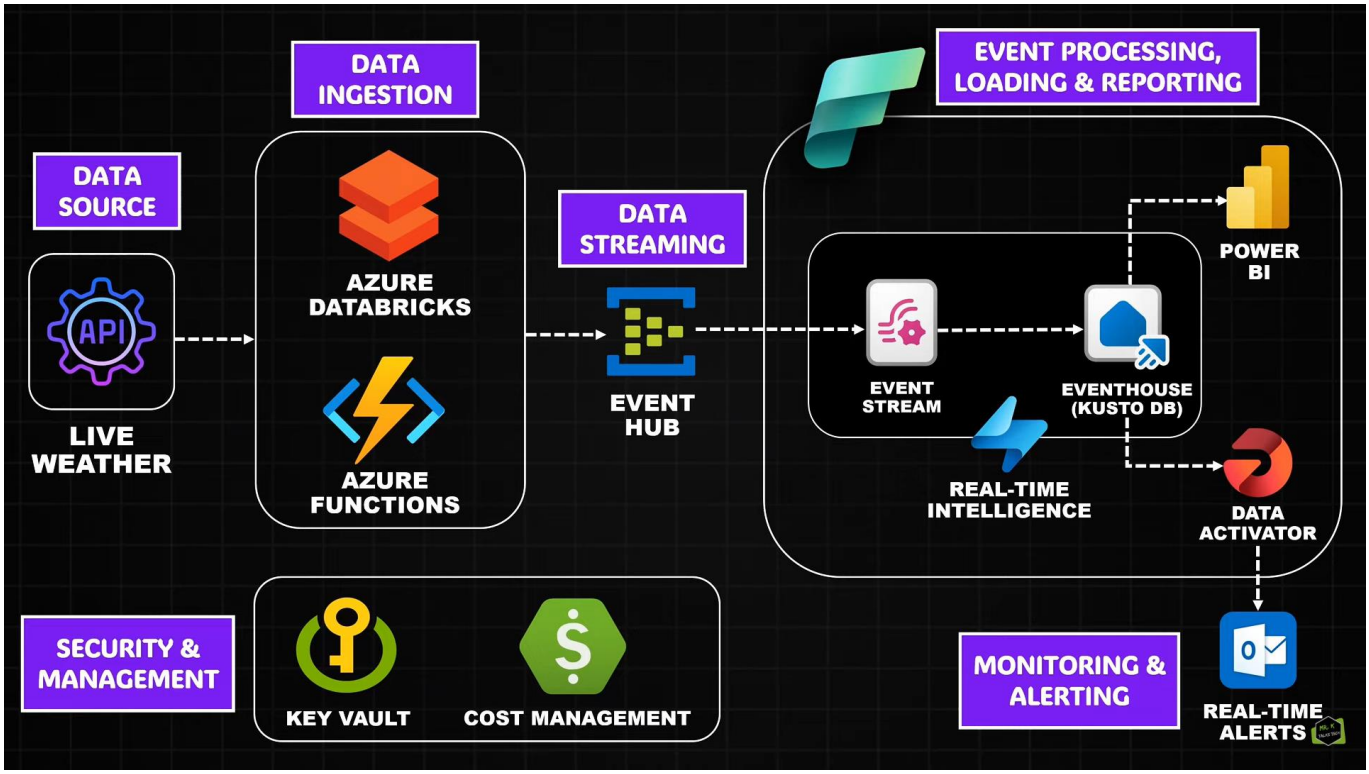
End to End Realtime Streaming Azure Data Engineering Project!

Part 1: Architecture



Welcome to an exciting session where we'll build a real-time streaming Data Engineering project using Azure services! This is one of the best ways to get hands-on experience with real-time streaming while understanding architectural decision-making for cost and performance optimization. In this project, we'll:

- Set up a real-time weather report system using Weather API
- Implement data ingestion using Azure Databricks and Azure Functions
- Use Event Hub for streaming and Microsoft Fabric for real-time intelligence
- Load data into Kusto DB and create an interactive Power BI dashboard
- Set up real-time alerts for extreme weather using Data Activator
- This project covers everything from data source setup to reporting and alerting, providing hands-on skills in modern data engineering! 💻

Agenda:

- Environment Setup
- Data Ingestion
- Cost Analysis & Architectural Decisions
- Real-Time Streaming & Processing
- Power BI Dashboard & Real-Time Alerts

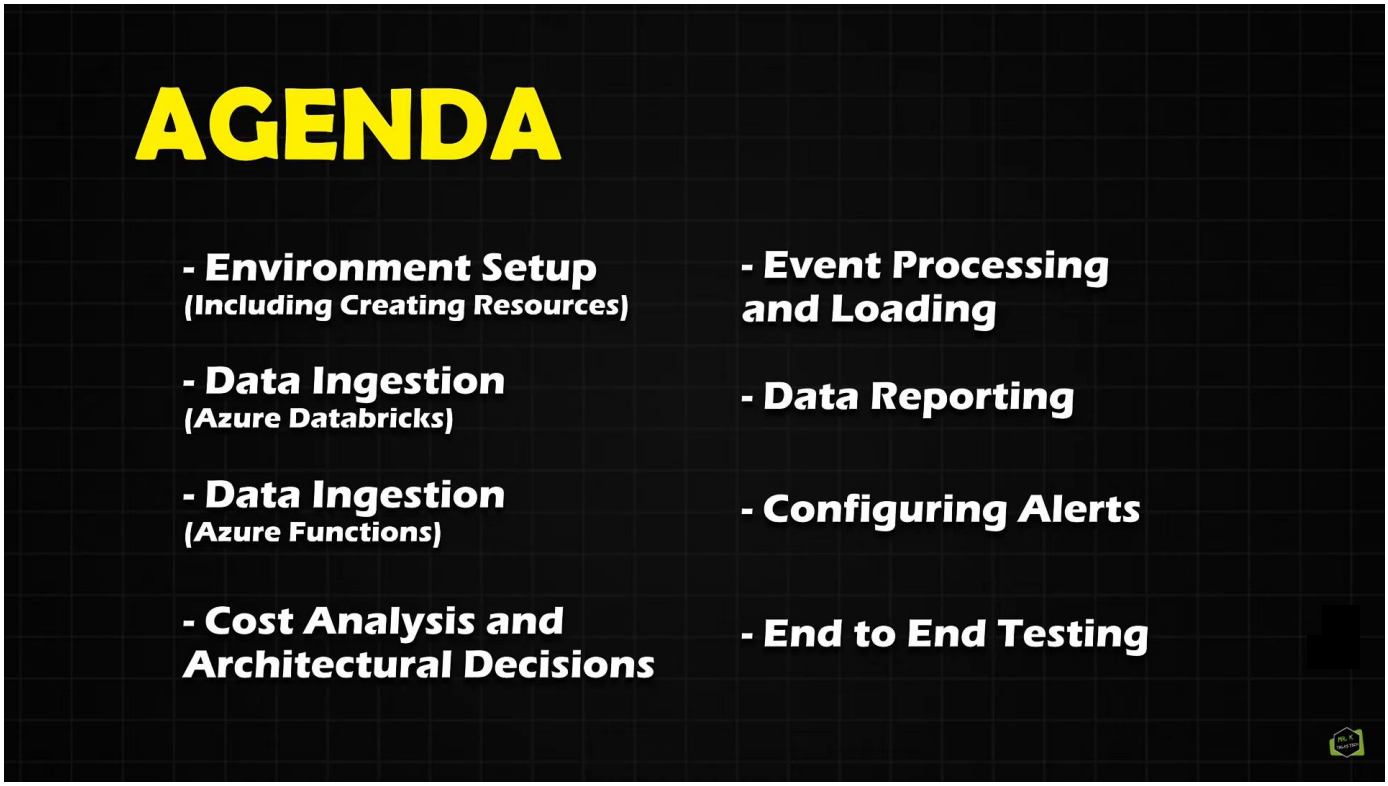
- **Introduction: Project Goal: Build a real-time weather reporting system with email alerts for extreme weather.**
- **Data Source: Weather API (free tier, 1 million API calls/month).**
- **Architecture:**
  - **Data Injection: Azure Data Bricks AND Azure Functions (comparison included).**
  - **Streaming: Azure Event Hub (millions of events/second).**
  - **Processing & Loading: Microsoft Fabric (Event Stream, Synapse dedicated SQL pool).**
  - **Reporting: Power BI (real-time dashboard).**
  - **Alerting: Azure Data Activator (email alerts via KQL queries).**
  - **Security: Azure Key Vault (stores keys and tokens).**
  - **Cost Management: Detailed cost analysis and optimization strategies discussed.**
- **Tools Used: Azure Data Bricks, Azure Functions, Azure Event Hub, Microsoft Fabric (Event Stream, Synapse dedicated SQL pool), Power BI, Azure Data Activator, Azure Key Vault.**
- **Project Phases:**
  - **Environment Setup: Comprehensive setup, addressing previous feedback by including detailed resource creation and data source configuration.**
  - **Data Injection (Azure Data Bricks): Connecting to Weather API, sending data as events to Event Hub.**
  - **Data Injection (Azure Functions): Same as above, for comparison.**
  - **Cost Analysis: Architectural decision-making based on cost and performance.**
  - **Real-time Event Streaming: Event Hub streams data to Microsoft Fabric's Event Stream.**
  - **Data Loading: Event Stream loads data into Synapse dedicated SQL pool.**
  - **Dashboard Creation: Interactive Power BI dashboard displaying real-time weather data.**

- **Alert Configuration: Setting up email alerts using Data Activator and KQL queries.**
- **End-to-End Testing: Thorough testing of the complete system.**

- **Key Learning Points:** Hands-on experience with real-time data streaming, cloud technologies, architectural decision-making, cost optimization, and modern data engineering solutions using Azure and Microsoft Fabric. Project showcases practical skills for potential employers.

API URL: https://www.weatherapi.com/

AGENDA:



Part 2 (Environment Setup):

- Exciting end-to-end real-time streaming Data Engineering project! Here, we'll walk through the entire environment setup process, covering all the essential resources we'll use to build a real-time weather reporting system using Azure services and Microsoft Fabric.

- **Setting up the Weather API**
- **Creating Resource Group**
- **Creating Azure Databricks**
- **Creating Azure Function APP**
- **Creating Azure Event HUB**
- **Creating Azure Key Vault, RBAC and Creating Secrets**
- **Creating Microsoft Fabric**

Summary:

- Her is through setting up a weather API data source and Azure resources for a real-time streaming data engineering project.

Highlights:

- 🌩️ Set up a free weather API from weatherai.com.
- 🔑 Created an Azure Resource Group for project organization.
- 💻 Established an Azure Databricks workspace for data processing.
- 🔁 Created an Azure Function App for executing serverless code.
- 📦 Set up an Azure Event Hub for streaming data ingestion.
- 🔒 Established an Azure Key Vault for secure storage of sensitive information.
- 📊 Enabled Microsoft Fabric in Power BI for data analytics.

Key Insights:

- 🌎 Weather API Setup: Using a free weather API allows for real-time data streaming without financial barriers, enhancing project feasibility.
- 🗂️ Resource Group Management: Organizing resources in Azure Resource Groups simplifies resource management and enhances project clarity.
- 🛠️ Databricks Usage: Databricks facilitates big data processing, making it an ideal choice for handling large datasets efficiently.
- ⏳ Function App Flexibility: Serverless architecture through Azure Function Apps offers a cost-effective way to run code on demand, optimizing resource usage.

- 📈 Event Hub Efficiency: Azure Event Hub's ability to handle thousands of events per second supports scalable data ingestion for real-time applications.
- 🔐 Key Vault Security: Storing sensitive keys in Azure Key Vault ensures secure access management, safeguarding against data breaches.
- 📊 Microsoft Fabric Integration: Leveraging Microsoft Fabric within Power BI enhances data analytics capabilities, allowing for deeper insights and reporting.

Setting up the Weather API:

To set up and use the WeatherAPI, follow these steps:

---

1. Create an Account

- Visit WeatherAPI's website.
- Click Sign Up and create an account using your email address.

---

2. Get Your API Key

- After logging in, navigate to the Dashboard.
- Copy your unique API key. You'll need it to authenticate your requests.

---

3. Choose Your Plan

- Review the available plans (free and paid options) on the Pricing page.
- The free tier provides basic access with limited API calls per month.

---

4. Read the Documentation

- Visit the WeatherAPI documentation to understand how to use its endpoints.
- Common endpoints include:
    - current.json: Get the current weather.
    - forecast.json: Get a weather forecast.
    - history.json: Retrieve historical weather data.

5. Make an API Request

Here's how you can make a basic API request:

```
import requests

# Your API key

API_KEY = "your_api_key_here"

BASE_URL = "http://api.weatherapi.com/v1"

# Example: Get current weather

location = "New York"

url = f"{BASE_URL}/current.json?key={API_KEY}&q={location}"

response = requests.get(url)

if response.status_code == 200:

    data = response.json()

    print("Location:", data['location']['name'])

    print("Temperature (C):", data['current']['temp_c'])

else:

    print("Error:", response.status_code, response.text)
```

b. Using cURL:

```
curl -X GET "http://api.weatherapi.com/v1/current.json?key=your_api_key&q=London"
```

6. Integrate WeatherAPI in Your Application

Use the WeatherAPI in your projects for features like weather tracking, forecasting, or alerts.

For advanced use cases, combine WeatherAPI data with your existing datasets (e.g., in Power BI, Python, or SQL).

Creating Azure Resource Group:

Creating an Azure Resource Group can be done through the **Azure Portal**, **Azure CLI**, **PowerShell**, or using an **SDK like Python**. Below are the instructions for each method:

1. Using Azure Portal (GUI)

1. **Log in to the Azure Portal**:
    - Navigate to Azure Portal.
2. **Search for Resource Groups**:
    - In the search bar at the top, type `Resource Groups` and click on the result.
3. **Create a Resource Group**:

- Click the **+ Create** button at the top.
- Fill in the required details:
  - **Subscription**: Choose the subscription to associate the group.
  - **Resource Group Name**: Enter a unique name for your resource group.
  - **Region**: Select the region (e.g., East US, West Europe) where the resources in this group will reside.

4. **Review and Create**:
   - Click **Review + Create**, verify the details, and then click **Create**.

Creating Azure Databricks

Creating an **Azure Databricks workspace** can be done using the **Azure Portal**, **Azure CLI**, **PowerShell**, or programmatically using the **Azure SDK for Python**. Here's how to do it step by step:

---

1. Using Azure Portal

1. **Log in to Azure Portal**:
   - Go to Azure Portal.
2. **Search for Databricks**:
   - In the search bar, type **Azure Databricks** and select it from the results.
3. **Create an Azure Databricks Workspace**:
   - Click the **+ Create** button.
4. **Fill in Required Details**:
   - **Subscription**: Select your Azure subscription.
   - **Resource Group**: Choose an existing resource group or create a new one.
   - **Workspace Name**: Provide a unique name for the Databricks workspace.
   - **Region**: Select the Azure region where the workspace will be hosted.
   - **Pricing Tier**: Choose the desired pricing tier (e.g., Standard, Premium, or Trial).
5. **Networking (Optional)**:
   - If needed, configure advanced networking options, such as VNet injection.
6. **Review and Create**:
   - Click **Review + Create**, verify the details, and click **Create**.

Creating Azure Function APP

To create an **Azure Function App**, you can use the **Azure Portal**, **Azure CLI**, **PowerShell**, or an SDK like Python. Here's how to set it up:

---

1. Using Azure Portal

1. **Log in to the Azure Portal**:

    - Go to Azure Portal.
2. **Search for Function App**:
    - In the search bar, type **Function App** and select it.
3. **Create a Function App**:
    - Click the **+ Create** button.
4. **Fill in the Details**:
    - **Subscription**: Select your Azure subscription.
    - **Resource Group**: Select an existing one or create a new resource group.
    - **Function App Name**: Enter a globally unique name (e.g., `myfunctionapp123`).
    - **Region**: Choose the Azure region where your app will be deployed.
    - **Runtime Stack**: Choose your preferred runtime (e.g., Python, .NET, Node.js, Java).
    - **Version**: Select the version of the runtime stack.
    - **Operating System**: Choose **Windows** or **Linux**.
    - **Plan Type**: Choose **Consumption** (serverless) or **Premium/Elastic**.
5. **Storage Account**:
    - Create a new storage account or select an existing one.
6. **Review and Create**:
    - Click **Review + Create**, review the configuration, and click **Create**.

Creating Azure Event HUB

To create an **Azure Event Hub**, you can use the **Azure Portal**, **Azure CLI**, **PowerShell**, or programmatically via the **Azure SDK for Python**. Here's how to do it:

---

1. Using Azure Portal

1. **Log in to Azure Portal**:
    - Go to Azure Portal.
2. **Search for Event Hubs**:
    - In the search bar, type **Event Hubs** and select **Event Hubs** under the **Azure Services** section.
3. **Create an Event Hubs Namespace**:
    - Click **+ Create**.
    - Fill in the details:
        - **Subscription**: Choose your Azure subscription.
        - **Resource Group**: Select an existing one or create a new resource group.
        - **Namespace Name**: Provide a unique name for your namespace.
        - **Region**: Select the Azure region for the namespace.
        - **Pricing Tier**: Choose the pricing tier (Basic, Standard, or Premium).
4. **Create the Namespace**:
    - Click **Review + Create**, then click **Create** to deploy the namespace.
5. **Create an Event Hub**:
    - Once the namespace is created, go to the namespace in the portal.
    - Under **Entities**, select **Event Hubs** and click **+ Event Hub**.
    - Provide the **Event Hub Name**, and configure optional settings like **Partition Count** and **Message Retention**.
    - Click **Create**.

Creating Azure Key Vault, RBAC and Creating Secrets

Here's how you can **create an Azure Key Vault**, configure **RBAC (Role-Based Access Control)**, and store **secrets** step-by-step using different methods.

1. Creating Azure Key Vault

Using Azure Portal

1. **Log in to Azure Portal**:
   - Go to Azure Portal.
2. **Search for Key Vaults**:
   - In the search bar, type **Key Vaults** and select it.
3. **Create a Key Vault**:
   - Click **+ Create**.
   - Fill in the required fields:
     - **Subscription**: Select your subscription.
     - **Resource Group**: Create or select an existing resource group.
     - **Key Vault Name**: Provide a globally unique name.
     - **Region**: Select your preferred region.
     - **Pricing Tier**: Choose **Standard** or **Premium**.
   - Click **Review + Create**, then **Create**.

Creating Microsoft Fabric

Steps to Create and Configure Microsoft Fabric in Your Organization

Microsoft Fabric provides a unified data and analytics platform that integrates data engineering, data warehousing, real-time analytics, and business intelligence tools. Here's how you can create and set it up:

---

1. Prerequisites

Before creating Microsoft Fabric, ensure the following:

1. You have **admin privileges** in the Microsoft 365 tenant.
2. You have an active **Microsoft Fabric license** (available with Power BI Premium or Fabric trial licenses).
3. Azure Active Directory (Azure AD) is configured for your organization.

---

2. Enabling Microsoft Fabric

Microsoft Fabric is an add-on feature within Power BI and requires Power BI Admin access to enable it.

Using the Microsoft 365 Admin Center

1. **Log in** to the Microsoft 365 Admin Center.
2. Navigate to **Settings** → **Org settings**.
3. Under **Services**, select **Microsoft Fabric**.
4. Enable the **Microsoft Fabric preview** feature and save the settings.

---

3. Licensing for Microsoft Fabric

Microsoft Fabric is available in the following subscriptions:

- **Power BI Premium Per Capacity**
- **Power BI Premium Per User**
- Free trial is available for organizations exploring Fabric.

Assigning Fabric Licenses:

1. Go to the Microsoft 365 Admin Center.
2. Navigate to **Billing** → **Licenses**.
3. Assign **Power BI Premium (Fabric)** licenses to users who will work with Microsoft Fabric.

---

4. Creating a Microsoft Fabric Workspace

To create a Fabric workspace for collaboration:

Using Power BI Service

1. Go to Power BI Service.
2. Click on **Workspaces** → **Create a workspace**.
3. Provide a **name** for the workspace and assign it a **Premium Capacity** (this enables Fabric features).
4. Configure permissions for users who can access the workspace.

Workspace Settings for Fabric:

- Enable **Microsoft Fabric features** like Data Factory, Synapse Data Engineering, Data Science, or Real-Time Analytics under the workspace settings.

---

5. Configuring Fabric Components

Once the Fabric workspace is set up, you can start using its components:

Fabric Data Engineering (Dataflows and Pipelines)

1. Navigate to **Power BI Workspace**.
2. Select **Create** → **Dataflow Gen2** or **Pipeline**.
3. Use connectors to load and transform data from sources (Azure Data Lake, SQL Server, Databricks, etc.).

Fabric Data Science

1. Go to the workspace and create a **Notebook**.
2. Use Python, R, or Spark to perform machine learning or advanced data analytics tasks.

Fabric Data Warehousing

1. Create a **Lakehouse** in the workspace to store structured and unstructured data.
2. Use T-SQL to query and analyze data.

Real-Time Analytics

1. Create **KQL databases** for real-time analytics.
2. Use streaming data sources such as Azure Event Hubs or IoT devices.

Power BI Integration

1. Build Power BI reports and dashboards directly from Fabric datasets.
2. Use **DirectQuery** or **Import** modes for live and fast data access.

---

6. Configuring Security

Enable Row-Level Security (RLS):

1. Within Power BI, define roles to restrict access to specific data rows.
2. Assign users or groups to these roles.

Data Protection Policies:

- Use **Microsoft Purview** to classify and protect sensitive data across Fabric components.

---

7. Monitoring and Optimization

Use Fabric Admin Portal:

- Monitor resource usage, workspace capacities, and Fabric features via the **Admin Portal**.
- Optimize queries and workloads using **Fabric Performance Analyzer**.

---

8. Automating Deployment

Use DevOps pipelines to deploy Fabric assets:

1. Export workspace assets using **Power BI REST APIs** or the **Fabric CLI**.
2. Use **Azure DevOps** or **GitHub Actions** for version control and CI/CD pipelines.

Next Steps

- **Build and Test**: Start building pipelines, notebooks, and Power BI dashboards using your Fabric resources.
- **Collaborate**: Share workspaces with team members and assign roles for effective collaboration.
- **Learn More**: Explore features like **Data Factory**, **Synapse Notebooks**, and **KQL databases**.

Let me know if you need help configuring a specific Fabric component!

Part -3: Data Ingestion using Azure Databricks

Step 1: Create and setup Event Hub

Let's got to event hub

Create new event hub---->click +

Basics **Capture (not support basic version)** **Review+Create**

Name

Partition count: 1

Retention

Cleanup policy: Delete

Retention time: 1

Now you see the EventHub is Active

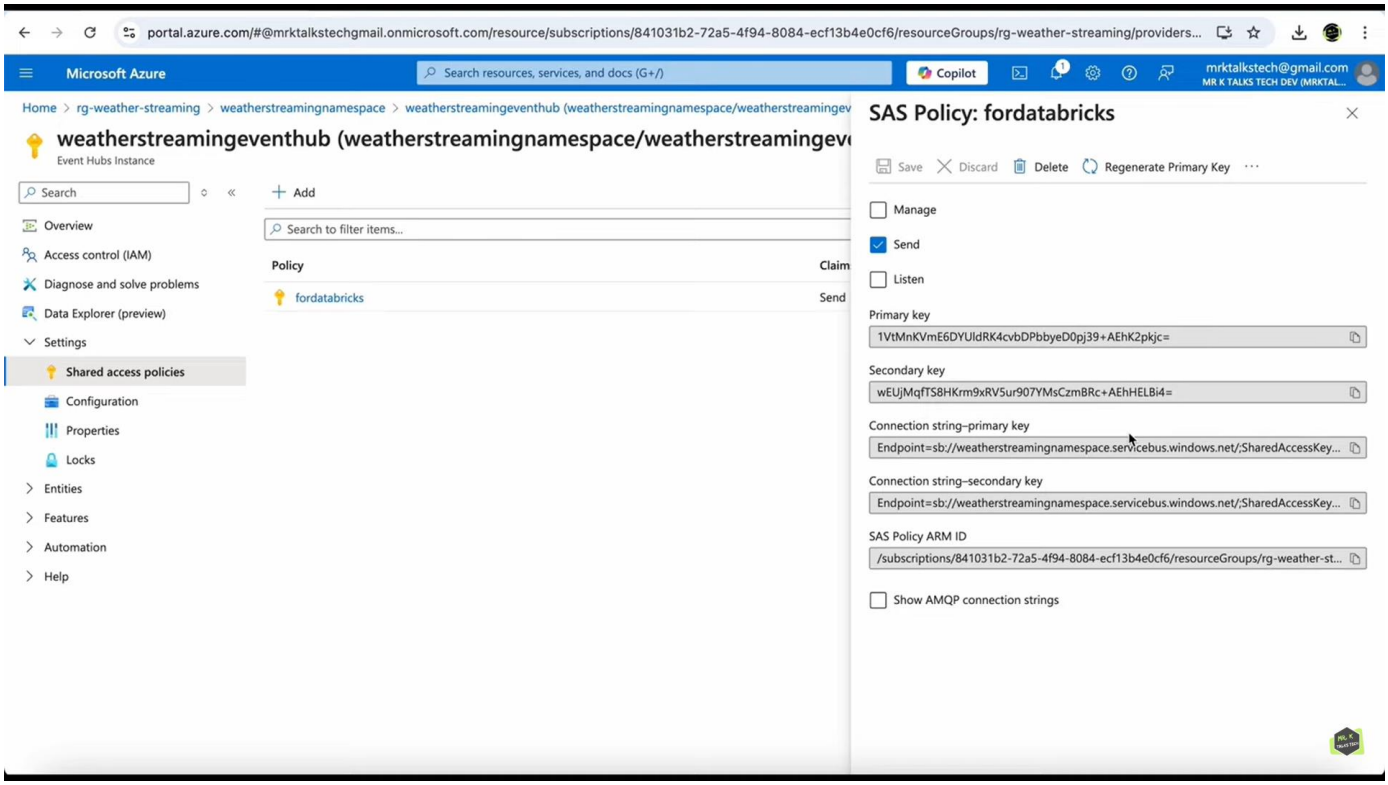
And click on

Firstly to provide required permission for azure data bricks to access this event hub but sadly this access cannot be given using the RBAC not supported currently so for this reason we have to use required access.

Next step is configure the event hub for data ingestion.

Goto setting--->Shared access Policies--->add--->add SAS Policy--->Policy Name(for Databricsk)--->send permission.

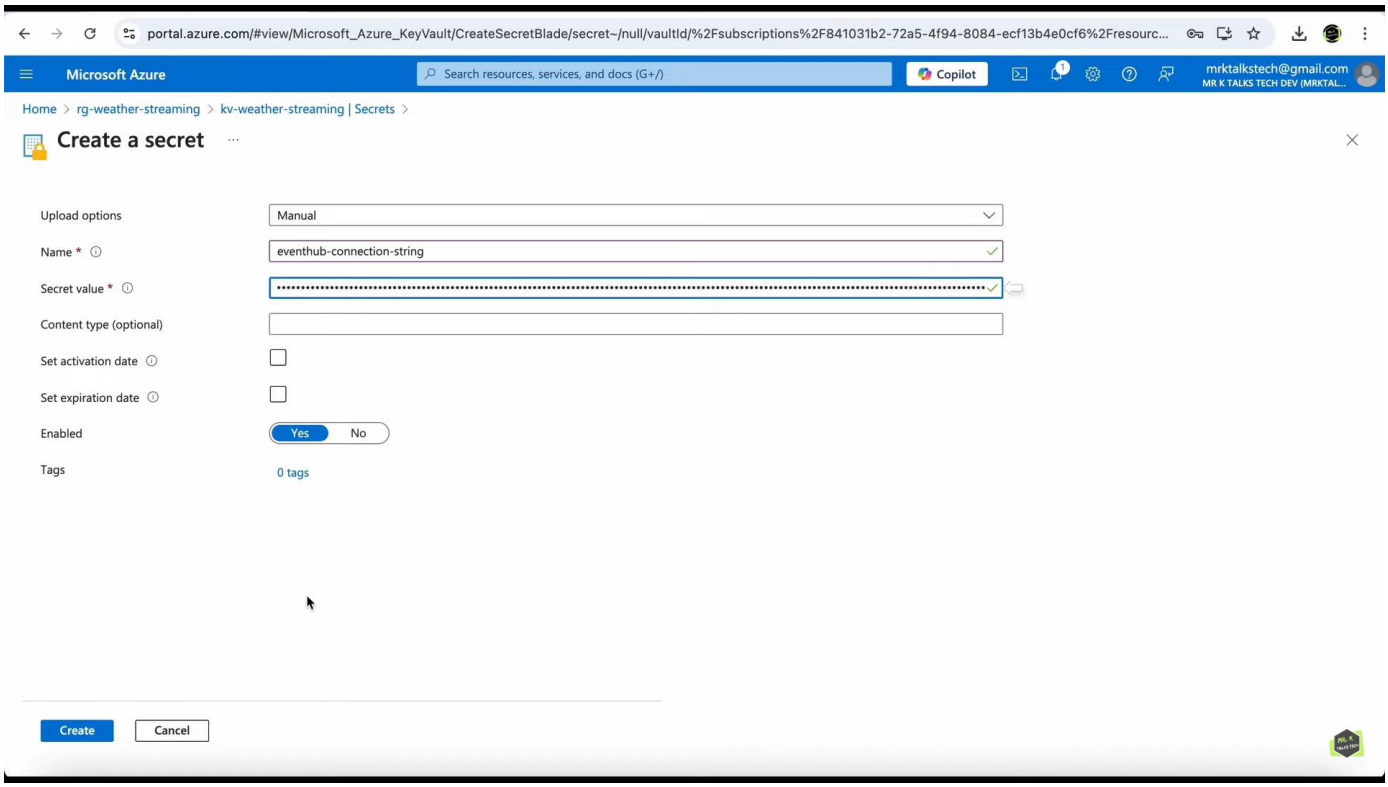
Policy--->



Keys and connection strings are sensitive values and secure.

