




Big data processing pipeline

Part A: Data processing pipeline setup

Before creating an Azure data pipeline, it is important to deploy a custom template within the resource group, which give 4 deployed resources as below:

<input type="checkbox"/>	 ComputerVision	Computer vision	Australia East
<input type="checkbox"/>	 cosmosdb-nclaf	Azure Cosmos DB account	Australia East
<input type="checkbox"/>	 DataFactory-nclaf	Data factory (V2)	Australia East
<input type="checkbox"/>	 datastoragenclaf	Storage account	Australia East

In the storage account, 2 containers (i.e. 'sensordata' and 'sensorinsights') need to be created with storing CSV files of data from IoT sensors and the extracted metrics respectively. It is important that the access level of 'sensordata' is Blob to allow individual blobs to be read.

A CosmosDB database need to be set up (i.e. 'sensor' database) for storing structured data being extracted to the 'sensorinsights' container, and its configuration is as below. The partition key is set to be '/location' to represent how the records for air quality, traffic and energy usage are stored, helping to group the data related to each location.

New Container

*** Database id** ?

☒ Create new ☐ Use existing

☒ Share throughput across containers ?

*** Database throughput (autoscale)** ?

☒ Autoscale ☐ Manual

Estimate your required RU/s with [capacity calculator](#).

Database Max RU/s ?

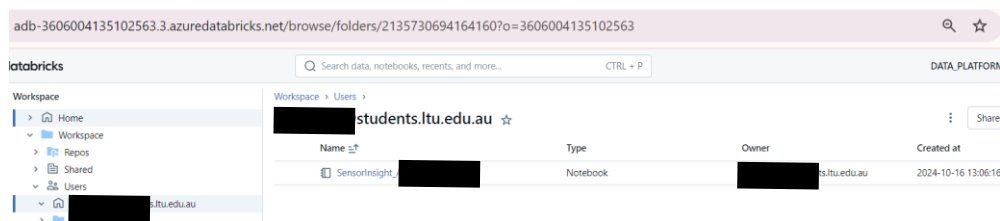
Your database throughput will automatically scale from **100 RU/s** (10% of max RU/s) - **1000 RU/s** based on usage.

Estimated monthly cost (USD) ?: **\$8.76 - \$87.60** (1 region, 100 - 1000 RU/s, \$0.00012/RU)

*** Container id** ?

*** Partition key** ?

Databricks notebook is executed from an Azure Data Factory pipeline to firstly read the CSV files from 'sensordata', calculate the relevant metrics, and return the structured data back to 'sensorinsights' and CosmosDB for storage. The notebook is located in Databricks intranet through a shared cluster, and to perform the required task of calculating metrics by different regions, the most important code chunk is captured as below.



The first function connects and retrieves the blobs being stored in 'sensordata' container, and the second function is used to load data from CSV files, while ensuring that the first row is treated as headers. PrintSchema() serves to validate if the new data aligns with the expected format and structure. After having a look into the structure of the CSV files, the third function is used to identify specific columns like timestamp, location, etc., and group data to calculate the average metrics. This means that any new CSV files uploaded to the 'sensordata' container must maintain a similar structure and format to ensure the functionality of the existing code. The last function

