Package ‘rkafka’

June 29, 2017

Type Package
Title Using Apache 'Kafka' Messaging Queue Through 'R'
Version 1.1
Date 2017-06-28
Author Shruti Gupta[aut,cre]
Maintainer Shruti Gupta <shrutigupta34@gmail.com>
Description Apache 'Kafka' is an open-source message broker project developed by the Apache Software Foundation which can be thought of as a distributed, partitioned, replicated commit log service. At a high level, producers send messages over the network to the 'Kafka' cluster which in turn serves them up to consumers. See <http://kafka.apache.org/> for more information. Functions included in this package enable: 1. Creating 'Kafka' producer 2. Writing messages to a topic 3. Closing 'Kafka' producer 4. Creating 'Kafka' consumer 5. Reading messages from a topic 6. Closing 'Kafka' consumer. The jars required for this package are included in a separate package 'rkafkajars'.
Depends rJava, RUnit, rkafkajars
SystemRequirements Oracle Java 7, Apache Kafka 2.8.0-0.8.1.1
License Apache License 2.0 | file LICENSE
NeedsCompilation no
Repository CRAN
Date/Publication 2017-06-28 22:01:11 UTC

R topics documented:

rkafka ......................................................... 2
rkafka.closeConsumer .................................. 3
rkafka.closeProducer .................................. 4
rkafka.closeSimpleConsumer ...................... 4
rkafka.createConsumer .............................. 5
rkafka.createProducer .............................. 7
rkafka.createSimpleConsumer .................. 8
rkafka.read ............................................. 9
rkafka.readFromSimpleConsumer .............. 10
rkafka

**Description**

It provides functionalities of creating a 'Kafka' producer, simple consumer, high level consumer and sending and receiving messages.

**Details**

- **Package:** rkafka
- **Type:** Package
- **Version:** 1.1
- **Date:** 2017-06-28
- **License:** Apache License 2.0

1) Start 'Zookeeper' server. 2) Start 'Kafka' server. 3) Start producer using 'rkafka.createProducer' function. 4) Send messages using 'rkafka.send' function. 5) Close producer using 'rkafka.closeProducer' function. 6) Start consumer using 'rkafka.createConsumer' function. 7) Read messages using 'rkafka.read' function. 8) Close consumer using 'rkafka.closeConsumer' function.

**Author(s)**

Shruti Gupta

Maintainer: Who to complain to shrutigupta34@gmail.com

**References**

To understand 'Kafka' [kafka.apache.org/documentation.html](https://kafka.apache.org/documentation.html)

**Examples**

```r
## Not run:
prod1=rkafka.createProducer("127.0.0.1:9092")
rkafka.send(prod1,"test","127.0.0.1:9092","Testing once")
rkafka.send(prod1,"test","127.0.0.1:9092","Testing twice")
rkafka.send(prod1,"test","127.0.0.1:9092","Testing thrice")
rkafka.closeProducer(prod1)
consumer1=rkafka.createConsumer("127.0.0.1:2181","test")
print(rkafka.read(consumer1))
print(rkafka.read(consumer1))
```
rkafka.closeConsumer

print(rkafka.read(consumer1))

## End(Not run)

---

Description

This function shuts down the KAFKA consumer.

Usage

rkafka.closeConsumer(ConsumerObj)

Arguments

ConsumerObj

- ConsumerObj: Consumer through which messages are to be read (Java Object)
- Required: Mandatory
- Type: Consumer

Value

Function doesn’t return anything

Author(s)

Shruti Gupta

Examples

## Not run:
consumer1=rkafka.createHighConsumer("127.0.0.1:2181")
rkafka.closeHighConsumer(consumer1)

## End(Not run)
**rkafka.closeProducer**  
*KAFKA producer shutdown*

**Description**  
This function closes the KAFKA producer

**Usage**  
rkafka.closeProducer(producer)

**Arguments**

- **producer**: Producer which is to be terminated  
  Required: Mandatory  
  Type: Producer

**Value**  
Doesn’t return anything

**Author(s)**

Shruti Gupta

**Examples**

```r
# Not run:
producer1=rkafka.createProducer("127.0.0.1:9092")
rkafka.closeProducer(producer1)

# End(Not run)
```

---

**rkafka.closeSimpleConsumer**  
*Closing KAFKA Simple consumer*

**Description**  
This function shuts down the KAFKA Simple consumer

**Usage**  
rkafka.closeSimpleConsumer(SimpleConsumer)

**Arguments**

- **SimpleConsumer**: SimpleConsumer: SimpleConsumer that has to be shut down  
  Required: Mandatory  
  Type: SimpleConsumer
Details

There are two types of KAFKA consumers: High-Level and Simple. This function shuts down the KAFKA Simple Consumer.

