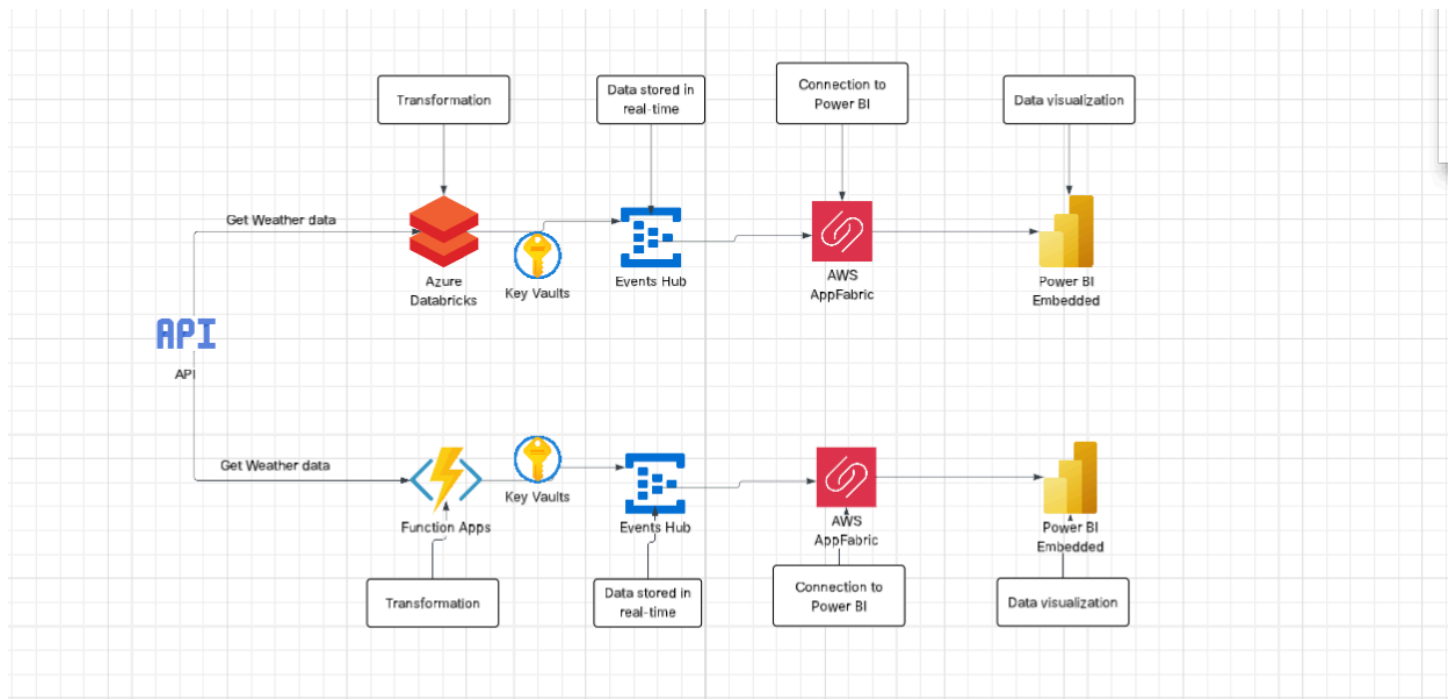


# Real-time Weather Streaming Project on Azure by Diljot Singh

Git: [Azure-Projects/Real-time Weather-API-Report at main · diljotrandhawaa/Azure-Projects](https://github.com/diljotrandhawaa/Azure-Projects/Real-time%20Weather-API-Report)

Workflow:



## The goal of the Project:

Fetch the Weather Data for a city (Chennai, India) every 30 seconds, Perform transformations and select appropriate data to create a visual report in Power BI.

## Tech used:

API (Weather API)  
Azure Databricks and Databricks Clusters  
Azure Key Vault  
Azure Event hubs  
Microsoft Fabric  
Event house and Event Stream  
Power BI

The code for Jupyter notebook and the Function app can be found in the git repository: [Azure-Projects/Real-time Weather-API-Report at main · diljotrandhawaa/Azure-Projects](https://github.com/diljotrandhawaa/Azure-Projects/Real-time%20Weather-API-Report)

The project can be implemented in two ways:

1. Using Databricks's Jupyter notebook to fetch data from the API, or

## 2. Using function app to fetch data from the API

Both ways are demonstrated and implemented in Azure

Before starting with implementation:

1. Create an Azure Databricks workspace and inside Databricks, create a Cluster.
2. Then create a Function App (for alternate implementation).
3. Next, create a Event Hub Namespace like below:
4. Create a Event Hub inside the Namespace.
5. Create a Microsoft Fabric capacity like below:
6. Create a Azure Key Vault and add Databricks and the Function App as Secrets Users.

