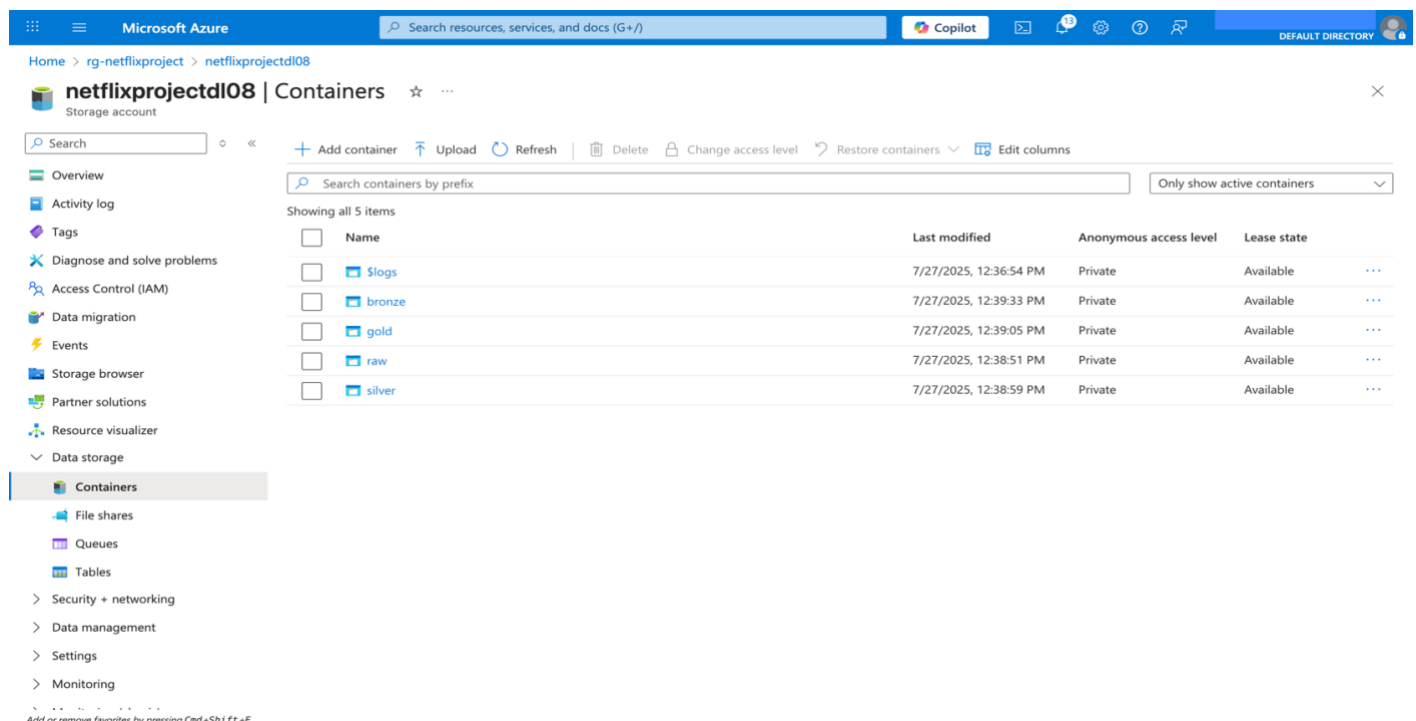


The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with the Microsoft Azure logo, a search bar, and various icons. Below the navigation bar, the breadcrumb path is 'Home > rg-netflixproject'. The main content area shows the 'Resources' tab for the 'rg-netflixproject' resource group. A table lists the resources:

Name	Type	Location
adf-netflix-ismail	Data factory (V2)	East US
netflixprojectdl08	Storage account	East US

## Creating Containers in Azure Data Lake Storage Gen2

I created four containers called raw, bronze, silver, and gold in Azure Data Lake Storage Gen2 to organize data based on its processing stage. Files from GitHub are ingested into the bronze container. I also uploaded the file named netflix\_titles.csv directly into the raw container to use it as a validation reference during pipeline execution.



The screenshot shows the Microsoft Azure portal interface for the 'netflixprojectdl08' storage account. The 'Containers' tab is active, displaying a table of containers within the account.

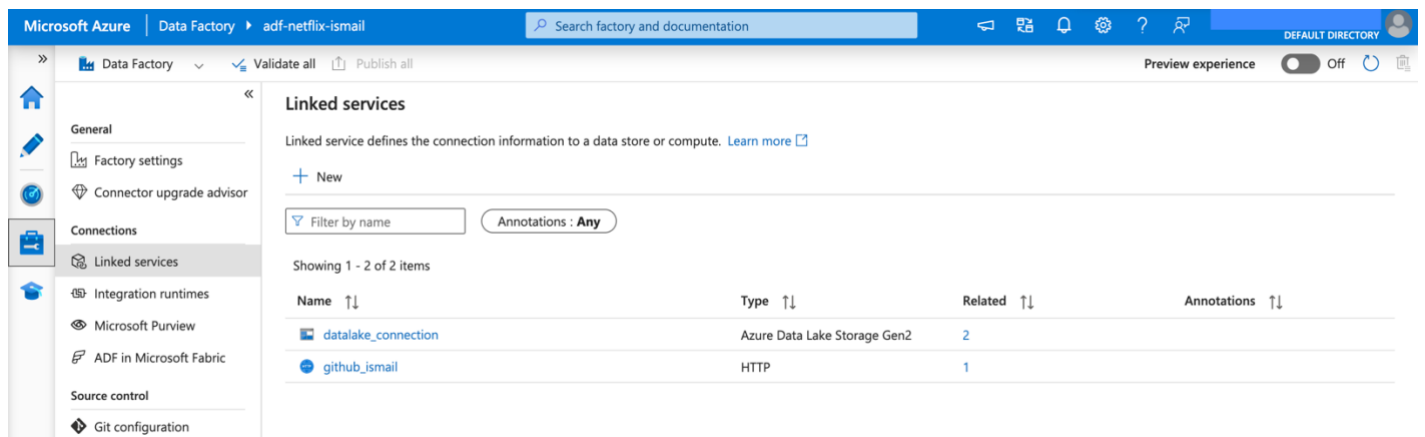
Name	Last modified	Anonymous access level	Lease state
\$logs	7/27/2025, 12:36:54 PM	Private	Available
bronze	7/27/2025, 12:39:33 PM	Private	Available
gold	7/27/2025, 12:39:05 PM	Private	Available
raw	7/27/2025, 12:38:51 PM	Private	Available
silver	7/27/2025, 12:38:59 PM	Private	Available

# Setting Up Linked Services in Azure Data Factory

After setting up the containers, I moved on to Azure Data Factory to set up linked services. These act like connection setups that let ADF access both the source and the destination.

Linked services act as connection strings that allow ADF to securely connect to various data sources and destinations, such as the GitHub data source and the Azure Data Lake Storage Gen2 containers.

In this project, I created one for GitHub to pull in the data and another for Azure Data Lake Storage Gen2, where the data gets stored. This setup is key to making the pipeline work end to end.



I created an HTTP linked service to connect to GitHub and an Azure Data Lake Storage Gen2 linked service to connect to ADLS.

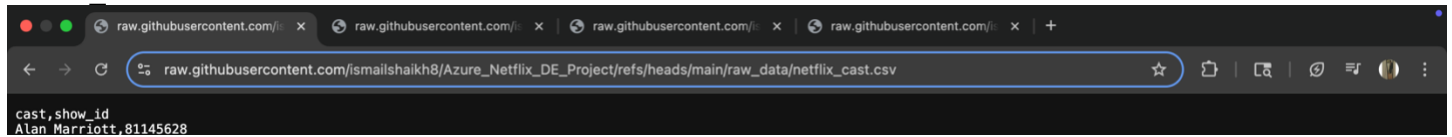
## *Linked Services Used*

- Azure Data Lake Storage Gen2  
Name: datalake\_connection  
Used to connect to the ADLS Gen2 container where data is stored after ingestion.
- HTTP (GitHub)  
Name: github\_ismail  
Configured to access raw data files hosted on a public GitHub repository.

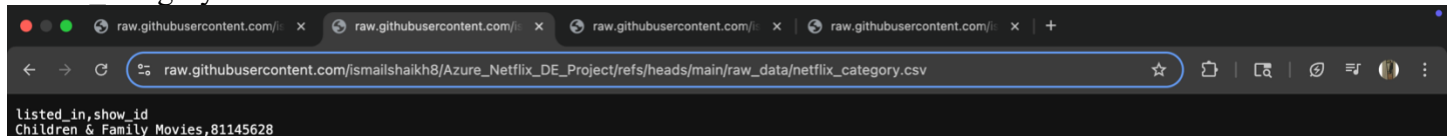
# Understanding the Need for a Dynamic Pipeline

Before building the pipeline, it was necessary to determine the most efficient approach. Since all the GitHub files shared a common relative URL, the only variation was in the file names. To handle this efficiently, I created a dynamic pipeline using parameters. This allowed a single pipeline to process multiple files by simply passing different file names as input, making the solution scalable and easier to maintain.

