

Fabric Lakehouse Loading using Data Pipelines & Notebooks – Inspired by MS End-to-End Tutorials

I joke about “I’m not much of a Spark Developer” all the time and I’m not joking, really I’m not. I’m a SQL developer. I’ve built Data Warehouses using SQL. I’ve cleansed data using SQL, transformed data using SQL, delivered data using SQL. But...I do see the value in understanding more of what Spark and Pyspark has to offer. The more I see just a few lines of Python doing quite a lot of work, the more interested I get.

Building **repeatable iterative data processing frameworks in just a few lines of code**? Yes please!

Microsoft has created a set of [End-to-End tutorials](#) for various workloads including Lakehouses. These are step-by-step walkthroughs to get you up and running with various workloads available within Fabric. Yes you can approach these as stand-alone guides, but you can also use them as a springboard into creating your own solutions. And that’s exactly what I did.

In this blog I’m going to step through what I did to modify the example end-to-end solution to meet my own objectives.

Microsoft’s End-to-End Scenarios can serve as a building block to your own solutions

Part of the **Lakehouse End-to-End** scenario contains a Spark Notebook that iterates over a list of source folders that contains Lakehouse file data, and creates/overwrites a Delta table in a Lakehouse for each folder. What I want to do is use this as a starting point and modify it to meet my own specific use-case.

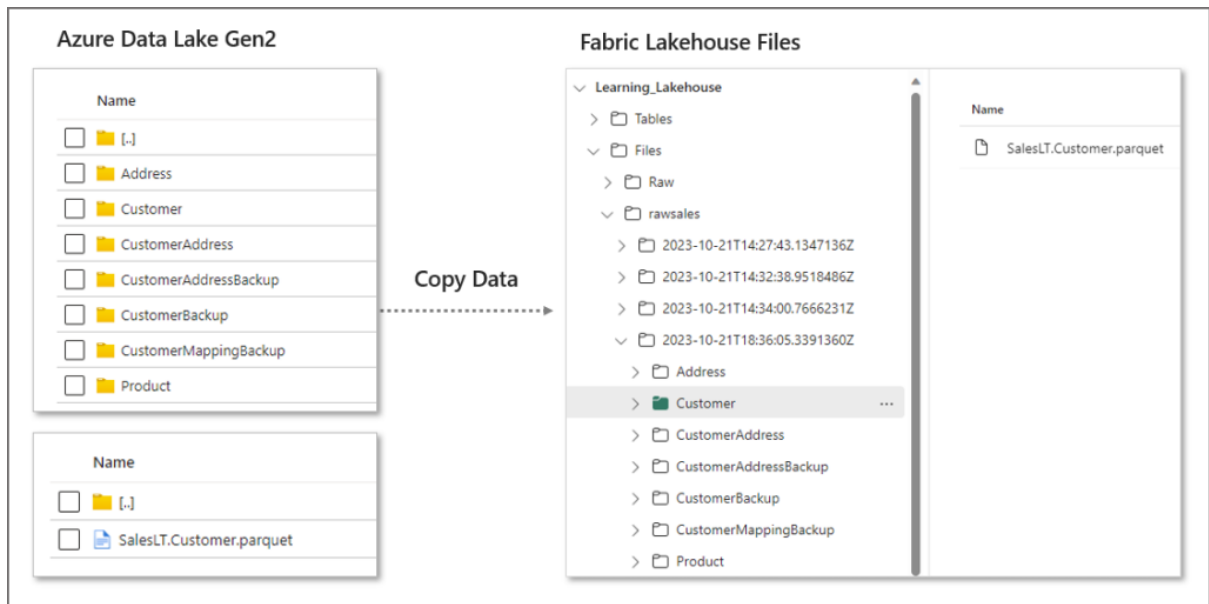
Scenario

My objective is to create a **Lakehouse** in **Fabric** that I can then load source data into and create Delta tables, in effect I want to create a **Raw/Bronze** zone. I can then query these Delta tables for further downstream processing (Silver/Gold etc). Now, I don't necessarily know what source data is going to be available so I need a process that can save available data to the **Files** section of a Lakehouse and then iterate over a list of available data to process.

