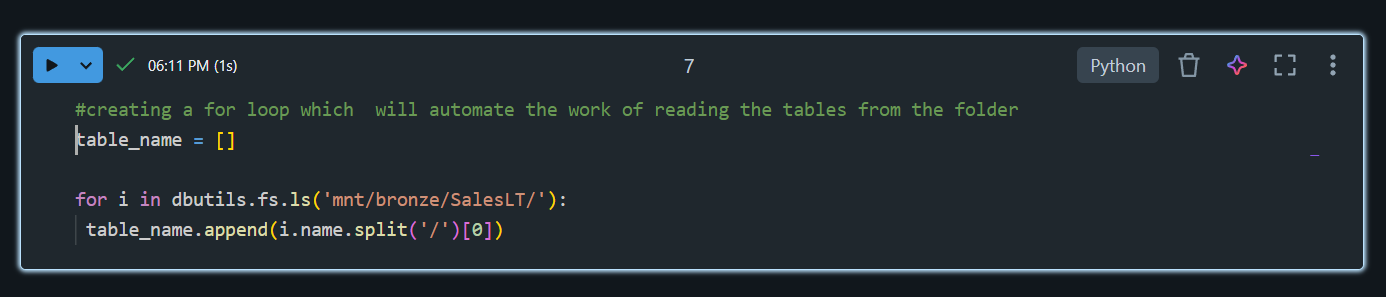
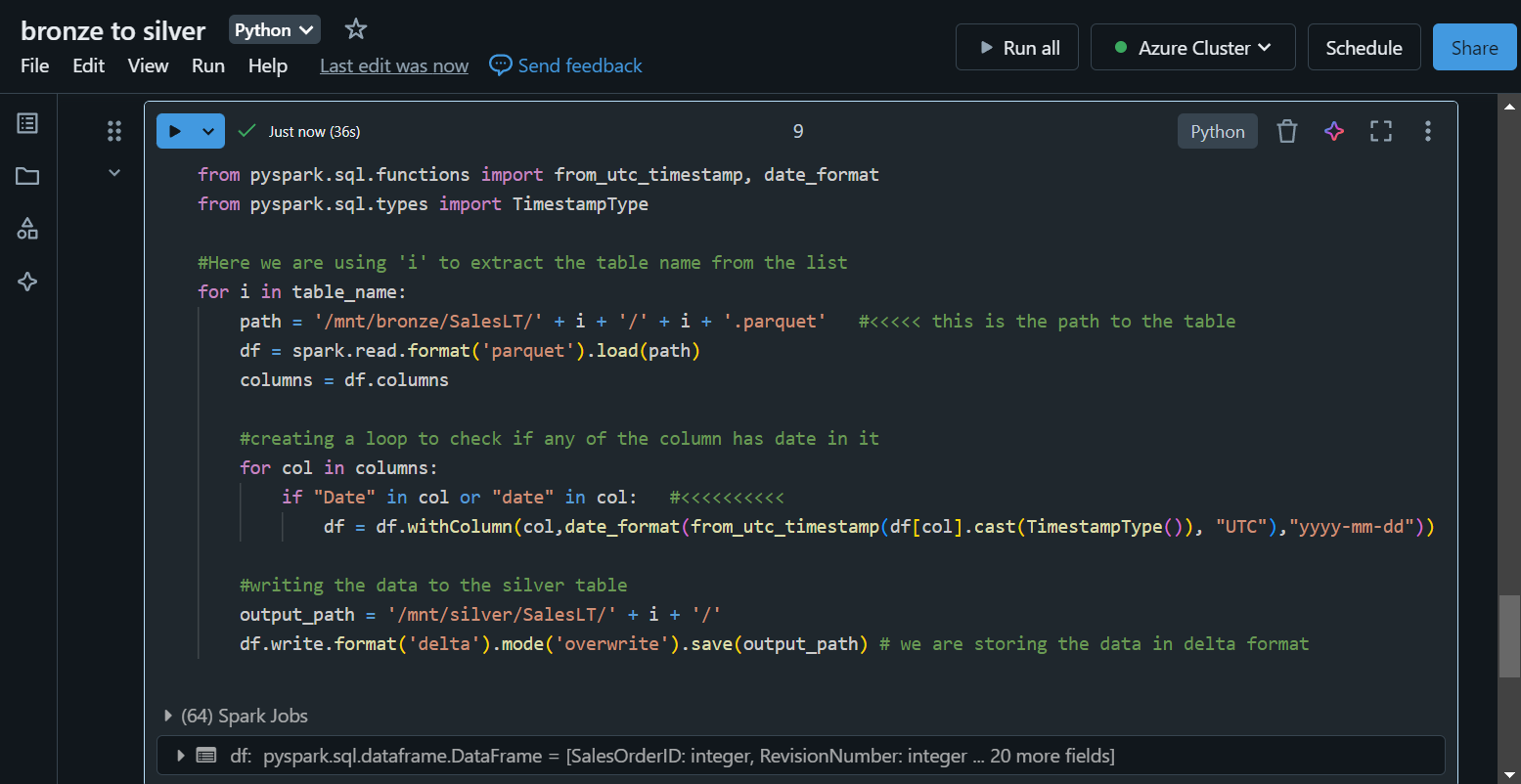
**Transformation -1**

🡪First level of transformation will include converting the date column from timestamp format to date format.

🡪We will be writing code which will transform the data of the date column of all the table at once.

* Create a new notebook in the same folder and name it Bronze to Silver





CODE:

table\_name = []

for i in dbutils.fs.ls('mnt/bronze/SalesLT/'):

 table\_name.append(i.name.split('/')[0])

from pyspark.sql.functions import from\_utc\_timestamp, date\_format

from pyspark.sql.types import TimestampType

#Here we are using 'i' to extract the table name from the list

for i in table\_name:

    path = '/mnt/bronze/SalesLT/' + i + '/' + i + '.parquet'   #<<<<< this is the path to the table

    df = spark.read.format('parquet').load(path)

    columns = df.columns

    #creating a loop to check if any of the column has date in it

    for col in columns:

        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"))

