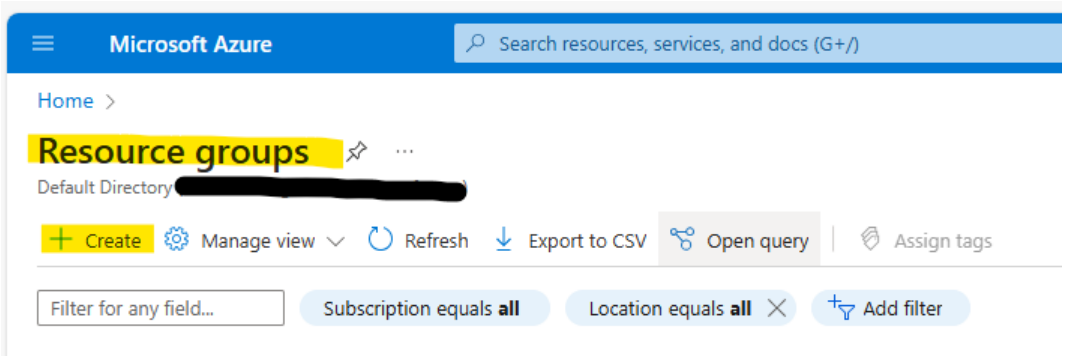
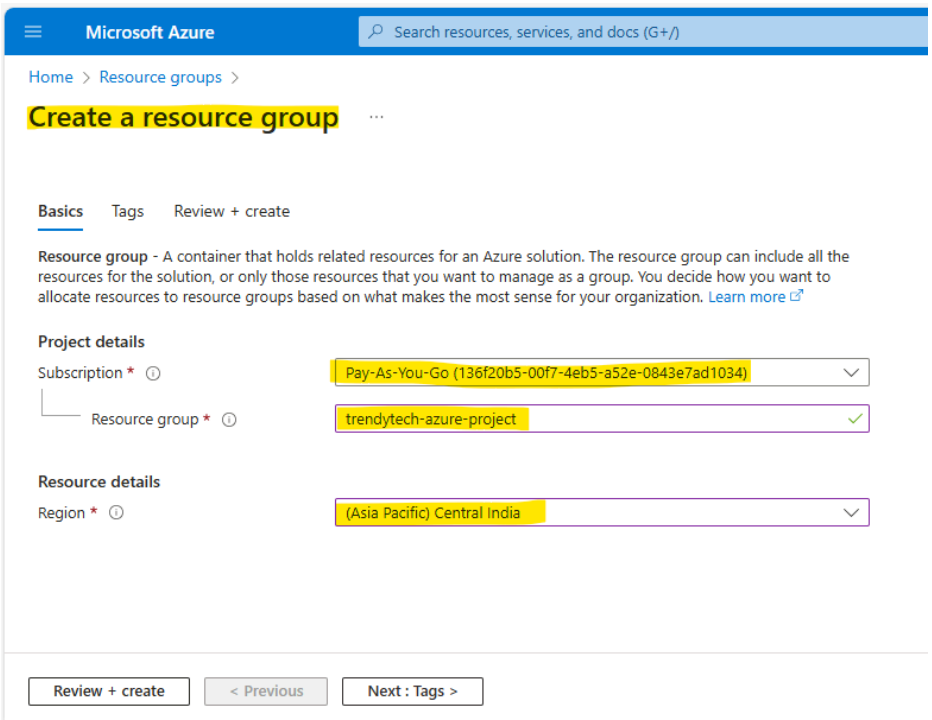
Part 1:

Before creating all the resources we will create the resource group in which we will create all the required resources.

Go to Azure Portal and log in with your Azure account. In the left-hand menu, select "Resource groups". If you don't see it, use the search bar at the top of the page and search for "Resource groups". Click the "+ Create" button or "Add" at the top of the Resource groups page.



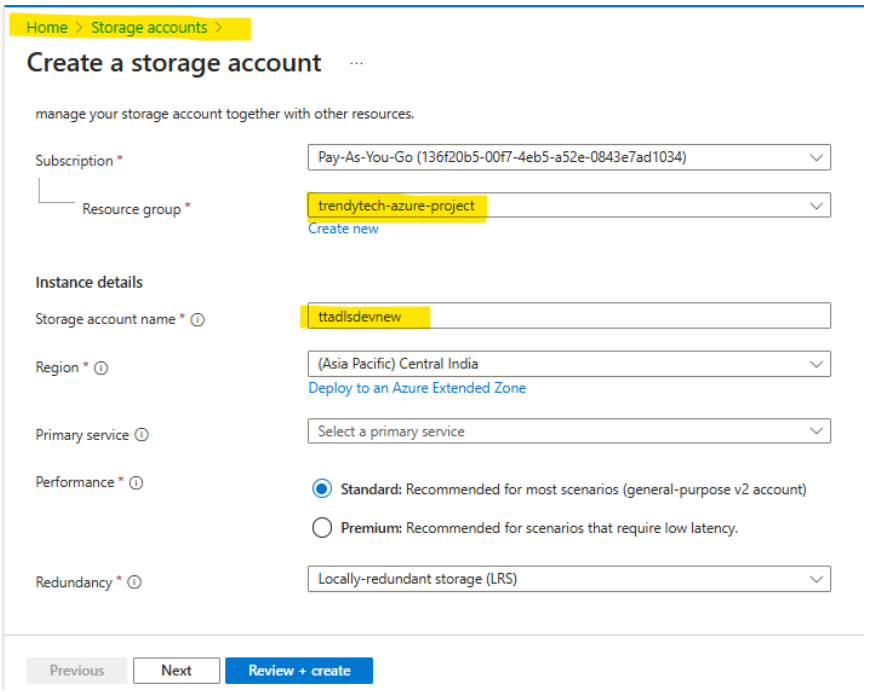
Select the subscription under which the resource group will be created. Enter a unique name for your resource group. Choose a location (region) where your resources will reside (e.g., East US, West Europe).



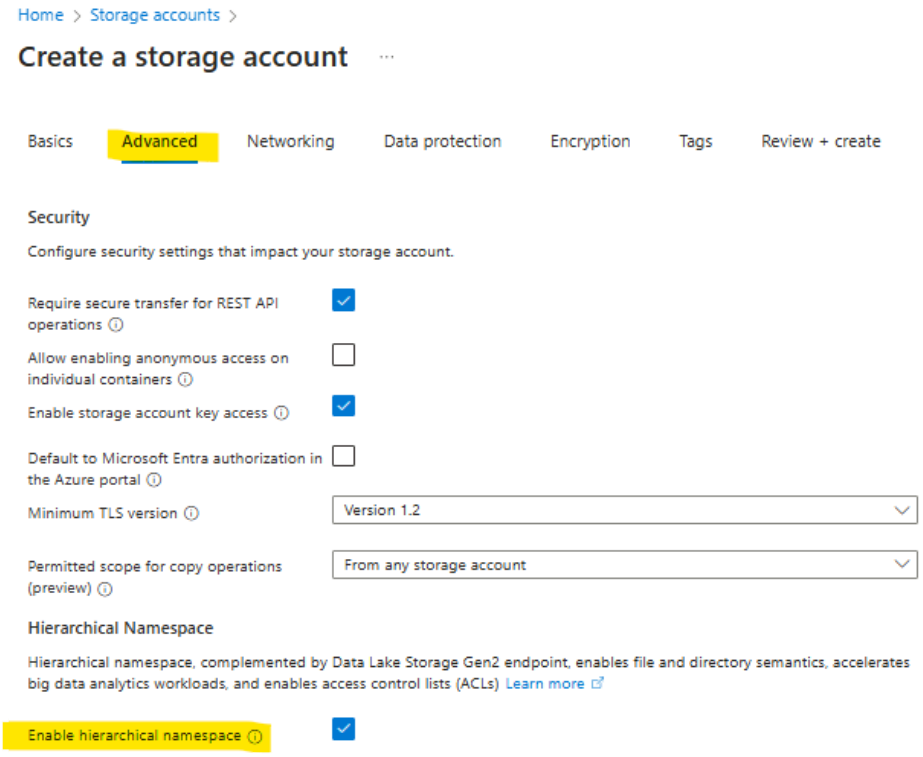
Click "Review + Create" and then "Create".

Azure Storage Account creation:

Search for "Storage accounts" and click "+ Create". Select subscription, resource group, region, and enter a unique storage account name as shown below.

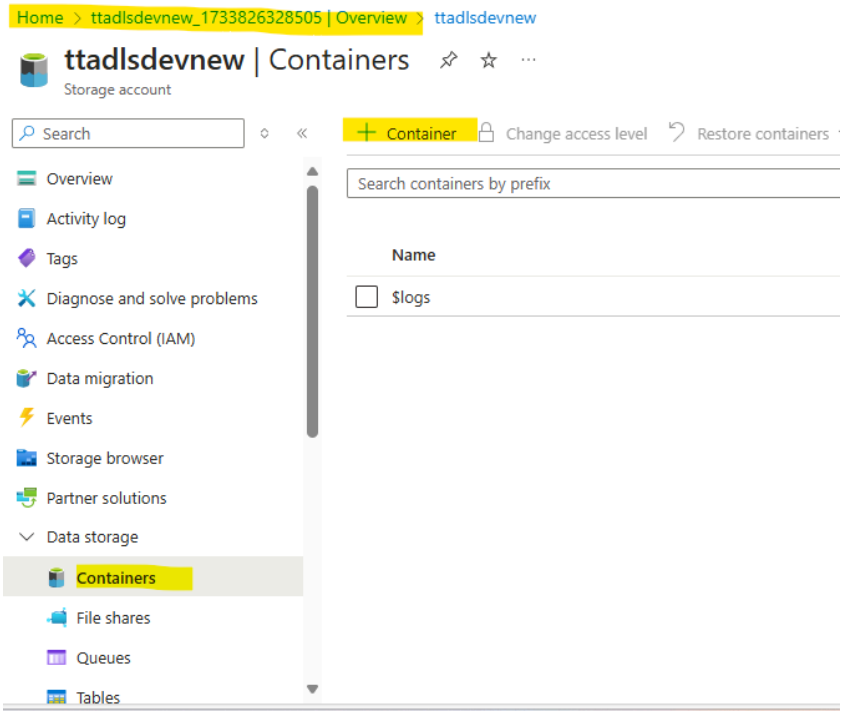


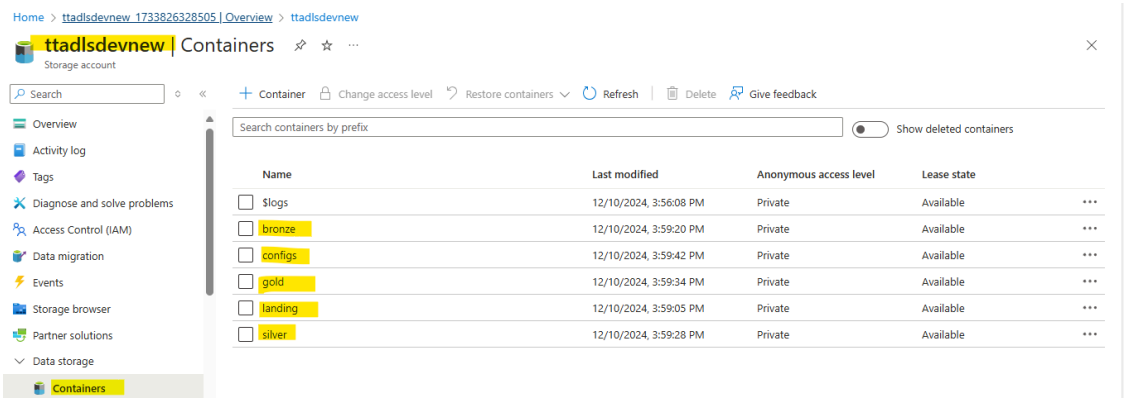
Enable ADLS Gen2: Go to the Advanced tab and enable Hierarchical namespace.



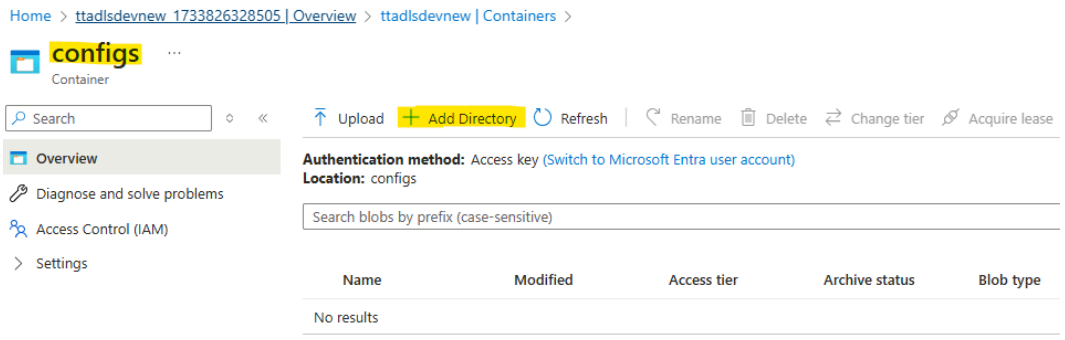
Click Review + create, validate settings, and click Create.

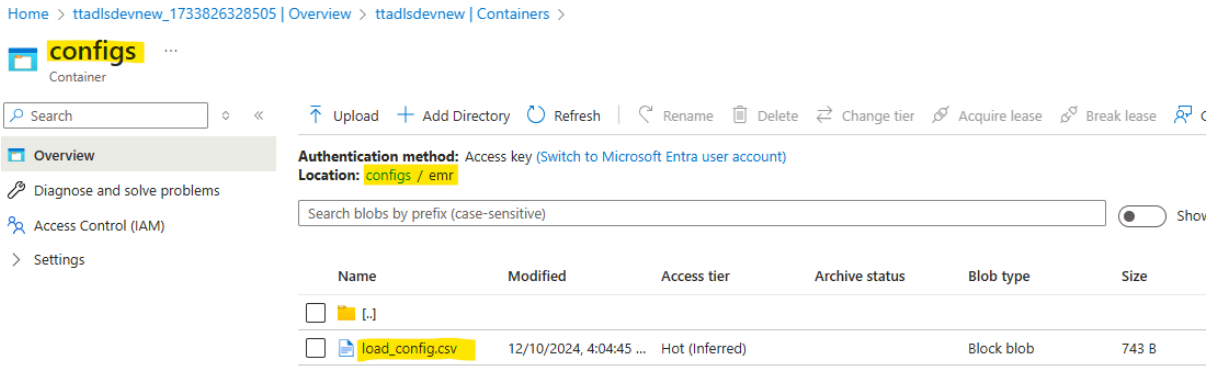
Now create the containers “landing”, “bronze”, “silver”, “gold”, “configs” in this storage account as shown below.





Then in the configs container create the directory “emr” and then upload the file “load_config.csv” in it.

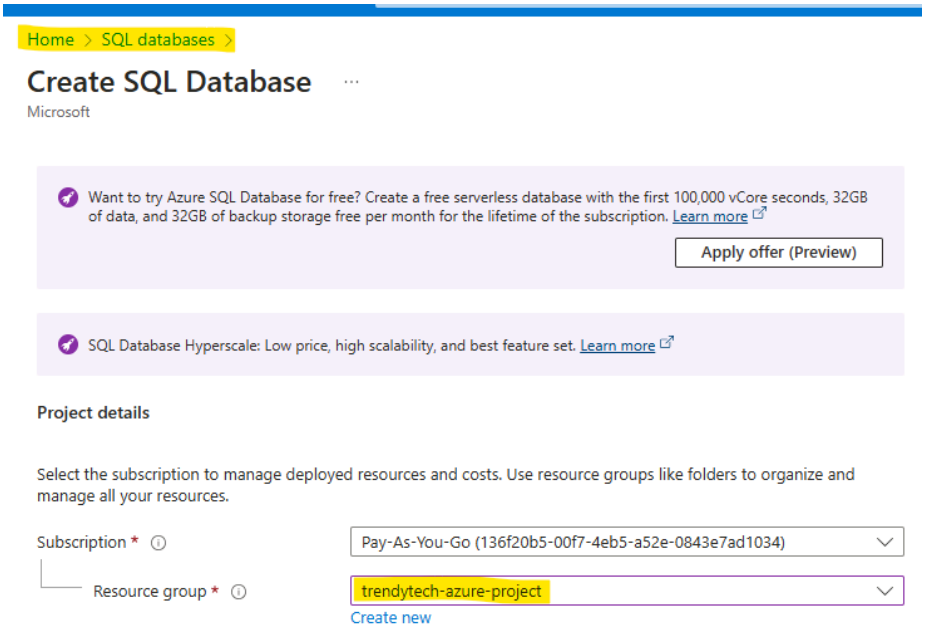


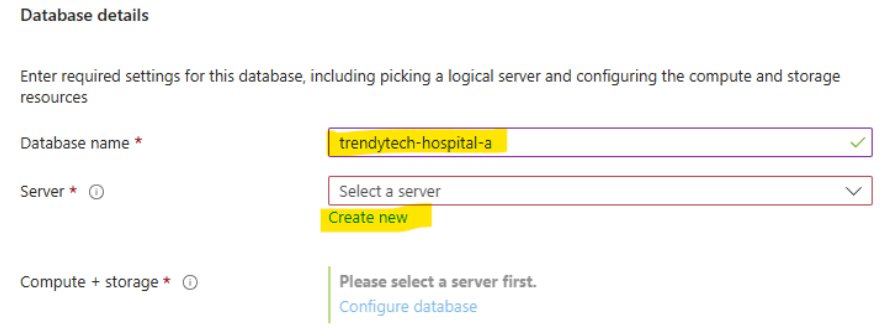


Steps to create Azure SQL database:

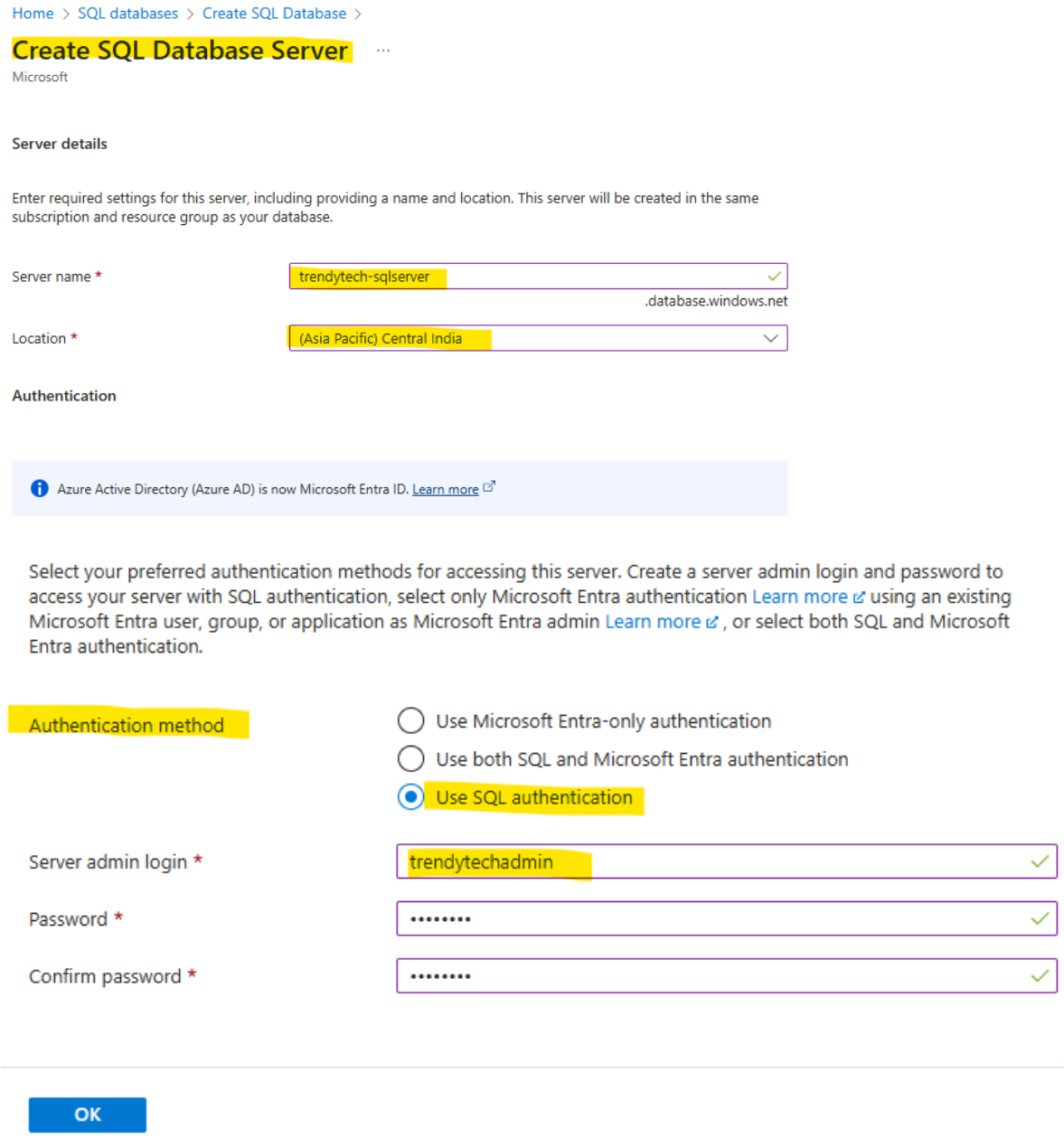
We will create 2 azure SQL db - trendytech-hospital-a, trendytech-hospital-b

In the search bar, type "SQL Database" and select "SQL Database" from the results. Choose your Subscription and Resource Group. Enter a Database Name. Also create a SQL Server.

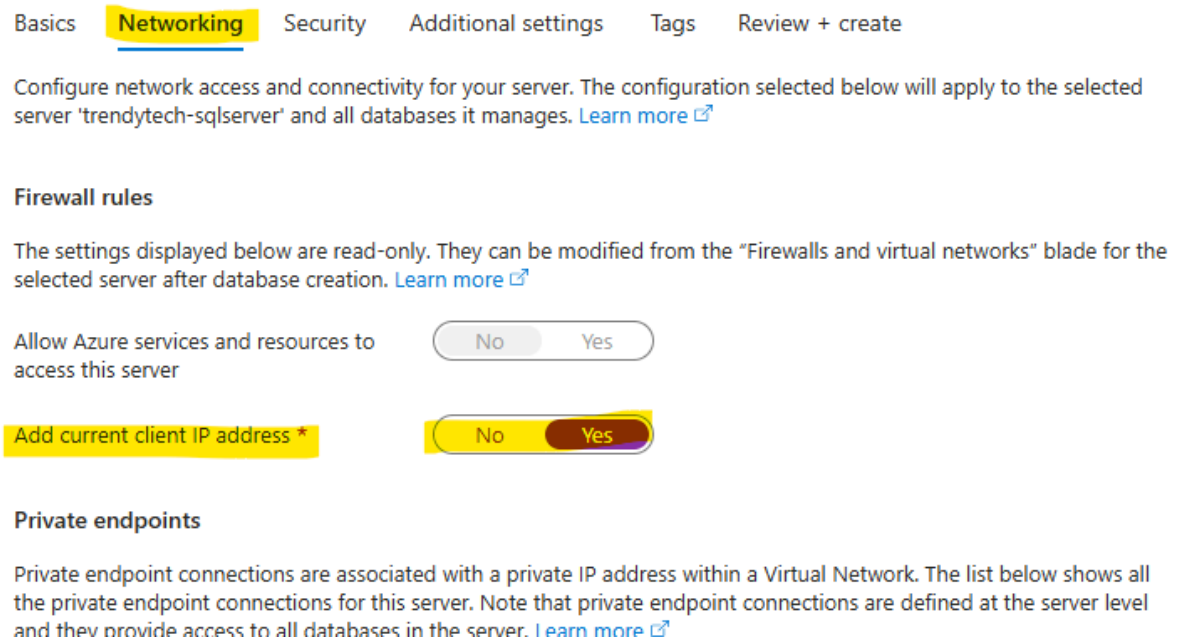




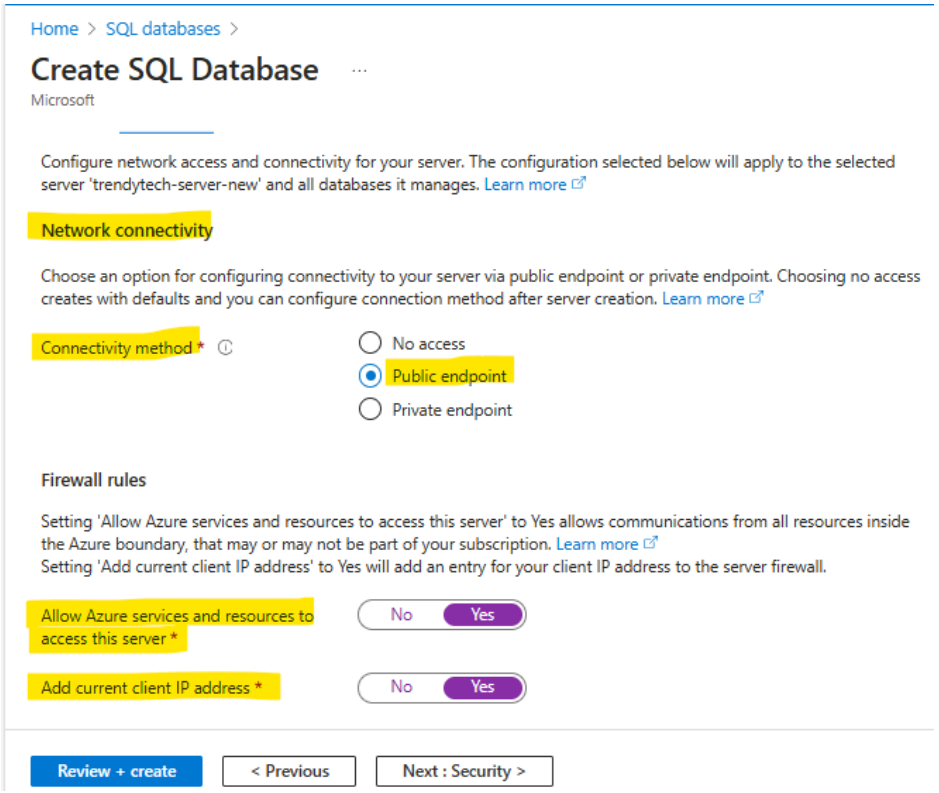
We will create the server as shown below. Choose a Compute + Storage tier (e.g., Basic, General Purpose).



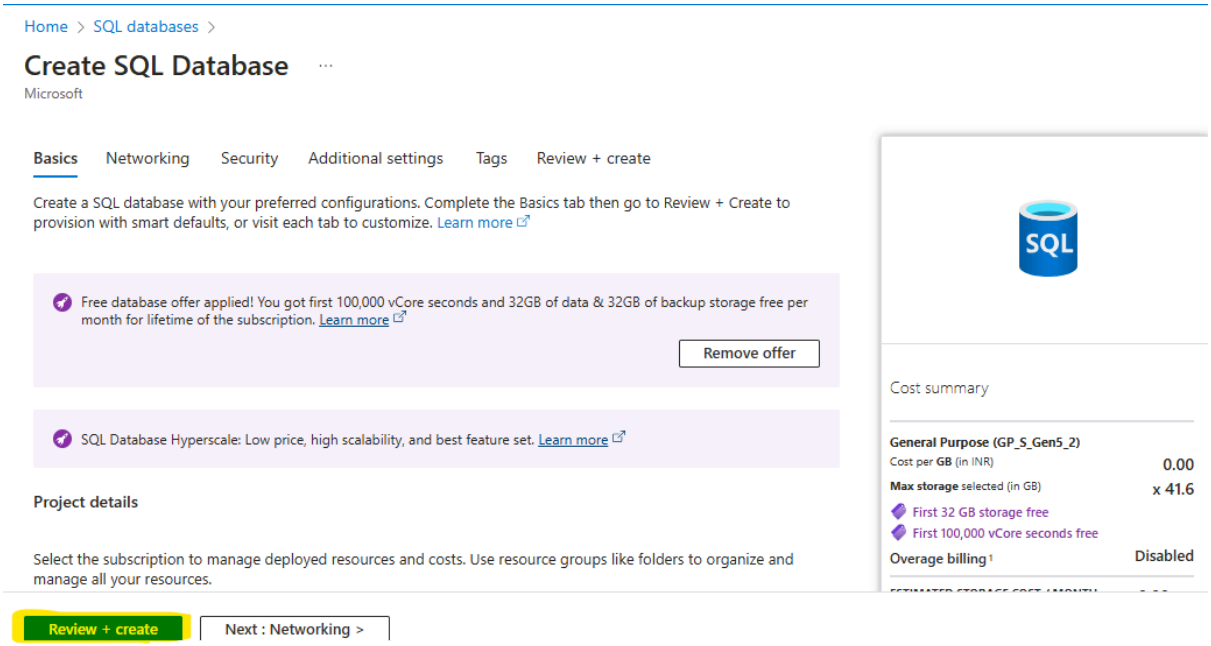
After this Go to Networking => In Network connectivity select “Public endpoint” option. Also set yes for the options “Allow Azure services and resources to access this server” and “Add current client IP address”



Note: Please note down this username and password for future reference.

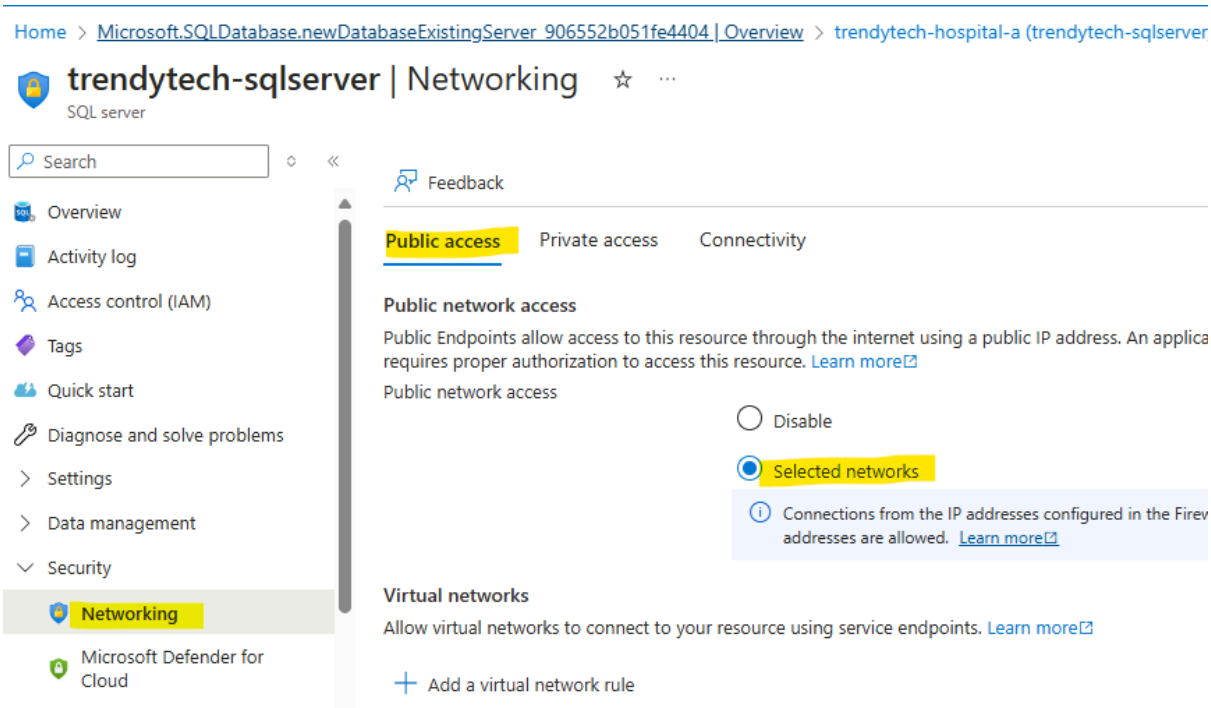


Click Review + Create, validate the details, and then click Create as shown below.

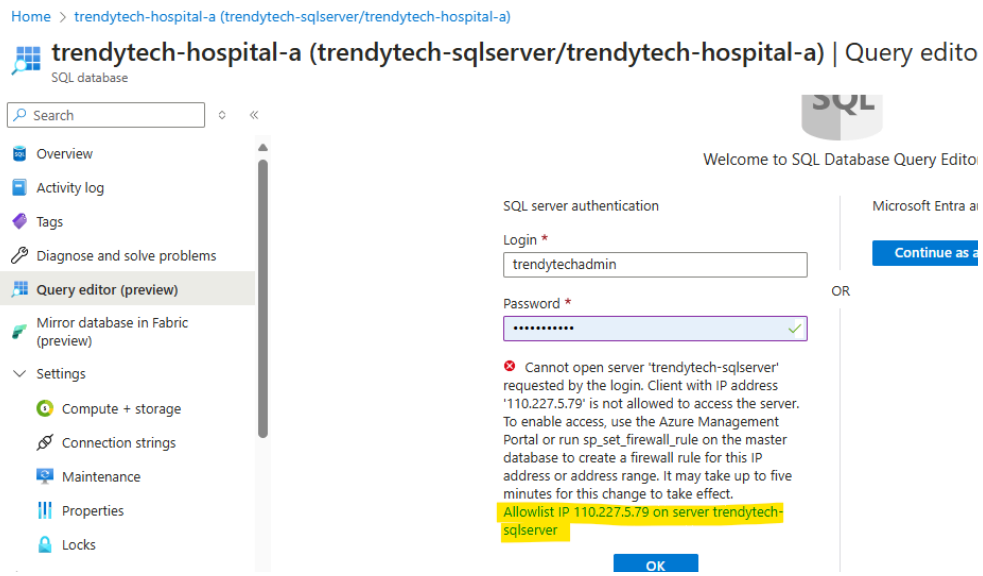


Note: While creating database if you are not able to allow public access and add client ip address you can follow below steps:

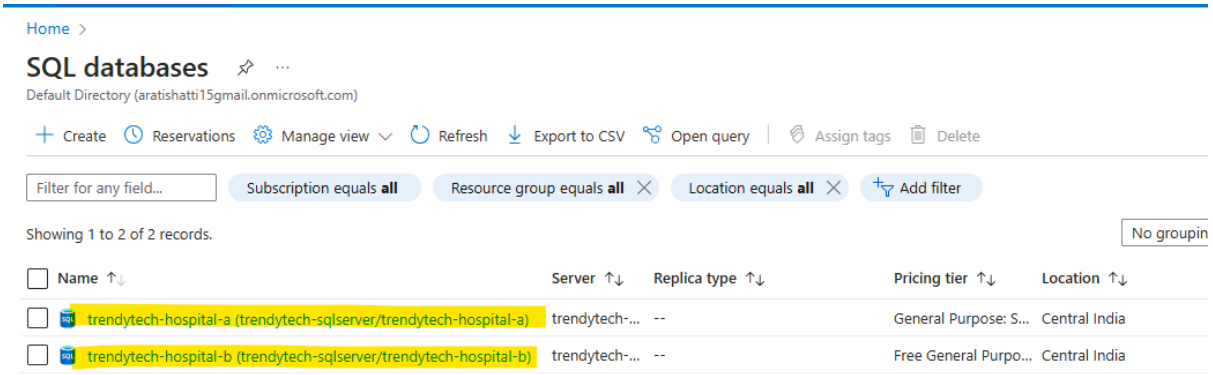
After creating this database Go to Networking => For Public access (select option Selected networks and save this) =>



Also while using query editor if you face below error click on “Allowing IP for current ip address” as shown below



Similarly we will create another database trendytech-hospital-b(We will use the same server i.e trendytech-sqlserver that we have created while creating trendytech-hospital-a database). Thus we have created 2 databases as shown below.

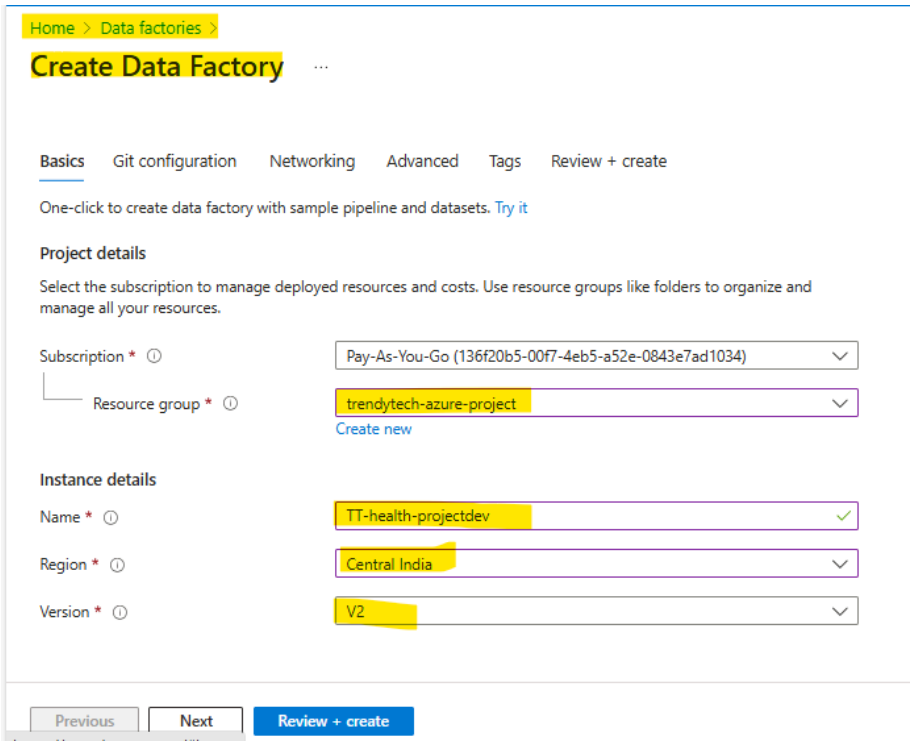


Then we will create the tables in these databases and for creating tables in the database use below scripts which are present on github account:
For trendytech-hospital-a =>
Trendytech_hospital_A_table_creation_commands

For trendytech-hospital-b =>
Trendytech_hospital_B_table_creation_commands

Steps to create ADF:

In the search bar, type "Data Factory" and select "Data Factory" from the results. Click the "Create" button on the Data Factory page. Provide a globally unique name for your Data Factory instance. Choose V2 (Data Factory Version 2) for the latest features.



Click "Review + Create" to validate the details. If validation passes, click "Create" to deploy the Data Factory.

