Denodo Kafka Custom Wrapper User Manual

Revision 20221107

Denodo Inc.: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA www.denodo.com

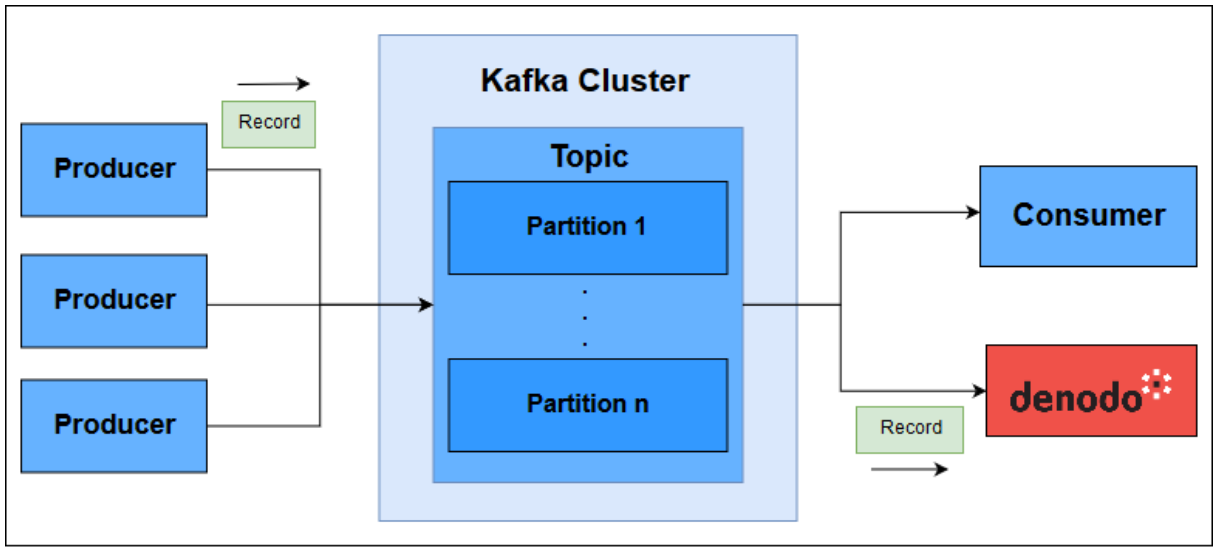
CONTENTS

1 INTRODUCTION

Apache Kafka is a distributed streaming platform. Initially conceived as a messaging queue, Kafka is based on an abstraction of a distributed commit log and is used for building real-time data pipelines and streaming apps.

As a streaming platform, it has three key capabilities:

- Publish and subscribe to streams of records.
- Store streams of records in a fault-tolerant durable way.
- Process streams of records as they occur.



Denodo Kafka Custom Wrapper as a Consumer

The Denodo Kafka Custom Wrapper allows you to consume records in two different ways:

1. **Between dates:** read the records of a topic within a specific time interval. If no interval is given all the records will be read.
2. **Incrementally:** read new records of a topic since the last query was made.

2 INSTALLATION

The Denodo Kafka Custom Wrapper distribution consists of:

- `/dist`:
  - `denodo-kafka-customwrapper-{denodo-version}-{version}.jar`. The custom wrapper.
  - `denodo-kafka-customwrapper-{denodo-version}-{version}-jar-with-dependencies.jar`. The custom wrapper plus its dependencies. This is the package we recommend to use, as it is easier to install in VDP.
  - `denodo-kafka-customwrapper-{denodo-version}-{version}-sources`. The custom wrapper source code.

- `/lib`: All the dependencies required by this wrapper in case you need to use the `denodo-kafka-customwrapper-{denodo-version}-{version}.jar`.

3 USAGE

3.1 IMPORTING THE CUSTOM WRAPPER

To import the custom wrapper, follow these steps:

1. In the VDP Administration Tool, go to `File → Extension management`
2. Click on "`Create`" button and select the "`denodo-kafka-customwrapper-{denodo-version}-{version}-jar-with-dependencies.jar`" file, located in the `dist` folder of the Denodo Kafka Custom Wrapper distribution, downloaded from the Denodo Support Site.

3.2 CREATING THE KAFKA DATA SOURCE

To create a new Kafka custom data source:

1. In the VDP Administration Tool, go to: `File → New… → Data source → Custom`
2. In the "`Create a New Custom Data Source`" window, do the following:
   - Set a name for the new Kafka data source in the "`Name`" field.
   - Click on "`Select Jars`" and select the file imported in the previous section.
   - In the "`Class name`" field, depending on the way you want to consume records, the selector component can be used to switch between these classes:
     - `com.denodo.connect.kafka.wrapper.KafkaDateRangeConsumerWrapper`: consume records between dates.
     - `com.denodo.connect.kafka.wrapper.KafkaIncrementalConsumerWrapper`: consume records incrementally.
3. Click to refresh the input parameters of the data source.
4. Configure the data source parameters:
   - **Connection String** (*mandatory*): is a comma-separated list of host and port pairs that are the addresses of the Kafka brokers. Host and port pair uses "`:`" as the separator, e.g.: `<kafka_host>:9092`.
   - **Connection Timeout (ms)**: is the maximum time (in milliseconds) that the custom wrapper will be connected to Kafka to consume data. This timeout is necessary because, since the Kafka consumer is continually waiting to receive new records, it is necessary to establish a maximum connection time so that eventually the consumption of data by the wrapper ends. Default is 10 minutes.

