Kafka Connect FileSystem Connector Documentation

Release 1.3

Mario Molina

Dec 19, 2021

Contents

Kafka Connect FileSystem Connector is a source connector for reading records from files in the file systems specified and load them into Kafka.

The connector supports:

- Several sort of File Systems (FS) to use.
- Dynamic and static URIs to ingest data from.
- Policies to define rules about how to look for files and clean them up after processing.
- File readers to parse and read different kind of file formats.

To learn more about the connector you can read *this section* and for more detailed configuration options you can read *this other one*.

Also, you can download the source code from here.

CHAPTER 1

Contents

1.1 Connector

The connector takes advantage of the abstraction provided from Hadoop Common using the implementation of the `org.apache.hadoop.fs.FileSystem` class. So, it's possible to use a wide variety of FS or if your FS is not included in the Hadoop Common API you can implement an extension of this abstraction and using it in a transparent way.

Among others, these are some file systems it supports:

- HDFS.
- S3.
- Google Cloud Storage.
- Azure Blob Storage & Azure Data Lake Store.
- FTP & SFTP.
- WebHDFS.
- Local File System.
- Hadoop Archive File System.

1.1.1 Getting started

Prerequisites

- Apache Kafka 2.6.0.
- Java 8.
- Confluent Schema Registry (recommended).

Building from source

```
mvn clean package
```

General config

The `kafka-connect-fs.properties` file defines the following properties as required:

```
name=FsSourceConnector
connector.class=com.github.mmolimar.kafka.connect.fs.FsSourceConnector
tasks.max=1
fs.uris=file:///data,hdfs://localhost:8020/data
topic=mytopic
policy.class=<Policy class>
policy.recursive=true
policy.regexp=.*
policy.batch_size=0
policy.cleanup=none
file_reader.class=<File reader class>
file_reader.batch_size=0
```

1. The connector name.
2. Class indicating the connector.
3. Number of tasks the connector is allowed to start.
4. Comma-separated URIs of the FS(s). They can be URIs pointing out directly to a file or a directory in the FS. These URIs can also be dynamic by using expressions for modifying them in runtime.
5. Topic in which copy data from the FS.
6. Policy class to apply (must implement `com.github.mmolimar.kafka.connect.fs.policy.Policy` interface).
7. Flag to activate traversed recursion in subdirectories when listing files.
8. Regular expression to filter files from the FS.
9. Number of files that should be handled at a time. Non-positive values disable batching.
10. Cleanup strategy to manage processed files.
11. File reader class to read files from the FS (must implement `com.github.mmolimar.kafka.connect.fs.file.reader.FileReader` interface).
12. Number of records to process at a time. Non-positive values disable batching.

A more detailed information about these properties can be found *here*.

Running in local

```
export KAFKA_HOME=/path/to/kafka/install/dir
```

```
mvn clean package
export CLASSPATH="$(find target/ -type f -name '*.jar'| grep '\-package' | tr '\n' ':
↪')"
$KAFKA_HOME/bin/connect-standalone.sh $KAFKA_HOME/config/connect-standalone.
↪properties config/kafka-connect-fs.properties
```

Running in Docker

```
mvn clean package
```

```
docker build --build-arg PROJECT_VERSION=<VERSION> .
docker-compose build
docker-compose up -d
docker logs --tail="all" -f connect
```

```
curl -sX GET http://localhost:8083/connector-plugins | grep FsSourceConnector
```

1.1.2 Components

There are two main concepts to decouple concerns within the connector. They are **policies** and **file readers**, described below.

Policies

In order to ingest data from the FS(s), the connector needs a **policy** to define the rules to do it.

Basically, the policy tries to connect to each FS included in the `fs.uris` connector property, lists files (and filter them using the regular expression provided in the `policy.regexp` property) and enables a file reader to read records.

The policy to be used by the connector is defined in the `policy.class` connector property.

Important: When delivering records from the connector to Kafka, they contain their own file offset so, if in the next eventual policy execution this file is processed again, the policy will seek the file to this offset and process the next records if any (**if the offset was committed**).

Note: If the URIs included in the `fs.uris` connector property contain any expression of the form `${XXX}`, this dynamic URI is built in the moment of the policy execution.

Currently, there are few policies to support some use cases but, for sure, you can develop your own one if the existing policies don't fit your needs. The only restriction is that you must implement the interface `com.github.mmolimar.kafka.connect.fs.policy.Policy`.

Simple

It's a policy which just filters and processes files included in the corresponding URIs one time.

Attention: This policy is more oriented for testing purposes.

Sleepy

The behaviour of this policy is similar to Simple policy but on each execution it sleeps and wait for the next one. Additionally, its custom properties allow to end it.

You can learn more about the properties of this policy *here*.

Cron

This policy is scheduled based on cron expressions and their format to put in the configuration are based on the library Quartz Scheduler.

After finishing each execution, the policy gets slept until the next one is scheduled, if applicable.

You can learn more about the properties of this policy *here*.

HDFS file watcher

It uses Hadoop notifications events and all create/append/rename/close events will be reported as files to be ingested.

Just use it when you have HDFS URIs.

You can learn more about the properties of this policy *here*.

> **Attention:** The URIs included in the general property `fs.uris` will be filtered and only those ones which start with the prefix `hdfs://` will be watched. Also, this policy will only work for Hadoop versions 2.6.0 or higher.

