

Orders Data Pub/Sub Using Kafka

1. In this project, I build a Kafka producer and a consumer group that worked with a MySQL database, Avro serialization, and multi-partition Kafka topics. The producer was fetching incremental data from a MySQL table and writing Avro serialized data into a Kafka topic. The consumers then deserializing this data and appending it to separate JSON files.
2. MySQL database storing product information such as product ID, name, category, price, and updated timestamp. Scenario is like I was updating the database frequently with new products and changing product information. I wanted to build a real-time system to stream these updates incrementally to a downstream system for real-time analytics.
3. Created a table named 'product' in MySQL database & initially loaded 12 records in the table 'Product' in MySQL database.

```
16 • select * from product;
```

```
17
```

	ID	name	category	price	last_updated
	4	Product 4	Category C	11.99	2023-07-08 16:13:05
	5	Product 5	Category A	12.99	2023-07-08 16:13:21
	6	Product 6	Category B	10.99	2023-07-08 16:13:36
	7	Product 7	Category C	15.99	2023-07-08 16:13:50
	8	Product 8	Category D	16.99	2023-07-08 16:14:05
	9	Product 9	Category A	16.99	2023-07-08 16:14:17
	10	Product 10	Category B	10.99	2023-07-08 16:14:44
	11	Product 11	Category B	10.99	2023-07-08 16:15:19
	12	Product 12	Category A	20.99	2023-07-08 16:15:33
	NULL	NULL	NULL	NULL	NULL

4. Wrote a Kafka producer in Python that used a MySQL connector to fetch data from the MySQL table. In my producer code, maintained a record of the last read timestamp. Each time I fetched data, used a SQL query to get records where the

last_updated timestamp is greater than the last read timestamp. Hence, records processed to producer successfully.

jupyter avro_data_producer Last Checkpoint: 6 minutes ago (autosaved)

```
File Edit View Insert Cell Kernel Widgets Help
+ %< ↺ ↻ ⬆ ⬇ ▶ Run ■ ↺ ⬆ Code
# Update the value in the config.json file
config_data['last_read_timestamp'] = max_date_str

with open('config.json', 'w') as file:
    json.dump(config_data, file)

# Close the cursor and database connection
cursor.close()
connection.close()

print("Data successfully published to Kafka")

Record b'1' successfully produced to product_updates [3] at offset 223
Record b'2' successfully produced to product_updates [7] at offset 153
Record b'3' successfully produced to product_updates [1] at offset 322
Record b'4' successfully produced to product_updates [8] at offset 158
Record b'5' successfully produced to product_updates [6] at offset 184
Record b'6' successfully produced to product_updates [0] at offset 90
Record b'7' successfully produced to product_updates [6] at offset 185
Record b'8' successfully produced to product_updates [1] at offset 323
Record b'9' successfully produced to product_updates [9] at offset 177
Record b'10' successfully produced to product_updates [1] at offset 324
Record b'11' successfully produced to product_updates [9] at offset 178
Record b'12' successfully produced to product_updates [5] at offset 87
Data successfully published to Kafka
```

In [17]:

5. Serialized the data into Avro format and published the data to a Kafka topic named "product_updates". Configured this topic with 10 partitions. Used the product ID as the key when producing messages. This ensured that all updates for the same product end up in the same partition.

```
# Create Avro Serializer for the value
key_serializer = StringSerializer('utf-8')
avro_serializer = AvroSerializer(schema_registry_client, schema_str)

# Define the SerializingProducer
producer = SerializingProducer({
    'bootstrap.servers': kafka_config['bootstrap.servers'],
    'security.protocol': kafka_config['security.protocol'],
    'sasl.mechanisms': kafka_config['sasl.mechanisms'],
    'sasl.username': kafka_config['sasl.username'],
    'sasl.password': kafka_config['sasl.password'],
    'key.serializer': key_serializer, # Key will be serialized as a string
    'value.serializer': avro_serializer # Value will be serialized as Avro
})

# Load the last read timestamp from the config file
config_data = {}

try:
    with open('config.json') as f:
        config_data = json.load(f)
        last_read_timestamp = config_data.get('last_read_timestamp')
except FileNotFoundError:
    pass

# Set a default value for last_read_timestamp
if last_read_timestamp is None:
    last_read_timestamp = '1900-01-01 00:00:00'

# Use the last_read_timestamp in the SQL query
query = "SELECT * FROM product WHERE last_updated > '{}'.format(last_read_timestamp)"

# Execute the SQL query
cursor.execute(query)

# Check if there are any rows fetched
rows = cursor.fetchall()
if not rows:
    print("No rows to fetch.")
else:
    # Iterate over the cursor and produce to Kafka
    for row in rows:
        # Get the column names from the cursor description
        columns = [column[0] for column in cursor.description]
        # Create a dictionary from the row values
        value = dict(zip(columns, row))
        # Produce to Kafka
        producer.produce(topic='product_updates', key=str(value['ID']), value=value, on_delivery=delivery_report)
        producer.flush()

# Fetch any remaining rows to consume the result
cursor.fetchall()
```

- Wrote a Kafka consumer in Python and set it up as a consumer group of 5 consumers. Each consumer read data from the "product_updates" topic. Deserialized the Avro data back into a Python object & implemented data transformation logic.

```
# Define the DeserializingConsumer
consumer = DeserializingConsumer({
    'bootstrap.servers': kafka_config['bootstrap.servers'],
    'security.protocol': kafka_config['security.protocol'],
    'saslm.echanisms': kafka_config['saslm.echanisms'],
    'saslm.username': kafka_config['saslm.username'],
    'saslm.password': kafka_config['saslm.password'],
    'key.deserializer': key_deserializer,
    'value.deserializer': avro_deserializer,
    'group.id': kafka_config['group.id'],
    'auto.offset.reset': kafka_config['auto.offset.reset'],
    'enable.auto.commit': True,
    'auto.commit.interval.ms': 5000 # Commit every 5000 ms, i.e., every 5 seconds
})

# To handle serialization of datetime objects, defining a custom encoder.
def datetime_encoder(obj):
    if isinstance(obj, datetime):
        return obj.isoformat()

# Path to the separate JSON file for each consumer
file_path = 'consumer1.json'

# Python function to load append the json string data into json file.
def write_to_json_file(json_string, file_path):
    with open(file_path, 'a') as file:
        file.write(json_string + '\n')

# Subscribe to the 'retail_data' topic
consumer.subscribe(['product_updates'])

# Continually read messages from Kafka
try:
    while True:
        msg = consumer.poll(1.0)

        if msg is None:
            continue
        if msg.error():
            print('Consumer error: {}'.format(msg.error()))
            continue

        # Change the category column to lowercase, in source it's in uppercase.
        msg.value()['category'] = msg.value()['category'].lower()

        # updating the price to half if product belongs to 'category a'
        if msg.value()['category'] == 'category a':
            msg.value()['price'] = msg.value()['price'] * 0.5
            msg.value()['price'] = round(msg.value()['price'], 2)

        print('Successfully consumed record with key {} and value {}'.format(msg.key(), msg.value()))
        json_string = json.dumps(msg.value(), default=datetime_encoder)
```

- Consumers start consuming the 12 records(screenshots attached for consumer 1 to 5 consuming all 12 records in the table initially).

Each consumer converting the transformed Python object into a JSON string and appending the JSON string to a separate JSON file. Opened the file in append mode

avro_data_consumer_1 Last Checkpoint: 15 minutes ago (unsaved changes)

```
View  Insert  Cell  Kernel  Widgets  Help

📄 📁 ⬆ ⬇ ▶ Run ⏏ ↺ ⏩ Code ▾ 🧾

# Check if the file exists
if not os.path.isfile(file_path):
    # Create the file and write the initial data
    with open(file_path, 'w') as file:
        file.write(json_string + '\n')
else:
    # Append the data to the existing file
    write_to_json_file(json_string, file_path)
    print("json_string data is added to the JSON file.")
file.close()

except KeyboardInterrupt:
    pass
finally:
    consumer.close()

Successfully consumed record with key 2 and value {'ID': 2, 'name': 'Product 2', 'category': 'Electronics', 'last_updated': datetime.datetime(2023, 7, 8, 16, 10, 36, tzinfo=datetime.timezone.utc)}
json_string data is added to the JSON file.
Successfully consumed record with key 5 and value {'ID': 5, 'name': 'Product 5', 'category': 'Electronics', 'last_updated': datetime.datetime(2023, 7, 8, 16, 13, 21, tzinfo=datetime.timezone.utc)}
json_string data is added to the JSON file.
Successfully consumed record with key 7 and value {'ID': 7, 'name': 'Product 7', 'category': 'Electronics', 'last_updated': datetime.datetime(2023, 7, 8, 16, 13, 50, tzinfo=datetime.timezone.utc)}
json_string data is added to the JSON file.
```

