Spring Cloud Data Flow Admin for Cloud Foundry

1.0.0.M1

Table of Contents

Part I. Spring Cloud Data Flow for Cloud Foundry

This project provides support for deploying Spring Cloud Stream modules to Cloud Foundry.

Part II. Deploying on Cloud Foundry

> **Note**
>
> The Cloud Foundry SPI implementation is a separate [project](#).

Spring Cloud Data Flow can be used to deploy modules in a Cloud Foundry environment. When doing so, the [Admin](#) application can either run itself on Cloud Foundry, or on another installation (e.g. a simple laptop).

The required configuration amounts to the same in either case, and is merely related to providing credentials to the Cloud Foundry instance so that the admin can spawn applications itself. Any Spring Boot compatible configuration mechanism can be used (passing program arguments, editing configuration files before building the application, using [Spring Cloud Config](#), using environment variables, etc.), although some may prove more practicable than others when running *on* Cloud Foundry.

1. provision a redis service instance on Cloud Foundry. Use `cf marketplace` to discover which plans are available to you, depending on the details of your Cloud Foundry setup. For example when using [Pivotal Web Services](#):

```
cf create-service rediscloud 30mb redis
```

2. download the Spring Cloud Data Flow Admin and Shell apps:

```
wget http://repo.spring.io/milestone/org/springframework/cloud/spring-cloud-dataflow-admin/1.0.0.M2/
spring-cloud-dataflow-admin-cloudfoundry-1.0.0.M2.jar
wget http://repo.spring.io/milestone/org/springframework/cloud/spring-cloud-dataflow-shell/1.0.0.M2/
spring-cloud-dataflow-shell-1.0.0.M2.jar
```

1. Deploying Admin app on Cloud Foundry

3a. push the admin application on Cloud Foundry, configure it (see below) and start it

> **Note**
>
> You must use a unique name for your app; an app with the same name in the same organization
> will cause your deployment to fail

```
cf push s-c-dataflow-admin --no-start -p spring-cloud-dataflow-admin-cloudfoundry-1.0.0.M2.jar
cf bind-service s-c-dataflow-admin redis
```

Now we can configure the app. This configuration is for Pivotal Web Services. You need to fill in {org},
{space}, {email} and {password} before running these commands.

> **Note**
>
> Only set 'Skip SSL Validation' to true if you're running on a Cloud Foundry instance using self-
> signed certs (e.g. in development). Do not use for production.

```
cf set-env s-c-dataflow-admin CLOUDFOUNDRY_API_ENDPOINT https://api.run.pivotal.io
cf set-env s-c-dataflow-admin CLOUDFOUNDRY_ORGANIZATION {org}
cf set-env s-c-dataflow-admin CLOUDFOUNDRY_SPACE {space}
cf set-env s-c-dataflow-admin CLOUDFOUNDRY_DOMAIN cfapps.io
cf set-env s-c-dataflow-admin CLOUDFOUNDRY_SERVICES redis
cf set-env s-c-dataflow-admin CLOUDFOUNDRY_USERNAME {email}
cf set-env s-c-dataflow-admin CLOUDFOUNDRY_PASSWORD {password}
cf set-env s-c-dataflow-admin CLOUDFOUNDRY_SKIP_SSL_VALIDATION false
```

We are now ready to start the app.

```
cf start s-c-dataflow-admin
```

Alternatively,

2. Running Admin app locally

3b. run the admin application locally, targeting your Cloud Foundry installation

First you need to configure the application either by passing in command line arguments (see below) or setting a number of environment variables.

To use environment variables set the following:

```
export CLOUDFOUNDRY_API_ENDPOINT=https://api.run.pivotal.io
export CLOUDFOUNDRY_ORGANIZATION={org}
export CLOUDFOUNDRY_SPACE={space}
export CLOUDFOUNDRY_DOMAIN=cfapps.io
export CLOUDFOUNDRY_SERVICES=redis
export CLOUDFOUNDRY_USERNAME={email}
export CLOUDFOUNDRY_PASSWORD={password}
export CLOUDFOUNDRY_SKIP_SSL_VALIDATION=false
```

You need to fill in {org}, {space}, {email} and {password} before running these commands.

Note

Only set 'Skip SSL Validation' to true if you're running on a Cloud Foundry instance using self-signed certs (e.g. in development). Do not use for production.

Now we are ready to start the admin application:

```
java -jar spring-cloud-dataflow-admin-cloudfoundry-1.0.0.M2.jar [--option1=value1] [--option2=value2]
 [etc.]
```

3. Running Spring Cloud Data Flow Shell locally

1. run the shell and optionally target the Admin application if not running on the same host (will typically
   be the case if deployed on Cloud Foundry as **3a.**)

```
$ java -jar spring-cloud-dataflow-shell-1.0.0.M2.jar
```

```
server-unknown:>admin config server http://s-c-dataflow-admin.cfapps.io
Successfully targeted http://s-c-dataflow-admin.cfapps.io
dataflow:>
```

4. Spring Cloud Data Flow Admin app configuration settings for Cloud Foundry

The following pieces of configuration must be provided, e.g. by setting variables in the apps environment, or passing variables on the Java invocation:

```
# Default values cited after the equal sign.
# Example values, typical for Pivotal Web Services, cited as a comment

# url of the CF API (used when using cf login -a for example), e.g. https://api.run.pivotal.io
# (for setting env var use CLOUDFOUNDRY_API_ENDPOINT)
cloudfoundry.apiEndpoint=

# name of the organization that owns the space above, e.g. youruser-org
# (for setting env var use CLOUDFOUNDRY_ORGANIZATION)
cloudfoundry.organization=

# name of the space into which modules will be deployed
# (for setting env var use CLOUDFOUNDRY_SPACE)
cloudfoundry.space=<same space as admin when running on CF, or 'development'>

# the root domain to use when mapping routes, e.g. cfapps.io
# (for setting env var use CLOUDFOUNDRY_DOMAIN)
cloudfoundry.domain=

# Comma separated set of service instance names to bind to the module.
# Amongst other things, this should include a service that will be used
# for Spring Cloud Stream binding
# (for setting env var use CLOUDFOUNDRY_SERVICES)
cloudfoundry.services=redis

# username and password of the user to use to create apps (modules)
# (for setting env var use CLOUDFOUNDRY_USERNAME and CLOUDFOUNDRY_PASSWORD)
cloudfoundry.username=
cloudfoundry.password=

# Whether to allow self-signed certificates during SSL validation
# (for setting env var use CLOUDFOUNDRY_SKIP_SSL_VALIDATION)
cloudfoundry.skipSslValidation=false
```

Part III. Modules

5. Sources

5.1 File

The file source provides the contents of a File as a byte array by default. However, this can be customized using the `--mode` option:

* **ref** Provides a `java.io.File` reference

* **lines** Will split files line-by-line and emit a new message for each line

* **contents** The default. Provides the contents of a file as a byte array

When using `--mode=lines`, you can also provide the additional option `--withMarkers=true`. If set to `true`, the underlying `FileSplitter` will emit additional *start-of-file* and *end-of-file* marker messages before and after the actual data. The payload of these 2 additional marker messages is of type `FileSplitter.FileMarker`. The option `withMarkers` defaults to `false` if not explicitly set.

