# Data Lake Compute

# Practical Tutorial

# Product Documentation

# Contents

# Practical Tutorial

# DLC Native Table

# DLC Source Table Core Capabilities

Last updated：2024-07-31 17:34:28

## Overview

The DLC Native Table (Iceberg) is a user-friendly table format with high performance based on the Iceberg lake format. It simplifies operations, making it easy for users to perform comprehensive data exploration and build applications like Lakehouse. When using DLC Native Table (Iceberg) for the first time, users should follow these five main steps:

1. Enable DLC managed storage.
2. Purchase the engine.
3. Create the database and table. Choose to create either an append or upsert table based on your use case, and include optimization parameters.
4. Configure data optimization. Select a dedicated optimization engine and configure optimization options based on the table type.
5. Import data into the DLC Native Table. DLC supports various data writing methods, such as insert into, merge into, and upsert, as well as multiple import methods, including Spark, Presto, Flink, InLong, and Oceanus.

### Iceberg Principle Parsing

The DLC Native Table (Iceberg) uses the Iceberg table format for its underlying storage. In addition to being compatible with the open-source Iceberg capabilities, it enhances performance through separation of storage and computation and improves usability.

The Iceberg table format manages user data by dividing it into data files and metadata files.

**Data layer**: It consists of a series of data files that store user table data. These data files support Parquet, Avro, and ORC formats, with Parquet being the default format in DLC.

Due to Iceberg's snapshot mechanism, data is not immediately deleted from storage when a user deletes it. Instead, a new delete file is written to record the deleted data. Depending on the use case, delete files are categorized into position delete files and equality delete files.

Position delete files record the information of specific rows that have been deleted within a data file.

Equality delete files record the deletion of specific key values and are typically used in upsert scenarios. Delete file is also a type of data file.

**Metadata layer**: It consists of a series of manifest files, manifest lists, and metadata files. Manifest files contain metadata for a series of data files, such as file paths, write times, min-max values, and statistics.
A manifest list is composed of manifest files, typically containing the manifest files for a single snapshot.
Metadata files are in JSON format and contain information about a series of manifest list files as well as table metadata, such as table schema, partitions, and all snapshots. Whenever the table status changes, a new metadata file is generated to replace the existing one, with the Iceberg kernel ensuring atomicity for this process.

## Use Cases for Native Tables

DLC Native Table (Iceberg) is the recommended format for DLC Lakehouse. It supports two main use cases: Append tables and Upsert tables. Append tables use the V1 format, while Upsert tables use the V2 format.
Append tables: These tables support only Append, Overwrite, and Merge Into write modes.
Upsert tables: Compared to Append tables, these tables also support the Upsert write mode.
The use cases and characteristics of native tables are described in the table below.

| Table Type | Use Cases and Recommendations | Characteristics |
|---|---|---|
| Native Table (Iceberg) | 1. Users have needs for scenarios requiring real-time data writing, including append, merge into, and upsert operations. It is not limited to real-time writing using InLong, Oceanus, or self-managed Flink setups.<br>2. Storage-related Ops that users do not want to manage directly can be left to DLC managed storage.<br>3. When users prefer do not want to handle the Ops of the Iceberg table format themselves, they can let DLC manage optimization and Ops.<br>4. Users who want to leverage DLC's automatic data optimization capabilities can continuously optimize data. | 1. Iceberg table format.<br>2. Managed storage must be enabled before use.<br>3. Data is stored in DLC's managed storage.<br>4. There is no need to specify external or location information.<br>5. Enabling DLC intelligent data optimization is supported. |

For better management and use of DLC Native Table (Iceberg), certain attributes need to be specified when you create this type of table. The attributes are as follows. Users can specify these attribute values when creating a table or modify the table's attribute values later. For detailed instructions, see DLC Native Table Operational Configuration.

| Attribute Values | Meaning | Configuration Guide |
|---|---|---|
| format-version | Iceberg table version: Valid values are 1 and 2, with a default of 1. | If the user's write scenario includes upsert, this value must be set to 2. |
| | | |

| | | |
|---|---|---|
| write.upsert.enabled | Whether to enable upsert: The value is true; if not set, it will not be enabled. | If the user's write scenario includes upsert, this must be set to true. |
| write.update.mode | Update Mode | Set to merge-on-read (MOR) for MOR tables; the default is copy-on-write (COW). |
| write.merge.mode | Merge Mode | Set to merge-on-read (MOR) for MOR tables; the default is copy-on-write (COW). |
| write.parquet.bloom-filter-enabled.column.{col} | Enable bloom: Set to true to enable it; it is disabled by default. | In upsert scenarios, this must be enabled and configured according to the primary keys from the upstream data. If there are multiple primary keys in the upstream, use up to the first two. Enabling this can improve MOR query performance and small file merging efficiency. |
| write.distribution-mode | Write Mode | The recommended value is hash. When the value is hash, data will be automatically repartitioned upon writing. However, the drawback is that this may impact write performance. |
| write.metadata.delete-after-commit.enabled | Enable automatic metadata file cleanup. | It is strongly recommended to set this to true. With this setting enabled, old metadata files will be automatically cleaned up during snapshot creation to prevent the buildup of excess metadata files. |
| write.metadata.previous-versions-max | Set the default quantity of retained metadata files. | The default value is 100. In certain special cases, users can adjust this value as needed. This setting should be used with write.metadata.delete-after-commit.enabled. |
| write.metadata.metrics.default | Set the column metrics mode. | The value must be set to full. |

## Core Capabilities of Native Tables

### Managed Storage

DLC Native Table (Iceberg) uses a managed data storage mode. When using native tables (Iceberg), users must first enable managed storage and import data into the storage space managed by DLC. By using DLC managed storage, users will gain the following benefits.

Enhanced Data Security: Iceberg table data is divided into metadata and data files. If any of these files are damaged, it can cause exceptions for querying the entire table (unlike Hive, where only the corrupted file's data may be inaccessible). Storing data in DLC can help prevent users from accidentally damaging files due to a lack of understanding of Iceberg.
Performance: DLC managed storage uses CHDFS by default and offers significantly better performance compared to standard COS.
Reduced Storage Ops: By using managed storage, users no longer need to set up and maintain Cloud Object Storage themselves, and this can reduce the Ops burden associated with storage.
Data Optimization: With the managed storage mode of DLC Native Table (Iceberg), DLC provides continuous optimization for the native tables.

## ACID Transactions

Writing of Iceberg allows deleting and inserting within a single operation and is not partially visible to users so that it can offer atomic write operations.
Iceberg uses optimistic concurrency control to ensure that data writes do not cause inconsistencies. Users can only see data that has been successfully committed in the read view.
Iceberg uses snapshot mechanisms and serializable isolation levels to ensure that reads and writes are isolated.
Iceberg ensures that transactions are durable; once a transaction is successfully committed, it is permanent.

### Writing

The writing process follows optimistic concurrency control. Writers assume that the current table version will not change before they commit their updates. They update, delete, or add data and create a new version of the metadata file. When the current version is replaced with the new version, Iceberg verifies that the updates are based on the current snapshot.
If not, it indicates a write conflict, meaning that another writer has already updated the current metadata. In this case, the write operation must be updated again based on the current metadata version. The entire submission and replacement process is ensured to be atomic by the metadata lock.

### Reading

Reading and writing of Iceberg are independent processes. Readers can only see snapshots that have been successfully committed. By accessing the version's metadata file, readers obtain snapshot information to read the current table data. Since metadata files are not updated until write operations are complete, this ensures that data is always read from completed operations and never from ongoing write operations.

### Conflict Parameter Configuration

When write concurrency increases, DLC managed tables (Iceberg) may encounter write conflicts. To reduce the frequency of conflicts, users can make reasonable adjustments to their businesses in the following ways.

Go to the setting of the table structure for merging, such as partitioning, to reasonably plan the write scope of jobs. This reduces the write time of tasks and, to some extent, lowers the probability of concurrent conflicts.
Merge jobs to a certain extent to reduce the level of write concurrency.
DLC also supports a series of conflict retry parameters and increases the success rate of retry operations to some extent, thereby reducing the impact on business operations. The meanings of parameters and configuration guidance are as follows.

| Attribute values | Default System values | Meanings | Configuration guide |
|---|---|---|---|
| commit.retry.num-retries | 4 | Number of retries after a submission failure | When retries occur, you can try increasing the number of attempts. |
| commit.retry.min-wait-ms | 100 | Minimum time for waiting before retrying, in milliseconds | If conflicts are very frequent and persist even after waiting for a while, you can try to adjust this value to increase the interval between retries. |
| commit.retry.max-wait-ms | 60000（1 min） | Maximum time for waiting before retrying, in milliseconds | Adjust this value with commit.retry.min-wait-ms. |
| commit.retry.total-timeout-ms | 1800000（30 min） | Timeout for the process of submitting the entire retry | - |

## Hidden Partitioning

DLC Native Table (Iceberg) hidden partitioning hides the partition information. Developers only need to specify the partition policy when creating the table. Iceberg maintains the logical relationship between table fields and data files according to this policy. During writing and querying, there is no need to be concerned about the partition layout. Iceberg finds the partition information based on the partitioning policy and records it in the metadata during data writing. When querying, it uses the metadata to filter out files that do not need to be scanned. The partition policies provided by DLC Native Table (Iceberg) are shown in the table below.

| Transformation policy | Description | Types of original fields | Types after transformation |
|---|---|---|---|
| identity | No transformation | All types | Being consistent with the original type |
| bucket[ N, col] | Hash | int, long, decimal, date, time, timestamp, timestamptz, | int |

| | | | |
|---|---|---|---|
| | bucketing | string, uuid, fixed, binary | |
| truncate[ col] | Fixed-length truncation | int, long, decimal, string | Being consistent with the original type |
| year | Extract year information from fields | date, timestamp, timestamptz | int |
| month | Extract month information from fields | date, timestamp, timestamptz | int |
| day | Extract day information from fields | date, timestamp, timestamptz | int |
| hour | Extract hour information from fields | timestamp, timestamptz | int |

## Process of Querying and Storing Metadata

DLC Native Table (Iceberg) allows you to call stored procedure statements to query information about various types of tables, such as file merges and snapshot expiration. The table below provides some common query methods.

