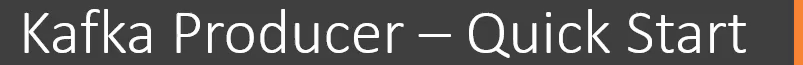
1. Text

   Description automatically generated  
   
2. **Agenda**:
   1. Mechanism behind sending events to Apache Kafka.
3. Let’s start creating some examples to understand the mechanism of sending events to Apache Kafka.
4. 
5. To help you build the concept more clearly, I want to take a problem solution approach.
6. In this approach, we will describe a problem first then see how we can build and implement a solution for the given problem.
7. **Here is the problem**.
8. A picture containing table

   Description automatically generated
9. Download the attached following two projects and import them into the IntelliJ.
   1. **There are two zip files**.
      1. **02-hello-producer-starter.zip**: Initial project.
      2. **02-hello-producer-completed.zip**: Complete project with code.
10. Everything is already configured like 🡺 pom.xml file, log4j2.xml file.
11. Text

    Description automatically generated
12. Now as project template is loaded, we’re ready to write our first producer example.
13. Let’s create a class with main().
    1. Graphical user interface

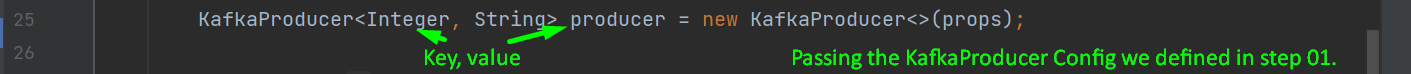
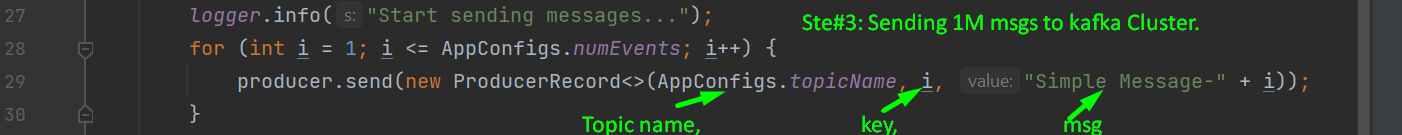
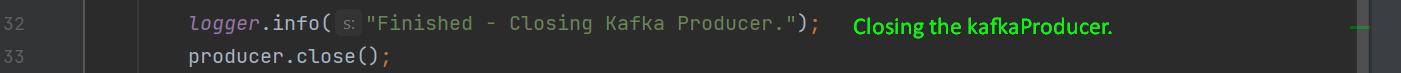
       Description automatically generated with medium confidence
14. Now, let’s come to the real stuff.   
    Sending data to Apache Kafka is multi-step process.
15. 1st step is to create **Java Properties class object** and put some **necessary kafka configurations** in it.  
    Kafka Producer API is highly configurable, and we customize the behavior by setting different **producer configurations**.  
    A screenshot of a computer

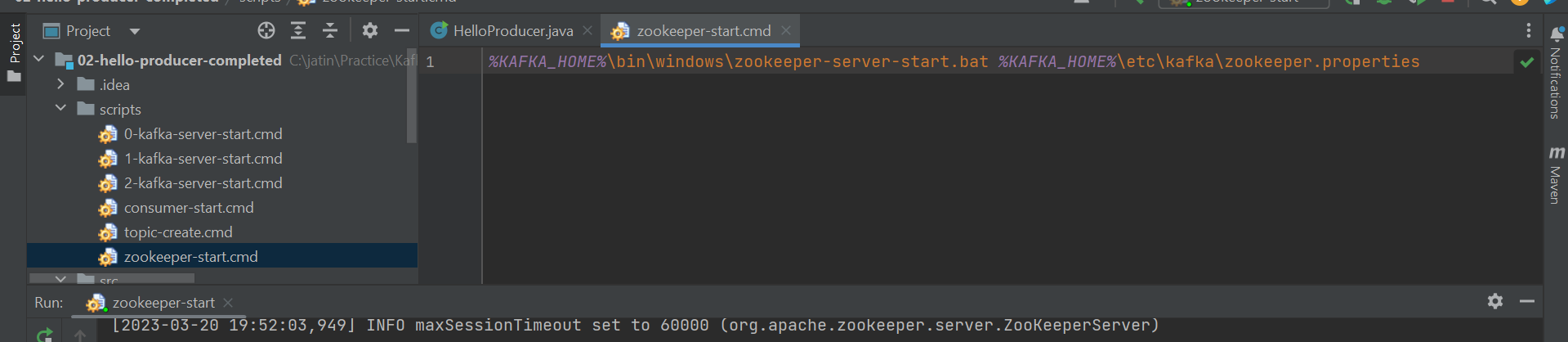
    Description automatically generated
16. In this first example, we will add 4 basic configurations. These are the bare minimum configurations for producer to work.

