Cloud Data Migration

Best Practices

Issue	01
Date	2022-09-30

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: https://www.huawei.com

Email: support@huawei.com

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process.* For details about this process, visit the following web page:
https://www.huawei.com/en/psirt/vul-response-process
For vulnerability information, enterprise customers can visit the following web page:
https://securitybulletin.huawei.com/enterprise/en/security-advisory

Contents

1 Advanced Data Migration Guidance

1.1 Incremental Migration

1.1.1 Incremental File Migration

CDM supports incremental migration of file systems. After full migration is complete, all new files or only specified directories or files can be exported.

Currently, CDM supports the following incremental migration modes:

1. **Exporting the files in a specified directory**
   - Application scenarios: The migration source is a file system (OBS/HDFS/FTP/SFTP). In incremental migration, only the specified files are written to the migration destination. The existing records are not updated or deleted.
   - Key configurations: **File/Path Filter** and Schedule Execution
   - Prerequisites: The source directory or file name contains the time field.
2. **Exporting the files modified after the specified time point**
   - Application scenarios: The migration source is a file system (OBS/HDFS/FTP/SFTP). The specified time point refers to the time when the file is modified. CDM migrates the files modified at or after the specified time point.
   - Key configurations: **Time Filter** and Schedule Execution
   - Prerequisites: None

NOTE

If you have configured a macro variable of date and time and schedule a CDM job through DataArts Studio DataArts Factory, the system replaces the macro variable of date and time with (*Planned start time of the data development job – Offset*) rather than (*Actual start time of the CDM job – Offset*).

File/Path Filter

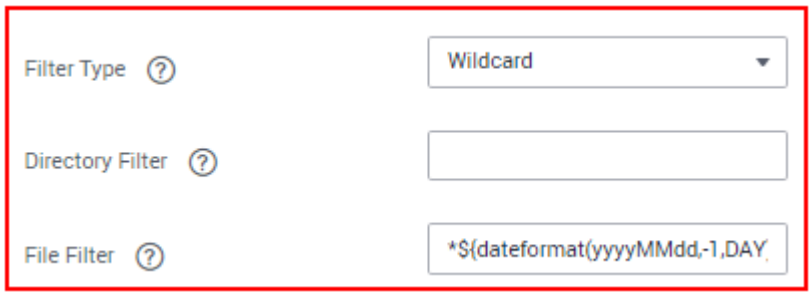
- Parameter position: When creating a table/file migration job, if the migration source is a file system, set **Filter Type** in advanced attributes of **Source Job Configuration** to **Wildcard** or **Regular expression**.
- Parameter principle: If you select **Wildcard** for **Filter Type**, CDM filters files or paths based on the configured wildcard character and migrates only files or paths that meet the specified condition.
- Example configurations:

  Suppose that the source file name contains the date and time field, such as **2017-10-15 20:25:26**, the **/opt/data/file_20171015202526.data** file is generated. Set the parameters as follows:

  a. **Filter Type**: Select **Wildcard**.

  b. **File Filter**: Enter **"*${dateformat(yyyyMMdd,-1,DAY)}*"**, which is the format of the macro variables of date and time supported by CDM. For details, see **Using Macro Variables of Date and Time**.

  Figure 1-1 Filtering files

  c. Schedule Execution: Set **Cycle (days)** to **1**.

In this way, you can import the files generated in the previous day to the destination directory every day to implement incremental synchronization.

In incremental file migration, **Path Filter** is used in the same way as **File Filter**. The path name must contain the time field. In this case, all files in the specified path can be synchronized periodically.

Time Filter

- Parameter position: When creating a table/file migration job, if the migration source is a file system, set select **Yes** for **Time Filter**.
- Parameter principle: After you specify the start time and end time, only files that are modified between the start time (included) and end time (excluded) will be migrated.
- Example configurations:

  For example, if you want CDM to synchronize only the files generated from January 1, 2021 to January 1, 2022 to the destination, configure the following parameters:

  a. **Time Filter**: select **Yes**.

  b. **Minimum Timestamp**: Enter a value in the format of *yyyy-MM-dd HH:mm:ss*, such as **2021-01-01 00:00:00**.

c. **Maximum Timestamp**: Enter a value in the format of *yyyy-MM-dd HH:mm:ss*, such as **2022-01-01 00:00:00**.

Figure 1-2 Time Filter

In this way, the CDM job migrates only the files generated from January 1, 2021 to January 1, 2022, and performs incremental synchronization next time it is started.

1.1.2 Incremental Migration of Relational Databases

CDM supports incremental migration of relational databases. After a full migration is complete, data in a specified period can be incrementally migrated. For example, data added on the previous day can be exported at 00:00:00 every day.

- **Migrating incremental data within a specified period of time**
  - Application scenarios: The source end is a relational database. The destination end can be of any type.
  - Key configurations: **WHERE Clause** and Schedule Execution
  - Prerequisites: The data table contains a date and time field or timestamp field.

In incremental migration, only the specified data is written to the data table. The existing records are not updated or deleted.

NOTE

If you have configured a macro variable of date and time and schedule a CDM job through DataArts Studio DataArts Factory, the system replaces the macro variable of date and time with (*Planned start time of the data development job – Offset*) rather than (*Actual start time of the CDM job – Offset*).

WHERE Clause

- Parameter position: When creating a table/file migration job, if the source end is a relational database, the **Where Clause** parameter is available in the advanced attributes of **Source Job Configuration**.
- Parameter principle: Set **WHERE Clause** to an SQL statement, for example, **age > 18 and age <= 60**, CDM exports only the data that meets the SQL statement requirement. If **WHERE Clause** is not specified, the entire table is exported.

  Where Clause can be set to **macro variables of date and time**. When the data table contains the **date** or **timestamp** field, **Where Clause** and Schedule Execution can be used together to extract data of a specified date.

- Example configurations:

  Suppose that the database table contains column **DS** indicating the time, the value type of the column is **varchar(30)**, and the inserted time format is similar to *2017-xx-xx*. See **Figure 1-3**. Set the parameters as follows:

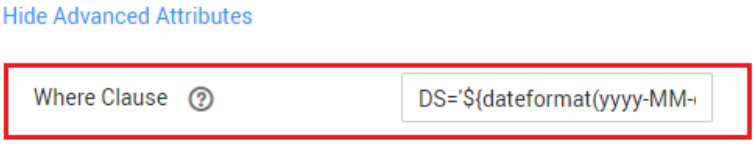
  Figure 1-3 Table data

  SELECT * FROM SQOOP.CDM_20171016

	FOO	BAR	DS
1	5	s	2017-05-01
2	5	s	2017-05-01
3	1	g	2017-05-02
4	4	o	2017-05-02
5	6	a	2017-05-02
6	7	n	2017-05-02
7	1	g	2017-05-02
8	4	o	2017-05-02
9	6	a	2017-05-02
10	7	n	2017-05-02
11	2	fa	2017-10-15
12	3	te	2017-10-15
13	2	fa	2017-10-15
14	3	te	2017-10-15

  a. **WHERE Clause**: Set this parameter to **DS='${dateformat(yyyy-MM-dd,-1,DAY)}'**.

  Figure 1-4 WHERE Clause

  b. Scheduling job execution: Set **Cycle (days)** to **1** and **Start Time** to **00:00:00**.

  In this way, all data generated on the previous day can be exported at 00:00:00 every day. **WHERE Clause** can be configured to various **macro variables of date and time**. You can use the macro variables of date and time and scheduled jobs with specified cycle of minutes, hours, days, weeks, or months together to automatically export data at a specific time.

1.1.3 HBase/CloudTable Incremental Migration

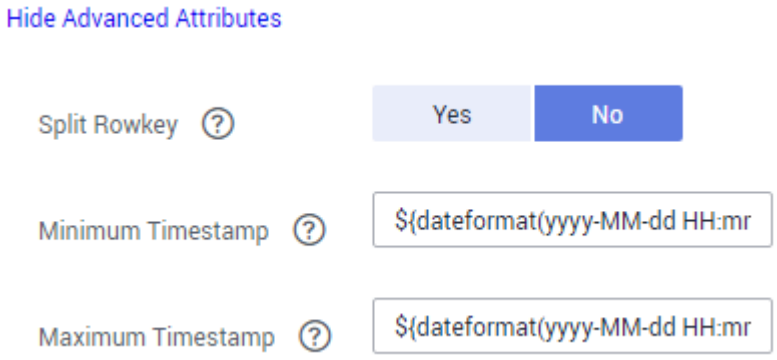
You can use CDM to export data in a specified period of time from HBase (including MRS HBase, FusionInsight HBase, and Apache HBase) and CloudTable. The CDM scheduled jobs can be used together to implement incremental migration of HBase and CloudTable.

NOTE

If you have configured a macro variable of date and time and schedule a CDM job through DataArts Studio DataArts Factory, the system replaces the macro variable of date and time with (*Planned start time of the data development job – Offset*) rather than (*Actual start time of the CDM job – Offset*).

When creating a table/file migration job and selecting the link to HBase or CloudTable as the source link, you can set the time range in advanced attributes.

Figure 1-5 Time range



- Start time (including the value) for extracting data. The format is *yyyy-MM-dd HH:mm:ss*. Only the data generated at the specified time and later is extracted.
- End time (excluding the value) for extracting data. The format is *yyyy-MM-dd HH:mm:ss*. Only the data generated before the time point is extracted.

The two parameters can be set to **macro variables of date and time**. Examples are as follows:

- If **Minimum Timestamp** is set to **${dateformat(yyyy-MM-dd HH:mm:ss, -1, DAY)}**, only the data generated after the day before is exported.
- If **Maximum Timestamp** is set to **${dateformat(yyyy-MM-dd HH:mm:ss)}**, only the data generated before the specified time point is exported.

If both parameters are configured, CDM exports only the data generated on the previous day. In addition, if the job is configured to execute at 00:00:00 every day, the data generated every day can be incrementally synchronized.

1.2 Using Macro Variables of Date and Time

During the creation of table/file migration jobs, CDM supports the macro variables of date and time in the following parameters of the source and destination links:

- Source directory or file
- Source table name
- Directory filter and file filter of the **wildcard** type
- Start time and end time of the **time filter** type
- Partition filter criteria and where clause

- Write directory
- Destination table name

You can use the **${}** macro variable definition identifier to define the macros of the time type. currently, dateformat and timestamp are supported.

By using the macro variables of date and time and scheduled job, you can implement incremental synchronization of databases and files.

NOTE

If you have configured a macro variable of date and time and schedule a CDM job through DataArts Studio DataArts Factory, the system replaces the macro variable of date and time with (*Planned start time of the data development job – Offset*) rather than (*Actual start time of the CDM job – Offset*).

dateformat

dateformat supports two types of parameters:

- **dateformat(format)**

  format indicates the date and time format. For details about the format definition, see the definition in **java.text.SimpleDateFormat.java**.

  For example, if the current date is **2017-10-16 09:00:00**, **yyyy-MM-dd HH:mm:ss** indicates **2017-10-16 09:00:00**.
- dateformat(format, dateOffset, dateType)
  - **format** indicates the format of the returned date.
  - **dateOffset** indicates the date offset.
  - **dateType** indicates the type of the date offset.

    Currently, **dateType** supports SECOND, MINUTE, HOUR, MONTH, YEAR, and DAY.

    NOTE

    Pay attention to the following special scenarios of **MONTH** and **YEAR**:
    - If the date does not exist after the offset, the latest date of the month in the calendar is used.
    - These two offset types cannot be used for the start time and end time in the **Time Filter** parameter of the source and destination jobs.

  For example, if the current date is **2023-03-01 09:00:00**, then:
  - **dateformat(yyyy-MM-dd HH:mm:ss, -1, YEAR)** indicates the year before the current time, that is, **2022-03-01 09:00:00**.
  - **dateformat(yyyy-MM-dd HH:mm:ss, -3, MONTH)** indicates three months before the current time, that is, **2022-12-01 09:00:00**.
  - **dateformat(yyyy-MM-dd HH:mm:ss, -1, DAY)** indicates the day before the current time, that is, **2023-02-28 09:00:00**.
  - **dateformat(yyyy-MM-dd HH:mm:ss, -1, HOUR)** indicates one hour before the current time, that is, **2023-03-01 08:00:00**.
  - **dateformat(yyyy-MM-dd HH:mm:ss, -1, MINUTE)** indicates one minute before the current time, that is, **2023-03-01 08:59:00**.
  - **dateformat(yyyy-MM-dd HH:mm:ss, -1, SECOND)** indicates one second before the current time, that is, **2023-03-01 08:59:59**.

timestamp

timestamp supports two types of parameters:

- **timestamp()**

  Indicates the returned timestamp of the current time, that is, the number of milliseconds that have elapsed since 00:00:00 on January 1, 1970 (1970-01-01 00:00:00 GMT). For example, 1508078516286.

- **timestamp(dateOffset, dateType)**

  Indicates the timestamp returned after time offset. **dateOffset** and **dateType** indicate the date offset and the offset type, respectively.

  For example, if the current date is **2017-10-16 09:00:00**, **timestamp(-10, MINUTE)** indicates that the timestamp generated 10 minutes before the current time point is returned, that is, **1508115000000**.

Macro Variable Definition of Time and Date

Suppose that the current time is **2017-10-16 09:00:00**, then **Table 1-1** describes the macro variable definitions of time and date.

NOTE

The examples in the table must be embedded in ''. For example, '${dateformat(yyyy-MM-dd)}' returns the current time in yyyy-MM-dd format.

Table 1-1 Macro variable definition of time and date

Macro Variable	Description	Display Effect
${dateformat(yyyy-MM-dd)}	Returns the current date in **yyyy-MM-dd** format.	2017-10-16
${dateformat(yyyy/MM/dd)}	Returns the current date in **yyyy/MM/dd** format.	2017/10/16
${dateformat(yyyy_MM_dd HH:mm:ss)}	Returns the current time in **yyyy_MM_dd HH:mm:ss** format.	2017_10_16 09:00:00
${dateformat(yyyy-MM-dd HH:mm:ss, -1, DAY)}	Returns the current time in **yyyy-MM-dd HH:mm:ss** format. The date is one day before the current day.	2017-10-15 09:00:00
${dateformat(yyyy-MM-dd, -1, DAY)} 00:00:00	Returns 00:00:00 of the day before the current day in *yyyy-MM-dd HH:mm:ss* format.	2017-10-15 00:00:00
${dateformat(yyyy-MM-dd, -1, DAY)} 12:00:00	Returns 12:00:00 of the day before the current day in *yyyy-MM-dd HH:mm:ss* format.	2017-10-15 12:00:00

Macro Variable	Description	Display Effect
${dateformat(yyyy-MM-dd, -N, DAY)} 00:00:00	Returns 00:00:00 of the day N days before the current day in *yyyy-MM-dd HH:mm:ss* format.	When N is 3: 2017-10-13 00:00:00
${dateformat(yyyy-MM-dd, -N, DAY)} 12:00:00	Returns 12:00:00 of the day N days before the current day in *yyyy-MM-dd HH:mm:ss* format.	When N is 3: 2017-10-13 12:00:00
${timestamp()}	Returns the timestamp of the current time, that is, the number of milliseconds that have elapsed since 00:00:00 on January 1, 1970.	1508115600000
${timestamp(-10, MINUTE)}	Returns the timestamp generated 10 minutes before the current time point.	1508115000000
${timestamp(dateformat(yyyyMMdd))}	Returns the timestamp of 00:00:00 of the current day.	1508083200000
${timestamp(dateformat(yyyyMMdd,-1,DAY))}	Returns the timestamp of 00:00:00 of the previous day.	1507996800000
${timestamp(dateformat(yyyyMMddHH))}	Returns the timestamp of the current hour.	1508115600000

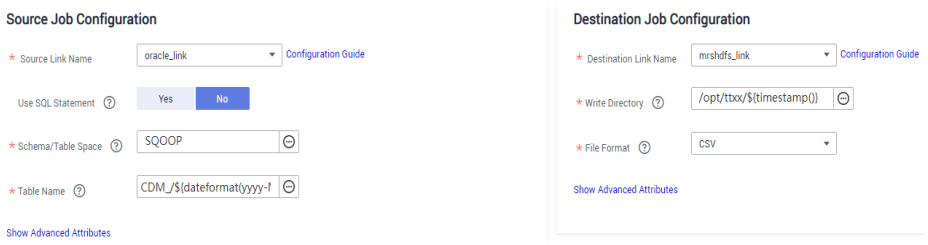
Time and Date Macro Variables of Paths and Table Names

Figure 1-6 shows an example. If:

- **Table Name** under **Source Link Configuration** is set to **CDM_/${dateformat(yyyy-MM-dd)}**.
- **Write Directory** under **Destination Link Configuration** is set to **/opt/ttxx/${timestamp()}**.

After the macro definition conversion, this job indicates that data in table **SQOOP.CDM_20171016** in the Oracle database is migrated to the **/opt/ttxx/1508115701746** directory of the HDFS server.

Figure 1-6 Setting **Table Name** and **Write Directory** to a time and date macro variable



Currently, a table name or path name can contain multiple macro variables. For example, **/opt/ttxx/${dateformat(yyyy-MM-dd)}/${timestamp()}** is converted to **/opt/ttxx/2017-10-16/1508115701746**.

Time and Date Macro Variables in the Where Clause

Figure 1-7 uses table **SQOOP.CDM_20171016** as an example. The table contains column **DS**, which indicates the time.

Figure 1-7 Table data

SELECT * FROM SQOOP.CDM_20171016

	FOO	BAR	DS
1	5	snap	2017-05-01
2	5	snap	2017-05-01
3	1	google	2017-05-02
4	4	oracle	2017-05-02
5	6	amd	2017-05-02
6	7	nvda	2017-05-02
7	1	google	2017-05-02
8	4	oracle	2017-05-02
9	6	amd	2017-05-02
10	7	nvda	2017-05-02
11	2	facebook	2017-10-15
12	3	tesla	2017-10-15
13	2	facebook	2017-10-15
14	3	tesla	2017-10-15

Suppose that the current date is **2017-10-16** and you want to export data generated the day before the current day (DS = 2017-10-15), then you can set the value of **Where Clause** to **DS='${dateformat(yyyy-MM-dd,-1,DAY)}'** when creating a job. In this way, you can export all data that complies with the DS = 2017-10-15 condition.

Implementing Incremental Synchronization by Configuring the Macro Variables of Date and Time and Scheduled Jobs

Two simple application scenarios are as follows:

- The database table contains column **DS** that indicates the time, the value type of the column is **varchar(30)**, and the inserted time format is similar to **2017-xx-xx**.

  In a scheduled job, the cycle is one day, and the scheduled job is executed at 00:00:00 every day. Set the value of **Where Clause** to **DS='${dateformat(yyyy-MM-dd,-1,DAY)}'**, and then data generated in the previous day will be exported at 00:00:00 every day.

- The database table contains column **time** that indicates the time, the type is **Number**, and the inserted time format is timestamp.

  In a scheduled job, the cycle is one day, and the scheduled job is executed at 00:00:00 every day. Set the value of **Where Clause** to **time between ${timestamp(-1,DAY)} and ${timestamp()}**, and then data generated on the previous day will be exported at 00:00:00 every day.

