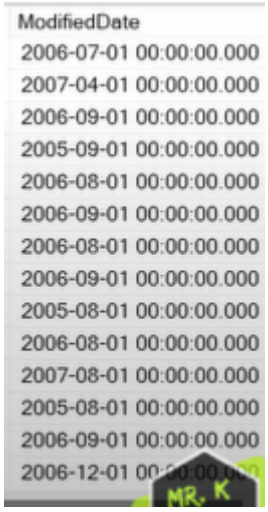
Data Transformation Part2(Bronze to Silver)

1. Now lets see what kind of transformations we can perform on our data



ModifiedDate
2006-07-01 00:00:00.000
2007-04-01 00:00:00.000
2006-09-01 00:00:00.000
2005-09-01 00:00:00.000
2006-08-01 00:00:00.000
2006-09-01 00:00:00.000
2006-08-01 00:00:00.000
2006-09-01 00:00:00.000
2005-08-01 00:00:00.000
2006-08-01 00:00:00.000
2007-08-01 00:00:00.000
2005-08-01 00:00:00.000
2006-09-01 00:00:00.000
2006-12-01 00:00:00.000

2. First we'll modify datetime column to date
3. This code will modify the datetime column to date format

```
from pyspark.sql.functions import from_utc_timestamp, date_format
from pyspark.sql.types import TimestampType

df = df.withColumn("ModifiedDate", date_format(from_utc_timestamp(df["ModifiedDate"].cast(TimestampType()),
"UTC"), "yyyy-MM-dd"))
```

4. In databricks with the help of magic commands we can write different code too
5. Now lets see how can modify columns of all the tables in bronze container at once
6. Here we'll retrieve all the table names..which are in our container

Doing transformation for all tables

Cmd 10

```
1  table_name = []
2
3  for i in dbutils.fs.ls('mnt/bronze/SalesLT/'):
4    table_name.append(i.name.split('/')[0])
```

```
1  table_name
```

```
Out[10]: ['Address',
 'Customer',
 'CustomerAddress',
 'Product',
 'ProductCategory',
 'ProductDescription',
 'ProductModel',
 'ProductModelProductDescription',
 'SalesOrderDetail',
 'SalesOrderHeader']
```

7. Next here..we'll iterate every table and retrieve its path and load the table in the df

```
from pyspark.sql.functions import from_utc_timestamp, date_format
from pyspark.sql.types import TimestampType

for i in table_name:
  path = '/mnt/bronze/SalesLT/' + i + '/' + i + '.parquet'
  df = spark.read.format('parquet').load(path)
  column = df.columns

  for col in column:
    if "Date" in col or "date" in col:
      df = df.withColumn(col, date_format(from_utc_timestamp(df[col].cast(TimestampType()), "UTC"), "yyyy-MM-dd"))

  output_path = '/mnt/silver/SalesLT/' +i +'/'
  df.write.format('delta').mode("overwrite").save(output_path)
```

After that we find if any columns has date type in every column..if there's a date col..then we change its format

8. Once that is done..we'll write this output in delta format ..to the silver container
9. Both Parquet and Delta are popular formats for storing big data, but they serve slightly different purposes. Here's a breakdown:

   Parquet

   - **File format:** Columnar. Stores data in columns instead of rows, allowing for efficient storage and faster queries that only need specific columns (column pruning).
   - **Focus:** Efficient storage and fast querying.
   - **Example:** Imagine a table with user data like name, age, and city. Parquet would store each column (name, age, city) separately, allowing you to query just the "age" column for faster results.

   Delta

   - **Table format:** Built on top of Parquet, adding a metadata layer. Stores data in Parquet files but adds features like versioning, ACID transactions (Atomicity, Consistency, Isolation, Durability), and time travel capabilities.
   - **Focus:** Transactional data management, data integrity, and historical analysis.
   - **Example:** Delta allows you to update specific user records (e.g., change a city) while keeping a history of previous versions. You can then "travel back in time" to analyze data at a specific point.

   - Use Parquet for data warehouses where fast querying and efficient storage are crucial, and data updates are less frequent.
   - Use Delta for data lakes where data is constantly changing, data integrity is important, and historical analysis is needed.
10. Now if we run our code..we can see the data in the silver container
11. So we have completed our level1 tranformation

Data Transformation Part2(Silver to Gold)

1. Lets see what kind of transformation we'll handle here
2. We'll transform the columns name here

   we'll add '_' in the bw the names
3. Our input path is

4. We load the data and display

```
df = spark.read.format('delta').load(input_path)
```

```
Cmd 5
```

```
display(df)
```

5. This code will convert the column names and add '_' if column has 2 or more words

```
Cmd 6

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, regexp_replace


# Get the list of column names
column_names = df.columns

for old_col_name in column_names:
    # Convert column name from ColumnName to Column_Name format
    new_col_name = "".join(["_" + char if char.isupper() and not old_col_name[i - 1].isupper() else char for i, char in enumerate(old_col_name)]).lstrip("_")

    # Change the column name using withColumnRenamed and regexp_replace
    df = df.withColumnRenamed(old_col_name, new_col_name)

Cancel  Uploading command
```

6. Now to transform all the tables…we'll do the same as we did in the bronze

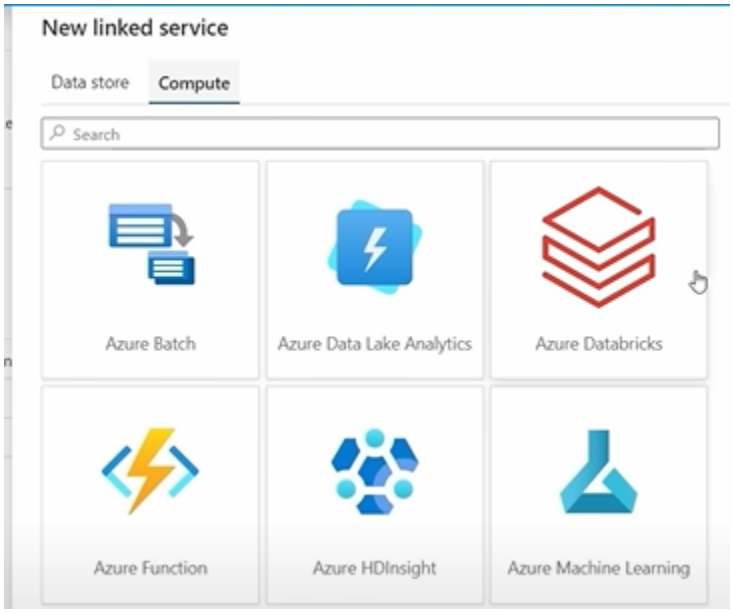
```
for name in table_name:
  path = '/mnt/silver/SalesLT/' + name
  print(path)
  df = spark.read.format('delta').load(path)

  # Get the list of column names
  column_names = df.columns

  for old_col_name in column_names:
      # Convert column name from ColumnName to Column_Name format
      new_col_name = "".join(["_" + char if char.isupper() and not old_col_name[i - 1].isupper() else char for i, char in enumerate(old_col_name)]).lstrip("_")

      # Change the column name using withColumnRenamed and regexp_replace
      df = df.withColumnRenamed(old_col_name, new_col_name)

  output_path = '/mnt/gold/SalesLT/' +name +'/'
  df.write.format('delta').mode("overwrite").save(output_path)
```

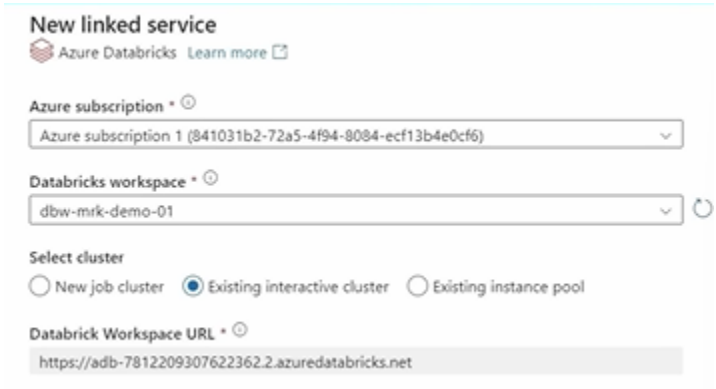
At last ..it dumps the transformed data in the gold container

Data Transformations Part 3

1. Now we'll use ADF to create a pipeline toi just run

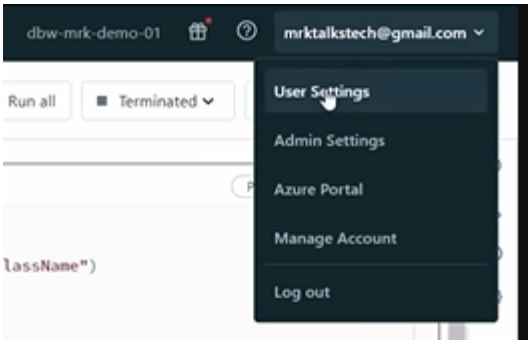
2. First we need to connect databricks with the ADF..for that we use linkedServive

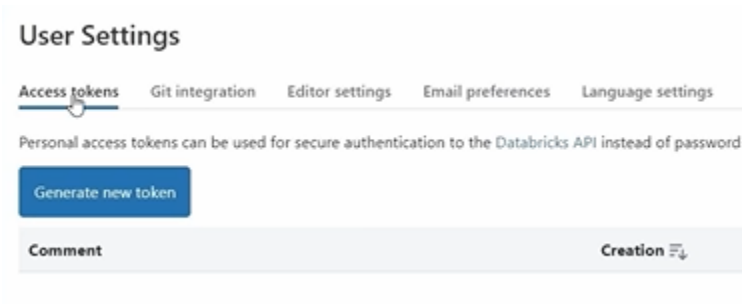


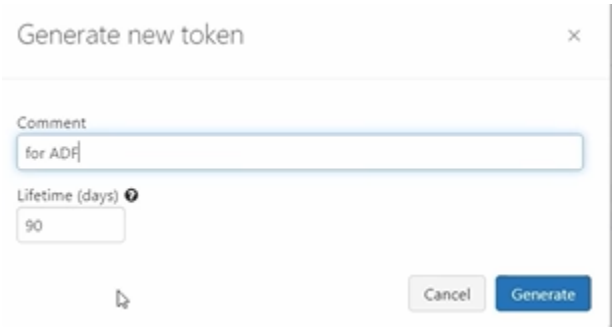


3. In the authentication type of linked service..we use access token

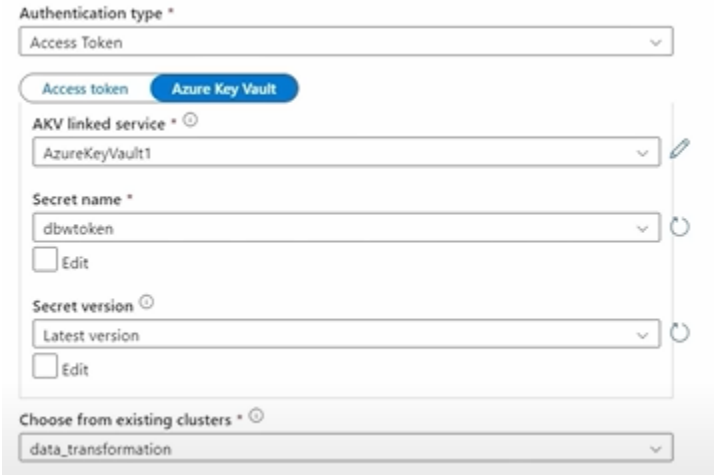
4. For that we go to databricks workspace





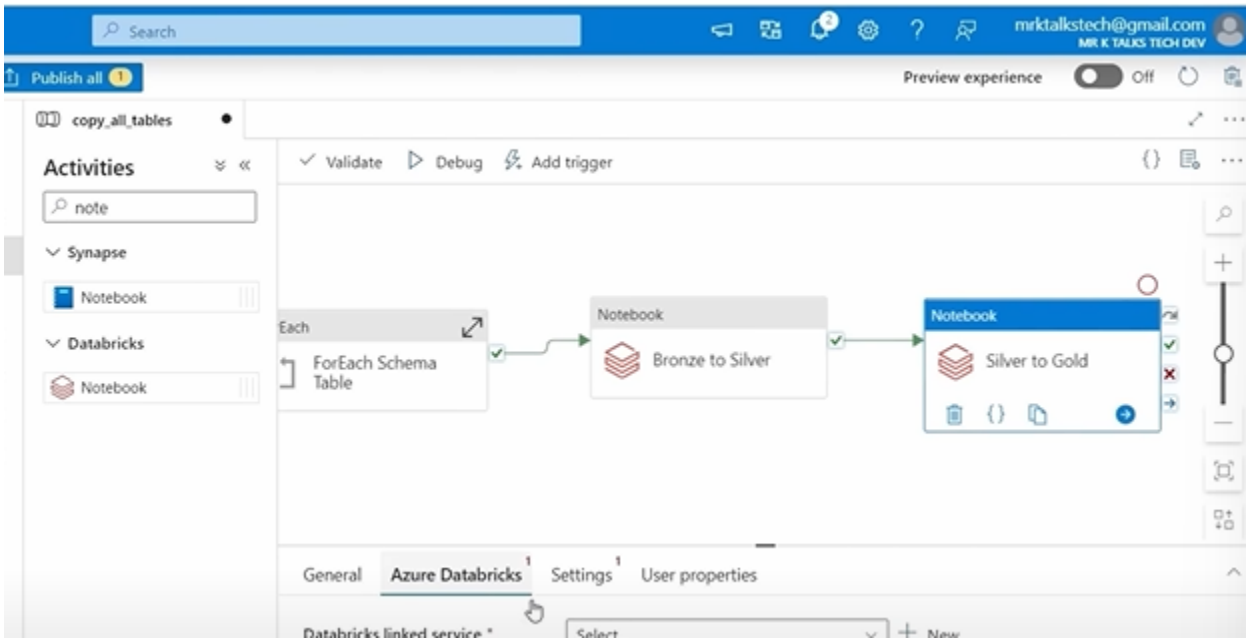


5. And we store these access token in the Azure KV..
6. We use these Azure KV and get our secret token in linked service

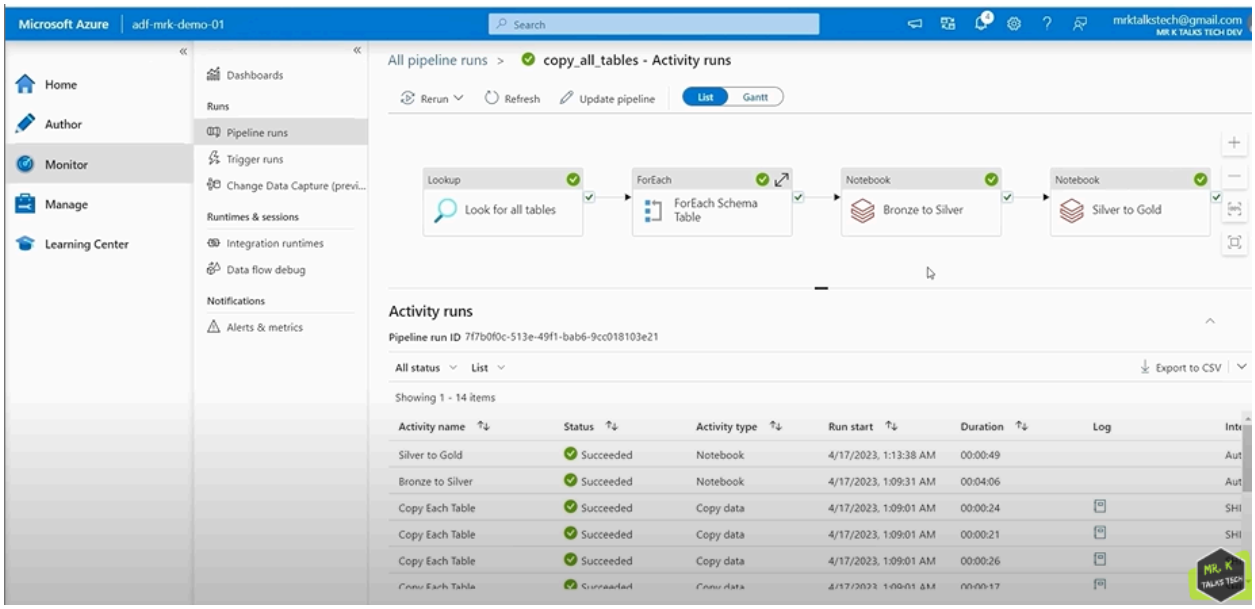


7. After that we click on create connection and publish this
8. Next we go to the author tab and proceed with our pipeline
9. Now we'll create two activities one for bronze and other for silver

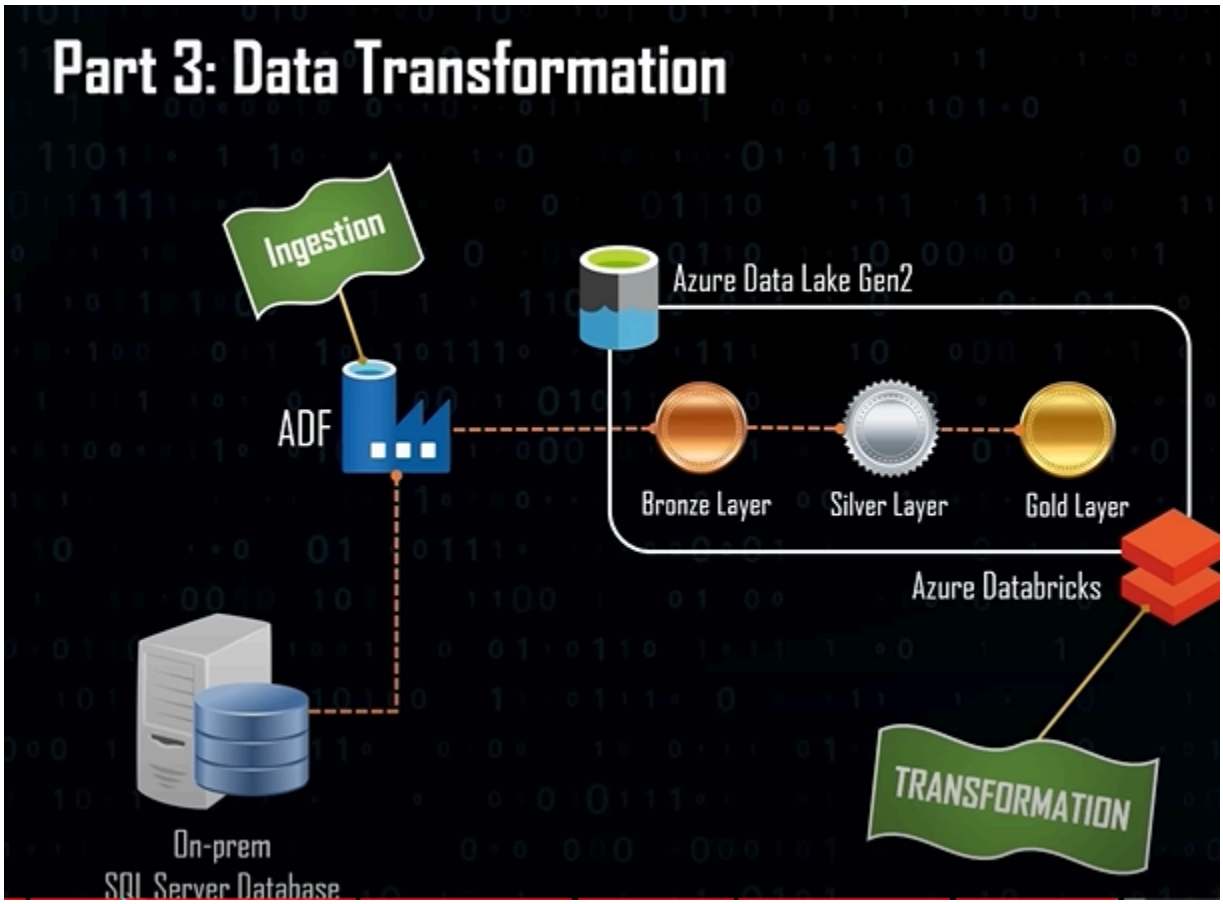
10. Here we have created two notebook activities and named them as per our notebook..and we link for each activity and bronze to silver and silver to gold



11. So what we did here…getting all the tables from SQL server db..and storing it in the bronze container and performing some transformations
12. Now we'll publish and run our entire pipeline



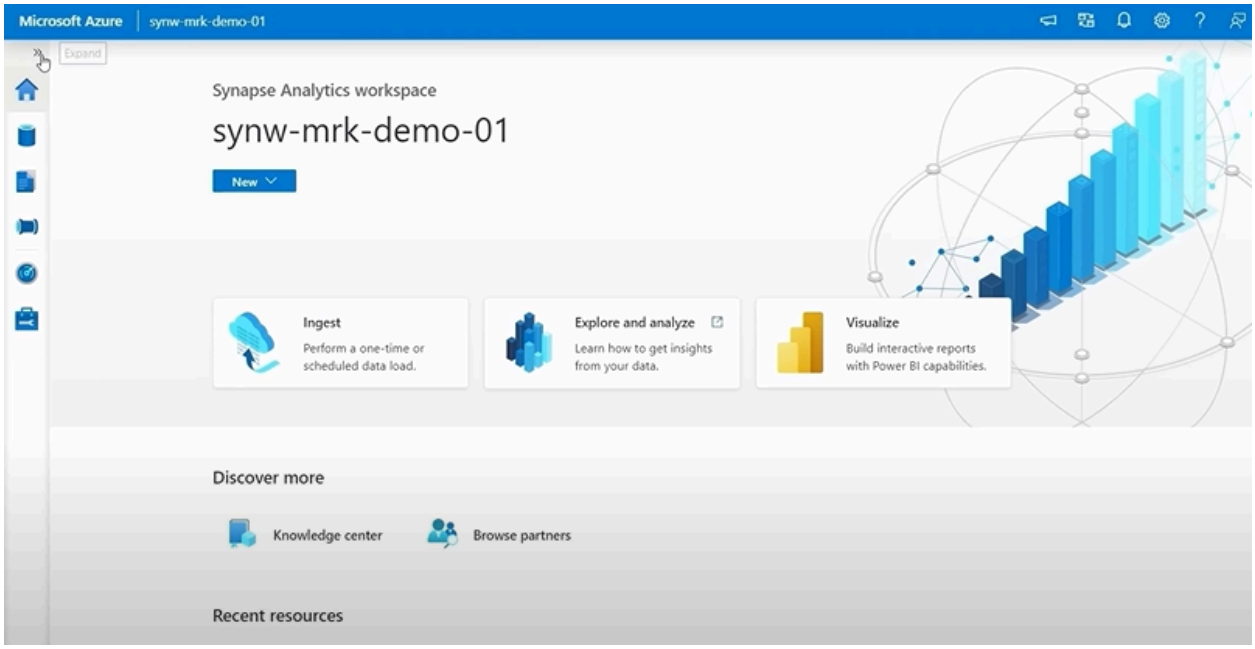
13. Now we have completed our transformation process..and the cleanest data is in gold container



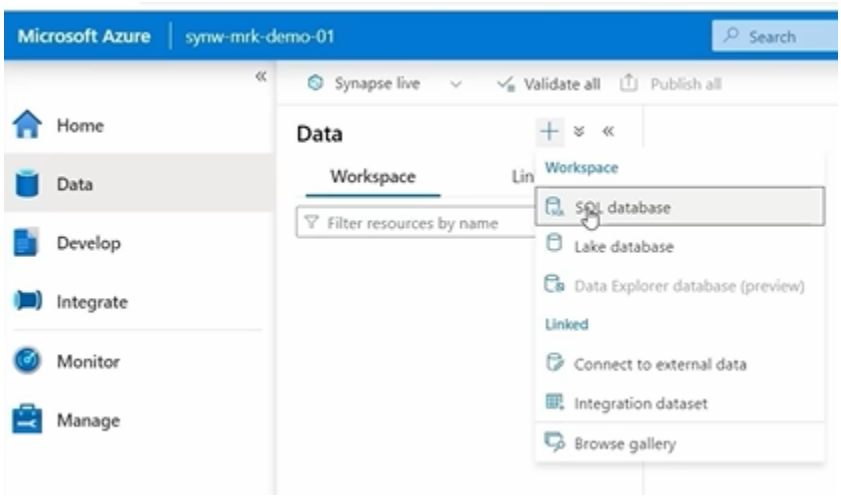
Data Loading using Azure Synapse Analytics

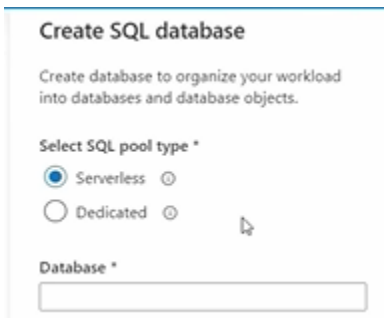
1. So we already have synapse workspace in our resource grp

2. This is how synapse workspace looks like



3. Synapse was built on top of ADF..so it can do whatever ADF can do
4. Here we have two additional features Data and develop
5. We can think Synapse is hybrid of ADF and databricks
6. Now lets create a Database in azure synapse analytics





There will be 2 types of DB's

7. Serverless is used for small workloads and dedicated is used for large workloads

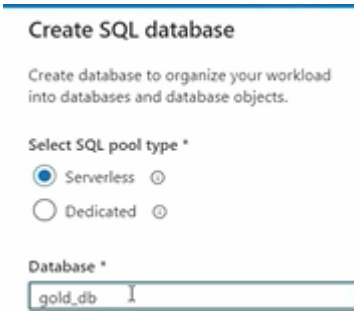
Serverless SQL Database:

- **Automatic Scaling:** Serverless databases handle scaling compute resources up or down based on your workload. You don't need to manage this yourself. This is ideal for workloads that fluctuate.
- **Pay-per-Use:** You only pay for the compute resources you use per second. There's no fixed cost for reserved resources when the database is inactive. This is cost-effective for unpredictable workloads.
- **Auto-Pausing/Resuming (General Purpose only):** The database can automatically pause during inactive periods, minimizing costs. It resumes when activity returns.