| Scenes | CALL statements | Execution engine |
|---|---|---|
| Querying history | select * from `DataLakeCatalog` . `db` . `sample$history` | DLC spark SQL engine, presto engine |
| Querying snapshot | select * from `DataLakeCatalog` . `db` . `sample$snapshots` | DLC spark SQL engine, presto engine |
| Querying data files | select * from `DataLakeCatalog` . `db` . `sample$files` | DLC spark SQL engine, presto engine |
| Querying manifests | select * from `DataLakeCatalog` . `db` . `sample$manifests` | DLC spark SQL engine, presto engine |
| Querying | select * from `DataLakeCatalog` . `db` . `sample$partitions` | DLC spark SQL |

| | | |
|---|---|---|
| partitions | | engine, presto engine |
| Rollback of the specific snapshot | CALL DataLakeCatalog. `system` .rollback_to_snapshot('db.sample', 1) | DLC spark SQL engine |
| Rolling back to a specific point in time | CALL DataLakeCatalog. `system` .rollback_to_timestamp('db.sample', TIMESTAMP '2021-06-30 00:00:00.000') | DLC spark SQL engine |
| Setting the current snapshot | CALL DataLakeCatalog. `system` .set_current_snapshot('db.sample', 1) | DLC spark SQL engine |
| Merging files | CALL DataLakeCatalog. `system` .rewrite_data_files(table => 'db.sample', strategy => 'sort', sort_order => 'id DESC NULLS LAST,name ASC NULLS FIRST') | DLC spark SQL engine |
| Expiration of snapshots | CALL DataLakeCatalog. `system` .expire_snapshots('db.sample', TIMESTAMP '2021-06-30 00:00:00.000', 100) | DLC spark SQL engine |
| Removing orphan files | CALL DataLakeCatalog. `system` .remove_orphan_files(table => 'db.sample', dry_run => true) | DLC spark SQL engine |
| Ewriting metadata | CALL DataLakeCatalog. `system` .rewrite_manifests('db.sample') | DLC spark SQL engine |

## Data Optimization

### Optimization Policies

DLC Native Table (Iceberg) provides optimization policies with inheritance capabilities, allowing users to configure these policies on the data management, database, and data table. For detailed configuration instructions, see Enable Data Optimization.

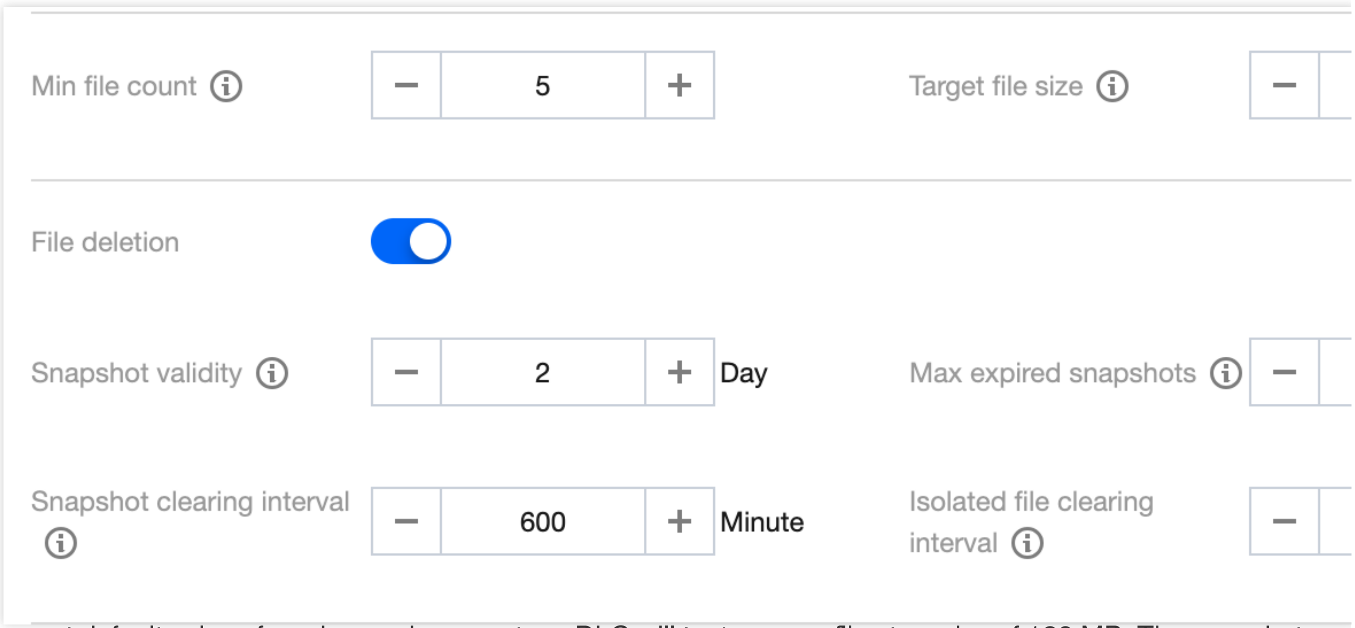
Policy for Optimizing the Configuration of the Data Management: All native tables (Iceberg) in all databases under this data management will by default inherit and use the policy for optimizing the configuration of the data management.

Policy for Optimizing the Configuration of the Database: All native tables (Iceberg) within this database will by default inherit and use the policy for optimizing the configuration of the database.

Policy for Optimizing the Configuration of the Data Table: This configuration only applies to the specified native table (Iceberg).

By using the above combination of configurations, users can implement customized optimization policies for specific databases and tables or policies for disabling certain tables.

DLC also provides advanced parameter configurations for optimization policies. If users are familiar with Iceberg, they can customize advanced parameters based on their specific scenarios, as shown in the figure below.



Min file count: 5
File deletion: on
Snapshot validity: 2 Day
Snapshot clearing interval: 600 Minute

DLC has set default values for advanced parameters. DLC will try to merge files to a size of 128 MB. The snapshot expiration time is 2 days. Five expired snapshots will be saved, and the snapshot expiration and orphan file cleanup tasks run every 600 minutes and 1440 minutes respectively.
For upsert write scenarios, DLC also provides default merge thresholds. These parameters are managed by DLC, and small file merging is triggered if new data written within a span of over 5 minutes meets any of the specified conditions, as shown in the table.

| Parameter | Meaning | Value |
|---|---|---|
| AddDataFileSize | Number of newly written data files | 20 |
| AddDeleteFileSize | Amount of newly written Delete file data | 20 |
| AddPositionDeletes | Number of newly written Position Delete records | 1000 |
| AddEqualityDeletes | Number of newly written Equality Delete records | 1000 |

**Optimization Engine**

DLC data optimization is performed by executing stored procedures, so a data engine is required to run these procedures. Currently, DLC supports using the Spark SQL engine as the optimization engine. When it is being used, please note the following points:
The Spark SQL engine for data optimization should be used separately from the business engine, and this can prevent data optimization tasks and business tasks from competing for resources and leading to significant queuing

and business disruptions.

For production scenarios, it is recommended to allocate at least 64 CU for optimization resources. For special tables with fewer than 10 tables and individual table data exceeding 2 GB, it is advised to enable auto scaling of resources to handle sudden traffic spikes. Additionally, using a monthly subscription cluster is recommended to prevent optimization task failures due to unavailability of clusters when tasks are submitted.

**Parameter Definitions**

Settings for optimizing parameters for databases and tables are on their database and table attributes. Users can specify these data optimization parameters when creating databases and tables (DLC Native Table provides a visual interface for configuring data optimization during creation). Additionally, users can modify data optimization parameters using the ALTER DATABASE/TABLE commands. For detailed instructions, see DLC Native Table Operation Configuration..

| Attribute value | Meaning | Default value | Value Description |
|---|---|---|---|
| smart-optimizer.inherit | Whether to inherit the upper level policy | default | none: Do not inherit it; default: Inherit it |
| smart-optimizer.written.enable | Whether to enable write optimization | disable | disable: No; enable: Yes. It is not enabled by default. |
| smart-optimizer.written.advance.compact-enable | (Optional) Advanced write optimization parameter: whether to enable small file merging | enable | disable: No; enable: Yes. |
| smart-optimizer.written.advance.delete-enable | (Optional) Advanced write optimization parameter: whether to enable data cleanup | enable | disable: No; enable: Yes. |
| smart-optimizer.written.advance.min- | (Optional) Minimum | 5 | When the number of files under a table or partition exceeds this |

| | | | |
|---|---|---|---|
| input-files | number of files for merging | | minimum number, the platform will automatically check them and start file optimization merging. File optimization merge can significantly improve analysis and query performance. A larger minimum file number increases resource load, while a smaller one allows for more flexible execution and more frequent tasks. A value of 5 is recommended. |
| smart-optimizer.written.advance.target-file-size-bytes | (Optional) Target size after merging | 134217728 (128 MB) | During file optimization merging, files will be merged to this target size as much as possible. The recommended value is 128 MB. |
| smart-optimizer.written.advance.before-days | (Optional) Snapshot expiration time (in days) | 2 | When the existence time of a snapshot exceeds this value, the platform will mark the snapshot as expired. The longer the snapshot expiration time, the slower the snapshot cleanup and more storage space will be occupied. |
| smart-optimizer.written.advance.retain-last | (Optional) Quantity of expired snapshots to retain | 5 | If the number of expired snapshots is bigger than that of those to be saved, the redundant expired snapshots will be cleaned up. The more expired snapshots are saved, the more storage space is used. A value of 5 is recommended. |
| smart-optimizer.written.advance.expired-snapshots-interval-min | (Optional) Snapshot expiration execution cycle | 600（10 hours） | The platform periodically scans and expires snapshots. A shorter execution cycle makes snapshot expiration more responsive but may consume more resources. |
| smart-optimizer.written.advance.remove-orphan-interval-min | (Optional) Execution cycle for removing orphan files | 1440（24 hours） | The platform periodically scans and cleans up orphan files. A shorter execution cycle makes orphan file cleanup more responsive but may consume more resources. |

**Optimization Types**

Currently, DLC provides two types of optimization: write optimization and data cleanup. Write optimization merges small files written by users into larger files to improve query efficiency. Data cleanup removes storage space occupied by historical expired snapshots, saving storage costs.

Write Optimization

Small File Merging: Merges small files written from the business side into larger files to improve file query efficiency; processes and merges deleted files and data files to enhance MOR query efficiency.

Data cleanup

Snapshot expiration: Delete expired snapshot information to free up storage space occupied by historical data.

Remove orphan files: Delete orphan files to free up storage space occupied by invalid files.

Depending on the user's usage scenario, there are certain differences among optimization types, as shown below.

| Optimization types | Recommended scenes for enabling |
|---|---|
| Write optimization | Upsert write scenarios: It must be enabled.<br>Merge into write scenarios: It must be enabled.<br>Append write scenarios: It can be enabled as needed. |
| Data cleanup | Upsert write scenarios: It must be enabled.<br>Merge into write scenarios: It must be enabled.<br>Append write scenarios: It is recommended to enable it and configure a reasonable time for deletion upon expiration based on advanced parameters and the need for rolling back historical data. |

DLC's write optimization not only merges small files but also allows for manual index creation. Users need to provide the fields and rules for the index, after which DLC will generate the corresponding stored procedure execution statements to complete the index creation. This can be done concurrently with small file merging in upsert scenarios, so that index creation is completed when small file merging is done, greatly improving index creation efficiency.

This feature is currently in the testing phase. If you need to use it, please Contact Us for configuration.

### Optimization Tasks

DLC optimization tasks are triggered in two ways: by time and by events.

### Time Triggering

Time triggers are based on the execution schedule of advanced optimization parameters. They periodically check if optimization is needed, and if the conditions for the corresponding governance item are met, a governance task is generated. The current minimum cycle for time triggers is 60 minutes, typically used for snapshot cleanup and orphan file removal.

Time triggers are still effective for tasks of optimizing small file merging, with a default trigger cycle of 60 minutes.

For V1 tables (requires activation of the backend), small file merging is triggered every 60 minutes.

For V2 tables; to prevent slow table writes and not meeting EventTriggering conditions for a long time, the V2 time trigger will start merging small files providing that it is more than 1 hour later since the last merging of small files.
If snapshot expiration or orphan file removal tasks fail or time out, they will be re-executed in the next check cycle which will start every 60 minutes.

#### EventTriggering

EventTriggering occurs in the scenarios where table upsert is written. The DLC data optimization service backend monitors the upsert writes to user tables, and when certain conditions are met, it triggers governance tasks. EventTriggering is used in small file merging scenarios, especially for real-time Flink upsert writes, as fast data writes frequently generate small file merge tasks.
For example, if the data file threshold is 20 and the deletes file threshold is 20, 20 files or 20 deletes files will be written. Meanwhile, if the minimum interval between the same task types is 5 minutes (by default), the merging of small files will be triggered.

## Lifecycle

The lifecycle of a DLC Native Table refers to the time from the last update of the table (partition) data. If there is no change after the specified time, the table (partition) will be automatically possessed. When the lifecycle of a DLC metadata table is executed, it only generates new snapshots to overwrite expired data instead of immediately removing the data from storage. The actual removal of data from storage depends on metadata table data cleanup (snapshot expiration and orphan file removal). Therefore, the lifecycle needs to be used in conjunction with data cleanup.
**Note:**
The lifecycle feature is offering test invitations. If you need activate it, please Contact Us.
When a partition is removed by the lifecycle, it is logically removed from the current snapshot. However, the removed files are not immediately deleted from the storage system. They will only be deleted from the storage system when the snapshot expires.

#### Parameter Definitions

Database and table lifecycle parameters are set on their database and table attributes. Users can carry lifecycle parameters when creating databases and tables (DLC Native Table provides a visual interface for configuring lifecycle). Users can also modify lifecycle parameters using the ALTER DATABASE/TABLE command. For detailed instructions, see DLC Native Table Operation Configuration.

| Attribute Values | Meaning | Default Values | Value description |
|---|---|---|---|
| smart-optimizer.lifecycle.enable | Enable Lifecycle | disable | disable: No; enable: Yes. It is not enabled by default. |
| smart- | Lifecycle execution | 30 | It can take effect when smart- |

| | | | |
|---|---|---|---|
| optimizer.lifecycle.expiration | cycle, unit: day | | optimizer.lifecycle.enable is set to enable, and it must be greater than 1. |

**Integrating WeData to Manage Native Table Lifecycle**

If user partition tables are partitioned by day, such as partition values yyyy-MM-dd or yyyyMMdd, WeData can be used to manage the data lifecycle.

## Data Import

DLC Native Table (Iceberg) supports multiple data import methods. According to different data sources, see the following methods for importing data.

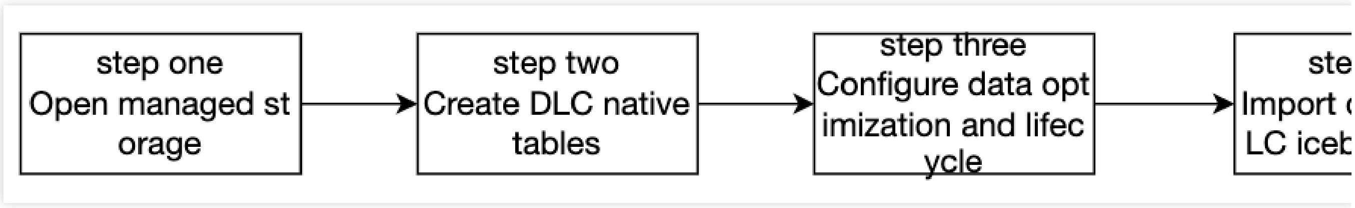
| Data location | Import recommendation |
|---|---|
| Data on the user's own COS bucket | Establish an external table in DLC, then import data using Insert into/overwrite. |
| Data is on user's local system (or other executors). | Users need to upload data to their own COS buckets, then establish an external table in DLC and import data using insert into/overwrite. |
| Data is on user's MySQL. | Users can import data using Flink/InLong/Oceanus. For detailed data lake operations, see DLC native tables (Iceberg) Lake Ingestion Practice. |
| Data is on user's self-built hive. | Users establish a Land Bond Hive data management, then import data using insert into/overwrite. |

# DLC Source Table Operation Configuration

Last updated：2024-07-31 17:34:44

## Overview

When using DLC Native Table (Iceberg), users can follow the process below to create native tables and complete the necessary configurations.



## Step I: Enabling Managed Storage

**Note:**
Managed storage must be enabled by a DLC administrator.
Enabling managed storage requires operations in the console. For details, see Managed Storage Configuration. If you use a metadata acceleration bucket, pay attention to permission configurations. For details, see Binding of Metadata Acceleration Bucket. Note that shared engines cannot access metadata acceleration buckets.

## Step II: Creating the DLC Native Table

There are two ways to create native tables.
1. Create a visual table through the console interface.
2. Create a table using SQL.
**Note:**
 A database must be created before a DLC Native Table is created.

### Creating Tables through the Console Interface

DLC provides a data management module for table creation. For detailed operations, see Data Management.

## Creating Tables through SQL

When creating tables through SQL, users write their own CREATE TABLE SQL statements. For DLC Native Table (Iceberg) creation, table descriptions, locations, and table formats do not need to be specified. However, some advanced parameters need to be included depending on the use case, and those parameters are added through TBLPROPERTIES.
If parameters were not included when you created the table or if certain attributes need to be modified, use the alter table set tblproperties command. After the alter table command is executed, restart the upstream import tasks to complete the attribute modification or addition.
Typical table creation statements for Append and Upsert scenarios are shown as follows. Users can adjust these statements based on their actual needs.

### Append Scenario Table Creation

```
  CREATE TABLE IF NOT EXISTS `DataLakeCatalog`.`axitest`.`append_case` (`id` int, `n
  PARTITIONED BY (`pt`)
  TBLPROPERTIES (
     'format-version' = '1',
     'write.upsert.enabled' = 'false',
     'write.distribution-mode' = 'hash',
     'write.metadata.delete-after-commit.enabled' = 'true',
     'write.metadata.previous-versions-max' = '100',
     'write.metadata.metrics.default' = 'full',
     'smart-optimizer.inherit' = 'default'
);
```

### Upsert Scenario Table Creation

For Upsert scenario table creation, specify the version as 2, and set the write.upsert.enabled attribute to true, and configure bloom filters according to upsert key-values. If users have multiple primary keys, generally use the first two key-values for bloom filter configuration. If the upsert table is not partitioned and updates frequently with large data volumes, consider doing bucketing by primary key for distribution.
Examples for both partitioned and non-partitioned tables are provided as follows.

```
// Partitioned table
```

```
CREATE TABLE IF NOT EXISTS `DataLakeCatalog`.`axitest`.`upsert_case` (`id` int, `na
PARTITIONED BY (bucket(4, `id`))
TBLPROPERTIES (
    'format-version' = '2',
    'write.upsert.enabled' = 'true',
    'write.update.mode' = 'merge-on-read',
    'write.merge.mode' = 'merge-on-read',
    'write.parquet.bloom-filter-enabled.column.id' = 'true',
    'dlc.ao.data.govern.sorted.keys' = 'id',
    'write.distribution-mode' = 'hash',
    'write.metadata.delete-after-commit.enabled' = 'true',
    'write.metadata.previous-versions-max' = '100',
    'write.metadata.metrics.default' = 'full',
    'smart-optimizer.inherit' = 'default'
);
```

```
// Non-partitioned table
CREATE TABLE IF NOT EXISTS `DataLakeCatalog`.`axitest`.`upsert_case` (`id` int, `na
TBLPROPERTIES (
    'format-version' = '2',
    'write.upsert.enabled' = 'true',
    'write.update.mode' = 'merge-on-read',
    'write.merge.mode' = 'merge-on-read',
    'write.parquet.bloom-filter-enabled.column.id' = 'true',
    'dlc.ao.data.govern.sorted.keys' = 'id',
    'write.distribution-mode' = 'hash',
    'write.metadata.delete-after-commit.enabled' = 'true',
    'write.metadata.previous-versions-max' = '100',
    'write.metadata.metrics.default' = 'full',
    'smart-optimizer.inherit' = 'default'
);
```

## Modifying Table Attributes

If related attribute values were not included when the user created the table, use the alter table to modify, add, or remove attribute values, as shown below. Any changes to table attribute values can be made this way. Note that the Iceberg format-version field cannot be modified. Additionally, if the table already has real-time imports from InLong/Oceanus/Flink, you need to restart the upstream import businesses after modifications.

```
// Modify conflict retry attempts to 10

ALTER TABLE `DataLakeCatalog`.`axitest`.`upsert_case` SET TBLPROPERTIES('commit.ret
// Cancel bloom filter setting for the name field

ALTER TABLE `DataLakeCatalog`.`axitest`.`upsert_case` UNSET TBLPROPERTIES('write.pa
```

## Step III: Data Optimization and Lifecycle Configuration

Data optimization and lifecycle configuration can be done in two ways.

1. Through the console interface for visual configuration

2. Through SQL for configuration

## Through the Console Interface for Configuration

DLC provides a data management module for configuration. For detailed operations, see Enable data optimization.

## Through SQL for Configuration

DLC defines detailed attributes for managing data optimization and lifecycle. You can flexibly configure data management and lifecycle based on business characteristics. For detailed data optimization and lifecycle configuration values, see Enable data optimization.

### Configuring the Database

The data optimization and lifecycle of the database can be adjusted through DBPROPERTIES, as shown below.

```
// Enable write optimization for the my_database table and do not inherit the data

ALTER DATABASE DataLakeCatalog.my_database SET  DBPROPERTIES ('smart-optimizer.inhe
// Set my_database to inherit the data management policy.

ALTER DATABASE DataLakeCatalog.my_database SET  DBPROPERTIES ('smart-optimizer.inhe
// Disable lifecycle for the my_database table and do not inherit the data manageme

ALTER DATABASE DataLakeCatalog.my_database SET  DBPROPERTIES ('smart-optimizer.inhe
```

### Configuring the Data Table

Data optimization and lifecycle for data tables are configured through TBLPROPERTIES, as shown below.

```
// Disable write optimization for the upsert_cast table and do not inherit the data

ALTER TABLE `DataLakeCatalog`.`axitest`.`upsert_case` SET TBLPROPERTIES('smart-opti
// Set the upsert_cast table to inherit the database policy.

ALTER TABLE `DataLakeCatalog`.`axitest`.`upsert_case` SET TBLPROPERTIES('smart-opti
// Enable lifecycle for the upsert_cast table, set the lifecycle duration to 7 days

ALTER TABLE `DataLakeCatalog`.`axitest`.`upsert_case` SET TBLPROPERTIES('smart-opti
```

## Step IV: Data Ingestion into Native Table

DLC Native Table supports multiple data ingestion methods. Depending on your data source, see DLC Native Table Lake Ingestion Practice.

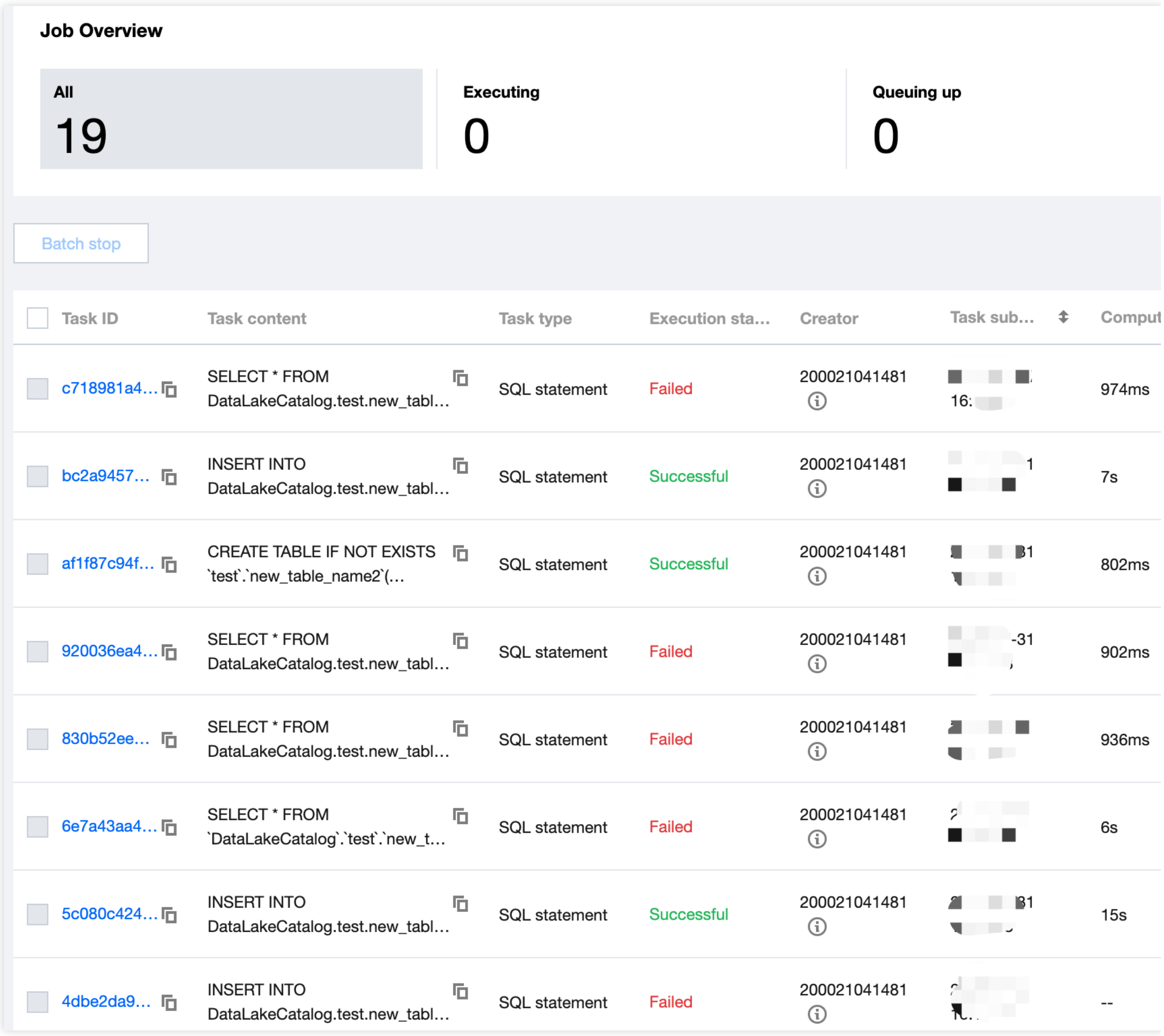
## Step V: Viewing Data Optimization Tasks

You can view data governance tasks in the DLC console under the **Data Operation and Maintenance** menu by navigating to the **Historical Tasks** page. You can query tasks using keywords such as CALL, Auto, database name, and table name.

**Note:**

To view system data optimization tasks, users need the permissions of DLC administrators.

Tasks with IDs starting with "Auto" are automatically generated data optimization tasks. As shown in the table below.



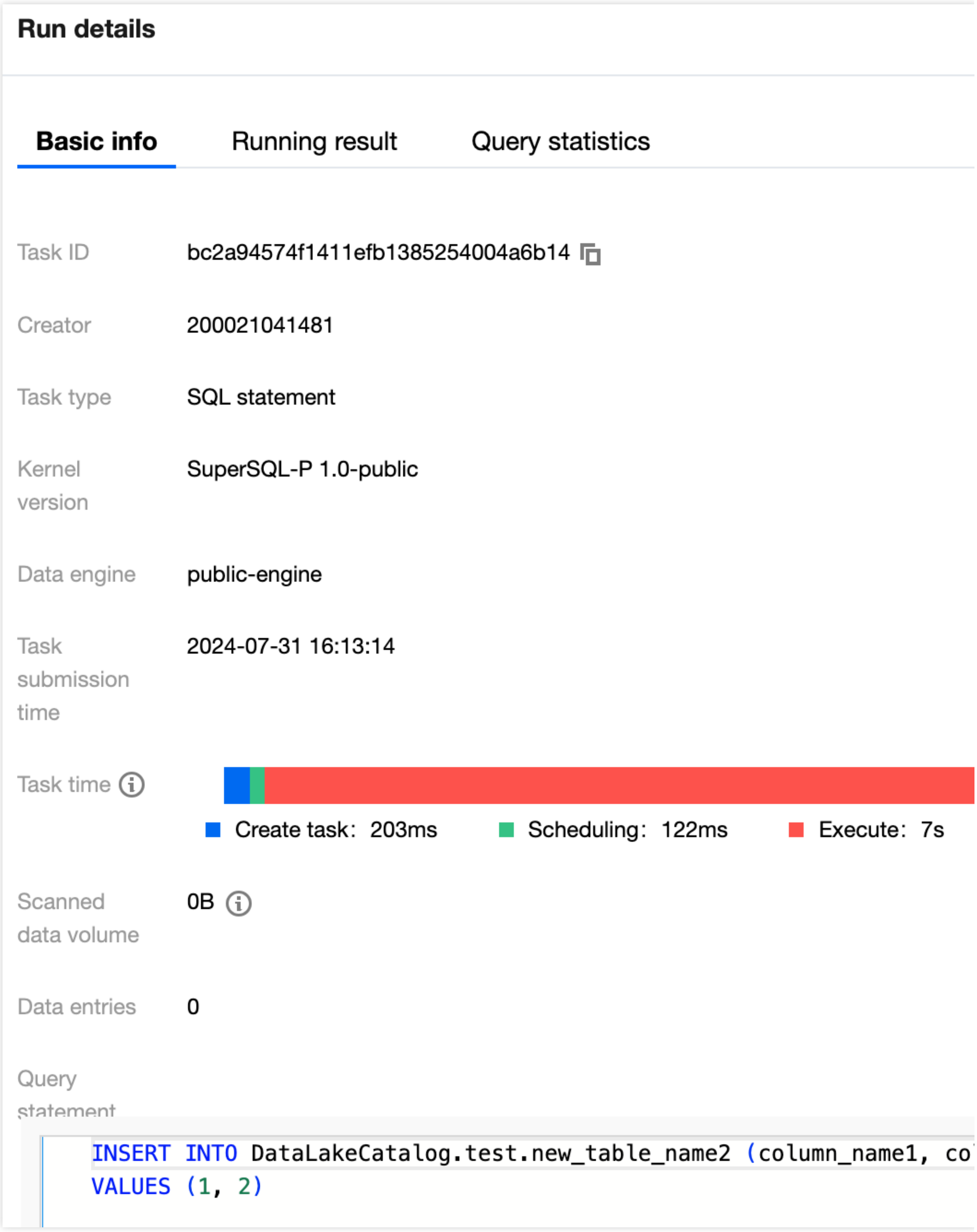
**Job Overview**

| All | Executing | Queuing up |
|---|---|---|
| 19 | 0 | 0 |

Batch stop

| Task ID | Task content | Task type | Execution sta... | Creator | Task sub... | Comput |
|---|---|---|---|---|---|---|
| c718981a4... | SELECT * FROM DataLakeCatalog.test.new_tabl... | SQL statement | Failed | 200021041481 | [redacted] | 974ms |
| bc2a9457... | INSERT INTO DataLakeCatalog.test.new_tabl... | SQL statement | Successful | 200021041481 | [redacted] | 7s |
| af1f87c94f... | CREATE TABLE IF NOT EXISTS \`test\`.\`new_table_name2\`(... | SQL statement | Successful | 200021041481 | [redacted] | 802ms |
| 920036ea4... | SELECT * FROM DataLakeCatalog.test.new_tabl... | SQL statement | Failed | 200021041481 | [redacted] | 902ms |
| 830b52ee... | SELECT * FROM DataLakeCatalog.test.new_tabl... | SQL statement | Failed | 200021041481 | [redacted] | 936ms |
| 6e7a43aa4... | SELECT * FROM \`DataLakeCatalog\`.\`test\`.\`new_t... | SQL statement | Failed | 200021041481 | [redacted] | 6s |
| 5c080c424... | INSERT INTO DataLakeCatalog.test.new_tabl... | SQL statement | Successful | 200021041481 | [redacted] | 15s |
| 4dbe2da9... | INSERT INTO DataLakeCatalog.test.new_tabl... | SQL statement | Failed | 200021041481 | [redacted] | -- |

You can also click **View Details** to check the basic information and results of running the tasks.