Compute > Simple form: OFF

### streaming-cluster

**Configuration** Notebooks (0) Libraries Event log Spark UI Driver logs Metrics Apps Spark cor

Policy ⓘ  
Personal Compute

Access mode ⓘ Single user or group access ⓘ  
Dedicated (formerly: Single user) Diljot Singh

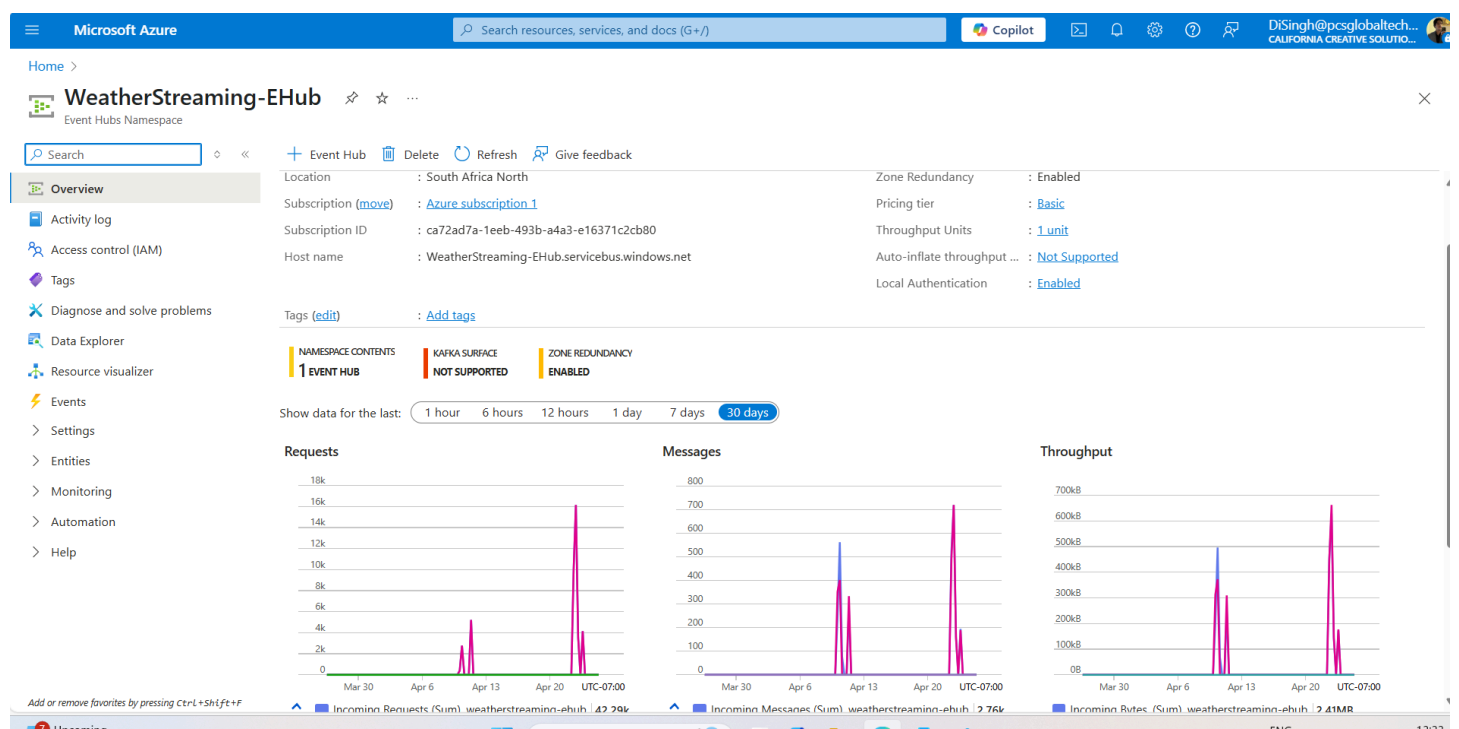
**Performance**  
Databricks Runtime Version  
16.3 ML (includes Apache Spark 3.5.2, Scala 2.12)


☐ Use Photon Acceleration ⓘ  
Photon boosts Apache Spark workloads. Not all ML workloads will see an improvement. [Learn more](#)

Node type ⓘ  
Standard\_DS3\_v2 14 GB Memory, 4 Cores

☒ Terminate after 60 minutes of inactivity ⓘ




**Tags** ⓘ  
No custom tags  
> Automatically added tags  
▶ Advanced options





weatherstreamingfabricwest

Fabric Capacity

Pause

Refresh

Move

Delete

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Essentials

Resource group (move)

Weather-Streaming-Project

Status

: Active

Location

: West US

Subscription (move)

: Azure subscription 1


Resource name

: weatherstreaming



SKU

: F2

Home > WeatherStreaming-KVDil



WeatherStreaming-KVDil | Access control (IAM)

Add

Download role assignments

Edit columns

Refresh

Delete

Feedback

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Access policies

Resource visualizer

Events

Objects

Settings

Monitoring

Automation

Help

Check access

Role assignments

Roles

Deny assignments

Classic administrators

Number of role assignments for this subscription

8 / 4000

Type: All

Role: All











Scope: All scopes

Group by: Role

All (5)

Job function roles (3)

Privileged administrator roles (2)