- **Consumer Timeout (ms)**: is the maximum time (in milliseconds) that the Kafka consumer will wait to receive data from the server during a poll operation. Must not be negative. Default is 7,000 (7s).
- **Max Poll Records**: maximum number of records that will be retrieved from the server with each poll operation. Default is 20,000, but this number might be configured in order to better adapt to the size of the messages being retrieved.
- **SSL Enabled**: if checked, the SSL protocol will be used. Note that, if the Kafka broker certificate is self signed, it must be imported into the truststore of the JRE used by Denodo Server, i.e., `<DENODO_HOME>/jre/lib/security/cacerts`.
- **Custom Properties Path**: is the route to a properties file that contains extra configuration properties of Kafka. Note that, if a configuration property is present both in the properties file and wrapper configuration fields, the property that have been entered through the wrapper configuration fields always will have more priority.

  Example of custom properties file for SSL configuration when you do not want to use the truststore of the JRE included in the Denodo Platform:

```
ssl.truststore.location=/path/to/truststore
ssl.truststore.password=test1234
```

  Valid configuration properties are documented at https://kafka.apache.org/documentation/#consumerconfigs

5. Click on the "Save" button.

3.3 DATE RANGE VS INCREMENTAL WRAPPERS

As mentioned, the Denodo Kafka Custom Wrapper offers two different implementations:

The **Date Range implementation** will be able to retrieve all messages in a topic between a specified date range (begin date inclusive, end date exclusive)

Executions of base views created on this wrapper can be concurrent, and they should always return the same result set as long as the message dataset at the server does not vary.

This wrapper requires a prefix to be defined for the *consumer group ID*, so that each execution uses a new, random consumer group ID starting with the defined prefix in order to avoid concurrent executions being assigned different sets of partitions in the topic and therefore returning partial data sets (standard partition balance behavior in Kafka).

The **Incremental implementation** will ask the server for the latest messages added to the topic's partitions since the last time the base view was executed.

Executions of base views created on this wrapper consume the available message and then commit their latest read offset to the server, so that the next time it is executed the server will only send messages with newer offsets (if any). For this reason the wrapper requires a *consumer group ID* to be fixed at the base view configuration, which will be used for all executions of that base view.

Note that concurrent executions of this wrapper for the same base view, or for base views reading from the same topic that also share the same consumer group ID (which is not recommended) will cause the Kafka server to re-balance topic partitions among the wrapper executions and therefore send a different set of results to each of the concurrent executions.

3.4 CREATING THE BASE VIEW

To create a new base view using the Kafka data source:

1. Double-click on the Kafka data source and then click on "`Create base view`".
2. Set the parameters as follows:
    - **Topic** (*mandatory*): Kafka brokers contain topics that act like a message queue where client applications can write and read their data. All Kafka messages are organized into topics. Producers write data to topics and consumers read from them.
    - **Message Format** (*mandatory*): is the format that the message has. Supported formats are:
        - String
        - Avro
        - Avro (Confluent) — *Confluent's specific Avro format.*
        - Protobuf
        - Protobuf (Confluent) — *Confluent's specific Protobuf format.*
        - JSON
    - **Schema Path**: When using "`Avro`", "`Protobuf`" or "`JSON`" as a message format, this field –mandatory in these cases– allows the specification of the schema to be used for deserializing messages in these formats. This can be either a local file or an HTTP route pointing to an endpoint in a schema registry that returns the specific schema to use (e.g. http://confuentserver:8081/schemas/ids/1 for the schema with id `1` in a Confluent Schema Registry).

      Avro Schema sample

      ```
      {
      ```

```
  "namespace": "Message",
  "type": "record",
  "name": "EventMessage",
  "fields": [
      { "name": "machine", "type": "string" },
      { "name": "id", "type": "string" },
      { "name": "date", "type": "long" },
      { "name": "status", "type": "float" },
      { "name": "building", "type": "string" }
  ]
}
```

JSON Schema sample

```
{
    "title": "Message",
    "type": "object",
    "properties": {
        "type": { "type": "string" },
        "t": { "type": "number" },
        "k": { "type": "integer" }
    }
}
```

Protobuf Schema sample

```
syntax = "proto3";

package test.protos;

option java_multiple_files = true;

message FirstMessage {
    int32 id = 1;
    string name = 2;
    map<int64, SecondMessage> messages = 3;

    oneof company_oneof{
       repeated int64 numInfo = 4;
       SecondMessage newMessage = 5;
    }
}

message SecondMessage {
```

```
  enum Enumeration {
    ONE = 0;
    TWO = 1;
  }

  int64 id = 1;
  string name = 2;
  Enumeration enumeration = 3;
}
```

