Part 1

Yapi Donatien Achou

September 22, 2024

1 Optimising a python script to leverage multicore

In this use case, a python script processing millions of records runs on a single thread. The question is, how can we leverage multicore to improve the execution time.

“A thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system. In many cases, a thread is a component of a process.”, [1], as shown in Figure 1

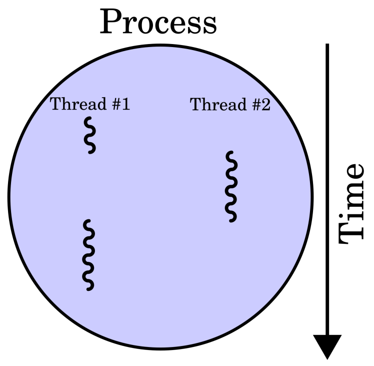


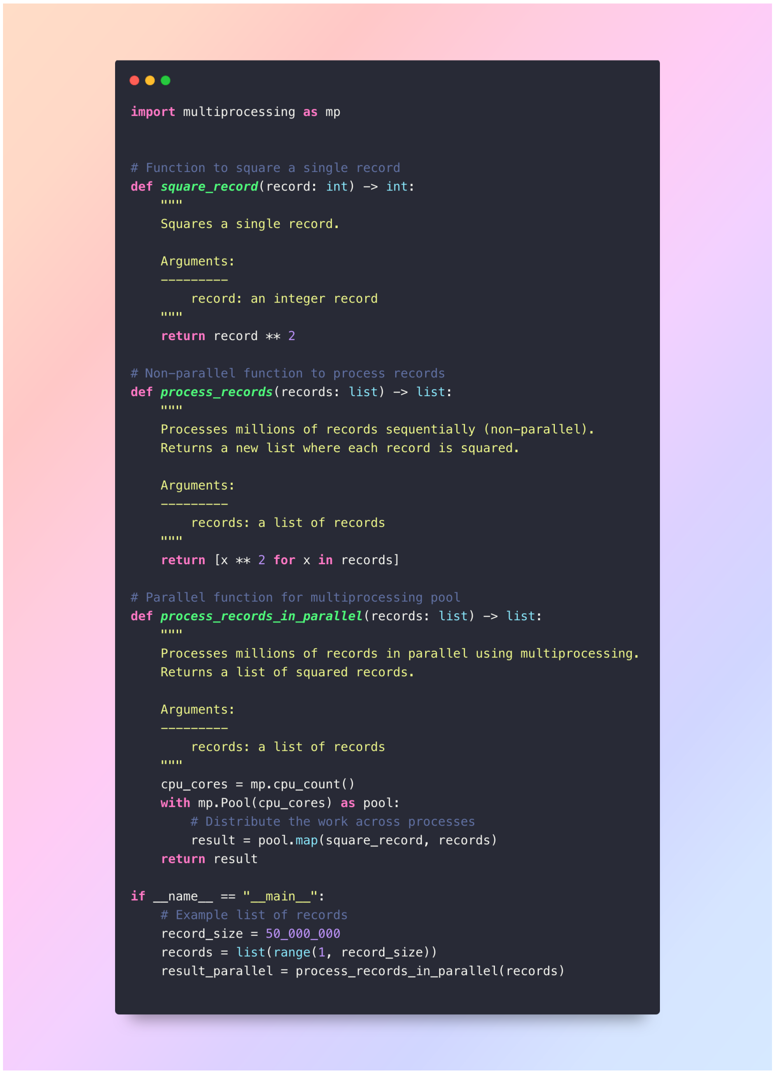
Figure 1: Threads in a process. Curtesy Wikipedia, [1]

Since in this case the script uses only a single thread to process millions of records, it makes sens to distribute the computational load accros multiple threads within one or more processes in order to improve execution time.

As opposed to a thread, a process has its own resource (memory, computational unit) and “each CPU (core) executes a single process at a time”, [2], so to improve execution time we can use multiple processes on a multicore architecture machine. In addition, processing millions of records is a CPU intensive task and processes shine in these king of tasks.

In Python parallelism can be achieved by using the **multiprocessing** module. We can use the **Pool** object in the multiprocessing module to distribute the input records across processes and achieve **data parallelism**, [3]. The **Pool** object creates multiples processes each running independently and leveraging multiples cpu cores.

