

..

Migrating Hive workloads to Iceberg

Date published: 2023-01-16

Date modified: 2023-01-17

CLOUDEXERA

Legal Notice

© Cloudera Inc. 2025. 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

Migrating Hive tables to Iceberg tables.....	4
Use cases for migrating to Iceberg.....	4
In-place migration.....	4
Prerequisites.....	5
Migrating a Hive table to Iceberg.....	5
In-place migration from Spark.....	6
Prerequisites and limitations for using Iceberg.....	6
Importing and migrating Iceberg table in Spark 3.....	7
Importing and migrating Iceberg table format v2.....	8
Best practices for Iceberg in Cloudera.....	9

Migrating Hive tables to Iceberg tables

Using Iceberg tables facilitates multi-cloud open lakehouse implementations. You can move Iceberg-based workloads in Cloudera across deployment environments on AWS and Azure. You can migrate existing external Hive tables from Hive and Impala to Iceberg in Cloudera Data Warehouse or from Spark to Iceberg in Cloudera Data Engineering.

Cloudera has chosen Apache Iceberg as the foundation for an [open lakehouse in Cloudera](#). Any compute engine can insert, update, and delete data in Iceberg tables. Any compute engine can read Iceberg tables.

The following Cloudera data services support Iceberg for performing multi-function analytics for the open lakehouse with Cloudera SDX shared security and governance:

- Cloudera Data Warehouse: Batch ETL, SQL and BI analytic workloads, row-level database operations like updates, deletes, and merge
- Cloudera Data Engineering: Batch ETL, row-level database operations, table maintenance
- Cloudera AI: Data Science through Python, R, and other languages, ML model training and inferencing, table maintenance
- Cloudera DataFlow: Nifi streaming ingestion
- Cloudera Streaming: Unified streaming ingestion with SQL

Use cases for migrating to Iceberg

A key use case for migrating Hive tables to Iceberg is the elimination of data segmentation, or data silos, and duplication. Removing these silos facilitates the tedious preparation, curation, cleansing, and moving of data before you can get any meaningful insights from your data. This use case and others are summarized.

The following list includes other use cases for migrating Hive tables to Iceberg:

- End-to-end, multi-function analytics

Cloudera provides an entire lifecycle for lower cost of ownership and architectural simplicity.

- Regulatory compliance

Improved visibility provided by quick insights into your data to lower risk and improve infosec management.

- Data protection and recovery

Data managed by Cloudera services are protected by Cloudera Shared Data Experience (SDX), an integrated set of security and governance technologies.

- Compliance with General Data Protection Regulation/California Consumer Privacy Act (GDPR/CCPA)

Iceberg supports ACID-compliance with row-level updates and deletes.

In-place migration

If you are looking for an efficient way to migrate Hive tables to Iceberg, you came to the right place. An overview of using Cloudera Data Warehouse prepares you to convert Apache Hive external tables to Apache Iceberg with no downtime. You learn the advantages of moving Hive tables to Iceberg for implementing an open lakehouse.

You can accelerate data ingestion, curation, and consumption at petabyte scale by migrating Hive tables to Iceberg.

Hive-to-Iceberg table migration is fast because the migration does not regenerate data files. Just metadata is rewritten. Cloudera Data Warehouse provides a simple API for migrating Hive tables to Iceberg to simplify adoption of Iceberg. An ALTER TABLE command sets the storage handler to convert the data without regeneration or remigration.

You can read and write files in following types of files from Hive:

- Avro
- Optimized Row Columnar (ORC)

- Parquet

Limitations

- Due to a bug in Apache Iceberg it is not possible to migrate tables that are partitioned by string columns having a partition value that contains the forward slash ‘/’ character. For more information see <https://github.com/apache/iceberg/issues/7612>
- Some column types supported by Hive tables are not supported by Iceberg tables, such as tinyint or smallint. Tables with such columns are not migrated to Iceberg.
- The original Hive table must be in AVRO, ORC, or Parquet format.

