Kafka Producer Consumer Library Design Document

ACI Risk Analytics

Exported on 06/05/2020

Table of Contents

Click here to expand TOC

Click here to expand Page History

Version	Date	Comment
Current Version[1] **(v. 35)**	**Apr 13, 2018 05:23**	**user-49ce5**
v. 34[2]	Apr 10, 2018 10:16	**user-49ce5**
v. 33[3]	Apr 06, 2018 11:36	**user-49ce5**
v. 32[4]	Apr 05, 2018 06:20	**user-49ce5**

1 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243776788
2 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243788468
3 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243785188
4 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243781472

v. 31[5]	Apr 05, 2018 05:30	**user-49ce5**
v. 30[6]	Apr 05, 2018 04:37	**user-49ce5**
v. 29[7]	Apr 05, 2018 03:55	**user-49ce5**
v. 28[8]	Apr 04, 2018 11:01	**user-49ce5**
v. 27[9]	Apr 04, 2018 09:25	**user-49ce5**
v. 26[10]	Apr 04, 2018 08:26	**user-49ce5**
v. 25[11]	Apr 04, 2018 04:58	**user-49ce5**
v. 24[12]	Apr 04, 2018 04:56	**user-49ce5**
v. 23[13]	Apr 04, 2018 04:55	**user-49ce5**
v. 22[14]	Apr 04, 2018 04:36	**user-49ce5**
v. 21[15]	Apr 04, 2018 03:32	**user-49ce5**
v. 20[16]	Apr 04, 2018 03:29	**user-49ce5**
v. 19[17]	Apr 03, 2018 10:42	**user-49ce5**
v. 18[18]	Apr 03, 2018 10:10	**user-49ce5**
v. 17[19]	Apr 03, 2018 10:08	**user-49ce5**
v. 16[20]	Apr 03, 2018 10:07	**user-49ce5**
v. 15[21]	Apr 03, 2018 09:58	**user-49ce5**
v. 14[22]	Apr 03, 2018 09:15	**user-49ce5**

5 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243779906
6 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243779840
7 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243779821
8 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243779781
9 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243778976
10 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243778823
11 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243778616
12 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243778206
13 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243778196
14 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243778195
15 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243778158
16 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243778065
17 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243778060
18 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243777412
19 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243777338
20 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243777325
21 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243777322
22 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243777299

v. 13[23]	Apr 03, 2018 08:58	**user-49ce5**
v. 12[24]	Apr 03, 2018 08:30	**user-49ce5**
v. 11[25]	Apr 03, 2018 08:29	**user-49ce5**
v. 10[26]	Apr 03, 2018 07:56	**user-49ce5**
v. 9[27]	Apr 03, 2018 07:47	**user-49ce5**
v. 8[28]	Apr 03, 2018 07:37	**user-49ce5**
v. 7[29]	Apr 03, 2018 07:31	**user-49ce5**
v. 6[30]	Apr 03, 2018 07:30	**user-49ce5**
v. 5[31]	Apr 03, 2018 06:15	**user-49ce5**
v. 4[32]	Apr 03, 2018 05:48	**user-49ce5**
v. 3[33]	Apr 03, 2018 05:36	**user-49ce5**
v. 2[34]	Apr 03, 2018 05:30	Diana Kayumova[35]: introduction and overview added
v. 1[36]	Apr 03, 2018 05:09	Diana Kayumova[37]

Doc	**Link**
Requirements	
JIRA Tasks	ACISAE-5800[38] - Development: Kafka Producer/Consumer library CLOSED

Date:

Author:

23 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243777197
24 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243777168
25 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243777068
26 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243777065
27 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243776999
28 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243776987
29 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243776975
30 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243776964
31 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243776957
32 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243776845
33 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243776817
34 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243776807
35 https://wiki.aciworldwide.com/display/~kayumovad
36 https://wiki.aciworldwide.com/display/RMS/viewpage.action?pageId=243776803
37 https://wiki.aciworldwide.com/display/~kayumovad
38 https://jira.aciworldwide.com/browse/ACISAE-5800?src=confmacro

Reviewer(s):

1 Introduction

This design document defines the design for the Kafka Consumer Producer library
The main goal of this library is to create client side library to simplify interaction with Kafka.

