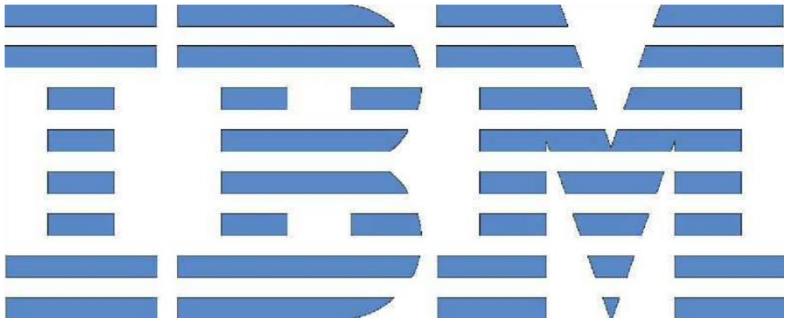

IBM Data Replication
Replicating Changed Data to MapR-FS™

Table of Contents

Introduction: Two Approaches to Replicating to MapR-FS

This document discusses two possible approaches for replicating data from an IBM Data Replication **"CDC"** Source to a MapR-FS.

IBM recommends as a best practice that a proof of concept be done to validate the choices made based on the guidance given here.

- Approach 1: Replicating to MapR-FS using Flat Files as the staging mechanism
- Approach 2: Replicating to MapR-FS Staged Through Kafka™

Replicating Change Data to MapR-FS using Flat Files

Overview

This section describes a strategy to replicate table change data, "CDC data", from a CDC Source to a MapR-FS through first replicating the change data into multiple flat files and then propagating these flat files into the MapR-FS.

While the CDC target engine that generates flat files is called the DataStage Flat File Target, an IBM DataStage installation is NOT required for the afore mentioned flat files to be generated.

Stage 1: Replicating Change Data into Flat Files

Logged changes from a source database's transaction log are scraped by a CDC Source agent and sent via TCP/IP to the CDC Flat File for DataStage Target agent. The CDC Target writes this data to a series of Flat Files, each corresponding to a table and having row data in a comma separated value format. *Note that DataStage is not required nor used. This is GUI artifact.*

Stage 2: Moving Flat Files to MapR-FS

The MapR-FS has several methods of consuming a CSV file. MapR-FS implements hadoop, HDFS, and Map Reduce API's. The specific method chosen is best determined by the MapR-FS admin who understands the workload and common data ingestion patterns for the environment, while taking into consideration the best practices governed by MapR-FS for copying files into the MapR-FS.

Sample View of a CDC Flat File

A sample record in a Flat File in multiple record format for a given table is structured as follows:

Metadata

DM_TIMESTAMP - The timestamp obtained from the log when the operation occurred
DM_TXID - Transaction identifier (dependent on the Source database)
DM_OPERATION_TYPE - I for an insert, D for a delete, "B" for the row containing the before image of an update, and "A" for the row containing the after image of an update.
DM_USER - The user who performed the operation.

Record Example

An update of a row (1, "abc) on the source database to (1, "def") would appear as:

"2017-03-01 20:30:00","73", "B","<user id>","1","abc "
"2017-03-01 20:30:00","73", "A","<user id>","1","def "

How CDC Flat Files are hardened

File Hardening

A consumer of the Flat File can determine that a file is ready for consumption when the naming format of the file changes. This is known as “hardening” of the file. How often a file is hardened is configurable in the CDC Target agent. There is an option to include the number of records in a flat file as part of the file naming convention below but it is not shown in the example.

File Name Hardening Example

For a table, "Tab1", made on Julian date 2018077 at time hh24mmss (GMT) T054618212

Currently Opened indicated by @ sign:
TAB1.@2018077 .T054618212

Completed Flat File Ready for Consumption indicated by "D":
TAB1.D2018077 .T054618212

Methods for Moving Flat Files to MapR-FS

The MapR-FS has several methods of consuming a flat file. MapR-FS implements hadoop, HDFS, and Map Reduce API's. Copying files into the MapR-FS and its best practices are governed by MapR-FS and should be considered. The specific method chosen is best determined by the MapR-FS admin who understands the workload and common data ingestion patterns for the environment.

