Kafka Transport For WSO2 ESB

Table of Contents

CHAPTER 1
Introduction

CHAPTER 2
KAFKA Inbound Endpoint

CHAPTER 3
KAFKA Producer Connector

CHAPTER 4
Conclusion

CHAPTER 1

Introduction

Introduction to Apache Kafka

Kafka is a distributed, partitioned, replicated commit log service. It provides the functionality of a messaging system, but with a unique design.

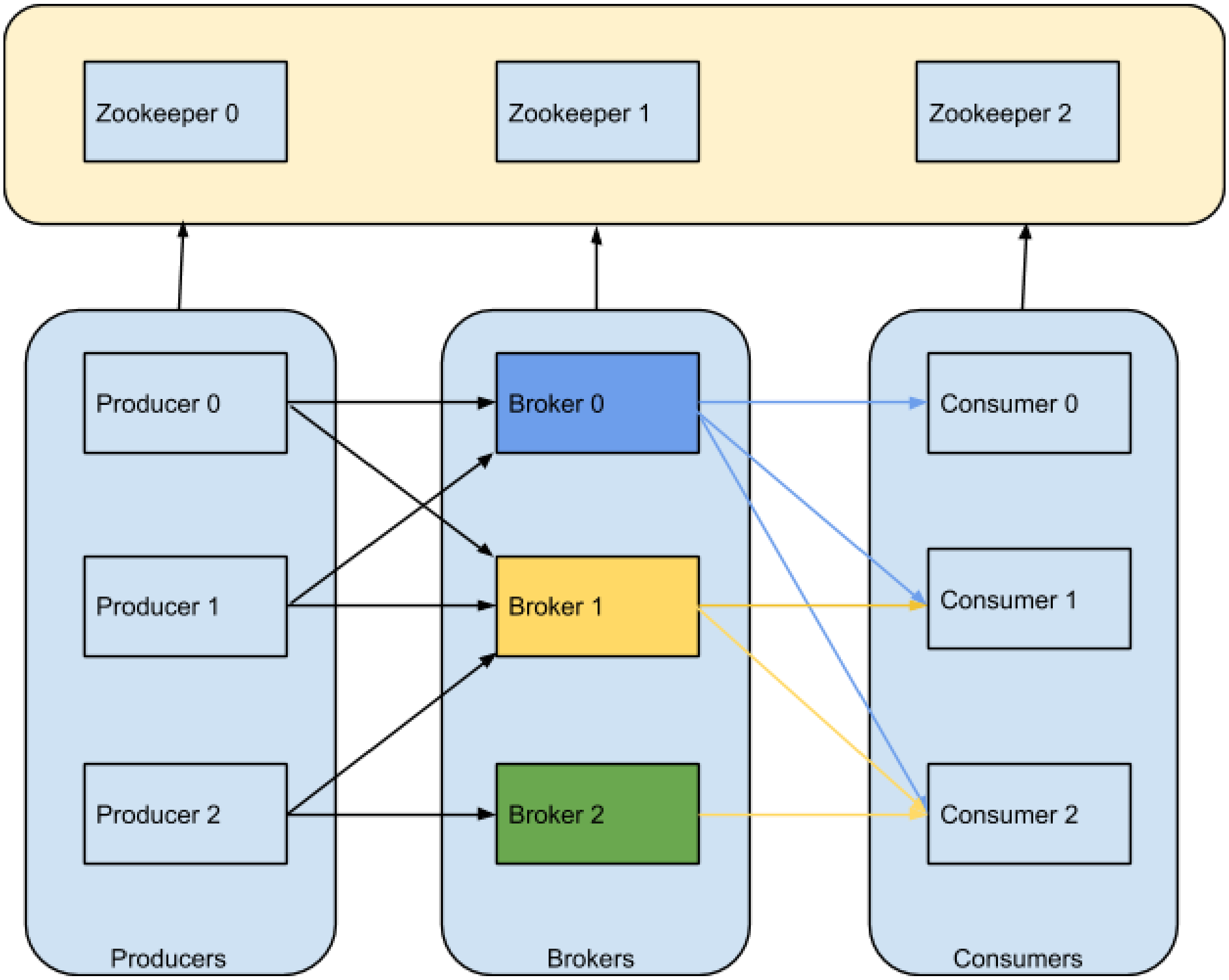


Fig 1: Kafka Messaging System

Kafka Cluster comprises of one or more servers each of which is called a **broker**.It maintains message feeds by categories which are called **topics**.Processes that publish messages to a topic are called **producers.**Processes that consume messages over topic/s are called **consumers.**

For each topic, the Kafka cluster maintains a partitioned log that is illustrated in below figure.

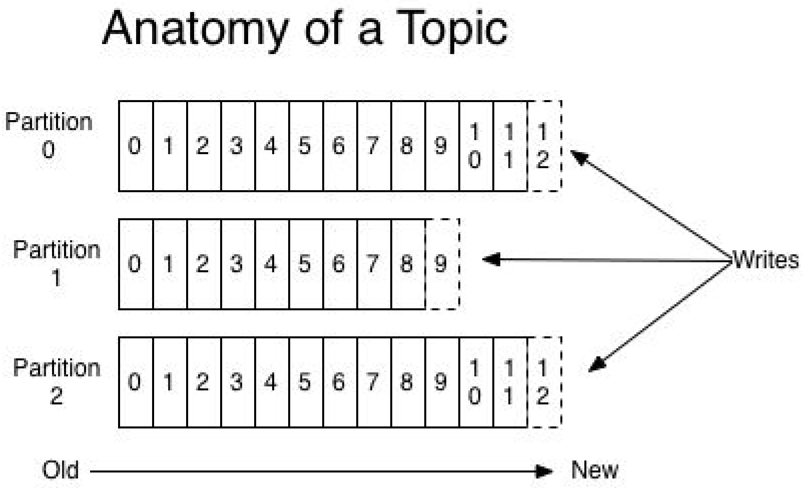
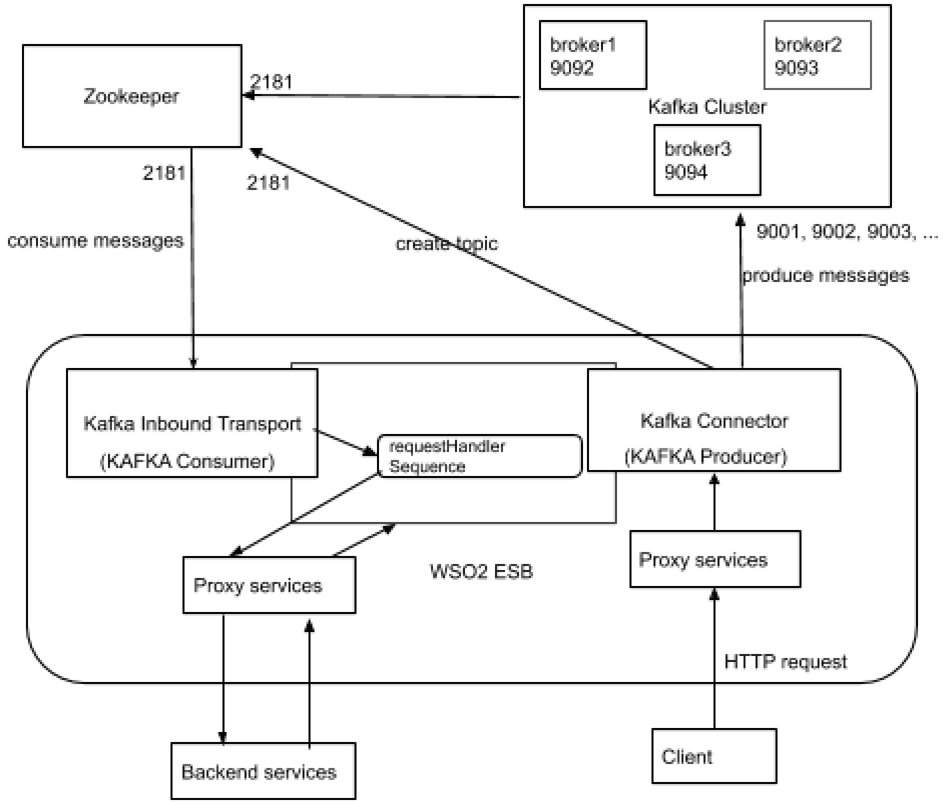


Fig 2: Anatomy of a topic

Detailed information could be found at https://kafka.apache.org



Introduction to Kafka Transport in WSO2 ESB

Fig 03: Architectute - Kafka Transport for WSO2 ESB

WSO2 ESB KAFKA transport comprises of two major components.