**Run details**

**Basic info** | Running result | Query statistics

| | |
|---|---|
| Task ID | bc2a94574f1411efb1385254004a6b14 |
| Creator | 200021041481 |
| Task type | SQL statement |
| Kernel version | SuperSQL-P 1.0-public |
| Data engine | public-engine |
| Task submission time | 2024-07-31 16:13:14 |
| Task time | 🟦 Create task: 203ms 🟩 Scheduling: 122ms 🟥 Execute: 7s |
| Scanned data volume | 0B |
| Data entries | 0 |
| Query statement | |

```
INSERT INTO DataLakeCatalog.test.new_table_name2 (column_name1, co
VALUES (1, 2)
```

# DLC Source Table Lake Ingestion Practice

Last updated：2024-07-31 17:34:58

## Use Cases

CDC (Change Data Capture) is an abbreviation for change data capture. It allows incremental changes in the source database to be synchronized in near real-time to other databases or applications. DLC supports using CDC technology to synchronize incremental changes from the source database to native DLC tables, completing the data lake ingestion.

## Prerequisites

DLC must be properly enabled, user permissions configured, and managed storage activated.
DLC database must be correctly created.
DLC database data optimization must be properly configured. For detailed configuration, see Enable data optimization.

## Ingesting Data into the Lake with InLong

DataInLong can be used to synchronize source data to DLC.

## Ingesting Stream Computing Data into the Lake with Oceanus

Source data can be synchronized to DLC via Oceanus.

## Ingesting Data into the Lake with Self-Managed Flink

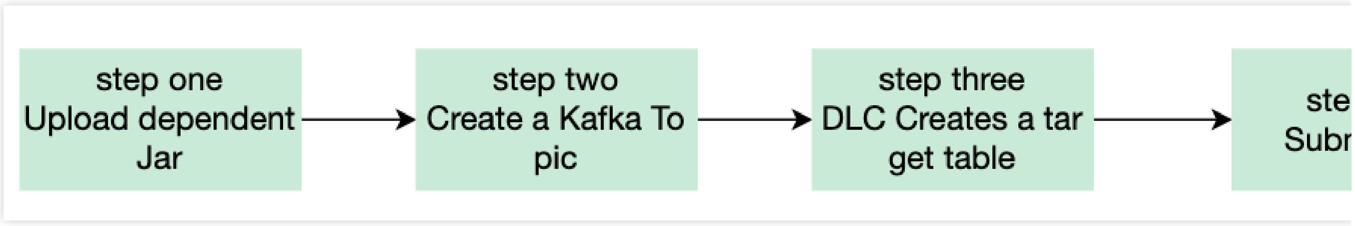
Flink can be used to synchronize source data to DLC. This example demonstrates how to synchronize data from a source Kafka to DLC, completing the data lake ingestion.

### Environment Preparation

Required clusters: Kafka 2.4.x, Flink 1.15.x, and Hadoop 3.x.
It is recommended to purchase EMR clusters for Kafka and Flink.

## Overall Operation Process

For detailed steps, see the diagram below:



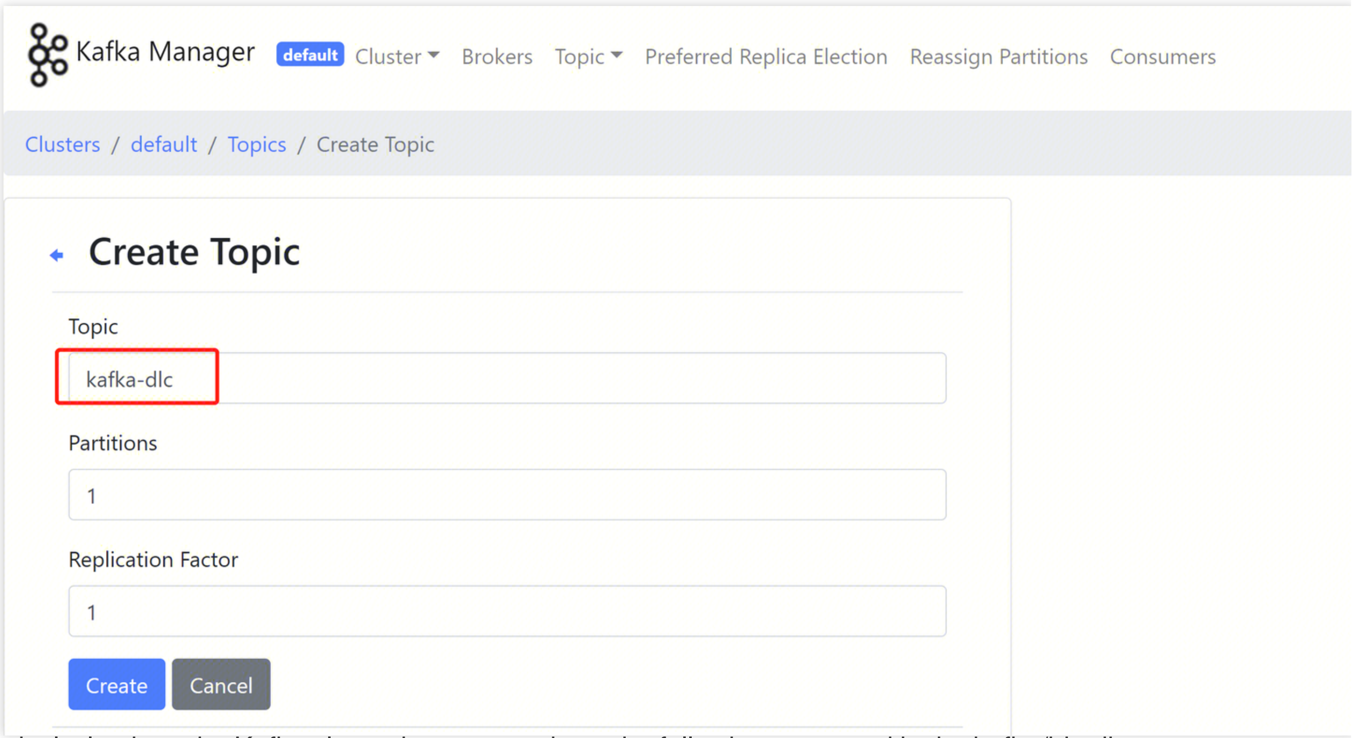
Step 1: Upload Required Jars: Upload the necessary Kafka, DLC connector Jar files, and Hadoop dependency Jars for synchronization.
Step 2: Create Kafka Topic: Create a Kafka topic for production and consumption.
Step 3: Create Target Table in DLC: Create a new target table in DLC data management.
Step 4: Submit Task: Submit the synchronization task in the Flink cluster.
Step 5: Send Message Data and Check Sync Results: Send message data through the Kafka cluster and check the synchronization results on the DLC.

### Step 1: Uploading Required Jars

1. Download required Jars.
It is recommended to upload the required Jars that match the version of Flink you are using. For example, if you are using Flink 1.15.x, download the flink-sql-connect-kafka-1.15.x.jar. See the attachments for the relevant files.
Kafka-related dependencies: flink-sql-connect-kafka-1.15.4.jar
DLC-related dependencies: sort-connector-iceberg-dlc-1.6.0.jar
Hadoop 3.x related dependencies: api-util-1.0.0-M20.jar, guava-27.0-jre.jar, hadoop-mapreduce-client-core-3.2.2.jar.
2. Log in to the Flink cluster and upload the prepared Jar files to the flink/ib directory.

### Step 2: Creating a Kafka Topic

Log in to Kafka Manager, click on **default cluster**, then click on **Topic > Create**.
Topic name: For this example, enter kafka_dlc
Number of partitions: 1
Number of replicas: 1



Topic: kafka-dlc

Partitions: 1

Replication Factor: 1

Alternatively, log in to the Kafka cluster instance and use the following command in the kafka/bin directory to create the Topic.

```
./kafka-topics.sh --bootstrap-server ip:port --create  --topic kafka-dlc
```

### Step 3: Creating a New Target Table in DLC

For details on creating a new target table, see DLC Native Table Operation Configuration.

### Step 4: Submitting the Task

There are two ways to synchronize data, i.e. using Flink: Flink SQL Write Mode and Flink Stream API. Both synchronization methods will be introduced below.

Before submitting the task, you need to create a directory to save checkpoint data. Use the following command to create the data management.
Create the hdfs  /flink/checkpoints directory:

```
hadoop fs -mkdir /flink
hadoop fs -mkdir /flink/checkpoints
```

**Flink SQL Synchronization Mode**

1. Create a new Maven project named "flink-demo" in IntelliJ IDEA.

2. Add the necessary dependencies in pom. For details on the dependencies, see Complete Sample Code Reference > Example 1.
3. Java synchronization code: The core code is shown in the steps below. For detailed code, see Complete Sample Code Reference > Example 2.
Create execution environment and configure checkpoint:

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment
env.setParallelism(1);
env.enableCheckpointing(60000);
env.getCheckpointConfig().setCheckpointStorage("hdfs:///flink/checkpoints");
env.getCheckpointConfig().setCheckpointTimeout(60000);
env.getCheckpointConfig().setTolerableCheckpointFailureNumber(5);
env.getCheckpointConfig().enableExternalizedCheckpoints(CheckpointConfig.Externaliz
```

Execute Source SQL:

```
  tEnv.executeSql(sourceSql);
```

Execute Synchronization SQL:

```
tEnv.executeSql(sql)
```

4. Use IntelliJ IDEA to compile and package the flink-demo project. The JAR file flink-demo-1.0-SNAPSHOT.jar will be generated in the project's target folder.
5. Log in to one of the instances in the Flink cluster and upload flink-demo-1.0-SNAPSHOT.jar to the /data/jars/ directory (create the directory if it does not exist).
6. Log in to one of the instances in the Flink cluster and execute the following command in the flink/bin directory to submit the synchronization task.

