

PROGRAMMING ASSIGNMENT № 2

Evangelia Panourgia, Athens University of Economics and Business University

27/03/2025

Part I

The python script for the first part.

The provided Python script is an asynchronous Kafka producer designed to simulate real-time movie rating events. It uses the Faker library to generate synthetic user names and reads actual movie titles from a CSV file using pandas. Each generated user rates a randomly selected movie with a random rating between 1 and 10, along with a current timestamp. These events are serialized to JSON and sent to a Kafka topic named "test" using the aiokafka library. The process runs continuously, with each user producing one rating per minute, making it ideal for streaming applications such as those built with Apache Spark Structured Streaming. Additionally, the script appends a specific user name, "Evangelia Panourgia", to ensure consistent personal data inclusion for testing or filtering.

Listing 1: python-kafka-example

```
1  """
2  Async Kafka Producer: Fake Movie Ratings Generator
3
4  Overview:
5  -----
6  This script continuously generates **fake movie rating events** and sends ↵
   them to a Kafka topic ("test").
7  It simulates users rating random movies with timestamps, mimicking real-↵
   time data.
8
9  Components:
10 -----
11 1. Faker: Generates fake human names.
12 2. Pandas: Loads real movie titles from a CSV file.
13 3. aiokafka: Asynchronous Kafka producer library.
14 4. Kafka: Acts as the message broker for event streaming.
15
16 Use Case:
17 -----
18 Run this script to simulate live rating data for a Spark Structured ↵
   Streaming application to consume and process.
19 """
```

```

20 import json
21 import asyncio
22 import random
23 from datetime import datetime
24
25 import pandas as pd
26 from faker import Faker
27 from aiokafka import AIOKafkaProducer
28
29 # Load movie titles
30 movies_df = pd.read_csv('data/movies.csv', header=None, names=["↵
    movie_title"])
31 movie_titles = movies_df['movie_title'].dropna().tolist()
32
33 # Generate names
34 fake = Faker()
35 names = [fake.name() for _ in range(10)]
36 names.append("Evangelia Panourgia") # add my own name
37
38 # Kafka settings
39 KAFKA_TOPIC = "test"
40 KAFKA_BOOTSTRAP_SERVERS = "localhost:29092"
41
42 # JSON serializer
43 def serializer(data):
44     return json.dumps(data).encode()
45
46 # Producer logic
47 async def produce():
48     producer = AIOKafkaProducer(
49         bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS,
50         value_serializer=serializer,
51         compression_type="gzip"
52     )
53
54     await producer.start()
55     try:
56         while True:
57             for name in names:
58                 movie = random.choice(movie_titles)
59                 rating = random.randint(1, 10)
60                 timestamp = datetime.now().isoformat()
61
62                 message = {
63                     "name": name,
64                     "movie": movie,
65                     "timestamp": timestamp,

```

```

66         "rating": rating
67     }
68
69     print(f"Sending: {message}")
70     await producer.send(KAFKA_TOPIC, message)
71     await asyncio.sleep(60) # One message per user per minute
72 finally:
73     await producer.stop()
74
75 # Run it (no deprecation warning)
76 if __name__ == "__main__":
77     loop = asyncio.new_event_loop()
78     asyncio.set_event_loop(loop)
79     loop.run_until_complete(produce())