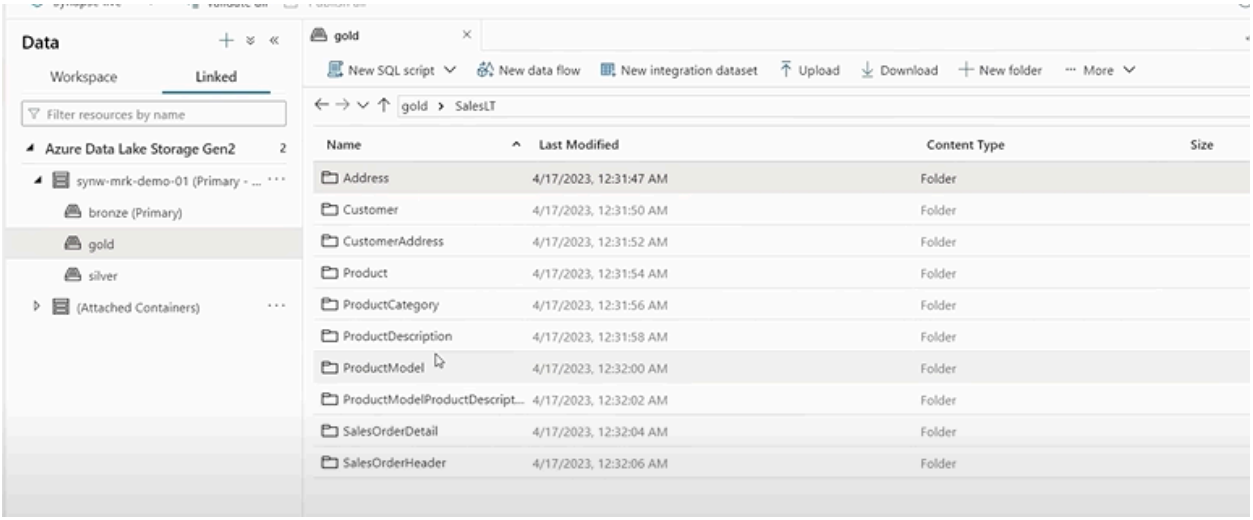
Dedicated SQL Database:

- **Predictable Performance:** Dedicated databases offer predictable performance because you allocate a fixed amount of compute resources in advance. This is suitable for workloads with consistent demands.
- **Full Control:** You have more control over server configuration and performance settings compared to serverless options.
- **Continuous Operation:** Dedicated databases run continuously, unlike serverless which might pause during inactivity.

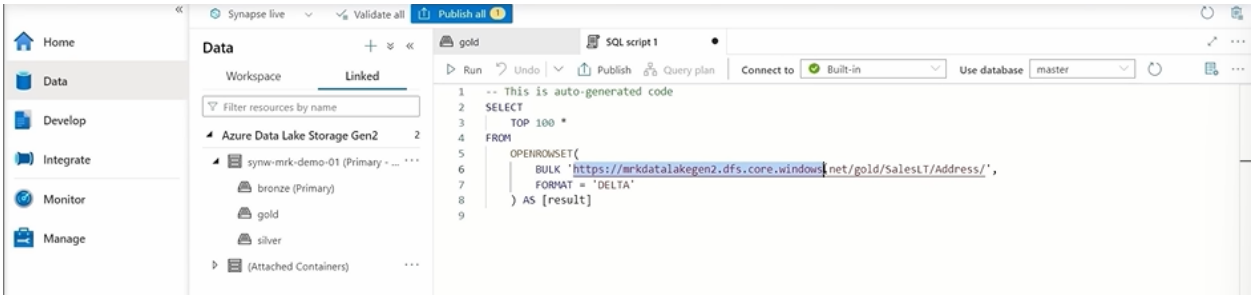
8. Dedicated DB is expensive than serverless
9. For our project we'll go ahead with serverless, as our data is small and properly structured
10. We give name as



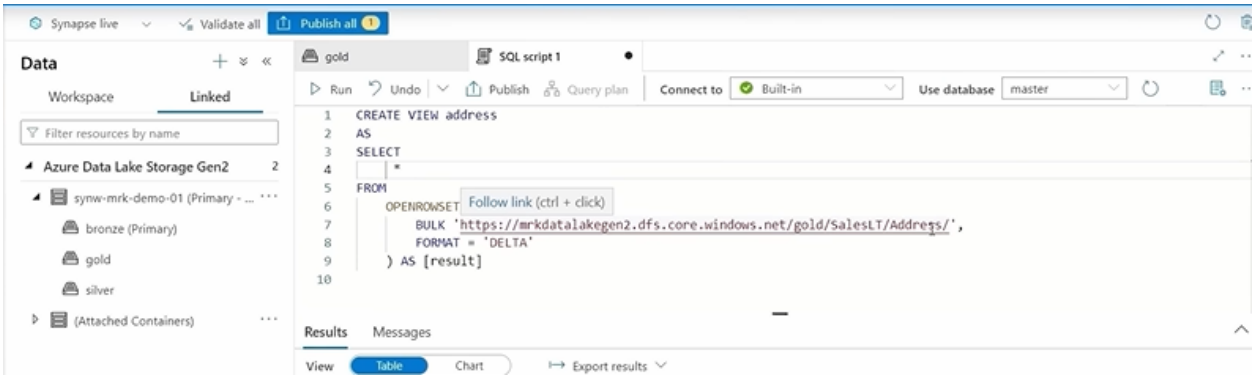
11. Now inside the linked service tab…we can see our data in the datalake gen2



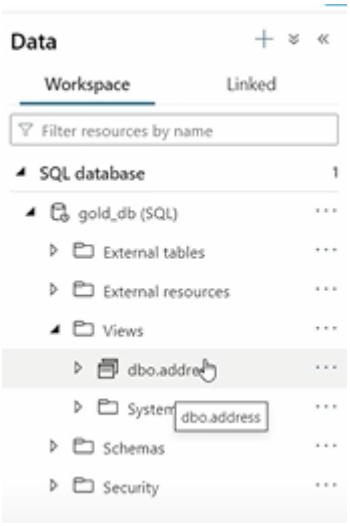
12. Now we can also write SQL scripts for which the data is in ADF…
13. We have to give location and file format and Azure Synapse auto generates the code



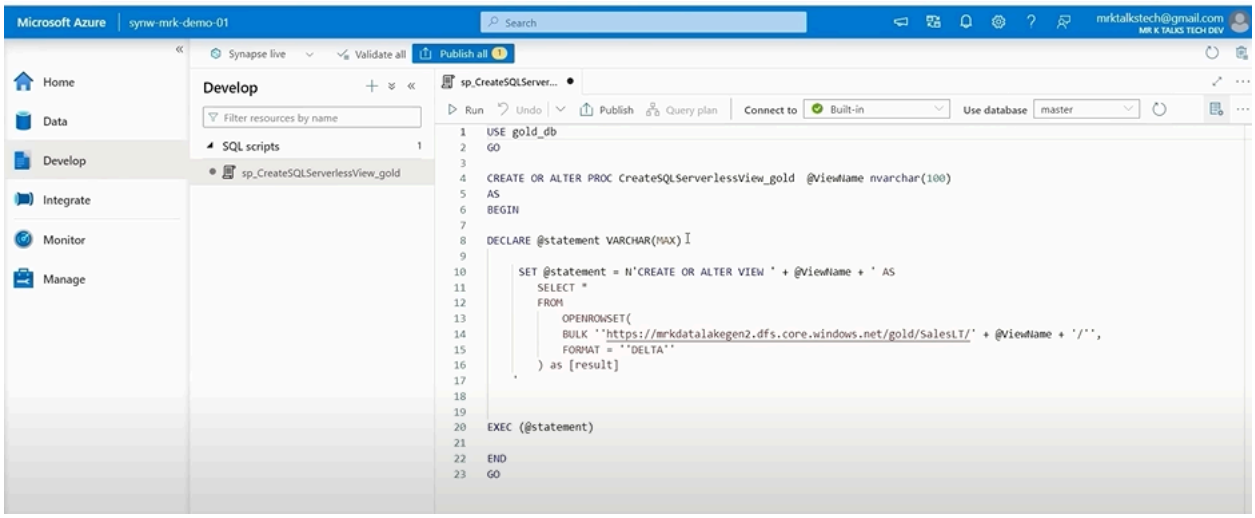
14. We can also use these scripts ..to create views in the serverless DB
15. The SQL Script which creates view on address table



16. After creating the view..we can access the views from the workspace tab in our Server Less Gold_DB

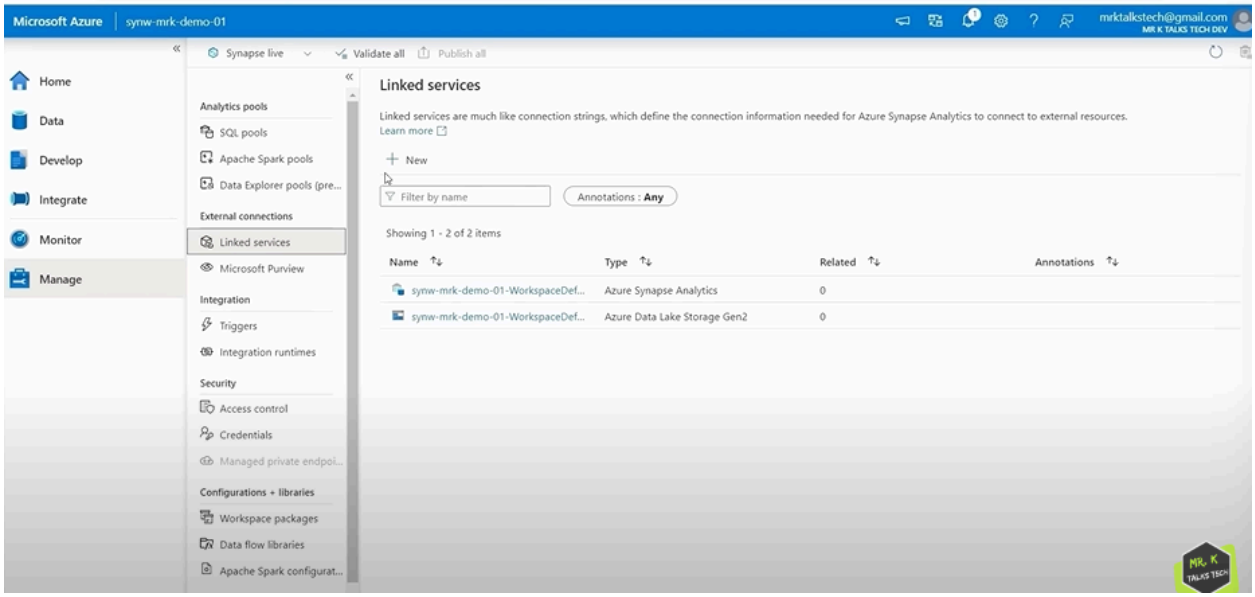


17. Now we'll create a pipeline which dynamically creates views for all the tables
18. For this pipeline ..we create stored procedure..with parameters ..that can create views for all the tables in our gold container..
19. So we will go to develop tab and write our script

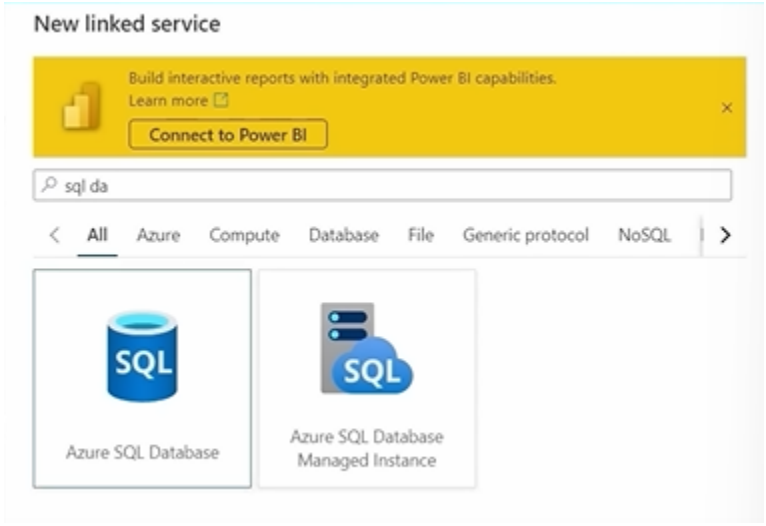


20. Next step would be creating the pipeline..that connect to our stored procedure

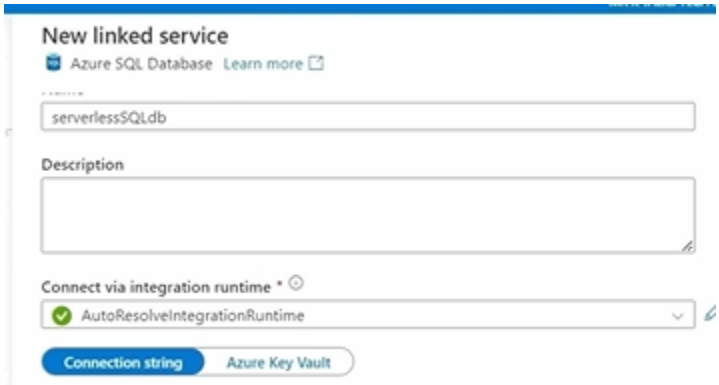
21. For that we need to use linked Services

