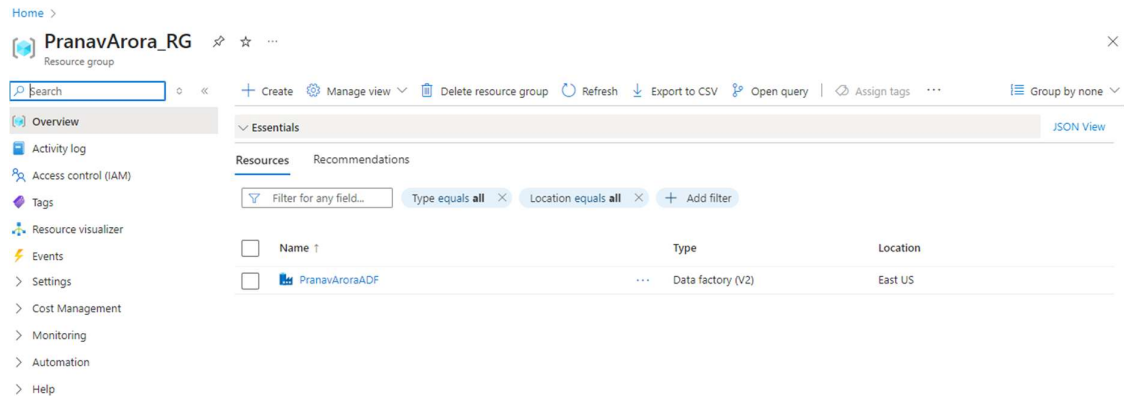


# Project Documentation – Pipeline Creation

## Steps Followed for Pipeline Creation:

1. Create a Resource Group and create an ADF resource.



2. Now we will create a SHIR, so that we can access the on-prem files. Below are the steps followed for the same:

Go to Manage tab in your ADF resource → Click on Integration Runtimes tab → Click on '+New' → Select 'Azure Self Hosted' and continue → Then select 'Self Hosted' and continue → Give Name to your Runtime → Now copy the Key 1 or Key 2 and keep it in a notepad or secure place, and click on the 'Download and install integration runtime' if we don't have the IR application on our local laptop/pc → setup the application, and paste one of the Keys, and now if we come back to the Integration runtime tab and refresh, our SHIR will be up and running

Install integration runtime on Windows machine or add further nodes using the Authentication Key.

Name ⓘ

integrationRuntime1

Self-contained interactive authoring ⓘ



☒ Disable ☐ Enable

Option 1: Express setup

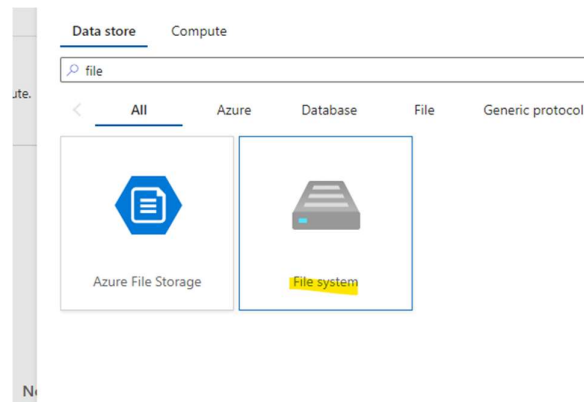
[Click here to launch the express setup for this computer](#)

Option 2: Manual setup

Step 1: **Download and install integration runtime**

Filter by name						
Showing 1 - 2 of 2 items						
Name ↑↓	Type ↑↓	Sub-type ↑↓	Status ↑↓	Related ↑↓	Region ↑↓	Version ↑↓
 AutoResolveIntegrationRuntime	Azure	Public	✓ Running	0	Auto Resolve	---
 PranavSHIR	Self-Hosted	---	✓ Running	0	---	5.54.9271.2

- Now we will create a Linked service to access our local data folder, and we will use our SHIR in it. To create Linked Service, follow the steps below:  
Go to manage tab in ADF → Go to Linked Service → Click on '+New' → Select 'File system' and continue → Give name to our Linked Service, Select our SHIR for the runtime, now in our Host (we will put the folder location of the files we want to access: Ex- "C:\Users\Apex\Desktop\PranavADF"), we will then add the Username and Password (this will be the Laptop/device user name and password, and if we have setup laptop with Microsoft ID then Microsoft mail id and its password (preferred method)), and then we Test Connection.



On Testing connection, we will get error, so we need to run two commands in our Power Shell, as it will disable the Local Folder Path Validation, which is by default on when we setup our SHIR. (We run these commands as Admin).

