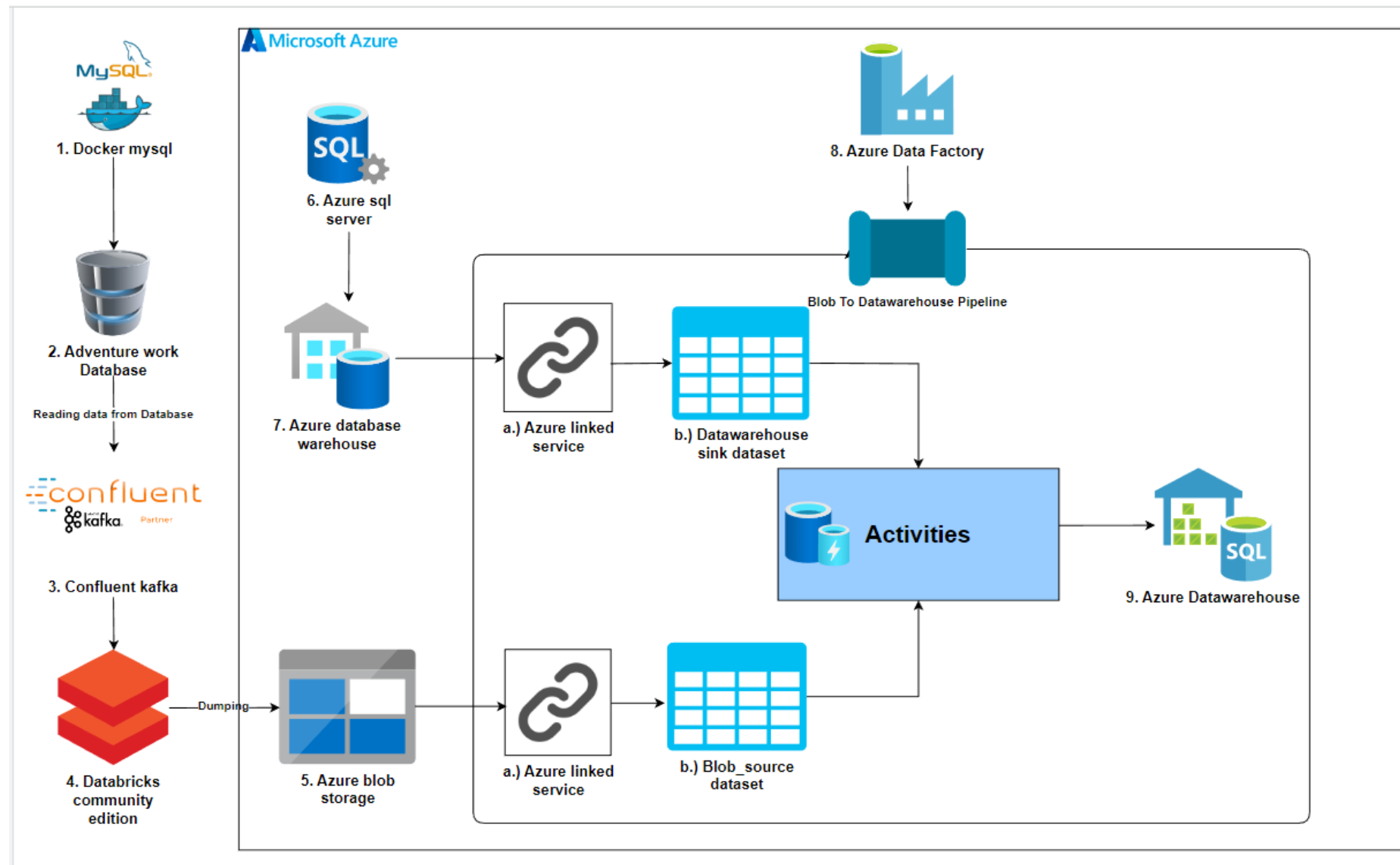


ETL pipeline in Microsoft Azure for streaming data

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:

First, we will pull MySQL image in docker after while running that MySQL server container we picked MySQL workbench and login into MySQL server that was up on running in docker container according to our credential's and we load Adventure work database (<https://github.com/Microsoft/sql-server-samples/releases/download/adventureworks/AdventureWorksDW2019.bak>) in MySQL after selecting top three tables with highest records those records will be dumping into azure blob.

Next, will create Kafka cluster with basic configuration setup and after creating topics will reading Adventure work database data with the help Kafka and dumping data into topics.

After, we read that Kafka topic data with the help of spark (Note: we will using Databricks Community Free Edition) and launching databricks workspace with basic version setup and attaching some confluent Kafka libraries to that workspace with help Kafka api & secret and Kafka schema registry keys and with boot strap URL end point will be connecting Kafka to databricks, and we will be reading data from Kafka to databricks after reading we will be writing data to azure blob storage with help of Mount point.

Now we will be creating Mount point to Azure blob storage to dump Kafka data into blob storage.

After the data had successfully extract from Kafka to blob storage, we will be creating azure SQL server to use azure Datawarehouse and next will be creating database for Datawarehouse with help of azure SQL server.

To make connection for blob storage to Datawarehouse will be using azure data factory, In data factory first we will be creating Linked services for blob storage as well as data warehouse, after will be launching pipeline to make data transfer from blob to data warehouse with the help of some azure data factory activities.

Finally we will be dumping data from azure blob storage to azure data ware house Note:- with out creating any kind of schema in data warehouse database.

2 Implementation:

2.1 Download MYSQL docker Image and run the image

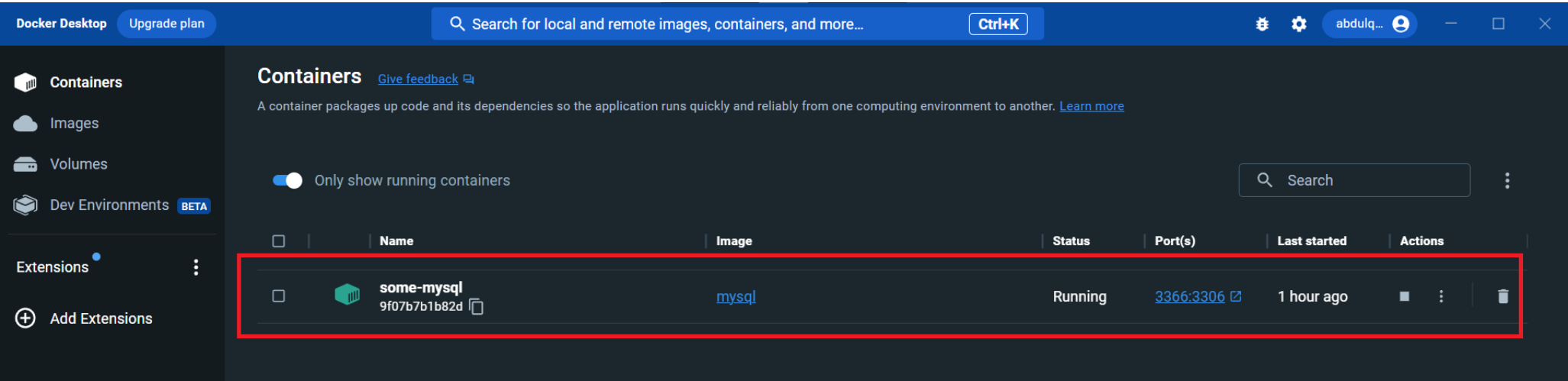
The implementation starts with pulling the docker MYSQL image from the docker hub.

```
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**.

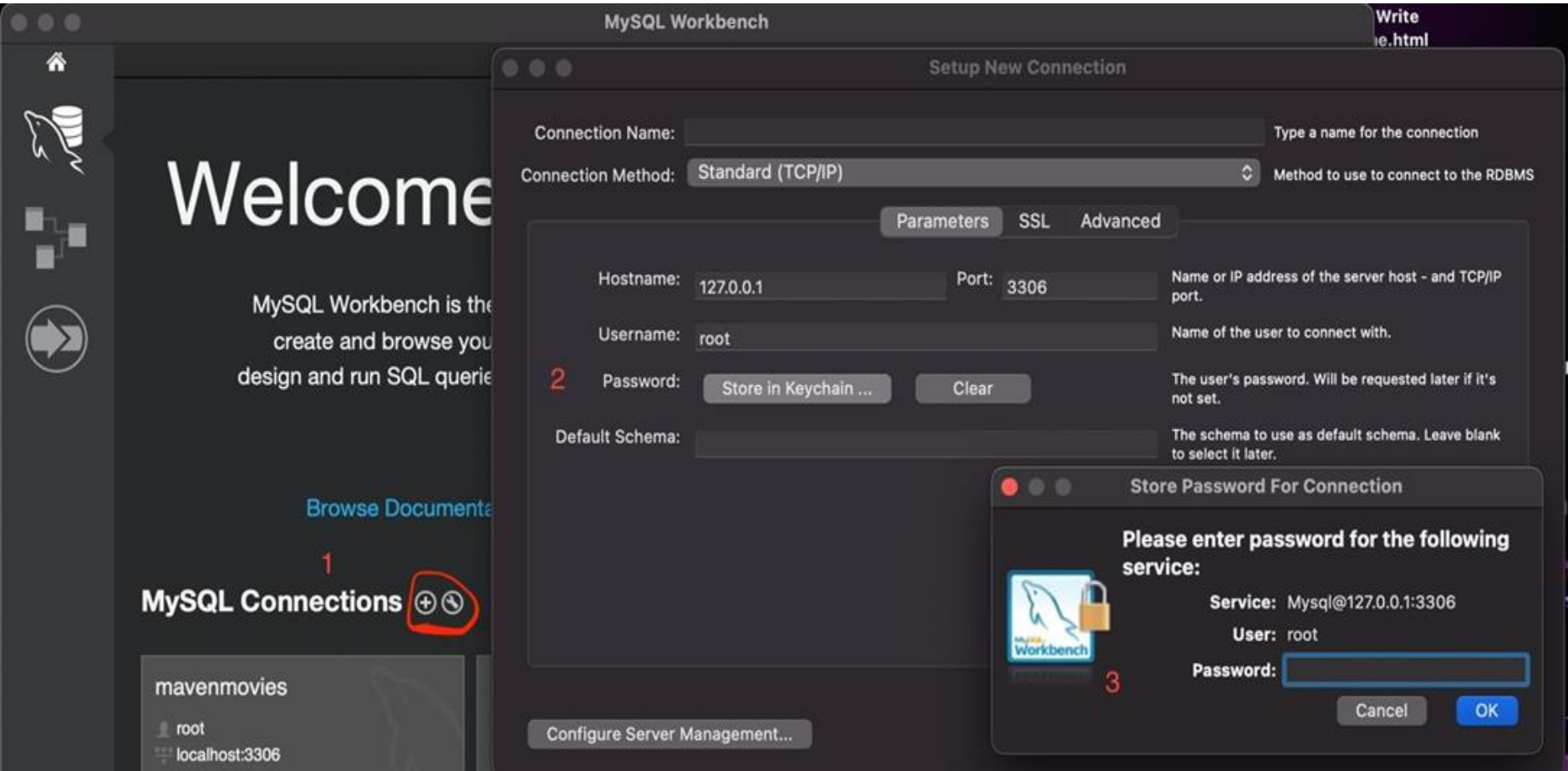
Once the image is download, run the docker MYSQL image. The below picture indicates the MYSQL image running in docker.



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)

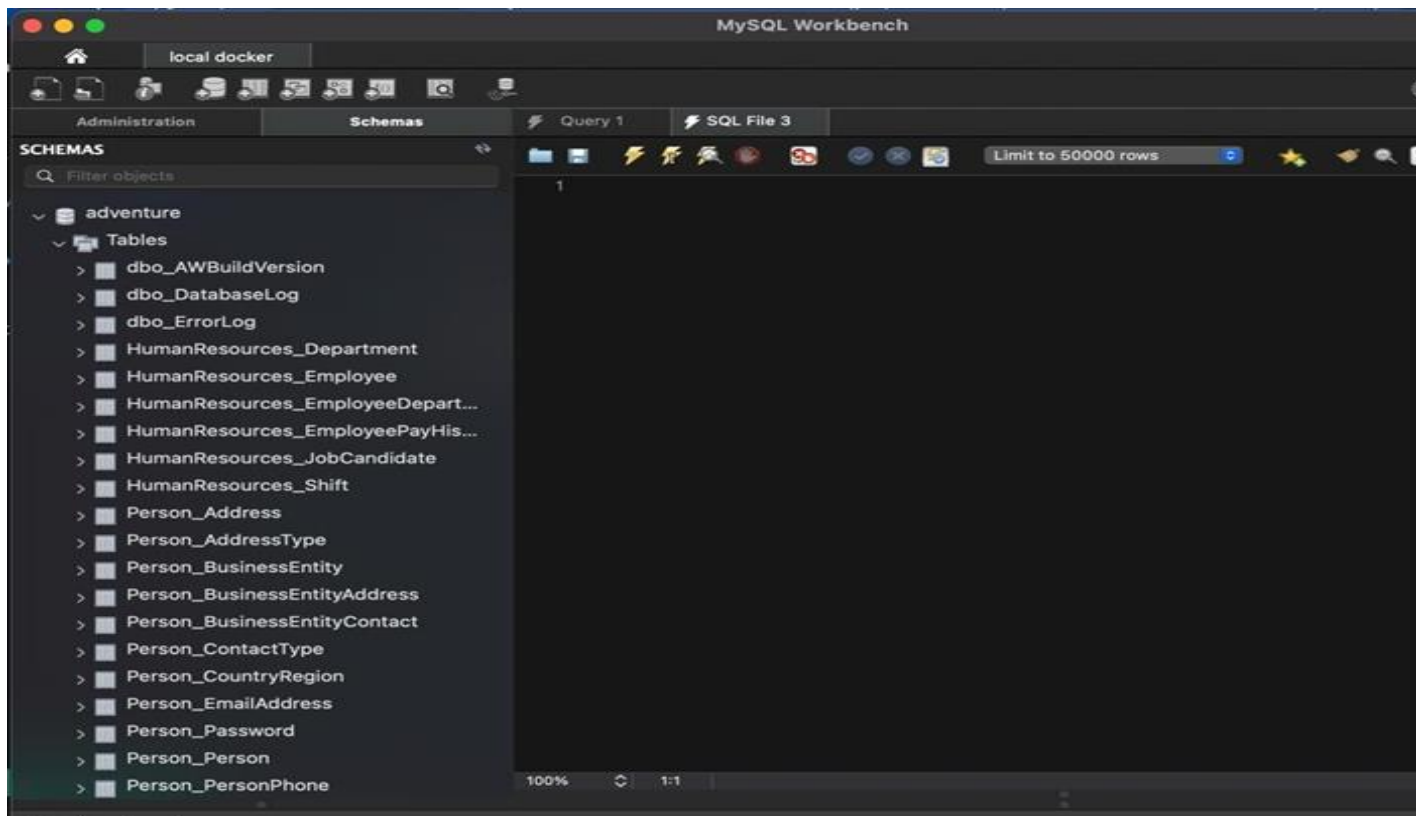
Click on the plus sign and add the MYSQL credentials that were used while pulling the docker image. The below picture shows the sequence of steps using number ordering.



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)
```