<input type="checkbox"/>	Name ↑↓	Type ↑↓	Role ↑↓	Scope ↑↓	Condition ↑↓
▼	Owner (2)				
<input type="checkbox"/>	 Diljot Singh DiSingh@pcsglobaltech.com	User	Owner	 Subscription (Inherited)	None
<input type="checkbox"/>	 Diljot Singh DiSingh@pcsglobaltech.com	User	Owner	 Subscription (Inherited)	None
▼	Key Vault Secrets Officer (1)				
<input type="checkbox"/>	 Diljot Singh DiSingh@pcsglobaltech.com	User	Key Vault Secrets Officer	 This resource	None
▼	Key Vault Secrets User (2)				
<input type="checkbox"/>	 AzureDataBricks	Service principal	Key Vault Secrets User	 This resource	None
<input type="checkbox"/>	 WeatherStreaming-FApp	Managed identity	Key Vault Secrets User	 This resource	None

Showing 1 - 5 of 5 results.

## Steps:

### Step 1:

Below is data fetch using Databricks:



a. A notebook is created called “weathers-streaming-notebook”

Microsoft Azure

databricks

CTRL + P

WeatherStreaming\_DBW

New

Workspace

Recents


Catalog

Workflows

weather-streaming-notebook

Python

Tabs: OFF



File

Edit

View

Run

Help

Last edit was 15 days ago

Run all

Terminated

Schedule

Share

Workspace

←

disingh@pcsglobaltech.com

weather-streaming-notebook

Trial expired. Upgrade to Premium in Azure Portal

b. The code to fetch weather API data

```

from azure.eventhub import EventHubProducerClient, EventData
import json
import requests
from datetime import datetime, timedelta

# Event Hub configuration
EVENT_HUB_NAME = "weatherstreamingnamehub"
eventhub_connection_string = dbutils.secrets.get(scope="key-vault-scope", key="eventhub-connectionstring")
weatherapikey = dbutils.secrets.get(scope="key-vault-scope", key="weatherapikey")

# Initialize the Event Hub producer
producer = EventHubProducerClient.from_connection_string(conn_str=eventhub_connection_string, eventhub_name=EVENT_HUB_NAME)

# Function to send events to Event Hub
def send_event(event):
    event_data_batch = producer.create_batch()
    event_data_batch.add(EventData(json.dumps(event)))
    producer.send_batch(event_data_batch)

# Function to handle the API response
def handle_response(response):
    if response.status_code == 200:
        return response.json()
    else:
        return f"Error: {response.status_code}, {response.text}"

```

c. Once the data is fetched, it is flattened to get the aspects we need from the data below is just a snippet of the code:

```

# Flatten and merge the data
def flatten_data(current_weather, forecast_weather, alerts):
    location_data = current_weather.get("location", {})
    current = current_weather.get("current", {})
    condition = current.get("condition", {})
    air_quality = current.get("air_quality", {})
    forecast = forecast_weather.get("forecast", {}).get("forecast", {})
    alert_list = alerts.get("alerts", {}).get("alert", [])

    flattened_data = {
        'name': location_data.get('name'),
        'region': location_data.get('region'),
        'country': location_data.get('country'),
        'lat': location_data.get('lat'),
        'lon': location_data.get('lon'),
        'localtime': location_data.get('localtime'),
        'temp_c': current.get('temp_c'),
        'is_day': current.get('is_day'),
        'condition_text': condition.get('text'),
        'condition_icon': condition.get('icon'),
        'wind_kph': current.get('wind_kph'),
        'wind_degree': current.get('wind_degree'),
        'wind_dir': current.get('wind_dir'),
        'pressure_in': current.get('pressure_in'),
        'precip_in': current.get('precip_in'),
        'humidity': current.get('humidity'),
        'cloud': current.get('cloud'),
        'feelslike_c': current.get('feelslike_c'),
        'uv': current.get('uv'),
        'air_quality': {
            'co': air_quality.get('co'),
            'no2': air_quality.get('no2'),
            'o3': air_quality.get('o3'),
            'so2': air_quality.get('so2'),
            'pm2_5': air_quality.get('pm2_5'),
            'pm10': air_quality.get('pm10'),
            'us-epa-index': air_quality.get('us-epa-index'),
            'gb-defra-index': air_quality.get('gb-defra-index')
        }
    },