serves to save processed data into 'sensorinsights' container and the use of JSON file is suitable for storing structured data in CosmosDB, which can be explained in later steps.

```

3: Define functions to load, process and create metrics
Python

# Define list files in a container
def list_files(container_name):
    return blob_service_client.get_container_client(container_name).list_blobs()

# Load data from a file path
def load_data(file_path):
    df = spark.read.format("csv").option("header", "true").load(file_path)
    df.printSchema()
    return df

# Take numeric columns for calculations
def aggregate_metrics(df):
    # Identify numeric columns and calculate
    numeric_columns = [col for col, dtype in df.dtypes
                        if col not in ("sensor_id", "timestamp", "location")]
    for col in numeric_columns:
        df = df.withColumn(col, df[col].cast("float"))
    avg_columns = [avg(col).alias(f"avg_{col}") for col in numeric_columns]




    # Group by 'location' and 'timestamp' and calculate averages
    aggregated_df = df.groupBy("location", "timestamp").agg(*avg_columns)

    # Retain 'sensor_id', 'timestamp', and 'location' columns
    final_df = aggregated_df.select("location", "timestamp", *[f"avg_{col}" for col in numeric_columns])
    return final_df

# Save to Azure Blob Storage
def save_to_blob(data, blob_name):
    json_data = [json.loads(item) for item in data]
    blob_client = blob_service_client.get_blob_client(container=output_container_name, blob=blob_name)
    blob_client.upload_blob(json.dumps(json_data, indent=2), overwrite=True)

```

After running Databricks notebook, these JSON files are created in 'sensorinsights' container. This notebook is then attached to a shared cluster.

Name	Modified	Access tier	Archive status	Blob type	Size
 avg_Melbourne_Air_Quality_Data.json	10/16/2024, 5:12:35 PM	Hot (Inferred)		Block blob	844 B
 avg_Melbourne_Energy_Consumption_Data.json	10/16/2024, 5:12:36 PM	Hot (Inferred)		Block blob	772 B
 avg_Melbourne_Traffic_Data.json	10/16/2024, 5:12:37 PM	Hot (Inferred)		Block blob	732 B

As shown below, three key linked services are created in Azure Data Factory to design an end-to-end pipeline for storing IoT sensor real-time data (AzureBlobStorage), copying and processing data with Databricks notebook (AzureDatabricks) and storing the processed and aggregated metrics in CosmosDB (CosmosDbNoSql).

Linked services




Linked service defines the connection information to a data store or compute. [Learn more](#)

+ New

Filter by name

Annotations : Any

Showing 1 - 3 of 3 items

Name ↑↓	Type ↑↓	Related ↑↓
 AzureBlobStorage	Azure Blob Storage	2
 AzureDatabricks	Azure Databricks	1
 CosmosDbNoSql	Azure Cosmos DB for NoSQL	1

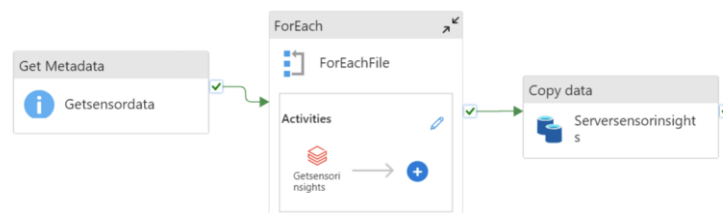
Three key datasets are also required in this data pipeline, including the raw data from IoT sensors (DataStorage_Sensordata in CSV format), processed data after the use of Databricks notebook (DataStorage_sensorinsights in JSON format) and ready-to-analyse structured data in database (CosmosDBSensor).

The first screenshot shows the configuration for a **DelimitedText** data source named **DataStorage_Sensordata**. The **Linked service** is **AzureBlobStorage**. The **File path** is **sensordata**.

The second screenshot shows the configuration for a **JSON** data source named **DataStorage_sensorinsights**. The **Linked service** is **AzureBlobStorage**. The **File path** is **sensorinsights**.

The third screenshot shows the configuration for an **Azure Cosmos DB for NoSQL** data source named **CosmosDBSensor**. The **Linked service** is **CosmosDbNoSql**. The **Container** is **sensorMetadata**, and the **Enter manually** checkbox is checked.

Creating an Azure Data Factory pipeline is the next step, with adding 'Get Metadata', 'Foreach', 'Notebook' and 'Copy Data' as the main activities, which is shown below. The first activity aims at retrieving data from the 'sensordata' container and linking them to the dataset **DataStorage_Sensordata**. The second activity will process each CSV file from the first one using Databricks notebook created. Finally, the last activity focuses on storing the processed data and metrics to the **CosmosDBSensor** collection in CosmosDB. After triggering the pipeline, there are 15 records stored in the database, which are ready for further analysis.



