

Managing Cloudera Search

Date published: 2019-11-19

Date modified:



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

Managing Cloudera Search.....	4
Creating Collections.....	4
Using Custom JAR Files with Search.....	4
Cloudera Search Configuration Files.....	6
Managing Configuration Using Configs or Instance Directories.....	6
Managing Configs.....	7
Managing Instance Directories.....	7
Securing Configs with ZooKeeper ACLs and Ranger.....	8
Config Templates.....	8
Updating the Schema in a Solr Collection.....	9
Managing Collections in Cloudera Search.....	9
Generating Collection Configuration.....	9
Creating a Solr Collection.....	10
Viewing Existing Solr Collections.....	11
Deleting All Documents in a Solr Collection.....	11
Backing Up and Restoring Solr Collections.....	11
Deleting a Solr Collection.....	11
Example solrctl Usage.....	12
Using solrctl with an HTTP proxy.....	12
Creating Replicas of Existing Shards.....	12
Converting Instance Directories to Configs.....	13
Migrating Solr Replicas.....	13
Backing Up and Restoring Cloudera Search.....	16
Backing Up a Solr Collection.....	16
Restoring a Solr Collection.....	18
Cloudera Search Backup and Restore Command Reference.....	19
solrctl Reference.....	22

Managing Cloudera Search

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

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

A typical deployment workflow with `solrctl` consists of:

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

For a comparison of configs and instance directories, see [Managing Configuration Using Configs or Instance Directories](#) on page 6.

Creating Collections

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

```
solrctl collection --create <COLLECTION_NAME> -s <SHARD_COUNT>
```

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

To verify that the collection is active, go to `http://SEARCH01.EXAMPLE.COM:8983/solr/<COLLECTION_NAME>/select?q=%3A*&wt=json&indent=true` in a browser. For example, for the collection weblogs, the URL is `http://SEARCH01.EXAMPLE.COM:8983/solr/weblogs/select?q=%3A*&wt=json&indent=true`. Replace `SEARCH01.EXAMPLE.COM` with the hostname of one of the Solr server hosts.

You can also view the SolrCloud topology using the URL `http://SEARCH01.EXAMPLE.COM:8983/solr/#/~cloud`.

For more information on completing additional collection management tasks, see [Managing Cloudera Search](#).

Using Custom JAR Files with Search

About this task

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

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

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

Procedure

1. Place your custom JAR in the same location on all hosts in your cluster.
2. For all collections where custom JARs will be used, modify `solrconfig.xml` to include references to the new JAR files. These directives can include explicit or relative references and can use wildcards. In the `solrconfig.xml` file, add `<lib>` directives to indicate the JAR file locations or `<path>` directives for specific jar files.

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

or

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

or

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

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

```
solrctl instancedir --update NAME PATH
```

- *NAME* specifies the *INSTANCEDIR* associated with the collection using `solrctl INSTANCEDIR --create`.
- *PATH* specifies the directory containing the collection's configuration files.

For example:

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

5. For all collections in which custom JARs will be used, use `RELOAD` to refresh information. 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=collection1
```

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`

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

Managing Configuration Using Configs or Instance Directories

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

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

Table 2: Config and Instance Directory Comparison

Attribute	Config	Instance Directory
Security	<ul style="list-style-type: none"> In a Kerberos-enabled cluster, the ZooKeeper znodes associated with configurations created using the <code>solrctl config</code> command automatically have proper ZooKeeper ACLs. 	<ul style="list-style-type: none"> No ZooKeeper security support. Any user can create, delete, or modify an instance dir directly in ZooKeeper. Because <code>instancedir</code> 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 <code>ConfigSets</code> API.	Manually edited locally and re-uploaded directly to ZooKeeper using <code>solrctl</code> utility.

Attribute	Config	Instance Directory
Template support	<ul style="list-style-type: none"> Several predefined templates are available. These can be used as the basis for creating additional configs. Additional templates can be created by creating configs that are immutable. Mutable configs that use a managed schema can only be modified using the Schema API as opposed to being manually edited. As a result, configs are less flexible, but they are also less error-prone than instance directories. 	One standard template.

Managing Configs

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

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