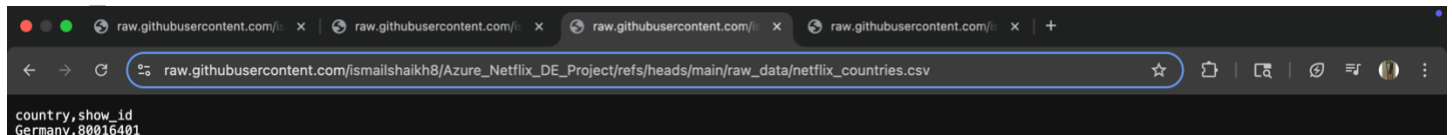
netflix\_cast.csv



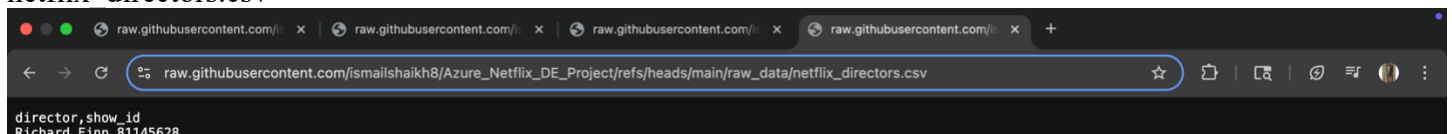
netflix\_category.csv



netflix\_countries.csv



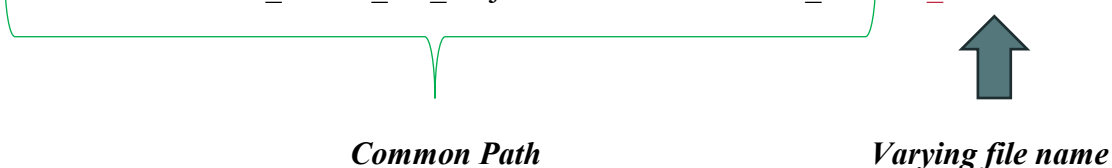
netflix\_directors.csv



If you look at the screenshot above, you'll notice that all the files share the same relative URL, with only the file names varying. Because of this, I created a dynamic pipeline that uses parameters to handle the different file names. This allows me to use a single pipeline to ingest all the files easily, without needing to build separate pipelines for each one. It's a straightforward but effective way to keep the process scalable and efficient.

Base URL: <https://raw.githubusercontent.com/>

Relative URL: ismailshaikh8/Azure\_Netflix\_DE\_Project/refs/heads/main/raw\_data/**file\_name.csv**

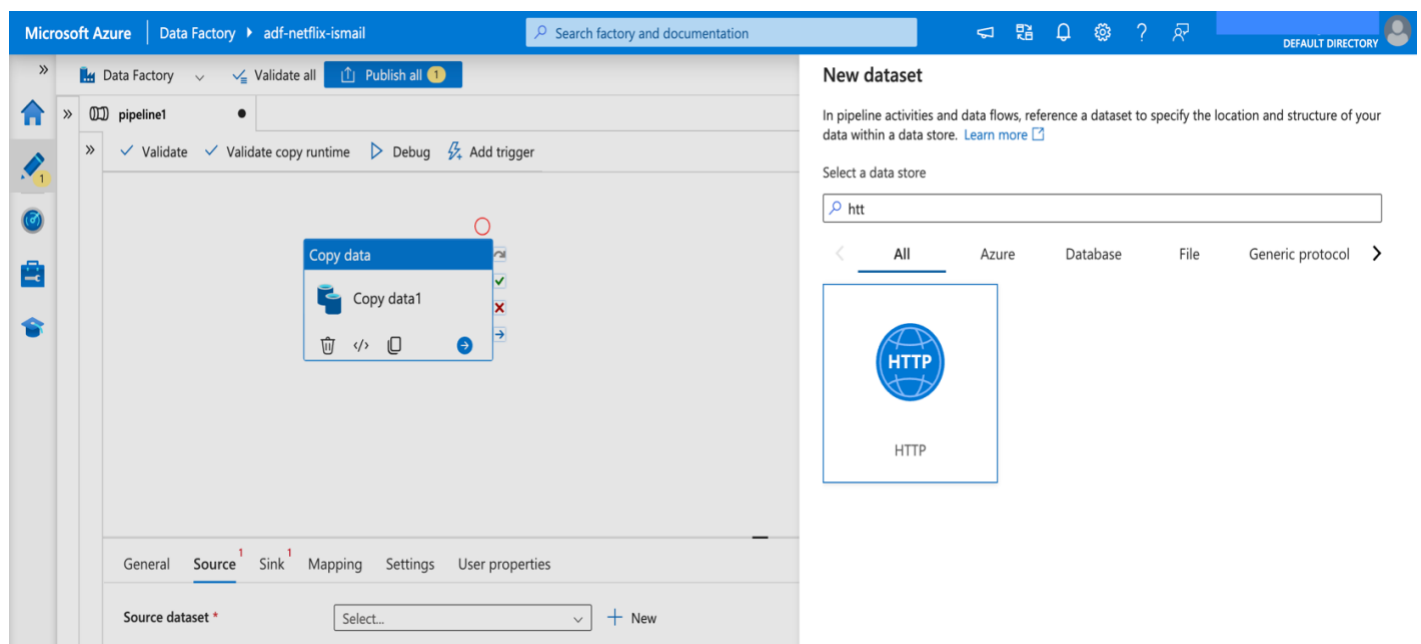
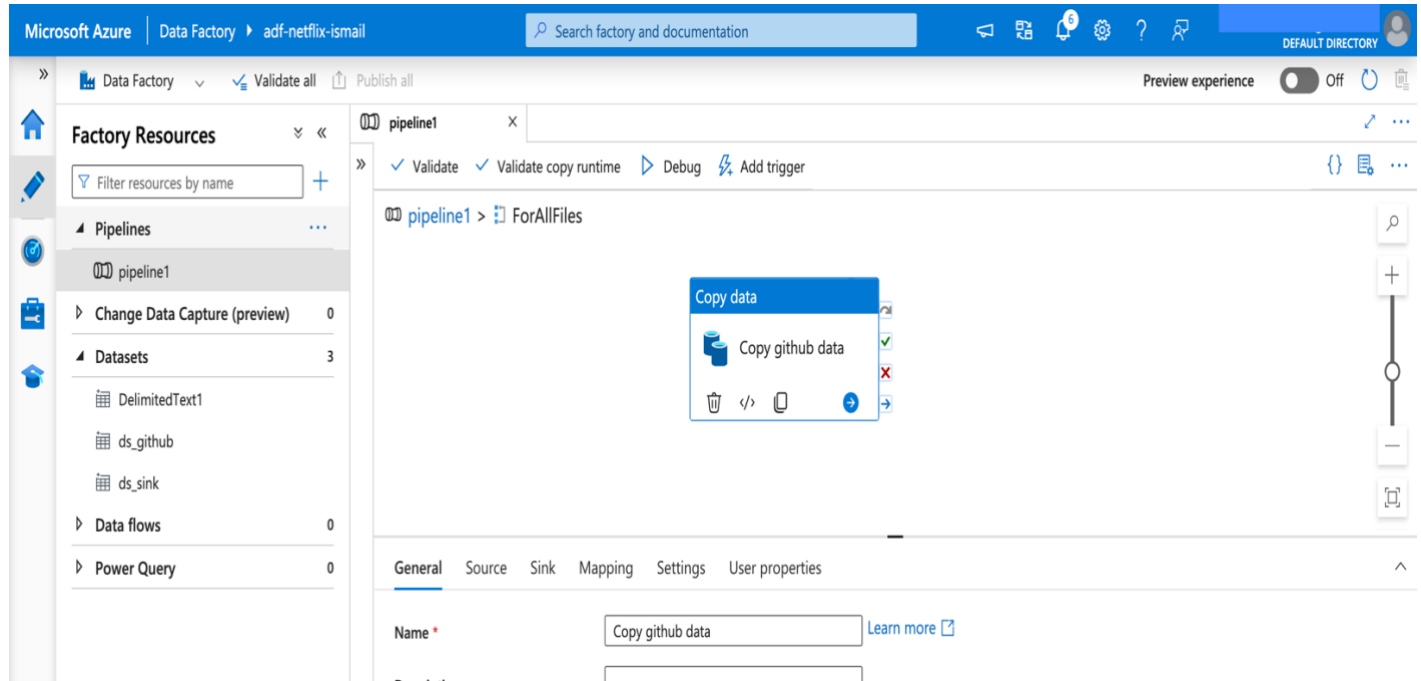


