

Dokumentation av Labb 2

Strömmande data med Event Hubs och Logic Apps

ARoos, Strömmande data och molnlösningar (BI24 Nackademin), 2025-10-10

Trafikverkets Rest API:

Genom POST kan vi skräddarsydda vårt anrop om vilken data behövs och om det var enklare data som skickar allt så kan GET funka också. Tillslut valde vi "TrafficFlow" för att det innehöll data som kan aggregeras och analyseras.

Event Hub:

The screenshot shows the Azure Data Explorer interface for the 'labb2-event-hub' in the 'labb2-eh-namespace'. The left sidebar has 'Data Explorer' selected. The main area displays a table of received events with the following columns: Sequence Number, Offset, Partition ID, and Enqueued Time. The data shows 13430 to 13436 events received on Friday, 10 Oct 25, between 04:53:00 pm and 04:59:00 pm CEST.

Sequence Number	Offset	Partition ID	Enqueued Time
13430	979252680096	0	Fri, 10 Oct 25, 04:53:00 pm CEST
13431	979252684504	0	Fri, 10 Oct 25, 04:54:01 pm CEST
13432	979252688912	0	Fri, 10 Oct 25, 04:55:00 pm CEST
13433	979252693320	0	Fri, 10 Oct 25, 04:56:00 pm CEST
13434	979252697720	0	Fri, 10 Oct 25, 04:57:00 pm CEST
13435	979252702128	0	Fri, 10 Oct 25, 04:58:01 pm CEST
13436	979252706536	0	Fri, 10 Oct 25, 04:59:00 pm CEST

Logic App: Valde fler steg i Logic App för enklare felsökning,

The screenshot shows the Azure Logic App designer for the 'labb2-api-post' app. The workflow consists of the following steps: Recurrence, Compose Time, Compose Body, HTTP, and Send event. The 'HTTP' step is expanded, showing its configuration details: URI is set to <https://api.trafikinfo.trafikverket.se/v2/data.json>, Method is POST, Headers include Content-Type: application/xml; charset=utf-8, and Body contains an Outputs object.

Läs event i Databricks: Fick problem med "startingPosition" och lösningen blev att ange fler variabler:

```
starting_pos = {
    "offset": None,
    "seqNo": -1,
    "enqueuedTime": t,
    "isInclusive": True
}
```

The screenshot shows a Databricks notebook interface. The notebook title is "Labb2 EventHub". The code cell contains the following Python code:

```
raw = (spark.readStream
       .format("eventhubs")
       .options(**conf)
       .load()
       )

df_min = raw.select("body")

q = (df_min.writeStream
      .format("console")
      .option("truncate","false")
      .outputMode("append")
      .start()
      )
```

Below the code cell, there is a "Raw Data" section showing metrics for a Spark job:

durationms	value
"addBatch"	137
"commitOffsets"	152
"getBatch"	8
"getOffset"	5
"queryPlanning"	10
"triggerExecution"	475
"walCommit"	161

Other sections visible include "Dashboard", "(1) Spark Jobs", and a log entry for a job ID.

Dokumentation av Labb 2 (VG)

Databricks med API och Medallion

ARoos, Strömmande data och molnlösningar (BI24 Nackademin), 2025-10-10

Cosmos DB: Denna gång hämtades data direkt i databricks från Trafikverkets API och datan temporärt lagras som:

```
df = pd.DataFrame(data["RESPONSE"]["RESULT"][0]["TrafficFlow"])
```

Efter att ha skapat en CosmosDB konfigurerades anslutningen i databricks notebook:
`client = CosmosClient(endpoint, key)`

```
database_name = "traffic"
database = client.create_database_if_not_exists(id=database_name)

container_name = "traffic"
container = database.create_container_if_not_exists(
    id=container_name,
    partition_key=PartitionKey(path="/partitionKey")
)

```

Nästa steg är att konvertera DataFrame till JSON med id och tidsstämpel
`timestamp = datetime.now().strftime("%Y%m%d %H%M%S")`

