IBM InfoSphere Data Replication

CDC for Apache Kafka

Installation and Configuration

Version: 1.4

Date: 2020-08-28

Author: Frank Ketelaars

TABLE OF CONTENTS

Introduction

This document covers the installation of IBM InfoSphere Data Replication for Apache Kafka, using the Confluent Platform. Additionally, a few common implementation topics, such as the manual creation of Kafka topics and implementation in a Kerberized Kafka cluster.

INSTALLATION AND RUNNING APACHE KAFKA

By default, CDC for Apache Kafka has been configured to produce Kafka messages serialized in the Apache AVRO format, using the Confluent Platform Schema Registry. Confluent provides easy installation methods using package managers such as yum (Red Hat and CentOS).

If you choose to install Kafka from the Confluent Platform, the ZooKeeper and Schema Registry are included in the package.

In the steps below we assume that the commands are run from the root account. If you do not have access to root and can also not use sudo, refer to the Confluent documentation for non-root installation instructions.

Installation of the Confluent Platform

Installing using the Yum package manager

When installing the Confluent Platform on Red Hat or CentOS, follow the steps below to install using the Yum package manager. You need to be logged on as root to execute the following steps.

1. Import the GPG key for the RPMs to be installed.

```
rpm --import http://packages.confluent.io/rpm/1.0/archive.key
```

2. Create the file with the repository definition, /etc/yum.repos.d/confluent.repo and insert the following lines:

```
[confluent-1.0]
name=Confluent repository for 1.x packages
baseurl=http://packages.confluent.io/rpm/1.0
gpgcheck=1
gpgkey=http://packages.confluent.io/rpm/1.0/archive.key
enabled=1
```

3. Install the Confluent Platform

```
yum install confluent-platform-*
```

Installing from the archive

If you do not have root access to the server you want to install on, or if you plainly prefer to install locally in for example the home directory of user kafka, you can follow the steps below.

1. Download the open source tarball from Confluent using the web browser or directly on the server using wget, for example:

```
wget http://packages.confluent.io/archive/3.3/confluent-oss-3.3.0-2.11.tar.gz
```

2. Untar the file; it will create a directory holding all the components.

```
tar xvzf confluent-oss-3.3.0-2.11.tar.gz
```

Because you have installed the software in your home directory, it cannot be run by other users on the system (if any). Additionally, the configuration files for the ZooKeeper, Kafka brokers and Schema Registry are not contained in the /etc directory structure, but rather reside under the CP installation directory.

Start the Confluent Platform components

The Kafka brokers (aka Kafka servers) and the schema registry are dependent on the ZooKeeper, so this component must be started first. In the below steps, we assume you define one partition per topic and the replication factor is set to 1 (no high availability). It is possible to define more than one partition, even when running on a single server. See section Configuring Kafka topics with multiple partitions on page 13 for more details on creation multi-partition topics.

Start the components as root

If you have installed the Confluent Platform using the package manager, the configuration files reside under /etc and all commands are in the /usr/bin directory where they are accessible to all users.

It is recommended to keep the logs of the components in a single directory so they can easily be found in case of errors.

1. Create a logging directory for the Confluent Platform components (first time only).

```
mkdir -p /var/log/confluent
```

2. Start the ZooKeeper.

```
nohup zookeeper-server-start /etc/kafka/zookeeper.properties >>
/var/log/confluent/zookeeper.log 2>&1 &
```

3. Start the Kafka broker. Please wait a few seconds after starting the ZooKeeper before starting the broker, otherwise the broker may fail to start because it cannot connect to ZooKeeper.

```
nohup kafka-server-start /etc/kafka/server.properties >> /var/log/confluent/kafka.log 2>&1 &
```

4. Start the Schema Registry.

```
nohup schema-registry-start /etc/schema-registry/schema-registry.properties >>
/var/log/confluent/schema-registry.log 2>&1 &
```

Start the components as a regular user

If you downloaded the Confluent Platform tarball and have installed it under your home directory, all commands and configuration files are contained in the directory you have unzipped.

It is recommended to keep the logs of the components in a single directory so they can easily be found in case of errors.

1. Create a logging directory for the Confluent Platform components (first time only).

```
cd ~/confluent-3.3.0
mkdir -p log
```

2. Start the ZooKeeper.

```
nohup ./bin/zookeeper-server-start ./etc/kafka/zookeeper.properties >> ./log/zookeeper.log
2>&1 &
```

3. Start the Kafka broker. Please wait a few seconds after starting the ZooKeeper before starting the broker, otherwise the broker may fail to start because it cannot connect to ZooKeeper.

```
nohup ./bin/kafka-server-start ./etc/kafka/server.properties >> ./log/kafka.log 2>&1 &
```

4. Start the Schema Registry.

```
nohup ./bin/schema-registry-start ./etc/schema-registry/schema-registry.properties >>
./log/schema-registry.log 2>&1 &
```

Validate the installation

The quickest method to validate that Kafka, the ZooKeeper and Schema Registry have been configured correctly is to manually produce a message and consume it. In the steps below we assume that you have installed the Confluent Platform through the package manager or that the ~/confluent-3.3.0/bin is in your path.