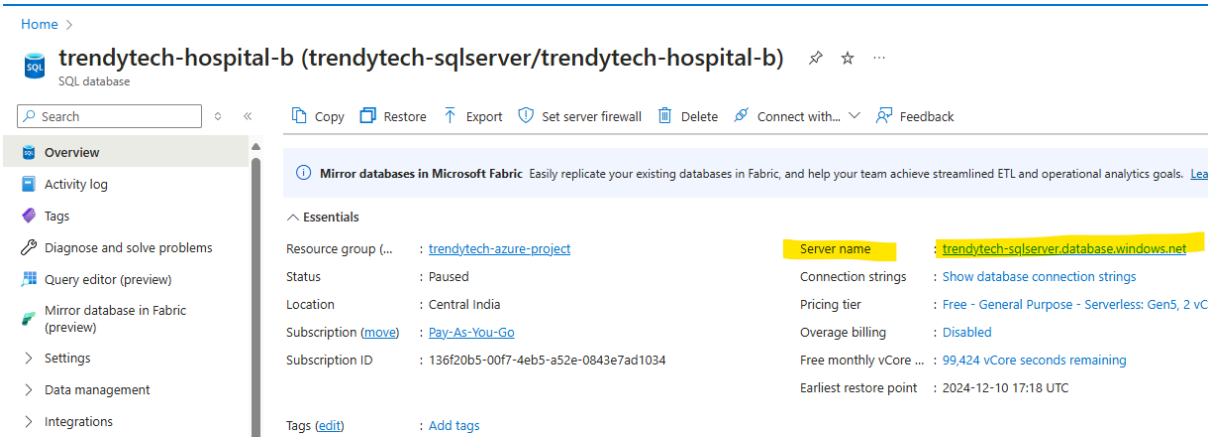
Steps to create ADF pipeline:

Linked Services creation:

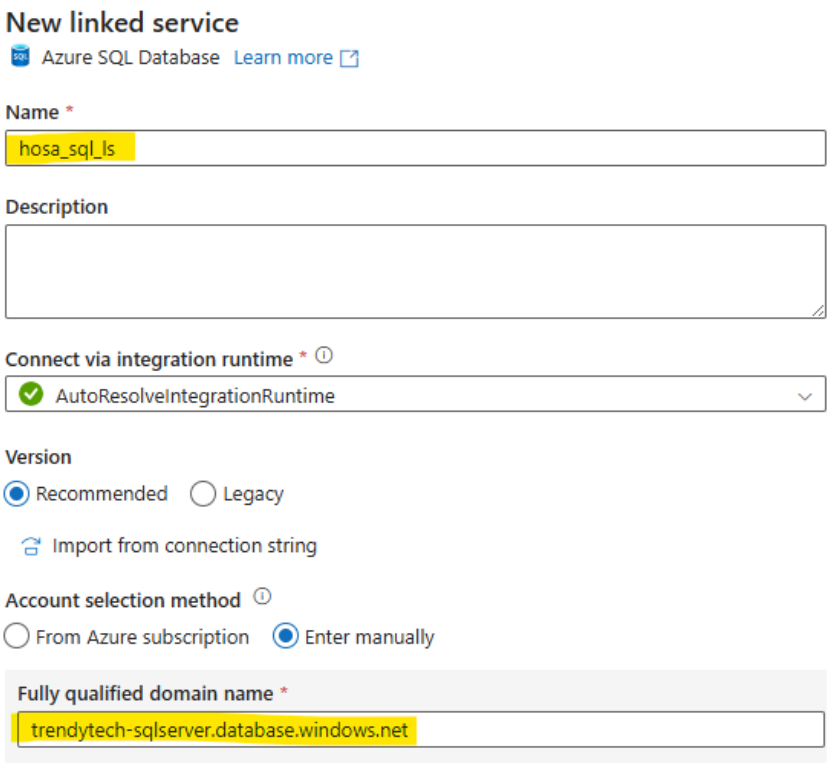
In the ADF interface, go to the Manage section on the left-hand panel. Under the Connections section, select Linked Services. Click on New to create a new Linked Service.

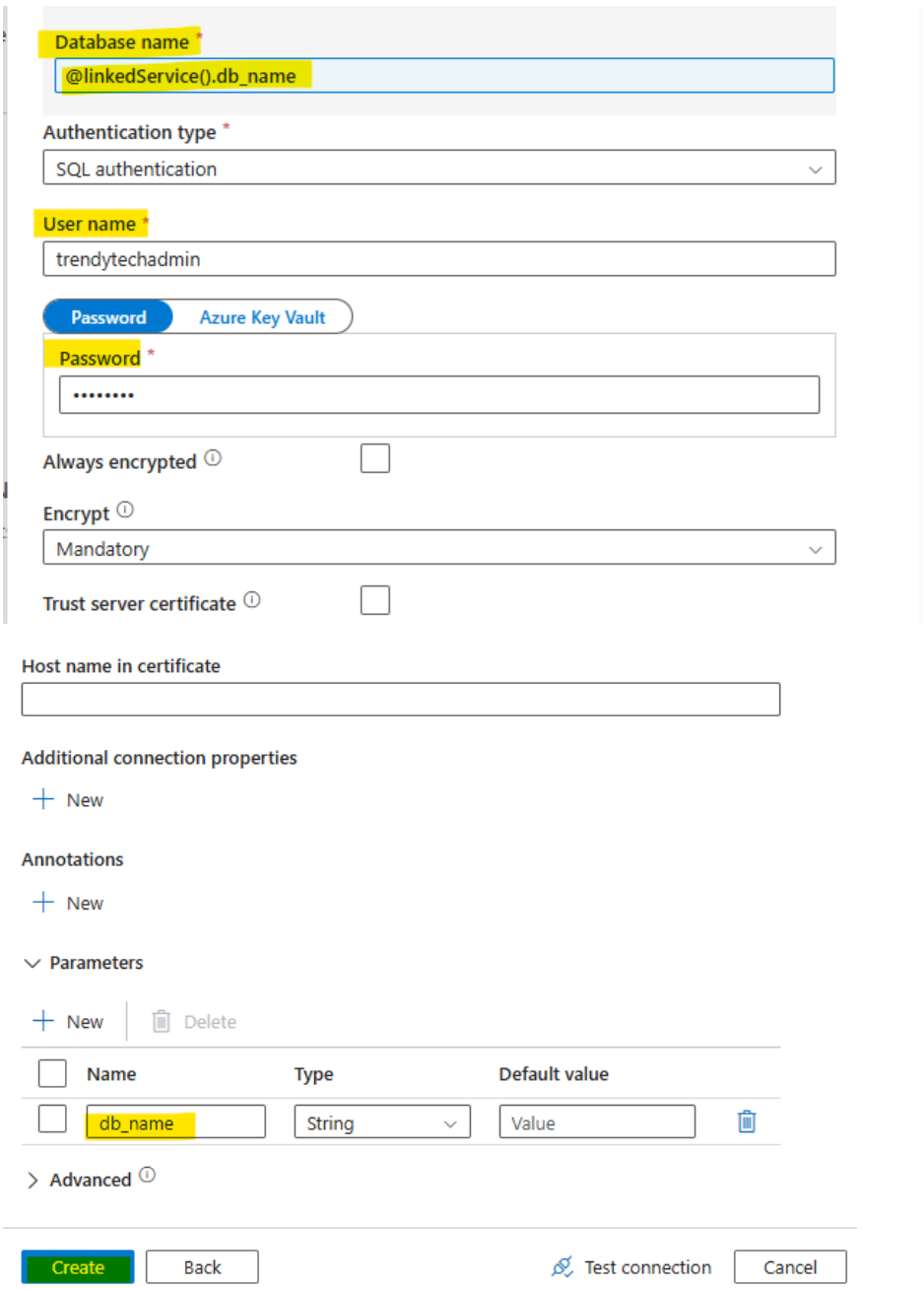
1. Azure SQL DB

Note down the server name for that sql database that we have created.



In fully qualified domain names, mention the server name, mention the username and password for sql server and define the parameter db_name and using this parameter we will pass the database name as shown below.



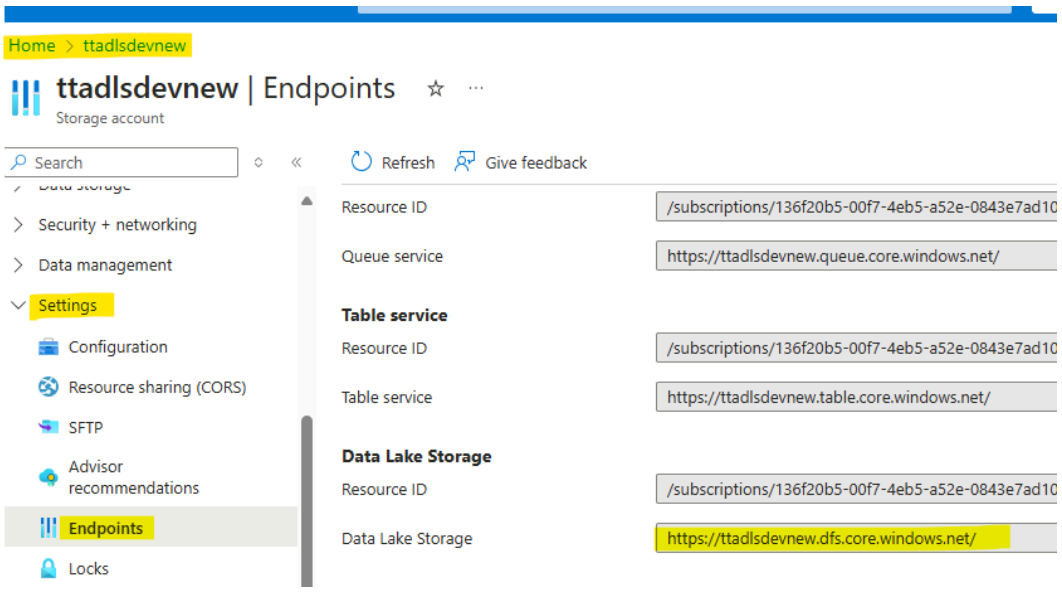


And click on create to create the linked service.

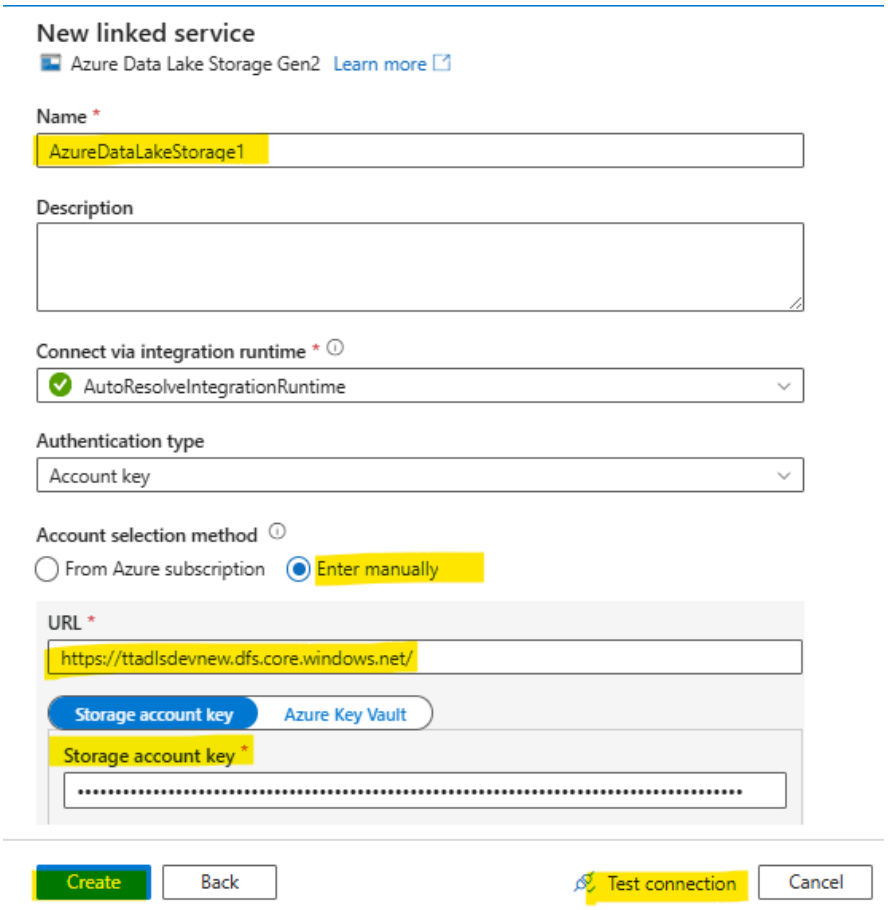
2. ADLS GEN2

Select Azure Data Lake Storage1 as the data store. Provide the following details- Name of your Blob Storage account, Authentication. Then click Test Connection to verify and save the Linked Service.

To get the url for Azure Data Lake Storage go to Adls gen2 storage that we have created => Setting => Endpoints => and copy the URL as shown below



Also copy the access key. Using these details create the linked service as shown below.

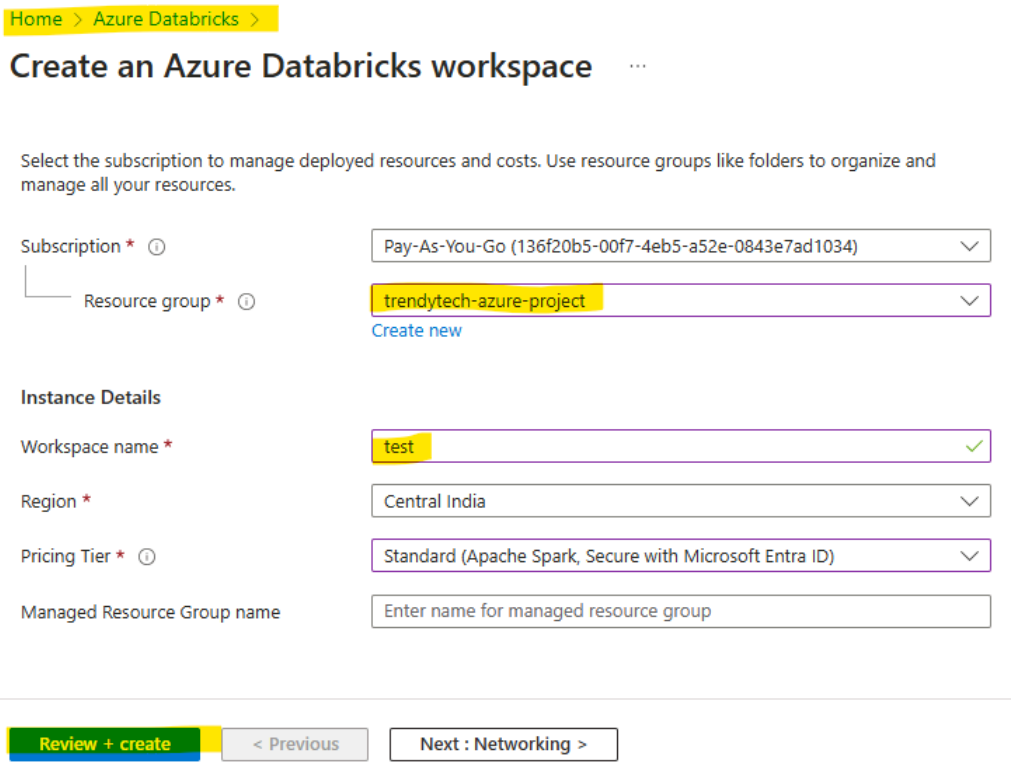


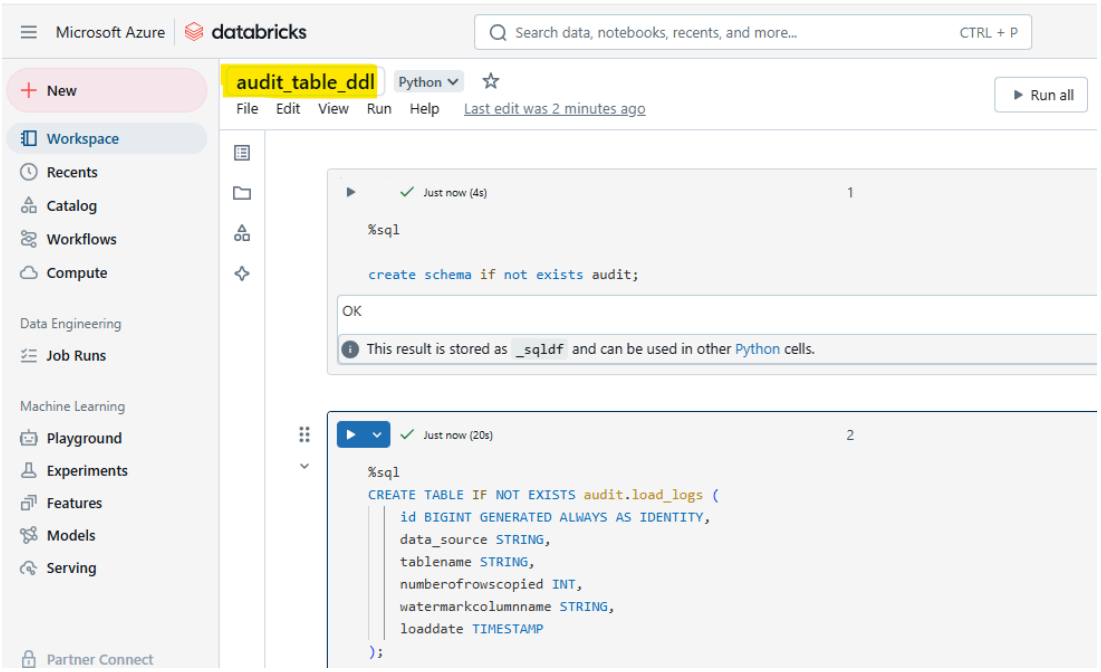
3. Delta table - Audit_logs

Create the databricks workspace test and then upload the code notebook “Audit_table_DDL” and start your databricks cluster and create the schema audit table in it using below commands. (Notebook name - audit_table_ddl)

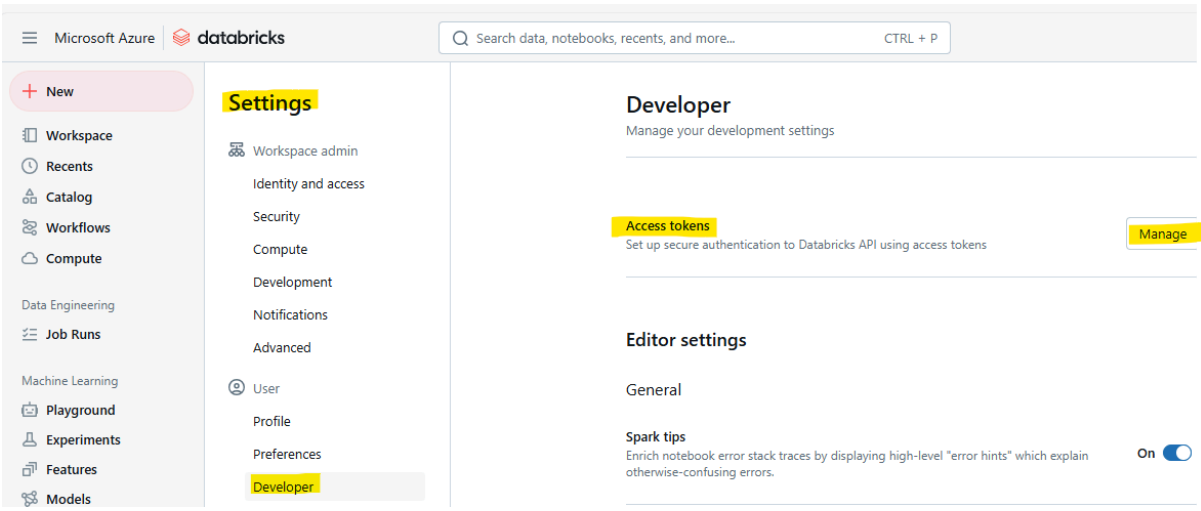
create schema if not exists audit;

```
CREATE TABLE IF NOT EXISTS audit.load_logs (
    id BIGINT GENERATED ALWAYS AS IDENTITY,
    data_source STRING,
    tablename STRING,
    numberofrowscopied INT,
    watermarkcolumnname STRING,
    loaddate TIMESTAMP
);
```



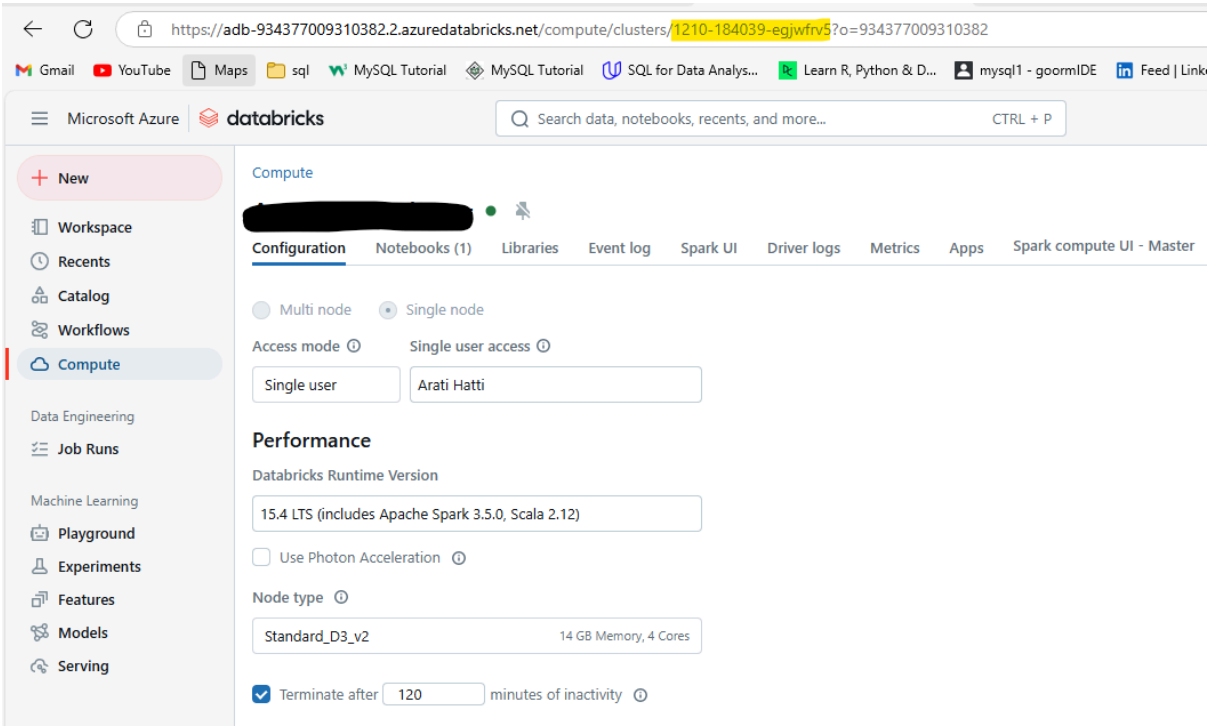


Note: To get access token Click your profile icon (top-right corner of the workspace) => Select Settings => Developer (In user setting) => Generate Access Token as shown below

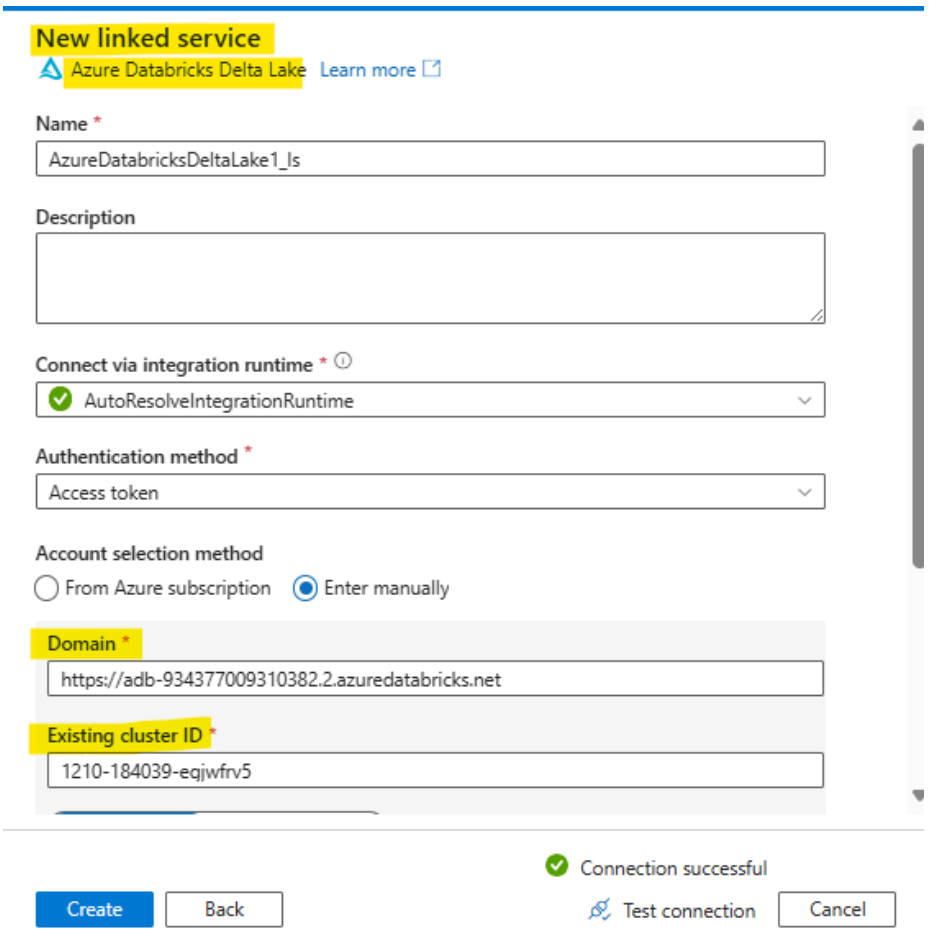


Then generate the token and copy for Future use.

While creating linked service in source mention “AzureDatabricksDeltaLake”. Then in the domain mention the URL of databricks workspace. And mention the cluster id for the cluster that we have created. To get Datbricks overview page to get Workspace URL and to get cluster id go to compute => Select the cluster => copy the cluster id as shown below



Refer below screenshot for more details.

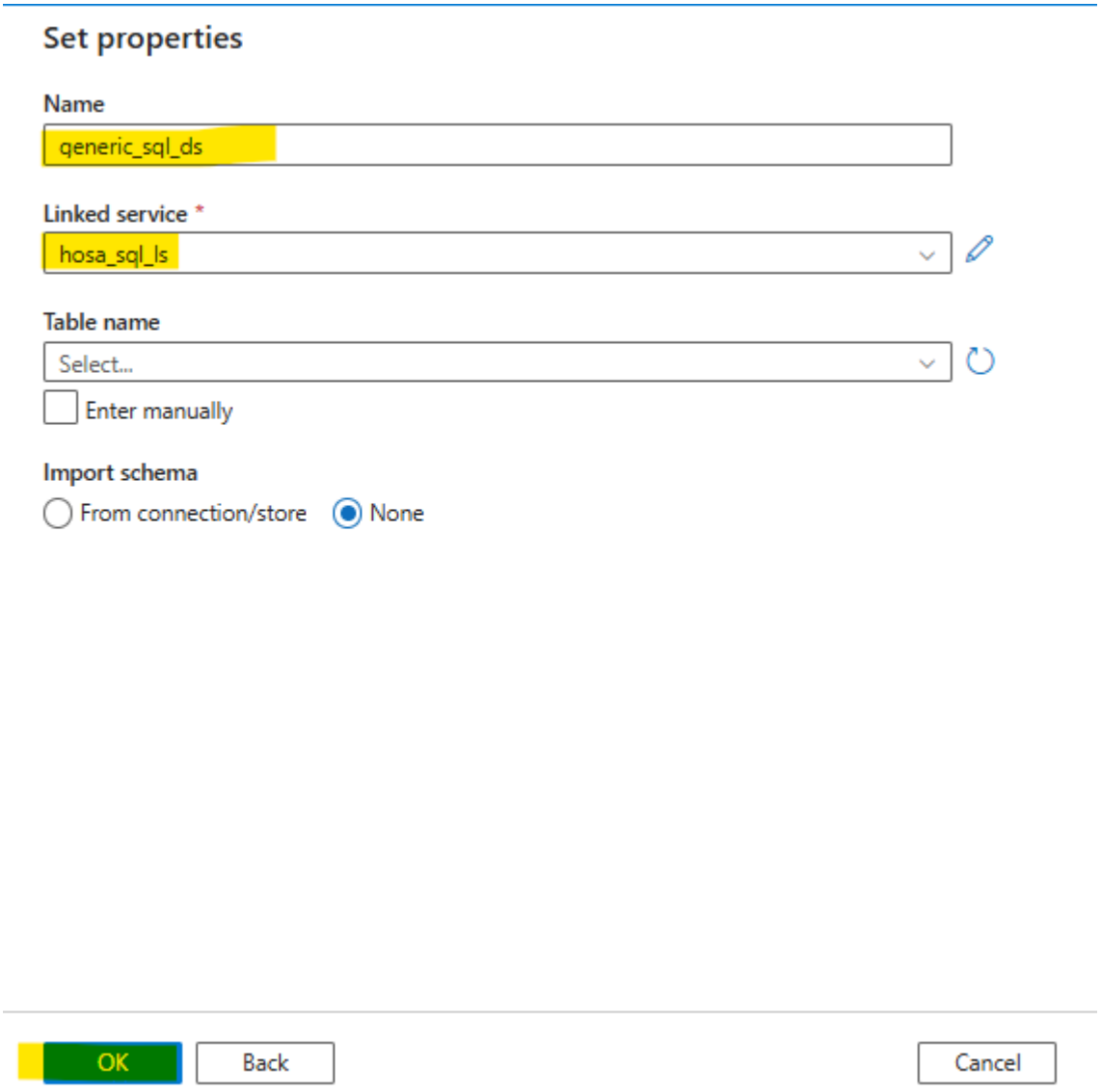


Dataset creation:

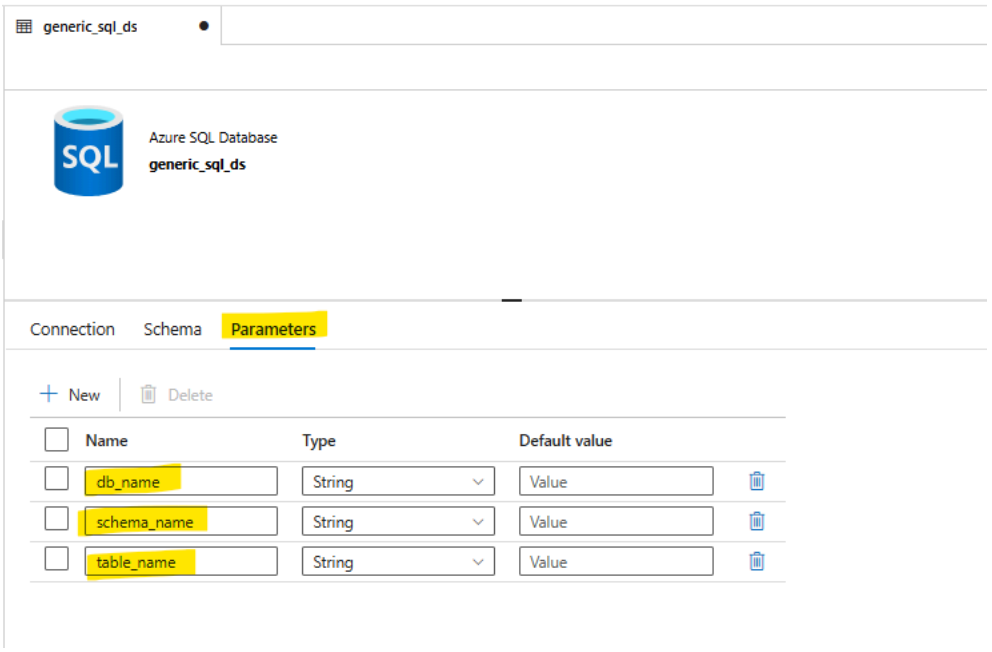
In the ADF interface, click on the Author section (left-hand panel). Expand the Datasets option. Click on the "…" next to Datasets in order to create the dataset.

1. Azure SQL DB

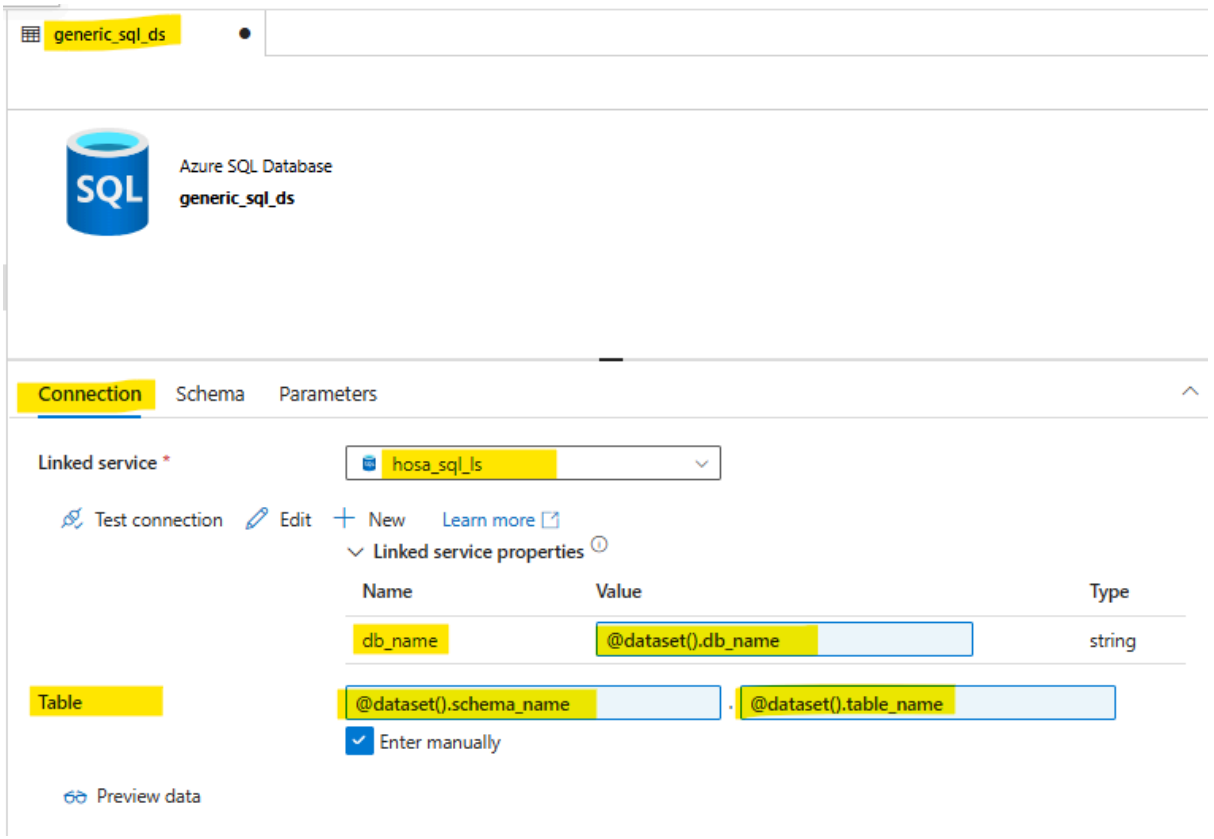
We will select the linked service that we have created for the SQL database. To create the datasets for the tables in a parameterized way in the sql database , we will create the parameter db_name, schema_name, table_name.



Then we will create parameters db_name, schema_name, table_name as shown below.

