# Backing up and Restoring Apache HBase Datasets

Backup-and-restore is a standard set of operations for many databases. An effective backup-and-restore strategy helps ensure that you can recover data in case of data loss or failures. The HBase backup-and-restore utility helps ensure that enterprises using HBase as a data repository can recover from these types of incidents. Another important feature of the backup-and-restore utility is the ability to restore the database to a particular point-in-time, commonly referred to as a *snapshot*.

The HBase backup-and-restore utility features both *full backups* and *incremental backups*. A full backup is required at least once. The full backup is the foundation on which incremental backups are applied to build iterative snapshots. Incremental backups can be run on a schedule to capture changes over time, for example by using a Cron job. Incremental backup is more cost effective because it only captures the changes. It also enables you to restore the database to any incremental backup version. Furthermore, the utilities also enable table-level data backup-and-recovery if you do not want to restore the entire dataset of the backup.

# Planning a Backup-and-Restore Strategy for Your Environment

There are a few strategies you can use to implement backup-and-restore in your environment. The following sections show how they are implemented and identify potential tradeoffs.

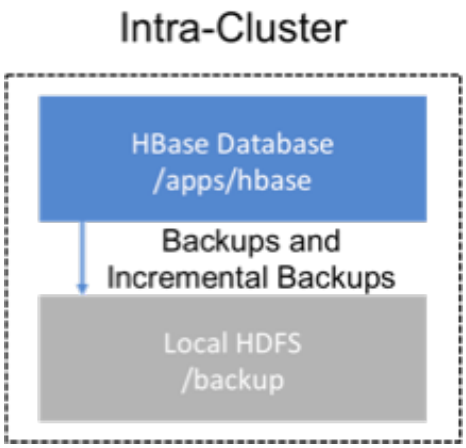
**IMPORTANT:**
HBase backup-and restore tools are currently not supported on Transparent Data Encryption (TDE)-enabled HDFS clusters. This is related to the Apache HBASE-16178 known issue.

## Backup within a Cluster

Backup-and-restore within the same cluster is only appropriate for testing. This strategy is not suitable for production unless the underlying HDFS layer is backed up and is

reliably recoverable.

**Figure 1: Intracluster Backup**

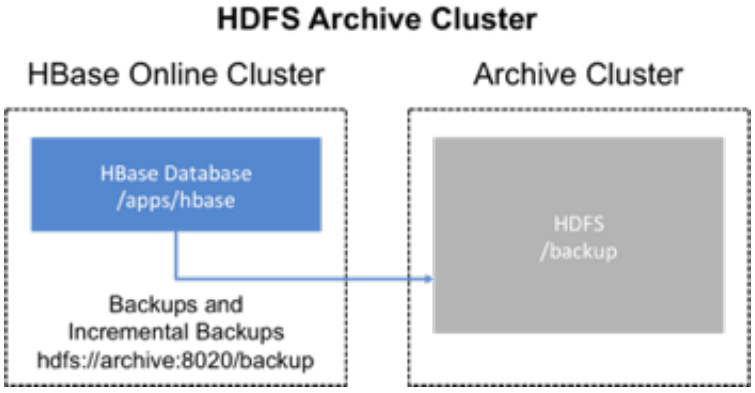



## Dedicated HDFS Archive Cluster

This strategy provides greater fault tolerance and provides a path towards disaster recovery. In this setting, you will store the backup on a separate HDFS cluster by supplying the backup destination cluster's HDFS URL to the backup utility. You should consider backing up to a different physical location, such as a different data center.

Typically, a backup-dedicated HDFS cluster uses a more economical hardware profile.

**Figure 2: Backup-Dedicated HDFS Cluster**

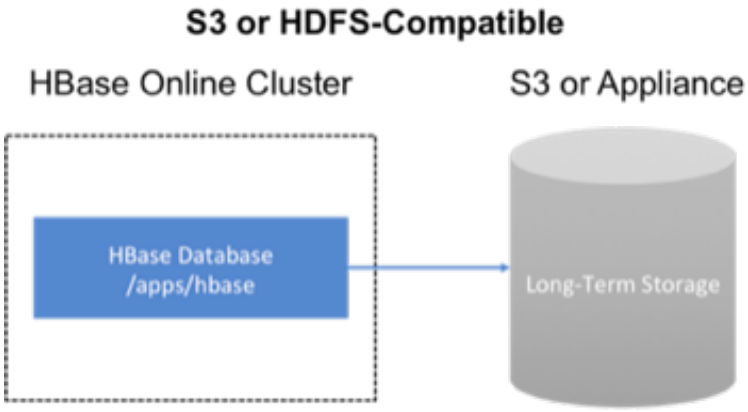



## Backup to the Cloud or a Storage Vendor

Another approach to safeguarding HBase incremental backups is to store the data on provisioned, secure servers that belong to third-party vendors and that are located off-site.