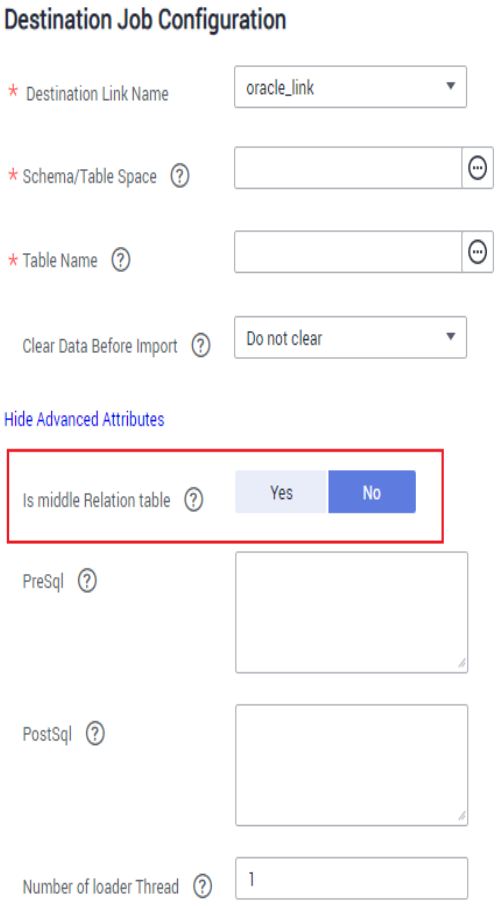
Configuration principles of other application scenarios are the same.

1.3 Migration in Transaction Mode

When a CDM job fails to be executed, CDM rolls back the data to the state before the job starts and automatically deletes data from the destination table.

- Parameter position: When creating a table/file migration job, if the migration source is a relational database, set **Import to Staging Table** in the advanced attributes of **Destination Job Configuration** to determine whether to enable the transaction mode.
- Parameter principle: If you set this parameter to **Yes**, CDM automatically creates a temporary table and imports the data to the temporary table. After the data is imported successfully, CDM migrates the data to the destination table in transaction mode of the database. If the import fails, the destination table is rolled back to the state before the job starts.

Figure 1-8 Migration in transaction mode



NOTE

If you select **Clear part of data** or **Clear all data** for **Clear Data Before Import**, CDM does not roll back the deleted data in transaction mode.

1.4 Encryption and Decryption During File Migration

When you migrate files to a file system, CDM can encrypt and decrypt those files. Currently, CDM supports the following encryption modes:

- **AES-256-GCM**
- **KMS Encryption**

AES-256-GCM

Currently, only AES-256-GCM (NoPadding) is supported. This algorithm is used for encryption at the migration destination and decryption at the migration source. The supported source and destination data sources are as follows:

- Data sources supported by the migration source: HDFS (supported in the binary format)
- Data sources supported by the migration destination: HDFS (supported in the binary format)

The following part describes how to use AES-256-GCM to decrypt the encrypted files to be exported from HDFS and encrypt the files to be imported to HDFS.

- **Configure decryption at the migration source.**

  When you use CDM to create a job for exporting files from HDFS, set the migration source to HDFS and file format to binary, and set the following parameters in the advanced settings of **Source Job Configuration**:

  a. **Encryption**: Select **AES-256-GCM**.

  b. **DEK**: The key must be the same as that configured in encryption. Otherwise, the decrypted data is incorrect and the system does not display an error message.

  c. **IV**: The initialization vector must be the same as that configured in encryption. Otherwise, the decrypted data is incorrect and the system does not display an error message.

  In this way, after CDM exports encrypted files from HDFS, the files written to the migration destination are decrypted plaintext files.

- **Configure encryption at the migration destination.**

  When you create a CDM job to import files to HDFS, set the migration destination to HDFS and file format to binary, and set the following parameters in the advanced settings of **Destination Job Configuration**:

  a. **Encryption**: Select **AES-256-GCM**.

  b. **DEK**: custom encryption key. The key consists of 64 hexadecimal numbers. It is case-insensitive but must contain 64 characters. For example, **DD0AE00DFECD78BF051BCFDA25BD4E320DB0A7AC75A1F3FC3D3C56A457DCDC1B**.

  c. **IV**: custom initialization vector. The initialization vector consists of 32 hexadecimal numbers. It is case-insensitive but must contain 32 characters. For example, **5C91687BA886EDCD12ACBC3FF19A3C3F**.

  In this way, after CDM imports files to HDFS, the files in the destination HDFS are encrypted using the AES-256-GCM algorithm.

KMS Encryption

NOTE

The migration source does not support KMS encryption.

CDM supports KMS encryption if tables, files, or a whole database is migrated to OBS. In the **Advanced Attributes** area of the **Destination Job Configuration** page, set the parameters.

A key must be created in KMS of DEW in advance. For details, see the *Data Encryption Workshop User Guide*.

After KMS encryption is enabled, objects to be uploaded will be encrypted and stored on OBS. When you download the encrypted objects, the encrypted data will be decrypted on the server and displayed in plaintext to users.

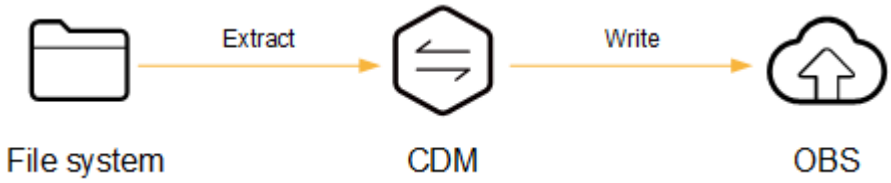
NOTE

- If KMS encryption is enabled, **MD5 verification** cannot be used.
- If the KMS ID of another project is used, change **Project ID** to the ID of the project to which KMS belongs. If KMS and CDM are in the same project, retain the default value of **Project ID**.
- After KMS encryption is performed, the encryption status of the objects on OBS cannot be changed.
- A key in use cannot be deleted. Otherwise, the object encrypted with this key cannot be downloaded.

1.5 MD5 Verification

CDM extracts data from the migration source and writes the data to the migration destination. **Figure 1-9** shows the migration mode when files are migrated to OBS.

Figure 1-9 Migrating files to OBS



During the process, CDM uses MD5 to verify file consistency.

- **Extract**
  - The migration source can be OBS, HDFS, FTP, SFTP, or HTTP. It can check whether the files extracted by CDM are consistent with source files.
  - This function is controlled by the **MD5 File Extension** parameter (available when **File Format** is set to **Binary**) in **Source Job Configuration**. Set this parameter to the file name extension of the MD5 file in the source file system.
  - If a source file **build.sh** and a file for saving MD5 value **build.sh.md5** are located in the same directory, and **MD5 File Extension** is configured, only the file **build.sh.md5** is migrated to the destination. Files without the MD5 value or whose MD5 values do not match fail to be migrated, and the MD5 file is not migrated.
  - If **MD5 File Extension** is not configured, all files are migrated.
- **Write**
  - Currently, this function can be used only when OBS serves as the migration destination. It can check whether the files written to OBS are consistent with those extracted from CDM.
  - This function is controlled by the **Validate MD5 Value** parameter in **Destination Job Configuration**. After the files are read and written to OBS, the MD5 value in the HTTP header is used to verify the files on OBS and the verification result is written to an OBS bucket (the bucket can be the one that does not store migration files). If the migration source does not have the MD5 file, the verification will not be performed.

NOTE

- When files are migrated to a file system, only the extracted files are verified.
- When files are migrated to OBS, both the extracted files and files written to OBS are verified.
- If MD5 verification is used, **KMS encryption** cannot be used.

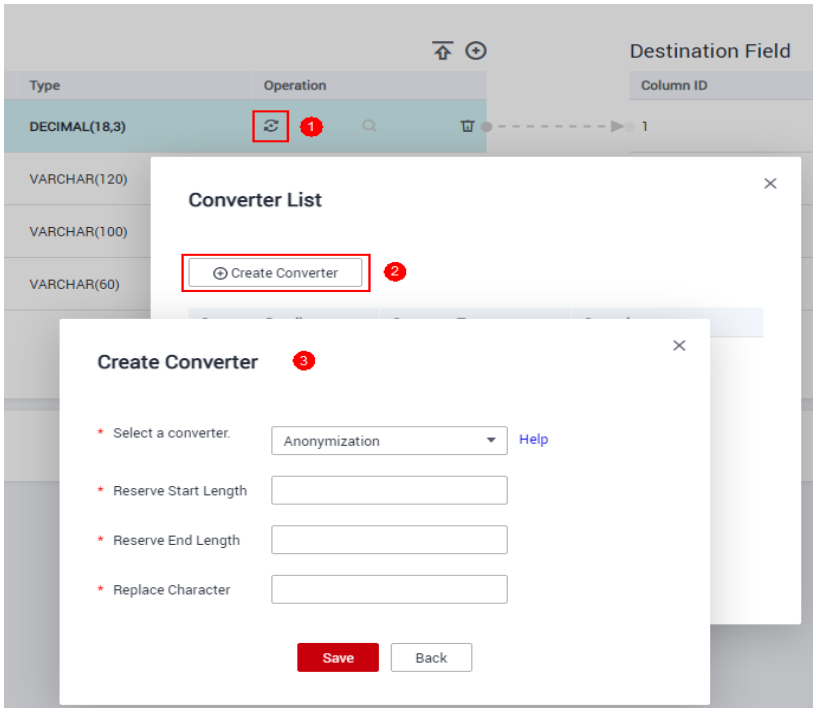
1.6 Configuring Field Converters

Scenario

- After the job parameters are configured, field mapping needs to be configured. You can click in the **Operation** column to create a field converter.
- If files are migrated between FTP, SFTP, OBS, and HDFS and the migration source's **File Format** is set to **Binary**, files will be directly transferred, free from field mapping.

You can create a field converter on the **Map Field** page when creating a table/file migration job.

Figure 1-10 Creating a field converter



CDM can convert fields during migration. Currently, the following field converters are supported:

- **Anonymization**
- **Trim**
- **Reverse String**

- **Replace String**
- **Remove line break**
- **Expression Conversion**

Constraints

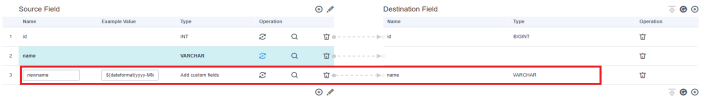
- If **Use SQL Statement** is set to **Yes** in the source job configuration, converters cannot be created.
- On the **Map Field** tab page, if CDM fails to obtain all columns by obtaining sample values (for example, when data is exported from HBase, CloudTable, or MongoDB, there is a high probability that CDM failed to obtain all columns), you can click  and select **Add a new field** to add new fields to ensure that the data imported to the migration destination is complete.
- When a relational database, Hive, DLI, or MRS Hudi is used as the migration source, sample values cannot be obtained.
- When SQLServer is the destination, fields of the timestamp type cannot be written. You must change their type (for example, to datetime) so that they can be written.
- Column names are displayed when the source of the migration job is OBS, CSV files are to be migrated, and parameter **Extract first row as columns** is set to **Yes**.
- Field converters configuration is not involved when the binary format is used to migrate files to files.
- In the automatic table creation scenario, you need to manually add fields to the destination table in advance and then add fields to the field mapping.
- After a field is added, its sample value is not displayed on the console. This does not affect the field value transmission. CDM directly writes the field value to the destination end.
- If the field mapping is incorrect, you can adjust the field mapping by dragging fields or clicking  to map fields in batches.
- An expression processes the data of a field. When you create an expression converter, do not use a time macro. If you need to use a time macro, use either of the following methods (if the source is of the file type, only **Method 1** is supported):
  - Method 1: When creating an expression converter, use two single quotation marks ('') to enclose the expression.

    For example, if expression **${dateformat(yyyy-MM-dd)}** is not enclosed in quotation marks, the hyphen (-) in the value **2017-10-16** parsed from the expression will be recognized as a minus sign, and further calculation will be performed to generate result **1991**, which is incorrect. If you enclose the expression in quotation marks, that is, **'${dateformat(yyyy-MM-dd)}'**, you will obtain **'2017-10-16'**, which is correct.

Figure 1-11 Using two single quotation marks ('') to enclose an expression

- Method 2: Add a custom source field, enter a macro variable of date and time for **Example Value**, and map the field to a destination field again.

  Figure 1-12 Adding a custom source field

- If the data is imported to GaussDB(DWS), you need to select the distribution columns in the destination fields. You are advised to select the distribution columns according to the following rules:
  a. Use the primary key as the distribution column.
  b. If multiple data segments are combined as primary keys, specify all primary keys as the distribution column.
  c. In the scenario where no primary key is available, if no distribution column is selected, DWS uses the first column as the distribution column by default. As a result, data skew risks exist.

Anonymization

This converter is used to hide key information about the character string. For example, if you want to convert **12345678910** to **123****8910**, configure the parameters as follows:

- Set **Reserve Start Length** to **3**.
- Set **Reserve End Length** to **4**.
- Set **Replace Character** to *****.

Trim

This converter is used to automatically delete the spaces before and after a string. No parameters need to be configured.

Reverse String

This converter is used to automatically reverse a string. For example, reverse **ABC** into **CBA**. No parameters need to be configured.

Replace String

This converter is used to replace a character string. You need to configure the object to be replaced and the new value.

Remove line break

This converter is used to delete the newline characters, such as \n, \r, and \r\n from the field.

Expression Conversion

This converter uses the JSP expression language (EL) to convert the current field or a row of data. The JSP EL is used to create arithmetic and logical expressions. In an expression, you can use integers, floating point numbers, strings, constants **true** and **false**, and **null**.

During data conversion, if the content to be replaced contains a special character, use a backslash (\) to escape the special character to a common one.

- The expression supports the following environment variables:
  - **value**: indicates the current field value.
  - **row**: indicates the current row, which is an array type.
- The expression supports the following Utils:
  a. If the field is of the string type, convert all character strings into lowercase letters, for example, convert **aBC** to **abc**.

     Expression: StringUtils.lowerCase(value)

  b. Convert all character strings of the current field to uppercase letters.

     Expression: StringUtils.upperCase(value)

  c. Convert the format of the first date field from 2018-01-05 15:15:05 to 20180105.

     Expression: DateUtils.format(DateUtils.parseDate(row[0],"yyyy-MM-dd HH:mm:ss"),"yyyyMMdd")

  d. Convert a timestamp to a date string in *yyyy-MM-dd hh:mm:ss* format, for example, convert **1701312046588** to **2023-11-30 10:40:46**.

     Expression: DateUtils.format(NumberUtils.toLong(value),"yyyy-MM-dd HH:mm:ss")

  e. Convert a date string in the yyyy-MM-dd hh:mm:ss format to a timestamp.

     Expression: DateUtils.getTime(DateUtils.parseDate(value,"yyyy-MM-dd hh:mm:ss"))

  f. If the field value is a date string in *yyyy-MM-dd* format, extract the year from the field value, for example, extract **2017** from **2017-12-01**.

     Expression: StringUtils.substringBefore(value,"-")

g. If the field value is of the numeric type, convert the value to a new value which is two times greater than the original value:

   Expression: value*2

h. Convert the field value **true** to **Y** and other field values to **N**.

   Expression: value=="true"?"Y":"N"

i. If the field value is of the string type and is left empty, convert it to **Default**. Otherwise, the field value will not be converted.

   Expression: empty value? "Default":value

j. Convert date format **2018/01/05 15:15:05** to **2018-01-05 15:15:05**:

   Expression: DateUtils.format(DateUtils.parseDate(value,"yyyy/MM/dd HH:mm:ss"),"yyyy-MM-dd HH:mm:ss")

k. Obtain a 36-bit universally unique identifier (UUID):

   Expression: CommonUtils.randomUUID()

l. If the field is of the string type, capitalize the first letter, for example, convert **cat** to **Cat**.

   Expression: StringUtils.capitalize(value)

m. If the field is of the string type, convert the first letter to a lowercase letter, for example, convert **Cat** to **cat**.

   Expression: StringUtils.uncapitalize(value)

n. If the field is of the string type, use a space to fill in the character string to the specified length and center the character string. If the length of the character string is not shorter than the specified length, do not convert the character string. For example, convert **ab** to meet the specified length 4.

   Expression: StringUtils.center(value,*4*)

o. Delete a newline (including **\n**, **\r**, and **\r\n**) at the end of a character string. For example, convert **abc\r\n\r\n** to **abc\r\n**.

   Expression: StringUtils.chomp(value)

p. If the string contains the specified string, **true** is returned; otherwise, **false** is returned. For example, **abc** contains **a** so that **true** is returned.

   Expression: StringUtils.contains(value,"*a*")

q. If the string contains any character of the specified string, **true** is returned; otherwise, **false** is returned. For example, **zzabyycdxx** contains either **z** or **a** so that **true** is returned.

   Expression: StringUtils.containsAny(value,"*za*")

r. If the string does not contain any one of the specified characters, **true** is returned. If any specified character is contained, **false** is returned. For example, **abz** contains one character of **xyz** so that **false** is returned.

   Expression: StringUtils.containsNone(value,"*xyz*")

s. If the string contains only the specified characters, **true** is returned. If any other character is contained, **false** is returned. For example, **abab** contains only characters among **abc** so that **true** is returned.

   Expression: StringUtils.containsOnly(value,"*abc*")

t. If the character string is empty or null, convert it to the specified character string. Otherwise, do not convert the character string. For example, convert the empty character string to null.

Expression: StringUtils.defaultIfEmpty(value,*null*)

u. If the string ends with the specified suffix (case sensitive), **true** is returned; otherwise, **false** is returned. For example, if the suffix of **abcdef** is not null, **false** is returned.

Expression: StringUtils.endsWith(value,*null*)

v. If the string is the same as the specified string (case sensitive), **true** is returned; otherwise, **false** is returned. For example, after strings **abc** and **ABC** are compared, **false** is returned.

Expression: StringUtils.equals(value,"*ABC*")

w. Obtain the first index of the specified character string in a character string. If no index is found, **-1** is returned. For example, the first index of **ab** in **aabaabaa** is 1.

Expression: StringUtils.indexOf(value,"*ab*")

x. Obtain the last index of the specified character string in a character string. If no index is found, **-1** is returned. For example, the last index of **k** in **aFkyk** is 4.

Expression: StringUtils.lastIndexOf(value,"*k*")

y. Obtain the first index of the specified character string from the position specified in the character string. If no index is found, **-1** is returned. For example, the first index of **b** obtained after the index 3 of **aabaabaa** is 5.

Expression: StringUtils.indexOf(value,"*b*",*3*)

z. Obtain the first index of any specified character in a character string. If no index is found, **-1** is returned. For example, the first index of **z** or **a** in **zzabyycdxx.** is 0.

Expression: StringUtils.indexOfAny(value,"*za*")

aa. If the string contains any Unicode character, **true** is returned; otherwise, **false** is returned. For example, **ab2c** contains only non-Unicode characters so that **false** is returned.

Expression: StringUtils.isAlpha(value)

ab. If the string contains only Unicode characters and digits, **true** is returned; otherwise, **false** is returned. For example, **ab2c** contains only Unicode characters and digits, so that **true** is returned.

Expression: StringUtils.isAlphanumeric(value)

ac. If the string contains only Unicode characters, digits, and spaces, **true** is returned; otherwise, **false** is returned. For example, **ab2c** contains only Unicode characters and digits, so that **true** is returned.

Expression: StringUtils.isAlphanumericSpace(value)

ad. If the string contains only Unicode characters and spaces, **true** is returned; otherwise, **false** is returned. For example, **ab2c** contains Unicode characters and digits so that **false** is returned.

Expression: StringUtils.isAlphaSpace(value)

ae. If the string contains only printable ASCII characters, **true** is returned; otherwise, **false** is returned. For example, for **!ab-c~**, **true** is returned.

Expression: StringUtils.isAsciiPrintable(value)

af. If the string is empty or null, **true** is returned; otherwise, **false** is returned.

Expression: StringUtils.isEmpty(value)

ag. If the string contains only Unicode digits, **true** is returned; otherwise, **false** is returned.

Expression: StringUtils.isNumeric(value)

ah. Obtain the leftmost characters of the specified length. For example, obtain the leftmost two characters **ab** from **abc**.

Expression: StringUtils.left(value,*2*)

ai. Obtain the rightmost characters of the specified length. For example, obtain the rightmost two characters **bc** from **abc**.

Expression: StringUtils.right(value,*2*)

aj. Concatenate the specified character string to the left of the current character string and specify the length of the concatenated character string. If the length of the current character string is not shorter than the specified length, the character string will not be converted. For example, if **yz** is concatenated to the left of **bat** and the length must be 8 after concatenation, the character string is **yzyzybat** after conversion.

Expression: StringUtils.leftPad(value,*8*,"*yz*")

ak. Concatenate the specified character string to the right of the current character string and specify the length of the concatenated character string. If the length of the current character string is not shorter than the specified length, the character string will not be converted. For example, if **yz** is concatenated to the right of **bat** and the length must be 8 after concatenation, the character string is **batyzyzy** after conversion.

Expression: StringUtils.rightPad(value,*8*,"*yz*")

al. If the field is of the string type, obtain the length of the current character string. If the character string is null, **0** is returned.

Expression: StringUtils.length(value)

am. If the field is of the string type, delete all the specified character strings from it. For example, delete **ue** from **queued** to obtain **qd**.

Expression: StringUtils.remove(value,"*ue*")

an. If the field is of the string type, remove the substring at the end of the field. If the specified substring is not at the end of the field, no conversion is performed. For example, remove **.com** at the end of **www.domain.com**.

Expression: StringUtils.removeEnd(value,"*.com*")

ao. If the field is of the string type, delete the substring at the beginning of the field. If the specified substring is not at the beginning of the field, no conversion is performed. For example, delete **www.** at the beginning of **www.domain.com**.

Expression: StringUtils.removeStart(value,"*www.*")

ap. If the field is of the string type, replace all the specified character strings in the field. For example, replace **a** in **aba** with **z** to obtain **zbz**.

Expression: StringUtils.replace(value,"*a*","*z*")

If the content to be replaced contains a special character, the special character must be escaped to a common character. For example, if you want to delete **\t** from a string, use the following expression:

StringUtils.replace(value,"\\t",""), which means escaping the backslash (\) again.

aq. If the field is of the string type, replace multiple characters in the character string at a time. For example, replace **h** in **hello** with **j** and **o** with **y** to obtain **jelly**.

Expression: StringUtils.replaceChars(value,"*ho*","*jy*")

ar. If the string starts with the specified prefix (case sensitive), **true** is returned; otherwise, **false** is returned. For example, **abcdef** starts with **abc**, so that **true** is returned.

Expression: StringUtils.startsWith(value,"*abc*")

as. If the field is of the string type, delete all the specified characters at the beginning and end of the field. the field. For example, delete all **x**, **y**, **z**, and **b** from **abcyx** to obtain **abc**.

Expression: StringUtils.strip(value,"*xyz*b")

at. If the field is of the string type, delete all the specified characters at the end of the field, for example, delete the "abc" string at the end of the field.

Expression: StringUtils.stripEnd(value, *"abc"*)

au. If the field is of the string type, delete all the specified characters at the beginning of the field, for example, delete all spaces at the beginning of the field.

Expression: StringUtils.stripStart(value,*null*)

av. If the field is of the string type, obtain the substring after the specified position (the index starts from 0, including the character at the specified position) of the character string. If the specified position is a negative number, calculate the position in the descending order. The first digit at the end is -1. For example, obtain the character whose index is 2 from **abcde** (that is, **c**) and the string after it, that is, **cde**.

Expression: StringUtils.substring(value,*2*)

aw. If the field is of the string type, obtain the substring in a specified range (the index starts from 0, including the character at the start and excluding the character at the end). If the range is a negative number, calculate the position in the descending order. The first digit at the end is -1. For example, obtain the string between the second character (c) and fourth character (e) of **abcde**, that is, **cd**.

Expression: StringUtils.substring(value,*2*,4)

ax. If the field is of the string type, obtain the substring after the first specified character. For example, obtain the substring after the first **b** in **abcba**, that is, **cba**.

Expression: StringUtils.substringAfter(value,"*b*")

ay. If the field is of the string type, obtain the substring after the last specified character. For example, obtain the substring after the last **b** in **abcba**, that is, **a**.

Expression: StringUtils.substringAfterLast(value,"*b*")

az. If the field is of the string type, obtain the substring before the first specified character. For example, obtain the substring before the first **b** in **abcba**, that is, **a**.

Expression: StringUtils.substringBefore(value,"*b*")

ba. If the field is of the string type, obtain the substring before the last specified character. For example, obtain the substring before the last **b** in **abcba**, that is, **abc**.

Expression: StringUtils.substringBeforeLast(value,"*b*")

bb. If the field is of the string type, obtain the substring nested within the specified string. If no substring is found, **null** is returned. For example, obtain the substring between **tag** in **tagabctag**, that is, **abc**.

Expression: StringUtils.substringBetween(value,"*tag*")

bc. If the field is of the string type, delete the control characters (char≤32) at both ends of the character string, for example, delete the spaces at both ends of the character string.

Expression: StringUtils.trim(value)

bd. Convert the character string to a value of the byte type. If the conversion fails, **0** is returned.

Expression: NumberUtils.toByte(value)

be. Convert the character string to a value of the byte type. If the conversion fails, the specified value, for example, **1**, is returned.

Expression: NumberUtils.toByte(value,*1*)

bf. Convert the character string to a value of the double type. If the conversion fails, **0.0d** is returned.

Expression: NumberUtils.toDouble(value)

bg. Convert the character string to a value of the double type. If the conversion fails, the specified value, for example, **1.1d**, is returned.

Expression: NumberUtils.toDouble(value,*1.1d*)

bh. Convert the character string to a value of the float type. If the conversion fails, **0.0f** is returned.

Expression: NumberUtils.toFloat(value)

bi. Convert the character string to a value of the float type. If the conversion fails, the specified value, for example, **1.1f**, is returned.

Expression: NumberUtils.toFloat(value,*1.1f*)

bj. Convert the character string to a value of the int type. If the conversion fails, **0** is returned.

Expression: NumberUtils.toInt(value)

bk. Convert the character string to a value of the int type. If the conversion fails, the specified value, for example, **1**, is returned.

Expression: NumberUtils.toInt(value,*1*)

bl. Convert the character string to a value of the long type. If the conversion fails, **0** is returned.

Expression: NumberUtils.toLong(value)

bm. Convert the character string to a value of the long type. If the conversion fails, the specified value, for example, **1L**, is returned.

Expression: NumberUtils.toLong(value,*1L*)

bn. Convert the character string to a value of the short type. If the conversion fails, **0** is returned.

Expression: NumberUtils.toShort(value)

bo. Convert the character string to a value of the short type. If the conversion fails, the specified value, for example, **1**, is returned.

Expression: NumberUtils.toShort(value,*1*)

bp. Convert the IP string to a value of the long type, for example, convert **10.78.124.0** to **172915712**.

Expression: CommonUtils.ipToLong(value)

bq. Read an IP address and physical address mapping file from the network, and download the mapping file to the map collection. *url* indicates the address for storing the IP mapping file, for example, **http://10.114.205.45:21203/sqoop/IpList.csv**.

Expression: HttpsUtils.downloadMap("*url*")

br. Cache the IP address and physical address mappings and specify a key for retrieval, for example, **ipList**.

Expression: CommonUtils.setCache("*ipList*",HttpsUtils.downloadMap("*url*"))

bs. Obtain the cached IP address and physical address mappings.

Expression: CommonUtils.getCache("*ipList*")

bt. Check whether the IP address and physical address mappings are cached.

Expression: CommonUtils.cacheExists("*ipList*")

bu. Based on the specified offset type (month/day/hour/minute/second) and offset (positive number indicates increase and negative number indicates decrease), convert the time in the specified format to a new time, for example, add 8 hours to **2019-05-21 12:00:00**.

Expression: DateUtils.getCurrentTimeByZone("*yyyy-MM-dd HH:mm:ss*",value, "*hour*", *8*)

bv. If the value is empty or null, "aaa" is returned. Otherwise, **value** is returned.

Expression: StringUtils.defaultIfEmpty(value,*"aaa"*)

1.7 Migrating Files with Specified Names

You can migrate files (a maximum of 50) with specified names from FTP, OBS, or SFTP at a time. The exported files can only be written to the same directory on the migration destination.

When creating a table/file migration job, if the migration source is FTP, OBS, or SFTP, **Source Directory/File** can contain a maximum of 50 file names, which are separated by vertical bars (|). You can also customize a file separator.

NOTE

1. CDM supports incremental file migration (by skipping repeated files), but does not support resumable transfer.

   For example, if three files are to be migrated and the second file fails to be migrated due to the network fault. When the migration task is started again, the first file is skipped. The second file, however, cannot be migrated from the point where the fault occurs, but can only be migrated again.

2. During file migration, a single task supports millions of files. If there are too many files in the directory to be migrated, you are advised to split the files into different directories and create multiple tasks.

1.8 Regular Expressions for Separating Semi-structured Text

During table/file migration, CDM uses delimiters to separate fields in CSV files. However, delimiters cannot be used in complex semi-structured data because the field values also contain delimiters. In this case, the regular expression can be used to separate the fields.

The regular expression is configured in **Source Job Configuration**. The migration source must be an object storage or file system, and **File Format** must be **CSV**.

Figure 1-13 Setting regular expression parameters

During the migration of CSV files, CDM can use regular expressions to separate fields and write parsed results to the migration destination. For details about the syntax of the regular expression, refer to the related documents. This section describes the regular expressions of the following log files:

- **Log4J Log**
- **Log4J Audit Log**
- **Tomcat Log**
- **Django Log**
- **Apache Server Log**

Log4J Log

- Log sample:
  ```
  2018-01-11 08:50:59,001 INFO
  [org.apache.sqoop.core.SqoopConfiguration.configureClassLoader(SqoopConfiguration.java:251)]
  Adding jars to current classloader from property: org.apache.sqoop.classpath.extra
  ```
- Regular expression:
  ```
  ^(\d.*\d) (\w*)  \[(.*)\] (\w.*).*
  ```
- Parsing result:

Table 1-2 Log4J log parsing result

Column Number	Example Value
1	2018-01-11 08:50:59,001
2	INFO
3	org.apache.sqoop.core.SqoopConfiguration.configureClassLoader(SqoopConfiguration.java:251)
4	Adding jars to current classloader from property: org.apache.sqoop.classpath.extra

Log4J Audit Log

- Log sample:
  ```
  2018-01-11 08:51:06,156 INFO
  [org.apache.sqoop.audit.FileAuditLogger.logAuditEvent(FileAuditLogger.java:61)]
  user=sqoop.anonymous.user    ip=189.xxx.xxx.75    op=show    obj=version    objId=x
  ```
- Regular expression:
  ```
  ^(\d.*\d) (\w*)  \[(.*)\] user=(\w.*)    ip=(\w.*)    op=(\w.*)    obj=(\w.*)    objId=(.*).*
  ```
- Parsing result:

Table 1-3 Log4J audit log parsing result

Column Number	Example Value
1	2018-01-11 08:51:06,156
2	INFO
3	org.apache.sqoop.audit.FileAuditLogger.logAuditEvent(FileAuditLogger.java:61)
4	sqoop.anonymous.user
5	189.xxx.xxx.75

Column Number	Example Value
6	show
7	version
8	x

Tomcat Log

- Log sample:
  ```
  11-Jan-2018 09:00:06.907 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name:  Linux
  ```
- Regular expression:
  ```
  ^(\d.*\d) (\w*) \[(.*)\] ([\w\.]*) (\w.*).*
  ```
- Parsing result:

  Table 1-4 Tomcat log parsing result

Column Number	Example Value
1	11-Jan-2018 09:00:06.907
2	INFO
3	main
4	org.apache.catalina.startup.VersionLoggerListener.log
5	OS Name:Linux

Django Log

- Log sample:
  ```
  [08/Jan/2018 20:59:07 ] settings     INFO     Welcome to Hue 3.9.0
  ```
- Regular expression:
  ```
  ^\[(.*)\] (\w*)     (\w*)     (.*).*
  ```
- Parsing result:

Table 1-5 Django log parsing result

Column Number	Example Value
1	08/Jan/2018 20:59:07
2	settings
3	INFO
4	Welcome to Hue 3.9.0

Apache Server Log

- Log sample:
  ```
  [Mon Jan 08 20:43:51.854334 2018] [mpm_event:notice] [pid 36465:tid 140557517657856] AH00489: Apache/2.4.12 (Unix) OpenSSL/1.0.1t configured -- resuming normal operations
  ```
- Regular expression:
  ```
  ^\[(.*)\] \[(.*)\] \[(.*)\] (.*).*
  ```
- Parsing result:

Table 1-6 Apache server log parsing result

Column Number	Example Value
1	Mon Jan 08 20:43:51.854334 2018
2	mpm_event:notice
3	pid 36465:tid 140557517657856
4	AH00489: Apache/2.4.12 (Unix) OpenSSL/1.0.1t configured -- resuming normal operations

1.9 Recording the Time When Data Is Written to the Database

When you create a job on the CDM console to migrate tables or files of a relational database, you can add a field to record the time when they were written to the database.

Prerequisites

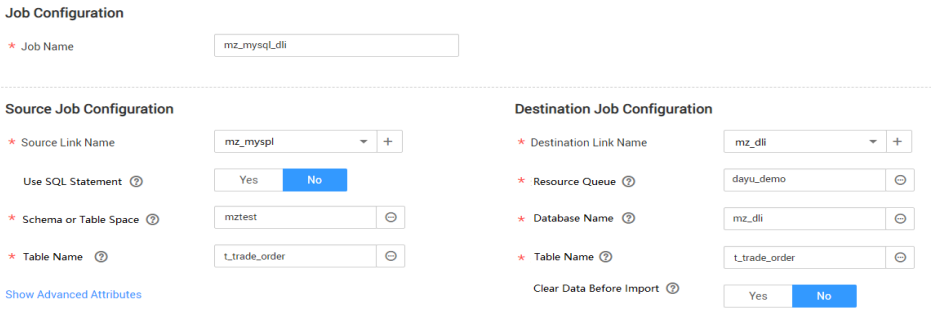
- A link has been created, and the source end of the connector is a relational database.

- The destination data table contains a date and time field or timestamp field. In the automatic table creation scenario, you need to manually create the date and time field or timestamp field in the destination table in advance.

Creating a Table/File Migration Job

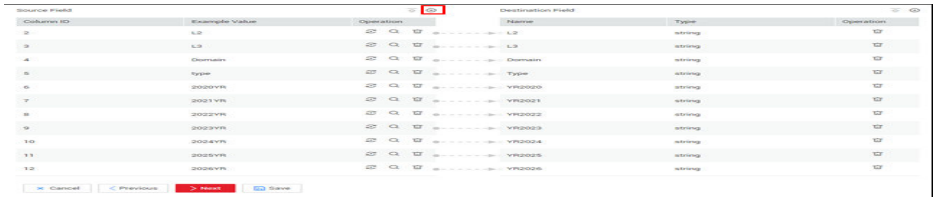
Step 1 Create a table/file migration job, and select the created source connector and destination connector.

Figure 1-14 Configuring the job



Step 2 Click **Next** to go to the **Map Field** page and click .

Figure 1-15 Configuring field mapping



Step 3 Click the **Custom Fields** tab, set the field name and value, and click **OK**.

Name: Enter **InputTime**.

Value: Enter **${timestamp()}**. For more time macro variables, see **Table 1-7**.

Figure 1-16 Add Field

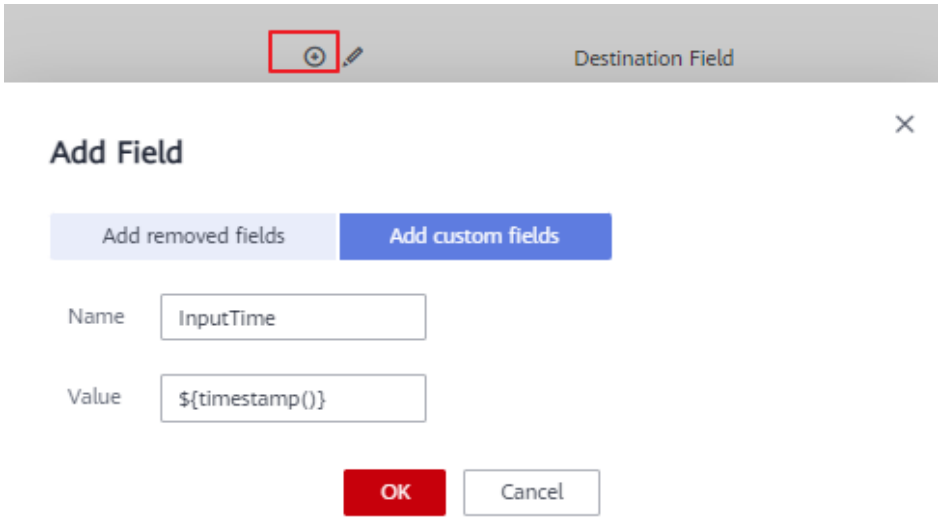


Table 1-7 Macro variable definition of time and date

Macro Variable	Description	Display Effect
${dateformat(yyyy-MM-dd)}	Returns the current date in **yyyy-MM-dd** format.	2017-10-16
${dateformat(yyyy/MM/dd)}	Returns the current date in **yyyy/MM/dd** format.	2017/10/16
${dateformat(yyyy_MM_dd HH:mm:ss)}	Returns the current time in **yyyy_MM_dd HH:mm:ss** format.	2017_10_16 09:00:00
${dateformat(yyyy-MM-dd HH:mm:ss, -1, DAY)}	Returns the current time in **yyyy-MM-dd HH:mm:ss** format. The date is one day before the current day.	2017-10-15 09:00:00
${dateformat(yyyy-MM-dd, -1, DAY)} 00:00:00	Returns 00:00:00 of the day before the current day in *yyyy-MM-dd HH:mm:ss* format.	2017-10-15 00:00:00
${dateformat(yyyy-MM-dd, -1, DAY)} 12:00:00	Returns 12:00:00 of the day before the current day in *yyyy-MM-dd HH:mm:ss* format.	2017-10-15 12:00:00
${dateformat(yyyy-MM-dd, -N, DAY)} 00:00:00	Returns 00:00:00 of the day N days before the current day in *yyyy-MM-dd HH:mm:ss* format.	When N is 3: 2017-10-13 00:00:00
${dateformat(yyyy-MM-dd, -N, DAY)} 12:00:00	Returns 12:00:00 of the day N days before the current day in *yyyy-MM-dd HH:mm:ss* format.	When N is 3: 2017-10-13 12:00:00
${timestamp()}	Returns the timestamp of the current time, that is, the number of milliseconds that have elapsed since 00:00:00 on January 1, 1970.	1508115600000
${timestamp(-10, MINUTE)}	Returns the timestamp generated 10 minutes before the current time point.	1508115000000
${timestamp(dateformat(yyyyMMdd))}	Returns the timestamp of 00:00:00 of the current day.	1508083200000
${timestamp(dateformat(yyyyMMdd,-1,DAY))}	Returns the timestamp of 00:00:00 of the previous day.	1507996800000

Macro Variable	Description	Display Effect
${timestamp(dateformat(yyyyMMddHH))}	Returns the timestamp of the current hour.	1508115600000

NOTE

- After a field is added, its sample value is not displayed on the console. This does not affect the field value transmission. CDM directly writes the field value to the destination end.
- The **Custom Fields** tab is available only when the source connector is JDBC, HBase, MongoDB, Elasticsearch, or Kafka, or the destination connector is HBase.
- After adding the fields, ensure that the customized import time field matches the field type of the destination table.

Step 4 Click **Next** and set task parameters. Generally, retain the default values of all parameters.

Step 5 Click **Save and Run**. On the **Table/File Migration** page, you can view the job execution progress and result.

Step 6 After the job is successfully executed, in the **Operation** column of the job, click **Historical Record** to view the job's historical execution records and read/write statistics.

On the **Historical Record** page, click **Log** to view the job logs.

Step 7 Go to the destination data source to check the time when the data is imported to the database.

----End

1.10 File Formats

When creating a CDM job, you need to specify **File Format** in the job parameters of the migration source and destination in some scenarios. This section describes the application scenarios, subparameters, common parameters, and usage examples of the supported file formats.

- **CSV**
- **JSON**
- **Binary**
- **Common parameters**
- **Solutions to File Format Problems**

CSV

To read or write a CSV file, set **File Format** to **CSV**. The CSV format can be used in the following scenarios:

- Import files to a database or NoSQL.
- Export data from a database or NoSQL to files.

After selecting the CSV format, you can also configure the following optional sub-parameters:

1. Line Separator

2. Field Delimiter

3. Encoding Type

4. Use Quote Character

5. Use RE to Separate Fields

6. Use First Row as Header

7. File Size

1. **Line Separator**

   Character used to separate lines in a CSV file. The value can be a single character, multiple characters, or special characters. Special characters can be entered using the URL encoded characters. The following table lists the URL encoded characters of commonly used special characters.

   Table 1-8 URL encoded characters of special characters

Special Character	URL Encoded Character
Space	%20
Tab	%09
%	%25
Enter	%0d
Newline character	%0a
Start of heading\u0001 (SOH)	%01

2. **Field Delimiter**

   Character used to separate columns in a CSV file. The value can be a single character, multiple characters, or special characters. For details, see **Table 1-8**.

3. **Encoding Type**

   Encoding type of a CSV file. The default value is **UTF-8**.

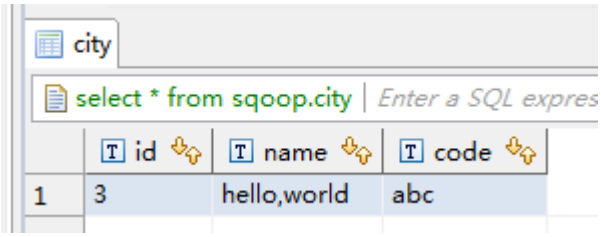
   If this parameter is specified at the migration source, the specified encoding type is used to parse the file. If this parameter is specified at the migration destination, the specified encoding type is used to write data to the file.

4. **Use Quote Character**

   - Exporting data from a database or NoSQL to CSV files (configuring **Use Quote Character** at the migration destination): If a field delimiter appears in the character string of a column of data at the migration source, set **Use Quote Character** to **Yes** at the migration destination to

quote the character string as a whole and write it into the CSV file. Currently, CDM uses double quotation marks ("") as the quote character only. **Figure 1-17** shows that the value of the **name** field in the database contains a comma (,).

Figure 1-17 Field value containing the field delimiter



If you do not use the quote character, the exported CSV file is displayed as follows:

```
3,hello,world,abc
```

If you use the quote character, the exported CSV file is displayed as follows:

```
3,"hello,world",abc
```

If the data in the database contains double quotation marks ("") and you set **Use Quote Character** to **Yes**, the quote character in the exported CSV file is displayed as three double quotation marks (""""). For example, if the value of a field is **a"hello,world"c**, the exported data is as follows:

```
"""a"hello,world"c"""
```

- Exporting CSV files to a database or NoSQL (configuring **Use Quote Character** at the migration source): If you want to import the CSV files with quoted values to a database correctly, set **Use Quote Character** to **Yes** at the migration source to write the quoted values as a whole.

5. **Use RE to Separate Fields**

   This function is used to parse complex semi-structured text, such as log files. For details, see **Using Regular Expressions to Separate Semi-structured Text**.

6. **Use First Row as Header**

   This parameter is used when CSV files are exported to other locations. If this parameter is specified at the migration source, CDM uses the first row as the header when extracting data. When the CSV files are transferred, the headers are skipped. The number of rows extracted from the migration source is more than the number of rows written to the migration destination. The log files will output the information that the header is skipped during the migration.

7. **File Size**

   This parameter is used when data is exported from the database to a CSV file. If a table contains a large amount of data, a large CSV file is generated after migration, which is inconvenient to download or view. In this case, you can specify this parameter at the migration destination so that multiple CSV files with the specified size can be generated. The value of this parameter is an integer. The unit is MB.

JSON

The following describes information about the JSON format:

- **JSON Types Supported by CDM**
- **JSON Reference Node**
- **Copying Data from a JSON File**

1. **JSON types supported by CDM: JSON object and JSON array**
   - JSON object: A JSON file contains a single object or multiple objects separated/merged by rows.
     i. The following is a single JSON object:

     ```
     {
         "took" : 190,
         "timed_out" : false,
         "total" : 1000001,
         "max_score" : 1.0
      }
     ```

     ii. The following are JSON objects separated by rows:

     ```
     {"took" : 188, "timed_out" : false, "total" : 1000003, "max_score" : 1.0 }
     {"took" : 189, "timed_out" : false, "total" : 1000004, "max_score" : 1.0 }
     ```

     iii. The following are merged JSON objects:

     ```
     {
        "took": 190,
        "timed_out": false,
        "total": 1000001,
        "max_score": 1.0
      }
      {
        "took": 191,
        "timed_out": false,
        "total": 1000002,
        "max_score": 1.0
      }
     ```

   - JSON array: A JSON file is a JSON array consisting of multiple JSON objects.

     ```
     [{
         "took" : 190,
         "timed_out" : false,
         "total" : 1000001,
         "max_score" : 1.0
      },
      {
         "took" : 191,
         "timed_out" : false,
         "total" : 1000001,
         "max_score" : 1.0
      }]
     ```

2. **JSON Reference Node**

   Root node that records data. The data corresponding to the node is a JSON array. CDM extracts data from the array in the same mode. Use periods (.) to separate multi-layer nested JSON nodes.

3. **Copying Data from a JSON File**

   a. Example 1

   Extract data from multiple objects that are separated or merged. A JSON file contains multiple JSON objects. The following gives an example:

   ```
   {
     "took": 190,
     "timed_out": false,
   ```



```
  "total": 1000001,
  "max_score": 1.0
}
{
  "took": 191,
  "timed_out": false,
  "total": 1000002,
  "max_score": 1.0
}
{
  "took": 192,
  "timed_out": false,
  "total": 1000003,
  "max_score": 1.0
}
```

To extract data from the JSON object and write data to the database in the following formats, set **File Format** to **JSON** and **JSON Type** to **JSON object**, and then map fields.

Table 1-9 Example

took	timedOut	total	maxScore
190	false	1000001	1.0
191	false	1000002	1.0
192	false	1000003	1.0

b. Example 2

Extract data from the reference node. A JSON file contains a single JSON object, but the valid data is on a data node. The following gives an example:

```
{
  "took": 190,
  "timed_out": false,
  "hits": {
     "total": 1000001,
     "max_score": 1.0,
     "hits":
      [{
        "_id": "650612",
        "_source": {
           "name": "tom",
           "books": ["book1","book2","book3"]
        }
      },
      {
         "_id": "650616",
         "_source": {
            "name": "tom",
            "books": ["book1","book2","book3"]
         }
      },
    {
         "_id": "650618",
         "_source": {
            "name": "tom",
            "books": ["book1","book2","book3"]
         }
      }]
  }
}
```


To write data to the database in the following formats, set **File Format** to **JSON**, **JSON Type** to **JSON object**, and **JSON Reference Node** to **hits.hits**, and then map fields.

Table 1-10 Example

ID	SourceName	SourceBooks
650612	tom	["book1","book2","book3"]
650616	tom	["book1","book2","book3"]
650618	tom	["book1","book2","book3"]

c. Example 3

Extract data from the JSON array. A JSON file is a JSON array consisting of multiple JSON objects. The following gives an example:

```
[{
    "took" : 190,
    "timed_out" : false,
    "total" : 1000001,
    "max_score" : 1.0
 },
 {
    "took" : 191,
    "timed_out" : false,
    "total" : 1000002,
    "max_score" : 1.0
 }]
```

To write data to the database in the following formats, set **File Format** to **JSON** and **JSON Type** to **JSON array**, and then map fields.

Table 1-11 Example

took	timedOut	total	maxScore
190	false	1000001	1.0
191	false	1000002	1.0

d. Example 4

Configure a converter when parsing the JSON file. On the premise of **example 2**, to add the **hits.max_score** field to all records, that is, to write the data to the database in the following formats, perform the following operations:

Table 1-12 Example

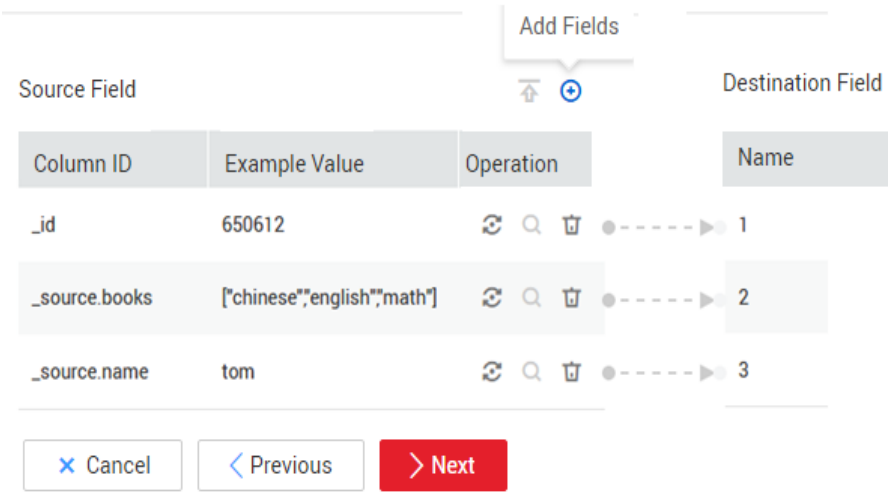
ID	SourceName	SourceBooks	MaxScore
650612	tom	["book1","book2","book3"]	1.0
650616	tom	["book1","book2","book3"]	1.0

ID	SourceName	SourceBooks	MaxScore
650618	tom	["book1","book2","book3"]	1.0

Set **File Format** to **JSON**, **JSON Type** to **JSON object**, and **JSON Reference Node** to **hits.hits**, and then create a converter.

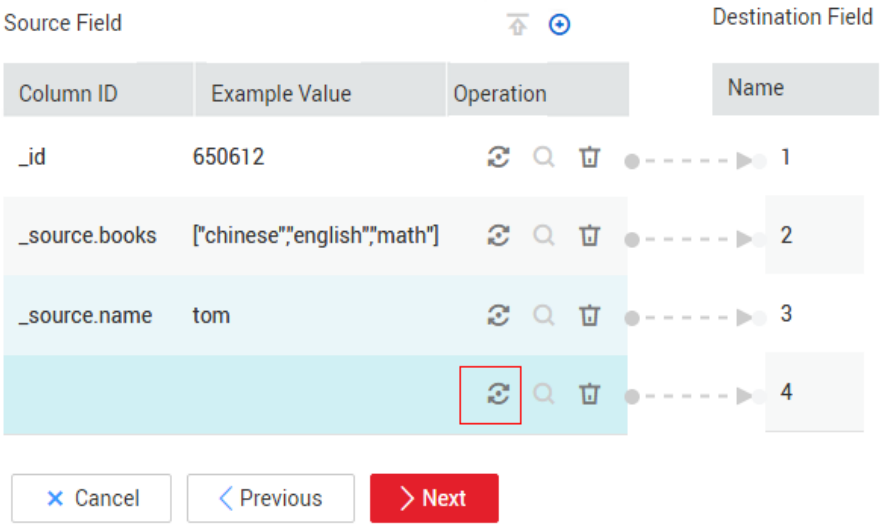
i. Click to add a field.

Figure 1-18 Adding a field



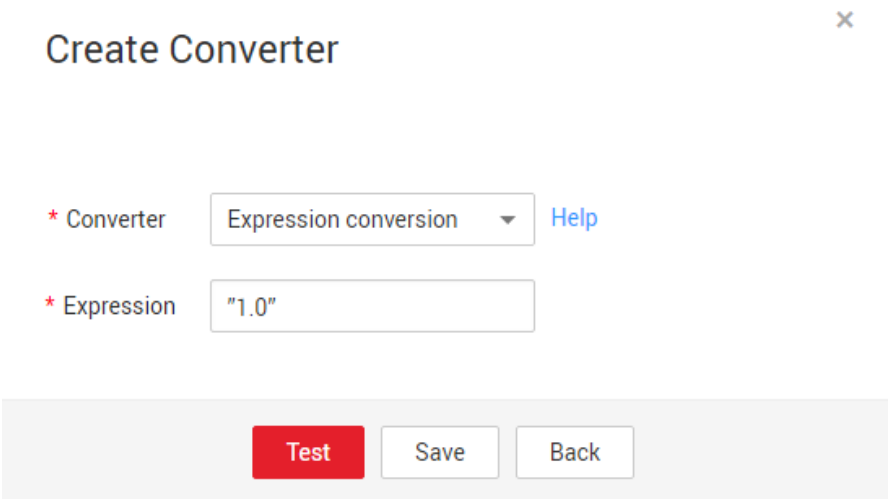
ii. Click to create a converter for the new field.

Figure 1-19 Creating a field converter



iii. Set **Converter** to **Expression conversion**, enter **"1.0"** in the **Expression** text box, and click **Save**.

Figure 1-20 Configuring a field converter



Binary

If you want to copy files between file systems, you can select the binary format. Files can be transferred in binary format at a high speed and stable performance. In addition, field mapping is not required in the second step of the job.

- **Directory structure for file transfer**

  CDM can transfer a single file or all files in a directory at a time. After the files are transferred to the migration destination, the directory structure remains unchanged.

- **Migrating incremental files**

  When you use CDM to transfer files in binary format, configure **Duplicate File Processing Method** at the migration destination for incremental file migration. For details, see **Incremental File Migration**.

  During incremental file migration, set **Duplicate File Processing Method** to **Skip**. If new files exist at the migration source or a failure occurs during the migration, run the job again, so that the migrated files will not be migrated repeatedly.

- **Write to Temporary File**

  When migrating files in binary format, you can specify whether to write the files to a temporary file at the migration destination. If this parameter is specified, the file is written to a temporary file during file replication. After the file is successfully migrated, run the **rename** or **move** command to restore the file at the migration destination.

- **Generate MD5 Hash Value**

  An MD5 hash value is generated for each transferred file, and the value is recorded in a new **.md5** file. You can specify the directory where the MD5 value is generated.

Common parameters

- **Start Job by Marker File**

  In automation scenarios, a scheduled task is configured on CDM to periodically read files from the migration source. However, files are being

generated at the migration source. As a result, CDM reads data repeatedly or fails to read data from the migration source. You can specify the marker file for starting a job as **ok.txt** in the job parameters of the migration source. After the file is successfully generated at the migration source, the **ok.txt** file is generated in the file directory. In this way, CDM can read the complete file.

In addition, you can set the suspension period. Within the suspension period, CDM periodically queries whether the marker file exists. If the file does not exist after the suspension period expires, the job fails.

The marker file will not be migrated.

- **Job Success Marker File**

  After data is successfully migrated to a file system, an empty file is generated in the destination directory. You can specify the file name. Generally, this parameter is used together with **Start Job by Marker File**.

  The name of the job success marker file cannot be the same as that of the transferred file, for example, finish.txt. If the two files have the same name, they will overwrite each other.

- **Filter**

  When using CDM to migrate files, you can specify a filter to filter files. Files can be filtered by wildcard character or time filter.

  – If you select **Wildcard**, CDM migrates only the paths or files that meet the filter condition.

  – If you select **Time Filter**, CDM migrates only the files modified after the specified time point.

  For example, the **/table/** directory stores a large number of data table directories divided by day. **DRIVING_BEHAVIOR_20180101** to **DRIVING_BEHAVIOR_20180630** store all data of **DRIVING_BEHAVIOR** from January to June. If you only want to migrate the table data of **DRIVING_BEHAVIOR** in March, set the source directory to **/table**, filter type to wildcard, and path filter to **DRIVING_BEHAVIOR_201803***.

Solutions to File Format Problems

1. When data in a database is exported to a CSV file, if the data contains commas (,), the data in the exported CSV file is disordered.

   The following solutions are available:

   – Specify a field delimiter.

     Use a character that does not exist in the database or a rare non-printable character as the field delimiter. For example, you can set **Field Delimiter** at the destination to **%01**. In this way, the exported field delimiter is **\u0001**. For details, see **Table 1-8**.

   – Use a quote character.

     Set **Use Quote Character** to **Yes** at the migration destination. In this way, if the field in the database contains the field delimiter, CDM quotes the field using the quote character and write the field as a whole to the CSV file.

2. The data in the database contains line separators.

   – Scenario: When you use CDM to export a table in the MySQL database (a field value contains the line separator **\n**) to a CSV file, and then use

CDM to import the exported CSV file to MRS HBase, data in the exported CSV file is truncated.

- Solution: Specify a line separator.

  When you use CDM to export MySQL table data to a CSV file, set **Line Separator** at the migration destination to **%01** (ensure that the value does not appear in the field value). In this way, the line separator in the exported CSV file is **%01**. Then use CDM to import the CSV file to MRS HBase. Set **Line Separator** at the migration source to **%01**. This avoids data truncation.

2 Scheduling a CDM Job by Transferring Parameters Using DataArts Factory

You can use EL expressions in DataArts Factory to transfer parameters to a CDM job to schedule it.

NOTE

- The parameter transfer function is supported by CDM 2.8.6 or later versions.
- This section uses a CDM job for migrating data from Oracle to MRS Hive as an example.

Prerequisites

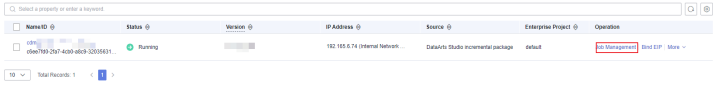
A CDM incremental package is available.

Creating a CDM Migration Job

Step 1 Log in to the console, locate an instance, click **Access**, and click **DataArts Migration**.

Step 2 On the **Cluster Management** page, click **Job Management** in the **Operation** column.

Figure 2-1 Cluster Management

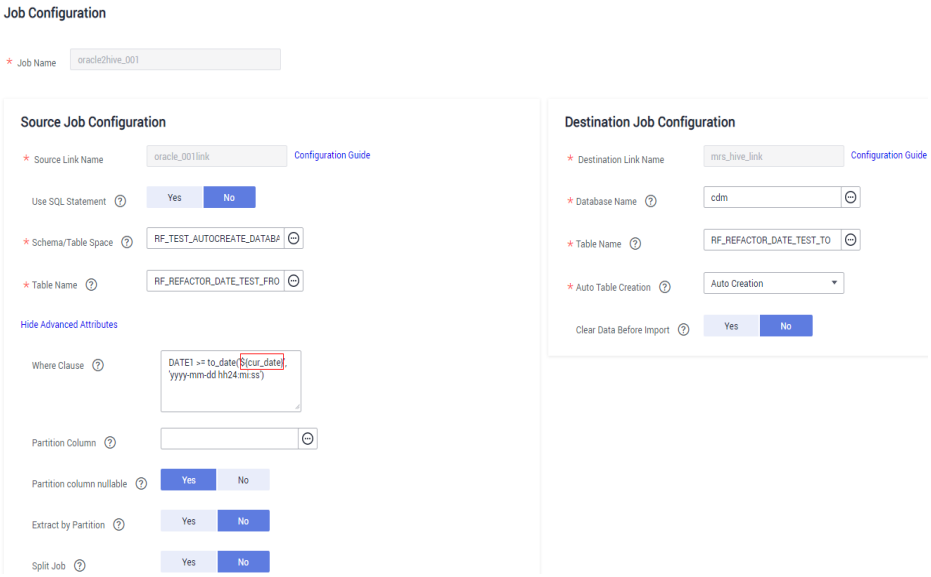


Step 3 Click the **Links** tab and then **Create Link** to create an Oracle link and an MRS Hive link. For details, see **Link to an Oracle Database** and **Link to Hive**.

Step 4 Click the **Table/File Migration** tab and then **Create Job** to create a data migration job.

Step 5 Configure parameters for the source Oracle link and destination MRS Hive link, and configure the parameter to transfer in **${***varName***}** format (**${cur_date}** in this example).

Figure 2-2 Creating a job



NOTE

The **Retry upon Failure** parameter is unavailable in the CDM migration job. You can configure this parameter on the CDM node in DataArts Factory.

----End

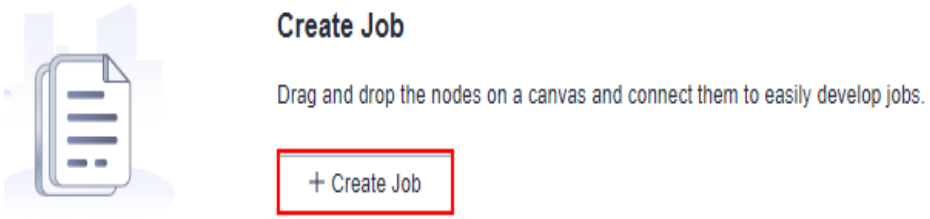
Creating and Executing a Data Development Job

Step 1 On the DataArts Studio console, locate a workspace and click **DataArts Factory**.

Step 2 In the navigation pane of the DataArts Factory homepage, choose **Data Development** > **Develop Job**.

Step 3 On the **Develop Job** page, click **Create Job**.

Figure 2-3 Create Job



Step 4 In the displayed dialog box, configure job parameters and click **OK**.

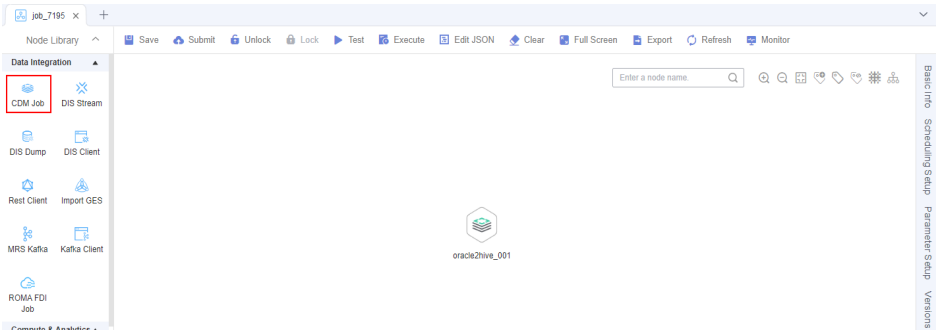
Table 2-1 Job parameters

Parameter	Description
Job Name	Name of the job. The name must contain 1 to 128 characters, including only letters, numbers, hyphens (-), underscores (_), and periods (.).
Job Type	Type of the job. • **Batch processing**: Data is processed periodically in batches based on the scheduling plan, which is used in scenarios with low real-time requirements. This type of job is a pipeline that consists of one or more nodes and is scheduled as a whole. It cannot run for an unlimited period of time, that is, it must end after running for a certain period of time. You can configure job-level scheduling tasks for batch processing jobs. For details, see **Setting Up Scheduling for a Job Using the Batch Processing Mode**. • **Real-time processing**: Data is processed in real time, which is used in scenarios with high real-time performance. This type of job is a business relationship that consists of one or more nodes. You can configure a scheduling policy for each node, and the tasks started by nodes can keep running for an unlimited period of time. In this type of job, lines with arrows represent only service relationships, rather than task execution processes or data flows. You can configure node-level scheduling tasks for real-time processing jobs. For details, see **Setting Up Scheduling for Nodes of a Job Using the Real-Time Processing Mode**.
Creation Method	Job creation method • **Create Empty Job**: Create an empty job. • **Create Based on Template**: Use a template provided by DataArts Factory to create a job.
Select Directory	Directory to which the job belongs. The default value is the root directory.
Owner	Owner of the job
Priority	Priority of the job. The options are **High**, **Medium**, and **Low**.
Agency	After an agency is configured, the job interacts with other services as an agency during job execution. **NOTE** A job-level agency takes precedence over a workspace-level agency.

Parameter	Description
Log Path	Path of the OBS bucket for storing job logs. By default, logs are stored in an OBS bucket named **dlf-log-**{*Projectid*}. **NOTE** • If you want to customize a storage path, select the bucket that you have created on OBS by following the instructions provided in **(Optional) Changing a Job Log Storage Path**. • Ensure that you have the read and write permissions on the OBS bucket specified by this parameter, or the system cannot write or display logs.

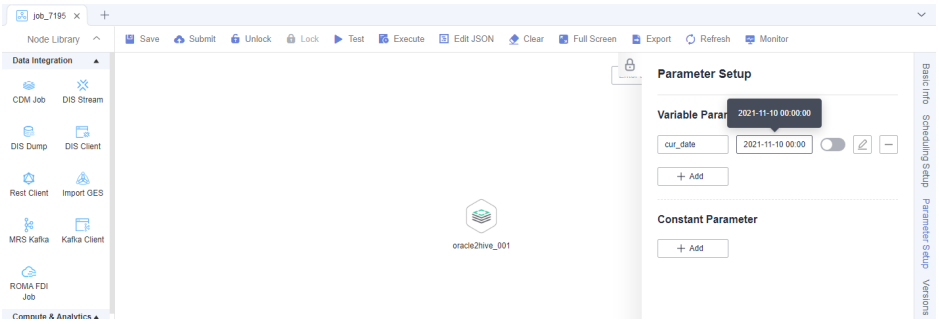
Step 5 Add a CDM Job node in the data development job and associate the node with the created CDM job.

Figure 2-4 Associating the CDM Job node with the created CDM job



Step 6 Configure the parameter to be transferred to the CDM job.

Figure 2-5 Configuring the parameter to be transferred



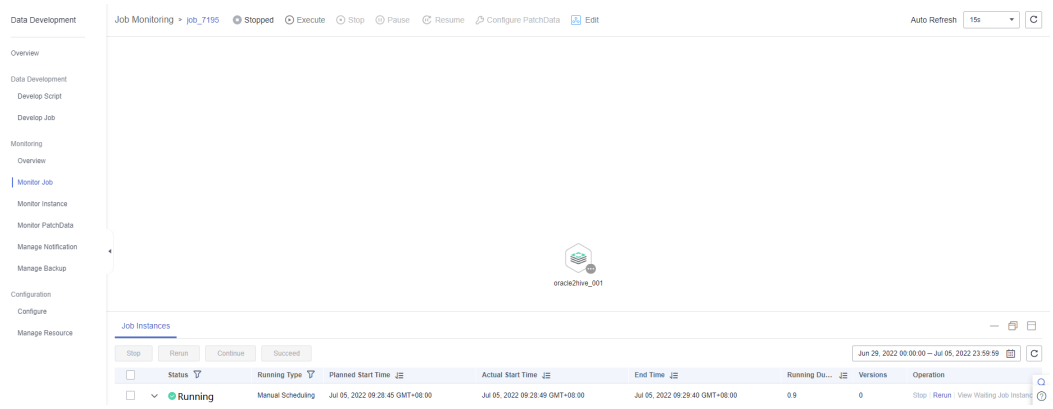
NOTE

When the job is scheduled and executed, the value of the configured parameter will be transferred to the CDM job. The value of the parameter **cur_date** can be set to a fixed value (for example, **2021-11-10 00:00:00**) or an EL expression (for example, **#{DateUtil.format(DateUtil.addDays(Job.planTime,-1),"yyyy-MM-dd")}** which means the day before the scheduled job execution date. For more EL expressions, see **EL expressions**.

Step 7 Save and submit a job version and click **Test** to execute the data development job.

Step 8 After the data development job is executed, click **Monitor** in the upper right corner to go to the **Monitor Job** page and check whether the generated task or instance meets requirements.

Figure 2-6 Viewing the execution result



----End

3 Enabling Incremental Data Migration Through DataArts Factory

The DataArts Factory module of DataArts Studio is a one-stop, collaborative big data development platform. You can enable incremental data migration through online script editing in DataArts Factory and periodic scheduling of CDM jobs.

This section describes how to use DataArts Factory together with CDM to migrate incremental data from GaussDB(DWS) to OBS.

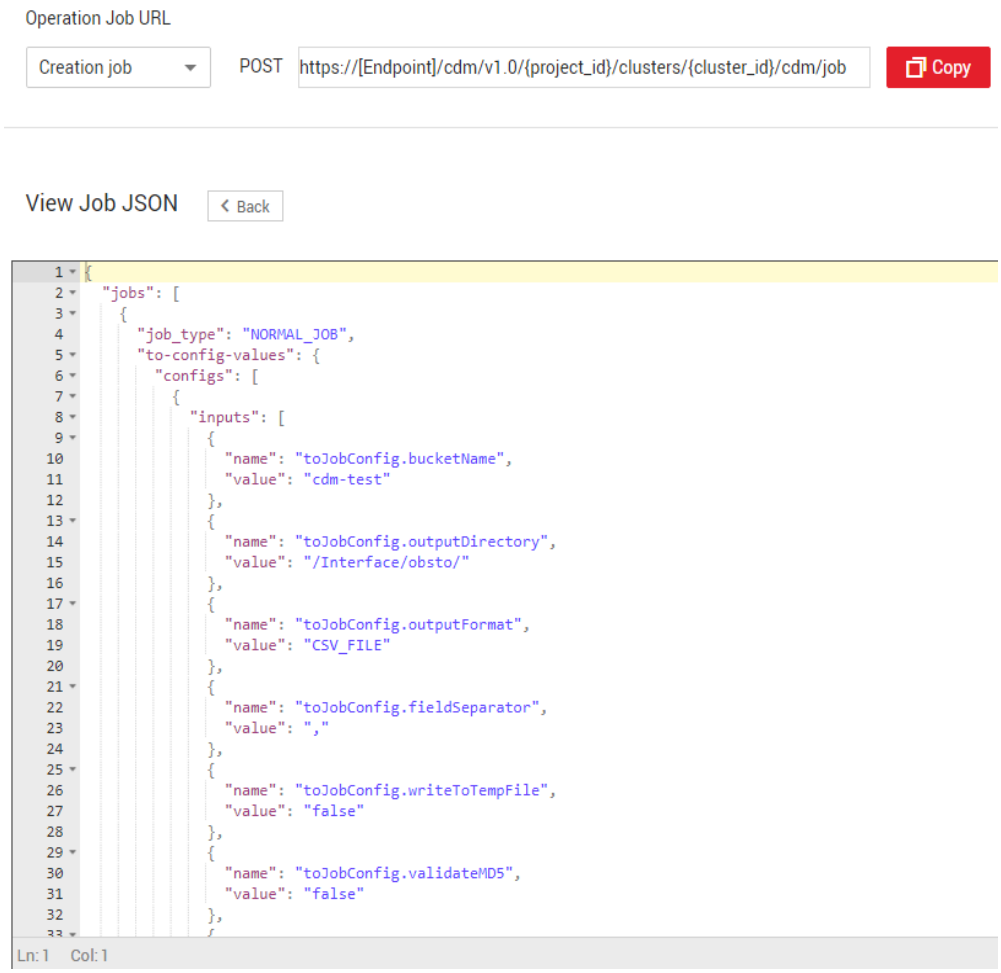
1. **Obtaining the CDM Job JSON**
2. **Modifying JSON**
3. **Creating a Job in DataArts Factory**

Obtaining the CDM Job JSON

1. On the CDM console, create a table/file migration job from GaussDB(DWS) to OBS.
2. On the **Table/File Migration** tab page of the **Job Management** page, locate the created job, click **More** in the **Operation** column, and select **View Job JSON** from the drop-down list.

   You can also view JSON of any other CDM job.

Figure 3-1 Viewing job JSON



3. The job JSON is the request body template for creating a CDM job. Replace **[Endpoint]**, **{project_id}**, and **{cluster_id}** in the URL with the actual values.
   - [Endpoint]: indicates the endpoint.

     An endpoint is the **request address** for calling an API. Endpoints vary depending on services and regions. You can obtain the endpoints of the service from **Endpoints**.
   - **{project_id}**: indicates the project ID.
   - **{cluster_id}**: Indicates the cluster ID. You can click the cluster name on the **Cluster Management** page to view the cluster ID.

Modifying JSON

You can modify the JSON body as required. In this example, the period is one day, and the WHERE clause is used for filtering the incremental data to be migrated (generally, the time range is used for filtering data). The data generated on the previous day is migrated every day.

1. Modify the WHERE clause to add incremental data in a certain period.

```
{
  "name": "fromJobConfig.whereClause",
  "value": "_timestamp >= '${startTime}' and _timestamp < '${currentTime}'"
}
```


NOTE

- If the source database is DWS or MySQL, the value can be set to:
  ```
  _timestamp >= '2018-10-10 00:00:00' and _timestamp < '2018-10-11 00:00:00'
  Or
  _timestamp between '2018-10-10 00:00:00' and '2018-10-11 00:00:00'
  ```
- If the source database is Oracle, the value should be set to:
  ```
  _timestamp >= to_date (2018-10-10 00:00:00' , 'yyyy-mm-dd hh24:mi:ss' ) and _timestamp < to_date (2018-10-10 00:00:00' , 'yyyy-mm-dd hh24:mi:ss' )
  ```

2. Import incremental data in each period to different directories.

   ```
   {
     "name": "toJobConfig.outputDirectory",
     "value": "dws2obs/${currentTime}"
   }
   ```

3. Change the job name to a dynamic one. Otherwise, the job cannot be created because the job name is duplicate.

   ```
   "to-connector-name": "obs-connector",
   "from-link-name": "dws_link",
   "name": "dws2obs-${currentTime}"
   ```

For details about how to modify more parameters, see ***Cloud Data Migration API Reference***. The following is an example of the modified JSON file:

```
{
  "jobs": [
    {
      "job_type": "NORMAL_JOB",
      "to-config-values": {
        "configs": [
          {
            "inputs": [
              {
                "name": "toJobConfig.bucketName",
                "value": "cdm-test"
              },
              {
                "name": "toJobConfig.outputDirectory",
                "value": "dws2obs/${currentTime}"
              },
              {
                "name": "toJobConfig.outputFormat",
                "value": "CSV_FILE"
              },
              {
                "name": "toJobConfig.fieldSeparator",
                "value": ","
              },
              {
                "name": "toJobConfig.writeToTempFile",
                "value": "false"
              },
              {
                "name": "toJobConfig.validateMD5",
                "value": "false"
              },
              {
                "name": "toJobConfig.encodeType",
                "value": "UTF-8"
              },
              {
                "name": "toJobConfig.duplicateFileOpType",
                "value": "REPLACE"
              },
              {
                "name": "toJobConfig.kmsEncryption",
                "value": "false"
              }
```



```
        ],
        "name": "toJobConfig"
      }
    ]
  },
  "from-config-values": {
    "configs": [
      {
        "inputs": [
          {
            "name": "fromJobConfig.schemaName",
            "value": "dws_database"
          },
          {
            "name": "fromJobConfig.tableName",
            "value": "dws_from"
          },
          {
            "name": "fromJobConfig.whereClause",
            "value": "_timestamp >= '${startTime}' and _timestamp < '${currentTime}'"
          },
          {
            "name": "fromJobConfig.columnList",
            "value":
"_tiny&_small&_int&_integer&_bigint&_float&_double&_date&_timestamp&_char&_varchar&_text"
          }
        ],
        "name": "fromJobConfig"
      }
    ]
  },
  "from-connector-name": "generic-jdbc-connector",
  "to-link-name": "obs_link",
  "driver-config-values": {
    "configs": [
      {
        "inputs": [
          {
            "name": "throttlingConfig.numExtractors",
            "value": "1"
          },
          {
            "name": "throttlingConfig.submitToCluster",
            "value": "false"
          },
          {
            "name": "throttlingConfig.numLoaders",
            "value": "1"
          },
          {
            "name": "throttlingConfig.recordDirtyData",
            "value": "false"
          },
          {
            "name": "throttlingConfig.writeToLink",
            "value": "obs_link"
          }
        ],
        "name": "throttlingConfig"
      },
      {
        "inputs": [],
        "name": "jarConfig"
      },
      {
        "inputs": [],
        "name": "schedulerConfig"
      },
      {
```



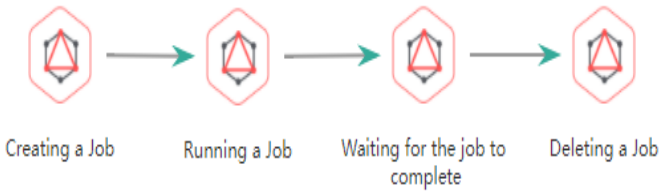
```
        "inputs": [],
        "name": "transformConfig"
      },
      {
        "inputs": [],
        "name": "smnConfig"
      },
      {
        "inputs": [],
        "name": "retryJobConfig"
      }
    ]
  },
  "to-connector-name": "obs-connector",
  "from-link-name": "dws_link",
  "name": "dws2obs-${currentTime}"
 }
]
}
```

Creating a Job in DataArts Factory

1. On the DataArts Factory console, create a data development job with Rest Client nodes shown in **Figure 3-2**. For details, see **Creating a Job** in *DataArts Studio User Guide*.

   For details about how to configure the nodes and the job, see the following steps.

   Figure 3-2 DataArts Factory job

2. Configure the **CreatingJob** node.

   DataArts Factory uses a Rest Client node to call a RESTful API to create a CDM migration job. Configure the properties of the Rest Client node.

   a. **Node Name**: Enter a custom name, for example, **CreatingJob**. Note that the CDM job is only used as a node in the DataArts Factory job.

   b. **URL Address**: Set it to the URL obtained in **Obtaining the CDM Job JSON**. The format is https://{*Endpoint*}/cdm/v1.0/{*project_id*}/clusters/{*cluster_id*}/cdm/job.

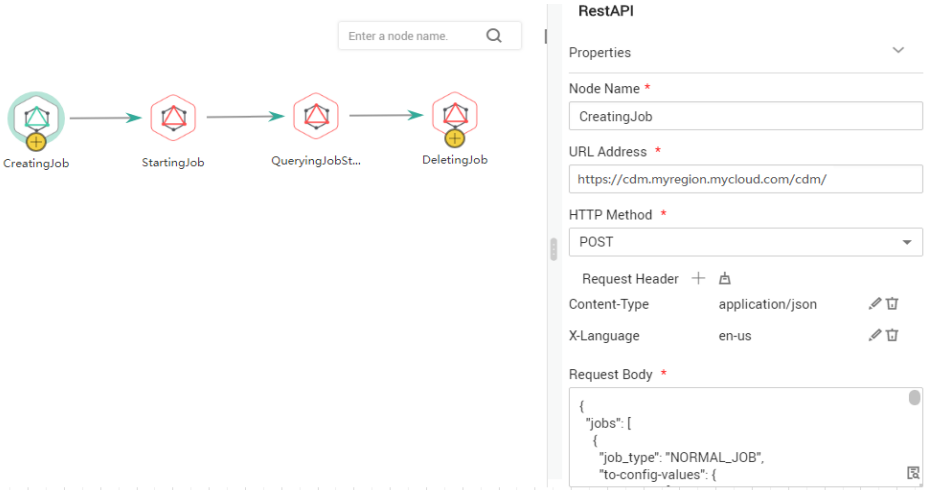
   c. **HTTP Method**: Enter **POST**.

   d. Add the following request headers:

      - Content-Type = application/json
      - X-Language = en-us

   e. **Request Body**: Enter the modified JSON of the CDM job in **Modifying JSON**.

Figure 3-3 Properties of the node for creating the CDM job



3. Configure the **StartingJob** node.

   After configuring the RESTful API node for creating a CDM job, you must add the RESTful API node for running the CDM job. For details, see section "Starting a Job" in ***Cloud Data Migration API Reference***. Configure the properties of the RestAPI node.

   a. **Node Name**: Enter the name of the node where the job is to be run.

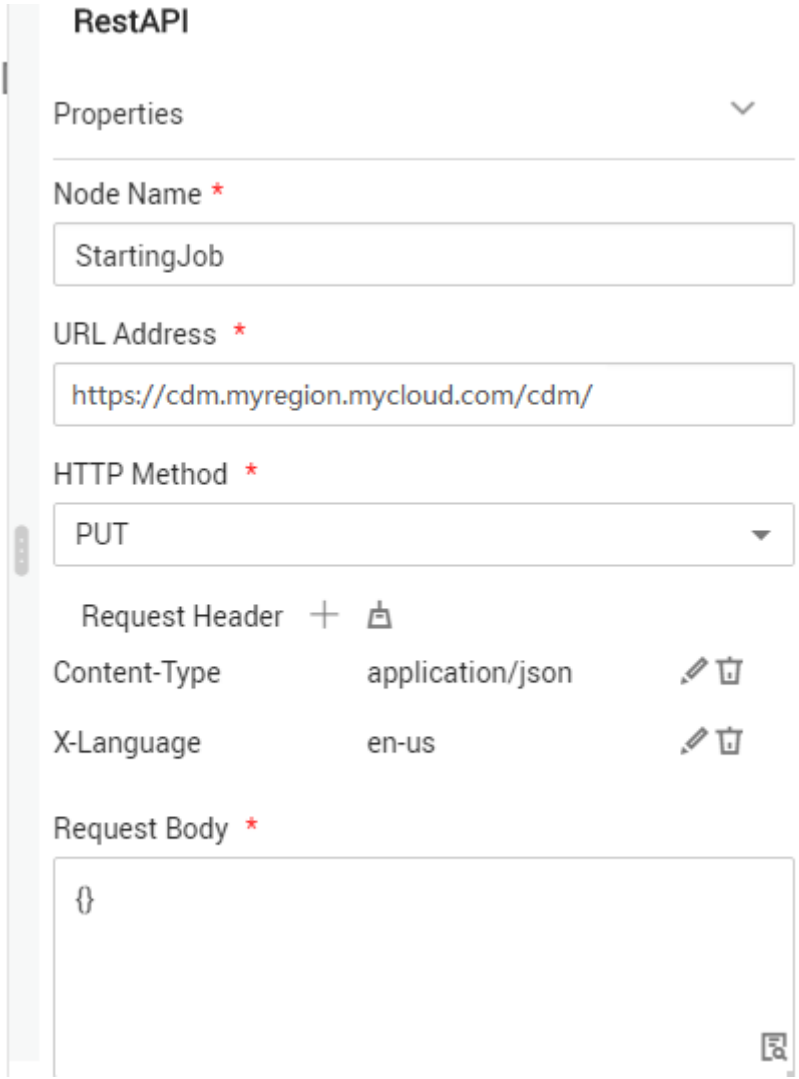
   b. **URL Address**: Keep the values of **project_id** and **cluster_id** consistent with those in **2**. Set the job name to **dws2obs-${currentTime}**. The format is https://{*Endpoint*}/cdm/v1.0/{*project_id*}/clusters/{*cluster_id*}/cdm/job/{*job_name*}/start.

   c. **HTTP Method**: Enter **PUT**.

   d. **Request Header**:

      - Content-Type = application/json
      - X-Language = en-us

Figure 3-4 Properties of the node for running the CDM job



4. Configure the **WaitingJobCompletion** node.

   CDM jobs are run asynchronously. Therefore, even if the REST request for running the job returns 200, it does not mean that the data has been migrated successfully. If a computing job depends on the CDM job, a RestAPI node is required to periodically check whether the migration is successful. Computing is performed only when the migration is successful. For details about the API used to check whether the CDM migration is successful, see section "Querying Job Status" in ***Cloud Data Migration API Reference***.

   After configuring the RestAPI node for running the CDM job, add the node for waiting for the CDM job completion. The node properties are as follows:

   a. Node Name: Wait until the job is complete.

   b. **URL Address**: The format is https://{*Endpoint*}/cdm/v1.0/{*project_id*}/clusters/{*cluster_id*}/cdm/job/{*job_name*}/status. Keep the values of **project_id** and **cluster_id** consistent with those in **2**. Set the job name to **dws2obs-${currentTime}**.

   c. **HTTP Method**: Enter **GET**.

d. **Request Header**:
   - Content-Type = application/json
   - X-Language = en-us

e. **Check Return Value**: Select **YES**.

f. **Property Path**: Enter **submissions[0].status**.

g. **Request Success Flag**: Set this parameter to **SUCCEEDED**.

h. Retain default values for other parameters.

5. (Optional) Configure the **DeletingJob** node.

   You can delete jobs as required. DataArts Factory periodically creates CDM jobs to implement incremental migration. Therefore, a large number of jobs exist in the CDM cluster. After the migration is successful, you can delete the jobs that have been successfully executed. To delete a CDM job, add a RestAPI node for deleting CDM jobs after the node for querying the CDM job status. DataArts Factory calls the API for deleting a job described in ***Cloud Data Migration API Reference***.

   Properties of the node for deleting the CDM job are as follows:

   a. **Node Name**: Enter **DeletingJob**.

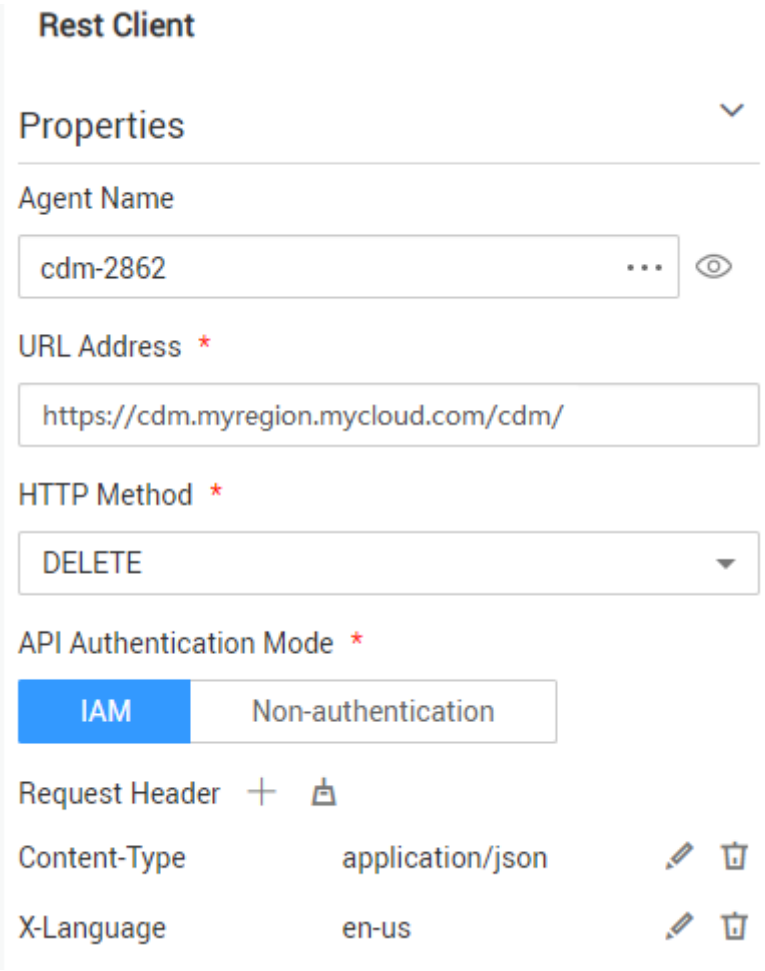
   b. **URL Address**: The format is https://{*Endpoint*}/cdm/v1.0/{*project_id*}/clusters/{*cluster_id*}/cdm/job/{*job_name*}. Keep the values of **project_id** and **cluster_id** consistent with those in **2**. Set the job name to **dws2obs-${currentTime}**.

   c. **HTTP Method**: Enter **DELETE**.

   d. **Request Header**:
      - Content-Type = application/json
      - X-Language = en-us

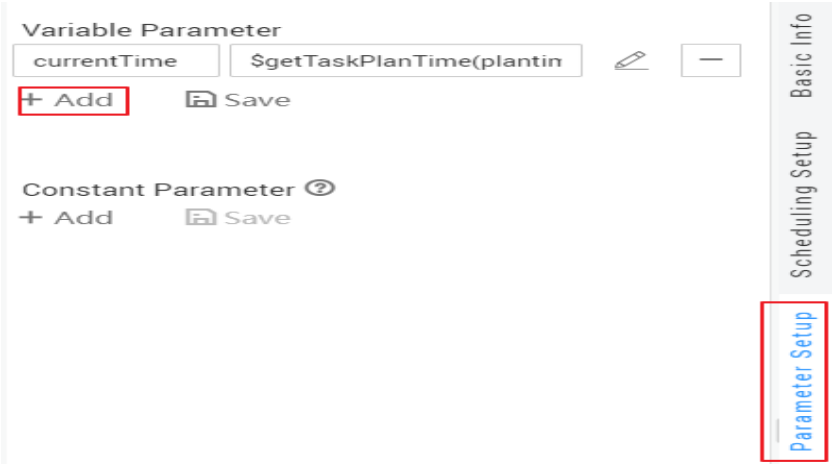
   e. Retain default values for other parameters.

Figure 3-5 Properties of the node for deleting the CDM job



6. To perform computing operations after the migration is complete, you can add various computing nodes.
7. Configure job parameters in DataArts Factory.
   a. Configure the job parameters shown in **Figure 3-6**.
      - startTime = $getTaskPlanTime(plantime,@@yyyyMMddHHmmss@@,-24*60*60)
      - currentTime = $getTaskPlanTime(plantime,@@yyyyMMdd-HHmm@@,0)

Figure 3-6 Configuring job parameters in DataArts Factory



b. After saving the job, choose **Scheduling Configuration** > **Periodic Scheduling** and set the scheduling period to one day.

   In this way, DataArts Factory works with CDM to migrate data generated on the previous day every day.

4 Creating Table Migration Jobs in Batches Using CDM Nodes

Scenario

In a service system, data sources are usually stored in different tables to reduce the size of a single table in complex application scenarios.

In this case, you need to create a data migration job for each table when using CDM to integrate data. This tutorial describes how to use the For Each and CDM nodes provided by the DataArts Factory module to create table migration jobs in batches.

In this tutorial, the source MySQL database has three tables, mail01, mail02, and mail03. The tables have the same structure but different data content. The destination is MRS Hive.

Prerequisites

- You have created a CDM cluster.
- MRS Hive has been enabled.
- Databases and tables have been created in MRS Hive.

Creating a Link

Step 1 Log in to the DataArts Studio console, locate the target DataArts Studio instance, and click **Access** on the instance card.

Step 2 Locate a workspace and click **DataArts Migration**.

Step 3 In the **Operation** column, click **Job Management**.

Step 4 Click the **Links** tab and then **Driver Management**. Upload the MySQL database driver by following the instructions in **Managing Drivers**.

Step 5 Click the **Links** tab and then **Create Link**. Select **MySQL** and click **Next** to configure parameters for the link. After the configuration is complete, click **Save** to return to the **Links** page.

Table 4-1 Parameters for a link to a MySQL database

Parameter	Description	Example Value
Name	Link name, which should be defined based on the data source type, so it is easier to remember what the link is for	mysql_link
Database Server	IP address or domain name of the database to connect Click **Select** next to the text box and select a MySQL DB instance in the displayed dialog box.	192.168.0.1
Port Number	Port of the database to connect	3306
Database	Name of the database to connect	dbname
Username	Username used for accessing the database This account must have the permissions required to read and write data tables and metadata.	cdm
Password	Password of the user	-
Use Local API	(Optional) Whether to use the local API of the database for acceleration. When you create a MySQL link, CDM automatically enables the **local_infile** system variable of the MySQL database to enable the LOAD DATA function, which accelerates data import to the MySQL database. If this parameter is enabled, the date type that does not meet the format requirements will be stored as 0000-00-00. For details, visit the official MySQL website. If CDM fails to enable this function, contact the database administrator to enable the **local_infile** system variable. Alternatively, set **Use Local API** to **No** to disable API acceleration. If data is imported to RDS for MySQL, the LOAD DATA function is disabled by default. In such a case, you need to modify the parameter group of the MySQL instance and set **local_infile** to **ON** to enable the LOAD DATA function. **NOTE** If **local_infile** on RDS is uneditable, it is the default parameter group. You need to create a parameter group, modify its values, and apply it to the RDS for MySQL instance. For details, see the *Relational Database Service User Guide*.	Yes
Use Agent	This parameter does not need to be configured. The agent function will be unavailable soon.	-

Parameter	Description	Example Value
Agent	This parameter does not need to be configured. The agent function will be unavailable soon.	-
local_infile Character Set	When using local_infile to import data to MySQL, you can configure the encoding format.	utf8
Driver Version	Select a driver version that adapts to the database type.	-
Fetch Size	(Optional) Displayed when you click **Show Advanced Attributes**. Number of rows obtained by each request. Set this parameter based on the data source and the job's data size. If the value is either too large or too small, the job may run for a long time.	1000
Commit Size	(Optional) Displayed when you click **Show Advanced Attributes**. Number of records submitted each time. Set this parameter based on the data destination and the job's data size. If the value is either too large or too small, the job may run for a long time.	-

Parameter	Description	Example Value
Link Attributes	(Optional) Click **Add** to add the JDBC connector attributes of multiple specified data sources. For details, see the JDBC connector document of the corresponding database. The following are some examples: ● **connectTimeout=360000** and **socketTimeout=360000**: When a large amount of data needs to be migrated or the entire table is retrieved using query statements, the migration fails due to connection timeout. In this case, you can customize the connection timeout interval (ms) and socket timeout interval (ms) to prevent failures caused by timeout. ● **tinyInt1isBit=false** or **mysql.bool.type.transform=false**: By default, **tinyInt1isBit** is **true**, indicating that **TINYINT(1)** is processed as a bit, that is, **Types.BOOLEAN**, and **1** or **0** is read as **true** or **false**. As a result, the migration fails. In this case, you can set **tinyInt1isBit** to **false** to avoid migration failures. ● **useCursorFetch=false**: By default, **useCursorFetch** is enabled, indicating that the JDBC connector communicates with relational databases using a binary protocol. Some third-party systems may have compatibility issues, causing migration time conversion errors. In this case, you can disable this function. Open-source MySQL databases support the **useCursorFetch** parameter, and you do not need to set this parameter. ● **allowPublicKeyRetrieval=true**: By default, public key retrieval is disabled for MySQL databases. If TLS is unavailable and an RSA public key is used for encryption, connection to a MySQL database may fail. In this case, you can enable public key retrieval to avoid connection failures.	sslmode=require
Reference Sign	(Optional) Delimiter between the names of the referenced tables or columns. For details, see the product documentation of the corresponding database.	'

Parameter	Description	Example Value
Batch Size	Number of rows written each time. It should be less than **Commit Size**. When the number of rows written reaches the value of **Commit Size**, the rows will be committed to the database.	100

Step 6 Click the **Links** tab and then **Create Link**. Select **MRS Hive** and click **Next** to configure parameters for the link. After the configuration is complete, click **Save** to return to the **Links** page.

Table 4-2 MRS Hive link parameters

Parameter	Remarks	Example
Metric Name	Link name, which should be defined based on the data source type, so it is easier to remember what the link is for	hive
Manager IP	Floating IP address of MRS Manager. Click **Select** next to the **Manager IP** text box to select an MRS cluster. CDM automatically fills in the authentication information.	127.0.0.1
Authentication Method	Authentication method used for accessing MRS • **SIMPLE**: Select this for non-security mode. • **KERBEROS**: Select this for security mode.	KERBEROS
Hive Version	Hive version. Set it to the Hive version on the server.	HIVE_3_X

Parameter	Remarks	Example
Username	If **Authentication Method** is set to **KERBEROS**, you must provide the username and password used for logging in to MRS Manager. If you need to create a snapshot when exporting a directory from HDFS, the user configured here must have the administrator permission on HDFS. To create a data connection for an MRS security cluster, do not use user **admin**. The **admin** user is the default management page user and cannot be used as the authentication user of the security cluster. You can create an MRS user and set **Username** and **Password** to the username and password of the created MRS user when creating an MRS data connection. **NOTE** ● If the CDM cluster version is 2.9.0 or later and the MRS cluster version is 3.1.0 or later, the created user must have the permissions of the **Manager_viewer** role to create links on CDM. To perform operations on databases, tables, and data of a component, you also need to add the user group permissions of the component to the user. ● If the CDM cluster version is earlier than 2.9.0 or the MRS cluster version is earlier than 3.1.0, the created user must have the permissions of **Manager_administrator**, **Manager_tenant**, or **System_administrator** to create links on CDM.	cdm
Password	Password for logging in to MRS Manager	-
OBS storage support	The server must support OBS storage. When creating a Hive table, you can store the table in OBS.	Disabled

Parameter	Remarks	Example
Run Mode	This parameter is used only when the Hive version is **HIVE_3_X**. Possible values are: • **EMBEDDED**: The link instance runs with CDM. This mode delivers better performance. • **STANDALONE**: The link instance runs in an independent process. If you want to connect CDM to multiple Hadoop data sources (MRS, Hadoop, or CloudTable), and both **KERBEROS** and **SIMPLE** authentication modes are available, you must select **STANDALONE** for this parameter. **NOTE** The **STANDALONE** mode is used to solve the version conflict problem. If the connector versions of the source and destination ends of the same link are different, a JAR file conflict occurs. In this case, you need to place the source or destination end in the STANDALONE process to prevent the migration failure caused by the conflict.	EMBEDDED
Use Cluster Config	You can use the cluster configuration to simplify parameter settings for the Hadoop connection.	Disabled

----End

Creating a Sample Job

Step 1 In the **Operation** column, click **Job Management**.

Step 2 Click the **Table/File Migration** tab and then **Create Job** to create a job for migrating data from the first MySQL subtable **mail001** to the MRS Hive table **mail**.

Figure 4-1 Creating a job

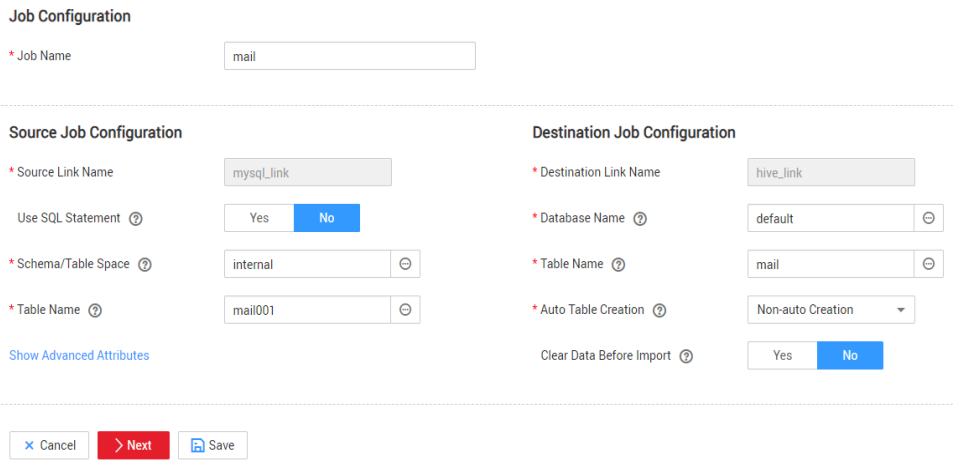
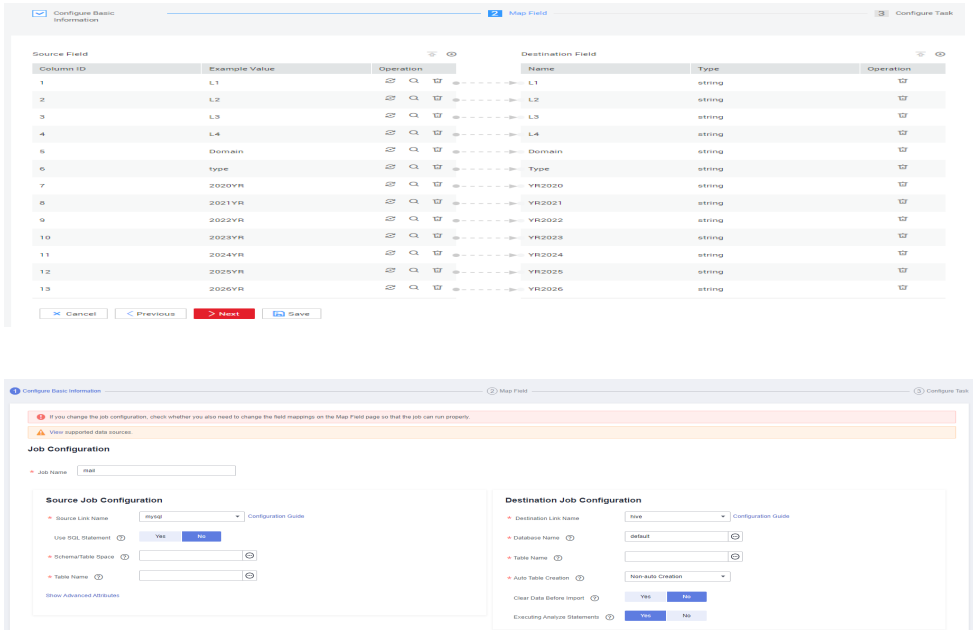


Figure 4-2 Configuring basic information



Step 3 After the sample job is created, view and copy the job JSON for subsequent configuration of data development jobs.

Figure 4-3 Viewing job JSON

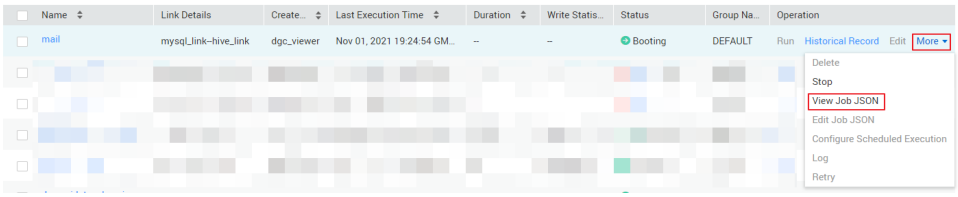
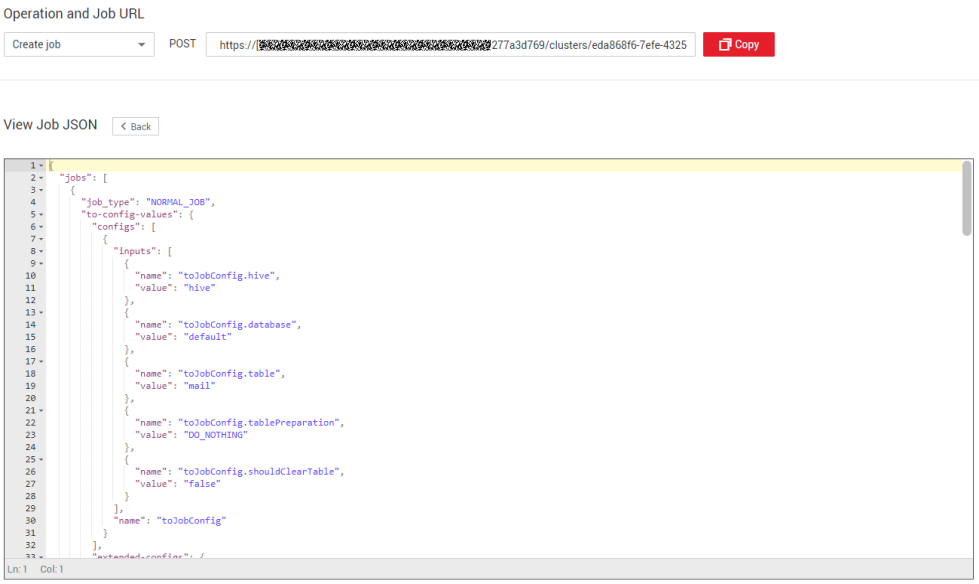


Figure 4-4 Copying job parameters



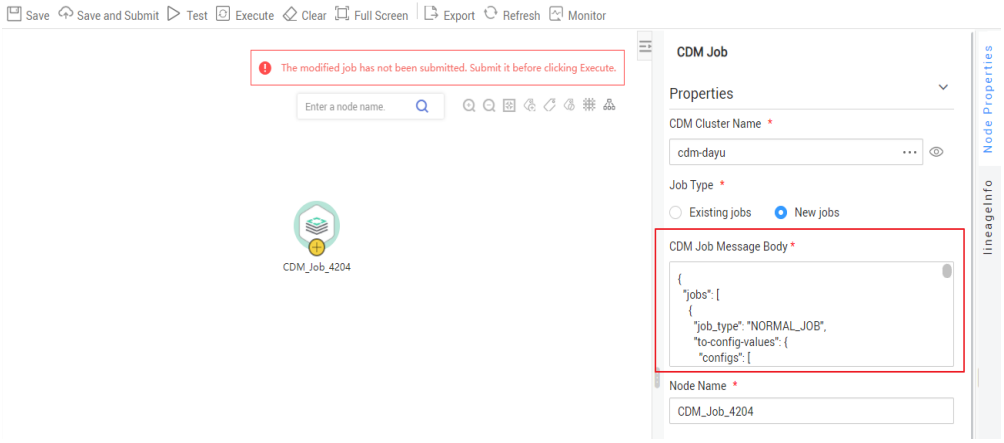
----End

Creating a Data Development Job

Step 1 Locate a workspace and click **DataArts Factory**.

Step 2 Create a subjob named **table**, select the CDM node, select **New jobs** for **Job Type** in **Properties**, and copy and paste the JSON file in **Step 2** to the **CDM Job Message Body**.

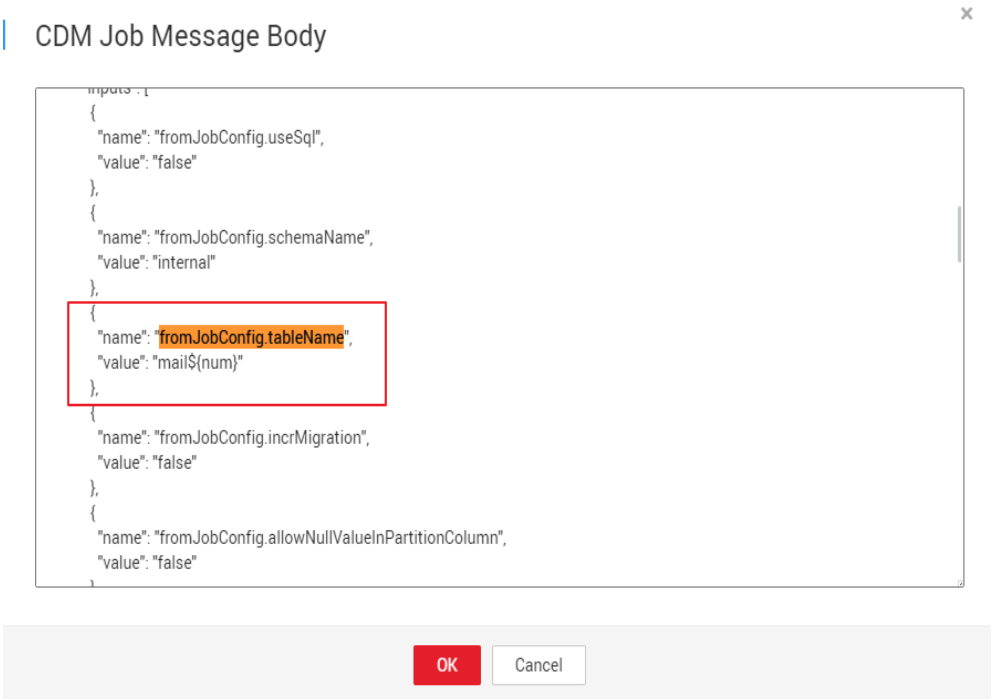
Figure 4-5 Configuring the CDM job message body



Step 3 Edit the CDM job message body.

1. Since there are three source tables **mail001**, **mail002**, and **mail003**, you need to set **fromJobConfig.tableName** to **mail${num}** in the JSON file of the job. The following figure shows the parameters for creating a main job.

Figure 4-6 Editing JSON

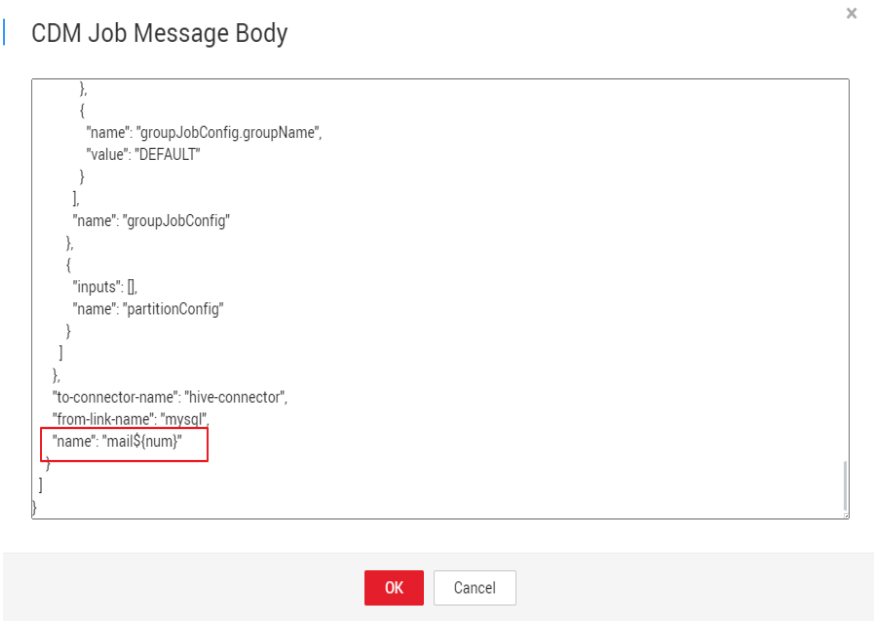


2. The name of each data migration job must be unique. Therefore, you need to change the value of **name** in the JSON file to **mail${num}** to create multiple CDM jobs. The following figure shows the parameters for creating a main job.

   NOTE

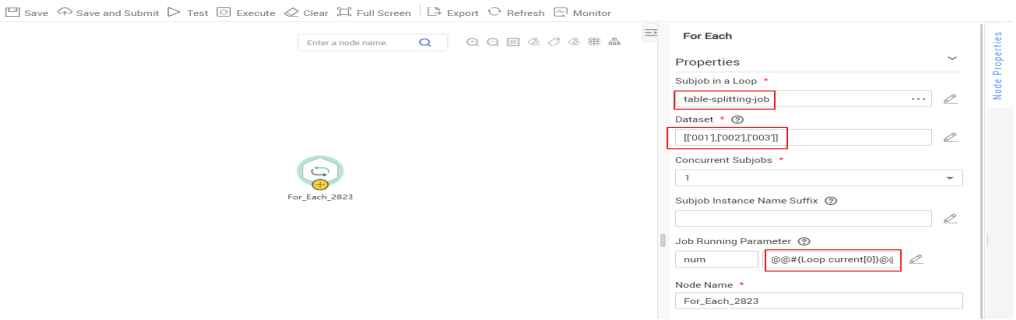
   To create a sharding job, you can change the source link in the job JSON file to a variable that can be easily replaced.

Figure 4-7 Editing JSON



Step 4 Add the **num** parameter, which is invoked in the job JSON file. The following figure shows the parameters for creating a main job.

Figure 4-8 Adding job parameter num



Click **Save and Submit** to save the subjobs.

Step 5 Create the main job **integration_management**. Select the For Each node that executes the subjobs in a loop and transfers parameters **001**, **002**, and **003** to the subjobs to generate different table extraction tasks.

The key configurations are as follows:

- **Subjob in a Loop**: Select **table**.

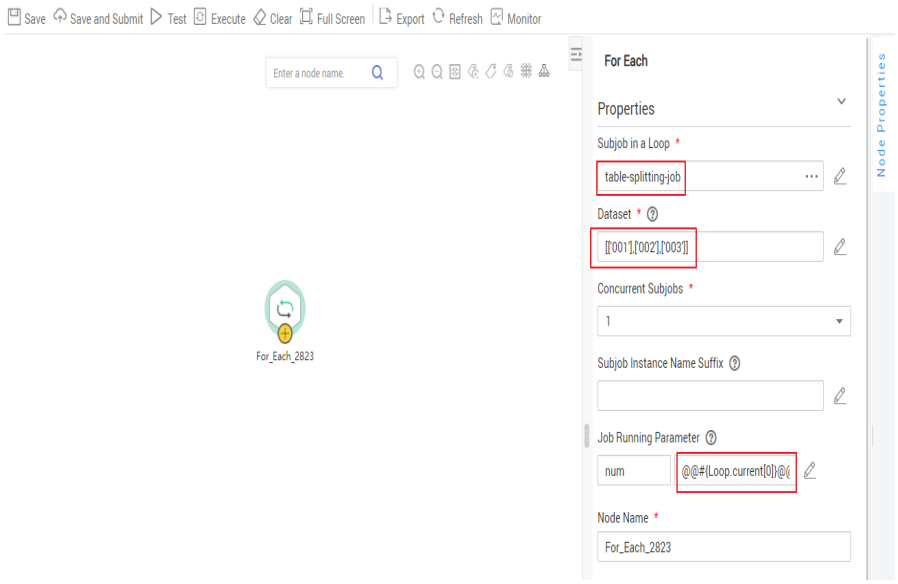
- **Dataset**: Enter **[['001'],['002'],['003']]**.
- **Subjob Parameter Name**: Enter **@@#{Loop.current[0]}@@**.

  NOTE

  Add **@@** to the EL expression of the subjob parameter. If **@@** is not added, dataset 001 will be identified as 1. As a result, the source table name does not exist.

The following figure shows the parameters for creating a main job.

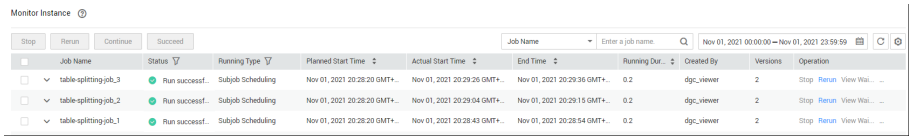
Figure 4-9 Configure key parameters



Click **Save and Submit** to save the main job.

Step 6 After the main job and subjobs are created, test and run the main job to check whether it is successfully created. If the job is successfully executed, the CDM subjobs are successfully created and executed.

Figure 4-10 Viewing the job creation status

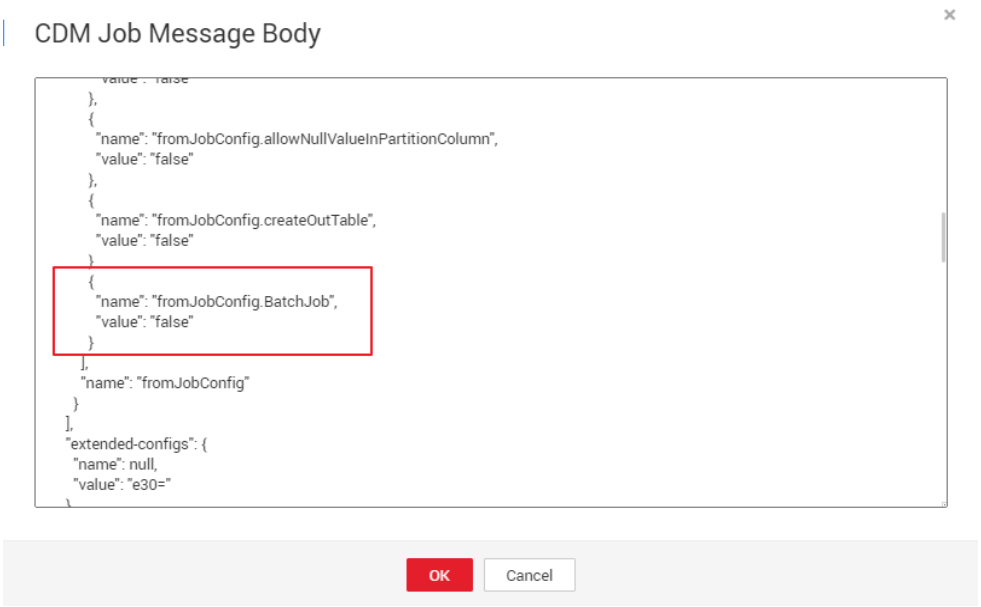


----End

Important Notes

- Some attributes, such as **fromJobConfig.BatchJob**, may not be supported in some CDM versions. If an error is reported during task creation, you need to delete the attribute from the request body. The following figure shows the parameters for creating a main job.

Figure 4-11 Modifying an attribute



- If a CDM node is configured to create a job, the node checks whether a CDM job with the same name is running.
  - If the CDM job is not running, update the job with the same name based on the request body.
  - If a CDM job with the same name is running, wait until the job is run. During this period, the CDM job may be started by other tasks. As a result, the extracted data may not be the same as expected (for example, the job configuration is not updated, or the macro of the running time is not correctly replaced). Therefore, do not start or create multiple jobs with the same name.

5 Simplified Migration of Trade Data to the Cloud and Analysis

5.1 Scenario

Consulting company H uses CDM to import local trade statistics to OBS, and Data Lake Insight (DLI) to analyze trade statistics. In this way, company H builds its big data analytics platform at an extremely low cost, allowing the company more time to focus on their businesses and make innovations continuously.

Background

Company H is a commercial organization in China that engages in collecting trade statistics of major trading nations and buyer data. It has a large-scale trade statistics database. The collected data is widely used in industry research, international trade promotion, and other fields.

In the past, company H used its own big data cluster with maintenance by dedicated personnel. Each year, company H purchased the dedicated bandwidth from China Telecom and China Unicom and invested heavily in equipment room, electric power, private networks, servers, and O&M. However, the company could not satisfy customers' ever-changing service requirements due to insufficient workforce and limited capabilities of its big data cluster. As a result, only 4% of 100 TB inventory data was useful.

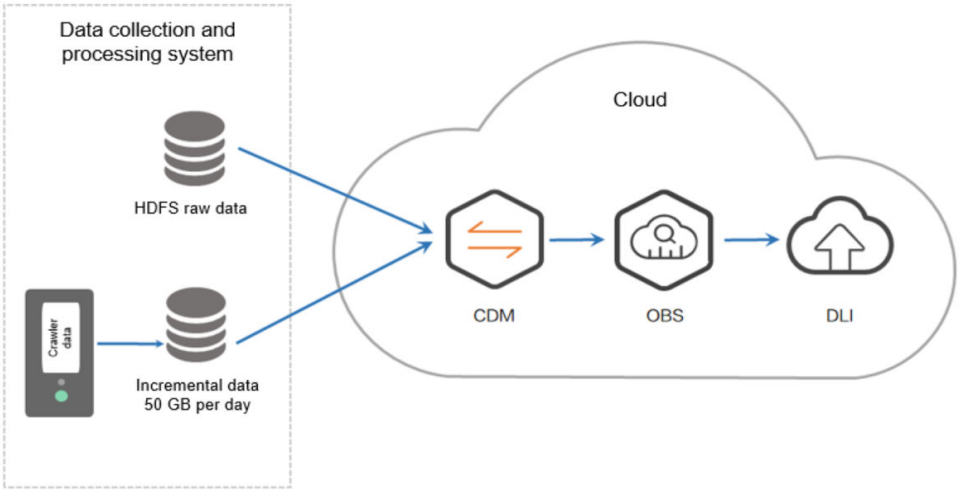
After migrating local trade statistics to Huawei Cloud, company H can make full use of the 100 TB inventory data in maximizing asset monetization, without the need of constructing and maintaining infrastructures but relying on Huawei Cloud's big data analysis capabilities.

CDM and DLI use the pay-per-use billing mode, so maintenance personnel are not required and the dedicated bandwidth cost is reduced. Compared with the on-premises data center, CDM and DLI save the maintenance cost by 70%. In addition, CDM and DLI have low skill demands for personnel and enable smooth migration of existing services, shortening the service rollout period by 50%.

Task

Use CDM, OBS, and DLI to complete trade statistics analysis using the existing data (for example, trade detail records and basic information) of company H's customer data collection and processing system.

Figure 5-1 Scenario scheme



NOTE

When creating an OBS foreign table on DLI, the data storage format of the OBS table must meet the following requirements:

- When you use the DataSource syntax to create an OBS table, the ORC, Parquet, JSON, CSV, Carbon, and Avro formats are supported.
- When you use the Hive syntax to create an OBS table, the Text file, Avro, ORC, SequenceFile, RCFile, Parquet, Carbon formats are supported.

If the storage format of the raw data table does not meet the requirements, you can use CDM to import the raw data to DLI for analysis without uploading the data to OBS.

Data Types

- Trade detail records

  Trade detail records include trade statistics of major trading nations.

  Table 5-1 Trade detail records

Field Name	Field Type	Field Description
hs_code	string	List of import and export offering code
country	smallint	Basic information about countries
dollar_value	double	Transaction amount
quantity	double	Transaction volume

Field Name	Field Type	Field Description
unit	smallint	Measurement unit
b_country	smallint	Basic information about the target country
imex	smallint	Import or export
y_year	smallint	Year
m_month	smallint	Month

- Basic information

  The basic information indicates the dictionary data corresponding to the fields in the trade detail records.

  Table 5-2 Basic information about countries (description of **country**)

Field Name	Field Type	Field Description
countryid	smallint	Country code
country_en	string	English name of a country
country_cn	string	Chinese name of a country

  Table 5-3 Information about the update time (description of **updatetime**)

Field Name	Field Type	Field Description
countryid	smallint	Country code
imex	smallint	Import or export
hs_len	smallint	Length of the offering code
minstartdate	string	Minimum start time
startdate	string	Start time
newdate	string	Update time
minnewdate	string	Last update time

Table 5-4 Information about import and export offering code (description of **hs246**)

Field Name	Field Type	Field Description
id	bigint	ID
hs	string	Offering code
hs_cn	string	Chinese name of an offering
hs_en	string	English name of an offering

Table 5-5 Information about units (description of **unit_general**)

Field Name	Field Type	Field Description
id	smallint	Measurement unit code
unit_en	string	English name of a measurement unit
unit_cn	string	Chinese name of a measurement unit

5.2 Analysis Process

Introduction

To use CDM, OBS, and DLI to analyze trade statistics, you need to perform the following steps:

1. **Using CDM to Upload Data to OBS**
   a. Use CDM to upload the inventory data of company H to OBS.
   b. Configure a scheduled task of CDM to automatically upload incremental data to OBS every day.
2. **Using DLI to Analyze Data**
   Use DLI to directly analyze the service data in OBS to support the customers of company H for trade statistics analysis.

5.3 Using CDM to Upload Data to OBS

5.3.1 Uploading Inventory Data

1. Use **Direct Connect** to establish a Direct Connect connection between the local data center and Huawei Cloud Virtual Private Cloud (VPC).

2. Create an OBS bucket and record the access domain name, port number, access key ID (AK), and secret access key (SK) of the OBS bucket.
3. Create a CDM cluster.

   NOTE

   If a DataArts Studio instance includes a CDM cluster (except the trial version) and the cluster meets your requirements, you do not need to buy a DataArts Migration incremental package.

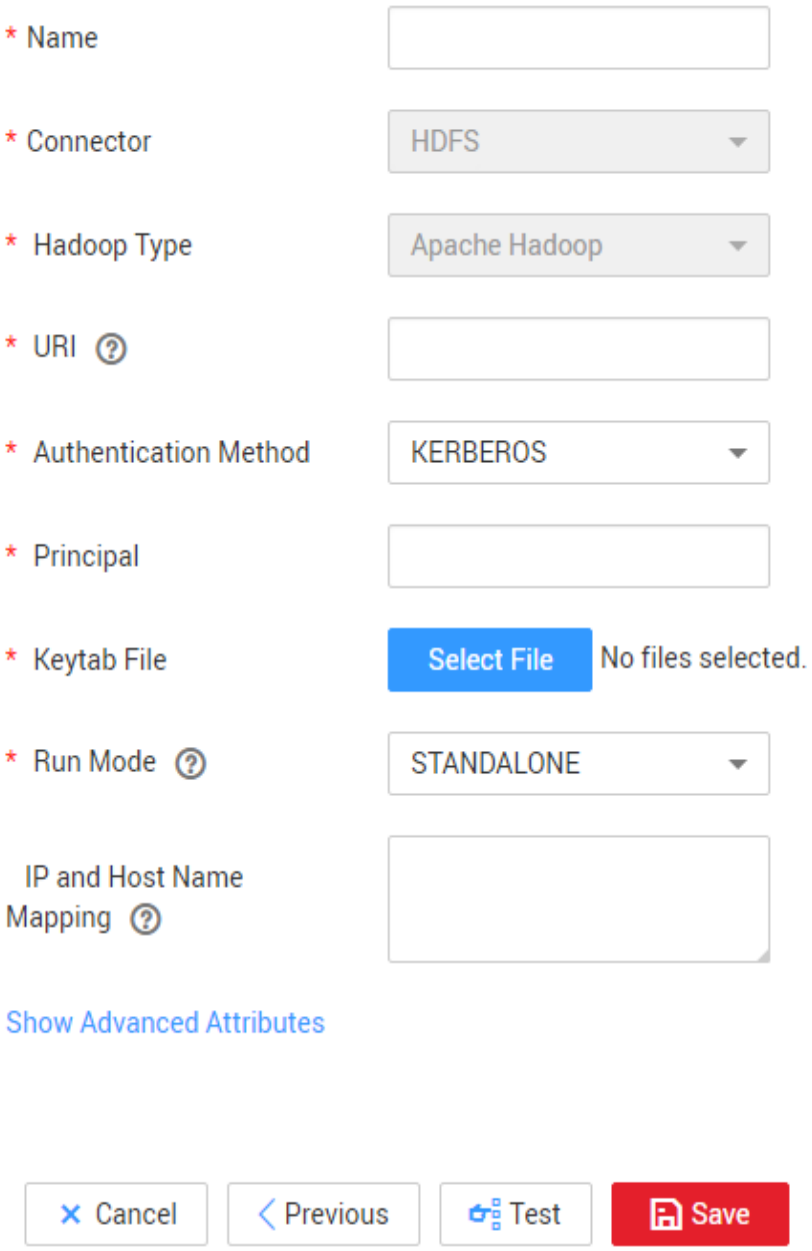
   If you need to create another CDM cluster, buy a CDM incremental package by referring to **Buying a CDM Incremental Package**.

   - **Instance Type**: Select **cdm.xlarge**, which applies to most migration scenarios.
   - **VPC**: VPC of the CDM cluster. Select the VPC that connects to the local data center through Direct Connect.
   - (Optional) **Subnet** and **Security Group**: You can configure either of them.
4. After the cluster is created, choose **Job Management** > **Link Management** > **Create Link**. The page for selecting a link type is displayed. See **Figure 5-2**.

   Figure 5-2 Selecting a connector

5. To connect to the local Apache HDFS of company *H*, select **Apache HDFS**, and click **Next**.

Figure 5-3 Creating an HDFS link



NOTE

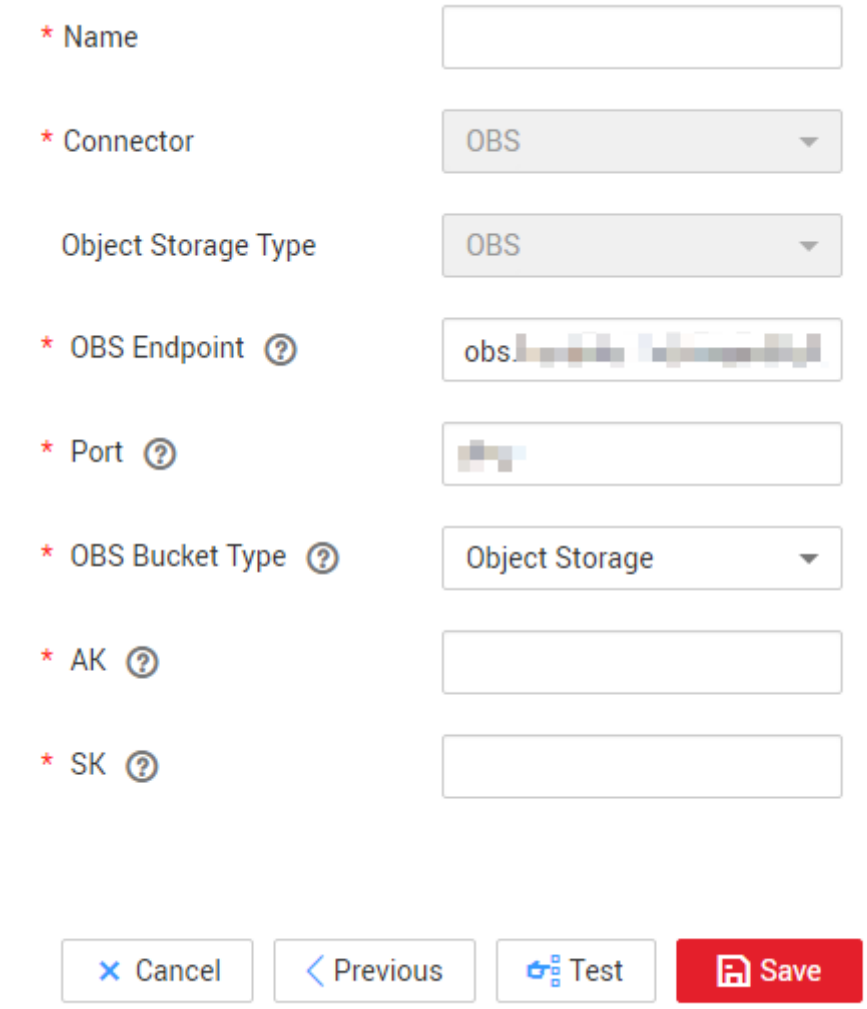
- **Name**: Enter a custom link name, for example, **hdfs_link**.
- **URI**: Enter the NameNode URI of HDFS of company *H*.
- **Authentication Method**: Select **KERBEROS** if Hadoop is in security mode to obtain the **principal** and **keytab** files from the client for authentication.
- **Principal** and **Keytab File**: Obtain the **principal** account and **keytab** file from the Hadoop administrator.

6. Click **Save**. CDM automatically checks whether the link is available.
   - If the link is available, a message is displayed, indicating that the link is successfully saved, and the link management page is displayed.

- If the link is unavailable, check whether the link parameters are correctly configured or whether the firewall of company *H* allows the elastic IP address (EIP) of the CDM cluster to access the data source.

7. Click **Create Link** to create an OBS link. On the page that is displayed, select **Object Storage Service (OBS)** and click **Next**. Set the OBS link parameters as required. See **Figure 5-4**.

Figure 5-4 Creating an OBS link

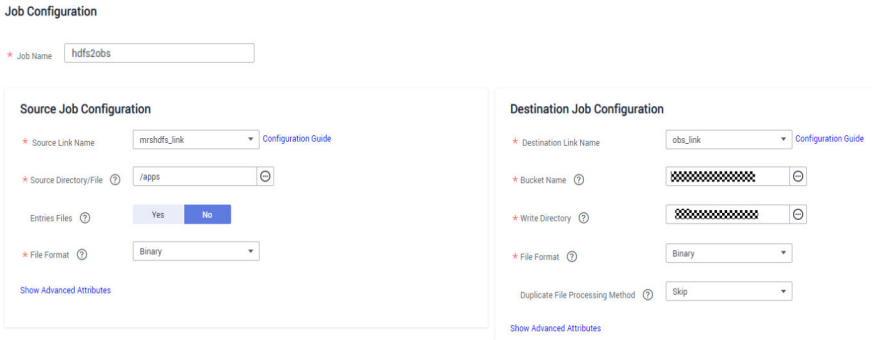


NOTE

- **Name**: Enter a custom link name, for example, **obslink**.
- **OBS Endpoint**: Enter the domain name or IP address of OBS, for example, **obs.myhuaweicloud.com**.
- **Port**: Enter the port number of OBS, for example, **443**.
- **OBS Bucket Type**: Select a value from the drop-down list box as required.
- **AK** and **SK**: Enter the AK and SK used for accessing the OBS database. To obtain the AK and SK, log in to the management console, click the username in the upper right corner, and select **My Credentials** from the drop-down list. On the displayed page, choose **Access Keys** in the left navigation pane.

8. Click **Save**. The **Link Management** page is displayed.
9. Choose **Table/File Migration** > **Create Job** to create a job for migrating trade statistics of company *H* to OBS. See **Figure 5-5**.

Figure 5-5 Creating a job



NOTE

- **Job Name**: Enter a user-defined job name.
- **Source Link Configuration**:
  - **Source Link Name**: Select the HDFS link **hdfs_link** created in 5.
  - **Source Directory/File**: Set this parameter to the local storage path of company *H*'s trade statistics. The value can be either a directory or a file. Set this parameter to a directory. CDM migrates all files in the directory to OBS.
  - **File Format**: Select **Binary**. The file format refers to the format used by CDM to transmit data. The formats of the original files are not changed. **Binary** is recommended for migration between files because the transmission efficiency and performance are optimal.
- **Destination Link Configuration**:
  - **Destination Link Name**: Select the OBS link **obslink** created in 7.
  - **Bucket Name** and **Write Directory**: Enter the path for storing trade statistics in OBS. CDM writes the files to this path.
  - **File Format**: Select **Binary**. Similar to the source link, the formats of the original files are not changed.
  - **Duplicate File Processing Method**: Select **Skip**. CDM determines that a file is a duplicate file only when the file name and file size are the same on the source and destination ends. In this case, CDM skips the file and does not migrate the file to OBS.

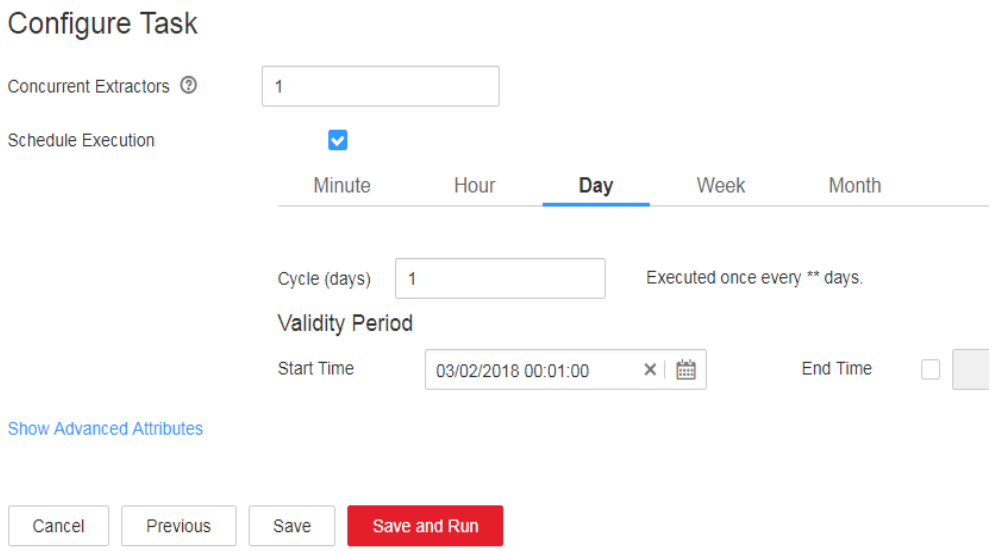
10. Click **Next** to go to the tab page for configuring the task parameters. For the migration of inventory data, retain the default values of the parameters.
11. Click **Save and Run**. On the displayed job management page, you can view the job execution progress and result.
12. After the job is successfully executed, click **Historical Record** to view the number of written rows, number of read rows, number of written bytes, number of written files, and execution logs.

5.3.2 Uploading Incremental Data

1. After uploading inventory data using CDM, click **Edit** in the **Operation** column to modify a job.

2. Retain the values of the basic parameters, and click **Next** to modify the task parameters. See **Figure 5-6**.

   Figure 5-6 Configuring a scheduled task



3. Select **Schedule Execution** and configure the scheduled task.
   - Set **Cycle (days)** to 1 day.
   - Set **Start Time** to 00:01:00 every day.

   In this way, CDM automatically performs full migration in the early morning every day. However, because **Duplicate File Processing Method** is set to **Skip**, files with the same name and size are not migrated. Therefore, only new files are uploaded every day.
4. Click **Save**.

5.4 Analyzing Data

Use DLI to analyze the trade statistics stored in OBS buckets.

Prerequisites

When creating an OBS foreign table on DLI, the data storage format of the OBS table must meet the following requirements:

- When you use the DataSource syntax to create an OBS table, the ORC, Parquet, JSON, CSV, Carbon, and Avro formats are supported.
- When you use the Hive syntax to create an OBS table, the Text file, Avro, ORC, SequenceFile, RCFile, Parquet, Carbon formats are supported.

If the storage format of the raw data table does not meet the requirements, you can use CDM to import the raw data to DLI for analysis without uploading the data to OBS.

Procedure

1. Log in to the DLI console and create a database by referring to **Creating a Database**.
2. Create an OBS foreign table by referring to **Creating an OBS Table**, including the trade statistics database, trade detail record table, and basic information table.
3. Develop SQL scripts on the DLI console for trade statistics analysis to meet service requirements.

