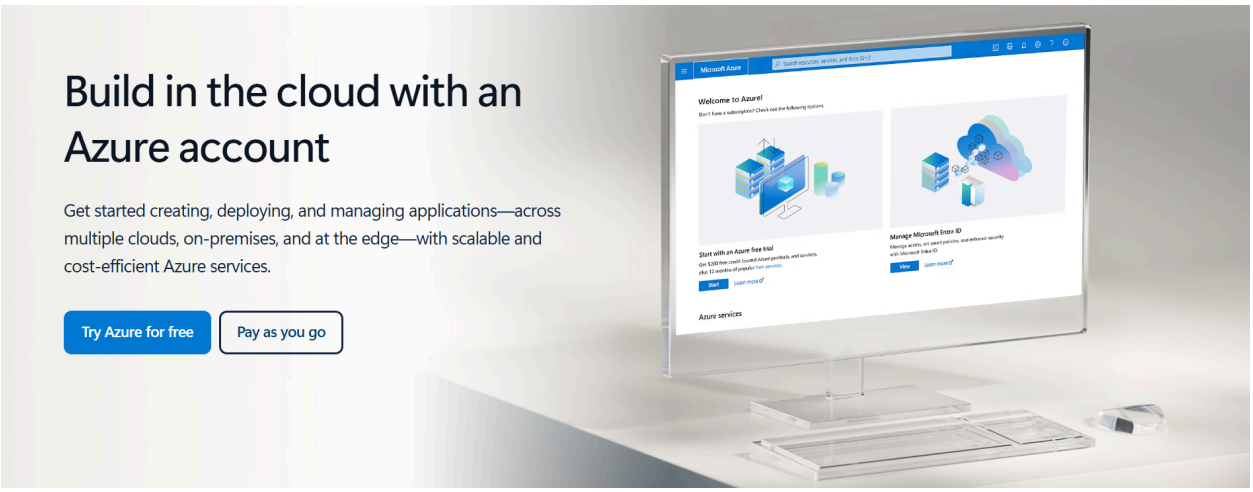
Project overview:
2 sources of data are chosen here. One is an HTTP (saved on GitHub and getting the same via HTTPS requests). But in industry, we will be connecting to a sophisticated data source so that the data won't be lying on GitHub or anywhere; it will primarily be in a database. Second is the SQL table source.

So, once we will be have the data in different sources, we will be ingesting it into Azure Data Factory and we will be moving it into a Data Lake solution. **Azure Data Factory** is a tool which help us to extract data from multiple sources and then we can do some minor transformations as well. Once, we have the data, we will be sending the raw data to a datalake ADLS (**Azure Data Lake Storage**).

Databricks is something which we will be using to transforming the data before sending to ADLS. Now, this transformed data will be reading via Azure Synapse.

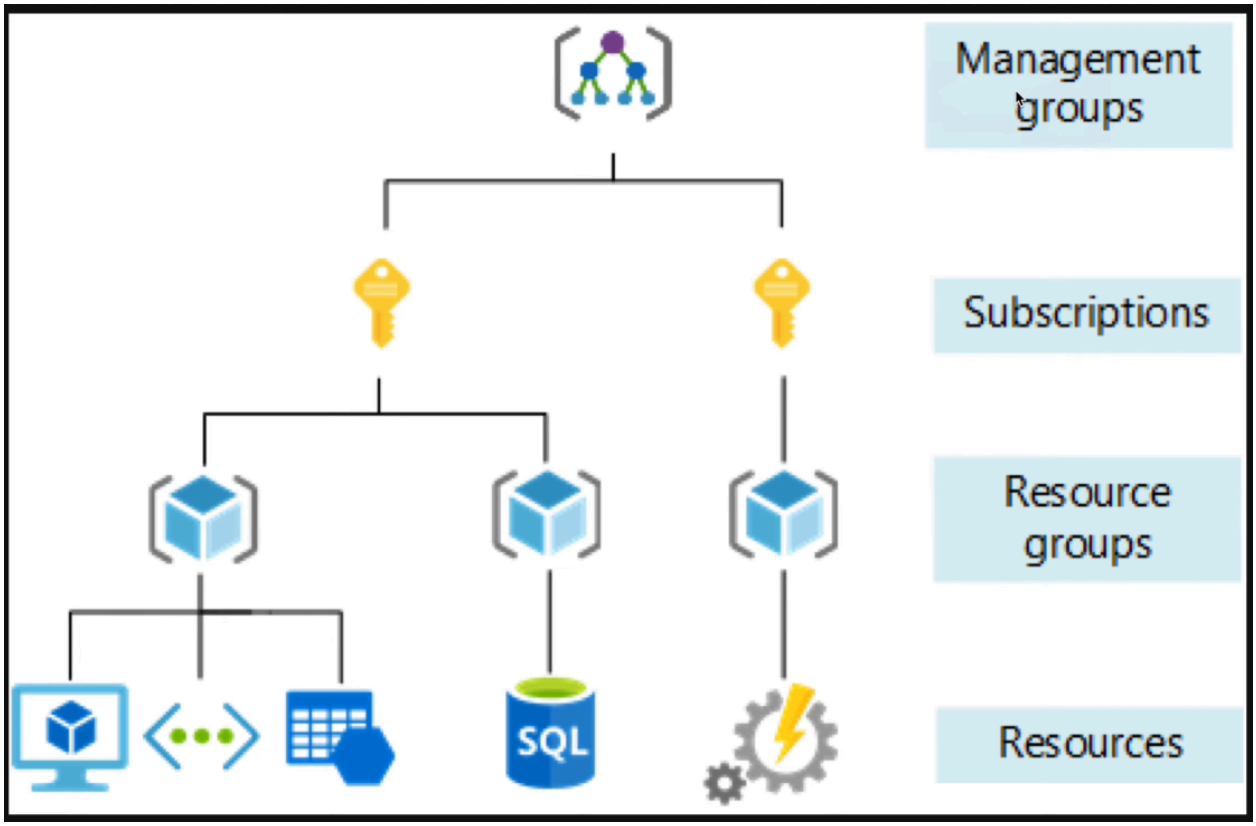
Medallion Architechture is used here which is a data lakehouse design pattern that organizes data into three increasingly refined layers—**Bronze**, **Silver**, and **Gold**—to improve data quality and consistency for analytics and business intelligence. Raw data lands in the Bronze layer, is cleaned and validated in the Silver layer, and finally transformed into enriched, aggregated data ready for reporting and machine learning in the Gold layer. We will be also reading the data from a NoSQL database inside the Azure Databricks. This is just to make sure the complexity of project increases.



Why Cloud?

So, MNCs like Amazon, GCP, Microsoft have their own data centres. The basic funda is that they will give us cloud service and they are going to charge us for it.

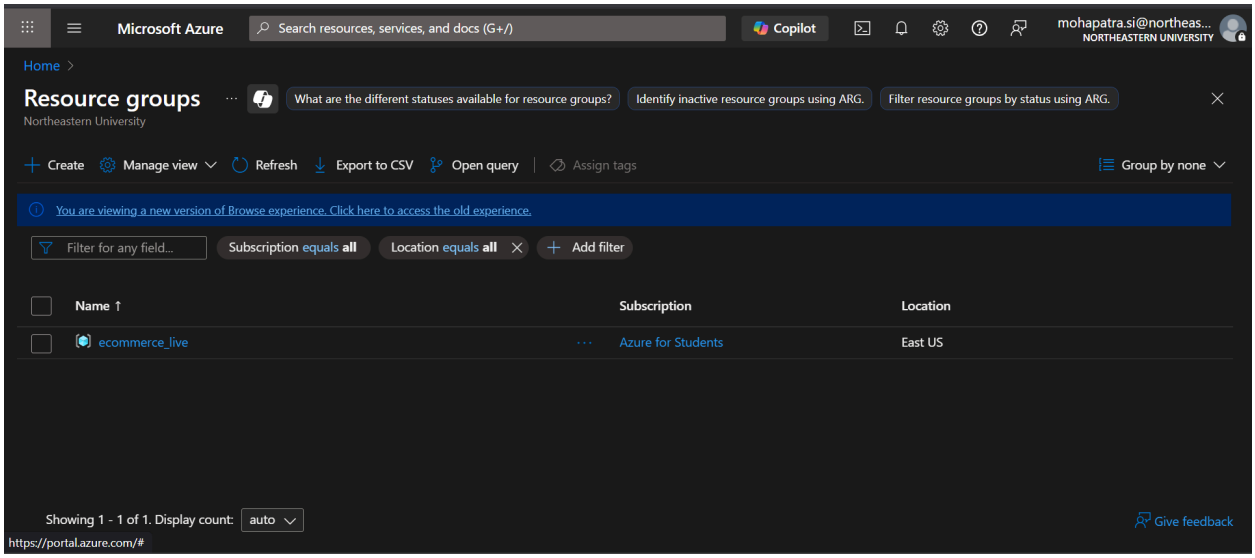
Azure account structure:



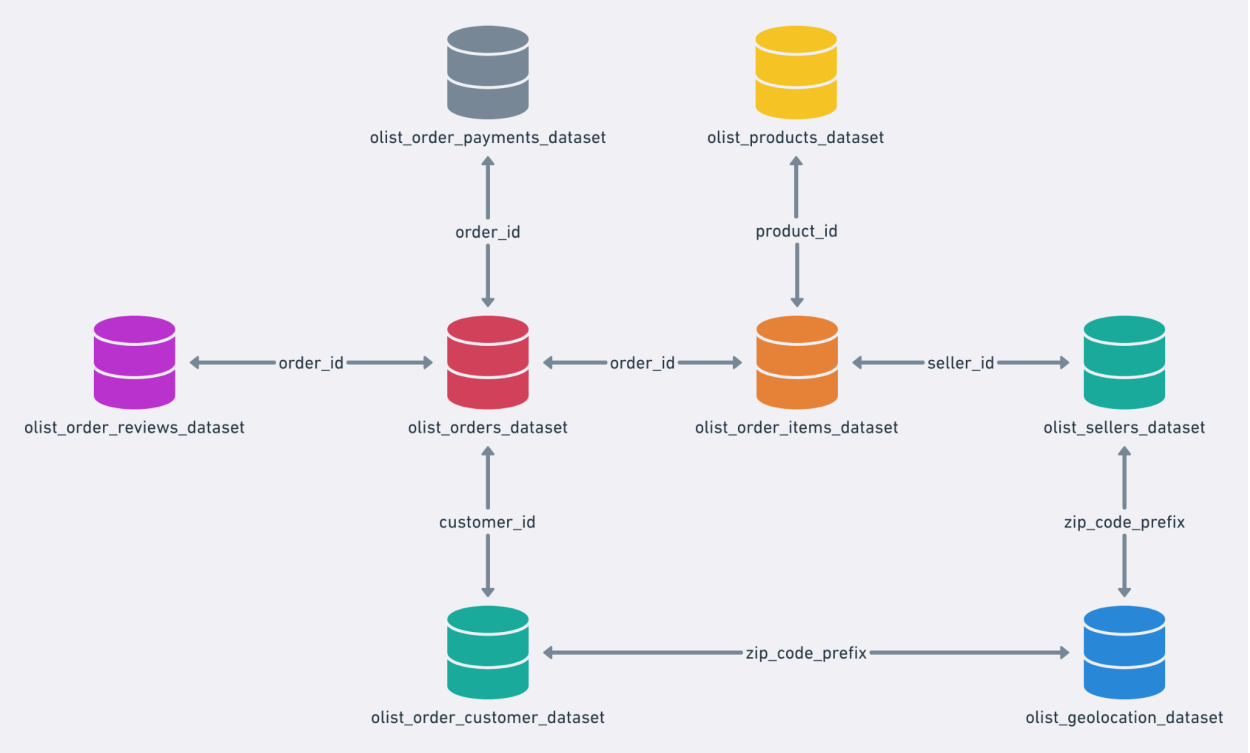
Management groups are like Principal of a school and **Subscriptions** can be like for different classes. And, classes has some **Resource Groups**. And, under it, we have **Resources**.

Steps:

1. Created a new subscription “Azure for Students”
2. Under it, created a new “Resource group”
3. Nearer the Azure Data Centre region, better will be the ping and project will work in a more better way
4. Resource group is created

5. 

6. Here, I am referring Azure Data Factory, Azure Data Lake, Azure Databricks, Azure Synapse Analytics as Resources: ecommerce_live
7. **Olist Dataset Schema:**



The reason of choosing this dataset is somewhat analogous to the user and have some complexities which is somehow similar to the stuff that happens in the industry or real life. Dataset link: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce

https://filess.io/

What is files.io?

Filess.io: A Free & Scalable Database Hosting Solution for Developers! 100% Free Database Hosting Forever. DBaaS – We offer plan-based database services for developers. Free Free plan Forever Availability 99 % Availability

While working on a data pipeline project, I needed a reliable database host for data ingestion. That's when I discovered Filess.io, an excellent platform offering free forever database hosting with support for MySQL, MariaDB, PostgreSQL, and MongoDB!

💡 Why Choose Filess.io?

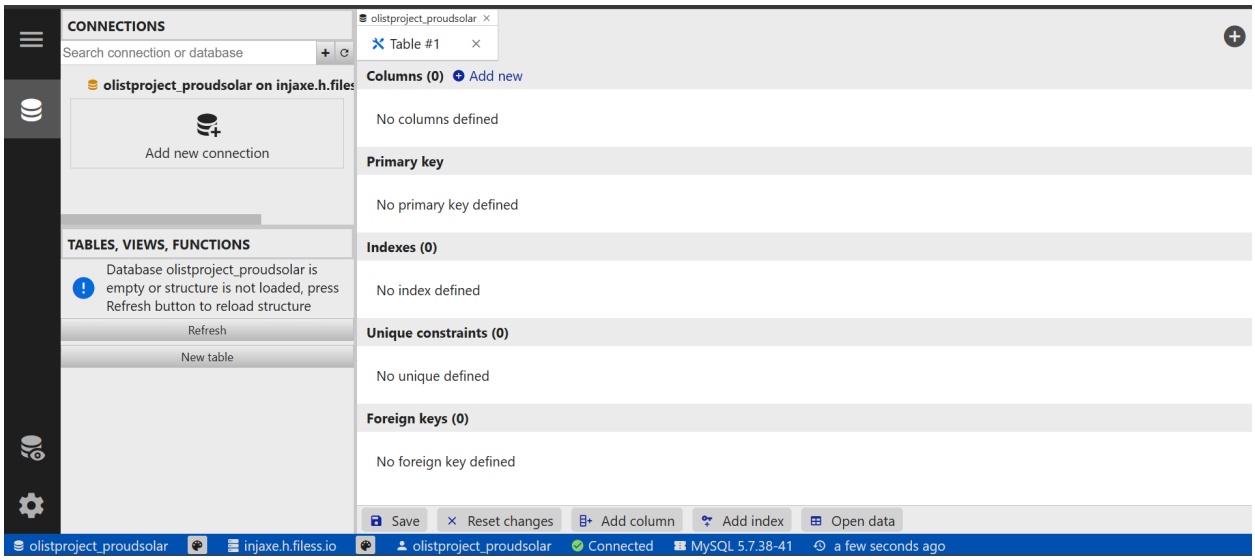
✅ Free Forever Plan – Get 10MB per database, support for up to 2 databases, and weekly backups.

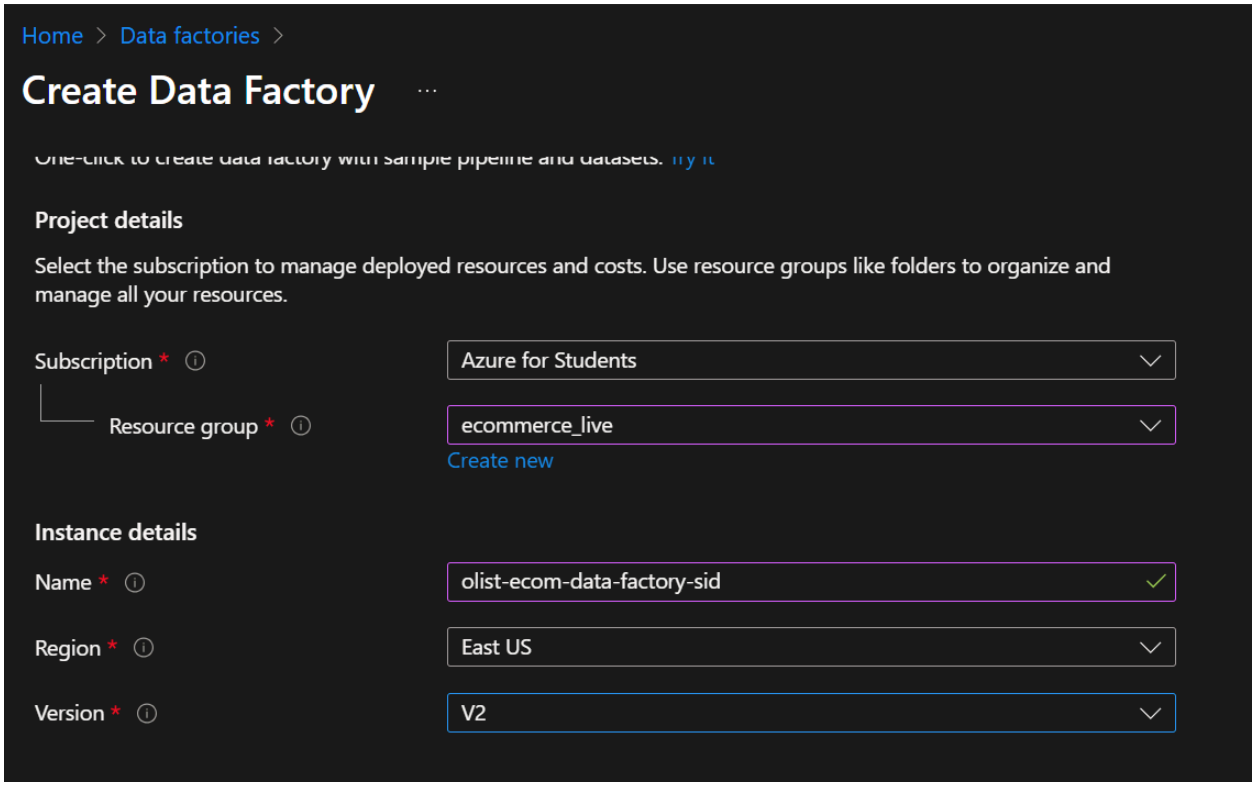
✅ Global Infrastructure – Deploy databases in Europe, Asia, and America for faster access.

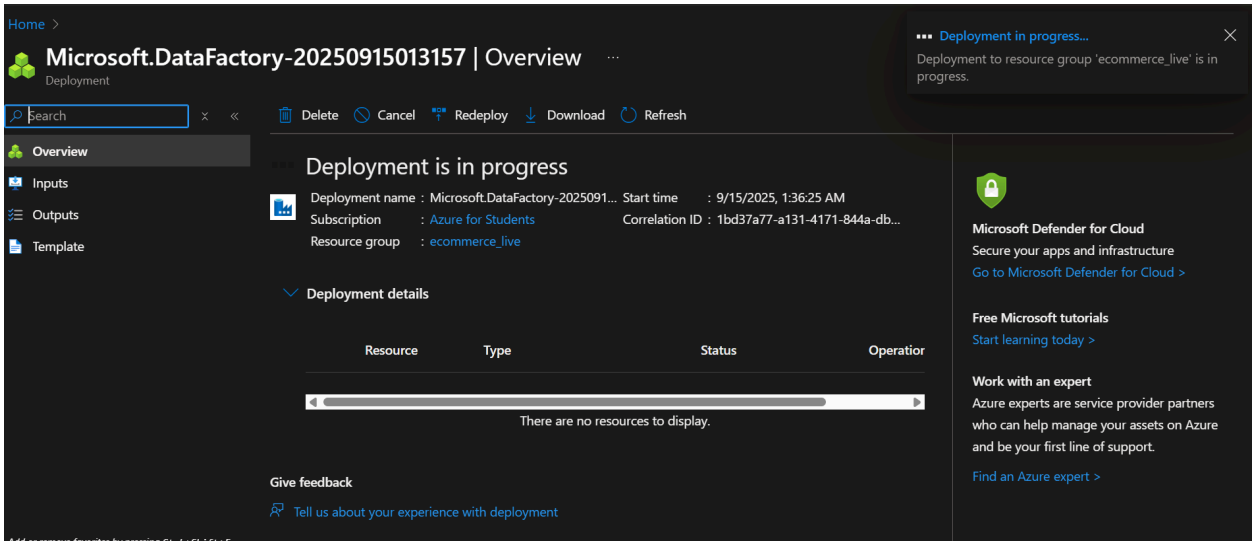
✅ Scalable & Flexible – Upgrade seamlessly with more storage, frequent backups, and security enhancements.

✅ Developer-Friendly – Simple UI for easy setup and management.

Here, we have created a SQL database in which we have to send the data.



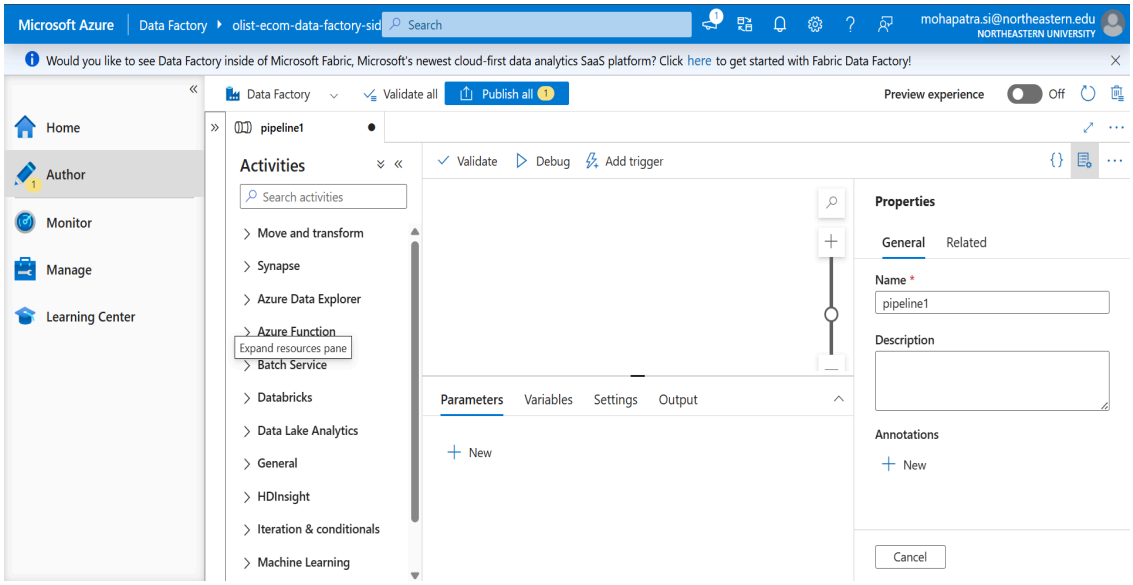
8. 

