

Cloudera Runtime 7.2.14

Cloudera Search ETL Using Morphlines

Date published: 2019-11-19

Date modified:

The Cloudera logo, featuring the word "CLOUDERA" in a bold, orange, sans-serif font. The letter "E" is stylized with a horizontal bar through its middle.

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

Legal Notice

© Cloudera Inc. 2025. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Using Morphlines to index Avro.....	4
Using Morphlines with Syslog.....	7

Using Morphlines to index Avro

This example illustrates using a morphline to index an Avro file with a schema.

1. View the content of the Avro file to understand the data:

```
$ wget http://archive.apache.org/dist/avro/avro-1.7.4/java/avro-tools-1.7.4.jar
$ java -jar avro-tools-1.7.4.jar tojson \
/usr/share/doc/search*/examples/test-documents/sample-statuses-20120906-141433.avro
```

2. Inspect the schema of the Avro file:

```
$ java -jar avro-tools-1.7.4.jar getschema /usr/share/doc/search*/examples/test-documents/sample-statuses-20120906-141433.avro

{
  "type" : "record",
  "name" : "Doc",
  "doc" : "adoc",
  "fields" : [ {
    "name" : "id",
    "type" : "string"
  }, {
    "name" : "user_statuses_count",
    "type" : [ "int", "null" ]
  }, {
    "name" : "user_screen_name",
    "type" : [ "string", "null" ]
  }, {
    "name" : "created_at",
    "type" : [ "string", "null" ]
  }, {
    "name" : "text",
    "type" : [ "string", "null" ]
  }
  ...
]
}
```

3. Extract the id, user_screen_name, created_at, and text fields from the Avro records, and then store and index them in Solr, using the following Solr schema definition in schema.xml:

```
<fields>
  <field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" />
  <field name="username" type="text_en" indexed="true" stored="true" />
  <field name="created_at" type="tdate" indexed="true" stored="true" />
  <field name="text" type="text_en" indexed="true" stored="true" />

  <field name="_version_" type="long" indexed="true" stored="true"/>
  <dynamicField name="ignored_*" type="ignored"/>
</fields>
```

The Solr output schema omits some Avro input fields, such as user_statuses_count. If your data includes Avro input fields that are not included in the Solr output schema, you may want to make changes to data as it is ingested. For example, suppose you need to rename the input field user_screen_name to the output field username.

Also suppose that the time format for the `created_at` field is `yyyy-MM-dd'T'HH:mm:ss'Z'`. Finally, suppose any unknown fields present are to be removed. Recall that Solr throws an exception on any attempt to load a document that contains a field that is not specified in `schema.xml`.

4. These transformation rules that make it possible to modify data so it fits your particular schema can be expressed with morphline commands called `readAvroContainer`, `extractAvroPaths`, `convertTimestamp`, `sanitizeUnknownSolrFields` and `loadSolr`, by editing a `morphline.conf` file.

```
# Specify server locations in a SOLR_LOCATOR variable; used later in
# variable substitutions:
SOLR_LOCATOR : {
  # Name of solr collection
  collection : collection1

  # ZooKeeper ensemble
  zkHost : "127.0.0.1:2181/solr"
}

# Specify an array of one or more morphlines, each of which defines an ETL
# transformation chain. A morphline consists of one or more potentially
# nested commands. A morphline is a way to consume records such as Flume e
# vents,
# HDFS files or blocks, turn them into a stream of records, and pipe the
# stream
# of records through a set of easily configurable transformations on its
# way to
# Solr.
morphlines : [
  {
    # Name used to identify a morphline. For example, used if there are mu
    # ltiple
    # morphlines in a morphline config file.
    id : morphline1
    # Import all morphline commands in these java packages and their su
    # bpackages.
    # Other commands that may be present on the classpath are not visible
    # to this
    # morphline.
    importCommands : ["org.kitesdk.**", "org.apache.solr.**"]
    commands : [
      {
        # Parse Avro container file and emit a record for each Avro object
        readAvroContainer {
          # Optionally, require the input to match one of these MIME ty
          # pes:
          # supportedMimeTypes : [avro/binary]
          # Optionally, use a custom Avro schema in JSON format inline:
          # readerSchemaString : ""<json can go here>""

          # Optionally, use a custom Avro schema file in JSON format:
          # readerSchemaFile : /path/to/syslog.avsc
        }
      }
    ]
  }
  {
    # Consume the output record of the previous command and pipe an
    # other
    # record downstream.
    #
    # extractAvroPaths is a command that uses zero or more Avro path
    # # excodeblockssions to extract values from an Avro object. Each ex
    # codeblockssion
    # consists of a record output field name, which appears to the le
    # ft of the
```

```

        # colon ':' and zero or more path steps, which appear to the right
        # Each path step is separated by a '/' slash. Avro arrays are
        # traversed with the '[]' notation.
        #
        # The result of a path expression is a list of objects, each
of which
        # is added to the given record output field.
        #
        # The path language supports all Avro concepts, including nested
        # structures, records, arrays, maps, unions, and others, as well a
s a flatten
        # option that collects the primitives in a subtree into a flat l
ist. In the
        # paths specification, entries on the left of the colon are the
target Solr
        # field and entries on the right specify the Avro source paths.
Paths are read
        # from the source that is named to the right of the colon and wri
tten to the
        # field that is named on the left.
        extractAvroPaths {
            flatten : false
            paths : {
                id : /id
                username : /user_screen_name
                created_at : /created_at
                text : /text
            }
        }
    }

    # Consume the output record of the previous command and pipe another
    # record downstream.
    #
    # convert timestamp field to native Solr timestamp format
    # such as 2012-09-06T07:14:34Z to 2012-09-06T07:14:34.000Z
    {
        convertTimestamp {
            field : created_at
            inputFormats : ["yyyy-MM-dd'T'HH:mm:ss'Z'", "yyyy-MM-dd"]
            inputTimezone : America/Los_Angeles
            outputFormat : "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"
            outputTimezone : UTC
        }
    }

    # Consume the output record of the previous command and pipe another
    # record downstream.
    #
    # This command deletes record fields that are unknown to Solr
    # schema.xml.
    #
    # Recall that Solr throws an exception on any attempt to load a do
cument
    # that contains a field that is not specified in schema.xml.
    {
        sanitizeUnknownSolrFields {
            # Location from which to fetch Solr schema
            solrLocator : ${SOLR_LOCATOR}
        }
    }

    # log the record at DEBUG level to SLF4J

```

```

    { logDebug { format : "output record: {}", args : ["@{}"] } }
    # load the record into a Solr server or MapReduce Reducer
    {
      loadSolr {
        solrLocator : ${SOLR_LOCATOR}
      }
    }
  ]
}
]

```

Related Information

[Morphlines Reference Guide](#)

Using Morphlines with Syslog

This example illustrates using a morphline to extract information from a syslog file.

A syslog file contains semi-structured lines of the following form:

```
<164>Feb  4 10:46:14 syslog sshd[607]: listening on 0.0.0.0 port 22.
```

The program extracts the following record from the log line and loads it into Solr:

```

syslog_pri:164
syslog_timestamp:Feb  4 10:46:14
syslog_hostname:syslog
syslog_program:sshd
syslog_pid:607
syslog_message:listening on 0.0.0.0 port 22.

```

Use the following rules to create a chain of transformation commands, which are expressed with the [readLine](#), [grok](#), and [logDebug](#) morphline commands, by editing a morphline.conf file.

```

# Specify server locations in a SOLR_LOCATOR variable; used later in
# variable substitutions:
SOLR_LOCATOR : {
  # Name of solr collection
  collection : collection1

  # ZooKeeper ensemble
  zkHost : "127.0.0.1:2181/solr"
}

# Specify an array of one or more morphlines, each of which defines an ETL
# transformation chain. A morphline consists of one or more potentially
# nested commands. A morphline is a way to consume records such as Flume e
# vents,
# HDFS files or blocks, turn them into a stream of records, and pipe the
# stream
# of records through a set of easily configurable transformations on the
# way to
# a target application such as Solr.
morphlines : [
  {
    id : morphline1
    importCommands : ["org.kitesdk.**"]
  }
]

```

```

    commands : [
      {
        readLine {
          charset : UTF-8
        }
      }
      {
        grok {
          # a grok-dictionary is a config file that contains prefabricated r
          # egular expressions
          # that can be referred to by name.
          # grok patterns specify such a regex name, plus an optional output
          # field name.
          # The syntax is %{REGEX_NAME:OUTPUT_FIELD_NAME}
          # The input line is expected in the "message" input field.
          dictionaryFiles : [target/test-classes/grok-dictionaries]
          expressions : {
            message : """<{%{POSINT:syslog_pri}>{%{SYSLOGTIMESTAMP:syslog_ti
            mestamp} %{SYSLOGHOST:syslog_hostname} %{DATA:syslog_program}(?:\[
            syslog_pid}\])?: %{GREEDYDATA:syslog_message}""""
          }
        }
      }

      # Consume the output record of the previous command and pipe another
      # record downstream.
      #
      # This command deletes record fields that are unknown to Solr
      # schema.xml.
      #
      # Recall that Solr throws an exception on any attempt to load a docum
      # ent
      # that contains a field that is not specified in schema.xml.
      {
        sanitizeUnknownSolrFields {
          # Location from which to fetch Solr schema
          solrLocator : ${SOLR_LOCATOR}
        }
      }

      # log the record at DEBUG level to SLF4J
      { logDebug { format : "output record: {}", args : ["@{}"] } }

      # load the record into a Solr server or MapReduce Reducer
      {
        loadSolr {
          solrLocator : ${SOLR_LOCATOR}
        }
      }
    ]
  }
]

```

Related Information

[Morphlines Reference Guide](#)