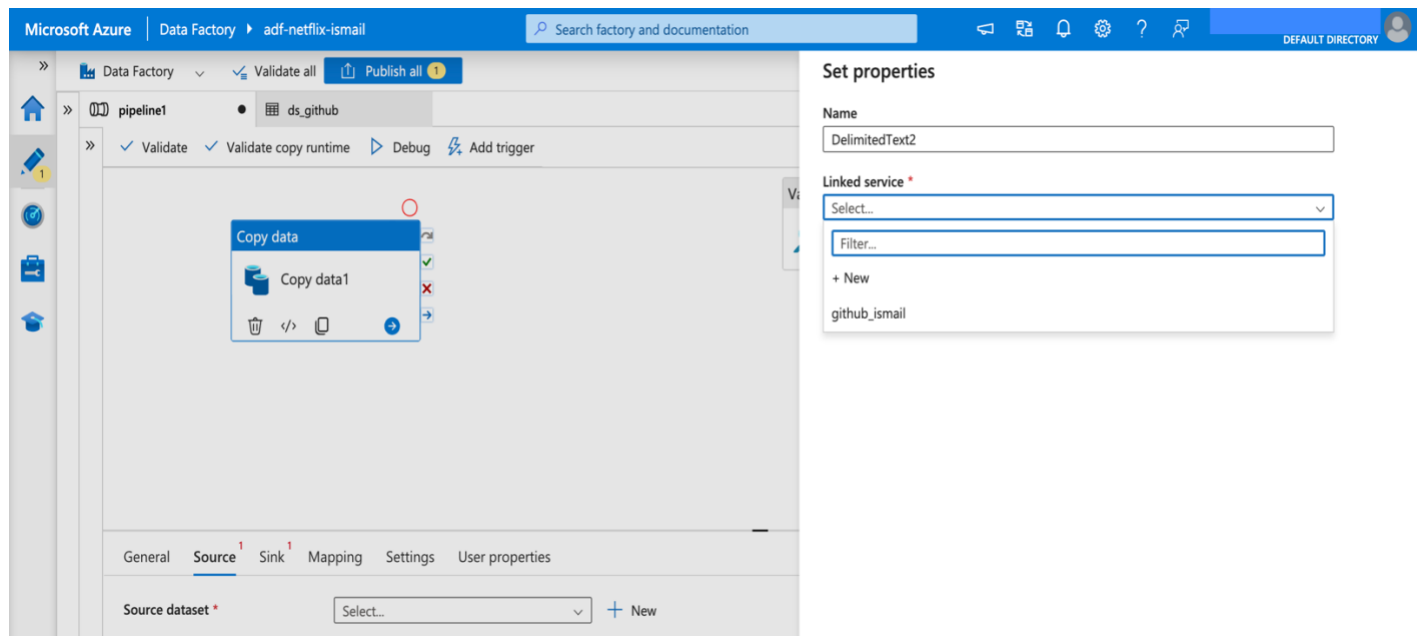
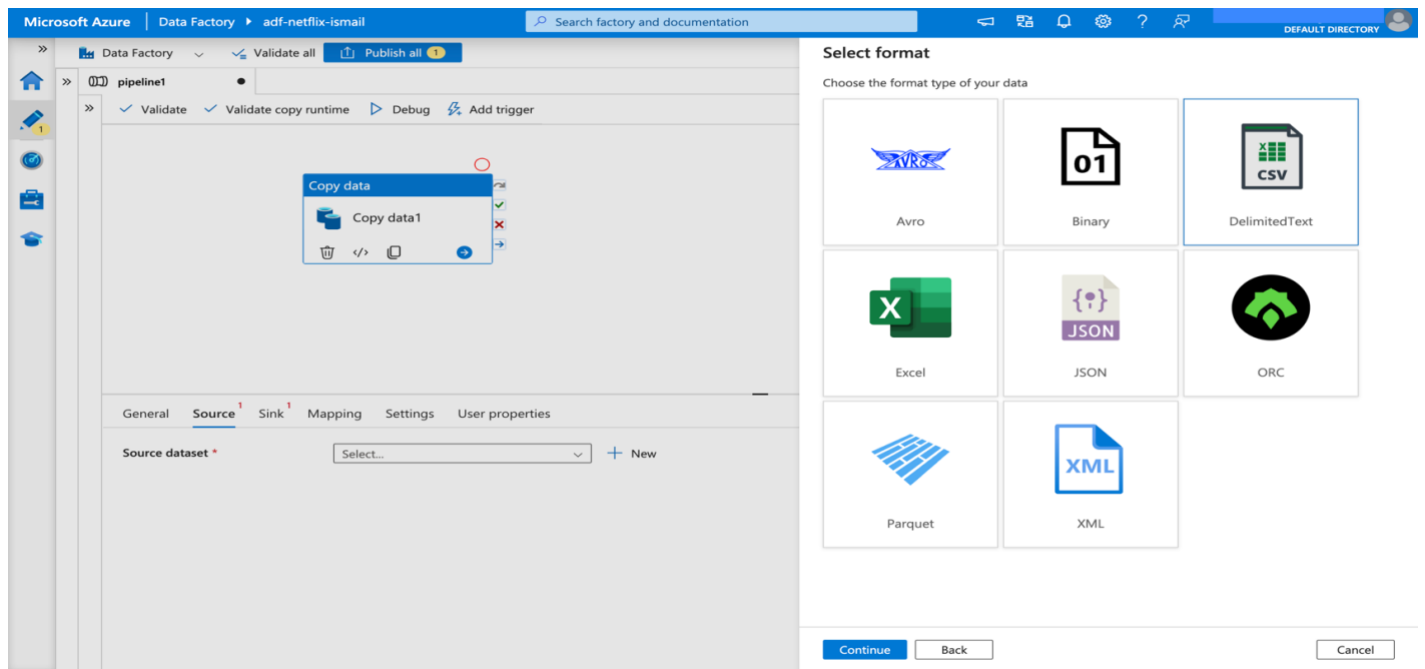
# How I Set Up My Azure Data Ingestion Pipeline

Once the linked services were set up, I navigated to the **Author** section in Azure Data Factory to create a new pipeline. The first activity I added was the **Copy Activity**, which copies files from the GitHub source and writes them to the ADLS Gen2 destination.

## *Creating Source and Sink Datasets for the Copy Activity*

As I began configuring the Copy Activity, the first step was to create new datasets for both the source and the sink. For the source dataset, I selected **HTTP** as the data store and chose **CSV** as the file format, since the GitHub files were in CSV format. I then linked it to GitHub using the HTTP linked service I had previously set up.

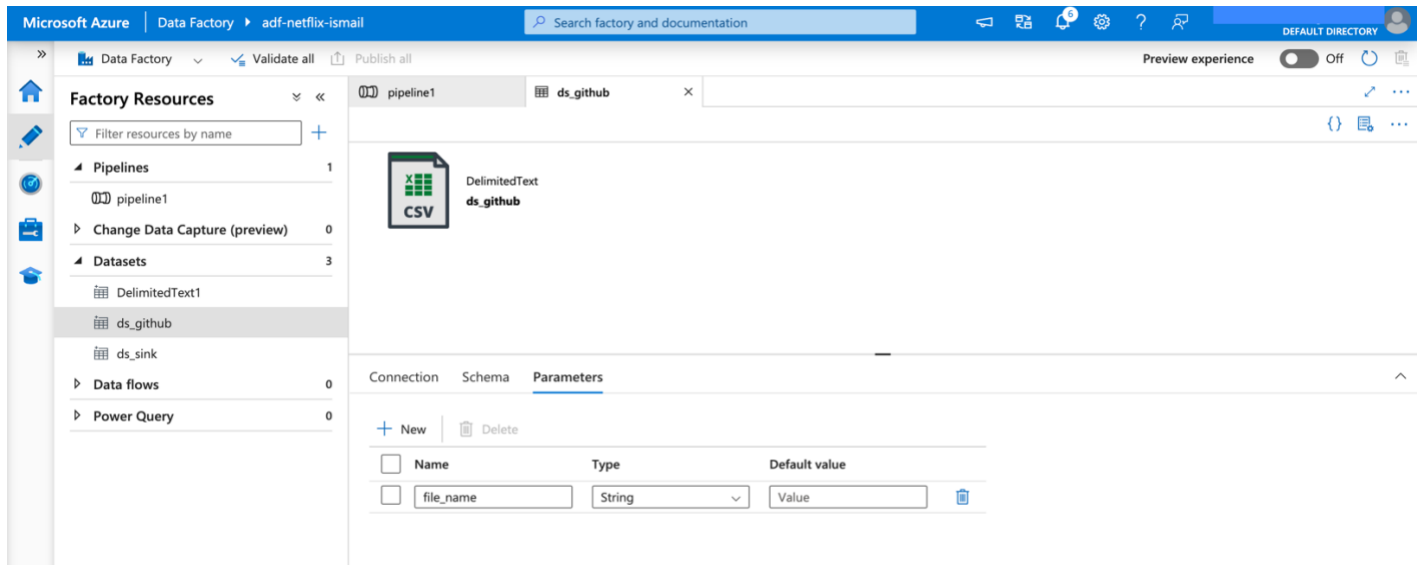




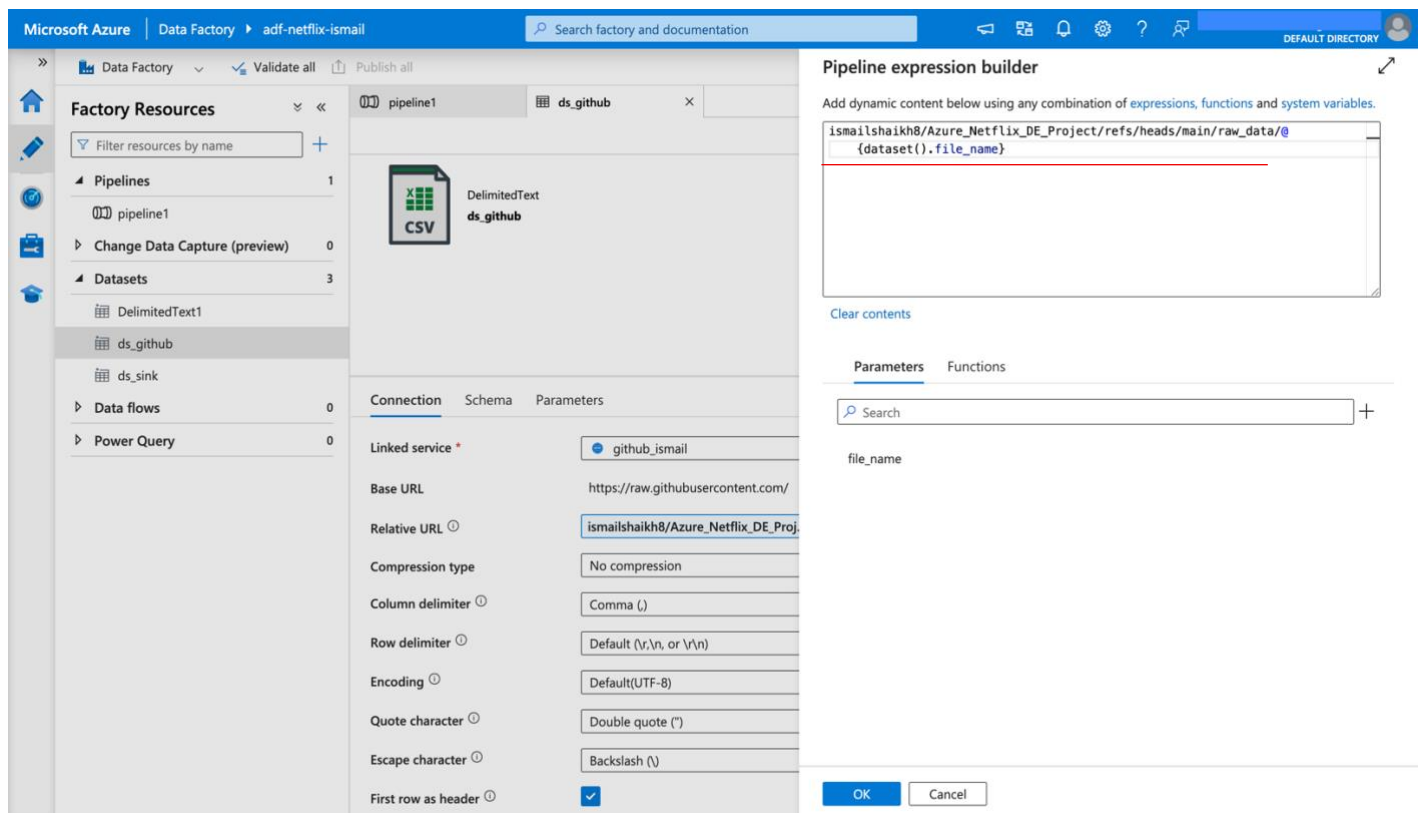
Similarly, for the sink dataset, I chose Azure Data Lake Storage Gen2 as the data store and selected CSV as the file format to match the source. I used the ADLS Gen2 linked service I had already set up to connect to the destination.

## Creating Source Parameter for the Copy Activity

In the Copy Activity source, I set up a parameter called `file_name` because all the files share the same relative URL, with only the file name changing. The relative URL is the part of the path that comes after the base URL. By using the `file_name` parameter, the pipeline can build the full file path dynamically during execution. This allows a single Copy Activity to handle multiple files easily without needing to create separate pipelines or steps for each one.

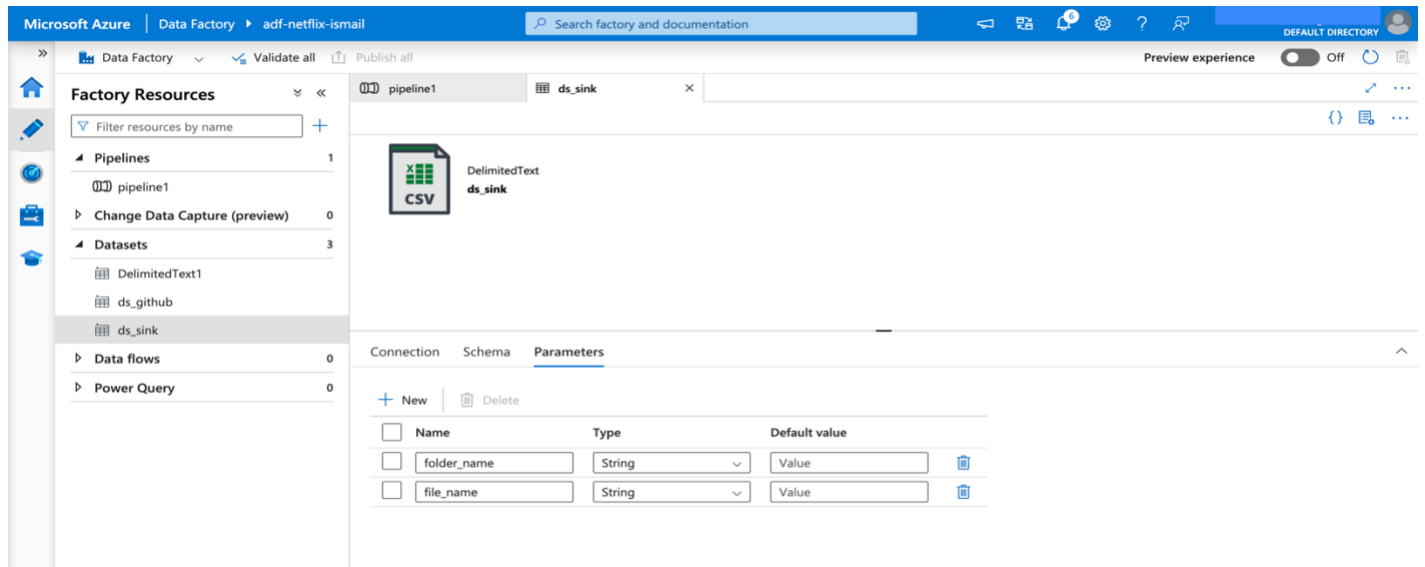


After creating the parameter, I used the pipeline expression builder to set the relative URL. I kept the entire relative URL the same except for the file name, where I used the parameter, I created. This way, the pipeline can dynamically insert different file names during execution.

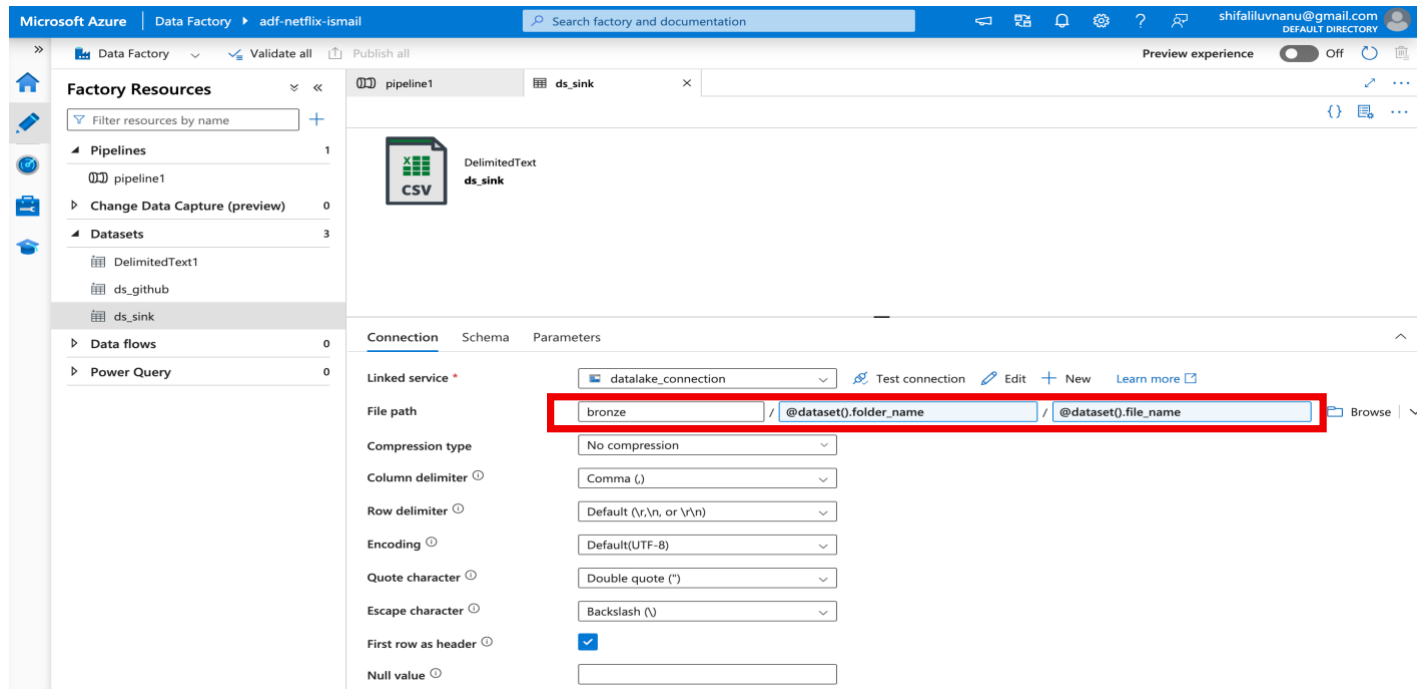


## Creating Sink Parameter for the Copy Activity

Similarly, I created two parameters for the sink: `folder_name` and `file_name`. This setup lets me control exactly where each file should be placed in the storage account, organizing them into specific folders based on their content or source.



