MarkLogic

MarkLogic Kafka Connector

Changes for version 1.2.3

Biju George
7-13-2020

Contents

Summary of this document

This document describes the changes made to the connector version 1.2.3. Below are the changes / capabilities added

New Capabilities

1. Added two configurable strategies for URI generation apart from the default UUID strategy

Documentation changes

1. Added examples for ProtoBuf messages
2. Added examples for JsonSchema messages

Configurations for URI strategy

Below are the new properties for the URI Strategy

Property	Description
`ml.id.strategy`	*JSONPATH*: Configure one document path here. The value in this path will become the last part of URI. *HASH:* Configure one or more comma separated document paths here. The value of last part of the URI would be the MD5 Hashed value of the values concatenated together. Refer the examples for details. This can be usable if you want to use a significant data in the document to form the URI, but do not want to explicitly have the value in the URI. *KAFKA_META_WITH_SLASH:*The URI is generated with the topic name, partition number and offset, separated by '/'. See the examples below. *KAFKA_META_HASHED:* The URI is generated with a MD5 hashed value of topic, partition and offset
`ml.id.strategy.paths`	The comma separated list of document paths for the id strategy. If `ml.id.strategy` is *JSONPATH*, only the first path is taken. For *HASH*, all values are concatenated for generating a HASH value. See the examples below. This property has no effect on *KAFKA_META_WITH_SLASH* and *KAFKA_META_HASHED* values

Examples

Take an example document as below.

```
{
    "Customer": {
        "id": 99102010,
        "FirstName": "Lee",
        "LastName": "Chung",
        "Address": {
            "Line1": "123 Main St.",
            "Line2": "",
            "City": "myCity",
            "State": "CA",
            "Zip": 90002
        },
        "IDs": [
            {
                "type": "SSN",
                "value": "323321111"
            },
            {
                "type": "RESPH",
                "value": "8888887333"
            },
            {
                "type": "MOBILE",
                "value": "8899287333"
            }
        ]
    }
}
```

Scenario 1: Generate URI with the id value.

Configuration:

```
ml.document.uriPrefix= /kafka-data/example/
ml.document.uriSuffix=.json
ml.id.strategy=JSONPATH
ml.id.strategy.paths=/Customer/id
```

URI Generated: `/kafka-data/example/99102010.json`

Notes:

1. If the path is not valid, the last part of the URI will be blank.
2. The connector will not ensure the uniqueness of the URI generated. So, make sure that the value chosen is unique or it is ok to overwrite.

Scenario2: Generate URI with a hash of id value

Configuration:

```
ml.document.uriPrefix= /kafka-data/example/
ml.document.uriSuffix=.json
ml.id.strategy=HASH
ml.id.strategy.paths=/Customer/id
```

URI Generated: `/kafka-data/example/95a033c8257bed4428205f1ee9b487f6.json`

Notes:

1. The connector will not ensure the uniqueness of the URI generated. So, make sure that the value chosen is unique or it is ok to overwrite.
2. MD5 Hashing algorithm is used to generate the hash.

Scenario3: Generate URI with a hash of id, AddressZip & SSN

Configuration:

```
ml.document.uriPrefix= /kafka-data/example/
ml.document.uriSuffix=.json
ml.id.strategy=HASH
ml.id.strategy.paths=/Customer/id,/Customer/Address/Zip,/Customer/IDs/0/value
```

URI Generated: `/kafka-data/example/d07cb9f262754f56b40f7caecee11ef3.json`

Here, `d07cb9f262754f56b40f7caecee11ef3` is the HASH of the string `991020109000232321111`

Scenario4: Generate URI from the Kafka meta data

Configuration:

```
topics = perf-queue
ml.document.uriPrefix= /kafka-data/example/
ml.document.uriSuffix=.json
ml.id.strategy=KAFKA_META_WITH_SLASH
ml.id.strategy.paths=/
```

URI Generated: `/kafka-data/example/perf-queue/0/678.json`

Scenario5: Generate URI from the hashed value from Kafka meta data

Configuration:

```
topics = perf-queue
ml.document.uriPrefix= /kafka-data/example/
ml.document.uriSuffix=.json
ml.id.strategy=KAFKA_META_HASHED
ml.id.strategy.paths=/
```

URI Generated: `/kafka-data/example/c41b46b198ead0aeade0e4fafa6a1743.json`

`c41b46b198ead0aeade0e4fafa6a1743` is the HASH of the string `perf-queue0378`

Notes:

1. The connector will not ensure the uniqueness of the URI generated. So, make sure that the value chosen is unique or it is ok to overwrite.
2. See the way of accessing the arrays by index and not by value. So, the order of the array is important is you are using this strategy. In future versions, more flexible options will be provided. For example, suppose non-existing array index is used
   `ml.id.strategy.paths=/Customer/id,/Customer/Address/Zip,/Customer/IDs/5/value`
   The output URI is `/kafka-data/example/25955243abb2bb2fd6964aff67fd7c27.json`

   Here, `25955243abb2bb2fd6964aff67fd7c27` is the HASH of `9910201090002`

Recipe for URI customization

If a different strategy which is not available now is required, then `extractId` method of interface `com.marklogic.client.ext.document. ContentIdExtractor` should be implemented and call the appropriate `extractId` method from the build method in `com.marklogic.client.ext.document. DocumentWriteOperation`. The implementations of current strategies can help in building unique strategies.

Support for ProtoBuf and Json Schema messages

Confluent Platform 5.5 adds support for Protocol Buffers and JSON Schema along with Avro, the original default format for Confluent Platform. The sink connector is updated and validated with this version of Confluent Platform. As in other schema registry enabled converters (Avro), the schema part is ignored when ingesting the message into MarkLogic. Instead only payload is ingested into MarkLogic. With these validations the MarkLogic Kafka sink connector supports all the Converters packaged with Confluent platform.

1. **AvroConverter** `io.confluent.connect.avro.AvroConverter`: use with Schema Registry
2. **ProtobufConverter** `io.confluent.connect.protobuf.ProtobufConverter`: use with Schema Registry
3. **JsonSchemaConverter** `io.confluent.connect.json.JsonSchemaConverter`: use with Schema Registry
4. **JsonConverter** `org.apache.kafka.connect.json.JsonConverter` (without Schema Registry): use with structured data
5. **StringConverter** `org.apache.kafka.connect.storage.StringConverter`: simple string format
6. **ByteArrayConverter** `org.apache.kafka.connect.converters.ByteArrayConverter`: provides a "pass-through" option that does no conversion

Note: The document does not intend to provide training about protocol buffers (protobuf) or json schema format. Please refer Confluent documentation for details about these formats.

Examples of Sink Configurations for Protobuf and JsonSchema

Below are some example configurations of the sink connector for using with Protobuf and Json with schema messages. One example is added for testing with the confluent tool. All examples are based on the below JSON document

```
{"Snack":{"name":"cake","calories":260,"colour":"white", "flavor":"vanilla"}}
```

Configuration for *JsonSchemaConverter*

```
name = MarkLogicSink
connector.class = com.marklogic.kafka.connect.sink.MarkLogicSinkConnector
tasks.max = 1
key.converter = io.confluent.connect.json.JsonSchemaConverter
value.converter = io.confluent.connect.json.JsonSchemaConverter
topics = perf-queue
ml.connection.host = 192.168.0.29
ml.connection.port = 8071
ml.connection.database = CertServer
ml.connection.securityContextType = BASIC
ml.connection.username = srportal
ml.connection.password = abc
ml.connection.type = DIRECT
ml.connection.simpleSsl = true
ml.connection.enableCustomSsl = false
ml.connection.certFile = \
ml.connection.certPassword = \
ml.connection.externalName = \
ml.connection.customSsl.hostNameVerifier = \
ml.connection.customSsl.tlsVersion = \
ml.connection.customSsl.mutualAuth = false
ml.dmsdk.batchSize = 200
ml.dmsdk.threadCount = 10
ml.dmsdk.transform = \
ml.dmsdk.transformParams = \
ml.document.collections = coll-simplessl
ml.document.format = JSON
ml.document.mimeType = \
ml.document.permissions = rest-reader,read,rest-writer,update
ml.document.uriPrefix = /kafka-data/simplessl/
ml.document.uriSuffix = .json
key.converter.schemas.enable = false
value.converter.schemas.enable = false
errors.tolerance = all
errors.log.enable = true
errors.deadletterqueue.topic.name = test-dlq
errors.deadletterqueue.context.headers.enable = true
consumer.max.poll.records=200
max.poll.records=200
batch-size-max=200
key.converter.schema.registry.url=http://localhost:8081
value.converter.schema.registry.url=http://localhost:8081
```

Testing example of *JsonSchemaConverter*

```
./kafka-json-schema-console-producer --broker-list localhost:9092 --topic perf-queue --property value.schema='{
    "$schema": "http://json-schema.org/draft-06/schema#",
    "$ref": "#/definitions/Snack",
    "definitions": {
        "Snack": {
            "type": "object",
            "additionalProperties": false,
            "properties": {
                "Snack": {
                    "$ref": "#/definitions/SnackClass"
                }
            },
            "required": [],
            "title": "Snack"
        },
        "SnackClass": {
            "type": "object",
            "additionalProperties": true,
            "properties": {
                "name": {
                    "type": "string"
                },
                "calories": {
                    "type": "integer"
                },
                "colour": {
                    "type": "string"
                }
            },
            "required": [],
            "title": "SnackClass"
        }
    }
}
'

{"Snack":{"name":"cake","calories":260,"colour":"white", "flavor":"vanilla"}}
```

Configuration for *ProtobufConverter*

```
name = MarkLogicSink
connector.class = com.marklogic.kafka.connect.sink.MarkLogicSinkConnector
tasks.max = 1
key.converter = io.confluent.connect.protobuf.ProtobufConverter
value.converter = io.confluent.connect.protobuf.ProtobufConverter
topics = perf-queue
ml.connection.host = 192.168.0.29
ml.connection.port = 8071
ml.connection.database = CertServer
ml.connection.securityContextType = BASIC
ml.connection.username = srportal
ml.connection.password = abc
ml.connection.type = DIRECT
ml.connection.simpleSsl = true
ml.connection.enableCustomSsl = false
ml.connection.certFile = \
ml.connection.certPassword = \
ml.connection.externalName = \
ml.connection.customSsl.hostNameVerifier = \
ml.connection.customSsl.tlsVersion = \
ml.connection.customSsl.mutualAuth = false
ml.dmsdk.batchSize = 200
ml.dmsdk.threadCount = 10
ml.dmsdk.transform = \
ml.dmsdk.transformParams = \
ml.document.collections = coll-simplessl
ml.document.format = JSON
ml.document.mimeType = \
ml.document.permissions = rest-reader,read,rest-writer,update
ml.document.uriPrefix = /kafka-data/simplessl/
ml.document.uriSuffix = .json
key.converter.schemas.enable = false
value.converter.schemas.enable = false
errors.tolerance = all
errors.log.enable = true
errors.deadletterqueue.topic.name = test-dlq
errors.deadletterqueue.context.headers.enable = true
consumer.max.poll.records=200
max.poll.records=200
batch-size-max=200
key.converter.schema.registry.url=http://localhost:8081
value.converter.schema.registry.url=http://localhost:8081
```

Testing example of *ProtobufConverter*

```
./kafka-protobuf-console-producer --broker-list localhost:9092 --topic perf-queue --property value.schema='
syntax = "proto3";
message Snack {
        message obj {
           string name = 1;
           double calories = 2;
           string colour = 3;
           string flavor = 4;
   }
   obj Snack = 1;
}
'
{"Snack":{"name":"cake","calories":260,"colour":"white", "flavor":"vanilla"}}
```

==========