

## Azure Data Factory - Part 4(a)

### Extract, Transform and Load

When you're extracting data from multiple sources, transforming it (filtering, sorting, aggregating, joining data, cleaning) and then loading the transformed data into the target location.

### Azure Data Factory Introduction –

This is a cloud based ETL tool. Can create data driven workflows which help orchestrate data movement. Also helps with transformation of the data.

The Azure Data Factory process looks somewhat like –

1. Connected to the required data sources
2. Ingest data from the source
3. Transform the data in the pipeline if required
4. Publish the data onto a destination – Azure Data Warehouse, Azure SQL Database, Cosmos DB
5. Can also monitor the pipeline as it is running

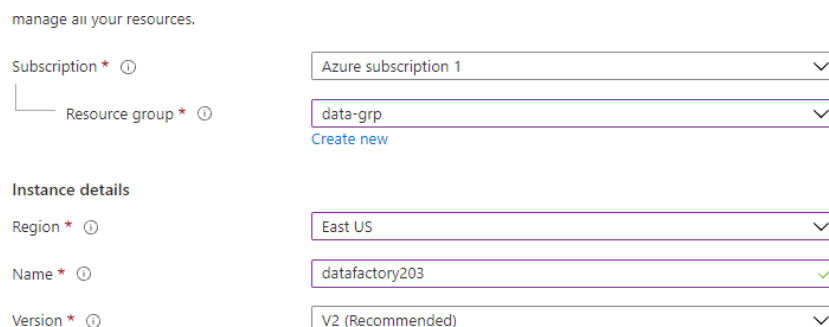
Azure Data Factory has different components –

1. Linked Service – This enables you to ingest data from a data source. The Linked Service can create the required compute resources to bring in the data from a data source.
2. Datasets – This represents the data structure within the data store that is being referenced by the linked service object
3. Activity – This is the actual transformation logic. You can also have simple copy activities to copy data from a source to a target.

In Azure Data Factory we would define our pipeline which is a logical grouping of activities in Data Factory. There could be many activities within a pipeline. For example, if you want to copy data from Azure SQL Database to Azure SQL Warehouse, you will have to create two linked services and one copy activity. Based on the requirements / activity there would be a compute infrastructure that would be managed by ADF.

### Getting started with Azure Data Factory – Copying CSV files from your data lake to the dedicated sql file

In your resource select Data Factory and hit create. Give your subscription, resource group, *unique* factory name and version.



The screenshot shows the 'Create new' form for an Azure Data Factory instance. It includes the following fields and options:

- Subscription \***: A dropdown menu with 'Azure subscription 1' selected.
- Resource group \***: A dropdown menu with 'data-grp' selected. Below it is a link that says 'Create new'.
- Instance details**: A section header.
- Region \***: A dropdown menu with 'East US' selected.
- Name \***: A text input field containing 'datafactory203', with a green checkmark icon to its right.
- Version \***: A dropdown menu with 'V2 (Recommended)' selected.

On the next screen select "Configure git later". Leave everything as it is and hit review and create.

Once Data Factory has been created, go to ADF Studio.

Now we would see how to copy our Log.csv file onto a table in our dedicated SQL pool from our data lake. On the ADS studio when you click on Ingest, it would open the Copy Data Tool that we saw in Azure Synapse Studio. Delete the contents of the LogData table in your dedicated sql pool and then on Copy Data tool –

- On the starting screen we select run once now
- We create a new connection for our source –
  1. We select Azure > Azure Data Lake Storage Gen2
  2. Give a name, select your subscription, select your data lake gen2 account name,
  3. Test the connection and create
- Once the connection is created, we browse for our file, remove the recursively copy option, hit next.
- Most of the settings have been auto detected like file format, column delimiter, hit next.
- Create a new connection for target. Choose Azure > Azure Synapse Analytics
  1. Give name, choose subscription, choose synapse workspace and database. For authentication, give the sql admin username for synapse that was created and the password.
  2. Test the connection and hit create.
- On the next screen select an existing table, the dbo.logdata table.
- On the next screen we see the type of the columns when they're being read and the type of the columns when they're being written onto the table.
- Add a storage account for staging area, if we try to proceed now it would give us an error because we have selected Polybase.
- To fix it, we go back and deselect type conversion and give the types of the incoming columns:
  1. Id – int32
  2. Time – Datetime
- Finish.

The table is populated in the dedicated sql pool now.

### Copying parquet data into dedicated sql pool table

Drop the contents of the logdata table.

*Side note* – in the Author section of the ADF, when you click on your pipeline, you can see all the activities part of that pipeline. Along with this each activity would have its source and target datasets which can also be viewed and reused if the dataset specifications of another activity match with this one.

Let's create a new pipeline from the Author section, click on the plus button and select pipeline.