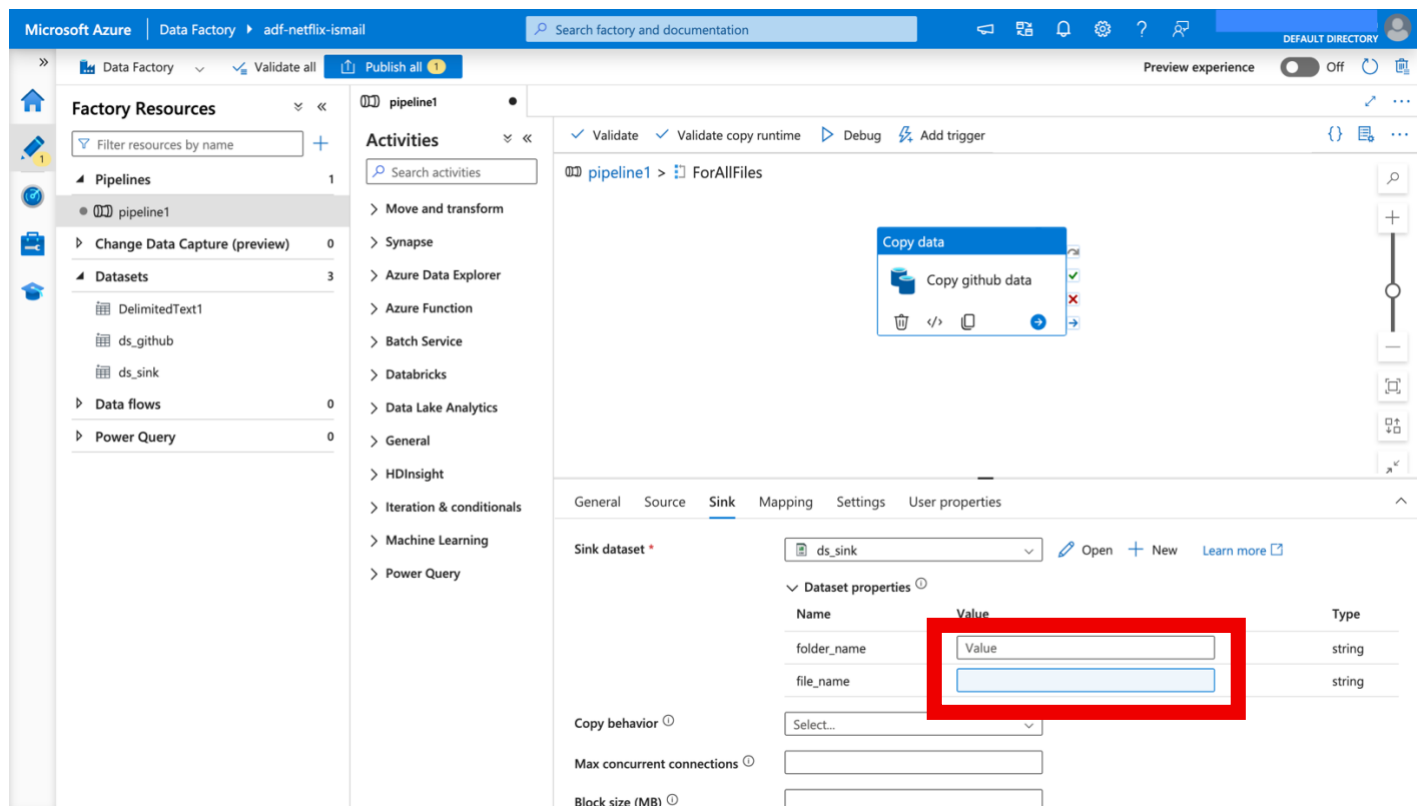
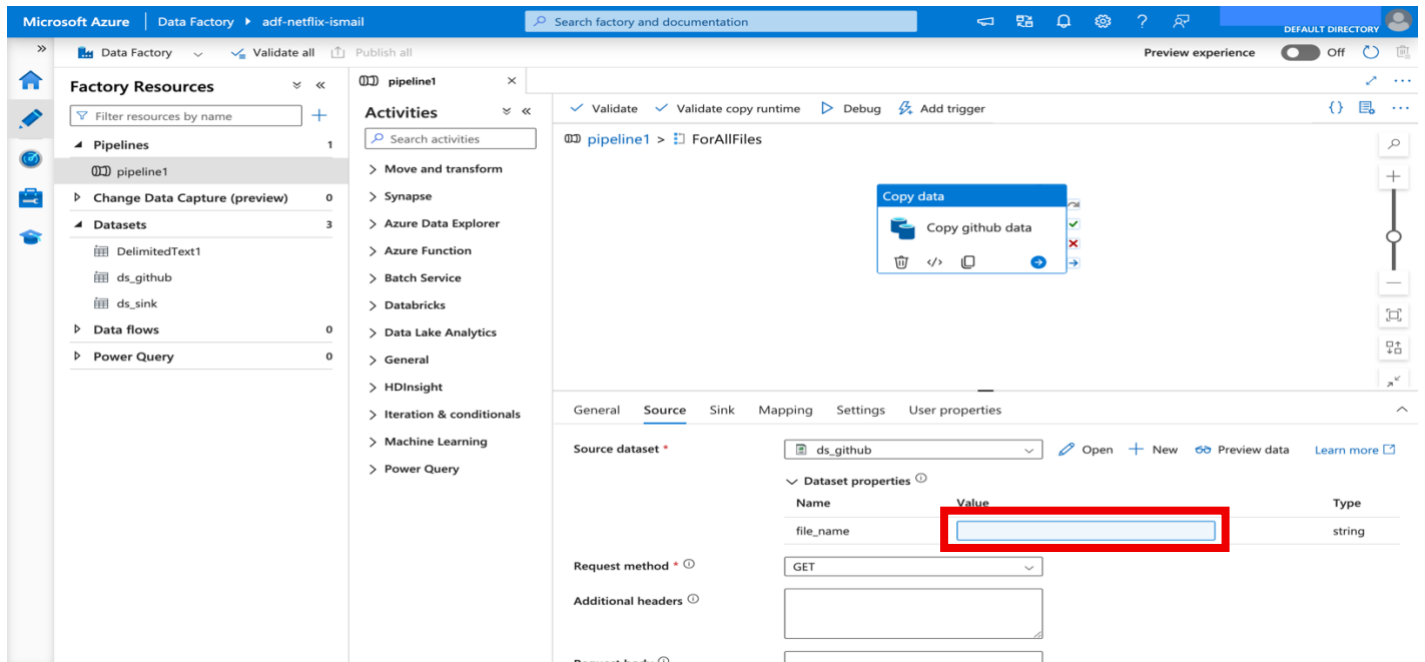
After creating the parameters for the sink, I set the directory path to `bronze` and used the `folder_name` and `file_name` parameters to define exactly where each file should be stored. This helped organize the ingested files into structured folders within the bronze container, making the data easier to manage and work with downstream.





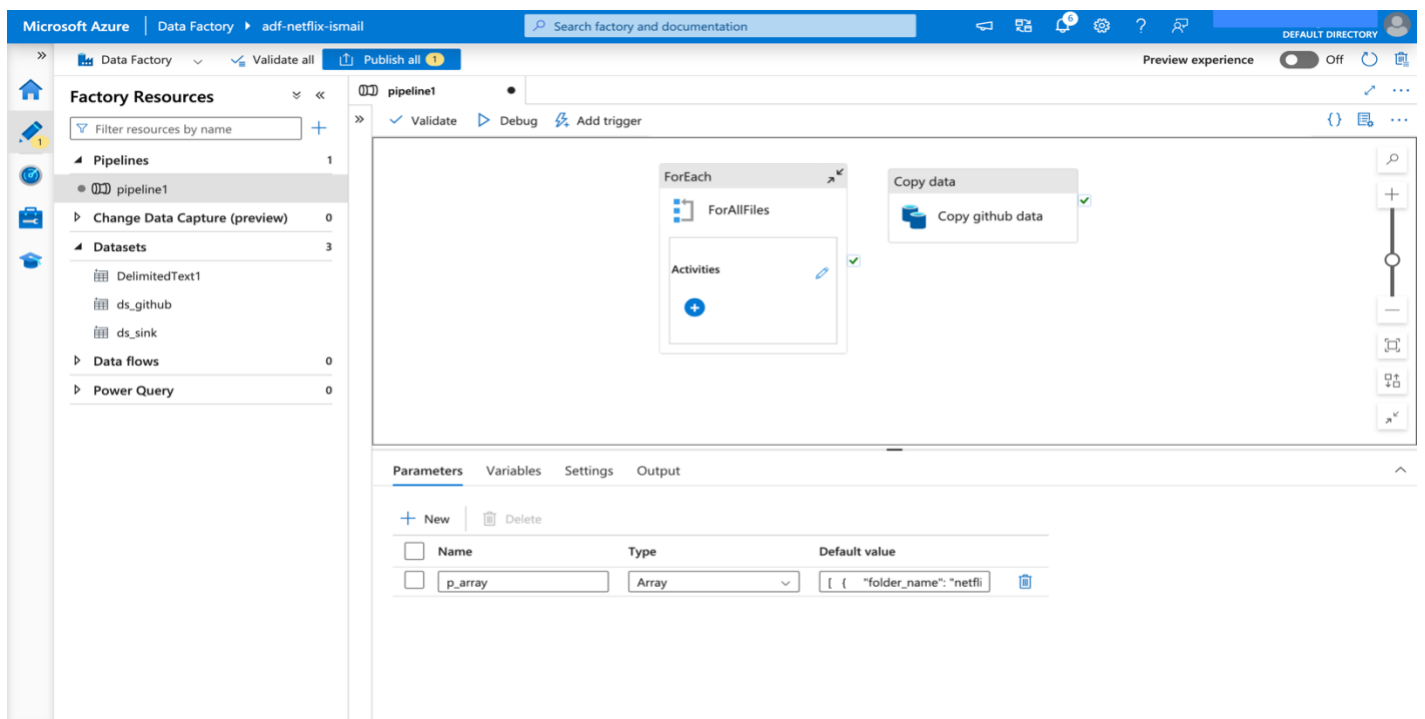
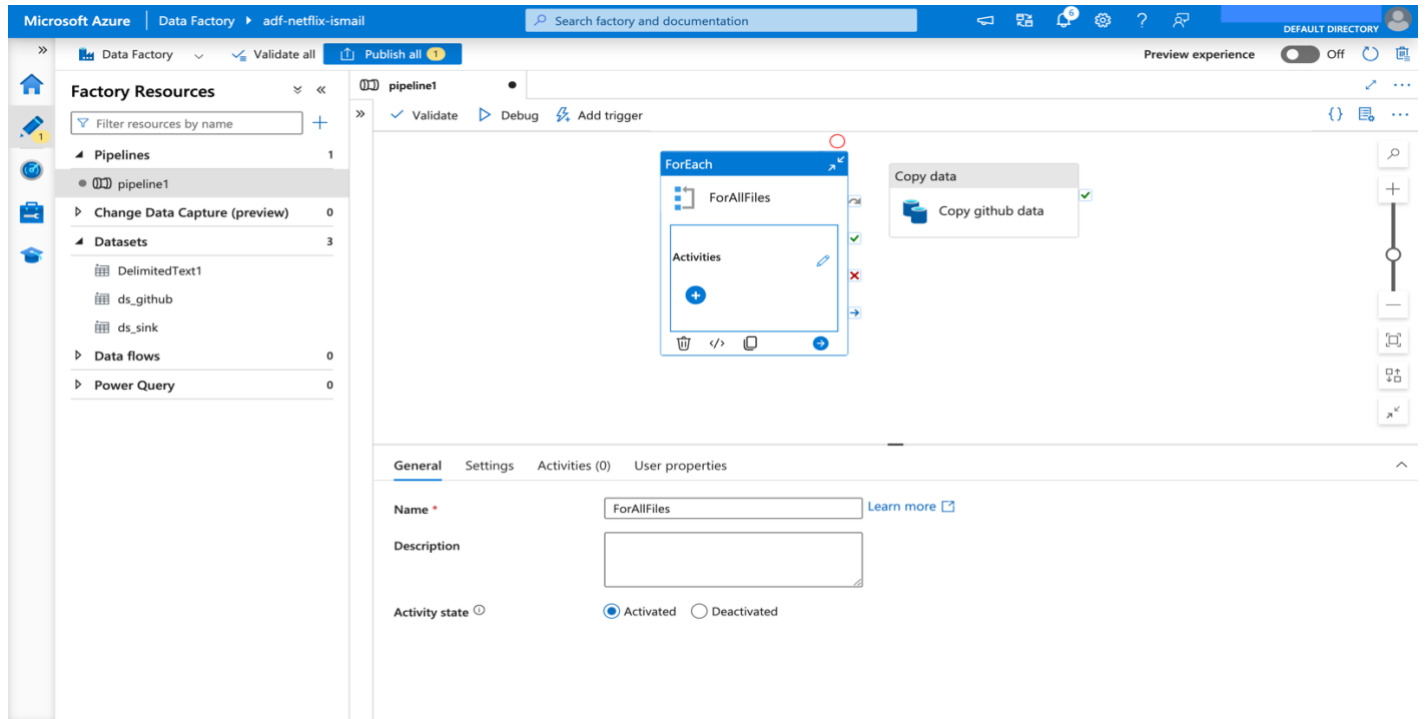
## Parameter Values in Dynamic Pipelines