The fields mentioned above are common to all the consumer ways. However, there are extra fields in some of the wrappers. These are:

- Incremental consumer:
  - **Group ID** (*mandatory*): specifies the name of the Kafka Consumer Group the custom wrapper will belong to.
- Consumer between dates:
  - **Group ID Prefix** (optional, recommended): Prefix to be used for computing a new consumer group ID for each execution so that concurrent executions are not affected by partition rebalancing. A default prefix ("denodo") will be used if not specified.
  - **Begin Date** (*optional*): is the minimum date from which records are obtained, (inclusive).
  - **End Date** (*optional*): is the maximum date from which records are obtained (exclusive).

3.5 SECURELY ACCESSING THE KAFKA CLUSTER WITH KERBEROS

The configuration for accessing Kafka with Kerberos enabled requires supplying the Kerberos credentials to the wrapper's configuration.

The Kerberos parameters are:

- **Kerberos principal name**: Kerberos v5 Principal name to access Kafka, e.g. `primary/instance@realm`.

! Note

If you enter a literal that contains one of the special characters used to indicate interpolation variables `@`, `\`, `^`, `{`, `}`, you have to escape these characters with `\`.

E.g if the Kerberos principal name contains @ you have to enter \@.

- **Kerberos keytab file**: Keytab file containing the key of the Kerberos principal.
- **Kerberos Distribution Center**: Kerberos Key Distribution Center.
- **Kerberos Service Name**: The custom wrapper's configuration must include a property called 'sasl.kerberos.service.name'. The default value of that property is set as 'kafka'. But, if a different value is desired instead of the default one, you must include a new entry with the new value for the property in the properties file loaded with the 'Custom Properties Path' field. For example, the file must contain a line like this: sasl.kerberos.service.name=your_service_name

! Note

Except 'Kerberos Service Name', all previous parameters are mandatory when Kerberos is enabled.

A **'krb5.conf' file should be present in the file system**. Below there is an example of the Kerberos configuration file:

```
[libdefaults]
  renew_lifetime = 7d
  forwardable = true
  default_realm = EXAMPLE.COM
  ticket_lifetime = 24h
  dns_lookup_realm = false
  dns_lookup_kdc = false

[domain_realm]
  <domain_name> = EXAMPLE.COM

[realms]
  EXAMPLE.COM = {
    admin_server = <your_admin_server>
    kdc = <your_kdc>
  }

[logging]
  default = FILE:/var/log/krb5kdc.log
  admin_server = FILE:/var/log/kadmind.log
  kdc = FILE:/var/log/krb5kdc.log
```

The algorithm to locate the krb5.conf file is the following:

- If the system property `java.security.krb5.conf` is set, its value is assumed to specify the path and file name.
- If that system property value is not set, then the configuration file is looked for in the directory
  - `<java-home>\lib\security` (Windows)
  - `<java-home>/lib/security` (Solaris and Linux)
- If the file is still not found, then an attempt is made to locate it as follows:
  - `/etc/krb5/krb5.conf` (Solaris)
  - `c:\winnt\krb5.ini` (Windows)
  - `/etc/krb5.conf` (Linux)

3.6 EXAMPLE

1. In order to show an example of execution of the wrapper, a set of messages have been published in Kafka. These records have been added in the topic '`my-test-topic`' by the command-line based producer included in the Kafka's in

   The execution of the command provides a prompt through which records with `key:value` format are inserted.

```
 (Re-)joining group
19/03/04 21:59:12 INFO internals.AbstractCoordinator: [Consumer clientId=consumer-1, groupId=console-consumer-20694]
 Successfully joined group with generation 1
19/03/04 21:59:12 INFO internals.ConsumerCoordinator: [Consumer clientId=consumer-1, groupId=console-consumer-20694]
 Setting newly assigned partitions [my-test-topic-0]
CreateTime:1551731989407        key1    test-message-1
CreateTime:1551732001215        key2    test-message-2
CreateTime:1551732010191        key3    test-message-3
CreateTime:1551732019229        key4    test-message-4
CreateTime:1551732038223        key5    test-message-5
```