9. 

Azure Data Factory is a cloud tool that collects data from different sources, organizes it and moves it where we need it: Data Collection (Ingestion), Data Transformation (Cleaning and organizing), and Data movement. (**Why do we use data movement?** Because it helps us to build perfect pipelines.

Click on Azure Data Factory (which we created) -> Launch Data Studio -> Go to Author -> Click on '+' icon -> Create a pipeline

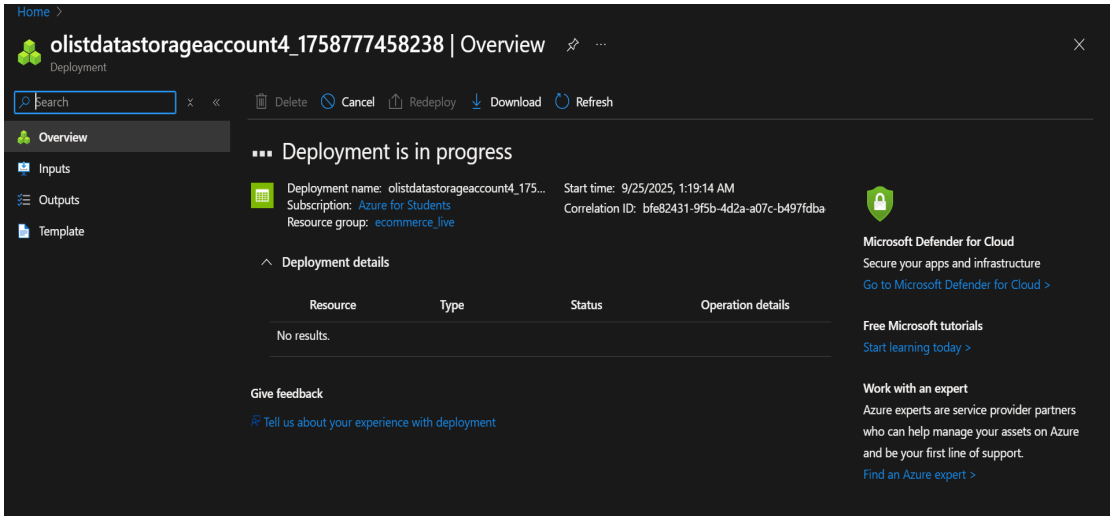
It will look something like this:



Azure Data Factory is like a data delivery and transformation service. It helps us move data from one place to another and prepare it for use, often combining multiple sources along the way. We can think of it as a factory assembly line for data.

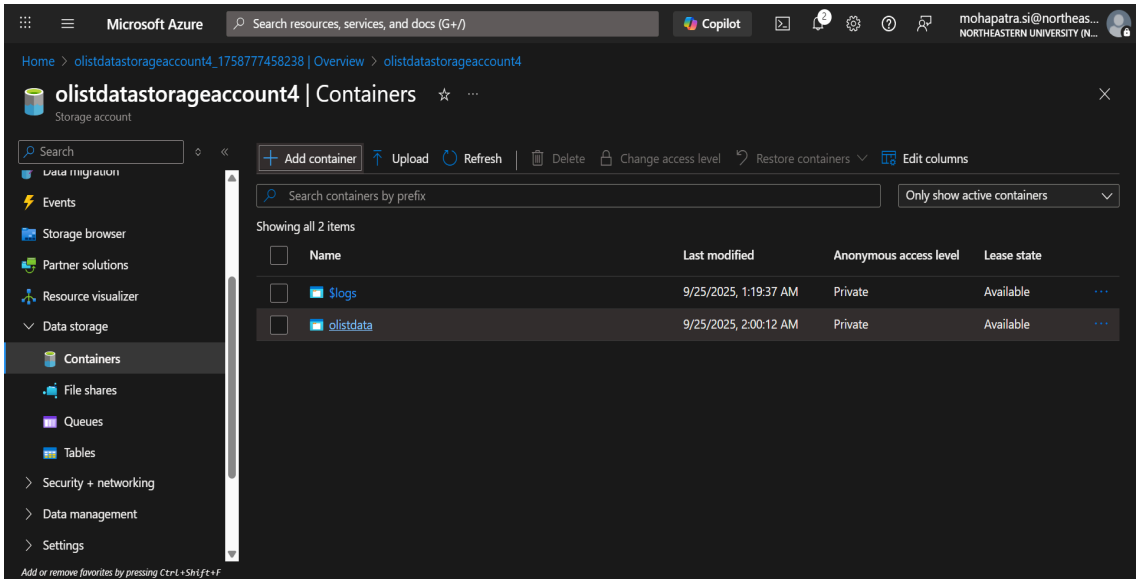
Azure Data Factory is a managed service that automates the movement, transformation, and integration of data, much like a factory assembly line prepares raw materials into a finished product.

While creating a storage account, there are few things to keep in mind and consider. There is something called **Hierarchical Namespace,** which we have to manually enable it. Just because ADF is a very big data lake where we can store any kind of data, be it structured or unstructured. But **Microsoft** uses a **Blob** format. Whereas **Amazon** uses **S3** format and Hadoop uses **HDFS** format. So, basically,y if it is disabled, it means we cannot create folders inside the container, which means that the container will be of no use. It's just like we have a new brand laptop, but we are unable to create a folder. Obviously, nobody would like to buy that laptop.

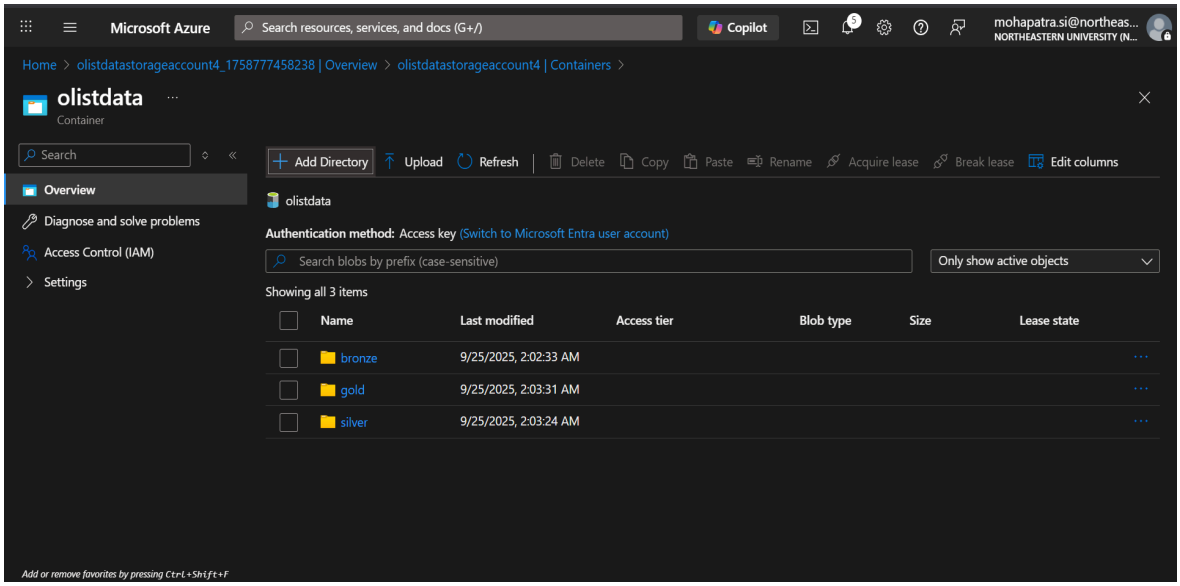


Azure Data Lake Storage (ADLS) Gen 2: It's a secure cloud platform that provides scalable, cost-effective storage for big data analytics. It stores a massive amount of data, keeps data organized, fast and efficient as well. And, it is also built on **Azure Blob Storage**. We can store files, logs, images, IoT data, or structured/unstructured data. Think of it like **Google Drive** for huge amounts of data, but made for businesses that deal with terabytes (or even petabytes) of information. Azure uses (Advanced Encryption Standard) AES-256 encryption, one of the strongest security standards used by banks and governments.

10. **Creating a container in ADLS:**



I created a directory inside olistdata storage named bronze followed by silver followed by gold.



Here, basically I am trying to follow the Medallion architecture which is followed in most of the industries.

So, instead of doing manually, if we have 100s or 1000s of csv data in github, we can put the base url as https://raw.githubusercontent.com/SIDDHARTH107/
And relative url as
Real-Time-E-Commerce-Data-Processing-and-Analytics-using-Medallion-architecture-on-Azure-Cloud/refs/heads/main/olist_customers_dataset.csv

This relative url is something that we can configure again and again when we have tons of csv data in github.

11. Now after Data Ingestion from source to ADF, I worked on Azure Data Lake Studio Gen 2 to transfer Raw Data from ADF to ADLS Gen 2. Sometimes, there will be a question why are we not transferring data directly from Github to ADLS Gen 2? Why **ADF** is used in between?(**Azure Data Factory** is like a cloud-based “data logistics & control center” that securely connects, moves, and orchestrates data from many sources into your storage or analytics systems—on time, at scale, with monitoring built-in.)

Key Benefits of ADF

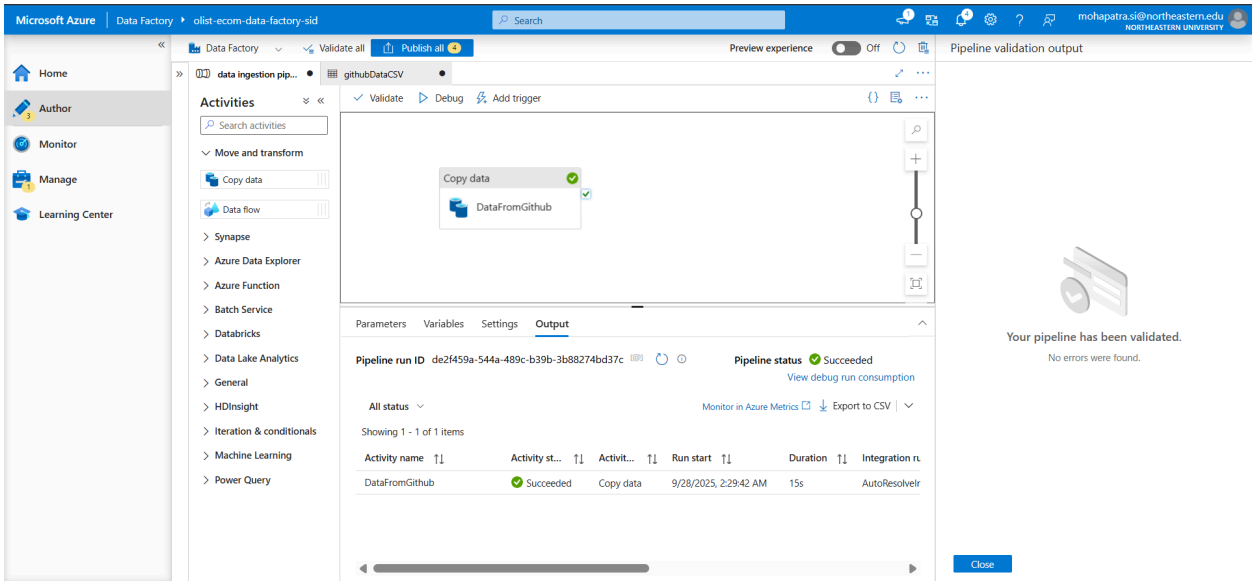
Feature	Why It's Important
Many Connectors	Easily connect to SQL, APIs, SaaS, on-prem
Scheduling	Automate runs (hourly, daily, etc.)
Monitoring	Dashboard for pipeline status, alerts, logs
Security	Secure credentials, managed identities, encryption
Scalability	Serverless – no infra to manage
Integration	Works smoothly with ADLS, Synapse, Databricks

ADF isn't just "copying" — it's "coordinating, scheduling, securing, and monitoring" our data pipelines at scale. Without it, we would be writing and maintaining a lot of custom scripts. **ADF** is basically the **ETL** engine at the front door.

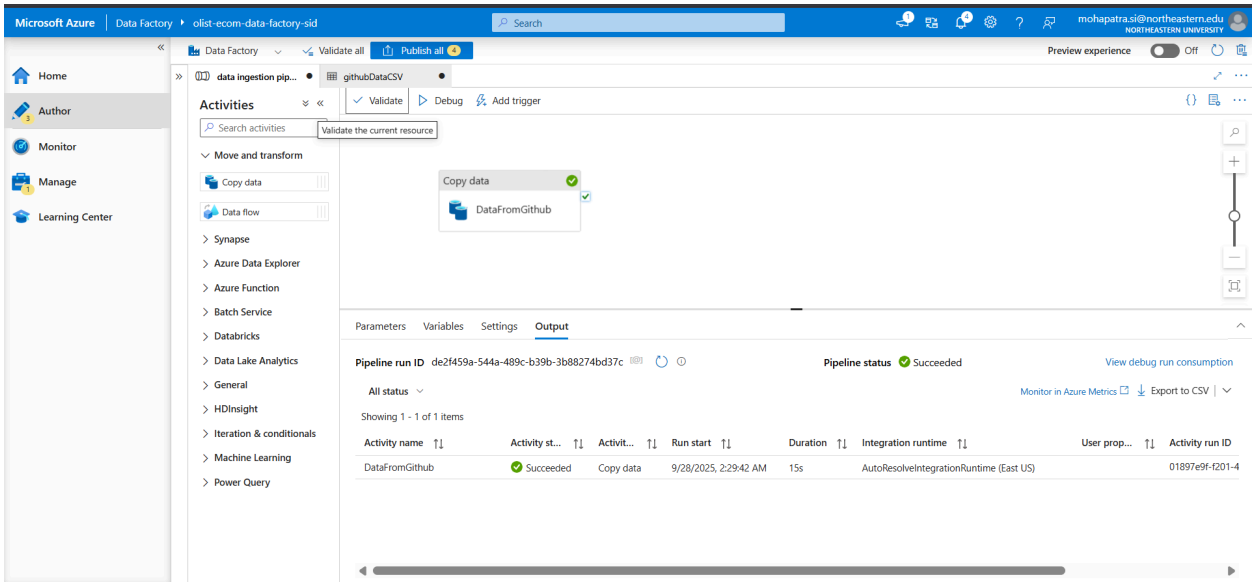
What Azure Data Factory Does That Direct Copy Doesn't

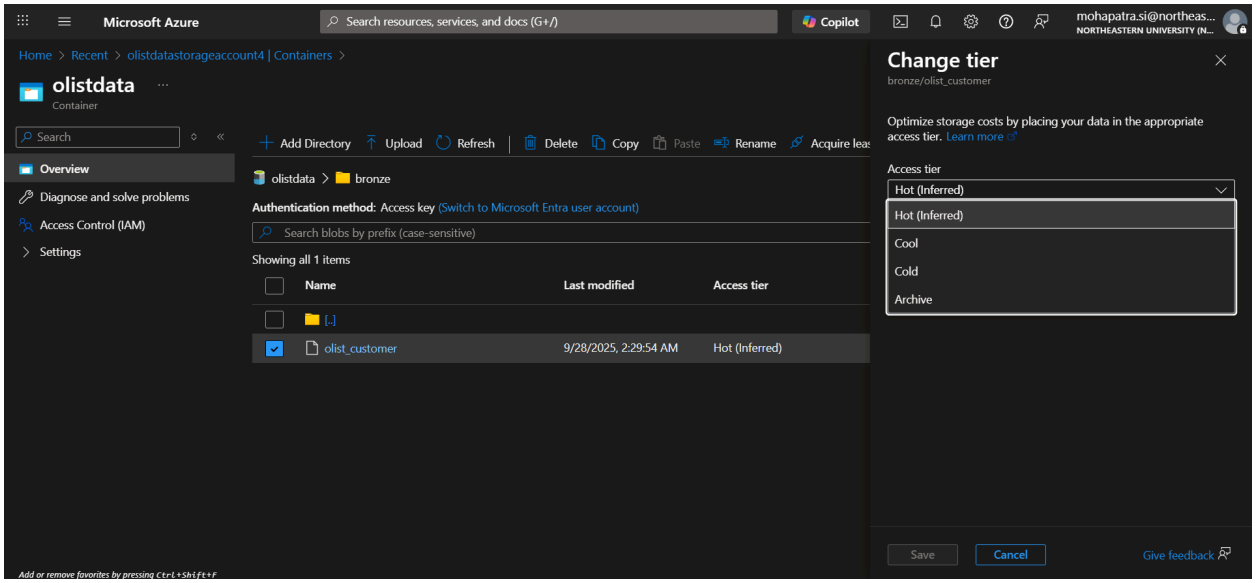
1. **Orchestration / Scheduling**
   - ADF lets you **schedule** when and how data moves.
   - Ex: Pull GitHub data at 2 a.m. daily, load new SQL table snapshots every hour.
   - Without ADF, you'd have to write and maintain your own code/scripts to do this.
2. **Data from Multiple Sources**
   - ADF can ingest from dozens of sources at once — APIs, FTP, on-prem SQL, SaaS apps.
   - You don't have to manually connect each one to ADLS Gen2.
3. **Pre-processing (Light Transformations)**
   - ADF can filter, map columns, join simple files before landing into ADLS Gen2.
   - Example: only pull rows from GitHub CSV where "status=active" before storage.
4. **Monitoring & Error Handling**
   - ADF has built-in dashboards, retry logic, and alerts.
   - Without it, if a job fails, you'd only know after the fact (or not at all).
5. **Security & Governance**
   - Secure credential management, data lineage, and compliance baked in.

12.



Now, we can see, the pipeline has been created connecting from Azure Data Factory to Azure Data Lake Studio Gen 2. And, just because both service belongs to Microsoft Azure, the connection took few time and was done seamlessly. While building the pipeline, I preferred "Validate" and "Validate copy runtime" and after clicking "Debug", it shows like this:

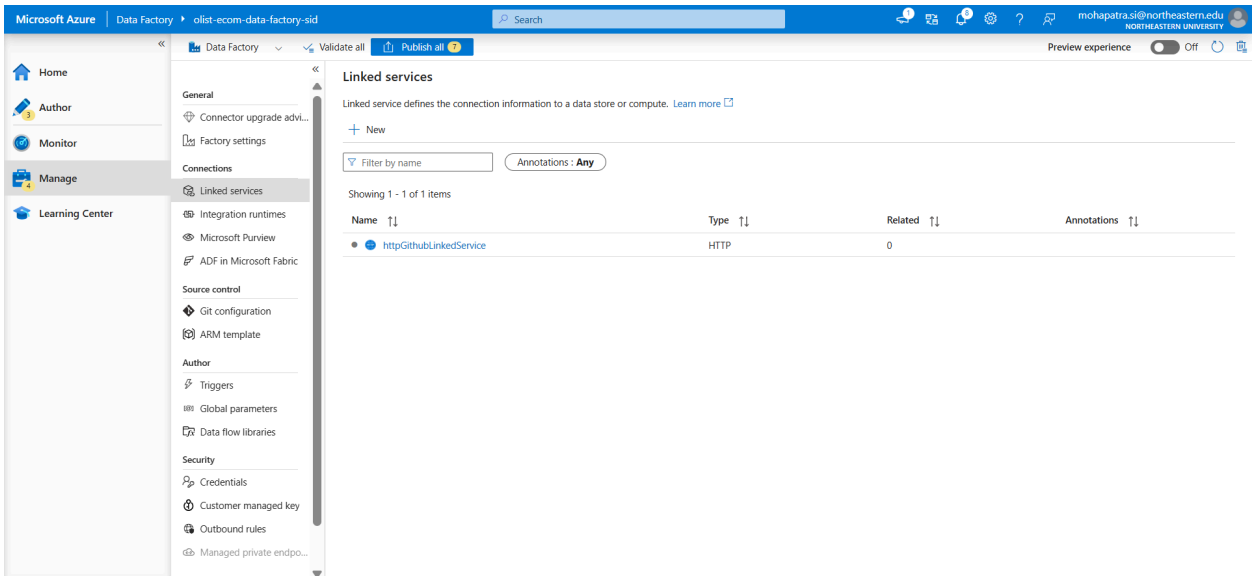




Tier	What it is (plain English)	How fast can you get data	Cost to store	Cost to read	Best for…
Hot	"Always in use" area	Fast (milliseconds)	Highest	Lowest	Files/data you use or update very often (daily/hourly)

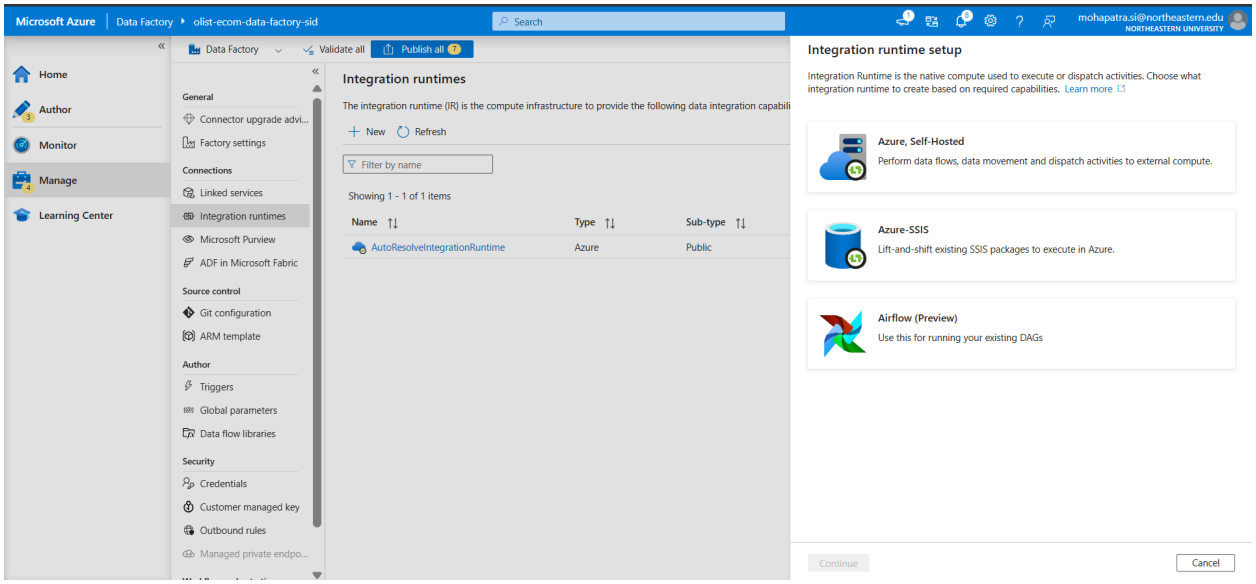
Cool	"Sometimes in use" area	Fast (milliseconds)	Cheaper than Hot	A bit more expensive to read	Files/data you use once or twice a month
Cold	"Rarely in use" area (newer)	Fast (milliseconds) but higher transaction cost	Even cheaper than Cool	Even more expensive to read	Files/data you might check only a few times a year
Archive	"Deep storage / vault" area	Slow (can take hours to rehydrate before you can read)	Cheapest	Highest retrieval cost	Old data you must keep but almost never use (e.g., legal or compliance records)

13. Now, I will create a **link** service so that I don't keep on entering this everytime. And, this should be done **parametrically**, (which means we will provide the base url as base url is same for all the datasets in github and the relative url will be provide as a parameter which changes from dataset to dataset) so that there are 1000s of csv file, we can do it smartly.

