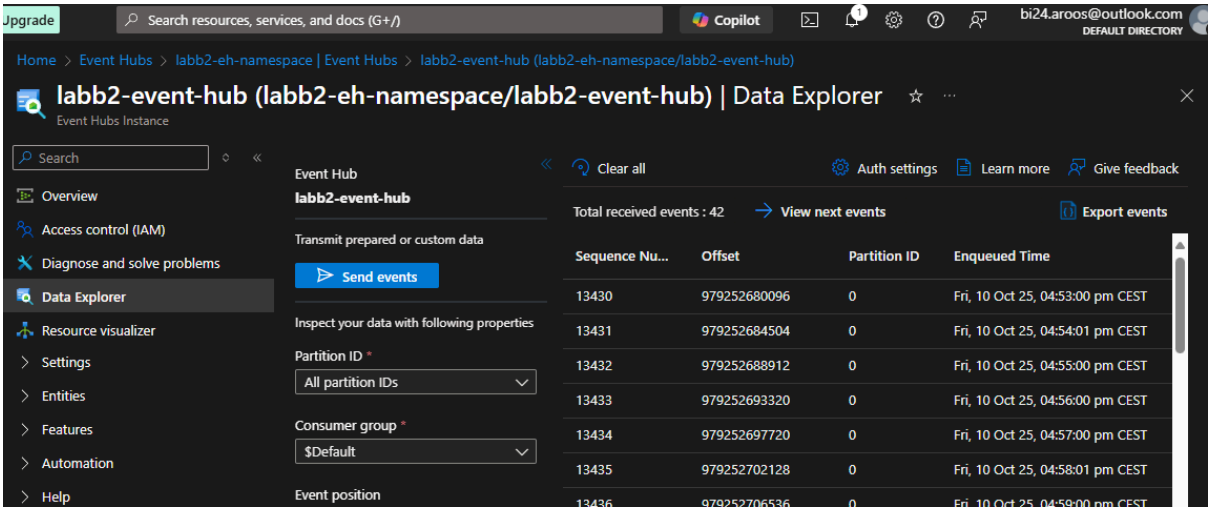
Dokumentation av Labb 2

Strömmande data med Event Hubs och Logic Apps

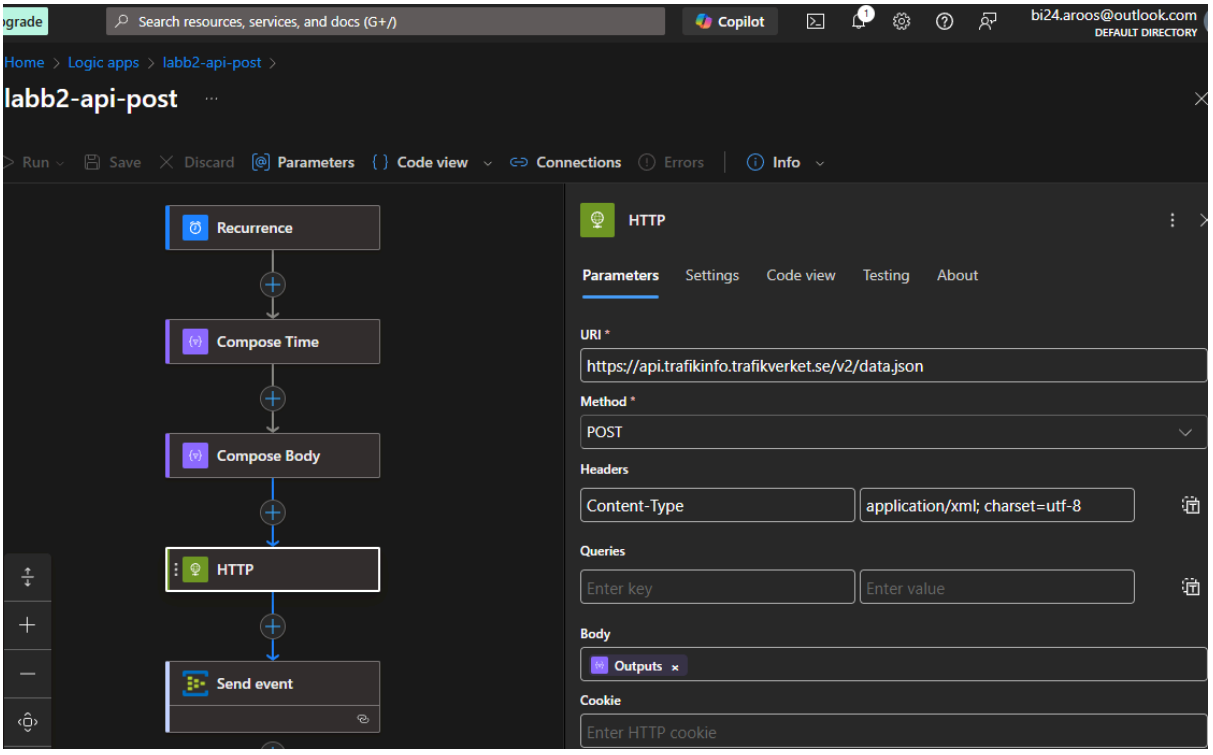
ARoos, Strömmande data och molnlösningar (BI24 Nackademin), 2025-10-10