1. Start the AVRO console producer.

```
kafka-avro-console-producer --broker-list localhost:9092 --topic test --property
value.schema='{"type":"record","name":"myrecord","fields":[{"name":"msg","type":"string"}]}'
```

2. Send a few messages by pasting the following text:

```
{"msg":"Hi"}
{"msg":"Kafka is now ready"}
{"msg":"to receive CDC messages"}
```

The above steps automatically create a Kafka topic “test” and also the AVRO schema is added to the schema registry.

Now that a few messages have been produced, you can consume them using the AVRO console consumer.

1. Start the AVRO console consumer from a different terminal window.

```
kafka-avro-console-consumer --topic test --zookeeper localhost:2181 --from-beginning
```

2. You should now see the 3 messages that were produced. As you enter new messages on the producer side, they are shown in the consumer terminal window.

You can stop the producer and consumer using Ctrl-C.

Install CDC and configure instance

To be able to receive database changes as messages in Kafka topics, you need to install CDC for Apache Kafka and create an instance.

Install CDC

The installation of CDC for Apache Kafka is a standard installation. Typically you would install CDC in the /opt/ibm/cdc_kafka directory or the appropriate directory for your platform.

1. Create a user to own the CDC installation:

```
useradd iidr
```

2. Optionally set the password for the iidr account. Please note that the password is optional and does not have to match the password you specify when creating the CDC instance.

```
passwd iidr
<enter password twice>
```

3. Create the CDC installation directory

```
mkdir -p /opt/ibm/cdc_kafka
chown iidr:iidr /opt/ibm/cdc_kafka
```

4. Install CDC (using user iidr)

```
./setup-iidr-11.4.0.0-xxxx-linux-x86.bin
```

Now follow the installation steps to get CDC installed.

Configure the CDC instance

Once CDC has been installed, configure the CDC instance.

```
cd /opt/ibm/cdc_kafka/bin
./dmconfigurets
```

You will be prompted to specify the instance name, port number. For instance name, choose a representative identifier such as “cdckafka”, the port number you can leave to the default of 11701. When being asked for the password for tsuser, specify one. This password does not have to match the password of the iidr account and is only used to connect to the CDC for Kafka instance from the Access Server and the subscriptions.

Start the CDC instance

Now that the instance has been created, you can start it (if you did not already do so).

```
cd /opt/ibm/cdc_kafka/bin
nohup ./dmts64 -I cdckafka &
```

The “cdckafka” identifier in the dmts64 command is the name of the instance.

SET UP REPLICATION

Now that Apache Kafka and the Confluent Schema Registry have been configured and CDC installed, you can set up the replication of tables. In the steps below, we assume you already have a CDC source engine installed and configured in the Access Server.

Add CDC for Kafka datastore

Before you can configure a subscription to target Apache Kafka, you need to set up the datastore that points to the CDC for Kafka instance.

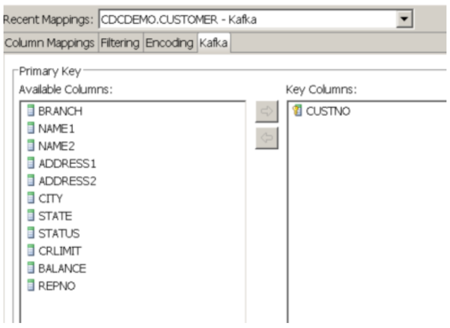
1. Log on to the CDC Management Console and open the Access Server tab.
2. Create a new datastore, for example CDC_Kafka.
3. Specify the host name (or IP address) of the CDC for Kafka server and the port number you defined when you created the CDC instance, typically 11701.
4. Ping the datastore to validate and click Connection Parameters.
5. For the user, specify "tsuser"
6. Enter the password you specified when creating the CDC instance, twice.
7. Once the datastore has been created, assign a user (for example admin) to the datastore so you can connect to the datastore from the Management Console.

Create CDC for Kafka subscription

Now that the target datastore has been set up in the CDC Management Console, you can create a subscription and start mapping tables to Kafka topics.

1. Connect to the source datastore (for example CDC_DB2) and the target datastore (CDC_Kafka).
2. Create a new subscription and select the source and target datastore.
3. Map one or more tables. When mapping tables, it is mandatory to specify the columns that make up the primary/unique key of the source table. These columns are eventually used to make up the Kafka message key.

