Cloudera Runtime 7.1.5

Data Protection

Date published: 2020-02-20
Date modified: 2020-11-30

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Data protection....4
Backing up HDFS metadata....4
Introduction to HDFS metadata files and directories....4
Back up HDFS metadata....8
Configuring HDFS trash....11
Trash behavior with HDFS Transparent Encryption enabled....11
Enabling and disabling trash....12
Setting the trash interval....12
Using HDFS snapshots for data protection....12
Considerations for working with HDFS snapshots....13
Enable snapshot creation on a directory....13
Create snapshots on a directory....13
Recover data from a snapshot....14
Options to determine differences between contents of snapshots....14
CLI commands to perform snapshot operations....15
Managing snapshot policies using Cloudera Manager....16
Enable and disable snapshot creation using Cloudera Manager....17
Create snapshots using Cloudera Manager....17
Delete snapshots using Cloudera Manager....18
Preventing inadvertent deletion of directories....18

Data protection

You can ensure data protection by preventing accidental deletion of files and backing up HDFS metadata.

Backing up HDFS metadata

HDFS metadata represents the structure and attributes of HDFS directories and files in a tree. You can back up the metadata without affecting NameNode availability.

Introduction to HDFS metadata files and directories

HDFS metadata represents the structure of HDFS directories and files in a tree. It also includes the various attributes of directories and files, such as ownership, permissions, quotas, and replication factor.

Files and directories

Persistence of HDFS metadata is implemented using fsimage file and edits files.

Attention:

Do not attempt to modify metadata directories or files. Unexpected modifications can cause HDFS downtime, or even permanent data loss. This information is provided for educational purposes only.

Persistence of HDFS metadata broadly consist of two categories of files:

fsimage

Contains the complete state of the file system at a point in time. Every file system modification is assigned a unique, monotonically increasing transaction ID. An fsimage file represents the file system state after all modifications up to a specific transaction ID.

edits file

Contains a log that lists each file system change (file creation, deletion or modification) that was made after the most recent fsimage.

Checkpointing is the process of merging the content of the most recent fsimage, with all edits applied after that fsimage is merged, to create a new fsimage. Checkpointing is triggered automatically by configuration policies or manually by HDFS administration commands.

NameNodes

Understand the HDFS metadata directory details taken from a NameNode.

The following example shows an HDFS metadata directory taken from a NameNode. This shows the output of running the tree command on the metadata directory, which is configured by setting *dfs.namenode.name.dir* in hdfs-site.xml.

```
data/dfs/name
### current#
### VERSION#
### edits_0000000000000000001-0000000000000000007
# ### edits_0000000000000000008-0000000000000000015
# ### edits_0000000000000000016-0000000000000000022
# ### edits_0000000000000000023-0000000000000000029
# ### edits_0000000000000000030-0000000000000000030
# ### edits_0000000000000000031-0000000000000000031
# ### edits_inprogress_0000000000000000032
# ### fsimage_0000000000000000030
# ### fsimage_0000000000000000030.md5
# ### fsimage_0000000000000000031
# ### fsimage_0000000000000000031.md5
```

```
# ### seen_txid
### in_use.lock
```

In this example, the same directory has been used for both fsimage and edits. Alternative configuration options are available that allow separating fsimage and edits into different directories. Each file within this directory serves a specific purpose in the overall scheme of metadata persistence:

VERSION

Text file that contains the following elements:

layoutVersion

Version of the HDFS metadata format. When you add new features that require a change to the metadata format, you change this number. An HDFS upgrade is required when the current HDFS software uses a layout version that is newer than the current one.

namespaceID/clusterID/blockpoolID

Unique identifiers of an HDFS cluster. These identifiers are used to prevent DataNodes from registering accidentally with an incorrect NameNode that is part of a different cluster. These identifiers also are particularly important in a federated deployment. Within a federated deployment, there are multiple NameNodes working independently. Each NameNode serves a unique portion of the namespace (namespaceID) and manages a unique set of blocks (blockpoolID). The clusterID ties the whole cluster together as a single logical unit. This structure is the same across all nodes in the cluster.

storageType

Always NAME_NODE for the NameNode, and never JOURNAL_NODE.

cTime

Creation time of file system state. This field is updated during HDFS upgrades.

edits_start transaction ID-end transaction ID

Finalized and unmodifiable edit log segments. Each of these files contains all of the edit log transactions in the range defined by the file name. In an High Availability deployment, the standby can only read up through the finalized log segments. The standby NameNode is not up-to-date with the current edit log in progress. When an HA failover happens, the failover finalizes the current log segment so that it is completely caught up before switching to active.