S3 event notifications

It uses S3 event notifications sent from S3 to process files which have been created or modified in S3. These notifications will be read from a AWS-SQS queue and they can be sent to SQS directly from S3 or via AWS-SNS, either as a SNS notification or a raw message in the subscription.

Just use it when you have S3 URIs and the event notifications in the S3 bucket must be enabled to a SNS topic or a SQS queue.

You can learn more about the properties of this policy *here*.

File readers

They read files and process each record from the FS. The **file reader** is needed by the policy to enable the connector to process each record and includes in the implementation how to seek and iterate over the records within the file.

The file reader to be used when processing files is defined in the `file_reader.class` connector property.

In the same way as policies, the connector provides several sort of readers to parse and read records for different file formats. If you don't have a file reader that fits your needs, just implement one with the unique restriction that it must implement the interface `com.github.mmolimar.kafka.connect.fs.file.reader.FileReader`.

The are several file readers included which can read the following file formats:

- Parquet.
- Avro.
- ORC.
- SequenceFile.
- Cobol / EBCDIC.

- Other binary files.
- CSV.
- TSV.
- Fixed-width.
- JSON.
- XML.
- YAML.
- Text.

Parquet

Reads files with Parquet format.

The reader takes advantage of the Parquet-Avro API and uses the Parquet file as if it was an Avro file, so the message sent to Kafka is built in the same way as the Avro file reader does.

More information about properties of this file reader *here*.

Avro

Files with Avro format can be read with this reader.

The Avro schema is not needed due to is read from the file. The message sent to Kafka is created by transforming the record by means of Confluent avro-converter API.

More information about properties of this file reader *here*.

ORC

ORC files are a self-describing type-aware columnar file format designed for Hadoop workloads.

This reader can process this file format, translating its schema and building a Kafka message with the content.

> **Warning:** If you have ORC files with `union` data types, this sort of data types will be transformed in a `map` object in the Kafka message. The value of each key will be `fieldN`, where `N` represents the index within the data type.

More information about properties of this file reader *here*.

SequenceFile

Sequence files are one kind of the Hadoop file formats which are serialized in key-value pairs.

This reader can process this file format and build a Kafka message with the key-value pair. These two values are named `key` and `value` in the message by default but you can customize these field names.

More information about properties of this file reader *here*.

Cobol

Mainframe files (Cobol / EBCDIC binary files) can be processed with this reader which uses the Cobrix parser.

By means of the corresponding copybook -representing its schema-, it parses each record and translate it into a Kafka message with the schema.

More information about properties of this file reader *here*.

Binary

All other kind of binary files can be ingested using this reader.

It just extracts the content plus some metadata such as: path, file owner, file group, length, access time, and modification time.

Each message will contain the following schema:

- `path`: File path (string).
- `owner`: Owner of the file. (string).
- `group`: Group associated with the file. (string).
- `length`: Length of this file, in bytes. (long).
- `access_time`: Access time of the file. (long).
- `modification_time`: Modification time of the file (long).
- `content`: Content of the file (bytes).

More information about properties of this file reader *here*.

CSV

CSV file reader using a custom token to distinguish different columns in each line.

It allows to distinguish a header in the files and set the name of their columns in the message sent to Kafka. If there is no header, the value of each column will be in the field named `column_N` (**N** represents the column index) in the message. Also, the token delimiter for columns is configurable.

This reader is based on the Univocity CSV parser.

More information about properties of this file reader *here*.

TSV

TSV file reader using a tab `\t` to distinguish different columns in each line.

Its behaviour is the same one for the CSV file reader regarding the header and the column names.

This reader is based on the Univocity TSV parser.

More information about properties of this file reader *here*.

FixedWidth

FixedWidth is a plain text file reader which distinguishes each column based on the length of each field.

Its behaviour is the same one for the CSV / TSV file readers regarding the header and the column names.

This reader is based on the Univocity Fixed-Width parser.

More information about properties of this file reader *here*.

JSON

Reads JSON files which might contain multiple number of fields with their specified data types. The schema for this sort of records is inferred reading the first record and marked as optional in the schema all the fields contained.

More information about properties of this file reader *here*.

XML

Reads XML files which might contain multiple number of fields with their specified data types. The schema for this sort of records is inferred reading the first record and marked as optional in the schema all the fields contained.

> **Warning:** Take into account the current limitations.

More information about properties of this file reader *here*.

YAML

Reads YAML files which might contain multiple number of fields with their specified data types. The schema for this sort of records is inferred reading the first record and marked as optional in the schema all the fields contained.

More information about properties of this file reader *here*.

Text

Reads plain text files.

Each line represents one record (by default) which will be in a field named `value` in the message sent to Kafka by default but you can customize these field names.

More information about properties of this file reader *here*.

Agnostic

Actually, this reader is a wrapper of the readers listing above.

It tries to read any kind of file format using an internal reader based on the file extension, applying the proper one (Parquet, Avro, ORC, SequenceFile, Cobol / EBCDIC, CSV, TSV, FixedWidth, JSON, XML, YAML, or Text). In case of no extension has been matched, the Text file reader will be applied.

Default extensions for each format (configurable):

- Parquet: `.parquet`
- Avro: `.avro`
- ORC: `.orc`
- SequenceFile: `.seq`
- Cobol / EBCDIC: `.dat`
- Other binary files: `.bin`
- CSV: `.csv`
- TSV: `.tsv`
- FixedWidth: `.fixed`
- JSON: `.json`
- XML: `.xml`
- YAML: `.yaml`
- Text: any other sort of file extension.

More information about properties of this file reader *here*.

1.2 Configuration Options

1.2.1 General

General config properties for this connector.

`name` The connector name.

- Type: string
- Importance: high

`connector.class` Class indicating the connector.

- Type: string
- Importance: high

`tasks.max` Number of tasks the connector is allowed to start.

- Type: int
- Importance: high

Tip: The number of URIs specified in the connector config will be grouped based on the number of tasks defined. So, if you have just one URI with one task is fine. Otherwise, if you want to improve the performance and process URIs in parallel you should adjust this number based on your requirements.

`fs.uris` Comma-separated URIs of the FS(s). They can be URIs pointing directly to a file in the FS and also can be dynamic using expressions for modifying the URIs in runtime. These expressions have the form `${XXX}` where XXX represents a pattern from `java.time.format.DateTimeFormatter` Java class.

- Type: string
- Importance: high

Tip: If you want to ingest data from dynamic directories, this is, directories created every day and avoiding to add new URIs or look for files from a parent directory, you can include expressions in the URIs to do that. For example, for this URI `file:///data/${yyyy}`, it will be converted to `file:///data/2020` (when executing whe policy).

You can use as many as you like in the URIs, for instance: `file:///data/${yyyy}/${MM}/${dd}/${HH}${mm}`

Tip: If you want to ingest data from S3, you can add credentials with: `policy.fs.fs.s3a.access.key=<ACCESS_KEY>` and `policy.fs.fs.s3a.secret.key=<SECRET_KEY>`. Also, in case you want to configure a custom credentials provider, you should use `policy.fs.fs.s3a.aws.credentials.provider=<CLASS>` property.

`topic` Topic in which copy data to.

- Type: string
- Importance: high

`poll.interval.ms` Frequency in milliseconds to poll for new data. This config just applies when the policies have ended.

- Type: int
- Default: `10000`
- Importance: medium

`policy.class` Policy class to apply (must implement `com.github.mmolimar.kafka.connect.fs.policy.Policy` interface).

- Type: string
- Importance: high

`policy.regexp` Regular expression to filter files from the FS.

- Type: string
- Importance: high

`policy.recursive` Flag to activate traversed recursion in subdirectories when listing files.

- Type: boolean
- Default: `false`
- Importance: medium

`policy.batch_size` Number of files that should be handled at a time. Non-positive values disable batching.

- Type: int
- Default: `0`
- Importance: medium

`policy.cleanup` Cleanup strategy to use when skipping files. It's possible to move these files to another folder, remove them or do nothing.

- Type: enum (available values `none`, `move` and `delete`)
- Default: `none`

- Importance: medium

policy.cleanup.move Target directory to move files for the `move` cleanup strategy. Mandatory just in case of using this strategy.

- Type: string
- Importance: medium

policy.cleanup.move.prefix Prefix to set to the filename in moved files.

- Type: string
- Default: ""
- Importance: low

policy.<policy_name>.<policy_property> This represents custom properties you can include based on the policy class specified.

- Type: based on the policy.
- Importance: based on the policy.

policy.fs.<fs_property> Custom properties to use for the FS.

- Type: based on the FS.
- Importance: based on the FS.

file_reader.class File reader class to read files from the FS (must implement `com.github.mmolimar.kafka.connect.fs.file.reader.FileReader` interface).

- Type: string
- Importance: high

file_reader.batch_size Number of records to process at a time. Non-positive values disable batching.

- Type: int
- Default: `0`
- Importance: medium

file_reader.<file_reader_name>.<file_reader_property> This represents custom properties you can include based on the file reader class specified.

- Type: based on the file reader.
- Importance: based on the file reader.

1.2.2 Policies

Some policies have custom properties to define and others don't. So, depending on the configuration you'll have to take into account their properties.

Simple

This policy does not have any additional configuration.

Sleepy

In order to configure custom properties for this policy, the name you must use is `sleepy`.

`policy.sleepy.sleep` Max sleep time (in ms) to wait to look for files in the FS. Once an execution has finished, the policy will sleep during this time to be executed again.

- Type: long
- Importance: high

`policy.sleepy.fraction` Sleep fraction to divide the sleep time to allow interrupting the policy faster.

- Type: long
- Default: `10`
- Importance: medium

`policy.sleepy.max_execs` Max executions allowed (negative to disable). After exceeding this number, the policy will end. An execution represents: listing files from the FS and its corresponding sleep time.

- Type: long
- Default: `-1`
- Importance: medium

Cron

In order to configure custom properties for this policy, the name you must use is `cron`.

`policy.cron.expression` Cron expression to schedule the policy.

- Type: string
- Importance: high

`policy.cron.end_date` End date to finish the policy with ISO date-time format.

- Type: date
- Default: `null`
- Importance: medium

HDFS file watcher

In order to configure custom properties for this policy, the name you must use is `hdfs_file_watcher`.

`policy.hdfs_file_watcher.poll` Time to wait (in milliseconds) until the records retrieved from the file watcher will be sent to the source task.

- Type: long
- Default: `5000`
- Importance: medium

`policy.hdfs_file_watcher.retry` Sleep time to retry connections to HDFS in case of connection errors happened.

- Type: long
- Default: `20000`

- Importance: medium

S3 event notifications

In order to configure custom properties for this policy, the name you must use is `s3_event_notifications`.

