Dbvisit Replicate Connector for Kafka documentation

Release 1.0.0-SNAPSHOT

Dbvisit Software Limited

January 12, 2017

Contents

Contents:

CHAPTER 1

Dbvisit Replicate Connector for Kafka

The **Dbvisit Replicate Connector for Kafka** is a SOURCE connector for the Kafka Connect framework, run within the Confluent Platform.

This connector enables you to stream DML change data records (inserts, deletes, updates) from an Oracle database, which have been made available in PLOG (parsed log) files generated by the Dbvisit Replicate application, through into Kafka topics.

You can find the Dbvisit Replicate Connector for Kafka project on GitHub, and which you can build from source, or you can download the prebuilt JAR file from this location.

1.1 Overview

1.1.1 The Oracle Source

Change data is identified and captured on the Oracle RDBMS by the Dbvisit Replicate application, using proprietary technology to mine the online redo logs in real time. Committed changes (pessimistic commit mode) made on the Oracle database are streamed in real time, in a custom binary format called a PLOG file, to a location where the **Dbvisit Replicate Connector for Kafka** picks them up, processes and delivers to Kafka topics. Note that only changes made to tables or schemas you are interested in listening for, and which have been configured to be included in the replication, are delivered to the PLOG files.

Please refer to the Dbvisit Replicate online user guide for information and instructions on setting and configuring that application to produce and deliver the PLOG file stream.

1.1.2 The Kafka Target

The **Dbvisit Replicate Connector for Kafka** polls the directory where the PLOGs will be delivered, picking up and streaming the changes identified in these files into Kafka, via the Kafka Connect framework.

By default all changes for a table are delivered to a single topic in Kafka. Topics are automatically generated, although can be pre-created, but it is important to note that the mapping is as follows:

```
Oracle table -> Kafka topic
```

In addition to this the **Dbvisit Replicate Connector for Kafka** will also automatically create and write to a meta-data topic (the name of which can be configured for the connector) in Kafka, which lists out Oracle transaction information from across all the tables that changes have been configured to listen for. This can be utilized by Kafka consumers to reconstruct the precise global ordering of changes, across the various topics, as they occurred in order on the Oracle database.

By default the **Dbvisit Replicate Connector for Kafka** works with the Avro converters, provided as part of the Kafka Connect framework, making use of the Schema Registry metadata service to govern the shape (and evolution) of the messages delivered to Kafka. This is a natural fit when working with highly structured RDBMS data, and the recommended approach for deployment.

1.2 Quickstart

To demonstrate the operation and functionality of the connector, we have provided a couple example sets of PLOG files, generated by running the Swingbench load generation utility, against an Oracle XE 11g database. The PLOGs containing a smaller dataset can be downloaded from this location, and contain the following number of change records, across the tables included in the replication:

```
SOE.CUSTOMERS:     Mine:156
SOE.ADDRESSES:     Mine:156
SOE.CARD_DETAILS: Mine:155
SOE.ORDER_ITEMS:  Mine:984
SOE.ORDERS:        Mine:809
SOE.INVENTORIES:  Mine:939
SOE.LOGON:         Mine:1006
TX.META:           Mine:1054
```

A set of PLOGs containing a larger dataset, and which also utilises the LOAD function, can be downloaded from the location. This contains the following number of change records, across the tables included in the replication:

```
SOE.CUSTOMERS:             Mine:112082
SOE.ADDRESSES:             Mine:112293
SOE.CARD_DETAILS:          Mine:112143
SOE.WAREHOUSES:            Mine:1000
SOE.ORDER_ITEMS:           Mine:768791
SOE.ORDERS:                Mine:300376
SOE.INVENTORIES:           Mine:902795
SOE.PRODUCT_INFORMATION:   Mine:1000
SOE.LOGON:                 Mine:820090
SOE.PRODUCT_DESCRIPTIONS: Mine:1000
SOE.ORDERENTRY_METADATA :  Mine:4
TX.META:                   Mine:3782
```

You can download the Dbvisit Replicate Connector QuickStart properties file (that you can also see on GitHub), which contains sensible starting configuration parameters, from this location.