Refer to this link for the commands and issue resolution:

<https://stackoverflow.com/questions/76402958/azure-data-factory-linked-service-to-c-drive>

And then again Test Connection, it works –

**New linked service**  
File system [Learn more](#)

Connect via integration runtime  
PranavSHIR

The credentials are stored in the machines of self-hosted integration runtime if you don't choose to store them in Azure Key Vault.

Host  
C:\Users\Apex\Desktop\PranavADF

User name  
Apex

Password  
\*\*\*\*\*

Annotations  
+ New

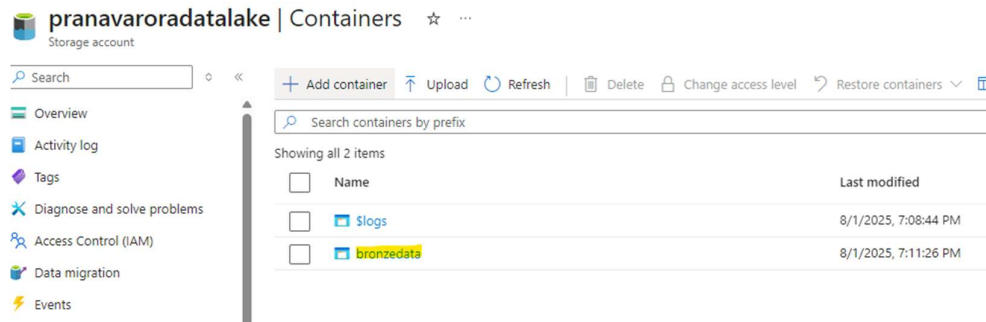
> Parameters  
> Advanced

Connection successful

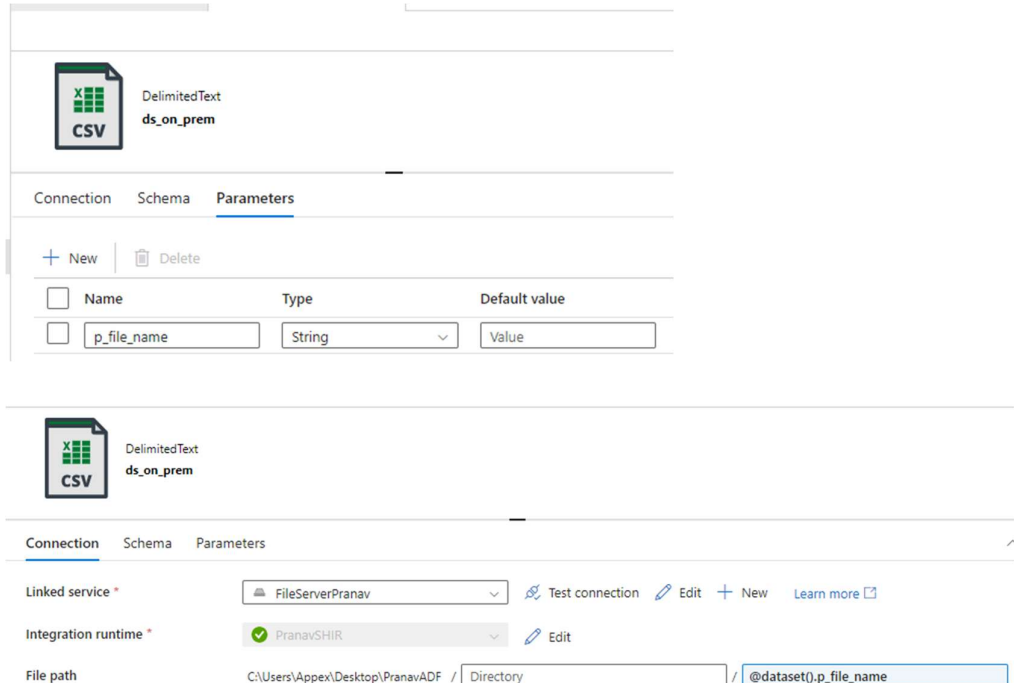
Create Back Test connection Cancel

And then we create our Linked service successfully.