- On the right, name the pipeline
- From the move and transform section, drag the copy data activity onto the canvas
  1. Name this activity
  2. Go to the source tab and create a new connection for the parquet files.
  3. Browse to the parquet directory after adding the Data lake. Click okay
  4. After returning to the source tab select wildcard file path and it would automatically add \*.parquet to the path
  5. Now we can open the dataset using the open button and import that schema from the schema tab and select import the schema from connection store
  6. In the sink tab we can select DestinationDataset of the previously run pipeline, give the copy method as polybase
  7. In mapping click on import schemas
  8. In settings click on enable staging and select you synapse container
- Click on Validate and it would validate the pipeline for any errors
- Hit on publish at the top
- Click on Add Trigger > Trigger now
- Head over to the SSMS to see the data in the dedicated sql pool

Sidenote – if you're not using a git repo, then you may not be able to save your pipeline at every step so you can save it ( publish it ) after it has been validated.

## Dealing with escape sequences in files

We are uploading the log.csv file but with escape characters in it. [Here](#) is the file.

We then go to ADF > Ingest to create a quick activity.

- In the source, select the previously made connection and browse and select the uploaded file
- In the next screen, if we preview the data, we can see the resource column which has the escape characters is being detected properly. *This is happening because in advanced setting on the same screen, under escape character you can specify whether it is a backslash or a forward slash or some other character or there are no escape characters at all.*
- Now go to your SSMS and drop the logdata table and recreate it with the extra column using the script –

```
CREATE TABLE [logdata]
(
    [Id] [int] ,
    [Correlationid] [varchar](200) ,
    [Operationname] [varchar](200),
    [Status] [varchar](100) ,
    [Eventcategory] [varchar](100),
    [Level] [varchar](100),
    [Time] [datetime] ,
    [Subscription] [varchar](200) ,
    [Eventinitiatedby] [varchar](1000),
    [Resourcetype] [varchar](1000) ,
    [Resourcegroup] [varchar](1000),
    [Resource] [varchar] (4000)
)
```

- Back on the Azure Data factory go to the next screen, select AzureSynapseAnalytics connection that we previously had, and select existing table > dbo.logdata
- On the next screen check the mapping, disable type conversion, give int to Id and DateTime to time columns and click next
- Next, name the pipeline and give the staging path
- Finish

Now we can go to the SSMS and query the logdata table in our dedicated sql pool to see that the Resource column has been loaded correctly.

## Generating Parquet based files

Begin by going to your Data lake gen2 and create a new directory – newparquet.

In ADF, in the author section, create a new pipeline –

- Give a name
- We'll drag and drop the copy activity onto the canvas
- Give the Copy Activity a name
- In the source tab use the data that was pointing to the Log.csv pipeline. We can also verify that it is the one by clicking on preview data. Uncheck the recursive option
- In sink, we need to create a new connection as we are creating a Parquet based file
- Select the Azure Data Lake Gen2 > Select the format of the file > Name it > select our data lake storage in linked service > browse to the newparquet folder. Select none for import schema
- In the mapping tab, click import schemas, fix the data type of the incoming columns (ID, Time)
- In settings, since we're not copying data to synapse, we would not need to enable any sort of staging area
- Publish and trigger pipeline

In our Data Lake the new parquet file will be present.

## Using a query for data transfer

Transferring data from the sql database to synapse. Go to your dedicated sql pool and delete the data from the SalesFact table. Going to ADF and selecting ingest.

- For source, Create a new connection > Azure > Azure SQL Database
- Give a name, Select Subscription, select server name, select database adventureworks, enter credentials and create
- Instead of choosing the existing tables option, we select the use query option and use the following query that we had used earlier to create the SalesFact table –

```
SELECT dt.[ProductID],dt.[SalesOrderID],dt.[OrderQty],dt.[UnitPrice],hd.[OrderDate],hd.[CustomerID],hd.[TaxAmt]
FROM [Sales].[SalesOrderDetail] dt
LEFT JOIN [Sales].[SalesOrderHeader] hd
ON dt.[SalesOrderID]=hd.[SalesOrderID]
```

- You can select the preview data button and it would show the result of your query if the query is working, click next
- Select the Azure Synapse Connection, use existing table, dbo.salesfact
- Check the column mapping in the next screen
- Next, give the name and the staging area location
- Finish

Check the SalesFact table in the Dedicated SQL pool and the data should be present.

Even though it doesn't charge us to keep the datasets and pipelines in place, it would be cleaner to delete the datasets and the pipelines. Once deleted, click publish all to save the changed. However, this would not delete the connections that we have created. Those would still be there and can be viewed from the Manage section under Linked Services.

## Mapping Data Flows

This helps us visualize the data transformations in ADF. Here you can write the required transformation logic without writing any code. The data flows are run on Apache Spark clusters. Here the Azure Data Factory will handle the transformation in the data flow.

There is also a *debug mode* – where you can see the results of each transformation. In the debug mode session the data flow is run interactively on a spark cluster.

## Using Mapping Data Flow to generate the SalesFact table generated earlier with queries and copy methods

We would use Mapping Data Flows to move data from Azure SQL Database to Azure Synapse to build our salesfact table. Delete the contents of the SalesFact table.