Using these examples files as a starting point means that you do not have to setup and configure the Dbvisit Replicate application to produce a stream of PLOG files. So this will enable you to quickly get the Dbvisit Replicate Connector for Kafka up and running, in order to see it ingest Oracle change data to Kafka, and view this from there with consumers, or route to some other end target. Of course this limited change set means that you will not see new changes flowing through from an Oracle source once they have been processed - but it is a good place to begin in terms of understanding the connector functionality and operation.

To move beyond the Quickstart please refer to the Dbvisit Replicate online user guide for information and instructions on setting and configuring that application to produce and deliver the PLOG file stream.

We also recommend reviewing the Confluent Kafka Connect Quickstart guide which is an excellent reference in terms of understanding source/sink data flows and providing background context for Kafka Connect itself.

Once the Zookeeper, Kafka server and Schema Registry processes have been started, along with the Replicate Connector itself running in Kafka Connect in standalone mode, it will then ingest and process these PLOG files, writing the change data record messages to Kafka. These can be viewed on the other side with the default Avro consumer provided with the Kafka Connect framework.

1.2.1 Steps

1. Download the Confluent Platform

```
The only requirement is Oracle Java >= 1.7. Java installation
#Download the software from the Confluent website, version 3.x
#Install onto your test server: i.e: /usr/confluent
 unzip confluent-3.1.1-2.11.zip
```

2. Install the Replicate Connector JAR file

```
#Create the following directory
 mkdir $CONFLUENT_HOME/share/java/kafka-connect-dbvisitreplicate
#Build the Replicate Connector JAR file from the Github Repo (or download as per instructions above)
#Install the JAR file to the location just created above
```

3. Install the Replicate Connector "Quickstart" properties file

```
#Create the following directory
 mkdir $CONFLUENT_HOME/etc/kafka-connect-dbvisitreplicate
#Install the Quickstart properties file (download link above) to the location just created
```

4. Work with the example PLOG files

```
#Create a directory to hold the example PLOG files, e.g:
 mkdir /usr/dbvisit/replicate/demo/mine
#Upload and unzip the example PLOG files (download links for small and large datasets provided above,
#Edit the plog.location.uri parameter in the Quickstart dbvisit-replicate.properties example configu
 plog.location.uri=file:/usr/dbvisit/replicate/demo/mine
```

5. Start the Zookeeper, Kafka and Schema Registry processes

```
#Start Zookeeper
 $CONFLUENT_HOME/bin/zookeeper-server-start -daemon $CONFLUENT_HOME/etc/kafka/zookeeper.properties
#Start Kafka
 $CONFLUENT_HOME/bin/kafka-server-start -daemon $CONFLUENT_HOME/etc/kafka/server.properties
#Start the Schema Registry
 $CONFLUENT_HOME/bin/schema-registry-start -daemon $CONFLUENT_HOME/etc/schema-registry/schema-regist
#Start the REST Proxy (optional)
 $CONFLUENT_HOME/bin/kafka-rest-start -daemon $CONFLUENT_HOME/etc/kafka-rest/kafka-rest.properties
```

NB: this default configuration is run on a single server with local zookeeper, schema registry and REST Proxy services.

As an alternative, for ease of use, these commands can be wrapped in a script and then invoked to start the processes. Name and save this script to a location of your choice), being sure to set the CONFLUENT_HOME correctly within it:

```
#! /bin/bash

echo $(hostname)
CONFLUENT_HOME=/usr/confluent/confluent-3.1.1

echo "INFO Starting Zookeeper"
$CONFLUENT_HOME/bin/zookeeper-server-start -daemon $CONFLUENT_HOME/etc/kafka/zookeeper.properties
sleep 10

echo "INFO Starting Kafka Server"
$CONFLUENT_HOME/bin/kafka-server-start -daemon $CONFLUENT_HOME/etc/kafka/server.properties
sleep 10
```

```
echo "INFO Starting Schema Registry"
$CONFLUENT_HOME/bin/schema-registry-start -daemon $CONFLUENT_HOME/etc/schema-registry/schema-registry
#sleep 10

echo "INFO Starting REST Proxy"
$CONFLUENT_HOME/bin/kafka-rest-start -daemon $CONFLUENT_HOME/etc/kafka-rest/kafka-rest.properties
sleep 10
```

And run this as follows:

```
./kafka-init.sh
```

6. Run Kafka Connect, and the Replicate Connector

To run the Replicate Connector in Kafka Connect standalone mode open another terminal window to your test server and execute the following from your CONFLUENT_HOME location:

```
./bin/connect-standalone ./etc/schema-registry/connect-avro-standalone.properties ./etc/kafka-connec
```

You should see the process start up, log some messages, and then locate and begin processing PLOG files. The change records will be extracted and written in batches, sending the results through to Kafka.

7. View the messages in Kafka with the default Consumer utilities

Default Kafka consumers (clients for consuming messages from Kafka) are provided by the Confluent Platform for both Avro and Json encoding, and they can be invoked as follows:

```
./bin/kafka-avro-console-consumer --new-consumer --bootstrap-server localhost:9092 --topic REP-SOE.C

{"XID":"0000.68d6.00000002","TYPE":"INSERT","CHANGE_ID":1021010014941,"CUSTOMER_ID":205158,"CUST_FIRS
```

This expected output shows the SOE.CUSTOMERS table column data in the JSON encoding of the Avro records. The JSON encoding of Avro encodes the strings in the format `{"type": value}`, and a column of type `STRING` can be `NULL`. So each row is represented as an Avro record and each column is a field in the record. Included also are the Transaction ID (XID) that the change to this particular record occurred in, the TYPE of DML change made (insert, delete or update), and the specific CHANGE_ID as recorded for this in Dbvisit Replicate.

NOTE: to use JSON encoding and the JSON consumer please see our notes on JsonConverter settings later in this guide.

If there are more PLOGS to process you should see changes come through the consumers in real-time, and the following "Processing PLOG" messages in the Replicate Connector log file output:

```
[2016-12-03 09:28:13,557] INFO Processing PLOG: 1695.plog.1480706183 (com.dbvisit.replicate.kafkaconn
[2016-12-03 09:28:17,517] INFO Reflections took 22059 ms to scan 265 urls, producing 14763 keys and 1
[2016-12-03 09:29:04,836] INFO Finished WorkerSourceTask{id=dbvisit-replicate-0} commitOffsets succes
[2016-12-03 09:29:04,838] INFO Finished WorkerSourceTask{id=dbvisit-replicate-1} commitOffsets succes
[2016-12-03 09:29:04,839] INFO Finished WorkerSourceTask{id=dbvisit-replicate-2} commitOffsets succes
[2016-12-03 09:29:04,840] INFO Finished WorkerSourceTask{id=dbvisit-replicate-3} commitOffsets succes
```

Ctrl-C to stop the consumer processing further, and which will then show a count of how many records (messages) the consumer has processed:

```
^CProcessed a total of 156 messages
```

You can then start another consumer session as follows (or alternatively use a new console window), to see the changes delivered to the REP-TX.META topic, which contains the meta-data about all the changes made on the source.

```
./bin/kafka-avro-console-consumer --new-consumer --bootstrap-server localhost:9092 --topic REP-TX.ME

{"XID":"0000.68d9.00000000","START_SCN":24893566,"END_SCN":24893566,"START_TIME":1479626569000,"END_
```

In this output we can see details relating to specific transactions (XID) including the total CHANGE_COUNT made within this to tables we are interested in, and these are then cataloged for convenience in SCHEMA_CHANGE_COUNT_ARRAY.

1.3 Features

Dbvisit Replicate Connector supports the streaming of Oracle database change data with a variety of Oracle data types, varying batch sizes and polling intervals, the dynamic addition/removal of tables from a Dbvisit Replicate configuration, and other settings.

When beginning with this connector the majority of the default settings will be more than adequate to start with, although `plog.location.uri`, which is where PLOG files will be read from, will need to be set according to your system and the specific location for these files.

All the features of Kafka Connect, including offset management and fault tolerance, work with the Replicate Connector. You can restart and kill the processes and they will pick up where they left off, copying only new data.

1.3.1 Change Row publishing

Dbvisit Replicate Connector will attempt to assemble a complete view of the row record, based on the information made available in a PLOG, once the change has been made and committed on the source. This is done by merging the various components of the change into one complete record that conforms to an Avro schema definition, which itself is a verbatim copy of the Oracle source table definition.

This type of change record is useful when the latest version of the data is all that's needed, irrespective of the change vector. However with state-full stream processing the change vectors are implicit and can be easily extracted.

To illustrate we create a simple table on the Oracle source database, as follows, and perform and insert, update and delete:

```
create table SOE.TEST2 (
user_id number (6,0),
user_name varchar2(100),
user_role varchar2(100));
```

The default Kafka Connect JSON consumer can be invoked as follows (see the notes below on JSON encoding). Note that using the default Avro encoding with the supplied Avro consumers produces output that does not include the JSON schema information, and effectively begins from XID as follows:

```
[oracle@dbvrep01 confluent-3.1.1]$ ./bin/kafka-console-consumer --new-consumer --bootstrap-server loc
```

Inserts

insert into SOE.TEST2 values (1, 'Matt Roberts', 'Clerk');

commit;

```
{"schema":{"type":"struct","fields":[{"type":"string","optional":false,"field":"XID"},{"type":"string
```

Updates

update SOE.TEST2 set user_role = 'Senior Partner' where user_id=1;

commit;

{"schema":{"type":"struct","fields":[{"type":"string","optional":false,"field":"XID"},{"type":"string

Note that a complete row is represented as a message delivered to Kafka. This is obtained by merging the existing and changed values to produce the current view of the record as it stands.

Deletes

delete from SOE.TEST2 where user_id=1;

commit;

{"schema":{"type":"struct","fields":[{"type":"string","optional":false,"field":"XID"},{"type":"string

Note that the detail for a delete shows the row values as they were at the time this operation was performed.

1.3.2 Topic Per Table

Data from each replicated table is published to its own topic, eg. all change row records for a replicated table will be published as Kafka messages in a single partition in a topic.

1.3.3 Topic Auto-creation

The automatic creation of topics is governed by the Kafka parameter `auto.create.topics.enable`, which is TRUE by default. This means that, as far as the Dbvisit Replicate Connector goes, any new tables detected in the PLOG files it processes will have new topics automatically generated for them – and change messages written to them without any additional intervention.

1.3.4 Metadata Topic

Dbvisit Replicate Connector for Kafka automatically creates and writes a meta-data topic which lists out the Transactions (TX), and an ordered list of the changes contained with these. This can be utilized/cross-referenced within consumers or applications to reconstruct change ordering across different tables, and manifested in different topics. This is a means of obtaining an authoratitive "global" view of the change order, as they occurred on the Oracle database, as may be important in specific scenarios and implementations.

So the output of a TX meta data record is as follows:

{"XID":"0002.019.00008295","START_SCN":24914841,"END_SCN":24914850,"START_TIME":1479630708000,"END_TI

Explanation:

1. **XID**: Transaction ID from the Oracle RDBMS
2. **START_SCN**: SCN of first change in transaction
3. **END_SCN**: SCN of last change in transaction
4. **START_TIME**: Time when transaction started
5. **END_TIME**: Time when transaction ended
6. **START_CHANGE_ID**: ID of first change record in transaction
7. **END_CHANGE_ID**: ID of last change record in transaction
8. **CHANGE_COUNT**: Number of data change records in transaction, not all changes are row level changes

9. **SCHEMA_CHANGE_COUNT_ARRAY: Number of data change records for each replicated table in the transaction, as a**

    (a) **SCHEMA_NAME**: Replicated table name (referred to as schema name because each table has their own Avro schema definition)

    (b) **CHANGE_COUNT**: Number of data records changed for table

Corresponding to this, each data message in all replicated table topics contain three additional fields in their payload, for example:

```
{"XID":"0003.007.00008168","TYPE":"INSERT","CHANGE_ID":1064010025000,"USER_ID":{"bytes":"\u0000"},"US
```

This allows linking it to the transaction meta data topic which holds the following transaction information aggregated from individual changes:

1. **XID**: Transaction ID - its parent transaction identifier
2. **TYPE**: the type of action that resulted in this change row record, eg. INSERT, UPDATE or DELETE
3. **CHANGE_ID**: its unique change ID in the replication

1.3.5 Data types

Information on the data types supported by Dbvisit Replicate, and so what can be delivered through to PLOG files for processing by the Replicate Connector for Kafka, can be found here.

Information on Dbvisit Replicate Connector for Kafka specific data type mappings and support can be found in the Configuration section of this documentation.

1.3.6 LOB Handling

Single part (inline - LOB size < 4000 bytes) LOBS are supported, but only partial updates for larger/out-of-line LOBs.

1.3.7 LOAD

Dbvisit Replicate's Load function can be used to instantiate or baseline all existing data in the Oracle database tables by generating special LOAD PLOG files, which can be processed by the **Dbvisit Replicate Connector for Kafka**. This function ensures that before any change data messages are delivered the application will write out all the current table data – effectively initializing or instantiating these data sets within the Kafka topics.

Regular and LOAD PLOGS on the file system.

```
1682.plog.1480557139
1671.plog.1480555838-000001-LOAD_26839-SOE.ADDRESSES-APPLY
1671.plog.1480555838-000010-LOAD_26845-SOE.PRODUCT_INFORMATION-APPLY
1680.plog.1480556667
1676.plog.1480556539
1677.plog.1480556547
1674.plog.1480555972
1672.plog.1480555970
1671.plog.1480555838-000008-LOAD_26842-SOE.ORDER_ITEMS-APPLY
1671.plog.1480555838-000006-LOAD_26848-SOE.ORDERENTRY_METADATA-APPLY
1671.plog.1480555838-000004-LOAD_26844-SOE.INVENTORIES-APPLY
1671.plog.1480555838-000009-LOAD_26847-SOE.PRODUCT_DESCRIPTIONS-APPLY
1671.plog.1480555838-000007-LOAD_26843-SOE.ORDERS-APPLY
1671.plog.1480555838-000013-LOAD_26841-SOE.WAREHOUSES-APPLY
```

```
1671.plog.1480555838
1678.plog.1480556557
1671.plog.1480555838-000002-LOAD_26840-SOE.CARD_DETAILS-APPLY
1671.plog.1480555838-000012-LOAD_42165-SOE.TEST2-APPLY
1673.plog.1480555971
1681.plog.1480556671
1679.plog.1480556659
1671.plog.1480555838-000011-LOAD_39367-SOE.TEST1-APPLY
1671.plog.1480555838-000003-LOAD_26838-SOE.CUSTOMERS-APPLY
```

The parameter `plog.global.scn.cold.start` can be invoked to specify a particular SCN that the connector should work from, before the LOAD operation was run to generate the LOAD plogs, to provide some known guarantees around the state of the tables on the Oracle source at this time.

Note: the system change number or SCN, is a stamp that defines a committed version of a database at a point in time. Oracle assigns every committed transaction a unique SCN.

1.3.8 Replicate Stream Limitation

Each replicated table publishes their data to their own topic in Kafka, identified by the fully qualified name (including user schema owner) of the replicated table. If more than one Dbvisit Replicate Connector process is mining the same REDO LOGs the PLOG sequences may overlap and the Kafka topics must be separated by adding a unique namespace identifier to the topic names in Kafka.

See the `topic.prefix` parameter, which has the default of "REP-".

1.3.9 DDL Support

At this point in time only there is only limited support for DDL.

New tables may be added to a replication, if enabled on the Dbvisit Replicate side, then these will automatically be detected by the Replicate Connector for Kafka, and written to a new topic. However, table/column renames, truncate and drop table statements are ignored, and will not impact on the existing associated Kafka topic.

The adding and removing of table columns is supported by default. Those records which existed prior to the addition of a new column will have default (empty) value assigned during their next operation, as in named EXTRA column in the following:

```
{"XID":"0004.012.00006500","TYPE":"UPDATE","CHANGE_ID":1076010000726,"USER_ID":{"bytes":"\u0000"},"US
```

Conversely any columns which are dropped will have null values assigned, as in the following, for any previously values which existed in the record set:

```
{"XID":"0005.013.0000826f","TYPE":"UPDATE","CHANGE_ID":1078010000324,"USER_ID":{"bytes":"\u0000"},"US
```

1.3.10 JSON

To use JSON encoding, rather than the default Avro option, use the JSON Converter options supplied as part of the Kafka Connect framework, by setting them as follows in the $CONFLUENT_HOME/etc/schema-registry/connect-avro-standalone.properties or the $CONFLUENT_HOME /etc/schema-registry/connect-avro-distributed.properties parameter files:

```
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
```

The non-Avro consumer can be invoked as follows, and will then display output as follows (here for the SOE.TEST1 table):

```
./bin/kafka-console-consumer --new-consumer --bootstrap-server localhost:9092 --topic REP-SOE.TEST1
```

```
{"schema":{"type":"struct","fields":[{"type":"string","optional":false,"field":"XID"},{"type":"string
```