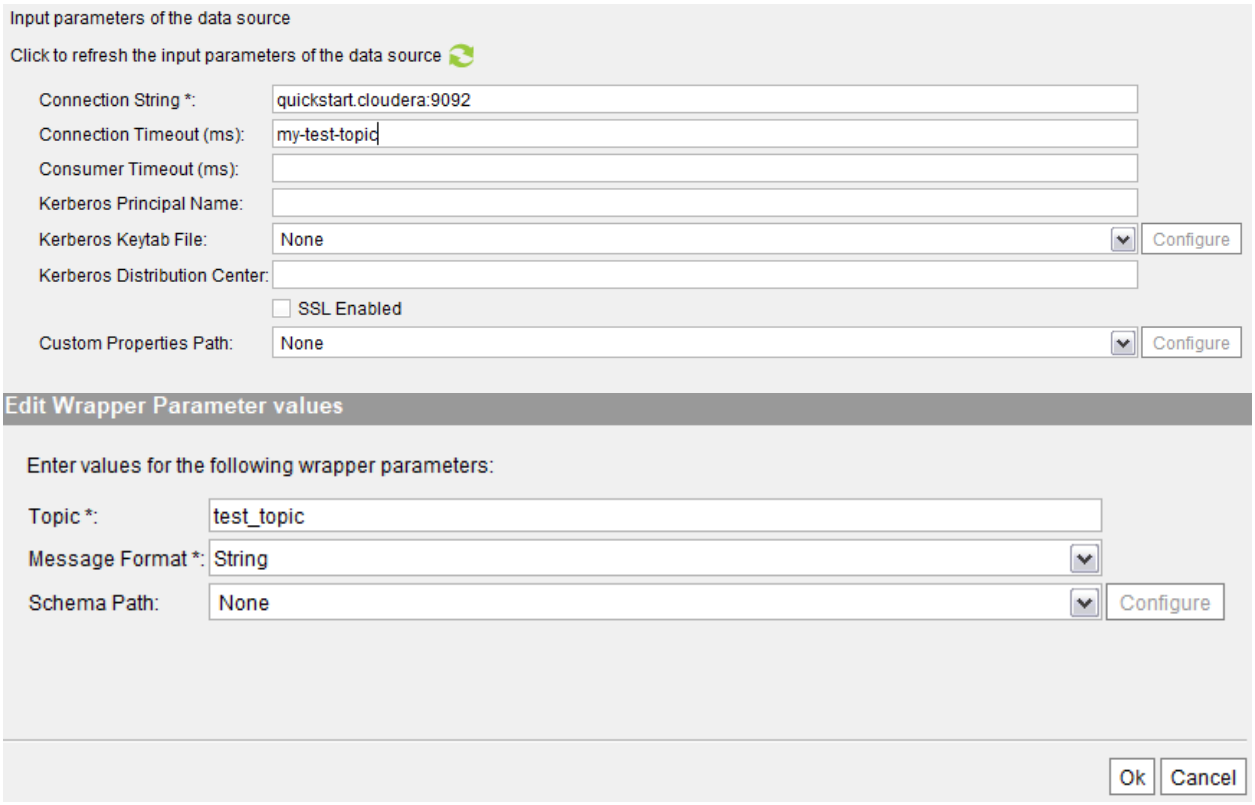
   Once inserted, the data can be consulted through the Kafka's command-line based consumer.

```
> kafka-console-consumer --new-consumer --topic test-topic --from-beginning \
  --bootstrap-server 192.168.56.101:9092 --property print.timestamp=true \
  --property print.key=true  --property print.value=true
```

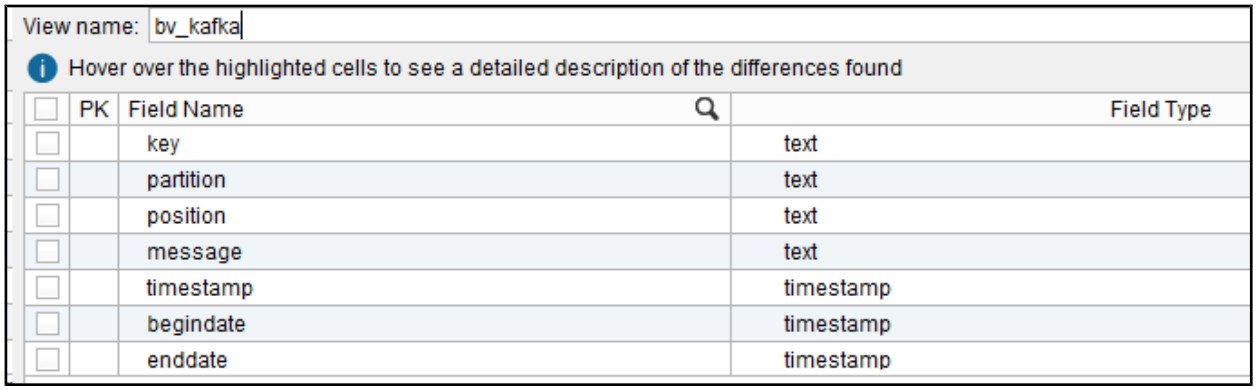
   The inserted records can be seen in the output of the command execution.

```
 (Re-)joining group
19/03/04 21:59:12 INFO internals.AbstractCoordinator: [Consumer clientId=consumer-1, groupId=console-consumer-20694]
 Successfully joined group with generation 1
19/03/04 21:59:12 INFO internals.ConsumerCoordinator: [Consumer clientId=consumer-1, groupId=console-consumer-20694]
 Setting newly assigned partitions [my-test-topic-0]
CreateTime:1551731989407        key1    test-message-1
CreateTime:1551732001215        key2    test-message-2
CreateTime:1551732010191        key3    test-message-3
CreateTime:1551732019229        key4    test-message-4
CreateTime:1551732038223        key5    test-message-5
```