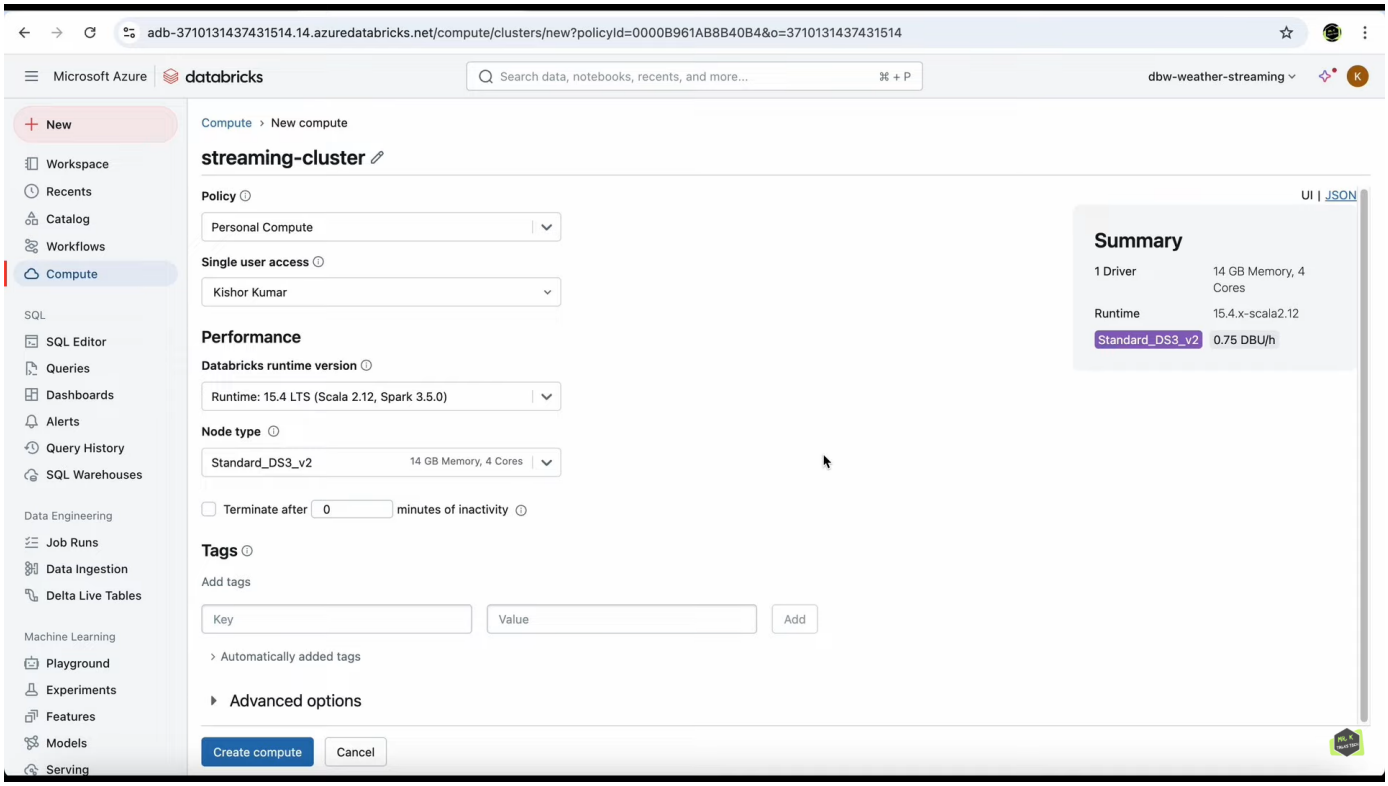
Copy the key--->

Open key vault--->generate/import



Step 2: Create Cluster in DataBricks

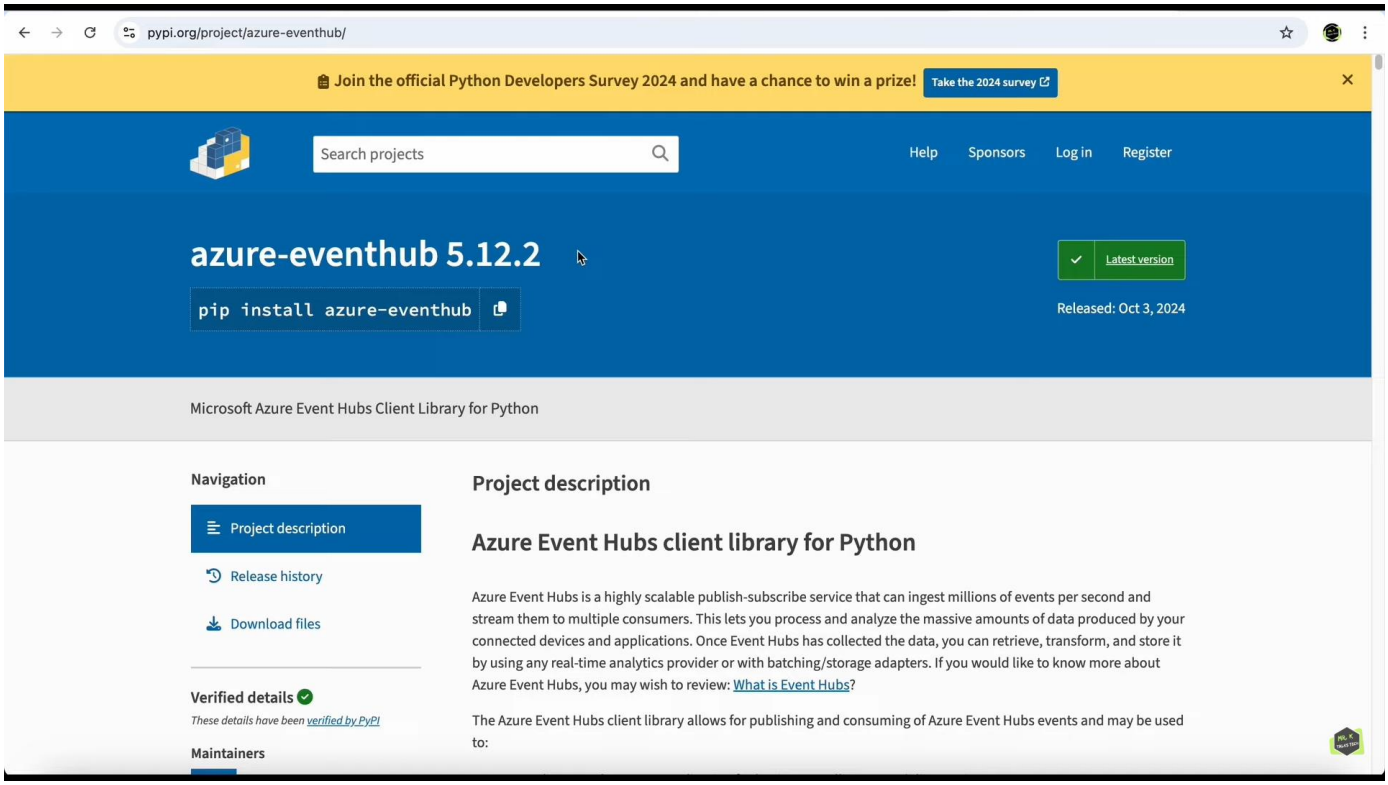
Lets go RG--->click DB Resource--->click on Launch Workspace--->goto compute--->create new.

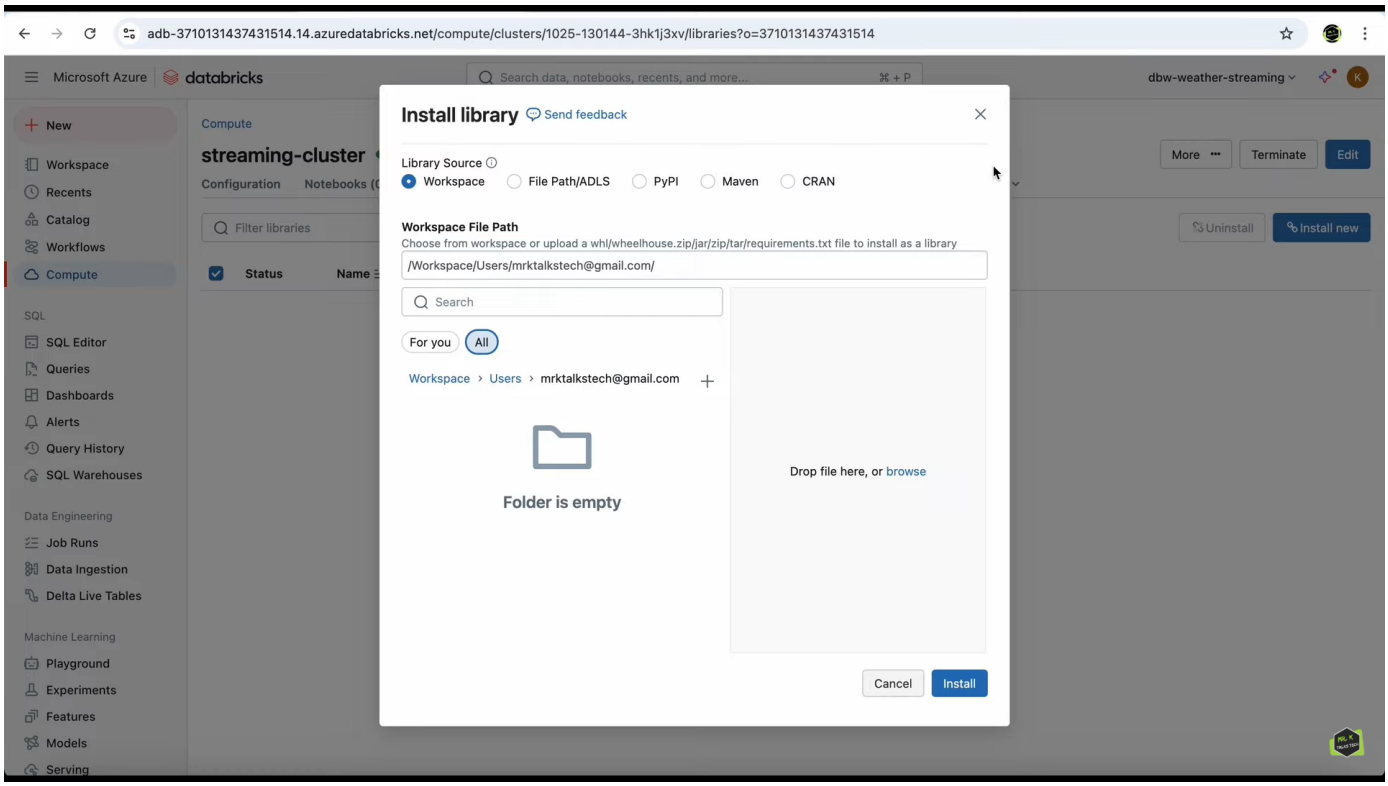


Step 3: Install Event HUB Libraries in Cluster.

Installing the event Hub Libraries in the cluster so basically as part of the data ingestion azure data bricks and write the python code in notebook to send the events to the event HUB so for sending the events we need to use a python library but by default in DB that specific library is not there then install it from an external package so that's what we are going to do in the.

Lets Go and Open streaming clustering and here we will libraries





Click on install option--->here we have PyPi package

Copy that and paste it on --->Package(azure-eventhub==5.12.2)

Click install button.

Installed successfully azure event library.

Step 4: Sending test event from DB to Event HUB

Restart cluster--->Cluster--->Click on three dots--->Restart.

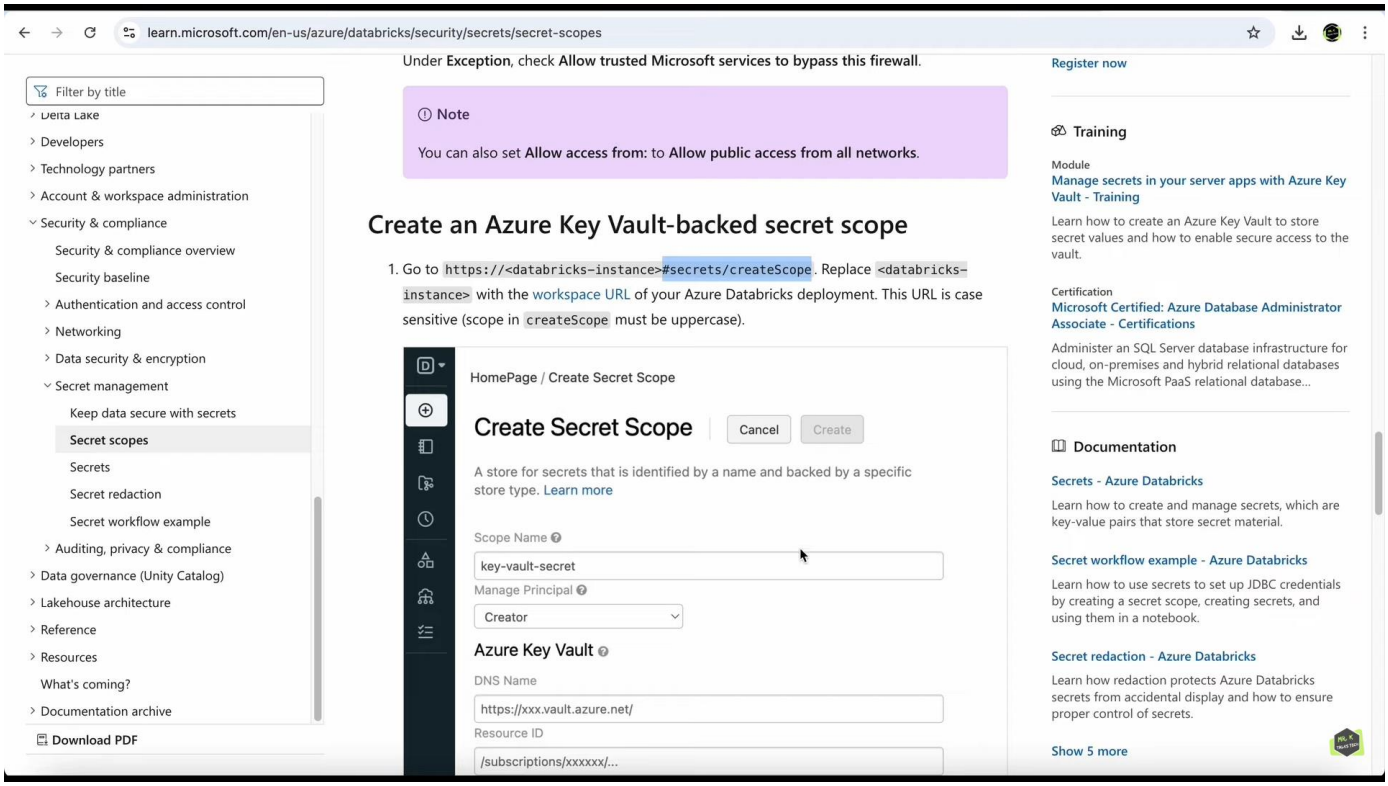
Lets goto RG--->Event HUB Namespace--->click on Event Hub which we created--->data explorer.

Lets goto--->DB--->workspace--->folder--->create(Notebook)--->weather-streaming-notebook.

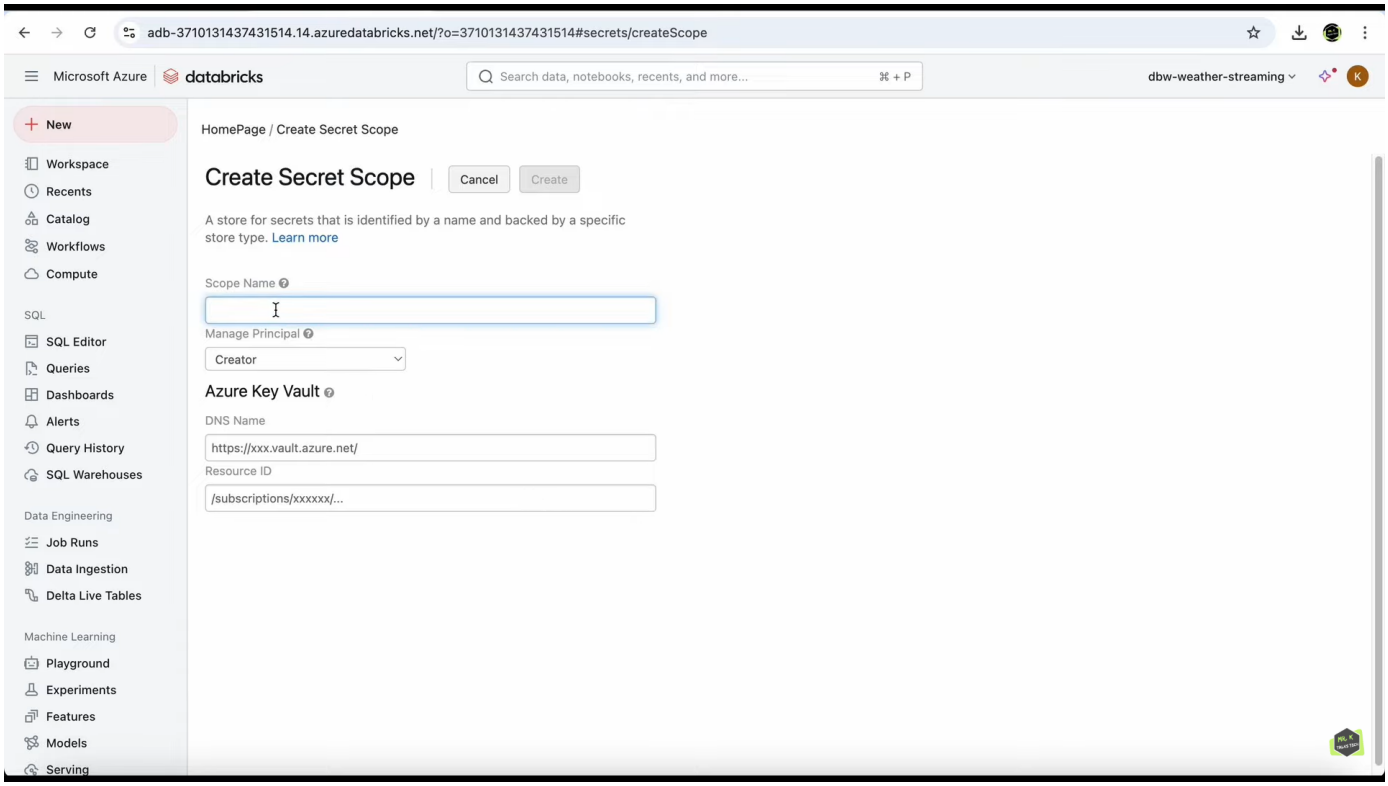
```
from azure.eventhub import EventhubProducerClient, EventData

Import json

Event_hub_connection_string = “xxxx” #copy from event hub Shared access policy connection string

Event_hub_name = ‘xxxx’ #event HUB name

Producer = EventhubProducerClient.from_connection_string(com_str= connection, 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)


#sample JSON event

Event = {

        “event_id”: 1111,

        “event_name”: :Test Event”,

}

#send the event

Send_event(event)

#close the producer

Producer.close()
```

Step 5: configure key Vault Secrets in DB

To establish secret scope



https://adb-{workspace_id}.azuredatabricks.net#secret/createScope



Lets go got key vault:

Goto settings--->properties--->vault and got o data bricks--->in DNS Name(Paste it)

Resource_Id--->paste it in(secret scope)--->create

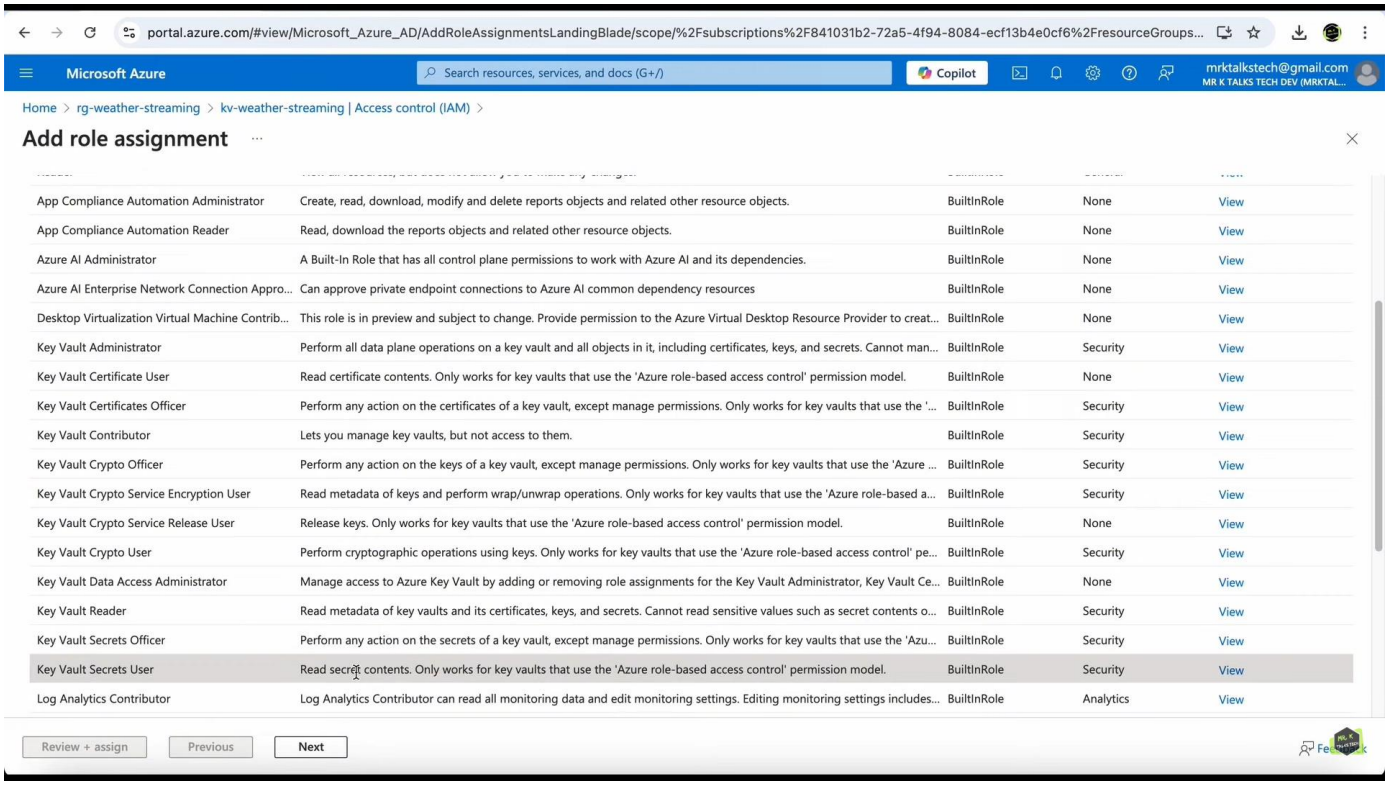
Now replace to hardcode to get the connection string value from the key-vault secret we can

```
from azure.eventhub import EventhubProducerClient, EventData

import json

#getting secret value from key vault.

eventhub_connection_string = dbutils.secrets.get(scope="key-vault-scope",key="eventhub-connection-string")

Event_hub_name = 'weatherstreamingeventhub #event HUB name

Producer = EventhubProducerClient.from_connection_string(com_str= connection, 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)


#sample JSON event

event = {

    "event_id": 2222,

    "event_name": "key vault Test",

}

#send the event

Send_event(event)

#close the producer

Producer.close()
```

Code failed

But this DB workspace doesn't have required permission that's why permission denied.

Goto key vault--->IAM--->add new role assignment--->key vault secrets User--->Next button.

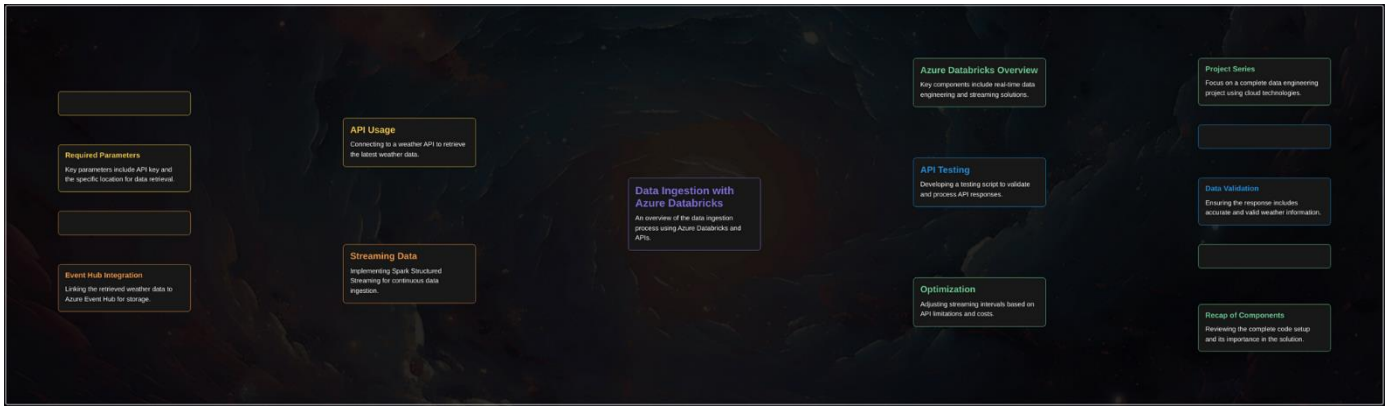


Select number--->azure Databricks--->click and assign

And lets jump into ADB--->run the notebook.

Lets goto RG group--->Event HUB name space--->event HUB(weatherstreamingeventHub)--->data explorer--->View Events--->check the data.

Part- 4 (Data Ingestion using Azure Databricks)- End to End Streaming Azure Data Engineering Project.



Step 6: Weather API Testing in DB.

Lets go to weather App----->weatherapi.com--->docs--->Authentication--->request--->Request Parameters

API Testing:

```
import requests

import json

#getting secret value from key vault

Weatherapikey = dbutils.secret.get(scope="key-value-scope", key="weatherapikey")

location="hyd" #you can replace with city name based on your preference

base_URL= "http:/api.weatheapi.com/v1/"

current_weather_url = f"{base_URL}/current.json"

params = {
                'key':weatherkey,
                'q': location,
        }

response = request.get(base_URL, params=params)

if response.status_code==200:

        current_weather=response.json()

        print("current weathe")
```

```
        print(json.dumps(current_weather, indent=3))

else:

        print(f"Error: {response.status_code}, {response.text}")
```

let's goto API weather

three main types of weather data

1. Current weather
2. Forecast
3. Alerts
4. Along with air quality index

Step 7: Developing complete code for getting weather Data

%md

Complete code for getting weather data

```
import requests

import json

#function to handle the API response

def handle_responce(response):

        if response.status_code==200:

                return response.json()

        else:

                return f"Erro: {response.status_code}, {response.text}"

#function to get current weather and air quality data

def get_current_weather(base_URL, api_key, location):

        current_weather_url  =f"{base_URL}/current.json"

        params = {
                        'key': api_key,
                        'q': location,
                        "aqi": 'yes',
                }

        response = request.get(base_URL, params=params)

        return handle_response(response)
```

```
#function to get forecast Data

def get_forecast_weather(base_URL, api_key, location, days):

        forecast_url = f"{base_url}/forecast.json"

                params = {
                        'key': api_key,
                        'q': location,
                        "days":  days,
                }

        response = request.get(forecast_URL, params=params)

        return handle_response(response)

#function to get alerts

def get_alerts(base_URL, api_key, location):

        forecast_url = f"{base_url}/alerts.json"

                params = {
                        'key': api_key,
                        'q': location,
                        "alerts":  'yes',
                }

        response = request.get(alert_URL, params=params)

        return handle_response(response)

# Flatten and merge the data

def flatten_data(current_weather, forecast_weathe, 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 = alert.get("alerts", {}). get("alerts", {})



        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(‘lot’),

‘location’: location_data.get(‘location’),

‘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),

‘is_day’: current.get(‘is_day’),

‘presurre_in’: current.get(‘presurre_in’),

‘precip_in’: current.get(‘precip_in’),

‘humidity’: current.get(‘humidity’),

‘cloud’: current.get(‘cloud’),

‘feelslike_c’: current.get(‘feelslike_c’),

‘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)

},

‘alerts’: [

```
                {
                    'headline': alert.get('headline'),
                    'severity': alert.get('severity'),
                    'description': alert.get('description'),
                    'instructions': alert.get('instructions')
                }
                for alert in alert_list
            ],
            'forecast': [
                {
                    'date': day.get('date'),
                    'maxtemp_c': day.get('day',{}).get('maxtemp_c'),
                    'mintemp_c': day.get('day',{}).get('mintemp_c'),
                    'condition: day.get('day',{}).get('condition', {}.get('text')
                }
                for day in forecast
            ]
            return flattend_data
#main program
def fetch_weather_data():
    base_url = http://api.weatherapi.com/v1/"
    location = "hyd" # you can replace with any city name based on your preference
    weatherapikey = dbutils.secrets.get(scope="key-vault-scope", key="weatherapikey")
    # Get data from API
    Current_weather  = get_current_weather(base_url, weatherapikey, location)
    Forecast_weather = get_forecats_weather(base_url, weatherapikey, location,3)
    Alerts = get_alerts(base_url,weatherapikey, location)
```

```
    # Flatten and merge data

    Merged_data = flatten_data(current_weather, forecast_weather, alerts)

    print("weather Data:", json.dumps(merged_data, indent=3))
```

#calling the main program

```
fetch_weather_data()
```

we can run the code.

Step 8: Final Data Ingestion from DB to Event HUB

Split into three steps.

Weather data and also we have a script to send a test event to the event hub now.

1. Sending the complete weather data to the event HUB

#Copy the above code but the code which sending above data output is here only.

```
import requests

import json

from azure.eventhub import EventhubProducerClient, EventData
```

#getting secret value from key vault.

```
eventhub_connection_string = dbutils.secrets.get(scope="key-vault-scope",key="eventhub-connection-string")

Event_hub_name = 'weatherstreamingeventhub #event HUB name

Producer = EventhubProducerClient.from_connection_string(com_str= connection, 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_responce(response):
    if response.status_code==200:
        return response.json()
    else:
        return f"Erro: {response.status_code}, {response.text}"
#function to get current weather and air quality data
def get_current_weather(base_URL, api_key, location):
    current_weather_url  =f"{base_URL}/current.json"
    params = {
            'key': api_key,
            'q': location,
            "aqi": 'yes',
        }
    response = request.get(base_URL, params=params)
    return handle_response(response)
#function to get forecast Data
def get_forecast_weather(base_URL, api_key, location, days):
    forecast_url = f"{base_url}/forecast.json"
        params = {
            'key': api_key,
            'q': location,
            "days":  days,
        }
    response = request.get(forecast_URL, params=params)
    return handle_response(response)
#function to get alerts
def get_alerts(base_URL, api_key, location):
    forecast_url = f"{base_url}/alerts.json"
        params = {
            'key': api_key,
            'q': location,
            "alerts":  'yes',
        }
    response = request.get(alert_URL, params=params)
```

```
        return handle_response(response)
# Flatten and merge the data
def flatten_data(current_weather, forecast_weathe, 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 = alert.get(“alerts”, {}). get(“alerts”, {})
        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(‘lot’),
                ‘location’: location_data.get(‘location’),
                ‘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),
                ‘is_day’: current.get(‘is_day’),
                ‘presurre_in’: current.get(‘presurre_in’),
                ‘precip_in’: current.get(‘precip_in’),
                ‘humidity’: current.get(‘humidity’),
                ‘cloud’: current.get(‘cloud’),
```

```
'feelslike_c': current.get('feelslike_c'),
'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)
        },
        'alerts': [
        {
                'headline': alert.get('headline'),
                'severity': alert.get('severity'),
                'description': alert.get('description'),
                'instructions': alert.get('instructions')
        }
        for alert in alert_list
],
'forecast': [
        {
                'date': day.get('date'),
                'maxtemp_c': day.get('day',{}).get('maxtemp_c'),
                'mintemp_c': day.get('day',{}).get('mintemp_c'),
                'condition: day.get('day',{}).get('condition', {}.get('text')
        }
        for day in forecast
```

```
                    ]

                    return flattend_data

#main program

def fetch_weather_data():

        base_url = http://api.weatherapi.com/v1/"

        location = "hyd" # you can replace with any city name based on your preference

        weatherapikey = dbutils.secrets.get(scope="key-vault-scope", key="weatherapikey")

        # Get data from API

        Current_weather  = get_current_weather(base_url, weatherapikey, location)

        Forecast_weather = get_forecats_weather(base_url, weatherapikey, location,3)

        Alerts = get_alerts(base_url,weatherapikey, location)

        # Flatten and merge data

        Merged_data = flatten_data(current_weather, forecast_weather, alerts)

        #print("weather Data:", json.dumps(merged_data, indent=3))

        #sending the weather data to event HUB

        send_event(merger_data)

#calling the main program

fetch_weather_data()
```

after getting the API data this merged data will be event to the event HUB single event.

First let's go to RG--->eventHUB--->event--->dataexplorer--->view events

Let's see the event in testing.

2. Sending the weather data in streaming.

```
import requests

import json

from azure.eventhub import EventhubProducerClient, EventData

#getting secret value from key vault.
```

```
eventhub_connection_string = dbutils.secrets.get(scope="key-vault-scope",key="eventhub-connection-string")

Event_hub_name = 'weatherstreamingeventhub #event HUB name

Producer = EventhubProducerClient.from_connection_string(com_str= connection, 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_responce(response):

    if response.status_code==200:

        return response.json()

    else:

        return f"Erro: {response.status_code}, {response.text}"

#function to get current weather and air quality data

def get_current_weather(base_URL, api_key, location):

    current_weather_url  =f"{base_URL}/current.json"

    params = {
            'key': api_key,
            'q': location,
            "aqi": 'yes',
        }

    response = request.get(base_URL, params=params)

    return handle_response(response)

#function to get forecast Data

def get_forecast_weather(base_URL, api_key, location, days):

    forecast_url = f"{base_url}/forecast.json"

        params = {
```

```
                'key': api_key,
                'q': location,
                "days":  days,
            }

    response = request.get(forecast_URL, params=params)

    return handle_response(response)
```

#function to get alerts

```
def get_alerts(base_URL, api_key, location):

    forecast_url = f"{base_url}/alerts.json"

        params = {
            'key': api_key,
            'q': location,
            "alerts":  'yes',
        }

    response = request.get(alert_URL, params=params)

    return handle_response(response)
```

 # Flatten and merge the data

```
def flatten_data(current_weather, forecast_weathe, 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 = alert.get("alerts", {}). get("alerts", {})


    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('lot'),
```

‘location’: location_data.get(‘location’),

‘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),

‘is_day’: current.get(‘is_day’),

‘presurre_in’: current.get(‘presurre_in’),

‘precip_in’: current.get(‘precip_in’),

‘humidity’: current.get(‘humidity’),

‘cloud’: current.get(‘cloud’),

‘feelslike_c’: current.get(‘feelslike_c’),

‘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)

},

‘alerts’: [

{

‘headline’: alert.get(‘headline’),

‘severity’: alert.get(‘severity’),

```
                'description': alert.get('description'),
                'instructions': alert.get('instructions')
            }
            for alert in alert_list
        ],
        'forecast': [
            {
                'date': day.get('date'),
                'maxtemp_c': day.get('day',{}).get('maxtemp_c'),
                'mintemp_c': day.get('day',{}).get('mintemp_c'),
                'condition: day.get('day',{}).get('condition', {}.get('text')
            }
            for day in forecast
        ]
        return flattend_data
#this is not main program
def fetch_weather_data():
    base_url = http://api.weatherapi.com/v1/"
    location = "hyd" # you can replace with any city name based on your preference
    weatherapikey = dbutils.secrets.get(scope="key-vault-scope", key="weatherapikey")
    # Get data from API
    Current_weather  = get_current_weather(base_url, weatherapikey, location)
    Forecast_weather = get_forecats_weather(base_url, weatherapikey, location,3)
    Alerts = get_alerts(base_url,weatherapikey, location)
    # Flatten and merge data
    Merged_data = flatten_data(current_weather, forecast_weather, alerts)
    #print("weather Data:", json.dumps(merged_data, indent=3))
```

```
        #sending the weather data to event HUB
        return merged_data


#main program
def process_batch(batch_df, batch_id):
        try:
                # fetch weather data
                Weather_data  = fetch_weather_data()
                #send the weather data (current weather part)
                Send_event(weather_data)
        except Exception as e:
                print(f"Error sending events in batch {batch_id}: {str(e)}")
                raise e
#set up a streaming source (for example, rate source for testing purposes)
Streaming_df  =spark.readStream.foramt("rate").option("rowsperSecon",1).load()
#write the streaming data using foreachBatch to send weather data to Event HUB
Query = streamin_df.writeStream.foreachBatch(process_batch).start()
Query.awaitTermination()
#close the producer after termination
Producer.close()
```

3. final step every 30 seconds sending the weather data to event HUB in every 30 seconds.