```

The image below depicts the output of kafka producer running in the first terminal.

```

(venv) venv\scripts\python3 examples\python-kafka-example.py
Sending: {'name': 'Paul Johnson', 'movie': 'Gurjan Saxena: The Kargil Girl', 'timestamp': '2025-03-27T09:24:09.168684', 'rating': 5}
Topic test is not available during auto-create initialization
Topic test is not available during auto-create initialization
Sending: {'name': 'Teresa Burns', 'movie': 'Legally Blonde', 'timestamp': '2025-03-27T09:25:09.479858', 'rating': 3}
Sending: {'name': 'Stephanie Carney', 'movie': 'Hometown Holiday', 'timestamp': '2025-03-27T09:26:09.540822', 'rating': 7}
Sending: {'name': 'Sandra Rhodes', 'movie': 'Berlin Calling', 'timestamp': '2025-03-27T09:27:09.600886', 'rating': 1}
Sending: {'name': 'April Henry', 'movie': 'Bombairiya', 'timestamp': '2025-03-27T09:28:09.661411', 'rating': 7}
Sending: {'name': 'Randy Sanders', 'movie': 'Stretch Armstrong: The Breakout', 'timestamp': '2025-03-27T09:29:09.662108', 'rating': 7}
Sending: {'name': 'Richard Garcia', 'movie': 'Jackie Brown', 'timestamp': '2025-03-27T09:30:09.717081', 'rating': 8}
Sending: {'name': 'Pamela Wilson', 'movie': 'Good Burger', 'timestamp': '2025-03-27T09:31:09.775918', 'rating': 4}
Sending: {'name': 'Erika Brown', 'movie': 'Lone House', 'timestamp': '2025-03-27T09:32:09.830701', 'rating': 6}
Sending: {'name': 'Jimmy Strickland', 'movie': 'The Birth Reborn', 'timestamp': '2025-03-27T09:33:09.831783', 'rating': 5}
Sending: {'name': 'Evangelia Panourgia', 'movie': 'The Girl King', 'timestamp': '2025-03-27T09:34:09.833471', 'rating': 9}
Sending: {'name': 'Paul Johnson', 'movie': 'Happilycious', 'timestamp': '2025-03-27T09:35:09.891655', 'rating': 9}
Sending: {'name': 'Teresa Burns', 'movie': 'No Escape Room', 'timestamp': '2025-03-27T09:36:09.951825', 'rating': 1}
Sending: {'name': 'Stephanie Carney', 'movie': 'The Stolen', 'timestamp': '2025-03-27T09:37:10.012058', 'rating': 2}
Sending: {'name': 'Sandra Rhodes', 'movie': 'Rory Scovel Tries Standup for the First Time', 'timestamp': '2025-03-27T09:38:10.072500', 'rating': 2}
Sending: {'name': 'April Henry', 'movie': 'The Water Horse: Legend of the Deep', 'timestamp': '2025-03-27T09:39:10.073160', 'rating': 10}
Sending: {'name': 'Randy Sanders', 'movie': 'Jack Whitehall: At Large', 'timestamp': '2025-03-27T09:40:10.122963', 'rating': 1}
Sending: {'name': 'Richard Garcia', 'movie': 'Boogie Nights', 'timestamp': '2025-03-27T09:41:10.133020', 'rating': 4}
Sending: {'name': 'Pamela Wilson', 'movie': 'Road to Sangam', 'timestamp': '2025-03-27T09:42:10.179353', 'rating': 10}
Sending: {'name': 'Erika Brown', 'movie': 'Don't Look Down', 'timestamp': '2025-03-27T09:43:10.230256', 'rating': 9}
Sending: {'name': 'Jimmy Strickland', 'movie': 'Revelations', 'timestamp': '2025-03-27T09:44:10.241583', 'rating': 7}
Sending: {'name': 'Evangelia Panourgia', 'movie': 'He Named Me Malala', 'timestamp': '2025-03-27T09:45:10.302480', 'rating': 5}
Sending: {'name': 'Paul Johnson', 'movie': 'Irish Muzik Megs Day', 'timestamp': '2025-03-27T09:46:10.302480', 'rating': 1}
Sending: {'name': 'Teresa Burns', 'movie': 'Kurt Williams: Live', 'timestamp': '2025-03-27T09:47:10.422983', 'rating': 8}
Sending: {'name': 'Stephanie Carney', 'movie': 'Klaus', 'timestamp': '2025-03-27T09:48:10.483824', 'rating': 5}
Sending: {'name': 'Sandra Rhodes', 'movie': 'Virunga', 'timestamp': '2025-03-27T09:49:10.488891', 'rating': 7}
Sending: {'name': 'April Henry', 'movie': 'The Girl from the Song', 'timestamp': '2025-03-27T09:50:10.543484', 'rating': 10}
Sending: {'name': 'Randy Sanders', 'movie': 'Residente', 'timestamp': '2025-03-27T09:51:10.603801', 'rating': 2}
Sending: {'name': 'Richard Garcia', 'movie': 'National Lampoon's Loaded Weapon 1', 'timestamp': '2025-03-27T09:52:10.661944', 'rating': 6}
Sending: {'name': 'Pamela Wilson', 'movie': 'Underdogs', 'timestamp': '2025-03-27T09:53:10.721194', 'rating': 7}
Sending: {'name': 'Erika Brown', 'movie': 'Santa Claus', 'timestamp': '2025-03-27T09:54:10.724119', 'rating': 1}