Prerequisites

You need to set up your environment and meet Data Lake prerequisites for querying Iceberg tables in Cloudera. You learn which query editors are supported and which roles are required.



Note: Do not drop, move, or change the old table during a migration operation. Doing so will delete the data files of the old and new tables.

The following list covers prerequisites for using Iceberg.

- You are using the Cloudera Data Warehouse.
- You must have access to an activated environment.
- One of the following query editors are required to query Iceberg tables:
 - Hue (recommended)
 - A JDBC client
 - The Impala shell for remote users
- You must have the role: DWUser.
- You must obtain permission to run SQL queries from the Env Admin, who must add you to the Hadoop SQL Storage Handler and the Hadoop SQL policies.
- You must use the HadoopFileIO. S3FileIO is not supported. URLs in metadata starting with s3://... cause query failure.

Migrating from Impala Prerequisites

- The original table is an EXTERNAL table from the Impala perspective: The EXTERNAL table property value is true.
- The original table is a non-ACID table.
- You have “ALL” privileges on the database containing the table.

Migrating a Hive table to Iceberg

You see how to use a simple ALTER TABLE statement from Hive or Impala to migrate an external Hive table to an Iceberg table. You see how to configure table input and output by setting table properties.

About this task



Note: To prevent loss of new and old table data during migration of a table to Iceberg, do not drop, move, or change the old table during migration.

Before you begin

In Impala, you can configure the NUM_THREADS_FOR_TABLE_MIGRATION query option to tweak the performance of the table migration. It sets the maximum number of threads to be used for the migration process but could also be limited by the number of CPUs. If set to zero then the number of available CPUs on the coordinator node is used as the maximum number of threads. Parallelism occurs on the basis of data files within a partition, which

means one partition is processed at a time with multiple threads processing the files inside the partition. In case there is only one file in each partition, sequential execution occurs.

Procedure

1. Log in to the Cloudera web interface and navigate to the Cloudera Data Warehouse service.
2. In the Cloudera Data Warehouse service, in the Overview page, locate your Virtual Warehouse, and click Hue.
Instead of using Hue, you can connect over JDBC to the Virtual Warehouse, and run the query.
3. Enter a query to use a database.
For example:

```
USE mydb;
```


4. Enter a query to migrate an existing external Hive table to an Iceberg v2 table.
Hive example:

```
ALTER TABLE tbl
SET TBLPROPERTIES ('storage_handler'='org.apache.iceberg.mr.hive.HiveIcebergStorageHandler',
'format-version' = '2');
```

Impala example, which requires two queries:

```
ALTER TABLE table_name CONVERT TO ICEBERG;
ALTER TABLE table_name SET TBLPROPERTIES ('format-version'='2');
```

The first ALTER command converts the Hive table to an Iceberg V1 table.

5. Click  to run the queries.
An Iceberg V2 table is created, replacing the Hive table.

In-place migration from Spark

In Cloudera Data Engineering, you can use Spark SQL to migrate Hive tables to Iceberg. You can convert Apache Hive external tables to Apache Iceberg with no downtime. Cloudera recommends moving Hive tables to Iceberg for implementing an open lakehouse.

You use one of the following, similar procedures to import and migrate Hive tables to Iceberg:

- Importing and migrating Iceberg table in Spark 3

A backup table is created, but does not incur the overhead of moving the physical location of the table on the object store.

- Importing and migrating Iceberg table format v2

The Iceberg merge on read operation is used.


By default, when you migrate a Hive external table to an Iceberg v2 table, the file is not rewritten. A write occurs to a new file. A read merges changes into the original file. The default merge on read tends to speed up the write and slow down the read. You can configure several other types of migration behavior if the default merge on read does not suit your use case.

For more information about using Iceberg in Cloudera Data Engineering, see [Using Apache Iceberg in Cloudera Data Engineering](#).

Prerequisites and limitations for using Iceberg

Learn about the supported versions for Cloudera Data Engineering, Spark, and Data Lake to use with Apache Iceberg.