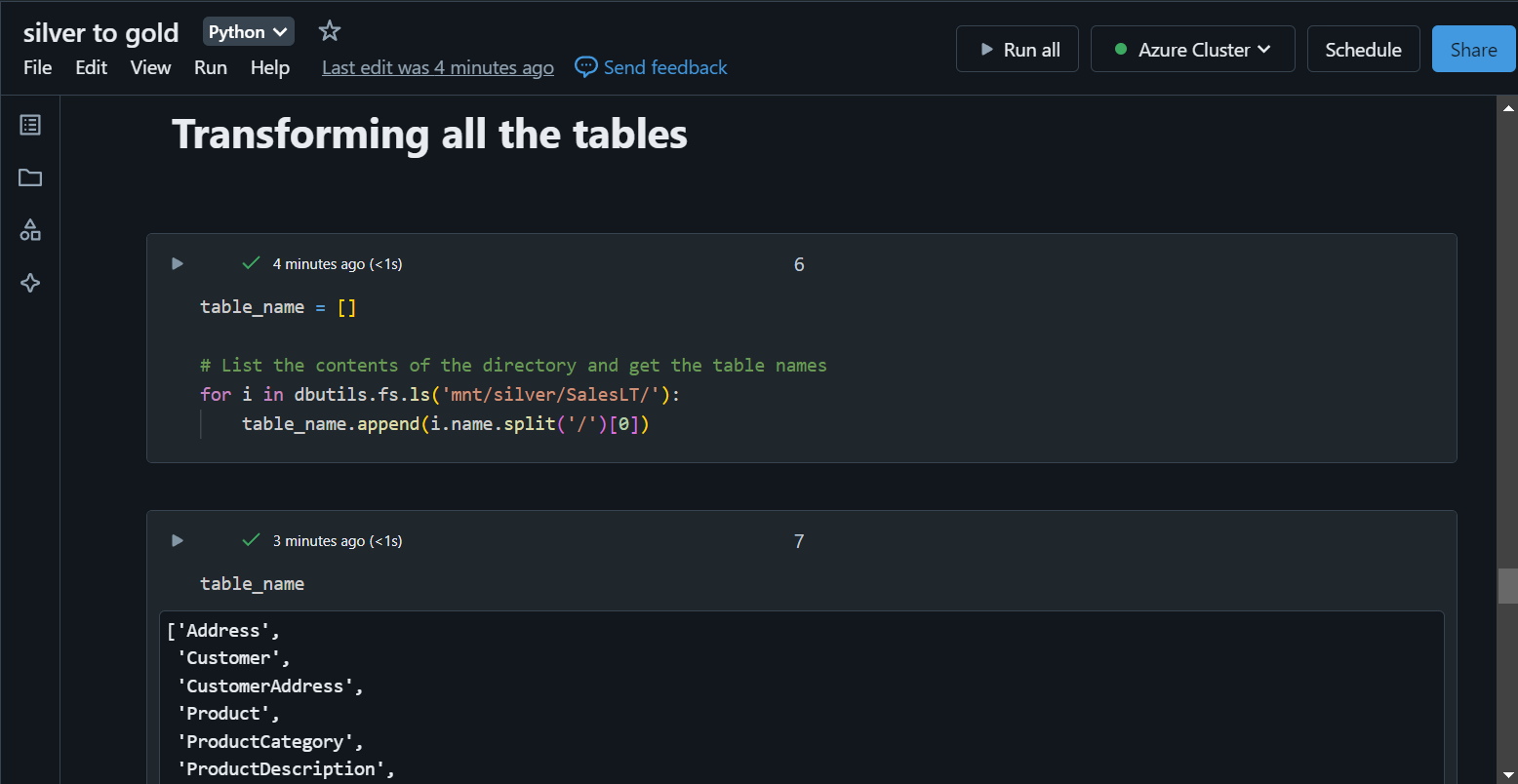
    #writing the data to the silver table

    output\_path = '/mnt/silver/SalesLT/' + i + '/'

    df.write.format('delta').mode('overwrite').save(output\_path) # we are storing the data in delta format

🡪Level two transformation:**Silver to gold**

* Joining multiple tables together
* Doing data aggregations etc.



Followed by the below code:  
# Loop through each table name

for name in table\_name:

    # Construct the path for reading the delta table

    path = '/mnt/silver/SalesLT/' + name

    print(f"Processing table: {name}, Path: {path}")

    # Read the delta table into a DataFrame

    df = spark.read.format('delta').load(path)

    # Get the list of column names

    column\_names = df.columns

    # Loop through each column name and rename them

    for old\_col\_name in column\_names:

        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("\_")

        # Rename the column in the DataFrame

        df = df.withColumnRenamed(old\_col\_name, new\_col\_name)

    # Define the output path for writing the transformed table

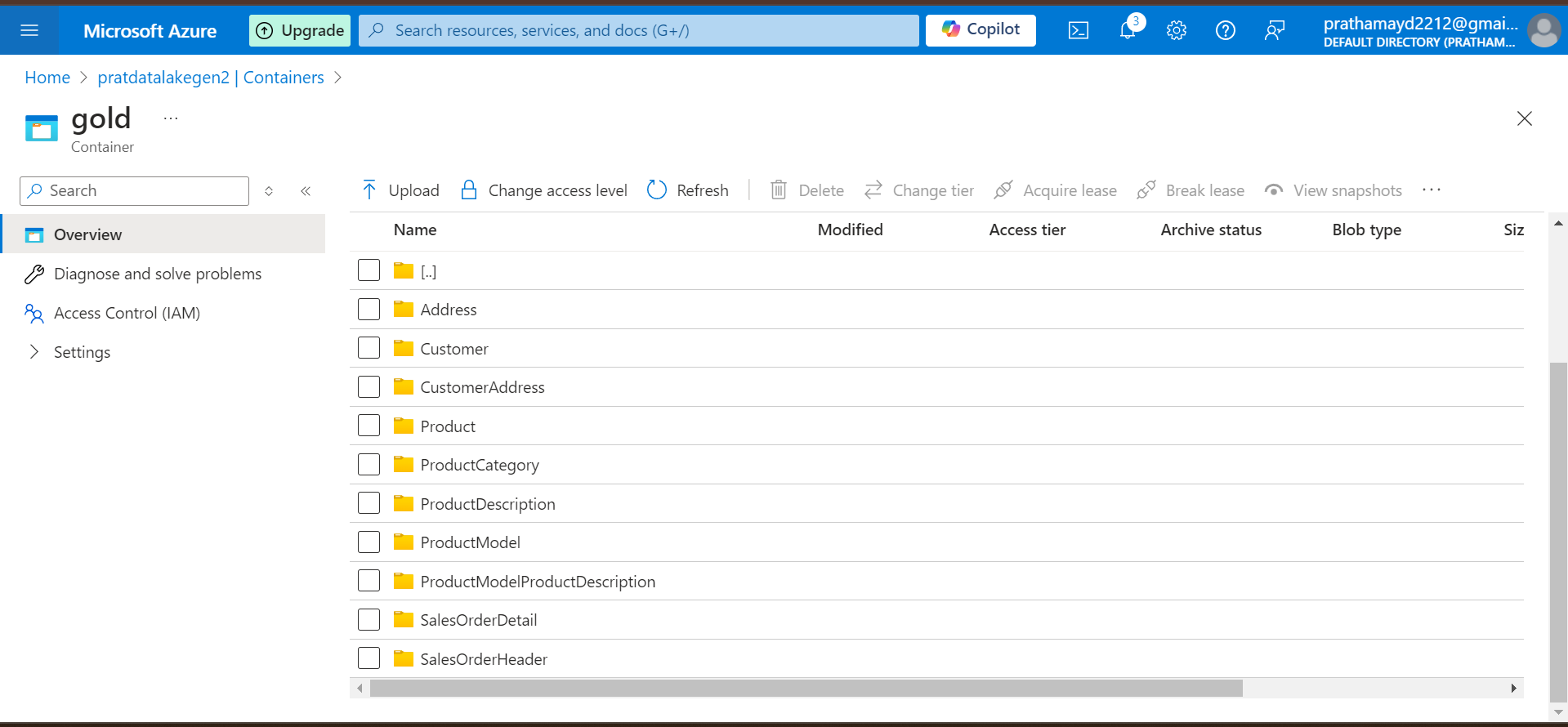
    output\_path = '/mnt/gold/SalesLT/' + name + '/'

    # Write the transformed DataFrame back to the output path

    df.write.format('delta').mode("overwrite").save(output\_path)

    print(f"Table {name} written to {output\_path}")