Sending: {'name': 'Jimmy Strickland', 'movie': 'Aelita', 'timestamp': '2025-03-27T09:55:10.783169', 'rating': 2}
Sending: {'name': 'Evangelia Panourgia', 'movie': 'Young Tiger', 'timestamp': '2025-03-27T09:56:10.841672', 'rating': 9}
Sending: {'name': 'Paul Johnson', 'movie': 'Aghista Ahista', 'timestamp': '2025-03-27T09:57:10.901136', 'rating': 10}
Sending: {'name': 'Teresa Burns', 'movie': 'Ma Chu Ka', 'timestamp': '2025-03-27T09:58:10.902805', 'rating': 10}
Sending: {'name': 'Stephanie Carney', 'movie': 'The Bridge Curse', 'timestamp': '2025-03-27T09:59:10.904081', 'rating': 1}
Sending: {'name': 'Sandra Rhodes', 'movie': 'Carlos Almaraz: Playing with Fire', 'timestamp': '2025-03-27T10:00:10.963486', 'rating': 10}
Sending: {'name': 'April Henry', 'movie': 'My Dog is My Guide', 'timestamp': '2025-03-27T10:01:11.023200', 'rating': 9}
Sending: {'name': 'Randy Sanders', 'movie': 'Teenage Mutant Ninja Turtles II: The Secret of the Ooze', 'timestamp': '2025-03-27T10:02:11.083831', 'rating': 8}
Sending: {'name': 'Richard Garcia', 'movie': 'Bobby', 'timestamp': '2025-03-27T10:03:11.142444', 'rating': 3}
Sending: {'name': 'Pamela Wilson', 'movie': 'Green White Green (and All the Beautiful Colours in My Mosaic of Madness)', 'timestamp': '2025-03-27T10:04:11.144625', 'rating': 8}
Sending: {'name': 'Erika Brown', 'movie': 'The Cruise', 'timestamp': '2025-03-27T10:05:11.202701', 'rating': 10}
Sending: {'name': 'Jimmy Strickland', 'movie': 'Krishna', 'timestamp': '2025-03-27T10:06:11.203789', 'rating': 2}
Sending: {'name': 'Evangelia Panourgia', 'movie': 'Little Lunch: The Nightmare Before Graduation', 'timestamp': '2025-03-27T10:07:11.264497', 'rating': 8}
Sending: {'name': 'Paul Johnson', 'movie': 'Manje Bistre', 'timestamp': '2025-03-27T10:08:11.324032', 'rating': 7}
Sending: {'name': 'Stephanie Carney', 'movie': 'Perfume Imaginary Museum *Time Warp*', 'timestamp': '2025-03-27T10:09:11.387712', 'rating': 10}
Sending: {'name': 'Sandra Rhodes', 'movie': 'Ip Man 4: The Finale', 'timestamp': '2025-03-27T10:10:11.450538', 'rating': 10}
Sending: {'name': 'April Henry', 'movie': 'The Dark Side of Lifer Mumbai City', 'timestamp': '2025-03-27T10:11:11.507645', 'rating': 1}
Sending: {'name': 'Randy Sanders', 'movie': 'Get Me Roger Stone', 'timestamp': '2025-03-27T10:12:11.567807', 'rating': 2}
Sending: {'name': 'Richard Garcia', 'movie': 'Motor Mitrani Di', 'timestamp': '2025-03-27T10:13:11.568938', 'rating': 8}
Sending: {'name': 'Pamela Wilson', 'movie': 'Into the Forest', 'timestamp': '2025-03-27T10:14:11.628594', 'rating': 2}
Sending: {'name': 'Erika Brown', 'movie': 'Legendary Weapons of China', 'timestamp': '2025-03-27T10:15:11.689802', 'rating': 10}
Sending: {'name': 'Jimmy Strickland', 'movie': 'Blue Valentine', 'timestamp': '2025-03-27T10:16:11.751284', 'rating': 7}
Sending: {'name': 'Evangelia Panourgia', 'movie': 'Rozeta', 'timestamp': '2025-03-27T10:17:11.809531', 'rating': 4}
Sending: {'name': 'Paul Johnson', 'movie': 'Brother', 'timestamp': '2025-03-27T10:18:11.811605', 'rating': 1}
Sending: {'name': 'Teresa Burns', 'movie': 'Sakhi', 'timestamp': '2025-03-27T10:19:11.871299', 'rating': 1}
Sending: {'name': 'Stephanie Carney', 'movie': 'Center Stage', 'timestamp': '2025-03-27T10:20:11.932184', 'rating': 3}
Sending: {'name': 'Sandra Rhodes', 'movie': 'Special 26', 'timestamp': '2025-03-27T10:21:11.993588', 'rating': 2}
Sending: {'name': 'April Henry', 'movie': 'Velvet Colección: Grand Finale', 'timestamp': '2025-03-27T10:22:11.995800', 'rating': 6}
Sending: {'name': 'Randy Sanders', 'movie': 'Jinda', 'timestamp': '2025-03-27T10:23:11.997601', 'rating': 2}
Sending: {'name': 'Richard Garcia', 'movie': 'Malicious', 'timestamp': '2025-03-27T10:24:11.997630', 'rating': 4}
Sending: {'name': 'Pamela Wilson', 'movie': 'Skiptrace', 'timestamp': '2025-03-27T10:25:11.997630', 'rating': 10}
Sending: {'name': 'Erika Brown', 'movie': 'Mad Max', 'timestamp': '2025-03-27T10:26:11.997630', 'rating': 6}
Sending: {'name': 'Jimmy Strickland', 'movie': 'The Perfect Picture: Ten Years Later', 'timestamp': '2025-03-27T10:27:11.997630', 'rating': 6}

```

Figure 1: Terminal 1: Kafka Producer.

An additional modification was applied to the console script located in the examples directory, specifically to the file named console-spark-streaming-example.py. This adjustment was necessary to tailor the script to the project's requirements and ensure compatibility with the streaming architecture.

The image below depicts the console output helping us to debug the kafka producer real-time data. It runs in another terminal.

```

vagrant@vagrant:~/vagrant$ spark-submit \
--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0 \
examples/console-spark-streaming-example.py
25/03/27 09:43:45 WARN Utils: Your hostname, vagrant resolves to a loopback address: 127.0.0.1; using 10.0.2.15 instead (on interface eth0)
25/03/27 09:43:45 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
:: loading settings :: url = jar:file:/opt/spark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
Ivy Default Cache set to: /home/vagrant/.ivy2/cache
The jars for the packages stored in: /home/vagrant/.ivy2/jars
org.apache.spark:spark-sql-kafka-0-10_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark:spark-submit-parent-95d9e7b5-e749-4df1-a08d-5702b41bbb16:1.0
conf:: [default]
found org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0 in central
found org.apache.spark:spark-token-provider-kafka-0-10_2.12:3.5.0 in central
found org.apache.kafka:kafka-clients:3.4.1 in central
found org.lz4:lz4-java:1.8.0 in central
found org.xerial.snappy:snappy-java:1.1.10.3 in central
found org.slf4j:slf4j-api:2.0.7 in central
found org.apache.hadoop:hadoop-client-runtime:3.4 in central
found org.apache.hadoop:hadoop-client-api:3.4 in central
found commons-logging:commons-logging:1.3 in central
found com.google.code.findbugs:jsr305:3.0.0 in central
found org.apache.commons:commons-pool2:2.11.1 in central
downloading https://repo1.maven.org/maven2/org/apache/spark/spark-sql-kafka-0-10_2.12/3.5.0/spark-sql-kafka-0-10_2.12-3.5.0.jar ...
[SUCCESSFUL] org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0!spark-sql-kafka-0-10_2.12.jar (343ms)
downloading https://repo1.maven.org/maven2/org/apache/spark/spark-token-provider-kafka-0-10_2.12/3.5.0/spark-token-provider-kafka-0-10_2.12-3.5.0.jar ...