After setting up the parameters for both the source and sink and referencing them in the relative URL and file path, I needed to provide actual values for each parameter. These values help the pipeline know which file to pull from the source and where exactly to place it in the sink, making the data flow specific to each dataset.



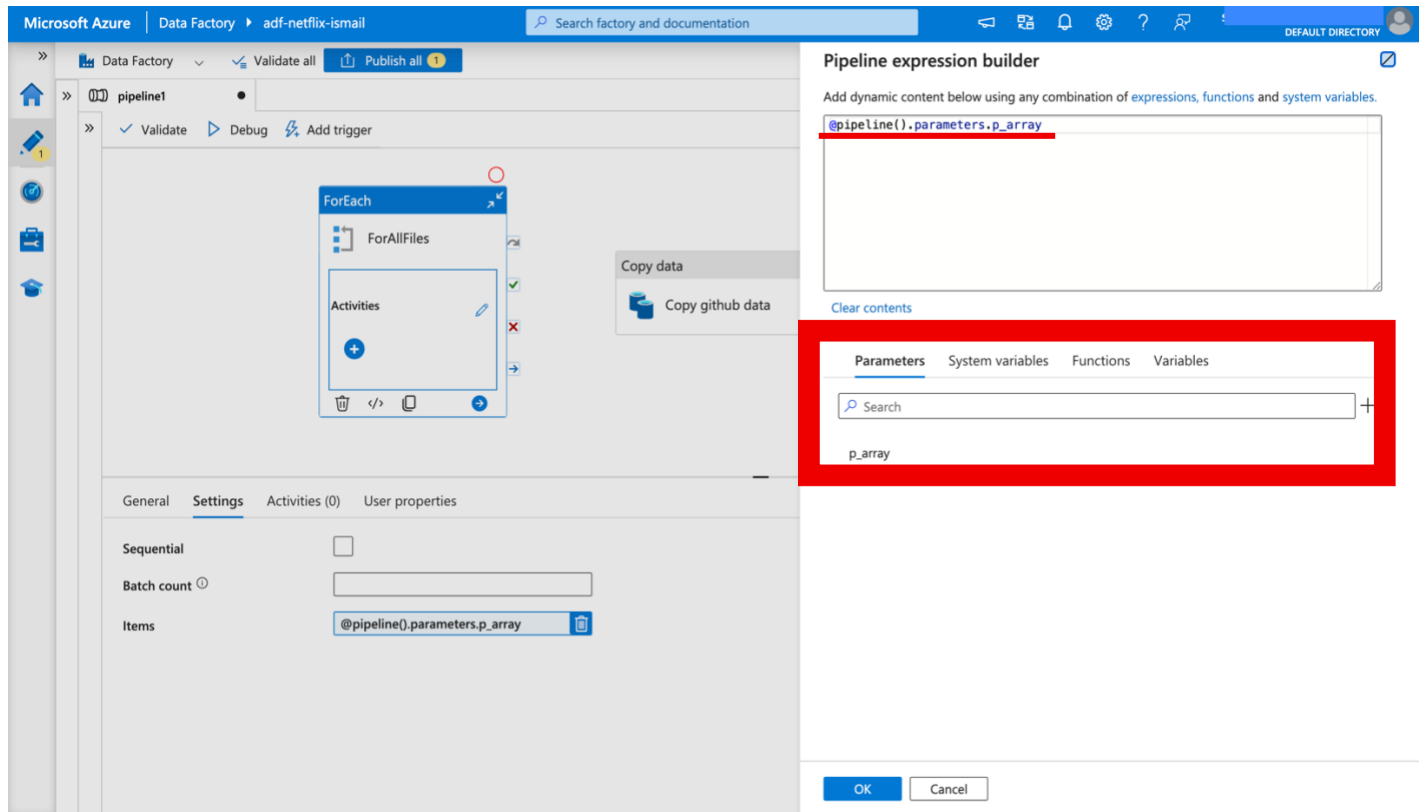
## How I Set Up Parameter Values in Dynamic Pipelines

To make use of the parameters for different files, I used the ForEach activity in Azure Data Factory. The ForEach activity helps repeat a set of steps for every item in a list. In my setup, I created an array parameter that holds values in JSON format. Each entry contains a folder name and a file name. This allowed the pipeline to loop through the list and process each file one by one using the same dynamic structure.



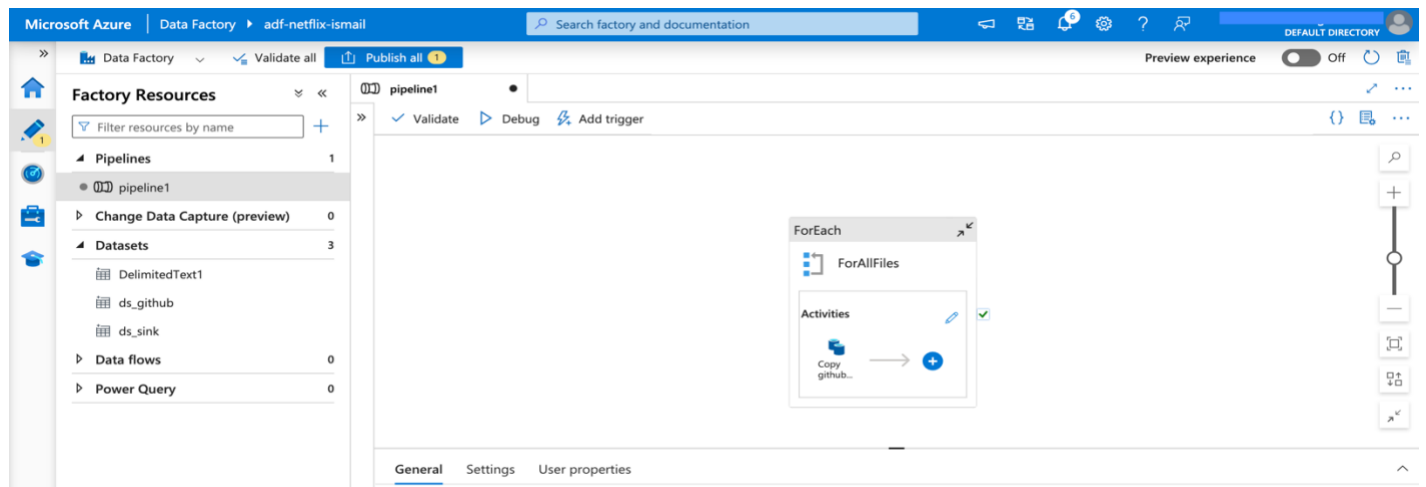
JSON File: [https://github.com/ismailshaikh8/Azure\\_Netflix\\_DE\\_Project/blob/main/Phase%201%20-%20Data%20Ingestion/array.json](https://github.com/ismailshaikh8/Azure_Netflix_DE_Project/blob/main/Phase%201%20-%20Data%20Ingestion/array.json)

Once the array was created, I used it in the settings of the ForEach activity to configure the items it would iterate over. This setup allowed the pipeline to loop through each file and folder combination from the array and process them accordingly.