avro_data_consumer_2 Last Checkpoint: 16 minutes ago (autosaved)

```
View  Insert  Cell  Kernel  Widgets  Help

📄 📁 ⬆ ⬇ ▶ Run ⏏ ↺ ⏩ Code ▾ 🧾

msg.value()['price'] = round(msg.value()['price'],2)

print('Successfully consumed record with key {} and value {}'.format(key, msg.value()))
json_string = json.dumps(msg.value(), default=datetime_encoder)

def write_to_json_file(json_string, file_path):
    with open(file_path, 'a') as file:
        file.write(json_string + '\n')

# Check if the file exists
if not os.path.isfile(file_path):
    # Create the file and write the initial data
    with open(file_path, 'w') as file:
        file.write(json_string + '\n')
else:
    # Append the data to the existing file
    write_to_json_file(json_string, file_path)
    print("json_string data is added to the JSON file.")
file.close()

except KeyboardInterrupt:
    pass
finally:
    consumer.close()

Successfully consumed record with key 12 and value {'ID': 12, 'name': 'Product 12', 'category': 'Electronics', 'last_updated': datetime.datetime(2023, 7, 8, 16, 15, 33, tzinfo=datetime.timezone.utc)}
```

avro_data_consumer_3 Last Checkpoint: 16 minutes ago (autosaved)

```
View Insert Cell Kernel Widgets Help
Run Code

with open(file_path, 'a') as file:
    file.write(json_string + '\n')

# Check if the file exists
if not os.path.isfile(file_path):
    # Create the file and write the initial data
    with open(file_path, 'w') as file:
        file.write(json_string + '\n')
else:
    # Append the data to the existing file
    write_to_json_file(json_string, file_path)
    print("json_string data is added to the JSON file.")
    file.close()

except KeyboardInterrupt:
    pass
finally:
    consumer.close()

Successfully consumed record with key 4 and value {'ID': 4, 'name': 'Product 4'
1118164, 'last_updated': datetime.datetime(2023, 7, 8, 16, 13, 5, tzinfo=dateti
Successfully consumed record with key 9 and value {'ID': 9, 'name': 'Product 9'
_updated': datetime.datetime(2023, 7, 8, 16, 14, 17, tzinfo=datetime.timezone.u
json_string data is added to the JSON file.
Successfully consumed record with key 11 and value {'ID': 11, 'name': 'Product
9771118164, 'last_updated': datetime.datetime(2023, 7, 8, 16, 15, 19, tzinfo=da
json_string data is added to the JSON file.
```

avro_data_consumer_4 Last Checkpoint: 16 minutes ago (autosaved)

```
View Insert Cell Kernel Widgets Help
Run Code

msg.value()['price'] = round(msg.value()['price'], 2)

print('Successfully consumed record with key {} and value {}'.format(ms
json_string = json.dumps(msg.value(), default=datetime_encoder)

def write_to_json_file(json_string, file_path):
    with open(file_path, 'a') as file:
        file.write(json_string + '\n')

# Check if the file exists
if not os.path.isfile(file_path):
    # Create the file and write the initial data
    with open(file_path, 'w') as file:
        file.write(json_string + '\n')
else:
    # Append the data to the existing file
    write_to_json_file(json_string, file_path)
    print("json_string data is added to the JSON file.")
    file.close()

except KeyboardInterrupt:
    pass
finally:
    consumer.close()

Successfully consumed record with key 1 and value {'ID': 1, 'name': 'Product 1'
_updated': datetime.datetime(2023, 7, 8, 16, 10, 19, tzinfo=datetime.timezone.u
```

avro_data_consumer_5 Last Checkpoint: 16 minutes ago (autosaved)

```
View    Insert    Cell    Kernel    Widgets    Help

📄 📄 ⬆ ⬇ ▶ Run ⏏ ↺ ⬆ Code ⌵ 🗨

# Check if the file exists
if not os.path.isfile(file_path):
    # Create the file and write the initial data
    with open(file_path, 'w') as file:
        file.write(json_string + '\n')
else:
    # Append the data to the existing file
    write_to_json_file(json_string, file_path)
    print("json_string data is added to the JSON file.")
file.close()

except KeyboardInterrupt:
    pass
finally:
    consumer.close()
```

Successfully consumed record with key 3 and value {'ID': 3, 'name': 'Product 3', 'last_updated': datetime.datetime(2023, 7, 8, 16, 11, 1, tzinfo=datetime.timezone.utc)}

Successfully consumed record with key 6 and value {'ID': 6, 'name': 'Product 6', 'last_updated': datetime.datetime(2023, 7, 8, 16, 13, 36, tzinfo=datetime.timezone.utc)}

json_string data is added to the JSON file.

Successfully consumed record with key 8 and value {'ID': 8, 'name': 'Product 8', 'last_updated': datetime.datetime(2023, 7, 8, 16, 14, 5, tzinfo=datetime.timezone.utc)}

json_string data is added to the JSON file.

Successfully consumed record with key 10 and value {'ID': 10, 'name': 'Product 10', 'last_updated': datetime.datetime(2023, 7, 8, 16, 14, 44, tzinfo=datetime.timezone.utc)}

json_string data is added to the JSON file.

8. Again loaded 3 records in the table.

16 • `select * from product;`

17

Result Grid					
		Filter Rows:		Edit:	
ID	name	category	price	last_updated	
7	Product 7	Category C	17.99	2023-07-08 16:13:50	
8	Product 8	Category C	10.99	2023-07-08 16:14:05	
9	Product 9	Category A	16.99	2023-07-08 16:14:17	
10	Product 10	Category B	10.99	2023-07-08 16:14:44	
11	Product 11	Category B	10.99	2023-07-08 16:15:19	
12	Product 12	Category A	20.99	2023-07-08 16:15:33	
13	Product 13	Category A	10.99	2023-07-10 10:25:01	
14	Product 14	Category B	13.99	2023-07-10 10:25:20	
15	Product 15	Category D	14.99	2023-07-10 10:25:37	
* NULL	NULL	NULL	NULL	NULL	

9. 3 new record processed to producer successfully.

er avro_data_producer Last Checkpoint: 11 minutes ago (autosaved)

```
View  Insert  Cell  Kernel  Widgets  Help
:  [Icons]  [Run]  [Code]

query = "SELECT MAX(last_updated) FROM product"
cursor.execute(query)

# Fetch the result
result = cursor.fetchone()
max_date = result[0] # Assuming the result is a single value

# Convert datetime object to string representation
max_date_str = max_date.strftime("%Y-%m-%d %H:%M:%S")

# Update the value in the config.json file
config_data['last_read_timestamp'] = max_date_str

with open('config.json', 'w') as file:
    json.dump(config_data, file)

# Close the cursor and database connection
cursor.close()
connection.close()

print("Data successfully published to Kafka")

Record b'13' successfully produced to product_updates [7] at offset 154
Record b'14' successfully produced to product_updates [2] at offset 146
Record b'15' successfully produced to product_updates [8] at offset 159
Data successfully published to Kafka
```