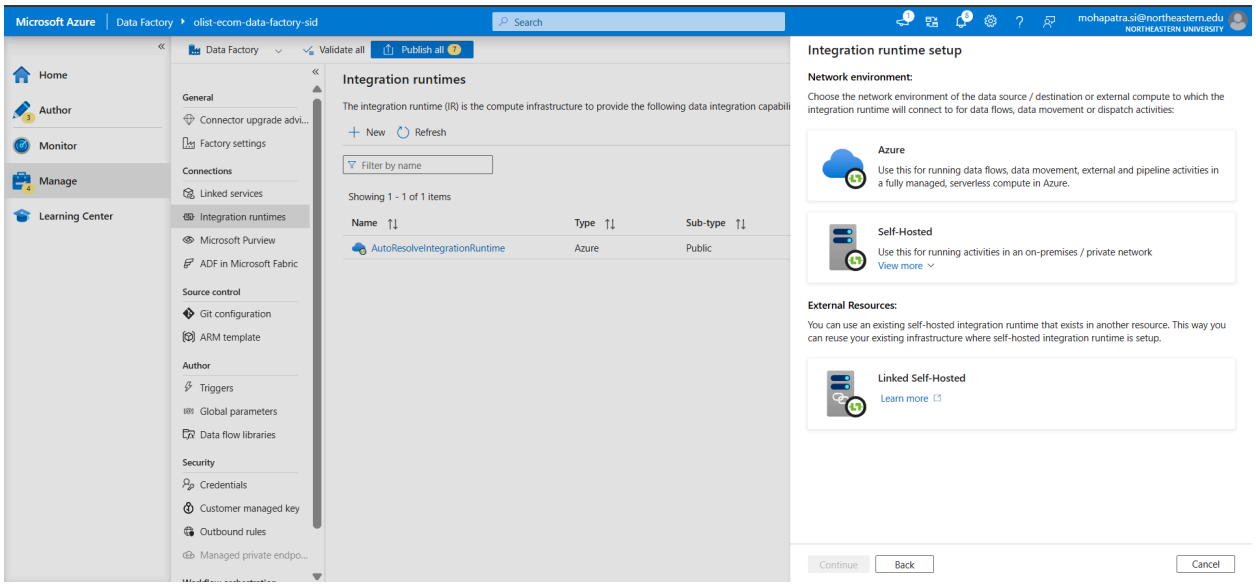


(Above figure: We can see, I have added 1 linked service i.e. github as a data source)

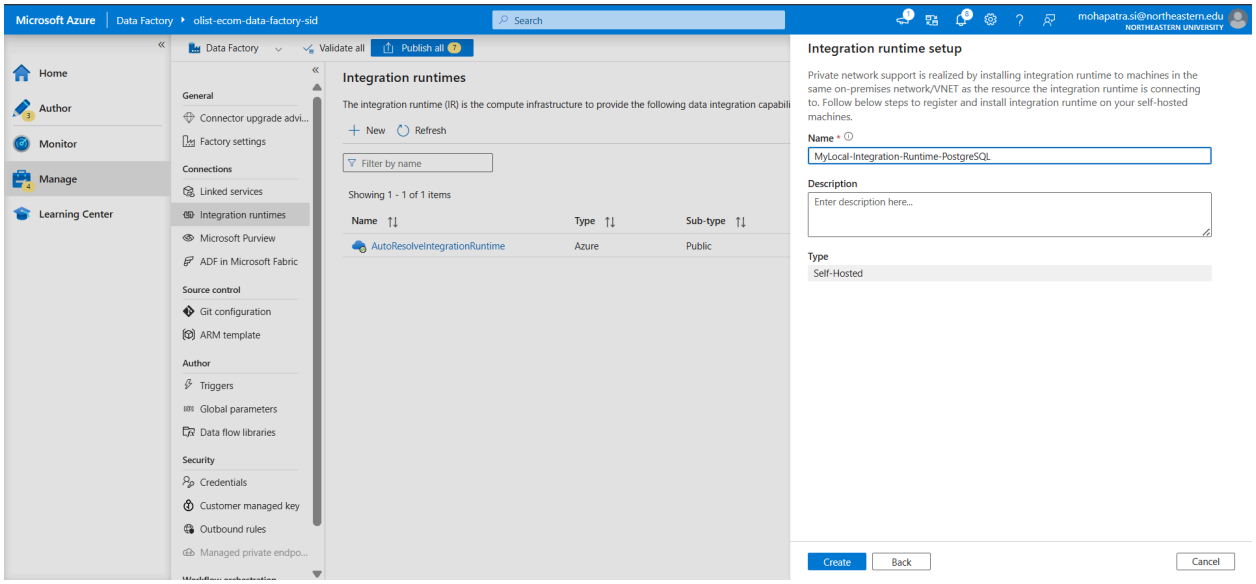
Now, in order to connect my local postgres with cloud based Microsoft server which is Azure Data Factory, I need **Self Hosted Integration Runtime**:



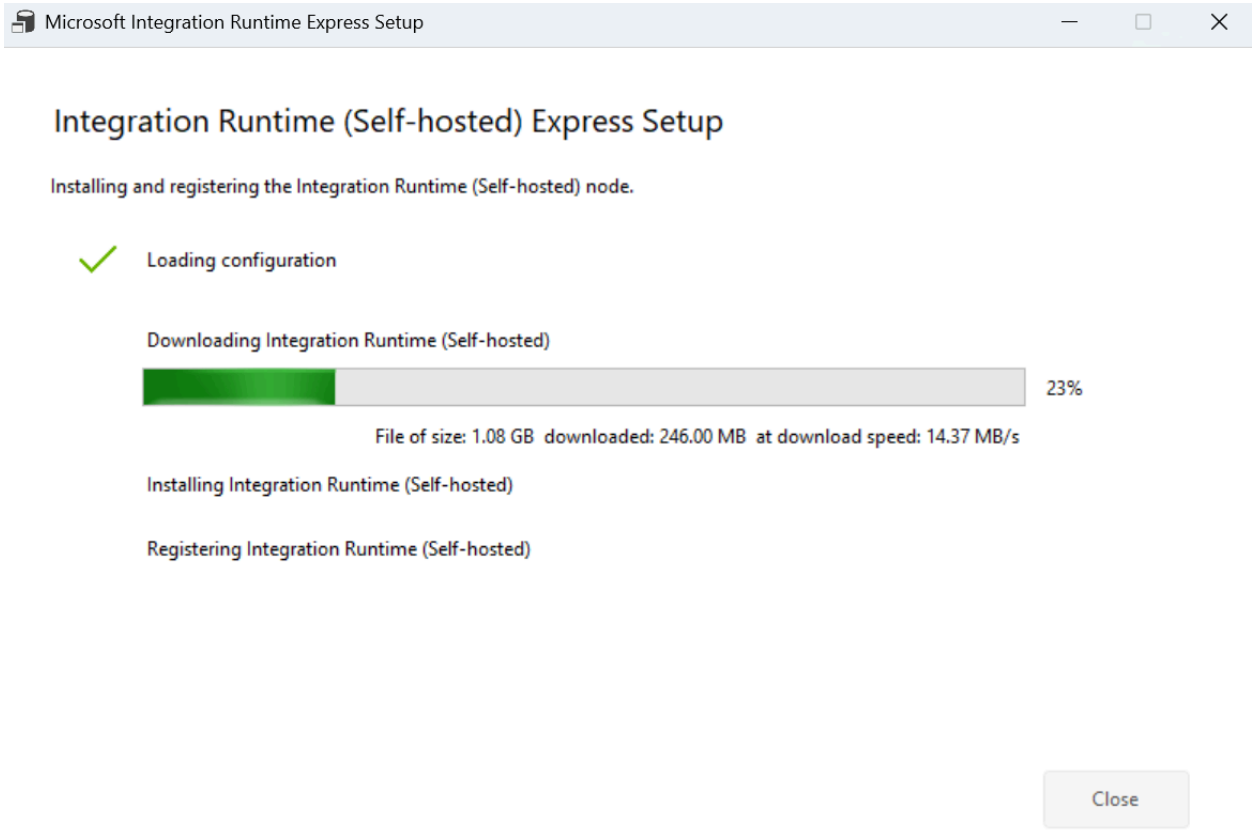
Here, after clicking **Azure Self-Hosted,** I got this:

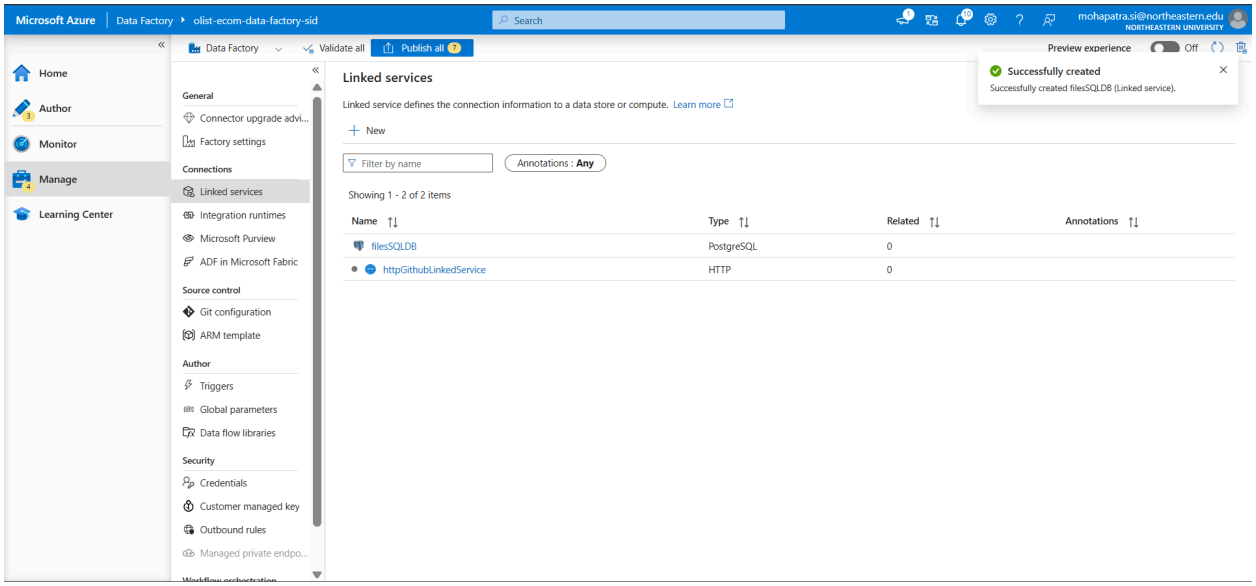


Then **Self-Hosted** is clicked,



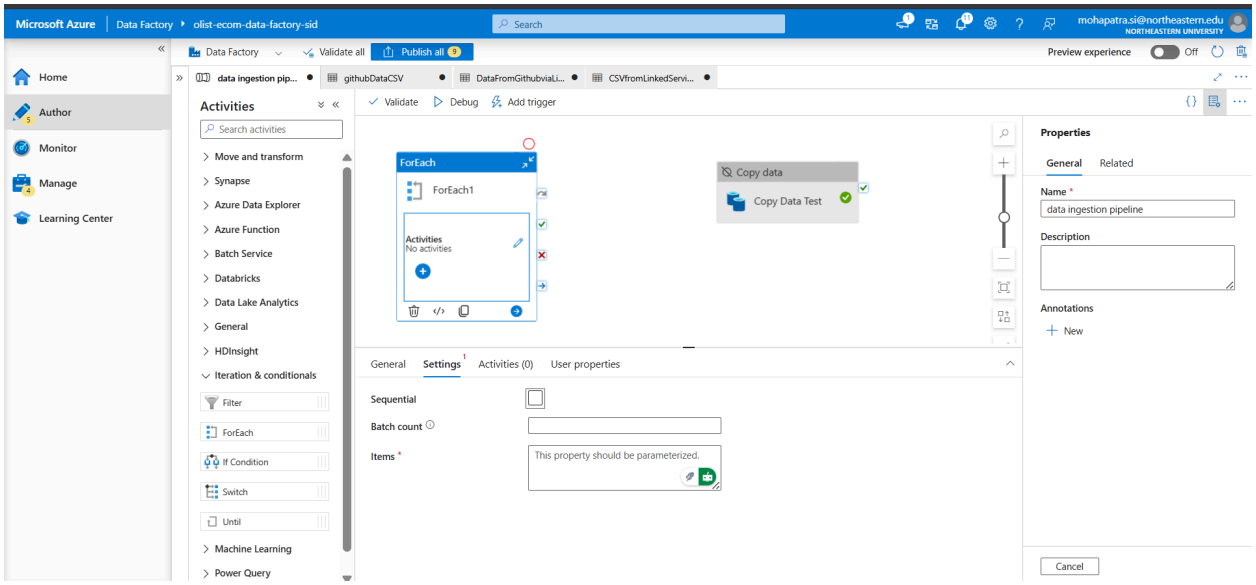
Then, I clicked **Express Setup** to install the Integration Runtime.





Now, we can see my **PostgreSQL** is connected to my **Azure Data Factory**.

We can see there are options like sequential and batch count:



The **Sequential** checkbox in a ForEach activity dictates how the loop runs. When we check this box, the ForEach activity will process each item in our input list one at a time, in order. The second iteration will not start until the first one has completely finished.

By default, if we leave this box unchecked, the ForEach activity runs in parallel, which is what "in batches" refers to. It processes multiple items simultaneously to speed up the total execution time.

Sequential Processing
Basically, it is used for control and safety. When we check the Sequential box, we are forcing the loop to operate in a strict, single-file line.

When to Use It?
Order is Critical: Need to use this when the outcome of one iteration depends on the completion of the previous one. For example, if we are processing daily sales files and day_2.csv can only be processed after day_1.csv is successfully loaded.

Avoiding Race Conditions: When multiple iterations writing to the same file or database table could corrupt the data if they run at the same time. Sequential processing ensures that only one operation is happening at any given moment.

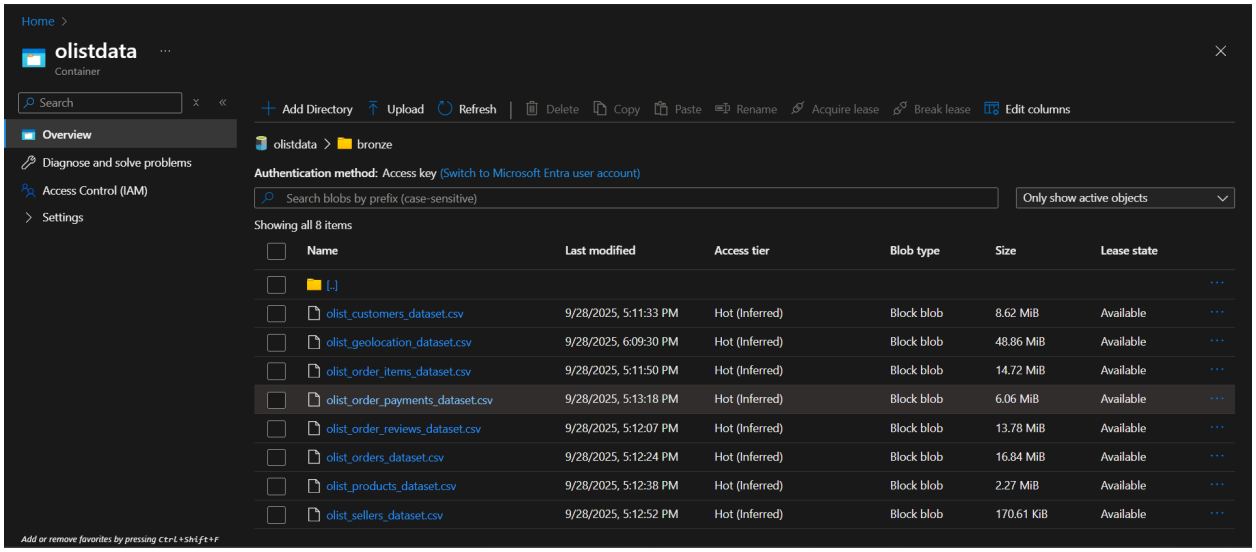
Batches (Parallel Processing)
Basically, it is used for speed and efficiency. When the order doesn't matter, parallel processing leverages the power of the cloud to get the job done much faster. When we uncheck the Sequential box, we are allowing the loop to run on a multi-lane highway. By default, it can run up to 20 iterations at once. We can control this by setting a Batch count (e.g., setting it to 5 would mean it runs 5 iterations at a time).

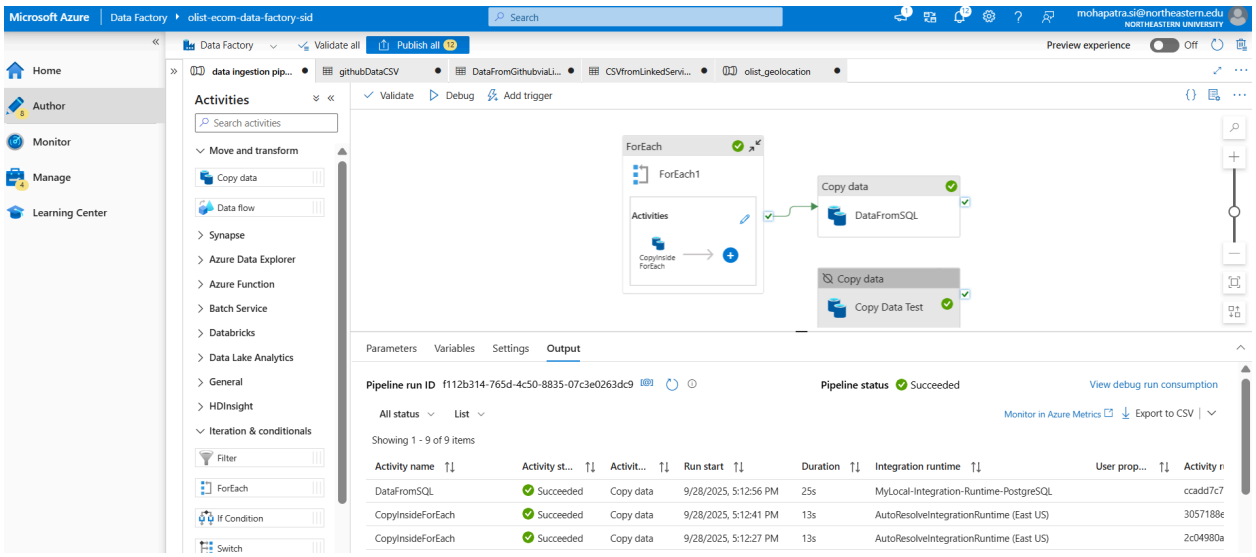
When to Use It:
Maximum Performance: This is the default and most common use case. Use it when your goal is to complete the entire job as fast as possible.

Independent Tasks: It's perfect when the items in your list have no dependency on each other. For example, copying 1,000 independent product images from one storage account to another. Copying product_A.jpg has no impact on the process for product_B.jpg.

Bulk Data Ingestion: When you need to copy hundreds of separate tables from a source database. Each table copy is an independent task, and running them in parallel drastically reduces the total pipeline run time.

14. Now we can see, all the data has been ingested successfully to the bronze layer:





RECAP:

Phase 1: Setup & Connection

Established a Hybrid Cloud Connection: You first installed and configured a Self-Hosted Integration Runtime (SHIR) on your local machine. This created a secure bridge to allow Azure to access your on-premises PostgreSQL database without exposing it to the internet.

Created Linked Services: You connected Azure Data Factory to your data stores by creating two key Linked Services: one for your local PostgreSQL database (using the SHIR) and another for your Azure Data Lake Storage (the "olistdata" container).

Phase 2: Building the Automated Pipeline

Created Reusable Data Templates: You designed two generic, parameterized datasets. One for the PostgreSQL source (using a tableName parameter) and one for the CSV sink in the data lake (using a fileName parameter).

Engineered a Dynamic Pipeline: You built a master pipeline (Ingest_All_PostgreSQL_Tables) to automate the entire workflow:

Discovered Tables Automatically: You used a Lookup activity with a SQL query to dynamically fetch a list of all table names from your PostgreSQL database that matched the "olist_%" pattern.

Looped Through Each Table: You added a ForEach activity that takes the list of tables from the Lookup activity as its input, preparing to process each one individually.

Configured the Data Copy: Inside the ForEach loop, you placed a Copy Data activity and used the parameterized datasets to make it generic. You passed the current table name from the loop (@item().table_name) to both the source and sink, dynamically creating the correct file name for each table.

Phase 3: Execution & Finalization
Tested and Validated the Pipeline: You ran the entire pipeline using the Debug button. This executed a live test run, proving that the logic worked by successfully copying all 9 tables and their data to the "bronze" folder in your data lake.

Troubleshot and Cleaned Up: You encountered a publishing error caused by old, incomplete datasets (GithubDataCSV and DelimitedText1). You successfully identified and deleted these unused assets to resolve the validation issue.

Saved the Final Pipeline: You successfully used the "Publish all" button to permanently save the entire pipeline structure—including the linked services, datasets, and pipeline logic—to the Azure Data Factory service.