fsimage_end transaction ID

Contains the complete metadata image up through . Each fsimage file also has a corresponding .md5 file containing a MD5 checksum, which HDFS uses to guard against disk corruption.

seen_txid

Contains the last transaction ID of the last checkpoint (merge of edits into an fsimage) or edit log roll (finalization of current edits_inprogress and creation of a new one). This is not the last transaction ID accepted by the NameNode. The file is not updated on every transaction, only on a checkpoint or an edit log roll. The purpose of this file is to try to identify if edits are missing during startup. It is possible to configure the NameNode to use separate directories for fsimage and edits files. If the edits directory accidentally gets deleted, then all transactions since the last checkpoint would go away, and the NameNode starts up using just fsimage at an old state. To guard against this, NameNode startup also checks seen_txid to verify that it can load transactions at least up through that number. It aborts startup if it cannot verify the load transactions.

in_use.lock

Lock file held by the NameNode process, used to prevent multiple NameNode processes from starting up and concurrently modifying the directory.

JournalNodes

Understand the components of the JournalNode metadata directory.

In an HA deployment, edits are logged to a separate set of daemons called JournalNodes. A JournalNode's metadata directory is configured by setting *dfs.journalnode.edits.dir*. The JournalNode contains a VERSION file, multiple edits__ files and an edits_inprogress_, just like the NameNode. The JournalNode does not have fsimage files or seen _txid. In addition, it contains several other files relevant to the HA implementation. These files help prevent a split-brain scenario, in which multiple NameNodes could think they are active and all try to write edits.

committed-txid

Tracks last transaction ID committed by a NameNode.

last-promised-epoch

Contains the "epoch," which is a monotonically increasing number. When a new NameNode, starts as active, it increments the epoch and presents it in calls to the JournalNode. This scheme is the NameNode's way of claiming that it is active and requests from another NameNode, presenting a lower epoch, must be ignored.

last-writer-epoch

Contains the epoch number associated with the writer who last actually wrote a transaction.

paxos

Specifies the directory that temporary files used in the implementation of the Paxos distributed consensus protocol. This directory often appears as empty.

DataNodes

Although DataNodes do not contain metadata about the directories and files stored in an HDFS cluster, they do contain a small amount of metadata about the DataNode itself and its relationship to a cluster.

This shows the output of running the tree command on the DataNode's directory, configured by setting *dfs.datanode.data.dir in hdfs-site.xml*.

```
data/dfs/data/
### current
# ### BP-1079595417-192.168.2.45-1412613236271
# # ### current
# # # ### VERSION
# # # ### finalized
# # # # ### subdir0# # # # ### subdir1
# # # # ### blk_1073741825
# # # # ### blk_1073741825_1001.meta
# # # ### lazyPersist
# # # ### rbw
# # ### dncp_block_verification.log.curr
# # ### dncp_block_verification.log.prev
# # ### tmp
# ### VERSION
### in_use.lock
```

The purpose of these files are as follows:

BP-random integer-NameNode-IP address-creation time

Top level directory for datanodes. The naming convention for this directory is significant and constitutes a form of cluster metadata. The name is a block pool ID. "BP" stands for "block pool," the abstraction that collects a set of blocks belonging to a single namespace. In the case of a federated deployment, there are multiple "BP" sub-directories, one for each block pool. The remaining components form a unique ID: a random integer, followed by the IP address of the NameNode that created the block pool, followed by creation time.

VERSION

Text file containing multiple properties, such as layoutVersion, clusterId and cTime, which is much like the NameNode and JournalNode. There is a VERSION file tracked for the entire DataNode as well as a separate VERSION file in each block pool sub-directory.

In addition to the properties already discussed earlier, the DataNode's VERSION files also contain:

storageType

storageType field is set to DATA_NODE.

blockpoolID

Repeats the block pool ID information encoded into the sub-directory name.

finalized/rbw

Both finalized and rbw contain a directory structure for block storage. This holds numerous block files, which contain HDFS file data and the corresponding .meta files, which contain checksum information. rbw stands for "replica being written". This area contains blocks that are still being written to by an HDFS client. The finalized sub-directory contains blocks that are not being written to by a client and have been completed.

lazyPersist

HDFS is incorporating a new feature to support writing transient data to memory, followed by lazy persistence to disk in the background. If this feature is in use, then a lazyPersist sub-directory is present and used for lazy persistence of in-memory blocks to disk. We'll cover this exciting new feature in greater detail in a future blog post.

scanner.cursor

File to which the "cursor state" is saved.

The DataNode runs a block scanner which periodically does checksum verification of each block file on disk. This scanner maintains a "cursor," representing the last block to be scanned in each block pool slice on the volume, and called the "cursor state."

in_use.lock

Lock file held by the DataNode process, used to prevent multiple DataNode processes from starting up and concurrently modifying the directory.

HDFS commands for metadata files and directories

You can use HDFS commands to manipulate metadata files and directories.

hdfs namenode

Automatically saves a new checkpoint at NameNode startup. As stated earlier, checkpointing is the process of merging any outstanding edit logs with the latest fsimage, saving the full state to a new fsimage file, and rolling edits. Rolling edits means finalizing the current edits_inprogress and starting a new one.

hdfs dfsadmin -safemode enter
hdfs dfsadmin -saveNamespace

Saves a new checkpoint (similar to restarting NameNode) while the NameNode process remains running. The NameNode must be in safe mode, and all attempted write activity fails while this command runs.

hdfs dfsadmin -rollEdits

Manually rolls edits. Safe mode is not required.

This can be useful if a standby NameNode is lagging behind the active NameNode and you want it to get caught up more quickly. The standby NameNode can only read finalized edit log segments, not the current in progress edits file.

hdfs dfsadmin -fetchImage

Downloads the latest fsimage from the NameNode. This can be helpful for a remote backup type of scenario.

Configuration properties

Use the NameNode and DataNode properties to configure the NameNode and DataNodes.

dfs.namenode.name.dir

Specifies where on the local filesystem the DFS name node stores the name table (fsimage). If this is a comma-delimited list of directories then the name table is replicated in all of the directories, for redundancy.

dfs.namenode.edits.dir

Specifies where on the local filesystem the DFS name node stores the transaction (edits) file. If this is a comma-delimited list of directories, the transaction file is replicated in all of the directories, for redundancy. The default value is set to the same value as dfs.namenode.name.dir.

dfs.namenode.checkpoint.period

Specifies the number of seconds between two periodic checkpoints.

dfs.namenode.checkpoint.txns

The standby creates a checkpoint of the namespace every dfs.namenode.checkpoint.txns transactions, regardless of whether dfs.namenode.checkpoint.period has expired.

dfs.namenode.checkpoint.check.period

Specifies how frequently to query for the number of un-checkpointed transactions.

dfs.namenode.num.checkpoints.retained

Specifies the number of image checkpoint files to be retained in storage directories. All edit logs necessary to recover an up-to-date namespace from the oldest retained checkpoint are also retained.

dfs.namenode.num.extra.edits.retained

Specifies the number of extra transactions which are retained beyond what is minimally necessary for a NN restart. This can be useful for audit purposes or for an HA setup where a remote Standby Node might have been offline and need to have a longer backlog of retained edits to start again.

dfs.namenode.edit.log.autoroll.multiplier.threshold

Specifies when an active namenode rolls its own edit log. The actual threshold (in number of edits) is determined by multiplying this value by *dfs.namenode.checkpoint.txns*. This prevents extremely large edit files from accumulating on the active namenode, which can cause timeouts during namenode start-up and pose an administrative hassle. This behavior is intended as a fail-safe for when the standby fails to roll the edit log by the normal checkpoint threshold.

dfs.namenode.edit.log.autoroll.check.interval.ms

Specifies the time in milliseconds that an active namenode checks if it needs to roll its edit log.

dfs.datanode.data.dir

Determines where on the local filesystem an DFS data node should store its blocks. If this is a comma-delimited list of directories, then data is stored in all named directories, typically on different devices. Directories that do not exist are ignored. Heterogeneous storage allows specifying that each directory resides on a different type of storage: DISK, SSD, ARCHIVE or RAM_DISK.

Back up HDFS metadata

You can back up HDFS metadata without taking down either HDFS or the NameNodes.

Prepare to back up the HDFS metadata

Regardless of the solution, a full, up-to-date continuous backup of the namespace is not possible. Some of the most recent data is always lost. HDFS is not an Online Transaction Processing (OLTP) system. Most data can be easily recreated if you re-run Extract, Transform, Load (ETL) or processing jobs.

- Normal NameNode failures are handled by the Standby NameNode. Doing so creates a safety-net for the very unlikely case where both master NameNodes fail.
- In the case of both NameNode failures, you can start the NameNode service with the most recent image of the namespace.
- Name Nodes maintain the namespace as follows:
  - Standby NameNodes keep a namespace image in memory based on edits available in a storage ensemble in Journal Nodes.
  - Standby NameNodes make a namespace checkpoint and saves an fsimage_* to disk.
  - Standby NameNodes transfer the fsimage to the primary NameNodes using HTTP.

Both NameNodes write fsimages to disk in the following sequence:

- NameNodes write the namespace to a file fsimage.ckpt_* on disk.
- NameNodes creates an fsimage_*.md5 file.
- NameNodes moves the file fsimage.ckpt_* to fsimage_.*.

The process by which both NameNodes write fsimages to disk ensures that:

- The most recent namespace image on disk in an fsimage_* file is on the standby NameNode.
- Any fsimage_* file on disk is finalized and does not receive updates.

Backing up NameNode metadata

You must back up the VERSION file and then back up the NameNode metadata.

Procedure

1. Make a single backup of the VERSION file.

   This does not need to be backed up regularly as it does not change, but it is important since it contains the clusterID, along with other details.
2. Use the following command to back up the NameNode metadata.

   It automatically determines the active NameNode, retrieves the current fsimage, and places it in the defined backup_dir.

   ```
   hdfs dfsadmin -fetchImage backup_dir
   ```

Results

On startup, the NameNode process reads the fsimage file and commits it to memory. If the JournalNodes are up and running, and there are edit files present, any edits newer than the fsimage are also applied. If the JournalNodes are unavailable, it is possible to lose any data transferred in the interim.

Back up HDFS metadata using Cloudera Manager

HDFS metadata backups can be used to restore a NameNode when both NameNode roles have failed. In addition, Cloudera recommends backing up HDFS metadata before a major upgrade.

About this task

This backup method requires you to shut down the cluster.

Procedure

1. Note the active NameNode.

2. Stop the cluster.

   It is particularly important that the NameNode role process is not running so that you can make a consistent backup.
3. Go to the HDFS service.
4. Click the Configuration tab.
5. In the Search field, search for "NameNode Data Directories" and note the value.
6. On the active NameNode host, back up the directory listed in the NameNode Data Directories property. If a file with the extension lock exists in the NameNode data directory, the NameNode most likely is still running. Repeat the steps, beginning with shutting down the NameNode role.

   If more than one is listed, make a backup of one directory, because each directory is a complete copy. For example, if the NameNode data directory is /data/dfs/nn, do the following as root:

   ```
   # cd /data/dfs/nn
   # tar -cvf /root/nn_backup_data.tar .
   ```

   You should see output like this:

   ```
   /dfs/nn/current
   ./
   ./VERSION
   ./edits_0000000000000000001-0000000000000008777
   ./edits_0000000000000008778-0000000000000009337
   ./edits_0000000000000009338-0000000000000009897
   ./edits_0000000000000009898-0000000000000010463
   ./edits_0000000000000010464-0000000000000011023
   <snip>
   ./edits_0000000000000063396-0000000000000063958
   ./edits_0000000000000063959-0000000000000064522
   ./edits_0000000000000064523-0000000000000065091
   ./edits_0000000000000065092-0000000000000065648
   ./edits_inprogress_0000000000000065649
   ./fsimage_0000000000000065091
   ./fsimage_0000000000000065091.md5
   ./fsimage_0000000000000065648
   ./fsimage_0000000000000065648.md5
   ./seen_txid
   ```

Restoring NameNode metadata

If both the NameNode and the secondary NameNode were to suddenly go offline, you can restore the NameNode.

Procedure

1. Add a new host to your Hadoop cluster.
2. Add the NameNode role to the host. Make sure it has the same hostname as the original NameNode.
3. Create a directory path for the NameNode name.dir (for example, /dfs/nn/current), ensuring that the permissions are set correctly.
4. Copy the VERSION and latest fsimage file to the /dfs/nn/current directory.
5. Run the following command to create the md5 file for the fsimage.

   ```
   md5sum fsimage > fsimage.md5
   ```

6. Start the NameNode process.

Restore HDFS metadata from a backup using Cloudera Manager

When both the NameNode hosts have failed, you can use Cloudera Manager to restore HDFS metadata.

Procedure

1. Remove the NameNode, JournalNode, and Failover Controller roles from the HDFS service.
2. Add the host on which the NameNode role will run.
3. Create the NameNode data directory, ensuring that the permissions, ownership, and group are set correctly.
4. Copy the backed up files to the NameNode data directory.
5. Add the NameNode role to the host.
6. Add the Secondary NameNode role to another host.
7. Enable high availability.

   If not all roles are started after the wizard completes, restart the HDFS service. Upon startup, the NameNode reads the fsimage file and loads it into memory. If the JournalNodes are up and running and there are edit files present, any edits newer than the fsimage are applied.

Perform a backup of the HDFS metadata

You can back up HDFS metadata without affecting the availability of NameNode.

Procedure

1. Make sure the Standby NameNode checkpoints the namespace to fsimage_ once per hour.
2. Deploy monitoring on both NameNodes to confirm that checkpoints are triggering regularly.

   This helps reduce the amount of missing transactions in the event that you need to restore from a backup containing only fsimage files without subsequent edit logs. It is good practice to monitor this because edit logs that are large in size and without checkpoints can cause long delays after a NameNode restart while it replays those transactions.
3. Back up the most recent “fsimage_*” and “fsimage_*.md5” from the standby NameNode periodically.

   Try to keep the latest version of the file on another machine in the cluster.
4. Back up the VERSION file from the standby NameNode.

Configuring HDFS trash

The Hadoop trash feature helps prevent accidental deletion of files and directories.

When you delete a file in HDFS, the file is not immediately expelled from HDFS. Deleted files are first moved to the /user/<username>/.Trash/Current directory, with their original filesystem path being preserved. After a user-configurable period of time (fs.trash.interval), a process known as trash checkpointing renames the Current directory to the current timestamp, that is, /user/<username>/.Trash/<timestamp>. The checkpointing process also checks the rest of the .Trash directory for any existing timestamp directories and removes them from HDFS permanently. You can restore files and directories in the trash simply by moving them to a location outside the .Trash directory.

Important: The trash feature is enabled by default. Cloudera recommends that you enable it on all production clusters.

Trash behavior with HDFS Transparent Encryption enabled

You can delete files or directories that are part of an HDFS encryption zone. Moving and renaming files or directories is an important part of trash handling in HDFS.

HDFS creates a local .Trash directory every time a new encryption zone is created. For example, when you create an encryption zone, /enc_zone, HDFS will also create the /enc_zone/.Trash/ sub-directory. Files deleted from enc_zone are moved to /enc_zone/.Trash/<username>/Current/. After the checkpoint, the Current directory is renamed to the current timestamp, /enc_zone/.Trash/<username>/<timestamp>.

If you delete the entire encryption zone, it will be moved to the .Trash directory under the user's home directory, /users/<username>/.Trash/Current/enc_zone. Trash checkpointing will occur only after the entire zone has been moved to /users/<username>/.Trash. However, if the user's home directory is already part of an encryption zone, then

attempting to delete an encryption zone will fail because you cannot move or rename directories across encryption zones.

If the trash directory is deleted by mistake, create the .Trash directory using the -provisionTrash option as follows:

```
hdfs crypto -provisionTrash -path /enc_zone
```

If required, you can use the following commands to manually create the .Trash directory within an encryption zone. Make sure you run the commands as an admin user.

```
hdfs dfs -mkdir /enc_zone/.Trash
hdfs dfs -chmod 1777 /enc_zone/.Trash
```

Enabling and disabling trash

You can use Cloudera Manager to enable and disable HDFS trash.

Procedure

1. Go to the HDFS service.
2. Click the Configurations tab.
3. Select Scope Gateway .
4. Select or clear the Use Trash checkbox.

   To apply this configuration property to other role groups as needed, edit the value for the appropriate role group.
5. Restart the cluster and deploy the cluster client configuration.

Setting the trash interval

You can use Cloudera Manager to specify the time period after which a trash checkpoint directory is deleted.

Procedure

1. Go to the HDFS service.
2. Click the Configurations tab.
3. Select Scope NameNode .
4. Specify the Filesystem Trash Interval property, which controls the number of minutes after which a trash checkpoint directory is deleted and the number of minutes between trash checkpoints.

   For example, to enable trash so that deleted files are deleted after 24 hours, set the value of the Filesystem Trash Interval property to 1440.

   Note: The trash interval is measured from the point at which the files are moved to trash, not from the last time the files were modified.

   To apply this configuration property to other role groups as needed, edit the value for the appropriate role group.
5. Restart all NameNodes.

Using HDFS snapshots for data protection

HDFS snapshots enable you to capture point-in-time copies of the file system and protect your important data against user or application errors. Cloudera recommends that you take snapshots of specified subtrees on the file system.

Using snapshots to protect data is efficient because of the following reasons:

- Snapshot creation is instantaneous regardless of the size and depth of the directory subtree.
- Snapshots capture the block list and file size for a specified subtree. Snapshots do not create extra copies of blocks on the file system.

You can either use the command-line interface or Cloudera Manager to manage HDFS snapshots.

Considerations for working with HDFS snapshots

You can create snapshots only for directories that allow the creation of snapshots. If a directory already contains snapshots, you cannot delete or rename the directory unless you remove all the snapshots.

You must consider the following when working with HDFS snapshots:

- You must enable snapshot creation on a particular directory before creating snapshots on that directory. Such a directory is termed as a *snapshottable* directory. However, you cannot create snapshots on a directory if its corresponding child or parent directory is already enabled for snapshot creation.
- You cannot delete or rename a directory that contains snapshots. You must first remove all the snapshots before attempting the delete or rename operation.
- For a snapshottable directory, its path component .snapshot can be used to access the snapshots.

  For example, consider the directory /foo that is enabled for snapshot creation. For the directory /foo with a snapshot snap1, the path /foo/.snapshot/snap1 refers to the snapshot of /foo.
- You can enable or disable snapshot creation on a particular directory only if you have the superuser privilege.
- On a snapshottable directory; you can create, delete, or rename snapshots. These operations require either the superuser privilege or the owner access to the directory. In addition, you can list directories that have snapshot creation enabled or view differences between contents of snapshots.
- You cannot rename a snapshot outside the snapshottable directory.

Enable snapshot creation on a directory

You must enable snapshot creation on a directory before creating snapshots on that directory. If the snapshot creation is enabled, the directory becomes *snapshottable*.

About this task

- You can perform this task only if you have the superuser privilege.
- You cannot enable snapshot creation on any directory if its parent or child directory is already enabled for snapshot creation.

Procedure

Run the hdfs dfsadmin command with the -allowSnapshot option and specify the directory on which you want to enable snapshot creation.
The following example shows how you can enable snapshot creation for the directory /data/dir1:

```
hdfs dfsadmin -allowSnapshot /data/dir1
```

If snapshot creation is successfully enabled on the specified directory, a confirmation message appears.

```
Allowing snapshot on /data/dir1 succeeded
```

Related Information

Enable and disable snapshot creation using Cloudera Manager

Create snapshots on a directory

You can create snapshots on a specified directory and protect your important data.

Before you begin

You must have enabled snapshot creation.

About this task

Only a user with either of the following privileges can perform this task:

- The owner privilege to the directory on which to create the snapshots
- The superuser privilege

Procedure

Run the hdfs dfs command with the -createSnapshot option and specify the path to the directory on which you want to create snapshots.
The following example shows how you can create a snapshot snap1 on the directory /data/dir1:

```
hdfs dfs -createSnapshot /data/dir1 snap1
```

If snapshot creation is successfully enabled on the specified directory, a confirmation message appears.

```
Created snapshot /data/dir1/.snapshot/snap2
```

Note: You can also run the command without mentioning the snapshot name. In such a situation, the new snapshot has the time stamp of creation as its name. See the following example:

```
Created snapshot /data/dir1/.snapshot/s20180412-065533.159
```

Related Information

Create snapshots using Cloudera Manager
Delete snapshots using Cloudera Manager

Recover data from a snapshot

If data is erroneously removed from a directory for which snapshots are available, you can recover the lost data using snapshots. The snapshot ensures that the file blocks corresponding to the deleted files or directories are not removed from the file system. Only the metadata is modified to reflect the deletion.

About this task

You must have read access to the files or directories that you want to restore.

Procedure

Run the hdfs dfs command with the cp option to copy the deleted data from the snapshot to the destination directory.
The following example shows how you can recover a file imp_details.xls from a snapshot of the directory (/data/dir1) that contained the file:

```
hdfs dfs -cp /data/dir1/.snapshot/s20180412-065533.159/imp_details.xls /data
/dir1/
```

Related Information

Delete snapshots using Cloudera Manager

Options to determine differences between contents of snapshots

Run the hdfs snapshotDiff command for a report that lists the difference between the contents of two snapshots. Run the distcp diff command to determine the difference between contents of specified source and target snapshots, and use the command with the -update option to move the difference to a specified target directory.

