SPARK MINI PROJECT

STREAMING SERIALIZATION

Using Spark

NANDAN PRABHU
2320667

Streaming Data Serialization: Serialize Streaming Data into Various Formats (E.G., JSON, Avro) For Storage or Exchange.

Abstract

This project aims to demonstrate the serialization of streaming data into various formats such as JSON, Avro, Parquet, ORC, and CSV on the Databricks platform. Leveraging Apache Spark and Python Faker library, the project covers data generation, transformation, aggregation, and streaming operations. Additionally, it includes uploading the serialized data to Google Cloud Storage and performing streaming operations on the downloaded data.

Introduction

Streaming data serialization is a crucial aspect of modern data processing systems where real-time data needs to be stored or exchanged efficiently. This project aims to demonstrate the process of serializing streaming data into popular formats like JSON, Avro, Parquet, ORC, and CSV. The project utilizes Apache Spark for data processing and manipulation. In the era of big data, efficient serialization of streaming data is crucial for real-time analytics and decision-making. This project showcases how Databricks, with its powerful distributed computing capabilities, enables seamless serialization of streaming data and integration with cloud storage solutions like Google Cloud Storage.

Streaming Data Serialization

Streaming data serialization is the process of converting data into a format that can be easily transmitted over a network or stored in a file, and then reconstructed at the receiving end. The term "streaming" implies that the data is processed in a continuous flow, rather than all at once. This approach is commonly used in scenarios where data needs to be transmitted or processed in real-time, such as in web applications, IoT devices, or communication protocols. Popular streaming

data serialization formats include JSON, XML, Protocol Buffers, Avro, and MessagePack.

Key stages in the data processing pipeline:

- Data ingestion
- Data cleansing
- Data preprocessing
- Data visualization

Project Implementation:

1. Data Generation:

➢ Python Faker library is used to generate synthetic data in RDD format.

```python
from pyspark.sql import SparkSession
from random import randint, choice
from faker import Faker

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("Generate Data") \
    .getOrCreate()
# Initialize Faker
fake=Faker()

# Create RDD with 5000 rows
data = spark.sparkContext.parallelize(range(5000))

# Function to generate random data for each row
def generate_row(_):
    movie_or_tv_show = choice(["Movie", "TV Show"])
    director = fake.name()
    cast = fake.name()
    rental_price=randint(49, 149)
    rating = randint(1, 10)
    release_year = randint(1950, 2024)
    country = choice(["USA", "UK", "India", "France", "Germany"])
    duration = randint(60, 240)

    return (movie_or_tv_show, director, cast, rating, release_year,rental_price, country, duration)

# Generate data
data = data.map(generate_row)
data.collect()
```

2. Data Transformation:

- The RDD format of Data and printed using ‘collect()’ keyword.

```
28  # Generate data
29  data = data.map(generate_row)
30  data.collect()
```

▸ (1) Spark Jobs

```
('TV Show', 'Heather Sherman', 'Cynthia Clark', 4, 1958, 123, 'USA', 125),
('Movie', 'Joshua Conway', 'Michelle Walker', 1, 1959, 54, 'USA', 240),
('TV Show', 'Clifford Moody', 'Diane Stanley', 8, 2003, 83, 'USA', 102),
('TV Show',
 'Jeffery Russell',
 'William Williams',
 10,
 1994,
 112,
 'France',
 128),
('TV Show', 'Charles Kelley', 'Susan Quinn', 2, 1989, 72, 'Germany', 83),
('Movie', 'Nicholas Cochran', 'Debbie Tucker', 7, 1997, 49, 'France', 138),
('TV Show', 'Jennifer Lang', 'Mitchell Butler', 2, 1964, 84, 'France', 153),
('TV Show', 'Crystal Carroll', 'William Werner', 8, 1971, 122, 'France', 84),
('Movie', 'Louis Olson', 'Brittany Nguyen', 10, 1977, 135, 'India', 230),
('TV Show', 'Kelsey Johnson', 'Janice Young', 3, 2004, 85, 'India', 87),
('TV Show', 'Roberta Guerrero', 'David Young', 3, 1952, 72, 'USA', 94),
('Movie', 'Johnathan Mayo', 'Marisa Price', 9, 1957, 134, 'USA', 75),
('TV Show', 'Kathy Mckenzie', 'Walter Williams', 1, 1950, 116, 'France', 67),
...]
```