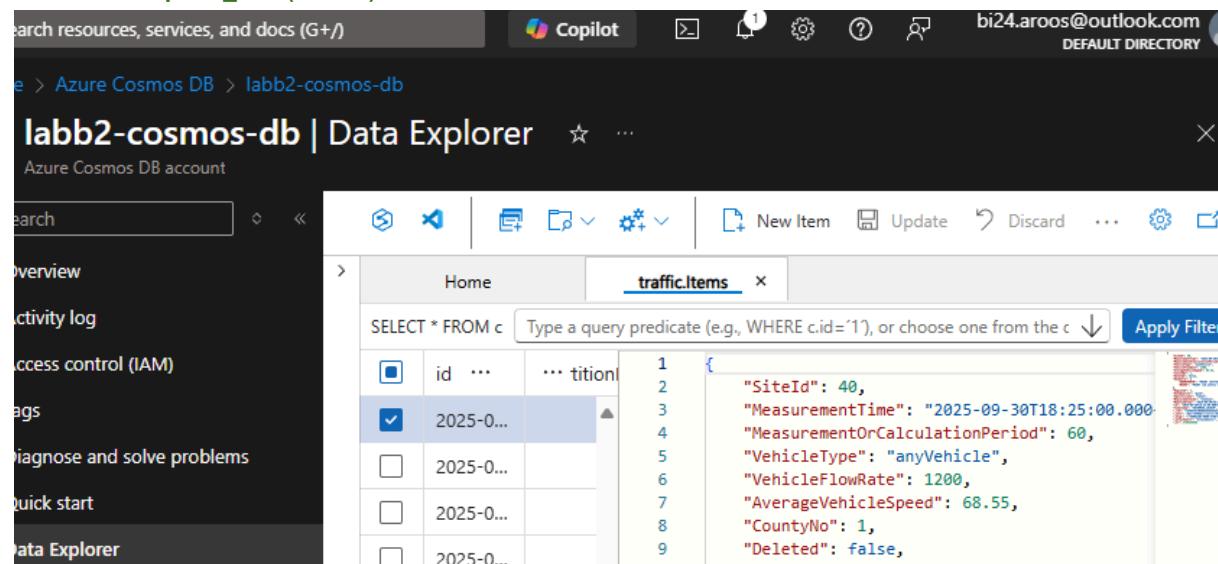
```
records = df.to_dict(orient="records")  
  
for record in records:  
    record["id"] = f"{record.get('MeasurementTime', 'unknown')}_{timestamp}_{hash(str(record))}"  
    record["timestamp"] = timestamp
```

Slutligen spara till CosmosDB:

```
container name = "traffic"
```

for record in records:

`container.upsert_item(record)`



Delta Lake – OPTIMIZE & VACUUM:

Valde att endast testa snabbt och deras funktioner är:

OPTIMIZE: `OPTIMIZE traffic_flow_gold.traffic_flow`

- Snabbar upp uppläsning av Quearies eftersom det blir mindre filer att gå igenom
- Minskar antal sparad metadata
- Bättre prestanda vid strömmande data

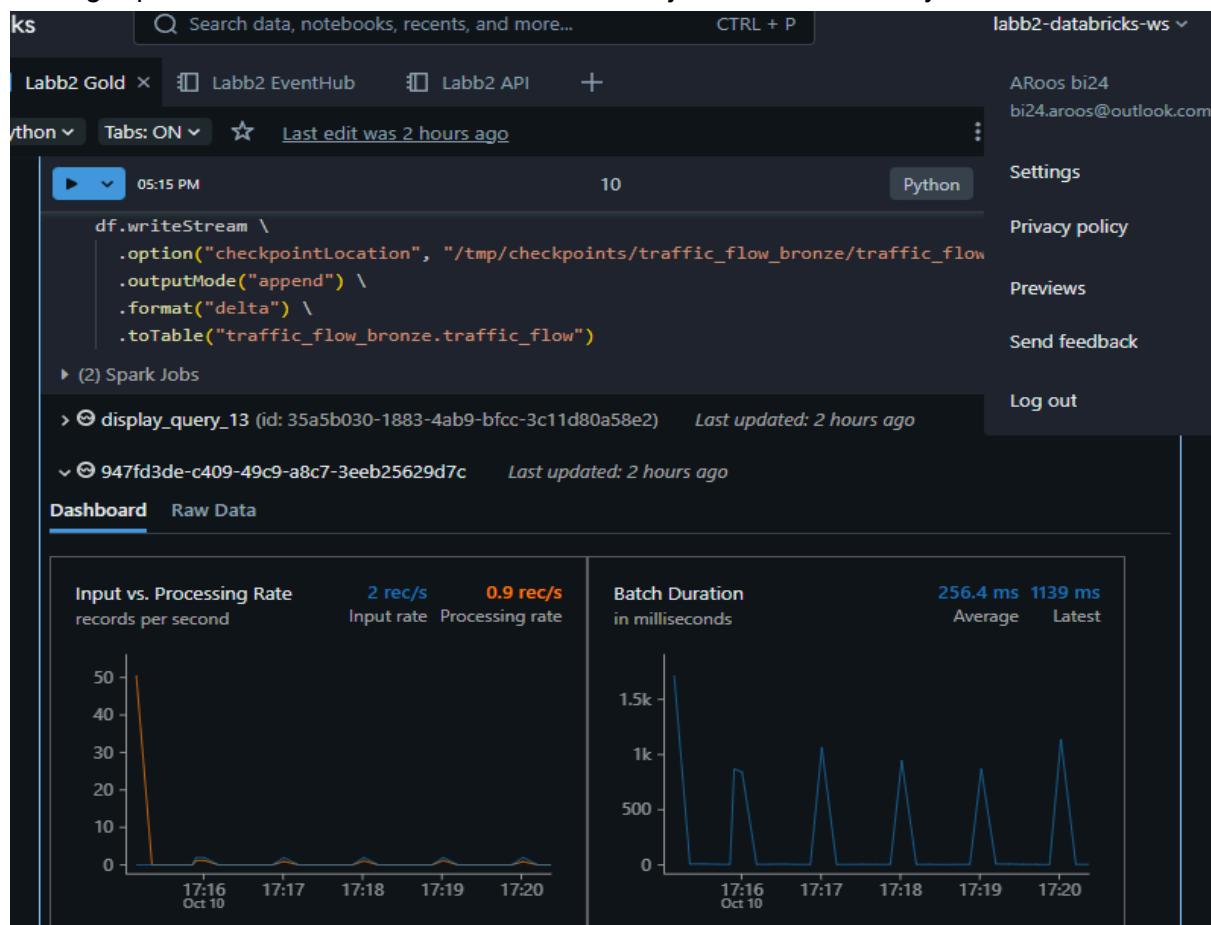
```
OPTIMIZE traffic_flow_gold.traffic_flow  
ZORDER BY (SiteId, MeasurementTime)
```