**Gold container now has all the tables:**

****

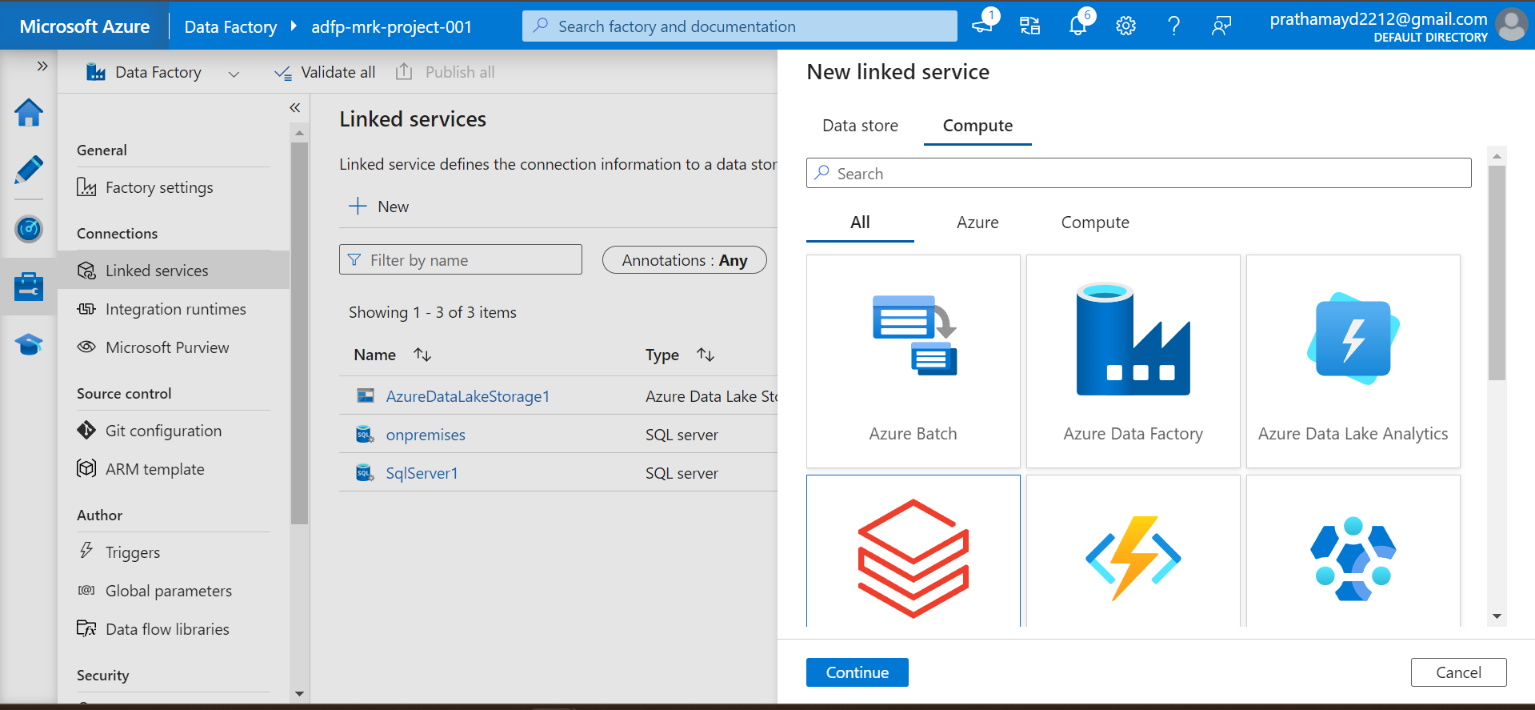
Now we will be creating Jobs to run the two notebooks (bronze to silver and silver to gold)

As the storagemount notebook needs to be run only once.

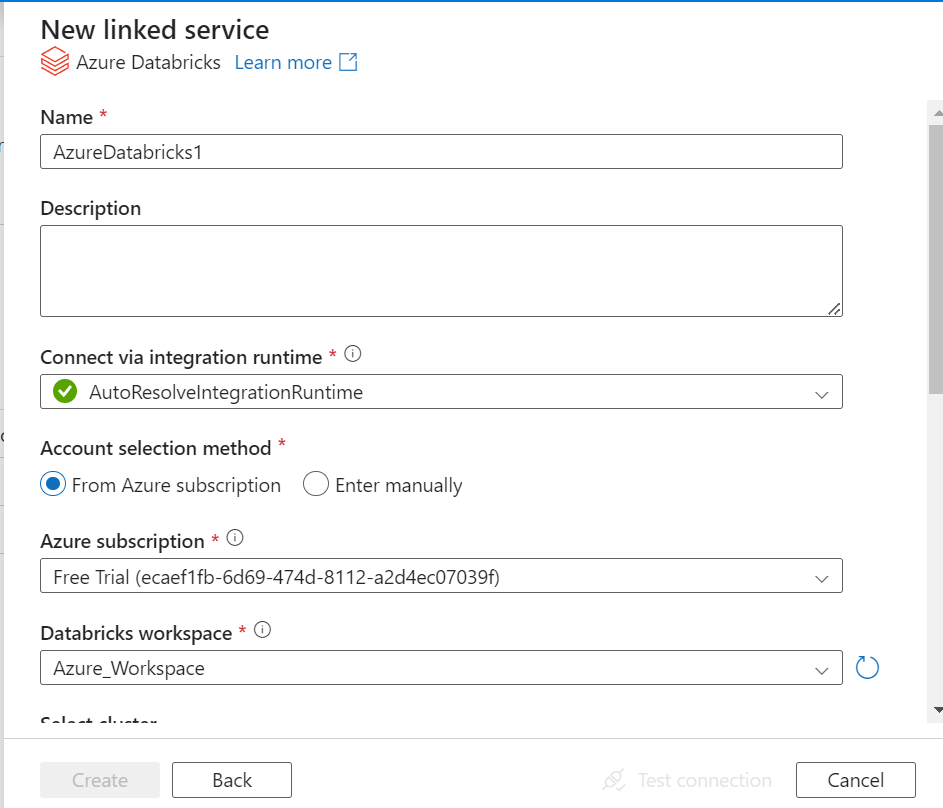
🡪We will be using Azure Data Factory for this purpose.

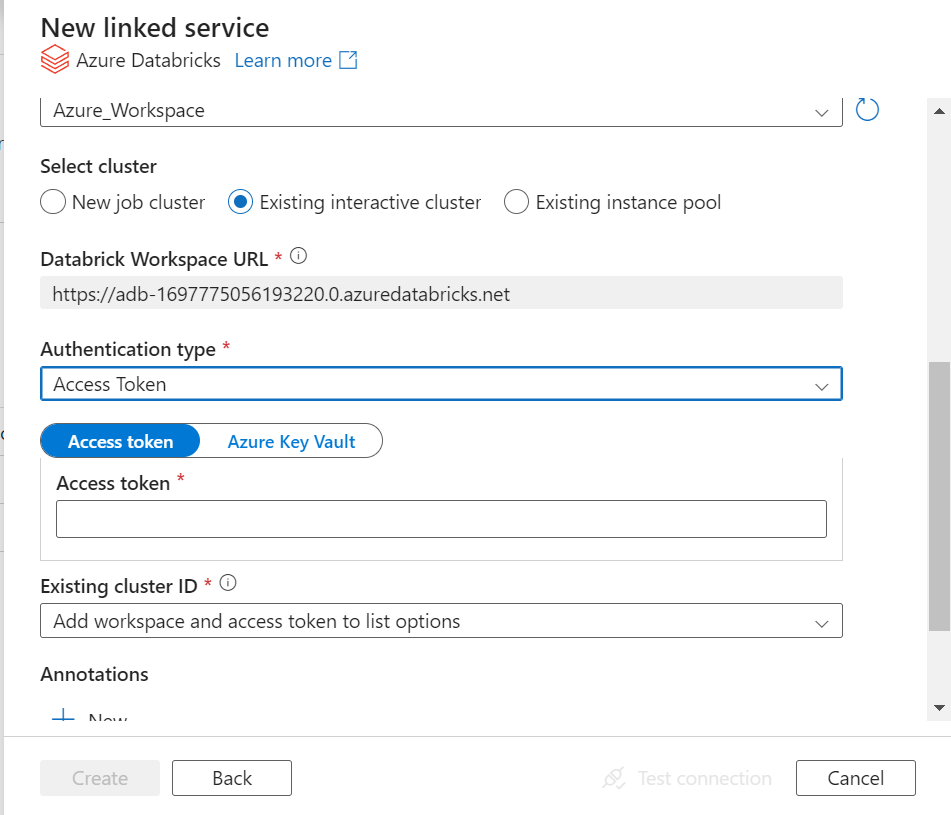
* We need to establish connection between the Azure Databricks and Azure Data Factory