```
import requests
import json
from datetime import datetime, timedelta
from azure.eventhub import EventhubProducerClient, EventData
#getting secret value from key vault.
```

```
eventhub_connection_string = dbutils.secrets.get(scope="key-vault-scope",key="eventhub-connection-string")

Event_hub_name = 'weatherstreamingeventhub #event HUB name

Producer = EventhubProducerClient.from_connection_string(com_str= connection, 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_responce(response):

    if response.status_code==200:

        return response.json()

    else:

        return f"Erro: {response.status_code}, {response.text}"

#function to get current weather and air quality data

def get_current_weather(base_URL, api_key, location):

    current_weather_url  =f"{base_URL}/current.json"

    params = {
            'key': api_key,
            'q': location,
            "aqi": 'yes',
        }

    response = request.get(base_URL, params=params)

    return handle_response(response)

#function to get forecast Data

def get_forecast_weather(base_URL, api_key, location, days):

    forecast_url = f"{base_url}/forecast.json"

        params = {
```

```
                'key': api_key,
                'q': location,
                "days":  days,
            }

    response = request.get(forecast_URL, params=params)

    return handle_response(response)
```

#function to get alerts

```
def get_alerts(base_URL, api_key, location):

    forecast_url = f"{base_url}/alerts.json"

        params = {
            'key': api_key,
            'q': location,
            "alerts":  'yes',
        }

    response = request.get(alert_URL, params=params)

    return handle_response(response)
```

 # Flatten and merge the data

```
def flatten_data(current_weather, forecast_weathe, 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 = alert.get("alerts", {}). get("alerts", {})


    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('lot'),
```

‘location’: location_data.get(‘location’),

‘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),

‘is_day’: current.get(‘is_day’),

‘presurre_in’: current.get(‘presurre_in’),

‘precip_in’: current.get(‘precip_in’),

‘humidity’: current.get(‘humidity’),

‘cloud’: current.get(‘cloud’),

‘feelslike_c’: current.get(‘feelslike_c’),

‘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)

},

‘alerts’: [

{

‘headline’: alert.get(‘headline’),

‘severity’: alert.get(‘severity’),

```
                    'description': alert.get('description'),
                    'instructions': alert.get('instructions')
                }
                for alert in alert_list
            ],
            'forecast': [
                {
                    'date': day.get('date'),
                    'maxtemp_c': day.get('day',{}).get('maxtemp_c'),
                    'mintemp_c': day.get('day',{}).get('mintemp_c'),
                    'condition: day.get('day',{}).get('condition', {}.get('text')
                }
                for day in forecast
            ]
            return flattend_data
#This is not main program
def fetch_weather_data():
    base_url = http://api.weatherapi.com/v1/"
    location = "hyd" # you can replace with any city name based on your preference
    weatherapikey = dbutils.secrets.get(scope="key-vault-scope", key="weatherapikey")
    # Get data from API
    Current_weather  = get_current_weather(base_url, weatherapikey, location)
    Forecast_weather = get_forecats_weather(base_url, weatherapikey, location,3)
    Alerts = get_alerts(base_url,weatherapikey, location)
    # Flatten and merge data
    Merged_data = flatten_data(current_weather, forecast_weather, alerts)
    #print("weather Data:", json.dumps(merged_data, indent=3))
```

```
        #sending the weather data to event HUB
        return merged_data


# function to process each batch of streaming data
last_sent_time = datetime.now()- timedelta(seconds=30) #initialize last sent time
#main program
def process_batch(batch_df, batch_id):
        global last_sent_time
        try:
                #Get current time
                current_time= datetime.now()
                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 sending Sent at {last_sent_time}')
        except Exception as e:
                print (f"Error sending events in batch {batch_id}: {str(e)}")
                raise e
#Set up a streaming source (for example, rate source for testing purposes)
Streaming_df  =spark.readStream.foramt("rate").option("rowsperSecon",1).load()
#Write the streaming data using foreachBatch to send weather data to Event HUB
Query = streamin_df.writeStream.foreachBatch(process_batch).start()
```

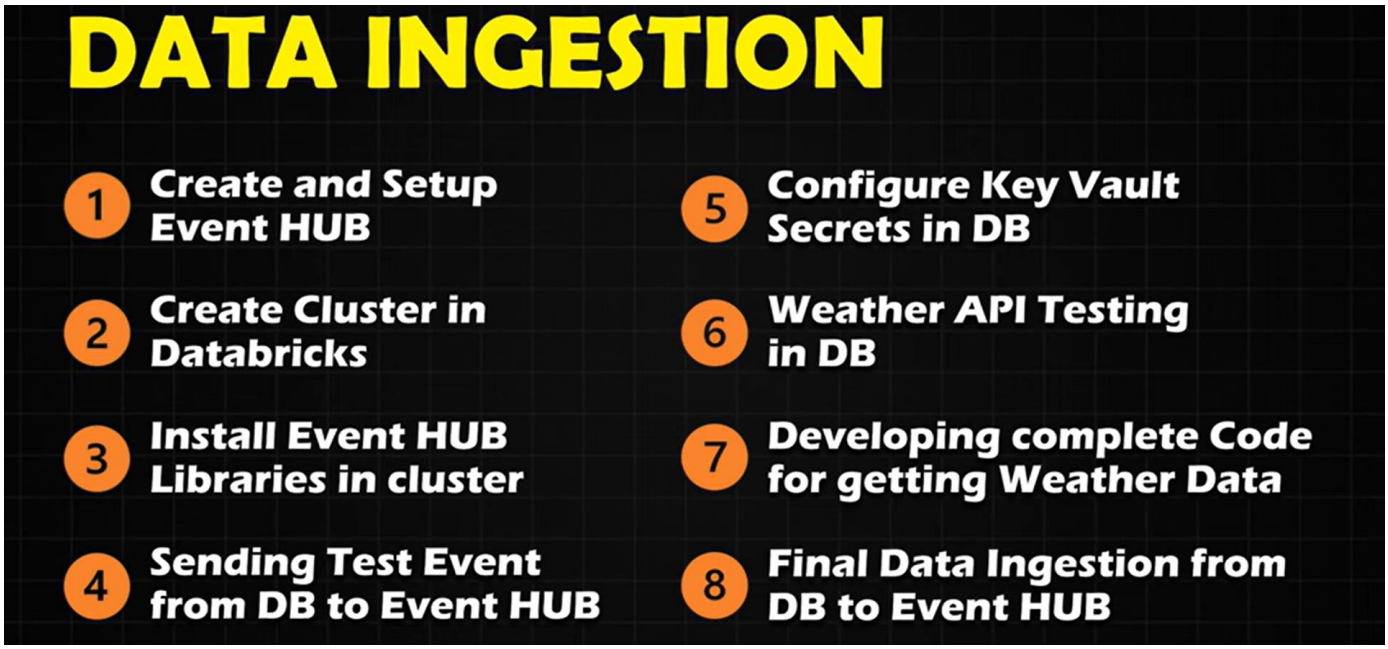
Query.awaitTermination()

#Close the producer after termination

Producer.close()

Every 30 minutes the events received and send it to events to events HUB.

Here using Azure Databricks to ingest weather data from an API, demonstrating both a test script and a complete streaming solution.



Highlights:

🌐 API Exploration: Overview of the weather API and its documentation.

🔑 Authentication: Using an API key for secure access to weather data.

📊 Data Retrieval: Techniques for fetching current weather, forecasts, and alerts.

💻 Script Development: Step-by-step creation of a test script in Azure Databricks.

🔄 Streaming Implementation: Transitioning from a static to a streaming data ingestion method.

⏰ Event Timing: Efficiently sending weather data every 30 seconds.

🚀 Final Integration: Combining multiple functions into a cohesive streaming solution.

Key Insights:

📖 Understanding API Documentation: The importance of familiarizing oneself with API documentation to effectively utilize its features, including authentication and required parameters.

🔒 Security in API Calls: Implementing API keys as a security measure is crucial for protecting sensitive data and ensuring authorized access.

⚙️ Function Modularity: Organizing code into functions enhances readability and reusability, making it easier to maintain and scale the application.

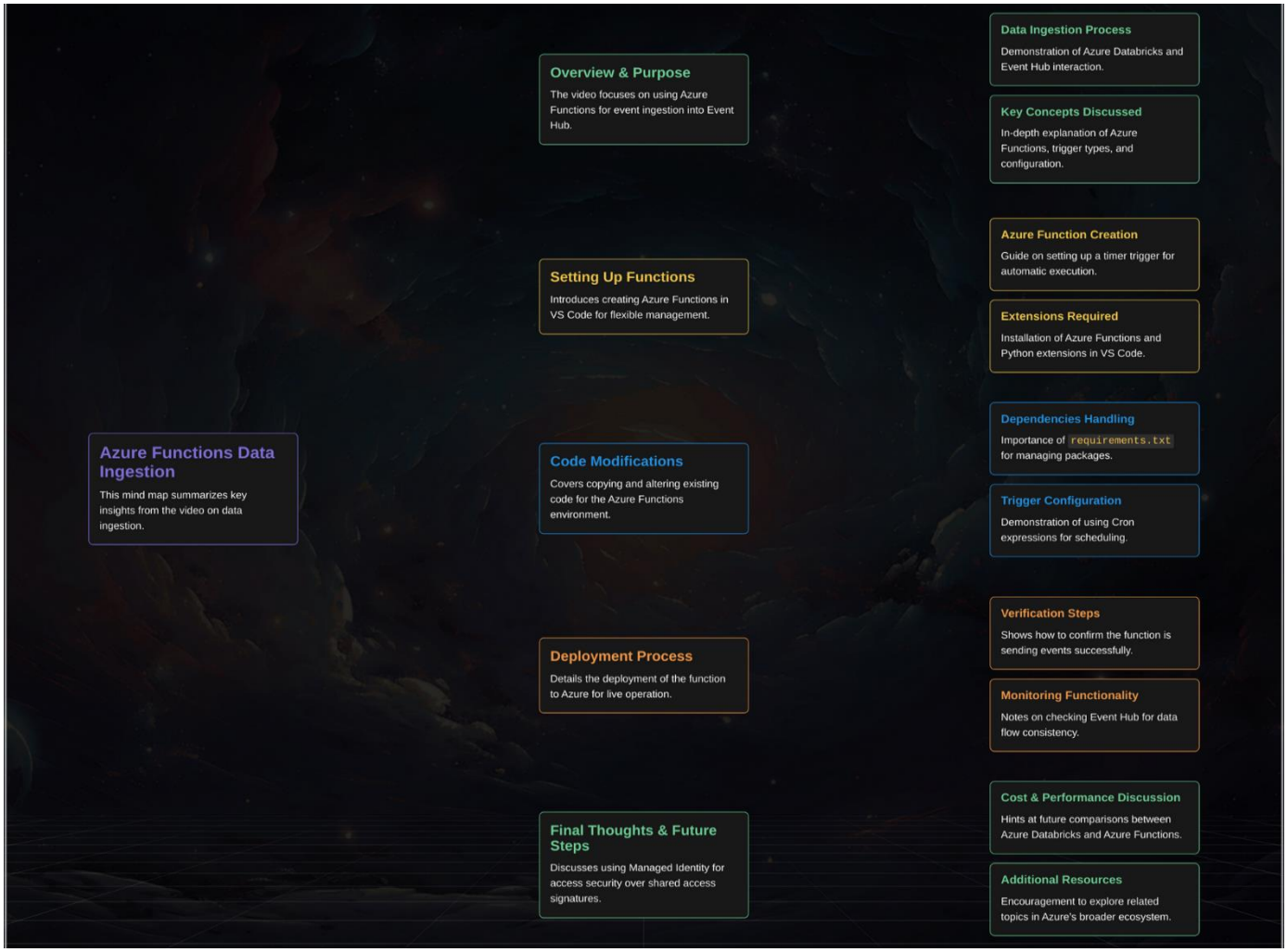
📈 Structured Streaming Benefits: Leveraging Spark's structured streaming allows for real-time data processing, making it suitable for scenarios where timely data is critical.

🗓️ Optimizing Data Frequency: Adjusting the data retrieval frequency to every 30 seconds strikes a balance between timely updates and resource efficiency, avoiding unnecessary API calls.

📊 Data Flattening Techniques: Effective data flattening helps in extracting the necessary information from complex JSON responses, which is vital for analysis and reporting.

🔗 Integration of Services: Combining Azure Databricks and Event Hub illustrates a powerful method for real-time analytics, showcasing how different Azure services can work together seamlessly.

Part- 5 (Data Ingestion using Azure Functions)- End to End Streaming Azure Data Engineering Project



Here explored using Azure Functions for data ingestion, replicating the process done with Azure Databricks, and setting up necessary configurations.

Highlights

- 🚀 Created a function app in Azure for data ingestion.

- 🖥️ Utilized Visual Studio Code for function development and deployment.

- 🔁 Established a timer trigger to send data every 30 seconds.

- 🔑 Implemented managed identity for secure access to Azure resources.

- 📦 Configured requirements.txt to manage external libraries.

- 📊 Successfully deployed the function and verified data in Azure Event Hub.

- 💡 Discussed cost and performance considerations for data ingestion methods.

Key Insights

- 🛠️ **Function App Flexibility**: Using Azure Functions allows for greater flexibility in coding and integration compared to Databricks, suitable for various data ingestion scenarios.

- 🔁 **Timer Triggers**: Timer triggers in Azure Functions automate data streaming tasks, ensuring timely and efficient data handling without manual intervention.

- 🔒 **Managed Identity Security**: Utilizing managed identity enhances security by eliminating the need for connection strings or secrets, simplifying access management.

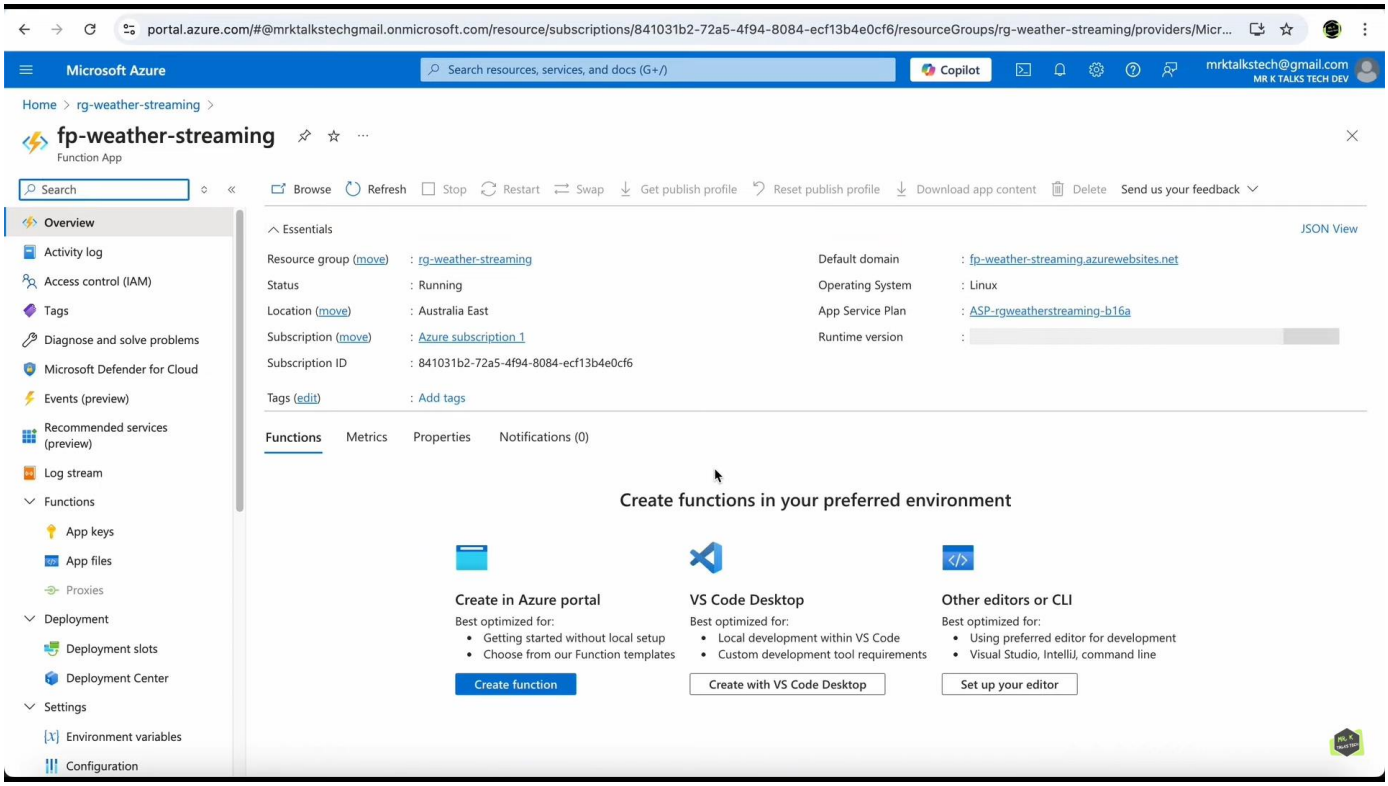
- 🌐 **VS Code Integration**: Visual Studio Code offers a robust environment for Azure development, providing tools that streamline coding, testing, and deployment processes.

- 📜 **Code Reusability**: The ability to reuse existing code from Databricks in Azure Functions reduces development time and ensures consistency in functionality.

- 📈 **Real-time Data Handling**: The successful deployment showed that Azure Functions could continuously stream data in real-time, showcasing their effectiveness for event-driven architectures.

- 💰 **Cost-Performance Analysis**: Evaluating the cost and performance of Azure Functions versus Databricks informs optimal resource allocation for data engineering projects.

So far data ingestion perform using Databricks and will use Azure Functions.



There are three ways to create function app.

1. Download visual studio code
2. Create in Azure Portal Tenant.
3. Other editors or CLI

Open VS code--->goto extensions--->search Azure Function--->and install--->sign in into Azure Portal

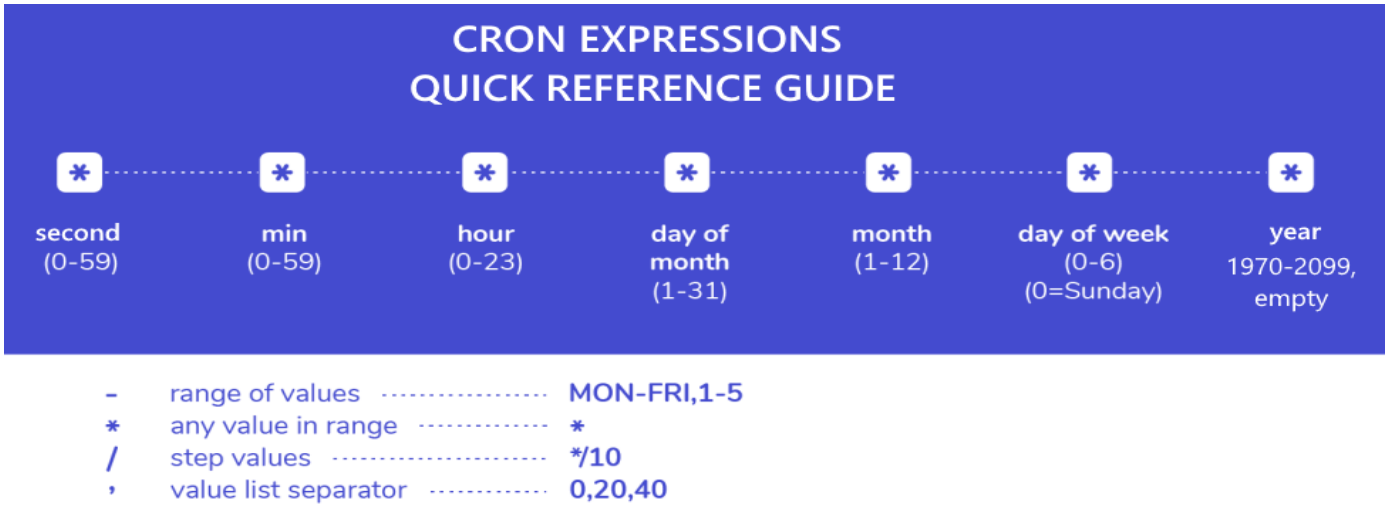
And install Python Extension

Let's move to workspace--->create function project--->now initially all files store in and once complete the function app finally deploy into cloud---> create folder--->weather streaming App.

Configure few things firstly--->python--->model v2--->(Interpreter) skip virtual environment option

After this choosing trigger type--->different timers--->among this choose (Timer Trigger)--->this is right choice for trigger.

After that name of function--->enter a cron expression of the format.



In VS code paste crone expression */30***** and automatically trigger every 30 seconds. Click enter and it will open current window option few files automatically created

.gitignore

.funcignore

Function_app.py

Host.json

Local.settings.json

Requirement.txt

There are two files important

1. Click on Function_app.py

Import logging

Import azure.functions as func

import requests

import json

from azure.eventhub import EventhubProducerClient, EventData

from azure.identity import DefaultAzureCredintial

from azure.keyvault.secrets import SecretClient

app= func.FunctionApp()

@app.timer_trigger(schedule="*/30 * * * * *", arg_name = "myTimer", run_on_startup=False, use_monitor = False)

def weatherfunction(myTimer: func.TimerRequest) -> None:

if myTimer.pass_due:

logging.info(“The timer is past due!”)

logging.info(‘python timer trigger function executed.’)

let’s navigate azure workspace --->Azure Databricks--->Notebook

however, spark functions can’t be used in Azure Functions for this reason we use different code section.

#getting secret value from key vault.

This secret_scope is specific to azure databricks

However this connection string is used to create the producer for the event HUB and now we don’t have it so we have to create this producer a different way for the azure function to work properly so lets see how we can create it so for that I’m going to paste a simple code right below the event hub name ok so

#eventhub_connection_string = dbutils.secrets.get(scope=”key-vault-scope”,key=”eventhub-connection-string”)

Event_hub_name = ‘weatherstreamingeventhub **#event HUB name**

Event_HUB_Namespace= ‘weatherstreamingnamespace.servicebus.windows.net’

#user managed identity of function App

Credential = DefaultAzureCredential{}

#initiaze the event HUB producer

Producer = EventHubProducerClient {

fully_Qualified_namespace= Event_HUB_NAMESPACE,

eventhub_name =EVENT_HUB_NAME,

Credential=credential

}

#initiaze the event HUB producer

#Producer = EventhubProducerClient.from_connection_string(com_str= connection, 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_responce(response):

    if response.status_code==200:

            return response.json()

    else:

            return f"Erro: {response.status_code}, {response.text}"

#function to get current weather and air quality data

def get_current_weather(base_URL, api_key, location):

    current_weather_url  =f"{base_URL}/current.json"

        params = {
                        'key': api_key,
                'q': location,
                "aqi": 'yes',
            }

        response = request.get(base_URL, params=params)

        return handle_response(response)

#function to get forecast Data

def get_forecast_weather(base_URL, api_key, location, days):

    forecast_url = f"{base_url}/forecast.json"

            params = {
                        'key': api_key,
                'q': location,
                "days":  days,
            }

        response = request.get(forecast_URL, params=params)

        return handle_response(response)

#function to get alerts

def get_alerts(base_URL, api_key, location):
```

```
    forecast_url = f"{base_url}/alerts.json"

            params = {
                    'key': api_key,
                'q': location,
                "alerts":  'yes',
            }

        response = request.get(alert_URL, params=params)

        return handle_response(response)
```

 # Flatten and merge the data

```
def flatten_data(current_weather, forecast_weathe, 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 = alert.get("alerts", {}). get("alerts", {})

    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('lot'),

                    'location': location_data.get('location'),

                    '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),
```

```
'is_day': current.get('is_day'),
'presurre_in': current.get('presurre_in'),
'precip_in': current.get('precip_in'),
'humidity': current.get('humidity'),
'cloud': current.get('cloud'),
'feelslike_c': current.get('feelslike_c'),
'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)
        },
        'alerts': [
        {
                'headline': alert.get('headline'),
                'severity': alert.get('severity'),
                'description': alert.get('description'),
                'instructions': alert.get('instructions')
        }
        for alert in alert_list
],
'forecast': [
        {
                'date': day.get('date'),
```

```
                'maxtemp_c': day.get('day',{}).get('maxtemp_c'),
                'mintemp_c': day.get('day',{}).get('mintemp_c'),
                'condition: day.get('day',{}).get('condition', {}.get('text')
            }
            for day in forecast
        ]
        return flattend_data
    #fetch secret from key vault
        def get_secret_from_keyvault(vault_url, secret_name):
            credential = DefaultAzureCredential()
            secret_client = SecretClient(vault_url=vault_url, credential= credential)
            retrieved_secret = secret_client.get_secret(secret_name)
            return retrieved_secret.value
#main program
def fetch_weather_data():
    base_url = http://api.weatherapi.com/v1/"
    location = "hyd" # you can replace with any city name based on your preference
    #fetch the API key from key Vault
    VAULT_URL = https://kv-weather-streaming.vault.azure.net/
    API_KEY_SECRET_NAME = "weatherapikey"
    Weatherapikey = get_secret_from_keyvault(VAULT_URL, API_KEY_SECRET_NAME)

    weatherapikey = dbutils.secrets.get(scope="key-vault-scope", key="weatherapikey")
    # Get data from API
    Current_weather  = get_current_weather(base_url, weatherapikey, location)
    Forecast_weather = get_forecats_weather(base_url, weatherapikey, location,3)
    Alerts = get_alerts(base_url,weatherapikey, location)
```

```
        # Flatten and merge data

        Merged_data = flatten_data(current_weather, forecast_weather, alerts)

        #print("weather Data:", json.dumps(merged_data, indent=3))

        #sending the weather data to event HUB

        send_event(merger_data)

    #calling the main program

    fetch_weather_data()
```

after getting the API data this merged data will be event to the event HUB single event.

First let's go to RG--->eventHUB--->event--->dataexplorer--->view events

Let's see the event in testing.

requeirement.txt this file is used list the all libraries is to create the function_app the purpose is when the code is deploy to cloud these libraries are install automatically

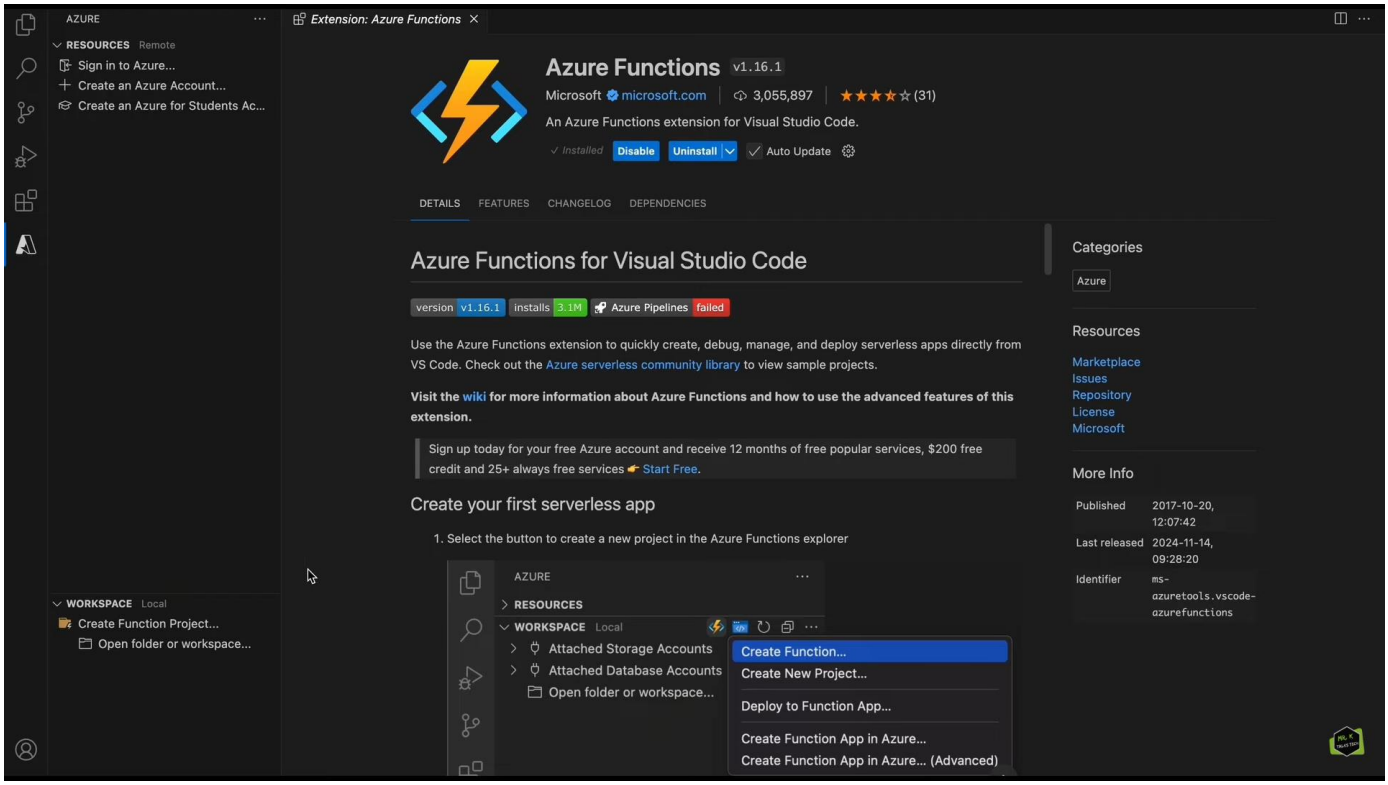
```
#do not include azure-functions-worker in this file

# the python worker is managed by azure function platform

#Manually managing azure -functions-worker may cause unexpected issues

azure-functions

azure.eventhub

azure.identity

azure.keyvault.secrets
```

Previously Azure databricks uses a shared access key connection string to send the events to the event hub since a direct access cannot given to the azure databricks so there fore added the connection string as a secret in the key and fetch it by creating a secret in databricks so all of these not needed in the function app. Since we used to provide a direct access to it using the managed identity option so due to this we have updated the code which uses the managed identity to authenticate and create the event hub and this event hub name is also called as host name so if

go to azure portal--->weatherstreamingnamespace--->overview--->hostname

lets goto azure portal--->RG Group--->Key-Vault--->access control(IAM)--->add role assignment--->

key vaults secrets user--->member--->choose--->managed identity(drop down)--->function APP--->review-assign.



Lets go to azure portal--->azure Event_HUB_Namespace--->Event_Hub--->shared access policy--->this policy is assigned send access the reason is using shared access policy currently we cant used.

However azure functions support IAM supported and which more secured and recommended.

IAM--->add--->azure event hub data explorer--->access to managed identity--->chose write subscription--->managed identity--->we can dropdown--->use cannot see options any azure function-app.

Lets go RG--->Azure Function app--->managed identity by default not enabled--->to enable this--->lets loot at the left side of the Azure Function app--->Identity--->system assigned--->enable--->save--->unique id.(managed) principal.

Lets go to --->refresh--->manged identity--->subscription--->managed identity(drop down--->function app)--->select--->review-create.

Now azure function is directed to access event-HUB

For testing the functionality in cloud

Lets go to azure function app

And goto vs code and inside azure icon workspace --->deploy.

After in the top find fp-function app--->

Before event hub see the event status--->weatherstreaminnamespace--->event hub--->data explorer-->view events.

Continue deployment stage--->it is active stage--->deploy pop window --->click--->successfully done.

Lets goto function app--->refresh--->weatherapifunction you see.

In that you App files--->u see all files deployed.

Open actual function app.

Lets go to event HUB--->view events

Click stop function app. Straight away not sending to events.

--->the vs code let's click on this assure icon and here in the workspace section you can see a button called deploy to a show so let's click on it and after doing that in the top you can find our function app name which is FP weather streaming which is in the resource Group RG weather streaming so this is the function app where we need to deploy this assure function that we created so as discussed earlier the reason why we are able to view this function app is because of this resources section by signing into the ASO tenant okay now we can choose our function app to start with the deployment but before that I would like to go to the even Hub and see the even status in order to properly test this function app so for that I'll jump back to the assure portal and go to the resource Group and from the resource Group let's open our even Hub name space and inside the even Hub let's go to the data Explorer Tab and click on.