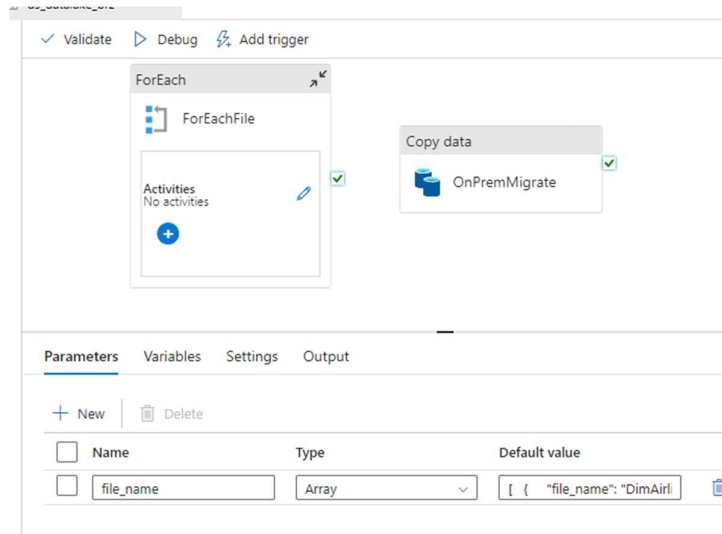
4. Now go to our Resource Group, and container a Storage Account Resource, in which we can create a container.



5. Now we will create a Linked Service for this storage, and we will not use SHIR, we will use the default Integration Runtime provided by Azure.
6. Now we will start creating our ADF pipeline:
  - a. We will create a Copy Activity → In source we will create a dataset for File System, and we select the file format as CSV (since the files in our laptop folder are CSV), and select the linked service (the one we created for our local data). Similarly, we will create a dataset for the sink, using Linked service of our data lake.
  - b. We will also parameterize our Source and Sink Datasets, so that we can make it dynamic in nature, as we have 3 files in our folder.



- c. Now we create a pipeline level variable to store our file names, and pass the default value through it in form of a list:

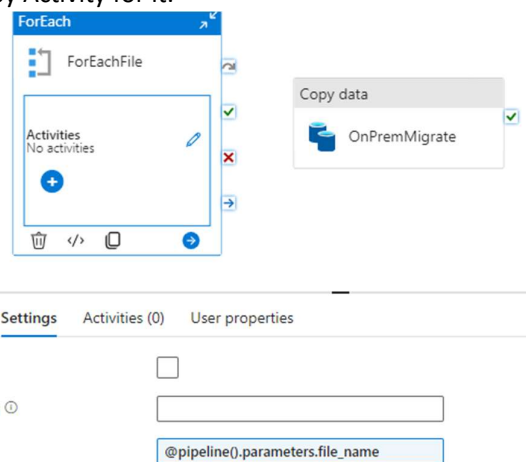


```

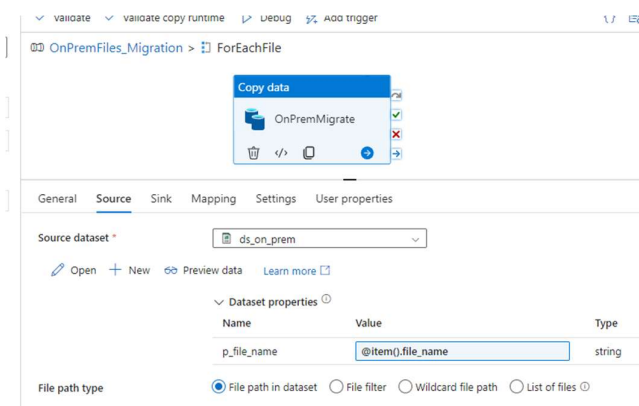
[
  {
    "file_name": "DimAirline.csv"
  },
  {
    "file_name": "DimFlight.csv"
  },
  {
    "file_name": "DimPassenger.csv"
  }
]

```

- d. Now we create a For Each Activity and pass the Pipeline parameter as an input to it, and move our Copy Activity inside the For Each Activity so that we can pass each file name, and run the Copy Activity for it.



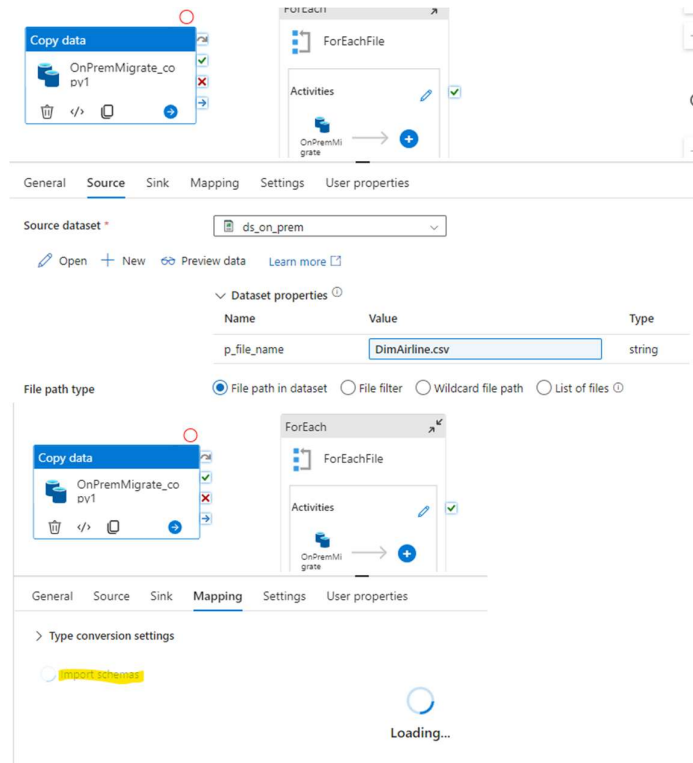
- e. Also now in the Copy Activity, we will put the default value of our activity as the Parameter value output of For Each Activity. (for both Source And Sink)