Go to Azure Data Factory🡪Manage 🡪New linked service 🡪Azure Databricks



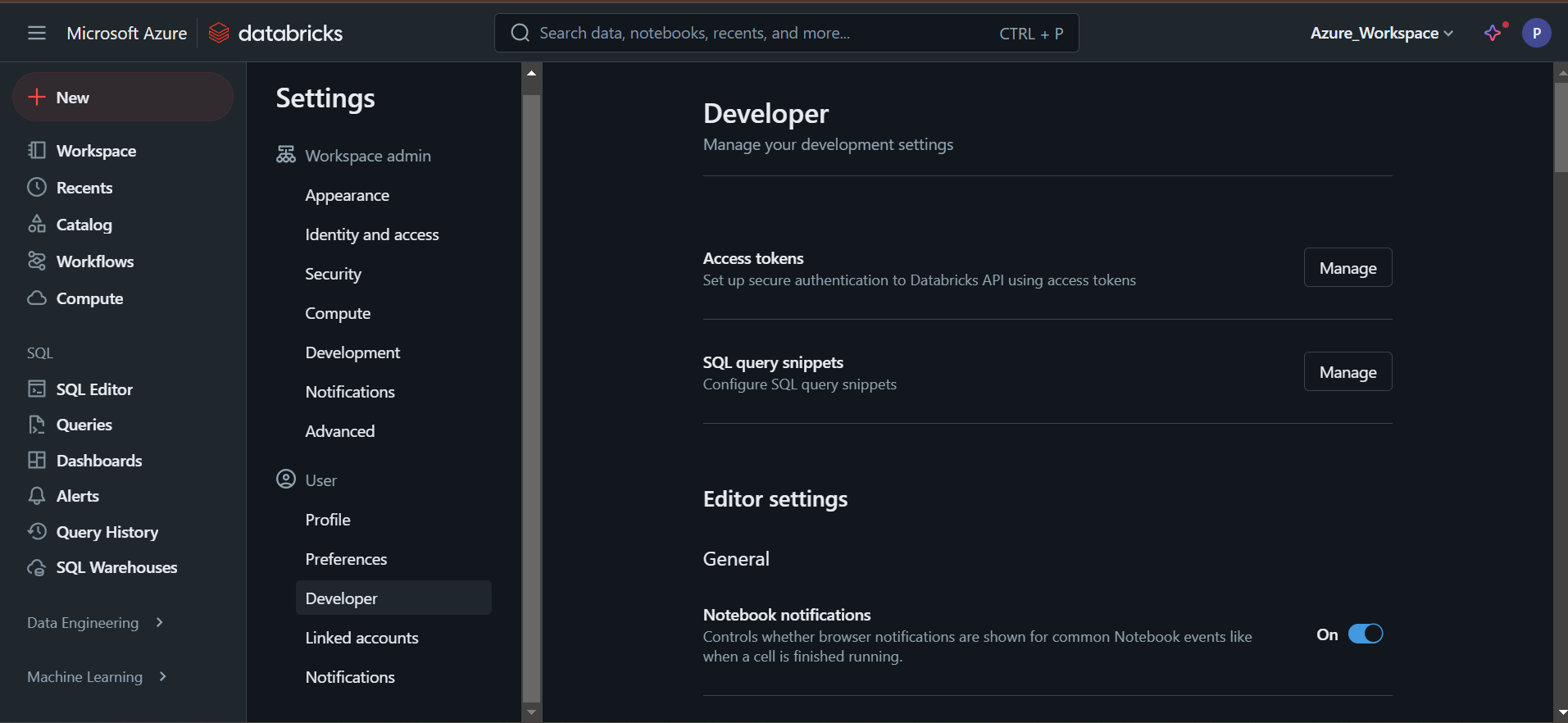
Enter the details:





Create access token in Databricks:

Go t azure databricks settings 🡪under users🡪 Developer 🡪 Access Tokens 🡪 Generate new token



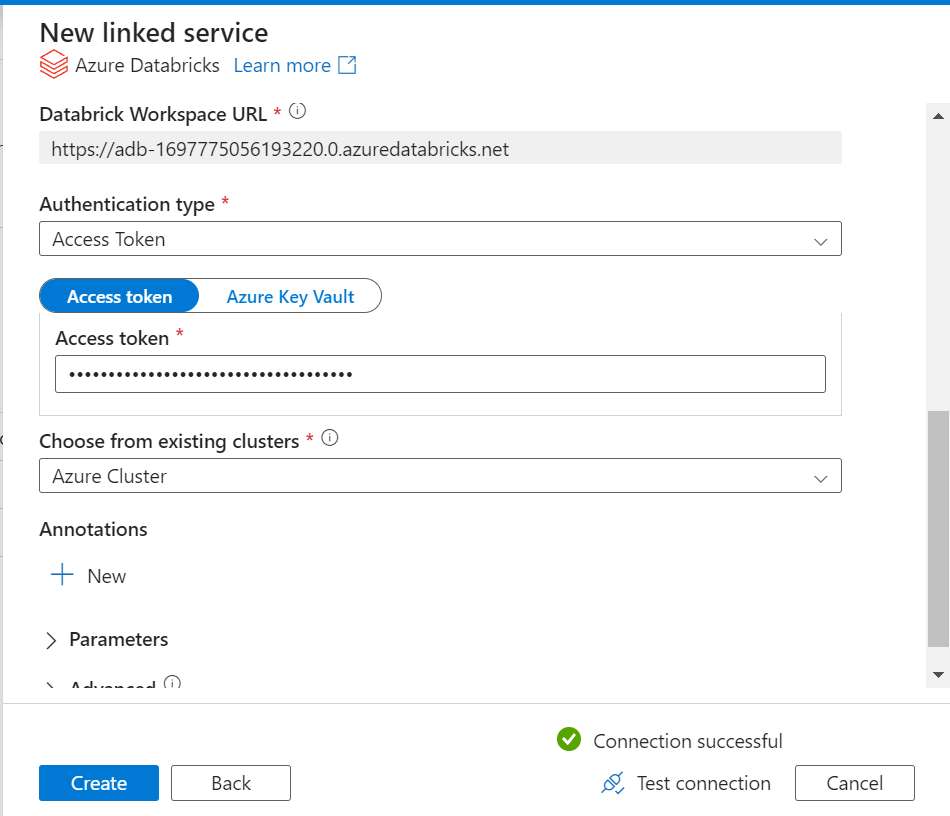
Token will be viewed only once so copy it and paste it somewhere.

dapi97a455248b120c6dba7ffc2984ba8494

🡪Now create a secret and add the token in the secret and then use the secret in the linked service.

