Creating ETL pipelines in Microsoft Azure for streaming data

Table of Contents

1 Introduction 2

1.1 Architecture 2

2 Implementation 3

2.1 Download MYSQL docker Image and run the image 3

2.2 Loading the adventure database in MYSQL 4

3 MYSQL to Python 5

3.1 Reading MYSQL data in python 5

4 KAFKA for streaming data 5

4.1 Configuring Confluent Kafka 6

4.2 Messages in kafka topic 7

5 Processing data in Azure Databricks 8

5.1 Configuring Azure Databricks. 8

5.2 Mounting data bricks to azure blob storage 9

6 Azure blob storage 10

7 Create an Azure Data warehouse Database 10

7.1 Creating a table in database 11

8 Azure Data factory 12

8.1 Creating dataset from blob 12

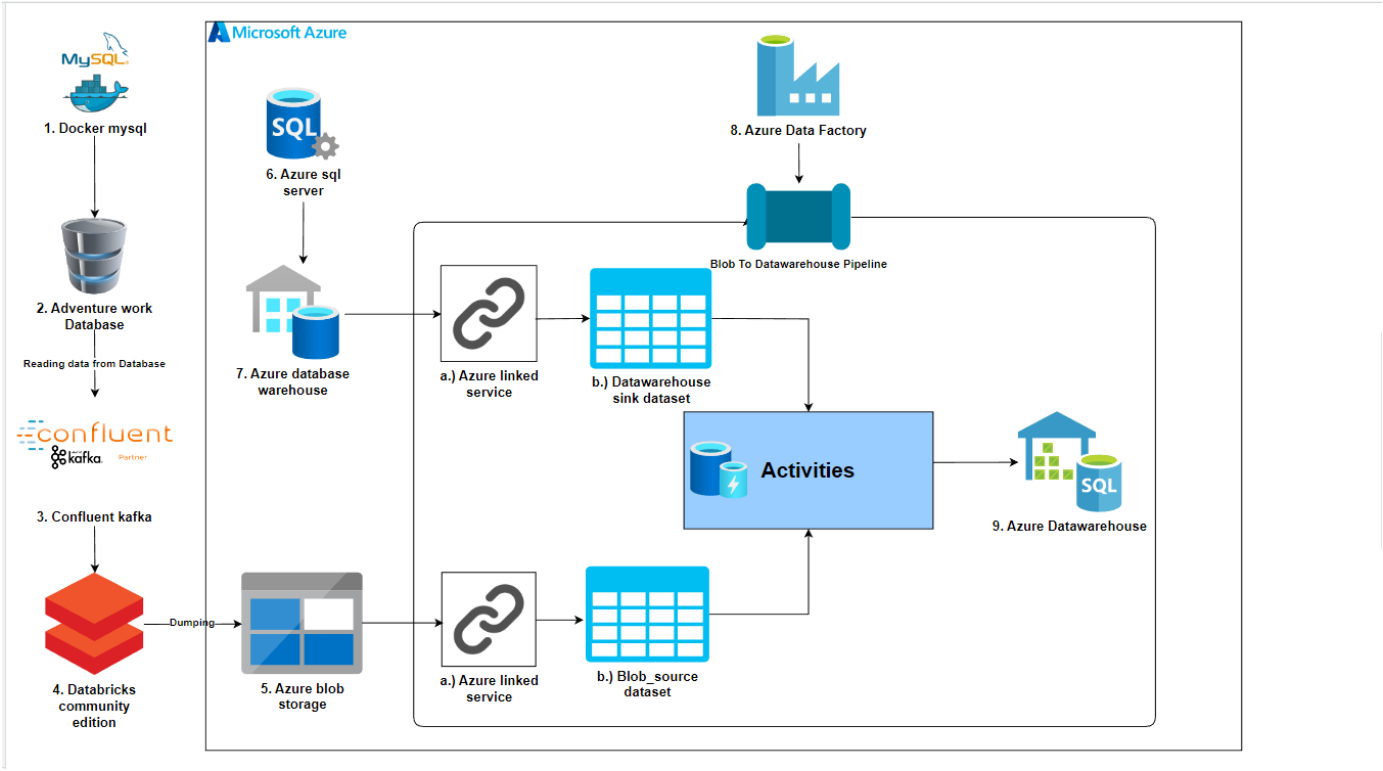
8.2 Building pipeline to dump data in Azure SQL 13

9 Checking data in Azure SQL 15

1 Introduction

The Aim of the project is to perform ETL Pipeline in Microsoft Azure using streaming data

1.1 Architecture



The above picture illustrates the architecture of the project. The numbering of each step explains the sequence of steps performed in ETL pipeline.

Explanation:

The process starts with pulling the docker MYSQL image from the docker hub and loading the adventure work data into MYSQL database. Once the database is loaded, the adventure data is published into confluent KAFKA topics. The published data is consumed using Spark Streaming for further processing (for the sake of simplicity and flexibility, we will be using Azure data bricks for spark, since we don't want to focus on spark configuration, but once can also pull spark docker image based on the availability). The spark

streaming data is then saved in the Azure blob storage using the concept of mounting (i.e., connecting the azure blob storage to the databricks). Finally, we will use the azure data factory service, to create the pipeline And dump the streaming data in Azure SQL database.