The vendor can be a public cloud provider or a storage vendor who uses a Hadoop-compatible file system, such as S3 and other HDFS-compatible destinations.

**Figure 3: Backup to Vendor Storage Solutions**




**IMPORTANT:**
The HBase backup utility does not support backup to multiple destinations. A workaround is to manually create copies of the backed up files from HDFS or S3.

# Best Practices for Backup-and-Restore

*Formulate a restore strategy and test it.*
Before you rely on a backup-and-restore strategy for your production environment, identify how backups must be performed, and more importantly, how restores must be performed. Test the plan to ensure that it is workable.

*At a minimum, store backup data from a production cluster on a different cluster or server. To further safeguard the data, use a backup location that is at a different site.*
If you have a unrecoverable loss of data on your primary production cluster as a result of computer system issues, you may be able to restore the data from a different cluster or server at the same site. However, a disaster that destroys the whole site renders locally stored backups useless. Consider storing the backup data and necessary resources (both computing capacity and operator expertise) to restore the data at a site sufficiently remote from the production site. In the case of a catastrophe at the whole primary site (fire, earthquake, etc.), the remote backup site can be very valuable.

*Secure a full backup image first.*
As a baseline, you must complete a full backup of HBase data at least once before you can rely on incremental backups. The full backup should be stored outside of the source cluster. To ensure complete dataset recovery, you must run the restore utility with the option to restore baseline full backup. The full backup is the foundation of your dataset. Incremental backup data is applied on top of the full backup during the restore operation to return you to the point in time when backup was last taken.

*Define and use backup sets for groups of tables that are logical subsets of the entire dataset.*
You can group tables into an object called a *backup set*. A backup set can save time when you have a particular group of tables that you expect to repeatedly back up or restore.

When you create a backup set, you type table names to include in the group. The backup set includes not only groups of related tables, but also retains the HBase backup metadata. Afterwards, you can invoke the backup set name to indicate what tables apply to the command execution instead of entering all the table names individually.

*Document the backup-and-restore strategy, and ideally log information about each backup.*
Document the whole process so that the knowledge base can transfer to new administrators after employee turnover. As an extra safety precaution, also log the calendar date, time, and other relevant details about the data of each backup. This metadata can potentially help locate a particular dataset in case of source cluster failure or primary site disaster. Maintain duplicate copies of all documentation: one copy at the production cluster site and another at the backup location or wherever it can be accessed by an administrator remotely from the production cluster.

# Running the Backup-and-Restore Utility

This section details the commands and their arguments of the backup-and-restore utility, as well as example usage based on task.

**TIP:**
Enter **hbase backup help *command*** in your HBase command-line interface to access the online help that provides basic information about a command and its options.

# Creating and Maintaining a Complete Backup Image

**IMPORTANT NOTE FOR APACHE PHOENIX SITES:**
Include the SQL system catalog tables in the backup. In the event that you need to restore the HBase backup, access to the system catalog tables enable you to resume Phoenix interoperability with the restored data.

The first step in running the backup-and-restore utilities is to perform a full backup and to store the data in a separate image from the source. At a minimum, you must do this to get a baseline before you can rely on incremental backups.

Run the following command as **hbase** superuser:

```
hbase backup create
      {{ full | incremental }
       {backup_root_path}
       {[tables] | [-set backup_set_name]}}

      [[-silent] |
       [-w number_of_workers] |
       [-b bandwidth_per_worker]]
```

After the command finishes running, the console prints a SUCCESS or FAILURE status message. The SUCCESS message includes a *backup ID*. The backup ID is the Unix time (also known as Epoch time) that the HBase master received the backup request from the client.

**TIP:**
Record the backup ID that appears at the end of a successful backup. In case the source cluster fails and you need to recover the dataset with a restore operation, having the backup ID readily available can save time.

## Required Command-Line Arguments

`full` or `incremental`
Using the `full` argument creates a full backup image. The `incremental` argument directs the command to create an incremental backup that has an image of data changes since the immediately preceding backup, either the full backup or the previous incremental backup.