Kan förbättras med ZORDER och sammanfoga relaterade rader tillsammans på t.ex de mest använda kolumnerna.

VACUUM: `spark.sql("VACUUM traffic_flow_gold.traffic_flow_summary RETAIN 1 HOURS")`

I Delta tabell tas data inte bort på en gång vid UPDATE/DELETE och beroende på inställning i Azure kan data vara kvar upp till 7 dagar för att kunna se historiska ändringar eller återgå till tidigare tidpunkt, som VACUUM kan ta bort.

Streaming från Event Hub → Delta Table: API data uppdateras varje minut som gör att data kan användas strömmande för att visa det senaste "LIVE", där batch är en samling av en längre period som kanske en månad eller år för jämförelse och analys.



Power BI integration – realtidsdashboard: Här valde jag att koppla PowerBI till Databricks med PAT (Personal Access Token) som genereras i Databricks inställningar och uppkoppling info som finns i Databricks>Cluster inställningar, sedan välja guld tabellen.

GULD: Data är rensad från onödiga kolumner och aggregerad för lättare analys.

SILVER: Kan funka om vidare ETL och aggregering kommer göras i PowerBI.

BRONZE: Funkar inte eftersom data är i ”binär” form och behöver transformeras först för PowerBI att kunna läsa det.

Bronze:

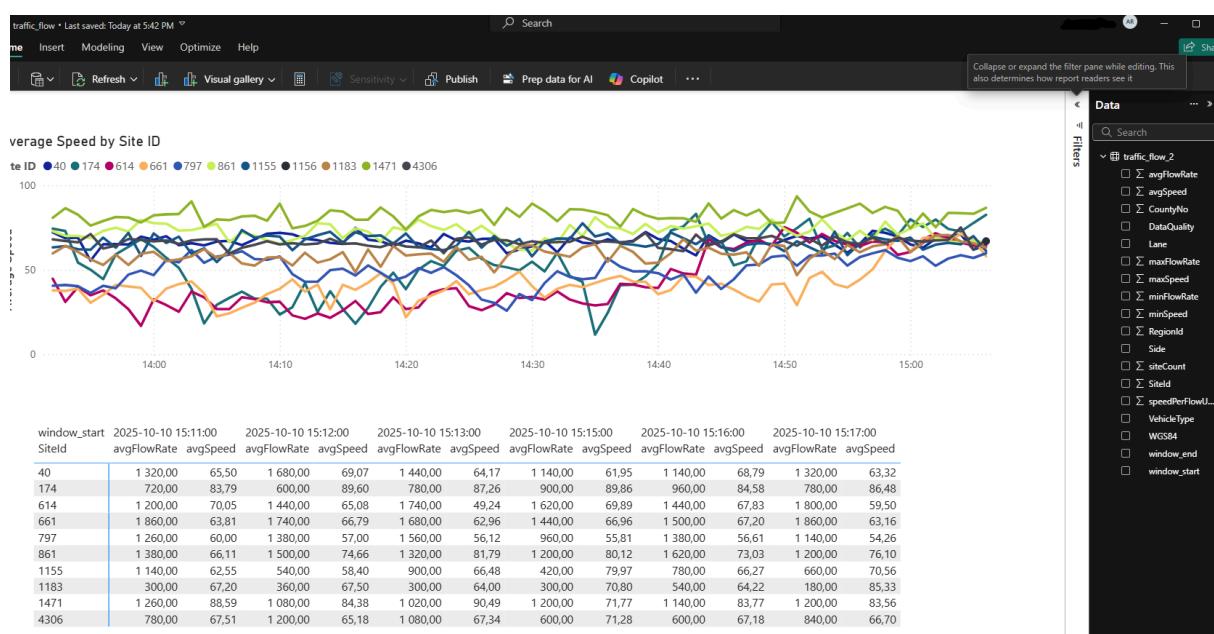
Hämta data -> lagra

Silver:

Binär -> String -> JSON Schema -> JSON med datatyper+valda kolumner -> lagra

Guld:

Aggregera data (ex. Vid tidsintervall 5 minuter) -> lagra -> ex. Analys i PowerBI



Budget: Avslutningsvis har jag följande saldo kvar.

Your remaining \$158.68 of free credit expires in 6 days.