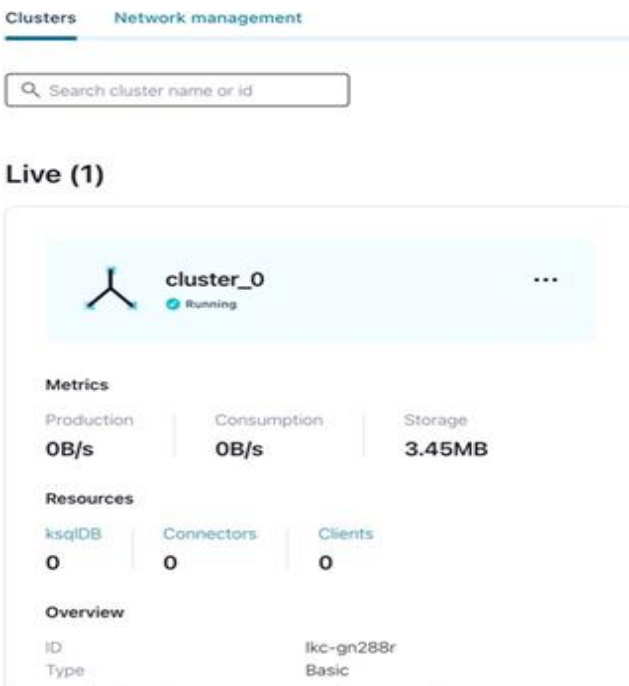
note that, username and password should be replaced with MYSQL credentials (i.e., username and password that were used while pulling the docker image). The port number should be replaced port number that we used while pulling the docker image. The database name should be the name of the database that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

4 KAFKA for streaming data

4.1 Configuring Confluent Kafka

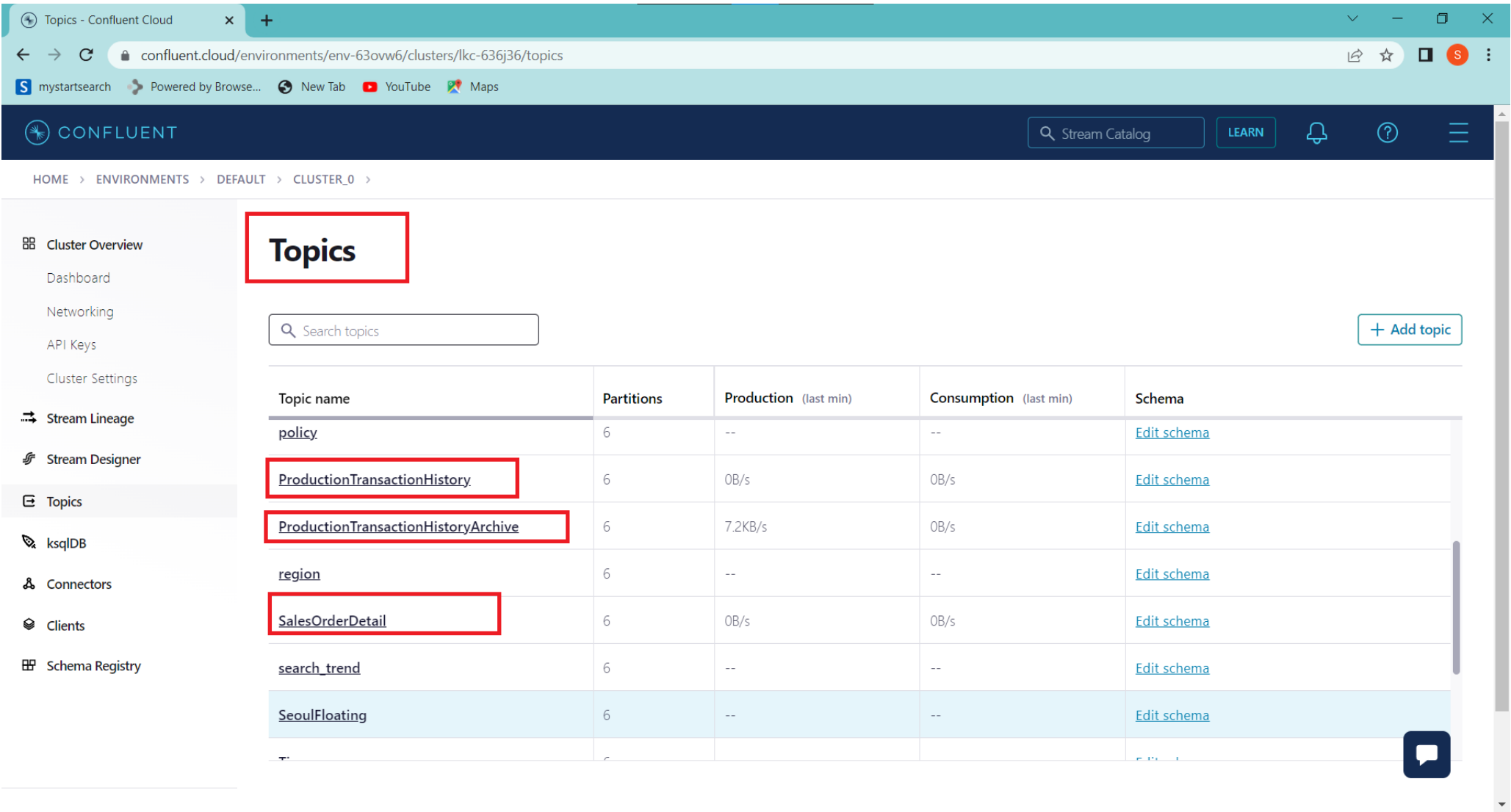
Now Once you are redirected to Confluent Cloud, click the **Create cluster** button. Select the cluster type **Basic** and click **Begin Configuration**.

Specify a cluster name as “**cluster_0**” and click **Launch cluster**.The picture below shows the cluster.



Copy the key and secret to a local file, and check **and saved my API key and secret and am ready to continue**. You will need the key and secret later for the Databricks code to connect to Kafka.

After created topics according to below names.



After creating topics and defining schema able to push records to Kafka

4.2 MySQL to Kafka Connection:

```
1. #!/usr/bin/env python
2. # -*- coding: utf-8 -*-
3. #
4. # Copyright 2020 Confluent Inc.
5. #
6. # Licensed under the Apache License, Version 2.0 (the "License");
7. # you may not use this file except in compliance with the License.
8. # You may obtain a copy of the License at
9. #
10. # http://www.apache.org/licenses/LICENSE-2.0
11. #
```

```

12. # Unless required by applicable law or agreed to in writing, software
13. # distributed under the License is distributed on an "AS IS" BASIS,
14. # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15. # See the License for the specific language governing permissions and
16. # limitations under the License.
17.
18.
19. # A simple example demonstrating use of JsonSerializer.
20.
21. import argparse
22. from uuid import uuid4
23. from six.moves import input
24. from confluent_kafka import Producer
25. from confluent_kafka.serialization import StringSerializer, SerializationContext, MessageField
26. from confluent_kafka.schema_registry import SchemaRegistryClient
27. from confluent_kafka.schema_registry.json_schema import JsonSerializer
28. #from confluent_kafka.schema_registry import *
29. import pandas as pd
30. from typing import List
31. import pandas as pd
32. import pymysql
33. from sqlalchemy import create_engine
34.
35. cnx = create_engine('mysql+pymysql://root:my-secret-pw@localhost:3366/adventure')
36. query = 'select * from Production_TransactionHistory'
37. # df = pd.read_sql(query, cnx) #read the entire table
38.
39.
40. API_KEY = '██████████'
41. ENDPOINT_SCHEMA_URL = '████████████████████████████████████████'
42. API_SECRET_KEY = '████████████████████████████████████████████████████████████'
43. BOOTSTRAP_SERVER = '████████████████████████████████████████'
44. SECURITY_PROTOCOL = 'SASL_SSL'
45. SSL_MACHENISM = 'PLAIN'
46. SCHEMA_REGISTRY_API_KEY = '██████████'
47. SCHEMA_REGISTRY_API_SECRET = '████████████████████████████████████████████████████████████'
48.
49. def sasl_conf():
50.
51.     sasl_conf = {'sasl.mechanism': SSL_MACHENISM,
52.                  # Set to SASL_SSL to enable TLS support.
53.                  # 'security.protocol': 'SASL_PLAINTEXT'}
54.                  'bootstrap.servers': BOOTSTRAP_SERVER,
55.                  'security.protocol': SECURITY_PROTOCOL,
56.                  'sasl.username': API_KEY,
57.                  'sasl.password': API_SECRET_KEY
58.                 }
59.     return sasl_conf
60.
61.
62.
63. def schema_config():
64.     return {'url': ENDPOINT_SCHEMA_URL,
65.            'basic.auth.user.info': f"{SCHEMA_REGISTRY_API_KEY}:{SCHEMA_REGISTRY_API_SECRET}"
66.           }
67.
68.
69.
70.
71. class Car:
72.     def __init__(self, record: dict):
73.         for k, v in record.items():
74.             setattr(self, k, v)
75.
76.         self.record = record
77.
78.     @staticmethod
79.     def dict_to_car(data: dict, ctx):
80.         return Car(record=data)
81.
82.     def __str__(self):
83.         return f"{self.record}"
84.
85.
86. def get_car_instance(query, cnx):
87.     df = pd.read_sql(query, cnx)
88.     df[['TransactionDate', 'ModifiedDate']] = df[['TransactionDate', 'ModifiedDate']].astype(str)
89.     df = df.iloc[:, 0:]
90.     columns = list(df.columns)
91.     cars: List[Car] = []
92.     for data in df.values:
93.         car = Car(dict(zip(columns, data)))
94.         cars.append(car)
95.     yield car
96.
97. def car_to_dict(car: Car, ctx):

```



```

98.         """
99.         Returns a dict representation of a User instance for serialization.
100.        Args:
101.            user (User): User instance.
102.            ctx (SerializationContext): Metadata pertaining to the serialization
103.                operation.
104.        Returns:
105.            dict: Dict populated with user attributes to be serialized.
106.        """
107.
108.        # User._address must not be serialized; omit from dict
109.        return car.record
110.
111.
112.    def delivery_report(err, msg):
113.        """
114.        Reports the success or failure of a message delivery.
115.        Args:
116.            err (KafkaError): The error that occurred on None on success.
117.            msg (Message): The message that was produced or failed.
118.        """
119.
120.        if err is not None:
121.            print("Delivery failed for User record {}: {}".format(msg.key(), err))
122.            return
123.        print('User record {} successfully produced to {} [{}] at offset {}'.format(
124.            msg.key(), msg.topic(), msg.partition(), msg.offset()))
125.
126.
127.    def main(topic):
128.
129.        schema_str = """
130.        {
131.        "$id": "http://example.com/myURI.schema.json",
132.        "$schema": "http://json-schema.org/draft-07/schema#",
133.        "additionalProperties": false,
134.        "description": "Sample schema to help you get started.",
135.        "properties": {
136.            "TransactionID": {
137.                "description": "The type(v) type is used.",
138.                "type": "number"
139.            },
140.            "ProductID": {
141.                "description": "The type(v) type is used.",
142.                "type": "number"
143.            },
144.            "ReferenceOrderID": {
145.                "description": "The type(v) type is used.",
146.                "type": "number"
147.            },
148.            "ReferenceOrderLineID": {
149.                "description": "The type(v) type is used.",
150.                "type": "number"
151.            },
152.            "TransactionDate": {
153.                "description": "The type(v) type is used.",
154.                "type": "string"
155.            },
156.            "TransactionType": {
157.                "description": "The type(v) type is used.",
158.                "type": "string"
159.            },
160.            "Quantity": {
161.                "description": "The type(v) type is used.",
162.                "type": "number"
163.            },
164.            "ActualCost": {
165.                "description": "The type(v) type is used.",
166.                "type": "number"
167.            },
168.            "ModifiedDate": {
169.                "description": "The type(v) type is used.",
170.                "type": "string"
171.            }
172.        },
173.        "title": "SampleRecord",
174.        "type": "object"
175.        }
176.        """
177.
178.        # schema_registry_conf = schema_config()
179.        # schema_registry_client = SchemaRegistryClient(schema_registry_conf)
180.        # #reading the latest version of schema and schema_str from schema registry and using it for data
181.        # serialization.
182.        # schema_name = schema_registry_client.get_schema(100002).schema_str
183.
184.        # string_serializer = StringSerializer('utf_8')

