

# DataEng: Data Transport Activity

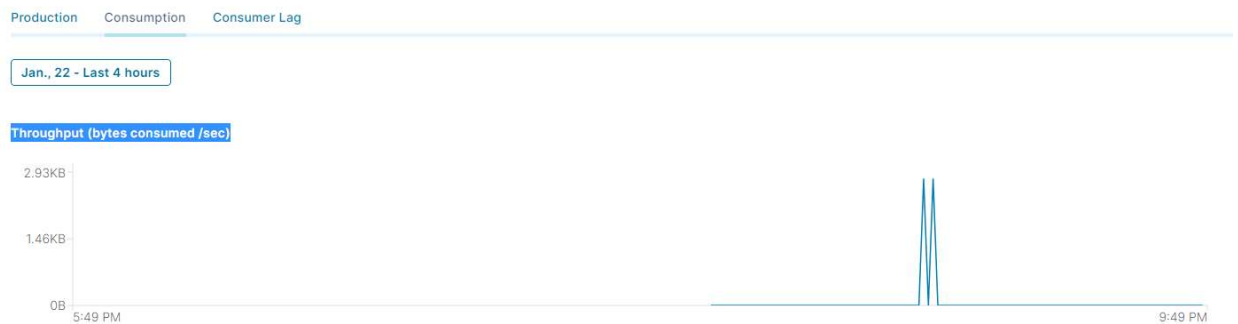
Student: Kaushik Chaudhary

## B. Kafka Monitoring

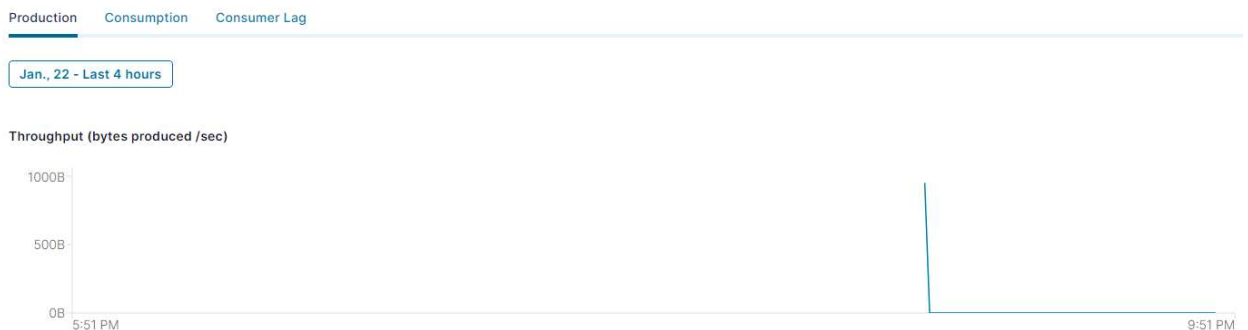
1. Find the Kafka monitoring console for your topic. Briefly describe its contents. Do the measured values seem reasonable to you?

It displays data production and consumption, with history.

### Throughput (bytes consumed /sec)



### Metrics



## C. Kafka Storage

1. Run the linux command "wc bcsample.json". Record the output here so that we can verify that your sample data file is of reasonable size.

**0 240736 2403605 bcsample.json**

2. What happens if you run your consumer multiple times while only running the producer once?

3.

**Producer will keep polling for data, which can be confirmed through below message.**

**“Waiting for message or event/error in poll ()”**

4. Before the consumer runs, where might the data go, where might it be stored?

**Data will be stored on Kafka cluster Queue waiting for consumption and can be removed with auto clean up.**

5. Is there a way to determine how much data Kafka/Confluent is storing for your topic? Do the Confluent monitoring tools help with this?

**2.96MB/ 5TB data storage in kafka. Yes, the confluent monitoring tools shows this information.**

6. Create a “topic\_clean.py” consumer that reads and discards all records for a given topic. This type of program can be very useful during debugging.

## D. Multiple Producers

1. Clear all data from the topic

**To delete all the messages from a kafka topic, you need to change the retention time of that topic the default retention time of Kafka topic is 168 hours, i.e. 7 days. In this case, you have to change the retention time to 1 second, after which the messages from the topic will be deleted. After changing the retention time, you can go ahead and change the retention time of the topic back to 168 hours.**

**You can change the retention of the topic by using the following command:**

```
kafka-topics.sh --zookeeper localhost:2181 --alter --topic topic1  
--config retention.ms=1000
```

2. Run two versions of your producer concurrently, have each of them send all 1000 of your sample records. When finished, run your consumer once. Describe the results.

**Both the producers together produced around 951K bytes/sec ,2000 messages and the consumer has consumed all the messages from message queue.**

## E. Multiple Concurrent Producers and Consumers

1. Clear all data from the topic

**Deleted the topic**

2. Update your Producer code to include a 250 msec sleep after each send of a message to the topic.

**producer.poll(0.25)**

3. Run two or three concurrent producers and two concurrent consumers all at the same time.
4. Describe the results.

**Ran two producers and two consumers concurrently with a 250msec latency on producer code. The messages produced by both the producers were pushed to the topic and I could see the consumers consuming them but one consumer after a few seconds gives poll result (stops consuming completely) whereas other consumer is still consuming the data.**

## F. Varying Keys

1. Clear all data from the topic

**Deleted the topic**

So far you have kept the “key” value constant for each record sent on a topic. But keys can be very useful to choose specific records from a stream.

2. Update your producer code to choose a random number between 1 and 5 for each record’s key.
3. Modify your consumer to consume only records with a specific key (or subset of keys).
4. Attempt to consume records with a key that does not exist. E.g., consume records with key value of “100”. Describe the results
5. Can you create a consumer that only consumes specific keys? If you run this consumer multiple times with varying keys then does it allow you to consume messages out of order while maintaining order within each key?

## G. Producer Flush

The provided tutorial producer program calls “`producer.flush()`” at the very end, and presumably your new producer also calls `producer.flush()`.

1. What does `Producer.flush()` do?

**Invoking this method makes all buffered records immediately available to send (even if `linger_ms` is greater than 0) and blocks on the completion of the requests associated with these records. The post-condition of `flush()` is that any previously sent record will have completed (e.g. `Future.is_done() == True`). A request is considered completed when either it is successfully acknowledged according to the ‘acks’ configuration for the producer, or it results in an error.**

2. What happens if you do not call `producer.flush()`?

**Producer will not be synchronous.**

3. What happens if you call `producer.flush()` after sending each record?

**It will block message until current message delivered, which will take long time to send all records.**

4. What happens if you wait for 2 seconds after every 5th record send, and you call flush only after every 15 record sends, and you have a consumer running concurrently? Specifically, does the consumer receive each message immediately? only after a flush? Something else?

**Only after flush.**

## H. Consumer Groups

1. Create two consumer groups with one consumer program instance in each group.
2. Run the producer and have it produce all 1000 messages from your sample file.
3. Run each of the consumers and verify that each consumer consumes all of the 50 messages.
4. Create a second consumer within one of the groups so that you now have three consumers total.
5. Rerun the producer and consumers. Verify that each consumer group consumes the full set of messages but that each consumer within a consumer group only consumes a portion of the messages sent to the topic.

## I. Kafka Transactions

6. Create a new producer, similar to the previous producer, that uses transactions.
7. The producer should begin a transaction, send 4 records in the transactions, then wait for 2 seconds, then choose True/False randomly with equal probability. If True then finish the transaction successfully with a commit. If False is picked then cancel the transaction.
8. Create a new transaction-aware consumer. The consumer should consume the data. It should also use the Confluent/Kafka transaction API with a "read\_committed" isolation level. (I can't find evidence of other isolation levels).
9. Transaction across multiple topics. Create a second topic and modify your producer to send two records to the first topic and two records to the second topic before randomly committing or canceling the transaction. Modify the consumer to consume from the two queues. Verify that it only consumes committed data and not uncommitted or canceled data.