

Producer (ajinusa-kafka-producer.ipynb)

Dalam tugas ini diminta untuk membuat producer dengan menggunakan data purchasing. Saya mensimulasikan data penjualan obat di Apotek pada script **ajinusa-kafka-producer.ipynb** dengan menambahkan kolom "ts" untuk event timestamp. Kemudian data dikirimkan ke broker Kafka

```
ajinusa-kafka- X  ajinusa-spark- X  Dibimbing-spark- X  Dibimbing-spark- X  Dibimbing-spark- X  ajinusa-kafka- X  +
Python 3 (ipykernel)

class DataGenerator(object):
    @staticmethod
    def get_data():
        transaction_date = faker.date_this_year() # Mendapatkan tanggal transaksi
        transaction_time = faker.time() # Mendapatkan waktu transaksi
        now = datetime.now()

        # Mengonversi tanggal dan waktu menjadi string dalam format yang dapat diterima JSON
        transaction_date_str = transaction_date.strftime('%Y-%m-%d') if isinstance(transaction_date, datetime) else transaction_date
        transaction_time_str = transaction_time # Faker time() sudah berupa string (jam:menit:detik)

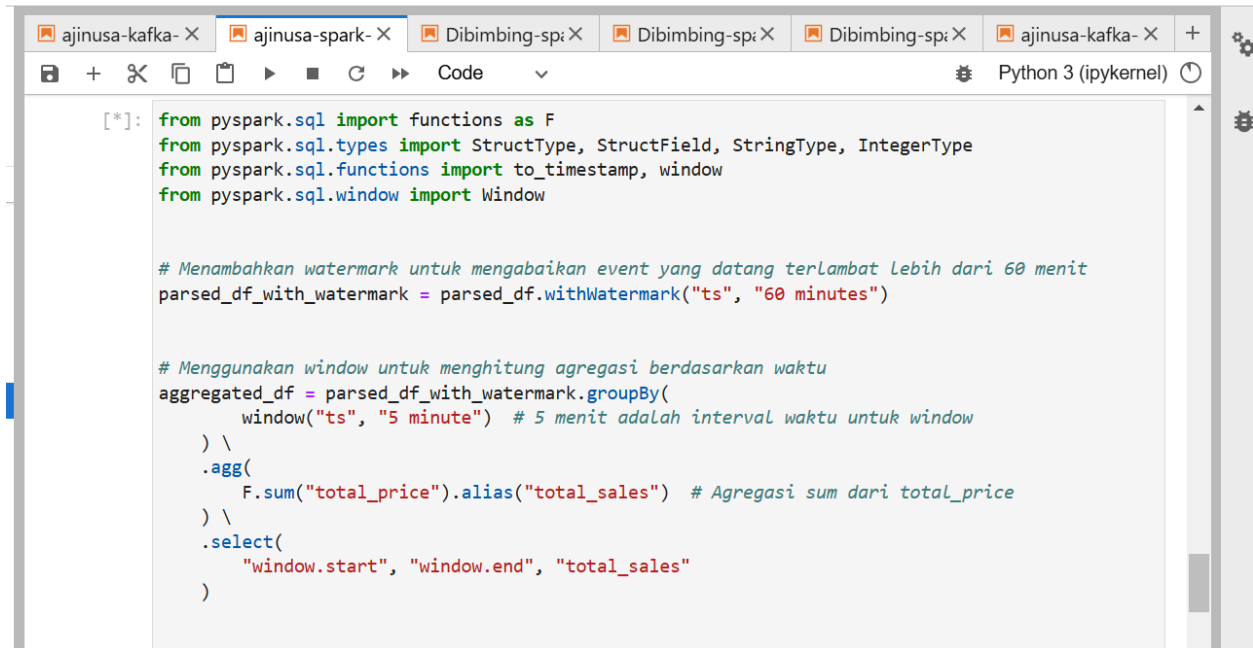
        # Data transaksi
        quantity = faker.random_int(min=1, max=5) # Jumlah obat yang dibeli
        unit_price = faker.random_int(min=1000, max=20000) # Harga per unit obat (dalam IDR)
        total_price = quantity * unit_price # Total harga transaksi

        return [
            uuid.uuid4().__str__(), # Transaction ID
            faker.name(), # Nama pembeli
            faker.random_element(elements=('Paracetamol', 'Ibuprofen', 'Amoxicillin', 'Vitamin C')), # Nama obat
            quantity, # Jumlah obat yang dibeli
            unit_price, # Harga per unit obat (dalam IDR)
            total_price, # Total harga transaksi
            faker.random_element(elements=('Cash', 'Credit Card', 'Debit Card', 'QRIS')), # Metode pembayaran
            transaction_date_str, # Tanggal transaksi sebagai string
            transaction_time_str, # Waktu transaksi sebagai string
            faker.unix_time(
                start_datetime=now - timedelta(minutes=60), end_datetime=now
            ),
        ]

# Mengirimkan data ke Kafka
for i in range(1, 500): # Jumlah data yang akan dibuat
    columns = ["transaction_id", "buyer_name", "medication_name", "quantity", "unit_price", "total_price", "payment_method", "transaction_date", "transaction_time", "timestamp"]
    data_list = DataGenerator.get_data()
    json_data = dict(zip(columns, data_list))
    _payload = json.dumps(json_data).encode("utf-8")
    response = producer.send(topic=kafka_topic_partition, value=_payload)
    print(json_data['transaction_id'], response.get()) # Menampilkan ID transaksi dan status pengiriman
    sleep(5) # Delay 5 detik antara setiap pengiriman
```