|  |  |  |
| --- | --- | --- |
| **Key** | **Value** | **Purpose** |
| **ProducerConfig**.**CLIENT\_ID\_CONFIG** | “HelloProducer” | 1. Simple String that is passed to Kafka Server. 2. Purpose is to track the source of the msg. |
| **ProducerConfig**.**BOOTSTRAP\_SERVERS\_CONFIG** | "localhost:9092,localhost:9093" | 1. List of host/port pairs separated by comma. 2. Producer will use this info to set up the initial connection with the Kafka Cluster. 3. The bootstrap configuration is used only for the initial connection. Once connected, the Kafka Producer will automatically query for the metadata and discover the full list of Kafka Brokers in the cluster. No matter if it is follower or leader broker IP. As after getting full list of broker Ips, producer will send msg only to leader It means you don’t need to send the complete list of Kafka Brokers as Bootstrap configuration because each broker leader or follower has complete broker list to send back to Producer. However, it is recommended to provide two to three Broker addresses of a multi-node cluster. Doing so will help the producer to check for the second or third Broker in case the first Broker in the list is down. |
| **ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG** | IntegerSerializer.class.getName() | 1. As key travels over network and so needs to be serialized. |
| **ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG** | StringSerializer.class.getName() | 1. As value travels over network and so needs to be serialized. |

Refer to the following snapshot having code for **step 01: Kafka Producer Configurations**.  
A picture containing text

Description automatically generated

1. Now here we have two topics/concepts:
   1. **The first thing is Key/Value Pair**:
      1. A Kafka msg must have a Key/Value Structure.
      2. That means each msg that we send must have key, value. (Later on, in some lectures maybe key is optional actually)
      3. We can have a null key but the msg is structured as key/value pair.
   2. **Serialization**:
      1. As Kafka Msgs are sent over network so the key/value must be serialized into bytes before they are streamed over the network.
      2. Kafka Producer API comes with a bunch of ready-to-use Serializer classes.
      3. In the above configuration, we’re setting integer serializer for key and String Serializer for msg.
2. **Step 2** is to create a **KafkaProducer.java**.
   1. Key will be integer and value will be String.  
      
3. **Step#03**: Sending 1M msgs to Kafka Cluster  
   In the below snapshot Line#29, we’re creating ProducerRecord as msg is sent by instantiating this class object and setting topic name, key, value into it.  
   
4. **Step#4**: Closing the KafkaProducer  
   
   1. The producer functionality is a little involved & it does a lot of things internally.
   2. We will cover producer internals as we progress in this course.
   3. It is essential to understand that a producer consists of some **buffer space and background I/O thread**.
   4. If you don’t close the KafkaProducer after sending all the msgs, you will leak the resources created by the KafkaProducer.
5. Complete Code we discussed so far.  
   Text

   Description automatically generated with medium confidence
6. Let’s run the project as the project as script files to run zookeeper, 3 kafka nodes, to create topic.
   1. **Run the Zookeeper:**
   2. **Run 3 nodes:**Graphical user interface, text

      Description automatically generated
   3. **Create Topic:**Graphical user interface, text

      Description automatically generated
   4. **Sending 1Million msgs in for loop Line#27:**Text

      Description automatically generated