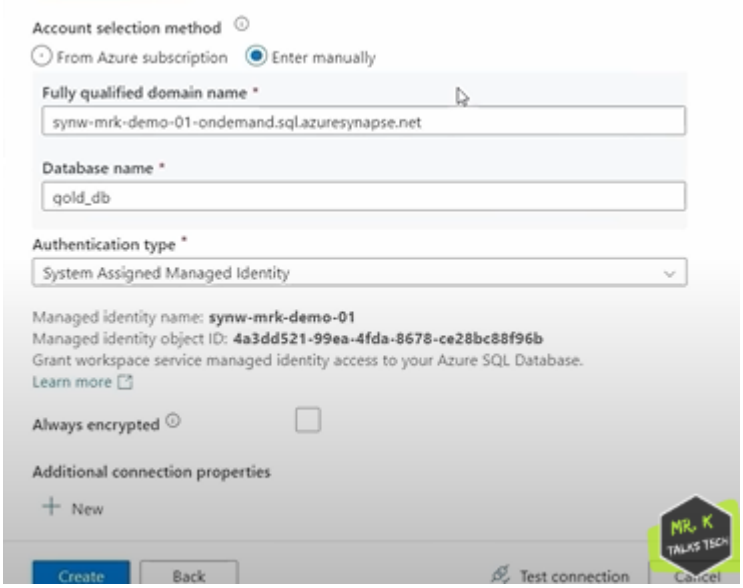
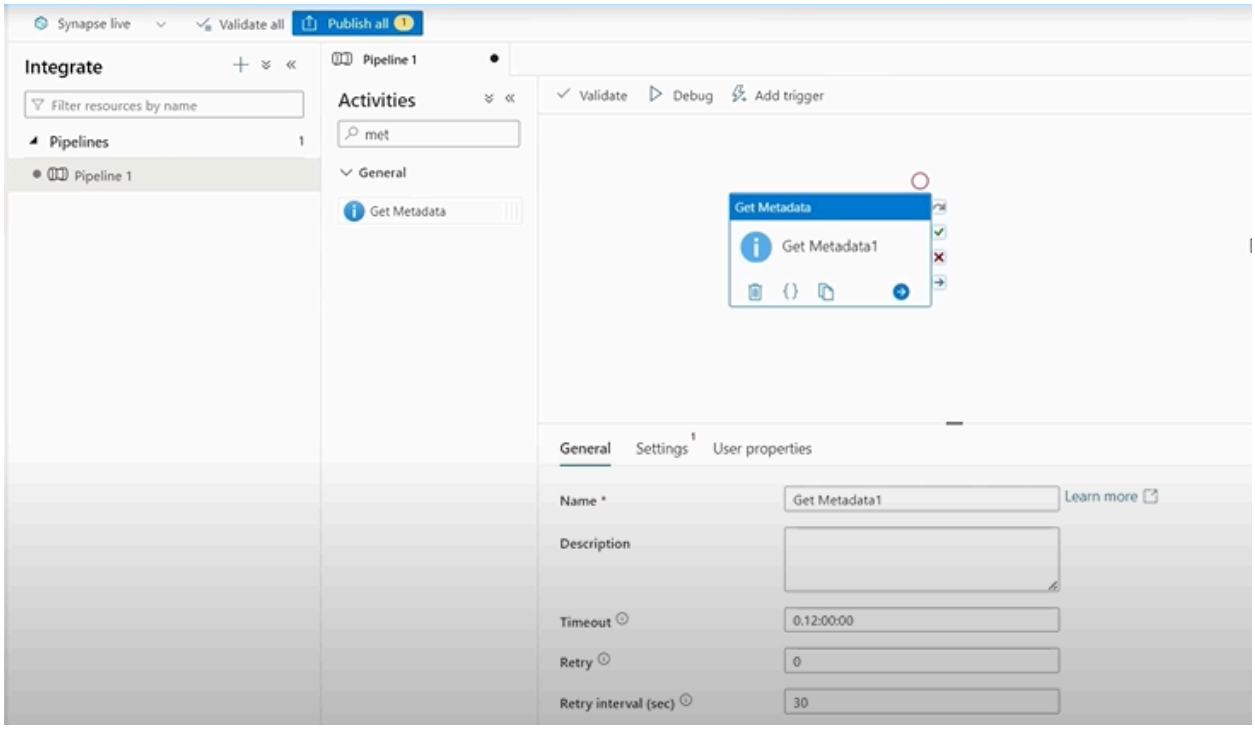


22. Click on new and select Azure SQL DB



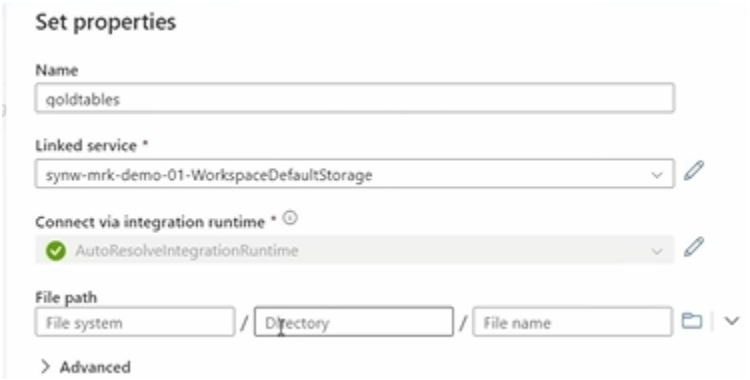
23. We'll give a name to this linked service



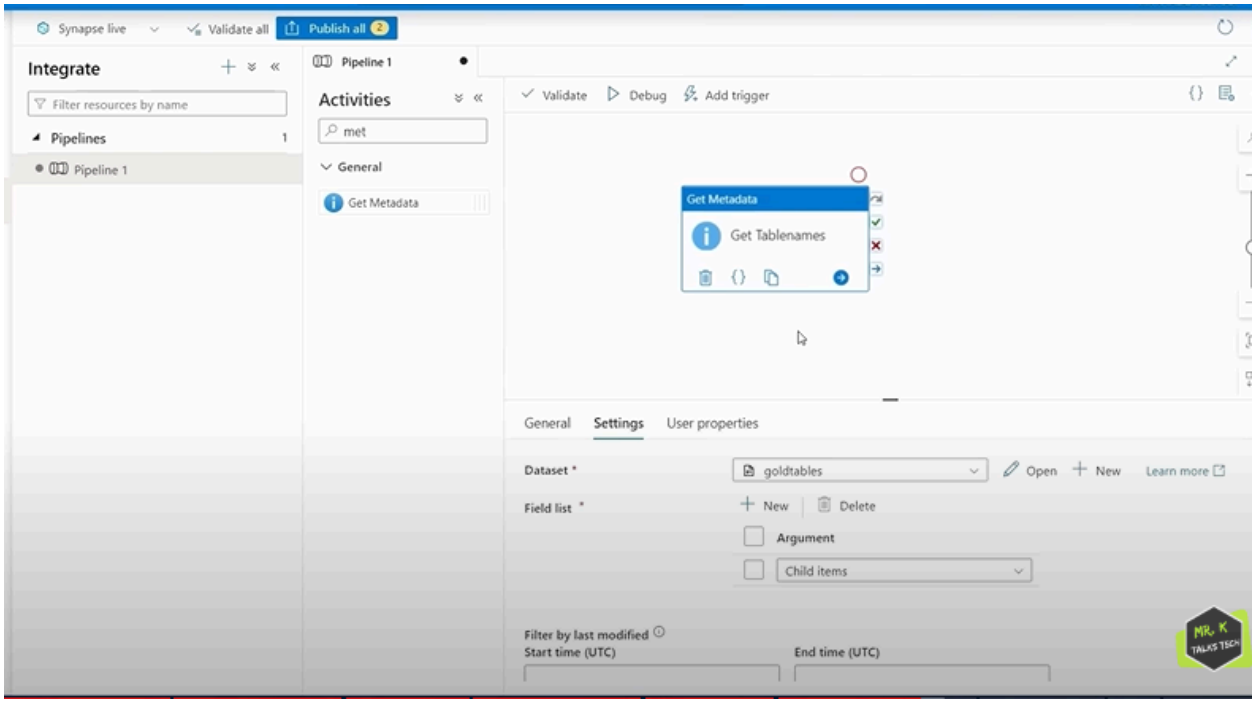
24. And we give all these details…domain name can be retrieved from synapse properties and we gave our DBname and authentication type as System Assigned Managed Identity 
 so system assigned managed ..it will use the signed in user access to get the authentication
25. Then we create this linked service and publish all the changes
26. Inside the Integration tab..we'll create pipeline and add metadata activity 

27. Later in the settings tab…we need to create a new dataset for our data lake and select the binary file format
28. In the properties..we give a new name to this and in linked services as synapse has direct connection to the Data lake we choose this

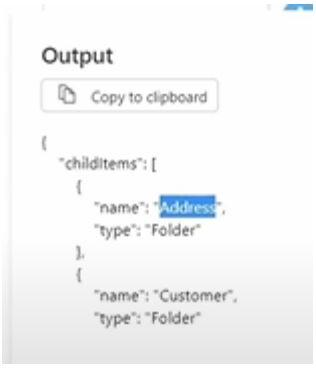
and in the file path ..we choose gold container with SalesTP schema to get the metadata of our tables



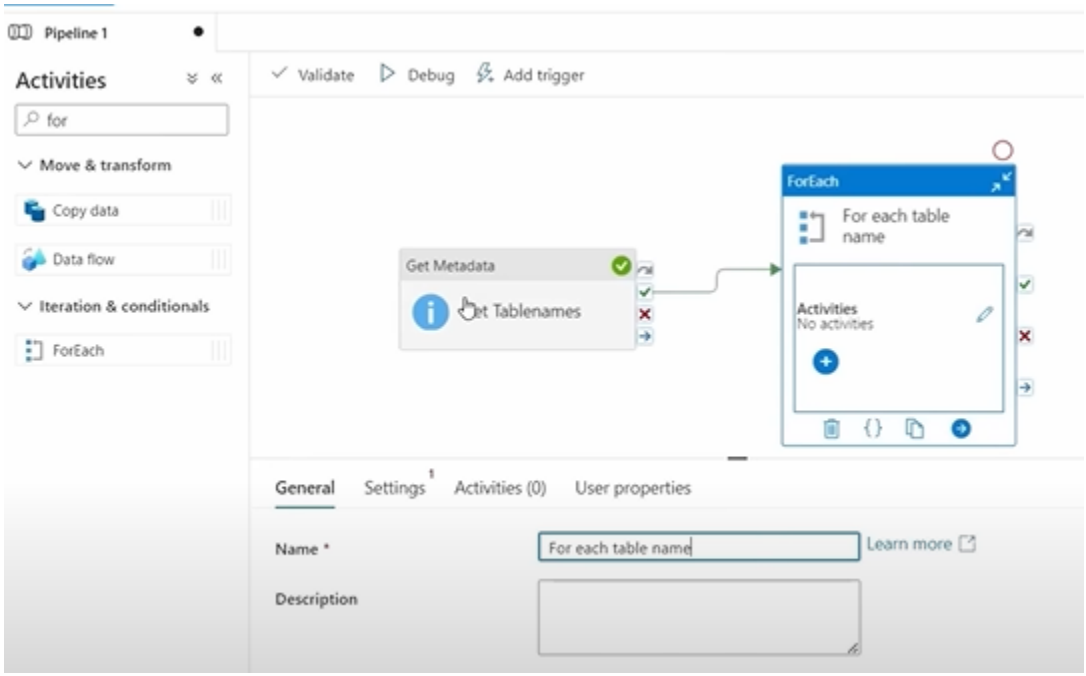
29. Now we have specified the dataset…and now we need to give child items in the field lists



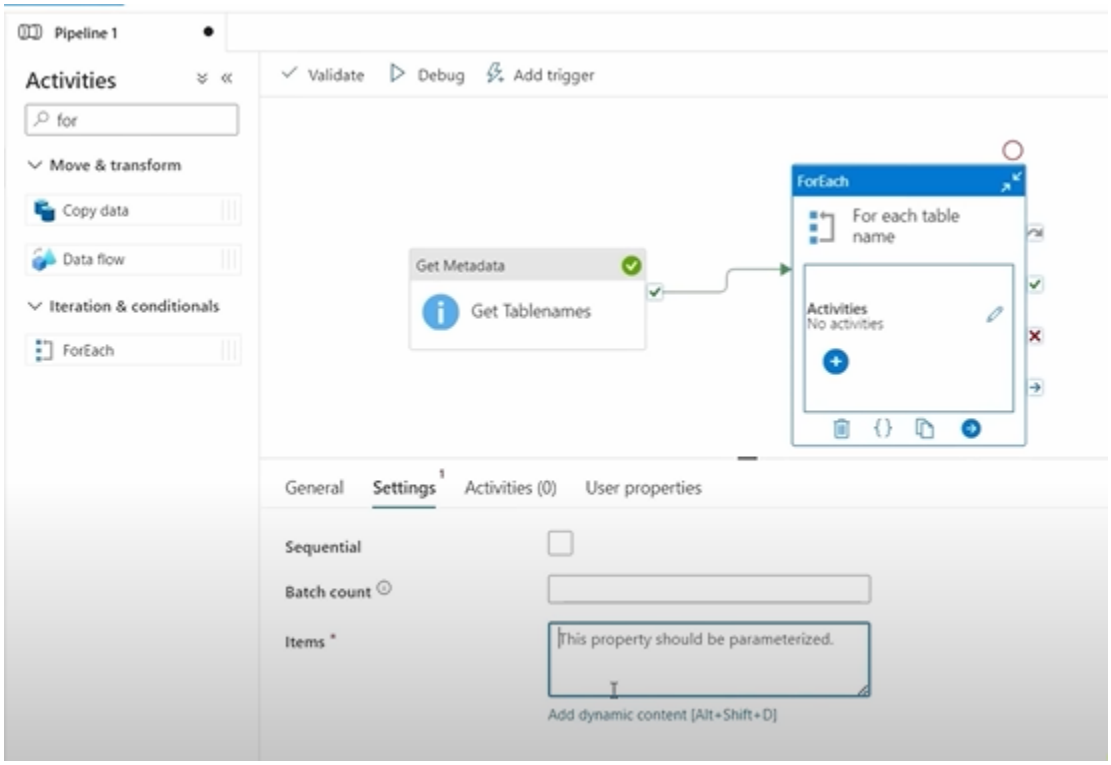
It gives all the child items in SalesTP schema

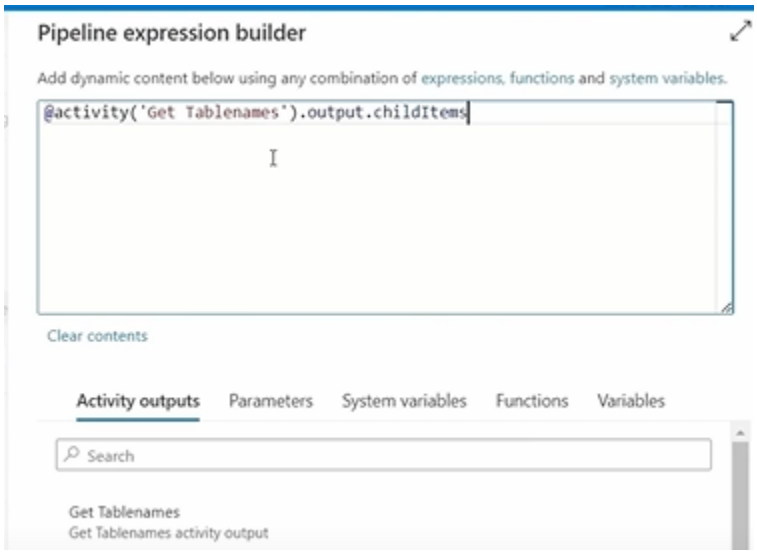


30. And next we use for each activity to iterate over this child items

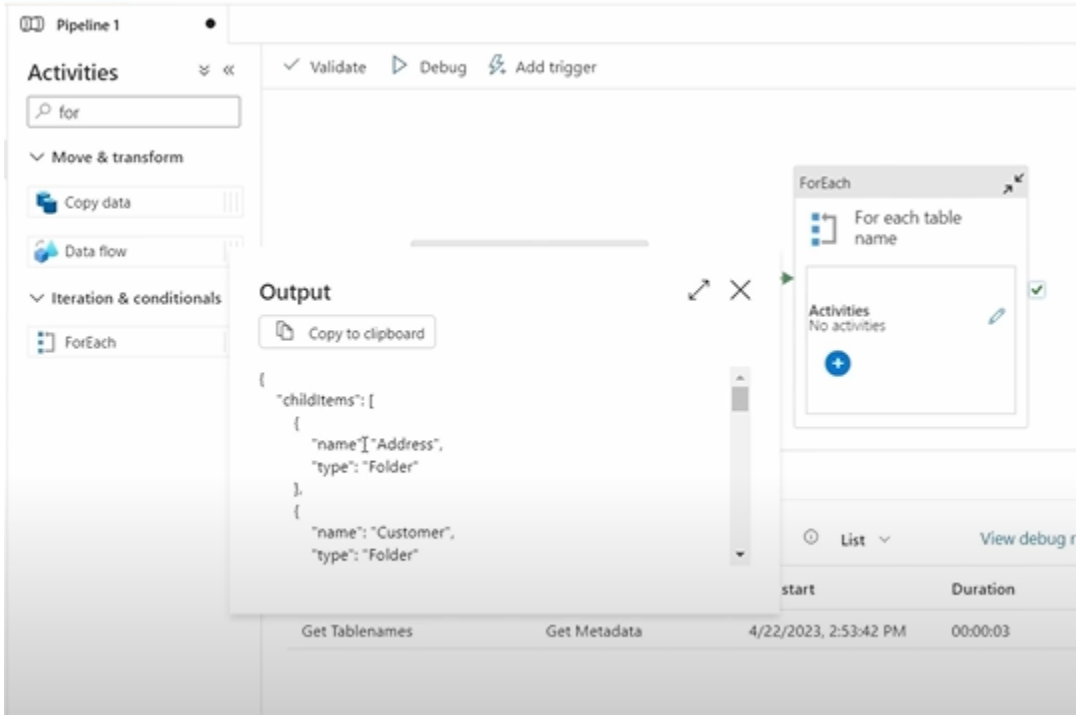


31. In the settings tab for items we will add dynamic content

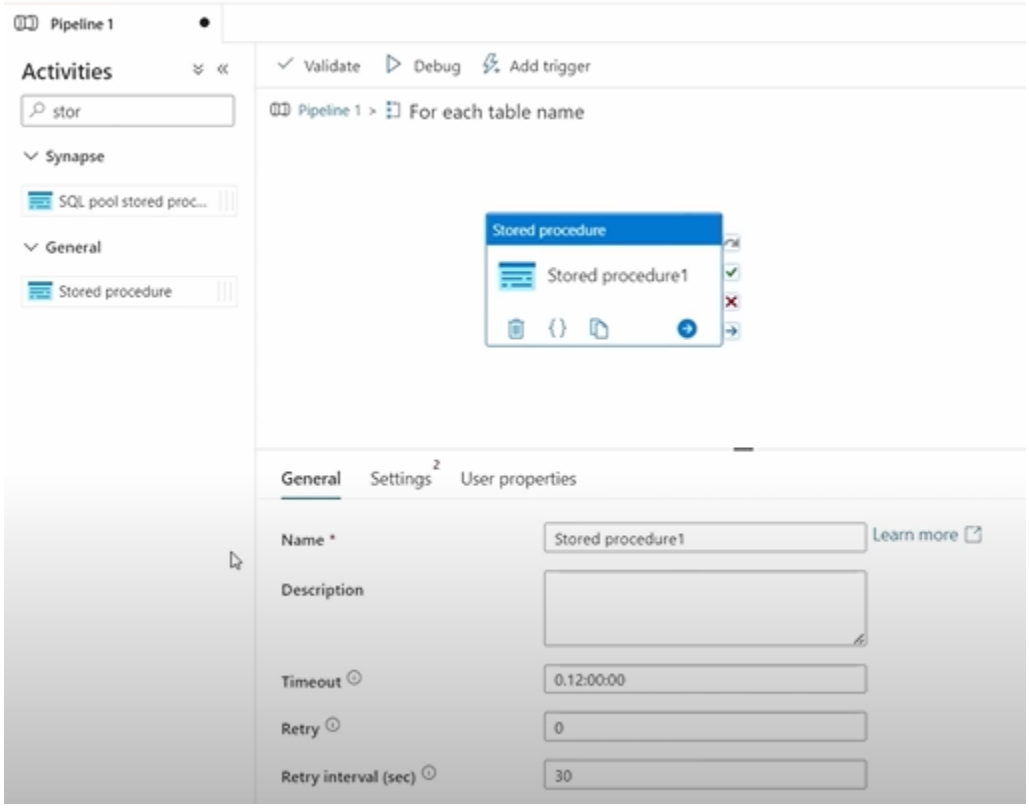




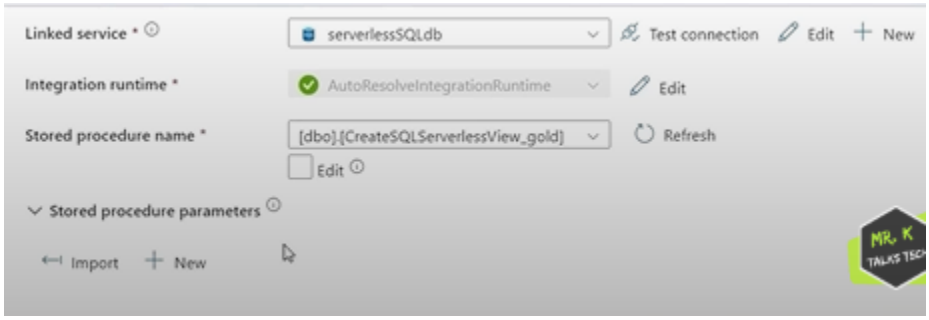
32. Now this whole list of child items…will passed as input in for each activity



33. Now inside the for each activity..we'll add storedProcedure activity

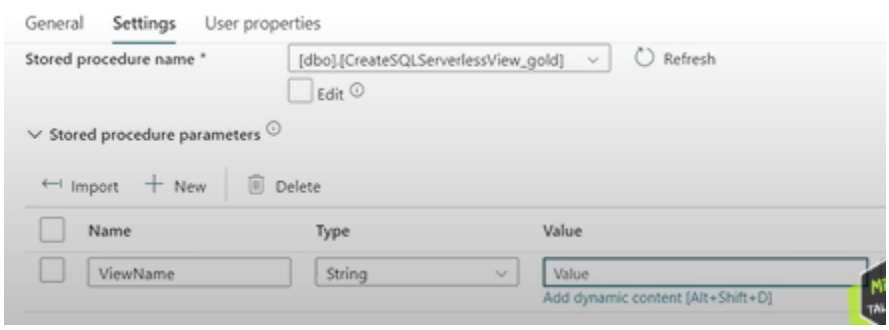


34. As we already have stored procedure in Serverless DB…in the settings tab..we'll link the serverless DB and stored Procedure