For the pipeline to capture real-time IoT sensor data, the trigger function is configured as below to happen every 15 minutes so that the city management to quickly up-to-date data and insights on air quality, traffic and energy consumption.

cosmosdb-nclaf | Data Explorer ☆ ...
Azure Cosmos DB account

Search

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Cost Management
- Quick start
- Data Explorer**
 - Settings
 - Integrations
 - Containers
 - Monitoring
 - Automation
 - Help

+ New Container

- Home
- sensor
 - Scale
 - sensorMetadata
 - Items
 - Settings
 - Stored Procedures
 - User Defined Functions
 - Triggers
 - Conflicts

id	location
4c68c932-ef3b-4d01-bc1e-14...	Docklands
d6625e38-48ec-4c11-9442-49...	West Gate Freeway
7091ba11-e5b2-4945-af58-bf...	Richmond
14d8d4b9-e555-4774-8635-c...	Industrial - Laverton
38992a43-bc7b-4ce5-b295-4...	St Kilda
f9f3e09-9e80-438b-8e65-be2...	Monash Freeway
58aa3265-01e6-4096-a53d-5...	Footscray
8611c0ed-8547-4ef2-880c-37...	St Kilda Road
e55af1a1-a09d-4968-bac6-10...	Eastern Freeway
5c3487cc-5d8d-4804-bbc6-8f...	Commercial - Melbourne CBD
81ffa4da-1a53-4d8c-b3be-a4...	Residential - South Yarra
213e6b92-4a34-42a9-b152-4...	Melbourne CBD
c5ca7dc5-0e85-4aa2-8da8-99...	Melbourne CBD
01126a6d-7e9d-47a8-932a-c...	Fitzroy
9f898919-0478-4ef7-bc23-01...	Essendon

New trigger

Description

Type *
Schedule

Start date * ⓘ
10/16/2024, 6:24:55 AM

Time zone * ⓘ
Canberra, Melbourne, Sydney (UTC+10)
 ⓘ This time zone observes daylight savings. Trigger will auto-adjust for one hour difference.