2 Implementation

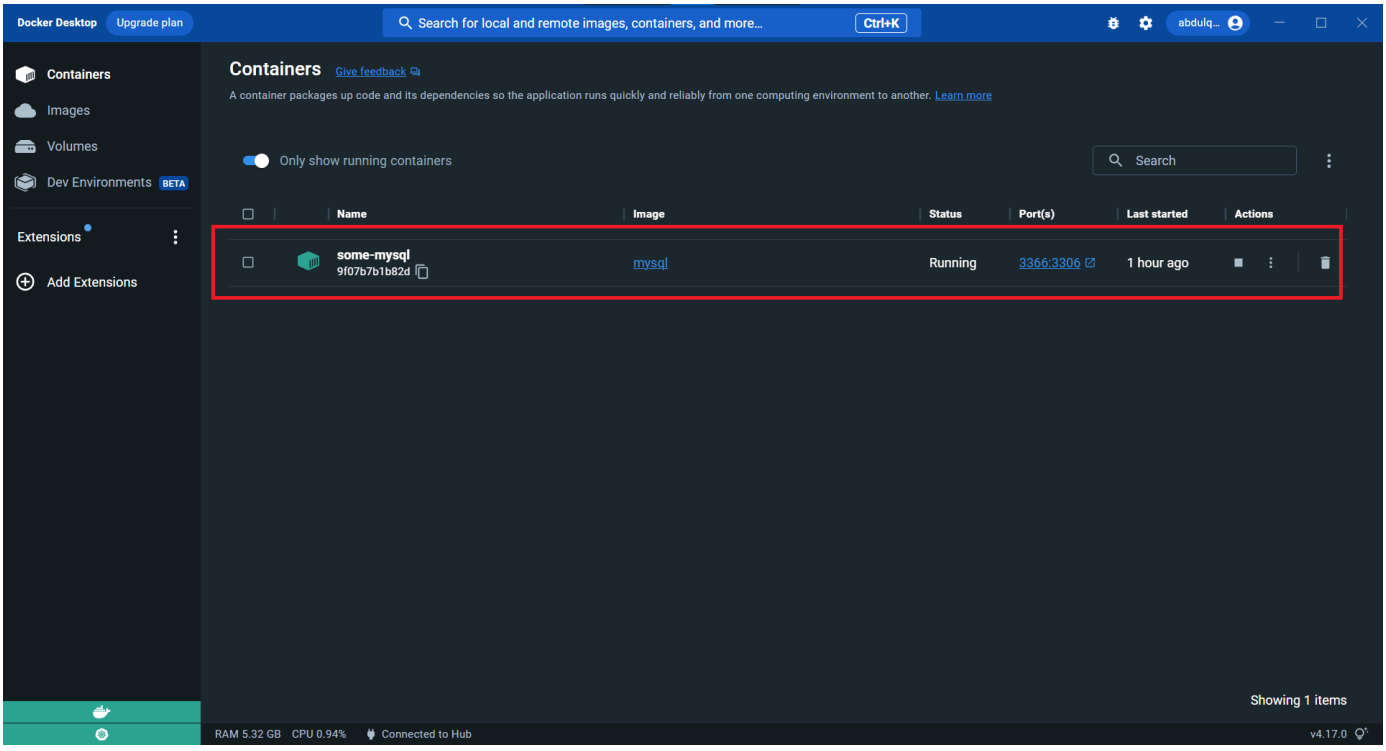
2.1 Download MYSQL docker Image and run the image

The implementation starts with pulling the docker MYSQL image from the docker hub.

docker run --name some-mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql

pasting the above line of code in terminal/command prompt and it will pull the MYSQL image from the docker hub.
The number “3306:3306” indicates the port numbers where the docker image runs (user can also give different port number (i.e., 7777:3306)). Note that, the user has to set the password in the password section (i.e., " **MYSQL_ROOT_PASSWORD =**") to communicate with database and by default, admin name is set to **root**.

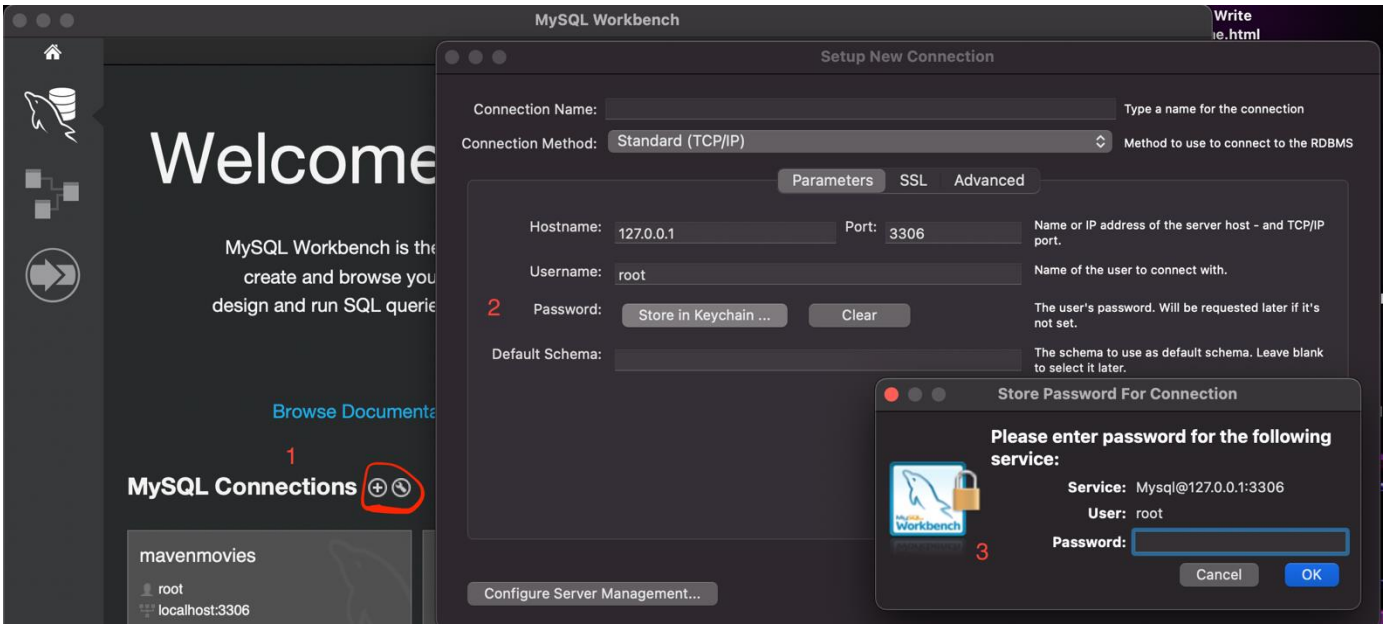
Once the image is download, run the docker MYSQL image. The below picture indicates the MYSQL image running in docker.



2.2 Loading the adventure database in MYSQL

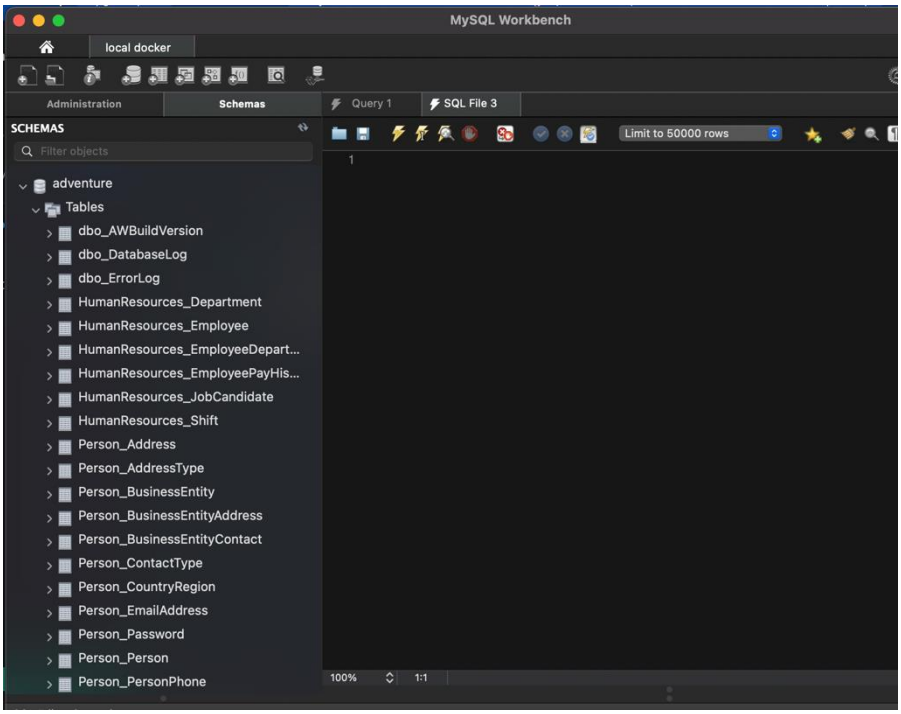
Open the workbench (we are using MYSQL work bench but one can use any workbench that connects with database)