Figure 2 shows the implementation of a function that uses the multiprocessing python module to achieve data parallelism using the Pool object, while Figure 3 show the different execution time, with different records size.



```python
import multiprocessing as mp


# Function to square a single record
def square_record(record: int) -> int:
    """
    Squares a single record.

    Arguments:
    ---------
        record: an integer record
    """
    return record ** 2

# Non-parallel function to process records
def process_records(records: list) -> list:
    """
    Processes millions of records sequentially (non-parallel).
    Returns a new list where each record is squared.

    Arguments:
    ---------
        records: a list of records
    """
    return [x ** 2 for x in records]

# Parallel function for multiprocessing pool
def process_records_in_parallel(records: list) -> list:
    """
    Processes millions of records in parallel using multiprocessing.
    Returns a list of squared records.

    Arguments:
    ---------
        records: a list of records
    """
    cpu_cores = mp.cpu_count()
    with mp.Pool(cpu_cores) as pool:
        # Distribute the work across processes
        result = pool.map(square_record, records)
    return result

if __name__ == "__main__":
    # Example list of records
    record_size = 50_000_000
    records = list(range(1, record_size))
    result_parallel = process_records_in_parallel(records)
```

Figure 2: Parallelism of a function using multiprocessing module in Python

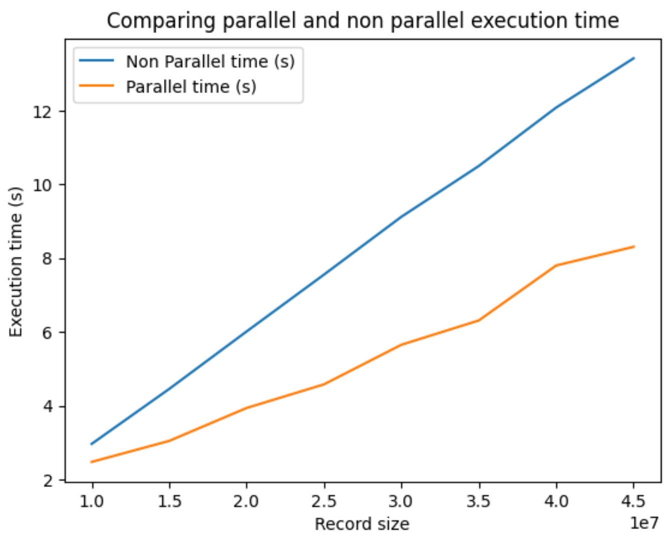


Figure 3: Execution time of a single thread vs a multi process code

2 Handling invalid data format in azure Data Factory

Data format can refer to constraints placed upon the interpretation of data in a type system (data type), a format for encoding data for storage in a computer file (file format), [4].

Inconsistency between data type and file format between data in a data lake source and a dataset in Azure Data Factory can cause an invalid data format error in a pipeline run. For illustration purpose we will be focusing on a file format invalid data format error.

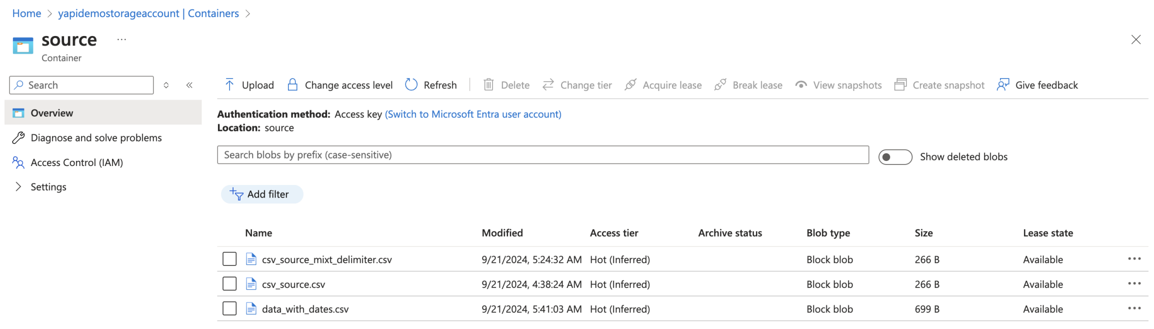
A data pipeline failure due to an invalid data format in Azure Data Factory can occur, for example, when there is a mismatch between the format of the source data (e.g., in an Azure Data Lake container) and the source dataset configuration in Azure Data Factory.

Let's say the source data in the Azure Data Lake container is in CSV format, but when configuring the dataset in Azure Data Factory, we mistakenly set the dataset format to JSON. This mismatch would cause the pipeline to fail, as Azure Data Factory would try to interpret the CSV file as JSON, leading to this case in a JsonInvalidDataFormat error.

I have reproduced this error by:

- Provisioning a storage account and populated a source container with csv files. See Figure 4

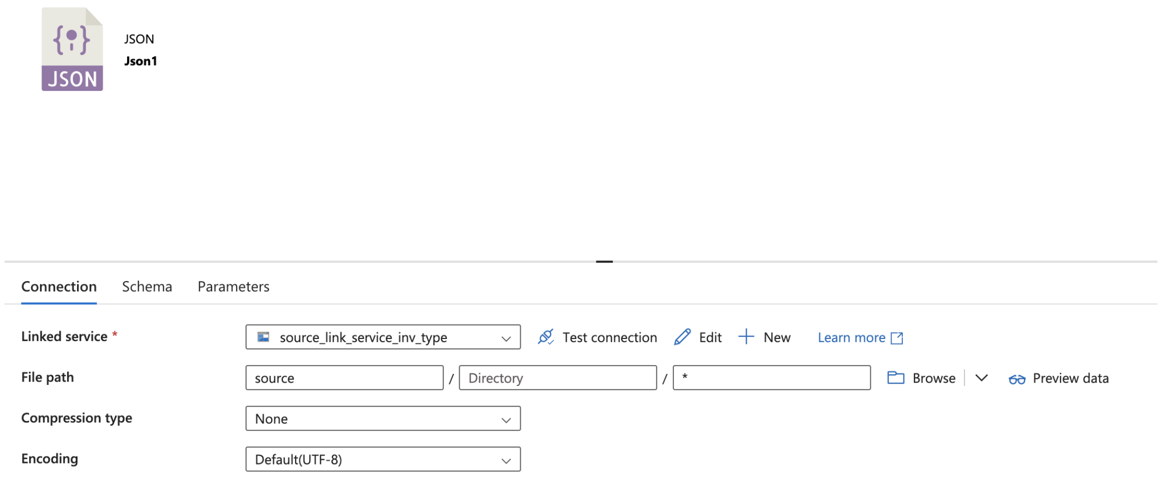
- Provisioning an Azure Data Factory resource
- Creating a source linked service
- Creating a sink linked service
- Creating a JSON source Dataset. See Figure 5
- Creating a CSV sink Dataset. See Figure 6
- Creating a copy pipeline from source to sink. See Figure 7
- running the pipeline to reproduce the error



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

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
csv_source_mixt_delimiter.csv	9/21/2024, 5:24:32 AM	Hot (Inferred)		Block blob	266 B	Available
csv_source.csv	9/21/2024, 4:38:24 AM	Hot (Inferred)		Block blob	266 B	Available
data_with_dates.csv	9/21/2024, 5:41:03 AM	Hot (Inferred)		Block blob	699 B	Available

Figure 4: Azure source container with CSV files



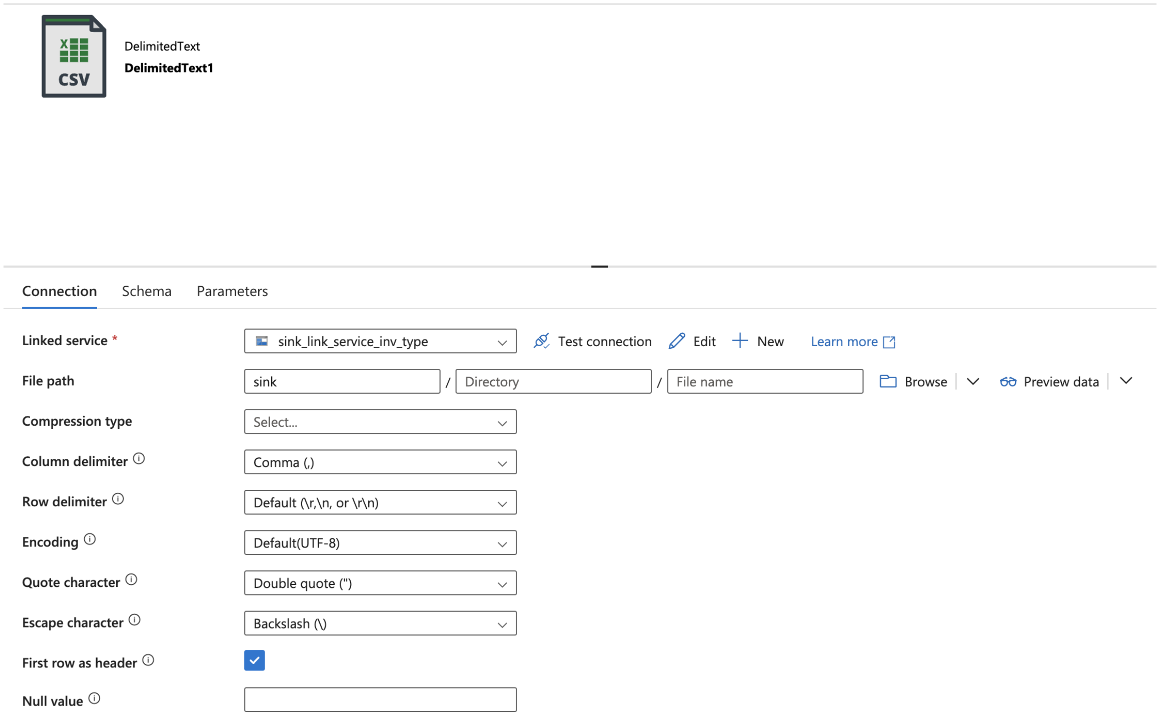
Linked service *: source_link_service_inv_type

File path: source / Directory / *

Compression type: None

Encoding: Default(UTF-8)

Figure 5: Azure Data Factory source JSON Dataset

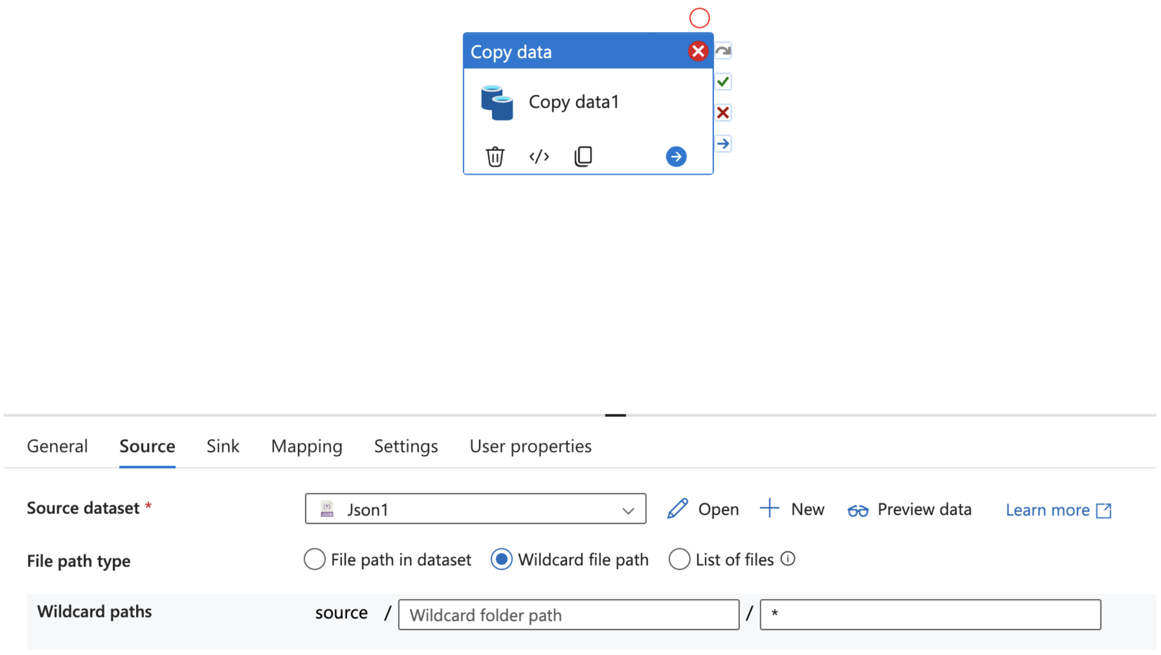


DelimitedText
DelimitedText1

Connection | Schema | Parameters

Linked service *	sink_link_service_inv_type
File path	sink / Directory / File name
Compression type	Select...
Column delimiter	Comma (,)
Row delimiter	Default (\r,\n, or \r\n)
Encoding	Default(UTF-8)
Quote character	Double quote (")
Escape character	Backslash (\)
First row as header	✓
Null value	

