**MongoDB Assignment Submission Doc**

This file details and describes all the attached files for the Kafka Assignment

**Tools Used:**

1. Python3
2. Confluent Kafka
3. Jupyter Notebook
4. Postman
5. MongoDB Atlas
6. MongoDB Compass
7. Flask

**Files Attached:**

1. Mongodb\_assignment.pdf – This file
2. delivery\_trip\_truck\_data.csv – The csv raw data used to push to the kafka topic
3. logistics\_data\_producer.py – Python producer script
4. logistics\_data\_consumer.py – Python consumer script
5. logistics\_data\_api1.py – Python script which creates a Flask API end pt where you can filter for a particular document using the ‘vehicle number’ from the MongoDB database.
6. logistics\_data\_api2.py – Python script that shows the aggregated counts of vehicle numbers for each GpsProvider type.
7. Dockerfile.txt – Docker file that can be used to build an docker image
8. docker-compose.yml– YAML docker compose file that details the docker image, environment details (Kafka config) and the consumer script to parallelize.

**Process and File Descriptions:**

**Step 1**

Created a kafka topic called ‘logistics\_data’ with 6 partitions and I made sure to save the API keys for the producer. I also created an appropriate schema value and key to prepare the kafka topic for data ingestion/retrieval looking at the delivery\_trip\_truck\_data.csv file. I especially made sure to handle the ‘Nan’ values by replacing them with the string ‘unknown value’ if the field is string type.

**Step 2**

I created a producer script called “logistics\_data\_producer.py” that fetches the data from the afore- mentioned Kafka topic. The script also serializes the data into Avro format and uses GPSProvider as the key. The below image shows the producer fetching data. It also sends out a message saying that the record value as been successfully produced in a particular partition.

A screen shot of a computer screen

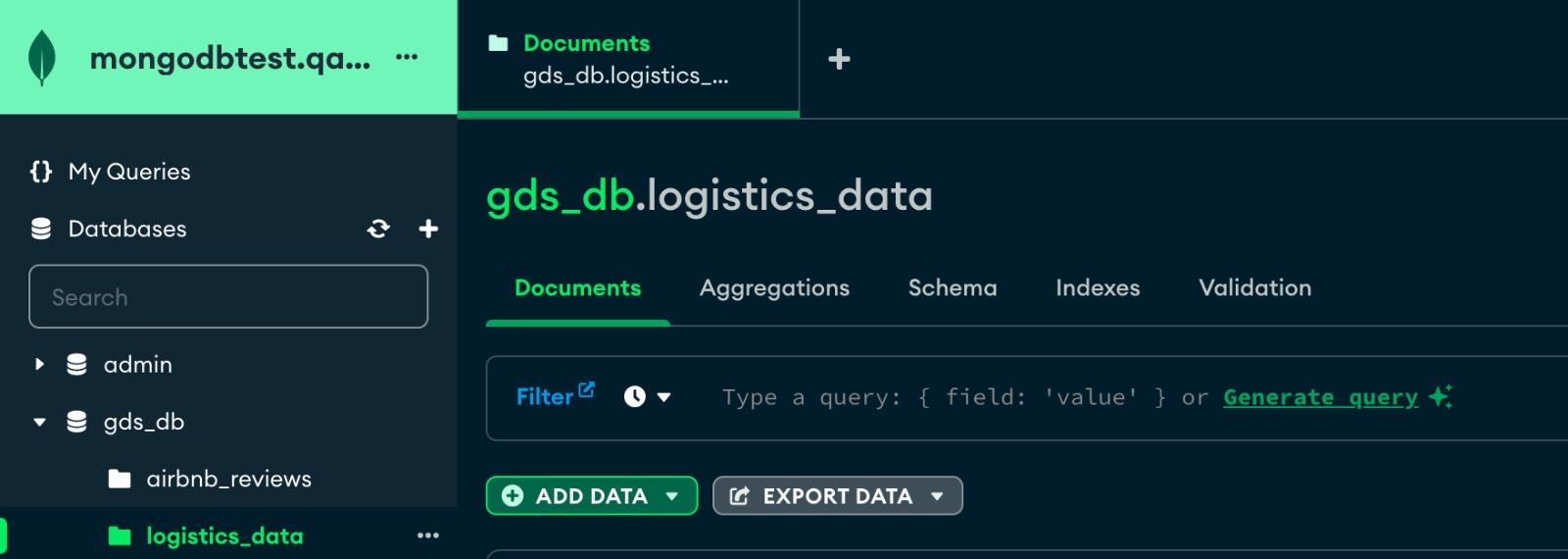
Description automatically generated

A screen shot of a computer screen

Description automatically generated

**Step 3**

Created a mongodb database called ‘gds\_db’ and created an empty collection called ‘logistics\_data’ so that data can be stored once a consumer script can be run.



**Step 4**

I then created a consumer script called “logistics\_data\_consumer.py” that deserializes the avro data back into a python object. I then implemented data validation checks in the code to make sure that it accounts for null values, correct data types and implements format checks. Before pushing the data into the logistics\_data collection that was created in the gdb\_db mongodb database, I make sure that there are no duplicate records pushed when the consumer runs.

**Step 5**

To achieve scaling, I first created a Dockerfile detailing the package dependencies which is used to create a docker image. Then I launch the Docker-compose file to set my consumer count = 3. Running this file shows the three consumers able to parallelize the output to then feed the data into the MongoDB database.



A screen shot of a computer screen

Description automatically generated

We can also check the data using Mongodb Compass:

A screenshot of a computer

Description automatically generated

**Step 6**

Using Flask API I was able to generate an API endpoint which takes in ‘vehicle\_no’ as a parameter to filter out the document. I then made sure to have this endpt point running and checked the output using Postman as well as just through the browser url.

The below API endpt is used to filter the document with vehicle number = ‘HR68B5696’

This is a snapshot of the code running.

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

The second API is used to count vehicle numbers grouped by each GPSProvider

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated