Cloudera Runtime 7.2.15

Managing Cloudera Search

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

Cloudera Search log files 4
Viewing and modifying log levels for Search and related services 4

Viewing and modifying Search configuration using Cloudera Manager 5
Cloudera Search configuration files 5

Managing collection configuration using configs or instance directories 6
Cloudera Search config templates 7
Generating collection configuration using configs 7
Securing configs with ZooKeeper ACLs and Ranger 8
Generating Solr collection configuration using instance directories 9
Modifying a collection configuration generated using an instance directory 10
Converting instance directories to configs 11
Using custom JAR files with Search 12
Retrieving the clusterstate.json file 13

Managing collections in Search 14
Creating a Solr collection 14
Viewing existing collections 15
Deleting all documents in a collection 15
Deleting a collection 16
Updating the schema in a collection 16

Creating a replica of an existing shard 17

Migrating Solr replicas 17

Backing up a collection from HDFS 20

Backing up a collection from local file system 22

Restoring a collection 24

Defining a backup target in solr.xml 26

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 1: 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 2: 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.

Viewing and modifying log levels for 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.

Viewing and modifying Search configuration using Cloudera Manager

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 configuration files

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

Configuration file	Description
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.
solrconfig.xml	Each collection in Solr uses a solrconfig.xml file, stored in ZooKeeper, to control collection behavior.
managed-schema or schema.xml	Cloudera recommends using a managed schema, and making schema changes using the Schema API. Collections 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

Configuration file	Description
	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.
core.properties	Unlike other configuration files, this file is stored in the local filesystem rather than ZooKeeper, and is used for core discovery.
Additional files	Any additional files referenced in the xml files, for example, custom JAR files.

Related Information

Solr configuration files
Solr cores and solr.xml
Configuring solrconfig.xml
Schema API
Schema factory definition in SolrConfig
Defining core.properties

Managing collection configuration using configs or instance directories

Both configs and instance directories are configuration sets for Solr collections that can be referenced by their respective names. Although configs and instance directories are functionally identical from the perspective of the Solr server, there are a number of important administrative differences between these two implementations.

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 *[***CONFIG NAME***]* command.

Table 3: 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.	• No ZooKeeper security support. Any user can create, delete, or modify an instance dir 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.

Attribute	Config	Instance Directory
Template support	• 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.

Cloudera Search config templates

Config templates are immutable configuration templates that you can use as a starting point when creating configs for Solr collections. Cloudera Search contains templates by default and you can define new ones based on existing configs.

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 4: Available Config Templates and Attributes

Template Name	Supports Schema API	Uses Schemaless Solr
managedTemplate	Yes	No
schemalessTemplate	Yes	Yes

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 managedTemplate template:

  ```
  solrctl config --create [***NEW CONFIG***] managedTemplate -p immutable=false
  ```

  Replace *[***NEW CONFIG***]* with the name of the config you want to create.
- To create a new template (immutable config) from an existing config:

  ```
  solrctl config --create [***NEW TEMPLATE***] [***EXISTING CONFIG***] -p immutable=true
  ```

  Replace *[***NEW TEMPLATE***]* with a name for the new template you want to create and *[***EXISTING CONFIG***]* with the name of the existing config that you want to base *[***NEW TEMPLATE***]* on.

Generating collection configuration using configs

You must create a collection configuration prior to creating a Solr collection. The configuration files are created in ZooKeeper based on existing templates using the ConfigSets API. Learn how to create one using configs.

About this task

Configs are named configuration sets that you can reference when creating collections.

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

```
solrctl config --create [***NEW CONFIG***] [***TEMPLATE***] [-
p [***NAME***]=[***VALUE***]]
```

Procedure

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

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

   Replace EXAMPLE.COM with your Kerberos realm name.
2. To generate configuration files for a collection, run the following command:

   ```
   solrctl config --create [***NEW CONFIG***] [***TEMPLATE***] -p immutabl
   e=false
   ```

   where

   [NEW CONFIG***]***

   is the user-specified name of the config

   [TEMPLATE***]***

   is the name of an existing config template

   To list all available config templates, use the solrctl instancedir --list command.

   -p ***[***NAME***]=[***VALUE***]***

   Overrides a *[***TEMPLATE***]* setting. The only config property that you can override is immu table, 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.

   For example, to create the configuration logs_config based on managedTemplate:

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

Securing configs with ZooKeeper ACLs and Ranger

Learn how you can restrict access to configuration sets by setting ZooKeeper Acces control Lists (ACLs) on all znodes under and including the /solr directory and using Ranger to control access to the ConfigSets API.

Before you begin

Ranger requires Kerberos authentication.

About this task

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

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

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.

Procedure

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 so that it points 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 FILE***] \
             -DzkACLProvider=org.apache.solr.common.cloud.SaslZkACLProvid
er \
             -Droot.logger=INFO,console"
```

   Replace *[***PATH TO JAAS.CONF FILE***]* with the path pointing to the jaas.conf file you just created.

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:

```
/opt/cloudera/parcels/CDH/lib/solr/bin/zkcli.sh -zkhost [***ZOOKEEPER
 SERVER HOSTNAME***]:2181 -cmd updateacls /solr
```

   Replace *[***ZOOKEEPER SERVER HOSTNAME***]* with the hostname of a ZooKeeper server.

Generating Solr collection configuration using instance directories

You must create a collection configuration prior to creating a Solr collection. The configuration files for a Solr collection are stored in a directory called instance directories. Learn how to create the directory and make it available to Solr by uploading the contents to Zookeeper.

Before you begin

Note:

If you want to control access to configuration sets, you must enable ZooKeeper ACLs and use configs instead.

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

About this task

In this case, configuration files for a collection are contained in a directory called an instance directory. 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.

Procedure

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

   ```
   kinit solradmin@[***EXAMPLE.COM***]
   ```

   Replace *[***EXAMPLE.COM***]* with your Kerberos realm name.
2. To generate a template instance directory, run the following command:

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

3. Customize the collection by directly editing the solrconfig.xml and schema.xml files created in $HOME/solr_configs/conf.
4. After completing the configuration, make it available to Solr by running the following command, which uploads the contents of the instance directory to ZooKeeper:

   ```
   solrctl config --upload [***COLLECTION_NAME***] $HOME/solr_configs
   ```

   For example:

   ```
   solrctl config --upload weblogs $HOME/solr_configs
   ```

5. 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
   ```

   If you used the --create command to create a collection named weblogs, the --list command returns weblogs.

Related Information

solrctl Reference

Modifying a collection configuration generated using an instance directory

The configuration files for a Solr collection are stored in a directory called instance directories. Learn how to modify the directory and make it available to Solr by uploading the contents to ZooKeeper.

About this task

In this case, configuration files for a collection are contained in a directory called an instance directory. An instance directory is a named set of configuration files. You can download and edit the configuration locally and then upload the directory to ZooKeeper as a named configuration set. If your instance directory is already referenced by a collection, make sure to reload it for configuration changes to take effect.

Procedure

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

   ```
   kinit solradmin@[***EXAMPLE.COM***]
   ```

   Replace *[***EXAMPLE.COM***]* with your Kerberos realm name.

2. To download a template instance directory, run the following command:

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

3. Customize the collection by directly editing the solrconfig.xml and schema.xml files.
4. After completing the configuration, make it available to Solr by running the following command, which uploads the contents of the instance directory to ZooKeeper:

```
solrctl config --upload [***COLLECTION_NAME***] $HOME/solr_configs
```

For example:

```
solrctl config --upload weblogs $HOME/solr_configs
```

5. 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
```

If you used the --create command to create a collection named weblogs, the --list command returns weblogs.

6. After uploading the updated collection configuration. you need to reload every collection referencing that instancce directory for the updates to take effect.

```
solrctl collection --reload [***COLLECTION_NAME***]
```

Converting instance directories to configs

Cloudera Search supports converting existing deployments that use instance directories to use configs. This allows you to implement access control using ZooKeeper Access Control Lists and Ranger.

Procedure

1. Create a temporary config based on the existing instance directory.

```
solrctl config --create [***TEMPORARY NEW CONFIG***] [***EXISTING
 INSTANCEDIR***] \
-p immutable=false
```

Replace *[***TEMPORARY NEW CONFIG***]* with a config name and *[***EXISTING INSTANCEDIR***]* with the name of the instancedir you want to convert.

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.

```
solrctl instancedir --delete [***EXISTING INSTANCEDIR***]
```

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 [***NEW CONFIG***] [***TEMPORARY NEW CONFIG***] \
```

```
-p immutable=false
```

Replace *[***NEW CONFIG***]* with the name of the instancedir you just deleted and *[***TEMPORARY NEW CONFIG***]* with the name of the temporary config created earlier in this procedure.

For example:

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

4. Delete the temporary config:

```
solrctl config --delete [***TEMPORARY NEW CONFIG***]
```

For example:

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

5. Reload the affected collection:

```
solrctl collection --reload [***COLLECTION NAME***]
```

Replace *[***COLLECTION NAME***]* with the name of the collection you want to reload.

For example:

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

Using custom JAR files with Search

Search supports custom plug-in code. You can load classes into JAR files and then configure Search to find these files.

About this task

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 Cloudera 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.

Procedure

1. Copy your custom JAR file 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.
   For example:

   ```
   <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
   For example:

   ```
   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. 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.

   ```
   http://example.com:8983/solr/admin/collections?action=RELOAD&name=collec
   tion1
   ```

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.

What to do next

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

Retrieving the clusterstate.json file

You can access the clusterstate.json file for Solr monitoring purposes through the Solr webUI.

Procedure

1. Go to the Solr webUI.
2. Select Cloud Tree .
3. From the tree view select /clusterstate.json.

Managing collections in 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.

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 *SchemaXml*.

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.

Collections are managed using the solrctl commandline utility.

Related Concepts

Managing collection configuration using configs or instance directories

Related Information

solrctl Reference

SchemaXml

Creating a Solr collection

Learn how to create a collection so that you can start indexing data with Solr.

Before you begin

- If you have enabled Ranger for authorization, you must have Solr Admin permission to be able to create collections.
- Before you can create a Solr collection you need to generate a collection configuration using either a config or an instance directory.

About this task

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.

Procedure

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
```

   This is automatically set on hosts with a Solr Server or Gateway role in Cloudera Manager.

3. Create a new collection using the following command:

```
solrctl collection --create [***COLLECTION NAME***] -s [***NUMBER OF
 SHARDS***] -c [***COLLECTION CONFIGURATION***]
```

   where

*[***COLLECTION NAME***]*	User-defined name of the collection. **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.
*[***NUMBER OF SHARDS***]*	The number of shards you want to split your collection into.
*[***COLLECTION CONFIGURATION***]*	The name of an existing collection configuration.

   For example:

```
solrctl collection --create logs -s 3 -c logs_config
```

Related Tasks

Generating collection configuration using configs

Generating Solr collection configuration using instance directories

Viewing existing collections

Learn how you can list existing Solr collections.

You can view existing collections using the solrctl collection --list command.

Deleting all documents in a 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.

Before you begin

If you have enabled Ranger for authorization, you must have Solr Admin permission to be able to delete documents in a collection.

Procedure

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
```

Deleting a collection

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

Before you begin

If you have enabled Ranger for authorization, you must have Solr Admin permission to be able to delete collections.

About this task

Procedure

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 [***COLLECTION NAME***]
```

   Replace *[***COLLECTION NAME***]* with the name of the collection you want to delete.

Updating the schema in a collection

It is sometimes necessary to change the underlaying schema behind a Solr collection. Find out how to do it for collections configured through instance directories and configs.

If your collection was configured using an instance directory, you can download the instance directory, edit sche ma.xml, then re-upload it to ZooKeeper. For instructions, see Generating Solr collection configuration using instance directories on page 9.

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.

Creating a replica of an existing shard

You can create additional replicas of existing shards using the solrctl utility. Replicating shards boosts query throughput and prevents data loss.

Procedure

To create additional replicas of existing shards, use the following command:

```
solrctl core --create [***NEW CORE***] -p collection=[***COLLECTION NAME***]
 \
-p shard=[***SHARD TO REPLICATE***]
```

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

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

Migrating Solr replicas

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

About this task

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 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:

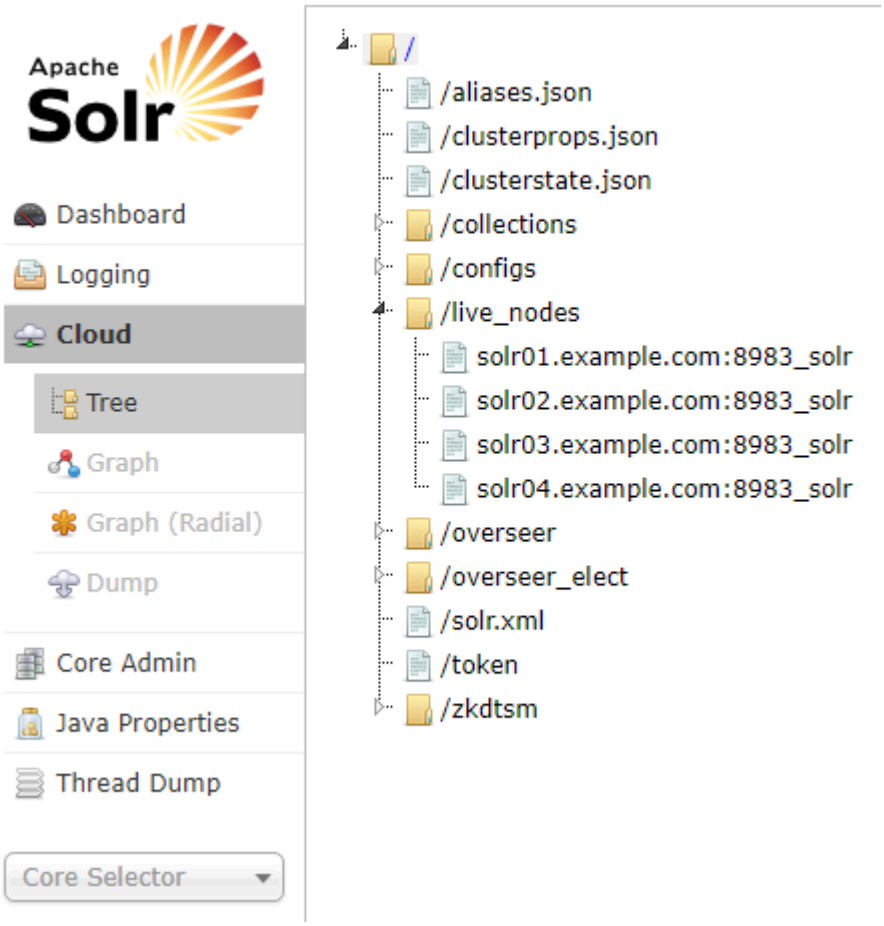
- Host names:
  - Origin: solr01.example.com.
  - Destination: solr02.example.com.
- Collection name: email

- 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.

Procedure

1. 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 Tree live_nodes .

   The Solr Administration User Interface, including live_nodes, appears.



   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 the clus terstate.json file.

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=DELETEREPLI
CA&collection=email&shard=shard1&replica=core_node2
```

The `DELETEREPLICA` call removes the datadir.

Related Information

Collections API in Apache Solr Reference Guide

Backing up a collection from HDFS

You can back up Solr collections to your local cluster or a remote cluster using the solrctl utility to minimize data loss caused by accidental or malicious administrative actions. Learn how to create, prepare, and export the Solr collection snapshot to create a backup of the Solr collection.

Before you begin

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=[***TRUST_STORE_PASSWORD***]"
```

Procedure

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

```
solrctl collection --create-snap
shot [***USER_DEFINED_NAME_OF_THE_SNAPSHOT***] -
c [***NAME_OF_THE_COLLECTION_TO_BE_BACKED_UP***]
```

For example, for a collection named tweets the command:

```
solrctl collection --create-snapshot tweets-$(date +%Y%m%d%H%M) -c tweets
```

creates the snapshot tweets-202103281043.

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.

The destination HDFS directory path (*[***DESTINATION_DIRECTORY***]*, 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.

```
solrctl collection --prepare-snapshot-ex
port [***NAME_OF_THE_SNAPSHOT_TO_BE_EXPORTED***] -
c [***COLLECTION_NAME***] -d [***DESTINATION_DIRECTORY***]
```

For example:

```
hdfs dfs -mkdir -p /path/to/backup-staging/tweets-202103281043
```

```
hdfs dfs -chown :solr /path/to/backup-staging/tweets-202103281043
solrctl collection --prepare-snapshot-export tweets-202103281043 -c twe
ets \
-d /path/to/backup-staging/tweets-202103281043
```

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-snap
shot [***NAME_OF_THE_SNAPSHOT_TO_BE_EXPORTED***] -
s [***SOURCE_DIRECTORY***] -
d [***PROTOCOL***]://[***NAMENODE***]:[***PORT***]/[***DESTINATION_DIRECTORY***]
```

   For example:

   For HDFS

   HDFS protocol:

```
solrctl collection --export-snapshot tweets-202103281043 -s /path/to/bac
kup-staging/tweets-202103281043 \
-d hdfs://nn01.example.com:8020/path/to/backups
```

   For webHDFS

   WebHDFS protocol:

```
solrctl collection --export-snapshot tweets-202103281043 -s /path/to/bac
kup-staging/tweets-202103281043 \
-d webhdfs://nn01.example.com:20101/path/to/backups
```

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

```
solrctl collection --export-snap
shot [***NAME_OF_THE_SNAPSHOT_TO_BE_EXPORTED***] -
c [***COLLECTION_NAME***] -d [***DESTINATION_DIRECTORY***]
```

   For example:

```
solrctl collection --export-snapshot tweets-202103281043 -c tweets -d /p
ath/to/backups/
```

4. Delete the snapshot after exporting:

```
solrctl collection --delete-snap
shot [***NAME_OF_THE_SNAPSHOT_TO_BE_DELETED***] -c [***COLLECTION_NAME***]
```

   For example:

```
solrctl collection --delete-snapshot tweets-202103281043 -c tweets
```

Related Information

solrctl Reference

Backing up a collection from local file system

Back up Solr collections to a shared file system to minimize data loss caused by accidental or malicious administrative actions. Learn how to create backup of a collection from the local file system (FS).

About this task

If you use local FS to store backups, each Solr host stores its backup directory locally, that is, server X contains the backup directory snapshots.shard1, server Y contains snapshots.shard2 and you need to copy those to a shared location in order to be able to restore them later. Because of this, Cloudera recommends to target backups to a shared file system, even if your Solr collection uses local FS.

Before you begin

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***]
```

Procedure

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

   ```
   solrctl collection --create-snap
   shot [***USER_DEFINED_NAME_OF_THE_SNAPSHOT***] -
   c [***NAME_OF_THE_COLLECTION_TO_BE_BACKED_UP***]
   ```

   This step is optional. You can back up this snapshot by specifying the *[***USER_DEFINED_NAME_OF_THE_SNAPSHOT***]* as the value of commitName parameter. If you do not create and specify a snapshot, the backup exports the index state corresponding to the current latest finished commit.

   For example, for a collection named tweets the command:

   ```
   solrctl collection --create-snapshot tweets-$(date +%Y%m%d%H%M) -c tweets
   ```

   creates the snapshot tweets-202103281043.
2. Create the backup. The destination directory must exist and be writable by the Solr superuser (solr by default).
   - To back up a snapshot, use the following command:

     ```
     curl -k --negotiate -u : 'http://[***HOST***]:[***PORT***]/solr/
     admin/collections?action=BACKUP&name=[***BACKUP_NAME***]&commitN
     ```

```
ame=[***SNAPSHOT_NAME***]&collection=[***COLLECTION_NAME***]&locatio
n=***BACKUP_LOCATION***'
```

For example:

```
curl -k --negotiate -u : 'http://host1.example.com:8983/solr/admin/colle
ctions?action=BACKUP&name=mybackup&commitName=tweets-202103281043&collec
tion=tweets&location=/tmp'
```

The example URL targets one (any one) of the Solr servers and creates a backup of the entire collection.

- To back up the current state of the index:

```
curl -k --negotiate -u : 'http://[***HOST***]:[***PORT***]/solr/admin/
collections?action=BACKUP&name=[***BACKUP_NAME***]&collection=tweets&l
ocation=/tmp'
```

For example:

```
curl -k --negotiate -u : 'http://host1.example.com:8983/solr/admin/colle
ctions?action=BACKUP&name=mybackup&collection=tweets&location=/tmp'
```

The example URL targets one (any one) of the Solr servers and creates a backup of the entire collection.

[HOST***]***

is a host name or IP address valid in your environment

[PORT***]***

is the port where you can access your Solr instance

[BACKUP_LOCATION***]***

specifies the directory (for example, /tmp) of the backup target defined in solr.xml where the backup is to be stored. If you have defined a HDFS target backup repository, the backup is stored on HDFS at *[***BACKUP_LOCATION***]*

[BACKUP_NAME***]***

specifies the name of the backup - the backup is created in the subdirectory *[***BACKUP_NAME***]* of the backup repository directory *[***BACKUP_LOCATION***]*.

[SNAPSHOT_NAME***]***

is the name of the snapshot you want to back up

[COLLECTION_NAME***]***

specifies the collection that you want to back up

Tip: To use a specific repository as a backup target, use the repository parameter.

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
<lst name="responseHeader"><int name="status">0</int><int name="QTime">363
6</int></lst>
</response>
```

After completing a backup, the data is stored in the standard backup format:

3. To check the backup files, run the following command:

```
hdfs dfs -ls /tmp/mybackup
```

```
Found 4 items
```

```
-rw-rw-rw-   2 solr supergroup        181 2021-01-13 21:33 /tmp/mybackup/b
ackup.properties
drwxrwxrwx   - solr supergroup          0 2021-01-13 21:33 /tmp/mybackup/
snapshot.shard1
drwxrwxrwx   - solr supergroup          0 2021-01-13 21:33 /tmp/mybackup/s
napshot.shard2
drwxrwxrwx   - solr supergroup          0 2021-01-13 21:33 /tmp/mybackup/
zk_backup
```

4. Delete the snapshot after exporting:

```
solrctl collection --delete-snap
shot [***NAME_OF_THE_SNAPSHOT_TO_BE_DELETED***] -c [***COLLECTION_NAME***]
```

For example:

```
solrctl collection --delete-snapshot tweets-202103281043 -c tweets
```

Related Tasks

Defining a backup target in solr.xml

Related Information

Backup/Restore Storage Repositories

BACKUP: Backup Collection

Restoring a collection

You can restore a Solr collection from a backup stored on either a remote cluster or the local cluster using the solrctl utility. You must pass a unique request identifier as part of the restore command in the solrctl utility while initiating the restore operation for tracking the process.

Before you begin

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=[***TRUST_STORE_PASSWORD***]"
```

Procedure

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 distc
   p [***PROTOCOL***]://[***NAMENODE***]:[***PORT***]/[***PATH/TO/
   BACKUP***] [***/PATH/TO/RESTORE-STAGING***]
   ```

   For example:

   For HDFS

   HDFS protocol:

   ```
   hadoop distcp hdfs://nn01.example.com:8020/path/to/backups/tweet
   s-202103281043 /path/to/restore-staging
   ```

   For webHDFS

   WebHDFS protocol:

   ```
   hadoop distcp webhdfs://nn01.example.com:20101/path/to/backups/t
   weets-202103281043 /path/to/restore-staging
   ```

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

   ```
   solrctl collection --restore [***NAME_OF_THE_RESTORED_COLLECTION***] -
   l [***BACKUP_LOCATION***] -b [***NAME_OF_THE_SNAPSHOT_TO_BE_RESTORED***] -
   i [***REQUEST_ID***]
   ```

   Make sure that you use a unique *[***REQUEST_ID***]* each time you run this command.

   Note:

   Statuses of historic job runs are stored in ZooKeeper and can be retrieved using the solrctl collection --req uest-status *[***REQUEST_ID***]* command. The number of async call responses stored in a cluster is limited to 10,000.

   Status information can be removed from ZooKeeper using the DELETESTATUS API call.

   For example:

   ```
   solrctl collection --restore tweets -l /path/to/restore-staging -b tweet
   s-202103281043 -i restore-tweets
   ```

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

   ```
   solrctl collection --request-status [***REQUEST_ID***]
   ```

   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="responseHea
   der"> <int name="status"> 0</int> <int name="QTime"> 1</int> </lst> \
   ```

```
<lst name="status"> <str name="state"> completed</str> <str name="msg"> fo
und 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 *[***REQUEST_ID***]* does not exist.

Related Information

solrctl Reference

Defining a backup target in solr.xml

If you want to to define or modify a backup target, you can do it by downloading, editing, and reuploading the solr.xml file from ZooKeeper.

About this task

The solr.xml file of your Solr installation, which is stored in ZooKeeper, can define a backup target repository, and depending on your installation it likely has a default target pointing to HDFS.

HDFS as a backup target is still fine even if your Solr collection uses a local file system (FS) / NRTCachingDirectoryFactory, so even with local FS collections you can store your backups on HDFS.

Similarly to this, other repositories like a LocalFileSystemRepository can also be defined in the solr.xml if you want to store the backups on a location other than HDFS.

Important: If you use a local FS to store backups, each Solr host stores its backup directory locally. That is, server X contains the backup directory snapshots.shard1, server Y contains snapshots.shard2 and you need to copy them to a shared location in order to be able to restore them later. Because of this, Cloudera recommends you to target backups to a shared file system.

If the solr.xml does not have a backup repository at all, it defaults to the local FS repository.

Note: If you use a HDFS backup repository, the backup also works if the Solr servers are located on nodes which do not have HDFS data node roles, they just need to have a HDFS client (gateway) role.

Procedure

1. To define or modify a backup target, download the solr.xml file from ZooKeeper using the following solrctl command:

```
solrctl cluster --get-solrxml solr.xml
```

2. Edit the contents of the solr.xml file.
   This is an example of defining a HDFS backup target in the solr.xml file:

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

3. Reupload the modified solr.xml file:

```
solrctl cluster --put-solrxml solr.xml
```