I'm not going to work through the actual end-to-end solution in this post as that's a bit pointless, you can do that yourself by following the [scenario guide here](#). What I am going to do is modify it for my own use, tweak it here and there to get it to do what I want. The section of the MS guide I'm going to use as a template is the [Prepare and transform data in the lakehouse](#)

Scenario Objective

- Load raw data into a new folder each time the process is run, this folder will contain child folders per source table/object
- Pass through a list of the child folders into a Notebook to iterate over, add some extra metadata columns like load datetime and source folder name.
- Allow the source schema to drift (e.g. a new source column is added) and save to a Delta table
- Each load is to be appended to the relevant Delta table



Each time the load is run, any available data in the source folder will be copied over to a **new datetime folder** in the Lakehouse Files section. This source data could be anything, a full copy of the source data, or incremental data. Either way, we are saving a new copy of the data with each load.

The Notebook code is available in [GitHub here](#).

Pipeline and Notebook Process

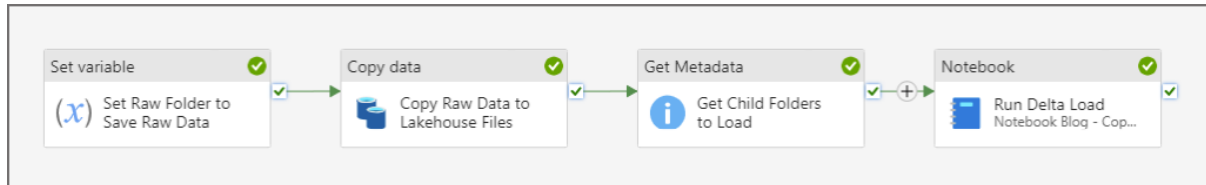
For the actual process, it'll be doing the following:

- Use a **Set Variable** to set a variable to be a datetime – this will be used as the name of the folder to store the raw Files
- Use a **Copy Data** task to move data into the Files section of a Lakehouse into a datetime folder
- Use a **Get Metadata Activity** task to read the list of source folders that have been copied to the datetime folder
- Pass this JSON metadata plus the source folder into a **Notebook** as parameters
- In the Notebook, it'll loop through the JSON metadata (folder list):
 - Extract each folder **name** from the JSON
 - Add datetime column to the dataframe

- Add source folder column to the dataframe
- Append into a Delta table

Pipeline Flow Diagram

The pipeline flow that we'll be creating is as follows:



Walkthrough

Let's now walkthrough the process of creating a new Lakehouse to store the raw Files and Delta tables, creating the Spark Notebook, then the Data Pipeline that ties everything together.

Create new Lakehouse

This step assumes you have a workspace in Power BI/Fabric that is assigned to a Fabric capacity (current Premium, Fabric trial, or Azure F SKU).

- In the workspace, click **New > Lakehouse**
- Give the lakehouse a name and click **Create**
- After a few moments you'll see the Lakehouse item and also a SQL Endpoint item (read only SQL queries)

Create Spark Notebook to load Delta Tables

I'll use the tutorial file **01 – Create Delta Tables.jpynb** (available on [GitHub here](#)) and amend as appropriate to get the results I want. The amended notebook is available in my GitHub repo here. To create a new notebook:

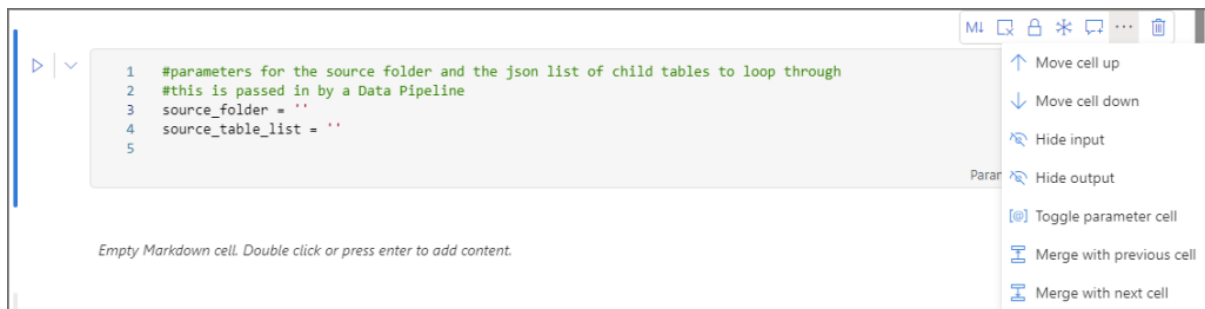
- In the workspace, click **New > Notebook**
- Then click **Add Lakehouse > Existing Lakehouse > Add**
- Select the new Lakehouse created earlier and click **Add**

