Cloudera Runtime 7.3.3

Configuring HDFS ACLs

Date published: 2020-07-28
Date modified: 2026-12-13



https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2026. All rights reserved.

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

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

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

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

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

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

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

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

Contents

HDFS ACLs....4

Configuring ACLs on HDFS....4

Using CLI commands to create and list ACLs....4

ACL examples....5

ACLS on HDFS features....8

Use cases for ACLs on HDFS....9

Enable authorization for HDFS web UIs....11

Enable authorization for additional HDFS web UIs....12

Configuring HSTS for HDFS Web UIs....12

HDFS ACLs

You can use Access Control Lists (ACLs) on the Hadoop Distributed File System (HDFS). ACLs extend the HDFS permission model to support more granular file access based on arbitrary combinations of users and groups.

Configuring ACLs on HDFS

ACLs are enabled by default. You can use to disable ACLs based on your requirements.

Procedure

1. Go to the and navigate to the HDFS service.
2. Click the Configuration tab.
3. Select Scope > *Service _name* .
4. Select Category > Security .
5. Locate the Enable Access Control Lists property and clear its checkbox to disable the HDFS ACLs.
6. Enter a Reason for change, and then click Save Changes to commit the changes.
7. Restart the NameNode.

Using CLI commands to create and list ACLs

You can use the sub-commands setfacl and getfacl to create and list ACLs on HDFS.These commands are modeled after the same Linux shell commands.

- setfacl

  Sets ACLs for files and directories.

  Example:

  ```
  -setfacl [-bkR] {-m|-x} <acl_spec> <path>
  ```

  ```
  -setfacl --set <acl_spec> <path>
  ```

  Options:

  ### Table 1: ACL Options

Option	Description
-b	Remove all entries, but retain the base ACL entries. The entries for User, Group, and Others are retained for compatibility with Permission Bits.
-k	Remove the default ACL.
-R	Apply operations to all files and directories recursively.
-m	Modify the ACL. New entries are added to the ACL, and existing entries are retained.
-x	Remove the specified ACL entries. All other ACL entries are retained.

Option	Description
--set	Fully replace the ACL and discard all existing entries. The acl_spec must include entries for User, Group, and Others for compatibility with Permission Bits.
<acl_spec>	A comma-separated list of ACL entries.
<path>	The path to the file or directory to modify.

Examples:

```
hdfs dfs -setfacl -m user:hadoop:rw- /file
hdfs dfs -setfacl -x user:hadoop /file
hdfs dfs -setfacl -b /file
hdfs dfs -setfacl -k /dir
hdfs dfs -setfacl --set user::rw-,user:hadoop:rw-,group::r--,other::r-- /
file
hdfs dfs -setfacl -R -m user:hadoop:r-x /dir
hdfs dfs -setfacl -m default:user:hadoop:r-x /dir
```

Exit Code:

Returns 0 on success and non-zero on error.

- getfacl

  Displays the ACLs of files and directories. If a directory has a default ACL, getfacl also displays the default ACL.

  Usage:

  ```
  -getfacl [-R] <path>
  ```

  Options:

  Table 2: getfacl Options

Option	Description
-R	List the ACLs of all files and directories recursively.
<path>	The path to the file or directory to list.

  Examples:

  ```
  hdfs dfs -getfacl /file
  hdfs dfs -getfacl -R /dir
  ```

  Exit Code:

  Returns 0 on success and non-zero on error.

ACL examples

Access Control Lists (ACLs) on HDFS help in addressing access-related issues better than Permission Bits.

Drawbacks of Permission Bits

Before the implementation of ACls, the HDFS permission model was equivalent to traditional UNIX Permission Bits. In this model, permissions for each file or directory are managed by a set of three distinct user classes: Owner, Group, and Others. There are three permissions for each user class: Read, Write, and Execute. Thus, for any file system object, its permissions can be encoded in 3*3=9 bits. When a user attempts to access a file system object, HDFS

enforces permissions according to the most specific user class applicable to that user. If the user is the owner, HDFS checks the Owner class permissions. If the user is not the owner, but is a member of the file system object's group, HDFS checks the Group class permissions. Otherwise, HDFS checks the Others class permissions.

This model can sufficiently address a large number of security requirements. For example, consider a sales department that would like a single user -- Bruce, the department manager -- to control all modifications to sales data. Other members of the sales department need to view the data, but must not be allowed to modify it. Everyone else in the company (outside of the sales department) must not be allowed to view the data. This requirement can be implemented by running chmod 640 on the file, with the following outcome:

```
-rw-r-----1 brucesales22K Nov 18 10:55 sales-data
```

Only Bruce can modify the file, only members of the sales group can read the file, and no one else can access the file in any way.

