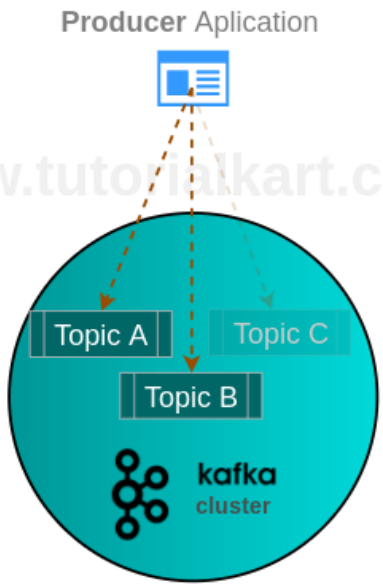
Kafka Producer – Example using Java

Learn about Kafka Producer and a Producer Example in Apache Kafka with step by step guide to realize a producer using Java.

What is a Producer in Apache Kafka ?

A producer is an application that is source of data stream. It generates tokens or messages and publish it to one or more topics in the Kafka cluster. The Producer API from Kafka helps to pack the message or token and deliver it to Kafka Server.

Following is a picture demonstrating the working of Producer in Apache Kafka.



Producer Application in Apache Kafka

Producer Example in Apache Kafka

In this Apache Kafka Tutorial, we shall learn Producer in Apache Kafka with a Java Example program. Following is a step by step process to write a simple Producer Example 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.

- 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 SampleProducer Thread

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

```
Sample Producer Thread
public class SampleProducer extends Thread {
    . . .
 }
```

4. Properties of Kafka Producer

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

```
Properties of Kafka Producer
Properties properties = new Properties();
properties.put("bootstrap.servers", KAFKA_SERVER_URL + ":" + KAFKA_SERVER_PORT);
properties.put("client.id", "DemoProducer");
properties.put("key.serializer", "org.apache.kafka.common.serialization.IntegerSerializer");
properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
```

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

5. Create Kafka Producer with the Properties

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

```
New KafkaProducer
KafkaProducer producer = new KafkaProducer&lt;&gt;(properties);
```

6. Synchronous or Asynchronous

You may send the events from Producer to the Kafka Server synchronously or asynchronously.

7. Send Messages Synchronously

You may send messages synchronously (i.e., a new message is sent only after completing the previous message/transaction) as shown below :

```
Send Messages Synchronously to Kafka Cluster
```

```
try {
    producer.send(new ProducerRecord&lt;&gt;(topic,
            messageNo,
            messageStr)).get();
 } catch (InterruptedException | ExecutionException e) {
    e.printStackTrace();
    // handle the exception
 }
```

When messages are sent synchronously, they are prone to interruption or stoppage of their transmission to the Kafka Server. InterruptedException and ExecutionException thrown by the send function have to be handled.

8. Send Messages Asynchronously

You may send messages asynchronously as shown below :

```
Send Messages Asynchronously to Kafka Cluster
producer.send(new ProducerRecord&lt;&gt;(topic,
        messageNo,
        messageStr), new DemoCallBack(startTime, messageNo, messageStr));
```

When a message is sent asynchronously, you need to provide a CallBack class that implements onCompletion() method which is called when a message is sent successfully and acknowledged by Kafka Server. We have provided a DemoCallBack class here for the call back purpose.

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 SampleProducer thread

```
Start SampleProducer
SampleProducer producerThread = new SampleProducer(TOPIC, isAsync);
producerThread.start();
```

Complete Java Producer Example in Apache Kafka

```
SampleProducer.java
import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;

import java.util.Properties;
import java.util.concurrent.ExecutionException;

/**
 * Producer Example in Apache Kafka
 * @author www.tutorialkart.com
 */
public class SampleProducer extends Thread {
    private final KafkaProducer&lt;Integer, String&gt; producer;
```

```
    private final String topic;
    private final Boolean isAsync;

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

    public SampleProducer(String topic, Boolean isAsync) {
        Properties properties = new Properties();
        properties.put("bootstrap.servers", KAFKA_SERVER_URL + ":" + KAFKA_SERVER_PORT);
        properties.put("client.id", CLIENT_ID);
        properties.put("key.serializer", "org.apache.kafka.common.serialization.IntegerSerializer");
        properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        producer = new KafkaProducer&lt;&gt;(properties);
        this.topic = topic;
        this.isAsync = isAsync;
    }

    public void run() {
        int messageNo = 1;
        while (true) {
            String messageStr = "Message_" + messageNo;
            long startTime = System.currentTimeMillis();
            if (isAsync) { // Send asynchronously
                producer.send(new ProducerRecord&lt;&gt;(topic,
                        messageNo,
                        messageStr), new DemoCallBack(startTime, messageNo, messageStr));
            } else { // Send synchronously
                try {
                    producer.send(new ProducerRecord&lt;&gt;(topic,
                            messageNo,
                            messageStr)).get();
                    System.out.println("Sent message: (" + messageNo + ", " + messageStr + ")");
                } catch (InterruptedException | ExecutionException e) {
                    e.printStackTrace();
                    // handle the exception
                }
            }
            ++messageNo;
        }
    }
}

class DemoCallBack implements Callback {

    private final long startTime;
    private final int key;
    private final String message;

    public DemoCallBack(long startTime, int key, String message) {
        this.startTime = startTime;
        this.key = key;
        this.message = message;
    }

    /**
     * onCompletion method will be called when the record sent to the Kafka Server has been acknowledged.
     *
     * @param metadata  The metadata contains the partition and offset of the record. Null if an error occurred.
     * @param exception The exception thrown during processing of this record. Null if no error occurred.
     */
    public void onCompletion(RecordMetadata metadata, Exception exception) {
        long elapsedTime = System.currentTimeMillis() - startTime;
        if (metadata != null) {
            System.out.println(
```

```
                    "message(" + key + ", " + message + ") sent to partition(" + metadata.partition() +
                    "), " +
                    "offset(" + metadata.offset() + ") in " + elapsedTime + " ms");
        } else {
            exception.printStackTrace();
        }
    }
}
```

KafkaProducerDemo.java

```
public class KafkaProducerDemo {
    public static final String TOPIC = "testTopic";

    public static void main(String[] args) {
        boolean isAsync = false;
        SampleProducer producerThread = new SampleProducer(TOPIC, isAsync);
        // start the producer
        producerThread.start();

    }
}
```

Run KafkaProducerDemo.java.

Output

```
Sent message: (1, Message_1)
Sent message: (2, Message_2)
Sent message: (3, Message_3)
Sent message: (4, Message_4)
Sent message: (5, Message_5)
Sent message: (6, Message_6)
Sent message: (7, Message_7)
Sent message: (8, Message_8)
Sent message: (9, Message_9)
Sent message: (10, Message_10)
Sent message: (11, Message_11)
Sent message: (12, Message_12)
```

Messages are sent synchronously. You may change the value of isAsync to true to send messages Asynchronously to Kafka Cluster.

Conclusion :

In this Apache Kafka Tutorial – **Kafka Producer Example**, we have learnt about Kafka Producer, and presented a step by step guide to realize a Kafka Producer 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