```
./flink run  --class com.tencent.dlc.iceberg.flink.AppendIceberg /data/jars/flink-d
```

**Flink Stream API Synchronization Mode**

1. Create a new Maven project named "flink-demo" in IntelliJ IDEA.
2. Add the necessary dependencies in pom: Complete sample code reference > Example 3.
3. Java synchronization code: The core code is shown in the steps below. For detailed code, see Complete sample code reference > Example 4.

Create the execution environment StreamTableEnvironment and configure checkpoint:

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment
env.setParallelism(1);
env.enableCheckpointing(60000);
env.getCheckpointConfig().setCheckpointStorage("hdfs:///data/checkpoints");
env.getCheckpointConfig().setCheckpointTimeout(60000);
env.getCheckpointConfig().setTolerableCheckpointFailureNumber(5);
env.getCheckpointConfig().enableExternalizedCheckpoints(CheckpointConfig.Externaliz
```

Get the Kafka input stream:

```
KafkaToDLC dlcSink = new KafkaToDLC();
DataStream<RowData> dataStreamSource = dlcSink.buildInputStream(env);
```

Configure Sink:

```
FlinkSink.forRowData(dataStreamSource)
           .table(table)
           .tableLoader(tableLoader)
           .equalityFieldColumns(equalityColumns)
           .metric(params.get(INLONG_METRIC.key()), params.get(INLONG_AUDIT.key()))
           .action(actionsProvider)
           .tableOptions(Configuration.fromMap(options))
           // It is false by default, which appends data. If it is set to be true, t
           .overwrite(false)
           .append();
```

Execute Synchronization SQL:

```
env.execute("DataStream Api Write Data To Iceberg");
```

4. Use IntelliJ IDEA to compile and package the flink-demo project. The JAR packet, flink-demo-1.0-SNAPSHOT.jar, will be generated in the project's target folder.
5. Log in to one of the instances in the Flink cluster and upload flink-demo-1.0-SNAPSHOT.jar to the /data/jars/ directory (create the directory if it does not exist).
6. Log in to one of the instances in the Flink cluster and execute the following command in the flink/bin directory to submit the task.

```
./flink run  --class com.tencent.dlc.iceberg.flink.AppendIceberg /data/jars/flink-d
```

**Step 5: Send Message Data and Query Synchronization Results**

1. Log in to the Kafka cluster instance, navigate to the kafka/bin directory, and use the following command to send message data.

```
./kafka-console-producer.sh --broker-list 122.152.227.141:9092 --topic kafka-dlc
```

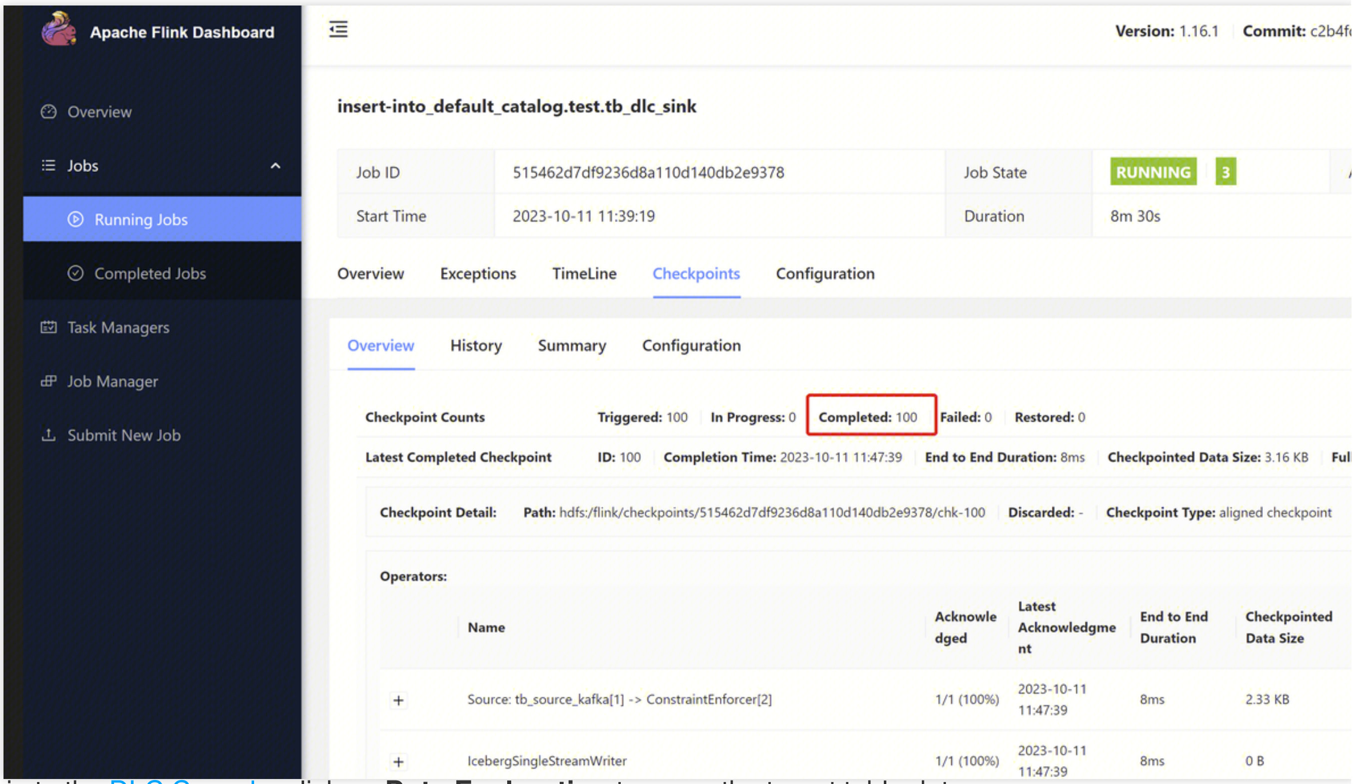
The data information is as follows:

```
{"id":1,"name":"Zhangsan","age":18}
{"id":2,"name":"Lisi","age":19}
{"id":3,"name":"Wangwu","age":20}
{"id":4,"name":"Lily","age":21}
{"id":5,"name":"Lucy","age":22}
{"id":6,"name":"Huahua","age":23}
{"id":7,"name":"Wawa","age":24}
{"id":8,"name":"Mei","age":25}
{"id":9,"name":"Joi","age":26}
{"id":10,"name":"Qi","age":27}
{"id":11,"name":"Ky","age":28}
```

```
{"id":12,"name":"Mark","age":29}
```

2. Query synchronization results

Open the Flink Dashboard, and click on **Running Job** > **Run Job** > **Checkpoint** > **Overview** to view the Job synchronization results.



**Version:** 1.16.1 **Commit:** c2b4f

insert-into_default_catalog.test.tb_dlc_sink

| | | | |
|---|---|---|---|
| Job ID | 515462d7df9236d8a110d140db2e9378 | Job State | RUNNING 3 |
| Start Time | 2023-10-11 11:39:19 | Duration | 8m 30s |

**Checkpoint Counts** **Triggered:** 100 **In Progress:** 0 **Completed:** 100 **Failed:** 0 **Restored:** 0

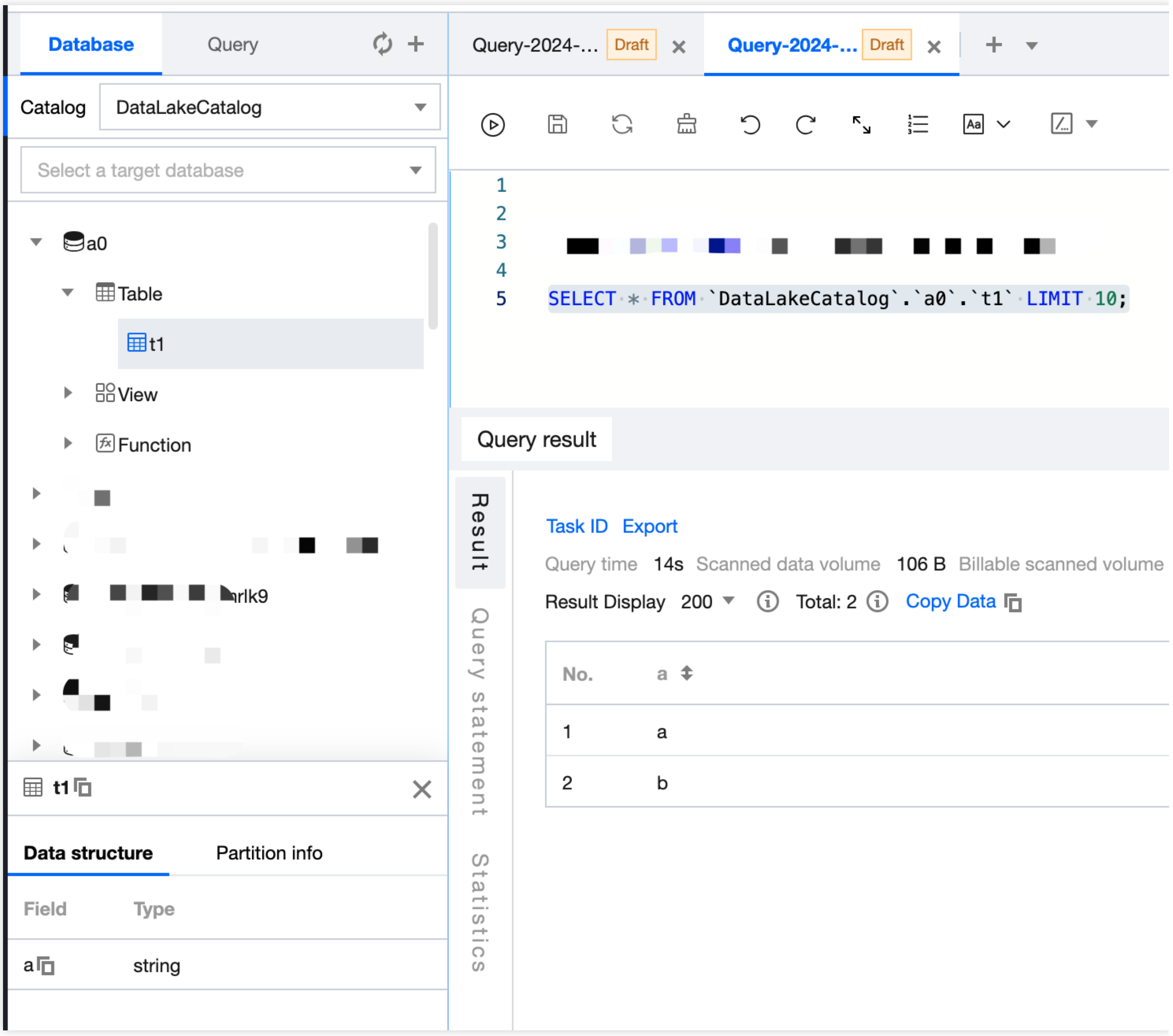
**Latest Completed Checkpoint** **ID:** 100 **Completion Time:** 2023-10-11 11:47:39 **End to End Duration:** 8ms **Checkpointed Data Size:** 3.16 KB **Ful**

