TiDB Data Migration Documentation

PingCAP Inc.

20220809

Table of Contents

1 Overview

1.1 Data Migration Overview

TiDB Data Migration (DM) is an integrated data migration task management platform that supports the full data migration and the incremental data replication from MySQL/-MariaDB into TiDB. It can help to reduce the operations cost and simplify the troubleshooting process.

> **Note:**
>
> DM migrates data to TiDB in the form of SQL statements, so each version of DM is compatible with **all versions** of TiDB. In the production environment, it is recommended to use the latest released version of DM. To install DM, see DM download link.

1.1.1 Architecture

The Data Migration tool includes three components: DM-master, DM-worker, and dmctl.

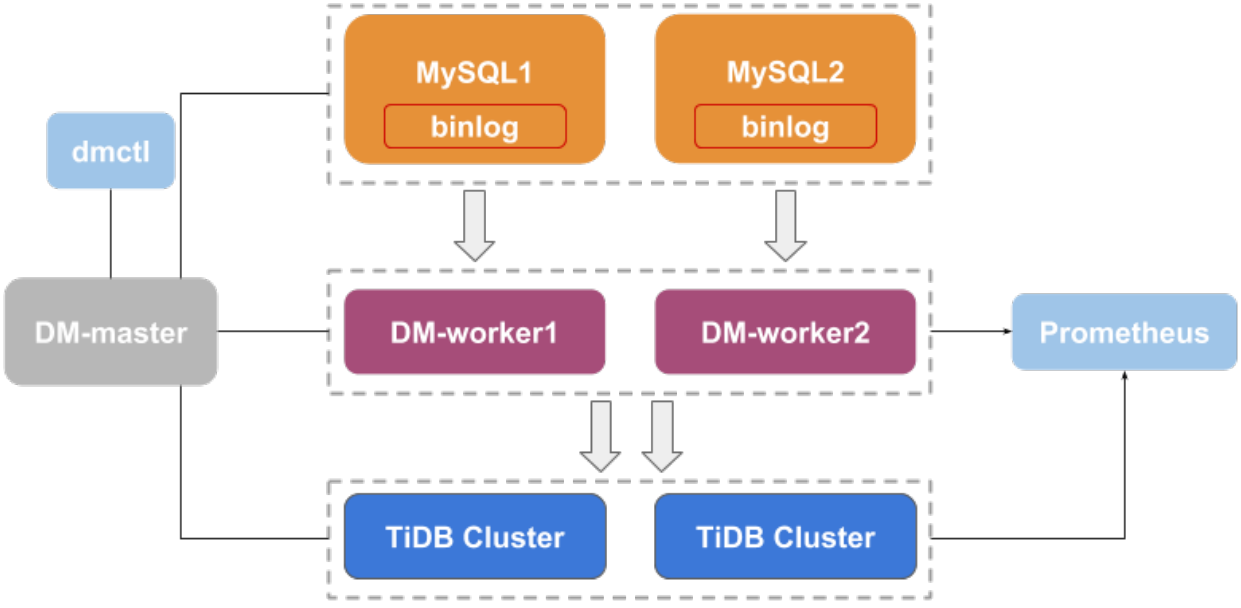


Figure 1: Data Migration architecture

1.1.1.1 DM-master

DM-master manages and schedules the operation of data migration tasks.

- Storing the topology information of the DM cluster
- Monitoring the running state of DM-worker processes
- Monitoring the running state of data migration tasks
- Providing a unified portal for the management of data migration tasks
- Coordinating the DDL migration of sharded tables in each instance under the sharding scenario

1.1.1.2 DM-worker

DM-worker executes specific data migration tasks.

- Persisting the binlog data to the local storage
- Storing the configuration information of the data migration subtasks
- Orchestrating the operation of the data migration subtasks
- Monitoring the running state of the data migration subtasks

After DM-worker is started, it automatically migrates the upstream binlog to the local configuration directory (the default migration directory is `<deploy_dir>/relay_log` if DM is deployed using `DM-Ansible`). For details about DM-worker, see DM-worker Introduction. For details about the relay log, see Relay Log.

1.1.1.3 dmctl

dmctl is the command line tool used to control the DM cluster.

- Creating/Updating/Dropping data migration tasks
- Checking the state of data migration tasks
- Handling the errors during data migration tasks
- Verifying the configuration correctness of data migration tasks

1.1.2 Data migration features

This section describes the data migration features provided by the Data Migration tool.

1.1.2.1 Schema and table routing

The schema and table routing feature means that DM can migrate a certain table of the upstream MySQL or MariaDB instance to the specified table in the downstream, which can be used to merge or migrate the sharding data.

1.1.2.2 Block and allow lists migration at the schema and table levels

The block and allow lists filtering rule of the upstream database instance tables is similar to MySQL `replication-rules-db/replication-rules-table`, which can be used to filter or only replicate all operations of some databases or some tables.

1.1.2.3 Binlog event filtering

Binlog event filtering is a more fine-grained filtering rule than the block and allow lists filtering rule. You can use statements like `INSERT` or `TRUNCATE TABLE` to specify the binlog events of `schema/table` that you need to migrate or filter out.

1.1.2.4 Sharding support

DM supports merging the original sharded instances and tables into TiDB, but with some restrictions.

1.1.3 Usage restrictions

Before using the DM tool, note the following restrictions:

- Database version
  - 5.5 < MySQL version < 8.0
  - MariaDB version >= 10.1.2

> **Note:**
> If there is a primary-secondary migration structure between the upstream MySQL/MariaDB servers, then choose the following version.
>
> - 5.7.1 < MySQL version < 8.0
> - MariaDB version >= 10.1.3

Data Migration prechecks the corresponding privileges and configuration automatically while starting the data migration task using dmctl.

- DDL syntax
  - Currently, TiDB is not compatible with all the DDL statements that MySQL supports. Because DM uses the TiDB parser to process DDL statements, it only supports the DDL syntax supported by the TiDB parser. For details, see MySQL Compatibility.

  - DM reports an error when it encounters an incompatible DDL statement. To solve this error, you need to manually handle it using dmctl, either skipping this DDL statement or replacing it with a specified DDL statement(s). For details, see Skip or replace abnormal SQL statements.

- Sharding

  - If conflict exists between sharded tables, solve the conflict by referring to handling conflicts of auto-increment primary key. Otherwise, data migration is not supported. Conflicting data can cover each other and cause data loss.
  - For other sharding restrictions, see Sharding DDL usage restrictions.

- Operations

  - After DM-worker is restarted, the data migration task cannot be automatically restored. You need to manually run `start-task`. For details, see Manage the Data Migration Task.
  - After DM-worker is restarted, the DDL lock migration cannot be automatically restored in some conditions. You need to manually handle it. For details, see Handle Sharding DDL Locks Manually.

- Switching DM-worker connection to another MySQL instance

  When DM-worker connects the upstream MySQL instance via a virtual IP (VIP), if you switch the VIP connection to another MySQL instance, DM might connect to the new and old MySQL instances at the same time in different connections. In this situation, the binlog migrated to DM is not consistent with other upstream status that DM receives, causing unpredictable anomalies and even data damage. To make necessary changes to DM manually, refer to Switch DM-worker connection via virtual IP.

1.2 DM-worker Introduction

DM-worker is a tool used to migrate data from MySQL/MariaDB to TiDB.

It has the following features:

- Acts as a secondary database of any MySQL or MariaDB instance
- Reads the binlog events from MySQL/MariaDB and persists them to the local storage
- A single DM-worker supports migrating the data of one MySQL/MariaDB instance to multiple TiDB instances
- Multiple DM-workers support migrating the data of multiple MySQL/MariaDB instances to one TiDB instance

1.2.1 DM-worker processing unit

A DM-worker task contains multiple logic units, including relay log, the dump processing unit, the load processing unit, and binlog replication.

1.2.1.1 Relay log

The relay log persistently stores the binlog data from the upstream MySQL/MariaDB and provides the feature of accessing binlog events for the binlog replication.

Its rationale and features are similar to the secondary relay log of MySQL. For details, see The Secondary Relay Log.

1.2.1.2 Dump processing unit/dump unit

The dump processing unit dumps the full data from the upstream MySQL/MariaDB to the local disk.

1.2.1.3 Load processing unit/load unit

The load processing unit reads the files of the dump unit and then loads these files to the downstream TiDB.

1.2.1.4 Binlog replication/sync processing unit

Binlog replication processing unit (namely, sync processing unit), reads the binlog events of the relay log, transforms these events to SQL statements, and then applies these statements to the downstream TiDB.

1.2.2 Privileges required by DM-worker

This section describes the upstream and downstream database users' privileges required by DM-worker, and the user privileges required by the respective processing unit.

1.2.2.1 Upstream database user privileges

The upstream database (MySQL/MariaDB) user must have the following privileges:

Privilege	Scope
`SELECT`	Tables
`RELOAD`	Global
`REPLICATION SLAVE`	Global
`REPLICATION CLIENT`	Global

If you need to migrate the data from `db1` to TiDB, execute the following `GRANT` statement:

```
GRANT RELOAD,REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'your_user'@'
    ↪ your_wildcard_of_host'
GRANT SELECT ON db1.* TO 'your_user'@'your_wildcard_of_host';
```

If you also need to migrate the data from other databases into TiDB, make sure the same privileges are granted to the user of the respective databases.

1.2.2.2 Downstream database user privileges

The downstream database (TiDB) user must have the following privileges:

Privilege	Scope
SELECT	Tables
INSERT	Tables
UPDATE	Tables
DELETE	Tables
CREATE	Databases, tables
DROP	Databases, tables
ALTER	Tables
INDEX	Tables

Execute the following GRANT statement for the databases or tables that you need to migrate:

```
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,ALTER,INDEX ON db.table TO '
    ↪ your_user'@'your_wildcard_of_host';
```

1.2.2.3 Minimal privilege required by each processing unit

Processing unit	Minimal upstream (MySQL/MariaDB) privilege	Minimal downstream (TiDB) privilege	Minimal system privilege
Relay log	REPLICATION SLAVE (reads the binlog)REPLICATION ↪ CLIENT (show master ↪ status, show slave status)	NULL	Read/Write local files

Processing unit	Minimal upstream (MySQL/MariaDB) privilege	Minimal downstream (TiDB) privilege	Minimal system privilege
Dump	SELECTRELOAD (flushes tables with Read lock and unlocks tables)	NULL	Write local files
Load	NULL	SELECT (Query the checkpoint history)CREATE (creates a database/table)DELETE ↪ (deletes checkpoint)INSERT (Inserts the Dump data)	Read/Write local files
Binlog replication	REPLICATION SLAVE (reads the binlog)REPLICATION ↪ CLIENT (show master ↪ status, show slave status)	SELECT (shows the index and column)INSERT (DML)UPDATE ↪ (DML)DELETE ↪ (DML)CREATE ↪ (creates a database/table)DROP ↪ (drops databases/tables)ALTER (alters a table)INDEX (creates/-drops an index)	Read/Write local files

Note:

These privileges are not immutable and they change as the request changes.

1.3 Data Migration Relay Log

The Data Migration (DM) relay log consists of a set of numbered files containing events that describe database changes, and an index file that contains the names of all used relay log files.

After DM-worker is started, it automatically replicates the upstream binlog to the local configuration directory (the default migration directory is `<deploy_dir>/relay_log` if DM is deployed using `DM-Ansible`). When DM-worker is running, it migrates the upstream binlog to the local file in real time. The sync processing unit of DM-worker, reads the binlog events of the local relay log in real time, transforms these events to SQL statements, and then migrates these statements to the downstream database.

This document introduces the directory structure, initial migration rules and data purge of DM relay logs.

1.3.1 Directory structure

An example of the directory structure of the local storage for a relay log:

```
<deploy_dir>/relay_log/
|-- 7e427cc0-091c-11e9-9e45-72b7c59d52d7.000001
|   |-- mysql-bin.000001
|   |-- mysql-bin.000002
|   |-- mysql-bin.000003
|   |-- mysql-bin.000004
|   `-- relay.meta
|-- 842965eb-091c-11e9-9e45-9a3bff03fa39.000002
|   |-- mysql-bin.000001
|   `-- relay.meta
`-- server-uuid.index
```

- `subdir`:
  - DM-worker stores the binlog migrated from the upstream database in the same directory. Each directory is a `subdir`.
  - `subdir` is named `<Upstream database UUID>.<Local subdir serial number >`.
  - After a switch between primary and secondary instances in the upstream, DM-worker generates a new `subdir` directory with an incremental serial number.

    * In the above example, for the `7e427cc0-091c-11e9-9e45-72b7c59d52d7.000001` directory, `7e427cc0-091c-11e9-9e45-72b7c59d52d7` is the upstream database UUID and `000001` is the local `subdir` serial number.

- `server-uuid.index`: Records a list of names of currently available `subdir` directory.
- `relay.meta`: Stores the information of the migrated binlog in each `subdir`. For example,

```
$ cat c0149e17-dff1-11e8-b6a8-0242ac110004.000001/relay.meta
binlog-name = "mysql-bin.000010"                          # The name of the currently replicated binlog.
binlog-pos = 63083620                                     # The position of the currently replicated binlog.
binlog-gtid = "c0149e17-dff1-11e8-b6a8-0242ac110004:1-3328" # GTID of the currently replicated binlog.
                                                          # There might be multiple GTIDs.
$ cat 92acbd8a-c844-11e7-94a1-1866daf8accc.000001/relay.meta
binlog-name = "mysql-bin.018393"
binlog-pos = 277987307
binlog-gtid = "3ccc475b-2343-11e7-be21-6c0b84d59f30:1-14,406a3f61-690d-11e7-87c5-6c92bf46f384:1-94321383,53bfca22-690d-11e7-8a62-18ded7a37b78:1-495,686e1ab6-c47e-11e7-a42c-6c92bf46f384:1-34981190,03fc0263-28c7-11e7-a653-6c0b84d59f30:1-7041423,05474d3c-28c7-11e7-8352-203db246dd3d:1-170,10b039fc-c843-11e7-8f6a-1866daf8d810:1-308290454"
```

1.3.2 Initial migration rules

For each start of DM-worker (or the relay log resuming migration after a pause), the starting position of migration includes the following conditions:

- If a valid local relay log (a valid relay log is a relay log with valid `server-uuid.index`, `subdir` and `relay.meta` files), DM-worker resumes migration from a position recorded by `relay.meta`.
- If a valid local relay log does not exist, and `relay-binlog-name` or `relay-binlog-gtid` is not specified in the DM configuration file:
  - In the non-GTID mode, DM-worker starts migration from the initial upstream binlog and migrates all the upstream binlog files to the latest successively.
  - In the GTID mode, DM-worker starts migration from the initial upstream GTID.

> **Note:**
> If the upstream relay log is purged, an error occurs. In this case, set `relay-binlog-gtid` to specify the starting position of migration.

- If a valid local relay log does not exist:
  - In the non-GTID mode, if `relay-binlog-name` is specified, DM-worker starts migration from the specified binlog file.
  - In the GTID mode, if `relay-binlog-gtid` is specified, DM-worker starts migration from the specified GTID.

1.3.3 Data purge

Through the detection mechanism of reading and writing files, DM-worker does not purge the relay log that is being used or will be used later by the existing data migration tasks.

The data purge methods for the relay log include automatic purge and manual purge.

1.3.3.1 Automatic data purge

You can configure the automatic data purge strategy of DM-worker using the following two methods:

Method 1: Use the command-line options.

- `purge-interval`
  - The interval of automatic purge in the background, in seconds.
  - "3600" by default, indicating a background purge task is performed every 3600 seconds.
- `purge-expires`
  - The number of hours that a relay log that is not written by the relay processing unit, or that does not need to be read by the existing data migration task, can be retained for before being purged in the automatic background purge.
  - "0" by default, indicating data purge is not performed according to the update time of the relay log.
- `purge-remain-space`
  - The amount of remaining disk space in GB less than which the specified DM-worker machine tries to purge the relay log that can be purged securely in the automatic background purge. If it is set to `0`, data purge is not performed according to the remaining disk space.
  - "15" by default, indicating when the available disk space is less than 15GB, DM-master tries to purge the relay log securely.

Method 2: Add the `[purge]` section in the configuration file of DM-worker.

```
## relay log purge strategy
[purge]
interval = 3600
expires = 24
remain-space = 15
```

1.3.3.2 Manual data purge

Manual data purge means using the `purge-relay` command provided by dmctl to specify `subdir` and the binlog name thus to purge all the relay logs **before** the specified binlog. If the `-subdir` option in the command is not specified, all relay logs **before** the current relay log sub-directory are purged.

Assuming that the directory structure of the current relay log is as follows:

```
$ tree .
.
|-- deb76a2b-09cc-11e9-9129-5242cf3bb246.000001
|   |-- mysql-bin.000001
|   |-- mysql-bin.000002
|   |-- mysql-bin.000003
|   `-- relay.meta
|-- deb76a2b-09cc-11e9-9129-5242cf3bb246.000003
|   |-- mysql-bin.000001
|   `-- relay.meta
|-- e4e0e8ab-09cc-11e9-9220-82cc35207219.000002
|   |-- mysql-bin.000001
|   `-- relay.meta
`-- server-uuid.index

$ cat server-uuid.index
deb76a2b-09cc-11e9-9129-5242cf3bb246.000001
e4e0e8ab-09cc-11e9-9220-82cc35207219.000002
deb76a2b-09cc-11e9-9129-5242cf3bb246.000003
```

- Executing the following `purge-relay` command in dmctl purges all relay log files **before** `e4e0e8ab-09cc-11e9-9220-82cc35207219.000002/mysql-bin.000001`, which is all relay log files in `deb76a2b-09cc-11e9-9129-5242cf3bb246.000001`.

  ```
  » purge-relay -w 10.128.16.223:10081 --filename mysql-bin.000001 --sub-
      ↪ dir e4e0e8ab-09cc-11e9-9220-82cc35207219.000002
  ```

- Executing the following `purge-relay` command in dmctl purges all relay log file **before the current** (`deb76a2b-09cc-11e9-9129-5242cf3bb246.000003`) directory's

`mysql-bin.000001`, which is all relay log files in `deb76a2b-09cc-11e9-9129-5242cf3bb246.000001` and `e4e0e8ab-09cc-11e9-9220-82cc35207219.000002`.

```
» purge-relay -w 10.128.16.223:10081 --filename mysql-bin.000001
```

2 Features

2.1 Data Migration Features

This document describes the data migration features provided by the Data Migration tool and explains the configuration of corresponding parameters.

For different DM versions, pay attention to the different match rules of schema or table names in the table routing, block & allow lists, and binlog event filter features:

- For DM v1.0.5 or later versions, all the above features support the wildcard match. For all versions of DM, note that there can be **only one** `*` in the wildcard expression, and `*` **must be placed at the end**.
- For DM versions earlier than v1.0.5, table routing and binlog event filter support the wildcard but do not support the `[...]` and `[!...]` expressions. The block & allow lists only supports the regular expression.

It is recommended that you use the wildcard for matching in simple scenarios.

2.1.1 Table routing

The table routing feature enables DM to migrate a certain table of the upstream MySQL or MariaDB instance to the specified table in the downstream.

Note:

- Configuring multiple different routing rules for a single table is not supported.
- The match rule of schema needs to be configured separately, which is used to migrate `create/drop schema xx`, as shown in `rule-2` of the parameter configuration.

2.1.1.1 Parameter configuration

```
routes:
  rule-1:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    target-schema: "test"
    target-table: "t"
  rule-2:
    schema-pattern: "test_*"
    target-schema: "test"
```

2.1.1.2 Parameter explanation

DM migrates the upstream MySQL or MariaDB instance table that matches the schema -pattern/table-pattern rule provided by Table selector to the downstream target-schema/target-table.

2.1.1.3 Usage examples

This sections shows the usage examples in different scenarios.

2.1.1.3.1 Merge sharded schemas and tables

Assuming in the scenario of sharded schemas and tables, you want to migrate the test_{1,2,3...}.t_{1,2,3...} tables in two upstream MySQL instances to the test.t table in the downstream TiDB instance.

To migrate the upstream instances to the downstream test.t, you must create two routing rules:

- rule-1 is used to migrate DML or DDL statements of the table that matches schema -pattern: "test_*" and table-pattern: "t_*" to the downstream test.t.
- rule-2 is used to migrate DDL statements of the schema that matches schema-pattern: "test_*", such as create/drop schema xx.

Note:

- If the downstream schema: test already exists and will not be deleted, you can omit rule-2.
- If the downstream schema: test does not exist and only rule-1 is configured, then it reports the schema test doesn't exist error during migration.

```
rule-1:
  schema-pattern: "test_*"
  table-pattern: "t_*"
  target-schema: "test"
  target-table: "t"
rule-2:
  schema-pattern: "test_*"
  target-schema: "test"
```

2.1.1.3.2 Merge sharded schemas

Assuming in the scenario of sharded schemas, you want to migrate the `test_{1,2,3...}` ↪ `.t_{1,2,3...}` tables in the two upstream MySQL instances to the `test.t_{1,2,3...}` tables in the downstream TiDB instance.

To migrate the upstream schemas to the downstream `test.t_[1,2,3]`, you only need to create one routing rule.

```
rule-1:
  schema-pattern: "test_*"
  target-schema: "test"
```

2.1.1.3.3 Incorrect table routing

Assuming that the following two routing rules are configured and `test_1_bak.t_1_bak` matches both `rule-1` and `rule-2`, an error is reported because the table routing configuration violates the number limitation.

```
rule-1:
  schema-pattern: "test_*"
  table-pattern: "t_*"
  target-schema: "test"
  target-table: "t"
rule-2:
  schema-pattern: "test_1_bak"
  table-pattern: "t_1_bak"
  target-schema: "test"
  target-table: "t_bak"
```

2.1.2 Block and allow table lists

The block and allow lists filtering rule of the upstream database instance tables is similar to MySQL replication-rules-db/tables, which can be used to filter or only migrate all operations of some databases or some tables.

