Cloudera Runtime 7.2.12

Using HBase Backup and Disaster Recovery

Date published: 2020-02-29
Date modified: 2022-11-25

https://docs.cloudera.com/

Legal Notice

Contents

HBase backup and disaster recovery strategies

Backup-and-restore is a standard set of operations for many databases. You must have an effective backup-and-restore strategy to ensure that you can recover data in case of data loss or failures.

HBase snapshot enables you to take a snapshot of a table without much impact on RegionServers, because snapshot, clone, and restore operations do not involve data copying. In addition, exporting a snapshot to another cluster has no impact on RegionServers.

If your data is already in an HBase cluster, replication is useful for getting the data into additional HBase clusters.

Configuring HBase snapshots

HBase snapshots enable you to take a point-in-time copy of a table with little performance impact on HBase. When you create and restore snapshots, it do not copy any data; therefore the impact is very low. In addition, you can export or copy snapshots from one cluster to another without significant impact on HBase because the table files are copied directly rather than reading table data from RegionServers.

About HBase snapshots

HBase snapshots allow you to clone a table without making data copies, and with minimal impact on RegionServers. Exporting the table to another cluster has little impact on the RegionServers.

Without using snapshots, the only way to backup or clone a table was to use the provided CopyTable or ExportTable tools. However, these methods have disadvantages:

- CopyTable and ExportTable can degrade RegionServer performance because they read the table data.
- Disabling the table means no reads or writes; this is usually unacceptable.

Use Cases

- Data migration
  - You can use HBase snapshots to migrate data from CDH 5.x or HDP 2.x to COD on CDP Public Cloud. Refer to the migration guide for more information.
- Recovery from user or application errors
  - Useful because it may be some time before the database administrator notices the error.

    Note:

    An administrator can create automation which takes new snapshots and deletes unneeded old snapshots.
  - The database administrator may want to save a snapshot before a major application upgrade or change.

    Note:

    Snapshots are not primarily used for system upgrade protection because they do not roll back binaries, and would not necessarily prevent bugs or errors in the system or the upgrade.
  - Recovery cases:
    - Roll back to the previous snapshot and merge in reverted data.
    - View previous snapshots and choose which snapshot to apply into production.

- Backup
  - Capture a copy of the database and store it outside HBase for disaster recovery.
  - Capture previous versions of data for compliance, regulation, and archiving.
  - Export from a snapshot on a live system provides a faster and simpler way of moving HBase data than the CopyTable and ExportTable.
- Audit or report view of data at a specific time
  - Capture monthly data for compliance.
  - Use for end-of-day/month/quarter reports.
- Application testing
  - Test schema or application changes on similar production data from a snapshot and then discard.

    For example:

    1. Take a snapshot.
    2. Create a new table from the snapshot content (schema and data)
    3. Manipulate the new table by changing the schema, adding and removing rows, and so on. The original table, the snapshot, and the new table remain independent of each other.
- Offload work
  - Capture, copy, and restore data to another site
  - Export data to another cluster

Zero-Copy Restore and Clone Table

From a snapshot, you can create a new table (clone_snapshot) or restore the original table (restore_snapshot). When cloning snapshots, a new table is created to reflect the state of the original table when the snapshot was taken. Restoring snapshots, however, reverts the original table state to that of when the snapshot was taken (any updates to the table after the snapshot was taken would be lost). These two operations do not involve data copies; instead, a link is created to point to the original hfiles.

Changes to a cloned or restored table do not affect the snapshot.

To clone a table to another cluster, you export the snapshot to the other cluster and then run the clone operation; see Exporting a snapshot to another cluster.

Storage Considerations

Because hfiles are immutable, a snapshot consists of a reference to the files in the table at the moment the snapshot is taken. No copies of the data are made during the snapshot operation, but copies may be made when a compaction or deletion is triggered. In this case, if a snapshot has a reference to the files to be removed, the files are moved to an archive folder, instead of being deleted. This allows the snapshot to be restored in full.

When a snapshot is deleted, all retained HFiles will be automatically deleted when the files are no longer needed.

Manage HBase snapshots using Cloudera Operational Database CLI

Data backup and restore operations are important for your Cloudera Operational Database (COD) database to support data recovery. COD CLI offers multiple commands that help you manage the HBase snapshots on your CDP environment.

Using the COD CLI, you can:

- Create a snapshot.
- View the list of saved snapshots currently maintained. These can include one-off immediate snapshots, as well as scheduled policy-based snapshots.
- Restore from a saved snapshot.
- List all the restore operations in your CDP environment.

- Delete snapshots.

Related Information

Notes about replication

Create a snapshot

You can use COD CLI to create snapshots of HBase tables. The `create-snapshot` command takes a snapshot of a table in HBase, exports the snapshot to the specified storage location and deletes the snapshot after the export is successful.

Command syntax

`cdp opdb create-snapshot --environment-name` *ENVIRONMENT_NAME* `--database-name` *DATABASE_NAME* `--table-name` *TABLE_NAME* `--snapshot-name` *SNAPSHOT_NAME* `--snapshot-location` *SNAPSHOT_LOCATION*

Parameters

--environment-name

The name of the CDP environment.

--database-name

The name of the operational database.

--table-name

The fully qualified name of the HBase table whose snapshot is to be taken.

--snapshot-name

The name of the new snapshot. The snapshot name is unique per database.

--snapshot-location

The name of the snapshot location URL on object store where the snapshot is to be stored.

```
cdp opdb create-snapshot --environment-name odx-wgel6j --database-name test
--table-name table_6uf7xt --snapshot-name s1 --snapshot-location s3a://cod-7
215/cod--yq3p370r6qro/backupdata
```

JSON output:

```
{
    "environmentName": "odx-wgel6j",
    "databaseName": "test",
    "status": "IN_PROGRESS",
    "creationTime": 0,
    "commandID": 1546336350
}
```

Related Information

COD CLI command reference GitHub repository

CDP CLI BETA command reference GitHub repository

List snapshots

You can list the snapshots created in your CDP environment against a COD database using the `list-snapshots` command. You can use different filters to list the snapshots, for example, table name, specific command ID, and time range.

Command syntax

`cdp opdb list-snapshots --environment-name` *ENVIRONMENT_NAME* `--database-name` *DATABASE_NAME* [`--table-name` *TABLE_NAME*] [`--command-id` *COMMAND_ID*] [`--from-creation-time` *FROM_CREATION_TIME*] [`--to-creation-time` *TO_CREATION_TIME*]

Parameters

--environment-name

The name of the CDP environment.

--database-name

The name of the operational database.

--table-name

The fully qualified name of the HBase table whose snapshots are to be listed.

--command-id

Command ID filter.

--from-creation-time

The starting snapshot creation time to search snapshots (inclusive).

--to-creation-time

The ending snapshot creation time to search snapshots (inclusive).