Options

The **file** source has the following options:

dir
> the absolute path to the directory to monitor for files **(String, default: ``)**

fixedDelay
> the fixed delay polling interval specified in seconds **(int, default: `5`)**

initialDelay
> an initial delay when using a fixed delay trigger, expressed in TimeUnits (seconds by default) **(int, default: `0`)**

maxMessages
> the maximum messages per poll; -1 for unlimited **(long, default: `-1`)**

mode
> specifies how the file is being read. By default the content of a file is provided as byte array **(FileReadingMode, default: `contents`, possible values: `ref,lines,contents`)**

pattern
> a filter expression (Ant style) to accept only files that match the pattern **(String, default: *)**

preventDuplicates
> whether to prevent the same file from being processed twice **(boolean, default: `true`)**

timeUnit
> the time unit for the fixed and initial delays **(String, default: `SECONDS`)**

withMarkers
> if true emits start of file/end of file marker messages before/after the data. Only valid with FileReadingMode 'lines' **(Boolean, no default)**

The `ref` option is useful in some cases in which the file contents are large and it would be more efficient to send the file path.

5.2 FTP (`ftp`)

This source module supports transfer of files using the FTP protocol. Files are transferred from the `remote` directory to the `local` directory where the module is deployed. Messages emitted by the source are provided as a byte array by default. However, this can be customized using the `--mode` option:

- **ref** Provides a `java.io.File` reference

- **lines** Will split files line-by-line and emit a new message for each line

- **contents** The default. Provides the contents of a file as a byte array

When using `--mode=lines`, you can also provide the additional option `--withMarkers=true`. If set to `true`, the underlying `FileSplitter` will emit additional *start-of-file* and *end-of-file* marker messages before and after the actual data. The payload of these 2 additional marker messages is of type `FileSplitter.FileMarker`. The option `withMarkers` defaults to `false` if not explicitly set.

Options

The **ftp** source has the following options:

autoCreateLocalDir
    local directory must be auto created if it does not exist **(boolean, default: `true`)**

clientMode
    client mode to use : 2 for passive mode and 0 for active mode **(int, default: `0`)**

deleteRemoteFiles
    delete remote files after transfer **(boolean, default: `false`)**

filenamePattern
    simple filename pattern to apply to the filter **(String, default: *)**

fixedDelay
    the rate at which to poll the remote directory **(int, default: `1`)**

host
    the host name for the FTP server **(String, default: `localhost`)**

initialDelay
    an initial delay when using a fixed delay trigger, expressed in TimeUnits (seconds by default) **(int, default: `0`)**

localDir
    set the local directory the remote files are transferred to **(String, default: ``)**

maxMessages
    the maximum messages per poll; -1 for unlimited **(long, default: `-1`)**

mode
    specifies how the file is being read. By default the content of a file is provided as byte array **(FileReadingMode, default: `contents`, possible values: `ref,lines,contents`)**

password
    the password for the FTP connection **(Password, no default)**

port
    the port for the FTP server **(int, default: `21`)**

preserveTimestamp
    whether to preserve the timestamp of files retrieved **(boolean, default: `true`)**

remoteDir
    the remote directory to transfer the files from **(String, default: `/`)**

remoteFileSeparator
    file separator to use on the remote side **(String, default: `/`)**

timeUnit
    the time unit for the fixed and initial delays **(String, default: `SECONDS`)**

tmpFileSuffix
    extension to use when downloading files **(String, default: `.tmp`)**

username
    the username for the FTP connection **(String, no default)**

withMarkers
    if true emits start of file/end of file marker messages before/after the data. Only valid with
    FileReadingMode 'lines' **(Boolean, no default)**

5.3 HTTP (`http`)

A source module that listens for HTTP requests and emits the body as a message payload. If the
Content-Type matches 'text/*' or 'application/json', the payload will be a String, otherwise the payload
will be a byte array.

To create a stream definition in the server using the Spring Cloud Data Flow shell

```
dataflow:> stream create --name httptest --definition "http --server.port=9000 | log" --deploy
```

Post some data to the http server on port 9000

```
dataflow:> http post --target http://localhost:9000 --data "hello world"
```

See if the data ended up in the log.

5.4 Load Generator (`load-generator`)

A source that sends generated data and dispatches it to the stream. This is to provide a method for users
to identify the performance of Spring Cloud Data Flow in different environments and deployment types.

Options

The **load-generator** source has the following options:

messageCount
    the number of messages to send **(Integer, default: `100`)**

messageSize
    the size of message to send **(Integer, `1000`)**

producers
    the number of producers **(Integer, `1`)**

outputType
    how this module should emit messages it produces **(MimeType, default: no default)**

5.5 SFTP (`sftp`)

This source module supports transfer of files using the SFTP protocol. Files are transferred from the `remote` directory to the `local` directory where the module is deployed.

Messages emitted by the source are provided as a byte array by default. However, this can be customized using the `--mode` option:

- **ref** Provides a `java.io.File` reference

- **lines** Will split files line-by-line and emit a new message for each line

- **contents** The default. Provides the contents of a file as a byte array

When using `--mode=lines`, you can also provide the additional option `--withMarkers=true`. If set to `true`, the underlying `FileSplitter` will emit additional *start-of-file* and *end-of-file* marker messages before and after the actual data. The payload of these 2 additional marker messages is of type `FileSplitter.FileMarker`. The option `withMarkers` defaults to `false` if not explicitly set.

Options

The **sftp** source has the following options:

allowUnknownKeys
    true to allow connecting to a host with an unknown or changed key **(boolean, default: `false`)**

autoCreateLocalDir
    if local directory must be auto created if it does not exist **(boolean, default: `true`)**

deleteRemoteFiles
    delete remote files after transfer **(boolean, default: `false`)**

fixedDelay
    fixed delay in SECONDS to poll the remote directory **(int, default: `1`)**

host
    the remote host to connect to **(String, default: `localhost`)**

initialDelay
    an initial delay when using a fixed delay trigger, expressed in TimeUnits (seconds by default) **(int, default: `0`)**

knownHostsExpression
    a SpEL expresssion location of known hosts file; required if 'allowUnknownKeys' is false; examples: systemProperties["user.home"]+"/.ssh/known_hosts", "/foo/bar/known_hosts" **(String, no default)**

localDir
    set the local directory the remote files are transferred to **(String, default: ``)**

maxMessages
: the maximum messages per poll; -1 for unlimited **(long, default: `-1`)**

mode
: specifies how the file is being read. By default the content of a file is provided as byte array **(FileReadingMode, default: `contents`, possible values: `ref,lines,contents`)**

passPhrase
: the passphrase to use **(String, default: ``)**

password
: the password for the provided user **(String, default: ``)**

pattern
: simple filename pattern to apply to the filter **(String, no default)**

port
: the remote port to connect to **(int, default: `22`)**

privateKey
: the private key location (a valid Spring Resource URL) **(String, default: ``)**

regexPattern
: filename regex pattern to apply to the filter **(String, no default)**

remoteDir
: the remote directory to transfer the files from **(String, no default)**

timeUnit
: the time unit for the fixed and initial delays **(String, default: `SECONDS`)**

tmpFileSuffix
: extension to use when downloading files **(String, default: `.tmp`)**

user
: the username to use **(String, no default)**

withMarkers
: if true emits start of file/end of file marker messages before/after the data. Only valid with FileReadingMode 'lines' **(Boolean, no default)**

5.6 TCP

The `tcp` source acts as a server and allows a remote party to connect to Spring Cloud Data Flow and submit data over a raw tcp socket.

TCP is a streaming protocol and some mechanism is needed to frame messages on the wire. A number of decoders are available, the default being 'CRLF' which is compatible with Telnet.