Figure 6: Azure Data Factory sink CSV Dataset



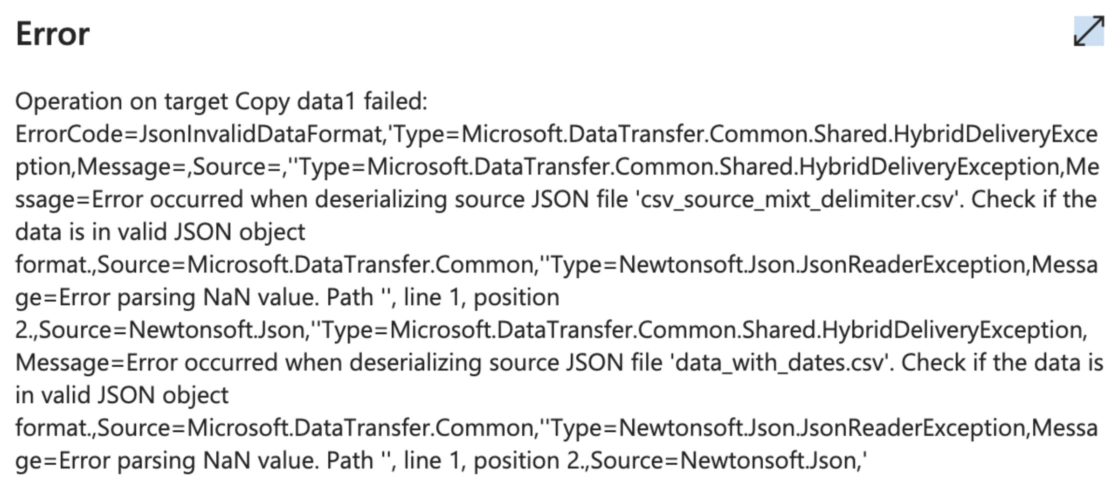
Source dataset * Json1

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

Wildcard paths: source / Wildcard folder path / *

Figure 7: Azure Data Factory Copy pipeline configuration

The pipeline failed with a JsonInvalidDataFormat as see in figure 8



Error

```
Operation on target Copy data1 failed:
ErrorCode=JsonInvalidDataFormat,'Type=Microsoft.DataTransfer.Common.Shared.HybridDeliveryExce
ption,Message=,Source=,''Type=Microsoft.DataTransfer.Common.Shared.HybridDeliveryException,Me
ssage=Error occurred when deserializing source JSON file 'csv_source_mixt_delimiter.csv'. Check if the
data is in valid JSON object
format.,Source=Microsoft.DataTransfer.Common,''Type=Newtonsoft.Json.JsonReaderException,Messa
ge=Error parsing NaN value. Path '', line 1, position
2.,Source=Newtonsoft.Json,''Type=Microsoft.DataTransfer.Common.Shared.HybridDeliveryException,
Message=Error occurred when deserializing source JSON file 'data_with_dates.csv'. Check if the data is
in valid JSON object
format.,Source=Microsoft.DataTransfer.Common,''Type=Newtonsoft.Json.JsonReaderException,Messa
ge=Error parsing NaN value. Path '', line 1, position 2.,Source=Newtonsoft.Json,'
```

Figure 8: JsonInvalidDataFormat error

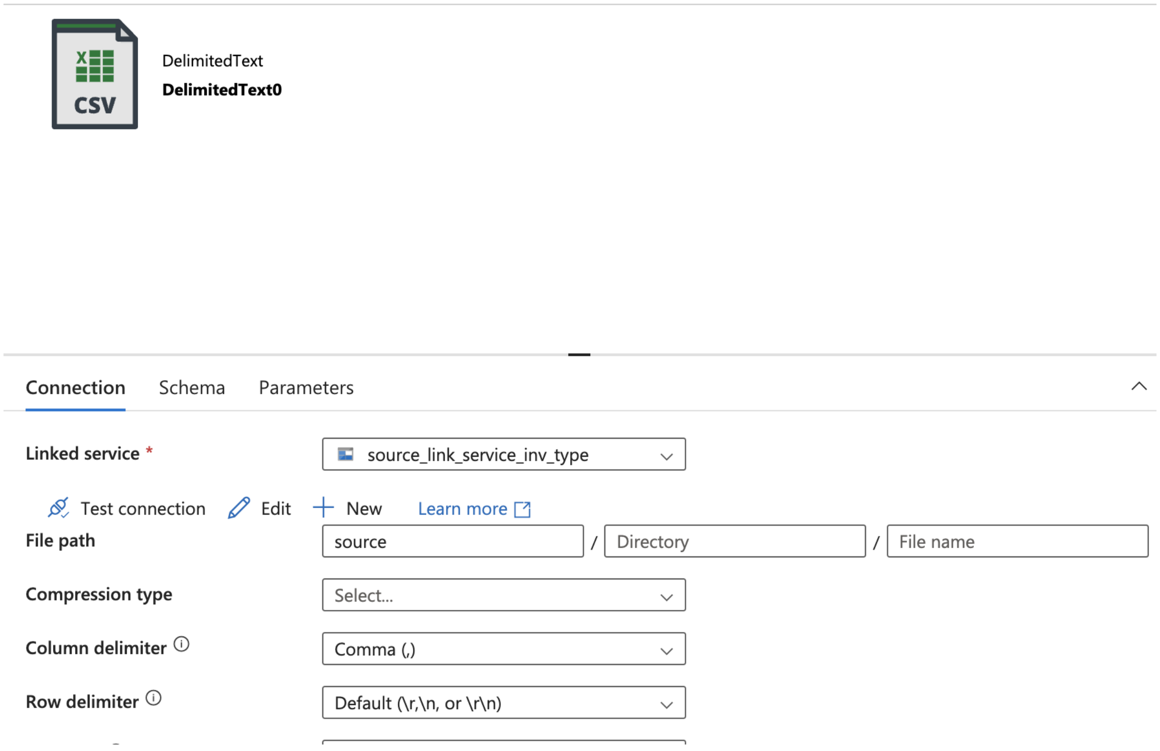
2.1 Debugging such error in Azure Data Factory

The First approach in debugging this issue is by

1. Carefully reading the error message for clues in finding the cause
2. Verifying that the dataset in Azure Data Factory configurations (data format) matches the format of the source data (files located in Azure data lake container for example)