Head to ADF > Author > Hover mouse over Data Flows and click on the three buttons > New Data Flow

- Give the Data Flow a name on the right

We would be making use of the query below and we can see the SalesFact table is being created from SalesOrderDetail table and SalesOrderHeader table

```
SELECT dt.[ProductID], dt.[SalesOrderID], dt.[OrderQty], dt.[UnitPrice], hd.[OrderDate],
hd.[CustomerID],hd.[TaxAmt]
FROM [Sales].[SalesOrderDetail] dt
LEFT JOIN [Sales].[SalesOrderHeader] hd
ON dt.[SalesOrderID]=hd.[SalesOrderID]
```

- Since there are multiple tables, we would need to add multiple sources. Begin by clicking on Add source
  1. Name the source
  2. Create a new dataset > Azure SQL Database > Name, select your database (mine seems to be added), select SalesOrderDetail table and import schema from the connection > Click okay
  3. Upon returning to the canvas, we can see it is taking 11 columns from the SalesOrderDetail table

- Click on Add Source again for the SalesOrderHeader table
  1. Name the source
  2. Create a new dataset > Azure SQL Database > Name, select your database (mine seems to be added), select SalesOrderHeader table and import schema from the connection > Click okay
  3. We can see that there are 26 columns being read from this dataset
- Now we need to perform a join on these tables. We can click on the plus and a list of operations that are possible on the data would show up, we select join
  1. We can now name the join, select the left and right streams (SalesOrderHeader), select the type (left join in our case)
  2. In the join conditions, we refer to the query and see that we are joining on SalesOrderID, so we enter that on the left stream column and right stream column
- We now have a left join in place with the join condition that we want but it has 37 columns in the output
- Click on the plus button for the Join card, scroll down to sink and select that to tell the DataFlow of our Destination dataset.
  1. Name the sink, select new Dataset > Select Azure Synapse > Select your connection > Select the SalesFact table
  2. Now we see there are 34 columns in it. We can go to the settings tab of the sink card and switch off automatic mapping
  3. We can then see which columns are being mapped to which columns in the SalesFact table
- Publish all the changes
- To execute it, we have to create a pipeline, drag and drop the Data Flow card onto the canvas from under the move and transform section
- Name your Data Flow card, in the settings tab choose the Data Flow, select the computer type and core count (choose the cheapest option as we would be charged for it based on the usage), give the staging area that we have set up
- Validate everything and Publish
- Trigger the run

This pipeline would take time to run and would not be as fast as the other copy data pipelines. This is because this activity takes time to build up the Apache Spark clusters and once the clusters are up, only then does the pipeline runs.

After about 6-7 minutes, the pipeline completed. The data shows up in the SalesFact table.

### Using the Mapping Data Flow feature to create Dimension Tables

For the CustomerView dimension Table, we used the following query. It has two tables as the source tables and there is an additional join condition.

```
SELECT ct.[CustomerID],ct.[StoreID],st.[BusinessEntityID],st.[Name] as StoreName
FROM [Sales].[Customer] as ct
LEFT JOIN [Sales].[Store] as st
ON ct.[StoreID]=st.[BusinessEntityID]
WHERE st.[BusinessEntityID] IS NOT NULL
```

Delete the contents of the DimCustomer from the dedicated sql pool. So, we create a data flow based on this.

- Create a new Data Flow, name it
- Click Add source, give the stream a name and going to the Azure SQL Database add the Sales.Customer table.
- Click Add source again and add the Sales.Store table
- Click the plus button and join, name the join, give the right stream as Store table, select the type of join (left) , give the columns for the first join condition
- Now we do not want all the columns, to remove unwanted columns, we would have to add a select operation
  1. In settings tab of the select card, we remove the columns that we do not want and rename the columns according to how the Customer\_View dimension table is supposed to be.
- Next using the plus button we can create an Alter Row card
  1. Name the Alter columns card, in the alter row conditions field, select Delete if

2. To fill out the condition we can use the expression builder which you can open when you click inside the textbox for the condition
  3. Once the expression builder is open, you can go to functions tab and search for null. We would be using the `isNull()` function here so we select that
  4. In the input schema tab, click on `BusinessEntityID`. Click save and finish.
- Add a sink from the remove null alter card and select Azure synapse analytics and add you dim customer table
  - Check if the mapping is correct
  - Validate it and publish.
  - Create a new pipeline, drag the data flow card on to the canvas, name the data flow, from settings select the data flow that we just created, give a staging area
  - Validate and publish
  - Trigger run

After about 5-6 minutes the pipeline would be done running. The data is loaded into the `DimCustomer` table.