This notebook will have 3 separate code cells, with 1 of the cells set as a **parameter** cell. Copy the code below into 3 separate cells. To create a cell, hover over the notebook and a menu should appear allowing you to click **Code** (Add Code Cell).

```
#standard settings in every Microsoft demo of the Lakehouse :)
#sets the V-Order special sauce
spark.conf.set("spark.sql.parquet.vorder.enabled", "true")
spark.conf.set("spark.microsoft.delta.optimizeWrite.enabled", "true")
spark.conf.set("spark.microsoft.delta.optimizeWrite.binSize", "1073741824")
```

With the code below, once the variables have been copied into the cell, click the ellipsis (More Commands) to the right of the cell and select **Toggle Parameter Cell**. You should then see the text **Parameters** appear to the bottom right of the cell.

```
#parameters for the source folder and the json list of child tables to loop through
#this is passed in by a Data Pipeline
source_folder = ""
source_table_list = ""
```



Then the 3rd cell contains the code to process the JSON string passed through via the pipeline, create a list and populate with the **folder name**, and then iterate over this list and call a function to load the raw data to Delta tables.

```
from pyspark.sql.types import *
from pyspark.sql.functions import *
import ast
import json
#####
#create function to save each source folder to a Delta table
def loadDataFromSource(source_folder, table_name):
#load raw data from Files
df = spark.read.format("parquet").load('Files/' + source_folder + '/' + table_name)
```

```

df = df.withColumn("RawLoadDateTime", current_timestamp()) \
.withColumn("RawFolderSource",source_folder)
#append new metadata columns and save to Delta table
df.write.mode("append").format("delta").option("mergeSchema", "true").save("Tables/" + table_name)
#####
#load json list of tables
table_list = json.loads(source_table_list)
#create new empty list
table_result_list = []
#loop over json object and save name of table to list
for json_dict in table_list:
table_result_list.append(json_dict['name'])
for i in table_result_list:
loadDataFromSource(source_folder,i)

```

Data Pipeline

In this section we'll create the Data Pipeline that will load the raw data into the Lakehouse Files area and trigger the notebook to load the raw files into the Lakehouse Tables area as Delta tables.

Create new Data Pipeline

In this section we'll create the Data Pipeline that will load the raw data into the Lakehouse Files area and trigger the notebook to load the raw files into the Lakehouse Tables area as Delta tables.

- In the workspace that the Lakehouse was created in, click **New > Data Pipeline**

Set Variable

This variable is used to store the datetime that the pipeline was run and it then passed through to the Copy Data and Notebook task as a parameter.

- In the pipeline top menu, click **Activities** and select **Set Variable**
- On the activity, select the **Settings** tab and ensure **Pipeline Variable** is set
- Click **New** next to **Name** and give the variable name like **rawfoldername**, make sure **String** is selected as the **Type** then click **Confirm**

- In **Value**, hover just underneath the textbox and click **Add Dynamic Content** and in the Pipeline expression builder add **@utcnow()** (or whatever datetime expression you wish the folders to be named).

Copy Data

This task will load raw files from a source data lake folder and save to the Files section of the Lakehouse. It uses the variable created earlier as the destination folder name.

- In the pipeline top menu click **Activities** and select **Copy Data > Add to canvas**
- In the **Source** tab of the Copy Data activity, add a connection to a relevant data source. In my case its to an **Azure Data Lake Gen2** account where the source data is stored in Parquet format.
- As I'm copying the files over, I'm choosing **Binary** as the file format.