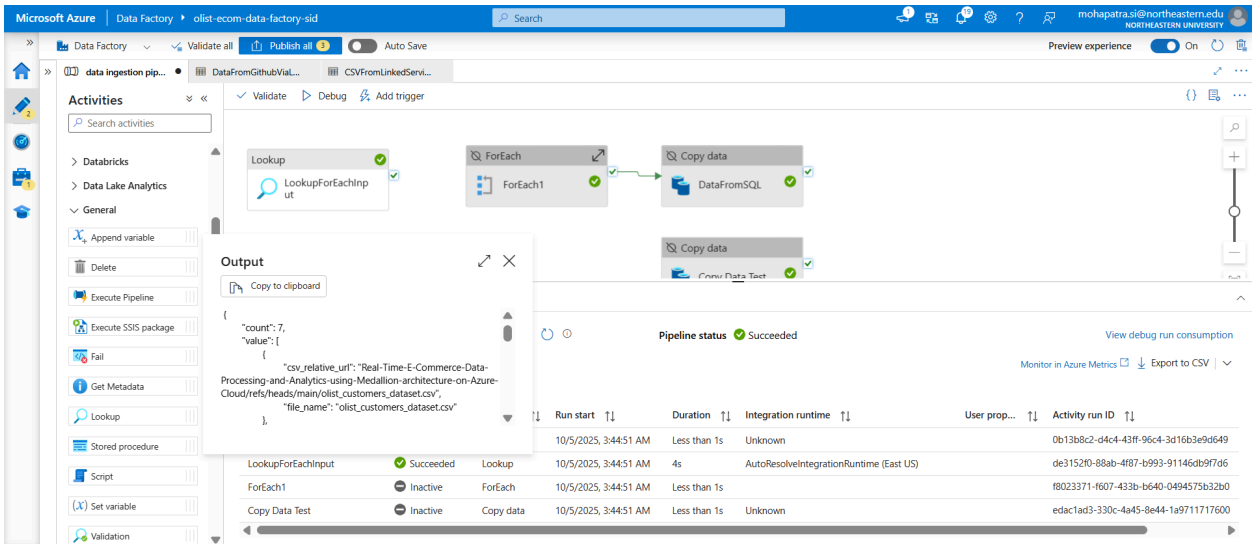
15.
In Azure Data Factory, the Lookup activity is used to retrieve a small amount of data from any data source. Think of it as asking a question and getting a small piece of information back that you can use in the next steps of your pipeline.

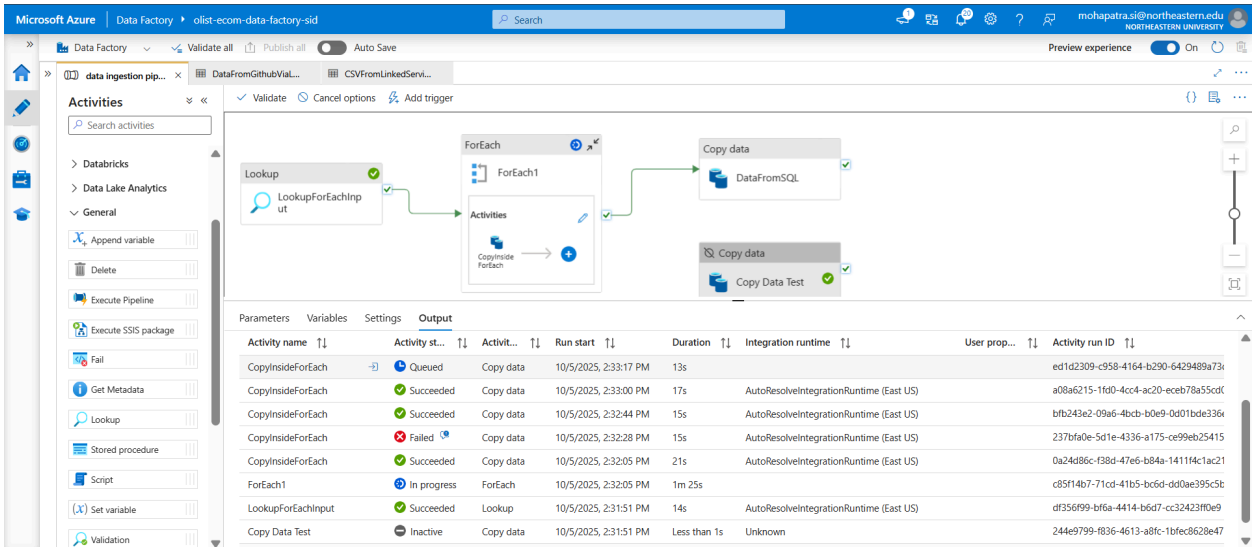
Instead of us manually telling the pipeline which tables or files to copy one by one, the Lookup activity does it for you. Here is the exact process:

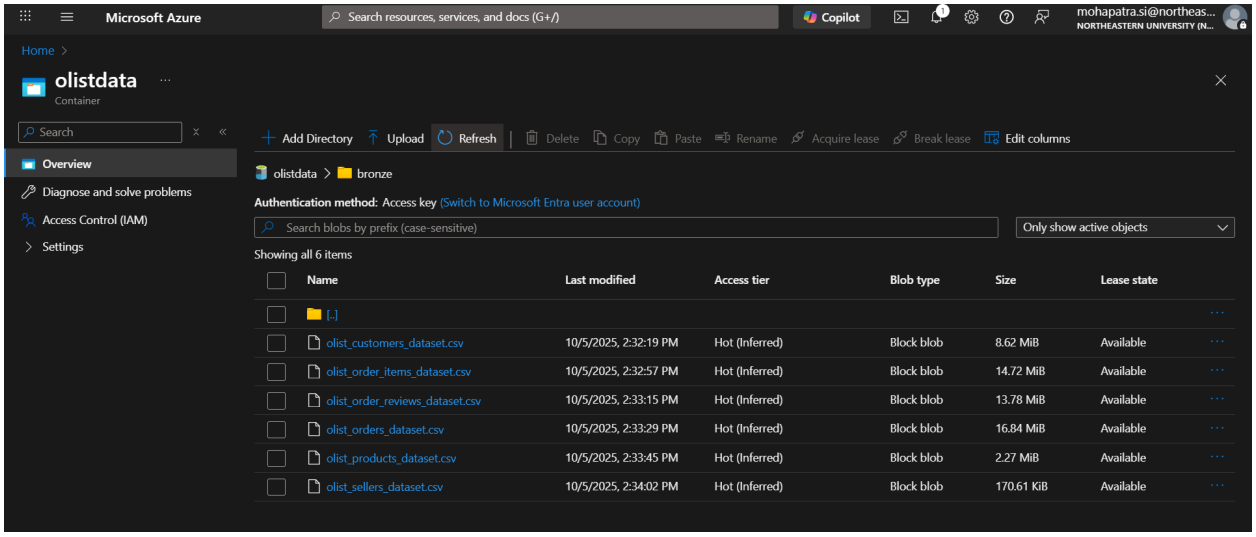
Get a To-Do List: The Lookup activity runs first. It connects to a source (like a database query or a configuration file) to get a list of items that need to be processed. In your case, it's likely getting a list of all the table names (e.g., 'olist_customers_dataset', 'olist_orders_dataset', etc.) from your PostgreSQL database.

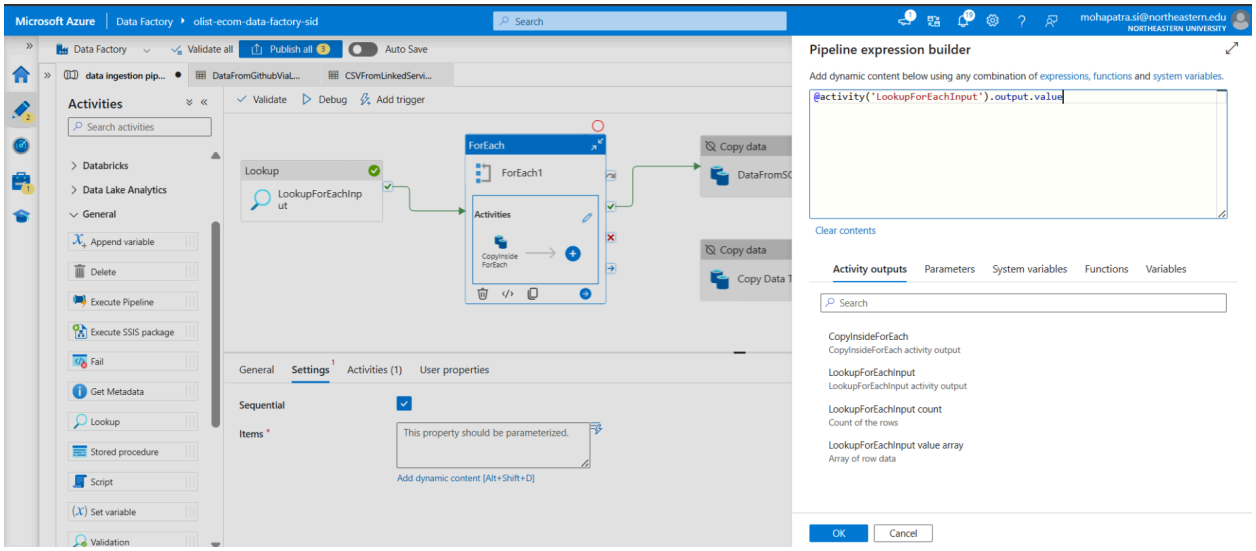
Pass the List to the Next Step: The output of the Lookup activity is a list (an array) of these table names.

Feed the Loop: This list is then passed directly into the ForEach activity. The ForEach loop takes that list and runs the Copy Data activity inside it once for every single item on the list.


Microsoft Azure
Data Factory
olist-ecom-data-factory-sid
Search
mohapatra.si@northeastern.edu
NORTHEASTERN UNIVERSITY
Validate all
Publish all
Auto Save
Preview experience
On
Activities
Search activities
Databricks
Data Lake Analytics
General
Append variable
Delete
Execute Pipeline
Execute SSIS package
Fail
Get Metadata
Lookup
Stored procedure
Script
Set variable
Validation
Validate
Debug
Add trigger
Lookup
LookupForEachInput
ForEach
ForEach1
Copy data
DataFromSQL
Copy data
Output
Copy to clipboard
{
"count": 7,
"value": [
{
"csv_relative_url": "Real-Time-E-Commerce-Data-Processing-and-Analytics-using-Medallion-architecture-on-Azure-Cloud/refs/heads/main/olist_customers_dataset.csv",
"file_name": "olist_customers_dataset.csv"
},
Pipeline status
Succeeded
View debug run consumption
Monitor in Azure Metrics
Export to CSV
Run start
Duration
Integration runtime
User prop...
Activity run ID
10/5/2025, 3:44:51 AM
Less than 1s
Unknown
0b13b8c2-d4c4-43ff-96c4-3d16b3e9d649
LookupForEachInput
Succeeded
Lookup
10/5/2025, 3:44:51 AM
4s
AutoResolveIntegrationRuntime (East US)
de3152f0-88ab-4f87-b993-91146db9f7d6
ForEach1
Inactive
ForEach
10/5/2025, 3:44:51 AM
Less than 1s
f8023371-f607-433b-b640-0494575b32b0
Copy Data Test
Inactive
Copy data
10/5/2025, 3:44:51 AM
Less than 1s
Unknown
edac1ad3-330c-4a45-8e44-1a9711717600

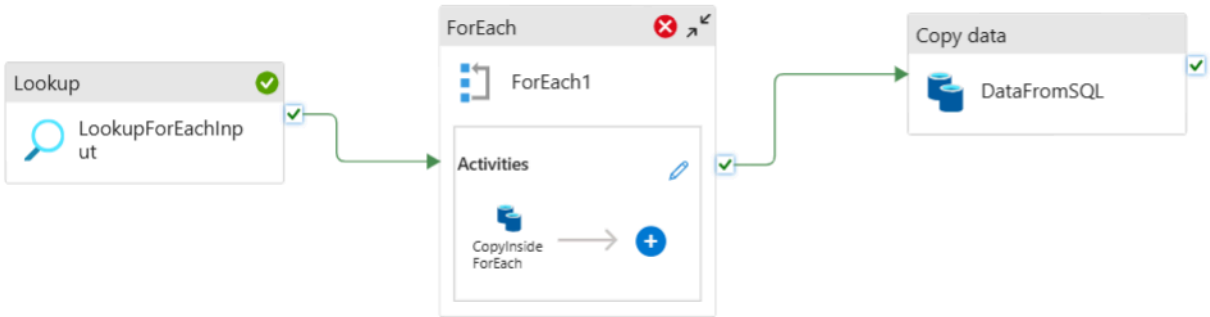

Microsoft Azure
Data Factory
olist-ecom-data-factory-sid
Search
mohapatra.si@northeastern.edu
NORTHEASTERN UNIVERSITY
Validate all
Publish all
Auto Save
Preview experience
On
Activities
Search activities
Databricks
Data Lake Analytics
General
Append variable
Delete
Execute Pipeline
Execute SSIS package
Fail
Get Metadata
Lookup
Stored procedure
Script
Set variable
Validation
Validate
Cancel options
Add trigger
Lookup
LookupForEachInput
ForEach
ForEach1
Activities
CopyInside ForEach
Copy data
DataFromSQL
Copy data
Copy Data Test
Parameters
Variables
Settings
Output
Activity name
Activity st...
Activit...
Run start
Duration
Integration runtime
User prop...
Activity run ID
CopyInsideForEach
Queued
Copy data
10/5/2025, 2:33:17 PM
13s
ed1d2309-c958-4164-b290-6429489a73
CopyInsideForEach
Succeeded
Copy data
10/5/2025, 2:33:00 PM
17s
AutoResolveIntegrationRuntime (East US)
a08a6215-1fd0-4cc4-ac20-eceb78a55cd
CopyInsideForEach
Succeeded
Copy data
10/5/2025, 2:32:44 PM
15s
AutoResolveIntegrationRuntime (East US)
bfb243e2-09a6-4bcb-b0e9-0d01bde336
CopyInsideForEach
Failed
Copy data
10/5/2025, 2:32:28 PM
15s
AutoResolveIntegrationRuntime (East US)
237bfa0e-5d1e-4336-a175-ce99eb25415
CopyInsideForEach
Succeeded
Copy data
10/5/2025, 2:32:05 PM
21s
AutoResolveIntegrationRuntime (East US)
0a24d86c-f38d-47e6-b84a-1411f4c1ac21
ForEach1
In progress
ForEach
10/5/2025, 2:32:05 PM
1m 25s
c85f14b7-71cd-41b5-bc6d-dd0ae395c5b
LookupForEachInput
Succeeded
Lookup
10/5/2025, 2:31:51 PM
14s
AutoResolveIntegrationRuntime (East US)
df356f99-bf6a-4414-b6d7-cc32423ff0e9
Copy Data Test
Inactive
Copy data
10/5/2025, 2:31:51 PM
Less than 1s
Unknown
244e9799-f836-4613-a8fc-1bfec8628e47





The value represents the JSON structure that we got from the raw link of GitHub, not the count.

What is LOOKUP, ForEach, Copy Data?



This combination of activities is a powerful and common pattern in data engineering. Think of it like a smart system for completing a list of chores.

Lookup (LookupForEachInput):

What it is: The "List Maker." The Lookup activity's job is to read a small amount of data from a source.

Its Role Here: It runs first to create a "to-do list." It queries a source (like a database control table or a configuration file) to get a list of all the tables or files that need to be copied. Its output is simply this list.

ForEach (ForEach1):

What it is: The "Worker." The ForEach is a loop. It takes the "to-do list" generated by the Lookup activity and performs a set of actions for every single item on that list.

Its Role Here: It examines the list of tables provided by the Lookup and states, "Okay, for table 1, I will do these tasks. Now for table 2, I will do the same tasks..." and so on, until the list is complete.

Copy Data (CopyInsideForEach):

What it is: The "Specific Chore." This is the workhorse activity in Azure Data Factory that moves data from a source to a sink.

Its Role Here: It is the task inside the ForEach loop. For every table name the ForEach loop processes, the Copy Data activity is the specific action that actually connects to that table and moves its data to the destination.

In summary: The Lookup finds out what to do, the ForEach manages doing it for everything on the list, and the Copy Data is the actual work being done for each item.

Can we assume this as a Production-Grade Pipeline? If yes, then why?
Yes, absolutely. The architectural pattern you have built (Lookup -> ForEach -> Copy) is not just a simple pipeline; it is a metadata-driven framework, which is a best practice and definitely considered production-grade.

While a full production deployment would also include robust error handling, logging, and monitoring, the core design pattern itself is excellent for the following reasons:

It is Dynamic and Data-Driven: The pipeline is not static or hardcoded. It intelligently discovers the work it needs to do at runtime by using the Lookup activity. If a new table is added to your source database that matches the Lookup's query, the pipeline will automatically include it in the next run without you ever needing to modify or republish the pipeline.

It is Highly Scalable: This pattern works just as well for 10 tables as it does for 1,000. Because you don't have to manually create a Copy data activity for each table, the solution scales effortlessly as your data sources grow.

It is Easy to Maintain: Imagine you needed to change how the data is copied (e.g., add a new mapping rule or change a setting). You only have to modify the single Copy Data activity inside the ForEach loop, and that change is instantly applied to all 1,000 tables. In a static design, you would have to manually edit 1,000 separate activities, which is slow and prone to errors.

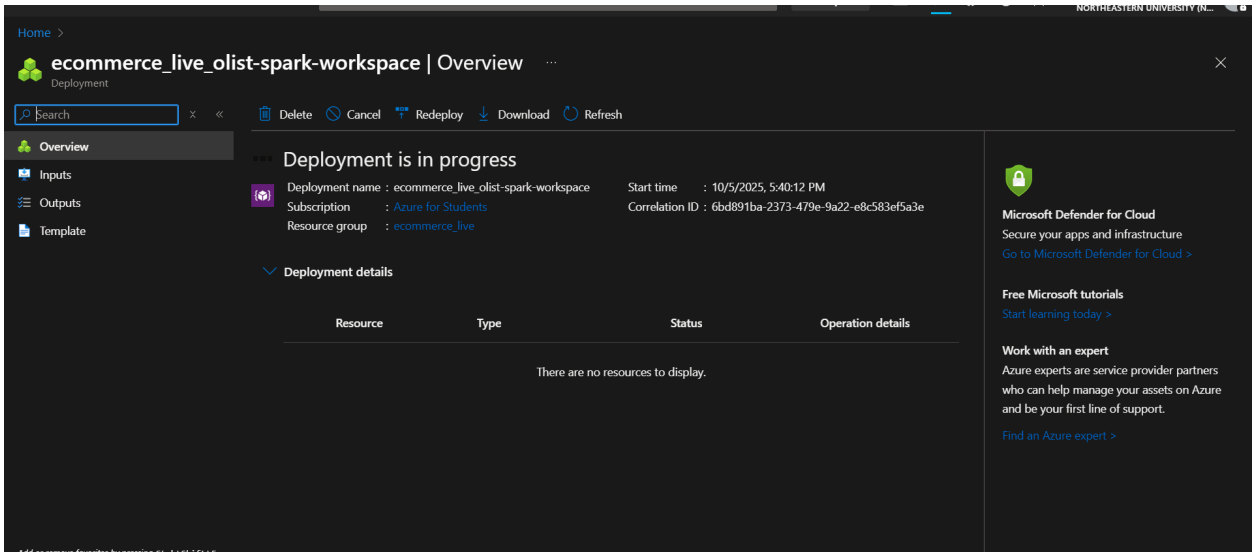
It is Efficient: The ForEach activity can be configured to run in parallel. This means it can copy multiple tables simultaneously (e.g., 20 at a time), drastically reducing the total time it takes to ingest all your data compared to processing them one by one.

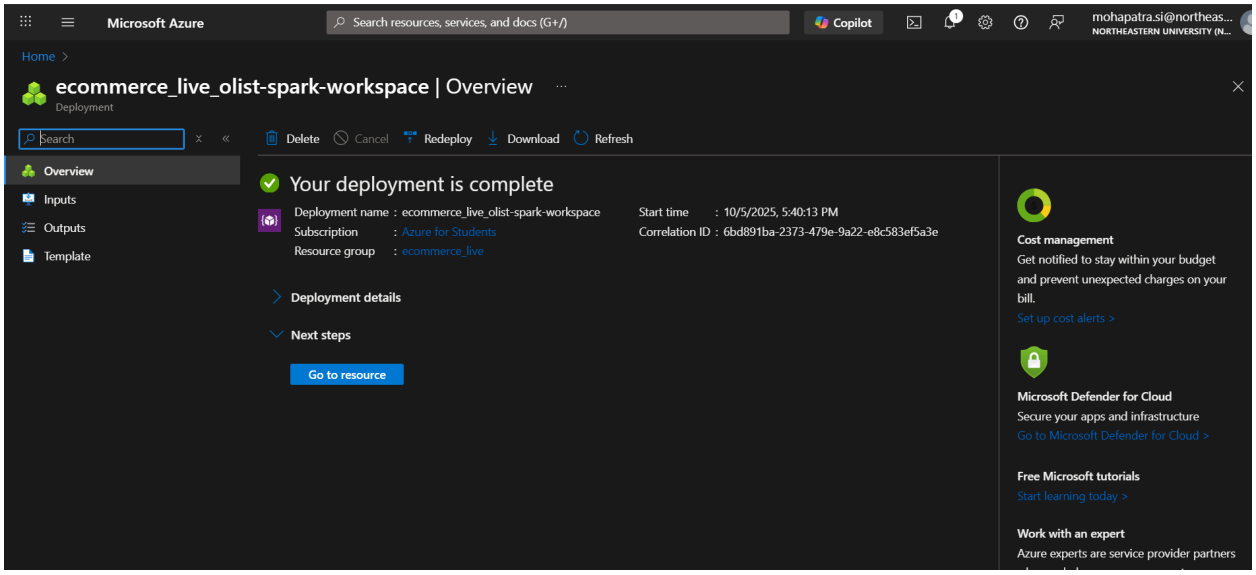
NOW THE BRONZE LEVEL OF WORK IS COMPLETED IN MEDALLION ARCHITECTURE.

—----------

NEXT STEPS: We need to read the bronze layer data to **Azure Databricks** and read some data from **MongoDB** for enrichment & transformations, and we will see how we can transform the data in the **Silver** layer.

DATABRICKS:
Databricks is actually a third-party software. All major cloud services provide Databricks on top of their platform.





You can see I have deployed my Databricks workspace in the Azure cloud service.