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:

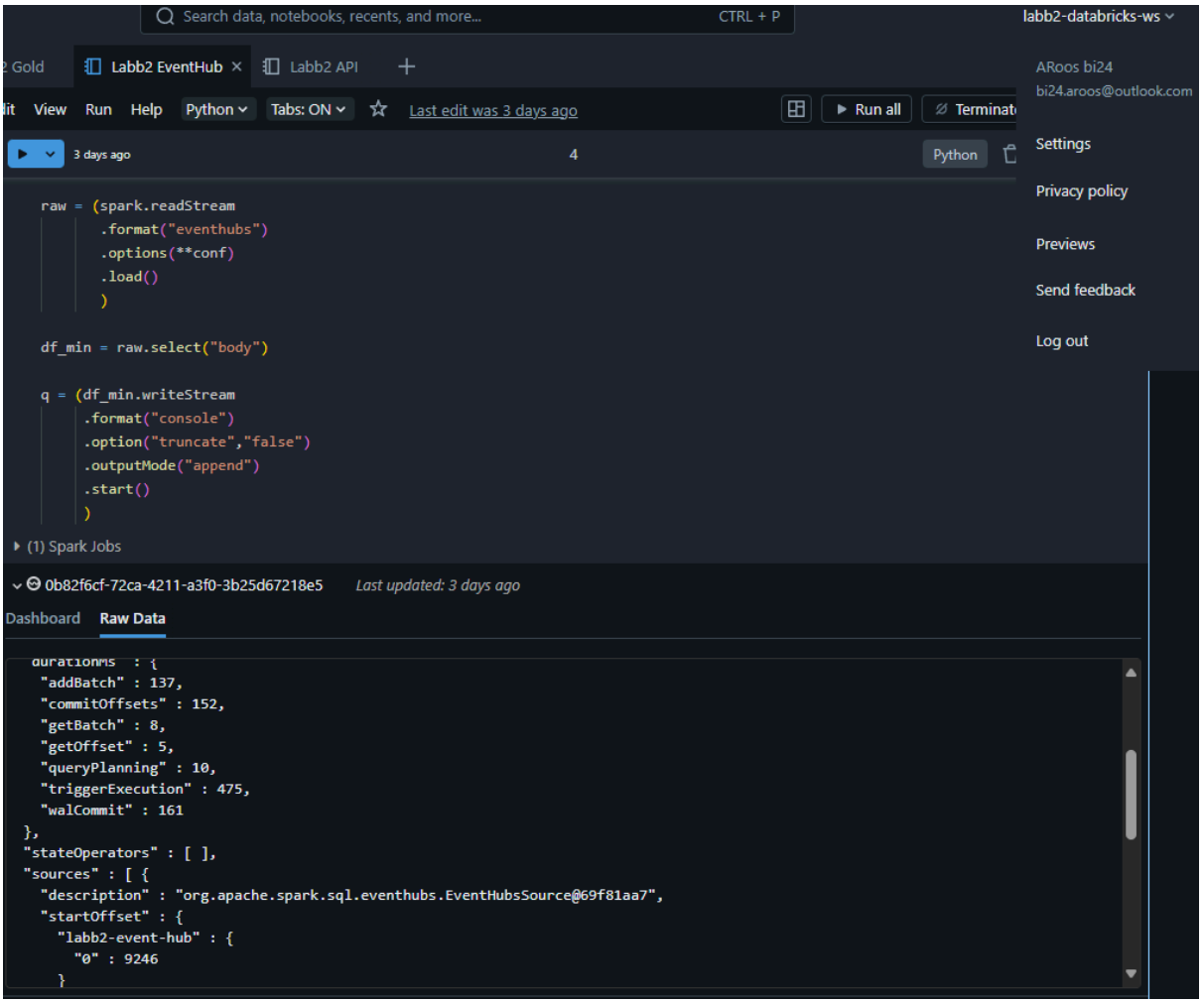


Logic App: Valde fler steg i Logic App för enklare felsökning,