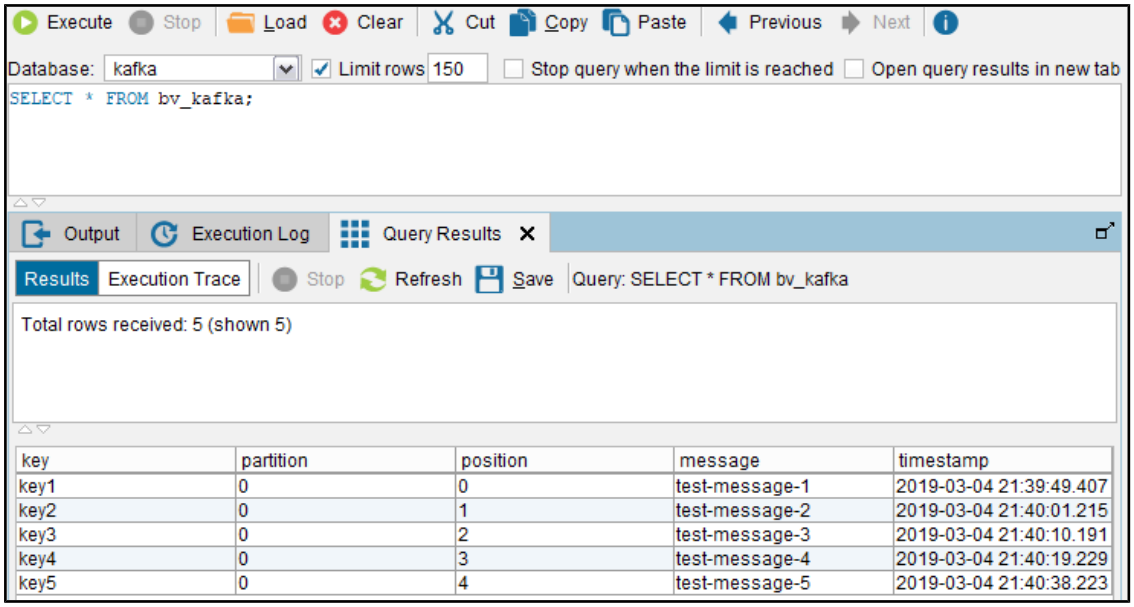
2. Create a base view:
    - Connection String = `192.168.56.102:9092`
    - Topic = `my-test-topic`
    - Message Format: `String`



3. The schema of the base view is shown and you can rename it:



4. After clicking on "Ok", you can execute `SELECT` queries, for example:
    - `SELECT * FROM bv_kafka;`



4 LIMITATIONS

4.1 INCREMENTAL WRAPPER DATA LOSS

As `OFFSET`, `FETCH` and `LIMIT` clauses are not delegated to the custom wrapper the use of these clauses can cause data loss. The same could happen with the `WHERE` clause.

4.2 USE OF SCHEMA REGISTRIES FOR AVRO / PROTOBUF / JSON

Schema Registries are not a standard feature of the Avro, Protobuf or JSON standards, but they are a more or less common tool offered by Kafka messaging providers. The Denodo Kafka Custom Wrapper provides support for retrieving schemas from any schema registries that can return a schema as the payload of an URL, and also support for the Confluent Schema Registry which returns schemas in a custom payload format.

Note however there are some limitations to the use of schema registries in this wrapper:

- Due to the static nature of the structure of Denodo views, the schema to be used for a Kafka-based view has to be uniquely identified by the URL that returns it (i.e. by specifying the ID it has been assigned in the registry). Schema registries are usually aimed at easily supporting the fact that each Avro/Protobuf-serialized message in a Kafka topic can (potentially) be shaped according to its own schema. The binary serialization of the message would contain, prefixed to the actual Avro payload, the ID of the schema to be used for the deserialization of that Avro payload (message), and such schema would then be retrieved on-the-fly for each message by the system performing the deserialization. This doesn't however match the requirements of a Denodo view, which requires the schema to be fixed and all the messages to be read from the view's topic to conform to that same schema.
- Schemas specified in views (either via a local file path or a schema registry URL) need to be fully self-contained. They cannot include type definitions from other schema files that would need separate download.
- Schemas specified as an URL will be retrieved every time the custom wrapper is executed, so specifying schemas as a local file path is always recommended when frequent calls to the schema registry might pose a performance issue.

4.3 AVRO LOGICAL TYPES

Only one logical type is supported in Avro schemas: `decimal`.

4.4 PROTOBUF TYPES AND SYNTAX

- The syntax used in the proto files must be proto3.
- The `any` type is not supported in `proto` files due to limitations in Google's own runtime libraries for protobuf.

5 USE CASE

5.1.1 Prices of an airline

Air tickets may have variations in their price every so often. In this context, Kafka can be very useful, since it allows each price change to be stored in a topic.