```

```

183.         # json_serializer = JsonSerializer(schema_name, schema_registry_client, car_to_dict)
184.
185.         schema_registry_conf = schema_config()
186.         schema_registry_client = SchemaRegistryClient(schema_registry_conf)
187.
188.         string_serializer = StringSerializer('utf_8')
189.         json_serializer = JsonSerializer(schema_str, schema_registry_client, car_to_dict)
190.
191.         producer = Producer(sasl_conf())
192.
193.         print("Producing user records to topic {}. ^C to exit.".format(topic))
194.         #while True:
195.             # Serve on_delivery callbacks from previous calls to produce()
196.         producer.poll(0.0)
197.         try:
198.             for idx,car in enumerate(get_car_instance(query,cnx)):
199.
200.                 print(car)
201.                 producer.produce(topic=topic, # publishing data in Kafka Topic one by one and use dynamic key
202.                                   key=string_serializer(str(uuid4()), car_to_dict),
203.                                   value=json_serializer(car, SerializationContext(topic, MessageField.VALUE)),
204.                                   on_delivery=delivery_report)
205.                 # if idx==1:    # index value is used for testing 1 or 2 records
206.                 #     break
207.         except KeyboardInterrupt:
208.             pass
209.         except ValueError:
210.             print("Invalid input, discarding record...")
211.             pass
212.
213.         print("\nFlushing records...")
214.         producer.flush()
215.
216.     main("ProductionTransactionHistory")

```

4.3 Messages in Kafka topic

The screenshot shows the Confluent Cloud interface for a Kafka topic named "ProductionTransactionHistoryArchive". The interface is divided into several sections:

- Left Sidebar:** Contains navigation links for Cluster Overview, Stream Lineage, Stream Designer, Topics (selected), KsqlDB, Connectors, Clients, and Schema Registry.
- Top Bar:** Displays the Confluent logo and navigation links for Stream Catalog, Learn, and Help.
- Topic Header:** The topic name "ProductionTransactionHistoryArchive" is highlighted with a red box. Below it are tabs for Overview, Messages (selected), Schema, and Configuration.
- Producers and Consumers:** Sections showing the status of producers and consumers for the topic.
- Message Fields:** A list of fields for the messages, including topic, partition, offset, timestamp, timestampType, headers, key, and value.
- Message List:** A list of messages with details such as TransactionID, ProductID, ReferenceOrderID, ReferenceOrderLineID, TransactionDate, Partition, Offset, and Timestamp.
- Right Sidebar:** Contains metadata for the topic, including Description, Tags, Date created, Date modified, Retention time, Retention size, Number of partitions, and Cleanup policy.
- Bottom Bar:** Includes a "Download" button and checkboxes for JSON and CSV formats.

Topics - Confluent Cloud

+

← → ↺

confluent.cloud/environments/env-63ovw6/clusters/lkc-636j36/topics/ProductionTransactionHistory/message-viewer

🔍 📄 ☆ 🏠 (S) ⋮

S mystartsearch

Powered by Browse...

New Tab

YouTube

Maps

CONFLUENT

Stream Catalog

LEARN

🔔

?

☰

HOME > ENVIRONMENTS > DEFAULT > CLUSTER_0 > TOPICS >

Cluster Overview

Dashboard

Networking

API Keys

Cluster Settings

Stream Lineage

Stream Designer

Topics

ksqlDB

Connectors

Clients

Schema Registry

CLI and Tools

Support

ProductionTransactionHistory

Overview Messages Schema Configuration

Producers

Bytes in/sec 0

⏮ ⏪

🔍 Filter by keyword

Jump to offset

▼ 🔍 0 / Partition: 0

☰ ⏮ ⏪

Consumers

Bytes out/sec 0

⏮ ⏪

Message fields

topic

partition

offset

timestamp

timestampType

headers

key

value

TransactionID

ProductID

+ Produce a new message to this topic

▼ {"TransactionID":"101372","ProductID":967,"ReferenceOrderID":53502,"ReferenceOrderLineID":30,"TransactionDate":"2013-07-30 18:30:00",...} Newest

Partition: 0 Offset: 209 Timestamp: 1678788371795

▼ {"TransactionID":"101362","ProductID":956,"ReferenceOrderID":53502,"ReferenceOrderLineID":8,"TransactionDate":"2013-07-30 18:30:00",...}

Partition: 0 Offset: 208 Timestamp: 1678788371780

▼ {"TransactionID":"101360","ProductID":953,"ReferenceOrderID":53502,"ReferenceOrderLineID":23,"TransactionDate":"2013-07-30 18:30:00",...}

Partition: 0 Offset: 207 Timestamp: 1678788371777

▼ {"TransactionID":"101344","ProductID":864,"ReferenceOrderID":53502,"ReferenceOrderLineID":13,"TransactionDate":"2013-07-30 18:30:00",...}

Partition: 0 Offset: 206 Timestamp: 1678788371755

Explore Stream Lineage

Description

Add description

Tags

Add tags to this topic

Add business metadata

Date created

Mar 14 2023 3:30 PM

Date modified

--

Retention time

1 week

Retention size

Infinite

Number of partitions

6

Cleanup policy

JSON CSV Download

Topics - Confluent Cloud

+

confluent.cloud/environments/env-63ovw6/clusters/lkc-636j36/topics/SalesOrderDetail/message-viewer

Search

Share

Star

Fullscreen

Settings

mystartsearch

Powered by Browse...

New Tab

YouTube

Maps

CONFLUENT

Stream Catalog

LEARN

Notifications

Help

Menu

HOME > ENVIRONMENTS > DEFAULT > CLUSTER_0 > TOPICS >

Cluster Overview

Dashboard

Networking

API Keys

Cluster Settings

Stream Lineage

Stream Designer

Topics

ksqlDB

Connectors

Clients

Schema Registry

SalesOrderDetail

Overview

Messages

Schema

Configuration

Producers

Bytes in/sec

0

Consumers

Bytes out/sec

0

Message fields

topic

partition

offset

timestamp

timestampType

headers

key

value

SalesOrderID

SalesOrderDetailID

Filter by keyword

Jump to offset

0 / Partition: 0

+ Produce a new message to this topic

✓ {"SalesOrderID":43776,"SalesOrderDetailID":432,"CarrierTrackingNumber":"None","OrderQty":1,"ProductID":753,"SpecialOfferID":1,"Unit...

Partition: 0 Offset: 49 Timestamp: 1678786910528

✓ {"SalesOrderID":43756,"SalesOrderDetailID":412,"CarrierTrackingNumber":"None","OrderQty":1,"ProductID":765,"SpecialOfferID":1,"Unit...

Partition: 0 Offset: 48 Timestamp: 1678786910493

✓ {"SalesOrderID":43755,"SalesOrderDetailID":411,"CarrierTrackingNumber":"None","OrderQty":1,"ProductID":750,"SpecialOfferID":1,"Unit...

Partition: 0 Offset: 47 Timestamp: 1678786910492

✓ {"SalesOrderID":43753,"SalesOrderDetailID":409,"CarrierTrackingNumber":"None","OrderQty":1,"ProductID":749,"SpecialOfferID":1,"Unit...

Partition: 0 Offset: 46 Timestamp: 1678786910488

Explore Stream Lineage

Description

Tags

Add business metadata

Date created

Date modified

Retention time

Retention size

Number of partitions

Cleanup policy

CLI and Tools

Support

JSON

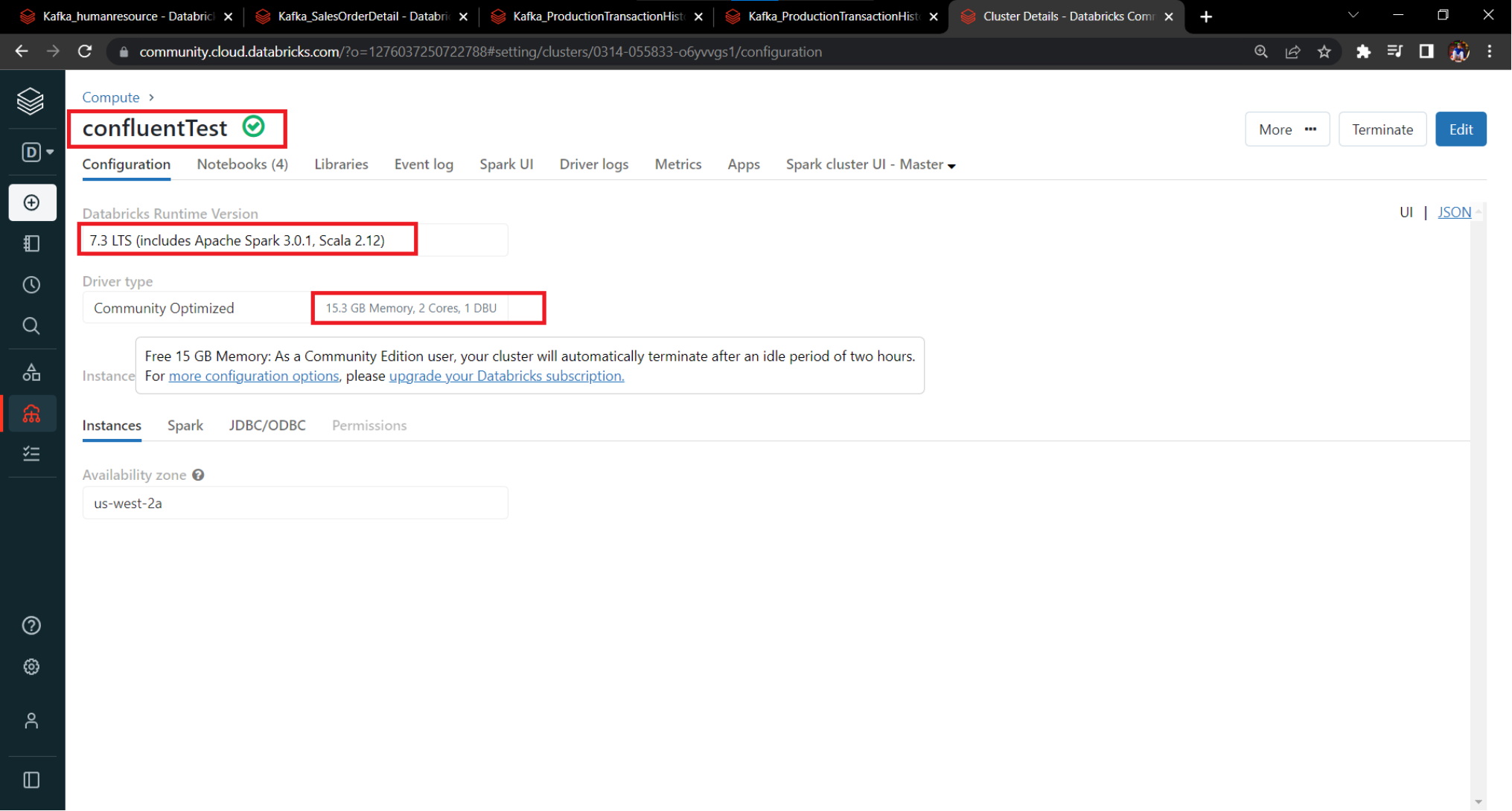
CSV

Download

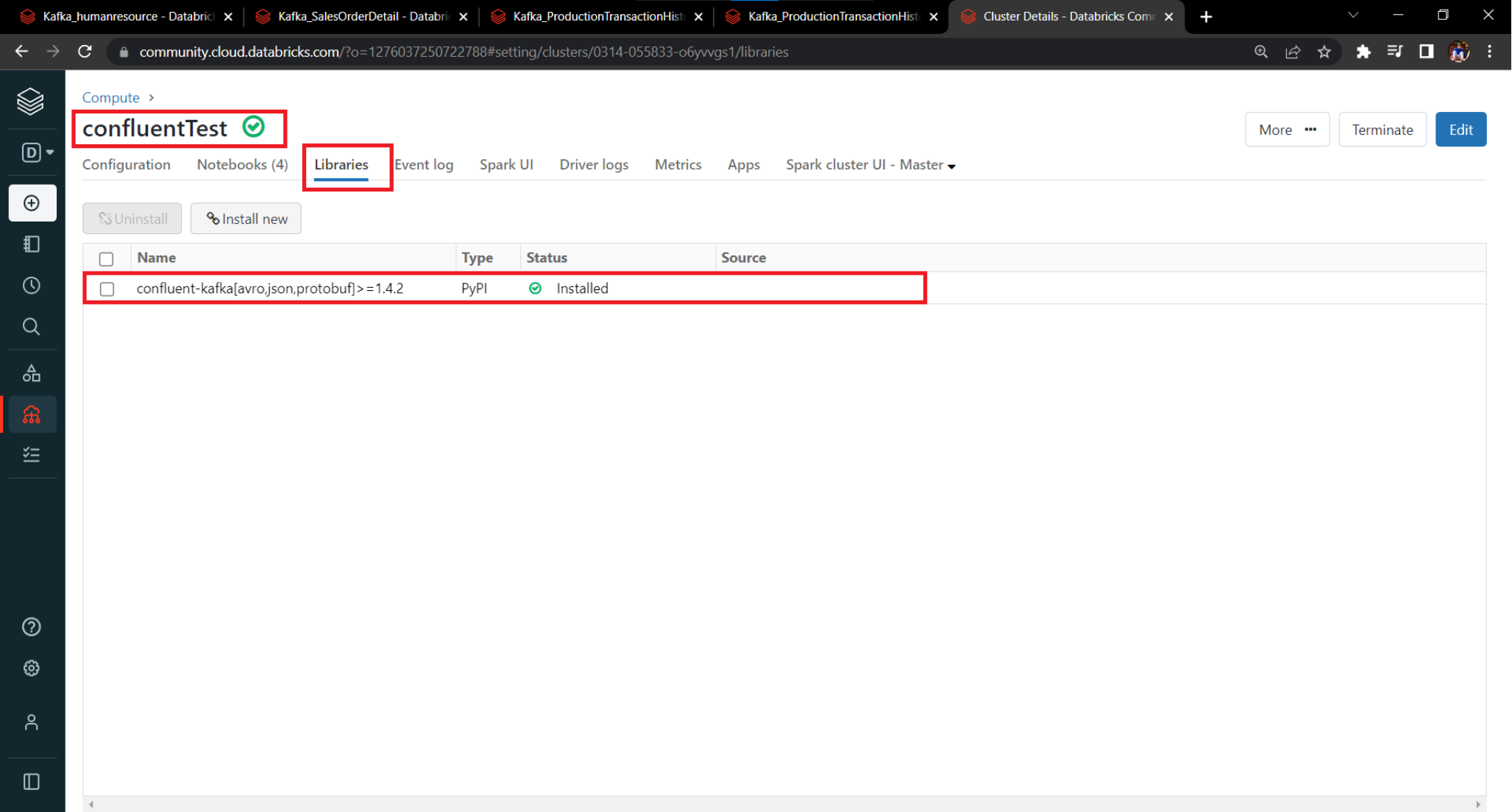
5 Process the Kafka Data with Databricks:

5.1 Configuring Databricks.

1. Log in to databricks community edition
2. Once you're logged in to the Databricks community edition, we will need a created cluster with cluster name **confluentTest** with Databricks runtime version and driver type seen below.