2 Overview

3 Requirements

4 Use Cases

4.1 Producer

1. Client send asynchronous message (no feedback about delivery)
2. Client send synchronous message - message delivered to the broker successfully
3. Client send synchronous message - failure to deliver message to the broker
4. Multiple clients send messages to the different topics independently(without any transactions)
5. ~~-Multiple clients send messages to the different topics within single transaction (atomically)-~~ we do not support this case
6. Application is shutting down while we still have pending messages (which have not been delivered to the broker yet)

5 Implementation

5.1 Key points

5.1.1 Producer

5.1.1.1 1. Do we use Spring Kafka Template?

At the moment - the answers is NO (more details?)

5.1.1.2 2. Number of producers used by application

From the Kafka docs[39]:

```
The producer is <i>thread safe</i> and sharing a single
producer instance across threads will generally be faster than having
multiple instances.
...
The producer consists of a pool of buffer space that holds records that haven't yet been transmitted to the
server
 as well as a background I/O thread that is responsible for turning these records into requests and
transmitting them
 to the cluster. Failure to close the producer after use will leak these resources
```

According to this information it is eligible to have multiple producers for the same topic while it will lead to performance/resource penalty.

So we can use the same (single) producer for all topics <u>until we have to use/support different properties for the different topics/producers.</u>
For instance following properties may be different for different producers (this is not a full list!!!):

1. key/value serializers
2. transaction ID (if we need to use transactions but we have 2 or more independent topics, which should not be grouped in single TX)
3. buffer settings
4. retry settings
5. etc

**If we have any possibility of different configs for different topics we should use separate producers for them.
Note: if we are OK with the <u>same settings for different topics</u> we can freely (re)use single KafkaProducer for all of them.**

Taking all things above into account we should re-use the KafkaProducer as much as it possible and provide flexible way to use different properties.
Actually Spring Kafka fully cover this part, but as far as we keeping to not use it we should implement it from the scratch.

39 https://kafka.apache.org/11/javadoc/index.html?org/apache/kafka/clients/producer/KafkaProducer.html

The main point here is to "cache" Kafka Producer on properties/config base, so all clients which decided to use the same configs (regardless of the topic) will have the same (single Kafka Consumer).

5.1.1.3 3. Type of interaction

Kafka by itself provides asynchronous API with additional means to obtains delivery/send status.

We will provide API which supports both: asynchronous ans synchronous message sending.

> Kafka producer have configurable retry functionality.
> If we have big number for retry attempts and rather big retry backoff then synchronous call may be blocked for significant amount of time in case of connectivity (or another) issue with the Kafka broker.

5.1.1.4 4. Do we need to hide Kafka details from the clients?

Ideally we should provide API which will be independent from the Kafka (so we can switch to another option if needed without huge impact in existing code)
In this case we should avoid of exposing/using any Kafka specific classes in public part of API (like `RecordMetadata and etc)`

5.1.1.5 5. When to close Kafka Producer?

From the Kafka Docs:

```
Failure to close the producer after use will leak these resources
```

But we can not close the shared Producer easily as it may be used by another clients. Current implementation of the Kafka uses daemon thread for the producer so it will not prevent application from shutting down.
On the other hand we may have some messages which were added to the buffer but have not been sent to the broker yet. I think it is mostly related to the messages sent in asynchronous mode.
Here we need to decide if we can ignore/lost them or not. If we can not tolerate such lost we should provide some means for the graceful shutdown.
Note: we still can "lost" messages in case of forcefull/abrubt application termination (JVM crash, SIGKILL, OS shutdown).

One of the possible approaches is to use "shutdown hook". But according to the javadocs there is recommendation to not perform long-running operation in hooks:

```
Shutdown hooks should also finish their work quickly. When a program invokes exit the expectation is that
the virtual machine will promptly shut down and exit. When the virtual machine is terminated due to user
logoff or system shutdown the underlying operating system may only allow a fixed amount of time in which to
shut down and exit. It is therefore inadvisable to attempt any user interaction or to perform a long-
running computation in a shutdown hook.
```

Spring also provides some application lifecycle "hooks" but we do NOT use Spring in our library so we can not use them (otherwise we will have dependency form the Spring). We can overcome this limitation by adding another one module which will add Spring ti the library (so we will have 2 output artifacts: non-Spring and Spring-aware)
Another option is just to provide/expose close/shutdown method to the ProducerFactories. So library users will be responsible to call it at the appropriate time.

5.1.1.6 6. Do we need Kafka transactions?

According to Kafka docs[40] there can be only one open transaction per producer.
This may create some issues in case of the shared KafkaProducer (when multiple client may send messages from different threads using the same producer).

1. Kafka transactions cover only Kafka and will not cover with external resources (like Hbase, Phoenix)
   In other words if we read message from the the Kafka and write it to the HBase we can NOT do this atomically. We have to use idempotent upserts to avoid duplicate records (in case of failures).
   Note: ‘consume-transform-produce’ part covered by Kafka transactions may be utilized only if both: "message source" and "target output" are Kafka topics (e.g. when we use Kafka Streams).
   Note2: some (not all) Kafka sink connectors supports exactly once semantics, but seems it again implemented with some addtional means except kafka transactions.
   For instance: HDFS connector[41] supports exactly-once semantics, while JDBC connector[42] will support it only in idempotent upsert cases.
2. Additionally at the moment we are going to have single topic, so multi-topic functionality covered by Kafka transaction is not required.
3. Transactional and idempotent producers may have un-recoverable errors and this case we have to close producer and create the new one (see Kafka docs[43])

Taking this into account I would suggest to skip TX support for now unless it is must have for us

5.1.2 Consumer

5.1.2.1 1. Message delivery guarantees?

We have 3 possible delivery guarantees:

1. at-least-one (duplicate are possible)
2. at-most-one (no duplicates, but we can lost a message)
3. exactly-once (message will be consumed and processed exactly one time)

Our current assumption: client will pass consumed data to the some external (to Kafka) component, for instance it may write data to the HBase.
In this case we can not use transaction stuff provided by Kafka (as it works only if both input and output are kafka topics).
So it is next to impossible to provide out-of-the-box support for the exactly-once semantic. At the same time we should not lost messages (transactions), so at-most-one mode is unacceptable for us.
In this case the best solution is to use "at-least-one" mode (the default mode for Kafka) and couple it with idempotent writes (when duplicate write will not affect anything) or client itself may detect and filter out duplicates.
In any of this 2 options **library user is responsible for duplicate handling.**

40 https://kafka.apache.org/11/javadoc/index.html?org/apache/kafka/clients/producer/KafkaProducer.html
41 https://docs.confluent.io/current/connect/connect-hdfs/docs/hdfs_connector.html#features
42 https://docs.confluent.io/current/connect/connect-jdbc/docs/sink_connector.html#features
43 https://kafka.apache.org/11/javadoc/org/apache/kafka/clients/producer/KafkaProducer.html#send-org.apache.kafka.clients.producer.ProducerRecord-org.apache.kafka.clients.producer.Callback-

To support "at-least-one" mode we should avoid usage of auto-commit otherwise we may have a case when client business logic failed (so message was not processed) but offset was already committed.
In other words we will use manual offset commits: offset will be committed only after ConsumerCallback returned without error.

To enforce this we need to explicitly define following properties before sending config to the Kafka's consumer:

- `enable.auto.commit=false`

5.1.2.2 2. Thread model for the consumer

The consumer will be poll the kafka topic in the infinity loop. This why we need a dedicated thread for it.
Here we have 2 options:

1. we create and handle this thread inside our library
2. client is responsible for thread creation and running consumer in this thread.

Seems the 1st one approach is more user friendly in regard to library users, so we will use it.

5.1.2.3 3. Configure maximum number of the polled records

When we poll messages from the Kafka topic we receive a set of messages. By default the maximum number of messages returned from the poll method is 500 (see https://kafka.apache.org/documentation/)
In case of the rather huge messages we may have too big memory consumption.
Additionally we need to remember that we use following model: read--process--commit. So the next poll request will be sent only after all messages from the previous poll are processed.
In this case we should be aware that poll actually used by Kafka as some kind of "health check" so in case of too big intervals between polls consumer may be considered as dead and its partitions will be reassigned (and corresponding messages will be re-processed) - that is NOT what we want.

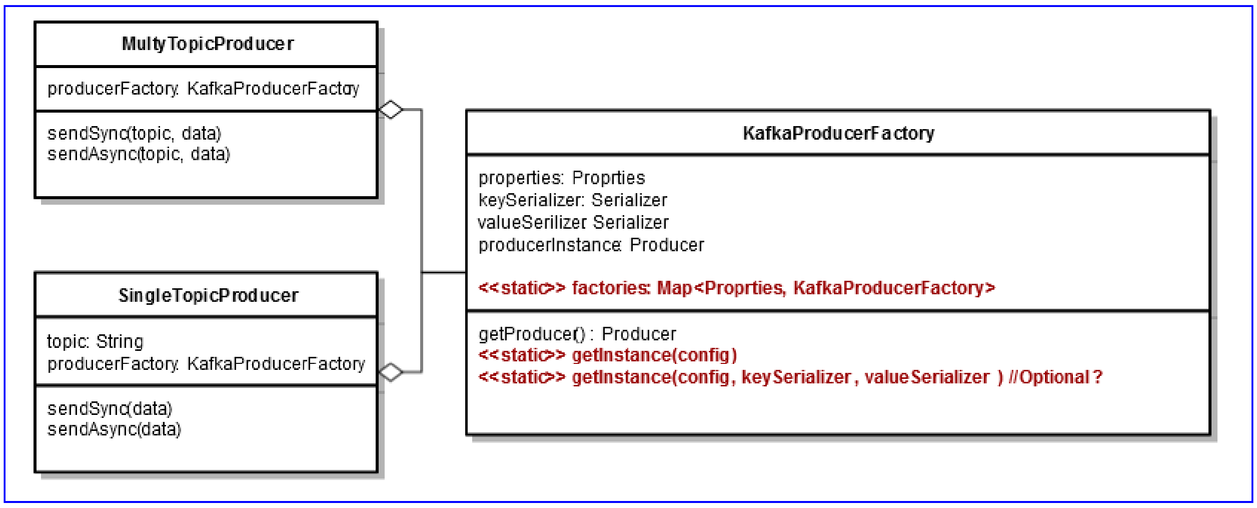
This part should be configured with following properties:

- `max.poll.interval.ms`
- `max.poll.records`

6 Class Diagram

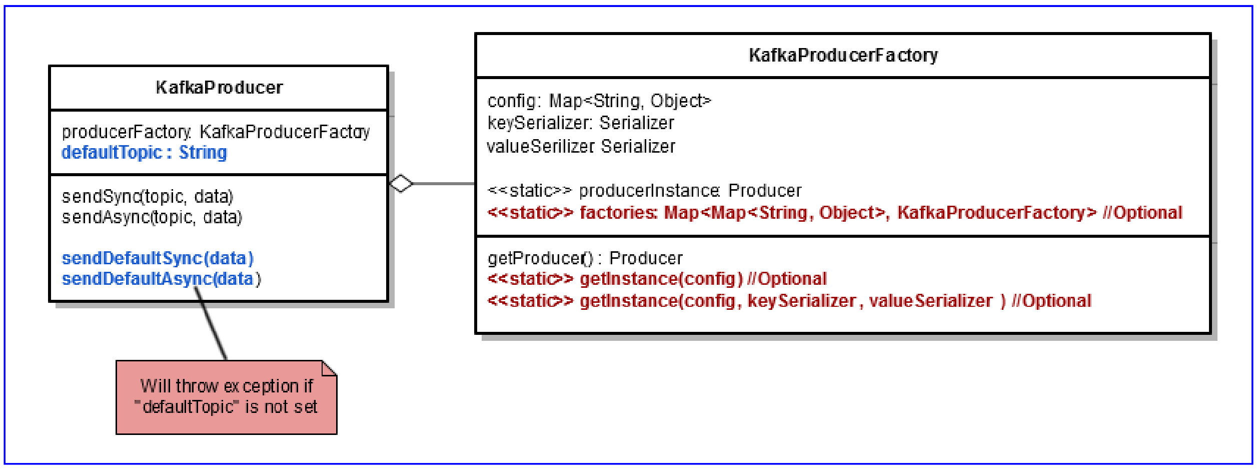
6.1 Producer part

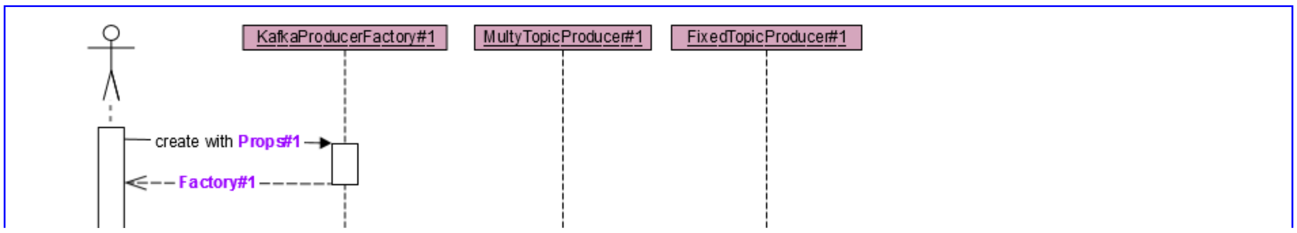
Click here to expand Approach1

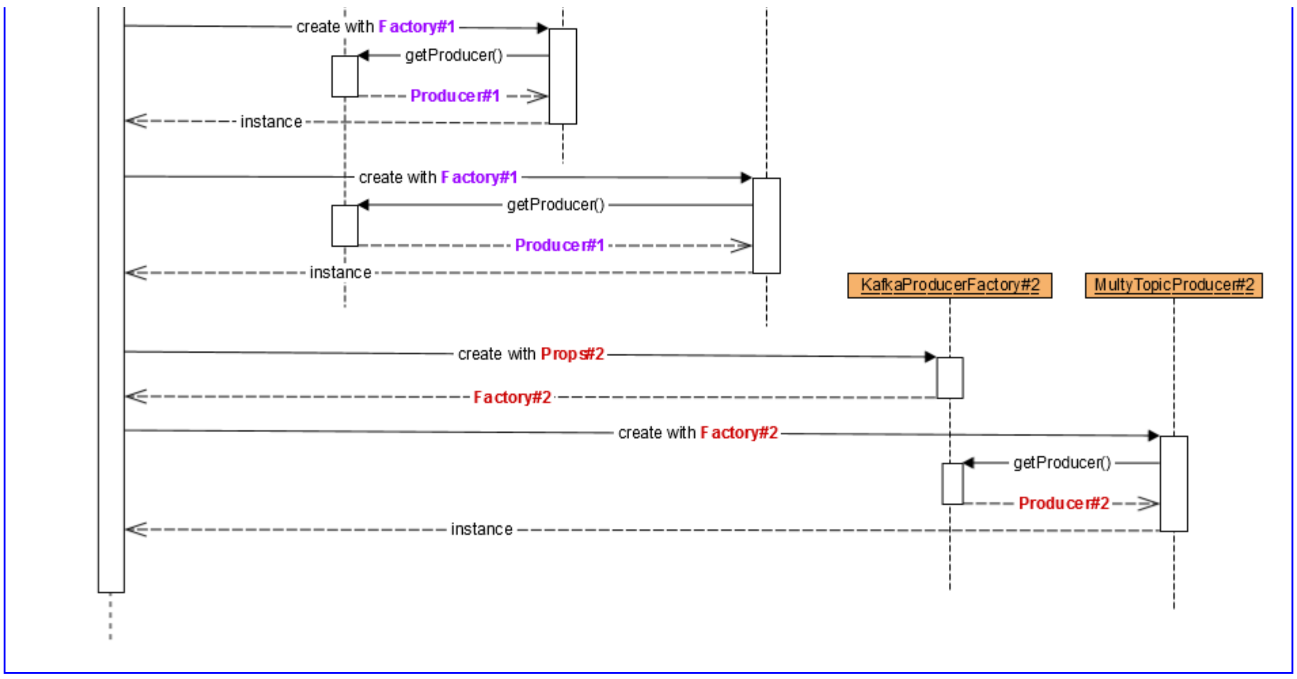

MultyTopicProducer
producerFactory: KafkaProducerFactory
sendSync(topic, data)
sendAsync(topic, data)
SingleTopicProducer
topic: String
producerFactory: KafkaProducerFactory
sendSync(data)
sendAsync(data)
KafkaProducerFactory
properties: Proprties
keySerializer: Serializer
valueSerilizer: Serializer
producerInstance: Producer
<<static>> factories: Map<Proprties, KafkaProducerFactory>
getProducer() : Producer
<<static>> getInstance(config)
<<static>> getInstance(config, keySerializer, valueSerializer) //Optional ?

Click here to expand Approach2

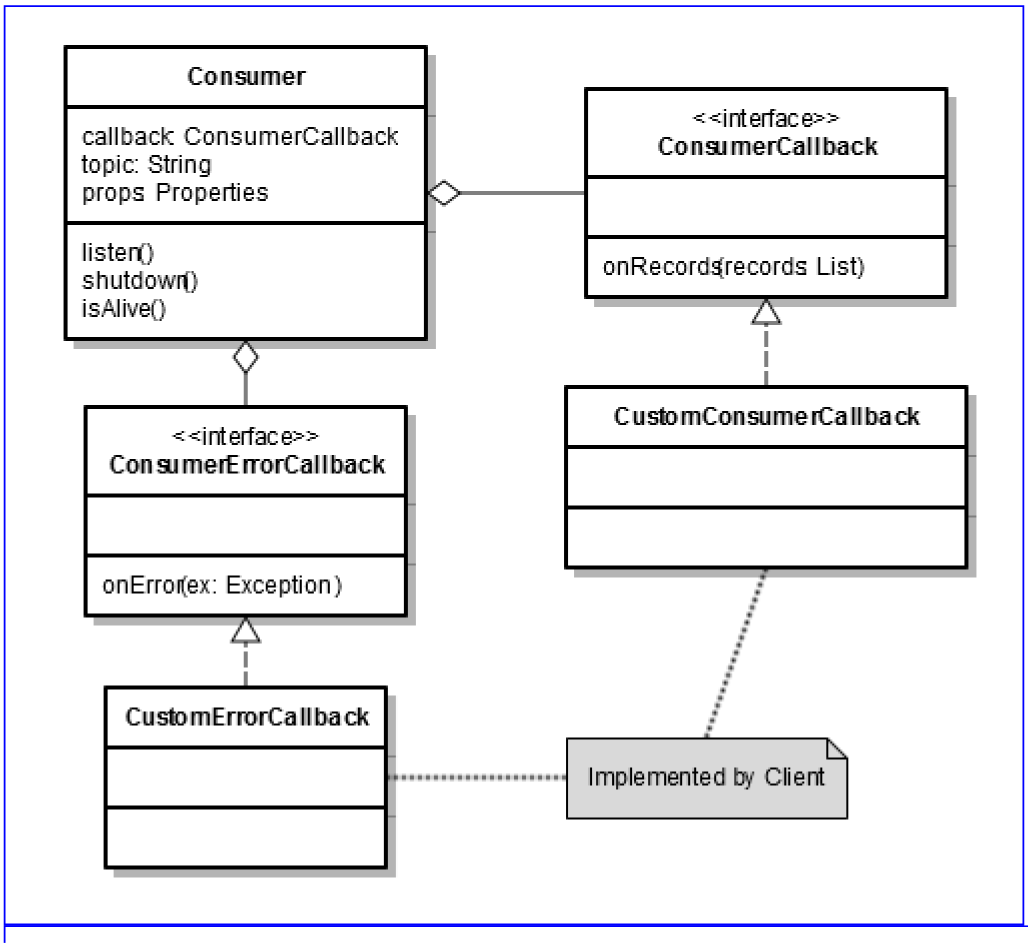
With this approach producer methods related to the "defaultTopic" will be un-usable (throw exceptions) if "defaultTopic" is not set.
So in this case client must be aware about Producer configuration which is somehow contradict with "Dependency Injection" goal.
I consider this approach as error prone and would suggest to avoid it usage and stick with the 1st one