AZURE DATABRICKS:

- Spark-powered cloud service
- Integrated with Azure
- Handles Big Data easily
- Great for Machine Learning

AZURE DATABRICKS WORKFLOW:

- Reading the data from ADLS Gen 2
- Performing Basic Transformation like cleaning, renaming, & filtering
- Using the JOIN operation to integrate multiple datasets
- Enriching the data via MongoDB
- Performing aggregation and deriving insights
- Writing final data back to ADLS Gen 2

What is a Compute in Databricks?

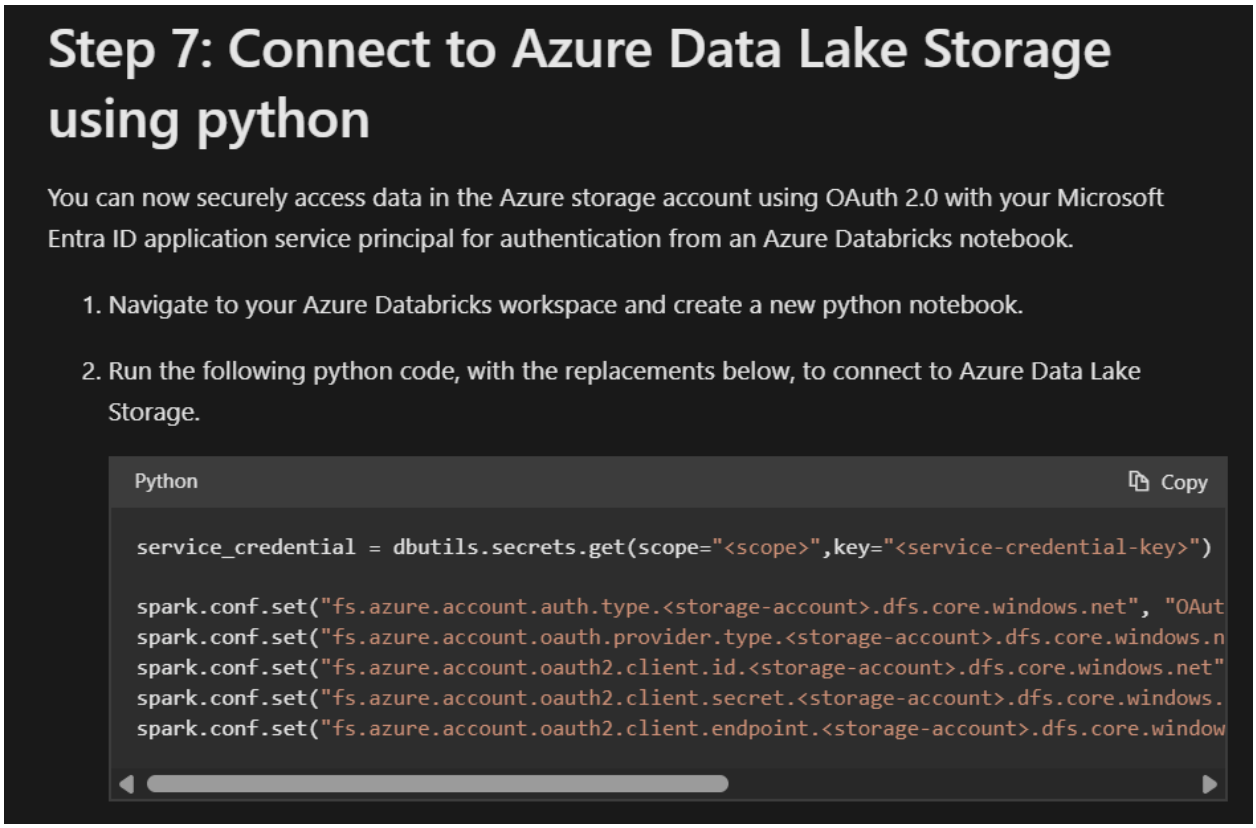
Compute is just a hardware that we will ask Azure to give to us. In **Databricks** (and most cloud platforms), “compute” means the actual processing power — the virtual machines (VMs), CPUs, RAM, and sometimes GPUs — that do the work. In **Databricks**, **compute** refers to the clusters of virtual machines that actually process your data and run your notebooks or jobs. Storage just holds the data, while compute does the heavy lifting. **Databricks** lets us scale this compute up or down automatically, so we only pay for what we use.

Azure Databricks Workflow which I am planning to implement:

- Reading the data from ADLS Gen 2
- Performing basic transformations like cleaning, renaming, & filtering
- Using the JOIN operation to integrate multiple datasets
- Enriching or transferring the data via MongoDB

- Performing aggregations & deriving insights
- Writing final data back to ADLS Gen 2

Step 1: Here, 1st we need to connect our Azure Databricks to Azure Data Lake Studio Gen2. (Link) So, to make the connection, we need some key or permission here to maintain the equality I mean, both are Microsoft Azure services so both of them get connected inside Microsoft and no third party can enter.



After selecting App Registrations in Azure search, This was the error I was getting when trying to connect Databricks with my ADLS Gen2.

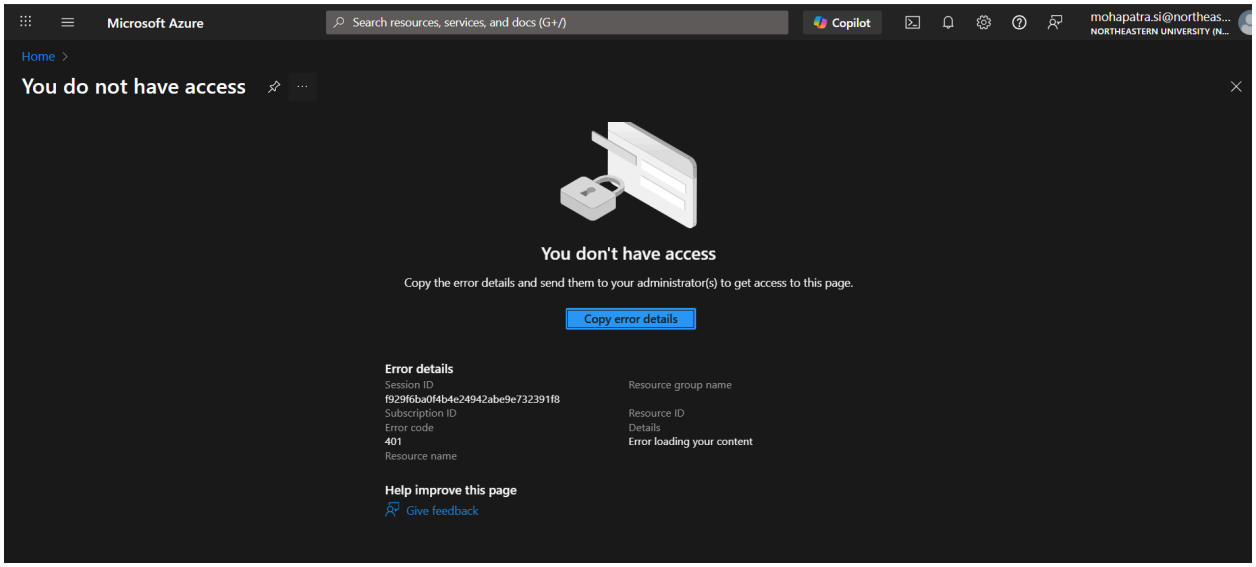
What This Error Means
The error "You do not have access" with Error code: 401 means "Unauthorized." It's a permissions issue. I was successfully logged into Azure, but my specific account (mohepatra.si@northeastern.edu) does not have the administrative rights to view or manage the "App registrations" section.

Why I didn't Have Access (The University Analogy)
Think of your university's entire Microsoft Azure setup as a large office building.

The Building (Azure Active Directory/Tenant): The entire building is managed by your university's IT administrators. They control the main entrance, the security office, and who gets what keys. The "App registrations" page is like the building's main security office. It's a central, high-level area.

Your Office (Your Azure for Students Subscription): The university has given you your own private office inside the building—this is your "Azure for Students" subscription. Inside your office, you are the owner. You can set up furniture (like Azure Data Factory, Databricks, etc.) and do whatever you want.

The error we are seeing is because our keycard (our student account) lets us into our own office, but it doesn't grant us access to the main security office for the entire building. This is a security measure to prevent students from viewing or changing administrative settings that affect the whole university.



However, the tutorial I was following demonstrated the most common and robust method for production environments: creating a **Service Principal**. A **Service Principal** is like creating a robot user with its own specific permissions. However, as you discovered, creating these "robot users" requires high-level administrative access to the "security office" (App Registrations), which I don't have with a student account.

There is another, simpler method that is perfect for this situation and still keeps the connection secure within the Microsoft cloud. I will use a method called **mounting with an Account Key.**

Instead of creating a new "**robot user**," you will temporarily use your own powerful key (the Storage Account Access Key) to establish a permanent link (a mount point) between Databricks and your Data Lake.

What I did here?
Think of it like giving your workshop (Databricks) a secure key to your storage unit (ADLS).

We got the Master Key: I went to my Azure storage account and copied its Access Key (key1). This key is like a master password that grants full access to all the data inside the service.

We Used a **Secure Lockbox**: Instead of pasting this sensitive key directly into your code (which is insecure), we created a **Databricks Widget** outside of the cell. This widget is like a secure lockbox at the top of my notebook. I pasted the key into this box, keeping it separate from my saved code.

We Built a **Secure Bridge**: Finally, I ran the **dbutils.fs.mount()** command. This command did two things:

- It securely took the key from the lockbox.
- It used the key to build a permanent, direct link—a mount point—from Databricks to my storage container.

The result is that my olistdata container now appears as a simple folder inside Databricks at the path /mnt/olistdata. Anyone can now read and write files to our storage as if it were a local directory. ✅

For Widget:

```
dbutils.widgets.text("storage_key", "Paste Your Key Here", "Storage Account Key")
```

For Connection:

```
# 1. Defining my storage account and container names
storage_account_name = "olistdatastorageaccount4"
container_name = "olistdata"
mount_point = f"/mnt/{container_name}"

# 2. Securely getting the key from the text box
storage_key_from_widget = dbutils.widgets.get("storage_key")

# 3. Unmounting if it already exists
try:
  dbutils.fs.unmount(mount_point)
except:
  pass

# 4. Creating the connection using a consistent protocol
# We will use 'wasbs' and '.blob.core.windows.net' for both the source and the configuration.
source_url = f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/"
config_key = f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net" # <-- This key is now consistent

dbutils.fs.mount(
  source = source_url,
  mount_point = mount_point,
  extra_configs = {config_key: storage_key_from_widget}
)

print(f"✅ Success! The '{container_name}' container is now available at {mount_point}")
```

# 5. Verifying the connection by listing the files INSIDE the 'bronze' folder
display(dbutils.fs.ls(f"{mount_point}/bronze/"))

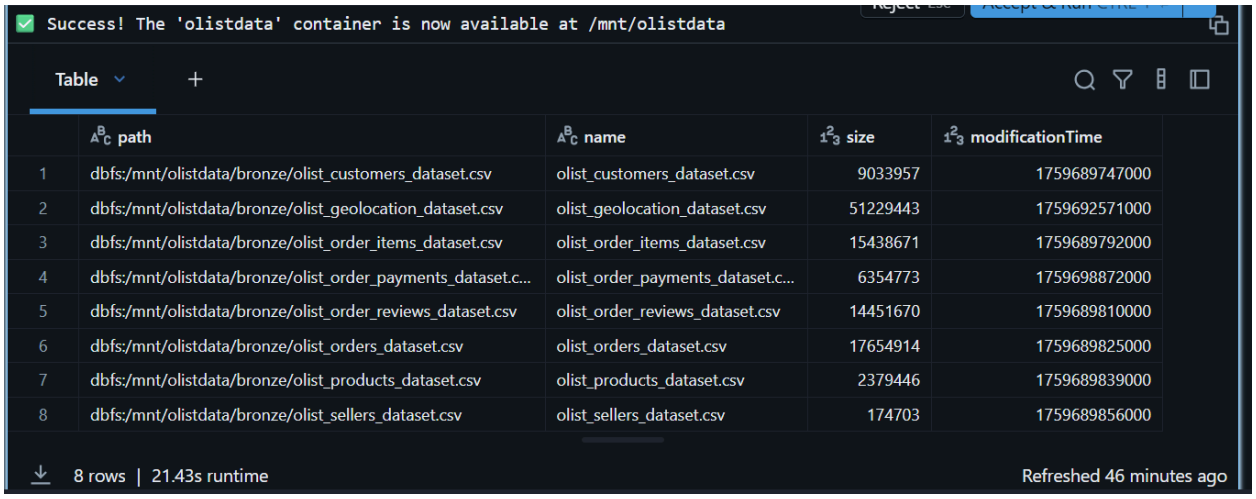

Success! The 'olistdata' container is now available at /mnt/olistdata

Table +

	path	name	size	modificationTime
1	dbfs:/mnt/olistdata/bronze/olist_customers_dataset.csv	olist_customers_dataset.csv	9033957	1759689747000
2	dbfs:/mnt/olistdata/bronze/olist_geolocation_dataset.csv	olist_geolocation_dataset.csv	51229443	1759692571000
3	dbfs:/mnt/olistdata/bronze/olist_order_items_dataset.csv	olist_order_items_dataset.csv	15438671	1759689792000
4	dbfs:/mnt/olistdata/bronze/olist_order_payments_dataset.c...	olist_order_payments_dataset.c...	6354773	1759698872000
5	dbfs:/mnt/olistdata/bronze/olist_order_reviews_dataset.csv	olist_order_reviews_dataset.csv	14451670	1759689810000
6	dbfs:/mnt/olistdata/bronze/olist_orders_dataset.csv	olist_orders_dataset.csv	17654914	1759689825000
7	dbfs:/mnt/olistdata/bronze/olist_products_dataset.csv	olist_products_dataset.csv	2379446	1759689839000
8	dbfs:/mnt/olistdata/bronze/olist_sellers_dataset.csv	olist_sellers_dataset.csv	174703	1759689856000

8 rows | 21.43s runtime Refreshed 46 minutes ago

# Reading data in pyspark (all 8 csv files)
# Defining the base path
base_path = "/mnt/olistdata/bronze"

# Reading each file into its own variable
orders_df = spark.read.option("header", "true").csv(f"{base_path}/olist_orders_dataset.csv")
payments_df = spark.read.option("header", "true").csv(f"{base_path}/olist_order_payments_dataset.csv")
reviews_df = spark.read.option("header", "true").csv(f"{base_path}/olist_order_reviews_dataset.csv")
items_df = spark.read.option("header", "true").csv(f"{base_path}/olist_order_items_dataset.csv")
customers_df = spark.read.option("header", "true").csv(f"{base_path}/olist_customers_dataset.csv")
sellers_df = spark.read.option("header", "true").csv(f"{base_path}/olist_sellers_dataset.csv")
geolocation_df = spark.read.option("header", "true").csv(f"{base_path}/olist_geolocation_dataset.csv")
products_df = spark.read.option("header", "true").csv(f"{base_path}/olist_products_dataset.csv")

# Now we can display any of them
display(customers_df)

Step 2: # Need to deal with mongodb inside pyspark.
To analyze the product_category_translation data, we first had to move it from a private, local database to a cloud database where Databricks could access it. First, we used MongoDB Compass to export the local product_category_translation collection into a JSON file. Then, we went to MongoDB Atlas (our cloud database service) and set up a new free cluster. This involved creating a database user (with a secure password) and whitelisting all IP addresses to allow our Databricks Factory and Databricks services to connect. Next, we used Compass again to connect to our new cloud cluster and imported the JSON file into a new database called olist_data. Finally, in our Databricks notebook, we wrote Python code using the pymongo

library. This code securely connected to our Atlas cluster, read all 71 documents from the collection, and loaded them into a new Spark DataFrame named translation_df for analysis.

Code:

```
# 1. Importing the necessary library
from pymongo import MongoClient
import pandas as pd
```

```
# 2. Setting up my connection string
# IMPORTANT: Replacing <username>, <password>, and the cluster URL with your actual credentials from MongoDB Atlas.
username = "abc"
password = "abc"
cluster_url = "abc" # Atlas connection string
```

```
# These are the names of the database and collection I created
db_name = "olist_data"
collection_name = "product_category_translation"
```

```
# Building the connection string
uri = f"mongodb+srv://{username}:{password}@{cluster_url}/?retryWrites=true&w=majority"
```

```
print("Variables are set. Ready to connect in the next cell.")
```

```
# This uses the 'uri' variable from Cell 1

print("Attempting to connect to MongoDB Atlas...")
try:
    client = MongoClient(uri)
    db = client[db_name]
    collection = db[collection_name]

    # Pinging the server to confirm the connection
    client.admin.command('ping')
    print("✅ MongoDB connection successful!")
```

```
except Exception as e:
    print("❌ Connection failed. Please check your username, password, or IP whitelist settings.")
    print(f"Error details: {e}")
```

```
# Cell 3: Reading the Data from the Collection
# This uses the 'collection' variable from Cell 2

print(f"Reading data from the '{collection_name}' collection...")

# Finding all documents ({}) and convert them to a list
data = list(collection.find({}))

if data:
    # Your file had 71 documents or records, so this should match i guess
    print(f"✅ Successfully found {len(data)} documents.")
else:
    print("❌ No data found. Make sure you imported the data to your Atlas collection.")
```

```
# Cell 4: Converting Data to a Spark DataFrame
# This uses the 'data' variable from Cell 3

if 'data' in locals() and data:
    print("Converting data to a Spark DataFrame...")

    # Converting the list of data to a Pandas DataFrame
    pandas_df = pd.DataFrame(data)

    # Dropping the '_id' column that MongoDB automatically adds
    if '_id' in pandas_df.columns:
        pandas_df = pandas_df.drop(columns=['_id'])

    # Converting the Pandas DataFrame into a Spark DataFrame
    translation_df = spark.createDataFrame(pandas_df)

    print("✅ Data successfully loaded into a Spark DataFrame named 'translation_df'.")
```

```
else:
    print("No data to convert. Please run Cell 3 first.")
```

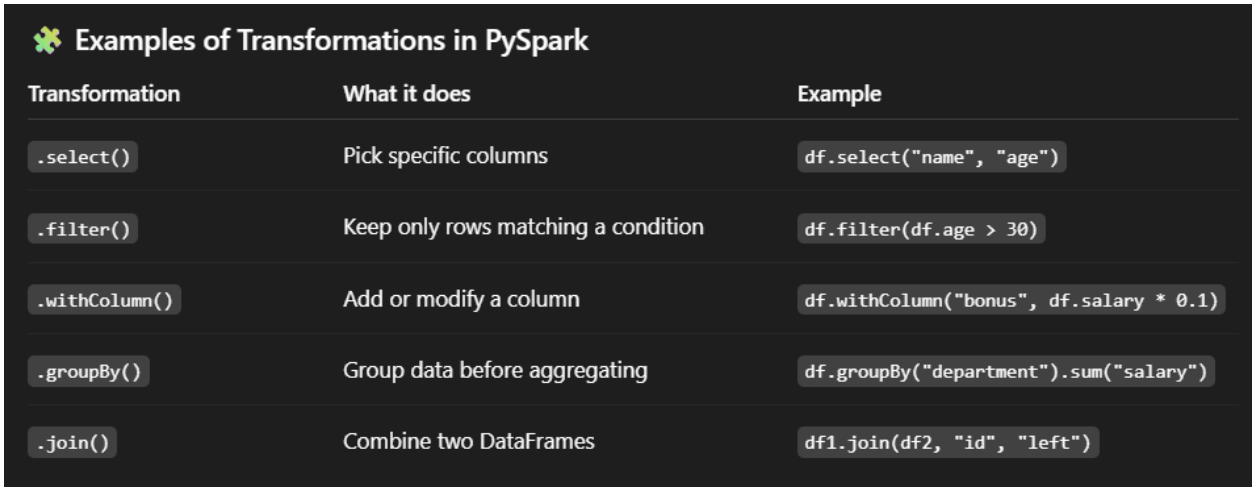
```
# Cell 5: Checking Shape and Display the Data
# This uses the 'translation_df' variable from Cell 4

if 'translation_df' in locals():
    # Getting the row and column counts
    row_count = translation_df.count()
    column_count = len(translation_df.columns)

    print(f"--- Data Shape ---")
    print(f"Number of rows: {row_count}")
    print(f"Number of columns: {column_count}")
    print(f"--------------------")

    # Displaying the final DataFrame
    display(translation_df)
else:
    print("No DataFrame to display. Please run Cell 4 first.")
```

Step 3: Joining the datasets.

```
# Joining Orders and Customers
orders_customers_df = orders_df.join(
   customers_df,
   orders_df.customer_id == customers_df.customer_id,
   "left"
)

# Joining the above result with Payments
orders_payments_df = orders_customers_df.join(
   payments_df,
   orders_customers_df.order_id == payments_df.order_id,
   "left"
)

# Joining Orders + Payments with Items using order_id
orders_items_df = orders_payments_df.join(items_df, "order_id", "left")

# Joining Orders + Items with Products
```

```
orders_items_products_df = orders_items_df.join(
    products_df,
    orders_items_df.product_id == products_df.product_id,
    "left"
)

final_df = orders_items_products_df.join(
    sellers_df,
    orders_items_products_df.seller_id == sellers_df.seller_id,
    "left"
)
```

(**NOTE: Spark Transformations & Actions: Transformations** are instructions or recipes you give Spark to describe what you want to do with data, but no work is actually done yet.



🧩 Examples of Transformations in PySpark

Transformation	What it does	Example
`.select()`	Pick specific columns	`df.select("name", "age")`
`.filter()`	Keep only rows matching a condition	`df.filter(df.age > 30)`
`.withColumn()`	Add or modify a column	`df.withColumn("bonus", df.salary * 0.1)`
`.groupBy()`	Group data before aggregating	`df.groupBy("department").sum("salary")`
`.join()`	Combine two DataFrames	`df1.join(df2, "id", "left")`

Actions are commands that instruct Spark to execute the transformations and compute the actual results.