3. Once the cluster is spun up, you will need to add an additional **library** to it, Click on **Libraries**, and add **confluent-Kafka[avro,json,protobuf]>=1.4.2**, and install it.



4. Create a notebook and added cluster **confluentTest** to notebook.

5. Gather keys, secrets, and paths

- . The name of the Kafka cluster
- . The Kafka API key for the cluster
- . The Kafka API secret for the cluster
- . The bootstrap server for the cluster
- . The Schema Registry URL
- . The Schema Registry API key
- . The Schema Registry API secret

```
1. confluentClusterName = "cluster_0"
2. confluentBootstrapServers = "[REDACTED]"
3. confluentTopicName = "[REDACTED]"
4. schemaRegistryUrl = "[REDACTED]"
5. confluentApiKey = "[REDACTED]"
6. confluentSecret = "[REDACTED]"
7. confluentRegistryApiKey = "[REDACTED]"
8. confluentRegistrySecret = "[REDACTED]"
```

6. Once we have the connection information that we need, the next step is to set up a Schema Registry client. Confluent Cloud requires the Schema Registry API key and secret to authenticate—note the use of some of the variables declared below.

```
1. from confluent_kafka.schema_registry import SchemaRegistryClient
2. import ssl
3. schema_registry_conf = {
4.     'url': schemaRegistryUrl,
5.     'basic.auth.user.info': '{}:{}'.format(confluentRegistryApiKey, confluentRegistrySecret)}
6. schema_registry_client = SchemaRegistryClient(schema_registry_conf)
```

7. The first part of the above Read Stream statement reads the data from our Kafka topic. First, we specify the format of the Read Stream as **“Kafka”**

8. Next, the bootstrap servers, protocol, authentication configuration, and topic need to be specified

9. The **“kafkashaded”** at the front of the **kafka.sasl.jaas.config** option is present so that the PlainLoginModule can be used.

10. Next, specify the **Kafka topic offset** at which to start at. By default, a Read Stream from a Kafka topic will use **“latest”** for all topic partitions. That means that it will only start pulling data from the time that the stream started reading and not pull anything older from the topic.

11. In our project, we will tell it to use **“earliest”**, meaning it will start reading data from the **earliest available offset in the topic**.

12. we can also specify the **failOnDataLoss** option—when set to **“true”**, it will stop the stream if there is a break in the sequence of offsets because it assumes data was lost.

13. When set to **“false”**, it will ignore missing offsets.

14. Finally, specify load.

```
1. df = (
2.     spark
3.     .readStream
4.     .format("kafka")
5.     .option("kafka.bootstrap.servers", confluentBootstrapServers)
6.     .option("kafka.security.protocol", "SASL_SSL")
7.     .option("kafka.sasl.jaas.config", "kafkashaded.org.apache.kafka.common.security.plain.PlainLoginModule required
username='{}' password='{}';".format(confluentApiKey, confluentSecret))
8.     .option("kafka.ssl.endpoint.identification.algorithm", "https")
9.     .option("kafka.sasl.mechanism", "PLAIN")
10.     .option("subscribe", confluentTopicName)
11.     .option("startingOffsets", "earliest")
12.     .option("failOnDataLoss", "false")
13.     .option("includeHeaders", "true")
14.     .load()
15. )
```

15. Finally, new columns need to be generated as part of our Spark Data Frame.

. The columns needs to be cast to a string

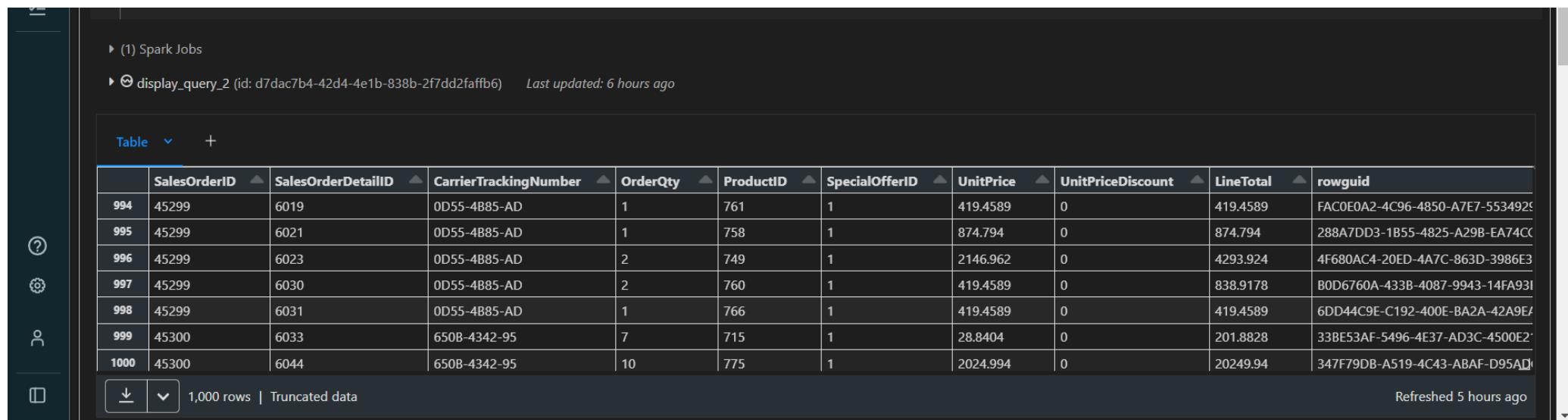
```
1. df = df.selectExpr("CAST(topic AS STRING)", "CAST(partition AS STRING)", "CAST(offset AS STRING)", "CAST(value AS STRING)")
```

16. The first five bytes need to be removed from the value

```
1. df1 = df.withColumn('VALUES', fn.expr("substring(value, 6, length(value)-5)"))
2. df2 = df1.select("VALUES")
```

17. We first use the `from_json` function from `from pyspark.sql.functions import from_json` to deserialize the record
And also defined new **schema** to it.

```
1. schema = StructType([
2.     StructField("SalesOrderID", IntegerType(), True),
3.     StructField("SalesOrderDetailID", IntegerType(), True),
4.     StructField("CarrierTrackingNumber", StringType(), True),
5.     StructField("OrderQty", IntegerType(), True),
6.     StructField("ProductID", IntegerType(), True),
7.     StructField("SpecialOfferID", IntegerType(), True),
8.     StructField("UnitPrice", FloatType(), True),
9.     StructField("UnitPriceDiscount", FloatType(), True),
10.    StructField("LineTotal", FloatType(), True),
11.    StructField("rowguid", StringType(), True),
12.    StructField("ModifiedDate", StringType(), True),
13. ])
14.
15. dfJSON = df2.withColumn("jsonData", from_json(col("VALUES"), schema))
16. DF = dfJSON.select("jsonData.*")
17. DF.display() Here is the output if we display the DF
```



The screenshot shows a Databricks table view with 10 rows of data. The table has 11 columns: SalesOrderID, SalesOrderDetailID, CarrierTrackingNumber, OrderQty, ProductID, SpecialOfferID, UnitPrice, UnitPriceDiscount, LineTotal, rowguid, and ModifiedDate. The data is truncated at the bottom, showing 1,000 rows.

	SalesOrderID	SalesOrderDetailID	CarrierTrackingNumber	OrderQty	ProductID	SpecialOfferID	UnitPrice	UnitPriceDiscount	LineTotal	rowguid
994	45299	6019	0D55-4B85-AD	1	761	1	419.4589	0	419.4589	FAC0E0A2-4C96-4850-A7E7-5534929
995	45299	6021	0D55-4B85-AD	1	758	1	874.794	0	874.794	288A7DD3-1B55-4825-A29B-EA74CC
996	45299	6023	0D55-4B85-AD	2	749	1	2146.962	0	4293.924	4F680AC4-20ED-4A7C-863D-3986E3
997	45299	6030	0D55-4B85-AD	2	760	1	419.4589	0	838.9178	B0D6760A-433B-4087-9943-14FA93I
998	45299	6031	0D55-4B85-AD	1	766	1	419.4589	0	419.4589	6DD44C9E-C192-400E-BA2A-42A9E/
999	45300	6033	650B-4342-95	7	715	1	28.8404	0	201.8828	33BE53AF-5496-4E37-AD3C-4500E2
1000	45300	6044	650B-4342-95	10	775	1	2024.994	0	20249.94	347F79DB-A519-4C43-ABAF-D95AD/

Note: will be defining all the above properties to read all files in same way.

5.2 Configure the sink:

18. To sink our stream in an Azure Blob Storage, we create a mount point from our Databricks Cluster to an existing Blob Container.

```
1. # BLOB STORAGE INITIALIZATION
2. try:
3.     container_name = "container1"
4.     storage_account_name = "qadir"
5.     accountkey = "\u003Caccountkey\u003E"
6.     config = "fs.azure.account.key.qadir.blob.core.windows.net"
7.
8.     dbutils.fs.mount(
9.         source = "wasbs://{}/@{}.blob.core.windows.net/".format(container_name, storage_account_name),
10.        mount_point = "/mnt/blobstorage", #blobstorage is mount point name
11.        extra_configs = {config: accountkey}
12.    )
13.
14. except Exception as e:
15.     print(e)
16.     print("already mounted. Try to unmount first")
```

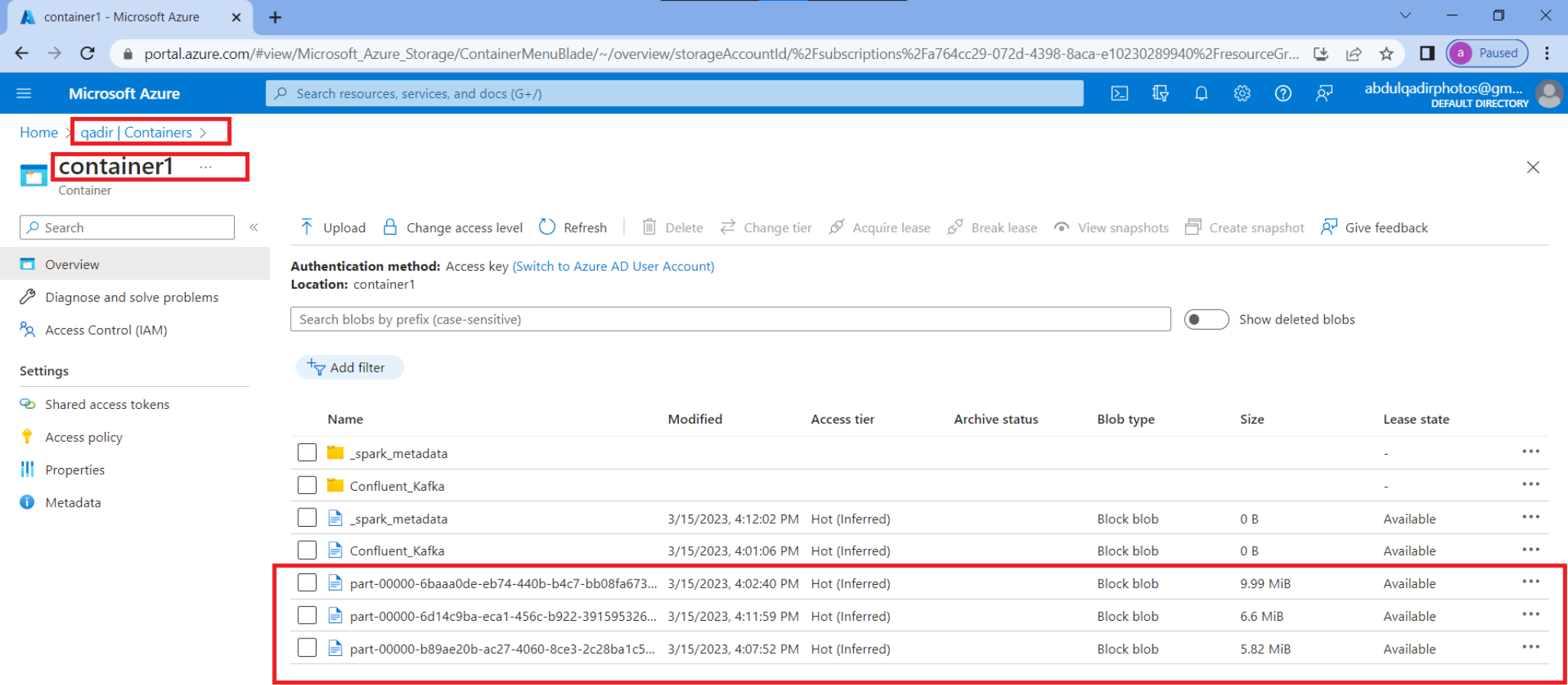
- . We use the **dbutils** Databricks library to **mount** under a Azure Blob Storage from a container named **container1** .
- . And we will be mounting to container with **mount point** name **blobstorage**.
- . In this project we use a **account key** to access the storage service.

19. The last task is to define the write Stream, We can now write this stream into a blob storage

```
1. DF.coalesce(1).writeStream \  
2. .format("csv") \  
3. .option("header",True) \  
4. .outputMode("append") \  
5. .option("mergeSchema", "true") \  
6. .option("checkpointLocation", "/mnt/blobstorage/Confluent_Kafka/_checkpoints/kafka") \  
7. .start("/mnt/blobstorage")
```

. The checkpoint Location option is not particular but to the Spark Structured Streaming API, It allow us to use checkpointing and write-ahead logs to recover in case of failure

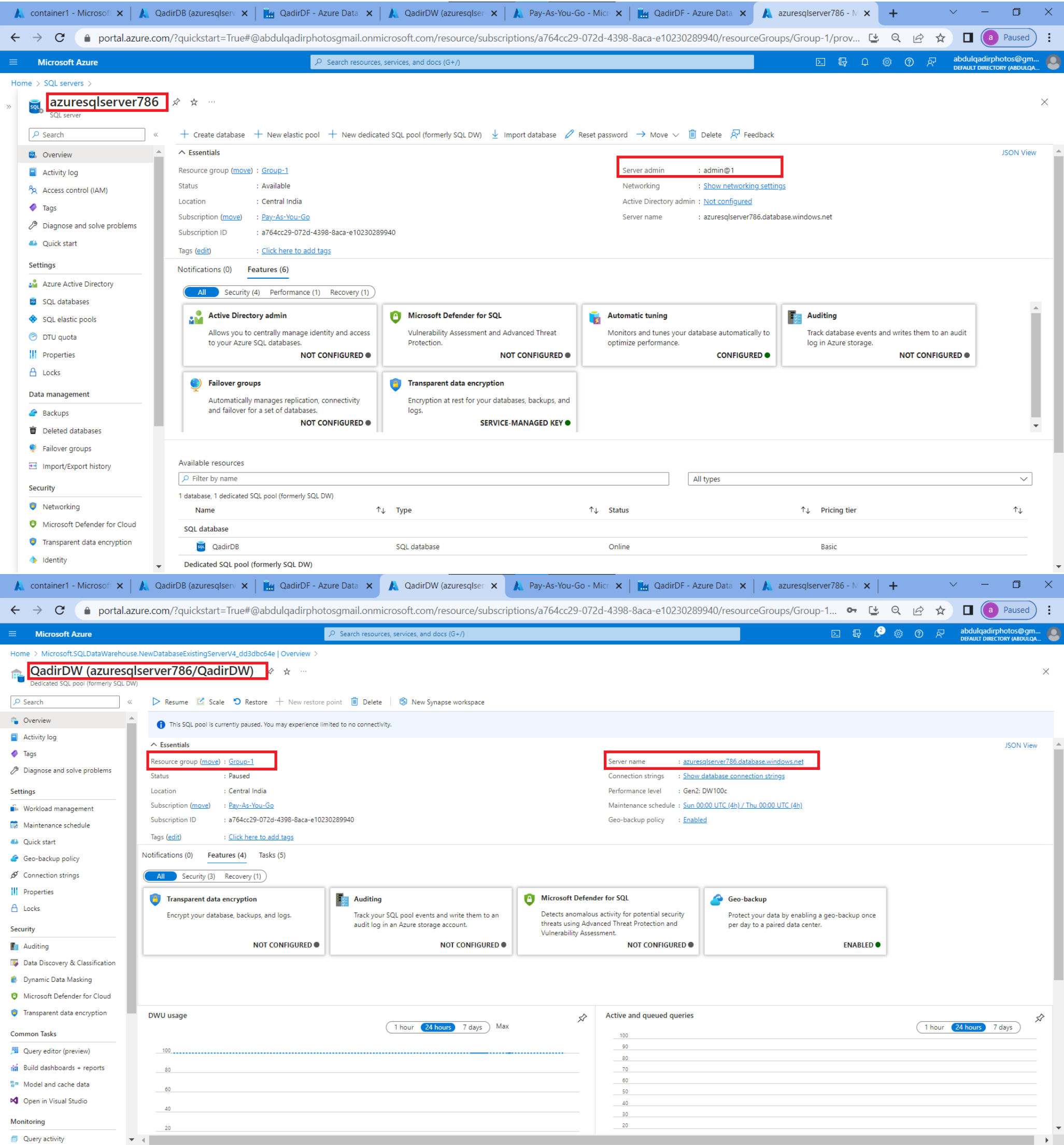
. Finally, we give the path under the mount point created within Azure Blob Storage to start writing under a dedicated container "container1" as you seen below.



6 Create an Azure Data warehouse Database:

When we log into the azure portal from portal.azure.com. we will see available services from Azure, we can select **Dedicated SQL pools (formerly SQL DW)**, before creating a database, we need to choose an Azure subscription, Then we need to select or create a new resource group.

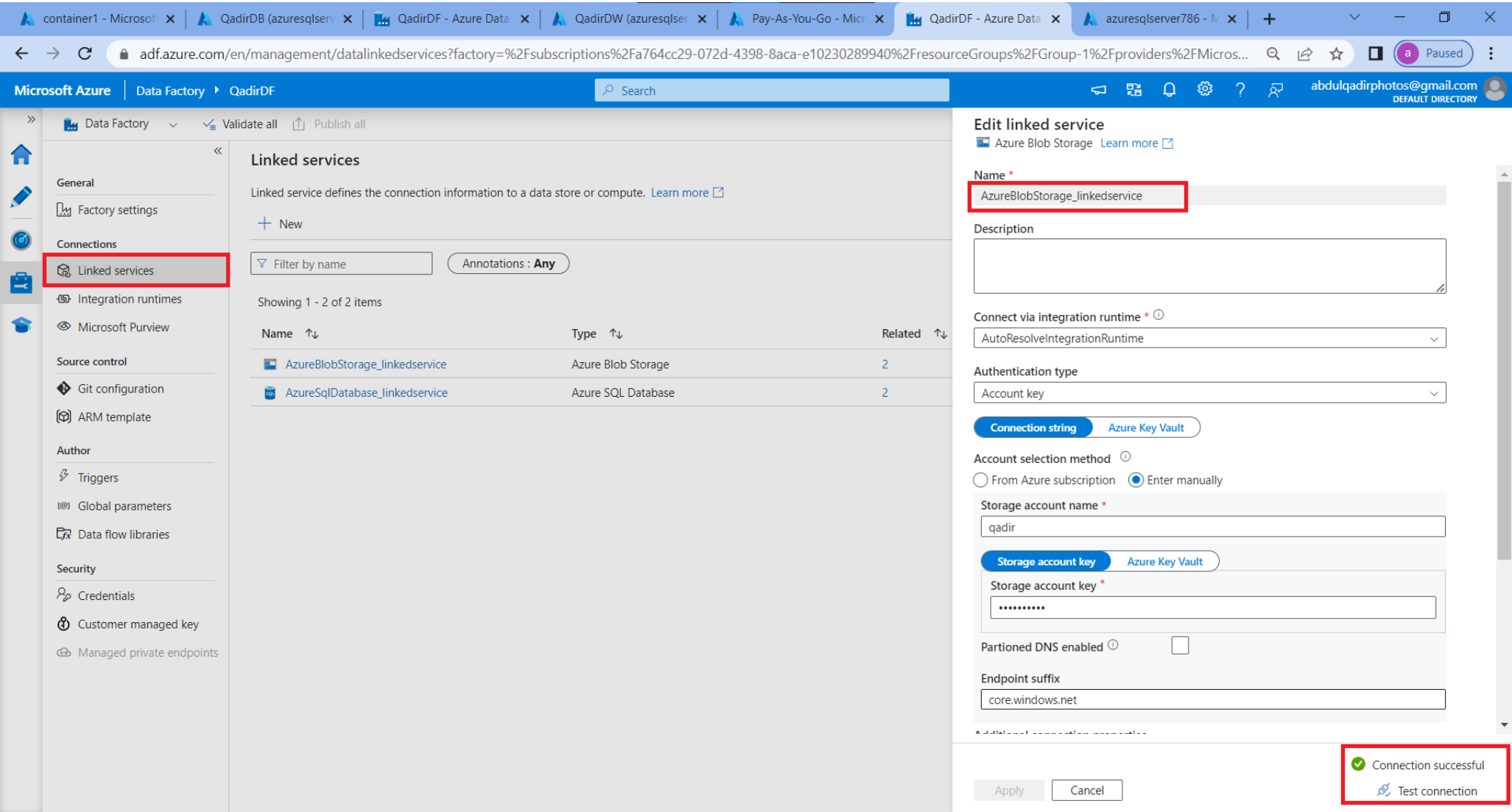
Next, we need to provide database details for SQL DW, Next, we need to configure the server for the database, In this configuration, the server name is **azuresqlserver786.database.windows.net** and you need to provide an administrator login to the server.



7 Create an Azure Linked services for Blob:

First, need to set up **Azure Data Factory Environment**

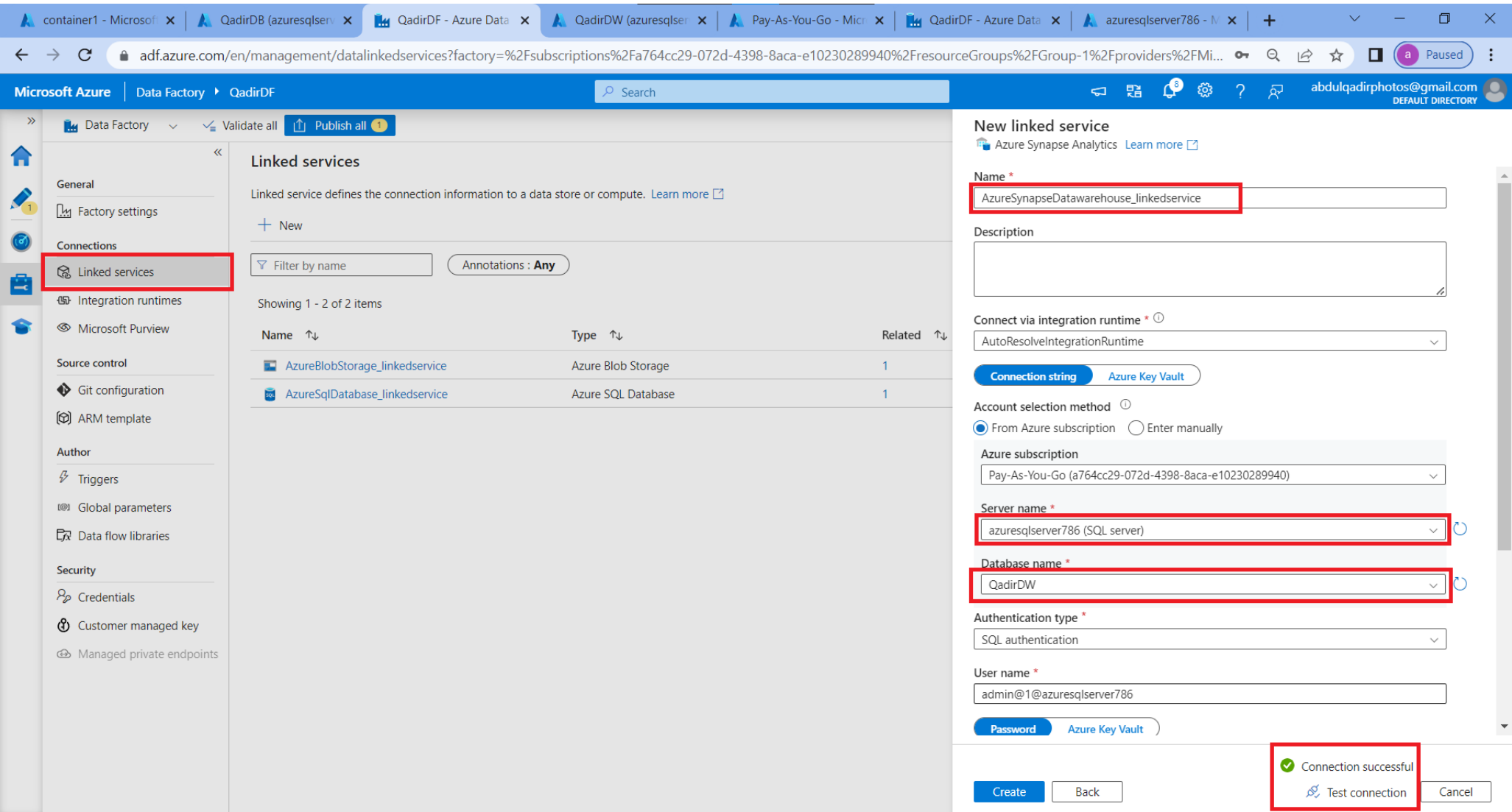
In the **Manage** section, go to **Linked Services** and click on **New**. Search for **Azure blob storage**, Give a name to the new linked service and use the default integration runtime and storage account name with storage account key and make test connection while before creating.