2.1.2.1 Parameter configuration

```
block-allow-list:          # Use black-white-list if the DM's version <= v1
   ↪ .0.6.
  rule-1:
    do-dbs: ["test*"]      # Starting with characters other than "~"
       ↪ indicates that it is a wildcard;
                           # v1.0.5 or later versions support the regular
                              ↪ expression rules.
     do-tables:
    - db-name: "test[123]" # Matches test1, test2, and test3.
      tbl-name: "t[1-5]"   # Matches t1, t2, t3, t4, and t5.
    - db-name: "test"
      tbl-name: "t"
  rule-2:
    do-dbs: ["~^test.*"]   # Starting with "~" indicates that it is a
       ↪ regular expression.
     ignore-dbs: ["mysql"]
    do-tables:
    - db-name: "~^test.*"
      tbl-name: "~^t.*"
    - db-name: "test"
      tbl-name: "t"
    ignore-tables:
    - db-name: "test"
      tbl-name: "log"
```

2.1.2.2 Parameter explanation

- `do-dbs`: allow lists of the schemas to be replicated, similar to `replicate-do-db` in MySQL
- `ignore-dbs`: block lists of the schemas to be replicated, similar to `replicate-ignore-db` in MySQL
- `do-tables`: allow lists of the tables to be replicated, similar to `replicate-do-table` in MySQL
- `ignore-tables`: block lists of the tables to be replicated, similar to `replicate-ignore-table` in MySQl

If a value of the above parameters starts with the `~` character, the subsequent characters of this value are treated as a regular expression. You can use this parameter to match schema or table names.

2.1.2.3 Filtering process

The filtering rules corresponding to `do-dbs` and `ignore-dbs` are similar to the Evaluation of Database-Level Replication and Binary Logging Options in MySQL. The filtering rules corresponding to `do-tables` and `ignore-tables` are similar to the Evaluation of Table-Level Replication Options in MySQL.

Note:

In DM and in MySQL, the allow and block lists filtering rules are different in the following ways:

- In MySQL, `replicate-wild-do-table` and `replicate-wild-ignore-table` support wildcard characters. In DM, some parameter values directly supports regular expressions that start with the `~` character.
- DM currently only supports binlogs in the `ROW` format, and does not support those in the `STATEMENT` or `MIXED` format. Therefore, the filtering rules in DM correspond to those in the `ROW` format in MySQL.
- MySQL determines a DDL statement only by the database name explicitly specified in the `USE` section of the statement. DM determines a statement first based on the database name section in the DDL statement. If the DDL statement does not contain such section, DM determines the statement by the `USE` section. Suppose that the SQL statement to be determined is `USE test_db_2; CREATE TABLE test_db_1.test_table (c1 INT PRIMARY KEY)`; that `replicate-do-db=test_db_1` is configured in MySQL and `do-dbs: ["test_db_1"]` is configured in DM. Then this rule only applies to DM and not to MySQL.

The filtering process is as follows:

1. Filter at the schema level:
    - If `do-dbs` is not empty, judge whether a matched schema exists in `do-dbs`.
        - If yes, continue to filter at the table level.
        - If not, filter `test.t`.
    - If `do-dbs` is empty and `ignore-dbs` is not empty, judge whether a matched schema exits in `ignore-dbs`.
        - If yes, filter `test.t`.
        - If not, continue to filter at the table level.
    - If both `do-dbs` and `ignore-dbs` are empty, continue to filter at the table level.
2. Filter at the table level:

1. If `do-tables` is not empty, judge whether a matched table exists in `do-tables`.
    - If yes, migrate `test.t`.
    - If not, filter `test.t`.
2. If `ignore-tables` is not empty, judge whether a matched table exists in `ignore-tables`.
    - If yes, filter `test.t`.
    - If not, migrate `test.t`.
3. If both `do-tables` and `ignore-tables` are empty, migrate `test.t`.

> **Note:**
>
> To judge whether the schema `test` is filtered, you only need to filter at the schema level.

2.1.2.4 Usage example

Assume that the upstream MySQL instances include the following tables:

```
`logs`.`messages_2016`
`logs`.`messages_2017`
`logs`.`messages_2018`
`forum`.`users`
`forum`.`messages`
`forum_backup_2016`.`messages`
`forum_backup_2017`.`messages`
`forum_backup_2018`.`messages`
```

The configuration is as follows:

```
block-allow-list: # Use black-white-list if the DM's version <= v1.0.6.
  bw-rule:
    do-dbs: ["forum_backup_2018", "forum"]
    ignore-dbs: ["~^forum_backup_"]
    do-tables:
    - db-name: "logs"
      tbl-name: "~_2018$"
    - db-name: "~^forum.*"
       tbl-name: "messages"
    ignore-tables:
    - db-name: "~.*"
       tbl-name: "^messages.*"
```

After using the `bw-rule` rule:

Table	Whether to filter	Why filter
`logs` ↪ `.messages_2016`	Yes	The schema `logs` fails to match any `do-dbs`.
`logs` ↪ `.messages_2017`	Yes	The schema `logs` fails to match any `do-dbs`.
`logs` ↪ `.messages_2018`	Yes	The schema `logs` fails to match any `do-dbs`.
`forum_backup_2016` ↪ `.messages`	Yes	The schema `forum_backup_2016` ↪ fails to match any `do-dbs`.
`forum_backup_2017` ↪ `.messages`	Yes	The schema `forum_backup_2017` ↪ fails to match any `do-dbs`.
`forum` ↪ `.users`	Yes	1. The schema `forum` matches `do-dbs` and continues to filter at the table level. 2. The schema and table fail to match any of `do-tables` and `ignore-tables` and `do-tables` is not empty.

Table	Whether to filter	Why filter
`forum` ↪ `.messages` ↪	No	1. The schema `forum` matches `do-dbs` and continues to filter at the table level. 2. The table `messages` is in the `db-name: "~^ ↪ forum.*", ↪ tbl-name: ↪ "messages"` of `do-tables`.
`forum_backup_2018` ↪ `.messages` ↪	No	The schema `forum_backup_2018` ↪ matches `do-dbs` and continues to filter at the table level. 2. The schema and table match the `db-name: "~^ ↪ forum.*", ↪ tbl-name: ↪ "messages"` of `do-tables`.

2.1.3 Binlog event filter

Binlog event filter is a more fine-grained filtering rule than the block and allow lists filtering rule. You can use statements like `INSERT` or `TRUNCATE TABLE` to specify the binlog events of `schema/table` that you need to migrate or filter out.

> **Note:**
>
> If a same table matches multiple rules, these rules are applied in order and the block list has priority over the allow list. This means if both the `Ignore` and `Do` rules are applied to a single table, the `Ignore` rule takes effect.

2.1.3.1 Parameter configuration

```
filters:
  rule-1:
    schema-pattern: "test_*"
     table-pattern: "t_*"
     events: ["truncate table", "drop table"]
    sql-pattern: ["^DROP\\s+PROCEDURE", "^CREATE\\s+PROCEDURE"]
     action: Ignore
```

2.1.3.2 Parameter explanation

- schema-pattern/table-pattern: the binlog events or DDL SQL statements of upstream MySQL or MariaDB instance tables that match `schema-pattern/table-pattern` are filtered by the rules below.
- `events`: the binlog event array.

Events	Type	Description
`all`		Includes all the events below
`all dml`		Includes all DML events below
`all ddl`		Includes all DDL events below
`none`		Includes none of the events below
`none ddl`		Includes none of the DDL events below
`none dml`		Includes none of the DML events below
`insert`	DML	The `INSERT` DML event
`update`	DML	The `UPDATE` DML event
`delete`	DML	The `DELETE` DML event
`create database`	DDL	The `CREATE DATABASE` DDL event
`drop database`	DDL	The `DROP DATABASE` DDL event
`create table`	DDL	The `CREATE TABLE` DDL event
`create index`	DDL	The `CREATE INDEX` DDL event
`drop table`	DDL	The `DROP TABLE` DDL event
`truncate table`	DDL	The `TRUNCATE TABLE` DDL event
`rename table`	DDL	The `RENAME TABLE` DDL event
`drop index`	DDL	The `DROP INDEX` DDL event
`alter table`	DDL	The `ALTER TABLE` DDL event

- `sql-pattern`: it is used to filter specified DDL SQL statements. The matching rule supports using a regular expression. For example, `"^DROP\\s+PROCEDURE"`.

- `action`: the string (`Do`/`Ignore`). Based on the following rules, it judges whether to filter. If either of the two rules is satisfied, the binlog will be filtered; otherwise, the binlog will not be filtered.
  - `Do`: the allow list. The binlog will be filtered in either of the following two conditions:
    * The type of the event is not in the `event` list of the rule.
    * The SQL statement of the event cannot be matched by `sql-pattern` of the rule.
  - `Ignore`: the block list. The binlog will be filtered in either of the following two conditions:
    * The type of the event is in the `event` list of the rule.
    * The SQL statement of the event can be matched by `sql-pattern` of the rule.

2.1.3.3 Usage examples

This sections shows the usage examples in the scenario of sharding (sharded schemas and tables).

2.1.3.3.1 Filter all sharding deletion operations

To filter out all deletion operations, configure the following two filtering rules:

- `filter-table-rule` filters out the `truncate table`, `drop table` and `delete statement` operations of all tables that match the `test_*.t_*` pattern.
- `filter-schema-rule` filters out the `drop database` operation of all schemas that match the `test_*` pattern.

```
filters:
  filter-table-rule:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    events: ["truncate table", "drop table", "delete"]
    action: Ignore
  filter-schema-rule:
    schema-pattern: "test_*"
    events: ["drop database"]
    action: Ignore
```

2.1.3.3.2 Only migrate sharding DML statements

To only migrate sharding DML statements, configure the following two filtering rules:

- `do-table-rule` only migrates the `create table`, `insert`, `update` and `delete` statements of all tables that match the `test_*.t_*` pattern.
- `do-schema-rule` only migrates the `create database` statement of all schemas that match the `test_*` pattern.

> **Note:**
>
> The reason why the `create database/table` statement is migrated is that you can migrate DML statements only after the schema and table are created.

```
filters:
  do-table-rule:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    events: ["create table", "all dml"]
    action: Do
  do-schema-rule:
    schema-pattern: "test_*"
    events: ["create database"]
    action: Do
```

2.1.3.3.3 Filter out the SQL statements that TiDB does not support

To filter out the `PROCEDURE` statements that TiDB does not support, configure the following `filter-procedure-rule`:

```
filters:
  filter-procedure-rule:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    sql-pattern: ["^DROP\\s+PROCEDURE", "^CREATE\\s+PROCEDURE"]
    action: Ignore
```

`filter-procedure-rule` filters out the `^CREATE\\s+PROCEDURE` and `^DROP\\s+PROCEDURE` statements of all tables that match the `test_*.t_*` pattern.

2.1.3.3.4 Filter out the SQL statements that the TiDB parser does not support

For the SQL statements that the TiDB parser does not support, DM cannot parse them and get the `schema/table` information. So you must use the global filtering rule: `schema-pattern: "*"`.

Note:

To avoid unexpectedly filtering out data that need to be migrated, you must configure the global filtering rule as strictly as possible.

To filter out the `PARTITION` statements that the TiDB parser does not support, configure the following filtering rule:

```
filters:
  filter-partition-rule:
    schema-pattern: "*"
    sql-pattern: ["ALTER\\s+TABLE[\\s\\S]*ADD\\s+PARTITION", "ALTER\\s+TABLE
       ↪ [\\s\\S]*DROP\\s+PARTITION"]
    action: Ignore
```

2.1.4 Column mapping

Note:

The column mapping is not recommended as the primary solution due to its usage restrictions. The preferable solution is handling conflicts of auto-increment primary key.

The column mapping feature supports modifying the value of table columns. You can execute different modification operations on the specified column according to different expressions. Currently, only the built-in expressions provided by DM are supported.

Note:

- It does not support modifying the column type and the table schema.
- It does not support configuring multiple different column mapping rules for a same table.

2.1.4.1 Parameter configuration

```
column-mappings:
  rule-1:
     schema-pattern: "test_*"
     table-pattern: "t_*"
     expression: "partition id"
     source-column: "id"
     target-column: "id"
     arguments: ["1", "test", "t", "_"]
  rule-2:
     schema-pattern: "test_*"
     table-pattern: "t_*"
     expression: "partition id"
     source-column: "id"
     target-column: "id"
     arguments: ["2", "test", "t", "_"]
```

2.1.4.2 Parameter explanation

- schema-pattern/table-pattern: to execute column value modifying operations on the upstream MySQL or MariaDB instance tables that match the `schema-pattern` ↪ /`table-pattern` filtering rule.
- `source-column`, `target-column`: to modify the value of the `source-column` column according to specified `expression` and assign the new value to `target-column`.
- `expression`: the expression used to modify data. Currently, only the `partition id` built-in expression is supported.

2.1.4.2.1 The `partition id` expression

`partition id` is used to resolve the conflicts of auto-increment primary keys of sharded tables.

`partition id` restrictions

Note the following restrictions:

- The `partition id` expression only supports the bigint type of auto-increment primary key.
- If the `schema prefix` is not empty, the schema name format must be `schema prefix` ↪ or `schema prefix + separator + number (the schema ID)`. For example, it supports `s` and `s_1`, but does not support `s_a`.
- If the `table prefix` is not empty, the table name format must be `table prefix` or `table prefix + separator + number (the table ID)`.
- If the schema/table name does not contain the `··· + separator + number` part, the corresponding ID is considered as 0.

- Restrictions on sharding size:
  - It supports 16 MySQL or MariaDB instances at most (Requirement: 0 <= instance ID <= 15).
  - Each instance supports 128 schemas at most (Requirement: 0 <= schema ID <= 127).
  - Each schema of each instance supports 256 tables at most (Requirement: 0 <= table ID <= 255).
  - The range of the mapped column should meet the requirement: 0 <= ID <= 17592186044415.
  - The `{instance ID, schema ID, table ID}` group must be unique.

`partition id` arguments configuration

Configure the following three or four arguments in order:

- `instance_id`: the ID of the upstream sharded MySQL or MariaDB instance (0 <= instance ID <= 15)
- `schema prefix`: used to parse the schema name and get the `schema ID`
- `table prefix`: used to parse the table name and get the `table ID`
- The separator: used to separate between the prefix and the IDs, and can be omitted to use an empty string as separator

Any of `instance_id`, `schema prefix` and `table prefix` can be set to an empty string (`""`) to indicate that the corresponding parts will not be encoded into the partition ID.

`partition id` expression rules

`partition id` fills the beginning bit of the auto-increment primary key ID with the argument number, and computes an int64 (MySQL bigint) type of value. The specific rules are as follows:

instance_id	schema prefix	table prefix	Encoding
defined	defined	defined	[S: 1 bit] [I: 4 bits] [D: 7 bits] [T: 8 bits] [P: 44 bits]
empty	defined	defined	[S: 1 bit] [D: 7 bits] [T: 8 bits] [P: 48 bits]
defined	empty	defined	[S: 1 bit] [I: 4 bits] [T: 8 bits] [P: 51 bits]

instance_id	schema prefix	table prefix	Encoding
defined	defined	empty	[S: 1 bit] [I: 4 bits] [D: 7 bits] [P: 52 bits]
empty	empty	defined	[S: 1 bit] [T: 8 bits] [P: 55 bits]
empty	defined	empty	[S: 1 bit] [D: 7 bits] [P: 56 bits]
defined	empty	empty	[S: 1 bit] [I: 4 bits] [P: 59 bits]

- S: the sign bit, reserved
- I: the instance ID, 4 bits by default if set
- D: the schema ID, 7 bits by default if set
- T: the table ID, 8 bits by default if set
- P: the auto-increment primary key ID, occupying the rest of bits (44 bits)

2.1.4.3 Usage example

Assuming in the sharding scenario where all tables have the auto-increment primary key, you want to migrate two upstream MySQL instances `test_{1,2,3...}.t_{1,2,3...}` to the downstream TiDB instances `test.t`.

Configure the following two rules:

```
column-mappings:
  rule-1:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    expression: "partition id"
    source-column: "id"
    target-column: "id"
    arguments: ["1", "test", "t", "_"]
  rule-2:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    expression: "partition id"
    source-column: "id"
    target-column: "id"
    arguments: ["2", "test", "t", "_"]
```

- The column ID of the MySQL instance 1 table `test_1.t_1` is converted from `1` to `1 << (64-1-4)| 1 << (64-1-4 -7)| 1 << 44 | 1 = 580981944116838401`.
- The row ID of the MySQL instance 2 table `test_1.t_2` is converted from `2` to `2 << (64-1-4)| 1 << (64-1-4 -7)| 2 << 44 | 2 = 1157460288606306306`.

2.1.5 Migration delay monitoring

The heartbeat feature supports calculating the real-time migration delay between each migration task and MySQL or MariaDB based on real migration data.

Note:

- The estimation accuracy of the migration delay is at the second level.
- The heartbeat related binlog will not be migrated into the downstream, which is discarded after calculating the migration delay.

2.1.5.1 System privileges

If the heartbeat feature is enabled, the upstream MySQL or MariaDB instances must provide the following privileges:

- SELECT
- INSERT
- CREATE (databases, tables)
- DELETE

2.1.5.2 Parameter configuration

In the task configuration file, enable the heartbeat feature:

```
enable-heartbeat: true
```

2.1.5.3 Principles introduction

- DM-worker creates the `dm_heartbeat` (currently unconfigurable) schema in the corresponding upstream MySQL or MariaDB.
- DM-worker creates the `heartbeat` (currently unconfigurable) table in the corresponding upstream MySQL or MariaDB.
- DM-worker uses `replace statement` to update the current `TS_primary` timestamp every second (currently unconfigurable) in the corresponding upstream MySQL or MariaDB `dm_heartbeat.heartbeat` tables.

- DM-worker updates the `TS_secondary_task` migration time after each migration task obtains the `dm_heartbeat.heartbeat` binlog.
- DM-worker queries the current `TS_primary` timestamp in the corresponding upstream MySQL or MariaDB `dm_heartbeat.heartbeat` tables every 10 seconds, and calculates `task_lag` = `TS_primary` - `TS_secondary_task` for each task.

See the `replicate lag` in the binlog replication processing unit of DM monitoring metrics.

2.2 DM online-ddl-scheme

This document introduces the `online-ddl-scheme` feature of DM.

2.2.1 Overview

DDL statements are always used in the database applications. MySQL 5.6 and later versions support online-ddl, but there are limitations for usage. For example, to acquire the MDL lock, some DDLs still need to be copied. In production scenario, the table lock during DDL execution can block the reads or writes to and from the database to a certain extent.

By using gh-ost and pt-osc, DDLs can be executed on the MySQL database more gracefully, and the impact on reads and writes is reduced as much as possible.

TiDB is implemented based on the online asynchronous schema change algorithm of Google F1. It does not block reads and writes during the DDL execution. Therefore, the large amount of intermediate table data and binlog events generated by gh-ost and pt-osc in the process of online-schema-change is not needed during the replication from MySQL to TiDB.

For Data Migration (DM), which supports the data replication from MySQL to TiDB, the online-ddl-scheme feature is to perform special processing on the above two online-schema-change tools (gh-ost and pt-osc). This way, the required DDL replication can be completed more rapidly.

2.2.2 Configuration

In the task configuration file, `online-ddl-scheme` is at the same level of `name`. For example:

```
## ----------- Global configuration -----------
### ********* Basic configuration *********
name: test                  # The name of the task. Should be globally
   ↪ unique.
task-mode: all              # The task mode. Can be set to `full`/`
   ↪ incremental`/`all`.
is-sharding: true           # Whether it is a task to merge shards.
```

```
meta-schema: "dm_meta"      # The downstream database that stores the `meta`
    ↪  information.
remove-meta: false          # Whether to remove the `meta` information (`
    ↪ checkpoint` and `onlineddl`) corresponding to the task name before
    ↪ starting the replication task.
enable-heartbeat: false     # Whether to enable the heartbeat feature.
online-ddl-scheme: "gh-ost" # Only "gh-ost" and "pt" are currently supported
    ↪ .
target-database:            # Configuration of the downstream database
    ↪ instance.
  host: "192.168.0.1"
  port: 4000
  user: "root"
  password: ""              # The password must be encrypted using dmctl if
     ↪ it is not empty.
```

For the advanced configuration and the description of each configuration parameter, refer to DM advanced task configuration file template.

2.2.3 online-schema-change: gh-ost

When gh-ost implements online-schema-change, 3 types of tables are created:

- gho: used to apply DDLs. When the data is fully replicated and the gho table is consistent with the origin table, the origin table is replaced by renaming.
- ghc: used to store information that is related to online-schema-change.
- del: created by renaming the origin table.

In the process of replication, DM divides the above tables into 3 categories:

- ghostTable: `\_\*\_gho`
- trashTable: `\_\*\_ghc`, `\_\*\_del`
- realTable: the origin table that executes online-ddl.

The SQL statements mostly used by gh-ost and the corresponding operation of DM are as follows:

1. Create the `_ghc` table:

```
Create /* gh-ost */ table `test`.`_test4_ghc` (
                    id bigint auto_increment,
                    last_update timestamp not null DEFAULT
                       ↪ CURRENT_TIMESTAMP ON UPDATE
                       ↪ CURRENT_TIMESTAMP,
```

```
                    hint varchar(64) charset ascii not null,
                    value varchar(4096) charset ascii not null,
                    primary key(id),
                    unique key hint_uidx(hint)
            ) auto_increment=256 ;
```

DM does not create the `_test4_ghc` table.

2. Create the `_gho` table:

```
Create /* gh-ost */ table `test`.`_test4_gho` like `test`.`test4` ;
```

DM does not create the `_test4_gho` table. DM deletes the `dm_meta.{task_name}\_onlineddl` record in the downstream according to `ghost_schema`, `ghost_table`, and the `server_id` of `dm_worker`, and clears the related information in memory.

```
DELETE FROM dm_meta.{task_name}_onlineddl WHERE id = {server_id} and
    ↪ ghost_schema = {ghost_schema} and ghost_table = {ghost_table};
```

3. Apply the DDL that needs to be executed in the `_gho` table:

```
Alter /* gh-ost */ table `test`.`_test4_gho` add column cl1 varchar
    ↪ (20) not null ;
```

DM does not perform the DDL operation of `_test4_gho`. It records this DDL in `dm_meta.{task_name}\_onlineddl` and memory.

```
REPLACE INTO dm_meta.{task_name}_onlineddl (id, ghost_schema ,
    ↪ ghost_table , ddls) VALUES (......);
```

4. Write data to the `_ghc` table, and replicate the origin table data to the `_gho` table:

```
Insert /* gh-ost */ into `test`.`_test4_ghc` values (......);
Insert /* gh-ost `test`.`test4` */ ignore into `test`.`_test4_gho` (`
    ↪ id`, `date`, `account_id`, `conversion_price`, `
    ↪ ocpc_matched_conversions`, `ad_cost`, `cl2`)
  (select `id`, `date`, `account_id`, `conversion_price`, `
      ↪ ocpc_matched_conversions`, `ad_cost`, `cl2` from `test`.`test4`
      ↪  force index (`PRIMARY`)
    where (((`id` > _binary'1') or ((`id` = _binary'1'))) and ((`id` <
        ↪ _binary'2') or ((`id` = _binary'2')))) lock in share mode
  )   ;
```

DM does not execute DML statements that are not for **realtable**.

