Connect CDC SQData

Kafka Quickstart

Version 4.0
# Kafka Quickstart

## Contents

Kafka Quickstart .......................................................................................................................... 4  
Quick Start Approach .................................................................................................................. 7  
Determine Kafka Topic Data Requirements ................................................................................. 8  
Prepare Source Capture Environment ......................................................................................... 9  
Prepare Target Apply Environment .............................................................................................. 10  
  Install Base Product .................................................................................................................... 10  
  Create Variable Directories ......................................................................................................... 10  
  Create Application Directory Structure ....................................................................................... 11  
  Resolve External Library Requirements ....................................................................................... 12  
  Set External Library Path ............................................................................................................ 13  
Configure Engine Controller Daemon ............................................................................................. 14  
Determine Kafka Payload Format .................................................................................................. 16  
Create Kafka Apply Engine Script ................................................................................................ 18  
  Add Source Data Structures ....................................................................................................... 18  
  Determine Kafka Datastore Specifications ..................................................................................... 20  
  Construct the Kafka Apply Engine Script .................................................................................... 24  
  Parse Engine ............................................................................................................................... 28  
  Run Apply Engine ......................................................................................................................... 29  
Create Kafka Replicator Engine Script ......................................................................................... 31  
  Identify Source Data Structures .................................................................................................... 31  
  Determine Kafka URL ..................................................................................................................... 32  
  Construct the Kafka Replicator Engine Script ............................................................................ 32  
  Run Kafka Replicator Engine ....................................................................................................... 33  
Configuration Complete .................................................................................................................. 35  
Kafka Operational Issues ............................................................................................................... 36  
  Engine Messages .......................................................................................................................... 36  
  Kcat Utility ................................................................................................................................. 36  
  Kafka Contingencies ...................................................................................................................... 37
Kafka Quickstart

Kafka is a robust clustered distributed streaming platform used to build real-time streaming data pipelines that reliably move data between systems or applications using a publish and subscribe architecture.

Customers choose it because they want to do more than simply replicate captured source data into a, dare we say legacy, relational datastore. Some use the stream of kafka messages as the mechanism to identify "events" that trigger subsequent downstream business processes. Others use it to populate a "big data" repository where other tools will be used for analytics or to answer questions that may not even be known at the start of a project.

This Quickstart illustrates two methods provided by Connect CDC SQData to replicate changes in virtually any source datastore into Kafka. The first utilizes our time tested high performance Capture agents and Apply Engine technology while the second introduces a new capability that eliminates maintenance of the Apply Engine when table structures are changed in Relational source datastores, beginning with Db2 z/OS.

Both methods simplify the creation, configuration and execution of the replication process by utilizing an Engine that Applies to Kafka Topics. Connect CDC SQData treats Kafka as a simple Target Datastore for information captured by any of Connect CDC SQData’s capture agents. Precisely's Connect CDC SQData product architecture provides for the direct point-to-point (Source to Target) transfer of captured data without the use of any sort of staging area. When properly configured, captured data does not require or even land on any intermediate storage device before being loaded into the target Kafka Topic.

While Connect CDC SQData can be used to implement a solution that customizes the data written to Kafka, we and the industry don't recommend it. There are several reasons but the most obvious are the ongoing maintenance requirements. Streaming data to Kafka is fundamentally different from replicating data, for example from mainframe Db2 to Oracle on Linux.

In a Db2 to Oracle use case, it is understood that both source and target table schemas will change and they may never be identical or even share identical column names. DBA maintenance of the Db2 schemas will be scheduled to minimize disruption of source applications and the downstream Oracle applications will have to decide if and how to absorb those changes. Oracle DBA’s will have to coordinate changes to their schemas with the Db2 DBA’s and the Oracle application developers. Consequently the Connect CDC SQData Apply Engine in the middle will need to have it's source Descriptions updated and possibly the Oracle target table Descriptions updated as well. Changes to mapping Procedures may also have to be changed. Solid configuration management procedures are required to fully implement these changes.

Implementing a Kafka architecture changes all that. Kafka data consumers read the JSON formatted data in a Kafka message that also contains the schemas describing that data. The biggest issue with this technique however are the JSON schemas included in the payload of every single Kafka message produced and consumed. That problem is solved by the AVRO data serialization system which separates the data from the schema. Data is read by the consumer using the schema that describes the data at the time it was written by a producer. This of course assumes that the tools and languages used by the producer and consumer are AVRO "aware". With Connect CDC SQData Version 4, we have embraced Confluent's Schema Registry for managing the schema versions of AVRO formatted Kafka messages through the automatic registration of Kafka topic schemas.
Kafka Quickstart

Apply engines that utilize the REPLICATE function, for DB2 as well as IMS and VSAM source data will still require manual intervention to replace the source DESCRIPTION parts that correspond to altered schemas or changed "copybooks". Once that has been accomplished however the Apply Engine need only beParsed and Started and the registration of the updated AVRO schemas will be performed automatically. Even Version 4 Apply Engines that have "customized" target DESCRIPTIONS and mapping PROCEDURES will benefit because the Target DESCRIPTIONS used to create the AVRO schemas will be automatically validated and registered, if necessary, when the Engine is Started.
Finally, Connect CDC SQData Version 4 also introduces our revolutionary Replicator Engine for relational source databases beginning with Db2 12 z/OS. The Replicator Engine fully automates the propagation of source schema changes and Kafka message production using AVRO and the Confluent Schema Registry. The Replicator also supports parallel processing of the replication workload through multiple Producer threads with the number of threads or workers specified at run-time. This means that Connect CDC SQData becomes a utility function within the enterprise architecture, reacting to Relational schema changes without interruption and without maintenance of the Connect CDC SQData Kafka producer configuration running in your Linux environment.

Notes:

1. Kafka is only supported on the Linux OS platform. Connect CDC SQData’s distributed architecture however enables data captured on virtually any platform and from most database management systems to become a Kafka topic.

2. Messages are written to Kafka asynchronously which means the Apply and Replicator Engines do not wait for Kafka to acknowledge receipt before writing another message. A delay in acknowledgment by Kafka may be due to replication occurring within Kafka itself (particularly with Kafka ACK mode 2). The Replicator does ensure that Units-of-Work (UOWs) are acknowledged to the Publisher in sequence preventing loss of captured data.

3. The Connect CDC SQData warning message SQDUR10I Max In-flight UOWS reached: <n> indicates a potential Kafka problem (slowdown or outage). The default maximum number of in-flight UOW’s is 100 but can be controlled by the --max-uow=<nnn> parameter.

To learn more, Connect CDC SQData recommends both Apache’s AVRO documentation and Confluent’s Schema Registry Tutorial available on web browsers everywhere.
Kafka Quickstart

Quick Start Approach

The Quickstart approach is a step by step guide to the installation, configuration, testing and operation of the Connect CDC SQData Capture and both the Apply and Replicator Engine components that will create Kafka Topics:

1. Determine your initial Kafka Topic data requirements
2. Preparation of the Source Data Capture and Target Apply Engine environments
3. Configure Engine Controller Daemon
4. Determine how you will control the structure of the target Kafka Topics
5. Create either Kafka Topic no-map replication Apply Engine script or a Replicator Engine Configuration script

Once these steps have been completed you will then be able to run an end to end test of each of the components in standalone mode. This allows you to work out any security or environmental issues before running alongside other Engines in a shared Capture/Publisher configuration.

After all components are working properly and your first Kafka Topic has been populated successfully, you are ready to add more source/target interfaces and Kafka topics to your configuration.

This Quick Start is intended to supplement, not replace, other documents including the various Data Capture and the Apply and Replicator Engine Reference documentation. We recommend you familiarize yourself with the Precisely MySupport portal where you can learn more about Connect CDC SQData's overall Architecture and approach to Change Data Capture. The answer to many of the questions that inevitably arise during initial installation, configuration and testing will be found in those documents.

Processing data placed into Kafka is beyond the scope of this Quick Start as are the myriad of tools that can be used to "consume" that data. In one example a customer wanted to use Elastic Search for analytics using "LogStash" to process the source data. Their source data was however mainframe application databases that don't naturally produce a usable source of data. Precisely recommended kafka as an intermediate datastore because of its performance, fault tolerance and because it places data into the open systems platform environment where Elastic Search operates.

There are many tools available to facilitate the kafka / ES interface, some are proprietary and others are open source, including but certainly not limited to:

https://github.com/confluentinc/kafka-connect-elasticsearch/blob/master/docs/elasticsearch_connector.rst
Kafka Quickstart

Determine Kafka Topic Data Requirements

While your Kafka Topic set may eventually contain lots of different types of data, Precisely recommends that you start with only a few sources. That usually means a small number of Relational database tables, a subset of segments in a legacy IMS database or a few VSAM file Copybooks. Since your data may eventually come from different platforms, pick just one to get started.

When implementing Connect CDC SQData's change data capture to collect the data sent to Kafka, it's easy to forget that downstream Kafka Consumers may need access to data that hasn't changed in some time and therefore has never been published as a Kafka topic. There are several methods for performing an "Initial Load" and they vary depending on the original source of data, be it hosted on the Mainframe, Linux or Windows. See the Initial Load sections of the applicable Change Data Capture reference documentation for more details. Precisely also recommends special consideration be given to the Topic ID's and Metadata associated with this historical data.
Kafka Quickstart

Prepare Source Capture Environment

Kafka is a Target only datastore that may have one or more Source datastores. This Quickstart assumes that you have already determined where your Kafka data will come from and that you have installed and configured Connect CDC SQData's Change Data Capture (CDC) component on the source platform.

The first step in this Quickstart is to ensure that the Source Data Capture installation and configuration tasks have been completed. Each of these steps are described in detail in the Connect CDC SQData Installation guide, Operating Systems specific $Start_Here_<operating_systems>.pdf and the source datastore "type" specific Capture reference documents.

<table>
<thead>
<tr>
<th>#</th>
<th>Task</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Confirm installation of required Capture components</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>Configure the Source data Capture/Publisher</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>Update Daemon registered agents (sqdagents.cfg) if necessary</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Capture Environment Preparation Complete</td>
<td></td>
</tr>
</tbody>
</table>

Note: The Replicator Engine introduced with Version 4 requires Connect CDC SQData's V4 Db2 z/OS Data Capture agent and supports only IBM's Db2 version 12 for z/OS. Prior to release 12 of Db2 z/OS it was impossible to ensure the availability of schema information describing the "before Image" in the Db2 Log.
**Prepare Target Apply Environment**

This section of this Quickstart focuses on preparation of the Linux environment where the Engines will operate.

<table>
<thead>
<tr>
<th>#</th>
<th>Task</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Download and install the Connect CDC SQData Product Files</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>Create the Connect CDC SQData Variables directory structure</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>Create the Connect CDC SQData Application directory structure</td>
<td>N/A</td>
</tr>
<tr>
<td>4</td>
<td>Resolve External Library Requirements</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>Set External Library Path</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Engine Environment Preparation Complete**

**Install Base Product**

This Quickstart describes the configuration and operation of both the Kafka Apply and Replicator Engines. Before beginning those tasks however, the base Connect CDC SQData product must be installed. Refer to the Connect CDC SQData Installation Guide for an overview of the product, installation instructions and prerequisites.

Follow the instructions in the $Start_Here_UNIX.pdf for the Linux specific installation instructions.

**Create Variable Directories**

Once Linux, UNIX and Windows source and target systems and datastores have been identified, the configuration of the Capture Agents, Apply Engines and their Controller Daemon's can begin. That will require the creation of directories and files for variable portions of the configuration. At this point we assume the base Connect CDC SQData product has already been installed according to the instructions in the Installation Guide and the Operating Systems specific $Start_Here_<operating_systems>.pdf. The recommended location and Environment Variable values for this static data were:

```
/opt/sqdata or
/home/<sqdata_user>/sqdata
```

If an Environment Variable will be used to reference the installation location, the recommended value is:

```
<SQDATA_DIR>
```

Controller Daemons, Capture Agents and Engines require the creation of directories and files for variable portions of their configurations. Just as the location of the base product installation can be modified, the location of variable directories can be adjusted conform to the operating system and to accommodate areas of responsibility, including the associated "application" and "environments" such as Test and Production. This document will refer to the location most commonly used on Linux, AIX and Windows:

```
/var/opt/sqdata[/<application>[/<environment>]] or
/home/<sqdata_user>[/<application>[/<environment>]] or simply
/home/sqdata[/<application>[/<environment>]]
```

If an Environment Variable will be used to reference the location of variable portions of the configuration, the recommended value is:

```
<SQDATA_VAR_DIR>
```

While only the base variable directory is required and the location of the daemon directory is optional, we recommend the structure described below:
**Kafka Quickstart**

<SQDATA_VAR_DIR>/daemon - The working directory used by the Daemon that also contains two sub directories.

<SQDATA_VAR_DIR>/daemon/cfg - A configuration directory that contains two configuration files.

<SQDATA_VAR_DIR>/daemon/logs - A logs directory, though not required, is suggested to store log files used by the controller daemon. Its suggested location below must match the file locations specified in the Global section of the sqdagents.cfg file created in the section "Setup Controller Daemon" later in this document.

Additional directories should be created for each Capture agent running on the system. Precisely recommend the structures described below:

<SQDATA_VAR_DIR>/<type>/cdc - The working directory of each capture agent where type might be ORA (Oracle), UDB (Db2/LUW)

<SQDATA_VAR_DIR>/<type>/cdc/data - A data directory is also required by each Capture agents. Files will be allocated in this directory as needed by the CDCStore Storage Agent when transient data exceeds allocated in-memory storage. The suggested location below must match the "data_path" specified in the Storage agent configuration (.cab file) described in the Capture References. A dedicated File System is required in production with this directory as the "mount point".

Example:

The following commands will create the directories described above:

```
$ mkdir -p <SQDATA_VAR_DIR>/daemon --mode=775
$ mkdir -p <SQDATA_VAR_DIR>/daemon/cfg --mode=775
$ mkdir -p <SQDATA_VAR_DIR>/daemon/log --mode=775
$ mkdir -p <SQDATA_VAR_DIR>/<type>/cdc --mode=775
$ mkdir -p <SQDATA_VAR_DIR>/<type>/cdc/data --mode=775
```

**Note**, the User-ID(s) under which the Capture and Engine agents and the Controller Daemon will run must be authorized for Read/Write access to these directories.

### Create Application Directory Structure

Connect CDC SQData’s Apply and Replicator Engines share a number of operational components including both NaCl Keys and the Connect CDC SQData daemon. The Linux directory structure described below should be used for Apply Engines.

While the Connect CDC SQData Variable Directory <SQDATA_VAR_DIR> location works fine for Capture Agents and the Controller daemon, Apply Engine Script development also requires a structure accommodating similar items from dissimilar platforms such as DDL from Db2 and Oracle. For that reason the following directory nodes are recommended at the next level for script development and parts management:

<table>
<thead>
<tr>
<th>/&lt;directory_name&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENGINE</td>
<td>Main Engine scripts</td>
</tr>
<tr>
<td>CDCPROC</td>
<td>CDC Engine Called Procedures referenced by #INCLUDE</td>
</tr>
<tr>
<td>LOADPROC</td>
<td>Load (UnLoad) Engine Called Procedures referenced by #INCLUDE</td>
</tr>
<tr>
<td>DSDEF</td>
<td>Datastore Definition referenced by #INCLUDE</td>
</tr>
<tr>
<td>&lt;TYPE&gt;DDL</td>
<td>RDBMS specific DDL, eg DB2DDL, ORADDL, MSSQLDDL, etc</td>
</tr>
</tbody>
</table>
### Kafka Quickstart

<table>
<thead>
<tr>
<th><code>&lt;directory_name&gt;</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMSSEG</td>
<td>IMS Segment Copybooks</td>
</tr>
<tr>
<td>IMSDBD</td>
<td>IMS DBD</td>
</tr>
<tr>
<td><code>&lt;TYPE&gt;COB</code></td>
<td>System specific COBOL copybooks, eg: VSAMCOB, SEQCBB (sequential files)</td>
</tr>
<tr>
<td>XMLDTD</td>
<td>XML Document Type Definitions that will be used in a DESCRIPTION command</td>
</tr>
<tr>
<td><code>&lt;TYPE&gt;CSR</code></td>
<td>RDBMS specific Cursors, eg DB2CSR, ORACSR, etc</td>
</tr>
<tr>
<td><code>&lt;TYPE&gt;LOAD</code></td>
<td>RDBMS specific Load Control, eg DB2LOAD, ORALOAD, etc</td>
</tr>
</tbody>
</table>

**Notes:**

1. While it may be more convenient to use lower case directory names, if your environment includes the z/OS Platform, consideration should be given to reusability as some z/OS references must be in Upper Case.

2. Engine scripts are typically Platform specific in that they cannot be used on another type of Platform, eg z/OS and UNIX without at least minor modification.

3. Called Procedures can frequently be used with little or no changes on another platform, even when they contain platform specific Functions, unless they require direct access to a datastore on another platform, an atypical requirement.

4. Throughout the remainder of this document, part locations will usually refer only to the last node of standard z/OS Partitioned Datasets and UNIX or Windows directory hierarchy.