```

Figure 2: Terminal 2: Console for Kafka Producer.

Part 2

The pyspark script for the second part.

This PySpark application establishes a real-time data pipeline that reads movie rating events from a Kafka topic, enriches them with metadata from a static Netflix dataset, and writes the processed results into a Cassandra database. The script defines two schemas: one for parsing JSON-encoded rating messages received from Kafka, and another for reading static metadata from a CSV file. It converts timestamps to proper `TimestampType`, and groups data by hourly buckets for time-based aggregations. Ratings are joined with metadata on the movie title to create an enriched `DataFrame`, which is then written to Cassandra using the `foreachBatch` method in micro-batch mode. This architecture enables structured streaming analysis and time-windowed queries for individual users and movies.

To improve performance and avoid repeated disk reads, the static Netflix metadata is cached in memory, ensuring it remains efficiently accessible during each micro-batch operation within the streaming query.

Listing 2: `cassandra-spark-streaming-example.py`.

```

1 from pyspark.sql import SparkSession
2 from pyspark.sql.types import StructType, StructField, StringType, ←
   IntegerType, TimestampType
3 from pyspark.sql.functions import from_json, col, to_timestamp, ←
   date_format
4
5 # Schema for Kafka messages (movie ratings)
6 ratingSchema = StructType([
7     StructField("name", StringType(), False),
8     StructField("movie", StringType(), False),
9     StructField("timestamp", StringType(), False),
10    StructField("rating", IntegerType(), False)
11 ])

```

```

12
13 # Schema for Netflix CSV
14 netflixSchema = StructType([
15     StructField("show_id", StringType(), True),
16     StructField("title", StringType(), False),
17     StructField("director", StringType(), True),
18     StructField("country", StringType(), True),
19     StructField("release_year", StringType(), True),
20     StructField("rating", StringType(), True),
21     StructField("duration", StringType(), True)
22 ])
23
24 # Initialize Spark session
25 spark = (
26     SparkSession.builder
27     .appName("MovieRatingStreamer")
28     .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10↵
        _2.12:3.5.0,com.datastax.spark:spark-cassandra-connector_2↵
        .12:3.4.1")
29     .config("spark.cassandra.connection.host", "localhost")
30     .getOrCreate()
31 )
32
33 spark.sparkContext.setLogLevel("ERROR")
34
35 # Read Netflix CSV with renamed rating column to avoid conflict with "↵
    rating" in Kafka schema
36 netflix_df = (
37     spark.read.schema(netflixSchema)
38     .option("header", True)
39     .csv("data/netflix.csv")
40     .withColumnRenamed("rating", "rating_category") # Rename avoids ↵
        conflict
41     .cache()
42 )
43
44 # Read streaming data from Kafka
45 df = (
46     spark.readStream.format("kafka")
47     .option("kafka.bootstrap.servers", "localhost:29092")
48     .option("subscribe", "test")
49     .option("startingOffsets", "latest")
50     .load()
51 )
52
53 # Parse JSON from Kafka messages
54 ratings_df = (

```

```

55     df.selectExpr("CAST(value AS STRING)")
56     .select(from_json(col("value"), ratingSchema).alias("data"))
57     .select("data.*")
58     .withColumn("timestamp", to_timestamp("timestamp")) # Convert ↔
                    string timestamp to TimestampType
59     .withColumn("hour_bucket", date_format("timestamp", "yyyy-MM-dd HH↔
                    :00")) # Grouping by hour
60 )
61
62 # Join ratings with static Netflix metadata
63 enriched_df = (
64     ratings_df.join(netflix_df, ratings_df.movie == netflix_df.title, "↔
                    left")
65     .drop("title")
66 )
67
68 # Optional: Print schema for debugging (can be removed later)
69 enriched_df.printSchema()
70
71 # Define Cassandra write logic
72 def writeToCassandra(writeDF, _):
73     (
74         writeDF.select(
75             "name", "movie", "timestamp", "rating", "hour_bucket",
76             "show_id", "director", "country", "release_year", "↔
                    rating_category", "duration"
77         )
78         .write
79         .format("org.apache.spark.sql.cassandra")
80         .mode('append')
81         .options(table="movie_ratings", keyspace="netflix_ks")
82         .save()
83     )
84
85 # Write to Cassandra in a streaming loop
86 result = None
87 while result is None:
88     try:
89         result = (
90             enriched_df.writeStream
91                 .foreachBatch(writeToCassandra)
92                 .outputMode("update")
93                 .option("checkpointLocation", "/tmp/checkpoints/movie")
94                 .trigger(processingTime="30 seconds")
95                 .start()
96                 .awaitTermination()
97         )