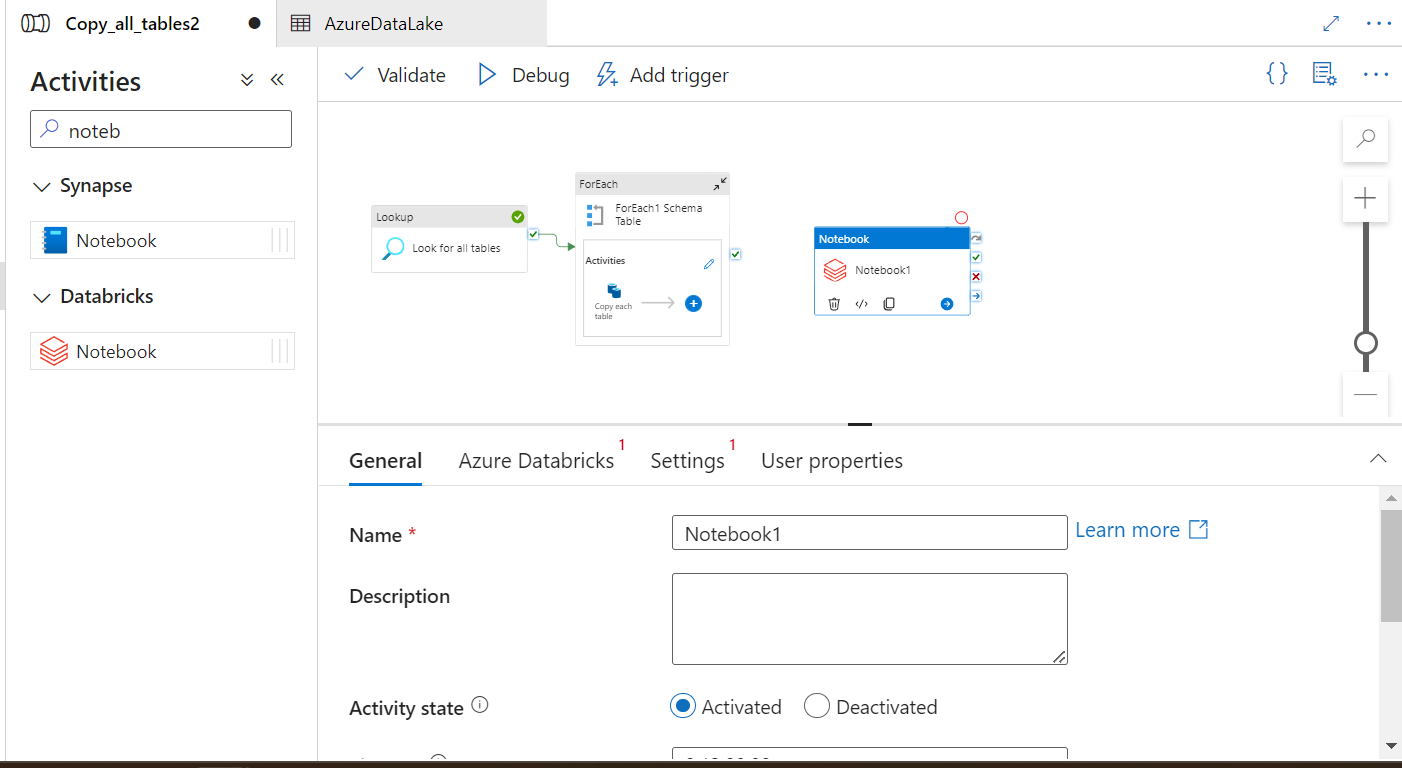
Or Directly type in the Access Token.

And test connection:

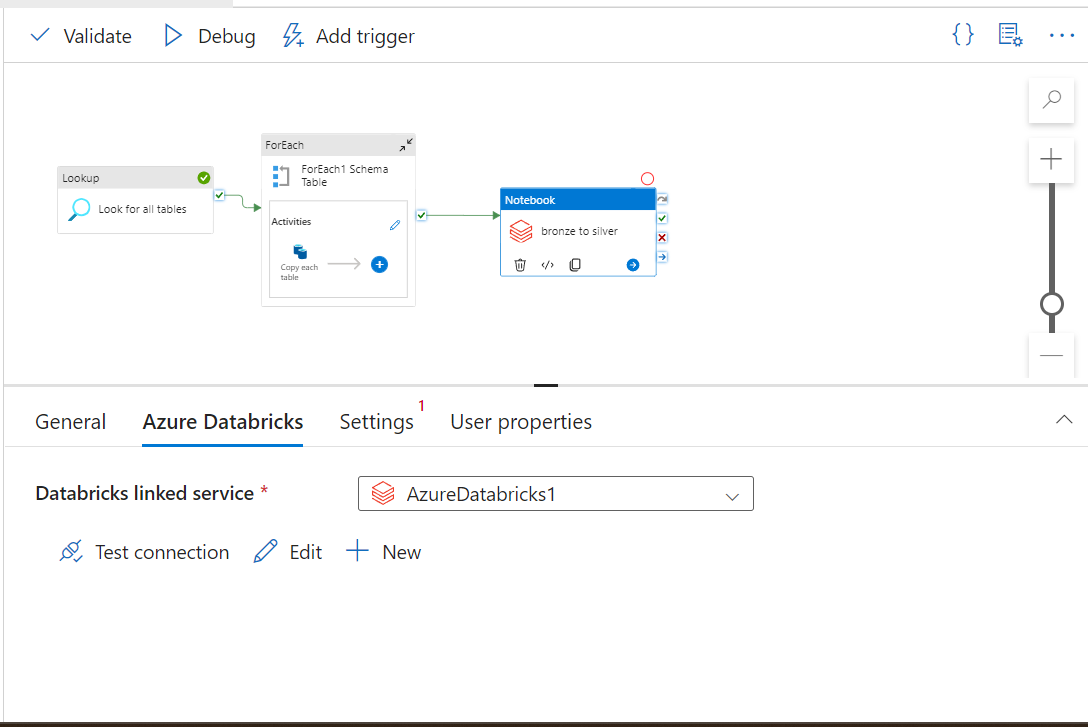


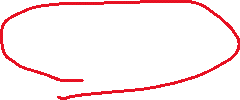
Once created, click Publish all to publish all the changes.

Navigate to ADF🡪Authors🡪 in your existing pipeline add a notebook activity.

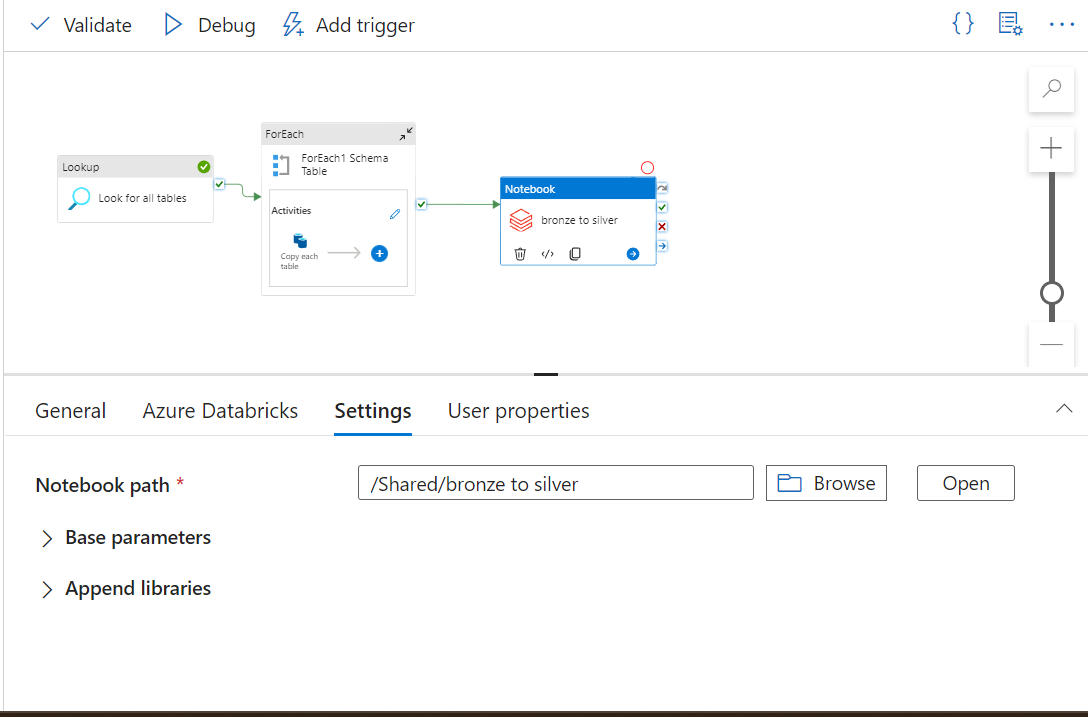


Link the service:

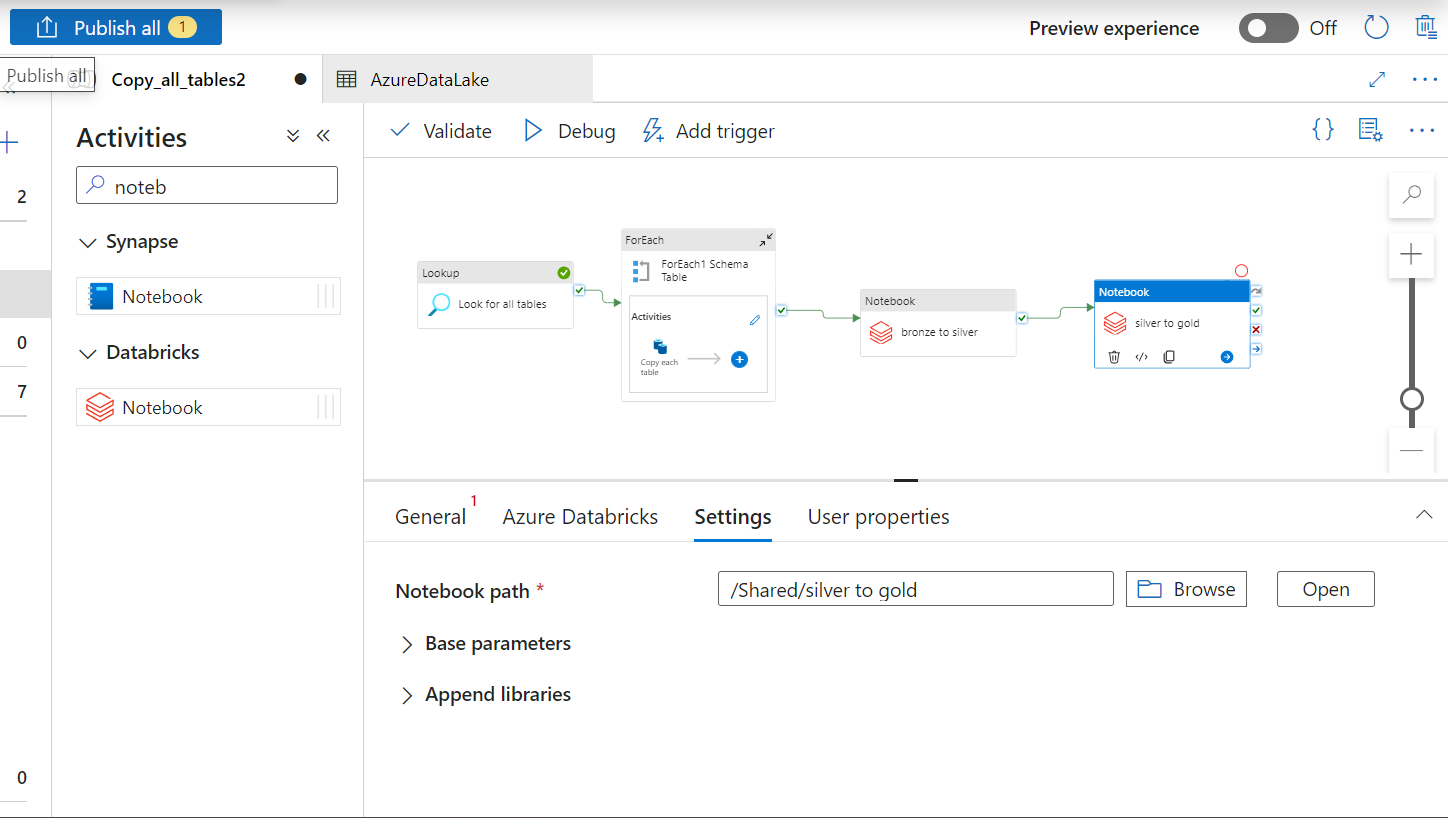




Go to settings 🡪 browse and add the bronze to silve notebook



Create another notebook activity and attach silver to gold notebook to the same

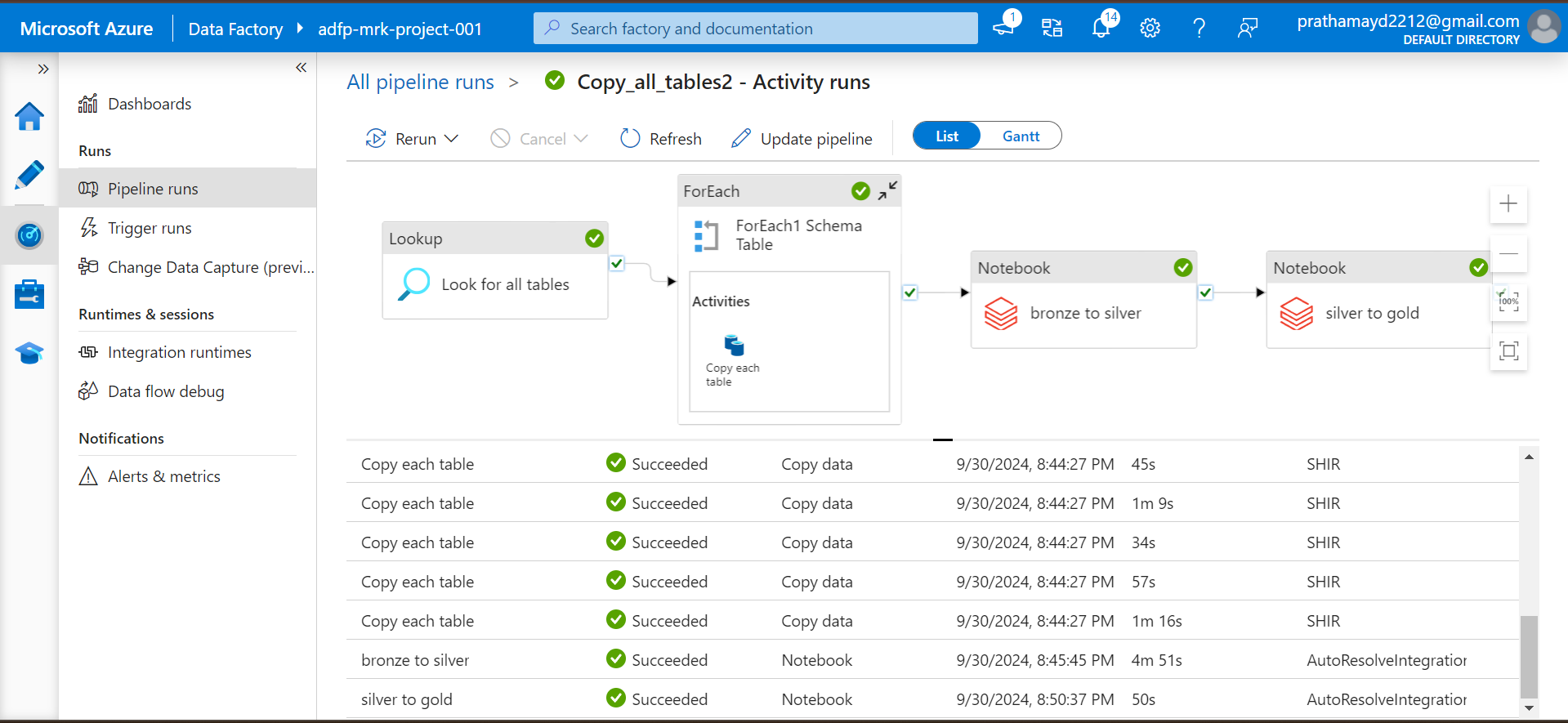


So an end to end pipeline has been created.

Click on publish all to save all changes.

* Once changes are published, we can trigger the pipeline to run now.