**Unzip the Basic/Linux_Parts_Structure.zip file to create the full Linux directory structure along with sample parts and shell scripts.**

Alternatively, commands similar to the following may be used to create the recommended directory structures.

```
$ mkdir -p <SQDATA_VAR_DIR>/DB2DDL --mode=775
$ mkdir -p <SQDATA_VAR_DIR>/ORADDL --mode=775
$ mkdir -p <SQDATA_VAR_DIR>/IMSDBD --mode=775
$ mkdir -p <SQDATA_VAR_DIR>/IMSSEG --mode=775
$ mkdir -p <SQDATA_VAR_DIR>/ENGINE --mode=775
$ mkdir -p <SQDATA_VAR_DIR>/CDCPROC --mode=775
```

The nature of Replicator Engine Configuration scripts do not require a complicated structure. Precisely recommends that you consider how your Source database and Applications are structured since it may be desirable to maintain Replicator Engine Configurations in a similar structure:

<table>
<thead>
<tr>
<th><code>&lt;directory_name&gt;</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENGINE</td>
<td>Replicator Engine Configuration Scripts</td>
</tr>
</tbody>
</table>

### Resolve External Library Requirements

**Librdkafka.so** provides the "C" language API for Producer, Consumer and Admin clients and is required by the Engines to connect to Kafka. The required version level of the Kafka external library is 0.80 or higher. This library can be installed using a distribution provided package if available or built from source which can be downloaded from github [https://github.com/edenhill/librdkafka](https://github.com/edenhill/librdkafka).

**Note,** Precisely recommends using the most current version of this open source library whenever possible. The popularity of both managed Kafka clusters and Kafka compatible message brokers like Azure EventHub present
tuning opportunities that the librdkafka developers are using to provide specialized defaults based on user experience.

**Kcat**, a command line tool from the developers of librdkafka that uses the same API should also be installed [https://github.com/edenhill/kcat](https://github.com/edenhill/kcat). It is a prerequisite and should be used to test and diagnose all installation specific configurations and connection issues. Once Kcat is working the same configuration parms can be used for Connect CDC SQData. Kcat can also be used to confirm topic content and Engine execution by acting as a local Kafka consumer, providing end to end validation of data replication.

**libcurl** is required to support communication with the Confluent Schema Registry by the Engines when using AVRO formatted Kafka topics. Libcurl can be installed using distribution provided package if available or built from source which can be downloaded from [https://curl.se/download.html](https://curl.se/download.html). You can read more about AVRO and the Confluent Platform at [https://www.confluent.io/download/](https://www.confluent.io/download/).

Other functionality beyond simply the Kafka interface may also be utilized:

- SSL - While disabled by default, SSL can be used for Encryption and Authentication by the Connect CDC SQData Kafka client. See [https://github.com/edenhill/librdkafka/wiki/Using-SSL-with-librdkafka](https://github.com/edenhill/librdkafka/wiki/Using-SSL-with-librdkafka) and for detailed configuration instructions, [http://kafka.apache.org/documentation.html#security_ssl](http://kafka.apache.org/documentation.html#security_ssl)

- A variety of Kafka dashboard tools, both open source and proprietary can be used to monitor the flow of data into Kafka.

**Set External Library Path**

If the Kafka external library is not already in the system library path, the environment variable `SQDATA_KAFKA_LIBRARY` must be set to point to the external library.
Configure Engine Controller Daemon

The Controller Daemon, SQDaemon plays a key role in the authentication process by being the first point of contact for any agent requesting communication with any other agent in both single and multi-platform environments. See the Secure Communications Guide for more details regarding the Controller Daemon's role in security. Controller Daemons are accessed via a TCP/IP interface to an assigned Port on the platform where they are running. Their symbolic name is often synonymous with a specific Host (platform or Environment on which they are running).

The primary difference between an Engine Controller Daemon and a Daemon on Capture platforms is that the Authorized Key File of the Engine Controller Daemon need only contain the Public keys of SQDaemon utility users on both the local and remote platforms.

Setup and configuration of the Engine Controller Daemon, SQDaemon, includes:

<table>
<thead>
<tr>
<th>#</th>
<th>Task</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Configure Engine Daemon</td>
<td></td>
</tr>
</tbody>
</table>
## Kafka Quickstart

<table>
<thead>
<tr>
<th></th>
<th>Step Description</th>
<th>Tool</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reserve TCP/IP port for Engine Daemon</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Generate Engine public / private keys</td>
<td>SQDutil</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Add the public key generated in step #2 to the Authorized Key List files on the Source system and target system</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Create the Access Control List Configuration</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Create the Agent Configuration File</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Prepare the Controller Daemon JCL, shell or batch script</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

**Engine Environment Preparation Complete**
Kafka Quickstart

Determine Kafka Payload Format

Precisely supports a variety of target formats for Kafka Topics, including JSON, AVRO and delimited. JSON is the preferred format for streaming changed data (CDC), particularly while testing as it is self-described. AVRO is essentially identical to JSON except that the Schema is maintained separately and the Kafka messages will have a significantly lighter payload compared to JSON.

With AVRO and JSON, you have the option of replicating an entire source record/row, including the before and after image for updates, without any mapping. This is performed via the REPLICATE function and the examples provided later in this document utilize the source descriptions to generate the target kafka payload.

You also have the option of customizing your output with the source to target mapping and dynamic construction of JSON with a custom payload format using Relational DDL type syntax and explicit source to target column mapping. While the custom target DDL can use any combination of upper and lower case as well as hyphens or underscores as the Kafka object names, we highly recommend against hyphens and also highly recommend using all lower case characters. The reason is that both are the defacto standards in the open source Kafka and Java communities. While you may be just getting started with Kafka, most large Kafka and Hadoop users later choose to separate the schemas from the data payload by using AVRO. While JSON is easy for a human to read it consumes significantly more storage. AVRO is one method of separating the two and it imposes much stricter rules on object names and data types.

You can also create a custom message using the PRINTF function. Please note that the scope of this document is not intended to discuss custom JSON. Visit Precisely https://www.precisely.com/support for assistance with this type of processing.

Note: Both JSON and AVRO formatted records are always in Unicode (UTF-8) with the translation from the source codepage/CCSID (i.e. EBCDIC/1047) performed automatically by the engine.

Syntax - JSON

```
DATASTORE kafka:///*                        -- specify dynamic topic
OF JSON                           -- specify JSON format
AS TARGET
DESCRIBED BY GROUP DB2_SOURCE     -- use source for REPLICATE

Sample

{"object_name":SQDATA.EMPLOYEE,
"stk":"00d16918eb9309dc0",
"timestamp":"2018-06-01-13.18.41.780264",
"change_op":"U",
"after_image":
{
  "empno":"000010",
  "first_name":"SANDRA",
  "midinit":"i",
  "last_name":"HAAS",
  "workdept":"A00",
  "phone_no":"3978",
  "hire_date":"1978-09-15",
  "job":"PRES",
  "ed_level":14,
  "sex":"f",
  "birth_date":"1954-03-31",
  "salary":59250.00,
  "bonus":6600.00,
  "comm":2340.00
},
"before_image":
{
```

Note

---

Both JSON and AVRO formatted records are always in Unicode (UTF-8) with the translation from the source codepage/CCSID (i.e. EBCDIC/1047) performed automatically by the engine.

```
Syntax - AVRO

```bash
DATASTORE kafka:///*                        -- specify dynamic topic
OF AVRO                                      -- specify AVRO format
AS TARGET                                    -- use source for REPLICATE
DESCRIBED BY GROUP DB2_SOURCE
```

Sample - AVRO Schema