Examples of Actions in PySpark

Action	What it does	Example
`.show()`	Displays data on screen	`df.show(5)`
`.count()`	Counts total rows	`df.count()`
`.collect()`	Returns all rows to the driver	`df.collect()`
`.first()`	Returns the first row	`df.first()`
`.save()` / `.write.csv()`	Saves results to storage	`df.write.csv("/path/output")`

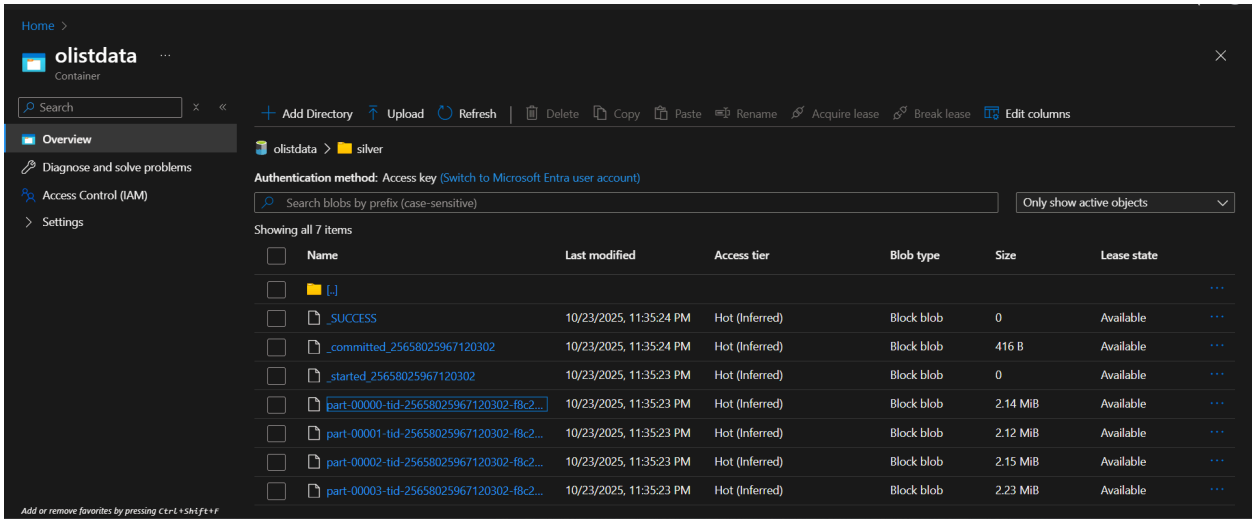
)

Step 4: Loading the transformed data into the Silver Layer of olistdata container.

```
# Dropping duplicate columns in the final_df
# This function scans through all the columns in your DataFrame, finds any column names that appear more than once, and removes the duplicate ones — keeping only the first occurrence.
def remove_duplicate_columns(df):
    columns = df.columns
    seen_columns = set()
    columns_to_drop = []

    for column in columns:
        if column in seen_columns:
            columns_to_drop.append(column)
        else:
            seen_columns.add(column)

    df_cleaned = df.drop(*columns_to_drop)
    return df_cleaned

final_df = remove_duplicate_columns(final_df)
```

```
# This line is saving the final Spark DataFrame (final_df) as a Parquet file into the Azure Databricks storage location /mnt/olistdata/silver (Silver) folder. It's like a read_csv in pandas.
```

```
final_df.write.mode("overwrite").parquet("/mnt/olistdata/silver")
```



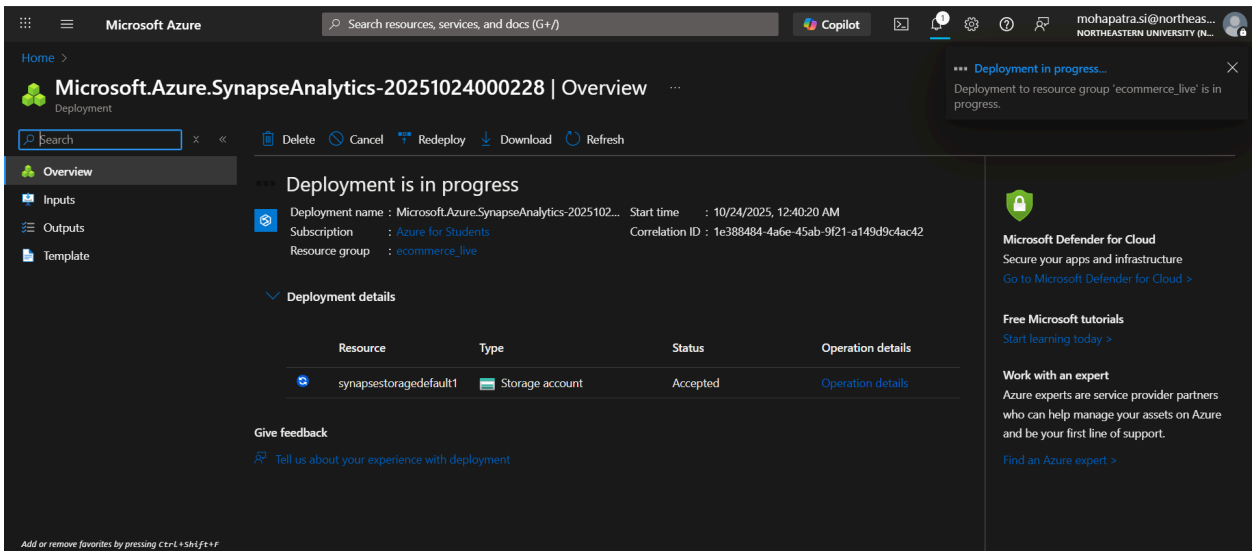
In the above container (olistdata), we can see that the transformed data has been stored in the **SILVER** layer.

NOW THE SILVER LEVEL OF WORK IS COMPLETED IN MEDALLION ARCHITECTURE.

—----------

NEXT STEPS: We need to work on **Synapse Analytics.** Here, we will be building the **GOLD** layer of the Medallion architecture.

Step 1: Setting up the Azure Synapse Analytics. **Azure Synapse** is a cloud-based data warehouse and analytics service that combines **data integration, enterprise data warehousing, & big data analytics.**



Sometimes, we get these kind of errors in Azure while deploying the Synapse Analytics. There are reasons for it as well.

Why do Synapse deployments fail?

Think of a Synapse workspace like opening a new mall. We're not just creating the mall—we also need parking (SQL server), security (managed identity/permissions), roads (networking), and a storage warehouse (Data Lake). If any of those can't be created in the location you selected, the entire "open the mall" action fails.

One reason for this can be:

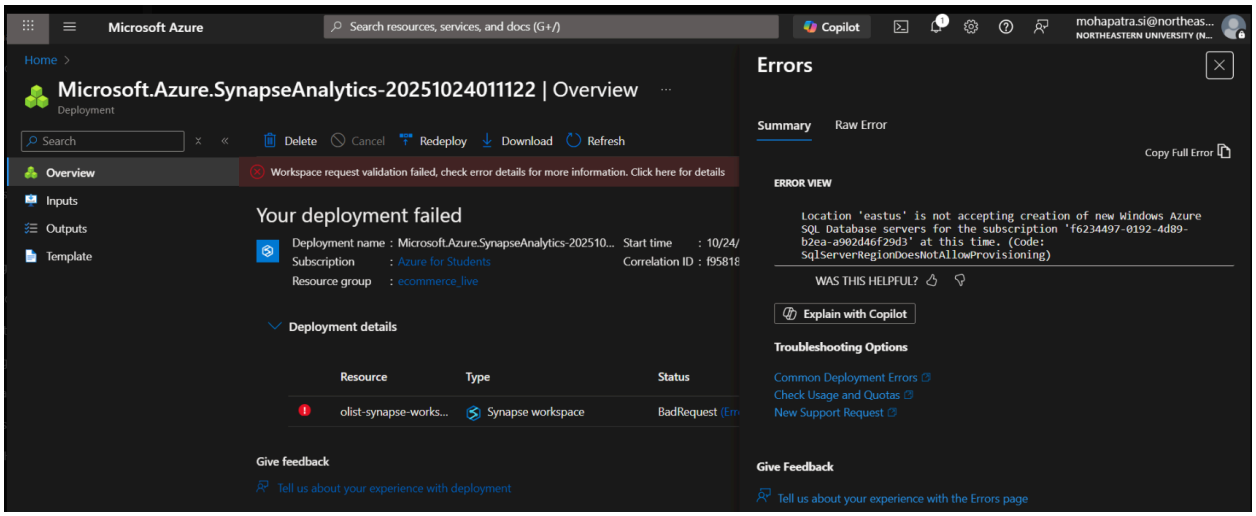
Region blocks / capacity limits (my exact error)

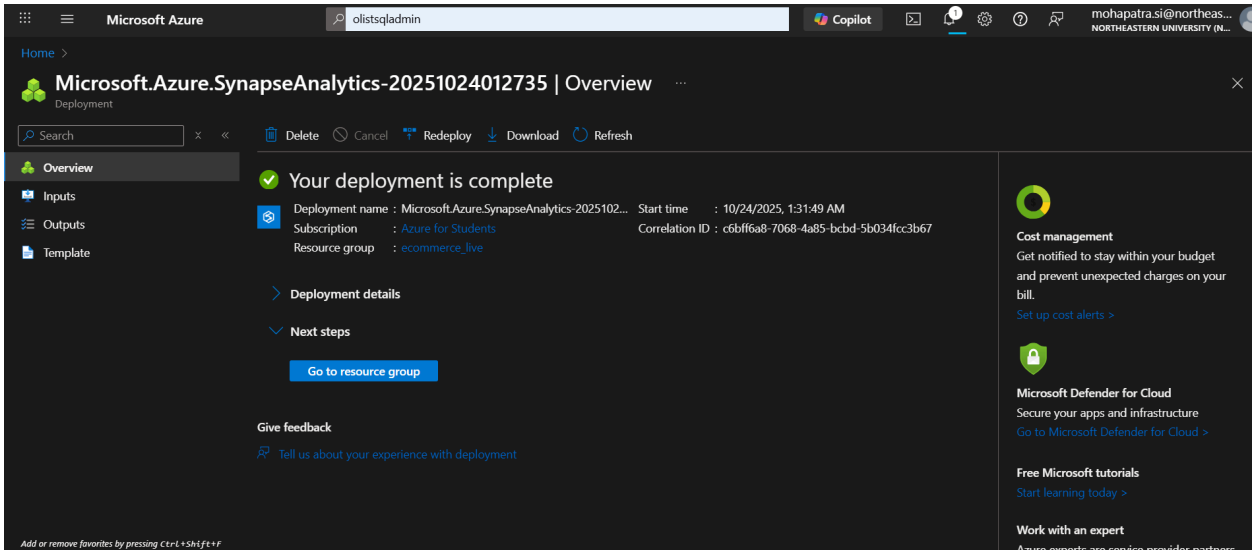
Message: SqlServerRegionDoesNotAllowProvisioning … Location 'eastus' is not accepting creation of new … SQL Database servers…

What it means: In that region (e.g., East US), my subscription isn't allowed to create new Azure SQL Server resources right now—often due to capacity or subscription-type limits (e.g., Free Trial/Azure for Students).

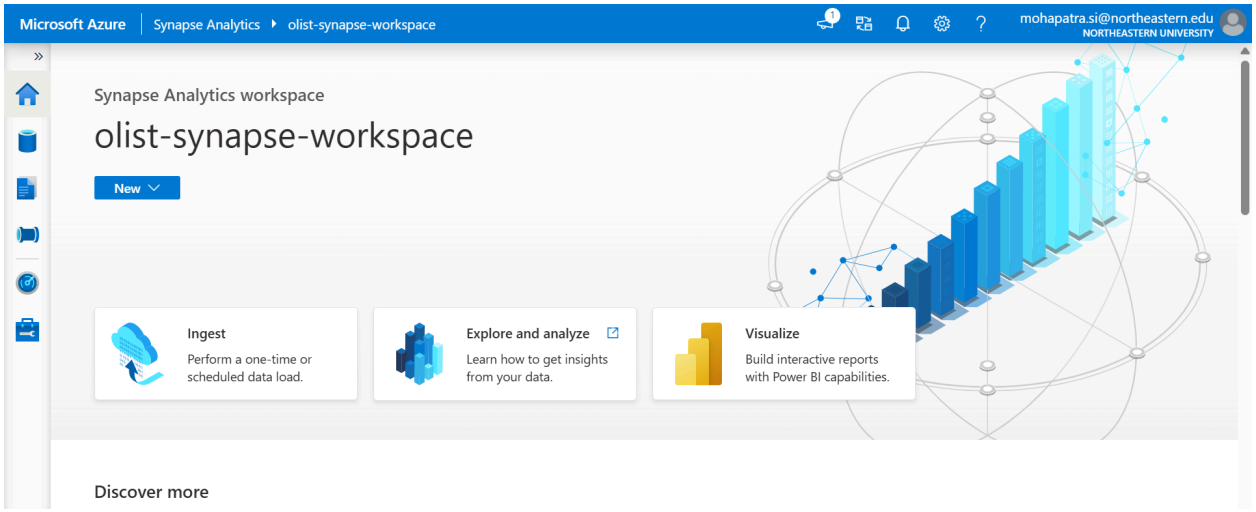
Real example: We pick East US for Synapse, but Synapse needs to spin up a SQL server there; the region says "no new SQL servers allowed," so deployment fails.

Instead we can pick another region (e.g., East US 2, Central US, West US 2) to solve this, which is what happened in my case.





Step 2: Now, we will work on Azure Synapse Analytics. Below is the UI of Synapse Analytics. Just like the way we gave a permission to Azure Databricks to read data from ADLS Gen2, similarly, we need to give permission to Azure Synapse Analytics to read data from ADLS Gen2.



It enables users to ingest, prepare, manage, and serve data for business intelligence (BI) and machine learning — all within a single, serverless environment. In other words, it's like a one-stop platform where we can bring in data (ingest), clean and analyze it (transform), and build dashboards (visualize)—without leaving Azure.

Component	Description	Example
Synapse Studio	The web interface for managing all Synapse tasks — similar to a "control center" for your data.	You access it via the Azure portal or a direct URL.

SQL Pools	Provide powerful querying for structured data using SQL. Includes **Dedicated SQL Pool** (provisioned) and **Serverless SQL Pool** (on-demand).	Running `SELECT * FROM sales_data` on large datasets.
Spark Pools	Allow data exploration and transformation using **PySpark, Scala, or .NET**.	Cleaning and aggregating big data stored in Data Lake.
Data Integration Pipelines	Automate movement and transformation of data from multiple sources (like Azure Data Lake, Databricks, or APIs).	A daily job that loads sales data into Synapse tables.
Linked Services	Secure connections to external data sources such as **Azure Storage**, **SQL DB**, **Databricks**, or **Power BI**.	Connecting Synapse to ADLS Gen2 to read Parquet files.
Data Flows	Visual, drag-and-drop transformations (no coding).	Converting CSV data into normalized tables visually.

—----------

Type	Description	Real-Life Example
SQL Database	Uses SQL language for querying and analytics. Best for structured, relational data (tables, columns).	Storing customer transactions, order records, or product details.
Lake Database	Schema-on-read database created directly over data in ADLS. Works well for raw or semi-structured files like CSV, JSON, or Parquet.	Creating a database view directly over millions of sales CSV files stored in your data lake.

—----------

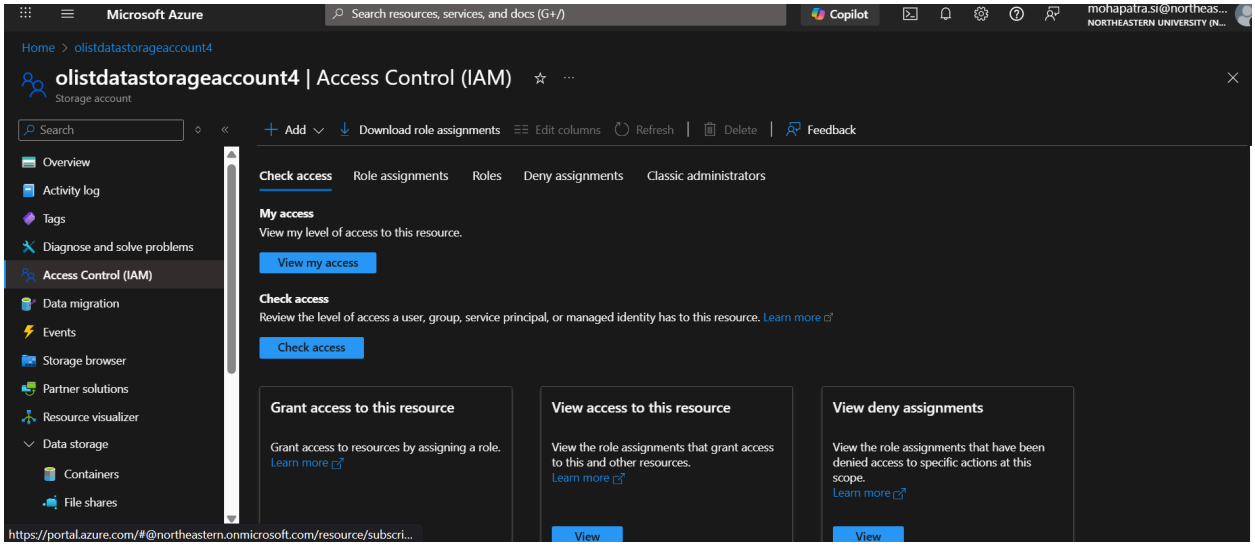
Category	Description	Example Use Case
Synapse	Run SQL scripts or notebooks directly inside Synapse.	Execute a SQL query to aggregate sales data.
Move and Transform	Copy, transform, or clean data between sources.	Copy data from ADLS → Synapse SQL database.

Azure Data Explorer	Integrate with Kusto Query Language (KQL) for log analytics.	Analyze IoT or telemetry data.
Azure Function	Trigger serverless functions during workflow.	Validate data quality before loading.
Batch Service	Execute batch compute jobs on Azure Batch.	Process multiple files in parallel.
Databricks	Run Databricks notebooks or jobs directly from Synapse.	Execute a PySpark job for feature engineering.
Data Lake Analytics	Run U-SQL jobs on large datasets.	Perform distributed queries on raw logs.
General	Simple tasks like “Wait,” “Execute Web,” or “Set Variable.”	Pause execution for 30 seconds.
HDInsight	Integrate with Hadoop or Spark-based clusters.	Run MapReduce or Hive queries.
Iteration & Conditionals	Control logic like loops or conditional branching.	“If file exists, then load; else skip.”
Machine Learning	Connect to Azure ML models for predictions.	Run a trained model to predict customer churn.

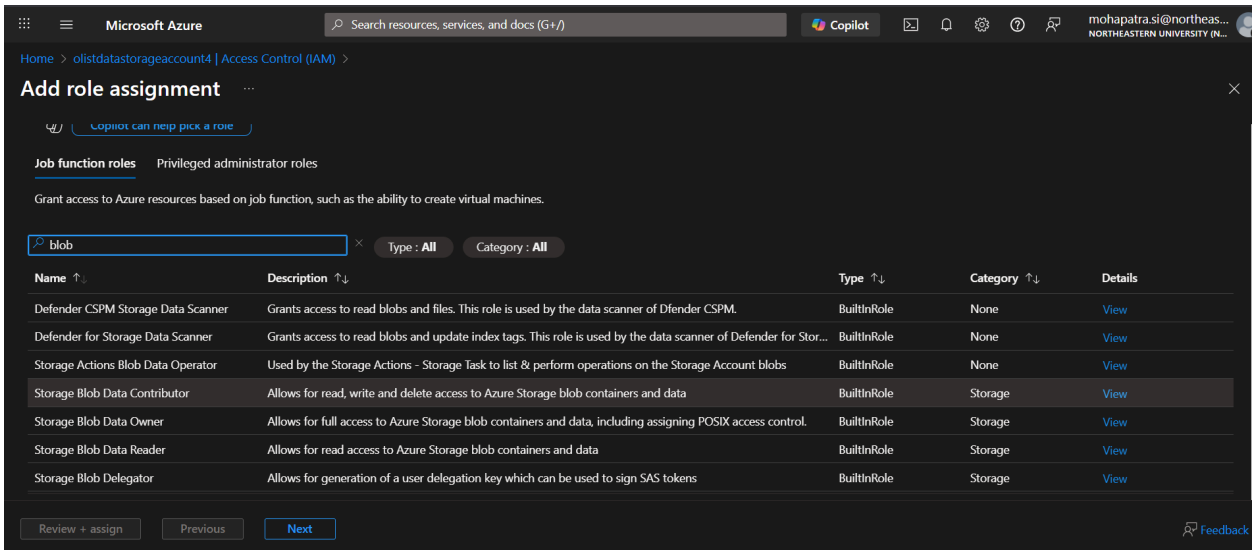
—----------

We will be reading data from **the SILVER** layer and creating our own database within Synapse Analytics.

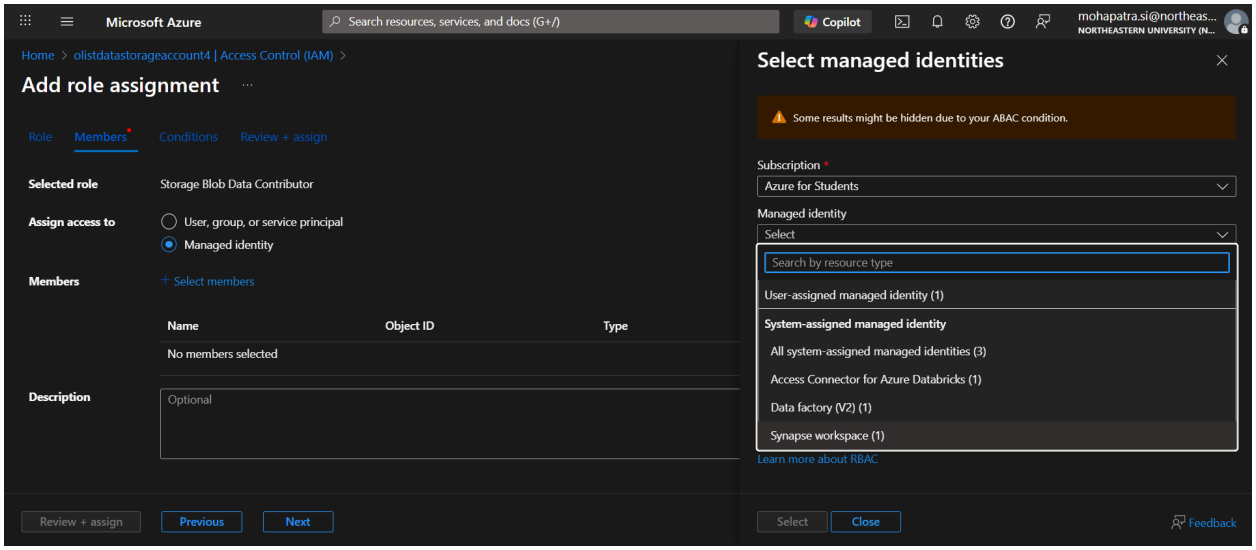
Step 3: We will give **permissions** to **Synapse,** so we will go to **App Registration** like this.