### Recreating the `DimProduct` table

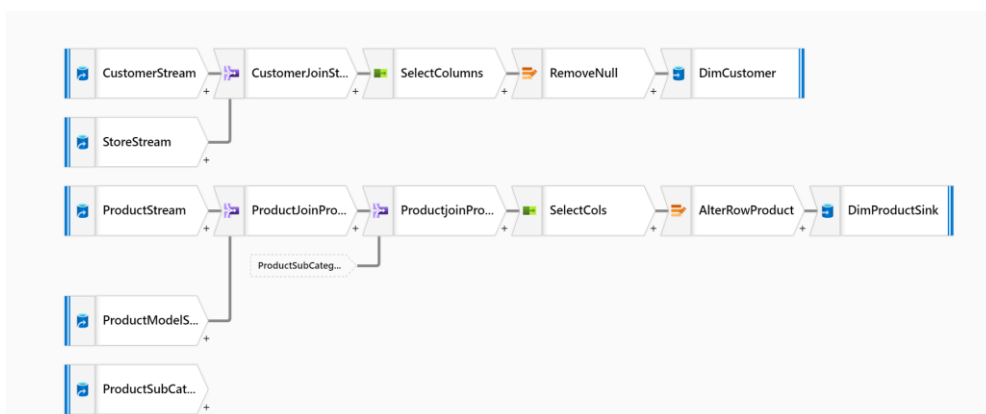
We will use the same mapping data flow. We used the following query to recreate the table. Delete the contents of the `DimProduct` and `DimCustomer` as we would be modifying the existing pipeline..

```
SELECT prod.[ProductID],prod.[Name] as ProductName,prod.[SafetyStockLevel],model.[ProductModelID],model.[Name] as ProductModelName,category.[ProductSubcategoryID],category.[Name] AS ProductSubCategoryName
FROM [Production].[Product] prod
LEFT JOIN [Production].[ProductModel] model ON prod.[ProductModelID] = model.[ProductModelID]
LEFT JOIN [Production].[ProductSubcategory] category ON
prod.[ProductSubcategoryID]=category.[ProductSubcategoryID]
WHERE prod.[ProductModelID] IS NOT NULL
```

We go to the Author section in the ADF Studio, open the previous dataflow.

- We add a new source for our product table
- Add another source for the `ProductModel` table
- Create another source for `ProductSubcategory`
- Click on the plus button for the `Product` and select join, give the right stream as `ProductModel` stream, select left outer join, select the column join condition (`productModelID == productModelID`)
- Click on the plus button for the join again to do a left join with `productsubcategory`
- To the previous join, add a select columns card, remove the unwanted columns from the select card
- Add an alter row card, add the condition to delete columns where `ProductModelID` is null
- Add a sink card and add your Dimension table from the dedicated sql pool, check for the correct mapping
- Validate and Publish

This is how the pipeline looks for our dimension tables –



- Now go to the existing pipeline and directly trigger the pipeline as there are no changes in the pipeline.

## Generating Surrogate keys in ADF

We generated surrogate keys in Synapse dedicated sql pool using the identity function.

Go to you SSMS and drop the DimProduct table and recreate it without the identity column surrogate key column. Use the following query –

```
CREATE TABLE [dbo].[DimProduct](
    [ProductSK] [int] NOT NULL,
    [ProductID] [int] NOT NULL,
    [ProductModelID] [int] NOT NULL,
    [ProductSubcategoryID] [int] NOT NULL,
    [ProductName] varchar(50) NOT NULL,
    [SafetyStockLevel] [smallint] NOT NULL,
    [ProductModelName] varchar(50) NULL,
    [ProductSubCategoryName] varchar(50) NULL
)
```

Go to author in ADF, go to the data flow that we have created earlier.

- On the plus button after the AlterRowProduct, select Surrogate Key
- Give the key product name – ProductSK, leave the starting value as 1
- Go to the sink and in the Mapping tab, add one more fixed mapping for ProductSK
- Publish it
- Delete the contents of dimCustomer again and contents of DimProduct have already been deleted
- Trigger the pipeline

We see in the dimProduct table that the surrogate key is generated in succession.

## Using Cache Sink and using Lookup to access cached data

This feature is used to write data into the Spark cache and not in the Data store. The cache lookup can then be use to reference the data in the cache sink.

For example while adding data to a dimension table we have our surrogate key upto a certain number.

We can go to our dimProduct table and see the MAX(ProductSK) to see the maximum surrogate key value and then know the number from where the next number should start.

In ADF go to your Data flow previously created, enable Data flow debug.

- Now what we need to first do is get the maximum surrogate key and then let the Surrogate Key card take it from there
- For that we need to add another source which would be our dimProduct table, add it from Azure Synapse
- Go to the source options for the card, select the query option and give

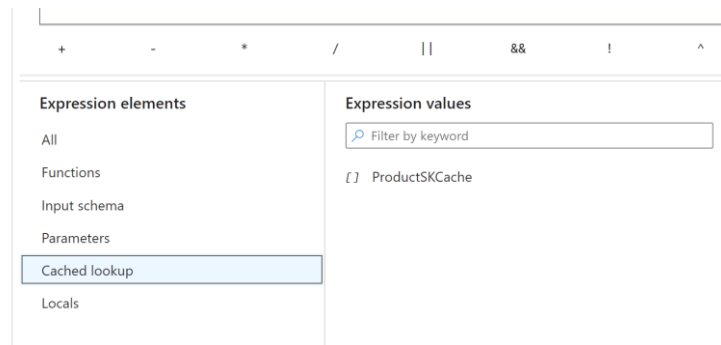
```
SELECT MAX ( [ProductSK] ) as ProductSK from [dbo].[DimProduct]
```