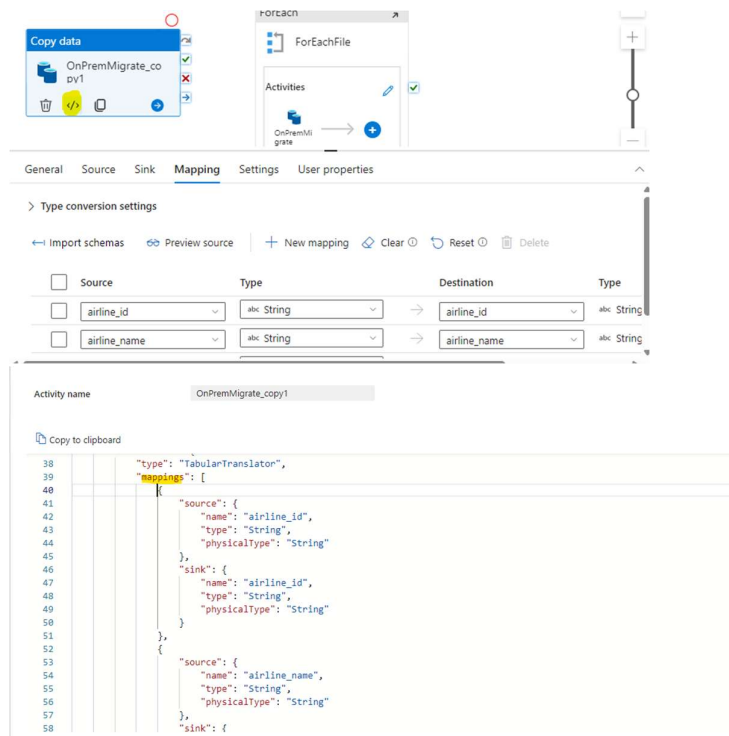


- f. Now we want Dynamic mapping as well, so we will create 3 pipeline variables (one for each CSV files mapping) –

Parameters Variables Settings Output			
<input type="checkbox"/>	Name	Type	Default value
<input type="checkbox"/>	file_name	Array	[ { "file_name": "DimAirli
<input type="checkbox"/>	p_airline_mapping	Object	Value
<input type="checkbox"/>	p_flight_mapping	Object	Value
<input type="checkbox"/>	p_passenger_mapping	Object	Value

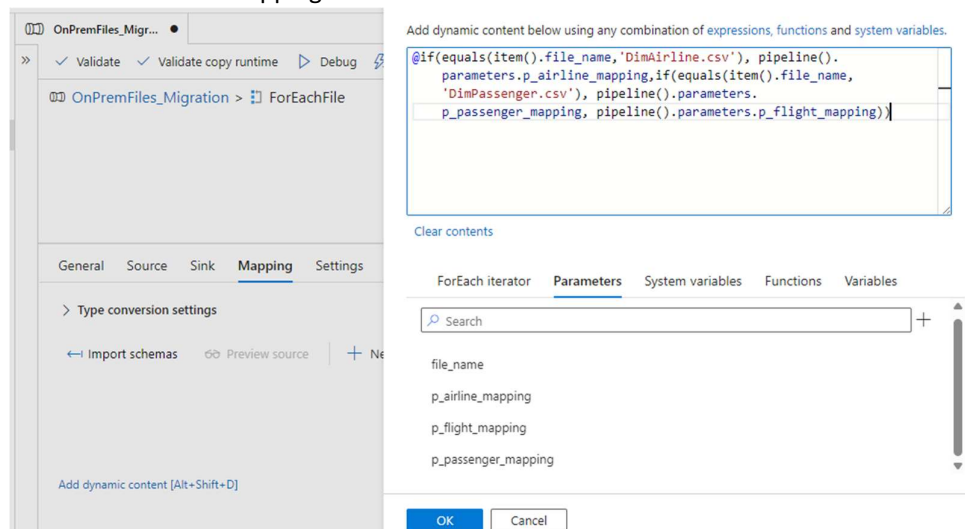
Now to get the mappings, we make a copy of our Copy Activity and instead of Parameterized file name, we put one file name at a time, import the mapping and using the code for the mapping fill the values for the Mapping parameters in the above screenshot.