7.1 Create an Azure Linked services for Datawarehouse:

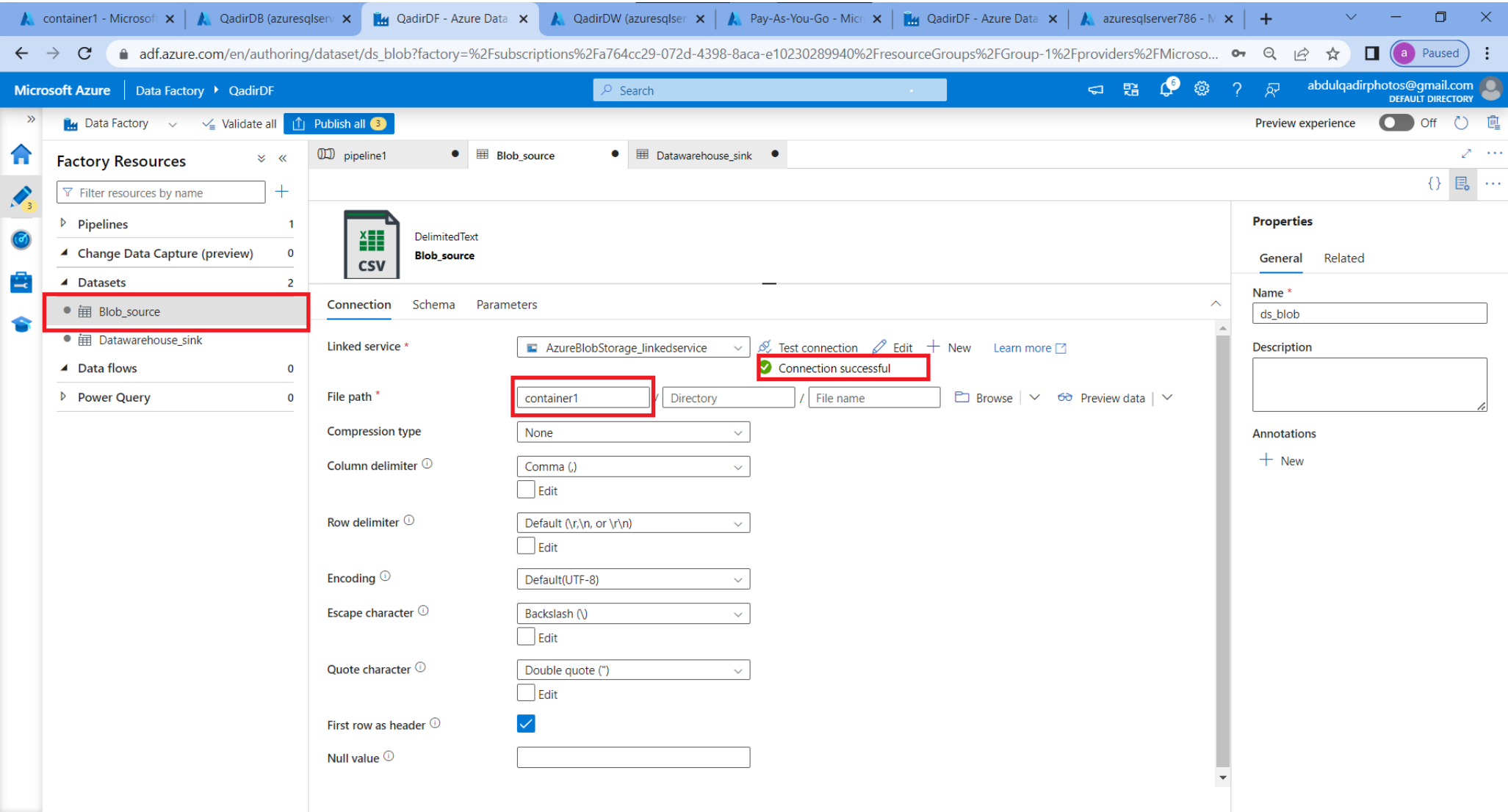
First, need to set up **Azure Data Factory Environment**

In the **Manage** section, go to **Linked Services** and click on **New**. Search for **Azure blob storage**, Give a name to the new linked service and use the default integration runtime and choose SQL authentication This means we're going to log into Azure Data warehouse DB using the user credentials of SQL Server with server name **azuresqlserver786.database.windows.net** with database name **QADIRDW** and make test connection while before creating.



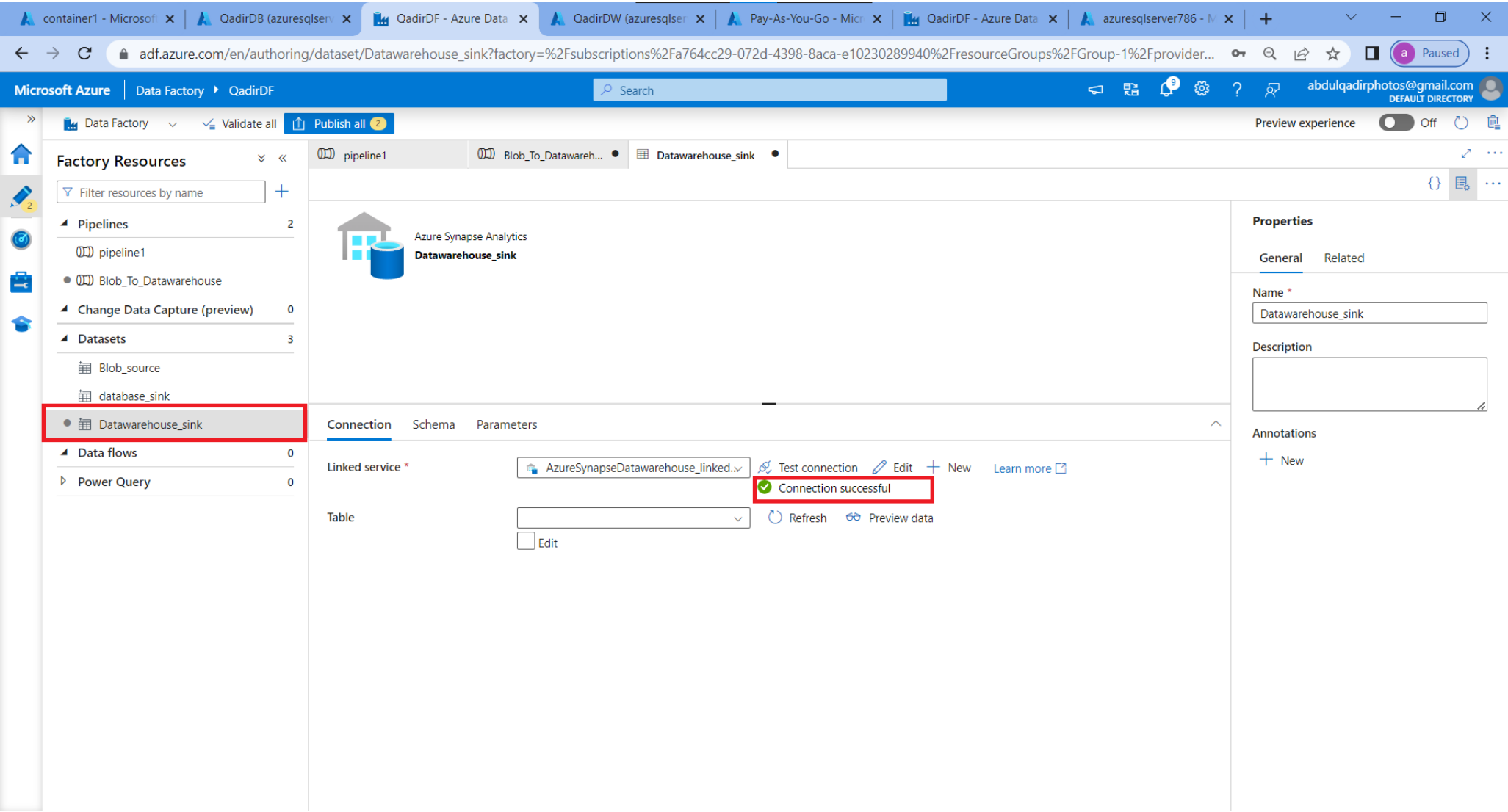
8 Creating a Datasets Manually for Blob storage:

Now we will going to create a dataset that reads in an csv file that was uploaded it to the same blob container that we had dumped from Kafka, In the **Author** section, expand the datasets section, choose **New dataset** in the popup, for data store choose **Azure blob storage** and choose **Delimited Text** file format and provide dataset name **Blob_source** and choose linked service for Azure blob storage and select container path, set the first row as header and choose to import the schema as none.



8.1 Creating a Datasets Manually for Data warehouse:

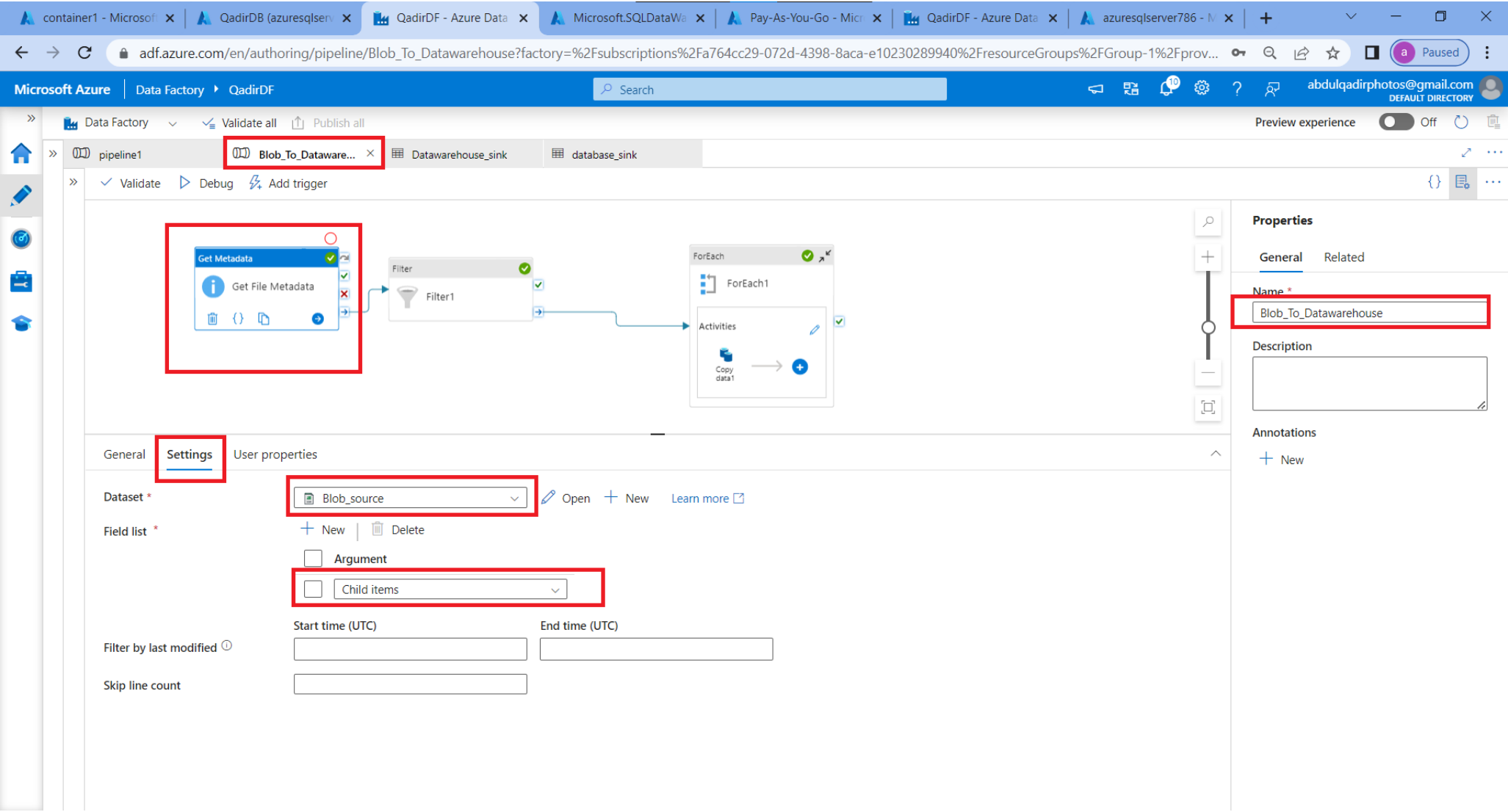
Next we will going to create a dataset for data warehouse, In the **Author** section, expand the datasets section, choose **New dataset** in the popup, for data store choose **Azure synapse Analytics** and provide dataset name **Datawarehouse_sink** and choose linked service for AzureSynapseDatawarehouse.



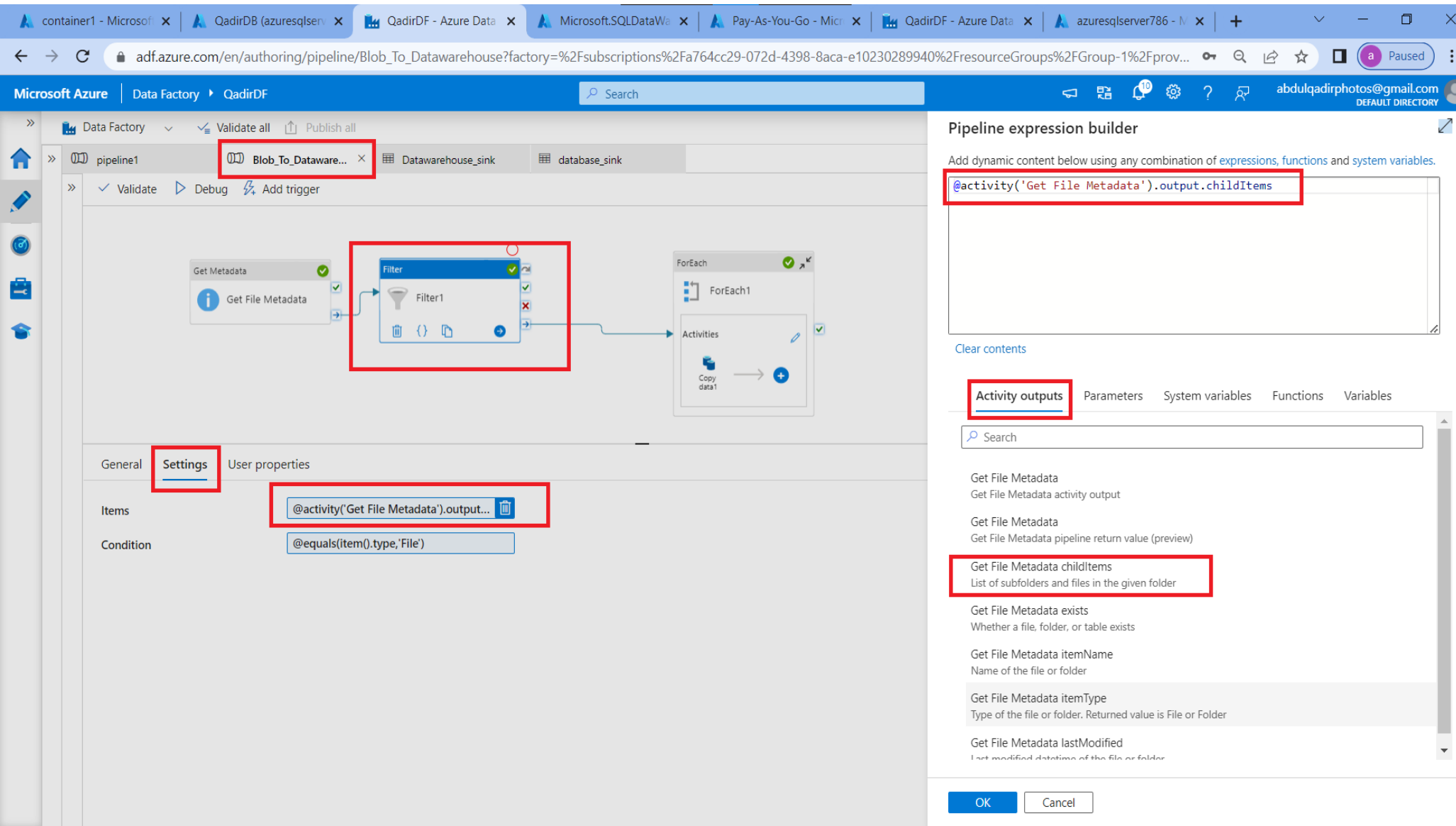
9 Building the Pipeline:

Go to the **Author** section of ADF Studio Go to pipeline > pipeline to create a new pipeline, Start by giving the new pipeline a decent name **Blob_To_Datawarehouse**

Next, add a **Get Metadata** activity to the canvas and name it "**Get File Metadata**", In the **Settings** tab choose dataset for blob storage as **Blob_source** and for Field list choose **Child items** on container.