5. After the replication is completed, both the origin table and `_gho` table are renamed, and the online DDL operation is completed:

```
Rename /* gh-ost */ table `test`.`test4` to `test`.`_test4_del`, `test
    ↪ `.`_test4_gho` to `test`.`test4`;
```

DM performs the following two operations:

- DM splits the above `rename` operation into two SQL statements.

```
rename test.test4 to test._test4_del;
rename test._test4_gho to test.test4;
```

- DM does not execute `rename to _test4_del`. When executing `rename ↪ ghost_table to origin table`, DM takes the following steps:
  - Read the DDL recorded in memory in Step 3
  - Replace `ghost_table` and `ghost_schema` with `origin_table` and its corresponding schema
  - Execute the DDL that has been replaced

```
alter table test._test4_gho add column c11 varchar(20) not null;
-- Replaced with:
alter table test.test4 add column c11 varchar(20) not null;
```

Note:

The specific SQL statements of gh-ost vary with the parameters used in the execution. This document only lists the major SQL statements. For more details, refer to the gh-ost documentation.

2.2.4 online-schema-change: pt

When pt-osc implements online-schema-change, 2 types of tables are created:

- `new`: used to apply DDL. When the data is fully replicated and the `new` table is consistent with the origin table, the origin table is replaced by renaming.
- `old`: created by renaming the origin table.
- 3 kinds of Trigger: `pt_osc\_\*\_ins`, `pt_osc\_\*\_upd`, `pt_osc\_\*\_del`. In the process of pt_osc, the new data generated by the origin table is replicated to `new` by the Trigger.

In the process of replication, DM divides the above tables into 3 categories:

- ghostTable: `\_\*\_new`

- trashTable: `\_\*\_old`
- realTable: the origin table that executes online-ddl.

The SQL statements mostly used by pt-osc and the corresponding operation of DM are as follows:

1. Create the `_new` table:

```
CREATE TABLE `test`.`_test4_new` ( id int(11) NOT NULL AUTO_INCREMENT,
date date DEFAULT NULL, account_id bigint(20) DEFAULT NULL,
    ↪ conversion_price decimal(20,3) DEFAULT NULL,
    ↪ ocpc_matched_conversions bigint(20) DEFAULT NULL, ad_cost
    ↪ decimal(20,3) DEFAULT NULL,cl2 varchar(20) COLLATE utf8mb4_bin
    ↪ NOT NULL,cl1 varchar(20) COLLATE utf8mb4_bin NOT NULL,PRIMARY
    ↪ KEY (id) ) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=
    ↪ utf8mb4 COLLATE=utf8mb4_bin ;
```

DM does not create the `_test4_new` table. DM deletes the `dm_meta.{task_name}\_onlineddl` record in the downstream according to `ghost_schema`, `ghost_table`, and the `server_id` of `dm_worker`, and clears the related information in memory.

```
DELETE FROM dm_meta.{task_name}_onlineddl WHERE id = {server_id} and
    ↪ ghost_schema = {ghost_schema} and ghost_table = {ghost_table};
```

2. Execute DDL in the `_new` table:

```
ALTER TABLE `test`.`_test4_new` add column c3 int;
```

DM does not perform the DDL operation of `_test4_new`. Instead, it records this DDL in `dm_meta.{task_name}\_onlineddl` and memory.

```
REPLACE INTO dm_meta.{task_name}_onlineddl (id, ghost_schema ,
    ↪ ghost_table , ddls) VALUES (......);
```

3. Create 3 Triggers used for data replication:

```
CREATE TRIGGER `pt_osc_test_test4_del` AFTER DELETE ON `test`.`test4`
    ↪ ...... ;
CREATE TRIGGER `pt_osc_test_test4_upd` AFTER UPDATE ON `test`.`test4`
    ↪ ...... ;
CREATE TRIGGER `pt_osc_test_test4_ins` AFTER INSERT ON `test`.`test4`
    ↪ ...... ;
```

DM does not execute Trigger operations that are not supported in TiDB.

4. Replicate the origin table data to the `_new` table:

```
INSERT LOW_PRIORITY IGNORE INTO `test`.`_test4_new` (`id`, `date`, `
    ↪ account_id`, `conversion_price`, `ocpc_matched_conversions`, `
    ↪ ad_cost`, `cl2`, `cl1`) SELECT `id`, `date`, `account_id`, `
    ↪ conversion_price`, `ocpc_matched_conversions`, `ad_cost`, `cl2`,
    ↪ `cl1` FROM `test`.`test4` LOCK IN SHARE MODE /*pt-online-schema-
    ↪ change 3227 copy table*/
```

DM does not execute the DML statements that are not for **realtable**.

5. After the data replication is completed, the origin table and `_new` table are renamed, and the online DDL operation is completed:

```
RENAME TABLE `test`.`test4` TO `test`.`_test4_old`, `test`.`_test4_new`
    ↪  TO `test`.`test4`
```

DM performs the following two operations:

- DM splits the above `rename` operation into two SQL statements:

  ```
  sql  rename test.test4 to test._test4_old; rename test._test4_new
  ↪ to test.test4;
  ```

- DM does not execute `rename to _test4_old`. When executing `rename ↪ ghost_table to origin table`, DM takes the following steps:
  - Read the DDL recorded in memory in Step 2
  - Replace `ghost_table` and `ghost_schema` with `origin_table` and its corresponding schema
  - Execute the DDL that has been replaced

  ```
  ALTER TABLE `test`.`_test4_new` add column c3 int;
  -- Replaced with:
  ALTER TABLE `test`.`test4` add column c3 int;
  ```

6. Delete the `_old` table and 3 Triggers of the online DDL operation:

```
DROP TABLE IF EXISTS `test`.`_test4_old`;
DROP TRIGGER IF EXISTS `pt_osc_test_test4_del` AFTER DELETE ON `test
    ↪ `.`test4` ..... ;
DROP TRIGGER IF EXISTS `pt_osc_test_test4_upd` AFTER UPDATE ON `test
    ↪ `.`test4` ..... ;
DROP TRIGGER IF EXISTS `pt_osc_test_test4_ins` AFTER INSERT ON `test
    ↪ `.`test4` ..... ;
```

DM does not delete `_test4_old` and Triggers.

Note:

The specific SQL statements of pt-osc vary with the parameters used in the execution. This document only lists the major SQL statements. For more details, refer to the pt-osc documentation.

2.3 Sharding Support

2.3.1 Merge and migrate Data from Sharded Tables

This document introduces the sharding support feature provided by Data Migration (DM). This feature allows you to merge and migrate the data of tables with the same table schema in the upstream MySQL or MariaDB instances into one same table in the downstream TiDB. It supports not only migrating the upstream DML statements, but also coordinating to migrate the table schema change using DDL statements in multiple upstream sharded tables.

Note:

To merge and migrate data from the sharded tables, you must configure the `is-sharding: true` item in the task configuration file.

2.3.1.1 Restrictions

DM has the following sharding DDL usage restrictions:

- In a logical **sharding group** (composed of all sharded tables that need to be merged and migrated into one same downstream table), the same DDL statements must be executed in the same order in all upstream sharded tables (the schema name and the table name can be different), and the next DDL statement cannot be executed unless the current DDL operation is completely finished.
  - For example, if you add `column A` to `table_1` before you add `column B`, then you cannot add `column B` to `table_2` before you add `column A`. Executing the DDL statements in a different order is not supported.
- For each sharding group, it is recommended to use one independent task to perform the migration.

  - If multiple sharding groups exist in a task, you cannot start to execute the DDL statements in other sharding groups until the DDL statements in one sharding group has been migrated successfully.

- In a sharding group, the corresponding DDL statements should be executed in all upstream sharded tables.
  - For example, if DDL statements are not executed on one or more upstream sharded tables corresponding to `DM-worker-2`, then other DM-workers that have executed the DDL statements pause their migration task and wait for `DM-worker-2` to receive the upstream DDL statements.
- The sharding group migration task does not support `DROP DATABASE`/`DROP TABLE`.
  - The sync unit in DM-worker automatically ignores the `DROP DATABASE`/`DROP TABLE` statement of upstream sharded tables.
- The sharding group migration task supports `RENAME TABLE`, but with the following limitations (Online DDL is supported in another solution):
  - A table can only be renamed to a new name that is not used by any other table.
  - A single `RENAME TABLE` statement can only involve a single `RENAME` operation.
- The table schema of each sharded table must be the same at the starting point of the incremental replication task, so as to make sure the DML statements of different sharded tables can be migrated into the downstream with a definite table schema, and the subsequent sharding DDL statements can be correctly matched and migrated.
- If you need to change the table routing rule, you have to wait for the migration of all sharding DDL statements to complete.
  - During the migration of sharding DDL statements, an error is reported if you use `dmctl` to change `router-rules`.
- If you need to `CREATE` a new table to a sharding group where DDL statements are being executed, you have to make sure that the table schema is the same as the newly modified table schema.
  - For example, both the original `table_1` and `table_2` have two columns (a, b) initially, and have three columns (a, b, c) after the sharding DDL operation, so after the migration the newly created table should also have three columns (a, b, c).
- Because the DM-worker that has received the DDL statements will pause the task to wait for other DM-workers to receive their DDL statements, the delay of data migration will be increased.

2.3.1.2 Background

Currently, DM uses the binlog in the `ROW` format to perform the migration task. The binlog does not contain the table schema information. When you use the `ROW` binlog to

migrate data, if you have not migrated multiple upstream tables into the same downstream table, then there only exist DDL operations of one upstream table that can update the table schema of the downstream table. The `ROW` binlog can be considered to have the nature of self-description. During the migration process, the DML statements can be constructed accordingly with the column values and the downstream table schema.

However, in the process of merging and migrating sharded tables, if DDL statements are executed on the upstream tables to modify the table schema, then you need to perform extra operations to migrate the DDL statements so as to avoid the inconsistency between the DML statements produced by the column values and the actual downstream table schema.

Here is a simple example:

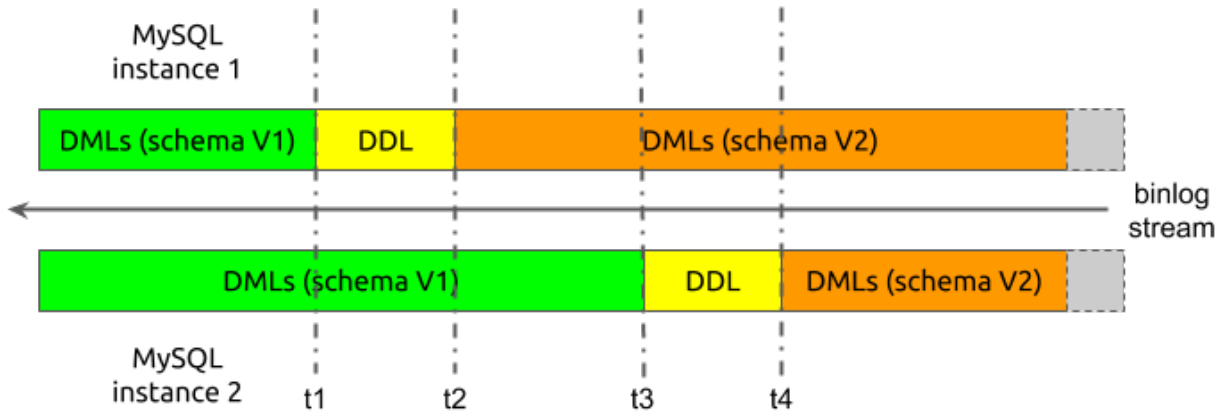


Figure 2: shard-ddl-example-1

In the above example, the merging process is simplified, where only two MySQL instances exist in the upstream and each instance has only one table. When the migration begins, the table schema version of two sharded tables is marked as `schema V1`, and the table schema version after executing DDL statements is marked as `schema V2`.

Now assume that in the migration process, the binlog data received from the two upstream sharded tables has the following time sequence:

1. When the migration begins, the sync unit in DM-worker receives the DML events of `schema V1` from the two sharded tables.
2. At `t1`, the sharding DDL events from instance 1 are received.
3. From `t2` on, the sync unit receives the DML events of `schema V2` from instance 1; but from instance 2, it still receives the DML events of `schema V1`.
4. At `t3`, the sharding DDL events from instance 2 are received.
5. From `t4` on, the sync unit receives the DML events of `schema V2` from instance 2 as well.

Assume that the DDL statements of sharded tables are not processed during the migration process. After DDL statements of instance 1 are migrated to the downstream, the downstream table schema is changed to `schema V2`. But for instance 2, the sync unit in DM-worker is still receiving DML events of `schema V1` from `t2` to `t3`. Therefore, when the DML statements of `schema V1` are migrated to the downstream, the inconsistency between the DML statements and the table schema can cause errors and the data cannot be migrated successfully.

2.3.1.3 Principles

This section shows how DM migrates DDL statements in the process of merging sharded tables based on the above example in the background section.

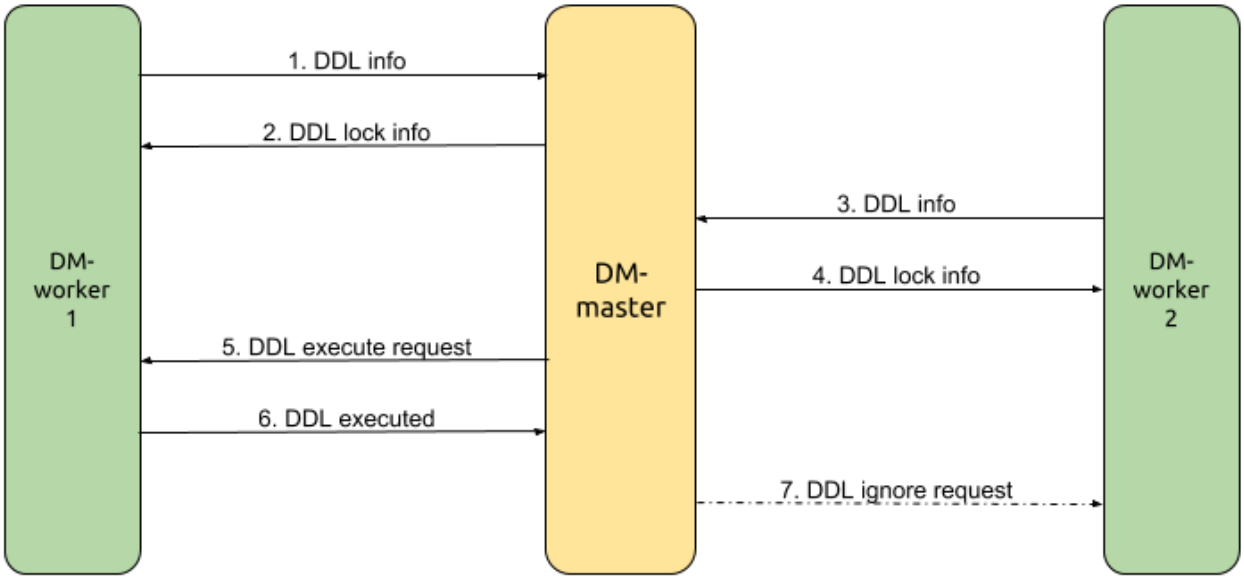


Figure 3: shard-ddl-flow

In this example, `DM-worker-1` migrates the data from MySQL instance 1 and `DM-worker-2` migrates the data from MySQL instance 2. `DM-master` coordinates the DDL migration among multiple DM-workers. Starting from `DM-worker-1` receiving the DDL statements, the DDL migration process is simplified as follows:

1. `DM-worker-1` receives the DDL statement from MySQL instance 1 at `t1`, pauses the data migration of the corresponding DDL and DML statements, and sends the DDL information to `DM-master`.
2. `DM-master` decides that the migration of this DDL statement needs to be coordinated based on the received DDL information, creates a lock for this DDL statement, sends the DDL lock information back to `DM-worker-1` and marks `DM-worker-1` as the owner of this lock at the same time.

3. `DM-worker-2` continues migrating the DML statement until it receives the DDL statement from MySQL instance 2 at `t3`, pauses the data migration of this DDL statement, and sends the DDL information to `DM-master`.
4. `DM-master` decides that the lock of this DDL statement already exists based on the received DDL information, and sends the lock information directly to `DM-worker-2`.
5. Based on the configuration information when the task is started, the sharded table information in the upstream MySQL instances, and the deployment topology information, `DM-master` decides that it has received this DDL statement of all upstream sharded tables to be merged, and requests the owner of the DDL lock (`DM-worker-1`) to migrate this DDL statement to the downstream.
6. `DM-worker-1` verifies the DDL statement execution request based on the DDL lock information received at Step #2, migrates this DDL statement to the downstream, and sends the results to `DM-master`. If this operation is successful, `DM-worker-1` continues migrating the subsequent (starting from the binlog at `t2`) DML statements.
7. `DM-master` receives the response from the lock owner that the DDL is successfully executed, and requests all other DM-workers (`DM-worker-2`) that are waiting for the DDL lock to ignore this DDL statement and then continue to migrate the subsequent (starting from the binlog at `t4`) DML statements.

The characteristics of DM handling the sharding DDL migration among multiple DM-workers can be concluded as follows:

- Based on the task configuration and DM cluster deployment topology information, a logical sharding group is built in `DM-master` to coordinate DDL migration. The group members are DM-workers that handle each sub-task divided from the migration task).
- After receiving the DDL statement from the binlog event, each DM-worker sends the DDL information to `DM-master`.
- `DM-master` creates or updates the DDL lock based on the DDL information received from each DM-worker and the sharding group information.
- If all members of the sharding group receive a same specific DDL statement, this indicates that all DML statements before the DDL execution on the upstream sharded tables have been completely migrated, and this DDL statement can be executed. Then DM can continue to migrate the subsequent DML statements.
- After being converted by the table router, the DDL statement of the upstream sharded tables must be consistent with the DDL statement to be executed in the downstream. Therefore, this DDL statement only needs to be executed once by the DDL owner and all other DM-workers can ignore this DDL statement.

In the above example, only one sharded table needs to be merged in the upstream MySQL instance corresponding to each DM-worker. But in actual scenarios, there might be multiple sharded tables in multiple sharded schemas to be merged in one MySQL instance. And when this happens, it becomes more complex to coordinate the sharding DDL migration.

Assume that there are two sharded tables, namely `table_1` and `table_2`, to be merged in one MySQL instance:

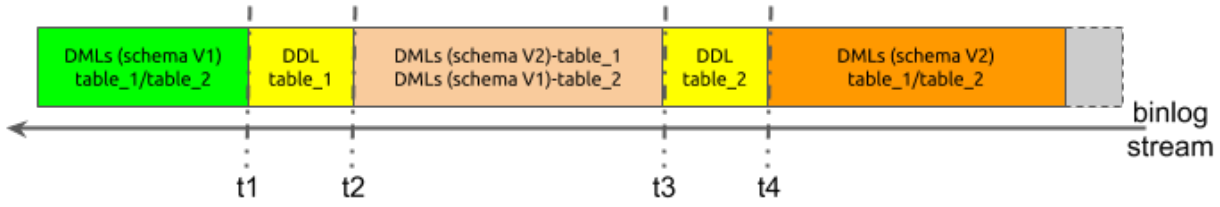


Figure 4: shard-ddl-example-2

Because data comes from the same MySQL instance, all the data is obtained from the same binlog stream. In this case, the time sequence is as follows:

1. The sync unit in DM-worker receives the DML statements of `schema V1` from both sharded tables when the migration begins.
2. At `t1`, the sync unit in DM-worker receives the DDL statements of `table_1`.
3. From `t2` to `t3`, the received data includes the DML statements of `schema V2` from `table_1` and the DML statements of `schema V1` from `table_2`.
4. At `t3`, the sync unit in DM-worker receives the DDL statements of `table_2`.
5. From `t4` on, the sync unit in DM-worker receives the DML statements of `schema V2` from both tables.

If the DDL statements are not processed particularly during the data migration, when the DDL statement of `table_1` is migrated to the downstream and changes the downstream table schema, the DML statement of `schema V1` from `table_2` cannot be migrated successfully. Therefore, within a single DM-worker, a logical sharding group similar to that within `DM-master` is created, except that members of this group are different sharded tables in the same upstream MySQL instance.

But when a DM-worker coordinates the migration of the sharding group within itself, it is not totally the same as that performed by `DM-master`. The reasons are as follows:

- When the DM-worker receives the DDL statement of `table_1`, it cannot pause the migration and needs to continue parsing the binlog to get the subsequent DDL statements of `table_2`. This means it needs to continue parsing between `t2` and `t3`.
- During the binlog parsing process between `t2` and `t3`, the DML statements of `schema V2` from `table_1` cannot be migrated to the downstream until the sharding DDL statement is migrated and successfully executed.

In DM, the simplified migration process of sharding DDL statements within the DM worker is as follows:

1. When receiving the DDL statement of `table_1` at `t1`, the DM-worker records the DDL information and the current position of the binlog.

2. DM-worker continues parsing the binlog between `t2` and `t3`.
3. DM-worker ignores the DML statement with the `schema V2` schema that belongs to `table_1`, and migrates the DML statement with the `schema V1` schema that belongs to `table_2` to the downstream.
4. When receiving the DDL statement of `table_2` at `t3`, the DM-worker records the DDL information and the current position of the binlog.
5. Based on the information of the migration task configuration and the upstream schemas and tables, the DM-worker decides that the DDL statements of all sharded tables in the MySQL instance have been received and migrates them to the downstream to modify the downstream table schema.
6. DM-worker sets the starting point of parsing the new binlog stream to be the position saved at Step #1.
7. DM-worker resumes parsing the binlog between `t2` and `t3`.
8. DM-worker migrates the DML statement with the `schema V2` schema that belongs to `table_1` to the downstream, and ignores the DML statement with the `schema V1` schema that belongs to `table_2`.
9. After parsing the binlog position saved at Step #4, the DM-worker decides that all DML statements that have been ignored in Step #3 have been migrated to the downstream again.
10. DM-worker resumes the migration starting from the binlog position at `t4`.