Click on the plus sign and add the MYSQL credentials that were used while pulling the docker image. The below picture shows the sequence of steps using number ordering



Now load the Adventure work database in to the MYSQL workbench.
We are attaching the **AdventureWorks2019.sql** file in the folder and one can use this file directly load into MYSQL database.

Once the database is loaded, it should look something like in the picture below



3 MYSQL to Python

3.1 Reading MYSQL data in python

We will read MYSQL data in python and publish the data to Kafka topics.

We used the following dependency libraries to connect MYSQL with python

- Pandas
- Pymysql
- Sqlalchemy

The below code snippet allows connecting mysql database with python.

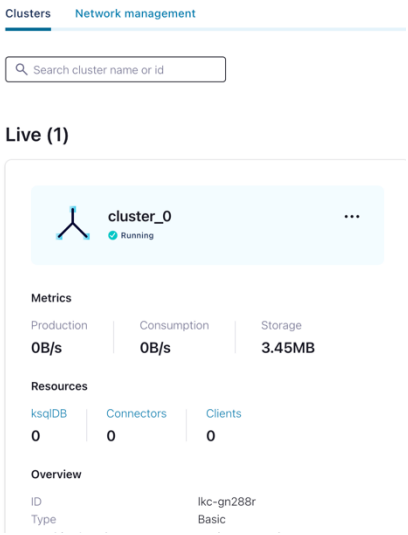
```
1. import pandas as pd
2. import pymysql
3. from sqlalchemy import create_engine
4. cnx = create_engine('mysql+pymysql://<username>:password@localhost:<portnumber>/databasename')
5. query = 'select * from HumanResources_Shift'
6. df = pd.read_sql(query, cnx)
```

note that, username and password should be replaced with MYSQL credentials (i.e., username and password that were used while pulling the docker image). The port number should be replaced port number that we used while pulling the docker image. The database name should be the name of the database that we created while importing the adventure data.

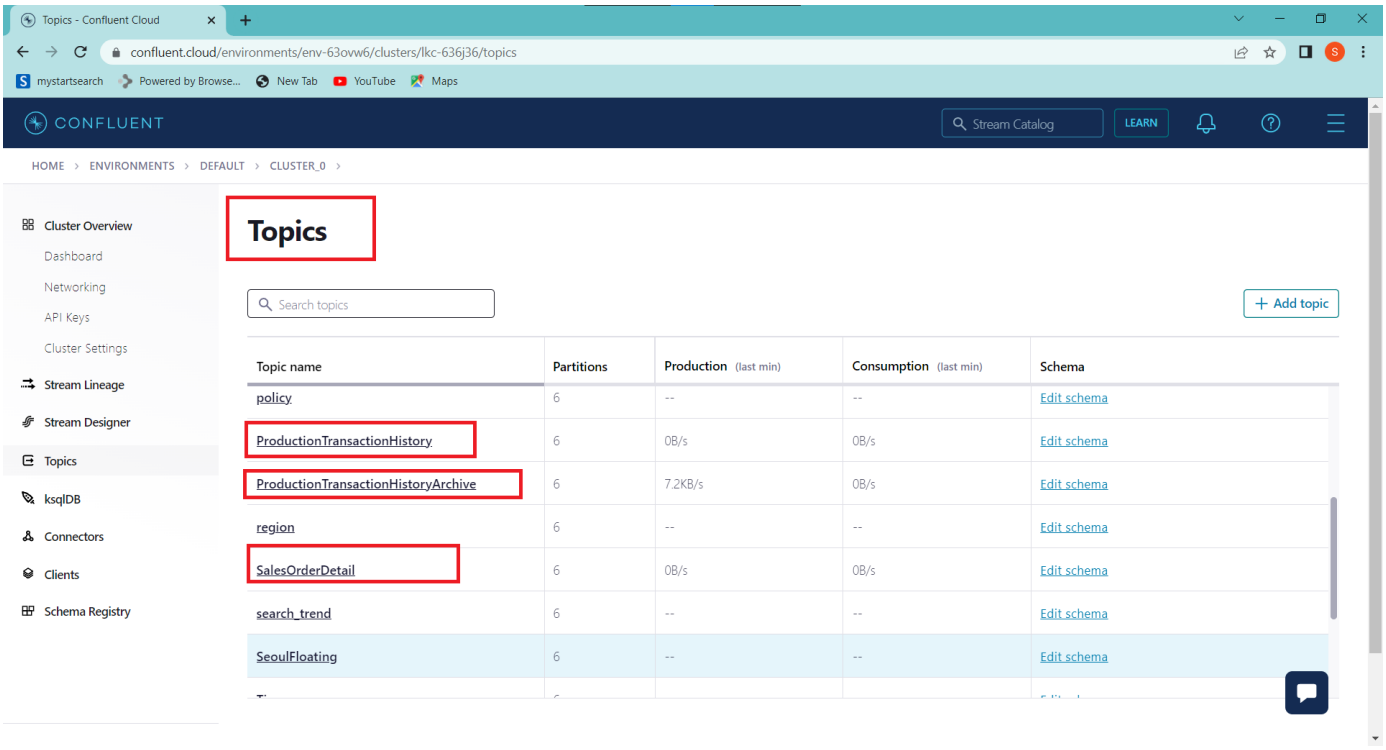
4 KAFKA for streaming data

4.1 Configuring Confluent Kafka

- Login to confluent Kafka and create a cluster. The picture below shows the cluster



- Create a Kafka topic and configure the schema. The picture below shows the topics created in Kafka



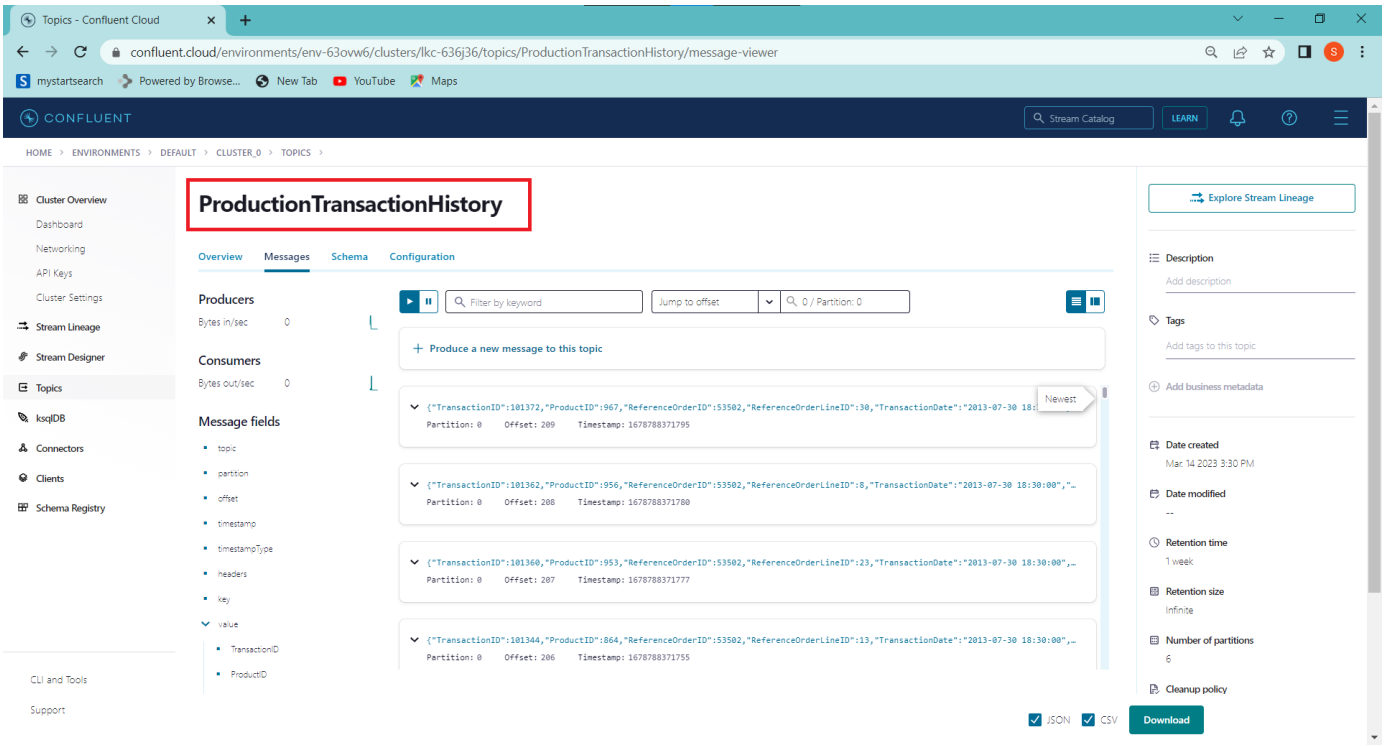
- Download the API keys, and Schema registry keys . Note that, we are attaching the **producer.py** file in the folder and this file is used to connect with Kafka cluster. one can replace or paste all the

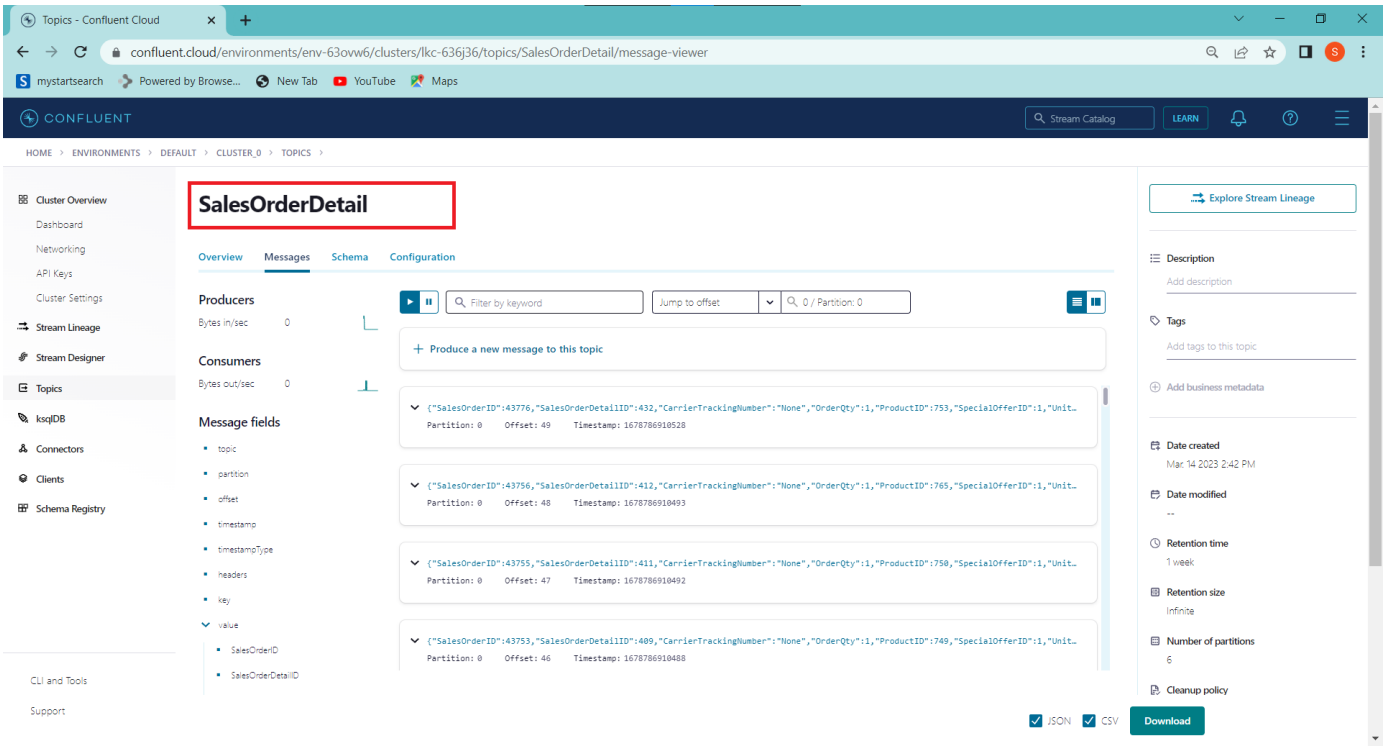
necessary keys in the producer.py, to connect to Kafka cluster and publish records to Kafka topic. Note that, we should also change the dataset schema in **producer.py** file before publishing the records to Kafka topic. the below pictures show the necessary keys to be downloaded to connect with confluent Kafka programmatically.

```
API_KEY = '4APZABSYMGAGJU5J'
ENDPOINT_SCHEMA_URL  = 'https://psrc-4nyjd.us-central1.gcp.conf
API_SECRET_KEY = 'S4Mpil9Wnlso8iyyqY+RY4YHUQHjSVyAId+pR4z3vpofT
BOOTSTRAP_SERVER = 'pkc-ymrq7.us-east-2.aws.confluent.cloud:909
SECURITY_PROTOCOL = 'SASL_SSL'
SSL_MACHENISM = 'PLAIN'
SCHEMA_REGISTRY_API_KEY = 'GO5XDKCD65YVGIRD'
SCHEMA_REGISTRY_API_SECRET = 'iz7yRGcygzQ4ZGSjEC25BLgTFTLcnkotm
```

4.2 Messages in Kafka topic

Once the messages are published to Kafka topic, we can conform by checking the messages in Kafka topic using offset numbers. The below picture shows the messages in Kafka.

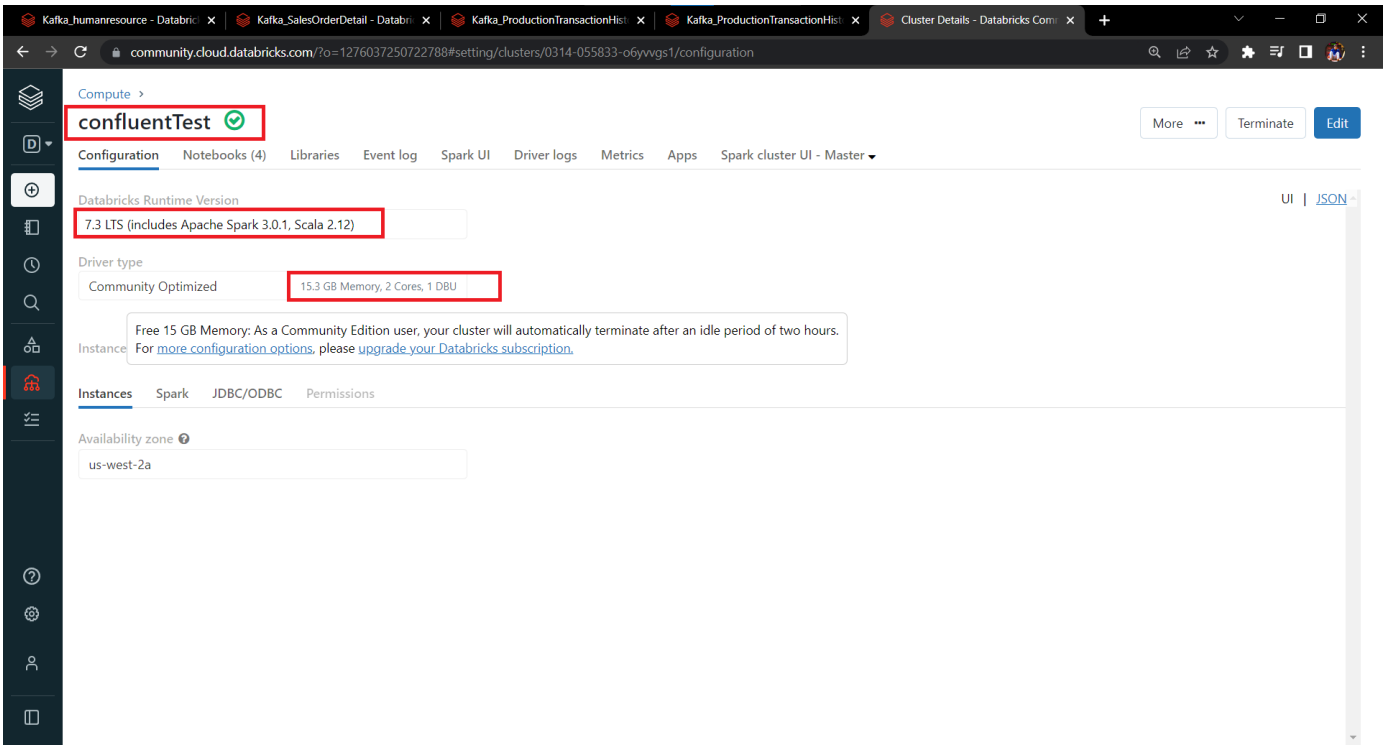




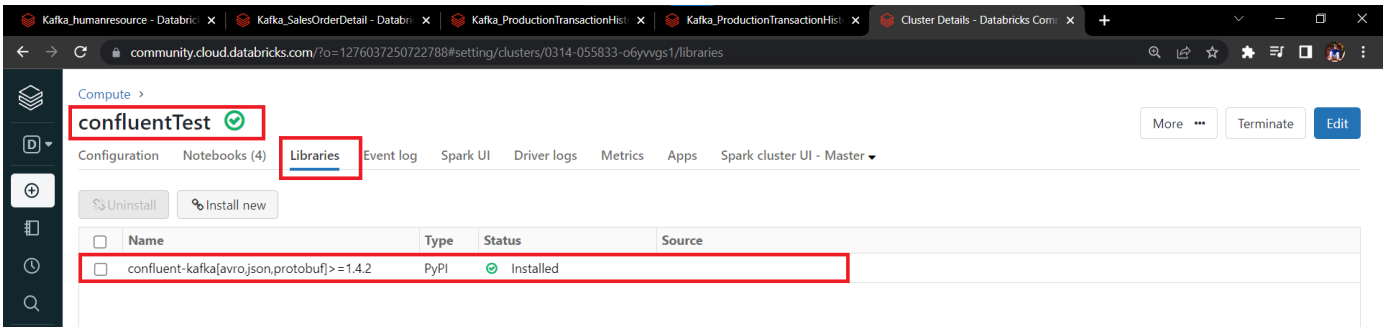
5 Processing data in Azure Databricks

5.1 Configuring Azure Databricks.

Login to Databricks using data bricks credentials (we have used data bricks community edition).
Create a cluster in data bricks. The below picture shows cluster in databricks



Now we need to import the confluent Kafka library, to read data from confluent Kafka.
Click on **Libraries,** and add **confluent-Kafka[avro,json,protobuf]>=1.4.2,** and install it.



5.2 Mounting data bricks to azure blob storage

Now we need to mount (i.e., creating a connection between services) the Azure blob storage to databricks to save the processed data directly in Azure blob storage.

The below code mounts the databricks with azure blob storage

```
# BLOB STORAGE INITIALIZATION
try:
    containe_name = "containername"
    storage_account_name = "storagename"
```

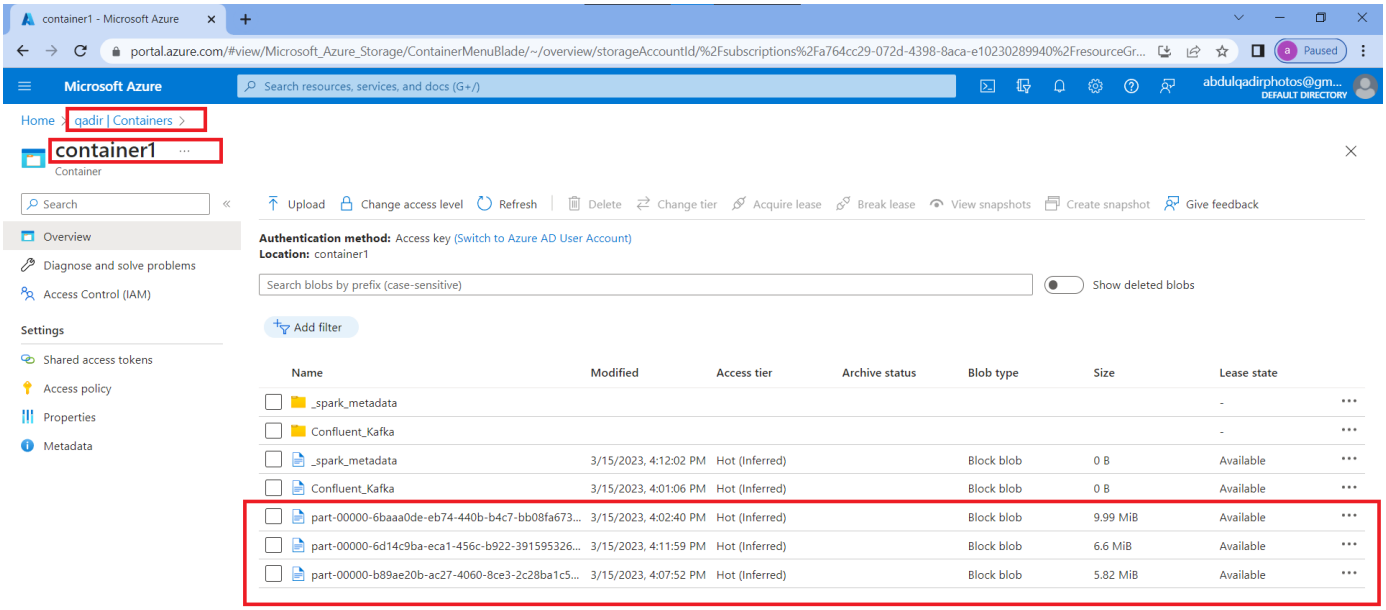
```
    accountkey = "Tp8pV6voD3lYkVja/ycqKLSBriDU1gwc0RFgedA7RNH68502aCEY7O4DjjkWC89tlGSbz6twUcFa+AStzNhR"
    config = "fs.azure.account.key.replace_storage_account_name.blob.core.windows.net"

    dbutils.fs.mount(
        source = "wasbs://{}@{}.blob.core.windows.net/".format(containe_name, storage_account_name),
        mount_point = "/mnt/blobstorage", #blobstorage is mount point name
        extra_configs = {config: accountkey}
    )

except Exception as e:
    print(e)
```

replace the container_name, storage_account_name and accountkey from the azure. This will mount the data bricks with azure blob storage

note that the **databricks notebook** is attached in the folder for the reference of spark code used for reading data from Kafka and processing it.

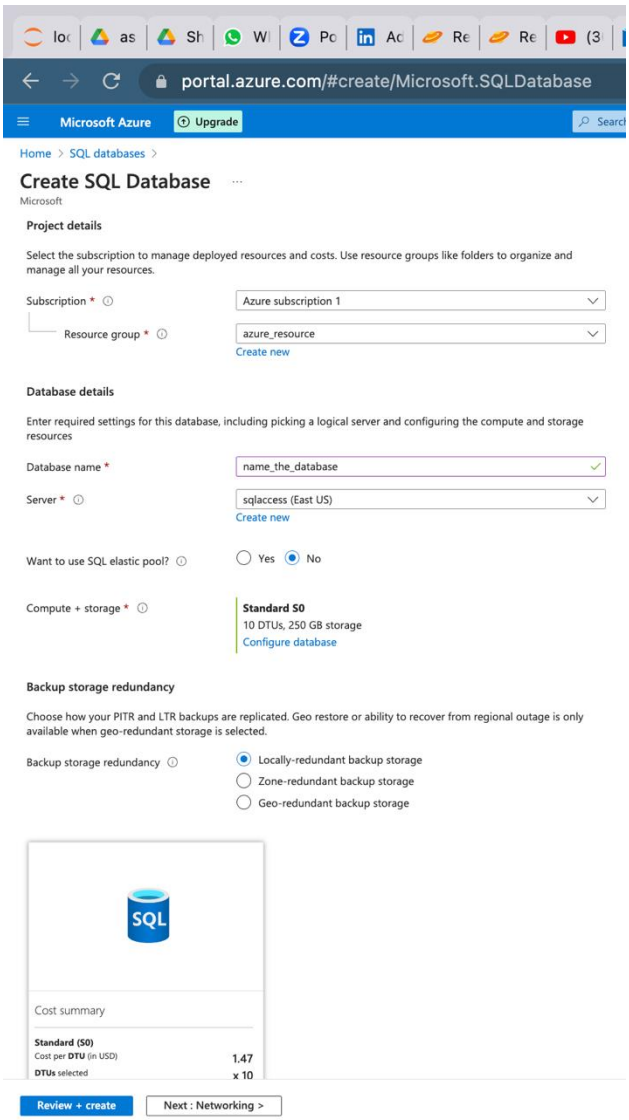
6 Azure blob storage

Once the processing is done, we should now able to see the processed data in blob storage.