```

d. Once, we have the data we need in a flattened form, a function is created to make sure that the **API data is fetched every 30 seconds** from the source and gets written to the destination, the **destination is Event Hub** (using send\_event function described above)

```

# Main program
def process_batch(batch_df, batch_id):
    global last_sent_time
    try:
        # Get current time
        current_time = datetime.now()

        # Check if 30 seconds have passed since last event was sent
        if (current_time - last_sent_time).total_seconds() >= 30:
            # Fetch weather data
            weather_data = fetch_weather_data()

            # Send the weather data (current weather part)
            send_event(weather_data)

            # Update last sent time
            last_sent_time = current_time
            print(f'Event Sent at {last_sent_time}')
    except:
        pass

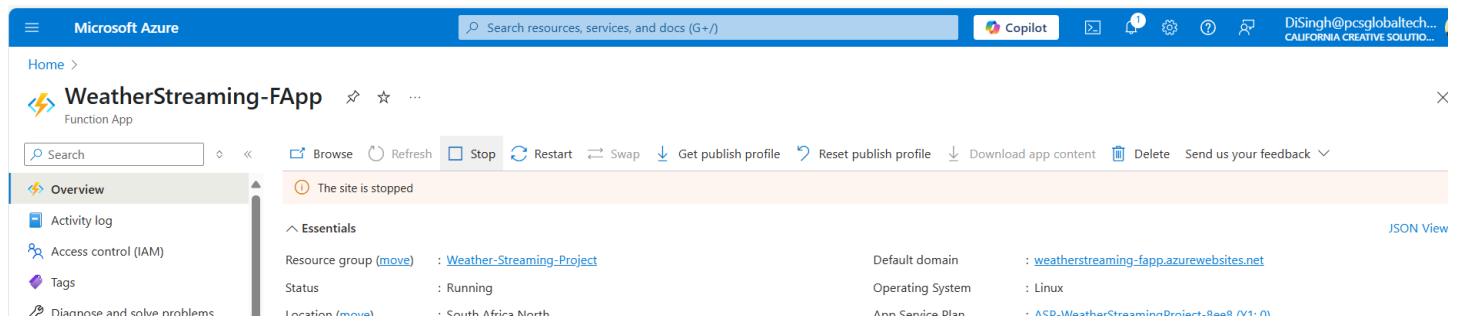
```

This completes the DataBricks section

The same is done using Function App, either one can be used.

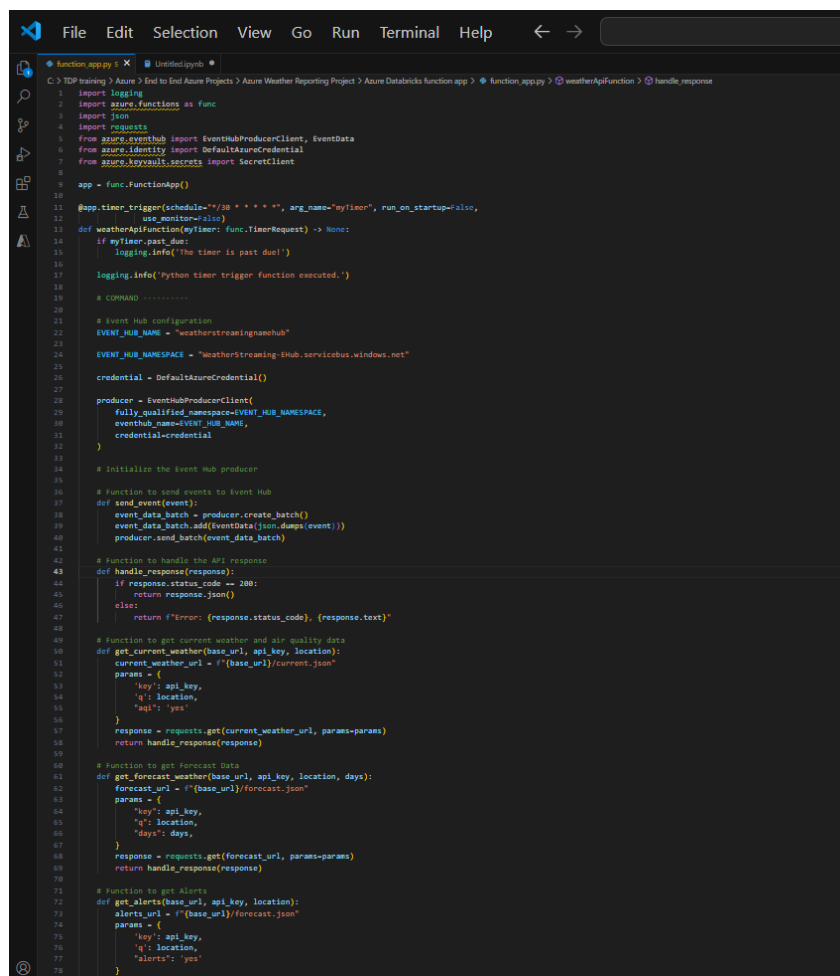
Below is data fetch using Function App:

a. A function app is created first in Azure



b. Then a Python Source file is written using Visual Studio Code, it has the same code as our Jupyter notebook.

Below is a snippet from the code:



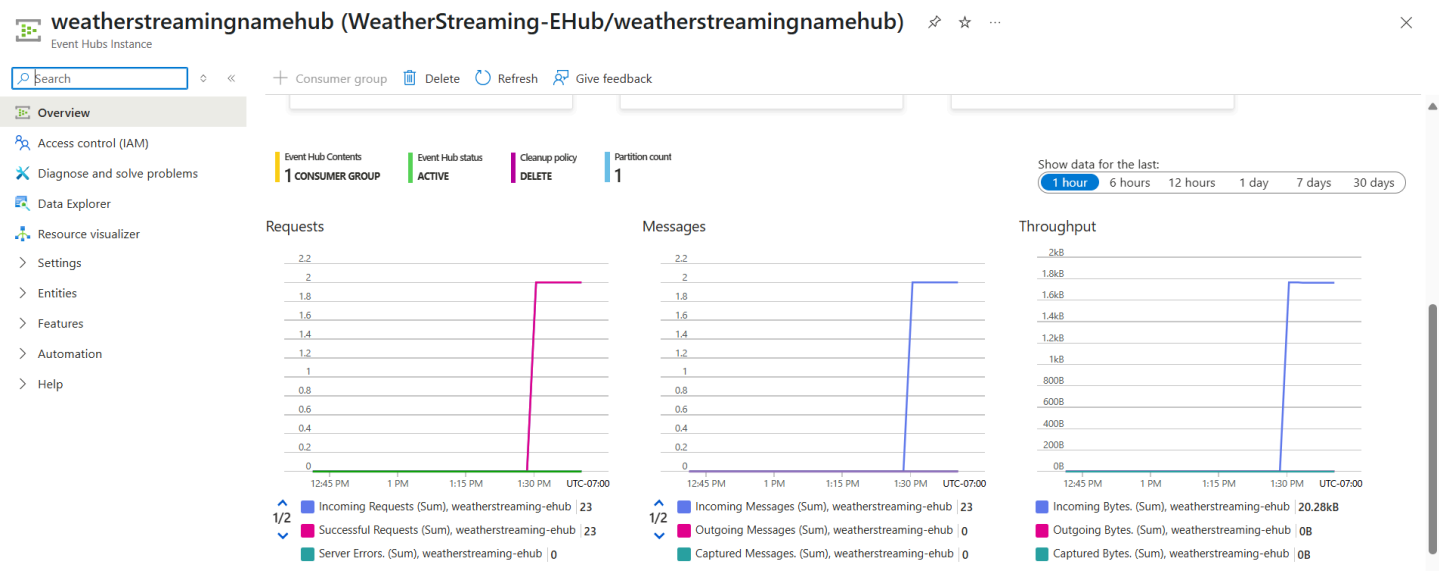
c. This Source code file is connected to the function app using Azure credentials. and The

code sends data to our Event Hub.

## This sums up the Step 1.

### Step 2:

Once the data is received in the Event hub



The data can be viewed in Data Explorer

The screenshot shows the Azure Event Hubs console for the instance 'weatherstreamingnamehub'. The Data Explorer page displays a table of events with columns: Sequence Number, Offset, Partition ID, Enqueued Time, Content Type, Message ID, and Event Body. The table shows a list of events starting from Sequence Number 2763. The Event Body column shows JSON data for each event.

Sequence Number	Offset	Partition ID	Enqueued Time	Content Type	Message ID	Event Body
2763	4296381216	0	Fri, 25 Apr, 25, 01:28:41 pm GMT-7			{"name": "Chennai", "region": "Tamil Nadu", "country": "India", "lat": ...
2764	4296382144	0	Fri, 25 Apr, 25, 01:29:02 pm GMT-7			{"name": "Chennai", "region": "Tamil Nadu", "country": "India", "lat": ...
2765	4296383072	0	Fri, 25 Apr, 25, 01:29:32 pm GMT-7			{"name": "Chennai", "region": "Tamil Nadu", "country": "India", "lat": ...
2766	4296384000	0	Fri, 25 Apr, 25, 01:30:02 pm GMT-7			{"name": "Chennai", "region": "Tamil Nadu", "country": "India", "lat": ...
2767	4296384928	0	Fri, 25 Apr, 25, 01:30:32 pm GMT-7			{"name": "Chennai", "region": "Tamil Nadu", "country": "India", "lat": ...
2768	4296385856	0	Fri, 25 Apr, 25, 01:31:02 pm GMT-7			{"name": "Chennai", "region": "Tamil Nadu", "country": "India", "lat": ...
2769	4296386784	0	Fri, 25 Apr, 25, 01:31:32 pm GMT-7			{"name": "Chennai", "region": "Tamil Nadu", "country": "India", "lat": ...
2770	4296387712	0	Fri, 25 Apr, 25, 01:32:02 pm GMT-7			{"name": "Chennai", "region": "Tamil Nadu", "country": "India", "lat": ...
2771	4296388640	0	Fri, 25 Apr, 25, 01:32:32 pm GMT-7			{"name": "Chennai", "region": "Tamil Nadu", "country": "India", "lat": ...
2772	4296389568	0	Fri, 25 Apr, 25, 01:33:02 pm GMT-7			{"name": "Chennai", "region": "Tamil Nadu", "country": "India", "lat": ...
2773	4296390496	0	Fri, 25 Apr, 25, 01:33:32 pm GMT-7			{"name": "Chennai", "region": "Tamil Nadu", "country": "India", "lat": ...

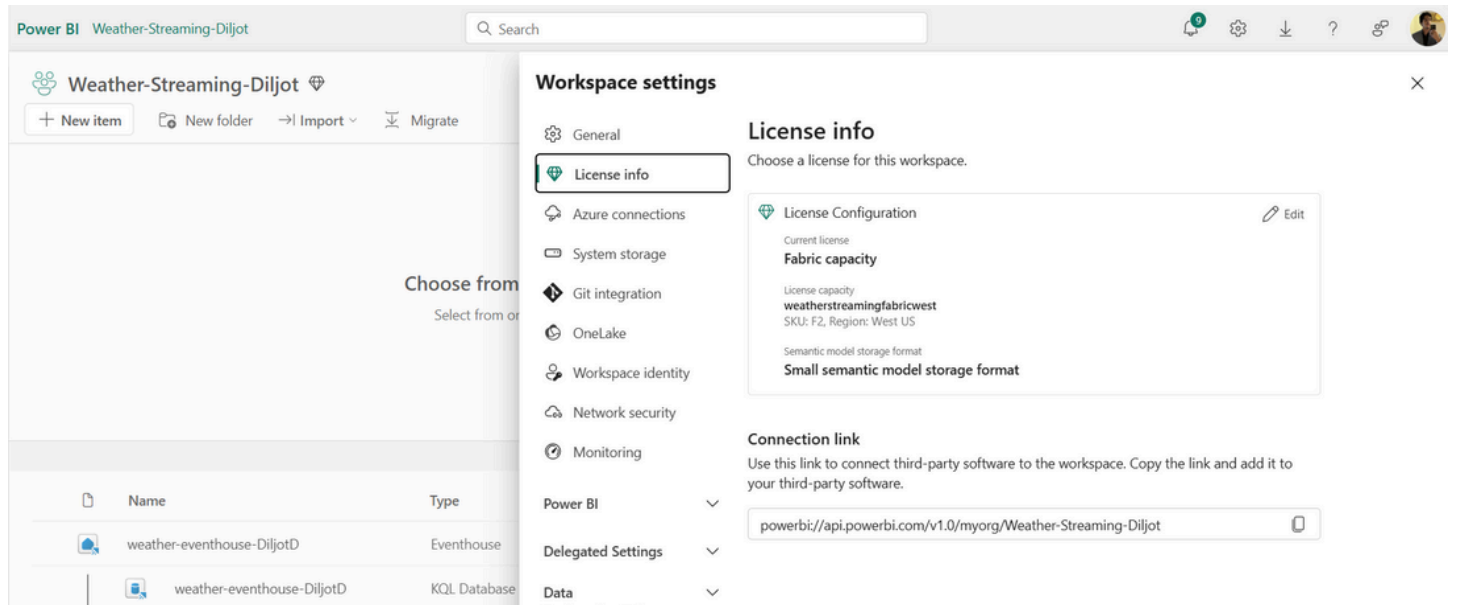
We are fetching data live for Chennai, India every 30 seconds.

### Step 3:

Now we have the data in the flattened form, we will export this data to Power BI using Fabric.

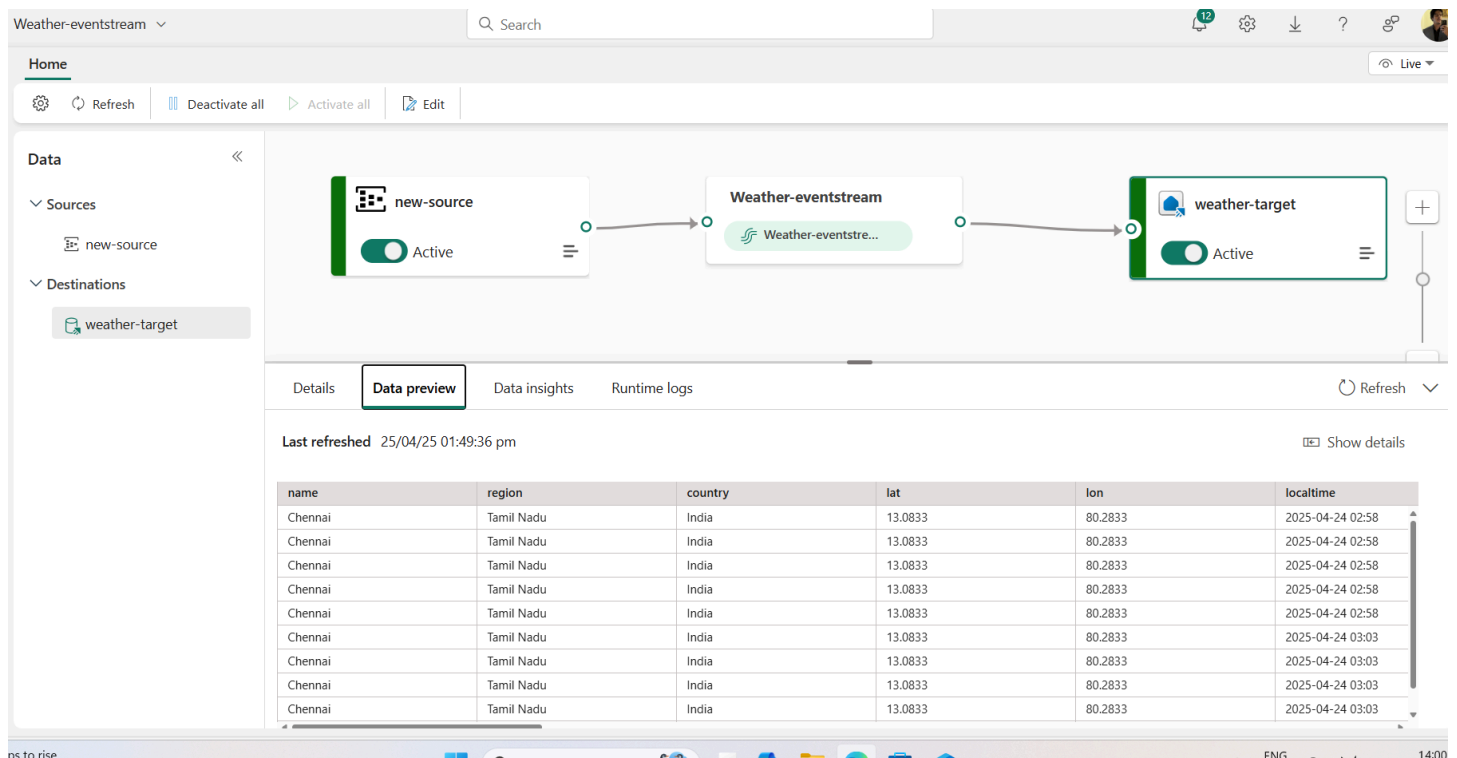
Go to Power BI and connect the Fabric capacity in Power BI

