Business Requirement

Scenario: A national weather monitoring organization (ABC), aims to enhance real-time weather analysis for operational efficiency and public safety. The organization needs a robust system to ingest, process, and analyze real-time weather data from sensors distributed across the country. The processed data must provide actionable insights, such as average weather metrics, trends, and conditions, for internal stakeholders and public users.

The primary objectives are:

1. **Real-Time Monitoring**: Collect and process live weather data, including temperature, humidity, wind speed, and precipitation, to ensure immediate responses to severe weather events.
2. **Data Reliability**: Store the streaming data with high accuracy for historical analysis and future predictions.
3. **Actionable Insights**: Aggregate data for trend analysis over time intervals to support decision-making.
4. **Public Dashboard**: Enable visualization of real-time weather trends via an interactive dashboard for internal stakeholders and public safety alerts.

Solution

Solution Overview: The Azure-based real-time streaming solution efficiently addresses the business requirement by leveraging structured streaming, scalable cloud services, and interactive visualizations.

1. **Data Ingestion with Azure Event Hub:**
    - The solution collects real-time weather data using Azure Event Hub with a throughput unit designed for a high volume of events.
    - Data is stored temporarily for immediate streaming to processing pipelines.
2. **Data Processing in Azure Databricks:**
    - **Bronze Layer**: Ingests raw weather data and persists it in Delta format for reliability and reprocessing, ensuring all data is stored with exactly-once semantics.
    - **Silver Layer**: Cleans and transforms the data, converting JSON structures into a structured tabular format with specific fields like temperature, humidity, wind speed, and conditions.
    - **Gold Layer**: Aggregates weather metrics into 5-minute intervals, providing summarized insights for trends and historical analysis.
3. **Data Visualization with Power BI:**

    - **Silver Layer Visualization**: Offers near-real-time data monitoring for immediate insights.
    - **Gold Layer Visualization**: Enables analytical dashboards with aggregated metrics, supporting trend analysis over time.
4. **Scalability & Fault Tolerance:**
    - Implemented checkpointing and Delta Lake for reliable and fault-tolerant data streaming.
    - Scalable architecture allows increasing throughput units or clusters to handle spikes in sensor data volume.
5. **Business Outcomes:**
    - **Enhanced Weather Awareness**: Real-time data visualization enables proactive responses to changing weather conditions.
    - **Efficient Decision-Making**: Aggregated metrics support better strategic planning for weather-dependent activities.
    - **Improved Public Safety**: Accurate and timely weather data can trigger alerts, reducing risks associated with extreme weather.

This solution demonstrates how Azure's real-time streaming capabilities, Databricks' processing power, and Power BI's visualization tools work cohesively to meet ClimaSecure's business needs, ensuring both immediate responsiveness and strategic weather insights.

Steps I followed:

I created a resource group in Azure with the name “streaming-rg”. This resource group consists of all the azure resources that I will use in this project.In the resource group, I created an event hub service. I gave a name to the namespace “real time-streaming”. I choose the same region in which I have created my resource group (UK south). In the pricing tier, I selected “Basic” . The “Basic” pricing tier allows only 1 consumer group and message retention period is 1 day. This means that event hubs store the messages only for 1 day. The “Basic” pricing tier does not have the “Capture” capability.
“Capture” capability allows to automatically capture the streaming data and store it to Azure data lake storage for processing and analytics.
For the throughput units, I have selected “1”

Throughput units (TUs) in Azure Event Hub represent a capacity measure used to determine the scale of data ingress (incoming data) and egress (outgoing data) for an Event Hub namespace. They are a key part of managing and scaling your Event Hub service. Each throughput unit provides a certain level of performance capacity for data streaming and is billed on a per-hour basis.

Key Characteristics of Throughput Units

1. **Data Ingress (Incoming Events):**

    - Each throughput unit allows for **up to 1 MB/sec of data ingress** or 1,000 events per second (whichever is reached first).
    - This includes all data sent to an Event Hub instance within the namespace.
2. **Data Egress (Outgoing Events):**
    - Each throughput unit allows for **up to 2 MB/sec of data egress**.
    - This includes data read from partitions by consumer applications or through Azure Stream Analytics.
3. **Scalability:**
    - You can scale up the number of throughput units to handle larger volumes of data. The maximum number depends on your pricing tier (Basic, Standard, or Dedicated).
    - For example, the Standard tier typically supports up to 20 throughput units by default (more available upon request).
4. **Shared Across Namespace:**
    - Throughput units are shared across all Event Hubs within a single namespace, making it a cost-effective way to manage multiple streams.
5. **Burst Mode:**
    - If data exceeds the allotted throughput unit capacity, Event Hub may throttle requests, causing potential delays or dropped messages.

How to Determine Throughput Units?

The number of throughput units you need depends on:

- **Event Size**: The size of the events being sent (in kilobytes).
- **Event Frequency**: The number of events per second being sent to or read from the Event Hub.
- **Number of Consumers**: The volume of data being consumed by receivers also counts toward egress.

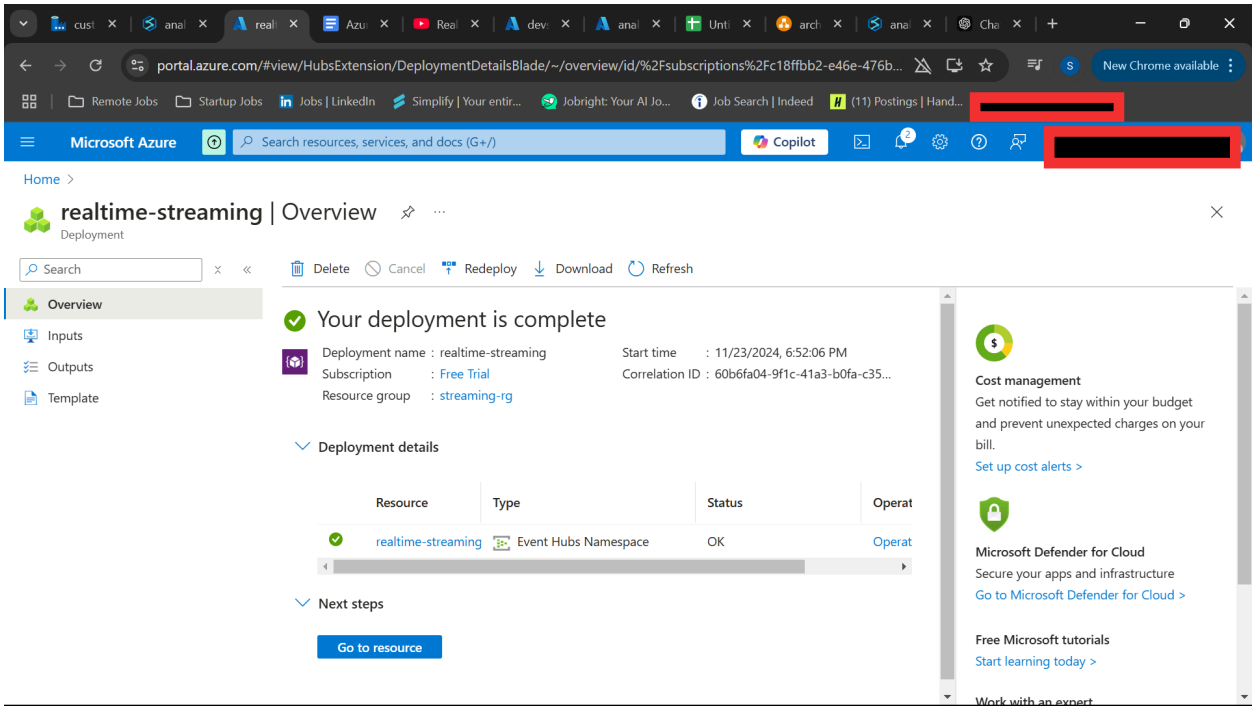
Example Use Case:

- You have an application sending 500 KB of data every second to Event Hub.
- At the same time, you have a consumer application reading the same amount of data.
- For 1 MB/sec ingress and 1 MB/sec egress, you'd need **at least 1 throughput unit**.

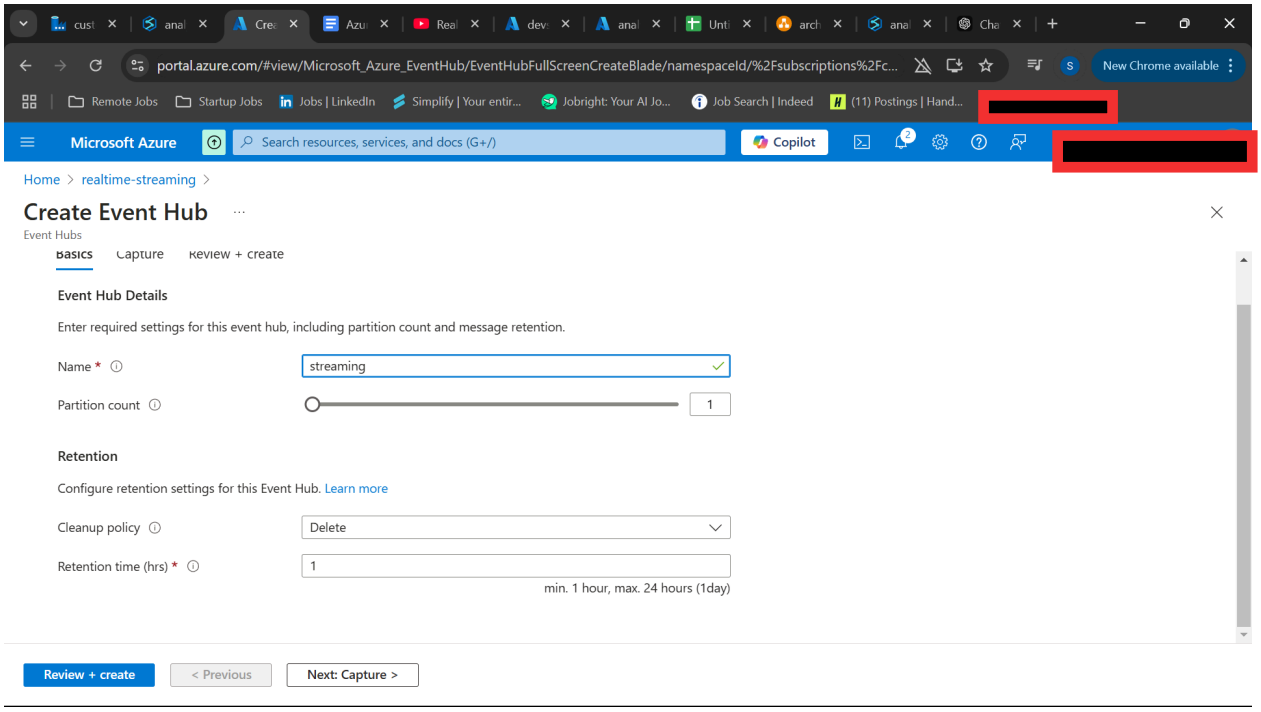
Pricing Impact:

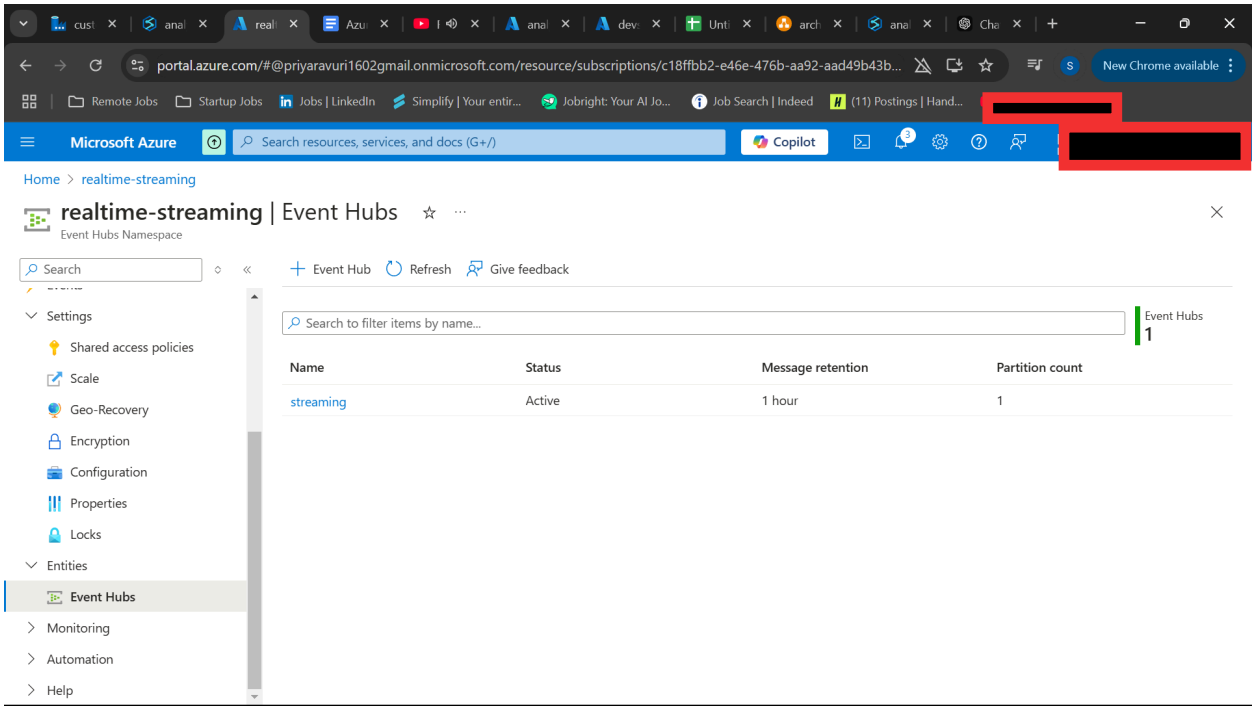
Throughput units affect the cost of running your Event Hub service. Make sure to monitor usage and scale up or down as necessary to optimize performance and costs.

I created the resource once the validation is completed.