7 Create an Azure Data warehouse Database

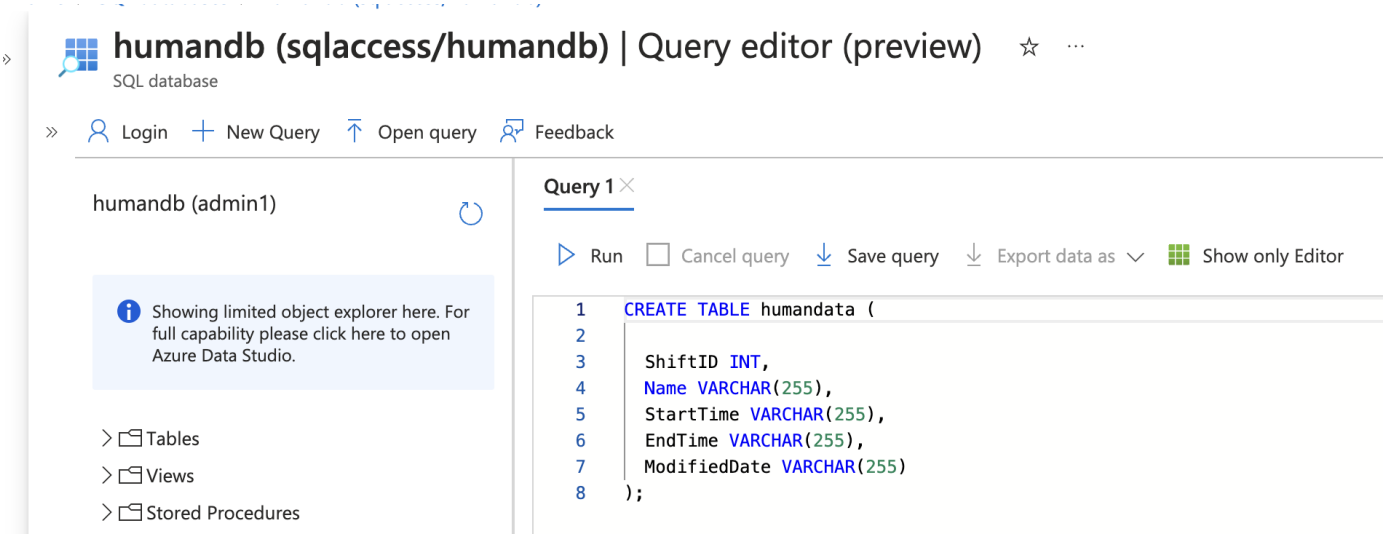
Create a database in azure sql and choose the database name.



Note that, we need to create a server to access the database. The creation of SQL server is not shown in the documentation.

7.1 Creating a table in database

Once the database is up and running, we need to create a table to dump the processed data.



8 Azure Data factory

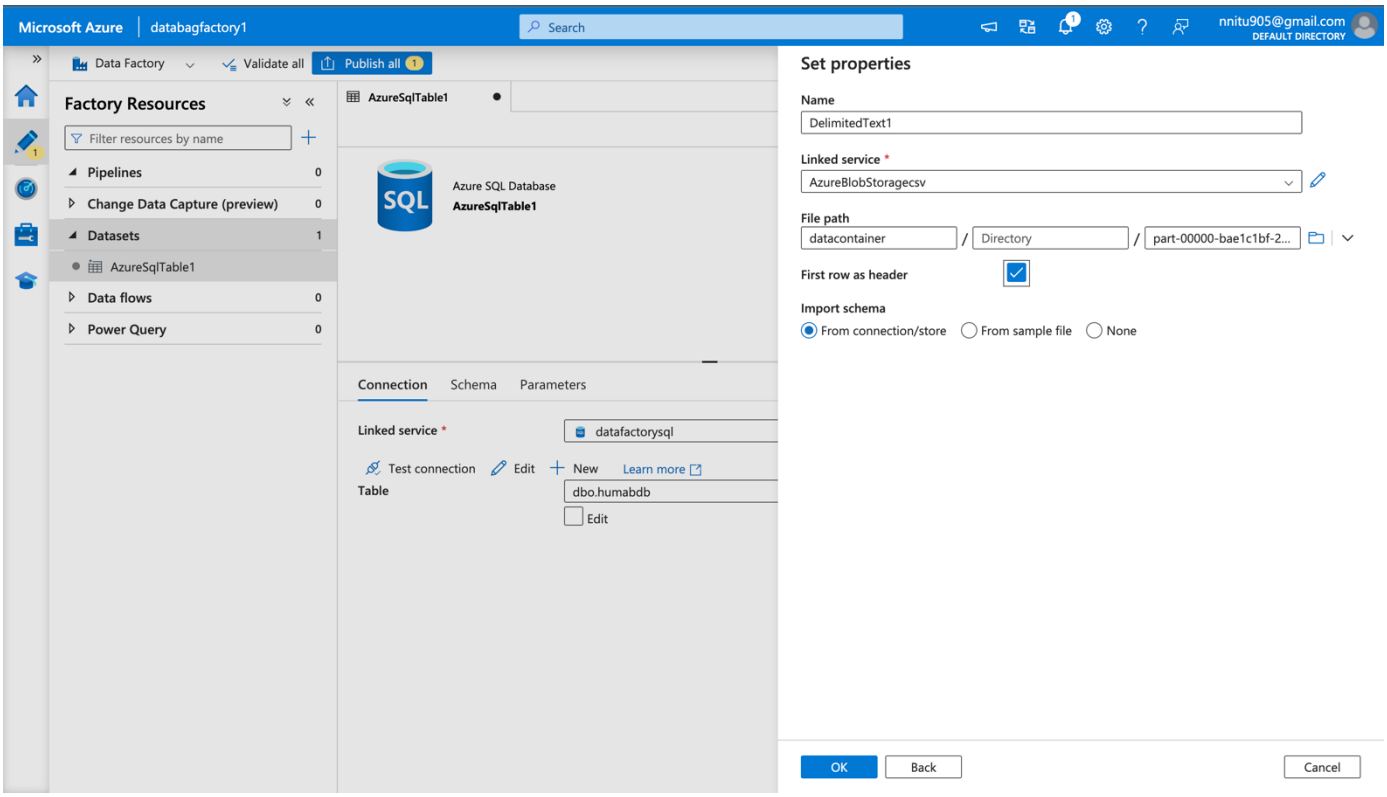
Aure blob storage ➔ (retrieving file schema) **Data Factory** (retrieving schema and testing connection)➔ **Azure SQL database**

8.1 Creating dataset from blob

We need azure data factory to build the data pipeline and dump from blob storage to Azure SQL

Navigate to the datafactory and create the dataset for Azure blob storage (i.e., we are retrieving the schema (metadata) of the file that was stored in blob storage). Give the container name and fine name with extention in the file path.

The below pictures show the image of creating the schema for blob storage.

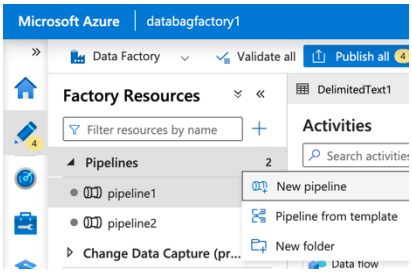


Once the dataset is created, we need to conform by checking the schema of the file.

