

Cloudera Runtime 7.1.3

Troubleshooting Cloudera Search

Date published: 2019-11-19

Date modified:

CLOUDERA

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

Legal Notice

© Cloudera Inc. 2025. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Troubleshooting Cloudera Search.....	4
Cloudera Search Configuration and Log Files.....	5
Cloudera Search Configuration Files.....	5
View and Modify Cloudera Search Configuration.....	6
Cloudera Search Log Files.....	6
View and Modify Log Levels for Cloudera Search and Related Services.....	7
Identifying problems in a Cloudera Search deployment.....	7

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.
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.

Cloudera Search Configuration Files

Cloudera Search configuration is primarily controlled by several configuration files, that are mostly stored in Apache ZooKeeper.

Table 1: Cloudera Search configuration files

Configuration File	Description
<code>solr.xml</code>	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 <code>solrctl</code> cluster command. For information about the <code>solr.xml</code> file, see Solr Configuration Files and Solr Cores and solr.xml in the Solr documentation.
<code>solrconfig.xml</code>	Each collection in Solr uses a <code>solrconfig.xml</code> file, stored in ZooKeeper, to control collection behavior. For information about the <code>solrconfig.xml</code> file, see Solr Configuration Files and Configuring solrconfig.xml in the Solr documentation.
<code>managed-schema</code> or <code>schema.xml</code>	Cloudera recommends using a managed schema, and making schema changes using the Schema API (Apache Solr documentation) . Collections use either a managed schema or the legacy <code>schema.xml</code> 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 <code>schema.xml</code> , see Schema Factory Definition in SolrConfig in the Solr documentation.
<code>core.properties</code>	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.

Configuration File	Description
Additional files	Any additional files referenced in the xml files, for example, custom JAR files.

View and Modify Cloudera Search Configuration

Learn about viewing and editing configuration parameters for the Solr service in Cloudera Manager.

Before you begin

Procedure

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. 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 Cloudera Manager.

Logs under /var/log/solr/

/var/log/solr/ is the parent directory for most Search logs.

Table 2: Logs and directories under /var/log/solr/

Log or Directory	Description	Configuration Options
audit/	Audit log directory.	The path may be specified by the Audit Log Directory configuration parameter.
stacks/	The directory in which stacks logs are placed.	Logs are collected if the Stacks Collection Enabled option is checked. Stacks Collection Directory - The directory in which stacks logs are placed. If not set, stacks are logged into a stacks subdirectory of the role's log directory.
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.	Solr Server Logging Threshold parameter sets log level for Solr Server roles. Gateway Logging Threshold parameter sets log level for Gateway roles.
solr_gc_log.*	Java garbage collection (GC) logs for the Solr Server process.	

Logs under /var/run/cloudera-scm-agent/process/<PROCESS_DIR_ID>-SOLR-SOLR_SERVER/

/var/run/cloudera-scm-agent/process/<PROCESS_DIR_ID>-SOLR-SOLR_SERVER/ is the configuration directory for the currently running Solr Server role. The <PROCESS_DIR_ID> portion changes each time the Solr Server is restarted.



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.

Table 3: Logs under /var/run/cloudera-scm-agent/process/<PROCESS_DIR_ID>-SOLR-SOLR_SERVER/

Log Directory	Description
logs/	Log directory containing the stderr and stdout logs for the Solr Server process.

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.

Related Tasks

[View and Modify Cloudera Search Configuration](#)

View and Modify Log Levels for Cloudera Search and Related Services

Learn about viewing and modifying log levels for Search and related services (such as Apache HBase).

Procedure



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 click Save Changes.
5. Restart the Solr service (Solr service Actions Restart) and any dependent services.

Identifying problems in a Cloudera Search deployment

Learn about common issues affecting Search performance and what you can do about them.

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 table for some common issues and what you can do about them:

Issue	Recommendation
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.

Issue	Recommendation
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	<p>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.</p> <ul style="list-style-type: none"> 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. <p> Note: The aforementioned flags are available from solrconfig.xml</p>
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	<p>Each cache takes up some space, consuming resources. There are several caches, each with its own hit rate.</p> <ul style="list-style-type: none"> 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	<p>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.</p> <p> 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.</p>
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 to be re-used.

Issue	Recommendation
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.
The schema and the index are out of sync	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. Whenever the schema is changed, re-index the entire data set.
Query stats	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: <ul style="list-style-type: none"> • 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.