Methodologies the MapR™ admin may find useful to consider include

- Scripting hdfs dfs –put commands
- Copying to a MapR-FS NFS mount

Sample Flat File Replication Methodology Using hdfs dfs –put

Steps

- Flat files are generated on a server running the CDC Flat File target.
- A scripted process performs an ls of the flat file directory looking for files which have hardened as indicated by the "D" in the name format.
- The file is copied to a staging location on the hadoop server
- The hdfs put command is run,
  e.g. hdfs dfs -put <location of csv file> <location of hdfs>

Sample Flat File Replication Methodology Using MapR-FS NFS Mount

Steps

- Mount the MapR-FS NFS on a linux client.
- Add an NFS mount to /etc/fstab,
  e.g. my-node01:/mapr /mapr nfs rw 0 0
- Configure the NFS Client settings such as the number of outstanding RPC requests on the NFS server to be 128

- Script performing an ls of the flat file directory looking for hardened files as indicated by the "D" in the name format
- Script copying the hardened files from the Flat File Location directory to the NFS mount, either as the files harden or when desired based on the MapR-FS workload.

Advantages of the Flat-File-to-MapR-FS Solution

- CSV file format is very exploitable and well known patterns for processing data in this format exist. MapR highlights this with Apache Drill
- Files placed into the MapR-FS can be limited to those hardened files which represent data at a commit boundary.
- The rate of ingest can be altered by selecting batching parameters in the CDC DataStage For Flat File subscription properties.
- Scheduled timing of the file copy can be arranged to minimize performance impact on the MapR-FS.
- Implementation is relatively simple, as copying files to MapR-Fs is well understood.

Considerations for the Flat-File-to-MapR-FS Solution

- Batching criteria is checked at commit boundaries and so file sizes can vary.
- Some hardened files may be smaller than threshold sizes, as a harden operation hardens all open files to help with data consistency.
- Flat File is not suitable when character columns contain binary data. A base 64 encoding scheme may need to be employed in the user exit for example.
- Tables are individually replicated making ordering of operations cross table not inherent to the format.
- Disk staging space needs to be administered, and I/O speed is in the critical path of throughput performance.

References - Replicating Change Data to MapR-FS using Flat Files

IBM Infosphere CDC for DataStage Flat File

https://www.ibm.com/support/knowledgecenter/SSTRGZ_11.4.0/com.ibm.cdcdoc.cdcfordatastage.doc/concepts/systemrequirements.html

https://www.ibm.com/developerworks/community/files/app?lang=en#/collection/131b8421-396b-488b-865a-68c35805105f

Supporting MapR

http://doc.mapr.com/display/MapR/MapR+Overview

http://doc.mapr.com/display/MapR/Accessing+Data+with+NFS

http://doc.mapr.com/display/MapR/hadoop+fs

https://www.mapr.com/developercentral/code/drilling-csv-files-simple-example#.WLcxDnpj6GQ

Replicating Change Data to MapR-FS Staged Through Kafka

Overview

This section describes a strategy to replicate table change data, “CDC” data, from an IBM CDC Source to a MapR-FS where the change data is first staged into an Apache Kafka Cluster, and MapR supported methodology is used to consume the Apache Kafka data into the MapR-FS.

Stage 1: Replicating Change Data into Apache™ Kafka

Logged changes from a source database's transaction log are scraped by a CDC Source agent and sent via TCP/IP to the CDC for Kafka Target. The CDC target writes the change record data to topics in an Apache Kafka Cluster.

Stage 2: Writing Kafka Records to MapR-FS

MapR provides documentation that confirm that consuming from Apache Kafka and writing the data to MapR-FS is possible via a Kafka HDFS connector.

"The HDFS connector allows you to export data from MapR Streams or Apache Kafka topics to MapR-FS or HDFS files in a variety of formats. " ~ MapR

Replicating Change Data into Apache Kafka – Table Mapping

Source Table to Kafka Cluster Mapping

- A Kafka topic is created for each replicated table in the CDC subscription.
- A Kafka record is comprised of metadata and a Key,Value pair.
- Unique rows in the source table are represented by the "Key" of the Kafka record.
- The "Value" contains the current values for all columns in the row in Avro format.
- Records sent to the topic represent the changes to the rows of a table over time.

Replicating Change Data into Apache Kafka – Table Records

Source Table Operation To Kafka Record Representation

The CDC Kafka targets only writes to Kafka targets through appending records

- Inserts: the Kafka record contains a key and a value
- Delete: Kafka represents a delete as a record with a key and a null for the value.
- Updates:
    - If the column(s) being updated are not key columns, the update is represented in the same way as an insert as described above.
    - If any column being updated is a key, two records are created: a delete record for the old key, and an insert record for the new key.

Sample View of Kafka Key-Value Pair

Avro Format displayed with the Avro console consumer:

Key:

{"COL_KEY":"a"}

Value:

{"COL_KEY":"a","COL_CHAR":{"string":"c"},"COL_VARCHAR":{"string":"v"},"COL_GRAPHIC":{"string":"b"},"COL_VARGRAPHIC":{"string":"H"},"COL_BINARY":{"bytes":"B"},"COL_VARBINARY":null,"COL_SMALLINT":{"int":12},"COL_INTEGER":{"int":25},"COL_BIGINT":{"long":2154},"COL_DECIMAL":{"int":2},"COL_DECFLOAT":{"string":"9.999965"},"COL_REAL":{"float":325.36792},"COL_DOUBLE":{"double":3.1414999999999997},"COL_DATE":{"string":"2016-11-07"},"COL_TIME":{"string":"12:11:02"},"COL_TIMESTAMP":{"string":"2016-11-08T12:11:02.718261000000"}}

Advantages of the Kafka-to-MapR-FS Solution

- Data is written in Avro format, which is compact, efficient to process, and which is a typed format.
- The Kafka Connector framework allows for customization of the format.
- The Confluent HDFS sink is open source.
- Data in a Kafka cluster offers new application options utilizing Kafka for real time event processing and streaming.
- Kafka is very fast with low latency.
- Kafka offers fault tolerance.
- Kafka offers scalability to increase parallelism as number of topics increases.
- By default connector behavior, an Avro schema is provided for each table enhancing the ability to leverage the data.

Considerations for the Kafka-to-MapR-FS Solution

- A Kafka Cluster is required to stage Data, which introduces added complexity compared to the use of a file system to store flat files.
- Kafka has at-least-once semantics.
- Tables are individually replicated making ordering of operations cross table not inherent to the format.

Writing Kafka Records to MapR-FS using the Confluent HDFS Connector sink

MapR leverages the Confluent™ Certified HDFS Sink Kafka Connector in their HDFS connector solution. Details regarding the level of support and methodologies for implementing the specified data export are the providence of MapR.

"http://maprdocs.mapr.com/home/Kafka/Connect-hdfs-connector.html".

A Kafka connector sink framework is included in Kafka and is designed to take streaming data from Kafka and apply it to other systems. The HDFS sink makes use of the HDFS API, implemented by MapR-FS, to push data into a HDFS file.

Configuring HDFS-Sink for MapR-FS

The HDFS Sink takes multiple configuration parameters. Key ones that can be specified include:

- the Kafka topic to read from
  e.g. "topics": "kafka1.subscription3.sourcedb.<schema>.<tablename>"
- the hdfs url - the HDFS connection URL in format hdfs::://hostname:port specifying the HDFS to export data to
  e.g. "hdfs.url": "hdfs://mapr:7222/tmp"
- the flush size, which determines how many records to write to HDFS before invoking file commits
  e.g. "flush.size": "5"
- the top-level HDFS directory to store the write ahead logs, “logs.dir”
- the top-level HDFS directory to store the ingested data from Kafka, ”topics.dir”

Sample Replication Methodology using the Confluent HDFS Connector sink (Kafka-to-MapR-FS solution)

1) On the Kafka Server, ensure Kafka is running and topics to be consumed exist.
2) The HDFS sink is present in the Confluent Kafka package. It is recommended that the latest Confluent Kafka package be used.
   Download and untar the Confluent Kafka package into a separate directory, if not present already, to retrieve it. Copy the following files from the MapR box into the Confluent installation folder. The numbers in the file and folder names could be different depending on the MapR version.

   /opt/mapr/hadoop/hadoop-2.7.0/share/hadoop/hdfs/hadoop-hdfs-2.7.0-mapr-1607.jar

   /opt/mapr/hadoop/hadoop-2.7.0/share/hadoop/common/hadoop-common-2.7.0-mapr-1607.jar

   /opt/mapr/hadoop/hadoop-2.7.0/share/hadoop/common/lib/hadoop-auth-2.7.0-mapr-1607.jar

   /opt/mapr/hadoop/hadoop-2.7.0/share/hadoop/tools/lib/htrace-core-3.1.0-incubating.jar

   /opt/mapr/lib/maprfs-5.2.0-mapr.jar

   /opt/mapr/conf/mapr.login.conf
3) Set the Classpath to point to the MapR implementation of the hadoop/hdfs API rather than the standard hadoop implementation.
   Run the following:

export CLASSPATH=hadoop-hdfs-2.7.0-mapr-1607.jar:hadoop-common-2.7.0-mapr-1607.jar:hadoop-auth-2.7.0-mapr-1607.jar:htrace-core-3.1.0-incubating.jar:maprfs-5.2.0-mapr.jar

This command enforces the MapR implementation of hadoop lib is used rather than pure hadoop lib.

4) In the Confluent platform installation folder, modify etc/kafka-connect-hdfs/quickstart-hdfs.properties. The content of the file should look like this:

```
name=hdfs-sink
connector.class=io.confluent.connect.hdfs.HdfsSinkConnector
tasks.max=1
topics=<provide topic name>
hdfs.url=hdfs://<MapR host name>:<port>/tmp
flush.size=<number of records in the file>
```

The MapR-FS port number and host name can be found in the file /opt/mapr/conf/mapr-clusters.conf.

Look for a line similar to “demo.mapr.com secure=false maprdemo:7222”.

"tmp" is added to the hdfs.url because this folder is usually accessible by any user to avoid permissions issues.

5) Modify the bin/kafka-run-class by adding the following path at the end of the script:

-Djava.security.auth.login.config=<full path to mapr.login.conf file>

e.g. *exec $JAVA $KAFKA_HEAP_OPTS $KAFKA_JVM_PERFORMANCE_OPTS $KAFKA_GC_LOG_OPTS $KAFKA_JMX_OPTS $KAFKA_LOG4J_OPTS*

- *Djava.security.auth.login.config=/home/<user>/confluent-3.x.x/mapr.login.conf -cp $CLASSPATH $KAFKA_OPTS "$@“*

6) Start the HDFS sink connector.

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

Other Technology Possibilities for Writing Kafka Records to MapR-FS

1. Example of consuming Avro data encoded with Confluent sterilizers from Kafka with Spark Streaming: https://github.com/seanpquig/confluent-platform-spark-streaming.
   Reference to MapR’s documented support for Apache Spark:
   https://www.mapr.com/products/apache-spark
2. Example of consuming generic Kafka data from Kafka using Apache Flume:
   http://howtoprogram.xyz/2016/08/06/apache-flume-kafka-source-and-hdfs-sink/
   https://flume.apache.org/releases/content/1.6.0/FlumeUserGuide.html#kafka-source

Reference to MapR's documented support for Flume:
http://doc.mapr.com/display/MapR/Flume

References - Replicating Change Data to MapR-FS Staged Through Kafka

IBM CDC for Kafka Target

https://www.ibm.com/support/knowledgecenter/SSTRGZ_11.4.0/com.ibm.cdcdoc.cdckafka.doc/concepts/systemrequirements.html

The Apache Avro Data Format

https://avro.apache.org/

Kafka Connect Framework info and quickstart

https://www.confluent.io/product/connectors/
http://docs.confluent.io/3.0.0/connect/connect-hdfs/docs/hdfs_connector.html#quickstart
http://docs.confluent.io/3.1.2/connect/quickstart.html

Open Source

https://github.com/confluentinc/kafka-connect-hdfs

MapR connect hdfs connector support

http://maprdocs.mapr.com/home/Kafka/Connect-hdfs-connector.html
http://maprdocs.mapr.com/home/Kafka/Connect-hdfs-example-fromKafka.html
http://maprdocs.mapr.com/home/Kafka/Connect-standalone-mode.html

Notices

IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada

IBM makes no warranties or representations with respect to the content hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. IBM assumes no responsibility for any errors that may appear in this document. The information contained in this document is subject to change without any notice. IBM reserves the right to make any such changes without obligation to notify any person of such revision or changes. IBM makes no commitment to keep the information contained herein up to date.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

All performance data contained in this publication was obtained in the specific operating environment and under the conditions described above and is presented as an illustration only. Performance obtained in other operating environments may vary, and customers should conduct their own testing

The information in this document concerning non-IBM products was obtained from the supplier(s) of those products. IBM has not tested such products and cannot confirm the accuracy of the performance, compatibility, or any other claims related to non-IBM products. Questions about the capabilities of non-IBM products should be addressed to the supplier(s) of those products.

IBM, and the IBM logo are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. Other company, product, or service names may be trademarks or service marks of others. References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.