Value

Function doesn’t return anything.

Author(s)

Shruti Gupta

References

To know when to use simple consumer and when to use High-level Consumer, refer the url below: https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example

Examples

```r
## Not run:
consumer1=rkafka.createSimpleConsumer("172.25.1.78","9092","10000","100000","test")
rkafka.receiveFromSimpleConsumer(consumer1,"test","0","0","test-group")
print(rkafka.readFromSimpleConsumer(consumer1))
rkafka.closeSimpleConsumer(consumer1)
## End(Not run)
```

---

rkafka.createConsumer  Creating KAFKA consumer

Description

This function creates a KAFKA consumer.

Usage

```r
rkafka.createConsumer(zookeeperConnect, topicName, 
　　groupId="test-consumer-group", zookeeperConnectionTimeoutMs="100000", 
　　consumerTimeoutMs="10000", autoCommitEnable="NULL", 
　　autoCommitInterval="NULL", autoOffsetReset="NULL")
```

Arguments

- **zookeeperConnect**
  - Zookeeper connection string comma separated host:port pairs, each corresponding to a zk server, e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002" Required:Mandatory
  - Type: String default: NONE

- **topicName**
  - Name of the topic from which to read messages Required:Mandatory Type: String

---
groupId - consumer group id Required: Mandatory Type: String default: test-consumer-group

zookeeperConnectionTimeoutMs
timeout in ms for connecting to zookeeper Required: Mandatory Type: String default: 100000

consumerTimeoutMs
Throw a timeout exception to the consumer if no message is available for consumption after the specified interval Required: Mandatory Type: String default: 10000

autoCommitEnable
If true, periodically commit to ZooKeeper the offset of messages already fetched by the consumer. This committed offset will be used when the process fails as the position from which the new consumer will begin. Required: Optional Type: String default: true

autoCommitInterval
The frequency in ms that the consumer offsets are committed to zookeeper. Required: Optional Type: String default: 60*1000

autoOffsetReset
smallest : automatically reset the offset to the smallest offset largest : automatically reset the offset to the largest offset anything else: throw exception to the consumer Required: Optional Type: String default: largest

Details
There are two types of KAFKA consumers: High-level and Simple. This functions creates a high level consumer

Value
Returns a consumer

Author(s)
Shruti Gupta

References
To know when to use simple consumer and when to use High-level Consumer, refer the url below: https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example
To know how to use a high level consumer refer this: https://cwiki.apache.org/confluence/display/KAFKA/Consumer+Group+Example

Examples
```
## Not run:
consumer1=rkafka.createConsumer("127.0.0.1:2181","test123")
consumer2=rkafka.createConsumer("127.0.0.1:2181","test123","test-consumer-group","50000","1000")

## End(Not run)
```
rkafka.createProducer  Creating KAFKA producer

Description

This function is used to create a KAFKA producer.

Usage

```java
rkafka.createProducer(metadataBrokerList, producerType="sync",
compressionCodec="none", serializerClass=\"kafka.serIALIZer.StringEncoder\",
partitionerClass=\"NULL\", compressedTopics=\"NULL\",
queueBufferingMaxTime=\"NULL\", queueBufferingMaxMessages=\"NULL\",
queueEnqueueTimeoutTime=\"NULL\", batchNumMessages=\"NULL\")
```

Arguments

- **metadataBrokerList**
  List of brokers used for bootstrapping knowledge about the rest of the cluster.
  Format: host1:port1,host2:port2... Required: Mandatory Type: String default: localhost:9092

- **producerType**
  Specifies whether the messages are sent asynchronously (async) or synchronously (sync).
  Required: Mandatory Type: String default: sync

- **compressionCodec**
  Specify the compression codec for all data generated: none, gzip, snappy.
  Required: Mandatory Type: String default: none

- **serializerClass**
  Specifies the class for serialization.
  Required: Mandatory Type: String default: kafka.serializer.StringEncoder

- **partitionerClass**
  Name of the partitioner class for partitioning events.
  Required: Optional Type: String default: NULL (default: partition spreads data randomly)

- **compressedTopics**
  Allow topic level compression.
  Required: Optional Type: String default: NULL

- **queueBufferingMaxTime**
  Maximum time, in milliseconds, for buffering data on the producer queue.
  Required: Optional (for Async Producer only) Type: String default: NULL

- **queueBufferingMaxMessages**
  The maximum size of the blocking queue for buffering on the producer.
  Required: Optional (for Async Producer only) Type: String default: NULL

- **queueEnqueueTimeoutTime**
  0: events will be enqueued immediately or dropped if the queue is full.
  -ve: enqueue will block indefinitely if the queue is full.
  +ve: enqueue will block up to this many milliseconds if the queue is full.
  Required: Optional (for Async Producer only) Type: String default: NULL

- **batchNumMessages**
  The number of messages batched at the producer.
  Required: Optional (for Async Producer only) Type: String default: NULL
Value

Returns Producer

Author(s)

Shruti Gupta

Examples

```
# Not run:
producer1=rkafka.createProducer("127.0.0.1:9092")
producer2=rkafka.createProducer("127.0.0.1:9092","sync","none","kafka.serializer.StringEncoder")

# End(Not run)
```

---

**rkafka.createSimpleConsumer**

*Creating simple KAFKA consumer*

**Description**

This function creates the Simple Consumer

**Usage**

```
rkafka.createSimpleConsumer(kafkaServerURL, kafkaServerPort, connectionTimeOut, kafkaProducerBufferSize, clientId)
```

**Arguments**

- **kafkaServerURL**
  - Port number of the KAFKA server Required:Mandatory Type:String
- **kafkaServerPort**
  - Connection Timeout in ms Required:Mandatory Type:String
- **connectionTimeOut**
  - Buffer size Required:Mandatory Type:String
- **kafkaProducerBufferSize**
  - ID of the client Required:Mandatory Type:String
- **clientId**

**Details**

There are two types of KAFKA consumers: High-Level and Simple. This function creates the Simple Consumer. Use caution on deciding to use the Simple Consumer as it doesn’t persist offset.
rkafka.read

Value

Doesn’t return anything

Note

Warning: Ensure to run the rkafka.receiveFromSimpleConsumer() function before executing the rkafka.runFromSimpleConsumer() function

Author(s)

Shruti Gupta

References

To know when to use simple consumer and when to use High-level Consumer, refer the url below: https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example

Examples

```r
cat
```

rkafka.read KAFKA consumer reading messages(single)  

Description

This function reads messages received by a KAFKA consumer. It fetches one message at a time

Usage

```r
rkafka.read(ConsumerObj)
```

Arguments

```r
ConsumerObj   Consumer through which messages are to be read Required:Mandatory Type:Consumer
```

Details

This function returns one message at a time from the topic to which the consumer is associated. If no new message is found with 'x' time(set by ConsumerTimeoutMs property), then it returns ""

Value

String
rkafka.readFromSimpleConsumer

Note

Warning: Ensure to close the consumer after reading messages. Won’t work correctly next time otherwise

Author(s)
Shruti Gupta

References
To know when to use simple consumer and when to use High-level Consumer, refer the url below: https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example
To know how to use a high level consumer refer this: https://cwiki.apache.org/confluence/display/KAFKA/Consumer+Group+Example

Examples

```r
## Not run:
consumer1=rkafka.createConsumer("127.0.0.1:2181","test123")
print(rkafka.read(consumer1))

## End(Not run)
```

rkafka.readFromSimpleConsumer

KAFKA Simple Consumer Reading

Description
This function returns one message at a time which are read by a KAFKA Simple Consumer

Usage
rkafka.readFromSimpleConsumer(SimpleConsumerObj)

Arguments

SimpleConsumerObj
Consumer through which messages were received Required:Mandatory Type:Consumer

Details
There are two types of KAFKA consumers:High-Level and Simple. This function receives messages using the Simple Consumer. Use caution on deciding to use the Simple Consumer as it doesn’t persist offset. The function rkafka.receiveFromSimpleConsumer needs to be executed before running this function
Description

This function reads messages received by a KAFKA consumer. It returns a batch of messages.

Usage

rkafka.readPoll(ConsumerObj)

Arguments

ConsumerObj    Consumer through which messages are to be read Required:Mandatory Type:Consumer

Details

This function returns messages as a batch from the topic to which the consumer is associated. If no new message is found with 'x' time(set by ConsumerTimeoutMs property), then it returns ""

Value

Array of Strings
rkafka.receiveFromSimpleConsumer

**Note**

Warning: Ensure to close the consumer after reading messages. Won’t work correctly next time otherwise.

**Author(s)**

Shruti Gupta

**References**

To know when to use simple consumer and when to use High-level Consumer, refer the url below: https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example

To know how to use a high level consumer refer this: https://cwiki.apache.org/confluence/display/KAFKA/Consumer+Group+Example

**Examples**

```cpp
## Not run:

consumer1=rkafka.createConsumer("127.0.0.1:2181","test123")
print(rkafka.readPoll(consumer1))

## End(Not run)
```

**Description**

This function allows the KAKFA Simple Consumer to receive messages from a particular topic. However, this doesn’t display the messages. To read the messages, use the rkafka.readFromSimpleConsumer function.

**Usage**

```cpp
rkafka.receiveFromSimpleConsumer(SimpleConsumerObj, topicName, partition, Offset, msgReadSize)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleConsumerObj</td>
<td>Simple Consumer object through which messages are to be read</td>
</tr>
<tr>
<td>topicName</td>
<td>Name of the topic from where to read messages</td>
</tr>
<tr>
<td>partition</td>
<td>Partition Number</td>
</tr>
<tr>
<td>Offset</td>
<td>Offset Number</td>
</tr>
<tr>
<td>msgReadSize</td>
<td>Size of the message to be read</td>
</tr>
</tbody>
</table>
Details

There are two types of KAFKA consumers: High-Level and Simple. This function receives messages using the Simple Consumer. Use caution on deciding to use the Simple Consumer as it doesn’t persist offset. This function needs to be run before executing the rkafka.readFromSimpleConsumer function.

Value

Nothing

Note

Warning: Ensure to close the consumer after reading messages. Won’t work correctly next time otherwise.

Author(s)

Shruti Gupta

References

To know when to use simple consumer and when to use High-level Consumer, refer the url below: https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example

Examples

```r
## Not run:
consumer1=rkafka.createSimpleConsumer("172.25.1.78","9092","10000","100000","test")
rkafka.receiveFromSimpleConsumer(consumer1,"test","0","0","test-group")

## End(Not run)
```

---

rkafka.send  

*KAFKA producer sending message*

Description

This function sends message to a particular name through a producer

Usage

```
rkafka.send(producer, topicName, ip, message)
```
**Arguments**

- **producer**: Producer through which messages are to be sent Required:Mandatory Type:String
- **topicName**: Topic to which messages are to be sent. If topicName doesn’t exist, new topic is created Required:Mandatory Type:String
- **ip**: ip on which producer is running Required:Mandatory Type:String
- **message**: message to be sent Required:Mandatory Type:String

**Value**

Doesn’t return a value

**Author(s)**

Shruti Gupta

**Examples**

```ruby
# Not run:
producer1=rkafka.createProducer("127.0.0.1:9092")
rkafka.send(producer1,"test","127.0.0.1:9092","Testing")