Once you fill the mapping values for the mapping parameters, you can delete the additional copy activity added to get the schemas.

- g. Now go to the Copy Activity inside the For Each Activity, and we will add dynamic content within the mapping tab here:



Now our ADF Pipeline is ready for the OnPrem Files ingestion.

OnPremFiles\_Migr... x

» ✓ Validate ▶ Debug ⚙️ Add trigger

ForEach

ForEachFile

Parameters Variables Settings **Output**

Pipeline run ID: a2bf06ac-86e3-47be-afdd-f07865a777e Pipeline status: Succeeded [View debug run consu](#)

All status ▾ List ▾ [Monitor in Azure Metrics](#) [Export to CS](#)

Showing 1 - 4 of 4 items

Activity name	Activity st...	Activit...	Run start	Duration	Integration runtime
OnPremMigrate	Succeeded	Copy data	8/2/2025, 6:35:50 PM	19s	PranavSHIR
OnPremMigrate	Succeeded	Copy data	8/2/2025, 6:35:50 PM	19s	PranavSHIR
OnPremMigrate	Succeeded	Copy data	8/2/2025, 6:35:50 PM	21s	PranavSHIR
ForEachFile	Succeeded	ForEach	8/2/2025, 6:35:49 PM	24s	

- Now we will build pipeline for GitHub data Ingestion. We will create one Web Activity and paste our GitHub file's raw URL, put Method as GET and Authentication as None (since it is a public repo)

anshlambagit Add files via upload 5b8ea03 · last week History

Code Blame 62 lines (62 loc) · 1.29 KB

Raw

```

1  [
2  {
3      "airport_id": 1,
4      "airport_name": "JFK International",
5      "city": "New York",
6      "country": "USA"
7  },
8  {
9      "airport_id": 2,
10     "airport_name": "Heathrow",
11     "city": "London",
12     "country": "UK"
13 },

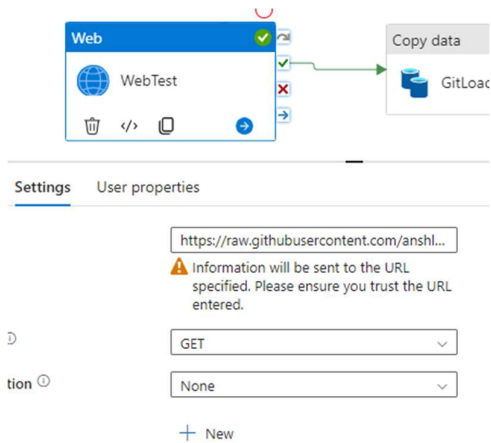
```

← → ↺ [raw.githubusercontent.com/anshlambagit/AnshLambaYoutube/refs/heads/main/ADF\\_Project/DimAirport.json](https://raw.githubusercontent.com/anshlambagit/AnshLambaYoutube/refs/heads/main/ADF_Project/DimAirport.json)