The View events button okay as you can see here currently I do not have any events in the even Hub since the last event was sent more than a day ago okay now let's go back to the visual studio code and continue our deployment so after deploying this the function would be in the active stage and the function will be triggered every 30 seconds which would send the latest weather data to the even Hub so for that let's choose the FP weather streaming function app after choosing it there would be a warning message asking us to confirm that upon making this deployment if there are any previous changes in the function app it would be overwritten by this new changes we do not have anything created at so let's click on the deploy button cool as you can see here the deployment is in progress and the function is currently deployed to the FP weather streaming function app nice as you can see here we have successfully deployed our function app changes

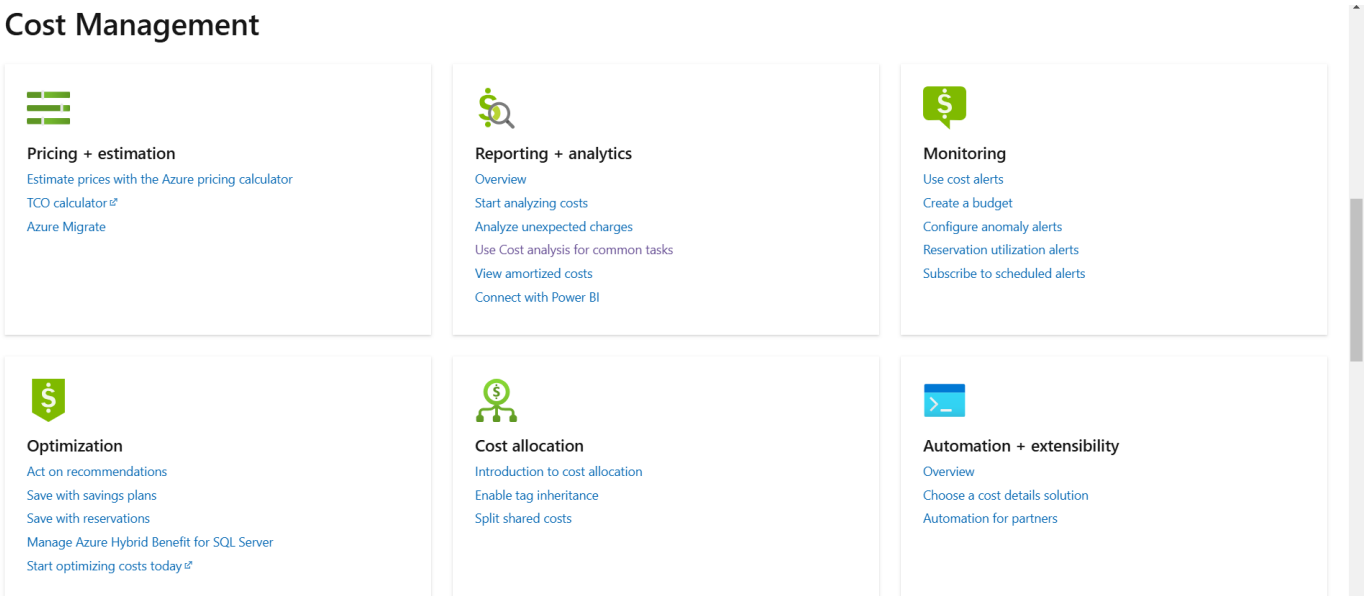
Now let's go back to the assure portal and firstly let's open our function app to view the deployment changes so if you click on the refresh button you can find a new function created called weather APP function which means that we have successfully deployed our local code to Cloud here you can see the trigger type which is timer trigger and the current status is active and one other important thing to note here is in the left side there will be an option called app files let's click on it and here you'll be able to see all the code files that is related to the Azure function that we deployed as you can see here these are the files and most importantly the function app.py file which contains all the code Logic for the data injection and then we have the requirements.txt file which contains the list of all packages and libraries that we used in our code so as said earlier when our function app runs in Cloud it would first install all these packages in the cloud environment and only then it would execute the function that is written in the function app.py file so that's the main reason why we added all the packages in this requirements.txt file I hope all that makes sense now okay now let's go back and open our actual function here you'll be able to see the code that we have written in the function app.py file and also we have multiple options here such as doing a test turn of the function and other monitoring features but we'll not be able to use any of the monitoring features since we did not enable the application

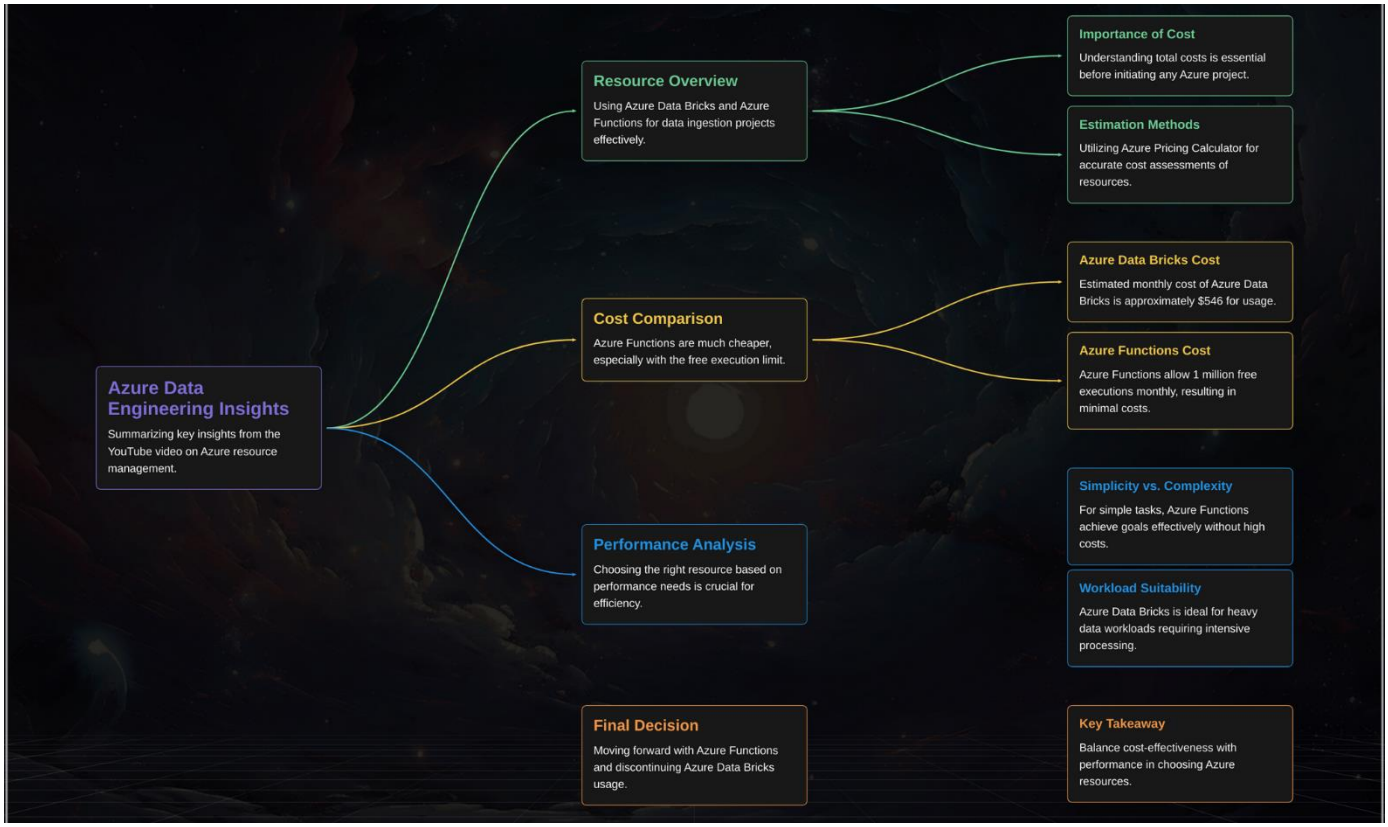
insights when creating the function app so therefore if something goes wrong it would be harder to test but we can always enable it later if needed but since we have tested this code completely in azure data bricks I'm pretty sure our code should work fine without any issues so for testing this let's go to

the even Hub and see if we have received any events or not so let's click on The View events button nice as you can see here we have received multiple events and if you notice the time the function app is sending the events approximately for every 30 seconds as configured in the timer trigger which is really cool and if you scroll to the right and view the even body you can see all the latest weather information for the Chennai location this means that our function app is running fine without any issues and will be keep sending events for every 30 seconds

So far we have received four events so if I click on The View events button again we should have received another one by now nice as you can see here we have five events now and have received the latest weather information with it so basically as far as the function app is running it would be continuously sending the weather data as the even to the even hub for every 30 seconds so if you want the function app to stop sending the events you can click on the stop button over here which would stop the function app straight away and will not be sending the events to the even Hub anymore but I'm going to keep this running to continue with the next steps in the architecture okay with this we have successfully completed the data injection process using function app and I hope now you have a clear understanding of how to create a function app from scratch and how to set up the managed identity and other requirement access configuration for the function app and how to read the secrets from the keyword using function app and also finally how we can use function app to send an event to the even Hub okay so now we have seen two types of data injection one with assure data break and another one with assure function app in the next session we are going to discuss which option would be the preferred one for this Inn project in terms of both cost and performance.

Part- 6 (Cost Analysis and Architectural decisions)- End to End Streaming Data Engineering Project





Azure cost calculator:

Analyze Azure Data Bricks and Azure Functions for data ingestion, focusing on cost and performance to determine the best resource for our project.

💰 Cost Analysis: Understanding total Azure costs is crucial for effective project management.

🚀 Performance Evaluation: Azure Functions offer better performance for simple tasks compared to Azure Data Bricks.

📊 Pricing Calculator: The Azure Pricing Calculator is a vital tool for estimating resource costs.

🆓 Free Executions: Azure Functions provide 1 million free executions per month, making it cost-effective.

⚙️ Architectural Decisions: Choosing the right resource is essential for minimizing costs while achieving optimal performance.

🔄 Simplified Workflows: Sometimes, simpler tools like Azure Functions suffice for data ingestion tasks.

🛠️ Cost Management: Monitoring costs regularly helps in optimizing resource usage and maintaining budgets.

Key Insights

📉 Cost Efficiency: Understanding Azure pricing structures enables data engineers to make informed decisions to minimize project costs. Azure Functions are significantly cheaper for low-complexity tasks.

⚡ Performance Consideration: For projects requiring rapid API calls without heavy data processing, Azure Functions outperform Azure Data Bricks due to their serverless nature and instant availability.

💡 Resource Selection: It's essential to evaluate project requirements and choose the most cost-effective resources from the outset, avoiding unnecessary complexity and expenses.

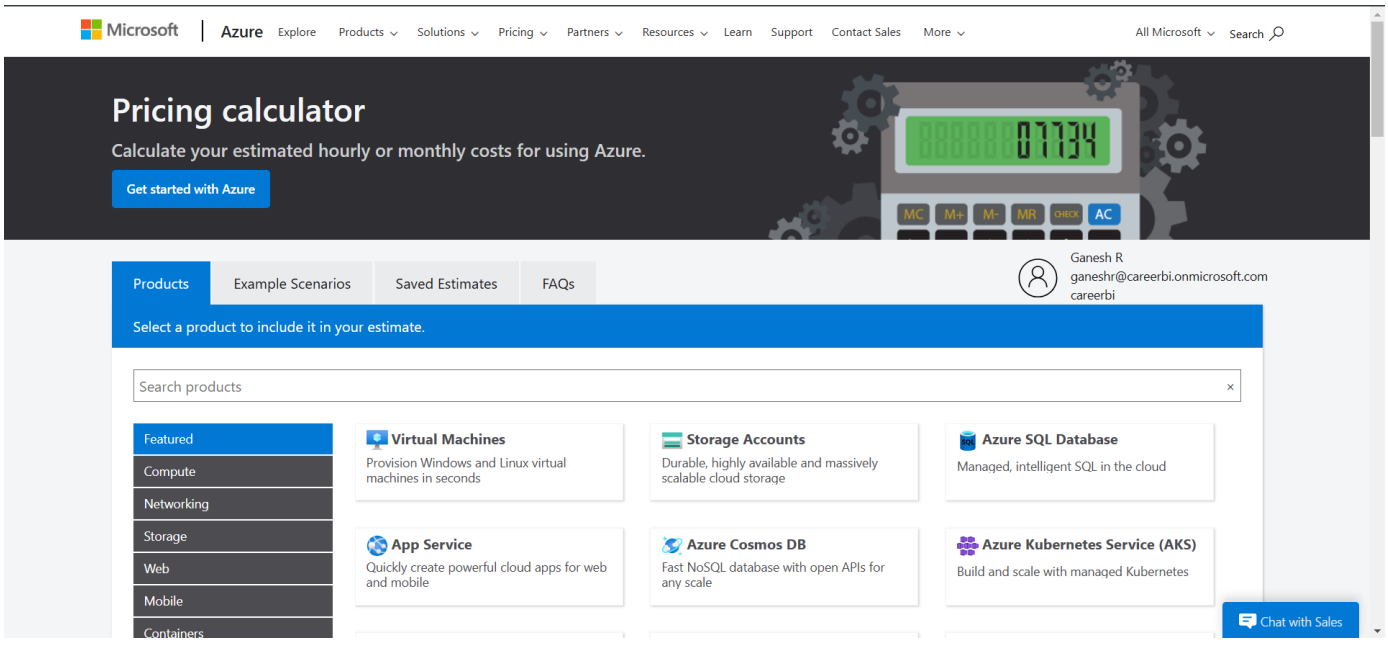
📈 Future Scalability: While Azure Functions are ideal for simple tasks, Data Bricks should be considered for larger, more complex data workloads, ensuring scalability for future project growth.

🔍 Regular Monitoring: Utilizing Azure's cost analysis tools allows teams to track resource usage and adjust their strategies based on real-time spending data.

🏷️ Simplifying Architecture: Emphasizing simpler architectures can lead to significant cost savings and operational efficiency in data ingestion processes.

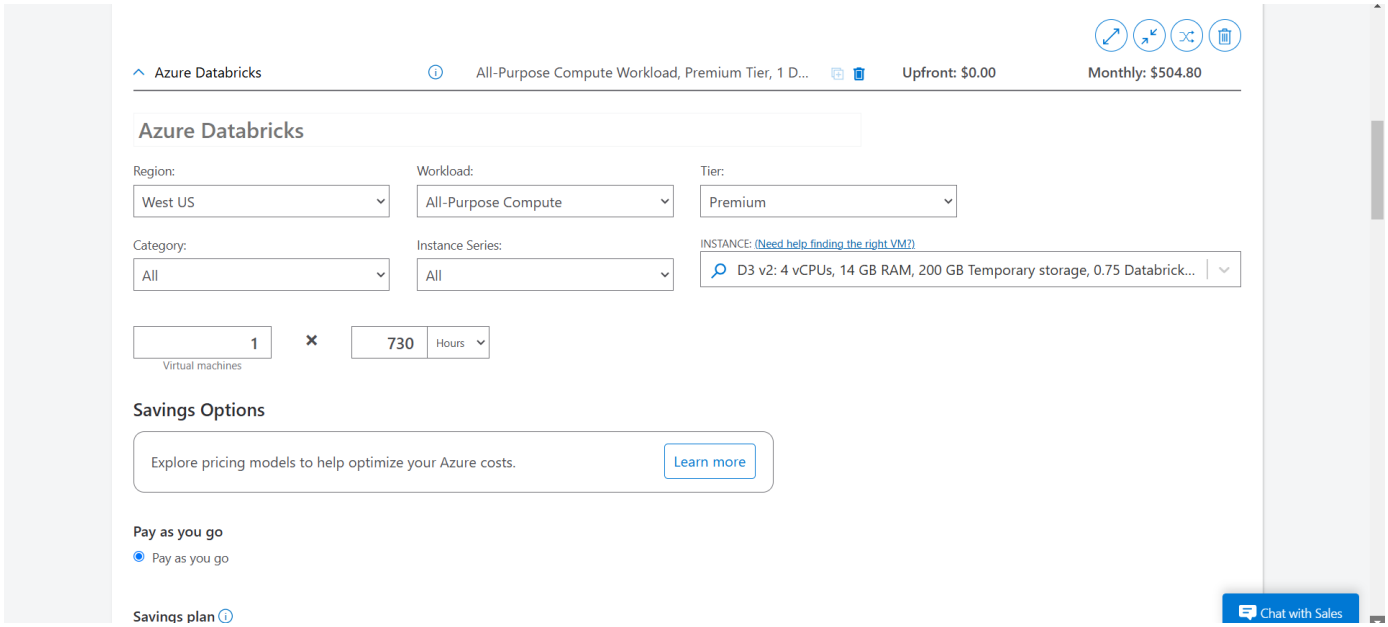
🔄 Continuous Learning: Mastering tools like the Azure Pricing Calculator aids data engineers in making better architectural decisions throughout the project lifecycle.

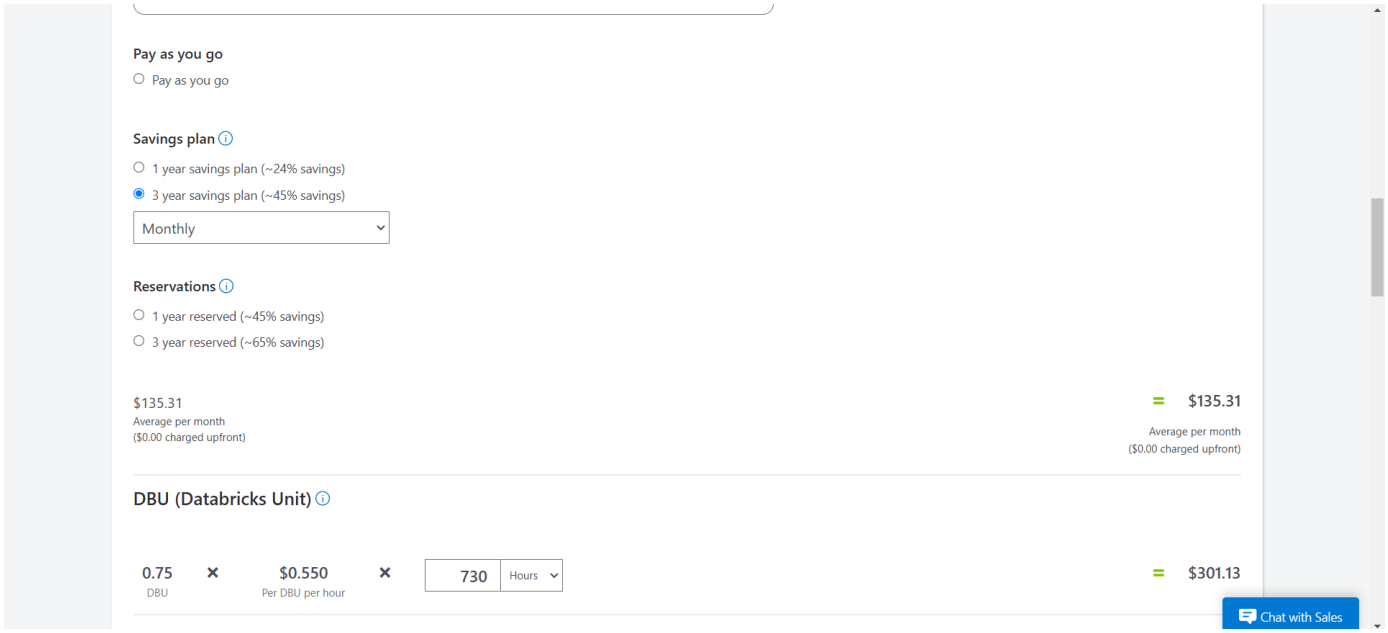
https://azure.microsoft.com/en-in/pricing/calculator/?ef_id=_k_Cj0KCQiA1Km7BhC9ARIsAFZfEIu4dmQNxPKLPDRJ1d_si71MywMyDov-oPdkIEz5ODNfuDvXtzT7Sj4aArQnEALw_wcB_k_&OCID=AIDcmmf1elj9v5_SEM__k_Cj0KCQiA1Km7BhC9ARIsAFZfEIu4dmQNxPKLPDRJ1d_si71MywMyDov-oPdkIEz5ODNfuDvXtzT7Sj4aArQnEALw_wcB_k_&gad_source=1&gclid=Cj0KCQiA1Km7BhC9ARIsAFZfEIu4dmQNxPKLPDRJ1d_si71MywMyDov-oPdkIEz5ODNfuDvXtzT7Sj4aArQnEALw_wcB

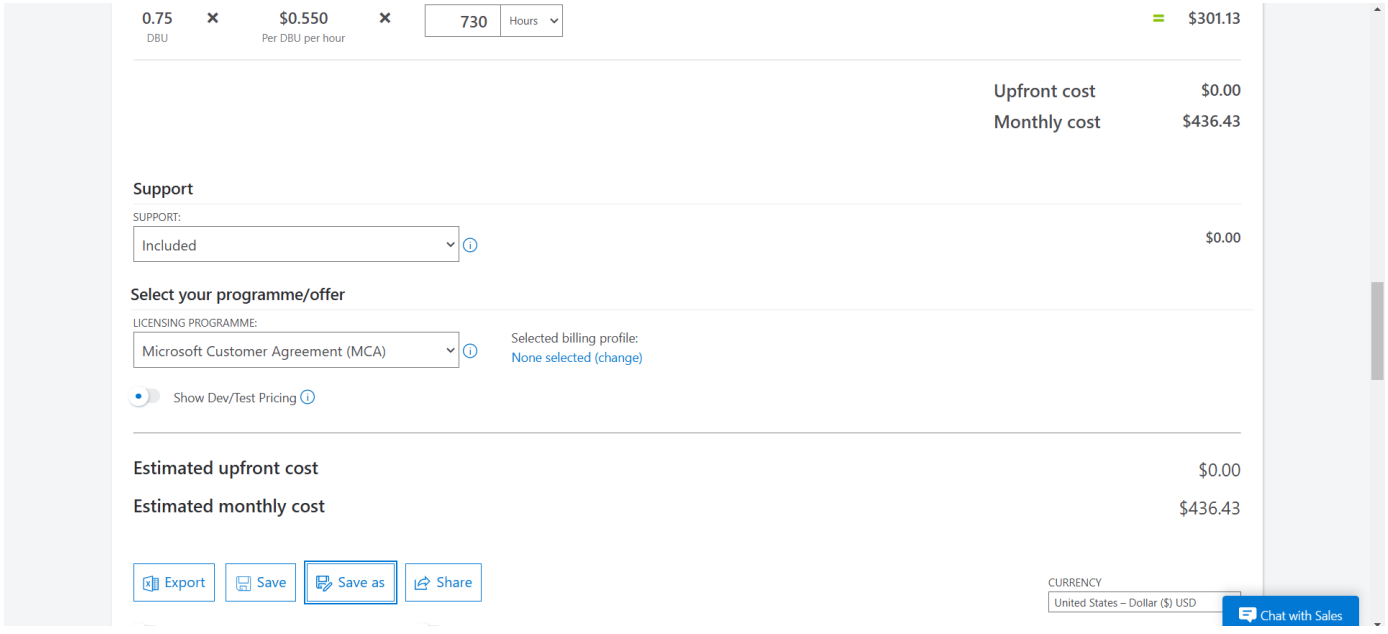


Firstly, Azure Databricks:

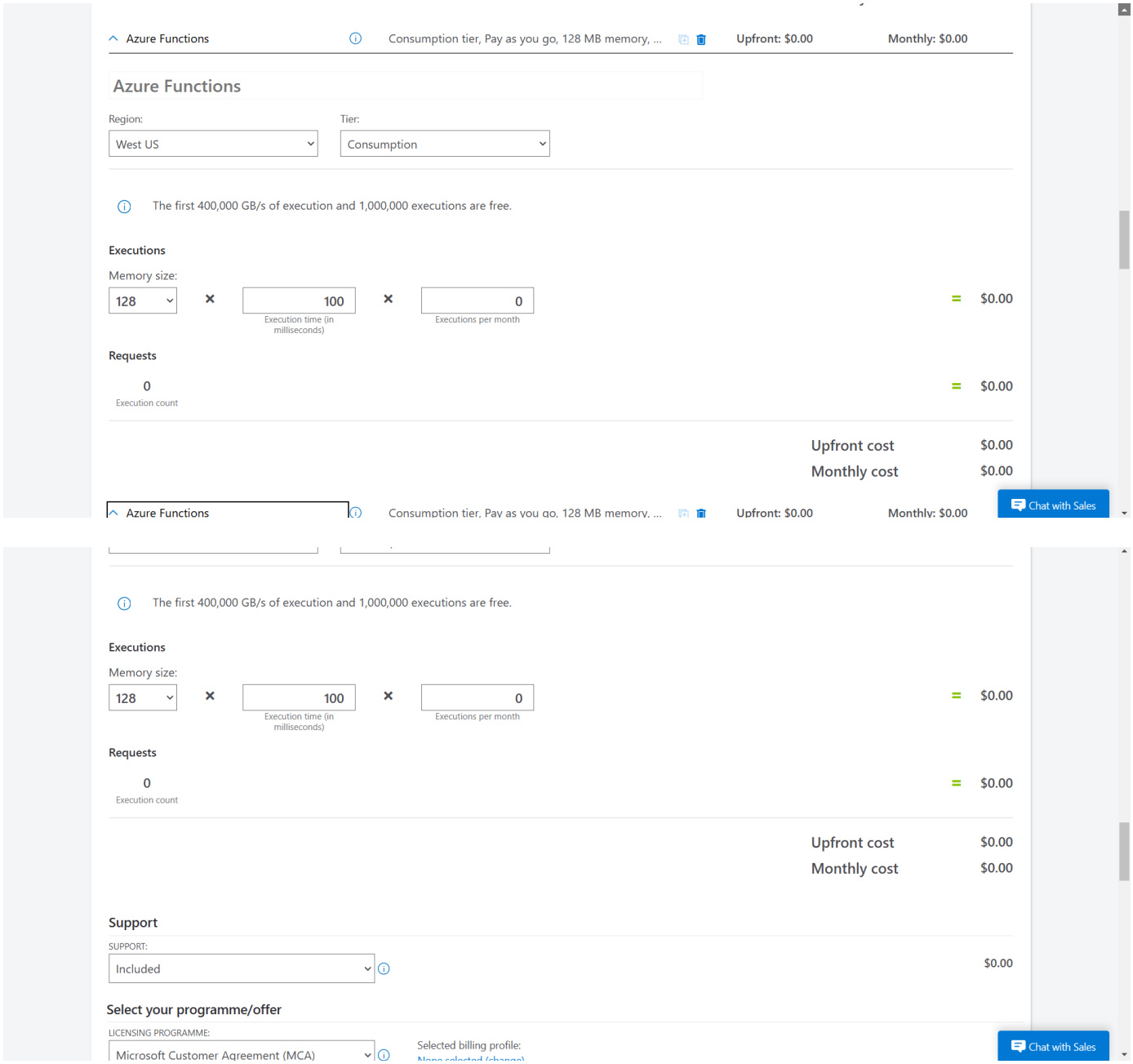
Search in





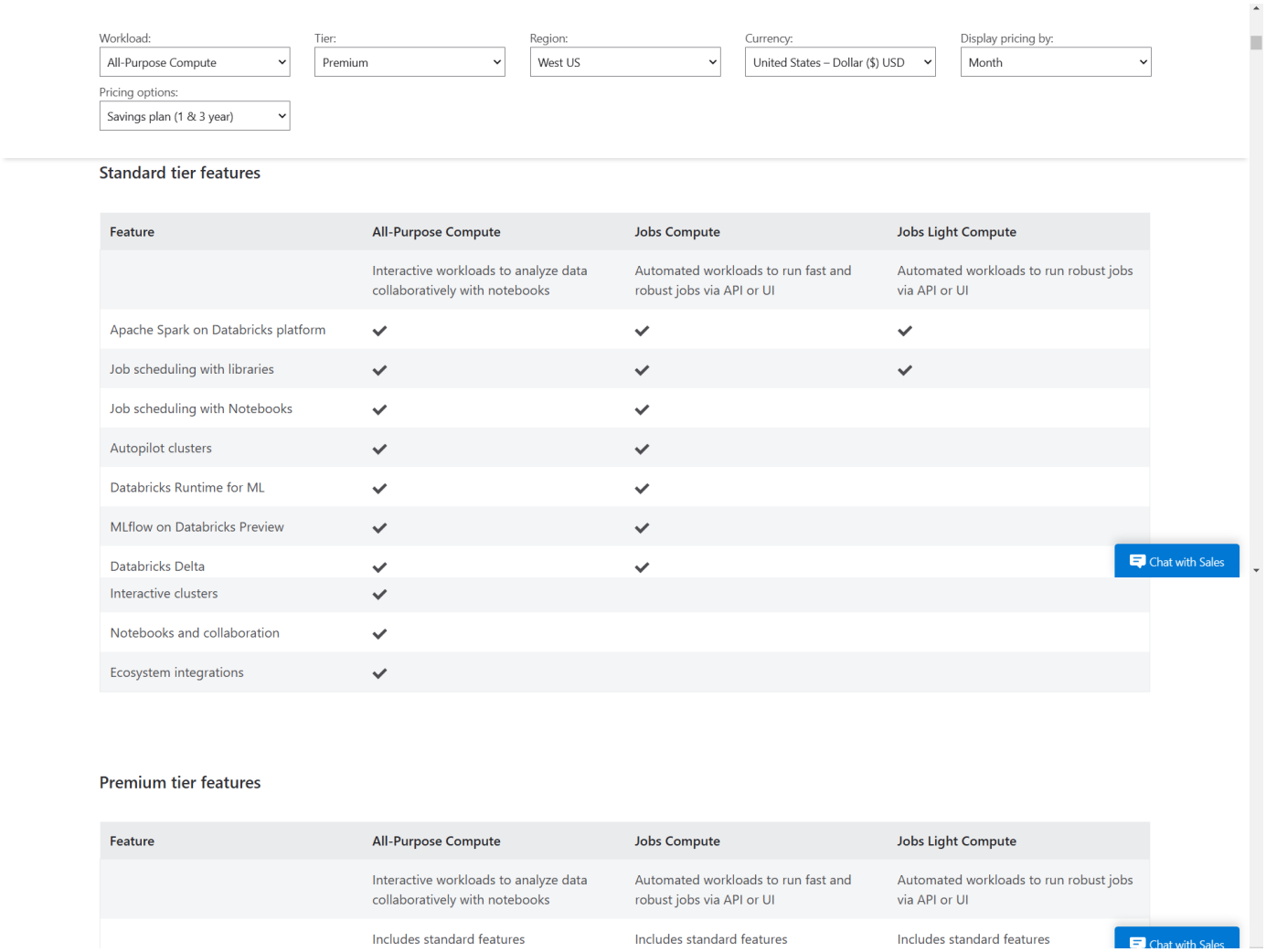


Next is Azure Function



Now clear winner is Azure function.

data brakes pricing so when you compare both assure data braks and assure functions on basing of pricing the clear winner is assure functions now let's discuss performance for our use case we are performing a very simple task we are calling an API and fetching the data and then sending it to the even hub here we are not processing any big data workloads such as aggregating millions of data points or performing complex transformation and Analysis we are not doing any of these things and we are just hitting AP calls and sending the data so therefore the performance is calculated based on how quickly we can execute the APA call and how readily our Compu resources are available for these things the best is assure function app so why assure function app the is reason for this is it is a serverless feature we do not maintain any infrastructure from our side everything is maintained by Microsoft azure we are just executing it using their infrastructure the advantage of using this is the assure function app compute is always available as it is completely managed by assure infrastructure in this project we have set a timer trigger for it to run every 30 seconds during this time.



Workload: All-Purpose Compute | Tier: Premium | Region: West US | Currency: United States – Dollar ($) USD | Display pricing by: Month

Pricing options: Savings plan (1 & 3 year)

Standard tier features

Feature	All-Purpose Compute	Jobs Compute	Jobs Light Compute
	Interactive workloads to analyze data collaboratively with notebooks	Automated workloads to run fast and robust jobs via API or UI	Automated workloads to run robust jobs via API or UI
Apache Spark on Databricks platform	✓	✓	✓
Job scheduling with libraries	✓	✓	✓
Job scheduling with Notebooks	✓	✓	
Autopilot clusters	✓	✓	
Databricks Runtime for ML	✓	✓	
MLflow on Databricks Preview	✓	✓	
Databricks Delta	✓	✓	
Interactive clusters	✓		
Notebooks and collaboration	✓		
Ecosystem integrations	✓		

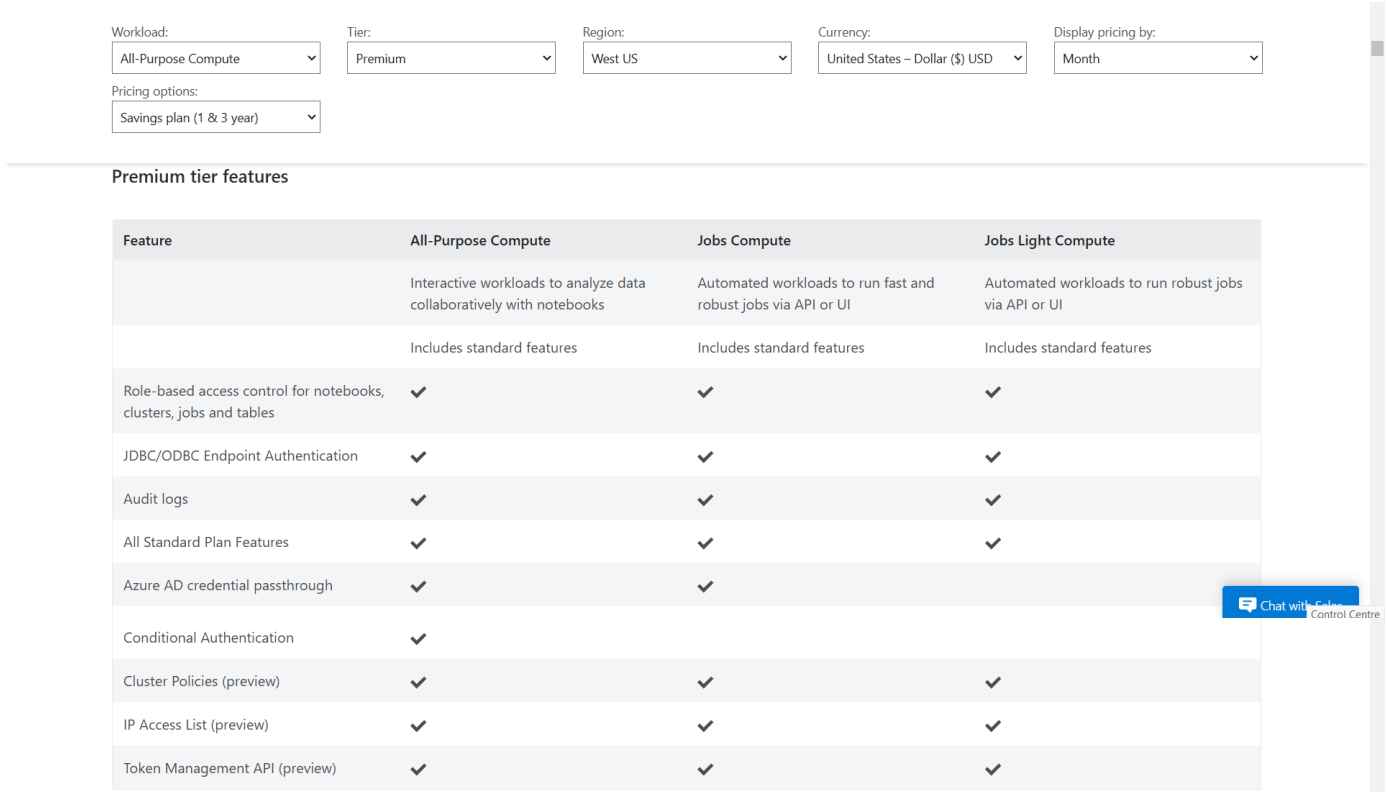
Chat with Sales

Premium tier features

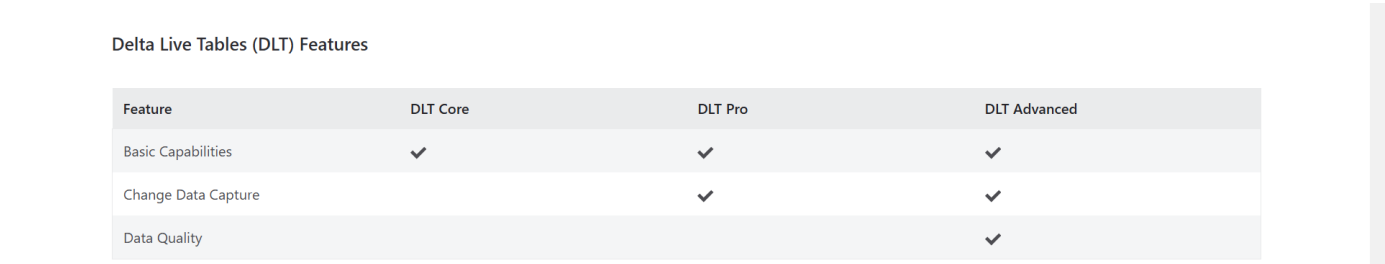
Feature	All-Purpose Compute	Jobs Compute	Jobs Light Compute
	Interactive workloads to analyze data collaboratively with notebooks	Automated workloads to run fast and robust jobs via API or UI	Automated workloads to run robust jobs via API or UI
	Includes standard features	Includes standard features	Includes standard features

Chat with Sales

we use their compute and send the data this will be so quick compared to azure data Brakes in azure data brakes we create and maintain the compute infrastructure if there are any issues or failures we need to fix them and manually enable it again given that our project involves minimal task azure function app is a better choice so you might then ask a question then when would you go for azure data breaks as I said earlier for huge workloads such as when processing massive terabytes of data which involves complex transformation and Analysis assure function app will not be able to handle it and we may need to go for assure data bricks to perform it in such cases we need to accept the cost of $546 which runs 24 hours a day for a month that can handle a massive capacity of data which is quite beneficial to pay for these kind of scenarios so in simpler terms for heavy data transformation workloads as should data bricks is the goto solution similarly when your use case requires Advanced Apache spark futures as data brakes becomes the ideal Choice when we are using normal python code involving a small script execution in a even driven way we should not go for assure data bricks we can directly use assure function and execute that script this is the big difference between them so that's the reason from now on in our project we are going to continue with assure functions to perform the data injection and we will not be using the ASU data bricks anymore so one thing to note here is this is a standard architecture where assure functions send vents to the even Hub so whenever you have a use case to send events to the even Hub you should definitely consider using the assure function as a potential option okay now let me show you one thing quickly let's go back to the resource Group again so here on the left side you can see an option called cost analysis under cost management section let's click on it this will take us to the cost analysis dashboard from here we can analyze the total cost of all the resources in this Resource Group let me give you a quick overview of how much of resources in the resource



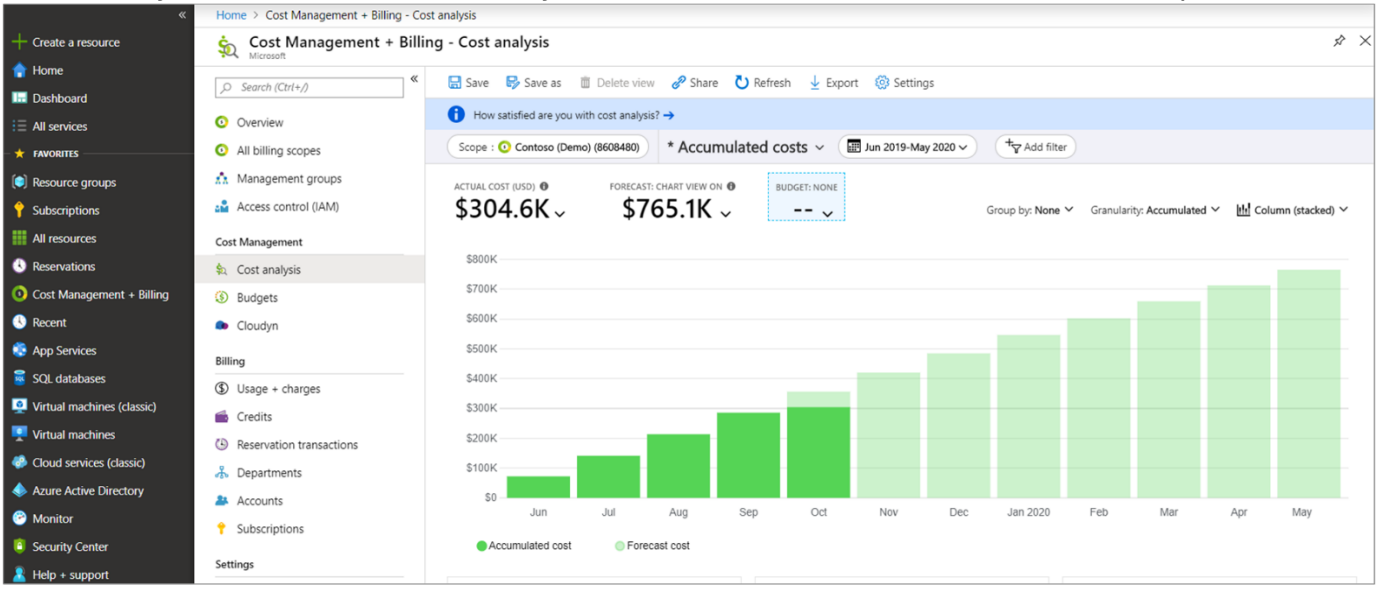
Group are going to cost for a month right now it's showing the cost for December but since the month has not ended yet let's switch to the previous month to get the full breakdown as you can see here the total cost for last month was $1 19 New Zealand door this amount reflects a scenario where all the resources were fully active and running so let's break this down the highest cost is associated with even Hub which is 8 New Zealand doll this is because even Hub requires dedicated Computer Resources and you start paying for it right from the moment it is created next we have the azure data brakes which cost us around $1 New Zealand dollar one thing to note here is EDI didn't keep it running all the time,



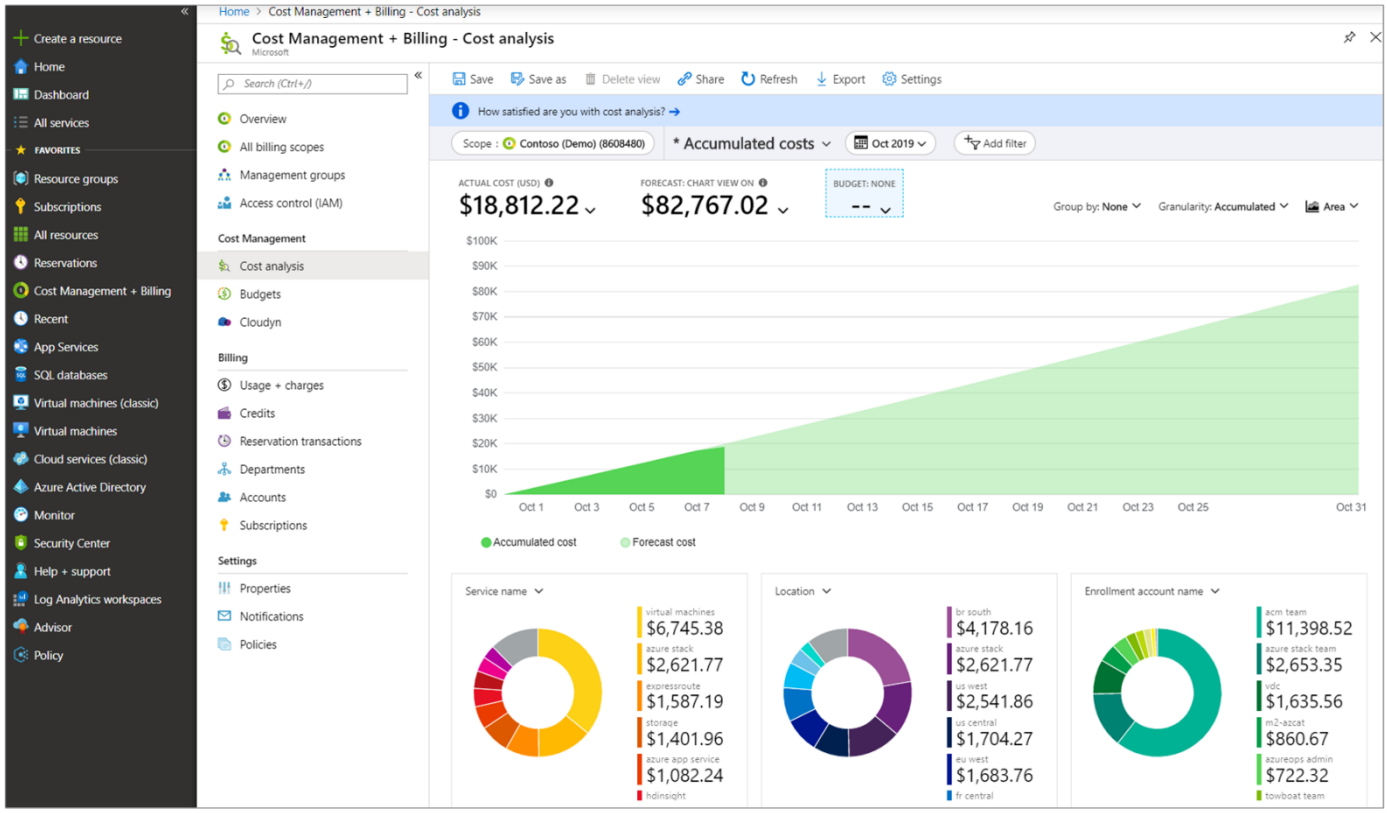
- I turned off the cluster when not in use and I ran it only during the demo sessions this helped me in keeping the cost low now coming back to the assure functions it cost $0 even though the function was fully available
- And we used it for data injection during the last session did not cost any charges it is because assure functions provide 1 million free execution per subscription per month also as discussed earlier even if you exit this limit the cost is just going to be around $0.26 per 1 million execution which is very cheap okay now let's switch to the last 30 days
- View and as you can see here even in this period azure function shows $0 which is really cool okay now let's analyze the daily cost and here the average daily spend is about 60 cents so when you're working on a similar project you can expect the cost to be more or less in the same range okay by now

- I hope you have a clear idea of how to use the azure pricing calculator to estimate cost and decide on resource usage all these discussions usually happen before starting an ASO project as I mentioned earlier as a data engineer it's not just about completing the project but doing it efficiently

is a key thing always a data engineer is involved while making this kind of architectural decision in the early stage of project while working in a real-time project so ne key takeaway for this session is that you don't always need to use complex .



- Tools like azure data bricks for every use case sometimes a simpler tool like assure functions can achieve the same result without incurring unnecessary cost making such decision is an essential skill for any data engineer this is why I wanted to cover this aspect in our project so from now on you should carefully consider your choices of resources in as I start by selecting the most cost effective option and assess if it meets your needs and go for more complex resources only when absolutely necessary and also learn.
- how to use the assure pricing calculator effectively as it is a useful tool which helps us in decision making okay that's all for this session let's say goodbye to assure data bricks for now as we won't be using it further in this project we'll be just using assure functions to perform the data ingestion going forward in our next session we'll focus on how to load and process the data that we ingested using azure functions.



Part- 7 (Event Processing and Loading)- End to End Streaming Azure Data Engineering Project

Function app is sending data every 30 seconds load the data KQL Database.

Open Azure event hub

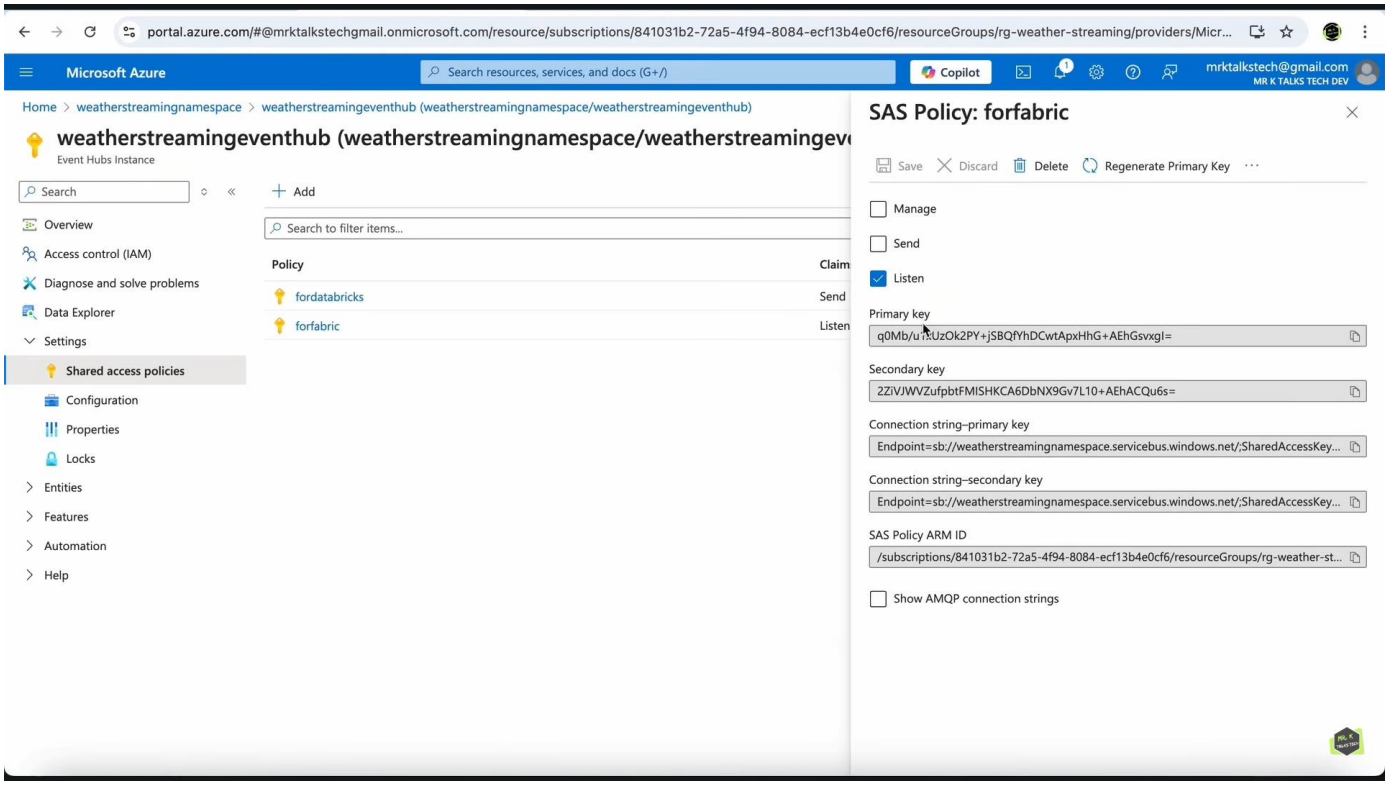
And open power App(Fabric)

Click workspace--->new workspace(create new one new project it is good practice)--->weather streaming.

Source already set up now target.

Open workspace--->new item--->search(eventhouse(weather-eventhouse))--->KQL DB automatically created.

Goto workspace--->weather-streaming--->new item--->search(eventstream)--->addsource(external source)--->azure event hub



connection(new connection)

connection settings

Event Hub Namespace(give the eventhub namespace)

Event Hub(give the event hub name)

Connection credentials

Connection(create new connection)

Connection name( )

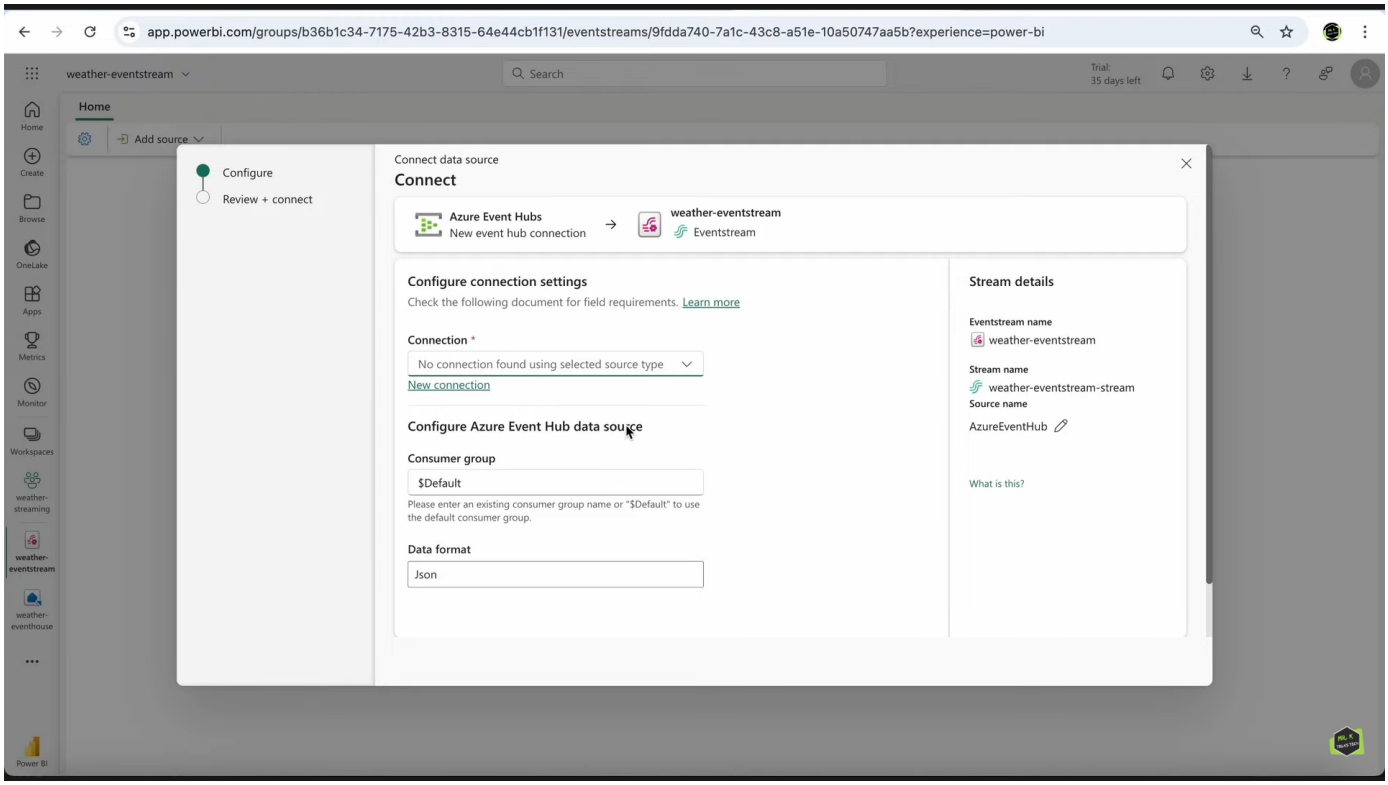
Authentication kind(shared access key)

Shared access key name (forfabric)

Shared access key ( )

Lets connect

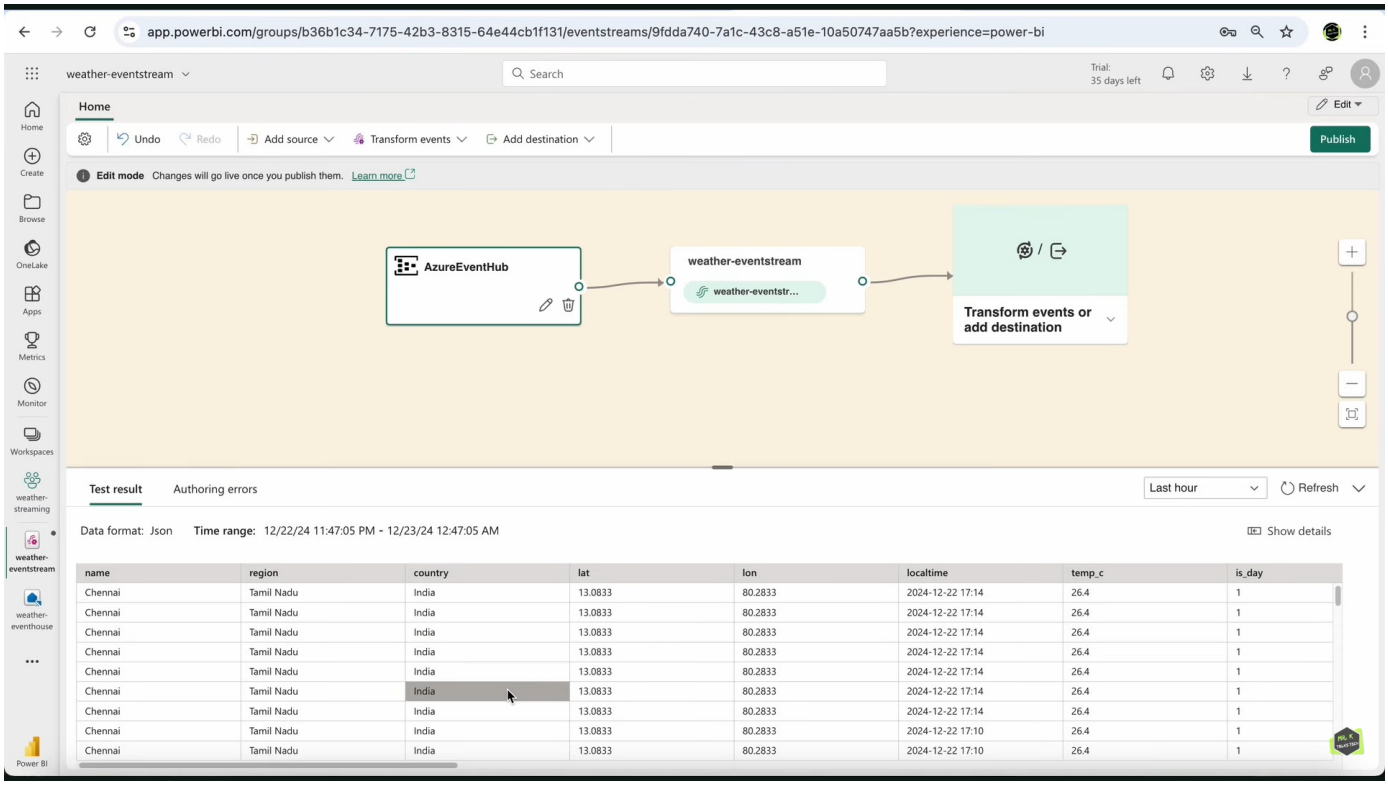
Goto eventhub--->settings--->shared access policies--->add--->policy name(forfabric)--->listen-→click on it--->primary key



We create successfully connection

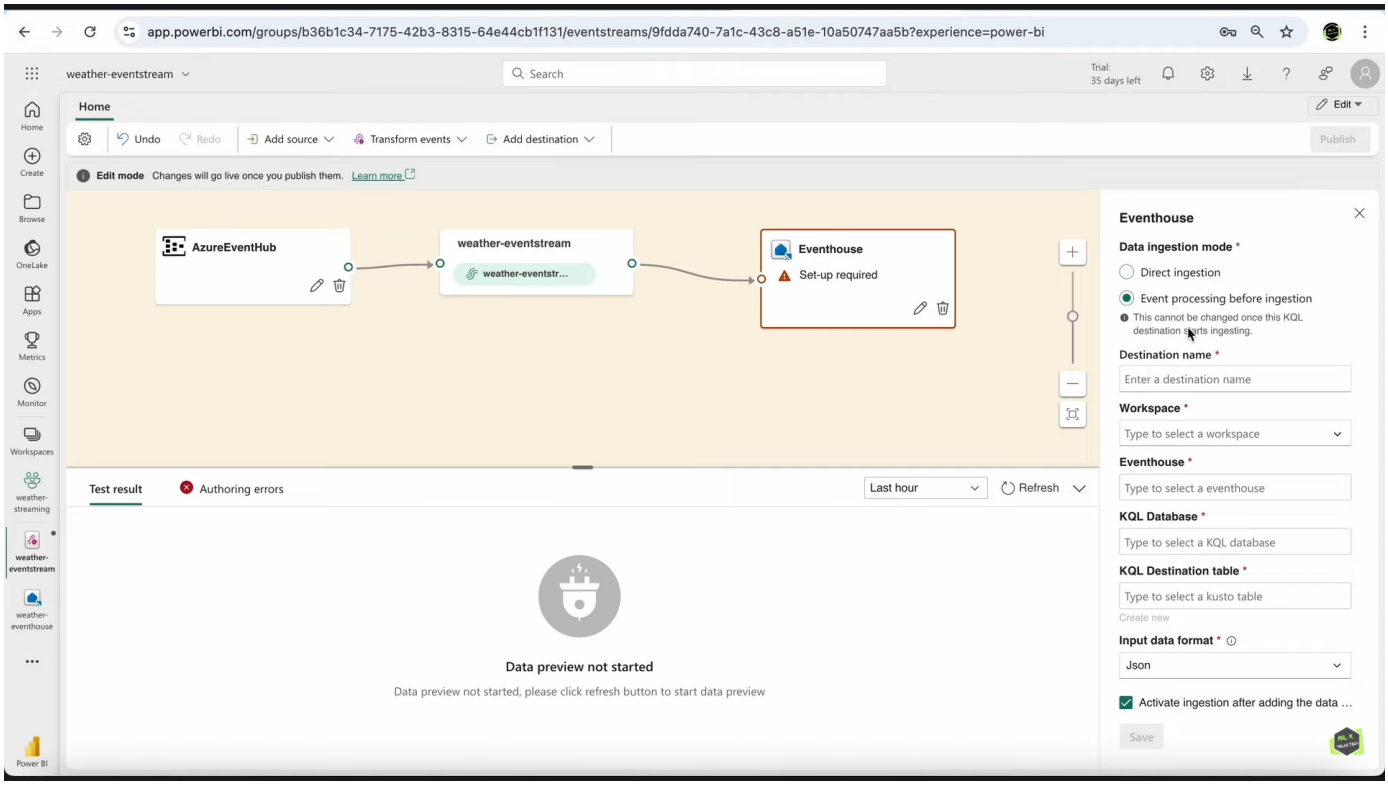
Here is consumer group--->default consumer group

Goto eventhub--->Data explorer--->lets verify the group--->click next--->add button click.



Configure the target KQL

Click on option --->eventhouse--->event processing before ingestion



Destination name(weather-target)

Workspace(weather-streaming)

Eventhouse(weather-eventhouse)

KQL database(weather-eventhouse)

KQL Destination table((new) weather-table)

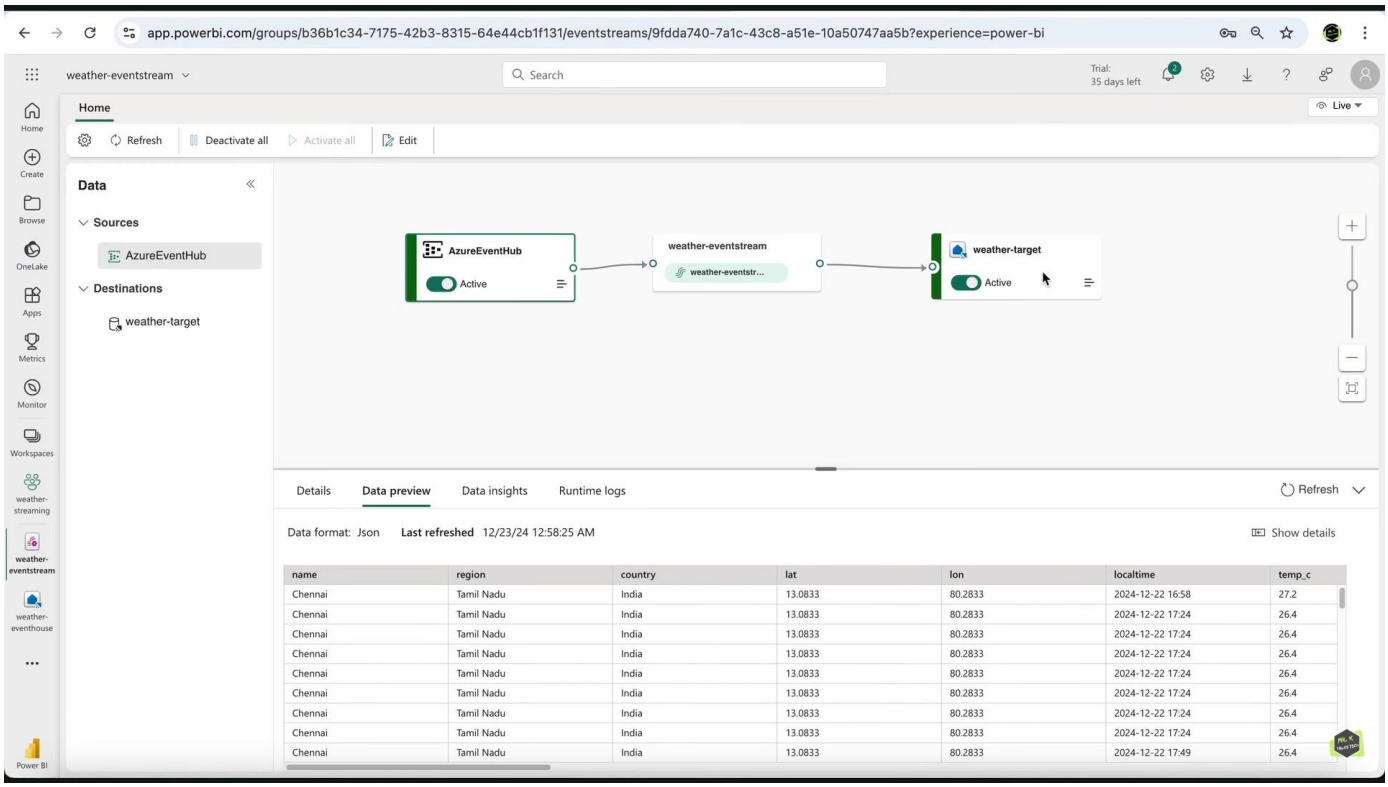
Input data format(JSON)

Save

Now we have configured source and target

Publish it

Now the loaded the data source to target to this pipeline



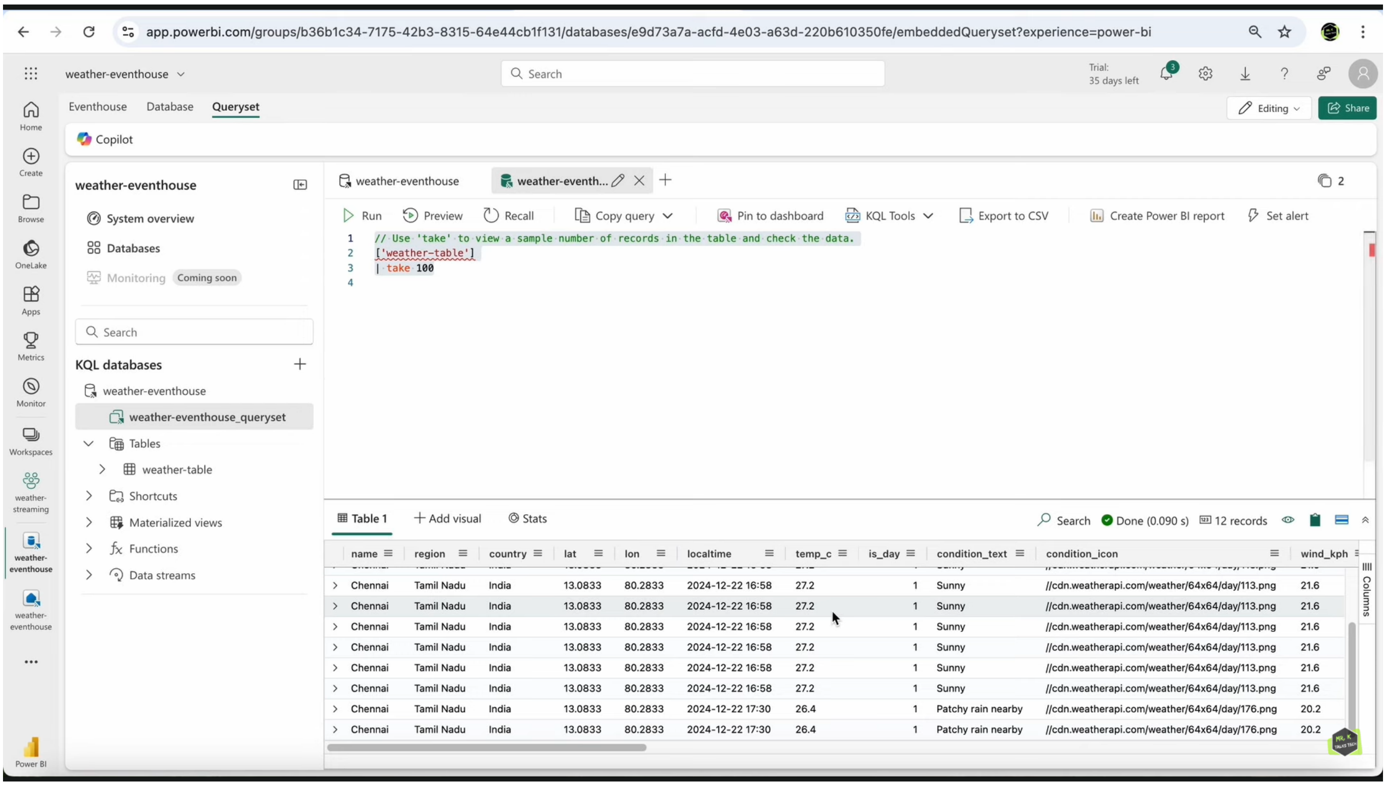
Now let's test it.

Open KQL Database--->KQL table--->click on right button--->query table(show any 100 records)

// use 'take' to view a sample number of records in the table and check the data.

['weather-table']

| take 100



//Let's test it

['weather-table']

| count

Click run button

Part- 8 (Data Reporting using Power BI)- End to End Streaming Azure Data Engineering Project

Open workspace

Let's discuss about query set what exactly query set it is collection of queries we can run to analyse data from the tables. Like filtering summarizing but in sql we use view.

KQL is bit different from SQL Syntax.

--

expalin

Select max(eventprocessdUtcTime) as latest_date from "weather-table"

For equivalent of SQL--->KQL right click on result copy that paste

["weather-table"]

| summarize NoColumnName=max[EventProcessedUtcTime]

| project NoColumnName1

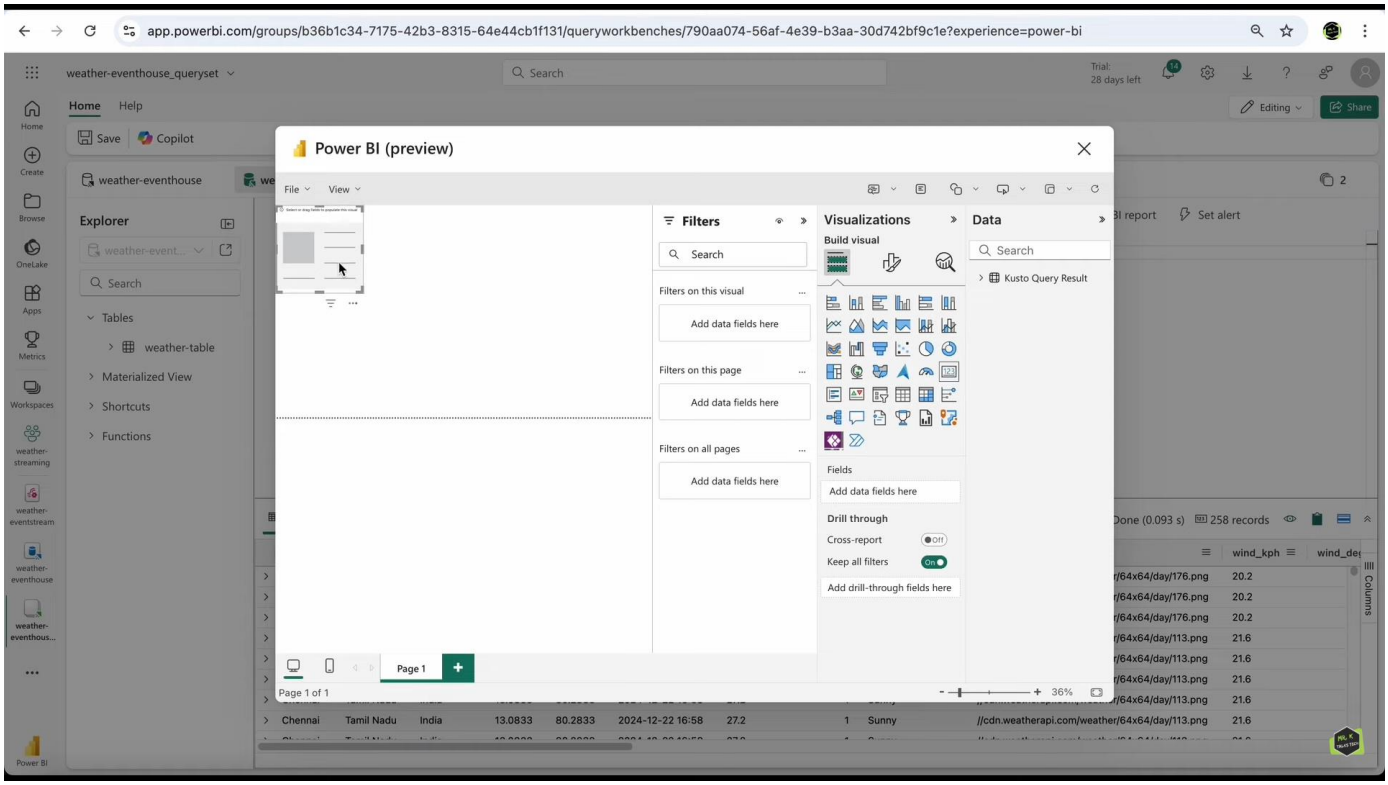
Let's fetch all records

// use 'take' to view a sample number of records in the table check the data.

['weather-table']

Let's create a power BI report

We can see popup window



Drag the card visual in canvas.

Add the column into that visual into that

Count of cloud changed into that.

Let’s use line chart

x-axis---->average of humidity

y-axis--->localtime--->click on file--->save(name)--->choose workspace

let’s got workspace--->refresh.

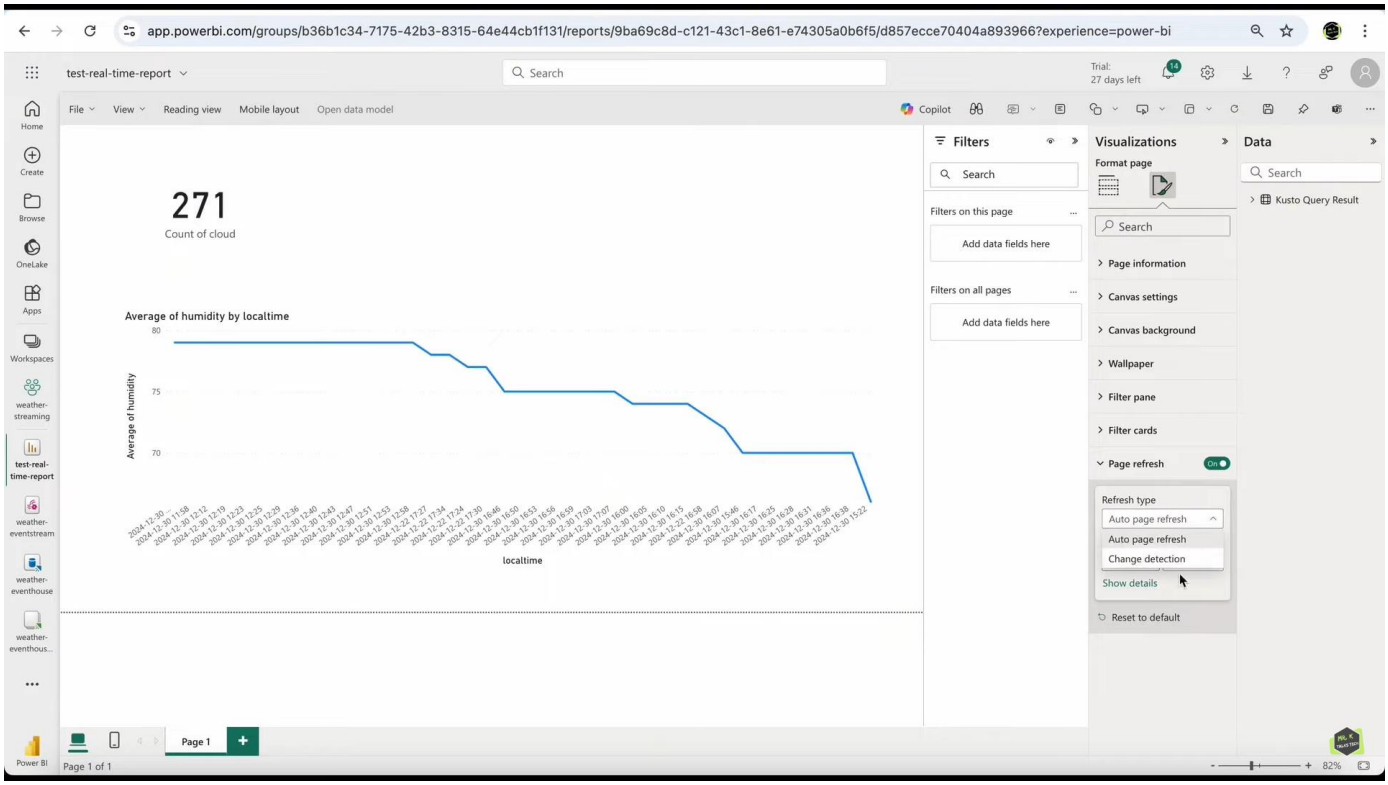
Open report

The is increases when refresh the button.

And report right side--->format on report page---> page refresh(enable)

Refresh type(auto refresh option)

Refresh



The count keeps on updating and this is basic requirement test case one of it.

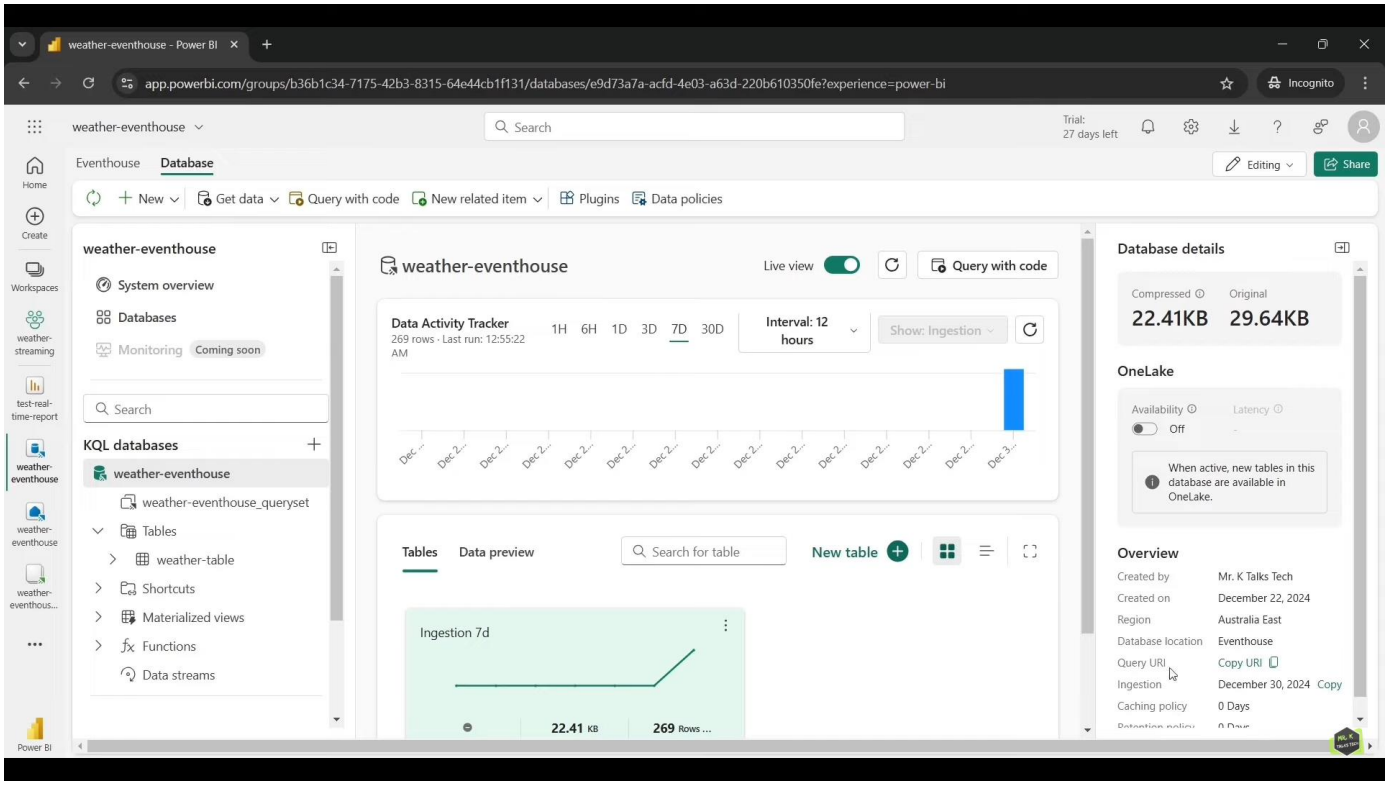
And also, how we can use Power BI desktop.

Login with domain mail credentials

Letes connect KQL database--->get data--->more options--->Microsoft Fabric--->KQL database.

Fill few details

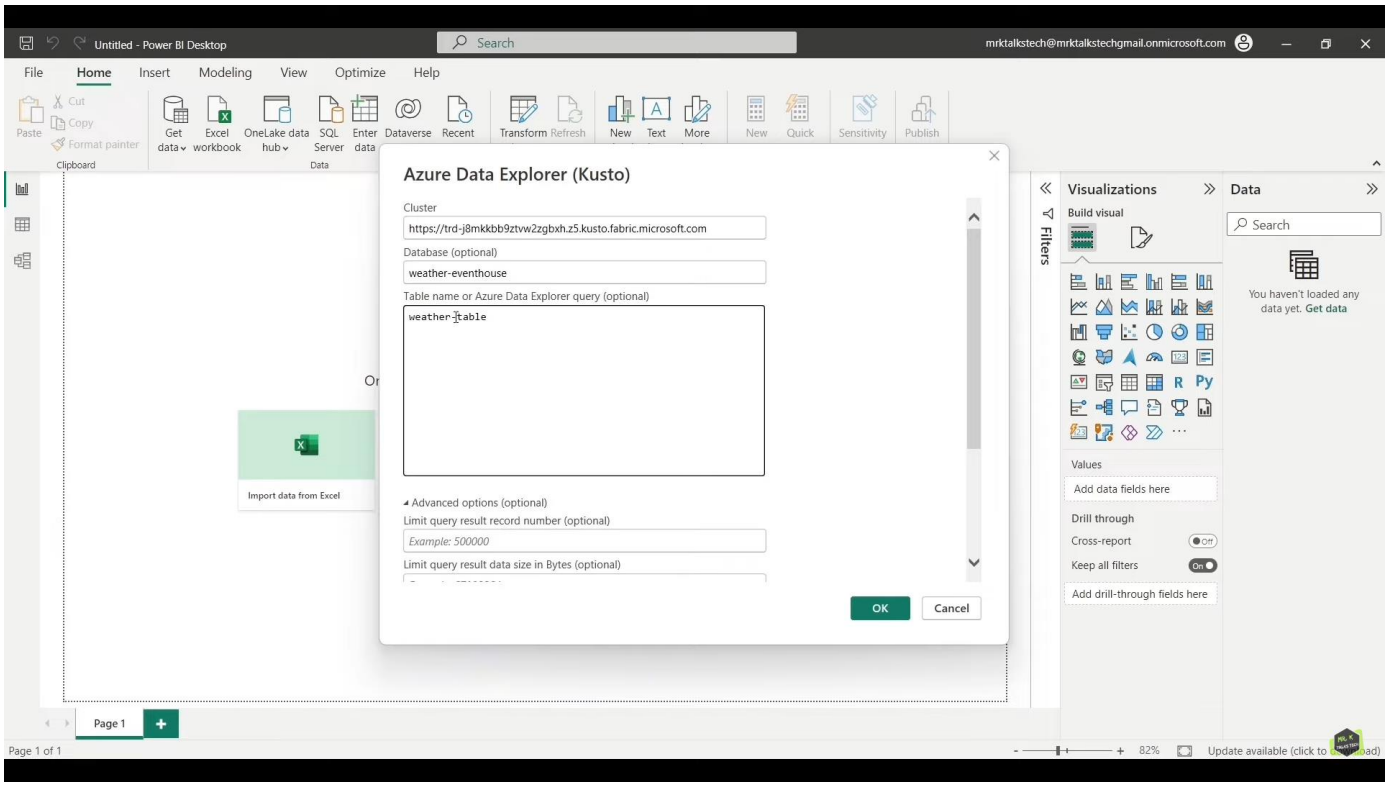
And for this lets go to power BI service--->click weathe-eventhouse--->KQL database--->in the right side there is a copy URL(Query URL)



Paste it in power bi desktop--->in cluster.

Database name--->weather-eventhouse

Table name---> [“weather-table”]



Let’s scroll down we see some options are optional we can skip it.

But in Data connectivity mode

Import (we can modify data when we use) choose this one

DirectQuery

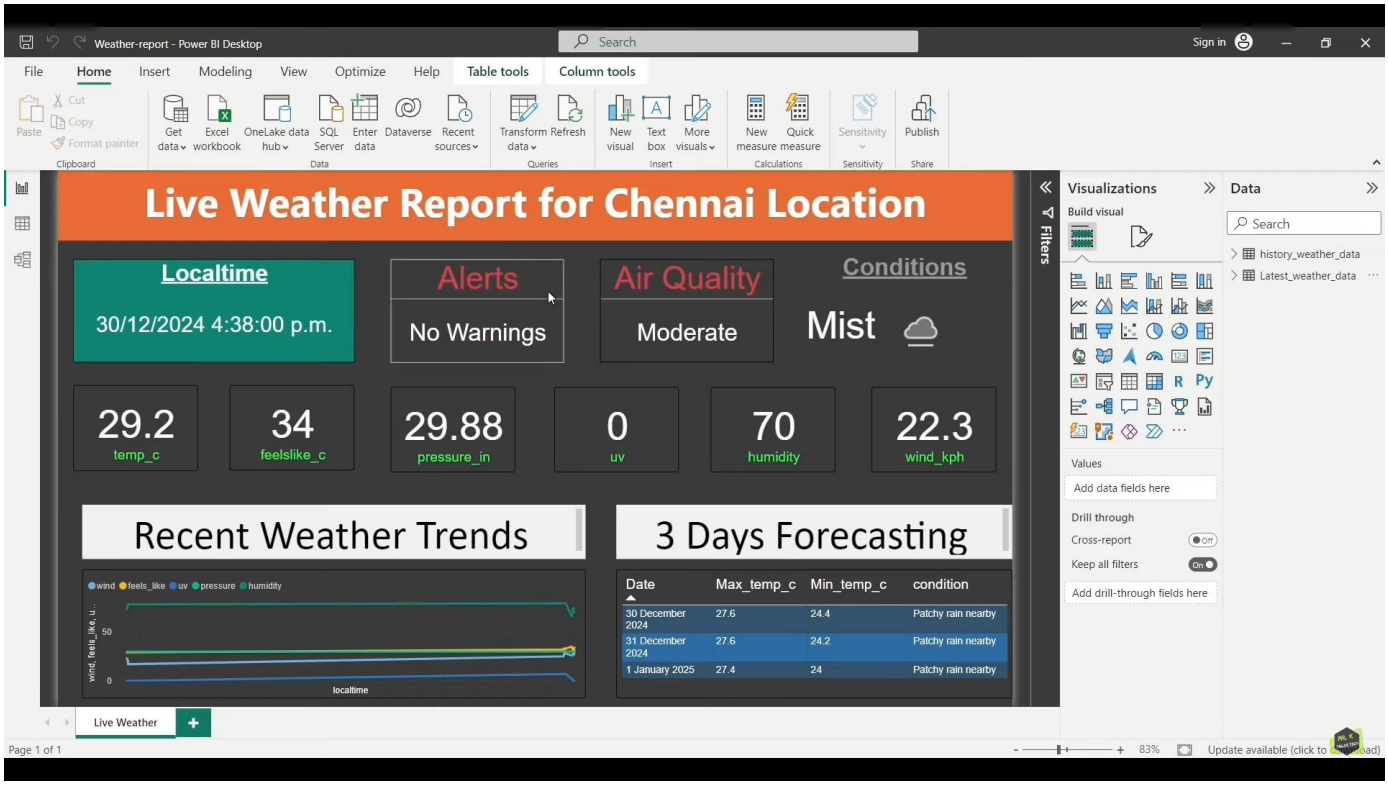
Now load the data in Power BI.

Now let's create report

Drag the card in canvas--->choose the column add in that card.

In the import mode the data report is not updating real time in such case we use manual refresh.

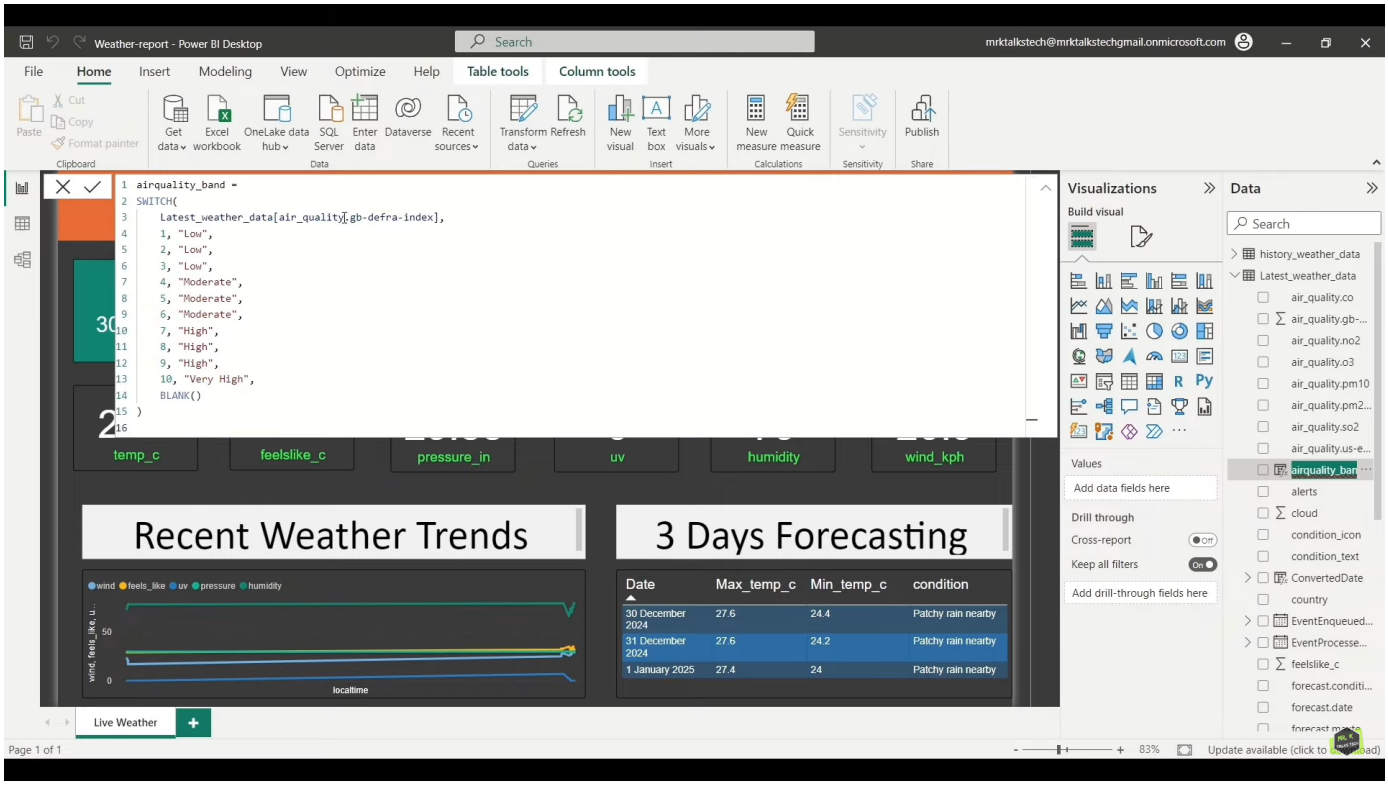
And final report page is.



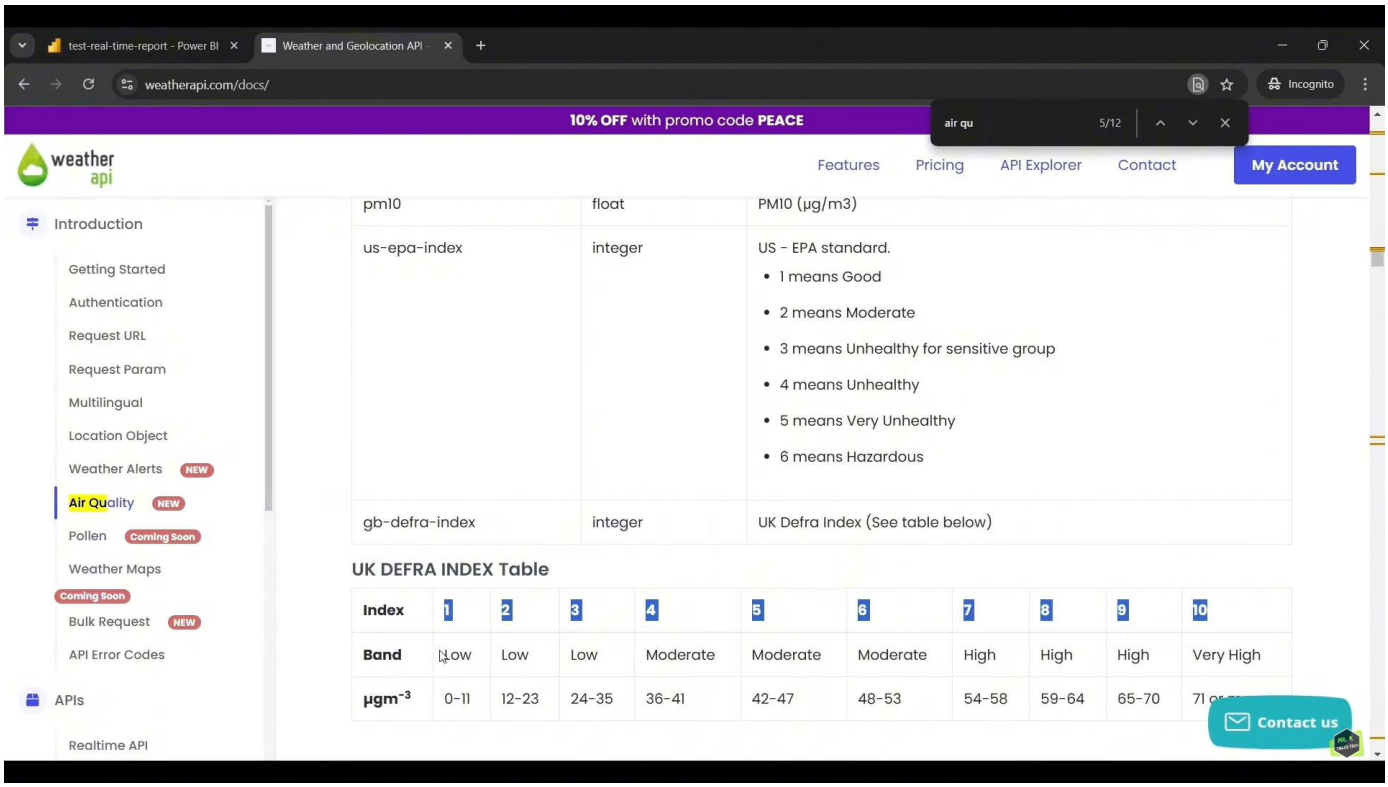
This report is weather fore casting real time streaming.

There two datasets

1. History
2. Latest another dataset



Lets go to weather API--->docs section--->Air Quality.



Let's go to Transformations option--->click history_weather_data--->advanced editor

Both the tables update the URL to KQL database--->click and save.

After that the report is connected to KQL Database. And all the database visuals updated to KQL DB.

Click on Refresh and see the latest changes in report.

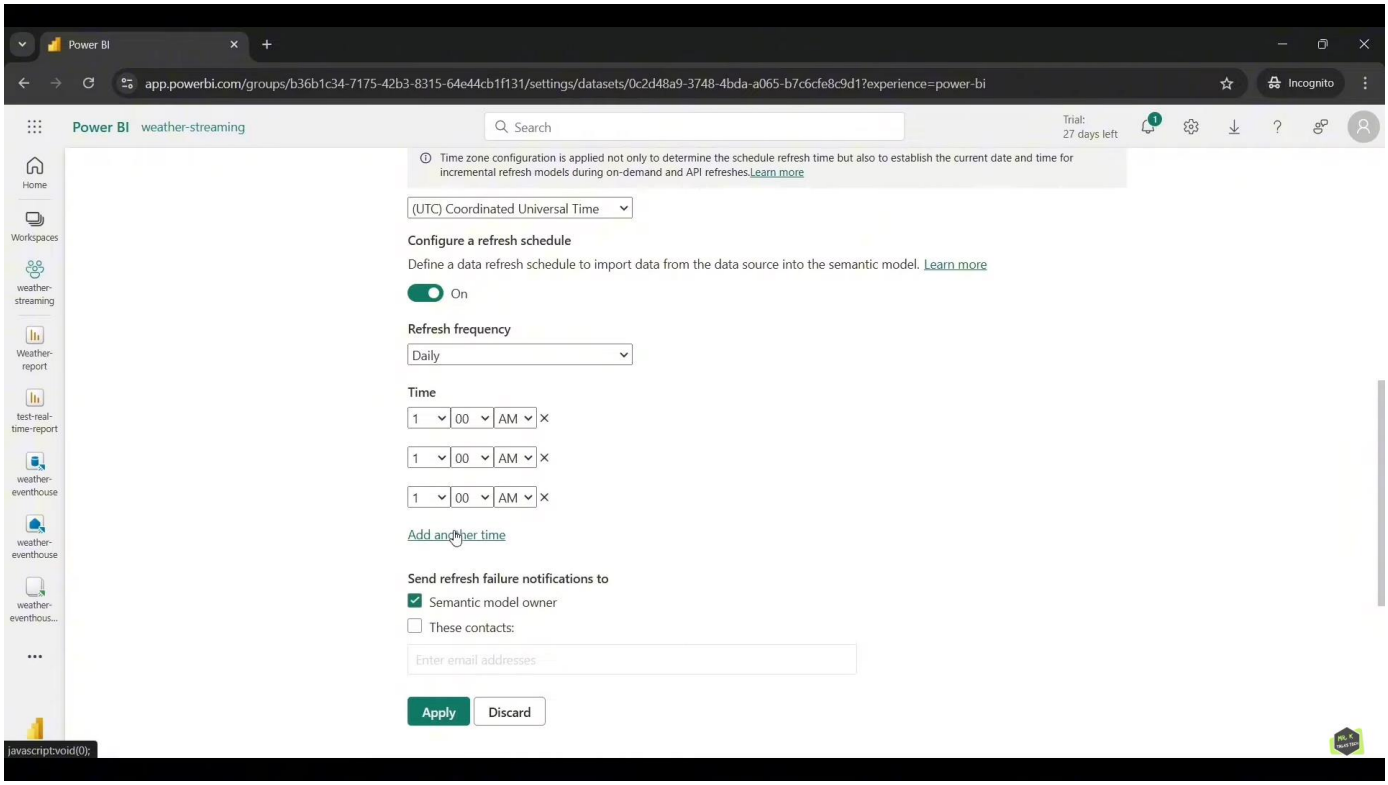
And publish the report---login to power bi desktop

Lets goto weather streaming workspace--->we see report which was created in desktop.

Let's open--->this report is not updated real time--->schedule refresh or manual refresh.