```
solrctl config [--create <NAME> <BASECONFIG> [-p <NAME>=<VALUE>] ...]
               [--delete <NAME>]
```

- `--create <NAME> <BASECONFIG>` : Creates a new config based on an existing config. The config is created with the specified `<NAME>`, using `<BASECONFIG>` as the template. For more information about config templates, see [Config Templates](#) on page 8.
- `-p <NAME>=<VALUE>` : Overrides a `<BASECONFIG>` setting. The only config property that you can override is `immutable`, so the possible options are `-p immutable=true` and `-p immutable=false`. If you are copying an immutable config, such as a template, use `-p immutable=false` to make sure that you can edit the new config.
- `--delete <NAME>` : Deletes the specified config. You cannot delete an immutable config without accessing ZooKeeper directly as the `solr` super user.

Managing Instance Directories

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

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

The `solrctl instancedir` command syntax is as follows:

```
solrctl instancedir [--generate <PATH> [-schemaless]]
                   [--create <NAME> <PATH>]
                   [--update <NAME> <PATH>]
                   [--get <NAME> <PATH>]
                   [--delete <NAME>]
                   [--list]
```

- `--generate <PATH>` : Generates an instance directory template on the local filesystem at `<PATH>`. The configuration files are located in the `conf` subdirectory under `<PATH>`.
 - `-schemaless`: Generates a schemaless instance directory template.
- `--create <NAME> <PATH>` : Uploads a copy of the instance directory from `<PATH>` on the local filesystem to ZooKeeper. If an instance directory with the specified `<NAME>` already exists, this command fails. Use `--update` to modify existing instance directories.

- `--update <NAME> <PATH>` : Overwrites an existing instance directory in ZooKeeper using the specified files on the local filesystem. This command is analogous to first running `--delete <NAME>` followed by `--create <NAME> <PATH>` .
- `--get <NAME> <PATH>` : Downloads the specified instance directory from ZooKeeper to the specified path on the local filesystem. You can then edit the configuration and then re-upload it using `--update`.
- `--delete <NAME>` : Deletes the specified instance directory from ZooKeeper.
- `--list`: Lists existing instance directories as well as configs created by the `solrctl config` command.

Securing Configs with ZooKeeper ACLs and Ranger

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

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):

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

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

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

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

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

3. Set the `ZKCLI_JVM_FLAGS` environment variable:

```
export ZKCLI_JVM_FLAGS="-Djava.security.auth.login.config=/PATH/TO/
JAAS.CONF \
-DzkACLProvider=org.apache.solr.common.cloud.SaslZkACLProvider \
-Droot.logger=INFO,console"
```

4. Authenticate as the `solr` user:

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

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

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

```
/opt/cloudera/parcels/CDH/lib/solr/bin/zkcli.sh -zkh
ost ZK01.EXAMPLE.COM:2181 -cmd updateacls /solr
```

Replace `ZK01.EXAMPLE.COM` with the hostname of a ZooKeeper server.

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

Config Templates

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

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

Table 3: Available Config Templates and Attributes

Template Name	Supports Schema API	Uses Schemaless Solr
managedTemplate	■	
schemalessTemplate	■	■



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 newConfig managedTemplate -p immutable=false
```

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

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

Updating the Schema in a Solr Collection

If your collection was configured using an instance directory, you can download the instance directory, edit schema.xml, then re-upload it to ZooKeeper. For instructions, see [Managing Instance Directories](#) on page 7.

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

Managing Collections in Cloudera Search

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

Collections are managed using the solrctl command. For a reference to the solrctl commands and options, see [solrctl Reference](#).

Generating Collection Configuration

To start using Solr and indexing data, you must configure a collection to hold the index.

A collection requires the following configuration files:

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

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

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

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

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

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

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

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

```
solrctl instancedir --list
```

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



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

Creating a Solr Collection

Before you begin

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

About this task



Note: Although it is not currently 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. Generate configuration files for the collection:

- Using Configs:

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

- Using Instance Directories:

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

For more information on configs and instance directories, see [Managing Configuration Using Configs or Instance Directories](#) on page 6.

3. Create a new collection using the specified configuration:

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

Viewing Existing Solr Collections

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

Deleting All Documents in a Solr Collection

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.

About this task

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.