Consumer (*ajinusa-spark-consumer-stream.ipynb*)

Data dari Kafka diambil oleh consumer dengan menjumlahkan data penjualan (***total_sales***) dengan mengelompokkan berdasarkan interval event timestamp setiap 5 menit. Dengan toleransi keterlambatan data sampai dengan 60 menit menggunakan Watermark.



```
[*]: from pyspark.sql import functions as F
from pyspark.sql.types import StructType, StructField, StringType, IntegerType
from pyspark.sql.functions import to_timestamp, window
from pyspark.sql.window import Window

# Menambahkan watermark untuk mengabaikan event yang datang terlambat lebih dari 60 menit
parsed_df_with_watermark = parsed_df.withWatermark("ts", "60 minutes")

# Menggunakan window untuk menghitung agregasi berdasarkan waktu
aggregated_df = parsed_df_with_watermark.groupBy(
    window("ts", "5 minute") # 5 menit adalah interval waktu untuk window
) \
    .agg(
        F.sum("total_price").alias("total_sales") # Agregasi sum dari total_price
    ) \
    .select(
        "window.start", "window.end", "total_sales"
    )
```

```
# Tampilkan hasil di console (output mode complete)
query = (
    aggregated_df.writeStream
        .outputMode("complete") # Gunakan "complete" untuk agregasi
        .format("console") # Menampilkan hasil ke console
        .trigger(processingTime="5 minutes") # Men-trigger event setiap 5 menit
        # .option("checkpointLocation", '/resources/logs') # Lokasi checkpoint
        .option("failOnDataLoss", "false")
        .start()
)

# Menunggu stream untuk berjalan
query.awaitTermination()
```

```






    )

# Tampilkan hasil di console (output mode complete)
query = (
    aggregated_df.writeStream
        .outputMode("complete") # Gunakan "complete" untuk agregasi
        .format("console") # Menampilkan hasil ke console
        .trigger(processingTime="5 minutes") # Men-trigger event setiap 5 menit
        .option("checkpointLocation", '/resources/logs') # Lokasi checkpoint
        # .option("failOnDataLoss", "false")
        .start()
)

# Menunggu stream untuk berjalan
query.awaitTermination()

```

[Containers](#) / dibimbing-jupyter



 6dc826f03cbf  [dataeng-dibimbing/jupyter:latest](#)
[4040:4040](#)  [4041:4041](#)  [Show all ports \(3\)](#)

STATUS
 Running (4 days ago)




[Logs](#)
[Inspect](#)
[Bind mounts](#)
[Exec](#)
[Files](#)
[Stats](#)

```

2025-01-26 09:51:46 Batch: 0
2025-01-26 09:51:46 -----
2025-01-26 09:51:46 +-----+-----+-----+
2025-01-26 09:51:46 |                start|                end|total_sales|
2025-01-26 09:51:46 +-----+-----+-----+
2025-01-26 09:51:46 |2025-01-26 01:40:00|2025-01-26 01:45:00|    1605307|
2025-01-26 09:51:46 |2025-01-26 02:20:00|2025-01-26 02:25:00|    358638|
2025-01-26 09:51:46 |2025-01-26 02:35:00|2025-01-26 02:40:00|    265256|
2025-01-26 09:51:46 |2025-01-26 01:20:00|2025-01-26 01:25:00|    488028|
2025-01-26 09:51:46 |2025-01-26 01:05:00|2025-01-26 01:10:00|    196933|
2025-01-26 09:51:46 |2025-01-26 02:15:00|2025-01-26 02:20:00|    940842|
2025-01-26 09:51:46 |2025-01-26 01:10:00|2025-01-26 01:15:00|    691584|
2025-01-26 09:51:46 |2025-01-26 01:35:00|2025-01-26 01:40:00|    826766|
2025-01-26 09:51:46 |2025-01-26 01:50:00|2025-01-26 01:55:00|    1854780|
2025-01-26 09:51:46 |2025-01-26 01:55:00|2025-01-26 02:00:00|    1792242|
2025-01-26 09:51:46 |2025-01-26 02:30:00|2025-01-26 02:35:00|    592862|
2025-01-26 09:51:46 |2025-01-26 02:50:00|2025-01-26 02:55:00|     11570|
2025-01-26 09:51:46 |2025-01-26 02:40:00|2025-01-26 02:45:00|    171991|
2025-01-26 09:51:46 |2025-01-26 02:25:00|2025-01-26 02:30:00|    516113|
2025-01-26 09:51:46 |2025-01-26 01:25:00|2025-01-26 01:30:00|    952669|
    
```



RAM 7.10 GB CPU 8.76% Disk: 24.28 GB used (limit 1006.85 GB)

>_ Terminal
  v4.37.1

dibimbing-jupyter

6dc826f03cbf dataeng-dibimbing/jupyter:latest
4040:4040 4041:4041 Show all ports (3)

STATUS

Running (4 days ago)

LogsInspectBind mountsExecFilesStats

2025-01-26 09:55:01Batch: 1

2025-01-26 09:55:01-----

2025-01-26 09:55:01+-----+-----+-----+

2025-01-26 09:55:01|start|end|total_sales|

2025-01-26 09:55:01+-----+-----+-----+

2025-01-26 09:55:01|2025-01-26 02:20:00|2025-01-26 02:25:00|451638|

2025-01-26 09:55:01|2025-01-26 02:35:00|2025-01-26 02:40:00|330228|

2025-01-26 09:55:01|2025-01-26 01:20:00|2025-01-26 01:25:00|488028|

2025-01-26 09:55:01|2025-01-26 01:05:00|2025-01-26 01:10:00|196933|

2025-01-26 09:55:01|2025-01-26 01:40:00|2025-01-26 01:45:00|1605307|

2025-01-26 09:55:01|2025-01-26 02:15:00|2025-01-26 02:20:00|1148697|

2025-01-26 09:55:01|2025-01-26 01:10:00|2025-01-26 01:15:00|691584|

2025-01-26 09:55:01|2025-01-26 01:35:00|2025-01-26 01:40:00|826766|

2025-01-26 09:55:01|2025-01-26 01:50:00|2025-01-26 01:55:00|1892255|

2025-01-26 09:55:01|2025-01-26 01:55:00|2025-01-26 02:00:00|1899601|

2025-01-26 09:55:01|2025-01-26 02:50:00|2025-01-26 02:55:00|81811|

2025-01-26 09:55:01|2025-01-26 02:30:00|2025-01-26 02:35:00|730967|

2025-01-26 09:55:01|2025-01-26 02:40:00|2025-01-26 02:45:00|387575|

2025-01-26 09:55:01|2025-01-26 02:25:00|2025-01-26 02:30:00|531773|

2025-01-26 09:55:01|2025-01-26 01:25:00|2025-01-26 01:30:00|952669|

RAM 7.10 GB CPU 2.00% Disk: 24.28 GB used (limit 1006.85 GB)Terminal v4.37.1