I have created a separate workspace and by editing the License Info, I have connected it to my Fabric capacity.



#### Step 4:

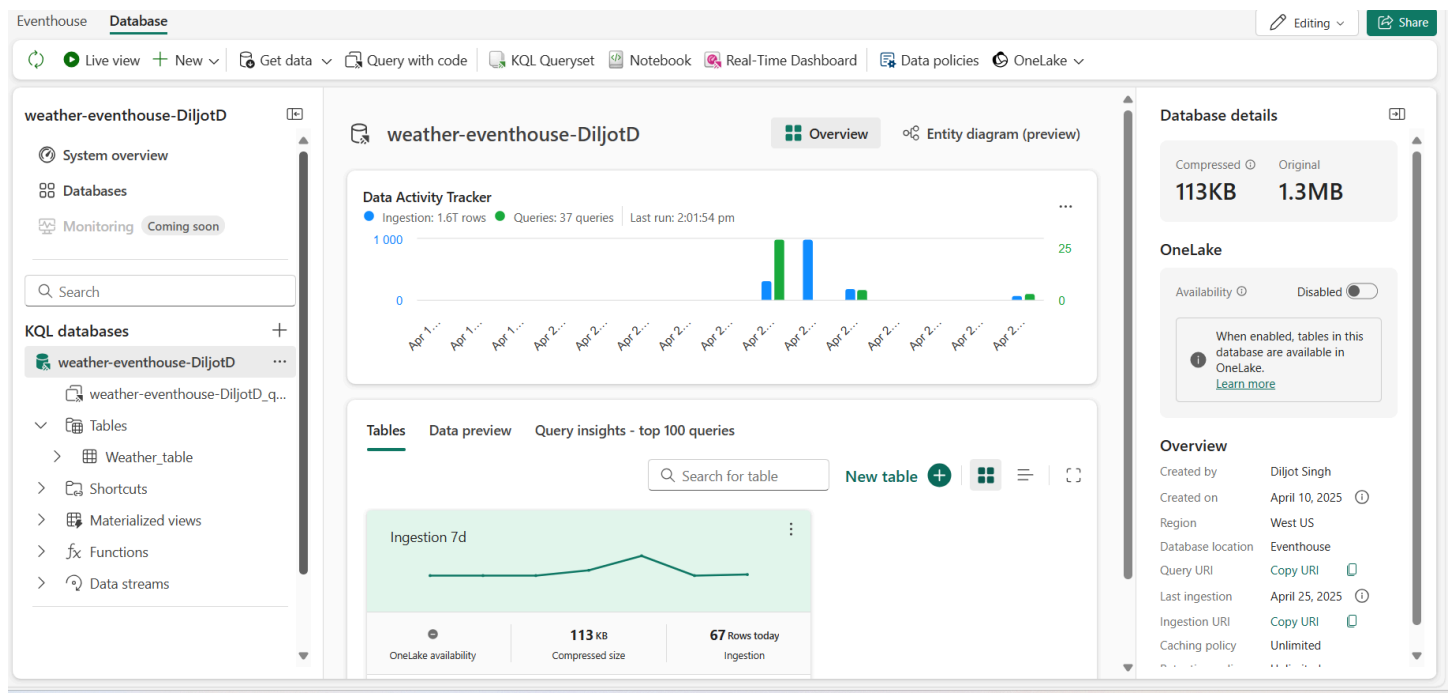
After connecting the fabric capacity to Power BI, we created an Eventhouse and an Eventstream to fetch data from Event Hub. In the Eventstream, the source is set as Event Hub and the target is Eventhouse