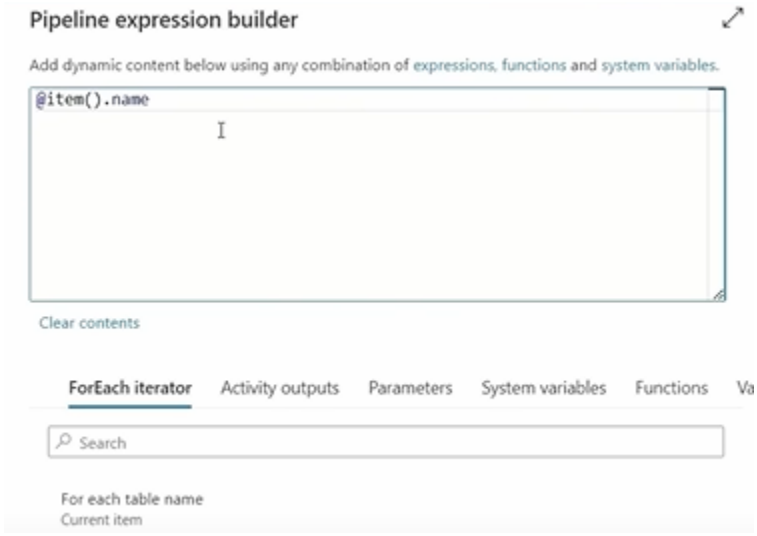


35. Next we click on the parameters and give this as parameter

```
@ViewName nvarchar(100)
```



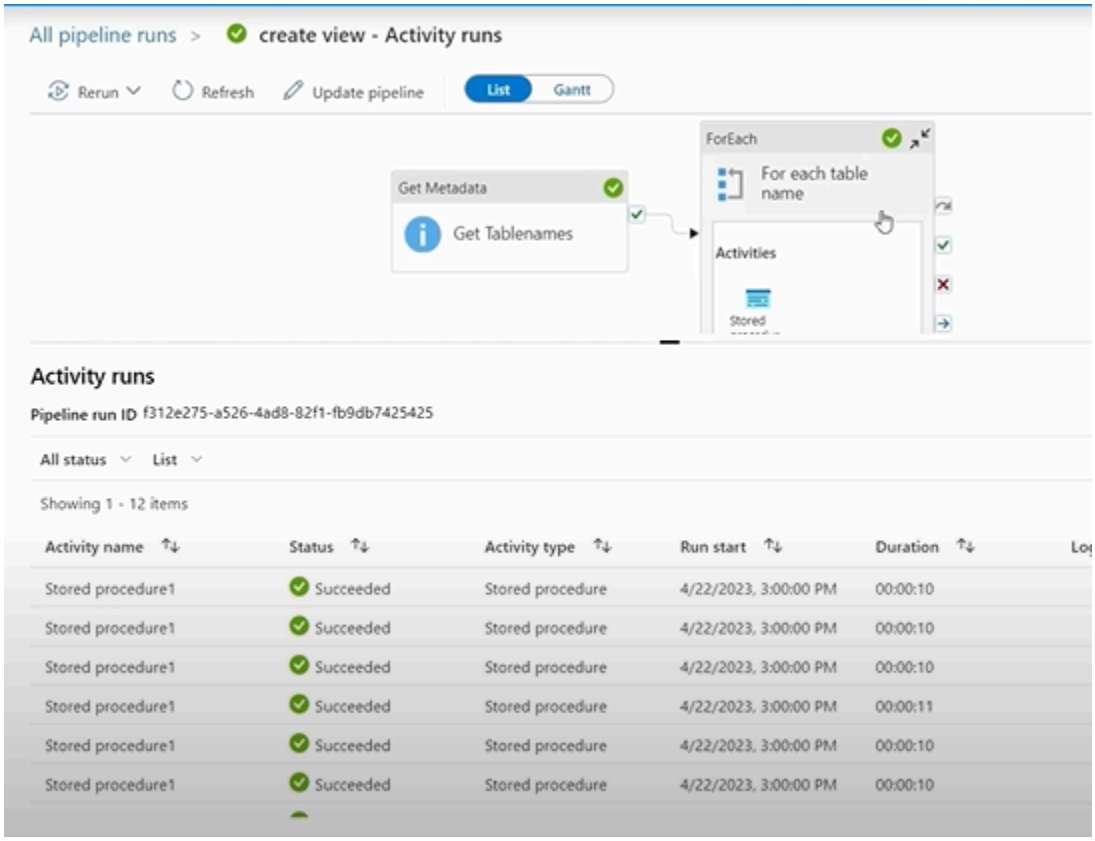
36. And for the value of stored procedure..we need to give the output of for each activity



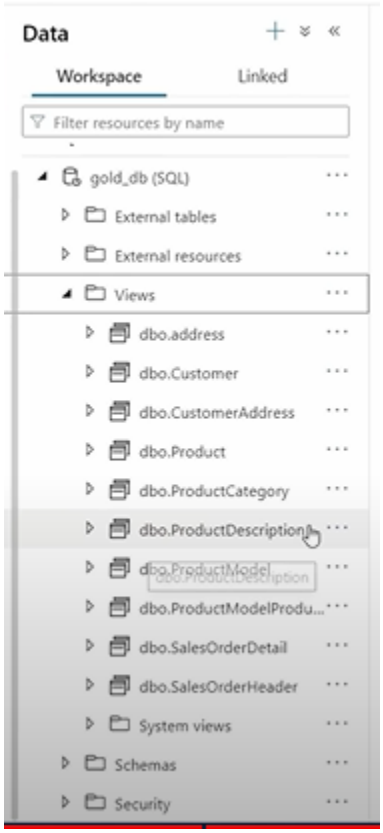
Now

37. So this stored procedure will get table name as the parameter from the outer loop..and it creates view for all the tables dynamically

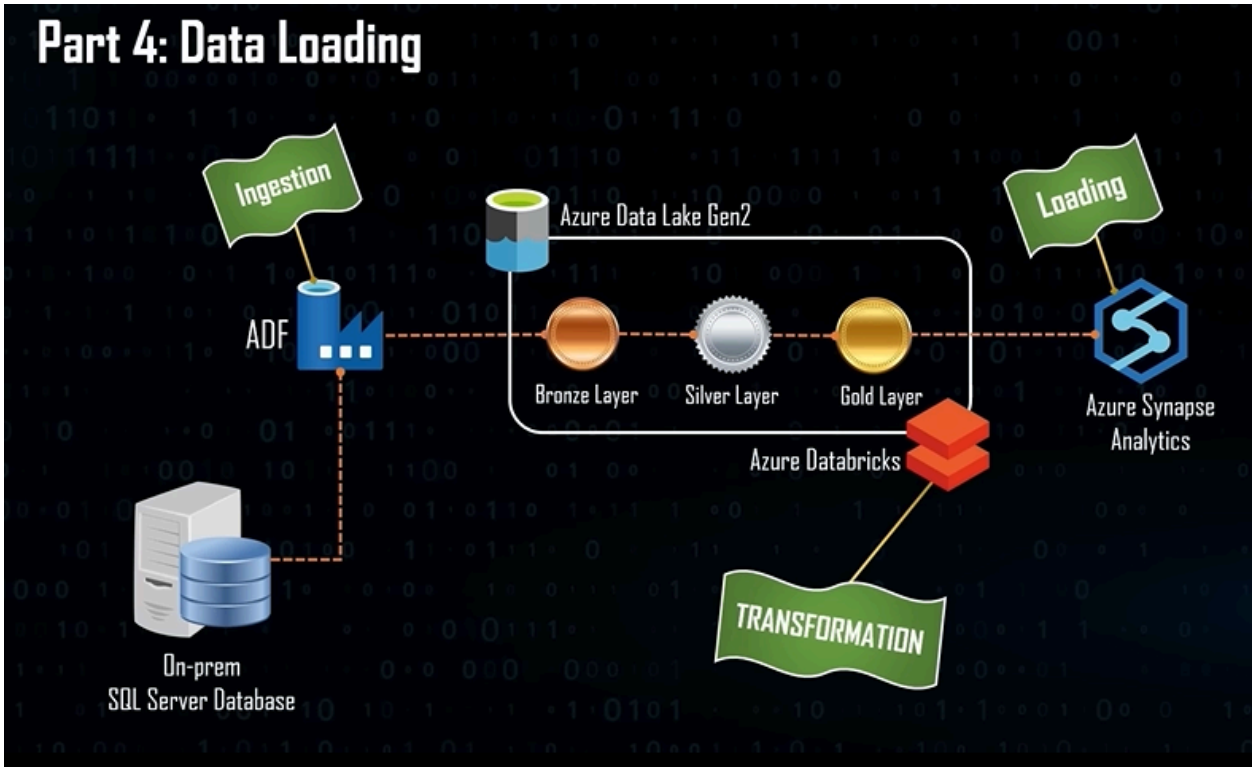
38. Now we'll publish all the changes and run this pipeline



39. Now after refreshing the our serverless DB..we can see the views created

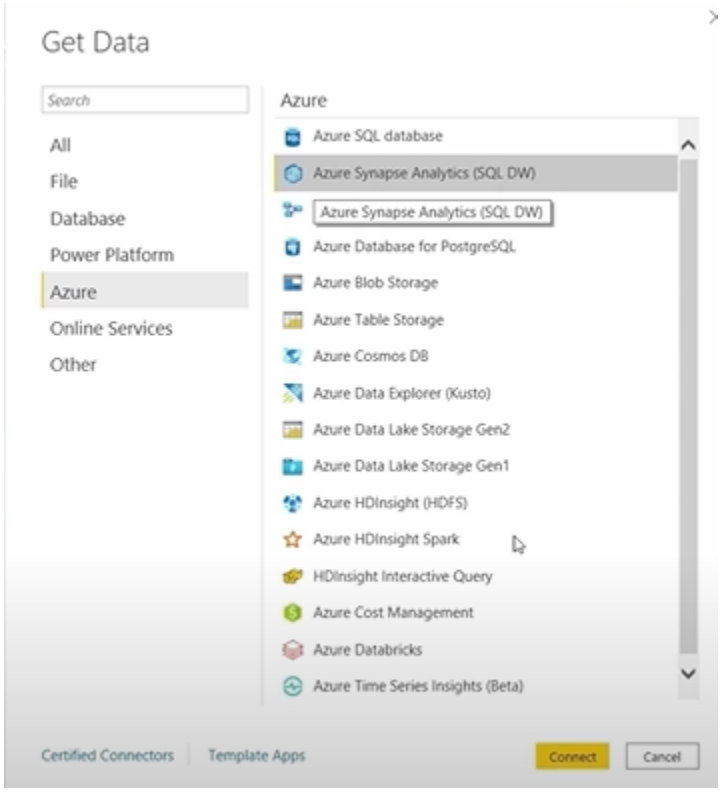


40. Now we have completed till part-4 ..where we have loaded data into Synapse

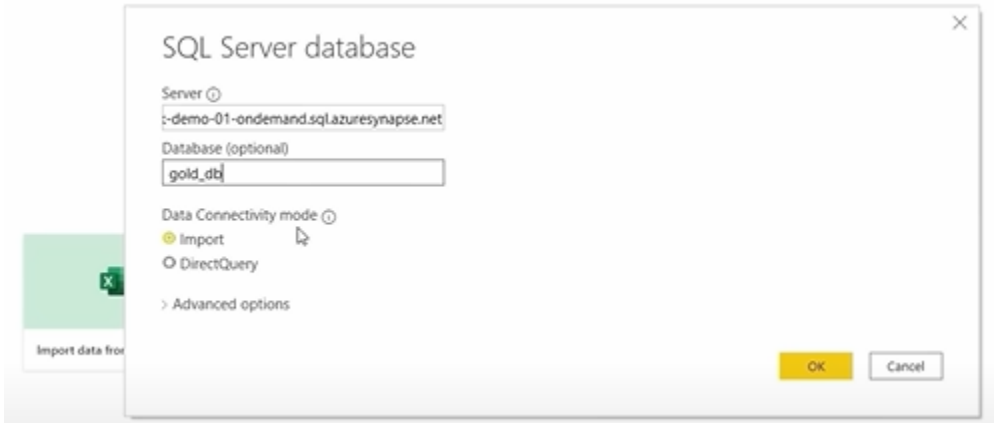


Data Reporting Using PowerBI

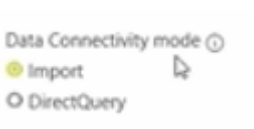
1. Now using the data in the serverless DB …we'll create reports using PowerBI
2. We'll connect our Azure Synapse services to get the data



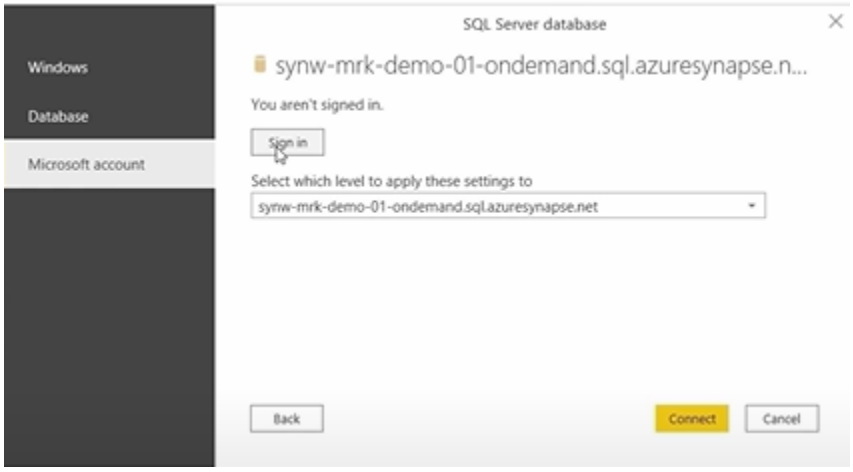
next we need to fill our gold_db end point and give the database name of ours



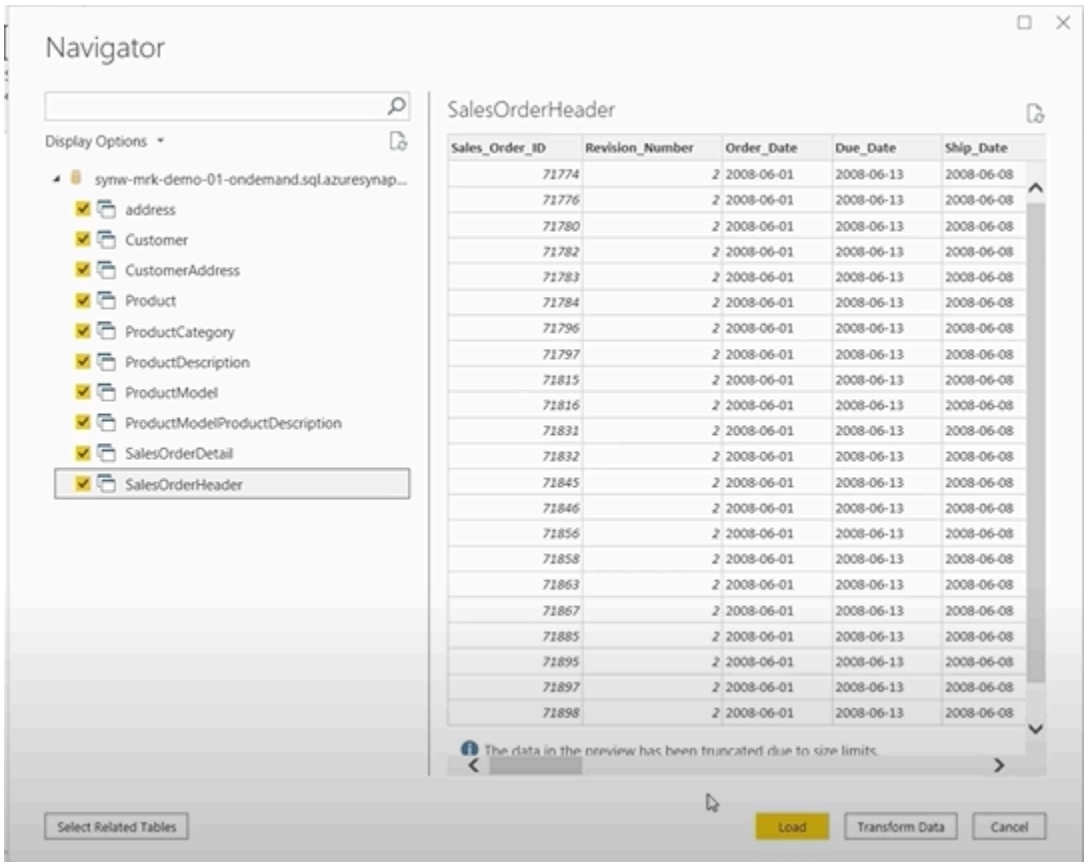
3. In data connectivity..we have 2 options