*`backup_root_path`*

The *backup_root_path* argument specifies the full root path of where to store the backup image. Valid prefixes are `hdfs:`, `webhdfs:`, `gpfs:`, and `s3fs:`

## Optional Command-Line Arguments

*tables*
Table or tables to back up. If no table is specified, all tables are backed up. The values for this argument must be entered directly after the *backup_root_path* argument. Specify tables in a comma-separated list. Namespace wildcards are not supported yet, so to backup a namespace you must enter a full list of tables in the namespace.

`-set` *backup_set_name*
The `-set` option invokes an existing backup set in the command. See Using Backup Sets for the purpose and usage of backup sets.

`-silent`
Directs the command to not display progress and completes execution without manual interaction.

`-w` *number*
Specifies the number of parallel workers to copy data to backup destination (for example, number of map tasks in a MapReduce job).

`-b` *bandwidth_per_worker*
Specifies the bandwidth of each worker in MB per second.

## Example of Usage

**hbase backup create** full hdfs://host5:399/data/backup SALES2,SALES3 -w 3

This command creates a full backup image of two tables, SALES2 and SALES3, in the HDFS root path of `//host5:399/data/backup`. The `-w` option specifies that no more than three parallel workers complete the operation.

# Managing Backup Progress

You can monitor or cancel a running backup by using **hbase backup** subcommands and specifying the backup ID.

Run the following command as **`hbase`** superuser to view the progress of a backup:
`hbase backup progress {`*backup_ID*`}`

Run the following command as **`hbase`** superuser to cancel a running backup:
`hbase backup cancel {`*`backup_ID`*`}`

## Required Command-Line Argument

*`backup_ID`*
Specifies the backup that you want to monitor by seeing the progress information or to cancel. The backup ID is case-sensitive.

## Examples of Usage

**hbase backup progress** 1467823988425
This command displays the status of the specified backup.

**hbase backup cancel** 1467823988425
This commands cancels the specified running backup.

# Using Backup Sets

Backup sets can ease the administration of HBase data backups and restores by reducing the amount of repetitive input of table names. You can group tables into a named backup set with the **hbase backup set add** command. You can then use the `-set` option to invoke the name of a backup set in the **hbase backup create** or **hbase backup restore** rather than list individually every table in the group. You can have multiple backup sets.

**+++++**
**IMPORTANT:**
Note the differentiation between the **hbase backup set add** command and the `-set` option. The **hbase backup set add** command must be run before using the `-set` option in a different command because backup sets must be named and defined before using backup sets as shortcuts.
**+++++**

If you run the **hbase backup set add** command and specify a backup set name that does not yet exist on your system, a new set is created. If you run the command with the name of an existing backup set name, then the tables that you specify are added to the set.

In the command, the backup set name is case-sensitive.

**+++++**
**IMPORTANT:**
The metadata of backup sets are stored within HBase. If you do not have access to the original HBase cluster with the backup set metadata, then you must specify individual table names to restore the data.
**+++++**

To create a backup set, run the following command as **`hbase`** superuser:
`hbase backup set` {[add] | [remove] | [list] | [describe] | [delete]}
*`backup_set_name tables`*

## Subcommands

The following list details subcommands of the **hbase backup set** command.

**NOTE:** You must enter one (and no more than one) of the following subcommands after **hbase backup set** to complete an operation. Also, the backup set name is case-sensitive in the command-line utility.

add
Add tables to a backup set. Specify a *`backup_set_name`* value after this argument to create a backup set.

remove
Removes tables from the set. Specify the tables to remove in the *`tables`* argument.

list
Lists all backup sets.

describe
Use this subcommand to display on the screen a description of a backup set. The information includes whether the set has full or incremental backups, start and end times of the backups, and a list of the tables in the set. This subcommand must precede a valid value for the *`backup_set_name`* value.

delete
Deletes a backup set. Enter the value for the *`backup_set_name`* option directly after the **hbase backup set delete** command.

## Optional Command-Line Arguments

*backup_set_name*
Use to assign or invoke a backup set name. The backup set name must contain only printable characters and cannot have any spaces.

*tables*
List of tables (or a single table) to include in the backup set. Enter the table names as a comma-separated list. If no tables are specified, all tables are included in the set.

**TIP:**
Maintain a log or other record of the case-sensitive backup set names and the corresponding tables in each set on a separate or remote cluster, backup strategy. This information can help you in case of failure on the primary cluster.