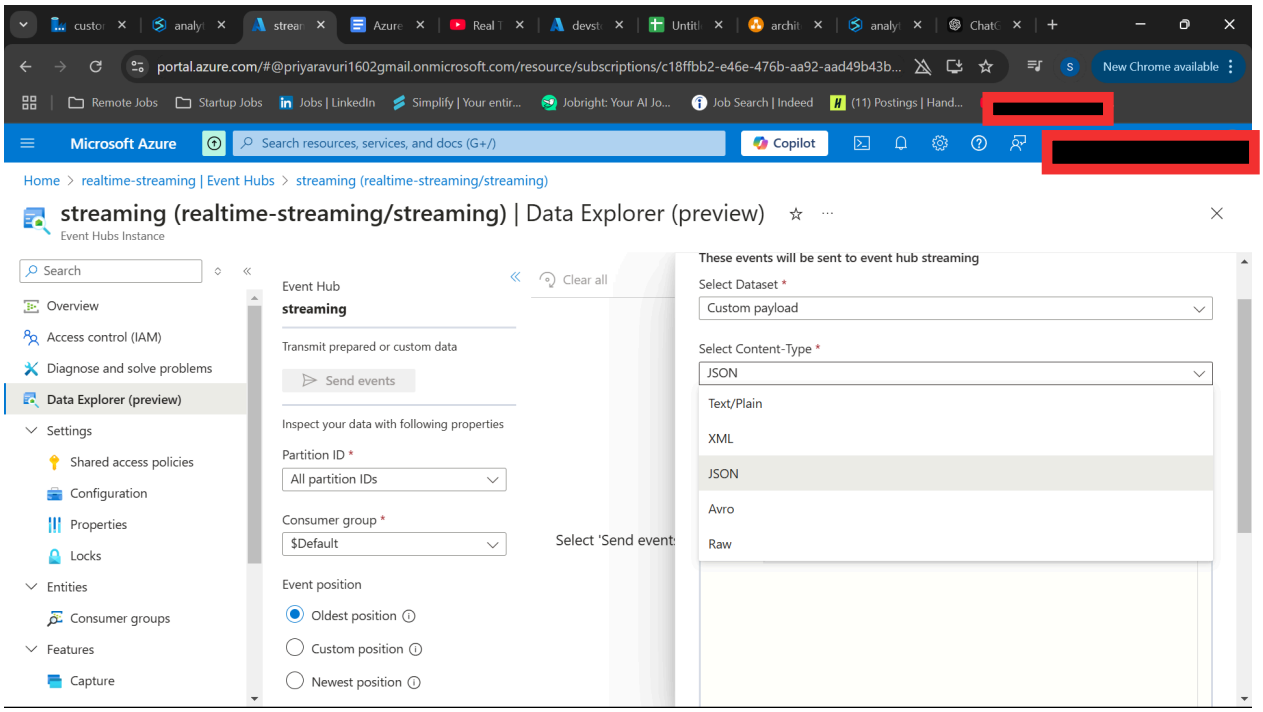


Once the namespace creation is completed, I created an Event Hub in the namespace with the name “streaming”. The partition count is set to “1” and the retention time is set to 1 hour (this can be adjusted) as shown below

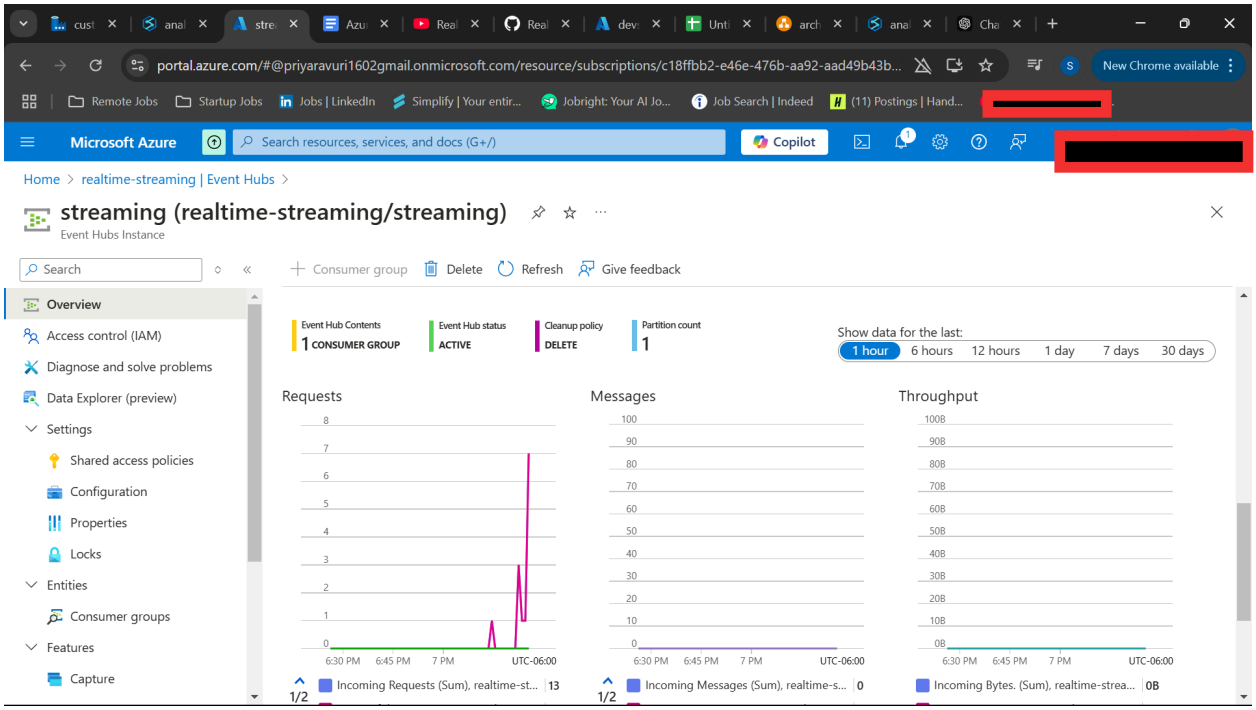




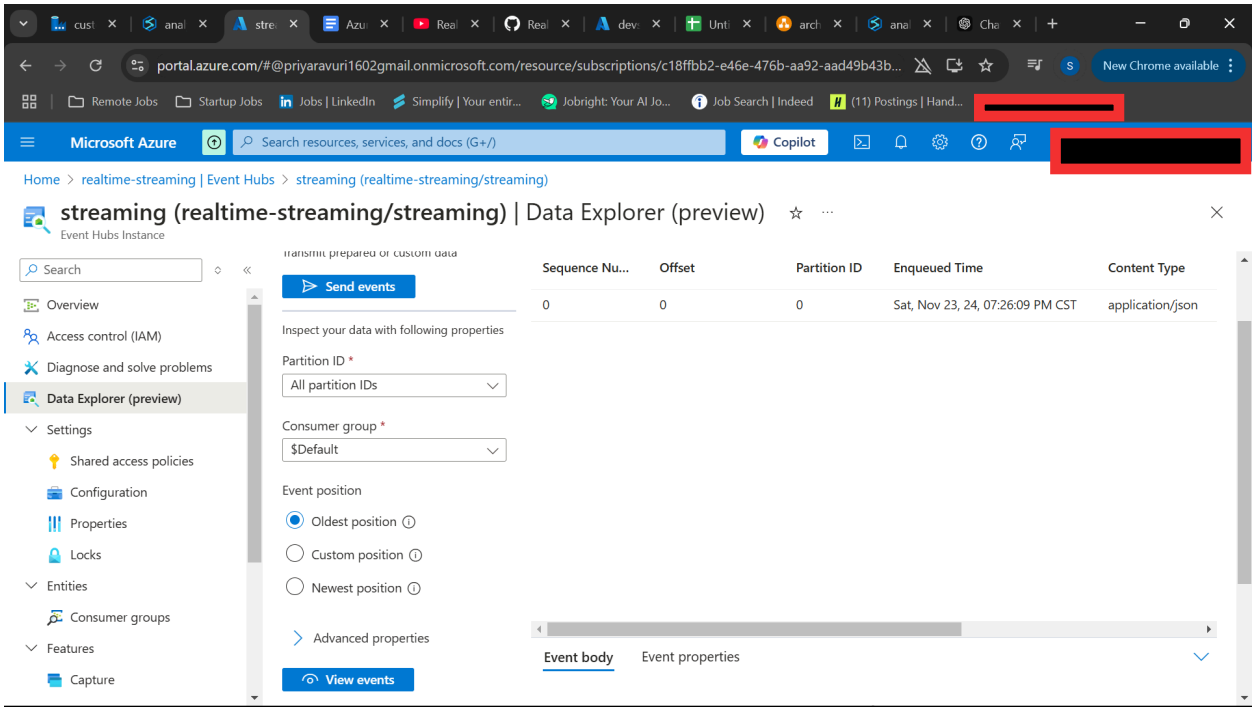
Event hubs also allows to generate data in a desired format such as binary, JSON, XML as shown below



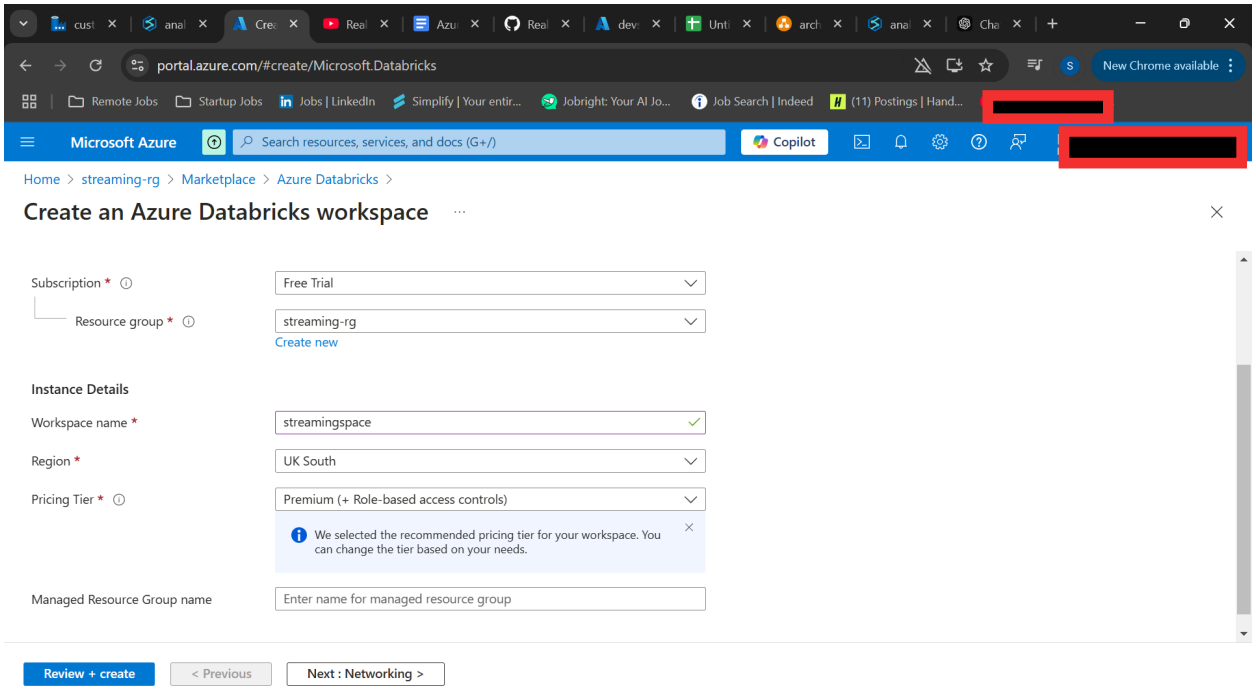
Once I send an event, I can see a spike in the requests

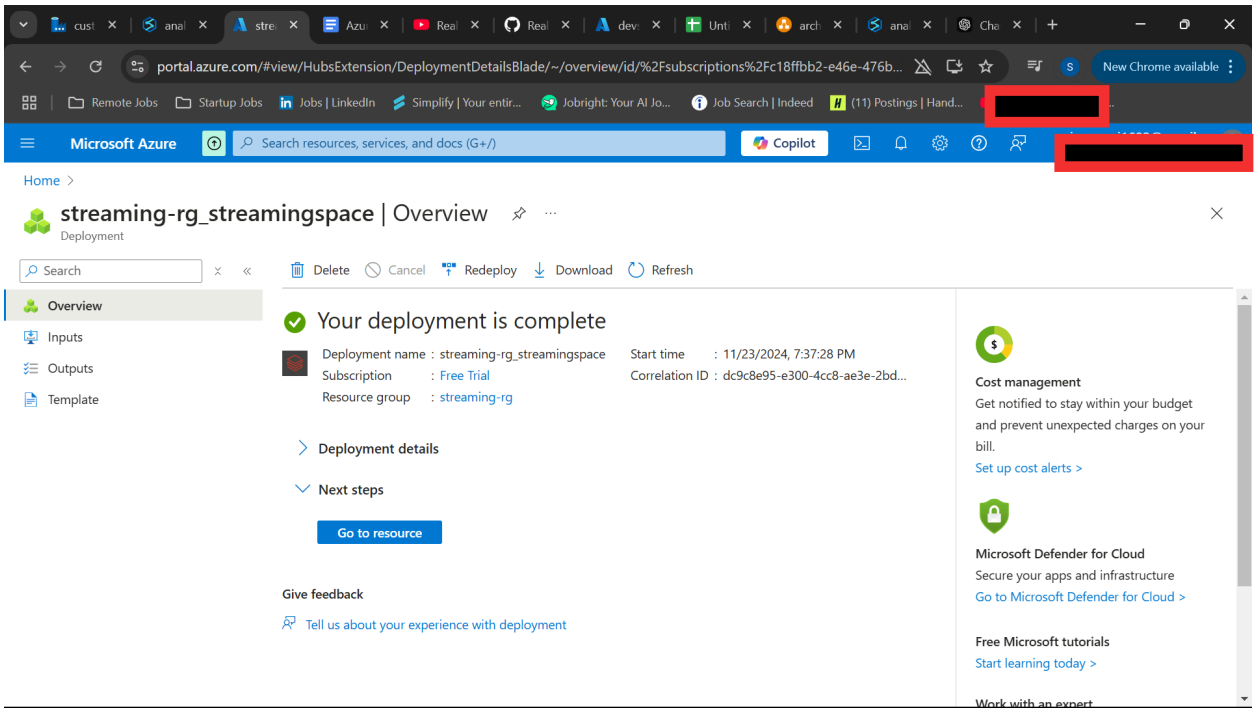


I can also see the event as shown below

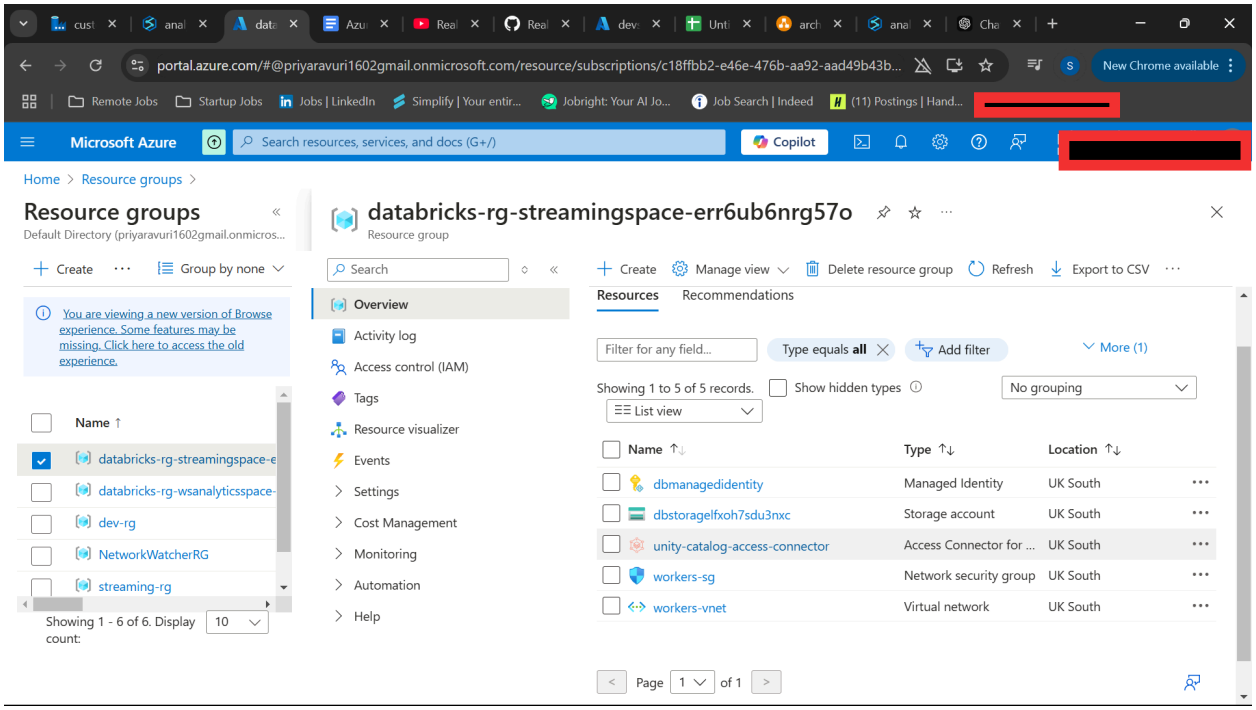


Now, I proceeded to create an azure databricks service with the following configuration