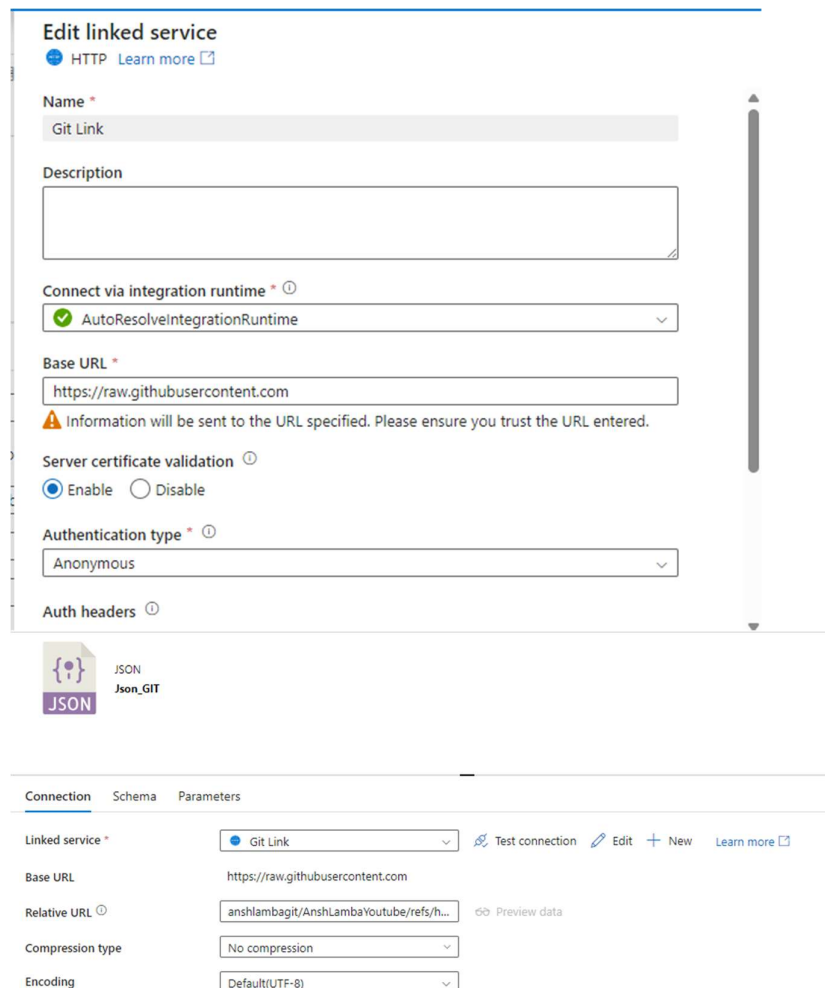
```

{
  "airport_id": 1,
  "airport_name": "JFK International",
  "city": "New York",
  "country": "USA"
},
{
  "airport_id": 2,
  "airport_name": "Heathrow",
  "city": "London",
  "country": "UK"
},
{
  "airport_id": 3,
  "airport_name": "Chhatrapati Shivaji",
  "city": "Mumbai",
  "country": "India"
},
{
  "airport_id": 4

```



Then on success of success of Web Activity, we will run a Copy Activity. In our sink we will keep ADLS, in our Source, we will keep JSON (since the GitHub file is a JSON File), and in our linked service –



The Base URL is the Raw Git Files starting bit of URL and Relative URL is post the backslash of our base URL.

Pipeline is ready, we can run it.



The screenshot shows an Azure Data Factory pipeline with two activities: 'Web' (containing 'WebTest') and 'Copy data' (containing 'GitLoad'). Both activities are marked as successful with green checkmarks. Below the pipeline diagram, the 'Output' tab is selected, showing the pipeline run ID 'd46a71c5-a2b8-474d-9eab-60d3bda25103' and a status of 'Succeeded'. A table lists the activities with their details.

Activity name	Activity status	Activity name	Run start	Duration	Integration runtime
GitLoad	Succeeded	Copy data	8/2/2025, 8:09:02 PM	12s	AutoResolveVeli
WebTest	Succeeded	Web	8/2/2025, 8:08:46 PM	15s	AutoResolveVeli

- Now we will create another Pipeline to get data from SQL DB. First we will go to our Resource Group, search for Azure SQL and select it → then we choose SQL databases (with Single Database Option) → Give our DB a name, create a server (Give name to server, For authentication method select 'Use both SQL and Microsoft Entra authentication', create admin login and password, also set your account as Admin as well), set workload env as 'Development', in "Compute+storage" select Configure DB (Choose Serverless, and set memory according to needs and click on Apply); now go to networking tab of SQL DB creation (set it as public endpoint, set "Allow Azure services and resources to access this server" to Yes and "Add current client IP address" to Yes) and then create your DB.

Once the resource is created, go to the Database from your Resource Group → click on Query Editor and login, and now run the script to create and insert data into dummy table for this project (Script available in this repo - fact\_bookings\_full.sql)

Now our table is ready, so we can go back to ADF to build a pipeline for getting data to our data lake from the SQL DB

Create a Copy activity for our SQL DB, where SQL DB is the source (create dataset and linked service) and Data Lake is the Sink (File type in Sink is Parquet this time)

The screenshot shows the configuration window for a 'Copy data' activity. The 'Source' tab is active, displaying the 'Source dataset' as 'AzureSqlTable1'. Under 'Use query', the 'Query' is set to 'select \* from dbo.FactBookings'. The 'Sink' tab is also visible, showing the 'Sink' as 'DataLake1'.

Now we want to maintain watermark json files in our data lake and load our data incrementally using those files, basis our watermark column (Booking Date) and also update these Json Files, so we will add more logic to this pipeline now:

Now we will create a folder called 'Monitor' in our data lake container, and add a JSON file there called lastload.json (This file will contain a timestamp with date 1900-01-01 so that we can have our first load as a full load, and once we update this it would become incremental in nature)

### monitor/lastload.json

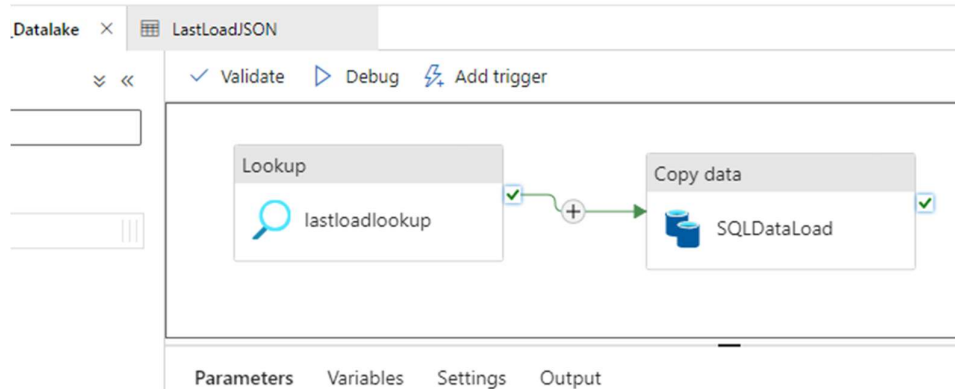
Blob

Save Discard Download Refresh Delete

Overview Versions Edit Generate SAS

```
1 [{"lastload": "1900-01-01T23:23:22.0335153Z"}]
```

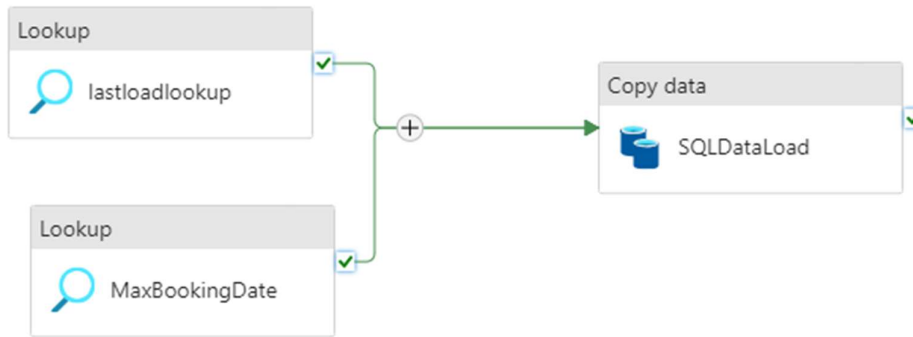
Now in our pipeline we will create a Lookup activity called "lastloadlookup" which will get the data from this above created JSON file. Now we will connect the 2 Activities, and the query in our copy activity will use the output of the Lookup activity in a filter.



The image shows two parts of the Azure Data Factory interface. On the left, the 'Source' tab of a 'Lookup' activity is selected, showing the 'Source dataset' dropdown and 'Use query' option. On the right, the 'Pipeline expression builder' is open, displaying a SQL query: `select * from dbo.FactBookings where booking_date > '{@activity('lastloadlookup').output.firstRow.lastloadlookup}'`. Below the query editor, the 'Activity outputs' tab is active, showing a search bar and a list of outputs: 'lastloadlookup', 'lastloadlookup activity output', and 'lastloadlookup first row' (Data of the first row).

Now we have lastloadlookup, which will give the last date of data load, but we will need the max of booking date so that we know till which date the data is loaded in our last load, so that we can update the value in the json used by the lastloadlookup activity (basically updating last load date basis the most recent booking date in our data), so we create this lookup activity with the following query –

The image shows a pipeline diagram with a 'Lookup' activity connected to an 'SQLDataLoad' activity. The 'Lookup' activity is configured with the name 'MaxBookingDate'. Below the diagram, the 'Settings' tab for the 'Lookup' activity is open, showing the following configuration: 'Source dataset' is 'AzureSqlTable1', 'First row only' is checked, 'Use query' is selected (with 'Table' and 'Stored procedure' also available), and the 'Query' is `SELECT MAX(booking_date) as MAX_DATE from dbo.FactBookings`. The 'Query timeout (minutes)' is set to 120.



Also, we can update our Copy Activity's query:

### Pipeline expression builder

Add dynamic content below using any combination of [expressions](#), [functions](#) and [system variables](#).

