Important Notice

(c) 2010-2014 Cloudera, Inc. All rights reserved.

Cloudera, the Cloudera logo, Cloudera Impala, and any other product or service names or slogans contained in this document are trademarks of Cloudera and its suppliers or licensors, and may not be copied, imitated or used, in whole or in part, without the prior written permission of Cloudera or the applicable trademark holder.

Hadoop and the Hadoop elephant logo are trademarks of the Apache Software Foundation. All other trademarks, registered trademarks, product names and company names or logos mentioned in this document are the property of their respective owners. Reference to any products, services, processes or other information, by trade name, trademark, manufacturer, supplier or otherwise does not constitute or imply endorsement, sponsorship or recommendation thereof by us.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Cloudera.

Cloudera may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Cloudera, the furnishing of this document does not give you any license to these patents, trademarks copyrights, or other intellectual property.

The information in this document is subject to change without notice. Cloudera shall not be liable for any damages resulting from technical errors or omissions which may be present in this document, or from use of this document.

Cloudera, Inc.
1001 Page Mill Road Bldg 2
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 1.3.0 for CDH 4
Date: May 30, 2014
# Table of Contents

About this Guide.........................................................................................................7

Introducing Cloudera Search.....................................................................................9
  How Cloudera Search Works.....................................................................................9
  Cloudera Search Features.......................................................................................10

Understanding Cloudera Search.............................................................................13
  Cloudera Search Components..................................................................................13
  Search Architecture................................................................................................15
  Cloudera Search Tasks and Processes......................................................................16
    Ingestion..................................................................................................................16
    Indexing..................................................................................................................16
    Querying.................................................................................................................18

Cloudera Search Tutorial.........................................................................................19
  Validating the Deployment with the Solr REST API..............................................19
  Preparing to Index Data..........................................................................................20
  Batch Indexing Using MapReduce........................................................................21
    Batch Indexing into Online Solr Servers Using GoLive Feature......................21
    Batch Indexing into Offline Solr Shards.............................................................22
  Near Real Time (NRT) Indexing Using Flume and the Solr Sink..........................23
    Deploying Solr Sink into Flume Agent.................................................................23
    Configuring Flume Solr Sink...............................................................................23
    Configuring Flume Solr Sink to Sip from the Twitter Firehose.........................24
    Starting Flume Agent..........................................................................................24
    Indexing a File Containing Tweets with Flume HTTPSource..........................25
    Indexing a File Containing Tweets with Flume SpoolingDirectorySource.........25
  Using Hue with Cloudera Search..........................................................................26
    Search User Interface in Hue...............................................................................27
    Hue Search Twitter Demo....................................................................................28

Solrctl Reference....................................................................................................29

MapReduce Batch Indexing Reference..................................................................31
About this 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.

Cloudera Search documentation includes:

- Cloudera Search Release Notes
- Cloudera Search Version and Download Information
- Search Installation
- Cloudera Search Frequently Asked Questions
Introducing Cloudera Search

Cloudera Search is one of Cloudera’s near-real-time access products. Cloudera Search enables non-technical users to search and explore data stored in or ingested into Hadoop and HBase. Users do not need SQL or programming skills to use Cloudera Search because it provides a simple, full-text interface for searching.

Another benefit of Cloudera Search, compared to stand-alone search solutions, is the fully integrated data processing platform. Search uses the flexible, scalable, and robust storage system included with CDH. This eliminates the need to move larger data sets across infrastructures to address business tasks.

Cloudera Search incorporates Apache Solr, which includes Apache Lucene, SolrCloud, Apache Tika, and Solr Cell. Cloudera Search is tightly integrated with Cloudera's Distribution, including Apache Hadoop (CDH). Cloudera Search provides these key capabilities:

- Near-real-time indexing
- Batch indexing
- Simple, full-text data exploration and navigated drill down

Using Search with the CDH infrastructure provides:

- Simplified infrastructure
- Better production visibility
- Quicker insights across various data types
- Quicker problem resolution
- Simplified interaction with the ability to open the platform to more users and use cases
- Scalability, flexibility, and reliability of search services on the same platform as where you can execute other types of workloads on the same data

How Cloudera Search Works

In a near-real-time indexing use case, Cloudera Search indexes events that are streamed through Apache Flume on their way into storage in CDH. Fields and events are mapped to standard Solr indexable schemas. Lucene indexes events, and the integration through Cloudera Search allows the index to be directly written and stored in standard Lucene index files in HDFS. Flume’s capabilities to route events and have data stored in partitions in HDFS can also be applied. Events can be routed and streamed through multiple Flume agents and written to separate Lucene indexers that can write into separate index shards, for better scale when indexing and quicker responses when searching. The indexes are loaded from HDFS to Solr cores, exactly like Solr would have read from local disk. The difference in the design of Cloudera Search is the robust, distributed, and scalable storage layer of HDFS, which helps eliminate costly downtime and allows for flexibility across workloads without having to move data. Search queries can then be submitted to Solr through either the standard Solr API, or through a simple search GUI application, included in Cloudera Search, which can easily be deployed in Hue.

Cloudera Search batch-oriented indexing capabilities can address needs for searching across batch uploaded files or large data sets that are less frequently updated and less in need of near-real-time indexing. For such cases, Cloudera Search includes a highly scalable indexing workflow based on MapReduce. A MapReduce workflow is launched onto specified files or folders in HDFS, and the field extraction and Solr schema mapping is executed during the mapping phase. Reducers use Solr to write the data as a single index or as index shards, depending on your configuration and preferences. Once the indexes are stored in HDFS, they can be queried using standard Solr mechanisms, as previously described above for the near-real-time indexing use case.

The Lily HBase Indexer Service is a flexible, scalable, fault tolerant, transactional, Near Real Time (NRT) oriented system for processing a continuous stream of HBase cell updates into live search indexes. Typically the time between data ingestion using the Flume sink to that content potentially appearing in search results is on the order of seconds, though this duration is tunable. The Lily HBase Indexer uses Solr to index data stored in HBase. As HBase applies inserts, updates, and deletes to HBase table cells, the indexer keeps Solr consistent with the
Introducing Cloudera Search

HBase table contents, using standard HBase replication features. 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.

Cloudera Search Features

This section contains information about current Cloudera Search features.

Unified Management and Monitoring with Cloudera Manager

Cloudera Manager provides a unified and centralized management and monitoring experience for both CDH and Cloudera Search. Cloudera Manager simplifies deployment, configuration, and monitoring of your search services. This differs from many existing search solutions that lack management and monitoring capabilities and that fail to provide deep insight into utilization, system health, trending, and various other supportability aspects.

Index Storage in HDFS

Cloudera Search is integrated with HDFS for index storage. Indexes created by Solr/Lucene can be directly written in HDFS with the data, instead of to local disk, thereby providing fault tolerance and redundancy.

Cloudera has optimized Cloudera Search for fast read and write of indexes in HDFS while indexes are served and queried through standard Solr mechanisms. Also, because data and indexes are co-located, once data is found, processing does not require transport or separately managed storage.

Batch Index Creation through MapReduce

To facilitate index creation for large sets of data, Cloudera Search has built-in MapReduce jobs for indexing data stored in HDFS. As a result, the linear scalability of MapReduce is applied to the indexing pipeline.

Real-time and Scalable Indexing at Data Ingest

Cloudera Search provides integration with Flume to support near-real-time indexing. As new events pass through a Flume hierarchy and are written to HDFS, those same events can be written directly to Cloudera Search indexers. In addition, Flume supports routing events, filtering, and adding annotations on data on its passage to CDH. These features work with Cloudera Search for improved index sharding, index separation, and document-level access control.

Easy Interaction and Data Exploration through Hue

A Cloudera Search GUI is provided as a Hue plug-in, enabling users to interactively query data, view result files, and do faceted exploration. Hue can also schedule standing queries and explore index files. This GUI uses the Cloudera Search API, which is based on the standard Solr API.

Simplified Data Processing for Search Workloads

Cloudera Search relies on Apache Tika for parsing and preparation of many of the standard file formats for indexing. Additionally, Cloudera Search supports Avro, Hadoop Sequence, and Snappy file format mappings, as well as support for Log file formats, JSON, XML, and HTML. Cloudera Search also provides data preprocessing using Morphlines. This built-in support simplifies index configuration for these formats, which you can use for other applications such as MapReduce jobs.
HBase Search

Cloudera Search integrates with HBase, enabling full-text search of data stored in HBase. This functionality, which does not affect HBase performance, is based on a listener that monitors the replication event stream. The listener captures each write or update-replicated event, enabling extraction and mapping. The event is then sent directly to Solr indexers, deployed on HDFS, and written to indexes in HDFS, using the same process as for other indexing workloads of Cloudera Search. The indexes can then immediately be served, enabling near real time search of HBase data.
Understanding Cloudera Search

Cloudera Search opens CDH to full-text search and exploration of data in HDFS and Apache HBase. Cloudera Search is powered by Apache Solr, enriching the industry standard open source search solution with Hadoop platform integration, enabling a new generation of Big Data search. Cloudera Search makes it especially easy to query large data sets.

Understanding How Search Fits into Cloudera Offerings

Cloudera Search is another tool that fits into the broader set of solutions available for analyzing information in large data sets today. With especially large sets of data, it is neither possible to store all information reliably on a single machine nor is it possible to query such massive sets of data. CDH provides both the means to store the large data sets in existence today and the tools to query this data. At present, some of the ways data can be explored include:

- MapReduce jobs
- Cloudera Impala queries
- Cloudera Search queries

While CDH alone allows storage and access of large data sets, without Cloudera Search, users must create MapReduce jobs. This requires technical knowledge and each job can take minutes or more to run, and the longer run-times associated with MapReduce jobs can interrupt the process of exploring data. To provide a more immediate query and response experience and to eliminate the need to write MapReduce applications, Cloudera offers Real-Time Query or Impala. Impala returns results in seconds rather than minutes.

While Impala is a fast and powerful application, it uses SQL-based querying syntax. For users who are not familiar with SQL, using Impala may be challenging. To provide rapid results for less technical users, there is Cloudera Search. Impala, Hive, and Pig also require a structure, which is applied at query time, whereas Search supports free-text search over any data or fields you have indexed.

Understanding How Search Leverages Existing Infrastructure

Any data already present in a CDH deployment can be indexed and made query-able by Cloudera Search. For data that is not stored in CDH, Cloudera Search offers tools for loading data into the existing infrastructure, as well as the ability to index data as it is moved to HDFS or written to HBase.

By leveraging existing infrastructure, Cloudera Search eliminates the need to create new, redundant structures. Furthermore, Cloudera Search leverages services provided by CDH and Cloudera Manager in such a way that it does not interfere with other tasks running in the same environment. This means that you get all the benefits of reusing existing infrastructure, without the costs and problems associated with running multiple services in the same set of systems.

Cloudera Search Components

Search interacts with existing CDH components, using many of them to solve different problems. The following table lists CDH components that contribute to Search process and the ways in which each component helps:

<table>
<thead>
<tr>
<th>Component</th>
<th>Contribution</th>
<th>Applicable To</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS</td>
<td>Source documents are typically stored in HDFS. These documents are indexed and made searchable. The files that support Search such as Lucene index files and write-ahead logs are also stored in</td>
<td>All cases</td>
</tr>
</tbody>
</table>

Cloudera Search User Guide | 13
<table>
<thead>
<tr>
<th>Component</th>
<th>Contribution</th>
<th>Applicable To</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HDFS. Using HDFS provides simpler provisioning on a larger base, redundancy, and fault tolerance. As a result of using HDFS, Search servers are essentially stateless, meaning there are minimal consequences from node failures. HDFS also provides additional benefits such as snapshotting, inter-cluster replication, and disaster recovery.</td>
<td>Many cases</td>
</tr>
<tr>
<td>MapReduce</td>
<td>Search includes a pre-built MapReduce-based job. This job can be used for on-demand or scheduled indexing of any supported data set stored in HDFS. This job utilizes cluster resources for scalable batch indexing.</td>
<td>Many cases</td>
</tr>
<tr>
<td>Flume</td>
<td>Cloudera Search includes a Flume sink that enables writing events directly to indexers deployed on the cluster, enabling data indexing during ingestion.</td>
<td>Many cases</td>
</tr>
<tr>
<td>Hue</td>
<td>Cloudera Search includes a Hue frontend search application that uses standard Solr APIs is included. The application can interact with data indexed in HDFS. The application provides support for the Solr standard query language, visualization of faceted search functionality, and a typical full text search GUI-based.</td>
<td>Many cases</td>
</tr>
<tr>
<td>ZooKeeper</td>
<td>Coordinates distribution of data and metadata, also known as shards. ZooKeeper provides automatic failover, increasing service resiliency.</td>
<td>Many cases</td>
</tr>
<tr>
<td>HBase</td>
<td>Supports indexing of stored data, extracting columns, column families, and key information as fields. Because HBase does not use secondary indexing, Search can complete full text searches of content in rows and tables in HBase.</td>
<td>Some cases</td>
</tr>
<tr>
<td>Cloudera Manager</td>
<td>Deploys, configures, manages, and monitors the search processes and resource utilization across services on the cluster. Search does not require Cloudera Manager, but</td>
<td>Some cases</td>
</tr>
</tbody>
</table>
Applicable To
Contribution

<table>
<thead>
<tr>
<th>Component</th>
<th>Contribution</th>
<th>Applicable To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloudera Manager</td>
<td>helps simplify Search administration.</td>
<td></td>
</tr>
<tr>
<td>Oozie</td>
<td>Automates scheduling and management of indexing jobs. Oozie can check for new data and begin indexing jobs, as required.</td>
<td>Some cases</td>
</tr>
<tr>
<td>Impala</td>
<td>Further analyzes search results.</td>
<td>Some cases</td>
</tr>
<tr>
<td>Hive</td>
<td>Further analyzes search results.</td>
<td>Some cases</td>
</tr>
<tr>
<td>Avro</td>
<td>Includes metadata that Search can use for indexing.</td>
<td>Some cases</td>
</tr>
<tr>
<td>Sqoop</td>
<td>Ingests data in batch and enables data availability for batch indexing.</td>
<td>Some cases</td>
</tr>
<tr>
<td>Mahout</td>
<td>Applies machine learning processing to search results.</td>
<td>Some cases</td>
</tr>
</tbody>
</table>

Search Architecture

Search runs as a distributed service on a set of servers, and each server is responsible for some portion of the entire set of content to be searched. The entire set of information to be searched is split into smaller pieces, copies are made of these pieces, and the pieces are distributed among the servers. This provides two main advantages:

- **Dividing** the content into smaller pieces distributes the task of indexing the content among the servers.
- **Duplicating** the pieces of the whole allows queries to be scaled more effectively and makes it possible for the system to provide higher levels of availability.

Each Search server can handle requests for information. This means that a client can send requests to index documents or carry out searches to any arbitrary Search server and the server routes the request to the correct Search server.
Cloudera Search Tasks and Processes

For content to be searchable, it must exist in CDH and be indexed. Content can either already exist in CDH and be indexed on demand or it can be updated and indexed continuously. The first step towards making content searchable is to ensure it is ingested or stored in CDH.

Ingestion

Content can be moved to CDH through techniques such as using:
- Flume, a flexible, agent-based data ingestion framework.
- A copy utility such as `distcp` for HDFS.
- Sqoop, a structured data ingestion connector.
- `fuse-dfs`.

In a typical environment, administrators establish systems for search. For example, HDFS is established to provide storage; Flume or `distcp` are established for content ingestion. Once administrators establish these services, users can use ingestion technologies such as file copy utilities or Flume sinks.

Indexing

Content must be indexed before it can be searched. Indexing is comprised of a set of steps:
- ETL Steps: Extraction, Transformation, and Loading (ETL) is handled using existing engines or frameworks such as Apache Tika or Cloudera Morphlines.
  - Content and metadata extraction
  - Schema mapping
- Index creation: Indexes are created by Lucene.
  - Index creation
  - Index serialization

Indexes are typically stored on a local file system. Lucene supports additional index writers and readers. One such index interface is HDFS-based and has been implemented as part of Apache Blur. This index interface has been integrated with Cloudera Search and modified to perform well with CDH-stored indexes. All index data in Cloudera Search is stored in HDFS and served from HDFS.

There are three ways to index content:
Batch indexing using MapReduce

To use MapReduce to index documents, documents are first written to HDFS. A MapReduce job can then be run on the content in HDFS, producing a Lucene index. The Lucene index is written to HDFS, and this index is subsequently used by search services to provide query results.

Batch indexing is most often used when bootstrapping a search cluster. The Map component of the MapReduce task parses input into indexable documents and the Reduce component contains an embedded Solr server that indexes the documents produced by the Map. A MapReduce-based indexing job can also be configured to utilize all assigned resources on the cluster, utilizing multiple reducing steps for intermediate indexing and merging operations, with the last step of reduction being to write to the configured set of shard sets for the service. This makes the batch indexing process as scalable as MapReduce workloads.

Near Real Time (NRT) indexing using Flume

Flume events are typically collected and written to HDFS. While any Flume event could be written, logs are a common case.

Cloudera Search includes a Flume sink that includes the option to directly write events to the indexer. This sink provides a flexible, scalable, fault tolerant, near real time (NRT) system for processing continuous streams of records, creating live-searchable, free-text search indexes. Typically it is a matter of seconds from data ingestion using the Flume sink to that content potentially appearing in search results, though this duration is tunable.

The Flume sink has been designed to meet the needs of identified use cases that rely on NRT availability. Data can flow from multiple sources through multiple flume nodes. These nodes, which can be spread across a network route this information to one or more Flume indexing sinks. Optionally, you can split the data flow, storing the data in HDFS while also writing it to be indexed by Lucene indexers on the cluster. In that scenario data exists both as data and as indexed data in the same storage infrastructure. The indexing sink extracts relevant data, transforms the material, and loads the results to live Solr search servers. These Solr servers are then immediately ready to serve queries to end users or search applications.

This system is flexible and customizable, and provides a high level of scaling as parsing is moved from the Solr server to the multiple Flume nodes for ingesting new content.

Search includes parsers for a set of standard data formats including Avro, CSV, Text, HTML, XML, PDF, Word, and Excel. While many formats are supported, you can extend the system by adding additional custom parsers for other file or data formats in the form of Tika plug-ins. Any type of data can be indexed: a record is a byte array of any format and parsers for any data format and any custom ETL logic can be established.

In addition, Cloudera Search comes with a simplifying Extract-Transform-Load framework called Cloudera Morphlines that can help adapt and pre-process data for indexing. This eliminates the need for specific parser deployments, replacing them with simple commands.

Cloudera Search has been designed to efficiently handle a variety of use cases.

- Search supports routing to multiple Solr collections as a way of making a single set of servers support multiple user groups (multi-tenancy).
- Search supports routing to multiple shards to improve scalability and reliability.
- Index servers can be either co-located with live Solr servers serving end user queries or they can be deployed on separate commodity hardware, for improved scalability and reliability.
- Indexing load can be spread across a large number of index servers for improved scalability, and indexing load can be replicated across multiple index servers for high availability.

This is a flexible, scalable, highly available system that provides low latency data acquisition and low latency querying. Rather than replacing existing solutions, Search complements use-cases based on batch analysis of HDFS data using MapReduce. In many use cases, data flows from the producer through Flume to both Solr and HDFS. In this system, NRT ingestion, as well as batch analysis tools can be used.
NRT indexing using some other client that uses the NRT API

Documents written by a third-party directly to HDFS can trigger indexing using the Solr REST API. This API can be used to complete a number of steps:

1. Extract content from the document contained in HDFS where the document is referenced by a URL.
2. Map the content to fields in the search schema.
3. Create or update a Lucene index.

This could be useful if you do indexing as part of a larger workflow. For example, you might choose to trigger indexing from an Oozie workflow.

Querying

Once data has been made available as an index, the query API provided by the search service allows for direct queries to be executed, or facilitated through some third party, such as a command line tool or graphical interface. Cloudera Search provides a simple UI application that can be deployed with Hue, but it is just as easy to develop a custom application, fitting your needs, based on the standard Solr API. Any application that works with Solr is compatible and runs as a search-serving application for Cloudera Search, as Solr is the core.
Cloudera Search Tutorial

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

This tutorial describes preparatory steps of:

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

There are two tutorial topics, including indexing strategies, are:

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

These tutorials use 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. Note that the original versions of these files include many additional options. For information on all available options, including those that were not required for the tutorial, see the Solr wiki:

- SchemaXml
- SolrConfigXml

Note: Depending on which installation approach you use, Search may be installed to different locations.

- Installing Search using Cloudera Manager using parcels results in changes under /opt/cloudera/parcels.
- Installing using packages, either manually or using Cloudera Manager, results in changes to various locations throughout the file system. Common locations for changes include /usr/lib/, /etc/default/, and /usr/share/doc/.

The tutorial provides examples that work with an environment established using a package-based installation. If you installed Cloudera Search using Parcels, adjust paths accordingly.

Validating the Deployment with the Solr REST API

Validate the deployment by indexing and querying documents with the Solr REST API. Before beginning this process, you must have access to the Solr admin web console, as detailed in Deploying Cloudera Search.

Indexing Data

Begin by indexing some data to be queried later. Sample data is provided in the installed packages. Replace $SOLRHOST in the example below with the name of any host running the Solr process.

$ cd /usr/share/doc/solr-doc*/example/exampledocs

Running Queries

Once you have indexed data, you can run a query.

To run a query:

1. Open the following link in a browser: http://$SOLRHOST:8983/solr.

   Note: Replace $SOLRHOST with the name of any host running the Solr process.
2. Click the collection name in the left panel.
3. Click Query in the Menu and select execute query.

**Note:** Choose wt as json and select the indent option in the web GUI to see more human readable output.

### Next Steps

Consider indexing more data using the Solr REST API or move to batch indexing with MapReduce or NRT indexing with Flume. To learn more about Solr capabilities, consider reviewing the [Apache Solr Tutorial](https://search.apache.org/)

### Preparing to Index Data

Complete the following steps in preparation for indexing example data with MapReduce or Flume:

1. Start a SolrCloud cluster containing two servers (this example uses two shards) as described in [Deploying Cloudera Search](https://www.cloudera.com/content/cloudera/en/documentation/). Stop and continue with the next step here after running the Starting Solr in SolrCloud Mode step and verifying that the two server processes are running.

2. Generate the configuration files for the collection, including a tweet specific `schema.xml`:

   ```bash
   $ solrctl instancedir --generate $HOME/solr_configs3
   $ cp /usr/share/doc/search*/examples/solr-nrt/collection1/conf/schema.xml \ $HOME/solr_configs3/conf
   ```

3. Upload the instance directory to ZooKeeper:

   ```bash
   $ solrctl instancedir --create collection3 $HOME/solr_configs3
   ```

4. Create the new collection:

   ```bash
   $ solrctl collection --create collection3 -s 2
   ```

5. Verify the collection is live. For example, for the localhost, use `http://localhost:8983/solr/#/~cloud`

6. Prepare the configuration layout for use with MapReduce:

   ```bash
   $ cp -r $HOME/solr_configs3 $HOME/collection3
   ```

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

   ```bash
   $ 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
   ```

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

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

9. Collect HDFS/MapReduce configuration details. You can download these from Cloudera Manager or use `/etc/hadoop`, depending on your installation mechanism for the Hadoop cluster. This example uses the configuration found in `/etc/hadoop/conf.cloudera.mapreduce1`. Substitute the correct Hadoop configuration path for your cluster.
Batch Indexing Using MapReduce

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 on page 20.
- 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 Feature

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

1. Delete all existing documents in Solr.

   ```bash
   $ solrctl collection --deletedocs collection3
   ```

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

   ```bash
   $ 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 $ZKHOST:2181/solr --collection collection3 
   hdfs://$NNHOST:8020/user/$USER/indir
   ```


4. Once the job completes, try some Solr queries. For example, for `myserver.example.com`:

   ```http
   ```

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

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

- **Note:** 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 execute in the client process without submitting a job to MapReduce. Executing in the client process provides quicker turnaround during early trial and debug sessions.

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

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

  The log4j.properties file can be passed via the MapReduceIndexerTool --log4j command line option.
Batch Indexing into Offline Solr Shards

You can run the MapReduce job again, but this time without the GoLive feature. This 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.

1. Delete all existing documents in Solr.

```bash
$ solrctl collection --deletedocs collection3
$ sudo -u hdfs hadoop fs -rm -r -skipTrash /user/$USER/outdir
```

2. Run the Hadoop MapReduce job. Be sure to replace $NNHOST in the command with your NameNode hostname and port number, as required.

```bash
$ 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 \ 
$HOME/collection3 --shards 2 hdfs://$NNHOST:8020/user/$USER/indir
```

3. Check the job tracker status. For example, for the localhost, use http://localhost:50030/jobtracker.jsp.

4. Once the job completes, check the generated index files. Individual shards are written to the results directory as with names of the form part-00000, part-00001, part-00002. There are only two shards in this example.

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

5. Stop Solr on each node of the cluster.

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

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

```bash
$ hadoop fs -ls /solr/collection3
```

7. Move index shards into place.

   a. Remove outdated files:

```bash
$ sudo -u solr hadoop fs -rm -r -skipTrash \ 
/solr/collection3/$HOSTNAME1/data/index
$ sudo -u solr hadoop fs -rm -r -skipTrash \ 
/solr/collection3/$HOSTNAME2/data/index
```

   b. Ensure correct ownership of required directories:

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

   c. Move the two index shards into place.

```bash
$ sudo -u solr hadoop fs -mv /user/$USER/outdir/results/part-00000/data/index \ 
/solr/collection3/$HOSTNAME1/data/
$ sudo -u solr hadoop fs -mv /user/$USER/outdir/results/part-00001/data/index
```

Note: You are moving the index shards to the two servers you set up in Preparing to Index Data on page 20.
8. Start Solr on each node of the cluster:

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

9. Run some Solr queries. For example, for myserver.example.com, use:

```
```

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

The following section describes how to using Flume to index tweets. Before beginning this process, you must have completed the process of Preparing to Index Data.

Deploying Solr Sink into Flume Agent

Copy the configuration files:

```
$ sudo cp -r $HOME/solr_configs3 /etc/flume-ng/conf/collection3
$ 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/morphline.conf
```

Configuring Flume Solr Sink

1. Edit `/etc/flume-ng/conf/flume.conf` to specify the Flume Source details and set up the flow. You must set the relative or absolute path to the morphline configuration file:

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

2. Edit `/etc/flume-ng/conf/morphline.conf` to specify the Solr location details.
   a. Specify the `collection` configuration parameter to identify the name of the Solr Collection to use:
      ```
      collection: collection3
      ```
   b. Point the `zkHost` configuration parameter to the address of the SolrCloud ZooKeeper ensemble of the Solr collection. The format is the same as for `MapReduceIndexerTool --zk-host`. Substitute the corresponding host name for `127.0.0.1`, if necessary:
      ```
      zkHost: "127.0.0.1:2181/solr"
      ```

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

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

4. Edit `/etc/flume-ng/conf/flume-env.sh`, inserting or replacing `JAVA_OPTS` as follows:

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

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

   ```
   $ sudo bash -c 'echo "log4j.logger.org.apache.flume.sink.solr=DEBUG" >> /etc/flume-ng/conf/log4j.properties'
   ```
6. (Optional) You can configure the location at which Flume finds Cloudera Search dependencies for Flume Solr Sink using SEARCH_HOME. 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 of the form:

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

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

### Configuring Flume Solr Sink to Sip from the Twitter Firehose

Edit `/etc/flume-ng/conf/flume.conf` and replace the following properties with credentials from a valid twitter.com 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`

Generate these four codes using the Twitter developer site by completing the follows steps:

1. Sign in to [https://dev.twitter.com](https://dev.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 will not be a publicly distributed application, the name, description, and website (required fields) do not matter much except to the owner.
4. Click Create my access token at the bottom of the page. You may have to refresh 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 nodes where Flume connects to Twitter. Options for setting the system clock include installing NTP and keeping the host synchronized by running the `ntpd` service or manually synchronizing using the command `sudo ntpdate pool.ntp.org`. Confirm time is set correctly by ensuring the output of the command `date --utc` matches the time shown at [http://www.time.gov/timezone.cgi?UTC/s/0/java](http://www.time.gov/timezone.cgi?UTC/s/0/java). You can also set the time manually using the `date` command.

### Starting Flume Agent

1. Delete all existing documents in Solr:

   ```
   $ solrctl --zk collection --deletedocs collection3
   ```

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 any 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/collection3/select?q=*%3A*&sort=created_at+desc&wt=json&indent=true to verify that new tweets have been ingested into Solr. Note that 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, consider enabling TRACE log level. For example, you can 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.

Navigate 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 42.

1. Delete all existing documents in Solr:
   ```
   $ sudo /etc/init.d/flume-ng-agent stop
   $ solrctl collection --deletedocs collection3
   ```

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:
   ```
   $ curl --data-binary "@
   /usr/share/doc/search-1.3.0/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/collection3/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 SpoolingDirectorySource

SpoolingDirectorySource specifies a directory on a local disk that Flume monitors. Flume automatically transfers data from files in this directory to Solr. SpoolingDirectorySource sends data using a channel to a sink, in this case a SolrSink.
1. Delete all existing documents in Solr:

```bash
$ sudo /etc/init.d/flume-ng-agent stop
$ solrctl collection --deletedocs collection3
```

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

```bash
# Comment out "agent.sources = httpSrc"
# uncomment "agent.sources = spoolSrc"
```

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

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

4. Restart the Flume Agent:

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

5. Send a file containing tweets to the SpoolingDirectorySource. Use the copy-then-atomic-move file system trick to ensure no partial files are ingested:

```bash
$ 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.

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

7. Check the completion status.

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

Use the Cloudera Search GUI. For example, for the localhost, use http://localhost:8983/solr/collection3/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 search UI. When using Hue with Cloudera Search, involves importing collections. After you have imported collections, you can work with the Hue user interface to work with these collections.

#### Importing Collections

The following screenshot is an example of the collection import feature within Hue.
Generally, only collections should be imported. Importing cores is rarely useful since it enables querying a shard of the index. See A little about SolrCores and Collections for more information.

Search User Interface in Hue

The following screenshot is an example of the appearance of the Search application that is integrated with the Hue user interface.

Customization UI

The following screenshot is an example of the appearance of the Search application customization interface provided in Hue.
Currently, only super users can access this view.

Hue Search Twitter Demo

The demo uses similar processes to those described in the Running Queries section of the Cloudera Search Tutorial. The demo illustrates the following features:

- Only regular Solr APIs are used.
- Show facets such as fields, range, or dates; sort by time in seconds.
- Result snippet editor and preview, function for downloading, extra css/js, labels, and field picking assist.
- Show multi-collections.
- Show highlighting of search term.
- Show facet ordering.
- Auto-complete handler using /suggest.

You can watch a recording of the Hue Search Twitter Demo at Tutorial: Search Hadoop in Hue 2.4.
Solrctl Reference

Use the solrctl utility to manage a SolrCloud deployment, completing tasks such as manipulating 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, though note that different collections can use the same instance directory. Each SolrCloud collection is typically replicated (also known as sharded) among several SolrCloud instances. Each replica is called a SolrCloud core and is assigned to an individual SolrCloud node. The assignment process is managed automatically, though users can apply fine-grained control over each individual core using the core command. A typical deployment workflow with solrctl consists of deploying ZooKeeper coordination service, deploying solr-server daemons to each node, initializing the state of the ZooKeeper coordination service using init command, starting each solr-server daemon, generating an instance directory, uploading it to ZooKeeper, and associating a new collection with the name of the instance directory.

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.

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

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

The solrctl core command affects a single SolrCloud node.

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.

Syntax

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

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

Each part of these elements and their possible values are described in the following sections.

Options

If options are provided, they must precede commands:

- --solr solr_uri: Directs solrctl to a SolrCloud web API available at a given URI. This option is required for nodes running outside of SolrCloud. A sample URI might be: http://node1.cluster.com:8983/solr.
- --zk zk_ensemble: Directs solrctl to a particular ZooKeeper coordination service ensemble. This option is required for nodes running outside of SolrCloud. For example: node1.cluster.com:2181,node2.cluster.com:2181/solr.
- --help: Prints help.
- --quiet: Suppresses most solrctl messages.

Commands

- init [--force]: The init command, which initializes the overall state of the SolrCloud deployment, must be executed before starting solr-server daemons for the first time. Use this command cautiously as it is a destructive command that erases all SolrCloud deployment state information. After a successful initialization, it is impossible to recover any previous state.
- **instancedir** [\[--generate path\] [\[--create name path\] [\[--update name path\] [\[--get name path\] [\[--delete name\] [\[--list\] \]: 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 given path in a local filesystem and it has the configuration files under /conf. See Solr's README.txt for the complete layout.
  - **--create name path**: Pushes a copy of the instance directory from local filesystem to SolrCloud. If an instance directory is already known to SolrCloud, this command fails. See **--update** for changing name paths that already exist.
  - **--update name path**: Updates an existing SolrCloud's copy of an instance directory based on the files present in a local filesystem. This can be thought of as first using **--delete name** followed by **--create name path**.
  - **--get name path**: Downloads the named collection instance directory at a given 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.

- **collection** [\[--create name -s <numShards> \[-c <collection.configName>\] \[-r <replicationFactor>\] \[-m <maxShardsPerNode>\] \[-n <createNodeSet>\]] [\[--delete name\] [\[--reload name\] [\[--stat name\] [\[--list\] [\[--deletedocs name\] : Manipulates collections. The supported options have the following purpose:
  - **--create name -s <numShards> \[-c <collection.configName>\] \[-r <replicationFactor>\] \[-m <maxShardsPerNode>\] \[-n <createNodeSet>\]**: Creates a new collection name that is sharded to <numShards> SolrCloud nodes, is configured using <collection.configName> instance directory, replicated to a factor of <replicationFactor> has a maximum shards per node of <maxShardsPerNode>, and is allocated to <createNodeSet> set of nodes. The only required parameters are name and numShards. If collection.configName is not given, 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 the two commands that you can execute on a particular SolrCloud node. Use this expert command with caution. The following options are supported:
  - **--create name \[-p name=value\]...\]**: Creates a new core on a given SolrCloud node. The core is configured using name=values pairs. For more details on configuration options see Solr documentation.
  - **--reload name**: Reloads a core.
  - **--unload name**: Unloads a core.
  - **--status name**: Prints status of a core.

- **cluster** [\[--get-solrxml file\] [\[--put-solrxml file\]**: Manages cluster configuration. The following options are supported:
  - **--get-solrxml file**: Downloads the cluster configuration file solr.xml from ZooKeeper to the local system.
  - **--put-solrxml file**: Uploads the specified file to ZooKeeper as the cluster configuration file solr.xml.
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 now by installing MapReduce tools on nodes 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. It also supports merging the output shards into a set of live customer-facing Solr servers, typically a SolrCloud.

More details are available through the command line help:

```
$ hadoop jar target/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] \
   [--update-conflict-resolver FQCN] [--mappers INTEGER] \
   [--reducers INTEGER] [--max-segments INTEGER] \
   [--fair-scheduler-pool STRING] [--dry-run] [--log4j FILE] \
   [--collection STRING] [--go-live-threads INTEGER] \
   [HDFS_URI [HDFS_URI ...]]
```

The MapReduce batch job is a 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, or Excel are provided out of the box, and additional custom commands and parsers for additional file or data formats can be added as morphline plug-ins. This is done by implementing a simple Java interface that consumes a record (for example 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 morphefileField._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 omitted the first (meaning the top-most) morphline within the config file is used. 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 HadoopReducer) 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
into which to index. To use one reducer per output shard, use 0 for Search 1.x and use -2 for Search for CDH 5. Using one reducer per output shard disables the mtree merge MR algorithm. The mtree merge MR algorithm improves scalability by distributing CPU load among a set of parallel reducers that can be more numerous 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. -1 indicates use all reduce slots available on the cluster. The subsequent mapper-only phase merges the reducer output to the number of shards expected by the user, again by utilizing a cluster's parallelism. (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 and 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.
```
**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)

---

**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:

# Index an Avro based Twitter tweet file into a live SolrCloud cluster:
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

# 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/cdk-morphlines-twitter-0.9.2.jar \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file \
  ../search-core/src/test/resources/test-morphlines/tutorialReadJsonTestTweets.conf \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --zk-host zk01.mycompany.com:2181/solr \
  --collection collection1 \
  --input-list -

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_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" />
```
HdfsFindTool

HdfsFindTool is essentially the HDFS version of the Linux file system `find` command. The command walks one or more HDFS directory trees and finds all HDFS files that match the specified expression and applies selected actions to them. By default, it simply 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.

More details are available through the command line help:

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

- `exec command [argument ...]`
- `ok command [argument ...]`
  Executes the specified Hadoop shell command with the given arguments. If the string `{}` is given as an argument then it is replaced by the current path name. If a `{}` argument is followed by a + character then multiple paths will be batched up and passed to a single execution of the command. A maximum of 500 paths will be passed to a single
command. The expression evaluates to true if the command
returns success and false if it fails.
If -ok is specified then confirmation of each command shall be
prompted for on STDERR prior to execution. If the response is
'y' or 'yes' then the command shall be executed else the command
shall not be invoked and the expression shall return false.

-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
-name 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 theclasspath.
-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) oriented system for processing a continuous stream of records into live search indexes. Latency from the time of data arrival to the time of data showing up in search query results is on the order of seconds and is tunable.

Data flows from one or more sources through one or more Flume nodes across the network to one or more Flume Solr Sinks. The Flume Solr 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 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, or Excel. are provided out of the box, and additional custom commands and parsers for additional file or data formats can be added as morphline plug-ins. This is done by implementing a simple Java interface that consumes a record such as a file in the form of an InputStream plus some headers plus contextual metadata. This record is used to generate output of 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.

Routing to multiple Solr collections is supported to improve multi-tenancy. Routing to a SolrCloud cluster is supported to improve 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. Indexing load can be spread across a large number of Flume SolrSink servers for improved 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.

This system provides low latency data acquisition and low latency querying. It complements (rather than 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 Flume features such as optional replicating channels to replicate an incoming flow into two output flows. Both near real time ingestion as well as batch analysis tools are used in practice.

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

Once Flume is configured, start Flume as detailed in Flume Installation.

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

Flume Morphline Solr Sink Configuration Options

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

Flume Morphline Solr Sink provides the following configuration options in the flume.conf file:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td></td>
<td>The FQCN of this class: org.apache.flume.sink.solr.morphline.MorphlineSolrSink</td>
</tr>
<tr>
<td>batchSize</td>
<td>100</td>
<td>The maximum number of events to take per flume transaction.</td>
</tr>
<tr>
<td>batchDurationMillis</td>
<td>1000</td>
<td>The maximum duration per flume transaction (ms). The transaction</td>
</tr>
</tbody>
</table>
Flume Morphline Interceptor Configuration Options

Flume has the capability to modify and drop events in-flight. This is done with the help of a **Interceptors**, which can be attached to any Flume Source. Flume MorphlineInterceptor is an Interceptor that executes the transformations of a morphline on events that are intercepted. 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. For example, this kind of packet sniffing can be used for content based routing in a Flume topology.

Flume supports multiplexing the event flow to one or more destinations. This is achieved by defining a flow multiplexer that can replicate or selectively route an event to one or more 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 case of replicating flow, each event is sent to all three channels. For the multiplexing case, 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”, then it should go to channel1 and channel3. If
the attribute is set to “avro/binary”, then it should go to channel2, otherwise channel3. You can set the mapping in the `flume.conf` file.

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

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td></td>
<td>The FQCN of this class: <code>org.apache.flume.sink.solr.morphline.MorphlineInterceptor$Builder</code></td>
</tr>
<tr>
<td>morphlineFile</td>
<td>n/a</td>
<td>The location of the morphline configuration file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• In a Cloudera Managed Deployment, use: <code>agent.sources.avroSrc.interceptors.morphlineinterceptor.morphlineFile = morphlines.conf</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• In unmanaged deployments, provide the relative or absolute path on the local file system to the morphline configuration file. For example, <code>/etc/flume-ng/conf/morphline.conf</code>.</td>
</tr>
<tr>
<td>morphlineId</td>
<td>null</td>
<td>Name used to identify a morphline if there are multiple morphlines in a morphline config file</td>
</tr>
</tbody>
</table>

For example, here is 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: Currently a morphline interceptor can not generate more than one output record for each input event.

Flume Solr UUIDInterceptor Configuration Options

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

Consider using UUIDInterceptor to automatically assign a UUID to a (document) event if no application level unique key for the event is available. It is important to assign UUIDs to events as soon as they enter the Flume network; that is, in the first Flume Source of the flow. This enables subsequent deduplication of documents in the face of replication and redelivery in a Flume network that is designed for high availability and high performance. If an application level key is available, this is preferable to an auto-generated UUID because it enables subsequent updates and deletes of the document in Solr using that well-known application-level key.

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

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td></td>
<td>The FQCN of this class: <code>org.apache.flume.sink.solr.morphline.UUIDInterceptor$Builder</code></td>
</tr>
</tbody>
</table>
Flume Near Real-Time Indexing Reference

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>headerName</td>
<td>id</td>
<td>The name of the Flume header to modify.</td>
</tr>
<tr>
<td>preserveExisting</td>
<td>true</td>
<td>If the UUID header already exists, determine whether it is preserved.</td>
</tr>
<tr>
<td>prefix</td>
<td>&quot;&quot;</td>
<td>The prefix string constant to prepend to each generated UUID.</td>
</tr>
</tbody>
</table>

For examples, see the [BlobHandler](#) and [BlobDeserializer](#).

Flume Solr BlobHandler Configuration Options

Flume has the capability to accept Flume Events by HTTP POST and GET. This is done with the help of [HTTPSource](#). By default HTTPSource splits JSON input into Flume events. As an alternative, Flume Solr BlobHandler is a handler for HTTPSource that returns an event that contains the request parameters as well as the Binary Large Object (BLOB) uploaded with this request. Note that 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:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler</td>
<td>org.apache.flume.sink.solr.morphline.BlobHandler</td>
<td>The FQCN of this class:</td>
</tr>
<tr>
<td></td>
<td>100000000 (100 MB)</td>
<td>The maximum number of bytes to read and buffer for a given request.</td>
</tr>
</tbody>
</table>

For example, here is a `flume.conf` section for a HTTPSource with a BlobHandler for the agent named “agent”:

```plaintext
agent.sources.httpSrc.port = 5140
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

Flume can ingest data from files placed in a spooling directory on disk. This is done with the help of [SpoolDirectorySource](#). Unlike other asynchronous sources, SpoolDirectorySource avoids data loss 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. If this is not desirable, Flume Solr BlobDeserializer can read Binary Large Objects (BLOBs) from SpoolDirectorySource. Note that 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:
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.

For example, here is a `flume.conf` section for a SpoolDirectorySource with a BlobDeserializer for the agent named "agent":

```plaintext
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$Builder
agent.sources.spoolSrc.deserializer.maxBlobLength = 200000000
agent.sources.spoolSrc.batchSize = 1
agent.sources.spoolSrc.fileHeader = true
agent.sources.spoolSrc.fileHeaderKey = resourceName
agent.sources.spoolSrc.interceptors = uuidinterceptor
agent.sources.spoolSrc.interceptors.uuidinterceptor.type = org.apache.flume.sink.solr.morphline.UUIDInterceptor$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
```
Cloudera Morphlines is an open source framework that reduces the time and skills necessary to build or change Search indexing applications. A morphline is a rich configuration file that simplifies defining an ETL transformation chain. These transformation chains support consuming any kind of data from any kind of data source, processing the data, and loading 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, for example as outlined in the following flow diagram:

![Flow Diagram](image)

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 the adding of new transformations and I/O commands and integrates existing functionality and third party systems in a straightforward manner.

This integration enables 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, as well as reuse of ETL logic building blocks across Search applications.

Cloudera ships a high performance runtime that compiles a morphline as required. The runtime processes all commands of a given morphline in the same thread, adding no artificial overhead. For high scalability, you can deploy many morphline instances on a cluster in many Flume agents and MapReduce tasks.

Currently there are three components that execute morphlines:
- MapReduceIndexerTool
- Flume Morphline Solr Sink and Flume MorphlineInterceptor

Cloudera also provides a corresponding Cloudera Search Tutorial.

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_mime type (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.
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:

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 node 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, nodes, 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](https://github.com/mroach/hocon) and is also used by [Akka](https://akka.io) and [Play](https://www.playframework.com).

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

This section provides some examples of how one might use morphlines.

#### Using Morphlines to Index Avro

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

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

   ```bash
   $ 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:

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

   ```json
   {
   "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 fields named `id`, `user_screen_name`, `created_at` and `text` from the given Avro records, then store and index them in Solr, using the following Solr schema definition in `schema.xml`:

   ```xml
   <fields>
   <field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" />
   ```
Note that the Solr output schema omits some Avro input fields such as `user_statuses_count`. Suppose you want 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`.

1. These transformation rules can be expressed with morphline commands called `readAvroContainer`, `extractAvroPaths`, `convertTimestamp`, `sanitizeUnknownSolrFields` and `loadSolr`, by editing a morphline.conf file.

   Note: The following example uses the Kite SDK, which applies to Search for CDH 5. In the case of morphlines.conf files used with Search 1.3 and earlier, which uses CDK, the importCommands are different.

For the following morphlines.conf file to apply to CDK, you would replace `importCommands`:

```
["org.kitesdk.**, "org.apache.solr."]
```

with `importCommands`:

```
["com.cloudera.**, "org.apache.solr."]
```

Note that the Solr output schema omits some Avro input fields such as `user_statuses_count`. Suppose you want 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`.

1. These transformation rules can be expressed with morphline commands called `readAvroContainer`, `extractAvroPaths`, `convertTimestamp`, `sanitizeUnknownSolrFields` and `loadSolr`, by editing a morphline.conf file.

   Note: The following example uses the Kite SDK, which applies to Search for CDH 5. In the case of morphlines.conf files used with Search 1.3 and earlier, which uses CDK, the importCommands are different.

For the following morphlines.conf file to apply to CDK, you would replace `importCommands`:

```
["org.kitesdk.**, "org.apache.solr."]
```

with `importCommands`:

```
["com.cloudera.**, "org.apache.solr."]
```

# 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.
    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
# expressions to extract values from an Avro object. Each
# expression 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 expression is a list of objects, each of which
# is added to the given record output field.
# The path language supports all Avro concepts, including nested
# structures, records, arrays, maps, unions, and others, as well as a
# flatten option that collects the primitives in a subtree into a flat list. In
# the Solr read path specification, entries on the left of the colon are the target
# 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}
}
Using Morphlines with Syslog

This section provides a sample that 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 should extract the following record from the log line and load it into Solr:

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

The following rules can be used to create a chain of transformation commands which are expressed with morphline commands called `readLine`, `grok`, and `logDebug` by editing a `morphline.conf` file.

**Note:** The following example uses the Kite SDK, which applies to Search for CDH 5. In the case of `morphlines.conf` files used with Search 1.3 and earlier, which uses CDK, the `importCommands` are different.

For the following `morphlines.conf` file to apply to CDK, you would replace `importCommands : ["org.kitesdk.**"]` with `importCommands : ["com.cloudera.**"]`.

```
# 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 { 50
```

---

Extracting, Transforming, and Loading Data With Cloudera Morphlines
Next Steps

Learn more about Morphlines and Kite. Cloudera Search 1.3.0 includes CDK version 0.9.1. For more information, see:

- [CDK Morphlines Reference Guide](#).
- More example morphlines can be found in the [unit tests](#).
Using the Lily HBase Batch Indexer for Indexing

Cloudera Search provides the ability to batch index HBase tables using MapReduce jobs. Such batch indexing does not require:

- The HBase replication feature
- 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 SolrCloud collection
- Creating a Lily HBase Indexer configuration
- Creating a Morphline configuration file
- Understanding the `extractHBaseCells` morphline command
- Running HBaseMapReduceIndexerTool

Populating an HBase table

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

```bash
$ 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 SolrCloud collection

A SolrCloud collection 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 SolrCloud collection using a command of the form:

```bash
$ 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

Individual Lily HBase Indexers are configured 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 and column families and corresponding collections in the SolrCloud. Each Lily HBase Indexer configuration is defined in an XML file such as `morphline-hbase-mapper.xml`.

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

```xml
<?xml version="1.0"?>
```
The Lily HBase Indexer configuration file also supports the standard attributes of any HBase Lily Indexer on the top-level `<indexer>` element, meaning the attributes `table`, `mapping-type`, `read-row`, `unique-key-formatter`, `unique-key-field`, `row-field`, `column-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, the first such command is an `extractHBaseCells` 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.

Note: The following example uses the Kite SDK, which applies to Search for CDH 5. In the case of `morphlines.conf` files used with Search 1.3 and earlier, which uses CDK, the `importCommands` are different.

For the following `morphlines.conf` file to apply to CDK, you would replace `importCommands`:

```
["org.kitesdk.morphline.", "com.ngdata."]
```
with `importCommands`:

```
["com.cloudera.cdk.morphline.", "com.ngdata."]
```

$ 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 {} 
      }
    ]
  }
]
```

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 to be used from HBase for populating a field in Solr. It takes 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:
    ```java
    inputColumn : "m:e:*"
    outputField : "belongs_to_*"
    ```
    In this case, if you make these puts in HBase:
    ```java
    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:
    ```java
    belongs_to_1 : foo
    belongs_to_9 : bar
    ```
  - The `type` parameter defines the datatype 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 from byte arrays to the actual datatype is required. The type parameter can be the name of a type that is supported by `org.apache.hadoop.hbase.util.Bytes.toXXX` (currently: "byte[]", "int", "long", "string", "boolean", "float", "double", "short", "bigdecimal"). Use type "byte[]" to pass the byte array through to the morphline without any 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`
Using the Lily HBase Batch Indexer for Indexing

- 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

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

- The source parameter determines what portion of an HBase KeyValue is used as indexing input. Valid choices are "value" or "qualifier". When "value" is specified, then 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 the HBaseMapReduceIndexerTool to index the HBase table using a MapReduce job, as follows:

```shell
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 execute in the client process without submitting a job to MapReduce. Executing in the client process provides quicker turnaround during early trial and debug sessions.

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

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

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

HBaseMapReduceIndexerTool

HBaseMapReduceIndexerTool is a 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.

More details are available through the command line help:

```shell
$ 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-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]
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/or 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
  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 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:
**Optional arguments:**

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

- **--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

---

Using the Lily HBase Batch Indexer for Indexing
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)

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
--go-live --log4j src/test/resources/log4j.properties

# (Re)index a table in GoLive mode using a local morphline-based 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 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
--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
--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
Configuring 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) oriented system for processing a continuous stream of HBase cell updates into live search indexes. Typically it is a matter of seconds from data ingestion into HBase to that content potentially appearing in search results, though 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 features. 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 The Lily HBase Indexer Service in Managing Clusters with Cloudera Manager for more information.

Enabling cluster-wide HBase replication

The Lily HBase Indexer is implemented using HBase replication, presenting indexers as region servers of the slave 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 presented 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 which does not need to be added. For example, you could do this using the Cloudera Manager HBase Indexer Service GUI. After making these updates, restart your HBase cluster.

Pointing an Lily HBase NRT Indexer Service at an HBase cluster that needs to be indexed

Configure individual Lily HBase NRT Indexer Services with the location of a ZooKeeper ensemble that is used for the target HBase cluster. This must be done before starting Lily HBase NRT Indexer Services. Add the following property to /etc/hbase-solr/conf/hbase-indexer-site.xml. Remember to replace hbase-cluster-zookeeper with the actual ensemble string as found in hbase-site.xml configuration file:

```xml
<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 among each other. Add the following property to /etc/hbase-solr/conf/hbase-indexer-site.xml. Remember to replace hbase-cluster-zookeeper:2181 with the actual ensemble string:

```xml
<property>
  <name>hbaseindexer.zookeeper.connectstring</name>
  <value>hbase-cluster-zookeeper:2181</value>
</property>
```
Starting an 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 an 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, you can 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 an 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

Configuring indexing for column families of tables in an HBase cluster requires:

- Enabling replication on HBase column families
- Creating collections and configurations
- Registering a Lily HBase Indexer configuration with the Lily HBase Indexer Service
- Verifying the 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. Do this by issuing a command of the form:

```
$ hbase shell
hbase shell> disable 'record'
```

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

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

Creating collections and configurations

Complete three tasks related to creating a collection and configurations. The tasks required for the Lily HBase NRT Indexer Services are the same as those described for the Lily HBase Batch Indexer. Therefore, follow the steps described in these sections:

- [Creating a corresponding SolrCloud collection](#) on page 53
- [Creating a Lily HBase Indexer configuration](#) on page 53
- [Creating a Morphline Configuration File](#) on page 54
Registering a Lily HBase Indexer configuration with the Lily HBase Indexer Service

Once the content of the Lily HBase Indexer configuration XML file is satisfactory, register it with the Lily HBase Indexer Service. This is done with a given SolrCloud collection 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
```

Existing Lily HBase Indexers can be further manipulated by using the update-indexer and delete-indexer command line options of the hbase-indexer utility.

For more help use the following help commands:

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

**Note:** The morphlines.conf configuration file must be present on every node that runs an indexer.

**Note:** The morphlines.conf configuration file can be updated using the Cloudera Manager Admin Console.

To update morphlines.conf using Cloudera Manager

1. On the Cloudera Manager Home page, click the Key-Value Indexer Store, often KS_INDEXER-1.
2. Click Configuration > View and Edit.
3. Expand Service-Wide and click Morphlines.
4. For the Morphlines File property, paste the new morphlines.conf content into the Value field.

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.

Cloudera Search User Guide | 65
Verifying the indexing is working

Add rows to the indexed HBase table. For example:

```bash
$ 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, then navigate to the SolrCloud's 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, consider enabling 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 4, this can be done by navigating to Services > KS_INDEXER > Configuration > View and Edit > Lily HBase Indexer > Advanced > Lily HBase Indexer Logging Safety Valve, followed by a restart of the Lily HBase Indexer Service.

**Note:** Prior to Cloudera Manager 4.8, the service was referred to as **Keystore Indexer service**.

In Cloudera Manager 5, this can be done by navigating to Clusters > KS_INDEXER-1 > Configuration > View and Edit > Lily HBase Indexer > Advanced > Lily HBase Indexer Logging Safety Valve, followed by a restart of the Lily HBase Indexer Service.

**Note:** The name of the particular key-value store indexer may 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 Search to Use Kerberos

Cloudera Search 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 Search in a Cloudera managed environment.

To create principals and keytabs

Repeat this process on all Solr server nodes.

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 nodes.

1. Ensure that the following properties appear in /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. 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.

```java
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>";
}
```

3. To use short principal names:
   
   - Appendix C - Configuring the Mapping from Kerberos Principals to Short Names in the CDH 4 Security Guide.
Using Kerberos

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

- **Using Kerberos and curl** on page 69
- **Using solrctl** on page 69
- **Configuring SolrJ Library Usage** on page 69

This enables technologies including:

- Command line solutions
- Java applications
- The MapReduceIndexerTool

- **Configuring Flume Morphline Solr Sink Usage** on page 70


### 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 execute the desired command. For example, you might use commands similar to the following:

```
$ kinit -kt username.keytab username
```

**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](https://docs.cloudera.com/Clinical-Guides/CDH-5/Security-Guide/GUID-DFF5261E-5EF2-412E-92A7-961620CD32E5.html) on page 29.

### 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:

```java
Client {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=false
    useTicketCache=true
}
```
Using Kerberos

```java
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:

```java
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 wish to use and `user/fully.qualified.domain.name@<YOUR-REALM>` is the principal in that keytab you wish 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 a jar, you might use:

```bash
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:

```java
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:

```bash
HADOOP_OPTS="-Djava.security.auth.login.config=/home/user/jaas.conf" \
hadoop jar MapReduceIndexerTool
```

Configuring Flume Morphline Solr Sink Usage

Repeat this process on all Flume nodes:

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 the CDH 5 Security Guide 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:

```java
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 for Search

Sentry enables role-based, fine-grained authorization for Cloudera Search. Follow the instructions below to configure Sentry under CDH 4.5 or later or CDH 5. Sentry is included in the Search installation.

Note: Sentry for Search depends on Kerberos authentication. For additional information on using Kerberos with Search, see Configuring Search to Use Kerberos on page 67 and Using Kerberos on page 69.

Note that this document is for configuring Sentry for Cloudera Search. For information about alternate ways to configure Sentry or for information about installing Sentry for other services, see:

- Setting Up Search Authorization with Sentry for instructions for using Cloudera Manager 4 to install and configure Search Authorization with Sentry.
- Impala Security for instructions on using Impala with Sentry.
- Sentry Installation to install the version of Sentry that was provided with CDH 4.
- Sentry Installation to install the version of Sentry that was provided with CDH 5.

Roles and Collection-Level Privileges

Sentry uses a role-based privilege model. A role is a set of rules for accessing a given Solr collection. Access to each collection is governed by privileges: Query, Update, or All (*).

For example, a rule for the Query privilege on collection logs would be formulated as follows:

```
collection=logs->action=Query
```

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
```

Users and Groups

- 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 use either Hadoop groups or local groups, but not both 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:


**Note:** Note that, by default, this uses local shell groups. See the Group Mapping section of the HDFS Permissions Guide for more information.

**OR**

To configure local groups:

1. Define local groups in a `[users]` section of the Sentry Configuration File on page 75, `sentry-site.xml`. 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>
   ```

**Setup and Configuration**

This release of Sentry stores the configuration as well as privilege policies in files. The `sentry-site.xml` file contains configuration options such as privilege policy file location. The Policy file on page 74 contains the privileges and groups. It has a `.ini` file format and should be stored on HDFS.

Sentry is automatically installed when you install Cloudera Search for CDH or Cloudera Search 1.1.0 or later.

**Policy file**

The sections that follow contain notes on creating and maintaining the policy file.

**Warning:** An invalid configuration disables all authorization while logging an exception.

**Storing the Policy File**

Considerations for storing the policy file(s) include:

1. Replication count - Because the file is read 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, the following results in role1 having privilege2, not privilege1 and privilege2.

\[
\text{role1} = \text{privilege1} \\
\text{role1} = \text{privilege2}
\]

Sample Configuration

This section provides a sample configuration.

**Note:** Sentry with CDH 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 CDH Search has no such support for multiple policies.

Policy File

The following is an example of a CDH Search policy file. The `sentry-provider.ini` would exist in an HDFS location such as `hdfs://ha-nn-uri/user/solr/sentry/sentry-provider.ini`.

```ini
[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

[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
```

Sentry Configuration File

The following is an example of a `sentry-site.xml` file.

```xml
<configuration>
    <property>
        <name>hive.sentry.provider</name>
    </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>
</configuration>
```
Enabling Secure Impersonation

Secure Impersonation is a feature that allows a user to make requests as another user in a secure way. 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".

Configure the following properties in `/etc/default/solr`:

```
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 in order to impersonate, and **SOLR_SECURITY_PROXYUSER_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.

Debugging Failed Sentry Authorization Requests

Sentry logs all facts that lead up to authorization decisions at the debug level. If you do not understand why Sentry is denying access, the best way to debug is to temporarily turn on debug logging:

- In Cloudera Manager, add `log4j.logger.org.apache.sentry=DEBUG` to the logging settings for your service through the corresponding **Logging Safety Valve** field for the Impala, Hive Server 2, or Solr Server services.
- On systems not managed by Cloudera Manager, add `log4j.logger.org.apache.sentry=DEBUG` to the `log4j.properties` file on each host in the cluster, in the appropriate configuration directory for each service.

Specifically, look for exceptions and messages such as:

```
FilePermission server..., RequestPermission server...., result [true|false]
```

which indicate each evaluation Sentry makes. The FilePermission is from the policy file, while RequestPermission is the privilege required for the query. A RequestPermission will iterate over all appropriate FilePermission settings until a match is found. If no matching privilege is found, Sentry returns false indicating “Access Denied”.

Appendix: Authorization Privilege Model for Search

The tables below refer to the request handlers defined in the generated `solrconfig.xml.secure`. If you are not using this configuration file, the below may not apply.

**admin** is a special collection in Sentry used to represent administrative actions. A non-administrative request may only require privileges on the collection on which the request is being performed. This is called **collection1** in this appendix. An administrative request may require privileges on both the **admin** collection and **collection1**. This is denoted as **admin, collection1** in the tables below.
Table 1: Privilege table for non-administrative request handlers

<table>
<thead>
<tr>
<th>Request Handler</th>
<th>Required Privilege</th>
<th>Collections that Require Privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>select</td>
<td>QUERY</td>
<td>collection1</td>
</tr>
<tr>
<td>query</td>
<td>QUERY</td>
<td>collection1</td>
</tr>
<tr>
<td>get</td>
<td>QUERY</td>
<td>collection1</td>
</tr>
<tr>
<td>browse</td>
<td>QUERY</td>
<td>collection1</td>
</tr>
<tr>
<td>tvrh</td>
<td>QUERY</td>
<td>collection1</td>
</tr>
<tr>
<td>clustering</td>
<td>QUERY</td>
<td>collection1</td>
</tr>
<tr>
<td>terms</td>
<td>QUERY</td>
<td>collection1</td>
</tr>
<tr>
<td>elevate</td>
<td>QUERY</td>
<td>collection1</td>
</tr>
<tr>
<td>analysis/field</td>
<td>QUERY</td>
<td>collection1</td>
</tr>
<tr>
<td>analysis/document</td>
<td>QUERY</td>
<td>collection1</td>
</tr>
<tr>
<td>update</td>
<td>UPDATE</td>
<td>collection1</td>
</tr>
<tr>
<td>update/json</td>
<td>UPDATE</td>
<td>collection1</td>
</tr>
<tr>
<td>update/csv</td>
<td>UPDATE</td>
<td>collection1</td>
</tr>
</tbody>
</table>

Table 2: Privilege table for collections admin actions

<table>
<thead>
<tr>
<th>Collection Action</th>
<th>Required Privilege</th>
<th>Collections that Require Privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>delete</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>reload</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>createAlias</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>deleteAlias</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>syncShard</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
</tbody>
</table>

**Note:** "collection1" here refers to the name of the alias, not the underlying collection(s). For example, http://YOUR-HOST:8983/solr/admin/collections?action=CREATE_ALIAS&name=collection1&collections=underlyingCollection
### Table 3: Privilege table for core admin actions

<table>
<thead>
<tr>
<th>Collection Action</th>
<th>Required Privilege</th>
<th>Collections that Require Privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>splitShard</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>deleteShard</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>create</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>rename</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>load</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>unload</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>status</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>persist</td>
<td>UPDATE</td>
<td>admin</td>
</tr>
<tr>
<td>reload</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>swap</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>mergeIndexes</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>split</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>prepRecover</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>requestRecover</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>requestSyncShard</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
<tr>
<td>requestApplyUpdates</td>
<td>UPDATE</td>
<td>admin, collection1</td>
</tr>
</tbody>
</table>

### Table 4: Privilege table for Info and AdminHandlers

<table>
<thead>
<tr>
<th>Request Handler</th>
<th>Required Privilege</th>
<th>Collections that Require Privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>LukeRequestHandler</td>
<td>QUERY</td>
<td>admin</td>
</tr>
<tr>
<td>SystemInfoHandler</td>
<td>QUERY</td>
<td>admin</td>
</tr>
<tr>
<td>SolrInfoMBeanHandler</td>
<td>QUERY</td>
<td>admin</td>
</tr>
<tr>
<td>PluginInfoHandler</td>
<td>QUERY</td>
<td>admin</td>
</tr>
<tr>
<td>ThreadDumpHandler</td>
<td>QUERY</td>
<td>admin</td>
</tr>
<tr>
<td>PropertiesRequestHandler</td>
<td>QUERY</td>
<td>admin</td>
</tr>
<tr>
<td>LogginHandler</td>
<td>QUERY, UPDATE (or *)</td>
<td>admin</td>
</tr>
<tr>
<td>ShowFileRequestHandler</td>
<td>QUERY</td>
<td>admin</td>
</tr>
</tbody>
</table>
High Availability and Fault Tolerance

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`:

```properties
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 optional replicating channels 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 nodes.

- **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 node names in your cluster. Begin by shutting down all services in the cluster.

**Prerequisites**

Before changing node names, back up the Cloudera Manager database using a tool such as `mysqldump`. For more information, see the [MySQL Reference Manual](https://dev.mysql.com/doc/refman/8.0/en/). 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 nodes 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
   ```
Renaming Cloudera Manager Managed Hosts

Editing the server_host Value

If you are renaming the node running Cloudera Manager, you must edit the server_host value in the config.ini file on all nodes 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 node 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 nodes that are managed by Cloudera Manager.

Updating Name Services

Update the names of the nodes using the name service method that applies for your operating system.

1. Edit the network or hostname file.

   For Red Hat-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 nodes 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:

```bash
# 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 nodes whose names you changed.

   **RHEL-compatible or SLES systems:**
   ```bash
   $ sudo service cloudera-scm-agent start
   ```

   **Debian/Ubuntu systems:**
   ```bash
   $ sudo /usr/sbin/service cloudera-scm-agent start
   ```

2. Start the Cloudera Manager Server.

   **RHEL-compatible or SLES systems:**
   ```bash
   $ sudo service cloudera-scm-server start
   ```

   **Debian/Ubuntu systems:**
   ```bash
   $ 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 FailoverController (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 node:

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 nodes 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:

   ```bash
   $ rm -f /usr/lib/zookeeper/bin/zkCli.sh
   ```
Renaming Cloudera Manager Managed Hosts

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, click the service and click **Configuration**.
2. Edit the **Database Hostname** value so it reflects the new hostname.

Returning the System to a Running State

Now that you have renamed nodes 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 Redeploying Client 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.

General information on Solr caching is available here on the [SolrCaching](#) page on the Solr Wiki.

Information on issues that influence performance is available on the [SolrPerformanceFactors](#) page on the Solr Wiki.

**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`:

```xml
<luceneMatchVersion>4.4</luceneMatchVersion>
```

**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 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 make use of the cache, while Solr servers (when serving queries or indexing documents) should. When running indexing using MapReduce, the MR jobs themselves do not make use of 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>solr.hdfs.blockcache.enabled</code></td>
<td>true</td>
<td>Enable the blockcache.</td>
</tr>
<tr>
<td><code>solr.hdfs.blockcache.read.enabled</code></td>
<td>true</td>
<td>Enable the read cache.</td>
</tr>
<tr>
<td><code>solr.hdfs.blockcache.write.enabled</code></td>
<td>false</td>
<td>Enable the write cache.</td>
</tr>
<tr>
<td><code>solr.hdfs.blockcache.direct.memory.allocation</code></td>
<td>true</td>
<td>Enable direct memory allocation. If this is false, heap is used.</td>
</tr>
<tr>
<td><code>solr.hdfs.blockcache.slab.count</code></td>
<td>1</td>
<td>Number of memory slabs to allocate. Each slab is 128 MB in size.</td>
</tr>
</tbody>
</table>
Tuning the Solr Server

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>solr.hdfs. \ blockcache.global</td>
<td>true</td>
<td>If enabled, a single HDFS block cache is used for all SolrCores on a node. If blockcache.global is disabled, each SolrCore on a node creates its own private HDFS block cache. Enabling this parameter simplifies managing HDFS block cache memory.</td>
</tr>
</tbody>
</table>

**Note:**
Increasing the direct memory cache size may make it necessary to increase the maximum direct memory size allowed by the JVM. Add the following to /etc/default/solr to do so. You must also replace MAXMEM with a reasonable upper limit. A typical default JVM value for this is 64 MB.

CATALINA_OPTS="-XX:MaxDirectMemorySize=MAXMEMg -XX:+UseLargePages"

Restart Solr servers after editing this 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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>solr.hdfs. \ nrtcachingdirectory.enable</td>
<td>true</td>
<td>Whether to enable the NRTCachingDirectory.</td>
</tr>
<tr>
<td>solr.hdfs. \ nrtcachingdirectory.maxcachedmb</td>
<td>192</td>
<td>Size of the cache in megabytes.</td>
</tr>
<tr>
<td>solr.hdfs. \ nrtcachingdirectory.maxmergesizemb</td>
<td>16</td>
<td>Maximum segment size to cache.</td>
</tr>
</tbody>
</table>

Here is an example of solrconfig.xml with defaults:

```xml
<directoryFactory name="DirectoryFactory"
  class="org.apache.solr.core.HdfsDirectoryFactory">
  <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">$({solr.hdfs.blockcache.direct.memory.allocation:true})</bool>
  <int name="solr.hdfs.blockcache.blocksperslab">$({solr.hdfs.blockcache.blocksperslab: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">$({solr.hdfs.nrtcachingdirectory.enable:true})</bool>
  <int name="solr.hdfs.nrtcachingdirectory.maxcachedmb">$({solr.hdfs.nrtcachingdirectory.maxcachedmb:192})</int>
  <int name="solr.hdfs.nrtcachingdirectory.maxmergesizemb">$({solr.hdfs.nrtcachingdirectory.maxmergesizemb:16})</int>
</directoryFactory>
```
The following example illustrates passing Java options by editing the /etc/default/solr configuration file:

```
CATALINA_OPTS="-Xmx10g -XX:MaxDirectMemorySize=20g -XX:+UseLargePages
-Dsolr.hdfs.blockcache.slab.count=100"
```

For better performance, Cloudera recommends disabling the Linux swap space on all Solr server nodes as shown below:

```
# minimize swapiness
sudo sysctl vm.swappiness=0
sudo bash -c 'echo "vm.swappiness=0">> /etc/sysctl.conf'
# disable swap space until next reboot:
sudo /sbin/swapoff -a
```

Solr Query Performance

The ImproveSearchingSpeed on the Lucene-java Wiki highlights some areas to consider for improving query performance.

Solr Indexing Performance

The ImproveIndexingSpeed on the Lucene-java Wiki highlights some areas to consider for improving indexing performance.

Resource Management with Cloudera Manager

Resource Management describes how to use Cloudera Manager to manage resources, for example with Linux cgroups.
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` for the port if you are in doubt.

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Explanation</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Varied</td>
<td>Examine Solr log. By default, the log can be found at <code>/var/log/solr/solr.out</code>.</td>
</tr>
<tr>
<td>No documents found</td>
<td>Server may not be running</td>
<td>Browse to http://<a href="">server:port</a>/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.</td>
</tr>
<tr>
<td>No documents found</td>
<td>Core may not have documents</td>
<td>Browsing http://<a href="">server:port</a>/solr/collectonname/select?q=<em>:</em>&amp;wt=json&amp;indent=true should show numFound, which is near the top, to be more than 0.</td>
</tr>
<tr>
<td>The secure Solr Server fails to respond to Solrj requests, but other clients such as curl can communicate successfully</td>
<td>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.</td>
<td>Ensure your application is using commons-codec 1.7 or later. 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.</td>
</tr>
</tbody>
</table>

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 executed 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 autowarming 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. It is uncommon for user to go beyond the first page and very rare 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”.

Troubleshooting Cloudera Search
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.