Cloudera Runtime 7.2.7

Cloudera Search Overview

Date published: 2019-11-19
Date modified:

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

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

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

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

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

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

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

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

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

Contents

What is Cloudera Search

Learn about how Cloudera Search is different from Apache Solr and the added value it provides.

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 Cloudera Data Platform (CDP). 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 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.

Using Cloudera Search with the CDP 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 Cloudera Runtime 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
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 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.
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.

Feature	Description
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 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.
Spark-Solr connector	You can use the Spark-Solr connector to index data into Solr in multiple ways, both batch and streaming. With the connector you get the benefits of Spark while you can access Solr easily and in a familiar way, from both Scala and Java.

How Cloudera Search works

Learn about the operation of Cloudera Search

In near real-time indexing use cases, such as log or event stream analytics, Cloudera Search indexes events that are streamed through 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.

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.

Cloudera Search and CDP

Discover how Search fits into Cloudera's Data Platform offering.

Cloudera Search fits into the broader set of solutions available for analyzing information in large data sets. Cloudera Data Platform (CDP) 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

CDP 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.

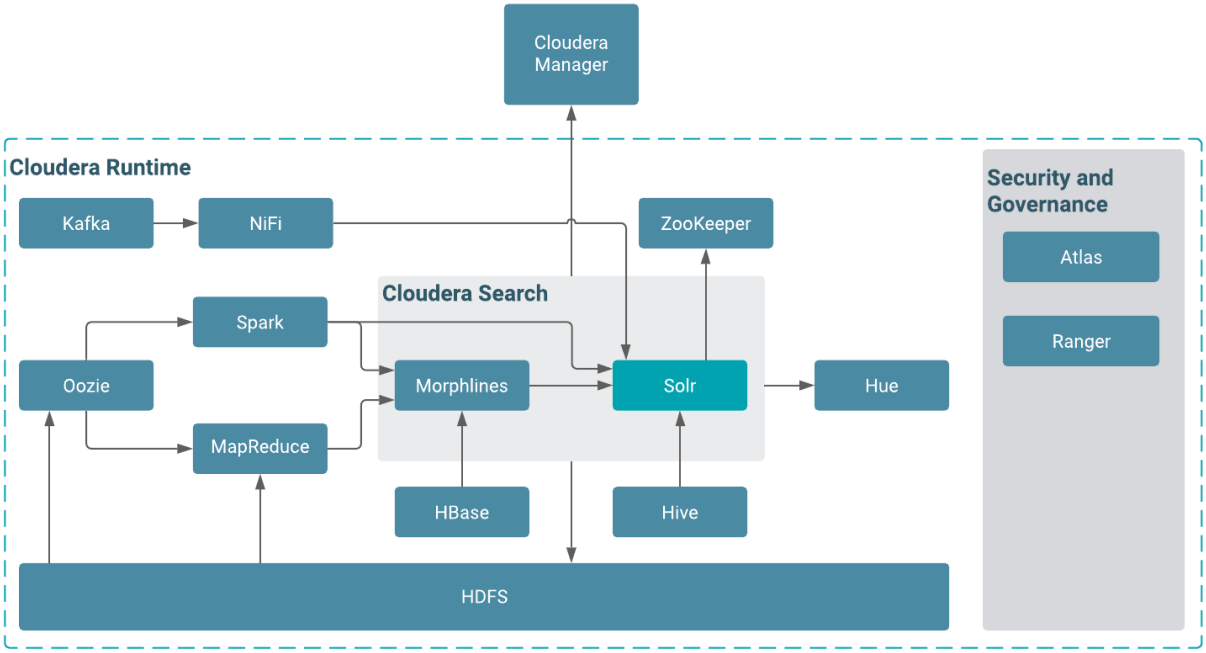
How Search Uses Existing Infrastructure

Any data already in a CDP deployment can be indexed and made available for query by Cloudera Search. For data that is not stored in CDP, 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 CDP 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 Runtime components

Cloudera Search interacts with other Cloudera Runtime components to solve different problems. Learn about Cloudera components that contribute to the Search process and see how they interact with Cloudera Search.



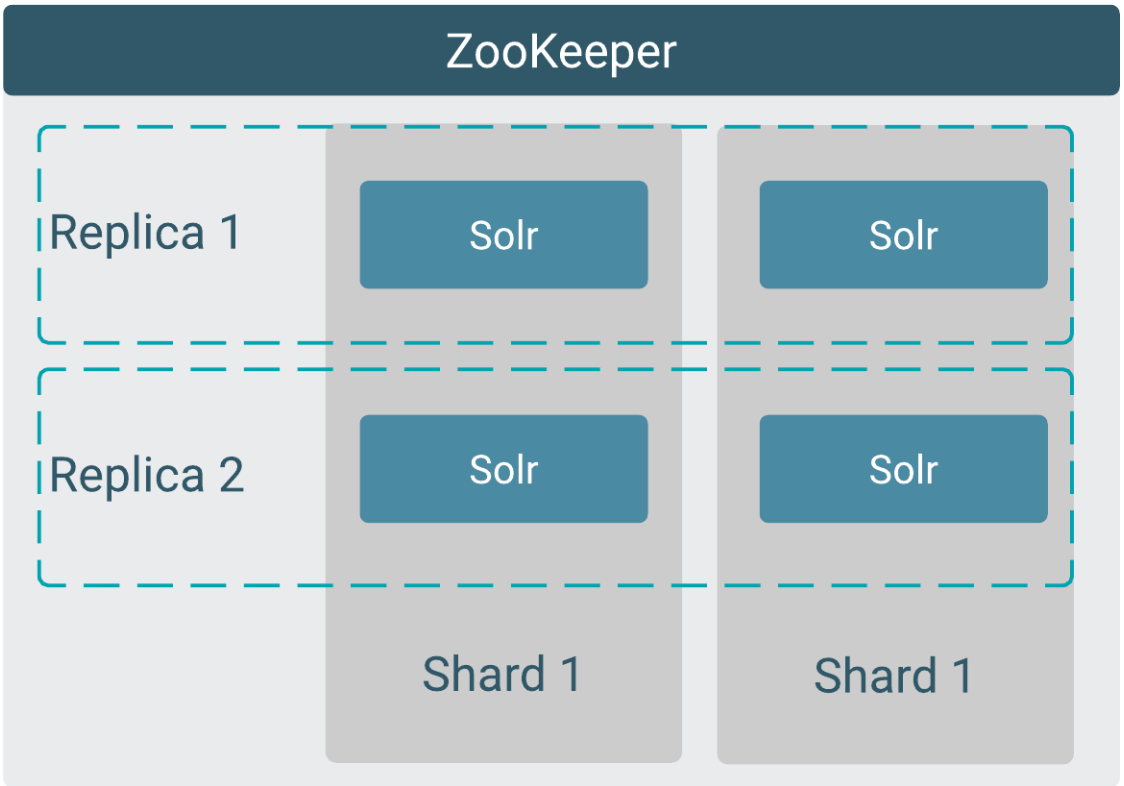
Component	Contribution	Applicable To
HDFS	Stores source documents. Search indexes source documents to make them searchable. Files that support Cloudera Search, such as Lucene index files and write-ahead logs, are also stored in HDFS. Using HDFS provides simpler provisioning on a larger base, redundancy, 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
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
NiFi	Used for real time and often voluminous incoming event streams that need to be explorable (e.g. logs, twitter feeds, file appends etc).	
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.	Some cases
Atlas	Atlas classifications drive data access control through Ranger.	Some cases

Component	Contribution	Applicable To
Ranger	Enables role-based, fine-grained authorization for Cloudera Search. Ranger can apply restrictions to various actions, such as accessing data, managing configurations through config objects, or creating collections. Restrictions are consistently enforced, regardless of how users attempt to complete actions. Ranger offers both resource-based and tag-based access control policies.	Some cases
Oozie	Automates scheduling and management of indexing jobs. Oozie can check for new data and begin indexing jobs as required.	Some cases
Hive	Further analyzes search results.	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.

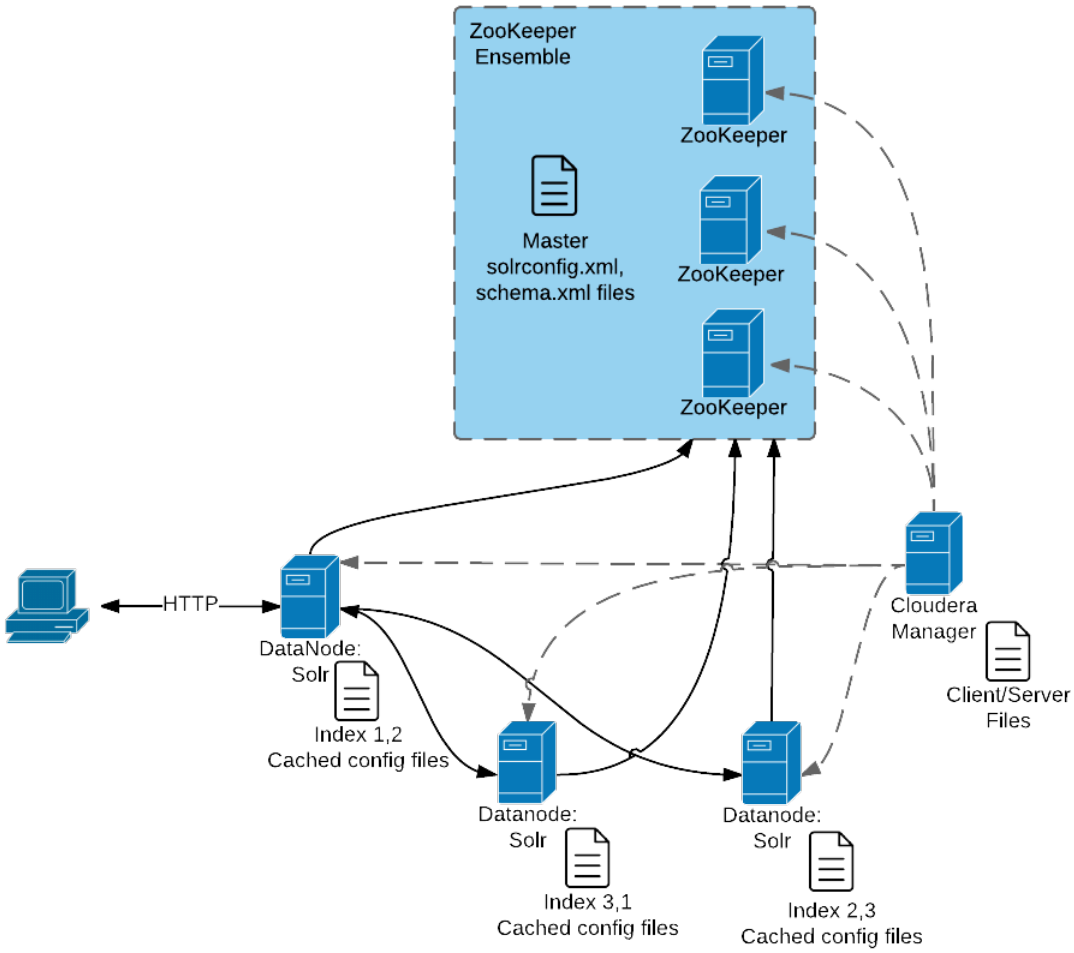
Each Search deployment requires:

- 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 co-location 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 tasks and processes

For content to be searchable, it must exist in Cloudera Data Platform (CDP) and be indexed. Content can either already exist in CDP 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 CDP.

Ingestion

Ingestion is about making data available in Cloudera Data Platform (CDP).

You can move content to CDP by using:

- Apache NiFi, a flexible data streaming 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. Learn about how indexing in Cloudera Search happens.

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 CDP-stored indexes. All index data in Cloudera Search is stored in and served from HDFS.

You can index content in the following 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.

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.

Indexing using the Spark-Solr connector

Using the Spark-Solr connector you have two options to index data into Solr: batch index documents or index streaming data.

For batch indexing you have multiple options, firstly you can use the spark-shell tool to reach Solr with Scala commands. Cloudera recommends this solution mostly for experimenting or smaller indexing jobs.

The second option is to use spark-submit with your spark job, for this you need to create a class which implements SparkApp.RDDProcesor interface. This way you can access the robustness of Spark with the phrases and concepts well-known from Solr. This works in Scala and Java as well.

If you want to index streaming data, you can do it by implementing the SparkApp.StreamingProcessor interface. With this solution you gain access to all the benefits of SparkStreaming and send on the data to Solr.

Querying

After data is available as an index, you can run queries against it.

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 at its core.