# End(Not run)
```
Index

* Topic **Apache Kafka**
  - rkafka, 2
* Topic **rkafka**
  - rkafka.createProducer, 7
  - rkafka.send, 13
* Topic **Message sending**
  - rkafka.send, 13
* Topic **Producer**
  - rkafka.send, 13
  - rkafka.closeConsumer, 3
  - rkafka.closeProducer, 4
* Topic **close**
  - rkafka.closeConsumer, 3
  - rkafka.closeProducer, 4
  - rkafka.closesimpleconsumer, 4
* Topic **create**
  - rkafka.closesimpleconsumer, 8
  - rkafka.read, 9
  - rkafka.readFromSimpleConsumer, 10
  - rkafka.readPoll, 11
  - rkafka.receiveFromSimpleConsumer, 12
* Topic **declose**
  - rkafka.closeConsumer, 3
  - rkafka.closeSimpleConsumer, 4
  - rkafka.createConsumer, 5
  - rkafka.createSimpleConsumer, 8
  - rkafka.read, 9
  - rkafka.readFromSimpleConsumer, 10
  - rkafka.readPoll, 11
  - rkafka.receiveFromSimpleConsumer, 12
* Topic **decreate**
  - rkafka.closesimpleconsumer, 8
* Topic **devkafka**
  - rkafka.closeConsumer, 3
  - rkafka.closeProducer, 4
  - rkafka.closeSimpleConsumer, 4
  - rkafka.createConsumer, 5
  - rkafka.createSimpleConsumer, 8
  - rkafka.read, 9
  - rkafka.readFromSimpleConsumer, 10
  - rkafka.readPoll, 11
  - rkafka.receiveFromSimpleConsumer, 12
* Topic **deproducer**
  - rkafka.closeProducer, 4
  - rkafka.createProducer, 7

* Topic **read**
  - rkafka.read, 9
  - rkafka.readFromSimpleConsumer, 10
  - rkafka.readPoll, 11
* Topic **simpleread**
  - rkafka.closeSimpleConsumer, 8
  - rkafka.createSimpleConsumer, 8
  - rkafka.readFromSimpleConsumer, 10
  - rkafka.receiveFromSimpleConsumer, 12
  - autoCommitEnable
    - (rkafka.createConsumer), 5
  - autoCommitInterval
    - (rkafka.createConsumer), 5
  - autoOffsetReset
    - (rkafka.createConsumer), 5
  - batchNumMessages
    - (rkafka.createProducer), 7
  - clientId
    - (rkafka.createSimpleConsumer), 8
  - compressedTopics
    - (rkafka.createProducer), 7
  - compressionCodec
    - (rkafka.createProducer), 7
  - connectionTimeOut
    - (rkafka.createSimpleConsumer), 8
  - consumerTimeoutMs
    - (rkafka.createConsumer), 5
  - groupId
    - (rkafka.createConsumer), 5
  - ip
    - (rkafka.send), 13
  - kafkaProducerBufferSize
    - (rkafka.createSimpleConsumer), 8
INDEX

kafkaServerPort
   (rkafka.createSimpleConsumer), 8
kafkaServerURL
   (rkafka.createSimpleConsumer), 8
message (rkafka.send), 13
metadataBrokerList
   (rkafka.createProducer), 7
msgReadSize
   (rkafka.receiveFromSimpleConsumer), 12
Offset
   (rkafka.receiveFromSimpleConsumer), 12
partition
   (rkafka.receiveFromSimpleConsumer), 12
partitionerClass
   (rkafka.createProducer), 7
producerType (rkafka.createProducer), 7
queueBufferingMaxMessages
   (rkafka.createProducer), 7
queueBufferingMaxTime
   (rkafka.createProducer), 7
queueEnqueueTimeoutTime
   (rkafka.createProducer), 7
rkafka, 2
rkafka.closeConsumer, 3
rkafka.closeProducer, 4
rkafka.closeSimpleConsumer, 4
rkafka.createConsumer, 5
rkafka.createProducer, 7
rkafka.createSimpleConsumer, 8
rkafka.read, 9
rkafka.readFromSimpleConsumer, 10
rkafka.readPoll, 11
rkafka.receiveFromSimpleConsumer, 12
rkafka.send, 13
serializersClass
   (rkafka.createProducer), 7
SimpleConsumer
   (rkafka.closeSimpleConsumer), 4
SimpleConsumerObj
   (rkafka.readFromSimpleConsumer), 10
topicName (rkafka.send), 13
zookeeperConnect
   (rkafka.createConsumer), 5
zookeeperConnectionTimeoutMs
   (rkafka.createConsumer), 5