Example of specifying key columns:

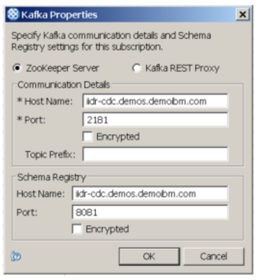

Recent Mappings: CDCDEMO.CUSTOMER - Kafka

Available Columns:	Key Columns:
BRANCH	CUSTNO
NAME1	
NAME2	
ADDRESS1	
ADDRESS2	
CITY	
STATE	
STATUS	
CRLIMIT	
BALANCE	
REPNO	

At this point, the table mappings have been set up. You also need to configure the Kafka properties for the subscription.

1. Right-click the subscription and select "Kafka properties".
2. For the ZooKeeper, specify the host on which the confluent platform was installed and the port (default 2181).
3. For the Schema Registry, again specify the host on which the confluent platform was installed and the port (default 8081).

Example Kafka properties:


Kafka Properties

Specify Kafka communication details and Schema Registry settings for this subscription.

[✓ ZooKeeper Server] Kafka REST Proxy

Communication Details

* Host Name:	iidr-cdc.demos.demoibm.com
* Port:	2181
	☐ Encrypted
Topic Prefix:	

Schema Registry

Host Name:	iidr-cdc.demos.demoibm.com
Port:	8081
	☐ Encrypted

You are now ready to start the replication of changes to Kafka topics.

Start the subscription

Before starting the subscription, determine if you want to refresh the source table contents or you only want to replicate changes from the current point. The default for newly configured tables is Refresh; if you want to replicate only the changes going forward, right click the table(s) and select “Mark Table Capture Point for Mirroring”.

Now you can start the subscription.

CONSUME KAFKA MESSAGES

By default, the Confluent Platform will allow for automatic topic creation. When you start the subscription and the first table change is replicated, Kafka will automatically create the topic(s) using default values (1 partition and replication factor of 1).

Please note that the topics will not get created until the first change is replicated and pushed to the designated Kafka topic. The following topics are created for CDC:

- For every subscription: <topic_prefix>-<datastore_name>-<publisher_ID>-commitstream. The publisher ID can be found in the subscription detail properties and is usually the first 8 positions of the subscription name, in upper case.
- For every table: <datastore_name>.<subscription_name>.<database>.<schema>.<table>, or <topic_prefix>.<schema>.<table>

So, for example if the name of the datastore is “cdckafka” and the name of the subscription is “db2tokafka” and you replicate tables CDCDEMO.CUSTOMER and CDCDEMO.PRODUCT, you will find the below Kafka topics created:

- cdckafka-DB2TOKAF-commitstream
- cdckafka.db2tokafka.sourcedb.cdcdemo.customer
- cdckafka.db2tokafka.sourcedb.cdcdemo.product

The <database> value is “sourcedb” for all tables that are **not** replicated from DB2 for z/OS. If DB2 for z/OS is the source datastore for the subscription, <database> will reflect the name of the database that holds the schema and table replicated in the subscription.

If a topic prefix was specified in the subscription’s Kafka properties (for example “treasury”), the following topics would have been created:

- treasury-cdckafka-DB2TOKAF-commitstream
- treasury.cdcdemo.customer
- treasury.cdcdemo.product

List the Kafka topics

You can check which Kafka topics have been created using the following command on the Kafka server.

```
kafka-topics --list --zookeeper localhost:2181
```

Example topics:

```
kafka-topics --list -zookeeper localhost:2181
__confluent.support.metrics
__consumer_offsets
_schemas
cdckafka-DB2TOKAFKA-commitstream
cdckafka.db2tokafka.sourcedb.cdcdemo.customer
cdckafka.db2tokafka.sourcedb.cdcdemo.product
```

Consume messages from the Kafka topic using the console

By default CDC will produce Kafka messages in Apache AVRO format, hence you need to consume the messages using the AVRO console consumer.

To consume message from topic:

```
kafka-avro-console-consumer --bootstrap-server localhost:9092 --topic
cdckafka.db2tokafka.sourcedb.cdcdemo.product --from-beginning
```

Example messages:

```
{"PRODUCTID":103,"DESCRIPTN":{"string":"DatEpXflLzgvNpgGtQrstEtTliiKTlCElLjcqIEg"},"LOCATION":
{"string":"DuV"},"STATUS":"o","UNITPRICE":"8262786.64","UNITCOST":"5604257.28","QTYONHAND":477
92,"QTYALLOC":40960,"QTYMINORD":36499}
{"PRODUCTID":104,"DESCRIPTN":{"string":"IkG YxWqcUcrSTRHZTk"},"LOCATION":{"string":"Aisle
1"},"STATUS":"u","UNITPRICE":"8218442.89","UNITCOST":"2514374.18","QTYONHAND":74549,"QTYALLOC"
:21575,"QTYMINORD":18882}
{"PRODUCTID":109,"DESCRIPTN":{"string":"qYZvuiXCAm"},"LOCATION":{"string":"Aisle
1"},"STATUS":"o","UNITPRICE":"5453304.44","UNITCOST":"5604257.28","QTYONHAND":49126,"QTYALLOC"
:68938,"QTYMINORD":44852}
null
```

If you would use the kafka-console-consumer, the output would look garbled (although you might recognize some of the replicated data).

By default, CDC for Kafka will generate AVRO messages with the after-image for Refreshed, Inserted and Updated records. In case of a deleted record, the Kafka message will be null.

To be able to determine which record has been inserted, updated or deleted, you will need to specify the consumer option to include the key values. For example:

```
kafka-avro-console-consumer --bootstrap-server localhost:9092 --topic
cdckafka.db2tokafka.sourcedb.cdcdemo.product --from-beginning --property print.key=true
```

The output now also includes the key for every message:

```
{"PRODUCTID":103}
{"PRODUCTID":103,"DESCRIPTN":{"string":"DatEpXflLzgvNpgGtQrstEtTliiKTlCElLjcqIEg"},"LOCATION":
{"string":"DuV"},"STATUS":"o","UNITPRICE":"8262786.64","UNITCOST":"5604257.28","QTYONHAND":477
92,"QTYALLOC":40960,"QTYMINORD":36499}
{"PRODUCTID":104} {"PRODUCTID":104,"DESCRIPTN":{"string":"IkG
YxWqcUcrSTRHZTk"},"LOCATION":{"string":"Aisle
1"},"STATUS":"u","UNITPRICE":"8218442.89","UNITCOST":"2514374.18","QTYONHAND":74549,"QTYALLOC"
:21575,"QTYMINORD":18882}
{"PRODUCTID":109}
{"PRODUCTID":109,"DESCRIPTN":{"string":"qYZvuiXCAm"},"LOCATION":{"string":"Aisle
1"},"STATUS":"o","UNITPRICE":"5453304.44","UNITCOST":"5604257.28","QTYONHAND":49126,"QTYALLOC"
:68938,"QTYMINORD":44852}
{"PRODUCTID":110} null
```

In the above output, you can see that product IDs 103, 104 and 109 were either inserted or updated and product ID 110 was deleted.

ADVANCED CONFIGURATION

This chapter covers several advanced configuration subjects related to CDC for Kafka.

Manually configuring Kafka topics

Many system administrators will not allow Kafka to automatically create new topics when messages arrive. In such cases, when you start a subscription and records get replicated to CDC for Kafka, the subscription will fail immediately.

CDC for Kafka requires the following topics:

- For every subscription: <datastore_name>-<publisher_ID>-commitstream. The publisher ID can be found in the subscription detail properties and is usually the first 8 positions of the subscription name, in upper case.
- For every table: <datastore_name>.<subscription_name>.<database>.<schema>.<table>

To manually create the Kafka topics, execute the below commands for subscription db2tokafka and tables CDCDEMO.CUSTOMER and CDCDEMO.PRODUCT:

```
kafka-topics --zookeeper localhost:2181 --create --topic cdckafka-DB2TOKAF-commitstream --
partitions 1 --replication-factor 1
kafka-topics --zookeeper localhost:2181 --create --topic
cdckafka.db2tokafka.sourcedb.cdcdemo.customer --partitions 1 --replication-factor 1
kafka-topics --zookeeper localhost:2181 --create --topic
cdckafka.db2tokafka.sourcedb.cdcdemo.product --partitions 1 --replication-factor 1
```

Once the topics have been created, you can start the replication.

Configuring Kafka topics with multiple partitions

There may be situations in which you need additional scalability when consuming messages, for example if certain topics have very high volumes of messages (i.e. the source tables generate many change records), or if the processing of the messages is time consuming or processor intensive. Kafka can distribute messages over topic partitions and thereby allow multiple processes to consume messages is parallel.

When Kafka automatically creates a topic upon the first message received, the number of partitions the topic will have is defined in the server.properties used when the Kafka broker was started, property name is num.partitions (default is 1).

If the topic that CDC produces messages on has multiple partitions, this does not mean that every partition will be fed with messages from CDC. You should define a "partitioner" Java class. The Confluent Platform provides a DefaultPartitioner class which takes the hash of the binary message key value to determine the partition number it sends the message to.

Note:	It might be tempting not to specify a key as the messages will then be distributed amongst the partitions in a "round robin" fashion, however not specifying a key means that delete operations will render messages without any reference of which record was removed from the source, so be careful.

You may not want to define multiple partitions for every topic CDC replicates to. We recommend that the topic that keeps subscription commit information is created with only 1 partition.

To create topics with multiple partitions, do the following:

```
kafka-topics --zookeeper localhost:2181 --create --topic cdckafka-DB2TOKAF-commitstream --
partitions 1 --replication-factor 1
kafka-topics --zookeeper localhost:2181 --create --topic
cdckafka.db2tokafka.sourcedb.cdcdemo.customer --partitions 5 --replication-factor 1
kafka-topics --zookeeper localhost:2181 --create --topic
cdckafka.db2tokafka.sourcedb.cdcdemo.product --partitions 7 --replication-factor 1
```

The above commands will only be allowed if the cluster has at least 7 Kafka brokers (because of the “product” topic). You can see that we created only 1 partition for the “commitstream” topic, 5 for the “customer” topic and 7 for the “product” topic.

If the topics were already created, you can change the number of partitions as follows:

```
kafka-topics --zookeeper localhost:2181 --alter --topic
cdckafka.db2tokafka.sourcedb.cdcdemo.product --partitions 7 --replication-factor 1
```

Now that the partitions have been defined, we need to configure CDC to use a partitioner when producing the messages.

Edit the kafkaproducer.properties file and add a property:

```
partitioner.class=org.apache.kafka.clients.producer.internals.DefaultPartitioner
```

To check whether the messages are distributed over multiple partitions, you can run the following command to check the partition offsets:

```
kafka-run-class kafka.tools.GetOffsetShell --broker-list
localhost:9092,localhost:9093,localhost:9094,localhost:9095,localhost:9096 --topic
cdckafka.db2tokafka.sourcedb.cdcdemo.customer --time -1
```

The output could look something like below. You see that partition 0’s offset is 14, for partitions 1 and 3, the offset is 4.

```
cdckafka.db2tokafka.sourcedb.cdcdemo.customer:2:3
cdckafka.db2tokafka.sourcedb.cdcdemo.customer:4:5
cdckafka.db2tokafka.sourcedb.cdcdemo.customer:1:4
cdckafka.db2tokafka.sourcedb.cdcdemo.customer:3:4
cdckafka.db2tokafka.sourcedb.cdcdemo.customer:0:14
```

Purging messages from Kafka topics

At some point during the initial exercises, you may want to purge the messages that have been placed on the Kafka topic(s).

Describe the topic configuration:

```
sudo kafka-configs --zookeeper localhost:2181 --entity-type topics --describe --entity-name
cdckafka.db2tokaf.sourcedb.cdcdemo.customer
```

Set the retention policy of messages to 1 second:

```
sudo kafka-configs --zookeeper localhost:2181 --entity-type topics --entity-name
cdckafka.db2tokaf.sourcedb.cdcdemo.customer --alter --add-config retention.ms=1000
```

Now, wait for 1 minute to allow for Kafka to purge the messages, then remove the configuration to restore to the original retention setting.

```
sudo kafka-configs --zookeeper localhost:2181 --entity-type topics --entity-name
cdckafka.db2tokaf.sourcedb.cdcdemo.customer --alter --delete-config retention.ms
```

Setting up Kerberos for Apache Kafka

While documenting the configuration for CDC with a Kerberized Kafka, we also set up Kerberos for Kafka. Below are the steps that were taken to activate Kerberos for the Confluent Platform; it is not intended to be all-encompassing but is probably helpful if you want to validate CDC with Kerberos and do not yet have a Kerberized Kafka installation.

We assume that a Key Distribution Center (KDC) is already available in the cluster. If you don’t have a KDC, there are many public tutorials available, for example https://gist.github.com/ashrithr/4767927948eca70845db and https://major.io/2012/02/05/the-kerberos-haters-guide-to-installing-kerberos/.

In the below section, the KDC server is called kdc-server.demos.demoibm.com and the Kafka server, which holds the ZooKeeper, Kafka services and the Schema registry is called kafka-server.demos.demoibm.com.

Install the Kerberos client

To install the Kerberos client, run the following steps as root:

```
yum install krb5-libs krb5-workstation
```

A post-installation check you should do is verify that the Kerberos configuration directory exists. While implementing Kerberos, we found that the directory was not created and this resulted in errors with the kadmin tool.

```
mkdir -p /etc/krb5.conf.d
```

Obtain the Kerberos configuration information

The Kerberos configuration information is typically kept in file /etc/krb5.conf on the KDC. You can copy it using scp, or update the /etc/krb5.conf on the Kerberos client yourself.

Example configuration:

```
[root@kafka-server ~]# cat /etc/krb5.conf
# Configuration snippets may be placed in this directory as well
includedir /etc/krb5.conf.d/

[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 dns_lookup_realm = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 rdns = false
 default_realm = IBM.COM
 default_ccache_name = KEYRING:persistent:%{uid}

[realms]
 IBM.COM = {
  kdc = kdc-server.demos.demoibm.com
  admin_server = kdc-server.demos.demoibm.com
 }

[domain_realm]
 .demos.demoibm.com = IBM.COM
 demos.demoibm.com = IBM.COM
```

Create Kerberos principals

We chose to add principals for all 3 services that make up the Confluent Platform: ZooKeeper, Kafka and the Schema registry. Start kadmin on the Kafka server and create the service principals:

```
kadmin -p admin/admin
Authenticating as principal admin/admin with password.
Password for admin/admin@IBM.COM: **********
kadmin: addprinc -randkey zookeeper/kafka-server.demos.demoibm.com@IBM.COM
kadmin: addprinc -randkey kafka/kafka-server.demos.demoibm.com@IBM.COM
kadmin: addprinc -randkey schmaregistry/kafka-server.demos.demoibm.com@IBM.COM
```

It is common practice to use a keytab file that holds the Kerberos principals and their associated keys (aka passwords) so that the services do not have to “log on” using a password. Export the keys of the servie principals to a keytab file that will be used on the Kafka server.

```
kadmin: ktadd -k /home/kafka/kerberos/kafka.keytab zookeeper/kafka-
server.demos.demoibm.com@IBM.COM
kadmin: ktadd -k /home/kafka/kerberos/kafka.keytab kafka/kafka-
server.demos.demoibm.com@IBM.COM
kadmin: ktadd -k /home/kafka/kerberos/kafka.keytab schemaregistry/kafka-
server.demos.demoibm.com@IBM.COM
```

The keytab file that was created should be secured against misuse, hence it is located under the /home/kafka directory.

If you want to check that the principals have been correctly registered, run the following:

```
kadmin: listprincs
```

You can also try to initialize a Kerberos ticket:

```
kinit -k -t /home/kafka/kerberos/kafka.keytab kafka/kafka-server.demos.demoibm.com@IBM.COM
klist
Ticket cache: KEYRING:persistent:1001:1001
Default principal: kafka/kafka-server.demos.demoibm.com@IBM.COM

Valid starting       Expires              Service principal
12/22/2017 11:13:56  12/23/2017 11:13:56  krbtgt/IBM.COM@IBM.COM
```

Create the JAAS files

All the three services within the Confluent Platform must have a JAAS file to configurate authentication with Kerberos.

JAAS file for the ZooKeeper (/home/kafka/kerberos/zookeeper_jaas.conf):

```
Server {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    keyTab="/home/kafka/kerberos/kafka.keytab"
    principal="zookeeper/kafka-server.demos.demoibm.com@IBM.COM";
};
```

JAAS file for the Kafka server (/home/kafka/kerberos/kafka_server_jaas.conf):

```
KafkaServer {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    keyTab="/home/kafka/kerberos/kafka.keytab"
    principal="kafka/kafka-server.demos.demoibm.com@IBM.COM";
};

// ZooKeeper client authentication
Client {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    keyTab="/home/kafka/kerberos/kafka.keytab"
    principal="kafka/kafka-server.demos.demoibm.com@IBM.COM";
};
```

JAAS file for the Schema Registry (/home/kafka/kerberos/schema_registry_jaas.conf):

```
KafkaClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    keyTab="/home/kafka/kerberos/kafka.keytab"
    principal="schemaregistry/kafka-server.demos.demoibm.com@IBM.COM";
};

// ZooKeeper client authentication
Client {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    keyTab="/home/kafka/kerberos/kafka.keytab"
    principal="schemaregistry/kafka-server.demos.demoibm.com@IBM.COM";
};
```

Configure the Kafka services

The following properties must be set for the Confluent Platform services to enable Kerberos authentication.

ZooKeeper properties (zookeeper.properties file under /etc/kafka or <confluent_home>/etc/kafka):

```
# JAAS config
authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
requireClientAuthScheme=sasl
jaasLoginRenew=3600000
```

Kafka service properties (server.properties file under /etc/kafka or <confluent_home>/etc/kafka):

```
zookeeper.connect=kafka-server.demos.demoibm.com:2181
listeners=SASL_PLAINTEXT://:9092

Also add the following:
# SASL interbroker configuration
security.inter.broker.protocol=SASL_PLAINTEXT
sasl.enabled.mechanisms=GSSAPI,PLAIN
sasl.mechanism.inter.broker.protocol=GSSAPI
sasl.kerberos.service.name=kafka
```

Schema registry properties (schema-registry.properties under /etc/schema-registry or <confluent_home>/etc/schema-registry):

```
listeners=http://0.0.0.0:8081
kafkastore.connection.url=kafka-server.demos.demoibm.com:2181
kafkastore.security.protocol=SASL_PLAINTEXT
kafkastore.bootstrap.servers=SASL_PLAINTEXT://kafka-server.demos.demoibm.com:9092
kafkastore.sasl.kerberos.service.name=kafka
```

Starting the services

The starting of the 3 services is almost the same as before, however you must specify the JAAS and Kerberos configuration using an environment variable, KAFKA_OPTS. Starting the services has been documented in sections Start the components as root on page 6 and Start the components as a regular user on page 6.

```
export KAFKA_OPTS="-Djava.security.auth.login.config=/home/kafka/kerberos/zookeeper_jaas.conf
-Djava.security.krb5.conf=/etc/krb5.conf"
nohup ./bin/zookeeper-server-start ./etc/kafka/zookeeper.properties >> ./log/zookeeper.log
2>&1 &

export KAFKA_OPTS="-
Djava.security.auth.login.config=/home/kafka/kerberos/kafka_server_jaas.conf -
Djava.security.krb5.conf=/etc/krb5.conf"
nohup ./bin/kafka-server-start ./etc/kafka/server.properties >> ./log/kafka.log 2>&1 &

export SCHEMA_REGISTRY_OPTS="-
Djava.security.auth.login.config=/home/kafka/kerberos/schema_registry_jaas.conf -
Djava.security.krb5.conf=/etc/krb5.conf"
nohup ./bin/schema-registry-start ./etc/schema-registry/schema-registry.properties >>
./log/schema-registry.log 2>&1 &
```

If the services fail to start and issue Kerberos authentication errors, you can add an addition JVM parameter to add more trace information the logs.

```
-Dsun.security.krb5.debug=true
```

Important. Make sure you remove the parameter and restart the instance as soon as you finish troubleshooting. Leaving the parameter set can severely impact the replication performance.

Configuring CDC with a Kerberized Kafka

CDC for Apache Kafka is typically installed on a separate server or virtual machine that is close to the Kafka cluster. Kerberos requires that all services and clients have been registered in the Kerberos Key Distribution Center (KDC) and when Apache Kafka has been

Kerberized, it would mean that the ZooKeeper and Kafka brokers have already been registered.

You will also need to register the CDC service (the instance) in the Kerberos cluster. Once that has been done, you can start the CDC instance with special properties so that it will authenticate with Kerberos.

At a high level, you need to do the following:

- Obtain or locate the Kerberos configuration information (krb5.conf)
- Create service principal for CDC in the KDC
- Obtain the keytab file that holds the principal information
- Create a JAAS file specifying how CDC must authenticate
- Specify the above attributes as JVM properties to the CDC instance
- Specify the Kafka producer properties
- Install a patch for the IBM Java security provider

Obtain or locate the Kerberos configuration information

If the server that is running CDC is already part of a Kerberos cluster, the cluster configuration is probably already available on the server. Typically, this is kept in file /etc/krb5.conf.

Example configuration:

```
[root@kdc-server ~]# cat /etc/krb5.conf
# Configuration snippets may be placed in this directory as well
includedir /etc/krb5.conf.d/

[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 dns_lookup_realm = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 rdns = false
 default_realm = IBM.COM
 default_ccache_name = KEYRING:persistent:%{uid}

[realms]
 IBM.COM = {
  kdc = kdc-server.demos.demoibm.com
  admin_server = kdc-server.demos.demoibm.com
 }

[domain_realm]
 .demos.demoibm.com = IBM.COM
 demos.demoibm.com = IBM.COM
```

When the client (CDC instance) needs to authenticate, it will use this information connect to the KDC and obtain a TGT.

Create principal for CDC service in the KDC

Start kadmin on any server in the Kerberos cluster, or kadmin.local on the KDC. Subsequently add the CDC service (cdckafka) as a principal. In the example below, the kadmin tool is run on the server that runs CDC for Apache Kafka.

```
kadmin -p admin/admin
Authenticating as principal admin/admin with password.
Password for admin/admin@IBM.COM: **********
kadmin: addprinc -randkey cdckafka/cdc-server.demos.demoibm.com@IBM.COM
```

A common practice for registering services with Kerberos is to create a local keytab file that holds the principals and their associated keys (aka passwords) so that the services do not have to "log on" using a password. Now that the principle has been registered in the KDC, you need to export its key to a keytab file that will be used on the CDC server.

```
kadmin: ktadd -k /home/iidr/kerberos/iidr-cdckafka.keytab cdckafka/cdc-
server.demos.demoibm.com@IBM.COM
```

The keytab file that was created should be secured against misuse. Best is to securely copy the keytab file to the CDC server and place it under a directory that is only accessible by the CDC instance user (iidr in our example).

Create a JAAS file

The Java Authentication and Authorization Service is used by JVMs to authenticate Java applications to Kerberos. Specifics such as the login module to use, the Kerberos principal and location of the keytab are specified in a JAAS configuration file that must be specified as a property to the CDC service (instance) that is started.

Below you will find a sample JAAS file (/home/cdckafka/kerberos/jaas.conf):

```
KafkaClient {
  com.ibm.security.auth.module.Krb5LoginModule required
  credsType=both
  useKeytab="/home/iidr/kerberos/iidr-cdckafka.keytab"
  principal="cdckafka/cdc-server.demos.demoibm.com@IBM.COM";
};
```

The JAAS file only contains an entry to authenticate to the Kafka server (i.e. the brokers). At the time of writing this document, CDC does not support authentication to a Kerberized ZooKeeper.

Note:	If you received the JAAS file from your security administrator, ensure that the login module is the one that is included in the IBM Java runtime engine, **com.ibm.security.auth.module.Krb5LoginModule**. The common module (com.sun.security.auth.module.Krb5LoginModule) does not exist in the jar files and CDC will fail with a ClassNotFoundException if you keep this property in your JAAS file.
Note (2):	The JAAS options for the IBM login module are also different; keyTab and storeKey are not valid IBM JAAS options. In the above sample JAAS file you will see that the useKeytab option specifies a location of the keytab file instead of the Boolean true/false value that is valid for the com.sun.security.auth.module.Krb5LoginModule. Information about valid JAAS options can be found here: https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.api.doc/jgss/com/ibm/security/auth/module/Krb5LoginModule.html

Specify the CDC instance JVM properties

Finally, once the Kerberos configuration has been done and the configuration files are in place on the CDC server, you must change the JVM properties of the CDC instance so that they are picked up when the instance is started.

You can specify JVM properties in the dmts64.vmargs file which is included in the CDC installation directory structure. A global configuration is kept under <cdc_home>/conf, but you can also specify a different configuration per instance by putting the file under <cdc_home>/instance/<instance_name>/conf.

Below, the cdc_home is /opt/ibm/cdc_kafka and the instance name is cdckafka.

```
[iidr@cdc-server ~]# cat /opt/ibm/cdc_kafka/instance/cdckafka/conf/dmts64.vmargs
-Djava.security.auth.login.config=/home/cdckafka/kerberos/jaas.conf -
Djava.security.krb5.conf=/etc/krb5.conf
```

Once the JVM arguments have been set, restart the instance. Also, if you change the contents of the JAAS file or the Kerberos configuration file, you must restart the instance.

Specify Kafka producer and consumer properties

CDC for Apache Kafka utilizes the Kafka producer/consumer APIs and specifies its options through properties file: kafkaproducer.properties and kafkaconsumer.properties.

There are a couple of changes you need to make to this properties file to let CDC successfully authenticate to the Kafka brokers:

```
[iidr@cdc-server conf]$ cat kafkaproducer.properties
batch.size=65536
bootstrap.servers=kafka-server.demos.demoibm.com:9092
security.protocol=SASL_PLAINTEXT
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka

[iidr@cdc-server conf]$ cat kafkaconsumer.properties
bootstrap.servers=kafka-server.demos.demoibm.com:9092
security.protocol=SASL_PLAINTEXT
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
```

A summary of the changed properties:

- bootstrap.servers: Specifies a list of Kafka brokers that can be reached by the producer API. You must specify this property to skip the connection to the ZooKeeper service; at this stage, CDC does not support authentication to a Kerberized ZooKeeper service hence the brokers must be addressed directly.
- sasl.kerberos.service.name: Indicates the service name component of the Kerberos service principal to which CDC will be authenticated. The combination of this identifier and the bootstrap server(s) will make up the server component of the Kerberos authentication. In the above example, CDC will authenticate to kafka/kafka-server.demos.demoibm.com@IBM.COM.
- security.protocol: Specifies the protocol that will be used for authentication; here we are using SASL authentication without encryption of the network traffic (PLAINTEXT).
- sasl.mechanism: SASL mechanism that is used, always specify GSSAPI.

Install patch for the IBM Java security provider

There is a known issue with the IBM Java security provider which has been fixed in the build level of 2017-02-14. You do not need to install a full new version of the JRE that is included in the CDC installation, but only a single file: ibmjgssprovider.jar.

Download an IBM Java 8 JDK or JRE from IBM developerWorks here: https://www.ibm.com/developerworks/java/jdk/. As you only need to copy one JAR file from the distribution, you may choose to download and install the “Simple unzip with license” for your platform.

Run the downloaded binary file and locate the ibmjgssprovider.jar file in the unzipped directory structure; you should find it under the jre/lib directory.

- Stop the CDC for Kafka instance
- Make a backup of the <cdc_home>/jre64/jre/lib/ibmjgssprovider.jar (for example ibmjgssprovider.jar_old) file
- Copy the new version of the ibmjgssprovider.jar to <cdc_home>/jre64/lib
- Restart the CDC for Kafka instance

Troubleshooting Kerberos authentication

If the CDC for Kafka engine fails when trying to connect to Kafka, you must review the trace logs which are kept in <cdc_home>/instance/<instance_name>/log. Here you will find a few details as to why the connection failed. Kerberos authentication errors do not reveal a lot of information so we recommend to switch on the Kerberos debugging option in the CDC engine’s JVM arguments to obtain additional information.

Add the following to the dmts64.vmargs and restart the CDC instance to activate additional logging when Kerberos authentication is done.

```
-Dcom.ibm.security.jgss.debug=all -Dcom.ibm.security.krb5.Krb5Debug=all
```

The Kerberos logging is not included in the CDC traces, but can be found in the logs of the dmts64 command. To view the logging, you can run the dmts64 command from a terminal and view the output; on Linux systems, you will typically find the output in the nohup.out file that is created when you run the dmts64 command in the background.

Important. Make sure you remove the parameters and restart the instance as soon as you finish troubleshooting. Leaving the parameters set can severely impact the replication performance.