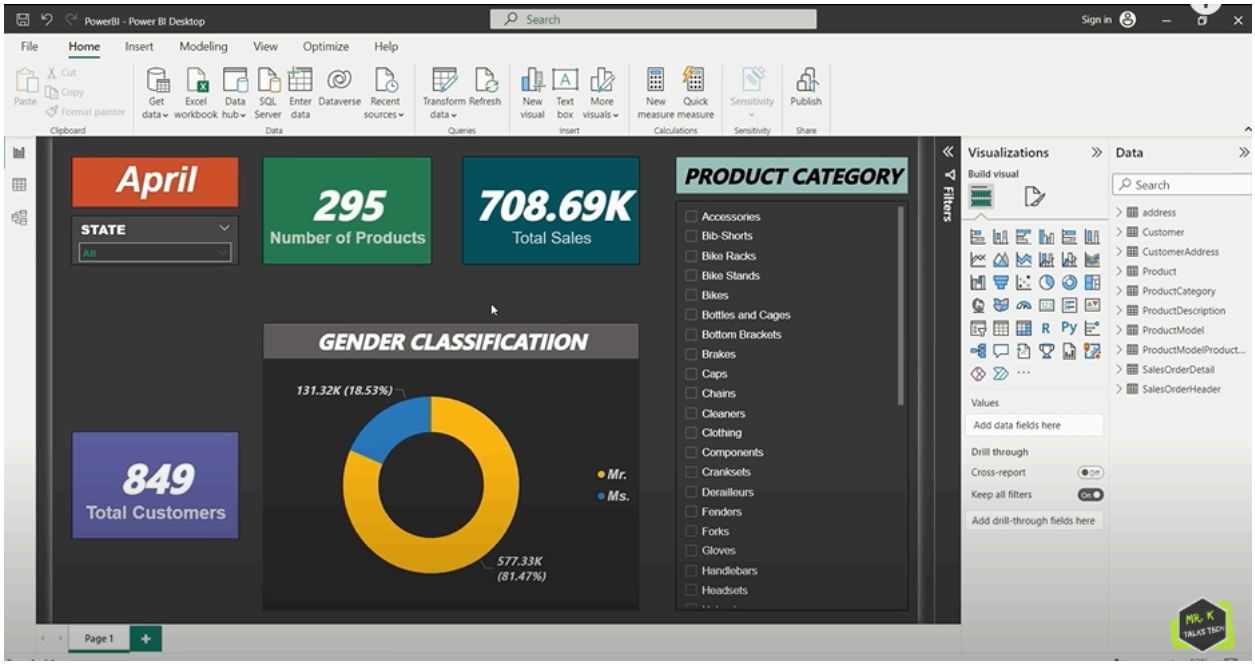
4. We'll authenticate to our azure Synapse DB endpoint using microsoft logins



5. Then we can choose to load the data or transform the data if required



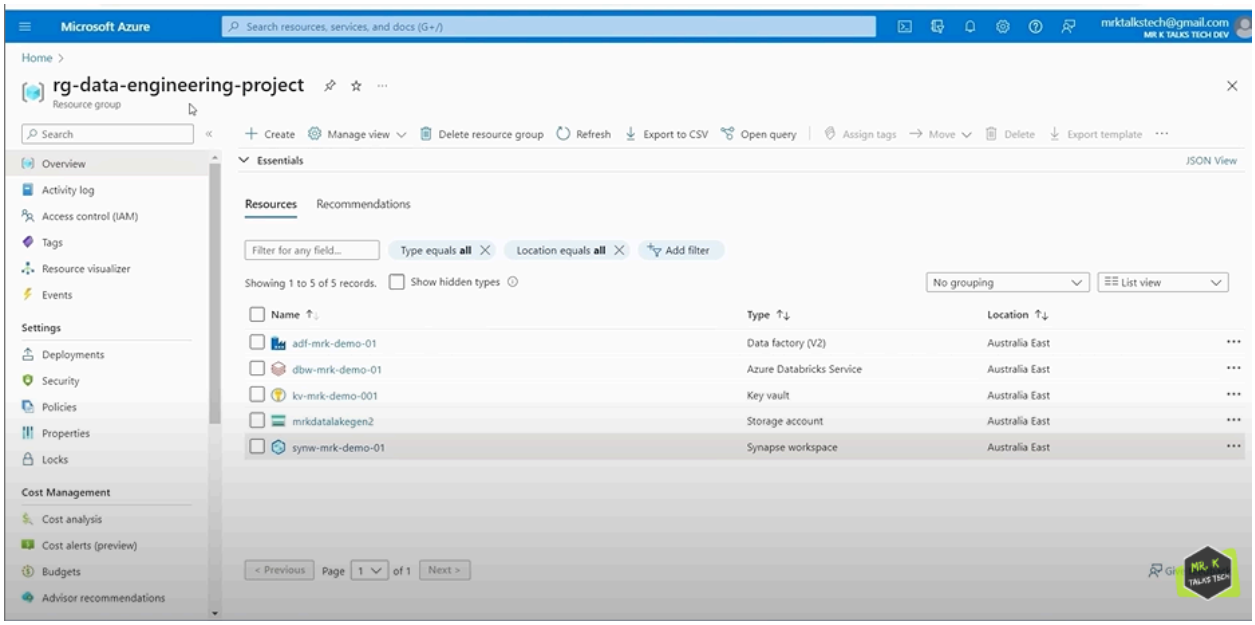
6. So using the powerBI tool we can create Dashboards like thois



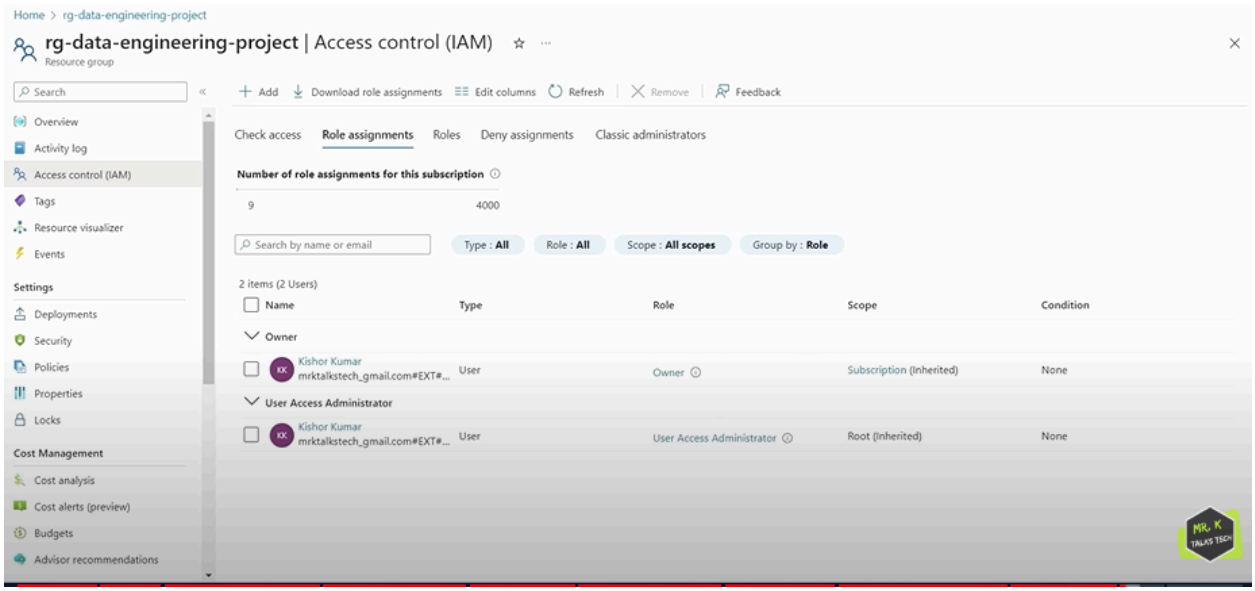
Security and Governance

1. 

2. We have this resource group in our account

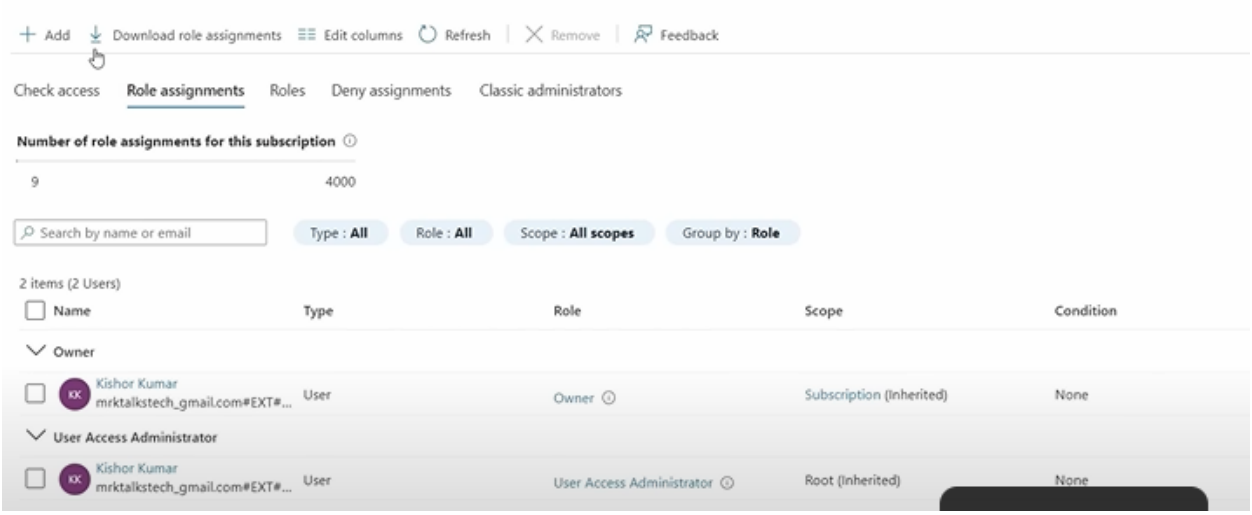


3. If we go to access control and then to role assignments..we can only see one user who has access to this resource grp

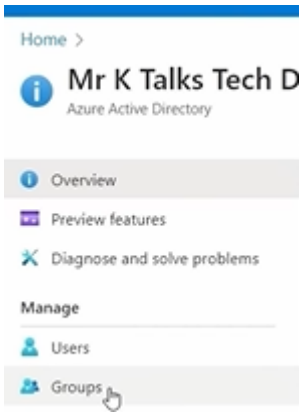


4. And here we have another user…who doesn't have access to this resource group
5. So we know that in real time..there will be multiple data engineer..who work on same project..and they need access to the project resources

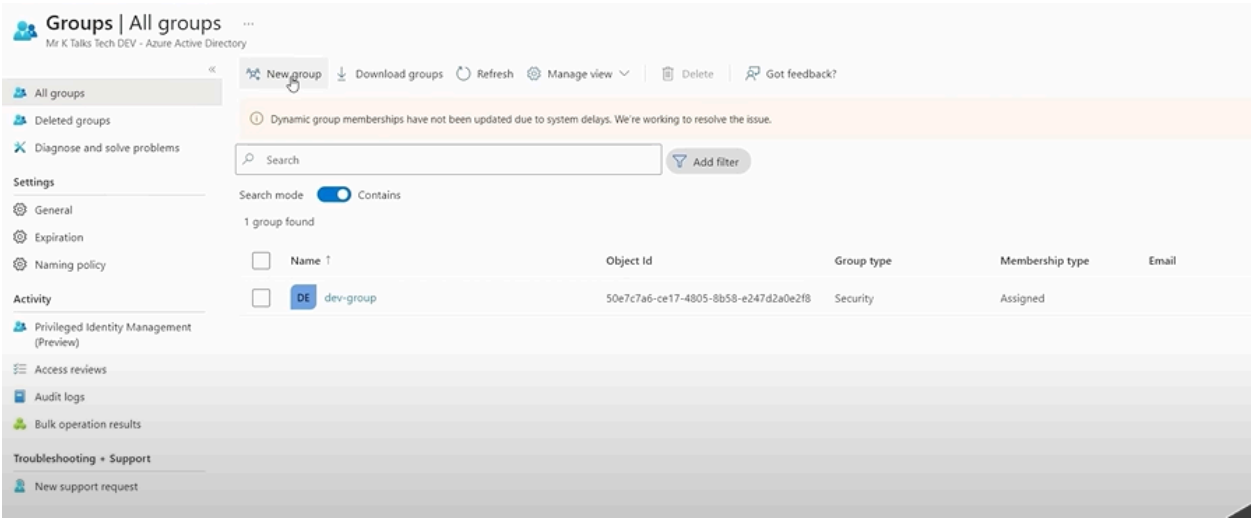
6. We cannot add each user individually here..which is time consuming



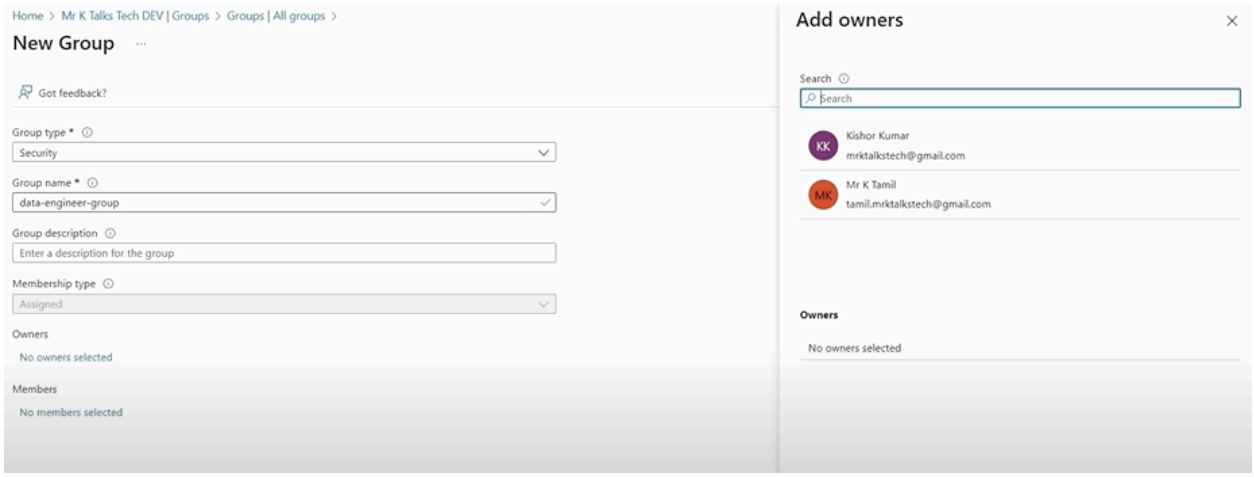
7. To resolve this..we can make use of Azure Active directory
8. For this we need to create security groups



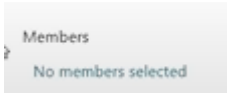
9. To create groups..go to Azure Ad …and click on Groups
10. So click on new group



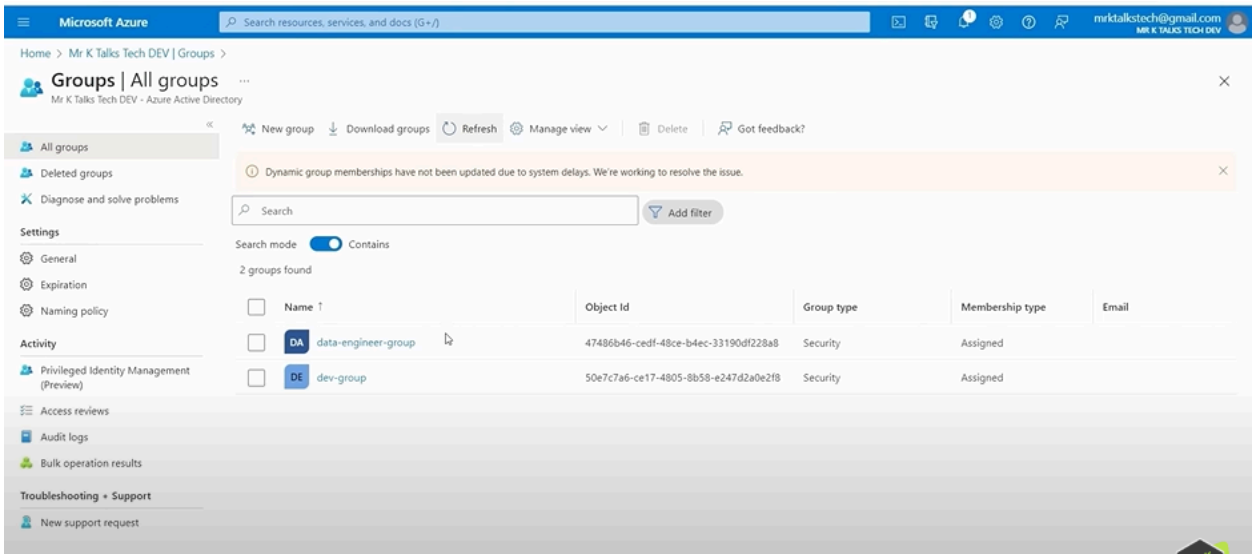
11. So we create a new group



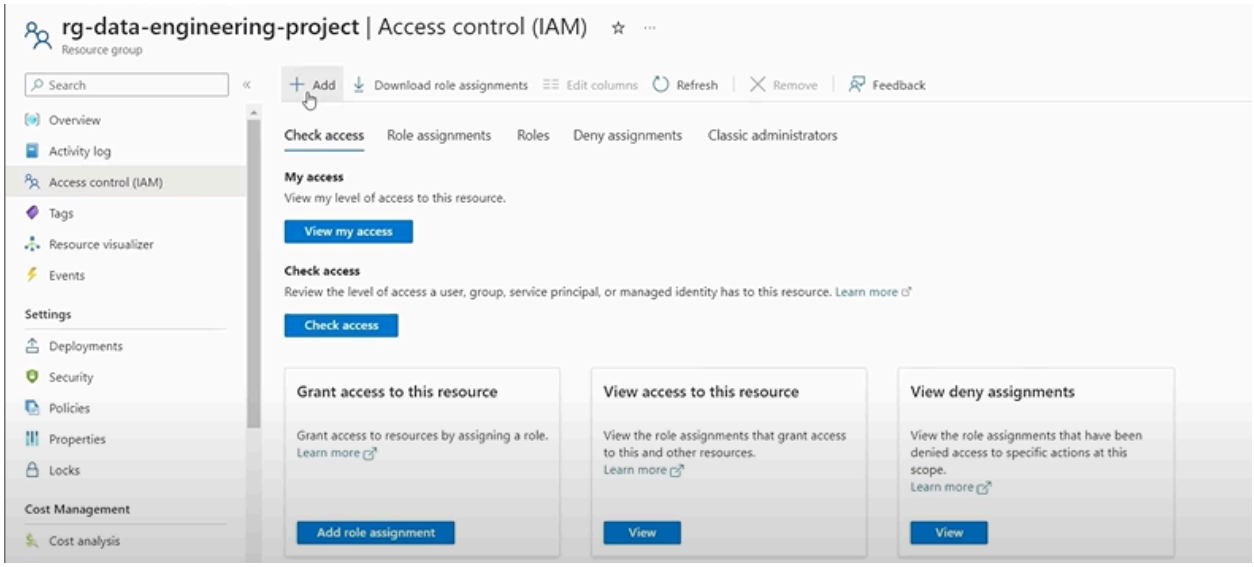
being the owner ..can add new users to the group or remove them

12. And we will add new members to the group 

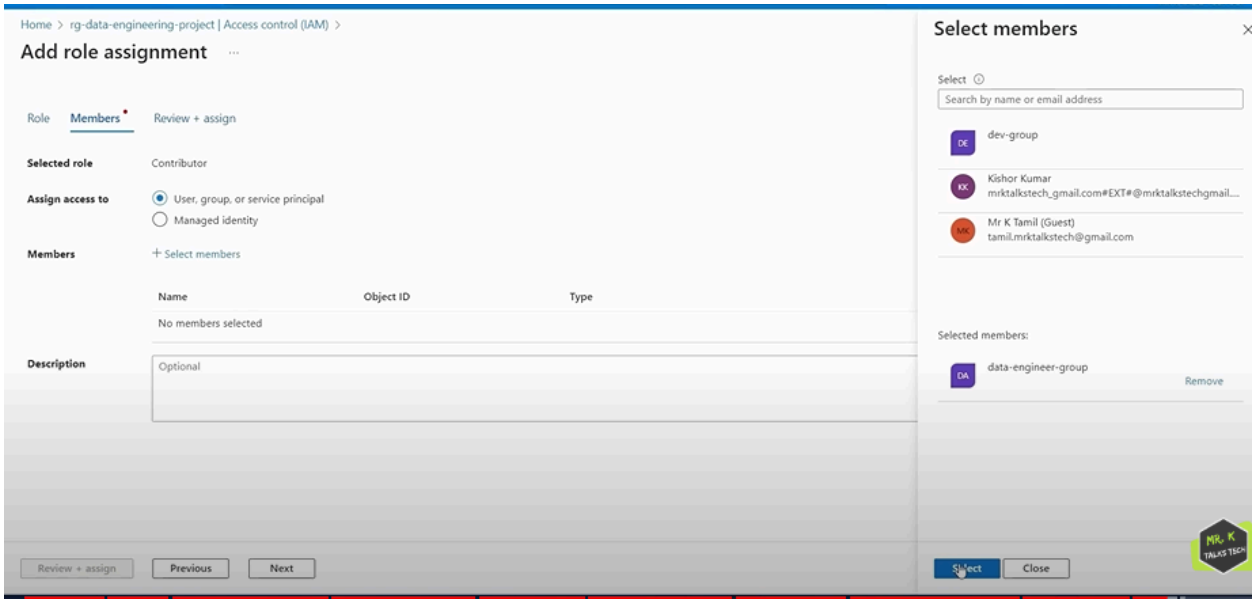
13. SO here we have created data engineer group



14. Now lets go back to the resource group and go to access control



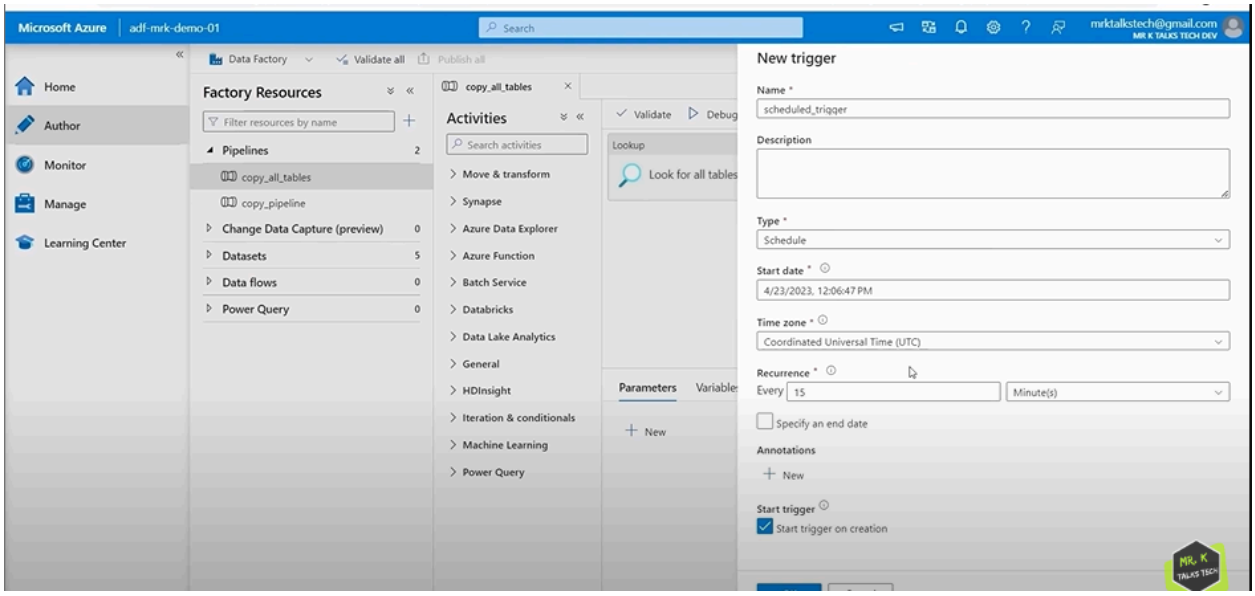
15. Now we'll click on add role assignment..and click on contributor and then next



16. Now we'll add our security group which we have created click on next and save
17. Till here we have completed the data pipeline

End to End Pipeline Testing

1. We'll test this pipeline by adding a new row in our SQL Onperm Database



2. So we'll add a schedule trigger…that triggers our pipeline