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
}
```



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:

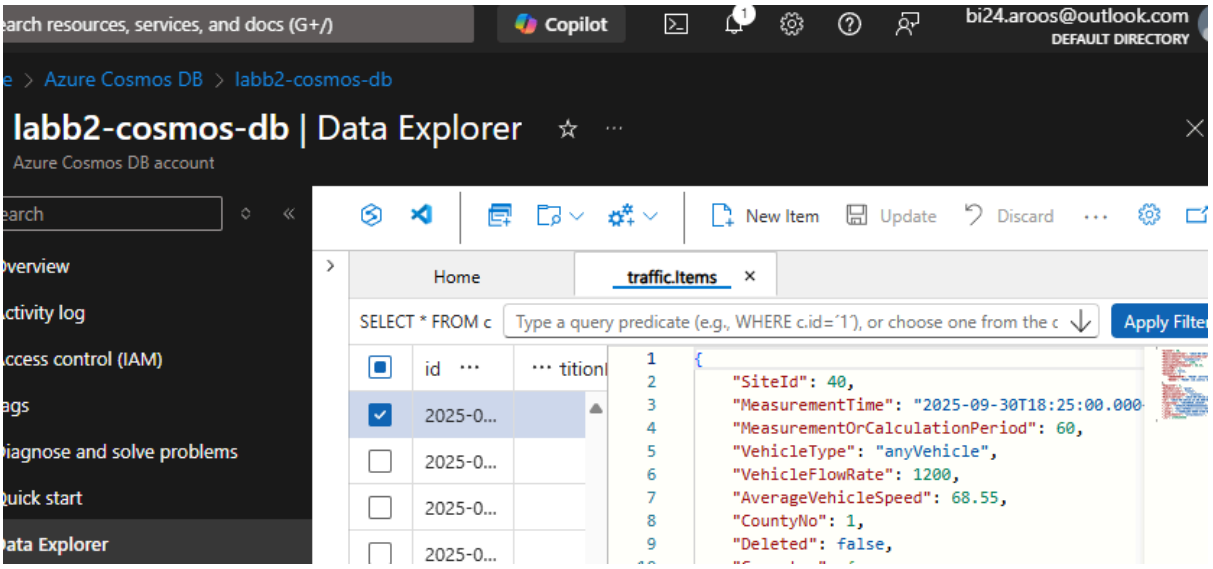
```
client = CosmosClient(endpoint, key)