Command took 7.77 seconds -- by nandanprabhu1046@gmail.com at 3/24/2024, 12:50:27 PM on Sunday_Afternoon

- The RDD data is converted into a DataFrame for easier manipulation.

```
1  # Create DataFrame
2  schema = ["movie_or_tv_show", "director", "cast", "rating", "release_year","rental_price", "country", "duration"]
3  netflix_dataframe = spark.createDataFrame(data, schema=schema)
4
5  # Show DataFrame
6  netflix_dataframe.show()
```

▸ (2) Spark Jobs

▸ netflix_dataframe: pyspark.sql.dataframe.DataFrame = [movie_or_tv_show: string, director: string ... 6 more fields]

```
|         TV Show|     Rachel Chen|     Rodney Flynn|    10|        1995|         144|     USA|     130|
|           Movie|     Amy Johnson| Danielle Daniel|     5|        1971|          81| Germany|     135|
|           Movie|     Alan Oliver| Victoria Reilly|     9|        2014|         131|  France|     182|
|           Movie|     Larry Smith|  Justin Hancock|     4|        1994|         116|   India|      74|
|         TV Show|    Peter Vargas|  Emily Anderson|     9|        2006|         137|  France|     107|
|           Movie|     Larry Greer|    Brett Brooks|     7|        2013|         118|  France|      76|
|           Movie|     Jacob Crane|  Michael Patton|    10|        1958|          99| Germany|     165|
|         TV Show|    Isaac Ramsey| Valerie Hawkins|     8|        2002|          97|   India|      65|
|         TV Show|    Kevin Lowery|   Cory Whitaker|     6|        1983|         143|   India|     240|
|           Movie| Patrick Johnson|     Dylan Davis|     2|        1990|         137|     USA|     158|
|         TV Show|  Rachel Compton|   Bobby Wallace|     4|        1989|          66|  France|     119|
|           Movie|  Patrick Austin|  Victoria Lewis|     3|        2023|          89| Germany|     149|
|           Movie|  Shannon Miller|Gwendolyn Bailey|     8|        1956|         149|   India|     238|
|           Movie| Tristan Bullock|   Shannon James|     7|        2017|          75|  France|     205|
|         TV Show|     Brian Moore|   Thomas Duncan|     4|        1995|         117|     USA|      92|
|           Movie|       Lori King|     Robert Tran|     8|        2023|          82|  France|     192|
|           Movie|    Heather Kane|  Taylor Maxwell|     3|        2008|          81|      UK|     126|
|         TV Show|  Eric Hernandez|   Mary Thompson|     4|        1961|          60| Germany|      69|
+----------------+----------------+----------------+------+------------+------------+--------+--------+
only showing top 20 rows
```

- Checking the schema of newly created DataFrame and Counting of number of rows.

```
1  netflix_dataframe.printSchema()

root
 |-- movie_or_tv_show: string (nullable = true)
 |-- director: string (nullable = true)
 |-- cast: string (nullable = true)
 |-- rating: long (nullable = true)
 |-- release_year: long (nullable = true)
 |-- rental_price: long (nullable = true)
 |-- country: string (nullable = true)
 |-- duration: long (nullable = true)

Command took 0.22 seconds -- by nandanprabhu1046@gmail.com at 3/24/2024, 12:50:27 PM on Sunday_Afternoon
```

```
Cmd 5

1  netflix_dataframe.count()

▸ (2) Spark Jobs

Out[4]: 5000

Command took 7.01 seconds -- by nandanprabhu1046@gmail.com at 3/24/2024, 12:50:27 PM on Sunday_Afternoon
```

- Various DataFrame operations are performed including:
  - Removing null valued rows.

```
1  netflix_no_nulls=netflix_dataframe.na.drop()
2  netflix_no_nulls.show()

▸ (1) Spark Jobs
▸ netflix_no_nulls: pyspark.sql.dataframe.DataFrame = [movie_or_tv_show: string, director: string ... 6 more fields]
```