For this, a record with information relative to the flight at a certain moment could be stored (in JSON format, for instance) in that topic. This record would consist, for example, of the flight identifier and its associated price.

Denodo Kafka Custom Wrapper, using the incremental topic reading mode, can act as a consumer of the information stored in that topic. In this way, you can check the price changes of a flight since the last time they were consulted.

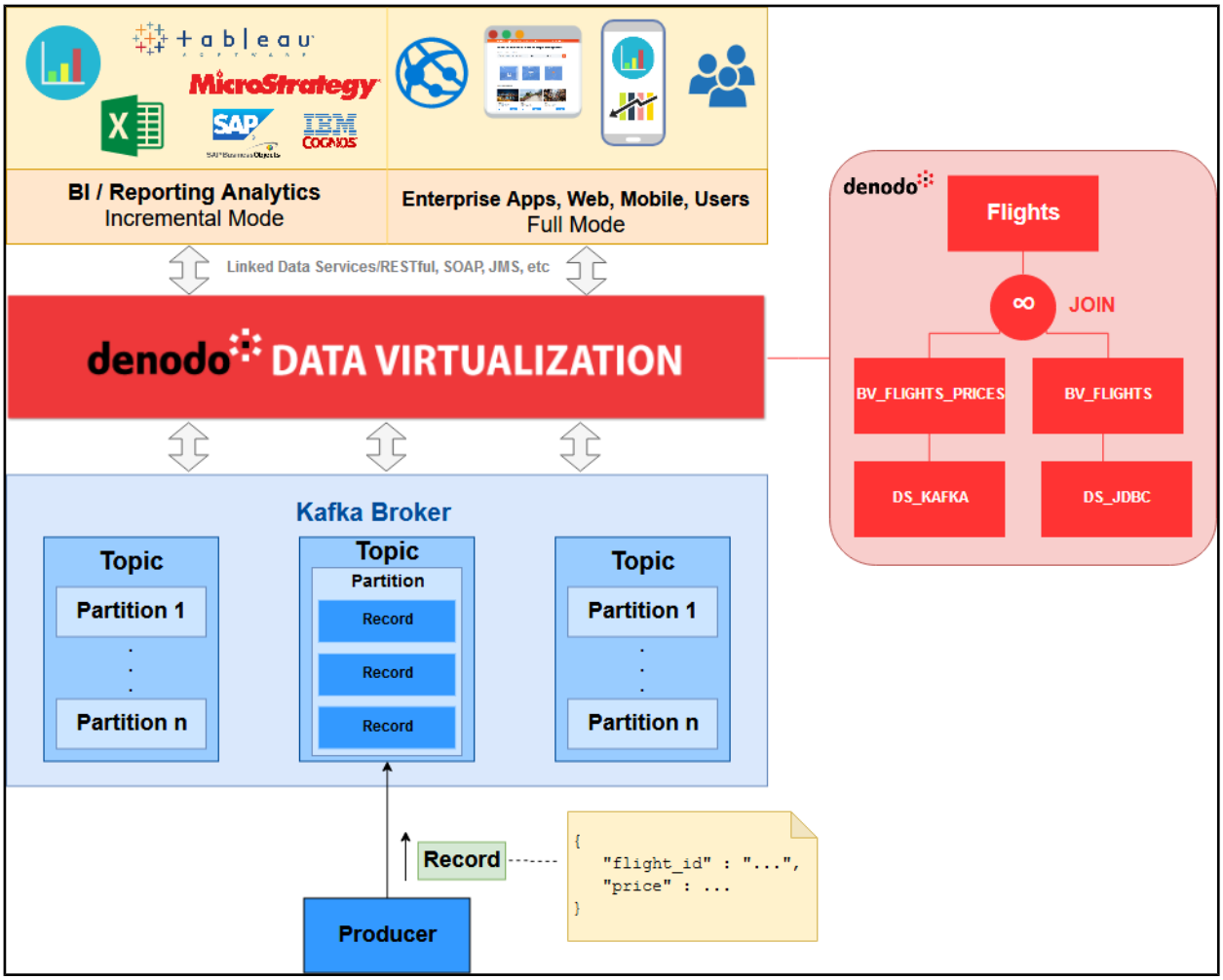
On the other hand, through the between dates consumption mode of a topic that has the Denodo Kafka Custom Wrapper, you can check how flight prices change over time.

The information obtained from the wrapper can be enriched by combining the information stored in the topic with the data of a view that contains the information about flights (flight identifier, airport of origin, airport of destination, date of departure, duration, etc.). This may be done through a JOIN clause.

Finally, to keep the cache of that join view updated, a Denodo Scheduler job of type VDPCache may be used to preload the cache of the cached views every certain time (every day, for instance).

Depending on the way of consumption used by the wrapper, the data can be applied to different contexts. For example, the data collected through the time interval reading mode can be very useful in Business Intelligence\Reporting Analytics applications. While, on the other hand, if the incremental reading mode is used, the data may be applicable in environments such as enterprise applications, web pages, etc.

In the following image you can see a general description of the air tickets use case.



6 TROUBLESHOOTING

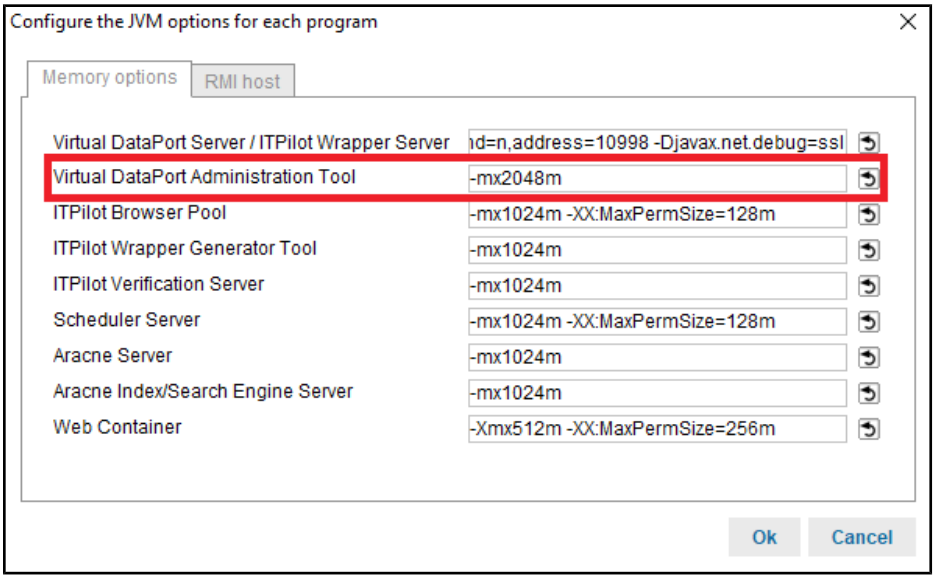
Symptom

Importing the custom wrapper produces a Java Heap Space:

Resolution

Increase the memory allocated to the VDP Admin Tool. To do so, follow the next steps:

1. In the Denodo Platform Control Center, go to '`Configure`'.
2. Click con '`JVM Options`'.
3. On the '`Memory Options`' tab, increase the allocated memory to the Virtual DataPort Administration Tool.



In this example, the memory has been increased to 2048Mb.

Symptom

`org.apache.kafka.common.network.SslTransportLayer [] - Failed to send SSL Close message`
`javax.net.ssl.SSLHandshakeException:` **`DHPublicKey does not comply to algorithm constraints`**

Resolution

The problem arises when the wrapper makes use of its SSL capabilities and the server only supports weak ciphers. To solve it there are two possibilities:

1. Upgrade the Java version of Kafka's server.
2. Open the file '`java.security`' located in '`<DENODO_HOME>\jre\lib\security`'.

Locate the next entry:

```
jdk.tls.disabledAlgorithms=SSLv3,  RC4,  MD5withRSA,  DH  keySize  <
1024, \
      EC keySize < 224, DES40_CBC, RC4_40
```

And replace it with:

```
jdk.tls.disabledAlgorithms=SSLv3, RC4, MD5withRSA, \
      EC keySize < 224, DES40_CBC, RC4_40
```

That is, '`DH keySize < 1024`' must be deleted.

! Note

The first solution offered is more secure, so we recommend using it instead of the second one.

7 REFERENCES

Kafka official page:

- https://kafka.apache.org/

Kafka & Confluent:

- https://www.confluent.io/what-is-apache-kafka/

JSON Schema:

- http://json-schema.org/

Authentication using Kerberos:

- https://docs.confluent.io/current/kafka/authentication_sasl/index.html

Encryption with SSL

- https://docs.confluent.io/current/kafka/encryption.html

Encryption and Authentication with SSL

- https://docs.confluent.io/current/kafka/authentication_ssl.html

8 APPENDIX

8.1 HOW TO CONNECT TO MAPR EVENT STORE (STREAMS)

From MapR documentation: *"MapR Event Store is a global publish-subscribe event streaming system for big data."*

As the Kafka API provides access to MapR Event Store, you can use the Kafka Custom Wrapper to connect to MapR Streams. This section explains how to do that.

8.1.1 Install MapR Client

To connect to the MapR cluster you need to install the MapR Client on your client machine (where the VDP server is running):

- Verify that the operating system on the machine where you plan to install the MapR Client is supported, see MapR Client Support Matrix.
- Obtain the MapR packages at https://package.mapr.com/releases/ and complete the installation steps explained in https://mapr.com/docs/home/AdvancedInstallation/SettingUptheClient-install-mapr-client.html. These steps are highly dependent on the operating system.

Set $MAPR_HOME environment variable to the directory where MapR client was installed. If MAPR_HOME environment variable is not defined /opt/mapr is the default path.

