

Cloudera Runtime 7.1.6

## Kafka Connect

Date published: 2020-05-21

Date modified: 2021-03-03

# CLOUdera

<https://docs.cloudera.com/>

# Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

# Contents

|   |           |
|---|-----------|
| <b>Kafka Connect Overview.....</b>  | <b>4</b>  |
| <b>Kafka Connect Setup.....</b>   | <b>4</b>  |
| Installing the Kafka Connect Role.....  | 4         |
| Configuring Streams Messaging Manager for Kafka Connect.....                            | 5         |
| <b>Using Kafka Connect.....</b>   | <b>6</b>  |
| Configuring the Kafka Connect Role.....   | 6         |
| Managing, Deploying and Monitoring Connectors.....                                      | 7         |
| Writing Kafka data to Ozone with Kafka Connect.....                                     | 7         |
| Writing data to Ozone in an unsecured cluster with Kafka Connect.....                   | 7         |
| Writing data to Ozone in a Kerberos and TLS/SSL enabled cluster with Kafka Connect..... | 10        |
| <b>Securing Kafka Connect.....</b>  | <b>12</b> |
| Configure TLS/SSL Encryption for the Kafka Connect Role.....                            | 12        |
| Configure Kerberos Authentication for the Kafka Connect role.....                       | 13        |
| Kafka Connect API Security.....   | 13        |
| <b>Connectors.....</b>  | <b>14</b> |
| Installing Connectors.....  | 14        |
| HDFS Sink Connector.....  | 15        |
| Configuration example for writing data to HDFS.....                                     | 16        |
| Configuration example for writing data to Ozone FS.....                                 | 17        |
| Amazon S3 Sink Connector.....   | 18        |
| Configuration Example.....  | 18        |

## Kafka Connect Overview

Get started with Kafka Connect.

Kafka Connect is a tool for streaming data between Apache Kafka and other systems in a reliable and scalable fashion. Kafka Connect makes it simple to quickly define connectors that move large collections of data into and out of Kafka. Source connectors can ingest entire databases or collect metrics from all your application servers into Kafka topics, making the data available for stream processing with low latency. Sink connectors can deliver data from Kafka topics into secondary storage and query systems or into batch systems for offline analysis.

Kafka Connect in CDP is shipped with many different Cloudera developed as well as publicly available sink and source connectors. Each of which cover a specific use case for streaming data. In addition to the connectors available by default, installing custom developed or third-party connectors is also possible. All connectors can be deployed, managed, and monitored using the Streams Messaging Manager UI (recommended), Streams Messaging Manager REST API, or Kafka Connect REST API.

See the *Related Information* section for more information regarding how to set up, configure, use, manage, and monitor Kafka Connect and its connectors.

### Related Information

[Kafka Connect Setup](#)

[Using Kafka Connect](#)

[Securing Kafka Connect](#)

[Connectors](#)

[Streams Messaging Reference](#)

## Kafka Connect Setup

Learn how to set up Kafka Connect.

In order to start using Kafka Connect, you need to deploy Kafka Connect roles and configure Streams Messaging Manager (SMM) so that it can interact with Kafka Connect.

## Installing the Kafka Connect Role

Learn how to install the Kafka Connect role.

### About this task

Kafka Connect in CDP is provided in the form of a Kafka service role. The role is called Kafka Connect. Complete the following steps to install the role on an already running Kafka instance.



**Note:** If you want to install Kafka Connect when deploying a new instance of Kafka, start by using the Add a Service wizard to add and configure a new Kafka service. Follow the prompts given by the wizard. On the Assign roles page, assign one or more hosts for the Kafka Connect role. On the Review Changes page configure the properties detailed in Step 7 and 8 below.

### Procedure

1. Select the Kafka service.
2. Go to Instances.
3. Click Add Role Instances.
4. Select hosts for the Kafka Connect role by clicking the Select Hosts box under Kafka Connect.

5. Select one or more host and click Ok.
6. Click Continue.
7. On the Review Changes page configure the Broker List for Kafka Connect property.

Enter a comma-separated list of IP:port or hostname:port pairs of the brokers that Kafka Connect should connect to. These should be the brokers that are running in the same cluster that Kafka Connect is being deployed on. While you can specify a single broker, Cloudera recommends that you specify multiple for high availability.

8. Review and modify other configuration properties available on this page, and click Continue.
9. Start the Kafka Connect role.

By default Kafka Connect roles will not start when first added.

- a) Select Kafka Connect roles by checking the checkbox next to each role.
- b) Click Actions for Selected>Start .
- c) Click Start when prompted.

## Results

One or more Kafka Connect roles are deployed and running on your cluster.

## What to do next

Configure Streams Messaging Manager for Kafka Connect.

# Configuring Streams Messaging Manager for Kafka Connect

Learn how to set up Streams Messaging Manager for monitoring and managing Kafka Connect.

## About this task

Cloudera recommends that you use Streams Messaging Manager (SMM) to manage and monitor Kafka Connect. In order to do this, first you need to configure SMM so that it can interact with Kafka Connect. Performing the following steps will enable the Connect view in SMM, which allows you to interact with Kafka Connect through a UI interface. Performing these steps will also enable you to use the SMM REST API to interact with Kafka Connect.

## Procedure

1. Select the Streams Messaging Manager service.
2. Go to Configuration.
3. Find and configure the following properties:

- Kafka Connect Host

Enter the hostname of the machine that the Kafka Connect role is deployed on. If you have multiple instances of the Kafka Connect role, you can choose to use any of them. Add a single hostname, as configuring multiple hostnames for high availability is currently not supported.

- Kafka Connect Port

Enter the port that the Kafka Connect role is using. The value of this property has to match the port set in the kafka.connect.rest.port or kafka.connect.secure.rest.port Kafka property. If the Kafka Connect role is not TLS/SSL enabled, use the port specified in kafka.connect.rest.port. If TLS/SSL is enabled for the Kafka Connect role, use the port specified in kafka.connect.secure.rest.port.

- Kafka Connect Protocol

Set this to http if SSL/TLS is not enabled for the Kafka Connect role. Set it to https if SSL/TLS is enabled for the Kafka Connect role.

4. Click Save.
5. Restart the service.

### Results

SMM is configured and is able to manage and interact with Kafka Connect.

### What to do next

Go to the SMM UI and access the Connect view from the left column. Alternatively, use the SMM REST API to interact with Kafka Connect.

## Using Kafka Connect

Learn how you can manage, monitor and configure Kafka Connect.

There are multiple ways that you can manage, monitor and configure Kafka connect in CDP. It is important to make a distinction between the Kafka Connect roles and Kafka Connect connectors as they are managed through different interfaces.

### The Kafka Connect Role

Kafka Connect roles are roles that you can deploy within a Kafka service, they represent Kafka Connect workers. Because these roles are deployed with and managed by Cloudera Manager, their management and configuration is done through Cloudera Manager. In other words the connect workers you deploy will be managed and configured in Cloudera Manager. For more information see [Configuring Kafka Connect](#) and the [Cloudera Manager Documentation](#).

### Kafka Connect Connectors

Connectors are not managed by Cloudera Manager, instead multiple other interfaces can be used to interact with them. These are the following:

- Streams Messaging Manager UI

The Streams Messaging Manager UI is a recommended interface in CDP to manage connectors. Comprehensive documentation is provided on the usage of the UI in [Monitoring Kafka Connect](#).

- The Streams Messaging Manager REST API

The Streams Messaging Manager REST API is a recommended interface in CDP to manage connectors. For more information, see the [SMM REST API Reference](#).



**Note:** Both the Streams Messaging Manager UI and Streams Messaging Manager REST API support the same management actions.

- The Kafka Connect REST API

The Kafka connect REST API can be used to manage connectors. However, its usage is not recommended by Cloudera, nor is separate documentation provided. For more information, see the upstream Apache Kafka documentation.

### Related Information

[Monitoring Kafka Connect](#)

[SMM REST API Reference](#)

[Apache Kafka Documentation](#)

## Configuring the Kafka Connect Role

Learn more about how you can configure the Kafka Connect role as well as its notable properties.

The Kafka Connect role is deployed with and managed by Cloudera Manager. You can view a list of configuration properties as well as configure them by going to Kafka serviceConfiguration and selecting the Kafka Connect filter in the Filters pane. Most of the properties available are the standard worker properties defined in upstream Apache Kafka. However, there are a number of notable properties that you should be familiar with when using Kafka Connect on CDP. The notable properties are the following:

### Broker List for Kafka Connect

These are the brokers Kafka Connect should connect to. You specify the brokers by adding IP:port or hostname:port pairs. The brokers you specify here should be the brokers that are running in the same cluster that Kafka Connect is deployed on. While you can specify a single broker, Cloudera recommends that you specify multiple for high availability.

### group.id

The group ID is a unique string that Kafka Connect roles use to form a cluster of connect workers. This property should always be set to the same string for all Kafka Connect roles that are deployed in the same cluster. Kafka Connect will not function properly if there is a mismatch between the group IDs. Therefore, Cloudera recommends that you use the default values for all roles that you deploy.

### plugin.path

The directory where the Kafka Connect connector plugins are stored. This is the directory where JAR files for custom connectors can be placed. Cloudera recommends that you use the default path which is /var/lib/kafka.

### kafka.connect.rest.port and kafka.connect.secure.rest.port

These are the ports that Kafka Connect API will accept requests on. If the Kafka Connect role is TLS/SSL enabled, it will use the port specified in kafka.connect.secure.rest.port, if TLS/SSL is not enabled, it will use the port specified in kafka.connect.rest.port. Take note of the ports as they are needed when you set up SMM to manage and monitor Kafka Connect.

A comprehensive list of all properties available for the Kafka Connect role is available in Kafka Properties in Cloudera Runtime.

### Related Information

[Kafka Properties in Cloudera Runtime](#)

## Managing, Deploying and Monitoring Connectors

Learn more about managing connectors.