- Go to the Projections tab and import the projects again and we would only have one column
- If we go to the data preview tab and refresh, we see

Source settings	Source options	Projection	Optimize	Inspect	Data preview <span>●</span>
Number of rows <span>+</span> INSERT 1 <span>+</span> UPDATE 0					
<span>↻</span> Refresh <span>▼</span> Typecast <span>▼</span> <span>⚙</span> Modify <span>▼</span> <span>📄</span> Map drifted <span>📊</span> Statistics <span>✕</span> Remove					
↕ ProductSK 123					
+ 504					

- Now we need to add this value to a sink but instead of destination in the sink we select cache
- So add the sink card, give it a name and select Cahce in the sink type, in the mapping check the mapping

- Now go to your Surrogate key and click the + button and select derived column. Select the ProductSK column and go to the Expression Builder. In the expression builder in the cached lookup we can see –



- Enter the following expression - `ProductSK + ProductSKCache#outputs()[1].ProductSK`
- This means take the value of ProductSK (which will be 1) and keep adding it with what's in the first row of the cached data.
- Click on refresh in the expression builder to see the output and click on save and finish

Data preview <span>Refresh</span>	
Output: ProductSK <sup>123</sup>	ProductSK <sup>123</sup>
505	1
506	2
507	3

- The sink would remain unchanged. Publish the changes.
- Trigger the pipeline to see the ProductSK column add data from the previous max value.

Going to the dimProduct table we can see the surrogate key has added new value after the previous max value.

### Ensuring Duplicate Values are not part of the Data Flow

Begin by deleting the first set of duplicate rows from the DimProduct table using following query –

```
delete from [dbo].[DimProduct] where [ProductSK] > 504
```

Next we check the existing ProductID and their Counts in the table. We see that all the ProductIDs are unique and are not repeat. Let's delete all the data where the ProductID is greater than 900 so that when we're entering the data again, the data would get entered but the existing data would not be entered since it would be duplicate data.

```
delete from dimproduct where productid > 900
```

To make sure the duplicate values are not entered, we have to add another source, in our data flow. Open ADF Studio and open the previously made data flow –

- Add the DimProduct table as a source
- After the Alter Row card of the Product table stream, add a new exists operation card.
- In the exists stream, the left stream would be the data coming in to the table and the right stream would be the original source that we added for the DimProduct table. Select the Doesn't Exist button to tell the flow to only accept the rows that don't exist based on the condition that `ProductID == ProductID`
- Leave everything as is. Publish everything.
- Run the pipeline.

Now we can see the DimProduct table has values with ProductID > 900 but no duplicate values.



## Issues that may arise with the Surrogate key if the existing table is empty

In the derived column card we had used an expression to generate surrogate keys from existing surrogate keys. If the table is empty, then it would give use null values for the expression that we used which was –

```
ProductSK + ProductSKCache#outputs()[1].ProductSK
```

But this is taking the max value in the cache storage. IF it is null we need to change the expression a bit.

```
ProductSK + (ifNull(ProductSKCache#outputs()[1].ProductSK),0)
```

The above expression would ensure that if the cached value is null then it would just take 0.

## Using Mapping Data Flow to transform DateTime to Formatted values

We would be using the original Log.csv file for it which we used before using the cleaned version.

Go to your ADF and create a new dataflow –

- Name the data flow in the right and enable data flow debug
- Add a new source, Add a new connection > Azure Data lake > Delimited Text file > Select your original Log File
- Now we want to add a derived column. Name it and then in columns, select the Time column and open the expression builder for it and enter the following expression

```
toTimestamp(concat(substring(Time,0,10),' ',substring(Time,12,8)), 'yyyy-MM-dd HH:mm:ss')
```

- In the above expression the Time column is being inferred as a string so we are getting the first substring to get the date and the second substring to get the time from the string. Hit save and finish.
- Now if we open the expression builder again and hit refresh we would see the input Time column that we are getting and the output that we are generating for that.
- Now add a sink to the derived column card and choose Data Lake Gen2. Select the folder where you want the cleaned log file, select first row as header and none for import schema.
- Now go to setting and in the file name option, select Output to a single file. Set it to a single partition when the warning message shows up. Give the name of the file.
- Publish everything
- Create a new pipeline, add the above data flow to it. Publish and Trigger.

You can now see the cleaned Log file show up in your Data Lake.

## Loading a JSON file onto Azure Synapse

Go to your ADF and create a new pipeline –

- Drag and Drop the Copy Data card on to the canvas. Add the log.json file that we have in our data lake on to here as the source.
- In the Sink, create a new dataset, select Azure Synapse, and select your log.json file
- Go to the mapping click import schemas to see the mapping
- Go to settings to give the staging location
- Delete the contents of your logdata table
- Publish and trigger the pipeline

Data should show up in the logdata table.