Messages produced by the TCP source module have a `byte[]` payload.

Options

bufferSize
: the size of the buffer (bytes) to use when decoding **(int, default: `2048`)**

decoder
    the decoder to use when receiving messages **(Encoding, default: `CRLF`, possible values: `CRLF,LF,NULL,STXETX,RAW,L1,L2,L4`)**

nio
    whether or not to use NIO **(boolean, default: `false`)**

port
    the port on which to listen **(int, default: `1234`)**

reverseLookup
    perform a reverse DNS lookup on the remote IP Address **(boolean, default: `false`)**

socketTimeout
    the timeout (ms) before closing the socket when no data is received **(int, default: `120000`)**

useDirectBuffers
    whether or not to use direct buffers **(boolean, default: `false`)**

Available Decoders

Text Data

CRLF (default)
    text terminated by carriage return (0x0d) followed by line feed (0x0a)

LF
    text terminated by line feed (0x0a)

NULL
    text terminated by a null byte (0x00)

STXETX
    text preceded by an STX (0x02) and terminated by an ETX (0x03)
Text and Binary Data

RAW
    no structure - the client indicates a complete message by closing the socket

L1
    data preceded by a one byte (unsigned) length field (supports up to 255 bytes)

L2
    data preceded by a two byte (unsigned) length field (up to 2^{16}-1 bytes)

L4
    data preceded by a four byte (signed) length field (up to 2^{31}-1 bytes)

5.7 Time (`time`)

The time source will simply emit a String with the current time every so often.

Options

The **time** source has the following options:

fixedDelay
  time delay between messages, expressed in TimeUnits (seconds by default) **(int, default: `1`)**

format
  how to render the current time, using SimpleDateFormat **(String, default: `yyyy-MM-dd HH:mm:ss`)**

initialDelay
  an initial delay when using a fixed delay trigger, expressed in TimeUnits (seconds by default) **(int, default: `0`)**

timeUnit
  the time unit for the fixed and initial delays **(String, default: `SECONDS`)**

5.8 Twitter Stream (`twitterstream`)

This source ingests data from Twitter's streaming API v1.1. It uses the sample and filter stream endpoints rather than the full "firehose" which needs special access. The endpoint used will depend on the parameters you supply in the stream definition (some are specific to the filter endpoint).

You need to supply all keys and secrets (both consumer and accessToken) to authenticate for this source, so it is easiest if you just add these as the following environment variables: CONSUMER_KEY, CONSUMER_SECRET, ACCESS_TOKEN and ACCESS_TOKEN_SECRET.

Stream creation is then straightforward:

```
dataflow:> stream create --name tweets --definition "twitterstream | log" --deploy
```

Options

The **twitterstream** source has the following options:

accessToken
  a valid OAuth access token **(String, no default)**

accessTokenSecret
  an OAuth secret corresponding to the access token **(String, no default)**

consumerKey
  a consumer key issued by twitter **(String, no default)**

consumerSecret
  consumer secret corresponding to the consumer key **(String, no default)**

language
  language code e.g. 'en' **(String, default: ``)**

> **Note**
>
> `twitterstream` emit JSON in the native Twitter format.

6. Processors

6.1 Filter (`filter`)

Use the filter module in a stream to determine whether a Message should be passed to the output channel.

The **filter** processor has the following options:

expression
    a SpEL expression used to transform messages **(String, default: `payload.toString()`)**

Filter with SpEL expression

The simplest way to use the filter processor is to pass a SpEL expression when creating the stream. The expression should evaluate the message and return true or false. For example:

```
dataflow:> stream create --name filtertest --definition "http --server.port=9000 | filter --
expression=payload=='good' | log" --deploy
```

This filter will only pass Messages to the log sink if the payload is the word "good". Try sending "good" to the HTTP endpoint and you should see it in the Spring Cloud Data Flow logs:

```
dataflow:> http post --target http://localhost:9000 --data "good"
```

Alternatively, if you send the word "bad" (or anything else), you shouldn't see the log entry.

6.2 Groovy Filter (`groovy-filter`)

A Processor module that retains or discards messages according to a predicate, expressed as a Groovy script.

Options

The **groovy-filter** processor has the following options:

script
    The script resource location **(String, default: ``)**

variables
    Variable bindings as a comma delimited string of name-value pairs, e.g. 'foo=bar,baz=car' **(String, default: ``)**

variablesLocation
    The location of a properties file containing custom script variable bindings **(String, default: ``)**

6.3 Http Client (`httpclient`)

A processor module that makes requests to an HTTP resource and emits the response body as a message payload. This processor can be combined, e.g., with a time source module to periodically poll results from a HTTP resource.

Options

The **httpclient** processor has the following options:

url

    The URL to issue an http request to, as a static value.

urlExpression

    A SpEL expression against incoming message to determine the URL to use.

httpMethod

    The kind of http method to use.

body

    The (static) body of the request to use.

bodyExpression

    A SpEL expression against incoming message to derive the request body to use.

headersExpression

    A SpEL expression used to derive the http headers map to use.

expectedResponseType

    The type used to interpret the response.

replyExpression

    A SpEL expression used to compute the final result, applied against the whole http response.

6.4 Noop (`noop`)

A Processor module that returns messages that is passed in for performance testing.

6.5 Groovy Transform (`groovy-transform`)

A Processor module that transforms messages using a Groovy script.

Options

The **groovy-transform** processor has the following options:

script

    The script resource location **(String, default: ``)**

variables

    Variable bindings as a comma delimited string of name-value pairs, e.g. 'foo=bar,baz=car' **(String, default: ``)**

variablesLocation

    The location of a properties file containing custom script variable bindings **(String, default: ``)**

6.6 Transform (`transform`)

Use the transform module in a stream to convert a Message's content or structure.

Options

The **transform** processor has the following options:

expression
    a SpEL expression used to transform messages **(String, default: `payload.toString()`)**

Transform with SpEL expression

The simplest way to use the transform processor is to pass a SpEL expression when creating the stream. The expression should return the modified message or payload. For example:

```
dataflow:> stream create --name transformtest --definition "http --server.port=9003 | transform --
expression=payload.toUpperCase() | log" --deploy
```

This transform will convert all message payloads to upper case. If sending the word "foo" to the HTTP endpoint and you should see "FOO" in the Spring Cloud Data Flow logs:

```
dataflow:> http post --target http://localhost:9003 --data "foo"
```

As part of the SpEL expression you can make use of the pre-registered JSON Path function. The syntax is #jsonPath(payload,'<json path expression>')

6.7 Splitter

The splitter module builds upon the concept of the same name in Spring Integration and allows the splitting of a single message into several distinct messages.

expression
    a SpEL expression which would typically evaluate to an array or collection **(String, default: `null`)**

delimiters
    A list of delimiters to tokenize a String payload ('expression' must be null) **(String, default: `null`)**

fileMarkers
    Split File payloads, when true, START and END marker messages will be emitted, when false no markers are emitted **(String, default: `null`)**

charset
    Split File payloads using this charset to convert bytes to String **(String, default: `null`)**

applySequence
    Add correlation and sequence information to the message headers **(String, default: `true`)**

When no `expression`, `fileMarkers`, or `charset` is provided, a `DefaultMessageSplitter` is configured with (optional) `delimiters`. When `fileMarkers` or `charset` is provided, a `FileSplitter` is configured (you must provide either a `fileMarkers` or `charset` to split files, which must be text-based - they are split into lines). Otherwise, an `ExpressionEvaluatingMessageSplitter` is configured.