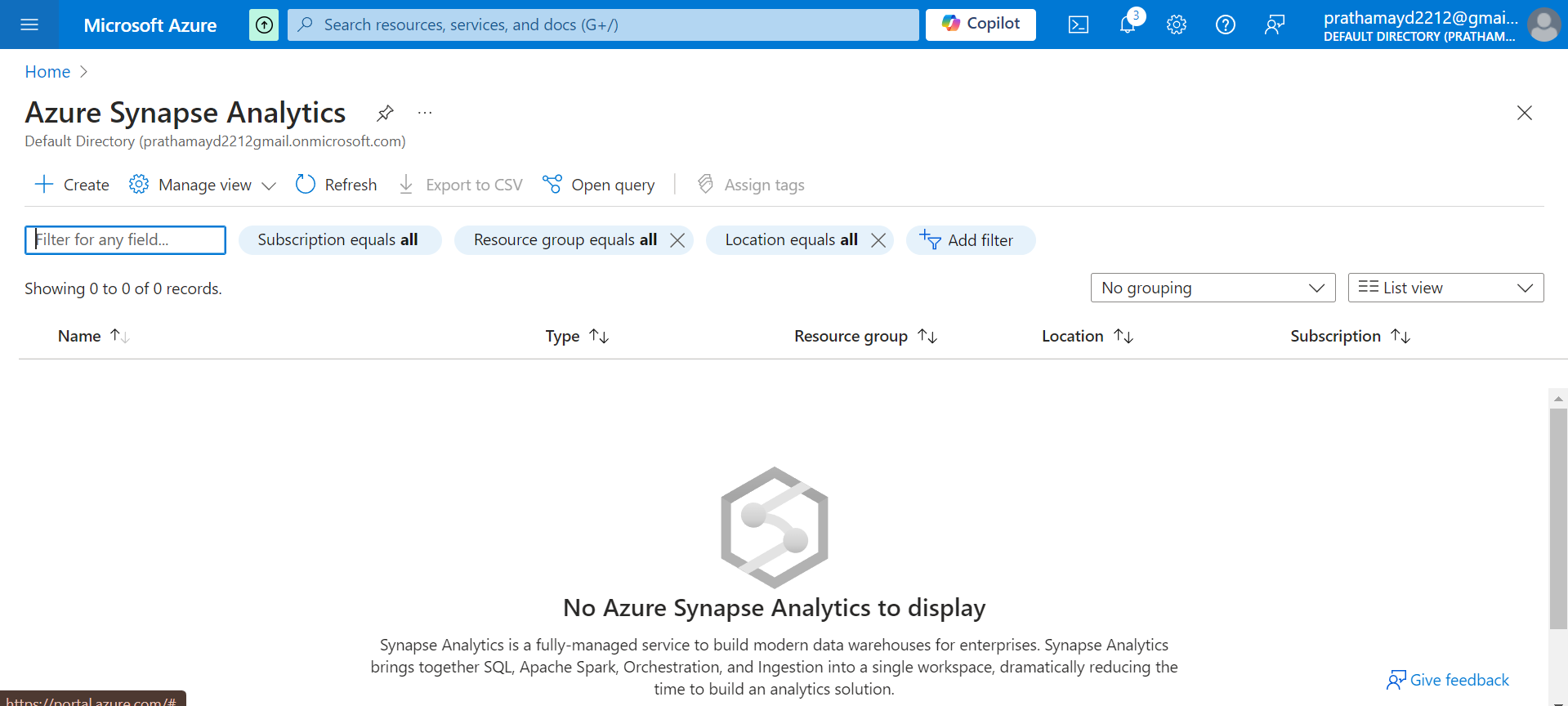
The pipeline runs successfully then you can move to the next step where you will integrate Azure Synapse.



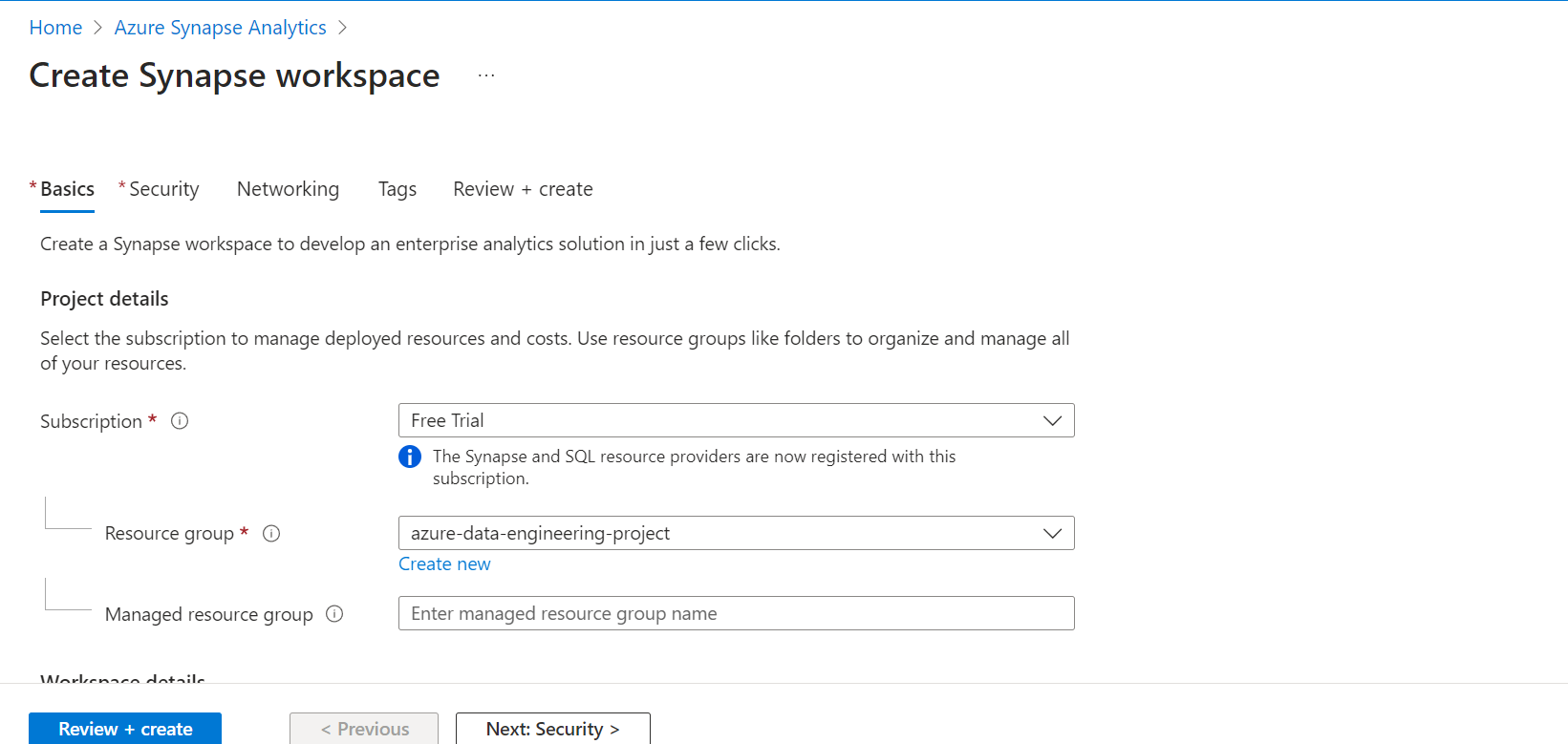
Now we will be doing the data ingestion using the Azure Synapse resource.

Data Ingestion using Azure Synapse.