Without filters:

```
cdp opdb list-snapshots  --environment-name odx-wgel6j --database-name test
{
    "environmentName": "odx-wgel6j",
    "databaseName": "test",
    "snapshots": [
        {
            "tableName": "table_tob8vc",
            "snapshotName": "snapshot3_table_tob8vc",
            "creationTime": 1662014265011,
            "status": "SUCCESSFUL",
            "commandID": 1546336058,
            "snapshotLocation": "abfs://qedailynat-filesystem@q.dfs.core.win
dows.net/hbase"
        },
        {
            "tableName": "table_tob8vc",
            "snapshotName": "snapshot2_table_tob8vc",
            "creationTime": 1662014023725,
            "status": "SUCCESSFUL",
            "commandID": 1546336035,
            "snapshotLocation": "abfs://qedailynat-filesystem@q.dfs.core.win
dows.net/hbase"
        },
        {
            "tableName": "table_tob8vc",
            "snapshotName": "snapshot1_table_tob8vc",
            "creationTime": 1662013782370,
            "status": "DELETED",
            "commandID": 1546336077,
            "snapshotLocation": "abfs://qedailynat-filesystem@q.dfs.core.wi
ndows.net/hbase"
        },
        {
```

```
            "tableName": "table_lpirwy",
            "snapshotName": "snapshot_table_lpirwy",
            "creationTime": 1662013568312,
            "status": "DELETED",
            "commandID": 1546336004,
            "snapshotLocation": "abfs://qedailynat-filesystem@q.dfs.core.wi
ndows.net/hbase"
        }
]
}
```

With command ID filter:

```
cdp opdb list-snapshots  --environment-name odx-wgel6j --database-name test
--command-id 1546336385
{
    "environmentName": "odx-wgel6j",
    "databaseName": "test",
    "snapshots": [
        {
            "tableName": "table_6uf7xt",
            "snapshotName": "s2",
            "creationTime": 1662035967244,
            "status": "SUCCESSFUL",
            "commandID": 1546336385,
            "snapshotLocation": "abfs://qedailynat-filesystem@qedailynats
torageaccount.dfs.core.windows.net/cod-rtzupxh843ua/hbase/"
        }
    ]
}
```

With table name filter:

```
cdp opdb list-snapshots  --environment-name odx-wgel6j --database-name test
--table-name table_6uf7xt
{
    "environmentName": "odx-wgel6j",
    "databaseName": "test",
    "snapshots": [
        {
            "tableName": "table_6uf7xt",
            "snapshotName": "s2",
            "creationTime": 0,
            "status": "SUCCESSFUL",
            "commandID": 1546336385,
            "snapshotLocation": "abfs://qedailynat-filesystem@qedailynatsto
rageaccount.dfs.core.windows.net/cod-rtzupxh843ua/hbase/"
        }
    ]
}
```

Related Information

COD CLI command reference GitHub repository

CDP CLI BETA command reference GitHub repository

Restore a snapshot

You can use the `restore-snapshot` command to import a snapshot automatically from the snapshot location. This command restores a snapshot and deletes it afterward.

Command syntax

`cdp opdb restore-snapshot --environment-name` *ENVIRONMENT_NAME* `--database-name` *DATABASE_NAME* `--snapshot-name` *SNAPSHOT_NAME* `--target-environment-name` *TARGET_ENVIRONMENT_NAME* `--target-database-name` *TARGET_DATABASE_NAME*

Parameters

--environment-name

The name of the original CDP environment.

--database-name

The name of the original operational database.

--snapshot-name

The name of the snapshot to be restored.

--target-environment-name

The name of the target environment where the snapshot should be restored.

--target-database-name

The name of the target database where the snapshot should be restored.

```
cdp opdb restore-snapshot --environment-name odx-wgel6j --database-name test
 --snapshot-name s2 --target-environment-name odx-wgel6j --target-database-n
ame test
```

JSON output:

```
{
    "environmentName": "odx-wgel6j",
    "databaseName": "test",
    "snapshotName": "s2",
    "targetEnvironmentName": "odx-wgel6j",
    "targetDatabaseName": "test",
    "status": "IN_PROGRESS",
    "restoreTime": 1662036779279,
    "commandID": 1546336475
}
```

Related Information

COD CLI command reference GitHub repository

CDP CLI BETA command reference GitHub repository

List restored snapshots

You can use the `list-restore-snapshots` command to list all the restored snapshots in your CDP environment against a COD database. You can use various filters to list the snapshots, for example, snapshot name, target environment name where the snapshot is restored, target COD database, specific command ID, and time range.

Command syntax

`cdp opdb list-restore-snapshots --environment-name` *ENVIRONMENT_NAME* `--database-name` *DATABASE_NAME* [`--snapshot-name` *SNAPSHOT_NAME*] [`--target-environment-name` *TARGET_ENVIRONMENT_NAME*] [`--target-database-name` *TARGET_DATABASE_NAME*] [`--command-id` *COMMAND_ID*] [`--from-restore-time` *FROM_RESTORE_TIME*] [`--to-restore-time` *TO_RESTORE_TIME*]

Parameters

--environment-name

The name of the CDP environment.

--database-name

The name of the operational database.

--snapshot-name

The snapshot name to be restored.

--target-environment-name

The name of the target environment.

--target-database-name

The name of the target database.

--command-id

Command ID filter.

--from-restore-time

The starting snapshot restore time to search restore snapshots (inclusive).

--to-restore-time

The ending snapshot restore time to search restore snapshots (inclusive).

```
cdp opdb list-restore-snapshots --environment-name odx-wgel6j --database-nam
e test
```

JSON output:

```
{
    "environmentName": "odx-wgel6j",
    "databaseName": "test",
    "restoreSnapshots": [
        {
            "snapshotName": "snapshot2_table_tob8vc",
            "targetEnvironmentName": "odx-wgel6j",
            "targetDatabaseName": "test",
            "status": "IN_PROGRESS",
            "restoreTime": 1662014023725,
            "commandID": 1546336085
        }
    ]
}
```

Using snapshot name filter:

```
cdp opdb list-restore-snapshots    --environment-name odx-wgel6j --database-
name test --snapshot-name snapshot2_table_tob8vc
```

JSON output:

```
{
    "environmentName": "odx-wgel6j",
    "databaseName": "test",
    "restoreSnapshots": [
        {
            "snapshotName": "snapshot2_table_tob8vc",
```

```
            "targetEnvironmentName": "odx-wgel6j",
            "targetDatabaseName": "test",
            "status": "IN_PROGRESS",
            "restoreTime": 1662014023725,
            "commandID": 1546336085
        }
    ]
}
```

Using target environment name and target database name:

```
cdp opdb list-restore-snapshots    --environment-name odx-wgel6j --database-
name test --target-environment-name odx-wgel6j --target-database-name test
```

JSON output:

```
{
    "environmentName": "odx-wgel6j",
    "databaseName": "test",
    "restoreSnapshots": [
        {
            "snapshotName": "s2",
            "targetEnvironmentName": "odx-wgel6j",
            "targetDatabaseName": "test",
            "status": "SUCCESSFUL",
            "restoreTime": 1662035967244,
            "commandID": 1546336475
        },
        {
            "snapshotName": "snapshot2_table_tob8vc",
            "targetEnvironmentName": "odx-wgel6j",
            "targetDatabaseName": "test",
            "status": "IN_PROGRESS",
            "restoreTime": 1662014023725,
            "commandID": 1546336085
        }
    ]
}
```

Related Information

COD CLI command reference GitHub repository

CDP CLI BETA command reference GitHub repository

Delete snapshots

You can use the `delete-snapshot` command to delete a snapshot from the snapshot location where the snapshot is exported.

Command syntax

`cdp opdb delete-snapshot --environment-name` *ENVIRONMENT_NAME* `--database-name` *DATABASE_NAME* `--snapshot-name` *SNAPSHOT_NAME*

Parameters

--environment-name

The name of the CDP environment.

--database-name

The name of the operational database.