You can conclude from the above analysis that DM mainly uses two-level sharding groups for coordination and control when handling migration of the sharding DDL. Here is the simplified process:

1. Each DM-worker independently coordinates the DDL statements migration for the corresponding sharding group composed of multiple sharded tables within the upstream MySQL instance.
2. After the DM-worker receives the DDL statements of all sharded tables, it sends the DDL information to `DM-master`.
3. `DM-master` coordinates the DDL migration of the sharding group composed of the DM-workers based on the received DDL information.
4. After receiving the DDL information from all DM-workers, `DM-master` requests the DDL lock owner (a specific DM-worker) to execute the DDL statement.
5. The DDL lock owner executes the DDL statement and returns the result to `DM-master` ↪ . Then the owner restarts the migration of the previously ignored DML statements during the internal coordination of DDL migration.
6. After `DM-master` confirms that the owner has successfully executed the DDL statement, it asks all other DM-workers to continue the migration.
7. All other DM-workers separately restart the migration of the previously ignored DML statements during the internal coordination of DDL migration.
8. After finishing migrating the ignored DML statements again, all DM-workers resume the normal migration process.

2.3.2 Handle Sharding DDL Locks Manually in DM

DM uses the sharding DDL lock to ensure operations are performed in the correct order. This locking mechanism resolves sharding DDL locks automatically in most cases, but you need to use the `unlock-ddl-lock` or `break-ddl-lock` command to manually handle the abnormal DDL locks in some abnormal scenarios.

Warning:

- Do not use `unlock-ddl-lock` or `break-ddl-lock` unless you are totally aware of the possible impacts brought by the command and you can accept them.
- Before manually handling the abnormal DDL locks, make sure that you have already read the DM shard merge principles.

2.3.2.1 Command

2.3.2.1.1 `show-ddl-locks`

This command queries the current DDL lock information on `DM-master`.

Command usage

```
show-ddl-locks [--worker=127.0.0.1:8262] [task-name]
```

Arguments description

- `worker`:
  - Flag; string; `--worker`; optional
  - It can be specified repeatedly multiple times.
  - If it is not specified, this command queries the lock information related to all DM-workers; if it is specified, this command queries the lock information related only to the specified DM-worker.
- `task-name`:
  - Non-flag; string; optional
  - If it is not specified, this command queries the lock information related to all tasks; if it is specified, this command queries the lock information related only to the specified task.

Example of results

```
» show-ddl-locks test
{
    "result": true,                                     # The result of the
        ↪ query for the lock information.
    "msg": "",                                          # The additional
        ↪ message for the failure to query the lock information or other
        ↪ descriptive information (for example, the lock task does not
        ↪ exist).
    "locks": [                                          # The lock information
        ↪ list on DM-master.
        {
            "ID": "test-`shard_db`.`shard_table`",      # The lock ID, which is
                ↪  made up of the current task name and the schema/table
                ↪ information corresponding to the DDL.
            "task": "test",                             # The name of the task
                ↪ to which the lock belongs.
            "owner": "127.0.0.1:8262",                  # The owner of the lock
                ↪ .
            "DDLs": [                                   # The DDL list
                ↪ corresponding to the lock.
                "USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` DROP
                    ↪ COLUMN `c2`;"
            ],
            "synced": [                                 # The list of DM-
                ↪ workers that have received all sharding DDL events in the
                ↪ corresponding MySQL instance.
                "127.0.0.1:8262"
            ],
            "unsynced": [                               # The list of DM-
                ↪ workers that have not yet received all sharding DDL events
                ↪  in the corresponding MySQL instance.
                "127.0.0.1:8263"
            ]
        }
    ]
}
```

2.3.2.1.2 `unlock-ddl-lock`

This command actively requests `DM-master` to unlock the specified DDL lock, including requesting the owner to execute the DDL statement, requesting all other DM-workers that are not the owner to skip the DDL statement, and removing the lock information on `DM-master`.

Command usage

```
unlock-ddl-lock [--worker=127.0.0.1:8262] [--owner] [--force-remove] <lock-
    ↪ ID>
```

Arguments description

- `worker`:
  - Flag; string; `--worker`; optional
  - It can be specified repeatedly multiple times.
  - If it is not specified, this command sends requests for all DM-workers (except for the owner) that are waiting for the lock to skip the DDL statement; if it is specified, this command sends requests only for the specified DM-worker to skip the DDL statement.
- `owner`:
  - Flag; string; `--owner`; optional
  - If it is not specified, this command requests for the default owner (the owner in the result of `show-ddl-locks`) to execute the DDL statement; if it is specified, this command requests for the DM-worker (the alternative of the default owner) to execute the DDL statement.
- `force-remove`:
  - Flag; boolean; `--force-remove`; optional
  - If it is not specified, this command removes the lock information only when the owner succeeds to execute the DDL statement; if it is specified, this command forcefully removes the lock information even though the owner fails to execute the DDL statement (after doing this you cannot query or operate on the lock again).
- `lock-ID`:
  - Non-flag; string; required
  - It specifies the ID of the DDL lock that needs to be unlocked (the `ID` in the result of `show-ddl-locks`).

Example of results

```
» unlock-ddl-lock test-`shard_db`.`shard_table`
{
    "result": true,                                 # The result of the
        ↪ unlocking operation.
    "msg": "",                                      # The additional
        ↪ message for the failure to unlock the lock.
```

```
    "workers": [                                    # The result list of
        ↪ the executing or skipping DDL operation of each DM-worker.
        {
            "result": true,                         # The result of the
                ↪ executing or skipping DDL operation.
            "worker": "127.0.0.1:8262",             # The DM-worker ID.
            "msg": ""                               # The reasons why the
                ↪ DM-worker failed to execute or skip the DDL statement.
        }
    ]
}
```

2.3.2.1.3 `break-ddl-lock`

This command actively asks the DM-worker to forcefully break the DDL lock that is to be unlocked, including asking the DM-worker to execute/skip the DDL and removing the DDL lock information on the DM-worker.

Command usage

```
break-ddl-lock <--worker=127.0.0.1:8262> [--remove-id] [--exec] [--skip] <
    ↪ task-name>
```

Arguments description

- `worker`:
  - Flag; string; `--worker`; required
  - It specifies the DM-worker that needs to execute the breaking operation.
- `remove-id`: deprecated.
- `exec`:
  - Flag; boolean; `--exec`; optional
  - It cannot be specified simultaneously with the `--skip` parameter.
  - If it is specified, this command asks the DM-worker to execute the corresponding DDL statement of the lock.
- `skip`:
  - flag; boolean; `--skip`; optional
  - It cannot be specified simultaneously with the `--exec` parameter.
  - If it is specified, this command asks the DM-worker to skip the corresponding DDL statement of the lock.

- `task-name`:
  - Non-flag; string; required
  - It specifies the name of the task containing the lock that is going to execute the breaking operation (you can check whether a task contains the lock via query-status).

Example of results

```
» break-ddl-lock -w 127.0.0.1:8262 --exec test
{
    "result": true,                                 # The result of the
       ↪ lock breaking operation.
    "msg": "",                                      # The reason why the
       ↪ breaking lock operation failed.
    "workers": [                                    # The list of DM-
       ↪ workers which break the lock (currently the lock can be broken
       ↪ by only one DM-worker at a single operation).
        {
            "result": false,                        # The result of the
               ↪ lock breaking operation by the DM-worker.
            "worker": "127.0.0.1:8262",             # The DM-worker ID.
            "msg": ""                               # The reason why the DM
               ↪ -worker failed to break the lock.
        }
    ]
}
```

2.3.2.2 Supported scenarios

Currently, the `unlock-ddl-lock` or `break-ddl-lock` command only supports handling sharding DDL locks in the following three abnormal scenarios.

2.3.2.2.1 Scenario 1: Some DM-workers go offline

The reason for the abnormal lock

Before `DM-master` tries to automatically unlock the sharding DDL lock, all the DM-workers need to receive the sharding DDL events (for details, see shard merge principles). If the sharding DDL event is already in the migration process, and some DM-workers have gone offline and are not to be restarted (these DM-workers have been removed according to the application demand), then the sharding DDL lock cannot be automatically migrated and unlocked because not all the DM-workers can receive the DDL event.

Note:

If you need to make some DM-workers offline when not in the process of migrating sharding DDL events, a better solution is to use `stop-task` to stop the running tasks first, make the DM-workers go offline, remove the corresponding configuration information from the task configuration file, and finally use `start-task` and the new task configuration to restart the migration task.

Manual solution

Suppose that there are two instances `MySQL-1` and `MySQL-2` in the upstream, and there are two tables `shard_db_1.shard_table_1` and `shard_db_1.shard_table_2` in `MySQL-1` and two tables `shard_db_2.shard_table_1` and `shard_db_2.shard_table_2` in `MySQL-2`. Now we need to merge the four tables and migrate them into the table `shard_db.shard_table` in the downstream TiDB.

The initial table structure is:

```
mysql> SHOW CREATE TABLE shard_db_1.shard_table_1;
+---------------+-----------------------------------------+
| Table         | Create Table                            |
+---------------+-----------------------------------------+
| shard_table_1 | CREATE TABLE `shard_table_1` (
  `c1` int(11) NOT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+---------------+-----------------------------------------+
```

The following DDL operation will be executed on the upstream sharded tables to alter the table structure:

```
ALTER TABLE shard_db_*.shard_table_* ADD COLUMN c2 INT;
```

The operation processes of MySQL and DM are as follows:

1. The corresponding DDL operations are executed on the two sharded tables of `DM-worker-1` in `MySQL-1` to alter the table structures.

   ```
   ALTER TABLE shard_db_1.shard_table_1 ADD COLUMN c2 INT;
   ```

   ```
   ALTER TABLE shard_db_1.shard_table_2 ADD COLUMN c2 INT;
   ```

2. `DM-worker-1` sends the DDL information related to `MySQL-1` to `DM-master`, and `DM-master` creates the corresponding DDL lock.

3. Use `show-ddl-lock` to check the information of the current DDL lock.

```
» show-ddl-locks test
{
    "result": true,
    "msg": "",
    "locks": [
        {
            "ID": "test-`shard_db`.`shard_table`",
            "task": "test",
            "owner": "127.0.0.1:8262",
            "DDLs": [
                "USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` ADD
                    ↪  COLUMN `c2` int(11);"
            ],
            "synced": [
                "127.0.0.1:8262"
            ],
            "unsynced": [
                "127.0.0.1:8263"
            ]
        }
    ]
}
```

4. Due to the application demand, the `DM-worker-2` data in `MySQL-2` is no longer needed to be migrated to the downstream TiDB, and `DM-worker-2` is made offline.

5. The lock whose ID is `test-`shard_db`.`shard_table`` on `DM-master` cannot receive the DDL information of `DM-worker-2`.

    - The returned result `unsynced` by `show-ddl-locks` has always included the information of `DM-worker-2` (`127.0.0.1:8263`).

6. Use `unlock-ddl-lock` to ask `DM-master` to actively unlock the DDL lock.

    - If the owner of the DDL lock has gone offline, you can use the parameter `--owner` to specify another DM-worker as the new owner to execute the DDL.
    - If any DM-worker reports an error, `result` will be set to `false`, and at this point you should check carefully if the errors of each DM-worker is acceptable and within expectations.
        - DM-workers that have gone offline will return the error `rpc error: code = Unavailable`, which is within expectations and can be neglected; but if other online DM-workers return errors, then you should deal with them based on the scenario.

```
» unlock-ddl-lock test-`shard_db`.`shard_table`
{
    "result": false,
    "msg": "github.com/pingcap/tidb-enterprise-tools/dm/master/
        ↪ server.go:1472: DDL lock test-`shard_db`.`shard_table`
        ↪ owner ExecuteDDL successfully, so DDL lock removed. but
        ↪ some dm-workers ExecuteDDL fail, you should to handle dm-
        ↪ worker directly",
    "workers": [
        {
            "result": true,
            "worker": "127.0.0.1:8262",
            "msg": ""
        },
        {
            "result": false,
            "worker": "127.0.0.1:8263",
            "msg": "rpc error: code = Unavailable desc = all
                ↪ SubConns are in TransientFailure, latest
                ↪ connection error: connection error: desc = \"
                ↪ transport: Error while dialing dial tcp
                ↪ 127.0.0.1:8263: connect: connection refused\""
        }
    ]
}
```

7. Use `show-ddl-locks` to confirm if the DDL lock is unlocked successfully.

```
» show-ddl-locks test
{
    "result": true,
    "msg": "no DDL lock exists",
    "locks": [
    ]
}
```

8. Check whether the table structure is altered successfully in the downstream TiDB.

```
mysql> SHOW CREATE TABLE shard_db.shard_table;
+-------------+-------------------------------------------------------+
| Table       | Create Table                                          |
+-------------+-------------------------------------------------------+
| shard_table | CREATE TABLE `shard_table` (
  `c1` int(11) NOT NULL,
```

```
  `c2` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin |
+-------------+-----------------------------------------------------+
```

9. Use `query-status` to confirm if the migration task is normal.

Impact

After you have manually unlocked the lock by using `unlock-ddl-lock`, if you don't deal with the offline DM-workers included in the task configuration information, the lock might still be unable to be migrated automatically when the next sharding DDL event is received.

Therefore, after you have manually unlocked the DDL lock, you should perform the following operations:

1. Use `stop-task` to stop the running tasks.
2. Update the task configuration file, and remove the related information of the offline DM-worker from the configuration file.
3. Use `start-task` and the new task configuration file to restart the task.

> **Note:**
>
> After you run `unlock-ddl-lock`, if the DM-worker that went offline becomes online again and tries to migrate the data of the sharded tables, a match error between the data and the downstream table structure might occur.

2.3.2.2.2 Scenario 2: Some DM-workers restart during the DDL unlocking process

The reason for the abnormal lock

After `DM-master` receives the DDL events of all DM-workers, automatically running `unlock DDL lock` mainly include the following steps:

1. Ask the owner of the lock to execute the DDL and update the checkpoints of corresponding sharded tables.
2. Remove the DDL lock information stored on `DM-master` after the owner successfully executes the DDL.
3. Ask all other DM-workers to skip the DDL and update the checkpoints of corresponding sharded tables after the owner successfully executes the DDL.

Currently, the above unlocking process is not atomic. Therefore, after the owner successfully executes the DDL, if a DM-worker restarts during the period of asking other DM-workers to skip the DDL, then the DM-worker might fail to skip the DDL.

At this point, the lock information on `DM-master` has been removed and the restarted DM-worker will continue to migrate the DDL, but as other DM-workers (including the previous owner) has migrated the DDL and continued the migration process, this DM-worker will never see the DDL lock be unlocked automatically.

Manual solution

Suppose that now we have the same upstream and downstream table structures and the same demand for merging tables and migration as in the manual solution of Some DM-workers go offline.

When `DM-master` automatically executes the unlocking process, the owner (`DM-worker-1` ↪) successfully executes the DDL and continues the migration process, and the DDL lock information has been removed from `DM-master`. But at this point, if `DM-worker-2` restarts during the period of asking `DM-worker-2` to skip the DDL, then the skipping process might fail.

After `DM-worker-2` restarts, it will try to migrate the waiting DDL lock before it restarted. At this point, a new lock will be created on `DM-master`, and the DM-worker will become the owner of the lock (other DM-workers have executed/skipped the DDL by now and are continuing the migration process).

The operation processes are:

1. Use `show-ddl-locks` to confirm if the corresponding lock of the DDL exists on `DM-` ↪ `master`.

   Only the restarted DM-worker (`127.0.0.1:8263`) is at the `synced` state.

```
» show-ddl-locks
{
    "result": true,
    "msg": "",
    "locks": [
        {
            "ID": "test-`shard_db`.`shard_table`",
            "task": "test",
            "owner": "127.0.0.1:8263",
            "DDLs": [
                "USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` ADD
                    ↪  COLUMN `c2` int(11);"
            ],
            "synced": [
                "127.0.0.1:8263"
            ],
```

```
            "unsynced": [
                "127.0.0.1:8262"
            ]
        }
    ]
}
```

2. Use `unlock-ddl-lock` to ask `DM-master` to unlock the lock.

    - Use the parameter `--worker` to limit the operation to only target at the restarted DM-worker (`127.0.0.1:8263`).
    - The DM-worker will try to execute the DDL to the downstream again during the unlocking process (the owner before restarting has executed the DDL to the downstream), so as to make sure that the DDL can be executed multiple times.

```
» unlock-ddl-lock --worker=127.0.0.1:8263 test-`shard_db`.`
    ↪ shard_table`
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "127.0.0.1:8263",
            "msg": ""
        }
    ]
}
```

3. Use `show-ddl-locks` to confirm if the DDL lock has been successfully unlocked.

4. Use `query-status` to confirm if the migration task is normal.

Impact

After manually unlocking the lock, the following sharding DDL can be migrated automatically and normally.

2.3.2.2.3 Scenario 3: Some DM-workers are temporarily unreachable during the DDL unlocking process

The reason for the abnormal lock

This scenario has the similar reason for the abnormal lock in Scenario 2: Some DM-workers restart during the DDL unlocking process. If the DM-worker is temporarily unreachable when you request the DM-worker to skip the DDL statement, this DM-worker might fail

to skip the DDL statement. At this point, the lock information is removed from `DM-master`, but the DM-worker will continue to be waiting for a DDL lock which is no longer existing.

The difference between Scenario 3 and Scenario 2: Some DM-workers restart during the DDL unlocking process is that the DM-master does not have a lock in Scenario 3, but the DM-master has a new lock in Scenario 2.

Manual solution

Suppose that now we have the same upstream and downstream table structures and the same demand for merging tables and migration as in the manual solution of Some DM-workers go offline.

When `DM-master` automatically executes the unlocking operation, the owner (`DM-worker -1`) successfully executes the DDL and continues the migration process, and the DDL lock information has been removed from `DM-master`. But at this point, if `DM-worker-2` is temporarily unreachable due to the Internet failure during the period of asking `DM-worker-2` to skip the DDL, then the skipping process might fail.

The operation processes are:

1. Use `show-ddl-locks` to confirm if the corresponding lock of the DDL no longer exists on `DM-master`.
2. Use `query-status` to confirm if the DM-worker is still waiting for the lock to migrate.

```
» query-status test
{
    "result": true,
    "msg": "",
    "workers": [
        ...
        {
            ...
            "worker": "127.0.0.1:8263",
            "subTaskStatus": [
                {
                    ...
                    "unresolvedDDLLockID": "test-`shard_db`.`shard_table`
                        ↪ ",
                    "sync": {
                        ...
                        "blockingDDLs": [
                            "USE `shard_db`; ALTER TABLE `shard_db`.`
                                ↪ shard_table` ADD COLUMN `c2` int(11);"
                        ],
                        "unresolvedGroups": [
                            {
```

```
                        "target": "`shard_db`.`shard_table`",
                        "DDLs": [
                            "USE `shard_db`; ALTER TABLE `shard_db
                                ↪ `.`shard_table` ADD COLUMN `c2`
                                ↪ int(11);"
                        ],
                        "firstPos": "(mysql-bin|000001.000003,
                            ↪ 1752)",
                        "synced": [
                            "`shard_db_2`.`shard_table_1`",
                            "`shard_db_2`.`shard_table_2`"
                        ],
                        "unsynced": [
                        ]
                    }
                ],
                "synced": false
            }
        }
    ]
    ...
  }
 ]
}
```

3. Use `break-ddl-lock` to compulsorily break the DDL lock which the DM-worker is waiting for.

   As the owner has executed the DDL to the downstream, you should use the parameter `--skip` to break the lock.

```
» break-ddl-lock --worker=127.0.0.1:8263 --skip test
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "127.0.0.1:8263",
            "msg": ""
        }
    ]
}
```

4. Use `query-status` to confirm if the migration task is normal and no longer at the state of waiting for the lock.

Impact

After manually breaking the lock, the following sharding DDL can be migrated automatically and normally.

3 Benchmark

3.1 DM 1.0-GA Benchmark Report

This benchmark report describes the test purpose, environment, scenario, and result for DM 1.0-GA.

3.1.1 Test purpose

The purpose of this test is to test the performance of DM full import and incremental replication.

3.1.2 Test environment

3.1.2.1 Machine information

System information:

Machine IP	Operation system	Kernel version	File system type
172.16.4.39	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
172.16.4.40	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
172.16.4.41	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
172.16.4.42	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
172.16.4.43	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
172.16.4.44	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4

Hardware information:

Type	Specification
CPU	40 CPUs, Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz
Memory	192GB, 12 * 16GB DIMM DDR4 2133 MHz
Disk	Intel DC P4510 4TB NVMe PCIe 3.0
Network card	10 Gigabit Ethernet

Others:

- Network rtt between servers: rtt min/avg/max/mdev = 0.074/0.088/0.121/0.019 ms

3.1.2.2 Cluster topology

Machine IP	Deployment instance
172.16.4.39	PD1, DM-worker1, DM-master
172.16.4.40	PD2, MySQL1
172.16.4.41	PD3, TiDB
172.16.4.42	TiKV1
172.16.4.43	TiKV2
172.16.4.44	TiKV3

3.1.2.3 Version information

- MySQL version: 5.7.27-log
- TiDB version: v4.0.0-alpha-198-gbde7f440e
- DM version: v1.0.1
- Sysbench version: 1.0.17

3.1.3 Test scenario

3.1.3.1 Data flow

MySQL1 (172.16.4.40) -> DM-worker1 (172.16.4.39) -> TiDB (172.16.4.41)

3.1.3.2 Public configuration or data

3.1.3.2.1 Database table structure used for the test

```
CREATE TABLE `sbtest` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `k` int(11) NOT NULL DEFAULT '0',
  `c` char(120) CHARSET utf8mb4 COLLATE utf8mb4_bin NOT NULL DEFAULT '',
  `pad` char(60) CHARSET utf8mb4 COLLATE utf8mb4_bin NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
```

3.1.3.2.2 Database configuration

We use TiDB Ansible to deploy the TiDB cluster, and use default configuration provided in TiDB Ansible.