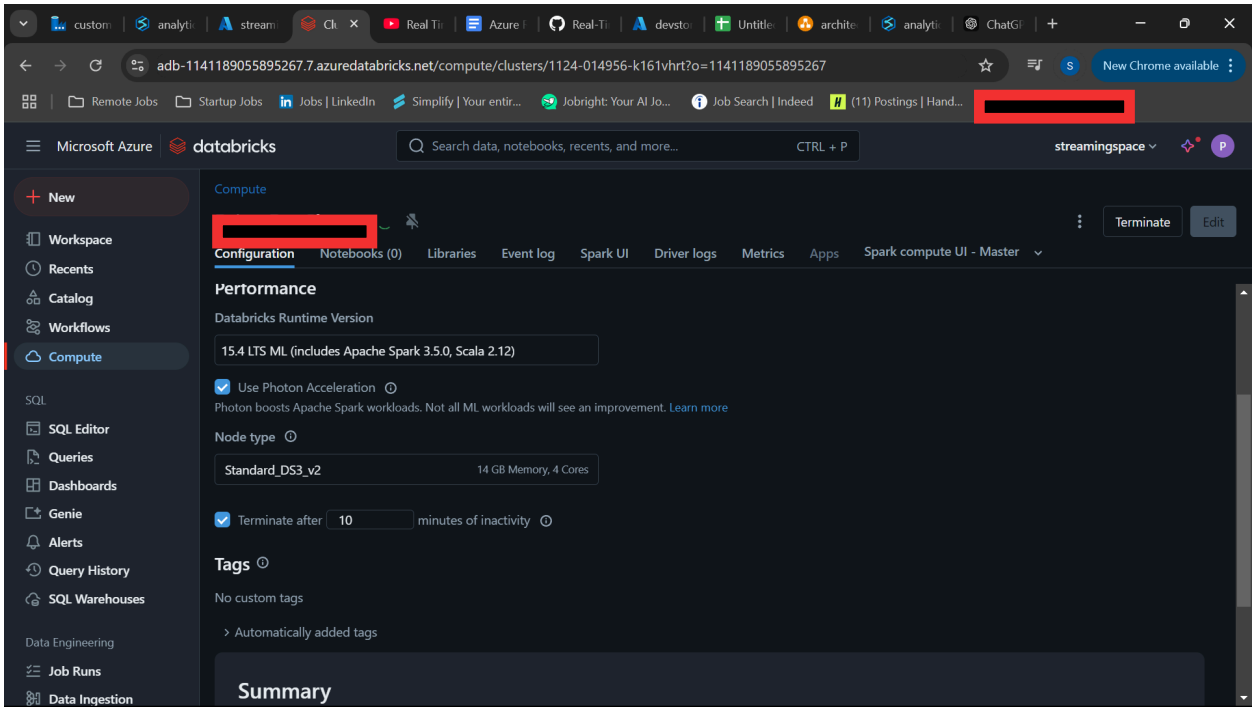




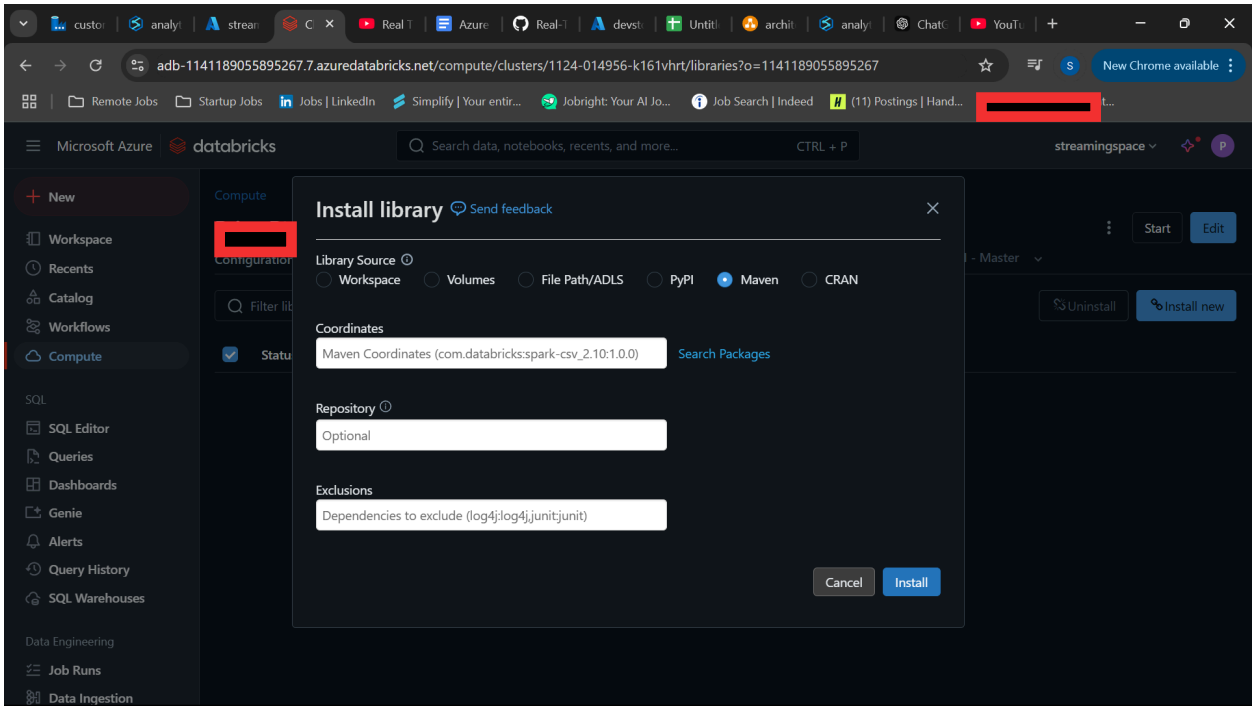
Once the Azure databricks service is deployed, I can see one more resource group which is automatically provisioned with the name "databricks-rg-streamingspace-err6ub6nrg57o". This is the resource group that will be used by databricks. It has the following resources provisioned

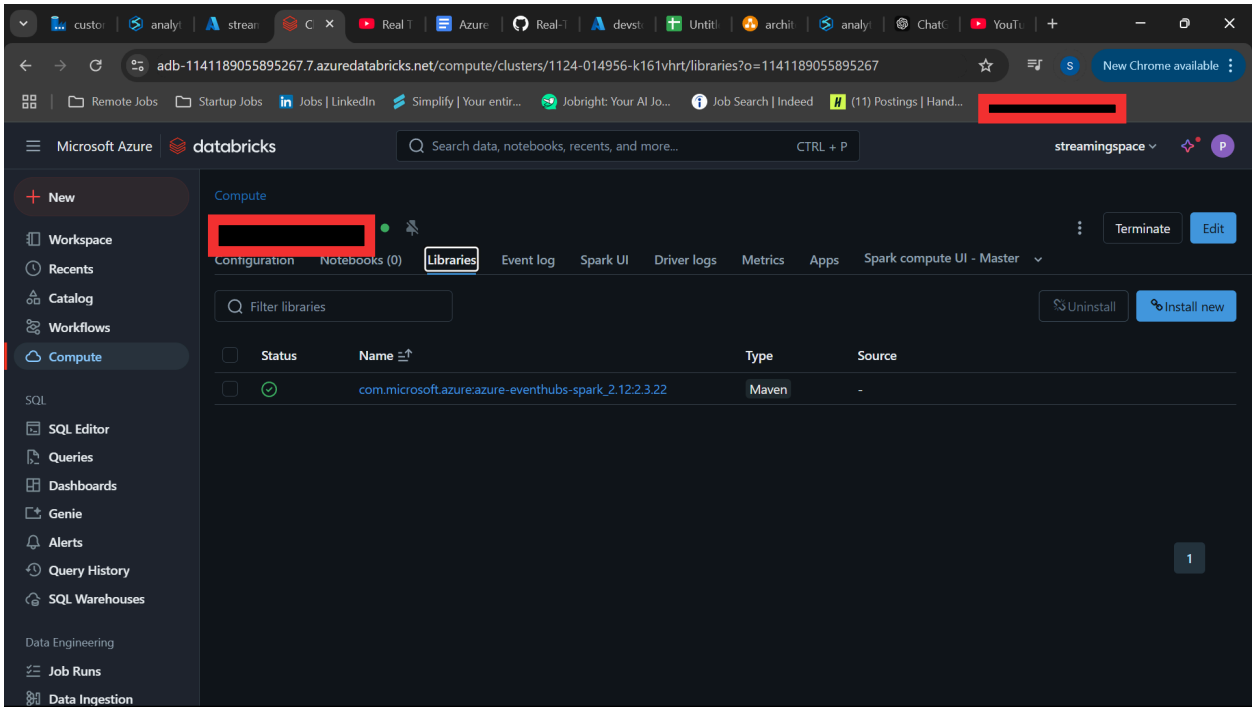


I launched the workspace and created a single node cluster with the following configuration

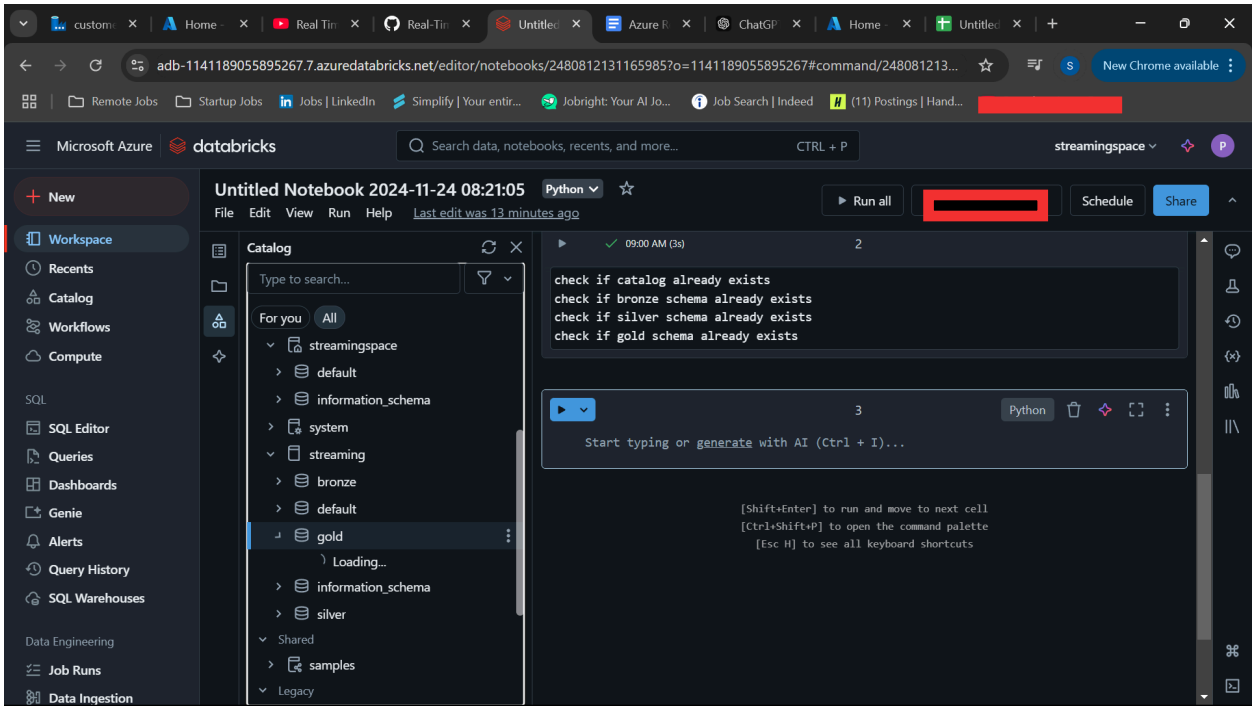


Then I installed an event hub for spark maven library on this cluster as shown below