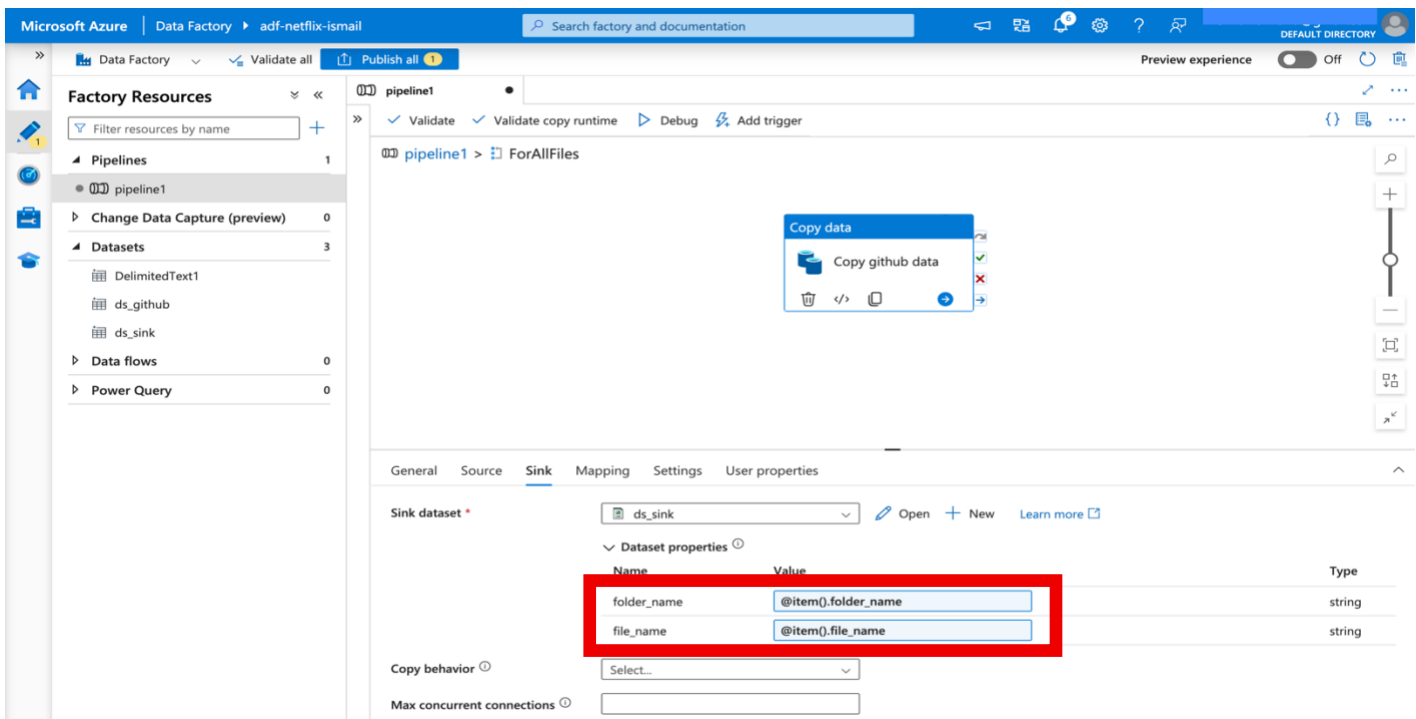
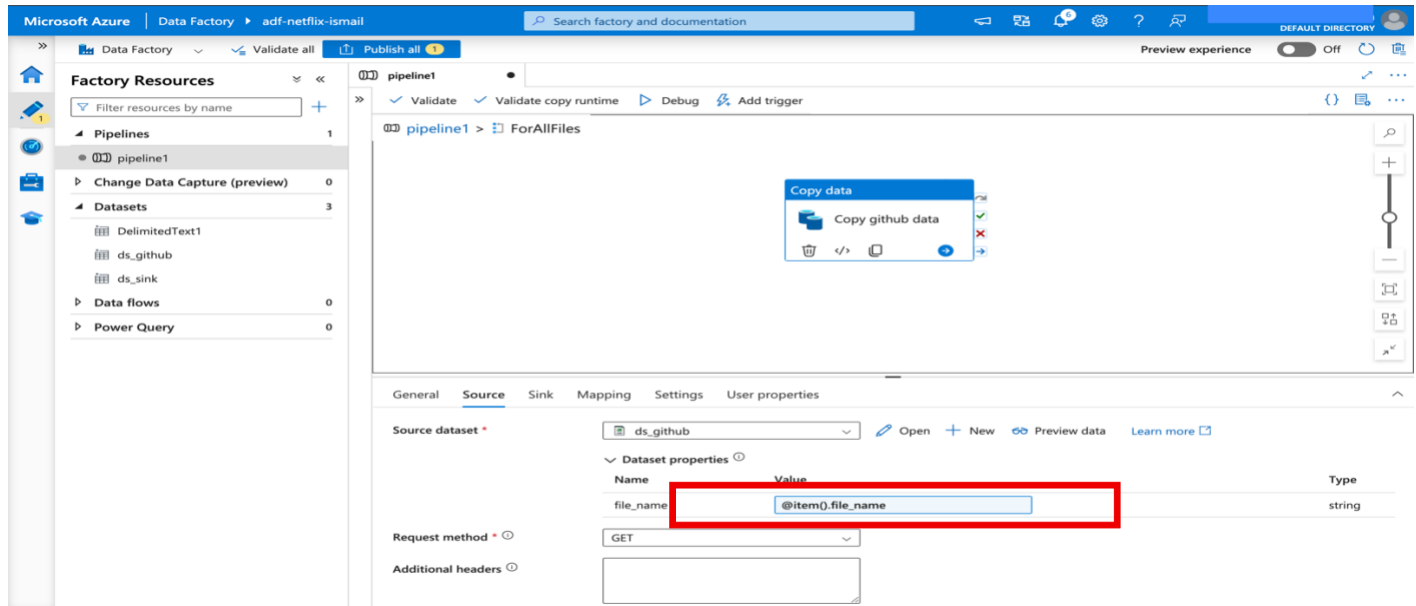
## *Embedding the Copy Activity within the ForEach loop*

After setting up the ForEach activity, I placed the Copy Activity inside it. This way, the loop can run through each item in the array and use the Copy Activity to transfer data from the source to the sink for each file. It ensures that every file is processed one by one based on the parameters defined.



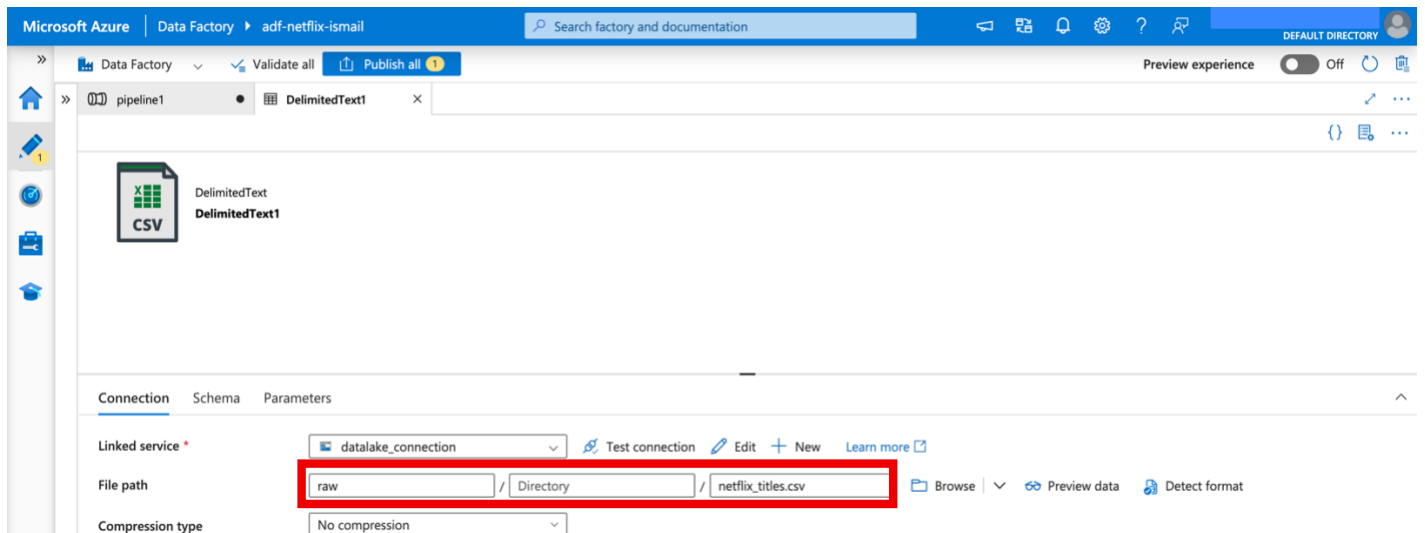
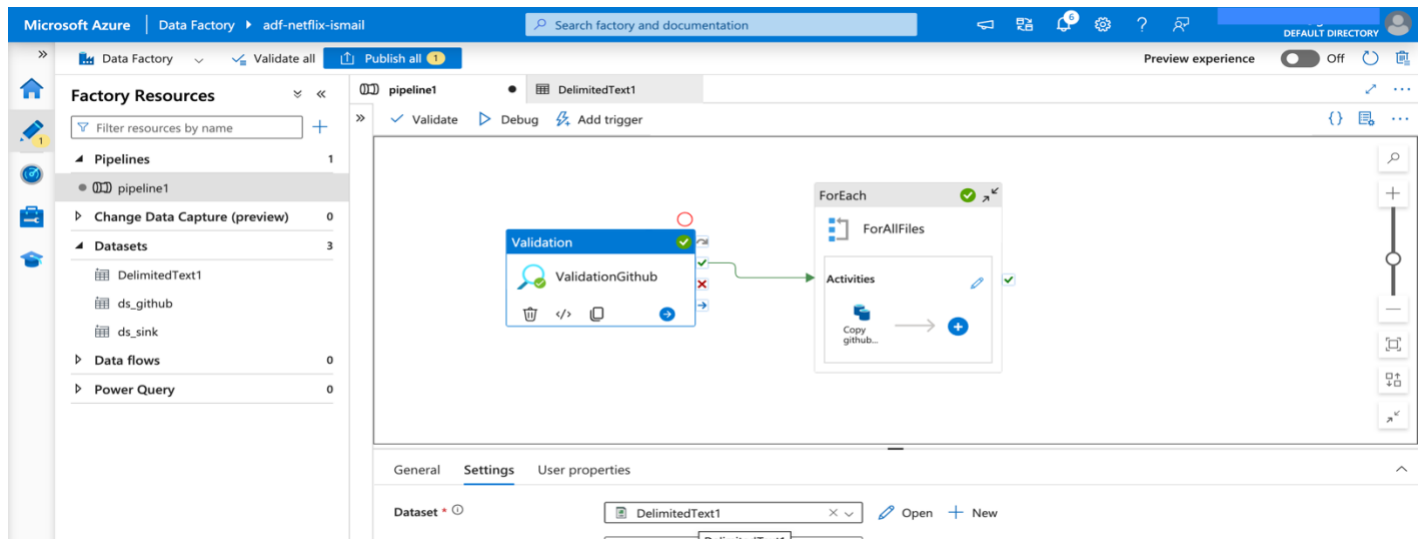
## Passing Dynamic Values to Copy Activity Parameters

After configuring the ForEach activity, I provided the actual values to the parameters in both the source and sink settings of the Copy activity. For the source, I used `@item().file_name`, and for the sink, I used both `@item().folder_name` and `@item().file_name`. This setup ensures that during each iteration, the pipeline dynamically picks the right file from the source and places it into the correct folder in the destination.

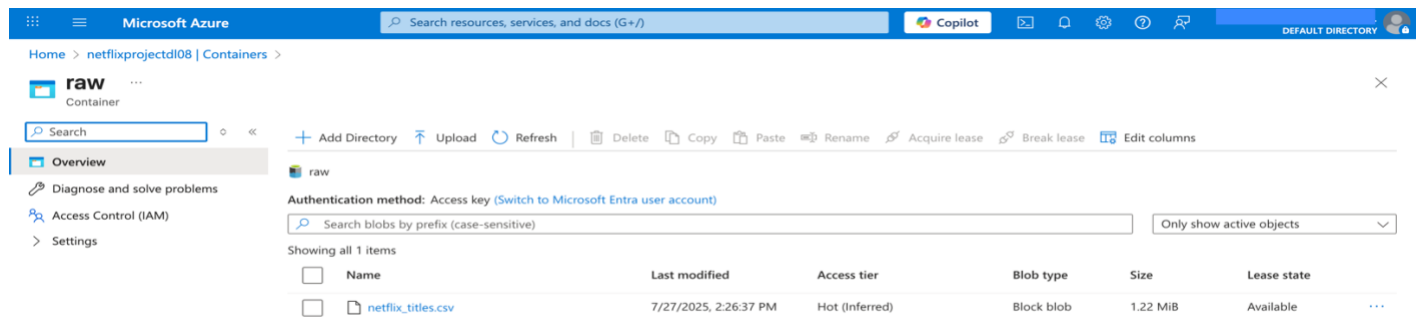


## Using Validation Activity to Control Pipeline Execution

I used the Validation activity in Azure Data Factory to check whether the `netflix_titles.csv` file exists in the raw container. Only if the file is present does the pipeline proceed with the ForEach activity. This ensures that the ingestion process only runs when the expected data is available.



I had already uploaded the `netflix_titles.csv` file into the raw container. To ensure the pipeline doesn't run unnecessarily, I used the Validation activity to confirm the file's presence. Only when the file is found does the pipeline move forward with the ForEach loop to ingest data into the bronze container.



## Running the pipeline

After setting up the Validation, ForEach, and Copy activities, I went ahead and triggered the pipeline. It ran successfully, first checking for the netflix\_titles.csv file in the raw container, and then looping through the files to load them into the bronze container without any issues.

The screenshot displays the Microsoft Azure Data Factory console for a factory named 'adf-netflix-ismail'. The left sidebar shows 'Factory Resources' with a tree view containing 'Pipelines' (1 item), 'Change Data Capture (preview)' (0 items), 'Datasets' (3 items: 'DelimitedText1', 'ds\_github', 'ds\_sink'), 'Data flows' (0 items), and 'Power Query' (0 items). The main canvas shows the 'pipeline1' diagram with a 'Validation' activity (ValidationGithub) followed by a 'ForEach' activity (ForAllFiles) which contains a 'Copy github...' activity. The 'Output' tab at the bottom shows the pipeline run details.

**Pipeline run ID:** b5f7355e-6ecb-45e1-97b3-3886d2a5ed12 **Pipeline status:** Succeeded

Showing 1 - 6 of 6 items

Activity name	Activity status	Activity type	Run start	Duration	Integration runtime	User
Copy github data	Succeeded	Copy data	7/27/2025, 2:32:39 PM	14s	AutoResolveIntegrationRuntime (East US)	
Copy github data	Succeeded	Copy data	7/27/2025, 2:32:39 PM	13s	AutoResolveIntegrationRuntime (East US)	
Copy github data	Succeeded	Copy data	7/27/2025, 2:32:39 PM	12s	AutoResolveIntegrationRuntime (East US)	
Copy github data	Succeeded	Copy data	7/27/2025, 2:32:39 PM	13s	AutoResolveIntegrationRuntime (East US)	
ForAllFiles	Succeeded	ForEach	7/27/2025, 2:32:39 PM	17s		
ValidationGithub	Succeeded	Validation	7/27/2025, 2:32:22 PM	16s		

# Success!

All the files were successfully ingested into the bronze container.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the Microsoft Azure logo, a search bar, and various icons. The breadcrumb trail indicates the path: Home > rg-netflixproject > netflixprojectdl08 | Containers >. The main content area displays the 'bronze' container. On the left, there is a sidebar with 'Overview' selected, and links for 'Diagnose and solve problems', 'Access Control (IAM)', and 'Settings'. The main area shows the container's overview, including an authentication method of 'Access key' and a search bar for blobs. Below this, a table lists four items:

Name	Last modified	Access tier	Blob type	Size	Lease state
netflix_cast	7/27/2025, 2:32:49 PM				...
netflix_category	7/27/2025, 2:32:49 PM				...
netflix_countries	7/27/2025, 2:32:52 PM				...
netflix_directors	7/27/2025, 2:32:49 PM				...

At the bottom left, a note states: 'Add or remove favorites by pressing Ctrl+Shift+F'.

This screenshot shows the same Azure portal interface but with the 'netflix\_cast' blob selected. The breadcrumb trail is updated to: Home > netflixprojectdl08 | Containers > bronze > netflix\_cast. The main content area now displays details for this specific blob. The authentication method remains 'Access key'. Below the search bar, a table shows one item:

Name	Last modified	Access tier	Blob type	Size	Lease state
netflix_cast.csv	7/27/2025, 2:32:50 PM	Hot (Inferred)	Block blob	1.02 MiB	Available

Microsoft Azure

Search resources, services, and docs (G+)

Copilot

DEFAULT DIRECTORY

Home > netflixprojectdl08 | Containers >

bronze

Container

Search

+ Add Directory

Upload

Refresh

Delete

Copy

Paste

Rename

Acquire lease

Break lease

Edit columns

Overview

Diagnose and solve problems

Access Control (IAM)

Settings

bronze > netflix\_category

Authentication method: Access key (Switch to Microsoft Entra user account)

Search blobs by prefix (case-sensitive)

Only show active objects

Showing all 1 items

Name	Last modified	Access tier	Blob type	Size	Lease state
[.]					
netflix_category.csv	7/27/2025, 2:32:50 PM	Hot (Inferred)	Block blob	335.81 KiB	Available

Microsoft Azure

Search resources, services, and docs (G+)

Copilot

DEFAULT DIRECTORY

Home > netflixprojectdl08 | Containers >

bronze

Container

Search

+ Add Directory

Upload

Refresh

Delete

Copy

Paste

Rename

Acquire lease

Break lease

Edit columns

Overview

Diagnose and solve problems

Access Control (IAM)

Settings

bronze > netflix\_countries

Authentication method: Access key (Switch to Microsoft Entra user account)

Search blobs by prefix (case-sensitive)

Only show active objects

Showing all 1 items

Name	Last modified	Access tier	Blob type	Size	Lease state
[.]					
netflix_countries.csv	7/27/2025, 2:32:52 PM	Hot (Inferred)	Block blob	144.04 KiB	Available

Microsoft Azure

Search resources, services, and docs (G+)

Copilot

DEFAULT DIRECTORY

Home > netflixprojectdl08 | Containers >

bronze

Container

Search

+ Add Directory

Upload

Refresh

Delete

Copy

Paste

Rename

Acquire lease

Break lease

Edit columns

Overview

Diagnose and solve problems

Access Control (IAM)

Settings

bronze > netflix\_directors

Authentication method: Access key (Switch to Microsoft Entra user account)

Search blobs by prefix (case-sensitive)

Only show active objects

Showing all 1 items

Name	Last modified	Access tier	Blob type	Size	Lease state
[.]					
netflix_directors.csv	7/27/2025, 2:32:49 PM	Hot (Inferred)	Block blob	115.53 KiB	Available