What is a Kafka Consumer?

A Consumer is an application that reads data from Kafka Topics. It subscribes to one or more topics in the Kafka cluster and feeds on tokens or messages from the Kafka Topics.

The connectivity of Consumer to Kafka Cluster is known using Heartbeat. Heartbeat is setup at Consumer to let Zookeeper or Broker Coordinator know if the Consumer is still connected to the Cluster. Absence of heartbeat means the Consumer is no longer connected to the Cluster, in which case the Broker Coordinator has to re-balance the load. Heartbeat is an overhead to the cluster. The interval at which the heartbeat at Consumer should happen is configurable by keeping the data throughput and overhead in consideration.

Also, consumers could be grouped and the consumers in the Consumer Group could share the partitions of the Topics they subscribed to. If there are N partitions in a Topic, N consumers in the Consumer Group, and the group has subscribed to a Topic, each consumer would read data from a partition of the topic. This is just a heads up that Consumers could be in groups. We shall go into details of Consumer Group in our next tutorial.

The Consumer API from Kafka helps to connect to Kafka cluster and consume the data streams.

Following is a picture demonstrating the working of Consumer in Apache Kafka.
1. **Create Java Project**

Create a new Java Project called KafkaExamples, in your favorite IDE. In this example, we shall use Eclipse. But the process should remain same for most of the other IDEs.

2. **Add Jars to Build Path.**

Add following jars to the Java Project Build Path

- **Note**: The jars are available in the lib folder of Apache Kafka download from [https://kafka.apache.org/downloads](https://kafka.apache.org/downloads).

  - kafka_2.11-0.11.0.0.jar
  - kafka-clients-0.11.0.0.jar
  - scala-library-2.12.3.jar
  - slf4j-api-1.7.25.jar
  - slf4j-log4j12-1.7.25.jar
  - log4j-1.2.17.jar

3. **New SampleConsumer Thread**

Create a new class for a sample Consumer, SampleConsumer.java, that extends Thread. So that Consumer could be launched as a new thread from a machine on demand.

```java
public class SampleConsumer extends Thread {
    ... 
}
```

4. **Properties of Kafka Consumer**

Provide the information like Kafka Server URL, Kafka Server Port, Consumer’s ID (Client ID), Serializers for Key and Value.

```java
Properties properties = new Properties();
    props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, KAFKA_SERVER_URL + ":" + KAFKA_SERVER_PORT);
    props.put(ConsumerConfig.GROUP_ID_CONFIG, CLIENT_ID);
    props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
    props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000");
    props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "30000");
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.IntegerDeserializer");
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");
```

**Note**: Make sure that the Server URL and PORT are in compliance with the values in /config/server.properties.

5. **Create Kafka Consumer with the Properties**

With the properties that have been mentioned above, create a new KafkaConsumer.

```java
KafkaConsumer consumer = new KafkaConsumer<> (props);
```

6. **Subscribe Consumer to a Topic**
Consumer has to subscribe to a Topic, from which it can receive records.

<table>
<thead>
<tr>
<th>Subscribe Consumer to a Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumer.subscribe(Collections.singleton(this.topic));</td>
</tr>
</tbody>
</table>

7. **Fetch Records for the Topic**

Fetch Records for the Topic that the Consumer has been subscribed to, using poll(long interval). interval is the time period over which, the records are aggregated.

<table>
<thead>
<tr>
<th>Send Messages Synchronously to Kafka Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConsumerRecords&lt;Integer, String&gt; records = consumer.poll(1000);</td>
</tr>
</tbody>
</table>

8. **Consume the records**

You may consume the records as per your need or use case. Here, in this tutorial, we shall print those messages to console output.

<table>
<thead>
<tr>
<th>Consume Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (ConsumerRecord&lt;Integer, String&gt; record : records) {</td>
</tr>
<tr>
<td>System.out.println(&quot;Received message: (&quot; + record.key() + &quot;, &quot; + record.value() + &quot;) at offset &quot;+ record.offset());</td>
</tr>
</tbody>
</table>

9. **Start Zookeeper and Kafka Cluster**

Navigate to the root of Kafka directory and run each of the following commands in separate terminals to start Zookeeper and Kafka Cluster.

```
$ bin/zookeeper-server-start.sh config/zookeeper.properties
$ bin/kafka-server-start.sh config/server.properties
```

10. **Start the Kafka Producer**

Well! There has to be a Producer of records for the Consumer to feed on. Start the Kafka Producer by following [Kafka Producer with Java Example](#). Also note that, if you are changing the Topic name, make sure you use the same topic name for the Kafka Producer Example and Kafka Consumer Example Java Applications.