1. Kafka Inbound Endpoint
2. Kafka Connector

Kafka Inbound Endpoint

kafka Inbound Endpoint acts as a message consumer. It creates a connection to zookeeper and request messages for a topic/s or topic filters.

Kafka Connector

Kafka Connector acts as a message producer for the kafka cluster. It creates connection to zookeeper to create topics and connects to each broker to produce messages to a topic.

CHAPTER 2

KAFKA Inbound Endpoint

Key functionality of the inbound endpoint is to consume messages from kafka cluster and inject them to ESB.

Use case:

1. establish a connection between zookeeper by creating a consumer connector
   1.1 if consumer connector is not created goto 1
2. create message stream
3. find topic
4. request kafka streams for a topic
   4.1 request kafka streams for topic filter
5. read message from stream
6. Inject message to esb

Message Flow

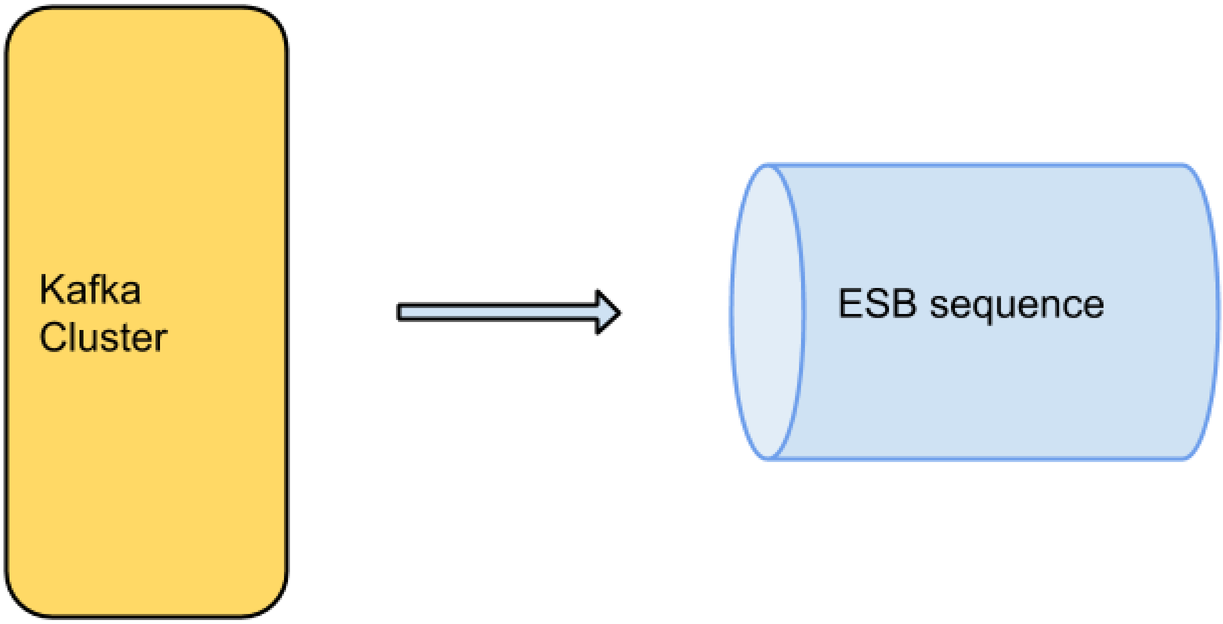


Fig 04 : Inbound Endpoint Message Flow

When a connection to the kafka service is created, it can be created a list of streams for a topic to consume. In this case it is considered only one stream. Then, messages are periodically read from the stream a]\]]]]]nd inject them to a esb sequence.