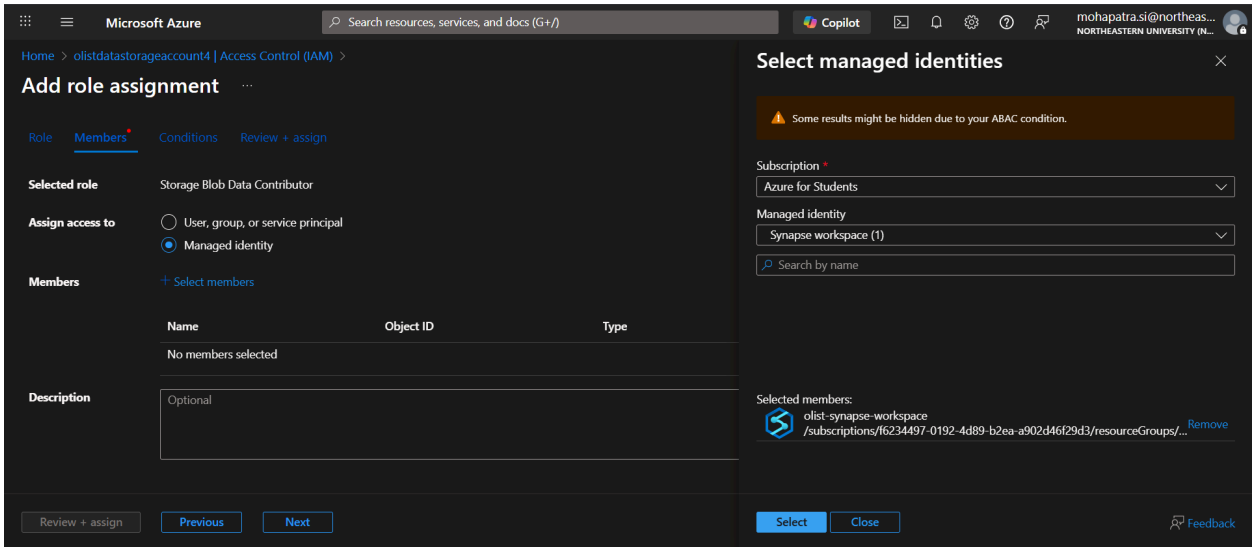


Go to IAM (Identity Access Management)



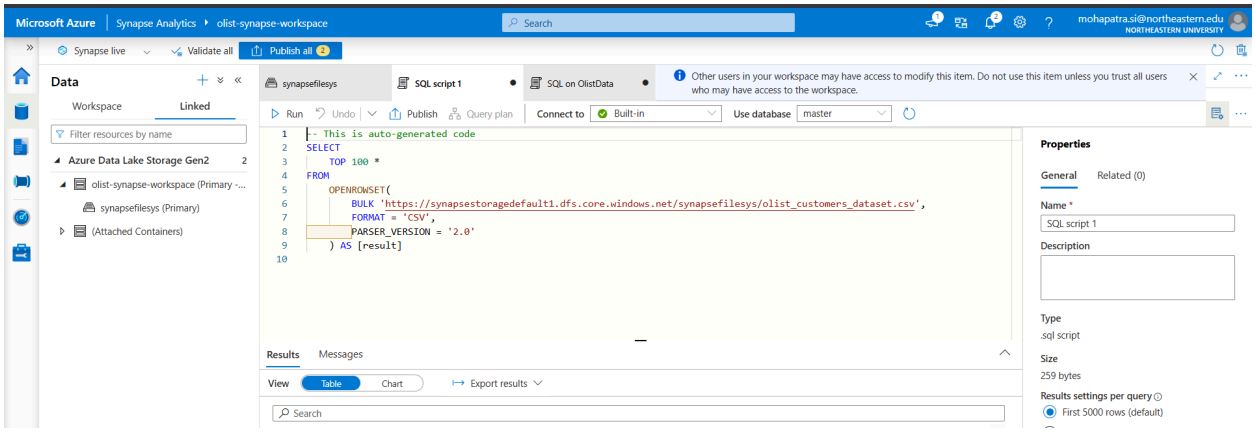
Search ‘Blob’ and click on ‘Storage Blob Data Contributor.’ It gives both **write** and **read** access.



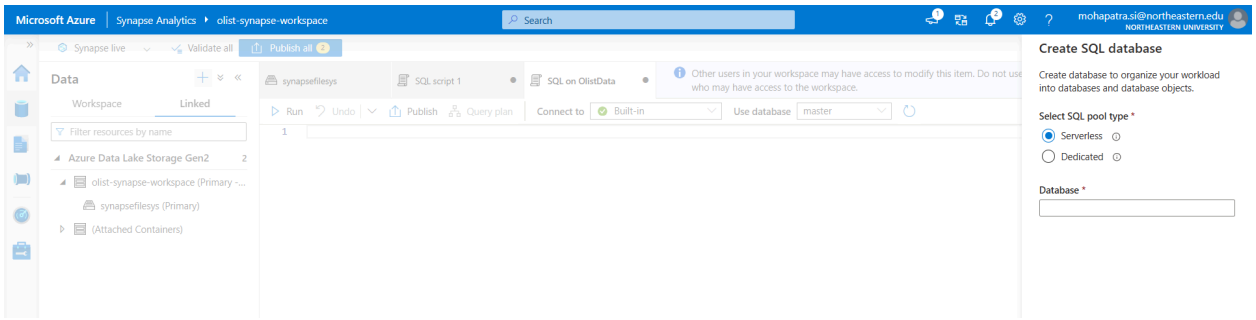


Click Next. Then Click on ‘Managed Identity’. Click on ‘Select Members’. Select ‘Synapse Workspace’

Step 4: This is how Azure Synapse Analytics looks like.



When I am trying to create a new SQL Database, it is showing me two types of SQL Pool Type.



There are 2 types of SQL Pool in Azure:

Factor	Dedicated SQL Pool	Serverless SQL Pool
Cost Model	Pay-per-capacity, charges apply whether resources are used or not	Pay-per-query, charged based on data processed during execution
Tech Stack	Uses MPP (Massively Parallel Processing) architecture	Uses distributed query processing, optimized for data lake storage
Implementation Effort	Requires more setup and configuration	Minimal setup, often ready-to-use out of the box
Performance	Consistent, optimized for high-performance workloads	Performance can vary based on the complexity and size of the query
Storage and Compute	Storage and computing are decoupled but need to be managed independently	Storage is handled via a data lake, and compute is allocated on demand
Use Cases	Ideal for data warehousing, large scale ETL processes, and fixed workload patterns	Best for ad-hoc queries, data exploration, and dynamic workloads

⚙️ 1. Serverless SQL Pool (Pay-as-you-go)

💡 Simple Explanation:

- You don't have to set up or manage any server.
- You just run a query directly on files in your **Data Lake (ADLS)** — like CSV, JSON, or Parquet files.
- Azure will automatically provide the computing power needed to run that query.
- Once done, it shuts down automatically.
- You only **pay for the data processed** by your queries (not for idle time).

🧍 Real-life Analogy:

Think of it like **using Uber or Ola** 🚗

- You **don't own the car** (no servers to maintain).
- You **call a ride only when you need it** (on-demand query).
- You **pay only for that trip** (pay-per-query).
- When your ride ends, you stop paying — no ongoing cost.

So, if you just need to **occasionally explore data**, check reports, or run analytics — this model is perfect.

⚙️ 2. Dedicated SQL Pool (Provisioned)

💡 Simple Explanation:

- You **reserve computing resources (servers)** ahead of time.
- The servers are **always running** and ready to process queries.
- You **load data into Synapse's internal storage** (not directly from Data Lake).
- You **pay a fixed cost**, even if the system is idle.

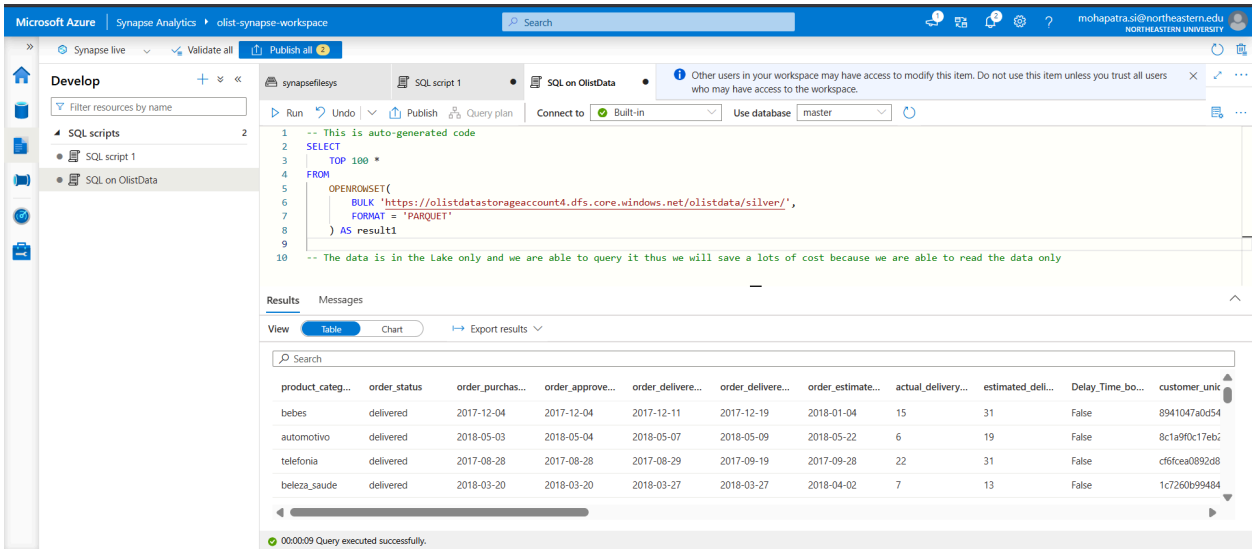
🧍 Real-life Analogy:

Think of it like **buying your own car** 🚗

- You **own the car** (fixed resources).
- It's **always available** whenever you want to drive.
- You **pay maintenance, fuel, insurance** — even if it sits in the garage unused.
- But it's **much faster and predictable** than waiting for an Uber.

This is ideal for companies with **regular heavy workloads**, like daily ETL pipelines, dashboards, or reports that must run at high speed.

Feature	Serverless SQL Pool (Pay-as-you-go)	Dedicated SQL Pool (Provisioned)
How it works	On-demand, no servers to manage	Fixed compute resources reserved
Payment model	Pay only for queries run	Pay fixed cost for reserved compute
Data location	Data stays in Data Lake (ADLS)	Data loaded into Synapse database
Use case	Great for exploring data occasionally	Best for large, frequent workloads
Setup	Easy — no setup or maintenance	Needs setup, configuration, and monitoring
Speed	Slower for big workloads	Very fast and consistent performance



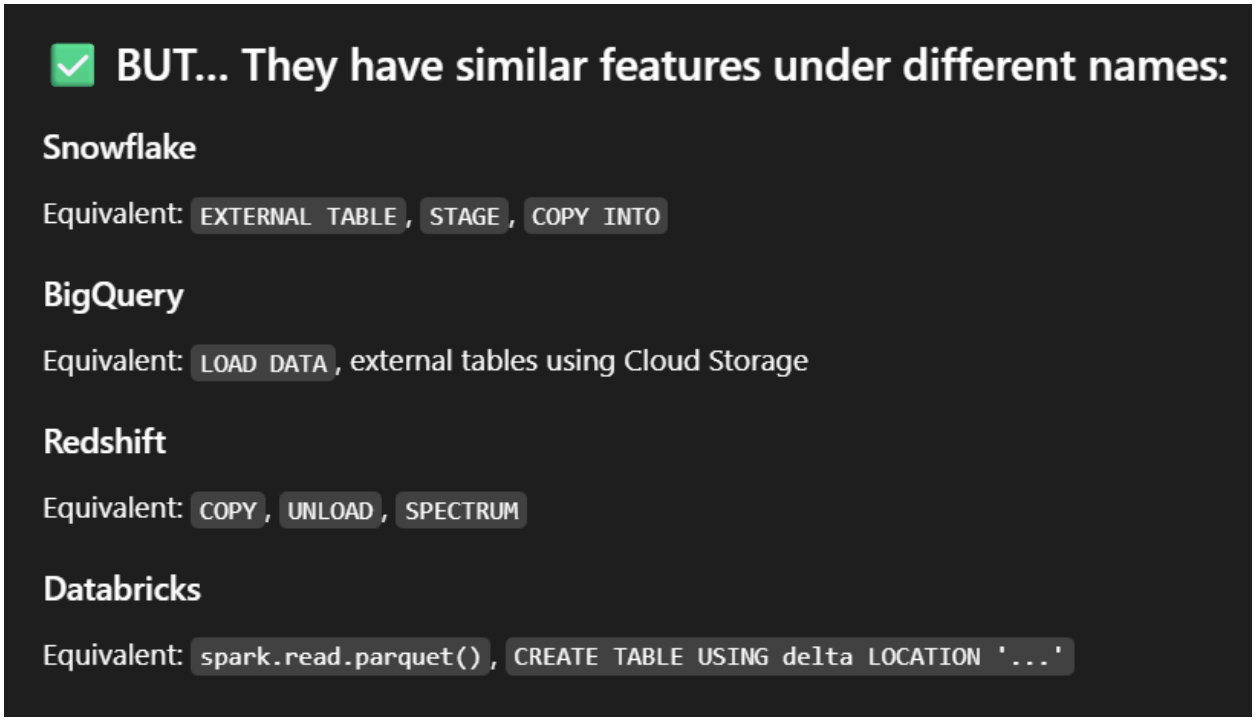
Now after running the query (with little modifications accordingly), we can see the data from the **SILVER** layer of the Azure Data Lake Studio Gen 2.

Parquet File Format:

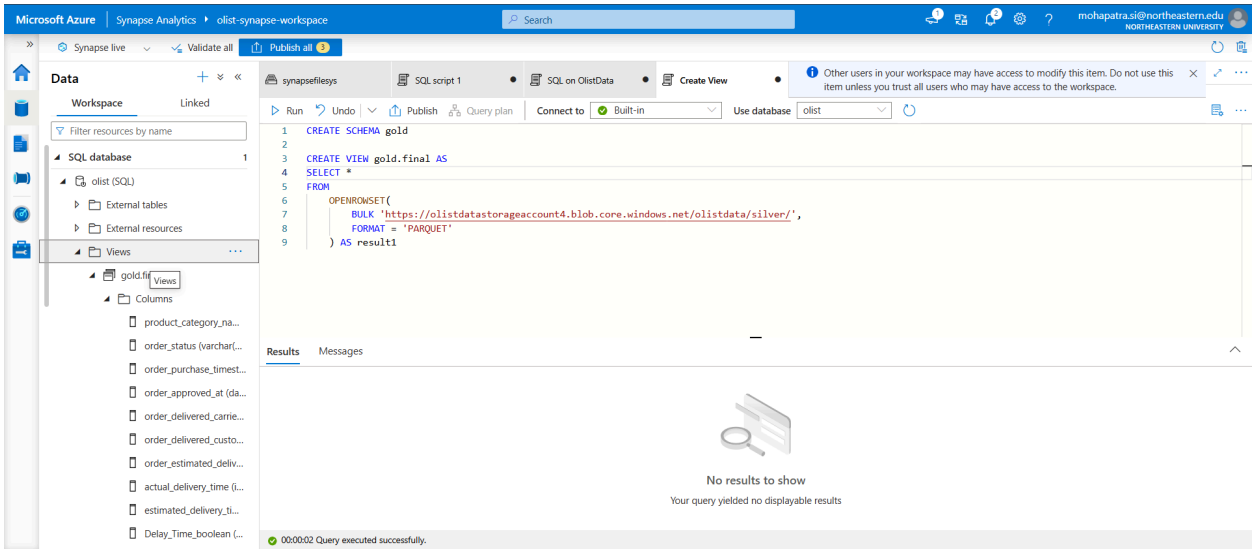
Parquet is a **columnar storage format** designed for efficient storage and processing of large datasets in distributed environments like **Azure Synapse, Databricks, Hadoop, and Spark**. Unlike traditional row-based formats (e.g., CSV), Parquet stores data **column by column**, allowing queries to read only the specific columns they need. This structure significantly reduces I/O operations, speeds up query performance, and minimizes storage space. It's especially suited for **analytical workloads**, where large datasets are scanned but only a few columns are used.

Snappy Compression Algorithm:

Snappy is a **fast data compression and decompression algorithm** developed by Google. Its primary goal is **speed over maximum compression ratio** — it compresses data efficiently enough to save space while ensuring very fast read/write performance. When combined with Parquet, Snappy makes file storage both **compact and quick to process**, which is crucial for big data systems that handle terabytes of information daily. Remember **OPENROWSET** is like a temporary connector between SQL tables and it is **Microsoft Specific**. But, it is only applicable **Azure Synapse Analytics** & **Microsft SQL Server**.



Step 5:



Here, I have created a view to store the data (that I retrieved from the **SILVER** layer) I created the View from the Silver layer because I think it's easy to fetch and see data from the SQL editor itself I mean Azure Synapse Analytics.

Step 5: Now, I am going to send those view's data to **GOLD** layer so that it can be used for serving. So, I will be creating the physical tables will be the one which anyone can now read from my data lake and this will be stored inside the **GOLD** layer. So, here we will use **CETAS** command in SQL (**Create External Table as Select)** (https://learn.microsoft.com/en-us/azure/synapse-analytics/sql/develop-tables-cetas)

SQL Copy

```
CREATE DATABASE [<mydatabase>];
GO

USE [<mydatabase>];
GO

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<strong password>';

CREATE DATABASE SCOPED CREDENTIAL [WorkspaceIdentity] WITH IDENTITY = 'Managed Identity';
GO

CREATE EXTERNAL FILE FORMAT [ParquetFF] WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
);
GO

CREATE EXTERNAL DATA SOURCE [SynapseSQLwriteable] WITH (
    LOCATION = 'https://<mystoageaccount>.dfs.core.windows.net/<mycontainer>/<mybaseoutputfold
    CREDENTIAL = [WorkspaceIdentity]
);
GO

CREATE EXTERNAL TABLE [dbo].[<myexternaltable>] WITH (
        LOCATION = '<myoutputsubfolder>/',
        DATA_SOURCE = [SynapseSQLwriteable],
        FILE_FORMAT = [ParquetFF]
) AS
SELECT * FROM [<myview>];
GO
```

AZURE SYNAPSE WORKFLOW:

1. Medallion Architecture and Lake House. (Pre-requisite)
2. Creating a Serverless SQL Pool. (For just reading & querying)
3. Creating a schema & then user/password

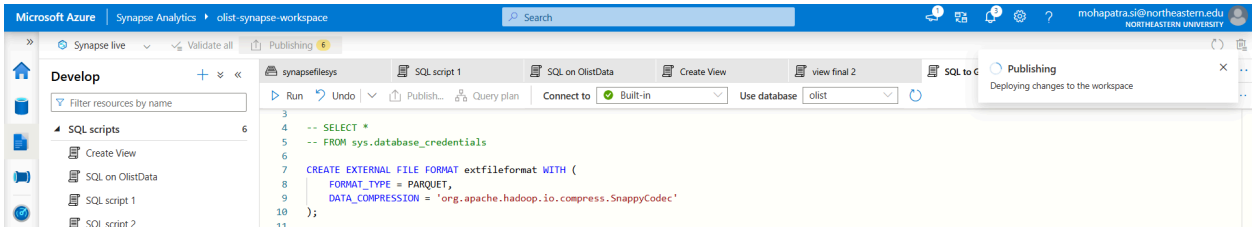
```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<strong password>';

CREATE DATABASE SCOPED CREDENTIAL [WorkspaceIdentity] WITH IDENTITY = 'Managed Identity';
```

.

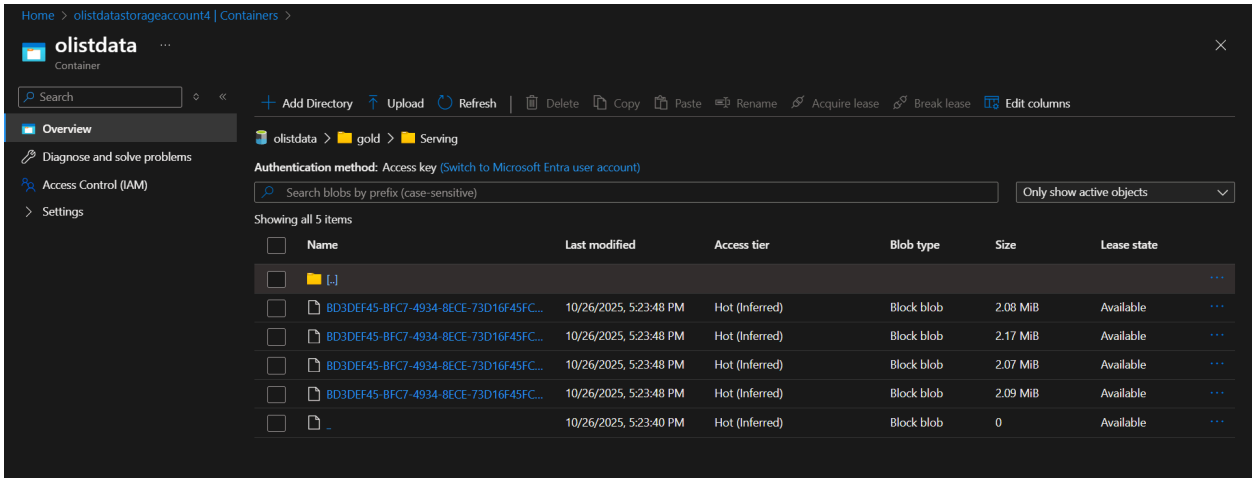
4. Using OPENROWSET to read data from the **SILVER** layer. (Here, data is not stored, just referenced to **SILVER** layer)
5. Creating a **VIEW** to easily query the data.