```

```
98     except Exception as e:
99         print(f"Streaming error: {e}")
```

Details about your Cassandra data model.

The Cassandra data model was carefully designed to optimize query performance for the specific requirement of aggregating movie ratings by user and hour. The primary key is composed of a composite partition key (`name`, `hour_bucket`), ensuring that all ratings provided by a specific user within a given hour are stored in the same partition. This design eliminates the need for inefficient filtering operations and allows for fast, targeted queries such as retrieving all movies rated by a user during a specific hour or calculating the average rating and duration of those movies. The clustering columns (`timestamp`, `movie`) further organize the data chronologically within each partition, enabling efficient range queries and ordered retrieval. Crucially, the fields `rating` and `duration` are stored as integers, allowing native support for aggregation functions like `AVG()`, which would not be possible with text fields. Overall, this schema supports the core analytical tasks required by the application while adhering to Cassandra's best practices for scalability and performance.

Listing 3: Cassandra schema for movie ratings

```
1 CREATE KEYSPACE IF NOT EXISTS netflix_ks
2 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
3
4 CREATE TABLE IF NOT EXISTS netflix_ks.movie_ratings (
5     name text,
6     hour_bucket text,
7     timestamp timestamp,
8     movie text,
9     rating int,
10    show_id text,
11    director text,
12    country text,
13    release_year int,           -- changed from text : int
14    rating_category text,
15    duration int,             -- changed from text : int
16    PRIMARY KEY ((name, hour_bucket), timestamp, movie)
17 );
```

A sample of persisted lines (around 50) of your Cassandra table.

To inspect the data stored in the `movie_ratings` table of the `netflix_ks` keyspace, we can use the `cqlsh` shell interface provided by Apache Cassandra. The following query retrieves

Figure 3: Terminal 3: Spark pre-process, enrich with metadata..

```
SELECT * FROM netflix_ks.movie_ratings LIMIT 50;
```