8.2 Building pipeline to dump data in Azure SQL

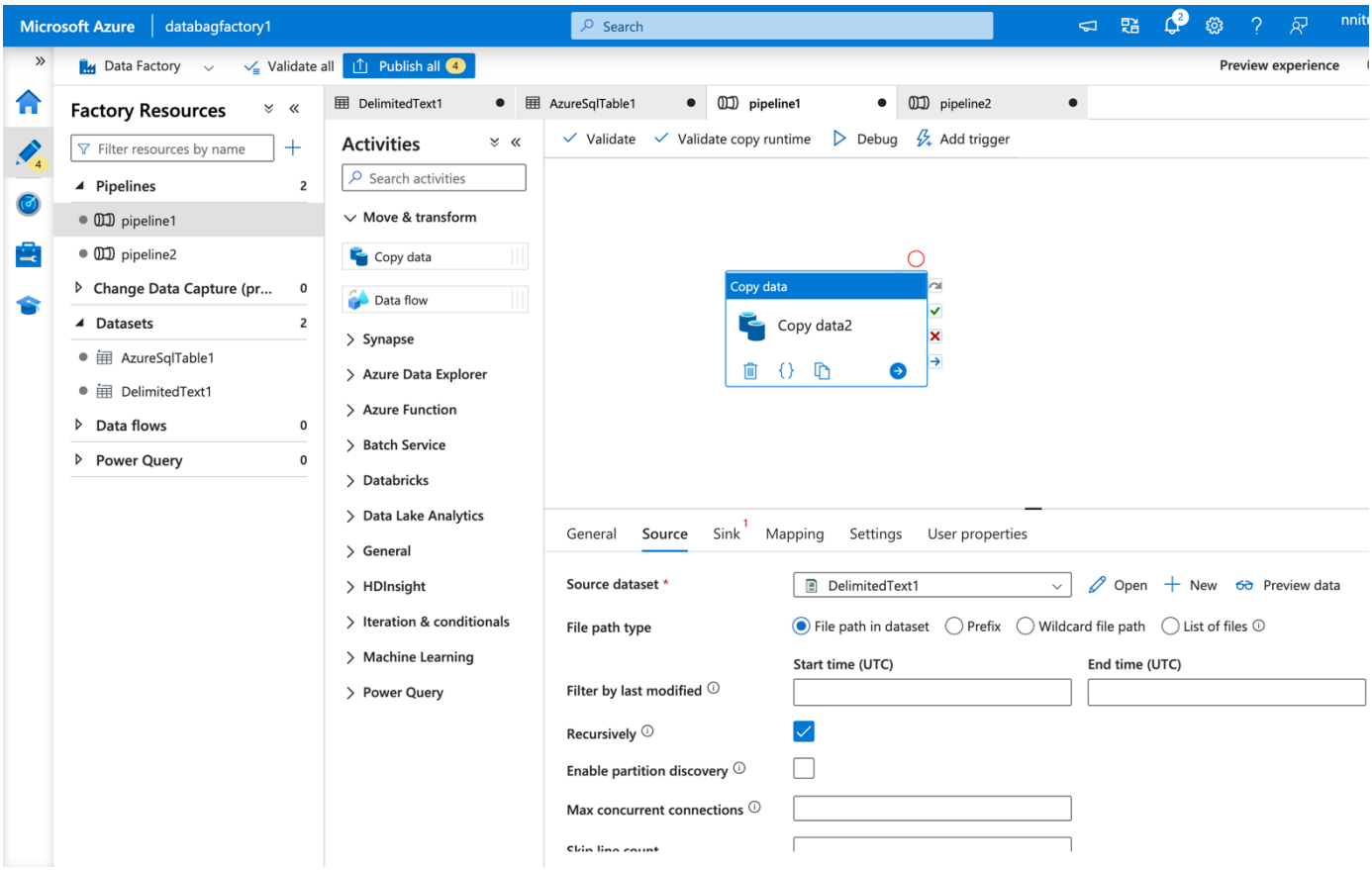
Once both the datasets are created, we need to build a pipeline

- Navigate to pilelines and click on new pipeline



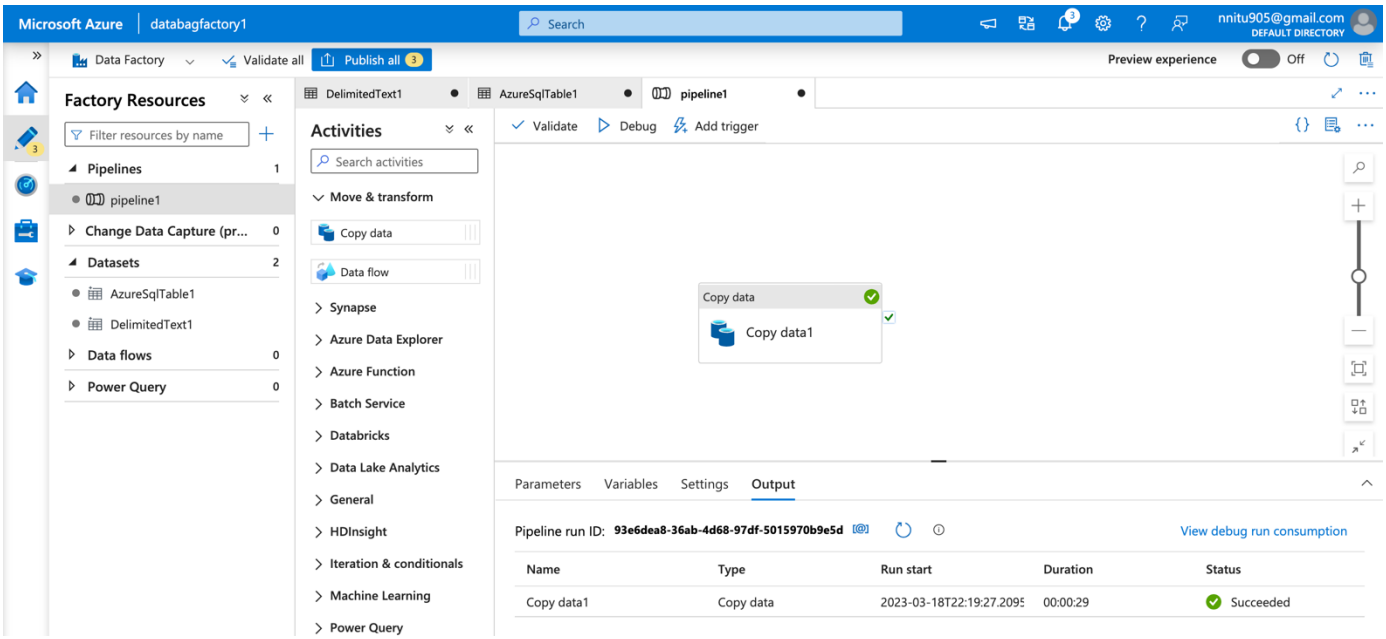
- Click on move & transfor from activities and choose copy data
- Select the dataset for source (i.e., choose the dataset which was created from blob storage)
- Click on sink and select the target dataset (i.e., select the dataset that was created using Azure SQL)

The below picture shows sample of selecting the source



- Once the source and sink is configured, click on debug to start the pipeline

The below picture shows the sample of successful dumping of data in Azure Sql



9 Checking data in Azure SQL

Once the data is dumped, navigate to the Azure sql and check the data by running the simple SELECT query