3.1.3.3 Full import benchmark case

3.1.3.3.1 Test procedure

- Set up environment
- Use sysbench to create the table and generate the initial data in upstream MySQL
- Start DM-task in the `full` mode

Sysbench test script used for preparing initial data:

```
sysbench --test=oltp_insert --tables=4 --mysql-host=172.16.4.40 --mysql-
    ↪ port=3306 --mysql-user=root --mysql-db=dm_benchmark --db-driver=mysql
    ↪  --table-size=50000000 prepare
```

3.1.3.3.2 Full import benchmark result

item	dump thread	mydumpers extra-args	dump speed (MB/s)
enable single table concurrent	32	“-r 320000 –regex ’^sbtest.*’“	191.03
disable single table concurrent	32	“–regex ’^sbtest.*’“	72.22

item	latency of execute transaction (s)	statement per transaction	data size (GB)	time (s)	import speed (MB/s)
load data	1.737	4878	38.14	2346.9	16.64

3.1.3.3.3 Benchmark result with different pool size in load unit

Full import data size in benchmark case is 3.78 GB, which is generated from sysbench by the following script:

```
sysbench --test=oltp_insert --tables=4 --mysql-host=172.16.4.40 --mysql-
    ↪ port=3306 --mysql-user=root --mysql-db=dm_benchmark --db-driver=mysql
    ↪  --table-size=5000000 prepare
```

load pool size	latency of execution txn (s)	import time (s)	import speed (MB/s)	TiDB 99 duration (s)
2	0.250	425.9	9.1	0.23
4	0.523	360.1	10.7	0.41
8	0.986	267.0	14.5	0.93
16	2.022	265.9	14.5	2.68
32	3.778	262.3	14.7	6.39
64	7.452	281.9	13.7	8.00

3.1.3.3.4 Benchmark result with different row count in per statement

Full import data size in this benchmark case is 3.78 GB, load unit pool size uses 32. The statement count is controlled by parameters of the dump unit.

row count in per statement	mydumpers extra-args	latency of execution txn (s)	import time (s)	import speed (MB/s)	TiDB 99 duration (s)
7426	-s 1500000 -r 320000	6.982	258.3	15.0	10.34
4903	-s 1000000 -r 320000	3.778	262.3	14.7	6.39
2470	-s 500000 -r 320000	1.962	271.36	14.3	2.00
1236	-s 250000 -r 320000	1.911	283.3	13.7	1.50
618	-s 125000 -r 320000	0.683	299.9	12.9	0.73
310	-s 62500 -r 320000	0.413	322.6	12.0	0.49

3.1.3.4 Increase migration benchmark case

3.1.3.4.1 Test procedure

- Set up environment
- Use sysbench to create the table and generate the initial data in upstream MySQL
- Start DM-task in the `all` mode, and wait until the task enters `sync` unit
- Use sysbench to generate incremental data in upstream MySQL, use `query-status` to watch the DM migration status, and observe the monitoring metrics of DM and TiDB on Grafana

3.1.3.4.2 Benchmark result for incremental replication

Upstream sysbench test script:

```
sysbench --test=oltp_insert --tables=4 --num-threads=32 --mysql-host
    ↪ =172.17.4.40 --mysql-port=3306 --mysql-user=root --mysql-db=
    ↪ dm_benchmark --db-driver=mysql --report-interval=10 --time=1800 run
```

DM sync unit `worker-count` is 32, and `batch` size is 100 in this benchmark case.

items	qps	tps	95% Latency
MySQL	42.79k	42.79k	1.18ms
DM relay log unit	-	11.3MB/s	45us (read duration)
DM binlog replication unit	22.97k (binlog event received qps, not including skipped events)	-	20ms (txn execution latency)
TiDB	31.30k (Begin/Commit 3.93k Insert 22.76k)	4.16k	95%: 6.4ms 99%: 9ms

3.1.3.4.3 Benchmark result with different sync unit concurrency

sync unit worker-count	DM tps	DM execution latency (ms)	TiDB qps	TiDB 99 duration (ms)
4	7074	63	7.1k	3
8	14684	64	14.9k	4
16	23486	56	24.9k	6
32	23345	28	29.2k	10
64	23302	30	31.2k	16
1024	22225	70	56.9k	70

3.1.3.4.4 Benchmark result with different SQL distribution

sysbench type	relay log flush speed (MB/s)	DM tps	DM execution latency (ms)	TiDB qps	TiDB 99 duration (ms)
insert_only	11.3	23345	28	29.2k	10
write_only	18.7	33470	129	34.6k	11

3.1.4 Recommended parameters

3.1.4.1 dump unit

We recommend that the statement size be 200 KB~1 MB, and row count in each statement be approximately 1000~5000, which is based on the actual row size in your scenario.

3.1.4.2 load unit

We recommend that you set `pool-size` to 16.

3.1.4.3 sync unit

We recommend that you set `batch` size to 100 and `worker-count` to 16~32.

3.2 DM 1.0-alpha Benchmark Report

This DM benchmark report describes the test purpose, environment, scenario, and result.

3.2.1 Test purpose

The purpose of this test is to test the performance of DM incremental replication.

> **Note:**
>
> The results of the testing might vary based on different environmental dependencies.

3.2.2 Test environment

3.2.2.1 Machine information

System information:

Machine IP	Operation system	Kernel version	File system type
192.168.0.6	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
192.168.0.7	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
192.168.0.8	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
192.168.0.9	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
192.168.0.10	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
192.168.0.11	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4

Hardware information:

Type	192.168.0.9, 192.168.0.10, 192.168.0.11	192.168.0.6, 192.168.0.7, 192.168.0.8
CPU	8 vCPUs, Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz	4 vCPUs, Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz
Memory	16G	8G
Disk	1T Aliyun ESSD	256G Aliyun ESSD
Network card	1 Gigabit Ethernet, 1000Mb/s	1 Gigabit Ethernet, 1000Mb/s

3.2.2.2 Cluster topology

Machine IP	Deployment instance
192.168.0.9	TiKV * 1, TiDB * 1
192.168.0.10	TiKV * 1
192.168.0.11	TiKV * 1
192.168.0.6	PD * 1, MySQL * 1, DM-worker * 1
192.168.0.8	PD * 1, MySQL * 1, DM-worker * 1
192.168.0.7	PD * 1, DM-master * 1

3.2.2.3 Version information

- MySQL version: 5.7.25-log
- TiDB version: v3.0.0-beta-27-g6398788
- DM version: v1.0.0-alpha-10-g4d01d79
- Sysbench version: 1.0.9

3.2.3 Test scenario

3.2.3.1 Data flow

MySQL1 (192.168.0.8) -> DM-worker1 (192.168.0.6) -> TiDB (192.168.0.9)

3.2.3.2 Test procedure

- Set up environment
- Use sysbench to create the table and generate the initial data in upstream MySQL
- Start DM-task in the `all` mode
- Use sysbench to generate incremental data in upstream MySQL

3.2.3.3 Use sysbench to generate data load in upstream MySQL

Upstream sysbench test script:

```
sysbench --test=oltp_insert --tables=2 --num-threads=1024 --mysql-host
    ↪ =192.168.0.8 --mysql-port=3306 --mysql-user=root --mysql-db=dm_poc --
    ↪ db-driver=mysql --report-interval=10 --time=900 run
```

The structure of the table used for the test:

```
CREATE TABLE `sbtest` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `k` int(11) NOT NULL DEFAULT '0',
  `c` char(120) CHARSET utf8mb4 COLLATE utf8mb4_bin NOT NULL DEFAULT '',
  `pad` char(60) CHARSET utf8mb4 COLLATE utf8mb4_bin NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
```

3.2.3.4 The deployment and configuration details

```
// TiKV configuration
sync-log = false
[defaultcf]
block-cache-size = "4GB"
[writecf]
block-cache-size = "4GB"
[raftdb.defaultcf]
block-cache-size = "4GB"

// DM task sync processing unit configuration
syncer:
    worker-count: 256
    batch: 100
    max-retry: 20
```

3.2.4 Test result

items	threads	qps	tps	95% Latency (ms)
MySQL	1024	15.10k	15.10k	121.08
DM	256	13.89k	13.89k	210 (txn execution latency)
TiDB	-	18.53k (Begin/Commit 2.4k Replace 13.80k)	2.27k	29

3.2.4.1 Monitor screenshots

3.2.4.1.1 DM key indicator monitor

Figure 5: DM benchmark

3.2.4.1.2 TiDB key indicator monitor

4 Usage Scenarios

4.1 Data Migration Simple Usage Scenario

This document shows how to use Data Migration (DM) in a simple data migration scenario where the data of three upstream MySQL instances needs to be migrated to a downstream TiDB cluster (no sharding data).

4.1.1 Upstream instances

Assume that the upstream schemas are as follows:

- Instance 1

Schema	Tables
user	information, log
store	store_bj, store_tj
log	messages

- Instance 2

Schema	Tables
user	information, log
store	store_sh, store_sz
log	messages

- Instance 3

Schema	Tables
user	information, log
store	store_gz, store_sz
log	messages

4.1.2 Migration requirements

1. Do not merge the `user` schema.
    1. Migrate the `user` schema of instance 1 to the `user_north` of TiDB.
    2. Migrate the `user` schema of instance 2 to the `user_east` of TiDB.
    3. Migrate the `user` schema of instance 3 to the `user_south` of TiDB.
    4. Never delete the table `log`.
2. Migrate the upstream `store` schema to the downstream `store` schema without merging tables.
    1. `store_sz` exists in both instances 2 and 3, which is migrated to `store_suzhou` and `store_shenzhen` respectively.
    2. Never delete `store`.
3. The `log` schema needs to be filtered out.

4.1.3 Downstream instances

Assume that the schemas migrated to the downstream are as follows:

Schema	Tables
user_north	information, log
user_east	information, log
user_south	information, log
store	store_bj, store_tj, store_sh, store_suzhou, store_gz, store_shenzhen

4.1.4 Migration solution

- To satisfy migration Requirements #1-i, #1-ii and #1-iii, configure the table routing rules as follows:

```
routes:
  ...
  instance-1-user-rule:
    schema-pattern: "user"
    target-schema: "user_north"
  instance-2-user-rule:
    schema-pattern: "user"
    target-schema: "user_east"
  instance-3-user-rule:
    schema-pattern: "user"
    target-schema: "user_south"
```

- To satisfy the migration Requirement #2-i, configure the table routing rules as follows:

```
routes:
  ...
  instance-2-store-rule:
    schema-pattern: "store"
    table-pattern: "store_sz"
    target-schema: "store"
    target-table: "store_suzhou"
  instance-3-store-rule:
    schema-pattern: "store"
    table-pattern: "store_sz"
    target-schema: "store"
    target-table: "store_shenzhen"
```

- To satisfy the migration Requirement #1-iv, configure the binlog filtering rules as follows:

```
filters:
  ...
  log-filter-rule:
    schema-pattern: "user"
```

```
    table-pattern: "log"
    events: ["truncate table", "drop table", "delete"]
    action: Ignore
  user-filter-rule:
    schema-pattern: "user"
    events: ["drop database"]
    action: Ignore
```

- To satisfy the migration Requirement #2-ii, configure the binlog filtering rule as follows:

```
filters:
  ...
  store-filter-rule:
    schema-pattern: "store"
    events: ["drop database", "truncate table", "drop table", "delete"]
    action: Ignore
```

> **Note:**
>
> `store-filter-rule` is different from `log-filter-rule & user-filter -rule`. `store-filter-rule` is a rule for the whole `store` schema, while `log-filter-rule` and `user-filter-rule` are rules for the `log` table in the `user` schema.

- To satisfy the migration Requirement #3, configure the block and allow lists as follows:

```
block-allow-list: # Use black-white-list if the DM's version <= v1.0.6.
  log-ignored:
    ignore-dbs: ["log"]
```

4.1.5 Migration task configuration

The complete migration task configuration is shown below. For more details, see configuration explanations.

```
name: "one-tidb-secondary"
task-mode: all
meta-schema: "dm_meta"
remove-meta: false

target-database:
  host: "192.168.0.1"
  port: 4000
  user: "root"
```

```
  password: ""

mysql-instances:
  -
    source-id: "instance-1"
    route-rules: ["instance-1-user-rule"]
    filter-rules: ["log-filter-rule", "user-filter-rule", "store-filter-rule
      ↪ "]
    block-allow-list: "log-ignored" # Use black-white-list if the DM's
      ↪ version <= v1.0.6.
    mydumper-config-name: "global"
    loader-config-name: "global"
    syncer-config-name: "global"
  -
    source-id: "instance-2"
    route-rules: ["instance-2-user-rule", instance-2-store-rule]
    filter-rules: ["log-filter-rule", "user-filter-rule", "store-filter-rule
      ↪ "]
    block-allow-list: "log-ignored" # Use black-white-list if the DM's
      ↪ version <= v1.0.6.
    mydumper-config-name: "global"
    loader-config-name: "global"
    syncer-config-name: "global"
  -
    source-id: "instance-3"
    route-rules: ["instance-3-user-rule", instance-3-store-rule]
    filter-rules: ["log-filter-rule", "user-filter-rule", "store-filter-rule
      ↪ "]
    block-allow-list: "log-ignored" # Use black-white-list if the DM's
      ↪ version <= v1.0.6.
    mydumper-config-name: "global"
    loader-config-name: "global"
    syncer-config-name: "global"

## other common configs shared by all instances

routes:
  instance-1-user-rule:
    schema-pattern: "user"
    target-schema: "user_north"
  instance-2-user-rule:
    schema-pattern: "user"
    target-schema: "user_east"
  instance-3-user-rule:
    schema-pattern: "user"
```

```
    target-schema: "user_south"
  instance-2-store-rule:
    schema-pattern: "store"
    table-pattern: "store_sz"
    target-schema: "store"
    target-table: "store_suzhou"
  instance-3-store-rule:
    schema-pattern: "store"
    table-pattern: "store_sz"
    target-schema: "store"
    target-table: "store_shenzhen"

filters:
  log-filter-rule:
    schema-pattern: "user"
    table-pattern: "log"
    events: ["truncate table", "drop table", "delete"]
    action: Ignore
  user-filter-rule:
    schema-pattern: "user"
    events: ["drop database"]
    action: Ignore
  store-filter-rule:
    schema-pattern: "store"
    events: ["drop database", "truncate table", "drop table", "delete"]
    action: Ignore

block-allow-list: # Use black-white-list if the DM's version <= v1.0.6.
  log-ignored:
    ignore-dbs: ["log"]

mydumpers:
  global:
    threads: 4
    chunk-filesize: 64
    skip-tz-utc: true

loaders:
  global:
    pool-size: 16
    dir: "./dumped_data"

syncers:
  global:
    worker-count: 16
```

```
  batch: 100
  max-retry: 100
```

4.2 Data Migration Shard Merge Scenario

This document shows how to use Data Migration (DM) in the shard merge scenario where the sharded schemas and sharded tables of three upstream MySQL instances need to be migrated to a downstream TiDB cluster.

4.2.1 Upstream instances

Assume that the upstream schemas are as follows:

- Instance 1

Schema	Tables
user	information, log_north, log_bak
store_01	sale_01, sale_02
store_02	sale_01, sale_02

- Instance 2

Schema	Tables
user	information, log_east, log_bak
store_01	sale_01, sale_02
store_02	sale_01, sale_02

- Instance 3

Schema	Tables
user	information, log_south, log_bak
store_01	sale_01, sale_02
store_02	sale_01, sale_02

4.2.2 Migration requirements

1. Merge tables with the same name. For example, merge the `user.information` tables of three upstream instances to the downstream `user.information` table in TiDB.
2. Merge tables with different names. For example, merge the `user.log_{north|south|east}` tables of three upstream instances to the downstream `user.log_{north|south|east}` table in TiDB.

3. Merge sharded tables. For example, merge the `store_{01|02}.sale_{01|02}` tables of three upstream instances to the downstream `store.sale` table in TiDB.
4. Filter delete operations. For example, filter out all the delete operations in the `user` ↪ `.log_{north|south|east}` table of three upstream instances.
5. Filter delete operations. For example, filter out all the delete operations in the `user` ↪ `.information` table of three upstream instances.
6. Filter delete operations. For example, filter out all the delete operations in the `store_` ↪ `{01|02}.sale_{01|02}` table of three upstream instances.
7. Use wildcards to filter specific tables. For example, filter out the `user.log_bak` tables of three upstream instances using wildcard `user.log_*`.
8. Troubleshoot primary key conflicts. Because the `store_{01|02}.sale_{01|02}` tables have auto-increment primary keys of the `bigint` type, the conflict occurs when these tables are merged into TiDB. The following text will show you solutions to resolve and avoid the conflict.

4.2.3 Downstream instances

Assume that the downstream schema after migration is as follows:

Schema	Tables
user	information, log_north, log_east, log_south
store	sale

4.2.4 Migration solution

- To satisfy the migration Requirements #1 and #2, configure the table routing rule as follows:

```
routes:
  ...
  user-route-rule:
    schema-pattern: "user"
    target-schema: "user"
```

- To satisfy the migration Requirement #3, configure the table routing rule as follows:

```
routes:
  ...
  store-route-rule:
    schema-pattern: "store_*"
    target-schema: "store"
  sale-route-rule:
    schema-pattern: "store_*"
    table-pattern: "sale_*"
```

```
    target-schema: "store"
    target-table: "sale"
```

- To satisfy the migration Requirements #4 and #5, configure the binlog event filtering rule as follows:

```
filters:
  ...
  user-filter-rule:
    schema-pattern: "user"
    events: ["truncate table", "drop table", "delete", "drop database"]
    action: Ignore
```

> **Note:**
>
> The migration Requirements #4 and #5 indicate that all the deletion operations in the `user` schema are filtered out, so a schema level filtering rule is configured here. And the deletion operations of tables in the `user` schema participating in the future migration will also be filtered out.

- To satisfy the migration Requirement #6, configure the binlog event filter rule as follows:

```
filters:
  ...
  sale-filter-rule:
    schema-pattern: "store_*"
    table-pattern: "sale_*"
    events: ["truncate table", "drop table", "delete"]
    action: Ignore
  store-filter-rule:
    schema-pattern: "store_*"
    events: ["drop database"]
    action: Ignore
```

- To satisfy the migration Requirement #7, configure the block and allow table lists as follows:

```
block-allow-list: # Use black-white-list if the DM's version <= v1.0.6.
  log-bak-ignored:
    ignore-tales:
    - db-name: "user"
      tbl-name: "log_bak"
```

- To satisfy the migration Requirement #8, first refer to handling conflicts of auto-increment primary key to solve conflicts. This guarantees that data is successfully migrated to the downstream when the primary key value of one sharded table is duplicate with that of another sharded table. Then, configure `ignore-checking-items` to skip checking the conflict of auto-increment primary key:

```
ignore-checking-items: ["auto_increment_ID"]
```

4.2.5 Migration task configuration

The complete configuration of the migration task is shown as below. For more details, see Data Migration Task Configuration File.

```
name: "shard_merge"
task-mode: all
meta-schema: "dm_meta"
remove-meta: false
ignore-checking-items: ["auto_increment_ID"]

target-database:
  host: "192.168.0.1"
  port: 4000
  user: "root"
  password: ""

mysql-instances:
  -
    source-id: "instance-1"
    route-rules: ["user-route-rule", "store-route-rule", "sale-route-rule"]
    filter-rules: ["user-filter-rule", "store-filter-rule" , "sale-filter-
       ↪ rule"]
    block-allow-list: "log-bak-ignored" # Use black-white-list if the DM's
       ↪ version <= v1.0.6.
    mydumper-config-name: "global"
    loader-config-name: "global"
    syncer-config-name: "global"

  -
    source-id: "instance-2"
    route-rules: ["user-route-rule", "store-route-rule", "sale-route-rule"]
    filter-rules: ["user-filter-rule", "store-filter-rule" , "sale-filter-
       ↪ rule"]
    block-allow-list: "log-bak-ignored" # Use black-white-list if the DM's
       ↪ version <= v1.0.6.
    mydumper-config-name: "global"
```

```
    loader-config-name: "global"
    syncer-config-name: "global"
  -
    source-id: "instance-3"
    route-rules: ["user-route-rule", "store-route-rule", "sale-route-rule"]
    filter-rules: ["user-filter-rule", "store-filter-rule" , "sale-filter-
       ↪ rule"]
    block-allow-list: "log-bak-ignored" # Use black-white-list if the DM's
       ↪ version <= v1.0.6.
    mydumper-config-name: "global"
    loader-config-name: "global"
    syncer-config-name: "global"

## Other common configs shared by all instances.

routes:
  user-route-rule:
    schema-pattern: "user"
    target-schema: "user"
  store-route-rule:
    schema-pattern: "store_*"
    target-schema: "store"
  sale-route-rule:
    schema-pattern: "store_*"
    table-pattern: "sale_*"
    target-schema: "store"
    target-table: "sale"

filters:
  user-filter-rule:
    schema-pattern: "user"
    events: ["truncate table", "drop table", "delete", "drop database"]
    action: Ignore
  sale-filter-rule:
    schema-pattern: "store_*"
    table-pattern: "sale_*"
    events: ["truncate table", "drop table", "delete"]
    action: Ignore
  store-filter-rule:
    schema-pattern: "store_*"
    events: ["drop database"]
    action: Ignore

block-allow-list: # Use black-white-list if the DM's version <= v1.0.6.
  log-bak-ignored:
```

```
    ignore-tales:
    - db-name: "user"
      tbl-name: "log_bak"

mydumpers:
  global:
    threads: 4
    chunk-filesize: 64
    skip-tz-utc: true

loaders:
  global:
    pool-size: 16
    dir: "./dumped_data"

syncers:
  global:
    worker-count: 16
    batch: 100
    max-retry: 100
```

4.3 Best Practices of Data Migration in the Shard Merge Scenario

This document describes the features and limitations of TiDB Data Migration (DM) in the shard merge scenario and provides a data migration best practice guide for your application.

4.3.1 Use a separate data migration task

In the Merge and migrate Data from Sharded Tables document, the definition of "sharding group" is given: A sharding group consists of all upstream tables that need to be merged and migrated into the same downstream table.

The current sharding DDL mechanism has some usage restrictions to coordinate the schema changes brought by DDL operations in different sharded tables. If these restrictions are violated due to unexpected reasons, you need to handle sharding DDL locks manually in DM, or even redo the entire data migration task.