#### Step 5:

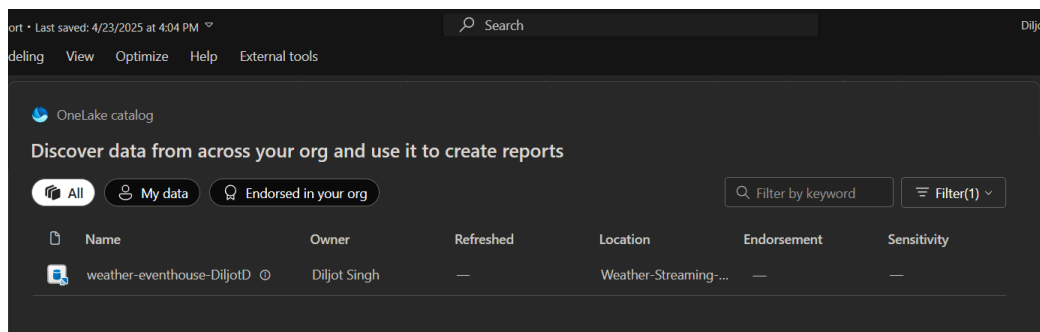
Next, a KQL database is created in Eventhouse to display and query data coming from EventHub





## Step 6:

Now, we will create a report using the KQL database in Power BI. Under “Get data”, we will select KQL database and import the table we created in KQL database.



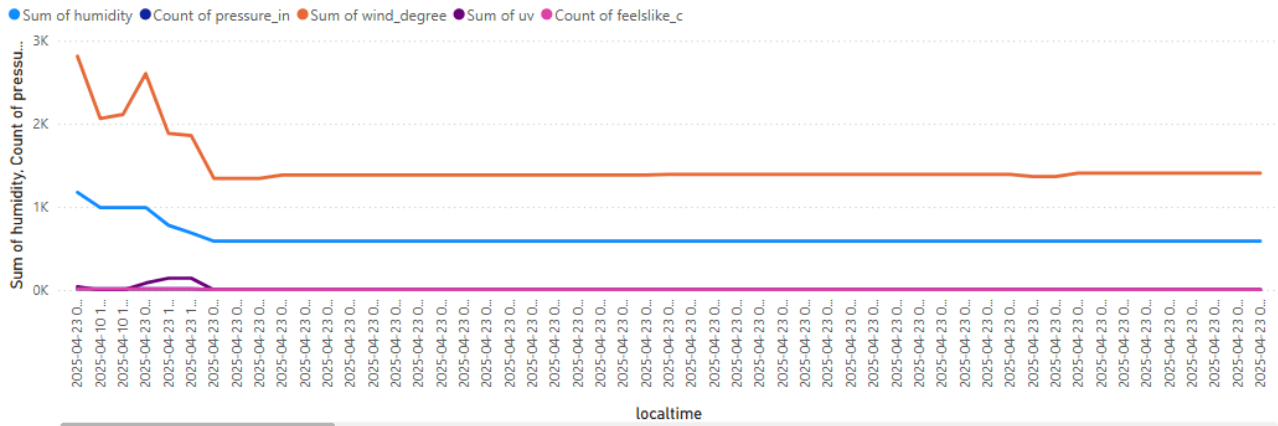
## Step 7:

Create extra Measures and add visualizations to the Report, a simple example is shown below:

# 1643

Count of temp\_c

Sum of humidity, Count of pressure\_in, Sum of wind\_degree, Sum of uv and Count of feelslike\_c by localtime



The Implementation is Done.

Example Run:

The project is Real-time, so it fetches the data every 30 seconds, send it to Event Hub, then to KQL database in Eventhouse, which then gets sent to Power BI.

The Power BI report is real-time as well, it updates every 30 seconds.

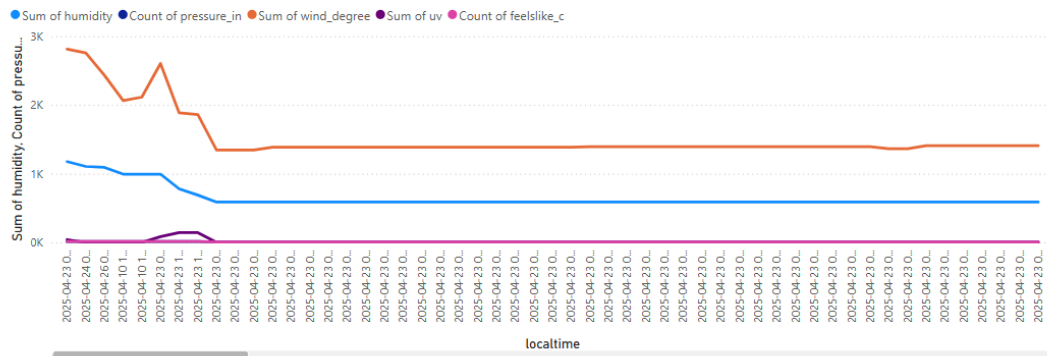
Below is the report at 14:10, April 25<sup>th</sup>, 2025 (Pacific time):

Number of records are 1907.

1907

Count of temp\_c

Sum of humidity, Count of pressure\_in, Sum of wind\_degree, Sum of uv and Count of feelslike\_c by localtime



Below is the report at 14:15, 5 minutes later:

Number of records: 1918.

1918

Count of temp\_c

Sum of humidity, Count of pressure\_in, Sum of wind\_degree, Sum of uv and Count of feelslike\_c by localtime

