Let us analyze a real time application to get the latest twitter feeds and its hashtags. Earlier, we have seen integration of Storm and Spark with Kafka. In both the scenarios, we created a Kafka Producer (using cli) to send message to the Kafka ecosystem. Then, the storm and spark integration reads the messages by using the Kafka consumer and injects it into storm and spark ecosystem respectively. So, practically we need to create a Kafka Producer, which should --

- Read the twitter feeds using "Twitter Streaming API",
- Process the feeds,
- Extract the HashTags and
- Send it to Kafka.

Once the "HashTags" are received by Kafka, the Storm / Spark integration receive the infor- mation and send it to Storm / Spark ecosystem.

Twitter StreamingAPI

The "Twitter Streaming API" can be accessed in any programming language. The "twitter4j" is an open source, unofficial Java library, which provides a Java based module to easily access the "Twitter Streaming API". The "twitter4j" provides a listener based framework to access the tweets. To access the "Twitter Streaming API", we need to sign in for Twitter developer account and should get the following **OAuth** authentication details.

- Customerkey
- CustomerSecret
- AccessToken
- AccessTookenSecret

Once the developer account is created, download the "twitter4j" jar files and place it in the java class path.

The Complete Twitter Kafka producer coding (KafkaTwitterProducer.java) is listed below --

```
import  java.util.Arrays;
import  java.util.Properties;
import  java.util.concurrent.LinkedBlockingQueue;

import twitter4j.*;
import  twitter4j.conf.*;

import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.KafkaProducer;

import org.apache.kafka.clients.producer.ProducerRecord;

public class KafkaTwitterProducer {

     public static void main(String\[\] args) throws Exception {

            LinkedBlockingQueue\<Status\> queue = new
```

```
LinkedBlockingQueue\<Status\>(1000);

            if(args.length \< 5)

            {

                    System.out.println(\"Usage: KafkaTwitterProducer
\<twitter-consumer-key\> \<twitter-consumer-secret\> \<twitter-access-token\>
\<twitter-access-token-secret\> \<topic-name\> \<twitter-search-keywords\>\");

                    return;

            }

            String consumerKey = args\[0\].toString();
            String consumerSecret = args\[1\].toString();
            String accessToken = args\[2\].toString();
            String accessTokenSecret = args\[3\].toString();
            String topicName = args\[4\].toString();
            String\[\] arguments = args.clone();
            String\[\] keyWords = Arrays.copyOfRange(arguments, 5,
arguments.length);

            ConfigurationBuilder cb = new ConfigurationBuilder();
            cb.setDebugEnabled(true)
                   .setOAuthConsumerKey(consumerKey)
                   .setOAuthConsumerSecret(consumerSecret)
                   .setOAuthAccessToken(accessToken)
                   .setOAuthAccessTokenSecret(accessTokenSecret);

            TwitterStream twitterStream = new
TwitterStreamFactory(cb.build()).get-Instance();

            StatusListener listener = new StatusListener() {
                   \@Override
                   public void onStatus(Status status) {
                            queue.offer(status);
                   }

                   @Override
                   public void onDeletionNotice(StatusDeletionNotice
statusDeletion-Notice) {

                        // System.out.println(\"Got a status deletion notice id:\"
+ statusDeletionNotice.getStatusId());

                  }

                  @Override
                  public void onTrackLimitationNotice(int numberOfLimitedStatuses)
{ }

                  @Override
                  public void onScrubGeo(long userId, long upToStatusId) { }

                  @Override
                  public void onStallWarning(StallWarning warning) { }
                  @Override
                  public void onException(Exception ex) {
                         ex.printStackTrace();

                  }

            };

            twitterStream.addListener(listener);

            FilterQuery query = new FilterQuery().track(keyWords);
            twitterStream.filter(query);

            Thread.sleep(5000);

            // Add Kafka producer config settings

            Properties props = new Properties();
            props.put("bootstrap.servers", "localhost:9092");
            props.put("acks", "all");
            props.put("retries", 0);
            props.put("batch.size", 16384);
            props.put("linger.ms", 1);
            props.put("buffer.memory", 33554432); props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

            props.put(\"value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

            Producer\<String, String\> producer = new KafkaProducer\<String,
String\>(props);

            int i = 0;
            int j = 0;

            while(i \< 10) {

                   Status ret = queue.poll();

                   if (ret == null) {

                          Thread.sleep(100);

                          i++;

                   } else {

                          for(HashtagEntity hashtage : ret.getHashtagEntities()) {
                                 System.out.println(\"Hashtag: \" +
hashtage.getText());

                                 producer.send(new ProducerRecord\<String,
String\>(top- icName, Integer.toString(j++), hashtage.getText()));

                          }

                   }
```

```
              }

              producer.close();

              Thread.sleep(5000);

              twitterStream.shutdown();

        }

  }
```

Compilation

Compile the application using the following command:

```
javac  -cp  "/path/to/kafka/libs/*":"/path/to/twitter4j/lib/*":.
KafkaTwitterProducer.java
```

Execution

Open two consoles. Run the above compiled application as shown below in one console.

```
java -cp "/path/to/kafka/libs/*":"/path/to/twitter4j/lib/*":. KafkaTwitterProducer
<twitter-consumer-key> <twitter-consumer-secret> <twitter-access-token>
<twitter-access-token-secret> my-first-topic food
```

Run any one of the Spark / Storm application explained in the previous chapter in another win- dow. The main point to note is that the topic used should be same in both cases. Here, we have used "my-first-topic" as the topic name.

Output

The output of this application will depend on the keywords and the current feed of the twitter. A sample output is specified below (storm integration).

```
...
food : 1
foodie  : 2
burger : 1
...
```