8.1.2 Copy `mapr-clusters.conf` file

Copy mapr-clusters.conf from the MapR cluster to the $MAPR_HOME/conf folder in the VDP machine.

```
demo.mapr.com secure=true maprdemo:7222
```

8.1.3 Generate MapR ticket (secure clusters only)

Every user who wants to access a secure cluster must have a MapR ticket (maprticket_<username>) in the temporary directory (the default location).

Use the $MAPR_HOME/maprlogin command line tool to generate one:

```
C:\opt\mapr\bin>maprlogin.bat password -user mapr

[Password for user 'mapr' at cluster 'demo.mapr.com': ]
MapR credentials of user 'mapr' for cluster 'demo.mapr.com' are written to 'C:\Users\<username>\AppData\Local\Temp/maprticket_<username>'
```

! Note

If you get an error like

```
java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be non-empty when executing maprlogin
```

you need to specify a truststore before executing the maprlogin command.

For this, you can copy the /opt/mapr/ssl_truststore from MapR Cluster to $MAPR_HOME/conf directory in the local machine.

8.1.4 Add JVM option

Add -Dmapr.library.flatclass to the VDP Server JVM options.

VDP Server JVM options

Otherwise, VDP will throw the exception java.lang.UnsatisfiedLinkError from JNISecurity.SetParsingDone() while executing the Kafka Custom Wrapper.

8.1.5 Create custom data source

In order to use the MapR vendor libraries you should not import the Kafka Custom Wrapper into Denodo.

You have to create the custom data source using the 'Classpath' parameter instead of the 'Select Jars' option. Click Browse to select the directory containing the required dependencies for this custom wrapper:

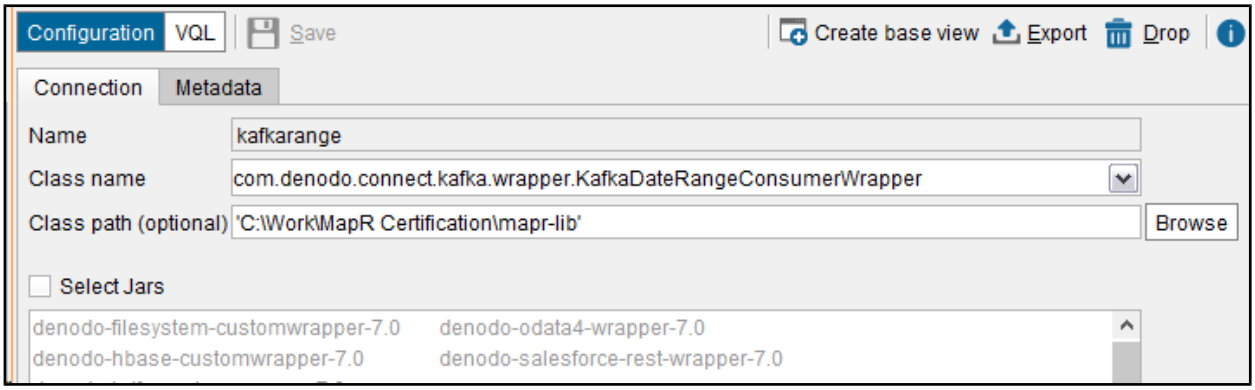
- The denodo-kafka-customwrapper-${version}.jar file of the dist directory of this custom wrapper distribution.

- The contents of the lib directory of this custom wrapper distribution, replacing the Apache Hadoop libraries with the MapR ones.

  The MapR Maven repository is located at http://repository.mapr.com/maven/. The name of the JAR files that you must use contains the version of Hadoop, Kafka, Zookeeper and MapR that you are using:

- hadoop-xxx-<hadoop_version>-<mapr_version>
- kafka-xxx-<hadoop_version>-<mapr_version>
- connect-xxx-<kafka version>-<mapr_version>
- maprfs-<mapr_version>
  As MapRClient native library is bundled in maprfs-<mapr_version> jar you should use the maprfs jar that comes with the Mapr Client, previously installed, as the library is dependent on the operating system.

- mapr-streams-<mapr_version>
- zookeeper-<zookeeper_version>-<mapr_version>
- json-<version>
- the other dependencies of the lib directory of this custom wrapper distribution



! Important

MapR native library is included in these Custom Wrapper dependencies and can be loaded only once.

Therefore, if you plan to access to other MapR sources with Denodo, like:

- MapR FileSystem with HDFS Custom Wrapper
- MapR Database with HBase Custom Wrapper
- Drill with JDBC Wrapper.

you have to use the same classpath to configure all the custom wrappers and the JDBC driver; see 'C:\Work\MapR Certification\mapr-lib' in the image above.

With this configuration Denodo can reuse the same classloader and load the native library only once.

8.1.6 Configure base view

Configure the Kafka wrapper parameters as usual, with these main differences:

- **`connection`** is the Mapr Cluster address as there is no broker in the Mapr Event Store implementation.

- the **`topic`** field should follow the MapR Streams naming convention `/stream:topic`, e.g. `/sample-stream:fast-messages`

MapR base view edition