cloudera®

Cloudera Search

Important Notice

Cloudera, Inc.
1001 Page Mill Road, Bldg 3
Palo Alto, CA 94304
info@cloudera.com
US: 1-888-789-1488
Intl: 1-650-362-0488
www.cloudera.com

Release Information

Version: Cloudera Search 5.9.x
Date: November 2, 2016

Table of Contents

Cloudera Search QuickStart Guide

This guide shows how to establish and use a sample deployment to query a real data set. At a high level, you set up a cluster, enable search, run a script to create an index and load data, and then run queries.

Prerequisites for Cloudera Search QuickStart Scenarios

Before installing Search, install Cloudera Manager and a CDH cluster. The scenario in this guide works with CDH 5.9.x and Cloudera Manager 5.9.x. The `quickstart.sh` script and supporting files are included with CDH. Install Cloudera Manager, CDH, and Solr using the Cloudera Manager and CDH QuickStart Guide.

The primary services that the Search Quick Start depends on are:

- HDFS: Stores data. Deploy on all hosts.
- ZooKeeper: Coordinates Solr hosts. Deploy on one host. Use default port 2181. The examples refer to a machine named `search-zk`. You may want to give your Zookeeper machine this name to simplify reusing content exactly as it appears in this document. If you choose a different name, you must adjust some commands accordingly.
- Solr with SolrCloud: Provides search services such as document indexing and querying. Deploy on two hosts.
- Hue: Includes the Search application, which you can use to complete search queries. Deploy Hue on one host.

After you have completed the installation processes outlined in the Cloudera Manager Quick Start Guide, you can Load and Index Data in Search on page 6.

Load and Index Data in Search

Run the script found in a subdirectory of the following locations. The path for the script often includes the product version, such as Cloudera Manager 5.9.x, so path details vary. To address this issue, use wildcards.

- Packages: `/usr/share/doc`. If Search for CDH 5.9.0 is installed to the default location using packages, the Quick Start script is found in `/usr/share/doc/search-*/quickstart`.
- Parcels: `/opt/cloudera/parcels/CDH/share/doc`. If Search for CDH 5.9.0 is installed to the default location using parcels, the Quick Start script is found in `/opt/cloudera/parcels/CDH/share/doc/search-*/quickstart`.

The script uses several defaults that you might want to modify:

Table 1: Script Parameters and Defaults

Parameter	Default	Notes
NAMENODE_CONNECT	`` `hostname`:8020 ``	For use on an HDFS HA cluster. If you use NAMENODE_CONNECT, do not use NAMENODE_HOST or NAMENODE_PORT.
NAMENODE_HOST	`` `hostname` ``	If you use NAMENODE_HOST and NAMENODE_PORT, do not use NAMENODE_CONNECT.
NAMENODE_PORT	8020	If you use NAMENODE_HOST and NAMENODE_PORT, do not use NAMENODE_CONNECT.
ZOOKEEPER_ENSEMBLE	`` `hostname`:2181/solr ``	Zookeeper ensemble to point to. For example: `zk1,zk2,zk3:2181/solr`

Parameter	Default	Notes
		If you use ZOOKEEPER_ENSEMBLE, do not use ZOOKEEPER_HOST or ZOOKEEPER_PORT, ZOOKEEPER_ROOT.
ZOOKEEPER_HOST	`hostname`	
ZOOKEEPER_PORT	2181	
ZOOKEEPER_ROOT	/solr	
HDFS_USER	${HDFS_USER:="${USER}"}	
SOLR_HOME	/opt/cloudera/parcels/SOLR/lib/solr	

By default, the script is configured to run on the NameNode host, which is also running ZooKeeper. Override these defaults with custom values when you start `quickstart.sh`. For example, to use an alternate NameNode and HDFS user ID, you could start the script as follows:

```
$ NAMENODE_HOST=nnhost HDFS_USER=jsmith ./quickstart.sh
```

The first time the script runs, it downloads required files such as the Enron data and configuration files. If you run the script again, it uses the Enron information already downloaded, as opposed to downloading this information again. On such subsequent runs, the existing data is used to re-create the `enron-email-collection` SolrCloud collection.

Note: Downloading the data from its server, expanding the data, and uploading the data can be time consuming. Although your connection and CPU speed determine the time these processes require, fifteen minutes is typical and longer is not uncommon.

The script also generates a Solr configuration and creates a collection in SolrCloud. The following sections describes what the script does and how you can complete these steps manually, if desired. The script completes the following tasks:

1. Set variables such as hostnames and directories.
2. Create a directory to which to copy the Enron data and then copy that data to this location. This data is about 422 MB and in some tests took about five minutes to download and two minutes to untar.
3. Create directories for the current user in HDFS, change ownership of that directory to the current user, create a directory for the Enron data, and load the Enron data to that directory. In some tests, it took about a minute to copy approximately 3 GB of untarred data.
4. Use `solrctl` to create a template of the instance directory.
5. Use `solrctl` to create a new Solr collection for the Enron mail collection.
6. Create a directory to which the MapReduceBatchIndexer can write results. Ensure that the directory is empty.
7. Use the MapReduceIndexerTool to index the Enron data and push the result live to `enron-mail-collection`. In some tests, it took about seven minutes to complete this task.

Using Search to Query Loaded Data

After loading data into Search as described in Load and Index Data in Search on page 6, you can use Hue to query data.

Hue must have admin privileges to query loaded data. This is because querying requires Hue import collections or indexes, and these processes can only be completed with admin permissions on the Solr service.

1. Connect to Cloudera Manager and click the **Hue** service, which is often named something like HUE-1. Click **Hue Web UI**.
2. Click the **Search** menu.
3. Select the Enron collection for import.

4. (Optional) Click the Enron collection to configure how the search results display. For more information, see Hue Configuration.
5. Type a search string in the **Search...** text box and press **Enter**.
6. Review the results of your Search.

Cloudera Search User Guide

This guide explains how to configure and use Cloudera Search. This includes topics such as extracting, transforming, and loading data, establishing high availability, and troubleshooting.

This guide covers the following topics:

Cloudera Search Tutorial

The topics in this tutorial document assume you have completed the instructions in the Cloudera Search Installation Guide.

This tutorial is intended for use in an unsecured environment. In an environment that requires Kerberos authentication, this tutorial cannot be completed without additional configuration.

This tutorial first describes preparatory steps:

- Validating the Deployment with the Solr REST API
- Preparing to Index Data

Following are two tutorial topics, including indexing strategies:

- Batch Indexing Using MapReduce
- Near Real Time (NRT) Indexing Using Flume and the Solr Sink

This tutorial uses a modified `schema.xml` and `solrconfig.xml` file. In the versions of these files included with the tutorial, unused fields have been removed for simplicity. Original versions of these files include many additional options. For information on all available options, see the Solr wiki:

- SchemaXml
- SolrConfigXml

Validating the Cloudera Search Deployment

After installing Cloudera Search, you can validate the deployment by indexing and querying sample documents. Before beginning this process, make sure you have access to the Apache Solr admin web console, as described in Deploying Cloudera Search.

Note: The following procedure succeeds only if Kerberos is disabled.

Creating a Solr Collection

1. On a host running Solr Server, make sure that the `SOLR_ZK_ENSEMBLE` environment variable is set in `/etc/solr/conf/solr-env.sh`. For example:

```
$ cat /etc/solr/conf/solr-env.sh
export
SOLR_ZK_ENSEMBLE=zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181/solr
```

   If you are using Cloudera Manager, this is automatically set on hosts with a **Solr Server** or **Gateway** role.

2. Generate configuration files for the collection:

```
$ solrctl instancedir --generate $HOME/test_config
```

3. Upload the configuration to ZooKeeper:

```
$ solrctl instancedir --create test_config $HOME/test_config
```

4. Create a new collection with two shards using the uploaded configuration directory:

```
$ solrctl collection --create test_collection -s 2 -c test_config
```

Indexing Sample Data

Cloudera Search includes sample data for testing and validation. Run the following commands to index this data for searching. Replace *search01.example.com* in the example below with the name of any host running the Solr Server process.

- **Parcel-based Installation:**

```
$ cd /opt/cloudera/parcels/CDH/share/doc/solr-doc*/example/exampledocs
$ java -Durl=http://search01.example.com:8983/solr/test_collection/update -jar post.jar
 *.xml
```

- **Package-based Installation:**

```
$ cd /usr/share/doc/solr-doc*/example/exampledocs
$ java -Durl=http://search01.example.com:8983/solr/test_collection/update -jar post.jar
 *.xml
```

Querying Sample Data

Run a query to verify that the sample data is successfully indexed and that you are able to search it:

1. Open the Solr admin web interface in a browser by accessing `http://search01.example.com:8983/solr`. Replace *search01.example.com* with the name of any host running the Solr Server process.
2. Select **Cloud** from the left panel.
3. Select one of the hosts listed for the `test_collection` collection.
4. From the **Core Selector** menu in the left panel, select the `test_collection` shard.
5. Select **Query** from the left panel and select **Execute Query**. If you see results such as the following, indexing was successful:

```
 "response": {
   "numFound": 32,
   "start": 0,
   "maxScore": 1,
   "docs": [
     {
       "id": "SP2514N",
       "name": "Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133",
       "manu": "Samsung Electronics Co. Ltd.",
       "manu_id_s": "samsung",
       "cat": [
         "electronics",
         "hard drive"
       ],
```

Note: For more readable output, set the **wt** parameter to **json** and select the **indent** option before running the query.

Next Steps

After you have verified that Cloudera Search is installed and running properly, you can experiment with other methods of ingesting and indexing data:

- Preparing to Index Data with Cloudera Search on page 11
- Using MapReduce Batch Indexing with Cloudera Search on page 12
- Near Real Time (NRT) Indexing Using Flume and the Solr Sink on page 15
- Using Hue with Cloudera Search on page 20

To learn more about Solr, see the Apache Solr Tutorial.

Preparing to Index Data with Cloudera Search

To prepare for indexing example data with MapReduce or Flume, complete the following steps

1. Start a SolrCloud cluster containing two servers (this example uses two shards) as described in Deploying Cloudera Search.
2. Verify the Runtime Solr Configuration.
3. On a host running Solr Server, make sure that the `SOLR_ZK_ENSEMBLE` environment variable is set in `/etc/solr/conf/solr-env.sh`. For example:

```
$ cat /etc/solr/conf/solr-env.sh
export
SOLR_ZK_ENSEMBLE=zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181/solr
```

   If you are using Cloudera Manager, this is automatically set on hosts with a **Solr Server** or **Gateway** role.

4. Generate the configuration files for the collection, including the tweet-specific `schema.xml`:

   - **Parcel-based Installation:**

```
$ solrctl instancedir --generate $HOME/twitter_config
$ cp
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/collection1/conf/schema.xml
 \
$HOME/twitter_config/conf
```

   - **Package-based Installation:**

```
$ solrctl instancedir --generate $HOME/twitter_config
$ cp /usr/share/doc/search*/examples/solr-nrt/collection1/conf/schema.xml \
$HOME/twitter_config/conf
```

5. Upload the configuration to ZooKeeper:

```
$ solrctl instancedir --create twitter_config $HOME/twitter_config/
```

6. Create a new collection:

```
$ solrctl collection --create tweets -s 2 -c twitter_config
```

7. Verify the collection is live. Open the Solr admin web interface in a browser by accessing `http://search01.example.com:8983/solr/#/~cloud`. Replace *solr01.example.com* with the name of any host running the Solr Server process. Look for the `tweets` collection.
8. Prepare the configuration for use with MapReduce:

```
$ cp -r $HOME/twitter_config $HOME/mr_twitter_config
```

9. Locate input files suitable for indexing, and check that the directory exists. This example assumes you are running the following commands as $USER with access to HDFS.

- **Parcel-based Installation:**

```
$ sudo -u hdfs hadoop fs -mkdir -p /user/$USER
$ sudo -u hdfs hadoop fs -chown $USER:$USER /user/$USER
$ hadoop fs -mkdir -p /user/$USER/indir
$ hadoop fs -copyFromLocal \
/opt/cloudera/parcels/CDH/share/doc/search*/examples/test-documents/sample-statuses-*.avro
 \
/user/$USER/indir/
$ hadoop fs -ls /user/$USER/indir
```

- **Package-based Installation:**

```
$ sudo -u hdfs hadoop fs -mkdir -p /user/$USER
$ sudo -u hdfs hadoop fs -chown $USER:$USER /user/$USER
$ hadoop fs -mkdir -p /user/$USER/indir
$ hadoop fs -copyFromLocal \
/usr/share/doc/search*/examples/test-documents/sample-statuses-*.avro \
/user/$USER/indir/
$ hadoop fs -ls /user/$USER/indir
```

10. Ensure that `outdir` is empty and exists in HDFS:

```
$ hadoop fs -rm -r -skipTrash /user/$USER/outdir
$ hadoop fs -mkdir /user/$USER/outdir
$ hadoop fs -ls /user/$USER/outdir
```

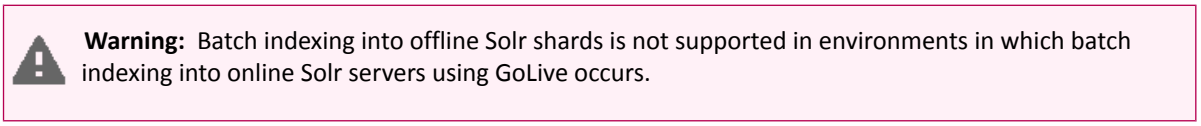
11. Depending on your installation mechanism for the Hadoop cluster, collect HDFS/MapReduce configuration details by downloading them from Cloudera Manager or by using `/etc/hadoop`, . This example uses the configuration in `/etc/hadoop/conf.cloudera.mapreduce1`. Substitute the correct Hadoop configuration path for your cluster.

Using MapReduce Batch Indexing with Cloudera Search

The following sections include examples that illustrate using MapReduce to index tweets. These examples require that you:

- Complete the process of Preparing to Index Data.
- Install the MapReduce tools for Cloudera Search as described in Installing MapReduce Tools for use with Cloudera Search.

Batch Indexing into Online Solr Servers Using GoLive



MapReduceIndexerTool is a MapReduce batch job driver that creates a set of Solr index shards from a set of input files and writes the indexes into HDFS in a flexible, scalable, and fault-tolerant manner. Using GoLive, MapReduceIndexerTool also supports merging the output shards into a set of live customer-facing Solr servers, typically a SolrCloud. The following sample steps demonstrate these capabilities.

1. Delete all existing documents in Solr.

```
$ solrctl --zk $SOLR_ZK_ENSEMBLE collection --deletedocs collection1
```

2. Run the MapReduce job using GoLive. Replace `$NNHOST` and `$ZKHOST` in the command with your NameNode and ZooKeeper hostnames and port numbers, as required. You do not need to specify `--solr-home-dir` because the job accesses it from ZooKeeper.

- **Parcel-based Installation:**

```
$ hadoop --config /etc/hadoop/conf.cloudera.mapreduce1 jar \
/opt/cloudera/parcels/CDH/lib/solr/contrib/mr/search-mr-*-job.jar \
org.apache.solr.hadoop.MapReduceIndexerTool -D \
'mapred.child.java.opts=-Xmx500m' --log4j \
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/log4j.properties
--morphline-file \
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/test-morphlines/tutorialReadAvroContainer.conf
 \
--output-dir hdfs://$NNHOST:8020/user/$USER/outdir --verbose --go-live \
--zk-host $ZKHOST:2181/solr --collection collection1 \
hdfs://$NNHOST:8020/user/$USER/indir
```

- **Package-based Installation:**

```
$ hadoop --config /etc/hadoop/conf.cloudera.mapreduce1 jar \
/usr/lib/solr/contrib/mr/search-mr-*-job.jar \
org.apache.solr.hadoop.MapReduceIndexerTool -D \
'mapred.child.java.opts=-Xmx500m' --log4j \
/usr/share/doc/search*/examples/solr-nrt/log4j.properties --morphline-file \
/usr/share/doc/search*/examples/solr-nrt/test-morphlines/tutorialReadAvroContainer.conf
 \
--output-dir hdfs://$NNHOST:8020/user/$USER/outdir --verbose --go-live \
--zk-host $ZKHOST:2181/solr --collection collection1 \
hdfs://$NNHOST:8020/user/$USER/indir
```

This command requires a morphline file, which must include a `SOLR_LOCATOR`. Any CLI parameters for `--zkhost` and `--collection` override the parameters of the `solrLocator`. The snippet that includes the `SOLR_LOCATOR` might appear as follows:

```
SOLR_LOCATOR : {
  # Name of solr collection
  collection : collection

  # ZooKeeper ensemble
  zkHost : "$ZK_HOST"
}

morphlines : [
  {
    id : morphline1
    importCommands : ["org.kitesdk.**", "org.apache.solr.**"]
    commands : [
      { generateUUID { field : id } }

      { # Remove record fields that are unknown to Solr schema.xml.
        # Recall that Solr throws an exception on any attempt to load a document that
        # contains a field that isn't specified in schema.xml.
        sanitizeUnknownSolrFields {
          solrLocator : ${SOLR_LOCATOR} # Location from which to fetch Solr schema
        }
      }

      { logDebug { format : "output record: {}", args : ["@{}"] } }

      {
        loadSolr {
          solrLocator : ${SOLR_LOCATOR}
        }
      }
    ]
  }
]
```

3. Check the job tracker status at `http://localhost:50030/jobtracker.jsp`.
4. When the job is complete, run some Solr queries. For example, for `myserver.example.com`, use: `http://myserver.example.com:8983/solr/collection1/select?q=*%3A*&wt=json&indent=true`

For help on how to run a Hadoop MapReduce job, use the following command:

- **Parcel-based Installation:**

```
$ hadoop jar /opt/cloudera/parcels/CDH/lib/solr/contrib/mr/search-mr-*-job.jar \
org.apache.solr.hadoop.MapReduceIndexerTool --help
```

- **Package-based Installation:**

```
$ hadoop jar /usr/lib/solr/contrib/mr/search-mr-*-job.jar \
org.apache.solr.hadoop.MapReduceIndexerTool --help
```

- For development purposes, use the `MapReduceIndexerTool --dry-run` option to run in local mode and print documents to `stdout`, instead of loading them to Solr. Using this option causes the morphline to run in the client process without submitting a job to MapReduce. Running in the client process provides faster turnaround during early trial and debug sessions.
- To print diagnostic information, such as the content of records as they pass through the morphline commands, enable TRACE log level diagnostics by adding the following entry to your `log4j.properties` file:

```
log4j.logger.com.cloudera.cdk.morphline=TRACE
```

The `log4j.properties` file can be passed using the `MapReduceIndexerTool --log4j` command-line option.

Batch Indexing into Offline Solr Shards

Running the MapReduce job without GoLive causes the job to create a set of Solr index shards from a set of input files and write the indexes to HDFS. You can then explicitly point each Solr server to one of the HDFS output shard directories.

Batch indexing into offline Solr shards is mainly intended for offline use-cases by experts. Cases requiring read-only indexes for searching can be handled using batch indexing without the `--go-live` option. By not using GoLive, you can avoid copying datasets between segments, thereby reducing resource demands.

1. Delete all existing documents in Solr.

```
$ solrctl --zk $SOLR_ZK_ENSEMBLE collection --deletedocs collection1
$ sudo -u hdfs hadoop fs -rm -r -skipTrash /user/$USER/outdir
```

2. Run the Hadoop MapReduce job, replacing `$NNHOST` in the command with your NameNode hostname and port number, as required.

- **Parcel-based Installation:**

```
$ hadoop --config /etc/hadoop/conf.cloudera.mapreduce1 jar \
/opt/cloudera/parcels/CDH/lib/solr/contrib/mr/search-mr-*-job.jar \
org.apache.solr.hadoop.MapReduceIndexerTool -D \
'mapred.child.java.opts=-Xmx500m' --log4j \
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/log4j.properties
--morphline-file \
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/test-morphlines/tutorialReadAvroContainer.conf
 \
--output-dir hdfs://$NNHOST:8020/user/$USER/outdir --verbose --solr-home-dir \
$HOME/collection1 --shards 2 hdfs://$NNHOST:8020/user/$USER/indir
```

- **Package-based Installation:**

```
$ hadoop --config /etc/hadoop/conf.cloudera.mapreduce1 jar \
/usr/lib/solr/contrib/mr/search-mr-*-job.jar \
org.apache.solr.hadoop.MapReduceIndexerTool -D \
'mapred.child.java.opts=-Xmx500m' --log4j \
/usr/share/doc/search*/examples/solr-nrt/log4j.properties --morphline-file \
/usr/share/doc/search*/examples/solr-nrt/test-morphlines/tutorialReadAvroContainer.conf
```

```
 \
--output-dir hdfs://$NNHOST:8020/user/$USER/outdir --verbose --solr-home-dir \
$HOME/collection1 --shards 2 hdfs://$NNHOST:8020/user/$USER/indir
```

3. Check the job tracker status. For example, for the localhost, use `http://localhost:50030/jobtracker.jsp`.
4. After the job is completed, check the generated index files. Individual shards are written to the results directory with names of the form `part-00000`, `part-00001`, `part-00002`. This example has two shards.

```
$ hadoop fs -ls /user/$USER/outdir/results
$ hadoop fs -ls /user/$USER/outdir/results/part-00000/data/index
```

5. Stop Solr on each host of the cluster.

```
$ sudo service solr-server stop
```

6. List the hostname folders used as part of the path to each index in the SolrCloud cluster.

```
$ hadoop fs -ls /solr/collection1
```

7. Move index shards into place.

   a. Remove outdated files:

```
$ sudo -u solr hadoop fs -rm -r -skipTrash \
/solr/collection1/$HOSTNAME1/data/index
$ sudo -u solr hadoop fs -rm -r -skipTrash \
/solr/collection1/$HOSTNAME2/data/tlog
```

   b. Ensure correct ownership of required directories:

```
$ sudo -u hdfs hadoop fs -chown -R solr /user/$USER/outdir/results
```

   c. Move the two index shards into place (the two servers you set up in Preparing to Index Data with Cloudera Search on page 11):

```
$ sudo -u solr hadoop fs -mv /user/$USER/outdir/results/part-00000/data/index \
/solr/collection1/$HOSTNAME1/data/
$ sudo -u solr hadoop fs -mv /user/$USER/outdir/results/part-00001/data/index \
/solr/collection1/$HOSTNAME2/data/
```

8. Start Solr on each host of the cluster:

```
$ sudo service solr-server start
```

9. Run some Solr queries. For example, for `myserver.example.com`, use:
   `http://myserver.example.com:8983/solr/collection1/select?q=*%3A*&wt=json&indent=true`

Near Real Time (NRT) Indexing Using Flume and the Solr Sink

The following section describes how to use Flume to index tweets. Before beginning, complete the process of Preparing to Index Data.

Deploying Solr Sink into the Flume Agent

Copy the configuration files.

- **Parcel-based Installation:**

```
$ sudo cp -r $HOME/twitter_config /etc/flume-ng/conf/tweets
$ sudo cp /opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/twitter-flume.conf
```

```
 \
/etc/flume-ng/conf/flume.conf
$ sudo cp
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/test-morphlines/tutorialReadAvroContainer.conf
 \
/etc/flume-ng/conf/morphlines.conf
```

- **Package-based Installation:**

```
$ sudo cp -r $HOME/twitter_config /etc/flume-ng/conf/tweets
$ sudo cp /usr/share/doc/search*/examples/solr-nrt/twitter-flume.conf \
/etc/flume-ng/conf/flume.conf
$ sudo cp
/usr/share/doc/search*/examples/solr-nrt/test-morphlines/tutorialReadAvroContainer.conf
 \
/etc/flume-ng/conf/morphlines.conf
```

Configuring the Flume Solr Sink

This topic describes how to configure Flume Solr Sink for both parcel-based and package-based installations:

- For parcel-based installations, use Cloudera Manager to edit the configuration files similar to the process described in Configuring the Flume Agents.
- For package-based installations, use command-line tools to edit files.

1. Modify the Flume configuration to specify the Flume source details and set up the flow. You must set the relative or absolute path to the morphline configuration file.
   - **Parcel-based Installation:** In the Cloudera Manager Admin Console, select **Flume** > **Configuration** and modify **Configuration File** to include:

```
agent.sinks.solrSink.morphlineFile = morphlines.conf
```

   - **Package-based Installation:** Edit `/etc/flume-ng/conf/flume.conf` to include:

```
agent.sinks.solrSink.morphlineFile = /etc/flume-ng/conf/morphlines.conf
```

2. Use a `SOLR_LOCATOR` to modify the Morphline configuration to specify the Solr location details.
   - **Parcel-based Installation:** In the Cloudera Manager Admin Console, select **Flume** > **Configuration** and modify **Morphline File**.
   - **Package-based Installation:** Edit `/etc/flume-ng/conf/morphline.conf`.

   The snippet that includes the `SOLR_LOCATOR` might appear as follows:

```
SOLR_LOCATOR : {
  # Name of solr collection
  collection : collection

  # ZooKeeper ensemble
  zkHost : "$ZK_HOST"
}

morphlines : [
  {
    id : morphline1
    importCommands : ["org.kitesdk.**", "org.apache.solr.**"]
    commands : [
      { generateUUID { field : id } }

      { # Remove record fields that are unknown to Solr schema.xml.
        # Recall that Solr throws an exception on any attempt to load a document that
        # contains a field that isn't specified in schema.xml.
        sanitizeUnknownSolrFields {
          solrLocator : ${SOLR_LOCATOR} # Location from which to fetch Solr schema
```

```
      }
    }

    { logDebug { format : "output record: {}", args : ["@{}"] } }

    {
      loadSolr {
        solrLocator : ${SOLR_LOCATOR}
      }
    }
  ]
}
]
```

3. Copy `flume-env.sh.template` to `flume-env.sh`:

   - **Parcel-based Installation:**

```
$ sudo cp /opt/cloudera/parcels/CDH/etc/flume-ng/conf/flume-env.sh.template \
/opt/cloudera/parcels/CDH/etc/flume-ng/conf/flume-env.sh
```

   - **Package-based Installation:**

```
$ sudo cp /etc/flume-ng/conf/flume-env.sh.template \
/etc/flume-ng/conf/flume-env.sh
```

4. Update the Java heap size.

   - **Parcel-based Installation:** In the Cloudera Manager Admin Console, select **Flume** > **Configuration**. In the **Search** box enter `Java Heap Size`. Modify **Java Heap Size of Agent in Bytes** to be `500` and choose **MiB** units.
   - **Package-based Installation:** Edit `/etc/flume-ng/conf/flume-env.sh` or `/opt/cloudera/parcels/CDH/etc/flume-ng/conf/flume-env.sh`, inserting or replacing `JAVA_OPTS` as follows:

```
JAVA_OPTS="-Xmx500m"
```

5. (Optional) Modify Flume logging settings to facilitate monitoring and debugging:

   - **Parcel-based Installation:** In the Cloudera Manager Admin Console, select **Flume** > **Configuration** and modify **Agent Logging Advanced Configuration Snippet (Safety Valve)** to include:

```
log4j.logger.org.apache.flume.sink.solr=DEBUG
log4j.logger.org.kitesdk.morphline=TRACE
```

   - **Package-based Installation:** Use the following commands:

```
$ sudo bash -c 'echo "log4j.logger.org.apache.flume.sink.solr=DEBUG" >> \
/etc/flume-ng/conf/log4j.properties'
$ sudo bash -c 'echo "log4j.logger.org.kitesdk.morphline=TRACE" >> \
/etc/flume-ng/conf/log4j.properties'
```

6. (Optional) In a packaged-based installation, you can use `SEARCH_HOME` to configure where Flume finds Cloudera Search dependencies for Flume Solr Sink. For example, if you installed Flume from a tarball package, you can configure it to find required files by setting `SEARCH_HOME`. To set `SEARCH_HOME` use a command similar to the following:

```
$ export SEARCH_HOME=/usr/lib/search
```

   Alternatively, you can add the same setting to `flume-env.sh`.

In a Cloudera Manager managed environment, Cloudera Manager automatically updates the SOLR_HOME location with any additional required dependencies.

Configuring Flume Solr Sink to Sip from the Twitter Firehose

Edit /etc/flume-ng/conf/flume.conf or /opt/cloudera/parcels/CDH/etc/flume-ng/conf/flume.conf and replace the following properties with credentials from a valid Twitter account. The Flume TwitterSource uses the Twitter 1.1 API, which requires authentication of both the consumer (application) and the user (you).

```
agent.sources.twitterSrc.consumerKey = YOUR_TWITTER_CONSUMER_KEY
agent.sources.twitterSrc.consumerSecret = YOUR_TWITTER_CONSUMER_SECRET
agent.sources.twitterSrc.accessToken = YOUR_TWITTER_ACCESS_TOKEN
agent.sources.twitterSrc.accessTokenSecret = YOUR_TWITTER_ACCESS_TOKEN_SECRET
```

Use the Twitter developer site to generate these four codes by completing the following steps:

1. Sign in to https://apps.twitter.com with a Twitter account.
2. Select **My applications** from the drop-down menu in the top-right corner, and **Create a new application**.
3. Fill in the form to represent the Search installation. This can represent multiple clusters, and does not require the callback URL. Because this is not a publicly distributed application, the values you enter for the required name, description, and website fields are not important.
4. Click **Create my access token** at the bottom of the page. You might have to refresh the page to see the access token.

Substitute the consumer key, consumer secret, access token, and access token secret into `flume.conf`. Consider this information confidential, just like your regular Twitter credentials.

To enable authentication, ensure the system clock is set correctly on all hosts where Flume connects to Twitter. You can install NTP and keep the host synchronized by running the `ntpd` service, or manually synchronize using the command `sudo ntpdate pool.ntp.org`. To confirm that the time is set correctly, make sure that the output of the command `date --utc` matches the time shown at http://www.timeanddate.com/worldclock/timezone/utc. You can also set the time manually using the `date` command.

Starting the Flume Agent

1. Delete all existing documents in Solr:

```
$ solrctl --zk collection --deletedocs collection1
```

2. Check the status of the Flume Agent to determine if it is running or not:

```
$ sudo /etc/init.d/flume-ng-agent status
```

3. Use the `start` or `restart` functions. For example, to restart a running Flume Agent:

```
$ sudo /etc/init.d/flume-ng-agent restart
```

4. Monitor progress in the Flume log file and watch for errors:

```
$ tail -f /var/log/flume-ng/flume.log
```

After restarting the Flume agent, use the Cloudera Search GUI. For example, for the localhost, use `http://localhost:8983/solr/collection1/select?q=*%3A*&sort=created_at+desc&wt=json&indent=true` to verify that new tweets have been ingested into Solr. The query sorts the result set such that the most recently ingested tweets are at the top, based on the `created_at` timestamp. If you rerun the query, new tweets show up at the top of the result set.

To print diagnostic information, such as the content of records as they pass through the morphline commands, enable TRACE log level diagnostics by adding the following to your `log4j.properties` file:

```
log4j.logger.com.cloudera.cdk.morphline=TRACE
```

In Cloudera Manager, you can use the safety valve to enable TRACE log level.

Go to **Menu Services** > **Flume** > **Configuration** > **View and Edit** > **Agent** > **Advanced** > **Agent Logging Safety Valve**. After setting this value, restart the service.

Indexing a File Containing Tweets with Flume HTTPSource

HTTPSource lets you ingest data into Solr by POSTing a file using HTTP. HTTPSource sends data using a channel to a sink, in this case a SolrSink. For more information, see Flume Solr BlobHandler Configuration Options on page 45.

1. Delete all existing documents in Solr:

```
$ sudo /etc/init.d/flume-ng-agent stop
$ solrctl --zk $SOLR_ZK_ENSEMBLE collection --deletedocs collection1
```

2. Comment out TwitterSource in `/etc/flume-ng/conf/flume.conf` and uncomment HTTPSource:

```
# comment out “agent.sources = twitterSrc”
# uncomment “agent.sources = httpSrc”
```

3. Restart the Flume Agent:

```
$ sudo /etc/init.d/flume-ng-agent restart
```

4. Send a file containing tweets to the HTTPSource:

   - **Parcel-based Installation:**

```
$ curl --data-binary \
@/opt/cloudera/parcels/CDH/share/doc/search-*/examples/test-documents/sample-statuses-20120906-141433-medium.avro
  \
'http://127.0.0.1:5140?resourceName=sample-statuses-20120906-141433-medium.avro' \
--header 'Content-Type:application/octet-stream' --verbose
```

   - **Package-based Installation:**

```
$ curl --data-binary \
@/usr/share/doc/search-*/examples/test-documents/sample-statuses-20120906-141433-medium.avro
  \
'http://127.0.0.1:5140?resourceName=sample-statuses-20120906-141433-medium.avro' \
--header 'Content-Type:application/octet-stream' --verbose
```

5. Check the log for status or errors:

```
$ cat /var/log/flume-ng/flume.log
```

Use the Cloudera Search GUI at `http://localhost:8983/solr/collection1/select?q=*%3A*&wt=json&indent=true` to verify that new tweets have been ingested into Solr as expected.

Indexing a File Containing Tweets with Flume SpoolDirectorySource

`SpoolDirectorySource` specifies a directory on a local disk that Flume monitors. Flume automatically transfers data from files in this directory to Solr. `SpoolDirectorySource` sends data using a channel to a sink, in this case a SolrSink.

1. Delete all existing documents in Solr:

```
$ sudo /etc/init.d/flume-ng-agent stop
$ solrctl collection --deletedocs collection1
```

2. Comment out `TwitterSource` and `HTTPSource` in `/etc/flume-ng/conf/flume.conf` and uncomment `SpoolDirectorySource`:

```
# Comment out "agent.sources = twitterSrc"
# Comment out “agent.sources = httpSrc”
“agent.sources = spoolSrc”
```

3. Delete any old spool directory and create a new spool directory:

```
$ rm -fr /tmp/myspooldir
$ sudo -u flume mkdir /tmp/myspooldir
```

4. Restart the Flume Agent:

```
$ sudo /etc/init.d/flume-ng-agent restart
```

5. Send a file containing tweets to the `SpoolDirectorySource`. To ensure no partial files are ingested, copy and then atomically move files:

   - **Parcel-based Installation:**

```
$ sudo -u flume cp \
/opt/cloudera/parcels/CDH/share/doc/search*/examples/test-documents/sample-statuses-20120906-141433-medium.avro
 \
/tmp/myspooldir/.sample-statuses-20120906-141433-medium.avro
$ sudo -u flume mv /tmp/myspooldir/.sample-statuses-20120906-141433-medium.avro \
/tmp/myspooldir/sample-statuses-20120906-141433-medium.avro
```

   - **Package-based Installation:**

```
$ sudo -u flume cp \
/usr/share/doc/search*/examples/test-documents/sample-statuses-20120906-141433-medium.avro
 \
/tmp/myspooldir/.sample-statuses-20120906-141433-medium.avro
$ sudo -u flume mv /tmp/myspooldir/.sample-statuses-20120906-141433-medium.avro \
/tmp/myspooldir/sample-statuses-20120906-141433-medium.avro
```

6. Check the log for status or errors.

```
$ cat /var/log/flume-ng/flume.log
```

7. Check the completion status.

```
$ find /tmp/myspooldir
```

Use the Cloudera Search GUI. For example, for the localhost, use `http://localhost:8983/solr/collection1/select?q=*%3A*&wt=json&indent=true` to verify that new tweets have been ingested into Solr.

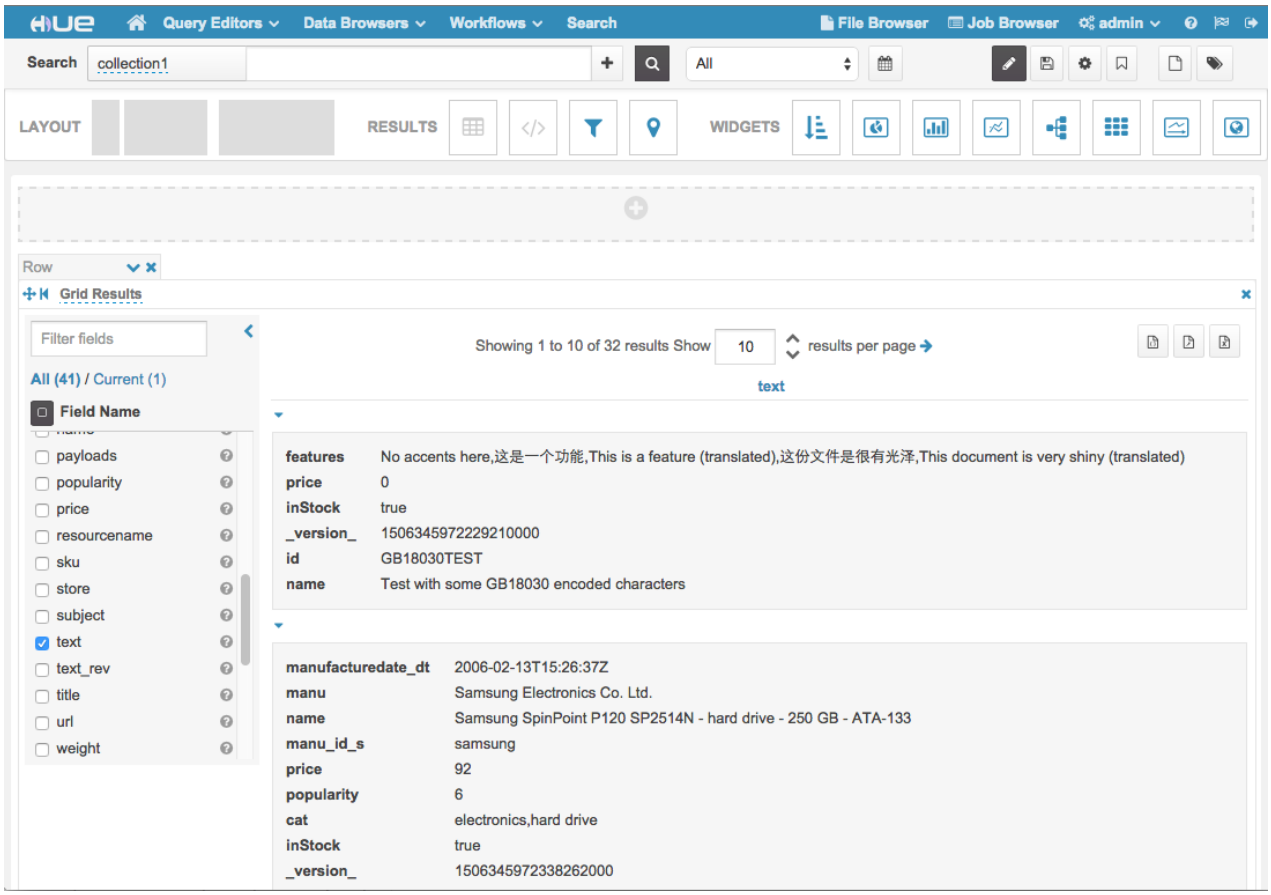
Using Hue with Cloudera Search

Hue includes a search application that provides a customizable UI. Using Hue with Cloudera Search involves importing collections. After you import collections, you can work with them through the Hue user interface.

You can use the Hue Web UI to create a new search. For example, for the server `myserver.example.com`, use: `http://myserver.example.com:8888/search/new_search`.

Search User Interface in Hue

The following figure shows the Search application integrated with the Hue user interface.



Managing Solr Using solrctl

Use the `solrctl` utility to manage a SolrCloud deployment. You can manipulate SolrCloud collections, SolrCloud collection instance directories, and individual cores.

A SolrCloud collection is the top-level object for indexing documents and providing a query interface. Each SolrCloud collection must be associated with an instance directory. Although, different collections can use the same instance directory. Each SolrCloud collection is typically replicated (sharded) among several SolrCloud instances. Each replica is called a SolrCloud core and is assigned to an individual SolrCloud host. The assignment process is managed automatically, although you can apply fine-grained control over each individual core using the `core` command. A typical deployment workflow with `solrctl` consists of:

- Deploying the ZooKeeper coordination service.
- Deploying solr-server daemons to each host.
- Initializing the state of the ZooKeeper coordination service using `init` command.
- Starting each solr-server daemon.
- Establishing a configuration.
  - If using Configs, creating a config from a config template.
  - If using instance directories, generating an instance directory and uploading the instance directory to ZooKeeper.
- Associating a new collection with the name of the config or instance directory.

Understanding configs and instancedirs

`solrctl` includes a `config` command that uses the Config API to directly manage configurations represented in `Config` objects. `Config` objects represent collection configuration information as specified by the `solrctl collection --create -c configName` command. instancedirs and `Config` objects handle the same information, meeting the same need from the Solr server perspective, but there a number of differences between these two implementations.

Table 2: Config and instancedir Comparison

Attribute	Config	instancedir
Security	Security support provided. • In a Kerberos-enabled cluster, the ZooKeeper hosts associated with configurations created using the Config API automatically has proper ZooKeeper ACLs. Because `instancedir` updates ZooKeeper directly, it is the client's responsibility to add the proper ACLs, which is cumbersome. • Sentry can be used to control access to the `Config` API, providing access control. For more information, see Configuring Sentry Authorization for Solr on page 87.	No ZooKeeper security support. Any user can create, delete, or modify instancedirs directly in ZooKeeper.
Creation method	Generated from existing configs or instancedirs in ZooKeeper using the ConfigSet API.	Manually edited locally and re-uploaded directly to ZooKeeper using `solrctl instancedir`.
Template support	Several predefined ***templates*** are available. These can be used as the basis for creating additional configs. Additional templates can be created by creating configs that are immutable. Mutable templates that use a Managed Schema can be modified using the Schema API as opposed to being manually edited. As a result, configs are less flexible, but they are also less error-prone than instancedirs.	One standard template.
Sentry support	Configs include a number of templates, each with Sentry-enabled and non-Sentry-enabled versions. To enable Sentry, choose a Sentry-enabled template.	instancedirs include a single template that supports enabling Sentry. To enable Sentry with instancedirs, overwrite the original `solrconfig.xml` file with `solrconfig.xml.secure` as described in Enabling Solr as a Client for the Sentry Service Using the Command Line.

Included Immutable Config Templates

Configs can be declared as `immutable`, which means they cannot be deleted or have their Schema updated by the Schema API. Immutable configs are uneditable config templates that are the basis for additional configs.

Solr provides a set of immutable config templates. These templates are only available after Solr initialization, so templates are not available in upgrades until after Solr is initialized or re-initialized. Templates include:

Table 3: Available Config Templates and Attributes

Template Name	Supports Schema API	Uses Schemaless Solr	Supports Sentry
predefinedTemplate			
managedTemplate	■		
schemalessTemplate	■	■	
predefinedTemplateSecure			■
managedTemplateSecure	■		■
schemalessTemplateSecure	■	■	■

solrctl Reference

Use the `solrctl` utility to manage a SolrCloud deployment. You can manipulate SolrCloud collections, SolrCloud collection instance directories, and individual cores.

In general, if an operation succeeds, `solrctl` exits silently with a success exit code. If an error occurs, `solrctl` prints a diagnostics message combined with a failure exit code. `solrctl` supports specifying a `log4j.properties` file by setting the `LOG4J_PROPS` environment variable. By default, the `LOG4J_PROPS` setting specifies the `log4j.properties` in the solr configuration directory. For example, `/etc/solr/conf/log4j.properties`. Many `solrctl` commands redirect `stderr` to `/dev/null`, so Cloudera recommends that your `log4j` properties file specify a location other than `stderr` for log output.

You can run `solrctl` on any host that is configured as part of the SolrCloud. To run any `solrctl` command on a host outside of SolrCloud deployment, ensure that SolrCloud hosts are reachable and provide `--zk` and `--solr` command line options.

If you are using `solrctl` to manage your deployment in an environment that requires Kerberos authentication, you must have a valid Kerberos ticket, which you can get using `kinit`.

You can see examples of using `solrctl` in Deploying Cloudera Search.

For collection configuration, users have the option of interacting directly with ZooKeeper using the `instancedir` option or using Solr's ConfigSet API using the `config` option. For more information, see Understanding configs and instancedirs on page 22.

Syntax

You can initialize the state of the entire SolrCloud deployment and each individual host within the SolrCloud deployment by using `solrctl`. The general `solrctl` command syntax is:

```
solrctl [options] command [command-arg] [command [command-arg]] ...
```

Each element and its possible values are described in the following sections.

Options

If used, the following options must precede commands:

- `--solr solr_uri`: Directs `solrctl` to a SolrCloud web API available at a given URI. This option is required for hosts running outside of SolrCloud. A sample URI might be: `http://host1.cluster.com:8983/solr`.

- `--zk zk_ensemble`: Directs `solrctl` to a particular ZooKeeper coordination service ensemble. This option is required for hosts running outside of SolrCloud. For example: `host1.cluster.com:2181,host2.cluster.com:2181/solr`. Output from solrctl commands that use the zkcli option is sent to /dev/null, so no results are displayed.
- `--jaas jaas.conf`: Used to identify a JAAS configuration that specifies the principal with permissions to modify solr metadata. The principal is typically "solr". In Kerberos-enabled environments where ZooKeeper ACLs protect solr metadata, you must use this parameter if you want to use `solrctl` to modify metadata.
- `--help`: Prints help.
- `--quiet`: Suppresses most `solrctl` messages.

Commands

The `solrctl` commands `init`, `instancedir`, `config`, `collection`, `core`, `cluster`, and `sentry` affect the entire SolrCloud deployment and are run only once per required operation.

The `solrctl core` command affects a single SolrCloud host.

- `init [--force]`: The `init` command, which initializes the overall state of the SolrCloud deployment, must be run before starting solr-server daemons for the first time. Use this command cautiously because it erases all SolrCloud deployment state information. After successful initialization, you cannot recover any previous state.
- `instancedir [--generate path [-schemaless]] [--create name path] [--update name path] [--get name path] [--delete name] [--list]`: Manipulates the instance directories. The following options are supported:
  - `--generate path`: Allows users to generate the template of the instance directory. The template is stored at a designated path in a local filesystem and has configuration files under /conf. See Solr's README.txt for the complete layout.
    - `-schemaless` A schemaless template of the instance directory is generated. For more information on schemaless support, see Schemaless Mode Overview and Best Practices on page 75.
  - `--create name path`: Pushes a copy of the instance directory from the local filesystem to SolrCloud. If an instance directory is already available to SolrCloud, this command fails. See `--update` for changing name paths that already exist.
  - `--update name path`: Updates an existiing SolrCloud copy of an instance directory based on the files in a local filesystem. This command is analogous to first using `--delete name` followed by `--create name path`.
  - `--get name path`: Downloads the named collection instance directory at a specified path in a local filesystem. Once downloaded, files can be further edited.
  - `--delete name`: Deletes the instance directory name from SolrCloud.
  - `--list`: Prints a list of all available instance directories known to SolrCloud.
- `config [--create name baseConfig [-p name=value]...] [--delete name]`: Manipulates configs. The following options are supported:
  - `--create name baseConfig [-p name=value]...`: Creates a new config based on an existing config. The config is created with the specified name, using baseConfig as the template. `-p` can be used to override a baseConfig setting. `immutable` is the only property that supports override. For more information about existing templates, see Included Immutable Config Templates on page 23.
  - `--delete name`: Deletes a config.
- `collection [--create name -s <numShards> [-c <collection.configName>] [-r <replicationFactor>] [-m <maxShardsPerHost>] [-n <createHostSet>]] [--delete name] [--reload name] [--stat name] [--list] [--deletedocs name]`: Manipulates collections. The following options are supported:

  - `--create name -s <numShards> [-a] [-c <collection.configName>] [-r <replicationFactor>] [-m <maxShardsPerHost>] [-n <createHostSet>]]`: Creates a new collection.

    New collections are given the specified name and are sharded to `<numShards>`.

    The `-a` option configures auto-addition of replicas if machines hosting existing shards become unavailable.

    SolrCloud hosts are configured using the `<collection.configName>` instance directory. Replication is configured by a factor of `<replicationFactor>`. The maximum shards per host is determined by `<maxShardsPerHost>`, and the collection is allocated to the hosts specified in `<createHostSet>`.

    The only required parameters are `name` and `numShards`. If `collection.configName` is not provided, it is assumed to be the same as the name of the collection.

  - `--delete name`: Deletes a collection.
  - `--reload name`: Reloads a collection.
  - `--stat name`: Outputs SolrCloud specific run-time information for a collection.
  - `--list`: Lists all collections registered in SolrCloud.
  - `--deletedocs name`: Purges all indexed documents from a collection.

- `core [--create name [-p name=value]...] [--reload name] [--unload name] [--status name]`: Manipulates cores. This is one of two commands that you can run on a particular SolrCloud host. The following options are supported:

  - `--create name [-p name=value]...`: Creates a new core on a specified SolrCloud host. The core is configured using `name=values` pairs. For more information about configuration options, see Solr documentation.
  - `--reload name`: Reloads a core.
  - `--unload name`: Unloads a core.
  - `--status name`: Prints status of a core.

- `sentry [--create-role role] [--drop-role role] [--add-role-group role group] [--delete-role-group role group] [--list-roles [-g group]] [--grant-privilege role privilege] [--revoke-privilege role privilege] [--list-privileges role] [--convert-policy-file file [-dry-run]]`: Manages sentry configuration. The following options are supported:

  - `[--create-role role]`: Creates a new Sentry role using the name specified.
  - `[--drop-role role]`: Deletes the specified Sentry role.
  - `[--add-role-group role group]`: Adds an existing Sentry role to the specified group.
  - `[--delete-role-group role group]`: Removes an existing Sentry role from the specified group.
  - `[--list-roles [-g group]]`: Lists all roles. Optionally, lists all roles in the specified group when `-g` is used.
  - `[--grant-privilege role privilege]`: Grants the specific privilege to the specified role.
  - `[--revoke-privilege role privilege]`: Revokes the privilege from the role.
  - `[--list-privileges role]`: Lists all privileges granted to the specified role.
  - `[--convert-policy-file file [-dry-run]]`: Converts the specified policy file to permissions in the Sentry service. This command adds existing roles, adds existing roles to group, and grants permissions.

    The file-based model allows case-sensitive role names. During conversion, all roles and groups are converted to lower case.

    - If a policy-file conversion will change the case of roles or groups, a warning is presented. Policy conversion can proceed, but if you have enabled document-level security and use role names as your tokens, you must re-index using the new lower case role names after conversion is complete.

- If a policy-file conversion will change the case of roles or groups, creating a name collision, an error occurs and conversion cannot occur. In such a case, you must eliminate the collisions before proceeding. For example, you could rename or delete all but one of the names that cause a collision.

The `dry-run` option runs the process of converting the policy file, but sends the results to stdout without applying the changes. This can be used for quicker turnaround during early trial debug sessions.

After converting the policy file to permissions in the Sentry service, you may want to enable Sentry for Solr, as described in Migrating from Sentry Policy Files to the Sentry Service.

Example solrctl Usage

This topic includes some examples of:

- Configuration changes that may be required for solrctl to function as desired.
- Common tasks completed with solrctl.

Using solrctl with an HTTP proxy

Using `solrctl` to manage a deployment in an environment that uses an `http_proxy` fails because `solrctl` uses `curl`, which attempts to use the web proxy. You can disable the proxy so `solrctl` succeeds:

- Modify the settings for the current shell by exporting the `NO_PROXY`. For example:

```
$ export NO_PROXY='*'
```

- Modify the settings for single commands by prefacing `solrctl` commands with `NO_PROXY='*'`. For example:

```
$ NO_PROXY='*' solrctl collection --create yourCollectionName
```

Adding Another Collection with Replication

To support scaling for the query load, create a second collection with replication. Having multiple servers with replicated collections distributes the request load for each shard. Create one shard cluster with a replication factor of two. Your cluster must have at least two running servers to support this configuration, so ensure Cloudera Search is installed on at least two servers. A replication factor of two causes two copies of the index files to be stored in two different locations.

1. Generate the config files for the collection:

```
$ solrctl instancedir --generate $HOME/solr_configs2
```

2. Upload the instance directory to ZooKeeper:

```
$ solrctl instancedir --create collection1 $HOME/solr_configs2
```

3. Create the second collection:

```
$ solrctl collection --create collection1 -s 1 -r 2
```

4. Verify that the collection is live and that the one shard is served by two hosts. For example, for the server `myhost.example.com`, you should receive content from: `http://myhost.example.com:8983/solr/#/~cloud`.

Creating Replicas of Existing Shards

You can create additional replicas of existing shards using a command of the following form:

```
$ solrctl --zk <zkensemble> --solr <target solr server> core \
--create <new core name> -p collection=<collection> -p shard=<shard to replicate>
```

For example to create a new replica of collection named `collection1` that is comprised of `shard1`, use the following command:

```
$ solrctl --zk myZKEnsemble:2181/solr --solr mySolrServer:8983/solr core \
--create collection1_shard1_replica2 -p collection=collection1 -p shard=shard1
```

Adding a New Shard to a Solr Server

You can use `solrctl` to add a new shard to a specified solr server.

```
$ solrctl --solr http://<target_solr_server>:8983/solr core --create <core_name> \
-p dataDir=hdfs://<nameservice>/<index_hdfs_path> -p collection.configName=<config_name>
 \
-p collection=<collection_name> -p numShards=<int> -p shard=<shard_id>
```

Where:

- `target_solr_server`: The server to host the new shard
- `core_name`: `<collection_name><shard_id><replica_id>`
- `shard_id`: New shard identifier

For example, to add a new second shard named `shard2` to a solr server named `mySolrServer`, where the collection is named `myCollection`, you would use the following command:

```
$ solrctl --solr http://mySolrServer:8983/solr core --create myCore \
-p dataDir=hdfs://namenode/solr/myCollection/index -p collection.configName=myConfig \
-p collection=myCollection -p numShards=2 -p shard=shard2
```

Converting instancedirs to configs

Cloudera Search supports converting existing deployments that use instancedirs to use configs.

To modify a collection to use configs

1. Make note of the existing instancedir name. This information is required later. For example, `myConfig`.
2. Delete the existing instancedir. For example:

```
solrctl instancedir --delete myConfig
```

3. Create a config using the same name as the instancedir you just deleted.

```
solrctl config --create myConfig baseConfig -p immutable=false
```

4. Reload the affected collection.

Spark Indexing Reference (CDH 5.2 and higher only)

Spark indexing uses the CrunchIndexerTool and requires a working MapReduce or Spark cluster, such as one installed using Cloudera Manager. Spark indexing is enabled when the CrunchIndexerTool is installed, as described in Installing the Spark Indexer.

CrunchIndexerTool is a Spark or MapReduce ETL batch job that pipes data from (splittable or non-splittable) HDFS files into Apache Solr, and runs the data through a morphline for extraction and transformation. The program is designed

for flexible, scalable, fault-tolerant batch ETL pipeline jobs. It is implemented as an Apache Crunch pipeline, allowing it to run on Apache Hadoop MapReduce or the Apache Spark execution engine.

Note: This command requires a morphline file, which must include a SOLR_LOCATOR. The snippet that includes the SOLR_LOCATOR might appear as follows:

```
SOLR_LOCATOR : {
  # Name of solr collection
  collection : collection

  # ZooKeeper ensemble
  zkHost : "$ZK_HOST"
}

morphlines : [
  {
    id : morphline1
    importCommands : ["org.kitesdk.**", "org.apache.solr.**"]
    commands : [
      { generateUUID { field : id } }

      { # Remove record fields that are unknown to Solr schema.xml.
        # Recall that Solr throws an exception on any attempt to load
a document that
        # contains a field that isn't specified in schema.xml.
        sanitizeUnknownSolrFields {
          solrLocator : ${SOLR_LOCATOR} # Location from which to fetch
 Solr schema
        }
      }

      { logDebug { format : "output record: {}", args : ["@{}"] } }

      {
        loadSolr {
          solrLocator : ${SOLR_LOCATOR}
        }
      }
    ]
  }
]
```

More details are available through command-line help. The `CrunchIndexerTool` jar does not contain all dependencies, unlike other Search indexing tools. Therefore, it is helpful to capture dependencies to variables that are used in invoking the help.

- To assign dependency information to variables and invoke help in a default parcels installation, use:

```
$ export myDriverJarDir=/opt/cloudera/parcels/CDH/lib/solr/contrib/crunch
$ export myDependencyJarDir=/opt/cloudera/parcels/CDH/lib/search/lib/search-crunch
$ export myDependencyJarPaths=$(find $myDependencyJarDir -name '*.jar' | sort | tr '\n'
 ':' | head -c -1)
$ export myDriverJar=$(find $myDriverJarDir -maxdepth 1 -name 'search-crunch.jar' !
-name '-job.jar' ! -name '*-sources.jar')
$ export HADOOP_CLASSPATH=$myDependencyJarPaths;
$ hadoop jar $myDriverJar org.apache.solr.crunch.CrunchIndexerTool -help
```

- To assign dependency information to variables and invoke help in a default packages installation, use:

```
$ export myDriverJarDir=/usr/lib/solr/contrib/crunch
$ export myDependencyJarDir=/usr/lib/search/lib/search-crunch
$ export myDependencyJarPaths=$(find $myDependencyJarDir -name '*.jar' | sort | tr '\n'
 ':' | head -c -1)
$ export myDriverJar=$(find $myDriverJarDir -maxdepth 1 -name 'search-crunch.jar' !
-name '-job.jar' ! -name '*-sources.jar')
```

```
$ export HADOOP_CLASSPATH=$myDependencyJarPaths;
$ hadoop jar $myDriverJar org.apache.solr.crunch.CrunchIndexerTool -help
```

```
MapReduceUsage: export HADOOP_CLASSPATH=$myDependencyJarPaths; hadoop jar $myDriverJar
org.apache.solr.crunch.CrunchIndexerTool --libjars $myDependencyJarFiles
[MapReduceGenericOptions]...
        [--input-file-list URI] [--input-file-format FQCN]
        [--input-file-projection-schema FILE]
        [--input-file-reader-schema FILE] --morphline-file FILE
        [--morphline-id STRING] [--pipeline-type STRING] [--xhelp]
        [--mappers INTEGER] [--dry-run] [--log4j FILE] [--chatty]
        [HDFS_URI [HDFS_URI ...]]

SparkUsage: spark-submit [SparkGenericOptions]... --master local|yarn --deploy-mode
client|cluster
--jars $myDependencyJarFiles --class org.apache.solr.crunch.CrunchIndexerTool $myDriverJar

        [--input-file-list URI] [--input-file-format FQCN]
        [--input-file-projection-schema FILE]
        [--input-file-reader-schema FILE] --morphline-file FILE
        [--morphline-id STRING] [--pipeline-type STRING] [--xhelp]
        [--mappers INTEGER] [--dry-run] [--log4j FILE] [--chatty]
        [HDFS_URI [HDFS_URI ...]]

Spark or MapReduce ETL batch job  that  pipes data from (splittable or non-
splittable) HDFS files into Apache Solr,  and  along  the way runs the data
through a Morphline  for  extraction  and  transformation.  The  program is
designed for  flexible,  scalable  and  fault-tolerant  batch  ETL pipeline
jobs. It is implemented as an  Apache  Crunch  pipeline and as such can run
on either the Apache Hadoop MapReduce or Apache Spark execution engine.

The program proceeds in several consecutive phases, as follows:

1) Randomization phase: This (parallel)  phase  randomizes the list of HDFS
input files in order to spread ingestion  load more evenly among the mapper
tasks of the  subsequent  phase.  This  phase  is  only  executed  for non-
splittables files, and skipped otherwise.

2) Extraction phase: This  (parallel)  phase  emits  a  series of HDFS file
input streams (for non-splittable files) or  a series of input data records
(for splittable files).

3) Morphline  phase:  This  (parallel)  phase  receives  the  items  of the
previous phase, and  uses  a  Morphline  to  extract  the relevant content,
transform  it  and  load  zero  or  more  documents  into  Solr.  The  ETL
functionality is  flexible  and  customizable  using  chains  of  arbitrary
morphline commands that pipe  records  from  one  transformation command to
another. Commands to parse and  transform  a  set  of standard data formats
such as Avro, Parquet,  CSV,  Text,  HTML,  XML,  PDF,  MS-Office, etc. are
provided out of the box,  and  additional  custom  commands and parsers for
additional file or data formats can  be added as custom morphline commands.
Any kind of data format can be processed  and any kind output format can be
generated by any custom Morphline ETL  logic.  Also, this phase can be used
to send data  directly  to  a  live  SolrCloud  cluster  (via  the loadSolr
morphline command).

The program  is  implemented  as  a  Crunch  pipeline  and  as  such Crunch
optimizes the logical phases  mentioned  above  into  an efficient physical
execution plan that runs a single  mapper-only job, or as the corresponding
Spark equivalent.

Fault Tolerance: Task attempts  are  retried  on  failure  per the standard
MapReduce or Spark semantics. If the  whole  job fails you can retry simply
by rerunning the program again using the same arguments.

Comparison with MapReduceIndexerTool:

1) CrunchIndexerTool can also run on  the  Spark execution engine, not just
on MapReduce.
2)  CrunchIndexerTool  enables  interactive  low  latency  prototyping,  in
particular in Spark 'local' mode.
3) CrunchIndexerTool supports updates  (and  deletes) of existing documents
```

```
in Solr, not just inserts.
4) CrunchIndexerTool can exploit data  locality for splittable Hadoop files
(text, avro, avroParquet).
We recommend  MapReduceIndexerTool  for  large  scale  batch  ingestion use
cases where updates (or  deletes)  of  existing  documents  in Solr are not
required, and we recommend CrunchIndexerTool for all other use cases.

CrunchIndexerOptions:
  HDFS_URI               HDFS URI of  file  or  directory  tree  to ingest.
                         (default: [])
  --input-file-list URI, --input-list URI
                         Local URI or  HDFS  URI  of  a  UTF-8 encoded file
                         containing a list of HDFS  URIs to ingest, one URI
                         per line in the  file.  If  '-' is specified, URIs
                         are read  from  the  standard  input.  Multiple --
                         input-file-list arguments can be specified.
  --input-file-format FQCN
                         The Hadoop FileInputFormat  to  use for extracting
                         data from splittable HDFS  files.  Can  be a fully
                         qualified Java  class  name  or  one  of  ['text',
                         'avro', 'avroParquet']. If this  option is present
                         the extraction phase will  emit  a series of input
                         data records rather  than  a  series  of HDFS file
                         input streams.
  --input-file-projection-schema FILE
                         Relative or absolute path  to  an Avro schema file
                         on the local file  system.  This  will  be used as
                         the projection schema for Parquet input files.
  --input-file-reader-schema FILE
                         Relative or absolute path  to  an Avro schema file
                         on the local file  system.  This  will  be used as
                         the  reader  schema  for  Avro  or  Parquet  input
                         files.      Example:      src/test/resources/test-
                         documents/strings.avsc
  --morphline-file FILE  Relative or absolute path  to  a local config file
                         that contains one  or  more  morphlines.  The file
                         must be UTF-8  encoded.  It  will  be  uploaded to
                         each remote task. Example: /path/to/morphline.conf
  --morphline-id STRING  The identifier  of  the  morphline  that  shall be
                         executed  within   the   morphline   config   file
                         specified by --morphline-file. If the --morphline-
                         id option is  omitted  the  first  (i.e. top-most)
                         morphline  within  the   config   file   is  used.
                         Example: morphline1
  --pipeline-type STRING
                         The engine to use  for  executing  the job. Can be
                         'mapreduce' or 'spark'. (default: mapreduce)
  --xhelp, --help, -help
                         Show this help message and exit
  --mappers INTEGER      Tuning knob that indicates  the  maximum number of
                         MR mapper tasks to use.  -1  indicates use all map
                         slots available  on  the  cluster.  This parameter
                         only  applies   to   non-splittable   input  files
                         (default: -1)
  --dry-run              Run the pipeline  but  print  documents  to stdout
                         instead of loading  them  into  Solr.  This can be
                         used for quicker turnaround  during  early trial &
                         debug sessions. (default: false)
  --log4j FILE           Relative or absolute  path  to  a log4j.properties
                         config file on the  local  file  system. This file
                         will be uploaded  to  each  remote  task. Example:
                         /path/to/log4j.properties
  --chatty               Turn on verbose output. (default: false)

SparkGenericOptions:     To print all options run 'spark-submit --help'

MapReduceGenericOptions: Generic options supported are
  --conf <configuration file>
                         specify an application configuration file
  -D <property=value>    use value for given property
  --fs <local|namenode:port>
                         specify a namenode
  --jt <local|resourcemanager:port>
```

```
                          specify a ResourceManager
  --files <comma separated list of files>
                          specify comma separated files to  be copied to the
                          map reduce cluster
  --libjars <comma separated list of jars>
                          specify comma separated  jar  files  to include in
                          the classpath.
  --archives <comma separated list of archives>
                          specify comma separated archives  to be unarchived
                          on the compute machines.

The general command line syntax is
bin/hadoop command [genericOptions] [commandOptions]


Examples:

# Prepare - Copy input files into HDFS:
export myResourcesDir=src/test/resources # for build from git
export myResourcesDir=/opt/cloudera/parcels/CDH/share/doc/search-*/search-crunch # for
 CDH with parcels
export myResourcesDir=/usr/share/doc/search-*/search-crunch # for CDH with packages
hadoop fs -copyFromLocal $myResourcesDir/test-documents/hello1.txt
hdfs:/user/systest/input/

# Prepare variables for convenient reuse:
export myDriverJarDir=target # for build from git
export myDriverJarDir=/opt/cloudera/parcels/CDH/lib/solr/contrib/crunch # for CDH with
 parcels
export myDriverJarDir=/usr/lib/solr/contrib/crunch # for CDH with packages
export myDependencyJarDir=target/lib # for build from git
export myDependencyJarDir=/opt/cloudera/parcels/CDH/lib/search/lib/search-crunch # for
 CDH with parcels
export myDependencyJarDir=/usr/lib/search/lib/search-crunch # for CDH with packages
export myDriverJar=$(find $myDriverJarDir -maxdepth 1 -name 'search-crunch-*.jar' !
-name '*-job.jar' ! -name '*-sources.jar')
export myDependencyJarFiles=$(find $myDependencyJarDir -name '*.jar' | sort | tr '\n'
',' | head -c -1)
export myDependencyJarPaths=$(find $myDependencyJarDir -name '*.jar' | sort | tr '\n'
':' | head -c -1)
export myJVMOptions="-DmaxConnectionsPerHost=10000 -DmaxConnections=10000
-Djava.io.tmpdir=/my/tmp/dir/" \
# connection settings for solrj, also custom tmp dir

# MapReduce on Yarn - Ingest text file line by line into Solr:
export HADOOP_CLIENT_OPTS="$myJVMOptions"; export \
HADOOP_CLASSPATH=$myDependencyJarPaths; hadoop \
  --config /etc/hadoop/conf.cloudera.YARN-1 \
  jar $myDriverJar org.apache.solr.crunch.CrunchIndexerTool \
  --libjars $myDependencyJarFiles \
  -D mapreduce.map.java.opts="-Xmx500m $myJVMOptions" \
  -D morphlineVariable.ZK_HOST=$(hostname):2181/solr \
  --files $myResourcesDir/test-documents/string.avsc \
  --morphline-file $myResourcesDir/test-morphlines/loadSolrLine.conf \
  --pipeline-type mapreduce \
  --chatty \
  --log4j $myResourcesDir/log4j.properties \
  /user/systest/input/hello1.txt

# Spark in Local Mode (for rapid prototyping) - Ingest into Solr:
spark-submit \
  --master local \
  --deploy-mode client \
  --jars $myDependencyJarFiles \
  --executor-memory 500M \
  --conf "spark.executor.extraJavaOptions=$myJVMOptions" \
  --driver-java-options "$myJVMOptions" \
  # --driver-library-path /opt/cloudera/parcels/CDH/lib/hadoop/lib/native \
  # for Snappy on CDH with parcels\
  # --driver-library-path /usr/lib/hadoop/lib/native \
  # for Snappy on CDH with packages \
  --class org.apache.solr.crunch.CrunchIndexerTool \
  $myDriverJar \
```

```
  -D morphlineVariable.ZK_HOST=$(hostname):2181/solr \
  --morphline-file $myResourcesDir/test-morphlines/loadSolrLine.conf \
  --pipeline-type spark \
  --chatty \
  --log4j $myResourcesDir/log4j.properties \
  /user/systest/input/hello1.txt

# Spark on Yarn in Client Mode (for testing) - Ingest into Solr:
Same as above, except replace '--master local' with '--master yarn'

# View the yarn executor log files (there is no GUI yet):
yarn logs --applicationId $application_XYZ

# Spark on Yarn in Cluster Mode (for production) - Ingest into Solr:
spark-submit \
  --master yarn \
  --deploy-mode cluster \
  --jars $myDependencyJarFiles \
  --executor-memory 500M \
  --conf "spark.executor.extraJavaOptions=$myJVMOptions" \
  --driver-java-options "$myJVMOptions" \
  --class org.apache.solr.crunch.CrunchIndexerTool \
  --files $(ls $myResourcesDir/log4j.properties),$(ls
$myResourcesDir/test-morphlines/loadSolrLine.conf)\
  $myDriverJar \
  -D hadoop.tmp.dir=/tmp \
  -D morphlineVariable.ZK_HOST=$(hostname):2181/solr \
  --morphline-file loadSolrLine.conf \
  --pipeline-type spark \
  --chatty \
  --log4j log4j.properties \
  /user/systest/input/hello1.txt

# Spark on Yarn in Cluster Mode (for production) - Ingest into Secure (Kerberos-enabled)
 Solr:
# Spark requires two additional steps compared to non-secure solr:
# (NOTE: MapReduce does not require extra steps for communicating with kerberos-enabled
 Solr)
# 1) Create a delegation token file
#    a) kinit as the user who will make solr requests
#    b) request a delegation token from solr and save it to a file:
#       e.g. using curl:
#       "curl --negotiate -u: http://solr-host:port/solr/?op=GETDELEGATIONTOKEN >
tokenFile.txt"
# 2) Pass the delegation token file to spark-submit:
#    a) add the delegation token file via --files
#    b) pass the file name via -D tokenFile
#       spark places this file in the cwd of the executor, so only list the file name,
 no path
spark-submit \
  --master yarn \
  --deploy-mode cluster \
  --jars $myDependencyJarFiles \
  --executor-memory 500M \
  --conf "spark.executor.extraJavaOptions=$myJVMOptions" \
  --driver-java-options "$myJVMOptions" \
  --class org.apache.solr.crunch.CrunchIndexerTool \
  --files $(ls $myResourcesDir/log4j.properties),$(ls
$myResourcesDir/test-morphlines/loadSolrLine.conf),tokenFile.txt\
  $myDriverJar \
  -D hadoop.tmp.dir=/tmp \
  -D morphlineVariable.ZK_HOST=$(hostname):2181/solr \
  -DtokenFile=tokenFile.txt \
  --morphline-file loadSolrLine.conf \
  --pipeline-type spark \
  --chatty \
  --log4j log4j.properties \
  /user/systest/input/hello1.txt
```

MapReduce Batch Indexing Reference

Cloudera Search provides the ability to batch index documents using MapReduce jobs.

If you did not install MapReduce tools required for Cloudera Search, do so on hosts where you want to submit a batch indexing job as described in Installing MapReduce Tools for use with Cloudera Search.

For information on tools related to batch indexing, see:

- MapReduceIndexerTool
- HDFSFindTool

Running an Example Indexing Job

See Cloudera Search Tutorial for examples of running a MapReduce job to index documents.

MapReduceIndexerTool

MapReduceIndexerTool is a MapReduce batch job driver that takes a morphline and creates a set of Solr index shards from a set of input files and writes the indexes into HDFS in a flexible, scalable, and fault-tolerant manner.

For more information on Morphlines, see:

- Extracting, Transforming, and Loading Data With Cloudera Morphlines on page 46 for an introduction to Morphlines.
- Example Morphline Usage on page 49 for morphline examples, discussion of those examples, and links to additional information.

MapReduceIndexerTool also supports merging the output shards into a set of live customer-facing Solr servers, typically a SolrCloud.

Important: Merging output shards into live customer-facing Solr servers can only be completed if all replicas are online.

The indexer creates an offline index on HDFS in the output directory specified by the `--output-dir` parameter. If the `--go-live` parameter is specified, Solr merges the resulting offline index into the live running service. Thus, the Solr service must have read access to the contents of the output directory to complete the `--go-live` step. In an environment with restrictive permissions, such as one with an HDFS umask of 077, the Solr user may not be able to read the contents of the newly created directory. To address this issue, the indexer automatically applies the HDFS ACLs to enable Solr to read the output directory contents. These ACLs are only applied if HDFS ACLs are enabled on the HDFS NameNode. For more information, see HDFS Extended ACLs.

The indexer only makes ACL updates to the output directory and its contents. If the output directory's parent directories do not include the run permission, the Solr service is not be able to access the output directory. Solr must have run permissions from standard permissions or ACLs on the parent directories of the output directory.

MapReduceIndexerTool Input Splits

Different from some other indexing tools, the MapReduceIndexerTool does not operate on HDFS blocks as input splits. This means that when indexing a smaller number of large files, fewer hosts may be involved. For example, indexing two files that are each one GB results in two hosts acting as mappers. If these files were stored on a system with a 128 MB block size, other mappers might divide the work on the two files among 16 mappers, corresponding to the 16 HDFS blocks that store the two files.

This intentional design choice aligns with MapReduceIndexerTool supporting indexing non-splittable file formats such as JSON, XML, jpg, or log4j.

In theory, this could result in inefficient use of resources when a single host indexes a large file while many other hosts sit idle. In reality, this indexing strategy typically results in satisfactory performance in production environments because in most cases the number of files is large enough that work is spread throughout the cluster.

While dividing tasks by input splits does not present problems in most cases, users may still want to divide indexing tasks along HDFS splits. In that case, use the CrunchIndexerTool, which can work with Hadoop input splits using the `input-file-format` option.

Invoking Command-Line Help

- To invoke the command-line help in a default parcels installation, use:

```
$ hadoop jar /opt/cloudera/parcels/CDH-*/jars/search-mr-*-job.jar \
org.apache.solr.hadoop.MapReduceIndexerTool --help
```

- To invoke the command-line help in a default packages installation, use:

```
$ hadoop jar /usr/lib/solr/contrib/mr/search-mr-*-job.jar \
org.apache.solr.hadoop.MapReduceIndexerTool --help
```

```
usage: hadoop [GenericOptions]... jar search-mr-*-job.jar
org.apache.solr.hadoop.MapReduceIndexerTool
       [--help] --output-dir HDFS_URI [--input-list URI]
       --morphline-file FILE [--morphline-id STRING] [--solr-home-dir DIR]
       [--update-conflict-resolver FQCN] [--mappers INTEGER]
       [--reducers INTEGER] [--max-segments INTEGER]
       [--fair-scheduler-pool STRING] [--dry-run] [--log4j FILE]
       [--verbose] [--show-non-solr-cloud] [--zk-host STRING] [--go-live]
       [--collection STRING] [--go-live-threads INTEGER]
       [HDFS_URI [HDFS_URI ...]]

MapReduce batch job driver that takes a morphline and creates a set of
Solr index shards from a set of input files and writes the indexes into
HDFS, in a flexible, scalable and fault-tolerant manner. It also supports
merging the output shards into a set of live customer facing Solr
servers, typically a SolrCloud. The program proceeds in several
consecutive MapReduce based phases, as follows:

1) Randomization phase: This (parallel) phase randomizes the list of
input files in order to spread indexing load more evenly among the
mappers of the subsequent phase.

2) Mapper phase: This (parallel) phase takes the input files, extracts
the relevant content, transforms it and hands SolrInputDocuments to a set
of reducers. The ETL functionality is flexible and customizable using
chains of arbitrary morphline commands that pipe records from one
transformation command to another. Commands to parse and transform a set
of standard data formats such as Avro, CSV, Text, HTML, XML, PDF, Word,
Excel, etc. are provided out of the box, and additional custom commands
and parsers for additional file or data formats can be added as morphline
plugins. This is done by implementing a simple Java interface that
consumes a record (e.g. a file in the form of an InputStream plus some
headers plus contextual metadata) and generates as output zero or more
records. Any kind of data format can be indexed and any Solr documents
for any kind of Solr schema can be generated, and any custom ETL logic
can be registered and executed.
Record fields, including MIME types, can also explicitly be passed by
force from the CLI to the morphline, for example: hadoop ... -D
morphlineField._attachment_mimetype=text/csv

3) Reducer phase: This (parallel) phase loads the mapper's
SolrInputDocuments into one EmbeddedSolrServer per reducer. Each such
reducer and Solr server can be seen as a (micro) shard. The Solr servers
store their data in HDFS.

4) Mapper-only merge phase: This (parallel) phase merges the set of
reducer shards into the number of solr shards expected by the user, using
a mapper-only job. This phase is omitted if the number of shards is
already equal to the number of shards expected by the user.

5) Go-live phase: This optional (parallel) phase merges the output shards
of the previous phase into a set of live customer facing Solr servers,
typically a SolrCloud. If this phase is omitted you can explicitly point
each Solr server to one of the HDFS output shard directories.
```

```
Fault Tolerance: Mapper and reducer task attempts are retried on failure
per the standard MapReduce semantics. On program startup all data in the
--output-dir is deleted if that output directory already exists. If the
whole job fails you can retry simply by rerunning the program again using
the same arguments.

positional arguments:
  HDFS_URI              HDFS URI of file or directory tree to index.
                        (default: [])

optional arguments:
  --help, -help, -h     Show this help message and exit
  --input-list URI      Local URI or HDFS URI of a UTF-8 encoded file
                        containing a list of HDFS URIs to index, one URI
                        per line in the file. If '-' is specified, URIs
                        are read from the standard input. Multiple --
                        input-list arguments can be specified.
  --morphline-id STRING The identifier of the morphline that shall be
                        executed within the morphline config file
                        specified by --morphline-file. If the --
                        morphline-id option is ommitted the first (i.e.
                        top-most) morphline within the config file is
                        used. Example: morphline1
  --solr-home-dir DIR   Optional relative or absolute path to a local
                        dir containing Solr conf/ dir and in particular
                        conf/solrconfig.xml and optionally also lib/
                        dir. This directory will be uploaded to each MR
                        task. Example: src/test/resources/solr/minimr
  --update-conflict-resolver FQCN
                        Fully qualified class name of a Java class that
                        implements the UpdateConflictResolver interface.
                        This enables deduplication and ordering of a
                        series of document updates for the same unique
                        document key. For example, a MapReduce batch job
                        might index multiple files in the same job where
                        some of the files contain old and new versions
                        of the very same document, using the same unique
                        document key.
                        Typically, implementations of this interface
                        forbid collisions by throwing an exception, or
                        ignore all but the most recent document version,
                        or, in the general case, order colliding updates
                        ascending from least recent to most recent
                        (partial) update. The caller of this interface
                        (i.e. the Hadoop Reducer) will then apply the
                        updates to Solr in the order returned by the
                        orderUpdates() method.
                        The                                  default
                        RetainMostRecentUpdateConflictResolver
                        implementation ignores all but the most recent
                        document version, based on a configurable
                        numeric Solr field, which defaults to the
                        file_last_modified timestamp (default: org.
                        apache.solr.hadoop.dedup.
                        RetainMostRecentUpdateConflictResolver)
  --mappers INTEGER     Tuning knob that indicates the maximum number of
                        MR mapper tasks to use. -1 indicates use all map
                        slots available on the cluster. (default: -1)
  --reducers INTEGER    Tuning knob that indicates the number of
                        reducers to index into. -1 indicates use all
                        reduce slots available on the cluster. 0
                        indicates use one reducer per output shard,
                        which disables the mtree merge MR algorithm. The
                        mtree merge MR algorithm improves scalability by
                        spreading load (in particular CPU load) among a
                        number of parallel reducers that can be much
                        larger than the number of solr shards expected
                        by the user. It can be seen as an extension of
                        concurrent lucene merges and tiered lucene
                        merges to the clustered case. The subsequent
                        mapper-only phase merges the output of said
                        large number of reducers to the number of shards
```

```
                        expected by the user, again by utilizing more
                        available parallelism on the cluster. (default:
                        -1)
  --max-segments INTEGER
                        Tuning knob that indicates the maximum number of
                        segments to be contained on output in the index
                        of each reducer shard. After a reducer has built
                        its output index it applies a merge policy to
                        merge segments until there are <= maxSegments
                        lucene segments left in this index. Merging
                        segments involves reading and rewriting all data
                        in all these segment files, potentially multiple
                        times, which is very I/O intensive and time
                        consuming. However, an index with fewer segments
                        can later be merged faster, and it can later be
                        queried faster once deployed to a live Solr
                        serving shard. Set maxSegments to 1 to optimize
                        the index for low query latency. In a nutshell,
                        a small maxSegments value trades indexing
                        latency for subsequently improved query latency.
                        This can be a reasonable trade-off for batch
                        indexing systems. (default: 1)
  --fair-scheduler-pool STRING
                        Optional tuning knob that indicates the name of
                        the fair scheduler pool to submit jobs to. The
                        Fair Scheduler is a pluggable MapReduce
                        scheduler that provides a way to share large
                        clusters. Fair scheduling is a method of
                        assigning resources to jobs such that all jobs
                        get, on average, an equal share of resources
                        over time. When there is a single job running,
                        that job uses the entire cluster. When other
                        jobs are submitted, tasks slots that free up are
                        assigned to the new jobs, so that each job gets
                        roughly the same amount of CPU time. Unlike the
                        default Hadoop scheduler, which forms a queue of
                        jobs, this lets short jobs finish in reasonable
                        time while not starving long jobs. It is also an
                        easy way to share a cluster between multiple of
                        users. Fair sharing can also work with job
                        priorities - the priorities are used as weights
                        to determine the fraction of total compute time
                        that each job gets.
  --dry-run             Run in local mode and print documents to stdout
                        instead of loading them into Solr. This executes
                        the morphline in the client process (without
                        submitting a job to MR) for quicker turnaround
                        during early trial & debug sessions. (default:
                        false)
  --log4j FILE          Relative or absolute path to a log4j.properties
                        config file on the local file system. This file
                        will be uploaded to each MR task. Example:
                        /path/to/log4j.properties
  --verbose, -v         Turn on verbose output. (default: false)
  --show-non-solr-cloud Also show options for Non-SolrCloud mode as part
                        of --help. (default: false)

Required arguments:
  --output-dir HDFS_URI HDFS directory to write Solr indexes to. Inside
                        there one output directory per shard will be
                        generated. Example: hdfs://c2202.mycompany.
                        com/user/$USER/test
  --morphline-file FILE Relative or absolute path to a local config file
                        that contains one or more morphlines. The file
                        must be UTF-8 encoded. Example:
                        /path/to/morphline.conf

Cluster arguments:
  Arguments that provide information about your Solr cluster.

  --zk-host STRING      The address of a ZooKeeper ensemble being used
                        by a SolrCloud cluster. This ZooKeeper ensemble
                        will be examined to determine the number of
```

```
                          output shards to create as  well as the Solr URLs
                          to merge the output shards into when using the --
                          go-live option. Requires that  you  also pass the
                          --collection to merge the shards into.

                          The  --zk-host   option   implements   the   same
                          partitioning semantics as  the standard SolrCloud
                          Near-Real-Time (NRT)  API.  This  enables  to mix
                          batch  updates  from   MapReduce  ingestion  with
                          updates from standard Solr  NRT  ingestion on the
                          same SolrCloud  cluster,  using  identical unique
                          document keys.

                          Format is: a  list  of  comma separated host:port
                          pairs,  each  corresponding   to   a  zk  server.
                          Example:              '127.0.0.1:2181,127.0.0.1:
                          2182,127.0.0.1:2183'  If   the   optional  chroot
                          suffix is  used  the  example  would  look  like:
                          '127.0.0.1:2181/solr,127.0.0.1:2182/solr,
                          127.0.0.1:2183/solr' where  the  client  would be
                          rooted  at  '/solr'  and   all   paths  would  be
                          relative     to     this      root      -     i.e.
                          getting/setting/etc...  '/foo/bar'  would  result
                          in operations being run  on '/solr/foo/bar' (from
                          the server perspective).

                          If --solr-home-dir  is  not  specified,  the Solr
                          home  directory  for   the   collection  will  be
                          downloaded from this ZooKeeper ensemble.

Go live arguments:
  Arguments for merging  the  shards  that  are  built  into  a  live Solr
  cluster. Also see the Cluster arguments.

  --go-live               Allows you to  optionally  merge  the final index
                          shards into a live  Solr  cluster  after they are
                          built. You can pass the ZooKeeper address with --
                          zk-host  and  the  relevant  cluster  information
                          will be auto detected.  (default: false)
  --collection STRING     The SolrCloud  collection  to  merge  shards into
                          when  using  --go-live  and  --zk-host.  Example:
                          collection1
  --go-live-threads INTEGER
                          Tuning knob that indicates  the maximum number of
                          live merges  to  run  in  parallel  at  one time.
                          (default: 1000)

Generic options supported are
  --conf <configuration file>
                          specify an application configuration file
  -D <property=value>     use value for given property
  --fs <local|namenode:port>
                          specify a namenode
  --jt <local|jobtracker:port>
                          specify a job tracker
  --files <comma separated list of files>
                          specify comma separated  files  to  be  copied to
                          the map reduce cluster
  --libjars <comma separated list of jars>
                          specify comma separated jar  files  to include in
                          the classpath.
  --archives <comma separated list of archives>
                          specify   comma   separated    archives   to   be
                          unarchived on the compute machines.

The general command line syntax is
bin/hadoop command [genericOptions] [commandOptions]

Examples:

# (Re)index an Avro based Twitter tweet file:
sudo -u hdfs hadoop \
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
```

```
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file
../search-core/src/test/resources/test-morphlines/tutorialReadAvroContainer.conf \
  --solr-home-dir src/test/resources/solr/minimr \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --shards 1 \
  hdfs:///user/$USER/test-documents/sample-statuses-20120906-141433.avro

# (Re)index all files that match all of the following conditions:
# 1) File is contained in dir tree hdfs:///user/$USER/solrloadtest/twitter/tweets
# 2) file name matches the glob pattern 'sample-statuses*.gz'
# 3) file was last modified less than 100000 minutes ago
# 4) file size is between 1 MB and 1 GB
# Also include extra library jar file containing JSON tweet Java parser:
hadoop jar target/search-mr-*-job.jar org.apache.solr.hadoop.HdfsFindTool \
  -find hdfs:///user/$USER/solrloadtest/twitter/tweets \
  -type f \
  -name 'sample-statuses*.gz' \
  -mmin -1000000 \
  -size -100000000c \
  -size +1000000c \
| sudo -u hdfs hadoop \
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool \
  --libjars /path/to/kite-morphlines-twitter-0.10.0.jar \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file
../search-core/src/test/resources/test-morphlines/tutorialReadJsonTestTweets.conf \
  --solr-home-dir src/test/resources/solr/minimr \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --shards 100 \
  --input-list -

# Go live by merging resulting index shards into a live Solr cluster
# (explicitly specify Solr URLs - for a SolrCloud cluster see next example):
sudo -u hdfs hadoop \
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file
../search-core/src/test/resources/test-morphlines/tutorialReadAvroContainer.conf \
  --solr-home-dir src/test/resources/solr/minimr \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --shard-url http://solr001.mycompany.com:8983/solr/collection1 \
  --shard-url http://solr002.mycompany.com:8983/solr/collection1 \
  --go-live \
  hdfs:///user/foo/indir

# Go live by merging resulting index shards into a live SolrCloud cluster
# (discover shards and Solr URLs through ZooKeeper):
sudo -u hdfs hadoop \
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file
../search-core/src/test/resources/test-morphlines/tutorialReadAvroContainer.conf \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --zk-host zk01.mycompany.com:2181/solr \
  --collection collection1 \
  --go-live \
  hdfs:///user/foo/indir

# MapReduce on Yarn - Pass custom JVM arguments (including a custom tmp directory)
HADOOP_CLIENT_OPTS='-DmaxConnectionsPerHost=10000 -DmaxConnections=10000
-Djava.io.tmpdir=/my/tmp/dir/'; \
sudo -u hdfs hadoop \
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool \
```

```
  -D 'mapreduce.map.java.opts=-DmaxConnectionsPerHost=10000 -DmaxConnections=10000' \
  -D 'mapreduce.reduce.java.opts=-DmaxConnectionsPerHost=10000 -DmaxConnections=10000'
 \
  --log4j src/test/resources/log4j.properties \
  --morphline-file
../search-core/src/test/resources/test-morphlines/tutorialReadAvroContainer.conf \
  --solr-home-dir src/test/resources/solr/minimr \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --shards 1 \
  hdfs:///user/$USER/test-documents/sample-statuses-20120906-141433.avro

# MapReduce on MR1 - Pass custom JVM arguments (including a custom tmp directory)
HADOOP_CLIENT_OPTS='-DmaxConnectionsPerHost=10000 -DmaxConnections=10000
-Djava.io.tmpdir=/my/tmp/dir/'; \
sudo -u hdfs hadoop \
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool \
  -D 'mapred.child.java.opts=-DmaxConnectionsPerHost=10000 -DmaxConnections=10000' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file
../search-core/src/test/resources/test-morphlines/tutorialReadAvroContainer.conf \
  --solr-home-dir src/test/resources/solr/minimr \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --shards 1 \
  hdfs:///user/$USER/test-documents/sample-statuses-20120906-141433.avro
```

MapReduceIndexerTool Metadata

The MapReduceIndexerTool generates metadata fields for each input file when indexing. These fields can be used in morphline commands. These fields can also be stored in Solr, by adding definitions like the following to your Solr `schema.xml` file. After the MapReduce indexing process completes, the fields are searchable through Solr.

```
<!-- file metadata -->
<field name="file_download_url" type="string" indexed="false" stored="true" />
<field name="file_upload_url" type="string" indexed="false" stored="true" />
<field name="file_scheme" type="string" indexed="true" stored="true" />
<field name="file_host" type="string" indexed="true" stored="true" />
<field name="file_port" type="int" indexed="true" stored="true" />
<field name="file_path" type="string" indexed="true" stored="true" />
<field name="file_name" type="string" indexed="true" stored="true" />
<field name="file_length" type="tlong" indexed="true" stored="true" />
<field name="file_last_modified" type="tlong" indexed="true" stored="true" />
<field name="file_owner" type="string" indexed="true" stored="true" />
<field name="file_group" type="string" indexed="true" stored="true" />
<field name="file_permissions_user" type="string" indexed="true" stored="true" />
<field name="file_permissions_group" type="string" indexed="true" stored="true" />
<field name="file_permissions_other" type="string" indexed="true" stored="true" />
<field name="file_permissions_stickybit" type="boolean" indexed="true" stored="true" />
```

Example output:

```
"file_upload_url":"foo/test-documents/sample-statuses-20120906-141433.avro",
"file_download_url":"hdfs://host1.mycompany.com:8020/user/foo/
test-documents/sample-statuses-20120906-141433.avro",
"file_scheme":"hdfs",
"file_host":"host1.mycompany.com",
"file_port":8020,
"file_name":"sample-statuses-20120906-141433.avro",
"file_path":"/user/foo/test-documents/sample-statuses-20120906-141433.avro",
"file_last_modified":1357193447106,
"file_length":1512,
"file_owner":"foo",
"file_group":"foo",
"file_permissions_user":"rw-",
"file_permissions_group":"r--",
"file_permissions_other":"r--",
"file_permissions_stickybit":false,
```

HdfsFindTool

HdfsFindTool is essentially the HDFS version of the Linux filesystem `find` command. The command walks one or more HDFS directory trees, finds all HDFS files that match the specified expression, and applies selected actions to them. By default, it prints the list of matching HDFS file paths to `stdout`, one path per line. The output file list can be piped into the MapReduceIndexerTool using the MapReduceIndexerTool `--inputlist` option.

Note: The Hadoop file system shell (hadoop fs) also includes find functionality for searching HDFS. For more information on the Hadoop find command, use `hadoop fs -help find`.

More details are available through command-line help. The command used to invoke the help varies by installation type and may vary further in custom installations.

- To invoke the command-line help in a default parcels installation, use:

```
$ hadoop jar /opt/cloudera/parcels/CDH-*/jars/search-mr-*-job.jar \
org.apache.solr.hadoop.HdfsFindTool -help
```

- To invoke the command-line help in a default packages installation, use:

```
$ hadoop jar /usr/lib/solr/contrib/mr/search-mr-job.jar \
 org.apache.solr.hadoop.HdfsFindTool -help
```

More details are available through the command line help:

```
Usage: hadoop fs [generic options]
 [-find <path> ... <expression> ...]
 [-help [cmd ...]]
 [-usage [cmd ...]]

-find <path> ... <expression> ...: Finds all files that match the specified expression
 and applies selected actions to them.

  The following primary expressions are recognised:
    -atime n
    -amin n
      Evaluates as true if the file access time subtracted from
      the start time is n days (or minutes if -amin is used).

    -blocks n
      Evaluates to true if the number of file blocks is n.

    -class classname [args ...]
      Executes the named expression class.

    -depth
      Always evaluates to true. Causes directory contents to be
      evaluated before the directory itself.

    -empty
      Evaluates as true if the file is empty or directory has no
      contents.

    -group groupname
      Evaluates as true if the file belongs to the specified
      group.

    -mtime n
    -mmin n
      Evaluates as true if the file modification time subtracted
      from the start time is n days (or minutes if -mmin is used)

    -name pattern
    -iname pattern
      Evaluates as true if the basename of the file matches the
      pattern using standard file system globbing.
      If -iname is used then the match is case insensitive.
```

```
    -newer file
      Evaluates as true if the modification time of the current
      file is more recent than the modification time of the
      specified file.

    -nogroup
      Evaluates as true if the file does not have a valid group.

    -nouser
      Evaluates as true if the file does not have a valid owner.

    -perm [-]mode
    -perm [-]onum
      Evaluates as true if the file permissions match that
      specified. If the hyphen is specified then the expression
      shall evaluate as true if at least the bits specified
      match, otherwise an exact match is required.
      The mode may be specified using either symbolic notation,
      eg 'u=rwx,g+x+w' or as an octal number.

    -print
    -print0
      Always evaluates to true. Causes the current pathname to be
      written to standard output. If the -print0 expression is
      used then an ASCII NULL character is appended.

    -prune
      Always evaluates to true. Causes the find command to not
      descend any further down this directory tree. Does not
      have any affect if the -depth expression is specified.

    -replicas n
      Evaluates to true if the number of file replicas is n.

    -size n[c]
      Evaluates to true if the file size in 512 byte blocks is n.
      If n is followed by the character 'c' then the size is in bytes.

    -type filetype
      Evaluates to true if the file type matches that specified.
      The following file type values are supported:
      'd' (directory), 'l' (symbolic link), 'f' (regular file).

    -user username
      Evaluates as true if the owner of the file matches the
      specified user.

  The following operators are recognised:
    expression -a expression
    expression -and expression
    expression expression
      Logical AND operator for joining two expressions. Returns
      true if both child expressions return true. Implied by the
      juxtaposition of two expressions and so does not need to be
      explicitly specified. The second expression will not be
      applied if the first fails.

    ! expression
    -not expression
      Evaluates as true if the expression evaluates as false and
      vice-versa.

    expression -o expression
    expression -or expression
      Logical OR operator for joining two expressions. Returns
      true if one of the child expressions returns true. The
      second expression will not be applied if the first returns
      true.

-help [cmd ...]: Displays help for given command or all commands if none
  is specified.
```

```
-usage [cmd ...]: Displays the usage for given command or all commands if none
  is specified.

Generic options supported are
-conf <configuration file>     specify an application configuration file
-D <property=value>            use value for given property
-fs <local|namenode:port>      specify a namenode
-jt <local|jobtracker:port>    specify a job tracker
-files <comma separated list of files>    specify comma separated files to be copied to
 the map reduce cluster
-libjars <comma separated list of jars>    specify comma separated jar files to include
 in the classpath.
-archives <comma separated list of archives>    specify comma separated archives to be
 unarchived on the compute machines.

The general command line syntax is
bin/hadoop command [genericOptions] [commandOptions]
```

For example, to find all files that:

- Are contained in the directory tree `hdfs:///user/$USER/solrloadtest/twitter/tweets`
- Have a name matching the glob pattern `sample-statuses*.gz`
- Were modified less than 60 minutes ago
- Are between 1 MB and 1 GB

You could use the following:

```
$ hadoop jar /usr/lib/solr/contrib/mr/search-mr-*-job.jar \
org.apache.solr.hadoop.HdfsFindTool -find \
hdfs:///user/$USER/solrloadtest/twitter/tweets -type f -name \
'sample-statuses*.gz' -mmin -60 -size -1000000000c -size +1000000c
```

Flume Near Real-Time Indexing Reference

The Flume Solr Sink is a flexible, scalable, fault tolerant, transactional, near real-time (NRT) system for processing a continuous stream of records into live search indexes. Latency from the time of data arrival to the time data appears in search query results is measured in seconds and is tunable.

Data flows from sources through Flume hosts across the network to Flume Solr sinks. The sinks extract the relevant data, transform it, and load it into a set of live Solr search servers, which in turn serve queries to end users or search applications.

The ETL functionality is flexible and customizable, using chains of morphline commands that pipe records from one transformation command to another. Commands to parse and transform a set of standard data formats such as Avro, CSV, text, HTML, XML, PDF, Word, or Excel, are provided out of the box. You can add additional custom commands and parsers as morphline plug-ins for other file or data formats. Do this by implementing a simple Java interface that consumes a record such as a file in the form of an InputStream plus some headers and contextual metadata. The record consumed by the Java interface is used to generate record output. Any kind of data format can be indexed, any Solr documents for any kind of Solr schema can be generated, and any custom ETL logic can be registered and run.

Routing to multiple Solr collections improves multi-tenancy, and routing to a SolrCloud cluster improves scalability. Flume SolrSink servers can be co-located with live Solr servers serving end user queries, or deployed on separate industry-standard hardware to improve scalability and reliability. Indexing load can be spread across a large number of Flume SolrSink servers, and Flume features such as Load balancing Sink Processor can help improve scalability and achieve high availability. .

Flume indexing provides low-latency data acquisition and querying. It complements (instead of replaces) use cases based on batch analysis of HDFS data using MapReduce. In many use cases, data flows simultaneously from the producer through Flume into both Solr and HDFS using features such as the Replicating Channel Selector to replicate an incoming flow into two output flows. You can use near real-time ingestion as well as batch analysis tools.

For a more comprehensive discussion of the Flume Architecture, see Large Scale Data Ingestion using Flume.

After configuring Flume, start it as detailed in Flume Installation.

See the Cloudera Search Tutorial for exercises that show how to configure and run a Flume SolrSink to index documents.

Flume Morphline Solr Sink Configuration Options

You can use the standard configuration file `flume.conf` to configure Flume agents, including their sources, sinks, and channels. For more information about `flume.conf`, see the Flume User Guide.

Flume Morphline SolrSink provides the following configuration options in the `flume.conf` file:

Property Name	Default	Description
`type`		The FQCN of this class: `org.apache.flume.sink.solr.morphline.MorphlineSolrSink`
`batchSize`	100	The maximum number of events to take per flume transaction.
`batchDurationMillis`	1000	The maximum duration per Flume transaction (ms). The transaction commits after this duration or when batchSize is exceeded, whichever comes first.
`indexerClass`	`org.apache.flume.sink.solr.morphline.MorphlineSolrIndexer`	The FQCN of a class implementing `org.apache.flume.sink.solr.morphline.SolrIndexer`

`morphlineFile`	n/a	The location of the morphline configuration file.
		• In a Cloudera Manager deployment, use:
		`agent.sinks.solrSink.morphlineFile=morphlines.conf`
		• In unmanaged deployments, provide the relative or absolute path on the local filesystem to the morphline configuration file. For example:
		`/etc/flume-ng/conf/tutorialReadAvroContainer.conf`

`morphlineId`	null	Name used to identify a morphline if there are multiple morphlines in a morphline configuration file.

This example shows a `flume.conf` section for a SolrSink for the agent named `agent`:

```
agent.sinks.solrSink.type = org.apache.flume.sink.solr.morphline.MorphlineSolrSink
agent.sinks.solrSink.channel = memoryChannel
agent.sinks.solrSink.batchSize = 100
agent.sinks.solrSink.batchDurationMillis = 1000
agent.sinks.solrSink.morphlineFile = /etc/flume-ng/conf/morphline.conf
agent.sinks.solrSink.morphlineId = morphline1
```

Note: The examples in this document use a Flume MemoryChannel to easily get started. For production use it is often more appropriate to configure a Flume FileChannel instead, which is a high performance transactional persistent queue.

Flume Morphline Interceptor Configuration Options

Flume can modify and drop events in-flight with the help of Interceptors, which can be attached to any Flume source. Flume MorphlineInterceptor runs the transformations of a morphline on intercepted events. For example the morphline can ignore events or alter or insert certain event headers using regular expression-based pattern matching, or it can auto-detect and set a MIME type using Apache Tika on events that are intercepted. This packet sniffing can be used for content-based routing in a Flume topology.

Flume supports multiplexing the event flow to destinations by defining a flow multiplexer that can replicate or selectively route an event to channels. This example shows a source from agent “foo” fanning out the flow to three different channels. This fan out can be replicating or multiplexing. In replicating, each event is sent to all three channels. In multiplexing, an event is delivered to a subset of available channels when that event's attribute matches a preconfigured value. For example, if an event attribute called `stream.type` is set to `application/pdf`, it goes to `channel1` and `channel3`. If the attribute is set to `avro/binary`, it goes to `channel2`. If that channel is unavailable then an exception is thrown and the event is replayed later when the channel becomes available again. You can set the mapping in the `flume.conf` file.

Flume MorphlineInterceptor provides the following configuration options in the `flume.conf` file:

Property Name	Default	Description
`type`		The FQCN of this class: `org.apache.flume.sink.solr.morphline.MorphlineInterceptor$Builder`
`morphlineFile`	n/a	The location of the morphline configuration file. • In a Cloudera Manager deployment, use: `agent.sources.avroSrc.interceptors.morphlineinterceptor.morphlineFile = morphlines.conf` • In unmanaged deployments, provide the relative or absolute path on the local filesystem to the morphline configuration file. For example, `/etc/flume-ng/conf/morphline.conf`.
`morphlineId`	null	The name used to identify a morphline if a config file has multiple morphlines.

This example shows a `flume.conf` section for a MorphlineInterceptor for the agent named "agent":

```
agent.sources.avroSrc.interceptors = morphlineinterceptor
agent.sources.avroSrc.interceptors.morphlineinterceptor.type =
org.apache.flume.sink.solr.morphline.MorphlineInterceptor$Builder
agent.sources.avroSrc.interceptors.morphlineinterceptor.morphlineFile =
/etc/flume-ng/conf/morphline.conf
agent.sources.avroSrc.interceptors.morphlineinterceptor.morphlineId = morphline1
```

Note: A morphline interceptor cannot generate more than one output record for each input event.

Flume Solr UUIDInterceptor Configuration Options

Flume can modify or drop events in-flight with the help of Interceptors, which can be attached to any Flume Source. Flume Solr UUIDInterceptor sets a universally unique identifier on all intercepted events. For example, UUID `b5755073-77a9-43c1-8fad-b7a586fc1b97` represents a 128-bit value.

You can use UUIDInterceptor to automatically assign a UUID to a document event if no application-level unique key for the event is available. Assign UUIDs to events as soon as they enter the Flume network—that is, in the first Flume source of the flow. This enables deduplicating documents that may be accidentally duplicated as a result of replication and redelivery in a Flume network that is designed for high availability and high performance. If available, an application-level key is preferable to an auto-generated UUID because it enables subsequent updates and deletion of the document in Solr using that key.

Flume Solr UUIDInterceptor provides the following configuration options in the `flume.conf` file:

Property Name	Default	Description
`type`		The FQCN of this class: `org.apache.flume.sink.solr.morphline.UUIDInterceptor$Builder`
`headerName`	id	The name of the Flume header to modify.
`preserveExisting`	true	If the UUID header already exists, determine whether it is preserved.
`prefix`	""	The prefix string constant to prepend to each generated UUID.

For examples, see the BlobHandler and BlobDeserializer.

Flume Solr BlobHandler Configuration Options

Flume accepts Flume events by HTTP POST and GET with the help of HTTPSource.

By default, HTTPSource splits JSON input into Flume events. As an alternative, Flume Solr BlobHandler for HTTPSource returns an event that contains the request parameters as well as the Binary Large Object (BLOB) uploaded with this request. This approach is not suitable for very large objects because it buffers the entire BLOB.

Flume Solr BlobHandler provides the following configuration options in the `flume.conf` file:

Property Name	Default	Description
`handler`		The FQCN of this class: `org.apache.flume.sink.solr.morphline.BlobHandler`
`handler.maxBlobLength`	100000000 (100 MB)	The maximum number of bytes to read and buffer for a request.

This example shows a `flume.conf` section for a HTTPSource with a BlobHandler for the agent named `agent`:

```
agent.sources.httpSrc.type = org.apache.flume.source.http.HTTPSource
agent.sources.httpSrc.port = 5140
agent.sources.httpSrc.handler = org.apache.flume.sink.solr.morphline.BlobHandler
agent.sources.httpSrc.handler.maxBlobLength = 2000000000
agent.sources.httpSrc.interceptors = uuidinterceptor
agent.sources.httpSrc.interceptors.uuidinterceptor.type =
org.apache.flume.sink.solr.morphline.UUIDInterceptor$Builder
agent.sources.httpSrc.interceptors.uuidinterceptor.headerName = id
#agent.sources.httpSrc.interceptors.uuidinterceptor.preserveExisting = false
#agent.sources.httpSrc.interceptors.uuidinterceptor.prefix = myhostname
agent.sources.httpSrc.channels = memoryChannel
```

Flume Solr BlobDeserializer Configuration Options

Using SpoolDirectorySource, Flume can ingest data from files located in a spooling directory on disk. Unlike other asynchronous sources, `SpoolDirectorySource` does not lose data even if Flume is restarted or fails. Flume watches the directory for new files and ingests them as they are detected.

By default, `SpoolDirectorySource` splits text input on newlines into Flume events. You can change this behavior by having Flume Solr `BlobDeserializer` read Binary Large Objects (BLOBs) from `SpoolDirectorySource`. This alternative approach is not suitable for very large objects because the entire BLOB is buffered.

Flume Solr `BlobDeserializer` provides the following configuration options in the `flume.conf` file:

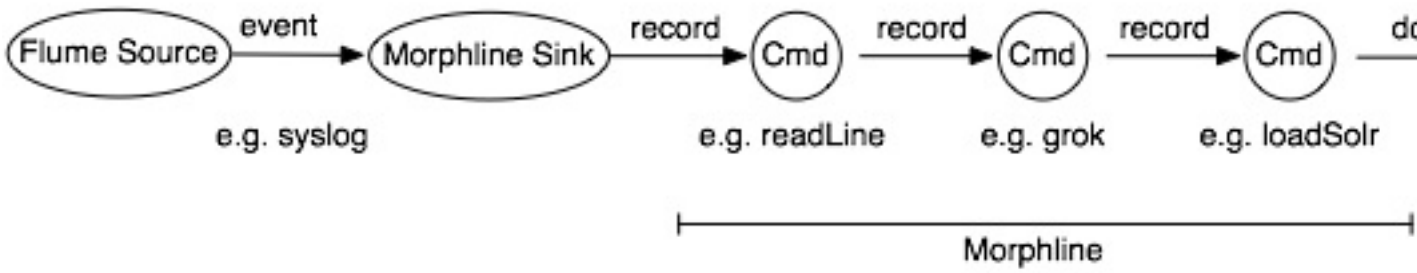
Property Name	Default	Description
`deserializer`		The FQCN of this class: `org.apache.flume.sink.solr.morphline.BlobDeserializer$Builder`
`deserializer.maxBlobLength`	100000000 (100 MB)	The maximum number of bytes to read and buffer for a given request.

This example shows a `flume.conf` section for a `SpoolDirectorySource` with a `BlobDeserializer` for the agent named `agent`:

```
agent.sources.spoolSrc.type = spooldir
agent.sources.spoolSrc.spoolDir = /tmp/myspooldir
agent.sources.spoolSrc.ignorePattern = \.
agent.sources.spoolSrc.deserializer =
org.apache.flume.sink.solr.morphline.BlobDeserializer$Builder
agent.sources.spoolSrc.deserializer.maxBlobLength = 2000000000
agent.sources.spoolSrc.batchSize = 1
agent.sources.spoolSrc.fileHeader = true
agent.sources.spoolSrc.fileHeaderKey = resourceName
agent.sources.spoolSrc.interceptors = uuidinterceptor
agent.sources.spoolSrc.interceptors.uuidinterceptor.type =
org.apache.flume.sink.solr.morphline.UUIDInterceptor$Builder
agent.sources.spoolSrc.interceptors.uuidinterceptor.headerName = id
#agent.sources.spoolSrc.interceptors.uuidinterceptor.preserveExisting = false
#agent.sources.spoolSrc.interceptors.uuidinterceptor.prefix = myhostname
agent.sources.spoolSrc.channels = memoryChannel
```

Extracting, Transforming, and Loading Data With Cloudera Morphlines

Cloudera Morphlines is an open-source framework that reduces the time and skills required to build or change Search indexing applications. A morphline is a rich configuration file that simplifies defining an ETL transformation chain. Use these chains to consume any kind of data from any data source, process the data, and load the results into Cloudera Search. Executing in a small, embeddable Java runtime system, morphlines can be used for near real-time applications as well as batch processing applications. The following diagram shows the process flow:



Morphlines can be seen as an evolution of Unix pipelines, where the data model is generalized to work with streams of generic records, including arbitrary binary payloads. Morphlines can be embedded into Hadoop components such as Search, Flume, MapReduce, Pig, Hive, and Sqoop.

The framework ships with a set of frequently used high-level transformation and I/O commands that can be combined in application-specific ways. The plug-in system allows you to add new transformations and I/O commands and integrates existing functionality and third-party systems.

This integration enables the following:

- Rapid Hadoop ETL application prototyping
- Complex stream and event processing in real time
- Flexible log file analysis
- Integration of multiple heterogeneous input schemas and file formats
- Reuse of ETL logic building blocks across Search applications

The high-performance Cloudera runtime compiles a morphline, processing all commands for a morphline in the same thread and adding no artificial overhead. For high scalability, you can deploy many morphline instances on a cluster in many Flume agents and MapReduce tasks.

The following components execute morphlines:

- MapReduceIndexerTool
- Flume Morphline Solr Sink and Flume MorphlineInterceptor

Cloudera also provides a corresponding Cloudera Search Tutorial.

Data Morphlines Support

Morphlines manipulate continuous or arbitrarily large streams of records. The data model can be described as follows: A record is a set of named fields where each field has an ordered list of one or more values. A value can be any Java Object. That is, a record is essentially a hash table where each hash table entry contains a String key and a list of Java Objects as values. (The implementation uses Guava's `ArrayListMultimap`, which is a `ListMultimap`). Note that a field can have multiple values and any two records need not use common field names. This flexible data model corresponds exactly to the characteristics of the Solr/Lucene data model, meaning a record can be seen as a `SolrInputDocument`. A field with zero values is removed from the record - fields with zero values effectively do not exist.

Not only structured data, but also arbitrary binary data can be passed into and processed by a morphline. By convention, a record can contain an optional field named `_attachment_body`, which can be a Java `java.io.InputStream` or Java `byte[]`. Optionally, such binary input data can be characterized in more detail by setting the fields named `_attachment_mimetype` (such as `application/pdf`) and `_attachment_charset` (such as `UTF-8`) and `_attachment_name` (such as `cars.pdf`), which assists in detecting and parsing the data type.

This generic data model is useful to support a wide range of applications.

How Morphlines Act on Data

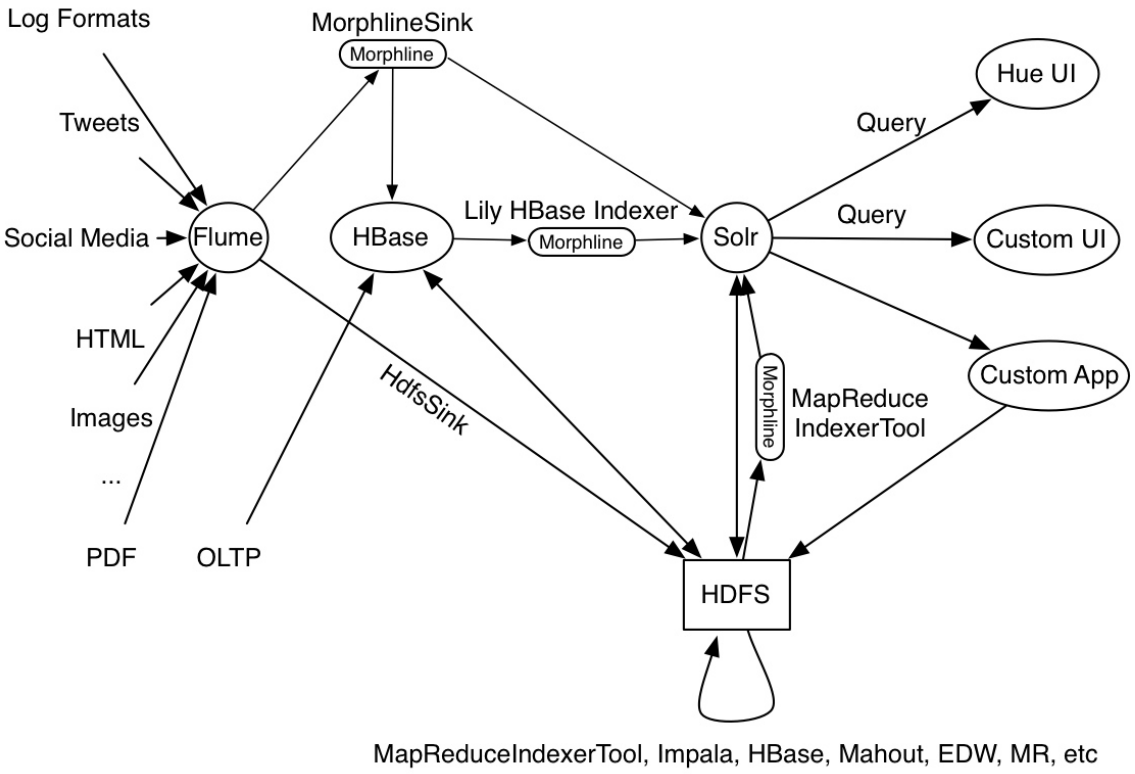
A command transforms a record into zero or more records. Commands can access all record fields. For example, commands can parse fields, set fields, remove fields, rename fields, find and replace values, split a field into multiple fields, split a field into multiple values, or drop records. Often, regular expression based pattern matching is used as part of the process of acting on fields. The output records of a command are passed to the next command in the chain. A command has a Boolean return code, indicating success or failure.

For example, consider the case of a multi-line input record: A command could take this multi-line input record and divide the single record into multiple output records, one for each line. This output could then later be further divided using regular expression commands, splitting each single line record out into multiple fields in application specific ways.

A command can extract, clean, transform, join, integrate, enrich and decorate records in many other ways. For example, a command can join records with external data sources such as relational databases, key-value stores, local files or IP

Geo lookup tables. It can also perform tasks such as DNS resolution, expand shortened URLs, fetch linked metadata from social networks, perform sentiment analysis and annotate the record accordingly, continuously maintain statistics for analytics over sliding windows, compute exact or approximate distinct values and quantiles.

A command can also consume records and pass them to external systems. For example, a command can load records into Solr or write them to a MapReduce Reducer or pass them into an online dashboard. The following diagram illustrates some pathways along which data might flow with the help of morphlines:



Morphline Characteristics

A command can contain nested commands. Thus, a morphline is a tree of commands, akin to a push-based data flow engine or operator tree in DBMS query execution engines.

A morphline has no notion of persistence, durability, distributed computing, or host failover. A morphline is basically just a chain of in-memory transformations in the current thread. There is no need for a morphline to manage multiple processes, hosts, or threads because this is already addressed by host systems such as MapReduce, Flume, or Storm. However, a morphline does support passing notifications on the control plane to command subtrees. Such notifications include BEGIN_TRANSACTION, COMMIT_TRANSACTION, ROLLBACK_TRANSACTION, SHUTDOWN.

The morphline configuration file is implemented using the HOCON format (Human-Optimized Config Object Notation). HOCON is basically JSON slightly adjusted for configuration file use cases. HOCON syntax is defined at HOCON github page and is also used by Akka and Play.

How Morphlines Are Implemented

Cloudera Search includes several maven modules that contain morphline commands for integration with Apache Solr including SolrCloud, flexible log file analysis, single-line records, multi-line records, CSV files, regular expression based pattern matching and extraction, operations on record fields for assignment and comparison, operations on record fields with list and set semantics, if-then-else conditionals, string and timestamp conversions, scripting support for dynamic Java code, a small rules engine, logging, metrics and counters, integration with Avro, integration with Apache SolrCell and all Apache Tika parsers, integration with Apache Hadoop Sequence Files, auto-detection of MIME types

from binary data using Apache Tika, and decompression and unpacking of arbitrarily nested container file formats, among others. These are described in the following chapters.

Example Morphline Usage

The following examples show how you can use morphlines.

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 events,
# 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 multiple
    # morphlines in a morphline config file.
    id : morphline1

    # Import all morphline commands in these java packages and their subpackages.
    # 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 types:
          # 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 another
        # record downstream.
        #
        # extractAvroPaths is a command that uses zero or more Avro path
       # excodeblockssions to extract values from an Avro object. Each excodeblockssion

        # consists of a record output field name, which appears to the left 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 excodeblockssion 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 as a flatten
        # option that collects the primitives in a subtree into a flat list. 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 written 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 document
      # 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}
        }
      }
    ]
  }
]
```

Using Morphlines with Syslog

The following 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 events,
# 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 regular
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_timestamp}
%{SYSLOGHOST:syslog_hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?:
%{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 document
      # 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}
        }
      }
```

```
    ]
  }
]
```

Next Steps

Learn more about morphlines. For more information, see:

- Morphlines Reference Guide

Using the Lily HBase Batch Indexer for Indexing

With Cloudera Search, you can batch index HBase tables using MapReduce jobs. This batch indexing does not require:

- HBase replication
- The Lily HBase Indexer Service
- Registering a Lily HBase Indexer configuration with the Lily HBase Indexer Service

The indexer supports flexible, custom, application-specific rules to extract, transform, and load HBase data into Solr. Solr search results can contain `columnFamily:qualifier` links back to the data stored in HBase. This way, applications can use the search result set to directly access matching raw HBase cells.

Batch indexing column families of tables in an HBase cluster requires:

- Populating an HBase table
- Creating a corresponding collection in Search
- Creating a Lily HBase Indexer configuration
- Creating a Morphline configuration file
- Understanding the `extractHBaseCells` morphline command
- Running HBaseMapReduceIndexerTool

Important: Do not use the Lily HBase Batch Indexer during a rolling upgrade. The indexer requires all replicas be hosted on the same HBase version. If an indexing job is running during a rolling upgrade, different nodes may be running pre- and post-upgrade versions of HBase, resulting in a temporarily unsupported configuration.

Populating an HBase Table

After configuring and starting your system, create an HBase table and add rows to it. For example:

```
$ hbase shell

hbase(main):002:0> create 'record', {NAME => 'data'}
hbase(main):002:0> put 'record', 'row1', 'data', 'value'
hbase(main):001:0> put 'record', 'row2', 'data', 'value2'
```

Creating a Corresponding Collection in Search

A collection in Search used for HBase indexing must have a Solr schema that accommodates the types of HBase column families and qualifiers that are being indexed. To begin, consider adding the all-inclusive `data` field to a default schema. Once you decide on a schema, create a collection using a command of the form:

```
$ solrctl instancedir --generate $HOME/hbase-collection1
$ edit $HOME/hbase-collection1/conf/schema.xml
$ solrctl instancedir --create hbase-collection1 $HOME/hbase-collection1
$ solrctl collection --create hbase-collection1
```

Creating a Lily HBase Indexer Configuration

Configure individual Lily HBase Indexers using the `hbase-indexer` command-line utility. Typically, there is one Lily HBase Indexer configuration for each HBase table, but there can be as many Lily HBase Indexer configurations as there are tables, column families, and corresponding collections in Search. Each Lily HBase Indexer configuration is defined in an XML file, such as `morphline-hbase-mapper.xml`.

An indexer configuration XML file must refer to the `MorphlineResultToSolrMapper` implementation and point to the location of a Morphline configuration file, as shown in the following `morphline-hbase-mapper.xml` indexer configuration file:

```
$ cat $HOME/morphline-hbase-mapper.xml

<?xml version="1.0"?>
<indexer table="record"
mapper="com.ngdata.hbaseindexer.morphline.MorphlineResultToSolrMapper">

   <!-- The relative or absolute path on the local file system to the
   morphline configuration file. -->
   <!-- Use relative path "morphlines.conf" for morphlines managed by
   Cloudera Manager -->
   <param name="morphlineFile" value="/etc/hbase-solr/conf/morphlines.conf"/>

   <!-- The optional morphlineId identifies a morphline if there are multiple
   morphlines in morphlines.conf -->
   <!-- <param name="morphlineId" value="morphline1"/> -->

</indexer>
```

The Lily HBase Indexer configuration file also supports the standard attributes of any HBase Lily Indexer on the top-level `<indexer>` element: `table`, `mapping-type`, `read-row`, `mapper`, `unique-key-formatter`, `unique-key-field`, `row-field`, `column-family-field`, and `table-family-field`. It does not support the `<field>` element and `<extract>` elements.

Creating a Morphline Configuration File

After creating an indexer configuration XML file, control its behavior by configuring morphline ETL transformation commands in a `morphlines.conf` configuration file. The `morphlines.conf` configuration file can contain any number of morphline commands. Typically, an `extractHBaseCells` command is the first command. The `readAvroContainer` or `readAvro` morphline commands are often used to extract Avro data from the HBase byte array. This configuration file can be shared among different applications that use morphlines.

```
$ cat /etc/hbase-solr/conf/morphlines.conf

morphlines : [
  {
    id : morphline1
    importCommands : ["org.kitesdk.morphline.**", "com.ngdata.**"]

    commands : [
      {
        extractHBaseCells {
          mappings : [
            {
              inputColumn : "data:*"
              outputField : "data"
              type : string
              source : value
            }

            #{
            #  inputColumn : "data:item"
            #  outputField : "_attachment_body"
            #  type : "byte[]"
            #  source : value
            #}
          ]
```

```
        }
      }

      #for avro use with type : "byte[]" in extractHBaseCells mapping above
      #{ readAvroContainer {} }
      #{
      #  extractAvroPaths {
      #    paths : {
      #      data : /user_name
      #    }
      #  }
      #}

      { logTrace { format : "output record: {}", args : ["@{}"] } }
    ]
  }
]
```

Note: To function properly, the morphline must not contain a `loadSolr` command. The enclosing Lily HBase Indexer must load documents into Solr, instead the morphline itself.

Understanding the `extractHBaseCells` Morphline Command

The `extractHBaseCells` morphline command extracts cells from an HBase result and transforms the values into a `SolrInputDocument`. The command consists of an array of zero or more mapping specifications.

Each mapping has:

- The `inputColumn` parameter, which specifies the data from HBase for populating a field in Solr. It has the form of a column family name and qualifier, separated by a colon. The qualifier portion can end in an asterisk, which is interpreted as a wildcard. In this case, all matching column-family and qualifier expressions are used. The following are examples of valid `inputColumn` values:
  - `mycolumnfamily:myqualifier`
  - `mycolumnfamily:my*`
  - `mycolumnfamily:*`
- The `outputField` parameter specifies the morphline record field to which to add output values. The morphline record field is also known as the Solr document field. Example: `first_name`.
- Dynamic output fields are enabled by the `outputField` parameter ending with a * wildcard. For example:

```
inputColumn : "m:e:*"
outputField : "belongs_to_*"
```

In this case, if you make these puts in HBase:

```
put 'table_name' , 'row1' , 'm:e:1' , 'foo'
put 'table_name' , 'row1' , 'm:e:9' , 'bar'
```

Then the fields of the Solr document are as follows:

```
belongs_to_1 : foo
belongs_to_9 : bar
```

- The `type` parameter defines the data type of the content in HBase. All input data is stored in HBase as byte arrays, but all content in Solr is indexed as text, so a method for converting byte arrays to the actual data type is required. The type parameter can be the name of a type that is supported by `org.apache.hadoop.hbase.util.Bytes.to*` (which currently includes `byte[]`, `int`, `long`, `string`, `boolean`, `float`, `double`, `short`, and `bigdecimal`). Use type `byte[]` to pass the byte array through to the morphline without conversion.
  - `type:byte[] copies the byte array unmodified into the record output field`

- type:int converts with org.apache.hadoop.hbase.util.Bytes.toInt
- type:long converts with org.apache.hadoop.hbase.util.Bytes.toLong
- type:string converts with org.apache.hadoop.hbase.util.Bytes.toString
- type:boolean converts with org.apache.hadoop.hbase.util.Bytes.toBoolean
- type:float converts with org.apache.hadoop.hbase.util.Bytes.toFloat
- type:double converts with org.apache.hadoop.hbase.util.Bytes.toDouble
- type:short converts with org.apache.hadoop.hbase.util.Bytes.toShort
- type:bigdecimal converts with org.apache.hadoop.hbase.util.Bytes.toBigDecimal

Alternatively, the `type` parameter can be the name of a Java class that implements the `com.ngdata.hbaseindexer.parse.ByteArrayValueMapper` interface.

HBase data formatting does not always match what is specified by `org.apache.hadoop.hbase.util.Bytes.*`. For example, this can occur with data of type `float` or `double`. You can enable indexing of such HBase data by converting the data. There are various ways to do so including:

- Using Java morphline command to parse input data, converting it to the expected output. For example:

```
{
 imports : "import java.util.*;" code: """ // manipulate the contents of a record field

 String stringAmount = (String) record.getFirstValue("amount");
 Double dbl = Double.parseDouble(stringAmount); record.replaceValues("amount",dbl);
 return child.process(record); // pass record to next command in chain """
}
```

- Creating table fields with binary format and then using types such as double or float in a `morphline.conf`. You could create a table in HBase for storing doubles using commands similar to:

```
CREATE TABLE sample_lily_hbase ( id string, amount double, ts timestamp )
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,ti:amount#b,ti:ts,')
TBLPROPERTIES ('hbase.table.name' = 'sample_lily');
```

- The `source` parameter determines which portion of an HBase KeyValue is used as indexing input. Valid choices are `value` or `qualifier`. When `value` is specified, the HBase cell value is used as input for indexing. When `qualifier` is specified, then the HBase column qualifier is used as input for indexing. The default is `value`.

Running HBaseMapReduceIndexerTool

Run HBaseMapReduceIndexerTool to index the HBase table using a MapReduce job, as follows:

```
hadoop --config /etc/hadoop/conf \
jar /usr/lib/hbase-solr/tools/hbase-indexer-mr-*-job.jar \
--conf /etc/hbase/conf/hbase-site.xml -D 'mapred.child.java.opts=-Xmx500m' \
--hbase-indexer-file $HOME/morphline-hbase-mapper.xml \
--zk-host 127.0.0.1/solr --collection hbase-collection1 \
--go-live --log4j src/test/resources/log4j.properties
```

Note: For development purposes, use the `--dry-run` option to run in local mode and print documents to `stdout`, instead of loading them to Solr. Using this option causes the morphline to run in the client process without submitting a job to MapReduce. Running in the client process provides quicker results during early trial and debug sessions.

Note: To print diagnostic information, such as the content of records as they pass through morphline commands, enable TRACE log level diagnostics by adding the following to your `log4j.properties` file:

```
log4j.logger.org.kitesdk.morphline=TRACE
log4j.logger.com.ngdata=TRACE
```

The `log4j.properties` file can be passed using the `--log4j` command-line option.

Understanding `--go-live` and HDFS ACLs

When run with a reduce phase, as opposed to as a mapper-only job, the indexer creates an offline index on HDFS in the output directory specified by the `--output-dir` parameter. If the `--go-live` parameter is specified, Solr merges the resulting offline index into the live running service. Thus, the Solr service must have read access to the contents of the output directory in order to complete the `--go-live` step. If `--overwrite-output-dir` is specified, the indexer deletes and recreates any existing output directory; in an environment with restrictive permissions, such as one with an HDFS umask of 077, the Solr user may not be able to read the contents of the newly created directory. To address this issue, the indexer automatically applies the HDFS ACLs to enable Solr to read the output directory contents. These ACLs are only applied if HDFS ACLs are enabled on the HDFS NameNode. For more information, see HDFS Extended ACLs.

The indexer only makes ACL updates to the output directory and its contents. If the output directory's parent directories do not include the run permission, the Solr service cannot access the output directory. Solr must have run permissions from standard permissions or ACLs on the parent directories of the output directory.

HBaseMapReduceIndexerTool

HBaseMapReduceIndexerTool is a MapReduce batch job driver that takes input data from an HBase table, creates Solr index shards, and writes the indexes into HDFS in a flexible, scalable, and fault-tolerant manner. It also supports merging the output shards into a set of live customer-facing Solr servers in SolrCloud.

Important: Merging output shards into live customer-facing Solr servers can only be completed if all replicas are online.

- To invoke the command-line help in a default parcels installation, use:

```
$ hadoop jar /opt/cloudera/parcels/CDH-*/jars/hbase-indexer-mr-*-job.jar --help
```

- To invoke the command-line help in a default packages installation, use:

```
$ hadoop jar /usr/lib/hbase-solr/tools/hbase-indexer-mr-*-job.jar --help
```

```
usage: hadoop [GenericOptions]... jar hbase-indexer-mr-*-job.jar
       [--hbase-indexer-zk STRING] [--hbase-indexer-name STRING]
       [--hbase-indexer-file FILE]
       [--hbase-indexer-component-factory STRING]
       [--hbase-table-name STRING] [--hbase-start-row BINARYSTRING]
       [--hbase-end-row BINARYSTRING] [--hbase-start-time STRING]
       [--hbase-end-time STRING] [--hbase-timestamp-format STRING]
       [--zk-host STRING] [--go-live] [--collection STRING]
       [--go-live-threads INTEGER] [--help] [--output-dir HDFS_URI]
       [--overwrite-output-dir] [--morphline-file FILE]
       [--morphline-id STRING] [--update-conflict-resolver FQCN]
       [--reducers INTEGER] [--max-segments INTEGER]
       [--fair-scheduler-pool STRING] [--dry-run] [--log4j FILE]
       [--verbose] [--clear-index] [--show-non-solr-cloud]

MapReduce batch job driver that takes  input  data  from an HBase table and
creates Solr index shards and writes the  indexes into HDFS, in a flexible,
```

```
scalable, and fault-tolerant manner. It also supports merging the output
shards into a set of live customer-facing Solr servers in SolrCloud.
Optionally, documents can be sent directly from the mapper tasks to
SolrCloud, which is a much less scalable approach but enables updating
existing documents in SolrCloud. The program proceeds in one or multiple
consecutive MapReduce-based phases, as follows:

1) Mapper phase: This (parallel) phase scans over the input HBase table,
extracts the relevant content, and transforms it into SolrInputDocuments.
If run as a mapper-only job, this phase also writes the SolrInputDocuments
directly to a live SolrCloud cluster. The conversion from HBase records
into Solr documents is performed via a hbase-indexer configuration and
typically based on a morphline.

2) Reducer phase: This (parallel) phase loads the mapper's
SolrInputDocuments into one EmbeddedSolrServer per reducer. Each such
reducer and Solr server can be seen as a (micro) shard. The Solr servers
store their data in HDFS.

3) Mapper-only merge phase: This (parallel) phase merges the set of
reducer shards into the number of Solr shards expected by the user, using
a mapper-only job. This phase is omitted if the number of shards is
already equal to the number of shards expected by the user

4) Go-live phase: This optional (parallel) phase merges the output shards
of the previous phase into a set of live customer-facing Solr servers in
SolrCloud. If this phase is omitted you can explicitly point each Solr
server to one of the HDFS output shard directories

Fault Tolerance: Mapper and reducer task attempts are retried on failure
per the standard MapReduce semantics. On program startup all data in the --
output-dir is deleted if that output directory already exists and --
overwrite-output-dir is specified. This means that if the whole job fails
you can retry simply by rerunning the program again using the same
arguments.

HBase Indexer parameters:
  Parameters for specifying the HBase indexer definition and where it
  should be loaded from.

  --hbase-indexer-zk STRING
                        The address of the ZooKeeper ensemble from which
                        to fetch the indexer definition named --hbase-
                        indexer-name. Format is: a list of comma
                        separated host:port pairs, each corresponding to
                        a zk server. Example: '127.0.0.1:2181,127.0.0.1:
                        2182,127.0.0.1:2183'
  --hbase-indexer-name STRING
                        The name of the indexer configuration to fetch
                        from the ZooKeeper ensemble specified with --
                        hbase-indexer-zk. Example: myIndexer
  --hbase-indexer-file FILE
                        Optional relative or absolute path to a local
                        HBase indexer XML configuration file. If supplied,
                        this overrides --hbase-indexer-zk and
                        --hbase-indexer-name. Example:
                        /path/to/morphline-hbase-mapper.xml
  --hbase-indexer-component-factory STRING
                        Classname of the hbase indexer component factory.

HBase scan parameters:
  Parameters for specifying what data is included while reading from HBase.

  --hbase-table-name STRING
                        Optional name of the HBase table containing the
                        records to be indexed. If supplied, this
                        overrides the value from the --hbase-indexer-*
                        options. Example: myTable
  --hbase-start-row BINARYSTRING
                        Binary string representation of start row from
                        which to start indexing (inclusive). The format
                        of the supplied row key should use two-digit hex
                        values prefixed by \x for non-ASCII characters (e.
```

```
                        g. 'row\x00'). The semantics of this argument are
                        the same as those for the HBase Scan#setStartRow
                        method. The default is to include the first row
                        of the table. Example: AAAA
  --hbase-end-row BINARYSTRING
                        Binary string representation of end row prefix at
                        which to stop indexing (exclusive). See the
                        description of --hbase-start-row for more
                        information. The default is to include the last
                        row of the table. Example: CCCC
  --hbase-start-time STRING
                        Earliest timestamp (inclusive) in time range of
                        HBase cells to be included for indexing. The
                        default is to include all cells. Example: 0
  --hbase-end-time STRING
                        Latest timestamp (exclusive) of HBase cells to be
                        included for indexing. The default is to include
                        all cells. Example: 123456789
  --hbase-timestamp-format STRING
                        Timestamp format to be used to interpret --hbase-
                        start-time and --hbase-end-time. This is a java.
                        text.SimpleDateFormat compliant format (see http:
                        //docs.oracle.
                        com/javase/6/docs/api/java/text/SimpleDateFormat.
                        html). If this parameter is omitted then the
                        timestamps are interpreted as number of
                        milliseconds since the standard epoch (Unix
                        time). Example: "yyyy-MM-dd'T'HH:mm:ss.SSSZ"

Solr cluster arguments:
  Arguments that provide information about your Solr cluster.

  --zk-host STRING       The address of a ZooKeeper ensemble being used by
                        a SolrCloud cluster. This ZooKeeper ensemble will
                        be examined to determine the number of output
                        shards to create as well as the Solr URLs to
                        merge the output shards into when using the --go-
                        live option. Requires that you also pass the --
                        collection to merge the shards into.

                        The --zk-host option implements the same
                        partitioning semantics as the standard SolrCloud
                        Near-Real-Time (NRT) API. This enables to mix
                        batch updates from MapReduce ingestion with
                        updates from standard Solr NRT ingestion on the
                        same SolrCloud cluster, using identical unique
                        document keys.

                        Format is: a list of comma separated host:port
                        pairs, each corresponding to a zk server.
                        Example: '127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:
                        2183' If the optional chroot suffix is used the
                        example would look like: '127.0.0.1:2181/solr,
                        127.0.0.1:2182/solr,127.0.0.1:2183/solr' where
                        the client would be rooted at '/solr' and all
                        paths would be relative to this root - i.e.
                        getting/setting/etc... '/foo/bar' would result in
                        operations being run on '/solr/foo/bar' (from the
                        server perspective).


Go live arguments:
  Arguments for merging the shards that are built into a live Solr
  cluster. Also see the Cluster arguments.

  --go-live              Allows you to optionally merge the final index
                        shards into a live Solr cluster after they are
                        built. You can pass the ZooKeeper address with --
                        zk-host and the relevant cluster information will
                        be auto detected. (default: false)
  --collection STRING    The SolrCloud collection to merge shards into
                        when using --go-live and --zk-host. Example:
                        collection1
```

```
  --go-live-threads INTEGER
                        Tuning knob that indicates the maximum number of
                        live merges to run in parallel at one time.
                        (default: 1000)

Optional arguments:
  --help, -help, -h     Show this help message and exit
  --output-dir HDFS_URI HDFS directory to write Solr indexes to. Inside
                        there one output directory per shard will be
                        generated. Example: hdfs://c2202.mycompany.
                        com/user/$USER/test
  --overwrite-output-dir
                        Overwrite the directory specified by --output-dir
                        if it already exists. Using this parameter will
                        result in the output directory being recursively
                        deleted at job startup. (default: false)
  --morphline-file FILE Relative or absolute path to a local config file
                        that contains one or more morphlines. The file
                        must be UTF-8 encoded. The file will be uploaded
                        to each MR task. If supplied, this overrides the
                        value from the --hbase-indexer-* options.
                        Example: /path/to/morphlines.conf
  --morphline-id STRING The identifier of the morphline that shall be
                        executed within the morphline config file, e.g.
                        specified by --morphline-file. If the --morphline-
                        id option is omitted the first (i.e. top-most)
                        morphline within the config file is used. If
                        supplied, this overrides the value from the --
                        hbase-indexer-* options. Example: morphline1
  --update-conflict-resolver FQCN
                        Fully qualified class name of a Java class that
                        implements the UpdateConflictResolver interface.
                        This enables deduplication and ordering of a
                        series of document updates for the same unique
                        document key. For example, a MapReduce batch job
                        might index multiple files in the same job where
                        some of the files contain old and new versions of
                        the very same document, using the same unique
                        document key.
                        Typically, implementations of this interface
                        forbid collisions by throwing an exception, or
                        ignore all but the most recent document version,
                        or, in the general case, order colliding updates
                        ascending from least recent to most recent
                        (partial) update. The caller of this interface (i.
                        e. the Hadoop Reducer) will then apply the
                        updates to Solr in the order returned by the
                        orderUpdates() method.
                        The                                    default
                        RetainMostRecentUpdateConflictResolver
                        implementation ignores all but the most recent
                        document version, based on a configurable numeric
                        Solr field, which defaults to the
                        file_last_modified timestamp (default: org.apache.
                        solr.hadoop.dedup.
                        RetainMostRecentUpdateConflictResolver)
  --reducers INTEGER    Tuning knob that indicates the number of reducers
                        to index into. 0 indicates that no reducers
                        should be used, and documents should be sent
                        directly from the mapper tasks to live Solr
                        servers. -1 indicates use all reduce slots
                        available on the cluster. -2 indicates use one
                        reducer per output shard, which disables the
                        mtree merge MR algorithm. The mtree merge MR
                        algorithm improves scalability by spreading load
                        (in particular CPU load) among a number of
                        parallel reducers that can be much larger than
                        the number of solr shards expected by the user.
                        It can be seen as an extension of concurrent
                        lucene merges and tiered lucene merges to the
                        clustered case. The subsequent mapper-only phase
                        merges the output of said large number of
                        reducers to the number of shards expected by the
```

```
                          user,   again   by    utilizing   more   available
                          parallelism on the cluster. (default: -1)
  --max-segments INTEGER
                          Tuning knob that indicates  the  maximum number of
                          segments to be contained  on  output  in the index
                          of each reducer shard.  After  a reducer has built
                          its output index  it  applies  a  merge  policy to
                          merge segments  until  there  are  <=  maxSegments
                          lucene  segments  left  in   this  index.  Merging
                          segments involves reading  and  rewriting all data
                          in all these  segment  files, potentially multiple
                          times,  which  is  very  I/O  intensive  and  time
                          consuming. However, an  index  with fewer segments
                          can later be merged  faster,  and  it can later be
                          queried  faster  once  deployed  to  a  live  Solr
                          serving shard. Set  maxSegments  to  1 to optimize
                          the index for low query  latency. In a nutshell, a
                          small maxSegments  value  trades  indexing latency
                          for subsequently improved query  latency. This can
                          be  a  reasonable  trade-off  for  batch  indexing
                          systems. (default: 1)
  --fair-scheduler-pool STRING
                          Optional tuning knob  that  indicates  the name of
                          the fair scheduler  pool  to  submit  jobs to. The
                          Fair Scheduler is a  pluggable MapReduce scheduler
                          that provides a way to  share large clusters. Fair
                          scheduling is a method  of  assigning resources to
                          jobs such that all jobs  get, on average, an equal
                          share of resources  over  time.  When  there  is a
                          single job  running,  that  job  uses  the  entire
                          cluster. When  other  jobs  are  submitted,  tasks
                          slots that free up are  assigned  to the new jobs,
                          so that each job gets  roughly  the same amount of
                          CPU time.  Unlike  the  default  Hadoop scheduler,
                          which forms a queue of  jobs, this lets short jobs
                          finish in reasonable time  while not starving long
                          jobs. It is also an  easy  way  to share a cluster
                          between multiple of users.  Fair  sharing can also
                          work with  job  priorities  -  the  priorities are
                          used as  weights  to  determine  the  fraction  of
                          total compute time that each job gets.
  --dry-run               Run in local mode  and  print  documents to stdout
                          instead of loading them  into  Solr. This executes
                          the  morphline  in  the  client  process  (without
                          submitting a job  to  MR)  for  quicker turnaround
                          during early  trial  &  debug  sessions. (default:
                          false)
  --log4j FILE            Relative or absolute  path  to  a log4j.properties
                          config file on the  local  file  system. This file
                          will  be  uploaded  to   each  MR  task.  Example:
                          /path/to/log4j.properties
  --verbose, -v           Turn on verbose output. (default: false)
  --clear-index           Will attempt  to  delete  all  entries  in  a solr
                          index before starting  batch  build.  This  is not
                          transactional so  if  the  build  fails  the index
                          will be empty. (default: false)
  --show-non-solr-cloud   Also show options for  Non-SolrCloud  mode as part
                          of --help. (default: false)

Generic options supported are
  --conf <configuration file>
                          specify an application configuration file
  -D <property=value>     use value for given property
  --fs <local|namenode:port>
                          specify a namenode
  --jt <local|jobtracker:port>
                          specify a job tracker
  --files <comma separated list of files>
                          specify comma separated files to  be copied to the
                          map reduce cluster
  --libjars <comma separated list of jars>
                          specify comma separated  jar  files  to include in
                          the classpath.
```

```
  --archives <comma separated list of archives>
                          specify comma separated archives  to be unarchived
                          on the compute machines.

The general command line syntax is
bin/hadoop command [genericOptions] [commandOptions]

Examples:

# (Re)index a table in GoLive mode based on a local indexer config file
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --hbase-indexer-file indexer.xml \
  --zk-host 127.0.0.1/solr \
  --collection collection1 \
  --go-live \
  --log4j src/test/resources/log4j.properties

# (Re)index a table in GoLive mode using a local morphline-based indexer config file
# Also include extra library jar file containing JSON tweet Java parser:
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  --libjars /path/to/kite-morphlines-twitter-0.10.0.jar \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --hbase-indexer-file src/test/resources/morphline_indexer_without_zk.xml \
  --zk-host 127.0.0.1/solr \
  --collection collection1 \
  --go-live \
  --morphline-file src/test/resources/morphlines.conf \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --overwrite-output-dir \
  --log4j src/test/resources/log4j.properties

# (Re)index a table in GoLive mode
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --hbase-indexer-file indexer.xml \
  --zk-host 127.0.0.1/solr \
  --collection collection1 \
  --go-live \
  --log4j src/test/resources/log4j.properties

# (Re)index a table with direct writes to SolrCloud
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --hbase-indexer-file indexer.xml \
  --zk-host 127.0.0.1/solr \
  --collection collection1 \
  --reducers 0 \
  --log4j src/test/resources/log4j.properties

# (Re)index a table based on a indexer config stored in ZK
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --hbase-indexer-zk zk01 \
  --hbase-indexer-name docindexer \
  --go-live \
  --log4j src/test/resources/log4j.properties

# MapReduce on Yarn - Pass custom JVM arguments
HADOOP_CLIENT_OPTS='-DmaxConnectionsPerHost=10000 -DmaxConnections=10000'; \
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
```

```
  -D 'mapreduce.map.java.opts=-DmaxConnectionsPerHost=10000 -DmaxConnections=10000' \
  -D 'mapreduce.reduce.java.opts=-DmaxConnectionsPerHost=10000 -DmaxConnections=10000'
 \
  --hbase-indexer-zk zk01 \
  --hbase-indexer-name docindexer \
  --go-live \
  --log4j src/test/resources/log4j.properties\n

# MapReduce on MR1 - Pass custom JVM arguments
HADOOP_CLIENT_OPTS='-DmaxConnectionsPerHost=10000 -DmaxConnections=10000'; \
hadoop --config /etc/hadoop/conf \
  jar hbase-indexer-mr-*-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  -D 'mapreduce.child.java.opts=-DmaxConnectionsPerHost=10000 -DmaxConnections=10000'
\
  --hbase-indexer-zk zk01 \ "  --hbase-indexer-name docindexer \
  --go-live \
  --log4j src/test/resources/log4j.properties\n\n");
```

Configuring the Lily HBase NRT Indexer Service for Use with Cloudera Search

The Lily HBase NRT Indexer Service is a flexible, scalable, fault-tolerant, transactional, near real-time (NRT) system for processing a continuous stream of HBase cell updates into live search indexes. Typically it takes seconds for data ingested into HBase to appear in search results; this duration is tunable. The Lily HBase Indexer uses SolrCloud to index data stored in HBase. As HBase applies inserts, updates, and deletes to HBase table cells, the indexer keeps Solr consistent with the HBase table contents, using standard HBase replication. The indexer supports flexible custom application-specific rules to extract, transform, and load HBase data into Solr. Solr search results can contain `columnFamily:qualifier` links back to the data stored in HBase. This way, applications can use the Search result set to directly access matching raw HBase cells. Indexing and searching do not affect operational stability or write throughput of HBase because the indexing and searching processes are separate and asynchronous to HBase.

The Lily HBase NRT Indexer Service must be deployed in an environment with a running HBase cluster, a running SolrCloud cluster, and at least one ZooKeeper cluster. This can be done with or without Cloudera Manager. See Managing Services for more information on adding services such as the Lily HBase Indexer Service.

Enabling Cluster-wide HBase Replication

The Lily HBase Indexer is implemented using HBase replication, presenting indexers as RegionServers of the worker cluster. This requires HBase replication on the HBase cluster, as well as the individual tables to be indexed. An example of settings required for configuring cluster-wide HBase replication is shown in `/usr/share/doc/hbase-solr-doc*/demo/hbase-site.xml`. You must add these settings to all of the `hbase-site.xml` configuration files on the HBase cluster, except the `replication.replicationsource.implementation` property. You can use the Cloudera Manager HBase Indexer Service GUI to do this. After making these updates, restart your HBase cluster.

Pointing a Lily HBase NRT Indexer Service at an HBase Cluster that Needs to Be Indexed

Before starting Lily HBase NRT Indexer services, you must configure individual services with the location of a ZooKeeper ensemble that is used for the target HBase cluster. Add the following property to `/etc/hbase-solr/conf/hbase-indexer-site.xml`. Remember to replace `hbase-cluster-zookeeper` with the actual ensemble string found in the `hbase-site.xml` configuration file:

```
<property>
   <name>hbase.zookeeper.quorum</name>
   <value>hbase-cluster-zookeeper</value>
</property>
```

Configure all Lily HBase NRT Indexer Services to use a particular ZooKeeper ensemble to coordinate with one another. Add the following property to `/etc/hbase-solr/conf/hbase-indexer-site.xml`, and replace `hbase-cluster-zookeeper:2181` with the actual ensemble string:

```
<property>
   <name>hbaseindexer.zookeeper.connectstring</name>
   <value>hbase-cluster-zookeeper:2181</value>
</property>
```

Configuring Lily HBase Indexer Security

Beginning with CDH 5.4 the Lily HBase Indexer includes an HTTP interface for the `list-indexers`, `create-indexer`, `update-indexer`, and `delete-indexer` commands. This interface can be configured to use Kerberos and to integrate with Sentry.

Configuring Lily HBase Indexer to Use Security

To configure the Lily HBase Indexer to use security, you must create principals and keytabs and then modify default configurations.

To create principals and keytabs

Repeat this process on all Lily HBase Indexer hosts.

1. Create a Lily HBase Indexer service user principal using the syntax: `hbase/<fully.qualified.domain.name>@<YOUR-REALM>`. This principal is used to authenticate with the Hadoop cluster. where: `fully.qualified.domain.name` is the host where the Lily HBase Indexer is running `YOUR-REALM` is the name of your Kerberos realm.

   ```
   $ kadmin
   kadmin: addprinc -randkey hbase/fully.qualified.domain.name@YOUR-REALM.COM
   ```

2. Create a HTTP service user principal using the syntax: `HTTP/<fully.qualified.domain.name>@<YOUR-REALM>`. This principal is used to authenticate user requests coming to the Lily HBase Indexer web-services. where: `fully.qualified.domain.name` is the host where the Lily HBase Indexer is running `YOUR-REALM` is the name of your Kerberos realm.

   ```
   kadmin: addprinc -randkey HTTP/fully.qualified.domain.name@YOUR-REALM.COM
   ```

   Note:

   The `HTTP/` component of the HTTP service user principal must be upper case as shown in the syntax and example above.

3. Create keytab files with both principals.

   ```
   kadmin: xst -norandkey -k hbase.keytab hbase/fully.qualified.domain.name \
   HTTP/fully.qualified.domain.name
   ```

4. Test that credentials in the merged keytab file work. For example:

   ```
   $ klist -e -k -t hbase.keytab
   ```

5. Copy the `hbase.keytab` file to the Lily HBase Indexer configuration directory. The owner of the `hbase.keytab` file should be the `hbase` user and the file should have owner-only read permissions.

To modify default configurations

Repeat this process on all Lily HBase Indexer hosts.

1. Modify the `hbase-indexer-site.xml` file as follows:

```
  <property>
    <name>hbaseindexer.authentication.type</name>
    <value>kerberos</value>
```

```
  </property>
  <property>
    <name>hbaseindexer.authentication.kerberos.keytab</name>
    <value>hbase.keytab</value>
  </property>
  <property>
    <name>hbaseindexer.authentication.kerberos.principal</name>
    <value>HTTP/localhost@LOCALHOST</value>
  </property>
  <property>
    <name>hbaseindexer.authentication.kerberos.name.rules</name>
    <value>DEFAULT</value>
  </property>
```

2. Set up the Java Authentication and Authorization Service (JAAS). Create a `jaas.conf` file in the HBase-Indexer configuration directory containing the following settings. Make sure that you substitute a value for `principal` that matches your particular environment.

   ```
   Client {
     com.sun.security.auth.module.Krb5LoginModule required
     useKeyTab=true
     useTicketCache=false
     keyTab="/etc/hbase/conf/hbase.keytab"
     principal="hbase/fully.qualified.domain.name@<YOUR-REALM>";
   };
   ```

   Then, modify `hbase-indexer-env.sh` in the `hbase-indexer` configuration directory to add the jaas configuration to the system properties. You can do this by adding `-Djava.security.auth.login.config` to the `HBASE_INDEXER_OPTS`. For example, you might add the following:

```
HBASE_INDEXER_OPTS = "$HBASE_INDEXER_OPTS
-Djava.security.auth.login.config=/path/to/your/jaas.conf"
```

Sentry integration

The Lily HBase Indexer uses a file-based access control model similar to that provided by Solr-Sentry integration, which is described in Configuring Sentry Authorization for Solr on page 87. For details on configuring the HTTP API, which Sentry requires, see Configuring Clients to Use the HTTP Interface on page 68. The Lily HBase Indexer's file-based access control model supports specifying READ and WRITE privileges for each indexer. The privileges work as follows:

- If role has `WRITE` privilege for indexer1, a call to create, update, or delete indexer1 succeeds.
- If role has `READ` privilege for indexer1, a call to list-indexers will list indexer1, if it exists. If an indexer called indexer2 exists, but the role does not have `READ` privileges for it, information about indexer2 is filtered out of the response.

To configure Sentry for the Lily HBase Indexer, add the following properties to `hbase-indexer-site.xml`:

```
  <property>
    <name>sentry.hbaseindexer.sentry.site</name>
    <value>sentry-site.xml</value> (full or relative path)
  </property>
  <property>
    <name>hbaseindexer.rest.resource.package</name>
    <value>org/apache/sentry/binding/hbaseindexer/rest</value>
  </property>
```

Note: These settings can be added using Cloudera Manager or by manually editing the `hbase-indexer-site.xml` file.

Starting a Lily HBase NRT Indexer Service

You can use the Cloudera Manager GUI to start Lily HBase NRT Indexer Service on a set of machines. In non-managed deployments, you can start a Lily HBase Indexer Daemon manually on the local host with the following command:

```
sudo service hbase-solr-indexer restart
```

After starting the Lily HBase NRT Indexer Services, verify that all daemons are running using the jps tool from the Oracle JDK, which you can obtain from the Java SE Downloads page. If you are running a pseudo-distributed HDFS installation and a Lily HBase NRT Indexer Service installation on one machine, jps shows the following output:

```
$ sudo jps -lm
31407 sun.tools.jps.Jps -lm
26393 com.ngdata.hbaseindexer.Main
```

Using the Lily HBase NRT Indexer Service

To index for column families of tables in an HBase cluster:

- Enable replication on HBase column families
- Create collections and configurations
- Register a Lily HBase Indexer configuration with the Lily HBase Indexer Service
- Verify that indexing is working

Enabling Replication on HBase Column Families

Ensure that cluster-wide HBase replication is enabled. Use the HBase shell to define column-family replication settings.

For every existing table, set the `REPLICATION_SCOPE` on every column family that needs to be indexed by issuing a command of the form:

```
$ hbase shell
hbase shell> disable 'record'
hbase shell> alter 'record', {NAME => 'data', REPLICATION_SCOPE => 1}
hbase shell> enable 'record'
```

For every new table, set the `REPLICATION_SCOPE` on every column family that needs to be indexed by issuing a command of the form:

```
$ hbase shell
hbase shell> create 'record', {NAME => 'data', REPLICATION_SCOPE => 1}
```

Creating Collections and Configurations

The tasks required for the Lily HBase NRT Indexer Services are the same as those described for the Lily HBase Batch Indexer. Follow the steps described in these sections:

- Creating a Corresponding Collection in Search on page 53
- Creating a Lily HBase Indexer Configuration on page 54
- Creating a Morphline Configuration File on page 54

Registering a Lily HBase Indexer Configuration with the Lily HBase Indexer Service

When the content of the Lily HBase Indexer configuration XML file is satisfactory, register it with the Lily HBase Indexer Service. Register the Lily HBase Indexer configuration file by uploading the Lily HBase Indexer configuration XML file to ZooKeeper. For example:

```
$ hbase-indexer add-indexer \
--name myIndexer \
--indexer-conf $HOME/morphline-hbase-mapper.xml \
```

```
--connection-param solr.zk=solr-cloude-zk1,solr-cloude-zk2/solr \
--connection-param solr.collection=hbase-collection1 \
--zookeeper hbase-cluster-zookeeper:2181
```

Verify that the indexer was successfully created as follows:

```
$ hbase-indexer list-indexers
Number of indexes: 1

myIndexer
  + Lifecycle state: ACTIVE
  + Incremental indexing state: SUBSCRIBE_AND_CONSUME
  + Batch indexing state: INACTIVE
  + SEP subscription ID: Indexer_myIndexer
  + SEP subscription timestamp: 2013-06-12T11:23:35.635-07:00
  + Connection type: solr
  + Connection params:
    + solr.collection = hbase-collection1
    + solr.zk = localhost/solr
  + Indexer config:
      110 bytes, use -dump to see content
  + Batch index config:
      (none)
  + Default batch index config:
      (none)
  + Processes
    + 1 running processes
    + 0 failed processes
```

Use the `update-indexer` and `delete-indexer` command-line options of the `hbase-indexer` utility to manipulate existing Lily HBase Indexers.

For more help, use the following commands:

```
$ hbase-indexer add-indexer --help
$ hbase-indexer list-indexers --help
$ hbase-indexer update-indexer --help
$ hbase-indexer delete-indexer --help
```

The `morphlines.conf` configuration file must be present on every host that runs an indexer.

You can use the Cloudera Manager Admin Console to update `morphlines.conf`:

1. Go to the Key-Value Store Indexer service.
2. Click the **Configuration** tab.
3. Select **Scope** > **KS_INDEXER (Service Wide)**
4. Select **Category** > **Morphlines**.
5. For the **Morphlines File** property, paste the new `morphlines.conf` content into the **Value** field.
6. Click **Save Changes** to commit the changes.

Cloudera Manager automatically copies pasted configuration files to the current working directory of all Lily HBase Indexer cluster processes on start and restart of the Lily HBase Indexer Service. In this case, the file location `/etc/hbase-solr/conf/morphlines.conf` is not applicable.

Morphline configuration files can be changed without re-creating the indexer itself. In such a case, you must restart the Lily HBase Indexer service.

Verifying that Indexing Works

Add rows to the indexed HBase table. For example:

```
$ hbase shell
hbase(main):001:0> put 'record', 'row1', 'data', 'value'
hbase(main):002:0> put 'record', 'row2', 'data', 'value2'
```

If the put operation succeeds, wait a few seconds, go to the SolrCloud UI query page, and query the data. Note the updated rows in Solr.

To print diagnostic information, such as the content of records as they pass through the morphline commands, enable the TRACE log level. For example, you might add two lines to your `log4j.properties` file:

```
log4j.logger.com.cloudera.cdk.morphline=TRACE
log4j.logger.com.ngdata=TRACE
```

In Cloudera Manager do the following:

1. Go to the Key-Value Store Indexer service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Lily HBase Indexer**.
4. Select **Category** > **Advanced**.
5. Locate the **Lily HBase Indexer Logging Advanced Configuration Snippet (Safety Valve)** property or search for it by typing its name in the Search box.

   If more than one role group applies to this configuration, edit the value for the appropriate role group. See Modifying Configuration Properties Using Cloudera Manager.
6. Click **Save Changes** to commit the changes.
7. Restart the Key-Value Store Indexer service.

> **Note:** The name of the particular Key-Value Store Indexer service can vary. The most common variation is a different number at the end of the name.

Examine the log files in `/var/log/hbase-solr/lily-hbase-indexer-*` for details.

Configuring Clients to Use the HTTP Interface

By default, the client does not use the new HTTP interface. Use the HTTP interface only if you want to take advantage of one of the features it provides, such as Kerberos authentication and Sentry integration. The client now supports passing two additional parameters to the `list-indexers`, `create-indexer`, `delete-indexer`, and `update-indexer` commands:

- `--http`: An HTTP URI to the hbase-indexer HTTP API. By default, this URI is of the form `http://host:11060/indexer/`. If this URI is passed, the Lily HBase Indexer uses the HTTP API. If this URI is not passed, the indexer uses the old behavior of communicating directly with ZooKeeper.
- `--jaas`: The specification of a `jaas` configuration file. This is only necessary for Kerberos-enabled deployments.

> **Note:** Cloudera recommends using FQDNs for both the Lily HBase Indexer host and the ZooKeeper host. Using FQDNs helps ensure proper Kerberos domain mapping.

For example:

```
hbase-indexer list-indexers --http http://host:port/indexer/ \
--jaas jaas.conf --zookeeper host:port
```

Backing Up and Restoring Cloudera Search

CDH 5.9 and higher include a backup/restore mechanism primarily designed to provide disaster recovery capability for Apache Solr. You can create a backup of a Solr collection and restore from this backup if the index is corrupted due to a software bug, or if an administrator accidentally or maliciously deletes a collection or a subset of documents. This procedure can also be used as part of a cluster migration (for example, if you are migrating to a cloud environment), or to recover from a failed upgrade.

At a high level, the steps to back up a Solr collection are as follows:

1. Create a snapshot.
2. If you are exporting the snapshot to a remote cluster, prepare the snapshot for export.
3. Export the snapshot to either the local cluster or a remote one. This step uses the Hadoop DistCP utility.

The backup operation uses the native Solr snapshot capability to capture a point-in-time, consistent state of index data for a specified Solr collection. You can then use the Hadoop DistCp utility to copy the index files and the associated metadata for that snapshot to a specified location in HDFS or a cloud object store (for example, Amazon S3).

The Solr snapshot mechanism is based on the Apache Lucene IndexDeletionPolicy abstraction, which enables applications such as Cloudera Search to manage the lifecycle of specific index commits. A Solr snapshot assigns a user-specified name to the latest hard-committed state. After the snapshot is created, the Lucene index files associated with the commit are retained until the snapshot is explicitly deleted. The index files associated with the snapshot are preserved regardless of document updates and deletions, segment merges during index optimization, and so on.

Creating a snapshot does not take much time because it only preserves the snapshot metadata and does not copy the associated index files. A snapshot does have some storage overhead corresponding to the size of the index because the index files are retained indefinitely until the snapshot is deleted.

Solr snapshots can help you recover from some scenarios, such as accidental or malicious data modification or deletion. They are insufficient to address others, such as index corruption and accidental or malicious administrative action (for example, deleting a collection, changing collection configuration, and so on). To address these situations, export snapshots regularly and before performing non-trivial administrative operations such as changing the schema, splitting shards, or deleting replicas.

Exporting a snapshot exports the collection metadata as well as the corresponding Lucene index files. This operation internally uses the Hadoop DistCp utility to copy the Lucene index files and metadata, which creates a full backup at the specified location. After the backup is created, the original Solr snapshot can be safely deleted if necessary.

Important: If you create a snapshot and do not export it, you do not have a complete backup, and cannot restore index files if they are accidentally or maliciously deleted.

Prerequisites

Before you begin backing up Cloudera Search, make sure that `solr.xml` contains following configuration section:

```
  <backup>
    <repository name="hdfs"
class="org.apache.solr.core.backup.repository.HdfsBackupRepository" default="true">
      <str name="solr.hdfs.home">${solr.hdfs.home:}</str>
      <str name="solr.hdfs.confdir">${solr.hdfs.confdir:}</str>
      <str
name="solr.hdfs.security.kerberos.enabled">${solr.hdfs.security.kerberos.enabled:false}</str>

      <str
name="solr.hdfs.security.kerberos.keytabfile">${solr.hdfs.security.kerberos.keytabfile:}</str>

      <str
name="solr.hdfs.security.kerberos.principal">${solr.hdfs.security.kerberos.principal:}</str>

      <str
name="solr.hdfs.permissions.umask-mode">${solr.hdfs.permissions.umask-mode:000}</str>
    </repository>
  </backup>
```

New installations of CDH 5.9 and higher include this section. For upgrades, you must manually add it and restart the Solr service.

To edit the `solr.xml` file:

1. Download `solr.xml` from ZooKeeper:

```
$ solrctl cluster --get-solrxml $HOME/solr.xml
```

2. Edit the file locally and add the <backup> section at the end of the `solr.xml` file before the closing </solr> tag:

```
$ vi $HOME/solr.xml
```

3. Upload the modified file to ZooKeeper:

```
$ solrctl cluster --put-solrxml $HOME/solr.xml
```

Sentry Configuration

As part of backup/restore mechanism, the following APIs are introduced. The following table identifies the Sentry authorization permissions to configure. For information on configuring Sentry privileges, see Configuring Sentry Authorization for Solr on page 87.

Table 4: Solr Snapshot Sentry Permissions

Action	Privilege	Collections
CREATESNAPSHOT	UPDATE	admin, *<collection_name>*
DELETESNAPSHOT	UPDATE	admin, *<collection_name>*
LISTSNAPSHOTS	QUERY	admin, *<collection_name>*
BACKUP	UPDATE	admin, *<collection_name>*
RESTORE	UPDATE	admin, *<collection_name>*

Backing Up a Solr Collection

Use the following procedure to back up a Solr collection. For more information on the commands used, see Cloudera Search Backup and Restore Command Reference on page 73.

If you are using a secure (Kerberos-enabled) cluster, specify your `jaas.conf` file by adding the following parameter to each command:

```
-jaas /path/to/jaas.conf
```

If TLS is enabled for the Solr service, specify the truststore and password using the `ZKCLI_JVM_FLAGS` environment variable before you begin the procedure:

```
$ export ZKCLI_JVM_FLAGS="-Djavax.net.ssl.trustStore=/path/to/truststore \
-Djavax.net.ssl.trustStorePassword=trustStorePassword"
```

1. Create a snapshot. On a host running Solr Server, run the following command:

```
$ solrctl collection --create-snapshot <snapshotName> -c <collectionName>
```

For example, to create a snapshot for a collection named `tweets`:

```
$ solrctl collection --create-snapshot tweets-$(date +%Y%m%d%H%M) -c tweets
Successfully created snapshot with name tweets-201610191429 for collection tweets
```

2. If you are backing up the Solr collection to a remote cluster, prepare the snapshot for export. If you are backing up the Solr collection to the local cluster, skip this step.

```
$ solrctl collection --prepare-snapshot-export <snapshotName> -c <collectionName> -d
<destDir>
```

The destination HDFS directory path (specified by the `-d` option) must exist on the local cluster before you run this command. Make sure that the Solr superuser (`solr` by default) has permission to write to this directory.

For example:

```
$ hdfs dfs -mkdir -p /path/to/backup-staging/tweets-201610191429
$ hdfs dfs -chown :solr /path/to/backup-staging/tweets-201610191429
$ solrctl collection --prepare-snapshot-export tweets-201610191429 -c tweets \
-d /path/to/backup-staging/tweets-201610191429
```

3. Export the snapshot. This step uses the DistCp utility to back up the collection metadata as well as the corresponding index files. The destination directory must exist and be writable by the Solr superuser (`solr` by default). To export the snapshot to a remote cluster, run the following command:

```
$ solrctl collection --export-snapshot <snapshotName> -s <sourceDir> -d
<protocol>://<namenode>:<port>/<destDir>
```

For example:

- HDFS protocol:

```
$ solrctl collection --export-snapshot tweets-201610191429 -s
/path/to/backup-staging/tweets-201610191429 \
-d hdfs://nn01.example.com:8020/path/to/backups
```

- WebHDFS protocol:

```
$ solrctl collection --export-snapshot tweets-201610191429 -s
/path/to/backup-staging/tweets-201610191429 \
-d webhdfs://nn01.example.com:20101/path/to/backups
```

To export the snapshot to the local cluster, run the following command:

```
$ solrctl collection --export-snapshot <snapshotName> -c <collectionName> -d <destDir>
```

For example:

```
$ solrctl collection --export-snapshot tweets-201610191429 -c tweets -d /path/to/backups/
```

4. Delete the snapshot:

```
$ solrctl collection --delete-snapshot <snapshotName> -c <collectionName>
```

For example:

```
$ solrctl collection --delete-snapshot tweets-201610191429 -c tweets
```

Restoring a Solr Collection

Use the following procedure to restore a Solr collection. For more information on the commands used, see Cloudera Search Backup and Restore Command Reference on page 73.

If you are using a secure (Kerberos-enabled) cluster, specify your `jaas.conf` file by adding the following parameter to each command:

```
-jaas /path/to/jaas.conf
```

If TLS is enabled for the Solr service, specify the truststore and password by using the `ZKCLI_JVM_FLAGS` environment variable before you begin the procedure:

```
$ export ZKCLI_JVM_FLAGS="-Djavax.net.ssl.trustStore=/path/to/truststore \
-Djavax.net.ssl.trustStorePassword=trustStorePassword"
```

1. If you are restoring from a backup stored on a remote cluster, copy the backup from the remote cluster to the local cluster. If you are restoring from a local backup, skip this step.

   Run the following commands on the cluster to which you want to restore the collection:

```
$ hdfs dfs -mkdir -p /path/to/restore-staging
$ hadoop distcp <protocol>://<namenode>:<port>/path/to/backup /path/to/restore-staging
```

   For example:

   - HDFS protocol:

```
$ hadoop distcp hdfs://nn01.example.com:8020/path/to/backups/tweets-201610191429
/path/to/restore-staging
```

   - WebHDFS protocol:

```
$ hadoop distcp webhdfs://nn01.example.com:20101/path/to/backups/tweets-201610191429
/path/to/restore-staging
```

2. Start the restore procedure. Run the following command:

```
$ solrctl collection --restore <restoreCollectionName> -l <backupLocation> -b
<snapshotName> -i <requestId>
```

   Make sure that you use a unique <*requestID*> each time you run this command. For example:

```
$ solrctl collection --restore tweets -l /path/to/restore-staging -b tweets-201610191429
 -i restore-tweets
```

3. Monitor the status of the restore operation. Run the following command periodically:

```
$ solrctl collection --request-status <requestId>
```

   Look for `<str name="state">` in the output. For example:

```
$ solrctl collection --request-status restore-tweets
 <?xml version="1.0" encoding="UTF-8"?> <response> <lst name="responseHeader"> <int
name="status"> 0</int> <int name="QTime"> 1</int> </lst> \
<lst name="status"> <str name="state"> completed</str> <str name="msg"> found
restore-tweets in completed tasks</str> </lst> </response>
```

   The `state` parameter can be one of the following:

   - `running`: The restore operation is running.
   - `completed`: The restore operation is complete.
   - `failed`: The restore operation failed.
   - `notfound`: The specified <*requestID*> does not exist.

Cloudera Search Backup and Restore Command Reference

Use the following commands to create snapshots, back up, and restore Solr collections. If you are using a secure (Kerberos-enabled) cluster, specify your `jaas.conf` file by adding the following parameter to the command:

```
-jaas /path/to/jaas.conf
```

If TLS is enabled for the Solr service, specify the truststore and password by using the `ZKCLI_JVM_FLAGS` environment variable:

```
$ export ZKCLI_JVM_FLAGS="-Djavax.net.ssl.trustStore=/path/to/truststore \
-Djavax.net.ssl.trustStorePassword=trustStorePassword"
```

Create a snapshot

Command: `solrctl collection --create-snapshot <snapshotName> -c <collectionName>`

Description: Creates a named snapshot for the specified collection.

Delete a snapshot

Command: `solrctl collection --delete-snapshot <snapshotName> -c <collectionName>`

Description: Deletes the specified snapshot for the specified collection.

Describe a snapshot

Command: `solrctl collection --describe-snapshot <snapshotName> -c <collectionName>`

Description: Provides detailed information about a snapshot for the specified collection.

List all snapshots

Command: `solrctl collection --list-snapshots <collectionName>`

Description: Lists all snapshots for the specified collection.

Prepare snapshot for export to a remote cluster

Command: `solrctl collection --prepare-snapshot-export <snapshotName> -c <collectionName> -d <destDir>`

Description: Prepares the snapshot for export to a remote cluster. If you are exporting the snapshot to the local cluster, you do not need to run this command. This command generates collection metadata as well as information about the Lucene index files corresponding to the snapshot.

The destination HDFS directory path (specified by the `-d` option) must exist on the local cluster before you run this command. Make sure that the Solr superuser (`solr` by default) has permission to write to this directory.

If you are running the snapshot export command on a remote cluster, specify the HDFS protocol (such as WebHDFS or HFTP) to be used for accessing the Lucene index files corresponding to the snapshot on the source cluster. For more information about the Hadoop DistCP utility, see Copying Data Between Two Clusters Using Distcp. This configuration is driven by the `-p` option which expects a fully qualified URI for the root filesystem on the source cluster, for example `webhdfs://namenode.example.com:20101/`.

Export snapshot to local cluster

Command: `solrctl collection --export-snapshot <snapshotName> -c <collectionName> -d <destDir>`

Description: Creates a backup copy of the Solr collection metadata as well as the associated Lucene index files at the specified location. The `-d` configuration option specifies the directory path where this backup copy is be created. This directory must exist before exporting the snapshot, and the Solr superuser must be able to write to it.

Export snapshot to remote cluster

Command: `solrctl collection --export-snapshot <snapshotName> -s <sourceDir> -d <destDir>`

Description: Creates a backup copy of the Solr collection snapshot, which includes collection metadata as well as Lucene index files at the specified location. The -d configuration option specifies the directory path where this backup copy is be created.

Make sure that you prepare the snapshot for export before exporting it to a remote cluster.

You can run this command on either the source or destination cluster, depending on your environment and the DistCp utility requirements. For more information, see Copying Data Between Two Clusters Using Distcp. If the destination cluster does not have the `solrctl` utility, you must run the command on the source cluster. The exported snapshot state can then be copied using standard tools, such as DistCp.

The source and destination directory paths (specified by the `-s` and `-d` options, respectively) must be specified relative to the cluster from which you are running the command. Directories on the local cluster are formatted as `/path/to/dir`, and directories on the remote cluster are formatted as `<protocol>://<namenode>:<port>/path/to/dir`. For example:

- Local path: `/solr-backup/tweets-2016-10-19`
- Remote HDFS path: `hdfs://nn01.example.com:8020/solr-backup/tweets-2016-10-19`
- Remote WebHDFS path: `webhdfs://nn01.example.com:20101/solr-backup/tweets-2016-10-19`

The source directory (specified by the `-s` option) is the directory containing the output of the `solrctl collection --prepare-snapshot-export` command. The destination directory (specified by the `-d` option) must exit on the destination cluster before running this command.

If your cluster is secured (Kerberos-enabled), initialize your Kerberos credentials by using `kinit` before executing this command.

Restore from a local snapshot

Command: `solrctl collection --restore <restoreCollectionName> -l <backupLocation> -b <snapshotName> -i <requestId>`

Description: Restores the state of an earlier created backup as a new Solr collection. Run this command on the cluster on which you want to restore the backup.

The `-l` configuration option specifies the local HDFS directory where the backup is stored. If the backup is stored on a remote cluster, you must copy it to the local cluster before restoring it. The Solr superuser (`solr` by default) must have permission to read from this directory.

The `-b` configuration option specifies the name of the backup to be restored.

Because the restore operation can take a long time to complete depending on the size of the exported snapshot, it is run asynchronously. The `-i` configuration parameter specifies a unique identifier for tracking operation. For more information, see Check the status of an operation on page 75.

The optional `-a` configuration option enables the `autoAddReplicas` feature for the new Solr collection.

The optional `-c` configuration option specifies the `configName` for the new Solr collection. If this option is not specified, the `configName` of the original collection at the time of backup is used. If the specified `configName` does not exist, the restore operation creates a new configuration from the backup.

The optional `-r` configuration option specifies the replication factor for the new Solr collection. If this option is not specified, the replication factor of the original collection at the time of backup is used.

The optional `-m` configuration option specifies the maximum number of replicas (`maxShardsPerNode`) to create on each Solr Server. If this option is not specified, the `maxShardsPerNode` configuration of the original collection at the time of backup is used.

If your cluster is secured (Kerberos-enabled), initialize your Kerberos credentials using `kinit` before running this command.

Check the status of an operation

Command: `solrctl collection --request-status <requestId>`

Description: Displays the status of the specified operation. The status can be one of the following:

- `running`: The restore operation is running.
- `completed`: The restore operation is complete.
- `failed`: The restore operation failed.
- `notfound`: The specified *requestID* is not found.

If your cluster is secured (Kerberos-enabled), initialize your Kerberos credentials (using `kinit`) before running this command.

Schemaless Mode Overview and Best Practices

Schemaless mode removes the need to design a schema before beginning to use Search. This can help you begin using Search more quickly, but schemaless mode is typically less efficient and effective than using a deliberately designed schema.

Note: Cloudera recommends pre-defining a schema before moving to production.

With the default non-schemaless mode, you create a schema by writing a `schema.xml` file before loading data into Solr so it can be used by Cloudera Search. You typically write a different schema definition for each type of data being ingested, because the different data types usually have different field names and values. Writing a custom schema is done by examining the data to be imported so its structure can be understood, and then creating a schema that accommodates that data. For example, emails might have a field for recipients and log files might have a field for IP addresses for machines reporting errors. Conversely, emails typically do not have an IP address field and log files typically do not have recipients. Therefore, the schema you use to import emails is different from the schema you use to import log files.

Cloudera Search offers schemaless mode to help facilitate sample deployments without the need to pre-define a schema. While schemaless is not suitable for production environments, it can help demonstrate the functionality and features of Cloudera Search. Schemaless mode operates based on three principles:

1. The schema is automatically updated using an API. When not using schemaless mode, users manually modify the `schema.xml` file or use the Schema API.
2. As data is ingested, it is analyzed and a guess is made about the type of data in the field. Supported types include Boolean, Integer, Long, Float, Double, Date, and Text.
3. When a new field is encountered, the schema is automatically updated using the API. The update is based on the guess about the type of data in the field.

For example, if schemaless encounters a field that contains "6.022", this would be determined to be type Float, whereas "Mon May 04 09:51:52 CDT 2009" would be determined to be type Date.

By combining these techniques, Schemaless:

1. Starts without a populated schema.
2. Intakes and analyzes data.
3. Modifies the schema based on guesses about the data.

4. Ingests the data so it can be searched based on the schema updates.

To generate a configuration for use in Schemaless mode, use `solrctl instancedir --generate path -schemaless`. Then, create the instancedir and collection as with non-schemaless mode. For more information, see solrctl Reference on page 23.

Best Practices

User Defined Schemas Recommended for Production Use Cases

Schemaless Solr is useful for getting started quickly and for understanding the underlying structure of the data you want to search. However, Schemaless Solr is not recommended for production use cases. Because the schema is automatically generated, a mistake like misspelling the name of the field alters the schema, rather than producing an error. The mistake may not be caught until much later and once caught, may require re-indexing to fix. Also, an unexpected input format may cause the type guessing to pick a field type that is incompatible with data that is subsequently ingested, preventing further ingestion until the incompatibility is manually addressed. Such a case is rare, but could occur. For example, if the first instance of a field was an integer, such as '9', but subsequent entries were text such as '10 Spring Street', the schema would make it impossible to properly ingest those subsequent entries. Therefore, Schemaless Solr may be useful for deciding on a schema during the exploratory stage of development, but Cloudera recommends defining the schema in the traditional way before moving to production.

Give each Collection its own unique Instancedir

Solr supports using the same instancedir for multiple collections. In schemaless mode, automatic schema field additions actually change the underlying instancedir. Thus, if two collections are using the same instancedir, schema field additions meant for one collection will actually affect the other one as well. Therefore, Cloudera recommended that each collection have its own instancedir.

Using Search through a Proxy for High Availability

Using a proxy server to relay requests to and from the Solr service can help meet availability requirements in production clusters serving many users.

A proxy server works a set of servers that is organized into a server group. A proxy server does not necessarily work with all servers in a deployment.

Overview of Proxy Usage and Load Balancing for Search

Configuring a proxy server to relay requests to and from the Solr service has the following advantages:

- Applications connect to a single well-known host and port, rather than keeping track of the hosts where the Solr service is running. This is especially useful for non-Java Solr clients such as web browsers or command-line tools such as curl.

  > **Note:** Solr Java client (solrj) can introspect Zookeeper metadata to automatically locate the individual Solr servers, so load-balancing proxy support is not necessary.

- If any host running the Solr service becomes unavailable, application connection requests still succeed because you always connect to the proxy server rather than a specific host running the Solr server.
- Users can configure an SSL terminating proxy for Solr to secure the data exchanged with the external clients without requiring SSL configuration for the Solr cluster itself. This is relevant only if the Solr cluster is deployed on a trusted network and needs to communicate with clients that may not be on the same network. Many of the advantages of SSL offloading are described in SSL Offloading, Encryption, and Certificates with NGINX.
- The "coordinator host" for each Search query potentially requires more memory and CPU cycles than the other hosts that process the query. The proxy server can issue queries using round-robin scheduling, so that each connection uses a different coordinator host. This load-balancing technique lets the hosts running the Solr service share this additional work, rather than concentrating it on a single machine.

The following setup steps are a general outline that apply to any load-balancing proxy software.

1. Download the load-balancing proxy software. It should only need to be installed and configured on a single host.
2. Configure the software, typically by editing a configuration file. Set up a port on which the load balancer listens to relay Search requests back and forth.
3. Specify the host and port settings for each Solr service host. These are the hosts that the load balancer chooses from when relaying each query. In most cases, use 8983, the default query and update port.
4. Run the load-balancing proxy server, pointing it at the configuration file that you set up.

Special Proxy Considerations for Clusters Using Kerberos

In a cluster using Kerberos, applications check host credentials to verify that the host they are connecting to is the same one that is actually processing the request, to prevent man-in-the-middle attacks. To clarify that the load-balancing proxy server is legitimate, perform these extra Kerberos setup steps:

1. This section assumes you are starting with a Kerberos-enabled cluster. See Solr Authentication on page 81 for instructions for setting up Search with Kerberos. See the *CDH Security Guide* for general steps to set up Kerberos: CDH 5 instructions.
2. Choose the host you will use for the proxy server. Based on the Kerberos setup procedure, it should already have an entry `solr/proxy_host@realm` in its keytab. If not, go back over the initial Kerberos configuration steps to the keytab on each host running solr as described in Solr Authentication on page 81.
3. Copy the keytab file from the proxy host to all other hosts in the cluster that run the `solr` daemon. (For optimal performance, `solr` should be running on all DataNodes in the cluster.) Put the keytab file in a secure location on each of these other hosts.
4. For each solr node, merge the existing keytab with the proxy's keytab using `ktutil`, producing a new keytab file. For example:

```
$ ktutil
ktutil: read_kt proxy.keytab
ktutil: read_kt solr.keytab
ktutil: write_kt proxy_Search.keytab
ktutil: quit
```

5. Make sure that the `Search` user has permission to read this merged keytab file.
6. For every host running Solr daemon, edit the `SOLR_AUTHENTICATION_KERBEROS_PRINCIPAL` property in `/etc/default/solr` file to set the value to *. The value should appear as:

```
SOLR_AUTHENTICATION_KERBEROS_PRINCIPAL=*
```

7. Restart the Search service to make the changes take effect.

Configuring Dependent Services

Other services that use Search must also be configured to use the load balancer. For example, Hue may need reconfiguration. To reconfigure dependent services, ensure that the service uses a URL constructed of the load balancer hostname and port number when referring to Solr service. For example, in case of Hue, update `hue.ini` file to set `solr_url` parameter to a url referring load balancer. URL referring load balancers are typically of the form `http://<load-balancer-host>:<port>/solr`. For example, the value might appear as:

```
solr_url=http://load-balancer.example.com:1518/solr
```

Migrating Solr Replicas

When you replace a host, migrating replicas on that host to the new host, instead of depending on failure recovery, can help ensure optimal performance.

Where possible, the Solr service routes requests to the proper host. Both ADDREPLICA and DELETEREPLICA calls can be sent to any host in the cluster.

- For adding replicas, the node parameter ensures the new replica is created on the intended host. If no host is specified, Solr selects a host with relatively fewer replicas.
- For deleting replicas, the request is routed to the host that hosts the replica to be deleted.

Adding replicas can be resource intensive. For best results, add replicas when the system is not under heavy load. For example, do not add additional replicas when heavy indexing is occurring or when MapReduceIndexerTool jobs are running.

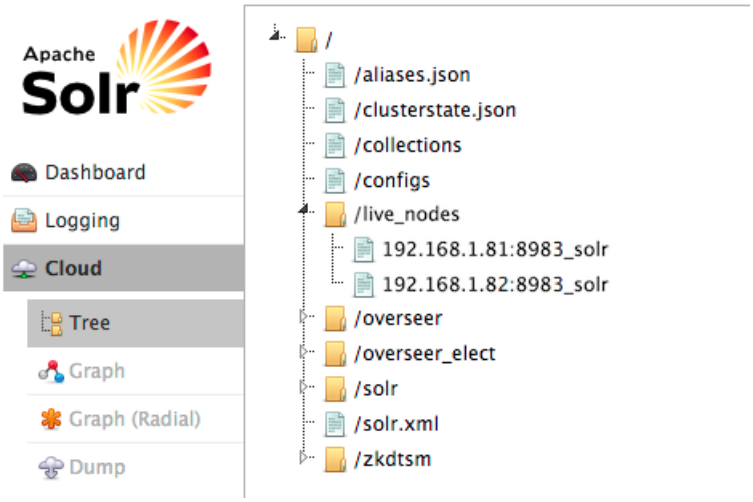
Cloudera recommends using API calls to create and unload cores. Do not use the Cloudera Manager Admin Console or the Solr Admin UI for these tasks.

This procedure uses the following names:

- Host names:
  - `origin` at the IP address 192.168.1.81:8983_solr.
  - `destination` at the IP address 192.168.1.82:8983_solr.
- Collection name `email`
- Replicas:
  - The original replica `email_shard1_replica1`, which is on `origin`.
  - The new replica `email_shard1_replica2`, which will be on `destination`.

To migrate a replica to a new host

1. (Optional) If you want to add a replica to a particular node, review the contents of the `live_nodes` directory on ZooKeeper to find all nodes available to host replicas. Open the Solr Administration User interface, click **Cloud**, click **Tree**, and expand **live_nodes**. The Solr Administration User Interface, including **live_nodes**, might appear as follows:



> **Note:** Information about Solr nodes can also be found in `clusterstate.json`, but that file only lists nodes currently hosting replicas. Nodes running Solr but not currently hosting replicas are not listed in `clusterstate.json`.

2. Add the new replica on `destination` server using the ADDREPLICA API.

```
http://192.168.1.81:8983/solr/admin/collections?action=ADDREPLICA&collection=email&shard=shard1&node=192.168.1.82:8983_solr
```

3. Verify that the replica creation succeeds and moves from recovery state to **ACTIVE**. You can check the replica status in the Cloud view, which can be found at a URL similar to: `http://192.168.1.82:8983/solr/#/~cloud`.

Note: Do not delete the original replica until the new one is in the **ACTIVE** state. When the newly added replica is listed as **ACTIVE**, the index has been fully replicated to the newly added replica. The total time to replicate an index varies according to factors such as network bandwidth and the size of the index. Replication times on the scale of hours are not uncommon and do not necessarily indicate a problem.

You can use the `details` command to get an XML document that contains information about replication progress. Use `curl` or a browser to access a URI similar to:

```
http://192.168.1.82:8983/solr/email_shard1_replica2/replication?command=details
```

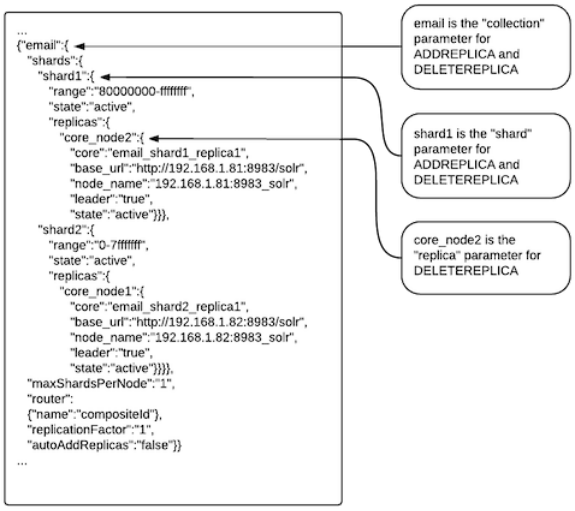
Accessing this URI returns an XML document that contains content about replication progress. A snippet of the XML content might appear as follows:

```
...
<str name="numFilesDownloaded">126</str>
<str name="replication StartTime">Tue Jan 21 14:34:43 PST 2014</str>
<str name="timeElapsed">457s</str>
<str name="currentFile">4xt_Lucene41_0.pos</str>
<str name="currentFileSize">975.17 MB</str>
<str name="currentFileSizeDownloaded">545 MB</str>
<str name="currentFileSizePercent">55.0</str>
<str name="bytesDownloaded">8.16 GB</str>
<str name="totalPercent">73.0</str>
<str name="timeRemaining">166s</str>
<str name="downloadSpeed">18.29 MB</str>
...
```

4. Use the CLUSTERSTATUS API to retrieve information about the cluster, including current cluster status:

```
http://192.168.1.81:8983/solr/admin/collections?action=clusterstatus&wt=json&indent=true
```

Review the returned information to find the correct replica to remove. An example of the JSON file might appear as follows:



5. Delete the old replica on `origin` server using the DELETEREPLICA API:

```
http://192.168.1.81:8983/solr/admin/collections?action=DELETEREPLICA&collection=email&shard=shard1&replica=core_node2
```

The DELTEREPLICA call removes the `datadir`.

Using Custom JAR Files with Search

Search for CDH supports custom plug-in code. You load classes into JAR files and then configure Search to find these files. To correctly deploy custom JARs, ensure that:

- Custom JARs are pushed to the same location on all hosts in your cluster that are hosting Cloudera Search (Solr Service).
- Supporting configuration files direct Search to find the custom JAR files.
- Any required configuration files such as `schema.xml` or `solrconfig.xml` reference the custom JAR code.

The following procedure describes how to use custom JARs. Some cases may not require completion of every step. For example, indexer tools that support passing JARs as arguments may not require modifying `xml` files. However, completing all configuration steps helps ensure the custom JARs are used correctly in all cases.

Note: Search for CDH supports custom plug-in code, but it does not support enabling custom JARs using the Blob Store API and special config API commands. Apache Solr 5.3 implements this functionality so that JARs can be loaded to a special system-level collection and dynamically loaded at run time.

1. Place your custom JAR in the same location on all hosts in your cluster.
2. For all collections where custom JARs will be used, modify `solrconfig.xml` to include references to the new JAR files.

   These directives can include explicit or relative references and can use wildcards. In the `solrconfig.xml` file, add `<lib>` directives to indicate the JAR file locations or `<path>` directives for specific jar files:

```
<lib path="/usr/lib/solr/lib/MyCustom.jar" />
```

   or

```
<lib dir="/usr/lib/solr/lib" />
```

   or

```
<lib dir="../../../myProject/lib" regex=".*\.jar" />
```

3. For all collections in which custom JARs will be used, reference custom JAR code in the appropriate Solr configuration file. The two configuration files that most commonly reference code in custom JARs are `solrconfig.xml` and `schema.xml`.
4. For all collections in which custom JARs will be used, use `solrctl` to update ZooKeeper's copies of configuration files such as `solrconfig.xml` and `schema.xml`:

```
solrctl instancedir --update name path
```

   - *`name`* specifies the *instancedir* associated with the collection using `solrctl instancedir --create`.
   - *`path`* specifies the directory containing the collection's configuration files.

   For example:

```
solrctl instancedir --update collection1 $HOME/solr_configs
```

5. For all collections in which custom JARs will be used, use RELOAD to refresh information:

```
http://<hostname>:<port>/solr/admin/collections?action=RELOAD&name=collectionname
```

   For example:

```
http://example.com:8983/solr/admin/collections?action=RELOAD&name=collection1
```

When the RELOAD command is issued to any host that hosts a collection, that host sends subcommands to all replicas in the collection. All relevant hosts refresh their information, so this command must be issued once per collection.

6. Ensure that the class path includes the location of the custom JAR file:
   a. For example, if you store the custom JAR file in `/opt/myProject/lib/`, add that path as a line to the `~/.profile` for the Solr user.
   b. Restart the Solr service to reload the PATH variable.
   c. Repeat this process of updating the PATH variable for all hosts.

The system is now configured to find custom JAR files. Some command-line tools included with Cloudera Search support specifying JAR files. For example, when using `MapReduceIndexerTool`, use the `--libjars` option to specify JAR files to use. Tools that support specifying custom JARs include:

- MapReduceIndexerTool
- Lily HBase Indexer
- HDFSFindTool
- CrunchIndexerTool
- Flume indexing

Solr Authentication

This section describes how to configure Solr to enable authentication.

When authentication is enabled, only specified hosts and users can connect to Solr. Authentication also verifies that clients connect to legitimate servers. This feature prevents spoofing such as impersonation and man-in-the-middle attacks. Search supports Kerberos and LDAP authentication.

Cloudera Search supports a variety of combinations of authentication protocols:

Table 5: Authentication Protocol Combinations

Solr Authentication	Use Case
No authentication	Insecure cluster
Kerberos only	The Hadoop cluster has kerberos turned on and every user (or client) connecting to Solr has a Kerberos principal.
Kerberos and LDAP	The Hadoop cluster has kerberos turned on. External Solr users (or clients) don't have Kerberos principal but do have an identity in the LDAP server.

Once you are finished setting up authentication, configure Sentry authorization. Authorization involves specifying which resources can be accessed by particular users when they connect through Search. See Configuring Sentry Authorization for Solr on page 87 for details.

Enabling Kerberos Authentication for Solr

Solr supports Kerberos authentication. All necessary packages are installed when you install Search. To enable Kerberos, create principals and keytabs and then modify default configurations.

The following instructions only apply to configuring Kerberos in an unmanaged environment. Kerberos configuration is automatically handled by Cloudera Manager if you are using in a Cloudera Manager environment.

To create principals and keytabs

Repeat this process on all Solr server hosts.

1. Create a Solr service user principal using the syntax: `solr/<fully.qualified.domain.name>@<YOUR-REALM>`. This principal is used to authenticate with the Hadoop cluster. where: `fully.qualified.domain.name` is the host where the Solr server is running `YOUR-REALM` is the name of your Kerberos realm.

   ```
   $ kadmin
   kadmin: addprinc -randkey solr/fully.qualified.domain.name@YOUR-REALM.COM
   ```

2. Create a HTTP service user principal using the syntax: `HTTP/<fully.qualified.domain.name>@<YOUR-REALM>`. This principal is used to authenticate user requests coming to the Solr web-services. where: `fully.qualified.domain.name` is the host where the Solr server is running `YOUR-REALM` is the name of your Kerberos realm.

   ```
   kadmin: addprinc -randkey HTTP/fully.qualified.domain.name@YOUR-REALM.COM
   ```

   > **Note:**
   >
   > The `HTTP/` component of the HTTP service user principal must be upper case as shown in the syntax and example above.

3. Create keytab files with both principals.

   ```
   kadmin: xst -norandkey -k solr.keytab solr/fully.qualified.domain.name \
   HTTP/fully.qualified.domain.name
   ```

4. Test that credentials in the merged keytab file work. For example:

   ```
   $ klist -e -k -t solr.keytab
   ```

5. Copy the `solr.keytab` file to the Solr configuration directory. The owner of the `solr.keytab` file should be the `solr` user and the file should have owner-only read permissions.

To modify default configurations

Repeat this process on all Solr server hosts.

1. Ensure that the following properties appear in `/etc/default/solr` or `/opt/cloudera/parcels/CDH-*/etc/default/solr` and that they are uncommented. Modify these properties to match your environment. The relevant properties to be uncommented and modified are:

   ```
   SOLR_AUTHENTICATION_TYPE=kerberos
   SOLR_AUTHENTICATION_SIMPLE_ALLOW_ANON=true
   SOLR_AUTHENTICATION_KERBEROS_KEYTAB=/etc/solr/conf/solr.keytab
   SOLR_AUTHENTICATION_KERBEROS_PRINCIPAL=HTTP/localhost@LOCALHOST
   SOLR_AUTHENTICATION_KERBEROS_NAME_RULES=DEFAULT
   SOLR_AUTHENTICATION_JAAS_CONF=/etc/solr/conf/jaas.conf
   ```

   > **Note:** Modify the values for these properties to match your environment. For example, the `SOLR_AUTHENTICATION_KERBEROS_PRINCIPAL=HTTP/localhost@LOCALHOST` must include the principal instance and Kerberos realm for your environment. That is often different from `localhost@LOCALHOST`.

2. Set `hadoop.security.auth_to_local` to match the value specified by `SOLR_AUTHENTICATION_KERBEROS_NAME_RULES` in `/etc/default/solr` or `/opt/cloudera/parcels/CDH-*/etc/default/solr`.

   > **Note:** For information on how to configure the rules, see Configuring the Mapping from Kerberos Principals to Short Names. For additional information on using Solr with HDFS, see Configuring Solr for Use with HDFS.

3. If using applications that use the `solrj` library, set up the Java Authentication and Authorization Service (JAAS).

   a. Create a `jaas.conf` file in the Solr configuration directory containing the following settings. This file and its location must match the `SOLR_AUTHENTICATION_JAAS_CONF` value. Make sure that you substitute a value for `principal` that matches your particular environment.

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  useTicketCache=false
  keyTab="/etc/solr/conf/solr.keytab"
  principal="solr/fully.qualified.domain.name@<YOUR-REALM>";
};
```

Enabling LDAP Authentication for Solr

Solr supports LDAP authentication for external Solr client including:

- Command-line tools
- curl
- Web browsers
- Solr Java clients

In some cases, Solr does not support LDAP authentication. Use Kerberos authentication instead in these cases. Solr does not support LDAP authentication with:

- Search indexing components including the MapReduce indexer, Lily HBase indexer, or Flume.
- Solr internal requests such as those for replication or querying.
- Hadoop delegation token management requests such as `GETDELEGATIONTOKEN` or `RENEWDELEGATIONTOKEN`.

Configuring LDAP Authentication for Solr using Cloudera Manager

You can configure LDAP-based authentication using Cloudera Manager at the Solr service level.

1. Go to the **Solr** service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Solr**
4. Select **Category** > **Security**
5. Select **Enable LDAP**.
6. Enter the LDAP URI in the **LDAP URI** property.
7. Configure only one of following mutually exclusive parameters:

   - **LDAP BaseDN**: Replaces the username with a "distinguished name" (DN) of the form: `uid=userid,ldap_baseDN`. Typically used for OpenLDAP server installation.

   -OR-

   - **LDAP Domain**: Replaces the username with a string `username@ldap_domain`. Typically used for Active Directory server installation.

Configuring LDAP Authentication for Solr Using the Command Line

To enable LDAP authentication using the command line, configure the following environment variables in `/etc/default/solr`:

```
SOLR_AUTHENTICATION_HTTP_SCHEMES=Negotiate,Basic
SOLR_AUTHENTICATION_HTTP_DELEGATION_MGMT_SCHEMES=Negotiate
SOLR_AUTHENTICATION_HTTP_BASIC_HANDLER=ldap
SOLR_AUTHENTICATION_HTTP_NEGOTIATE_HANDLER=kerberos
SOLR_AUTHENTICATION_LDAP_PROVIDER_URL=ldap://www.example.com

# Specify value for only one of SOLR_AUTHENTICATION_LDAP_BASE_DN or
SOLR_AUTHENTICATION_LDAP_BIND_DOMAIN property.
SOLR_AUTHENTICATION_LDAP_BASE_DN=ou=Users,dc=example,dc=com
# SOLR_AUTHENTICATION_LDAP_BIND_DOMAIN=
# Required when using 'Start TLS' extension
# SOLR_AUTHENTICATION_LDAP_ENABLE_START_TLS=false
```

Securing LDAP Connections

You can secure communications using LDAP-based encryption.

To avoid sending credentials over the wire in clear-text, you must configure a secure connection between both the client and Solr, and between Solr and the LDAP server. The secure connection could use SSL or TLS.

Secure LDAP connections through SSL:

For SSL-enabled LDAP connections, specify a prefix of `ldaps://` instead of `ldap://`. Also, the default port for SSL-enabled LDAP connections is 636 instead of 389.

Secure LDAP connections through TLS:

TLS, the successor to the SSL protocol, is supported by most modern LDAP servers. Unlike SSL connections, TLS connections can be made on the same server port as non-TLS connections. You can enable xxx using Cloudera Manager.

1. Go to the **Solr** service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Solr**
4. Select **Category** > **Security**
5. Select **Enable LDAP TLS**.
6. Import the LDAP server security certificate in the Solr Trust Store file:
    a. Enter the location for the Solr Trust Store File in **Solr TLS/SSL Certificate Trust Store File**.
    b. Enter the password for the Solr Trust Store File in **Solr TLS/SSL Certificate Trust Store Password**.

LDAP Client Configuration

Some HTTP clients such as curl or the Apache Http Java client must be configured to use a particular scheme. For example:

- curl tool supports using Kerberos or username/password authentication. Kerberos is activated using the `--negotiate` flag and username/password based authentication is activated using the `--basic` and `-u` flags.
- Apache HttpClient library can be configured to use specific authentication scheme. For more information, see the HTTP authentication chapter of Apache's HttpClient Tutorial.

Typically, web browsers automatically choose a preferred authentication scheme. For more information, see the HTTP authentication topic in The Chromium Projects.

To use LDAP authentication with Solr Java clients, `HttpClientConfigurer` needs to configured for Solr. This can either be done programmatically or using Java system properties.

For example, programmatic initialization might appear as:

SampleSolrClient.java

```
import org.apache.solr.client.solrj.impl.HttpClientUtil;
import org.apache.solr.client.solrj.impl.PreemptiveBasicAuthConfigurer;
import org.apache.solr.common.params.ModifiableSolrParams;

/**
 * This method initializes the Solr client to use LDAP authentication
 * This configuration is applicable to all Solr clients.
 * @param ldapUserName LDAP user name
 * @param ldapPassword  LDAP user password
 */
public static void initialize(String ldapUserName, String ldapPassword) {
  HttpClientUtil.setConfigurer(new PreemptiveBasicAuthConfigurer());
  ModifiableSolrParams params = new ModifiableSolrParams();
  params.set(HttpClientUtil.PROP_BASIC_AUTH_USER, ldapUserName);
  params.set(HttpClientUtil.PROP_BASIC_AUTH_PASS, ldapPassword);
  // Configure the JVM default parameters.
  PreemptiveBasicAuthConfigurer.setDefaultSolrParams(params);
}
```

For configuration using system properties, configure the following system properties:

Table 6: System properties configuration for LDAP authentication

System property	Description
`solr.httpclient.configurer`	Fully qualified classname of `HttpClientConfigurer` implementation. For example, `org.apache.solr.client.solrj.impl.PreemptiveBasicAuthConfigurer`.
`solr.httpclient.config`	Http client configuration properties file path. For example, `ldap-credentials.properties`.

For example, the entry in `ldap-credentials.properties` might appear as:

```
ldap-credentials.properties

httpBasicAuthUser=user1
httpBasicAuthPassword=passwd
```

Using Kerberos with Solr

The process of enabling Solr clients to authenticate with a secure Solr is specific to the client. This section demonstrates:

- Using Kerberos and curl
- Using solrctl
- Configuring SolrJ Library Usage
- This enables technologies including:
  - Command line solutions
  - Java applications
  - The MapReduceIndexerTool
- Configuring Flume Morphline Solr Sink Usage

Secure Solr requires that the CDH components that it interacts with are also secure. Secure Solr interacts with HDFS, ZooKeeper and optionally HBase, MapReduce, and Flume.

Using Kerberos and curl

You can use Kerberos authentication with clients such as `curl`. To use `curl`, begin by acquiring valid Kerberos credentials and then run the desired command. For example, you might use commands similar to the following:

```
$ kinit -kt username.keytab username
$ curl --negotiate -u foo:bar http://solrserver:8983/solr/
```

Note: Depending on the tool used to connect, additional arguments may be required. For example, with `curl`, `--negotiate` and `-u` are required. The username and password specified with `-u` is not actually checked because Kerberos is used. As a result, any value such as foo:bar or even just : is acceptable. While any value can be provided for `-u`, note that the option is required. Omitting `-u` results in a 401 Unauthorized error, even though the `-u` value is not actually used.

Using solrctl

If you are using `solrctl` to manage your deployment in an environment that requires Kerberos authentication, you must have valid Kerberos credentials, which you can get using `kinit`. For more information on `solrctl`, see solrctl Reference on page 23

Configuring SolrJ Library Usage

If using applications that use the solrj library, begin by establishing a Java Authentication and Authorization Service (JAAS) configuration file.

Create a JAAS file:

- If you have already used `kinit` to get credentials, you can have the client use those credentials. In such a case, modify your `jaas-client.conf` file to appear as follows:

```
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=false
 useTicketCache=true
 principal="user/fully.qualified.domain.name@<YOUR-REALM>";
 };
```

  where `user/fully.qualified.domain.name@<YOUR-REALM>` is replaced with your credentials.

- You want the client application to authenticate using a keytab you specify:

```
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 keyTab="/path/to/keytab/user.keytab"
 storeKey=true
 useTicketCache=false
 principal="user/fully.qualified.domain.name@<YOUR-REALM>";
};
```

  where `/path/to/keytab/user.keytab` is the keytab file you want to use and `user/fully.qualified.domain.name@<YOUR-REALM>` is the principal in that keytab you want to use.

Use the JAAS file to enable solutions:

- **Command line solutions**

  Set the property when invoking the program. For example, if you were using a jar, you might use:

```
java -Djava.security.auth.login.config=/home/user/jaas-client.conf -jar app.jar
```

- **Java applications**

  Set the Java system property `java.security.auth.login.config`. For example, if the JAAS configuration file is located on the filesystem as `/home/user/jaas-client.conf`. The Java system property `java.security.auth.login.config` must be set to point to this file. Setting a Java system property can be done programmatically, for example using a call such as:

```
System.setProperty("java.security.auth.login.config", "/home/user/jaas-client.conf");
```

- **The MapReduceIndexerTool**

  The MapReduceIndexerTool uses SolrJ to pass the JAAS configuration file. Using the MapReduceIndexerTool in a secure environment requires the use of the `HADOOP_OPTS` variable to specify the JAAS configuration file. For example, you might issue a command such as the following:

```
HADOOP_OPTS="-Djava.security.auth.login.config=/home/user/jaas.conf" \
hadoop jar MapReduceIndexerTool
```

- **Configuring the hbase-indexer CLI**

  Certain hbase-indexer CLI commands such as `replication-status` attempt to read ZooKeeper hosts owned by HBase. To successfully use these commands in Solr in a secure environment, specify a JAAS configuration file

with the HBase principal in the HBASE_INDEXER_OPTS environment variable. For example, you might issue a command such as the following:

```
HBASE_INDEXER_OPTS="-Djava.security.auth.login.config=/home/user/hbase-jaas.conf" \
hbase-indexer replication-status
```

Configuring Flume Morphline Solr Sink Usage

Repeat this process on all Flume hosts:

1. If you have not created a keytab file, do so now at /etc/flume-ng/conf/flume.keytab. This file should contain the service principal flume/<fully.qualified.domain.name>@<YOUR-REALM>. See Flume Authentication for more information.
2. Create a JAAS configuration file for flume at /etc/flume-ng/conf/jaas-client.conf. The file should appear as follows:

```
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 useTicketCache=false
 keyTab="/etc/flume-ng/conf/flume.keytab"
 principal="flume/<fully.qualified.domain.name>@<YOUR-REALM>";
};
```

3. Add the flume JAAS configuration to the JAVA_OPTS in /etc/flume-ng/conf/flume-env.sh. For example, you might change:

```
JAVA_OPTS="-Xmx500m"
```

to:

```
JAVA_OPTS="-Xmx500m -Djava.security.auth.login.config=/etc/flume-ng/conf/jaas-client.conf"
```

Configuring Sentry Authorization for Solr

Sentry enables role-based, fine-grained authorization for Cloudera Search. Sentry can apply a range of restrictions to various tasks, such as accessing data, managing configurations through config objects, or creating collections. Restrictions are consistently applied, regardless of the way users attempt to complete actions. For example, restricting access to data in a collection restricts that access whether queries come from the command line, a browser, Hue, or through the admin console. For additional information on Sentry, see Authorization With Apache Sentry.

- You can use either Cloudera Manager or the following command-line instructions to complete this configuration.
- This information applies specifically to CDH 5.9.x. If you use an earlier version of CDH, see the documentation for that version located at Cloudera Documentation.

Follow the instructions below to configure Sentry for Solr. Sentry and Solr are included in the Search installation.

Note: Sentry for Search depends on Kerberos authentication. For additional information on using Kerberos with Search, see Solr Authentication on page 81.

Using Roles and Privileges with Sentry

Sentry uses a role-based privilege model. A role is a set of rules for accessing a given Solr collection or Solr config. Access to each collection is governed by three privileges: Query, Update, and *. The wildcard (*) indicates all privileges. In contrast, access to each config is governed by a single privilege *, meaning all privileges. Specifying * as the name of the config or collection grants the specified privilege to all instances of configs or collections, respectively. The

following sample syntax applies to both native Sentry privileges and file-based privileges, though native Sentry privileges are set through `solrctl sentry` commands as shown in Using Solr with the Sentry Service on page 89 and file-based privileges are set through policy files as shown in Using Solr with a Policy File on page 90.

- A rule for the `Query` privilege on collection called logs would be formulated as follows:

```
collection=logs->action=Query
```

- A rule for the * privilege, meaning all privileges, on the config called `myConfig` would be formulated as follows:

```
config=myConfig->action=*
```

No action implies * and * is the only valid action. Because `config` objects only support *, the following `config` privilege is invalid:

```
config=myConfig->action=Update
```

- A rule granting all configs the Query privilege would be formulated as follows:

```
config=*->action=Query
```

- For example, granting all configs the Query privilege would be formulated as follows:

```
config=*->action=Query
```

`config` objects cannot be combined with `collection` objects in a single privilege. For example, the following combinations are illegal:

```
config=myConfig->collection=myCollection->action=*
```

```
collection=myCollection->config=myConfig
```

You may specify these privileges separately. For example:

```
myRole = collection=myCollection->action=QUERY, config=myConfig->action=*
```

A role can contain multiple such rules, separated by commas. For example the `engineer_role` might contain the `Query` privilege for `hive_logs` and `hbase_logs` collections, and the `Update` privilege for the `current_bugs` collection. You would specify this as follows:

```
engineer_role = collection=hive_logs->action=Query, collection=hbase_logs->action=Query,
 collection=current_bugs->action=Update
```

Using Users and Groups with Sentry

- A user is an entity that is permitted by the Kerberos authentication system to access the Search service.
- A group connects the authentication system with the authorization system. It is a set of one or more users who have been granted one or more authorization roles. Sentry allows a set of roles to be configured for a group.
- A configured group provider determines a user's affiliation with a group. The current release supports HDFS-backed groups and locally configured groups. For example,

```
dev_ops = dev_role, ops_role
```

Here the group `dev_ops` is granted the roles `dev_role` and `ops_role`. The members of this group can complete searches that are allowed by these roles.

User to Group Mapping

You can configure Sentry to use either Hadoop groups or groups defined in the policy file.

Important: You can not use both Hadoop groups or local groups at the same time. Use local groups if you want to do a quick proof-of-concept. For production, use Hadoop groups.

To configure Hadoop groups:

Set the `sentry.provider` property in `sentry-site.xml` to `org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider`.

By default, this uses local shell groups. See the Group Mapping section of the HDFS Permissions Guide for more information.

In this case, Sentry uses the Hadoop configuration described in Configuring LDAP Group Mappings. Cloudera Manager automatically uses this configuration. In a deployment not managed by Cloudera Manager, manually set these configuration parameters parameters in the `hadoop-conf` file that is passed to Solr.

OR

To configure local groups:

1. Define local groups in a `[users]` section of the Sentry Policy file. For example:

```
[users]
user1 = group1, group2, group3
user2 = group2, group3
```

2. In `sentry-site.xml`, set `search.sentry.provider` as follows:

```
<property>
    <name>sentry.provider</name>
   <value>org.apache.sentry.provider.file.LocalGroupResourceAuthorizationProvider</value>

  </property>
```

Sample Sentry Configuration

This section provides sample configurations.

Using Solr with the Sentry Service

In CDH 5.8, Cloudera Search adds support for storing permissions in the Sentry service. You can enable storing permissions in the Sentry service by Enabling the Sentry Service for Solr. If you have already configured Sentry's policy file-based approach, you can migrate existing authorization settings as described in Migrating from Sentry Policy Files to the Sentry Service. `solrctl` has been extended to support:

- Migrating existing policy files to the Sentry service
- Managing managing permissions in the Sentry service

The following is an example of the commands used to configure Sentry for Solr using `solrctl` using the Sentry option. These sample commands that follow illustrate establishing two different roles, each of which have different access requirements. The process of creating roles, adding roles to groups, and granting privileges to roles is a typical workflow used to provide different groups varied degrees of access to resources. For reference information, see solrctl Reference on page 23.

Begin by creating roles. The following command creates `ops_role` and `dev_ops_role`:

```
solrctl sentry --create-role ops_role
solrctl sentry --create-role dev_ops_role
```

Next, add existing Hadoop groups to the roles you created. The following command adds `ops_role` to the existing `ops_group` Hadoop group and adds `dev_ops_role` to the existing `dev_ops_group` Hadoop group:

```
solrctl sentry --add-role-group ops_role ops_group
solrctl sentry --add-role-group dev_ops_role dev_ops_group
```

Finally, add privileges to collections and configs to roles. The following command adds the QUERY privilege to `ops_role` for the `logs` collection and all privileges (meaning QUERY and UPDATE) to the `dev_ops_role` for all (`*`) collections:

```
solrctl sentry --grant-privilege ops_role collection=logs->action=Query
solrctl sentry --grant-privilege dev_ops_role collection=*->action=*
```

Using Solr with a Policy File

Use separate policy files for each Sentry-enabled service. Using one file for multiple services results in each service failing on the other services' entries. For example, with a combined Hive and Search file, Search would fail on Hive entries and Hive would fail on Search entries.

Sentry with Search does not support multiple policy files. Other implementations of Sentry such as Sentry for Hive do support different policy files for different databases, but Sentry for Search has no such support for multiple policies.

The following is an example of a Search policy file. The This location must be readable by Solr.

sentry-provider.ini

```
[groups]
# Assigns each Hadoop group to its set of roles
engineer = engineer_role
ops = ops_role
dev_ops = engineer_role, ops_role
hbase_admin = hbase_admin_role

[roles]
# The following grants all access to source_code.
# "collection = source_code" can also be used as syntactic
# sugar for "collection = source_code->action=*"
engineer_role = collection = source_code->action=*

# The following imply more restricted access.
ops_role = collection = hive_logs->action=Query
dev_ops_role = collection = hbase_logs->action=Query

#give hbase_admin_role the ability to create/delete/modify the hbase_logs collection
#as well as to update the config for the hbase_logs collection, called hbase_logs_config.
hbase_admin_role = collection=admin->action=*, collection=hbase_logs->action=*,
config=hbase_logs_config->action=*
```

Sentry Configuration File

Sentry can store configuration as well as privilege policies in files. The `sentry-site.xml` file contains configuration options such as privilege policy file location. The policy files contains the privileges and groups. It has a `.ini` file format and should be stored on HDFS.

The following is an example of a `sentry-site.xml` file.

sentry-site.xml

```
<configuration>
  <property>
    <name>hive.sentry.provider</name>

<value>org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider</value>

  </property>

  <property>
    <name>sentry.solr.provider.resource</name>
```

```
    <value>/path/to/authz-provider.ini</value>
    <!--
        If the HDFS configuration files (core-site.xml, hdfs-site.xml)
        pointed to by SOLR_HDFS_CONFIG in /etc/default/solr
        point to HDFS, the path will be in HDFS;
        alternatively you could specify a full path,
        e.g.:hdfs://namenode:port/path/to/authz-provider.ini
    -->
  </property>
```

Using Policy Files with Sentry

This section contains notes on creating and maintaining the policy file.

Storing the Policy File

Considerations for storing the policy file(s) include:

1. Replication count - Because Sentry reads the file for each query, you should increase this. 10 is a reasonable value.
2. Updating the file - Updates to the file are only reflected when the Solr process is restarted.

Defining Roles

Keep in mind that role definitions are not cumulative. The newer definition replaces the older one. For example, consider the following definition:

```
role1 = privilege1
role1 = privilege2
```

This definition results in `role1` having `privilege2`, not `privilege1` and `privilege2`.

Providing Document-Level Security Using Sentry

For role-based access control of a collection, an administrator modifies a Sentry role so it has query, update, or administrative access.

Collection-level authorization is useful when the access control requirements for the documents in the collection are the same, but users may want to restrict access to a subset of documents in a collection. This finer-grained restriction can be achieved by defining separate collections for each subset, but this is difficult to manage, requires duplicate documents for each collection, and requires that these documents be kept synchronized.

Document-level access control solves this issue by associating authorization tokens with each document in the collection. This enables granting Sentry roles access to sets of documents in a collection.

Document-Level Security Model

Document-level security depends on a chain of relationships between users, groups, roles, and documents.

- Users are assigned to groups.
- Groups are assigned to roles.
- Roles are stored as "authorization tokens" in a specified field in the documents.

Document-level security supports restricting which documents can be viewed by which users. Access is provided by adding roles as "authorization tokens" to a specified document field. Conversely, access is implicitly denied by omitting roles from the specified field. In other words, in a document-level security enabled environment, a user might submit a query that matches a document; if the user is not part of a group that has a role has been granted access to the document, the result is not returned.

For example, Alice might belong to the administrators group. The administrators group may belong to the doc-mgmt role. A document could be ingested and the doc-mgmt role could be added at ingest time. In such a case, if Alice submitted a query that matched the document, Search would return the document, since Alice is then allowed to see any document with the "doc-mgmt" authorization token.

Similarly, Bob might belong to the guests group. The guests group may belong to the public-browser role. If Bob tried the same query as Alice, but the document did not have the public-browser role, Search would not return the result because Bob does not belong to a group that is associated with a role that has access.

Note that collection-level authorization rules still apply, if enabled. Even if Alice is able to view a document given document-level authorization rules, if she is not allowed to query the collection, the query will fail.

Roles are typically added to documents when those documents are ingested, either using the standard Solr APIs or, if using morphlines, the `setValues` morphline command.

Enabling Document-Level Security

Cloudera Search supports document-level security in Search for CDH 5.1 and higher. Document-level security requires collection-level security. Configuring collection-level security is described earlier in this topic.

Document-level security is disabled by default, so the first step in using document-level security is to enable the feature by modifying the `solrconfig.xml.secure` file. Remember to replace the `solrconfig.xml` with this file, as described in Enabling Solr as a Client for the Sentry Service Using the Command Line.

To enable document-level security, change `solrconfig.xml.secure`. The default file contents are as follows:

```
<searchComponent name="queryDocAuthorization">
    <!-- Set to true to enabled document-level authorization -->

    <bool name="enabled">false</bool>


    <!-- Field where the auth tokens are stored in the document -->
    <str name="sentryAuthField">sentry_auth</str>


    <!-- Auth token defined to allow any role to access the document.
         Uncomment to enable. -->

    <!--<str name="allRolesToken">*</str>-->

</searchComponent>
```

- The `enabled` Boolean determines whether document-level authorization is enabled. To enable document level security, change this setting to `true`.
- The `sentryAuthField` string specifies the name of the field that is used for storing authorization information. You can use the default setting of `sentry_auth` or you can specify some other string to be used for assigning values during ingest.

  Note: This field must exist as an explicit or dynamic field in the schema for the collection you are creating with document-level security. `sentry_auth` exists in the default `schema.xml`, which is automatically generated and can be found in the same directory as `solrconfig.xml`.

  for the collection you are creating with document-level security. Schema.xml is in the generated configuration in the same directory as the solrconfig.xml

- The `allRolesToken` string represents a special token defined to allow any role access to the document. By default, this feature is disabled. To enable this feature, uncomment the specification and specify the token. This token should be different from the name of any sentry role to avoid collision. By default it is "*". This feature is useful when first configuring document level security or it can be useful in granting all roles access to a document when the set of roles may change. See Best Practices on page 92 for additional information.

Best Practices

Using `allGroupsToken`

You may want to grant every user that belongs to a role access to certain documents. One way to accomplish this is to specify all known roles in the document, but this requires updating or re-indexing the document if you add a new

role. Alternatively, an `allUser` role, specified in the Sentry `.ini` file, could contain all valid groups, but this role would need to be updated every time a new group was added to the system. Instead, specifying `allGroupsToken` allows any user that belongs to a valid role to access the document. This access requires no updating as the system evolves.

In addition, `allGroupsToken` may be useful for transitioning a deployment to use document-level security. Instead of having to define all the roles upfront, all the documents can be specified with `allGroupsToken` and later modified as the roles are defined.

Consequences of Document-Level Authorization Only Affecting Queries

Document-level security does not prevent users from modifying documents or performing other update operations on the collection. Update operations are only governed by collection-level authorization.

Document-level security can be used to prevent documents being returned in query results. If users are not granted access to a document, those documents are not returned even if that user submits a query that matches those documents. This does not have affect attempted updates.

Consequently, it is possible for a user to not have access to a set of documents based on document-level security, but to still be able to modify the documents using their collection-level authorization update rights. This means that a user can delete all documents in the collection. Similarly, a user might modify all documents, adding their authorization token to each one. After such a modification, the user could access any document using querying. Therefore, if you are restricting access using document-level security, consider granting collection-level update rights only to those users you trust and assume they will be able to access every document in the collection.

Limitations on Query Size

By default queries support up to 1024 Boolean clauses. As a result, queries containing more that 1024 clauses may cause errors. Because authorization information is added by Sentry as part of a query, using document-level security can increase the number of clauses. In the case where users belong to many roles, even simple queries can become quite large. If a query is too large, an error of the following form occurs:

```
org.apache.lucene.search.BooleanQuery$TooManyClauses: maxClauseCount is set to 1024
```

To change the supported number of clauses, edit the `maxBooleanClauses` setting in `solrconfig.xml`. For example, to allow 2048 clauses, you would edit the setting so it appears as follows:

```
<maxBooleanClauses>2048</maxBooleanClauses>
```

For `maxBooleanClauses` to be applied as expected, make any change to this value to all collections and then restart the service. You must make this change to all collections because this option modifies a global Lucene property, affecting all Solr cores. If different `solrconfig.xml` files have different values for this property, the effective value is determined per host, based on the first Solr core to be initialized.

Enabling Secure Impersonation

Secure Impersonation allows a user to make requests as another user in a secure way. The user who has been granted impersonation rights receives the same access as the user being impersonated.

Configure custom security impersonation settings using the **Solr Service Environment Advanced Configuration Snippet (Safety Valve)**. For example, to allow the following impersonations:

- User hue can make requests as any user from any host.
- User `foo` can make requests as any member of group `bar`, from `host1` or `host2`.

  Enter the following values into the **Solr Service Environment Advanced Configuration Snippet (Safety Valve)**:

```
SOLR_SECURITY_ALLOWED_PROXYUSERS=hue,foo
SOLR_SECURITY_PROXYUSER_hue_HOSTS=*
SOLR_SECURITY_PROXYUSER_hue_GROUPS=*
SOLR_SECURITY_PROXYUSER_foo_HOSTS=host1,host2
SOLR_SECURITY_PROXYUSER_foo_GROUPS=bar
```

SOLR_SECURITY_ALLOWED_PROXYUSERS lists all of the users allowed to impersonate. For a user x in SOLR_SECURITY_ALLOWED_PROXYUSERS, SOLR_SECURITY_PROXYUSER_x_HOSTS list the hosts x is allowed to connect from to impersonate, and SOLR_SECURITY_PROXYUSERS_x_GROUPS lists the groups that the users is allowed to impersonate members of. Both GROUPS and HOSTS support the wildcard * and both GROUPS and HOSTS must be defined for a specific user.

Note: Cloudera Manager has its own management of secure impersonation for Hue. To add additional users for Secure Impersonation, use the environment variable safety value for Solr to set the environment variables as above. Be sure to include hue in SOLR_SECURITY_ALLOWED_PROXYUSERS if you want to use secure impersonation for hue.

Search High Availability

Mission critical, large-scale online production systems need to make progress without downtime despite some issues. Cloudera Search provides two routes to configurable, highly available, and fault-tolerant data ingestion:

- Near Real Time (NRT) ingestion using the Flume Solr Sink
- MapReduce based batch ingestion using the MapReduceIndexerTool

Production versus Test Mode

Some exceptions are generally transient, in which case the corresponding task can simply be retried. For example, network connection errors or timeouts are recoverable exceptions. Conversely, tasks associated with an unrecoverable exception cannot simply be retried. Corrupt or malformed parser input data, parser bugs, and errors related to unknown Solr schema fields produce unrecoverable exceptions.

Different modes determine how Cloudera Search responds to different types of exceptions.

- **Configuration parameter isProductionMode=false** (Non-production mode or test mode): Default configuration. Cloudera Search throws exceptions to quickly reveal failures, providing better debugging diagnostics to the user.
- **Configuration parameter isProductionMode=true** (Production mode): Cloudera Search logs and ignores unrecoverable exceptions, enabling mission-critical large-scale online production systems to make progress without downtime, despite some issues.

Note: Categorizing exceptions as recoverable or unrecoverable addresses most cases, though it is possible that an unrecoverable exception could be accidentally misclassified as recoverable. Cloudera provides the `isIgnoringRecoverableExceptions` configuration parameter to address such a case. In a production environment, if an unrecoverable exception is discovered that is classified as recoverable, change `isIgnoringRecoverableExceptions` to `true`. Doing so allows systems to make progress and avoid retrying an event forever. This configuration flag should only be enabled if a misclassification bug has been identified. Please report such bugs to Cloudera.

If Cloudera Search throws an exception according the rules described above, the caller, meaning Flume Solr Sink and MapReduceIndexerTool, can catch the exception and retry the task if it meets the criteria for such retries.

Near Real Time Indexing with the Flume Solr Sink

The Flume Solr Sink uses the settings established by the `isProductionMode` and `isIgnoringRecoverableExceptions` parameters. If a SolrSink does nonetheless receive an exception, the SolrSink rolls the transaction back and pauses. This causes the Flume channel, which is essentially a queue, to redeliver the transaction's events to the SolrSink approximately five seconds later. This redelivering of the transaction event retries the ingest to Solr. This process of rolling back, backing off, and retrying continues until ingestion eventually succeeds.

Here is a corresponding example Flume configuration file `flume.conf`:

```
agent.sinks.solrSink.isProductionMode = true
agent.sinks.solrSink.isIgnoringRecoverableExceptions = true
```

In addition, Flume SolrSink automatically attempts to load balance and failover among the hosts of a SolrCloud before it considers the transaction rollback and retry. Load balancing and failover is done with the help of ZooKeeper, which itself can be configured to be highly available.

Further, Cloudera Manager can configure Flume so it automatically restarts if its process crashes.

To tolerate extended periods of Solr downtime, you can configure Flume to use a high-performance transactional persistent queue in the form of a FileChannel. A FileChannel can use any number of local disk drives to buffer significant amounts of data. For example, you might buffer many terabytes of events corresponding to a week of data. Further, using the Replicating Channel Selector Flume feature, you can configure Flume to replicate the same data both into HDFS as well as into Solr. Doing so ensures that if the Flume SolrSink channel runs out of disk space, data delivery is still delivered to HDFS, and this data can later be ingested from HDFS into Solr using MapReduce.

Many machines with many Flume Solr Sinks and FileChannels can be used in a failover and load balancing configuration to improve high availability and scalability. Flume SolrSink servers can be either co-located with live Solr servers serving end user queries, or Flume SolrSink servers can be deployed on separate industry standard hardware for improved scalability and reliability. By spreading indexing load across a large number of Flume SolrSink servers you can improve scalability. Indexing load can be replicated across multiple Flume SolrSink servers for high availability, for example using Flume features such as Load balancing Sink Processor.

Batch Indexing with MapReduceIndexerTool

The Mappers and Reducers of the MapReduceIndexerTool follow the settings established by the `isProductionMode` and `isIgnoringRecoverableExceptions` parameters. However, if a Mapper or Reducer of the `MapReduceIndexerTool` does receive an exception, it does not retry at all. Instead it lets the MapReduce task fail and relies on the Hadoop Job Tracker to retry failed MapReduce task attempts several times according to standard Hadoop semantics. Cloudera Manager can configure the Hadoop Job Tracker to be highly available. On `MapReduceIndexerTool` startup, all data in the output directory is deleted if that output directory already exists. To retry an entire job that has failed, rerun the program using the same arguments.

For example:

```
hadoop ... MapReduceIndexerTool ... -D isProductionMode=true -D
isIgnoringRecoverableExceptions=true ...
```

Renaming Cloudera Manager Managed Hosts

Cloudera Search supports renaming hosts.

Note: This will require a cluster-wide outage.

Note: This procedure should not be used in environments running JobTracker high availability (HA). If you are running JobTracker HA, contact Cloudera customer support for further assistance.

Renaming hosts involves stopping services and agents, changing settings, and restarting services and agents. You must not restart services or agents before you are instructed to do so. Starting services or agents early may result in a nonfunctional system state.

This topic describes how to change some or all host names in your cluster. Begin by shutting down all services in the cluster.

Prerequisites

Before changing host names, back up the Cloudera Manager database using a tool such as `mysqldump`. For more information, see mysqldump. Store this backup in a safe location. If problems occur, this backup can be used to restore the original cluster state.

Stopping Cloudera Manager Services

Shut down all CDH and Cloudera Manager management services in the cluster.

1. For services that are managed as part of the cluster, click the down-arrow and choose **Stop**.
2. For any services that are still running, right-click each running service, and click **Stop**.
3. After you have stopped all services, shutdown Cloudera manager server.

   RHEL-compatible or SLES systems:

```
$ sudo service cloudera-scm-server stop
```

   Debian/Ubuntu systems:

```
$ sudo /usr/sbin/service cloudera-scm-server stop
```

4. Shutdown the Cloudera agents on the hosts whose names you are changing.

   RHEL-compatible or SLES systems:

```
$ sudo service cloudera-scm-agent stop
```

   Debian/Ubuntu systems:

```
$ sudo /usr/sbin/service cloudera-scm-agent stop
```

Editing the server_host Value

If you are renaming the host running Cloudera Manager, you must edit the `server_host` value in the `config.ini` file on all hosts that are managed by Cloudera Manager. In most cases, the `config.ini` file is found at `/etc/cloudera-scm-agent/`. The `config.ini` file may be found elsewhere if tarballs were used for installation. For example, if you were renaming the Cloudera Manager host to `newhostname.example.com`, you would modify the `server_host` value so it read as follows:

```
server_host=newhostname.example.com
```

Repeat this edit for all hosts that are managed by Cloudera Manager.

Updating Name Services

Update the names of the hosts using the name service method that applies for your operating system.

1. Edit the network or hostname file.

   For RHEL-compatible systems, edit the `HOSTNAME` value in the `network` file to be the new hostname. For example, you might change `HOSTNAME` in `/etc/sysconfig/network` to:

```
HOSTNAME=new.host.name.FQDN
```

   For Debian systems, edit the hostname entries in the `hostname` file to include new hostname. For example, you might delete the old hostname and add the new hostname to the `/etc/hostname` file so it reads:

```
new.host.name.FQDN
```

For SLES systems, edit the hostname entries in the HOSTNAME file to include new hostname. For example, you might delete the old hostname and add the new hostname to the /etc/HOSTNAME file so it reads:

```
new.host.name.FQDN
```

2. Edit the /etc/hosts file. Replace all instances of the old hostname with the new hostname.

Updating the Cloudera Manager Database

Modify the Cloudera Manager database to reflect the new names. The commands vary depending on the type of database you are using. For example, for MySQL, using the following process:

1. Log into mysql as root and use the Cloudera Manager database. For example, for a database named cm, you might use the following command

```
# mysql -h localhost -u scm -p
use cm;

mysql> select HOST_ID, HOST_IDENTIFIER, NAME from HOSTS;
```

Note the HOST_ID value for each of the hosts you are modifying. This will be $ROW_ID in the subsequent commands.

2. For the hosts you're changing use a command of the form:

```
mysql> update HOSTS set HOST_IDENTIFIER = 'new.host.name.FQDN' where HOST_ID = $ROW_ID;
```

For example, a full transcript of user input from such a process might be:

```
 # mysql -u root -p
password>

mysql> show databases;
mysql> use cm;
mysql> select HOST_ID, HOST_IDENTIFIER, NAME from HOSTS;
mysql> update HOSTS set HOST_IDENTIFIER = 'new.host.name.FQDN' where HOST_ID = 2;
```

Starting Cloudera Manager Services

Restart the Cloudera Manager server and agents using commands of the form:

1. Start Cloudera agent on the hosts whose names you changed.

RHEL-compatible or SLES systems:

```
$ sudo service cloudera-scm-agent start
```

Debian/Ubuntu systems:

```
$ sudo /usr/sbin/service cloudera-scm-agent start
```

2. Start the Cloudera Manager Server.

RHEL-compatible or SLES systems:

```
$ sudo service cloudera-scm-server start
```

Debian/Ubuntu systems:

```
$ sudo /usr/sbin/service cloudera-scm-server start
```

Updating for NameNode High Availability Automatic Failover

If NameNode high availability automatic failover is enabled, you must update the ZooKeeper Failover Controller (ZKFC) to reflect the name changes. If you are not using NameNode high availability, skip to the next section.

> **Note:** As stated earlier, this procedure should not be used in environments running JobTracker High Availability (HA). If you have already completed the preceding steps in an environment with JobTracker HA enabled, the subsequent steps should not be completed in your environment. Contact support now.

To update the ZKFC host:

1. Start the ZooKeeper services using the Cloudera Manager Admin Console.

   > **Note:** Do not start any other services. It is especially important that you not start HDFS.

2. Log into one of the hosts that is hosting the ZooKeeper server role.
3. Delete the nameservice znode. For a package based installation, delete the `zkCli.sh` file using a command similar to:

```
$ rm -f /usr/lib/zookeeper/bin/zkCli.sh
```

   For a parcel-based installation, delete the `zkCli.sh` file using a command similar to:

```
$ rm -f /opt/cloudera/parcels/CDH/lib/zookeeper/bin/zkCli.sh
```

4. Verify that the HA znode exists by checking for the `hadoop-ha`. For example:

```
zkCli$ ls /hadoop-ha
```

   If the HA znode does not exist, use the Cloudera Manager Admin Console to select the HDFS service and then choose **Initialize High Availability State in ZooKeeper**.
5. Delete the old znode. For example use a command similar to:

```
zkCli$ rm -rf /hadoop-ha/nameservice1
```

6. Use the Cloudera Manager Admin Console to initialize HA in ZooKeeper by clicking **HDFS** > **Instances** > **Action** > **Initialize High Availability State in Zookeeper…**.

Updating Cloudera Management Service Host Information

If you have changed the names of hosts hosting management services, you must update the management service with the new host name. Management services include Host Monitor, Service Monitor, Reports Manager, Activity Monitor, and Navigator. You must do this for each service that is hosted on a host whose name has changed.

To update management service host name configuration

1. In the Cloudera Manager Admin Console, select the service
2. Click the **Configuration** tab.
3. Type `Database Hostname` in the **Search** box to locate all the database hostname properties.
4. Edit the ***Management Service Name*** **Database Hostname** property for the new hostname.
5. Click **Save Changes** to commit the changes.

Returning the System to a Running State

Now that you have renamed hosts and updated settings to reflect these new names, redeploy client configuration files.

- For Cloudera Manager 4, see Redeploying the Client Configuration Files Manually in Deploying Client Configuration Files.
- For Cloudera Manager 5, see Manually RedeployingClient Configuration Files in Client Configuration Files.

Start any services that were previously stopped.

Tuning the Solr Server

Solr performance tuning is a complex task. The following sections provide more details.

Tuning to Complete During Setup

Some tuning is best completed during the setup of you system or may require some re-indexing.

Configuring Lucene Version Requirements

You can configure Solr to use a specific version of Lucene. This can help ensure that the Lucene version that Search uses includes the latest features and bug fixes. At the time that a version of Solr ships, Solr is typically configured to use the appropriate Lucene version, in which case there is no need to change this setting. If a subsequent Lucene update occurs, you can configure the Lucene version requirements by directly editing the `luceneMatchVersion` element in the `solrconfig.xml` file. Versions are typically of the form `x.y`, such as `4.4`. For example, to specify version 4.4, you would ensure the following setting exists in `solrconfig.xml`:

```
<luceneMatchVersion>4.4</luceneMatchVersion>
```

Designing the Schema

When constructing a schema, use data types that most accurately describe the data that the fields will contain. For example:

- Use the `tdate` type for dates. Do this instead of representing dates as strings.
- Consider using the `text` type that applies to your language, instead of using `String`. For example, you might use `text_en`. `Text` types support returning results for subsets of an entry. For example, querying on "john" would find "John Smith", whereas with the string type, only exact matches are returned.
- For IDs, use the string type.

General Tuning

The following tuning categories can be completed at any time. It is less important to implement these changes before beginning to use your system.

General Tips

- Enabling multi-threaded faceting can provide better performance for field faceting. When multi-threaded faceting is enabled, field faceting tasks are completed in a parallel with a thread working on every field faceting task simultaneously. Performance improvements do not occur in all cases, but improvements are likely when all of the following are true:
  - The system uses highly concurrent hardware.
  - Faceting operations apply to large data sets over multiple fields.
  - There is not an unusually high number of queries occurring simultaneously on the system. Systems that are lightly loaded or that are mainly engaged with ingestion and indexing may be helped by multi-threaded faceting; for example, a system ingesting articles and being queried by a researcher. Systems heavily loaded by user queries are less likely to be helped by multi-threaded faceting; for example, an e-commerce site with heavy user-traffic.

Note: Multi-threaded faceting only applies to field faceting and not to query faceting.

- Field faceting identifies the number of unique entries for a field. For example, multi-threaded faceting could be used to simultaneously facet for the number of unique entries for the fields, "color" and "size". In such a case, there would be two threads, and each thread would work on faceting one of the two fields.
- Query faceting identifies the number of unique entries that match a query for a field. For example, query faceting could be used to find the number of unique entries in the "size" field are between 1 and 5. Multi-threaded faceting does not apply to these operations.

To enable multi-threaded faceting, add `facet-threads` to queries. For example, to use up to 1000 threads, you might use a query as follows:

```
http://localhost:8983/solr/collection1/select?q=*:*&facet=true&fl=id&facet.field=f0_ws&facet.threads=1000
```

If `facet-threads` is omitted or set to 0, faceting is single-threaded. If `facet-threads` is set to a negative value, such as -1, multi-threaded faceting will use as many threads as there are fields to facet up to the maximum number of threads possible on the system.

- If your environment does not require Near Real Time (NRT), turn off soft auto-commit in `solrconfig.xml`.
- In most cases, do not change the default batch size setting of 1000. If you are working with especially large documents, you may consider decreasing the batch size.
- To help identify any garbage collector (GC) issues, enable GC logging in production. The overhead is low and the JVM supports GC log rolling as of 1.6.0_34.
  - The minimum recommended GC logging flags are: `-XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintGCDetails`.
  - To rotate the GC logs: `-Xloggc: -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles= -XX:GCLogFileSize=`.

Solr and HDFS - the Block Cache

Cloudera Search enables Solr to store indexes in an HDFS filesystem. To maintain performance, an HDFS block cache has been implemented using Least Recently Used (LRU) semantics. This enables Solr to cache HDFS index files on read and write, storing the portions of the file in JVM "direct memory" (meaning off heap) by default or optionally in the JVM heap. Direct memory is preferred as it is not affected by garbage collection.

Batch jobs typically do not use the cache, while Solr servers (when serving queries or indexing documents) should. When running indexing using MapReduce, the MR jobs themselves do not use the block cache. Block caching is turned off by default and should be left disabled.

Tuning of this cache is complex and best practices are continually being refined. In general, allocate a cache that is about 10-20% of the amount of memory available on the system. For example, when running HDFS and Solr on a host with 50 GB of memory, typically allocate 5-10 GB of memory using `solr.hdfs.blockcache.slab.count`. As index sizes grow you may need to tune this parameter to maintain optimal performance.

Note: Block cache metrics are currently unavailable.

Configuration

The following parameters control caching. They can be configured at the Solr process level by setting the respective system property or by editing the `solrconfig.xml` directly.

Parameter	Default	Description
`solr.hdfs.blockcache.enabled`	true	Enable the block cache.

Parameter	Default	Description
`solr.hdfs.blockcache.read.enabled`	true	Enable the read cache.
`solr.hdfs.blockcache.write.enabled`	false	Enable the write cache.
`solr.hdfs.blockcache.direct.memory.allocation`	true	Enable direct memory allocation. If this is false, heap is used.
`solr.hdfs.blockcache.slab.count`	1	Number of memory slabs to allocate. Each slab is 128 MB in size.
`solr.hdfs.blockcache.global`	true	If enabled, a single HDFS block cache is used for all SolrCores on a host. If `blockcache.global` is disabled, each SolrCore on a host creates its own private HDFS block cache. Enabling this parameter simplifies managing HDFS block cache memory.

Note:

Increasing the direct memory cache size may make it necessary to increase the maximum direct memory size allowed by the JVM. Each Solr slab allocates the slab's memory, which is 128 MB by default, as well as allocating some additional direct memory overhead. Therefore, ensure that the `MaxDirectMemorySize` is set comfortably above the value expected for slabs alone. The amount of additional memory required varies according to multiple factors, but for most cases, setting `MaxDirectMemorySize` to at least 20-30% more than the total memory configured for slabs is sufficient. Setting the `MaxDirectMemorySize` to the number of slabs multiplied by the slab size does not provide enough memory.

To set `MaxDirectMemorySize` using Cloudera Manager

1. Go to the Solr service.
2. Click the **Configuration** tab.
3. In the Search box, type **Java Direct Memory Size of Solr Server in Bytes**.
4. Set the new direct memory value.
5. Restart Solr servers after editing the parameter.

Solr HDFS optimizes caching when performing NRT indexing using Lucene's `NRTCachingDirectory`.

Lucene caches a newly created segment if both of the following conditions are true:

- The segment is the result of a flush or a merge and the estimated size of the merged segment is <= `solr.hdfs.nrtcachingdirectory.maxmergesizemb`.
- The total cached bytes is <= `solr.hdfs.nrtcachingdirectory.maxcachedmb`.

The following parameters control NRT caching behavior:

Parameter	Default	Description
`solr.hdfs.nrtcachingdirectory.enable`	true	Whether to enable the NRTCachingDirectory.
`solr.hdfs.nrtcachingdirectory.maxcachedmb`	192	Size of the cache in megabytes.
`solr.hdfs.nrtcachingdirectory.maxmergesizemb`	16	Maximum segment size to cache.

Here is an example of `solrconfig.xml` with defaults:

```
 <directoryFactory name="DirectoryFactory">
   <bool name="solr.hdfs.blockcache.enabled">${solr.hdfs.blockcache.enabled:true}</bool>

   <int name="solr.hdfs.blockcache.slab.count">${solr.hdfs.blockcache.slab.count:1}</int>

    <bool
name="solr.hdfs.blockcache.direct.memory.allocation">${solr.hdfs.blockcache.direct.memory.allocation:true}</bool>

    <int
name="solr.hdfs.blockcache.blocksperbank">${solr.hdfs.blockcache.blocksperbank:16384}</int>

    <bool
name="solr.hdfs.blockcache.read.enabled">${solr.hdfs.blockcache.read.enabled:true}</bool>

    <bool
name="solr.hdfs.blockcache.write.enabled">${solr.hdfs.blockcache.write.enabled:true}</bool>

    <bool
name="solr.hdfs.nrtcachingdirectory.enable">${solr.hdfs.nrtcachingdirectory.enable:true}</bool>

    <int
name="solr.hdfs.nrtcachingdirectory.maxmergesizemb">${solr.hdfs.nrtcachingdirectory.maxmergesizemb:16}</int>

    <int
name="solr.hdfs.nrtcachingdirectory.maxcachedmb">${solr.hdfs.nrtcachingdirectory.maxcachedmb:192}</int>
</directoryFactory>
```

The following example illustrates passing Java options by editing the `/etc/default/solr` or `/opt/cloudera/parcels/CDH-*/etc/default/solr` configuration file:

```
CATALINA_OPTS="-Xmx10g -XX:MaxDirectMemorySize=20g -XX:+UseLargePages
-Dsolr.hdfs.blockcache.slab.count=100"
```

For better performance, Cloudera recommends setting the Linux swap space on all Solr server hosts as shown below:

```
# minimize swappiness
sudo sysctl vm.swappiness=1
sudo bash -c 'echo "vm.swappiness=1">> /etc/sysctl.conf'
# disable swap space until next reboot:
sudo /sbin/swapoff -a
```

Note: Cloudera previously recommended setting `vm.swappiness` to 0. However, a change in Linux kernel 3.5-rc1 (fe35004f), can lead to frequent out of memory (OOM) errors. For details, see Evan Klitzke's blog post. This commit was backported to RHEL / CentOS 6.4 and Ubuntu 12.04 LTS (Long Term Support).

Threads

Configure the Tomcat server to have more threads per Solr instance. Note that this is only effective if your hardware is sufficiently powerful to accommodate the increased threads. 10,000 threads is a reasonable number to try in many cases.

To change the maximum number of threads, add a `maxThreads` element to the Connector definition in the Tomcat server's `server.xml` configuration file. For example, if you installed Search for CDH 5 using parcels installation, you would modify the Connector definition in the `<parcel path>/CDH/etc/solr/tomcat-conf.dist/conf/server.xml` file so this:

```
    <Connector port="${solr.port}" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

Becomes this:

```
    <Connector port="${solr.port}" protocol="HTTP/1.1"
               maxThreads="10000"
               connectionTimeout="20000"
               redirectPort="8443" />
```

Garbage Collection

Choose different garbage collection options for best performance in different environments. Some garbage collection options typically chosen include:

- **Concurrent low pause collector**: Use this collector in most cases. This collector attempts to minimize "Stop the World" events. Avoiding these events can reduce connection timeouts, such as with ZooKeeper, and may improve user experience. This collector is enabled using `-XX:+UseConcMarkSweepGC`.
- **Throughput collector**: Consider this collector if raw throughput is more important than user experience. This collector typically uses more "Stop the World" events so this may negatively affect user experience and connection timeouts such as ZooKeeper heartbeats. This collector is enabled using `-XX:+UseParallelGC`. If `UseParallelGC` "Stop the World" events create problems, such as ZooKeeper timeouts, consider using the `UseParNewGC` collector as an alternative collector with similar throughput benefits.

You can also affect garbage collection behavior by increasing the Eden space to accommodate new objects. With additional Eden space, garbage collection does not need to run as frequently on new objects.

Replication

You can adjust the degree to which different data is replicated.

Replication Settings

Note: Do not adjust HDFS replication settings for Solr in most cases.

To adjust the Solr replication factor for index files stored in HDFS

1. Configure the `solr.hdfs.confdir` system property to refer to the Solr HDFS configuration files. Typically the value is: `-Dsolr.hdfs.confdir=/etc/solrhdfs/`
   - In a Cloudera Manager deployment, set this value using the advanced Solr setting box advanced configuration snippet.
   - In a deployment not managed by Cloudera Manager, set the solr `confdir` system property by adding the following to the command you used to invoke solr: `-Dsolr.hdfs.confdir=/etc/solrhdfs`
2. Set the DFS replication value in the HDFS configuration file at the location you specified in the previous step. For example, to set the replication value to 2, you would change the `dfs.replication` setting as follows:

```
<property>
<name>dfs.replication<name>
<value>2<value>
<property>
```

3. Restart the Solr service.

Optionally, you can also configure the transaction log replication factor. Cloudera recommends leaving the value unchanged at 3 or, barring that, leaving it greater than 1. For more information on changing this setting, see Transaction Log Replication.

Replicas

If you have sufficient additional hardware, add more replicas for a linear boost of query throughput. Note that adding replicas may slow write performance on the first replica, but otherwise this should have minimal negative consequences.

Transaction Log Replication

Beginning with CDH 5.4.1, Search for CDH supports configurable transaction log replication levels for replication logs stored in HDFS.

Configure the replication factor by modifying the `tlogDfsReplication` setting in `solrconfig.xml`. The `tlogDfsReplication` is a new setting in the `updateLog` settings area. An excerpt of the `solrconfig.xml` file where the transaction log replication factor is set is as follows:

```
<updateHandler class="solr.DirectUpdateHandler2">

   <!-- Enables a transaction log, used for real-time get, durability, and
        and solr cloud replica recovery.  The log can grow as big as
        uncommitted changes to the index, so use of a hard autoCommit
        is recommended (see below).
        "dir" - the target directory for transaction logs, defaults to the
               solr data directory.  -->
   <updateLog>
     <str name="dir">${solr.ulog.dir:}</str>
     <int name="tlogDfsReplication">3</int>
   </updateLog>
```

You might want to increase the replication level from the default level of 1 to some higher value such as 3. Increasing the transaction log replication level can:

- Reduce the chance of data loss, especially when the system is otherwise configured to have single replicas of shards. For example, having single replicas of shards is reasonable when `autoAddReplicas` is enabled, but without additional transaction log replicas, the risk of data loss during a host failure would increase.
- Facilitate rolling upgrade of HDFS while Search is running. If you have multiple copies of the log, when a host with the transaction log becomes unavailable during the rolling upgrade process, another copy of the log can continue to collect transactions.
- Facilitate HDFS write lease recovery.

Initial testing shows no significant performance regression for common use cases.

Shards

In some cases, oversharding can help improve performance including intake speed. If your environment includes massively parallel hardware and you want to use these available resources, consider oversharding. You might increase the number of replicas per host from 1 to 2 or 3. Making such changes creates complex interactions, so you should continue to monitor your system's performance to ensure that the benefits of oversharding do not outweigh the costs.

Commits

Changing commit values may improve performance in some situation. These changes result in tradeoffs and may not be beneficial in all cases.

- For hard commit values, the default value of 60000 (60 seconds) is typically effective, though changing this value to 120 seconds may improve performance in some cases. Note that setting this value to higher values, such as 600 seconds may result in undesirable performance tradeoffs.
- Consider increasing the auto-commit value from 15000 (15 seconds) to 120000 (120 seconds).
- Enable soft commits and set the value to the largest value that meets your requirements. The default value of 1000 (1 second) is too aggressive for some environments.

Other Resources

- General information on Solr caching is available on the SolrCaching page on the Solr Wiki.
- Information on issues that influence performance is available on the SolrPerformanceFactors page on the Solr Wiki.
- Resource Management describes how to use Cloudera Manager to manage resources, for example with Linux cgroups.
- For information on improving querying performance, see How to make searching faster.

- For information on improving indexing performance, see How to make indexing faster.

Troubleshooting Cloudera Search

After installing and deploying Cloudera Search, use the information in this section to troubleshoot problems.

Troubleshooting

The following table contains some common troubleshooting techniques.

Note: In the URLs in the following table, you must replace entries such as `<server:port>` with values from your environment. The port defaults value is 8983, but see `/etc/default/solr` or `/opt/cloudera/parcels/CDH-*/etc/default/solr` for the port if you are in doubt.

Symptom	Explanation	Recommendation
All	Varied	Examine Solr log. By default, the log can be found at `/var/log/solr/solr.out`.
The Solr Admin UI is unavailable	If no privileges are granted, no access is possible. For example, accessing the Solr Admin UI requires the QUERY privilege. If no users are granted the QUERY privilege, no access to the Solr Admin UI is possible.	Ensure users attempting to access the UI are granted the QUERY privilege. For more information, see Configuring Sentry Authorization for Solr on page 87.
No documents found	Server may not be running	Browse to http://*server*:*port*/solr to see if the server responds. Check that cores are present. Check the contents of cores to ensure that numDocs is more than 0.
No documents found	Core may not have documents	Browsing `http://server:port/solr/[collection name]/select?q=*:*&wt=json&indent=true` should show numFound, which is near the top, to be more than 0.
The secure Solr Server fails to respond to Solrj requests, but other clients such as curl can communicate successfully	This may be a version compatibility issue. `Httpclient` 4.2.3, which ships with solrj in Search 1.x, has a dependency on `commons-codec` 1.7. If an earlier version of `commons-codec` is on the classpath, `httpclient` may be unable to communicate using Kerberos.	Ensure your application is using `commons-codec` 1.7 or higher. Alternatively, use `httpclient` 4.2.5 instead of version 4.2.3 in your application. Version 4.2.3 behaves correctly with earlier versions of `commons-codec`.

Dynamic Solr Analysis

Any JMX-aware application can query Solr for information and display results dynamically. For example, Zabbix, Nagios, and many others have been used successfully. When completing Static Solr Log Analysis, many of the items related to extracting data from the log files can be seen from querying Solr, at least the last value (as opposed to the history which is available from the log file). These are often useful for status boards. In general, anything available from the Solr admin page can be requested on a live basis from Solr. Some possibilities include:

- `numDocs`/`maxDoc` per core. This can be important since the difference between these numbers indicates the number of deleted documents in the index. Deleted documents take up disk space and memory. If these numbers vary greatly, this may be a rare case where optimizing is advisable.
- Cache statistics, including:
  - Hit ratios

  - Autowarm times
  - Evictions

- Almost anything available on the admin page. Note that drilling down into the “schema browser” can be expensive.

Other Troubleshooting Information

Since the use cases for Solr and search vary, there is no single solution for all cases. That said, here are some common challenges that many Search users have encountered:

- Testing with unrealistic data sets. For example, a users may test a prototype that uses faceting, grouping, sorting, and complex schemas against a small data set. When this same system is used to load of real data, performance issues occur. Using realistic data and use-cases is essential to getting accurate results.
- If the scenario seems to be that the system is slow to ingest data, consider:
  - Upstream speed. If you have a SolrJ program pumping data to your cluster and ingesting documents at a rate of 100 docs/second, the gating factor may be upstream speed. To test for limitations due to upstream speed, comment out only the code that sends the data to the server (for example, `SolrHttpServer.add(doclist)`) and time the program. If you see a throughput bump of less than around 10%, this may indicate that your system is spending most or all of the time getting the data from the system-of-record.
  - This may require pre-processing.
  - Indexing with a single thread from the client. `ConcurrentUpdateSolrServer` can use multiple threads to avoid I/O waits.
  - Too-frequent commits. This was historically an attempt to get NRT processing, but with SolrCloud hard commits this should be quite rare.
  - The complexity of the analysis chain. Note that this is rarely the core issue. A simple test is to change the schema definitions to use trivial analysis chains and then measure performance.
  - When the simple approaches fail to identify an issue, consider using profilers.

SolrCloud and ZooKeeper

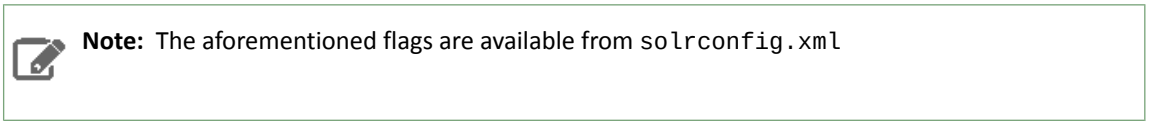
SolrCloud is relatively new and relies on ZooKeeper to hold state information. There are not yet best practices related to SolrCloud. Monitoring ZooKeeper is valuable in this case and is available through Cloudera Manager.

Static Solr Log Analysis

To do a static analysis, inspect the log files, schema files, and the actual index for issues. If possible, connect to the live Solr instance while simultaneously examining log files so you can compare the schema with the index. The schema and the index can be out of sync in situations where the schema is changed, but the index was never rebuilt. Some hints are:

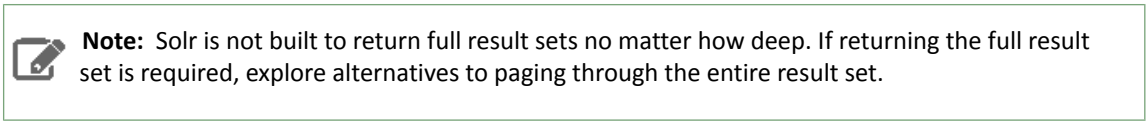
- A high number or proportion of 0-match queries. This indicates that the user-facing part of the application is making it easy for users to enter queries for which there are no matches. In Cloudera Search, given the size of the data, this should be an extremely rare event.
- Queries that match an excessive number of documents. All documents that match a query have to be scored, and the cost of scoring a query goes up as the number of hits increases. Examine any frequent queries that match millions of documents. An exception to this case is “constant score queries”. Queries, such as those of the form ":" bypass the scoring process entirely.
- Overly complex queries. Defining what constitutes overly complex queries is difficult to do, but a very general rule is that queries over 1024 characters in length are likely to be overly complex.
- High autowarm times. Autowarming is the process of filling caches. Some queries are run before a new searcher serves the first live user request. This keeps the first few users from having to wait. Autowarming can take many seconds or can be instantaneous. Excessive autowarm times often indicate excessively generous autowarm parameters. Excessive autowarming usually has limited benefit, with longer runs effectively being wasted work.
  - Cache autowarm. Each Solr cache has an autowarm parameter. You can usually set this value to an upper limit of 128 and tune from there.

  - `FirstSearcher/NewSearcher`. The `solrconfig.xml` file contains queries that can be fired when a new searcher is opened (the index is updated) and when the server is first started. Particularly for `firstSearcher`, it can be valuable to have a query that sorts relevant fields.



Note: The aforementioned flags are available from `solrconfig.xml`

- Exceptions. The Solr log file contains a record of all exceptions thrown. Some exceptions, such as exceptions resulting from invalid query syntax are benign, but others, such as Out Of Memory, require attention.
- Excessively large caches. The size of caches such as the filter cache are bounded by maxDoc/8. Having, for instance, a filterCache with 10,000 entries is likely to result in Out Of Memory errors. Large caches occurring in cases where there are many documents to index is normal and expected.
- Caches with low hit ratios, particularly filterCache. Each cache takes up some space, consuming resources. There are several caches, each with its own hit rate.
  - `filterCache`. This cache should have a relatively high hit ratio, typically around 80%.
  - `queryResultCache`. This is primarily used for paging so it can have a very low hit ratio. Each entry is quite small as it is basically composed of the raw query as a string for a key and perhaps 20-40 `ints`. While useful, unless users are experiencing paging, this requires relatively little attention.
  - `documentCache`. This cache is a bit tricky. It's used to cache the document data (stored fields) so various components in a request handler don't have to re-read the data from the disk. It's an open question how useful it is when using `MMapDirectory` to access the index.
- Very deep paging. Users seldom go beyond the first page and very rarely to go through 100 pages of results. A `&start=<pick your number>` query indicates unusual usage that should be identified. Deep paging may indicate some agent is completing scraping.



Note: Solr is not built to return full result sets no matter how deep. If returning the full result set is required, explore alternatives to paging through the entire result set.

- Range queries should work on `trie` fields. `Trie` fields (numeric types) store extra information in the index to aid in range queries. If range queries are used, it's almost always a good idea to be using `trie` fields.
- `fq` clauses that use bare NOW. `fq` clauses are kept in a cache. The cache is a map from the `fq` clause to the documents in your collection that satisfy that clause. Using bare NOW clauses virtually guarantees that the entry in the filter cache is not be re-used.
- Multiple simultaneous searchers warming. This is an indication that there are excessively frequent commits or that autowarming is taking too long. This usually indicates a misunderstanding of when you should issue commits, often to simulate Near Real Time (NRT) processing or an indexing client is improperly completing commits. With NRT, commits should be quite rare, and having more than one simultaneous autowarm should not happen.
- Stored fields that are never returned (`fl=` clauses). Examining the queries for `fl=` and correlating that with the schema can tell if stored fields that are not used are specified. This mostly wastes disk space. And `fl=*` can make this ambiguous. Nevertheless, it's worth examining.
- Indexed fields that are never searched. This is the opposite of the case where stored fields are never returned. This is more important in that this has real RAM consequences. Examine the request handlers for "edismax" style parsers to be certain that indexed fields are not used.
- Queried but not analyzed fields. It's rare for a field to be queried but not analyzed in any way. Usually this is only valuable for "string" type fields which are suitable for machine-entered data, such as part numbers chosen from a pick-list. Data that is not analyzed should not be used for anything that humans enter.
- String fields. String fields are completely unanalyzed. Unfortunately, some people confuse `string` with Java's `String` type and use them for text that should be tokenized. The general expectation is that string fields should be used sparingly. More than just a few string fields indicates a design flaw.
- Whenever the schema is changed, re-index the entire data set. Solr uses the schema to set expectations about the index. When schemas are changed, there's no attempt to retrofit the changes to documents that are currently indexed, but any new documents are indexed with the new schema definition. So old and new documents can

have the same field stored in vastly different formats (for example, `String` and `TrieDate`) making your index inconsistent. This can be detected by examining the raw index.

- Query stats can be extracted from the logs. Statistics can be monitored on live systems, but it is more common to have log files. Here are some of the statistics you can gather:
  - Longest running queries
  - 0-length queries
  - average/mean/min/max query times
  - You can get a sense of the effects of commits on the subsequent queries over some interval (time or number of queries) to see if commits are the cause of intermittent slowdowns
- Too-frequent commits have historically been the cause of unsatisfactory performance. This is not so important with NRT processing, but it is valuable to consider.
- Optimizing an index, which could improve search performance before, is much less necessary now. Anecdotal evidence indicates optimizing may help in some cases, but the general recommendation is to use `expungeDeletes`, instead of committing.
  - Modern Lucene code does what `optimize` used to do to remove deleted data from the index when segments are merged. Think of this process as a background optimize. Note that merge policies based on segment size can make this characterization inaccurate.
  - It still may make sense to optimize a read-only index.
  - `Optimize` is now renamed `forceMerge`.

Cloudera Search Glossary

commit

An operation that forces documents to be made searchable.

- **hard** - A commit that starts the autowarm process, closes old searchers and opens new ones. It may also trigger replication.
- **soft** - New functionality with NRT and SolrCloud that makes documents searchable without requiring the work of hard commits.

embedded Solr

The ability to execute Solr commands without having a separate servlet container. Generally, use of embedded Solr is discouraged because it is often used due to the mistaken belief that HTTP is inherently too expensive to go fast. With Cloudera Search, and especially if the idea of some kind of MapReduce process is adopted, embedded Solr is probably advisable.

faceting

“Counting buckets” for a query. For example, suppose the search is for the term “shoes”. You might want to return a result that there were various different quantities, such as "X brown, Y red and Z blue shoes" that matched the rest of the query.

filter query (fq)

A clause that limits returned results. For instance, “fq=sex:male” limits results to males. Filter queries are cached and reused.

Near Real Time (NRT)

The ability to search documents very soon after they are added to Solr. With SolrCloud, this is largely automatic and measured in seconds.

replica

In SolrCloud, a complete copy of a shard. Each replica is identical, so only one replica has to be queried (per shard) for searches.

sharding

Splitting a single logical index up into some number of sub-indexes, each of which can be hosted on a separate machine. Solr (and especially SolrCloud) handles querying each shard and assembling the response into a single, coherent list.

SolrCloud

ZooKeeper-enabled, fault-tolerant, distributed Solr. This is new in Solr 4.0.

SolrJ

A Java API for interacting with a Solr instance.

Cloudera Search Frequently Asked Questions

General

The following are general questions about Cloudera Search and the answers to those questions.

What is Cloudera Search?

Cloudera Search is Apache Solr integrated with CDH, including Apache Lucene, Apache SolrCloud, Apache Flume, Apache Tika, and Apache Hadoop MapReduce and HDFS. Cloudera Search also includes valuable integrations that make searching more scalable, easy to use, and optimized for both near-real-time and batch-oriented indexing. These integrations include Cloudera Morphlines, a customizable transformation chain that simplifies loading any type of data into Cloudera Search.

What is the difference between Lucene and Solr?

Lucene is a low-level search library that is accessed by a Java API. Solr is a search server that runs in a servlet container and provides structure and convenience around the underlying Lucene library.

What is Apache Tika?

The Apache Tika toolkit detects and extracts metadata and structured text content from various documents using existing parser libraries. Using the `solrCell` morphline command, the output from Apache Tika can be mapped to a Solr schema and indexed.

How does Cloudera Search relate to web search?

Traditional web search engines crawl web pages on the Internet for content to index. Cloudera Search indexes files and data that are stored in HDFS and HBase. To make web data available through Cloudera Search, it needs to be downloaded and stored in (CDH).

How does Cloudera Search relate to enterprise search?

Enterprise search connects with different backends (such as RDBMS and filesystems) and indexes data in all those systems. Cloudera Search is intended as a full-text search capability for data in CDH. Cloudera Search is a tool added to the Cloudera data processing platform and does not aim to be a stand-alone search solution, but rather a user-friendly interface to explore data in Hadoop and HBase.

How does Cloudera Search relate to custom search applications?

Custom and specialized search applications are an excellent complement to the Cloudera data-processing platform. Cloudera Search is not designed to be a custom application for niche vertical markets. However, Cloudera Search does include a simple search GUI through a plug-in application for Hue. It is based on the Solr API and allows for easy exploration, along with all of the other Hadoop frontend applications in Hue.

Do Search security features use Kerberos?

Yes, Cloudera Search includes support for Kerberos authentication. Search continues to use simple authentication with the anonymous user as the default configuration, but Search now supports changing the authentication scheme to Kerberos. All required packages are installed during the installation or upgrade process. Additional configuration is required before Kerberos is available in your environment.

Do I need to configure Sentry restrictions for each access mode, such as for the admin console and for the command line?

Sentry restrictions are consistently applied regardless of the way users attempt to complete actions. For example, restricting access to data in a collection consistently restricts that access, whether queries come from the command line, from a browser, or through the admin console.

Does Search support indexing data stored in JSON files and objects?

Yes, you can use the `readJson` and `extractJsonPaths` morphline commands that are included with the CDK to access JSON data and files. For more information, see cdk-morphlines-json.

How can I set up Cloudera Search so that results include links back to the source that contains the result?

You can use stored results fields to create links back to source documents. For information on data types, including the option to set results fields as stored, see the Solr Wiki page on SchemaXml.

For example, with `MapReduceIndexerTool` you can take advantage of fields such as `file_path`. See MapReduceIndexerTool Metadata on page 39 for more information. The output from the MapReduceIndexerTool includes file path information that can be used to construct links to source documents.

If you use the Hue UI, you can link to data in HDFS by inserting links of the form:

```
<a href="/filebrowser/download/{{file_path}}?disposition=inline">Download</a>
```

Performance and Fail Over

The following are questions about performance and fail over in Cloudera Search and the answers to those questions.

How large of an index does Cloudera Search support per search server?

There are too many variables to provide a single answer to this question. Typically, a server can host from 10 to 300 million documents, with the underlying index as large as hundreds of gigabytes. To determine a reasonable maximum document quantity and index size for servers in your deployment, prototype with realistic data and queries.

What is the response time latency I can expect?

Many factors affect how quickly responses are returned. Some factors that contribute to latency include whether the system is also completing indexing, the type of fields you are searching, whether the search results require aggregation, and whether there are sufficient resources for your search services.

With appropriately-sized hardware, if the query results are found in memory, they may be returned within milliseconds. Conversely, complex queries requiring results aggregation over huge indexes may take a few seconds.

The time between when Search begins to work on indexing new data and when that data can be queried can be as short as a few seconds, depending on your configuration.

This high performance is supported by custom caching optimizations that Cloudera has added to the Solr/HDFS integration. These optimizations allow for rapid read and writes of files in HDFS, performing at or above the speeds of stand-alone Solr reading and writing from local disk.

What happens when a write to the Lucene indexer fails?

Cloudera Search provides two configurable, highly available, and fault-tolerant data ingestion schemes: near real-time ingestion using the Flume Solr Sink and MapReduce-based batch ingestion using the MapReduceIndexerTool. These approaches are discussed in more detail in Search High Availability on page 94.

What hardware or configuration changes can I make to improve Search performance?

Search performance can be constrained by CPU limits. If you're seeing bottlenecks, consider allocating more CPU to Search.

How can I redistribute shards across a cluster?

You can move shards between hosts using the process described in Migrating Solr Replicas on page 77.

Can I adjust replication levels?

For information on adjusting replication levels, see Replication Settings. Do not adjust replication settings for HDFS or Solr for most cases.

Schema Management

The following are questions about schema management in Cloudera Search and the answers to those questions.

If my schema changes, will I need to re-index all of my data and files?

When you change the schema, Cloudera recommends re-indexing. For example, if you add a new field to the index, apply the new field to all index entries through re-indexing. Re-indexing is required in such a case because existing documents do yet not have the field. Cloudera Search includes a MapReduce batch-indexing solution for re-indexing and a GoLive feature that assures updated indexes are dynamically served.

Note that, you should typically re-index after adding a new field, this is not necessary if the new field applies only to new documents or data. This is because, were indexing to be completed, existing documents would still have no data for the field, making the effort unnecessary.

For schema changes that only apply to queries, re-indexing is not necessary.

Can I extract fields based on regular expressions or rules?

Cloudera Search supports limited regular expressions in Search queries. For details, see Lucene Regular Expressions.

On data ingestion, Cloudera Search supports easy and powerful extraction of fields based on regular expressions. For example the `grok` morphline command supports field extraction using regular expressions.

Cloudera Search also includes support for rule directed ETL with an extensible rule engine, in the form of the `tryRules` morphline command.

Can I use nested schemas?

Cloudera Search does not support nesting documents in this release. Cloudera Search assumes documents in the Cloudera Search repository have a flat structure.

What is Apache Avro and how can I use an Avro schema for more flexible schema evolution?

To learn more about Avro and Avro schemas, see the Avro Overview page and the Avro Specification page.

To see examples of how to implement inheritance, backwards compatibility, and polymorphism with Avro, see this InfoQ article.

Supportability

The following are questions about supportability in Cloudera Search and the answers to those questions.

Does Cloudera Search support multiple languages?

Cloudera Search supports approximately 30 languages, including most Western European languages, as well as Chinese, Japanese, and Korean.

Which file formats does Cloudera Search support for indexing? Does it support searching images?

Cloudera Search uses the Apache Tika library for indexing many standard document formats. In addition, Cloudera Search supports indexing and searching Avro files and a wide variety of other file types such as log files, Hadoop Sequence Files, and CSV files. You can add support for indexing custom file formats using a morphline command plug-in.