TV Show	Rachel Chen	Rodney Flynn	10	1995	144	USA	130
Movie	Amy Johnson	Danielle Daniel	5	1971	81	Germany	135
Movie	Alan Oliver	Victoria Reilly	9	2014	131	France	182
Movie	Larry Smith	Justin Hancock	4	1994	116	India	74
TV Show	Peter Vargas	Emily Anderson	9	2006	137	France	107
Movie	Larry Greer	Brett Brooks	7	2013	118	France	76
Movie	Jacob Crane	Michael Patton	10	1958	99	Germany	165
TV Show	Isaac Ramsey	Valerie Hawkins	8	2002	97	India	65
TV Show	Kevin Lowery	Cory Whitaker	6	1983	143	India	240
Movie	Patrick Johnson	Dylan Davis	2	1990	137	USA	158
TV Show	Rachel Compton	Bobby Wallace	4	1989	66	France	119
Movie	Patrick Austin	Victoria Lewis	3	2023	89	Germany	149
Movie	Shannon Miller	Gwendolyn Bailey	8	1956	149	India	238
Movie	Tristan Bullock	Shannon James	7	2017	75	France	205
TV Show	Brian Moore	Thomas Duncan	4	1995	117	USA	92
Movie	Lori King	Robert Tran	8	2023	82	France	192
Movie	Heather Kane	Taylor Maxwell	3	2008	81	UK	126
TV Show	Eric Hernandez	Mary Thompson	4	1961	60	Germany	69

only showing top 20 rows

- Count of table after dropping null values and saved the file in CSV format in DBFS.

```
1  netflix_no_nulls.count()
```

▸ (2) Spark Jobs

```
Out[6]: 5000
```

Command took 2.89 seconds -- by nandanprabhu1046@gmail.com at 3/24/2024, 12:50:27 PM on Sunday_Afternoon

Cmd 8

```
1  # Writing data to DBFS in CSV format
2  netflix_no_nulls.write.csv("dbfs:/FileStore/tables/Netflix_DataCSV.csv")
```

▸ (1) Spark Jobs

Command took 7.12 seconds -- by nandanprabhu1046@gmail.com at 3/24/2024, 12:50:27 PM on Sunday_Afternoon

- Using `withColumn` to add or update columns.

```
1  netflix_new_rent = netflix_rent_datatype.withColumn("new_rental_price", col('rental_price') + 12)
2  netflix_new_rent.show()
```

▸ (1) Spark Jobs

▸ netflix_new_rent: pyspark.sql.dataframe.DataFrame = [movie_or_tv_show: string, director: string ... 7 more fields]

movie_or_tv_show	director	cast	rating	release_year	rental_price	country	duration	new_rental_price
Movie	Mark Burke	Nathan Perez	10	2018	117	India	66	129
TV Show	Jessica Woodward	Amanda Smith	3	1960	106	USA	219	118
TV Show	Rachel Chen	Rodney Flynn	10	1995	144	USA	130	156
Movie	Amy Johnson	Danielle Daniel	5	1971	81	Germany	135	93
Movie	Alan Oliver	Victoria Reilly	9	2014	131	France	182	143
Movie	Larry Smith	Justin Hancock	4	1994	116	India	74	128
TV Show	Peter Vargas	Emily Anderson	9	2006	137	France	107	149
Movie	Larry Greer	Brett Brooks	7	2013	118	France	76	130
Movie	Jacob Crane	Michael Patton	10	1958	99	Germany	165	111
TV Show	Isaac Ramsey	Valerie Hawkins	8	2002	97	India	65	109
TV Show	Kevin Lowery	Cory Whitaker	6	1983	143	India	240	155
Movie	Patrick Johnson	Dylan Davis	2	1990	137	USA	158	149
TV Show	Rachel Compton	Bobby Wallace	4	1989	66	France	119	78
Movie	Patrick Austin	Victoria Lewis	3	2023	89	Germany	149	101
Movie	Shannon Miller	Gwendolyn Bailey	8	1956	149	India	238	161
Movie	Tristan Bullock	Shannon James	7	2017	75	France	205	87
TV Show	Brian Moore	Thomas Duncan	4	1995	117	USA	92	129
Movie	Lori King	Robert Tran	8	2023	82	France	192	94

Command took 0.97 seconds -- by nandanprabhu1046@gmail.com at 3/24/2024, 1:21:03 PM on Sunday_Afternoon

- Filtering rows based on certain conditions.

```
1  netflix_flitered = netflix_new_rent.filter((netflix_new_rent.country == "India") & (netflix_new_rent.duration > 50) )
2  netflix_flitered.show()
```

(1) Spark Jobs

netflix_flitered: pyspark.sql.dataframe.DataFrame = [movie_or_tv_show: string, director: string ... 7 more fields]

Movie	Mark Burke	Nathan Perez	10	2018	117	India	66	129
Movie	Larry Smith	Justin Hancock	4	1994	116	India	74	128
TV Show	Isaac Ramsey	Valerie Hawkins	8	2002	97	India	65	109
TV Show	Kevin Lowery	Cory Whitaker	6	1983	143	India	240	155
Movie	Shannon Miller	Gwendolyn Bailey	8	1956	149	India	238	161
TV Show	Angela Foster	Jasmin Williams	9	1986	70	India	68	82
Movie	Erica Bishop	Lisa Harris	4	1956	119	India	171	131
Movie	Karina Tate	Katherine Kim	3	1960	146	India	122	158
TV Show	Christopher Pope	Amy Fernandez	7	2023	146	India	221	158
TV Show	Jerry Shaw	Jodi Mcdowell	9	1975	112	India	230	124
Movie	Randy Houston	Jessica Garrett	6	2001	78	India	160	90
Movie	Patricia Drake	Michael Jordan	8	1978	121	India	95	133
TV Show	Charles Buckley	Margaret Day DDS	6	1951	50	India	215	62
TV Show	Wayne Robinson	Candice Yates	10	2005	79	India	89	91
Movie	Jennifer Becker	Kevin Jackson	1	1972	94	India	171	106
TV Show	Warren Henry	Michael Brown	10	2006	149	India	100	161
Movie	Alex Jackson	Emily Baker	4	1995	55	India	110	67
Movie	Ashley Walker	Kimberly Garcia	8	1973	131	India	104	143
Movie	Tracy Cruz	Amanda Little	4	1998	76	India	101	88
Movie	Danny Harris	Lisa Johnson	3	2023	106	India	90	118

Command took 0.83 seconds -- by nandanprabhu1046@gmail.com at 3/24/2024, 1:40:18 PM on Sunday_Afternoon

Cmd 14

```
1  netflix_flitered.count()
```

(2) Spark Jobs

Out[22]: 1016

Command took 3.19 seconds -- by nandanprabhu1046@gmail.com at 3/24/2024, 1:40:27 PM on Sunday_Afternoon

- Calculating averages.

```
1  # Adding new column and averaging the data.
2  netflix_total_rating = netflix_flitered.withColumn("total_rating", lit(10))
3  netflix_average_rating = netflix_total_rating.withColumn("Average_Rating", (col("rating") / col ("total_rating"))*100 )
4  netflix_movie = netflix_average_rating.filter((netflix_average_rating.movie_or_tv_show == "Movie") & (netflix_average_rating.Average_Rating > 70))
5  netflix_movie.select("director","release_year","Average_Rating").show()
```

(2) Spark Jobs

netflix_total_rating: pyspark.sql.dataframe.DataFrame = [movie_or_tv_show: string, director: string ... 8 more fields]

netflix_average_rating: pyspark.sql.dataframe.DataFrame = [movie_or_tv_show: string, director: string ... 9 more fields]

netflix_movie: pyspark.sql.dataframe.DataFrame = [movie_or_tv_show: string, director: string ... 9 more fields]

Chad Johnston	1984	80.0
Erik Martinez	2019	100.0
Ronald Montoya	2021	100.0
Kyle Gregory	1951	80.0
Paul Smith	1980	80.0
James Nichols	1994	90.0
Kimberly Wood	1992	90.0
Steven Fields	2007	80.0
Trevor Doyle	1982	90.0
Jennifer Lane	1958	80.0
Rhonda Lewis	1960	90.0
Juan Jenkins	1993	80.0
Thomas Miller	2011	100.0
Ronald Turner	1967	90.0
Heather Rose	2017	80.0
Mr. Larry Todd	2017	90.0
Daniel Jones	1978	100.0
Paul Lewis	1953	80.0

only showing top 20 rows

Command took 2.33 seconds -- by nandanprabhu1046@gmail.com at 3/24/2024, 5:25:06 PM on sunday_evening

- Using `lit` function to create columns with literal values.

```
1  from pyspark.sql import SparkSession
2  from pyspark.sql.functions import col, lit
3  spark = SparkSession.builder.appName("Spark DataFrames").getOrCreate()
4
5  netflix_new_rent = netflix_no_nulls.withColumn("rental_price", col("rental_price").cast
   ("Integer"))
6  netflix_new_rent.printSchema()
```

▸ netflix_new_rent: pyspark.sql.dataframe.DataFrame = [movie_or_tv_show: string, director: string ... 6 more fields]

```
root
 |-- movie_or_tv_show: string (nullable = true)
 |-- director: string (nullable = true)
 |-- cast: string (nullable = true)
 |-- rating: long (nullable = true)
 |-- release_year: long (nullable = true)
 |-- rental_price: integer (nullable = true)
 |-- country: string (nullable = true)
 |-- duration: long (nullable = true)
```

Command took 0.30 seconds -- by nandanprabhu1046@gmail.com at 3/24/2024, 1:17:09 PM on Sunday_Afternoon

3. Data Aggregation:

- Grouping data based on certain criteria using `groupBy` operation.

```
1  # Grouping and counting operation
2  show_group=netflix_no_nulls.groupBy("movie_or_tv_show", "country").count()
3  show_group.show()
```

▸ (2) Spark Jobs

▸ show_group: pyspark.sql.dataframe.DataFrame = [movie_or_tv_show: string, country: string ... 1 more field]

```
+----------------+-------+-----+
|movie_or_tv_show|country|count|
+----------------+-------+-----+
|         TV Show| France|  512|
|           Movie|  India|  528|
|         TV Show|  India|  488|
|           Movie|    USA|  456|
|           Movie| France|  608|
|         TV Show|Germany|  496|
|           Movie|     UK|  456|
|         TV Show|     UK|  632|
|         TV Show|    USA|  408|
|           Movie|Germany|  416|
+----------------+-------+-----+
```

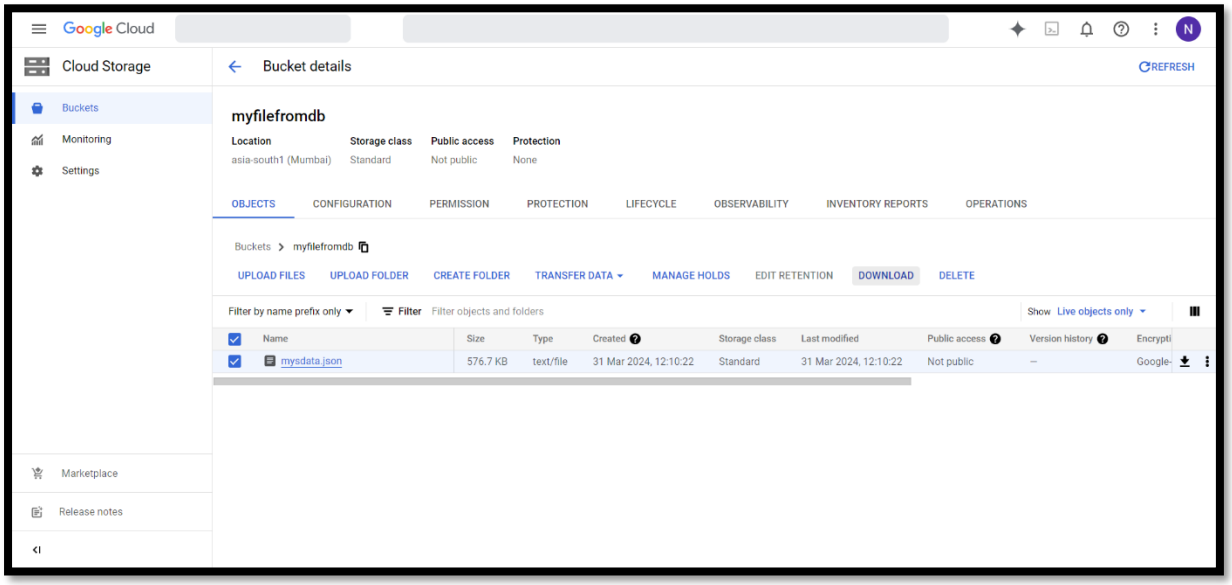
4. Serialization:

- Processed data is saved into different serialization formats including JSON, Avro, Parquet, ORC, and CSV.

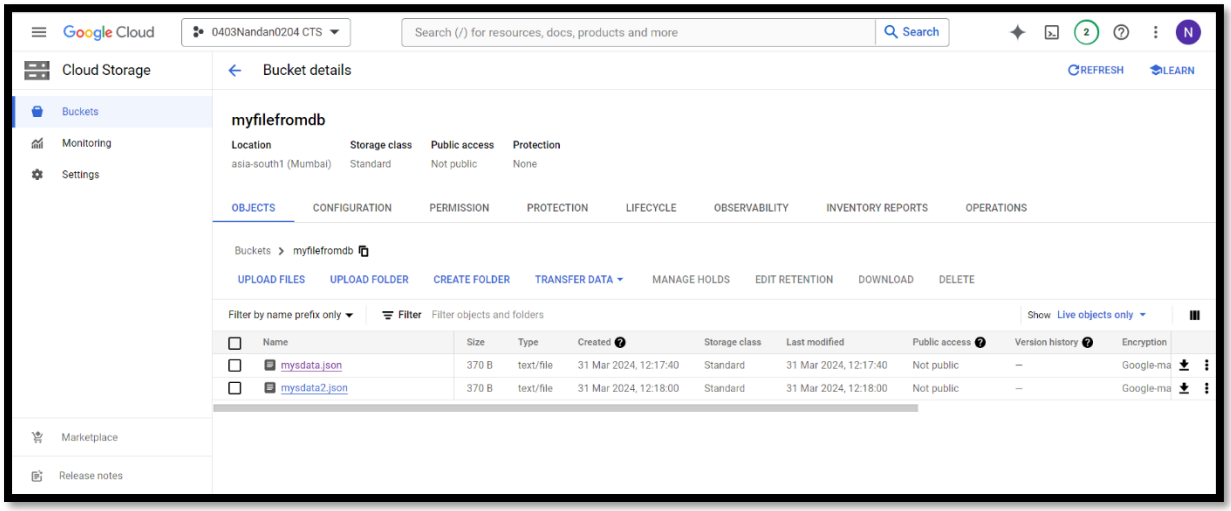
- Each format offers distinct advantages in terms of compression, schema evolution, and query performance.

5. Load the DataFrame to Google cloud Bucket

- Creating a google bucket.



- Buckets are the basic containers that hold your data within Cloud Storage.



- Everything you store in Cloud Storage must be contained in a bucket.
- Create connection and upload the required file to Cloud.

```
1  dbfs_path='/FileStore/tables/nandan0204_cts_6606f3d29e04-1.json'
2  new_local_path="file:///tmp/nandan0204_cts_6606f3d29e04-1.json"
3  dbutils.fs.cp(dbfs_path,new_local_path)
```

```
Command skipped
```

```
Command took 0.00 seconds -- by nandanprabhu1046@gmail.com at 3/31/2024, 11:28:36 AM on Sunday@11
```

```
Cmd 20
```

```
1  ls /tmp
```

```
Command skipped
```

```
Command took 0.00 seconds -- by nandanprabhu1046@gmail.com at 3/31/2024, 11:28:36 AM on Sunday@11
```

```
Cmd 21
```

```
1  pip install --upgrade google-cloud-storage
```

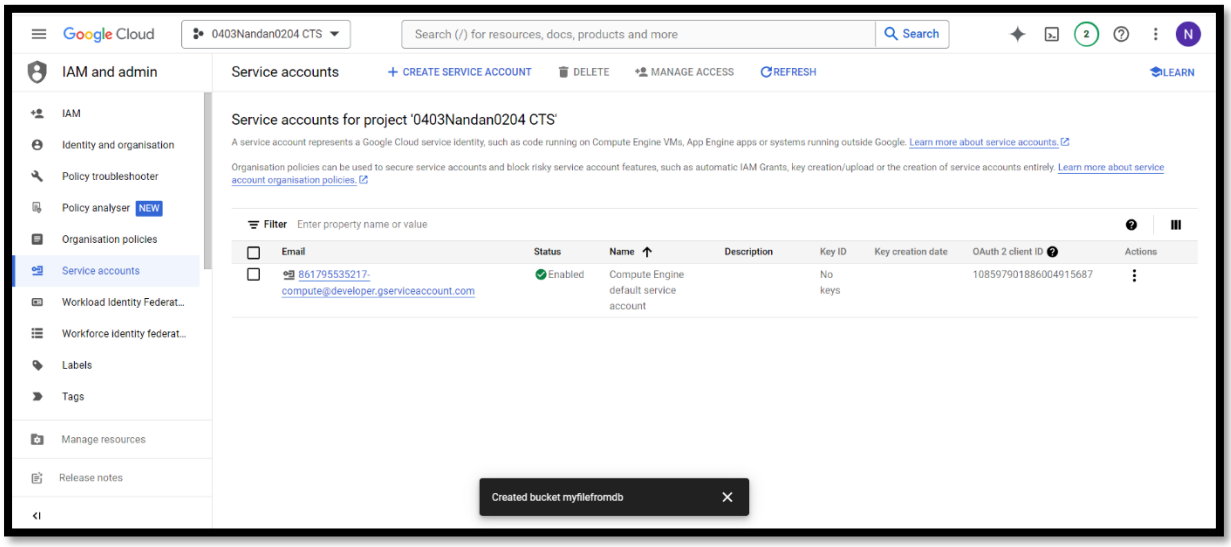
```
Command skipped
```

```
Command took 0.00 seconds -- by nandanprabhu1046@gmail.com at 3/31/2024, 11:28:36 AM on Sunday@11
```

```
Cmd 22
```

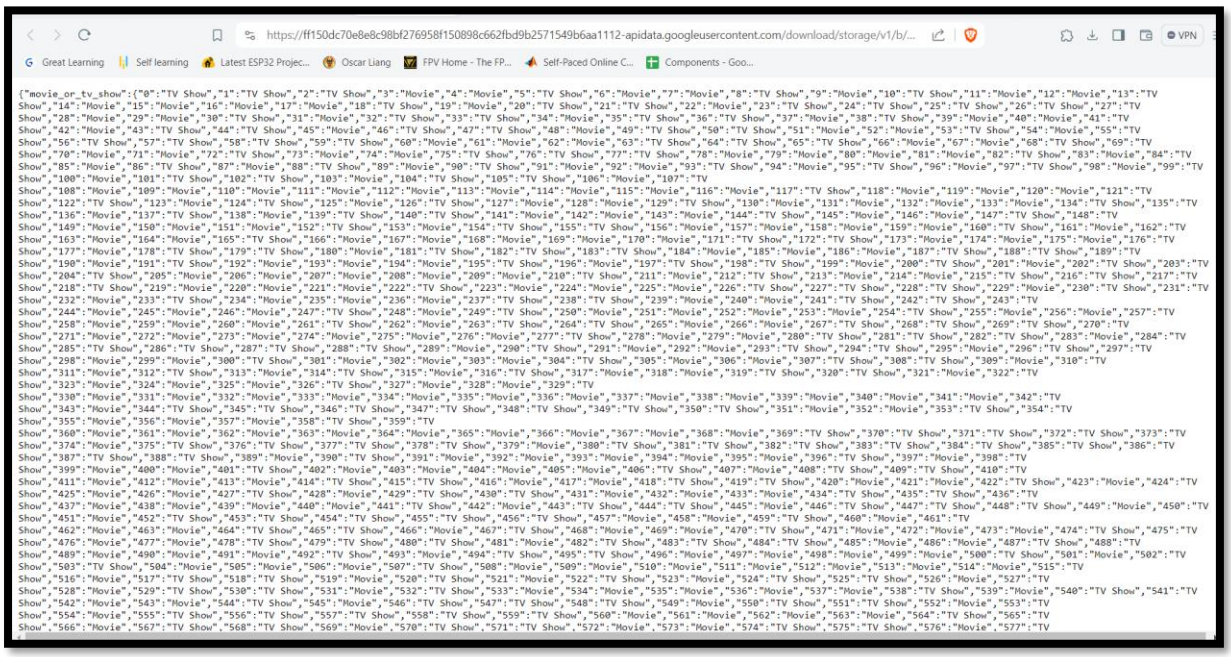
```
1  from google.cloud import storage
2
3  path_to_private_key ='/tmp/nandan0204_cts_6606f3d29e04-1.json'
4  client=storage.Client.from_service_account_json(json_credentials_path=path_to_private_key)
5  bucket=storage.Bucket(client,'databrickstocld')
6  blob=bucket.blob('mysdata.json')
7  blob.upload_from_string(netflix_dataframe.toPandas().to_json(),'text/file')
```

- Creating a service account and giving required permissions.

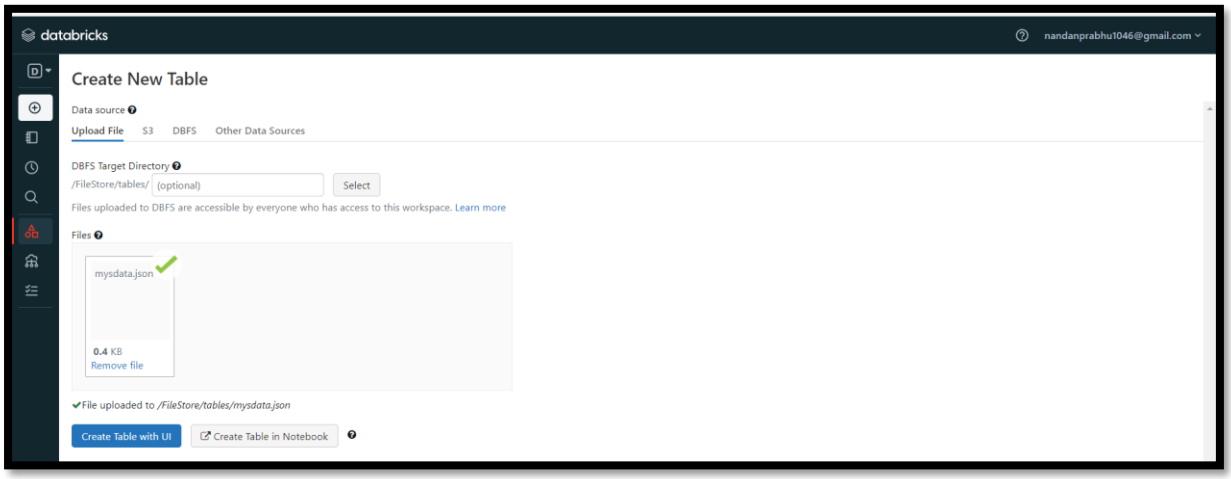


6. Downloading and Streaming:

- Retrieve the serialized data from the Google Cloud Storage bucket to the Databricks environment.



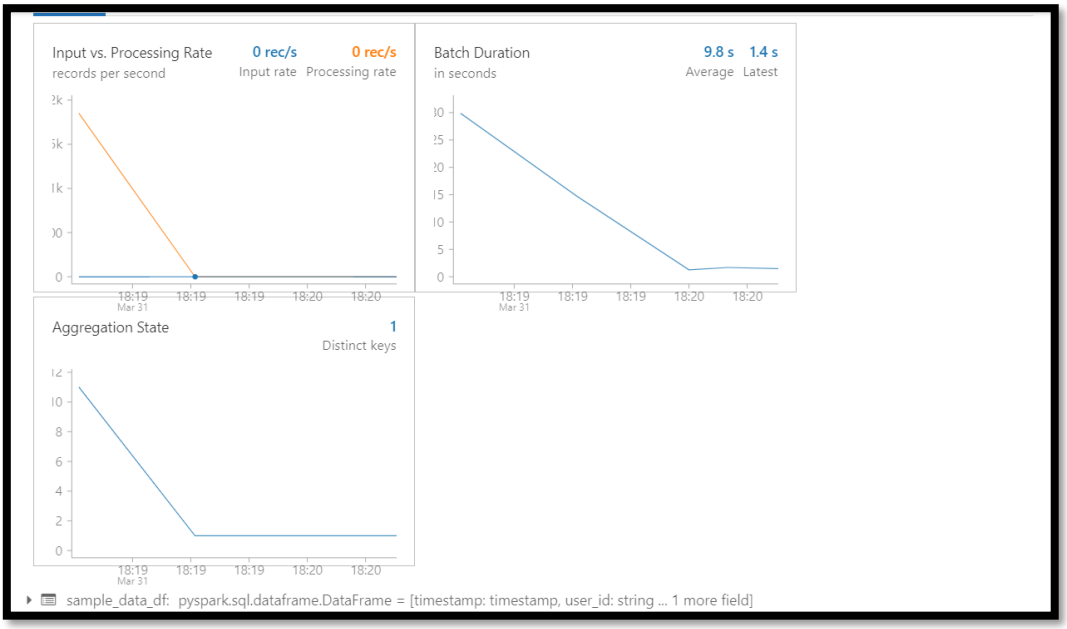
- Implement streaming operations on the downloaded data.



7. Streaming Operation:

- Implementing streaming operations to handle continuous data streams.

- The project demonstrates how to process streaming data and serialize it into various formats in real-time.



Conclusion

This project demonstrates the end-to-end process of streaming data serialization on the Databricks platform. By leveraging Apache Spark and Databricks' integrations with cloud storage services like Google Cloud Storage, organizations can efficiently process, serialize, and analyse streaming data for real-time insights.