To mitigate the impact on data migration when an exception occurs, it is recommended to merge and migrate each sharding group as a separate data migration task. **This might enable that only a small number of data migration tasks need to be handled manually while others remain unaffected.**

4.3.2 Handle sharding DDL locks manually

You can easily conclude from Merge and migrate Data from Sharded Tables that DM's sharding DDL lock is a mechanism for coordinating the execution of DDL operations to the downstream from multiple upstream sharded tables.

Therefore, when you find any sharding DDL lock on `DM-master` through `show-ddl-locks` command, or any `unresolvedGroups` or `blockingDDLs` on some DM-workers through `query-status` command, do not rush to manually release the sharding DDL lock through `unlock-ddl-lock` or `break-ddl-lock` commands.

Instead, you can:

- Follow the corresponding manual solution to handle the scenario if the failure of automatically releasing the sharding DDL lock is one of the listed abnormal scenarios.
- Redo the entire data migration task if it is an unsupported scenario: First, empty the data in the downstream database and the `dm_meta` information associated with the migration task; then, re-execute the full and incremental data migration.

4.3.3 Handle conflicts of auto-increment primary key

DM offers the column mapping feature to handle conflicts that might occur in merging the `bigint` type of auto-increment primary key. However, it is **strongly discouraged** to choose this approach. If it is acceptable in the production environment, the following two alternatives are recommended.

4.3.3.1 Remove the `PRIMARY KEY` attribute from the column

Assume that the upstream schemas are as follows:

```
CREATE TABLE `tbl_no_pk` (
  `auto_pk_c1` bigint(20) NOT NULL,
  `uk_c2` bigint(20) NOT NULL,
  `content_c3` text,
  PRIMARY KEY (`auto_pk_c1`),
  UNIQUE KEY `uk_c2` (`uk_c2`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

If the following requirements are satisfied:

- The `auto_pk_c1` column has no impact on the application and does not depend on the column's `PRIMARY KEY` attribute.
- The `uk_c2` column has the `UNIQUE KEY` attribute, and it is globally unique in all upstream sharded tables.

Then you can perform the following steps to fix the `ERROR 1062 (23000): Duplicate entry '***' for key 'PRIMARY'` error that is possibly caused by the `auto_pk_c1` column when you merge sharded tables.

1. Before the full data migration, create a table in the downstream database for merging and migrating data, and modify the `PRIMARY KEY` attribute of the `auto_pk_c1` column to normal index.

```
CREATE TABLE `tbl_no_pk_2` (
  `auto_pk_c1` bigint(20) NOT NULL,
  `uk_c2` bigint(20) NOT NULL,
  `content_c3` text,
  INDEX (`auto_pk_c1`),
  UNIQUE KEY `uk_c2` (`uk_c2`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

2. Add the following configuration in `task.yaml` to skip the check of auto-increment primary key conflict:

```
ignore-checking-items: ["auto_increment_ID"]
```

3. Start the full and incremental data migration task.

4. Run `query-status` to verify whether the data migration task is successfully processed and whether the data from the upstream has already been merged and migrated to the downstream database.

4.3.3.2 Use a composite primary key

Assume that the upstream schemas are as follows:

```
CREATE TABLE `tbl_multi_pk` (
  `auto_pk_c1` bigint(20) NOT NULL,
  `uuid_c2` bigint(20) NOT NULL,
  `content_c3` text,
  PRIMARY KEY (`auto_pk_c1`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

If the following requirements are satisfied:

- The application does not depend on the `PRIMARY KEY` attribute of the `auto_pk_c1` column.
- The composite primary key that consists of the `auto_pk_c1` and `uuid_c2` columns is globally unique.
- It is acceptable to use a composite primary key in the application.

Then you can perform the following steps to fix the `ERROR 1062 (23000): Duplicate entry '***' for key 'PRIMARY'` error that is possibly caused by the `auto_pk_c1` column when you merge sharded tables.

1. Before the full data migration, create a table in the downstream database for merging and migrating data. Do not specify the `PRIMARY KEY` attribute for the `auto_pk_c1` column, but use the `auto_pk_c1` and `uuid_c2` columns to make up a composite primary key.

```
CREATE TABLE `tbl_multi_pk_c2` (
  `auto_pk_c1` bigint(20) NOT NULL,
  `uuid_c2` bigint(20) NOT NULL,
  `content_c3` text,
  PRIMARY KEY (`auto_pk_c1`,`uuid_c2`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

2. Start the full and incremental data migration task.
3. Run `query-status` to verify whether the data migration task is successfully processed and whether the data from upstream has already been merged and migrated to the downstream database.

4.3.4 Create/drop tables in the upstream

In Merge and migrate Data from Sharded Tables, it is clear that the coordination of sharding DDL lock depends on whether the downstream database receives the DDL statements of all upstream sharded tables. In addition, DM currently **does not support** dynamically creating or dropping sharded tables in the upstream. Therefore, to create or drop sharded tables in the upstream, it is recommended to perform the following steps.

4.3.4.1 Create sharded tables in the upstream

If you need to create a new sharded table in the upstream, perform the following steps:

1. Wait for the coordination of all executed sharding DDL in the upstream sharded tables to finish.
2. Run `stop-task` to stop the data migration task.
3. Create a new sharded table in the upstream.
4. Make sure that the configuration in the `task.yaml` file allows the newly added sharded table to be merged in one downstream table with other existing sharded tables.
5. Run `start-task` to start the task.
6. Run `query-status` to verify whether the data migration task is successfully processed and whether the data from upstream has already been merged and migrated to the downstream database.

4.3.4.2 Drop sharded tables in the upstream

If you need to drop a sharded table in the upstream, perform the following steps:

1. Drop the sharded table, run `SHOW BINLOG EVENTS` to fetch the `End_log_pos` corresponding to the `DROP TABLE` statement in the binlog events, and mark it as *Pos-M*.
2. Run `query-status` to fetch the position (`syncerBinlog`) corresponding to the binlog event that has been processed by DM, and mark it as *Pos-S*.
3. When *Pos-S* is greater than *Pos-M*, it means that DM has processed all of the `DROP TABLE` statements, and the data of the table before dropping has been migrated to the downstream, so the subsequent operation can be performed. Otherwise, wait for DM to finish migrating the data.
4. Run `stop-task` to stop the task.
5. Make sure that the configuration in the `task.yaml` file ignores the dropped sharded table in the upstream.
6. Run `start-task` to start the task.
7. Run `query-status` to verify whether the data migration task is successfully processed.

4.3.5 Speed limits and traffic flow control

When data from multiple upstream MySQL or MariaDB instances is merged and migrated to the same TiDB cluster in the downstream, every DM-worker corresponding to each upstream instance executes full and incremental data migration concurrently. This means that the default degree of concurrency (`pool-size` in full data migration and `worker-count` in incremental data replication) accumulates as the number of DM-workers increases, which might overload the downstream database. In this case, you need to conduct a preliminary performance analysis based on TiDB and DM monitoring metrics and adjust the value of each concurrency parameter. In the future, DM is expected to support partially automated traffic flow control.

4.4 Switch DM-worker Connection between Upstream MySQL Instances

When the upstream MySQL instance that DM-worker connects to needs downtime maintenance or when the instance crashes unexpectedly, you need to switch the DM-worker connection to another MySQL instance within the same migration group.

Note:

- You can switch the DM-worker connection to only an instance within the same primary-secondary migration cluster.
- The MySQL instance to be newly connected to must have the binlog required by DM-worker.
- DM-worker must operate in the GTID sets mode, which means you must specify `enable_gtid=true` when you deploy DM using DM-Ansible.
- The connection switch only supports the following two scenarios. Strictly follow the procedures for each scenario. Otherwise, you might have to re-deploy the DM cluster according to the newly connected MySQL instance and perform the data migration task all over again.

For more details on GTID set, refer to MySQL documentation.

4.4.1 Switch DM-worker connection via virtual IP

When DM-worker connects the upstream MySQL instance via a virtual IP (VIP), switching the VIP connection to another MySQL instance means switching the MySQL instance connected to DM-worker, without the upstream connection address changed.

Note:

Make necessary changes to DM in this scenario. Otherwise, when you switch the VIP connection to another MySQL instance, DM might connect to the new and old MySQL instances at the same time in different connections. In this situation, the binlog replicated to DM is not consistent with other upstream status that DM receives, causing unpredictable anomalies and even data damage.

To switch one upstream MySQL instance (when DM-worker connects to it via a VIP) to another, perform the following steps:

1. Use the `query-status` command to get the GTID sets (`relayBinlogGtid`) corresponding to the binlog that relay log has replicated from the old MySQL instance. Mark the sets as `gtid-W`.
2. Use the `SELECT @@GLOBAL.gtid_purged;` command on the new MySQL instance to get the GTID sets corresponding to the purged binlogs. Mark the sets as `gtid-P`.
3. Use the `SELECT @@GLOBAL.gtid_executed;` command on the new MySQL instance to get the GTID sets corresponding to all successfully executed transactions. Mark the sets as `gtid-E`.

4. Make sure that the following conditions are met. Otherwise, you cannot switch the DM-work connection to the new MySQL instance:

    - `gtid-W` contains `gtid-P`. `gtid-P` can be empty.
    - `gtid-E` contains `gtid-W`.

5. Use `pause-relay` to pause relay.
6. Use `pause-task` to pause all running tasks of data migration.
7. Change the VIP for it to direct at the new MySQL instance.
8. Use `switch-relay-master` to tell relay to execute the primary-secondary switch.
9. Use `resume-relay` to make relay resume to read binlog from the new MySQL instance.
10. Use `resume-task` to resume the previous migration task.

4.4.2 Change the address of the upstream MySQL instance that DM-worker connects to

To make DM-worker connect to a new MySQL instance in the upstream by modifying the DM-worker configuration, perform the following steps:

1. Use the `query-status` command to get the GTID sets (`relayBinlogGtid`) corresponding to the binlog that relay log has replicated from the old MySQL instance. Mark this sets as `gtid-W`.
2. Use the `SELECT @@GLOBAL.gtid_purged;` command on the new MySQL instance to get the GTID sets corresponding to the purged binlogs. Mark this sets as `gtid-P`.
3. Use the `SELECT @@GLOBAL.gtid_executed;` command on the new MySQL instance to get the GTID sets corresponding to all successfully executed transactions. Mark this sets as `gtid-E`.
4. Make sure that the following conditions are met. Otherwise, you cannot switch the DM-work connection to the new MySQL instance:

    - `gtid-W` contains `gtid-P`. `gtid-P` can be empty.
    - `gtid-E` contains `gtid-W`.

5. Use `stop-task` to stop all running tasks of data migration.
6. Update the DM-worker configuration in the `inventory.ini` file and use DM-Ansible to perform a rolling upgrade on DM-worker.
7. Use `start-task` to restart the migration task.

5 TiDB DM (Data Migration) Tutorial

TiDB DM (Data Migration) is a platform that supports migrating large, complex, production data sets from MySQL or MariaDB to TiDB.

DM supports creating and importing an initial dump of data, as well as keeping data migrated during migration by reading and applying binary logs from the source data store. DM can migrate sharded topologies from in-production databases by merging tables from

multiple separate upstream MySQL/MariaDB instances/clusters. In addition to its use for migrations, DM is often used on an ongoing basis by existing MySQL or MariaDB users who deploy a TiDB cluster as a secondary library, to either provide improved horizontal scalability or run real-time analytical workloads on TiDB without needing to manage an ETL pipeline.

In this tutorial, we'll see how to migrate a sharded table from multiple upstream MySQL instances. We'll do this a couple of different ways. First, we'll merge several tables/shards that do not conflict; that is, they're partitioned using a scheme that does not result in conflicting unique key values. Then, we'll merge several tables that **do** have conflicting unique key values.

This tutorial assumes you're using a new, clean CentOS 7 instance. You can virtualize locally (using VMware, VirtualBox, etc.), or deploy a small cloud VM on your favorite provider. You'll have the best luck if you have at least 1GB of memory, since we're going to run quite a few services.

> **Warning:**
>
> The methodology used to deploy TiDB in this tutorial should **not** be used to deploy TiDB in a production or development setting.

5.1 Architecture

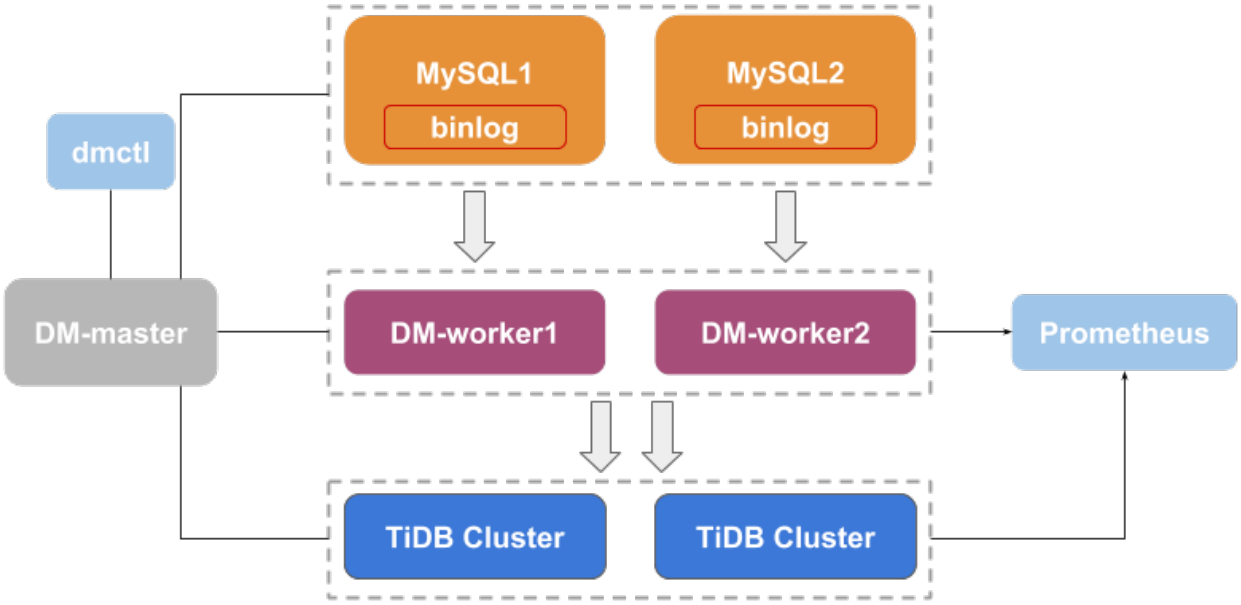


Figure 6: TiDB DM architecture

The TiDB DM (Data Migration) platform consists of 3 components: DM-master, DM-worker, and dmctl.

- DM-master manages and schedules the operation of data migration tasks.
- DM-worker executes specific data migration tasks.
- dmctl is the command line tool used to control the DM cluster.

Individual tasks are defined in .yaml files that are read by dmctl and submitted to DM-master. DM-master then informs each instance of DM-worker of its responsibilities for a given task.

For additional information about DM, please consult Data Migration Overview in the TiDB documentation.

5.2 Setup

We're going to deploy 3 instances of MySQL Server, and 1 instance each of pd-server, tikv-server, and tidb-server. Then we'll start a single DM-master and 3 instances of DM-worker.

1. Install MySQL 5.7, download and extract the TiDB v3.0 and DM v1.0.2 packages we'll use:

```
sudo yum install -y http://repo.mysql.com/yum/mysql-5.7-community/el/7/
    ↪ x86_64/mysql57-community-release-el7-10.noarch.rpm
sudo yum install -y mysql-community-server
curl https://download.pingcap.org/tidb-v3.0-linux-amd64.tar.gz | tar
    ↪ xzf -
curl https://download.pingcap.org/dm-v1.0.2-linux-amd64.tar.gz | tar
    ↪ xzf -
curl -L https://github.com/pingcap/docs/raw/
    ↪ a164f19957e4cd2126961fad2fc8d96965b1651c/dev/how-to/get-started/
    ↪ dm-cnf/dm-cnf.tgz | tar xvzf -
```

2. Create some directories and symlinks:

```
mkdir -p bin data logs
ln -sf -t bin/ "$HOME"/*/bin/*
[[ :$PATH: = *:$HOME/bin:* ]] || echo 'export PATH=$PATH:$HOME/bin' >>
    ↪ ~/.bash_profile && . ~/.bash_profile
```

3. Set up configuration for the 3 instances of MySQL Server we'll start:

```
tee -a "$HOME/.my.cnf" <<EoCNF
[server]
socket=mysql.sock
pid-file=mysql.pid
log-error=mysql.err
log-bin
auto-increment-increment=5
[server1]
datadir=$HOME/data/mysql1
server-id=1
port=3307
auto-increment-offset=1
[server2]
datadir=$HOME/data/mysql2
server-id=2
port=3308
auto-increment-offset=2
[server3]
datadir=$HOME/data/mysql3
server-id=3
port=3309
auto-increment-offset=3
EoCNF
```

4. Initialize and start our MySQL instances:

```
for i in 1 2 3
do
    echo "mysql$i"
    mysqld --defaults-group-suffix="$i" --initialize-insecure
    mysqld --defaults-group-suffix="$i" &
done
```

5. To make sure your MySQL server instances are all running, you can execute `jobs` and/or `pgrep -a mysqld`:

```
jobs
```

```
[1]   Running                 mysqld --defaults-group-suffix="$i" &
[2]-  Running                 mysqld --defaults-group-suffix="$i" &
[3]+  Running                 mysqld --defaults-group-suffix="$i" &
```

```
pgrep -a mysqld
```

```
17672 mysqld --defaults-group-suffix=1
17727 mysqld --defaults-group-suffix=2
17782 mysqld --defaults-group-suffix=3
```

5.3 Migrating shards

Our first scenario consists of 3 "shards" with the same schema, but non-overlapping auto-increment primary keys.

We achieve that by having set `auto-increment-increment=5` and `auto-increment-offset` in our .my.cnf file. `auto-increment-increment` tells each instance to increment by 5 for each new auto-increment ID it generates, and `auto-increment-offset`, set differently for each instance, tells that instance the offset from 0 to start counting. For example, an instance with `auto-increment-increment=5` and `auto-increment-offset=2` will generate the auto-increment ID sequence {2,7,12,17,22,...}.

1. Create our MySQL database and table in each of the 3 MySQL Server instances:

```
for i in 1 2 3
do
    mysql -h 127.0.0.1 -P "$((3306+i))" -u root <<EoSQL
        create database dmtest1;
        create table dmtest1.t1 (id bigint unsigned not null
            ↪ AUTO_INCREMENT primary key, c char(32), port int);
EoSQL
done
```

2. Insert a few hundred rows into each of the MySQL instances:

```
for i in 1 2 3; do
    mysql -h 127.0.0.1 -P "$((3306+i))" -u root dmtest1 <<EoSQL
        insert into t1 values (),(),(),(),(),(),(),();
        insert into t1 (id) select null from t1;
        insert into t1 (id) select null from t1;
        insert into t1 (id) select null from t1;
        insert into t1 (id) select null from t1;
        insert into t1 (id) select null from t1;
        update t1 set c=md5(id), port=@@port;
EoSQL
done
```

3. Select the rows back from the MySQL instances to make sure things look right:

```
for i in 1 2 3; do
    mysql -N -h 127.0.0.1 -P "$((3306+i))" -u root -e 'select * from
        ↪ dmtest1.t1'
done | sort -n
```

Note that we have incrementing, non-overlapping IDs in the left-hand column. The port number in the right-hand column shows which instance the rows were inserted into and are being selected from:

```
...
1841   e8dfff4676a47048d6f0c4ef899593dd   3307
1842   57c0531e13f40b91b3b0f1a30b529a1d   3308
1843   4888241374e8c62ddd9b4c3cfd091f96   3309
1846   f45a1078feb35de77d26b3f7a52ef502   3307
1847   82cadb0649a3af4968404c9f6031b233   3308
1848   7385db9a3f11415bc0e9e2625fae3734   3309
1851   ff1418e8cc993fe8abcfe3ce2003e5c5   3307
1852   eb1e78328c46506b46a4ac4a1e378b91   3308
1853   7503cfacd12053d309b6bed5c89de212   3309
1856   3c947bc2f7ff007b86a9428b74654de5   3307
1857   a3545bd79d31f9a72d3a78690adf73fc   3308
1858   d7fd118e6f226a71b5f1ffe10efd0a78   3309
```

5.4 Starting DM master and workers

Our goal in this exercise is to use DM to combine the data from these distinct MySQL instances into a single table in TiDB.

The package of configuration files we unpacked earlier (dm-cnf.tgz) contains the configuration for the components of the TiDB cluster, the DM components, and for the 2 DM tasks we'll explore in this tutorial.

We'll start a single tidb-server instance, one DM-worker process for each of the MySQL server instances (3 total), and a single DM-master process:

```
tidb-server --log-file=logs/tidb-server.log &
for i in 1 2 3; do dm-worker --config=dm-cnf/dm-worker$i.toml & done
dm-master --config=dm-cnf/dm-master.toml &
```

You can execute `jobs` and/or `ps -a` to make sure these processes are all running:

```
jobs
```

```
[1]   Running                 mysqld --defaults-group-suffix="$i" &
[2]   Running                 mysqld --defaults-group-suffix="$i" &
[3]   Running                 mysqld --defaults-group-suffix="$i" &
[4]   Running                 tidb-server --log-file=logs/tidb-server.log &
[5]   Running                 dm-worker --config=dm-cnf/dm-worker$i.toml &
[6]   Running                 dm-worker --config=dm-cnf/dm-worker$i.toml &
[7]-  Running                 dm-worker --config=dm-cnf/dm-worker$i.toml &
[8]+  Running                 dm-master --config=dm-cnf/dm-master.toml &
```

```
ps -a
```

```
  PID TTY          TIME CMD
17317 pts/0    00:00:00 screen
17672 pts/1    00:00:04 mysqld
17727 pts/1    00:00:04 mysqld
17782 pts/1    00:00:04 mysqld
18586 pts/1    00:00:02 tidb-server
18587 pts/1    00:00:00 dm-worker
18588 pts/1    00:00:00 dm-worker
18589 pts/1    00:00:00 dm-worker
18590 pts/1    00:00:00 dm-master
18892 pts/1    00:00:00 ps
```

Each of the upstream MySQL Server instances corresponds to a separate DM-worker instance, each of which has its own configuration file. These files describe the details of the connection to the upstream MySQL Server as well as where to store the relay log files (the local copy of the upstream server's binary log) and the output of Mydumper. Each DM-worker should listen on a different port (defined by `worker-addr`). Here's dm-worker1.toml, for example:

```
# Worker Configuration.
```

```
server-id = 1
source-id = "mysql1"
flavor = "mysql"
worker-addr = ":8262"
log-file = "logs/worker1.log"
relay-dir = "data/relay1"
meta-dir = "data/meta1"

[from]
host = "127.0.0.1"
user = "root"
password = ""
port = 3307
```

- If you migrate data from MySQL Server, Percona Server, Percona XtraDB Cluster, Amazon Aurora or RDS, set the `flavor` option to `"mysql"`, which is the default value. This value is valid only when you are using a MySQL version between 5.5 (not included) and 8.0 (not included).
- If you migrate data from MariaDB Server or MariaDB (Galera) Cluster, set `flavor = "mariadb"`. You can set this value only when you are using a MariaDB version later than 10.1.2.
- Starting with DM 1.0.2, DM automatically generates the values of the `flavor` and `server-id` options. You do not need to manually configure these options in normal situations.
- If `password` in the `[from]` configuration is not an empty string, you need to use dmctl to encrypt the password. Refer to Encrypt the upstream MySQL user password using dmctl for detailed steps.