name	hour_bucket	lifetime	movie	country	director	duration	rating	rating_category	release_year	show_id	
SELECT * FROM test_ko.movie_ratings LIMIT 50;											
Erin Brown	2025-03-27 09:00	2025-03-27 09:54.10.720000000000000000	Santa Claus	United States	Glenn Miller	98	1	TV-PG	2014	17914	
Erin Brown	2025-03-27 09:00	2025-03-27 10:30.000000000000000000	Tom Patrol	United States	Leslie Jones	85	1	TV-14	2014	18533	
Teresa Brown	2025-03-27 09:00	2025-03-27 10:49.13.720000000000000000	The C Word	United States	Stephen O'Rourke	90	1	TV-14	2014	17924	
Teresa Brown	2025-03-27 09:00	2025-03-27 11:08.13.720000000000000000	Julia	United States	Julia Roberts	90	1	TV-14	2017	17924	
Teresa Brown	2025-03-27 09:00	2025-03-27 11:42.12.720000000000000000	Strains of Fire	United States	John Ford	90	1	TV-14	2011	18111	
Teresa Brown	2025-03-27 09:00	2025-03-27 12:01.12.720000000000000000	Girl on the Edge	United States	Robert De Niro	90	1	TV-14	2011	18111	
Stephanie Carney	2025-03-27 09:00	2025-03-27 12:20.12.720000000000000000	Strained	United States	Robert De Niro	90	1	TV-14	2011	18111	
Stephanie Carney	2025-03-27 09:00	2025-03-27 09:46.16.480000000000000000	The Bridge	Spain	Sergio Pablos	88	5	M	2017	18274	
April Henry	2025-03-27 09:00	2025-03-27 10:05.11.480000000000000000	My Dog Is My Guide	Egypt	Gertie Haas	90	9	TV-14	2017	18274	
April Henry	2025-03-27 09:00	2025-03-27 10:24.11.480000000000000000	The Dark Side of Life	Iran	Tarzi Kham	90	9	TV-14	2017	18274	
April Henry	2025-03-27 09:00	2025-03-27 10:43.11.480000000000000000	Velvet Collection	Grand France	Bob Garmy	92	6	TV-14	2018	18289	
April Henry	2025-03-27 09:00	2025-03-27 11:02.11.480000000000000000	Barry Kamelion	New Year	Iran	Shahin Shariati	90	9	TV-14	2018	18289
April Henry	2025-03-27 09:00	2025-03-27 11:21.11.480000000000000000	Jeepers	Accord	United States	John Ford	90	1	TV-14	2011	18111
April Henry	2025-03-27 09:00	2025-03-27 11:40.11.480000000000000000	Jeepers	Accord	United States	John Ford	90	1	TV-14	2011	18111
April Henry	2025-03-27 09:00	2025-03-27 12:00.11.480000000000000000	Jeepers	Accord	United States	John Ford	90	1	TV-14	2011	18111
Randy Sanders	2025-03-27 09:00	2025-03-27 11:51.13.720000000000000000	Tenage Mutant Ninja Turtles II: The Secret of the Ooze	United States	Michael Bay	90	2	TV-14	2017	18444	
Randy Sanders	2025-03-27 09:00	2025-03-27 12:10.13.720000000000000000	Let Me Show You	United States	Dylan Beck, Daniel Shifano	90	2	TV-14	2017	18444	
Randy Sanders	2025-03-27 09:00	2025-03-27 12:29.13.720000000000000000	Let Me Show You	United States	Dylan Beck, Daniel Shifano	90	2	TV-14	2017	18444	
Randy Sanders	2025-03-27 09:00	2025-03-27 12:48.13.720000000000000000	Let Me Show You	United States	Dylan Beck, Daniel Shifano	90	2	TV-14	2017	18444	
Randy Sanders	2025-03-27 09:00	2025-03-27 13:07.13.720000000000000000	Let Me Show You	United States	Dylan Beck, Daniel Shifano	90	2	TV-14	2017	18444	
Erin Brown	2025-03-27 09:00	2025-03-27 13:26.13.720000000000000000	Let Me Show You	United States	Dylan Beck, Daniel Shifano	90	2	TV-14	2017	18444	
Erin Brown	2025-03-27 09:00	2025-03-27 13:45.13.720000000000000000	Let Me Show You	United States	Dylan Beck, Daniel Shifano	90	2	TV-14	2017	18444	
Erin Brown	2025-03-27 09:00	2025-03-27 14:04.13.720000000000000000	Let Me Show You	United States	Dylan Beck, Daniel Shifano	90	2	TV-14	2017	18444	
Erin Brown	2025-03-27 09:00	2025-03-27 14:23.13.720000000000000000	Let Me Show You	United States	Dylan Beck, Daniel Shifano	90	2	TV-14	2017	18444	
Erin Brown	2025-03-27 09:00	2025-03-27 14:42.13.720000000000000000	Let Me Show You	United States	Dylan Beck, Daniel Shifano	90	2	TV-14	2017	18444	
Erin Brown	2025-03-27 09:00	2025-03-27 15:01.13.720000000000000000	Let Me Show You	United States	Dylan Beck, Daniel Shifano	90	2	TV-14	2017	18444	
Erin Brown	2025-03-27 09:00	2025-03-27 15:20.13.720000000000000000	Let Me Show You	United States	Dylan Beck, Daniel Shifano	90	2	TV-14	2017	18444	
Erin Brown	2025-03-27 09:00										

Figure 4: A sample output of the Cassandra table with 50 persisted entries.