```json
{""type"":"record",""namespace"":"sqdata",""name"":"employee__replicate",""fields"":[
{""name"":"change_op",""type"":"string"},
{""name"":"object_name",""type"":"string"},
{""name"":"alias",""type"":"string"},
{""name"":"stck",""type"":"bytes",""default"":{"\u0000":1}},{
{""name"":"timestamp",""type"":"string"},{""name"":"seq",""type"":"long"},
{""name"":"after_image",""type"":null},{""type"":"record",""name"":"employee"},
{""name"":"empno",""type"":"string"},
{""name"":"first_name",""type"":"int"},
{""name"":"midinit",""type"":"string"},
{""name"":"last_name",""type"":"string"},
{""name"":"workdept",""type"":"string"},
{""name"":"phone_no",""type"":"string"},
{""name"":"hire_date",""type"":"string"},
{""name"":"job",""type":null},
{""name"":"ed_level",""type":null},
{""name"":"sex",null},
{""name"":"birth_date",null},
{""name"":"salary",null},
{""name"":"bonus",null},
{""name"":"comm",null}]
]
```

Sample - AVRO data payload

No sample payload is included because in addition to removing the schema information and adding a header containing the Schema name, Registry ID number and date/time stamps, other formatting changes make the content of the file all but unreadable. For this reason Precisely suggests testing first with human readable JSON formatted output.
Kafka Quickstart

Create Kafka Apply Engine Script

Kafka Topics are basically message bodies and as such can contain virtually any type of data. In the case of Topics populated by the Connect CDC SQData Apply Engine they generally represent either the complete before and after image of the source data from the Capture Agent or only selected fields from the Captured data that might represent a particular business event. The Payload Format type for Kafka topics may be either AVRO or JSON.

The Kafka Apply Engine script generated in this step is a no-map replication script where the source datastore and target topics contain the identical data. If you have a use case that involves replication of non-identical source/target datastores/topics, you will need to create Kafka target descriptions before creating the Apply Engine.

This section guides you through the following tasks:

1. Add the Source Data Structures
2. Determine the Kafka Datastore specifications
3. Construct the Kafka Apply Engine Script
4. Parse Engine
5. Run Engine

Notes: A Confluent Schema Registry must be installed and configured for use by the Replicator Engine.

Add Source Data Structures

This Quickstart assumes that full before and after data is being placed in a Kafka topic for subsequent downstream processing. The Apply Engine simplifies creation of the topic with a single "REPLICATE" Command. Replicate uses the source data structure "DESCRIPTION" for both source and target automatically generating either AVRO Schemas and Payloads or pure JSON payloads. While Precisely highly recommends using Replicate, some applications and "legacy" Kafka Consumers may require a subset or super-set of source schema components. In those cases Precisely recommends creation of a Relational table like SQL statement or DDL containing the "Column Names" corresponding to each of the source data fields or columns required. The Apply Engine will automatically generate either AVRO Schemas and Payloads or pure JSON payloads from the provided DDL.

Notes:

1. Apply Engines automatically handle both Code page translation and Data Type transformation as well as provide for the handling of invalid source data when applicable.
2. The source data structures, typically in the form of DDL for Relational source data or COBOL copybooks for IMS or VSAM source data are placed into the Data Type specific directories created previously.

Example 1 - DB2

BEGIN GROUP DB2_SOURCE;
DESCRIPTION DB2SQL ./DB2DDL/EMP.ddl AS EMPLOYEE
  KEY IS EMP_NO;
DESCRIPTION DB2SQL ./DB2DDL/DEPT.ddl AS DEPARTMENT
  KEY IS DEPT_NO;
END GROUP;

The content of the Db2 source DDL for the EMPLOYEE table in the EMP.ddl file:

CREATE TABLE EMPLOYEE
  ( EMP_NO CHAR(6) NOT NULL,
    FIRST_NAME VARCHAR(12) NOT NULL,
    MIDINIT CHAR(1) NOT NULL
  );
Example 2 - IMS

```cobol
BEGIN GROUP IMS_DBD;
DESCRIPTION IMSDBD ./IMSDBD/HREMPLDB.dbd AS HREMPLDB;
END GROUP;

BEGIN GROUP IMS_SEG;
DESCRIPTION COBOL ./IMSSEG/EMPLOYEE.cob AS EMPLOYEE
FOR SEGMENT EMPLOYEE IN DATABASE HREMPLDB;
DESCRIPTION COBOL ./IMSSEG/ANNULREV.cob AS ANNULREV
FOR SEGMENT ANNULREV IN DATABASE HREMPLDB;
END GROUP;
```

The content of the IMS source COBOL copybook for the EMPLOYEE segment in the EMP.cob file:

```cobol
05 EMPLOYEE. 00000010
  08 KFLDYK. 00000020
    10 EMP-NO PIC X(6). 00000030
    10 FIRST-NAME PIC X(12). 00000040
    10 MIDINIT PIC X(1). 00000050
    10 LAST-NAME PIC X(15). 00000060
    10 WORK-DEPT PIC X(3). 00000070
    10 PHONE-NO PIC X(4). 00000080
    10 HIRE-DATE PIC 9(8). 00000090
    10 JOB PIC X(8). 00000120
    10 ED-LEVEL PIC 9(2). 00000130
    10 SEX PIC X(1). 00000140
    10 BIRTHDATE PIC 9(8). 00000150
    10 SALARY PIC 9(7)V9(2) COMP-3. 00000160
    10 BONUS PIC 9(7)V9(2) COMP-3. 00000170
    10 COMM PIC 9(7)V9(2) COMP-3. 00000180
    10 FILLER PIC X(10). 00000190
```

Example 3 - VSAM

```cobol
BEGIN GROUP VSAM_REC;
DESCRIPTION COBOL ./COBOL/EMPLOYEE.cob AS EMPLOYEE
DESCRIPTION COBOL ./COBOL/ANNULREV.cob AS ANNULREV
END GROUP;
```

A VSAM source COBOL copybook for the EMPLOYEE record might look identical to the IMS segment example above.
Kafka Quickstart

Determine Kafka Datastore Specifications
The Kafka Topic Target Datastore is identified by the Kafka broker "url" consisting in its most basic form of the Kafka cluster host address and port number along with fully qualified Kafka Topic names and an optional Partition.

Syntax

\[
\text{DATASTORE kafka://[<hostname>[:<port_number>]] / [<kafka_topic_id>] [/<partition> | /key
 | /root_key | /]}
\]

\ OF JSON | AVRO

\ AS <target_alias>

\ DESCRIBED BY GROUP <group_name>

Keyword and Parameter Descriptions

<hostname>:<port_number> Optionally identify specific Kafka Broker Hostname and TCP/IP port. Precisely recommends the dynamic specification of the Kafka Cluster including its host name using the sqdata_kafka_producer.conf file located in the Working Directory of the Apply Engine at launch. The file may actually contain all the configuration options documented by Librdkafka https://github.com/edenhill/librdkafka/blob/master/CONFIGURATION.md. Typically however, only a small subset of those options are specified, including producer specific security information and a list of Kafka Broker hosts. While you will find good reading here https://kafka.apache.org/documentation/#security and here https://docs.confluent.io/4.0.0/security.html, Precisely recommends that you speak to your Kafka Cluster administrator regarding the configuration. These are just three examples:

```
security.protocol=SSL
ssl.ca.location=/app/certificates/dev/abc_root_ca.cert
```

string (PEM format) used for authentication

```
ssl.certificate.location=/home/<kafka_app_user>/kafkassl/client.pem
```

```
ssl.key.location=/home/<kafka_app_user>/kafkassl/client.key
```

```
ssl.key.password=test1234
```

```
metadata.broker.list=<broker_host_01>:<port>,<broker_host_02>:<port>,<broker_host_03>:<port>
```

```
security.protocol=SSL
ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
ssl.truststore.password=test1234
```

```
ssl.key.password=test1234
```

```
metadata.broker.list=<broker_host_01>:<port>,<broker_host_02>:<port>,<broker_host_03>:<port>
```

```
security.protocol=SASL_SSL
sasl.kerberos.service.name=kafka
sasl.kerberos.principal=<kafka_app_user@domain>
```