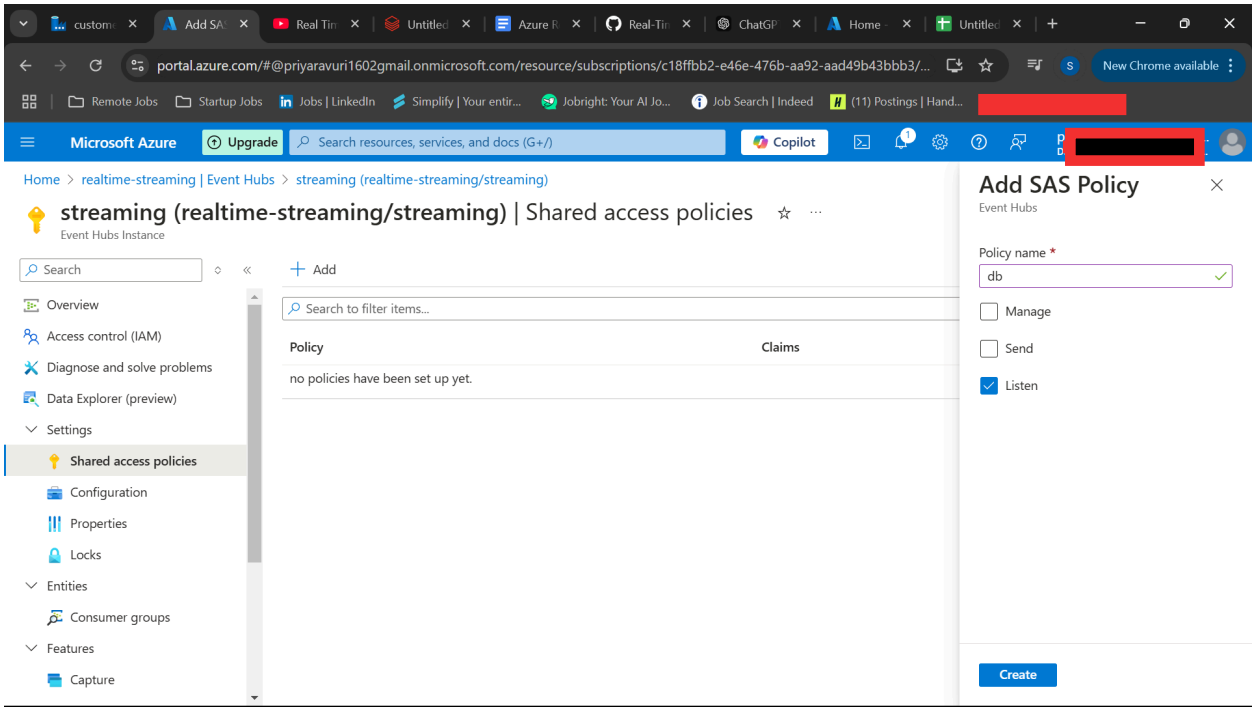




Then,I created a catalog with the name “streaming” and I created three schemas with the names “Bronze”, ‘silver”, “gold”



I created a shared access policy with the name “db” and “Listen” permission in the event hub which I created earlier.



I copied the “Connection string–primary key” and the event hub name and pasted it in the below code in the databricks notebook

```
# Config
# Replace with your Event Hub namespace, name, and key
```

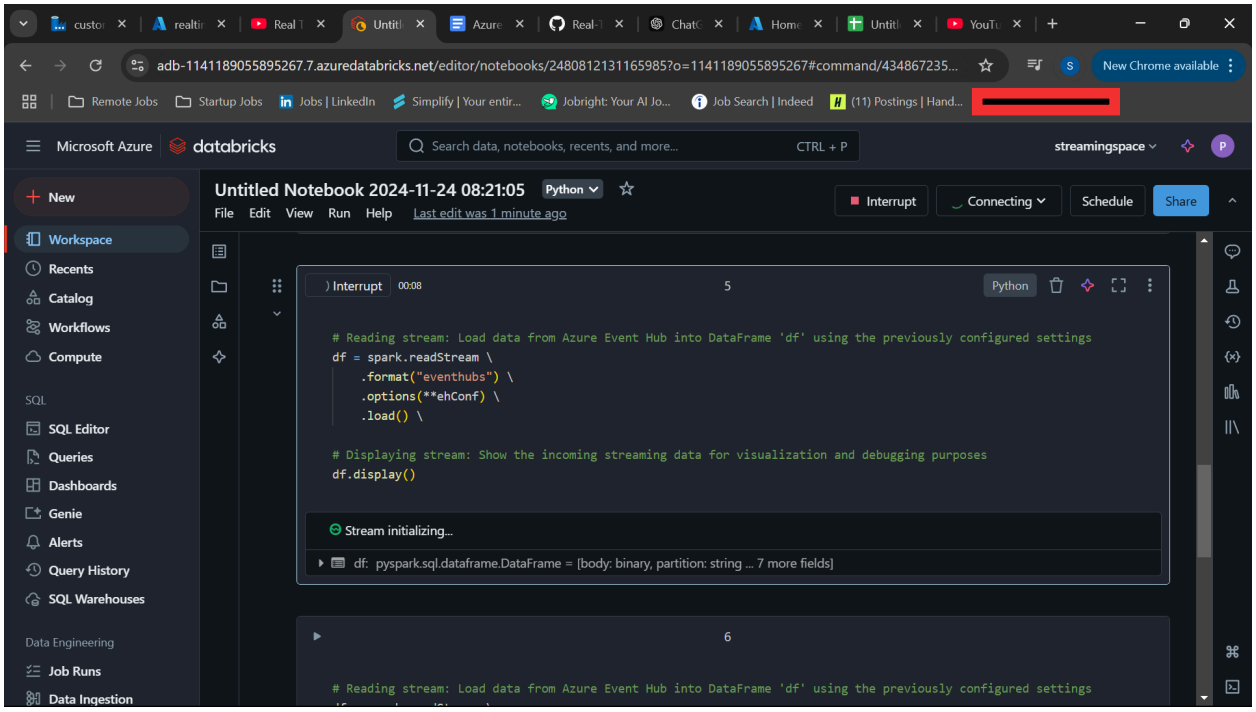
```
connectionString = "Paste connection string here"
eventHubName = "Type event hub name here"

ehConf = {
  'eventhubs.connectionString' :
sc._jvm.org.apache.spark.eventhubs.EventHubsUtils.encrypt(connectionString),
  'eventhubs.eventHubName': eventHubName
}
```

I initiated the read stream with the below code

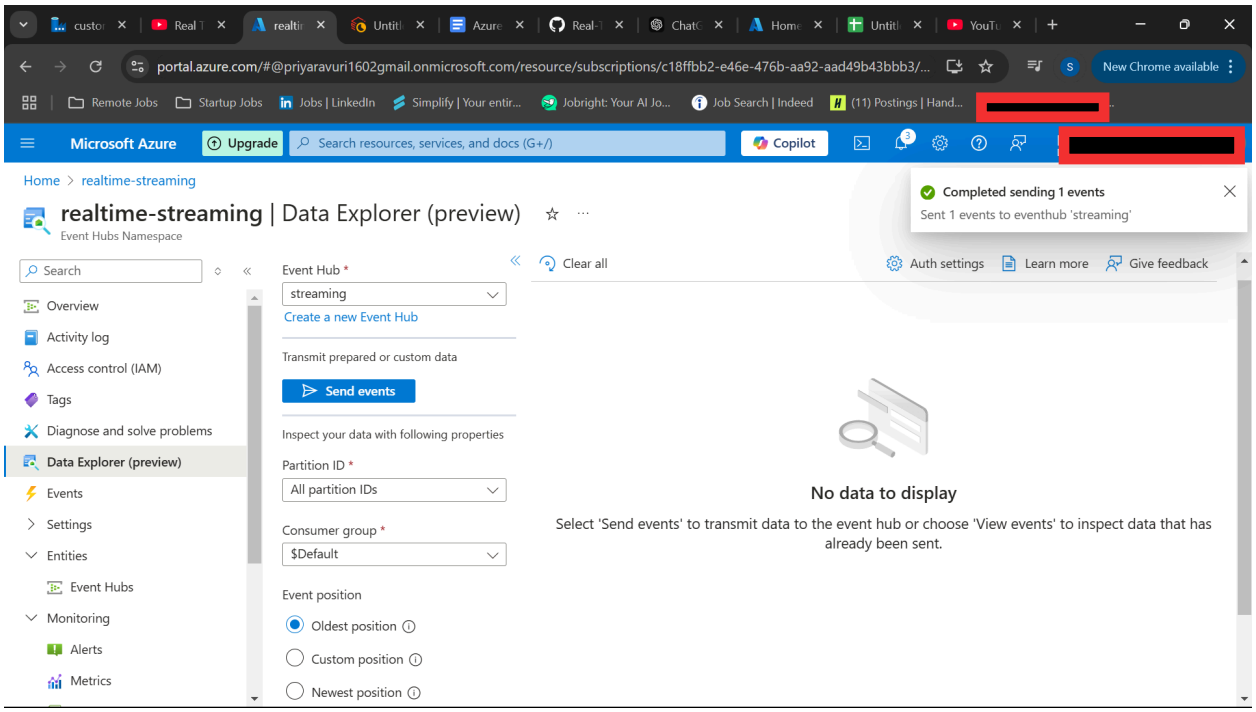
```
# Reading stream: Load data from Azure Event Hub into DataFrame 'df' using the previously configured settings
df = spark.readStream \
    .format("eventhubs") \
    .options(**ehConf) \
    .load() \

# Displaying stream: Show the incoming streaming data for visualization and debugging purposes
df.display()
```

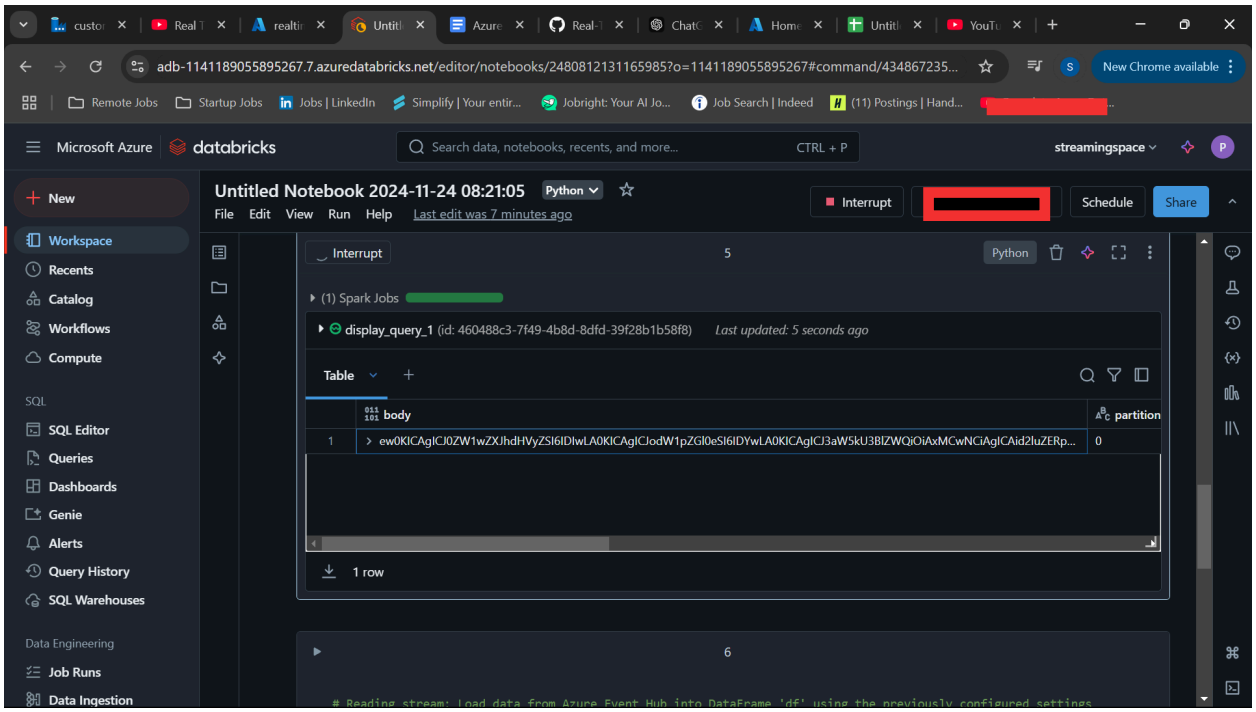


I can see that no data is generated in the stream as I haven't generated any new data. The stream is a live stream, so it won't show the historical data I have generated earlier.

Now, I created one more event in azure event hub to check if i can see live stream here in the databricks notebook