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

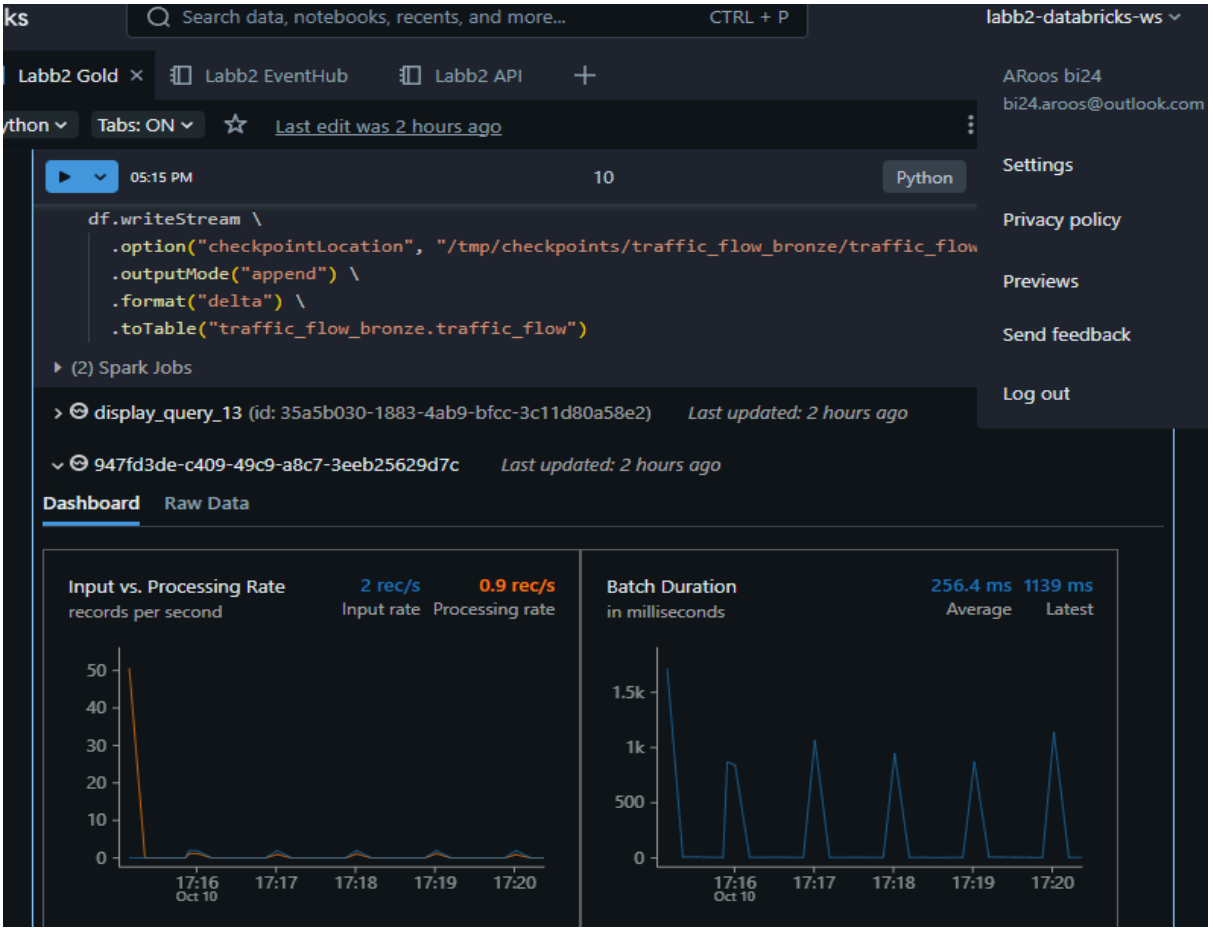
```
OPTIMIZE traffic_flow_gold.traffic_flow
ZORDER BY (SiteId, MeasurementTime
```

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.