`policy.s3_event_notifications.queue` SQS queue name to retrieve messages from.

- Type: string
- Importance: high

`policy.s3_event_notifications.poll` Time to wait (in milliseconds) until the records retrieved from the queue will be sent to the source task.

- Type: long
- Default: `5000`
- Importance: medium

`policy.s3_event_notifications.event_regex` Regular expression to filter event based on their types.

- Type: string
- Default: `.*`
- Importance: medium

`policy.s3_event_notifications.delete_messages` If messages from SQS should be removed after reading them.

- Type: boolean
- Default: `true`
- Importance: medium

`policy.s3_event_notifications.max_messages` Maximum number of messages to retrieve at a time (must be between 1 and 10).

- Type: int
- Importance: medium

`policy.s3_event_notifications.visibility_timeout` Duration (in seconds) that the received messages are hidden from subsequent retrieve requests.

- Type: int
- Importance: low

1.2.3 File readers

Some file readers have custom properties to define and others don't. So, depending on the configuration you'll have to take into account their properties.

Parquet

In order to configure custom properties for this reader, the name you must use is `parquet`.

`file_reader.parquet.schema` Avro schema in JSON format to use when reading a file.

- Type: string
- Importance: medium

file_reader.parquet.projection Avro schema in JSON format to use for projecting fields from records in a file.

- Type: string
- Importance: medium

Avro

In order to configure custom properties for this reader, the name you must use is `avro`.

file_reader.avro.schema Avro schema in JSON format to use when reading a file. If not specified, the reader will use the schema defined in the file.

- Type: string
- Importance: medium

ORC

In order to configure custom properties for this reader, the name you must use is `orc`.

file_reader.orc.use_zerocopy Use zero-copy when reading a ORC file.

- Type: boolean
- Default: `false`
- Importance: medium

file_reader.orc.skip_corrupt_records If reader will skip corrupt data or not. If disabled, an exception will be thrown when there is corrupted data in the file.

- Type: boolean
- Default: `false`
- Importance: medium

SequenceFile

In order to configure custom properties for this reader, the name you must use is `sequence`.

file_reader.sequence.field_name.key Custom field name for the output key to include in the Kafka message.

- Type: string
- Default: `key`
- Importance: medium

file_reader.sequence.field_name.value Custom field name for the output value to include in the Kafka message.

- Type: string
- Default: `value`

- Importance: medium

file_reader.sequence.buffer_size Custom buffer size to read data from the Sequence file.

- Type: int
- Default: `4096`
- Importance: low

Cobol

In order to configure custom properties for this reader, the name you must use is `cobol`.

file_reader.cobol.copybook.content The content of the copybook. It is mandatory if property `file_reader.cobol.copybook.path` is not set.

- Type: string
- Default: `null`
- Importance: high

file_reader.cobol.copybook.path Copybook file path in the file system to be used. It is mandatory if property `file_reader.cobol.copybook.content` is not set.

- Type: string
- Default: `null`
- Importance: high

file_reader.cobol.reader.is_ebcdic If the input data file encoding is EBCDIC, otherwise it is ASCII.

- Type: boolean
- Default: `true`
- Importance: medium

file_reader.cobol.reader.is_text If line ending characters will be used (LF / CRLF) as the record separator.

- Type: boolean
- Default: `false`
- Importance: medium

file_reader.cobol.reader.ebcdic_code_page Code page to be used for EBCDIC to ASCII / Unicode conversions.

- Type: string
- Default: `common`
- Importance: medium

file_reader.cobol.reader.is_record_sequence If the input file has 4 byte record length headers.

- Type: boolean
- Default: `false`
- Importance: medium

file_reader.cobol.reader.floating_point_format Format used for the floating-point numbers.

- Type: enum (available values `ibm`, `ibm_little_endian`, `ieee754`, and `ieee754_little_endian`)
- Default: `ibm`
- Importance: medium

`file_reader.cobol.reader.schema_policy` Specifies a policy to transform the input schema.

- Type: enum (available values `keep_original` and `collapse_root`)
- Default: `keep_original`
- Importance: medium

`file_reader.cobol.reader.string_trimming_policy` The trim to apply for records with string data types.

- Type: enum (available values `both`, `left`, `right` and `none`)
- Default: `both`
- Importance: medium

`file_reader.cobol.reader.start_offset` An offset to the start of the record in each binary data block.

- Type: int
- Default: `0`
- Importance: medium

`file_reader.cobol.reader.end_offset` An offset from the end of the record to the end of the binary data block.

- Type: int
- Default: `0`
- Importance: medium

`file_reader.cobol.reader.file_start_offset` A number of bytes to skip at the beginning of each file.

- Type: int
- Default: `0`
- Importance: medium

`file_reader.cobol.reader.file_end_offset` A number of bytes to skip at the end of each file.

- Type: int
- Default: `0`
- Importance: medium

`file_reader.cobol.reader.ebcdic_code_page_class` Custom code page conversion class provided.

- Type: string
- Default: `null`
- Importance: low

`file_reader.cobol.reader.ascii_charset` Charset for ASCII data.

- Type: string

- Default: ""
- Importance: low

file_reader.cobol.reader.is_uft16_big_endian Flag to consider UTF-16 strings as big-endian.

- Type: boolean
- Default: `true`
- Importance: low

file_reader.cobol.reader.variable_size_occurs If true, occurs depending on data size will depend on the number of elements.

- Type: boolean
- Default: `false`
- Importance: low

file_reader.cobol.reader.record_length Specifies the length of the record disregarding the copybook record size. Implied the file has fixed record length.

- Type: int
- Default: `null`
- Importance: low

file_reader.cobol.reader.length_field_name The name for a field that contains the record length. If not set, the copybook record length will be used.

- Type: string
- Default: `null`
- Importance: low

file_reader.cobol.reader.is_rdw_big_endian If the RDW is big endian.

- Type: boolean
- Default: `false`
- Importance: low

file_reader.cobol.reader.is_rdw_part_rec_length If the RDW count itself as part of record length itself.

- Type: boolean
- Default: `false`
- Importance: low

file_reader.cobol.reader.rdw_adjustment Controls a mismatch between RDW and record length.

- Type: int
- Default: `0`
- Importance: low

file_reader.cobol.reader.is_index_generation_needed If the indexing input file before processing is requested.

- Type: boolean
- Default: `false`

- Importance: low

file_reader.cobol.reader.input_split_records The number of records to include in each partition.

- Type: int
- Default: `null`
- Importance: low

file_reader.cobol.reader.input_split_size_mb A partition size to target.

- Type: int
- Default: `null`
- Importance: low

file_reader.cobol.reader.hdfs_default_block_size Default HDFS block size for the HDFS filesystem used.

- Type: int
- Default: `null`
- Importance: low

file_reader.cobol.reader.drop_group_fillers If true the parser will drop all FILLER fields, even GROUP FILLERS that have non-FILLER nested fields.

- Type: boolean
- Default: `false`
- Importance: low

file_reader.cobol.reader.drop_value_fillers If true the parser will drop all value FILLER fields.

- Type: boolean
- Default: `true`
- Importance: low

file_reader.cobol.reader.non_terminals A comma-separated list of group-type fields to combine and parse as primitive fields.

- Type: string[]
- Default: `null`
- Importance: low

file_reader.cobol.reader.debug_fields_policy Specifies if debugging fields need to be added and what should they contain.

- Type: enum (available values `hex`, `raw` and `none`)
- Default: `none`
- Importance: low

file_reader.cobol.reader.record_header_parser Parser to be used to parse data field record headers.

- Type: string
- Default: `null`
- Importance: low

file_reader.cobol.reader.record_extractor Parser to be used to parse records.

- Type: string
- Default: `null`
- Importance: low

file_reader.cobol.reader.rhp_additional_info Extra option to be passed to a custom record header parser.

- Type: string
- Default: `null`
- Importance: low

file_reader.cobol.reader.re_additional_info A string provided for the raw record extractor.

- Type: string
- Default: ““
- Importance: low

file_reader.cobol.reader.input_file_name_column A column name to add to each record containing the input file name.

- Type: string
- Default: ““
- Importance: low

Binary

There are no extra configuration options for this file reader.

CSV

To configure custom properties for this reader, the name you must use is `delimited` (even though it's for CSV).

file_reader.delimited.settings.format.delimiter Field delimiter.

- Type: string
- Default: `,`
- Importance: high

file_reader.delimited.settings.header If the file contains header or not.

- Type: boolean
- Default: `false`
- Importance: high

file_reader.delimited.settings.schema A comma-separated list of ordered data types for each field in the file. Possible values: `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, `bytes` and `string`)

- Type: string[]
- Default: `null`
- Importance: medium

file_reader.delimited.settings.data_type_mapping_error Flag to enable/disable throwing errors when mapping data types based on the schema is not possible. If disabled, the returned value which could not be mapped will be `null`.

- Type: boolean
- Default: `true`
- Importance: medium

file_reader.delimited.settings.allow_nulls If the schema supports nullable fields. If `file_reader.delimited.settings.data_type_mapping_error` config flag is disabled, the value set for this config will be ignored and set to `true`.

- Type: boolean
- Default: `false`
- Importance: medium

file_reader.delimited.settings.header_names A comma-separated list of ordered field names to set when reading a file.

- Type: string[]
- Default: `null`
- Importance: medium

file_reader.delimited.settings.null_value Default value for `null` values.

- Type: string
- Default: `null`
- Importance: medium

file_reader.delimited.settings.empty_value Default value for empty values (empty values within quotes).

- Type: string
- Default: `null`
- Importance: medium

file_reader.delimited.settings.format.line_separator Line separator to be used.

- Type: string
- Default: `\n`
- Importance: medium

file_reader.delimited.settings.max_columns Default value for `null` values.

- Type: int
- Default: `512`
- Importance: low

file_reader.delimited.settings.max_chars_per_column Default value for `null` values.

- Type: int
- Default: `4096`
- Importance: low

file_reader.delimited.settings.rows_to_skip Number of rows to skip.

- Type: long
- Default: `0`
- Importance: low

file_reader.delimited.settings.line_separator_detection If the reader should detect the line separator automatically.

- Type: boolean
- Default: `false`
- Importance: low

file_reader.delimited.settings.delimiter_detection If the reader should detect the delimiter automatically.

- Type: boolean
- Default: `false`
- Importance: low

file_reader.delimited.settings.ignore_leading_whitespaces Flag to enable/disable skipping leading whitespaces from values.

- Type: boolean
- Default: `true`
- Importance: low

file_reader.delimited.settings.ignore_trailing_whitespaces Flag to enable/disable skipping trailing whitespaces from values.