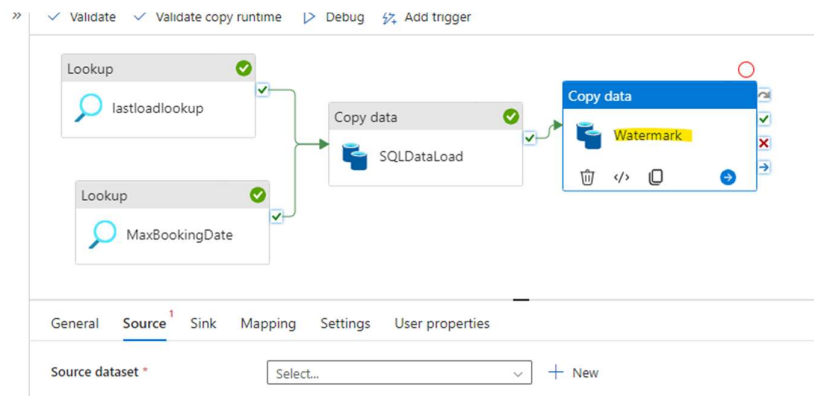
```

select * from dbo.FactBookings
where booking_date > '{@activity('lastloadlookup').output.
  firstRow.lastload}'
and booking_date <= '{@activity('MaxBookingDate').output.
  firstRow.MAX_DATE}'
  
```

[Clear contents](#)

[Activity outputs](#) [Parameters](#) [System variables](#) [Functions](#) [Variables](#)

Now we will create another Copy Activity (named Watermark), which will take the output from the MaxBookingDate Activity, and it will replace the lastload.json in our monitor with a JSON file with the same name and value from the output of the MaxBookingDate Activity, thereby updating our last load date. Also, the source of this activity will be an empty.json file which is a JSON file which is empty so that we can add value to it and load it into the Sink.



Run the pipeline, and the lastload.json will have the updated date automatically.

## monitor/lastload.json

Blob

Save Discard Download Refresh Delete

Overview Versions Edit Generate SAS

```
1 [{"lastload": "2025-06-30T00:00:00"}]
2
```

9. Now we will create a parent pipeline for executing these pipelines in our desired order.

General Settings User properties

Invoked pipeline \* OnPremFiles\_Migration [Open](#) [+ New](#)

Wait on completion ☒

Parameters

Name	Type	Value	Default value
file_name	array	Value	[{"file_name": "DimAirline.csv"}, {"file_name": "DimFlightcs...
p_airline_mapping	object	Value	{"source": {"name": "airline_id", "type": "String", "physicalTyp...
p_flight_mapping	object	Value	{"source": {"name": "flight_id", "type": "String", "physicalTyp...
p_passenger_mapping	object	Value	{"source": {"name": "passenger_id", "type": "String", "physica...

Since our OnPremFiles\_Migration pipeline is parameterized, we will have to create these parameters with the same default values in our Execute Pipeline for this pipeline, and pass it in the values shown in above screenshot.

General Settings User properties

Invoked pipeline \* OnPremFiles\_Migration [Open](#) [+ New](#)

Wait on completion ☒

Parameters

Name	Type	Value	Default value
file_name	array	@pipeline0.parameters.file_name	[{"file_name": "DimAirline.csv"}, {"file_name": "DimFlightcs...
p_airline_mapping	object	@pipeline0.parameters.p_airline_ma...	{"source": {"name": "airline_id", "type": "String", "physicalTyp...
p_flight_mapping	object	@pipeline0.parameters.p_flight_map...	{"source": {"name": "flight_id", "type": "String", "physicalTyp...
p_passenger_mapping	object	@pipeline0.parameters.p_passenger_...	{"source": {"name": "passenger_id", "type": "String", "physica...

Run the pipeline:

Parent Pipeline

OnPremFiles Migrati...

Validate

Debug

Add trigger

Execute Pipeline

Execute On Prem...  
OnPremFiles Migrati...

Execute Pipeline

Execute GIT  
GIT Data

Execute Pipeline

Execute SQL  
SQLDB to Datalake

Expand toolbox pane

Parameters

Variables

Settings

Output

Pipeline run ID

d89d16bb-fffe-4704-a0d7-329469dad6df

Pipeline status

Succeeded

View debug run

All status

Showing 1 - 3 of 3 items

Activity name	Activity st...	Activit...	Run start	Duration	Integration runtime	User prop...	Activity run ID
Execute SQL	Succeeded	Execute Pipelin	8/3/2025, 1:55:10 AM	1m 5s			be163bec-fb58-4t
Execute GIT	Succeeded	Execute Pipelin	8/3/2025, 1:54:39 AM	32s			c8eb14ba-79c7-4i
Execute On Prem Files	Succeeded	Execute Pipelin	8/3/2025, 1:54:12 AM	27s			0f2a8f2e-2355-45