```
sasl.kerberos.keytab=/app/kafkalib/<kafka_app_user>.keytab
```

```
security.protocol=SASL_SSL
sasl.kerberos.service.name=kafka
```

```
sasl.kerberos.principal=<kafka_app_user@domain>
```

```
sasl.kerberos.keytab=/app/kafkalib/<kafka_app_user>.keytab
```

```
metadata.broker.list=<broker_host_01>:<port>,<broker_host_02>:<port>,<broker_host_03>:<port>
```

<kafka_topic_id> | <prefix>*<suffix> Optionally specify a unique Kafka Topic ID or dynamically specify the Topic using a wildcard. This is particularly useful when creating many topics or unwieldy long topic ID's. A topic containing an "*" indicates that the url is dynamic and the "*" will be replaced with the alias name of the source DESCRIPTION by default or the TOPIC <name> specified as part of the DESCRIPTION. The "*" may be preceded and followed by a string of characters to complete the full Topic name. Whether topics need to be defined to Kafka in advance depends on how Kafka has been configured.
Kafka Quickstart

Optional parameter, in its absence random partitioning amongst the available partitions for a topic will be used. While a specific valid partition number may be specified, Precisely strongly advises not using partition numbers as it become an additional point of maintenance.

The keyword "key" is used for Relational, VSAM and Keyed File sources by Kafka to determine the target partition. This can be very important to insure that successive changes to the same row/record are sent to the same partition, ensuring they will be processed by the consumer in the order of capture. The default functionality for Relational sources is to use the full concatenated list of source key columns while VSAM and Keyed File sources must specify a KEY IS clause on each source DESCRIPTIONS. A KEY IS clause may also be specified for Relational sources to override the default with a specific set of column names from the source description to be used by Kafka for partitioning.

The keyword "root_key" is used only for IMS sources and by default specifies that the only the root key of any captured IMS Segments will be used by Kafka to determine the target partition. Using the root key for all the segments captured in the hierarchy ensures that they will be processed by the consumer in the order of capture and together with all segments updated under a particular root segment.

"/" is required as a placeholder if the SETURLKEY function is to be used to create a custom partitioning Key.

OF JSON | AVRO Kafka "Topics" formatted as either JSON or AVRO
AS <target_alias> Alias of the Target DATASTORE

Notes:

1. Target datastores described by Confluent managed schemas may only be written using the APPLY or the REPLICATE Function.

2. The relationship between the DESCRIPTION Alias, Topic and Subject are matters determined by the planners and architects of the organization's Confluent Schema Registry. The examples used here are arbitrary but were selected based on the source Table Name, and the source application, in the examples below, the EMPLOYEE and DEPARTMENT Tables and a Db2 "IVP_HR" Database. The choice of SUBJECT was made based on the default supported by the Confluent Control Center which requires the SUBJECT to be the same as the TOPIC with the addition of "-value".

3. The Confluent Schema Registry supports multiple Topic naming strategy and all are supported but they may or may not be compatible with other tools including Confluent's own Control Center.

4. The AVRO "schema id" will be supplied at run-time by Confluent based on the TOPIC and SUBJECT parameters specified on the Source DESCRIPTIONs. See the Apply Engine Reference for alternative methods of assigning a schema id.

5. The creation of the partition can be controlled as described above and/or explicitly controlled using the SETURLKEY Function.

Example 1 - Db2 JSON

A capture has been configured for the Department and Employee Db2 tables. An Apply Engine will stream Topics that provide the complete before and after image of the source data resulting from a z/OS business application's normal Insert, Update and Delete processing. The Apply Engine script may consist of only a single REPLICATE statement after providing Descriptions for the source tables.
The url 'kafka:///hr_*_cdc/key' would be interpreted as follows with the brokers specified in the sqdata_kafka_producer.conf file and the topic_id and partition based on the source DESCRIPTIONS.

A topic named 'cdc_EMPLOYEE_db2' will be created for each CDC source record from the EMPLOYEE table whose description was aliased as 'EMPLOYEE'. The value of the EMP_NO column and the EMP_STATE column in the CDC record will be used by Kafka to determine the partition rather than only the default which would be the EMP_NO Key column.

Similarly, the topic named 'cdc_DEPARTMENT_db2' will be created for each CDC source record from the DEPARTMENT table whose description was aliased as 'DEPARTMENT'. The value of the table's key column DEPT_NO will be used by default by Kafka to determine the partition.

```
BEGIN GROUP DB2_SOURCE;
  DESCRIPTION DB2SQL ./DB2DDL/EMP.ddl AS EMPLOYEE
  KEY IS EMP_NO, EMP_STATE;
  DESCRIPTION DB2SQL ./DB2DDL/DEPT.ddl AS DEPARTMENT;
END GROUP;
```

Specification of the Kafka Datastore is thus simplified with only the static portion of the Topic specified and looks like the following:

```
DATASTORE kafka:///hr_*_cdc/key
  OF JSON
  AS TARGET
  DESCRIBED BY GROUP DB2_SOURCE;
REPLICATE (TARGET)
```

Example 2 - Db2 JSON

Using similar Source DESCRIPTIONS from Example 1, the Kafka Cluster can be dynamically specified in the sqdata_kafka_producer.conf file but with a single Topic ID for all CDC records, a randomized partition and a single REPLICATE command:

```
BEGIN GROUP DB2_SOURCE;
  DESCRIPTION DB2SQL ./DB2DDL/EMP.ddl AS EMPLOYEE;
  DESCRIPTION DB2SQL ./DB2DDL/DEPT.ddl AS DEPARTMENT;
END GROUP;
...

DATASTORE kafka:///hr_all_cdc
  OF JSON
  AS TARGET
  DESCRIBED BY GROUP DB2_SOURCE;
...

REPLICATE (TARGET)
```

Example 3 - Db2 JSON

Using the same Source DESCRIPTIONS from Example 2, the Kafka Cluster can be dynamically specified in the sqdata_kafka_producer.conf file but with explicit specification of Topic ID and Partition using the SETURL and SETURLKEY functions:

```
DATASTORE kafka:///*
...
```

Used with the following logic in the Apply Engine script:
Kafka Quickstart

```
CASE RECNAMES(CDICIN)
  WHEN 'EMP' CALLPROC(P_EMP)
  WHEN 'DEPT' CALLPROC(P_DEPT)

CREATE PROC P_EMP AS SELECT
{ SETURL(TARGET, 'kafka:///hr_EMPLOYEE_cdc/')
  SETURLKEY(TARGET, EMP_NO)
  REPLICATE(TARGET, EMP)
}
CREATE PROC P_DEPT AS SELECT
{ SETURL(TARGET, 'kafka:///hr_DEPARTMENT_cdc/')
  SETURLKEY(TARGET, DEPT_NO)
  REPLICATE(TARGET, DEPT)
}
```

Example 4 - Db2 AVRO

Using similar Source DESCRIPTIONS from Example 2 and the Kafka Cluster dynamically specified as in Example 3, a Confluent Schema Registry will be used to automatically manage AVRO Topic Schemas for each source table as those schemas evolve over time:

```
BEGIN GROUP DB2_SOURCE;
DESCRIPTION DB2SQL ./DB2DDL/EMP.ddl AS EMPLOYEE
  KEY IS EMP_NO
  TOPIC hr_EMPLOYEE_cdc
  SUBJECT hr_EMPLOYEE_cdc-value;
DESCRIPTION DB2SQL ./DB2DDL/DEPT.ddl AS DEPARTMENT
  KEY IS DEPT_NO
  TOPIC hr_DEPARTMENT_cdc
  SUBJECT hr_DEPARTMENT_cdc-value;
END GROUP;
```

Specification of the Kafka Datastore is simplified and looks like the following:

```
DATASTORE kafka:///*
  OF AVRO
  FORMAT CONFLUENT
  AS TARGET
  DESCRIBED BY GROUP DB2_SOURCE
  
  REPLICATE (TARGET)
```

Example 5 - IMS AVRO

An IMS Source is very different from Relational in that data relationships are defined by both keys, foreign keys and physical hierarchy. Those differences are minimized as much as possible by using the REPLICATE Command with Kafka targets. One critical difference is how partitions are handled. By specifying "root_key" rather than "key" or defaulting to random partitioning you can ensure that Kafka consumers will process all the data associated with a particular root segment key together and in the proper sequence within a unit-of-work. Like Example 2, a Confluent Schema Registry will be used to automatically manage AVRO Topic Schemas for each source segment as those COBOL descriptions evolve over time:

```
BEGIN GROUP IMS_DBD;
DESCRIPTION IMSDBD ./IMSDBD/HREMPLDB.dbd AS HREMPLDB;
END GROUP;

BEGIN GROUP IMS_SEG;
DESCRIPTION COBOL ./IMSSEG/HREMPLDB/EMPLOYEE.cob AS EMPLOYEE
```
Kafka Quickstart

SPECIFICATION OF THE KAFKA DATASTORE IS SIMPLIFIED AND LOOKS LIKE THE FOLLOWING:

```
DATASOURCE kafka:///*
OF AVRO
FORMAT CONFLUENT
AS TARGET
DESCRIBED BY GROUP IMS_SEG;
```

PROCESSING REQUIRES ONLY ONE STATEMENT:

```
REPLICATE (TARGET)
```

**Construct the Kafka Apply Engine Script**

Three simple Apply Engine scripts are provided below. The first two process Db2 changed data and the third IMS change data. All three utilize Connect CDC SQData's ability to transform the source data DESCRIPTION or schema into the desired JSON or AVRO formatted Kafka message payload. See the Apply Engine Reference for more details regarding its highly extensible capabilities.

**Notes:** Apply Engines utilizing AVRO and the Confluent Schema Registry may not use both APPLY and REPLICATE functions for the same Target Datastore.

**Example 1 - Db2 to JSON formatted Kafka**

Replicate Db2 changed data (CDC) for the IVP EMPLOYEE and DEPARTMENT tables into unique JSON formatted Kafka Topics with default partitioning. An example of the JSON output can be seen above in Determine Kafka Output Format. The example also includes a filter for the EMPLOYEE table. Only updates to employees with a bonus over $5,000 will cause the record to be written to Kafka. All changes to the DEPT table are Replicated with no filter applied.

```
-- Name: DB2TOKAF: Z/OS DB2 To Kafka JSON on Linux
-- Client/Project: client/project
-- SUBSTITUTION PARMS USED IN THIS SCRIPT:
--   %(ENGINE) - ENGINE Name
--   %(SHOST) - Source HOST of the Capture/Publisher
--   %(SPORT) - Source HOST SQDaemon PORT
--   %(PUBNM) - Source Capture/Publisher Agent Name
-- Change Log:
-- 2019-02-01 INITIAL RELEASE using JSON
```

```
JOBNAME DB2TOKAF;
```

```
OPTIONS
CDCOP('I','U','D') -- Set CHANGE OP Constants
,USE AVRO COMPATIBLE NAMES -- Recommended for JSON
```
Kafka Quickstart

BEGIN GROUP DB2_SOURCE;
DESCRIPTION DB2SQL ./DB2DDL/EMP.ddl AS EMPLOYEE
    KEY IS EMP_NO;
DESCRIPTION DB2SQL ./DB2DDL/DEPT.ddl AS DEPARTMENT
    KEY IS DEPT_NO;
END GROUP;

-- Target Descriptions
-- None required

-- Datastore Section

DATASTORE cdc://%(SHOST):%(SPORT)/%(PUBNM)/%(ENGINE)
    OF UTSCDC
    AS CDCIN
    DESCRIBED BY GROUP DB2_SOURCE
;

DATASTORE kafka:///*/key -- specify dynamic topic
    OF JSON -- specify JSON format
    AS TARGET
    DESCRIBED BY GROUP DB2_SOURCE -- use source for REPLICATE
;
-- Field Specification Section

DATEFORMAT 'ISOIBM';

-- Procedure Section

CREATE PROC P_EMPLOYEE AS SELECT
    { IF EMPLOYEE.BONUS > '5000'
        { REPLICATE(TARGET, EMPLOYEE) 
        }
    }
FROM CDCIN;

-- Main Section

PROCESS INTO TARGET
SELECT
{
**Example 2 - Db2 to AVRO formatted Kafka**

After confirming the desired results from the Apply Engine script in Example 1, the Output format will be switched to AVRO, including a Confluent Schema Registry. Only a few elements of the script in Example 1 need to be added or altered as identified by a green bar in the first character of modified lines:

```
CASE RECNAME(CDCIN)
        WHEN 'EMP'  CALLPROC(P_EMPLOYEE)
        WHEN 'DEPT' REPLICATE(TARGET, DEPARTMENT)
}
FROM CDCIN;
```

**Example 3 IMS to AVRO formatted Kafka**

Replicate IMS changed data (CDC) for the IVP EMPLOYEE and ANNULREV segments in the HREMPDLD IMS database into unique AVRO formatted Kafka Topics with partitioning based on the Root Segment key. The example also
Kafka Quickstart

includes a filter for the EMPLOYEE segment. Only updates to employees with a bonus over $5,000 will cause the record to be written to Kafka. All changes to the ANNULREV segment are Replicated with no filter applied.

**Note**, the user friendly AS <alias> names specified in the source DESCRIPTION statements which will be used in the AVRO schema header.

```sql
OPTIONS
    CDCOP('I','U','D')                  -- Set CHANGE OP Constants
    ,USE AVRO COMPATIBLE NAMES           -- Required for AVRO Targets
    ,CONFLUENT REPOSITORY 'http://schema_registry.precisely.com:8081'
;
BEGIN GROUP IMS_SEG;
    DESCRIPTION COBOL ./IMSSEG/EMPLOYEE.cob AS EMPLOYEE   -- User friendly alias
      FOR SEGMENT EMPLOYEE
      IN DATABASE HREMPLDB
      TOPIC IVP_HR_EMPLOYEE
      SUBJECT IVP_HR_EMPLOYEE-value;
    DESCRIPTION COBOL ./IMSSEG/ANNULREV.cob AS ANNULREV    -- User friendly alias
      FOR SEGMENT ANNULREV
      IN DATABASE HREMPLDB
      TOPIC IVP_HR_ANNUAL_REVIEW
      SUBJECT IVP_HR_ANNUAL_REVIEW-value;
END GROUP;
```

**Connect CDC SQData Kafka Quickstart**
Kafka Quickstart

**DATASTORE**

cdc://%(HOST):%(PORT)/%(PUBNM)/%(ENGINE)

OF IMSCDC

AS CDCIN

DESCRIBED BY GROUP IMS_SEG

-- Target Datastore(s)

**DATASTORE**

kafka:///*/

-- specify dynamic topic

OF AVRO

-- specify AVRO format

FORMAT CONFLUENT

-- use Confluent Schema Registry

AS TARGET

DESCRIBED BY GROUP IMS_SEG

-- use source for REPLICATE

-- Field Specification Section

DATEFORMAT 'ISOIBM';

-- Procedure Section

CREATE PROC P_EMPLOYEE AS SELECT

{ IF EMPLOYEE.BONUS > '5000'

{ REPLICATE(TARGET, EMPLOYEE)

}

} FROM CDCIN;

-- Main Section

PROCESS INTO TARGET SELECT

{ CASE RECNAME(CDCIN)

WHEN 'EMPLOYEE' CALLPROC(P_EMPLOYEE)

WHEN 'ANNULREV' REPLICATE(TARGET, ANNULREV)

} FROM CDCIN;

**Note**, Replication of IMS requires that the Target message descriptions maintain the full parent key sequence. This is ensured by SQData when it generates the AVRO schema / JSON message from the Source Datastore Segment Descriptions.

**Parse Engine**

Scripts can be parsed on the Linux platform at the command line, inheriting the working directory and other previously established environmental settings or using a shell script.

**Syntax**

```
SQDPARSE <engine>.sqd <engine>.prc [LIST=ALL|SCRIPT] [ <parm1> <parm2> ...<parmn> ] [>

<engine>.prt]
```

**Examples**
**Kafka Quickstart**

1. Parse, at the command line, an engine script named DB2TOKAF that will perform near real-time replication to Kafka Topics.

   ```bash
   sqdparse ./ENGINE/DB2TOKAF.sqd ./ENGINE/DB2TOKAF.prc ENGINE=DB2TOKAF SHOST=ZOS21 SPORT=2626 PUBNM=DB2TOKAF 2>&1 | tee ./ENGINE/DB2TOKAF.prt
   ``

   **Note:** The last part of the command, "2>&1 | tee ./ENGINE/DB2TOKAF.prt" causes a copy of the parser report to be displayed on the screen and also written to a file.

2. Shell scripts can be used to invoke the Parser in the Linux environment. Shell scripts are useful for automating the Parser processing and/or for parsing several scripts within the same command. The following shell script named Parse DB2TOKAF.sh uses the same command line from the previous example.

   ```bash
   #!/bin/sh
   sqdparse ./ENGINE/DB2TOKAF.sqd ./ENGINE/DB2TOKAF.prc ENGINE=DB2TOKAF SHOST=ZOS21 SPORT=2626 PUBNM=DB2TOKAF 2>&1 | tee ./ENGINE/DB2TOKAF.prt
   ```

**Notes:**

1. The shell scripts used to parse and run Apply Engines are usually stored in what was previously described as the `<SQDATA_VAR_DIR>/<your_hierarchy_here>` directory, making this the "Working Directory". That allows for relative paths to be specified for the Engine script file ./ENGINE/DB2TOKAF.sqd as well as other "parts" files referenced in the main engine script such as source descriptions ./DB2DDL/EMP.ddl, etc.

2. At this time IMSTOKAF Apply Engines that will produce AVRO formatted data will require TWO parses, IF one or more of the Source/Target Datastore DESCRIPTIONS have changed. That is because the AVRO schemas will change and the new schemas will need to be added to the Confluent Schema Repository.

**Run Apply Engine**

On the Linux platform, the Apply Engine can be invoked from the command line, inheriting the working directory and other previously established environmental setting, a shell script or using the Engine Controller Daemon.

**Syntax**

```bash
sqdata <engine>.prc > <engine>.rpt [--parse]
```

**Keyword and Parameter Descriptions**

- `<engine>` Name of the Apply Engine script file used as input to the parser.

- `[--parse]` Optional parameter that invokes the Apply Engine and produces the Schemas for your review.

**Examples**

1. The following command line text invokes the Engine using the Parsed script named DB2TOKAF.prc. The Runtime report is to be written to the file named DB2TOKAF.rpt.

   ```bash
   sqdata ./ENGINE/DB2TOKAF.prc > DB2TOKAF.rpt
   ```

2. The following shell script invokes the Engine using the same Parsed script file named DB2TOKAF.prc. The Runtime report is again written to the file in the working directory named DB2TOKAF.rpt.

   ```bash
   #!/bin/sh
   cd ./ENGINE
   sqdata DB2TOKAF.prc > ../DB2TOKAF.rpt
   ```
Kafka Quickstart

3. The Apply Engine can also be started using the `sqdmon` utility and the previously configured Apply Engine controller daemon. Once again the command can be executed from the command line or using a simple Shell script.

```
  sqdmon START DB2TOKAF
```

Notes:

1. An example of the resulting Kafka Topic for a changed EMPLOYEE table row was illustrated previously in Determine Kafka Output Format.

2. Because the Apply Engine controller daemon was configured to listen on the default TCP/IP port 2626 it is unnecessary to specify the port when running sqdmon. If another port number was used the `--service=<port>` or `-s <port>` would be included in the command.

3. The previously mentioned `sqdata_kafka_producer.conf` file is located in the Working Directory of the Apply Engine at launch therefore, depending on how the engine is started, it may need to be relocated.

4. Apply Engines configured to use AVRO with a Confluent Schema Registry will automatically attempt to retrieve existing registered schemas for all source DESCRIPTIONS at run time. If the schema retrieved matches the one automatically generated by the Apply Engine then processing will proceed. If the schema does not exist or there are differences detected the newly generated schema will be added to the Schema Registry and subsequent Kafka message payloads will be generated based on the new schema. This activity will only occur when this type of Apply Engine is Started. Changes to Source system Schemas will still require coordination to ensure that Engines are Stopped and source Schemas are Updated as part of the normal "Source system Maintenance Schedule".

5. Precisely strongly recommends use of the `--parse` option before running the engine the first time to ensure the generated Schemas meet your needs and expectations before they are registered in the Confluent Schema Registry. This is the proper way to document and review the schema before they are registered.
Kafka Quickstart

Create Kafka Replicator Engine Script

The Kafka Replicator Engine for relational databases beginning with Db2 12 z/OS, fully automates the propagation of source schema changes and Kafka message production using AVRO and the Confluent Schema Registry. The AVRO formatted Kafka Topics produced by the Replicator Engine represent the complete before and after image of the source data from the Capture Agent. The Replicator also supports parallel processing of the replication workload through multiple Workers with the number of threads specified at run-time. Connect CDC SQData truly becomes a utility function within your enterprise architecture, reacting to Relational schema changes without interruption and without maintenance of the SQData Kafka producer configuration running in your Linux environment.

This section guides you through the following tasks:

1. Identify the Source platforms and Relational Tables
2. Determine the Target Kafka URL
3. Create the Kafka Replicator Configuration Script
4. Execute the Replicator Engine

Notes:

- A Confluent Schema Registry must be installed and configured for use by the Replicator Engine.
- The Confluent AVRO convention consists of a small header in each Kafka topic providing the Schema_id used to encode each AVRO formatted Kafka message.
- The Replicator Engine can produce either JSON or AVRO formatted Kafka message payloads.  are produced by a Replicator Engine.

Identify Source Data Structures

While the Replicator Engine will receive source database schemas from the Capture/Publisher and automatically generate the required AVRO Schemas, it is necessary to identify the source tables that will be processed by the Replicator Engine in it's script.

Note, The Engines will automatically handle both Code page translation and Data Type transformation as well as provide for the handling of invalid source data when applicable.

Example

Unlike the Apply Engine example above, the Replicator Engine script requires only the Source Table Names and specification of the Topic and Subject whose content will contain values defined by the planners and architects of the organization’s Confluent Schema Registry. The examples used here are arbitrary but were selected based on the source Table Names, source application and schema source, in this example, the EMPLOYEE and DEPARTMENT Tables, a Db2 "IVP_HR" Database and SQData respectively.

```
MAPPINGS
  SOURCE 'IVP_HR.EMPLOYEE'
  TOPIC IVP_HR.EMPLOYEE
  SUBJECT IVP_HR.EMPLOYEE-value;
  ALIAS 'EMPLOYEE'
  ,SOURCE 'IVP_HR.DEPARTMENT'
  TOPIC IVP_HR.DEPARTMENT
  SUBJECT IVP_HR.DEPARTMENT-value;
  ALIAS 'DEPARTMENT';
```
Kafka Quickstart

Determine Kafka URL
The typical Kafka Topic Target "url" consists of the host address and port number and in the case of the Replicator Engine only dynamically generated Topics that have been automatically registered in the Confluent Schema Registry by the Replicator Engine.

Syntax

```
kafka://[<hostname>:<port_number>]
```

Keyword and Parameter Descriptions

- `<hostname>:<port_number>` Optionally identify specific Kafka Broker Hostname and TCP/IP port. Precisely recommends the dynamic specification of the Kafka Cluster including its host name using the `sqdata_kafka_producer.conf` file located in the Working Directory of the Apply Engine at launch. The file may actually contain all the configuration options documented by Librdkafka https://github.com/edenhill/librdkafka/blob/master/CONFIGURATION.md. Typically however, only a small subset of those options are specified, including producer specific security information and a list of Kafka Broker hosts for example:

```
security.protocol=SASL_SSL
sasl.kerberos.service.name=kafka
sasl.kerberos.principal=<kafka_app_user@domain>
sasl.kerberos.keytab=/app/kafkalib/<kafka_app_user>.keytab
ssl.ca.location=/app/certificates/dev/SAI_RBC_G2_Root_CA.cer
metadata.broker.list=<broker_host_01>:<port>,<broker_host_02>:<port>,<broker_host_03>:<port>
```

Example

Once you have determined the content of the `sqdata_kafka_producer.conf` required by your site, the Kafka URL will be simply specified as follows with Topic ID's retrieved from the Confluent Schema Registry:

```
kafka://[
```

Construct the Kafka Replicator Engine Script
The Replicator Engine utilizes a very simple Script syntax with none of the additional "Parts" files used in an Apply Engine script. No source or target DESCRIPTIONS are required because the Relational Capture agent (initially only Db2 12 z/OS) will utilize the source relational database catalog and subsequent source database maintenance "ALTER TABLE" activity to provide the source Table schema information to the Replicator Engine. The Replicator Engine will similarly interact with the Confluent Schema Registry to maintain the time sequenced Kafka AVRO schemas. No "CDCPROCs" (change data mapping procedures) are required because the entire before and after content of the source Table will be replicated.

Notes:

1. The relationship between the SOURCE Alias, Topic and Subject are matters determined by the planners and architects of the organization's Confluent Schema Registry.
2. The kafka URL is simplified with the brokers specified in the `sqdata_kafka_producer.conf` file and the topic_id and partition retrieved from the AVRO Schema Registry for each Source Table Reference.
3. The addition of a new Table to the source Relational database will not automatically initiate its Replication because a SOURCE reference for each Table, as seen in the examples, is required by the Replicator Engine script.

Example - Db2 to Kafka Replicator
Kafka Quickstart

Replicate Db2 changed data (CDC) for the IVP_HR database, EMPLOYEE and DEPARTMENT tables into unique AVRO formatted Kafka Topics with default partitioning based on table keys. All changes to the source tables are Replicated and no filtering is supported. The values for TOPIC and SUBJECT used here are arbitrary but were selected based on the source Table Names, source application and schema source, in this example, the EMPLOYEE and DEPARTMENT Tables, a Db2 "IVP_HR" Database and SQData respectively. Operation of the Producer will be optimized using one worker thread.

---
-- Name: DB2TOKAF: Z/OS DB2 To Kafka AVRO on Linux
-- Client/Project: client/project
-- Change Log:
-- 2019-02-01 INITIAL RELEASE using Kafka Replicator Engine
-- Replicate Source/Target
---
REPLICATE
  DB2 cdc://<src_host_name>:<src_sqdaemon_port>/<publisher_name>/<replicator_engine>
  TO
  AVRO CONFLUENT 'kafka:///*'
  WITH 1 WORKER
;
-- Processing Option References
---
OPTIONS
  STRIP TRAILING SPACES
  ,CONFLUENT REPOSITORY 'http://schema_registry.precisely.com:8081'
;
-- Source References
---
MAPPINGS
  SOURCE 'IVP_HR.EMPLOYEE'
    TOPIC IVP_HR_EMPLOYEE
    SUBJECT IVP_HR_EMPLOYEE-value;
  ALIAS 'EMPLOYEE'
  ,SOURCE 'IVP_HR.DEPARTMENT'
    TOPIC IVP_HR_DEPARTMENT
    SUBJECT IVP_HR_DEPARTMENT-value;
  ALIAS 'DEPARTMENT';

Run Kafka Replicator Engine
The Replicator Engine can be invoked from the command line, inheriting the working directory and other previously established environmental setting, a shell script or using the Engine Controller Daemon..

Syntax

sqdrpl [-check] <engine>.rpl > <engine>.rpt

Keyword and Parameter Descriptions

<engine> Name of the Replicator Engine script file.

[-check] Optional parameter used to confirm the receipt of published schemas and the location of the Confluent Schema Registry
Kafka Quickstart

Examples

1. The following command line text invokes the Engine using the Replicator Engine script named DB2TOKAF.rpl. The Runtime report is to be written to the file named DB2TOKAF.rpt.

   `sqdrpl [-check] ./ENGINE/DB2TOKAF.rpl > DB2TOKAF.rpt`

2. The Replicator Engine can also be started using the sqdmon utility and the previously configured Engine controller daemon. Once again the command can be executed from the command line or using a simple Shell script.

   `sqdmon START DB2TOKAF`

Notes:

1. We recommend the optional "-check" parameter be used at least once to confirm the location of the Confluent Schema Registry and receipt of published schemas and to ensure the generated Schemas meet your needs and expectations. Schemas not present in the Registry will be output to a file for review.

2. Because the Apply Engine controller daemon was configured to listen on the default TCP/IP port 2626 it is unnecessary to specify the port when running sqdmon. If another port number was used the --service=<port> or -s <port> would be included in the command.

3. The previously mentioned `sqdata_kafka_producer.conf` file is located in the Working Directory of the Apply Engine at launch therefore, depending on how the engine is started, it may need to be relocated.
**Kafka Quickstart**

**Configuration Complete**

Your Capture/Publish and Kafka Apply framework is now setup and running.

You can add more source Tables / Descriptions and configure additional Kafka topics to be populated by the Apply Engine or Replicator Engine script.

Please visit Precisely [https://www.precisely.com/support](https://www.precisely.com/support) for assistance getting started.
Kafka Quickstart

Kafka Operational Issues

This section describes some operational issues that may be encountered while using the Kafka Apply Engine, what they mean and how you may respond to them up to and including a contingency plan are described in the next sections.

Engine Messages

**Maximum UOW limit of 100 Reached** - This message indicates that more than 100 source UOW’s have not been acknowledged by the engine back to the source capture/publisher. This message may simply be a symptom of a slowdown in the Kafka cluster or that the Kafka configuration in use by the engine is either not being properly referenced or contains one or more errors. First thing to check is the location of the sqdata_kafka_producer.conf file. The "working directory" specified in the sqdagents.cfg file may conflict with the files actual location and the Engine’s requirement for it to be in the "working directory" at run time.

Kcat Utility

The open source Kcat utility is a generic command line non-JVM Apache Kafka producer and consumer utility. While it's best known for operating as both a Producer and a Consumer, it provides another important function when starting to test a Connect CDC SQData Apply or Replicator Engine; Kcat is written in C rather than Java and therefore exercises the same libraries and configuration prepared for the Engine.

Kcat like Connect CDC SQData is sometimes referred to as a Swiss Army Tool, in this case a quick and simple tool for inspecting and creating data in Kafka. You can find Kcat on github at [https://github.com/edenhill/kcat](https://github.com/edenhill/kcat). You may find others are already using it inside your organization or that it is already installed on your system.

Among it's many functions, Kcat can be used to quickly confirm your configuration without having to connect to a publisher for test data or wonder what has happened to all those Kafka topics you think you have just written:

- In Producer mode Kcat reads messages from stdin, delimited with a configurable delimiter (-D, defaults to newline), and produces them to the provided Kafka cluster (-b), topic (-t) and partition (-p).
- In Consumer mode Kcat reads messages from a topic and partition and prints them to stdout using the configured message delimiter.
- Kcat features a Metadata list (-L) mode to display the current state of the Kafka cluster and its topics and partitions.
- Most importantly it can also be configured to use the sqdata_kafka_producer.config file.

Finally, it also supports Avro message deserialization using the Confluent Schema-Registry when you are ready to move up to AVRO from JSON, and generic primitive deserializers.

**Examples:**

1. Connect as a Consumer) to "my_broker", read the last 10 messages from the topic named "my_sqdata_test_topic" and then exit.
   
   kcat -C -b my_broker -t my_sqdata_test_topic -p 0 -o -10 -e

2. Connect as a Consumer to the default broker, read and display all the messages from the topic "my_sqdata_test_topic" with the output in hex
   
   kcat -C -b 0 -t my_sqdata_test_topic -o 0 -e | hexdump

3. Connect as a Producer using the same configuration used by the Apply or Replicator Engine as specified in the sqdata_kafka_producer.config file. Continue to run until interrupted by CTL-C.
Kafka Quickstart

    kcat -P -F ./<path_to>/sqdata_kafka_producer.config -t my_sqdata_test_topic -o 0

4. Decode Avro key (-s key=avro), value (-s value=avro) or both (-s avro) to JSON using schema from the Schema-Registry.

    kcat -b my_broker -t my_sqdata_test_topic -s avro -r http://schema-registry-url:8081

Note, Kcat is the new name for the former Kafkacat project. It was renamed to kcat in August 2021 to adhere to the Apache Software Foundation's (ASF) trademark policies. Other than the name, nothing else was changed.

Kafka Contingencies
While very nature of Kafka and its cluster based architecture is to accommodate large volumes of data from Producers, issues can occur. Various factors may make a contingency plan for a Kafka cluster desirable or even a requirement. One such contingency scenario is described below.

The principal issue associated with an extended Kafka outage traces back to the source of the data where the Connect CDC SQData Capture and Publishing occur. Achieving high end to end throughput is accomplished through the careful management of the data that has been captured and the efforts made to avoid the I/O operations required when that transient data must be "landed", in other words written to disk before it is consumed and written to its eventual target, in this case Kafka.

When the Apply or Replicator Engine is unable to write to Kafka that eventually translates to the need to hold the captured data and/or slow down its capture at the source. That can become problematic, particularly when the source itself generates a very high volume of Change Data. When an Engine stops, data cannot be published, committed units-of-work (UOWs) are captured and the data ordinarily held in memory must be written to a transient Storage area. Depending on the Capture that may be a zOS high performance LogStream or disk storage dedicated for this purpose. Eventually the transient data area will be exhausted. When that happens the Capture will slow and eventually stop reading the database log. This is considered an acceptable state albeit not normal or desirable but manageable. The problem that can occur is that the source database log files may be archived, moved to slower storage or even deleted. When that happens Capture will be unable to continue from where it left off and the possibility of data loss becomes reality. Talk to Precisely about the best practices we recommend for log archiving.

If a contingency plan for this situation is required, the following is two approach's may be considered. While it does "land" the data, it will do so on disk often much lower in cost than the on the source machine:

Using SQDUtil

Stop the Apply Engine, if it is still running.

1. Start SQDUtil, using the same source URL as the Engine using the "move" parameter to reconnect to the MF publisher, which will resume publishing from the start of the last "uncommitted" Unit-of-Work. The file system written to by SQDUTIL must be able to accommodate the published "cdcraw" data which will be maintained in its original form, EBCDIC in the case of zOS source change data.

2. When Kafka becomes operational Start a special version of the Engine where the source URL points to the file created by SQDUTIL. If multiple files had been created, the engine would need to be restarted after each file was processed, using the same input file name or a reparsed special Engine (depending on how the files are managed).

3. Once the single or last file has been processed, the special Engine will stop normally.
4. Start the normal Apply Engine to resume normal operation.

**Using a special Engine and a Kafka utility**

1. Start a special version of the engine where the target URL has been modified to write JSON formatted data to files. Depending on the nature of your normal target URL (does it use a single topic, an "*" dynamic topic, or SETURL specified topics), one or more files will created that can contain timestamped file names.

2. When Kafka becomes operational, use a Kafka utility (Kcat comes to mind) to read the JSON file(s) and write all the data to Kafka.

3. Once the last file has been processed, Start the normal Apply Engine to resume operation.

Some things to be considered:

1. None of this may be necessary, given both Kafka's fault tolerance and the capacity of the source to hold the unpublished data. You should consult your Kafka support folks before going down either path.

2. The size of the files created by the SQDUTIL approach may require more or less space than the size of the JSON formatted files created by the second approach. This is a factor of source data volume and your significantly fewer number of target fields derived from each source segment.

3. The number of files created by the second approach will vary based on the style of your normal target URL and the need to create more than one file based on size or time constraints, potentially complicating the scheduling/operation of the Kafka utility.

4. If the normal destination utilized AVRO rather than traditional JSON formatting, there will be no Confluent Schema Registry in that case only the SQDUtility option can be utilized.

Please review the real need for any sort of contingency and then schedule a call to discuss what steps to take next with Precisely Support.
## Index

### A
- Apply Engine 18
- AVRO 4, 16, 24

### C
- Code Page translation 18
- Confluent 12
- Confluent Registry 4

### D
- Db2 24
- DB2 12 z/OS 4
- DB2 Kafka Replicator 31
- DB2 version 12 9
- DB2 z/OS 9
- default libraries 7
- delimited 16
- distributed streaming platform 4

### E
- external library 12

### I
- IMS 20, 24
- installation 18

### J
- JSON 16, 24

### K
- Kafka 4
- Kafka external library 12
- kafka_producer 20
- kafkacat 36
- Kcat 36

### L
- Library Path 18, 31
- Library Requirements 18, 31
- librdkafka.so 12
- Linux 4

### M
- Maximum UOW limit 36
- Maximum UOW limit of 100 Reached 36

### P
- Parse 28
- producer 20

### R
- REPLICATE 16
- Replicator Engine 4

### S
- Segments 20
- SETURLKEY 20
- Shell scripts 28
- SQDATA_KAFKA_LIBRARY 13
- sqdata_kafka_producer.conf 20
- SQDATA_VAR_DIR 28
- system library path 13

### U
- UOW limit 36

### X
- XML 16