REPORT

ABSTRACT

The test was implemented with simple ConsumerRecord<String, String>. The consumption process was carried out against a real broker in a concurrent fashion, thus increasing performance and lowering processing time.
The messages were consumed for the specified amount of time in minutes located in "consume.time.minutes" value in the "application.properties" file.

After the consumption was done under the specified "consume.time.minutes", the consumer successfully closes and application shutdown gracefully.

Every possible error and exception that could occur were handled and unit tested.
The unit tests in the application run for 2*(consume.time.minutes) value specified in the application.properties. The runtime is shared equally among the "consume function" and "start function" test cases.

IMPLEMENTATION

Configuration (Mandatory):

The following keys should be provided their broker's value in the application.properties file in resources package to ensure successful boot up of application.
1. bootstrap.servers=XXXXXX
2. group.id=XXXXXX
3. topic= XXXXXX
4. consume.time.minutes=XXXXXX

Appropriate exceptions with corresponding intuitive messages are thrown on failure to provide any or all of the above configs.

Motivation: the use of application.properties enforces modularity, clean and solid design principle.

Service Class:

The service class is comprised of its interface class with 4 methods listed below in their flow of event and its implementation class.
The explanations for these methods can be found in the service class.
1. Start

2. Connect

3. Subscribe

4. Consume

All of the above methods except the start method takes inputs/parameters to enforce/ensure a testable application.
Details can be seen in repository.

Unit test:

Unit test of the consumer can be found in the "/src/test" package.

Test covers all service logic implementation of success and failure conditions/exceptions.

Code coverage:

The application has 93% lines covered. 7% short of 100% as a result of design decision of not exposing the KafkaConsumer object to the Main method client.

The reverse of this could give the application a 100% coverage.