- Type: boolean
- Default: `true`
- Importance: low

file_reader.delimited.settings.format.comment Character that represents a line comment at the beginning of a line.

- Type: char
- Default: #
- Importance: low

file_reader.delimited.settings.escape_unquoted Flag to enable/disable processing escape sequences in unquoted values.

- Type: boolean
- Default: `false`
- Importance: low

file_reader.delimited.settings.format.quote Character used for escaping values where the field delimiter is part of the value.

- Type: char
- Default: "

- Importance: low

file_reader.delimited.settings.format.quote_escape Character used for escaping quotes inside an already quoted value.

- Type: char
- Default: "
- Importance: low

file_reader.delimited.encoding Encoding to use for reading a file. If not specified, the reader will use the default encoding.

- Type: string
- Default: based on the locale and charset of the underlying operating system.
- Importance: medium

file_reader.delimited.compression.type Compression type to use when reading a file.

- Type: enum (available values `bzip2`, `gzip` and `none`)
- Default: `none`
- Importance: medium

file_reader.delimited.compression.concatenated Flag to specify if the decompression of the reader will finish at the end of the file or after the first compressed stream.

- Type: boolean
- Default: `true`
- Importance: low

TSV

To configure custom properties for this reader, the name you must use is `delimited` (even though it's for TSV).

file_reader.delimited.settings.header If the file contains header or not.

- Type: boolean
- Default: `false`
- Importance: high

file_reader.delimited.settings.schema A comma-separated list of ordered data types for each field in the file. Possible values: `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, `bytes` and `string`)

- Type: string[]
- Default: `null`
- Importance: medium

file_reader.delimited.settings.data_type_mapping_error Flag to enable/disable throwing errors when mapping data types based on the schema is not possible. If disabled, the returned value which could not be mapped will be `null`.

- Type: boolean
- Default: `true`
- Importance: medium

file_reader.delimited.settings.allow_nulls If the schema supports nullable fields. If `file_reader.delimited.settings.data_type_mapping_error` config flag is disabled, the value set for this config will be ignored and set to `true`.

- Type: boolean
- Default: `false`
- Importance: medium

file_reader.delimited.settings.header_names A comma-separated list of ordered field names to set when reading a file.

- Type: string[]
- Default: `null`
- Importance: medium

file_reader.delimited.settings.null_value Default value for `null` values.

- Type: string
- Default: `null`
- Importance: medium

file_reader.delimited.settings.format.line_separator Line separator to be used.

- Type: string
- Default: `\n`
- Importance: medium

file_reader.delimited.settings.max_columns Default value for `null` values.

- Type: int
- Default: `512`
- Importance: low

file_reader.delimited.settings.max_chars_per_column Default value for `null` values.

- Type: int
- Default: `4096`
- Importance: low

file_reader.delimited.settings.rows_to_skip Number of rows to skip.

- Type: long
- Default: `0`
- Importance: low

file_reader.delimited.settings.line_separator_detection If the reader should detect the line separator automatically.

- Type: boolean
- Default: `false`
- Importance: low

file_reader.delimited.settings.line_joining Identifies whether or lines ending with the escape character and followed by a line separator character should be joined with the following line.

- Type: boolean
- Default: `true`
- Importance: low

`file_reader.delimited.settings.ignore_leading_whitespaces` Flag to enable/disable skipping leading whitespaces from values.

- Type: boolean
- Default: `true`
- Importance: low

`file_reader.delimited.settings.ignore_trailing_whitespaces` Flag to enable/disable skipping trailing whitespaces from values.

- Type: boolean
- Default: `true`
- Importance: low

`file_reader.delimited.settings.format.comment` Character that represents a line comment at the beginning of a line.

- Type: char
- Default: `#`
- Importance: low

`file_reader.delimited.settings.format.escape` Character used for escaping special characters.

- Type: char
- Default: `\`
- Importance: low

`file_reader.delimited.settings.format.escaped_char` Character used to represent an escaped tab.

- Type: char
- Default: `t`
- Importance: low

`file_reader.delimited.encoding` Encoding to use for reading a file. If not specified, the reader will use the default encoding.

- Type: string
- Default: based on the locale and charset of the underlying operating system.
- Importance: medium

`file_reader.delimited.compression.type` Compression type to use when reading a file.

- Type: enum (available values `bzip2`, `gzip` and `none`)
- Default: `none`
- Importance: medium

`file_reader.delimited.compression.concatenated` Flag to specify if the decompression of the reader will finish at the end of the file or after the first compressed stream.

- Type: boolean
- Default: `true`
- Importance: low

FixedWidth

To configure custom properties for this reader, the name you must use is `delimited` (even though it's for FixedWidth).

`file_reader.delimited.settings.field_lengths` A comma-separated ordered list of integers with the lengths of each field.

- Type: int[]
- Importance: high

`file_reader.delimited.settings.header` If the file contains header or not.

- Type: boolean
- Default: `false`
- Importance: high

`file_reader.delimited.settings.schema` A comma-separated list of ordered data types for each field in the file. Possible values: `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, `bytes` and `string`)

- Type: string[]
- Default: `null`
- Importance: medium

`file_reader.delimited.settings.data_type_mapping_error` Flag to enable/disable throwing errors when mapping data types based on the schema is not possible. If disabled, the returned value which could not be mapped will be `null`.

