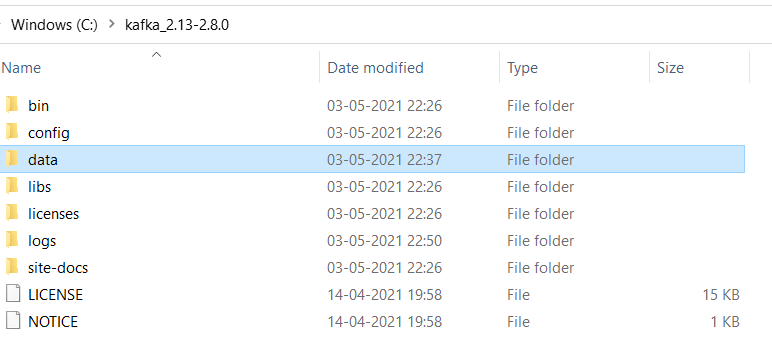
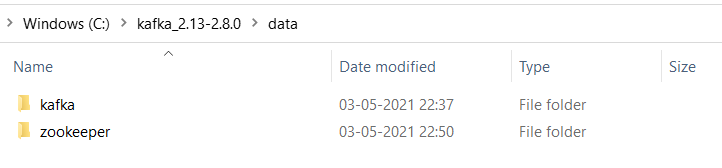
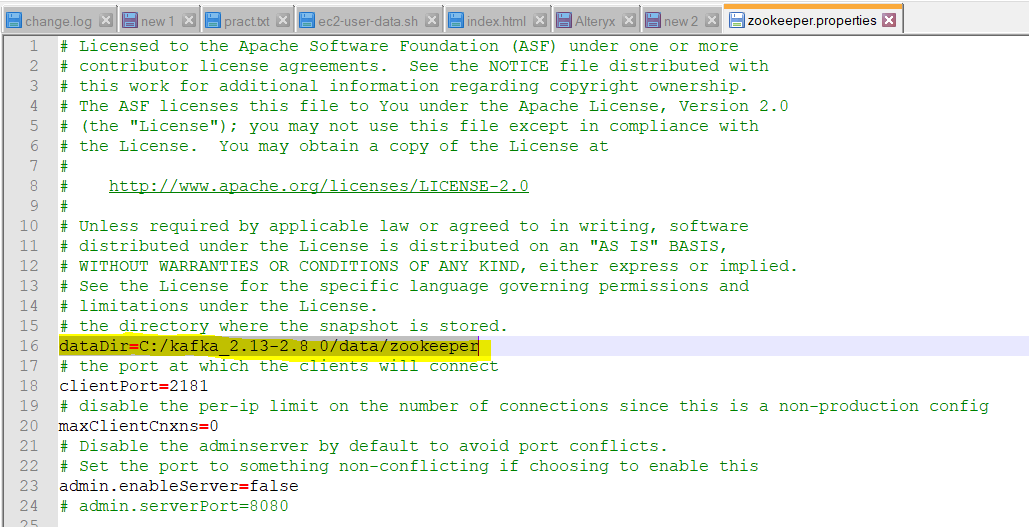
Kafka Folder Structure



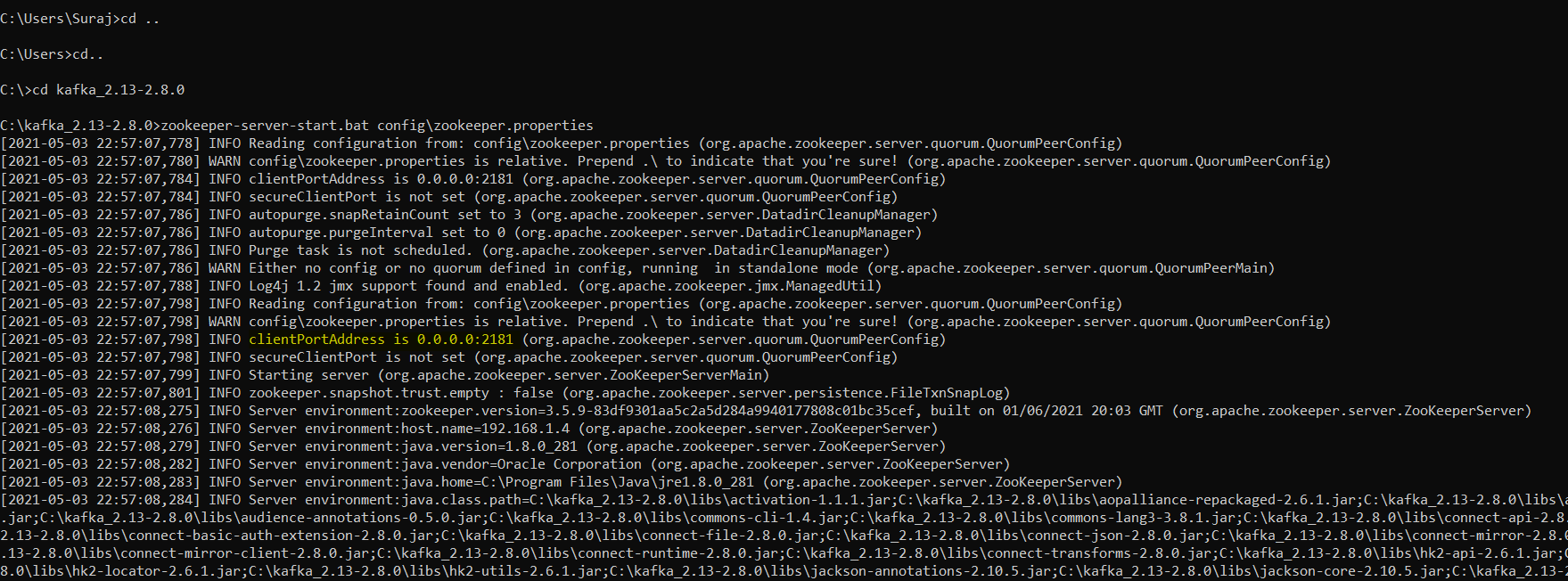
Create Separate Folders for Kafka and Zookeeper



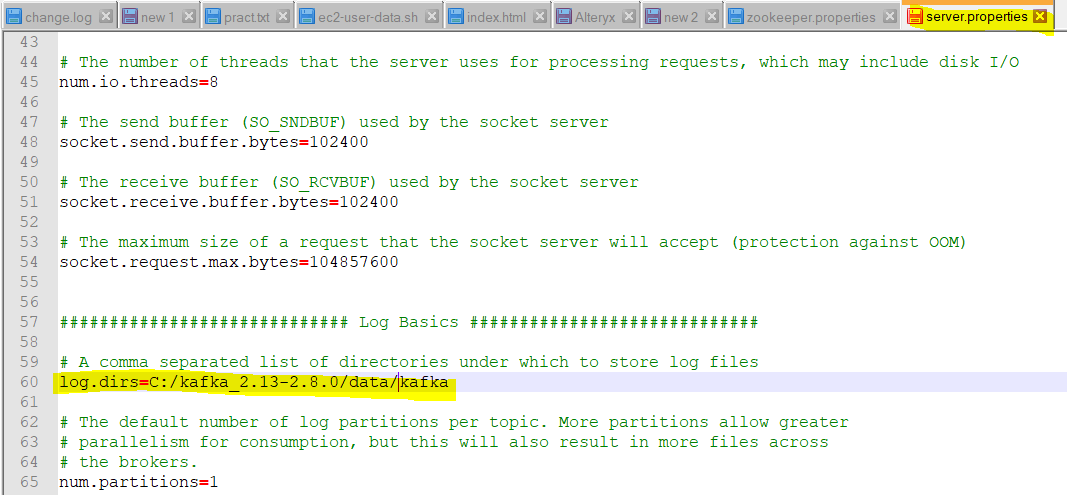
Copy the Zookeper Folder path above and change the dataDir in Zookeeper.properties file:



ZooKeeper Launch

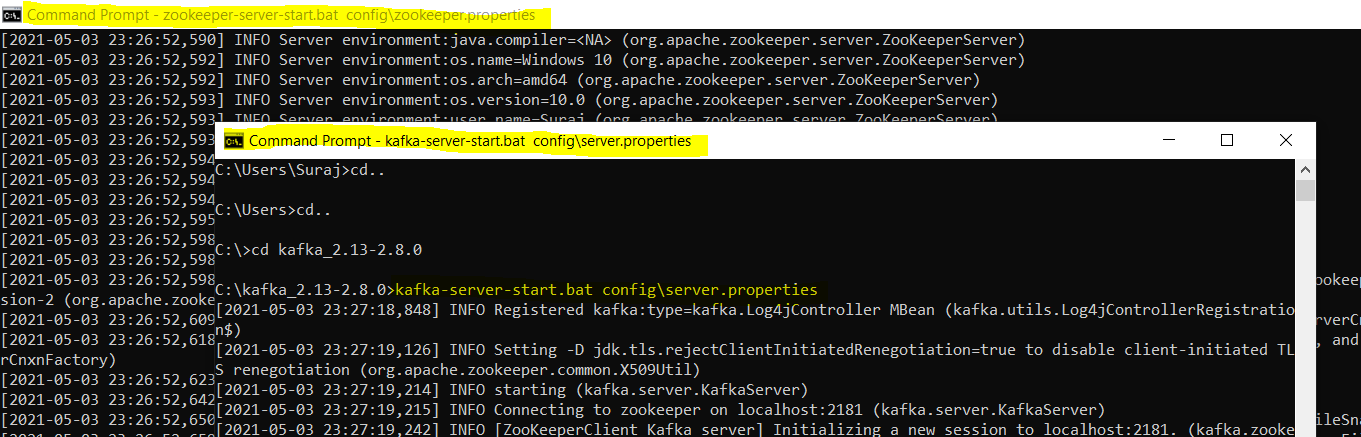


Copy the kafka folder path and change the log.dirs in server.properties



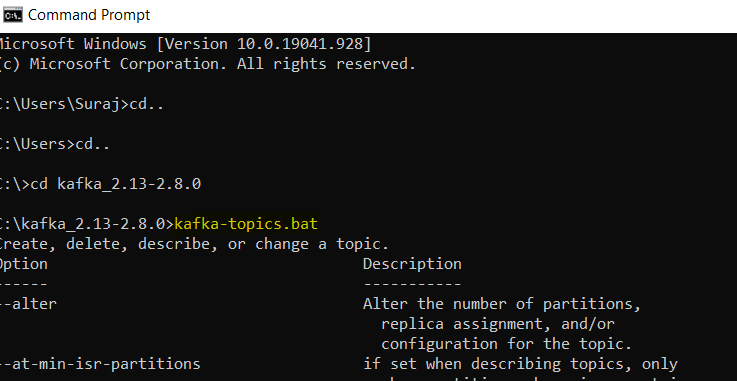
Open another prompt and do the same for sever:

The Kafka cmd implies one broker, so the topics cannot be replicated more than the available broker(here its just one)



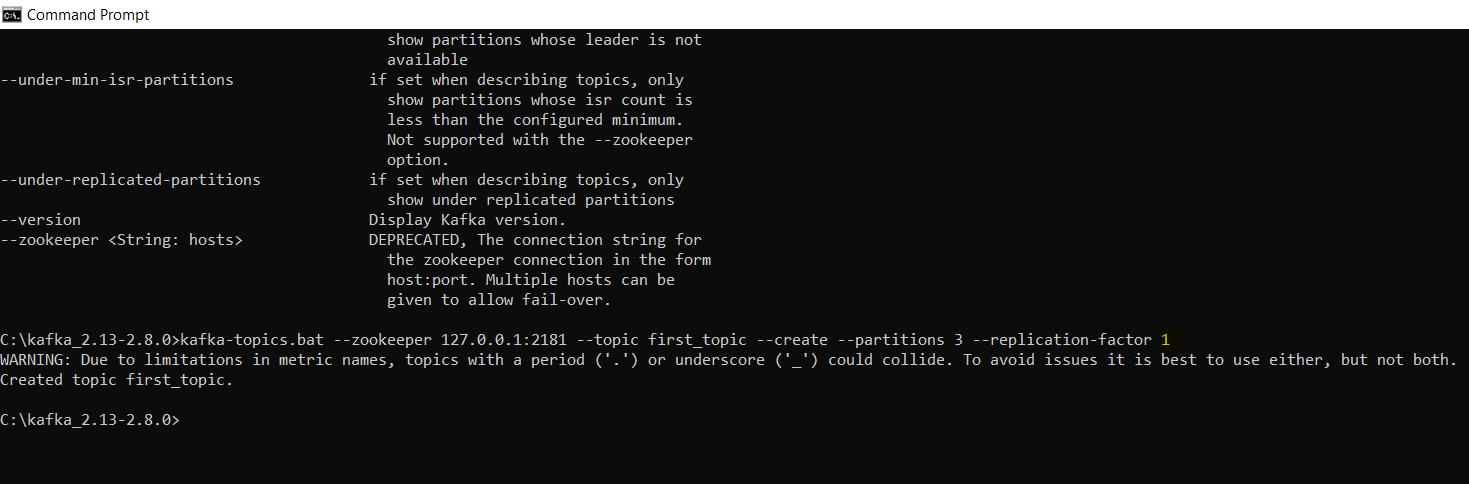
Once both the Kafka and Zookeeper launched some files get created in the kafka and zookeeper folder.

Now open another cmd and type

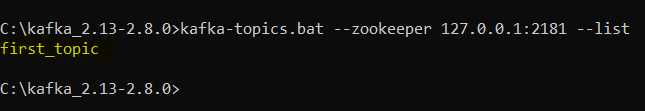


Created My first topic in kafka

In the below command if I had given replication factor as more than 1 I would get error saying that I have only one broker available. If I have to give more than 1 we need to launch another broker instance.

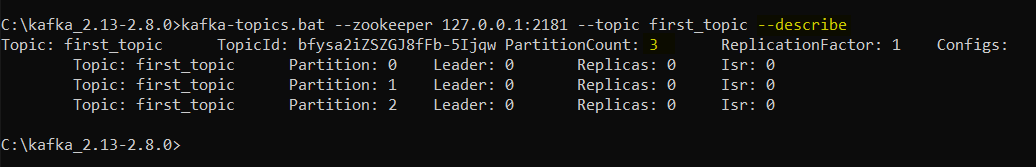


Verify If your topic is created

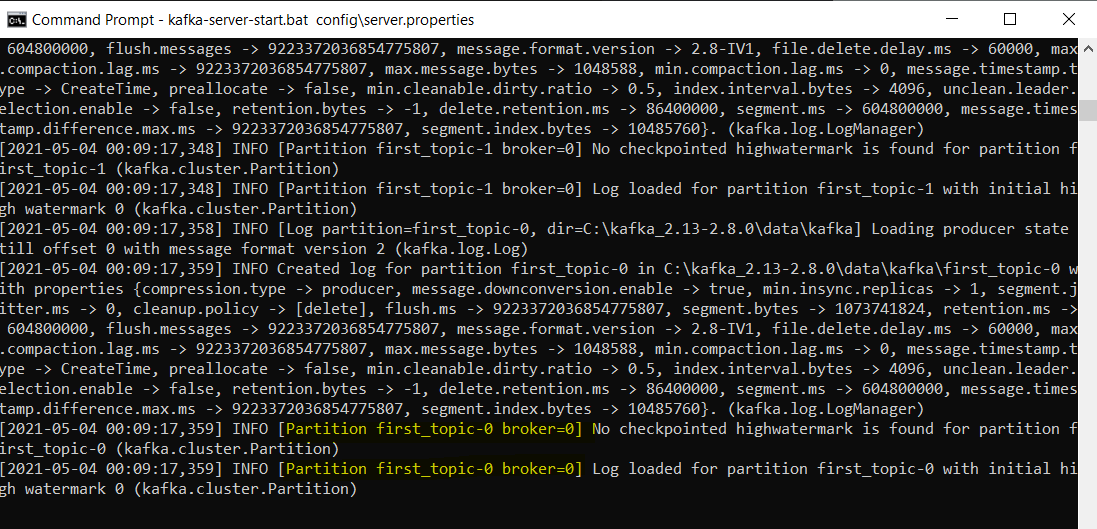


To get general info about the topic

The 0 refers to the broker id. Here broker 0 is the leader



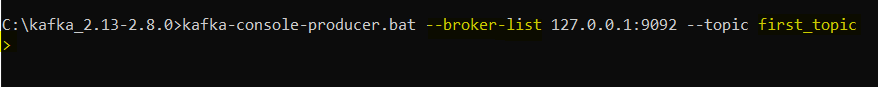
We can verify the same in our kafka broker promt



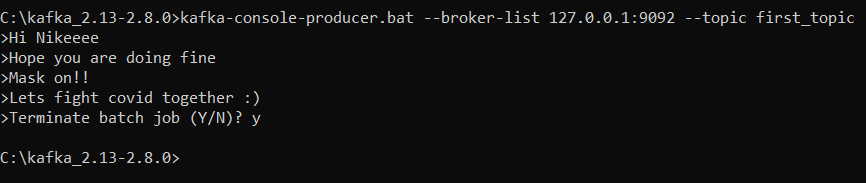
Windows users should not delete Kafka topic as it crashes the kafka. That’s the downside of using windows for Kafka.

Lets create our first producer:

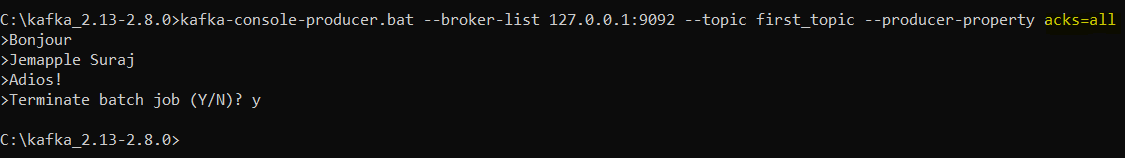
The producer produces the messages to the broker in turn to the topic that is mentioned in the command



Here are few messages produced by producer

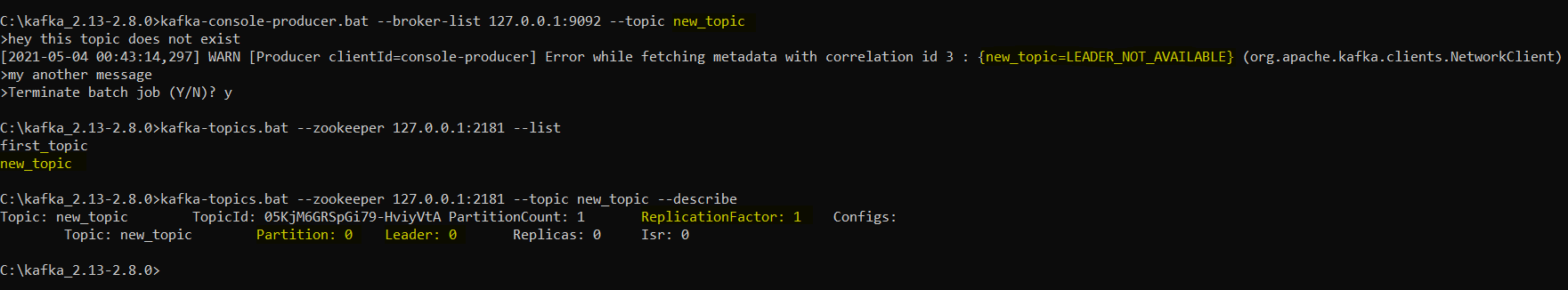


If we want acknowledgement from the leader and follower partitions implying that the messages has been received from producer set the acks=all parameter



What if we try to produce messages to the topic that is not created:

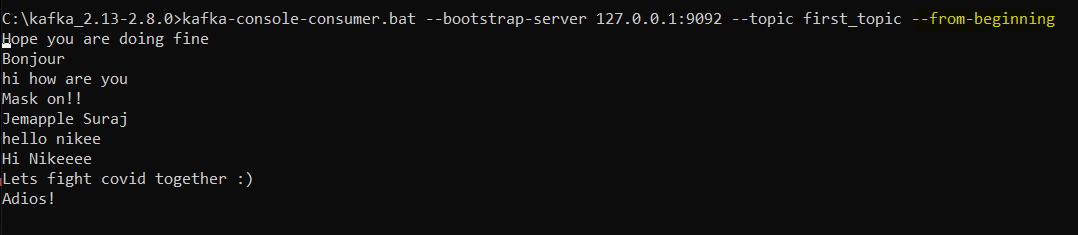
Here we produced message to the new\_topic(not created) and tried to produce messages. It threw a warning for first message. The producer has capability to recover. The new\_topic is assigned the default partitions and replication-factor. The second message we produce would not throw any warning. We can change the default number of partitions in the server.properties file



Lets see how the Producer and Consumer operate in sync..

Paste Gif

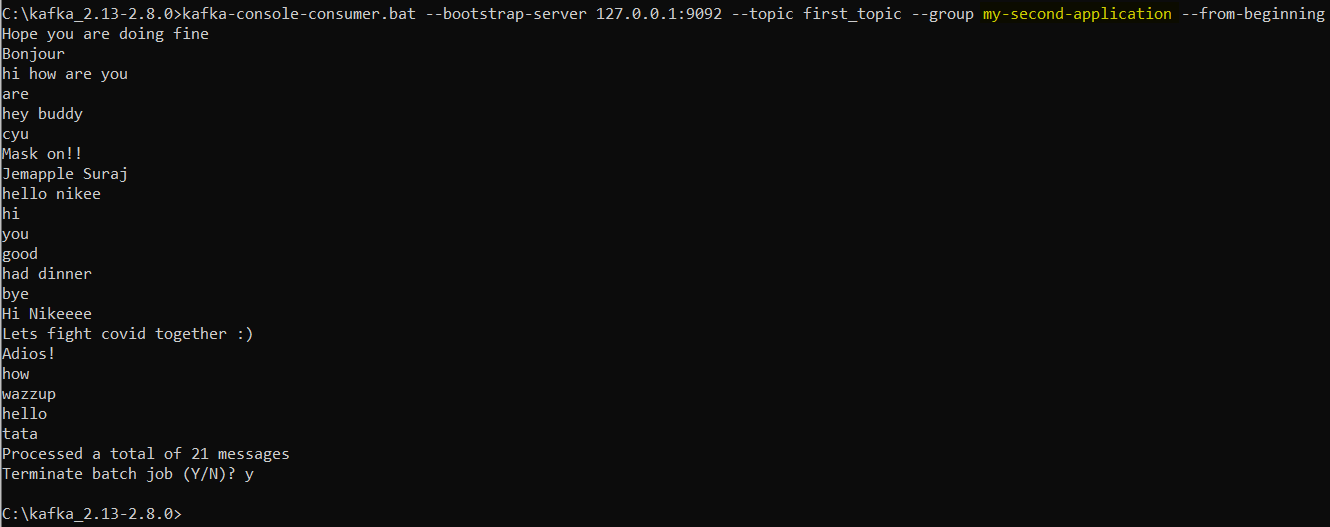
Kafka process the real time messages in the topics. But sometimes we need to fetch all the data in the topic from beginning. Here we consumed all the messages that this topic streamed. The order of messages is the not of the total. The ordering happens at partition level. So you might feel the messages are shuffled up. If we just had single partition then we could expect the same order that we had produced.



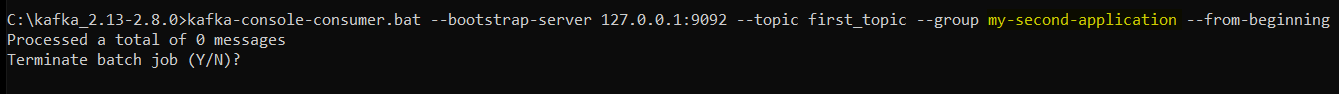
Consumer Group – We can group set of console consumer under one group name. When we launch several console consumer under the same group and when start producing messages from producers the messages get shared across the console consumers as they have same group.

Paste GIF

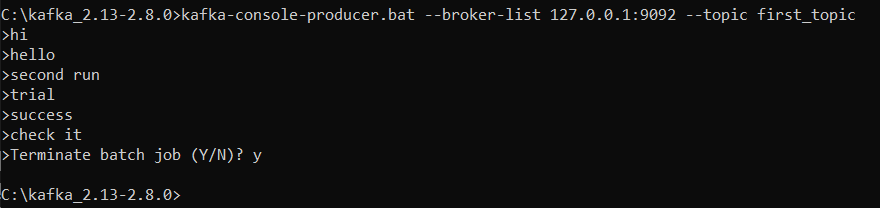
Now we create new console consumer group and consume from the beginning. Obviously it will show all the messages from the beginning.



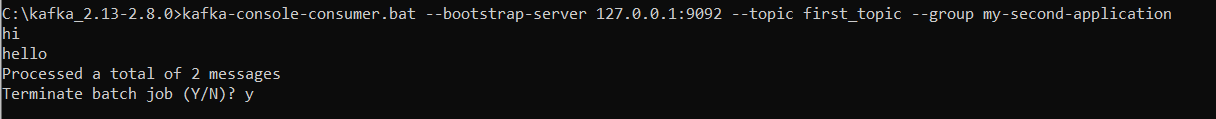
If I run the above command again I wont get any messages(assuming no producer produced any msg). Even though we specified from beginning it wont display all because offset has been committed in Kafka. The my-second -application group has already run all the 29 messages so the offset is checkpointed till there. If again any new msgs are produced then it will consume them.



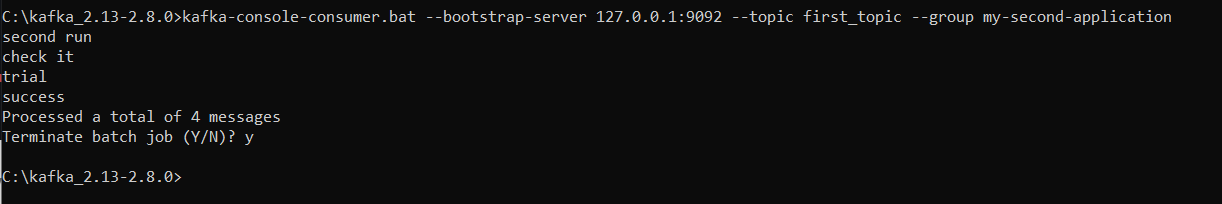
At first we just produced two messages:



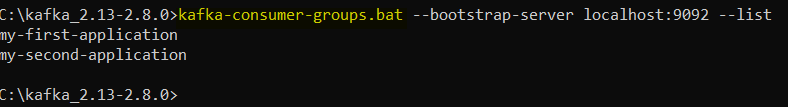
The Same is reflected by the consumer



The consumer was terminated and 3 new messages were produced. The consumer group started again and consumed only the new produced messages as it had offset committed previous two messages.

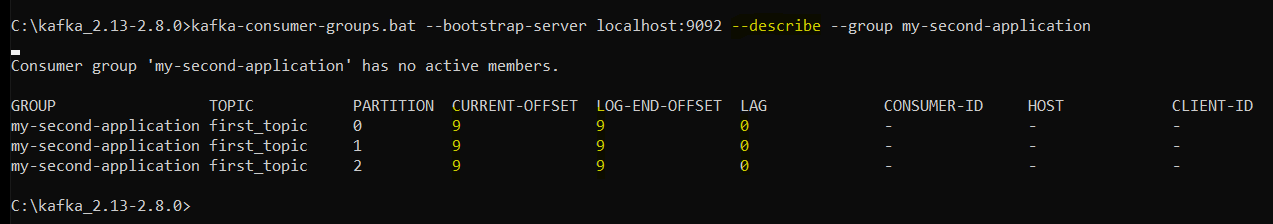


To list the consumer-groups:

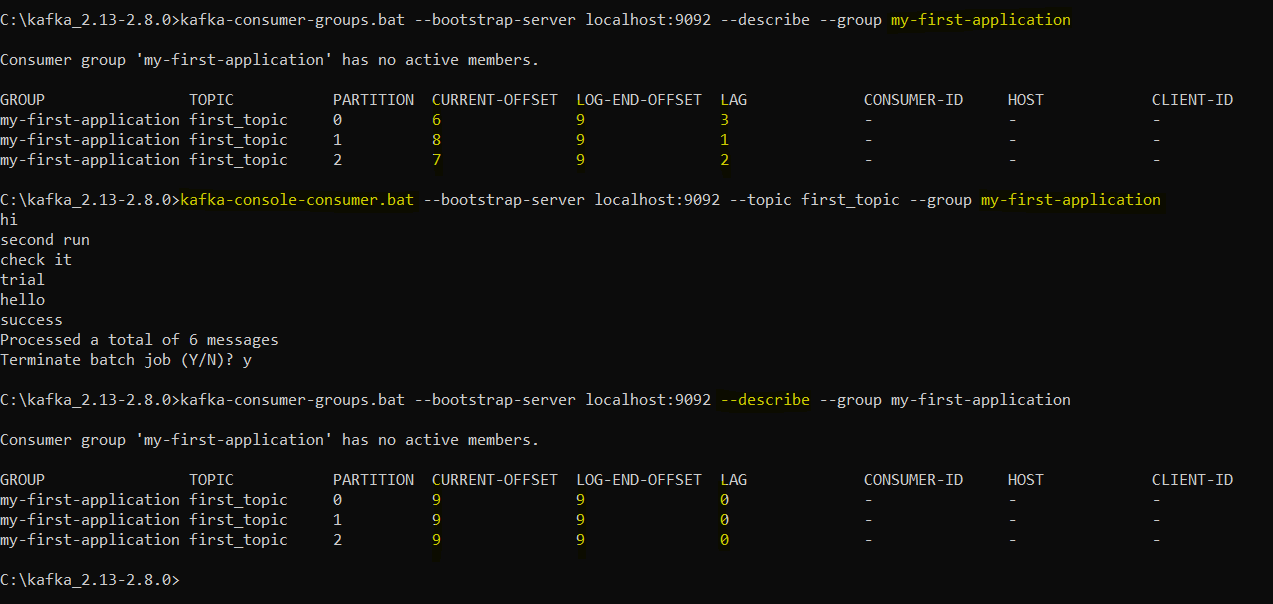


To describe any consumer group:

Here the LAG = LOG-END-OFFSET – CURRENT-OFFSET. Lag implies how many pending messages yet to consume. Here the LAG is 0 for all our partitions. So the ‘my-second-application’ is up to date in consuming messages from the producer.

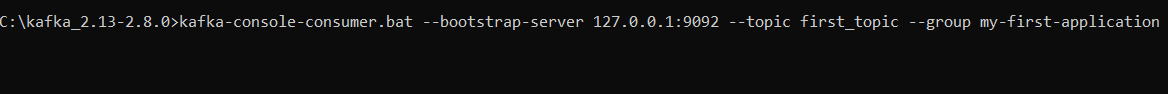


In case of my-first-application consumer group this is not the case. This consumer group has some lags across partitions. The total lag is 6(3+1+2). So I open the console consumer for this group and consume those pending messages. You can see that there are 6 messages which tallies with the total lag number. Now again if I describe the consumer group we can see that lag is 0. Now our consumer group is up-to-date

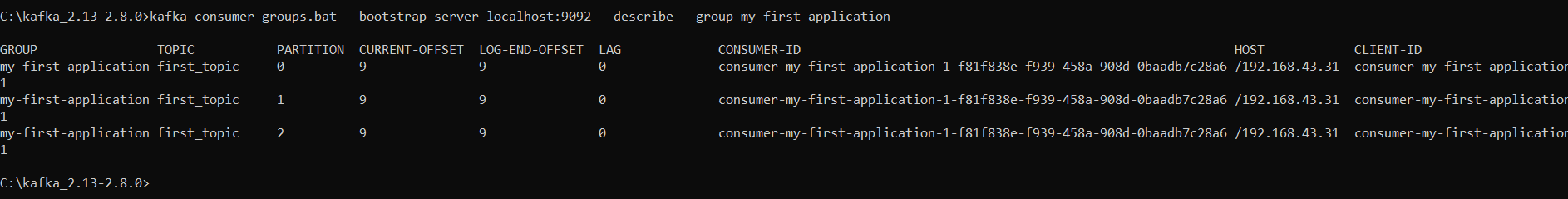


Inorder to know the consumerIDs consuming the messages in the consumer group.

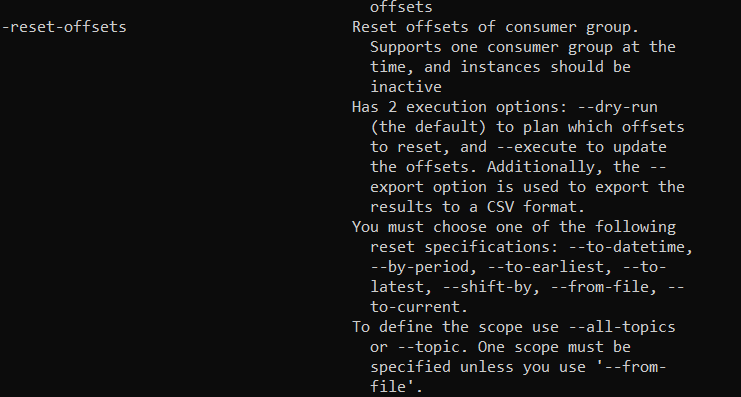
First run the console consumer command for that group.



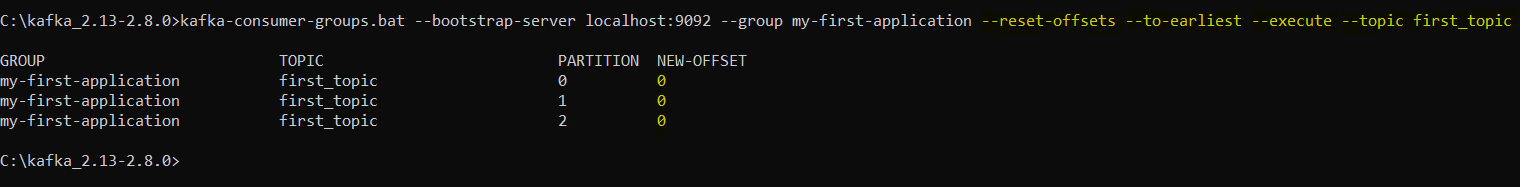
Second run the describe command for this consumer group:



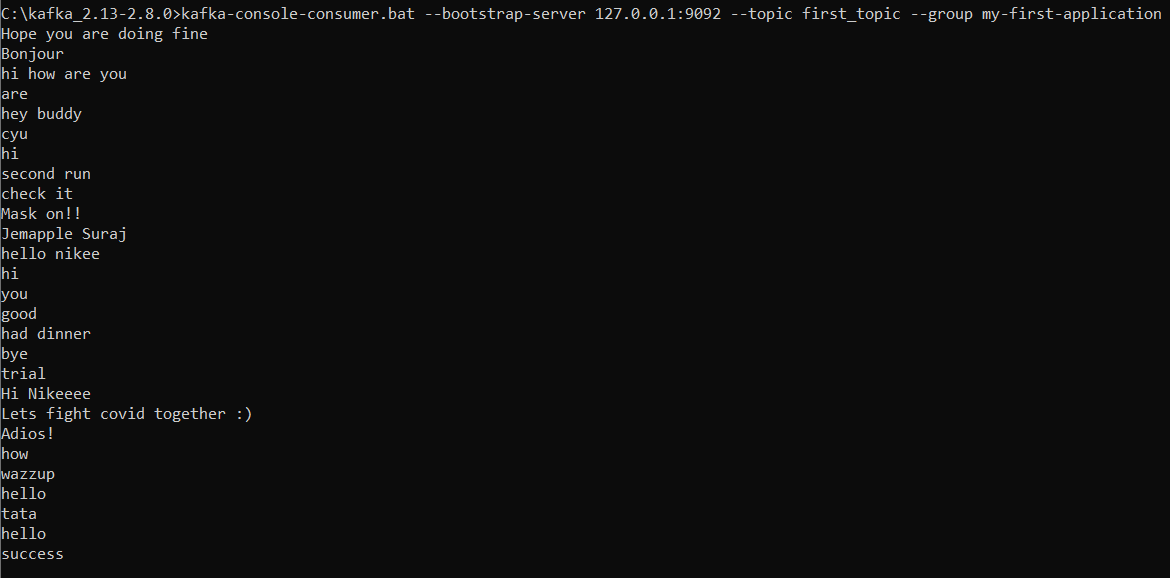
Sometime we would want to reset the offsets to consume all the messages from the starting offset to the current offset



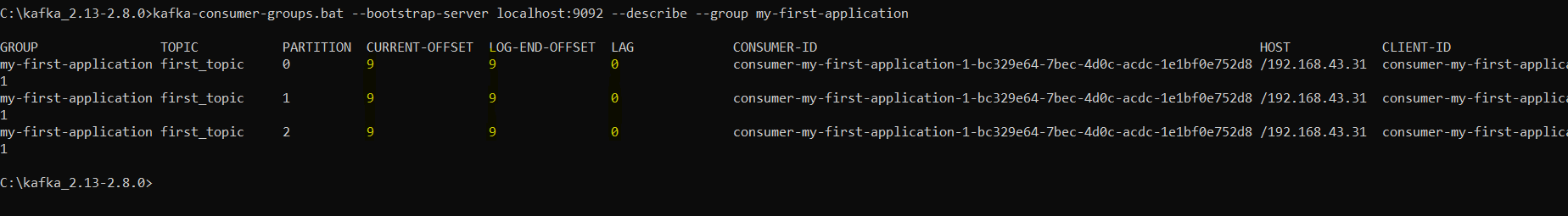
To reset the offsets to the beginning of each partition



Run the consumer-console for the group again we will see all the messages from 0th offset



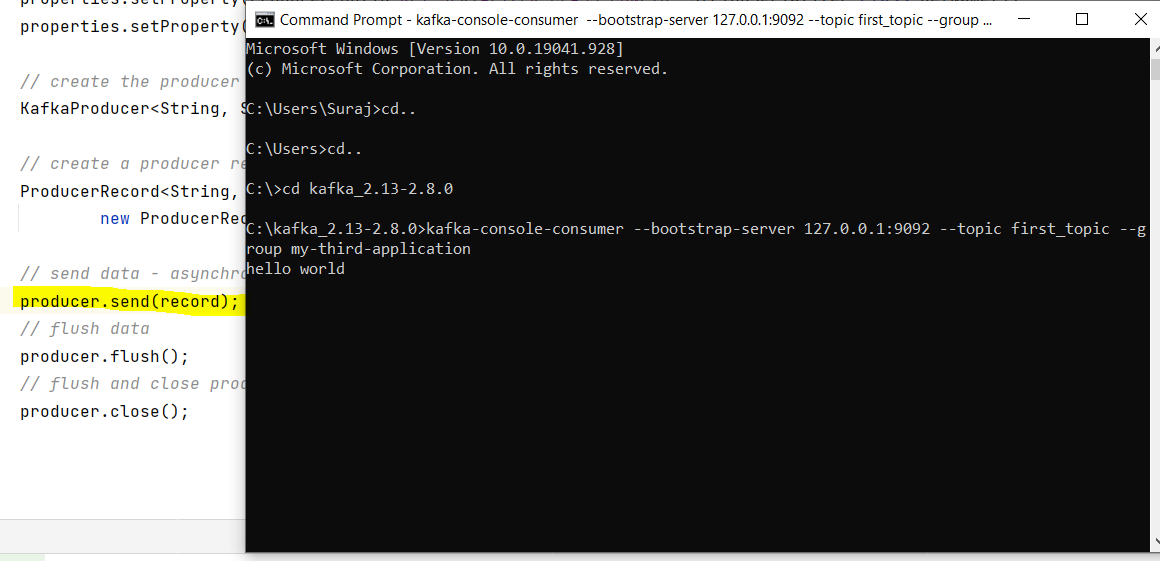
And if we describe our consumer group we see the lag to be 0



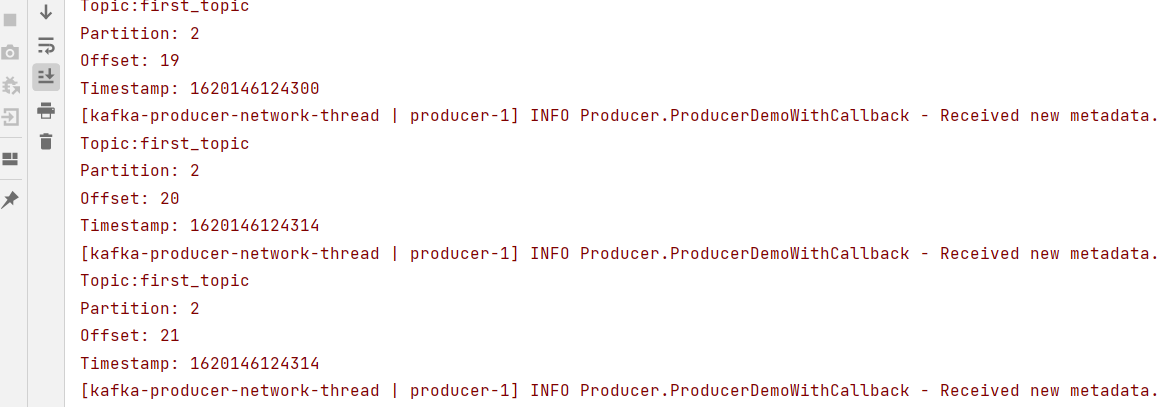
Kafka Java Programming:

Producer-Code:

On executing the producer code in java we can see the console consumer consuming the data.

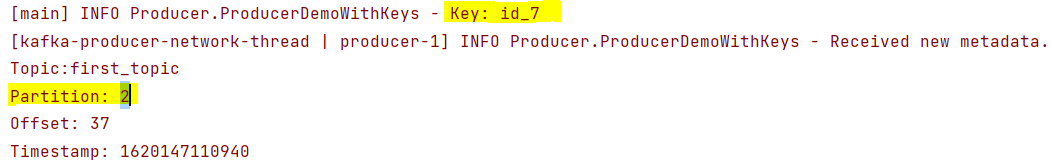


Kafka Producer with Callback

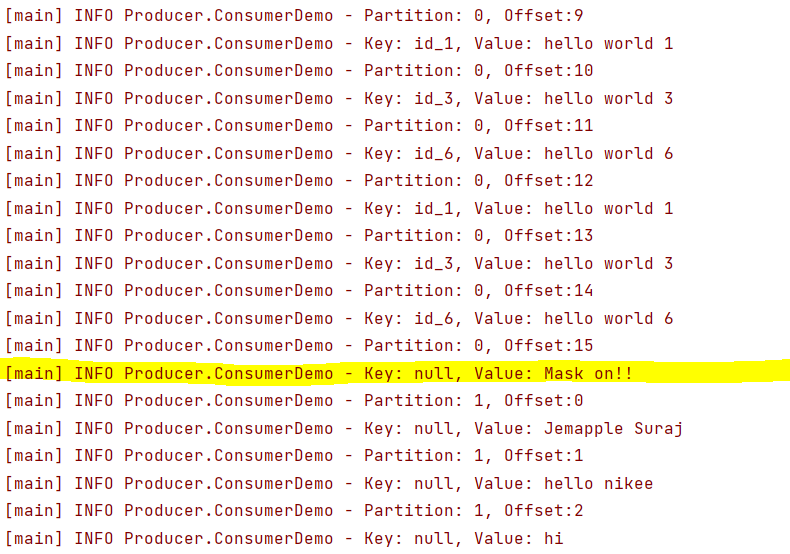


Kafka producer with keys

Each key will always go to the same partition. If we execute it again, the key id\_7 will go to partition 2 only

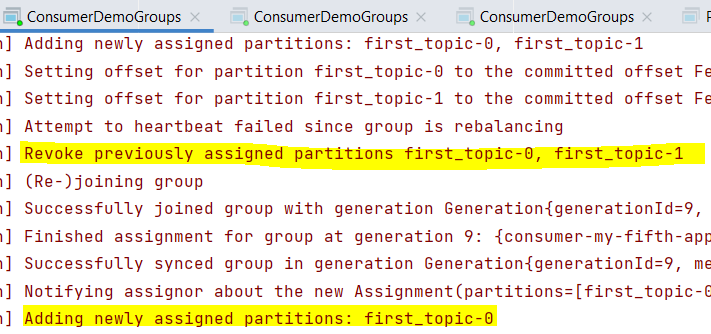


Once all the messages from partition 0 is read consumer starts reading it from next partition 1

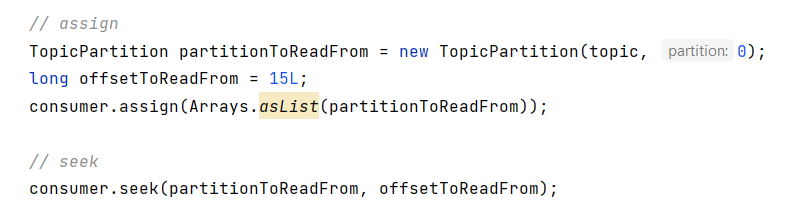


Consumers from the same group always rebalance the partitions among themselves when new consumer is created

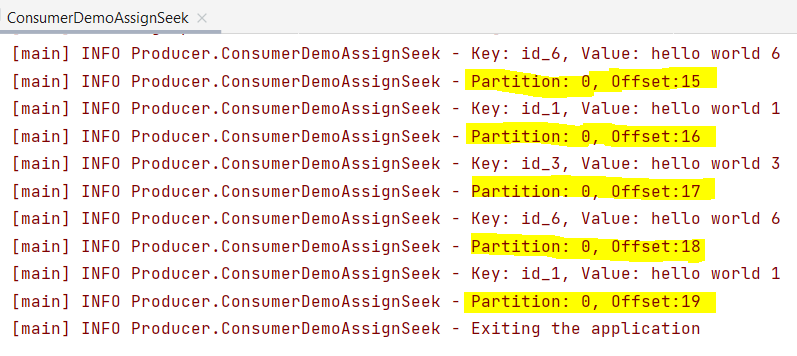
1) When there were 2 consumers and 3rd consumer got created. So each consumer will get one partition.



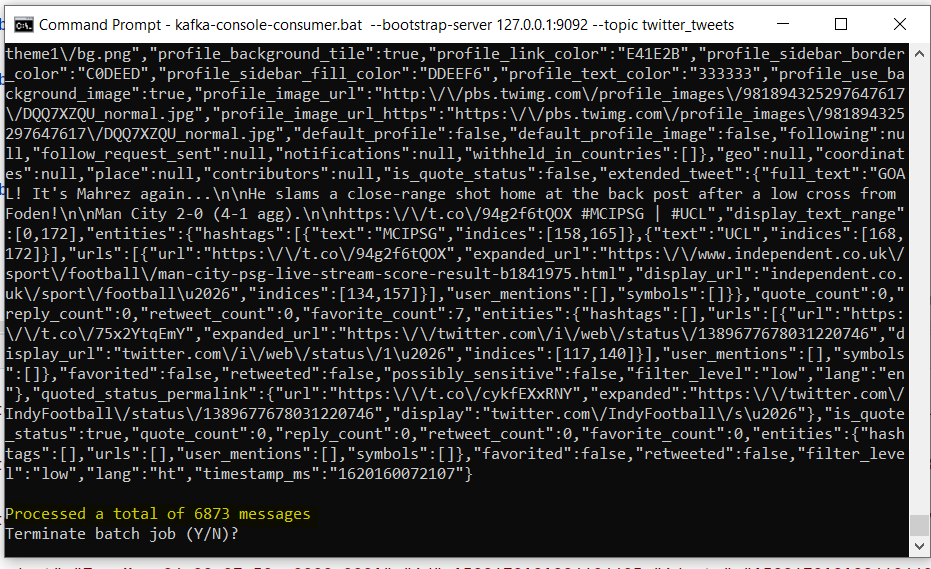
Assign and Seek: Suppose you want to consume data only from partition 0 of the first topic with offset = 15. We could do this using assign and seek method (TopicPartition library)



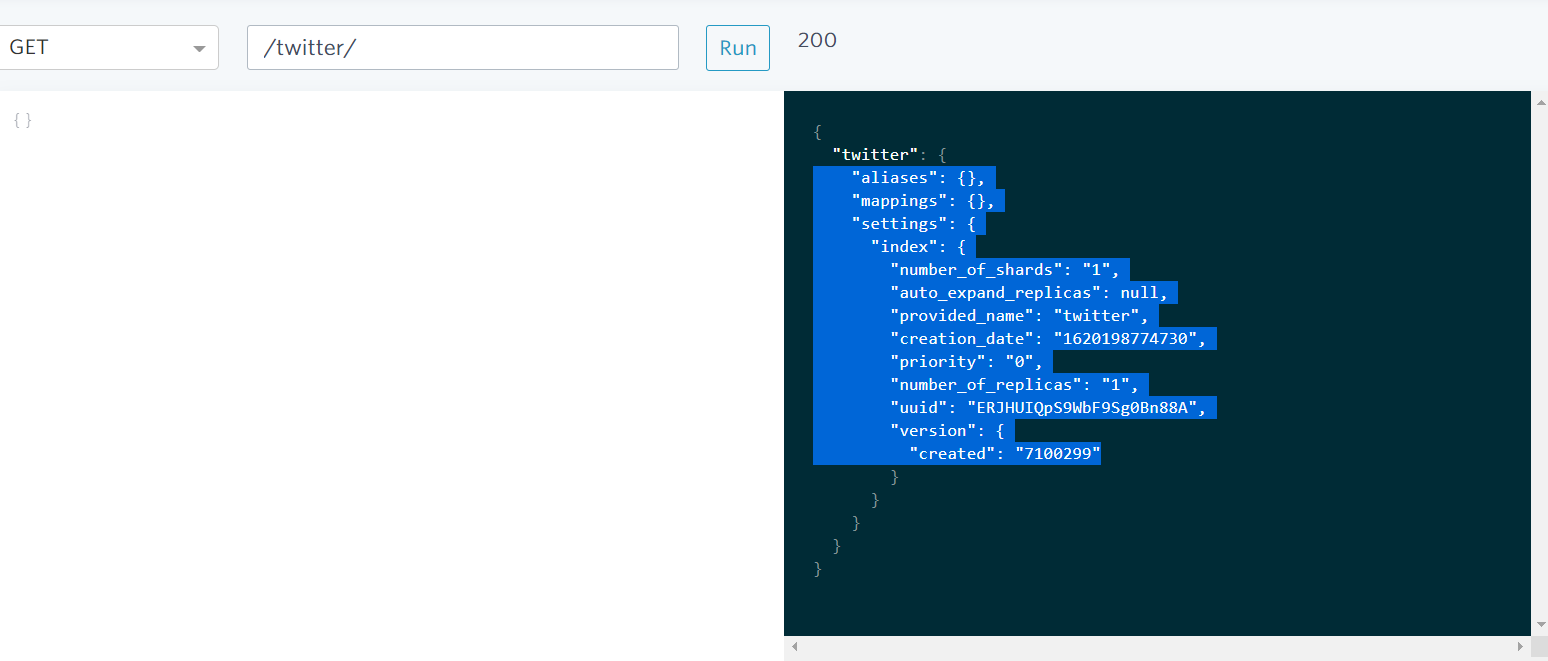
Results consisting only from partition 0 with offset 15



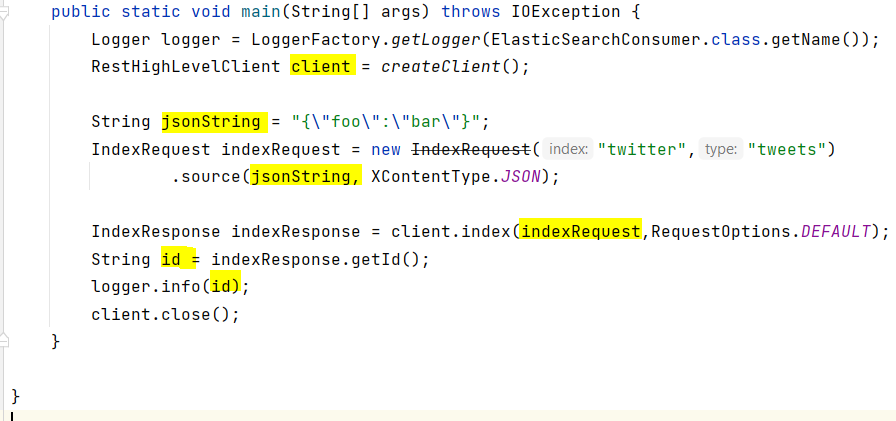
Twitter Producer-Consumer



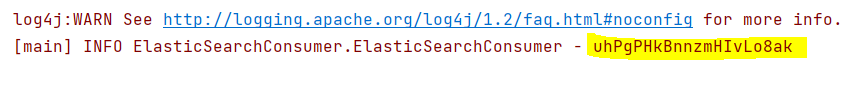
ElasticSearch cloud twitter index response to our Kafka Configuration:



Let the consumer dump single dictionary key value pair into the Elasticsearch. Here is the below code. ElasticSearch stores data in form of indexes. We created an index object that points to the “twitter” index that we already created as shown above. We use the elasticsearch client object to fetch the response id.



We fetched the index\_id of the record we inserted

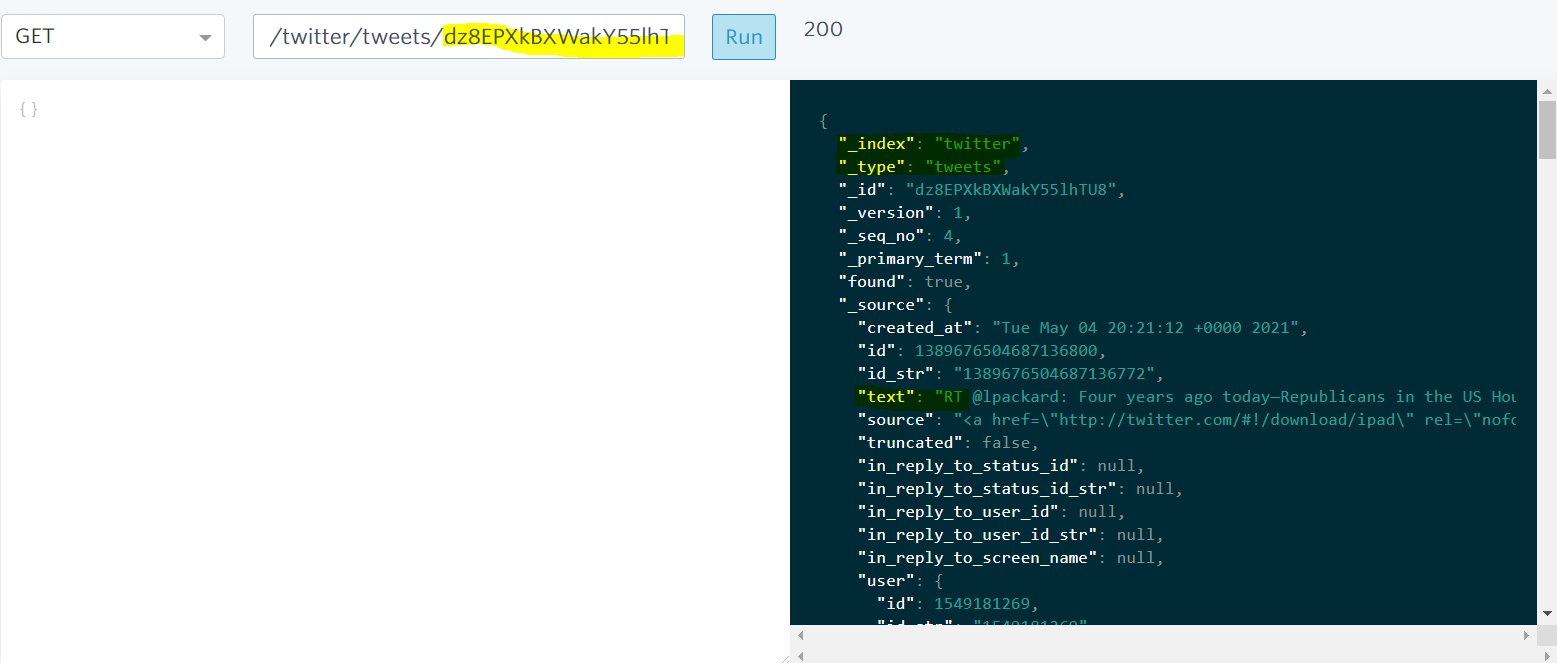


Lets use this id to verify if we the data is inserted into the elasticsearch.



We will use above concepts to dump tweets into elasticsearch from consumers.

Inserted tweets from twitter into elasticsearch

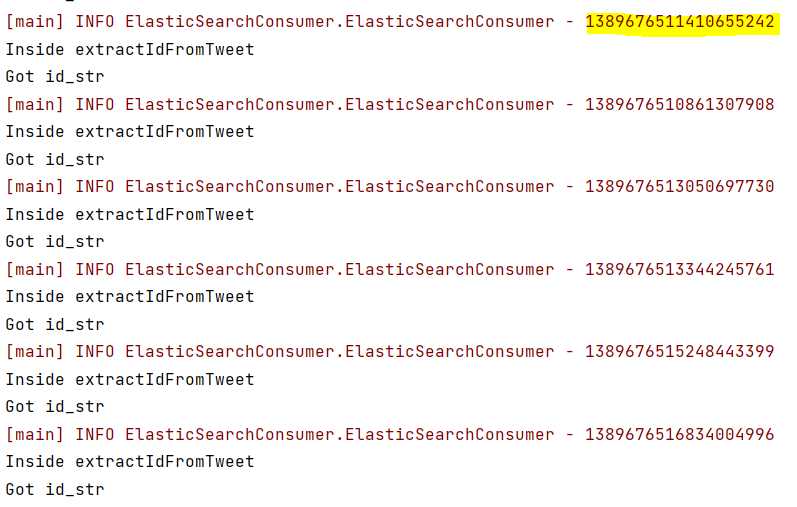


Delivery Semantics

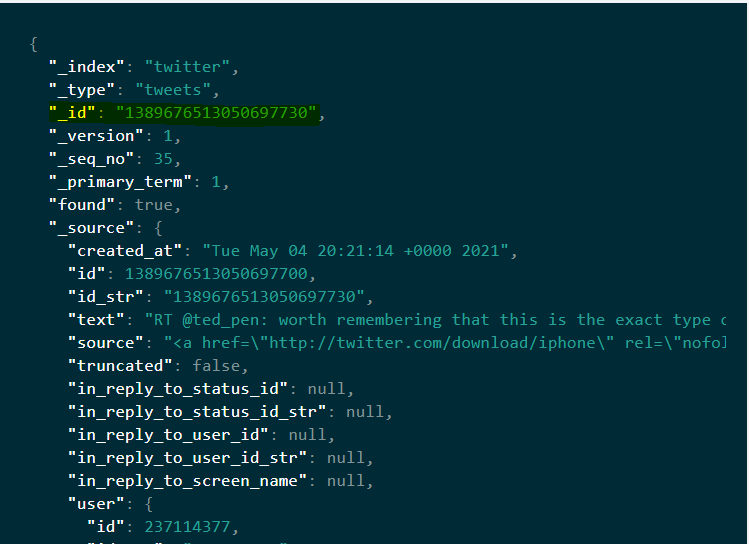
- At most once – offsets are committed as soon as the message batch is received. If the processing goes wrong message will be lost.

- At least once (default)– offsets are committed after the message is processed. If the processing goes wrong the message will be read again. This can result duplicate processing of images. Make sure your processing is idempotent. (i.e processing again the messages wont impact your systems.)

Each tweet will have unique id which could be used to make the consumer idempotent. We extract the keys as shown below:



Test for one tweet id response

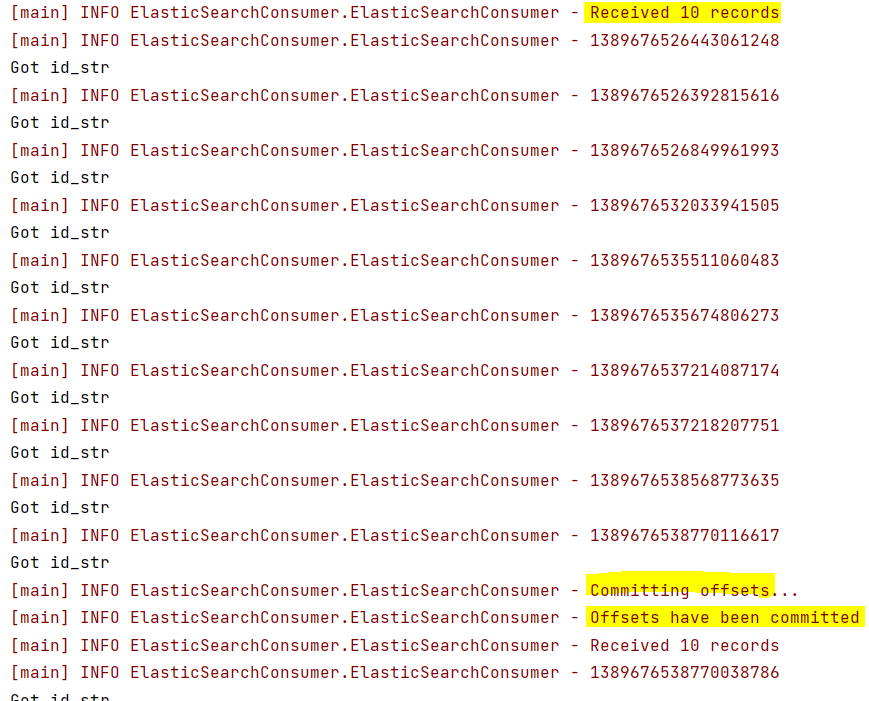


Consumer Offset Commits Strategies:

(easy) enable.auto.commit – true and synchronous processing of batches. Here the offsets will be committed automatically for us at regular intervals(5sec by default) every time you call poll

(medium) enable.auto.commit – false and manual commit of offsets. Will show “at-most-once” behaviour. We can control when to commit our offsets

Below is the results of my code which controls committing the offsets



If consumers has not read the offset in 7 days then the offsets will be lost.

To Replay the entire log reset the offset to the earliest