10. All 3(with id as 13,14,15) records consumed by consumers 1,4 and 3 respectively.

r avro_data_consumer_1 Last Checkpoint: 22 minutes ago (autosaved)

```
View  Insert  Cell  Kernel  Widgets  Help
[Icons]  [Run]  [Code]

with open(file_path, 'w') as file:
    file.write(json_string + '\n')
else:
    # Append the data to the existing file
    write_to_json_file(json_string, file_path)
    print("json_string data is added to the JSON file.")
    file.close()

except KeyboardInterrupt:
    pass
finally:
    consumer.close()

Successfully consumed record with key 2 and value {'ID': 2, 'name': 'Product 2', 'category':
1118164, 'last_updated': datetime.datetime(2023, 7, 8, 16, 10, 36, tzinfo=datetime.timezone.
json_string data is added to the JSON file.
Successfully consumed record with key 5 and value {'ID': 5, 'name': 'Product 5', 'category':
_updated': datetime.datetime(2023, 7, 8, 16, 13, 21, tzinfo=datetime.timezone.utc)}
json_string data is added to the JSON file.
Successfully consumed record with key 7 and value {'ID': 7, 'name': 'Product 7', 'category':
1118164, 'last_updated': datetime.datetime(2023, 7, 8, 16, 13, 50, tzinfo=datetime.timezone.
json_string data is added to the JSON file.
Successfully consumed record with key 13 and value {'ID': 13, 'name': 'Product 13', 'category':
ast_updated': datetime.datetime(2023, 7, 10, 10, 25, 1, tzinfo=datetime.timezone.utc)}
json_string data is added to the JSON file.
```


avro_data_consumer_4 Last Checkpoint: 21 minutes ago (autosaved)

```
View  Insert  Cell  Kernel  Widgets  Help

Copy Paste Undo Redo Run Stop Restart Code Console

msg.value()['price'] = round(msg.value()['price'],2)

print('Successfully consumed record with key {} and value {}'.format(msg.key(), msg.value()))
json_string = json.dumps(msg.value(), default=datetime_encoder)

def write_to_json_file(json_string, file_path):
    with open(file_path, 'a') as file:
        file.write(json_string + '\n')

    # Check if the file exists
    if not os.path.isfile(file_path):
        # Create the file and write the initial data
        with open(file_path, 'w') as file:
            file.write(json_string + '\n')
    else:
        # Append the data to the existing file
        write_to_json_file(json_string, file_path)
    print("json_string data is added to the JSON file.")
    file.close()

except KeyboardInterrupt:
    pass
finally:
    consumer.close()

Successfully consumed record with key 1 and value {'ID': 1, 'name': 'Product 1', 'category': 'category
_updated': datetime.datetime(2023, 7, 8, 16, 10, 19, tzinfo=datetime.timezone.utc)}
Successfully consumed record with key 14 and value {'ID': 14, 'name': 'Product 14', 'category': 'categ
9771118164, 'last_updated': datetime.datetime(2023, 7, 10, 10, 25, 20, tzinfo=datetime.timezone.utc)}
json_string data is added to the JSON file.
```

avro_data_consumer_3 Last Checkpoint: 22 minutes ago (autosaved)

```
View  Insert  Cell  Kernel  Widgets  Help

Copy Paste Undo Redo Run Stop Restart Code Console

    # Check if the file exists
    if not os.path.isfile(file_path):
        # Create the file and write the initial data
        with open(file_path, 'w') as file:
            file.write(json_string + '\n')
    else:
        # Append the data to the existing file
        write_to_json_file(json_string, file_path)
    print("json_string data is added to the JSON file.")
    file.close()

except KeyboardInterrupt:
    pass
finally:
    consumer.close()

Successfully consumed record with key 4 and value {'ID': 4, 'name': 'Product 4', 'category': 'category
1118164, 'last_updated': datetime.datetime(2023, 7, 8, 16, 13, 5, tzinfo=datetime.timezone.utc)}
Successfully consumed record with key 9 and value {'ID': 9, 'name': 'Product 9', 'category': 'category
_updated': datetime.datetime(2023, 7, 8, 16, 14, 17, tzinfo=datetime.timezone.utc)}
json_string data is added to the JSON file.
Successfully consumed record with key 11 and value {'ID': 11, 'name': 'Product 11', 'category': 'categ
9771118164, 'last_updated': datetime.datetime(2023, 7, 8, 16, 15, 19, tzinfo=datetime.timezone.utc)}
json_string data is added to the JSON file.
Successfully consumed record with key 15 and value {'ID': 15, 'name': 'Product 15', 'category': 'categ
9771118164, 'last_updated': datetime.datetime(2023, 7, 10, 10, 25, 37, tzinfo=datetime.timezone.utc)}
json_string data is added to the JSON file.
```