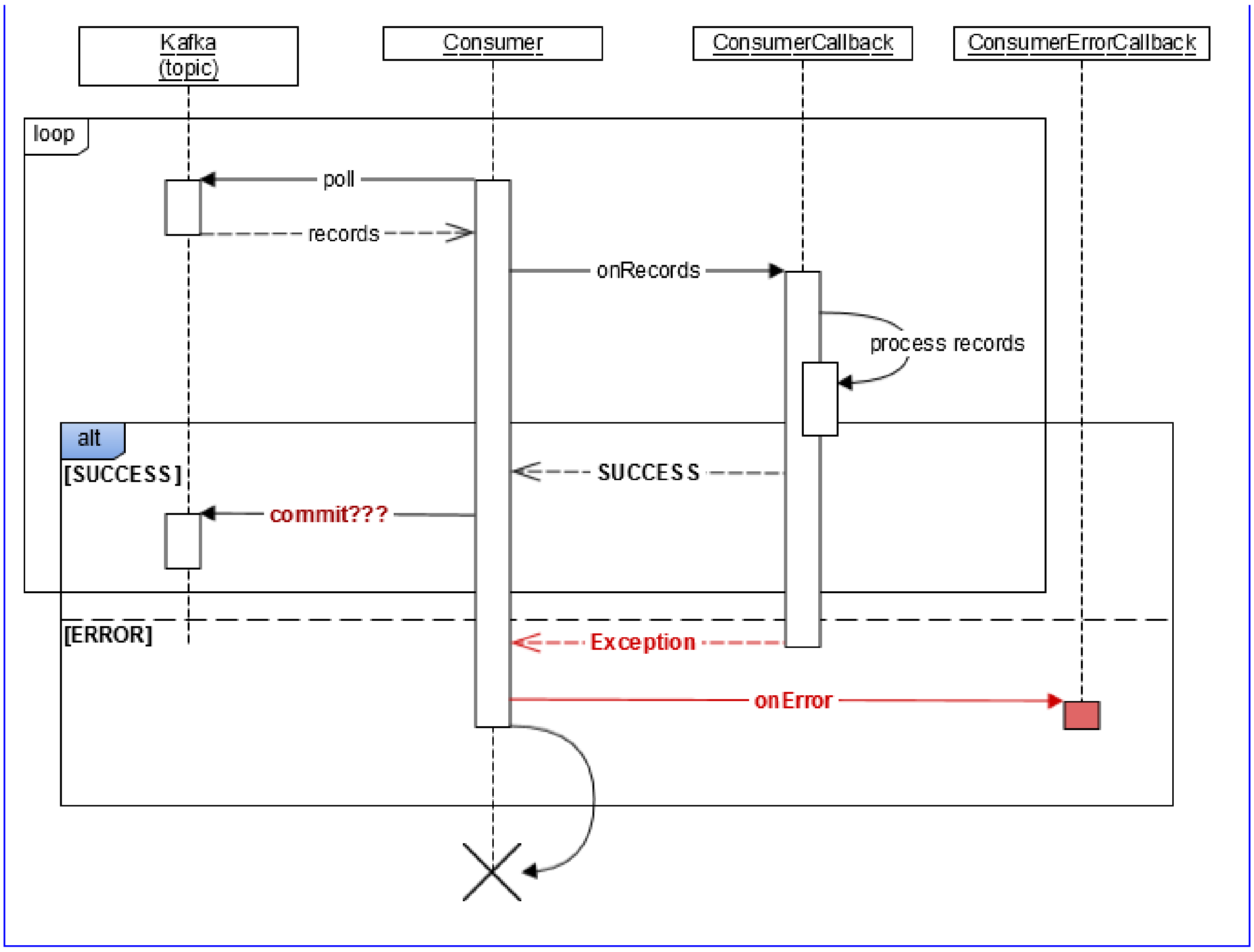

KafkaProducer
producerFactory: KafkaProducerFactory
defaultTopic : String
sendSync(topic, data)
sendAsync(topic, data)
sendDefaultSync(data)
sendDefaultAsync(data)
Will throw exception if "defaultTopic" is not set
KafkaProducerFactory
config: Map<String, Object>
keySerializer: Serializer
valueSerilizer: Serializer
<<static>> producerInstance: Producer
<<static>> factories: Map<Map<String, Object>, KafkaProducerFactory> //Optional
getProducer() : Producer
<<static>> getInstance(config) //Optional
<<static>> getInstance(config, keySerializer, valueSerializer) //Optional





6.2 Consumer part

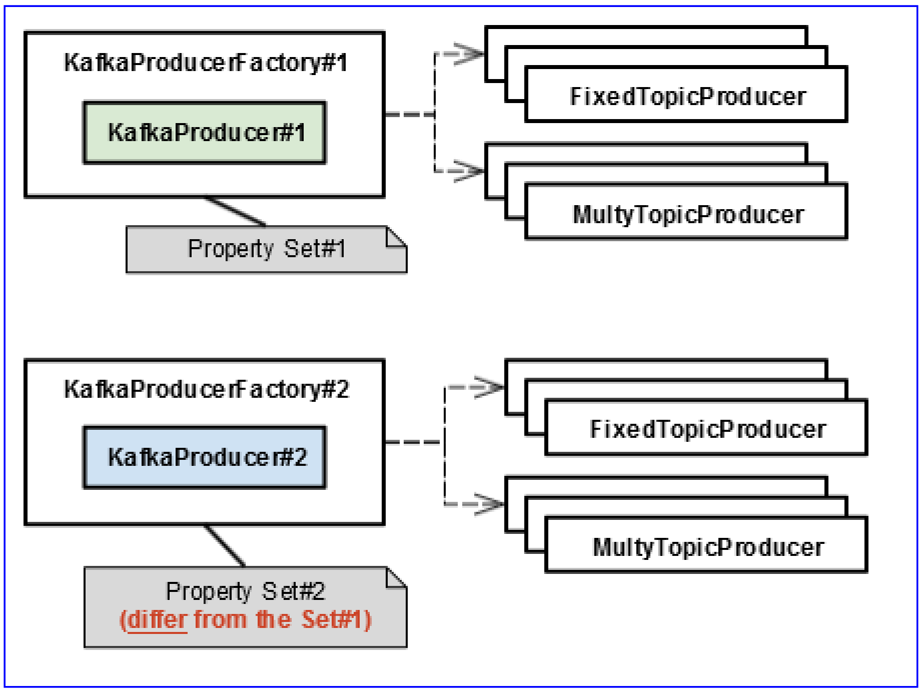



Kafka (topic)
Consumer
ConsumerCallback
ConsumerErrorCallback
loop
poll
records
onRecords
process records
alt
[SUCCESS]
SUCCESS
commit???
[ERROR]
Exception
onError

7 Class Dictionary

7.1 KafkaProducerFactory

Kafka Producer have some resources associated with them. This why creation of unnecessary producers may lead to performance penalty.
At the same time producer configuration can be set only at creation time, so if we need different settings we have to create more then 1 producers.
The KafkaProducerFactory goal is to simplify producer creation for the our library clients. It will enforce that for one set of the properties we have exactly one KafkaProducer.
The main point here is to have single KafkaProducerFactory and multiple Producers (SharedProducer or FixedTopicProducer) accosited with it



This just a matter of the taste what to use FixedTopicProducer or MultyTopicProducer
Thanks to the KafkaProducerFactory we can create multiple instances of both classes without performance penalty (if they backed by the same/single factory)

7.2 MultyTopicProducer

This kind of producer may be used to send messages to the different topics (as far as they use the same properties).
All send() methods of this producer expect topic name as argument.

7.3 FixedTopicProducer

This kind of producer is intended to send messages to the single/fixed topic. The topic name is set at creation time so client do NOT have to specify it in send() methods.

7.4 Consumer

This class is responsible for consuming/polling data from the Kafka queue.
It keeps internal thread and all actions including message polling and processing will be done in this thread.
The thread is "daemon thread" so it will not prevent JVM termination, but please be aware that it may be "killed" in the middle of the iteration.
To cover this we may add shutdownHook to perform attempt to shutdown the Consumer (and give it some time to perform cleanup).

⚠ It is possible to have cases when message processing is failed due to Kafka issues or because of malfunction of CustomerCallback (RuntimeExceptions and etc).
In this case we can not proceed with message polling as it may lead to lost/unprocessed messages (and our consumer uses at-least-once guarantee)
This why in case of error consumer will be closed (without commititng offset to the Kafka) and client should create and use new consumer (it will starts from the last successfully committed message)
To be able to track such cases client should/may implement own ConsumerErrorCallback.

7.5 ConsumerCallback

Library client should implement own callback on base of this interface. This callback will be called when we have new records/messages received from Kafka

7.6 ConsumerErrorCallback

Library client can implement this callback to be able to intercept errors occurred during message processing. This callback should be optional

8 Implementation Details

As we are working with sensitive information we should protect it:

1. In transit (both client and broker configuration)
2. At rest (broker configuration, **this part is outside of the scope for this design**)

We can configure both authentication and connection protection using (only) Kafka properties, see https://kafka.apache.org/documentation/#security_overview for more details.

9 Testing

10 Risks