Next, drag a **Filter** activity to the canvas. Connect the **Get File Metadata** activity with the new activity. Name it "**Filter1**", and in Setting tab click on **items** and then on “**Add dynamic content**” this will take us to the **expression builder** of ADF and choose **Activity outputs** where it will define it will choosing child items in container and after click on **Condition** and then on “**Add dynamic content**” this will take us to the **expression builder** of ADF and choose **Filter iterator** where it will define it will **choosing only files not folders** in a container.

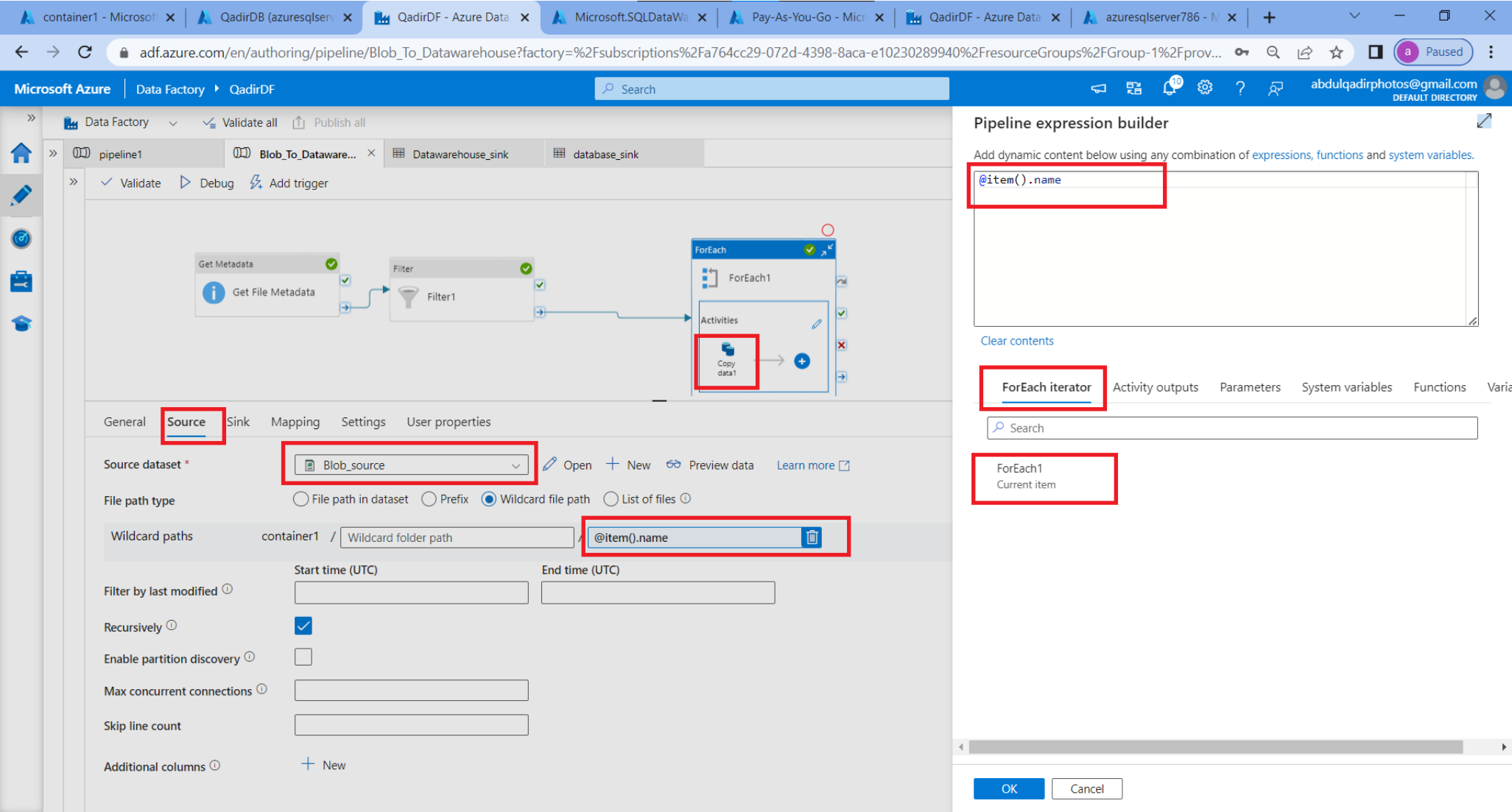


The screenshot shows the Microsoft Azure Data Factory interface. In the top navigation bar, the 'Pipeline expression builder' is open. The main canvas displays a pipeline with a 'Filter' activity named 'Filter1' and a 'ForEach' activity named 'ForEach1'. The 'Filter1' activity is highlighted with a red box. Below the canvas, the 'Settings' tab is selected, showing the 'Condition' property set to '@equals(item().type,'File')'. The 'Pipeline expression builder' on the right is also open, showing the expression '@equals(item().type,'File')' in the text area. The 'Filter iterator' tab is selected in the builder, and 'Filter1 Current item' is chosen from the search results.

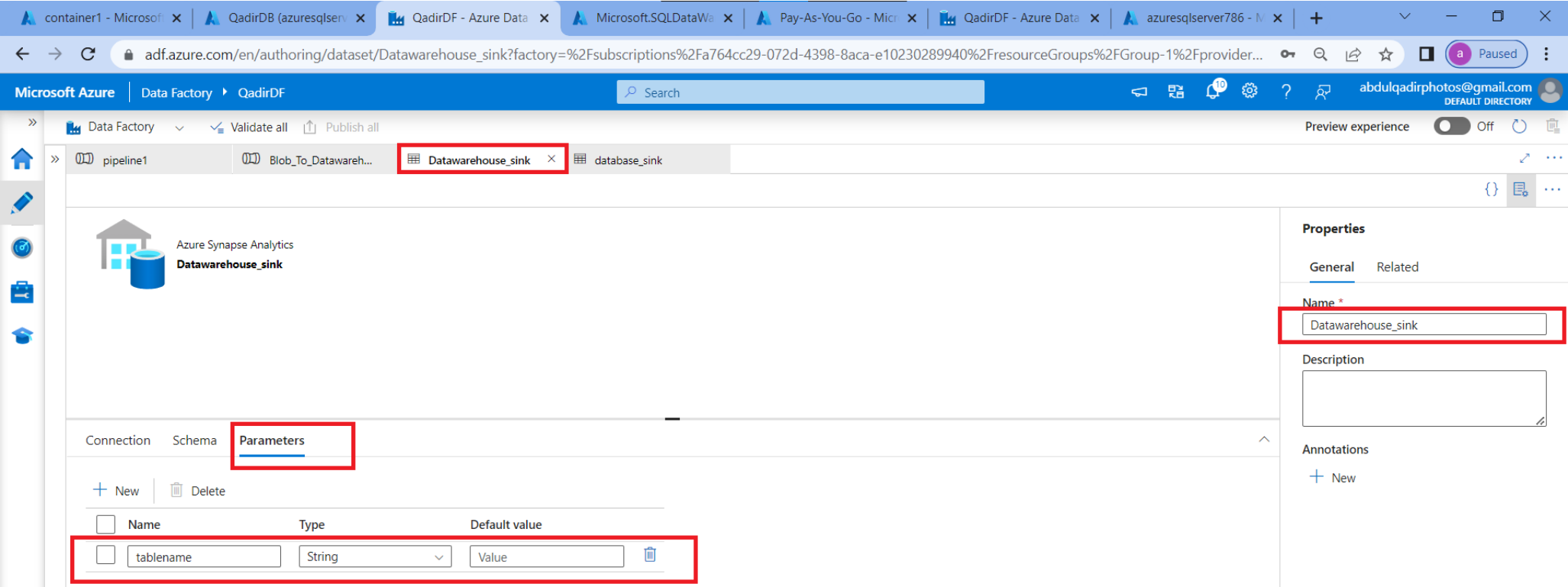
Next, drag a **ForEach** activity to the canvas. Connect the **Filter1** activity with the new activity. Name it "**ForEach1**", and in Setting tab click on **items** and then on “**Add dynamic content**” this will take us to the **expression builder** of ADF and choose **Activity outputs** where it will define it will choose output values from the filter activity.

The screenshot shows the Microsoft Azure Data Factory interface. In the top navigation bar, the 'Pipeline expression builder' is open. The main canvas displays a pipeline with a 'Filter' activity named 'Filter1' and a 'ForEach' activity named 'ForEach1'. The 'ForEach1' activity is highlighted with a red box. Below the canvas, the 'Settings' tab is selected, showing the 'Items' property set to '@activity('Filter1').output.Value'. The 'Pipeline expression builder' on the right is also open, showing the expression '@activity('Filter1').output.Value' in the text area. The 'Activity outputs' tab is selected in the builder, and 'Get File Metadata Get File Metadata pipeline return value (preview)' is chosen from the search results.

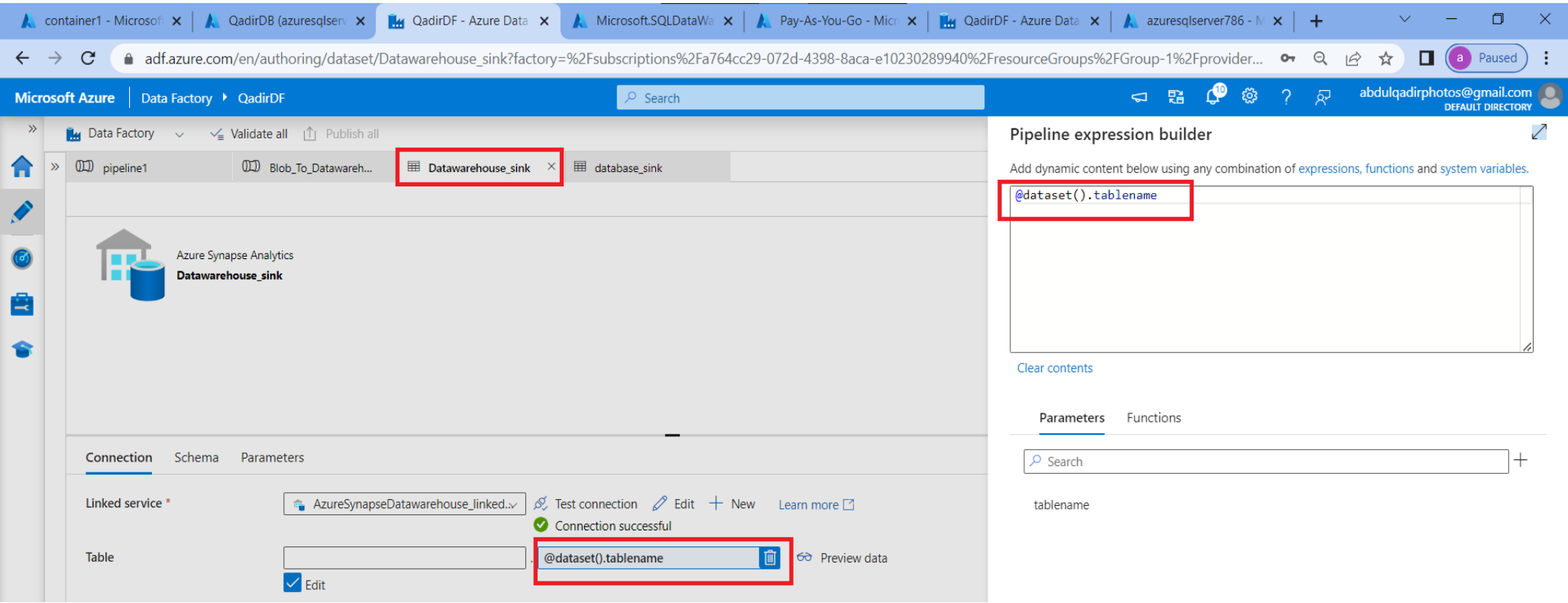
Next, drag a **Copy activity** into **ForEach activity**, Name it as **Copydata1** click on “**Source**” tab choose **Source dataset** select dataset of **Blob_source** pointing towards blob storage container and we will be not mentioning any kind of files will be selecting **Wildcard file path** and then on “**Add dynamic content**” this will take us to the **expression builder** of ADF and choose **ForEach1** where it will define it will choose files from the ForEach activity iterating to one by one.