1.3.11 Delivery Semantics

The Replicate Connector for Kafka manages offsets committed by encoding then storing and retrieving them (see the log file extract below). This is done in order that the connector can start from the last committed offsets in case of failures and task restarts. The replicate offset object is serialized as JSON and stored as a String schema in Kafka offset storage. This method should ensure that, under normal circumstances, records delivered from Oracle are only written once to Kafka.

```
[2016-11-17 11:41:31,757] INFO Offset JSON - TX.META:{"plogUID":4030157521414,"plogOffset":2870088}
```

1.4 Schema Evolution

The Replicate Connector supports schema evolution when the Avro converter is used. When there is a change in a database table schema, the Replicate Connector can detect the change, create a new Kafka Connect schema and try to register a new Avro schema in the Schema Registry. Whether it is able to successfully register the schema or not depends on the compatibility level of the Schema Registry, which is backward by default.

For example, if you add or remove a column from a table, these changes are backward compatible by default (as mentioned above) and the corresponding Avro schema can be successfully registered in the Schema Registry.

You can change the compatibility level of Schema Registry to allow incompatible schemas or other compatibility levels by setting `avro.compatibility.level` in Schema Registry. Note that this is a global setting that applies to all schemas in the Schema Registry.

1.5 Deployment Guidelines

1.5.1 Upgrading

To upgrade to a newer version of the Dbvisit Replicate Connector for Kafka simply stop this process running in Kafka Connect, and replace the associated JAR file in the following location:

```
$CONFLUENT_HOME/share/java/kafka-connect-dbvisitreplicate
```

1.6 Troubleshooting

1.6.1 Logging

To alter logging levels for the connector all you need to do is update the log4j.properties file used by the invocation of the Kafka Connect worker. You can either edit the default file directly (see bin/connect-distributed and bin/connect-standalone) or set the env variable KAFKA_LOG4J_OPTS before invoking those scripts (exact syntax is ‘ export KAFKA_LOG4J_OPTS=”-Dlog4j.configuration=file:${CFG_DIR}/dbvisitreplicate-log4j.properties” ‘)

In the following example, the settings were set to DEBUG to increase the log level for this connector class (and other options are ERROR, WARNING and INFO):

```
log4j.rootLogger=INFO, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%d] %p %m (%c:%L)%n
log4j.logger.org.apache.zookeeper=WARN
log4j.logger.org.I0Itec.zkclient=WARN
log4j.logger.dbvisit.replicate.kafkaconnect.ReplicateSourceTask=DEBUG
log4j.logger.dbvisit.replicate.kafkaconnect.ReplicateSourceConnector=DEBUG
```

CHAPTER 2

Configuration Options

2.1 Configuration Parameters

2.1.1 Key User Configuration Properties

Note: **plog.location.uri** - is the location from which Dbvisit Replicate Connector will search for and read PLOG files delivered by the Dbvisit Replicate application. This directory must be local to the filesystem on which Kafka Connect is running, or accessible to it via a mounted filesystem. It cannot be an otherwise remote filesystem.

Note: **plog.data.flush.size** - for low transactional volumes (and for testing) it is best to change the default value of `plog.data.flush.size` in the configuration file to a value less than your total number of change records, eg. for manual testing you can use 1 to verify that each and every record is emitted correctly. This configuration parameter is used as the internal PLOG reader's cache size which translates at run time to the size of the polled batch of Kafka messages. It is more efficient under high transactional load to publish to Kafka (particularly in distributed mode where network latency might be an issue) in batches. Please note that in this approach the data from the PLOG reader is only emitted once the cache is full (for the specific Kafka source task) and/or the PLOG is done, so an Oracle redo log switch has occurred. This means that if the `plog.data.flush.size` is greater than total number of LCRs in cache it will wait for more data to arrive or a log switch to occur.

2.1.2 All User Configuration Properties

`tasks.max` Maximum number of tasks to start for processing PLOGs.

- Type: string
- Default: 4

`topic.prefix` Prefix for all topic names.

- Type: string
- Default: REP-

`plog.location.uri` Replicate PLOG location URI, output of Replicate MINE.