Now, I can see the live stream in databricks notebook as below

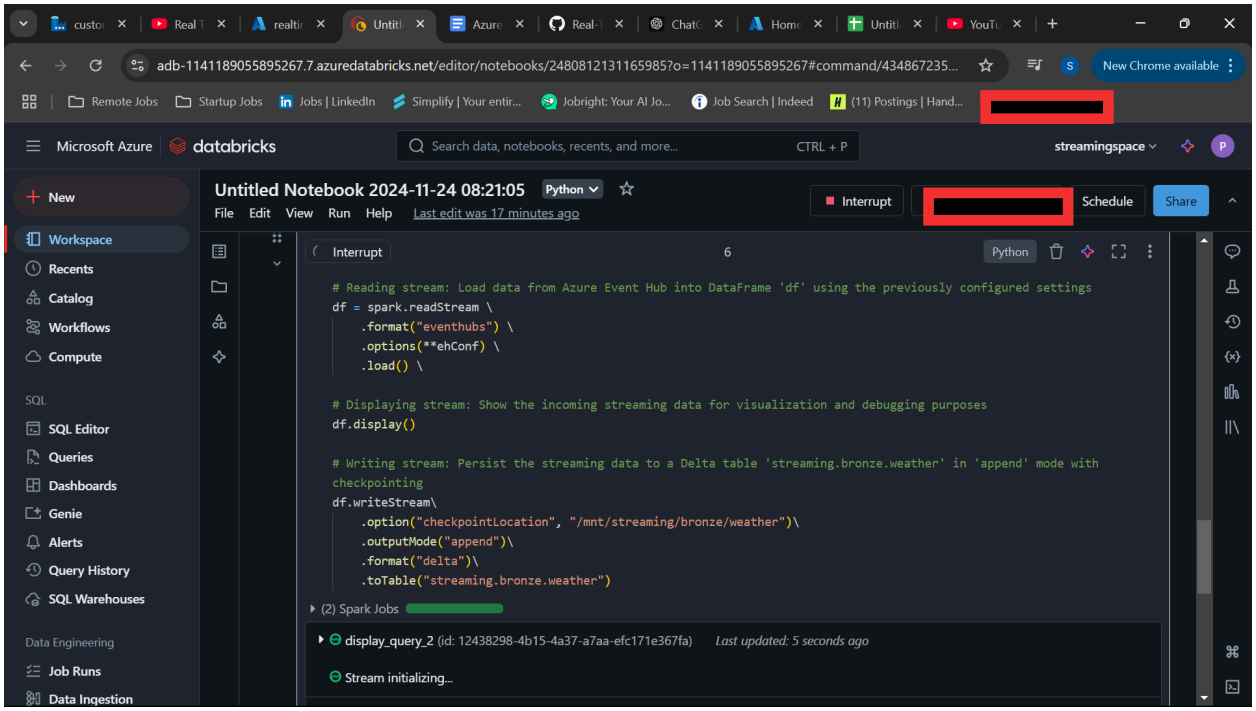


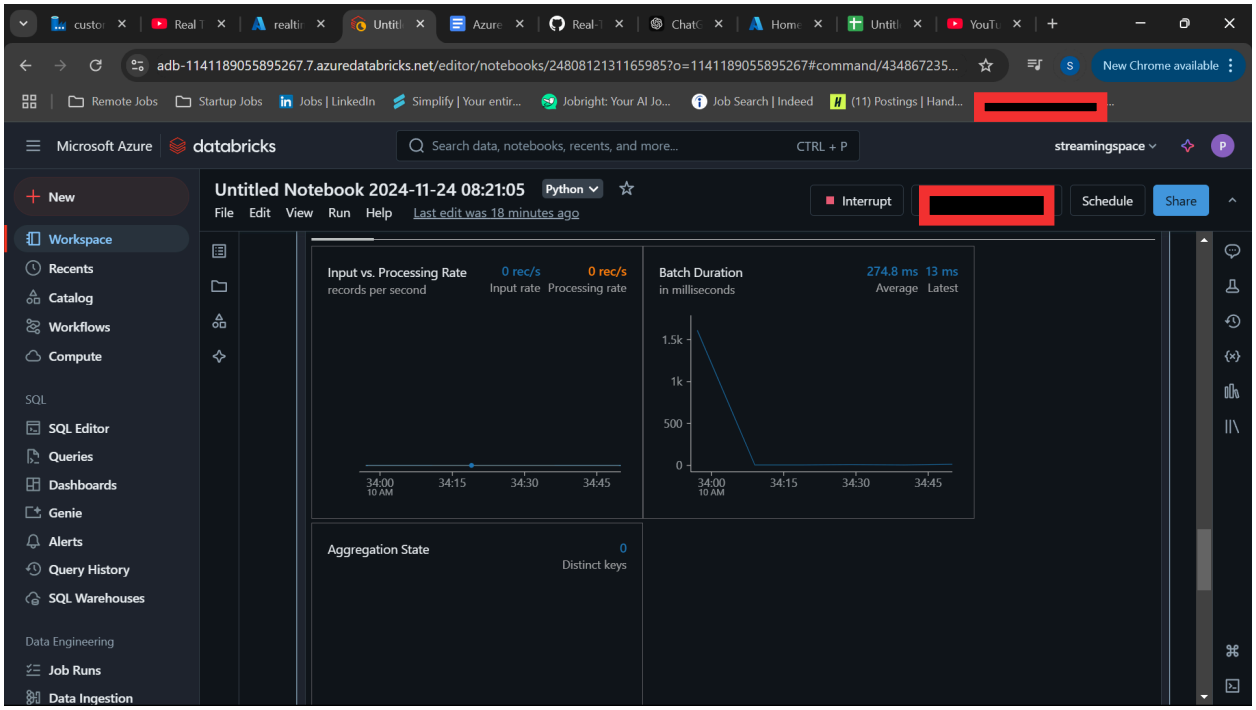
When I am using the readstream method, the data generated is temporary, it won't persist in databricks environment. The data will build up in the stream while the stream is active.

To persist the data, I wrote to storage using the writestream method using the below code

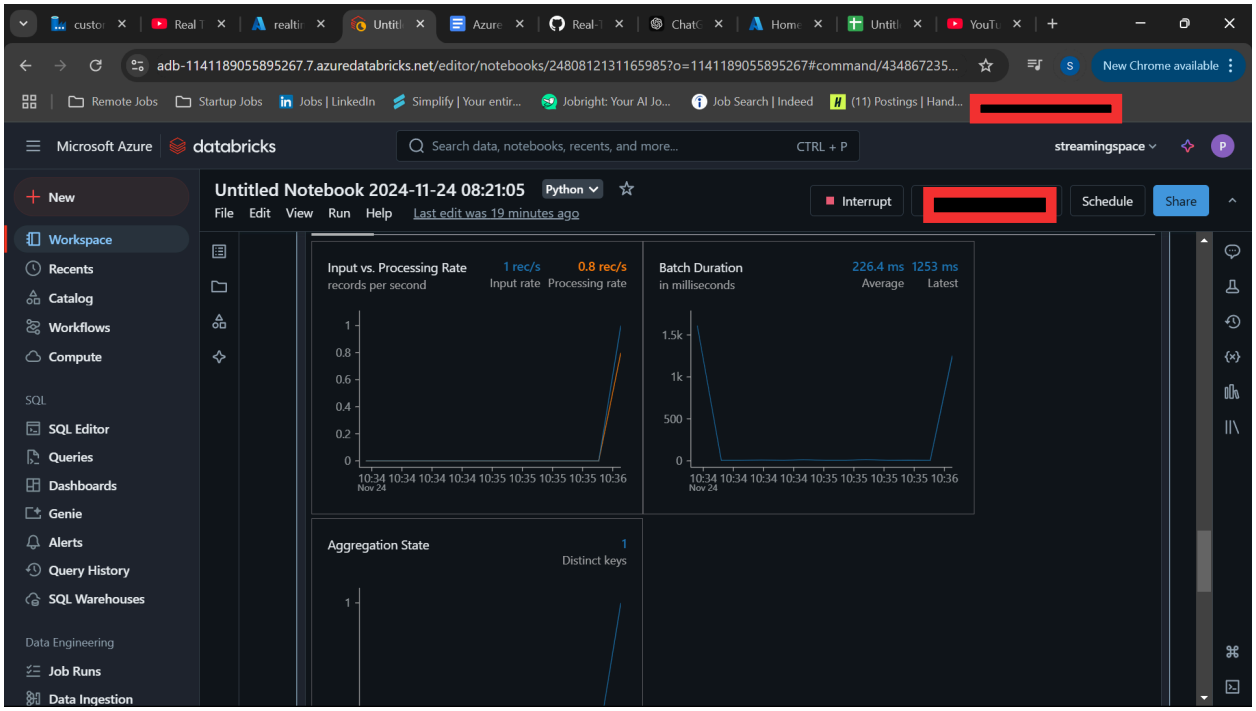
```
# Writing stream: Persist the streaming data to a Delta table 'streaming.bronze.weather' in 'append' mode with checkpointing
df.writeStream\
    .option("checkpointLocation", "/mnt/streaming/bronze/weather")\
    .outputMode("append")\
    .format("delta")\
    .toTable("streaming.bronze.weather")
```

I specified a checkpoint location in the databricks file system (DBFS) to store the state and progress of the streaming job.
There are different output modes such as “Complete”, “Append”, “Update”
I can see that the stream is initialized

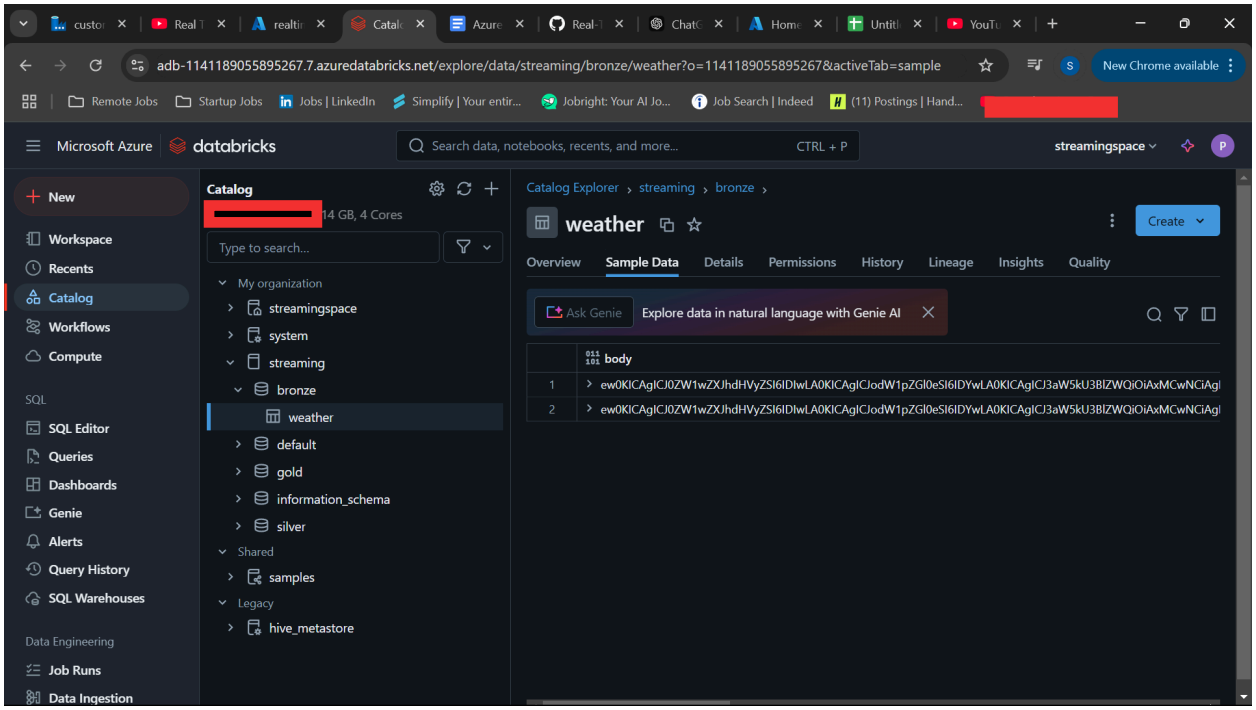




As soon as I generate event , I can see the below image

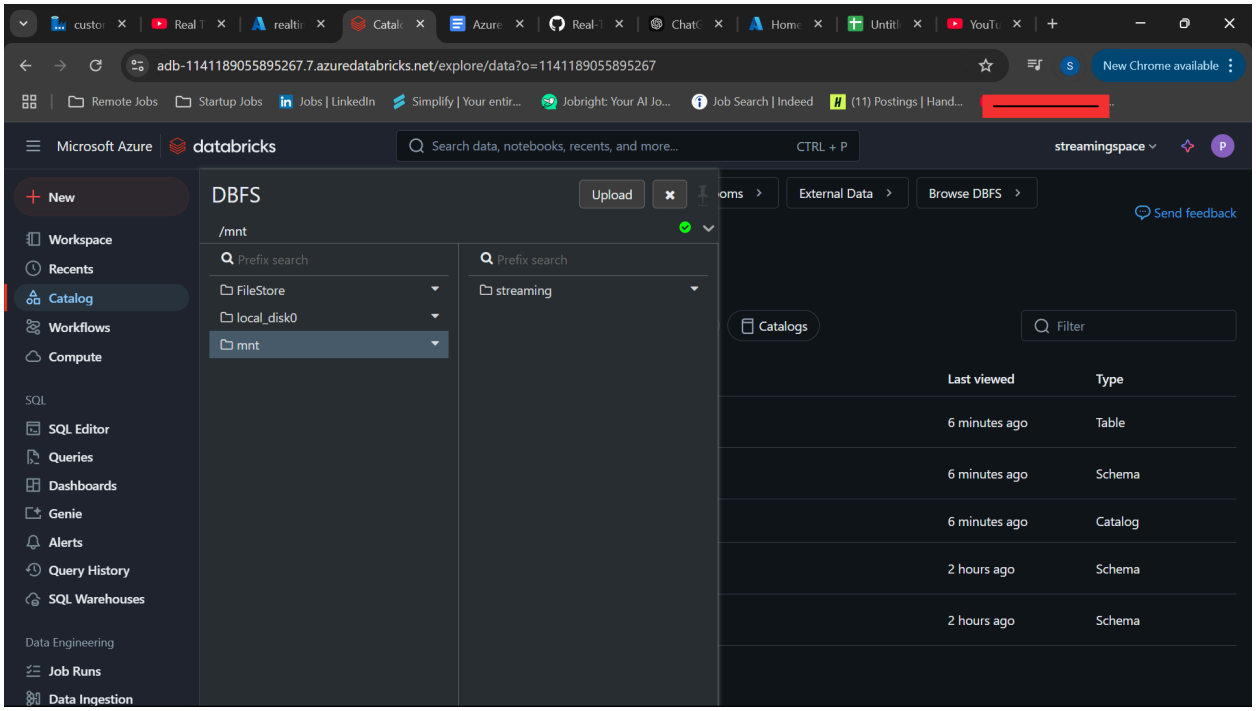


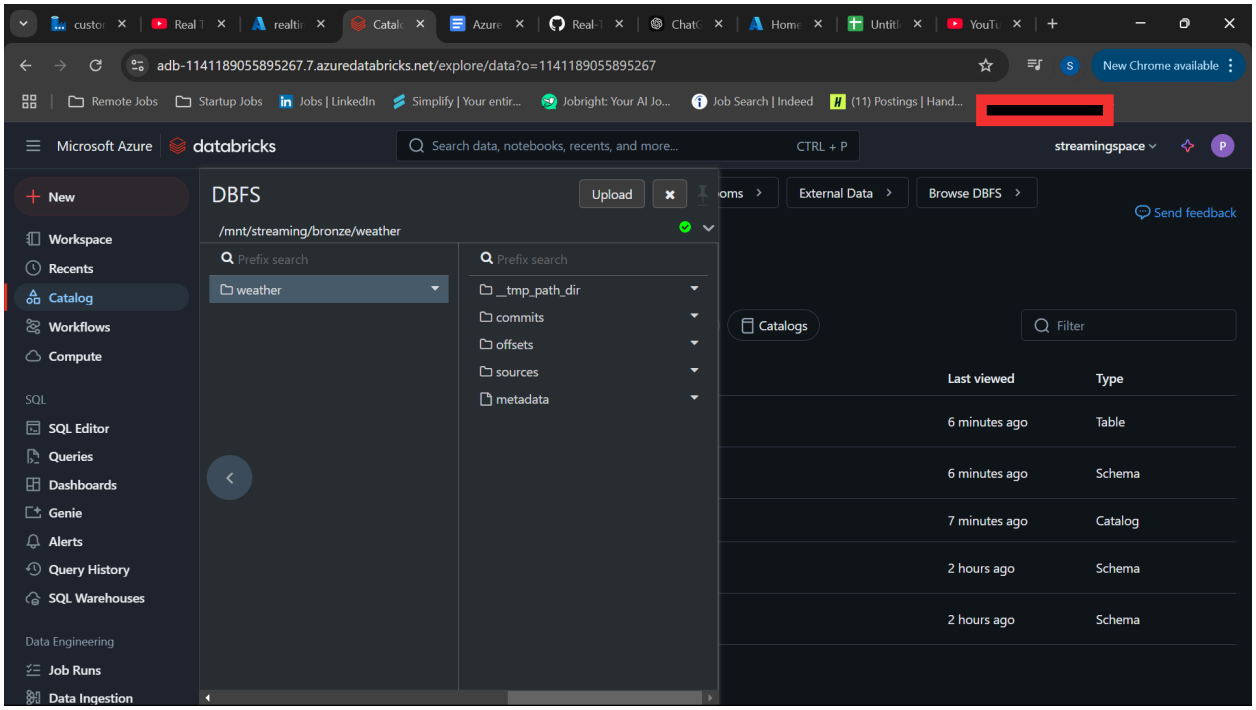
I can also see the table “weather” is created in the bronze layer with the sample data generated as shown below