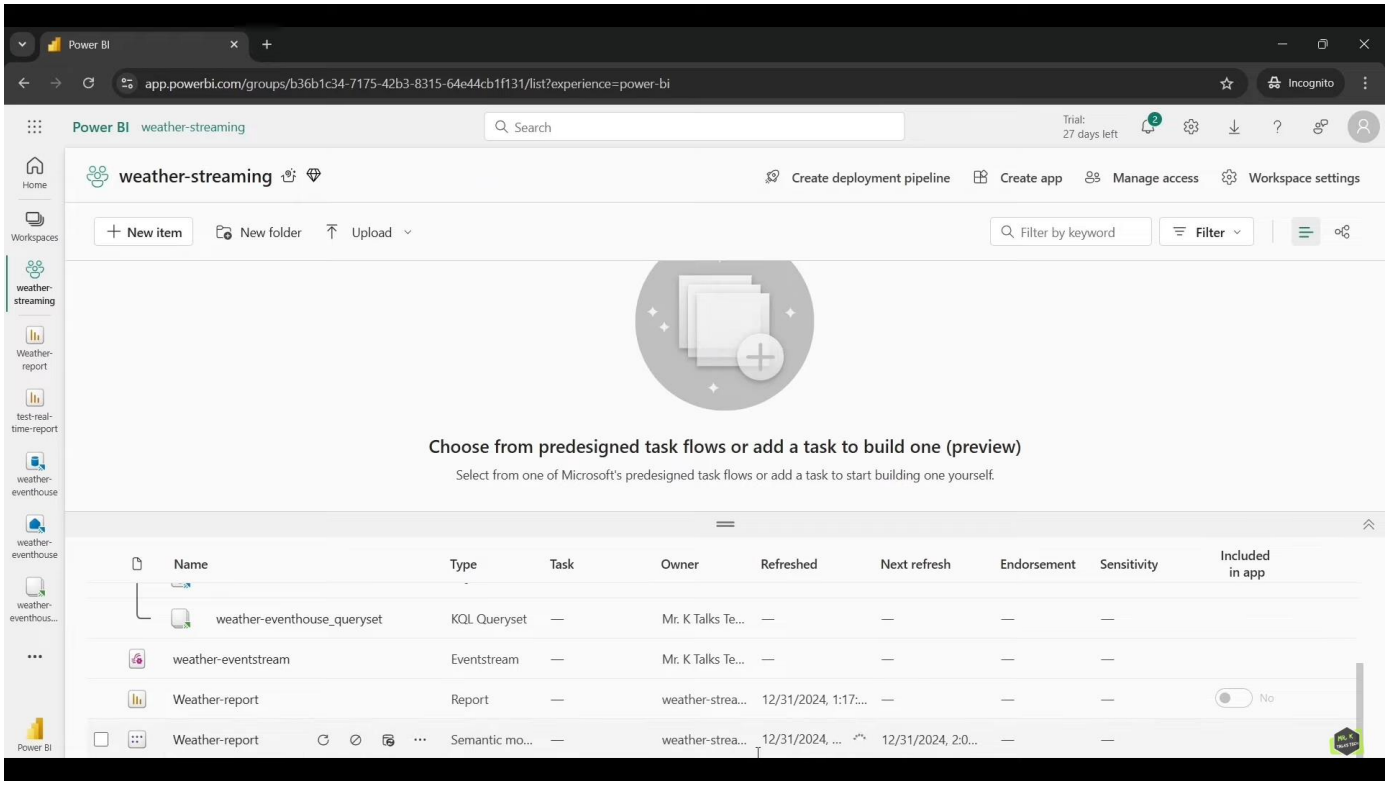
If we setup schedule refresh--->go to weather-streaming workspace--->semantic model--->click on right option--->settings

Refresh --->configure a refresh schedule(enable)



We can add 48 refresh schedule time stamps.

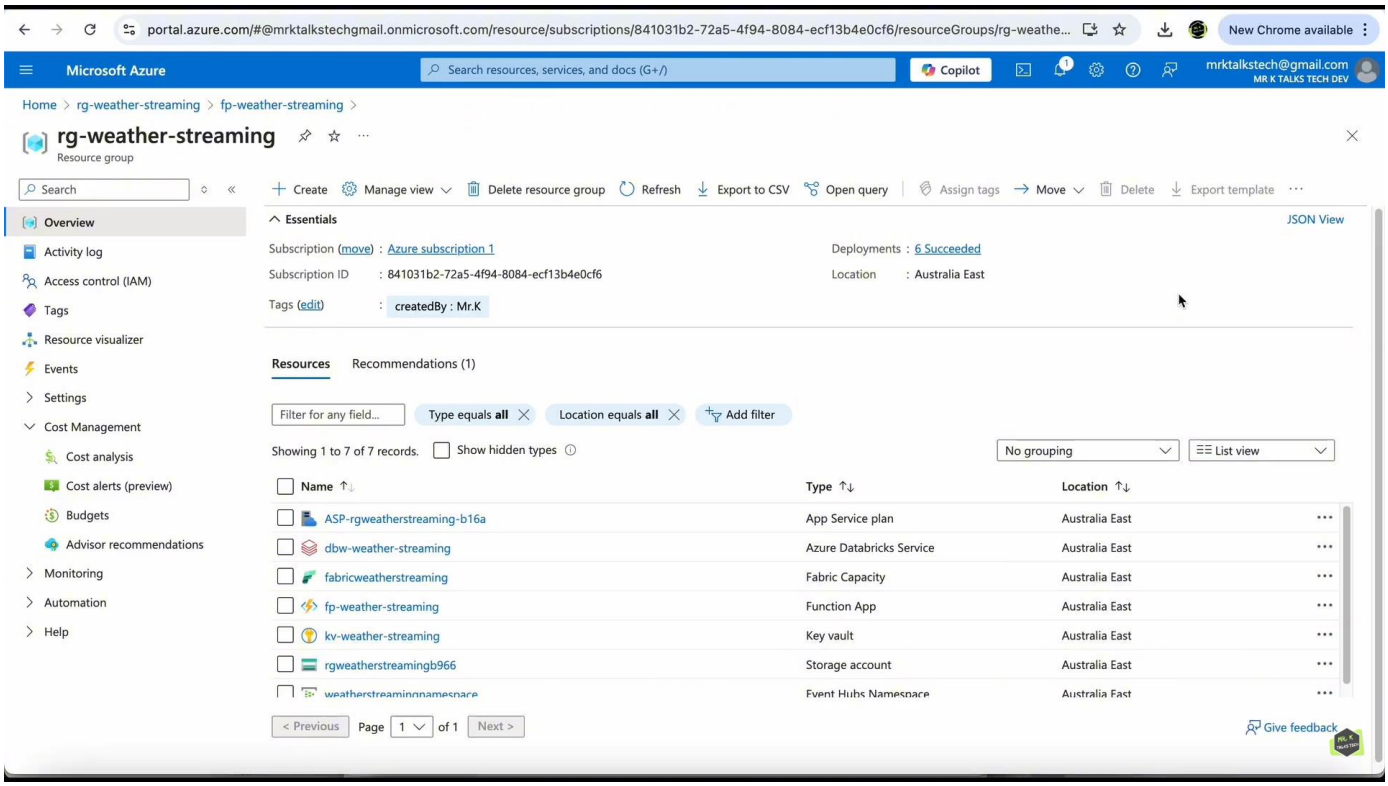
And if we can do manual refresh here is the option in workspace--->report refresh option.



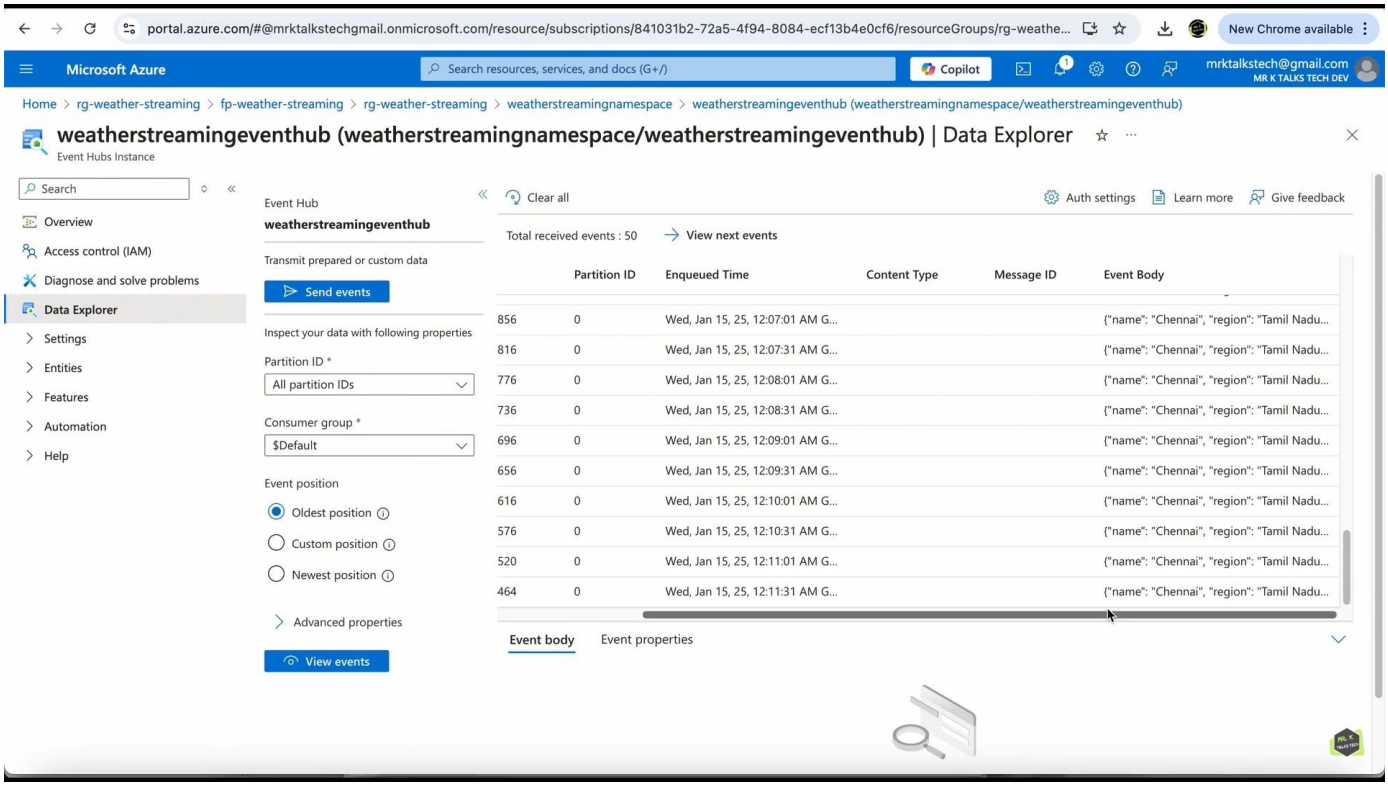
We can see the latest updated data in report.

Part- 9 (Configuring Real-Time Alerts)

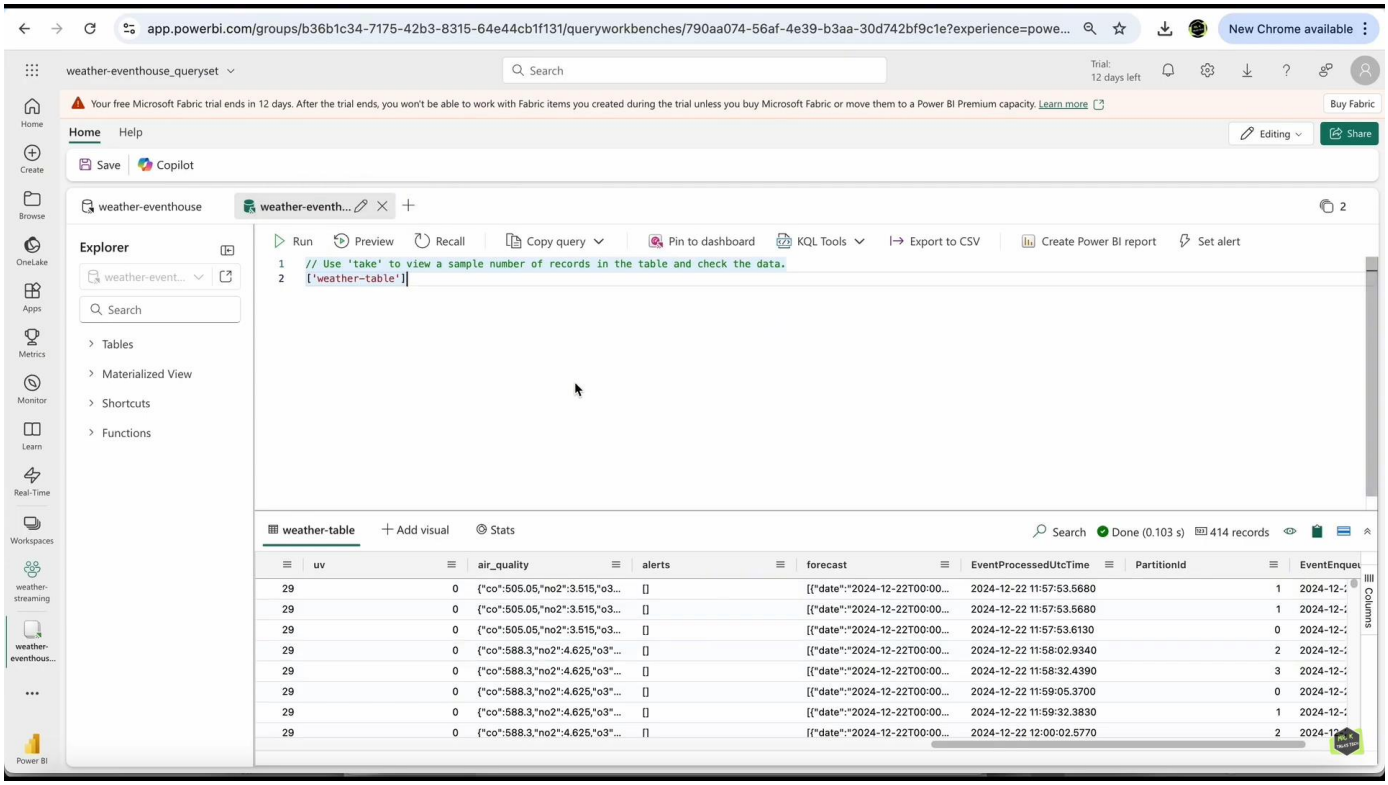
In this we discuss configuring the alert mechanism for information so basically from this alert mechanism for our entire pipeline so basically as you all know that currently weather API and ingesting the weather information from it and in the ingested data we are also getting.



Open event hub namespace--->lets go to event hub--->data explorer--->view event



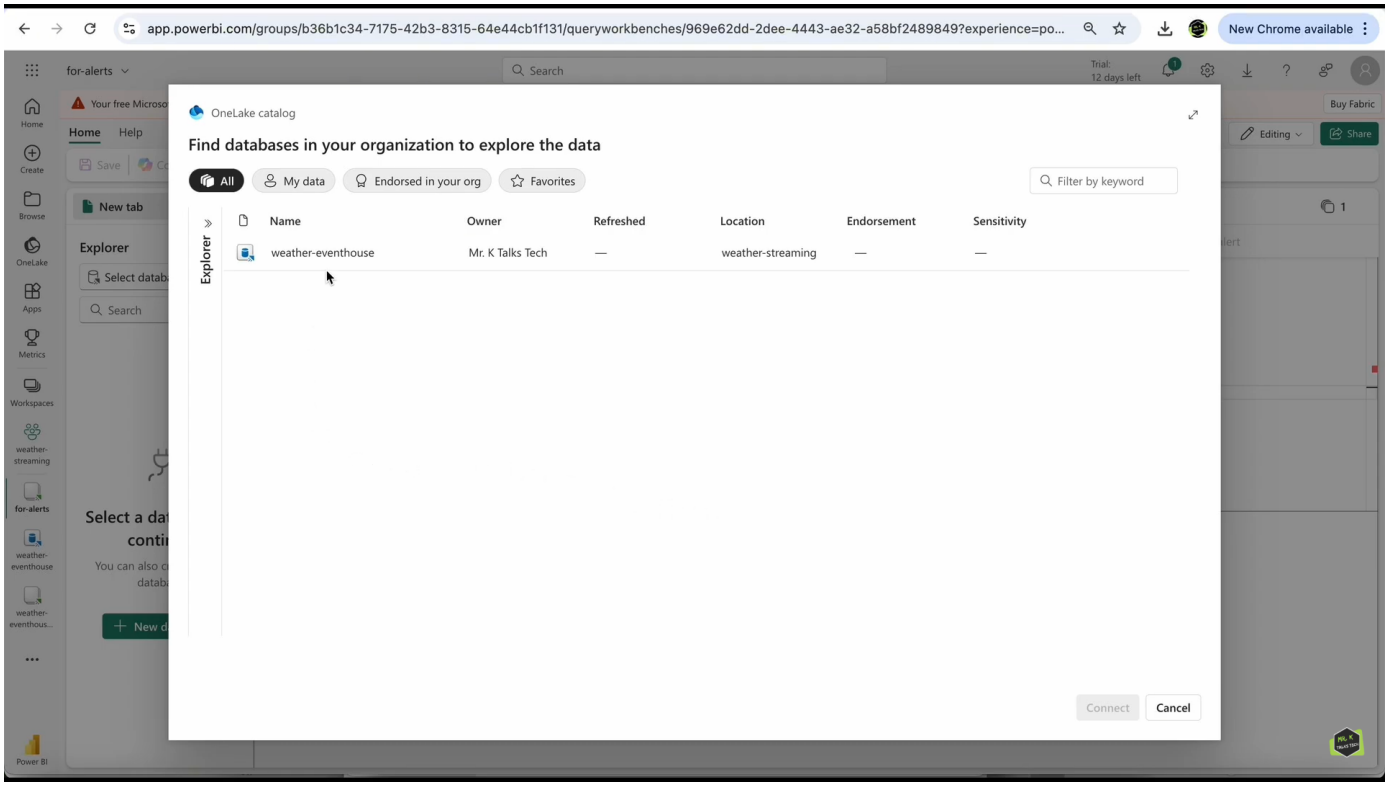
Goto Fabric---->workspace--->open existing quey set



Write this query logic we need a query set but we should not modify existing query set because it is already connect to power bi report make any changes to this it will reflect the power BI report as well as.

Let's create a query set and then see

Let's goto workspace--->KQL Database--->create new related item(KQL QuerySet(name))--->weather-eventhouse



// see how many records are in the table.

Your_table_here

| count

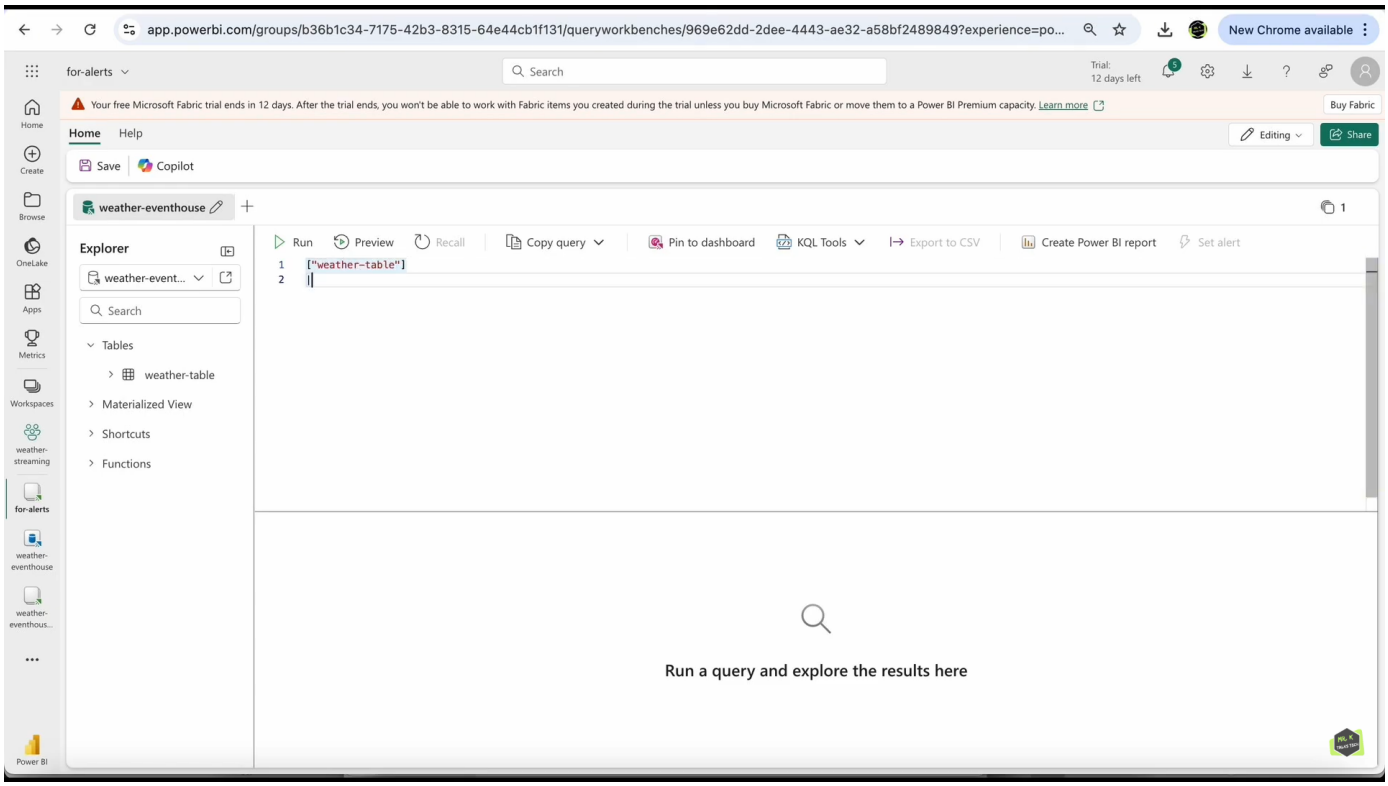
// this query returns the number of ingestions per hour in the given table.

your_table_here

| summarize IngestionCount = count() by bin(ingestion_time(), 1h)

Lets goto weather workspace and confirm KQL Queryset

Lets open new query set



[“weather-table”]

| take 10

// let’s write a quey you find this information u’ll be able to send the alerts to the email so in order to find this information we need to write the query

[“weather-table”]

|where alerts !=’[]’ // filter for records where an alert condition exists

|extend Alertvalue= tostring(alerts) // Extract alerts as a string

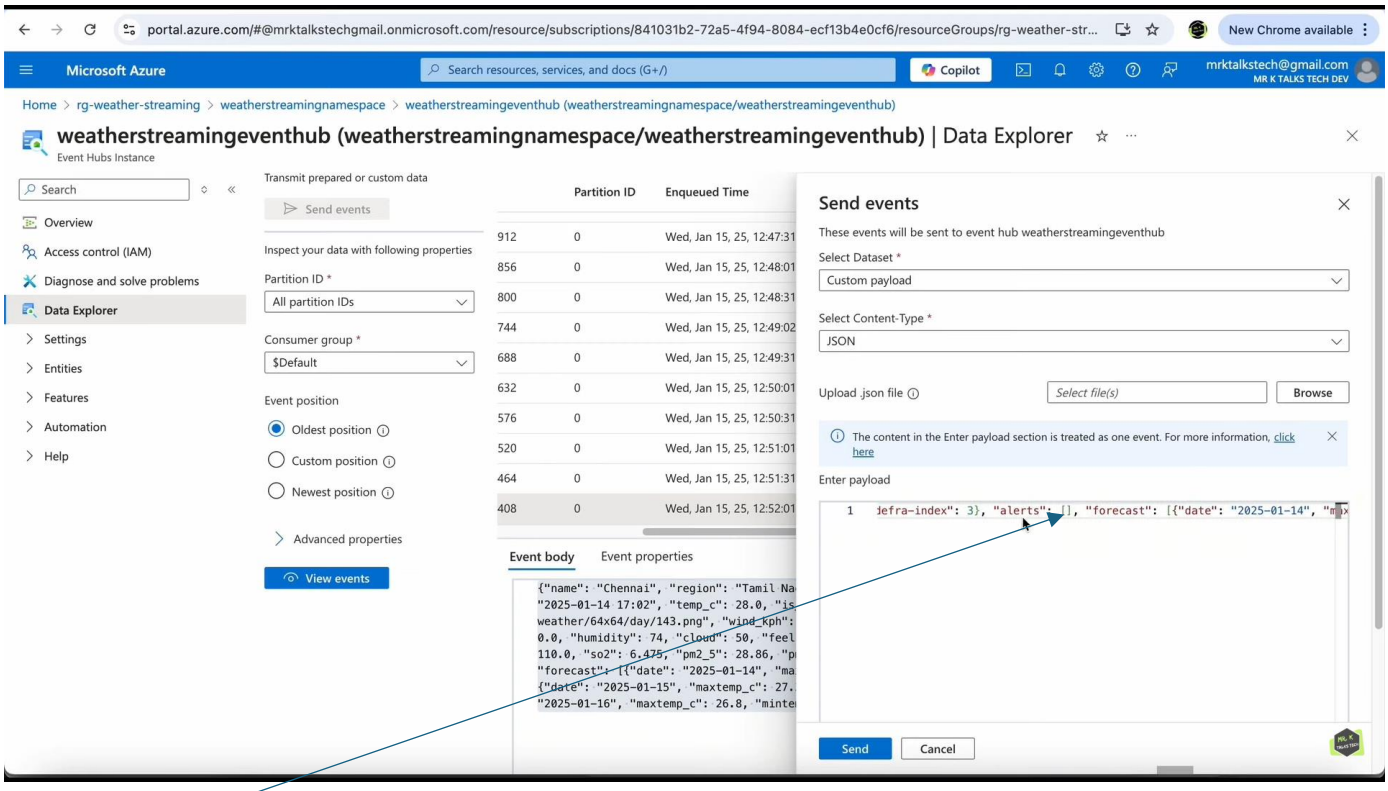
| summarize LastTriggered = max[EventProcessedUtcTime) by AlertValue

Click and run

Empty results which means that there is no active alert at the location manually set the alerts

Lets goto event hub--->view event button--->lets scroll down--->see the last event body--->copy

Manual send--->send events

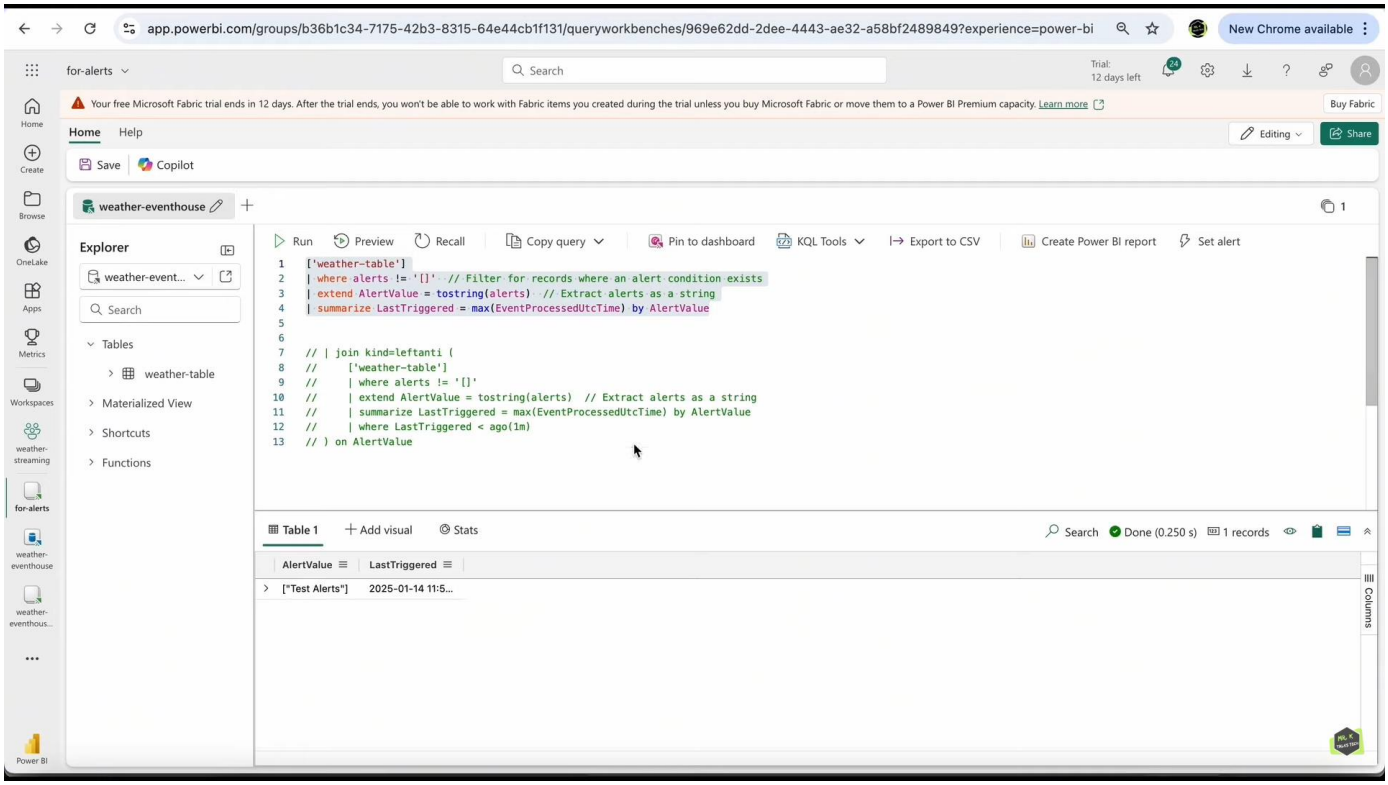


“alerts”: [“test Alerts”]

Then send events button.

You see here the content type for this event is application /JSON which is our custom created here will be able to see the test alerts value that we have defined and now.

Lets goto Fabric--->weather-eventhouse--->select first four line and rub it.



The same alert will send email notification.

| join kind=leftanti (

['weather-table']

| where alerts != '[]'

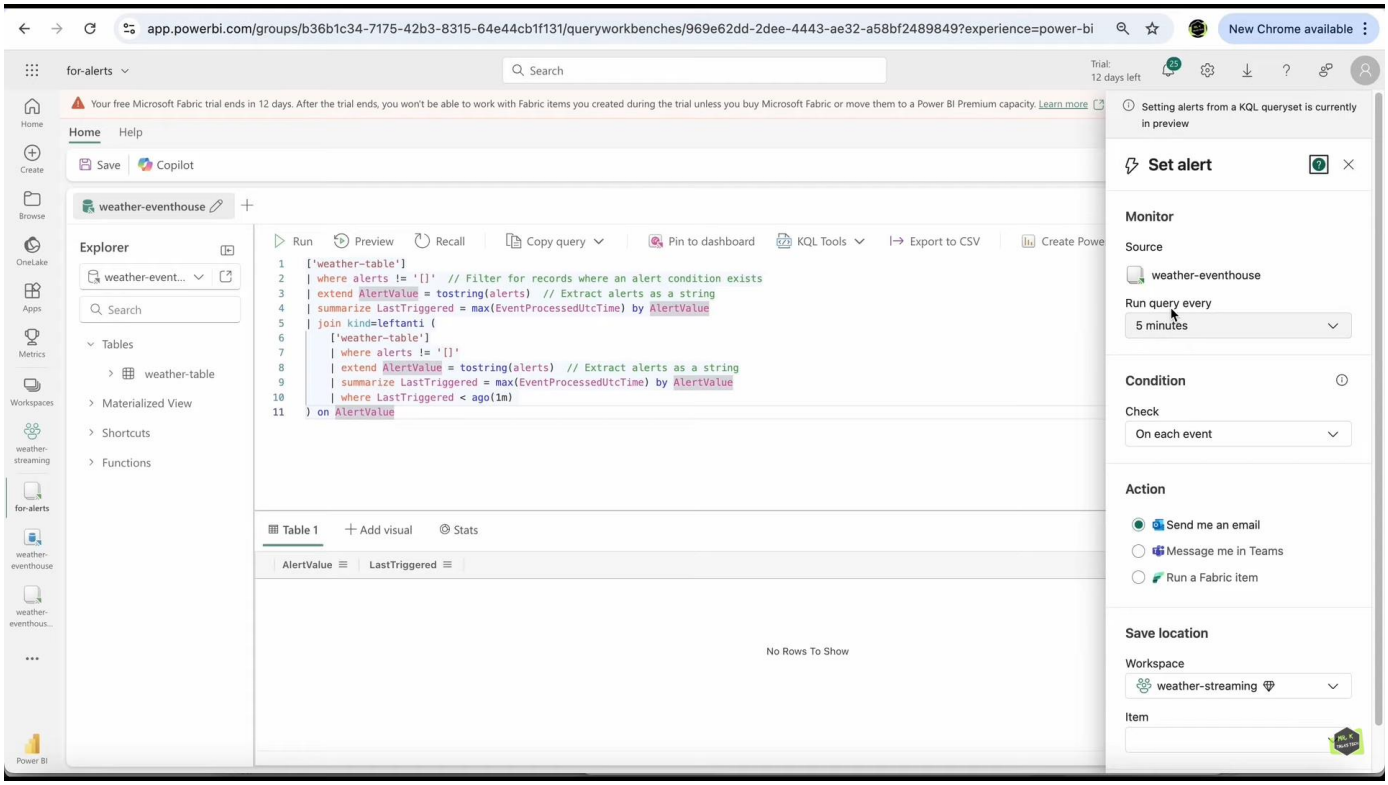
| extend AlertValue = tostring(alerts) // Extracts alerts as a string

| summarize LastTriggered = max[EventProcessodUtcTime) by AlertValue

| where LastTriggered < ago(1m)

) on AlertValue

Let's test and Click and run.