When splitting `File` payloads, the `sequenceSize` header is zero because the size cannot be determined at the beginning.

Ambiguous properties are not allowed.

JSON Example

As part of the SpEL expression you can make use of the pre-registered JSON Path function. The syntax is #jsonPath(payload, '<json path expression>').

For example, consider the following JSON:

```json
{ "store": {
    "book": [
        {
            "category": "reference",
            "author": "Nigel Rees",
            "title": "Sayings of the Century",
            "price": 8.95
        },
        {
            "category": "fiction",
            "author": "Evelyn Waugh",
            "title": "Sword of Honour",
            "price": 12.99
        },
        {
            "category": "fiction",
            "author": "Herman Melville",
            "title": "Moby Dick",
            "isbn": "0-553-21311-3",
            "price": 8.99
        },
        {
            "category": "fiction",
            "author": "J. R. R. Tolkien",
            "title": "The Lord of the Rings",
            "isbn": "0-395-19395-8",
            "price": 22.99
        }
    ],
    "bicycle": {
        "color": "red",
        "price": 19.95
    }
}}
```

and an expression `#jsonPath(payload, '$.store.book')`; the result will be 4 messages, each with a `Map` payload containing the properties of a single book.

7. Sinks

7.1 Cassandra (`cassandra`)

The Cassandra sink writes into a Cassandra table. Here is a simple example

```
dataflow:>stream create cassandrastream --definition "http --server.port=8888 --
spring.cloud.stream.bindings.output.contentType='application/json' | cassandra --ingestQuery='insert
 into book (id, isbn, title, author) values (uuid(), ?, ?, ?)' --spring.cassandra.keyspace=clouddata" --
deploy
```

Create a keyspace and a `book` table in Cassandra using:

```
CREATE KEYSPACE clouddata WITH REPLICATION = { 'class' : 'org.apache.cassandra.locator.SimpleStrategy',
 'replication_factor': '1' } AND DURABLE_WRITES = true;
USE clouddata;
CREATE TABLE book  (
    id          uuid PRIMARY KEY,
    isbn        text,
    author      text,
    title       text
);
```

You can then send data to this stream via

```
dataflow:>http post --contentType 'application/json' --data '{"isbn": "1599869772", "title": "The Art of
 War", "author": "Sun Tzu"}' --target http://localhost:8888/
> POST (application/json;charset=UTF-8) http://localhost:8888/ {"isbn": "1599869772", "title": "The Art
 of War", "author": "Sun Tzu"}
> 202 ACCEPTED
```

and see the table contents using the CQL

```
SELECT * FROM clouddata.book;
```

Options

The **cassandra** sink has the following options:

compressionType
    the compression to use for the transport **(CompressionType, default: `NONE`, possible values: `NONE,SNAPPY`)**

consistencyLevel
    the consistencyLevel option of WriteOptions **(ConsistencyLevel, no default, possible values: `ANY,ONE,TWO,THREE,QUOROM,LOCAL_QUOROM,EACH_QUOROM,ALL,LOCAL_ONE,SERIAL,LOCAL_SERIAL`)**

spring.cassandra.contactPoints
    the comma-delimited string of the hosts to connect to Cassandra **(String, default: `localhost`)**

entityBasePackages
    the base packages to scan for entities annotated with Table annotations **(String[], default: `[]`)**

ingestQuery
    the ingest Cassandra query **(String, no default)**

spring.cassandra.initScript
    the path to file with CQL scripts (delimited by ';') to initialize keyspace schema **(String, no default)**

spring.cassandra.keyspace
    the keyspace name to connect to **(String, default: `<stream name>`)**

metricsEnabled
    enable/disable metrics collection for the created cluster **(boolean, default: `true`)**

spring.cassandra.password
    the password for connection **(String, no default)**

spring.cassandra.port
    the port to use to connect to the Cassandra host **(int, default: `9042`)**

queryType
    the queryType for Cassandra Sink **(Type, default: `INSERT`, possible values: `INSERT,UPDATE,DELETE,STATEMENT`)**

retryPolicy
    the retryPolicy option of WriteOptions **(RetryPolicy, no default, possible values: `DEFAULT,DOWNGRADING_CONSISTENCY,FALLTHROUGH,LOGGING`)**

statementExpression
    the expression in Cassandra query DSL style **(String, no default)**

spring.cassandra.schemaAction
    schema action to perform **(SchemaAction, default: `NONE`, possible values: `CREATE,NONE,RECREATE,RECREATE_DROP_UNUSED`)**

ttl
    the time-to-live option of WriteOptions **(int, default: `0`)**

spring.cassandra.username
    the username for connection **(String, no default)**

7.2 Counter (`counter`)

A simple module that counts messages received, using Spring Boot metrics abstraction.

The **counter** sink has the following options:

name
    The name of the counter to increment. **(String, default: `counts`)**

nameExpression
    A SpEL expression (against the incoming Message) to derive the name of the counter to increment. **(String, default: `` ` ` `)**

store
    The name of a store used to store the counter. **(String, default: `memory`, possible values: `memory, redis`)**

7.3 Field Value Counter (`field-value-counter`)

A field value counter is a Metric used for counting occurrences of unique values for a named field in a message payload. Spring Cloud Data Flow supports the following payload types out of the box:

• POJO (Java bean)

- Tuple

- JSON String

For example suppose a message source produces a payload with a field named *user*:

```
class Foo {
    String user;
    public Foo(String user) {
        this.user = user;
    }
}
```

If the stream source produces messages with the following objects:

```
    new Foo("fred")
    new Foo("sue")
    new Foo("dave")
    new Foo("sue")
```

The field value counter on the field *user* will contain:

```
fred:1, sue:2, dave:1
```

Multi-value fields are also supported. For example, if a field contains a list, each value will be counted once:

```
users:["dave","fred","sue"]
users:["sue","jon"]
```

The field value counter on the field *users* will contain:

```
dave:1, fred:1, sue:2, jon:1
```

Options

The **field-value-counter** sink has the following options:

fieldName
> the name of the field for which values are counted **(String, no default)**

name
> the name of the metric to contribute to (will be created if necessary) **(String, default: `<stream name>`)**

nameExpression
> a SpEL expression to compute the name of the metric to contribute to **(String, no default)**

7.4 File (`file`)

This module writes each message it receives to a file.

Options

The **file** sink has the following options:

binary
> if false, will append a newline character at the end of each line **(boolean, default: `false`)**

charset
    the charset to use when writing a String payload **(String, default: `UTF-8`)**

dir
    the directory in which files will be created **(String, default: ``)**

dirExpression
    spring expression used to define directory name **(String, no default)**

mode
    what to do if the file already exists **(Mode, default: `APPEND`, possible values: `APPEND,REPLACE,FAIL,IGNORE`)**

name
    filename pattern to use **(String, default: `<stream name>`)**

nameExpression
    spring expression used to define filename **(String, no default)**

suffix
    filename extension to use **(String, no default)**

7.5 FTP Sink (`ftp`)

FTP sink is a simple option to push files to an FTP server from incoming messages.

It uses an `ftp-outbound-adapter`, therefore incoming messages could be either a `java.io.File` object, a `String` (content of the file) or an array of `bytes` (file content as well).

To use this sink, you need a username and a password to login.

> **Note**
>
> By default Spring Integration will use `o.s.i.file.DefaultFileNameGenerator` if none is specified. `DefaultFileNameGenerator` will determine the file name based on the value of the `file_name` header (if it exists) in the `MessageHeaders`, or if the payload of the `Message` is already a `java.io.File`, then it will use the original name of that file.

7.6 Gemfire (`gemfire`)

A sink module that allows one to write message payloads to a Gemfire server.

Options

The **gemfire** sink has the following options:

hostAddresses
    a comma separated list of [host]:[port] specifying either locator or server addresses for the client connection pool **(String, `localhost:10334`)**

keyExpression
    a SpEL expression which is evaluated to create a cache key **(String, default: `the value is currently the message payload'`)**

port

> port of the cache server or locator (if useLocator=true). May be a comma delimited list **(String, no default)**

regionName

> name of the region to use when storing data **(String, default: ${spring.application.name})**

connectType

> 'server' or 'locator' **(String, default: locator)**

7.7 Hadoop (HDFS) (`hdfs`)

If you do not have Hadoop installed, you can install Hadoop as described in our separate guide.

Once Hadoop is up and running, you can then use the hdfs sink when creating a stream

```
dataflow:> stream create --name myhdfsstream1 --definition "time | hdfs" --deploy
```

In the above example, we've scheduled time source to automatically send ticks to hdfs once in every second. If you wait a little while for data to accumuluate you can then list can then list the files in the hadoop filesystem using the shell's built in hadoop fs commands. Before making any access to HDFS in the shell you first need to configure the shell to point to your name node. This is done using the hadoop config command.

```
dataflow:>hadoop config fs --namenode hdfs://localhost:8020
```

In this example the hdfs protocol is used but you may also use the webhdfs protocol. Listing the contents in the output directory (named by default after the stream name) is done by issuing the following command.

```
dataflow:>hadoop fs ls /xd/myhdfsstream1
Found 1 items
-rw-r--r--   3 jvalkealahti supergroup          0 2013-12-18 18:10 /xd/myhdfsstream1/
myhdfsstream1-0.txt.tmp
```

While the file is being written to it will have the tmp suffix. When the data written exceeds the rollover size (default 1GB) it will be renamed to remove the tmp suffix. There are several options to control the in use file file naming options. These are --inUsePrefix and --inUseSuffix set the file name prefix and suffix respectfully.

When you destroy a stream

```
dataflow:>stream destroy --name myhdfsstream1
```

and list the stream directory again, in use file suffix doesn't exist anymore.

```
dataflow:>hadoop fs ls /xd/myhdfsstream1
Found 1 items
-rw-r--r--   3 jvalkealahti supergroup        380 2013-12-18 18:10 /xd/myhdfsstream1/myhdfsstream1-0.txt
```

To list the list the contents of a file directly from a shell execute the hadoop cat command.

```
dataflow:> hadoop fs cat /xd/myhdfsstream1/myhdfsstream1-0.txt
2013-12-18 18:10:07
2013-12-18 18:10:08
2013-12-18 18:10:09
...
```

In the above examples we didn't yet go through why the file was written in a specific directory and why it was named in this specific way. Default location of a file is defined as `/xd/<stream name>/<stream name>-<rolling part>.txt`. These can be changed using options `--directory` and `--fileName` respectively. Example is shown below.

```
dataflow:>stream create --name myhdfsstream2 --definition "time | hdfs --directory=/xd/tmp --
fileName=data" --deploy
dataflow:>stream destroy --name myhdfsstream2
dataflow:>hadoop fs ls /xd/tmp
Found 1 items
-rw-r--r--   3 jvalkealahti supergroup        120 2013-12-18 18:31 /xd/tmp/data-0.txt
```

It is also possible to control the size of a files written into HDFS. The `--rollover` option can be used to control when file currently being written is rolled over and a new file opened by providing the rollover size in bytes, kilobytes, megatypes, gigabytes, and terabytes.

```
dataflow:>stream create --name myhdfsstream3 --definition "time | hdfs --rollover=100" --deploy
dataflow:>stream destroy --name myhdfsstream3
dataflow:>hadoop fs ls /xd/myhdfsstream3
Found 3 items
-rw-r--r--   3 jvalkealahti supergroup        100 2013-12-18 18:41 /xd/myhdfsstream3/myhdfsstream3-0.txt
-rw-r--r--   3 jvalkealahti supergroup        100 2013-12-18 18:41 /xd/myhdfsstream3/myhdfsstream3-1.txt
-rw-r--r--   3 jvalkealahti supergroup        100 2013-12-18 18:41 /xd/myhdfsstream3/myhdfsstream3-2.txt
```

Shortcuts to specify sizes other than bytes are written as `--rollover=64M`, `--rollover=512G` or `--rollover=1T`.

The stream can also be compressed during the write operation. Example of this is shown below.

```
dataflow:>stream create --name myhdfsstream4 --definition "time | hdfs --codec=gzip" --deploy
dataflow:>stream destroy --name myhdfsstream4
dataflow:>hadoop fs ls /xd/myhdfsstream4
Found 1 items
-rw-r--r--   3 jvalkealahti supergroup         80 2013-12-18 18:48 /xd/myhdfsstream4/
myhdfsstream4-0.txt.gzip
```

From a native os shell we can use hadoop's fs commands and pipe data into gunzip.

```
# bin/hadoop fs -cat /xd/myhdfsstream4/myhdfsstream4-0.txt.gzip | gunzip
2013-12-18 18:48:10
2013-12-18 18:48:11
...
```

Often a stream of data may not have a high enough rate to roll over files frequently, leaving the file in an opened state. This prevents users from reading a consistent set of data when running mapreduce jobs. While one can alleviate this problem by using a small rollover value, a better way is to use the `idleTimeout` option that will automatically close the file if there was no writes during the specified period of time. This feature is also useful in cases where burst of data is written into a stream and you'd like that data to become visible in HDFS.

> **Note**
>
> The `idleTimeout` value should not exceed the timeout values set on the Hadoop cluster. These are typically configured using the `dfs.socket.timeout` and/or `dfs.datanode.socket.write.timeout` properties in the `hdfs-site.xml` configuration file.

```
dataflow:> stream create --name myhdfsstream5 --definition "http --server.port=8000 | hdfs --rollover=20
 --idleTimeout=10000" --deploy
```

In the above example we changed a source to `http` order to control what we write into a `hdfs` sink. We defined a small rollover size and a timeout of 10 seconds. Now we can simply post data into this stream via source end point using a below command.

```
dataflow:> http post --target http://localhost:8000 --data "hello"
```

If we repeat the command very quickly and then wait for the timeout we should be able to see that some files are closed before rollover size was met and some were simply rolled because of a rollover size.

```
dataflow:>hadoop fs ls /xd/myhdfsstream5
Found 4 items
-rw-r--r--   3 jvalkealahti supergroup          12 2013-12-18 19:02 /xd/myhdfsstream5/myhdfsstream5-0.txt
-rw-r--r--   3 jvalkealahti supergroup          24 2013-12-18 19:03 /xd/myhdfsstream5/myhdfsstream5-1.txt
-rw-r--r--   3 jvalkealahti supergroup          24 2013-12-18 19:03 /xd/myhdfsstream5/myhdfsstream5-2.txt
-rw-r--r--   3 jvalkealahti supergroup          18 2013-12-18 19:03 /xd/myhdfsstream5/myhdfsstream5-3.txt
```

Files can be automatically partitioned using a `partitionPath` expression. If we create a stream with `idleTimeout` and `partitionPath` with simple format `yyyy/MM/dd/HH/mm` we should see writes ending into its own files within every minute boundary.

```
dataflow:>stream create --name myhdfsstream6 --definition "time|hdfs --idleTimeout=10000 --
partitionPath=dateFormat('yyyy/MM/dd/HH/mm')" --deploy
```

Let a stream run for a short period of time and list files.

```
dataflow:>hadoop fs ls --recursive true --dir /xd/myhdfsstream6
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 09:42 /xd/myhdfsstream6/2014
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 09:42 /xd/myhdfsstream6/2014/05
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 09:42 /xd/myhdfsstream6/2014/05/28
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 09:45 /xd/myhdfsstream6/2014/05/28/09
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 09:43 /xd/myhdfsstream6/2014/05/28/09/42
-rw-r--r--   3 jvalkealahti supergroup         140 2014-05-28 09:43 /xd/myhdfsstream6/2014/05/28/09/42/
myhdfsstream6-0.txt
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 09:44 /xd/myhdfsstream6/2014/05/28/09/43
-rw-r--r--   3 jvalkealahti supergroup        1200 2014-05-28 09:44 /xd/myhdfsstream6/2014/05/28/09/43/
myhdfsstream6-0.txt
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 09:45 /xd/myhdfsstream6/2014/05/28/09/44
-rw-r--r--   3 jvalkealahti supergroup        1200 2014-05-28 09:45 /xd/myhdfsstream6/2014/05/28/09/44/
myhdfsstream6-0.txt
```

Partitioning can also be based on defined lists. In a below example we simulate feeding data by using a `time` and a `transform` elements. Data passed to `hdfs` sink has a content `APP0:foobar`, `APP1:foobar`, `APP2:foobar` or `APP3:foobar`.

```
dataflow:>stream create --name myhdfsstream7 --definition "time | transform --expression=
\"'APP'+T(Math).round(T(Math).random()*3)+':foobar'\" | hdfs --idleTimeout=10000 --
partitionPath=path(dateFormat('yyyy/MM/dd/HH'),list(payload.split(':')[0],{{'0TO1','APP0','APP1'},
{'2TO3','APP2','APP3'}}))" --deploy
```

Let the stream run few seconds, destroy it and check what got written in those partitioned files.

```
dataflow:>stream destroy --name myhdfsstream7
Destroyed stream 'myhdfsstream7'
dataflow:>hadoop fs ls --recursive true --dir /xd
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 19:24 /xd/myhdfsstream7
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 19:24 /xd/myhdfsstream7/2014
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 19:24 /xd/myhdfsstream7/2014/05
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 19:24 /xd/myhdfsstream7/2014/05/28
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 19:24 /xd/myhdfsstream7/2014/05/28/19
drwxr-xr-x   - jvalkealahti supergroup           0 2014-05-28 19:24 /xd/
myhdfsstream7/2014/05/28/19/0TO1_list
-rw-r--r--   3 jvalkealahti supergroup         108 2014-05-28 19:24 /xd/
myhdfsstream7/2014/05/28/19/0TO1_list/myhdfsstream7-0.txt
```

```
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:24 /xd/
myhdfsstream7/2014/05/28/19/2TO3_list
-rw-r--r--   3 jvalkealahti supergroup        180 2014-05-28 19:24 /xd/
myhdfsstream7/2014/05/28/19/2TO3_list/myhdfsstream7-0.txt
dataflow:>hadoop fs cat /xd/myhdfsstream7/2014/05/28/19/0TO1_list/myhdfsstream7-0.txt
APP1:foobar
APP1:foobar
APP0:foobar
APP0:foobar
APP1:foobar
```

Partitioning can also be based on defined ranges. In a below example we simulate feeding data by using a `time` and a `transform` elements. Data passed to `hdfs` sink has a content ranging from `APP0` to `APP15`. We simple parse the number part and use it to do a partition with ranges `{3,5,10}`.

```
dataflow:>stream create --name myhdfsstream8 --definition "time | transform --
expression=\"'APP'+T(Math).round(T(Math).random()*15)\" | hdfs --idleTimeout=10000 --
partitionPath=path(dateFormat('yyyy/MM/dd/HH'),range(T(Integer).parseInt(payload.substring(3)),
{3,5,10}))" --deploy
```

Let the stream run few seconds, destroy it and check what got written in those partitioned files.

```
dataflow:>stream destroy --name myhdfsstream8
Destroyed stream 'myhdfsstream8'
dataflow:>hadoop fs ls --recursive true --dir /xd
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:34 /xd/myhdfsstream8
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:34 /xd/myhdfsstream8/2014
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:34 /xd/myhdfsstream8/2014/05
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:34 /xd/myhdfsstream8/2014/05/28
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:34 /xd/myhdfsstream8/2014/05/28/19
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:34 /xd/
myhdfsstream8/2014/05/28/19/10_range
-rw-r--r--   3 jvalkealahti supergroup         16 2014-05-28 19:34 /xd/
myhdfsstream8/2014/05/28/19/10_range/myhdfsstream8-0.txt
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:34 /xd/
myhdfsstream8/2014/05/28/19/3_range
-rw-r--r--   3 jvalkealahti supergroup         35 2014-05-28 19:34 /xd/
myhdfsstream8/2014/05/28/19/3_range/myhdfsstream8-0.txt
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:34 /xd/
myhdfsstream8/2014/05/28/19/5_range
-rw-r--r--   3 jvalkealahti supergroup          5 2014-05-28 19:34 /xd/
myhdfsstream8/2014/05/28/19/5_range/myhdfsstream8-0.txt
dataflow:>hadoop fs cat /xd/myhdfsstream8/2014/05/28/19/3_range/myhdfsstream8-0.txt
APP3
APP3
APP1
APP0
APP1
dataflow:>hadoop fs cat /xd/myhdfsstream8/2014/05/28/19/5_range/myhdfsstream8-0.txt
APP4
dataflow:>hadoop fs cat /xd/myhdfsstream8/2014/05/28/19/10_range/myhdfsstream8-0.txt
APP6
APP15
APP7
```

Partition using a `dateFormat` can be based on content itself. This is a good use case if old log files needs to be processed where partitioning should happen based on timestamp of a log entry. We create a fake log data with a simple date string ranging from `1970-01-10` to `1970-01-13`.

```
dataflow:>stream create --name myhdfsstream9 --definition "time | transform --expression=
\"'1970-01-'+1+T(Math).round(T(Math).random()*3)\" | hdfs --idleTimeout=10000 --
partitionPath=path(dateFormat('yyyy/MM/dd/HH',payload,'yyyy-MM-DD'))" --deploy
```

Let the stream run few seconds, destroy it and check what got written in those partitioned files. If you see the partition paths, those are based on year 1970, not present year.

```
dataflow:>stream destroy --name myhdfsstream9
Destroyed stream 'myhdfsstream9'
dataflow:>hadoop fs ls --recursive true --dir /xd
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:56 /xd/myhdfsstream9
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:56 /xd/myhdfsstream9/1970
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:56 /xd/myhdfsstream9/1970/01
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:56 /xd/myhdfsstream9/1970/01/10
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:57 /xd/myhdfsstream9/1970/01/10/00
-rw-r--r--   3 jvalkealahti supergroup         44 2014-05-28 19:57 /xd/myhdfsstream9/1970/01/10/00/
myhdfsstream9-0.txt
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:56 /xd/myhdfsstream9/1970/01/11
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:57 /xd/myhdfsstream9/1970/01/11/00
-rw-r--r--   3 jvalkealahti supergroup         99 2014-05-28 19:57 /xd/myhdfsstream9/1970/01/11/00/
myhdfsstream9-0.txt
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:56 /xd/myhdfsstream9/1970/01/12
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:57 /xd/myhdfsstream9/1970/01/12/00
-rw-r--r--   3 jvalkealahti supergroup         44 2014-05-28 19:57 /xd/myhdfsstream9/1970/01/12/00/
myhdfsstream9-0.txt
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:56 /xd/myhdfsstream9/1970/01/13
drwxr-xr-x   - jvalkealahti supergroup          0 2014-05-28 19:57 /xd/myhdfsstream9/1970/01/13/00
-rw-r--r--   3 jvalkealahti supergroup         55 2014-05-28 19:57 /xd/myhdfsstream9/1970/01/13/00/
myhdfsstream9-0.txt
dataflow:>hadoop fs cat /xd/myhdfsstream9/1970/01/10/00/myhdfsstream9-0.txt
1970-01-10
1970-01-10
1970-01-10
1970-01-10
```

Options

The **hdfs** sink has the following options:

closeTimeout
    timeout in ms, regardless of activity, after which file will be automatically closed **(long, default: `0`)**

codec
    compression codec alias name (gzip, snappy, bzip2, lzo, or slzo) **(String, default: ``)**

directory
    where to output the files in the Hadoop FileSystem **(String, default: `/tmp/hdfs-sink`)**

fileExtension
    the base filename extension to use for the created files **(String, default: `txt`)**

fileName
    the base filename to use for the created files **(String, default: `<stream name>`)**

fileOpenAttempts
    maximum number of file open attempts to find a path **(int, default: `10`)**

fileUuid
    whether file name should contain uuid **(boolean, default: `false`)**

fsUri
    the URI to use to access the Hadoop FileSystem **(String, default: `${spring.hadoop.fsUri}`)**

idleTimeout
    inactivity timeout in ms after which file will be automatically closed **(long, default: `0`)**

inUsePrefix
    prefix for files currently being written **(String, default: ``)**

inUseSuffix
    suffix for files currently being written **(String, default: `.tmp`)**

overwrite
    whether writer is allowed to overwrite files in Hadoop FileSystem **(boolean, default: `false`)**

partitionPath
    a SpEL expression defining the partition path **(String, default: ` `` `)**

rollover
    threshold in bytes when file will be automatically rolled over **(String, default: `1G`)**

> **Note**
>
> In the context of the `fileOpenAttempts` option, attempt is either one rollover request or failed stream open request for a path (if another writer came up with a same path and already opened it).

Partition Path Expression

SpEL expression is evaluated against a Spring Messaging `Message` passed internally into a HDFS writer. This allows expression to use `headers` and `payload` from that message. While you could do a custom processing within a stream and add custom headers, `timestamp` is always going to be there. Data to be written is then available in a `payload`.

Accessing Properties

Using a `payload` simply returns whatever is currently being written. Access to headers is via `headers` property. Any other property is automatically resolved from headers if found. For example `headers.timestamp` is equivalent to `timestamp`.

Custom Methods

Addition to a normal SpEL functionality, few custom methods has been added to make it easier to build partition paths. These custom methods can be used to work with a normal partition concepts like `date formatting`, `lists`, `ranges` and `hashes`.

path

```
path(String... paths)
```

Concatenates paths together with a delimiter `/`. This method can be used to make the expression less verbose than using a native SpEL functionality to combine path parts together. To create a path `part1/part2`, expression `'part1' + '/' + 'part2'` is equivalent to `path('part1','part2')`.
Parameters

paths
    Any number of path parts

Return Value.   Concatenated value of paths delimited with `/`.

dateFormat

```
dateFormat(String pattern)
dateFormat(String pattern, Long epoch)
dateFormat(String pattern, Date date)
dateFormat(String pattern, String datestring)
dateFormat(String pattern, String datestring, String dateformat)
```

Creates a path using date formatting. Internally this method delegates into `SimpleDateFormat` and needs a `Date` and a `pattern`. On default if no parameter used for conversion is given, `timestamp` is expected. Effectively `dateFormat('yyyy')` equals to `dateFormat('yyyy', timestamp)` or `dateFormat('yyyy', headers.timestamp)`.

Method signature with three parameters can be used to create a custom `Date` object which is then passed to `SimpleDateFormat` conversion using a `dateformat` pattern. This is useful in use cases where partition should be based on a date or time string found from a payload content itself. Default `dateformat` pattern if omitted is `yyyy-MM-dd`.
Parameters

pattern
   Pattern compatible with `SimpleDateFormat` to produce a final output.

epoch
   Timestamp as `Long` which is converted into a `Date`.

date
   A `Date` to be formatted.

dateformat
   Secondary pattern to convert `datestring` into a `Date`.

datestring
   `Date` as a `String`

Return Value. A path part representation which can be a simple file or directory name or a directory structure.

list

```
list(Object source, List<List<Object>> lists)
```

Creates a partition path part by matching a `source` against a lists denoted by `lists`.

Lets assume that data is being written and it's possible to extrace an `appid` either from headers or payload. We can automatically do a list based partition by using a partition method `list(headers.appid,{{'1TO3','APP1','APP2','APP3'}, {'4TO6','APP4','APP5','APP6'}})`. This method would create three partitions, `1TO3_list`, `4TO6_list` and `list`. Latter is used if no match is found from partition lists passed to `lists`.
Parameters

source
   An `Object` to be matched against `lists`.

lists
   A definition of list of lists.

Return Value. A path part prefixed with a matched key i.e. `XXX_list` or `list` if no match.

range

```
range(Object source, List<Object> list)
```

Creates a partition path part by matching a `source` against a list denoted by `list` using a simple binary search.

The partition method takes a `source` as first argument and `list` as a second argument. Behind the scenes this is using jvm's `binarySearch` which works on an `Object` level so we can pass in anything. Remember that meaningful range match only works if passed in `Object` and types in list are of same type like `Integer`. Range is defined by a binarySearch itself so mostly it is to match against an upper bound except the last range in a list. Having a list of {1000,3000,5000} means that everything above 3000 will be matched with 5000. If that is an issue then simply adding `Integer.MAX_VALUE` as last range would overflow everything above 5000 into a new partition. Created partitions would then be `1000_range`, `3000_range` and `5000_range`.

Parameters

source

An `Object` to be matched against `list`.

list

A definition of list.

Return Value. A path part prefixed with a matched key i.e. `XXX_range`.

hash

```
hash(Object source, int bucketcount)
```

Creates a partition path part by calculating hashkey using `source`s hashCode and `bucketcount`. Using a partition method `hash(timestamp,2)` would then create partitions named `0_hash`, `1_hash` and `2_hash`. Number suffixed with `_hash` is simply calculated using `Object.hashCode() % bucketcount`.

Parameters

source

An `Object` which `hashCode` will be used.

bucketcount

A number of buckets

Return Value. A path part prefixed with a hash key i.e. `XXX_hash`.

7.8 JDBC (`jdbc`)

A module that writes its incoming payload to an RDBMS using JDBC.

Options

The **jdbc** sink has the following options:

expression

a SpEL expression used to transform messages **(String, default: ``)**

tableName

String **(String, default: `<stream name`)**

columns

the names of the columns that shall receive data, as a set of column[:SpEL] mappings, also used at initialization time to issue the DDL **(String, default: `payload`)**

initialize
　　String **(Boolean, default: `false`)**

batchSize
　　String **(long, default: `10000`)**

The module also uses Spring Boot's [DataSource support](#) for configuring the database connection, so properties like `spring.datasource.url` *etc.* apply.

7.9 Log (`log`)

Probably the simplest option for a sink is just to log the data. The `log` sink uses the application logger to output the data for inspection. The log level is set to `WARN` and the logger name is created from the stream name. To create a stream using a `log` sink you would use a command like

```
dataflow:> stream create --name mylogstream --definition "http --server.port=8000 | log" --deploy
```

You can then try adding some data. We've used the `http` source on port 8000 here, so run the following command to send a message

```
dataflow:> http post --target http://localhost:8000 --data "hello"
```

and you should see the following output in the Spring Cloud Data Flow console.

```
13/06/07 16:12:18 INFO Received: hello
```

7.10 Redis (`redis`)

Redis sink can be used to ingest data into redis store. You can choose `queue`, `topic` or `key` with selcted collection type to point to a specific data store.

For example,

```
dataflow:>stream create store-into-redis --definition "http | redis --queue=myList" --deploy
dataflow:>Created and deployed new stream 'store-into-redis'
```

Options

The **redis** sink has the following options:

topicExpression
　　a SpEL expression to use for topic **(String, no default)**

queueExpression
　　a SpEL expression to use for queue **(String, no default)**

keyExpression
　　a SpEL expression to use for keyExpression **(String, no default)**

key
　　name for the key **(String, no default)**

queue
　　name for the queue **(String, no default)**

topic
   name for the topic **(String, no default)**

7.11 TCP Sink

The TCP Sink provides for outbound messaging over TCP; messages sent to the sink can have `String` or `byte[]` payloads.

TCP is a streaming protocol and some mechanism is needed to frame messages on the wire. A number of encoders are available, the default being 'CRLF'.

Options

The **tcp** sink has the following options:

charset
   the charset used when converting from String to bytes **(String, default: `UTF-8`)**

close
   whether to close the socket after each message **(boolean, default: `false`)**

encoder
   the encoder to use when sending messages **(Encoding, default: `CRLF`, possible values: `CRLF,LF,NULL,STXETX,RAW,L1,L2,L4`)**

host
   the remote host to connect to **(String, default: `localhost`)**

nio
   whether or not to use NIO **(boolean, default: `false`)**

port
   the port on the remote host to connect to **(int, default: `1234`)**

reverseLookup
   perform a reverse DNS lookup on the remote IP Address **(boolean, default: `false`)**

socketTimeout
   the timeout (ms) before closing the socket when no data is received **(int, default: `120000`)**

useDirectBuffers
   whether or not to use direct buffers **(boolean, default: `false`)**

Available Encoders
Text Data

CRLF (default)
   text terminated by carriage return (0x0d) followed by line feed (0x0a)

LF
   text terminated by line feed (0x0a)

NULL
   text terminated by a null byte (0x00)

STXETX

text preceded by an STX (0x02) and terminated by an ETX (0x03)

Text and Binary Data

RAW

no structure - the client indicates a complete message by closing the socket

L1

data preceded by a one byte (unsigned) length field (supports up to 255 bytes)

L2

data preceded by a two byte (unsigned) length field (up to 2^{16}-1 bytes)

L4

data preceded by a four byte (signed) length field (up to 2^{31}-1 bytes)

Part IV. Appendices

Appendix A. Building

To build the source you will need to install JDK 1.7.

The build uses the Maven wrapper so you don't have to install a specific version of Maven. To enable the tests for Redis you should run the server before bulding. See below for more information on how run Redis.

The main build command is

```
$ ./mvnw clean install
```

You can also add '-DskipTests' if you like, to avoid running the tests.

> **Note**
>
> You can also install Maven (>=3.3.3) yourself and run the `mvn` command in place of `./mvnw` in the examples below. If you do that you also might need to add `-P spring` if your local Maven settings do not contain repository declarations for spring pre-release artifacts.

> **Note**
>
> Be aware that you might need to increase the amount of memory available to Maven by setting a `MAVEN_OPTS` environment variable with a value like `-Xmx512m -XX:MaxPermSize=128m`. We try to cover this in the `.mvn` configuration, so if you find you have to do it to make a build succeed, please raise a ticket to get the settings added to source control.

The projects that require middleware generally include a `docker-compose.yml`, so consider using Docker Compose to run the middeware servers in Docker containers. See the README in the scripts demo repository for specific instructions about the common cases of mongo, rabbit and redis.

A.1 Documentation

There is a "full" profile that will generate documentation. You can build just the documentation by executing

```
$ ./mvnw clean package -DskipTests -P full -pl {project-doc-module} -am
```

A.2 Working with the code

If you don't have an IDE preference we would recommend that you use Spring Tools Suite or Eclipse when working with the code. We use the m2eclipe eclipse plugin for maven support. Other IDEs and tools should also work without issue.

Importing into eclipse with m2eclipse

We recommend the m2eclipe eclipse plugin when working with eclipse. If you don't already have m2eclipse installed it is available from the "eclipse marketplace".

Unfortunately m2e does not yet support Maven 3.3, so once the projects are imported into Eclipse you will also need to tell m2eclipse to use the `.settings.xml` file for the projects. If you do not do this

you may see many different errors related to the POMs in the projects. Open your Eclipse preferences, expand the Maven preferences, and select User Settings. In the User Settings field click Browse and navigate to the Spring Cloud project you imported selecting the `.settings.xml` file in that project. Click Apply and then OK to save the preference changes.

> **Note**
>
> Alternatively you can copy the repository settings from <ins>`.settings.xml`</ins> into your own `~/.m2/settings.xml`.

Importing into eclipse without m2eclipse

If you prefer not to use m2eclipse you can generate eclipse project metadata using the following command:

```
$ ./mvnw eclipse:eclipse
```

The generated eclipse projects can be imported by selecting `import existing projects` from the `file` menu.

Appendix B. Contributing

Spring Cloud is released under the non-restrictive Apache 2.0 license, and follows a very standard Github development process, using Github tracker for issues and merging pull requests into master. If you want to contribute even something trivial please do not hesitate, but follow the guidelines below.

B.1 Sign the Contributor License Agreement

Before we accept a non-trivial patch or pull request we will need you to sign the contributor's agreement. Signing the contributor's agreement does not grant anyone commit rights to the main repository, but it does mean that we can accept your contributions, and you will get an author credit if we do. Active contributors might be asked to join the core team, and given the ability to merge pull requests.

B.2 Code Conventions and Housekeeping

None of these is essential for a pull request, but they will all help. They can also be added after the original pull request but before a merge.

- Use the Spring Framework code format conventions. If you use Eclipse you can import formatter settings using the `eclipse-code-formatter.xml` file from the Spring Cloud Build project. If using IntelliJ, you can use the Eclipse Code Formatter Plugin to import the same file.

- Make sure all new `.java` files to have a simple Javadoc class comment with at least an `@author` tag identifying you, and preferably at least a paragraph on what the class is for.

- Add the ASF license header comment to all new `.java` files (copy from existing files in the project)

- Add yourself as an `@author` to the .java files that you modify substantially (more than cosmetic changes).

- Add some Javadocs and, if you change the namespace, some XSD doc elements.

- A few unit tests would help a lot as well — someone has to do it.

- If no-one else is using your branch, please rebase it against the current master (or other target branch in the main project).

- When writing a commit message please follow these conventions, if you are fixing an existing issue please add `Fixes gh-XXXX` at the end of the commit message (where XXXX is the issue number).