To support time-based analytics in our Cassandra data model, we designed the schema to partition data by name and hour_bucket, allowing efficient access to all ratings a specific user made during a given hour. The first query retrieves the titles of movies rated by the user *Evangelia Panourgia* during the hour of 2025-03-27 09:00 using a direct lookup on the partition key. The second query, which includes the ALLOW FILTERING clause, calculates the average

duration of those rated movies within the same hour. While Cassandra does not natively support aggregations without filtering, our schema minimizes performance issues by narrowing queries to a specific partition, ensuring practical execution even with the filtering clause.

```
cqlsh:netflix_ks> SELECT movie
... FROM netflix_ks.movie_ratings
... WHERE name = 'Evangelia Panourgia'
... AND hour_bucket = '2025-03-27 09:00';

movie
-----
Young Tiger

(1 rows)
cqlsh:netflix_ks> █
```

Figure 5: Names of the movies you rated during a particular hour SELECT movie.

```
cqlsh:netflix_ks> SELECT avg(duration)
... FROM netflix_ks.movie_ratings
... WHERE name = 'Evangelia Panourgia'
... AND hour_bucket = '2025-03-27 09:00'
... ALLOW FILTERING;

system.avg(duration)
-----
81

(1 rows)
```

Figure 6: Average duration of the movies you rated during the same hour SELECT avg(duration).

Furthermore, after allowing the real-time streaming process to run for an additional hour, we re-executed the following queries—this time updating the hour_bucket value from 09:00 to 10:00 to reflect the new time window.

```
cqlsh:netflix_ks> SELECT movie FROM netflix_ks.movie_ratings
... WHERE name = 'Evangelia Panourgia' AND hour_bucket = '2025-03-27 10:00';

movie
-----
Little Lunch: The Nightmare Before Graduation
Rezeta
Baby Dolls
Clair Obscur
Deadly Scholars

(5 rows)
```

Figure 7: Names of the movies you rated during a particular hour SELECT movie.

```
cqlsh:netflix_ks> SELECT avg(duration) FROM netflix_ks.movie_ratings WHERE name = 'Evangelia Panourgia' AND hour_bucket = '2025-03-27 10:00';

system.avg(duration)
-----
84

(1 rows)
```

Figure 8: Average duration of the movies you rated during the same hour SELECT avg(duration).

Note: It is not necessary to explicitly include conditions such as IS NOT NULL in the CQL

queries, as Cassandra automatically excludes null values from aggregation functions like AVG(). Therefore, rows with missing values for the targeted column are implicitly ignored during computation.

Hosted on GitHub Repository - Deploy Instructions.

The complete implementation of this project is hosted on GitHub and is publicly accessible at <https://github.com/e-panourgia/large-data-kafka-cassandra>. The repository contains all necessary components, including Kafka producers, Spark Structured Streaming jobs, and Cassandra schemas. To execute the project, clone the repository using `git clone https://github.com/e-panourgia/large-data-kafka-cassandra`, then follow the instructions in the README file to set up Docker containers, start Kafka and Cassandra services, and run the streaming application using `spark-submit`. Ensure that you have installed Docker, Docker Compose, and Spark locally or are running within the provided Vagrant virtual machine for a consistent and reproducible development environment.

Important note, this implementation was designed to run on mac m4 pc.

Open source software (OSS): Contribution to Faker

As part of the Applied Software Engineering course during my BSc studies, I had the opportunity to contribute to the open-source project Faker, a Python library widely used for generating synthetic data. My contribution focused on enhancing the tag-handling functionality, which plays a critical role in structured data generation. Specifically, I improved the way tags are processed within data pipelines, enabling more accurate and flexible population of fields with fake data. This work was incorporated through a merged pull request on GitHub, addressing limitations in tag parsing and significantly improving Faker's usability in scenarios involving automated testing and data anonymization. This contribution was relevant, as Faker was used in our project to auto-generate event data for a producer component.

Acknowledgements

I would like to sincerely thank **Professor Liakos Panayiotis** for his invaluable guidance, support, and dedication throughout the entire teaching process. His insightful lectures and continuous encouragement greatly contributed to the successful completion of this project.