- Type: boolean
- Default: `true`
- Importance: medium

`file_reader.delimited.settings.allow_nulls` If the schema supports nullable fields. If `file_reader.delimited.settings.data_type_mapping_error` config flag is disabled, the value set for this config will be ignored and set to `true`.

- Type: boolean
- Default: `false`
- Importance: medium

`file_reader.delimited.settings.header_names` A comma-separated list of ordered field names to set when reading a file.

- Type: string[]
- Default: `null`
- Importance: medium

`file_reader.delimited.settings.keep_padding` If the padding character should be kept in each value.

- Type: boolean
- Default: `false`
- Importance: medium

`file_reader.delimited.settings.padding_for_headers` If headers have the default padding specified.

- Type: boolean
- Default: `true`
- Importance: medium

`file_reader.delimited.settings.null_value` Default value for `null` values.

- Type: string
- Default: `null`
- Importance: medium

`file_reader.delimited.settings.format.ends_on_new_line` Line separator to be used.

- Type: boolean
- Default: `true`
- Importance: medium

`file_reader.delimited.settings.format.line_separator` Line separator to be used.

- Type: string
- Default: `\n`
- Importance: medium

`file_reader.delimited.settings.format.padding` The padding character used to represent unwritten spaces.

- Type: char
- Default: ‘‘ ‘‘
- Importance: medium

`file_reader.delimited.settings.max_columns` Default value for `null` values.

- Type: int
- Default: `512`
- Importance: low

`file_reader.delimited.settings.max_chars_per_column` Default value for `null` values.

- Type: int
- Default: `4096`
- Importance: low

`file_reader.delimited.settings.skip_trailing_chars` If the trailing characters beyond the record's length should be skipped.

- Type: boolean
- Default: `false`

- Importance: low

file_reader.delimited.settings.rows_to_skip Number of rows to skip.

- Type: long
- Default: `0`
- Importance: low

file_reader.delimited.settings.line_separator_detection If the reader should detect the line separator automatically.

- Type: boolean
- Default: `false`
- Importance: low

file_reader.delimited.settings.ignore_leading_whitespaces Flag to enable/disable skipping leading whitespaces from values.

- Type: boolean
- Default: `true`
- Importance: low

file_reader.delimited.settings.ignore_trailing_whitespaces Flag to enable/disable skipping trailing whitespaces from values.

- Type: boolean
- Default: `true`
- Importance: low

file_reader.delimited.settings.format.comment Character that represents a line comment at the beginning of a line.

- Type: char
- Default: `#`
- Importance: low

file_reader.delimited.encoding Encoding to use for reading a file. If not specified, the reader will use the default encoding.

- Type: string
- Default: based on the locale and charset of the underlying operating system.
- Importance: medium

file_reader.delimited.compression.type Compression type to use when reading a file.

- Type: enum (available values `bzip2`, `gzip` and `none`)
- Default: `none`
- Importance: medium

file_reader.delimited.compression.concatenated Flag to specify if the decompression of the reader will finish at the end of the file or after the first compressed stream.

- Type: boolean
- Default: `true`

- Importance: low

JSON

To configure custom properties for this reader, the name you must use is `json`.

`file_reader.json.record_per_line` If enabled, the reader will read each line as a record. Otherwise, the reader will read the full content of the file as a record.

- Type: boolean
- Default: `true`
- Importance: medium

`file_reader.json.deserialization.<deserialization_feature>` Deserialization feature to use when reading a JSON file. You can add as much as you like based on the ones defined here.

- Type: boolean
- Importance: medium

`file_reader.json.encoding` Encoding to use for reading a file. If not specified, the reader will use the default encoding.

- Type: string
- Default: based on the locale and charset of the underlying operating system.
- Importance: medium

`file_reader.json.compression.type` Compression type to use when reading a file.

- Type: enum (available values `bzip2`, `gzip` and `none`)
- Default: `none`
- Importance: medium

`file_reader.json.compression.concatenated` Flag to specify if the decompression of the reader will finish at the end of the file or after the first compressed stream.

- Type: boolean
- Default: `true`
- Importance: low

XML

To configure custom properties for this reader, the name you must use is `xml`.

`file_reader.xml.record_per_line` If enabled, the reader will read each line as a record. Otherwise, the reader will read the full content of the file as a record.

- Type: boolean
- Default: `true`
- Importance: medium

`file_reader.xml.deserialization.<deserialization_feature>` Deserialization feature to use when reading a XML file. You can add as much as you like based on the ones defined here.

- Type: boolean

- Importance: medium

file_reader.xml.encoding Encoding to use for reading a file. If not specified, the reader will use the default encoding.

- Type: string
- Default: based on the locale and charset of the underlying operating system.
- Importance: medium

file_reader.xml.compression.type Compression type to use when reading a file.

- Type: enum (available values `bzip2`, `gzip` and `none`)
- Default: `none`
- Importance: medium

file_reader.xml.compression.concatenated Flag to specify if the decompression of the reader will finish at the end of the file or after the first compressed stream.

- Type: boolean
- Default: `true`
- Importance: low

YAML

To configure custom properties for this reader, the name you must use is `yaml`.

file_reader.yaml.deserialization.<deserialization_feature> Deserialization feature to use when reading a YAML file. You can add as much as you like based on the ones defined here.

- Type: boolean
- Importance: medium

file_reader.yaml.encoding Encoding to use for reading a file. If not specified, the reader will use the default encoding.