11. **Start the SampleConsumer thread**

```
Start SampleProducer
Consumer consumerThread = new Consumer("testTopic");
consumerThread.start();
```

---

**Example Java Application that works as Kafka Consumer**

```
SampleConsumer.java
```
import kafka.utils.ShutdownableThread;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import java.util.Collections;
import java.util.Properties;

/**
 * Kafka Consumer with Example Java Application
 */
public class SampleConsumer extends ShutdownableThread {
    private final KafkaConsumer<Integer, String> consumer;
    private final String topic;

    public static final String KAFKA_SERVER_URL = "localhost";
    public static final int KAFKA_SERVER_PORT = 9092;
    public static final String CLIENT_ID = "SampleConsumer";

    public SampleConsumer(String topic) {
        super("KafkaConsumerExample", false);
        Properties props = new Properties();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, KAFKA_SERVER_URL + ":" + KAFKA_SERVER_PORT);
        props.put(ConsumerConfig.GROUP_ID_CONFIG, CLIENT_ID);
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
        props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000");
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "30000");
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.IntegerDeserializer");
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

        consumer = new KafkaConsumer<>(props);
        this.topic = topic;
    }

    @Override
    public void doWork() {
        consumer.subscribe(Collections.singletonList(this.topic));
        ConsumerRecords<Integer, String> records = consumer.poll(1000);
        for (ConsumerRecord<Integer, String> record : records) {
            System.out.println("Received message: (" + record.key() + ", " + record.value() + ") at offset "+ record.offset);
        }
    }

    @Override
    public String name() {
        return null;
    }

    @Override
    public boolean isInterruptible() {
        return false;
    }
}

KafkaConsumerDemo.java
/**
 * Kafka Consumer with Example Java Application
 */
public class KafkaConsumerDemo {
    public static void main(String[] args) {
        Consumer consumerThread = new Consumer("testTopic");
        consumerThread.start();
    }
}
Run KafkaConsumerDemo.java.

Kafka Producer Console Output

- message(1, Message_1) sent to partition(0), offset(111467) in 419 ms
- message(2, Message_2) sent to partition(0), offset(111468) in 80 ms
- message(3, Message_3) sent to partition(0), offset(111469) in 76 ms
- message(4, Message_4) sent to partition(0), offset(111470) in 76 ms
- message(5, Message_5) sent to partition(0), offset(111471) in 76 ms
- message(6, Message_6) sent to partition(0), offset(111472) in 75 ms
- message(7, Message_7) sent to partition(0), offset(111473) in 73 ms
- message(8, Message_8) sent to partition(0), offset(111474) in 81 ms
- message(9, Message_9) sent to partition(0), offset(111475) in 75 ms
- message(10, Message_10) sent to partition(0), offset(111476) in 75 ms

Kafka Consumer Console Output

- Received message: (1, Message_1) at offset 111467
- Received message: (2, Message_2) at offset 111468
- Received message: (3, Message_3) at offset 111469
- Received message: (4, Message_4) at offset 111470
- Received message: (5, Message_5) at offset 111471
- Received message: (6, Message_6) at offset 111472
- Received message: (7, Message_7) at offset 111473
- Received message: (8, Message_8) at offset 111474
- Received message: (9, Message_9) at offset 111475
- Received message: (10, Message_10) at offset 111476

Conclusion:

In this Apache Kafka Tutorial – Kafka Consumer with Example Java Application, we have learnt about Kafka Consumer, and presented a step by step guide to realize a Kafka Consumer Application using Java.
Apache Kafka Tutorial

- Kafka Tutorial
- Kafka Installation on Ubuntu
- Kafka Installation on Mac
- Kafka Architecture

Kafka Topic

- Create Kafka Topic
- Describe Kafka Topic

Kafka APIs

- Kafka Console Producer and Consumer Example
- Kafka Producer - Java Example
- Kafka Consumer - Java Example
- Kafka Connector - Data Source Example
- Kafka Multi-Broker Cluster

Kafka - Confluent

- Kafka - Confluent Platform
- Kafka Connector to MySQL Source using JDBC

Useful Resources

- How to Learn Programming