Generating a report listing the difference between contents of two snapshots

Using the hdfs snapshotDiff between two snapshots on a specified directory path provides the list of changes to the directory. Consider the following example:

```
hdfs snapshotDiff /data/dir1 snap1 snap2
M .
- ./file1.csv
R ./file2.txt -> ./fileold.txt
+ ./filenew.txt
```

This example shows the following changes to the directory /data/dir1 after the creation of snap1 and before the creation of snap2:

Statement	Explanation
M .	The directory /data/dir1 is modified.
- ./file1.csv	The file file1.csv is deleted.
R ./file2.txt -> ./fileold.txt	The file file2.txt is renamed to fileold.txt.
+ ./filenew.txt	The file filenew.txt is added to the directory /data/dir1.

Moving the differences between the contents of two snapshots to a specified directory

Using the distcp diff command with the -update option on snapshots enables you to determine the difference between the contents of two snapshots and move the difference to a specified target directory. Consider the following example:

hadoop distcp -diff snap_old snap_new -update /data/source_dir /data/target_dir

The command in this example determines the changes between the snapshots snap_old and snap_new present in the source_dir directory, and updates the target_dirdirectory with the changes.

The following conditions must be satisfied for the content changes to be moved to /data/target_dir:

- Both /data/source_dir and /data/target_dir are distributed file system paths.
- The snapshots snap_old and snap_new are created for /data/source_dir such that snap_old is older than snap_new.
- The /data/target_dir path also contains snap_old. In addition, no changes are made to /data/target_dir after the creation of snap_old.

CLI commands to perform snapshot operations

As an administrator, you can enable or disable snapshot creation on a directory. These operations require the superuser privilege. As a user; you can create, delete, or rename snapshots on a directory that has snapshot creation enabled. These operations require either the superuser privilege or the owner privilege on the directory.

Administrator operations

The following table lists the snapshot-related administrator operations that you can on specified directories:

Operation	Command
Enable snapshot creation on a directory	hdfs dfsadmin -allowSnapshot <path>
Disable snapshot creation on a directory	hdfs dfsadmin -disallowSnapshot <path>

For more information about these commands, see https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsSnapshots.html#Administrator_Operations.

User operations

The following table lists the user operations that you can perform on snapshots:

Operation	Command
Create snapshots	hdfs dfs -createSnapshot <path> [<snapshotName>]
Delete snapshots	hdfs dfs -deleteSnapshot <path> <snapshotName>

Operation	Command
Rename snapshots	hdfs dfs -renameSnapshot <path> <oldName> <newName>
List directories on which snapshot creation is enabled (*snapshottable* directories)	hdfs lsSnapshottableDir
List snapshots on a snapshottable directory with their IDs and timestamp of creation	hdfs lsSnapshot
List differences between contents of snapshots	hdfs snapshotDiff <path> <fromSnapshot> <toSnapshot>

For more information about these commands, see https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsSnapshots.html#User_Operations.

Managing snapshot policies using Cloudera Manager

Cloudera Manager enables the creation of snapshot policies that define the directories or tables for which snapshots should be taken, the intervals at which snapshots should be taken, and the number of snapshots that should be kept for each snapshot interval.

For example, you can create a policy that takes both daily and weekly snapshots, and specify that seven daily snapshots and five weekly snapshots should be maintained.

Create a snapshot policy

You can use Cloudera Manager to create snapshot policies for a directory.

Before you begin

The directory for which you want to create a snapshot policy must be *snapshottable*.

Procedure

1. Select Backup Snapshot Policies in the left navigation bar.

   Existing snapshot policies are shown in a table on the Snapshot Policies page.
2. Click Create Snapshot Policy.

   The Create Snapshot Policy window displays.
3. From the drop-down list, select the service (HDFS or HBase) and cluster for which you want to create a policy.
4. Provide a name for the policy and, optionally, a description.
5. Specify the directories, namespaces or tables to include in the snapshot.

   Important: Do not take snapshots of the root directory.

   - For an HDFS service, select the paths of the directories to include in the snapshot. The drop-down list allows you to select only directories that are enabled for snapshot creation. If no directories are enabled for snapshot creation, a warning displays.

     Click ➕ to add a path and ➖ to remove a path.
   - For an HBase service, list the tables to include in your snapshot. You can use a Java regular expression to specify a set of tables. For example, finance.* match all tables with names starting with finance. You can also create a snapshot for all tables in a given namespace, using the {*namespace*}:.* syntax.
6. Specify the snapshot Schedule.

   You can schedule snapshots hourly, daily, weekly, monthly, or yearly, or any combination of those. Depending on the frequency you select, you can specify the time of day to take the snapshot, the day of the week, day of the month, or month of the year, and the number of snapshots to keep at each interval. Each time unit in the schedule

information is shared with the time units of larger granularity. That is, the minute value is shared by all the selected schedules, hour by all the schedules for which hour is applicable, and so on. For example, if you specify that hourly snapshots are taken at the half hour, and daily snapshots taken at the hour 20, the daily snapshot will occur at 20:30.

7. Specify whether Alerts should be generated for various state changes in the snapshot workflow.

   You can alert on failure, on start, on success, or when the snapshot workflow is aborted.

8. Click Save Policy.

   The new Policy displays on the Snapshot Policies page.

Edit or delete a snapshot policy

You can use Cloudera Manager to edit or delete existing snapshot policies.

Procedure

1. Select Backup Snapshot Policies in the left navigation bar.

   Existing snapshot policies are shown in a table on the Snapshot Policies page.

2. Click next to a policy and select Edit Configuration or Delete.

3. If you want to edit the selected policy, make the required changes and click Save Policy.

Enable and disable snapshot creation using Cloudera Manager

For snapshots to be created, HDFS directories must be enabled for snapshots. You cannot specify a directory as part of a snapshot policy unless it has been enabled for snapshots.

Procedure

1. From the Clusters tab, select the HDFS service.
2. Go to the File Browser tab.
3. Go to the directory you want to enable for snapshots.
4. In the File Browser, click the drop-down menu next to the full file path and select Enable Snapshots.

   Note: Once you enable snapshots for a directory, you cannot enable snapshots on any of its subdirectories. Snapshots can be taken only on directories that have snapshots enabled.

   Note: To disable snapshots for a directory that has snapshots enabled, use Disable Snapshots from the drop-down menu specified earlier. If snapshots of the directory exist, they must be deleted before snapshots can be disabled.

Related Information

Enable snapshot creation on a directory

Create snapshots using Cloudera Manager

You can use Cloudera Manager to create snapshots on a *snapshottable* directory.

Procedure

1. From the Clusters tab, select the HDFS service.
2. Go to the File Browser tab.
3. Go to the directory for which you want to create the snapshot.
4. Click the drop-down menu next to the full path name and select Take Snapshot.

   The Take Snapshot screen displays.

5. Enter a name for the snapshot.

6. Click OK.

   The Take Snapshot button is present, enabling an immediate snapshot of the directory.
7. To take a snapshot, click Take Snapshot, specify the name of the snapshot, and click Take Snapshot.

   The snapshot is added to the snapshot list. Any snapshots that have been taken are listed by the time at which they were taken, along with their names and a menu button.

Related Information

Create snapshots on a directory

Delete snapshots using Cloudera Manager

You can use Cloudera Manager select from a snapshot configured for a directory and delete it.

About this task

Procedure

1. From the Clusters tab, select the HDFS service.
2. Go to the File Browser tab.
3. Go to the directory with the snapshot you want to delete.
4. In the list of snapshots, locate the snapshot you want to delete and click .
5. Select Delete.

Related Information

Create snapshots on a directory

Recover data from a snapshot

Preventing inadvertent deletion of directories

You can prevent inadvertent deletion of important data from your HDFS cluster by marking specific directories as *protected*. Marking a directory as protected prevents its recursive deletion. However, this does not protect against graceful deletion of files under the directory. You can delete the files by moving them to trash.

Procedure

1. Go to the HDFS service.
2. Click the Configuration tab.
3. Set the Cluster-wide Advanced Configuration Snippet (Safety Valve) for core-site.xml property as specified:
   - Name: fs.protected.directories
   - Value: Specify a comma-separated list of the directories that you want to mark as *protected*; for example, /user, /data, and so on.
4. Enter a Reason for Change, and then click Save Changes to save the property changes.
5. Restart the cluster.

Note: By default, subdirectories present under directories (fs.protected.directories) are not protected. You must set the dfs.protected.subdirectories.enable parameter to true to protect subdirectories. For example, if you set the fs.protected.directories parameter to true for the parent directory testA, the subdirectory testB under testA(/testA/testB) can be deleted or renamed if dfs.protected.subdirectories.enable is set to false.