## Self-Hosted Integration Runtime

If we have a data source that could be hosted on a virtual machine, this machine could be on azure or could be hosted in your on-premises infrastructure. Let's say you want to move data files or a sql database that is hosted on this machine onto your Azure Synapse. To do so, you would install the Integration Runtime on this machine and register this server with Azure Data Factory.

To install nginx, begin by creating a virtual machine in Azure –

- Click on the virtual machines section, click on create > new virtual machine
- Select your resource group, give the name for the vm, in availability select no infrastructure redundancy required, select image as Windows server 2019 Datacenter – Gen1, size as standard, give the username and password
- In the inbound ports, also select port 80 because we would be installing the nginx webserver on this machine and that listens on port 80.
- Click next and leave everything as is.
- In the Networking section you would see Virtual Network. Our machines need to be part of a virtual network. This virtual network would be created for us by Azure. It would also create a subnet for our machine and also a public ip for our machine.
- Review and create
- Once the resource is created, connect to the resource using RDP. Download the RDP file. Should open up automatically upon opening in a windows machine. Click connect and enter your credentials.
- To install nginx go to local server in the Server Manager that is open, click on the IE Enhance Security Configuration on the right hand menu in the middle – turn it off to download it from internet explorer.
- Open up internet explorer – enter url – <http://nginx.org/en/download.html>
- Download the stable version, extract the files and store them
- Run the command prompt as administrator and go to the folder location to run nginx.exe
- Now in the logs folder you would be able to see an access and log file meaning nginx is running
- If we type localhost in internet explorer, we would be able to see the Nginx page.

***Now let's say we want to send the log data from this machine on to a table in synapse (access.log -> synapse)***

We can use ADF for this process. We can have an activity in place to copy the log data on to a table. For this we use the self-hosted integration run time. Go to ADF and click on manage from the left and click on integration runtime –

- Create a new runtime, select Azure Self Hosted > Self Hosted. Give a name and create
- Going through the instructions, you can see how to set up integration runtime
- Go to Option 2 > download and install integration runtime using the link on your virtual machine

Integration runtime setup

Settings Nodes Auto update Sharing Links

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

Name ⓘ

runtime203

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](#)

Step 2: Use this key to register your integration runtime

Name	Authentication key
Key1	IR@4f0972f1-48e6-48d9-87d5-a1fa501df28a@datafactory203@ServiceEn
Key2	IR@4f0972f1-48e6-48d9-87d5-a1fa501df28a@datafactory203@ServiceEn

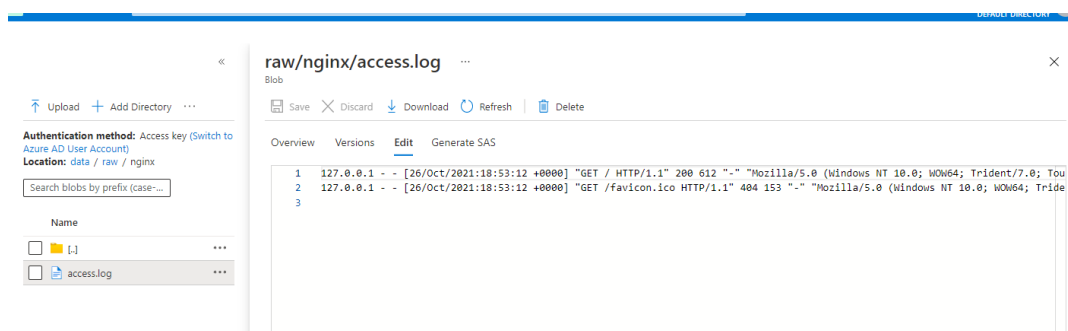
- Install it from the file downloaded.
- Add the authentication key in the text box that showed up and click on register and finish.
- On ADF in the integration runtime, you can see the integration runtime show up.

## Copying the access.log file into our Data Lake Gen2

Go to ADF and create a new pipeline.

- Drag and drop the Copy Data activity on to the canvas also create a new folder in your raw folder of the data lake for this
- Name the copy data activity
- Go to the source tab > click new > File tab > File System > Binary File
- Name the dataset and create a new link service
- Name the connection and select your vm from the connection via integration runtime option. Give the location of the folder where the file is in *host* and enter your username and password. Test the connection and create.
- On returning to the previous screen browse to your access.log file and click okay.
- Go to the sink and select a new dataset. Select azure data lake gen2 > binary.
- Give the name and linked service (your data lake) and browse to the folder where to store this file and enter the file name in the last text box.
- Publish and trigger.

We see the access.log file was copied onto the data lake.