- Type: string
- Default: based on the locale and charset of the underlying operating system.
- Importance: medium

file_reader.yaml.compression.type Compression type to use when reading a file.

- Type: enum (available values `bzip2`, `gzip` and `none`)
- Default: `none`
- Importance: medium

file_reader.yaml.compression.concatenated Flag to specify if the decompression of the reader will finish at the end of the file or after the first compressed stream.

- Type: boolean
- Default: `true`
- Importance: low

Text

To configure custom properties for this reader, the name you must use is `text`.

`file_reader.text.record_per_line` If enabled, the reader will read each line as a record. Otherwise, the reader will read the full content of the file as a record.

- Type: boolean
- Default: `true`
- Importance: medium

`file_reader.text.field_name.value` Custom field name for the output value to include in the Kafka message.

- Type: string
- Default: `value`
- Importance: medium

`file_reader.text.encoding` Encoding to use for reading a file. If not specified, the reader will use the default encoding.

- Type: string
- Default: based on the locale and charset of the underlying operating system.
- Importance: medium

`file_reader.text.compression.type` Compression type to use when reading a file.

- Type: enum (available values `bzip2`, `gzip` and `none`)
- Default: `none`
- Importance: medium

`file_reader.text.compression.concatenated` Flag to specify if the decompression of the reader will finish at the end of the file or after the first compressed stream.

- Type: boolean
- Default: `true`
- Importance: low

Agnostic

To configure custom properties for this reader, the name you must use is `agnostic`.

`file_reader.agnostic.extensions.parquet` A comma-separated string list with the accepted extensions for Parquet files.

- Type: string[]
- Default: `parquet`
- Importance: medium

`file_reader.agnostic.extensions.avro` A comma-separated string list with the accepted extensions for Avro files.

- Type: string[]
- Default: `avro`

- Importance: medium

file_reader.agnostic.extensions.orc A comma-separated string list with the accepted extensions for ORC files.

- Type: string[]
- Default: `orc`
- Importance: medium

file_reader.agnostic.extensions.sequence A comma-separated string list with the accepted extensions for Sequence files.

- Type: string[]
- Default: `seq`
- Importance: medium

file_reader.agnostic.extensions.cobol A comma-separated string list with the accepted extensions for Cobol files.

- Type: string[]
- Default: `dat`
- Importance: medium

file_reader.agnostic.extensions.binary A comma-separated string list with the accepted extensions for binary files.

- Type: string[]
- Default: `bin`
- Importance: medium

file_reader.agnostic.extensions.csv A comma-separated string list with the accepted extensions for CSV files.

- Type: string[]
- Default: `csv`
- Importance: medium

file_reader.agnostic.extensions.tsv A comma-separated string list with the accepted extensions for TSV files.

- Type: string[]
- Default: `tsv`
- Importance: medium

file_reader.agnostic.extensions.fixed A comma-separated string list with the accepted extensions for fixed-width files.

- Type: string[]
- Default: `fixed`
- Importance: medium

file_reader.agnostic.extensions.json A comma-separated string list with the accepted extensions for JSON files.

- Type: string[]

- Default: `json`
- Importance: medium

`file_reader.agnostic.extensions.xml` A comma-separated string list with the accepted extensions for XML files.

- Type: string[]
- Default: `xml`
- Importance: medium

`file_reader.agnostic.extensions.yaml` A comma-separated string list with the accepted extensions for YAML files.

- Type: string[]
- Default: `yaml`
- Importance: medium

Note: The Agnostic reader uses the previous ones as inner readers. So, in case of using this reader, you'll probably need to include also the specified properties for those readers in the connector configuration as well.

1.3 FAQs

My file was already processed and the connector, when it's executed again, processes the same records again.

If during the previous executions the records were sent successfully to Kafka, their offsets were sent too. Then, when executing the policy again, it retrieves the offset and seeks the file. If this didn't happen, it's possible that the offset was not committed yet and, consequently, the offset retrieved is non-existent or too old.

Have a look when the offsets are committed in Kafka and/or try to execute the policy when you are sure the offsets have been committed.

The connector started but does not process any kind of file.

This can be for several reasons:

- Check if the files contained in the FS match the regexp provided.
- Check if there is any kind of problem with the FS. The connector tolerates FS connection exceptions to process them later but in log files you'll find these possible errors.
- The file reader is reading files with an invalid format so it cannot process the file and continues with the next one. You can see this as an error in the log.

I have directories in the FS created day by day and I have to modify the connector everyday.

Don't do this! Take advantage of the dynamic URIs using expressions.

For instance, if you have this URI `hdfs://host:9000/data/2020`, you can use this URI `hdfs://host:9000/data/${yyyy}` instead.

The connector is too slow to process all URIs I have.

Obviously, this depends of the files in the FS(s) but having several URIs in the connector might be a good idea to adjust the number of tasks to process those URIs in parallel (`tasks.max` connector property).

Also, using the properties `policy.batch_size` and/or `file_reader.batch_size` in case you have tons of files or files too large might help.

I removed a file from the FS but the connector is still sending messages with the contents of that file.

This is a tricky issue. The file reader is an iterator and processes record by record but part of the file is buffered and, even though the file was removed from the FS, the file reader continues producing records until throws an exception. It's a matter of time.

But the main thing is that you don't have to worry about removing files from the FS when they are being processed. The connector tolerates errors when reading files and continues with the next file.