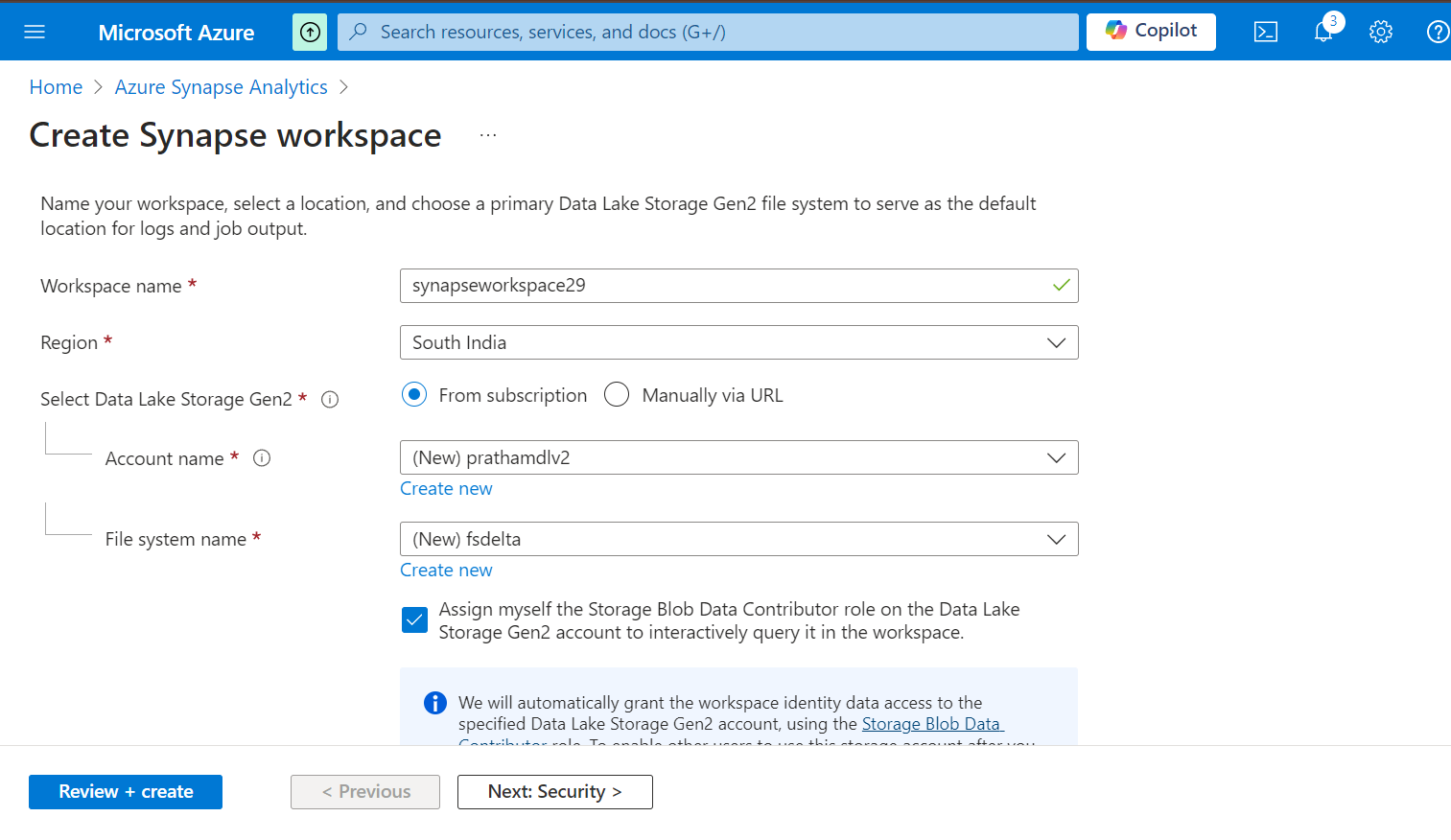
Search for Azure Synapse Analytics and create a resource.



Enter the details

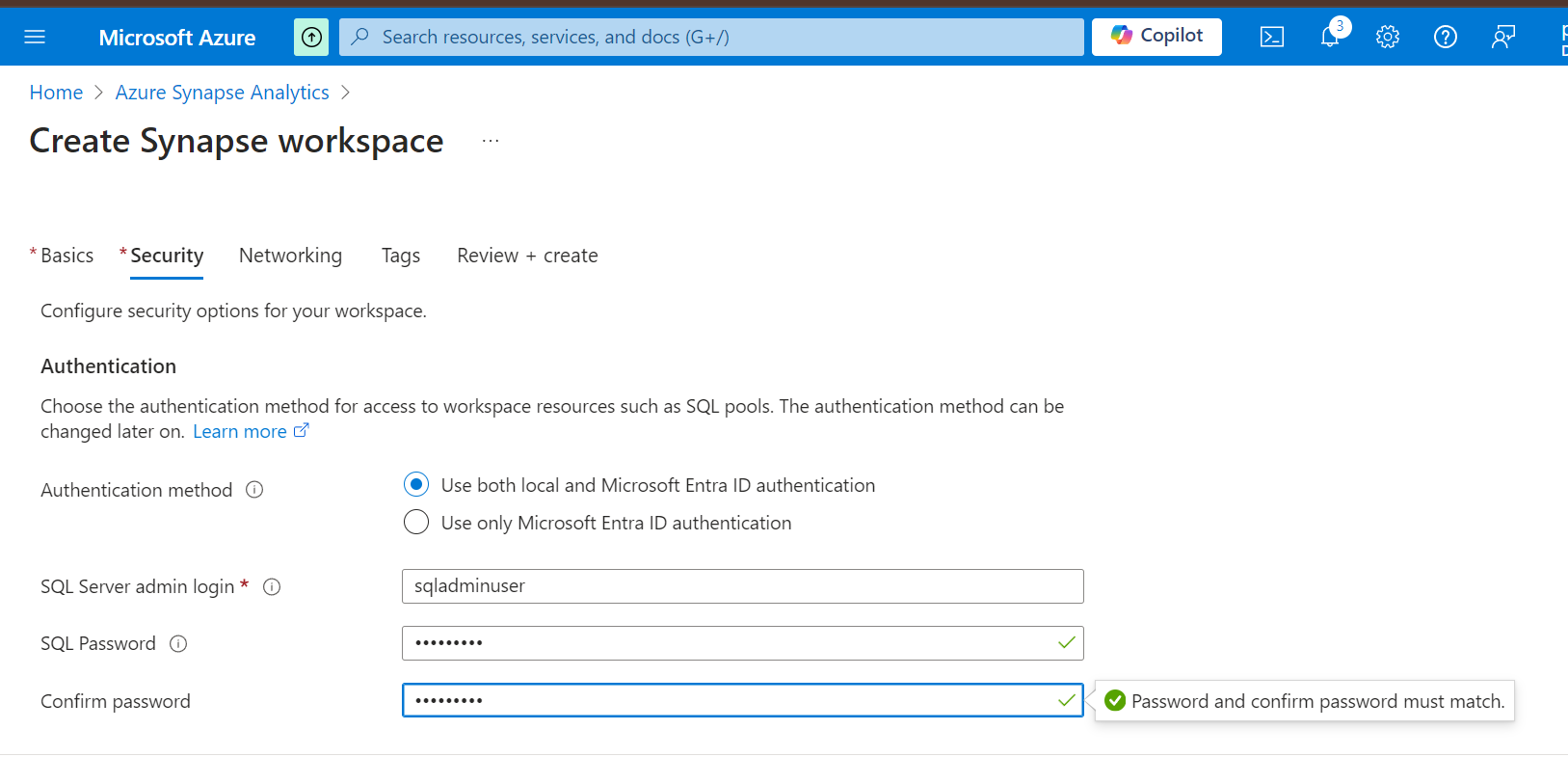


Create a new account name and file system name:



In securities create a sql admin login and provide it a password  
admin login: sqladminuser

Password : @azure123



Create the workspace and this can take some time.

If using South India region throws error while creating the workspace, use East Australia and it should work.