Now let’s set alert

Run query every--------->1 minute

Condition

Check---->on each event

Action

Send me an email

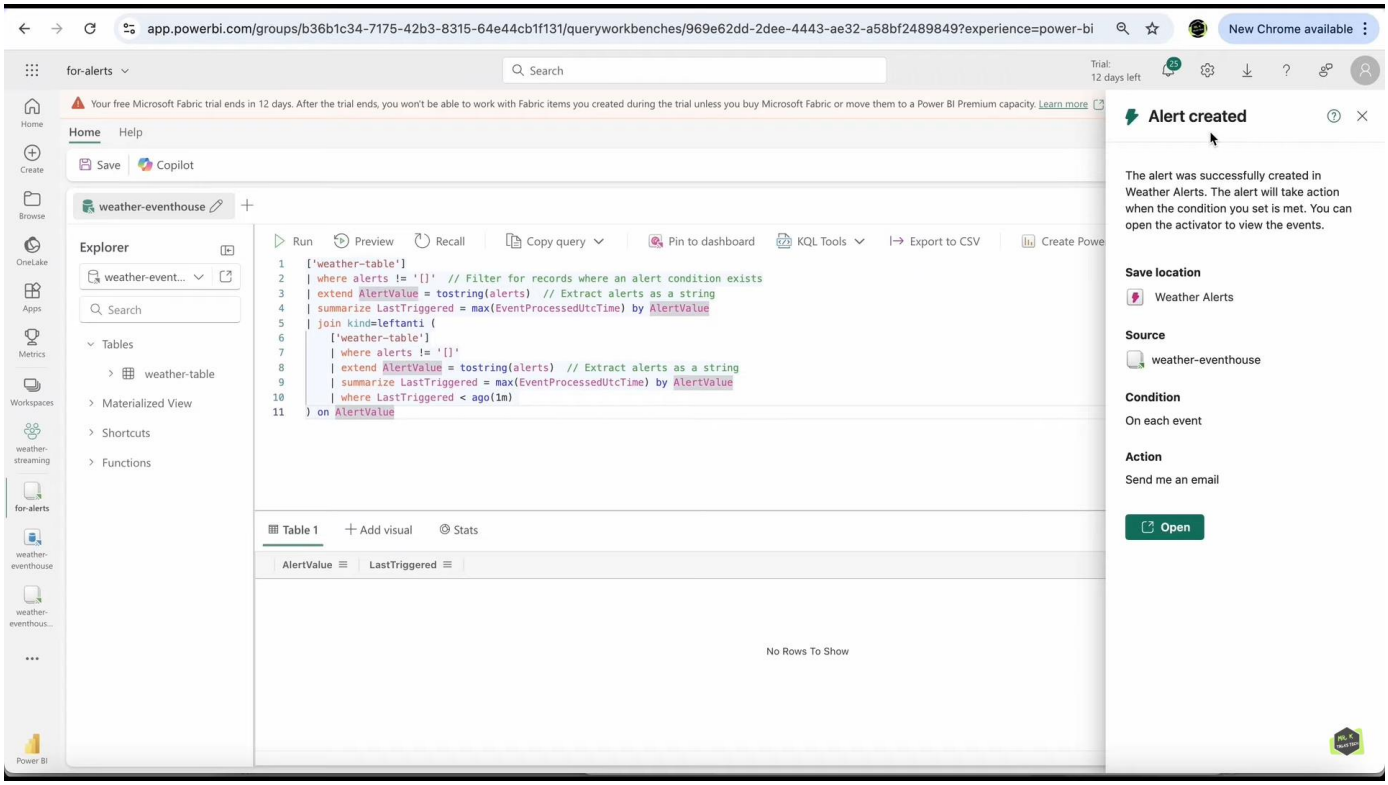
Message me in teams

Run a Fabric item

Save Location

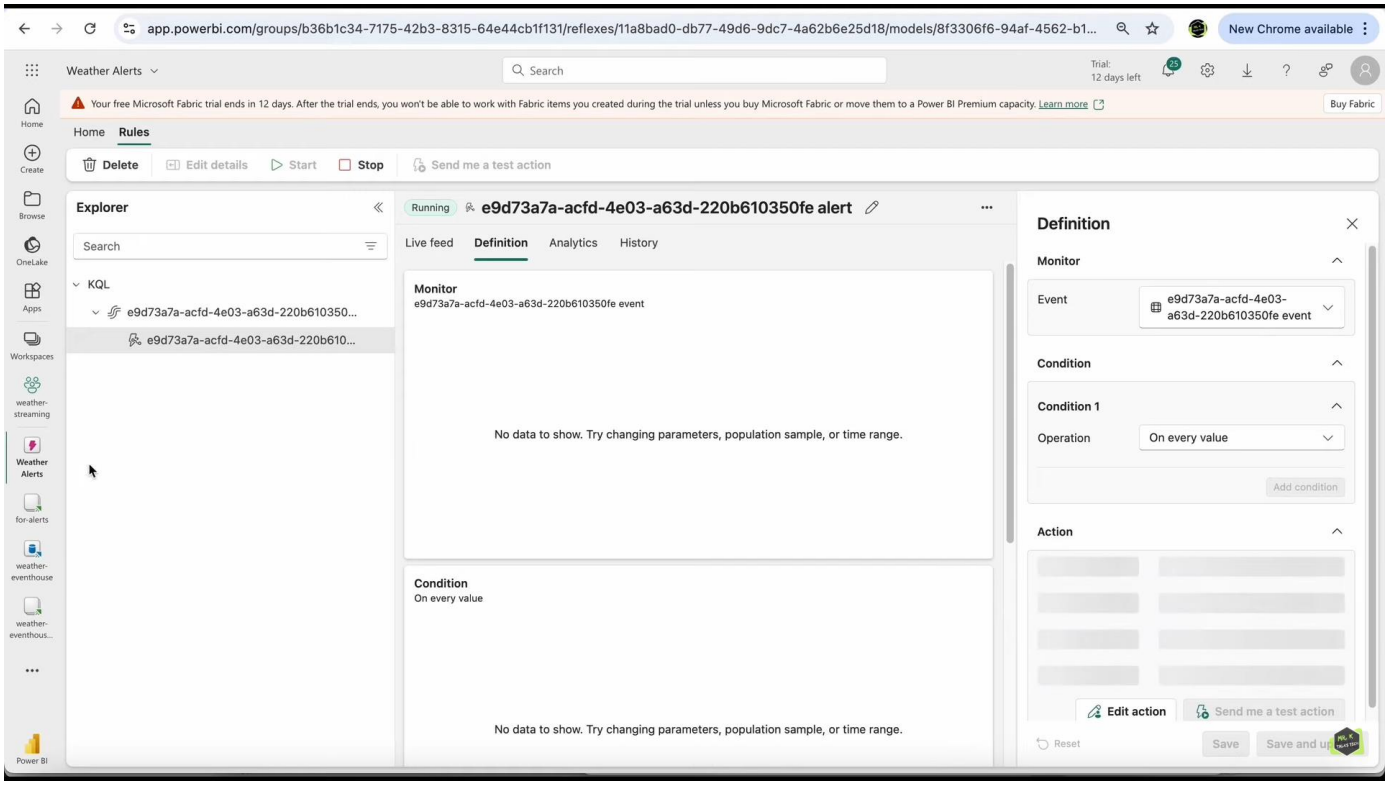
Workspace--->weather-streaming

Item--->create a new item(new item(weather alerts))



Click open.

It will navigate Data activator



In the action menu.

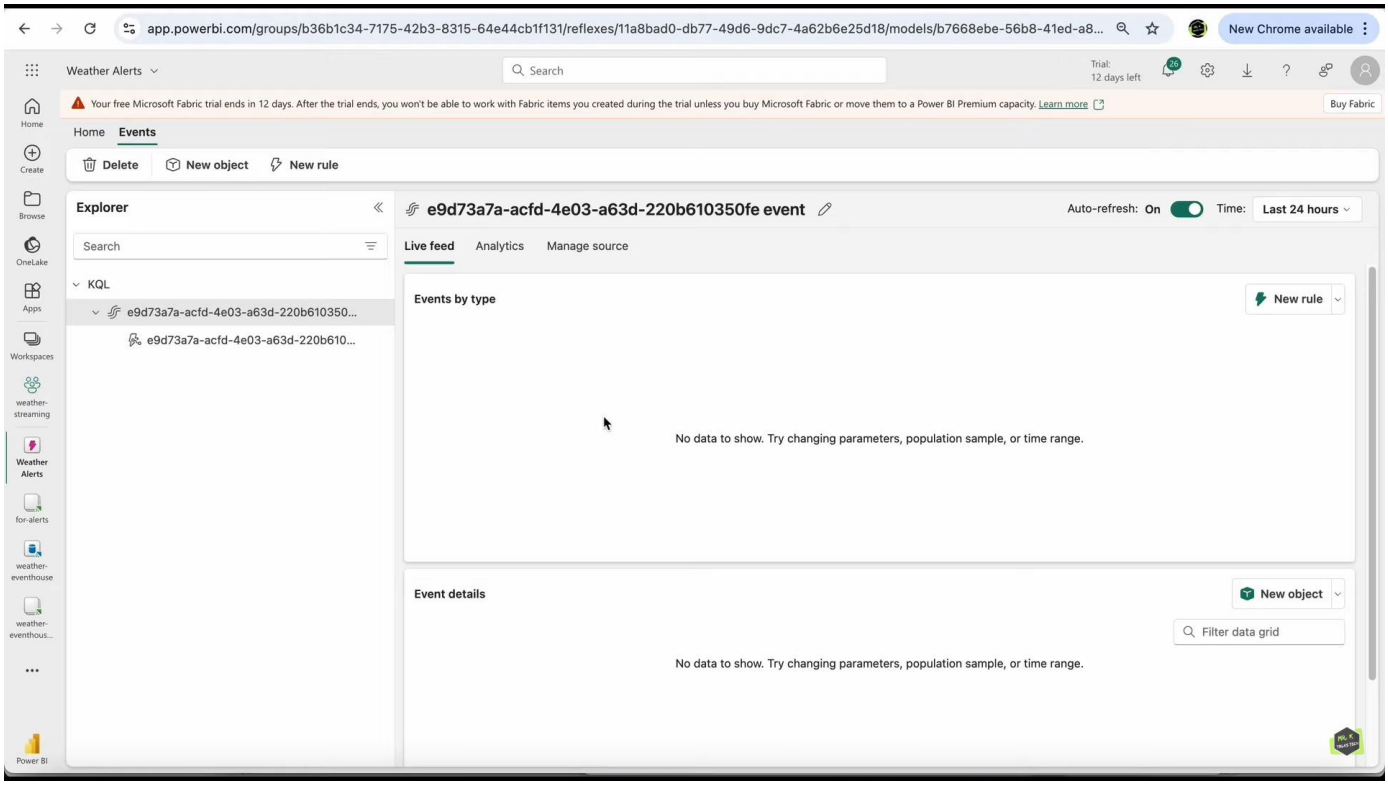
Type
To
Subject
Headline

Save and update

And the status is running which means this activator is active and the KQL query logic 1 minute automatically and if you receive any weather alerts and detect and send it as an email and once it is detected it will be able to see.

And lets click on alert detector.

Click auto-refresh



Let’s test this functionality.

Currently no alerts in location. In this no alerts information the only way to test this by sending a test using event hub as earlier so send a test it should automatically detected and send email.

Let’s go to event hub

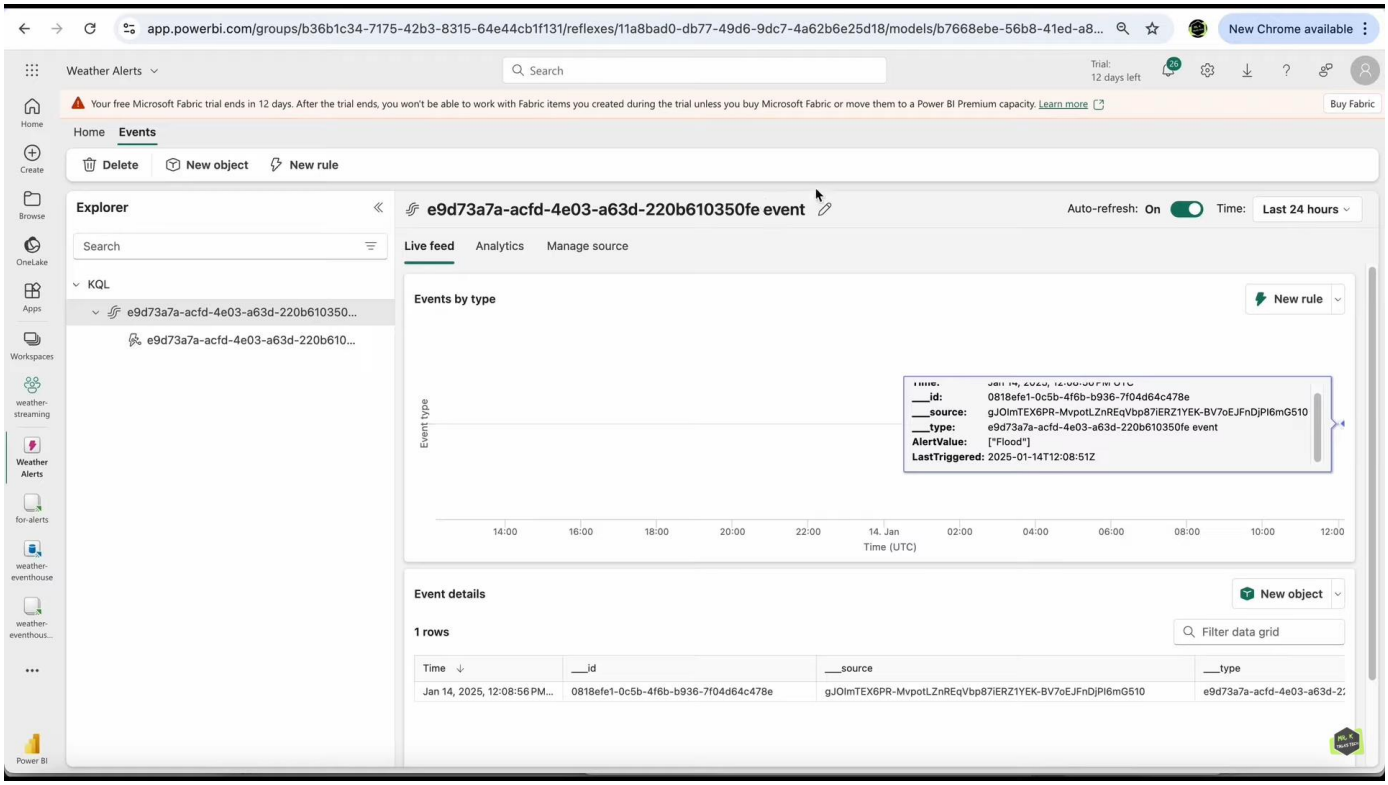
Get the latest weather event data ok so now lets the event body the last event and then then copy the latest weather information.

Copy and let’s send events.

Giver test alerts.

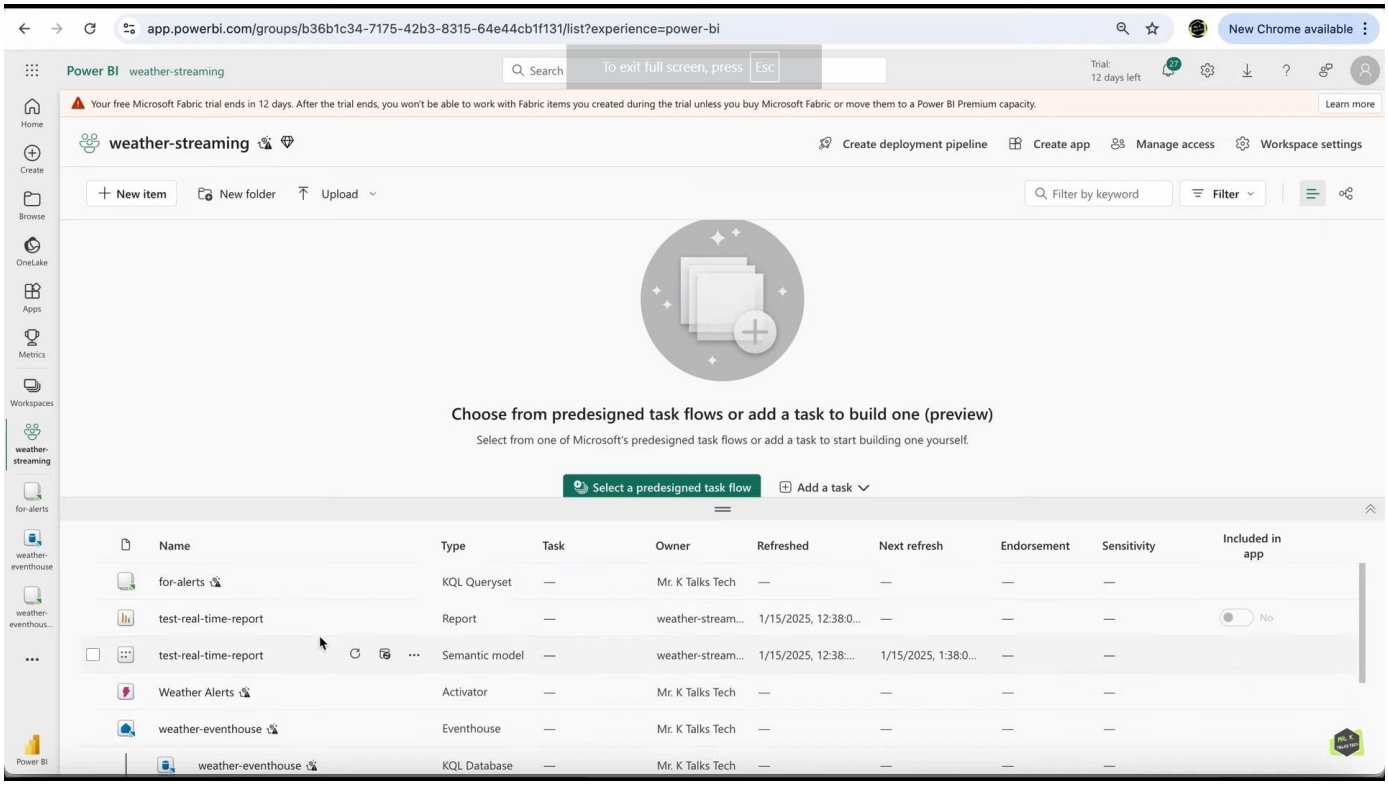
And send and let’s verify.

Let’s goto data activator.

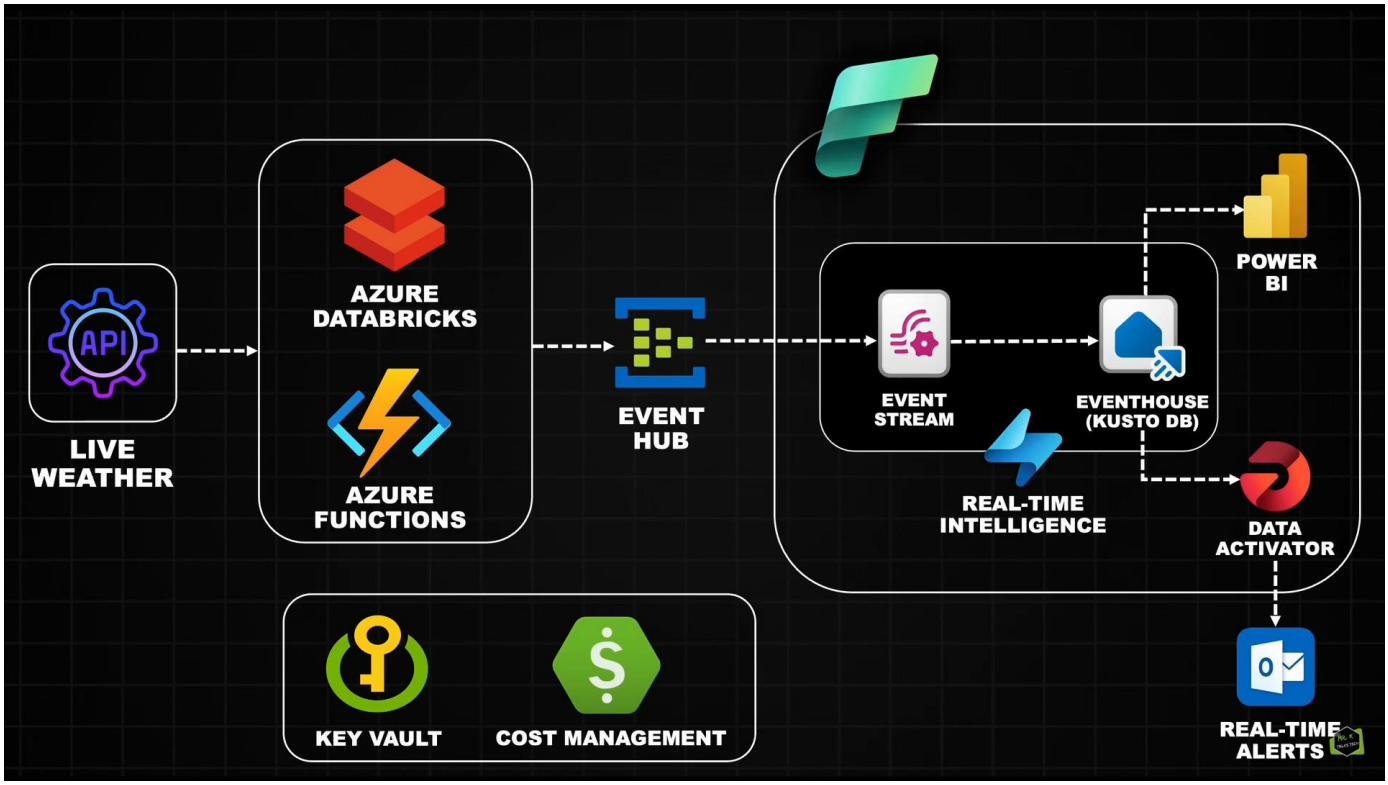


We can see the alerts in alert value, and we should mail automatically.

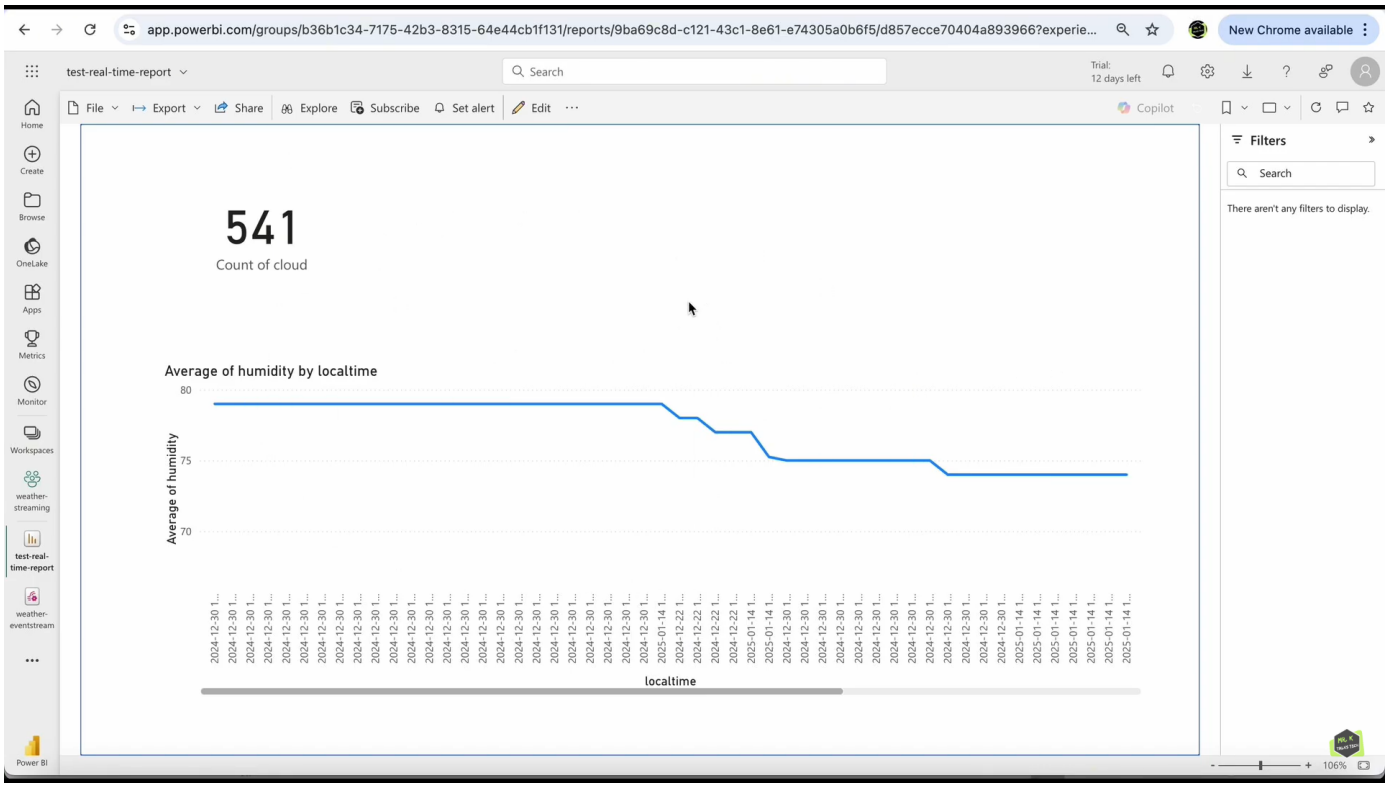
can see all the different resources that we have created so far such as the data activator **KQL** query set PowerBI report even house and K database okay, so I think now you have a clear understanding of how to configure alerts in Microsoft Fabric and with this we have successfully completed the alert configuration part.



Part- 10 (Final) (End to End Pipeline Testing)



Now here inside fabric power BI report



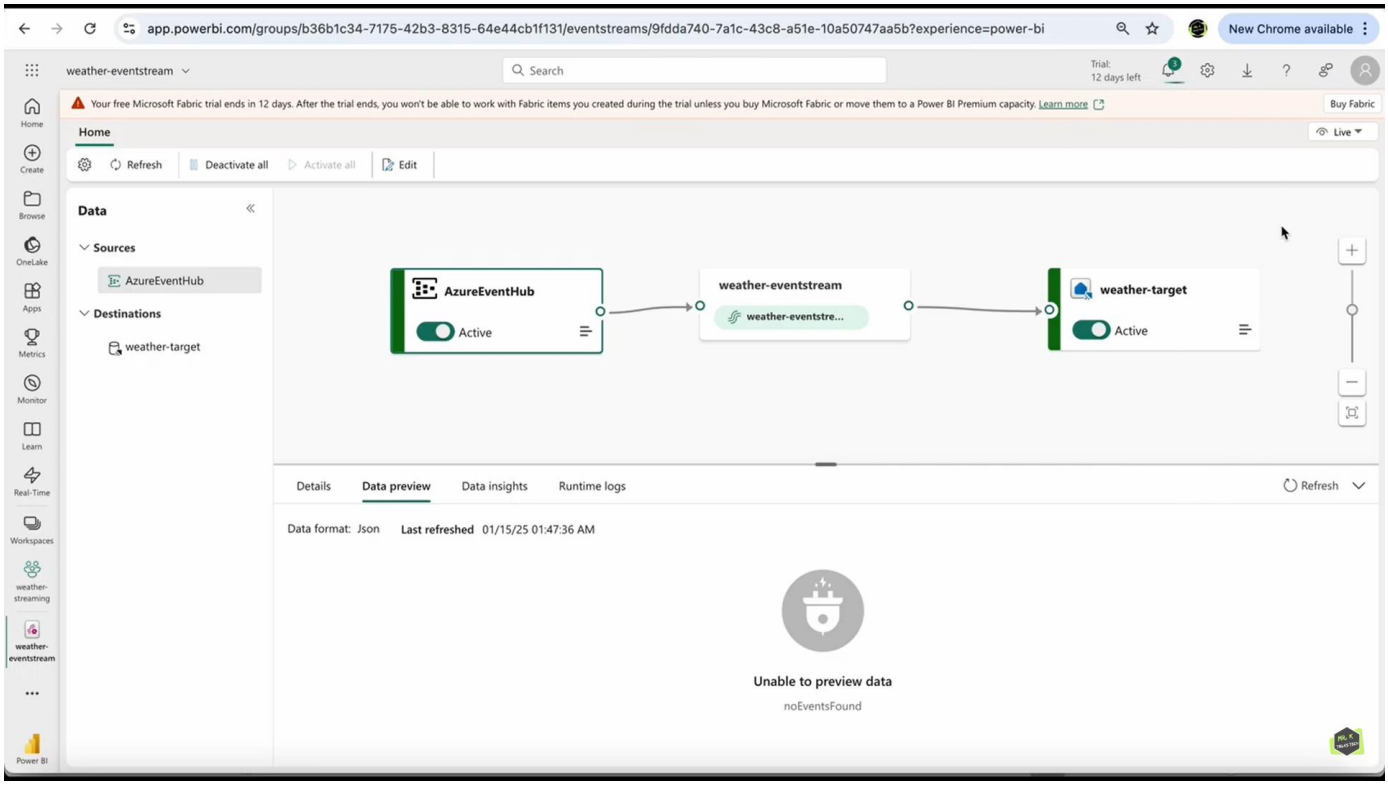
This is data is refreshed every one minute with latest data but its data is intentionally turned off the function APP which is responsible for the data ingestion this was done specifically to perform this end to end pipeline testing.

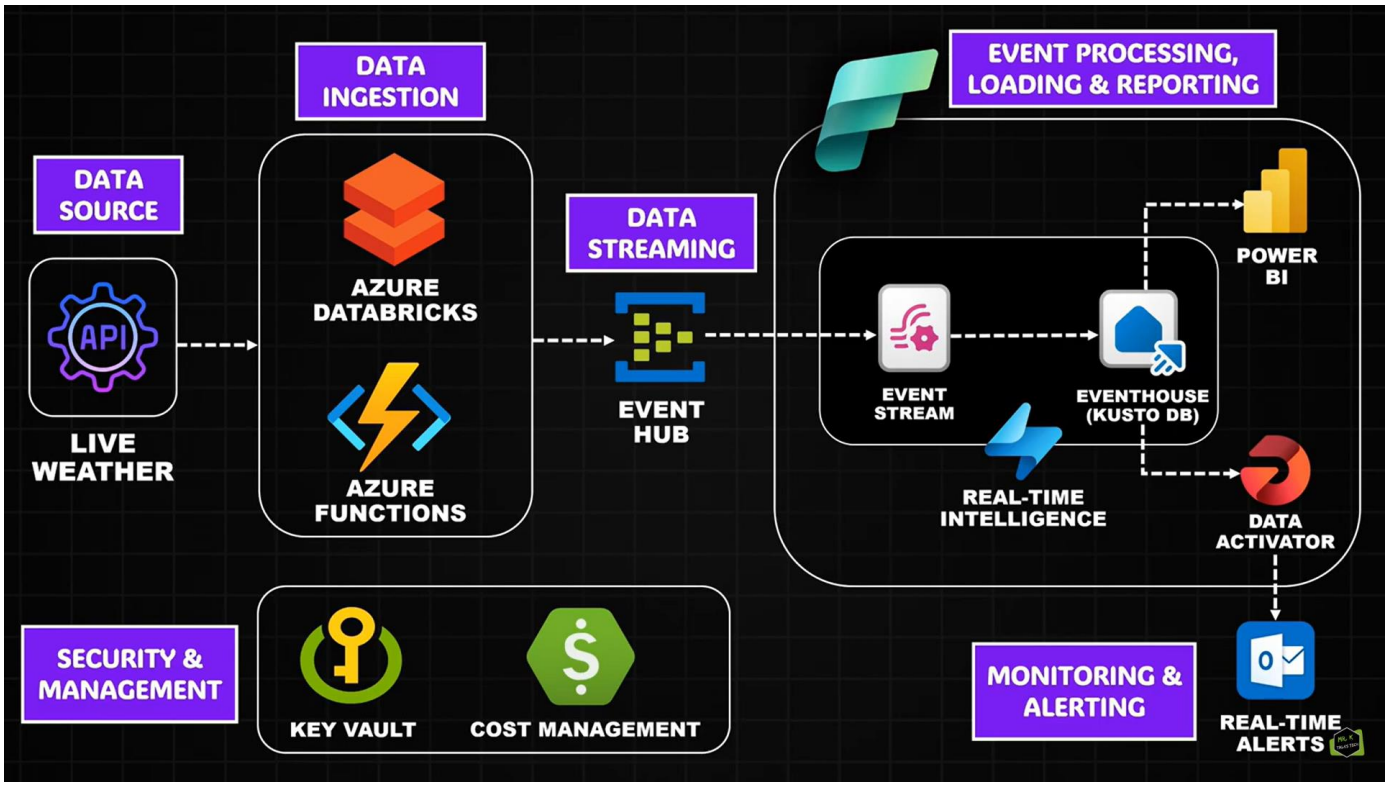
Now let’s go to resource group--->Function app--->lets run function App

This app connects to weather API data and stream to event HUB.

And event trigger pipeline created in fabric which connects to the event hub and get the latest data and loaded in KQL table in real time.

And the count is updated to





In this we configured the alert mechanism so basically we implemented a query logic in the KQL query set and then with this query set we configure the alerts using a tool called Data activator this setup allows a real-time detection of weather warnings and triggers email notification whenever such alerts are identified also we covered the security and the management aspects of this end to end project specifically we utilized assure key vaults to securely store all credentials additionally.

we demonstrated how multiple resources such as data brakes and the function app can connect to the keyboard and retrieve secret values securely following this approach aligns with the best practices for data engineers in terms of cost management we retrieve the total expenditure for each resources involved in building this end to end project specifically we analyze the cost associated with Azure data bricks and the function app based on these insights we made architectural decisions to identify the most cost effective and efficient tool for this project which was a able exercise so overall this project represents a complete streaming solution that covers essential aspects for every data engineer should understand when working on a real-time data engineering project I hope this project has been valuable and that everyone was able to follow along and successfully implemented.