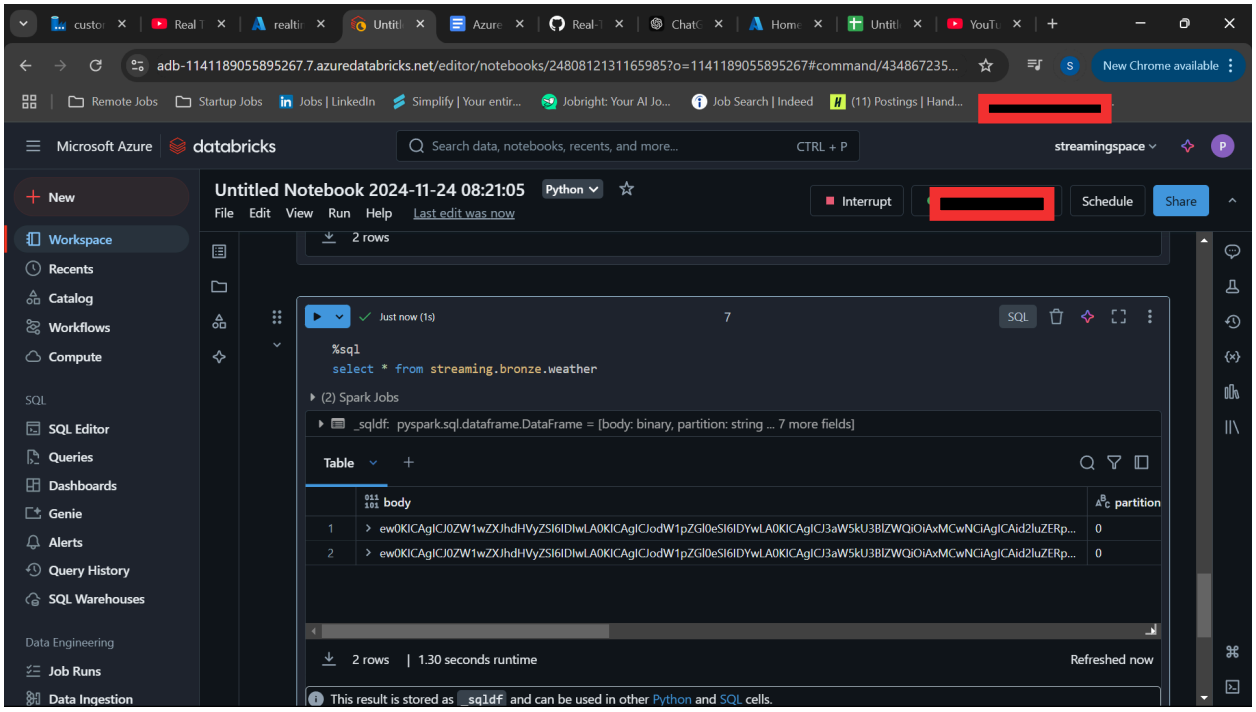
Now, even if I terminate the stream, the data still persists as I have written the stream to weather table in the bronze layer

I enabled the DBFS file browser to see the checkpointing information. I can see the checkpointing information as shown in the image below. This keeps track of the state and progress of the stream





I can query the weather table in the bronze layer as shown below



I created the silver layer by initiating the stream from the bronze layer and writing the stream to the checkpoint location. I used the below code to create the silver layer

```
# Defining the schema for the JSON object

json_schema = StructType([
    StructField("temperature", IntegerType()),
```

```
    StructField("humidity", IntegerType()),
    StructField("windSpeed", IntegerType()),
    StructField("windDirection", StringType()),
    StructField("precipitation", IntegerType()),
    StructField("conditions", StringType())
])
```

```
# Reading and Transforming: Load streaming data from the
'streaming.bronze.weather' Delta table, cast 'body' to string, parse JSON, and
select specific fields
df = spark.readStream\
    .format("delta")\
    .table("streaming.bronze.weather")\
    .withColumn("body", col("body").cast("string"))\
    .withColumn("body",from_json(col("body"), json_schema))\
    .select("body.temperature", "body.humidity", "body.windSpeed",
"body.windDirection", "body.precipitation", "body.conditions",
col("enqueuedTime").alias('timestamp'))

# Displaying stream: Visualize the transformed data in the DataFrame for
verification and analysis
df.display()

# Writing stream: Save the transformed data to the 'streaming.silver.weather'
Delta table in 'append' mode with checkpointing for data reliability
df.writeStream\
    .option("checkpointLocation", "/mnt/streaming/silver/weather")\
    .outputMode("append")\
    .format("delta")\
    .toTable("streaming.silver.weather")
```

Code Explanation:
Define JSON Schema:

- A schema (`json_schema`) is defined using `StructType` to describe the structure of the incoming JSON objects. It includes fields like `temperature`, `humidity`, `windSpeed`, etc., with appropriate data types.

Reading Streaming Data:

- A streaming DataFrame (`df`) is created by reading data from the Delta table `streaming.bronze.weather`.
- The `body` column (which contains JSON data) is cast to a string and parsed into a structured format using the `from_json` function, applying the `json_schema`.

- Specific fields (`temperature`, `humidity`, `windSpeed`, etc.) are extracted from the parsed JSON along with the `enqueuedTime` (renamed as `timestamp`).

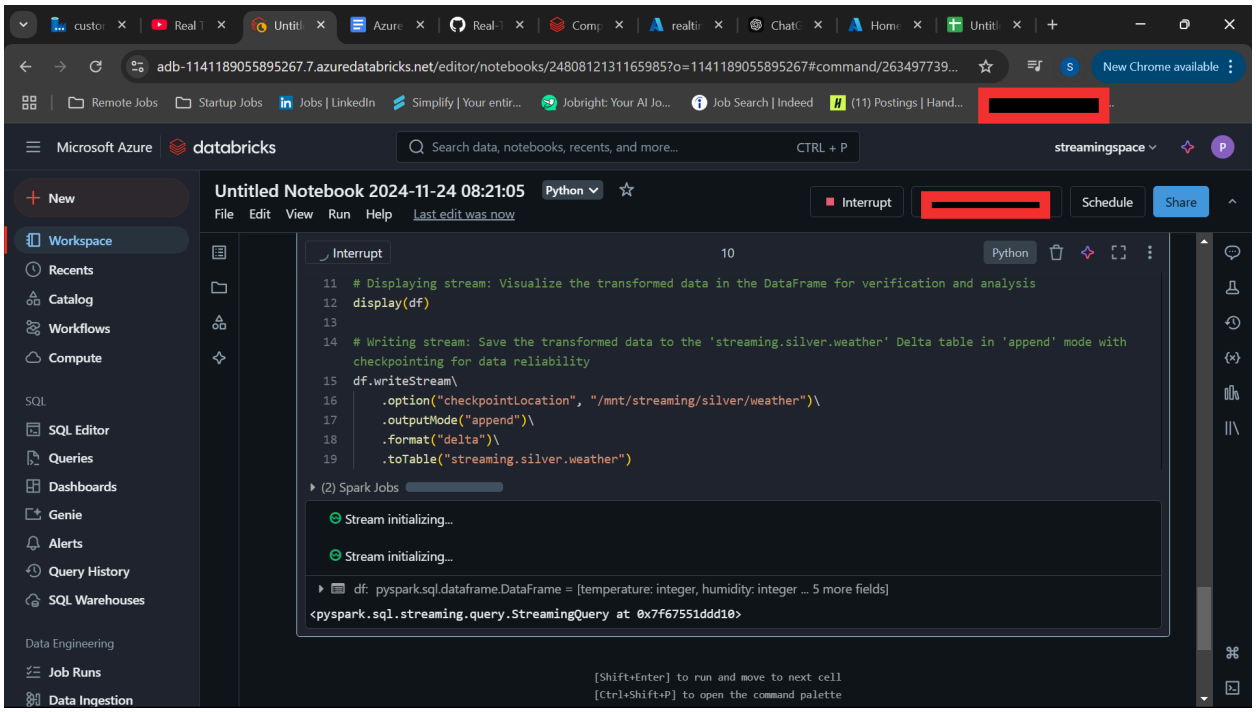
Displaying the Stream:

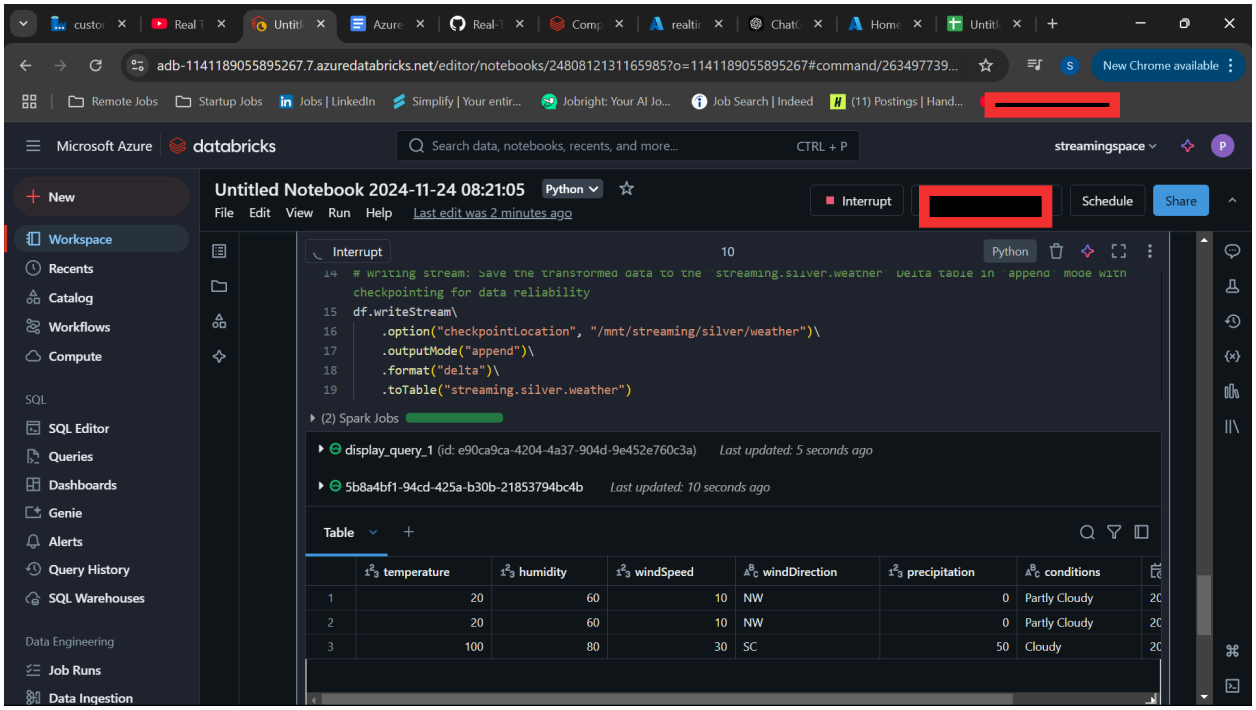
- The `df.display()` function is used to visualize the real-time streaming data for verification and analysis in an interactive notebook environment (e.g., Databricks).

Writing Transformed Data to Silver Table:

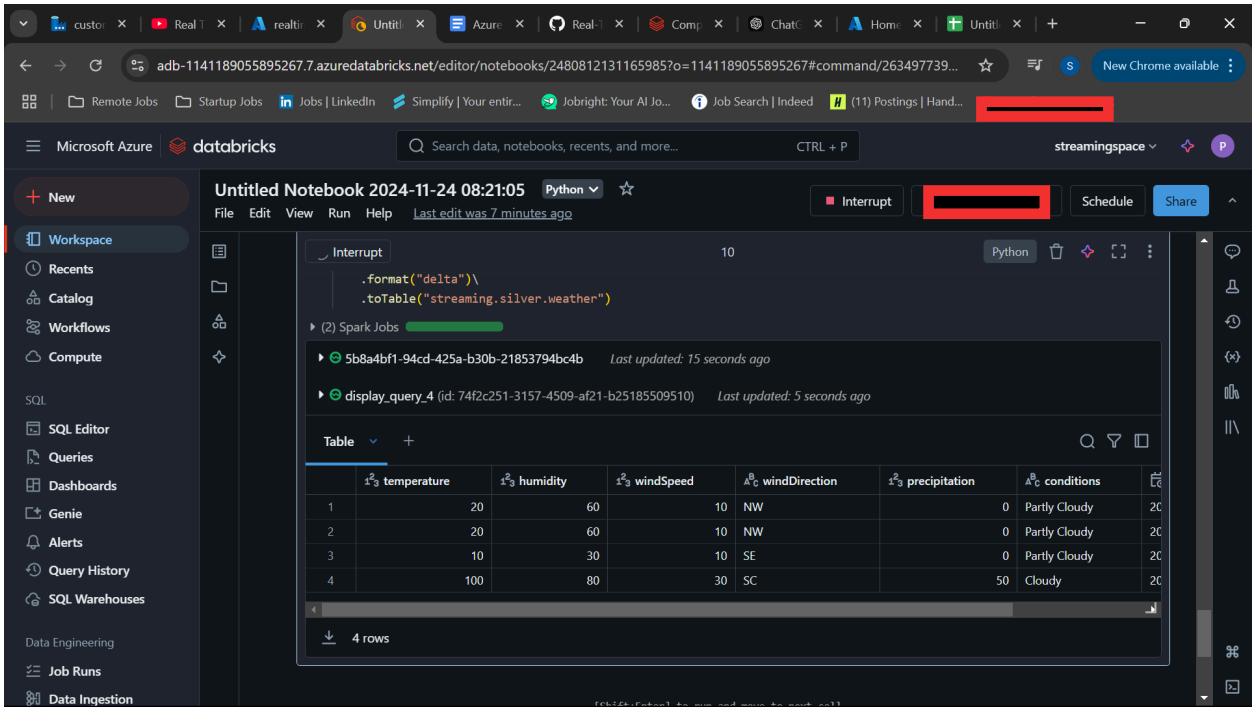
- The transformed data is written to the Delta table `streaming.silver.weather` in "append" mode, meaning new data is continuously added to the table.
- A checkpoint directory (`/mnt/streaming/silver/weather`) is specified to ensure data reliability and fault tolerance during the streaming process.