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
```

Backing Up and Restoring Solr Collections

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

For more information, see [Backing Up and Restoring Cloudera Search](#) on page 16.

Deleting a Solr Collection

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

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

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 logs
```

Example solrctl Usage

This topic includes some examples of:

- Configuration changes that may be required for [solrctl](#) to function as desired.
- Common tasks completed with solrctl.

Using solrctl with an HTTP proxy

About this task

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

Procedure

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

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

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

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

Creating Replicas of Existing Shards

Procedure

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

```
solrctl core --create <NEWCORE> -p collection=<NAME> \
-p shard=<SHARD_TO_REPLICATE>
```

For example, to create a new replica of 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
```

Converting Instance Directories to Configs

About this task

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

Procedure

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

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

2. Delete the existing instance directory. For example:

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

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

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

4. Delete the temporary config:

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

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

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

Migrating Solr Replicas

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

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

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

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

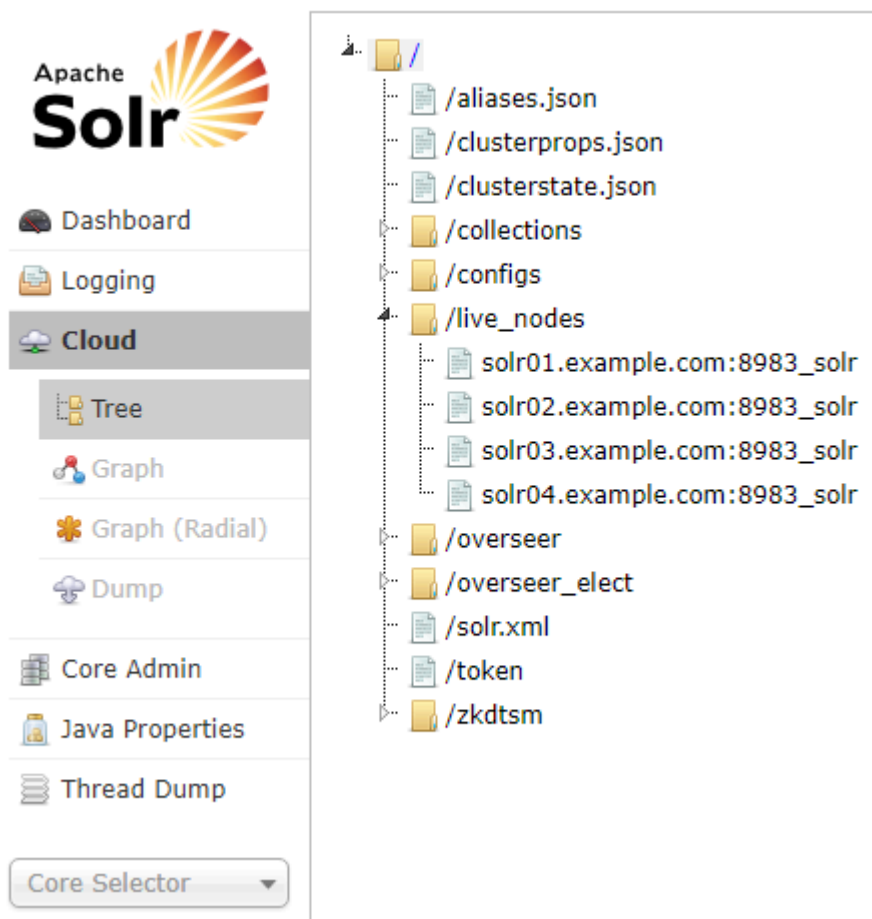
Cloudera recommends using API calls to create and unload cores. Do not use the Cloudera Manager Admin Console or the Solr Admin UI for these tasks.

This procedure uses the following names:

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

To migrate a replica to a new host:

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



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

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

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

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



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

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

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

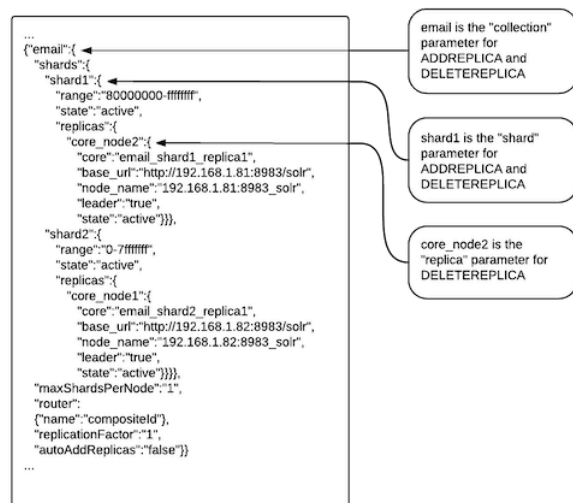
Accessing this URI returns an XML document that contains content about replication progress. A snippet of the XML content might appear as follows:

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

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

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

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



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

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

The DELETEREPLICA call removes the datadir.

Backing Up and Restoring Cloudera Search



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

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

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

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

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

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

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

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

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



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

Backing Up a Solr Collection

About this task

Use the following procedure to back up a Solr collection. For more information on the commands used, see [Cloudera Search Backup and Restore Command Reference](#) on page 19.

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=TRUSTSTOREPASSWORD"
```

Procedure

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

```
solrctl collection --create-snapshot <SNAPSHOTNAME> -c <COLLECTIONNAME>
```

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

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

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

```
solrctl collection --prepare-snapshot-export <SNAPSHOTNAME> -
c <COLLECTIONNAME> -d <DESTDIR>
```

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

For example:

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

3. Export the snapshot. This step uses the DistCp utility to back up the collection metadata as well as the corresponding index files. The destination directory must exist and be writable by the Solr superuser (solr by default).

To export the snapshot to a remote cluster, run the following command:

```
solrctl collection --export-snapshot <SNAPSHOTNAME> -s <SOURCEDIR> -
d <PROTOCOL>://<NAMENODE>:<PORT>/<DESTDIR>
```

For example:

- HDFS protocol:

```
solrctl collection --export-snapshot tweets-201803281043 -s /path/to/backup-staging/tweets-201803281043 \
```

```
-d hdfs://nn01.example.com:8020/path/to/backups
```

- WebHDFS protocol:

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

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

```
solrctl collection --export-snapshot <SNAPSHOTNAME> -c <COLLECTIONNAME> -d <DESTDIR>
```

For example:

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

4. Delete the snapshot:

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

For example:

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

Restoring a Solr Collection

About this task

Use the following procedure to restore a Solr collection. For more information on the commands used, see [Cloudera Search Backup and Restore Command Reference](#) on page 19.

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=TRUSTSTOREPASSWORD"
```

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 distcp <PROTOCOL>://<NAMENODE>:<PORT>/PATH/TO/BACKUP /path/to/re
store-staging
```

For example:

- HDFS protocol:

```
hadoop distcp hdfs://nn01.example.com:8020/path/to/backups/tweets-201803
281043 /path/to/restore-staging
```

- WebHDFS protocol:

```
hadoop distcp webhdfs://nn01.example.com:20101/path/to/backups/tweets-20
1803281043 /path/to/restore-staging
```

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

```
solrctl collection --restore <RESTORECOLLECTIONNAME> -l <BACKUPLOCATION> -
b <SNAPSHOTNAME> -i <REQUESTID>
```

Make sure that you use a unique `<REQUESTID>` each time you run this command.

For example:

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

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

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

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

```
solrctl collection --request-status restore-tweets
<?xml version="1.0" encoding="UTF-8"?> <response> <lst name="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 `<REQUESTID>` does not exist.

Cloudera Search Backup and Restore Command Reference

Use the following commands to create snapshots, back up, and restore Solr collections.



Note:

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

```
-jaas /PATH/TO/JAAS.CONF
```

**Note:**

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

```
export ZKCLI_JVM_FLAGS="-Djavax.net.ssl.trustStore=/PATH/TO/TRUSTSTORE  
-Djavax.net.ssl.trustStorePassword=TRUSTSTOREPASSWORD"
```

Create a snapshot

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

Description: Creates a named snapshot for the specified collection.

Delete a snapshot

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

Description: Deletes the specified snapshot for the specified collection.

Describe a snapshot

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

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

List all snapshots

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

Description: Lists all snapshots for the specified collection.

Prepare snapshot for export to a remote cluster

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

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

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

If you are running the snapshot export command on a remote cluster, specify the HDFS protocol (such as WebHDFS or HFTP) to be used for accessing the Lucene index files corresponding to the snapshot on the source cluster. This configuration is driven by the `-p` option which expects a fully qualified URI for the root filesystem on the source cluster, for example `webhdfs://namenode.example.com:20101/`.

Export snapshot to local cluster

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

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

Export snapshot to remote cluster

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

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

Make sure that you [prepare the snapshot for export](#) before exporting it to a remote cluster.

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

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

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

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

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

Restore from a local snapshot

Command: `solrctl collection --restore <RESTORECOLLECTIONNAME> -l <BACKUPLOCATION> -b <SNAPSHOTNAME> -i <REQUESTID>`

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

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

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

Because the restore operation can take a long time to complete depending on the size of the exported snapshot, it is run asynchronously. The `-i` configuration parameter specifies a unique identifier for tracking operation. For more information, see [Check the status of an operation](#) on page 21.

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

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

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

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

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

Check the status of an operation

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

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

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

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

solrctl Reference

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

For some examples of common tasks using solrctl, see [Example solrctl Usage](#).

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

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

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

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

For collection configuration, users have the option of interacting directly with ZooKeeper using the instancedir option or using the Solr ConfigSets API using the config option. For more information, see [Managing Configuration Using Configs or Instance Directories](#).

Command Syntax

The general solrctl command syntax is:

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

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

solrctl Options

If used, the following options must precede commands:

Table 4: solrctl options

Option	Description
--solr <SOLR_URI>	Directs solrctl to a SolrCloud web API available at the specified URI. This option is required for hosts running outside of SolrCloud. A sample URI might be: http://search01.example.com:8983/solr.
--zk <ZK_ENSEMBLE>	Directs solrctl to a particular ZooKeeper quorum. This option is required for hosts running outside of SolrCloud. For example: zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181/solr. Output from solrctl commands that use the --zk option is sent to /dev/null, so no results are displayed.


Option	Description
--jaas <i>/PATH/TO/JAAS.CONF</i>	Used to identify a JAAS configuration that specifies the principal with permissions to modify Solr metadata. The principal is typically solr@EXAMPLE.COM. In Kerberos-enabled environments where ZooKeeper ACLs protect Solr metadata, you must use this parameter when modifying metadata.
--help	Prints help.
--quiet	Suppresses most solrctl messages.
--debug	Prints errors to stdout.
--trace	Prints the executed commands to stdout.

Commands and Arguments


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

The solrctl core command affects a single SolrCloud host.

Table 5: solrctl commands and arguments

Command	Argument	Description
init		Initializes a SolrCloud deployment. Must be run before starting solr-server daemons for the first time. The command has a built-in security check that prevents it from running on a deployment that has already been initialized.
	[--force]	Allows you to re-initialize an already initialized SolrCloud deployment. Use this command cautiously because it erases all SolrCloud deployment state information from ZooKeeper, including all configuration files. It does not delete collections.
instancedir		<p>Manipulates instance directories.</p> <p> Note: solrctl instancedir commands talk directly to ZooKeeper. Starting from CDH version 6.0, the ZooKeeper node that stores the instance configuration files (e.g.: /solr/configs) is only writable by the user 'solr'. This means that in a Kerberized environment you will have to create a Java Authentication and Authorization Service (JAAS) configuration file and a keytab for the solr user, and specifying this jaas.conf file when using solrctl. For example: solrctl --jaas jaas.conf instancedir --create myInstanceDir /tmp/myInstanceDir.</p> <p>Cloudera recommends using the solrctl config --upload command instead, as it does not require a keytab of the solr user (a simple kinit is enough) and additionally it can utilize Ranger to control who is authorized to modify instance configurations.</p>

Command	Argument	Description
	--generate <PATH> [-schemaless]	--generate <PATH>: Generates an instance directory template on the local filesystem at <PATH>. The configuration files are located in the conf subdirectory under <PATH>. If used with the -schemaless option, it generates a schemaless instance directory template. For more information on schemaless support, see Schemaless Mode Overview and Best Practices .
	--create <NAME> <PATH>	Uploads a copy of the instance directory from <PATH> on the local filesystem to ZooKeeper. If an instance directory with the specified <NAME> already exists, this command fails. Use --update to modify existing instance directories.
	--update <NAME> <PATH>	Overwrites an existing instance directory in ZooKeeper using the specified files on the local filesystem. This command is analogous to first running --delete <NAME> followed by --create <NAME> <PATH>.
	--get <NAME> <PATH>	Downloads the specified instance directory from ZooKeeper to the specified path on the local filesystem. You can then edit the configuration and then re-upload it using --update.
	--delete <NAME>	Deletes the specified instance directory from ZooKeeper.
	--list	Lists existing instance directories, including configs created by the solrctl config command.
config		Manipulates configs.
	--create name <BASECONFIG> [-p <NAME>=<VALUE>]	Creates a new config based on an existing config. The config is created with the specified NAME, using <BASECONFIG> as the template. For more information about config templates, see Config Templates . The -p name=value option overrides a <BASECONFIG> setting. The only config property that you can override is immutable, so the possible options are -p immutable=true and -p immutable=false. If you are copying an immutable config, such as a template, use -p immutable=false to make sure that you can edit the new config.

Command	Argument	Description
	<code>--upload NAME PATH</code>	<p>Uploads a new configset in zip file format. You cannot use this option to update an existing config. The script will ask you to delete the existing version before allowing you to upload the new one.</p> <p>The <i>PATH</i> argument of this command needs to point to the local directory containing the instance configuration (meaning it has a <i>conf</i> subdirectory and the config files like <i>conf/solrconfig.xml</i>). This can also be an instance configuration directory generated using <code>solrctl instancedir --generate NAME</code> or downloaded using <code>solrctl instancedir --get NAME PATH</code>. The underlying Solr API requires a .zip archive to be created, this is automatically performed by the command.</p> <p> Note: This function needs the zip binary to be installed and executable by the user running the <code>solrctl config --upload</code> command.</p>
	<code>--delete name</code>	Deletes the specified config. You cannot delete an immutable config without accessing ZooKeeper directly as the solr super user.
collection		Manipulates collections.
	<code>--create <NAME> -s <NUMSHARDS> [-a] [-c <CONFIGNAME>] [-r <REPLICATIONFACTOR>] [-m <MAXSHARDSPERHOST>] [-n <HOSTLIST>]</code>	<p>Creates a new collection with <i><NUMSHARDS></i> shards.</p> <p>The <i>-a</i> option enables automatic addition of replicas (<code>autoAddReplicas=true</code>) if machines hosting existing shards become unavailable.</p> <p>The collection uses the specified <i><CONFIGNAME></i> for its configuration set, and the specified <i><REPLICATIONFACTOR></i> controls the number of replicas for the collection. Keep in mind that this replication factor is on top of the HDFS replication factor.</p> <p>The maximum shards per host is determined by <i><MAXSHARDSPERHOST></i>, and you can specify specific hosts for the collection in the <i><HOSTLIST></i>.</p> <p>The only required parameters are <i><NAME></i> and <i>-s <NUMSHARDS></i>. If <i>-c <CONFIGNAME></i> is not provided, it is assumed to be the same as the name of the collection.</p>
	<code>--delete <NAME></code>	Deletes a collection.
	<code>--reload <NAME></code>	Reloads a collection.
	<code>--stat <NAME></code>	Outputs SolrCloud specific run-time information for a collection.
	<code>--deletedocs <NAME></code>	Purges all indexed documents from a collection.
	<code>--list</code>	Lists all collections.
	<code>--*-snapshot</code>	The snapshot-related commands are covered in detail in Cloudera Search Backup and Restore Command Reference .
core		Manipulates cores.

Command	Argument	Description
	--create <NAME> [-p <NAME>=<VALUE>]	Creates a new core. The core is configured using <NAME>=<VALUE> pairs. For more information about configuration options, see Solr documentation
	--reload <NAME>	Reloads a core.
	--unload <NAME>	Unloads a core.
	--status <NAME>	Prints status of a core.
cluster		Manages cluster configuration.
	--get-solrxml <FILE>	Downloads the cluster configuration file solr.xml from ZooKeeper to the local system.
	--put-solrxml <FILE>	Uploads the specified file to ZooKeeper as the cluster configuration file solr.xml.
	--set-property <NAME> <VALUE>	Sets property names and values. For example, to configure a cluster to use TLS/SSL: <pre>solrctl cluster --set-property urlScheme https</pre>
	--remove-property <NAME>	Removes the specified property.
	--get-clusterstate <FILE>	Downloads the clusterstate.json file from ZooKeeper to the local system.