**Checkpoint Detail:** **Path:** hdfs:/flink/checkpoints/515462d7df9236d8a110d140db2e9378/chk-100 **Discarded:** - **Checkpoint Type:** aligned checkpoint

**Operators:**

| | Name | Acknowledged | Latest Acknowledgment | End to End Duration | Checkpointed Data Size |
|---|---|---|---|---|---|
| + | Source: tb_source_kafka[1] -> ConstraintEnforcer[2] | 1/1 (100%) | 2023-10-11 11:47:39 | 8ms | 2.33 KB |
| + | IcebergSingleStreamWriter | 1/1 (100%) | 2023-10-11 11:47:39 | 8ms | 0 B |

3. Log in to the DLC Console, click on **Data Exploration** to query the target table data.



```
SELECT * FROM `DataLakeCatalog`.`a0`.`t1` LIMIT 10;
```

| Field | Type |
|---|---|
| a | string |

Query time 14s Scanned data volume 106 B Billable scanned volume

Result Display 200 Total: 2

| No. | a |
|---|---|
| 1 | a |
| 2 | b |

## Complete Sample Code Reference Example

**Note:**

Data marked with "*****" in the examples should be replaced with actual data used during development.

### Example 1

```
<properties>
  <flink.version>1.15.4</flink.version>
  <cos.lakefs.plugin.version>1.0</cos.lakefs.plugin.version>
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
```

```
      </dependency>
      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-java</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
      </dependency>

      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-clients</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
      </dependency>

      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-streaming-java</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
      </dependency>

      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-connector-kafka</artifactId>
        <version>${flink.version}</version>
      </dependency>

      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-table-planner_2.12</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
      </dependency>

      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-json</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
      </dependency>
      <dependency>
        <groupId>com.qcloud.cos</groupId>
        <artifactId>lakefs-cloud-plugin</artifactId>
        <version>${cos.lakefs.plugin.version}</version>
        <exclusions>
          <exclusion>
```

```
            <groupId>com.tencentcloudapi</groupId>
            <artifactId>tencentcloud-sdk-java</artifactId>
          </exclusion>
        </exclusions>
      </dependency>
    </dependencies>
```

**Example 2**

```
public class AppendIceberg {

    public static void main(String[] args) {
        // Create execution environment and configure the checkpoint
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnv
        env.setParallelism(1);
        env.enableCheckpointing(60000);
        env.getCheckpointConfig().setCheckpointStorage("hdfs:///flink/checkpoints")
        env.getCheckpointConfig().setCheckpointTimeout(60000);
        env.getCheckpointConfig().setTolerableCheckpointFailureNumber(5);
        env.getCheckpointConfig()
                .enableExternalizedCheckpoints(CheckpointConfig.ExternalizedCheckpo
        EnvironmentSettings settings = EnvironmentSettings
                .newInstance()
                .inStreamingMode()
                .build();
        StreamTableEnvironment tEnv = StreamTableEnvironment.create(env, settings);

        // Create the input table
        String sourceSql = "CREATE TABLE tb_kafka_sr ( \\n"
                + "   id INT, \\n"
                + "   name STRING, \\n"
                + "   age INT \\n"
                + ") WITH ( \\n"
                + "  'connector' = 'kafka', \\n"
                + "  'topic' = 'kafka_dlc', \\n"
                + "  'properties.bootstrap.servers' = '10.0.126.***:9092', \\n" //
                + "  'properties.group.id' = 'test-group', \\n"
                + "  'scan.startup.mode' = 'earliest-offset', \\n"  // start from t
                + "  'format' = 'json' \\n"
                + ");";
        tEnv.executeSql(sourceSql);

        // Create the output table
        String sinkSql = "CREATE TABLE tb_dlc_sk ( \\n"
                + "  id INT PRIMARY KEY NOT ENFORCED, \\n"
                + "  name STRING,\\n"
                + "  age INT\\n"
                + ") WITH (\\n"
                + "  'qcloud.dlc.managed.account.uid' = '1000***79117',\\n" //User
                + "  'qcloud.dlc.secret-id' = 'AKIDwjQvBHCsKYXL3***pMdkeMsBH8lAJEt'
                + "  'qcloud.dlc.region' = 'ap-***',\\n" // Database and table regi
                + "  'qcloud.dlc.user.appid' = 'AKIDwjQvBHCsKYXL3***pMdkeMsBH8lAJEt
                + "  'qcloud.dlc.secret-key' = 'kFWYQ5WklaCYgbLtD***cyAD7sUyNiVP',\
                + "  'connector' = 'iceberg-inlong', \\n"
                + "  'catalog-database' = 'test_***', \\n" // Target database
                + "  'catalog-table' = 'kafka_dlc', \\n" // Target data table
                + "  'default-database' = 'test_***', \\n" //Default database
                + "  'catalog-name' = 'HYBRIS', \\n"
                + "  'catalog-impl' = 'org.apache.inlong.sort.iceberg.catalog.hybri
                + "  'uri' = 'dlc.tencentcloudapi.com', \\n"
                + "  'fs.cosn.credentials.provider' = 'org.apache.hadoop.fs.auth.Dl
                + "  'qcloud.dlc.endpoint' = 'dlc.tencentcloudapi.com', \\n"
                + "  'fs.lakefs.impl' = 'org.apache.hadoop.fs.CosFileSystem', \\n"
                + "  'fs.cosn.impl' = 'org.apache.hadoop.fs.CosFileSystem', \\n"
                + "  'fs.cosn.userinfo.region' = 'ap-guangzhou', \\n" // Region inf
                + "  'fs.cosn.userinfo.secretId' = 'AKIDwjQvBHCsKYXL3***pMdkeMsBH8l
                + "  'fs.cosn.userinfo.secretKey' = 'kFWYQ5WklaCYgbLtD***cyAD7sUyNi
                + "  'service.endpoint' = 'dlc.tencentcloudapi.com', \\n"
                + "  'service.secret.id' = 'AKIDwjQvBHCsKYXL3***pMdkeMsBH8lAJEt', \
                + "  'service.secret.key' = 'kFWYQ5WklaCYgbLtD***cyAD7sUyNiVP', \\n
                + "  'service.region' = 'ap-***', \\n"  // Database and table regio
                + "  'user.appid' = '1305424723', \\n"
                + "  'request.identity.token' = '1000***79117', \\n"
                + "  'qcloud.dlc.jdbc.url'='jdbc:dlc:dlc.internal.tencentcloudapi.c
                + ");";
        tEnv.executeSql(sinkSql);
        // Execute computation and output results
        String sql = "insert into tb_dlc_sk select * from tb_kafka_sr";
        tEnv.executeSql(sql);
    }

}
```

**Example 3**

```
<properties>
  <flink.version>1.15.4</flink.version>
</properties>

<dependencies>
  <dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>2.0.22</version>
    <scope>provided</scope>
  </dependency>
```

```
      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-java</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
      </dependency>

      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-clients</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
      </dependency>

      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-streaming-java</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
      </dependency>

      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-connector-kafka</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
      </dependency>

      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-table-planner_2.12</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
      </dependency>
      <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-json</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
      </dependency>
      <dependency>
        <groupId>org.apache.inlong</groupId>
        <artifactId>sort-connector-iceberg-dlc</artifactId>
        <version>1.6.0</version>
        <scope>system</scope>
        <systemPath>${project.basedir}/lib/sort-connector-iceberg-dlc-1.6.0.jar</syst
      </dependency>
      <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>${kafka-version}</version>
        <scope>provided</scope>
      </dependency>
      <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.25</version>
      </dependency>
      <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.25</version>
      </dependency>
    </dependencies>
```

**Example 4**

```
public class KafkaToDLC {

    public static void main(String[] args) throws Exception {
        final MultipleParameterTool params = MultipleParameterTool.fromArgs(args);
        final Map<String, String> options = setOptions();
        //1. Create the execution environment StreamTableEnvironment and configure
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnv
        env.setParallelism(1);
        env.enableCheckpointing(60000);
        env.getCheckpointConfig().setCheckpointStorage("hdfs:///data/checkpoints");
        env.getCheckpointConfig().setCheckpointTimeout(60000);
        env.getCheckpointConfig().setTolerableCheckpointFailureNumber(5);
        env.getCheckpointConfig()
                .enableExternalizedCheckpoints(CheckpointConfig.ExternalizedCheckpo
        env.getConfig().setGlobalJobParameters(params);

        //2. Get input stream
        KafkaToDLC dlcSink = new KafkaToDLC();
        DataStream<RowData> dataStreamSource = dlcSink.buildInputStream(env);

        //3. Create Hadoop configuration and Catalog configuration
        CatalogLoader catalogLoader = FlinkDynamicTableFactory.createCatalogLoader(
        TableLoader tableLoader = TableLoader.fromCatalog(catalogLoader,
                TableIdentifier.of(params.get(CATALOG_DATABASE.key()), params.get(C
        tableLoader.open();
        Table table = tableLoader.loadTable();
        ActionsProvider actionsProvider = FlinkDynamicTableFactory.createActionLoad
                Thread.currentThread().getContextClassLoader(), options);
        //4. Create Schema
        Schema schema = Schema.newBuilder()
                .column("id", DataTypeUtils.toInternalDataType(new IntType(false)))
                .column("name", DataTypeUtils.toInternalDataType(new VarCharType())
                .column("age", DataTypeUtils.toInternalDataType(new DateType(false)
                .primaryKey("id")
                .build();
        List<String> equalityColumns = schema.getPrimaryKey().get().getColumnNames(
        //5. Configure Slink
        FlinkSink.forRowData(dataStreamSource)
                //This .table can be omitted; just specify the corresponding path f
                .table(table)
                .tableLoader(tableLoader)
                .equalityFieldColumns(equalityColumns)
                .metric(params.get(INLONG_METRIC.key()), params.get(INLONG_AUDIT.ke
                .action(actionsProvider)
                .tableOptions(Configuration.fromMap(options))
                //It is false by default, which appends data. If it is set to be tr
                .overwrite(false)
                .append();
        //6. Execute synchronization
        env.execute("DataStream Api Write Data To Iceberg");
    }

    private static Map<String, String> setOptions() {
        Map<String, String> options = new HashMap<>();
        options.put("qcloud.dlc.managed.account.uid", "1000***79117"); //User Uid
        options.put("qcloud.dlc.secret-id", "AKIDwjQvBHCsKYXL3***pMdkeMsBH8lAJEt");
        options.put("qcloud.dlc.region", "ap-***"); // Database and table region in
        options.put("qcloud.dlc.user.appid", "AKIDwjQvBHCsKYXL3***pMdkeMsBH8lAJEt")
        options.put("qcloud.dlc.secret-key", "kFWYQ5WklaCYgbLtD***cyAD7sUyNiVP"); /
        options.put("connector", "iceberg-inlong");
        options.put("catalog-database", "test_***"); // Target database
        options.put("catalog-table", "kafka_dlc"); // Target data table
>       options.put("default-database", "test_***"); //Default database
        options.put("catalog-name", "HYBRIS");
        options.put("catalog-impl", "org.apache.inlong.sort.iceberg.catalog.hybris.
        options.put("uri", "dlc.tencentcloudapi.com");
        options.put("fs.cosn.credentials.provider", "org.apache.hadoop.fs.auth.DlcC
        options.put("qcloud.dlc.endpoint", "dlc.tencentcloudapi.com");
        options.put("fs.lakefs.impl", "org.apache.hadoop.fs.CosFileSystem");
        options.put("fs.cosn.impl", "org.apache.hadoop.fs.CosFileSystem");
        options.put("fs.cosn.userinfo.region", "ap-guangzhou"); // Region informati
        options.put("fs.cosn.userinfo.secretId", "AKIDwjQvBHCsKYXL3***pMdkeMsBH8lAJ
        options.put("fs.cosn.userinfo.secretKey", "kFWYQ5WklaCYgbLtD***cyAD7sUyNiVP
        options.put("service.endpoint", "dlc.tencentcloudapi.com");
        options.put("service.secret.id", "AKIDwjQvBHCsKYXL3***pMdkeMsBH8lAJEt"); //
        options.put("service.secret.key", "kFWYQ5WklaCYgbLtD***cyAD7sUyNiVP"); // U
        options.put("service.region", "ap-***");  // Database and table region info
        options.put("user.appid", "1305***23");
        options.put("request.identity.token", "1000***79117");
        options.put("qcloud.dlc.jdbc.url",
                "jdbc:dlc:dlc.internal.tencentcloudapi.com?task_type,SparkSQLTask&d
        return options;
    }

    /**
     * Create the input stream
     *
     * @param env
     * @return
     */
    private DataStream<RowData> buildInputStream(StreamExecutionEnvironment env) {
        //1. Configure the execution environment
        EnvironmentSettings settings = EnvironmentSettings
                .newInstance()
                .inStreamingMode()
                .build();
        StreamTableEnvironment sTableEnv = StreamTableEnvironment.create(env, setti
        org.apache.flink.table.api.Table table = null;
        //2. Execute SQL to get the data input stream
        try {
            sTableEnv.executeSql(createTableSql()).print();
            table = sTableEnv.sqlQuery(transformSql());
            DataStream<Row> rowStream = sTableEnv.toChangelogStream(table);
            DataStream<RowData> rowDataDataStream = rowStream.map(new MapFunction<R
                @Override
                public RowData map(Row rows) throws Exception {
                    GenericRowData rowData = new GenericRowData(3);
                    rowData.setField(0, rows.getField(0));
                    rowData.setField(1, (String) rows.getField(1));
                    rowData.setField(2, rows.getField(2));
                    return rowData;
                }
            });
            return rowDataDataStream;
        } catch (Exception e) {
            throw new RuntimeException("kafka to dlc transform sql execute error.",
        }
    }

    private String createTableSql() {
        String tableSql = "CREATE TABLE tb_kafka_sr ( \\n"
                + "   id INT, \\n"
                + "   name STRING, \\n"
                + "   age INT \\n"
                + ") WITH ( \\n"
                + "  'connector' = 'kafka', \\n"
                + "  'topic' = 'kafka_dlc', \\n"
                + "  'properties.bootstrap.servers' = '10.0.126.30:9092', \\n"
                + "  'properties.group.id' = 'test-group-10001', \\n"
                + "  'scan.startup.mode' = 'earliest-offset', \\n"
                + "  'format' = 'json' \\n"
                + ");";
        return tableSql;
    }

    private String transformSql() {
        String transformSQL = "select * from tb_kafka_sr";
        return transformSQL;
    }
}
```

# DLC Source Table FAQs

Last updated：2024-07-31 17:35:14

### Why Must Data Optimization Be Enabled for Upsert Write Scenarios in DLC Native Table (Iceberg)?

1. DLC Native Table (Iceberg) uses the MOR (Merge On Read) table format. When Upsert writes occur upstream, updates write a delete file marking a record as deleted and then add a new data file to the new modification record.
2. Without committing and merging, the job engine needs to merge the original data it has read, the delete file, and the new data file when reading data to get the latest data. This will lead the job engine to consume significant resources and time. Small file merging in data optimization reads and merges these files in advance, writing them into new data files so that the job engine can directly read the latest files without needing to merge data files.
3. DLC metadata (Iceberg) uses a snapshot mechanism, and even if new snapshots are generated during the write, historical snapshots are not cleaned up. The snapshot expiration capability of the data optimization can remove old snapshots, freeing up storage space and preventing unused historical data from occupying storage space.

### How to Handle Timeout in Data Optimization Tasks?

The system sets a default timeout for running data optimization tasks (2 hours by default) to prevent a task from occupying resources for too long and hindering other tasks. When the timeout expires, the system cancels the optimization task. According to different types of tasks, see the following handling procedures.
1. If small file merge tasks frequently time out, it indicates data accumulation and that current resources are insufficient for merging. Temporarily expanding resources (or setting the table to use dedicated optimization resources) can address the accumulated data, and then revert the settings.
2. If small file merge tasks occasionally time out, it may indicate insufficient optimization resources. Consider scaling-out data resources to some extent and monitoring if there are timeouts in subsequent governance tasks of multiple cycles. Occasional small file merge timeouts will not immediately impact query performance but may lead to continuous timeouts and eventually affect query performance if the issue is not addressed timely. DLC enables segmented submissions for small file merges by default, so parts of the finished task can still be submitted successfully and are still effective.
3. If a snapshot expiration task times out, it occurs in two stages. In the first stage, the snapshot is removed from the metadata, and this process usually does not time out. In the second stage, the data files associated with the removed snapshot are deleted from storage. This stage requires individually comparing files to be deleted. There might be timeouts if there are many files to be deleted. Timeouts for this type of task can be ignored. Files that were not deleted due to the timeout will be treated as orphan files and will be cleaned up in subsequent orphaned file removal processes.
4. If orphan file removal tasks time out, the handling of orphan files is similar to removing orphan files. As long as the deleted files are still valid when scanned, the system will continue to scan and execute in subsequent cycles, as orphan file removal is a periodic task. If a task times out, it will be retried in the next cycle.

## Why Does Iceberg Occasionally Read an Old Snapshot Shortly after Inserting Data?

1. Iceberg provides a default caching capability for the catalog, with a default duration of 30 seconds. In extreme cases, if two queries for the same table occur very close together in time and are not executed in the same session, there is a very low probability that the query will access the previous snapshot before the cache expires and updates are fetched.
2. The Iceberg community recommends enabling this parameter. DLC also enabled it by default in earlier versions to speed up task execution and reduce visits to metadata during queries. However, if two tasks have very close read and write intervals, the described situation may occur in extreme cases.
3. In the latest versions of the DLC engine, this parameter is disabled by default. When it comes to the scenes users may encounter, if users who purchased the engine before January 2024 need to ensure strong data consistency in queries, they can manually disable this parameter by following the configuration method below to modify the engine parameters:

```
"spark.sql.catalog.DataLakeCatalog.cache-enabled": "false"
"spark.sql.catalog.DataLakeCatalog.cache.expiration-interval-ms": "0"
```

### Why Should DLC Native Table (Iceberg) Be Partitioned?

1. Data optimization jobs are first divided by partitions. If the native table (Iceberg) has no partitions, most small file merges that involve modifying tables will only have a single job operate. Therefore, the merges cannot be parallel, and this significantly reduces merge efficiency.

2. If the Table Has No Upstream Partition Fields, How Can It Be Partitioned? In this case, consider using Iceberg's bucket partitioning. For detailed description, see DLC Native Table Core Capabilities.

### How to Handle Write Conflicts in DLC Native Table (Iceberg)?

1. To ensure ACID compliance, Iceberg checks the current view for changes during commits. If changes are detected, a conflict is assumed. Then, the commit operation is rolled back. The current view is merged, and the commit is retried.
2. The system provides default retry counts and intervals for conflicts. If multiple commit attempts still result in conflicts, the write operation fails. For default conflict parameters, see DLC Native Table Core Capabilities.
3. If conflicts occur, users can adjust the number and interval of retries. The following example sets the number of conflict retries to 10. For more details on parameter meanings, see DLC Native Table Core Capabilities.

```
// Set conflict retry count to 10

ALTER TABLE `DataLakeCatalog`.`axitest`.`upsert_case` SET TBLPROPERTIES('commit.ret
```

## The DLC Native Table (Iceberg) has Been Deleted, But Why Is The Storage Space Capacity Not Released?

When the DLC native table (Iceberg) is dropped, the metadata is deleted immediately, and the data is deleted asynchronously. The data is first moved to the recycle bin directory, and the data is removed from the storage one day later.