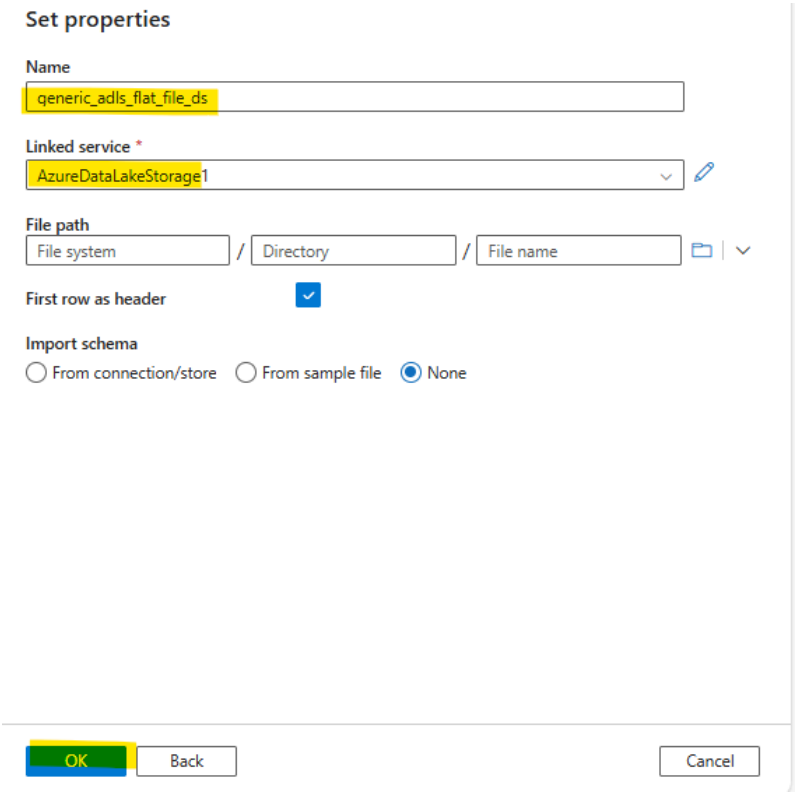


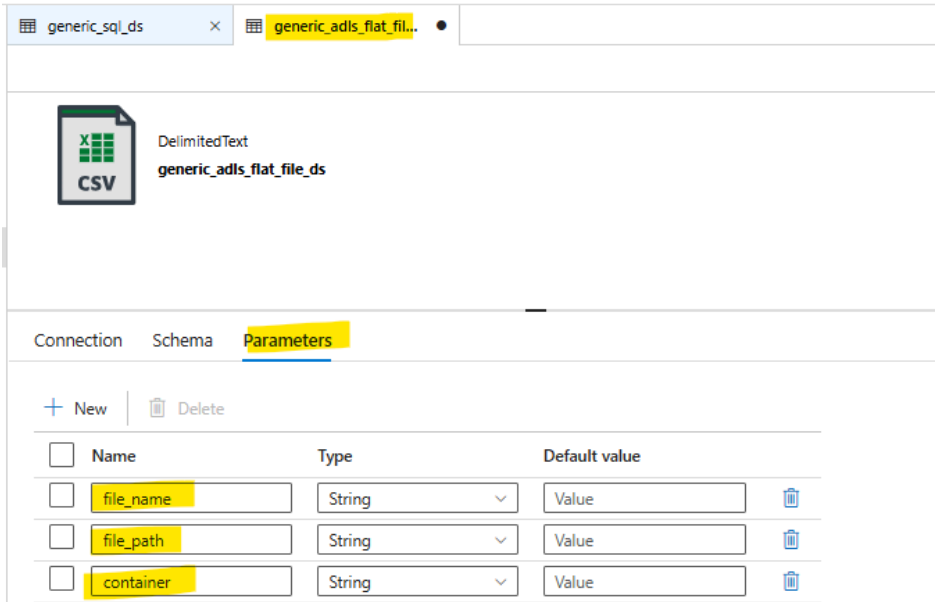
And we will pass dynamic value for table name and schema name as shown below

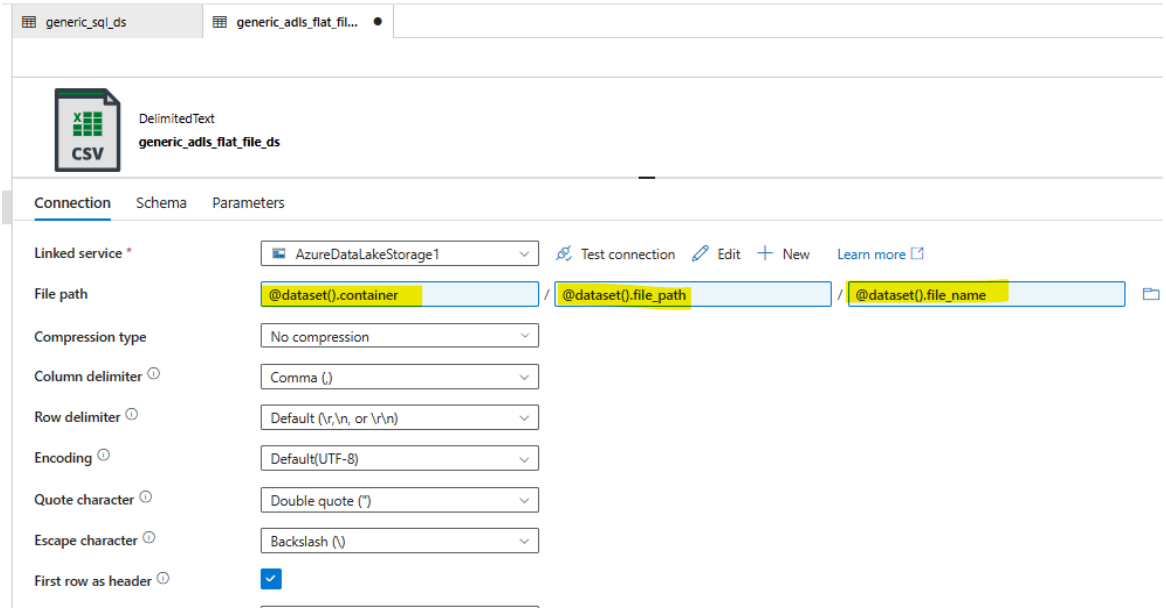


2. Dataset for Flatfile in ADLS GEN2

Select source as ADLS gen2 and file format as delimited text. Also in order to make it generic we will create the parameter file_name, file_path and container.



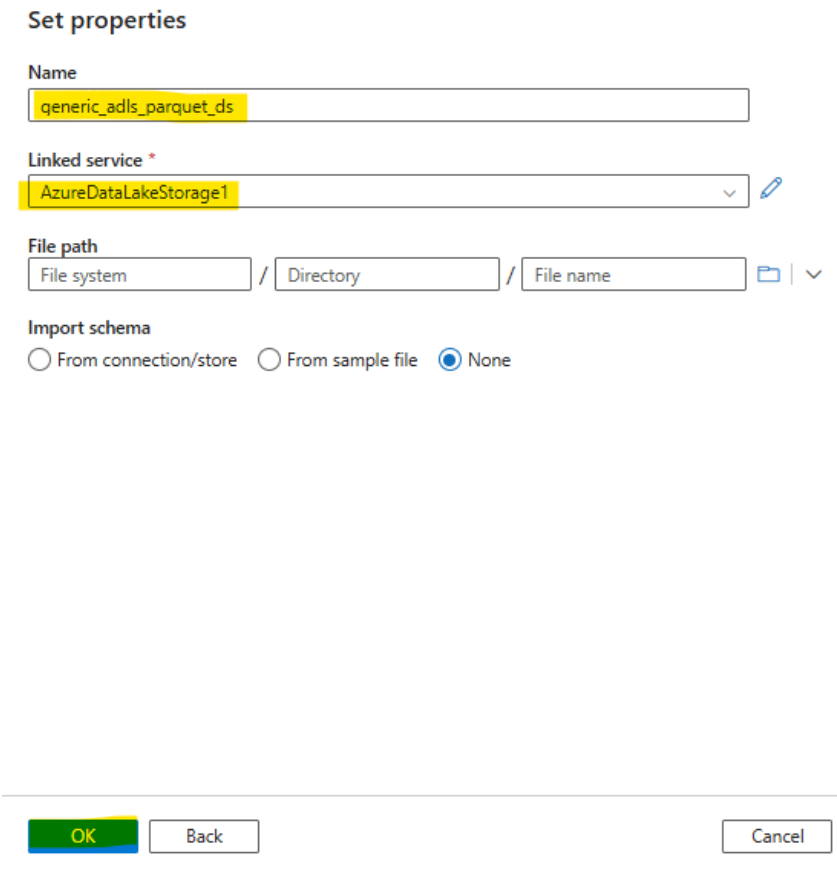


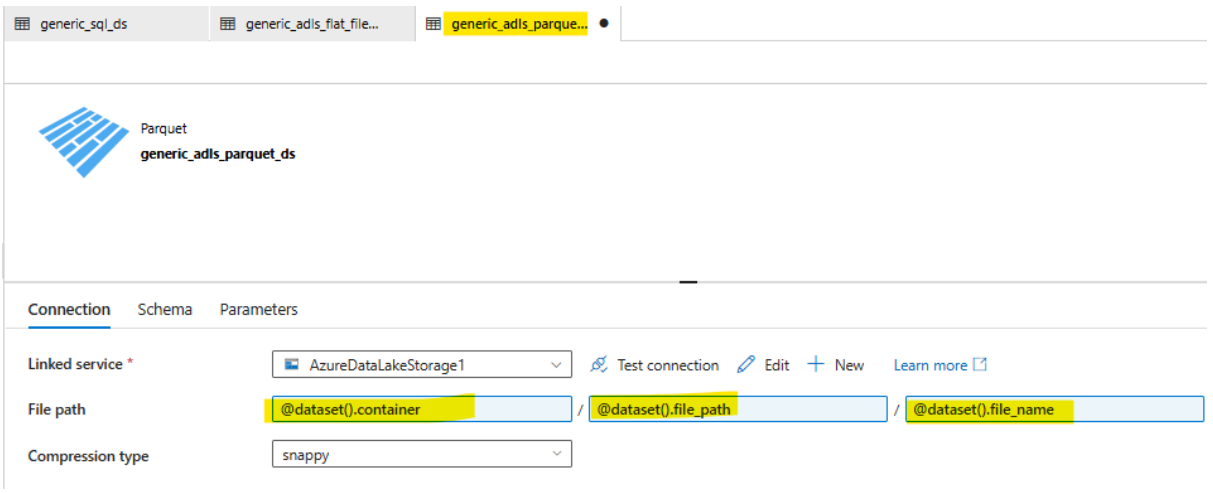


Now publish the changes.

3. Dataset for Parquet file in ADLS GEN2

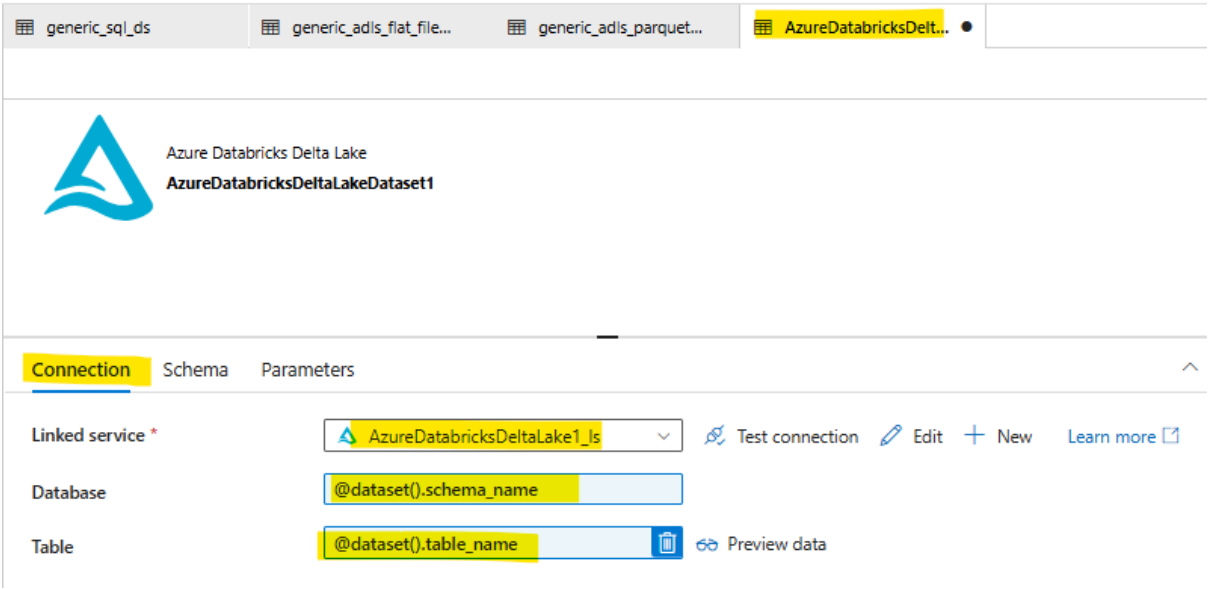
In order to store data in ADLS gen2 in parquet format we will need the dataset.
While creating this dataset we will select source as ADLS gen2, fileformat as parquet and we will create parameters file_name, file_path and container.





4. Databricks Delta Lake for Delta lake

We will select the source as Azure Databricks Delta Lake. For this we will create the parameter schema_name and table_name.



Once all the dataset and linked service are created, publish all in order to save them.

Creation of Pipelines:

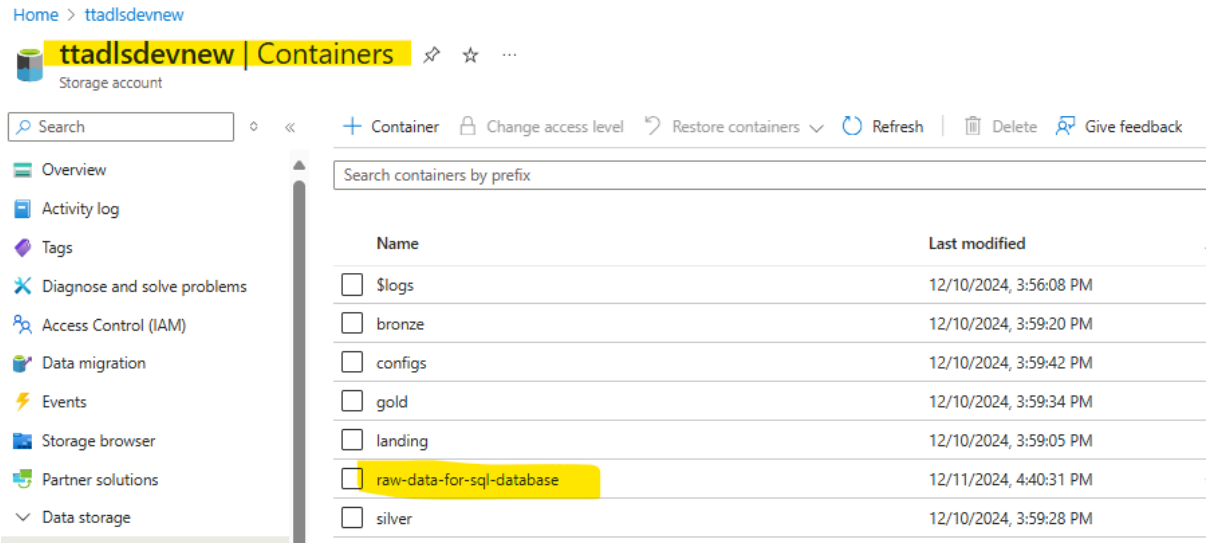
Background activity : Creation of pipeline to copy data into sql tables (pl_to_insert_data_to_sql_table_preprocessing).

Before proceeding with the main pipeline, we will create a simple pipeline in Azure Data Factory (ADF) to copy data from ADLS Gen2 storage into tables in an SQL database. This serves as a prerequisite to ensure that the SQL tables contain the data needed for the main pipeline.

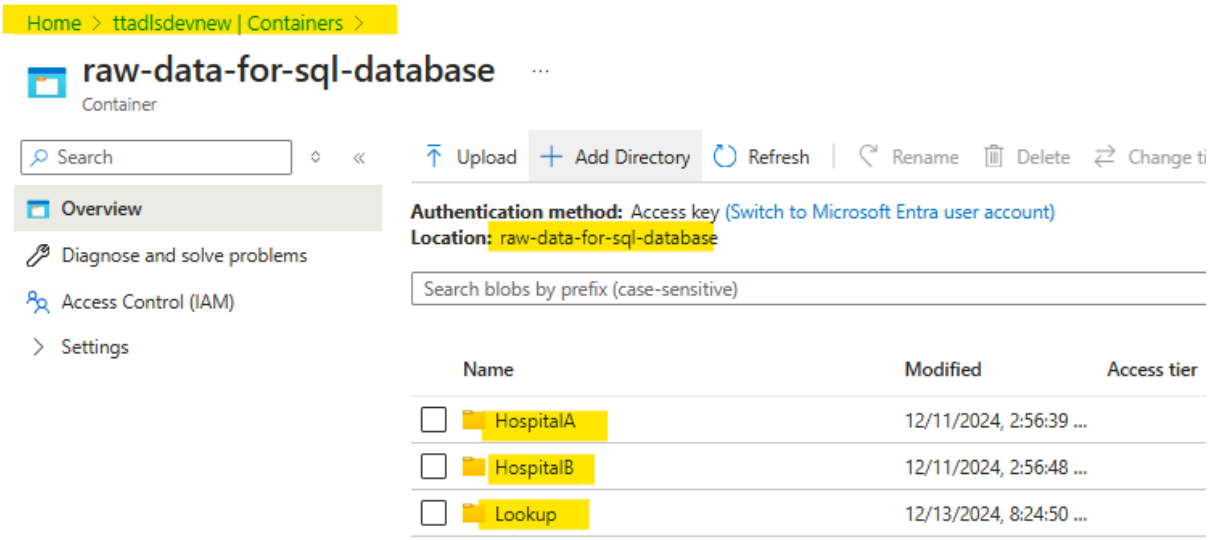
Note: We will create a new container (raw-data-for-sql-database) in the given ADLS Gen2 storage (adlsdevnew) and upload our CSV files, which will serve as the source for the pipeline, along with a lookup file. Additionally, we will create a dataset for the lookup file to use in this pipeline. Using the copy activity in ADF, we will transfer the data into the following tables: Departments, Providers, Encounters, Patients, and Transactions, located in the SQL databases trendytech-hospital-a and trendytech-hospital-b.

Source: ADLS gen2 -adlsdevnew

We will create a new container (raw-data-for-sql-database) in the given ADLS Gen2 storage (adlsdevnew) and upload our CSV files, along with a lookup file.

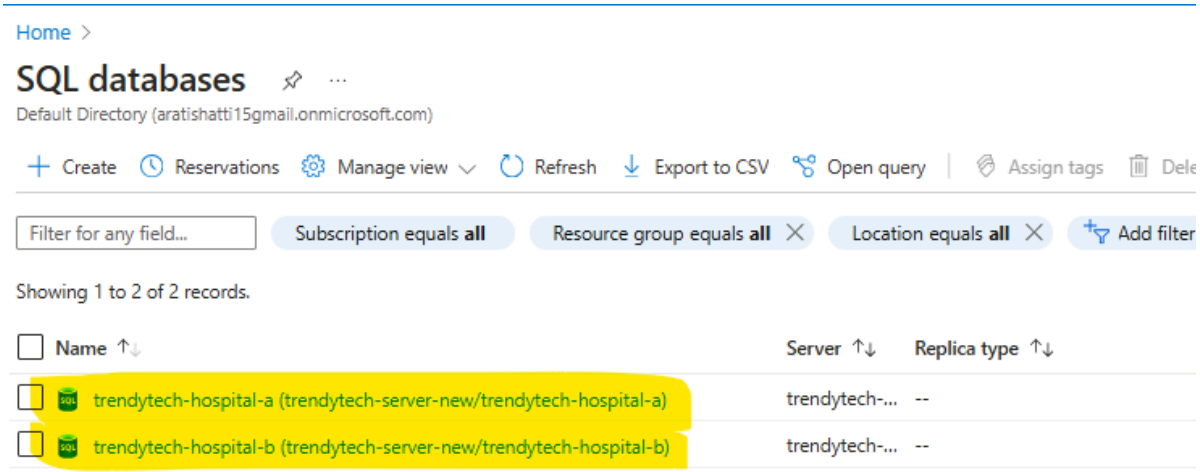


Folder: HospitalA, HospitalB for datafiles, Lookup for lookup file



Sink: SQL DB - trendytech-hospital-a, trendytech-hospital-b:

Note: We have already created these databases so no need to create again.



Pipeline creation Steps:

1. **Creation of Linked Services:**
   => For ADLS gen2 storage(source):

We will use the same linked service “AzureDataLakeStorage1” that we have created earlier for ADLS Gen2 storage.

=> For SQL DB(Sink):

We will use the same linked service “hosa_sql_ls” that we have created earlier for the database.

2. Creation of Datasets:

=> For source:

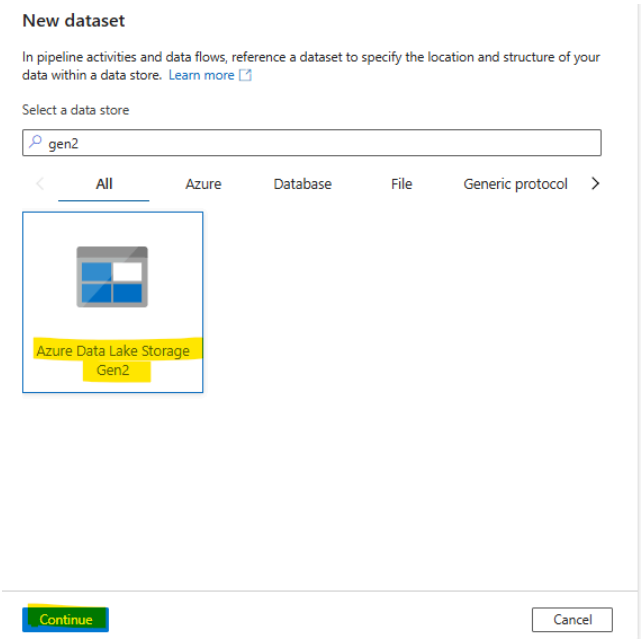
We will use the same generic dataset “generic_adls_flat_file_ds” that we have created earlier.

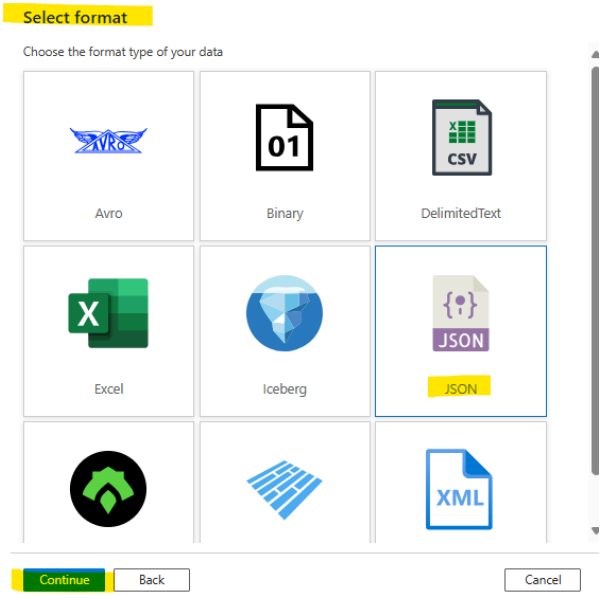
=> For sink:

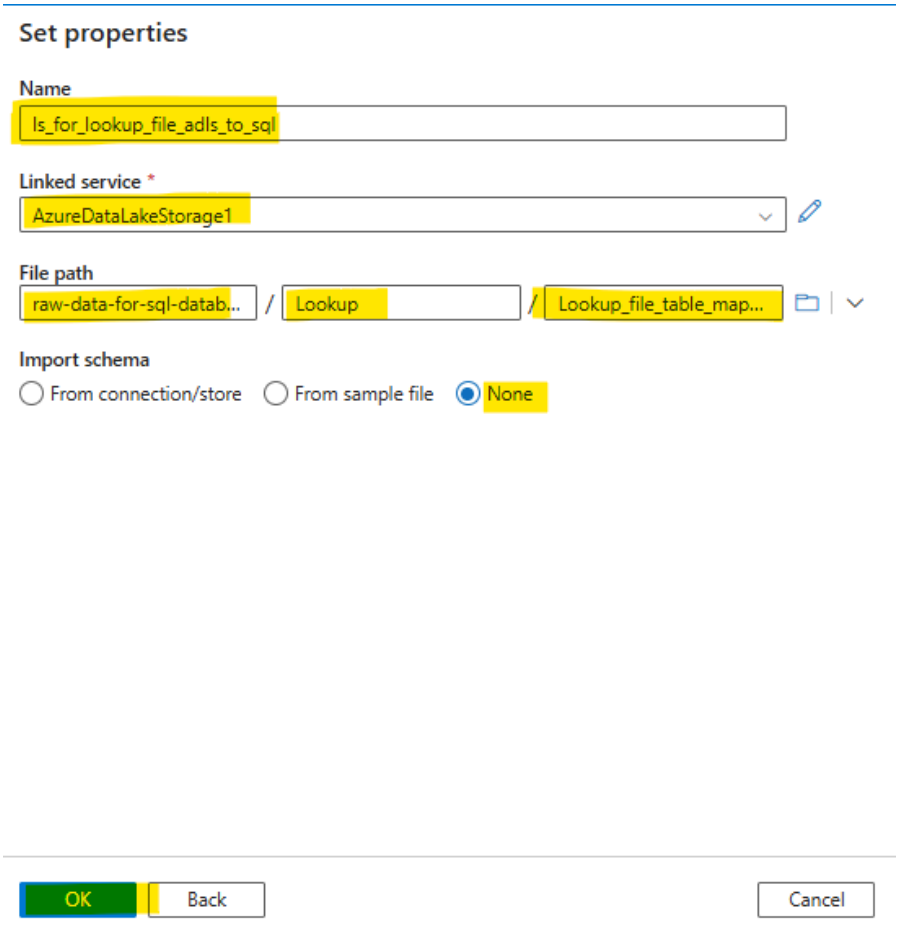
We will use the same generic dataset “generic_sql_ds” that we have created earlier.

=>For Lookup we will create a new dataset as shown below.

Select source as ADLS Gen2 storage, then in file format select json as our lookup file is a json file as shown below



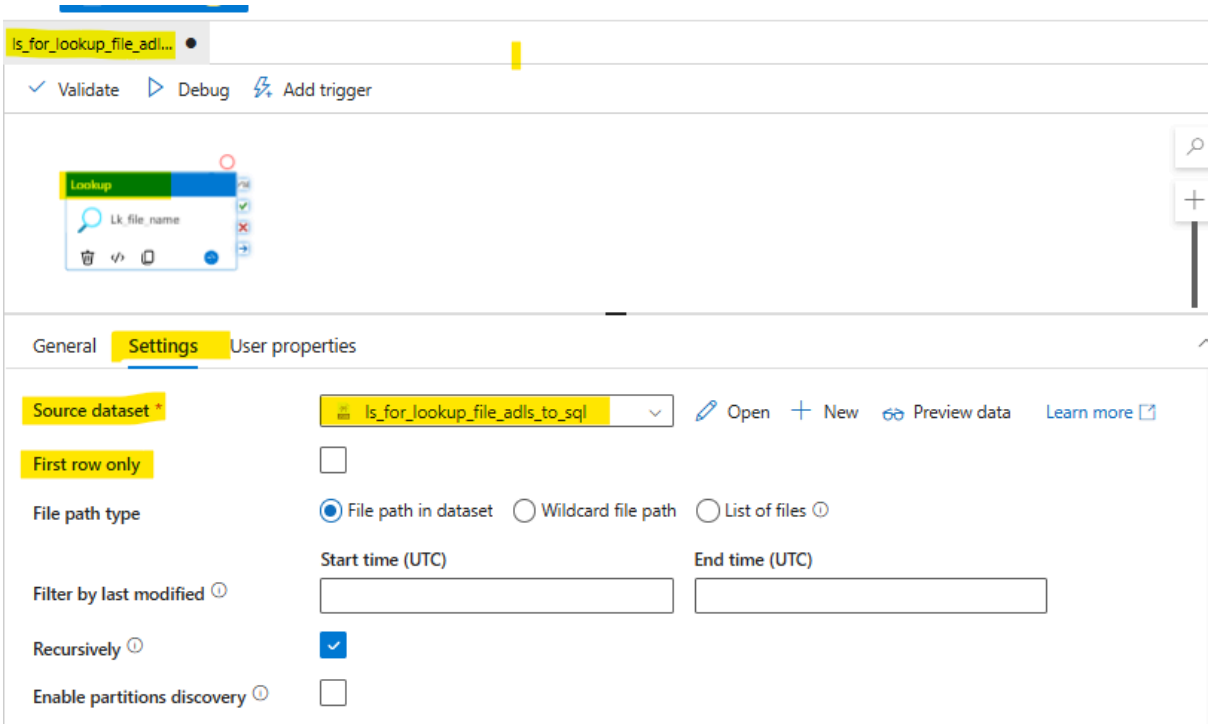




Steps to Configure the Pipeline:

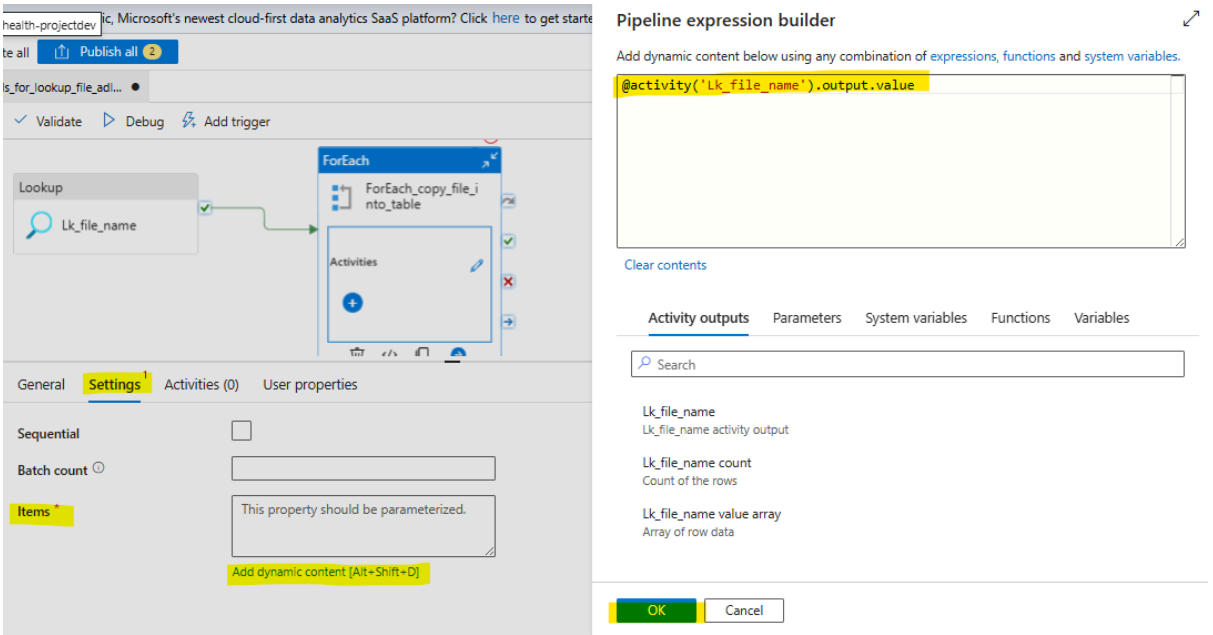
Add a Lookup Activity:

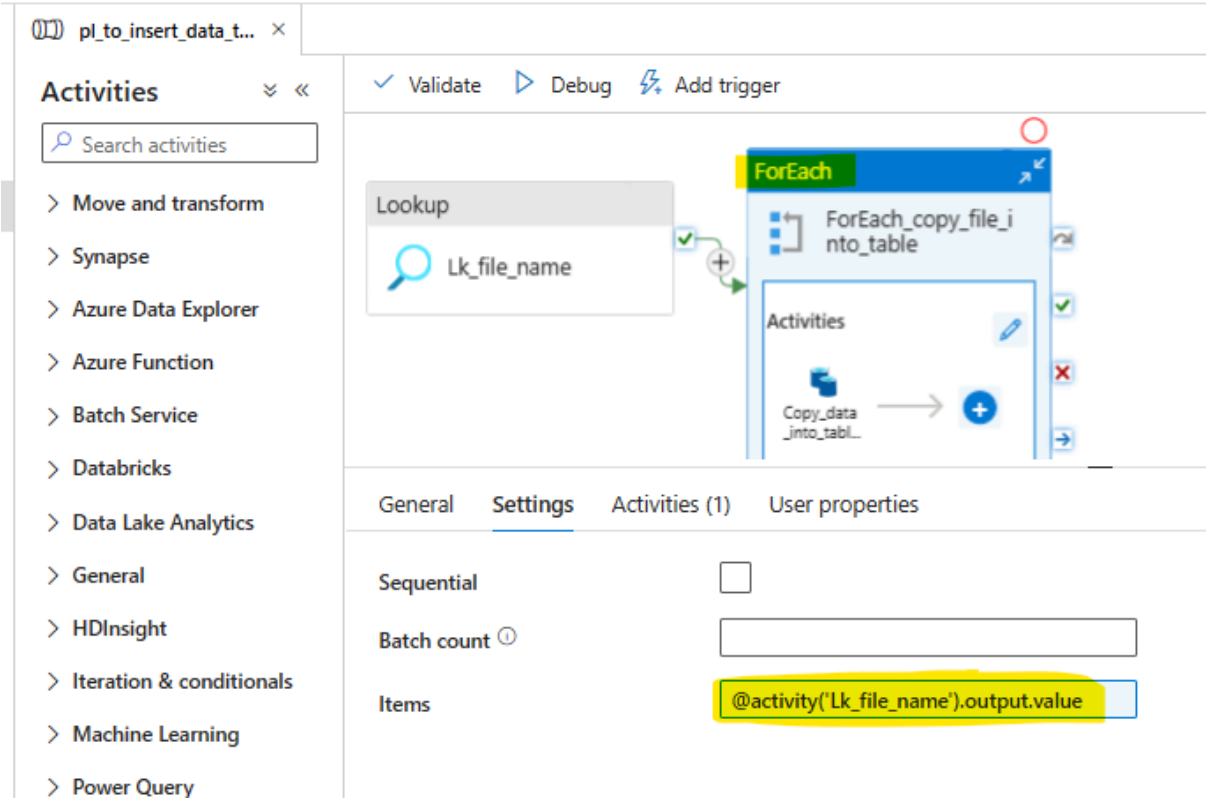
Drag a Lookup activity to the canvas. Point it to the mapping CSV dataset.Set First row only to false to read all rows. Refer below screenshot for more clarity.



Add a ForEach Activity:

Drag a ForEach activity and connect it to the Lookup activity.
Set its Items property to @activity('Lk_file_name').output.value
Refer below screenshot for more clarity.

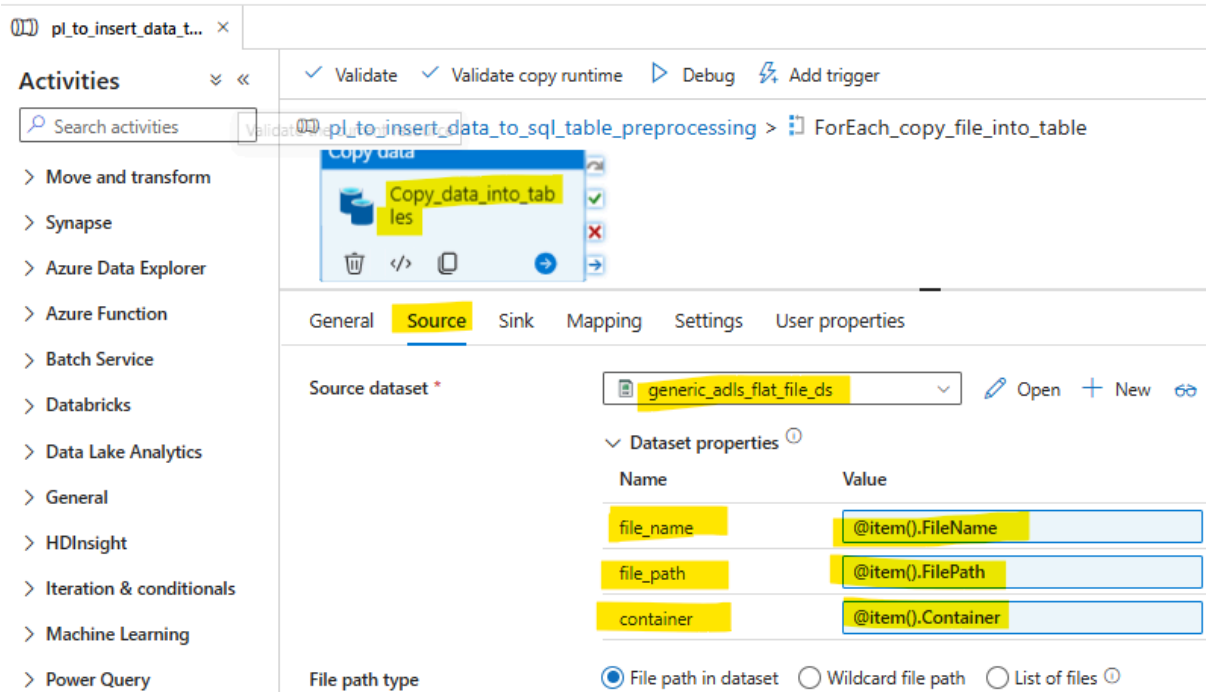




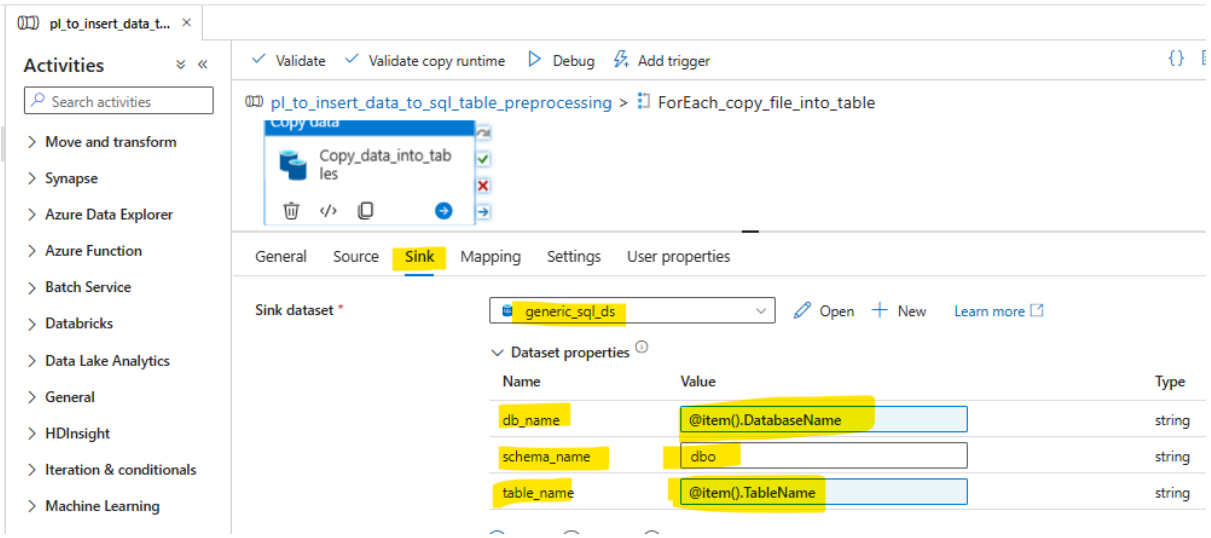
Configure the ForEach Activity:

Inside the ForEach activity, add a Copy Data activity.

