Hortonworks Data Platform

User Guides

(Aug 19, 2013)

docs.hortonworks.com

Hortonworks Data Platform : User Guides

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, Zookeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain free and open source.

Please visit the Hortonworks Data Platform page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to Contact Us directly to discuss your specific needs.

Table of Contents

List of Tables

1. HBase Import Tools

HBase includes several methods of loading data into tables. Various methods exist for loading data from relational format into non-relational format.

The most straightforward method is to either use the `TableOutputFormat` class from a MapReduce job, or use the normal client APIs; however, these are not always the most efficient methods because these APIs cannot handle bulk loading.

Bulk Importing bypasses the HBase API and writes contents, which are properly formatted as HBase data files – HFiles, directly to the file system. Analyzing HBase data with MapReduce requires custom coding.

Using bulk load will use less CPU and network resources than simply using the HBase API. `ImportTsv` is a custom MapReduce application that will load data in Tab Separated Value TSV format into HBase.

The following discusses typical use cases for bulk loading data into HBase:

- HBase can act as ETL data sink
- HBase can be used as data source

Bulk load workflows generate HFiles offline and have two distinct stages:

- Use either `ImportTsv` or `import` utilities or write a custom application to generate HFiles from Hive/Pig.
- Use `completebulkload` to load the HFiles to HDFS

Note

By default, the bulk loader class `ImportTsv` in HBase imports a tab separated files.

1.1. Using Pig to Bulk Load Data Into HBase

Use the following instructions to bulk load data into HBase using Pig:

1. Prepare the input file.

   For example, consider the sample `data.tsv` file as shown below:

```
row1 c1 c2
row2 c1 c2
row3 c1 c2
row4 c1 c2
row5 c1 c2
row6 c1 c2
row7 c1 c2
row8 c1 c2
row9 c1 c2
row10    c1 c2
```

2. Make the data available on the cluster. Execute the following command on your HBase Server machine:

```
hadoop fs -put $filename /tmp/
```

   Using the previous example:

```
hadoop fs -put data.tsv /tmp/
```

3. Create or register the HBase table in HCatalog. Execute the following command on your HBase Server machine:

```
hcat -f $HBase_Table_Name
```

   For example, for a sample `simple.ddl` table as shown below:

```
CREATE TABLE
simple_hcat_load_table (id STRING, c1 STRING, c2 STRING)
STORED BY 'org.apache.hcatalog.hbase.HBaseHCatStorageHandler'
TBLPROPERTIES (
  'hbase.table.name' = 'simple_hcat_load_table',
  'hbase.columns.mapping' = 'd:c1,d:c2',
  'hcat.hbase.output.bulkMode' = 'true'
);
```

   Execute the following command:

```
hcat -f simple.ddl
```

4. Create the import file. For example, create a file named `simple.bulkload.pig` with the following contents:

Note

This import file uses the `data.tsv` file and `simple.ddl` table created previously. Ensure that you modify the contents of this file according to your environment.

```
A = LOAD 'hdfs:///tmp/data.tsv' USING PigStorage('\t') AS (id:chararray,
 c1:chararray, c2:chararray);
-- DUMP A;
STORE A INTO 'simple_hcat_load_table' USING org.apache.hcatalog.pig.
HCatStorer();
```

5. Use Pig to populate the HBase table via HCatalog bulkload.

   Continuing with the previous example, execute the following command on your HBase Server machine:

```
pig -useHCatalog simple.bulkload.pig
```

2. HBase Snapshots

HBase Snapshots allow you to take a snapshot of a table without too much impact on Region Servers. Snapshot, Clone and restore operations don't involve data copying. Also, Exporting the snapshot to another cluster doesn't have impact on the Region Servers.

Prior to version 0.94.6, the only way to backup or to clone a table is to use CopyTable/ExportTable, or to copy all the hfiles in HDFS after disabling the table. The disadvantages of these methods are that you can degrade region server performance (Copy/Export Table) or you need to disable the table, that means no reads or writes; and this is usually unacceptable. In this section:

2.1. Configuration

To turn on the snapshot support just set the `hbase.snapshot.enabled` property to true. (Snapshots are enabled by default in 0.95+ and off by default in 0.94.6+)

```
<property>
  <name>hbase.snapshot.enabled</name>
  <value>true</value>
</property>
```

2.2. Take a Snapshot

You can take a snapshot of a table regardless of whether it is enabled or disabled. The snapshot operation doesn't involve any data copying.

```
  $ ./bin/hbase shell
  hbase> snapshot 'myTable', 'myTableSnapshot-122112'
```

2.3. Listing Snapshots

List all snapshots taken (by printing the names and relative information).

```
$ ./bin/hbase shell
hbase> list_snapshots
```

2.4. Deleting Snapshots

You can remove a snapshot, and the files retained for that snapshot will be removed if no longer needed.

```
$ ./bin/hbase shell
hbase> delete_snapshot 'myTableSnapshot-122112'
```

2.5. Clone a table from snapshot

From a snapshot you can create a new table (clone operation) with the same data that you had when the snapshot was taken. The clone operation, doesn't involve data copies, and a change to the cloned table doesn't impact the snapshot or the original table.

```
$ ./bin/hbase shell
hbase> clone_snapshot 'myTableSnapshot-122112', 'myNewTestTable'
```

2.6. Restore a snapshot

The restore operation requires the table to be disabled, and the table will be restored to the state at the time when the snapshot was taken, changing both data and schema if required.

```
$ ./bin/hbase shell
hbase> disable 'myTable'
hbase> restore_snapshot 'myTableSnapshot-122112'
```

Note

Since Replication works at log level and snapshots at file-system level, after a restore, the replicas will be in a different state from the master. If you want to use restore, you need to stop replication and redo the bootstrap.

In case of partial data-loss due to misbehaving client, instead of a full restore that requires the table to be disabled, you can clone the table from the snapshot and use a Map-Reduce job to copy the data that you need, from the clone to the main one.

2.7. Snapshots operations and ACLs

If you are using security with the AccessController Coprocessor, only a global administrator can take, clone, or restore a snapshot, and these actions do not capture the ACL rights. This

means that restoring a table preserves the ACL rights of the existing table, while cloning a table creates a new table that has no ACL rights until the administrator adds them.

2.8. Export to another cluster

The ExportSnapshot tool copies all the data related to a snapshot (hfiles, logs, snapshot metadata) to another cluster. The tool executes a Map-Reduce job, similar to distcp, to copy files between the two clusters, and since it works at file-system level the hbase cluster does not have to be online. The HBase Snapshot Export tool must be run as hbase user. The HBase Snapshot Export tool must have temp directory, specified by "hbase.tmp.dir" (ie / grid/0/var/log/hbase), created on HDFS with hbase user as the owner.

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

```
$ bin/hbase class org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot
 MySnapshot -copy-to hdfs://yourserver:8020/hbase_root_dir -mappers 16
```

3. User Guide - HDFS NFS Gateway

The NFS Gateway for HDFS allows HDFS to be mounted as part of the client's local file system.

This release of NFS Gateway supports and enables the following usage patterns:

- Users can browse the HDFS file system through their local file system on NFSv3 client compatible operating systems.
- Users can download files from the the HDFS file system on to their local file system
- Users can upload files from their local file system directly to the HDFS file system

Note

NFS access to HDFS does not support random write and file appends in this release of HDP. If you need support for file appends to stream data to HDFS through NFS, upgrade to HDP 2.0.

Prerequisites:

- The NFS gateway machine needs everything to run an HDFS client like Hadoop core JAR file, HADOOP_CONF directory.
- The NFS gateway can be on any DataNode, NameNode, or any HDP client machine. Start the NFS server on that machine.

Instructions: Use the following instructions to configure and use the HDFS NFS gateway:

1. Configure settings for the HDFS NFS gateway:

   NFS gateway uses the same configurations as used by the NameNode and DataNode. Configure the following three properties based on your application's requirement:

   a. Edit the `hdfs-default.xml` file on your NFS gateway machine and modify the following property:

```
<property>
  <name>dfs.access.time.precision</name>
  <value>3600000</value>
  <description>The access time for HDFS file is precise upto this value.
               The default value is 1 hour. Setting a value of 0 disables
               access times for HDFS.
  </description>
</property>
```

Note

If the export is mounted with access time update allowed, make sure this property is not disabled in the configuration file. Only NameNode needs to restart after this property is changed. If you have disabled access time

update by mounting with "noatime" you do NOT have to change this property nor restart your NameNode.

b. Update the following property to `hdfs-site.xml`:

```
<property>
    <name>dfs.datanode.max.xcievers</name>
    <value>1024</value>
</property>
```

Note

If the number files being uploaded in parallel through the NFS gateway exceeds this value (1024), increase the value of this property accordingly. The new value must be based on the maximum number of files being uploaded in parallel.

Restart your DataNodes after making this change to the configuration file.

c. Add the following property to `hdfs-site.xml`:

```
<property>
    <name>dfs.nfs3.dump.dir</name>
    <value>/tmp/.hdfs-nfs</value>
</property>
```

Note

NFS client often reorders writes. Sequential writes can arrive at the NFS gateway at random order. This directory is used to temporarily save out-of-order writes before writing to HDFS. One needs to make sure the directory has enough space. For example, if the application uploads 10 files with each having 100MB, it is recommended for this directory to have 1GB space in case if a worst-case write reorder happens to every file.

d. Optional - Customize log settings.

Edit the `log4j.property` file to add the following:

To change trace level, add the following:

```
log4j.logger.org.apache.hadoop.hdfs.nfs=DEBUG
```

To get more details on RPC requests, add the following:

```
log4j.logger.org.apache.hadoop.oncrpc=DEBUG
```

2. Start NFS gateway service.

Three daemons are required to provide NFS service: `rpcbind` (or `portmap`), `mountd` and `nfsd`. The NFS gateway process has both `nfsd` and `mountd`. It shares the HDFS root "/" as the only export. It is recommended to use the `portmap` included in NFS gateway package as shown below:

a. Stop `nfs`/`rpcbind`/`portmap` services provided by the platform:

```
service nfs stop
service rpcbind stop
```

b. Start package included `portmap` (needs root privileges):

```
hadoop portmap
```

OR

```
hadoop-daemon.sh start portmap
```

c. Start `mountd` and `nfsd`.

No root privileges are required for this command. However, verify that the user starting the Hadoop cluster and the user starting the NFS gateway are same.

```
hadoop nfs3
```

OR

```
hadoop-daemon.sh start nfs3
```

Note

If the `hadoop-daemon.sh` script starts the NFS gateway, its log can be found in the hadoop log folder.

d. Stop NFS gateway services.

```
hadoop-daemon.sh stop nfs3
hadoop-daemon.sh stop portmap
```

3. Verify validity of NFS related services.

a. Execute the following command to verify if all the services are up and running:

```
rpcinfo -p $nfs_server_ip
```

You should see output similar to the following:

```
program vers proto   port

    100005    1   tcp   4242  mountd

    100005    2   udp   4242  mountd

    100005    2   tcp   4242  mountd

    100000    2   tcp    111  portmapper

    100000    2   udp    111  portmapper

    100005    3   udp   4242  mountd

    100005    1   udp   4242  mountd

    100003    3   tcp   2049  nfs

    100005    3   tcp   4242  mountd
```

b. Verify if the HDFS namespace is exported and can be mounted by any client.

```
showmount -e $nfs_server_ip
```

You should see output similar to the following:

```
Exports list on $nfs_server_ip :
/ (everyone)
```

4. Mount the export "/".

Currently NFS v3 is supported and uses TCP as the transportation protocol is TCP. The users can mount the HDFS namespace as shown below:

```
mount -t nfs -o vers=3,proto=tcp,nolock $server:/  $mount_point
```

Then the users can access HDFS as part of the local file system except that, hard/symbolic link and random write are not supported in this release. We do not recommend using tools like vim, for creating files on the mounted directory. The supported use cases for this release are file browsing, uploading, and downloading.

User authentication and mapping:

NFS gateway in this release uses `AUTH_UNIX` style authentication which means that the the login user on the client is the same user that NFS passes to the HDFS. For example, if the NFS client has current user as `admin`, when the user accesses the mounted directory, NFS gateway will access HDFS as user `admin`. To access HDFS as `hdfs` user, you must first switch the current user to `hdfs` on the client system before accessing the mounted directory.

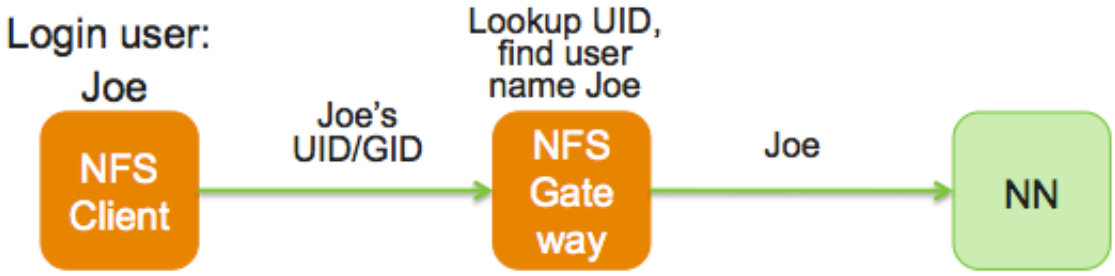
5. Set up client machine users to interact with HDFS through NFS.

NFS gateway converts UID to user name and HDFS uses username for checking permissions.

The system administrator must ensure that the user on NFS client machine has the same name and UID as that on the NFS gateway machine. This is usually not a problem if you use same user management system (e.g., LDAP/NIS) to create and deploy users to HDP nodes and to client node.

If the user is created manually, you might need to modify UID on either client or NFS gateway host in order to make them the same.

The following illustrates how the user ID and name are communicated between NFS client, NFS gateway, and NameNode.


Login user:
Joe
NFS Client
Joe's UID/GID
Lookup UID, find user name Joe
NFS Gate way
Joe
NN

4. User Guide - HDFS Snapshots

HDFS Snapshots are read-only point-in-time copies of the file system. Snapshots can be taken on a subtree of the file system or the entire file system. Some common use cases of snapshots are data backup, protection against user errors and disaster recovery.

The implementation of HDFS Snapshots is efficient:

1. Snapshot creation is instantaneous: the cost is **O(1)** excluding the inode lookup time.
2. Additional memory is used only when modifications are made relative to a snapshot: memory usage is **O(M)** where **M** is the number of modified files/directories.
3. Blocks in datanodes are not copied: the snapshot files record the block list and the file size. There is no data copying.
4. Snapshots do not adversely affect regular HDFS operations: modifications are recorded in reverse chronological order so that the current data can be accessed directly. The snapshot data is computed by subtracting the modifications from the current data.

In this document:

4.1. Snapshottable Directories

Snapshots can be taken on any directory once the directory has been set as **snapshottable**. A snapshottable directory is able to accommodate 65,536 simultaneous snapshots. There is no limit on the number of snapshottable directories. Administrators may set any directory to be snapshottable. If there are snapshots in a snapshottable directory, the directory can be neither deleted nor renamed before all the snapshots are deleted.

4.2. Snapshot Paths

For a snapshottable directory, the path component **".snapshot"** is used for accessing its snapshots. Suppose `/foo` is a snapshottable directory, `/foo/bar` is a file/directory in `/foo`, and `/foo` has a snapshot `s0`. Then, the path `/foo/.snapshot/s0/bar` refers to the snapshot copy of `/foo/bar`. The usual API and CLI can work with the ".snapshot" paths. The following are some examples:

- Listing all the snapshots under a snapshottable directory: `hadoop dfs -ls /foo/.snapshot`
- Listing the files in snapshot `s0`: `hadoop dfs -ls /foo/.snapshot/s0`
- Copying a file from snapshot `s0`: `hadoop dfs -cp /foo/.snapshot/s0/bar /tmp`

The name ".snapshot" is now a reserved file name in HDFS so that users cannot create a file/directory with ".snapshot" as the name. If ".snapshot" is used in a previous version of HDFS, it must be renamed before upgrade; otherwise, upgrade will fail.

4.3. Snapshot Operations

Snapshot operations are grouped into the following two categories:

- Administrator Operations
- User Operations

4.3.1. Administrator Operations

The operations described in this section require superuser privileges.

- Allow Snapshots: Allowing snapshots of a directory to be created. If the operation completes successfully, the directory becomes snapshottable.
  - Command:

```
hadoop dfsadmin -allowSnapshot $path
```

  - Arguments:

Table 4.1. Administrator Operations - Allow Snapshots

Parameter name	Description
path	The path of the snapshottable directory.

See also the corresponding Java API `void allowSnapshot(Path path)` in `HdfsAdmin`.

- Disallow Snapshots: Disallowing snapshots of a directory to be created. All snapshots of the directory must be deleted before disallowing snapshots.
  - Command:

```
hadoop dfsadmin -disallowSnapshot $path
```

  - Arguments:

Table 4.2. Administrator Operations - Disallow Snapshots

Parameter name	Description
path	The path of the snapshottable directory.

See also the corresponding Java API `void disallowSnapshot(Path path)` in `HdfsAdmin`.

4.3.2. User Operations

The section describes user operations. Note that HDFS superuser can perform all the operations without satisfying the permission requirement in the individual operations.

- Create Snapshots: Create a snapshot of a snapshottable directory. This operation requires owner privilege to the snapshottable directory.
  - Command:

```
hadoop dfs -createSnapshot $path $snapshotName
```

  - Arguments:

Table 4.3. User Operations - Create Snapshots

Parameter name	Description
path	The path of the snapshottable directory.
snapshotName	The snapshot name, which is an optional argument. When it is omitted, a default name is generated using a timestamp with the format "'s'yyyyMMdd-HHmmss.SSS", e.g. "s20130412-151029.033".

See also the corresponding Java API `Path createSnapshot(Path path)` and `Path createSnapshot(Path path, String snapshotName)` in `FileSystem`. The snapshot path is returned in these methods.

- Delete Snapshots: Delete a snapshot of from a snapshottable directory. This operation requires owner privilege of the snapshottable directory.
  - Command:

```
hadoop dfs -deleteSnapshot $path $snapshotName
```

  - Arguments:

Table 4.4. User Operations - Delete Snapshots

Parameter name	Description
path	The path of the snapshottable directory.
snapshotName	The snapshot name.

See also the corresponding Java API `void deleteSnapshot(Path path, String snapshotName)` in `FileSystem`.

- Rename Snapshots: Rename a snapshot. This operation requires owner privilege of the snapshottable directory..
  - Command:

```
hadoop dfs -renameSnapshot $path $oldName $newName
```

  - Arguments:

Table 4.5. User Operations - Rename Snapshots

Parameter name	Description
path	The path of the snapshottable directory.
oldName	The old snapshot name.
newName	The new snapshot name.

See also the corresponding Java API `void renameSnapshot(Path path, String oldName, String newName)` in `FileSystem`.

- Get Snapshottable Directory Listing: Get all the snapshottable directories where the current user has permission to take snapshots.
  - Command:

```
hadoop lsSnapshottableDir $path $snapshotName
```

  - Arguments:

Table 4.6. User Operations - Get Snapshottable Directory Listing

Parameter name	Description
path	The path of the snapshottable directory.
snapshotName	The snapshot name.

See also the corresponding Java API `SnapshottableDirectoryStatus[] getSnapshottableDirectoryListing()` in `DistributedFileSystem`.

- Get Snapshots Difference Report: Get the differences between two snapshots. This operation requires read access privilege for all files/directories in both snapshots.
  - Command:

```
hadoop snapshotDiff $path $fromSnapshot $toSnapshot
```

  - Arguments:

Table 4.7. User Operations - Get Snapshots Difference Report

Parameter name	Description
path	The path of the snapshottable directory.
fromSnapshot	The name of the starting snapshot.
toSnapshot	The name of the ending snapshot.

5. Add HDP Maven Repository to Existing Project

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Maven projects are defined by their **Project Object Model** or `pom`. This file is located in the base directory of a maven project and is called pom.xml.

Use one of the following options to add HDP Maven repository as a default repository in your existing project:

- **Option I: Add HDP Maven repository to existing Maven project**

  A repository in Maven is used to hold build artifacts and dependencies of varying types. There are strictly only two types of repositories: local and remote.

  The local repository refers to a copy on your own installation that is a cache of the remote downloads, and also contains the temporary build artifacts that you have not yet released. Remote repositories refer to any other type of repository, accessed by a variety of protocols such as `file://` and `http://` These repositories might be a truly remote repository set up by a third party to provide their artifacts for downloading (for example, `repo.maven.apache.org` hosts Maven's central repository). Other "remote" repositories may be internal repositories set up on a file or HTTP server within your company, used to share private artifacts between development teams and for releases.

  To add HDP Maven repository, add the following lines to your Maven project's pom.xml file:

```
<repositories>
    <repository>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>always</updatePolicy>
        <checksumPolicy>warn</checksumPolicy>
      </releases>
      <snapshots>
        <enabled>false</enabled>
        <updatePolicy>never</updatePolicy>
        <checksumPolicy>fail</checksumPolicy>
      </snapshots>
      <id>HDPReleases</id>
      <name>HDP Releases</name>
      <url>http://repo.hortonworks.com/content/repositories/releases/</url>
      <layout>default</layout>
    </repository>
  </repositories>
```

- **Option II: Add HDP Maven repository to existing Ant/Ivy project**

  Apache Ivy repositories are configured inside the `<resolvers>` element of an `ivysettings.xml` file. Usually, the resolvers (where to get those required artifacts) are provided through a separate file,`ivysettings.xml` file.

The `ivysettings.xml` file holds a chain of Ivy resolvers used for both resolution and publishing (deployment). Resolvers exist for both regular artifacts and Ivy module files. Apache Ivy uses `chain` to define the preference order for the repositories. Inside the `<chain>` element, you will find a `<url>` element. The `<url>` element is a remote site that contains bundle dependencies.

To add HDP Maven repository to existing Ant/Ivy project, add a new resolver to the existing Ivy chain so that HDP versioned artifacts can be resolved.

- **Option III: Setup Maven proxy**

  It is often the case that users wish to set up a Maven proxy repository inside their corporate firewall and have developer instances resolve artifacts through such a proxy. Proxy repositories provide a single point of remote download for an organization. In addition to control and security concerns, Proxy repositories are primarily important for increased speed across a team. These scenarios can be realized by using internal Maven repositories and a Maven proxy.

  To setup maven proxy pointing to HDP Maven or Nexus repository, use the following URL (http://repo.hortonworks.com/content/repositories/releases/) for caching the HDP artifacts to your local or internal Maven, Nexus, or Archiva repositories respectively.