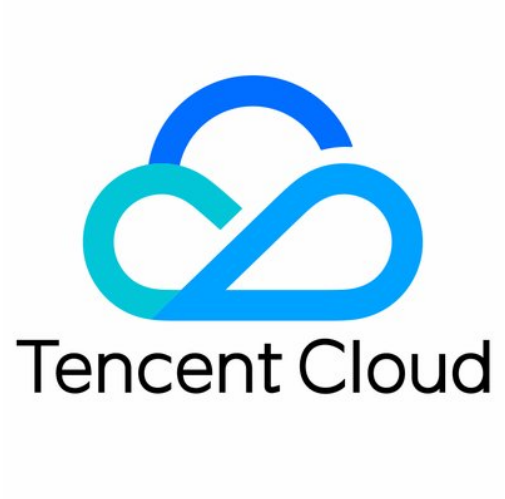
Data Lake Compute

Practical Tutorial

Product Documentation



Contents

Practical Tutorial
Table Creation Practice

Last updated：2025-03-12 18:15:01

DLC (Data Lake Compute) supports creating native tables (Iceberg) and external tables in various scenarios. For Table creation, refer to the following practice cases.

Create Native Table (Iceberg)

Spark ETL Scenarios

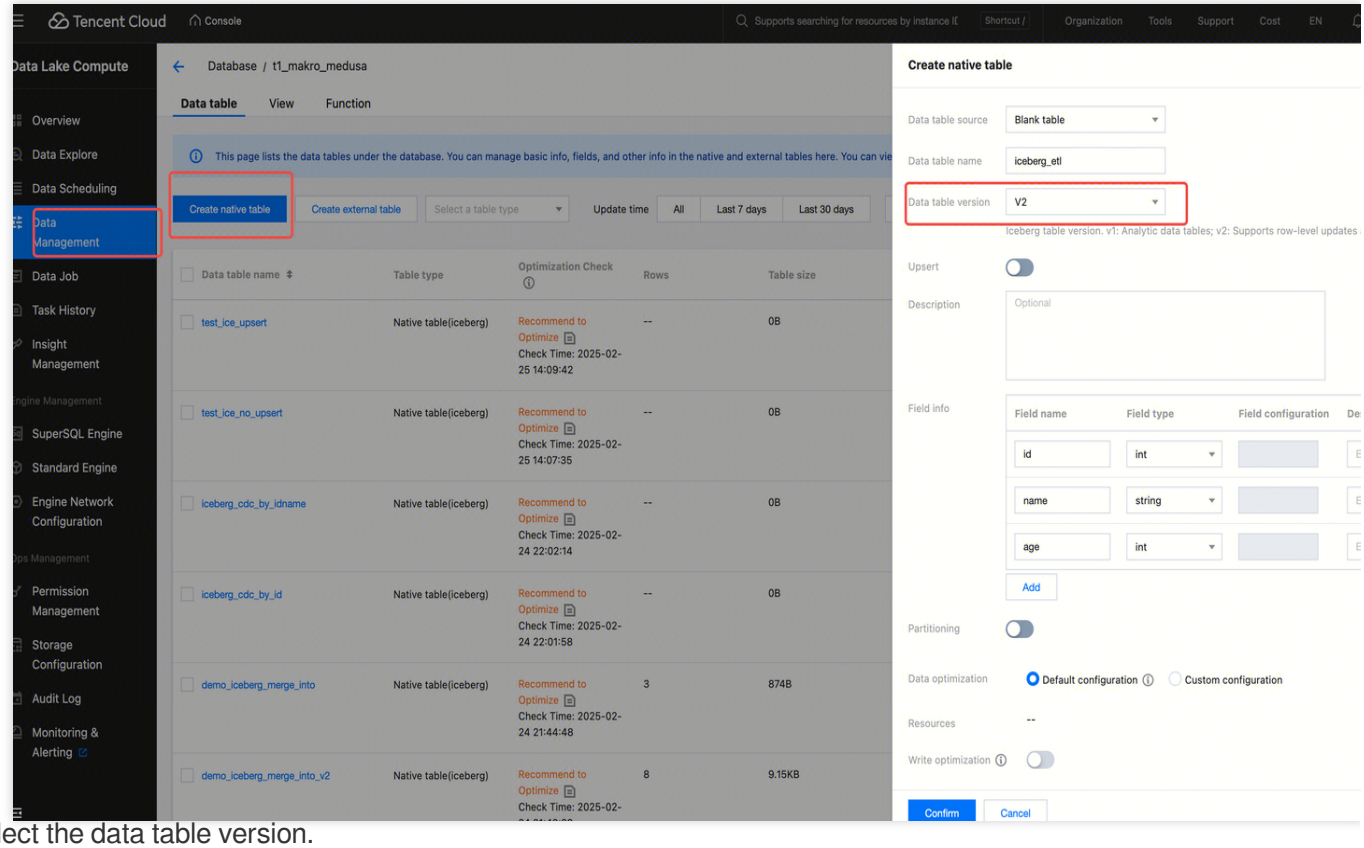
Applicable to: Periodically perform batch job operations such as insert into, insert overwrite, merge into, etc.

```
/**
Copy-on-write is the default mode. If it cannot be determined which one of the two
Applicable scenarios of copy-on-write: Featuring relatively higher query performanc
Applicable scenarios of merge-on-read: Featuring relatively lower query performance
/** Copy-on-write table */
CREATE TABLE dlc_db.iceberg_etl (
      id          INT,
      name        string,
      age         INT
  ) TBLPROPERTIES (
    'format-version' = '2',
    'write.metadata.previous-versions-max' = '100',
    'write.metadata.delete-after-commit.enabled' = 'true');

/** merge on read table */
CREATE TABLE dlc_db.iceberg_etl (
      id          INT,
      name        string,
      age         INT
  ) TBLPROPERTIES (
    'format-version' = '2',
    'write.metadata.previous-versions-max' = '100',
    'write.metadata.delete-after-commit.enabled' = 'true',
    'write.update.mode' = 'merge-on-read',
    'write.merge.mode' = 'merge-on-read',
    'write.delete.mode' = 'merge-on-read'
  );
```

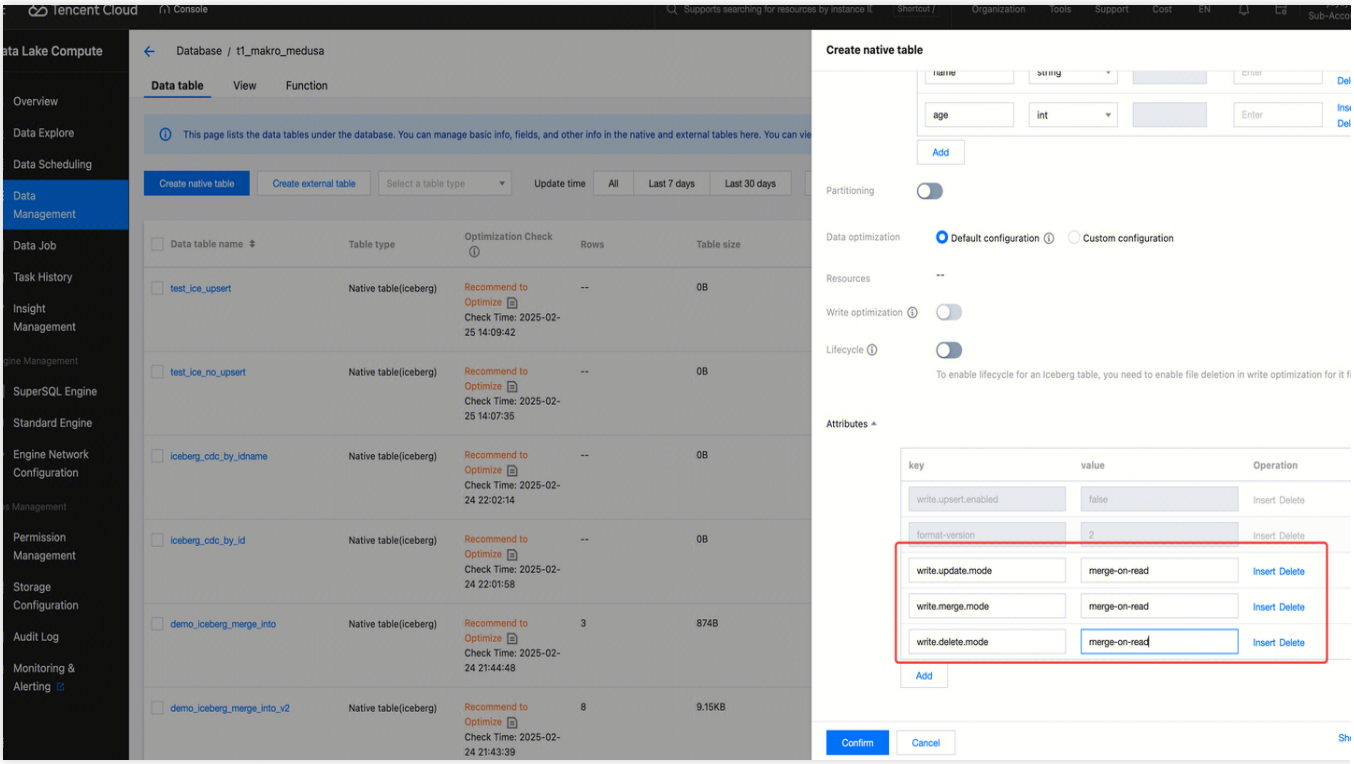
Console Creation: Copy-On-Write Mode

1. lick to create native tables.



2. Select the data table version.

Console Creation: Merge-On-Read Mode (Requires Adding Three Additional Attribute Values)



Flink Stream Writing Scenarios

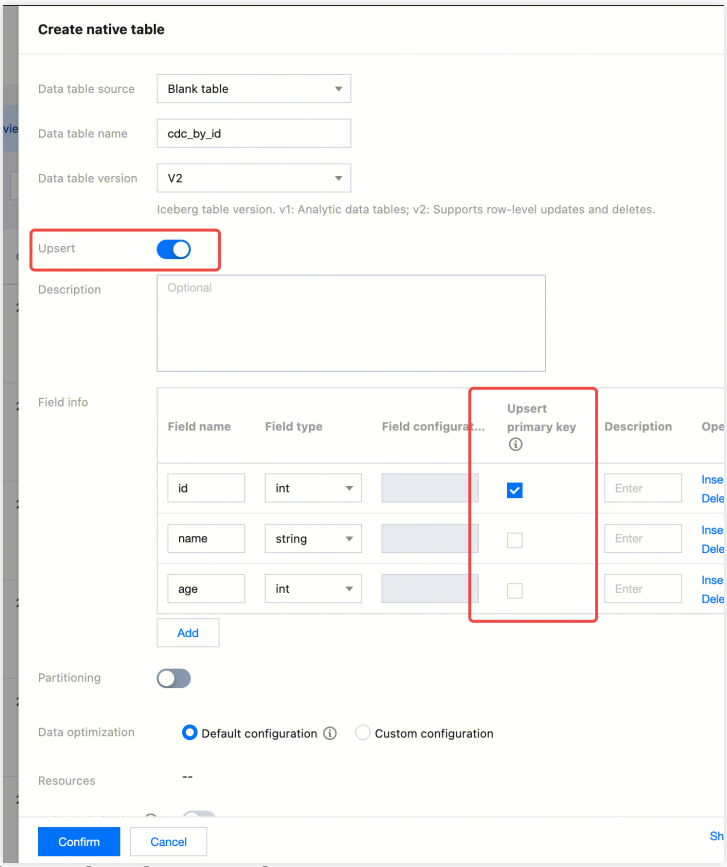
Suitable for Oceanus (Flink stream writing) scenarios.

```
/** The Flink stream writing primary key is the ID */
CREATE TABLE dlc_db.iceberg_cdc_by_id (
      id          INT,
      name        string,
      age         INT
  ) TBLPROPERTIES (
    'format-version' = '2',
    'write.metadata.previous-versions-max' = '100',
    'write.metadata.delete-after-commit.enabled' = 'true',
    'write.upsert.enabled' = 'true',
    'write.update.mode' = 'merge-on-read',
    'write.merge.mode' = 'merge-on-read',
    'write.delete.mode' = 'merge-on-read',
    'write.distribution-mode' = 'hash',
    'write.parquet.bloom-filter-enabled.column.id' = 'true',
    'dlc.ao.data.govern.sorted.keys' = 'id'
  );

/** The Flink stream writing primary key is a composite primary key of ID and name
CREATE TABLE dlc_db.iceberg_cdc_by_id_and_name (
      id          INT,
      name        string,
      age         INT
  ) TBLPROPERTIES (
```

```
    'format-version' = '2',
    'write.metadata.previous-versions-max' = '100',
    'write.metadata.delete-after-commit.enabled' = 'true',
    'write.upsert.enabled' = 'true',
    'write.update.mode' = 'merge-on-read',
    'write.merge.mode' = 'merge-on-read',
    'write.delete.mode' = 'merge-on-read',
    'write.distribution-mode' = 'hash',
    'write.parquet.bloom-filter-enabled.column.id' = 'true',
    'write.parquet.bloom-filter-enabled.column.name' = 'true',
    'dlc.ao.data.govern.sorted.keys' = 'id,name'
);
```

Create a table with the primary key of ID in the console.