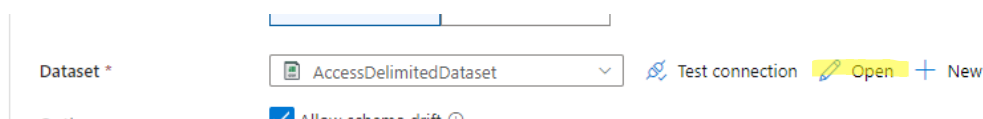
There are two reasons as to why we copy and then ingest into the Synapse sql pool –

- The Data lake is supposed to be a central place for all the file
- In mapping data flow in ADF there is no way to take a file from a virtual machine

## Mapping Data Flow to move file from Data Lake to Synapse

Go to ADF create a new data flow and enable data flow.

- Add a new source, add a new source. Name it and select Azure Data Lake Gen2 by clicking new. Select Delimited Text file and select your file. Click none on import schema.
- Upon clicking on data preview we can see that all the data is just showing up under one column. We can see the data for different column can be obtained by breaking down the data based upon the space
- So we edit the source dataset again



- Give the column delimiter as a single space after checking the edit box.
- Now go to data preview and hit refresh again. Now we some improvements. Out of this, we can use the columns that we want.

_col3_ abc	_col4_ abc	_col5_ abc	_col6_ abc
[26/Oct/2021:18:53:12	+0000]	GET / HTTP/1.1	200
[26/Oct/2021:18:53:12	+0000]	GET /favicon.ico HTTP/1.1	404

- Go to the projection tab and select import projection and all the columns would load up.
- Add a Select card to the source card and give the following mapping –

<input type="checkbox"/>	abc_col0_	→	remote_addr	+
<input type="checkbox"/>	abc_col3_	→	time_local	+
<input type="checkbox"/>	abc_col5_	→	request	+
<input type="checkbox"/>	12s_col6_	→	status	+
<input type="checkbox"/>	12s_col7_	→	bytes	+
<input type="checkbox"/>	abc_col8_	→	remote_user	+
<input type="checkbox"/>	abc_col9_	→	http_user_agent	+

- In the data preview for select card we can see the columns

↑↓	remote_addr	abc	time_local	abc	request	abc	status	12s
+	127.0.0.1		[26/Oct/2021:18:53:12		GET / HTTP/1.1		200	
+	127.0.0.1		[26/Oct/2021:18:53:12		GET /favicon.ico HTTP/1.1		404	

- Now we use a derived column card to fix the time\_local column
- Select the time\_local column and open the expression builder. In the expression builder click on the functions tab select substring function and then input the column time local to create the expression - `substring(time_local,2,20)`
- Click on refresh to see the result of this – the square bracket at the beginning is removed

Output: time_local	abc	time_local	abc
26/Oct/2021:18:53:12		[26/Oct/2021:18:53:12	
26/Oct/2021:18:53:12		[26/Oct/2021:18:53:12	

- Save and finish.
- Add bytes and status columns and convert them to integer format –

<input type="checkbox"/> Column	Expression
<input type="checkbox"/> time_local	substring(time_local,2,20) abc +
<input type="checkbox"/> status	toInteger(status) 123 +
<input type="checkbox"/> bytes	toInteger(bytes) 123 +

- Create the server logs table in Synapse sql pool with the query

```
CREATE TABLE [serverlogs]
(
[remote_addr] varchar(20), [time_local] varchar(100), [request] varchar(200), [status] int, [bytes] int,
[remote_user] varchar(100),
[http_user_agent] varchar(500)
)
```

- Add a sink to the derived column card and select Synapse > serverlog table
- Check the mapping, check the data preview.
- Publish if everything looks good
- Go to the pipeline which was used to copy access log to the data lake and add the above data flow, add a staging service as well
- Connect the two activities together
- Publish everything.
- Delete the access.log file from the data lake and trigger the pipeline

After the pipeline is run we would be able to see the file in the data lake and data in synapse.

100 %							
Results							
	remote_addr	time_local	request	status	bytes	remote_user	http_user_agent
1	127.0.0.1	26/Oct/2021:18:53:12	GET / HTTP/1.1	200	612	-	Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7....
2	127.0.0.1	26/Oct/2021:18:53:12	GET /favicon.ico HTTP/1.1	404	153	-	Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7....

## Mapping Data Flow that uses Flatten Transformation for JSON files

Begin by creating a new directory – customer in the raw folder of your Data Lake Storage. Upload [this](#) file to it.

When you open the customer.json file you would see the courses field has multiple courses for each customer. When this data would be converted into the tabular format, we would want one course per row. This is where the flattening operation comes in.

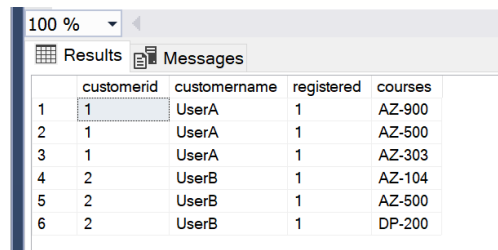
Create the customer table using the query –

```
CREATE TABLE [customercourse]
(
[customerid] int, [customername] varchar(200),[registered] BIT, [courses] varchar(200)
)
```

Begin by going to ADF and creating a new data flow.