--snapshot-name

The snapshot name to be deleted.

```
cdp opdb delete-snapshot --environment-name odx-wgel6j --database-name test
--snapshot-name s3
```

JSON output:

```
{
    "environmentName": "odx-wgel6j",
    "databaseName": "test",
    "status": "DELETING",
    "commandID": 1546336526,
    "snapshotName": "s3"
}
```

Once the snapshot is deleted, the status is displayed as DELETED.

```
cdp opdb list-snapshots --environment-name odx-wgel6j --database-name test -
-command-id 1546336526
```

JSON output:

```
{
    "environmentName": "odx-wgel6j",
    "databaseName": "test",
    "snapshots": [
        {
            "tableName": "table_6uf7xt",
            "snapshotName": "s3",
            "creationTime": 1662036921064,
            "status": "DELETED",
            "commandID": 1546336526,
            "snapshotLocation": "abfs://qedailynat-filesystem@qe.dfs.core.wi
ndows.net/hbase/"
        }
    ]
}
```

Related Information

COD CLI command reference GitHub repository

CDP CLI BETA command reference GitHub repository

Manage HBase snapshots using the HBase shell

You can manage snapshots by using the HBase shell, or you can use a shell script.

Shell commands

The following table shows actions you can take from the shell.

Action	Shell command	Comments
Take a snapshot of tableX called snap shotX	`snapshot 'tableX', 'snapshotX'`	Snapshots can be taken while a table is disabled, or while a table is online and serving traffic. • If a table is disabled (using disable <table>), an offline snapshot is taken. This snapshot is managed by the master and fully consistent with the state when the table was disabled. This is the simplest and safest method, but it involves a service interruption because the table must be disabled to take the snapshot. • In an online snapshot, the table remains available while the snapshot is taken, and incurs minimal performance degradation of normal read/write loads. This snapshot is managed by the master and run on the RegionServers. The current implementation—simple-flush snapshots—provides no causal consistency guarantees. Despite this shortcoming, it offers the same degree of consistency as CopyTable and is a significant improvement.
Restore snapshot snapshotX (replaces the source table content)	`restore_snapshot 'snapshotX'`	Restoring a snapshot replaces the current version of a table with the data in that table when the snapshot was taken. To run this command, you must disable the target table. The restore_snapshot command takes the name of a snapshot e (appending a timestamp code), and then clones data into the original data and removes data not in the snapshot. If the operation succeeds, the target table is enabled.
List all available snapshots	`list_snapshots`	
List all available snapshots starting with 'mysnapshot_' (regular expression)	`list_snapshots 'my_snapshot_.*'`	
Remove a snapshot called snapshotX	`delete_snapshot 'snapshotX'`	
Create a new table tableY from a snapshot snapshotX	`clone_snapshot 'snapshotX', 'tableY'`	Cloning a snapshot creates a new table that serves the data kept at the time of the snapshot. The original table and the cloned table can be modified independently; new data written to one table does not show up on the other.

Take a snapshot using a shell script

You can take a snapshot using an operating system shell script, such as a Bash script, in the HBase Shell noninteractive mode.

This example Bash script shows how to take a snapshot in this way. This script is provided as an illustration only; do not use it in production.

```
#!/bin/bash
# Take a snapshot of the table passed as an argument
# Usage: snapshot_script.sh table_name
# Names the snapshot in the format snapshot-YYYYMMDD

# Parse the arguments
if [ -z $1 ]||[$1 == '-h' ]; then
 echo "Usage: $0 &lt;table&gt;"
 echo "       $0 -h"
```

```
 exit 1
fi

# Modify to suit your environment
export HBASE_PATH=/home/user/hbase
export DATE=`date +"%Y%m%d"`
echo "snapshot '$1', 'snapshot-$DATE'" | $HBASE_PATH/bin/hbase shell -n
status=$?
if [$status -ne 0]; then
 echo "Snapshot may have failed: $status"
fi
exit $status
```

HBase Shell returns an exit code of 0 on successA non-zero exit code indicates the possibility of failure, not a definite failure. Your script should check to see if the snapshot was created before taking the snapshot again, in the event of a reported failure.

Export a snapshot to another cluster

You can export any snapshot from one cluster to another. Exporting the snapshot copies the table's hfiles, logs, and the snapshot metadata, from the source cluster to the destination cluster.

Specify the -copy-from option to copy from a remote cluster to the local cluster or another remote cluster. If you do not specify the -copy-from option, the hbase.rootdir in the HBase configuration is used, which means that you are exporting from the current cluster. You must specify the -copy-to option, to specify the destination cluster.

Note: Snapshots must be enabled on the destination cluster.

The ExportSnapshot tool executes a MapReduce Job similar to distcp to copy files to the other cluster. It works at file-system level, so the HBase cluster can be offline.

Run ExportSnapshot as the hbase user or the user that owns the files. If the user, group, or permissions need to be different on the destination cluster than the source cluster, use the -chuser, -chgroup, or -chmod options as in the second example below, or be sure the destination directory has the correct permissions. In the following examples, replace the HDFS server path and port with the appropriate ones for your cluster.

To copy a snapshot called MySnapshot to an HBase cluster srv2 (hdfs://srv2:8020/hbase) using 16 mappers:

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot MySnapshot -
copy-to hdfs://srv2:<hdfs_port>/hbase -mappers 16
```

To export the snapshot and change the ownership of the files during the copy:

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot MySnapshot -
copy-to hdfs://srv2:<hdfs_port>/hbase -chuser MyUser -chgroup MyGroup -chmod
 700 -mappers 16
```

You can also use the Java -D option in many tools to specify MapReduce or other configuration properties. For example, the following command copies MY_SNAPSHOT to hdfs://cluster2/hbase using groups of 10 hfiles per mapper:

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -Dsnapshot.export.defa
ult.map.group=10 -snapshot MY_SNAPSHOT -copy-to hdfs://cluster2/hbase
```

(The number of mappers is calculated as TotalNumberOfHFiles/10.)

To export from one remote cluster to another remote cluster, specify both -copy-from and -copy-to parameters.

You can then reverse the direction to restore the snapshot back to the first remote cluster.

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot snapshot-tes
t -copy-from hdfs://machine1/hbase -copy-to hdfs://machine2/my-backup
```

To specify a different name for the snapshot on the target cluster, use the -target option.

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot snapshot-tes
t -copy-from hdfs://machine1/hbase -copy-to hdfs://machine2/my-backup -targe
t new-snapshot
```

Information and debugging

You can use the SnapshotInfo tool to get information about a snapshot, including status, files, disk usage, and debugging information.

Examples:

Use the -h option to print usage instructions for the SnapshotInfo utility.

```
$ hbase org.apache.hadoop.hbase.snapshot.SnapshotInfo -h
Usage: hbase snapshot info [options]
Options:
    --snapshot <arg>     Snapshot to examine.
    --remote-dir <arg>   Root directory that contains the snapshots.
    --list-snapshots     List all the available snapshots and exit.
    --files              Files and logs list.
    --stats              Files and logs statistics.
    --schema             Describe the snapshotted table.
    --size-in-bytes      Print the size of the files in bytes.
```

For example,

```
hbase snapshot info --snapshot MySnapshot --files
```

Use the --list-snapshots option to list all snapshots and exit.

```
$ hbase org.apache.hadoop.hbase.snapshot.SnapshotInfo –-list-snapshots --sna
pshot foobar
SNAPSHOT             | CREATION TIME        | TABLE NAME
snapshot-test        |  2014-06-24T19:02:54 | test
```

Important: Add a dummy *SNAPSHOT* argument for the SnapshotInfo command to work.

Use the --remote-dir option with the --list-snapshots option to list snapshots located on a remote system.

```
$ hbase org.apache.hadoop.hbase.snapshot.SnapshotInfo --remote-dir s3a://myb
ucket/mysnapshot-dir --list-snapshots --snapshot foobar
SNAPSHOT |   CREATION TIME  | TABLE NAME
snapshot-test        2014-05-01 10:30    myTable
```

Use the --snapshot option to print information about a specific snapshot.

```
$ hbase org.apache.hadoop.hbase.snapshot.SnapshotInfo --snapshot test-snapsh
ot
Snapshot Info
----------------------------------------
   Name: test-snapshot
```

```
   Type: DISABLED
  Table: test-table
Version: 0
Created: 2012-12-30T11:21:21
*************************************************************
```

Use the --snapshot with the --stats options to display additional statistics about a snapshot.

```
$ hbase org.apache.hadoop.hbase.snapshot.SnapshotInfo --stats --snapshot sna
pshot-test
Snapshot Info
----------------------------------------
   Name: snapshot-test
   Type: FLUSH
  Table: test
 Format: 0
Created: 2014-06-24T19:02:54


1 HFiles (0 in archive), total size 1.0k (100.00% 1.0k shared with the sourc
e table)
```

Use the --schema option with the --snapshot option to display the schema of a snapshot.

```
$ hbase org.apache.hadoop.hbase.snapshot.SnapshotInfo  --schema --snapshot s
napshot-test
Snapshot Info
----------------------------------------
   Name: snapshot-test
   Type: FLUSH
  Table: test
 Format: 0
Created: 2014-06-24T19:02:54
Table Descriptor
----------------------------------------
'test', {NAME => 'cf', DATA_BLOCK_ENCODING => 'FAST_DIFF', BLOOMFILTER =>
 'ROW', REPLICATION_SCOPE => '0',
COMPRESSION => 'GZ', VERSIONS => '1', TTL => 'FOREVER', MIN_VERSIONS => '
0', KEEP_DELETED_CELLS => 'false',
BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
```

Use the --files option with the --snapshot option to list information about files contained in a snapshot.

```
$ hbase org.apache.hadoop.hbase.snapshot.SnapshotInfo --snapshot test-snapsh
ot --files
Snapshot Info
----------------------------------------
   Name: test-snapshot
   Type: DISABLED
  Table: test-table
Version: 0
Created: 2012-12-30T11:21:21

Snapshot Files
----------------------------------------
   52.4k test-table/02ba3a0f8964669520cf96bb4e314c60/cf/bdf29c39da2a4f2b8188
9eb4f7b18107 (archive)
   52.4k test-table/02ba3a0f8964669520cf96bb4e314c60/cf/1e06029d0a2a4a709
051b417aec88291 (archive)
   86.8k test-table/02ba3a0f8964669520cf96bb4e314c60/cf/506f601e14dc4c74a0
58be5843b99577 (archive)
```

```
   52.4k test-table/02ba3a0f8964669520cf96bb4e314c60/cf/5c7f6916ab724eacbce
a218a713941c4 (archive)
  293.4k test-table/02ba3a0f8964669520cf96bb4e314c60/cf/aec5e33a6564441d9bd4
23e31fc93abb (archive)
   52.4k test-table/02ba3a0f8964669520cf96bb4e314c60/cf/97782b2fbf0743eda
acd8fef06ba51e4 (archive)

6 HFiles (6 in archive), total size 589.7k (0.00% 0.0 shared with the sou
rce table)
0 Logs, total size 0.0
```

Using HBase replication

If your data is already in an HBase cluster, replication is useful for getting the data into additional HBase clusters.

Common replication topologies

You must learn about the common replication topologies before you configure and deploy HBase replication.

Some common replication topologies are:

- A central source cluster might propagate changes to multiple destination clusters, for failover or due to geographic distribution.
- A source cluster might push changes to a destination cluster, which might also push its own changes back to the original cluster.
- Many different low-latency clusters might push changes to one centralized cluster for backup or resource-intensive data-analytics jobs. The processed data might then be replicated back to the low-latency clusters.
- Multiple levels of replication can be chained together to suit your needs.

Related Information

Apache documentation: Cluster replication

Replication caveats

Notes about replication

You must consider a few things before using HBase replication.

- The timestamps of the replicated HLog entries are kept intact. In case of a collision (two entries identical as to row key, column family, column qualifier, and timestamp) only the entry arriving later write will be read.
- Increment Column Values (ICVs) are treated as simple puts when they are replicated. In the case where each side of replication is active (new data originates from both sources, which then replicate each other), this may be undesirable, creating identical counters that overwrite one another. Therefore Cloudera recommends that different cluster do not write increments to the same coordinates. For more information about this issue, see https://issues.apache.org/jira/browse/HBase-2804.
- Make sure the source and destination clusters are time-synchronized with each other. Cloudera recommends you use Network Time Protocol (NTP).
- Some changes are not replicated and must be propagated through other means, such as Snapshots or CopyTable.
  - Data that existed in the active cluster before replication was enabled.
  - Operations that bypass the WAL, such as when using BulkLoad or API calls such as writeToWal(false).
  - Table schema modifications.

Related Information

Manage HBase snapshots using Cloudera Operational Database CLI

Use CopyTable

Replication requirements

Before configuring replication, make sure your environment meets all requirements.

- Replication is supported from CDH 5 to CDP Private Cloud Base , and from CDP Private Cloud Base to CDH 5, if the version of CDP Private Cloud Base is 7.1 or higher.
- Each host in both clusters must be able to reach every other host, including those in the ZooKeeper cluster.
- Every table that contains families that are scoped for replication must exist on each cluster and have exactly the same name. If your tables do not yet exist on the destination cluster, see *Create empty table on the destination cluster*.
- Replication is supported from HDP 3.1.5 to CDP Private Cloud Base, if the version of CDP Private Cloud Base is 7.1.6 or higher.

Related Information

Create empty table on the destination cluster
Deploy HBase replication

Deploy HBase replication

Once you have source and destination clusters, you can enable and configure HBase replication to use it as your data import method.

About this task

Important: You need to run this as the HBase superuser, usually called "hbase". To run replication-related HBase commands, you must have HBase user permissions. If ZooKeeper uses Kerberos, configure HBase Shell to authenticate to ZooKeeper using Kerberos before attempting to run replication-related commands. No replication-related ACLs are available at this time.

Procedure

1. Configure and start the source and destination clusters.
2. Create tables with the same names and column families on both the source and destination clusters, so that the destination cluster knows where to store data it receives. All hosts in the source and destination clusters have to be reachable to each other. see *Create empty table on the destination cluster*.
3. On the source cluster, enable replication in Cloudera Manager, or by setting hbase.replication to true in hbase-site.xml.
4. Obtain Kerberos credentials as the HBase principal. Substitute your fully.qualified.domain.name and realm in the following command:

   ```
   kinit -k -t /etc/hbase/conf/hbase.keytab
   hbase/FULLY.QUALIFIED.DOMAIN.NAME@YOUR-REALM.COM
   ```

5. On the source cluster, in HBase Shell, add the destination cluster as a peer, using the add_peer command.

   ```
   add_peer 'ID', 'CLUSTER_KEY'
   ```

   To compose the CLUSTER_KEY, use the following template:

   ```
   hbase.zookeeper.quorum:hbase.zookeeper.property.clientPort:zookeeper.znode.parent
   ```

   For example:

   ```
   host1.com,host2.com,host3.com:2181:/hbase
   ```

6. On the source cluster, configure each column family to be replicated by setting its REPLICATION_SCOPE to 1, using commands such as the following in HBase Shell:

```
hbase> disable 'example_table'
hbase> alter 'example_table', {NAME => 'example_family', REPLICATION_SCOPE
 => '1'}
hbase> enable 'example_table'
```

7. Verify that replication is occurring by examining the logs on the source cluster for messages such as the following.

```
Considering 1 rs, with ratio 0.1
Getting 1 rs from peer cluster # 0
Choosing peer 192.0.2.49:62020
```

8. To verify the validity of replicated data, use the included VerifyReplication MapReduce job on the source cluster, providing it with the ID of the replication peer and table name to verify. Other options are available, such as a time range or specific families to verify.

   The command has the following form:

```
hbase org.apache.hadoop.hbase.mapreduce.replication.VerifyReplication [-
-starttime=timestamp1] [--stoptime=timestamp] [--families=comma separated
 list of families] <peerId> <tablename>
```

   The VerifyReplication command prints GOODROWS and BADROWS counters to indicate rows that did and did not replicate correctly.

Related Information

Replication requirements

Create empty table on the destination cluster

Replication across three or more clusters

When configuring replication among three or more clusters, Cloudera recommends to enable KEEP_DELETED_CELLS on column families in the destination cluster, where REPLICATION_SCOPE=1 in the source cluster.

The following commands show how to enable this configuration using HBase Shell.

- On the source cluster:

```
create 't1',{NAME=>'f1', REPLICATION_SCOPE=>1}
```

- On the destination cluster:

```
create 't1',{NAME=>'f1', KEEP_DELETED_CELLS=>'true'}
```

Enable replication on a specific table

You can enable HBase replication for a specific table on the source cluster.

To enable replication for a specific table on the source cluster, run the enable_table_replication <*TABLE*> command from the HBase shell on a cluster where a peer has been configured.

Running enable_table_replication <*TABLE*> does the following:

1. Verifies that the table exists on the source cluster.
2. If the table does not exist on the remote cluster, uses the peer configuration to duplicate the table schema, inclusing splits, on the remote cluster.
3. Enables replication on that table.

Configure secure replication

Secure replication configuration is the same whether your clusters are all in the same realm or not, with the exception of the last step.

About this task

The last step involves setting up custom secure replication configurations per peer. This can be convenient when you need to replicate to a cluster that uses different cross-realm authentication rules than the source cluster. For example, a cluster in Realm A may be allowed to replicate to Realm B and Realm C, but Realm B may not be allowed to replicate to Realm C. If you do not need this feature, skip the last step.

To use this feature, service-level principals and keytabs (specific to HBase) must be specified when you create the cluster peers using HBase Shell.

Important: Secure HBase replication is not supported if the source or the destination cluster uses custom ZooKeeper server principal. The default ZooKeeper principal is zookeeper.

Procedure

1. Set up Kerberos on your cluster.

   Note: HBase peer-to-peer replication from a non-Kerberized cluster to a Kerberized cluster is not supported.

2. If necessary, configure Kerberos cross-realm authentication:
   a. At the comman line, use the list_principals command to list the kdc, admin_server, and default_domain for each realm.
   b. Add this information to each cluster using Cloudera Manager:
      1. In Cloudera Manager, select the HDFS service.
      2. Click the Configuration tab.
      3. Find the Truster Kerberos Realms property.
      4. Add the target and source.
      5. Restart HDFS.
3. Configure ZooKeeper.
4. Configure the following HDFS parameters on both cluster:

```
<!-- In hdfs-site.xml or advanced configuration snippet -->
<property>
  <name>dfs.encrypt.data.transfer</name>
  <value>true</value>
</property>
<property>
  <name>dfs.data.transfer.protection</name>
  <value>privacy</value>
</property>

<!-- In core-site.xml or advanced configuration snippet -->
<property>
  <name>hadoop.security.authorization</name>
  <value>true</value>
</property>
<property>
  <name>hadoop.rpc.protection</name>
  <value>privacy</value>
</property>
<property>
```

```
  <name>hadoop.security.crypto.cipher.suite</name>
  <value>AES/CTR/NoPadding</value>
</property>
<property>
  <name>hadoop.ssl.enabled</name>
  <value>true</value>
</property>
```

If you use Cloudera Manager to manage your cluster, do not set these properties directly in configuration files, because Cloudera Manager will overwrite or ignore these settings. You must set these properties in Cloudera Manager.

For brevity, the Cloudera Manager setting names are not listed here, but you can search by property name. For instance, in the HDFS service configuration screen, search for dfs.encrypt.data.transfer. The Enable Data Transfer Encryption setting is shown. Selecting the box is equivalent to setting the value to true.

5. Configure the following HBase parameters on both clusters, using Cloudera Manager or in hbase-site.xml if you do not use Cloudera Managert.

```
<!-- In hbase-site.xml -->
<property>
  <name>hbase.rpc.protection</name>
  <value>privacy</value>
</property>
<property>
  <name>hbase.thrift.security.qop</name>
  <value>auth-conf</value>
</property>
<property>
  <name>hbase.thrift.ssl.enabled</name>
  <value>true</value>
</property>
<property>
  <name>hbase.rest.ssl.enabled</name>
  <value>true</value>
</property>
<property>
  <name>hbase.ssl.enabled</name>
  <value>true</value>
</property>
<property>
  <name>hbase.security.authentication</name>
  <value>kerberos</value>
</property>
<property>
  <name>hbase.security.authorization</name>
  <value>true</value>
</property>
<property>
  <name>hbase.secure.rpc.engine</name>
  <value>true</value>
</property>
```

6. Add the cluster peers using the simplified add_peer syntax:

```
add_peer 'ID', 'CLUSTER_KEY'
```

7. If you need to add any peers which require custom security configuration, modify the add_peer syntax, using the following examples as a model.

```
add_peer 'vegas', CLUSTER_KEY => 'zk1.vegas.example.com:2181:/hbase',
        CONFIG => {'hbase.master.kerberos.principal'           => 'hbase/
_HOST@TO_VEGAS',
```

```
                      'hbase.regionserver.kerberos.principal'     => 'hbase/
_HOST@TO_VEGAS',
                      'hbase.regionserver.keytab.file'            => '/key
tabs/vegas_hbase.keytab',
                      'hbase.master.keytab.file'                  => '/keyta
bs/vegas_hbase.keytab'},
          TABLE_CFS => {"tbl" => [cf1']}

add_peer 'atlanta', CLUSTER_KEY => 'zk1.vegas.example.com:2181:/hbase',
         CONFIG => {'hbase.master.kerberos.principal'           => 'hbase/
_HOST@TO_ATLANTA',
                      'hbase.regionserver.kerberos.principal'     => 'hbase/
_HOST@TO_ATLANTA',
                      'hbase.regionserver.keytab.file'            => '/ke
ytabs/atlanta_hbase.keytab',
                      'hbase.master.keytab.file'                  => '/ke
ytabs/atlanta_hbase.keytab'},
          TABLE_CFS => {"tbl" => [cf2']}
```

Related Information

Initiate replication when data already exist

Configure bulk load replication

Bulk loading is the process of preparing and loading HFiles directly into HBase RegionServers bypassing the write path. If you bulk load data into HBase frequently and want to replicate this data, you must configure and use bulk load replication.

Because bulk loading data bypasses the write path, and this process does not generate WALs, your data will not be replicated to the backup cluster.

Enable bulk load replication using Cloudera Manager

You can enable bulk load replication using Cloudera Manager.

Before you begin

If you have enabled Kerberos cross-realm authentication:

1. At the command line, use the list_principals command to list the kdc, admin_server, and default_domain for each realm.
2. Add this information to each cluster using Cloudera Manager. For each cluster, go to HDFS Configuration Trusted Kerberos Realms . Add the target and source realms. This requires a restart of HDFS.
3. Enter a Reason for change, and then click Save Changes to commit the changes. Restart the role and service when Cloudera Manager prompts you to restart.

Procedure

1. Go to the source cluster from which you want to replicate the buck loaded data.
2. Go to HBase Configuration .
3. Select Scope > (Service-Wide).
4. Locate the HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml property or search for it by typing its name in the Search box.

5. Add the following property values:
   - Name: hbase.replication.bulkload.enabled

     Value: true

     Description: Enable bulk load replication
   - Name: hbase.replication.cluster.id

     Value: source1

     Description: Provide a source cluster-ID. For example, source1.
6. Manually copy and paste the source cluster's HBase client configuration files in the target cluster where you want the data to be replicated. Copy core-site.xml, hdfs-site.xml, and hbase-site.xml to the target cluster. Do this for all RegionServers.
7. Go to the target cluster where you want the data to be replicated.
8. Go to HBase Configuration .
9. Select Scope > (Service-Wide).
10. Locate the HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml property or search for it by typing its name in the Search box.
11. Add the following property value:
    - Name: hbase.replication.conf.dir

      Value: /opt/cloudera/fs_conf

      Description: Path to the configuration directory where the source cluster's configuration files have been copied. The path for copying the configuration file is [hbase.replication.conf.dir]/[hbase.replication.cluster.id], i.e.:/opt/cloudera/fs_conf/source/
12. Restart the HBase service on both clusters to deploy the new configurations.

    Important:
    - In the target cluster, ensure that you copy the configuration files to the path set in [hbase.replication.conf.dir]. There, you must create a directory with the [hbase.replication.cluster.id] name.
    - Make sure to set the correct file permissions to hbase user using the command:

      ```
      chown -R hbase:hbase /opt/cloudera/fs_conf/source
      ```
13. Add the peer to the source cluster as you would with normal replication.
    - In the HBase shell, add the target cluster as a peer using the following command:

      ```
      add_peer '1', CLUSTER_KEY => '<cluster_name>:<hbase_port>:/hbase'
      ```
    - Enable the replication for the table to which you will be bulk loading data using the command:

      ```
      enable_table_replication 'IntegrationTestBulkLoad'
      ```
    - Alternatively, you can allow replication on a column family using the command:

      ```
      disable 'IntegrationTestBulkLoad'
      alter 'IntegrationTestBulkLoad', {NAME => 'D', REPLICATION_SCOPE => '1'}
      enable 'IntegrationTestBulkLoad'
      ```

    You can verify if BulkLoad Replication is working in your set up by following the example in this blog post:

    https://blog.cloudera.com/blog/2013/09/how-to-use-hbase-bulk-loading-and-why/

Create empty table on the destination cluster

You can create table on the destination cluster by extracting the schema using HBase Shell.

About this task

If the table to be replicated does not yet exist on the destination cluster, you must create it.

Procedure

1. On the source cluster, describe the tahble using HBase Shell.

   The ouput is reformatted for readability:

   ```
   hbase> describe acme_users

   Table acme_users is ENABLED
   acme_users
   COLUMN FAMILIES DESCRIPTION
   {NAME => 'user', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE',
   REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE',
   MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE',
   BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'false'}
   ```

2. Copy the output and make the following changes:

   - For the TTL, change FOREVER to org.apache.hadoop.hbase.HConstants::FOREVER.
   - Add the word CREATE before the table name.
   - Remove the line COLUMN FAMILIES DESCRIPTION and everything above the table name.

   A command like the following:

   ```
   "CREATE 'cme_users' ,

   {NAME => 'user', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE',
   REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE',
   MIN_VERSIONS => '0', TTL => org.apache.hadoop.hbase.HConstants::FOREVER,
    KEEP_DELETED_CELLS => 'FALSE',
   BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'false'}
   ```

3. Paste the command into HBase Shell on the destination cluster.
   The table is created.

Related Information

Replication requirements

Deploy HBase replication

Disable replication at the peer level

You can disable replication for a specific peer.

Use the command disable_peer ("<peerID>") to disable replication for a specific peer. This will stop replication to the peer, but the logs are kept for future reference.

Note: This log accumulation is a powerful side effect of the disable_peer command and can be used to your advantage.

To re-enable the peer, use the command enable_peer(<"peerID">). Replication resumes.

Examples:

- To disable peer 1:

```
disable_peer("1")
```

- To re-enable peer 1:

```
enable_peer("1")
```

Stop replication in an emergency

If replication is causing serious problems, you can stop it while the clusters are running.

Open the shell on the source cluster and use the disable_peer command for each peer, then the disable_table_replic ation command. For example:

```
hbase> disable_peer("1")
   hbase> disable_table_replication 'table_name'
```

Already queued edits will be replicated after you use the disable_table_replication command, but new entries will not.

To start replication again, use the enable_peer command.

Initiate replication when data already exist

You can initiate replication when data already exist by taking advantage of the accumulation that happens when a replication peer is disabled.

About this task

You may need to start replication from some point in the past. For example, suppose you have a primary HBase cluster in one location and are setting up a disaster-recovery (DR) cluster in another. To initialize the DR cluster, you need to copy over the existing data from the primary to the DR cluster, so that when you need to switch to the DR cluster you have a full copy of the data generated by the primary cluster. Once that is done, replication of new data can proceed as normal.

Procedure

1. Start replication.
2. Add the destination cluster as a peer.
3. Immediately disable it using disable_peer. This will cause the source HBase cluster to temporarily spool replicated updates to tables while the next steps are completed.
4. Run hbase shell and issue a snapshot 'myTable', 'myTableSnapshot-122112' for each table on the source cluster. The snapshot command flushes the table from memory.
5. Export each snapshot from the source cluster and stage it on the destination cluster. On the source cluster, for example, run `hbase org.apache.hadoop.hbase.snapshot. ExportSnapshot -snapshot MySnapshot -copy-to hdfs://yourserver:8020/hbase_root_dir -mappers 16` for each table snapshot.
6. Import and restore the snapshot on the destination cluster.
   - On the destination cluster, run hbase shell and issue a restore_snapshot for each table.

   If you are replicating data from or to a secure cluster, see Configure Replication
7. Run enable_peer to re-enable the destination cluster. Re-enabling the peer will cause the source HBase cluster to send temporarily spooled updates to the destination cluster.

Related Information

Configure secure replication

Replicate pre-exist data in an active-active deployment

You can replicate pre-exist data in an active-active deployment if you run the copyTable job before starting the replication.

About this task

You have to run the copyTable job before starting the replication. If you start the job after enabling replication, the second cluster will re-send the data to the first cluster, because copyTable does not edit the clusterId in the mutation objects.

Procedure

1. Run the copyTable job and note the start timestamp of the job.
2. Start replication.
3. Run the copyTable job again with a start time equal to the start time you noted in step 1.
   Some data being pushed back and forth between the two clusters; but it minimizes the amount of data.

Using the CldrCopyTable utility to copy data

You can use the CldrCopyTable utility to copy data from one cluster to another. You can use it to bring data in sync for replication.

About this task

CldrCopyTable is Cloudera's version of the upstream CopyTable utility.

The --cldr.cross.domain option of CldrCopyTable enables you to copy data cross-realm.

Procedure

1. Ensure that the following properties have the correct values in the hbase-site.xml configuration file of the target cluster:

```
<property>
<name>hbase.security.replication.credential.provider.path</name>
<value>cdprepjceks://hdfs@[***NAMENODE_HOST***]:[***NAMENODE_PORT***]/hba
se-replication/credentials.jceks</value>
</property>

<property>
<name>hbase.security.replication.user.name</name>
<value>srv_[***WORKLOAD USER NAME***]</value>
</property>
```

2. Ensure that the source cluster can communicate with the target cluster:
   a) Get the ZooKeeper quorum address of the target cluster.
   b) Set the address as an environment parameter in the source cluster.
   c) Set a subnet that allows connection from the source cluster.

      For example by enabling the port 2181 for the ZooKeeper client.
3. Issue the CldrCopyTable command from the source cluster to write to the target cluster.

   Based on your target cluster setup you have to use either the --cldr.cross.domain or the --cldr.unsecure.peer option.

   For Secure target cluster

   Use the --cldr.cross.domain option:

```
hbase org.apache.hadoop.hbase.mapreduce.CldrCopyTable --cldr
.cross.domain --peer.adr=[***ZOOKEEPER QUORUM***]:[***ZOOKEEPER
```

```
 PORT***]:[***ZOOKEEPER ROOT FOR HBASE***] --new.name="[***NEW TABLE
 NAME***]" "[***SOURCE TABLE NAME***]"
```

For Unsecure target cluster

Use the --cldr.unsecure.peer option:

```
hbase.org.apache.hadoop.hbase.mapreduce.CldrCopyTable --cldr
.unsecure.peer --peer.adr=[***ZOOKEEPER QUORUM***]:[***ZOOKEEPER
 PORT***]:[***ZOOKEEPER ROOT FOR HBASE***] --new.name="[***NEW TABLE
 NAME***]" "[***SOURCE TABLE NAME***]"
```

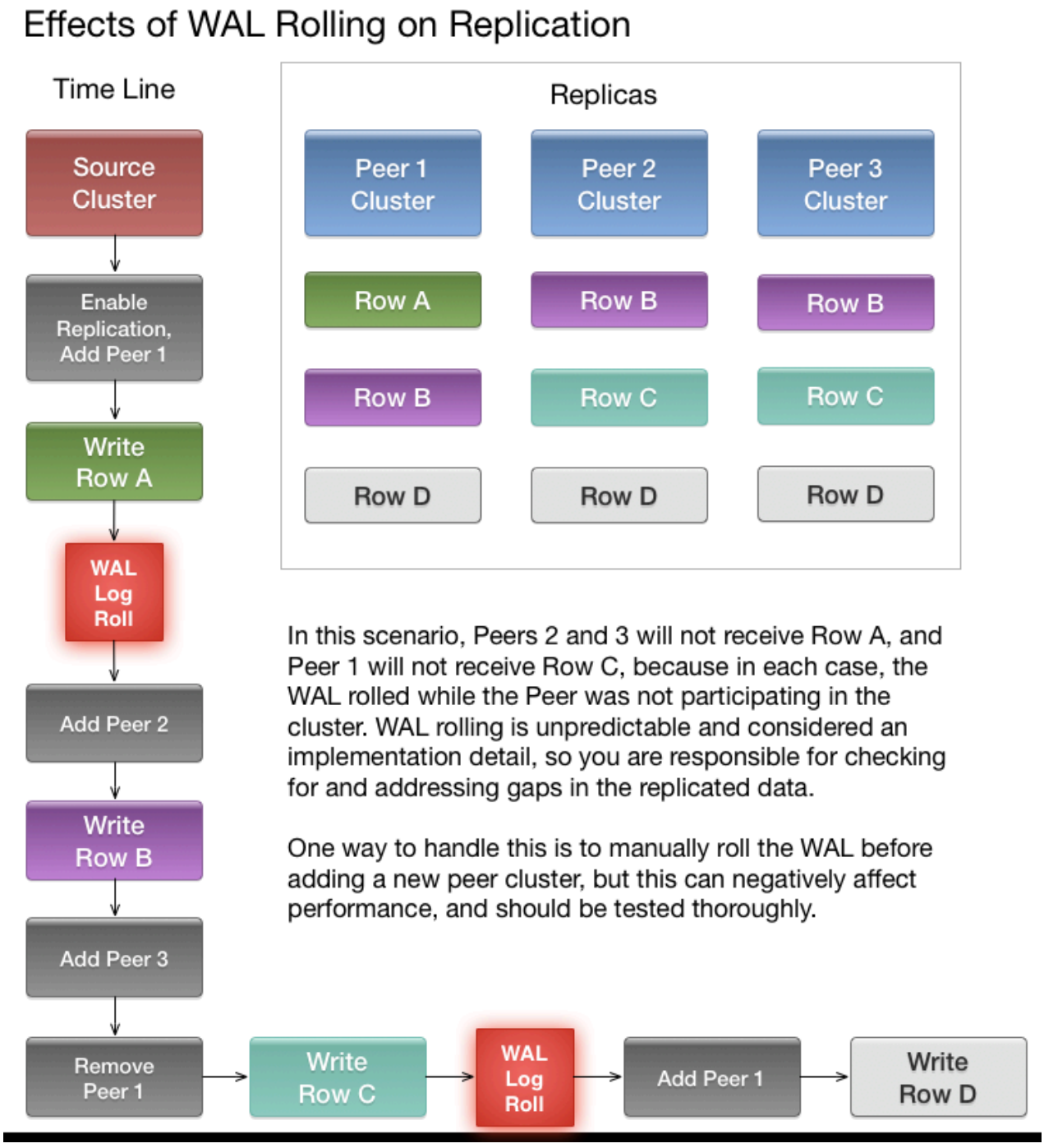
4. Once the job is finished, check the target cluster and ensure that the copy was successful.

Effects of WAL rolling on replication

Adding and removing peer clusters with unpredictable WAL rolling occurring have effects on the replication.

When you add a new peer cluster, it only receives new writes from the source cluster since the last time the WAL was rolled.

The following diagram shows the consequences of adding and removing peer clusters with unpredictable WAL rolling occurring. Follow the time line and notice which peer clusters receive which writes. Writes that occurred before the WAL is rolled are not retroactively replicated to new peers that were not participating in the cluster before the WAL was rolled.



Configuring secure HBase replication

You must configure cross realm support for Kerberos, ZooKeeper, and Hadoop to configure secure HBase replication.

About this task

There must be at least one common encryption mode between the two realms.

Note: HBase peer-to-peer replication from a non-Kerberized cluster to a Kerberized cluster is not supported.

Procedure

1. Create krbtgt principals for the two realms.

   For example, if you have two realms called EXAMPLE.com and COMPANY.TEST, you need to add the following principelas: krbtgt/EXAMPLE.COM@COMPANY.TEST and krbtgt/COMPANY.TEST@EXAMPLE.COM
2. Add the two principals at both realms.

```
kadmin: addprinc -e "<enc_type_list>" krbtgt/EXAMPLE.COM@COMPANY.TEST
kadmin: addprinc -e "<enc_type_list>" krbtgt/COMPANY.TEST@EXAMPLE.COM
```

Add rules creating short names in ZooKeeper:

3. Add a system level property in java.env, defined in the conf directory.

   The following example rule illustrates how to add support for the realm called EXAMPLE.COM and have two members in the principal (such as service/instance@EXAMPLE.COM):

```
-Dzookeeper.security.auth_to_local=RULE:[2:\$1@\$0](.*@\\QEXAMPLE.COM\\E
$)s/@\\QEXAMPLE.COM\\E$//DEFAULT
```

   This example adds support for the EXAMPLE.COM realm in a different realm. So, in the case of replication, you must add a rule for the primary cluster realm in the replica cluster realm. DEFAULT is for defining the default rule

Add rules for creating short names in the Hadoop processes:

4. Add the hadoop.security.auth_to_local property in the core-site.xml file in the replica cluster.

   For example to add support for the EXAMPLE.COM realm:

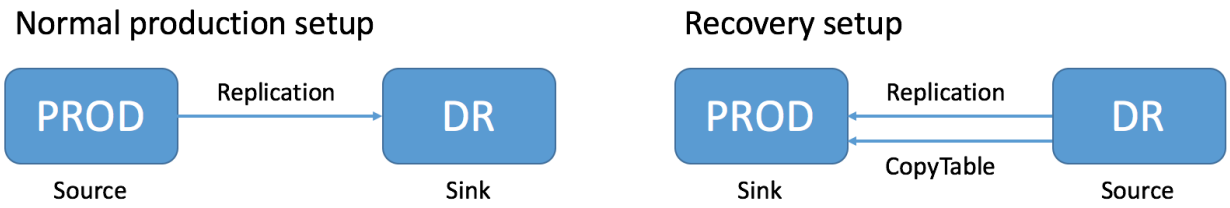
```
<property>
  <name>hadoop.security.auth_to_local</name>
  <value>
    RULE:[2:$1@$0](.*@\QEXAMPLE.COM\E$)s/@\QEXAMPLE.COM\E$//
  DEFAULT
  </value>
</property>
```

Restore data from a replica

Recover HBase data from a replicated cluster in a disaster recovery scenario.

About this task

One of the main reasons for replications is to be able to restore data, whether during disaster recovery or for other reasons. During restoration, the source and sink roles are reversed. The source is the replica cluster, and the sink is the cluster that needs restoration. This can be confusing, especially if you are in the middle of a disaster recovery scenario. The following image illustrates the role reversal between normal production and disaster recovery.



Procedure

1. Change the value of the column family property REPLICATION_SCOPE on the sink to 0 for each column to be restored, so that its data will not be replicated during the restore operation.
2. Change the value of the column family property REPLICATION_SCOPE on the source to 1 for each column to be restored, so that its data will be replicated.
3. Use the copyTable or distcp commands to import the data from the backup to the sink cluster.
4. Add the sink as a replication peer to the source, using the add_peer command.
5. If you used distcp in step 3 on page 30,restart or rolling restart both clusters.
   The RegionServers pick up the new files.

When restoration is complete, do the following:

6. Change the REPLICATION_SCOPE values back to their values before initiating the restoration.

Verify that replication works

Confirm data has been replicated from a source cluster to a remote destination cluster.

Procedure

1. Install and configure YARN on the source cluster.

   If YARN cannot be used in the source cluster, configure YARN on the destination cluster to verify replication.

   If neither the source nor the destination clusters can have YARN installed, you can configure the tool to use local mode; however, performance and consistency could be negatively impacted.
2. Ensure that you have the required permissions:
   - You have sudo permissions to run commands as the hbase user, or a user with admin permissions on both clusters.
   - You are an hbase user configured for submitting jobs with YARN.

   Note: To use the hbase user in a secure cluster, use Cloudera Manager to add the hbase user as a YARN whitelisted user. For a new installation, the hbase user is already added to the whitelisted users. In addition, /user/hbase should exist on HDFS and owned as the hbase user, because YARN will create a job staging directory there.
3. Run the VerifyReplication command:

```
src-node$ sudo -u hbase hbase org.apache.hadoop.hbase.mapreduce.replicat
ion.VerifyReplication peer1 table1
...
        org.apache.hadoop.hbase.mapreduce.replication.VerifyReplication$V
erifier$Counters
                BADROWS=2
                CONTENT_DIFFERENT_ROWS=1
                GOODROWS=1
                ONLY_IN_PEER_TABLE_ROWS=1
        File Input Format Counters
```

```
                Bytes Read=0
        File Output Format Counters
                Bytes Written=0
```

The following table describes the VerifyReplication counters:

Table 1: VerifyReplication Counters

Counter	Description
GOODROWS	Number of rows. On both clusters, and all values are the same.
CONTENT_DIFFERENT_ROWS	The key is the same on both source and destination clusters for a row, but the value differs.
ONLY_IN_SOURCE_TABLE_ROWS	Rows that are only present in the source cluster but not in the destination cluster.
ONLY_IN_PEER_TABLE_ROWS	Rows that are only present in the destination cluster but not in the source cluster.
BADROWS	Total number of rows that differ from the source and destination clusters; the sum of CONTENT_DIFFERENT_ROWS + ONLY_IN_SOURCE_TABLE_ROWS + ONLY_IN_PEER_TABLE_ROWS

By default, VerifyReplication compares the entire content of table1 on the source cluster against table1 on the destination cluster that is configured to use the replication peer peer1.

Use the following options to define the period of time, versions, or column families

Table 2: VerifyReplication Counters

Option	Description
--starttime=<timestamp>	Beginning of the time range, in milliseconds. Time range is forever if no end time is defined.
--endtime=<timestamp>	End of the time range, in milliseconds.
--versions=<versions>	Number of cell versions to verify.
--families=<cf1,cf2,..>	Families to copy; separated by commas.

The following example, verifies replication only for rows with a timestamp range of one day:

```
src-node$ sudo -u hbase hbase org.apache.hadoop.hbase.mapreduce.replicat
ion.VerifyReplication --starttime=1472499077000 --endtime=1472585477000 --
families=c1  peer1 table1
```

Replication caveats

Note these caveats when you use HBase replication.

- Two variables govern replication: hbase.replication and a replication znode. Stopping replication (using disable_table_replication as above) sets the znode to false. Two problems can result:
  - If you add a new RegionServer to the active cluster while replication is stopped, its current log will not be added to the replication queue, because the replication znode is still set to false. If you restart replication at this point (using enable_peer), entries in the log will not be replicated.
  - Similarly, if a log rolls on an existing RegionServer on the active cluster while replication is stopped, the new log will not be replicated, because the replication znode was set to false when the new log was created.

- In the case of a long-running, write-intensive workload, the destination cluster may become unresponsive if its meta-handlers are blocked while performing the replication. Cloudera Runtime has three properties to deal with this problem:
  - hbase.regionserver.replication.handler.count - the number of replication handlers in the destination cluster (default is 3). Replication is now handled by separate handlers in the destination cluster to avoid the above-mentioned sluggishness. Increase it to a high value if the ratio of active to passive RegionServers is high.
  - replication.sink.client.retries.number - the number of times the HBase replication client at the sink cluster should retry writing the WAL entries (default is 1).
  - replication.sink.client.ops.timeout - the timeout for the HBase replication client at the sink cluster (default is 20 seconds).
- For namespaces, tables, column families, or cells with associated ACLs, the ACLs themselves are not replicated. The ACLs need to be re-created manually on the target table. This behavior opens up the possibility for the ACLs could be different in the source and destination cluster.

Related Information

Common replication topologies