Suppose new requirements arise as a result of which users Bruce, Diana, and Clark are allowed to make modifications. Permission Bits cannot address this requirement, because there can be only one owner and one group, and the group is already used to implement the read-only requirement for the sales team. A typical workaround is to set the file owner to a synthetic user account, such as "salesmgr," and allow Bruce, Diana, and Clark to use the "salesmgr" account through sudo or similar impersonation mechanisms. The drawback with this workaround is that it forces complexity on end-users, requiring them to use different accounts for different actions.

Consider another example where the sales staff and all the executives require access to the sales data. This is another requirement that Permission Bits cannot address, because there is only one group, and it is already used by sales. A typical workaround is to set the file's group to a new synthetic group, such as "salesandexecs," and add all users of "sales" and all users of "execs" to that group. The drawback with this workaround is that it requires administrators to create and manage additional users and groups.

The preceding examples indicate that Permission Bits cannot address permission requirements that differ from the natural organizational hierarchy of users and groups. ACLs enable you to address these requirements more naturally by allowing multiple users and multiple groups to have different sets of permissions.

Examples of using ACLs for addressing access permission requirements

The following examples explain how you can use ACLs and address different requirements related to access permissions:

Using a Default ACL for Automatic Application to New Children

In addition to an ACL that is enforced during permission checks, a default ACL is also available. A default ACL can only be applied to a directory -- not to a file. Default ACLs have no direct effect on permission checks for existing child files and directories, but instead define the ACL that new child files and directories will receive when they are created.

Suppose we have a "monthly-sales-data" directory that is further subdivided into separate directories for each month. We will set a default ACL to guarantee that members of the "execs" group automatically get access to new subdirectories as they get created each month.

- Set a default ACL on the parent directory:

  ```
  > hdfs dfs -setfacl -m default:group:execs:r-x /monthly-sales-data
  ```

- Make subdirectories:

  ```
  > hdfs dfs -mkdir /monthly-sales-data/JAN
  > hdfs dfs -mkdir /monthly-sales-data/FEB
  ```

- Verify that HDFS has automatically applied the default ACL to the subdirectories:

  ```
  > hdfs dfs -getfacl -R /monthly-sales-data
  # file: /monthly-sales-data
  ```

```
# owner: bruce
# group: sales
user::rwx
group::r-x
other::---
default:user::rwx
default:group::r-x
default:group:execs:r-x
default:mask::r-x
default:other::---

# file: /monthly-sales-data/FEB
# owner: bruce
# group: sales
user::rwx
group::r-x
group:execs:r-x
mask::r-x
other::---
default:user::rwx
default:group::r-x
default:group:execs:r-x
default:mask::r-x
default:other::---

# file: /monthly-sales-data/JAN
# owner: bruce
# group: sales
user::rwx
group::r-x
group:execs:r-x
mask::r-x
other::---
default:user::rwx
default:group::r-x
default:group:execs:r-x
default:mask::r-x
default:other::---
```

Blocking Access to a Sub-Tree for a Specific User

Suppose there is a need to immediately block access to an entire sub-tree for a specific user. Applying a named user ACL entry to the root of that sub-tree is the fastest way to accomplish this without accidentally revoking permissions for other users.

- Add an ACL entry to block user Diana's access to "monthly-sales-data":

```
> hdfs dfs -setfacl -m user:diana:--- /monthly-sales-data
```

- Run getfacl to check the results:

```
> hdfs dfs -getfacl /monthly-sales-data
# file: /monthly-sales-data
# owner: bruce
# group: sales
user::rwx
user:diana:---
group::r-x
mask::r-x
other::---
default:user::rwx
default:group::r-x
default:group:execs:r-x
```

```
default:mask::r-x
default:other::---
```

It is important to keep in mind the order of evaluation for ACL entries when a user attempts to access a file system object:

- If the user is the file owner, the Owner Permission Bits are enforced.
- Else, if the user has a named user ACL entry, those permissions are enforced.
- Else, if the user is a member of the file's group or any named group in an ACL entry, then the union of permissions for all matching entries are enforced. (The user may be a member of multiple groups.)
- If none of the above are applicable, the Other Permission Bits are enforced.

In this example, the named user ACL entry accomplished our goal because the user is not the file owner and the named user entry takes precedence over all other entries.

ACLS on HDFS features

ACLs on HDFS support features such as associating with any files or directories, access through multiple user-facing endpoints, backward compatibility with Permission Bits and so on.

POSIX ACL Implementation

ACLs on HDFS have been implemented with the POSIX ACL model. If you have ever used POSIX ACLs on a Linux file system, the HDFS ACLs work the same way.

Compatibility and Enforcement

HDFS can associate an optional ACL with any file or directory. All HDFS operations that enforce permissions expressed with Permission Bits must also enforce any ACL that is defined for the file or directory. Any existing logic that bypasses Permission Bits enforcement also bypasses ACLs. This includes the HDFS super-user and setting dfs.permissions to "false" in the configuration.

Access Through Multiple User-Facing Endpoints

HDFS supports operations for setting and getting the ACL associated with a file or directory. These operations are accessible through multiple user-facing endpoints. These endpoints include the FsShell CLI, programmatic manipulation through the FileSystem and FileContext classes, WebHDFS, and NFS.

User Feedback: CLI Indicator for ACLs

The plus symbol (+) is appended to the listed permissions of any file or directory with an associated ACL. To view, use the ls -l command.

Backward-Compatibility

The implementation of ACLs is backward-compatible with existing usage of Permission Bits. Changes applied via Permission Bits (chmod) are also visible as changes in the ACL. Likewise, changes applied to ACL entries for the base user classes (Owner, Group, and Others) are also visible as changes in the Permission Bits. Permission Bit and ACL operations manipulate a shared model, and the Permission Bit operations can be considered a subset of the ACL operations.

Low Overhead

The addition of ACLs will not cause a detrimental impact to the consumption of system resources in deployments that choose not to use ACLs. This includes CPU, memory, disk, and network bandwidth.

Using ACLs does impact NameNode performance. It is therefore recommended that you use Permission Bits, if adequate, before using ACLs.

ACL Entry Limits

The number of entries in a single ACL is capped at a maximum of 32. Attempts to add ACL entries over the maximum will fail with a user-facing error. This is done for two reasons: to simplify management, and to limit

resource consumption. ACLs with a very high number of entries tend to become difficult to understand, and may indicate that the requirements are better addressed by defining additional groups or users. ACLs with a very high number of entries also require more memory and storage, and take longer to evaluate on each permission check. The number 32 is consistent with the maximum number of ACL entries enforced by the "ext" family of file systems.

Snapshots

Within a snapshot, all ACLs are frozen at the moment that the snapshot was created. ACL changes in the parent of the snapshot are not applied to the snapshot.

Tooling

Tooling that propagates Permission Bits will propagate ACLs. This includes the cp -p shell command and distcp -p.

Use cases for ACLs on HDFS

Depending on your requirements, you can configure ACLs on HDFS to ensure that the right users and groups have the required access permissions to your data.

The following examples indicate different use cases for configuring ACLs on HDFS:

Multiple Users

In this use case, multiple users require Read access to a file. None of the users are the owner of the file. The users are not members of a common group, so it is impossible to use group Permission Bits.

This use case can be addressed by setting an access ACL containing multiple named user entries:

```
ACLs on HDFS supports the following use cases:
```

Multiple Groups

In this use case, multiple groups require Read and Write access to a file. There is no group containing all of the group members, so it is impossible to use group Permission Bits.

This use case can be addressed by setting an access ACL containing multiple named group entries:

```
group:sales:rw-
group:execs:rw-
```

Hive Partitioned Tables

In this use case, Hive contains a partitioned table of sales data. The partition key is "country". Hive persists partitioned tables using a separate subdirectory for each distinct value of the partition key, so the file system structure in HDFS looks like this:

```
user
`-- hive
 `-- warehouse
 `-- sales
 |-- country=CN
 |-- country=GB
 `-- country=US
```

All of these files belong to the "salesadmin" group. Members of this group have Read and Write access to all files. Separate country groups can run Hive queries that only read data for a specific country, such as "sales_CN", "sales_GB", and "sales_US". These groups do not have Write access.

This use case can be addressed by setting an access ACL on each subdirectory containing an owning group entry and a named group entry:

```
country=CN
```

```
group::rwx
group:sales_CN:r-x

country=GB
group::rwx
group:sales_GB:r-x

country=US
group::rwx
group:sales_US:r-x
```

Note that the functionality of the owning group ACL entry (the group entry with no name) is equivalent to setting Permission Bits.

Note:

Storage-based authorization in Hive does not currently consider the ACL permissions in HDFS. Rather, it verifies access using the traditional POSIX permissions model.

Default ACLs

In this use case, a file system administrator or sub-tree owner would like to define an access policy that will be applied to the entire sub-tree. This access policy must apply not only to the current set of files and directories, but also to any new files and directories that are added later.

This use case can be addressed by setting a default ACL on the directory. The default ACL can contain any arbitrary combination of entries. For example:

```
default:user::rwx
default:user:bruce:rw-
default:user:diana:r--
default:user:clark:rw-
default:group::r--
default:group:sales::rw-
default:group:execs::rw-
default:others::---
```

It is important to note that the default ACL gets copied from the directory to newly created child files and directories at time of creation of the child file or directory. If you change the default ACL on a directory, that will have no effect on the ACL of the files and subdirectories that already exist within the directory. Default ACLs are never considered during permission enforcement. They are only used to define the ACL that new files and subdirectories will receive automatically when they are created.

Minimal ACL/Permissions Only

HDFS ACLs support deployments that may want to use only Permission Bits and not ACLs with named user and group entries. Permission Bits are equivalent to a minimal ACL containing only 3 entries. For example:

```
user::rw-
group::r--
others::---
```

Block Access to a Sub-Tree for a Specific User

In this use case, a deeply nested file system sub-tree was created as world-readable, followed by a subsequent requirement to block access for a specific user to all files in that sub-tree.

This use case can be addressed by setting an ACL on the root of the sub-tree with a named user entry that strips all access from the user.

For this file system structure:

```
dir1
`-- dir2
```

```
  `-- dir3
  |-- file1
  |-- file2
  `-- file3
```

Setting the following ACL on "dir2" blocks access for Bruce to "dir3,""file1,""file2," and "file3":

```
user:bruce:---
```

More specifically, the removal of execute permissions on "dir2" means that Bruce cannot access "dir2", and therefore cannot see any of its children. This also means that access is blocked automatically for any new files added under "dir2". If a "file4" is created under "dir3", Bruce will not be able to access it.

ACLs with Sticky Bit

In this use case, multiple named users or named groups require full access to a shared directory, such as "/tmp". However, Write and Execute permissions on the directory also give users the ability to delete or rename any files in the directory, even files created by other users. Users must be restricted so that they are only allowed to delete or rename files that they created.

This use case can be addressed by combining an ACL with the sticky bit. The sticky bit is existing functionality that currently works with Permission Bits. It will continue to work as expected in combination with ACLs.

Enable authorization for HDFS web UIs

You can enforce authorization for the following HDFS web UIs: the NameNode, DataNode, and JournalNode.

Before you begin

You must have Kerberos authentication for HTTP web consoles and Hadoop Secure Authorization enabled. When both configurations are set, only the hdfs user can access the HDFS web UIs by default. Any other user who attempts to access the web UI will encounter an error because the user is not authorized to access the page.

For users and groups other than hdfs to access the web UIs, you must add them to hdfs-site.xml with an HDFS Service Advanced Configuration Snippet (Safety Valve).

Procedure

1. In the , go to Clusters <*HDFS SERVICE*> .
2. Navigate to the Configurations tab and search for the following property: HDFS Service Advanced Configurat ion Snippet (Safety Valve) for hdfs-site.xml.
3. Add a value for the dfs.cluster.administrators property.

   For example, a sample property might look like this:

   - Name: dfs.cluster.administrators
   - Description: ACL for the admins, this configuration is used to control who can access the default servlets in the namenode and so on. The value should be a comma separated list of users and groups. The user list comes first and is separated by a space followed by the group list. For example, user1,user2 group1,group2. Both users and groups are optional. So "user1", " group1", "", "user1 group1", "user1,user2 group1,group2" are all valid. You must note the leading space in " group1". '*' grants access to all users and groups, for example, ", ' ' and ' *' are all valid.

   These values would allow the users and groups to the following web UIs: NameNode, DataNode, and JournalNode.

   

   Note: You can view the Service Monitor Kerberos Principal by navigating to >Configuration and searching for Role-Specific Kerberos Principal.

4. Save the configuration.

5. Restart all stale HDFS services.

Enable authorization for additional HDFS web UIs

You can enforce authorization for the following HDFS web UI servlets, which may contain sensitive data: /jmx, /stack, /conf, and /metrics. When you enforce authorization for the servlets, only the users listed in the dfs.cluster.administrators property can access them.

Before you begin

requires access to the /jmx and /metrics servlets and uses the HTTP user as well as the Service Monitor Kerberos Principal to access them. Make sure to add both users to dfs.cluster.administrators, as mentioned in Enable authorization for HDFS web UIs on page 11.

Procedure

1. In the , go to Clusters <*HDFS SERVICE*> .
2. Navigate to the Configurations tab and search for the following property: HDFS Service Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml.
3. Add the hadoop.security.instrumentation.requires.admin property and set its value to true.
4. Save the configuration.
5. Restart all stale HDFS services.

Configuring HSTS for HDFS Web UIs

Configuring the HTTP Strict Transport Security (HSTS) for HDFS ensures that a web browser does not load the service information using HTTP. Additionally, all attempts to load the information using HTTP will automatically be converted to HTTPS.

Procedure

1. Go to the HDFS service.
2. Click Configuration.
3. Set the HSTS credentials in Cluster-wide Advanced Configuration Snippet (Safety Valve) for core-site.xml.

```
<property>
<name>hadoop.http.header.Strict-Transport-Security</name>
<value>max-age=63072000;includeSubDomains;preload</value>
</property>
```

4. Restart the HDFS service.