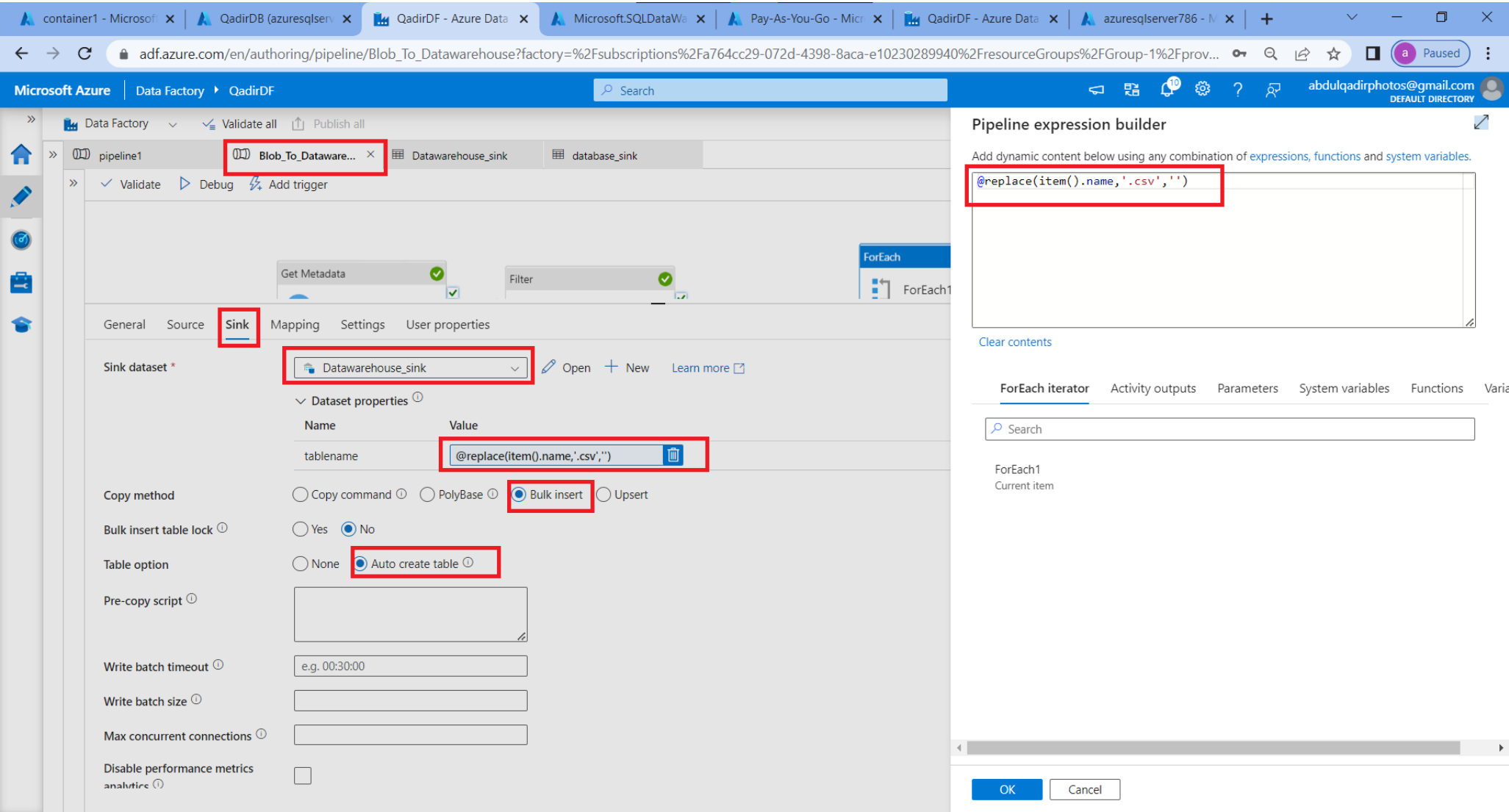
After, on **Copydata1** click on “**Source**” tab choose **Sink dataset** select dataset of **Datawarehouse_sink** pointing towards data warehouse database, and now **Implementing Parameters** sink side, Instead of creating two new datasets and another Copy Data activity, we're going to use parameters in the existing ones. This will allow us to use one single dataset for all our Csv files, Then go to the **Parameters** tab, and create the following one parameters as **tablename**.



Back in the **Connection** tab, click on Table . empty box and then on **"Add dynamic content"** This will take us to the **expression builder** of ADF. Choose the parameter **tablename** from the list below.



If we open the Copy Data activity, we can see ADF asks us now to provide **value** for the parameters we just added click on it and then on **"Add dynamic content"** This will take us to the **expression builder** of ADF and will choosing ForEach iterator and we will be replacing .csv with empty string ' ' , and selected copy method to Bulk insert and selected auto create table in data warehouse database.



10 Debugging the Pipeline:

Start debugging of the pipeline. In the output pane, we can see the Copy Data activity has been run successfully.

The screenshot shows the Microsoft Azure Data Factory portal interface. At the top, there's a navigation bar with the 'Data Factory' tab selected. Below it, the pipeline 'Blob_To_Datwarehouse' is highlighted. The 'Output' tab is active, showing a table of pipeline runs. The table has columns for Name, Type, Run start, Duration, Status, Integration runtime, and Run ID. All runs are marked as 'Succeeded'.

Name	Type	Run start	Duration	Status	Integration runtime	Run ID
Copy data1	Copy data	2023-03-15T17:10:40.47925	00:00:11	Succeeded	AutoResolveIntegrationRun	d6079158-ff25-4f7c-836f-d2
Copy data1	Copy data	2023-03-15T17:10:40.36796	00:00:10	Succeeded	AutoResolveIntegrationRun	27293770-d630-4332-a8f4-e
Copy data1	Copy data	2023-03-15T17:10:40.39834	00:00:32	Succeeded	AutoResolveIntegrationRun	9c86f249-d6c1-4708-b491-f
Copy data1	Copy data	2023-03-15T17:10:40.39072	00:00:32	Succeeded	AutoResolveIntegrationRun	d3956413-f115-4cb2-8382-5
Copy data1	Copy data	2023-03-15T17:10:40.36984	00:00:31	Succeeded	AutoResolveIntegrationRun	164a38a0-f845-4a0d-bd9d-f
ForEach1	ForEach	2023-03-15T17:10:40.00904	00:00:35	Succeeded		8c4da31e-34ef-456e-a6c5-2
Filter1	Filter	2023-03-15T17:10:38.12356	00:00:01	Succeeded		605bfa69-3b26-4cbe-b291-c
Get File Metadata	Get Metadata	2023-03-15T17:10:34.16325	00:00:04	Succeeded	AutoResolveIntegrationRun	a475d5cb-98fe-4a07-8d73-e

Details Refresh

Learn more on copy performance details from here.

Activity run id: 164a38a0-f845-4a0d-bd9d-f4406ba2e8ce

Succeeded

Source: Azure Blob Storage
Region: Central India

Destination: Azure Synapse Analytics
Region: Central India

Metric	Value
Data read: ⓘ	6.107 MB
Files read: ⓘ	1
Rows read: ⓘ	89,253
Peak connections: ⓘ	3

Metric	Value
Data written: ⓘ	10,606 MB
Rows written: ⓘ	89,253
Peak connections: ⓘ	3

Copy duration: 00:00:29
Throughput: ⓘ 265.501 KB/s

▼ Azure Blob Storage → Azure Synapse Analytics

Start time: 3/15/2023, 10:40:40 PM
Used DIUs ⓘ: 4
Used parallel copies ⓘ: 1

▼ Duration: 00:00:29

Details	Working duration	Total duration
Queue ⓘ		00:00:04
Pre copy script ⓘ		00:00:00
Transfer ⓘ		
Listing source ⓘ	00:00:00	
Reading from source ⓘ	00:00:00	
Writing to sink ⓘ	00:00:19	00:00:23

Data consistency verification ⓘ: Not verified

How satisfied or dissatisfied are you with the performance of this copy activity?

★ ★ ★ ★ ★

container1 - Microsoft...QadirDB (azuresqlserv...QadirDF - Azure Data...Microsoft.SQLDataW...Pay-As-You-Go - Micr...QadirDF - Azure Data...azuresqlserver786 - M...+...-...x

adf.azure.com/en/authoring/pipeline/Blob_To_Datawarehouse?factory=%2Fsubscriptions%2Fa764cc29-072d-4398-8aca-e10230289940%2FresourceGroups%2FGroup-1%2Fprov...Paused

DetailsRefresh

Learn more on copy performance details from here.

Activity run id: d3956413-f115-4cb2-8382-5df5fa18d481

Azure Blob Storage
Region: Central India

Succeeded

Azure Synapse Analytics
Region: Central India

Data read: ⓘ10.475 MB

Files read: ⓘ1

Rows read: ⓘ100,000

Peak connections: ⓘ4

Data written: ⓘ18.751 MB

Rows written: ⓘ100,000

Peak connections: ⓘ3

Copy duration00:00:30

Throughput: ⓘ436.476 KB/s

▼ Azure Blob Storage → Azure Synapse Analytics

Start time3/15/2023, 10:40:40 PM

Used DIUs ⓘ4

Used parallel copies ⓘ1

▼ Duration00:00:30

Details	Working duration	Total duration
Queue ⓘ		00:00:04
Pre copy script ⓘ		00:00:01
Transfer ⓘ	<div><div>Listing source ⓘ00:00:00</div><div>Reading from source ⓘ00:00:00</div><div>Writing to sink ⓘ00:00:20</div></div>	00:00:23

Data consistency verification ⓘNot verified

How satisfied or dissatisfied are you with the performance of this copy activity?

★★★★★

container1 - Microsoft...QadirDB (azuresqlserv...QadirDF - Azure Data...Microsoft.SQLDataW...Pay-As-You-Go - Micr...QadirDF - Azure Data...azuresqlserver786 - M...+...-...x

adf.azure.com/en/authoring/pipeline/Blob_To_Datawarehouse?factory=%2Fsubscriptions%2Fa764cc29-072d-4398-8aca-e10230289940%2FresourceGroups%2FGroup-1%2Fprov...Paused

DetailsRefresh

Learn more on copy performance details from here.

Activity run id: 9c86f249-d6c1-4708-b491-fcc5abdceb5c

Azure Blob Storage
Region: Central India

Succeeded

Azure Synapse Analytics
Region: Central India

Data read: ⓘ6.919 MB

Files read: ⓘ1

Rows read: ⓘ99,859

Peak connections: ⓘ3

Data written: ⓘ12.041 MB

Rows written: ⓘ99,859

Peak connections: ⓘ3

Copy duration00:00:30

Throughput: ⓘ276.778 KB/s

▼ Azure Blob Storage → Azure Synapse Analytics

Start time3/15/2023, 10:40:40 PM

Used DIUs ⓘ4

Used parallel copies ⓘ1

▼ Duration00:00:30

Details	Working duration	Total duration
Queue ⓘ		00:00:04
Pre copy script ⓘ		00:00:01
Transfer ⓘ	<div><div>Listing source ⓘ00:00:00</div><div>Reading from source ⓘ00:00:00</div><div>Writing to sink ⓘ00:00:20</div></div>	00:00:24

Data consistency verification ⓘNot verified

How satisfied or dissatisfied are you with the performance of this copy activity?

★★★★★

We've now successfully loaded all csv files to an Azure data warehouse database by using one single pipeline.

21 | Page

container1 - Microsoft | QadirDB (azuresqlserv... | QadirDF - Azure | Pay-As-You-Go - Micr... | QadirDW - Azure | azuresqlserver786 - M... | +

portal.azure.com/?quickstart=True#@abdulqadirphotosgmail.onmicrosoft.com/resource/subscriptions/a764cc29-072d-4398-8aca-e10230289940/resourceGroups/Group-1... Paused

Microsoft Azure Search resources, services, and docs (G+)

Home > Microsoft.SQLDataWarehouse.NewDatabaseExistingServerV4_dd3dbc64e | Overview > QadirDW (azuresqlserver786/QadirDW)

QadirDW (azuresqlserver786/QadirDW) | Query editor (preview) ☆

Dedicated SQL pool (formerly SQL DW)

Login + New Query ↑ Open query Feedback

QadirDW (admin@1@azures...

Showing limited object explorer here. For full capability please click here to open Azure Data Studio.

Tables

- dbo.part-00000-6b9aa0de-eb74-440b-b4c7-bb08fa673ed9-c000
- dbo.part-00000-6d14c9ba-eca1-456c-b922-391595326386-c000
- dbo.part-00000-b89ae20b-ac27-4060-8ce3-2c28ba1c5f83-c000

Views

Stored Procedures

Query 1 × Query 2 × **Query 3 ×** Query 4 ×

Run ☐ Cancel query Save query Export data as Show only Editor

```
1 SELECT TOP (1000) * FROM [dbo].[part-00000-6d14c9ba-eca1-456c-b922-391595326386-c000]
```

Results Messages

Search to filter items...

TransactionID	ProductID	ReferenceOrderID	ReferenceOrderLineID	TransactionDate	TransactionType	Quantity	ActualCost	ModifiedDate
111148	998	55229	1	2013-08-28 18:30:00	S	1	539.99	2013-08-28 18:30:00
111954	858	55264	2	2013-08-29 18:30:00	S	4	15.9185	2013-08-29 18:30:00
112592	958	55300	10	2013-08-29 18:30:00	S	2	534.492	2013-08-29 18:30:00
113364	707	55428	3	2013-08-30 18:30:00	S	1	34.99	2013-08-30 18:30:00
114138	827	44852	0	2013-09-02 18:30:00	W	3	0.0	2013-09-02 18:30:00
114903	922	55733	3	2013-09-04 18:30:00	S	1	3.99	2013-09-04 18:30:00
115646	878	55907	1	2013-09-07 18:30:00	S	1	21.98	2013-09-07 18:30:00
116379	708	56048	3	2013-09-10 18:30:00	S	1	34.99	2013-09-10 18:30:00
117176	823	45923	0	2013-09-14 18:30:00	W	2	0.0	2013-09-14 18:30:00

Query succeeded | 2s

22 | Page