I can see that the stream is initialized and all the data is converted to tabular format from a nested JSON

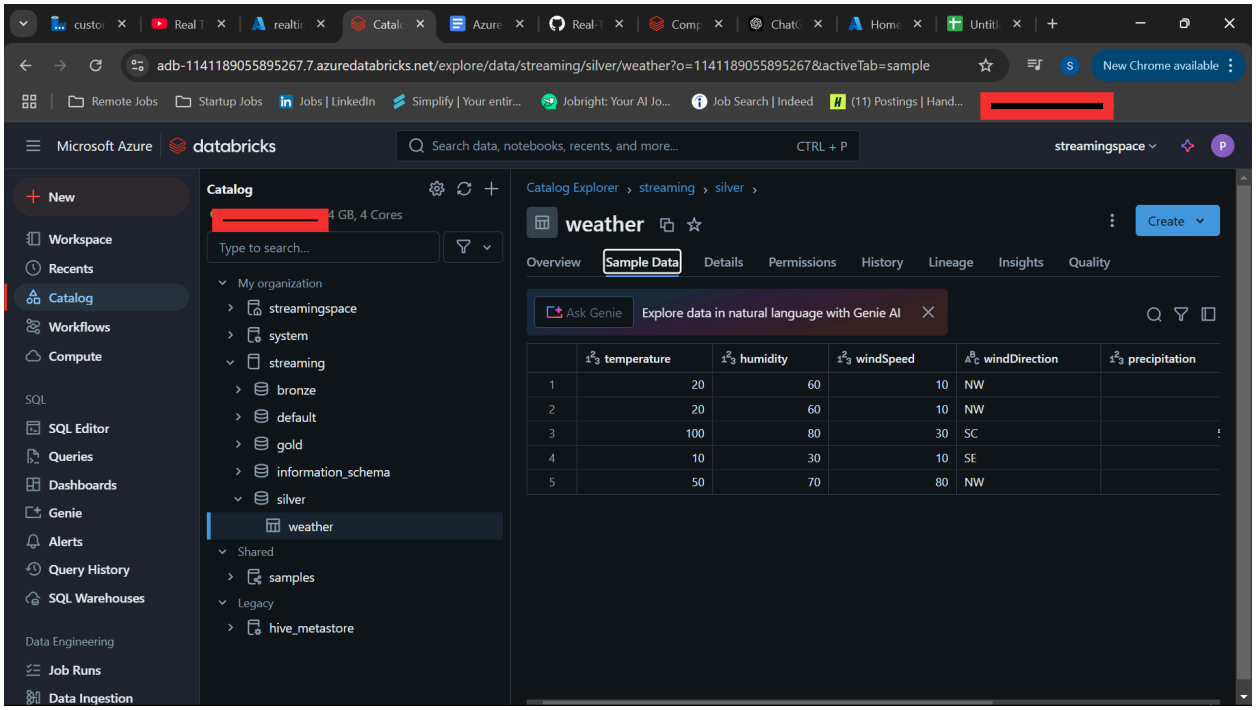




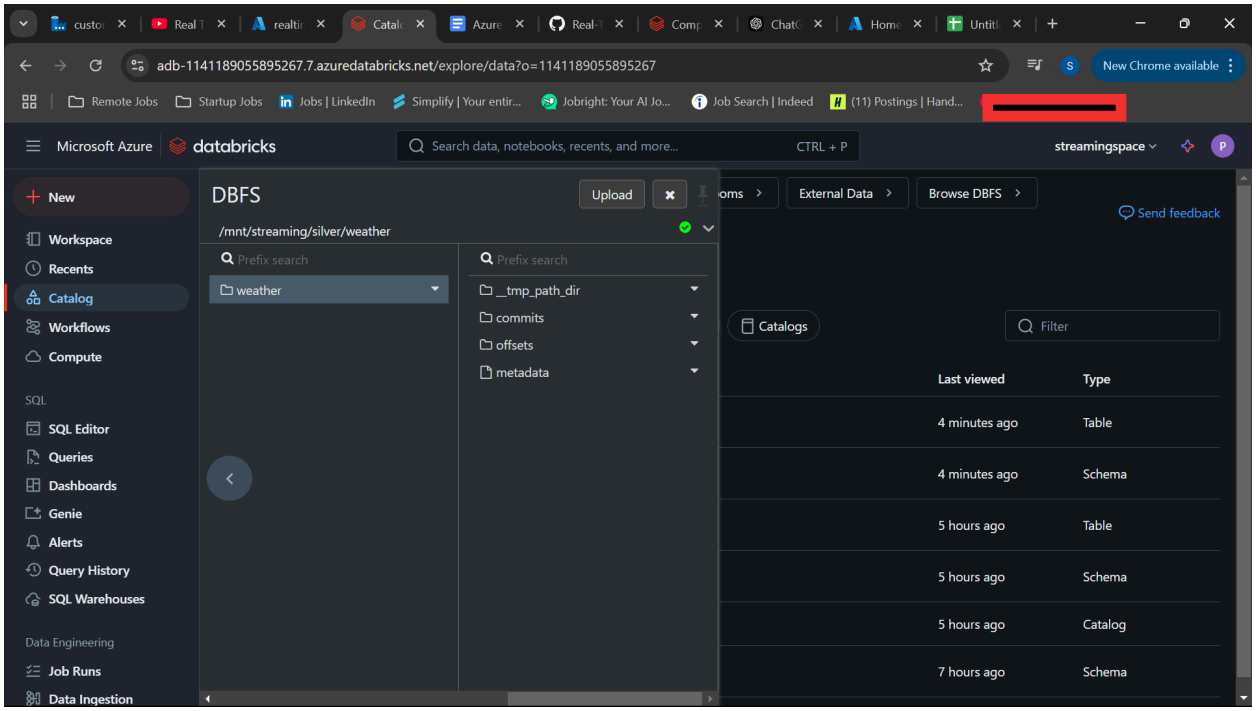
As the spark structured streaming processes the historical data and real time with exactly once semantics, I can see the historical records too in the output, If I generate a new event now, the output will be changed as below



I can also view the sample weather data in the silver layer as below



The data is persisted in the databricks environment. I can see the checkpointing information from DBFS browser as below



For the gold layer, I aggregated the data into 5 minute intervals using a timestamp column. I created the gold layer using the below code

```
# Aggregating Stream: Read from 'streaming.silver.weather', apply watermarking and windowing, and calculate average weather metrics
```

```
df = spark.readStream\
    .format("delta")\
    .table("streaming.silver.weather")\
    .withWatermark("timestamp", "5 minutes") \
    .groupBy(window("timestamp", "5 minutes")) \
    .agg(avg("temperature").alias('temperature'),
avg("humidity").alias('humidity'), avg("windSpeed").alias('windSpeed'),
avg("precipitation").alias('precipitation'))\
      .select('window.start', 'window.end', 'temperature', 'humidity',
'windSpeed', 'precipitation')

# Displaying Aggregated Stream: Visualize aggregated data for insights into
weather trends
df.display()

# Writing Aggregated Stream: Store the aggregated data in
'streaming.gold.weather_aggregated' with checkpointing for data integrity
df.writeStream\
    .option("checkpointLocation", "/mnt/streaming/weather_summary")\
    .outputMode("append")\
    .format("delta")\
    .toTable("streaming.gold.weather_summary")
```

Code Explanation:

1. Reading the Silver Table Stream:

- A streaming DataFrame (`df`) is created by reading data from the Delta table `streaming.silver.weather`.
- **Watermarking** is applied on the `timestamp` column with a 5-minute threshold to handle late-arriving data, ensuring efficient memory usage and consistency.

2. Windowing and Aggregation:

- **Windowing**: The `window` function groups data into 5-minute intervals based on the `timestamp` column.
- **Aggregation**: The `avg` function calculates the average of weather metrics (`temperature`, `humidity`, `windSpeed`, `precipitation`) for each window.
- The resulting DataFrame includes:
    - `window.start` and `window.end` to indicate the time range for each aggregation.
    - Average values of the specified weather metrics.

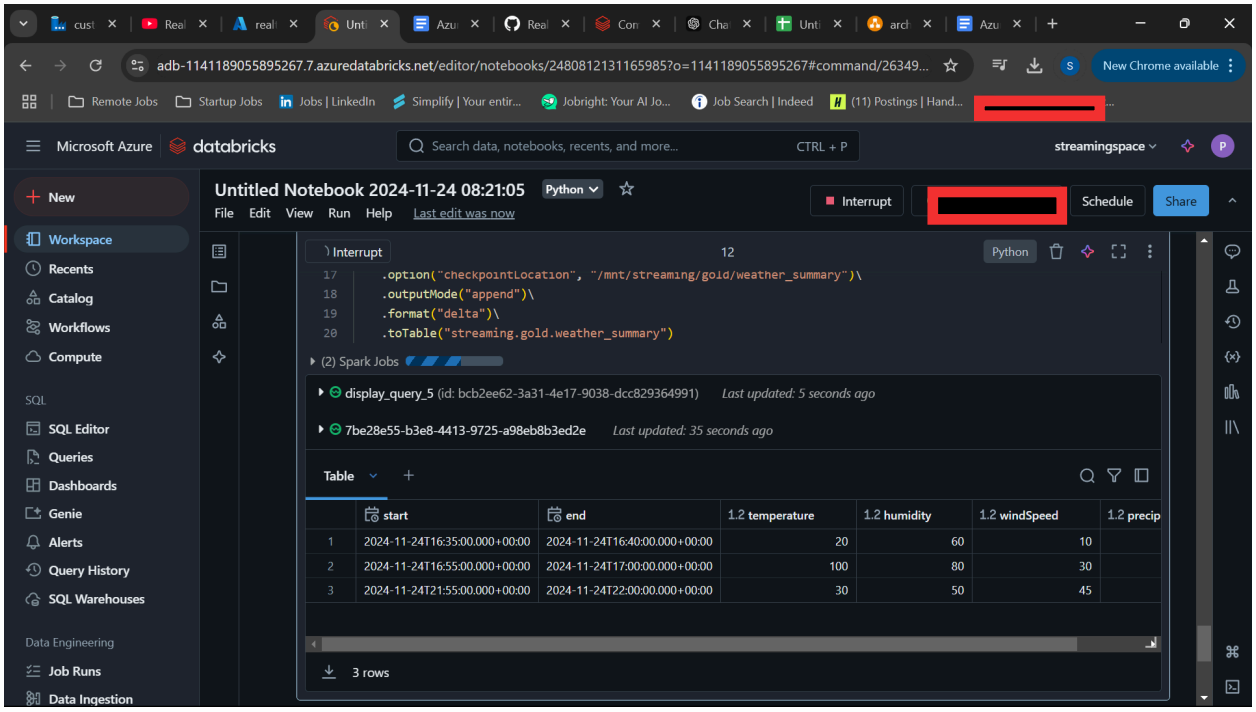
3. Displaying Aggregated Data:

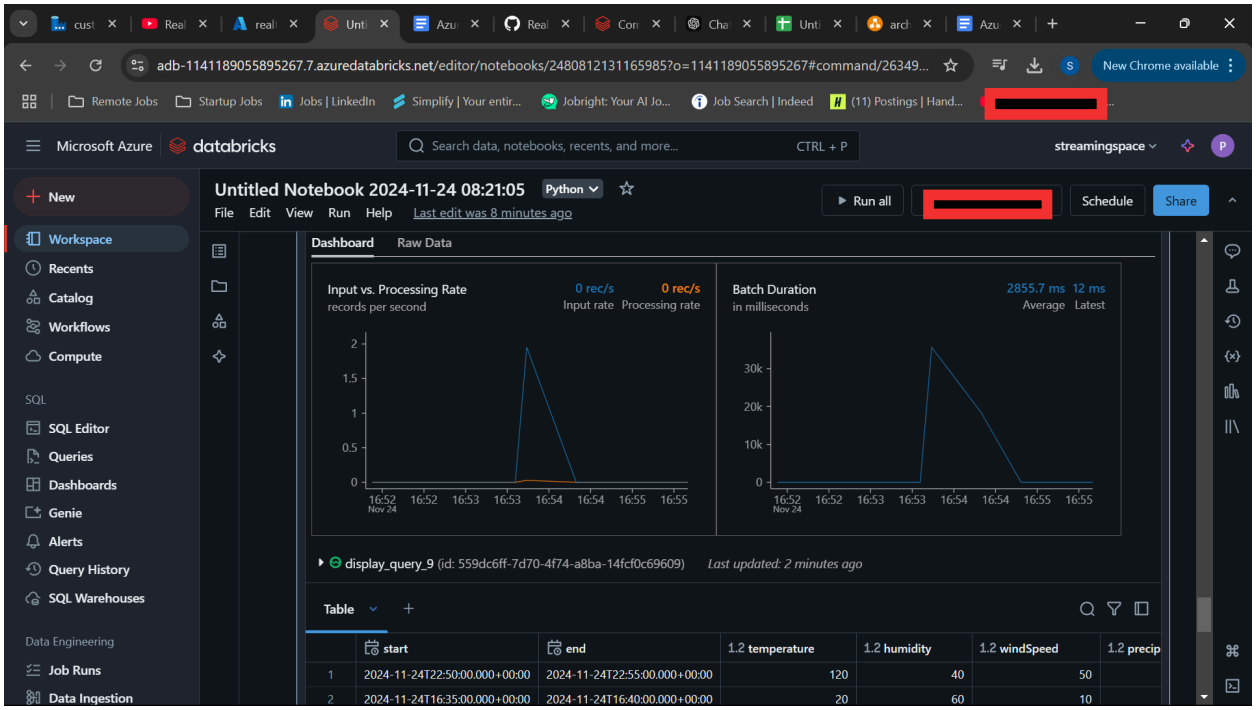
- `df.display()` visualizes the aggregated streaming data in an interactive notebook environment, enabling real-time monitoring of weather trends.