You can manage, monitor, deploy and interact with Kafka Connect and Kafka Connect connectors either through the SMM UI or SMM REST API. For more information see Monitoring Kafka Connect, as well as the SMM REST API Reference.

### Related Information

[Monitoring Kafka Connect](#)

[SMM REST API Reference](#)

## Writing Kafka data to Ozone with Kafka Connect

You can use the Cloudera developed HDFS Sink Connector shipped with Cloudera Runtime to write Kafka data to the Ozone filesystem. This can be done on both secure and unsecure clusters by deploying and configuring a new connector with the SMM UI.

## Writing data to Ozone in an unsecured cluster with Kafka Connect

You can use the Cloudera developed HDFS Sink Connector in an unsecure cluster to write Kafka topic data to Ozone. Connector deployment and configuration is done using the SMM UI.

## About this task

The following list of steps walk you through how the Cloudera developed HDFS Sink Connector can be set up to write data from a Kafka topic to the Ozone file system in an unsecure cluster. The connector is set up and deployed using the SMM UI.

In addition to connector setup, these steps also describe how you can create a test topic and populate it with data using Kafka command line tools. If you already have a topic that is ready for use and do not want to create a test topic, you can skip steps 1 through 3. These steps deal with topic creation, message consumption and message production. They are not necessary to carry out.

## Before you begin

An unsecure CDP PvC Base cluster with Kafka, SMM and Ozone is set up and configured.

## Procedure

### 1. Create a Kafka topic:

- a) SSH into one of the hosts in your cluster.

```
ssh [***USER***]@[***MY-CLUSTER-HOST.COM***]
```

- b) Create a topic with the kafka-topics tool.

```
kafka-topics --create --bootstrap-server [***MY-CLUSTER-  
HOST.COM:9092***] --replication-factor 1 --partitions 1 --top  
ic [***TOPIC***]
```

If an out of memory exception is thrown while running this command, increase the JVM heap with the following command and try again:

```
export KAFKA_OPTS="-Xmx1g -Xms1g"
```

- c) Verify that the topic was created.

```
kafka-topics --list --bootstrap-server [***MY-CLUSTER-HOST.COM:9092***]
```

### 2. Produce messages to your topic with the kafka-console-producer.

```
kafka-console-producer --broker-list [***MY-CLUSTER-HOST.COM:9092***] --t  
opic [***TOPIC***]
```

Start typing messages once the tool is running.

```
>my first message  
>my second message
```

### 3. Consume messages:

- a) Open a new terminal session and log in to one of the hosts in your cluster.
- b) Consume messages with the kafka-console-consumer.

```
kafka-console-consumer --from-beginning --bootstrap-server [***MY-  
CLUSTER-HOST.COM:9092***] --topic [***TOPIC***]
```

The messages you produced with the console producer appear. In addition, you can switch back to the terminal session that is running the kafka-console-producer and produce additional messages. These new messages will appear in real time in the session running the kafka-console-consumer.



#### 4. Deploy and configure a HDFS Sink Connector:

- a) In Cloudera Manager, select the Streams Messaging Manager service.
- b) Click Streams Messaging Manager Web UI.
- c) Click the Connect option in the left-side menu.
- d) Click + New Connector to add a new connector.
- e) Go to the Sink Connectors tab and select the HDFS Sink Connector.

On the UI the HDFS Sink Connector is represented by its class name, which is `com.cloudera.dim.kafka.connect.hdfs.HdfsSinkConnector`.

- f) Enter a name for the connector.
- g) Configure the connector.

Use the following example as a template:

```
{
  "connector.class": "com.cloudera.dim.kafka.connect.hdfs.HdfsSinkConnector",
  "hdfs.uri": "o3fs://bucket1.vol1.ozone1/",
  "hdfs.output": "/topics_output/",
  "tasks.max": "1",
  "topics": "testTopic",
  "hadoop.conf.path": "file:///etc/hadoop/conf",
  "output.writer": "com.cloudera.dim.kafka.connect.partition.writers.txt.TxtPartitionWriter",
  "value.converter": "org.apache.kafka.connect.storage.StringConverter",
  "output.storage": "com.cloudera.dim.kafka.connect.hdfs.HdfsPartitionStorage",
  "hdfs.kerberos.authentication": "false"
}
```

Ensure that you replace the values of `hdfs.uri` and `hdfs.output` with valid Ozone paths. The template gives an example of how these paths should look like. Replace any other values depending on your cluster and requirements.

- h) Click Validate.

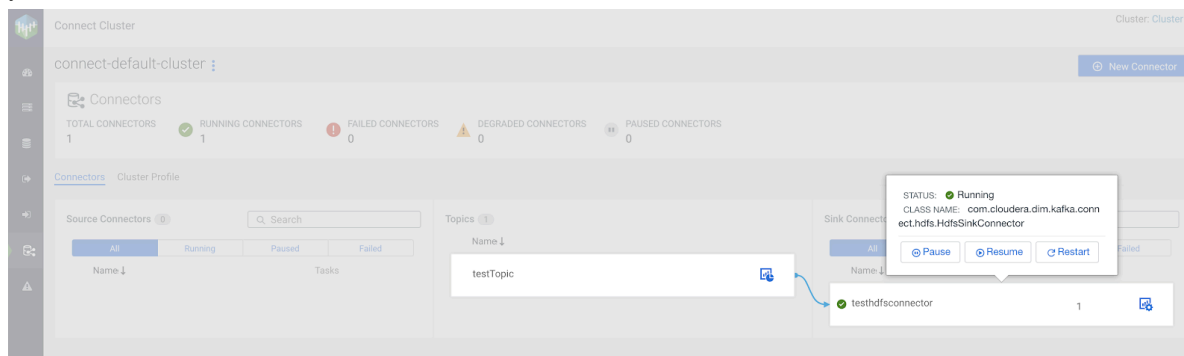
The validator displays any JSON errors in your configuration. Fix any errors that are displayed. If your JSON is valid, the JSON is valid message is displayed in the validator.

- i) Click Next.
- j) Review your connector configuration.
- k) Click Deploy to deploy the connector.

#### 5. Verify that connector deployment is successful:

- a) In the SMM UI, click the Connect option in the left-side menu.
- b) Click on either the topic or the connector you created.

If connector deployment is successful, a flow is displayed between the topic you specified and the connector you created.



## 6. Verify that topic data is written to Ozone.

You can do this by listing the files under the `o3fs://` location you specified in the connector configuration.

## Writing data to Ozone in a Kerberos and TLS/SSL enabled cluster with Kafka Connect

You can use the Cloudera developed HDFS Sink Connector in a secure cluster to write Kafka topic data to Ozone. Connector deployment and configuration is done using the SMM UI.

### About this task

The following list of steps walk you through how the Cloudera developed HDFS Sink Connector can be set up to write data from a Kafka topic to the Ozone file system in a secure cluster. The connector is set up and deployed using the SMM UI.

In addition to connector setup, these steps also describe how you can create a test topic and populate it with data using Kafka command line tools. If you already have a topic that is ready for use and do not want to create a test topic, you can skip steps 1 through 3. These steps deal with topic creation, message consumption and message production. They are not necessary to carry out.

### Before you begin

- A Kerberos and TLS/SSL enabled CDP PvC Base cluster with Kafka, SMM and Ozone is set up and configured.
- If you want to create a test topic with the Kafka console tools, you must ensure that a `.properties` client configuration file is available for use.

You can create one using the following example as a template:

```
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required u
seKeyTab=true keyTab="[***PATH TO KEYTAB FILE***]" principal="[***KERBEROS
PRINCIPAL***]";
security.protocol=SASL_SSL
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
ssl.truststore.location=[***TRUSTSTORE LOCATION***]
```

### Procedure

#### 1. Create a Kafka topic:

- SSH into one of the hosts in your cluster.

```
ssh [***USER***]@[***MY-CLUSTER-HOST.COM***]
```

- Create a topic with the `kafka-topics` tool.

```
kafka-topics --create --bootstrap-server [***MY-CLUSTER-
HOST.COM:9093***] --replication-factor 1 --partitions 1 --topic [***T
OPIC***] --command-config [***CLIENT CONFIG FILE***]
```

If an out of memory exception is thrown, increase the JVM heap with the following command and try again:

```
export KAFKA_OPTS="-Xmx1g -Xms1g"
```

- Verify that the topic was created.

```
kafka-topics --list --bootstrap-server [***MY-CLUSTER-HOST.COM:9093***]
--command-config [***CLIENT CONFIG FILE***]
```

## 2. Produce messages to your topic with the kafka-console-producer.

```
kafka-console-producer --broker-list [***MY-CLUSTER-HOST.COM:9093***] --topic [***TOPIC***] --producer.config [***CLIENT CONFIG FILE***]
```

Start typing messages once the tool is running.

```
>my first message
>my second message
```

## 3. Consume messages:

- Open a new terminal session and log in to one of the hosts in your cluster.
- Consume messages with the kafka-console-consumer.

```
kafka-console-consumer --from-beginning --bootstrap-server [***MY-CLUSTER-HOST.COM:9093***] --topic [***TOPIC***] --consumer.config [***CLIENT CONFIG FILE***]
```

The messages you produced with the console producer appear. In addition, you can switch back to the terminal session that is running the kafka-console-producer and produce additional messages. These new messages will appear in real time in the session running the kafka-console-consumer.

## 4. Deploy and configure a HDFS Sink Connector:

- In Cloudera Manager, select the Streams Messaging Manager service.
- Click Streams Messaging Manager Web UI to log in to the UI.  
If prompted, enter a valid username as well as a password and click SIGN IN.
- Click the Connect option in the left-side menu.
- Click + New Connector to add a new connector.
- Go to the Sink Connectors tab and select the HDFS Sink Connector.

On the UI the HDFS Sink Connector is represented by its class name, which is `com.cloudera.dim.kafka.connector.hdfs.HdfsSinkConnector`.

- Enter a name for the connector.
- Configure the connector.

Use the following example as a template:

```
{
  "connector.class": "com.cloudera.dim.kafka.connect.hdfs.HdfsSinkConnector",
  "hdfs.uri": "o3fs://bucket1.voll.ozonel/",
  "tasks.max": "1",
  "topics": "testTopic",
  "hdfs.kerberos.authentication": "true",
  "hdfs.kerberos.user.principal": "${cm-agent:ENV:kafka_connect_service_principal}",
  "hdfs.kerberos.keytab.path": "${cm-agent:keytab}",
  "hdfs.kerberos.namenode.principal": "hdfs/_HOST@REALM",
  "hdfs.output": "/topics_output/",
  "hadoop.conf.path": "file:///etc/hadoop/conf",
  "output.writer": "com.cloudera.dim.kafka.connect.partition.writers.txt.TxtPartitionWriter",
  "value.converter": "org.apache.kafka.connect.storage.StringConverter",
  "output.storage": "com.cloudera.dim.kafka.connect.hdfs.HdfsPartitionStorage"
}
```

Ensure that you replace the values of `hdfs.uri` and `hdfs.output` with valid Ozone paths. The template gives an example of how these paths should look like. Replace other values depending on your cluster and requirements.

h) Click Validate.

The validator displays any JSON errors in your configuration. Fix any errors that are displayed. If your JSON is valid, the JSON is valid message is displayed in the validator.

i) Click Next.

j) Review your connector configuration.

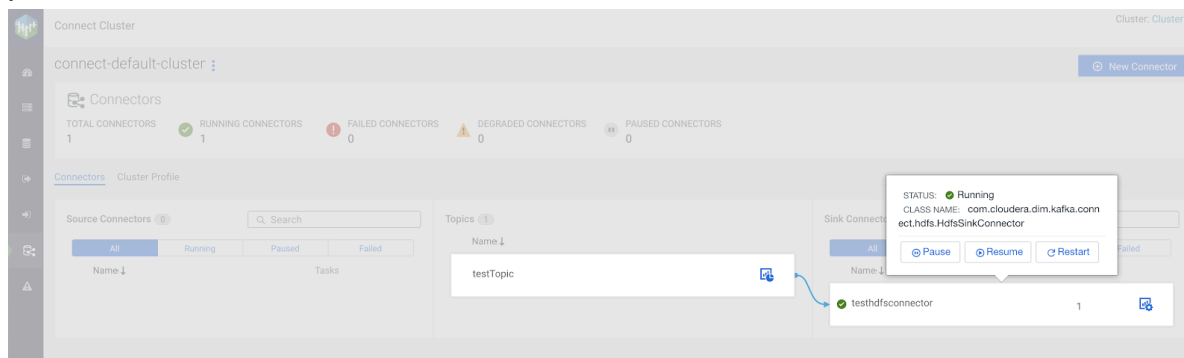
k) Click Deploy to deploy the connector.

5. Verify that connector deployment is successful:

a) In the SMM UI, click the Connect option in the left-side menu.

b) Click on either the topic or the connector you created.

If connector deployment is successful, a flow is displayed between the topic you specified and the connector you created.



6. Verify that topic data is written to Ozone.

You can do this by listing the files under the `o3fs://` location you specified in the connector configuration.

## Securing Kafka Connect

Learn how to secure Kafka Connect role instances (workers) as well as the Kafka Connect API.

### Configure TLS/SSL Encryption for the Kafka Connect Role

Kafka Connect roles inherit the TLS/SSL configuration of the parent Kafka service. If you are deploying Kafka Connect roles under a Kafka service that already has TLS/SSL enabled, Cloudera Manager will automatically enable TLS/SSL for Connect as well. If required however, you can manually enable or disable TLS/SSL.

#### Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to Configuration.
3. Find and configure the following properties based on your cluster and requirements.

**Table 1:**

| Cloudera Manager Property  | Description  |
|--|--|
| Enable TLS/SSL for Kafka Connect<br><code>ssl_enabled</code>   | Encrypt communication between clients and Kafka Connect using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).                      |
| Kafka Connect TLS/SSL Server JKS Keystore File Location<br><code>ssl_server_keystore_location</code> | The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Kafka Connect is acting as a TLS/SSL server. |

| Cloudera Manager Property   | Description   |
|---|---|
| Kafka Connect TLS/SSL Server JKS Keystore File Password<br>ssl_server_keystore_password   | The password for the Kafka Connect keystore file.   |
| Kafka Connect TLS/SSL Server JKS Keystore Key Password<br>ssl_server_keystore_keypassword | The password that protects the private key contained in the JKS keystore used when Kafka Connect is acting as a TLS/SSL server.   |
| Kafka Connect TLS/SSL Trust Store File<br>ssl_client_truststore_location                  | The location on disk of the trust store, used to confirm the authenticity of TLS/SSL servers that Kafka Connect might connect to. This trust store must contain the certificate(s) used to sign the service(s) connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.                                       |
| Kafka Connect TLS/SSL Trust Store Password<br>ssl_client_truststore_password              | The password for the Kafka Connect TLS/SSL Trust Store File. This password is not required to access the trust store; this field can be left blank. This password provides optional integrity checking of the file. The contents of trust stores are certificates, and certificates are public information.   |
| ssl.client.auth   | Client authentication mode for SSL connections. This configuration has three valid values, required, requested, and none. If set to required, client authentication is required. If set to requested, client authentication is requested and clients without certificates can still connect. If set to none, which is the default value, no client authentication is required |

4. Click Save Changes.

5. Restart the service.

### Results

TLS/SSL encryption is configured for the Kafka Connect role.

## Configure Kerberos Authentication for the Kafka Connect role

Learn how Kerberos authentication is configured for the Kafka Connect role.

Kafka Connect roles inherit the Kerberos configuration of the parent Kafka service. If you are deploying Kafka Connect roles under a Kafka service that already has Kerberos enabled, Cloudera Manager will automatically enable Kerberos for Kafka Connect as well. Other than making sure that the Kafka service's Kerberos configuration is correctly set, no additional user action is required.

## Kafka Connect API Security

Learn about Kafka Connect API and how to configure it.

### About this task

You can secure the Kafka Connect API by configuring the Kafka Connect roles to require SSL Client authentication. This can be done by setting the SSL Client Authentication property to required. When set to required, only clients that pass SSL client authentication will be able to access the Kafka Connect API. As a result, any client that you would like to give access to should have its certificate added to the Kafka Connect truststore. This includes Streams Messaging Manager (SMM) as well. Cloudera recommends that in secure environments only SMM is given access to the Kafka Connect API.

In addition to setting client authentication to required, you may also want to consider setting up a firewall using third party tools to further secure access to the Kafka Connect API. Note however, that even with a firewall in place and SSL authentication set to required, if SMM is given access to the Kafka Connect API, then any user that has access to SMM will be able to interact with the Kafka Connect API. This is due to SMM not enforcing authorization checks

when users are accessing Kafka Connect functionality within SMM. This is true for both the SMM UI and SMM REST API. As a result, caution is advised even if the Kafka Connect API itself is secured.

Complete the following steps to set SSL Client Authentication to required.

### Procedure

1. Select the Kafka Service.
2. Go to Configuration.
3. Find the SSL Client Authentication property.
4. Set the property to required.



#### Important:

The SSL Client Authentication property is available for both Broker and Connect roles. Make sure that you are configuring the property for the Kafka Connect role or for the role group that includes your Kafka Connect roles.

5. Click Save Changes.
6. Restart the service.

### Results

Only authenticated clients are allowed to connect to the Kafka Connect API.

### What to do next

If you are using SMM to manage and monitor Kafka Connect, and you are not using auto TLS, add SMM's certificate to the Kafka Connect truststore.

## Connectors

Learn more about the connectors available Runtime.

CDP Runtime comes prepackaged with two Cloudera developed Kafka Connect sink connectors. These are the following:

- HDFS Sink Connector
- Amazon S3 Sink Connector

In addition, connectors that come packaged with the version of Apache Kafka that is included in Runtime are also available for use. You can also manually install and use your own connectors.

## Installing Connectors

Learn how to install connectors for use with Streams Messaging Manager.

### About this task

Kafka Connect connectors are distributed as:

- A directory of JAR files:  
The directory includes the JAR for the connector itself, as well as all its dependencies.
- An uber JAR/FAT JAR/JAR with dependencies file:

This a single JAR file that contains the connector, as well as its dependencies.

In CDP all connectors that do not come prepackaged with the Runtime distribution have to be installed manually. This is done by making the connector JAR files available on the cluster hosts in a specific location.

### Which host?

The connector files have to be made available on all hosts that are running Kafka Connect roles.

### What location?

Kafka Connect discovers connectors by looking at a specific directory path on the host machine. The path it checks is determined by the Kafka Connect role's `plugin.path` property. By default the `plugin.path` property is set to `/var/lib/kafka`. This means that by default any connector placed in this directory will be discovered by Kafka Connect. Cloudera recommends that you use the default path.



**Important:** Steps 1 and 2 have to be carried out on all hosts in the cluster that have Kafka Connect roles deployed on them.

### Procedure

1. Log in to a host that is running a Kafka Connect role.
2. Make the connector files available in or readable from `/var/lib/kafka`.

There are multiple ways you can achieve this, how you choose to complete this step will largely depend on your cluster environment. For example:

- You can download or copy the files directly to `/var/lib/kafka`.
- You can choose to place connector files in a location different from `/var/lib/kafka` and create symlinks that point to the location where the connector files are available.

Regardless of what method you choose, this step is considered complete once the connector files are readable from `/var/lib/kafka`.

3. Restart all Kafka Connect roles
  - a) In Cloudera Manager, select the Kafka service.
  - b) Go to Instances.
  - c) Select all Kafka Connect instances by checking the checkbox next to each instance.
  - d) Click Actions for selected Restart.
  - e) Click Restart to confirm.

The roles are restarted once a Finished status is displayed.

### Results

The connectors are installed and available for use. You are now able to deploy and manage the new connector from the SMM UI.

## HDFS Sink Connector

Learn more about the HDFS Sink Connector.

The HDFS Sink Connector can be used to transfer data from Kafka topics to files on HDFS clusters. Each partition of every topic results in a collection of files named in the following pattern:

```
{topic name}_{partition number}_{end_offset}.{file extension}
```

For example, running the HDFS Sink Connector on partition 0 of a topic named `sourceTopic` can yield the following series of files:

```
sourceTopic_0_50.avro - for record 0 ~ 50
sourceTopic_0_79.avro - holding record 51 ~ 79
...
```

The HDFS Sink Connector periodically commits records to final result files. Each commit results in a separate "chunk" file.

## Configuration example for writing data to HDFS

A simple configuration example for the HDFS Sink Connector.

The following is a simple configuration example for the HDFS Sink Connector. Short descriptions of the properties set in this example are also provided. For a full properties reference, see the [HDFS Sink Connector Properties Reference](#).

```
{
  "connector.class": "com.cloudera.dim.kafka.connect.hdfs.HdfsSinkConnector",
  "tasks.max": 1,
  "key.converter": "org.apache.kafka.connect.storage.StringConverter",
  "value.converter": "com.cloudera.dim.kafka.connect.converts.AvroConverter",
  "value.converter.passthrough.enabled": true,
  "value.converter.schema.registry.url": "http://localhost:9090/api/v1",
  "topics": "avro_topic",
  "hdfs.uri": "hdfs://my-host.my-realm.com:8020",
  "hdfs.output": "/topics_output/",
  "output.writer": "com.cloudera.dim.kafka.connect.hdfs.avro.AvroPartitionWriter",
  "output.avro.passthrough.enabled": true,
  "hdfs.kerberos.authentication": true,
  "hdfs.kerberos.user.principal": "user_account@MY-REALM.COM",
  "hdfs.kerberos.keytab.path": "/path/to/user_account.keytab",
  "hdfs.kerberos.namenode.principal": "hdfs/_HOST@MY-REALM.COM",
  "hadoop.conf.path": "/etc/hadoop/"
}
```

### connector.class

Class name of the HDFS Sink Connector.

### key.converter

The converter capable of understanding the data format of the key of each record on this topic.

### value.converter

The converter capable of understanding the data format of the value of each record on this topic.



**Note:** When the AvroConverter is used, you can specify Schema Registry properties to be used by the AvroConverter's Schema Registry client. This is done by adding the required Schema Registry property as a suffix to the value.converter property. For example, value.converter.schema.registry.url. Properties defined this way are passed on to the Schema Registry client used by the AvroConverter.

### value.converter.passthrough.enabled

This property controls whether or not data is converted into the Kafka Connect intermediate data format before writing into an output file. Because in this example the input and output format is the same, the property is set to true, that is, data is not converted.

### value.converter.schema.registry.url

The URL to Schema Registry. This is a mandatory property if the topic has records encoded in Avro format.

### topics

List of topics to consume data from.

### hdfs.uri

The URI to the namenode of the HDFS cluster.

### hdfs.output

The destination folder on the HDFS cluster where output files will reside.



**output.writer**

Determines the output file format. Because in this example the output format is Avro, `AvroPartitionWriter` is used.

**output.avro.passthrough.enabled**

This property has to match the configuration of the `value.converter.passthrough.enabled` property because both the input and output formats are Avro.

**hdfs.kerberos.authentication**

Enables or disables kerberos authentication.

**hdfs.kerberos.user.principal**

The user principal that the Kafka Connect role will use.

**hdfs.kerberos.keytab.path**

The path to the kerberos keytab file.

**hdfs.kerberos.namenode.principal**

The Kerberos principal used by the namenode. This is necessary when the HDFS cluster has data encryption turned on.

**hadoop.conf.path**

The path to the hadoop configuration files. This is necessary when the HDFS cluster has data encryption turned on.

**Related Information**

[HDFS Sink Connector Properties Reference](#)

**Configuration example for writing data to Ozone FS**

A simple configuration example for the HDFS Sink Connector that writes data to the Ozone FS.

The following is a simple configuration example for the HDFS Sink Connector. In this example data is written to the Ozone FS. Short descriptions of the properties set in this example are also provided. For a full properties reference, see the HDFS Sink Connector Properties Reference.

```
{
  "connector.class": "com.cloudera.dim.kafka.connect.hdfs.HdfsSinkConnector",
  "tasks.max": 1,
  "key.converter": "org.apache.kafka.connect.storage.StringConverter",
  "value.converter": "com.cloudera.dim.kafka.connect.converts.AvroConverter",
  "value.converter.passthrough.enabled": true,
  "value.converter.schema.registry.url": "http://localhost:9090/api/v1",
  "topics": "avro_topic",
  "hdfs.uri": "o3fs://bucket1.volumel.ozone1/",
  "hdfs.output": "/topics_output/",
  "output.writer": "com.cloudera.dim.kafka.connect.hdfs.avro.AvroPartitionWriter",
  "output.avro.passthrough.enabled": true,
  "hdfs.kerberos.authentication": true,
  "hdfs.kerberos.user.principal": "user_account@MY-REALM.COM",
  "hdfs.kerberos.keytab.path": "/path/to/user_account.keytab",
  "hadoop.conf.path": "/etc/hadoop/"
}
```

**connector.class**

Class name of the HDFS Sink Connector.

**key.converter**

The converter capable of understanding the data format of the key of each record on this topic.

**value.converter**

The converter capable of understanding the data format of the value of each record on this topic.



**Note:** When the AvroConverter is used, you can specify Schema Registry properties to be used by the AvroConverter's Schema Registry client. This is done by adding the required Schema Registry property as a suffix to the value.converter property. For example, value.converter.schema.registry.url. Properties defined this way are passed on to the Schema Registry client used by the AvroConverter.

**value.converter.passthrough.enabled**

This property controls whether or not data is converted into the Kafka Connect intermediate data format before writing into an output file. Because in this example the input and output format is the same, the property is set to true, that is, data is not converted.

**value.converter.schema.registry.url**

The URL to Schema Registry. This is a mandatory property if the topic has records encoded in Avro format.

**topics**

List of topics to consume data from.

**hdfs.uri**

The o3fs URI.

**hdfs.output**

The destination folder on the HDFS cluster where output files will reside.

**output.writer**

Determines the output file format. Because in this example the output format is Avro, AvroPartitionWriter is used.

**output.avro.passthrough.enabled**

This property has to match the configuration of the value.converter.passthrough.enabled property because both the input and output formats are Avro.

**hdfs.kerberos.authentication**

Enables or disables kerberos authentication.

**hdfs.kerberos.user.principal**

The user principal that the Kafka Connect role will use.

**hdfs.kerberos.keytab.path**

The path to the kerberos keytab file.

**hadoop.conf.path**

The path to the hadoop configuration files. This is necessary when the HDFS cluster has data encryption turned on.

## Amazon S3 Sink Connector

Learn more about the S3 Sink Connector

The S3 connector allows users to stream Kafka data into S3 buckets.

## Configuration Example

A simple configuration example for the Amazon S3 Sink Connector.

The following is a simple configuration example for the Amazon S3 Sink Connector. Short descriptions of the properties set in this example are also provided. For a full properties reference, see the Amazon S3 Sink Connector Properties Reference.

```
{
  "aws.s3.bucket": "bring-me-the-bucket",
  "aws.s3.service_endpoint": "http://myendpoint:9090/",
  "aws.access_key_id": "EXAMPLEID",
  "aws.secret_access_key": "EXAMPLEKEY",
  "connector.class": "com.cloudera.dim.kafka.connect.s3.S3SinkConnector",
  "tasks.max": 1,
  "key.converter": "org.apache.kafka.connect.storage.StringConverter",
  "value.converter": "com.cloudera.dim.kafka.connect.converts.AvroConverter",
  "value.converter.passthrough.enabled": true,
  "value.converter.schema.registry.url": "http://schema-registry:9090/api/v1",
  "topics": "avro_topic",
  "output.storage": "com.cloudera.dim.kafka.connect.s3.S3PartitionStorage",
  "output.writer": "com.cloudera.dim.kafka.connect.partition.writers.avro.AvroPartitionWriter",
  "output.avro.passthrough.enabled": true
}
```

#### **aws.s3.bucket**

Target S3 bucket name.

#### **aws.s3.service\_endpoint**

Target S3 host and port.

#### **aws.access\_key\_id**

The AWS secret key ID used for authentication.

#### **aws.secret\_access\_key**

The AWS secret access key used for authentication.

#### **connector.class**

Class name of the Amazon S3 Sink Connector.

#### **tasks.max**

Maximum number of tasks.

#### **key.converter**

The converter capable of understanding the data format of the key of each record on this topic.

#### **value.converter**

The converter capable of understanding the data format of the value of each record on this topic.



**Note:** When the AvroConverter is used, you can specify Schema Registry properties to be used by the AvroConverter's Schema Registry client. This is done by adding the required Schema Registry property as a suffix to the value.converter property. For example, value.converter.schema.registry.url. Properties defined this way are passed on to the Schema Registry client used by the AvroConverter.

#### **value.converter.passthrough.enabled**

This property controls whether or not data is converted into the Kafka Connect intermediate data format before writing into an output file. Because in this example the input and output format is the same, the property is set to true, that is, data is not converted.

#### **value.converter.schema.registry.url**

The URL to Schema Registry. This is a mandatory property if the topic has records encoded in Avro format.

**topics**

List of topics to consume data from.

**output.storage**

The S3 storage implementation class.

**output.writer**

Determines the output file format. Because in this example the output format is Avro, AvroPartitionWriter is used.

**output.avro.passthrough.enabled**

This property has to match the configuration of the value.converter.passthrough.enabled property because both the input and output formats are Avro.

**Related Information**

[Amazon S3 Sink Connector Properties Reference](#)