cloudera®

Cloudera Search

Important Notice

Cloudera, Inc.
395 Page Mill Road
Palo Alto, CA 94306
info@cloudera.com
US: 1-888-789-1488
Intl: 1-650-362-0488
www.cloudera.com

Release Information

Version: Cloudera Search 6.1.x
Date: August 2, 2021

Table of Contents

Cloudera Search Overview

Cloudera Search provides easy, natural language access to data stored in or ingested into Hadoop, HBase, or cloud storage. End users and other web services can use full-text queries and faceted drill-down to explore text, semi-structured, and structured data as well as quickly filter and aggregate it to gain business insight without requiring SQL or programming skills.

Cloudera Search is Apache Solr fully integrated in the Cloudera platform, taking advantage of the flexible, scalable, and robust storage system and data processing frameworks included in CDH. This eliminates the need to move large data sets across infrastructures to perform business tasks. It further enables a streamlined data pipeline, where search and text matching is part of a larger workflow.

Cloudera Search incorporates Apache Solr, which includes Apache Lucene, SolrCloud, and Apache Tika. Cloudera Search is included with CDH.

Using Cloudera Search with the CDH infrastructure provides:

- Simplified infrastructure
- Better production visibility and control
- Quicker insights across various data types
- Quicker problem resolution
- Simplified interaction and platform access for more users and use cases beyond SQL
- Scalability, flexibility, and reliability of search services on the same platform used to run other types of workloads on the same data
- A unified security model across all processes with access to your data
- Flexibility and scale in ingest and pre-processing options

The following table describes Cloudera Search features.

Table 1: Cloudera Search Features

Feature	Description
Unified management and monitoring with Cloudera Manager	Cloudera Manager provides unified and centralized management and monitoring for CDH and Cloudera Search. Cloudera Manager simplifies deployment, configuration, and monitoring of your search services. Many existing search solutions lack management and monitoring capabilities and fail to provide deep insight into utilization, system health, trending, and other supportability aspects.
Simple cluster and collection management using the `solrctl` tool	Solrctl is a command line tool that allows convenient, centralized management of: • Solr instance configurations and schemas in ZooKeeper • Solr collection life cycle including low level control (Solr cores) • Collection snapshots, Backup and restore operations • SolrCloud cluster initialization and configuration • Sentry authorization and permissions
Index storage in HDFS	Cloudera Search is integrated with HDFS for robust, scalable, and self-healing index storage. Indexes created by Solr/Lucene are directly written in HDFS with the data, instead of to local disk, thereby providing fault tolerance and redundancy. Cloudera Search is optimized for fast read and write of indexes in HDFS while indexes are served and queried through standard Solr mechanisms. Because

Feature	Description
	data and indexes are co-located, data processing does not require transport or separately managed storage.
Batch index creation through MapReduce	To facilitate index creation for large data sets, Cloudera Search has built-in MapReduce jobs for indexing data stored in HDFS or HBase. As a result, the linear scalability of MapReduce is applied to the indexing pipeline, off-loading Solr index serving resources.
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 events can be written directly to Cloudera Search indexers. In addition, Flume supports routing events, filtering, and annotation of data passed 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. The drag-and-drop dashboard interface makes it easy for anyone to create a Search dashboard.
Simplified data processing for Search workloads	Cloudera Search can use 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 Log file formats, JSON, XML, and HTML. Cloudera Search also provides Morphlines, an easy-to-use, pre-built library of common data preprocessing functions. Morphlines simplifies data preparation for indexing over a variety of file formats. Users can easily implement Morphlines for Flume, Kafka, and HBase, or re-use the same Morphlines for other applications, such as MapReduce or Spark jobs.
HBase search	Cloudera Search integrates with HBase, enabling full-text search of HBase data without affecting HBase performance or duplicating data storage. A listener monitors the replication event stream from HBase RegionServers and captures each write or update-replicated event, enabling extraction and mapping (for example, using Morphlines). The event is then sent directly to Solr for indexing and storage in HDFS, using the same process as for other indexing workloads of Cloudera Search. The indexes can be served immediately, enabling free-text searching of HBase data.

How Cloudera Search Works

In near real-time indexing use cases, such as log or event stream analytics, Cloudera Search indexes events that are streamed through Apache Flume, Apache Kafka, Spark Streaming, or HBase. Fields and events are mapped to standard Solr indexable schemas. Lucene indexes the incoming events and the index is written and stored in standard Lucene index files in HDFS. Regular Flume event routing and storage of data 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 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. It can also be conveniently used for re-indexing (a common pain point in stand-alone Solr) or ad-hoc indexing for on-demand data exploration. Often, batch indexing is done on a regular basis (hourly, daily, weekly, and so on) as part of a larger workflow.

For such cases, Cloudera Search includes a highly scalable indexing workflow based on MapReduce or Spark. A MapReduce or Spark workflow is launched for specified files or folders in HDFS, or tables in HBase, and the field extraction and Solr schema mapping occurs during the mapping phase. Reducers use embedded Lucene to write the data as a single index or as index shards, depending on your configuration and preferences. After the indexes are stored in HDFS, they can be queried by using standard Solr mechanisms, as previously described above for the near-real-time indexing use case. You can also configure these batch indexing options to post newly indexed data directly into live, active indexes, served by Solr. This *GoLive* option enables a streamlined data pipeline without interrupting service to process regularly incoming batch updates.

The Lily HBase Indexer Service is a flexible, scalable, fault tolerant, transactional, near real-time oriented system for processing a continuous stream of HBase cell updates into live search indexes. 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 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.

Understanding Cloudera Search

Cloudera Search fits into the broader set of solutions available for analyzing information in large data sets. CDH provides both the means and the tools to store the data and run queries. You can explore data through:

- MapReduce or Spark jobs
- Impala queries
- Cloudera Search queries

CDH provides storage for and access to large data sets by using MapReduce jobs, but creating these jobs requires technical knowledge, and each job can take minutes or more to run. The longer run times associated with MapReduce jobs can interrupt the process of exploring data.

To provide more immediate queries and responses and to eliminate the need to write MapReduce applications, you can use Apache Impala. Impala returns results in seconds instead of minutes.

Although Impala is a fast, powerful application, it uses SQL-based querying syntax. Using Impala can be challenging for users who are not familiar with SQL. If you do not know SQL, you can use Cloudera Search. Although Impala, Apache Hive, and Apache Pig all require a structure that is applied at query time, Search supports free-text search on any data or fields you have indexed.

How Search Uses Existing Infrastructure

Any data already in a CDH deployment can be indexed and made available for query by Cloudera Search. For data that is not stored in CDH, Cloudera Search provides tools for loading data into the existing infrastructure, and for indexing data as it is moved to HDFS or written to Apache HBase.

By leveraging existing infrastructure, Cloudera Search eliminates the need to create new, redundant structures. In addition, Cloudera Search uses services provided by CDH and Cloudera Manager in a way that does not interfere with other tasks running in the same environment. This way, you can reuse existing infrastructure without the cost and problems associated with running multiple services in the same set of systems.

Cloudera Search and Other Cloudera Components

Cloudera Search interacts with other Cloudera components to solve different problems. The following table lists Cloudera components that contribute to the Search process and describes how they interact with Cloudera Search:

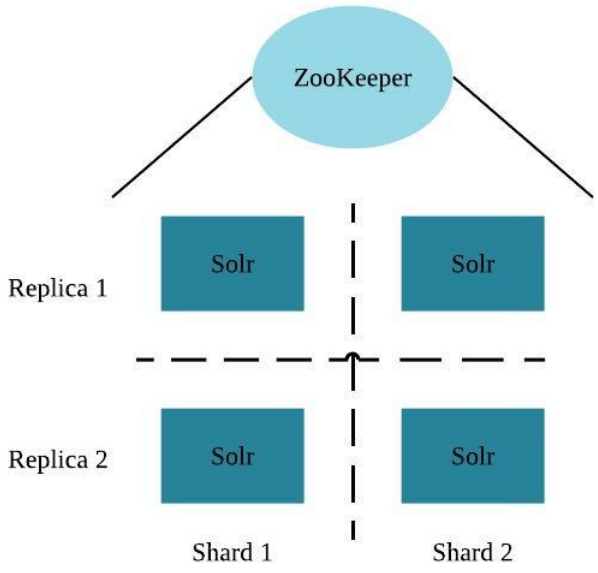
Component	Contribution	Applicable To
HDFS	Stores source documents. Search indexes source documents to make them searchable. Files that support Cloudera Search, such as Lucene index files and write-ahead logs, are also stored in HDFS. Using HDFS provides simpler provisioning on a larger base, redundancy, and fault tolerance. With HDFS, Cloudera Search servers are essentially stateless, so host failures have minimal consequences. HDFS also provides snapshotting, inter-cluster replication, and disaster recovery.	All cases
MapReduce	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 uses cluster resources for scalable batch indexing.	Many cases
Flume	Search includes a Flume sink that enables writing events directly to indexers deployed on the cluster, allowing data indexing during ingestion.	Many cases
Hue	Hue includes a GUI-based Search application that uses standard Solr APIs and can interact with data indexed in HDFS. The application provides support for the Solr standard query language and visualization of faceted search functionality.	Many cases
Morphlines	A morphline is a rich configuration file that defines an ETL transformation chain. Morphlines can consume any kind of data from any data source, process the data, and load the results into Cloudera Search. Morphlines run in a small, embeddable Java runtime system, and can be used for near real-time applications such as the flume agent as well as batch processing applications such as a Spark job.	Many cases
ZooKeeper	Coordinates distribution of data and metadata, also known as shards. It provides automatic failover to increase service resiliency.	Many cases
Spark	The CrunchIndexerTool can use Spark to move data from HDFS files into Apache Solr, and run the data through a morphline for extraction and transformation.	Some cases
HBase	Supports indexing of stored data, extracting columns, column families, and key information as fields. Although HBase does not use secondary indexing, Cloudera Search can facilitate full-text searches of content in rows and tables in HBase.	Some cases
Cloudera Manager	Deploys, configures, manages, and monitors Cloudera Search processes and resource utilization across services on the cluster. Cloudera Manager helps simplify Cloudera Search administration, but it is not required.	Some cases
Cloudera Navigator	Cloudera Navigator provides governance for Hadoop systems including support for auditing Search operations.	Some cases
Sentry	Apache Sentry enables role-based, fine-grained authorization for Cloudera Search. Sentry can apply restrictions to various actions, such as accessing data, managing configurations through config objects, or creating collections. Restrictions are consistently enforced, regardless	Some cases

Component	Contribution	Applicable To
	of how users attempt to complete actions. For example, restricting access to data in a collection restricts that access whether queries come from the command line, a browser, Hue, or through the admin console.	
Oozie	Automates scheduling and management of indexing jobs. Oozie can check for new data and begin indexing jobs as required.	Some cases
Impala	Further analyzes search results.	Some cases
Hive	Further analyzes search results.	Some cases
Parquet	Provides a columnar storage format, enabling especially rapid result returns for structured workloads such as Impala or Hive. Morphlines provide an efficient pipeline for extracting data from Parquet.	Some cases
Avro	Includes metadata that Cloudera Search can use for indexing.	Some cases
Kafka	Search uses this message broker project to increase throughput and decrease latency for handling real-time data.	Some cases
Sqoop	Ingests data in batch and enables data availability for batch indexing.	Some cases

Cloudera Search Architecture

Cloudera Search runs as a distributed service on a set of servers, and each server is responsible for a portion of the searchable data. The data 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 enables the system to provide higher levels of availability.



Each Cloudera Search server can handle requests independently. Clients can send requests to index documents or perform searches to any Search server, and that server routes the request to the correct server.

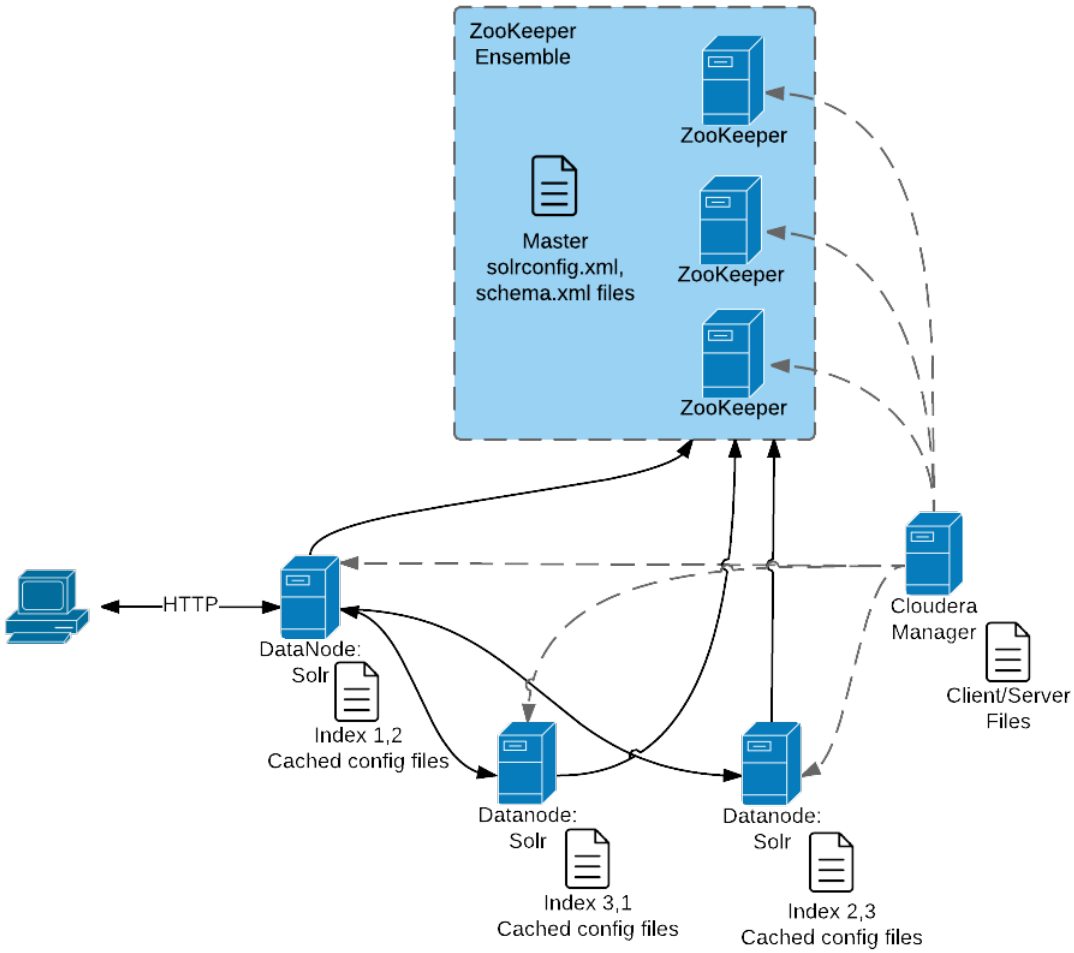
Each Search deployment requires:

- ZooKeeper on at least one host. You can install ZooKeeper, Search, and HDFS on the same host.
- HDFS on at least one, but as many as all hosts. HDFS is commonly installed on all cluster hosts.
- Solr on at least one but as many as all hosts. Solr is commonly installed on all cluster hosts.

More hosts with Solr and HDFS provides the following benefits:

- More Search servers processing requests.
- More Search and HDFS collocation increasing the degree of data locality. More local data provides faster performance and reduces network traffic.

The following graphic illustrates some of the key elements in a typical deployment.



This graphic illustrates:

1. A client submits a query over HTTP.
2. The response is received by the NameNode and then passed to a DataNode.
3. The DataNode distributes the request among other hosts with relevant shards.
4. The results of the query are gathered and returned to the client.

Also notice that the:

- Cloudera Manager provides client and server configuration files to other servers in the deployment.
- ZooKeeper server provides information about the state of the cluster and the other hosts running Solr.

The information a client must send to complete jobs varies:

- For queries, a client must have the hostname of the Solr server and the port to use.
- For actions related to collections, such as adding or deleting collections, the name of the collection is required as well.
- Indexing jobs, such as `MapReduceIndexer` jobs, use a MapReduce driver that starts a MapReduce job. These jobs can also process morphlines and index the results to Solr.

Cloudera Search Configuration Files

Configuration files in a Cloudera Search deployment include:

- Solr config files stored in ZooKeeper:
  - `solrconfig.xml`: Contains Solr configuration parameters.
  - `schema.xml`: Contains configuration that specifies the fields a document can contain, and how those fields are processed when adding documents to the index, or when querying those fields.
- Files copied from `hadoop-conf` in HDFS configurations to Solr servers:
  - `core-site.xml`
  - `hdfs-site.xml`
  - `ssl-client.xml`
  - `hadoop-env.sh`
  - `topology.map`
  - `topology.py`
- Cloudera Manager manages the following configuration files:
  - `cloudera-monitor.properties`
  - `cloudera-stack-monitor.properties`
- Logging and security configuration files:
  - `log4j.properties`
  - `jaas.conf`
  - `solr.keytab`
  - `sentry-site.xml`

You can use parcels or packages to deploy Search. Some files are always installed to the same location and some files are installed to different locations based on whether the installation is completed using parcels or packages.

Client Files

Client files are always installed to the same location and are required on any host where corresponding services are installed. In a Cloudera Manager environment, Cloudera Manager manages settings. In an unmanaged deployment, all files can be manually edited. Client configuration locations are:

- `/etc/solr/conf` for Solr client settings files
- `/etc/hadoop/conf` for HDFS, MapReduce, and YARN client settings files
- `/etc/zookeeper/conf` for ZooKeeper configuration files

Server Files

Server configuration file locations vary based on how services are installed.

- Cloudera Manager environments store configuration files in `/var/run/`.
- Unmanaged environments store configuration files in `/etc/<service>/conf`. For example:
  - `/etc/solr/conf`
  - `/etc/zookeeper/conf`

- `/etc/hadoop/conf`

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. To make content searchable, first ensure that it is ingested or stored in CDH.

Ingestion

You can move content to CDH by using:

- Flume, a flexible, agent-based data ingestion framework.
- A copy utility such as `distcp` for HDFS.
- Sqoop, a structured data ingestion connector.

Indexing

Content must be indexed before it can be searched. Indexing comprises the following steps:

1. Extraction, transformation, and loading (ETL) - Use existing engines or frameworks such as Apache Tika or Cloudera Morphlines.
   a. Content and metadata extraction
   b. Schema mapping
2. Index creation using Lucene.
   a. Index creation
   b. Index serialization

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

You can index content in three ways:

Batch indexing using MapReduce

To use MapReduce to index documents, run a MapReduce job on content in HDFS to produce 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 phase of the MapReduce task parses input into indexable documents, and the Reduce phase indexes the documents produced by the Map. You can also configure a MapReduce-based indexing job to use all assigned resources on the cluster, utilizing multiple reducing steps for intermediate indexing and merging operations, and then writing the reduction 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

Cloudera Search includes a Flume sink that enables you to write events directly to the indexer. This sink provides a flexible, scalable, fault-tolerant, near real-time (NRT) system for processing continuous streams of records to create live-searchable, free-text search indexes. Typically, data ingested using the Flume sink appears in search results in seconds, although you can tune this delay.

Data can flow from multiple sources through multiple flume hosts. These hosts, 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 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 immediately ready to serve queries to end users or search applications.

This flexible, customizable system scales effectively because parsing is moved from the Solr server to the multiple Flume hosts for ingesting new content.

Search includes parsers for standard data formats including Avro, CSV, Text, HTML, XML, PDF, Word, and Excel. 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 custom ETL logic can handle any format variation.

In addition, Cloudera Search includes a simplifying ETL 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 is designed to handle a variety of use cases:

- Search supports routing to multiple Solr collections to assign a single set of servers to support multiple user groups (multi-tenancy).
- Search supports routing to multiple shards to improve scalability and reliability.
- Index servers can be collocated 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 can be replicated across multiple index servers for high availability.

This flexible, scalable, highly available system provides low latency data acquisition and low latency querying. Instead of 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, you can use NRT ingestion and batch analysis tools.

NRT indexing using the API

Other clients can complete NRT indexing. This is done when the client first writes files directly to HDFS and then triggers indexing using the Solr REST API. Specifically, the API does the following:

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 is useful if you index as part of a larger workflow. For example, you could trigger indexing from an Oozie workflow.

Querying

After data is available as an index, the query API provided by the Search service allows direct queries to be completed or to be facilitated through a command-line tool or graphical interface. Hue includes a simple GUI-based Search application, or you can create a custom application based on the standard Solr API. Any application that works with Solr is compatible and runs as a search-serving application for Cloudera Search because Solr is the core.

Cloudera Search Tutorial

This tutorial introduces you to Cloudera Search and demonstrates some of its basic capabilities to help you become familiar with the concepts and components involved in Search. It demonstrates creating a simple test collection to validate your Cloudera Search installation, and then continues on to more advanced use cases of batch indexing, and then ingesting and indexing tweets from the Twitter public stream.

The topics in this tutorial assume you have completed the instructions in Deploying Cloudera Search on page 41. The examples in this tutorial use two shards, so make sure that your deployment includes at least two Solr servers.

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

- SchemaXml
- SolrConfigXml

Validating the Cloudera Search Deployment

After installing and deploying Cloudera Search, you can validate the deployment by indexing and querying sample documents. You can think of this as a type of "Hello, World!" for Cloudera Search to make sure that everything is installed and working properly.

Before beginning this process, make sure you have access to the Apache Solr admin web console. If your cluster is Kerberos-enabled, make sure you have access to the `solr@EXAMPLE.COM` Kerberos principal (where `EXAMPLE.COM` is your Kerberos realm name).

Configuring Sentry for Test Collection

If you have enabled Apache Sentry for authorization, you must have UPDATE permission for the `admin=collections` object as well as the collection you are creating (`test_collection` in this example). You can also use the wildcard (*) to grant permissions to create any collection.

For more information on configuring Sentry and granting permissions, see Configuring Sentry Authorization for Cloudera Search.

To grant your user account (`jdoe` in this example) the necessary permissions:

1. Switch to the Sentry admin user (`solr` in this example) using `kinit`:

```
kinit solr@EXAMPLE.COM
```

2. Create a Sentry role for your user account:

```
solrctl sentry --create-role cloudera_tutorial_role
```

3. Map a group to this role. In this example, user `jdoe` is a member of the `eng` group:

```
solrctl sentry --add-role-group cloudera_tutorial_role eng
```

4. Grant UPDATE privileges to the `cloudera_tutorial_role` role for the `admin=collections` object and `test_collection` collection:

```
solrctl sentry --grant-privilege cloudera_tutorial_role 'admin=collections->action=UPDATE'
solrctl sentry --grant-privilege cloudera_tutorial_role
'collection=test_collection->action=UPDATE'
```

You also need to grant UPDATE privileges on the Config object to be able to upload instance configurations on which you will base your collection:

```
solrctl sentry --grant-privilege cloudera_tutorial_role
'config=test_collection_config->action=UPDATE'
```

For more information on the Sentry privilege model for Cloudera Search, see Authorization Privilege Model for Cloudera Search.

Creating a Test Collection

1. If you are using Kerberos, `kinit` as the user that has privileges to create the collection:

```
kinit jdoe@EXAMPLE.COM
```

Replace `EXAMPLE.COM` with your Kerberos realm name.

2. Make sure that the `SOLR_ZK_ENSEMBLE` environment variable is set in `/etc/solr/conf/solr-env.sh`. For example:

```
cat /etc/solr/conf/solr-env.sh
```

```
export
SOLR_ZK_ENSEMBLE=zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181/solr
```

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

3. Generate configuration files for the collection:

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

4. If you are using Sentry for authorization, overwrite `solrconfig.xml` with `solrconfig.xml.secure`. If you omit this step, Sentry authorization *is not* enabled for the collection:

```
cp $HOME/test_collection_config/conf/solrconfig.xml.secure
$HOME/test_collection_config/conf/solrconfig.xml
```

This example uses collection level security only, so you need to disable document level security which is enabled by default in the generated instance configuration files. To do this, edit the newly generated `solrconfig.xml`:

```
vi $HOME/test_collection_config/conf/solrconfig.xml
```

Locate the section:

```
<searchComponent name="queryDocAuthorization"
class="org.apache.solr.handler.component.QueryDocAuthorizationComponent" >
```

and change

```
<bool name="enabled">true</bool>
```

to

```
<bool name="enabled">false</bool>
```

For more information, see Providing Document-Level Security Using Sentry and Solr Query Returns no Documents when Executed with a Non-Privileged User on page 131.

5. Use the ConfigSets API to upload the configuration to ZooKeeper:

```
(cd $HOME/test_collection_config/conf && zip -r - *) | curl -k --negotiate -u : -X POST
 --header "Content-Type:application/octet-stream" --data-binary @-
"https://search01.example.com:8985/solr/admin/configs?action=UPLOAD&name=test_collection_config"
```

> **Note:** This command requires the sentry privilege `'config=test_collection_config->action=UPDATE'` you granted previously.

> **Note:** Starting from CDH 6, the security model has changed to only allow direct ZooKeeper writes of the Solr znodes to the `solr` user only. Therefore, the `solrctl instancedir` command used in CDH 5.x will not work here:
>
> ```
> solrctl instancedir --create test_collection_config
> $HOME/test_collection_config
> ```
>
> To make the command work you could create a jaas configuration file to create the instancedir as the `solr` user. However, it is a more secure method to use the ConfigSets API as provided in the curl example above. The client of the ConfigSets API does not connect to ZooKeeper directly, it connects to Solr therefore, it goes through Kerberos authentication and Sentry authorization.

6. Create a new collection with two shards (specified by the `-s` parameter) using the named configuration (specified by the `-c` parameter):

```
solrctl collection --create test_collection -s 2 -c test_collection_config
```

Indexing Sample Data

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

- **Parcel-based Installation (Security Enabled):**

```
cd /opt/cloudera/parcels/CDH/share/doc/solr-doc*/example/exampledocs
find *.xml -exec curl -i -k --negotiate -u:
https://search01.example.com:8985/solr/test_collection/update -H "Content-Type: text/xml"
 --data-binary @{} \;
```

- **Parcel-based Installation (Security Disabled):**

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

- **Package-based Installation (Security Enabled):**

```
cd /usr/share/doc/solr-doc*/example/exampledocs
find *.xml -exec curl -i -k --negotiate -u:
https://search01.example.com:8985/solr/test_collection/update -H "Content-Type: text/xml"
 --data-binary @{} \;
```

- **Package-based Installation (Security Disabled):**

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

Querying Sample Data

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

1. Open the Solr admin web interface in a browser by accessing the following URL:
   - **Security Enabled:** `https://search01.example.com:8985/solr`
   - **Security Disabled:** `http://search01.example.com:8983/solr`

   Replace *search01.example.com* with the name of any host running the Solr Server process. If you have security enabled on your cluster, enter the credentials for the `solr@EXAMPLE.COM` principal when prompted.
2. Select **Cloud** from the left panel.
3. Select one of the hosts listed for the `test_collection` collection.
4. From the **Core Selector** drop-down menu in the left panel, select the `test_collection` shard.
5. Select **Query** from the left panel and click **Execute Query**. If you see results such as the following, indexing was successful:

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

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

Next Steps

After you have verified that Cloudera Search is installed and running properly, you can experiment with other methods of ingesting and indexing data. This tutorial uses tweets to demonstrate batch indexing and near real time (NRT) indexing. Continue on to the next portion of the tutorial:

- Preparing to Index Sample Tweets with Cloudera Search on page 19

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

Preparing to Index Sample Tweets with Cloudera Search

In this section of the Cloudera Search tutorial, you will create a collection for tweets. The remaining examples in the tutorial use the same collection, so make sure that you follow these instructions carefully.

Configuring Sentry for Tweet Collection

If you have enabled Apache Sentry for authorization, you must have UPDATE permission for the `admin=collections` object as well as the collection you are creating (`cloudera_tutorial_tweets` in this example). You can also use the wildcard (`*`) to grant permissions to create any collection.

For more information on configuring Sentry and granting permissions, see Configuring Sentry Authorization for Cloudera Search.

To grant your user account (`jdoe` in this example) the necessary permissions:

1. Switch to the Sentry admin user (`solr` in this example) using `kinit`:

```
kinit solr@EXAMPLE.COM
```

2. Grant `UPDATE` privileges to the `cloudera_tutorial_role` role for the `admin=collections` object and `cloudera_tutorial_tweets` collection:

```
solrctl sentry --grant-privilege cloudera_tutorial_role 'admin=collections->action=UPDATE'
solrctl sentry --grant-privilege cloudera_tutorial_role
'collection=cloudera_tutorial_tweets->action=UPDATE'
```

The `cloudera_tutorial_role` role was created in Configuring Sentry for Test Collection on page 16. For more information on the Sentry privilege model for Cloudera Search, see Authorization Privilege Model for Cloudera Search.

Create a Collection for Tweets

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

```
cat /etc/solr/conf/solr-env.sh
```

```
export
SOLR_ZK_ENSEMBLE=zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181/solr
```

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

2. If you are using Kerberos, `kinit` as the user that has privileges to create the collection:

```
kinit jdoe@EXAMPLE.COM
```

Replace `EXAMPLE.COM` with your Kerberos realm name.