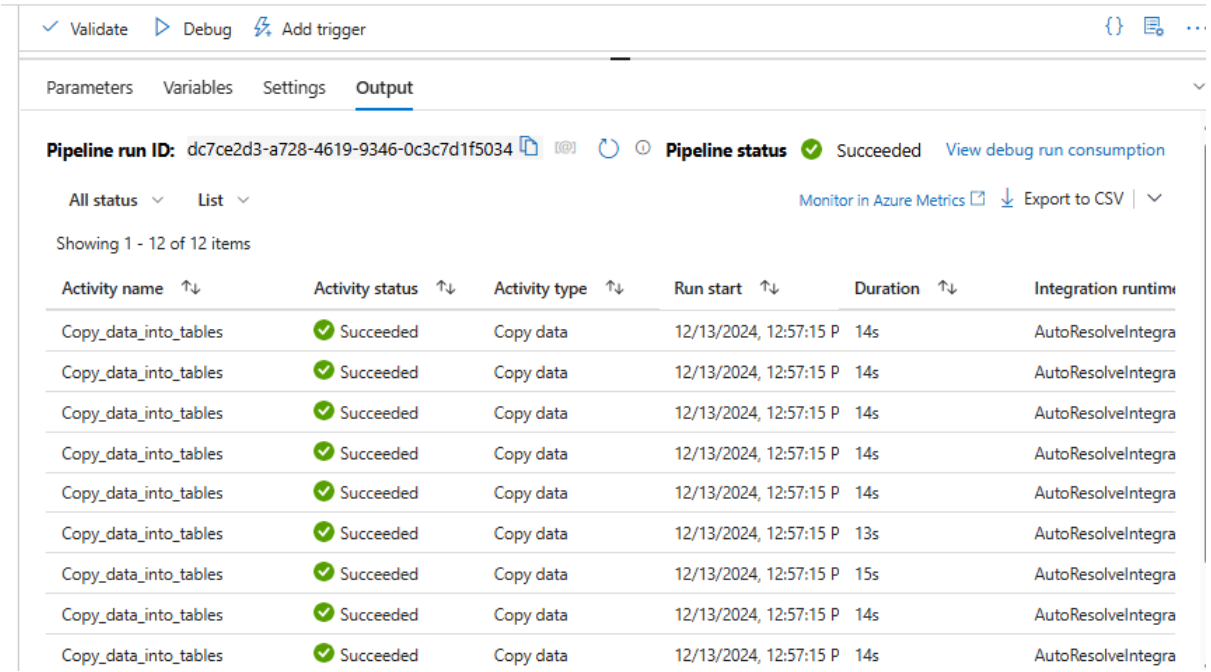
Source: Use the source dataset.



Sink: Use the destination dataset.



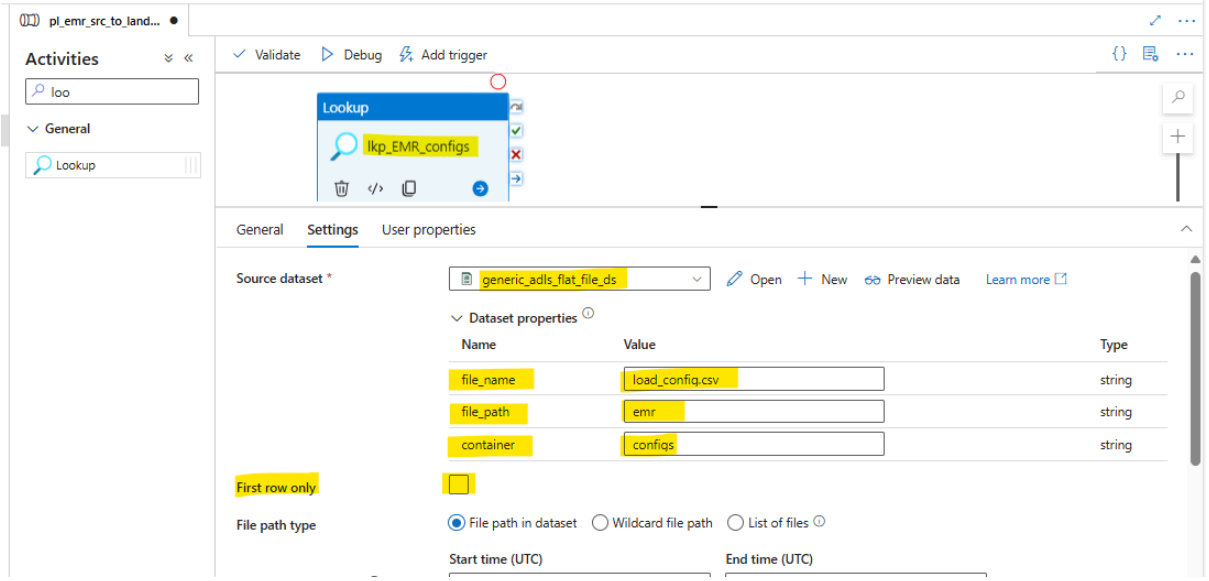
This pipeline will copy the data from the file into the tables in the sql database. On successfully running the pipeline we will get below output.



Pipeline to copy data from Azure Sql db to Landing Folder in ADLS Gen2

1. To read the config file we will use Lookup activity.

In this for source dataset will be for configs file and we will pass the parameter values as shown below.

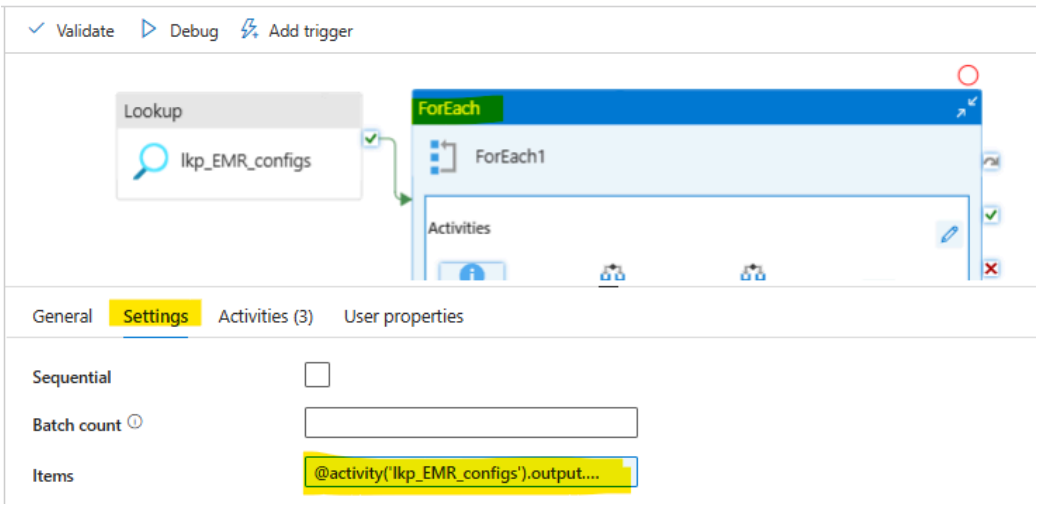


Then additionally we can preview the data.

2. In order to iterate through each row of configuration data we will use ForEach Activity.

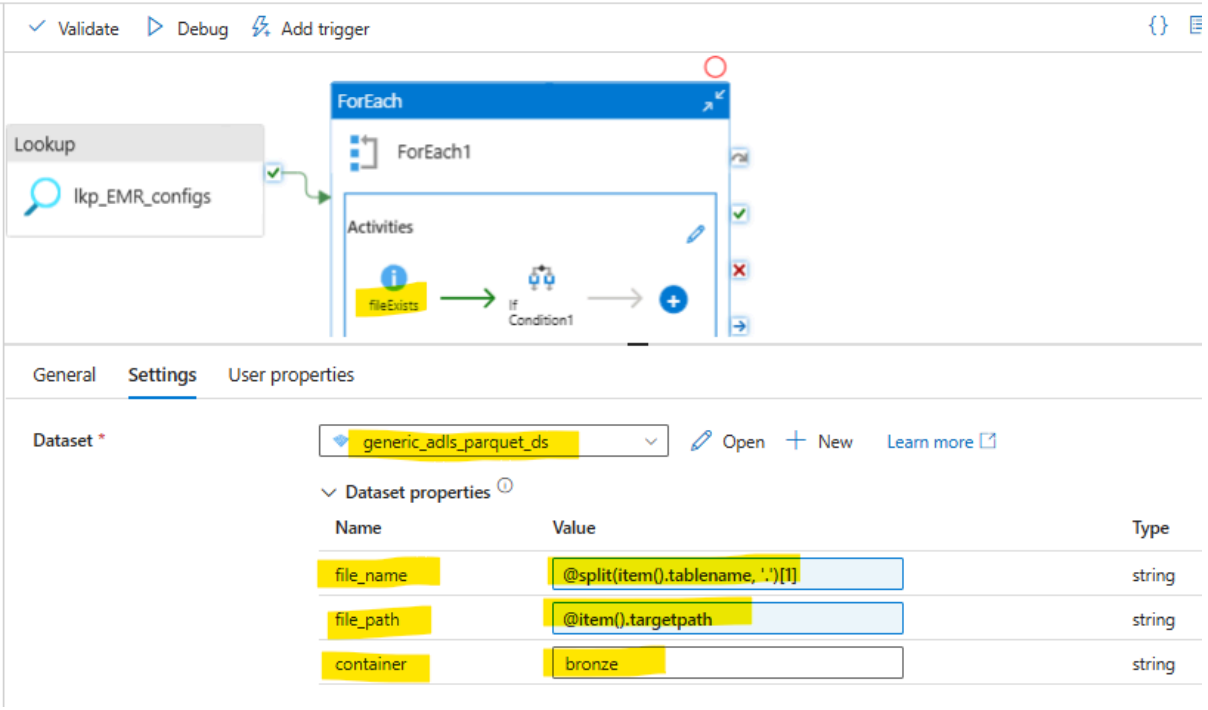
Processing Logic Within ForEach Activity:

@activity('lkp_EMR_configs').output.value



a. We will use get metadata activity in order to check whether file exists in Bronze container:

To file name we will use below logic - @split(item().tablename, '.')[1]
file_path is present in lookup file as targetpath
And container name we will explicitly mention as bronze as shown below

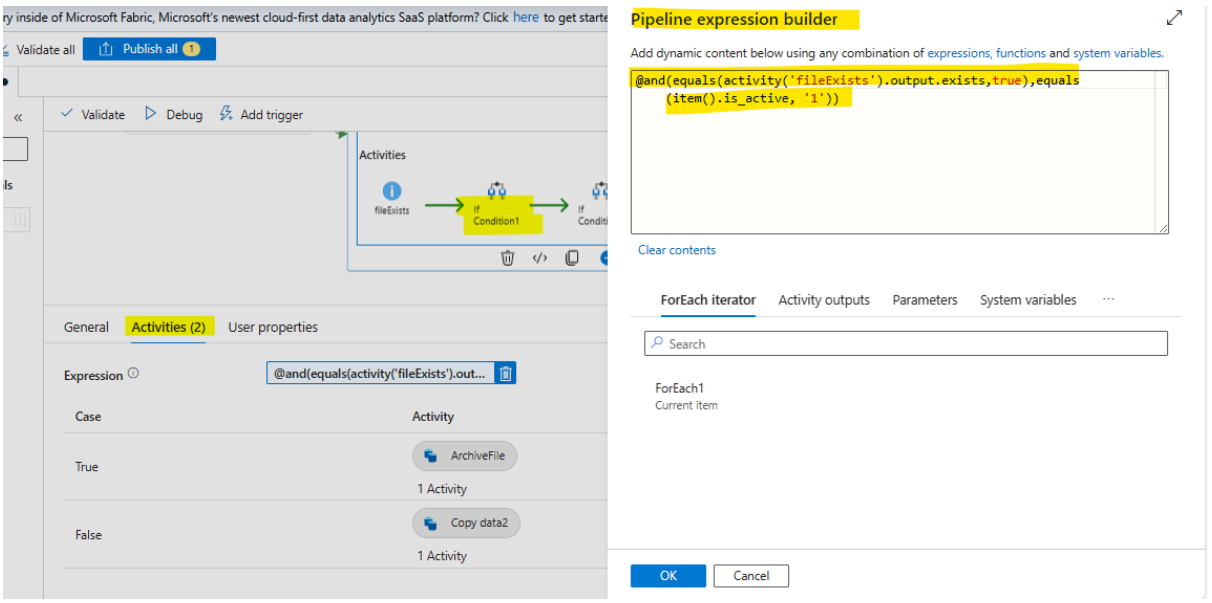


This will check if the file exists in the Bronze container. Based on the file's presence or absence, we will use an If Condition activity to determine the subsequent processing steps.

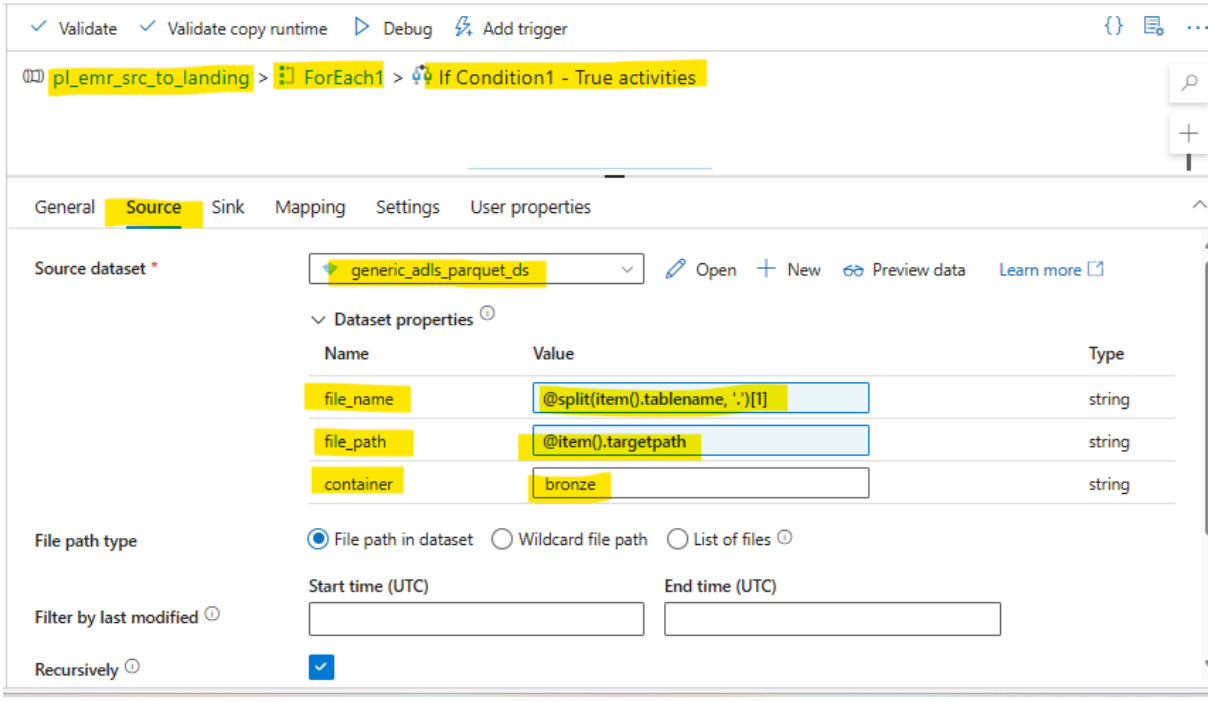
b. Use an If Condition activity based on the file's existence.

Condition 1: File Exists (True) => Move the file to the Archive folder.

condition:
@and(equals(activity('fileExists').output.exists,true),equals(item().is_active, '1'))



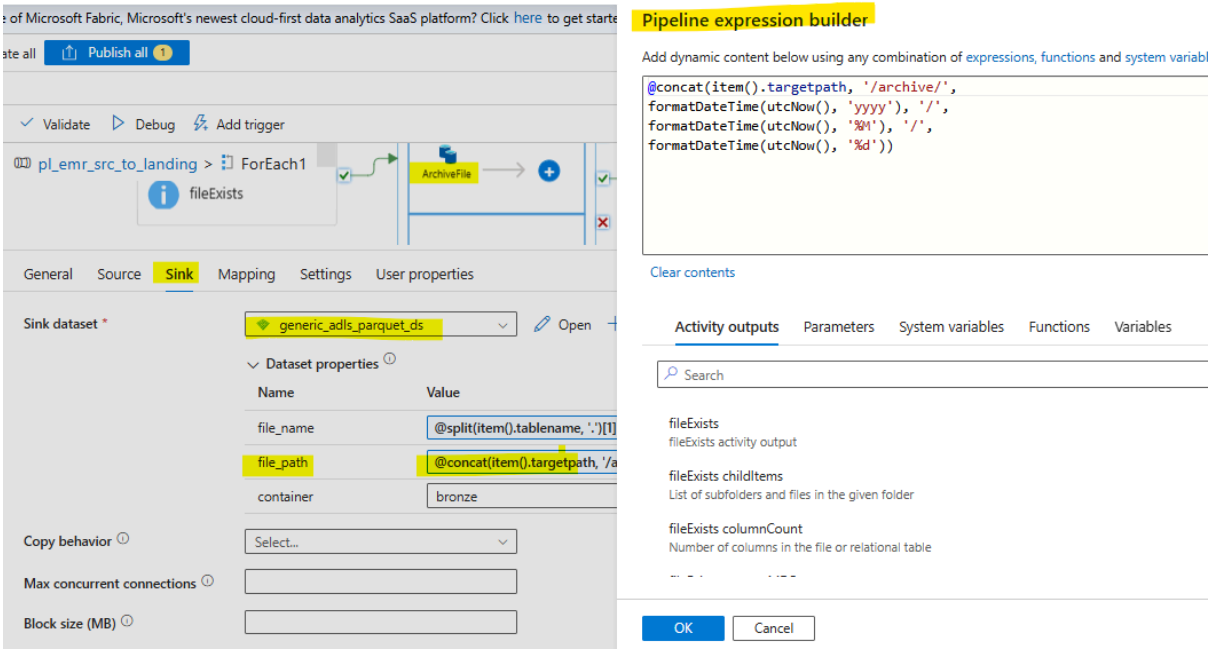
Source: Container: Bronze, Path: hosa, File: encounters



Target: Container: Bronze,

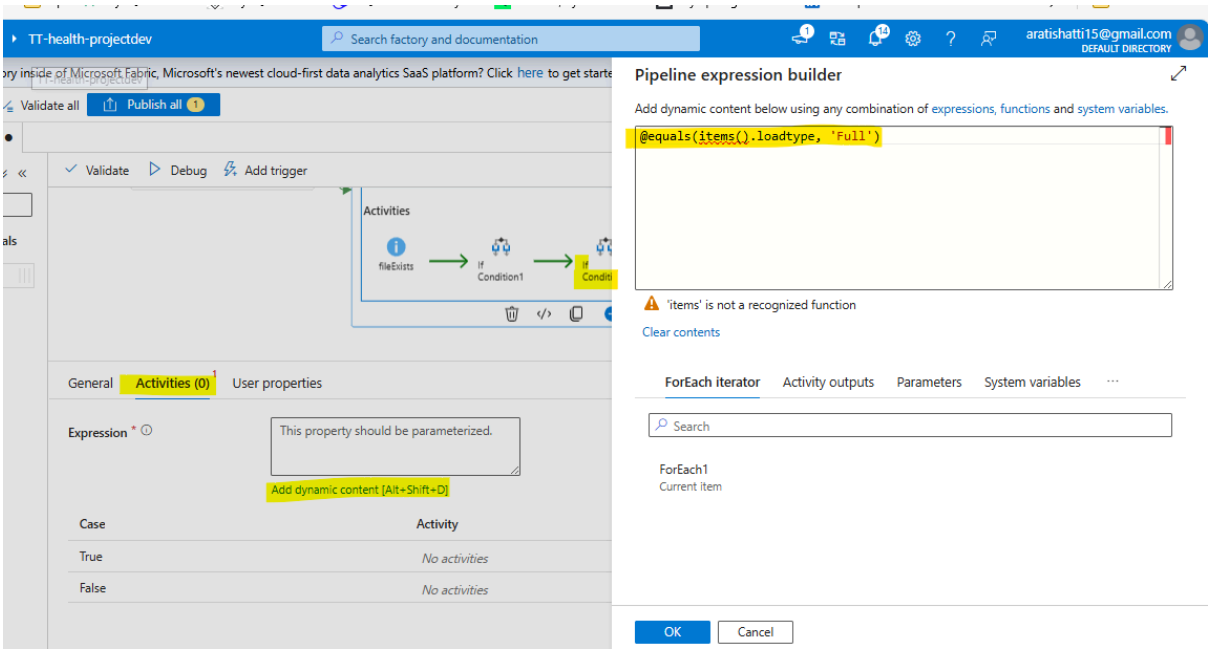
File_path -Path: hosa/archive/<year>/<month>/<day> =>
@concat(item().targetpath, '/archive/',
formatDateTime(utcNow(), 'yyyy'), '/',
formatDateTime(utcNow(), '%M'), '/',
formatDateTime(utcNow(), '%d'))

File_name - @split(item().tablename, '.')[1]



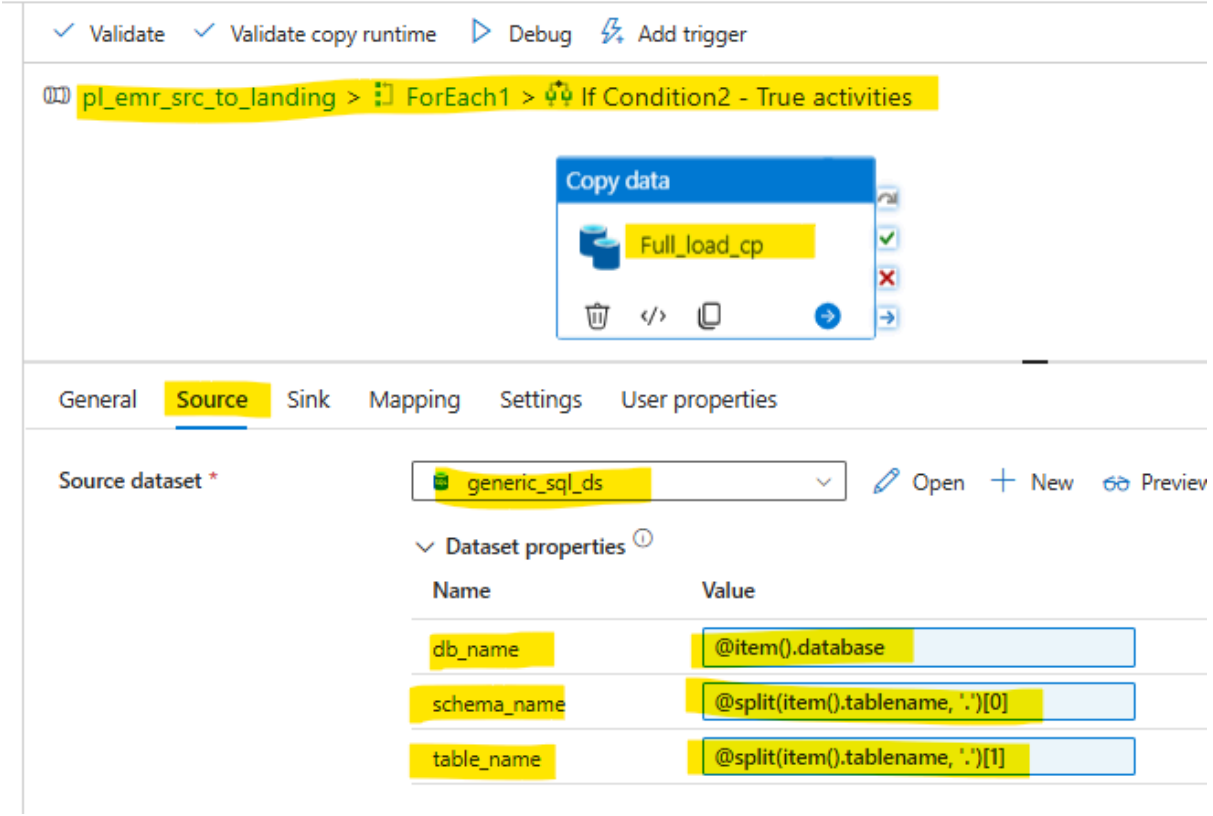
c. Determine if it's a full load or incremental load using If condition.

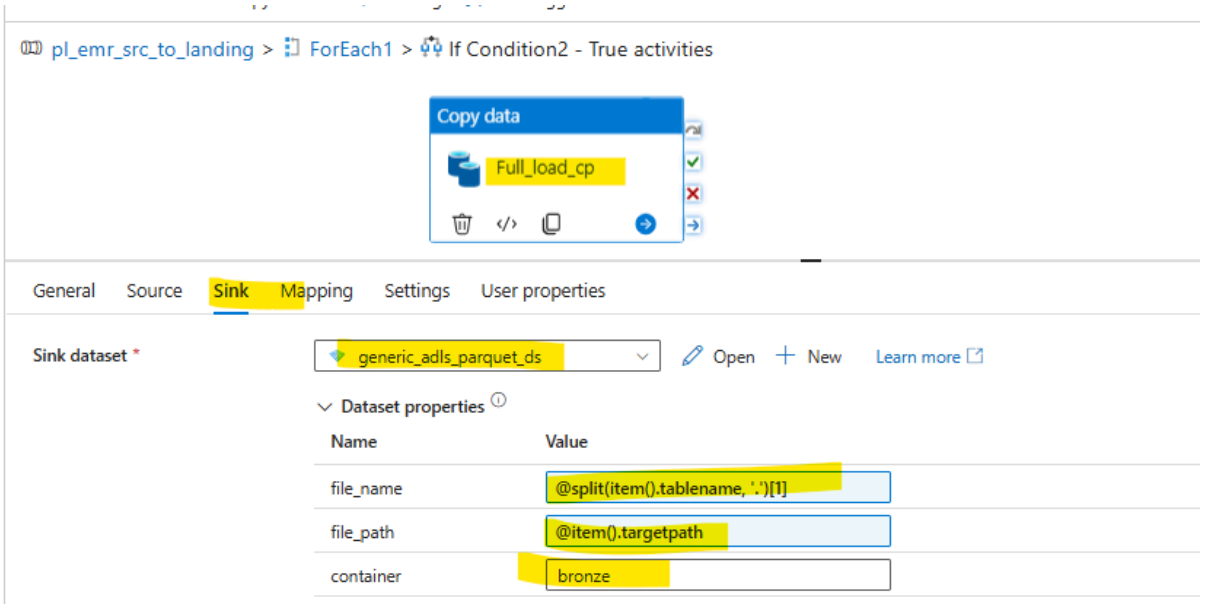
@equals(items().loadtype, 'Full')



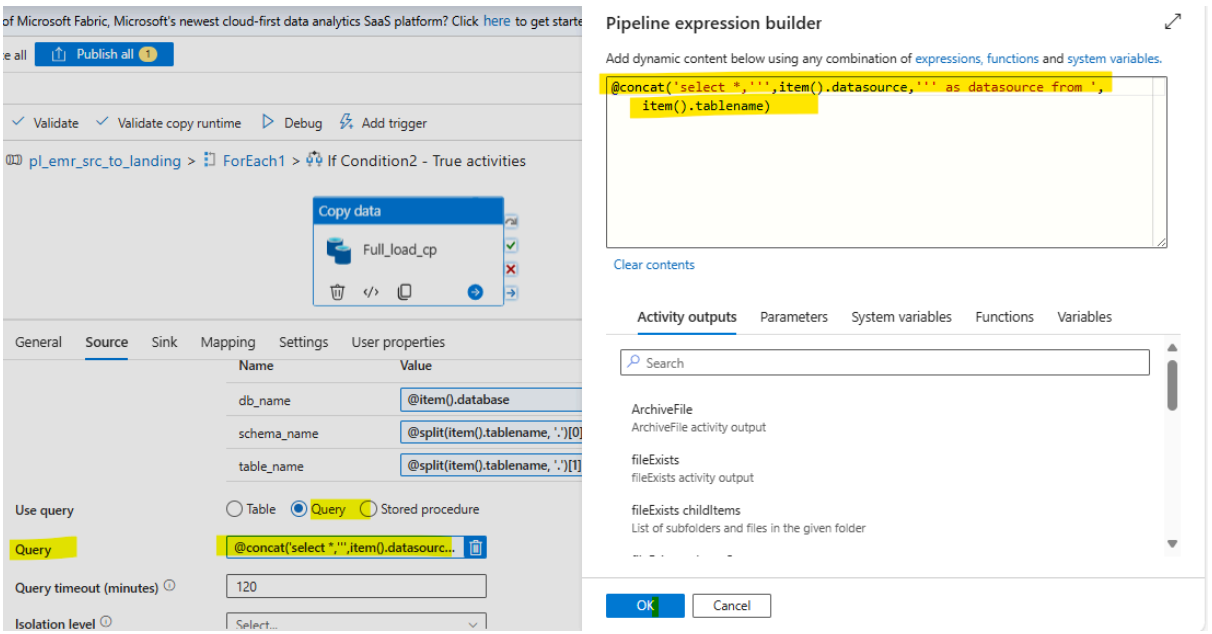
If "If condition" holds true => Full Load => Copy all data from the database table. => Enter Log details in the audit table:

Folder and File Structure
Bronze Container:
Source Path: bronze/hosa
Target Path for Data Loads: bronze/<target-path>



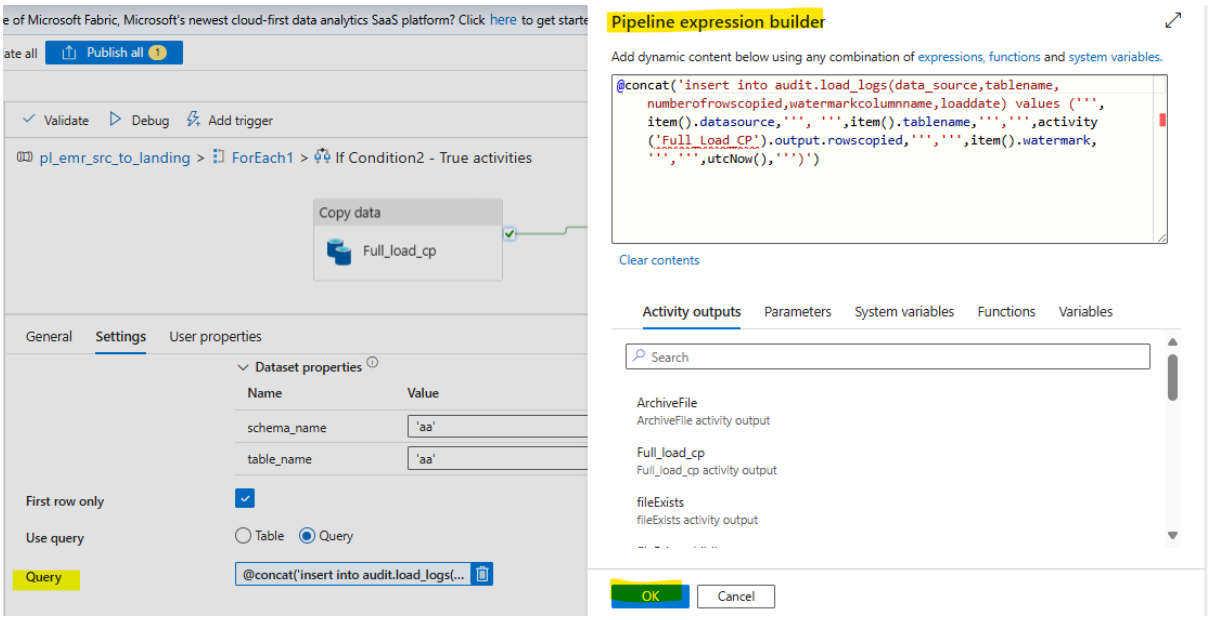


Query: @concat('select *,'',item().datasource,'' as datasource from ',item().tablename)



Enter Log details in the audit table:

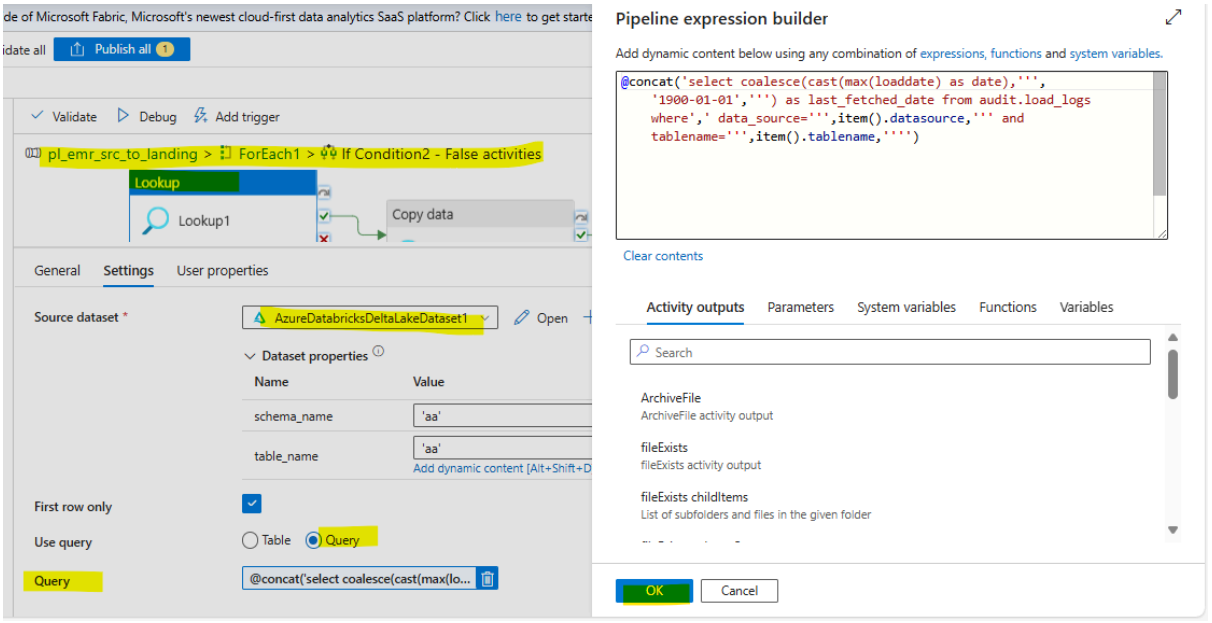
Query: @concat('insert into audit.load_logs(data_source,tablename,numberofrowscopied,watermarkcolumnname,loaddate) values ('',item().datasource,'',''',item().tablename,''','',activity('Full_Load_CP').output.rowscopied,''','',item().watermark,''','',utcNow(),''')')



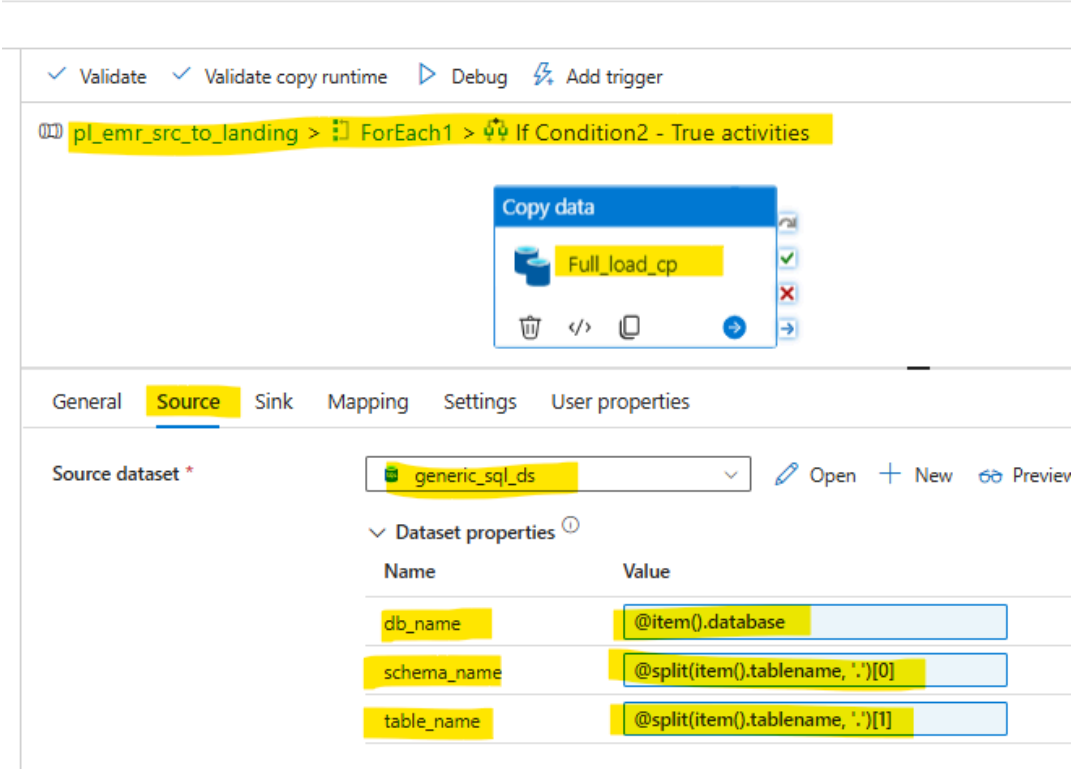
If condition is false => Incremental Load

(Fetch incremental data using the last fetched date) using Lookup=> Incremental load using copy activity =>Enter log details in the audit table:

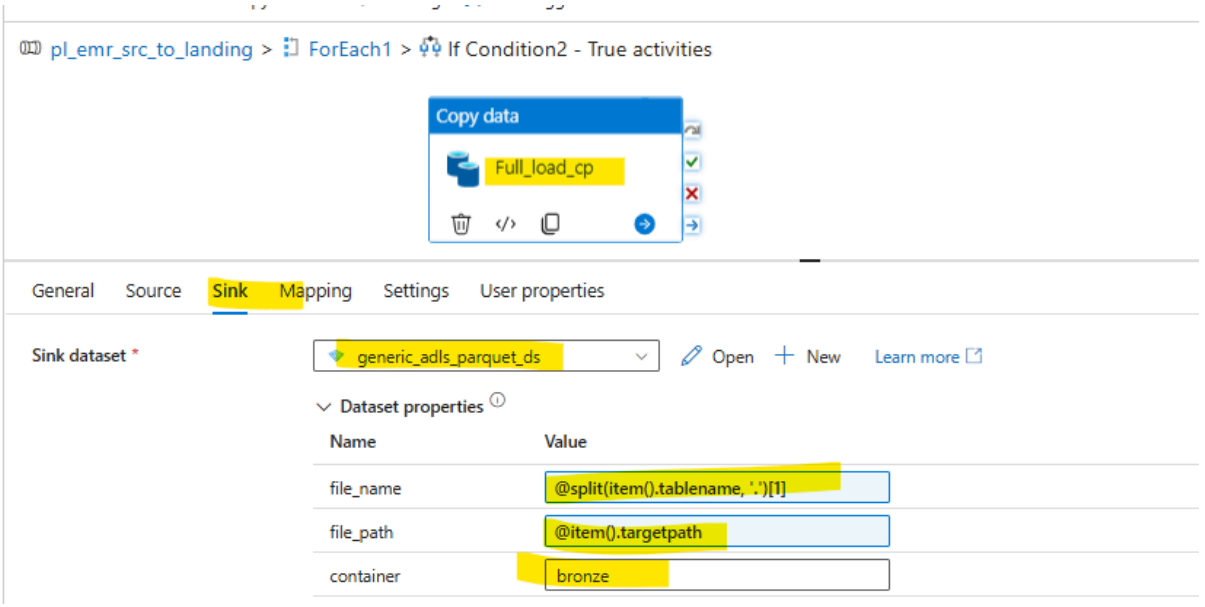
Lookup:



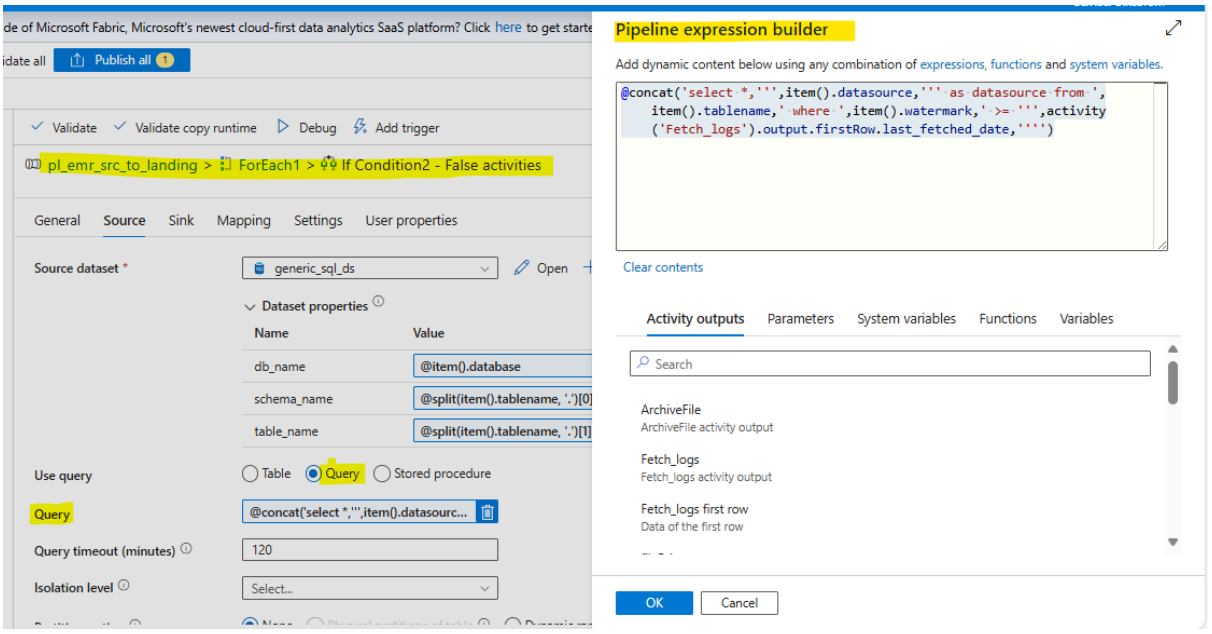
Incremental load:
Source Path: bronze/hosa



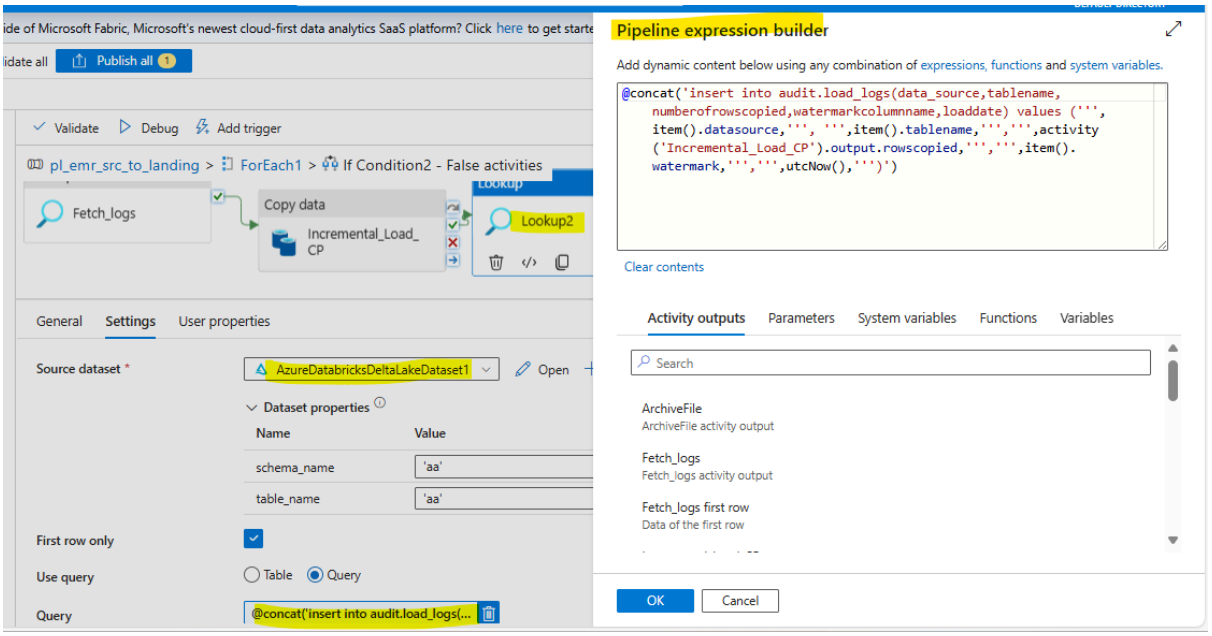
Target Path for Data Loads: bronze/<target-path>



Query: @concat('select *,'''',item().datasource,''' as datasource from ',item().tablename,' where ',item().watermark,' >= ''',activity('Fetch_logs').output.firstRow.last_fetched_date,'''')

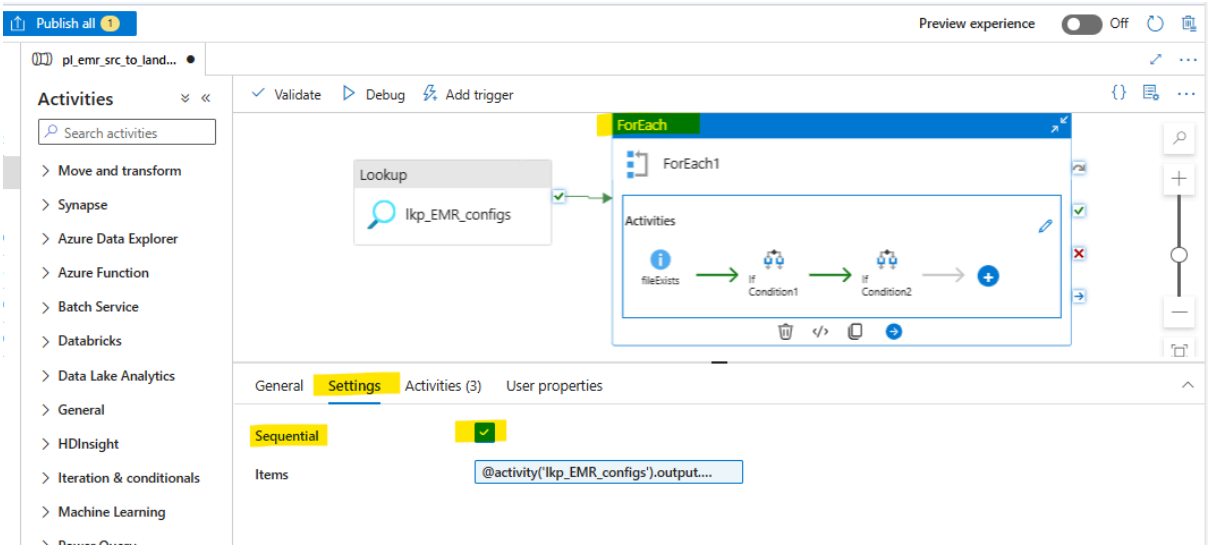


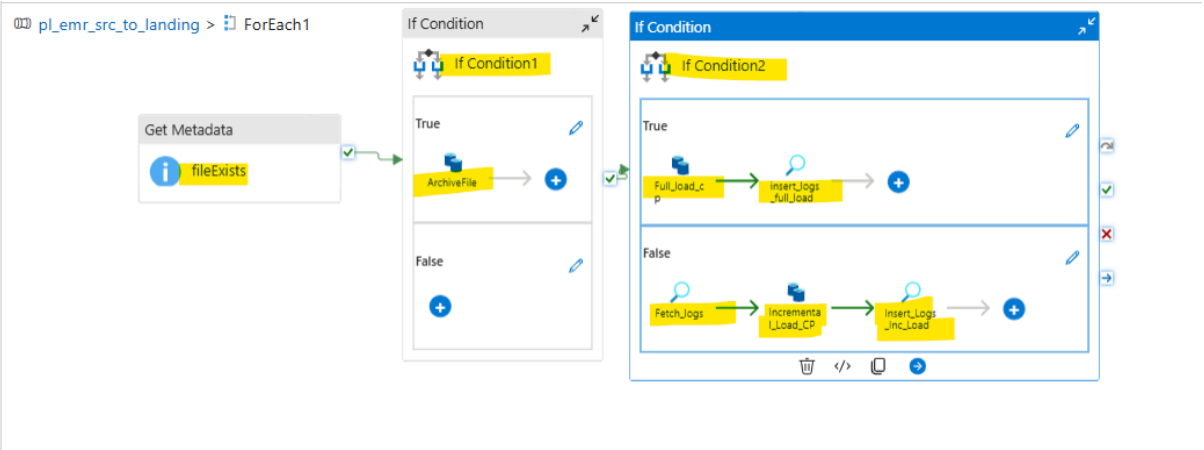
Lookup:



This is our complete pipeline:

Before running the pipeline for each activity select the “sequential” option as shown below.





But limitation with this pipeline is it is sequential which will we resolve in part 2

Part 2:

In this section, we will focus on improving our data pipeline and governance.

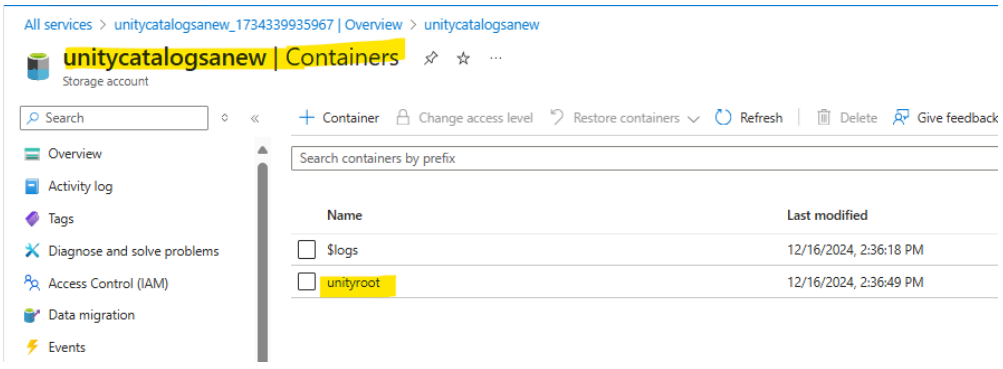
Transition from a local Hive Metastore to Databricks Unity Catalog for centralized metadata management and improved data governance.

We will first create a new databricks workspace “tt-hc-adb-ws”, select the “Premium (+ Role-based access controls)” while creating the workspace.

To configure the Metastore for the Unity catalog follow below steps:

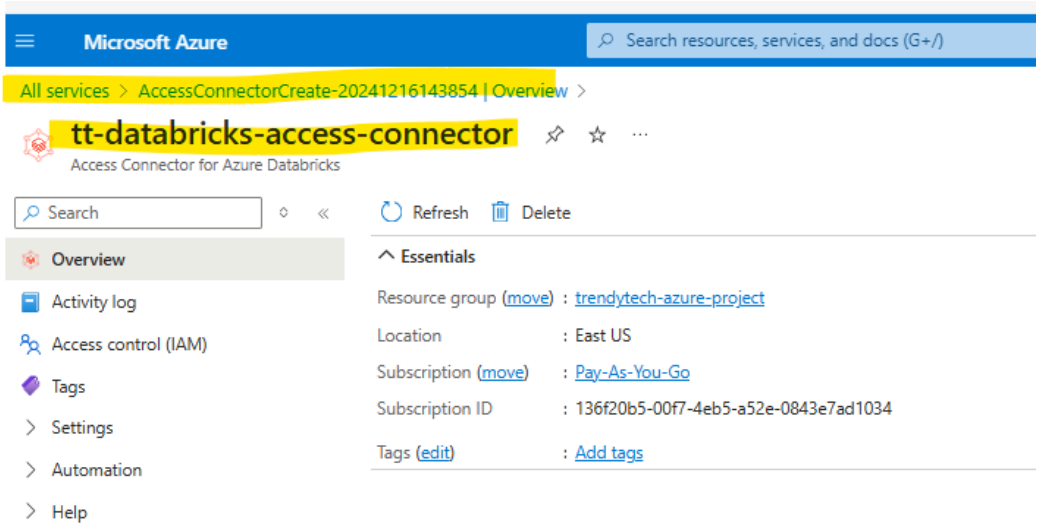
Step 1: Configure Storage for your Metastore - ADLS gen2 table -

Created a storage account for ADLS Gen2.- “ttunitycatalogsa”. Create a storage container that will hold your unity catalog metastore's metadata and managed data. container - unityroot

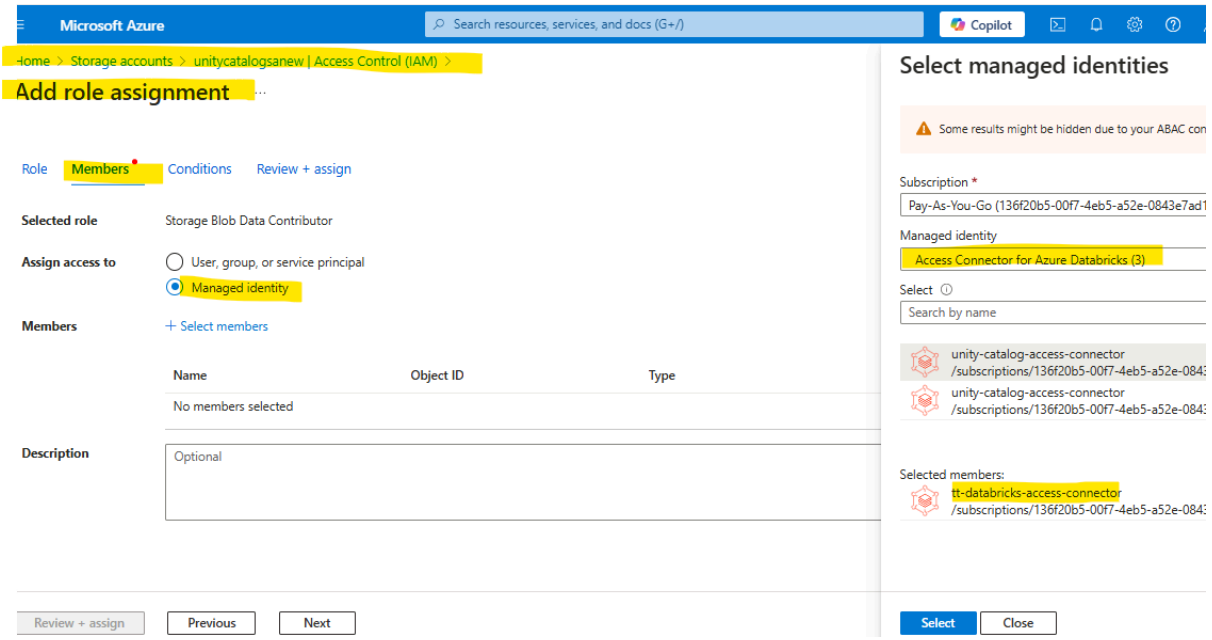


2. In Azure, create a databricks access connector that holds a managed identity and give it access to the storage container.

Go to “Access Connector for Azure Databricks” => create databricks access connector “ttdatabricksaccesscon”



Then Go to your storage account and click on access control (IAM)=> click on add -> role assignment -> storage blob data contributor -> in managed identity select the databricks connector that we have created.

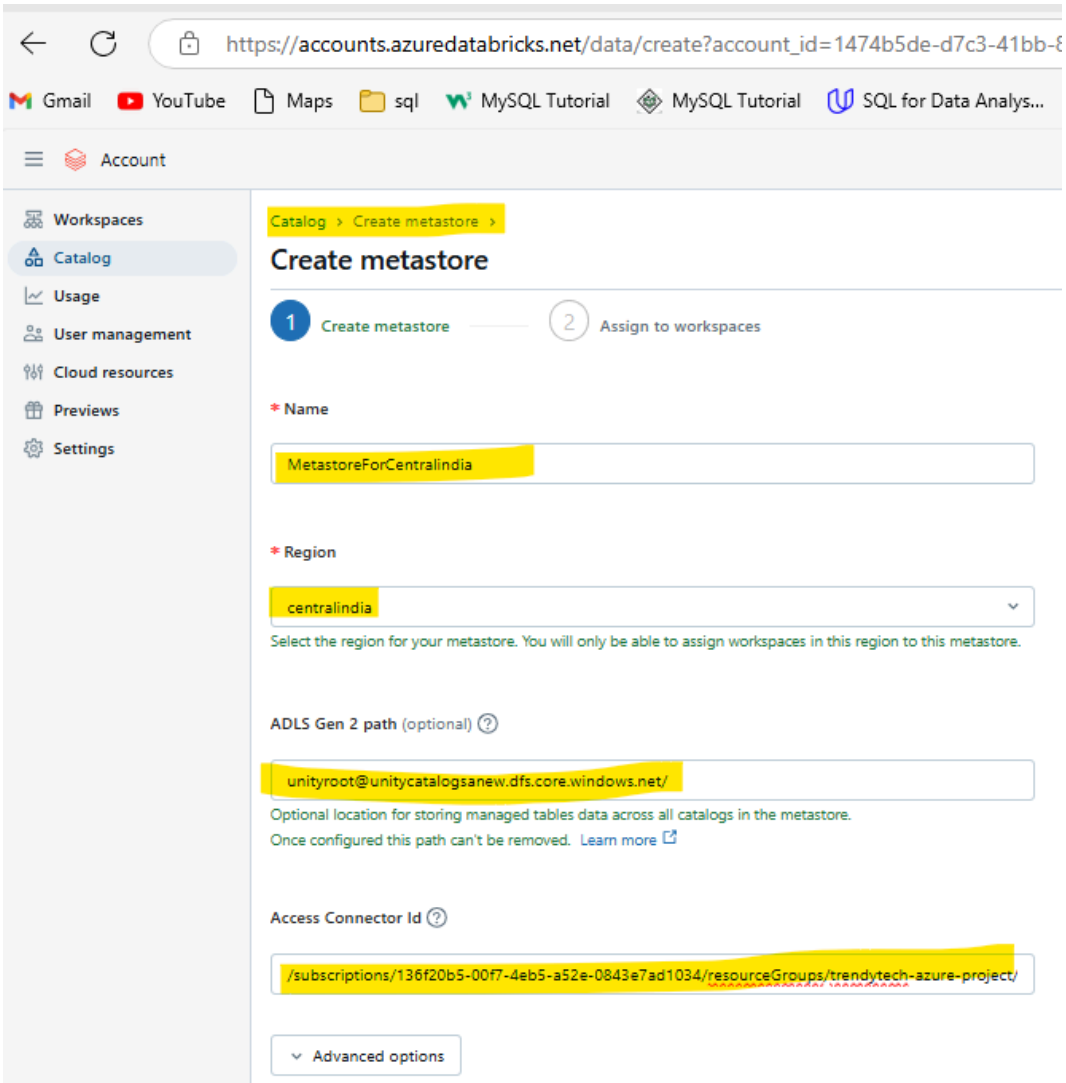


Save it and then review and assign it.

Step 2: You will go to “https://accounts.azuredatabricks.net” in your browser. & there we will create the metastore.

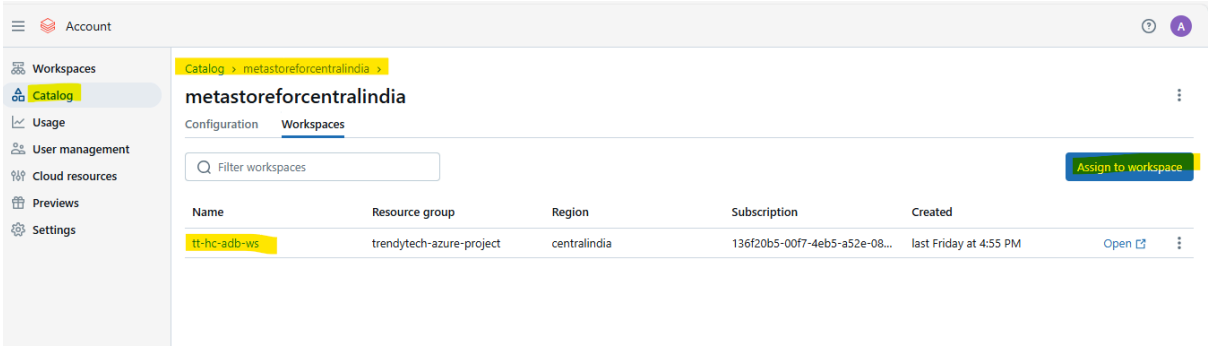
Clicked on Catalog => create metastore -> region should be the same where you have the storage account and the workspaces -> storage account path:
Ex:
unityroot@ttunitycatalogsa.dfs.core.windows.net/

-> access connector id (mention the access connector id)

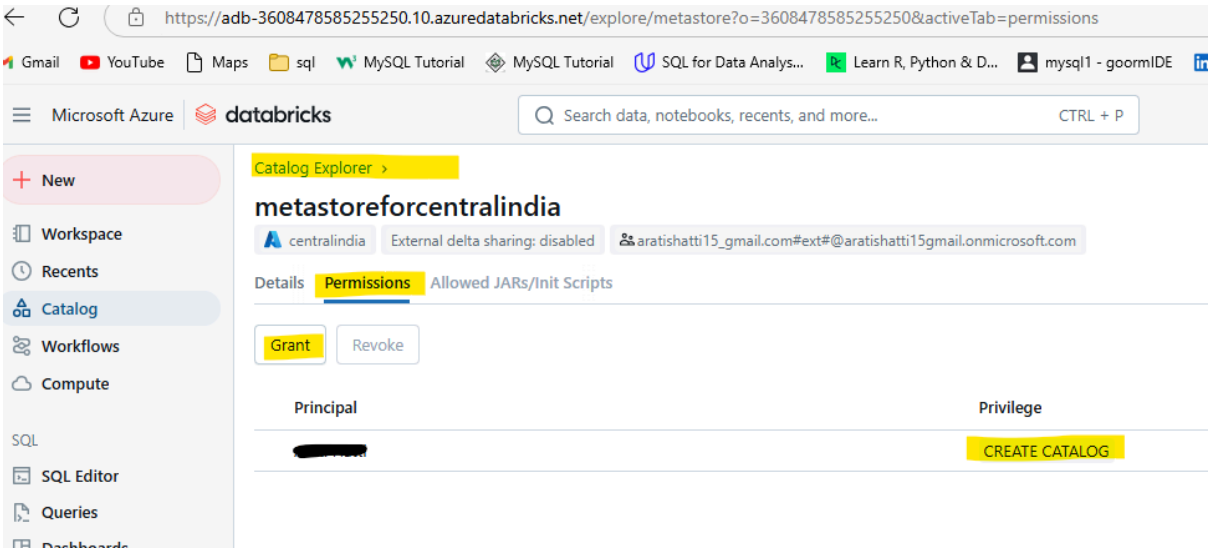


Also we will assign this metastore to our workspace.

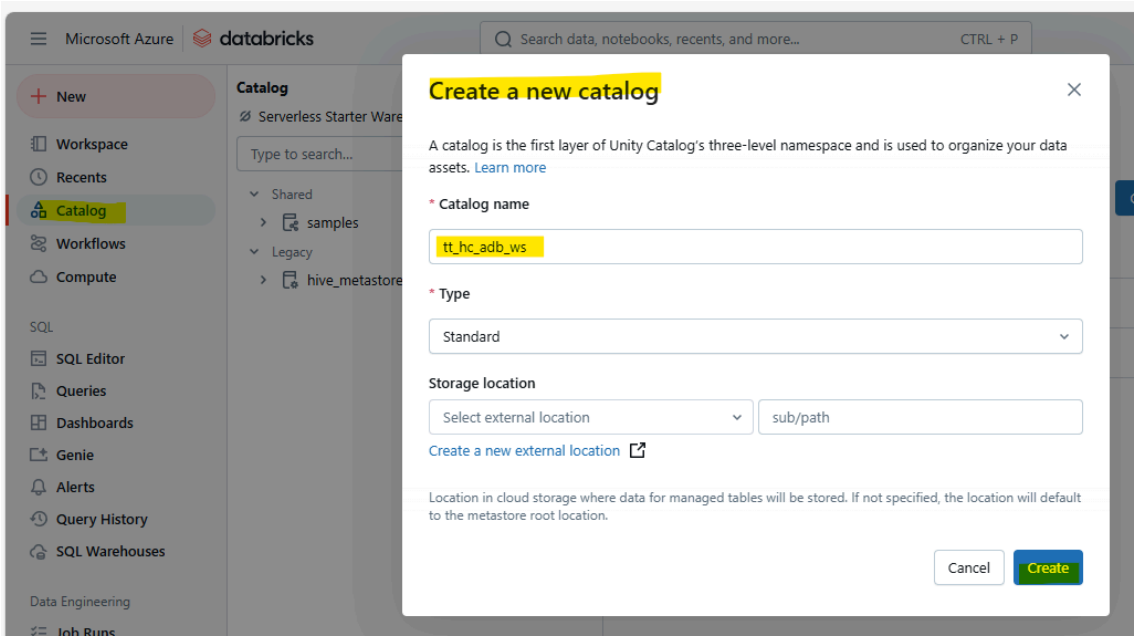
Go to Catalog => select the recently created metastore => click on Assign to workspace => select the workspace that you want to assign.

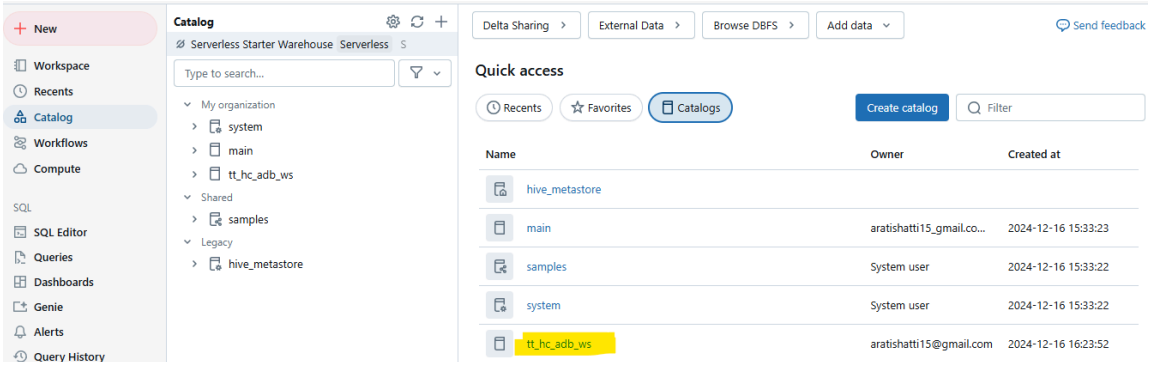


And we will assign “create catalog” permission to the user for this metastore if you are not getting the option to create a new Catalog.



Now we will create the catalog “tt_hc_adb_ws”.

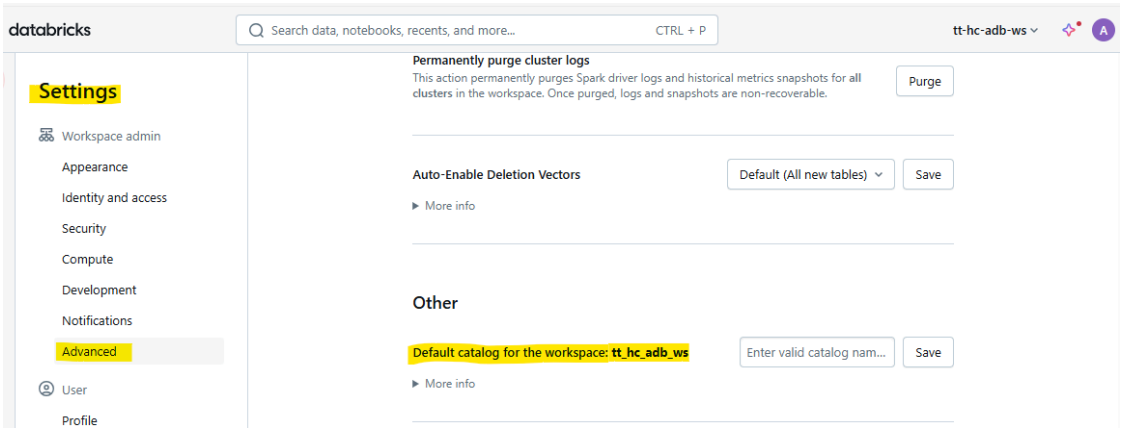




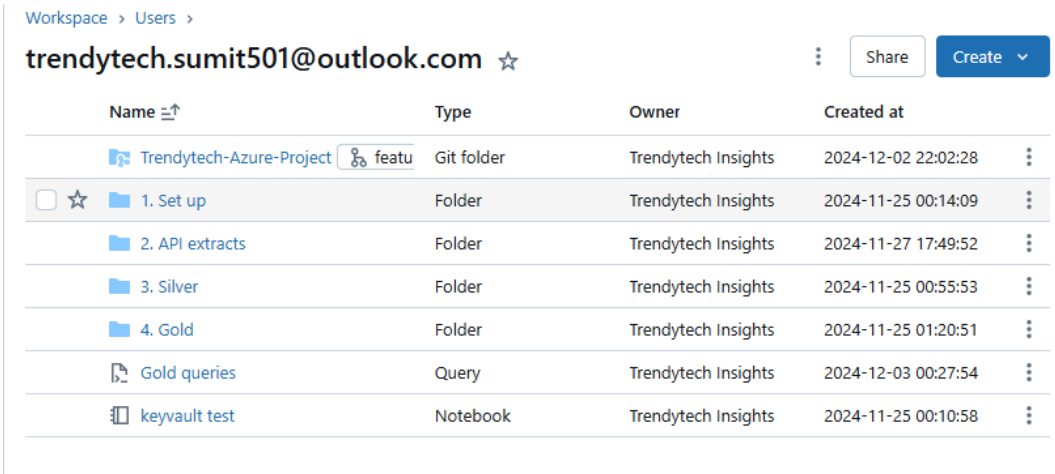
After creating this catalogue, set this catalogue as a default catalog for the workspace “tt-hc-adb-ws”. For this follow below steps:

Open your Databricks Workspace.
Navigate to Admin Settings → Workspace Settings.
Set the Default Catalog to your desired catalog (e.g., tt_hc_adb_ws).

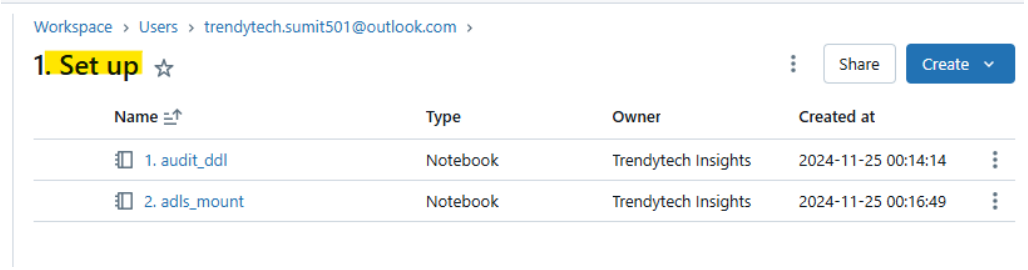
Refer below screenshot for more clarity.



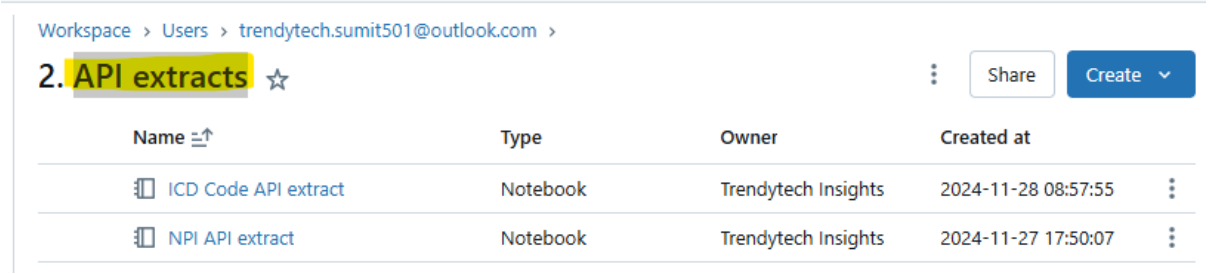
Also to organize the notebook we will create the folder as shown below.



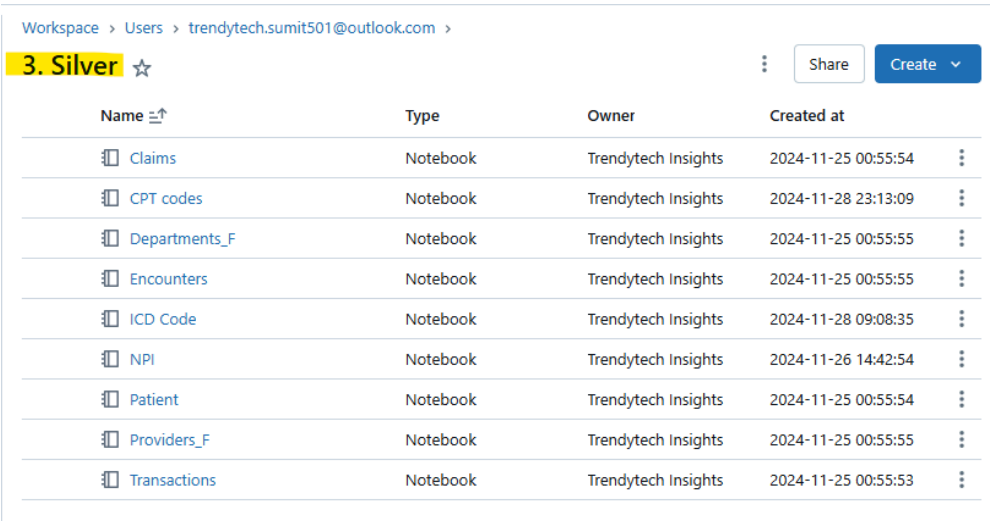
1. Set up:



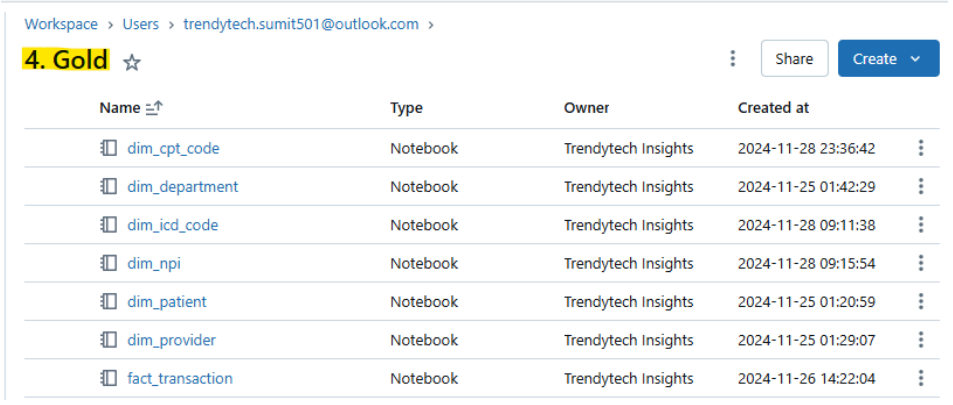
2. API extracts



3. Silver

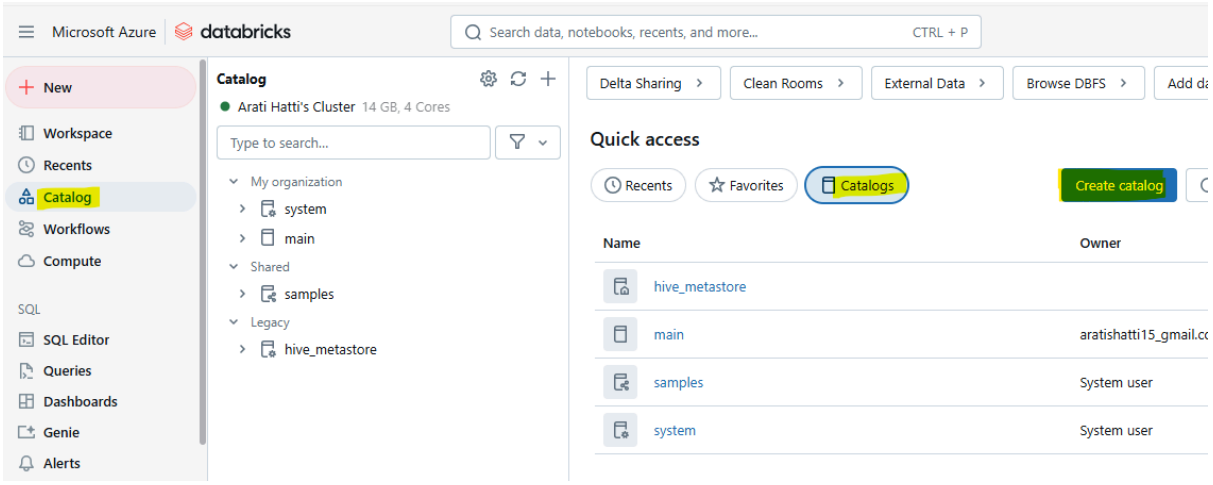


4. Gold



Note: After creating the databricks workspace, enable the DBFS.

Create the catalog “tt-hc-adb-ws” as shown below.



We will create the audit database using as shown below

Note: Before proceeding ahead we will create the key vault in azure and will store key for our ADLS gen2 storage in it. Refer below steps for more clarification.