## Example of Usage

**hbase backup set add** Q1Data TEAM_3,TEAM_4

Depending on the environment, this command results in *one* of the following actions:

• If the `Q1Data` backup set does not exist, a backup set containing tables `TEAM_3` and `TEAM_4` is created.

• If the `Q1Data` backup set exists already, the tables `TEAM_3` and `TEAM_4` are added to the `Q1Data` backup set.

# Restoring a Backup Image

Run the following command as **`hbase`** superuser. You can only restore on a live HBase cluster because the data must be redistributed to complete the restore operation successfully.

`hbase restore` {[-set *backup_set_name*] | [*backup_root_path*] | [*backup_ID*] | [*tables*]} [[*table_mapping*] | [-overwrite] | [-check]]

## Required Command-Line Arguments

-set *backup_set_name*
The `-set` option here directs the utility to restore the backup set that you specify in *backup_set_name* argument.

*backup_root_path*
The *backup_root_path* argument specifies the parent location of the stored backup image.

*backup_ID*
The backup ID that uniquely identifies the backup image to be restored.

*tables*
Table or tables to restore. The values for this argument must be entered directly after the *backup_ID* argument. Specify tables in a comma-separated list.

## Optional Command-Line Arguments

*table_mapping*
Directs the utility to restore data in the tables that are specified in the *tables* option. Each table must be mapped prior to running the command. Enter tables as a comma-separated list.

-overwrite
Truncates one or more tables in the target restore location and loads data from the backup image. The existing table must be online before the **hbase restore** command is run to successfully overwrite the data in the table. Compaction is *not* required for the data restore operation when you use the `-overwrite` argument.

-check
Verifies that the restore sequence and dependencies are in working order without actually executing a data restore.

## Example of Usage

**hbase restore** /tmp/backup_full $BACKUP_ID mytable1,mytable2 -overwrite

This command restores a full backup image. The restored data overwrites the data of an existing table. In this example:

• `/tmp/backup_full` is the path to the directory containing the backup image.

• `$BACKUP_ID` is the backup ID.

• `mytable` is the name of a table in the backup image to be restored.

• `-overwrite` is an argument that indicates the restored table overwrites all existing data in *mytable1* and *mytable2*, which is the target destination of the restore operation.

## Administering and Deleting Backup Images

The **hbase backup** command has several subcommands that help with administering backup images as they accumulate. Most production environments require recurring backups, so it is necessary to have utilities to help manage the data of the backup repository. Some subcommands enable you to find information that can help identify backups that are relevant in a search for particular data. You can also delete backup images.

The following list details each **hbase backup** ***subcommand*** that can help administer backups. Run the full command-subcommand line as **`hbase`** superuser.

hbase backup history [-n *`number_of_backups`*]
Displays a log of backup sessions. The information for each session includes backup ID, type (full or incremental), the tables in the backup, status, and start and end time. Specify the number of backup sessions to display with the optional `-n` argument. If no number is specified, the command displays a log of 10 backup sessions.

hbase backup describe {backup_ID}
Lists the backup image content, time when the backup was taken, whether the backup is full or incremental, all tables in the backup, and backup status. The `backup_ID` option is required.

hbase backup delete {backup_ID}
Deletes the specified backup image from the system. The `backup_ID` option is required.

## Technical Details of Incremental Backup-and-Restore

HBase incremental backups enable more efficient capture of HBase table images than previous attempts at serial backup-and-restore solutions, such as those that only used HBase Export and Import APIs. Incremental backups use Write Ahead Logs (WALs) to capture the data changes since the previous backup was created. A roll log is executed across all RegionServers to track the WALs that need to be in the backup.

After the incremental backup image is created, the source backup files usually are on same node as the data source. A process similar to the DistCp (distributed copy) tool is used to move the source backup files to the target filesystems. When a table restore operation starts, a two-step process is initiated. First, the full backup is restored from the full backup image. Second, all WAL files from incremental backups between the last full backup and the incremental backup being restored are converted to HFiles, which the HBase Bulk Load utility automatically imports as restored data in the table.

You can only restore on a live HBase cluster because the data must be redistributed to complete the restore operation successfully.