- Type: string
- Default: file:/home/oracle/ktest/mine

plog.data.flush.size LCRs to cache before flushing, for connector this is the batch size, choose this value according to transactional volume, for high throughput to kafka the default value may suffic, for low or sporadic volume lower this value, eg. for testing use 1 which will not use cache and emit every record immediately.

- Type: string
- Default: 1000

plog.interval.time.ms Time in milliseconds for one wait interval, used by scans and health check.

- Type: string
- Default: 500

plog.scan.interval.count Number of intervals between scans, eg. 5 x 0.5s = 2.5s scan wait time.

- Type: string
- Default: 5

plog.health.check.interval Number of intervals between health checks, these are used when initially waiting for MINE to produce PLOGs, eg. 10 * 0.5s = 5.0s.

- Type: string
- Default: 10

plog.scan.offline.interval Default number of health check scans to decide whether or not replicate is offline, this is used as time out value. NOTE for testing use 1, i.e. quit after first health check 1 * 10 * 0.5s = 5s where 10 is plog.health.check.interval value and 0.5s is plog.interval.time.ms value.

- Type: string
- Default: 1000

topic.name.transaction.info Topic name for transaction meta data stream.

- Type: string
- Default: TX.META

plog.global.scn.cold.start Global SCN when to start loading data during cold start.

- Type: string
- Default: 0

2.2 Data Types

Oracle Data Type	Connect Data Type	Default Value	Conversion Rule
NUMBER	Int32	-1	scale <= 0 and precision - scale < 10
NUMBER	Int64	-1L	scale <= 0 and precision - scale > 10 and < 20
NUMBER	Decimal	BigDecimal.ZERO	scale > 0 or precision - scale > 20
CHAR	Type.String	Empty string (zero length)	Encoded as UTF8 string
VARCHAR	“”	“”	“”
VARCHAR2	“”	“”	“”
LONG	“”	“”	“”
NCHAR	Type.String	Empty string (zero length)	Encoded as UTF8, attempt is made to auto-detect if national character set was UTF-16
NVARCHAR	“”	“”	“”
NVARCHAR2	“”	“”	“”
INTERVAL DAY TO SECOND	Type.String	Empty string (zero length)	
INTERVAL YEAR TO MONTH	“”	“”	
CLOB	Type.String	Empty string (zero length)	UTF8 string
NCLOB	“”	“”	“”
DATE	Timestamp	Epoch time	
TIMESTAMP	“”	“”	
TIMESTAMP WITH TIME ZONE	“”	“”	
TIMESTAMP WITH LOCAL TIME ZONE	“”	“”	
BLOB	Bytes	Empty byte array (zero length)	Converted from SerialBlob to bytes
RAW	Bytes	Empty byte array (zero length)	No conversion
LONG RAW	“”	“”	“”

2.3 Distributed Mode Settings

Use the following to start Dbvisit Replicate Connector for Kafka in Distributed mode, once the Kafka Connect worker has been started on the host node. Postman is an excellent utility for working with cUrl commands.

```
   curl -v -H "Content-Type: application/json" -X PUT 'http://localhost:8083/connectors/kafka-connect
'{
  "connector.class": "com.dbvisit.replicate.kafkaconnect.ReplicateSourceConnector",
  "tasks.max": "2",
  "topic.prefix": "REP-",
  "plog.location.uri": "file:/foo/bar",
  "plog.data.flush.size": "1",
  "plog.interval.time.ms": "500",
  "plog.scan.interval.count": "5",
  "plog.health.check.interval": "10",
  "plog.scan.offline.interval": "1000",
```

```
  "topic.name.transaction.info": "TX.META"
}'
```

Or save this to a file <json_file>:

```
{
  "name": "TSource",
  "config": {
    "connector.class": "com.dbvisit.replicate.kafkaconnect.ReplicateSourceConnector",
  "tasks.max": "2",
  "topic.prefix": "REP-",
  "plog.location.uri": "file:/foo/bar",
  "plog.data.flush.size": "1",
  "plog.interval.time.ms": "500",
  "plog.scan.interval.count": "5",
  "plog.health.check.interval": "10",
  "plog.scan.offline.interval": "1000",
  "topic.name.transaction.info": "TX.META"
  }
}

 curl -X POST -H "Content-Type: application/json" http://localhost:8083 --data "@<json_file>"
```

CHAPTER 3

Changelog

3.1 Replicate Source Connector

- DATE () - Early Release