

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. MongoDB Atlas
4. MongoDB Compass
5. Flask

Files Attached:

1. mongo_Assignment.pdf – This file
2. delivery_trip_truck_data_new.csv – The csv raw data used to push to the kafka topic
3. producer_Code.py – Python producer script
4. Consumer_code.py – Python consumer script
5. flaskmongo.py – Python script which creates a Flask App by opening on 5000 port just paste your aggregation pipeline to get the result
6. Dockerfile – Docker file that can be used to build an docker image for flask app.
7. producer.Dockerfile – Docker file that can be used for producer.
8. consumer.Dockerfile-Docker file that can be used for consumer
9. docker-compose.yml-Docker compose file for flask app and mongodb
10. docker-compose.yml– YAML docker compose file that details the docker image, environment details (Kafka config) and the consumer script to parallelize.\
11. .env – for environment variables.
12. Avro_schema.json – which contain avro_schema
13. Index.html – which contain code for frontend of flask app.
14. Sample_mongo_query.py- which contain query for aggregation which can be used in flask app.

The code and additional resources can be found in the following GitHub repository: [\[GitHub Link\]](#)

Process and File Descriptions:

Step 1

Created a kafka topic called 'logistics_data1' 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_new.csv file. I especially made sure to handle the 'Nan' values by replacing them with the None value' if the field is string type. And also make all data type to string.

Step 2

I created a producer script called “Producer_code.py” that fetches the data from the above- mentioned Kafka topic. The script also serializes the data into Avro format and uses index value as the key.

Step 3

Created a mongodb database called ‘logistic_data and created an empty collection called ‘delivery_truck_data’ so that data can be stored once a consumer script can be run.

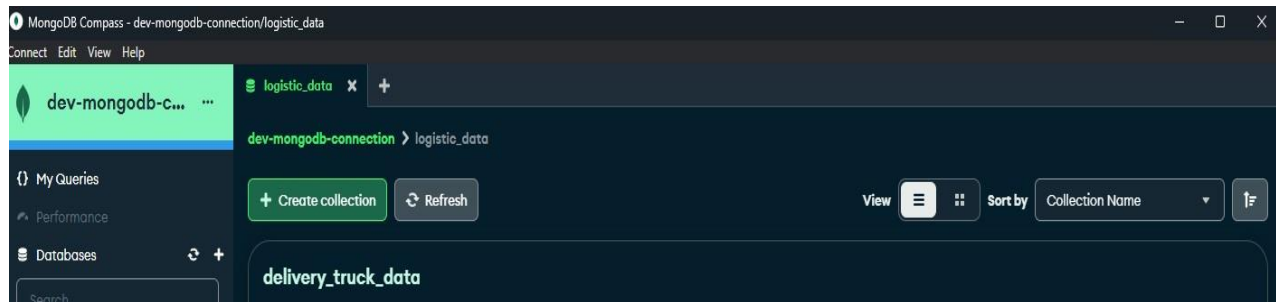


Fig. 1: MongoDB database name and Collection Name

Step 4

Then created a “Consumer_code.py” where i wrote the logic for the consumer in which before dumping data to the mongodb collection made sure to convert all the string type data which were published by producer will get convert back to it’s original data type also give the mongodb collection connection in the consumer code.I make sure that there are no duplicate records pushed when the consumer runs.

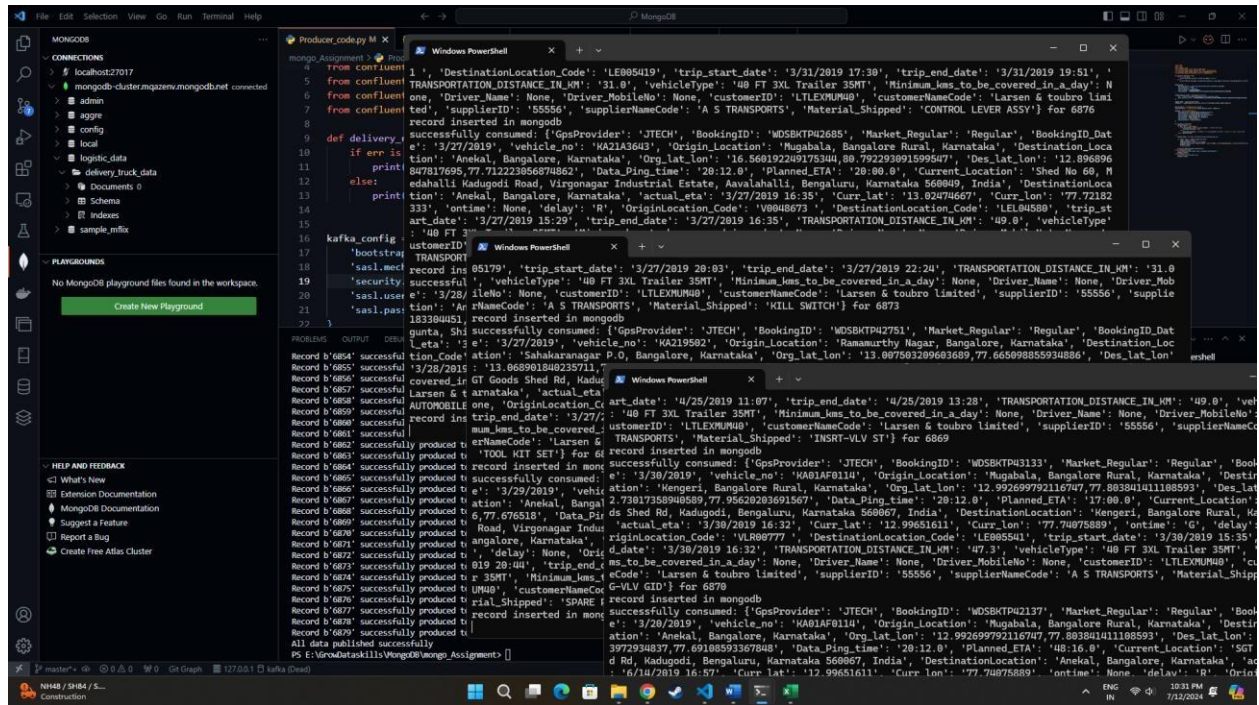


Fig. 2: A Producer along with three Consumers to parallelize the processing

Step 5

To achieve scaling, I first created a Dockerfile for producer and one for consumer then create the docker compose file and used both of this dockerfile in it as we need to scaled the system hence created 3 consumer instance in the docker-compose file and also create the dockerfile and docker-compose file for the flask app.

We can also check the data using Mongoddb Compass:

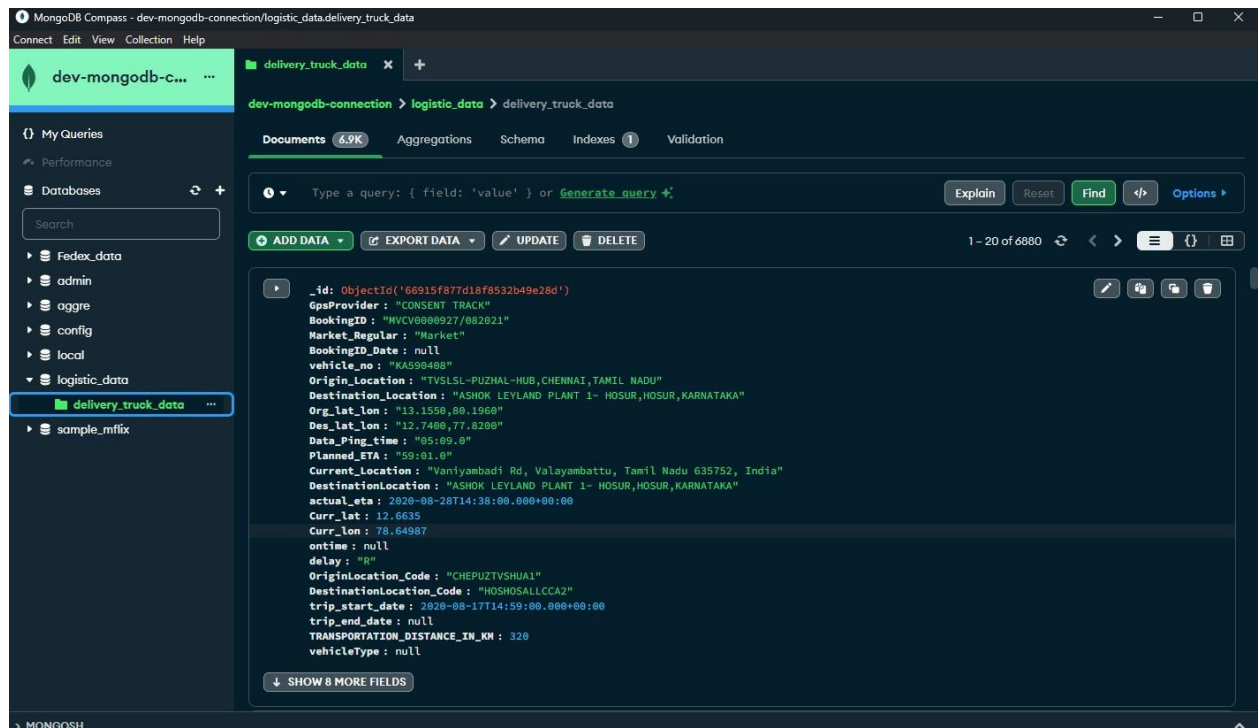


Fig. 3: MongoDB data after dumping records using 3 consumers.

Step 6

Using Flask I was able to made an APP by using little bit of Css and html forms which run on 5000 port on localhost by opening in browser we can see the output.

This is a snapshot of the code running.

```
28 # Endpoint to process aggregation pipeline form submission
29 @app.route('/aggregate', methods=['POST'])
30 def aggregate_documents():
31     try:
32         pipeline = request.form.get('pipeline', '') # Get pipeline from form data
33         pipeline = eval(pipeline) # Convert string to Python object (dict/list)
34
35         # Perform aggregation using the submitted pipeline
36         results = list(collection.aggregate(pipeline))
37
38         # Return JSON response or render template with results
39         return render_template('index.html', results=dumps(results))
40
41     except Exception as e:
42         return jsonify({"error": str(e)}), 500
43
44 if __name__ == '__main__':
45     app.run(debug=True)
46
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
PS E:\GrowDataskills\MongoDB\mongo_Assignment> python -u "e:\GrowDataskills\MongoDB\mongo_Assignment\APP\flaskmongo.py"
* Serving Flask app 'flaskmongo'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 618-571-020
[]
```

Fig. 4: Running Flask Application on localhost:5000

Below image shows the flask app with a aggregation pipeline (i.e. to get the top 3 suppliers who provide most vehicles).

Complete Data Engineering 3.0 x | (600) YouTube x | omkardesai98/kafka_mongodb x

localhost:5000

MongoDB Aggregation Pipeline Form

Enter MongoDB Aggregation Pipeline:

```
[
  {
    "$group": {
      "_id": "$supplierNameCode",
      "count": {
        "$sum": 1
      }
    }
  },
  {"$sort": {'count': -1}},
  {"$limit": 3}
]
```

Submit

Fig. 5: Running Flask Application with aggregation pipeline

Complete Data Engineering 3.0 x | (600) YouTube x | omkardesai98/kafka_mongodb x | Confluent Cloud x | MongoDB Aggregation Pipeline x +

localhost:5000/aggregate

MongoDB Aggregation Pipeline Form

Enter MongoDB Aggregation Pipeline:

Submit

Aggregation Results:

```
[{"supplier_name": "Unknown", "num_vehicles": 742}, {"supplier_name": "A S TRANSPORTS", "num_vehicles": 496}, {"supplier_name": "SUNITA CARRIERS PRIVATE LIMITED", "num_vehicles": 496}]
```

Fig. 6: Aggregation result after clicking on submit button

Reference: