Cloudera Runtime 7.2.14

Kafka Connect

Date published: 2020-05-21
Date modified: 2022-02-24

https://docs.cloudera.com/

Legal Notice

Contents

Kafka Connect Overview [Technical Preview]

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 [Technical Preview]

Using Kafka Connect [Technical Preview]

Securing Kafka Connect [Technical Preview]

Connectors [Technical Preview]

Streams Messaging Reference

Kafka Connect Setup [Technical Preview]

Learn how you can set up Kafka Connect in CDP Public Cloud with the Streams Messaging cluster templates.

Kafka Connect in CDP is provided in the form of a Kafka service role. The role is called Kafka Connect. In CDP Public Cloud, Kafka Connect can be deployed with the Streams Messaging Light Duty, Heavy Duty, and High Availability templates available in the Data Hub service. However, there are some differences between how Kafka Connect is deployed with each of the templates.

Light Duty

In the Streams Messaging Light Duty template, Kafka Connect roles share a host group and nodes with other service roles. Specifically, Kafka Connect is deployed in the Broker and Core_broker host groups and shares nodes with Kafka broker, ZooKeeper, and SRM Driver roles. The Core_broker is a mandatory host group in Light Duty clusters and has an instance (node) count of three. This means that by default, all Light Duty cluster you deploy will include a minimum of three Kafka Connect roles. If required, Kafka Connect can also be scaled in Light Duty clusters by scaling the Broker host group.

Heavy Duty and High Availability

In the Heavy Duty and High Availability templates, Kafka Connect roles are deployed in a dedicated host group named Connect. The instance count of this host group is set to zero by default. This means that, unless specifically provisioned, Kafka Connect roles are **not** deployed in Heavy Duty or High Availability clusters by default. The instance (node) count of the Connect host group can be configured in Data Hub during cluster provisioning. Additionally, you can add Kafka Connect to already running Heavy Duty or High Availability cluster by scaling the Connect host group.

For more information regarding the Streams Messaging cluster templates, scaling, and cluster deployment with Data Hub, see the *Related Information* section below.

Streams Messaging Manager integration

Kafka Connect deployed in CDP Public Cloud is by default configured to integrate with the Streams Messaging Manager (SMM) instance running in the cluster. This means that, if you provisioned your cluster with Kafka Connect, you will be able to manage, monitor, and deploy Kafka Connect connectors with SMM as soon as the cluster is deployed with no additional configuration required. Kafka Connect can be managed in SMM on the Connect page. For more information, see *Monitoring Kafka Connect using Streams Messaging Manager*.

Related Information

Setting up your Streams Messaging cluster

Streams Messaging cluster layout

Scaling Kafka Connect

Monitoring Kafka Connect using Streams Messaging Manager

Using Kafka Connect [Technical Preview]

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

Important: This is a technical preview feature and considered under development. Do not use this in your production systems. If you have feedback, contact Support by logging a case on our Cloudera Support Portal at https://my.cloudera.com/support.html. Technical preview features are not guaranteed troubleshooting and fixes.

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 though 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 HDFS Sink Connector developed by Cloudera and 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 HDFS Sink Connector developed by Cloudera, i to write Kafka topic data to Ozone in an unsecure cluster. Connector deployment and configuration is done using the SMM UI.

About this task

The following steps walk you through how the Cloudera-developed HDFS Sink Connector can be set up and deployed using the SMM UI.

In addition to the 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. These steps are optional.

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 displayed 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 is 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***]
```

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

4. Deploy and configure an 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.conne ct.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.HdfsSinkConne
ctor",
 "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.HdfsPartitionS
torage",
 "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 configuration 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.

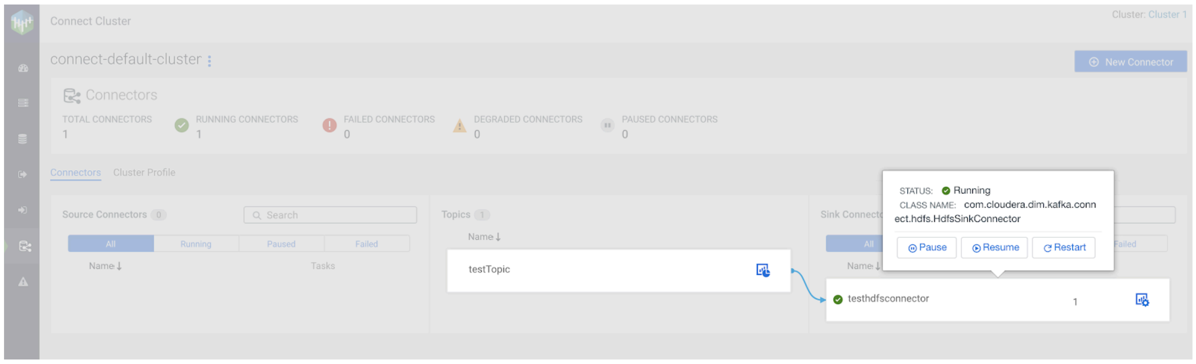
k) Click Deploy .

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

About this task

The following steps walk you through how the Cloudera-developed HDFS Sink Connector can be set up and deployed using the SMM UI.

In addition to the 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

- Ensue that a Kerberos and TLS/SSL enabled CDP PvC Base cluster with Kafka, SMM, and Ozone is set up and configured.
- 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:
   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:9093***] --replication-factor 1 --partitions 1 --topic [***T
   OPIC***] --command-config [***CLIENT CONFIG FILE***]
   ```

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

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

   c) Verify that the topic is 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***] --t
   opic [***TOPIC***] --producer.config [***CLIENT CONFIG FILE***]
   ```

   Enter test 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:9093***] --topic [***TOPIC***] --consumer.conf
   ig [***CLIENT CONFIG FILE***]
   ```

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

4. Deploy and configure an HDFS Sink Connector:
   a) In Cloudera Manager, select the Streams Messaging Manager service.
   b) 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.

   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.conne ct.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.HdfsSinkConne
ctor",
 "hdfs.uri": "o3fs://bucket1.vol1.ozone1/",
 "tasks.max": "1",
 "topics": "testTopic",
 "hdfs.kerberos.authentication": "true",
 "hdfs.kerberos.user.principal": "${cm-agent:ENV:kafka_connect_servic
e_principal}",
 "hdfs.kerberos.keytab.path": "${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.t
xt.TxtPartitionWriter",
 "value.converter": "org.apache.kafka.connect.storage.StringConverter",
 "output.storage": "com.cloudera.dim.kafka.connect.hdfs.HdfsPartitionSto
rage"
}
```

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 configuration 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.

i) Click Next.

j) Review your connector configuration.

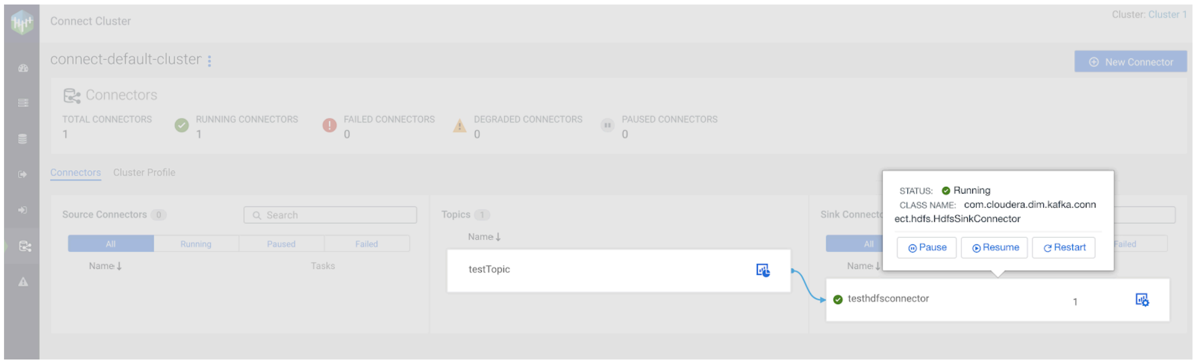
k) Click Deploy.

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 [Technical Preview]

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

Important: This is a technical preview feature and considered under development. Do not use this in your production systems. If you have feedback, contact Support by logging a case on our Cloudera Support Portal at https://my.cloudera.com/support.html. Technical preview features are not guaranteed troubleshooting and fixes.

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 ssl_enabled	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 ssl_server_keystore_location	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.
Kafka Connect TLS/SSL Server JKS Keystore File Password ssl_server_keystore_password	The password for the Kafka Connect keystore file.
Kafka Connect TLS/SSL Server JKS Keystore Key Password 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 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 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 [Technical Preview]

Cloudera Runtime comes prepackaged with a number of Cloudera developed Kafka Connect connectors. In addition, connectors that come packaged with the version of Apache Kafka that is included in Cloudera Runtime are also available for use. Manually installing and using your own custom connectors is also possible.

Important: This is a technical preview feature and considered under development. Do not use this in your production systems. If you have feedback, contact Support by logging a case on our Cloudera Support Portal at https://my.cloudera.com/support.html. Technical preview features are not guaranteed troubleshooting and fixes.

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 selectedRestart.
   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.

HTTP Source connector

The HTTP Source connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka Connect framework. Learn about the connector, its properties, and configuration.

The HTTP Source connector listens on a port for HTTP POST requests and transfers the request body to a Kafka topic. The Kafka topic this connector transfers messages to is determined by the value of the topics property. The connector does not perform record processing, the messages are transferred to Kafka as they are received in the HTTP request. The connector uses HTTPS for secure communication. Optionally, client certificate authentication (mutual TLS) can also be configured.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Source properties

These are the properties that are specific to the Stateless NiFi Source connector. All Stateless NiFi Source connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Source property reference*.

Connector/dataflow-specific properties

These properties are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *HTTP Source properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.

- Properties not marked as required must be completely removed from the configuration JSON if not set.
- The HTTP Source connector must use at least one-way SSL, it cannot be used without SSL.
- The ListenHTTP processor in the NiFi flow keeps running when the connector is paused. Due to this, the connector accepts requests, but ultimately times out and sends an error response to the client.
- The tasks.max property must be 1 to avoid port conflict due to executing multiple tasks on the same Kafka Connect node. Cloudera recommends that you use a load balancer to keep track of the actual node which runs the connector.

Configuration example

In this example, the connector accepts client requests sent to the connector’s URL. The connector's URL is based on the configuration of multiple properties. For example assume the following:

- The connector is running on kc-host-1.
- Listening Port is set to port 8080.
- Base Path is set to data-ingest.

In a case like this, the connector's URL will be the following:

```
https://kc-host-1:8080/data-ingest
```

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case and therefore can be removed from the configuration JSON.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSourceConn
ector",
 "meta.smm.predefined.flow.name": "HTTP Source",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter
",
 "tasks.max": "1",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.HTTP Source Parameters:Listening Port": "[***PORT***]",
 "parameter.HTTP Source Parameters:Base Path": "[***BASE PATH***]",
 "parameter.HTTP Source Parameters:Keystore Filename": "[***PATH TO
 KEYSTORE***]",
 "parameter.HTTP Source Parameters:Keystore Key Password": "[***KEYSTORE KEY
 PASSWORD***]",
 "parameter.HTTP Source Parameters:Keystore Password": "[***KEYSTORE
 PASSWORD***]",
 "parameter.HTTP Source Parameters:Keystore Type": "[***KEYSTORE TYPE***]"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic that the connector sends messages to.

Listening Port

The port that the client needs to use in the URL.

Base Path

The path that the client needs to use in the URL.

Keystore *

These are the properties for accessing the keystore used for HTTPS.

Related Information

Apache Kafka documentation

Stateless NiFi Source properties reference

HTTP Source properties reference

JDBC Source connector

The JDBC Source connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka Connect framework. Learn about the connector, its properties, and configuration.

The JDBC Source connector reads records from a database table and transfers each record to Kafka in Avro or JSON format. The Kafka topic this connector transfers messages to is determined by the value of the topics parameter in the configuration. The schema of the records can be inherited from the schema of the database table, or it can be a predefined schema retrieved from Schema Registry.

The connector supports incremental loading based on a table column containing increasing values (typically an ID sequence or timestamp). When the connector is started for the first time with incremental loading turned on, it can load all the existing data from the source table, or it can start from the current maximum value of the incrementing column.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Source properties

These are the properties that are specific to the Stateless NiFi Source connector. All Stateless NiFi Source connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Source property reference*.

Connector/dataflow-specific properties

These are the properties that are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *JDBC Source properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.
- Properties not marked as required must be completely removed from the configuration JSON if not set.
- The tasks.max property must be set to 1. Setting the property to any other value will not have an effect. This is because this connector performs incremental loading and must run as a single task.

Configuration example

In this example, the connector connects to a PostgreSQL database using username/password authentication, queries the records from a table, converts each record to JSON format, and sends them to a Kafka topic. The schema of the records is determined from the schema of the database table. The data loading is incremental, only new records are loaded based on a column containing incrementing values in the table.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSourceConn
ector",
 "meta.smm.predefined.flow.name": "JDBC Source",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter
",
 "tasks.max": "1",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "key.attribute": "kafka.message.key",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.JDBC Source Parameters:Database Connection URL": "[***JDBC
 URL***]",
 "parameter.JDBC Source Parameters:Database Driver Location": "[***PATH TO
 JDBC DRIVER***]",
 "parameter.JDBC Source Parameters:Database Driver Class Name": "org.post
gresql.Driver",
 "parameter.JDBC Source Parameters:Database Type": "PostgreSQL",
 "parameter.JDBC Source Parameters:Database User Name": "[***USERNAME***]",
 "parameter.JDBC Source Parameters:Database User Password":
 "[***PASSWORD***]",
 "parameter.JDBC Source Parameters:Database Table Name": "[***TABLE
 NAME***]",
 "parameter.JDBC Source Parameters:Kafka Message Key Column": "[***COLUMN
 NAME***]",
 "parameter.JDBC Source Parameters:Maximum-value Columns": "[***COLUMN
 NAME***]",
 "parameter.JDBC Source Parameters:Kafka Message Data Format": "JSON",
 "parameter.JDBC Source Parameters:Schema Access Strategy": "Inherit Schema"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic that the connector sends messages to.

Database Connection URL

The JDBC URL of the PostgreSQL database. For example, jdbc:postgresql://myhost:5432/mydb.

Database Driver Location

A comma-separated list of files or folders containing the JDBC client libraries.

Database Driver Class Name

The Java class name of the PostgreSQL Driver implementation.

Database Type

The type of the database. Because this is a PostgreSQL example, the property is set to PostgreSQL.

Database User Name

The username used for authenticating to the database.

Database User Password

The password of the user.

Database Table Name

The name of the database table to query data from.

Kafka Message Key Column

Specifies a database table column. The value of the column specified is used as the key of the Kafka message.

Maximum-value Columns

The name of the column used for incremental loading. The column value must be increasing, for example, coming from a sequence or system timestamp.

Kafka Message Data Format

The format of the messages the connector sends to Kafka.

Schema Access Strategy

Specifies the strategy used for determining the schema of the database record. In this example, this property is set to Inherit Schema, meaning that the schema is determined (inherited) from the database table.

Related Information

Apache Kafka documentation

Stateless NiFi Source properties reference

JDBC Source properties reference

JMS Source connector

The JMS Source connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka Connect framework. Learn about the connector, its properties, and configuration.

The JMS Source Connector consumes messages from a JMS broker and transfers the message body to Kafka. The Kafka topic this connector transfers messages to is determined by the value of the topics parameter in the configuration. The JMS Source connector does not perform record processing, the messages are transferred to Kafka as they are consumed from the JMS broker.

TLS can be used to establish a secure connection between the connector and the JMS broker. The keystore and truststore files necessary for securing the connection must be present on the cluster node that the connector runs on.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Source properties

These are the properties that are specific to the Stateless NiFi Source connector. All Stateless NiFi Source connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Source property reference*.

Connector/dataflow-specific properties

These properties are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *JMS Source properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.
- Properties not marked as required must be completely removed from the configuration JSON if not set.

Configuration example

In this example, the connector connects to an ActiveMQ Message Broker using a secured connection and username/ password authentication.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSourceConn
ector",
 "meta.smm.predefined.flow.name": "JMS Source",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter"
,,
 "tasks.max": "1"
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",,
 "topics": "[***TOPIC NAME***]"
 "parameter.JMS Source Parameters:JMS Broker URI": "[***BROKER URI***]",
 "parameter.JMS Source Parameters:JMS Client Libraries": "[***PATH TO CLIENT
 LIBRARIES***]",
 "parameter.JMS Source Parameters:JMS Connection Factory Class Name": "or
g.apache.activemq.ActiveMQSslConnectionFactory",
 "parameter.JMS Source Parameters:JMS Destination Name": "[***QUEUE
 NAME***]",
 "parameter.JMS Source Parameters:JMS Destination Type": "QUEUE",
 "parameter.JMS Source Parameters:JMS User Name": "[***USERNAME***]",
 "parameter.JMS Source Parameters:JMS User Password": "[***PASSWORD***]",
 "parameter.JMS Source Parameters:Truststore Filename": "[***PATH TO
 TRUSTSTORE***]",
 "parameter.JMS Source Parameters:Truststore Password": "[***TRUSTSTORE
 PASSWORD***]",
 "parameter.JMS Source Parameters:Truststore Type": "[***TRUSTSTORE
 TYPE***]"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic that the connector sends messages to.

JMS Broker URI

The URI of the ActiveMQ Message Broker. For example, ssl://activemq-hostname:61617.

JMS Client Libraries

The path to the directory containing ActiveMQ client libraries.

JMS Connection Factory Class Name

The Java class name of the ActiveMQ ConnectionFactory implementation.

JMS Destination Name

The name of the destination (in this case a queue) on the ActiveMQ Message Broker that the messages are received from.

JMS Destination Type

The type of the destination. In this example, the type is QUEUE.

JMS User Name

The username used for authenticating to the ActiveMQ Message Broker.

JMS User Password

The password of the user.

Truststore *

These are the properties for accessing the truststore containing the ActiveMQ Message Broker's certificate used for secure communication.

Related Information

Apache Kafka documentation

Stateless NiFi Source properties reference

JMS Source properties reference

MQTT Source connector

The MQTT Source connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka Connect framework. Learn about the connector, its properties, and configuration.

The MQTT Source connector consumes messages from an MQTT broker and transfers them to Kafka. The Kafka topic this connector transfers messages to is determined by the value of the topics property in the configuration. This connector does not perform record processing. The messages are transferred to Kafka as they are consumed from the MQTT broker. TLS can be used to establish a secure connection between the connector and the MQTT broker. The keystore and truststore files necessary for securing the connection must be present on the cluster node that the connector runs on.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Source properties

These are the properties that are specific to the Stateless NiFi Source connector. All Stateless NiFi Source connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Source property reference*.

Connector/dataflow-specific properties

These properties are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *MQTT Source properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.
- Properties not marked as required must be completely removed from the configuration JSON if not set.
- If the keystore-related properties are removed and an empty truststore is provided, the connector does not use TLS for connecting to the MQTT broker. TLS is used if a truststore is provided that has a key in it.

Configuration example

In this example, the connector fetches messages from an MQTT broker and transfers them to a Kafka topic.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSourceConn
ector",
 "meta.smm.predefined.flow.name": "MQTT Source",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter
",
 "tasks.max": "1",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.MQTT Source Parameters:MQTT Broker URI": "tcp
://[***HOST***]:[***PORT***]",
 "parameter.MQTT Source Parameters:MQTT Quality of Service": "0",
 "parameter.MQTT Source Parameters:MQTT Topics": "[***MQTT TOPIC NAME***]"
}
```

The following list collects the properties from the configuration example that must be customized for this use case:

topics

The name of the Kafka topic that the connector sends messages to.

MQTT Broker URI

The URI of the MQTT broker. In this example, the connection is not secure. As a result, the URI starts with tcp://. The port number you typically use in a case like this is 1883. If TLS is used, the URI should start with ssl://, and typically you use port number 88883.

MQTT Topics

The comma-separated list of MQTT topics that the connector fetches messages from.

Related Information

Apache Kafka documentation

Stateless NiFi Source properties reference

MQTT Source properties reference

SFTP Source connector

The SFTP Source connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka Connect framework. Learn about the connector, its properties, and configuration.

The SFTP Source connector obtains files from an SFTP server and transfers them to Kafka. The SFTP Source connector supports record processing. In case record processing is enabled, input file is split into records and the records are transferred to Kafka. If record processing is not enabled, the connector forwards files to Kafka as they are fetched from the SFTP server.

If record processing is enabled, the input file can contain records in either JSON, CSV or Grok format. If the input records are in JSON or CSV format, the connector can either infer the schema of the records based on the data or read the schema from Schema Registry. If the input records are in Grok format, the connector can either derive the schema using the field names from the value of the Grok Expression property or read the schema from Schema Registry.

In addition, if record processing is enabled, the connector writes the records to Kafka in Avro format. The record schema gets embedded in these Avro messages if the schema is set to be either inferred or determined based on the Grok Expression property. If Schema Registry is used for getting the schema of the records, then the schema does not get embedded in the Avro messages.

If Schema Registry is used, and it is on a Kerberized cluster, the krb5.file property must point to the krb5.conf file that provides access to the cluster on which Schema Registry is present. This means that the krb5.conf file must be on the same cluster node that the connector runs on. The Kerberos keytab used to access Schema Registry must also be on the same cluster node that the connector runs on. The connection to Schema Registry can be secured by TLS. The truststore file necessary for securing the connection must also be on the same cluster node that the connector runs on.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Source properties

These are the properties that are specific to the Stateless NiFi Source connector. All Stateless NiFi Source connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Source property reference*.

Connector/dataflow-specific properties

These properties are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *SFTP Source properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.
- Properties not marked as required must be completely removed from the configuration JSON if not set.
- The Schema Registry URL property is mandatory even if Schema Registry is not used. If Schema Registry is not used, use the default value, or completely remove the property from the configuration JSON.
- The value of the Schema Access Strategy property is not independent of the value of the Input Data Format property. As a result, you must exercise caution when configuring Schema Access Strategy.
  - If the value of Input Data Format is JSON, the possible values for Schema Access Strategy are Schema R egistry or Infer Schema.
  - If the value of Input Data Format is CSV, the possible values for Schema Access Strategy are Schema Regis try or Infer Schema.
  - If the value of Input Data Format is GROK, the possible values for Schema Access Strategy are Schema R egistry or Field Names From Grok Expression.
- If the Enable Record Processing property is set to false (record processing disabled), the entire input file is transferred to Kafka as one message.
- If the Enable Record Processing property is set to true (record processing enabled), the output is always in Avro format. However, the contents of the output message transferred to Kafka depends on the value of the Schema A ccess Strategy property.
  - If Schema Access Strategy is set to Schema Registry, the output Avro message does not contain the schema (the schema is not embedded in the output).
  - If Schema Access Strategy is set to either Infer Schema or Field Names From Grok Expression, the output Avro message contains the schema (the schema is embedded in the output).

Configuration example

In this example, the connector fetches files from an SFTP server that only requires Basic Authentication (username and password) and transfers these files to a Kafka topic.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSourceConn
ector",
 "meta.smm.predefined.flow.name": "SFTP Source",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter
",
 "tasks.max": "1",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "topics": "[*** KAFKA TOPIC NAME***]",
 "parameter.SFTP Source Parameters:Hostname": "[***SFTP Server
 Hostname***]",
 "parameter.SFTP Source Parameters:Port": "22",
 "parameter.SFTP Source Parameters:Username": "[***SFTP USERNAME***]",
 "parameter.SFTP Source Parameters:Password": "[***SFTP PASSWORD***]",
 "parameter.SFTP Source Parameters:Remote Path": "[***FOLDER PATH***]",
 "parameter.SFTP Source Parameters:File Filter Regex": ".*",
 "parameter.SFTP Source Parameters:Enable Record Processing": "false"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic that the connector sends messages to.

Hostname

The hostname of the remote system where the SFTP server runs. For example, my.sftp-server.com.

Port

The port that the remote system is listening on for file transfers.

Username

The username for connecting to the SFTP server.

Password

The password used to authenticate the user towards the SFTP server. In this example, the SFTP server requires Basic Authentication.

Remote Path

The path on the remote system that points to the directory from which to pull files. For example, /uploads.

File Filter Regex

The Java regular expression to use for filtering filenames. Only files whose names match the regular expression are fetched.

Enable Record Processing

Determines whether the contents of a file get parsed as records before sending them to Kafka. In this example, the property is set to false, meaning that the entire file gets forwarded to Kafka as one message.

Related Information

Apache Kafka documentation

Stateless NiFi Source properties reference

SFTP Source properties reference

Stateless NiFi Source and Sink

The Stateless NiFi Source and Sink connectors allow you to run NiFi dataflows within Kafka Connect. Using these connectors can grant you access to a number of NiFi features without having the need to deploy or maintain NiFi on your cluster.

Apache NiFi is a powerful tool for authoring and running dataflows. It provides many capabilities that are necessary for large-scale enterprise deployments, such as data persistence and resilience, data lineage and traceability, as well as multi-tenancy. This, however, requires an administrator who ensures that this process is running and operational. Additionally, in most cases, adding more capabilities to your deployments results in more complexity.

There are times, however, when users do not need all the power of NiFi, and running it in a much simpler form factor is sufficient. A common use case is to use NiFi to pull data from many different sources, perform manipulations (for example, convert JSON to Avro), filter some records, and then publish the data to Apache Kafka. Another common use case is to pull data from Apache Kafka, perform manipulations and filtering, and then publish the data elsewhere.

For deployments where NiFi acts only as a bridge into and out of Kafka, it can be simpler to operationalize such a deployment by running the dataflow within Kafka Connect. The Stateless NiFi Source and Sink connectors allow users to do just that.

Stateless NiFi

Dataflows within Kafka Connect run using the Stateless NiFi dataflow engine. Stateless NiFi differs from the traditional NiFi engine in the following ways:

- Stateless NiFi is an engine that is designed to be embedded. This makes it convenient to run it within the Kafka Connect framework.
- Stateless NiFi does not provide a user interface (UI) or a REST API. Additionally, it does not support modifying the dataflow while it is running.
- Stateless NiFi does not persist FlowFile content to disk. Instead, it holds the content in memory.
- Stateless NiFi does not use data prioritizers. Instead, it operates on data in a First-In-First-Out order. Dataflows built for Stateless NiFi must have a single source and a single destination. The only exception to this is when the data is routed to exactly one of multiple destinations, such as a Failure destination or a Success destination.
- Stateless NiFi does not currently provide access to data lineage/provenance.
- Stateless NiFi does not support cyclic graphs. While it is common and desirable in traditional NiFi to have a failure relationship from a Processor route back to the same Processor, this can result in a StackOverflowException in Stateless NiFi. The preferred approach in Stateless NiFi is to create an Output Port for failures and route the data to that Output Port.

Dataflow development best practices for Stateless NiFi

By leveraging Stateless NiFi, you can build your own dataflows and use those as Kafka Connect connectors. However, there are some principles to keep in mind when building such a dataflow in NiFi.

General recommendations and criteria

- If your dataflow depends on external resources (like a JDBC driver), the resource must be made available on all Kafka Connect workers. Additionally, the resource must be in the same location on all nodes with proper file permissions. Cloudera also recommends parameterizing the file path so that you can specify it when deploying the connector.
- The dataflow should be designed in its dedicated Process Group and should have a Parameter Context assigned to its Process Group. This way you have the ability to parameterize your connector which enables you to provide custom property values when deploying the connector using SMM.
- A source connector flow should always have a single source. This means that the flow should have only one Processor with no incoming connections.
- A sink connector flow should always have a single destination. If you want to transfer the data consumed from the Kafka topic to two different destinations at the same time, it is preferred to have two distinct sink connectors with different consumer groups. If using a single sink connector, you must take into consideration the possibility of creating duplicates. If one destination is working as expected but not the other, the Kafka message is not acknowledged and consumed again at a later time while the data was still successfully sent to one destination.

Source connectors

A Stateless NiFi Source connector is responsible for obtaining data from one source and delivering that data to Kafka. Any source connector flow you build should not attempt to deliver directly to Kafka itself using a Processor such as PublishKafka. Instead, the data should be routed to an Output Port. Any FlowFile delivered to that Output Port is obtained by the connector and delivered to Kafka.

Each FlowFile is delivered as a single Kafka record. Therefore, if a FlowFile contains thousands of JSON records, totaling 5 MB for example, it is important to split those FlowFiles into individual records using a SplitRecord processor before transferring the data to the Output Port. Otherwise, this results in a single Kafka record that is 5 MB while the default record size for Kafka is 1 MB.

Sink connectors

A Stateless NiFi Sink connector is responsible for obtaining data from Kafka and delivering that data to some other service. Any sink connector flow you build should not attempt to source data directly from Kafka using a Processor such as ConsumeKafka. Instead, the data should be received from an Input Port. Each Kafka record is enqueued into the outbound connection of the Input Port so that the next processor in the flow is able to process it.

Each Kafka record FlowFile is delivered as a single FlowFile. Depending on the destination, Cloudera recommends that you consider merging your Kafka records together into bigger chunks before delivering them. For example, assume you are delivering data to HDFS. Because HDFS is not built for accessing and handling many small files, it is not recommended to send each individual Kafka message to HDFS as a separate file.

Considerations for listen-type source connectors

A listen-type source connector consists of a NiFi dataflow in which the first Processor is a ListenX processor. These Processors listen on specific interfaces and specific ports on the host where they are running. For example, a flow starting with a ListenHTTP Processor listens on a specific port for HTTP requests made by external clients. There are a number of limitations that apply to listen-type connectors that you should be aware of.

The Kafka Connect framework does not provide the concept of task affinity mapping, meaning that if a connector is deployed with multiple tasks, it is not possible to ensure that tasks will be on different Kafka Connect workers. Two tasks started on the same worker in a listen-type connector results in a failure. This happens because both connectors try to bind to the same port. For this reason, it is required that all listen-type source connectors are deployed with a single task.

A common deployment model is to have a load balancer in front of the Kafka Connect workers and a mapping rule linking to the port on each Kafka Connect worker. This way, the load balancer FQDN can be provided to the client without assuming which worker the task is going to be deployed on.

If multiple tasks are required for performance, you can opt to have multiple standalone Kafka Connect workers and deploy the connector independently on each worker. However, in such a case, Cloudera recommends using Cloudera Flow Management and NiFi as they provide a significantly more powerful environment for this type of use cases.

Merging considerations for sink connectors

NiFi supports many Processors that can be used as sinks for Apache Kafka data. The data from Kafka can often be delivered directly to a sink processor if the destination service operates well with many small messages. For example, a PublishJMS Processor can easily handle many small messages. However, other services, like S3 or HDFS, perform much better if the data is first batched or merged together. For this reason, the MergeContent and MergeRecord Processors are extremely popular in NiFi. They allow many small FlowFiles to be merged together into one larger FlowFile.

With traditional NiFi, you can set a minimum and maximum size for the merged data along with a timeout. However, with Stateless NiFi and Kafka Connect, this might not work. This is because only a limited number of FlowFiles are made available to the Processor. You can still use these Processors to merge the data together, but some limitations apply.

If MergeContent or MergeRecord is triggered but not enough FlowFiles are present to create a batch, the Processor does nothing. Therefore, Cloudera recommends that you set Minimum Number of Entries to 1 and Minimum Group Size to 0 B (these are default values). With these settings, the MergeContent and MergeRecord Processors create a merged FlowFile from the input FlowFiles available in the given execution of the connector and do not wait for more data to arrive. Max Bin Age does not need to be configured in this case.

The Maximum Number of Entries and Maximum Group Size processor properties can also be used to limit the maximum size of the merged FlowFile. In a case like this, the Processor might create multiple merged FlowFiles per connector execution.The last FlowFile will contain the leftover items. Because of this, the last file might be smaller than the maximum size.

Additionally, you can configure Kafka Connect properties such as offset.flush.timeout.ms to control the amount of data that is provided by the Kafka Connect framework for a single execution of the connector.

Dataflow execution and scheduling

Stateless NiFi does not consider the scheduling settings of processors. When you deploy a connector, Kafka Connect starts a Stateless NiFi engine. The Stateless NiFi engine then runs the dataflow.

For source connectors, the dataflow is triggered continuously. This means that every time the dataflow finishes running, it is triggered to run again.

For sink connectors, messages are continuously enqueued to the input port of the dataflow. The dataflow is then triggered based on an interval to run as many times as necessary to consume all the messages enqueued to the input port. The trigger interval is determined by the Kafka Connect role's Offset Flush Interval (offset.flush.interval.ms) property. Once all messages are consumed from the input port, the dataflow stops running and Kafka Connect resumes enqueuing messages on the dataflow's input port. The next set of enqueued messages are processed after the next trigger interval is reached.

Developing a dataflow for Stateless NiFi

Learn about the recommended process of building a dataflow that you can deploy with the Stateless NiFi Sink or Source connectors. This process involves building and designing a parameterized dataflow within a Process Group and then downloading the dataflow as a flow definition.

About this task

The general steps for building a dataflow are identical for both source and sink connector flows. For ease of understanding a dataflow example for a simple MQTT Source and MQTT Sink connector is provided without going into details (what processors to use, what parameters and properties to set, what relationships to define and so on).

Before you begin

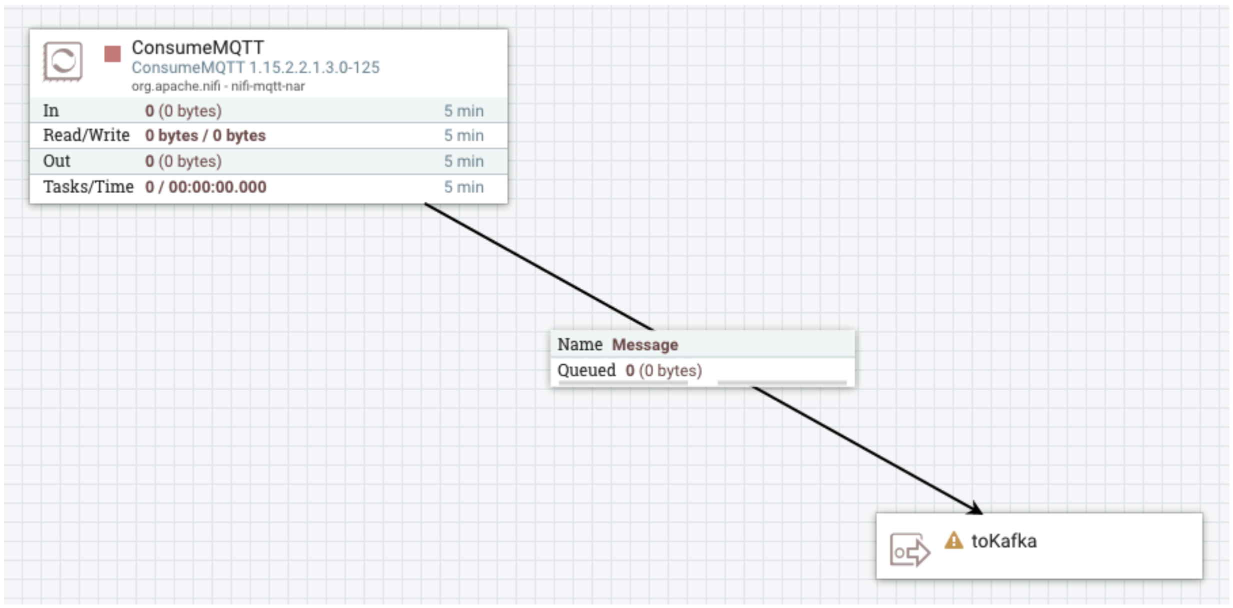
- Ensure that you reviewed Dataflow development best practices for Stateless NiFi.
- You have access to a running instance of NiFi.

Procedure

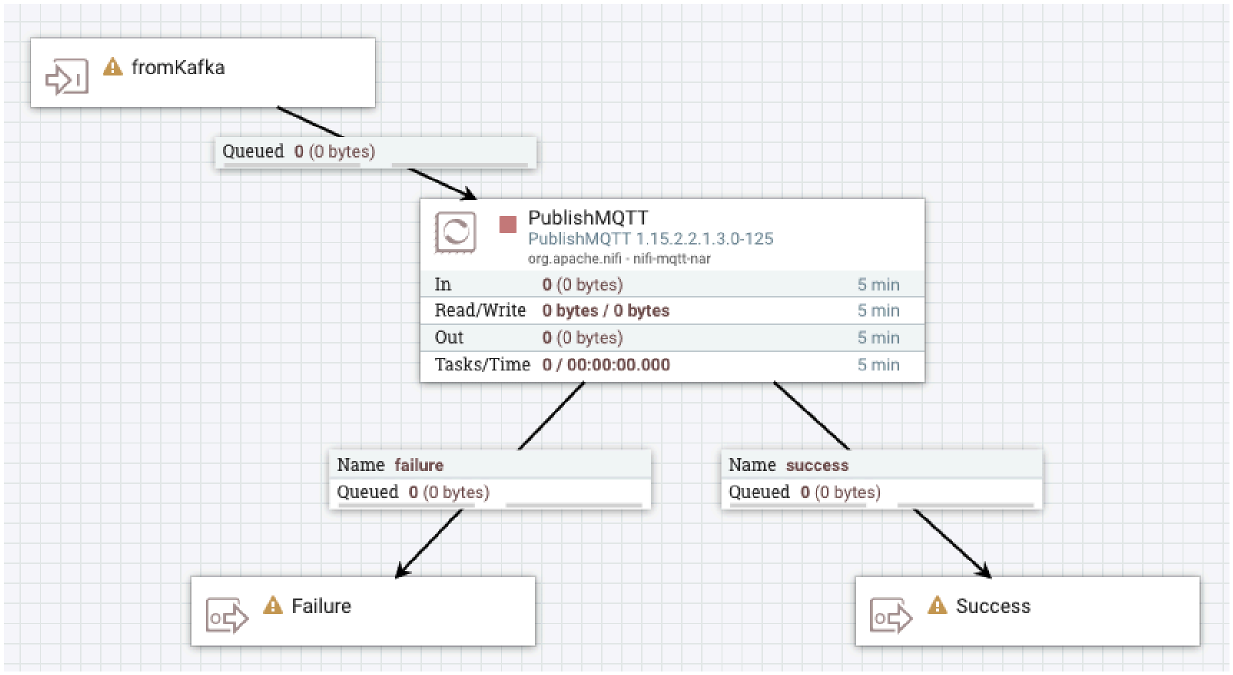
1. Access the NiFi UI.
2. Create a new Process Group.
3. Right-click the Process Group and select Configure.
4. Create a new Parameter Context and assign it to the process group:
   a) Select Process Group Parameter ContextCreate new parameter context...
   b) Enter a name for the Parameter Context.
   c) Click Apply.
5. Click Apply and then click OK.
6. Close the configuration dialog.
7. Double-click the Process Group you created.

8. Design and parameterize your dataflow.

For example, you can build a dataflow that you deploy as a source connector. The following is an example dataflow for a simple MQTT source connector. It consists of a ConsumeMQTT Processor and an output port (representing a Kafka topic).

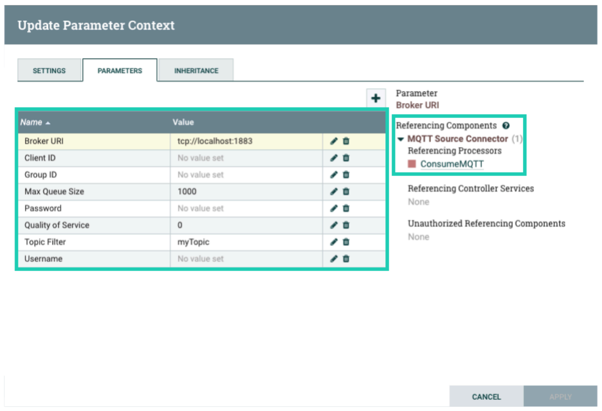

ConsumeMQTT
ConsumeMQTT 1.15.2.2.1.3.0-125
org.apache.nifi - nifi-mqtt-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name Message
Queued 0 (0 bytes)
toKafka

Alternatively, you can also build a dataflow to deploy as a sink connector. The following is an example dataflow for a simple MQTT sink connector. It consists of an input port (representing a Kafka topic), a PublishMQTT Processor, and two output ports.



In the case of this example, a Failure output port is required. This is done so that in case the destination is not available, the session can be rolled back and the Kafka message can be declined. The Failure port is specified in the configuration of the connector when deploying it through SMM.

When designing and building your dataflow, ensure that you update your Parameter Context and reference the parameters in your components. For example:



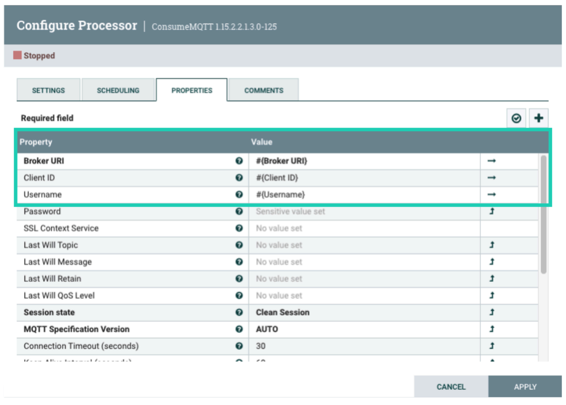
Name	Value
Broker URI	tcp://localhost:1883
Client ID	No value set
Group ID	No value set
Max Queue Size	1000
Password	No value set
Quality of Service	0
Topic Filter	myTopic
Username	No value set

Parameter
Broker URI

Referencing Components
MQTT Source Connector (1)
Referencing Processors
ConsumeMQTT

Referencing Controller Services
None

Unauthorized Referencing Components
None



Property	Value
Broker URI	#{Broker URI}
Client ID	#{Client ID}
Username	#{Username}
Password	Sensitive value set
SSL Context Service	No value set
Last Will Topic	No value set
Last Will Message	No value set
Last Will Retain	No value set
Last Will QoS Level	No value set
Session state	Clean Session
MQTT Specification Version	AUTO
Connection Timeout (seconds)	30

For more information on building dataflows, parameters, and referencing parameters see the *Flow Management* library.

9. Exit the Process Group.
10. Right-click the Process Group and select Download flow definition.

Results

The dataflow is downloaded as a flow definition (JSON file).

What to do next

Deploy the dataflow as a Kafka Connect connector using the Stateless NiFi Source or Sink connectors. Continue with *Deploying a dataflow using stateless NiFi*.

Related Information

Deploying a dataflow using Stateless NiFi

Flow Management

Deploying a dataflow using Stateless NiFi

A custom developed NiFi dataflow can be deployed with the Stateless NiFi Source or Sink connectors using Streams Messaging Manager (SMM).

About this task

After building and downloading your dataflow, you can deploy it in Kafka Connect as a source or sink connector. This can be done using SMM. The following list of steps walk you through the process of deploying a Stateless NiFi Sink or Source connector and provide examples on how dataflows are configured using SMM.

These steps are for the SMM UI, however, the actions described here can also be completed using the SMM REST API. For more information, see *Cloudera Streams Messaging Manager REST API Reference*.

Before you begin

- A CDP cluster containing SMM and Kafka is available.
- You have access to the SMM UI.
- The flow definition of the dataflow that you want to deploy is available to you. For more information on how to design, build, and export a dataflow using the NiFi UI, see Developing a dataflow for Stateless NiFi .

Procedure

1. Access the SMM UI.

2. Click (Connect) on the navigation sidebar.
3. Click + New Connector to add a new connector.
4. Select a connector.

   Which connector you select depends on the type of dataflow you want to run. Select StatelessNiFiSourceConnector found on the **Source Templates** tab if your dataflow collects (sources) data from systems and publishes it to Kafka. Select the StatelessNiFiSinkConnector from the Sink **Templates tab** if your dataflow delivers (sinks) Kafka data into other systems.
5. Enter a name for the connector in the Enter Name field.

   Ensure that you add a unique and easily identifiable name. The name of the connector cannot be changed once the connector is deployed.
6. Configure the connector

   Connector configuration can be broken into three steps. First, you configure the properties available in the default configuration template. Second, you configure other connector properties. Finally, you specify the flow parameters specific to your dataflow. These are the parameters that you set up in the Parameter Context when you built the flow in NiFi. Even though the following breaks the process up into multiple substeps, all properties are configured using the same input field in the SMM UI.

   a) Configure the properties available in the default template.

   The following properties from the default template are the ones that you should configure. Other properties in the template that are not highlighted here have working default values configured. Configuring the properties that are present in the template but not highlighted in the following list is not recommended by Cloudera.

   flow.snapshot

   Specifies the dataflow to run. Copy and paste the contents of the flow definition JSON you exported from NiFi. The value of this property must be a JSON object containing the full dataflow.

   input.port

   The name of the Input Port in the NiFi dataflow that Kafka records are sent to. If the dataflow contains exactly one Input Port, this property is optional and can be omitted. However, if the dataflow contains multiple Input Ports, this property must be specified. This property is specific to the Stateless NiFi Sink connector.

   output.port

   The name of the Output Port in the NiFi dataflow that is the source of records for the Kafka topic. If the dataflow contains exactly one port, this can be omitted. However, if the dataflow contains multiple ports (for example, a Success and a Failure port), this property must be specified. If any FlowFile is sent to any port other than the specified Port, it is considered as a failure. The session is rolled back and no data is collected. This property is specific to the Stateless NiFi Source connector.

   topics

   The name of the topic to deliver data to or fetch data from.

   krb5.file

   Specifies the krb5.conf file to use if the dataflow interacts with any services that are secured using Kerberos. This property is optional and is only required if a connection is established with a Kerberized service. Use the default value if an appropriate krb5.conf is located in /etc.

   b) Add other connector properties.

   There are many other properties that you can set. These include the properties specific to the Stateless NiFi connectors as well as the Kafka Connect properties that come from the base framework. Which of these you require depends on your dataflow and use case.

   Cloudera recommends that at minimum you configure the converters used for key and value conversion. What converters you use depends on the dataflow however, the following configuration is appropriate in the majority of cases:

```
"key.converter": "org.apache.kafka.connect.storage.StringConverter"
```

```
"value.converter": "org.apache.kafka.connect.converters.ByteArrayConvert
er"
```

A comprehensive list of all other properties that the connectors accept can be found in *Stateless NiFi Source connector reference*, *Stateless NiFi Sink Connector reference*, and the *Apache Kafka documentation*.

c) Configure dataflow-specific parameters.

If you followed the recommendations provided in *Dataflow development best practices for Stateless NiFi* and *Developing a dataflow for Stateless NiFi*, your dataflow is parameterized. The parameters that you defined within the dataflow can be assigned values using the configuration pane in SMM. This is done by adding a configuration entry for each of the dataflow parameters that you want to configure. Dataflow parameters can be added to the configuration as follows:

```
"parameter.[***PARAMETER NAME***]": "[***VALUE***]"
```

For example, assume that you have a the following configuration entry:

```
"parameter.Directory": "/mydir"
```

In a case like this, any Parameter Context in the dataflow that has a parameter named Directory gets the specified value (/mydir). If the dataflow has child Process Groups, and those child Process Groups have their own Parameter Contexts, the value is used for all Parameter Contexts that contain a parameter named Dire ctory.

You can also apply a parameter to a specific Parameter Context. This is done by prefixing the parameter name with the name of the Parameter Context followed by a colon.

```
"parameter.[***PARAMETER CONTEXT NAME***]:[***PARAMETER NAME***]":
 "[***VALUE***]"
```

For example, assume you have the following configuration entry:

```
"parameter.My Context:Directory": "/mydir"
```

In a case like this, only the Parameter Context called My Context gets the specified value for the Directory parameter.

Once all properties are configured, your configuration should look similar to the following example:

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSource
Connector",
 "flow.snapshot": "[***FLOW DEFINITION JSON***]",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConve
rter",
 "tasks.max": "1",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.MQTT Source Parameters:MQTT Broker URI": "tcp
://[***HOST***]:[***PORT***]",
 "parameter.MQTT Source Parameters:MQTT Quality of Service": "0",
 "parameter.MQTT Source Parameters:MQTT Topics": "[***MQTT TOPIC
 NAME***]"
}
```

This example is for a custom built MQTT source connector. The example uses additional Connect properties such as key.converter and value.converter. The example also specifies a number of flow parameters such as MQTT Broker URI and MQTT Topics.

7. 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.
8. Click Next.
9. Review your connector configuration.
10. Click Deploy.

Related Information

Cloudera Streams Messaging Manager REST API Reference

Stateless NiFi Source properties reference

Stateless NiFi Sink properties reference

Apache Kafka documentation

Dataflow development best practices for Stateless NiFi

Developing a dataflow for Stateless NiFi

Syslog TCP Source connector

The Syslog TCP Source connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka Connect framework. Learn about the connector, its properties, and configuration.

The Syslog TCP Source connector listens on a port for syslog messages over TCP and transfers them to Kafka. The connector accepts messages in one of the following formats: Syslog 3164, Syslog 5424, or Grok. If the input messages are in Grok format, the connector can either derive the schema using the field names from the value of the Grok Expression property or read the schema from Schema Registry.

The connector can write messages into Kafka in one of the following formats: Avro, JSON, or text. If the output format is text, the raw message is transferred to Kafka. If the output format is JSON, the input message is processed and converted into JSON format before it is transferred to Kafka. If the output format is Avro, the input message is processed and converted into Avro format before it is transferred to Kafka. The record schema is embedded into Avro messages if the schema is not fetched from Schema Registry. If Schema Registry is used for getting the schema of the records, then the schema does not get embedded in the Avro messages.

If Schema Registry is used, and it is on a Kerberized cluster, the krb5.file property must point to the krb5.conf file that provides access to the cluster on which Schema Registry is present. This means that the krb5.conf file must be on the same cluster node that the connector runs on. The Kerberos keytab that is used to access Schema Registry must also be on the same cluster node that the connector runs on.

The connection to Schema Registry can be secured by TLS. The truststore file necessary for securing the connection must be on the same cluster node that the connector runs on. Mutual TLS for securing the communication between the message sender and the connector itself is also supported. The keystore and truststore files necessary for securing the connection must be on the same cluster node that the connector runs on.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Source properties

These are the properties that are specific to the Stateless NiFi Source connector. All Stateless NiFi Source connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Source property reference*.

Connector/dataflow-specific properties

These properties are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *Syslog TCP Source properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.
- Properties not marked as required must be completely removed from the configuration JSON if not set.
- The Syslog TCP Source connector must use at least one-way SSL. It cannot be used without SSL.
- The Schema Registry URL property is mandatory even if Schema Registry is not used. If Schema Registry is not used, use the default value, or completely remove the property from the configuration JSON.
- Schemas are only read from Schema Registry if Input Data Format is set to GROK and Schema Access Stra tegy is set to Schema Registry.
- If Output Format is AVRO, the schema of the records can be embedded in the output data. Whether the schema is embedded is determined by the Schema Access Strategy property. If this property is set to Schema Registry, the schema is not embedded in the output messages. The schema is embedded in all other cases.

Configuration example

In this example, the connector uses mutual TLS to receive data in Syslog 3164 format which is then transferred to Kafka in JSON format.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSourceConn
ector",
 "meta.smm.predefined.flow.name": "Syslog TCP Source",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConvert
er",
 "tasks.max": "1",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.Syslog TCP Source Parameters:Port": "[***PORT***]",
 "parameter.Syslog TCP Source Parameters:Input Data Format": "Syslog 3164",
 "parameter.Syslog TCP Source Parameters:Output Format": "JSON",
 "parameter.Syslog TCP Source Parameters:Client Authentication": "REQUIRED",
 "parameter.Syslog TCP Source Parameters:SSL Keystore Filename": "[***THE
 FULLY-QUALIFIED FILENAME OF THE KEYSTORE***]",
 "parameter.Syslog TCP Source Parameters:SSL Keystore Key Password":
 "[***KEYSTORE KEY PASSWORD***]",
 "parameter.Syslog TCP Source Parameters:SSL Keystore Password":
 "[***KEYSTORE PASSWORD***]",
 "parameter.Syslog TCP Source Parameters:SSL Keystore Type": "[***KEYSTORE
 TYPE***]",
 "parameter.Syslog TCP Source Parameters:SSL Truststore Filename": "[***THE
 FULLY-QUALIFIED FILENAME OF THE TRUSTSTORE***]",
 "parameter.Syslog TCP Source Parameters:SSL Truststore Password":
 "[***TRUSTSTORE PASSWORD***]",
 "parameter.Syslog TCP Source Parameters:SSL Truststore Type":
 "[***TRUSTSTORE TYPE***]"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic that the connector sends messages to.

Port

The port that the connector listens on for incoming messages.

Input Data Format

Determines what format incoming messages are expected in.

Output Format

Determines the format in which messages are transferred to Kafka.

Client Authentication

Determines if one-way or two-way SSL is used. In this example, this property is set to REQUIRED, meaning that two-way SSL is used.

Keystore *

These are the properties for accessing the keystore containing the keypair used for secure communication.

Truststore *

These are the parameters for accessing the truststore containing the message sender's certificate used for secure communication.

Related Information

Apache Kafka documentation

Stateless NiFi Source properties reference

Syslog TCP Source properties reference

Syslog UDP Source connector

The Syslog UDP Source connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka Connect framework. Learn about the connector, its properties, and configuration.

The Syslog UDP Source connector listens on a port for syslog messages over UDP and transfers them to Kafka. The connector accepts messages in one of the following formats: Syslog 3164, Syslog 5424, or Grok. If the input messages are in Grok format, the connector can either derive the schema using the field names from the value of the Grok Expression property or read the schema from Schema Registry.

The connector can write messages into Kafka in one of the following formats: Avro, JSON, or text. If the output format is text, the raw message is transferred to Kafka. If the output format is JSON, the input message is processed and converted into JSON format before it is transferred to Kafka. If the output format is Avro, the input message is processed and converted into Avro format before it is transferred to Kafka. The record schema is embedded into Avro messages if the schema is not fetched from Schema Registry. If Schema Registry is used for getting the schema of the records, then the schema does not get embedded in the Avro messages.

If Schema Registry is used, and it is on a Kerberized cluster, the krb5.file property must point to the krb5.conf file that provides access to the cluster on which Schema Registry is present. This means that the krb5.conf file must be on the same cluster node that the connector runs on.

The Kerberos keytab that is used to access Schema Registry must also be on the same cluster node that the connector runs on. The connection to Schema Registry can be secured by TLS. The truststore file necessary for securing the connection must also be on the same cluster node that the connector runs on.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Source properties

These are the properties that are specific to the Stateless NiFi Source connector. All Stateless NiFi Source connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Source property reference*.

Connector/dataflow-specific properties

These properties are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *Syslog UDP Source properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.
- Properties not marked as required must be completely removed from the configuration JSON if not set.
- The Schema Registry URL property is mandatory even if Schema Registry is not used. If Schema Registry is not used, use the default value, or completely remove the property from the configuration JSON.
- Schemas are only read from Schema Registry if Input Data Format is set to GROK and Schema Access Stra tegy is set to Schema Registry.
- If Output Format is AVRO, the schema of the records can be embedded in the output data. Whether the schema is embedded is determined by the Schema Access Strategy property. If this property is set to Schema Registry, the schema is not embedded in the output messages. The schema is embedded in all other cases.

Configuration example

In this example, the connector receives data in Syslog 3164 format and transfers it to Kafka in JSON format.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSourceConn
ector",
 "meta.smm.predefined.flow.name": "Syslog UDP Source",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConvert
er",
 "tasks.max": "1",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.Syslog UDP Source Parameters:Port": "[***PORT***]",
 "parameter.Syslog UDP Source Parameters:Input Data Format": "Syslog 3164",
 "parameter.Syslog UDP Source Parameters:Output Format": "JSON",
 "parameter.Syslog UDP Source Parameters:Output Grouping for JSON": "output-
oneline"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic that the connector sends messages to.

Port

The port that the connector listens on for incoming messages.

Input Data Format

Determines what format incoming messages are expected in.

Output Format

Determines the format in which messages are transferred to Kafka.

Related Information

Apache Kafka documentation

Stateless NiFi Source properties reference

Syslog UDP Source properties reference

ADLS Sink connector

The ADLS Sink connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka connect framework. Learn about the connector, its properties, and configuration.

The ADLS Sink connector fetches messages from Kafka and uploads them to ADLS. The topic this connector receives messages from is determined by the value of the topics property in the configuration. The messages can contain unstructured (character or binary) data or they can be in Avro or JSON format.

If the input is unstructured data, record processing is disabled. In a case like this, multiple messages can be concatenated into a single output file on ADLS using a demarcator (for example, newline for text messages). Merging is optional, large binary data can be forwarded to Azure at a 1:1 ratio, one Kafka message equals a single ADLS file.

If the input is either Avro or JSON, record processing is enabled. In a case like this, the schema of the records can be a predefined schema retrieved from Schema Registry (for both Avro and JSON data), it can be embedded in the Avro data, or inferred from the JSON data. The output can be Avro, JSON or CSV. Multiple records are typically merged into a single output file before uploading to ADLS.

The connector can authenticate to Azure using an Account Key, SAS Token, Service Principal, or a Managed Identity.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Sink properties

These are the properties that are specific to the Stateless NiFi Sink connector. All Stateless NiFi Sink connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Sink property reference*.

Connector/dataflow-specific properties

These properties are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *ADLS Sink properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.
- Properties not marked as required must be completely removed from the configuration JSON if not set.

Configuration example for fetching unstructured data

In this example, the connector fetches unstructured messages containing single line character data from Kafka. The connector concatenates the messages using newline characters as the demarcator. The files that the connector creates and uploads to ADLS are maximum 10 MB in size. The files are named according to the following pattern: mess ages_*[***UUID***]*. The connector authenticates to Azure using an Account Key.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSinkConnec
tor",
 "meta.smm.predefined.flow.name": "ADLS Sink",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter
",
 "tasks.max": "1",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "failure.ports": "PutAzureDataLakeStorage Failure",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.ADLS Sink Parameters:ADLS Account Name": "[***ACCOUNT NAME***]",
 "parameter.ADLS Sink Parameters:ADLS Account Key": "[***ACCOUNT KEY***]",
 "parameter.ADLS Sink Parameters:ADLS Filesystem Name": "[***FILESYSTEM
 NAME***]",
 "parameter.ADLS Sink Parameters:ADLS Directory Name": "[***DIRECTORY
 NAME***]",
 "parameter.ADLS Sink Parameters:Maximum File Size": "10 MB",
 "parameter.ADLS Sink Parameters:Kafka Message Data Format": "Raw",
 "parameter.ADLS Sink Parameters:Output File Demarcator": "\n",
 "parameter.ADLS Sink Parameters:Output Filename Pattern": "messages_${file
name.uuid}"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic the connector fetches messages from.

ADLS Account Name

The Storage Account Name to use for authentication to Azure. This is also the target account where the output file is uploaded.

ADLS Account Key

The Storage Account Key to use for authentication to Azure.

ADLS Filesystem Name

The filesystem (or container) in the Storage Account to upload data to.

ADLS Directory Name

The target directory in the filesystem.

Maximum File Size

The maximum size of the output data file. In this example, the maximum size is 10 MB.

Kafka Message Data Format

The format of the messages the connector receives from Kafka. In this example, this property is set to Raw. This means that the connector expects unstructured text or binary data.

Output File Demarcator

Specifies the character sequence for demarcating (delimiting) message boundaries when multiple Kafka messages are ingested into an output file as raw messages. In this example, the property is set to \n (newline). This means that the newline character is used to separate the single line character data of the Kafka messages.

Output Filename Pattern

Specifies the structure of the name of the output file. This property accepts string literals (fixed text) as well as various expressions. In this example, the property is set to messages_${filename.uuid}.

messages_ is a string literal. ${filename.uuid} is an expression that inserts a generated UUID in the filename. As a result, the files are named according to the following pattern: messages_*[***UUID***]*.

Configuration example for fetching JSON data

In this example, the connector fetches messages in JSON format from Kafka. The connector parses the JSON records and converts them to Avro format using the schema inferred from the JSON data. The schema is also embedded in the output file. The connector merges multiple records (all messages that are available for a single execution of the connector) into an Avro file, generates a name for the file using the following pattern data_*[***TIMESTAMP***]*_*[***SEQUENCE***]*.avro, and uploads the file to ADLS. The connector authenticates to Azure using a Service Principal.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSinkConnec
tor",
 "meta.smm.predefined.flow.name": "ADLS Sink",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter
",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "failure.ports": "PutAzureDataLakeStorage Failure",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.ADLS Sink Parameters:ADLS Account Name": "[***ACCOUNT NAME***]",
 "parameter.ADLS Sink Parameters:Azure Service Principal Tenant ID":
 "[***SERVICE PRINCIPAL TENANT ID***]",
 "parameter.ADLS Sink Parameters:Azure Service Principal Client ID":
 "[***SERVICE PRINCIPAL CLIENT ID***]",
 "parameter.ADLS Sink Parameters:Azure Service Principal Client Secret":
 "[***SERVICE PRINCIPAL CLIENT SECRET***]",
 "parameter.ADLS Sink Parameters:ADLS Filesystem Name": "[***FILESYSTEM
 NAME***]",
 "parameter.ADLS Sink Parameters:ADLS Directory Name": "[***DIRECTORY
 NAME***]",
 "parameter.ADLS Sink Parameters:Kafka Message Data Format": "JSON",
 "parameter.ADLS Sink Parameters:Schema Access Strategy": "Infer Schema",
 "parameter.ADLS Sink Parameters:Schema Write Strategy": "Embed Avro Schem
a",
 "parameter.ADLS Sink Parameters:Output File Data Format": "Avro",
 "parameter.ADLS Sink Parameters:Output Filename Pattern": "data_${filena
me.timestamp}_${filename.sequence}.avro",
 "parameter.ADLS Sink Parameters:Output Filename Timestamp Format": "yyyyM
Mdd_HHmmss_SSS"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic the connector fetches messages from.

ADLS Account Name

The Storage Account Name to use for authentication to Azure. This is also the target account where the output file is uploaded.

Service Principal *

The properties of the Service Principal to use for authentication to Azure.

ADLS Filesystem Name

The filesystem (or container) in the Storage Account to upload data to.

ADLS Directory Name

The target directory in the filesystem.

Kafka Message Data Format

The format of the messages the connector receives from Kafka. In this example, this property is set to JSON. This means that the connector expects JSON data.

Schema Access Strategy

Specifies the strategy used for determining the schema of the Kafka record. In this example, this property is set to Infer Schema, meaning that the schema is determined (inferred) from the JSON data.

Schema Write Strategy

Specifies whether the record schema is written to the output data file. In this example, this property is set to Embed Avro Schema, meaning that the schema is embedded in the output Avro file.

Output File Data Format

Specifies the format of the records written to the output file. In this example, this property is set to Avro, meaning that the output file format is Avro.

Output Filename Pattern

Specifies the structure of the name of the output file. This property accepts string literals (fixed text) as well as various expressions. In this example, the property is set to data_${filename.timestamp}_${filename.sequence}.avro.

data_, the underscore (_), and .avro are string literals. ${filename.timestamp} and ${filename.sequence} are expressions. ${filename.timestamp} inserts the current timestamp in the filename. ${filename.sequence} inserts an incrementing sequence value in the filename. As a result, the files are named according to the following pattern: data_*[***TIMESTAMP***]*_*[***SEQUENCE***]*.avro.

Output Filename Timestamp Format

Specifies the timestamp format used in the ${filename.timestamp} expression. The expression is used when generating the output filename.

Related Information

Apache Kafka documentation

Stateless NiFi Sink properties reference

ADLS Sink properties reference

Amazon S3 Sink

Learn more about the Amazon S3 Sink connector

The Amazon S3 Sink 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 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.AvroConver
ter",
    "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.avr
o.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, AvroPartitio nWriter 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

HDFS Sink

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 properties reference*.

```
{
    "connector.class": "com.cloudera.dim.kafka.connect.hdfs.HdfsSinkConnec
tor",
    "tasks.max": 1,
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "value.converter": "com.cloudera.dim.kafka.connect.converts.AvroConve
rter",
    "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.AvroPartitio
nWriter",
    "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, AvroPartitio nWriter 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 properties reference*.

```
{
    "connector.class": "com.cloudera.dim.kafka.connect.hdfs.HdfsSinkConnec
tor",
    "tasks.max": 1,
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "value.converter": "com.cloudera.dim.kafka.connect.converts.AvroConve
rter",
    "value.converter.passthrough.enabled": true,
    "value.converter.schema.registry.url": "http://localhost:9090/api/v1",
    "topics": "avro_topic",
    "hdfs.uri": "o3fs://bucket1.volume1.ozone1/",
    "hdfs.output": "/topics_output/",
    "output.writer": "com.cloudera.dim.kafka.connect.hdfs.avro.AvroPartit
ionWriter",
    "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, AvroPartitio nWriter 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.

HTTP Sink connector

The HTTP Sink connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka Connect framework. Learn about the connector, its properties, and configuration.

The HTTP Sink connector obtains messages from a Kafka topic and transfers their content in a HTTP POST requests to a specified endpoint. The topic the connector receives messages from is determined by the value of the topics parameter in the configuration. The connector can forward the data it reads from Kafka as is (raw data) or can be configured to execute record processing. When record processing is enabled, the connector expects the incoming data in Avro or JSON format. In case of Avro, the connector can either read the record schema from the Avro file it receives (provided that the schema is embedded) or it can fetch the schema from Schema Registry. In case of JSON, it can either infer the schema or fetch it from Schema Registry. What strategy is used to retrieve the schema is determined by the Schema Access Strategy property.

If Schema Registry is used, and it is on a Kerberized cluster, the krb5.file property must point to the krb5.conf file that provides access to the cluster on which Schema Registry is present. This means that the krb5.conf file must be on the same cluster node that the connector runs on. The connections to Schema Registry and the HTTP server can be secured by TLS. The keystore and truststore files necessary for securing these connections must also be on the same cluster node that the connector runs on.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Sink properties

These are the properties that are specific to the Stateless NiFi Sink connector. All Stateless NiFi Sink connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Sink property reference*.

Connector/dataflow-specific properties

These properties are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *HTTP Sink properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.
- Properties not marked as required must be completely removed from the configuration JSON if not set.
- The value of the Schema Access Strategy property is not independent of the value of the Kafka Message Data F ormat property. As a result, you must exercise caution when configuring Schema Access Strategy.
  - If the value of Kafka Message Data Format is AVRO, the possible values for Schema Access Strategy are Schema Registry or Embedded Schema.
  - If the value of Kafka Message Data Format is JSON, the possible values for Schema Access Strategyare Sche ma Registryor Infer Schema.

Configuration example

In this example, the connector receives data in any format and forwards the raw data as the content of an HTTP POST request.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSinkConnec
tor",
 "meta.smm.predefined.flow.name": "HTTP Sink",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.converters.ByteArrayConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConvert
er",
 "tasks.max": "1",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "input.port": "Input from Kafka",
 "failure.ports": "Failure",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.HTTP Sink Parameters:Forward Raw Data": "true",
 "parameter.HTTP Sink Parameters:Remote URL": "http://[***SERVER
 HOSTNAME***]:[***PORT***]/[***PATH***]"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic the connector fetches messages from.

Forward Raw Data

Specifies whether messages from Kafka should be forwarded as is or converted to JSON. In this example, the property is set to true, meaning that the connector does not process any records. It forwards incoming data as is.

Remote URL

Identifies the HTTP endpoint that receives the messages sent by this connector. In this example, SSL is not used. As a result, the URL starts with http not https. For example, http://my.http-serve r.com:22000/contentListener.

Related Information

Apache Kafka documentation

Stateless NiFi Sink properties reference

HTTP Sink properties reference

JDBC Sink connector

The JDBC Sink connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka Connect framework. Learn about the connector, its properties, and configuration.

The JDBC Sink connector fetches messages from Kafka and loads them into a database table. The topic this connector receives messages from is determined by the value of the topics property in the configuration. The messages the connector receives from Kafka can be in either Avro or JSON format and must contain records that can be inserted into the database table. The schema of the records can be a predefined schema retrieved from Schema Registry (for both Avro and JSON data), it can be embedded in the Avro data, or inferred from the JSON data. The strategy that is used to retrieve the schema is determined by the Schema Access Strategy property.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Sink properties

These are the properties that are specific to the Stateless NiFi Sink connector. All Stateless NiFi Sink connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Sink property reference*.

Connector/dataflow-specific properties

These properties are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *JDBC Sink properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.
- Properties not marked as required must be completely removed from the configuration JSON if not set.

Configuration example

In this example, the connector fetches messages in JSON format from Kafka, translates it to a SQL INSERT statement in order to load the data into a PostgreSQL database. The schema of the records must match the schema of the database table. The record schema in this example is inferred from the input JSON.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSinkConnec
tor",
 "meta.smm.predefined.flow.name": "JDBC Sink",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter
",
 "tasks.max": "1",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "failure.ports": "Retry from PutDatabaseRecord",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.JDBC Sink Parameters:Database Connection URL": "[***JDBC
 URL***]",
 "parameter.JDBC Sink Parameters:Database Driver Location": "[***PATH TO
 JDBC DRIVER***]",
 "parameter.JDBC Sink Parameters:Database Driver Class Name": "org.postgr
esql.Driver",
 "parameter.JDBC Sink Parameters:Database Type": "PostgreSQL",
 "parameter.JDBC Sink Parameters:Database User Name": "[***USERNAME***]",
 "parameter.JDBC Sink Parameters:Database User Password":
 "[***PASSWORD***]",
 "parameter.JDBC Sink Parameters:Database Table Name": "[***TABLE NAME***]",
 "parameter.JDBC Sink Parameters:Kafka Message Data Format": "JSON",
 "parameter.JDBC Sink Parameters:Schema Access Strategy": "Infer Schema"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic the connector fetches messages from.

Database Connection URL

The JDBC URL of the PostgreSQL database. For example, jdbc:postgresql://myhost:5432/mydb.

Database Driver Location

The path to the PostgreSQL JDBC driver JAR file (or the directory containing the JAR).

Database Driver Class Name

The Java class name of the PostgreSQL Driver implementation.

Database Type

The type of the database. Because this is a PostgreSQL example, this property is set to PostgreSQL.

Database User Name

The username used for authenticating to the database.

Database User Password

The name of the database table to load data to.

Kafka Message Data Format

The format of the messages the connector receives from Kafka.

Schema Access Strategy

Specifies the strategy used for determining the schema of the Kafka record. In this example, the property is set to Infer Schema. This means that the schema is determined (inferred) from the JSON data.

Related Information

Apache Kafka documentation

Stateless NiFi Sink properties reference

JDBC Sink properties reference

Kudu Sink connector

The Kudu Sink connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka Connect framework. Learn about the connector, its properties, and configuration.

The Kudu Sink connector fetches messages from Kafka and loads them into a table in Kudu. The topic this connector receives messages from is determined by the value of the topics property in the configuration. The messages the connector receives from Kafka can be in either Avro or JSON format and must contain records that can be inserted into a Kudu table. If the connector input is in Avro format, then it can either read the schema from the Avro file it receives (provided that the schema is embedded) or it can fetch the schema from Schema Registry. If the connector's input is in JSON format, then it can either infer the schema, or fetch it from Schema Registry. The strategy that is used to retrieve the schema is determined by the Schema Access Strategy property.

Kudu is expected to be on a Kerberized cluster. Additionally, Schema Registry should be on the same cluster as Kudu. In the connector's configuration the krb5.file property must point to the krb5.conf file that provides access to the cluster that Kudu and Schema Registry are on. The keytab and truststore files that are necessary to access Kudu and Schema Registry must be present on the cluster node that the connector runs on.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Sink properties

These are the properties that are specific to the Stateless NiFi Sink connector. All Stateless NiFi Sink connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Sink property reference*.

Connector/dataflow-specific properties

These properties are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *Kudu Sink properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.
- Properties not marked as required must be completely removed from the configuration JSON if not set.

- Schema Registry URL is mandatory even if no registry is used. In a case like this, use the default value, or remove the property completely from the configuration JSON. Alternatively, if your input data is JSON, you can use the Infer Schema value to infer the schema from the input, or if your input data is Avro, you can use the Embedded Schema value to read the schema embedded in the input.
- Truststore parameters are mandatory even if connecting to a non-secure Schema Registry or no registry is used. In that case the default (empty) truststore can be used.
- Kerberos parameters for Schema Registry are mandatory even if connecting to a non-secure Schema Registry or no registry is used. In that case the default (empty) keytab can be used.
- Kudu is expected to be on a Kerberized cluster. As a result, Kerberos parameters for Kudu are mandatory.

Configuration example

In this example, the connector receives data in JSON format from a Kafka topic and forwards it to a Kudu table. The record schema is inferred based on the data.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSinkConnec
tor",
 "meta.smm.predefined.flow.name": "Kudu Sink",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter
",
 "tasks.max": "1",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "input.port": "Input from Kafka",
 "failure.ports": "Failure",
 "krb5.file": "[***PATH TO KRB5 FILE***]",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.Kudu Sink Parameters:Kafka Message Data Format": "JSON",
 "parameter.Kudu Sink Parameters:Kerberos Keytab for Kudu": "[***PATH TO
 KUDU KEYTAB***]",
 "parameter.Kudu Sink Parameters:Kerberos Principal for Kudu": "[***KUDU
 PRINCIPAL***]",
 "parameter.Kudu Sink Parameters:Kudu Masters": "[***KUDU MASTER 1 WITH
 PORT***],[***KUDU MASTER 2 WITH PORT***],[***KUDU MASTER 3 WITH PORT***]",
 "parameter.Kudu Sink Parameters:Schema Access Strategy": "Infer Schema",
 "parameter.Kudu Sink Parameters:Table Name With Schema": "[***KUDU
 SCHEMA***].[***KUDU TABLE***]"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

krb5.file

The path to a krb5.conf file that contains the information needed to access the Kudu service on a Kerberized cluster

topics

The name of the Kafka topic that the connector fetches messages from.

Kafka Message Data Format

The format of the messages the connector receives from Kafka. In this example, this property is set to JSON. This means that the connector expects data in JSON format.

Kerberos Keytab for Kudu

The path to the keytab file that enables access to the Kudu service on a Kerberized cluster.

Kerberos Principal for Kudu

The principal used to access Kudu

Kudu Masters

A comma-separated list of server URLs that identify the Kudu service's master nodes. For example: localhost:7051,localhost:7151,localhost:7251

Schema Access Strategy

Specifies the strategy used for determining the schema of the Kafka record. In this example, the property is set to Infer Schema. This means that the record schema is inferred based on the data the connector receives.

Table Name With Schema

The schema and name of the Kudu table in which the records get inserted.

Related Information

Apache Kafka documentation

Stateless NiFi Sink properties reference

Kudu Sink properties reference

S3 Sink connector

The S3 Sink Connector is a Stateless NiFi dataflow developed by Cloudera that is running in the Kafka Connect framework. Learn about the connector, its properties, and configuration.

The S3 Sink connector fetches messages from Kafka and uploads them to AWS S3. The topic this connector receives messages from is determined by the value of the topics property in the configuration. The messages can contain unstructured (character or binary) data or they can be in Avro or JSON format.

If the input is unstructured data, record processing is disabled. In a case like this, multiple messages can be concatenated into a single output file on S3 using a demarcator (for example, newline for text messages). Merging is optional, large binary data can be forwarded to S3 at a 1:1 ratio, one Kafka message equals a single S3 object.

If the input is either Avro or JSON, record processing is enabled. In a case like this, the schema of the records can be a predefined schema retrieved from Schema Registry (for both Avro and JSON data), it can be embedded in the Avro data, or inferred from the JSON data. The output can be Avro, JSON or CSV. Multiple records are typically merged into a single output file before uploading to S3.

The connector can authenticate to AWS using an Access Key.

Properties and configuration

Configuration is passed to the connector in a JSON file during creation. The properties of the connector can be categorized into three groups. These are as follows:

Common connector properties

These are the properties of the Kafka Connect framework that are accepted by all connectors. For a comprehensive list of these properties, see the *Apache Kafka documentation*.

Stateless NiFi Sink properties

These are the properties that are specific to the Stateless NiFi Sink connector. All Stateless NiFi Sink connectors share and accept these properties. For a comprehensive list of these properties, see the *Stateless NiFi Sink property reference*.

Connector/dataflow-specific properties

These properties are unique to this specific connector. Or to be more precise, unique to the dataflow running within the connector. These properties use the following prefix:

```
parameter.[***CONNECTOR NAME***] Parameters:
```

For a comprehensive list of these properties, see the *ADLS Sink properties reference*.

Tip: When creating a new connector using the SMM UI, all valid properties for this connector are presented in the default configuration template. You can use the template in the UI as a quick reference to view all accepted properties.

Notes and limitations

- Required properties must be assigned a valid value even if they are not used in the particular configuration. If a required property is not used, either leave its default value, or completely remove the property from the configuration JSON.
- If a property that has a default value is completely removed from the configuration JSON, the system uses the default value.
- Properties not marked as required must be completely removed from the configuration JSON if not set.

Configuration example for fetching unstructured data

In this example, the connector fetches unstructured messages containing single line character data from Kafka. The connector concatenates the messages using newline characters as the demarcator. The files that the connector creates and uploads to ADLS are maximum 10 MB in size. The files are named according to the following pattern: mess ages_*[***UUID***]*. The connector authenticates to AWS using an Access Key.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSinkConnec
tor",
 "meta.smm.predefined.flow.name": "S3 Sink",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConvert
er",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "failure.ports": "PutS3Object Failure",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.S3 Sink Parameters:AWS Access Key ID": "[***ACCESS KEY ID***]",
 "parameter.S3 Sink Parameters:AWS Secret Access Key": "[***SECRET ACCESS
 KEY***]",
 "parameter.S3 Sink Parameters:S3 Region": "[***S3 REGION***]",
 "parameter.S3 Sink Parameters:S3 Bucket": "[***S3 Bucket***]",
 "parameter.S3 Sink Parameters:Maximum File Size": "10 MB",
 "parameter.S3 Sink Parameters:Kafka Message Data Format": "Raw",
 "parameter.S3 Sink Parameters:Output File Demarcator": "\n",
 "parameter.ADLS Sink Parameters:Output Filename Pattern": "messages_${fi
lename.uuid}"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic the connector fetches messages from.

AWS Access Key ID

The Access Key ID to use for authentication to AWS.

AWS Secret Access Key

The Secret Access Key to use for authentication to AWS.

S3 Region

The AWS Region of the S3 Bucket to upload data to.

S3 Bucket

The S3 Bucket to upload data to.

Maximum File Size

The maximum size of the output data file. In this example, the maximum size is 10 MB.

Kafka Message Data Format

The format of the messages the connector receives from Kafka. In this example, this property is set to Raw. This means that the connector expects unstructured text or binary data.

Output File Demarcator

Specifies the character sequence for demarcating (delimiting) message boundaries when multiple Kafka messages are ingested into an output file as raw messages. In this example, the property is set to \n (newline). This means that the newline character is used to separate the single line character data of the Kafka messages.

Output Filename Pattern

Specifies the structure of the name of the output file. This property accepts string literals (fixed text) as well as various expressions. In this example, the property is set to messages_${filename.uuid}.

messages_ is a string literal, ${filename.uuid} is an expression that inserts a generated UUID in the filename. As a result, the files are named according to the following pattern: mess ages_*[***UUID***]*.

Configuration example for fetching JSON data

In this example, the connector fetches messages in JSON format from Kafka. The connector parses the JSON records and converts them to Avro format using the schema inferred from the JSON data. The schema is also embedded in the output file. The connector merges multiple records (all messages that are available for a single execution of the connector) into an Avro file, generates a name for the file using the following pattern data_*[***TIMESTAMP***]*_*[***SEQUENCE***]*.avro, and uploads the file to S3. The connector authenticates to AWS using an Access Key.

Note: All properties that are not present in this example have suitable default values or are unnecessary for this specific use case.

```
{
 "connector.class": "org.apache.nifi.kafka.connect.StatelessNiFiSinkConnec
tor",
 "meta.smm.predefined.flow.name": "S3 Sink",
 "meta.smm.predefined.flow.version": "1.0.0",
 "key.converter": "org.apache.kafka.connect.storage.StringConverter",
 "value.converter": "org.apache.kafka.connect.converters.ByteArrayConvert
er",
 "nexus.url": "https://repository.cloudera.com/artifactory/repo",
 "extensions.directory": "/tmp/nifi-stateless-extensions",
 "working.directory": "/tmp/nifi-stateless-working",
 "failure.ports": "PutS3Object Failure",
 "topics": "[***KAFKA TOPIC NAME***]",
 "parameter.S3 Sink Parameters:AWS Access Key ID": "[***ACCESS KEY ID***]",
 "parameter.S3 Sink Parameters:AWS Secret Access Key": "[***SECRET ACCESS
 KEY***]",
 "parameter.S3 Sink Parameters:S3 Region": "[***S3 REGION***]",
 "parameter.S3 Sink Parameters:S3 Bucket": "[***S3 BUCKET***]"
 "parameter.ADLS Sink Parameters:Kafka Message Data Format": "JSON",
 "parameter.ADLS Sink Parameters:Schema Access Strategy": "Infer Schema",
 "parameter.ADLS Sink Parameters:Schema Write Strategy": "Embed Avro Schema
",
 "parameter.ADLS Sink Parameters:Output File Data Format": "Avro",
 "parameter.ADLS Sink Parameters:Output Filename Pattern": "data_${filenam
e.timestamp}_${filename.sequence}.avro",
 "parameter.ADLS Sink Parameters:Output Filename Timestamp Format": "yyyyMM
dd_HHmmss_SSS"
}
```

The following list collects the properties from the configuration example that must be customized for this use case.

topics

The name of the Kafka topic the connector fetches messages from.

AWS Access Key ID

The Access Key ID to use for authentication to AWS.

AWS Secret Access Key

The Secret Access Key to use for authentication to AWS.

S3 Region

The AWS Region of the S3 Bucket to upload data to.

S3 Bucket

The S3 Bucket to upload data to.

Kafka Message Data Format

The format of the messages the connector receives from Kafka. In this example, this property is set to JSON. This means that the connector expects JSON data.

Schema Access Strategy

Specifies the strategy used for determining the schema of the Kafka record. In this example, this property is set to Infer Schema, meaning that the schema is determined (inferred) from the JSON data.

Schema Write Strategy

Specifies whether the record schema is written to the output data file. In this example, this property is set to Embed Avro Schema, meaning that the schema is embedded in the output Avro file.

Output File Data Format

Specifies the format of the records written to the output file. In this example, this property is set to Avro, meaning that the output file format is Avro.

Output Filename Pattern

Specifies the structure of the name of the output file. This property accepts string literals (fixed text) as well as various expressions. In this example, the property is set to data_${filename.timestamp}_${filename.sequence}.avro.

data_, the underscore (_), and .avro are string literals. ${filename.timestamp} and ${filename.sequence} are expressions. ${filename.timestamp} inserts the current timestamp in the filename. ${filename.sequence} inserts an incrementing sequence value in the filename. As a result, the files are named according to the following pattern: data_*[***TIMESTAMP***]*_*[***SEQUENCE***]*.avro.

Output Filename Timestamp Format

Specifies the timestamp format used in the ${filename.timestamp} expression. The expression is used when generating the output filename.

Related Information

Apache Kafka documentation

Stateless NiFi Sink properties reference

S3 Sink properties reference