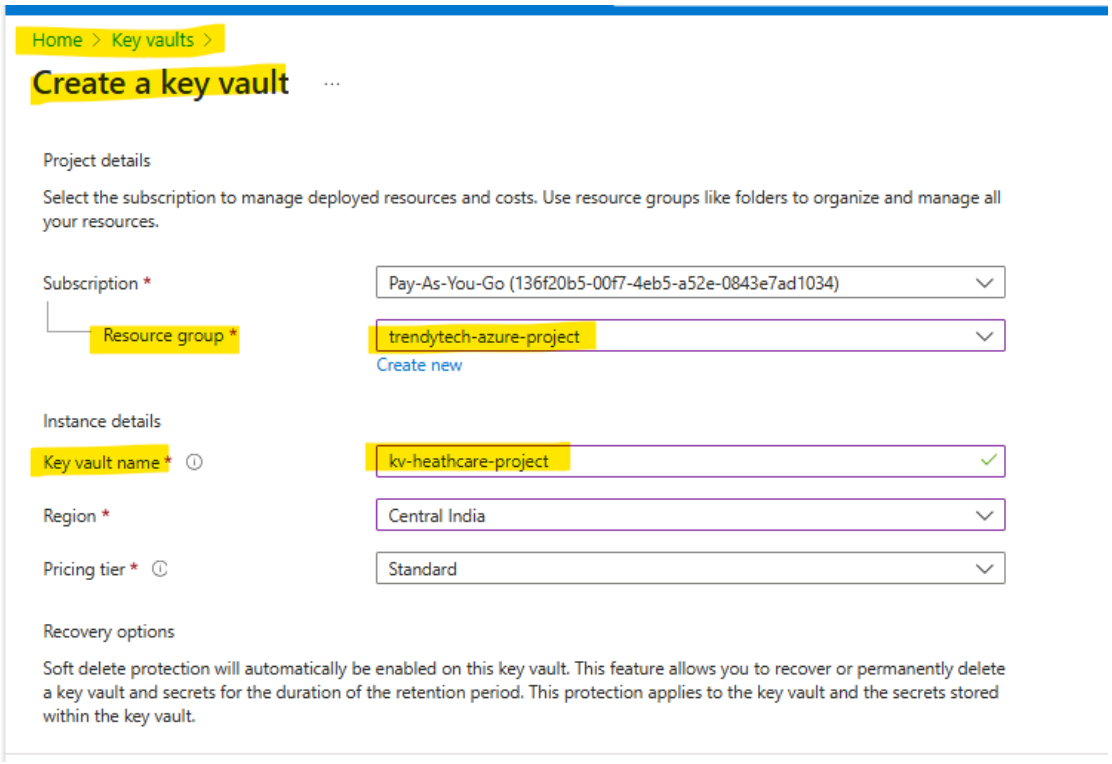
Key Vault creation in Azure:

Steps to Store Key in Key Vault and Create Secret Scope in Databricks:

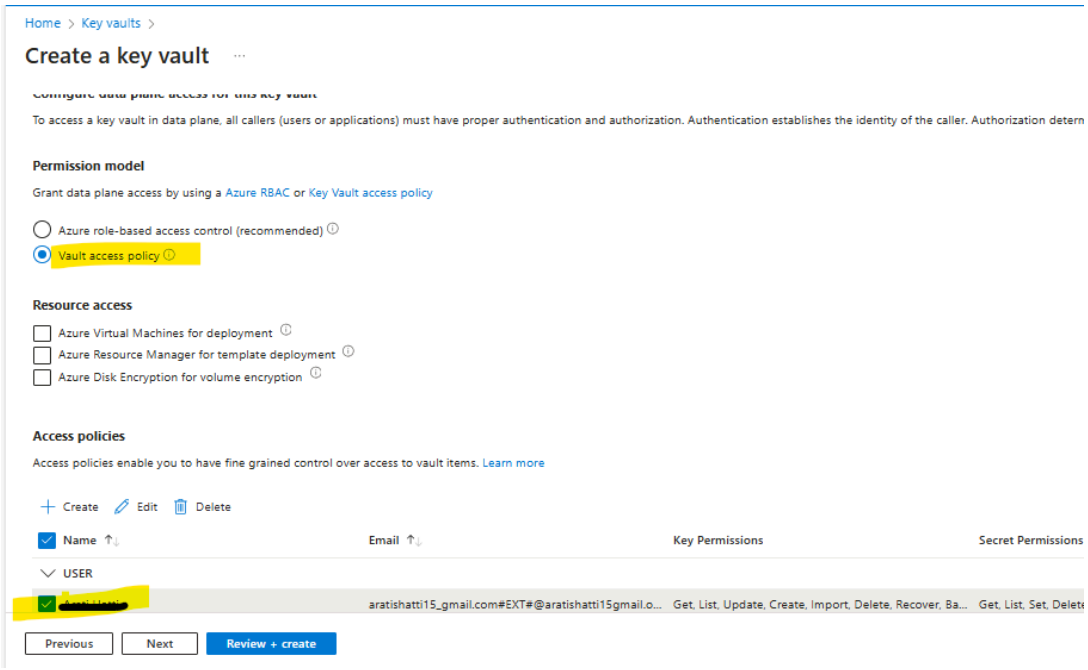
1. Store the Storage Account Access Key in Azure Key Vault

Create an Azure Key Vault (if not already created):

Go to the Azure Portal. => Search for "Key Vault" and click Create.=> Fill in the necessary details (Resource Group, Key Vault Name, Region) and click Review + Create.
Refer below screenshot for more details.



In Access configuration select option vault access policy option and select define permission for your username.

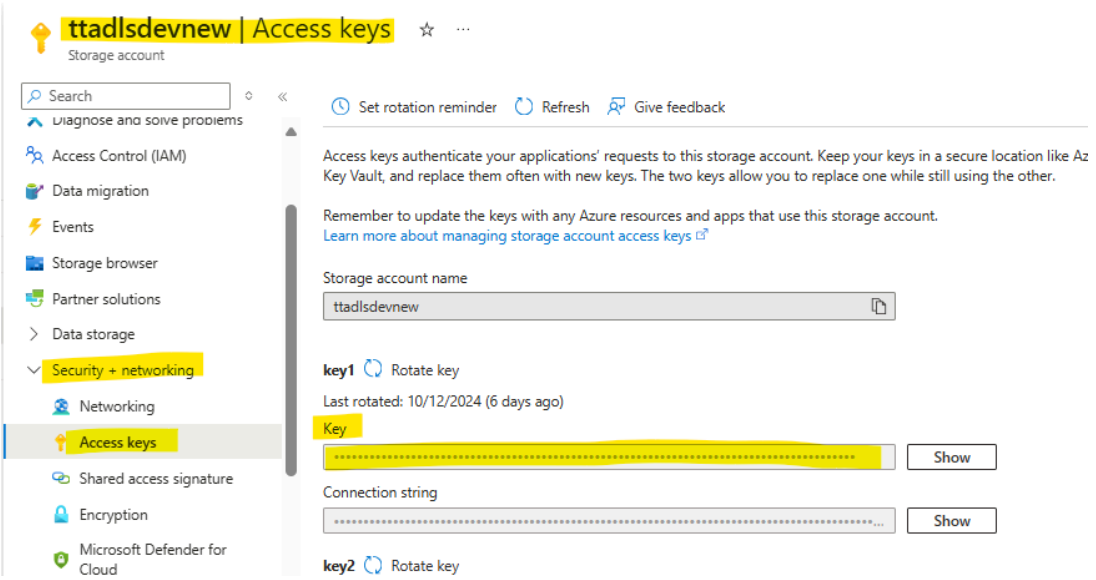


Then create the key vault.

Add the Storage Account Access Key to the Key Vault, to add follow below steps:

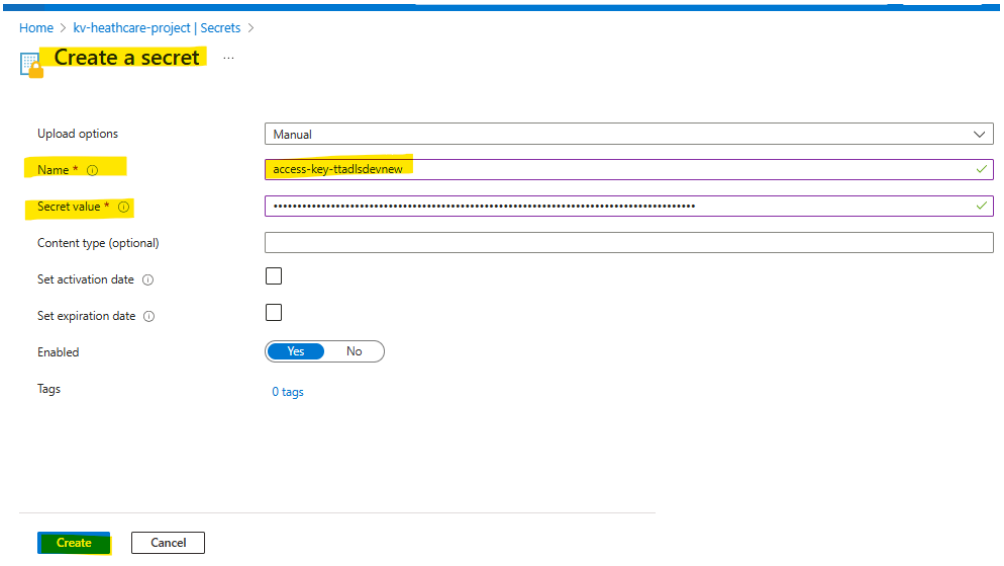
First get the storage account key, to get the storage account access key follow below steps:

=> Go to your Azure Storage Account => Security & Networking> Access Keys.
=> Copy the key under the "Key1" or "Key2" section.



To store keys in the key vault, navigate to your Key Vault.

Click Secrets > + Generate/Import. => Enter the following details:
Name: Provide a descriptive name for the secret (e.g., adls-access-key).
Value: Copy the access key of your storage account and paste it here.
=> Click Create.

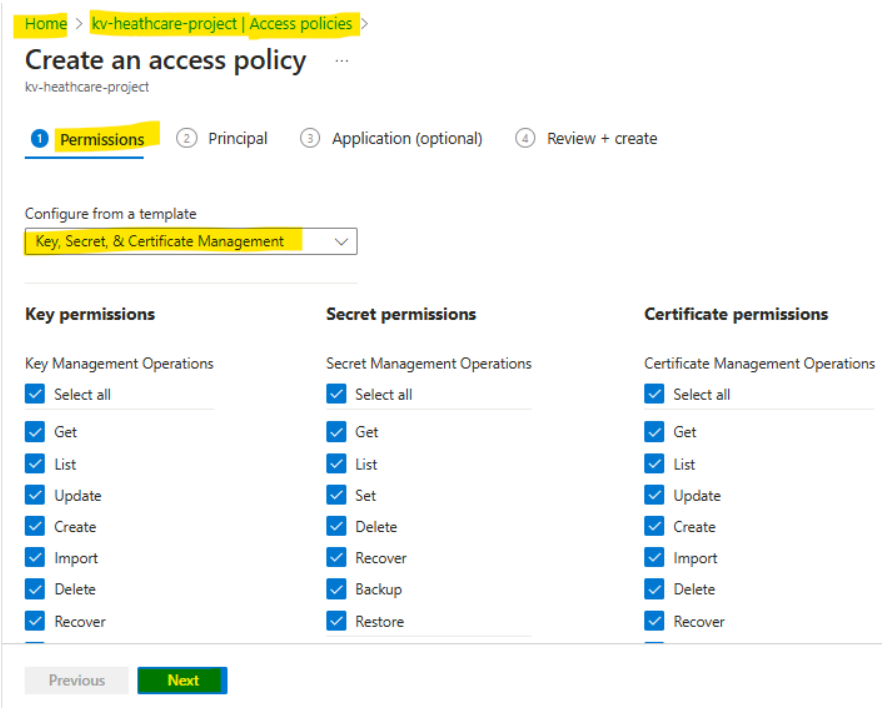


2. Assign Key Vault Access to Databricks

To allow Databricks to access the Key Vault:

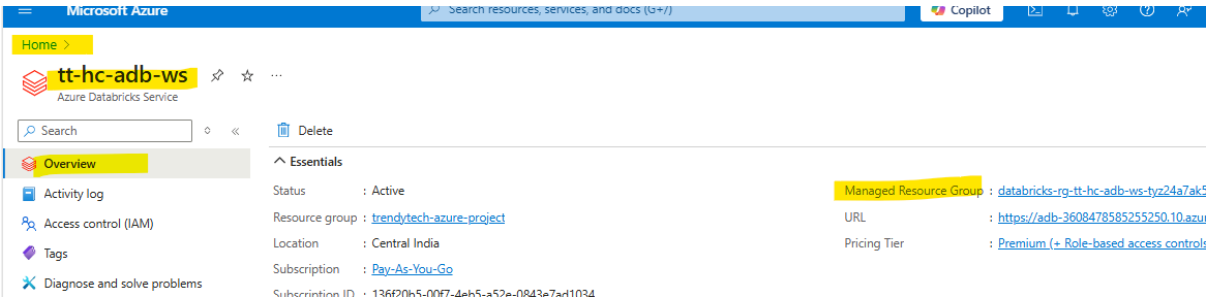
Go to the Key Vault => Navigate to your Key Vault in the Azure Portal. => Click on Access Policies > + Add Access Policy. => Grant Permissions:

Template: Select Secret Management.

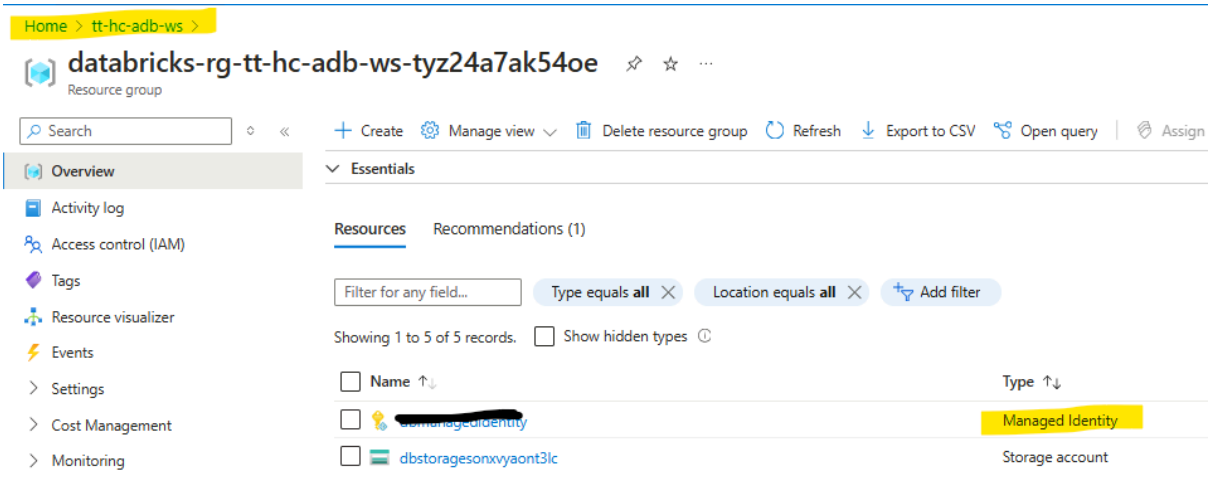


Principal: Add your Azure Databricks Managed Identity.

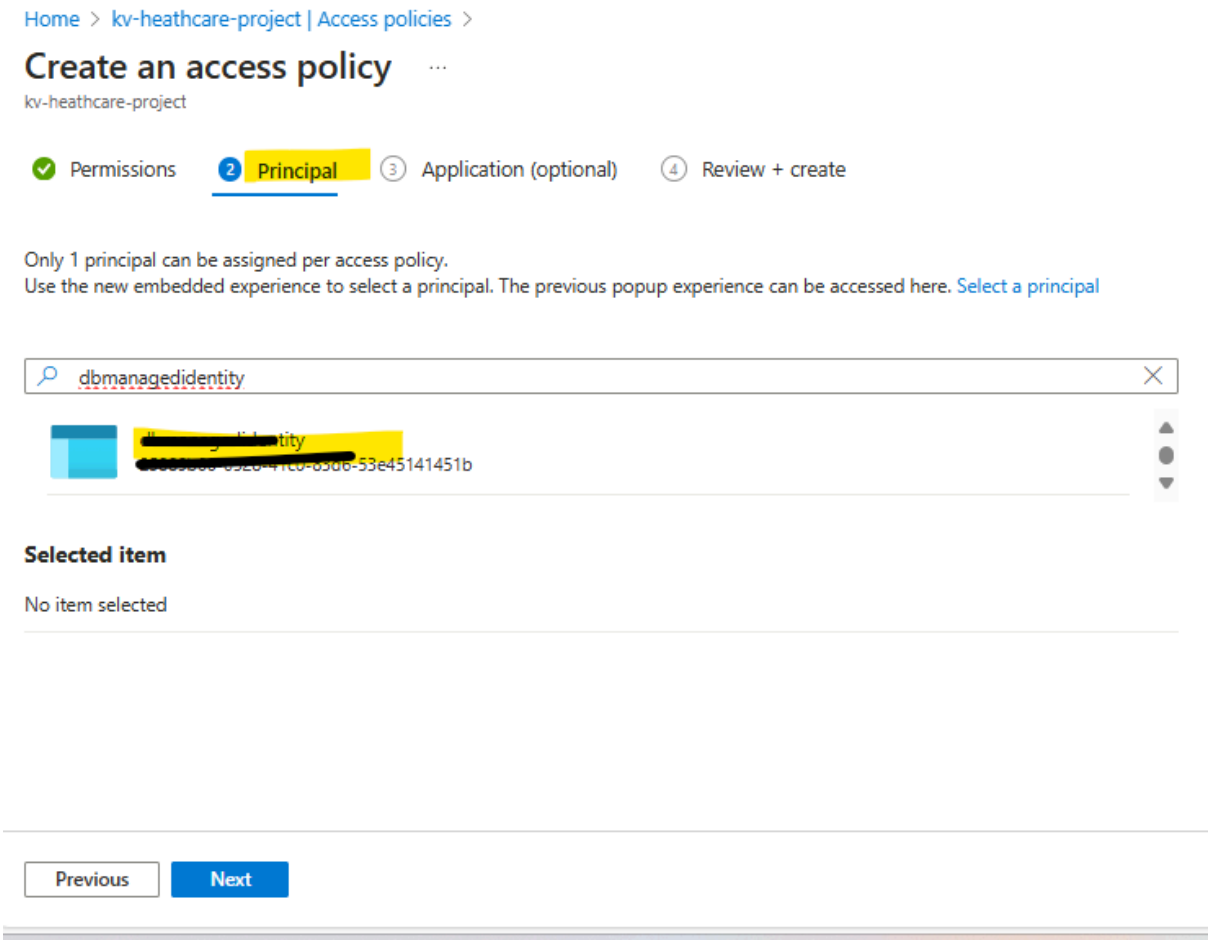
You can find the Managed Identity from the Databricks Resource in Azure Portal under Managed Identity.



Now select this managed resource group and check the managed identity and define policy for it.



Save the Policy:



Click Add, then Save the changes.

3. Create a Secret Scope in Databricks

Once the secret is stored in Key Vault, create a secret scope in Databricks.

Open Databricks Workspace:

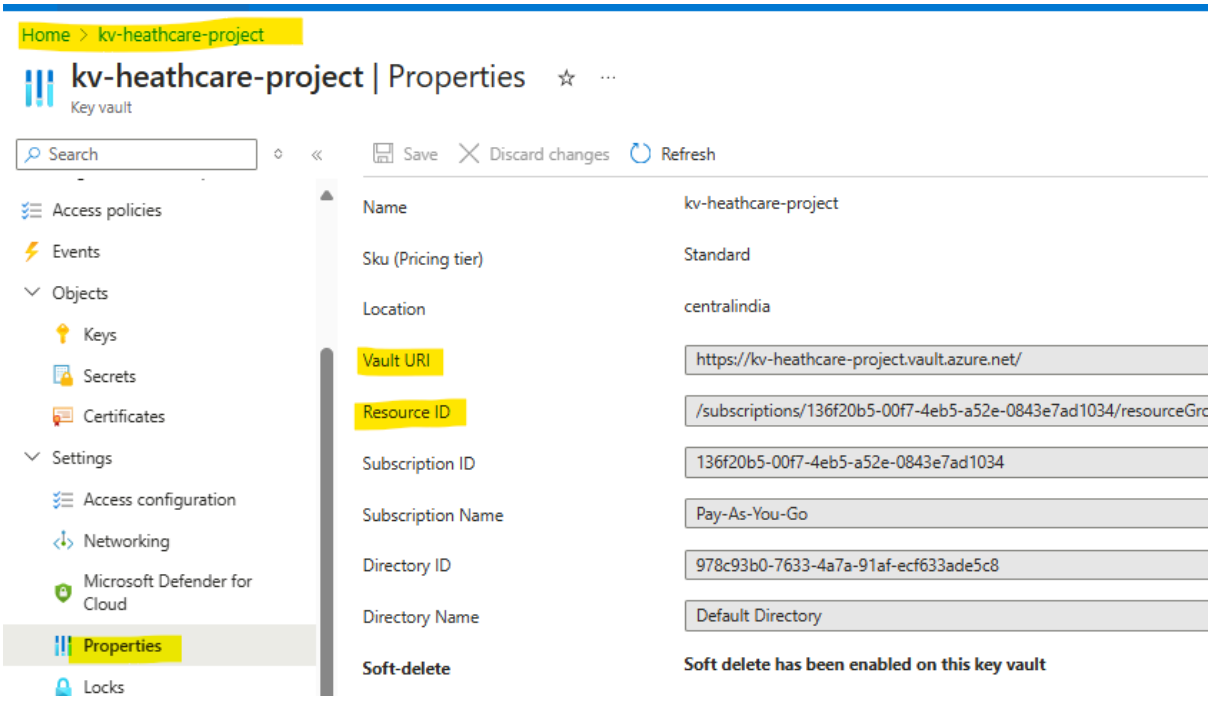
Go to your Azure Databricks Workspace.

While creating the secret scope, edit the url till .net and add "#secrets/createScope"

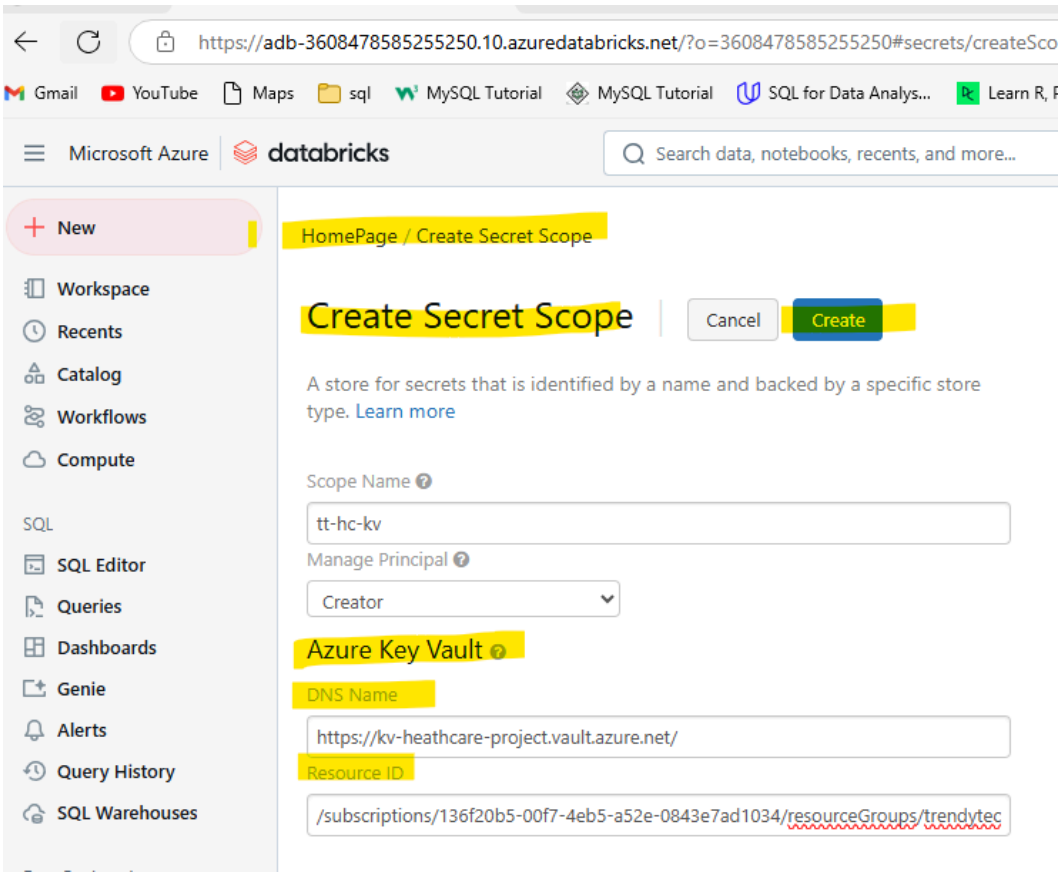
Create a Secret Scope Linked to Key Vault:

Enter the following:
Scope Name: Give a name to the scope (e.g., adlsdevnew).
Scope Backing: Select Azure Key Vault.
DNS Name: Enter the Key Vault URL.
To get these details go to key vault => Setting => Properties

Refer below screenshot for more clarity.



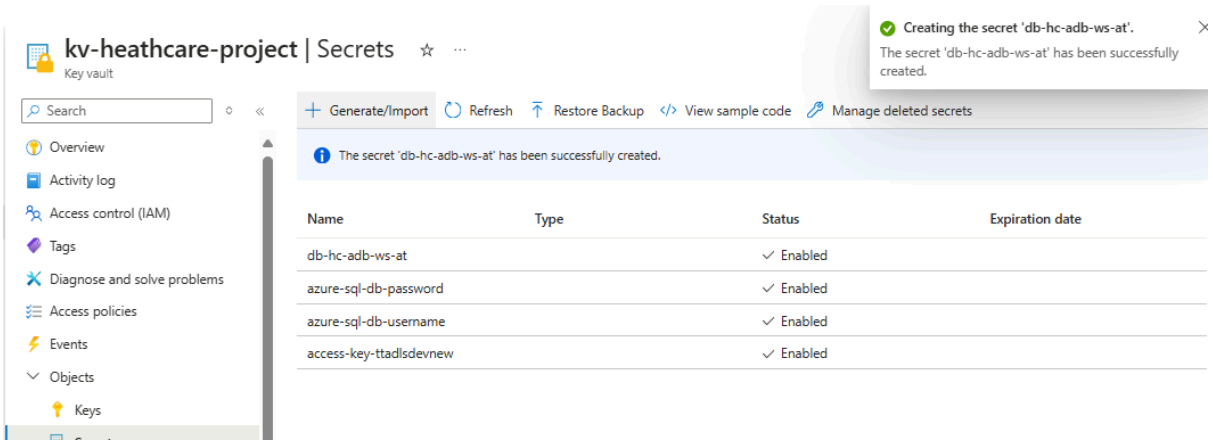
Now provide these details in scope creation and create the scope.



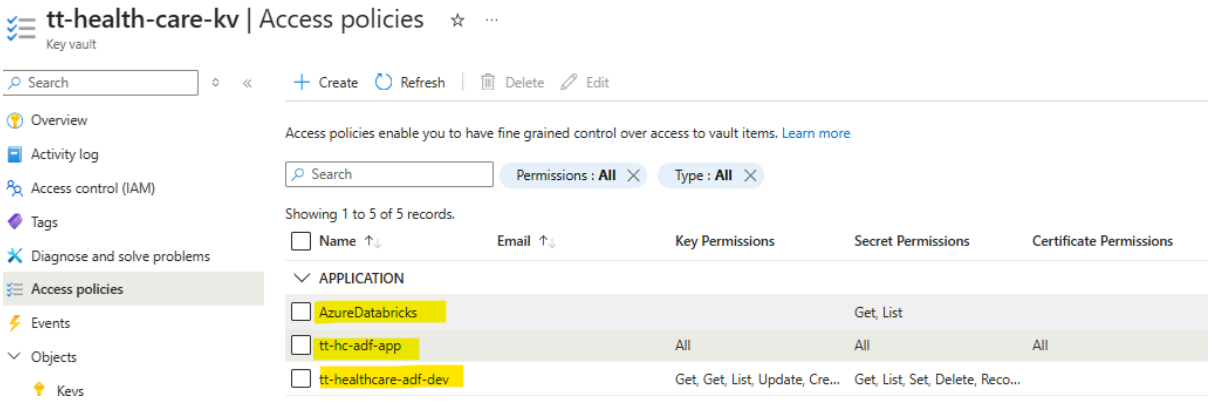
Verify the Secret Scope:

Similarly first we store credentials for sql database (Username and password) and databricks also (In case of Databricks we will store access token).

To get access token in Databricks got to Setting => Developer => Access token => Generate Access token



Now for ADF and ADLS gen we will create application and will provide permission (You can Provide all the permission) as shown by Sumit Sir in the video

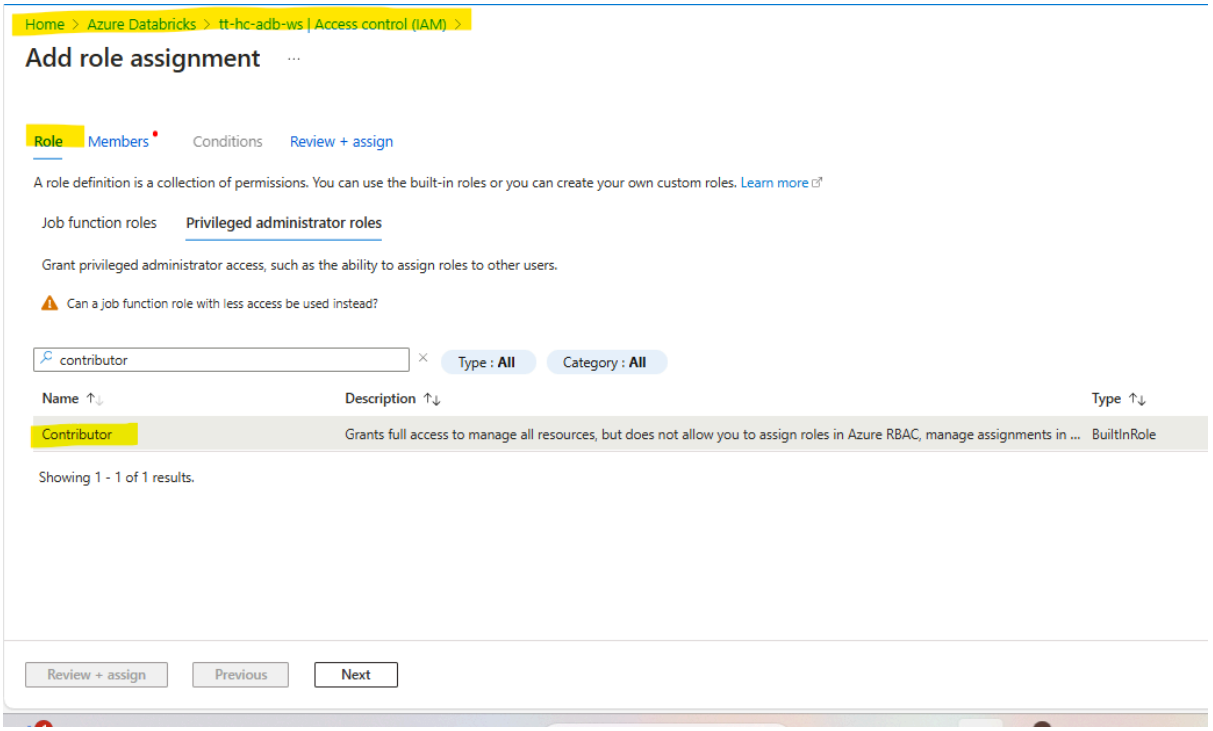


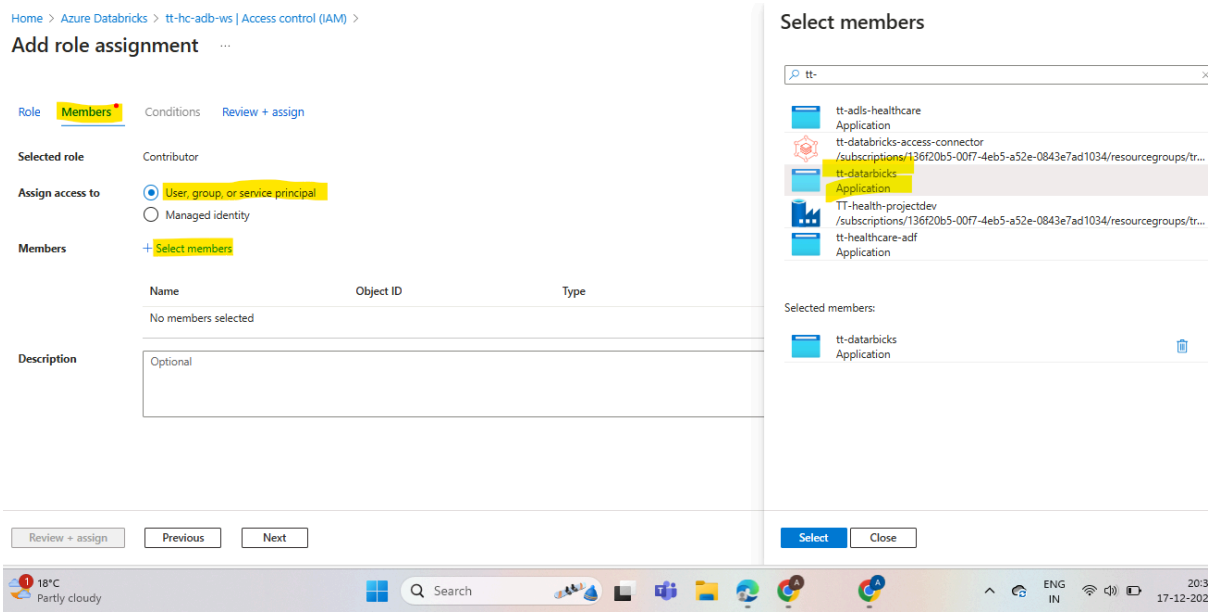
To perform role assignments in Azure Databricks and Azure Data Factory (ADF), follow the steps outlined below.

For Azure Databricks:

To assign the role : Go to resource => IAM => Add => Role Assignment.

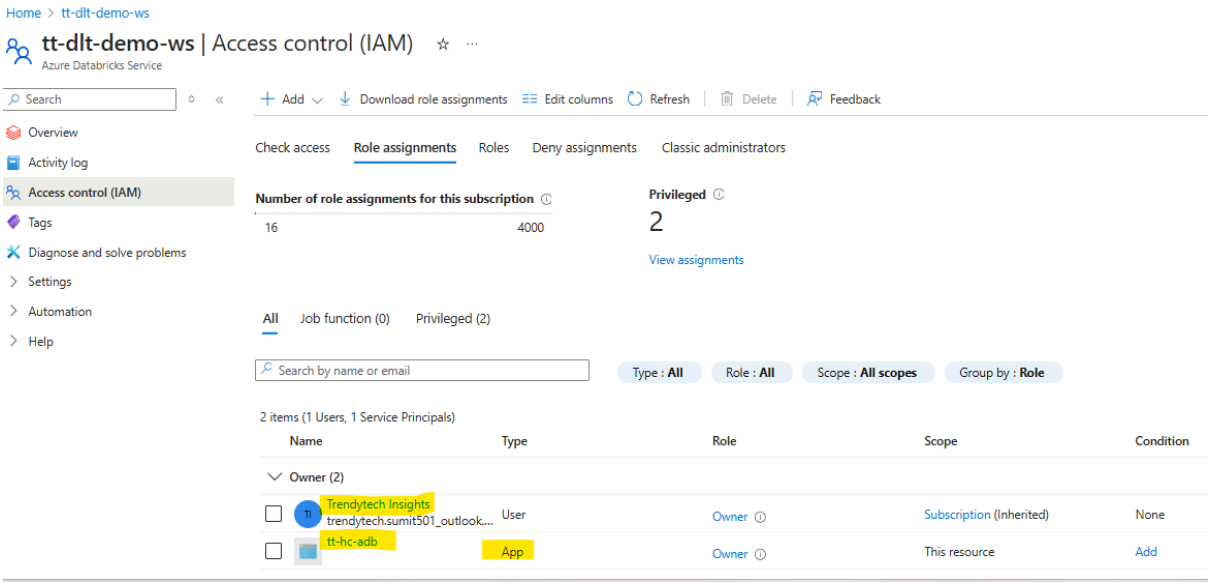
Define Contributor role for the application that is created for databricks as shown below.





Note: In condition give select “Allow user to assign all roles (highly privileged)”

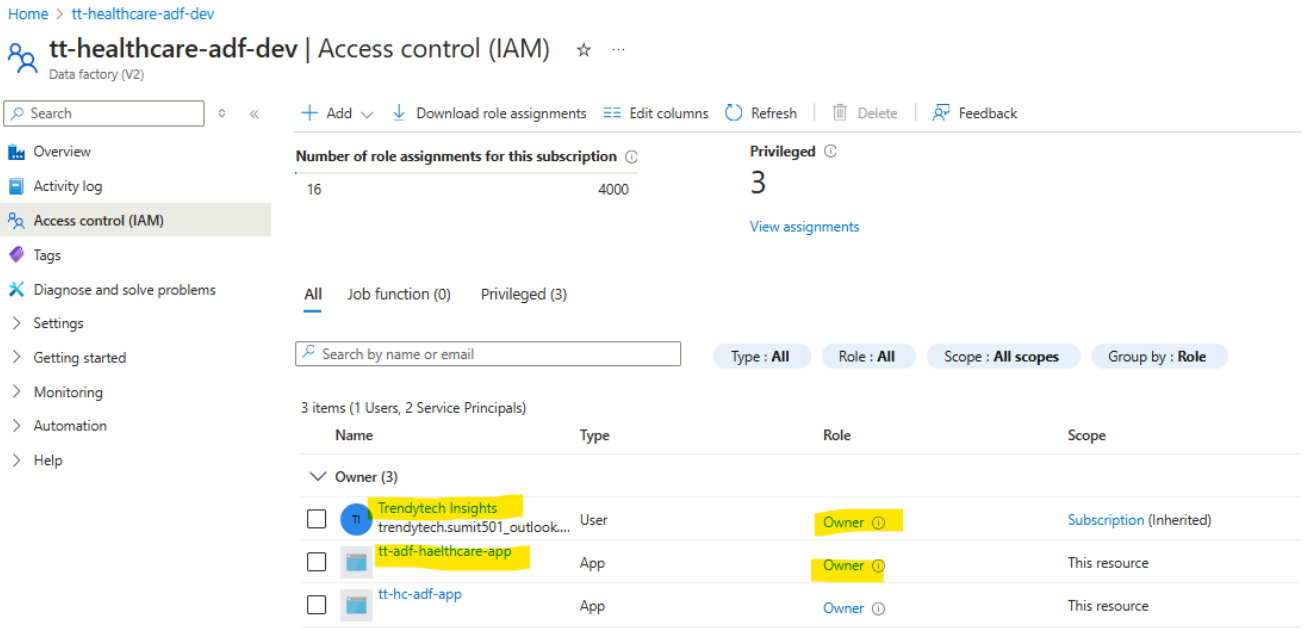
Also for your username define the owner role if it is not present as shown below.