- Name the data flow
- Add a new source which would be the json file that we just uploaded
- In the source options, because our json objects are in square brackets meaning they're in an array, in the source options select Array of documents in the document form
- Next we add the flatten card. Name the card. Select the json object to unroll by (in our case "Course"). Unroll root is again "Courses" itself.
- Add a sink card and select the customercourse table create on the synapse pool.
- Go to mapping to see the mapping
- Publish everything
- Create a new pipeline, add the above data flow, give a staging area.
- Publish and trigger pipeline

Data is loaded as expected –



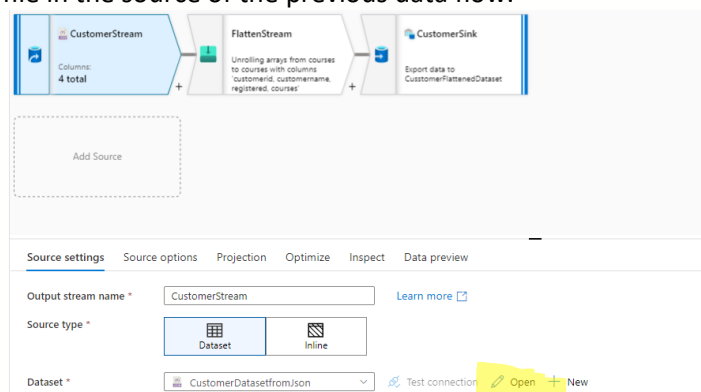
	customerid	customername	registered	courses
1	1	UserA	1	AZ-900
2	1	UserA	1	AZ-500
3	1	UserA	1	AZ-303
4	2	UserB	1	AZ-104
5	2	UserB	1	AZ-500
6	2	UserB	1	DP-200

*Now if our json file has a List of objects inside an object. Eg –*

```
"courses":["AZ-104","AZ-500","DP-200"],
"details":
{
  "mobile":"333-1112",
  "city":"CityB"
}
```

Then there would be a slightly different way to handle it. Upload [this](#) file to the data lake.

- Edit the source to reflect this file in the source of the previous data flow.



- In the same edit screen, open the schema and click import schema from connection. It would detect that under details we have mobile and city objects.
- Now in the flatten card reset the input columns and add the columns manually. You would be able to add the nested objects as well.

<input type="checkbox"/>	abc details.mobile	→	mobile	+	🗑
<input type="checkbox"/>	abc details.city	→	city	+	🗑

- Drop the table in Synase and recreate it with the mobile and city columns.
- Now in the sink, open the dataset and select import schema from the schema tab.

Sink
Settings
Mapping
Optimize
Inspect
Data preview

Output stream name \*
CustomerSink
Learn more

Incoming stream \*
FlattenStream

Sink type \*

Dataset
Inline
Cache

Dataset \*
CusustomerFlattenedDataset
Test connection
Open
New

Options
☒ Allow schema drift ⓘ
☐ Validate schema ⓘ

- Now go to the mapping tab and click reset.
- Publish everything and trigger the earlier pipeline.

Data is loaded as expected –

	customerid	customername	registered	courses	mobile	city
1	1	UserA	1	AZ-900	111-1112	CityA
2	1	UserA	1	AZ-500	111-1112	CityA
3	1	UserA	1	AZ-303	111-1112	CityA
4	2	UserB	1	AZ-104	333-1112	CityB
5	2	UserB	1	AZ-500	333-1112	CityB
6	2	UserB	1	DP-200	333-1112	CityB

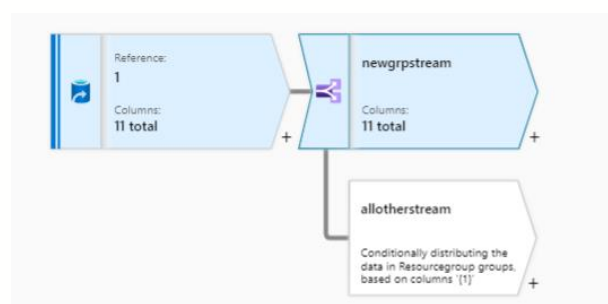
**Using Conditional Split in Data flow** – Used to split the output streams based on different conditions

We would be using our Log.csv file for this. In the Resource group of our Log file if we want the rows of the new-grp resource group we can use conditional split.

Begin by creating a new data flow.

- Add the source to the log.csv file in the data lake
- Delete the contents of the log data table in Synapse
- Now add a new conditional split card after the source.
- Name it, Under split condition name the first stream and give the condition - {Resourcegroup}=='new-grp'
- Name the other stream for all the other rows that do not meet this condition

Now we have two different streams based on what the ResourceGroup value is –



- Add a sink to the new-grp stream and select the logdata table in synapse
- Publish the changes
- Create pipeline with the above data flow , give a staging area, publish and trigger.

We can now see data with only new-grp resource group in the logdata table on synapse.