To use Apache Iceberg in Cloudera Data Engineering, you'll need the following prerequisites:

- Spark 3.2 or higher
- A compatible version of Data Lake as listed in Cloudera Data Engineering and Data Lake compatibility linked below
- Cloudera Data Engineering 1.16 or higher
-  **Note:** Only AWS is supported in Cloudera Data Engineering 1.16 (which supports Iceberg 0.13)
- AWS or Azure is supported starting in Cloudera Data Engineering 1.17-h1 (which supports Iceberg 0.14)

Iceberg table format version 2

Iceberg table format version 2 (v2) is available starting in Iceberg 0.14. Iceberg table format v2 uses row-level UPDATE and DELETE operations that add deleted files to encoded rows that were deleted from existing data files. The DELETE, UPDATE, and MERGE operations function by writing delete files instead of rewriting the affected data files. Additionally, upon reading the data, the encoded deletes are applied to the affected rows that are read. This functionality is called merge-on-read.

To use Iceberg table format v2, you'll need the following prerequisites:

- Cloudera Data Engineering 1.17-h1 or higher
- Iceberg 0.14
- Spark 3.2 or higher

With Iceberg table format version 1 (v1), the above-mentioned operations are only supported with copy-on-write where data files are rewritten in their entirety when rows in the files are deleted. Merge-on-read is more efficient for writes, while copy-on-write is more efficient for reads.



Note: Unless otherwise indicated, the operations in the subsequent documentation apply to both v1 and v2 formats.

Importing and migrating Iceberg table in Spark 3

Importing or migrating tables are supported only on existing external Hive tables. When you import a table to Iceberg, the source and destination remain intact and independent. When you migrate a table, the existing Hive table is converted into an Iceberg table. You can use Spark SQL to import or migrate a Hive table to Iceberg.

Importing

Call the snapshot procedure to import a Hive table into Iceberg using a Spark 3 application.

```
spark.sql("CALL <catalog>.system.snapshot('<src>', '<dest>')")
```

Definitions:

- <src> is the qualified name of the Hive table
- <dest> is the qualified name of the Iceberg table to be created
- <catalog> is the name of the catalog, which you pass in a configuration file. For more information, see [Configuring Catalog](#) linked below.

For example:

```
spark.sql("CALL spark_catalog.system.snapshot('hive_db.hive_tbl',
'iceberg_db.iceberg_tbl')")
```

For information on compiling Spark 3 application with Iceberg libraries, see [Iceberg library dependencies for Spark applications](#) linked below.

Migrating

When you migrate a Hive table to Iceberg, a backup of the table, named <table_name>backup, is created.

Ensure that the `TRANSLATED_TO_EXTERNAL` property, that is located in `TBLPROPERTIES`, is set to `false` before migrating the table. This ensures that a table backup is created by renaming the table in Hive metastore (HMS) instead of moving the physical location of the table. Moving the physical location of the table would entail copying files in Amazon S3.

We recommend that you refrain from dropping the backup table, as doing so will invalidate the newly migrated table.

If you want to delete the backup table, set the following:

```
'external.table.purge' = 'FALSE'
```



Note: For Cloudera Data Engineering 1.19 and above, the property will be set automatically.

Deleting the backup table in the manner above will prevent underlying data from being deleted, therefore, only the table will be deleted from the metastore.

To undo the migration, drop the migrated table and restore the Hive table from the backup table by renaming it.

Call the migrate procedure to migrate a Hive table to Iceberg.

```
spark.sql("CALL <catalog>.system.migrate('<src>')")
```

Definitions:

- `<src>` is the qualified name of the Hive table
- `<catalog>` is the name of the catalog, which you pass in a configuration file. For more information, see [Configuring Catalog](#) linked below.

For example:

```
spark.sql("CALL  
    spark_catalog.system.migrate('hive_db.hive_tbl')")
```

Importing and migrating Iceberg table format v2

Importing or migrating Hive tables Iceberg table formats v2 are supported only on existing external Hive tables. When you import a table to Iceberg, the source and destination remain intact and independent. When you migrate a table, the existing Hive table is converted into an Iceberg table. You can use Spark SQL to import or migrate a Hive table to Iceberg.

Importing

Call the snapshot procedure to import a Hive table into Iceberg table format v2 using a Spark 3 application.

```
spark.sql("CALL <catalog>.system.snapshot(source_table => '<src>', table =>  
'<dest>', properties => map('format-version', '2', 'write.delete.mode', '<de  
lete-mode>', 'write.update.mode', '<update-mode>', 'write.merge.mode', '<mer  
ge-mode>'))")
```

Definitions:

- `<src>` is the qualified name of the Hive table
- `<dest>` is the qualified name of the Iceberg table to be created
- `<catalog>` is the name of the catalog, which you pass in a configuration file. For more information, see [Configuring Catalog](#) linked below.
- `<delete-mode>` `<update-mode>` and `<merge-mode>` are the modes that shall be used to perform the respective operation. If unspecified, they default to 'merge-on-read'

For example:

```
spark.sql("CALL spark_catalog.system.snapshot('hive_db.hive_tbl',
'iceberg_db.iceberg_tbl')")
```

For information on compiling Spark 3 application with Iceberg libraries, see [Iceberg library dependencies for Spark applications](#) linked below.

Migrating

Call the migrate procedure to migrate a Hive table to Iceberg.

```
spark.sql("CALL <catalog>.system.migrate('<src>', map('format-version', '2',
'write.delete.mode', '<delete-mode>', 'write.update.mode', '<update-mode>',
'write.merge.mode', '<merge-mode>'))")
```

Definitions:

- <src> is the qualified name of the Hive table
- <catalog> is the name of the catalog, which you pass in a configuration file. For more information, see [Configuring Catalog](#) linked below.
- <delete-mode> <update-mode> and <merge-mode> are the modes that shall be used to perform the respective operation. If unspecified, they default to 'merge-on-read'

For example:

```
spark.sql("CALL spark_catalog.system.migrate('hive_db.hive_tbl', map('format-version', '2', 'write.delete.mode', 'merge-on-read', 'write.update.mode', 'merge-on-read', 'write.merge.mode', 'merge-on-read'))")
```

Upgrading Iceberg table format v1 to v2

To upgrade an Iceberg table format from v1 to v2, run an ALTER TABLE command as follows:

```
spark.sql("ALTER TABLE <table_name> SET TBLPROPERTIES('merge-on-read', '2')")
```

<delete-mode>, <update-mode>, and <merge-mode> can be specified as the modes that shall be used to perform the respective operation. If unspecified, they default to 'merge-on-read'

Best practices for Iceberg in Cloudera

Based on large scale TPC-DS benchmark testing, performance testing and real-world experiences, Cloudera recommends several best practices when using Iceberg.

Follow the best practices listed below when using Iceberg:

- Use Iceberg as intended for analytics.

The table format is designed to manage a large, slow-changing collection of files. For more information, see the [Iceberg spec](#).

- Increase parallelism to handle large manifest list files in Spark.

By default, the number of processors determines the preset value of the iceberg.worker.num-threads system property. Try increasing parallelism by setting the iceberg.worker.num-threads system property to a higher value to speed up query compilation.

- Reduce read amplification

Monitor the growth of positional delta files, and perform timely compactions.

- Speed up drop table performance, preventing deletion of data files by using the following table properties:

```
Set external.table.purge=false and gc.enabled=false
```

- Tune the following table properties to improve concurrency on writes and reduce commit failures: `commit.retry.num-retries` (default is 4), `commit.retry.min-wait-ms` (default is 100)
- Maintain a relatively small number of data files under the iceberg table/partition directory for efficient reads. To alleviate poor performance caused by too many small files, run the following queries:

```
TRUNCATE TABLE target;  
INSERT OVERWRITE TABLE target select * from target FOR SYSTEM_VERSION AS OF <preTruncateSnapshotId>;
```

- To minimize the number of delete files and file handles and improve performance, ensure that the `Spark write.distribution.mode` table property value is “hash” (the default setting for Spark Iceberg 1.2.0 onwards).