For ADF:

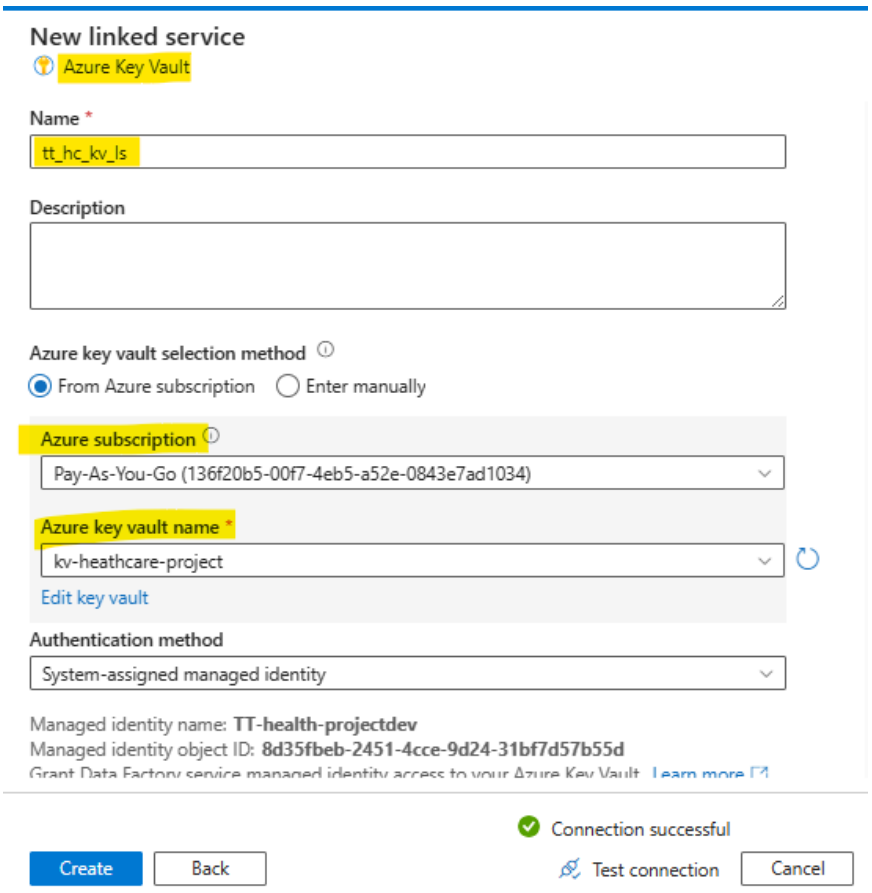
To assign the role : Go to resource => IAM => Add => Role Assignment.

Define Contributor role for the application that is created for databricks as shown below.

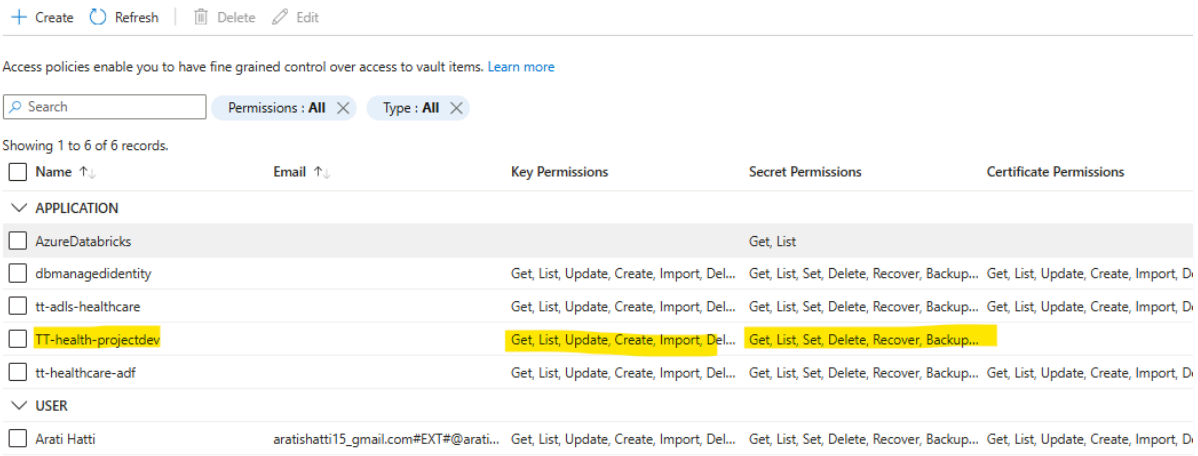


We will use the code in the adls_mount notebook to mount the containers in our storage account.

Now we will create a linked service for key vault and Databricks as shown below.

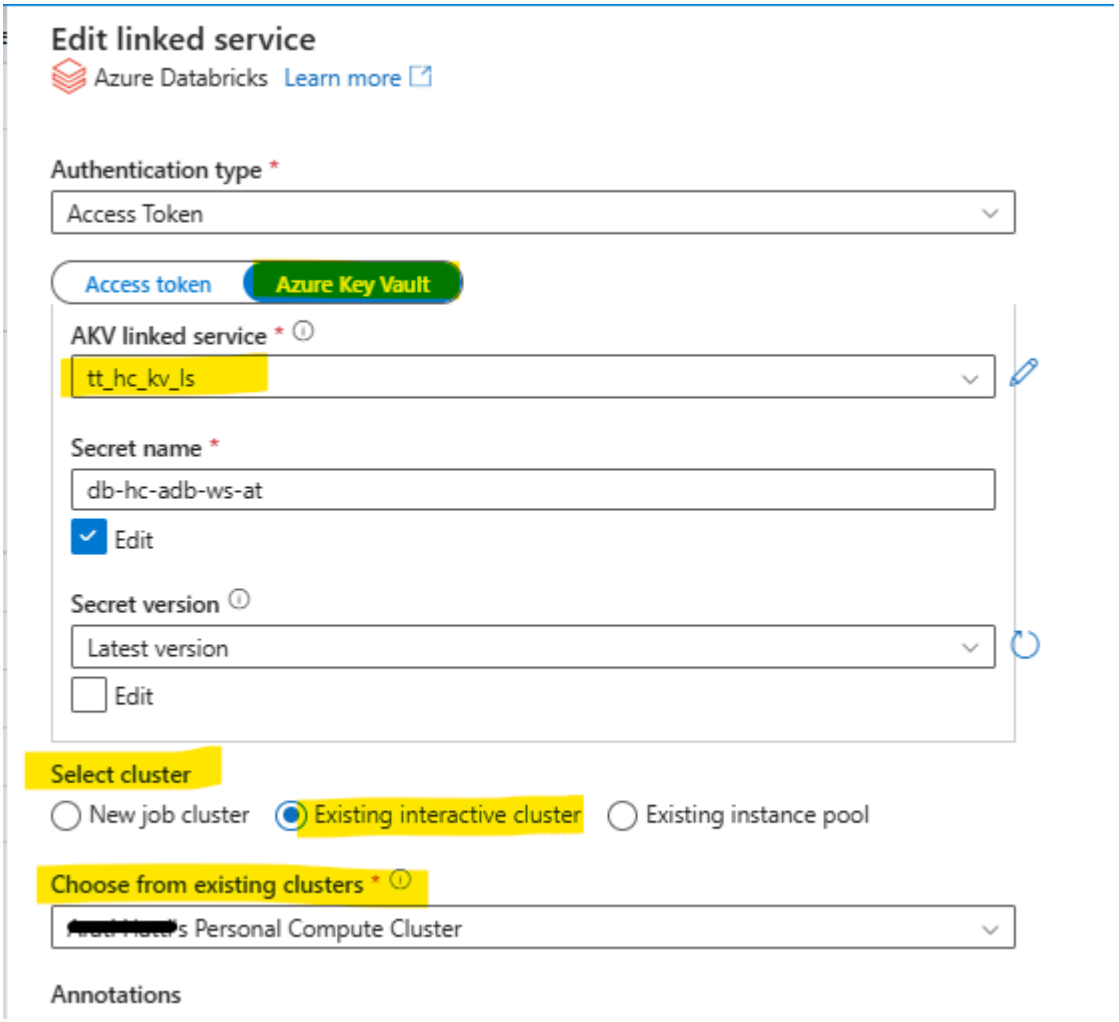


Before creating the linked services, grant access permissions to the Data Factory service principal under the access policies in Key Vault and provide key and secret access permission.



After this you will be able to access secrets while creating linked services.

Note: While creating Linked service for Databricks select the existing cluster option as shown below.

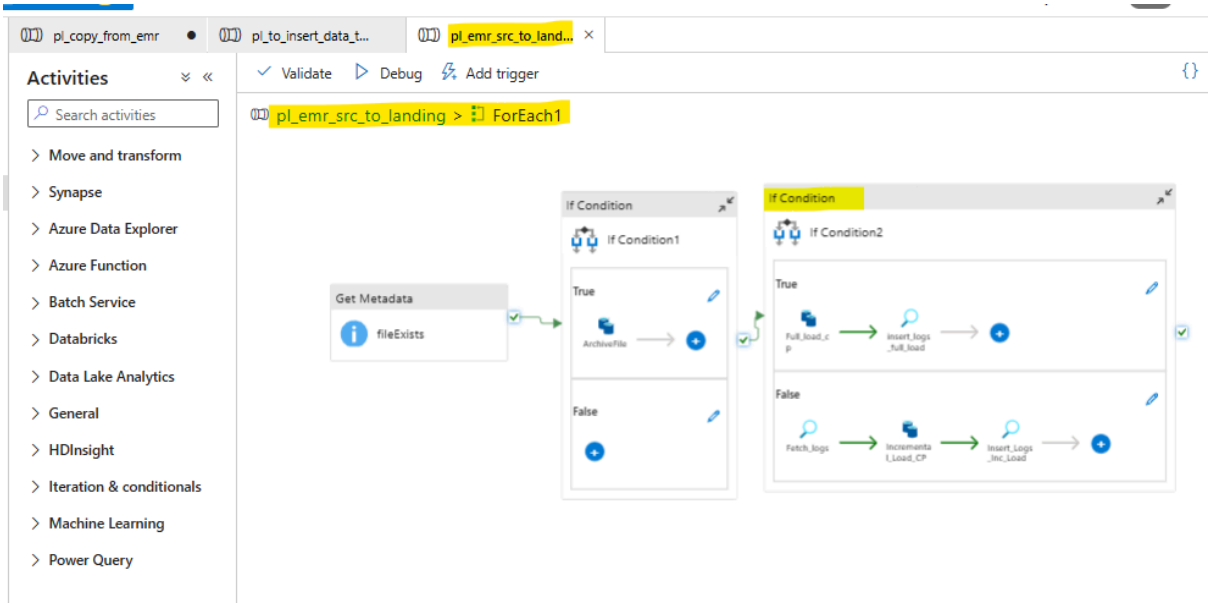


We will update the linked services for ADLS Gen2, sql database and now instead of providing the storage account key directly, we will select the secret scope for respective ADLS gen storage, SQL DB, etc. as shown by Sumit sir in the video.

How to implement active and inactive Flag.

1. **Creating new pipeline - pl_copy_from_emr**

To create the pipeline "pl_copy_from_emr" and we will copy the "2nd If condition (having condition @equals(item().loadtype, 'Full'))" for copying the data into this pipeline.



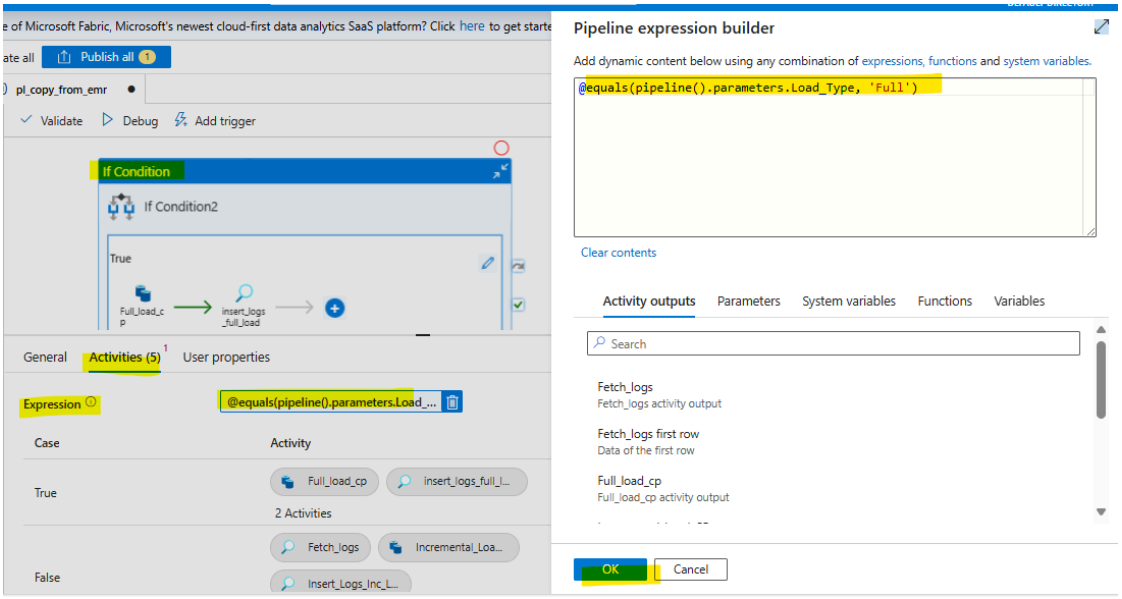
In the pipeline pl_emr_src_to_landing, when using @equals(item().loadtype, 'Full'), it works because the copy activity is directly inside the For Each loop, and item() refers to the current element being processed during the loop's iteration.

However, when you move the copy activity to a child pipeline (triggered using the Execute Pipeline activity) and place the If condition in the parent pipeline's For Each loop, the child pipeline no longer has direct access to item(). Instead, you need to pass the loop data (e.g., loadtype) as a parameter to the child pipeline.

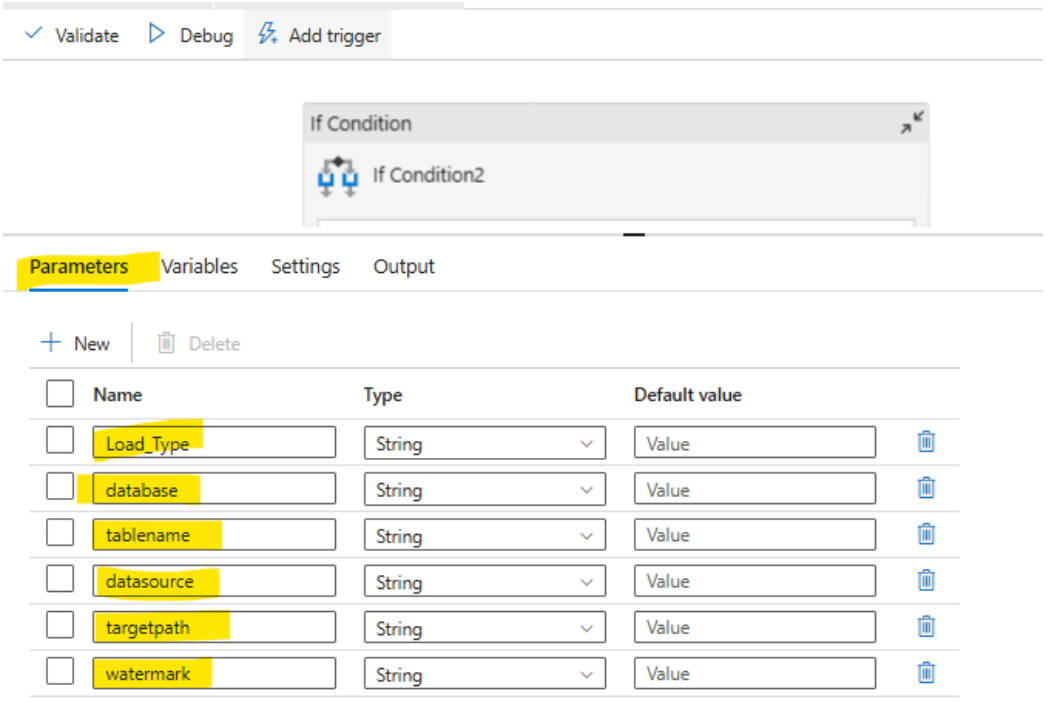
This is where @equals(pipeline().parameters.Load_Type, 'Full') comes into play. The parameter Load_Type is explicitly passed to the child pipeline when it's invoked, and the condition uses this parameter to determine the action. By referencing the parameter, the child pipeline can work independently while still receiving the necessary context from the parent pipeline.

So, in this pipeline update the Expression as shown below and also we have to add the parameter as shown below

```
@equals(pipeline().parameters.Load_Type,'Full' )
```



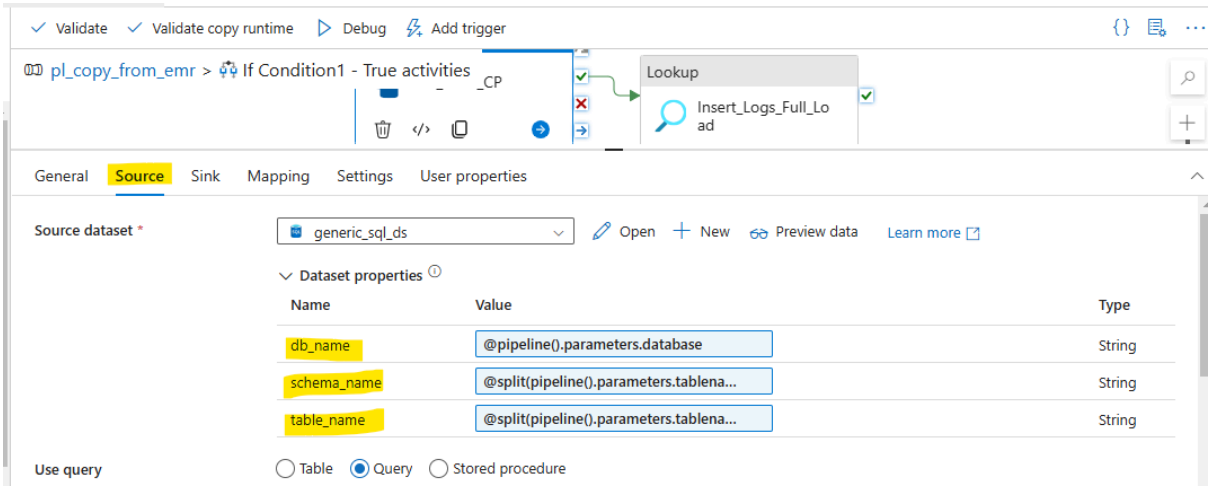
Parameter:



We will update the values for the source and sink variables for both the full load and incremental load copy activities. Since the variable values are the same for both, we will use the same variable, as demonstrated below.

Source:

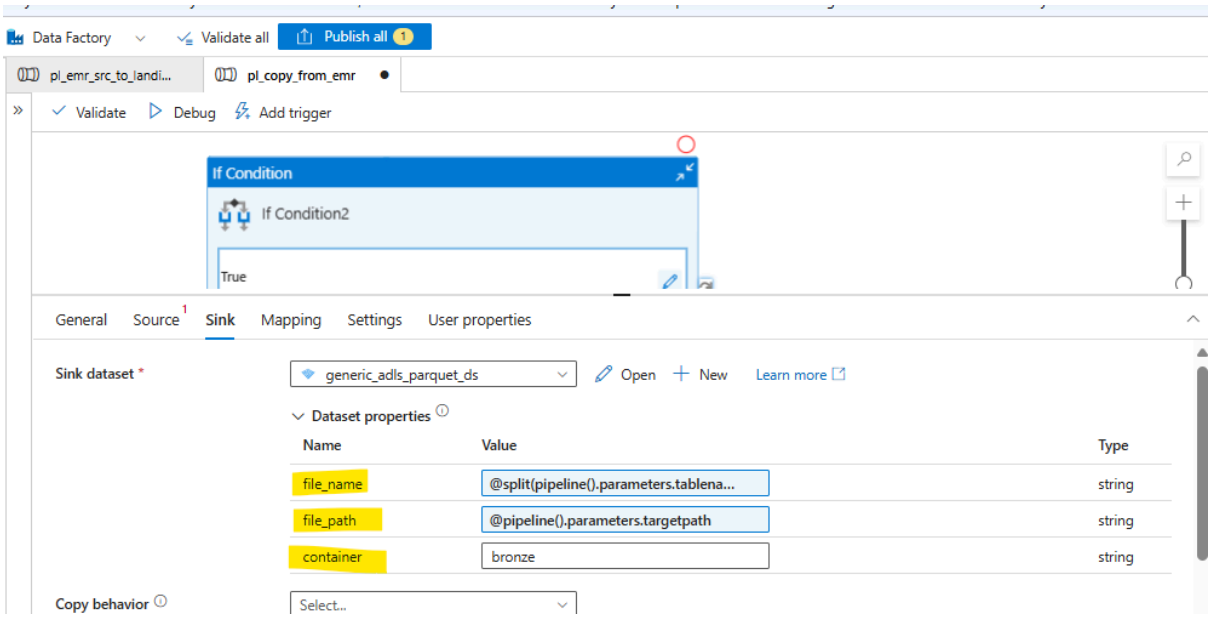
```
db_name - @pipeline().parameters.database
schema_name - @split(pipeline().parameters.tablename, '.')[0]
table_name - @split(pipeline().parameters.tablename, '.')[1]
```



Sink:

```
file_name - @split(pipeline().parameters.tablename, '.')[1]
file_path - @pipeline().parameters.targetpath
```

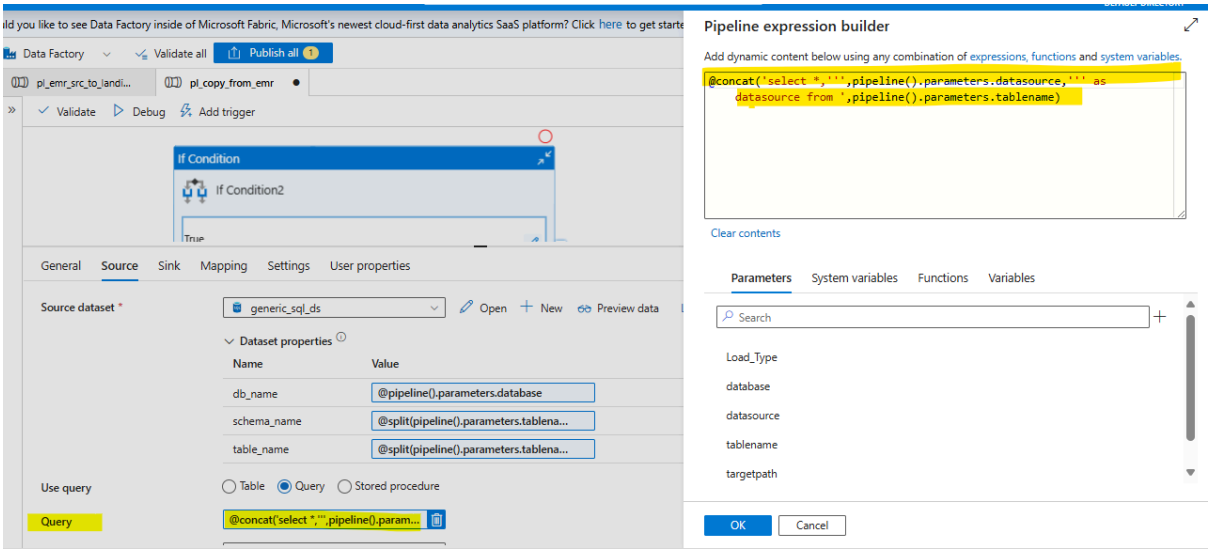
Container -bronze



Also we have to update the queries for Full load as shown below

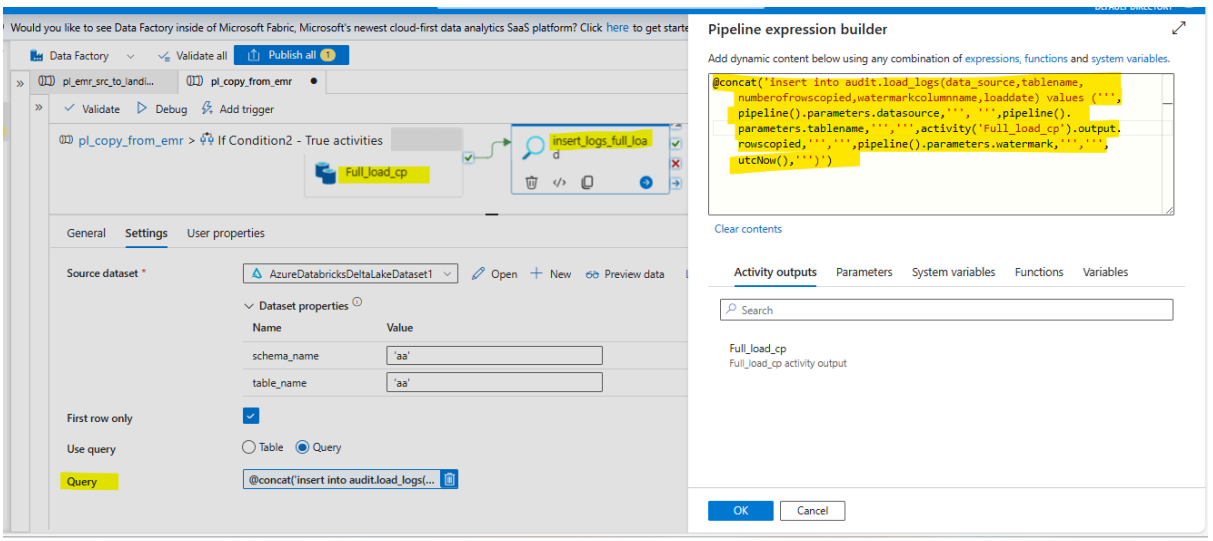
Full Load:

```
@concat('select *,''',pipeline().parameters.datasource,''' as
datasource from ',pipeline().parameters.tablename)
```



Additionally, we will update the lookup query to insert logs into the audit table for Full load as shown below.

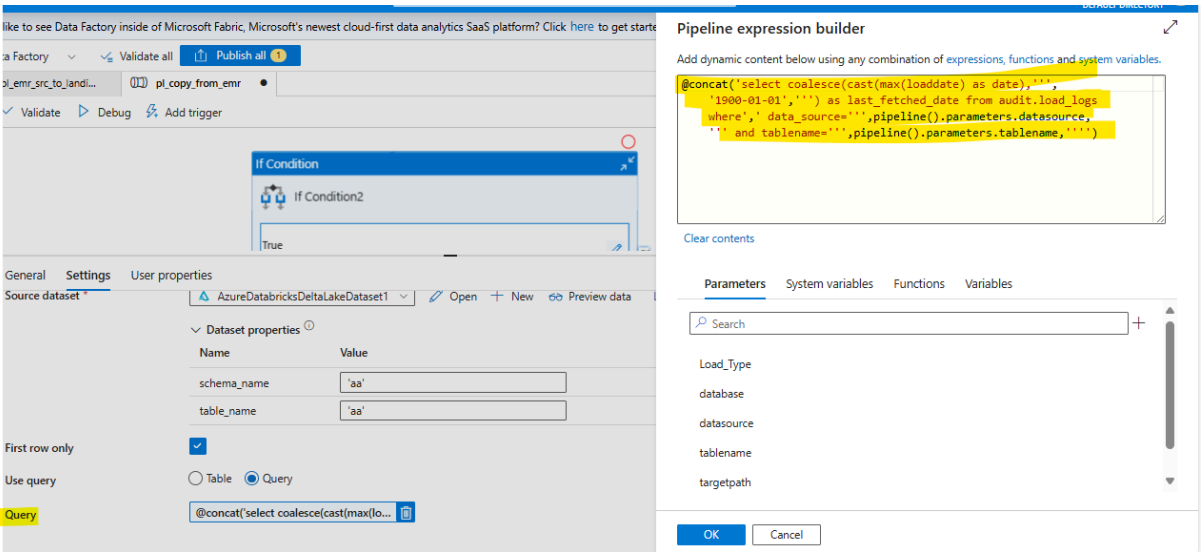
```
@concat('insert into audit.load_logs(data_source,tablename,numberofrowscopied,watermarkcolumnname,loaddate) values ('''',pipeline().parameters.datasource,'''','''',pipeline().parameters.tablename,'''','''',activity('Incremental_Load_CP').output.rowscopied,'''','''',pipeline().parameters.watermark,'''','''',utcNow(),''')')
```



Incremental load:

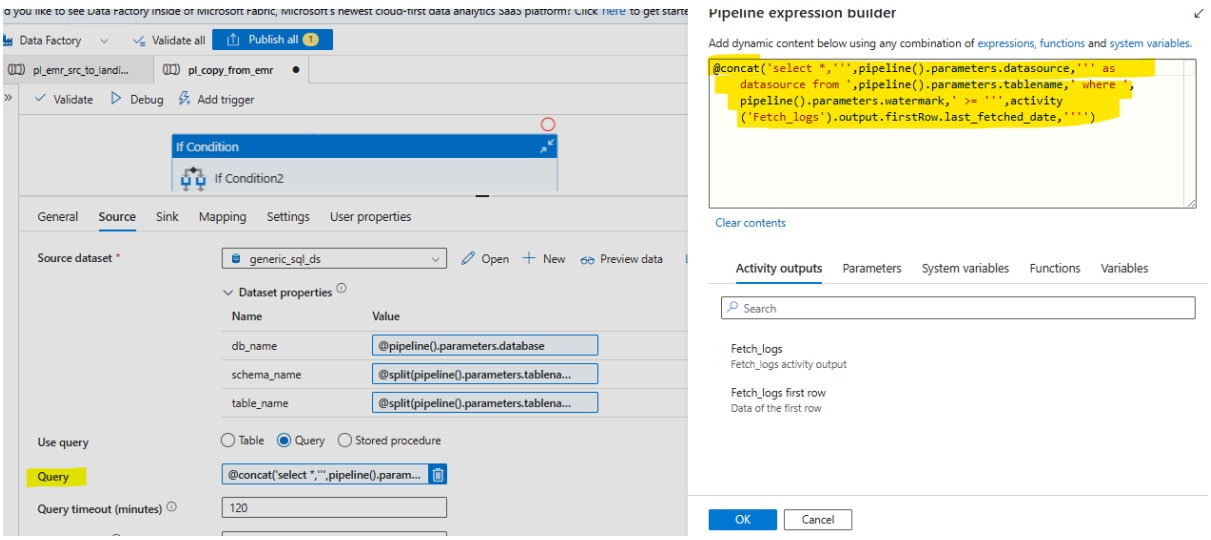
For activity “Fetch logs” we have to update the query instead of item().

```
@concat('select coalesce(cast(max(loaddate) as
date),''','1900-01-01',''') as last_fetched_date from audit.load_logs
where',' data_source=''',pipeline().parameters.datasource,''' and
tablename=''',pipeline().parameters.tablename,'''')
```



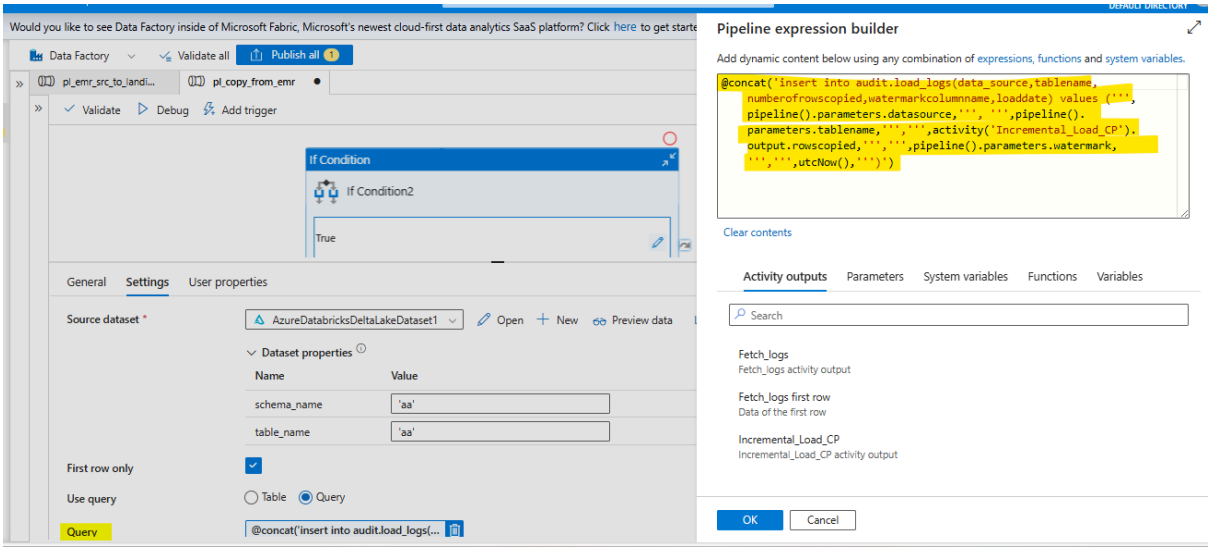
Also we have to update the queries for incremental load as shown below

```
@concat('select *,''',pipeline().parameters.datasource,''' as
datasource from ',pipeline().parameters.tablename,' where
',pipeline().parameters.watermark,' >=
''',activity('Fetch_logs').output.firstRow.last_fetched_date,'''')
```

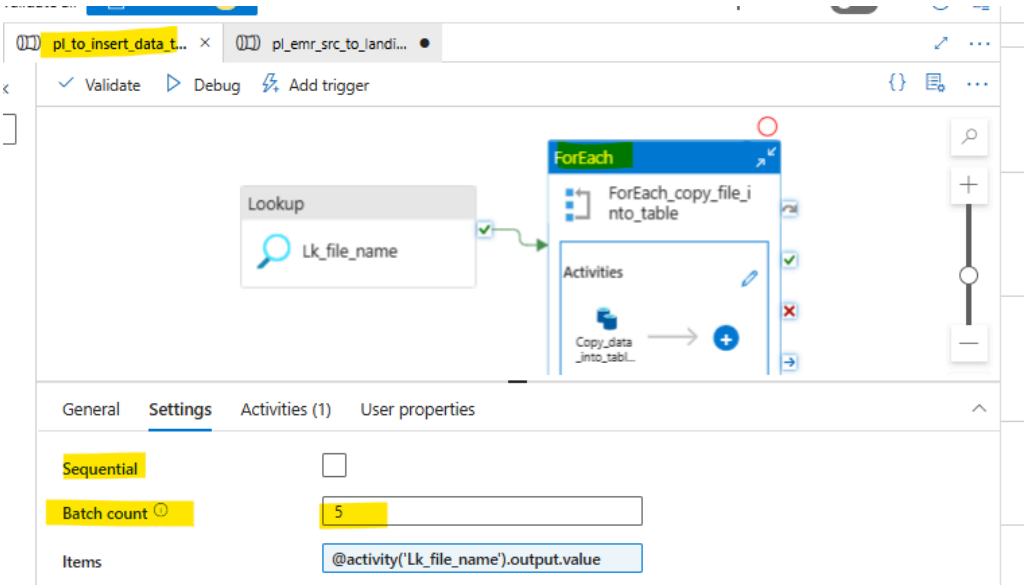


Additionally, we will update the lookup query to insert logs into the audit table for Incremental load as shown below.

```
@concat('insert into
audit.load_logs(data_source,tablename,numberofrowscopied,watermarkcolum
nname,loaddate) values (''',pipeline().parameters.datasource,''',
''',pipeline().parameters.tablename,''',''',activity('Incremental_Load_
CP').output.rowscopied,''',''',pipeline().parameters.watermark,''',''',
utcNow(),''')')
```



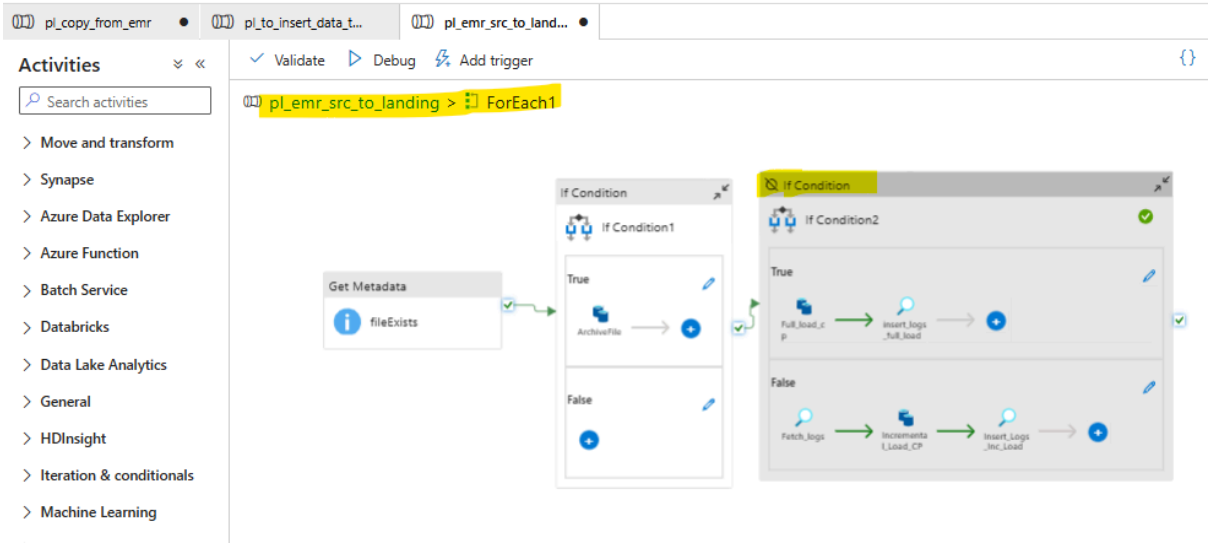
In order to make pipeline -pl_emr_src_to_landing parallel, for “For each” Activity in the pipeline “pl_emr_src_to_landing” we will remove sequential options and will add batch count as 5.



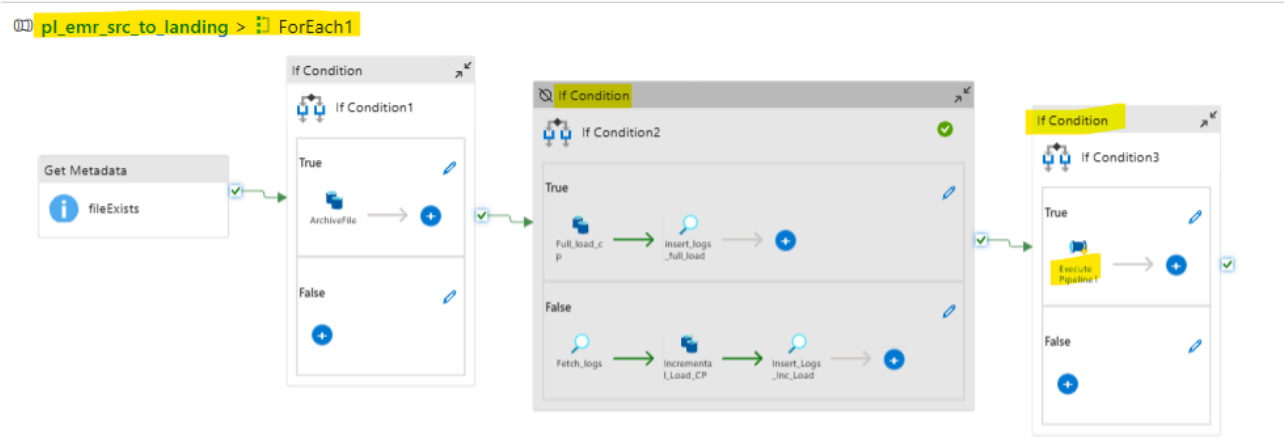
Once this pipeline is updated, publish these changes.

2. Updating the pipeline - pl_emr_src_to_landing

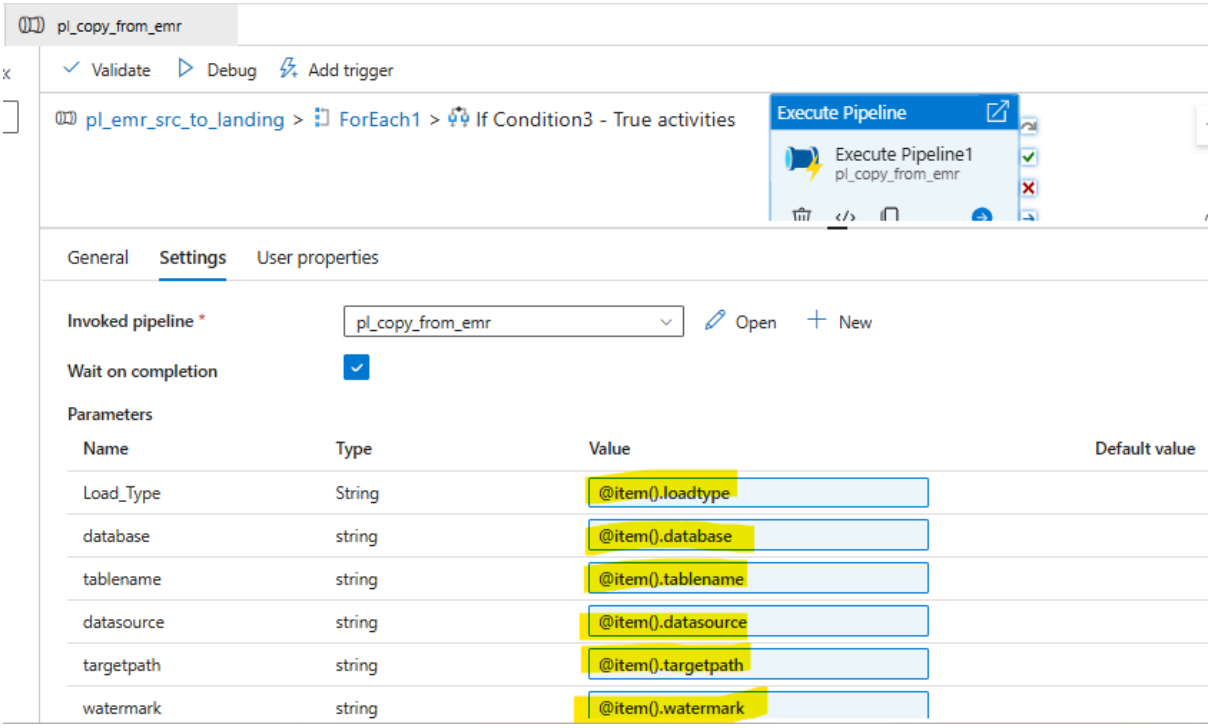
In this pipeline “pl_emr_src_to_landing” deactivate this activity as shown below



Now in this pipeline “pl_emr_src_to_landing” we will add one more if condition with logic @equals(item().is_active,'1') as shown below and in this we will add the execute pipeline in which we will attach the recently created pipeline “pl_copy_from_emr” as shown below



Also for this execute pipeline we will pass the parameter value as shown below.



Steps to move NPI and ICD from API to bronze layer:

For this, we will use the provided code notebook and implement the logic in it.

2. API extracts

=> ICD Code API extract

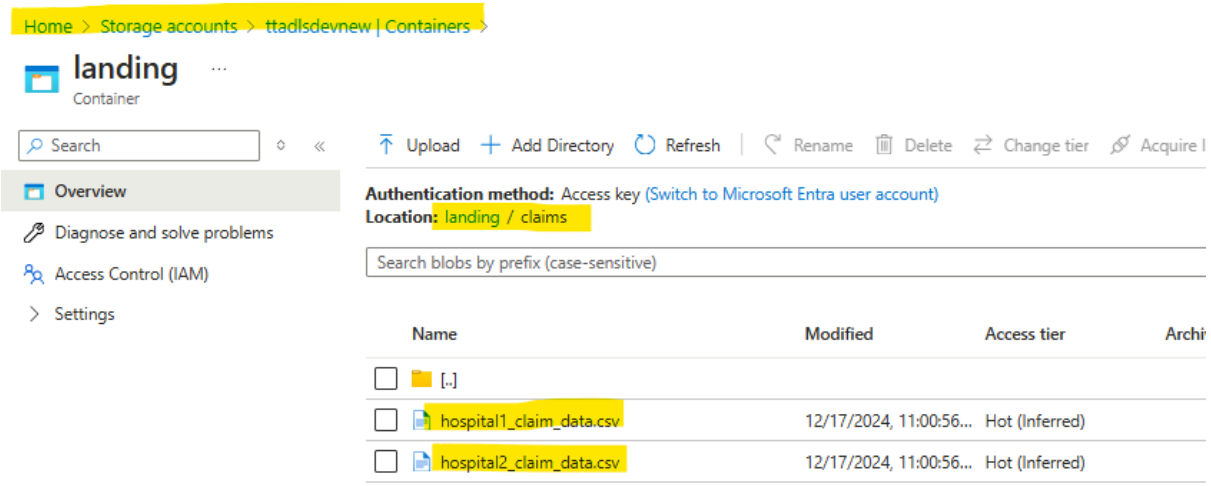
=> NPI API extract

Note: Run this code notebooks in order to fetch data before going ahead.

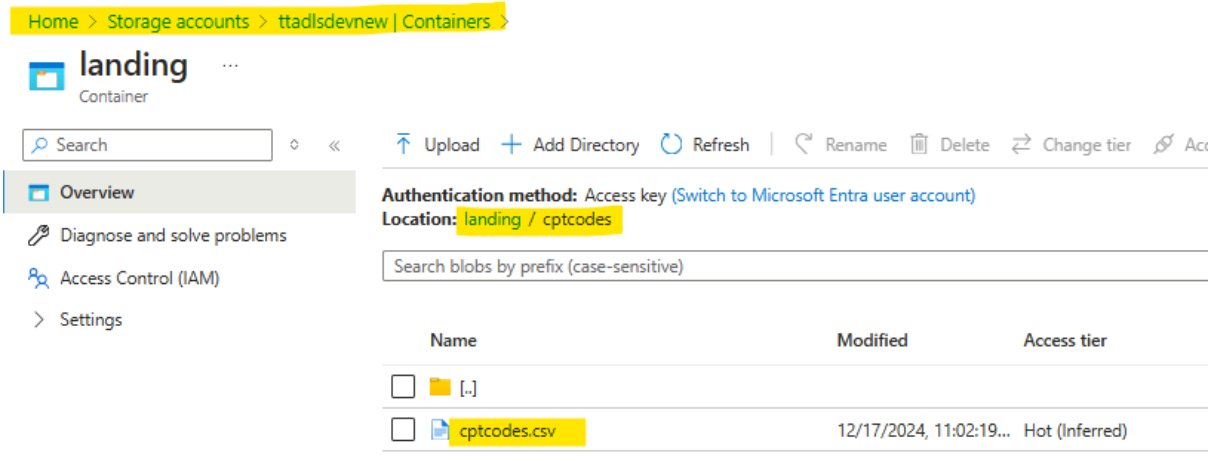
Steps to move Claims and CPT data from landing to bronze layer:

As part of the background activity, we will manually place the claims and CPT data files into their respective folders within the landing folder.

Claims data:



CPT codes data:



Now using the logic present in the code notebooks we will move this data to the bronze layer in parquet format.

3. Silver :

=> Claims

=> CPT codes

Moving data from Bronze layer to Silver.

In this layer, we implement the following logic to transform and refine the data:

Data Cleaning (clean):

=> Removed invalid, null, or duplicate records to ensure data quality.

=> Standardized the data format to align with the Common Data Model for consistency and compatibility across systems.

=> Implement SCD Type 2 logic to maintain historical changes in the data, enabling tracking of changes over time.
Delta Table:

=> Stored the transformed data in Delta tables to support ACID transactions, incremental loads, and versioned data.

This transformation ensures that data from the Bronze layer is cleansed, standardized, and enriched before moving to the Silver layer for further analytics or downstream processing.

We have documented the logic for this transformation in the notebook located in the Silver folder within our Databricks workspace. We have shared the code notebook on Github to which you can refer.

Note: For both claims and CPT codes, we have used the same notebook. This notebook contains the code for moving data from the landing layer to the bronze layer, followed by the necessary transformations to clean and move the data from bronze to the silver layer.

Path: Notebooks → Silver → claims, CPT codes

Moving data from Silver to Gold layer:

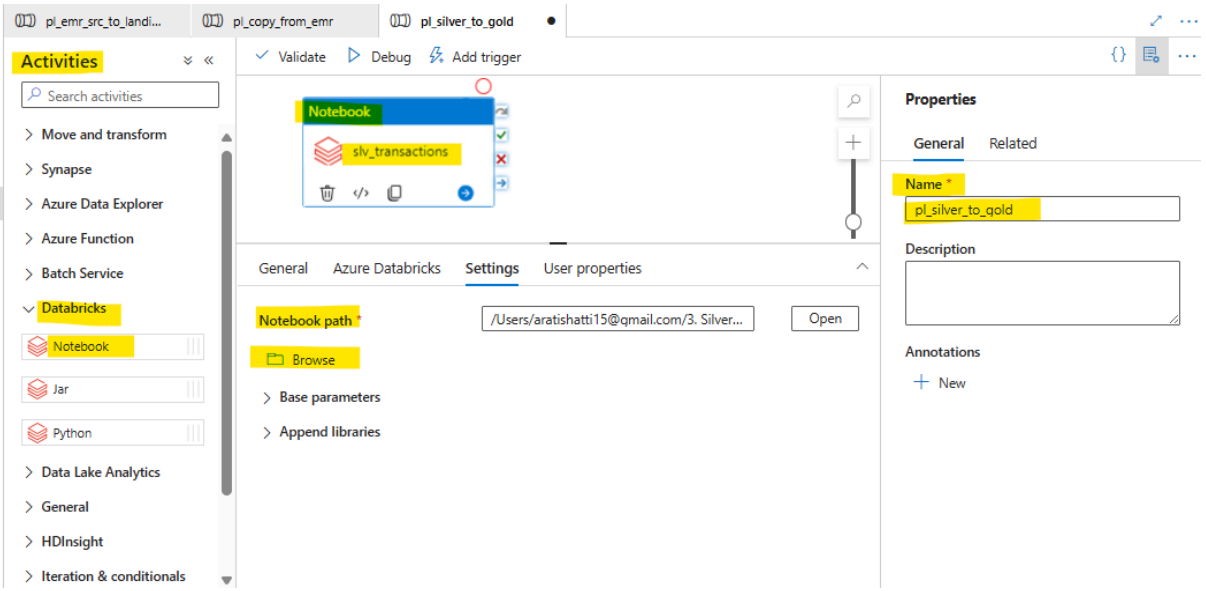
We have created notebooks to implement the required logic, and all these notebooks are located in the "Gold" folder. We have shared the code notebook on Github to which you can refer.

Pipeline to Move data from Silver to Gold layer:

We will create a pipeline “pl_silver_to_gold” to move the data from the Silver layer to the Gold layer.

First create the pipeline “pl_silver_to_gold”, and add the activity Databricks notebook as shown below.

Ex: Here, we have added a notebook activity and named it slv_transaction. Using the Browse Path option, we selected the Transactions notebook located in the Silver folder. Refer below screenshot for more information.



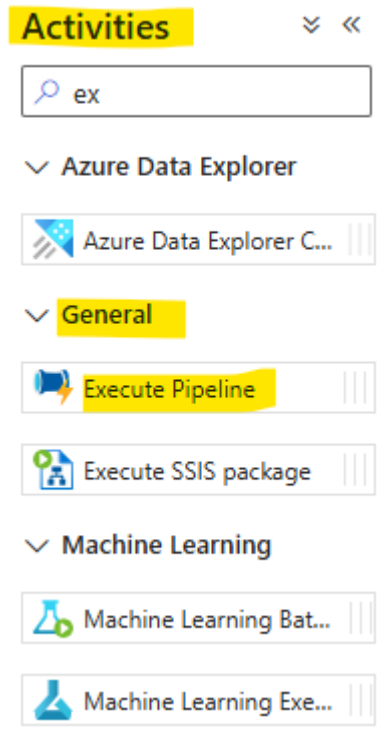
Similarly, we will add the remaining notebooks as demonstrated by Sumit Sir in the video. And our complete pipeline will be as shown below.

Once this pipeline is created, publish these changes.

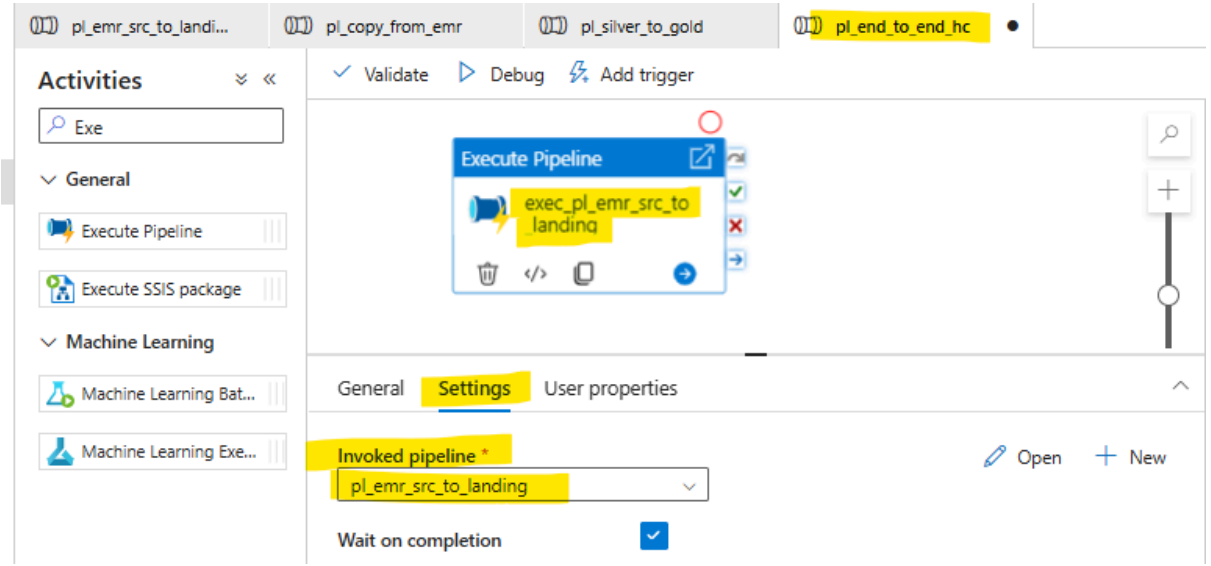
Final Pipeline:

The final pipeline (pl_end_to_end_hc) will include two execution pipelines:

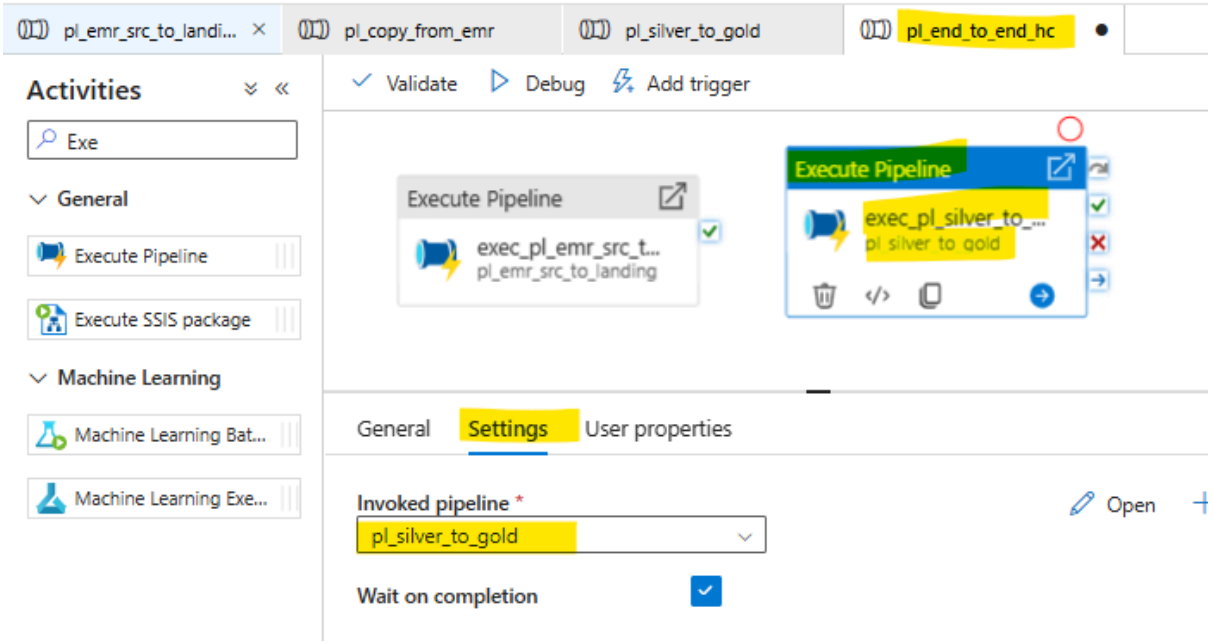
In this we will use an activity Execution pipeline.



=> The first execution pipeline (exec_pl_emr_src_to_landing) will move data from the SQL database to the landing folder.



=> The second execution pipeline (exec_pl_silver_to_gold) will move the cleaned and transformed data from the Silver layer to the Gold layer.

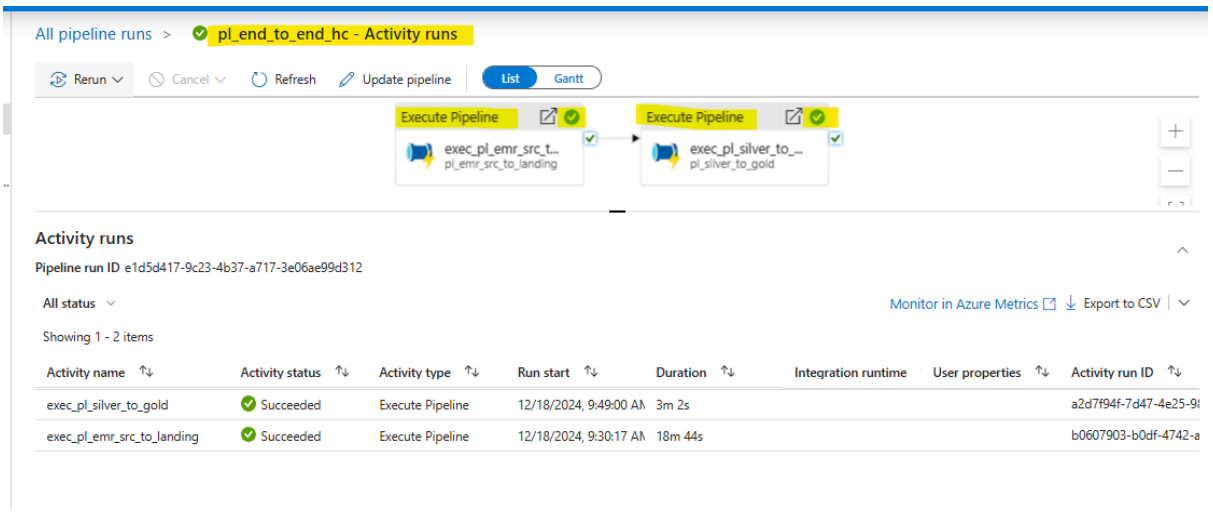


Before running these pipelines make sure to create the silver and gold schema using below commands in catalog `tt_hc_adb_ws`.

```
CREATE SCHEMA IF NOT EXISTS tt_hc_adb_ws.silver;

CREATE SCHEMA IF NOT EXISTS tt_hc_adb_ws.gold;
```

Upon successful execution of the pipeline, the results will be displayed as shown below.



To link a project on Github follow below steps.

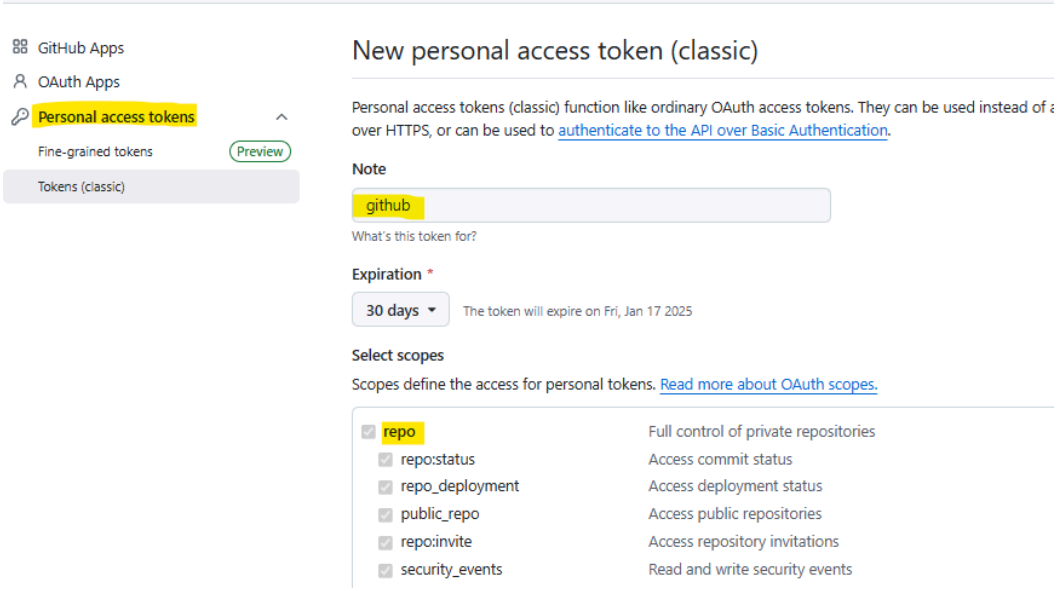
In Azure Datbricks: Get the username and token for your databricks

In Github:

Get the username of your github account.

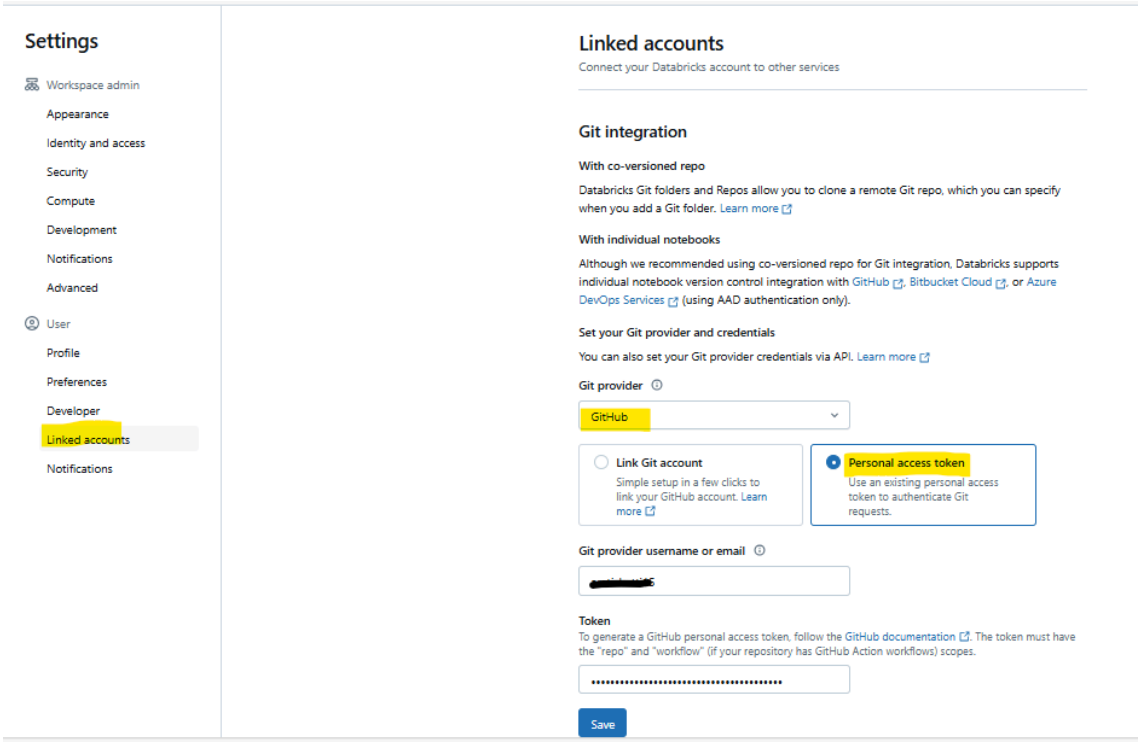
Generate personal access token:
Go to Github => Settings => Developer Settings => Personal Access token => Tokens (Classic) => select all the scopes => generate token => Copy the token

You can provide below permissions:
repo – Full access to repositories.
read:org – Access organizational-level data.
workflow – Trigger GitHub Actions workflows.
read:packages (Optional, for package access).



Create the token and note it once created.

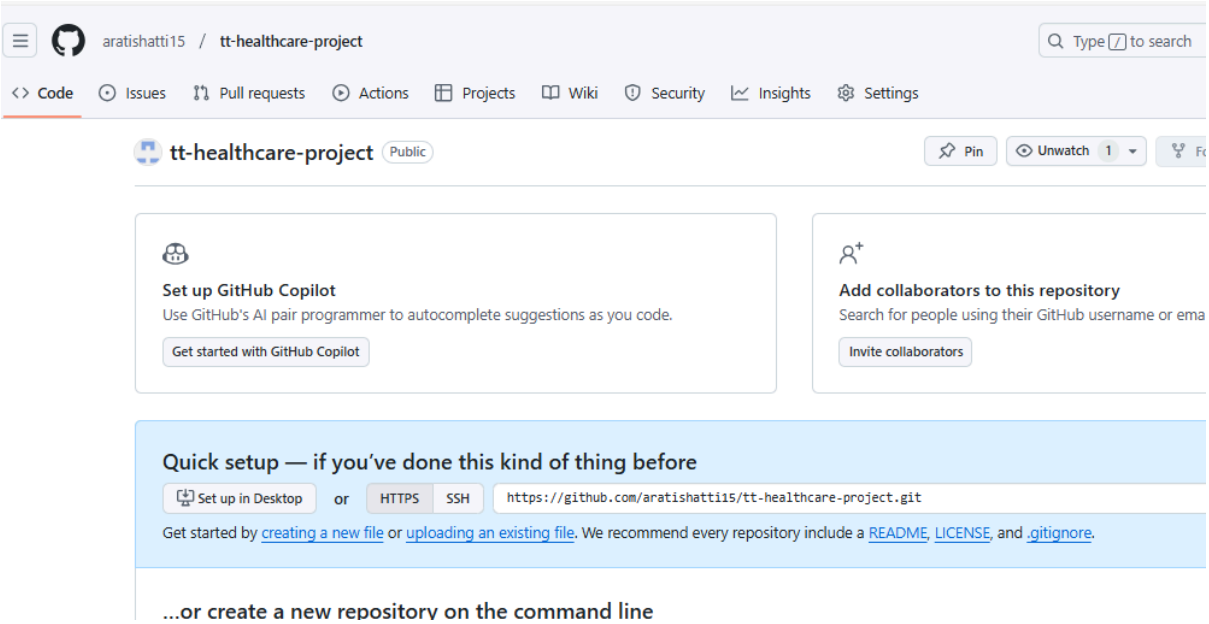
Now go to databricks => setting => Linked accounts = > Git provider (Github) then select personal access token => then provide the username and the token for github



Now create the repository for healthcare project on github account:

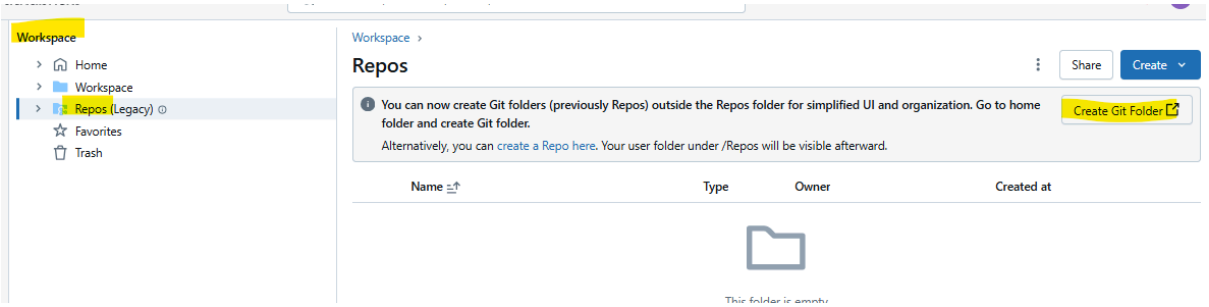
To create a repository on GitHub:

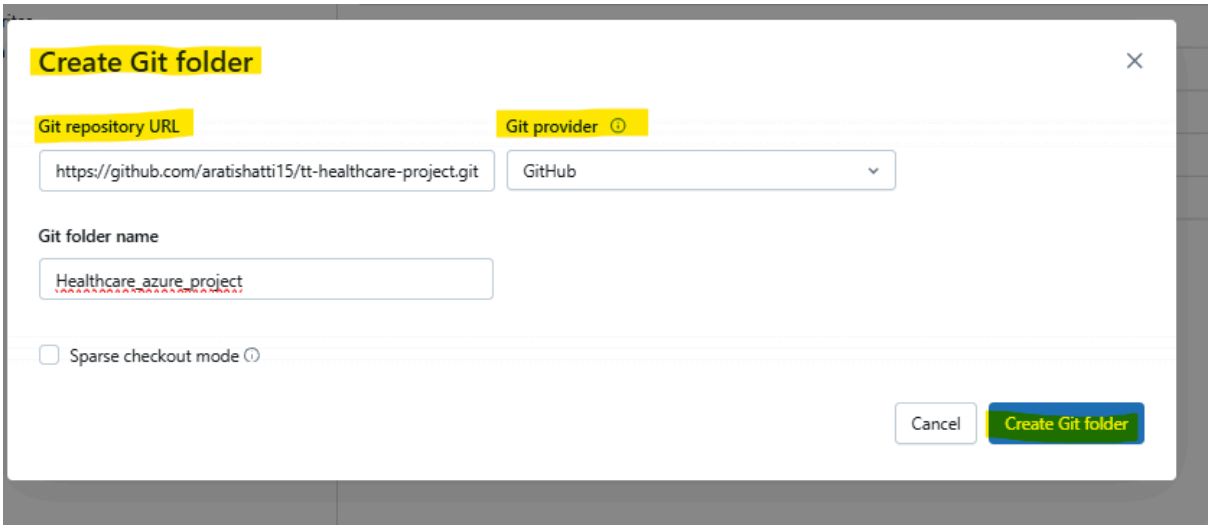
1. Go to GitHub.
2. Click on **New** under "Repositories" in your GitHub profile.
3. Provide:
    - Repository Name: e.g., tt-healthcare-project.
    - Description (optional).
    - Choose **Public** or **Private** visibility.
4. Click **Create repository**.



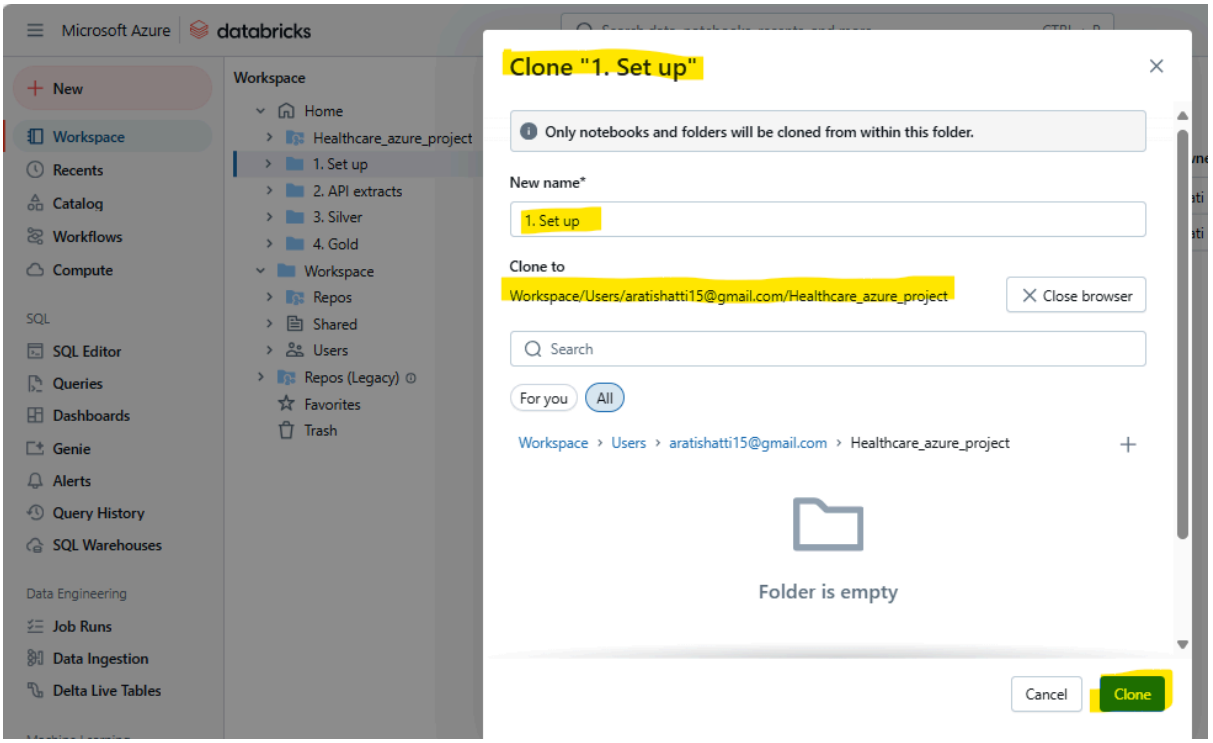
Cloning GitHub Repository into Databricks

Go to databricks => Workspace => Repos => select option create git folder



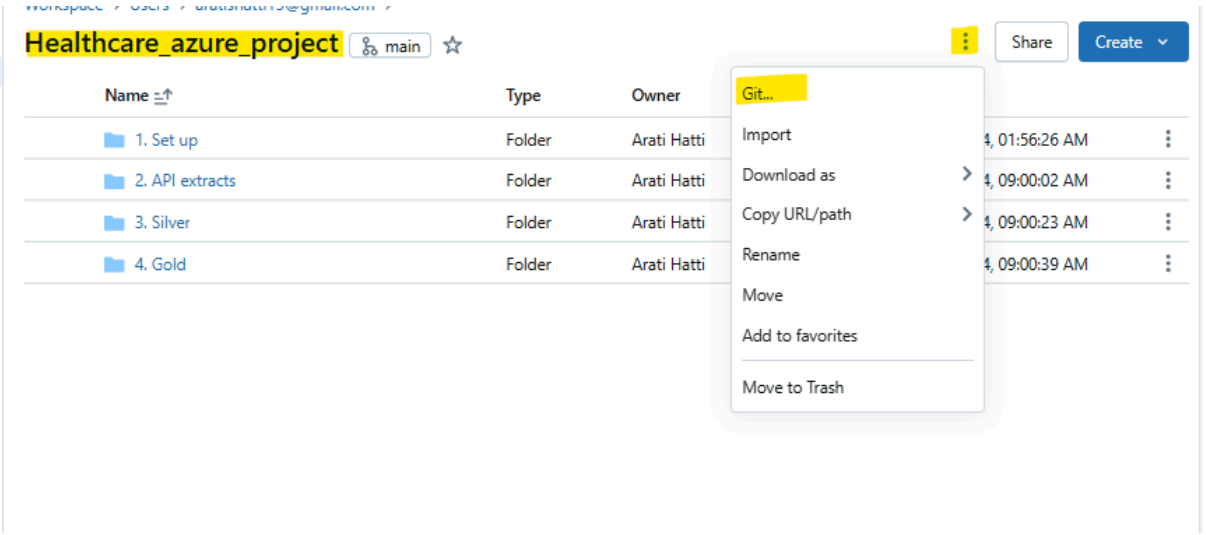


Now you can clone existing folders and files into this git folder.
Right click on file/folder => clone => then using the browse option select this git folder.



Similarly clone all the required files and folders.

To use Git in Databricks, click on the three dots next to the Git-linked folder and select the Git option. This allows you to pull and push changes, as well as create branches, directly from the Databricks interface.


Healthcare_azure_project
main
Share
Create
Name
Type
Owner
1. Set up
Folder
Arati Hatti
2. API extracts
Folder
Arati Hatti
3. Silver
Folder
Arati Hatti
4. Gold
Folder
Arati Hatti
Git...
Import
Download as
Copy URL/path
Rename
Move
Add to favorites
Move to Trash