Automatic Reconnection

At the time where ESB starts, if the kafka service is not available, it will periodically retry for a connection.

Inbound Endpoint supports two kafka APIs:

1. High level Consumer API
2. Simple Consumer API

High level Consumer API

To enable High level Consumer API,
<parameter name="consumer.type">**highlevel**</parameter> should be enabled.

- If it needs to be consumed by topic, <parameter name="topics">test2</parameter> should be enabled.

- If it needs to be consumed by topic filters, <parameter name="topics">test2</parameter> should be removed and two properties <parameter name="topic.filter">test</parameter> <parameter name="filter.from.whitelist">false</parameter> should have been specified.

If filter.from.whitelist = true, messages will be consumed from whitelist(include) where as if it is false then will be consumed from blacklist(exclude).

Consumer configurations specifies in https://kafka.apache.org/documentation.html#consumerconfigs can be added to parameters section as a new parameter if required.

For most of the applications high level Consumer API is good enough.This can be used to request messages using topic/s or topic filters.

Messages can be filtered either from whitelist or blacklist.It can be configured as
<parameter name="filter.from.whitelist">**true**</parameter>

or

<parameter name="filter.from.whitelist">**false**</parameter> respectively.

Simple Consumer API

To enable Simple Consumer API,
<parameter name="consumer.type">**simple**</parameter> should be enabled.

Using simple consumer API, it is possible to request messages

- from a specific broker and from a specific partition.
- can be configured to receive required no of messages by setting
  <parameter name="simple.max.messages.to.read">5</parameter>

Sample configuration:

broker host ip and port
*<parameter name="simple.brokers">**localhost**</parameter>*
*<parameter name="simple.port">**9092**</parameter>*

partition
*<parameter name="simple.partition">**0**</parameter>*

no. of messages to read
*<parameter name="simple.max.messages.to.read">**5**</parameter>*

topic
*<parameter name="simple.topic">**test**</parameter>*

Class Diagram

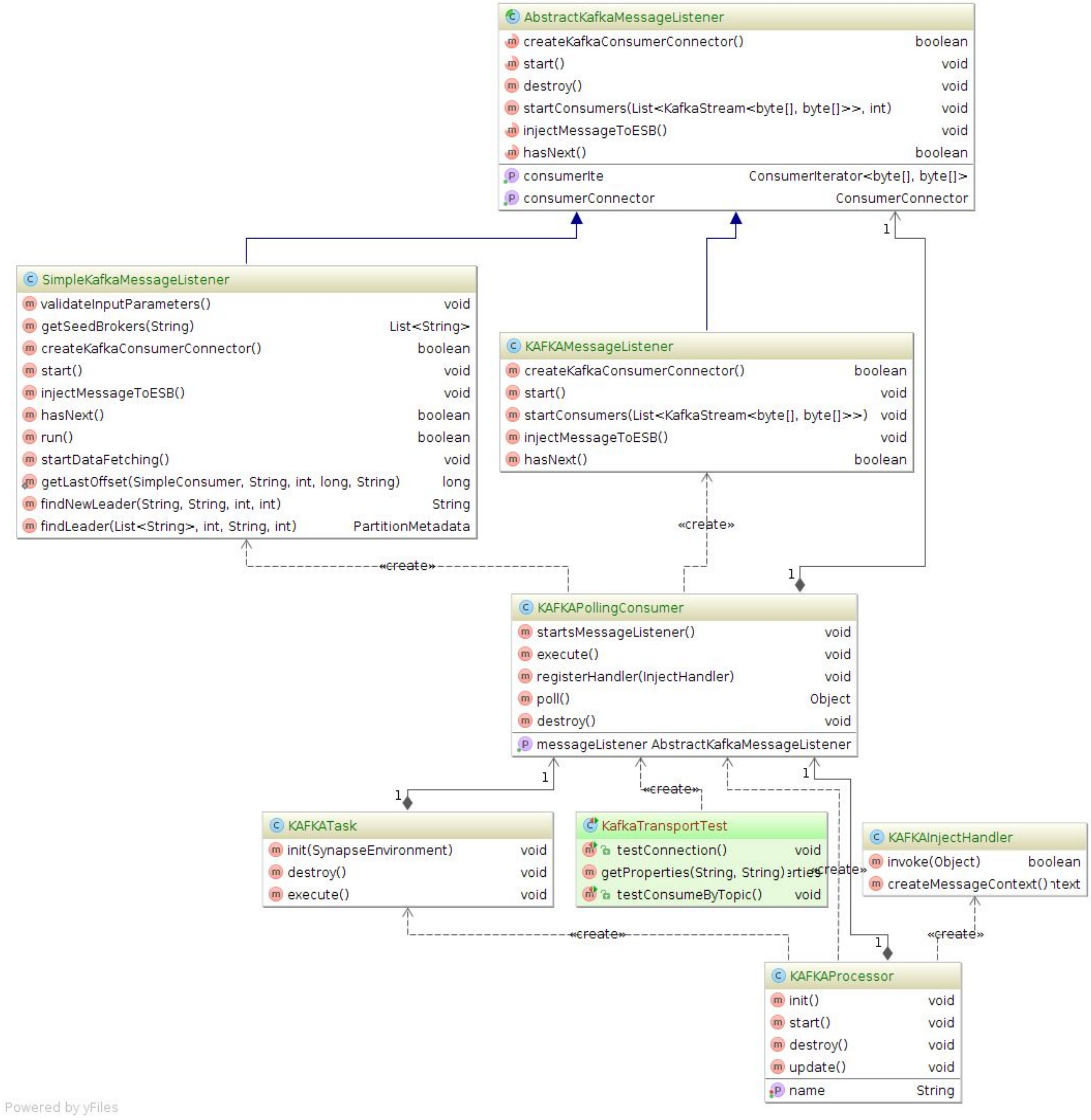

AbstractKafkaMessageListener
createKafkaConsumerConnector() boolean
start() void
destroy() void
startConsumers(List<KafkaStream<byte[], byte[]>>, int) void
injectMessageToESB() void
hasNext() boolean
consumerIte ConsumerIterator<byte[], byte[]>
consumerConnector ConsumerConnector
1
SimpleKafkaMessageListener
validateInputParameters() void
getSeedBrokers(String) List<String>
createKafkaConsumerConnector() boolean
start() void
injectMessageToESB() void
hasNext() boolean
run() boolean
startDataFetching() void
getLastOffset(SimpleConsumer, String, int, long, String) long
findNewLeader(String, String, int, int) String
findLeader(List<String>, int, String, int) PartitionMetadata
KAFKAMessageListener
createKafkaConsumerConnector() boolean
start() void
startConsumers(List<KafkaStream<byte[], byte[]>>) void
injectMessageToESB() void
hasNext() boolean
«create»
«create»
1
KAFKAPollingConsumer
startsMessageListener() void
execute() void
registerHandler(InjectHandler) void
poll() Object
destroy() void
messageListener AbstractKafkaMessageListener
1
1
«create»
1
KAFKATask
init(SynapseEnvironment) void
destroy() void
execute() void
KafkaTransportTest
testConnection() void
getProperties(String, String) erties
testConsumeByTopic() void
«create»
KAFKAInjectHandler
invoke(Object) boolean
createMessageContext() text
«create»
«create»
1
KAFKAProcessor
init() void
start() void
destroy() void
update() void
name String
Powered by yFiles

Fig 05 : Kafka Inbound EP - class diagram

Souce code

1. Create consumer connector

```
        consumerConnector = Consumer.createJavaConsumerConnector(new
        ConsumerConfig(kafkaProperties));
```

2. Create list of streams
   - By topic (message streams can be obtained for a specific topic)

```
        Map<String, Integer> topicCount = new HashMap<String, Integer>();
        topicCount.put(topic, threadCount);
        Map<String, List < KafkaStream<byte[],
    byte[]>>>consumerStreams=consumerConnector.createMessageStreams(topicCount);
        List<KafkaStream<byte[], byte[]>> streams = consumerStreams.get(topic);
        startConsumers(streams);
```

   - By topic filters(message stream can be obtained for a topic given as a regular expression)

```
         // Define #threadCount thread/s for topic filter
    List<KafkaStream<byte[], byte[]>> consumerStreams ;
            boolean isFromWhitelist =
    (kafkaProperties.getProperty(KAFKAConstants.FILTER_FROM_WHITELIST) ==null||
    kafkaProperties.getProperty(KAFKAConstants.FILTER_FROM_WHITELIST) .isEmpty())
        ? Boolean.TRUE :
Boolean.parseBoolean(kafkaProperties.getProperty(KAFKAConstants.FILTER_FROM_WHITEL
IST));
        if(isFromWhitelist) {
        consumerStreams =  consumerConnector
            .createMessageStreamsByFilter(new
Whitelist(kafkaProperties.getProperty(KAFKAConstants.TOPIC_FILTER)), threadCount);
        }else{
        consumerStreams =  consumerConnector
            .createMessageStreamsByFilter(new
Blacklist(kafkaProperties.getProperty(KAFKAConstants.TOPIC_FILTER)), threadCount);
        }

        startConsumers(consumerStreams,threadNo);
```

Sorce code for the inbound EP can be found at https://github.com/isharac/KAFKAInboud

CHAPTER 3

KAFKA Producer Connector In WO2 ESB

Kafka Producer connector allows you to send messages to broker list. The connector uses to configure the producer and send the message with topic name and key.producers are applications that create messages and publish them to the Kafka broker for further consumption

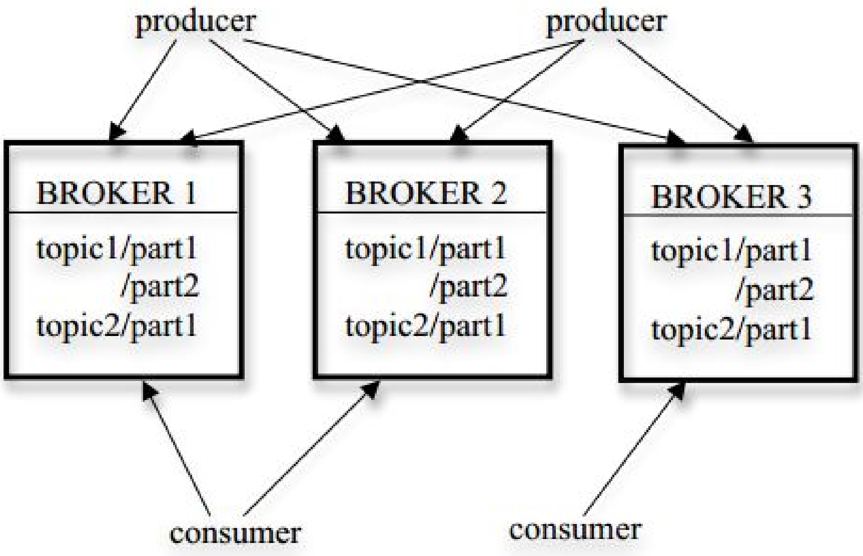


Fig 06 : Kafka Producer

Connecting to kafka brokers

To use kafka producer, create a proxy service and add the <kafkaTransport.init> element in your configuration before any other kafka produce operation. This configuration is used for setting the broker list,serialization class,request required acks,producer type,etc.

To send message to single broker,add the following configuration element in a proxy service as parameters

```
<kafkaTransport.init>
        <brokerlist>localhost:9092</brokerlist>
        <serializationclass>kafka.serializer.StringEncoder</serializationclass>
        <requiredacks>1</requiredacks>
        <producertype>sync</producertype>
</kafkaTransport.init>
```

brokerlist : the broker name and port (hostname:port)

serializationclass : serializer class for messages.The default encoder takes a byte array and returns the same byte array
default value is *kafka.serializer.StringEncoder*

requiredAcks : *add number such as -1,0 or 1*

producerType : sync or async

*Producer configurations specifies in https://kafka.apache.org/documentation.html#producerconfigs can be added to parameters section as a new parameter if required.

To send message to multi broker list ,setup the brokers

Setting up a multi-broker cluster
Broker List can be more than one which are separated by commas

```
<brokerlist>host1:port1 , host2:port2, …..</brokerlist>
```

First we make a config file for each of the brokers:

```
> cp config/server.properties config/server-1.properties
> cp config/server.properties config/server-2.properties
```

Now edit these new files and set the following properties:

```
config/server-1.properties:
    broker.id=1
    port=9093
    log.dir=/tmp/kafka-logs-1

config/server-2.properties:
    broker.id=2
    port=9094
    log.dir=/tmp/kafka-logs-2
```

The broker.id property is the unique and permanent name of each node in the cluster. We have to override the port and log directory only because we are running these all on the same machine and we want to keep the brokers from all trying to register on the same port or overwrite each others data.

Kafka transport connector supports to optionally configure using following properties

```
<kafkaTransport.init>
        <brokerlist>localhost:9092,localhost:9093</brokerlist>
        <serializationclass>kafka.serializer.StringEncoder</serializationclass>
        <requiredacks>1</requiredacks>
        <producertype>sync</producertype>
        <keyserializerclass>kafka.serializer.DefaultEncoder</keyserializerclass>
        <partitionerclass>kafka.producer.DefaultPartitioner</partitionerclass>
        <compressioncodec>none</compressioncodec>
        <compressedtopics>null</compressedtopics>
        <messagesendmaxretries>3</messagesendmaxretries>
        <retrybackoff>100</retrybackoff>
        <refreshinterval>60000</refreshinterval>
        <bufferingmaxtime>5000</bufferingmaxtime>
        <bufferingmaxmessages>10000</bufferingmaxmessages>
        <timeoutevent>-1</timeoutevent>
        <batchnomessages>200</batchnomessages>
        <sendbuffersize>102400</sendbuffersize>
        <clientid></clientid>
</kafkaTransport.init>
```

The above values are default of of each configuration element.The user can change the configuration based on the messaged size,buffering max time,etc.

create topic

```
        <kafkaTransport.kafkaproduce-operation>
                <topic>test2</topic>
         </kafkaTransport.kafkaproduce-operation>
```

create topic with key

```
        <kafkaTransport.kafkaproduce-operation>
                <topic>test2</topic>
                <key>key1</key>
           </kafkaTransport.kafkaproduce-operation>
```

class diagram

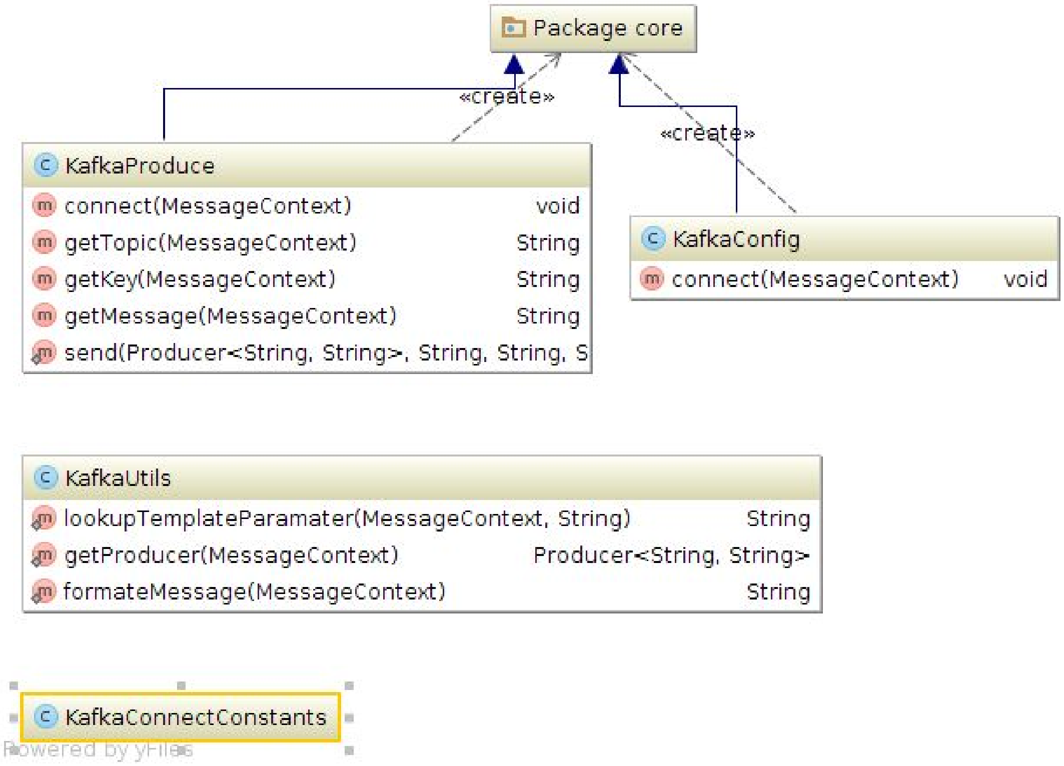

Package core
«create»
«create»
KafkaProduce
connect(MessageContext) void
getTopic(MessageContext) String
getKey(MessageContext) String
getMessage(MessageContext) String
send(Producer<String, String>, String, String, S
KafkaConfig
connect(MessageContext) void
KafkaUtils
lookupTemplateParamater(MessageContext, String) String
getProducer(MessageContext) Producer<String, String>
formateMessage(MessageContext) String
KafkaConnectConstants
Powered by yFiles

Fig 07 : Kafka Producer Connector - class diagram

source code

1) Create producer configuration

```
        new Producer<String, String>(new ProducerConfig(props));
```

2) formate message

```
public static String formateMessage(
                org.apache.axis2.context.MessageContext ctxt) throws AxisFault {
        OMOutputFormat format = BaseUtils.getOMOutputFormat(ctxt);
        MessageFormatter messageFormatter = null;

                messageFormatter = MessageProcessorSelector
```

```
                    .getMessageFormatter(ctxt);
            OutputStream out = null;
        StringWriter sw = null;
        sw = new StringWriter();
                out = new WriterOutputStream(sw, format.getCharSetEncoding());
            try {
            if (out != null) {
                messageFormatter.writeTo(ctxt, format, out, true);
                out.close();
            }
        } catch (IOException e) {
        //todo handle
        }
        return sw.toString();
    }
```

3) send messages to brokers

```
public static void send(Producer<String, String> producer,String topic,String key,String message)
    {
        if (key == null) {
            producer.send(new KeyedMessage<String, String>(topic, message));
        } else
        {
            producer.send(new KeyedMessage<String, String>(topic, key,
                    message));
        }
      }
```

Source code for the kafka producer connector can be found at
https://github.com/isharac/KAFKAInboud/connector

CHAPTER 4

Conclusion

Tests Carried out

Test No	Test	Configuration	expected results	observed results	Status
Functional Testing					
1	Produce 1000 messages to a single topic with Single broker / single producer and consume by the inbound EP		receive all	received all	Pass
2	Produce 1000 messages to a single topic with 2 brokers / single producer and consume by the inbound EP		receive all	received all	Pass
3	Produce 1000 messages to a single topic with Single broker / 2 producers and consume by the inbound EP		receive all	received all	Pass
4	create topics test, test2, testing, aa consume from.whitelist=true and topic.filter=test	<parameter name="topic.filter">test</parameter> <parameter name="filter.from.whitelist">true</parameter>	receive messages from topics test, test2, testing	received test, test2, testing	Pass
5	create topic test, test2, testing, aa consume from.whitelist=false ang topic.filter=test	<parameter name="topic.filter">test</parameter> <parameter name="filter.from.whitelist">false</parameter>	receive messages from aa	received aa	Pass
Load Testing					
1	produce 5000 messages and consume		receive all	received all	Pass

2	produce 10000 messages and consume		receive all	received all	Pass
3	produce 15000 messages and consume		receive all	received all	Pass

Load Testing

$$\text{Throughput} = \frac{\#\text{ messages}}{\text{time to consume}}$$

$$= \frac{15\ 000}{15\ 000\text{ s}}$$

$$=1/\text{s}$$

where polling interval is 1000mS for the inbound.

Future Work:

- Externalize carbon component
- Verify the functionality in MT mode
- Enable multiple streams support
- Extend to support Kafka Hadoop Consumer API