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

- **Parcel-based Installation:**

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

- **Package-based Installation:**

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

4. If you are using Apache Sentry for authorization, overwrite `solrconfig.xml` with `solrconfig.xml.secure`. If you omit this step, Sentry authorization *is not* enabled for the collection:

```
cp $HOME/cloudera_tutorial_tweets_config/conf/solrconfig.xml.secure
$HOME/cloudera_tutorial_tweets_config/conf/solrconfig.xml
```

5. Upload the configuration to ZooKeeper:

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

6. Create a new collection with two shards (specified by the `-s` parameter) using the named configuration (specified by the `-c` parameter):

```
solrctl collection --create cloudera_tutorial_tweets -s 2 -c
cloudera_tutorial_tweets_config
```

7. Verify that the collection is live. Open the Solr admin web interface in a browser by accessing the following URL:

   - **TLS/SSL Enabled:** `https://search01.example.com:8985/solr/#/~cloud`
   - **TLS/SSL Disabled:** `http://search01.example.com:8983/solr/#/~cloud`

   If you have Kerberos authentication enabled on your cluster, enter the credentials for the `solr@EXAMPLE.COM` principal when prompted. Replace *search01.example.com* with the name of any host running the Solr Server process. Look for the `cloudera_tutorial_tweets` collection to verify that it exists.

8. Prepare the configuration for use with MapReduce:

```
cp -r $HOME/cloudera_tutorial_tweets_config $HOME/cloudera_tutorial_tweets_mr_config
```

Copy Sample Tweets to HDFS

1. Copy the provided sample tweets to HDFS. These tweets will be used to demonstrate the batch indexing capabilities of Cloudera Search:

   - **Parcel-based Installation (Security Enabled):**

```
kinit hdfs@EXAMPLE.COM
```

```
hdfs dfs -mkdir -p /user/jdoe
```

```
hdfs dfs -chown jdoe:jdoe /user/jdoe
```

```
kinit jdoe@EXAMPLE.COM
```

```
hdfs dfs -mkdir -p /user/jdoe/indir
```

```
hdfs dfs -put
/opt/cloudera/parcels/CDH/share/doc/search*/examples/test-documents/sample-statuses-*.avro
 /user/jdoe/indir/
```

```
hdfs dfs -ls /user/jdoe/indir
```

   - **Parcel-based Installation (Security Disabled):**

```
sudo -u hdfs hdfs dfs -mkdir -p /user/jdoe
```

```
sudo -u hdfs hdfs dfs -chown jdoe:jdoe /user/jdoe
```

```
hdfs dfs -mkdir -p /user/jdoe/indir
```

```
hdfs dfs -put
/opt/cloudera/parcels/CDH/share/doc/search*/examples/test-documents/sample-statuses-*.avro
 /user/jdoe/indir/
```

```
hdfs dfs -ls /user/jdoe/indir
```

- **Package-based Installation (Security Enabled):**

```
kinit hdfs@EXAMPLE.COM
```

```
hdfs dfs -mkdir -p /user/jdoe
```

```
hdfs dfs -chown jdoe:jdoe /user/jdoe
```

```
kinit jdoe@EXAMPLE.COM
```

```
hdfs dfs -mkdir -p /user/jdoe/indir
```

```
hdfs dfs -put /usr/share/doc/search*/examples/test-documents/sample-statuses-*.avro
/user/jdoe/indir/
```

```
hdfs dfs -ls /user/jdoe/indir
```

- **Package-based Installation (Security Disabled):**

```
sudo -u hdfs hdfs dfs -mkdir -p /user/jdoe
```

```
sudo -u hdfs hdfs dfs -chown jdoe:jdoe /user/jdoe
```

```
hdfs dfs -mkdir -p /user/jdoe/indir
```

```
hdfs dfs -put /usr/share/doc/search*/examples/test-documents/sample-statuses-*.avro
/user/jdoe/indir/
```

```
hdfs dfs -ls /user/jdoe/indir
```

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

```
hdfs dfs -rm -r -skipTrash /user/jdoe/outdir
```

```
hdfs dfs -mkdir /user/jdoe/outdir
```

```
hdfs dfs -ls /user/jdoe/outdir
```

The sample tweets are now in HDFS and ready to be indexed. Continue to Using MapReduce Batch Indexing to Index Sample Tweets on page 22 to index the sample tweets or to Near Real Time (NRT) Indexing Tweets Using Flume on page 28 to index live tweets from the Twitter public stream.

Using MapReduce Batch Indexing to Index Sample Tweets

Note: This page contains references to CDH 5 components or features that have been removed from CDH 6. These references are only applicable if you are managing a CDH 5 cluster with Cloudera Manager 6. For more information, see Deprecated Items.

The following sections include examples that illustrate how to use MapReduce for batch indexing. Batch indexing is useful for periodically indexing large amounts of data, or for indexing a dataset for the first time. Before continuing, make sure that you have:

- Completed the procedures in .
- If necessary, completed the Sentry configuration in Configuring Sentry for Tweet Collection on page 19.

Batch Indexing into Online Solr Servers Using GoLive

Warning: Batch indexing into offline Solr shards is not supported in environments in which batch indexing into online Solr servers using GoLive occurs.

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

1. If you are working with a secured cluster, configure your client `jaas.conf` file as described in Configuring a jaas.conf File.
2. If you are using Kerberos, `kinit` as the user that has privileges to update the collection:

```
kinit jdoe@EXAMPLE.COM
```

   Replace `EXAMPLE.COM` with your Kerberos realm name.

3. Delete any existing documents in the `cloudera_tutorial_tweets` collection. If your cluster does not have security enabled, run the following commands as the `solr` user by adding `sudo -u solr` before the command:

```
solrctl collection --deletedocs cloudera_tutorial_tweets
```

4. Run the MapReduce job with the `--go-live` parameter. Replace *nn01.example.com* and *zk01.example.com* with your NameNode and ZooKeeper hostnames, respectively.

   - **Parcel-based Installation (Security Enabled):**

```
YARN_OPTS="-Djava.security.auth.login.config=/path/to/jaas.conf" yarn jar
/opt/cloudera/parcels/CDH/lib/solr/contrib/mr/search-mr-*-job.jar
org.apache.solr.hadoop.MapReduceIndexerTool -D 'mapred.child.java.opts=-Xmx500m' -D
'mapreduce.job.user.classpath.first=true' --log4j
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/log4j.properties
--morphline-file
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/test-morphlines/tutorialReadAvroContainer.conf
 --output-dir hdfs://nn01.example.com:8020/user/jdoe/outdir --verbose --go-live --zk-host
 zk01.example.com:2181/solr --collection cloudera_tutorial_tweets
hdfs://nn01.example.com:8020/user/jdoe/indir
```

   - **Parcel-based Installation (Security Disabled):**

```
yarn jar /opt/cloudera/parcels/CDH/lib/solr/contrib/mr/search-mr-*-job.jar
org.apache.solr.hadoop.MapReduceIndexerTool -D 'mapred.child.java.opts=-Xmx500m' -D
'mapreduce.job.user.classpath.first=true' --log4j
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/log4j.properties
--morphline-file
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/test-morphlines/tutorialReadAvroContainer.conf
 --output-dir hdfs://nn01.example.com:8020/user/jdoe/outdir --verbose --go-live --zk-host
 zk01.example.com:2181/solr --collection cloudera_tutorial_tweets
hdfs://nn01.example.com:8020/user/jdoe/indir
```

   - **Package-based Installation (Security Enabled):**

```
YARN_OPTS="-Djava.security.auth.login.config=/path/to/jaas.conf" yarn jar
/usr/lib/solr/contrib/mr/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool
 -D 'mapred.child.java.opts=-Xmx500m' -D 'mapreduce.job.user.classpath.first=true'
--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://nn01.example.com:8020/user/jdoe/outdir --verbose --go-live --zk-host
```

```
 zk01.example.com:2181/solr --collection cloudera_tutorial_tweets
hdfs://nn01.example.com:8020/user/jdoe/indir
```

- **Package-based Installation (Security Disabled):**

```
yarn jar /usr/lib/solr/contrib/mr/search-mr-*-job.jar
org.apache.solr.hadoop.MapReduceIndexerTool -D 'mapred.child.java.opts=-Xmx500m' -D
'mapreduce.job.user.classpath.first=true' --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://nn01.example.com:8020/user/jdoe/outdir --verbose --go-live --zk-host
 zk01.example.com:2181/solr --collection cloudera_tutorial_tweets
hdfs://nn01.example.com:8020/user/jdoe/indir
```

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

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

  # ZooKeeper ensemble
  zkHost : "zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181/solr"
}

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

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

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

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

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

- **Parcel-based Installation:**

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

- **Package-based Installation:**

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

For development purposes, you can use the `--dry-run` option to run in local mode and print documents to `stdout` instead of loading them into Solr. Using this option causes the morphline to run locally without submitting a job to MapReduce. Running locally provides faster turnaround during early trial and debug sessions.

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

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

The `log4j.properties` file location can be specified by using the `MapReduceIndexerTool --log4j /path/to/log4j.properties` command-line option.

5. Check the job status at:

```
http://rm01.example.com:8090/cluster
```

For secure clusters, replace `http` with `https`.

6. When the job is complete, run a Solr query. For example, for a Solr server running on `search01.example.com`, go to one of the following URLs in a browser, depending on whether you have enabled security on your cluster:

   - **Security Enabled:** `https://search01.example.com:8985/solr/cloudera_tutorial_tweets/select?q=*%3A*&wt=json&indent=true`
   - **Security Disabled:** `http://search01.example.com:8983/solr/cloudera_tutorial_tweets/select?q=*%3A*&wt=json&indent=true`

   If indexing was successful, this page displays the first 10 query results.

Batch Indexing into Offline Solr Shards

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

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

1. If you are working with a secured cluster, configure your client `jaas.conf` file as described in Configuring a jaas.conf File.
2. If you are using Kerberos, `kinit` as the user that has privileges to update the collection:

```
kinit jdoe@EXAMPLE.COM
```

Replace `EXAMPLE.COM` with your Kerberos realm name.

3. Delete any existing documents in the `cloudera_tutorial_tweets` collection. If your cluster does not have security enabled, run the following commands as the `solr` user by adding `sudo -u solr` before the command:

```
solrctl collection --deletedocs cloudera_tutorial_tweets
```

4. Delete the contents of the `outdir` directory:

```
hdfs dfs -rm -r -skipTrash /user/jdoe/outdir/*
```

5. Run the MapReduce job as follows, replacing *nn01.example.com* in the command with your NameNode hostname.

   - **Parcel-based Installation (Security Enabled):**

```
YARN_OPTS="-Djava.security.auth.login.config=/path/to/jaas.conf" yarn jar
/opt/cloudera/parcels/CDH/lib/solr/contrib/mr/search-mr-*-job.jar
org.apache.solr.hadoop.MapReduceIndexerTool -D 'mapred.child.java.opts=-Xmx500m' -D
'mapreduce.job.user.classpath.first=true' --log4j
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/log4j.properties
--morphline-file
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/test-morphlines/tutorialReadAvroContainer.conf
 --output-dir hdfs://nn01.example.com:8020/user/jdoe/outdir --verbose --solr-home-dir
```

```
$HOME/cloudera_tutorial_tweets_config --collection cloudera_tutorial_tweets --shards 2
 hdfs://nn01.example.com:8020/user/jdoe/indir
```

- **Parcel-based Installation (Security Disabled):**

```
yarn jar /opt/cloudera/parcels/CDH/lib/solr/contrib/mr/search-mr-*-job.jar
org.apache.solr.hadoop.MapReduceIndexerTool -D 'mapred.child.java.opts=-Xmx500m' -D
'mapreduce.job.user.classpath.first=true' --log4j
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/log4j.properties
--morphline-file
/opt/cloudera/parcels/CDH/share/doc/search*/examples/solr-nrt/test-morphlines/tutorialReadAvroContainer.conf
 --output-dir hdfs://nn01.example.com:8020/user/jdoe/outdir --verbose --solr-home-dir
$HOME/cloudera_tutorial_tweets_config --collection cloudera_tutorial_tweets --shards 2
 hdfs://nn01.example.com:8020/user/jdoe/indir
```

- **Package-based Installation (Security Enabled):**

```
YARN_OPTS="-Djava.security.auth.login.config=/path/to/jaas.conf" yarn jar
/usr/lib/solr/contrib/mr/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool
 -D 'mapred.child.java.opts=-Xmx500m' -D 'mapreduce.job.user.classpath.first=true'
--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://nn01.example.com:8020/user/jdoe/outdir --verbose --solr-home-dir
$HOME/cloudera_tutorial_tweets_config --collection cloudera_tutorial_tweets --shards 2
 hdfs://nn01.example.com:8020/user/jdoe/indir
```

- **Package-based Installation (Security Disabled):**

```
yarn jar /usr/lib/solr/contrib/mr/search-mr-*-job.jar
org.apache.solr.hadoop.MapReduceIndexerTool -D 'mapred.child.java.opts=-Xmx500m' -D
'mapreduce.job.user.classpath.first=true' --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://nn01.example.com:8020/user/jdoe/outdir --verbose --solr-home-dir
$HOME/cloudera_tutorial_tweets_config --collection cloudera_tutorial_tweets --shards 2
 hdfs://nn01.example.com:8020/user/jdoe/indir
```

6. Check the job status at:

```
http://rm01.example.com:8090/cluster
```

   For secure clusters, replace `http` with `https`.

7. After the job is completed, check the generated index files. Individual shards are written to the `results` directory with names of the form `part-00000`, `part-00001`, `part-00002`, and so on. This example has two shards:

```
hdfs dfs -ls /user/jdoe/outdir/results
```

```
hdfs dfs -ls /user/jdoe/outdir/results/part-00000/data/index
```

8. Stop the Solr service:

   - **Cloudera Manager: Solr service** > **Actions** > **Stop**
   - **Unmanaged:** On each Solr server host, run:

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

9. Identify the paths to each Solr core:

```
hdfs dfs -ls /solr/cloudera_tutorial_tweets
```

```
Found 2 items
drwxr-xr-x   - solr solr          0 2017-03-13 06:20
/solr/cloudera_tutorial_tweets/core_node1
drwxr-xr-x   - solr solr          0 2017-03-13 06:20
/solr/cloudera_tutorial_tweets/core_node2
```

10. Move the index shards into place.

    a. **(Kerberos only)** Switch to the `solr` user:

```
kinit solr@EXAMPLE.COM
```

    b. Remove outdated files. If your cluster does not have security enabled, run the following commands as the `solr` user by adding `sudo -u solr` before the command:

```
hdfs dfs -rm -r -skipTrash /solr/cloudera_tutorial_tweets/core_node1/data/index
hdfs dfs -rm -r -skipTrash /solr/cloudera_tutorial_tweets/core_node1/data/tlog
hdfs dfs -rm -r -skipTrash /solr/cloudera_tutorial_tweets/core_node2/data/index
hdfs dfs -rm -r -skipTrash /solr/cloudera_tutorial_tweets/core_node2/data/tlog
```

    c. Change ownership of the `results` directory to `solr`. If your cluster has security enabled, `kinit` as the HDFS superuser (`hdfs` by default) before running the following command. If your cluster does not have security enabled, run the command as the HDFS superuser by adding `sudo -u hdfs` before the command:

```
hdfs dfs -chown -R solr /user/jdoe/outdir/results
```

    d. **(Kerberos only)** Switch to the `solr` user:

```
kinit solr@EXAMPLE.COM
```

    e. Move the two index shards into place. If your cluster does not have security enabled, run the following commands as the `solr` user by adding `sudo -u solr` before the command

```
hdfs dfs -mv /user/jdoe/outdir/results/part-00000/data/index
/solr/cloudera_tutorial_tweets/core_node1/data
```

```
hdfs dfs -mv /user/jdoe/outdir/results/part-00001/data/index
/solr/cloudera_tutorial_tweets/core_node2/data
```

11. Start the Solr service:

- **Cloudera Manager: Solr service** > **Actions** > **Start**
- **Unmanaged:** On each Solr server host, run:

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

12. Run some Solr queries. For example, for a Solr server running on `search01.example.com`, go to one of the following URLs in a browser, depending on whether you have enabled security on your cluster:

- **Security Enabled:**
  `https://search01.example.com:8985/solr/cloudera_tutorial_tweets/select?q=*%3A*&wt=json&indent=true`
- **Security Disabled:**
  `http://search01.example.com:8983/solr/cloudera_tutorial_tweets/select?q=*%3A*&wt=json&indent=true`

If indexing was successful, this page displays the first 10 query results.

To index live tweets from the Twitter public stream, continue on to Near Real Time (NRT) Indexing Tweets Using Flume on page 28.

Near Real Time (NRT) Indexing Tweets Using Flume

The following section describes how to use Flume for near real time (NRT) indexing using tweets from the Twitter public stream as an example. Near real time indexing is generally used when new data needs to be returned in query results in time frames measured in seconds. Before continuing, make sure that you have completed the procedures in Preparing to Index Sample Tweets with Cloudera Search on page 19.

Install Flume

If you have not already done so, install Flume. For Cloudera Manager installations, Flume is included in CDH, and no additional action is required for installation. Add the Flume service to the cluster following the instructions in Adding a Service.

Sentry Configuration for NRT Indexing Using Flume

If your cluster has security enabled and is using Apache Sentry for authorization, make sure that the Flume system user (`flume` by default) has permission to update the collection (`cloudera_tutorial_tweets` in this example):

1. Switch to the Sentry admin user (`solr` in this example) using `kinit`:

```
kinit solr@EXAMPLE.COM
```

2. Create a Sentry role:

```
solrctl sentry --create-role cloudera_tutorial_flume
```

3. Map the `flume` group to this role:

```
solrctl sentry --add-role-group cloudera_tutorial_flume flume
```

4. Grant UDPATE privileges to the `cloudera_tutorial_flume` role for the `cloudera_tutorial_tweets` collections:

```
solrctl sentry --grant-privilege cloudera_tutorial_flume
'collection=cloudera_tutorial_tweets->action=update'
```

For more information on the Sentry privilege model for Cloudera Search, see Authorization Privilege Model for Cloudera Search.

Copy Configuration Template Files

Copy the configuration files as follows:

- **Parcel-based Installation:** For Cloudera Manager environments, the Flume agent is configured in a later section. Skip to Configuring the Flume Solr Sink on page 29.
- **Package-based Installation:**

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

Configuring the Flume Solr Sink

> **Warning:** Using more than one Flume agent for this tutorial can result in blocked access to the Twitter public stream. When you configure the Flume agent as described in this section, make sure that you are configuring a single agent on a single host, and not the entire Flume service.

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

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

1. Modify the Flume configuration for a single agent to specify the Flume source details and configure the flow. You must set the relative or absolute path to the morphline configuration file.
   - **Parcel-based Installation:** In the Cloudera Manager Admin Console, go to **Flume service** > **Instances** > **Agent (select one)** > **Configuration**. When prompted to make configuration changes on the service configuration page, click **Cancel**. Set **Agent Name** to `twitter_stream` and modify **Configuration File** exactly as follows. You will replace the `YOUR_TWITTER_*` values in a later step:

```
twitter_stream.sources = twitterSrc
twitter_stream.channels = memoryChannel
twitter_stream.sinks = solrSink

twitter_stream.sources.twitterSrc.type = org.apache.flume.source.twitter.TwitterSource
twitter_stream.sources.twitterSrc.consumerKey = YOUR_TWITTER_CONSUMER_KEY
twitter_stream.sources.twitterSrc.consumerSecret = YOUR_TWITTER_CONSUMER_SECRET
twitter_stream.sources.twitterSrc.accessToken = YOUR_TWITTER_ACCESS_TOKEN
twitter_stream.sources.twitterSrc.accessTokenSecret = YOUR_TWITTER_ACCESS_TOKEN_SECRET
twitter_stream.sources.twitterSrc.maxBatchDurationMillis = 200
twitter_stream.sources.twitterSrc.channels = memoryChannel

twitter_stream.channels.memoryChannel.type = memory
twitter_stream.channels.memoryChannel.capacity = 10000
twitter_stream.channels.memoryChannel.transactionCapacity = 1000

twitter_stream.sinks.solrSink.type =
org.apache.flume.sink.solr.morphline.MorphlineSolrSink
twitter_stream.sinks.solrSink.channel = memoryChannel
twitter_stream.sinks.solrSink.morphlineFile = morphlines.conf
```

     Click **Save Changes**.

   - **Package-based Installation:** If you copied the configuration templates as described in Copy Configuration Template Files on page 28, no further action is required in this step.

2. Edit the Morphline configuration to specify Solr environment details.
   - **Parcel-based Installation:** In the Cloudera Manager Admin Console, go to **Flume service** > **Instances** > **Agent (select one)** > **Configuration**. When prompted to make configuration changes on the service configuration page, click **Cancel**. Make sure that you selected the same agent that you selected in step 1. Edit the `SOLR_LOCATOR` directive in the **Morphlines File** as follows. Edit the `SOLR_LOCATOR` entry only. Leave the rest of the configuration unedited.

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

  # ZooKeeper ensemble
  zkHost : "zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181/solr"
}
```

     Replace the example ZooKeeper hostnames with the hostnames of your ZooKeeper servers.

     Click **Save Changes**.

- **Package-based Installation:** Edit the SOLR_LOCATOR section in `/etc/flume-ng/conf/morphline.conf` as follows:

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

  # ZooKeeper ensemble
  zkHost : "zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181/solr"
}
```

Replace the example ZooKeeper hostnames with the hostnames of your ZooKeeper servers.

3. **(Unmanaged environments only)** 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. Update the Java heap size.

- **Parcel-based Installation:** In the Cloudera Manager Admin Console, go to **Flume service** > **Instances** > **Agent (select one)** > **Configuration**. When prompted to make configuration changes on the service configuration page, click **Cancel**. Make sure that you selected the same agent that you selected in step 1. Select **Category** > **Resource Management**, and then set **Java Heap Size of Agent in Bytes** to `500` and select **MiB** in the unit drop-down menu. Click **Save Changes**.
- **Package-based Installation:** Edit `/etc/flume-ng/conf/flume-env.sh` or `/opt/cloudera/parcels/CDH/etc/flume-ng/conf/flume-env.sh`, inserting or replacing JAVA_OPTS as follows:

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

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

- **Parcel-based Installation:** In the Cloudera Manager Admin Console, go to **Flume service** > **Instances** > **Agent (select one)** > **Configuration**. When prompted to make configuration changes on the service configuration page, click **Cancel**. Make sure that you selected the same agent that you selected in step 1. Select **Category** > **Advanced**, and then modify **Agent Logging Advanced Configuration Snippet (Safety Valve)** to include:

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

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

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

6. **(Optional)** In an unmanaged environment you can use SEARCH_HOME to configure where Flume finds Cloudera Search dependencies for the Flume Solr Sink. To set SEARCH_HOME use a command similar to the following:

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

Alternatively, you can add the same setting to `/etc/flume-ng/conf/flume-env.sh`.

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

Configuring Flume Solr Sink to Access the Twitter Public Stream

Use the Twitter developer site to generate credentials to access the Twitter public stream:

1. Sign in to https://apps.twitter.com with a Twitter account.
2. On the **Application Management** page, click **Create New App**.
3. Fill in the form to represent the Search installation. This can represent multiple clusters, and does not require the callback URL. Because this is not a publicly distributed application, the values you enter for the required name, description, and website fields are not important.
4. Read and accept the **Developer Agreement**, then click **Create your Twitter application** at the bottom of the page.
5. Click on the **Keys and Access Tokens** tab, then click **Create my access token** button at the bottom.

The Flume TwitterSource uses the Twitter 1.1 API, which requires authentication of both the consumer (application) and the user (you). Consider this information confidential, just like your regular Twitter credentials. Edit the Flume configuration and replace the following properties with the credentials you generated:

```
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
```

To edit the Flume configuration:

- **Parcel-based Installation:** In the Cloudera Manager Admin Console, go to **Flume service** > **Instances** > **Agent (select one)** > **Configuration**. When prompted to make configuration changes on the service configuration page, click **Cancel**. Make sure that you selected the same agent that you configured earlier. Modify the **Configuration File** parameter.
- **Package-based Installation:** Edit `/etc/flume-ng/conf/flume.conf`.

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

Starting the Flume Agent

1. If you are using Kerberos, `kinit` as the user that has privileges to update the collection:

```
kinit jdoe@EXAMPLE.COM
```

   Replace `EXAMPLE.COM` with your Kerberos realm name.

2. Delete all existing documents in the `cloudera_tutorial_tweets` collection. If your cluster does not have security enabled, run the following command as the `solr` user by adding `sudo -u solr` before the command:

```
solrctl collection --deletedocs cloudera_tutorial_tweets
```

3. Start or restart the Flume agent configured in Configuring the Flume Solr Sink on page 29:

   - **Parcel-based Installation: Flume service** > **Instances** > **Agent (select one)** > **Actions** > **Restart this Agent**. Make sure that you selected the same agent that you configured earlier.
   - **Package-based Installation:**

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

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

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

After restarting the Flume agent, use the Cloudera Search web UI. For example, if you have a Solr server running on `search01.example.com`, go to one of the following URLs in a browser to verify that new tweets have been ingested into Solr:

- **Security Enabled:**
  https://search01.example.com:8985/solr/cloudera_tutorial_tweets/select?q=*%3A*&sort=created_at+desc&wt=json&indent=true
- **Security Disabled:**
  http://search01.example.com:8983/solr/cloudera_tutorial_tweets/select?q=*%3A*&sort=created_at+desc&wt=json&indent=true

The query sorts the result set such that the most recently ingested tweets are at the top, based on the `created_at` timestamp. If you rerun the query, new tweets show up at the top of the result set.

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

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

In Cloudera Manager, you can use the safety valve to add the setting.

Go to **Flume service** > **Configuration** > **View and Edit** > **Agent** > **Advanced** > **Agent Logging Advanced Configuration Snippet (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 over 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 85.

1. Stop the Flume agent that you configured in Configuring the Flume Solr Sink on page 29:
   - **Parcel-based Installation: Flume service** > **Instances** > **Agent (select one)** > **Actions** > **Stop this Agent**. Make sure that you selected the same agent that you configured earlier.
   - **Package-based Installation:**

```
sudo /etc/init.d/flume-ng-agent stop
```

2. If you are using Kerberos, `kinit` as the user that has privileges to update the collection:

```
kinit jdoe@EXAMPLE.COM
```

   Replace `EXAMPLE.COM` with your Kerberos realm name.

3. Delete all existing documents in the `cloudera_tutorial_tweets` collection. If your cluster does not have security enabled, run the following commands as the `solr` user by adding `sudo -u solr` before the command:

```
solrctl collection --deletedocs cloudera_tutorial_tweets
```

4. Modify the Flume configuration:
   - **Parcel-based Installation:** In the Cloudera Manager Admin Console, go to **Flume service** > **Instances** > **Agent (select one)** > **Configuration**. When prompted to make configuration changes on the service configuration page, click **Cancel**. Replace the **Configuration File** configuration with the following:

```
twitter_stream.sources = httpSrc
twitter_stream.channels = memoryChannel
twitter_stream.sinks = solrSink

twitter_stream.sources.httpSrc.type = org.apache.flume.source.http.HTTPSource
twitter_stream.sources.httpSrc.port = 5140
twitter_stream.sources.httpSrc.handler = org.apache.flume.sink.solr.morphline.BlobHandler
twitter_stream.sources.httpSrc.handler.maxBlobLength = 2000000000
twitter_stream.sources.httpSrc.channels = memoryChannel


twitter_stream.channels.memoryChannel.type = memory
twitter_stream.channels.memoryChannel.capacity = 10000
twitter_stream.channels.memoryChannel.transactionCapacity = 1000
```

```
twitter_stream.sinks.solrSink.type =
org.apache.flume.sink.solr.morphline.MorphlineSolrSink
twitter_stream.sinks.solrSink.channel = memoryChannel
twitter_stream.sinks.solrSink.morphlineFile = morphlines.conf
```

Click **Save Changes**.

- **Package-based Installation:** If you copied the configuration templates as described in Copy Configuration Template Files on page 28, edit `/etc/flume-ng/conf/flume.conf` and comment out all sources except the HTTP source as follows:

```
#agent.sources = twitterSrc
agent.sources = httpSrc
#agent.sources = spoolSrc
#agent.sources = avroSrc
```

5. Start the Flume agent:
   - **Parcel-based Installation: Flume service** > **Instances** > **Agent (select one)** > **Actions** > **Start this Agent**. Make sure that you selected the same agent that you configured earlier.
   - **Package-based Installation:**

```
sudo /etc/init.d/flume-ng-agent start
```

6. Send a file containing tweets to the HTTP source. Run the following commands on a cluster host, replacing `flume01.example.com` with the hostname of the Flume agent you configured as the HTTP source:
   - **Parcel-based Installation:**

```
cd /opt/cloudera/parcels/CDH/share/doc/search-*/examples/test-documents
curl --data-binary @sample-statuses-20120906-141433-medium.avro
'http://flume01.example.com:5140?resourceName=sample-statuses-20120906-141433-medium.avro'
 --header 'Content-Type:application/octet-stream' --verbose
```

   - **Package-based Installation:**

```
cd /usr/share/doc/search-*/examples/test-documents
curl --data-binary @sample-statuses-20120906-141433-medium.avro
'http://flume01.example.com:5140?resourceName=sample-statuses-20120906-141433-medium.avro'
 --header 'Content-Type:application/octet-stream' --verbose
```

7. Check the log for status or errors:

```
tail /var/log/flume-ng/flume*.log
```

8. Run a Solr query. For example, for a Solr server running on `search01.example.com`, go to one of the following URLs in a browser, depending on whether you have enabled TLS on your cluster:
   - **Security Enabled:**
     `https://search01.example.com:8985/solr/cloudera_tutorial_tweets/select?q=*%3A*&wt=json&indent=true`
   - **Security Disabled:**
     `http://search01.example.com:8983/solr/cloudera_tutorial_tweets/select?q=*%3A*&wt=json&indent=true`

   If indexing was successful, this page displays the first 10 query results.

Indexing a File Containing Tweets with Flume SpoolDirectorySource

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

1. Stop the Flume agent configured in Configuring the Flume Solr Sink on page 29:

- **Parcel-based Installation: Flume service** > **Instances** > **Agent (select one)** > **Actions** > **Stop this Agent**. Make sure that you selected the same agent that you configured earlier.
- **Package-based Installation:**

```
sudo /etc/init.d/flume-ng-agent stop
```

2. If you are using Kerberos, `kinit` as the user that has privileges to update the collection:

```
kinit jdoe@EXAMPLE.COM
```

   Replace `EXAMPLE.COM` with your Kerberos realm name.

3. Delete all existing documents in the `cloudera_tutorial_tweets` collection. If your cluster does not have security enabled, run the following commands as the `solr` user by adding `sudo -u solr` before the command:

```
solrctl collection --deletedocs cloudera_tutorial_tweets
```

4. Modify the Flume configuration:

   - **Parcel-based Installation:** In the Cloudera Manager Admin Console, go to **Flume service** > **Instances** > **Agent (select one)** > **Configuration**. When prompted to make configuration changes on the service configuration page, click **Cancel**. Replace the **Configuration File** configuration with the following:

```
twitter_stream.sources = spoolSrc
twitter_stream.channels = memoryChannel
twitter_stream.sinks = solrSink

twitter_stream.sources.spoolSrc.type = spooldir
twitter_stream.sources.spoolSrc.spoolDir = /tmp/myspooldir
twitter_stream.sources.spoolSrc.ignorePattern = \.
twitter_stream.sources.spoolSrc.deserializer =
org.apache.flume.sink.solr.morphline.BlobDeserializer$Builder
twitter_stream.sources.spoolSrc.deserializer.maxBlobLength = 2000000000
twitter_stream.sources.spoolSrc.batchSize = 1
twitter_stream.sources.spoolSrc.fileHeader = true
twitter_stream.sources.spoolSrc.fileHeaderKey = resourceName
twitter_stream.sources.spoolSrc.channels = memoryChannel

twitter_stream.channels.memoryChannel.type = memory
twitter_stream.channels.memoryChannel.capacity = 10000
twitter_stream.channels.memoryChannel.transactionCapacity = 1000

twitter_stream.sinks.solrSink.type =
org.apache.flume.sink.solr.morphline.MorphlineSolrSink
twitter_stream.sinks.solrSink.channel = memoryChannel
twitter_stream.sinks.solrSink.morphlineFile = morphlines.conf
```

     Click **Save Changes**.

   - **Package-based Installation:** If you copied the configuration templates as described in Copy Configuration Template Files on page 28, edit `/etc/flume-ng/conf/flume.conf` and comment out all sources except the spool source as follows:

```
#agent.sources = twitterSrc
#agent.sources = httpSrc
agent.sources = spoolSrc
#agent.sources = avroSrc
```

5. Delete the spool directory if it exists, and then create a new spool directory. Run the following commands on the host running the Flume agent that you configured:

```
rm -rf /tmp/myspooldir
sudo -u flume mkdir /tmp/myspooldir
```

6. Start the Flume agent:
   - **Parcel-based Installation: Flume service** > **Instances** > **Agent (select one)** > **Actions** > **Start this Agent**. Make sure that you selected the same agent that you configured earlier.
   - **Package-based Installation:**

```
sudo /etc/init.d/flume-ng-agent start
```

7. Copy a file containing tweets to the `/tmp/myspooldir` directory. To ensure that no partial files are ingested, copy and then atomically move files. Run the following commands on the host running the Flume agent that you configured:
   - **Parcel-based Installation:**

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

   - **Package-based Installation:**

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

8. Check the log for status or errors:

```
tail /var/log/flume-ng/flume*.log
```

9. Check the completion status:

```
find /tmp/myspooldir
```

10. Run a Solr query. For example, for a Solr server running on `search01.example.com`, go to one of the following URLs in a browser, depending on whether you have enabled security on your cluster:
    - **Security Enabled:**
      `https://search01.example.com:8985/solr/cloudera_tutorial_tweets/select?q=*%3A*&wt=json&indent=true`
    - **Security Disabled:**
      `http://search01.example.com:8983/solr/cloudera_tutorial_tweets/select?q=*%3A*&wt=json&indent=true`

    If indexing was successful, this page displays the first 10 query results.

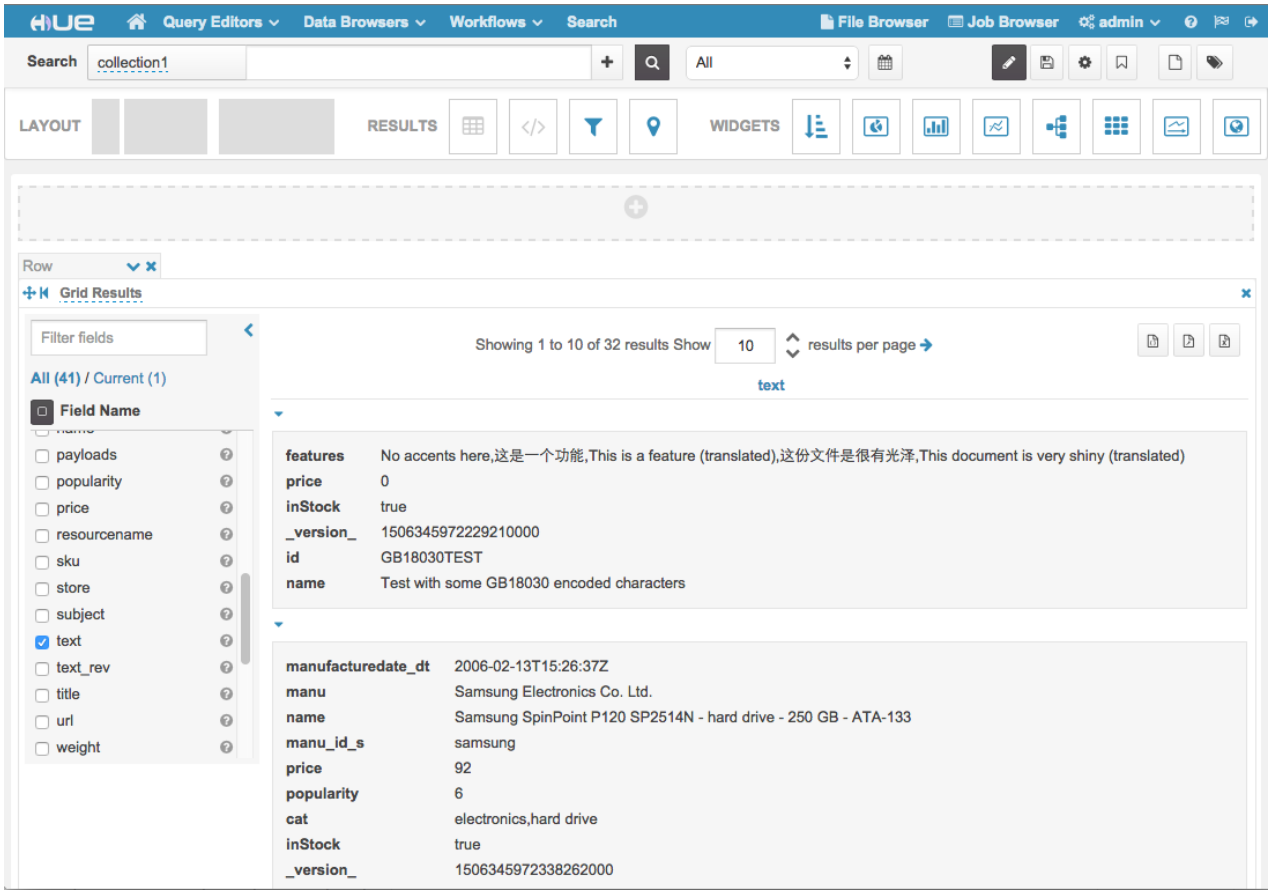
Using Hue with Cloudera Search

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

You can use the Hue Web UI to run search queries. For example, for the server `search01.example.com`, use: `http://search01.example.com:8889/hue/indexer/new_search`.

Search User Interface in Hue

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


Query Editors
Data Browsers
Workflows
Search
File Browser
Job Browser
admin
Search
collection1
All
LAYOUT
RESULTS
WIDGETS
Row
Grid Results
Filter fields
All (41) / Current (1)
Field Name
payloads
popularity
price
resourcename
sku
store
subject
text
text_rev
title
url
weight
Showing 1 to 10 of 32 results Show
10
results per page
text
features No accents here,这是一个功能,This is a feature (translated),这份文件是很有光泽,This document is very shiny (translated)
price 0
inStock true
version 1506345972229210000
id GB18030TEST
name Test with some GB18030 encoded characters
manufacturedate_dt 2006-02-13T15:26:37Z
manu Samsung Electronics Co. Ltd.
name Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133
manu_id_s samsung
price 92
popularity 6
cat electronics,hard drive
inStock true
version 1506345972338262000

Deployment Planning for Cloudera Search

Cloudera Search provides interactive search and scalable indexing. Before you begin installing Cloudera Search:

- Review the Cloudera Enterprise 6 Requirements and Supported Versions.
- Decide whether to install Cloudera Search by using Cloudera Manager or by using package management tools.
- Select the machines on which you want to install Cloudera Search, as well as any other services you are collocating with Search.
- Identify the tasks and workloads you need to manage and the types of data you need to search. This information can help guide the deployment process.

Important: Cloudera Search does not support Solr *contrib* modules, such as DataImportHandler. Morphlines, Spark Crunch indexer, MapReduce and Lily HBase indexers are part of the Cloudera Search product itself, therefore they are supported.

Choosing Where to Deploy the Cloudera Search Processes

You can collocate Solr Server roles with YARN NodeManager and HDFS DataNode roles. When collocating with NodeManager roles, make sure that machine resources are not oversubscribed.

Guidelines for Deploying Cloudera Search

Memory Considerations

CDH initially deploys Solr with a Java virtual machine (JVM) size of 1 GB, but this is insufficient for most use cases. Consider the following when determining an optimal JVM size for production usage:

- The more searchable data you have, the more memory you need. In general, 10 TB of searchable data requires more memory than 1 TB of searchable data.
- Indexing more fields requires more memory. For example, indexing all fields in a collection of logs, email messages, or Wikipedia entries requires more memory than indexing only the `Date Created` field.
- If the system must be stable and respond quickly, more memory might help. If slow responses are acceptable, you might be able to use less memory.

To ensure an appropriate amount of memory, consider your requirements and experiment in your environment. In general:

- Machines with 16 GB RAM are sufficient for some smaller loads or for evaluation.
- Machines with 32 GB RAM are sufficient for some production environments.
- Machines with 96 GB RAM are sufficient for most situations.

Deployment Guidelines

Consider the information in this topic as guidance instead of absolute requirements. You can use a sample application to benchmark different use cases and data types and sizes to help you identify the most important performance factors.

To determine how best to deploy Search in your environment, define use cases. The same Solr index can have different hardware requirements, depending on the types of queries performed. The most common variation in hardware requirements is memory. For example, the memory requirements for faceting vary depending on the number of unique terms in the faceted field. Suppose you want to use faceting on a field that has 10 unique values. In this case, only 10 logical containers are required for counting. No matter how many documents are in the index, memory overhead is almost nonexistent.

Conversely, the same index could have unique timestamps for every entry, and you want to facet on that field with a `: -type` query. In this case, each index requires its own logical container. With this organization, if you had a large number of documents—500 million, for example—then faceting across 10 fields would increase the RAM requirements significantly.

For this reason, you must consider use cases and data characteristics before you can estimate hardware requirements. Important parameters to consider are:

- Number of documents. For Cloudera Search, sharding is almost always required.
- Approximate word count for each potential field.
- What information is stored in the Solr index and what information is only for searching? Information stored in the index is returned with the search results.
- Foreign language support:
  - How many different languages appear in your data?
  - What percentage of documents are in each language?
  - Is language-specific search supported? This determines whether accent folding and storing the text in a single field is sufficient.
  - What language families will be searched? For example, you could combine all Western European languages into a single field, but combining English and Chinese into a single field is not practical. Even with similar languages, using a single field for different languages can be problematic. For example, sometimes accents alter the meaning of a word, and in such cases, accent folding loses important distinctions.
- Faceting requirements:
  - Be wary of faceting on fields that have many unique terms. For example, faceting on timestamps or free-text fields typically has a high cost. Faceting on a field with more than 10,000 unique values is typically not useful. Ensure that any such faceting requirement is necessary.
  - What types of facets are needed? You can facet on queries as well as field values. Faceting on queries is often useful for dates. For example, “in the last day” or “in the last week” can be valuable. Using Solr Date Math to facet on a bare “NOW” is almost always inefficient. Facet-by-query is not memory-intensive because the number of logical containers is limited by the number of queries specified, no matter how many unique values are in the underlying field. This can enable faceting on fields that contain information such as dates or times, while avoiding the problem described for faceting on fields with unique terms.
- Sorting requirements:
  - Sorting requires one integer for each document (`maxDoc`), which can take up significant memory. Additionally, sorting on strings requires storing each unique string value.
- Paging requirements. End users rarely look beyond the first few pages of search results. For use cases requiring *deep paging* (paging through a large number of results), using *cursors* can improve performance and resource utilization. For more information, see Pagination of Results on the Apache Solr wiki. Cursors are supported in CDH 5.2 and higher.
- Is advanced search capability planned? If so, how will it be implemented? Significant design decisions depend on user requirements:
  - Can users be expected to learn about the system? Advanced screens can intimidate e-commerce users, but these screens can be more effective if users can be expected to learn them.
  - How long will users wait for results? Data mining or other design requirements can affect response times.
- How many simultaneous users must your system accommodate?
- Update requirements. An update in Solr refers both to adding new documents and changing existing documents:
  - Loading new documents:
    - Bulk. Will the index be rebuilt from scratch periodically, or will there only be an initial load?
    - Incremental. At what rate will new documents enter the system?
  - Updating documents:

  - Can you characterize the expected number of modifications to existing documents?
  - How much latency is acceptable between when a document is added to Solr and when it is available in Search results?

- Security requirements. Solr has no built-in security options, although Cloudera Search supports authentication using Kerberos and authorization using Sentry. In Solr, document-level security is usually best accomplished by indexing authorization tokens with the document. The number of authorization tokens applied to a document is largely irrelevant; for example, thousands are reasonable but can be difficult to administer. The number of authorization tokens associated with a particular user should be no more than 100 in most cases. Security at this level is often enforced by appending an `fq` clause to the query, and adding thousands of tokens in an `fq` clause is expensive.
  - A *post filter*, also known as a *no-cache filter*, can help with access schemes that cannot use an `fq` clause. These are not cached and are applied only after all less-expensive filters are applied.
  - If grouping, faceting is not required to accurately reflect true document counts, so you can use some shortcuts. For example, ACL filtering is expensive in some systems, sometimes requiring database access. If completely accurate faceting is required, you must completely process the list to reflect accurate facets.
- Required query rate, usually measured in queries-per-second (QPS):
  - At a minimum, deploy machines with sufficient hardware resources to provide an acceptable response rate for a single user. Some types of queries can utilize the system so much that performance for even a small number of users is unacceptable. In this case, resharding can help.
  - If QPS is only somewhat lower than required and you do not want to reshard, you can often improve performance by adding replicas to each shard.
  - As the number of shards in your deployment increases, the general QPS rate starts to slowly decrease. This typically occurs when there are hundreds of shards.

Schemaless Mode Overview and Best Practices

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

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

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

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

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

3. When a new field is encountered, the schema is automatically updated using the API. The update is based on the guess about the type of data in the field.

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

By combining these techniques, Schemaless:

1. Starts without a populated schema.
2. Intakes and analyzes data.
3. Modifies the schema based on guesses about the data.
4. Ingests the data so it can be searched based on the schema updates.

To generate a configuration for use in Schemaless mode, use `solrctl instancedir --generate path -schemaless`. Then, create the instancedir and collection as with non-schemaless mode. For more information, see solrctl Reference on page 53.

Best Practices

User Defined Schemas Recommended for Production Use Cases

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

Give each Collection its own unique Instancedir

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

Deploying Cloudera Search

When you deploy Cloudera Search, SolrCloud partitions your data set into multiple indexes and processes, and uses ZooKeeper to simplify management, which results in a cluster of coordinating Apache Solr servers.

Note: Before you start

Review Deployment Planning for Cloudera Search on page 37.

Installing and Starting ZooKeeper Server

SolrCloud mode uses Apache ZooKeeper as a highly available, central location for cluster management. For a small cluster, running ZooKeeper collocated with the NameNode is recommended. For larger clusters, use multiple ZooKeeper servers.

If you do not already have a ZooKeeper service added to your cluster, add it using the instructions in Adding a Service for Cloudera Manager installations.

Initializing Solr

For Cloudera Manager installations, if you have not yet added the Solr service to your cluster, do so now using the instructions in Adding a Service. The **Add a Service** wizard automatically configures and initializes the Solr service.

Generating Collection Configuration

To start using Solr and indexing data, you must configure a collection to hold the index. A collection requires the following configuration files:

- `solrconfig.xml`
- `schema.xml`
- Any additional files referenced in the `xml` files

The `solrconfig.xml` file contains all of the Solr settings for a given collection, and the `schema.xml` file specifies the schema that Solr uses when indexing documents. For more details on how to configure a collection, see http://wiki.apache.org/solr/SchemaXml.

Configuration files for a collection are contained in a directory called an **instance directory**. To generate a template instance directory, run the following command:

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

You can customize a collection by directly editing the `solrconfig.xml` and `schema.xml` files created in `$HOME/solr_configs/conf`.

After you completing the configuration, you can make it available to Solr by running the following command, which uploads the contents of the instance directory to ZooKeeper:

```
solrctl instancedir --create <collection_name> $HOME/solr_configs
```

Use the `solrctl` utility to verify that your instance directory uploaded successfully and is available to ZooKeeper. List the uploaded instance directories as follows:

```
solrctl instancedir --list
```

For example, if you used the `--create` command to create a collection named `weblogs`, the `--list` command should return `weblogs`.

Important: Although you can create a collection directly in `/var/lib/solr`, Cloudera recommends using the `solrctl` utility instead.

Creating Collections

The Solr server does not include any default collections. Create a collection using the following command:

```
solrctl collection --create <collection_name> -s <shard_count>
```

To use the configuration that you provided to Solr in previous steps, use the same collection name (`weblogs` in our example). The `-s <shard_count>` parameter specifies the number of SolrCloud shards you want to partition the collection across. The number of shards cannot exceed the total number of Solr servers in your SolrCloud cluster.

To verify that the collection is active, go to `http://search01.example.com:8983/solr/<collection_name>/select?q=*%3A*&wt=json&indent=true` in a browser. For example, for the collection `weblogs`, the URL is `http://search01.example.com:8983/solr/weblogs/select?q=*%3A*&wt=json&indent=true`. Replace *search01.example.com* with the hostname of one of the Solr server hosts.

You can also view the SolrCloud topology using the URL `http://search01.example.com:8983/solr/#/~cloud`.

For more information on completing additional collection management tasks, see Managing Cloudera Search on page 46.

Using Search through a Proxy for High Availability

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

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

Overview of Proxy Usage and Load Balancing for Search

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

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

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

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

connection uses a different coordinator host. This load-balancing technique lets the hosts running the Solr service share this additional work, rather than concentrating it on a single machine.

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

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

Special Proxy Considerations for Clusters Using Kerberos

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

1. This section assumes you are starting with a Kerberos-enabled cluster. For more information, see Enabling Kerberos Authentication for CDH.
2. Choose the host you will use for the proxy server. Based on the Kerberos setup procedure, it should already have an entry `solr/proxy_host@realm` in its keytab:
   - **Cloudera Manager:**
     1. Navigate to **Solr service** > **Configuration** > **Category** > **Main**
     2. Set the value of **Solr Load Balancer** to `<hostname>:<port>`, specifying the hostname and port of the proxy host.
     3. Click **Save Changes**.
     4. Launch the Stale Configurations wizard to restart the Solr service and any dependent services.

     Cloudera Manager transparently handles the keytab and dependent service updates.

Using Custom JAR Files with Search

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

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

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

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

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

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

or

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

or

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

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

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

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

   For example:

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

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

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

   For example:

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

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

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

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

- MapReduceIndexerTool
- Lily HBase Indexer
- CrunchIndexerTool
- Flume indexing

Cloudera Search Security

Cloudera Search supports Kerberos for authentication, and Apache Sentry for authorization. Auditing is handled by Cloudera Navigator.

For information on enabling Kerberos for Cloudera Search, see Enabling Kerberos Authentication for CDH. For information on enable Sentry for authorization, see Configuring Sentry Authorization for Cloudera Search.

For information on using Cloudera Navigator for auditing, see Cloudera Navigator Auditing.

Managing Cloudera Search

Most Cloudera Search configuration is managed using `solrctl`, a wrapper script included with Cloudera Search. You can manipulate collections, instance directories, configs, individual cores, and so on.

A SolrCloud *collection* is the top-level object for indexing documents and providing a query interface. Each collection must be associated with a configuration, using either an *instance directory* or a *config* object. Different collections can use the same configuration. Each collection is typically replicated among several SolrCloud instances. Each replica is called a *core* and is assigned to an individual Solr service. The assignment process is managed automatically, but you can apply fine-grained control over individual cores using the `solrctl core` command.

A typical deployment workflow with `solrctl` consists of:

1. Establishing a configuration
   - If using configs, creating a config object from a template.
   - If using instance directories, generating an instance directory and uploading it to ZooKeeper.
2. Creating a collection associated with the name of the config or instance directory.

For a comparison of configs and instance directories, see Managing Configuration Using Configs or Instance Directories on page 47.

For more information on managing Cloudera Search, see the following topics:

Managing Cloudera Search Configuration

Cloudera Search configuration is primarily controlled by several configuration files, some of which are stored in Apache ZooKeeper:

- **`solr.xml`**

  This file is stored in ZooKeeper, and controls global properties for Apache Solr. To edit this file, you must download it from ZooKeeper, make your changes, and then upload the modified file back to ZooKeeper using the `solrctl cluster` command. For information about the `solr.xml` file, see Solr Configuration Files and Solr Cores and solr.xml in the Solr documentation.

- **`solrconfig.xml`**

  Each collection in Solr uses a `solrconfig.xml` file, stored in ZooKeeper, to control collection behavior. For information about the `solrconfig.xml` file, see Solr Configuration Files and Configuring solrconfig.xml in the Solr documentation.

- **`managed-schema` or `schema.xml`**

  In CDH 6, Cloudera recommends using a managed schema, and making schema changes using the Schema API (Apache Solr documentation). Collections in CDH 6 use either a managed schema or the legacy `schema.xml` file. These files, also stored in ZooKeeper and assigned to a collection, define the schema for the documents you are indexing. For example, they specify which fields to index, the expected data type for each field, the default field to query when the field is unspecified, and so on. For information about `managed-schema` and `schema.xml`, see Schema Factory Definition in SolrConfig in the Solr documentation.

- **`core.properties`**

  Unlike other configuration files, this file is stored in the local filesystem rather than ZooKeeper, and is used for core discovery. For more information on this process and the structure of the file, see Defining core.properties in the Solr documentation.

Managing Configuration Using Configs or Instance Directories

The `solrctl` utility includes the `config` and `instancedir` commands for managing configuration. Configs and instance directories refer to the same thing: named configuration sets used by collections,as specified by the `solrctl collection --create -c <configName>` command.

Although configs and instance directories are functionally identical from the perspective of the Solr server, there a number of important administrative differences between these two implementations:

Table 2: Config and Instance Directory Comparison

Attribute	Config	Instance Directory
Security	• In a Kerberos-enabled cluster, the ZooKeeper znodes associated with configurations created using the `solrctl config` command automatically have proper ZooKeeper ACLs. • Apache Sentry can be used to control access to the ConfigSets API (which is used by the `solrctl config` command). For more information, see Configuring Sentry Authorization for Cloudera Search and Securing Configs with ZooKeeper ACLs and Sentry on page 49.	• No ZooKeeper security support. Any user can create, delete, or modify an `instancedir` directly in ZooKeeper. • Because `instancedir` updates ZooKeeper directly, it is the client's responsibility to add the proper ACLs, which can be cumbersome.
Creation method	Generated from existing configs or instance directories in ZooKeeper using the ConfigSets API.	Manually edited locally and re-uploaded directly to ZooKeeper using `solrctl` utility.
Template support	• Several predefined templates are available. These can be used as the basis for creating additional configs. Additional templates can be created by creating configs that are immutable. • Mutable configs that use a managed schema can only be modified using the Schema API as opposed to being manually edited. As a result, configs are less flexible, but they are also less error-prone than instance directories.	One standard template.
Sentry support	Configs include a number of templates, each with Sentry-enabled and non-Sentry-enabled versions. To enable Sentry, choose a Sentry-enabled template.	Instance directories include a single template that supports enabling Sentry. To enable Sentry with instancedirs, overwrite the original `solrconfig.xml` file with `solrconfig.xml.secure` as described in Enabling Sentry for a Solr Collection.

Managing Configs

You can manage configuration objects directly using the `solrctl config` command, which is a wrapper script for the ConfigSets API.

Configs are named configuration sets that you can reference when creating collections. The `solrctl config` command syntax is as follows:

```
solrctl config [--create <name> <baseConfig> [-p <name>=<value>]...]
               [--delete <name>]
```

- `--create <name> <baseConfig>`: Creates a new config based on an existing config. The config is created with the specified *<name>*, using *<baseConfig>* as the template. For more information about config templates, see Config Templates on page 50.
  - `-p <name>=<value>`: Overrides a *<baseConfig>* setting. The only config property that you can override is `immutable`, so the possible options are `-p immutable=true` and `-p immutable=false`. If you are copying an immutable config, such as a template, use `-p immutable=false` to make sure that you can edit the new config.
- `--delete <name>`: Deletes the specified config. You cannot delete an immutable config without accessing ZooKeeper directly as the `solr` super user.

If you are using Apache Sentry, you must have permissions for the specific config you are creating or deleting, as well as the `admin=collections` privilege object.

Managing Instance Directories

An instance directory is a named set of configuration files. You can generate an instance directory template locally, edit the configuration, and then upload the directory to ZooKeeper as a named configuration set. You can then reference this named configuration set when creating a collection.

Creating configuration sets using instance directories cannot be restricted using Sentry. If you want to control access to configuration sets, you must enable ZooKeeper ACLs and use configs instead.

The `solrctl instancedir` command syntax is as follows:

```
solrctl instancedir [--generate <path> [-schemaless]]
                    [--create <name> <path>]
                    [--update <name> <path>]
                    [--get <name> <path>]
                    [--delete <name>]
                    [--list]
```

- `--generate <path>`: Generates an instance directory template on the local filesystem at *<path>*. The configuration files are located in the `conf` subdirectory under *<path>*.
  - `-schemaless`: Generates a schemaless instance directory template. For more information on schemaless support, see Schemaless Mode Overview and Best Practices on page 39.
- `--create <name> <path>`: Uploads a copy of the instance directory from *<path>* on the local filesystem to ZooKeeper. If an instance directory with the specified *<name>* already exists, this command fails. Use `--update` to modify existing instance directories.
- `--update <name> <path>`: Overwrites an existing instance directory in ZooKeeper using the specified files on the local filesystem. This command is analogous to first running `--delete <name>` followed by `--create <name> <path>`.
- `--get <name> <path>`: Downloads the specified instance directory from ZooKeeper to the specified path on the local filesystem. You can then edit the configuration and then re-upload it using `--update`.
- `--delete <name>`: Deletes the specified instance directory from ZooKeeper.
- `--list`: Lists existing instance directories as well as configs created by the `solrctl config` command.

Securing Configs with ZooKeeper ACLs and Sentry

You can restrict access to configuration sets by setting ZooKeeper ACLs on all *znodes* under and including `/solr` and using Sentry to control access to the ConfigSets API. Sentry requires Kerberos authentication.

The `solrctl instancedir` command interacts directly with ZooKeeper, and therefore cannot be protected by Sentry. Because the `solrctl config` command is a wrapper script for the ConfigSets API, it can be protected by Sentry.

To force users to use the ConfigSets API, you must set all ZooKeeper znodes under and including `/solr` to read-only (except the `solr` user):

1. Create a `jaas.conf` file containing the following:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=false
  useTicketCache=true
  principal="solr@EXAMPLE.COM";
};
```

   Replace `EXAMPLE.COM` with your Kerberos realm name.

2. Set the `LOG4J_PROPS` environment variable to a `log4j.properties` file:

```
export LOG4J_PROPS=/etc/zookeeper/conf/log4j.properties
```

3. Set the `ZKCLI_JVM_FLAGS` environment variable:

```
export ZKCLI_JVM_FLAGS="-Djava.security.auth.login.config=/path/to/jaas.conf \
-DzkACLProvider=org.apache.solr.common.cloud.SaslZkACLProvider \
-Droot.logger=INFO,console"
```

4. Authenticate as the `solr` user:

```
kinit solr@EXAMPLE.COM
```

   Replace `EXAMPLE.COM with your Kerberos realm name.`

5. Run the `zkcli.sh` script as follows:

   - **Cloudera Manager Deployment:**

```
/opt/cloudera/parcels/CDH/lib/solr/bin/zkcli.sh -zkhost zk01.example.com:2181 -cmd
updateacls /solr
```

   - **Unmanaged Deployment:**

```
/usr/lib/solr/bin/zkcli.sh -zkhost zk01.example.com:2181 -cmd updateacls /solr
```

   Replace *`zk01.example.com`* `with the hostname of a ZooKeeper server.`

After completing these steps, you cannot run commands such as `solrctl instancedir --create` or `solrctl instancedir --delete` without first authenticating as the `solr@EXAMPLE.COM` super user principal. Unauthenticated users can still run `solrctl instancedir --list` and `solrctl instancedir --get`, because those commands only perform read operations against ZooKeeper.

After setting ZooKeeper ACLs, you must configure Sentry to allow users to create and delete configs. For instructions on configuring Sentry for configs, see Configuring Sentry Authorization for Cloudera Search.

Config Templates

Configs can be declared as `immutable`, which means they cannot be deleted or have their Schema updated by the Schema API. Immutable configs are uneditable config templates that are the basis for additional configs. After a config is made immutable, you cannot change it back without accessing ZooKeeper directly as the `solr` (or `solr@EXAMPLE.COM` principal, if you are using Kerberos) super user.

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

Table 3: Available Config Templates and Attributes

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

Note: `schemalessTemplate` is the same as the template generated by the `solrctl instancedir --generate` command.

Config templates are managed using the `solrctl config` command. For example:

- To create a new config based on the `managedTemplateSecure` template:

```
solrctl config --create newConfig managedTemplateSecure -p immutable=false
```

- To create a new template (immutable config) from an existing config:

```
solrctl config --create newTemplate existingConfig -p immutable=true
```

Updating the Schema in a Solr Collection

If your collection was configured using an instance directory, you can download the instance directory, edit `schema.xml`, then re-upload it to ZooKeeper. For instructions, see Managing Instance Directories on page 48.

If your collection was configured using a config, you can update the schema using the Schema API. For information on using the Schema API, see Schema API in the *Apache Solr Reference Guide*.

Managing Collections in Cloudera Search

A *collection* in Cloudera Search refers to a repository for indexing and querying documents. Collections typically contain the same types of documents with similar schemas. For example, you might create separate collections for email, Twitter data, logs, forum posts, customer interactions, and so on.

Collections are managed using the `solrctl` command. For a reference to the `solrctl` commands and options, see solrctl Reference on page 53.

Creating a Solr Collection

If you have enabled Apache Sentry for authorization, you must have UPDATE permission for the `admin=collections` object as well as the collection you are creating. For example, if you want to create a collection named `logs`, you need the following Sentry permissions:

```
solr_admin = admin=collections->action=UDPATE, collection=logs->action=UDPATE
```

If you want to be able to create any collection, you can use the wildcard permission:

```
solr_admin = admin=collections->action=UDPATE, collection=*->action=UDPATE
```

For more information on configuring Sentry and granting permissions, see Configuring Sentry Authorization for Cloudera Search.

Note: Although it is not currenly strictly enforced, you are strongly recommended to observe the following limitations on collection names:

- Use only ASCII alphanumeric characters (`A-Za-z0-9`), hyphen (`-`), or underscore (_).
- Avoid using the strings `shard` and `replica`.

To create a collection:

1. If you are using Kerberos, `kinit` as a user with permission to create the collection:

```
kinit solradmin@EXAMPLE.COM
```

   Replace `EXAMPLE.COM` with your Kerberos realm name.

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

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

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

3. Generate configuration files for the collection:

   - **Using Configs with Sentry:**

```
solrctl config --create logs_config managedTemplateSecure -p immutable=false
```

   - **Using Configs without Sentry:**

```
solrctl config --create logs_config managedTemplate -p immutable=false
```

   - **Using Instance Directories with Sentry:**

```
solrctl instancedir --generate $HOME/logs_config
cp $HOME/logs_config/conf/solrconfig.xml.secure $HOME/logs_config/conf/solrconfig.xml
# Edit the configuration files as needed
solrctl instancedir --create logs_config $HOME/logs_config
```

   - **Using Instance Directories without Sentry:**

```
solrctl instancedir --generate $HOME/logs_config
# Edit the configuration files as needed
solrctl instancedir --create logs_config $HOME/logs_config
```

For more information on configs and instance directories, see Managing Configuration Using Configs or Instance Directories on page 47.

4. Create a new collection using the specified configuration:

```
solrctl collection --create logs -s <numShards> -c logs_config
```

Viewing Existing Solr Collections

You can view existing collections using the `solrctl collection --list` command. This command does not require Sentry authorization, because it reads the information from ZooKeeper.

Deleting All Documents in a Solr Collection

Deleting all documents in a Solr collection does not delete the collection or its configuration files. It only deletes the index. This can be useful for rapid prototyping of configuration changes in test environments.

If you have enabled Sentry for authorization, you must have `UDPATE` permission for the `admin=collections` object as well as the collection in which you are deleting documents. For example, if you want to delete documents in a collection named `logs`, you need the following Sentry permissions:

```
solr_admin = admin=collections->action=UDPATE, collection=logs->action=UDPATE
```

If you want to be able to delete documents in any collection, you can use the wildcard permission:

```
solr_admin = admin=collections->action=UDPATE, collection=*->action=UDPATE
```

For more information on configuring Sentry and granting permissions, see Configuring Sentry Authorization for Cloudera Search.

To delete all documents in a collection:

1. If you are using Kerberos, `kinit` as a user with permission to delete the collection:

```
kinit solradmin@EXAMPLE.COM
```

   Replace `EXAMPLE.COM` with your Kerberos realm name.

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

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

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

3. Delete the documents:

```
solrctl collection --deletedocs logs
```

Backing Up and Restoring Solr Collections

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

For more information, see Backing Up and Restoring Cloudera Search on page 67.

Deleting a Solr Collection

Deleting a Solr collection deletes the collection and its index, but does not delete its configuration files.

If you have enabled Sentry for authorization, you must have UDPATE permission for the `admin=collections` object as well as the collection you are deleting. For example, if you want to delete a collection named `logs`, you need the following Sentry permissions:

```
solr_admin = admin=collections->action=UDPATE, collection=logs->action=UDPATE
```

If you want to be able to delete any collection, you can use the wildcard permission:

```
solr_admin = admin=collections->action=UDPATE, collection=*->action=UDPATE
```

For more information on configuring Sentry and granting permissions, see Configuring Sentry Authorization for Cloudera Search.

To delete a collection:

1. If you are using Kerberos, `kinit` as a user with permission to delete the collection:

```
kinit solradmin@EXAMPLE.COM
```

   Replace `EXAMPLE.COM` with your Kerberos realm name.

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

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

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

3. Delete the collection:

```
solrctl collection --delete logs
```

solrctl Reference

The `solrctl` utility is a wrapper shell script included with Cloudera Search for managing collections, instance directories, configs, Apache Sentry permissions, and more.

For some examples of common tasks using `solrctl`, see Example solrctl Usage on page 62.

Make sure that the host on which you are running the `solrctl` utility has either a **Gateway** or **Solr Server** role assigned.

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

You can run `solrctl` on any host that is configured as part of the SolrCloud deployment (the **Solr** service in Cloudera Manager environments) . To run any `solrctl` command on a host outside of SolrCloud deployment, ensure that SolrCloud hosts are reachable and provide `--zk` and `--solr` command line options.

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

For collection configuration, users have the option of interacting directly with ZooKeeper using the `instancedir` option or using the Solr ConfigSets API using the `config` option. For more information, see Managing Configuration Using Configs or Instance Directories on page 47.

Syntax

The general `solrctl` command syntax is:

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

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

Options

If used, the following options must precede commands:

- `--solr <solr_uri>`: Directs `solrctl` to a SolrCloud web API available at the specified URI. This option is required for hosts running outside of SolrCloud. A sample URI might be: `http://search01.example.com:8983/solr`.
- `--zk <zk_ensemble>`: Directs `solrctl` to a particular ZooKeeper quorum. This option is required for hosts running outside of SolrCloud. For example: `zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181/solr`. Output from `solrctl` commands that use the `--zk` option is sent to `/dev/null`, so no results are displayed.
- `--jaas /path/to/jaas.conf`: Used to identify a JAAS configuration that specifies the principal with permissions to modify Solr metadata. The principal is typically `solr@EXAMPLE.COM`. In Kerberos-enabled environments where ZooKeeper ACLs protect Solr metadata, you must use this parameter when modifying metadata.
- `--help`: Prints help.
- `--quiet`: Suppresses most `solrctl` messages.
- `--debug`: Prints errors to `stdout`.
- `--trace`: Prints the executed commands to `stdout`.

Commands

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

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

- `init [--force]`: The `init` command, which initializes the overall state of the SolrCloud deployment, must be run before starting solr-server daemons for the first time. Use this command cautiously because it erases all SolrCloud deployment state information from ZooKeeper, including all configuration files. It does not delete collections. After successful initialization, you cannot recover any previous state.

```
instancedir [--generate <path> [-schemaless]]
            [--create <name> <path>]
            [--update <name> <path>]
            [--get <name> <path>]
            [--delete <name>]
            [--list]
```

Manipulates instance directories. The following options are supported:

- `--generate <path>`: Generates an instance directory template on the local filesystem at *<path>*. The configuration files are located in the `conf` subdirectory under *<path>*.
  - `-schemaless`: Generates a schemaless instance directory template. For more information on schemaless support, see Schemaless Mode Overview and Best Practices on page 39.
- `--create <name> <path>`: Uploads a copy of the instance directory from *<path>* on the local filesystem to ZooKeeper. If an instance directory with the specified *<name>* already exists, this command fails. Use `--update` to modify existing instance directories.
- `--update <name> <path>`: Overwrites an existing instance directory in ZooKeeper using the specified files on the local filesystem. This command is analogous to first running `--delete <name>` followed by `--create <name> <path>`.
- `--get <name> <path>`: Downloads the specified instance directory from ZooKeeper to the specified path on the local filesystem. You can then edit the configuration and then re-upload it using `--update`.
- `--delete <name>`: Deletes the specified instance directory from ZooKeeper.
- `--list`: Lists existing instance directories, including configs created by the `solrctl config` command.

```
config [--create <name> <baseConfig> [-p <name>=<value>]...]
       [--delete name]
```

Manipulates configs. The following options are supported:

- `--create name <baseConfig> [-p <name>=<value>`: Creates a new config based on an existing config. The config is created with the specified *<name>*, using *<baseConfig>* as the template. For more information about config templates, see Config Templates on page 50. The `-p name=value` option verrides a *<baseConfig>* setting. The only config property that you can override is `immutable`, so the possible options are `-p immutable=true` and `-p immutable=false`. If you are copying an immutable config, such as a template, use `-p immutable=false` to make sure that you can edit the new config.
- `--delete name`: Deletes the specified config. You cannot delete an immutable config without accessing ZooKeeper directly as the `solr` super user.

```
collection [--create <name> -s <numShards>
                           [-a]
                           [-c <configName>]
                           [-r <replicationFactor>]
                           [-m <maxShardsPerHost>]
                           [-n <createHostSet>]]
           [--delete <name>]
           [--reload <name>]
           [--stat <name>]
           [--deletedocs <name>]
           [--list]
           [--create-snapshot <snapshotName> -c <collectionName>]
           [--delete-snapshot <snapshotName> -c <collectionName>]
           [--list-snapshots <collectionName>]
           [--describe-snapshot <snapshotName> -c <collectionName>]
           [--prepare-snapshot-export <snapshotName> -c <collectionName> -d <destDir>
[-p <fsPathPrefix>]]
           [--export-snapshot <snapshotName> [-s <sourceDir>] [-c <collectionName>] -d
 <destDir>]
           [--restore name  -b <backupName> -l <backupLocation> -i <requestId>
                            [-a]
                            [-c <configName>]
                            [-r <replicationFactor>]
```

```
                          [-m <maxShardsPerNode>]]
          [--request-status <requestId>]
```

Manipulates collections. The following options are supported:

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

  The `-a` option enables automatic addition of replicas (`autoAddReplicas=true`) if machines hosting existing shards become unavailable.

  The collection uses the specified <*configName*> for its configuration set, and the specified <*replicationFactor*> controls the number of replicas for the collection. Keep in mind that this replication factor is on top of the HDFS replication factor.

  The maximum shards per host is determined by <*maxShardsPerHost*>, and you can specify specific hosts for the collection in the <*hostList*>.

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

- `--delete <name>`: Deletes a collection.
- `--reload <name>`: Reloads a collection.
- `--stat <name>`: Outputs SolrCloud specific run-time information for a collection.
- `--deletedocs <name>`: Purges all indexed documents from a collection.
- `--list`: Lists all collections.
- The snapshot-related commands are covered in detail in Backing Up and Restoring Cloudera Search on page 67.

```
core [--create <name> [-p <name>=<value>]...]
     [--reload <name>]
     [--unload <name>]
     [--status <name>]
```

Manipulates cores. The following options are supported:

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

```
cluster [--get-solrxml <file>]
        [--put-solrxml <file>]
        [--set-property <name> <value>]
        [--remove-property <name>]
        [--get-clusterstate <file>]
```

Manages cluster configuration. The following options are supported:

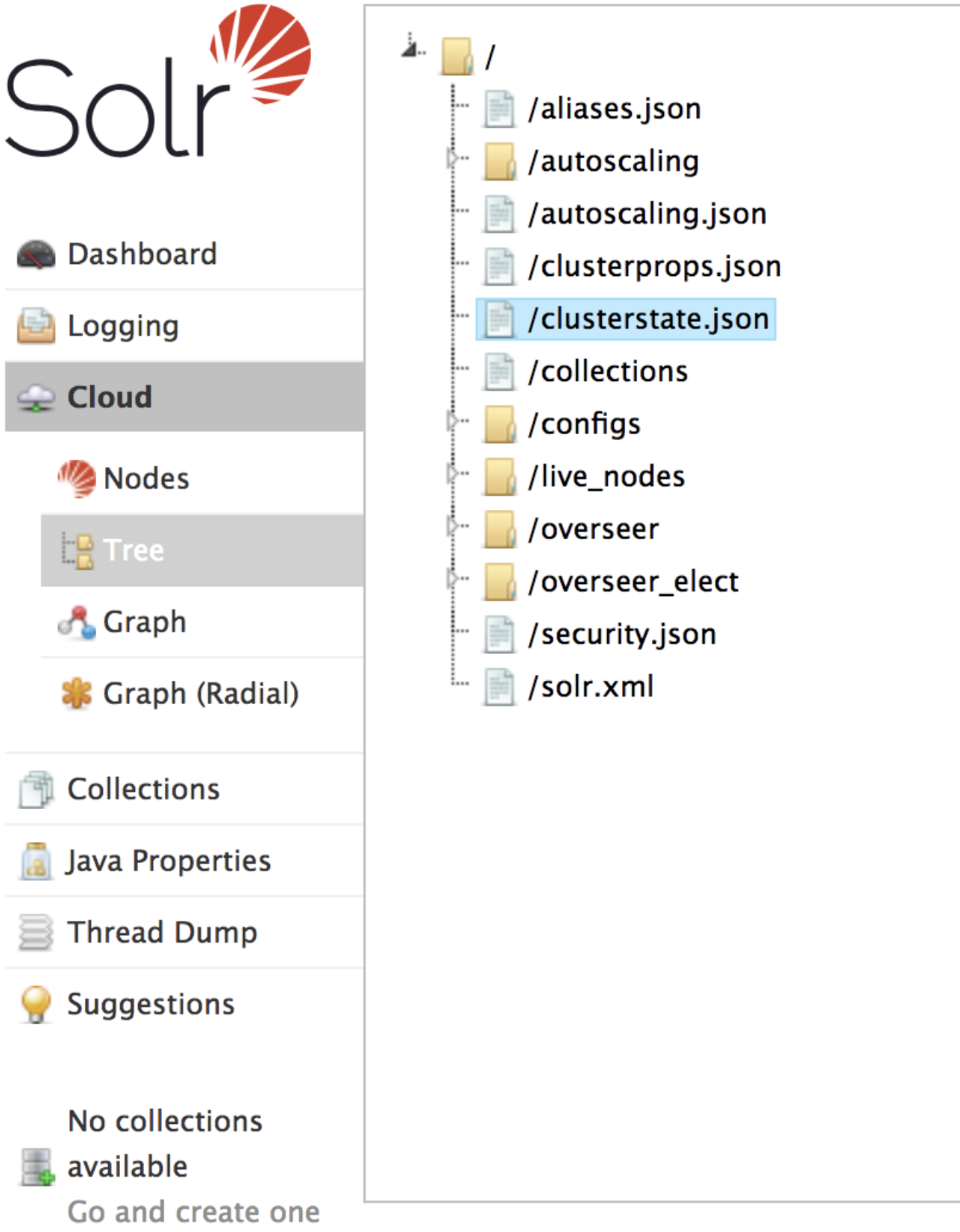
- `--get-solrxml <file>`: Downloads the cluster configuration file `solr.xml` from ZooKeeper to the local system.
- `--put-solrxml <file>`: Uploads the specified file to ZooKeeper as the cluster configuration file `solr.xml`.
- `[--set-property <name> <value>]`: Sets property names and values. Typically used in a deployment that is not managed by Cloudera Manager. For example, to configure a cluster to use TLS/SSL:

```
solrctl cluster --set-property urlScheme https
```

- `[--remove-property <name>]`: Removes the specified property.
- `[--get-clusterstate <file>]`: Downloads the `clusterstate.json` file from ZooKeeper to the local

system.

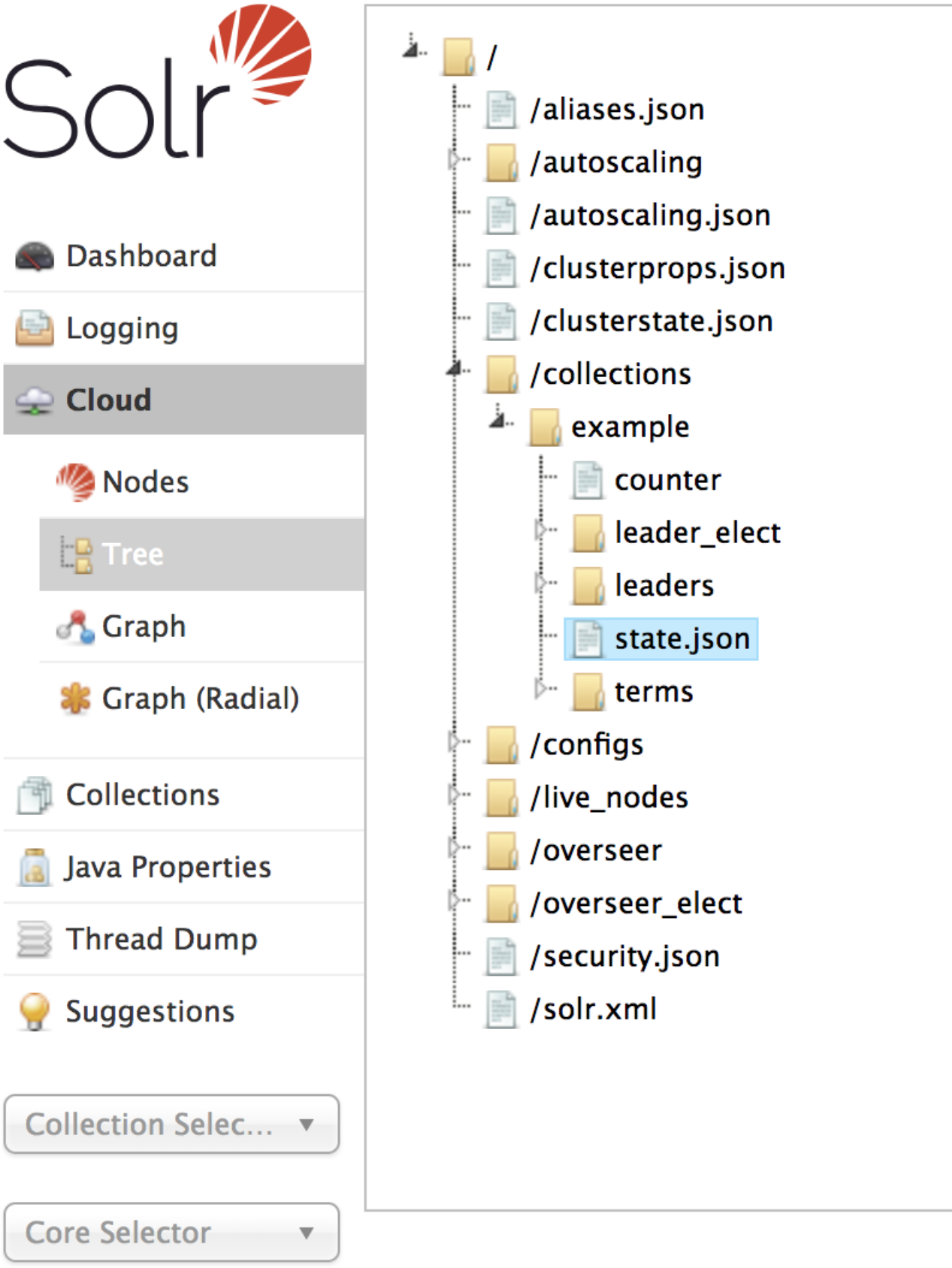
Note: In CDH 6, the `clusterstate.json` file is split per collection, and cluster state information is stored in separate, collection-specific files under `/collections/collectionname/state.json`. The `solrctl` command still has the `get-clusterstate` command, but the `clusterstate.json` file that it downloads is empty.



There are three ways to access a collection's `state.json` file in CDH 6.

Using the Solr Administration User Interface:

1. Open the **Solr Administration User Interface**.
2. Click **Cloud** > **Tree**, and expand **/collections**.
3. Open the collection of your choice.
4. Select the `state.json` file. The contents of the file are displayed on the right.



Using Solr's ZooKeeper CLI:

1. Run the `zkcli.sh` script as follows:

```
/usr/lib/solr/bin/zkcli.sh -zkhost zk01.example.com:2181 -cmd getfile
/solr/collections/example/state.json ./state_examplecollection.json
```

- Replace *zk01.example.com* with the hostname of your ZooKeeper server.
- Replace *example* with the name of your collection.
- Replace *./* with the path where you want to save the file.
- Replace *examplecollection* with the name you want to use for the saved file.

The `state.json` file is downloaded from ZooKeeper to the local filesystem.

2. Run `cat state_examplecollection.json` to display the contents of the `state.json` file.

Sample output would be similar to the following.

```
{"example":{
            "pullReplicas":"0",
            "replicationFactor":"1",
            "router":{"name":"compositeId"},
            "maxShardsPerNode":"1",
            "autoAddReplicas":"false",
            "nrtReplicas":"1",
            "tlogReplicas":"0",
            "shards":{"shard1":{
                "range":"80000000-7fffffff",
                "state":"active",
                "replicas":{"core_node2":{
                "dataDir":"hdfs://solr01.com:8020/solr/example/core_node2/data/",
                "base_url":"http://solr01.com.com:8983/solr",
                "node_name":"solr01.com.com:8983_solr",
                "type":"NRT",
                "force_set_state":"false",
              "ulogDir":"hdfs://solr01.com.com:8020/solr/example/core_node2/data/tlog",

                "core":"example_shard1_replica_n1",
                "shared_storage":"true",
                "state":"active",
                "leader":"true"}}}}}}[root@solr01 ~]#
```

Using Solr API:

1. Run the `curl` command and retrieve a json file that contains the status of every collection:

```
$ curl --negotiate -u :
'http://solr01.com:8983/solr/admin/collections?action=CLUSTERSTATUS'
```

> **Note:** The use of the `curl` command depends on whether Kerberos and SSL are enabled or not. If Kerberos is enabled, it needs `kinit`. If SSL is enabled, the http port changes to https and the port default value changes from 8983 to 8985. For more information, see Using Kerberos with curl.

```
sentry [--create-role <role>]
       [--drop-role <role>]
       [--add-role-group <role> <group>]
       [--delete-role-group <role> <group>]
       [--list-roles [-g <group>]]
       [--grant-privilege <role> <privilege>]
       [--revoke-privilege <role> <privilege>]
       [--list-privileges <role>]
       [--convert-policy-file <file> [-dry-run]]
```

Manages Sentry configuration. The following options are supported:

- `[--create-role <role>]`: Creates a new Sentry role using the specified name.
- `[--drop-role <role>]`: Deletes the specified Sentry role.
- `[--add-role-group <role> <group>]`: Assigns an existing Sentry role to the specified group.
- `[--delete-role-group <role> <group>]`: Removes the specified role from the specified group.
- `[--list-roles [-g group]]`: Lists all roles. Optionally, lists all roles assigned to the specified group when `-g <group>` is used.
- `[--grant-privilege <role> <privilege>]`: Grants the specified privilege to the specified role. For information on privilege syntax, see Using Roles and Privileges with Sentry.
- `[--revoke-privilege <role> <privilege>]`: Revokes the specified privilege from the specified role.
- `[--list-privileges <role>]`: Lists all privileges granted to the specified role.
- `[--convert-policy-file <file> [-dry-run]]`: Converts the specified policy file to permissions in the Sentry service. This command adds roles, assigns roles to group, and grants permissions. For CDH versions 5.12 and higher, the Solr **Gateway** role includes HDFS configuration files to facilitate Sentry integration. Because of this, the `<file>` path must be specified as `file:///path/to/file`, or else the `solrctl sentry` command attempts to locate the file in HDFS.

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

  - If a policy-file conversion will change the case of roles or groups, a warning is presented. Policy conversion can proceed, but if you have enabled document-level security and use role names as your tokens, you must re-index using the new lower case role names after conversion is complete.
  - If a policy-file conversion will change the case of roles or groups, creating a name collision, an error occurs and conversion cannot occur. In such a case, you must eliminate the collisions before proceeding. For example, you could rename or delete the roles or groups that cause a collision.

  The `-dry-run` option runs the process of converting the policy file, but sends the results to stdout without applying the changes. This can be used for quick turnaround when debugging failures.

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

Example solrctl Usage

This topic includes some examples of:

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

Using solrctl with an HTTP proxy

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

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

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

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

```
NO_PROXY='*' solrctl collection --create collectionName -s 3
```

Creating Replicas of Existing Shards

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

```
solrctl core --create <newCore> -p collection=<name> \
-p shard=<shard_to_replicate>
```

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

```
solrctl core --create collection1_shard1_replica2 \
-p collection=collection1 -p shard=shard1
```

Converting Instance Directories to Configs

Cloudera Search supports converting existing deployments that use instance directories to use configs:

1. Create a temporary config based on the existing instance directory. For example, if the instance directory name is `weblogs_config`:

```
solrctl config --create weblogs_config_temp weblogs_config \
-p immutable=false
```

2. Delete the existing instance directory. For example:

```
solrctl instancedir --delete weblogs_config
```

3. Create a config using the same name as the instance directory you just deleted, based on the temporary config you created earlier.

```
solrctl config --create weblogs_config weblogs_config_temp \
-p immutable=false
```

4. Delete the temporary config:

```
solrctl config --delete weblogs_config_temp
```

5. Reload the affected collection (`weblogs` in this example):

```
solrctl collection --reload weblogs
```

Configuring Sentry for a Collection

If you have enabled Apache Sentry for authorization, you must have UPDATE permission for the `admin=collections` object as well as the collection you are creating (`test_collection` in this example). You can also use the wildcard (`*`) to grant permissions to create any collection.

For more information on configuring Sentry and granting permissions, see Configuring Sentry Authorization for Cloudera Search.

To grant your user account (`jdoe` in this example) the necessary permissions:

1. Switch to the Sentry admin user (`solr` in this example) using `kinit`:

```
kinit solr@EXAMPLE.COM
```

2. Create a Sentry role for your user account:

```
solrctl sentry --create-role test_role
```

3. Map a group to this role. In this example, user `jdoe` is a member of the `eng` group:

```
solrctl sentry --add-role-group test_role eng
```

4. Grant `UPDATE` privileges to the `test_role` role for the `admin=collections` object and `test_collection` collection:

```
solrctl sentry --grant-privilege test_role 'admin=collections->action=UPDATE'
solrctl sentry --grant-privilege test_role 'collection=test_collection->action=UPDATE'
```

For more information on the Sentry privilege model for Cloudera Search, see Authorization Privilege Model for Cloudera Search.

Migrating Solr Replicas

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

Where possible, the Solr service routes requests to the proper host. Both `ADDREPLICA` and `DELETEREPLICA` Collections API calls can be sent to any host in the cluster. For more information on the Collections API, see the Collections API section of Apache Solr Reference Guide 4.10 (PDF).

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

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

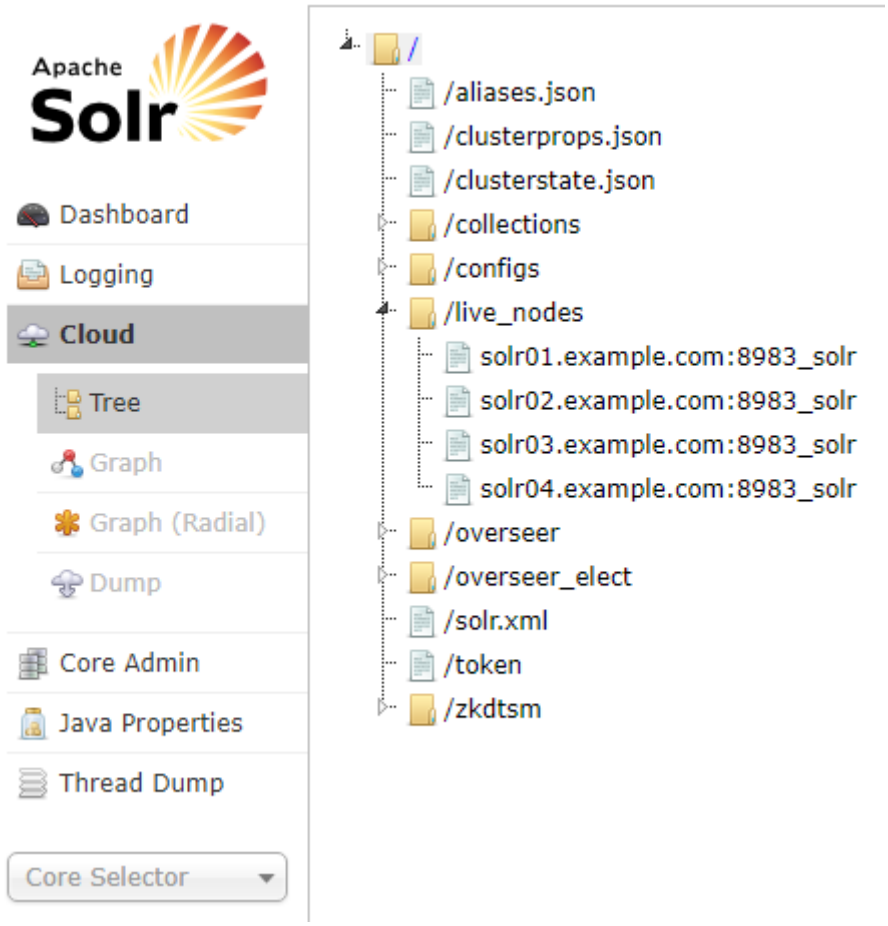
Cloudera recommends using API calls to create and unload cores. Do not use the Cloudera Manager Admin Console or the Solr Admin UI for these tasks.

This procedure uses the following names:

- Host names:
  - Origin: `solr01.example.com`.
  - Destination: `solr02.example.com`.
- Collection name: `email`
- Replicas:
  - The original replica `email_shard1_replica1`, which is on `solr01.example.com`.
  - The new replica `email_shard1_replica2`, which will be on `solr02.example.com`.

To migrate a replica to a new host:

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



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

2. Add the new replica on `solr02.example.com` using the ADDREPLICA API call.

```
http://solr01.example.com:8983/solr/admin/collections?action=ADDREPLICA&collection=email&shard=shard1&node=solr02.example.com:8983_solr
```

3. Verify that the replica creation succeeds and moves from recovery state to **ACTIVE**. You can check the replica status in the Cloud view, which can be found at a URL similar to: `http://solr02.example.com:8983/solr/#/~cloud`.

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

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

```
http://solr02.example.com:8983/solr/email_shard1_replica2/replication?command=details
```

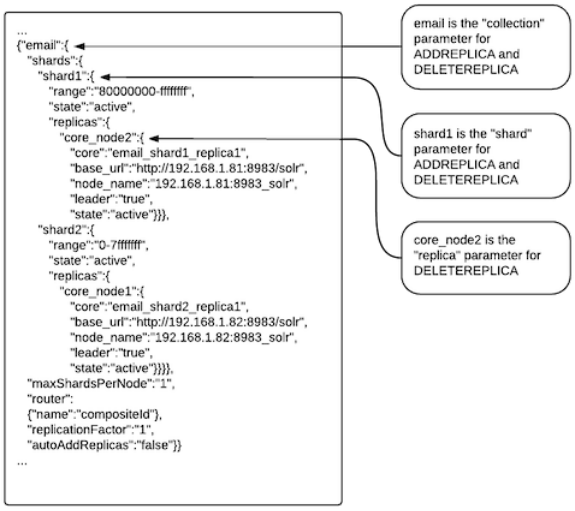
Accessing this URI returns an XML document that contains content about replication progress. A snippet of the XML content might appear as follows:

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

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

```
http://solr01.example.com:8983/solr/admin/collections?action=clusterstatus&wt=json&indent=true
```

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



5. Delete the old replica on `solr01.example.com` server using the `DELETEREPLICA` API call:

```
http://solr01.example.com:8983/solr/admin/collections?action=DELETEREPLICA&collection=email&shard=shard1&replica=core_node2
```

The `DELETEREPLICA` call removes the `datadir`.

Backing Up and Restoring Cloudera Search

Important: The following documentation is about backing up and restoring Cloudera Search, which is based on the SolrCloud implementation of Apache Solr. Cloudera Search does not support backups using the Solr replication handler.

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

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

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

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

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

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

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

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

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

Prerequisites

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

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

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

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

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

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

To edit the `solr.xml` file:

1. Download `solr.xml` from ZooKeeper:

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

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

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

Sentry Configuration

As part of backup/restore mechanism, the following APIs are introduced. The following table shows the required Sentry authorization privileges for each action. For more information on configuring Sentry privileges, see Configuring Sentry Authorization for Cloudera Search.

Table 4: Solr Snapshot Sentry Permissions

Required Privileges	Action
`admin=collections->action=UPDATE collection=<collectionName>->action=UPDATE`	CREATESNAPSHOT
	DELETESNAPSHOT
	RESTORE
`admin=collections->action=QUERY collection=<collectionName>->action=QUERY`	LISTSNAPSHOTS
	BACKUP

Backing Up a Solr Collection

Use the following procedure to back up a Solr collection. For more information on the commands used, see Cloudera Search Backup and Restore Command Reference on page 71.

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

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

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

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

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

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

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

```
solrctl collection --create-snapshot tweets-$(date +%Y%m%d%H%M) -c tweets
Successfully created snapshot with name tweets-201803281043 for collection tweets
```

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

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

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

   For example:

```
hdfs dfs -mkdir -p /path/to/backup-staging/tweets-201803281043
hdfs dfs -chown :solr /path/to/backup-staging/tweets-201803281043
solrctl collection --prepare-snapshot-export tweets-201803281043 -c tweets \
-d /path/to/backup-staging/tweets-201803281043
```

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

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

   For example:

   - HDFS protocol:

```
solrctl collection --export-snapshot tweets-201803281043 -s
/path/to/backup-staging/tweets-201803281043 \
-d hdfs://nn01.example.com:8020/path/to/backups
```

   - WebHDFS protocol:

```
solrctl collection --export-snapshot tweets-201803281043 -s
/path/to/backup-staging/tweets-201803281043 \
-d webhdfs://nn01.example.com:20101/path/to/backups
```

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

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

   For example:

```
solrctl collection --export-snapshot tweets-201803281043 -c tweets -d /path/to/backups/
```

4. Delete the snapshot:

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

For example:

```
solrctl collection --delete-snapshot tweets-201803281043 -c tweets
```

Restoring a Solr Collection

Use the following procedure to restore a Solr collection. For more information on the commands used, see Cloudera Search Backup and Restore Command Reference on page 71.

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

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

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

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

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

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

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

   For example:

   - HDFS protocol:

```
hadoop distcp hdfs://nn01.example.com:8020/path/to/backups/tweets-201803281043
/path/to/restore-staging
```

   - WebHDFS protocol:

```
hadoop distcp webhdfs://nn01.example.com:20101/path/to/backups/tweets-201803281043
/path/to/restore-staging
```

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

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

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

```
solrctl collection --restore tweets -l /path/to/restore-staging -b tweets-201803281043
 -i restore-tweets
```

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

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

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

```
solrctl collection --request-status restore-tweets
 <?xml version="1.0" encoding="UTF-8"?> <response> <lst name="responseHeader"> <int
name="status"> 0</int> <int name="QTime"> 1</int> </lst> \
```

```
<lst name="status"> <str name="state"> completed</str> <str name="msg"> found
restore-tweets in completed tasks</str> </lst> </response>
```

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

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

Cloudera Search Backup and Restore Command Reference

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

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

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

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

Create a snapshot

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

Description: Creates a named snapshot for the specified collection.

Delete a snapshot

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

Description: Deletes the specified snapshot for the specified collection.

Describe a snapshot

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

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

List all snapshots

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

Description: Lists all snapshots for the specified collection.

Prepare snapshot for export to a remote cluster

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

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

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

If you are running the snapshot export command on a remote cluster, specify the HDFS protocol (such as WebHDFS or HFTP) to be used for accessing the Lucene index files corresponding to the snapshot on the source cluster. For more

information about the Hadoop DistCP utility, see Copying Cluster Data Using DistCp. This configuration is driven by the `-p` option which expects a fully qualified URI for the root filesystem on the source cluster, for example `webhdfs://namenode.example.com:20101/`.

Export snapshot to local cluster

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

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

Export snapshot to remote cluster

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

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

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

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

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

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

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

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

Restore from a local snapshot

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

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

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

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

Because the restore operation can take a long time to complete depending on the size of the exported snapshot, it is run asynchronously. The `-i` configuration parameter specifies a unique identifier for tracking operation. For more information, see Check the status of an operation on page 73.

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

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

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

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

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

Check the status of an operation

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

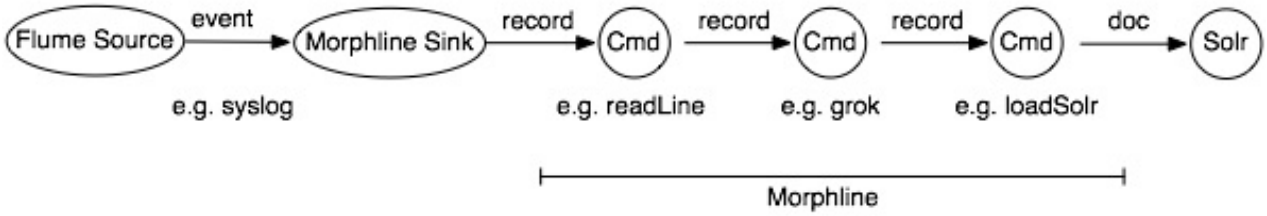
Description: Displays the status of the specified operation. The status can be one of the following:

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

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

Extracting, Transforming, and Loading Data With Cloudera Morphlines

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



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

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

This integration enables the following:

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

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

The following components execute morphlines:

- MapReduceIndexerTool
- Flume Morphline Solr Sink and Flume MorphlineInterceptor

Cloudera also provides a corresponding Cloudera Search Tutorial.

Data Morphlines Support

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

Not only structured data, but also arbitrary binary data can be passed into and processed by a morphline. By convention, a record can contain an optional field named `_attachment_body`, which can be a Java `java.io.InputStream` or Java `byte[]`. Optionally, such binary input data can be characterized in more detail by setting the fields named

`_attachment_mimetype` (such as `application/pdf`) and `_attachment_charset` (such as `UTF-8`) and `_attachment_name` (such as `cars.pdf`), which assists in detecting and parsing the data type.

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

Important: Cloudera Search does not support Solr *contrib* modules, such as DataImportHandler. Morphlines, Spark Crunch indexer, MapReduce and Lily HBase indexers are part of the Cloudera Search product itself, therefore they are supported.

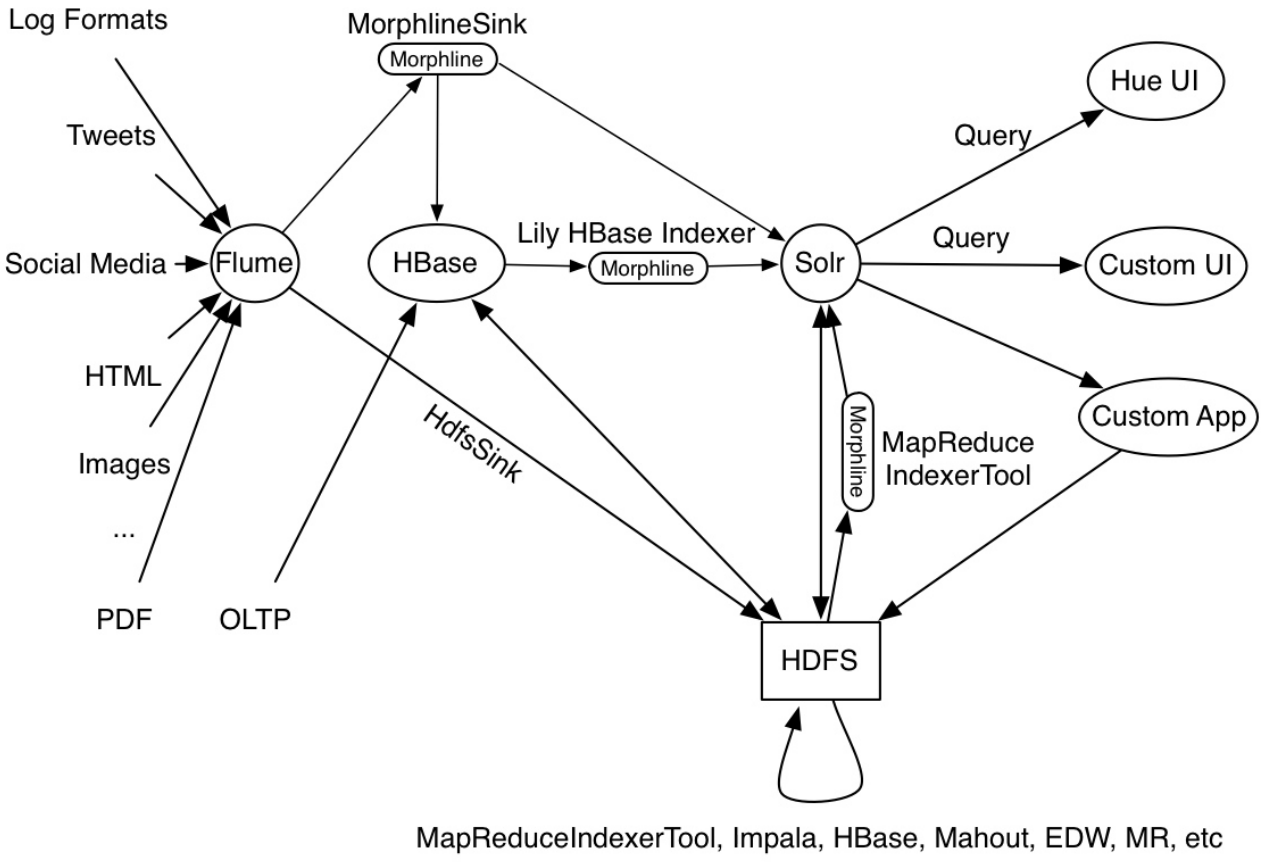
How Morphlines Act on Data

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

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

A command can extract, clean, transform, join, integrate, enrich and decorate records in many other ways. For example, a command can join records with external data sources such as relational databases, key-value stores, local files or IP Geo lookup tables. It can also perform tasks such as DNS resolution, expand shortened URLs, fetch linked metadata from social networks, perform sentiment analysis and annotate the record accordingly, continuously maintain statistics for analytics over sliding windows, compute exact or approximate distinct values and quantiles.

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



Morphline Characteristics

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

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

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

How Morphlines Are Implemented

Cloudera Search includes several maven modules that contain morphline commands for integration with Apache Solr including SolrCloud, flexible log file analysis, single-line records, multi-line records, CSV files, regular expression based pattern matching and extraction, operations on record fields for assignment and comparison, operations on record fields with list and set semantics, if-then-else conditionals, string and timestamp conversions, scripting support for dynamic Java code, a small rules engine, logging, metrics and counters, integration with Avro, integration with Apache Tika parsers, integration with Apache Hadoop Sequence Files, auto-detection of MIME types from binary data using Apache Tika, and decompression and unpacking of arbitrarily nested container file formats, among others. These are described in the following chapters.

Example Morphline Usage

The following examples show how you can use morphlines.

Using Morphlines to Index Avro

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

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

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

2. Inspect the schema of the Avro file:

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

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

  ...

  ]
}
```

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

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

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

The Solr output schema omits some Avro input fields, such as `user_statuses_count`. If your data includes Avro input fields that are not included in the Solr output schema, you may want to make changes to data as it is ingested. For example, suppose you need to rename the input field `user_screen_name` to the output field `username`. Also suppose that the time format for the `created_at` field is `yyyy-MM-dd'T'HH:mm:ss'Z'`. Finally, suppose any unknown fields present are to be removed. Recall that Solr throws an exception on any attempt to load a document that contains a field that is not specified in `schema.xml`.

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

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

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

# Specify an array of one or more morphlines, each of which defines an ETL
# transformation chain. A morphline consists of one or more potentially
# nested commands. A morphline is a way to consume records such as Flume events,
# HDFS files or blocks, turn them into a stream of records, and pipe the stream
# of records through a set of easily configurable transformations on its way to
# Solr.
morphlines : [
  {
    # Name used to identify a morphline. For example, used if there are multiple
    # morphlines in a morphline config file.
    id : morphline1

    # Import all morphline commands in these java packages and their subpackages.
    # Other commands that may be present on the classpath are not visible to this
    # morphline.
    importCommands : ["org.kitesdk.**", "org.apache.solr.**"]

    commands : [
      {
        # Parse Avro container file and emit a record for each Avro object
        readAvroContainer {
          # Optionally, require the input to match one of these MIME types:
          # supportedMimeTypes : [avro/binary]

          # Optionally, use a custom Avro schema in JSON format inline:
          # readerSchemaString : """<json can go here>"""

          # Optionally, use a custom Avro schema file in JSON format:
          # readerSchemaFile : /path/to/syslog.avsc
        }
      }

      {
        # Consume the output record of the previous command and pipe another
        # record downstream.
        #
        # extractAvroPaths is a command that uses zero or more Avro path
       # excodeblockssions to extract values from an Avro object. Each excodeblockssion

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

```
            created_at : /created_at
            text : /text
          }
        }
      }

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

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

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

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

Using Morphlines with Syslog

The following example illustrates using a morphline to extract information from a `syslog` file. A syslog file contains semi-structured lines of the following form:

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

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

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

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

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

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

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

    commands : [
      {
        readLine {
          charset : UTF-8
        }
      }

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

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

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

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

```
    ]
  }
]
```

Next Steps

Learn more about morphlines and Kite. Cloudera Search for CDH 6.1 includes a build of Kite 0.10.0 that includes Kite 0.17.0 fixes and features. For more information, see:

- Kite Software Development Kit
- Morphlines Reference Guide

Indexing Data Using Cloudera Search

There are generally two approaches to indexing data using Cloudera Search:

1. Near real time (NRT) indexing
2. Batch indexing

Near real time indexing is generally used when new data needs to be returned in query results in time frames measured in seconds, whereas batch indexing is useful for situations where large amounts of data is indexed at regular intervals, or for indexing a new dataset for the first time.

Near real time indexing generally uses a framework such as Apache Flume or Apache Kafka to continuously ingest and index data. The Lily HBase Indexer can also be used for NRT indexing on Apache HBase tables.

Batch indexing usually relies on MapReduce/YARN jobs to periodically index large datasets. The Lily HBase Indexer can also be used for batch indexing HBase tables.

For more information, see:

Near Real Time Indexing Using Cloudera Search

For near real time (NRT) indexing, there are multiple options for ingesting and indexing data:

- Using Flume with the MorphlinesSolrSink
- Using the Lily HBase Indexer to index updates to HBase tables

Near Real Time Indexing Using Flume

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

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

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

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

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

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

Flume is included in CDH. If you are using Cloudera Manager, add the Flume service using the **Add a Service** wizard.

For exercises demonstrating using Flume with Solr for NRT indexing, see Cloudera Search Tutorial.

For Flume configuration options relevant to Solr, see the following topics:

Flume MorphlineSolrSink Configuration Options

The MorphlineSolrSink is a Flume sink used to ingest and index documents into Cloudera Search. For Cloudera Manager environments, the Flume agent is configured using procedures described in Configuring the Flume Agents. For unmanaged environments, you can use the standard configuration file `flume.conf` to configure Flume agents, including their sources, sinks, and channels. For more information about `flume.conf`, see the Flume User Guide.

Flume Morphline SolrSink supports the following configuration options (required options in **bold**):

Property Name	Default	Description
`type`		Must be set to the fully qualified class name (FQCN) `org.apache.flume.sink.solr.morphline.MorphlineSolrSink`.
`channel`		Specifies the channel to use for transferring records. For more information, see Flume Channels in the Flume User Guide.
`morphlineFile`		The location of the morphline configuration file. • For Cloudera Manager deployments, use: `<agentName>.sinks.<sinkName>.morphlineFile=morphlines.conf` • For unmanaged deployments, provide the relative or absolute path on the local filesystem to the morphline configuration file. For example, `/etc/flume-ng/conf/tutorialReadAvroContainer.conf`
`batchSize`	100	The maximum number of Flume events per transaction. The transaction is committed when the specified `batchSize` or `batchDurationMillis` is reached, whichever comes first.
`batchDurationMillis`	1000	The maximum duration for a transaction, in milliseconds. The transaction is committed when the specified `batchSize` or `batchDurationMillis` is reached, whichever comes first.
`indexerClass`	`org.apache.flume.sink.solr.morphline.MorphlineSolrIndexer`	The FQCN of a class implementing `org.apache.flume.sink.solr.morphline.SolrIndexer`.
`morphlineId`	`null`	The name of the morphline to use when there is more than one morphline in a morphline configuration file.

This example shows a section for a MorphlineSolrSink named `solrSink` for an agent named `agent`:

```
agent.sinks.solrSink.type = org.apache.flume.sink.solr.morphline.MorphlineSolrSink
agent.sinks.solrSink.channel = memoryChannel
```

```
agent.sinks.solrSink.batchSize = 100
agent.sinks.solrSink.batchDurationMillis = 1000
agent.sinks.solrSink.morphlineFile = /etc/flume-ng/conf/morphline.conf
agent.sinks.solrSink.morphlineId = morphline1
```

Note: This example uses a Flume MemoryChannel to easily get started. For production environments, it is often more appropriate to configure a Flume FileChannel instead, which is a high performance transactional persistent queue.

Flume MorphlineInterceptor Configuration Options

Flume can modify and drop events using Interceptors, which can be attached to any Flume source. The MorphlineInterceptor runs morphline transformations on intercepted events. For example, a 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. Interceptors can be used for content-based routing in a Flume topology.

Flume also supports multiplexing event flows by defining a flow multiplexer that can replicate or selectively route an event to specific channels. The Flume User Guide example demonstrates a source from agent `foo` fanning out the flow to three different channels. This can be a ***replicating*** or ***multiplexing***fan-out. In a replicating fan-out, each event is sent to all three channels. In a multiplexing fan-out, an event is sent to a subset of available channels when an event attribute matches a specified value. For example, if an event attribute called `stream.type` is set to `application/pdf`, it goes to `channel1` and `channel3`. If the attribute is set to `avro/binary`, it goes to `channel2`. If a channel is unavailable, then an exception is thrown, and the event is replayed after the channel becomes available again. You can configure this mapping in the `flume.conf` file.

The MorphlineInterceptor supports the following configuration options (required options in **bold**):

Property Name	Default	Description
`type`		Must be set to the fully qualified class name (FQCN) `org.apache.flume.sink.solr.morphline.MorphlineInterceptor$Builder`.
`morphlineFile`		The location of the morphline configuration file. • For Cloudera Manager deployments, use: `<agentName>.sources.<sourceName>.interceptors.<interceptorName>.morphlineFile = morphlines.conf` • For unmanaged deployments, provide the relative or absolute path on the local filesystem to the morphline configuration file. For example, `/etc/flume-ng/conf/morphline.conf`.
`morphlineId`	`null`	The name of the morphline to use when there is more than one morphline in a morphline configuration file.

This example shows a section for a MorphlineInterceptor named `morphlineinterceptor`for a source named `avroSrc` for an agent named `agent`:

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

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

Flume Solr UUIDInterceptor Configuration Options

Flume can modify and drop events using Interceptors, which can be attached to any Flume source. The Solr UUIDInterceptor sets a universally unique 128-bit identifier (such as `f692639d-483c-1b5f-cd61-183cb1726ae0`) on each event.

Cloudera recommends assigning UUIDs to events as early as possible (for example, in the first Flume source of your data flow). This allows you to de-duplicate events that are duplicated as a result of replication or re-delivery in a Flume pipeline that is designed for high availability and high performance. If available, application-level UUIDs are preferable to auto-generated UUIDs because they enable subsequent updates and deletion of the document in Solr using that key. If application-level UUIDs are not present, you can use UUIDInterceptor to automatically assign UUIDs to document events.

The UUIDInterceptor supports the following configuration options (required options in **bold**):

Property Name	Default	Description
`type`		Must be set to the fully qualified class name (FQCN)`org.apache.flume.sink.solr.morphline.UUIDInterceptor$Builder`.
`headerName`	`id`	The name of the Flume header to use for setting the UUID.
`preserveExisting`	`true`	Determines whether to preserve existing UUID headers.
`prefix`	`""`	Specifies a string to prepend to each generated UUID.

For examples, see BlobHandler and BlobDeserializer.

Flume Solr BlobHandler Configuration Options

Flume accepts events by HTTP `POST` and `GET` operations using the HTTPSource Flume source.

HTTPSource transforms JSON input into events by default. You can also configure a BlobHandler for HTTPSource to return events containing the request parameters along with the ***binary large object*** (BLOB) uploaded with the request. Because the entire BLOB is buffered in RAM, this usage is not generally appropriate for very large objects.

The Flume Solr BlobHandler supports the following configuration options (required options in **bold**):

Property Name	Default	Description
`handler`		Must be set to the fully qualified class name (FQCN) `org.apache.flume.sink.solr.morphline.BlobHandler`.
`handler.maxBlobLength`	`100000000` (100 MB)	Specifies the maximum number of bytes to read and buffer per request.

This example shows a section for an HTTPSource named `httpSrc` with a BlobHandler for an agent named `agent`:

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

```
#agent.sources.httpSrc.interceptors.uuidinterceptor.preserveExisting = false
#agent.sources.httpSrc.interceptors.uuidinterceptor.prefix = flume01.example.com
agent.sources.httpSrc.channels = memoryChannel
```

Flume Solr BlobDeserializer Configuration Options

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

By default, `SpoolDirectorySource` uses the newline (`\n`) delimiter to split input into Flume events. You can change this behavior by configuring the Solr `BlobDeserializer` to read ***binary large objects*** (BLOBs) from `SpoolDirectorySource`. Generally, each file is one BLOB (such as a PDF or image file). Because the entire BLOB is buffered in RAM, this usage is not generally appropriate for very large objects.

The Solr `BlobDeserializer` supports the following configuration options (required options in **bold**):

Property Name	Default	Description
`deserializer`		Must be set to the fully qualified class name (FQCN) `org.apache.flume.sink.solr.morphline.BlobDeserializer$Builder`.
`deserializer.maxBlobLength`	`100000000` (100 MB)	Specifies the maximum number of bytes to read and buffer per request.

This example shows a section for a `SpoolDirectorySource` named `spoolSrc` with a `BlobDeserializer` for an agent named `agent`:

```
agent.sources.spoolSrc.type = spooldir
agent.sources.spoolSrc.spoolDir = /tmp/myspooldir
agent.sources.spoolSrc.ignorePattern = \.
agent.sources.spoolSrc.deserializer =
org.apache.flume.sink.solr.morphline.BlobDeserializer$Builder
agent.sources.spoolSrc.deserializer.maxBlobLength = 2000000000
agent.sources.spoolSrc.batchSize = 1
agent.sources.spoolSrc.fileHeader = true
agent.sources.spoolSrc.fileHeaderKey = resourceName
agent.sources.spoolSrc.interceptors = uuidinterceptor
agent.sources.spoolSrc.interceptors.uuidinterceptor.type =
org.apache.flume.sink.solr.morphline.UUIDInterceptor$Builder
agent.sources.spoolSrc.interceptors.uuidinterceptor.headerName = id
#agent.sources.spoolSrc.interceptors.uuidinterceptor.preserveExisting = false
#agent.sources.spoolSrc.interceptors.uuidinterceptor.prefix = flume01.example.com
agent.sources.spoolSrc.channels = memoryChannel
```

Lily HBase Near Real Time Indexing for Cloudera Search

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

To accommodate the HBase ingest load, you can run as many Lily HBase Indexer services on different hosts as required. Because the indexing work is shared by all indexers, you can scale the service by adding more indexers. The recommended number of indexer is 1 for each HBase RegionServer but in a High Availability environment five worker nodes is the minimum for acceptable performance and reliability. You can co-locate Lily HBase Indexer services with Solr servers

on the same set of hosts. RegionServers can also be co-locate with Lily HBase Indexer on the same host to improve performance.

> **Note:** Specific workloads and usage patterns might require additional fine-tuning beyond these general recommendation.

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

Enabling Cluster-wide HBase Replication

The Lily HBase Indexer is implemented using HBase replication, presenting indexers as RegionServers of the worker cluster. This requires HBase replication on the HBase cluster, as well as the individual tables to be indexed. To enable replication:

Cloudera Manager:

1. Go to **HBase service** > **Configuration** > **Category** > **Backup**
2. Select the **Enable Replication** checkbox.
3. Set **Replication Source Ratio** to `1`.
4. Set **Replication Batch Size** to `1000`.
5. Click **Save Changes**.
6. Restart the HBase service (**HBase service** > **Actions** > **Restart**).

Unmanaged:

1. Add the following properties within the `<configuration>` tags in `/etc/hbase/conf/hbase-site.xml` on every HBase cluster node:

```
  <property>
    <name>hbase.replication</name>
    <value>true</value>
  </property>
  <!-- Source ratio of 100% makes sure that each SEP consumer is actually
       used (otherwise, some can sit idle, especially with small clusters) -->
  <property>
    <name>replication.source.ratio</name>
    <value>1.0</value>
  </property>
  <!-- Maximum number of hlog entries to replicate in one go. If this is
       large, and a consumer takes a while to process the events, the
       HBase rpc call will time out. -->
  <property>
    <name>replication.source.nb.capacity</name>
    <value>1000</value>
  </property>
</configuration>
```

2. Restart the HBase services on all HBase cluster nodes.

Adding the Lily HBase Indexer Service

In Cloudera Manager, the Lily HBase Indexer service is called **Key-Value Store Indexer**, and the service role is called **Lily HBase Indexer**. To add the service, follow the instructions in Adding a Service.

Configure HBase Dependency for Lily HBase NRT Indexer Service

Before starting Lily HBase NRT Indexer services, you must configure individual services with the location of the ZooKeeper ensemble that is used by the target HBase cluster. In Cloudera Manager, this is handled automatically when you set the **HBase Service** dependency (**Key-Value Store Indexer service** > **Configuration**).

For unmanaged environments:

1. Add the following property to `/etc/hbase-solr/conf/hbase-indexer-site.xml` on the hosts you are using to run the Lily HBase Indexer service. Replace the ZooKeeper quorum with the value for `hbase.zookeeper.quorum` from `/etc/hbase/conf/hbase-site.xml`.

   Unlike other ZooKeeper quorum configuration properties, the `hbase.zookeeper.quorum` property does not include the ZooKeeper port or Znode:

```
<property>
   <name>hbase.zookeeper.quorum</name>
   <value>zk01.example.com,zk02.example.com,zk03.example.com</value>
</property>
```

2. Configure all Lily HBase Indexers to use a ZooKeeper quorum to coordinate with one another. You can use the same ZooKeeper quorum as the HBase service. Add the following property to `/etc/hbase-solr/conf/hbase-indexer-site.xml`, and replace the hostnames with your ZooKeeper hostnames.

   For this configuration property, the ZooKeeper designation includes the port:

```
<property>
   <name>hbaseindexer.zookeeper.connectstring</name>
   <value>zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181</value>
</property>
```

Configuring Lily HBase Indexer Security

The Lily HBase indexer supports Kerberos for authentication, and Apache Sentry for authorization. For more information, see Configuring Lily HBase Indexer Security on page 94.

Starting the Lily HBase NRT Indexer Service

You can use Cloudera Manager to start the Lily HBase Indexer Service (**Key-Value Store Indexer service** > **Actions** > **Start**). In unmanaged deployments, you can start or restart a Lily HBase Indexer Daemon manually on a host using the following command:

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

After starting the Lily HBase NRT Indexer Services, verify that all daemons are running using the `jps` tool from the Oracle JDK, which you can obtain from the Java SE Downloads page. For example:

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

Once the service is running, you can create and manage indexers. Continue to Using the Lily HBase NRT Indexer Service on page 88.

Using the Lily HBase NRT Indexer Service

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

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

Enabling Replication on HBase Column Families

Ensure that cluster-wide HBase replication is enabled, as described in Enabling Cluster-wide HBase Replication on page 87. Use the HBase shell to define column-family replication settings.

For every existing table, set the REPLICATION_SCOPE on every column family that you want to index:

```
hbase shell
hbase shell> disable 'sample_table'
hbase shell> alter 'sample_table', {NAME => 'columnfamily1', REPLICATION_SCOPE => 1}
hbase shell> enable 'sample_table'
```

For every new table, set the REPLICATION_SCOPE on every column family that you want to index using a command such as the following:

```
hbase shell
hbase shell> create 'test_table', {NAME => 'testcolumnfamily', REPLICATION_SCOPE => 1}
```

Creating a Collection in Cloudera Search

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

```
solrctl instancedir --generate $HOME/hbase_collection_config
## Edit $HOME/hbase_collection_config/conf/schema.xml as needed ##
## If you are using Sentry for authorization, copy solrconfig.xml.secure to solrconfig.xml
 as follows: ##
## cp $HOME/hbase_collection_config/conf/solrconfig.xml.secure
$HOME/hbase_collection_config/conf/solrconfig.xml ##
solrctl instancedir --create hbase_collection_config $HOME/hbase_collection_config
solrctl collection --create hbase_collection -s <numShards> -c hbase_collection_config
```

Creating a Lily HBase Indexer Configuration File

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

An indexer configuration XML file must refer to the MorphlineResultToSolrMapper implementation and point to the location of a Morphline configuration file, as shown in the following morphline-hbase-mapper.xml indexer configuration file. For Cloudera Manager managed environments, set morphlineFile to the relative path morphlines.conf. For unmanaged environments, specify the absolute path to a morphlines.conf that exists on the Lily HBase Indexer host. Make sure the file is readable by the HBase system user (hbase by default).

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

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

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

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

</indexer>
```

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

Creating a Morphline Configuration File

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

If you are using Cloudera Manager, the `morphlines.conf` file is edited within Cloudera Manager (**Key-Value Store Indexer service** > **Configuration** > **Category** > **Morphlines** > **Morphlines File**).

For unmanaged environments, create the `morphlines.conf` file on the Lily HBase Indexer host:

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

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

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

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

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

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

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

Understanding the `extractHBaseCells` Morphline Command

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

Each mapping has:

- The `inputColumn` parameter, which specifies the data from HBase for populating a field in Solr. It has the form of a column family name and qualifier, separated by a colon. The qualifier portion can end in an asterisk, which

is interpreted as a wildcard. In this case, all matching column-family and qualifier expressions are used. The following are examples of valid `inputColumn` values:

  - `mycolumnfamily:myqualifier`
  - `mycolumnfamily:my*`
  - `mycolumnfamily:*`

- The `outputField` parameter specifies the morphline record field to which to add output values. The morphline record field is also known as the Solr document field. Example: `first_name`.
- Dynamic output fields are enabled by the `outputField` parameter ending with a wildcard (`*`). For example:

```
inputColumn : "mycolumnfamily:*"
outputField : "belongs_to_*"
```

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

```
put 'table_name' , 'row1' , 'mycolumnfamily:1' , 'foo'
put 'table_name' , 'row1' , 'mycolumnfamily:9' , 'bar'
```

Then the fields of the Solr document are as follows:

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

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

  - `type:byte[] copies the byte array unmodified into the record output field`
  - `type:int converts with org.apache.hadoop.hbase.util.Bytes.toInt`
  - `type:long converts with org.apache.hadoop.hbase.util.Bytes.toLong`
  - `type:string converts with org.apache.hadoop.hbase.util.Bytes.toString`
  - `type:boolean converts with org.apache.hadoop.hbase.util.Bytes.toBoolean`
  - `type:float converts with org.apache.hadoop.hbase.util.Bytes.toFloat`
  - `type:double converts with org.apache.hadoop.hbase.util.Bytes.toDouble`
  - `type:short converts with org.apache.hadoop.hbase.util.Bytes.toShort`
  - `type:bigdecimal converts with org.apache.hadoop.hbase.util.Bytes.toBigDecimal`

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

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

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

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

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

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

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

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

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

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

1. If your cluster has security enabled, create a Java Authentication and Authorization Service (JAAS) configuration file named `jaas.conf` in your home directory with the following contents:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=false
  useTicketCache=true
    principal="jdoe@EXAMPLE.COM";
};
```

   Replace `jdoe@EXAMPLE.COM` with your user principal. Your user account must have `WRITE` permission to create an indexer. For more information, see Configuring the Lily HBase Indexer Service to Use the Sentry Service on page 94.

2. If your cluster has security enabled, authenticate with the user principal specified in your `jaas.conf` file:

```
kinit jdoe@EXAMPLE.COM
```

3. Run the following command to register your indexer configuration file with the indexer service. If you have enabled Sentry authorization, add the `--http http://lily01.example.com:11060/indexer/` parameter, replacing `lily01.example.com` with your Lily HBase Indexer hostname:

```
hbase-indexer add-indexer \
--name myIndexer \
--indexer-conf $HOME/morphline-hbase-mapper.xml \
--connection-param solr.zk=zk01.example.com,zk02.example.com,zk03.example.com/solr \
--connection-param solr.collection=hbase_collection \
--zookeeper zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181
```

Verify that the indexer was successfully created as follows. If you have enabled Sentry authorization, add the `--http http://lily01.example.com:11060/indexer/` parameter, replacing `lily01.example.com` with your Lily HBase Indexer hostname:

```
hbase-indexer list-indexers -zookeeper
zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181
Number of indexes: 1

myIndexer
  + Lifecycle state: ACTIVE
  + Incremental indexing state: SUBSCRIBE_AND_CONSUME
  + Batch indexing state: INACTIVE
  + SEP subscription ID: Indexer_myIndexer
  + SEP subscription timestamp: 2013-06-12T11:23:35.635-07:00
  + Connection type: solr
  + Connection params:
    + solr.collection = hbase-collection1
```

```
    + solr.zk = localhost/solr
  + Indexer config:
      110 bytes, use -dump to see content
  + Batch index config:
      (none)
  + Default batch index config:
      (none)
  + Processes
    + 1 running processes
    + 0 failed processes
```

Use the `update-indexer` and `delete-indexer` command-line options of the `hbase-indexer` utility to manipulate existing Lily HBase Indexers. If you have enabled Sentry authorization, you must include the `--http http://lily01.example.com:11060/indexer/` parameter in all commands.

For more help, use the following commands:

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

The `morphlines.conf` configuration file must be present on every host that runs an indexer. For Cloudera Manager environments, this is handled automatically.

Morphline configuration files can be changed without re-creating the indexer itself, but you must restart the Lily HBase Indexer service for the changes to take effect.

Verifying that Indexing Works

Add rows to the indexed HBase table. For example:

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

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

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

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

For Cloudera Manager environments, the logging configuration is modified as follows:

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

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

Using the Indexer HTTP Interface

Beginning with CDH 5.4 the Lily HBase Indexer includes an HTTP interface for the `list-indexers`, `create-indexer`, `update-indexer`, and `delete-indexer` commands. This interface can be secured with Kerberos for authentication and Apache Sentry for authorization. For information on configuring security for the Lily HBase Indexer service, see Configuring Lily HBase Indexer Security on page 88.

By default, the `hbase-indexer`command line client does not use the HTTP interface. Use the HTTP interface to take advantage of the features it provides, such as Kerberos authentication and Sentry integration. The `hbase-indexer`

command supports two additional parameters to the `list-indexers`, `create-indexer`, `delete-indexer`, and `update-indexer` commands:

- `--http`: An HTTP URI for the HTTP interface. By default, this URI is of the form `http://lily01.example.com:11060/indexer/`. If this parameter is specified, the Lily HBase Indexer uses the HTTP API. If this parameter is not specified, the indexer communicates directly with ZooKeeper.
- `--jaas`: Specifies a Java Authentication and Authorization Service (JAAS) configuration file. This is only necessary for Kerberos-enabled deployments.

> **Note:** Make sure that you use fully qualified domain names (FQDN) when specifying hostnames for both the Lily HBase Indexer host and the ZooKeeper hosts. Using FQDNs helps ensure proper Kerberos realm mapping.

For example:

```
hbase-indexer list-indexers --http http://lily01.example.com:11060/indexer/ \
--jaas $HOME/jaas.conf --zookeeper
zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181
```

Configuring Lily HBase Indexer Security

The Lily HBase indexer supports Kerberos for authentication, and Apache Sentry for authorization. For more information, see Configuring Lily HBase Indexer Security on page 94.

Configuring Lily HBase Indexer Security

> **Note:** This page contains references to CDH 5 components or features that have been removed from CDH 6. These references are only applicable if you are managing a CDH 5 cluster with Cloudera Manager 6. For more information, see Deprecated Items.

Beginning with CDH 5.4 the Lily HBase Indexer includes an HTTP interface for the `list-indexers`, `create-indexer`, `update-indexer`, and `delete-indexer` commands. This interface can be secured with Kerberos for authentication and Apache Sentry for authorization.

Configuring Lily HBase Indexer Service to Use Kerberos Authentication

To configure the Lily HBase Indexer to use Kerberos authentication, you must create principals and keytabs and then modify certain configuration properties. If you are using Cloudera Manager to manage your cluster, much of this is handled automatically. For unmanaged environments, you must generate the Kerberos principals and keytabs manually.

For more an overview of Kerberos concepts (including principals and keytabs), see Kerberos Security Artifacts Overview.

To enable Kerberos authentication for the Lily HBase Indexer service:

1. Go to **Key-Value Store Indexer service** > **Configuration** > **Category** > **Security**.
2. Select the **kerberos** option for **HBase Indexer Secure Authentication**.
3. Click **Save Changes**.
4. Go to **Administration** > **Security** > **Kerberos Credentials**.
5. Click **Generate Missing Credentials**.
6. Restart the indexer service (**Key-Value Store Indexer service** > **Actions** > **Restart**).

Configuring the Lily HBase Indexer Service to Use the Sentry Service

The Lily HBase Indexer service uses Apache Sentry for authorization. To use Sentry for authorization, you must use the indexer HTTP interface.

If you are using policy files for Sentry, and want to switch to the Sentry service, see Migrating HBase Indexer Sentry Policy Files to the Sentry Service on page 97.

To configure the Lily HBase Indexer to use the Sentry Service:

1. Go to **Key-Value Store Indexer service** > **Configuration** > **Category** > **Policy File Based Sentry**.
2. Make sure that the box labeled **Enable Sentry Authorization using Policy Files** is unchecked.
3. Click the **Main** category in the left pane.
4. Select the **Sentry Service** for the cluster (`SENTRY-1` by default).
5. Click **Save Changes**.
6. Restart stale services (**Key-Value Store Indexer service** > **Actions** > **Restart**).

Configuring the Lily HBase Indexer Service to Use Sentry Policy Files

The Lily HBase Indexer service uses Apache Sentry for authorization. To use Sentry for authorization, you must use the indexer HTTP interface.

Before CDH 5.14.0, Lily HBase Indexer supported only Sentry policy files. In CDH 5.14.0 and higher, it supports the Sentry service, and includes a command line utility (`hbase-indexer-sentry`) for configuring Sentry. Cloudera recommends using the Sentry Service. To migrate your policy files to the Sentry Service, see Migrating HBase Indexer Sentry Policy Files to the Sentry Service on page 97.

To configure Sentry policy files for the Lily HBase Indexer:

1. Go to **Key-Value Store Indexer service** > **Configuration** > **Category** > **Policy File Based Sentry**.
2. Check the box labeled **Enable Sentry Authorization using Policy Files**.
3. If necessary, edit **Sentry Global Policy File** to change the HDFS location of the `sentry-provider.ini` file.
4. Click **Save Changes**.
5. Restart the service (**Key-Value Store Indexer service** > **Actions** > **Restart**).
6. Upload the `sentry-provider.ini` file to the specified location in HDFS. For example:

   - **Security Enabled:**

```
kinit hdfs@EXAMPLE.COM
hdfs dfs -mkdir -p /user/hbaseindexer/sentry/
hdfs dfs -put /path/to/local/sentry-provider.ini /user/hbaseindexer/sentry/
hdfs dfs -chown -R hbase:hbase /user/hbaseindexer
```

   - **Security Disabled:**

```
sudo -u hdfs hdfs dfs -mkdir -p /user/hbaseindexer/sentry/
sudo -u hdfs hdfs dfs -put /path/to/local/sentry-provider.ini /user/hbaseindexer/sentry/
sudo -u hdfs hdfs dfs -chown -R hbase:hbase /user/hbaseindexer
```

Managing Sentry Permissions for the Lily HBase Indexer

The Lily HBase Indexer privilege model specifies READ and WRITE privileges for each indexer. The privileges work as follows:

- If a role has `WRITE` privilege for `indexer1`, that role can create, update, or delete an indexer named `indexer1`, using the `hbase-indexer` command.
- `READ` privileges control what a user can see when running the `hbase-indexer list-indexers` command. If a role has `READ` privileges for `indexer1`, the command output lists `indexer1` if it exists. If an indexer named `indexer2` exists, but the role does not have `READ` privileges for it, that indexer is omitted from the response.

For example, consider the following privileges defined in a policy file:

```
[groups]
jdoe = admin
psherman = readonly

[roles]
admin = indexer=*
readonly = indexer=*->action=READ
```

This policy file grants the `jdoe` user full access to all indexers, and the `psherman` user read access to all indexers. User `psherman` can see all indexers, but cannot create new ones or modify existing ones.

Important: The Sentry privileges for the Lily HBase Indexer service apply only to indexers, and not collections. You must separately grant the `hbase` user permission to update collections. For more information on configuring Sentry authorization for collections, see Configuring Sentry Authorization for Cloudera Search.

Before CDH 5.14.0, Lily HBase Indexer supported only Sentry policy files. In CDH 5.14.0 and higher, it supports the Sentry service, and includes a command line utility (`hbase-indexer-sentry`) for configuring Sentry. The command syntax is as follows:

```
/opt/cloudera/parcels/CDH/bin/hbase-indexer-sentry
Missing required option: [-lp List privilege, -rpr Revoke privilege from role, -lr List
 role, -arg Add role to group, -drg Delete role from group, -gpr Grant privilege to
role, -mgr Migrate ini file to Sentry service, -dr Drop role, -cr Create role]
usage: sentryShell
 -arg,--add_role_group          Add role to group
 -c,--checkcompat               Check compatibility with Sentry Service
 -conf,--sentry_conf <arg>      sentry-site file path
 -cr,--create_role              Create role
 -dr,--drop_role                Drop role
 -drg,--delete_role_group       Delete role from group
 -f,--policy_ini <arg>          Policy file path
 -g,--groupname <arg>           Group name
 -gpr,--grant_privilege_role    Grant privilege to role
 -h,--help                      Shell usage
 -i,--import                    Import policy file
 -lp,--list_privilege           List privilege
 -lr,--list_role                List role
 -mgr,--migrate                 Migrate ini file to Sentry service
 -p,--privilege <arg>           Privilege string
 -r,--rolename <arg>            Role name
 -rpr,--revoke_privilege_role   Revoke privilege from role
 -s,--service <arg>             Name of the service being managed
 -t,--type <arg>                [hive|solr|sqoop|.....]
 -v,--validate                  Validate policy file
```

Granting Privileges to a Role

The following is an example of how to add priviliges to the `test` role, which is part of the `testGroup`, for the `lilytestindexer`.

1. Authenticate as Sentry admin.
2. Create the `test` role:

```
hbase-indexer-sentry -s "KS_INDEXER-1" -cr -r test
```

If you have modified your service name from the default, replace *KS_INDEXER-1* with your custom service name.

3. Assign the role to the group `testGroup`:

```
hbase-indexer-sentry -s "KS_INDEXER-1" -arg -r test -g testGroup
```

4. Verify that the test role is part of the group `testGroup`:

```
hbase-indexer-sentry -s "KS_INDEXER-1" -lr -g testGroup
```

5. Grant priviliges to `test` role:

```
hbase-indexer-sentry -s "KS_INDEXER-1" -gpr -r test -p "indexer=lilytestindexer->action=*"
```

6. Revoke priviliges from `test` role:

```
hbase-indexer-sentry -s "KS_INDEXER-1" -rpr -r test -p "indexer=lilytestindexer->action=*"
```

Migrating HBase Indexer Sentry Policy Files to the Sentry Service

If you are using CDH 5.14.0 or higher, and want to use the Sentry Service, you can migrate your policy files using the `hbase-indexer-sentry` utility:

1. Make sure that you are running the Sentry Service in your cluster. If not, add the service, following the instructions in Adding a Service.
2. Run the following command:

```
/opt/cloudera/parcels/CDH/bin/hbase-indexer-sentry -s "KS_INDEXER-1" -mgr -i -v -f
"hdfs://<namenode>:8020/path/to/sentry-provider.ini"
```

   If you have modified your service name from the default, replace *KS_INDEXER-1* with your custom service name.

   This command validates and imports the specified policy file to the Sentry Service. For more information on the command usage and syntax, see Managing Sentry Permissions for the Lily HBase Indexer on page 95.

3. Configure the Lily HBase Indexer service to use the Sentry service:
   a. Uncheck the box labeled **Enable Sentry Authorization using Policy Files** (**Key-Value Store Indexer service** > **Configuration** > **Category** > **Policy File Based Sentry**)
   b. Configure the Sentry Service (**Key-Value Store Indexer service** > **Configuration** > **Category** > **Main** > **Sentry Service**).
4. Restart the Lily HBase Indexer service (**Key-Value Store Indexer service** > **Actions** > **Restart**).

Batch Indexing Using Cloudera Search

Batch indexing usually relies on MapReduce/YARN or Spark jobs to periodically index large datasets, or to index new datasets for the first time. The Lily HBase indexer, also called HBaseMapReduceIndexerTool, can be used for batch indexing HBase tables.

Spark Indexing

If you are using Apache Spark, you can batch index data using CrunchIndexerTool.

CrunchIndexerTool is a Spark or MapReduce ETL batch job that pipes data from HDFS files into Apache Solr through a morphline for extraction and transformation. The program is designed for flexible, scalable, fault-tolerant batch ETL pipeline jobs. It is implemented as an Apache Crunch pipeline, allowing it to run on MapReduce or Spark execution engines.

CrunchIndexerTool requires a working MapReduce or Spark cluster, such as one installed using Cloudera Manager. It is included in parcel-based installations.

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

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

  # ZooKeeper ensemble
  zkHost :
"zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181/solr"
}

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

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

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

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

You can see the usage syntax `CrunchIndexerTool` by running the job with the `-help` argument. Unlike other Search indexing tools, the `CrunchIndexerTool` jar does not contain all dependencies. If you try to run the job without addressing this, you get an error such as the following:

```
hadoop jar /opt/cloudera/parcels/CDH/lib/solr/contrib/crunch/search-crunch.jar
org.apache.solr.crunch.CrunchIndexerTool -help
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/crunch/types/PType

        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:348)
        at org.apache.hadoop.util.RunJar.run(RunJar.java:214)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:136)
Caused by: java.lang.ClassNotFoundException: org.apache.crunch.types.PType
        at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
        ... 4 more
```

To see the command usage (or to run the job), you must first add the dependencies to the classpath:

- For parcel-based installations:

```
export HADOOP_CLASSPATH="/opt/cloudera/parcels/CDH/lib/search/lib/search-crunch/*"
hadoop jar /opt/cloudera/parcels/CDH/lib/solr/contrib/crunch/search-crunch.jar
org.apache.solr.crunch.CrunchIndexerTool -help
```

- For package-based installations:

```
export HADOOP_CLASSPATH="/usr/lib/search/lib/search-crunch/*"
hadoop jar /usr/lib/solr/contrib/crunch/search-crunch.jar
org.apache.solr.crunch.CrunchIndexerTool -help
```

For reference, here is the command usage syntax:

Important: The command usage help incorrectly notes the following:

```
NOTE: MapReduce does not require extra steps for communicating with
kerberos-enabled Solr
```

To run the MapReduce job on a Kerberos-enabled cluster, you must create and specify a `jaas.conf` file, as described in Configuring a jaas.conf File.

For example:

```
HADOOP_OPTS="-Djava.security.auth.login.config=/path/to/jaas.conf" \
hadoop jar
/opt/cloudera/parcels/CDH/lib/solr/contrib/crunch/search-crunch.jar \
org.apache.solr.crunch.CrunchIndexerTool [...]
```

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

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

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

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

The program proceeds in several consecutive phases, as follows:

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

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

3) Morphline  phase:  This  (parallel)  phase  receives  the  items  of the
previous phase, and  uses  a  Morphline  to  extract  the relevant content,
transform  it  and  load  zero  or  more  documents  into  Solr.  The  ETL
functionality is  flexible  and  customizable  using  chains  of  arbitrary
morphline commands that pipe  records  from  one  transformation command to
another. Commands to parse and  transform  a  set  of standard data formats
```

```
such as Avro, Parquet, CSV, Text, HTML, XML, PDF, MS-Office, etc. are
provided out of the box, and additional custom commands and parsers for
additional file or data formats can be added as custom morphline commands.
Any kind of data format can be processed and any kind output format can be
generated by any custom Morphline ETL logic. Also, this phase can be used
to send data directly to a live SolrCloud cluster (via the loadSolr
morphline command).

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

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

Comparison with MapReduceIndexerTool:

1) CrunchIndexerTool can also run on the Spark execution engine, not just
on MapReduce.
2) CrunchIndexerTool enables interactive low latency prototyping, in
particular in Spark 'local' mode.
3) CrunchIndexerTool supports updates (and deletes) of existing documents
in Solr, not just inserts.
4) CrunchIndexerTool can exploit data locality for splittable Hadoop files
(text, avro, avroParquet).
We recommend MapReduceIndexerTool for large scale batch ingestion use
cases where updates (or deletes) of existing documents in Solr are not
required, and we recommend CrunchIndexerTool for all other use cases.

CrunchIndexerOptions:
  HDFS_URI              HDFS URI of file or directory tree to ingest.
                        (default: [])
  --input-file-list URI, --input-list URI
                        Local URI or HDFS URI of a UTF-8 encoded file
                        containing a list of HDFS URIs to ingest, one URI
                        per line in the file. If '-' is specified, URIs
                        are read from the standard input. Multiple --
                        input-file-list arguments can be specified.
  --input-file-format FQCN
                        The Hadoop FileInputFormat to use for extracting
                        data from splittable HDFS files. Can be a fully
                        qualified Java class name or one of ['text',
                        'avro', 'avroParquet']. If this option is present
                        the extraction phase will emit a series of input
                        data records rather than a series of HDFS file
                        input streams.
  --input-file-projection-schema FILE
                        Relative or absolute path to an Avro schema file
                        on the local file system. This will be used as
                        the projection schema for Parquet input files.
  --input-file-reader-schema FILE
                        Relative or absolute path to an Avro schema file
                        on the local file system. This will be used as
                        the reader schema for Avro or Parquet input
                        files.      Example:      src/test/resources/test-
                        documents/strings.avsc
  --morphline-file FILE Relative or absolute path to a local config file
                        that contains one or more morphlines. The file
                        must be UTF-8 encoded. It will be uploaded to
                        each remote task. Example: /path/to/morphline.conf
  --morphline-id STRING The identifier of the morphline that shall be
                        executed within the morphline config file
                        specified by --morphline-file. If the --morphline-
                        id option is omitted the first (i.e. top-most)
                        morphline within the config file is used.
                        Example: morphline1
  --pipeline-type STRING
                        The engine to use for executing the job. Can be
                        'mapreduce' or 'spark'. (default: mapreduce)
  --xhelp, --help, -help
                        Show this help message and exit
```

```
  --mappers INTEGER      Tuning knob that indicates  the  maximum number of
                         MR mapper tasks to use.  -1  indicates use all map
                         slots available  on  the  cluster.  This parameter
                         only  applies   to   non-splittable   input  files
                         (default: -1)
  --parallel-morphline-inits INTEGER
                         Tuning knob that indicates  the  maximum number of
                         morphline instances  to  initialize  at  the  same
                         time. This kind of rate  limiting on rampup can be
                         useful  to  avoid  overload   conditions  such  as
                         ZooKeeper connection limits  or  DNS lookup limits
                         when using  many  parallel  mapper  tasks  because
                         each  such   task   contains   one   morphline.  1
                         indicates initialize  each  morphline  separately.
                         This feature  is  implemented  with  a distributed
                         semaphore. The default is to  use no rate limiting
                         (default: 2147483647)
  --dry-run              Run the pipeline  but  print  documents  to stdout
                         instead of loading  them  into  Solr.  This can be
                         used for quicker turnaround  during  early trial &
                         debug sessions. (default: false)
  --log4j FILE           Relative or absolute  path  to  a log4j.properties
                         config file on the  local  file  system. This file
                         will be uploaded  to  each  remote  task. Example:
                         /path/to/log4j.properties
  --chatty               Turn on verbose output. (default: false)

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

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

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


Examples:

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

# Prepare variables for convenient reuse:
export myDriverJarDir=target # for build from git
export myDriverJarDir=/opt/cloudera/parcels/CDH/lib/solr/contrib/crunch # for CDH with
 parcels
export myDriverJarDir=/usr/lib/solr/contrib/crunch # for CDH with packages
export myDependencyJarDir=target/lib # for build from git
export myDependencyJarDir=/opt/cloudera/parcels/CDH/lib/search/lib/search-crunch # for
 CDH with parcels
export myDependencyJarDir=/usr/lib/search/lib/search-crunch # for CDH with packages
export myDriverJar=$(find $myDriverJarDir -maxdepth 1 -name 'search-crunch-*.jar' !
-name '*-job.jar' ! -name '*-sources.jar')
export myDependencyJarFiles=$(find $myDependencyJarDir -name '*.jar' | sort | tr '\n'
```

```
',' | head -c -1)
export myDependencyJarPaths=$(find $myDependencyJarDir -name '*.jar' | sort | tr '\n'
':' | head -c -1)
export myJVMOptions="-DmaxConnectionsPerHost=10000 -DmaxConnections=10000
-Djava.io.tmpdir=/my/tmp/dir/" # connection settings for solrj, also custom tmp dir

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

# Spark in Local Mode (for rapid prototyping) - Ingest into Solr:
spark-submit \
  --master local \
  --deploy-mode client \
  --jars $myDependencyJarFiles \
  --executor-memory 500M \
  --conf "spark.executor.extraJavaOptions=$myJVMOptions" \
  --driver-java-options "$myJVMOptions" \
  # --driver-library-path /opt/cloudera/parcels/CDH/lib/hadoop/lib/native # for Snappy
 on CDH with parcels\
  # --driver-library-path /usr/lib/hadoop/lib/native # for Snappy on CDH with packages
 \
  --class org.apache.solr.crunch.CrunchIndexerTool \
  $myDriverJar \
  -D morphlineVariable.ZK_HOST=$(hostname):2181/solr \
  --morphline-file $myResourcesDir/test-morphlines/loadSolrLine.conf \
  --pipeline-type spark \
  --chatty \
  --log4j $myResourcesDir/log4j.properties \
  /user/systest/input/hello1.txt

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

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

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

# Spark on Yarn in Cluster Mode (for production) - Ingest into Secure (Kerberos-enabled)
 Solr:
# Spark requires two additional steps compared to non-secure solr:
# (NOTE: MapReduce does not require extra steps for communicating with kerberos-enabled
 Solr)
```

```
# 1) Create a delegation token file
#    a) kinit as the user who will make solr requests
#    b) request a delegation token from solr and save it to a file:
#       e.g. using curl:
#       "curl --negotiate -u: http://solr-host:port/solr/?op=GETDELEGATIONTOKEN >
tokenFile.txt"
# 2) Pass the delegation token file to spark-submit:
#    a) add the delegation token file via --files
#    b) pass the file name via -D tokenFile
#       spark places this file in the cwd of the executor, so only list the file name,
 no path
spark-submit \
  --master yarn \
  --deploy-mode cluster \
  --jars $myDependencyJarFiles \
  --executor-memory 500M \
  --conf "spark.executor.extraJavaOptions=$myJVMOptions" \
  --driver-java-options "$myJVMOptions" \
  --class org.apache.solr.crunch.CrunchIndexerTool \
  --files $(ls $myResourcesDir/log4j.properties),$(ls
$myResourcesDir/test-morphlines/loadSolrLine.conf),tokenFile.txt\
  $myDriverJar \
  -D hadoop.tmp.dir=/tmp \
  -D morphlineVariable.ZK_HOST=$(hostname):2181/solr \
  -DtokenFile=tokenFile.txt \
  --morphline-file loadSolrLine.conf \
  --pipeline-type spark \
  --chatty \
  --log4j log4j.properties \
  /user/systest/input/hello1.txt
```

MapReduce Indexing

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

For information on tools related to batch indexing, continue reading:

Running an Example Indexing Job

For examples of running a MapReduce job to index documents, see Cloudera Search Tutorial on page 16

MapReduceIndexerTool

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

For more information on Morphlines, see:

- Extracting, Transforming, and Loading Data With Cloudera Morphlines on page 74 for an introduction to Morphlines.
- Example Morphline Usage on page 77 for morphline examples, discussion of those examples, and links to additional information.

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

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

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

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

Note: Using `--libjars` parameter in `dry-run` mode does not work. Instead, specify the JAR files using the HADOOP_CLASSPATH environmental variable.

MapReduceIndexerTool Input Splits

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

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

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

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

MapReduceIndexerTool Metadata

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

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

Example output:

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

```
"file_permissions_other":"r--",
"file_permissions_stickybit":false,
```

MapReduceIndexerTool Usage Syntax

Important: In CDH 6, you must run the indexer tool with the following command-line argument:

```
-D 'mapreduce.job.user.classpath.first=true'
```

Running the tool without this argument triggers the following error:

```
ERROR [main] org.apache.hadoop.mapred.YarnChild: Error running child :
 java.lang.NoSuchMethodError:
com.codahale.metrics.MetricRegistry.meter(Ljava/lang/String;Lcom/codahale/metrics/MetricSupplier;)Lcom/codahale/metrics/Meter;
```

- To view the usage syntax in a default parcel-based deployment, run:

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

- To view the usage syntax in a default package-based deployment, use:

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

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

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

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

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

3) Reducer phase: This (parallel) phase loads the mapper's
```

```
SolrInputDocuments into one EmbeddedSolrServer per reducer. Each such
reducer and Solr server can be seen as a (micro) shard. The Solr servers
store their data in HDFS.

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

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

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

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

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

```
                        to index into.  0  is  reserved  for a mapper-only
                        feature that may  ship  in  a  future  release. -1
                        indicates use all  reduce  slots  available on the
                        cluster. -2 indicates use  one  reducer per output
                        shard,  which   disables   the   mtree   merge  MR
                        algorithm. The mtree  merge  MR algorithm improves
                        scalability by spreading  load  (in particular CPU
                        load) among a  number  of  parallel  reducers that
                        can be much larger than  the number of solr shards
                        expected by  the  user.  It  can  be  seen  as  an
                        extension of concurrent  lucene  merges and tiered
                        lucene  merges   to   the   clustered   case.  The
                        subsequent mapper-only phase merges  the output of
                        said large number  of  reducers  to  the number of
                        shards expected by  the  user,  again by utilizing
                        more  available   parallelism   on   the  cluster.
                        (default: -1)
  --max-segments INTEGER
                        Tuning knob that indicates  the  maximum number of
                        segments to be contained  on  output  in the index
                        of each reducer shard.  After  a reducer has built
                        its output index  it  applies  a  merge  policy to
                        merge segments  until  there  are  <=  maxSegments
                        lucene  segments  left  in   this  index.  Merging
                        segments involves reading  and  rewriting all data
                        in all these  segment  files, potentially multiple
                        times,  which  is  very  I/O  intensive  and  time
                        consuming. However, an  index  with fewer segments
                        can later be merged  faster,  and  it can later be
                        queried  faster  once  deployed  to  a  live  Solr
                        serving shard. Set  maxSegments  to  1 to optimize
                        the index for low query  latency. In a nutshell, a
                        small maxSegments  value  trades  indexing latency
                        for subsequently improved query  latency. This can
                        be  a  reasonable  trade-off  for  batch  indexing
                        systems. (default: 1)
  --dry-run             Run in local mode  and  print  documents to stdout
                        instead of loading them  into  Solr. This executes
                        the  morphline  in  the  client  process  (without
                        submitting a job  to  MR)  for  quicker turnaround
                        during early  trial  &  debug  sessions. (default:
                        false)
  --log4j FILE          Relative or absolute  path  to  a log4j.properties
                        config file on the  local  file  system. This file
                        will  be  uploaded  to   each  MR  task.  Example:
                        /path/to/log4j.properties
  --verbose, -v         Turn on verbose output. (default: false)
  --show-non-solr-cloud Also show options for  Non-SolrCloud  mode as part
                        of --help. (default: false)

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

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

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

                        The   --zk-host   option   implements   the   same
                        partitioning semantics as  the  standard SolrCloud
```

```
                        Near-Real-Time (NRT) API. This enables to mix
                        batch updates from MapReduce ingestion with
                        updates from standard Solr NRT ingestion on the
                        same SolrCloud cluster, using identical unique
                        document keys.

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

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

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

  --go-live             Allows you to optionally merge the final index
                        shards into a live Solr cluster after they are
                        built. You can pass the ZooKeeper address with --
                        zk-host and the relevant cluster information will
                        be auto detected. (default: false)
  --collection STRING   The SolrCloud collection to merge shards into
                        when using --go-live and --zk-host. Example:
                        collection1
  --go-live-min-replication-factor INTEGER
                        The minimum number of SolrCloud replicas to
                        successfully merge any final index shard into.
                        The go-live job phase attempts to merge final
                        index shards into all SolrCloud replicas. Some of
                        these merge operations may fail, for example if
                        some SolrCloud servers are down. This option
                        enables indexing jobs to succeed even if some
                        such merge operations fail on SolrCloud
                        followers. Successful merge operations into all
                        leaders are always required for job success,
                        regardless of the value of --go-live-min-
                        replication-factor. -1 indicates require
                        successful merge operations into all replicas. 1
                        indicates require successful merge operations
                        only into leader replicas. (default: -1)
  --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>   define a value for a given property
-fs <file:///|hdfs://namenode:port> specify default filesystem URL to use, overrides
'fs.defaultFS' property from configurations.
  --jt <local|resourcemanager:port>
                        specify a ResourceManager
  --files <file1,...>   specify a comma-separated list of files to be
                        copied to the map reduce cluster
  --libjars <jar1,...>  specify a comma-separated list of jar files to be
                        included in the classpath
  --archives <archive1,...>
                        specify a comma-separated list of archives to be
                        unarchived on the compute machines

The general command line syntax is:
command [genericOptions] [commandOptions]
```

```
Examples:

# (Re)index an Avro based Twitter tweet file:
sudo -u hdfs hadoop \
  --config /etc/hadoop/conf.cloudera.mapreduce1 \
  jar target/search-mr-*-job.jar org.apache.solr.hadoop.MapReduceIndexerTool \
  -D 'mapred.child.java.opts=-Xmx500m' \
  --log4j src/test/resources/log4j.properties \
  --morphline-file
../search-core/src/test/resources/test-morphlines/tutorialReadAvroContainer.conf \
  --solr-home-dir src/test/resources/solr/minimr \
  --output-dir hdfs://c2202.mycompany.com/user/$USER/test \
  --shards 1 \
  hdfs:///user/$USER/test-documents/sample-statuses-20120906-141433.avro

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

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

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

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

Lily HBase Batch Indexing for Cloudera Search

With Cloudera Search, you can batch index HBase tables using the Lily HBase batch indexer MapReduce job, also known as HBaseMapReduceIndexerTool. This batch indexing *does not* require:

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

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

The following procedures demonstrate creating a small HBase table and using the HBaseMapReduceIndexerTool to index the table into a collection:

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

Populating an HBase Table

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

```
hbase shell

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

Creating a Collection in Cloudera Search

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

```
solrctl instancedir --generate $HOME/hbase_collection_config
## Edit $HOME/hbase_collection_config/conf/schema.xml as needed ##
## If you are using Sentry for authorization, copy solrconfig.xml.secure to solrconfig.xml
 as follows: ##
## cp $HOME/hbase_collection_config/conf/solrconfig.xml.secure
$HOME/hbase_collection_config/conf/solrconfig.xml ##
solrctl instancedir --create hbase_collection_config $HOME/hbase_collection_config
solrctl collection --create hbase_collection -s <numShards> -c hbase_collection_config
```

Creating a Lily HBase Indexer Configuration File

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

An indexer configuration XML file must refer to the `MorphlineResultToSolrMapper` implementation and point to the location of a Morphline configuration file, as shown in the following `morphline-hbase-mapper.xml` indexer configuration file. For Cloudera Manager managed environments, set `morphlineFile` to the relative path `morphlines.conf`. For unmanaged environments, specify the absolute path to a `morphlines.conf` file that exists on the Lily HBase Indexer host. Make sure the file is readable by the HBase system user (`hbase` by default).

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

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

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

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

</indexer>
```

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

Creating a Morphline Configuration File

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

If you are using Cloudera Manager, the `morphlines.conf` file is edited within Cloudera Manager (**Key-Value Store Indexer service** > **Configuration** > **Category** > **Morphlines** > **Morphlines File**).

For unmanaged environments, create the `morphlines.conf` file on the Lily HBase Indexer host:

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

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

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

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

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

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

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

Understanding the `extractHBaseCells` Morphline Command

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

Each mapping has:

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

```
inputColumn : "mycolumnfamily:*"
outputField : "belongs_to_*"
```

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

```
put 'table_name' , 'row1' , 'mycolumnfamily:1' , 'foo'
put 'table_name' , 'row1' , 'mycolumnfamily:9' , 'bar'
```

Then the fields of the Solr document are as follows:

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

- The `type` parameter defines the data type of the content in HBase. All input data is stored in HBase as byte arrays, but all content in Solr is indexed as text, so a method for converting byte arrays to the actual data type is required. The type parameter can be the name of a type that is supported by

`org.apache.hadoop.hbase.util.Bytes.to*` (which currently includes `byte[]`, `int`, `long`, `string`, `boolean`, `float`, `double`, `short`, and `bigdecimal`). Use type `byte[]` to pass the byte array through to the morphline without conversion.

- `type:byte[]` copies the byte array unmodified into the record output field
- `type:int` converts with `org.apache.hadoop.hbase.util.Bytes.toInt`
- `type:long` converts with `org.apache.hadoop.hbase.util.Bytes.toLong`
- `type:string` converts with `org.apache.hadoop.hbase.util.Bytes.toString`
- `type:boolean` converts with `org.apache.hadoop.hbase.util.Bytes.toBoolean`
- `type:float` converts with `org.apache.hadoop.hbase.util.Bytes.toFloat`
- `type:double` converts with `org.apache.hadoop.hbase.util.Bytes.toDouble`
- `type:short` converts with `org.apache.hadoop.hbase.util.Bytes.toShort`
- `type:bigdecimal` converts with `org.apache.hadoop.hbase.util.Bytes.toBigDecimal`

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

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

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

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

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

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

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

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

Running HBaseMapReduceIndexerTool

Important: In CDH 6, you must run the indexer tool with the following command-line argument:

```
-D 'mapreduce.job.user.classpath.first=true'
```

Running the tool without this argument triggers the following error:

```
ERROR [main] org.apache.hadoop.mapred.YarnChild: Error running child :
 java.lang.NoSuchMethodError:
com.codahale.metrics.MetricRegistry.register(Ljava/lang/String;Lcom/codahale/metrics/MetricRegistry$MetricSupplier;)Lcom/codahale/metrics/Met
```

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

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

Run the command as follows:

- **For package-based deployments:**

```
hadoop --config /etc/hadoop/conf \
jar /usr/lib/hbase-solr/tools/hbase-indexer-mr-*-job.jar \
--conf /etc/hbase/conf/hbase-site.xml -Dmapreduce.map.java.opts="-Xmx512m" \
-Dmapreduce.reduce.java.opts="-Xmx512m" \
--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
```

- **For parcel-based deployments:**

```
hadoop --config /etc/hadoop/conf \
jar /opt/cloudera/parcels/CDH/lib/hbase-solr/tools/hbase-indexer-mr-*-job.jar \
--conf /etc/hbase/conf/hbase-site.xml -Dmapreduce.map.java.opts="-Xmx512m" \
-Dmapreduce.reduce.java.opts="-Xmx512m" \
--hbase-indexer-file $HOME/morphline-hbase-mapper.xml \
--zk-host 127.0.0.1/solr --collection hbase-collection1 \
--go-live --log4j src/test/resources/log4j.properties
```

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

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

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

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

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

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

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

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

The full command line usage help is as follows:

```
usage: hadoop [GenericOptions]... jar hbase-indexer-mr-*-job.jar
       [--hbase-indexer-zk STRING] [--hbase-indexer-name STRING]
       [--hbase-indexer-file FILE]
       [--hbase-indexer-component-factory STRING]
       [--hbase-table-name STRING] [--hbase-start-row BINARYSTRING]
       [--hbase-end-row BINARYSTRING] [--hbase-start-time STRING]
       [--hbase-end-time STRING] [--hbase-timestamp-format STRING]
       [--zk-host STRING] [--go-live] [--collection STRING]
       [--go-live-min-replication-factor INTEGER]
       [--go-live-threads INTEGER] [--help] [--output-dir HDFS_URI]
       [--overwrite-output-dir] [--morphline-file FILE]
       [--morphline-id STRING] [--solr-home-dir DIR]
```

```
       [--update-conflict-resolver FQCN] [--reducers INTEGER]
       [--max-segments INTEGER] [--fair-scheduler-pool STRING] [--dry-run]
       [--log4j FILE] [--verbose] [--clear-index] [--show-non-solr-cloud]

MapReduce batch job driver that takes input data from an HBase table and
creates Solr index shards and writes the indexes into HDFS, in a flexible,
scalable, and fault-tolerant manner. It also supports merging the output
shards into a set of live customer-facing Solr servers in SolrCloud.
Optionally, documents can be sent directly from the mapper tasks to
SolrCloud, which is a much less scalable approach but enables updating
existing documents in SolrCloud. The program proceeds in one or multiple
consecutive MapReduce-based phases, as follows:

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

2) Reducer phase: This (parallel) phase loads the mapper's
SolrInputDocuments into one EmbeddedSolrClient 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-indexer-component-factory STRING
                         Classname of the hbase indexer component factory.

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

  --hbase-table-name STRING
                         Optional name of the HBase table containing the
                         records to be indexed. If supplied, this
                         overrides the value from the --hbase-indexer-*
                         options. Example: myTable
```

```
  --hbase-start-row BINARYSTRING
                        Binary string representation of start row from
                        which to start indexing (inclusive). The format
                        of the supplied row key should use two-digit hex
                        values prefixed by \x for non-ascii characters (e.
                        g. 'row\x00'). The semantics of this argument are
                        the same as those for the HBase Scan#setStartRow
                        method. The default is to include the first row
                        of the table. Example: AAAA
  --hbase-end-row BINARYSTRING
                        Binary string representation of end row prefix at
                        which to stop indexing (exclusive). See the
                        description of --hbase-start-row for more
                        information. The default is to include the last
                        row of the table. Example: CCCC
  --hbase-start-time STRING
                        Earliest timestamp (inclusive) in time range of
                        HBase cells to be included for indexing. The
                        default is to include all cells. Example: 0
  --hbase-end-time STRING
                        Latest timestamp (exclusive) of HBase cells to be
                        included for indexing. The default is to include
                        all cells. Example: 123456789
  --hbase-timestamp-format STRING
                        Timestamp format to be used to interpret --hbase-
                        start-time and --hbase-end-time. This is a java.
                        text.SimpleDateFormat compliant format (see http:
                        //docs.oracle.
                        com/javase/8/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).

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

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

```
  --go-live              Allows you to optionally merge the final index
                         shards into a live Solr cluster after they are
                         built. You can pass the ZooKeeper address with --
                         zk-host and the relevant cluster information will
                         be auto detected. (default: false)
  --collection STRING    The SolrCloud collection to merge shards into
                         when using --go-live and --zk-host. Example:
                         collection1
  --go-live-min-replication-factor INTEGER
                         The minimum number of SolrCloud replicas to
                         successfully merge any final index shard into.
                         The go-live job phase attempts to merge final
                         index shards into all SolrCloud replicas. Some of
                         these merge operations may fail, for example if
                         some SolrCloud servers are down. This option
                         enables indexing jobs to succeed even if some
                         such merge operations fail on SolrCloud
                         followers. Successful merge operations into all
                         leaders are always required for job success,
                         regardless of the value of --go-live-min-
                         replication-factor. -1 indicates require
                         successful merge operations into all replicas. 1
                         indicates require successful merge operations
                         only into leader replicas. (default: -1)
  --go-live-threads INTEGER
                         Tuning knob that indicates the maximum number of
                         live merges to run in parallel at one time.
                         (default: 1000)

Optional arguments:
  --help, -help, -h      Show this help message and exit
  --output-dir HDFS_URI  HDFS directory to write Solr indexes to. Inside
                         there one output directory per shard will be
                         generated. Example: hdfs://c2202.mycompany.
                         com/user/$USER/test
  --overwrite-output-dir
                         Overwrite the directory specified by --output-dir
                         if it already exists. Using this parameter will
                         result in the output directory being recursively
                         deleted at job startup. (default: false)
  --morphline-file FILE  Relative or absolute path to a local config file
                         that contains one or more morphlines. The file
                         must be UTF-8 encoded. The file will be uploaded
                         to each MR task. If supplied, this overrides the
                         value from the --hbase-indexer-* options.
                         Example: /path/to/morphlines.conf
  --morphline-id STRING  The identifier of the morphline that shall be
                         executed within the morphline config file, e.g.
                         specified by --morphline-file. If the --morphline-
                         id option is ommitted 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
  --solr-home-dir DIR    Optional relative or absolute path to a local dir
                         containing Solr conf/ dir and in particular
                         conf/solrconfig.xml and optionally also lib/ dir.
                         This directory will be uploaded to each MR task.
                         Example: src/test/resources/solr/minimr
  --update-conflict-resolver FQCN
                         Fully qualified class name of a Java class that
                         implements the UpdateConflictResolver interface.
                         This enables deduplication and ordering of a
                         series of document updates for the same unique
                         document key. For example, a MapReduce batch job
                         might index multiple files in the same job where
                         some of the files contain old and new versions of
                         the very same document, using the same unique
                         document key.
                         Typically, implementations of this interface
                         forbid collisions by throwing an exception, or
                         ignore all but the most recent document version,
                         or, in the general case, order colliding updates
                         ascending from least recent to most recent
```

```
                        (partial) update. The caller of this interface (i.
                        e. the Hadoop Reducer) will then apply the
                        updates to Solr in the order returned by the
                        orderUpdates() method.
                        The                                        default
                        RetainMostRecentUpdateConflictResolver
                        implementation ignores all but the most recent
                        document version, based on a configurable numeric
                        Solr field, which defaults to the
                        file_last_modified timestamp (default: org.apache.
                        solr.hadoop.dedup.
                        RetainMostRecentUpdateConflictResolver)
  --reducers INTEGER    Tuning knob that indicates the number of reducers
                        to index into. 0 indicates that no reducers
                        should be used, and documents should be sent
                        directly from the mapper tasks to live Solr
                        servers. -1 indicates use all reduce slots
                        available on the cluster. -2 indicates use one
                        reducer per output shard, which disables the
                        mtree merge MR algorithm. The mtree merge MR
                        algorithm improves scalability by spreading load
                        (in particular CPU load) among a number of
                        parallel reducers that can be much larger than
                        the number of solr shards expected by the user.
                        It can be seen as an extension of concurrent
                        lucene merges and tiered lucene merges to the
                        clustered case. The subsequent mapper-only phase
                        merges the output of said large number of
                        reducers to the number of shards expected by the
                        user, again by utilizing more available
                        parallelism on the cluster. (default: -1)
  --max-segments INTEGER
                        Tuning knob that indicates the maximum number of
                        segments to be contained on output in the index
                        of each reducer shard. After a reducer has built
                        its output index it applies a merge policy to
                        merge segments until there are <= maxSegments
                        lucene segments left in this index. Merging
                        segments involves reading and rewriting all data
                        in all these segment files, potentially multiple
                        times, which is very I/O intensive and time
                        consuming. However, an index with fewer segments
                        can later be merged faster, and it can later be
                        queried faster once deployed to a live Solr
                        serving shard. Set maxSegments to 1 to optimize
                        the index for low query latency. In a nutshell, a
                        small maxSegments value trades indexing latency
                        for subsequently improved query latency. This can
                        be a reasonable trade-off for batch indexing
                        systems. (default: 1)
  --dry-run             Run in local mode and print documents to stdout
                        instead of loading them into Solr. This executes
                        the morphline in the client process (without
                        submitting a job to MR) for quicker turnaround
                        during early trial & debug sessions. (default:
                        false)
  --log4j FILE          Relative or absolute path to a log4j.properties
                        config file on the local file system. This file
                        will be uploaded to each MR task. Example:
                        /path/to/log4j.properties
  --verbose, -v         Turn on verbose output. (default: false)
  --clear-index         Will attempt to delete all entries in a solr
                        index before starting batch build. This is not
                        transactional so if the build fails the index
                        will be empty. (default: false)
  --show-non-solr-cloud Also show options for Non-SolrCloud mode as part
                        of --help. (default: false)

Generic options supported are:
  --conf <configuration file>
                        specify an application configuration file
  -D <property=value>   define a value for a given property
-fs <file:///|hdfs://namenode:port> specify default filesystem URL to use, overrides
```

```
'fs.defaultFS' property from configurations.
  --jt <local|resourcemanager:port>
                          specify a ResourceManager
  --files <file1,...>     specify a  comma-separated  list  of  files  to be
                          copied to the map reduce cluster
  --libjars <jar1,...>    specify a comma-separated list of  jar files to be
                          included in the classpath
  --archives <archive1,...>
                          specify a comma-separated list  of  archives to be
                          unarchived on the compute machines

The general command line syntax is:
command [genericOptions] [commandOptions]

Examples:

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

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

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

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

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

```
  --log4j src/test/resources/log4j.properties

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

Using `--go-live` with SSL or Kerberos

The go-live phase of the indexer jobs sends a `MERGEINDEXES` request from the indexer client (the node from which the MR job was submitted) to the live Solr servers. If the Solr server has SSL enabled, you need to ensure that the indexer client trusts the certificate presented by the Solr server(s), otherwise you get an `SSLPeerUnverifiedException`.

1. Specify the location of the trust store by setting the following `HADOOP_OPTS` variable before launching the indexer job:

```
HADOOP_OPTS="-Djavax.net.ssl.trustStore=/etc/cdep-ssl-conf/CA_STANDARD/truststore.jks
"
```

2. If the Solr servers have Kerberos authentication enabled, you need to ensure that the indexer client can authenticate via Kerberos to the Solr servers. For this, you need to create a Java Authentication and Authorization Service configuration (JAAS) file locally on the node where the indexing job is launched:
   - If you are authenticating using `kinit` to obtain credentials, you can configure the client to use your credential cache by creating a `jaas.conf` file with the following contents:

```
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=false
 useTicketCache=true
 principal="<user>@EXAMPLE.COM";
 };
```

     Replace *<user>* with your username, and *EXAMPLE.COM* with your Kerberos realm.
   - If you want the client application to authenticate using a keytab, modify `jaas-client.conf` as follows:

```
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 keyTab="/path/to/user.keytab"
 storeKey=true
 useTicketCache=false
 principal="<user>@EXAMPLE.COM";
};
```

     Replace */path/to/user.keytab* with the keytab file you want to use and *<user>@EXAMPLE.COM* with the principal in the keytab. If you are using a service principal that includes the hostname, make sure that it is included in the `jaas.conf` file (for example, `solr/solr01.example.com@EXAMPLE.COM`).

3. If you are using a ticket cache, you need to do a `kinit` to acquire a ticket for the configured principal before launching the indexer.

4. Specify the authentication configuration in the `HADOOP_OPTS` environment variable:

- **For package-based installations:**

```
HADOOP_OPTS="-Djava.security.auth.login.config=jaas.conf
-Djavax.net.ssl.trustStore=/etc/cdep-ssl-conf/CA_STANDARD/truststore.jks" \
hadoop --config /etc/hadoop/conf \
jar /usr/lib/hbase-solr/tools/hbase-indexer-mr-*-job.jar \
--conf /etc/hbase/conf/hbase-site.xml -Dmapreduce.map.java.opts="-Xmx512m"
-Dmapreduce.reduce.java.opts="-Xmx512m" \
--hbase-indexer-file /home/systest/hbasetest/morphline-hbase-mapper.xml \
--zk-host 127.0.0.1/solr \
--collection hbase-collection1 \
--go-live --log4j src/test/resources/log4j.properties
```

- **For parcel-based installations:**

```
HADOOP_OPTS="-Djava.security.auth.login.config=jaas.conf
-Djavax.net.ssl.trustStore=/etc/cdep-ssl-conf/CA_STANDARD/truststore.jks" \
hadoop --config /etc/hadoop/conf \
jar /opt/cloudera/parcels/CDH/lib/hbase-solr/tools/hbase-indexer-mr-*-job.jar \
--conf /etc/hbase/conf/hbase-site.xml -Dmapreduce.map.java.opts="-Xmx512m"
-Dmapreduce.reduce.java.opts="-Xmx512m" \
--hbase-indexer-file /home/systest/hbasetest/morphline-hbase-mapper.xml \
--zk-host 127.0.0.1/solr \
--collection hbase-collection1 \
--go-live --log4j src/test/resources/log4j.properties
```

Note:

Communication to Solr servers is only occurring in the go-live phase, not from the MapReduce jobs. Therefore it is enough to place the `jaas.conf` and the SSL trust store on the node from which the indexer client is started as it will be the one which communicates to Solr.

Understanding `--go-live` and HDFS ACLs

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

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

Cloudera Search Frequently Asked Questions

This section includes answers to questions commonly asked about Search for CDH. Questions are divided into the following categories:

General

The following are general questions about Cloudera Search and the answers to those questions.

What is Cloudera Search?

Cloudera Search is Apache Solr integrated with CDH, including Apache Lucene, Apache SolrCloud, Apache Flume, Apache Tika, and Apache Hadoop MapReduce and HDFS. Cloudera Search also includes valuable integrations that make searching more scalable, easy to use, and optimized for both near-real-time and batch-oriented indexing. These integrations include Cloudera Morphlines, a customizable transformation chain that simplifies loading any type of data into Cloudera Search.

What is the difference between Lucene and Solr?

Lucene is a low-level search library that is accessed by a Java API. Solr is a search server that runs in a servlet container and provides structure and convenience around the underlying Lucene library.

What is Apache Tika?

The Apache Tika toolkit detects and extracts metadata and structured text content from various documents using existing parser libraries.

How does Cloudera Search relate to web search?

Traditional web search engines crawl web pages on the Internet for content to index. Cloudera Search indexes files and data that are stored in HDFS and HBase. To make web data available through Cloudera Search, it needs to be downloaded and stored in Cloudera Enterprise.

How does Cloudera Search relate to enterprise search?

Enterprise search connects with different backends (such as RDBMS and filesystems) and indexes data in all those systems. Cloudera Search is intended as a full-text search capability for data in CDH. Cloudera Search is a tool added to the Cloudera data processing platform and does not aim to be a stand-alone search solution, but rather a user-friendly interface to explore data in Hadoop and HBase.

How does Cloudera Search relate to custom search applications?

Custom and specialized search applications are an excellent complement to the Cloudera data-processing platform. Cloudera Search is not designed to be a custom application for niche vertical markets. However, Cloudera Search does include a simple search GUI through a plug-in application for Hue. The Hue plug-in application is based on the Solr API and allows for easy exploration, along with all of the other Hadoop frontend applications in Hue.

Which Solr Server should I send my queries to?

Any Solr Server can accept and process client connections.

Do Search security features use Kerberos?

Yes, Cloudera Search includes support for Kerberos authentication. Search continues to use simple authentication with the anonymous user as the default configuration, but Search now supports changing the authentication scheme to

Kerberos. All required packages are installed during the installation or upgrade process. Additional configuration is required before Kerberos is available in your environment.

Can I restrict access to collections?

Yes, Cloudera Search supports Apache Sentry for authorization. For more information, see Configuring Sentry Authorization for Cloudera Search.

Do I need to configure Sentry restrictions for each access mode, such as for the admin console and for the command line?

Sentry restrictions are consistently applied regardless of the way users attempt to complete actions. For example, restricting access to data in a collection consistently restricts that access, whether queries come from the command line, from a browser, or through the admin console.

Does Search support indexing data stored in JSON files and objects?

Yes, you can use the `readJson` and `extractJsonPaths` morphline commands that are included with the CDK to access JSON data and files. For more information, see cdk-morphlines-json.

How can I set up Cloudera Search so that results include links back to the source that contains the result?

You can use stored results fields to create links back to source documents. For information on data types, including the option to set results fields as stored, see the Solr Wiki page on SchemaXml.

For example, with `MapReduceIndexerTool` you can take advantage of fields such as `file_path`. See MapReduceIndexerTool Metadata on page 104 for more information. The output from the MapReduceIndexerTool includes file path information that can be used to construct links to source documents.

If you use the Hue UI, you can link to data in HDFS by inserting links of the form:

```
<a href="/filebrowser/download/{{file_path}}?disposition=inline">Download</a>
```

Why do I get an error “no field name specified in query and no default specified via 'df' param" when I query a Schemaless collection?

Schemaless collections initially have no default or `df` setting. As a result, simple searches that might succeed on non-Schemaless collections may fail on Schemaless collections.

When a user submits a search, it must be clear which field Cloudera Search should query. A default field, or `df`, is often specified in `solrconfig.xml`, and when this is the case, users can submit queries that do not specify fields. In such situations, Solr uses the `df` value.

When a new collection is created in Schemaless mode, there are initially no fields defined, so no field can be chosen as the `df` field. As a result, when query request handlers do not specify a `df`, errors can result. This issue can be addressed in several ways:

- Queries can specify any valid field name on which to search. In such a case, no `df` is required.
- Queries can specify a default field using the `df` parameter. In such a case, the `df` is specified in the query.
- You can uncomment the `df` section of the generated schemaless `solrconfig.xml` file and set the `df` parameter to the desired field. In such a case, all subsequent queries can use the `df` field in `solrconfig.xml` if no field or `df` value is specified.

Performance and Fail Over

The following are questions about performance and fail over in Cloudera Search and the answers to those questions.

How large of an index does Cloudera Search support per search server?

This question includes too many variables to provide a single answer. Typically, a server can host from 10 to 300 million documents, with the underlying index as large as hundreds of gigabytes. To determine a reasonable maximum document quantity and index size for servers in your deployment, prototype with realistic data and queries.

What is the response time latency I can expect?

Many factors affect how quickly responses are returned. Some factors that contribute to latency include whether the system is also completing indexing, the type of fields you are searching, whether the search results require aggregation, and whether there are sufficient resources for your search services.

With appropriately-sized hardware, if the query results are found in memory, they may be returned within milliseconds. Conversely, complex queries requiring results aggregation over huge indexes may take a few seconds.

The time between when Search begins to work on indexing new data and when that data can be queried can be as short as a few seconds, depending on your configuration.

This high performance is supported by custom caching optimizations that Cloudera has added to the Solr/HDFS integration. These optimizations allow for rapid read and writes of files in HDFS, performing at or above the speeds of stand-alone Solr reading and writing from local disk.

How does Cloudera Search performance compare to Apache Solr?

Assuming the same number of documents, size, hardware, and so on, with similar memory cache available, query performance is comparable. This is due to caching implemented by Cloudera Search. After the cache is warmed, the code paths for both Apache Solr and Cloudera Search are the same. You might see some performance degradation with cache misses when Cloudera Search must read from disk (HDFS), but that is mitigated somewhat by the HDFS block cache.

What happens when a write to the Lucene indexer fails?

Cloudera Search provides two configurable, highly available, and fault-tolerant data ingestion schemes: near real-time ingestion using the Flume Solr Sink and MapReduce-based batch ingestion using the MapReduceIndexerTool. These approaches are discussed in more detail in Search High Availability.

What hardware or configuration changes can I make to improve Search performance?

Search performance can be constrained by CPU limits. If you're seeing bottlenecks, consider allocating more CPU to Search.

Where should I deploy Solr Servers?

Cloudera recommends running them on DataNodes for locality. You can run a Solr Server on each DataNode, or a subset of DataNodes. Do not run Solr on any master nodes, such as NameNodes. Solr does not have a master server process. All Solr servers are the same. For more information, see Recommended Cluster Hosts and Role Distribution.

What happens if a host running a Solr Server process fails?

Cloudera Search uses the HDFS client API. Read and write requests for HDFS blocks will be automatically redirected as appropriate. If the failed or decommissioned host runs a DataNode process, any HDFS blocks on that host are re-replicated according to your HDFS configuration.

How is performance affected if the Solr Server is running on a node with no local data?

Because of the caching capabilities in Solr, performance impact is negligible.

Are there settings that can help avoid out of memory (OOM) errors during data ingestion?

A data ingestion pipeline can be made up of many connected segments, each of which may need to be evaluated for sufficient resources. A common limiting factor is the relatively small default amount of permgen memory allocated to

the flume JVM. Allocating additional memory to the Flume JVM may help avoid OOM errors. For example, for JVM settings for Flume, the following settings are often sufficient:

```
-Xmx2048m -XX:MaxPermSize=256M
```

How can I redistribute shards across a cluster?

You can move shards between hosts using the process described in Migrating Solr Replicas on page 64.

Can I adjust replication levels?

For information on adjusting replication levels, see Replication Settings. Do not adjust HDFS replication settings for Solr in most cases.

How do I manage resources for Cloudera Search and other components on the same cluster?

You can use cgroups to allocate resources among your cluster components.

Schema Management

The following are questions about schema management in Cloudera Search and the answers to those questions.

If my schema changes, will I need to re-index all of my data and files?

When you change the schema, Cloudera recommends re-indexing. For example, if you add a new field to the index, apply the new field to all index entries through re-indexing. Re-indexing is required in such a case because existing documents do not yet have the field. Cloudera Search includes a MapReduce batch-indexing solution for re-indexing and a GoLive feature that assures updated indexes are dynamically served.

While you should typically re-index after adding a new field, this is not necessary if the new field applies only to new documents or data. This is because, were indexing to be completed, existing documents would still have no data for the field, making the effort unnecessary.

For schema changes that only apply to queries, re-indexing is not necessary.

Can I extract fields based on regular expressions or rules?

Cloudera Search supports limited regular expressions in Search queries. For details, see Lucene Regular Expressions.

On data ingestion, Cloudera Search supports easy and powerful extraction of fields based on regular expressions. For example the `grok` morphline command supports field extraction using regular expressions.

Cloudera Search also includes support for rule directed ETL with an extensible rule engine, in the form of the `tryRules` morphline command.

Can I use nested schemas?

Cloudera Search supports nesting documents in this release. To learn about schemas with nested documents and their limitations, see Indexing Nested Documents.

What is Apache Avro and how can I use an Avro schema for more flexible schema evolution?

To learn more about Avro and Avro schemas, see the Avro Overview page and the Avro Specification page.

To see examples of how to implement inheritance, backwards compatibility, and polymorphism with Avro, see this InfoQ article.

Supportability

The following are questions about supportability in Cloudera Search and the answers to those questions.

Does Cloudera Search support multiple languages?

Cloudera Search supports approximately 30 languages, including most Western European languages, as well as Chinese, Japanese, and Korean.

Which file formats does Cloudera Search support for indexing? Does it support searching images?

Cloudera Search uses the Apache Tika library for indexing many standard document formats. In addition, Cloudera Search supports indexing and searching Avro files and a wide variety of other file types such as log files, Hadoop Sequence Files, and CSV files. You can add support for indexing custom file formats using a morphline command plug-in.

Troubleshooting Cloudera Search

After installing and deploying Cloudera Search, use the information in this section to troubleshoot problems.

Troubleshooting

The following table contains some common troubleshooting techniques.

Note: In the URLs in the following table, you must replace entries such as `<server:port>` with values from your environment. The port defaults value is 8983, but see `/etc/default/solr` or `/opt/cloudera/parcels/CDH-*/etc/default/solr` for the port if you are in doubt.

Symptom	Explanation	Recommendation
All	Varied	Examine Solr log. By default, the log can be found at `/var/log/solr/solr.out`.
The Solr Admin UI is unavailable	If no privileges are granted, no access is possible. For example, accessing the Solr Admin UI requires the QUERY privilege. If no users are granted the QUERY privilege, no access to the Solr Admin UI is possible.	Ensure users attempting to access the UI are granted the QUERY privilege. For more information, see Configuring Sentry Authorization for Cloudera Search.
No documents found	Server may not be running	Browse to http://*server*:*port*/solr to see if the server responds. Check that cores are present. Check the contents of cores to ensure that numDocs is more than 0.
No documents found	Core may not have documents	Browsing `http://server:port/solr/[collection name]/select?q=*:*&wt=json&indent=true` should show numFound, which is near the top, to be more than 0.
The secure Solr Server fails to respond to Solrj requests, but other clients such as curl can communicate successfully	This may be a version compatibility issue. `Httpclient` 4.2.3, which ships with solrj in Search 1.x, has a dependency on `commons-codec` 1.7. If an earlier version of `commons-codec` is on the classpath, `httpclient` may be unable to communicate using Kerberos.	Ensure your application is using `commons-codec` 1.7 or higher. Alternatively, use `httpclient` 4.2.5 instead of version 4.2.3 in your application. Version 4.2.3 behaves correctly with earlier versions of `commons-codec`.

Dynamic Solr Analysis

Any JMX-aware application can query Solr for information and display results dynamically. For example, Zabbix, Nagios, and many others have been used successfully. When completing Static Solr Log Analysis, many of the items related to extracting data from the log files can be seen from querying Solr, at least the last value (as opposed to the history which is available from the log file). These are often useful for status boards. In general, anything available from the Solr admin page can be requested on a live basis from Solr. Some possibilities include:

- `numDocs`/`maxDoc` per core. This can be important since the difference between these numbers indicates the number of deleted documents in the index. Deleted documents take up disk space and memory. If these numbers vary greatly, this may be a rare case where optimizing is advisable.
- Cache statistics, including:
  - Hit ratios

  - Autowarm times
  - Evictions
- Almost anything available on the admin page. Note that drilling down into the “schema browser” can be expensive.

Other Troubleshooting Information

Since the use cases for Solr and search vary, there is no single solution for all cases. That said, here are some common challenges that many Search users have encountered:

- Testing with unrealistic data sets. For example, a users may test a prototype that uses faceting, grouping, sorting, and complex schemas against a small data set. When this same system is used to load of real data, performance issues occur. Using realistic data and use-cases is essential to getting accurate results.
- If the scenario seems to be that the system is slow to ingest data, consider:
  - Upstream speed. If you have a SolrJ program pumping data to your cluster and ingesting documents at a rate of 100 docs/second, the gating factor may be upstream speed. To test for limitations due to upstream speed, comment out only the code that sends the data to the server (for example, `SolrHttpServer.add(doclist)`) and time the program. If you see a throughput bump of less than around 10%, this may indicate that your system is spending most or all of the time getting the data from the system-of-record.
  - This may require pre-processing.
  - Indexing with a single thread from the client. `ConcurrentUpdateSolrServer` can use multiple threads to avoid I/O waits.
  - Too-frequent commits. This was historically an attempt to get NRT processing, but with SolrCloud hard commits this should be quite rare.
  - The complexity of the analysis chain. Note that this is rarely the core issue. A simple test is to change the schema definitions to use trivial analysis chains and then measure performance.
  - When the simple approaches fail to identify an issue, consider using profilers.

Cloudera Search Configuration and Log Files

Cloudera Search (powered by Apache Solr) configuration is managed using Cloudera Manager. You can view log files on the individual Solr server or using Cloudera Manager.

Viewing and Modifying Cloudera Search Configuration

To view and edit configuration parameters for the **Solr** service in Cloudera Manager:

1. Go to **Solr service** > **Configuration**.
2. Use the **SCOPE** and **CATEGORY** filters to restrict the displayed configuration parameters. You can also enter text in the **Search** field to dynamically filter configuration parameters.
3. After making changes, enter a **Reason for change**, and click **Save Changes**.
4. Restart the **Solr** service (**Solr service** > **Actions** > **Restart**) and any dependent services.

Viewing and Modifying Log Levels for Cloudera Search and Related Services

You can view and modify log levels for Search and related services (such as Apache Flume, Apache HBase, and so on) by:

1. Go to the service configuration page. For example: **Solr service** > **Configuration**
2. Select the **CATEGORY** > **Logs** filter.
3. Modify the logging threshold for the service roles. Available options (in descending verbosity) are:
   - **TRACE**
   - **DEBUG**
   - **INFO**
   - **WARN**

- **ERROR**
- **FATAL**

4. Enter a **Reason for change**, and then click **Save Changes** to commit the changes.
5. Restart the service. For example: **Solr service** > **Actions** > **Restart**

   You can also restart dependent services using the Restart Stale Services wizard.

Cloudera Search Log Files

Cloudera Search has several log files, stored on each **Solr Server** host. See the following for log file locations and a brief description of each. The referenced configuration parameters can be viewed or modified as described in Viewing and Modifying Cloudera Search Configuration on page 128:

- **`/var/log/solr/`** - Parent directory for most Search logs.
  - **`audit/`** - Audit log directory, specified by the **Audit Log Directory** configuration parameter.
  - **`stacks/`** - Directory containing period stack dumps, if **Stacks Collection Enabled** is checked. The directory name can be changed by setting the **Stacks Collection Directory** parameter.
  - `solr-cmf-`*`SOLR-1`*`-SOLR_SERVER-`*`hostname.example.com`*`.log.out` - Solr server log file. The *SOLR-1* and *hostname.example.com* portion varies depending on your environment. The log default log level is **INFO**, and is controlled by the **Solr Server Logging Threshold** parameter for **Solr Server** roles, and the **Gateway Logging Threshold** parameter for **Gateway** roles.
  - `solr_gc_log.*` - Java garbage collection (GC) logs for the Solr Server process.
- **`/var/run/cloudera-scm-agent/process/<`***`process_dir_id`***`>-solr-SOLR_SERVER/`** - Configuration directory for the currently running **Solr Server** role.

  > **Warning:** Do not modify any of the files in this directory. These files are automatically generated by Cloudera Manager. To modify any configuration parameters, use Cloudera Manager.

  The <*process_dir_id*> portion changes each time the **Solr Server** is restarted. To identify the current or most recent directory, list the `/var/run/cloudera-scm-agent/process/*solr*` directories sorted by time in reverse as follows:

```
sudo ls -ltrd /var/run/cloudera-scm-agent/process/*solr*
```

  The entry at the bottom is the current or most recent process directory.

  - **`logs/`** - Log directory containing the `stderr` and `stdout` logs for the **Solr Server** process.

Identifying Problems in Your Cloudera Search Deployment

To investigate your Cloudera Search deployment for performance problems or misconfigurations, inspect the log files, schema files, and the actual index for issues. If possible, connect to the live Solr instance while watching log files so you can compare the schema with the index. For example, the schema and the index can be out of sync in situations where the schema is changed, but the index was never rebuilt. See the following list for some common issues and what you can do about them:

- A high number or proportion of 0-match queries. This indicates that the user-facing part of the application is making it easy for users to enter queries for which there are no matches. In Cloudera Search, given the size of the data, this should be an extremely rare event.
- Queries that match an excessive number of documents. All documents that match a query have to be scored, and the cost of scoring a query goes up as the number of hits increases. Examine any frequent queries that match millions of documents. An exception to this case is “constant score queries”. Queries, such as those of the form ":" bypass the scoring process entirely.

- Overly complex queries. Defining what constitutes overly complex queries is difficult to do, but a very general rule is that queries over 1024 characters in length are likely to be overly complex.
- High autowarm times. Autowarming is the process of filling caches. Some queries are run before a new searcher serves the first live user request. This keeps the first few users from having to wait. Autowarming can take many seconds or can be instantaneous. Excessive autowarm times often indicate excessively generous autowarm parameters. Excessive autowarming usually has limited benefit, with longer runs effectively being wasted work.
  - Cache autowarm. Each Solr cache has an autowarm parameter. You can usually set this value to an upper limit of 128 and tune from there.
  - `FirstSearcher/NewSearcher`. The `solrconfig.xml` file contains queries that can be fired when a new searcher is opened (the index is updated) and when the server is first started. Particularly for `firstSearcher`, it can be valuable to have a query that sorts relevant fields.

Note: The aforementioned flags are available from `solrconfig.xml`

- Exceptions. The Solr log file contains a record of all exceptions thrown. Some exceptions, such as exceptions resulting from invalid query syntax are benign, but others, such as Out Of Memory, require attention.
- Excessively large caches. The size of caches such as the filter cache are bounded by maxDoc/8. Having, for instance, a filterCache with 10,000 entries is likely to result in Out Of Memory errors. Large caches occurring in cases where there are many documents to index is normal and expected.
- Caches with low hit ratios, particularly filterCache. Each cache takes up some space, consuming resources. There are several caches, each with its own hit rate.
  - `filterCache`. This cache should have a relatively high hit ratio, typically around 80%.
  - `queryResultCache`. This is primarily used for paging so it can have a very low hit ratio. Each entry is quite small as it is basically composed of the raw query as a string for a key and perhaps 20-40 `ints`. While useful, unless users are experiencing paging, this requires relatively little attention.
  - `documentCache`. This cache is a bit tricky. It's used to cache the document data (stored fields) so various components in a request handler don't have to re-read the data from the disk. It's an open question how useful it is when using `MMapDirectory` to access the index.
- Very deep paging. Users seldom go beyond the first page and very rarely to go through 100 pages of results. A `&start=<pick your number>` query indicates unusual usage that should be identified. Deep paging may indicate some agent is completing scraping.

Note: Solr is not built to return full result sets no matter how deep. If returning the full result set is required, explore alternatives to paging through the entire result set.

- Range queries should work on `trie` fields. `Trie` fields (numeric types) store extra information in the index to aid in range queries. If range queries are used, it's almost always a good idea to be using `trie` fields.
- `fq` clauses that use bare NOW. `fq` clauses are kept in a cache. The cache is a map from the `fq` clause to the documents in your collection that satisfy that clause. Using bare NOW clauses virtually guarantees that the entry in the filter cache is not be re-used.
- Multiple simultaneous searchers warming. This is an indication that there are excessively frequent commits or that autowarming is taking too long. This usually indicates a misunderstanding of when you should issue commits, often to simulate Near Real Time (NRT) processing or an indexing client is improperly completing commits. With NRT, commits should be quite rare, and having more than one simultaneous autowarm should not happen.
- Stored fields that are never returned (`fl=` clauses). Examining the queries for `fl=` and correlating that with the schema can tell if stored fields that are not used are specified. This mostly wastes disk space. And `fl=*` can make this ambiguous. Nevertheless, it's worth examining.
- Indexed fields that are never searched. This is the opposite of the case where stored fields are never returned. This is more important in that this has real RAM consequences. Examine the request handlers for "edismax" style parsers to be certain that indexed fields are not used.

- Queried but not analyzed fields. It's rare for a field to be queried but not analyzed in any way. Usually this is only valuable for "string" type fields which are suitable for machine-entered data, such as part numbers chosen from a pick-list. Data that is not analyzed should not be used for anything that humans enter.
- String fields. String fields are completely unanalyzed. Unfortunately, some people confuse `string` with Java's `String` type and use them for text that should be tokenized. The general expectation is that string fields should be used sparingly. More than just a few string fields indicates a design flaw.
- Whenever the schema is changed, re-index the entire data set. Solr uses the schema to set expectations about the index. When schemas are changed, there's no attempt to retrofit the changes to documents that are currently indexed, but any new documents are indexed with the new schema definition. So old and new documents can have the same field stored in vastly different formats (for example, `String` and `TrieDate`) making your index inconsistent. This can be detected by examining the raw index.
- Query stats can be extracted from the logs. Statistics can be monitored on live systems, but it is more common to have log files. Here are some of the statistics you can gather:
  - Longest running queries
  - 0-length queries
  - average/mean/min/max query times
  - You can get a sense of the effects of commits on the subsequent queries over some interval (time or number of queries) to see if commits are the cause of intermittent slowdowns
- Too-frequent commits have historically been the cause of unsatisfactory performance. This is not so important with NRT processing, but it is valuable to consider.
- Optimizing an index, which could improve search performance before, is much less necessary now. Anecdotal evidence indicates optimizing may help in some cases, but the general recommendation is to use `expungeDeletes`, instead of committing.
  - Modern Lucene code does what `optimize` used to do to remove deleted data from the index when segments are merged. Think of this process as a background optimize. Note that merge policies based on segment size can make this characterization inaccurate.
  - It still may make sense to optimize a read-only index.
  - `Optimize` is now renamed `forceMerge`.

Solr Query Returns no Documents when Executed with a Non-Privileged User

With a default secure collection configuration, you might have noticed that even if your user has the full set of Sentry privileges to access a particular collection, there are no documents returned for queries executed by the user, but executing the same query with the Solr superuser (`solr` by default) gives the expected results.

Why this Happens:

Starting with CDH 6, the default secure templates for `solrconfig.xml`, in addition to the collection level security, also enable the Solr document level security. For more information, see Providing Document-Level Security Using Sentry. You can verify this by looking at the `solrconfig.xml` of your collection and verifying whether it has the `QueryDocAuthorizationComponent` enabled:

```
<searchComponent name="queryDocAuthorization"
class="org.apache.solr.handler.component.QueryDocAuthorizationComponent" >
     <!-- Set to true to enabled document-level authorization -->
     <bool name="enabled">true</bool>
     <!-- Field where the auth tokens are stored in the document -->
     <str name="sentryAuthField">sentry_auth</str>
     <!-- Auth token defined to allow any role to access the document.
          Uncomment to enable. -->
     <!--<str name="allRolesToken">*</str>-->
   </searchComponent>
```

This article uses an example with a hypothetical `systest` user, belonging to a group which has the `my_role` Sentry role assigned, granting access to the collection 'myTestCollection'.

If you add a document to 'myTestCollection' as the `systest` user:

```
kinit systest
```

```
curl -k --negotiate -u : -X POST
'https://mysolrserver-1.example.com:8985/solr/myTestCollection/update?wt=json' -H
'content-type: application/json' -d '[{"id":"1","title":"My document without access
token defined"}]'
```

The add operation is successful, but whenever you query the collection, this document is not returned among search results.

> **Note:** If you execute the queries using the `solr` user, which is a superuser for Solr, no authorization checks will be done, therefore all documents matching the query will be returned.

What to Do:

You may choose one of the following options to make the queries working as expected:

- Leave the document level security enabled, and whenever inserting a document, specify the role that will be able to query that document.
- Leave document level security enabled, and also enable the All Roles Token.
- Disable document level security and rely only on collection level security.

Leaving the Document Level Security Enabled, and Specifying the Role Able to Query a Document at Insertion

You may opt for leaving document level security enabled and specifying the role that will be able to query a given document at insertion time.

When document level security is enabled, each document stores a field which specifies the roles that are able to query that particular document. In the default configuration this field is called `sentry_auth`. If you add a new document to this Solr collection where the `sentry_auth` field of this document has the value `'my_role'`, only the users who have the Sentry role `'my_role'` will get this document returned in any search results.

```
kinit systest
```

```
curl -k --negotiate -u : -X POST
'https://mysolrserver-1.example.com:8985/solr/myTestCollection/update?wt=json' -H
'content-type: application/json' -d '[{"id":"19","title":"My document accessible by
my_role", "sentry_auth":"my_role"}]'
```

Since the `systest` user in the example is in a group which has the role `my_role`, the recently added document is successfully retrieved:

```
curl -k --negotiate -u :
"https://mysolrserver-1.example.com:8985/solr/myTestCollection/select?q=:"
```

```
{
  "responseHeader":{
    "zkConnected":true,
    "status":0,
    "QTime":18,
    "params":{
      "q":"*:*",
      "doAs":"systest"}},
  "response":{"numFound":1,"start":0,"docs":[
      {
        "id":"19",
```

```
        "title":["My document accessible by myrole"],
        "_version_":1616957635311960064}]
  }}
```

Leaving Document Level Security Enabled and Enabling the All Roles Token

The document level security feature also allows defining a token (called the `allRolesToken`) which allows any role to access a particular document which has this token in the `sentryAuthField` specified. If in the `solrconfig.xml` file you uncomment the line `<str name="allRolesToken">*</str>` it results in all Sentry roles being able to query all documents that are submitted with the `sentry_auth` field value set to `*`.

```
<searchComponent name="queryDocAuthorization"
class="org.apache.solr.handler.component.QueryDocAuthorizationComponent" >
     <!-- Set to true to enabled document-level authorization -->
     <bool name="enabled">true</bool>
     <!-- Field where the auth tokens are stored in the document -->
     <str name="sentryAuthField">sentry_auth</str>
     <!-- Auth token defined to allow any role to access the document.
          Uncomment to enable. -->
     <!--<str name="allRolesToken">*</str>-->
   </searchComponent>
```

This option enables you to specify for individual documents that all users can access them, regardless of roles.

1. Modify the `solrconfig.xml` of your collection and uncomment: `<str name="allRolesToken">*</str>`
2. After updating the `solrconfig.xml`, do a reload operation on the collection (In this example: `solrctl collection --reload myTestCollection`).
3. You can now add a document with setting the `sentry_auth` field to the defined `allRolesToken` (`*`):

```
kinit systest
```

```
curl -k --negotiate -u : -X POST
'https://mysolrserver-1.example.com:8985/solr/myTestCollection/update?wt=json' -H
'content-type: application/json' -d '[{"id":"20","title":"My document with allRolesToken",
 "sentry_auth":"*"}]'
```

Do a query with the `systest` user:

```
kinit systest
```

```
curl -k --negotiate -u :
"https://mysolrserver-1.example.com:8985/solr/myTestCollection/select?q=:"
```

It returns two documents (id `19` had the `sentry_auth` set to `my_role` which is the Sentry role associated with the `systest` user and id `20` which has `sentry_auth` set to `*` which is 'any roles')

```
"response":{"numFound":2,"start":0,"docs":[
      {
        "id":"19",
        "title":["My document accessible by myrole"],
        "_version_":1616957635311960064},
      {
        "id":"20",
        "title":["My document with allRolesToken"],
        "_version_":1616958682453508096}]
```

Disabling Document Level Security and Relying Only on Collection Level Security

This option means that whoever has a Sentry role giving access to a collection will be able to access all documents in that collection. To implement this option:

1. Locate the following entry in `solrconfig.xml`:

```
<searchComponent name="queryDocAuthorization"
class="org.apache.solr.handler.component.QueryDocAuthorizationComponent" >
```

2. Change

```
<bool name="enabled">true</bool>
```

to

```
<bool name="enabled">false</bool>
```

3. Update the instance configuration. Update and reload the collection configuration.

Repeat the query and it returns all documents in the collection (The document with id `1` did not have a value for the `sentry_auth` field, meaning that no one was able to query it, however you have disabled document level security and use collection level security only, so the `systest` user who has access to the collection will now get all documents in that collection):

```
kinit systest
```

```
curl -k --negotiate -u :
"https://mysolrserver-1.example.com:8985/solr/myTestCollection/select?q=:"
```

```
{
  "responseHeader":{
    "zkConnected":true,
    "status":0,
    "QTime":7,
    "params":{
      "q":"*:*",
      "doAs":"systest"}},
  "response":{"numFound":3,"start":0,"docs":[
        {
          "id":"1",
          "title":"My document without access token defined",
          "_version_":1616949033619685376},
        {
          "id":"19",
          "title":"My document accessible by myrole",
          "_version_":1616957635311960064},
        {
          "id":"20",
          "title":"My document with allRolesToken",
          "_version_":1616958682453508096}]
  }}
```

Appendix: Apache License, Version 2.0

SPDX short identifier: Apache-2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims

licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
2. You must cause any modified files to carry prominent notices stating that You changed the files; and
3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

```
Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```