Configuration Instructions

Attribute Value	Description	Configuration Guidance
format-version	Iceberg table version. The values include 1 and 2. The default value is 2 for Spark Standard-S 1.1 and SuperSQL Spark 3.5 and 1 for other scenarios.	It is recommended that it be set to 2.

write.upsert.enabled	Specifies whether to enable upsert. The value is true. If it is not specified, upsert is not enabled.	If upsert is involved in a writing scenario, it should be set to true.
write.update.mode	Update mode	The default value is copy-on-write. Copy-on-write is the default mode. If it cannot be determined which one of the two modes is used, the copy-on-write mode can be used. The merge-on-read mode has better performance in row-level update scenarios. Applicable scenarios of copy-on-write: Featuring relatively higher query performance but relatively slower writing speed, copy-on-write is suitable for scenarios that have periodic ETL tasks or batch update operations with a large data volume. Applicable scenarios of merge-on-read: Featuring relatively lower query performance but relatively faster writing speed, merge-on-read is suitable for scenarios with high write performance requirements. It has strong row-level update capabilities and significantly improves the write performance for frequent and small-scale (< 10%) merge-into/update/delete operations or Oceanus (Flink stream writing) scenarios.
write.merge.mode	Merge mode	The default value is copy-on-write. Copy-on-write is the default mode. If it cannot be determined which one of the two modes is used, the copy-on-write mode can be used. The merge-on-read mode has better performance in row-level update scenarios. Applicable scenarios of copy-on-write: Featuring relatively higher query performance but relatively slower writing speed, copy-on-write is suitable for scenarios that have periodic ETL tasks or batch update operations with a large data volume. Applicable scenarios of merge-on-read: Featuring relatively lower query performance but relatively faster writing speed, merge-on-read is suitable for scenarios with high write performance requirements. It has strong row-

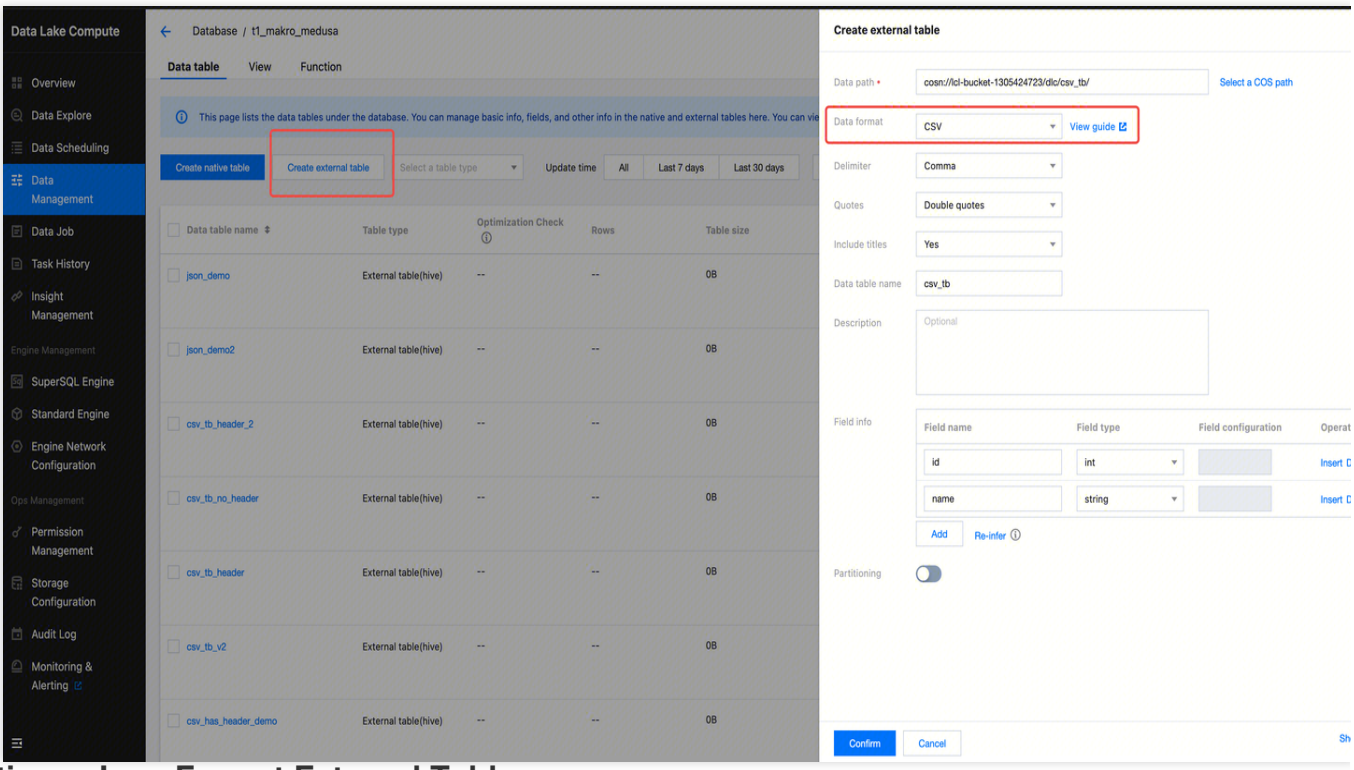
		level update capabilities and significantly improves the write performance for frequent and small-scale (< 10%) merge-into/update/delete operations or Oceanus (Flink stream writing) scenarios.
write.parquet.bloom-filter-enabled.column.{col}	Specifies whether to enable bloom. It is only suitable for Oceanus (Flink stream writing) scenarios. The value true indicates that it is enabled, and it is not enabled by default.	For Flink stream writing scenarios, it should be enabled and configured based on the upstream primary key. If there are multiple primary keys in the upstream, the first two (at most) primary keys are used. After it is enabled, the MOR query and small file merge performance can be improved.
write.distribution-mode	Write mode	When it is set to hash, written data is repartitioned automatically. However, the write performance is affected for some write situations. It is recommended that it is not configured by default. For Oceanus (Flink stream writing) scenarios, it is recommended that it be set to hash to optimize the write performance. For other scenarios, it is recommended that the default value be retained, that is, it is not configured.
write.metadata.delete-after-commit.enabled	Starts auto cleanup of metadata files.	It is strongly recommended that it be set to true. After it is enabled, Iceberg automatically cleans historical metadata files when generating snapshots, which prevents the accumulation of a large number of metadata files.
write.metadata.previous-versions-max	Sets the default number of metadata files to be retained.	The default value is 100. In some special cases, users can appropriately adjust this value. It needs to be used together with write.metadata.delete-after-commit.enabled.

Create External Table

Creating a CSV Format External Table

```
/**
1. separatorChar: Separator, which is a comma (,) by default. It is used to specify
2. quoteChar: Quote character, which are quotation marks (") by default. If the ori
```

```
3. LOCATION: It needs to be changed to the COS storage path.
4. Table attribute skip.header.line.count in TBLPROPERTIES: The default value is 0.
*/
CREATE EXTERNAL TABLE IF NOT EXISTS  dlc_db.`csv_tb`(
    `id` int,
    `name` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  'quoteChar'='"',
  'separatorChar'=',')
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  'cosn://your_cos_location'
TBLPROPERTIES (
  'skip.header.line.count'='1');
```



Creating a Json Format External Table

```
/**
 LOCATION： Refers to the COS storage path.
Others remain unchanged.
*/
CREATE EXTERNAL TABLE IF NOT EXISTS dlc_db.json_demo
```

```
(`id` bigint, `name` string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION 'cosn://your_cos_location'
```

Example of JSON file content. Each line is an independent JSON string:

```
{"id":1,"name":"tom"}
{"id":2,"name":"tony"}
```

Creating an External Table in Parquet Format

The SQL statements for table creation are as follows:

```
/**
 LOCATION： Refers to the COS storage path.
Others remain unchanged.
*/
CREATE EXTERNAL TABLE IF NOT EXISTS dlc_db.parquet_demo
 (`id` int, `name` string)
 PARTITIONED BY (`dt` string)
 STORED AS PARQUET LOCATION 'cosn://your_cos_location';
```

Creating an ORC Format External Table

```
/**
 LOCATION： Refers to the COS storage path.
Others remain unchanged.
*/
CREATE EXTERNAL TABLE IF NOT EXISTS dlc_db.orc_demo
(`id` int,`name` string)
 PARTITIONED BY (`dt` string)
 STORED AS ORC LOCATION 'cosn://your_cos_location'
```

Creating an External Table in AVRO Format

```
/**
 LOCATION： Refers to the COS storage path.
Others remain unchanged.
*/
CREATE EXTERNAL TABLE IF NOT EXISTS dlc_db.avro_demo
(`id` int,`name` string)
 PARTITIONED BY (`dt` string)
 STORED AS ORC LOCATION 'cosn://your_cos_location'
```

Supplementary Description

Column Types and Partition Field Types

For details, see the Parameter section in CREATE TABLE.

Note:

When a SELECT query statement is executed to query binary fields, the following error may occur because the engine writes the result set to a CSV file by default, which does not support binary data.

```
Error operating ExecuteStatement: org.apache.spark.sql.AnalysisException:
[UNSUPPORTED_DATA_TYPE_FOR_DATASOURCE] The CSV datasource doesn't support the column `bno` of the type
"BINARY". at
```

Solution (engine or task-level configuration are all supported):

1. Change the results file format for saving: kyuubi.operation.result.saveToFile.format=parquet (Set the format of the stored file, Parquet or ORC.)
2. Change the configuration to save the results to another place other than COS: kyuubi.operation.result.saveToFile.enabled=false

Complex Column Types

```
/**
LOCATION： Refers to the COS storage path.
Others remain unchanged.
*/
CREATE EXTERNAL TABLE dlc_db.orc_demo_with_complex_type(
    col_bigint bigint COMMENT 'id number',
    col_int int,
    col_struct struct<x: double, y: double>,
    col_array array<struct<x: double, y: double>>,
    col_map map<struct<x: int>, struct<a: int>>,
    col_decimal DECIMAL(10,2),
    col_float FLOAT,
    col_double DOUBLE,
    col_string STRING,
    col_boolean BOOLEAN,
    col_date DATE,
    col_timestamp TIMESTAMP
)
PARTITIONED BY (`dt` string)
STORED AS ORC LOCATION 'cosn://your_cos_location';
```

Note：

1. AVRO data sources do not support nested struct fields to map or array fields.
2. The key of map fields in an AVRO data source can only be string type.

3. When struct, array, or map fields are used in a CSV data source, the following error may occur because the engine performs strong verification on the data format.
Disable strong verification settings of the engine and set the static parameter of the engine with spark.sql.storeAssignmentPolicy=legacy.

Complex Partition Field Types

1. LOCATION: It needs to be changed to the COS storage path.
2. Supported partition field types include TINYINT, SMALLINT, INT, BIGINT, DECIMAL, FLOAT (not recommended, and DECIMAL is recommended), DOUBLE (not recommended, and DECIMAL is recommended), STRING, BOOLEAN, DATE, and TIMESTAMP.

```
CREATE EXTERNAL TABLE dlc_db.orc_demo_with_complex_partition(
    col_int int
)
PARTITIONED BY (
    pt_tinyint TINYINT,
    pt_smallint SMALLINT,
    pt_decimal DECIMAL(10,2),
    pt_string STRING,
    pt_date DATE,
    pt_timestamp TIMESTAMP )
STORED AS ORC LOCATION 'cosn://lcl-bucket-1305424723/dlc/orc_demo_with_complex_part
```

Note：

The sum of Hive table partition names cannot exceed 767 characters.

Case-insensitive Metadata

Table names and column names in metadata are case-insensitive when they are used. However, the original case format during creation is retained on the Data Management page.

Using Apache Airflow to Schedule DLC Engine to Submit Tasks

Last updated：2025-05-22 15:45:02

This document introduces DLC's support for the Apache Airflow scheduling tool and provides examples demonstrating how to use Apache Airflow to run different types of DLC engine tasks.

Overview

Apache Airflow is an open-source scheduling tool developed by Airbnb, written in Python. It defines and schedules a set of interdependent tasks using Directed Acyclic Graphs (DAGs). Apache Airflow supports sub-tasks written in Python and offers various operators to execute tasks, such as Bash commands, Python functions, SQL queries, and Spark jobs, providing high flexibility and scalability. Widely used in fields like data engineering, data processing, and workflow automation, Apache Airflow allows users to easily monitor and manage the state and execution of workflows through its rich features and visual interface. For more information about Apache Airflow, see Apache Airflow.

Prerequisites

1. Prepare the Apache Airflow environment.
2. Install and start Apache Airflow. For detailed steps on installation and startup, see Quick Start.
3. Install the jaydebeapi dependency package, pip install jaydebeapi.
4. Prepare the DLC environment.
5. Enable the DLC engine service.
6. If using the standard Spark engine, prepare the Hive JDBC driver. Click to download hive-jdbc-3.1.2-standalone.jar.
7. If using the standard Presto engine, prepare the Presto JDBC driver. Click to download presto-jdbc-0.284.jar.
8. If using the SuperSQL engine, prepare the DLC JDBC driver. Click to download the JDBC driver.

Key Steps

Creating Connection and Scheduling Tasks

In the Apache Airflow working directory, create a dags directory. Inside the dags directory, create a scheduling script and save it as a .py file. For example, in this document, the scheduling script is created as /root/airflow/dags/airflow-dlc-test.py as shown below:

```
import time
from datetime import datetime, timedelta

import jaydebeapi
from airflow import DAG
from airflow.operators.python_operator import PythonOperator

jdbc_url='jdbc:dlc:dlc.tencentcloudapi.com?task_type=SparkSQLTask&database_name={da
user = 'xxx'
pwd = 'xxx'
dirver = 'com.tencent.cloud.dlc.jdbc.DlcDriver'
jar_file = '/root/airflow/jars/dlc-jdbc-2.5.3-jar-with-dependencies.jar'

def createTable():
    sqlStr = 'create table if not exists db.tb1 (c1 int, c2 string)'
    conn = jaydebeapi.connect(dirver, jdbc_url, [user, pwd], jar_file)
    curs = conn.cursor()
    curs.execute(sqlStr)
    rows = curs.rowcount.real
    if rows != 0:
        result = curs.fetchall()
        print(result)
    curs.close()
    conn.close()


def insertValues():
    sqlStr = "insert into db.tb1 values (111, 'this is test')"
    conn = jaydebeapi.connect(dirver,jdbc_url, [user, pwd], jar_file)
    curs = conn.cursor()
    curs.execute(sqlStr)
    rows = curs.rowcount.real
    if rows != 0:
        result = curs.fetchall()
        print(result)
    curs.close()
    conn.close()


def selectColums():
    sqlStr = 'select * from db.tb1'
    conn = jaydebeapi.connect(dirver, jdbc_url, [user, pwd], jar_file)
    curs = conn.cursor()
    curs.execute(sqlStr)
    rows = curs.rowcount.real
    if rows != 0:
```

```
        result = curs.fetchall()
        print(result)
    curs.close()
    conn.close()


def get_time():
    print('Current time is:', datetime.now().strftime('%Y-%m-%d %H:%M:%S'))
    return time.time()


default_args = {
    'owner': 'tencent',  # owner's name
    'start_date': datetime(2024, 11, 1),  # the first execution start time, in UTC
    'retries': 2,  # number of retry attempts on failure
    'retry_delay': timedelta(minutes=1),  # retry interval on failure
}
dag = DAG(
    dag_id='airflow_dlc_test',  # DAG ID, should consist only of letters, numbers,
    default_args=default_args,  # externally defined parameters in dic format
    schedule_interval=timedelta(minutes=1),  # defines the frequency at which the D
    catchup=False  # when executing the DAG, all tasks scheduled from the start tim
)

t1 = PythonOperator(
    task_id='create_table',
    python_callable=createTable,
    dag=dag)

t2 = PythonOperator(
    task_id='insert_values',
    python_callable=insertValues,
    dag=dag)

t3 = PythonOperator(
    task_id='select_values',
    python_callable=selectColums,
    dag=dag)

t4 = PythonOperator(
    task_id='print_time',
    python_callable=get_time,
    dag=dag)

t1 >> t2 >> [t3, t4]
```

Parameter description:

Parameters	Description
jdbc_url	JDBC connection address and configuration parameters. For more details, see Hive JDBC Access, Presto JDBC Access, and DLC JDBC Access.
user	SecretId
pwd	SecretKey
dirver	Load the JDBC driver. For more details, see Hive JDBC Access, Presto JDBC Access, and DLC JDBC Access.
jar_file	The storage path of the driver JAR package. Replace it with the absolute path where the corresponding engine's JDBC driver JAR package is stored. For more details, see Hive JDBC Access, Presto JDBC Access, and DLC JDBC Access.

Running Scheduled Tasks

You can access the Web interface, navigate to the DAGs tab, locate the submitted scheduling workflow, and start the scheduling.

Viewing Task Execution Results

Best Practices of Implementing Machine Learning with DLC + WeData

Last updated：2025-06-25 16:03:18

Data Lake Compute (DLC) supports creating machine learning resource groups via Spark engines, assisting users in machine learning scenarios, including model training.
Through this document, you can experience the practice of model training within the scikit-learn framework based on our provided demo dataset and example code.
Note:
Resource group: As a secondary queue division of the computing resources within a Spark Standard Engine, resource groups belong to the parent Standard Engine. Resource groups under the same engine share resources with each other.
The computing units (CUs) of the DLC Spark Standard Engine can be allocated into multiple resource groups as needed. You can configure the minimum and maximum usable CU limits of each resource group, start and stop policies, concurrency quantity, dynamic or static parameters, and other items to efficiently manage computing resources isolation and workload in complex scenarios, including multi-tenant and multi-tasking. This feature ensures resource isolation between different types of tasks and prevents resources from being preempted for extended periods by a few large queries.
Currently, the DLC machine learning resource group, WeData Notebook exploration, and machine learning are all **allowlisted features**. If needed, submit a ticket to contact the DLC and WeData teams to enable the machine learning resource group, Notebook, and MLflow services.

Activating Accounts and Products

Activating Accounts and Products

The DLC account and product activation features need to be enabled with the Tencent Cloud root account. Once the root account completes the operations, all sub-accounts under the default root account can use these features. Adjustments can be made using the CAM feature if needed. For specific operation guides, see the Complete Process for New User Activation.
For WeData account and product activation, see Preparations and Data Lake Compute (DLC).
The feature and MLflow service activation are performed at the root account granularity. Once the operations are completed with the root account, all sub-accounts under the root account can use these features.
You need to provide the customer regional information, APPID, root account UIN, VPC ID, and subnet ID. The VPC and subnet information are used for the network interconnection operation of the MLflow service.
Note:

Since multiple features on the product require network access operations, to ensure network connectivity, it is recommended that subsequent operations (including purchasing execution resource groups and creating Notebook workspaces) be performed within this VPC and subnet.

Configuring Data Access Policies

A data access policy (CAM role arn) allows users to configure data access permissions in Cloud Access Management (CAM) to ensure the security of data sources and Cloud Object Storage (COS) during data job execution.
When configuring a data job in DLC, you need to specify the corresponding data access policy to ensure data security. For the configuration methods, see Configuring Data Access Policies.

Purchasing Computing Resources on DLC

After the product service activation is completed, you can first purchase computing resources through DLC. If you need to use the machine learning feature, **confirm that the purchased engine type is Standard Engine-spark, and the kernel version is Standard-S 1.1.**

1. Go to the Data Lake Compute (DLC) > Standard Engine page.
2. Select "Creating Resources".
3. Purchase Standard Engine-spark, and select the kernel version: Standard-S 1.1.

Note:

1. Purchase should be made with the root account or an account having financial permission.
2. You can select the billing mode according to your business scenarios.
3. It is recommended to select cluster specifications with 64 CUs or more.
4. The initial launch may require several minutes of waiting time after purchase. If the startup cannot be completed for a long time, submit a ticket.

Creating Machine Learning Resource Groups

After purchasing the Standard Engine, return to the Engine Management page. You need to create a machine learning resource group under this engine to start performing machine learning-related features.

Note:

Once the resource group is created, it cannot be edited or modified. You can manage it by deleting and recreating the resource group.

1. Click **Manage Resource Group**.

2. After going to the resource group page, click the **Create Resource Group** button in the upper-left corner.
3. Create a resource group for machine learning.

Note:

DLC AI resource groups currently support machine learning through the following frameworks: scikit-learn (1.6.0), TensorFlow (2.18.0), PyTorch (2.5.1), Python (3), and Spark MLlib (3.5).

Business scenario selection: Machine learning.

Framework type: You can select a suitable framework to create based on actual business scenarios.

If you need to experience the features through the demo, select an ML open-source framework. For the image package, select: scikit-learn-v1.6.0.

Resource configuration: Select resources as needed.

After the configuration is completed, click Confirm to return to the resource group list page. After several minutes, you can click the Refresh button at the top of the list page for confirmation.

Uploading Machine Learning Datasets to COS

If you need to perform machine learning via DLC and WeData, use Cloud Object Storage (COS) in combination, because only interaction with cloud data is supported currently.

Note:

Currently, only direct reading of COS data via Spark is supported.

If you have requirements for other frameworks, try the following workaround: first, upload the data to COS, then download it to a local device to generate a local file, and proceed with the learning operation. This workaround may result in relatively long upload and download times.

We are currently developing this feature to optimize the support.

1. Activate the COS product service and create a bucket. For activation methods, see the console quick start.
2. Log in to COS, select a bucket, and upload the dataset.
3. After the upload is completed, go to Metadata Management, click Create Data Catalog, or use an existing data catalog to upload an external table.
4. Go to the DLC console > Metadata Management and click on the Database tab.
5. Click to create a database named: database_testnotebook.
6. Go to the created database and click to create an external table.

Note:

Check the database table name of the uploaded external table. The Notebook will use select to call the database and table names.

7. Select the COS bucket path and find the demo dataset.

8. Select the data format as CSV and configure related options.

9. Create a table named: demo_test_sklearn.
10. After the creation is completed, click Confirm to return.

You can also perform creation via SQL. For details, see Table Creation Practice > Creating External Tables in CSV Format.

```
CREATE TABLE database_testnotebook.demo_test_sklearn (
  at STRING COMMENT 'from deserializer',
  v STRING COMMENT 'from deserializer',
  ap STRING COMMENT 'from deserializer',
  rh STRING COMMENT 'from deserializer',
  pe STRING COMMENT 'from deserializer')
USING csv
LOCATION 'cosn://your cos location'
```

Going to the WeData-Notebook Feature for Demo Practice

After the resource group and demo dataset are created, go to WeData to practice model training using Notebook and MLflow.

Creating WeData Projects and Associating Them with DLC Engines

1. Create a project or select an existing project. For details, see Project List.
2. Select the required DLC engine in the configuration of the storage and computing engine.

Purchasing Execution Resource Groups and Associating Them with Projects

If you need to schedule Notebook tasks periodically in the orchestration space, purchase a scheduling resource group and associate it with a designated project. For details, see Scheduling Resource Group Configuration.

Directions:

1. Go to "Execution Resource Group > Scheduling Resource Group > Standard Scheduling Resource Group" and click **Create**.
2. Configure the resource group.

Region: The region where the scheduling resource group is located should be consistent with the region where the storage and computing engine is located. For example, if you purchase a DLC engine in the Singapore region of the international site, you need to purchase a scheduling resource group in the same region.

VPC and subnet: It is recommended to select the VPC and subnet in Standard-S 1.1. If other VPCs and subnets are selected, you need to ensure that the selected VPCs and subnets are interconnected with the VPCs and subnets in Standard-S 1.1.

Specifications: Select specifications according to the task volume.

3. After the creation is completed, click "Associate Project" in the operation column of the resource group list to associate this scheduling resource group with the desired project.

Creating Notebook Workspaces

1. In the project, select the data governance feature, click the Notebook feature, and create or use an existing workspace.

2. When creating a workspace, select to purchase **a Standard Spark Engine, and the version is Standard-S 1.1. Select the machine learning and MLflow service options.**

Basic Information	Attribute Item Name	Attribute Item Configuration
	Engines	Select a DLC engine that you need to use for accessing a Notebook task. The DLC engine bound in the current project management.
	DLC Data Engine	Select a DLC data engine that you need to use for accessing a Notebook task.
	Machine Learning	If the DLC data engine you select contains a "machine learning" type resource group, this option will appear and be selected by default.
	Network	It is recommended to select the VPC and subnet in Standard-S 1.1. If other VPCs and subnets are selected, you need to ensure that the selected VPCs and subnets are interconnected with the VPCs and subnets in Standard-S 1.1.
	RoleArn	RoleArn is the data access policy (CAM role arn) for the DLC engine to access Cloud Object Storage (COS). For details, see Configuring Data Access Policies.
Advanced Configuration	MLflow Service	Use MLflow to manage experiments and models, which is not selected by default. After selecting, the creation of experiments and machine learning using MLflow functions in Notebook tasks will be reported to the MLflow service deployed in Standard-S 1.1. You can later view them in Machine Learning > Experiment Management and Model Management.

Creating Notebook Files

In the resource explorer on the left, you can create folders and Notebook files. Note: Notebook files need to end with (.ipynb.) The resource explorer includes 3 preset Big Data series tutorials for out-of-the-box use.

Selecting Kernels

1. Click **Select Kernel**.

2. Select "DLC resource group" in the pop-up dropdown option.
3. Select the scikit-learn resource group you created in the DLC data engine from the next-level options.

Running Notebook Files

1. Confirm the initialization configuration.

2. Execute best practices: Use the public Iris dataset for demonstration, implement a logistic regression model to classify different types of flowers, and visualize the classification results.

Note:

Before running the model, you need to install the required tencentcloud-dlc-connector and complete the corresponding configurations.

```
#Install the driver.
!pip install tencentcloud-dlc-connector
!pip  install --upgrade 'sqlalchemy<2.0'

#Installation version.
!pip install --upgrade pandas==2.2.3
!pip install numpy
!pip install matplotlib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import tdlc_connector
from tdlc_connector import constants
import mlflow
mlflow.sklearn.autolog()

#Use tdlc-connector to access in table mode.
conn = tdlc_connector.connect(region="ap-***", #Fill in the correct address, for ex
    secret_id="*******",
    secret_key="*******",
    engine="your engine", #Fill in the purchased engine name.
    resource_group=None,
    engine_type=constants.EngineType.AUTO,
    result_style=constants.ResultStyles.LIST,
    download=True
    )
query = """
```

```
SELECT sepal.length, sepal.width,petal.length,petal.width,species FROM at_database_
"""


#Read data.
iris = pd.read_sql(query, conn)
iris.head()
#Divide the dataset.
X = iris[['petal.length', 'petal.width']].values
category_map = {
    'setosa': 0,
    'versicolor': 1,
    'virginica': 2
}

y= iris['species'].replace(category_map)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=1, stratify=y)
print('Labels count in y:', np.bincount(y))
print('Labels count in y_train:', np.bincount(y_train))
print('Labels count in y_test:', np.bincount(y_test))

#Data normalization.
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

#Perform logistic regression classification, and visualize the classification resul
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=100.0, random_state=1, solver='lbfgs', multi_class='ovr')
lr.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=lr, test_idx=range(105, 150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()

#View model accuracy.
y_pred = lr.predict(X_test_std)
```

```
print(accuracy_score(y_test, y_pred))
```

Going to MLflow to View Training Results and Registered Models

1. Select the machine learning feature.

2. View the experiment records and select the best training result to register as a model.

Direct Query of DLC Internal Storage with StarRocks

Last updated：2024-10-31 11:13:10

Introduction of StarRocks

StarRocks is a new-generation, high-performance, and all-scene MPP database. It leverages the best research outcomes of relational OLAP databases and distributed storage systems in the era of big data. Building on industry practices, StarRocks optimizes and upgrades its architecture, and introduces numerous new features, resulting in a cutting-edge enterprise-level product. StarRocks is committed to delivering a fast and unified analysis experience, meeting enterprises' various data analysis needs. It supports multiple data models (detailed, aggregate, and update models) and diverse data import methods (batch and real-time). StarRocks can handle data of up to 10,000 columns and integrate with various existing systems such as Spark, Flink, Hive, and ElasticSearch.
In Tencent Cloud EMR, we offer a fully out-of-the-box StarRocks service. For more details, see StarRocks Introduction.

StarRocks + DLC Lakehouse Unified Query Acceleration

Tencent Cloud Data Lake Compute (DLC) supports a lakehouse unified query acceleration mode based on EMR StarRocks. This allows you to seamlessly analyze DLC data sources and perform complex SQL queries without the need to import DLC data into StarRocks or create external tables of DLC in StarRocks. Leveraging StarRocks' MPP vectorized query capabilities improves data analysis efficiency while reducing ops complexity and costs.
Next, this document will guide you through how to enable the DLC + (EMR) StarRocks lakehouse unified query acceleration.

Prerequisites

1. You have purchased an EMR StarRocks cluster.
2. You have enabled the DLC service.

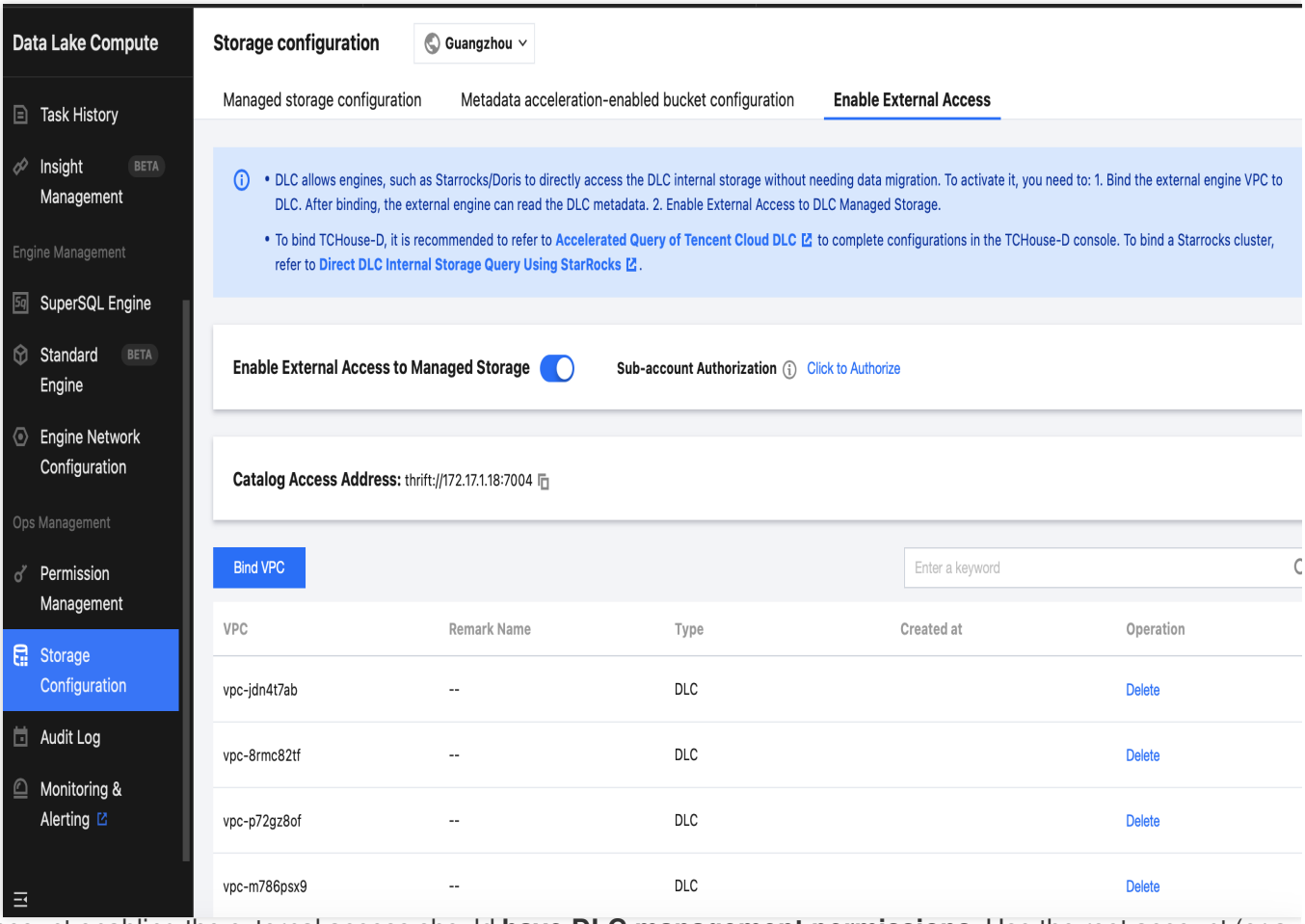
Note:

1. Currently, cross-region federated DLC is not supported. Plan your environment accordingly to ensure that EMR StarRocks and DLC are in the same region.
2. Queries on DLC native tables created before June 12, 2024, are not supported (these tables have storage paths in the format of lakefs://***, which is currently incompatible). However, DLC native tables created after June 12,***

2024 (with storage paths in the format of cosn://) and all DLC external tables are supported.
3. StarRocks only supports querying DLC data; it does not support writing to or deleting DLC data.
To enable query acceleration of DLC + StarRocks lakehouse unification, you first need to **enable DLC external access**, allowing the StarRocks cluster to access data stored in DLC's internal managed storage. Then, create a **DLC External Catalog** in the StarRocks cluster, allowing you to use the StarRocks compute engine to directly analyze data stored in DLC.

Enabling DLC External Access

Go to the DLC console, click the **Storage Configuration** menu, and select **Enable External Access**.
Step 1: Click "**enable external access to managed storage**". Once it is enabled, the EMR StarRocks cluster under the same Tencent Cloud account will be able to access DLC's internal managed storage. Subsequently, you can use this switch to enable or disable external access to DLC's internal managed storage.



The account enabling the external access should **have DLC management permissions**. Use the root account (or a sub-account with DLC administrator permissions) to perform this operation.
Note:
1. If you already have DLC administrator permissions, you can skip this step.

2. If you're using a sub-account without DLC administrator permissions, see Sub-account Permission Management to request authorization from an account with DLC administrator permissions.

Step 2: To ensure that your EMR StarRocks cluster can correctly access the DLC metadata Catalog service, you need to bind the VPC of the EMR StarRocks cluster to the DLC network.

1. Click Bind VPC, select EMR StarRocks as the type in the dialog box, and select the EMR StarRocks cluster instance ID you want to bind from the EMR instance dropdown list.
2. The VPC of the StarRocks cluster will be automatically filled. You can enter a recognizable alias in the remarks field for easier identification.

Step 3: After completing the VPC binding, you can connect to the DLC metadata service from the StarRocks cluster using the URI connection string displayed in the Catalog access address. For example:

Example of URI Connection String:

Catalog access address: thrift://172.17.1.18:7004

At this point, your EMR StarRocks cluster is ready to directly analyze internal managed data of DLC. Before you start the analysis, you need to create a DLC External Catalog in StarRocks.

Creating DLC External Catalog

Log in to StarRocks and create a DLC Catalog in StarRocks. For more details about External Catalogs, see CREATE EXTERNAL CATALOG | Overview.

Syntax

```
CREATE EXTERNAL CATALOG dlc_iceberg_cos_catalog
PROPERTIES
(
    "type" = "iceberg",
        "iceberg.catalog.type" = "hive",
    "iceberg.catalog.hive.metastore.uris" = "thrift://169.254.0.171:8007",
    "aws.s3.endpoint" = "cos.ap-chongqing.myqcloud.com",
    "aws.s3.access_key" = "Tencent Cloud secret id",
    "aws.s3.secret_key" = "Tencent Cloud secret_key"
);
```

Parameter Description

Parameter	Description
type	The type of data source, which is set to Iceberg by default.
iceberg.catalog.type	The type of metadata service used by the Iceberg cluster, which is set to

	hive.
iceberg.catalog.hive.metastore.uris	The URI for the DLC metadata.
aws.s3.endpoint	The endpoint for accessing S3-compatible object storage. For Tencent Cloud COS, the format is cos.<region>.myqcloud.com, where <region> can be ap-beijing, ap-shanghai, ap-guangzhou, etc.
aws.s3.access_key	The SecretID from the Tencent Cloud account keys.
aws.s3.secret_key	The SecretKey from the Tencent Cloud account keys.

Note:

1. For security reasons, you need to use the SecretID and SecretKey of the root account in the above configuration to properly access the DLC internal storage. You can obtain the SecretID and SecretKey by logging into Tencent Cloud with the root account and navigating to the CAM Control Console > log in to - Tencent Cloud page.
2. If you need to use the SecretID and SecretKey of a sub-account to access DLC internal managed storage, see the final section of this document **Using Sub-account SecretID/SecretKey**.

Example

The following example demonstrates how to create a DLC Catalog named DLC_catalog:

```
CREATE EXTERNAL CATALOG dlc_hive_cos_catalog
PROPERTIES
(
    "type" = "Iceberg",
    "iceberg.catalog.type" = "hive",
    "hive.metastore.uris" = "thrift://169.254.0.171:8007",
    "aws.s3.endpoint" = "cos.ap-chongqing.myqcloud.com",
        "aws.s3.access_key" = "**********************************",
    "aws.s3.secret_key" = "*******************************"
)
```

If you need to query Hive tables stored in DLC managed storage directly through StarRocks, you will need to create a separate DLC Hive catalog, setting the type to hive. An example is provided below:

```
CREATE EXTERNAL CATALOG dlc_hive_cos_catalog
PROPERTIES
(
    "type" = "hive",
    "iceberg.catalog.type" = "hive",
    "hive.metastore.uris" = "thrift://169.254.0.171:8007",
    "aws.s3.endpoint" = "cos.ap-chongqing.myqcloud.com",
        "aws.s3.access_key" = "**********************************",
```

```
    "aws.s3.secret_key" = "******************************"
)
```

Querying DLC Data

Preparing Tables and Data in DLC

Here is an example of creating an Iceberg table:

```
CREATE TABLE test_sr_ofs.`customer`(
    `c_custkey` bigint,
    `c_name` string,
    `c_address` string,
    `c_nationkey` int,
    `c_phone` string,
    `c_acctbal` double,
    `c_mktsegment` string,
    `c_comment` string) using iceberg;
)
```

Querying Data in StarRocks

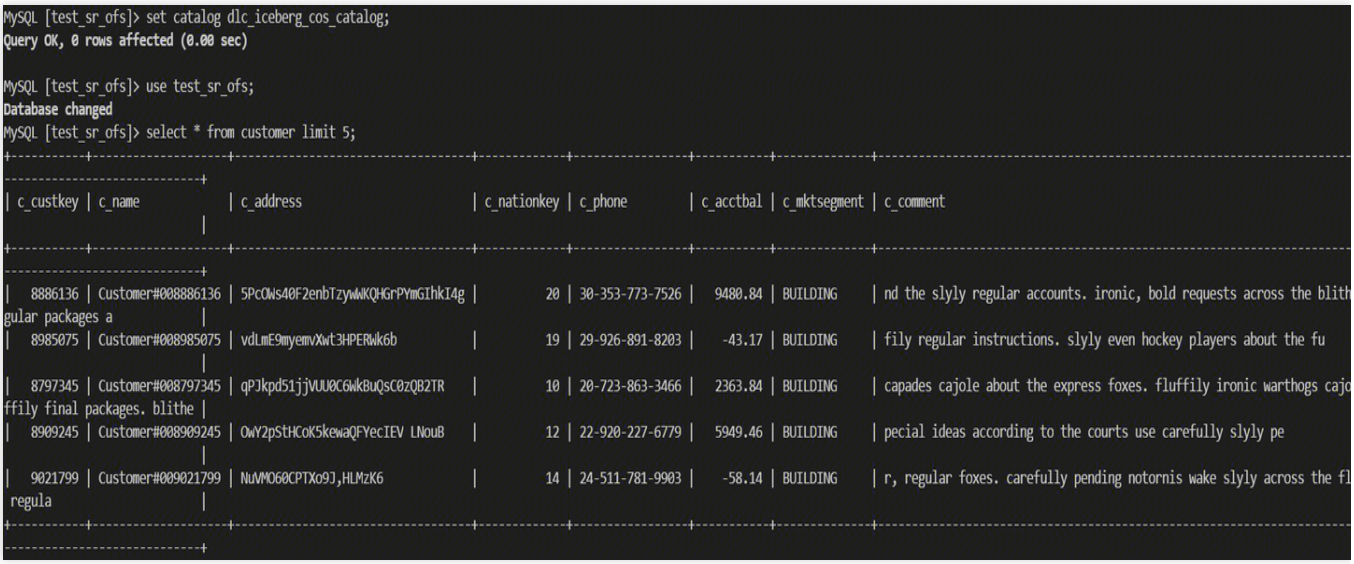
```
# Log in to the StarRocks Node.
mysql -h 172.30.0.xxx -P9030 -u root -p

#Specify the Iceberg Catalog.
set catalog dlc_iceberg_cos_catalog;

#Specify the Database.
use test_sr_ofs;

#Query the customer table data.
select * from customer limit 5;
```

The query results are as follows:



```
MySQL [test_sr_ofs]> set catalog dlc_iceberg_cos_catalog;
Query OK, 0 rows affected (0.00 sec)

MySQL [test_sr_ofs]> use test_sr_ofs;
Database changed
MySQL [test_sr_ofs]> select * from customer limit 5;
+-----------+--------------------+---------------------------------------+-------------+-----------------+-----------+-------------+-----------------------------------------------------------------------------------------------------------------------+
| c_custkey | c_name             | c_address                             | c_nationkey | c_phone         | c_acctbal | c_mktsegment | c_comment                                                                                                             |
+-----------+--------------------+---------------------------------------+-------------+-----------------+-----------+-------------+-----------------------------------------------------------------------------------------------------------------------+
|   8886136 | Customer#008886136 | 5PcOWs40F2enbTzywWKQHGrPYmGIhkI4g     |          20 | 30-353-773-7526 |   9480.84 | BUILDING    | nd the slyly regular accounts. ironic, bold requests across the blithegular packages a                                |
|   8985075 | Customer#008985075 | vdLmE9myemvXwt3HPERWk6b               |          19 | 29-926-891-8203 |    -43.17 | BUILDING    | fily regular instructions. slyly even hockey players about the fu                                                     |
|   8797345 | Customer#008797345 | qPJkpd51jjVUU0C6WkBuQsC0zQB2TR        |          10 | 20-723-863-3466 |   2363.84 | BUILDING    | capades cajole about the express foxes. fluffily ironic warthogs cajoffily final packages. blithe                     |
|   8909245 | Customer#008909245 | OwY2pStHCoK5kewaQFYecIEV LNouB        |          12 | 22-920-227-6779 |   5949.46 | BUILDING    | pecial ideas according to the courts use carefully slyly pe                                                           |
|   9021799 | Customer#009021799 | NuVMO60CPTXo9J,HLMzK6                 |          14 | 24-511-781-9903 |    -58.14 | BUILDING    | r, regular foxes. carefully pending notornis wake slyly across the flregula                                           |
+-----------+--------------------+---------------------------------------+-------------+-----------------+-----------+-------------+-----------------------------------------------------------------------------------------------------------------------+
```

Using Sub-account SecretID/SecretKey for Access (Optional)

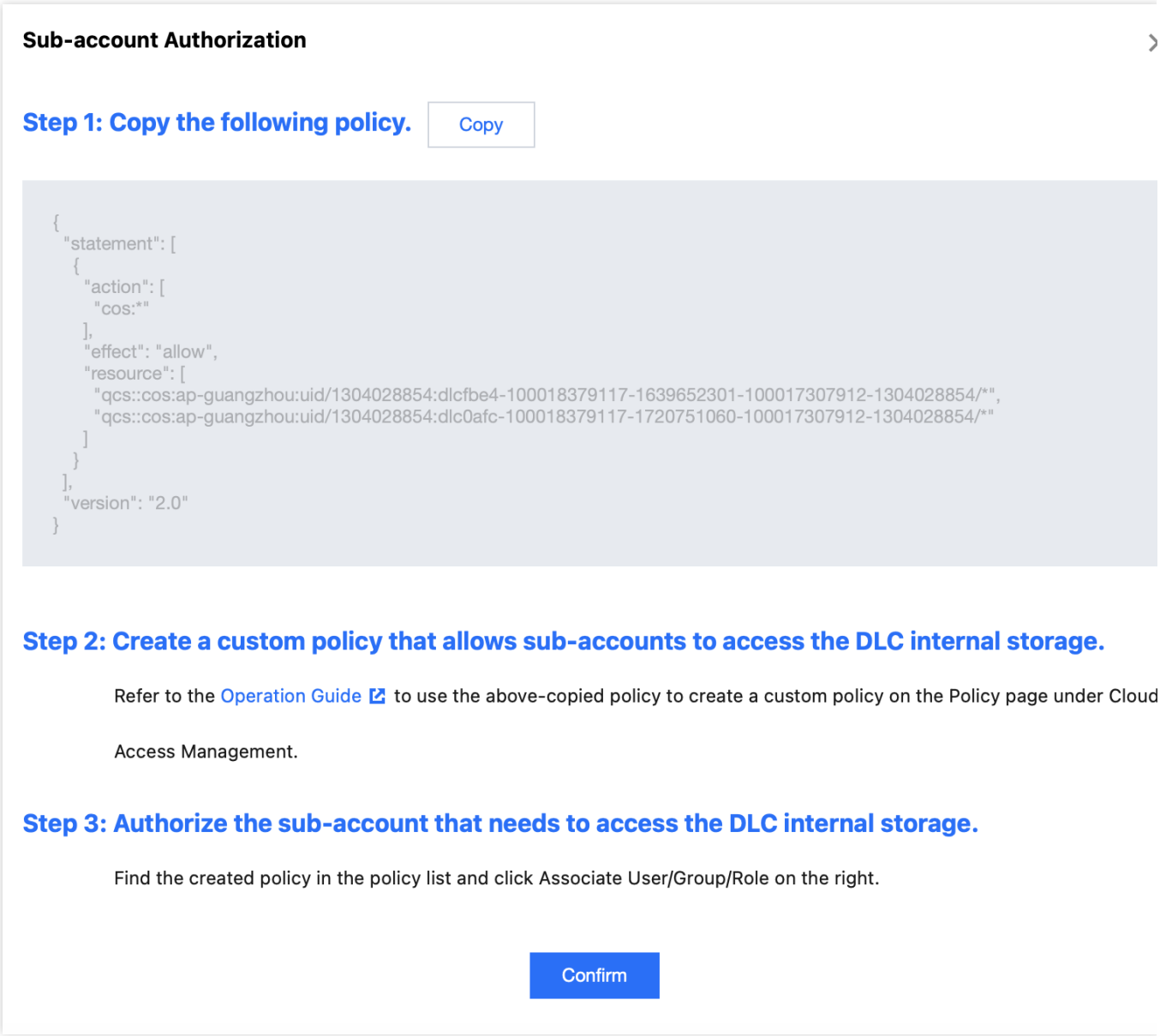
For **data security reasons**, after DLC external access is enabled, it is required by default to use the root account's SecretID and SecretKey to access DLC internal storage from EMR StarRocks. However, if your business scene requires the use of a sub-account's SecretID and SecretKey, **you will need to use the root account to create a custom policy in CAM and bind it to the corresponding sub-account**.

Explanation of the Principle

After you follow the instructions in this document to enable DLC external access, the system essentially grants your root account permissions to access DLC's internal managed storage. However, for data security reasons, sub-accounts under your root account **do not have this access by default**. To grant a sub-account access to DLC's internal managed storage, you need to create a custom policy in CAM by Tencent Cloud and bind it to the sub-account. The specific steps are as follows:

Step 1: Generating a Custom Policy

Log in to Tencent Cloud using the root account and go to the DLC console, click the **Storage Configuration** menu and select **Enable External Access**. In the **Sub-account Authorization** section, click "**Click to Handle**". In the pop-up dialog box, click **Copy** to obtain the custom CAM policy that you need to create.



Step 2: Creating a Custom Policy to Allow Sub-account Access to DLC Internal Storage

1. Log in to the Policy page of the CAM console using the root account.
2. Select **Create Your Own Definition Policy** , create it based on the policy syntax, select the blank template and click Next to proceed.

Note:

To grant sub-accounts data access permissions, the root account can only do so via custom policies. Predefined policies do not support this authorization.

3. Fill out the form as follows:

Policy name: Create a unique and meaningful policy name, such as cos-child-account.

Remarks: Optional, you can write your own description.

Policy content: Paste the custom policy copied from **Step 1**. For example:

```
{
  "statement": [
```

```
    {
      "action": [
        "cos:*"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-shanghai:uid/1305424723:dlc03ff-100018379117-1647867281-100017
        "qcs::cos:ap-shanghai:uid/1305424723:dlc0a65-100018379117-1680005779-100017
      ]
    }
  ],
  "version": "2.0"
}
```

Note:

The above policy grants the sub-account permissions to operate the DLC managed storage for which the root account has operation permissions. In this example, uid/1305424723 refers to the APPID of the root account (A), and dlc0a65-100018379117-1680005779-100017307912-1304028854/* represents the DLC internal managed storage that you are authorized to operate.

4. Click **Complete** to finish the creation of the policy.

Step 3: Authorizing the Sub-account to Access DLC Internal Storage

1. In the **Policy** list, find the policy you just created and click **Associate User/Group/Role** on the right.
2. In the pop-up window, select the sub-account that needs access to the DLC internal managed storage, and click **OK**.
3. Once the authorization is completed, the sub-account will be able to access the DLC internal managed storage using its SecretID and SecretKey.

Spark cost optimization practice

Last updated：2025-06-26 11:25:06

Cost optimization is a continuous process. Due to the dynamic and ever-changing characteristics of big data, enterprise users should continuously perform cost optimization activities. This topic introduces related practices on how to perform cost optimization in Data Lake Compute (DLC) based on Spark computing resources. You can refer to the provided usage scenarios and apply optimization as needed.

How to Choose a Suitable Payment Method for Computing Resources?

DLC supports purchasing engines in pay-as-you-go and monthly subscription billing modes.

Resource Description	Pay-as-You-Go Computing Resources	Monthly Subscription Computing Resources	Elastic Computing Resources
Fee Standard	0.35 USD/CU x Hour Using 1 CU of resources for 1 hour is charged at 0.35 USD. Charges are calculated based on the actual CU usage.	150 USD/CU x Month Using 1 CU of resources for 1 month is charged at 150 USD.	0.35 USD/CU x Hour Using 1 CU of resources for 1 hour is charged at 0.35 USD. Charges are calculated based on the actual CU usage.
Payment Method	Postpaid	Prepaid	Postpaid
Features	1. Users can flexibly choose when to use resources. When users do not use pay-as-you-go resources, they can choose to suspend (release) the resources. No charges will be incurred after suspension. 2. Pay-as-you-go resources are allocated to a user only when the user starts a cluster to run tasks. Once these resources are allocated, they are exclusively available to the current user. **However, due to limitations in the**	1. Resources are allocated to users for exclusive use upon purchase. Therefore, there will be a situation where resources are unavailable for allocation. 2. Resources are available at any time. Meanwhile, additional **elastic resources can be purchased, and they are charged based on the actual usage**.	1. Elastic computing resources are additional pay-as-you-go resources activated on the **foundation of having purchased a monthly subscription Spark cluster**. 2. Elastic resources can accelerate task execution and reduce the entire system load when necessary. At the same

	resource pool, there may be a situation where resources are insufficient and no pay-as-you-go resources are available for allocation.		time, when there are few tasks, elastic resources are automatically released and no charges are incurred, effectively reducing costs.
Recommended Scenarios	1. POC testing phase 2. The usage duration per month does not exceed 18 days.	1. Official production environment 2. Data computing scenarios with a large and stable workload 3. If the usage duration exceeds 60% of the total duration in 1 month, using the monthly subscription is more cost-effective.	1. Official production environment 2. Purchased monthly subscription computing resources run tasks, but the completion time of the tasks does not meet expectations.
Whether Payment Method Switchover is Supported	Yes	**No**	**No**

Scenario: Task Completion Time Fails to Meet Expectations Due to Insufficient Monthly Subscription Computing Resources

An e-commerce platform has purchased 128 CUs of monthly subscription computing resources to ensure the completion and result returns of 600 analysis tasks on the promotion day.
With the arrival of the major e-commerce promotion, the data volume surged. It was found that the time efficiency of task completion could not be guaranteed during this period. Through analysis, it was concluded that the currently purchased resources could not meet the data processing requirements during the major promotion period, leading to task queuing and thus delaying progress. To address this issue, the enterprise is seeking a solution that can guarantee the timely completion of tasks in the short term while keeping costs within a reasonable range.

Recommended Solution:

During the major promotion period, an additional 128 CUs of elastic computing resources are purchased on top of the existing 128 CUs of monthly subscription computing resources. According to the task load situation, when all the 128 CUs of monthly subscription computing resources are in use, the elastic resources are triggered to enhance the operation efficiency. After the major promotion ends, the elastic feature is disabled to effectively control the cost investment.

Low workload: Pay-as-you-go elastic resources are automatically released, and no charges are incurred, effectively reducing costs.
High workload: Pay-as-you-go elastic resources can speed up task execution when necessary. You pay for what you use.

Activation Steps:

1. Enter the SuperSQL Engine page and find the monthly subscription engine that needs to be configured with elastic computing resources.
2. Click **Spec configuration** in the operation column.
3. Enable the elastic computing resources switch and select the elastic configuration specifications.

Note:

The elastic computing specification cannot exceed the monthly subscription specifications.

4. After the major promotion ends, normal task operations are resumed. You can click Spec configuration to disable the elastic cluster feature.

How to Reasonably Plan the Allocation of Computing Resources?

Method 1: Allocating Computing Resources Through Multiple Engines

Multiple computing resources are purchased and allocated to different user groups or feature scenarios. Each computing resource runs tasks independently.

Strengths	Resources are completely isolated among different engines, and the resources of each engine are exclusively available to the current user group. The configurations, management, tasks, and failures of different user groups do not affect each other.
Applicable Scenarios	1. Multiple departments need to use the platform, but there is no business overlap among them. These mostly occur in independent business analysis scenarios. 2. Enterprises have relatively high requirements for cost auditing and security Ops.
Limitations	Resources may experience idle periods, making it unable to maximize utilization. As shown in the figure, User Group 01 primarily conducts SQL analysis and uses resources between 9 am and 5 pm, leaving resources idle at other times. **Solution:** Switch the computing resources to pay-as-you-go resources and adjust the resource allocation method.

Method 2: Allocating Resources Through Resource Groups with an Engine

In this method, all user groups or feature scenarios share the same engine, but each user group uses resource groups with different configurations to allocate resources.

Strengths	Computing resource utilization can be maximized.
Applicable Scenarios	1. There are relatively high requirements for cost control. 2. A small number of resource groups need to be configured, and most of the tasks are linear startup tasks, while a few are concurrent tasks. 3. The running times of tasks do not overlap significantly, and the computing workload consumed by overlapping tasks does not exceed the total resource amount.
Limitations	Resource competition may occur among different resource groups. As shown in the figure, when all 3 user groups are using resources, if User Group 01 occupies 256 CUs of resources and User Group 02 also occupies 256 CUs of resources, it means that all resources in the engine are occupied at this point. In this case, no resources can be allocated to User Group 03. As a result, User Group 03 cannot run tasks. **Solution:** Add computing resources and adjust the resource group configurations and allocation method.

Scenario: Reasonably Plan Time-Sharing Running of Tasks to Maximize Resource Utilization

An enterprise currently needs to purchase computing resources and allocate them to 3 departments for interactive SQL analysis and batch Spark job tasks. With relatively high requirements for cost control, the enterprise hopes the sales team can recommend a reasonable purchase solution. This solution needs to ensure that resources are reasonably allocated to the 3 departments for use and no task failures occur, on the basis of maximizing resource utilization.

Recommended Solution:

1. For interactive SQL tasks, the running time, minimum number of resources, and elasticity of interactive resource groups need to be assessed based on the number and complexity of tasks to be run per day.
After analysis, the department concludes that interactive SQL tasks need to run from 9 am to 5 pm daily. Based on the maximum concurrency quantity and resource requirements of SQL tasks, it is determined that the resource group needs the maximum specifications of 128 CUs. Resource usage can be configured from 4 CUs to 128 CUs, where 4 CUs is the minimum resource usage of the resource group.
2. For job tasks, the required resources need to be assessed based on the time efficiency of task execution.
For example, in this enterprise, Department B has a scheduled task that runs once per hour. If this task needs to be completed within the current hour, at least 128 CUs of resources are required to ensure timely completion. In this case, the resources for the job can be configured from 4 CUs to 128 CUs.
Department C also has a task that runs once a day. If this task only needs to run in the early hours and should be completed before 8 am, it requires 256 CUs of resources.
Here, we find:
The task running times of Department A and Department B overlap. Thus, you need to configure the maximum amount of resources available for the engine. The formula is as follows: 128 + 128 = 256 CUs.

The task running times of Department B and Department C also overlap. To ensure the tasks are completed on time, you need to configure the maximum amount of resources. The formula is as follows: 128 + 256 = 384 CUs.
Based on the daily running time and usage, the sales team provides the resource purchase suggestion "128 CUs of monthly subscription + 128 CUs of elastic billing".

How to Perform Cost Tracking?

Method 1: Achieving Cost Tracking Through Cost Allocation

DLC supports cost allocation by tag in the engine dimension. By tagging engines, you can view cost allocation by tag in the Billing Center.
As shown in the figure, after different tags are assigned to engines, you can view the cost statistics of Engine 1 and 2 with Tag A, Engine 3 with Tag B, and Engine 2 and 3 with Tag C.

Note:
For details on how to tag an engine, see Associating Tag with Private Engine Resource.
For details on how to achieve cost allocation by tag, see Cost Allocation Tags.

Method 2: Achieving Cost Tracking Through the Task Insight Feature

Enter the Historical Task Instances page of DLC and find the completed task. Click the task name/ID, then you can view the consumed CUs x hours of the task.
Note:
1. Consumed CUs x Hours: Specifies the sum of the CPU execution duration of each core of the Spark executor used in computing. The unit is CU x hour.
2. The resource consumption is the actual consumption generated by the resources occupied by this task. It does not include the statistics such as the resource startup time and idle time of the resource group. Therefore, the consumption value will be much smaller than the total consumption of the resource group and cannot be entirely consistent with the data on the billing side. This consumption value is recommended for analyzing the actual amount of resources used by a single task, for optimizing a single task, or for conducting cost analysis, such as analyzing the usage amount.

How to Perform Task Governance, Enhance the Time Efficiency of Task Completion, and Achieve Cost Optimization?

DLC Insight Management provides a visual and intuitive interface to help you quickly understand the current query performance and potential factors affecting performance, and obtain performance optimization suggestions. For details, see Task Insights.

Applicable Scenarios	1. Insight analysis of the overall running condition of the Spark engine. For example, the resource preemption situation of each task running under the engine, the resource usage situation in the engine, the engine execution duration, the data scan size, and the data shuffle size are all intuitively displayed and analyzed. 2. Self-service troubleshooting and analysis of task running conditions. For example, numerous tasks can be filtered and sorted by time consumption to quickly find problematic large tasks and identify the reasons for slow or failed Spark task execution, including resource preemption, shuffle anomalies, and insufficient disk space, with clear positioning.
Key Metrics	1. Engine execution time: Reflects the time the first task was executed on the Spark engine (namely, the time when the task first preempted the CPU for execution). 2. Execution time within the engine: Reflects the time actually required for computing, namely, the time taken from the start of the first task execution in a Spark job to the completion of the Spark job. 3. Queuing time: Reflects the time taken from a job submission to the start of execution of the first Spark job. The time taken may include the cold startup duration of the first execution of the engine, the queueing time caused by the concurrent limit of the configuration task, the time spent waiting for executor resources due to full resources within the engine, and the queuing time caused by a lack of available resources for allocation to the resource group. 4. Consumed CUs * Hours: Specifies the sum of the CPU execution duration of each core of the Spark executor used in computing. The unit is CU x hour.

Scenario 1: Automatically Analyzing Task Issues to Quickly Locate the Issues and Enhance the Time Efficiency of Task Execution

Inconsistent task quality levels make it difficult to operate and maintain numerous tasks, resulting in low utilization of cluster resources.

Slow task execution mainly results from factors including data skew, insufficient shuffle concurrency, and long-tail tasks that slow down the overall execution time. The DLC Task Insights feature supports analyzing some issues encountered during task execution and providing optimization solutions. When time is limited, prioritize finding the top large tasks for better optimization results.

1. For example, sort by engine execution time (or sort by consumed CUs x hours) and then filter out problematic tasks.
2. For example, since the Spark shuffle phase is a critical factor affecting task execution speed and overall cluster stability, targeted optimization for tasks with large shuffle data volumes can yield good results. Specific operations include sorting tasks by shuffle size, filtering out tasks with shuffle anomalies (even successful tasks may have shuffle retry situations internally), or conducting targeted optimization for top large tasks with large shuffle data volumes. All of these can benefit the overall cluster consumption performance.

Pay close attention to the following metrics:

Insight Type	Algorithm Description (Continuously Improving and Adding Algorithms)
Resource Preemption	The delay time of an SQL execution task is greater than 1 minute after a stage submission, or the delay exceeds 20% of the total runtime (the threshold formula is dynamically adjusted based on task runtime and data volume).
Shuffle Anomaly	Shuffle-related error stack information occurs during stage execution.
Slow Task	The task duration in a stage is greater than twice the average duration of other tasks in the same stage (the threshold formula is dynamically adjusted based on task runtime and data volume).
Data Skew	Task shuffle data is greater than twice the average shuffle data size of other tasks (the threshold formula is dynamically adjusted based on task runtime and data volume).
Disk or Memory Insufficiency	Error stack information during stage execution includes OOM, insufficient disk space, or bandwidth limitation errors of Cloud Object Storage (COS).
Excessive Small File Output	(The collection of this insight type requires upgrading the Spark engine kernel to a version after November 16, 2024) See the metric "Number of Output Small Files" in the list. The presence of excessive small file output is determined if one of the following conditions is met: 1. Partitioned tables: The number of small files written out by a partition exceeds 200. 2. Non-partitioned tables: The total number of output small files exceeds 1000. 3. Partitioned or non-partitioned tables: The total number of files written out exceeds 3000, and the average file size is less than 4 MB.

Scenario 2: Analyze Resource Preemption Situations Rapidly Through Engine Insights, Reasonably Arrange the Number of Running Tasks, and Enhance the Time Efficiency of Task Execution

Slow execution does not necessarily mean slow computation. When cluster resources are limited, resource preemption is likely to occur among tasks. By using the Insight Management feature and considering the 2 metrics of execution time and queuing time in the engine, you can identify tasks that affect each other and reasonably adjust the task queuing plan.

As shown in the figure below, submitting a job does not necessarily mean the engine will start execution immediately. The time taken from a job submission to the start of execution of the first Spark job may include the cold startup duration of the first execution of the engine, the queuing time caused by the concurrent limit of the configuration task, the time spent waiting for executor resources due to full resources within the engine, and the time taken to generate and optimize the Spark execution plan.

When cluster resources are insufficient, the time spent waiting for resources at the preliminary stage is more obvious.

The Task Insights feature also supports automatic identification of resource preemption. Meanwhile, it allows for checking whether other concurrent tasks are preempting resources.

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. 2. Storage-related Ops that users do not want to manage directly can be left to DLC managed storage. 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. 4. Users who want to leverage DLC's automatic data optimization capabilities can continuously optimize data.	1. Iceberg table format. 2. Managed storage must be enabled before use. 3. Data is stored in DLC's managed storage. 4. There is no need to specify external or location information. 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	If the user's write scenario includes upsert, this must be set to true.

	will not be enabled.	
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 bucketing	int, long, decimal, date, time, timestamp, timestamptz, string, uuid, fixed, binary	int
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 partitions	select * from `DataLakeCatalog` . `db` . `sample$partitions`	DLC spark SQL engine, presto engine
Rollback of	CALL DataLakeCatalog. `system` .rollback_to_snapshot('db.sample',	DLC spark SQL

the specific snapshot	1)	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.

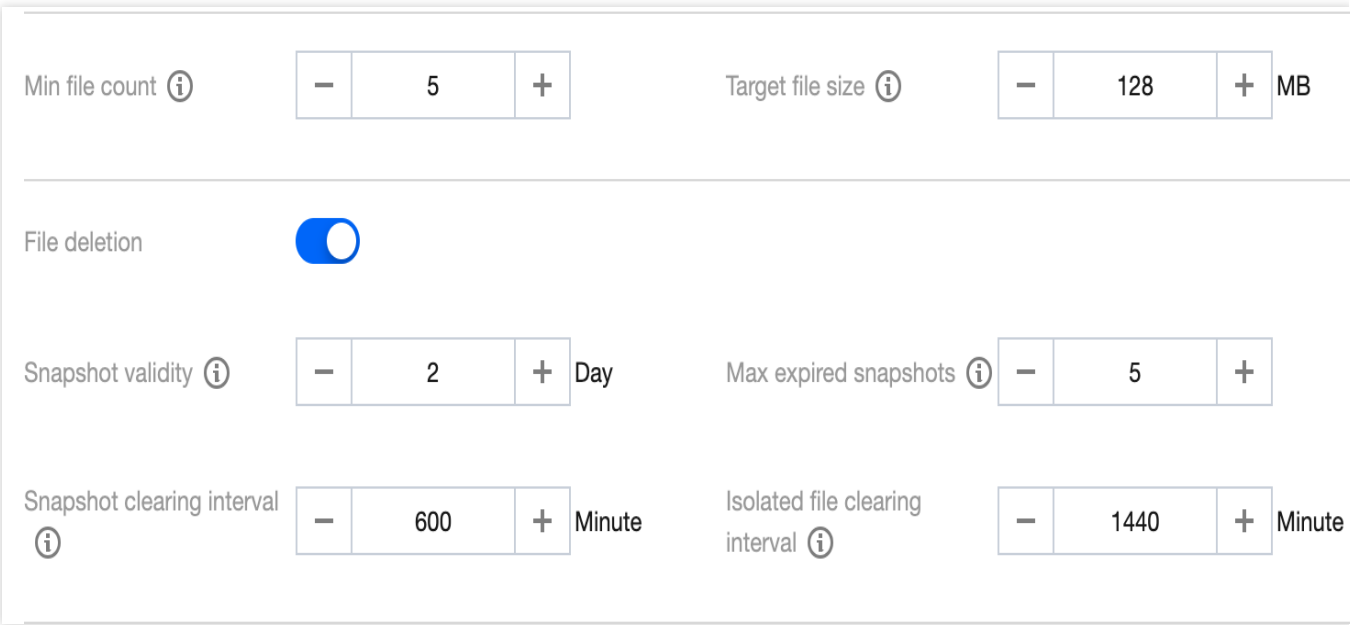
Policy for Optimizing the Configuration of the Data Management: All native tables (Iceberg) in all databases under this data management will by default inherit and use the policy for optimizing the configuration of the data management.

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.



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-input-files	(Optional) Minimum number of files for merging	5	When the number of files under a table or partition exceeds this 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. Merge into write scenarios: It must be enabled. Append write scenarios: It can be enabled as needed.
Data cleanup	Upsert write scenarios: It must be enabled. Merge into write scenarios: It must be enabled. 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-optimizer.lifecycle.expiration	Lifecycle execution cycle, unit: day	30	It can take effect when smart-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.

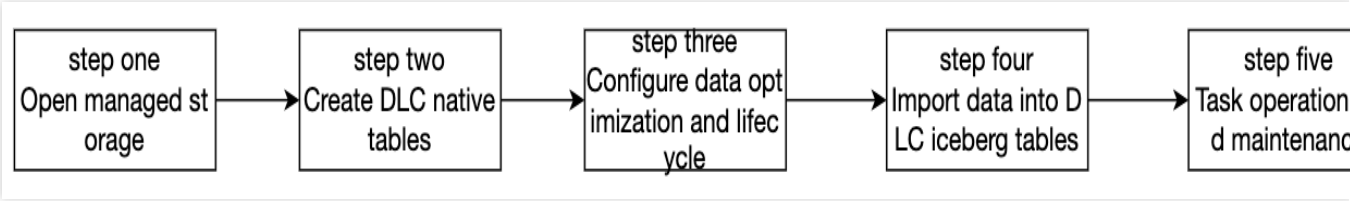
Data location	Import recommendation
Data on the user's own COS bucket	Establish an external table in DLC, then import data using Insert into/overwrite.
Data is on user's local system (or other executors).	Users need to upload data to their own COS buckets, then establish an external table in DLC and 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.

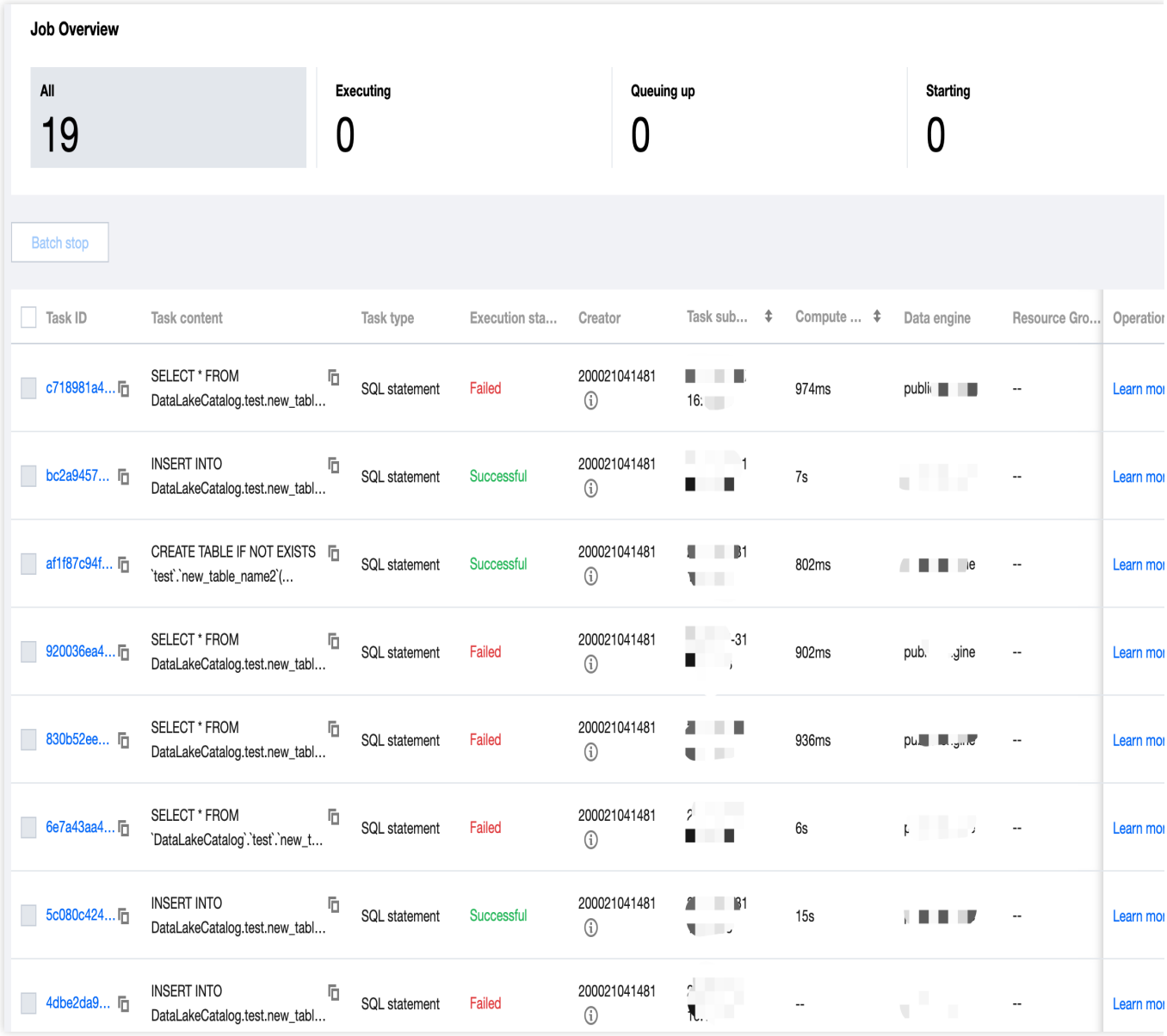
Step V: Viewing Data Optimization Tasks

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.

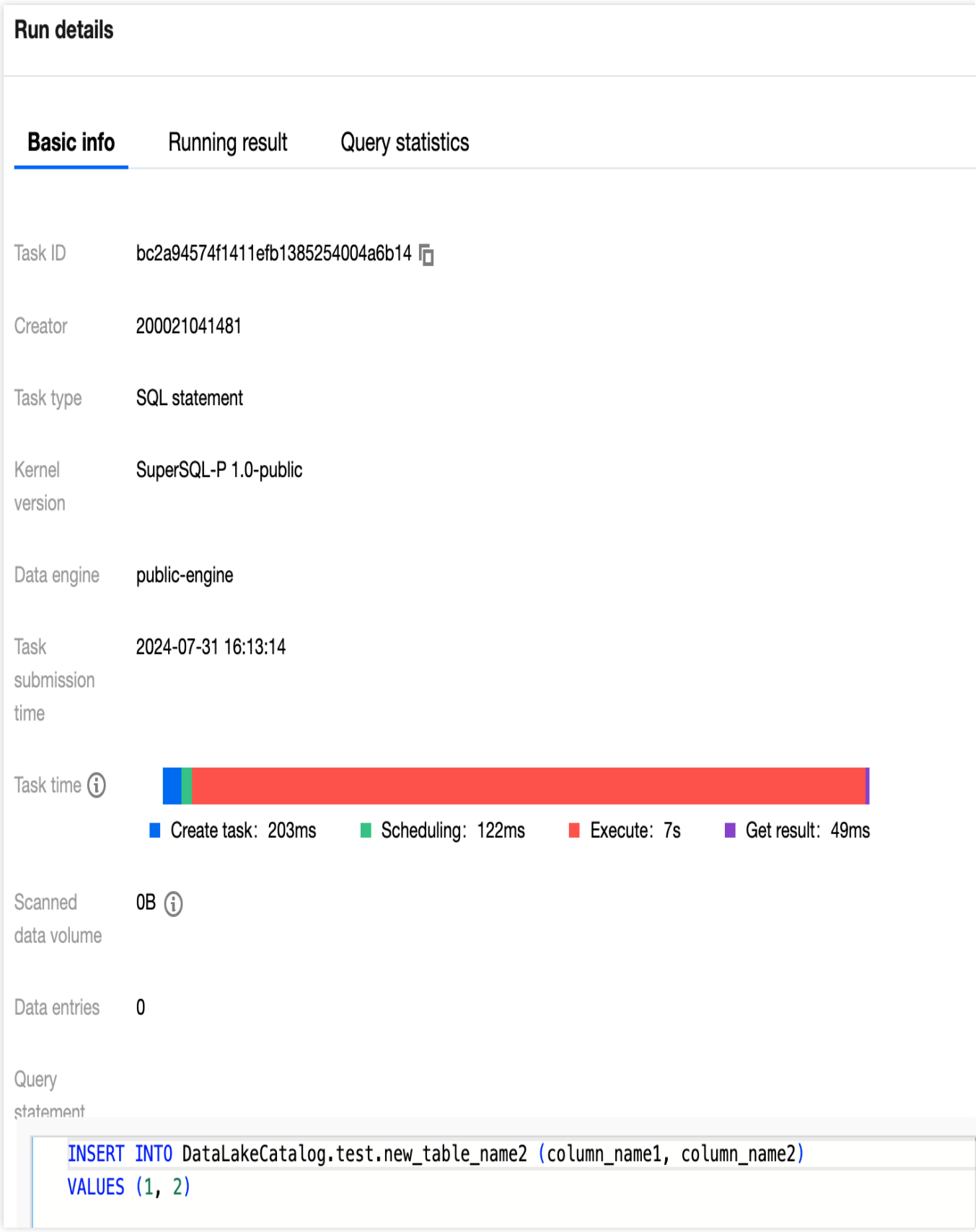
Note:

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.



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
Get result: 49ms
Scanned data volume
0B
Data entries
0
Query statement
INSERT INTO DataLakeCatalog.test.new_table_name2 (column_name1, column_name2)
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:

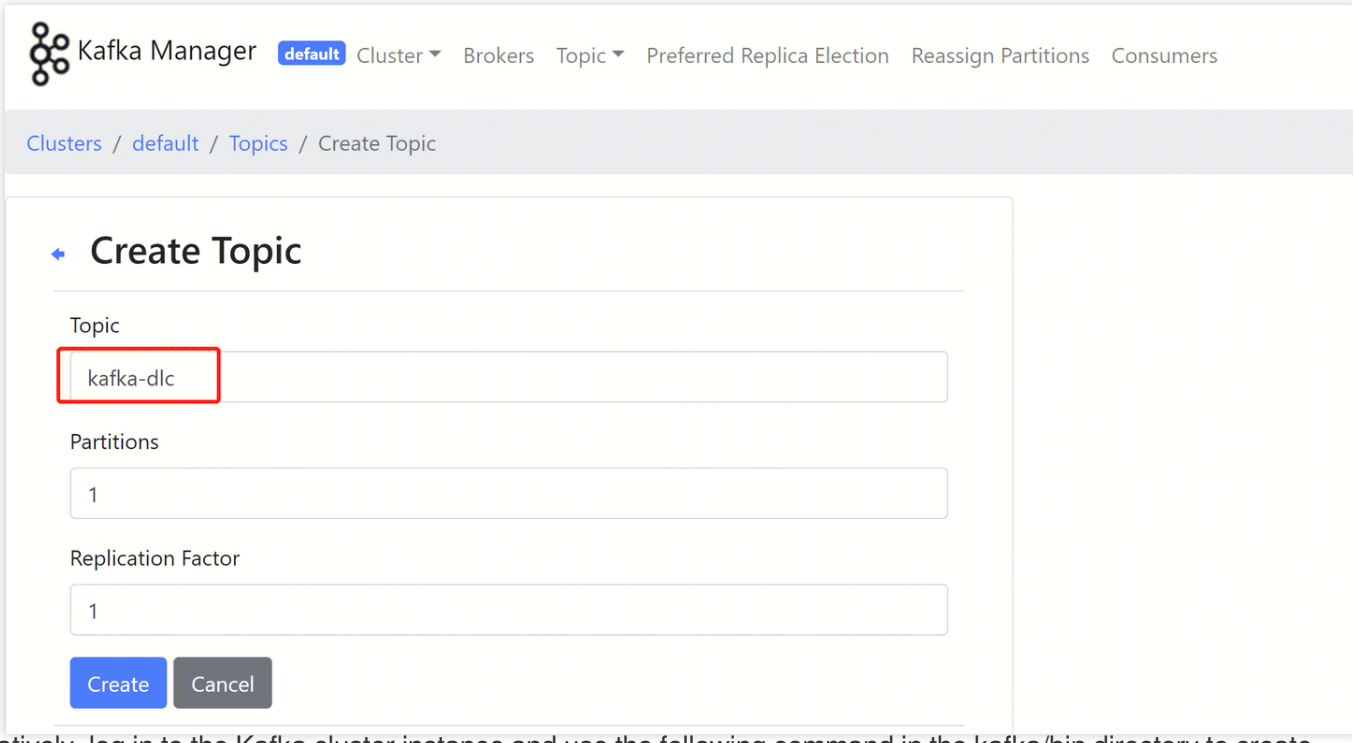
Step 1: Upload Required Jars: Upload the necessary Kafka, DLC connector Jar files, and Hadoop dependency Jars for synchronization.
Step 2: Create Kafka Topic: Create a Kafka topic for production and consumption.
Step 3: Create Target Table in DLC: Create a new target table in DLC data management.
Step 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



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-demo-1.0-SNAPSHOT.jar
```

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-demo-1.0-SNAPSHOT.jar
```

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
```

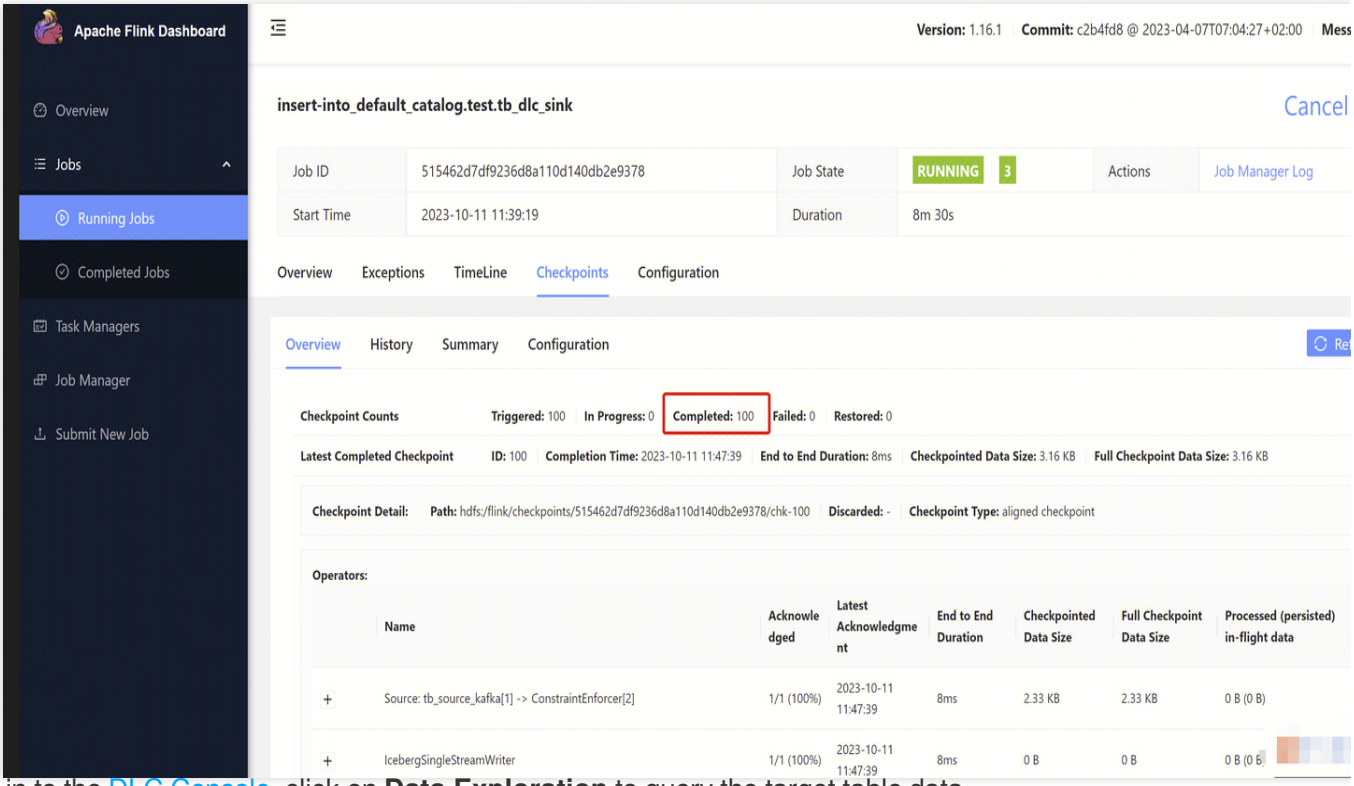
The data information is as follows:

```
{"id":1,"name":"Zhangsan","age":18}
{"id":2,"name":"Lisi","age":19}
{"id":3,"name":"Wangwu","age":20}
{"id":4,"name":"Lily","age":21}
```

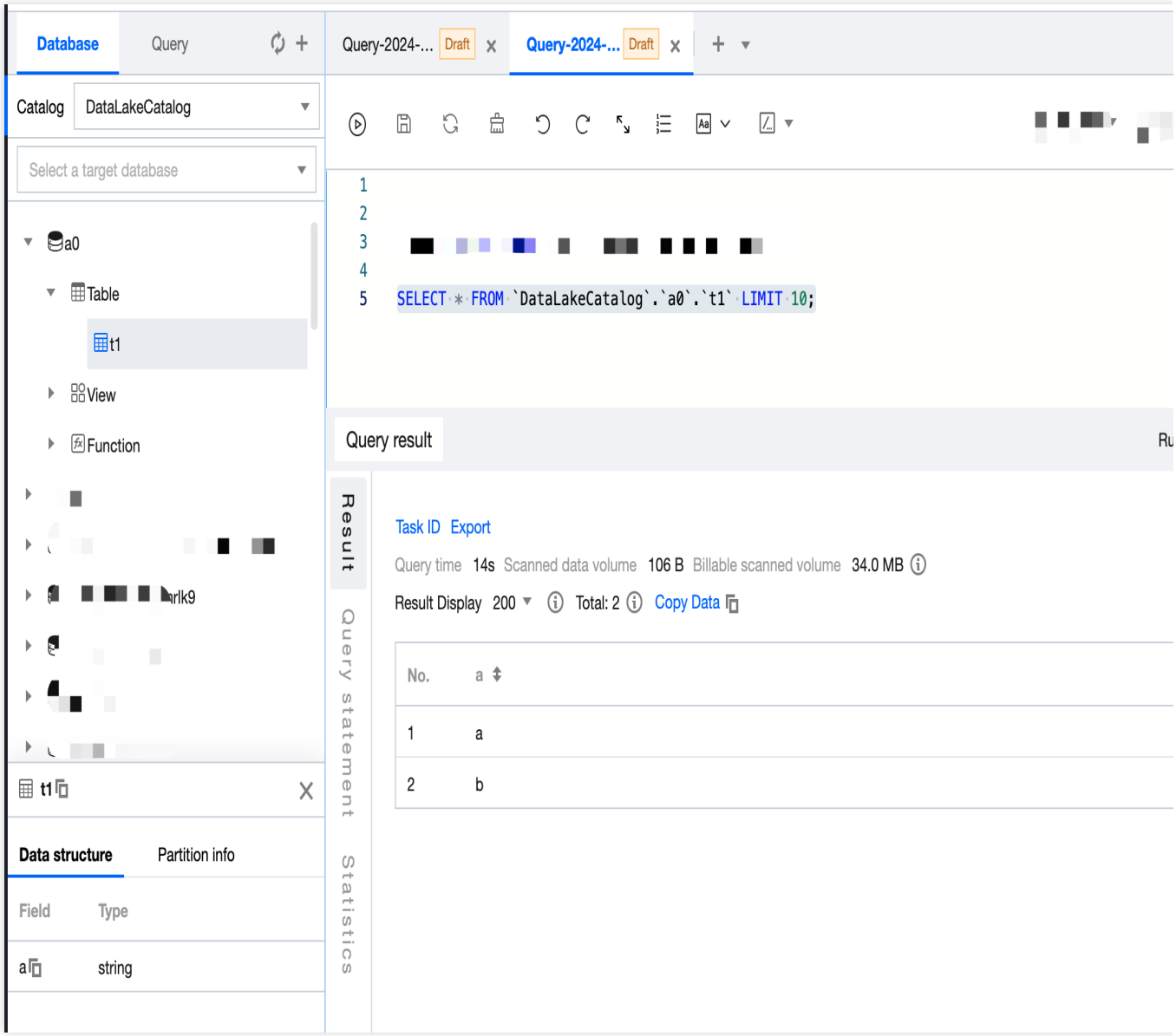
```
{"id":5,"name":"Lucy","age":22}
{"id":6,"name":"Huahua","age":23}
{"id":7,"name":"Wawa","age":24}
{"id":8,"name":"Mei","age":25}
{"id":9,"name":"Joi","age":26}
{"id":10,"name":"Qi","age":27}
{"id":11,"name":"Ky","age":28}
{"id":12,"name":"Mark","age":29}
```

2. Query synchronization results

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



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



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.