4. Writing Aggregated Data to Gold Table:

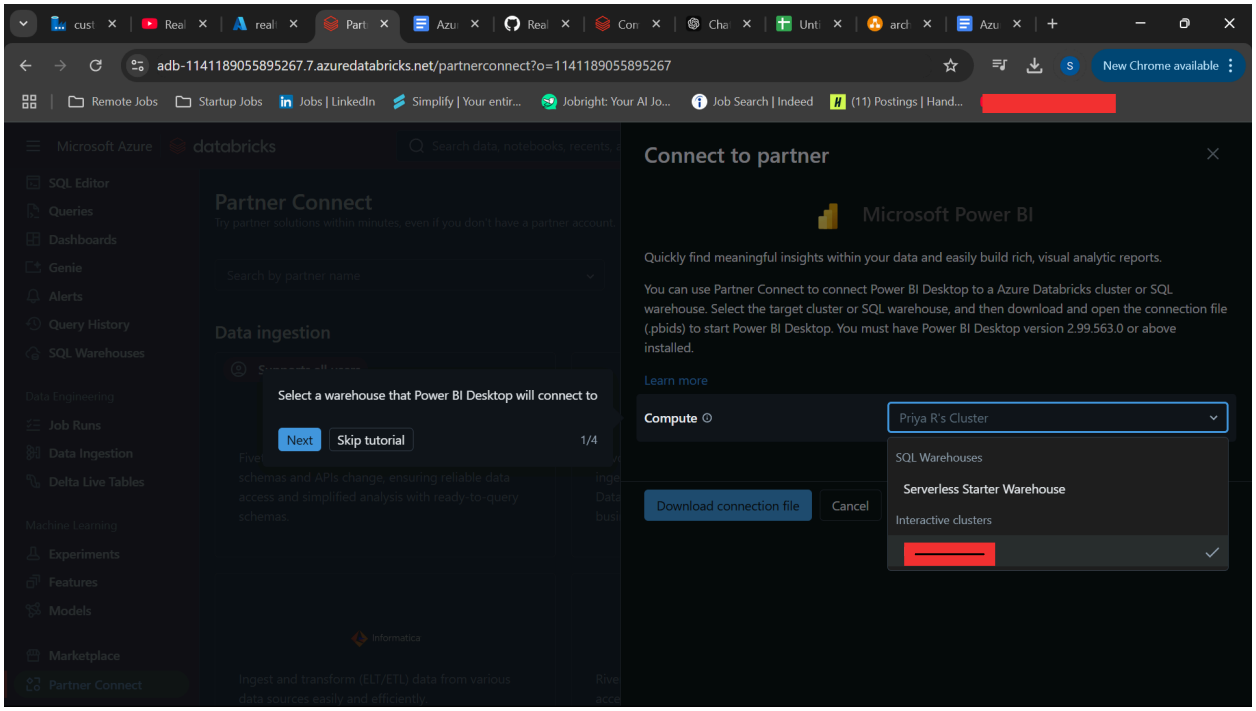
- The aggregated data is written to the Delta table `streaming.gold.weather_summary` in "append" mode, continuously appending new summaries as the stream progresses.
- A checkpoint directory (`/mnt/streaming/weather_summary`) ensures data integrity and fault tolerance during the streaming process.

I can see the stream is initialized for gold layer as below

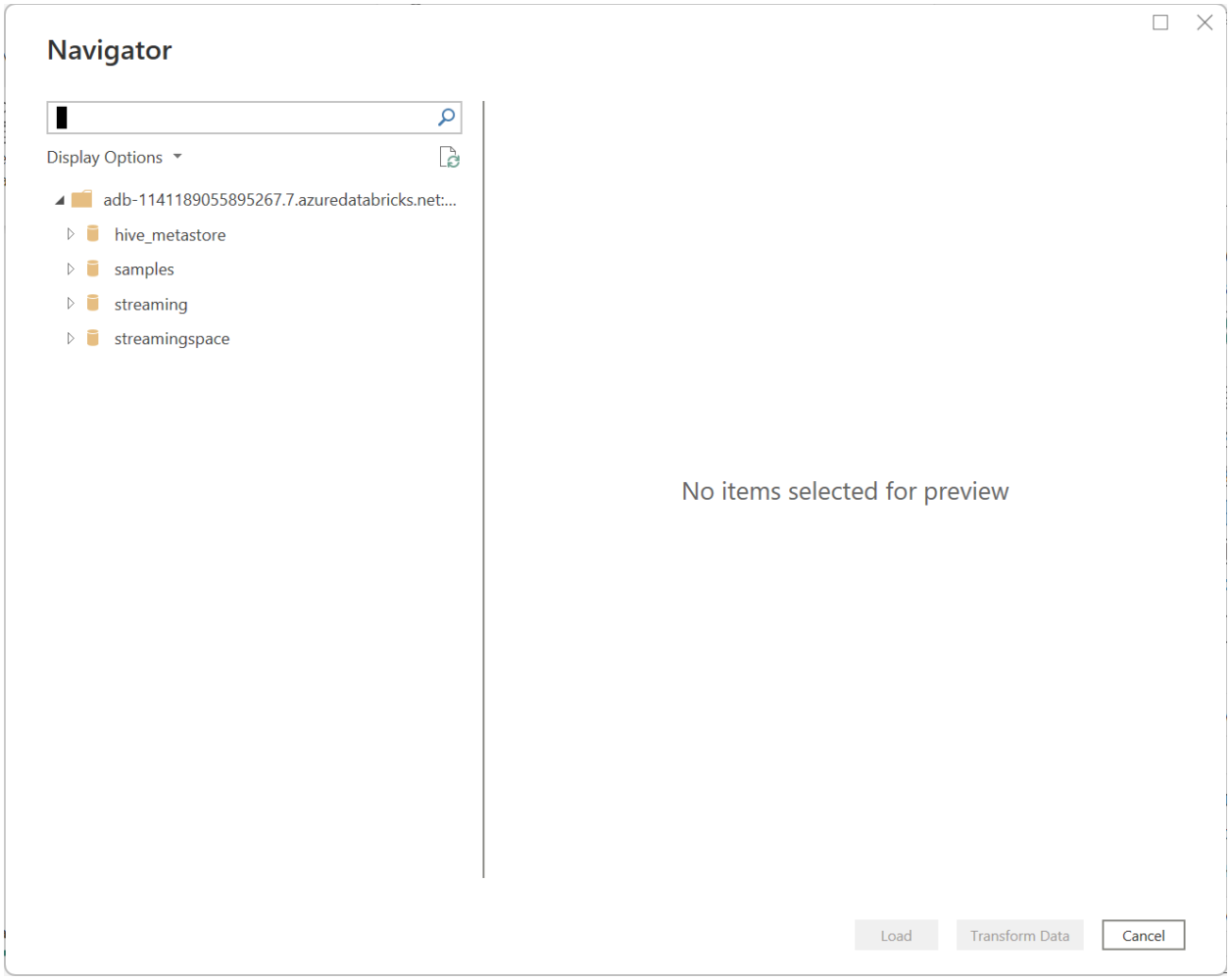




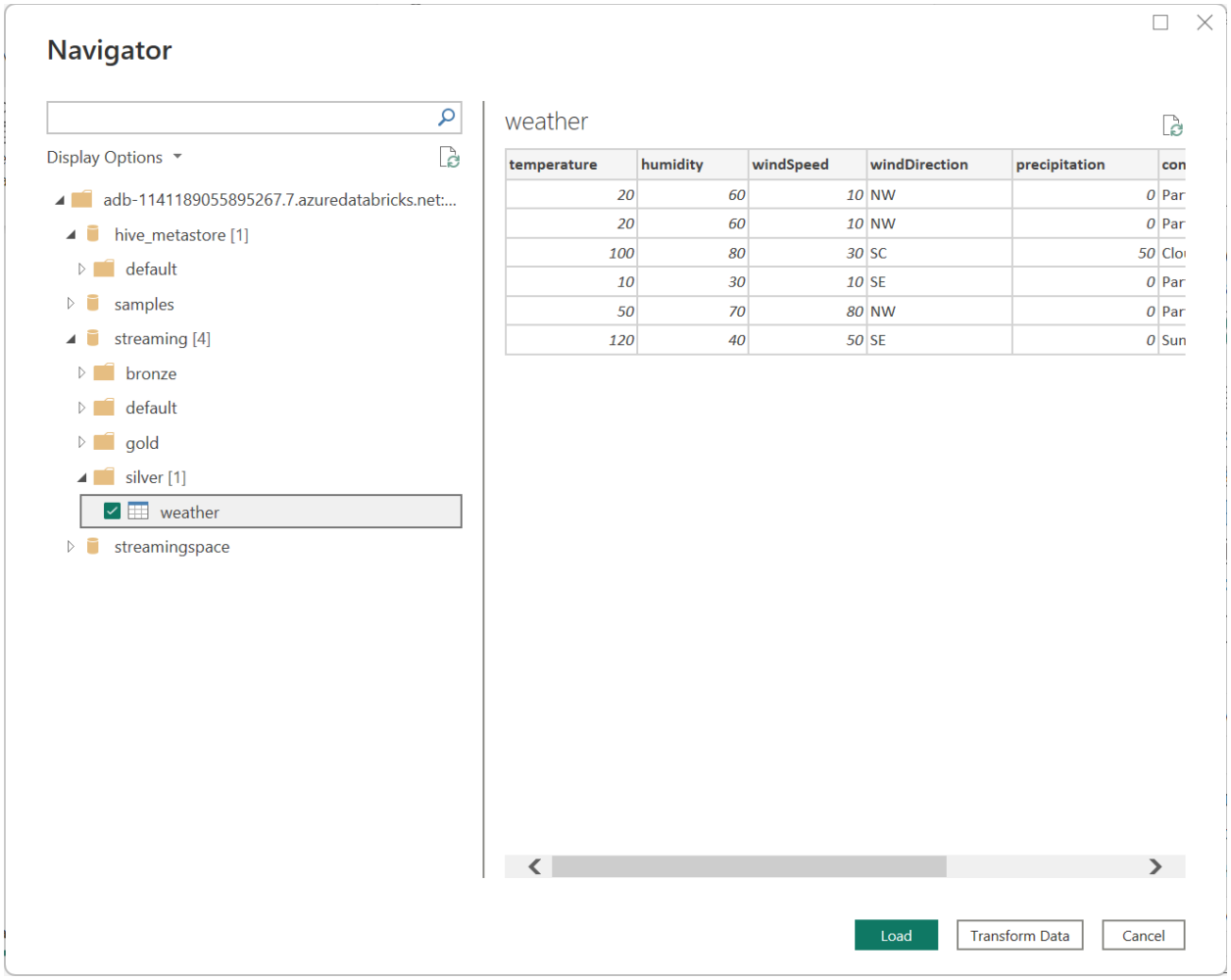
Now, i connected the gold layer to Power BI inorder to monitor the streaming on a dashboard in real time. For this, i used the “Partner Connect” option as below



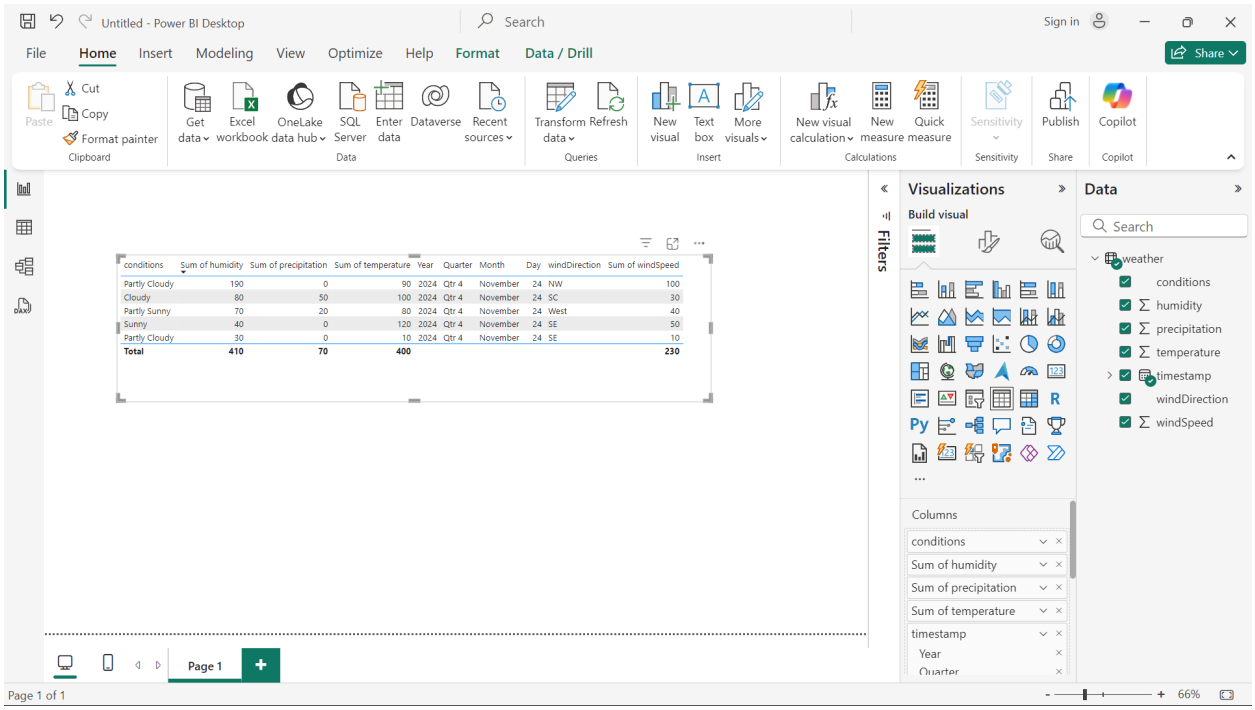
I signed in through Azure Active directory and I can see the data in Power BI as below



Hare, I selected the silver layer to observe the changes in real time as the gold layer has a watermark for 5 minutes and it will take longer to observe the changes.



Here, I was using the “DirectQuery” mode in power BI, so it took time to load the data as the data is not in the power BI’s internal memory. I switched to import mode later to see the changes faster. Also, to see the streaming data change in the power BI dashboard, I refreshed the query and I can see the dashboard below.


Untitled - Power BI Desktop
Search
Sign in
File
Home
Insert
Modeling
View
Optimize
Help
Format
Data / Drill
Share
Cut
Copy
Format painter
Paste
Clipboard
Get data
Excel workbook
OneLake data hub
SQL Server
Enter data
Dataverse
Recent sources
Data
Transform data
Refresh
Queries
New visual
Text box
More visuals
Insert
New visual calculation
New measure
Quick measure
Calculations
Sensitivity
Publish
Share
Copilot
conditions Sum of humidity Sum of precipitation Sum of temperature Year Quarter Month Day windDirection Sum of windSpeed
Partly Cloudy 190 0 90 2024 Qtr 4 November 24 NW 100
Cloudy 80 50 100 2024 Qtr 4 November 24 SC 30
Partly Sunny 70 20 80 2024 Qtr 4 November 24 West 40
Sunny 40 0 120 2024 Qtr 4 November 24 SE 50
Partly Cloudy 30 0 10 2024 Qtr 4 November 24 SE 10
Total 410 70 400 230
Filters
Visualizations
Build visual
Columns
conditions
Sum of humidity
Sum of precipitation
Sum of temperature
timestamp
Year
Quarter
Data
Search
weather
conditions
humidity
precipitation
temperature
timestamp
windDirection
windSpeed
Page 1
Page 1 of 1
66%