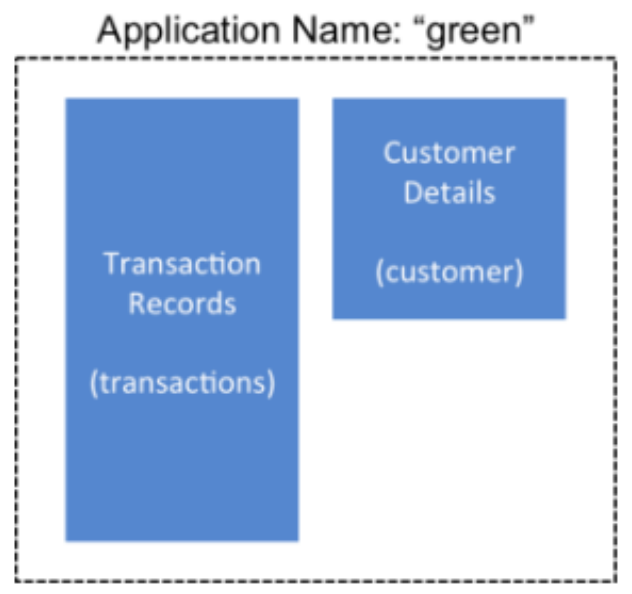
## Scenario: Safeguarding Application Datasets on Amazon S3

This scenario describes how a hypothetical retail business uses backups to safeguard application data and then restore the dataset after failure.

The HBase administration team uses backup sets to store data from a group of tables that have interrelated information for an application called *green*. In this example, one table contains transaction records and the other contains customer details. The two tables need to be backed up and be recoverable as a group.

The admin team also wants to ensure daily backups occur automatically.

**Figure 4: Tables Composing the Backup Set**



The following is an outline of the steps and examples of commands that are used to backup the data for the *green* application and to recover the data later. All commands are run when logged in as **`hbase`** superuser.

1. A backup set called *green_set* is created as an alias for both the *transactions* table and the *customer* table. The backup set can be used for all operations to avoid typing each table name. The backup set name is case-sensitive and should be formed with only printable characters and without spaces.

```
$ hbase backup set add green_set transactions
$ hbase backup set add green_set customer
```

2. The first backup of *green_set* data must be a full backup. The following command example shows how credentials are passed to Amazon S3 and specifies the file system with the **`s3a:`** prefix.

```
$ ACCESS_KEY=ABCDEFGHIJKLMNOPQRST
$ SECRET_KEY=0123456789abcdefghijklmnopqrstuvwxyzABCD
$ sudo -u hbase hbase backup create full \
  s3a://$ACCESS_KEY:$SECRET_KEY@prodhbasebackups/backups -set green_set
```

3. Incremental backups should be run according to a schedule that ensures essential data recovery in the event of a catastrophe. At this retail company, the HBase admin team decides that automated daily backups secures the data sufficiently. The team decides that they can implement this by modifying an existing Cron job that is defined in `/etc/crontab`. Consequently, IT modifies the Cron job by adding the following line:

```
@daily hbase /path/to/hbase/bin/hbase backup create incremental
s3a://$ACCESS_KEY:$SECRET_KEY@prodhbasebackups/backups -set green_set
```

4. A catastrophic IT incident disables the production cluster that the *green* application uses. An HBase system administrator of the backup cluster must restore the *green_set* dataset to the point in time closest to the recovery objective.

   **NOTE:** If the administrator of the backup HBase cluster has the backup ID with relevant details in accessible records, the following search with the **hadoop fs -ls** command and manually scanning the backup ID list can be bypassed. Consider continuously maintaining and protecting a detailed log of backup IDs outside the production cluster in your environment.

The HBase administrator runs the following command on the directory where backups are stored to print the list of successful backup IDs on the console:

```
hadoop fs -ls -t /prodhbasebackups/backups
```

5. The admin scans the list to see which backup was created at a date and time closest to the recovery objective. To do this, the admin converts the calendar timestamp of the recovery point in time to Unix time because backup IDs are uniquely identified with Unix time. The backup IDs are listed in reverse chronological order, meaning the most recent successful backup appears first.

   The admin notices that the following line in the command output corresponds with the *green_set* backup that needs to be restored:

   ```
   /prodhbasebackups/backups/backup_1467823988425
   ```

6. The admin restores *green_set* invoking the backup ID and the `-overwrite` option. The `-overwrite` option truncates all existing data in the destination and populates the tables with data from the backup dataset. Without this flag, the backup data is appended to the existing data in the destination. In this case, the admin decides to overwrite the data because it is corrupted.

```
sudo -u hbase hbase restore -set green_set \
  s3a://$ACCESS_KEY:$SECRET_KEY@prodhbasebackups/backups
 backup_1467823988425 \ -overwrite
```