Tasks are defined in YAML files. First, let's look at dmtask1.yaml:

```
name: dmtask1
task-mode: all
is-sharding: true
enable-heartbeat: true
ignore-checking-items: ["auto_increment_ID"]

target-database:
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: ""

mysql-instances:
  - source-id: "mysql1"
```

```
    server-id: 1
    block-allow-list: "dmtest1" # Use black-white-list if the DM's version
        ↪ <= v1.0.6.
    loader-config-name: "loader1"
  - source-id: "mysql2"
    server-id: 2
    block-allow-list: "dmtest1" # Use black-white-list if the DM's version
        ↪ <= v1.0.6.
    loader-config-name: "loader2"
  - source-id: "mysql3"
    server-id: 3
    block-allow-list: "dmtest1" # Use black-white-list if the DM's version
        ↪ <= v1.0.6.
    loader-config-name: "loader3"

block-allow-list: # Use black-white-list if the DM's version <= v1.0.6.
  dmtest1:
    do-dbs: ["dmtest1"]

loaders:
  loader1:
    dir: "data/dump1"
  loader2:
    dir: "data/dump2"
  loader3:
    dir: "data/dump3"
```

There are a number of global options, and several groups of options that define various behaviors.

- `task-mode: all` tells DM to both import a full backup of the upstream instances as well as replicate incremental updates using the upstream MySQL server's binary log.
  - Alternatively, you can give `task-mode` the `full` or `incremental` value, respectively, to get only one of those two behaviors.
- `is-sharding: true` tells DM that we want multiple DM-worker instances to work on a single task to merge several upstream shards into a single downstream table.
- `ignore-checking-items: ["auto_increment_ID"]` disables DM's detection of potential auto-increment conflicts among the upstream instances. DM can detect that all 3 upstream MySQL servers have an auto-increment column for a table with the same name in the same schema, and that this situation would be expected to lead to conflicts among the several tables. We've avoided that by setting `auto-increment-increment` and `auto-increment-offset` so that each of the MySQL servers gives non-overlapping IDs. So, we tell DM to ignore checking for overlapping auto-increment IDs in this task.

- The `target-database` section defines the information of the connected target database. If `password` is not an empty string, you need to use dmctl to encrypt the password. Refer to Encrypt the upstream MySQL user password using dmctl for detailed steps.
- We use `block-allow-list` to limit the scope of this task to database `dmtest`.
- The `loaders` section defines where to find the output of each instance of Mydumper that was executed by the respective instance of DM-worker.

The `dmctl` tool is an interactive client that facilitates interaction with the DM cluster. You use it to start tasks, query task status, et cetera. Start the tool by executing `dmctl -master-addr :8261` to get the interactive prompt:

```
dmctl -master-addr :8261
```

```
Welcome to dmctl
Release Version: v1.0.0-alpha-69-g5134ad1
Git Commit Hash: 5134ad19fbf6c57da0c7af548f5ca2a890bddbe4
Git Branch: master
UTC Build Time: 2019-04-29 09:36:42
Go Version: go version go1.12 linux/amd64

»
```

To start dmtask1, execute `start-task dm-cnf/dmtask1.yaml`:

```
» start-task dm-cnf/dmtask1.yaml
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "127.0.0.1:8262",
            "msg": ""
        },
        {
            "result": true,
            "worker": "127.0.0.1:8263",
            "msg": ""
        },
        {
            "result": true,
            "worker": "127.0.0.1:8264",
            "msg": ""
```

```
        }
    ]
}
```

Starting the task will kick off the actions defined in the task configuration file. That includes executing instances of Mydumper and loader, and connecting the workers to the upstream MySQL servers as migration secondaries after the initial data dump has been loaded.

We can see that all rows have been migrated to the TiDB server:

```
mysql -h 127.0.0.1 -P 4000 -u root -e 'select * from t1' dmtest1 | tail
```

Expect this output:

```
...
1843   4888241374e8c62ddd9b4c3cfd091f96   3309
1846   f45a1078feb35de77d26b3f7a52ef502   3307
1847   82cadb0649a3af4968404c9f6031b233   3308
1848   7385db9a3f11415bc0e9e2625fae3734   3309
1851   ff1418e8cc993fe8abcfe3ce2003e5c5   3307
1852   eb1e78328c46506b46a4ac4a1e378b91   3308
1853   7503cfacd12053d309b6bed5c89de212   3309
1856   3c947bc2f7ff007b86a9428b74654de5   3307
1857   a3545bd79d31f9a72d3a78690adf73fc   3308
1858   d7fd118e6f226a71b5f1ffe10efd0a78   3309
```

DM is now acting as a secondary library to each of the MySQL servers, reading their binary logs to apply updates in realtime to the downstream TiDB server:

```
for i in 1 2 3
do
    mysql -h 127.0.0.1 -P "$((3306+i))" -u root -e 'select host, command,
       ↪ state from information_schema.processlist where command="Binlog
       ↪ Dump"'
done
```

Expect this output:

```
+-----------------+------------+-----------------------------------------------------------
   ↪
| host          | command    | state
   ↪                                                    |
+-----------------+------------+-----------------------------------------------------------
   ↪
| localhost:42168 | Binlog Dump | Master has sent all binlog to slave;
   ↪ waiting for more updates |
```

```
+-----------------+------------+-------------------------------------------------------------
   ↪
+-----------------+------------+-------------------------------------------------------------
   ↪
| host          | command    | state
   ↪                                                        |
+-----------------+------------+-------------------------------------------------------------
   ↪
| localhost:42922 | Binlog Dump | Master has sent all binlog to slave;
   ↪ waiting for more updates |
+-----------------+------------+-------------------------------------------------------------
   ↪
+-----------------+------------+-------------------------------------------------------------
   ↪
| host          | command    | state
   ↪                                                        |
+-----------------+------------+-------------------------------------------------------------
   ↪
| localhost:56798 | Binlog Dump | Master has sent all binlog to slave;
   ↪ waiting for more updates |
+-----------------+------------+-------------------------------------------------------------
   ↪
```

We can see that this is the case by inserting some rows into the upstream MySQL servers, selecting those rows from TiDB, updating those same rows in MySQL, and selecting them again:

```
for i in 1 2 3; do
    mysql -N -h 127.0.0.1 -P "$((3306+i))" -u root -e 'insert into t1 (id)
       ↪ select null from t1' dmtest1
done
mysql -h 127.0.0.1 -P 4000 -u root -e 'select * from t1' dmtest1 | tail
```

Expect this output:

```
6313    NULL    NULL
6316    NULL    NULL
6317    NULL    NULL
6318    NULL    NULL
6321    NULL    NULL
6322    NULL    NULL
6323    NULL    NULL
6326    NULL    NULL
6327    NULL    NULL
6328    NULL    NULL
```

Now update those rows, so we can see that changes to data are correctly propagated to TiDB:

```
for i in 1 2 3; do
    mysql -N -h 127.0.0.1 -P "$((3306+i))" -u root -e 'update t1 set c=md5(
        ↪ id), port=@@port' dmtest1
done | sort -n
mysql -h 127.0.0.1 -P 4000 -u root -e 'select * from t1' dmtest1 | tail
```

Expect this output:

```
6313    2118d8a1b7004ed5baf5347a4f99f502    3309
6316    6107d91fc9a0b04bc044aa7d8c1443bd    3307
6317    0e9b734aa25ca8096cb7b56dc0dd8929    3308
6318    b0eb9a95e8b085e4025eae2f0d76a6a6    3309
6321    7cb36e23529e4de4c41460940cc85e6e    3307
6322    fe1f9c70bdf347497e1a01b6c486bdb9    3308
6323    14eac0d254a6ccaf9b67584c7830a5c0    3309
6326    17b65afe58c49edc1bdd812c554ee3bb    3307
6327    c54bc2ded4480856dc9f39bdcf35a3e7    3308
6328    b294504229c668e750dfcc4ea9617f0a    3309
```

As long as the DM master and workers are running the "dmtest1" task, they'll continue to keep the downstream TiDB server migrated with the upstream MySQL server instances.

5.5 Conclusion

In this tutorial, a shard migration has been performed from three upstream MySQL server instances. You can see how DM imports an initial dump of data in the cluster, reads binlogs from MySQL to replicate incremental data, and keeps the downstream TiDB cluster in sync with the upstream instances.

For additional information about DM, consult Data Migration Overview in the TiDB documentation or join the TiDB Community Slack channel!

6 Deploy

6.1 Deploy a DM Cluster

6.1.1 Deploy Data Migration Using DM-Ansible

DM-Ansible is a cluster deployment tool developed by PingCAP based on the Playbooks feature of Ansible (an IT automation tool). This guide shows how to quickly deploy a Data Migration (DM) cluster using DM-Ansible.

6.1.1.1 Prepare

Before you start, make sure you have the following machines as required.

1. Several target machines that meet the following requirements:

    - CentOS 7.3 (64-bit) or later, x86_64 architecture (AMD64)
    - Network between machines
    - Closing the firewall, or opening the service port

2. A Control Machine that meets the following requirements:

> **Note:**
> The Control Machine can be one of the target machines.

    - CentOS 7.3 (64-bit) or later, with Python 2.7 installed
    - Ansible 2.5 or later installed
    - Access to the Internet

6.1.1.2 Step 1: Install system dependencies on the Control Machine

Log in to the Control Machine using the `root` user account, and run the corresponding command according to your operating system.

- If you use a Control Machine installed with CentOS 7, run the following command:

```
yum -y install epel-release git curl sshpass &&
um -y install python-pip
```

- If you use a Control Machine installed with Ubuntu, run the following command:

```
apt-get -y install git curl sshpass python-pip
```

6.1.1.3 Step 2: Create the `tidb` user on the Control Machine and generate the SSH key

Make sure you have logged in to the Control Machine using the `root` user account, and then perform the following steps.

1. Create the `tidb` user.

```
useradd -m -d /home/tidb tidb
```

2. Set a password for the `tidb` user account.

```
passwd tidb
```

3. Configure sudo without password for the `tidb` user account by adding `tidb ALL=(ALL ↪ )NOPASSWD: ALL` to the end of the sudo file:

```
visudo
```

```
tidb ALL=(ALL) NOPASSWD: ALL
```

4. Generate the SSH key.

   Execute the `su` command to switch the user from `root` to `tidb`.

```
su - tidb
```

   Create the SSH key for the `tidb` user account and hit the Enter key when `Enter ↪ passphrase` is prompted. After successful execution, the SSH private key file is `/home/tidb/.ssh/id_rsa`, and the SSH public key file is `/home/tidb/.ssh/id_rsa. ↪ pub`.

```
ssh-keygen -t rsa
```

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tidb/.ssh/id_rsa):
Created directory '/home/tidb/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/tidb/.ssh/id_rsa.
Your public key has been saved in /home/tidb/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:eIBykszR1KyECA/h0d7PRKz4fhAeli7IrVphhte7/So tidb@172.16.10.49
The key's randomart image is:
+---[RSA 2048]----+
|=+o+.o.          |
|o=o+o.oo         |
| .O.=.=          |
| . B.B +         |
|o B * B S        |
| * + * +         |
|  o + .          |
| o  E+ .         |
|o   ..+o.        |
+----[SHA256]-----+
```

6.1.1.4 Step 3: Download DM-Ansible to the Control Machine

Make sure you have logged in to the Control Machine using the `tidb` user account.

1. Go to the `/home/tidb` directory.
2. Run the following command to download DM-Ansible.

```
wget http://download.pingcap.org/dm-ansible-{version}.tar.gz
```

`{version}` is the DM version that you expect to download, like `v1.0.1` and `v1.0.2`.

You can check out DM's published versions on DM Release page. You can also replace `{version}` with `latest` to download the latest development version that has not been published.

6.1.1.5 Step 4: Install DM-Ansible and its dependencies on the Control Machine

Make sure you have logged in to the Control Machine using the `tidb` user account.

It is required to use `pip` to install DM-Ansible and its dependencies, otherwise a compatibility issue occurs. Currently, DM-Ansible is compatible with Ansible 2.5 or later.

1. Install DM-Ansible and the dependencies on the Control Machine:

```
tar -xzvf dm-ansible-{version}.tar.gz &&
mv dm-ansible-{version} dm-ansible &&
cd /home/tidb/dm-ansible &&
sudo pip install -r ./requirements.txt
```

DM-Ansible and the related dependencies are in the `dm-ansible/requirements.txt` file.

2. View the version of Ansible:

```
ansible --version
```

```
ansible 2.5.0
```

6.1.1.6 Step 5: Configure the SSH mutual trust and sudo rules on the Control Machine

Make sure you have logged in to the Control Machine using the `tidb` user account.

1. Add the IPs of your deployment target machines to the `[servers]` section of the `hosts.ini` file.

```
cd /home/tidb/dm-ansible &&
vi hosts.ini
```

```
[servers]
172.16.10.71
172.16.10.72
172.16.10.73

[all:vars]
username = tidb
```

2. Run the following command and input the password of the `root` user account of your deployment target machines.

```
ansible-playbook -i hosts.ini create_users.yml -u root -k
```

This step creates the `tidb` user account on the deployment target machines, configures the sudo rules and the SSH mutual trust between the Control Machine and the deployment target machines.

6.1.1.7 Step 6: Download DM and the monitoring component installation package to the Control Machine

Make sure the Control Machine is connected to the Internet and run the following command:

```
ansible-playbook local_prepare.yml
```

6.1.1.8 Step 7: Edit the `inventory.ini` file to orchestrate the DM cluster

Log in to the Control Machine using the `tidb` user account, and edit the `/home/tidb/dm-ansible/inventory.ini` file to orchestrate the DM cluster.

> **Note:**
>
> It is required to use the internal IP address to deploy. If the SSH port of the target machines is not the default 22 port, you need to add the `ansible_port` variable, as shown in the following example:

```
dm-worker1 ansible_host=172.16.10.72 ansible_port=5555 server_id=101
    ↪ mysql_host=172.16.10.72 mysql_user=root mysql_password='VjX8cEeTX+
    ↪ qcvZ3bPa04h0C80pe/1aU=' mysql_port=3306
```

You can choose one of the following two types of cluster topology according to your scenario:

- The cluster topology of a single DM-worker instance on each node
- The cluster topology of multiple DM-worker instances on each node

  Generally, it is recommended to deploy one DM-worker instance on each node. However, if the CPU and memory of your machine are much better than the required in Hardware and Software Requirements, and you have more than 2 disks in one node or the capacity of one SSD is larger than 2 TB, you can deploy no more than 2 DM-worker instances on a single node.

6.1.1.8.1 Option 1: Use the cluster topology of a single DM-worker instance on each node

Name	Host IP	Services
node1	172.16.10.71	DM-master, Prometheus, Grafana, Alertmanager, DM Portal
node2	172.16.10.72	DM-worker1
node3	172.16.10.73	DM-worker2
mysql-replica-01	172.16.10.81	MySQL
mysql-replica-02	172.16.10.82	MySQL

```
### DM modules.
[dm_master_servers]
dm_master ansible_host=172.16.10.71

[dm_worker_servers]
dm_worker1 ansible_host=172.16.10.72 server_id=101 source_id="mysql-replica
    ↪ -01" mysql_host=172.16.10.81 mysql_user=root mysql_password='
    ↪ VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306

dm_worker2 ansible_host=172.16.10.73 server_id=102 source_id="mysql-replica
    ↪ -02" mysql_host=172.16.10.82 mysql_user=root mysql_password='
    ↪ VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306

[dm_portal_servers]
dm_portal ansible_host=172.16.10.71

### Monitoring modules.
[prometheus_servers]
prometheus ansible_host=172.16.10.71

[grafana_servers]
```

```
grafana ansible_host=172.16.10.71

[alertmanager_servers]
alertmanager ansible_host=172.16.10.71

### Global variables.
[all:vars]
cluster_name = test-cluster

ansible_user = tidb

dm_version = {version}

deploy_dir = /data1/dm

grafana_admin_user = "admin"
grafana_admin_password = "admin"
```

`{version}` is the version number of the DM-Ansible you use. For details about DM-worker parameters, see DM-worker configuration parameters description.

6.1.1.8.2 Option 2: Use the cluster topology of multiple DM-worker instances on each node

Name	Host IP	Services
node1	172.16.10.71	DM-master, Prometheus, Grafana, Alertmanager, DM Portal
node2	172.16.10.72	DM-worker1-1, DM-worker1-2
node3	172.16.10.73	DM-worker2-1, DM-worker2-2

When you edit the `inventory.ini` file, pay attention to distinguish between the following variables: `server_id`, `deploy_dir`, and `dm_worker_port`.

```
### DM modules.
[dm_master_servers]
dm_master ansible_host=172.16.10.71

[dm_worker_servers]
dm_worker1_1 ansible_host=172.16.10.72 server_id=101 deploy_dir=/data1/
   ↪ dm_worker dm_worker_port=8262 mysql_host=172.16.10.81 mysql_user=root
   ↪  mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
dm_worker1_2 ansible_host=172.16.10.72 server_id=102 deploy_dir=/data2/
   ↪ dm_worker dm_worker_port=8263 mysql_host=172.16.10.82 mysql_user=root
   ↪  mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
```

```
dm_worker2_1 ansible_host=172.16.10.73 server_id=103 deploy_dir=/data1/
    ↪ dm_worker dm_worker_port=8262 mysql_host=172.16.10.83 mysql_user=root
    ↪  mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
dm_worker2_2 ansible_host=172.16.10.73 server_id=104 deploy_dir=/data2/
    ↪ dm_worker dm_worker_port=8263 mysql_host=172.16.10.84 mysql_user=root
    ↪  mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306

[dm_portal_servers]
dm_portal ansible_host=172.16.10.71

### Monitoring modules.
[prometheus_servers]
prometheus ansible_host=172.16.10.71

[grafana_servers]
grafana ansible_host=172.16.10.71

[alertmanager_servers]
alertmanager ansible_host=172.16.10.71

### Global variables.
[all:vars]
cluster_name = test-cluster

ansible_user = tidb

dm_version = {version}

deploy_dir = /data1/dm

grafana_admin_user = "admin"
grafana_admin_password = "admin"
```

`{version}` is the version number of the DM-Ansible you use.

6.1.1.8.3 DM-worker configuration parameters description

Variable name	Description
source_id	DM-worker binds to a unique database instance or a replication group with the primary-secondary architecture. When the primary and secondary switch, you only need to update `mysql_host` ↪ or `mysql_port` ↪ and do not need to update the `source_id` ↪ .

Variable name	Description
server_id	DM-worker connects to MySQL as a secondary database. This variable is the server ID of the secondary database. Keep it globally unique in the MySQL cluster, where the value range is 0 ~ 4294967295. In v1.0.2 and later versions, the server ID is automatically generated by DM.
mysql_host	The upstream MySQL host.

Variable name	Description
mysql_user	The upstream MySQL username (`"root"` by default).
mysql_password	The upstream MySQL user password. You need to encrypt the upstream MySQL user password using dmctl.
mysql_port	The upstream MySQL port (3306 by default).

Variable name	Description
enable_gtid	Whether DM-worker uses GTID to pull the binlog. The pre-requisite is that the upstream MySQL has enabled the GTID mode.
relay_binlog_name	Specifies the file name from which DM-worker starts to pull the binlog. Only used when the local has no valid relay log. In v1.0.2 and later versions, DM pulls the binlog starting from the latest file by default.

Variable name	Description
relay_binlog_gtid	Specifies the GTID from which DM-worker starts to pull the binlog. Only used when the local has no valid relay log and `enable_gtid` ↪ is `true`. In v1.0.2 and later versions, DM pulls the binlog from the latest file by default.

Variable name	Description
flavor	Indicates the release type of MySQL (`"mysql"` by default). For the official version, Percona, and cloud MySQL, fill in `"mysql"`; for MariaDB, fill in `"mariadb"`. In v1.0.2 and later versions, DM automatically detects the upstream version and fills in the release type.

For details about the `deploy_dir` configuration, see Configure the deployment directory.

6.1.1.8.4 Encrypt the upstream MySQL user password using dmctl

Assuming that the upstream MySQL user password is `123456`, configure the generated string to the `mysql_password` variable of DM-worker.

```
cd /home/tidb/dm-ansible/resources/bin &&
./dmctl -encrypt 'abc!@#123'
```

```
MKxn0Qo3m3XOyjCnhEMtsUCm83EhGQDZ/T4=
```

Note:

- If the database has no password, you can skip this step.
- DM v1.0.6 and later versions can configure the plaintext database password.

6.1.1.9 Step 8: Edit variables in the `inventory.ini` file

This step shows how to make configuration changes to the `inventory.ini` file.

6.1.1.9.1 Configure the deployment directory

Edit the `deploy_dir` variable to configure the deployment directory.

- The global variable is set to `/home/tidb/deploy` by default, and it applies to all services. If the data disk is mounted on the `/data1` directory, you can set it to `/data1 ↪ /dm`. For example:

```
## Global variables.
[all:vars]
deploy_dir = /data1/dm
```

- If you need to set a separate deployment directory for a service, you can configure the host variable while configuring the service host list in the `inventory.ini` file. It is required to add the first column alias, to avoid confusion in scenarios of mixed services deployment.