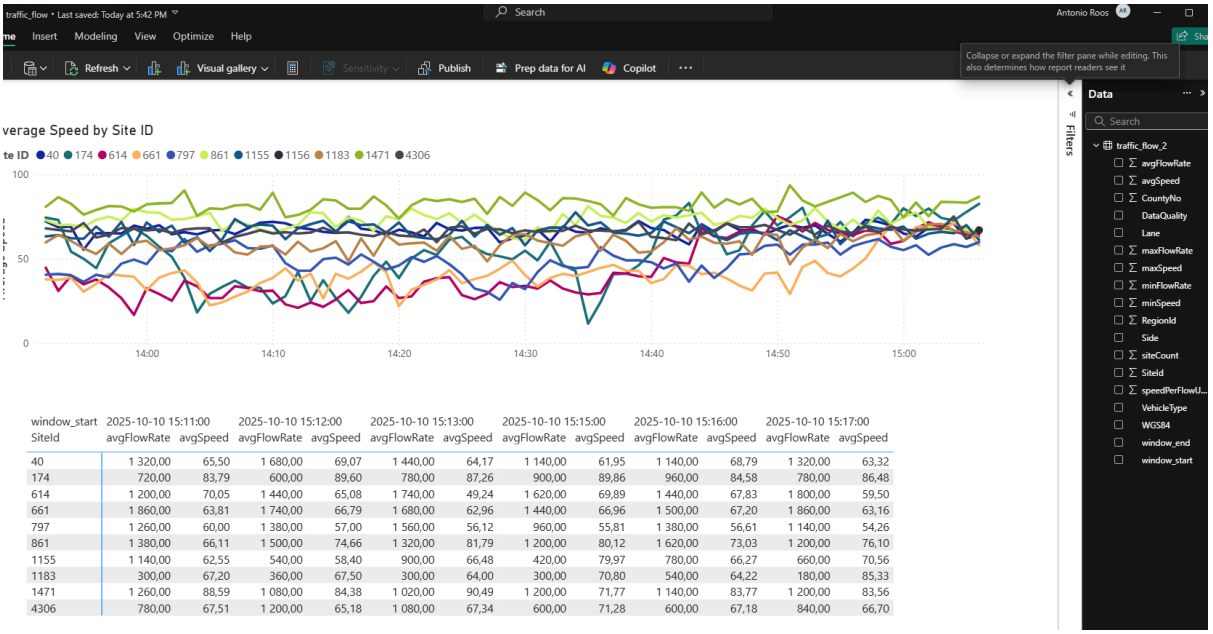
Streaming från Event Hub → Delta Table: API data uppdateras varje minut som gör att data kan användas strömmande för att visa det senaste “LIVE”, där batch är en samling av en längre period som kanske en månad eller år för jämförelse och analys.



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



window_start	2025-10-10 15:11:00		2025-10-10 15:12:00		2025-10-10 15:13:00		2025-10-10 15:15:00		2025-10-10 15:16:00		2025-10-10 15:17:00	
SiteId	avgFlowRate	avgSpeed	avgFlowRate	avgSpeed	avgFlowRate	avgSpeed	avgFlowRate	avgSpeed	avgFlowRate	avgSpeed	avgFlowRate	avgSpeed
40	1 320,00	65,50	1 680,00	69,07	1 440,00	64,17	1 140,00	61,95	1 140,00	68,79	1 320,00	63,32
174	720,00	83,79	600,00	89,60	780,00	87,26	900,00	89,86	960,00	84,58	780,00	86,48
614	1 200,00	70,05	1 440,00	65,08	1 740,00	49,24	1 620,00	69,89	1 440,00	67,83	1 800,00	59,50
661	1 860,00	63,81	1 740,00	66,79	1 680,00	62,96	1 440,00	66,96	1 500,00	67,20	1 860,00	63,16
797	1 260,00	60,00	1 380,00	57,00	1 560,00	56,12	960,00	55,81	1 380,00	56,61	1 140,00	54,26
861	1 380,00	66,11	1 500,00	74,66	1 320,00	81,79	1 200,00	80,12	1 620,00	73,03	1 200,00	76,10
1155	1 140,00	62,55	540,00	58,40	900,00	66,48	420,00	79,97	780,00	66,27	660,00	70,56
1183	300,00	67,20	360,00	67,50	300,00	64,00	300,00	70,80	540,00	64,22	180,00	85,33
1471	1 260,00	88,59	1 080,00	84,38	1 020,00	90,49	1 200,00	71,77	1 140,00	83,77	1 200,00	83,56
4306	780,00	67,51	1 200,00	65,18	1 080,00	67,34	600,00	71,28	600,00	67,18	840,00	66,70

Budget: Avslutningsvis har jag följande saldo kvar.