In this case the error message clearly says that there is a difference between the source files format in the Azure data lake container and the configured Dataset format in Azure Data Factory. To fix the issue:

1. Create a new Dataset that matches the format of the files in the source Azure container in this case CSV. See Figure 9
2. Update the pipeline configuration to take the new Dataset as source. See Figure 10



Linked service * source_link_service_inv_type

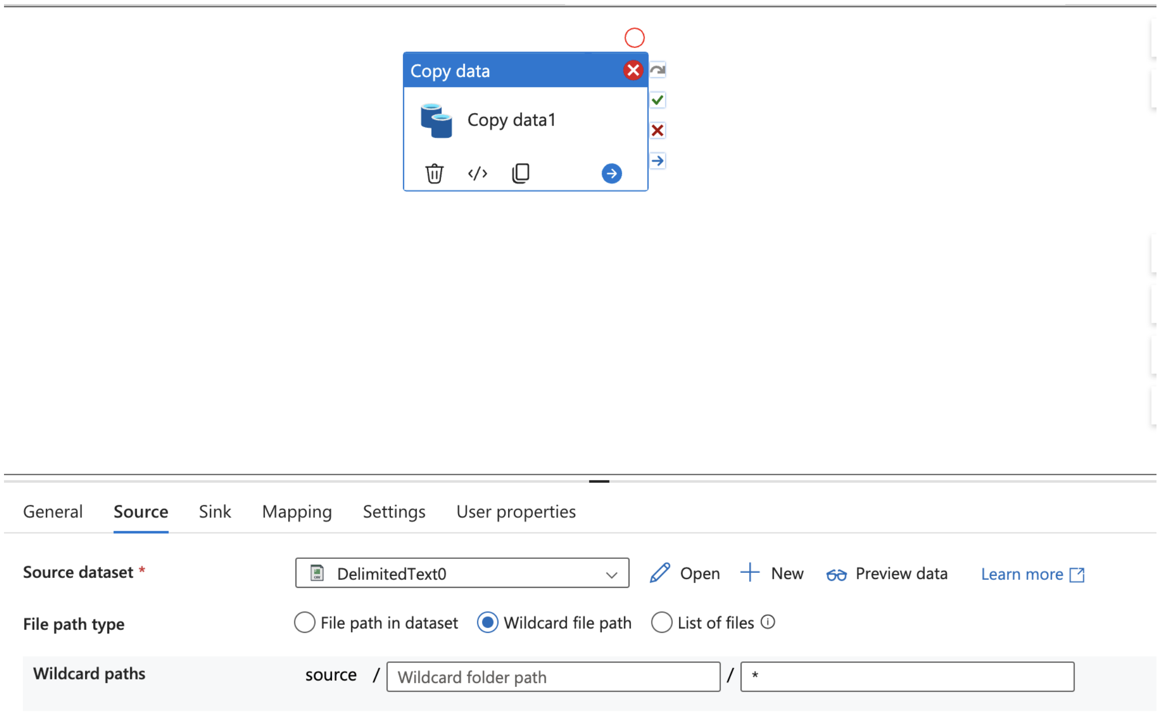
File path source / Directory / File name

Compression type Select...

Column delimiter Comma (,)

Row delimiter Default (\r,\n, or \r\n)

Figure 9: Azure Data Factory source CSV Dataset



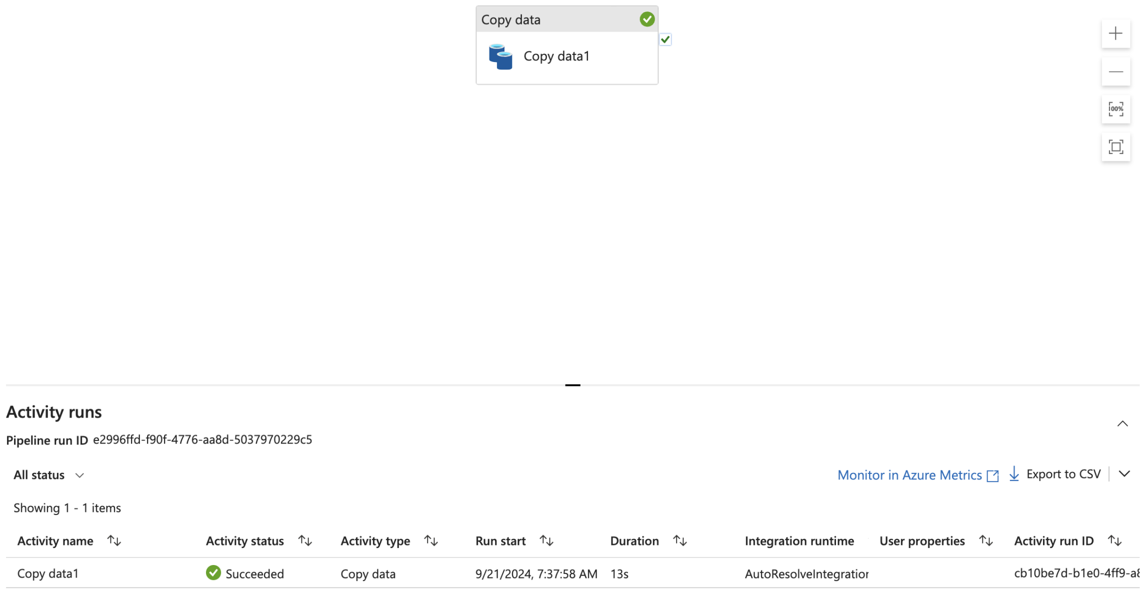
Source dataset * DelimitedText0

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

Wildcard paths: source / Wildcard folder path / *

Figure 10: Azure Data Factory updated pipeline

After taking these measures the pipeline runs successfully as seen in Figure 11



Activity runs

Pipeline run ID e2996ffd-f90f-4776-aa8d-5037970229c5

Showing 1 - 1 items

Activity name	Activity status	Activity type	Run start	Duration	Integration runtime	User properties	Activity run ID
Copy data1	✓ Succeeded	Copy data	9/21/2024, 7:37:58 AM	13s	AutoResolveIntegratior		cb10be7d-b1e0-4ff9-a{

Figure 11: Azure Data Factory successful data copy

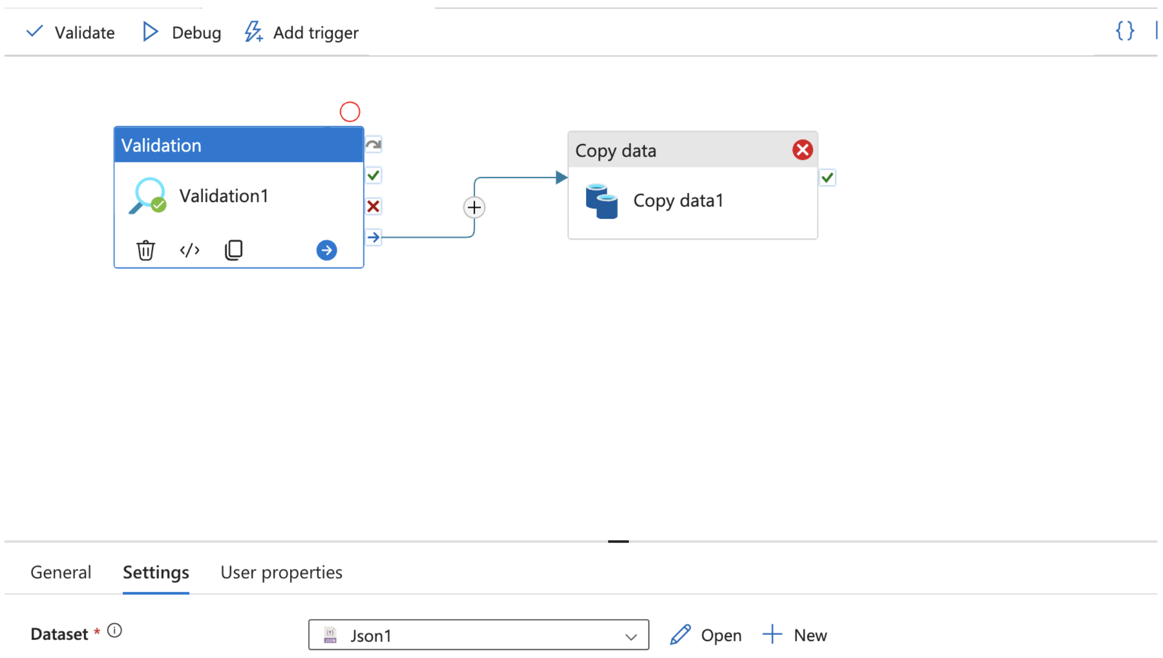
2.2 Making sure it can be avoided in the future

In general, it is a good idea to validate [5], and test your data pipeline before deployment. Validation can be done by setting up data quality contraints during pipeline development.

Testing can be achieved by using debugging tools available in Azure Data Factory, [6]

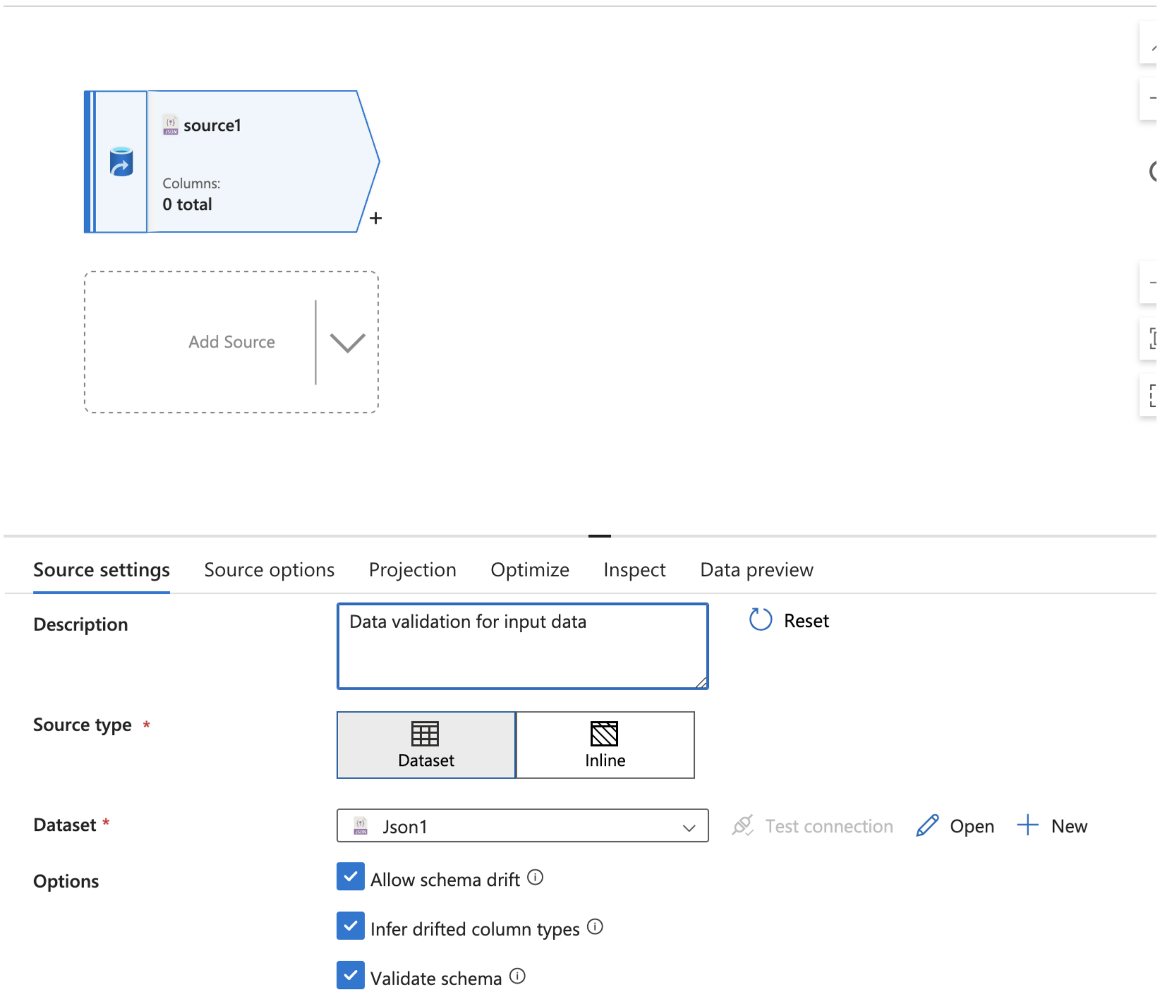
In this particular case proper approaches to avoid data format error in the future:

- Debugging the pipeline by pressing the Debug button in the pipeline creation menue. This will reveal any issue before running the pipeline
- Add a validation activity in the pipeline by validating the source dataset before the pipeline is run [5]. This will ensure that the pipeline only runs after the validation is passed. See Figure 12
- Use Azure Data factory Dataflow [7], for a more detailed and thorough data validation. This will allow for data type validation, schema validation, and more in depth data format validation. See Figure 13



Dataset * Json1

Figure 12: Validation activity attached to the source dataset in a copy pipeline



Description: Data validation for input data

Source type *: [✓ Dataset] Inline

Dataset *: Json1

Options:
- ✓ Allow schema drift
- ✓ Infer drifted column types
- ✓ Validate schema

Figure 13: Using data flow for a more in depth data validation

References

[1] Wikipedia, Threads https://en.wikipedia.org/wiki/Thread_(computing)

[2] Wikipedia, Processes https://en.wikipedia.org/wiki/Process_(computing)

[3] Python package Multiprocessing documentation, https://docs.python.org/3/library/multiprocessing.html

[4] Wikipedia, https://en.wikipedia.org/wiki/Data_format

[5] Microsoft Azure Documentation, https://learn.microsoft.com/en-us/azure/data-factory/control-flow-validation-activity

[6] Microsoft Azure Documentation, https://learn.microsoft.com/en-us/azure/data-factory/iterative-development-debugging?tabs=data-factory

[7] Microsoft Azure Documentation, https://learn.microsoft.com/en-us/azure/data-factory/control-flow-execute-data-flow-activity