Cloudera Runtime 7.1.2

Managing Cloudera Search

Date published: 2019-11-19
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Managing Collection Configuration Using Configs or Instance Directories

Configs and instance directories are named configuration sets for Solr collections.

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 1: 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.
Template support	• Several predefined templates are available. These can be used as the basis for creating additional configs. Additional templates can be created by creating configs that are immutable. • Mutable configs that use a managed schema can only be modified using the Schema API as opposed to being manually edited. As a result, configs are less flexible, but they are also less error-prone than instance directories.	One standard template.

Managing Configs

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 ConfigSets API.

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 5.
  - -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.

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 2: 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
```

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.

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.

Note:

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

The solrctl instancedir command syntax is as follows:

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

- --generate <*path*> : Generates an instance directory template on the local filesystem at <*path*>. The configuration files are located in the conf subdirectory under <*path*>.
  - -schemaless: Generates a schemaless instance directory template. For more information on schemaless support, see Schemaless Mode Overview and Best Practices.
- --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.

Generate a Collection Configuration Using Configs

A collection configuration is a necessary prerequisite of creating a collection. Learn how to do it using configs.

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 <configName> <templateName> -p immutable=false
   ```

   For available

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

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

Generate a Collection Configuration Using Instance Directories

A collection configurations is a necessary prerequisite of creating a collection. Learn how to do it using instance directories.

About this task

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

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

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 instancedir --create [collection_name] $HOME/solr_configs
```

```
solrctl instancedir --create 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 should return weblogs.

Convert Instance Directories to Configs

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.

```
solrctl config --create <new_config_temp> <existing_instancedir> \
-p immutable=false
```

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> <new_config_temp> \
-p immutable=false
```

For example:

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

4. Delete the temporary config:

```
solrctl config --delete <new_config_temp>
```

For example:

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

5. Reload the affected collection:

```
solrctl collection --reload <collection>
```

For example:

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

Cloudera Search Configuration Files

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

Table 3: 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 solrctl cluster command. For information about the solr.xml file, see Solr Configuration Files and Solr Cores and solr.xml in the Solr documentation.
solrconfig.xml	Each collection in Solr uses a solrconfig.xml file, stored in ZooKeeper, to control collection behavior. For information about the solrconfig.xml file, see Solr Configuration Files and Configuring solrconfig.xml in the Solr documentation.
managed-schema or schema.xml	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 schema.xml file. These files, also stored in ZooKeeper and assigned to a collection, define the schema for the documents you are indexing. For example, they specify which fields to index, the expected data type for each field, the default field to query when the field is unspecified, and so on. For information about managed-schema and schema.xml, see Schema Factory Definition in SolrConfig in the Solr documentation.
core.properties	Unlike other configuration files, this file is stored in the local filesystem rather than ZooKeeper, and is used for core discovery. For more information on this process and the structure of the file, see Defining core.properties in the Solr documentation.
Additional files	Any additional files referenced in the xml files, for example, custom JAR files.

Using Custom JAR Files with Cloudera Search

Search supports custom plug-in code. You 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. 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 solrconf ig.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=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

Related Information

solrctl Reference

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.

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 utility.

Related Concepts

Managing Collection Configuration Using Configs or Instance Directories

Related Information

solrctl Reference

Create 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.
- 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 <collectionName> -s <numShards> -c <collectionConfName>
   ```

   where

<collectionName>	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.
<numShards>	The number of shards you want to split your collection into.
<collectionConfName>	The name of an existing collection configuration.

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

Related Tasks

Generate a Collection Configuration Using Configs

Generate a Collection Configuration Using Instance Directories

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 17.

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
```

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 6.

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.

Create a Replica of an Existing Shard

You can create additional replicas of existing shards using the solrctl utility.

Procedure

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

```
solrctl core --create <newCore> -p collection=<collectionName> \
-p shard=<shardToReplicate>
```

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
```

Migrate 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.

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 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.

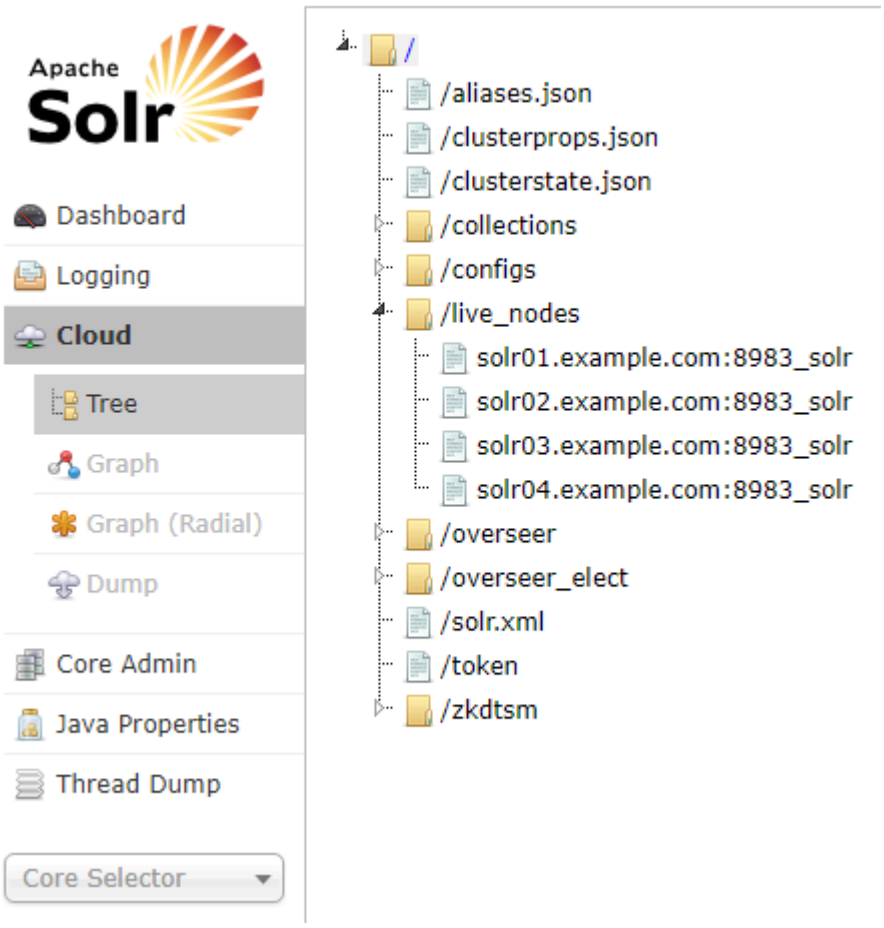
This procedure uses the following names:

- 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.

Procedure

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 clusters tate.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/replicatio
n?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.

Back Up a Solr Collection

Learn about backing up a Solr collection using the solrctl utility.

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 twe
ets \
-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/bac
kup-staging/tweets-201803281043 \
-d hdfs://nn01.example.com:8020/path/to/backups
```

- WebHDFS protocol:

```
solrctl collection --export-snapshot tweets-201803281043 -s /path/to/bac
kup-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 /p
ath/to/backups/
```

4. Delete the snapshot:

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

For example:

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

Related Information

solrctl Reference

Restore a Solr Collection

Learn about restoring a Solr collection using the solrctl utility.

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.

Related Information

solrctl Reference