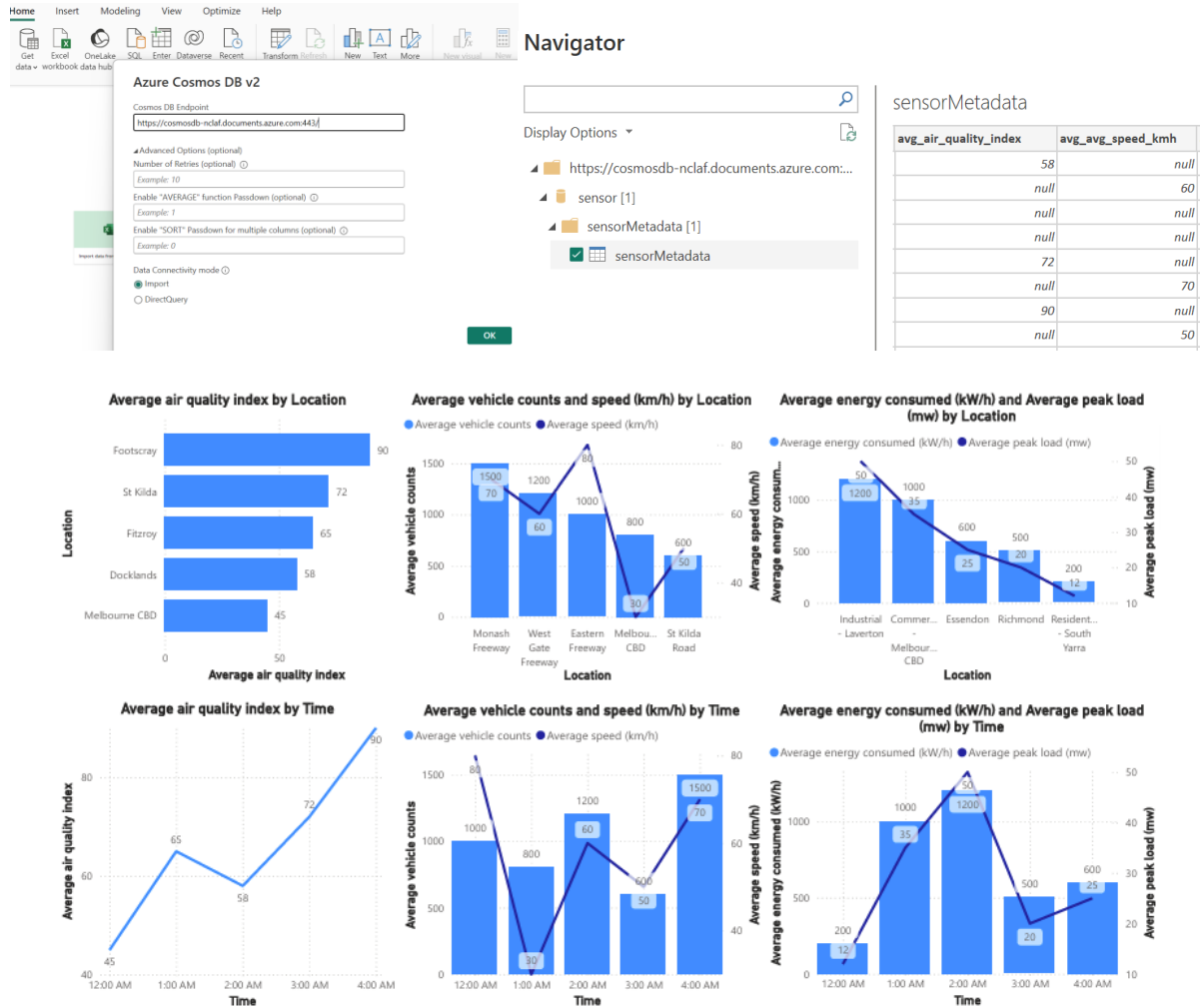
Recurrence * ⓘ
Every 15 Minute(s)
☒ Specify an end date

End On * ⓘ
11/30/2024, 6:24:55 AM

OK Cancel

Part B: Insights visualization

To visualize key insights in Power BI, the CosmoDB can be used as a data source for that BI report through inserting its URI into the Azure Cosmo BD panel of Power BI. The connected data can be transformed to perform relevant analysis, as shown below.



The use cases of this dashboard can be widely applied by city planners to improve the urban living conditions and optimize the energy usage, including four main aspects:

- A monitoring track of air quality by locations: the average air quality index by locations over time allows the city planners to set a standard threshold, based on which they identify areas with poor air quality and send pollution alerts to residents surrounding or take corrective actions as necessary. As more data points enter the pipeline, city planners can also track the air quality index by dates and times during the day, identifying certain hours to issue relevant alerts.
- An analysis on traffic flows and peak hours: the insight on average vehicle counts and average speed time by locations facilitate the city planners identify high traffics and vehicle velocity, based on which management actions on traffic signals, road rules, etc. can help alleviate congestion. Additionally, tracking the traffic flows on a time-based manner allows the allocation of roadway police to ensure the relevant safety and smooth flows of traffic.

- A visualization of energy consumption by locations and time: the average amount of energy consumption and the peak load by locations highlights the areas consuming most energy, allowing the city planners to take initiatives on promoting energy efficiency and savings, or shifting the focus to renewable sources. The average energy consumption and peak load by times during the day unveil the peak hours of usage, where they can promote energy-saving programs or ensure the energy workload meets the expected demand.
- Short-term and long-term policy planning: the overall dashboard allows a more data-driven approach for city planners to monitor the living standards of different areas and tailor their actions in both short and long run to support the overall community, such as initiatives on energy savings, pollution control and traffic safety.