The screenshot shows the configuration window for the Copy Data activity, specifically the **Source** tab. The window has tabs for General, Source, Destination, Mapping, and Settings. The Source tab is active and contains the following settings:

- Data store type:** Radio buttons for Workspace, External (selected), and Sample dataset.
- Connection:** A dropdown menu showing a URL: `https://dhstordatalakeuk.dfs.core.wi...`. There are buttons for Refresh, Test connection, Edit, and New.
- File path type:** Radio buttons for File path (selected), Wildcard file path, and List of files.
- File path *:** A series of text boxes: `datalakehouseuk`, `/`, `raw/salesdata`, `/`, and `File name`. There is a Browse button and a dropdown arrow.
- Recursively:** A checked checkbox.
- File format *:** A dropdown menu set to `Binary`. There is a Settings icon next to it.
- > Advanced:** A link to expand advanced settings.

- On the **Destination** tab, select **Workspace** as the Data store type and select the new **Lakehouse** created earlier.
- Ensure that **Files** is selected as the **Root folder**
- Under the **File path** textbox, click **Add Dynamic Content** and add the following expression:
 - **`rawsales/{variables('rawfoldername')}`**
- Select **Binary** as the **File format**.

The screenshot shows the 'Destination' configuration page in Fabric. It includes the following settings:

- Data store type:** Workspace (selected), External
- Workspace data store type:** Lakehouse
- Lakehouse:** Learning_Lakehouse (with Refresh, Open, and New buttons)
- Root folder:** Files (selected), Tables
- File path:** rawsales/@{variables('rawfoldername...')} / File name (with a Browse button)
- File format *:** Binary (with a Settings button)
- Advanced:** > Advanced

FYI there is a **shortcut** feature within Fabric that can link to external data in an ADLS Gen2 account instead of copying the files into the Lakehouse (backed by OneLake), however for my objective, I wish to be able to extend the usage of the pipeline to any data source I can extract from and land into OneLake.

Get Metadata

This is the activity that will now generate a list of all the child folders that were copied over as part of the Copy Data activity. This generation is dynamic, as I may not know at pipeline runtime what data is available in the source folder. E.G. the source folder could be cleared down and an incremental set of data copied over.

- In the pipeline top menu click **Activities** and select **Get Metadata**
- On the **Settings** tab, select the new **Lakehouse** in the **Workspace**
- Ensure **Files** is selected as the **Root** folder
- Under the **File path** textbox, click **Add Dynamic Content** and add the following expression:
 - **`rawsales/@{variables('rawfoldername')}`**
- In **Field list**, click **+ New** and select **Child items**

This will then generate a JSON object with all the available folder names that were copied over, and can be passed as a parameter into the Notebook.

General **Settings**

Data store type Workspace External

Workspace data store type ⌵

Lakehouse ⌵ 🔄 Refresh ✎ Open + New

Root folder Tables Files

File path 📁 Browse

File format * ⌵ ⚙️ Settings

Field list * + New | 🗑️ Delete

Argument

⌵

Notebook

Let's now add a Notebook activity to the pipeline and pass in the JSON object of child folders into the **json_table_list** parameter, plus the **source folder** parameter.

- In the pipeline top menu click **Activities** and select **Notebook**
- On the **Settings** tab, select the **Notebook** created earlier
- In the **Base parameters** section, create 2 parameters matching the names of the Notebook parameters, and for each **Value**, click **Add**

Dynamic Content:

- source_folder: **rawsales/@{variables('rawfoldername')}**
- json_table_list: **@string(activity('Get Child Folders to Load').output.childItems)**

General **Settings**

Notebook * ⌵ 🔄 Refresh ✎ Open + New

▼ Base parameters

+ New | 🗑️ Delete

<input type="checkbox"/>	Name	Type	Value
<input type="checkbox"/>	<input type="text" value="source_folder"/>	<input type="text" value="String"/> ⌵	<input type="text" value="rawsales/@{variables('rawfoldername...')}"/> ⌵
<input type="checkbox"/>	<input type="text" value="source_table_list"/>	<input type="text" value="String"/> ⌵	<input type="text" value="@string(activity('Get Child Folders to...'))"/> ⌵

Running the Pipeline

Now that we have created all our activities and wired up the Notebook, let's trigger the pipeline and see the output. On the pipeline top-menu, click **Run > Validate**. If the pipeline is validated successfully then click **Run > Run**. We'll be able to see each activity status in the **Output** window.

The screenshot shows the 'Output' tab of a pipeline run. The pipeline run ID is 64e7eb64-b337-46c9-8327-f1bf520a65bb and the status is 'Succeeded'. The output table lists the following activities:

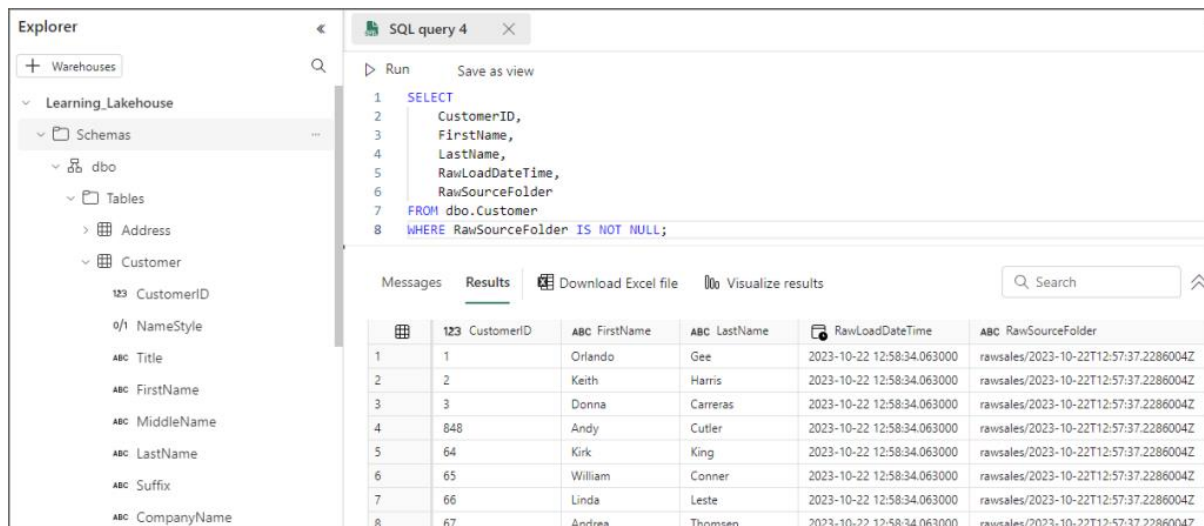
Activity name	Activity status	Run start	Duration	Input	Output
Run Delta Load	✔ Succeeded	10/22/2023, 1:33:13 PM	1m 8s	→	↔
Get Child Folders to Load	✔ Succeeded	10/22/2023, 1:33:10 PM	3s	→	↔
Copy Raw Data to Lakehouse Files	✔ Succeeded	10/22/2023, 1:32:47 PM	23s	→	↔
Set Raw Folder to Save Raw Data	✔ Succeeded	10/22/2023, 1:32:46 PM	Less than 1s	→	↔

Now if we open the Lakehouse and expand the **Tables** area, we should see the list of Delta tables. By clicking the ellipsis next to a table name and selecting **View Files** we can now see all the Parquet files (and the `_delta_log` folder) that underpin that table.

The screenshot shows the 'Explorer' view of a Lakehouse. The 'Customer' table is selected, and its files are displayed in a table view:

Name	Date modified
<code>_delta_log</code>	10/21/2023 9:10:03 PM
<code>part-00000-586b845d-7d56-454a-abf5-8fc0b991d273-c000.snappy.parquet</code>	10/21/2023 9:10:04 PM
<code>part-00000-a098e6b9-b1a9-4395-8124-1ba74d71a5af-c000.snappy.parquet</code>	10/22/2023 1:49:25 PM
<code>part-00000-a0d75d4c-7257-4aba-b86c-40fdebfe1325-c000.snappy.parquet</code>	10/22/2023 1:58:34 PM
<code>part-00000-aa027e59-22bf-4109-9644-a9be16d0d1a9-c000.snappy.parquet</code>	10/21/2023 9:33:48 PM
<code>part-00000-cc092df1-9320-42fc-9ee1-d2a7c012382a-c000.snappy.parquet</code>	10/22/2023 1:33:33 PM
<code>part-00000-cfdb6b96-eb6a-4628-8067-06d1c1e298d0-c000.snappy.parquet</code>	10/22/2023 1:30:41 PM
<code>part-00000-d1719c53-e166-4b56-ad71-3150bcba8e8-c000.snappy.parquet</code>	10/21/2023 9:39:44 PM

By switching to the **SQL Endpoint** view (top-right menu in the Lakehouse) and writing a SQL query (yes, back on familiar territory) I can then query the table and also see the 2 new derived columns. I can then use this downstream is further loading processes.



The screenshot shows a SQL query interface with an Explorer pane on the left and a query editor on the right. The Explorer pane shows a hierarchy of Warehouses, Schemas, and Tables. The query editor shows a SQL query that selects CustomerID, FirstName, LastName, RawLoadDateTime, and RawSourceFolder from the dbo.Customer table, filtered by RawSourceFolder IS NOT NULL. The results pane shows a table with 8 rows of data.

```
SELECT
CustomerID,
FirstName,
LastName,
RawLoadDateTime,
RawSourceFolder
FROM dbo.Customer
WHERE RawSourceFolder IS NOT NULL;
```

	123 CustomerID	ABC FirstName	ABC LastName	RawLoadDateTime	ABC RawSourceFolder
1	1	Orlando	Gee	2023-10-22 12:58:34.063000	rawsales/2023-10-22T12:57:37.2286004Z
2	2	Keith	Harris	2023-10-22 12:58:34.063000	rawsales/2023-10-22T12:57:37.2286004Z
3	3	Donna	Carreras	2023-10-22 12:58:34.063000	rawsales/2023-10-22T12:57:37.2286004Z
4	848	Andy	Cutler	2023-10-22 12:58:34.063000	rawsales/2023-10-22T12:57:37.2286004Z
5	64	Kirk	King	2023-10-22 12:58:34.063000	rawsales/2023-10-22T12:57:37.2286004Z
6	65	William	Conner	2023-10-22 12:58:34.063000	rawsales/2023-10-22T12:57:37.2286004Z
7	66	Linda	Leste	2023-10-22 12:58:34.063000	rawsales/2023-10-22T12:57:37.2286004Z
8	67	Andrea	Thomsen	2023-10-22 12:58:34.063000	rawsales/2023-10-22T12:57:37.2286004Z

Conclusion

As I said at the beginning of this blog post, I'm not much of a Spark Developer, but I'm willing to jump in and get going, and using the Microsoft End-to-End scenarios has really helped me take a step towards using PySpark for data loading and transformation.

And here's the great thing, now that this basic process is in-place, I can start to extend and modify it as I add more objective to the process. Ultimately I want this to be a generic **module** I can plug into any data loading process for my raw layer. I want to look at using Python's multithreading to execute the Delta table loads in parallel, and adding error handling to the process too. All in good time, and I will of course share the journey.

As always, feel free to reach out to me to discuss anything in this blog.

References

- [End-to-end tutorials in Microsoft Fabric – Microsoft Fabric | Microsoft Learn](#)
- [Lakehouse tutorial – Prepare and transform data in the lakehouse – Microsoft Fabric | Microsoft Learn](#)
- [Azure Data Factory Get Metadata Example \(mssqltips.com\)](#)
- [azure data factory – how to pass the outputs from Get metadata stage and use it for file name comparison in databricks notebook – Stack Overflow](#)
- [python 3.x – How to create a list from json key:values in python3 – Stack Overflow](#)
- [Develop, execute, and manage notebooks – Microsoft Fabric | Microsoft Learn](#)
- [Apache Spark Tutorial with Examples – Spark By {Examples} \(sparkbyexamples.com\)](#)