Now, I will be making sure, I am able to create external physical tables which will be stored in the **GOLD** layer which can be used further by Data Analysts and Data Scientists.

```
CREATE EXTERNAL FILE FORMAT [ParquetFF] WITH (
   FORMAT_TYPE = PARQUET,
   DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
);

CREATE EXTERNAL DATA SOURCE [SynapseSQLwriteable] WITH (
   LOCATION =
'https://<mystoageaccount>.dfs.core.windows.net/<mycontainer>/<mybaseoutputfolderpath>',
   CREDENTIAL = [WorkspaceIdentity]
);
```



After this, I published my final data to **Gold** Layer in the Serving Folder:



—----------

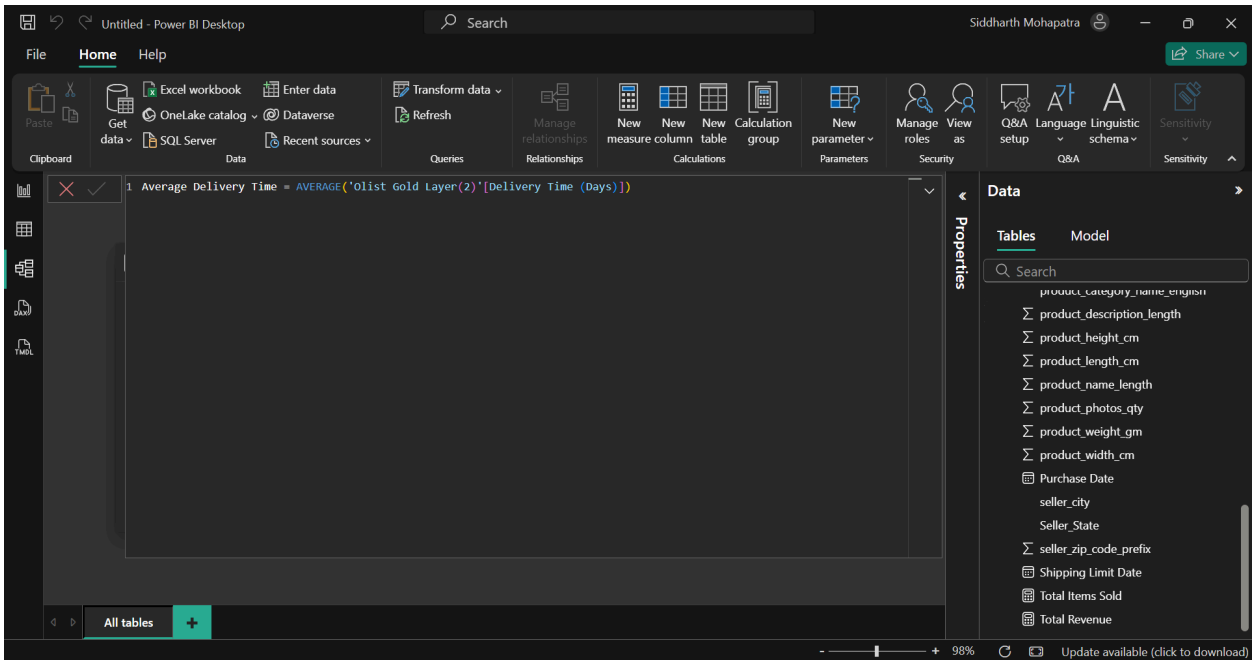
Final Cleaning of Gold Layer Data before dashboarding in Power BI

- Removed duplicate rows
- Removed null value - rows in **product category name** column
- Collected a State - State code data of Brazilian states and used a **LOOKUP** function to replace all the customer and seller state codes with actual full state names for better understanding for the users and audience

POWER BI DASHBOARD GOAL:

Now, I will show how my **Azure Medallion architecture** (Bronze → Silver → Gold) powers real-time insights on e-commerce performance, focusing on **customer behavior**, **sales performance**, and **delivery efficiency**.

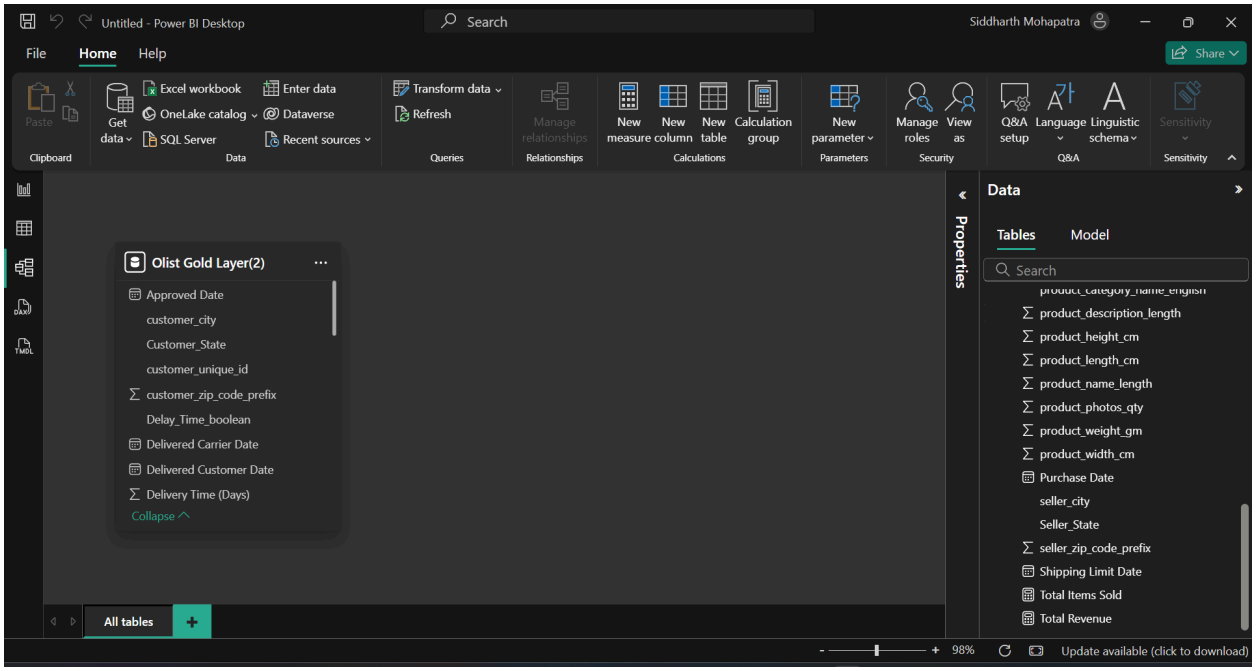
POWER BI INTERFACE



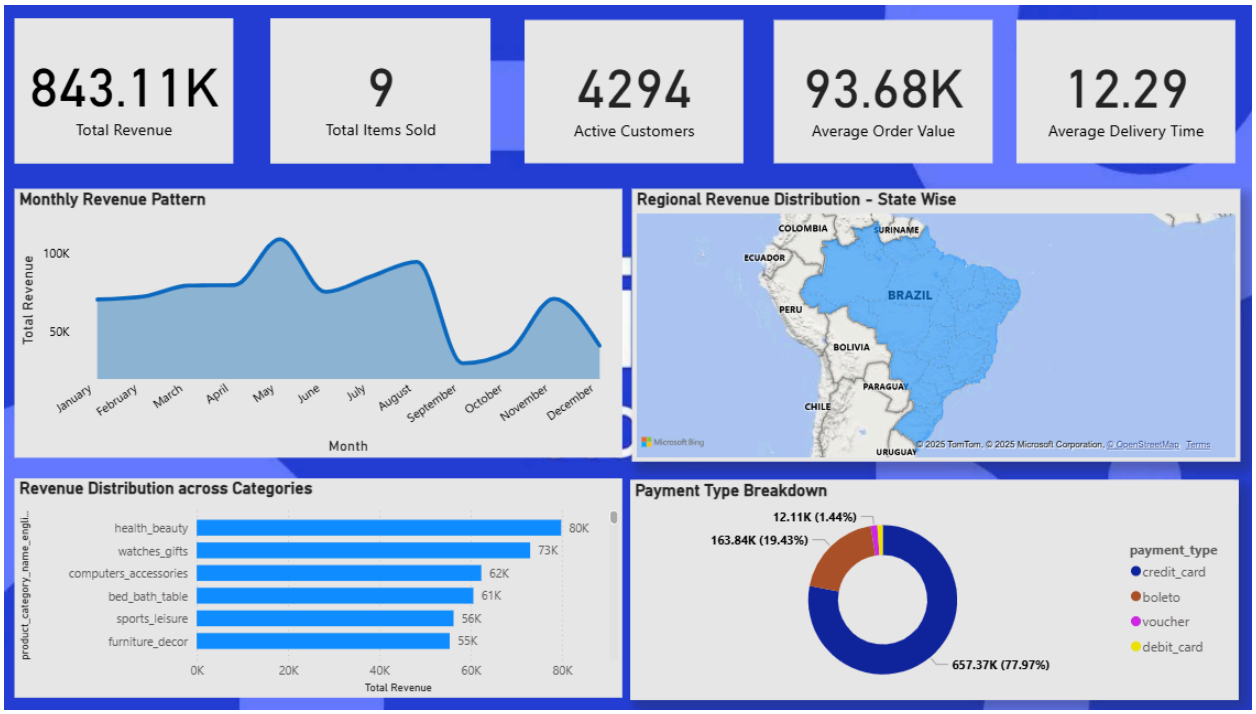
DAX (Data Analytics Expressions) for KPIs

- Total Revenue = SUM('Olist Gold Layer(2)'[payment_value (Brazilian Real)])
- Total Items Sold = DISTINCTCOUNT('Olist Gold Layer(2)'[order_item_id])
- Active Customers = DISTINCTCOUNT('Olist Gold Layer(2)'[customer_unique_id])
- Average Order Value = DIVIDE([Total Revenue], [Total Items Sold])

- Average Delivery Time = AVERAGE('Olist Gold Layer(2)'[Delivery Time (Days)])



MY POWER BI DASHBOARD:



This dashboard provides a high-level view of Olist's e-commerce performance across Brazil — showing revenue trends, regional distribution, product performance, and customer insights. It helps management and analysts monitor performance, identify growth opportunities, and improve operational efficiency.

1️⃣ KPI: Total Revenue (843.11K BRL)

- **What it shows:** The total value of all payments received across all orders.
- **Why it matters:** This is the most direct measure of overall business performance and sales growth.
- **What we infer:** Revenue is healthy and concentrated in a few regions and categories. Any significant monthly drop could indicate seasonal trends or logistics issues.
- **Improvement:** Drill down by region or product category to find where sales are underperforming and launch promotions there.

2️⃣ KPI: Total Items Sold (9)

- **What it shows:** Number of product categories or unique SKUs (Stock Keeping Unit) sold.
- **Why it matters:** Reflects product variety and inventory turnover.
- **What we infer:** Olist focuses on a smaller curated set of products, which simplifies supply chain but limits category diversification.
- **Improvement:** Expand high-demand categories or add complementary products to increase basket size.

3️⃣ KPI: Active Customers (4,294)

- **What it shows:** Total number of unique customers who made at least one purchase.
- **Why it matters:** Indicates customer base strength and market reach.
- **What we infer:** The platform has a good number of active buyers, but retention trends can be explored further.
- **Improvement:** Build a loyalty or referral program to retain customers and boost repeat purchases.

4️⃣ KPI: Average Order Value (93.68K BRL)

- **What it shows:** Average amount spent per customer order.

- **Why it matters:** Higher AOV means customers are buying more expensive or multiple items per transaction.
- **What we infer:** The pricing strategy or bundling seems effective.
- **Improvement:** Introduce cross-selling or up-selling strategies to lift AOV further.

5️⃣ KPI: Average Delivery Time (12.29 Days)

- **What it shows:** Average duration between order placement and delivery completion.
- **Why it matters:** Affects customer satisfaction and repeat business.
- **What we infer:** Delivery speed is moderate; some regions likely experience delays.
- **Improvement:** Optimize last-mile logistics and vendor coordination to reduce delivery time.

6️⃣ Chart: Monthly Revenue Pattern (Line Chart)

- **What it shows:** Revenue trends from January to December.
- **Why it matters:** Helps identify seasonality, peak months, and slow periods.
- **What we infer:** Revenue spikes mid-year (May–August), possibly due to sales events or festivals, and dips toward year-end.
- **Improvement:** Introduce marketing campaigns during low-sales months and increase inventory before peak months.

7️⃣ Chart: Regional Revenue Distribution (Map Visual)

- **What it shows:** State-wise contribution to total revenue across Brazil.
- **Why it matters:** Helps pinpoint high-performing and low-performing regions.
- **What we infer:** Most sales are concentrated in southeastern Brazil (SP, RJ), showing strong urban demand.
- **Improvement:** Expand operations and seller partnerships in underperforming states like northern or northeastern Brazil.

8️⃣ Chart: Revenue Distribution Across Categories (Bar Chart)

- **What it shows:** Top-performing product categories by total revenue.
- **Why it matters:** Guides inventory planning, marketing, and product focus.
- **What we infer:** *Health & Beauty* and *Watches & Gifts* generate the most revenue, suggesting strong lifestyle-driven demand.
- **Improvement:** Increase marketing for high-margin categories and run discount campaigns for weaker categories like *Furniture Décor*.

9️⃣ Chart: Payment Type Breakdown (Donut Chart)

- **What it shows:** Share of different payment modes (Credit Card, Boleto, Voucher, Debit Card).
- **Why it matters:** Reveals customer payment preferences and financial inclusivity.
- **What we infer:** Over 75% of transactions are made via credit cards — customers rely heavily on credit-based purchases.
- **Improvement:** Offer more flexible payment options (e.g., UPI, installment plans) to cater to different customer segments.

🧩 Key Takeaways

- **Top Categories:** Health & Beauty and Watches & Gifts are the main revenue drivers.
- **Geographic Focus:** São Paulo and Rio de Janeiro dominate the market share.
- **Customer Experience:** Improving delivery time could increase satisfaction and reduce churn.
- **Strategic Focus:** Marketing and logistics optimization in lower-performing regions and months will balance overall revenue.

Visual / KPI	Purpose	Insight Derived	Business Impact	Improvement Suggestions

Total Revenue	Measure total payments received from all transactions	Indicates overall business performance and sales health	Reveals financial strength and helps evaluate marketing or pricing effectiveness	Drill down by region and product to identify underperforming areas
Total Items Sold	Shows total unique products/categories sold	Reflects product diversity and SKU performance	Helps in inventory planning and demand forecasting	Add more products in high-demand segments to increase variety and sales
Active Customers	Tracks the number of unique customers who purchased	Indicates customer base size and engagement	Helps identify customer growth or retention rate	Launch loyalty/referral programs to boost repeat customers
Average Order Value (AOV)	Measures average spend per order	High AOV means effective pricing and cross-selling	Correlates with profitability and marketing effectiveness	Encourage product bundling and targeted promotions to lift AOV
Average Delivery Time	Tracks mean time from order to delivery	Measures efficiency of logistics and fulfillment	Impacts customer satisfaction and brand trust	Optimize delivery routes and improve warehouse coordination
Monthly Revenue Pattern (Line Chart)	Visualizes monthly revenue trend	Shows seasonality — peaks and dips in sales	Aids budgeting and forecasting	Introduce offers in low-sales months and prepare inventory for peaks
Regional Revenue Distribution (Map)	Highlights state-wise revenue performance	SE region (SP, RJ) dominates; others underperform	Guides regional marketing and logistics	Expand seller network and delivery centers in low-performing states
Revenue by Category (Bar Chart)	Compares total revenue across product categories	Health & Beauty and Watches lead in revenue	Helps focus investment on high-performing categories	Rebalance stock and create campaigns for slow-selling items

Payment Type Breakdown (Donut Chart)	Shows customer payment preferences	78% of users use credit cards; few use vouchers/debit	Helps tailor financial partnerships and promotions	Introduce flexible payment options (installments, wallets)

STRATEGIC INSIGHTS & RECOMMENDATION

Insight	Business Interpretation	Recommended Action
Revenue is heavily concentrated in SE Brazil (SP, RJ)	**Indicates high dependency on a few markets**	**Expand seller onboarding and delivery infrastructure in underpenetrated states (North & Northeast)**
Health & Beauty and Watches & Gifts dominate revenue	**Product mix is skewed toward lifestyle categories**	**Diversify inventory — promote tech/home products through discounts or cross-sell campaigns**
Average Delivery Time is relatively high (12.29 days)	**Slower deliveries may hurt customer satisfaction**	**Partner with regional carriers or introduce local warehouses for faster fulfillment**
Majority of payments are via credit cards (~78%)	**High credit dependency may limit financially constrained customers**	**Add more payment options (Pix, digital wallets, UPI equivalents) to expand reach**
Revenue dips in certain months	**Possible seasonality or supply issues**	**Use predictive analytics to forecast demand and plan marketing campaigns during off-peak months**

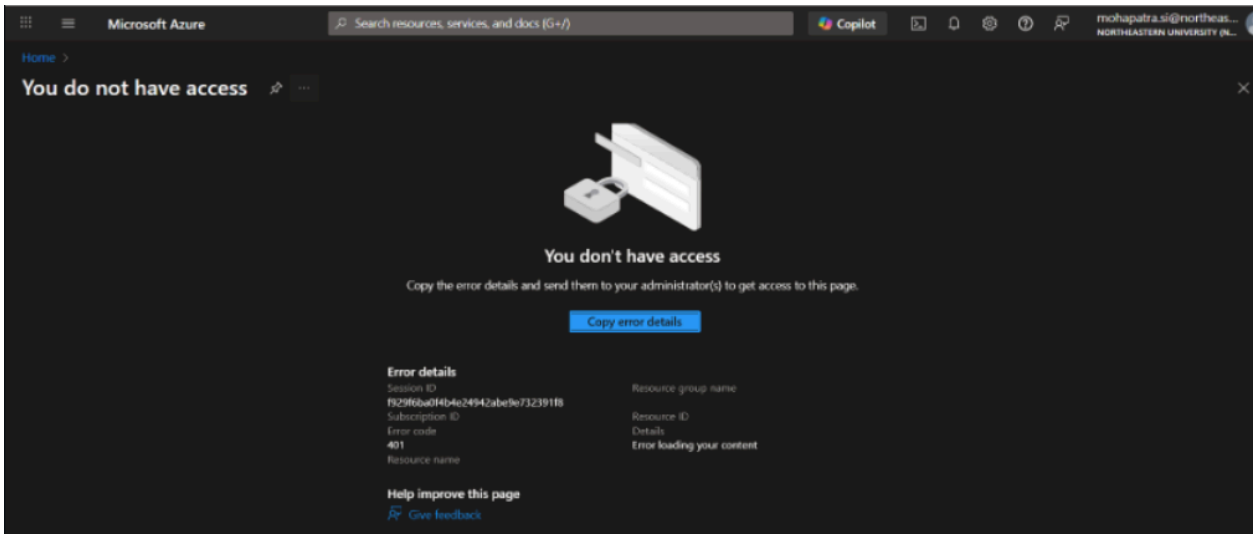
—----------

SOME OF THE MAJOR & MINOR CHALLENGES I HAVE FACED DURING THE PROJECT

 # Challenges Faced

1. Built a separate pipeline for olist_geolocation from PostgreSQL to Azure bronze layer (ADLS Gen2) because of the timeout issue due to the large volume of data (around 10 lakhs records).
2. Used Self Hosted Integrated Runtime (SHIR), which acts as a gateway from the Local PostgreSQL server to the Azure cloud, as local to cloud is not possible without any gateway.
3. Used the Publish All button for permanent save otherwise I would have taken around 2-3 weeks more to start building the pipeline from scratch, which took me a lot of time.

4. To make the connection, we need some key or permission here to maintain the equality. Both are Microsoft Azure services, so they are connected within Microsoft, and no third party can access them. So, here in this case when I tried to connect Azure Databricks and ADLS, I got this error.



What This Error Means
The error "You do not have access" with Error code: 401 means "Unauthorized." It's a permissions issue. I was successfully logged into Azure, but my specific account (mohapatra.si@northeastern.edu) does not have the administrative rights to view or manage the "App registrations" section.

Why I didn’t Have Access (The University Analogy)
Think of your university's entire Microsoft Azure setup as a large office building.

The Building (Azure Active Directory/Tenant): The entire building is managed by your university's IT administrators. They control the main entrance, the security office, and who gets what keys. The "App registrations" page is like the building's main security office. It's a central, high-level area.

Your Office (Your Azure for Students Subscription): The university has given you your own private office inside the building—this is your "Azure for Students" subscription. Inside your office, you are the owner. You can set up furniture (like Azure Data Factory, Databricks, etc.) and do whatever you want.

The error we are seeing is because our keycard (our student account) lets us into our own office, but it doesn't grant us access to the main security office for the entire building. This is a security measure to prevent students from viewing or changing administrative settings that affect the whole university.

However, the tutorial I was following demonstrated the most common and robust method for production environments: creating a Service Principal. A Service Principal is like creating a robot user with its own specific permissions. However, as you discovered, creating these "robot users" requires high-level administrative access to the "security office" (App Registrations), which I don't have with a student account.

There is another, simpler method that is perfect for this situation and still keeps the connection secure within the Microsoft cloud. I will use a method called mounting with an Account Key.

Instead of creating a new "robot user," you will temporarily use your own powerful key (the Storage Account Access Key) to establish a permanent link (a mount point) between Databricks and your Data Lake.

What I did here?
Think of it like giving your workshop (Databricks) a secure key to your storage unit (ADLS).

We got the Master Key: I went to my Azure storage account and copied its Access Key (key1). This key is like a master password that grants full access to all the data inside the service.

We Used a Secure Lockbox: Instead of pasting this sensitive key directly into your code (which is insecure), we created a Databricks Widget outside of the cell. This widget is like a secure lockbox at the top of my notebook. I pasted the key into this box, keeping it separate from my saved code.

We Built a Secure Bridge: Finally, I ran the dbutils.fs.mount() command. This command did two things:

It securely took the key from the lockbox.
It used the key to build a permanent, direct link—a mount point—from Databricks to my storage container.

The result is that my olistdata container now appears as a simple folder inside Databricks at the path /mnt/olistdata. Anyone can now read and write files to our storage as if it were a local directory.

5. Why do Synapse deployments fail?

Think of a Synapse workspace like opening a new mall. We're not just creating the mall—we also need parking (SQL server), security (managed identity/permissions), roads (networking), and a storage warehouse (Data Lake). If any of those can't be created in the location you selected, the entire "open the mall" action fails.

One reason for this can be:

Region blocks / capacity limits (my exact error)

Message: SqlServerRegionDoesNotAllowProvisioning … Location 'eastus' is not accepting creation of new … SQL Database servers…

What it means: In that region (e.g., East US), my subscription isn't allowed to create new Azure SQL Server resources right now—often due to capacity or subscription-type limits (e.g., Free Trial/Azure for Students).

Real example: We pick East US for Synapse, but Synapse needs to spin up a SQL server there; the region says "no new SQL servers allowed," so deployment fails.

Instead we can pick another region (e.g., East US 2, Central US, West US 2) to solve this, which is what happened in my case.

Dataset Link: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce