



Let us know how you like to work in this quick survey!

Take the Survey ▶

DZone > Big Data Zone > Case Study to Understand Kafka Consumer and Its Offsets

Case Study to Understand Kafka Consumer and Its Offsets

by Simarpreet Kaur Monga  MVB · Jan. 31, 18 · Big Data Zone · Analysis

Get \$10 for a ten-minute survey. DZone is working with GLG to help their clients learn how IT orgs deploy innovative tech so

[\\$10 And Start Survey](#)

Presented by GLG

In this post, we will discuss Kafka Consumer and its offsets. We will understand this using a case study implemented in Scala. This post assumes that you are aware of basic Kafka terminology.

In this example, the Producer is continuously producing records to the source topic. The Consumer is consuming those records from the same topic as it has subscribed to for that topic. Obviously, in a real-world scenario, the speed of the Consumer and Producer do not match. In fact, the Consumer is mostly slow in consuming records — it has some processing to do on those records. Whatever the reason, our aim for this post is to find how much our Consumer lags behind in reading data/records from the source topic.

This can be done by calculating the difference between the last offset the consumer has read and the latest offset that has been produced by the producer in the Kafka source topic.

First of all, let's make a Kafka Consumer and set some of its properties.

```
1 val properties = new Properties()
2 properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092")
3 properties.put(ConsumerConfig.GROUP_ID_CONFIG, "KafkaExampleNewConsumer")
4 properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")
5 properties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")
6 properties.put("auto.offset.reset", "latest")
7
8 val consumer = new KafkaConsumer[String, String](properties)
```

These are necessary Consumer config properties that you need to set.

- **Bootstrap_Servers config**, as specified in the Kafka documentation, is “a list of host/port pairs to use for establishing the initial connection to the Kafka cluster.” A Kafka server, by default, starts at port 9092.
- **Group_Id** is the ID of the group to which our consumer belongs.
- **Key.deserializer** and **Value.deserializer** are to specify how to deserialize the record's key and value. As my Producer serializes the record's key and value using String Serializer, I need to deserialize it using String Deserializer.

Note: You can see the code for my Kafka Producer from my GitHub repository. I am not showing the code for my Kafka Producer in this article, as we are discussing Kafka Consumers.

- **Auto.offset.reset** is a property to specify whether you want to consume the records from the beginning (earliest) or the last committed offset (latest).

I have just created my Consumer with the properties set above.

Let's now make our Consumer subscribe to a topic. To subscribe to a topic, you can use:

```
1 consumer.subscribe(util.Arrays.asList("topic-1"))
```

Here, "topic-1" is the name of my topic.

The Consumer can subscribe to multiple topics; you just need to pass the list of topics you want to consume from. For the sake of simplicity, I have just passed a single topic to consume from.

Now that the Consumer has subscribed to the topic, it can consume from that topic. The consumer maintains an offset to keep the track of the next record it needs to read.

Now, let's see how we can find the consumer offsets.

The Consumer offsets can be found using the method `offset` of class `ConsumerRecord`. This offset points to the record in a Kafka partition. The Consumer consumes the records from the

topic in the form of an object of class `ConsumerRecord`. The class `ConsumerRecord` also consists of a topic name and a partition number from which the record is being received and a timestamp as marked by the corresponding `ProducerRecord` (the record sent by the producer).

Now, the consumer can consume the data from the subscribed topic using `consumer.poll(long)`.

The method `poll` accepts a long parameter to specify timeout — the time, in milliseconds, spent waiting in the poll if data is not available in the buffer.

Note: It is an error to not have subscribed to any topics or partitions before polling for data. The consumer needs to be subscribed to some topic or partition before making a call to `poll`.

On each poll, the Consumer will try to use the last consumed offset as the starting offset and fetch sequentially.

When the Consumer polls the data from the topic, we get all the records of that topic read by the Consumer in the form of an object of class `ConsumerRecords`...

```
1 val recordsFromConsumer = consumer.poll(10000)
```

...which acts as a container to hold the list of `ConsumerRecords` per partition for a particular topic. We can retrieve all the records of a particular topic read by the Consumer as a list of `ConsumerRecords` using the method `records` of class `ConsumerRecords`.

```
1 val recordsFromConsumerList = recordsFromConsumer.records("topic-1").toList
```

Or you can do:

```
1 val recordsFromConsumerList = recordsFromConsumer.asScala.toList
```

For this, you need to import:

```
1 import scala.collection.JavaConverters._
```

To find the offset of the latest record read by the consumer, we can retrieve the last `ConsumerRecord` from the list of records in `ConsumerRecords` and then call the `offset` method on that record.

```
1 val lastOffset = recordsFromConsumerList.last.offset()
```

Now, this offset is the last offset that is read by the consumer from the topic.

Now, to find the last offset of the topic, i.e. the offset of the last record present in the topic, we can use the `endOffsets` method of `KafkaConsumer`. It gives the last offset for the given partitions. Its return type is `Map<TopicPartition, Long>`.

The last offset of a partition is the offset of the upcoming message, i.e. the offset of the last available message + 1.

```
1 val partitionsAssigned = consumer.assignment()
2 val endOffsetsPartitionMap = consumer.endOffsets(partitionsAssigned)
```

The method `endOffsets` accepts a collection of `TopicPartition`, for which you want to find the `endOffsets`.

As I want to find the `endOffsets` of the partitions that are assigned to my topic, I have passed the value of `consumer.assignment()` in the parameter of `endOffsets`. `consumer.assignment` gives the set of `TopicPartitions` that the Consumer has been assigned.

Note: You should call the method `assignment` only after calling `poll` on the consumer; otherwise, it will give null as the result. Additionally, the method `endOffsets` doesn't change the position of the consumer, unlike `seek` methods, which do change the consumer position/offset.

You can find the current position of the Consumer using:

```
1 val currentPosition = consumer.position(consumer.assignment().toList.head)
```

This method accepts a `TopicPartition` as a parameter for which you want to find the current position.

Now that we have with us the last read offset by the Consumer and the `endOffset` of a partition of the source topic, we can find their difference to find the Consumer lag.

```
1 val consumerLag = endOffsets.get(topicPartition.head) - lastReadOffset
```

Now, finally, we have the Consumer lag that we wanted in this case study thanks to the class `ConsumerRecords`, which not only lets you find the offsets but also various other useful

things.

That's all for this post. I hope you found it useful. You can download the complete code from my GitHub repository.

To know more about Kafka and its API, you can see the official site, which explains everything very clearly.

Also, if you have any questions, you can comment below. I would be very happy to help you.

Happy coding!

Get \$10 for a ten-minute survey. DZone is working with GLG to gather insights for their cli Professionals. [Start Survey](#)

Presented by GLG

Like This Article? Read More From DZone



Custom Partitioner in Kafka Using Scala: Take Quick Tour!



Custom Partitioner in Kafka: Let's Take a Quick Tour!




How to Implement a Kafka Producer



**Free DZone Refcard
Understanding Apache Spark
Failures and Bottlenecks**

Topics: KAFKA , SCALA , BIG DATA , TUTORIAL , PRODUCER , CONSUMER

Published at DZone with permission of Simarpreet Kaur Monga , DZone MVB. [See the original article here.](#) 

Opinions expressed by DZone contributors are their own.