```
dm-master ansible_host=172.16.10.71 deploy_dir=/data1/deploy
```

6.1.1.9.2 Configure the relay log position

When you start DM-worker for the first time, you need to configure `relay_binlog_name` to specify the position where DM-worker starts to pull the corresponding upstream MySQL or MariaDB binlog.

```
[dm_worker_servers]
```

```
dm-worker1 ansible_host=172.16.10.72 source_id="mysql-replica-01" server_id
    ↪ =101 relay_binlog_name="binlog.000011" mysql_host=172.16.10.81
    ↪ mysql_user=root mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU='
    ↪ mysql_port=3306

dm-worker2 ansible_host=172.16.10.73 source_id="mysql-replica-02" server_id
    ↪ =102 relay_binlog_name="binlog.000002" mysql_host=172.16.10.82
    ↪ mysql_user=root mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU='
    ↪ mysql_port=3306
```

Note:

If `relay_binlog_name` is not specified, DM-worker pulls the binlog starting from the earliest existing binlog file of the upstream MySQL or MariaDB by default. In this event, it may take a significant amount of time to retrieve all of the binlog files. In v1.0.2 and later versions, DM defaults to pulling the binlog starting from the latest file.

6.1.1.9.3 Enable the relay log GTID migration mode

In a DM cluster, the relay log processing unit of DM-worker communicates with the upstream MySQL or MariaDB to pull its binlog to the local file system.

You can enable the relay log GTID migration mode by configuring the following items. Currently, DM supports MySQL GTID and MariaDB GTID.

- `enable_gtid`: to enable the GTID mode. This helps improve the handling of migration topology changes, such as a switch between primary and secondary
- `relay_binlog_gtid`: to specify the position where DM-worker starts to pull the corresponding upstream MySQL or MariaDB binlog

```
[dm_worker_servers]
dm-worker1 ansible_host=172.16.10.72 source_id="mysql-replica-01" server_id
    ↪ =101 enable_gtid=true relay_binlog_gtid="aae3683d-f77b-11e7-9e3b-02
    ↪ a495f8993c:1-282967971,cc97fa93-f5cf-11e7-ae19-02915c68ee2e
    ↪ :1-284361339" mysql_host=172.16.10.81 mysql_user=root mysql_password
    ↪ ='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306

dm-worker2 ansible_host=172.16.10.73 source_id="mysql-replica-02" server_id
    ↪ =102 relay_binlog_name=binlog.000002 mysql_host=172.16.10.82
    ↪ mysql_user=root mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU='
    ↪ mysql_port=3306
```

6.1.1.9.4 Global variables description

Variable name	Description
cluster_name	The name of a cluster, adjustable.
dm_version	The version of DM, configured by default.
grafana_admin_user	The username of the Grafana administrator (`admin` by default).
grafana_admin_password	The password of the Grafana administrator account, used to import Dashboard by Ansible (`admin` by default). Update this variable if you have modified it through the Grafana web.

6.1.1.10 Step 9: Deploy the DM cluster

When `ansible-playbook` runs Playbook, the default concurrent number is 5. If many deployment target machines are deployed, you can add the `-f` parameter to specify the concurrency, such as `ansible-playbook deploy.yml -f 10`.

The following example uses `tidb` as the user who runs the service.

1. Edit the `dm-ansible/inventory.ini` file to make sure `ansible_user = tidb`.

```
ansible_user = tidb
```

> **Note:**
> Do not configure `ansible_user` to `root`, because `tidb-ansible` limits the user that runs the service to the normal user.

Run the following command and if all servers return `tidb`, then the SSH mutual trust is successfully configured:

```
ansible -i inventory.ini all -m shell -a 'whoami'
```

Run the following command and if all servers return `root`, then sudo without password of the `tidb` user is successfully configured:

```
ansible -i inventory.ini all -m shell -a 'whoami' -b
```

2. Modify kernel parameters, and deploy the DM cluster components and monitoring components.

```
ansible-playbook deploy.yml
```

> **Note:**
> Currently, both DM and TiDB overwrite the original running configuration of the monitoring components during deployment and rolling upgrade. Therefore, it is highly recommended to deploy independent monitoring components for DM and TiDB.

3. Start the DM cluster.

```
ansible-playbook start.yml
```

This operation starts all the components in the entire DM cluster in order, which include DM-master, DM-worker, and the monitoring components. You can use this command to start a DM cluster after it is stopped.

6.1.1.11 Step 10: Stop the DM cluster

If you need to stop the DM cluster, run the following command:

```
ansible-playbook stop.yml
```

This operation stops all the components in the entire DM cluster in order, which include DM-master, DM-worker, and the monitoring components.

6.1.1.12 Common deployment issues

6.1.1.12.1 Service default ports

Component	Port variable	Default port	Description
DM-master	`dm_master_port`	8261	DM-master service communication port

Component	Port variable	Default port	Description
DM-worker	`dm_worker_port`	8262	DM-worker service communication port
Prometheus	`prometheus_port`	9090	Prometheus service communication port
Grafana	`grafana_port`	3000	The port for the external service of web monitoring service and client (browser) access

Component	Port variable	Default port	Description
Alertmanager	alertmanager_port	9093	Alertmanager service communication port

6.1.1.12.2 Customize ports

Edit the `inventory.ini` file and add the related host variable of the corresponding service port after the service IP:

```
dm_master ansible_host=172.16.10.71 dm_master_port=18261
```

6.1.1.12.3 Update DM-Ansible

1. Log in to the Control Machine using the `tidb` account, enter the `/home/tidb` directory, and back up the `dm-ansible` folder.

```
cd /home/tidb &&
mv dm-ansible dm-ansible-bak
```

2. Download the specified version of DM-Ansible and extract it.

```
cd /home/tidb &&
wget http://download.pingcap.org/dm-ansible-{version}.tar.gz &&
tar -xzvf dm-ansible-{version}.tar.gz &&
mv dm-ansible-{version} dm-ansible
```

3. Migrate the `inventory.ini` configuration file.

```
cd /home/tidb &&
cp dm-ansible-bak/inventory.ini dm-ansible/inventory.ini
```

4. Migrate the `dmctl` configuration.

```
cd /home/tidb/dm-ansible-bak/dmctl &&
cp * /home/tidb/dm-ansible/dmctl/
```

5. Use Playbook to download the latest DM binary file, which substitutes for the binary file in the `/home/tidb/dm-ansible/resource/bin/` directory automatically.

```
ansible-playbook local_prepare.yml
```

6.1.2 Deploy Data Migration Cluster Using DM Binary

This document introduces how to quickly deploy the Data Migration (DM) cluster using DM binary.

6.1.2.1 Preparations

Download the official binary using the download link in the following table:

Package name	OS	Architecture	SHA256 checksum
`https://download.pingcap.org/dm-{version}-linux-amd64.tar.gz`	Linux	amd64	`https://download.pingcap.org/dm-{version}-linux-amd64.sha256`

Note:

`{version}` in the above download link indicates the version number of TiDB. For example, the download link for `v1.0.1` is `https://download.pingcap.`

> ↪ `org/dm-v1.0.1-linux-amd64.tar.gz`. You can check the published DM versions in the DM Release page.

The downloaded files have two subdirectories, `bin` and `conf`. The `bin` directory contains the binary files of DM-master, DM-worker, dmctl and Mydumper. The `conf` directory contains the sample configuration files.

6.1.2.2 Sample scenario

Suppose that you are going to deploy a DM cluster based on this sample scenario:

- Two MySQL instances are deployed on two servers.
- One TiDB instance is deployed on one server (in the mocktikv mode).
- Two DM-worker nodes and one DM-master node are deployed on three servers.

Here is the address of each node:

Instance or node	Server address
MySQL1	192.168.0.1
MySQL2	192.168.0.2
TiDB	192.168.0.3
DM-master	192.168.0.4
DM-worker1	192.168.0.5
DM-worker2	192.168.0.6

Enable the binlog on MySQL1 and on MySQL2. DM-worker1 migrates the data from MySQL1 and DM-worker2 migrates the data from MySQL2.

Based on this scenario, the following sections describe how to deploy the DM cluster.

6.1.2.2.1 Deploy DM-worker

Establish the connection between DM-worker and the upstream MySQL instances, and for safety reasons, you must configure the encrypted password.

Encrypt the MySQL password by executing the following command. Suppose the password is "123456".

```
./bin/dmctl --encrypt "123456"
```

Then, you get the encrypted password as shown below. Record this encrypted value, which is used for deploying DM-worker in the following steps.

```
fCxfQ9XKCezSzuCDOWf5dUD+LsKegSg=
```

You can configure DM-worker by using command-line parameters or the configuration file.

Deployment method 1: DM-worker command-line parameters

Below is the description of the DM-worker command-line parameters:

```
./bin/dm-worker --help
```

```
Usage of worker:
  -L string
        Log level. Available values: "debug", "info" (default value), "warn",
            ↪  "error" or "fatal"
  -V    The output version number
  -checker-backoff-max duration
        The longest waiting time for the automatic recovery after errors are
            ↪ found in the task check module. The default value is "5m0s"
            ↪ which generally needs no change. It is not recommended to
            ↪ change this default value unless you have an in-depth
            ↪ understanding of this parameter.
  -checker-backoff-rollback duration
        The time interval for adjusting the waiting time of the automatic
            ↪ recovery in the task check module. The default value is "5m0s"
            ↪  which generally needs no change. It is not recommended to
            ↪ change this default value unless you have an in-depth
            ↪ understanding of this parameter.
  -checker-check-enable
        Enables or disables the task status check. When it is enabled, DM
            ↪ automatically tries to resume the data migration tasks
            ↪ interrupted by errors. Default value: "true".
  -config string
        The path of the configuration file
  -log-file string
        The path of log files
  -print-sample-config
        Prints the sample configuration
  -purge-expires int
        The expiration time of relay logs. DM-worker tries to delete the
            ↪ relay logs whose last modified time exceeds this value. Unit:
            ↪ hour.
  -purge-interval int
        The time interval at which relay logs are regularly checked for
            ↪ expiration. Default value: "3600". Unit: second.
  -purge-remain-space int
```

```
      Sets the minimum available disk space. When the disk space is smaller
         ↪  than this value, DM-worker tries to delete relay logs.
         ↪ Default value: "15". Unit: GB.
  -relay-dir string
      The path in which relay logs are stored. Default value: "./relay_log
         ↪ ".
  -worker-addr string
      DM-worker address
```

> **Note:**
>
> In some situations, you cannot use the above method to configure DM-worker because some configurations are not exposed to the command line. Then use the configuration file instead.

Deployment method 2: configuration file

Below is the DM-worker configuration file. It is recommended that you use this method and write the following configuration to `conf/dm-worker1.toml`.

```
### Worker Configuration.

### Log configuration
log-level = "info"
log-file = "dm-worker.log"

### DM-worker address
worker-addr = ":8262"

### The server ID of MySQL secondary, used when pulling binlog from MySQL
### In a migration group, each instance (primary and secondary) must have a
   ↪ unique server ID
server-id = 101
### In v1.0.2 and later versions, the server ID is automatically generated
   ↪ by DM

### Used to mark a migration group or MySQL/MariaDB instance
source-id = "mysql-replica-01"

### The type of the upstream instance
### Available values: "mysql", "mariadb"
###  In v1.0.2 and later versions, DM automatically detects and fills in the
   ↪  type of the upstream instance
```

```
flavor = "mysql"

### MySQL connection address
[from]
host = "192.168.0.1"
user = "root"
password = "fCxfQ9XKCezSzuCD0Wf5dUD+LsKegSg="
port = 3306
```

Then, execute the following command in the terminal to run DM-worker:

```
bin/dm-worker -config conf/dm-worker1.toml
```

In DM-worker2, change `source-id` in the configuration file to `mysql-replica-02` and change the `from` configuration to the address of MySQL2. If you deploy Dm-worker2 and Dm-worker1 on one machine, you need to deploy two dm-worker instances in different paths, otherwise the default path for saving meta-information and relay log will conflict.

6.1.2.2.2 Deploy DM-master

You can configure DM-master by using command-line parameters or the configuration file.

Deployment method 1: DM-master command-line parameters

Below is the description of DM-master command-line parameters:

```
./bin/dm-master --help
```

```
Usage of dm-master:
  -L string
        Log level. Available values: "debug", "info" (default value), "warn",
          ↪  "error" or "fatal"
  -V    Outputs the version information
  -config string
        The path of the configuration file
  -log-file string
        The path of log files
  -master-addr string
        DM-master address
  -print-sample-config
        Prints the sample configuration of DM-master
```

Note:

> In some situations, you cannot use the above method to configure DM-master because some configurations are not exposed to the command line. Then use the configuration file instead.

Deployment method 2: configuration file

Below is the configuration file of DM-master. It is recommended that you use this method and write the following configuration to `conf/dm-master.toml`.

```
### Master Configuration.

### Log configurations
log-level = "info"
log-file = "dm-master.log"

### The listening address of DM-master
master-addr = ":8261"

### DM-worker configuration
[[deploy]]
### Corresponding to the source-id in the DM-worker1 configuration file
source-id = "mysql-replica-01"
### The service address of DM-worker1
dm-worker = "192.168.0.5:8262"

[[deploy]]
### Corresponding to the source-id in the DM-worker2 configuration file
source-id = "mysql-replica-02"
### The service address of DM-worker1
dm-worker = "192.168.0.6:8262"
```

Then, execute the following command in the terminal to run DM-master:

```
bin/dm-master -config conf/dm-master.toml
```

Now, a DM cluster is successfully deployed.

6.1.2.2.3 Create a data migration task

Suppose that there are several sharded tables on both MySQL1 and MySQL2 instances. These tables have the same structure and the same prefix “t” in their table names. The databases where they are located are named with the same prefix “sharding”. In each sharded table, the primary key and unique key are different from those of all other tables.

Now you need to migrate these sharded tables to the `db_target.t_target` table in TiDB.

1. Create the configuration file of the task:

```
---
name: test
task-mode: all
is-sharding: true

target-database:
  host: "192.168.0.3"
  port: 4000
  user: "root"
  password: "" # if the password is not empty, you also need to
      ↪ configure the encrypted password using dmctl.

mysql-instances:
  - source-id: "mysql-replica-01"
    block-allow-list: "instance" # Use black-white-list if the DM's
        ↪ version <= v1.0.6.
    route-rules: ["sharding-route-rules-table", "sharding-route-rules-
        ↪ schema"]
    mydumper-thread: 4          # The number of threads that the dump
        ↪ unit uses for dumping data, new in v1.0.2 and later versions.
    loader-thread: 16           # The number of threads that the load
        ↪ unit uses for loading data, new in v1.0.2 and later versions.
    syncer-thread: 16           # The number of threads that the sync
        ↪ unit uses for replicating incremental data, new in v1.0.2 and
        ↪  later versions.

  - source-id: "mysql-replica-02"
    block-allow-list: "instance" # Use black-white-list if the DM's
        ↪ version <= v1.0.6.
    route-rules: ["sharding-route-rules-table", "sharding-route-rules-
        ↪ schema"]
    mydumper-thread: 4
    loader-thread: 16
    syncer-thread: 16

block-allow-list: # Use black-white-list if the DM's version <= v1.0.6.
  instance:
    do-dbs: ["~^sharding[\\d]+"]
    do-tables:
    -  db-name: "~^sharding[\\d]+"
       tbl-name: "~^t[\\d]+"

routes:
```

```
  sharding-route-rules-table:
    schema-pattern: sharding*
    table-pattern: t*
    target-schema: db_target
    target-table: t_target

  sharding-route-rules-schema:
    schema-pattern: sharding*
    target-schema: db_target
```

2. Write the above configuration to the `conf/task.yaml` file and create the task using dmctl:

```
bin/dmctl -master-addr 192.168.0.4:8261
```

```
Welcome to dmctl
Release Version: v1.0.0-69-g5134ad1
Git Commit Hash: 5134ad19fbf6c57da0c7af548f5ca2a890bddbe4
Git Branch: master
UTC Build Time: 2019-04-29 09:36:42
Go Version: go version go1.12 linux/amd64
»
```

```
» start-task conf/task.yaml
```

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "192.168.0.5:8262",
            "msg": ""
        },
        {
            "result": true,
            "worker": "192.168.0.6:8262",
            "msg": ""
        }
    ]
}
```

Now, you have successfully created a task to migrate the sharded tables from the MySQL1 and MySQL2 instances to TiDB.

6.1.3 Use Kubernetes (Experimental)

6.2 Migrate Data Using Data Migration

This guide shows how to migrate data using the Data Migration (DM) tool.

6.2.1 Step 1: Deploy the DM cluster

It is recommended to deploy the DM cluster using DM-Ansible. For detailed deployment, see Deploy Data Migration Using DM-Ansible.

You can also deploy the DM cluster using binary for trial or test. For detailed deployment, see Deploy Data Migration Cluster Using DM Binary.

Note:

- For database passwords in all the DM configuration files, use the passwords encrypted by `dmctl`. If a database password is empty, it is unnecessary to encrypt it. See Encrypt the upstream MySQL user password using dmctl.
- The user of the upstream and downstream databases must have the corresponding read and write privileges.

6.2.2 Step 2: Check the cluster information

After the DM cluster is deployed using DM-Ansible, the configuration information is like what is listed below.

- The configuration information of related components in the DM cluster:

Component	Host	Port
dm_worker1	172.16.10.72	8262
dm_worker2	172.16.10.73	8262
dm_master	172.16.10.71	8261

- The information of upstream and downstream database instances:

Database instance	Host	Port	Username	Encrypted password
Upstream MySQL-1	172.16.10.81	3306	root	VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=
Upstream MySQL-2	172.16.10.82	3306	root	VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=

Database instance	Host	Port	Username	Encrypted password
Downstream TiDB	172.16.10.83	4000	root	

- The configuration in the DM-master process configuration file `{ansible deploy}/conf/dm-master.toml`:

```
# Master configuration.

# This indicates that whether DM-worker uses Global Transaction Identifier (GTID) to pull binlog. Before you use this configuration item, make sure that the GTID mode is enabled in the upstream MySQL.
enable-gtid = false

[[deploy]]
source-id = "mysql-replica-01"
dm-worker = "172.16.10.72:8262"

[[deploy]]
source-id = "mysql-replica-02"
dm-worker = "172.16.10.73:8262"
```

> **Note:**
>
> The `{ansible deploy}` in `{ansible deploy}/conf/dm-master.toml` indicates the directory where DM-Ansible is deployed. It is the directory configured in the `deploy_dir` parameter.

6.2.3 Step 3: Configure the data migration task

The following example assumes that you need to migrate all the `test_table` table data in the `test_db` database of both the upstream MySQL-1 and MySQL-2 instances, to the downstream `test_table` table in the `test_db` database of TiDB, in the full data plus incremental data mode.

Copy the `{ansible deploy}/conf/task.yaml.example` file and edit it to generate the `task.yaml` task configuration file as below:

```
## The task name. You need to use a different name for each of the multiple tasks that
## run simultaneously.
name: "test"
## The full data plus incremental data (all) migration mode.
task-mode: "all"
```

```
## The downstream TiDB configuration information.
target-database:
  host: "172.16.10.83"
  port: 4000
  user: "root"
  password: ""

## Configuration of all the upstream MySQL instances required by the current
    ↪  data migration task.
mysql-instances:
-
  # The ID of upstream instances or the migration group. You can refer to
      ↪ the configuration of `source_id` in the "inventory.ini" file or in
      ↪ the "dm-master.toml" file.
  source-id: "mysql-replica-01"
  # The configuration item name of the block and allow lists of the name of
      ↪ the
  # database/table to be migrated, used to quote the global block and allow
  # lists configuration that is set in the global block-allow-list below.
  block-allow-list: "global" # Use black-white-list if the DM's version <=
      ↪ v1.0.6.
  # The configuration item name of the dump unit, used to quote the global
      ↪ dump unit configuration.
  mydumper-config-name: "global"

-
  source-id: "mysql-replica-02"
  block-allow-list: "global" # Use black-white-list if the DM's version <=
      ↪ v1.0.6.
  mydumper-config-name: "global"

## The global configuration of block and allow lists. Each instance can
    ↪ quote it by the
## configuration item name.
block-allow-list:                 # Use black-white-list if the DM's version
    ↪  <= v1.0.6.
  global:
    do-tables:                    # The allow list of upstream tables to be
        ↪ migrated.
    - db-name: "test_db"          # The database name of the table to be
        ↪ migrated.
      tbl-name: "test_table"      # The name of the table to be migrated.

## The global configuration of the dump unit. Each instance can quote it by
    ↪ the configuration item name.
```

```
mydumpers:
  global:
    mydumper-path: "./bin/mydumper" # The file path of the dump unit binary.
    extra-args: "-B test_db -T test_table" # Extra arguments of the dump
        ↪ unit. Since DM 1.0.2, DM automatically generates the "--tables-
        ↪ list" configuration. For versions earlier than 1.0.2, you need to
        ↪ configure this option manually.
```

6.2.4 Step 4: Start the data migration task

To detect possible errors of data migration configuration in advance, DM provides the precheck feature:

- DM automatically checks the corresponding privileges and configuration while starting the data migration task.
- You can also use the `check-task` command to manually precheck whether the upstream MySQL instance configuration satisfies the DM requirements.

For details about the precheck feature, see Precheck the upstream MySQL instance configuration.

> **Note:**
>
> Before starting the data migration task for the first time, you should have got the upstream configured. Otherwise, an error is reported while you start the task.

1. Come to the dmctl directory `/home/tidb/dm-ansible/resources/bin/`.
2. Run the following command to start dmctl.

```
./dmctl --master-addr 172.16.10.71:8261
```

3. Run the following command to start the data migration tasks.

```
# `task.yaml` is the configuration file that is edited above.
start-task ./task.yaml
```

   - If the above command returns the following result, it indicates the task is successfully started.

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "172.16.10.72:8262",
            "msg": ""
        },
        {
            "result": true,
            "worker": "172.16.10.73:8262",
            "msg": ""
        }
    ]
}
```

- If you fail to start the data migration task, modify the configuration according to the returned prompt and then run the `start-task task.yaml` command to restart the task.

6.2.5 Step 5: Check the data migration task

If you need to check the task state or whether a certain data migration task is running in the DM cluster, run the following command in dmctl:

```
query-status
```

6.2.6 Step 6: Stop the data migration task

If you do not need to migrate data any more, run the following command in dmctl to stop the task:

```
## `test` is the task name that you set in the `name` configuration item
    ↪ of
## the `task.yaml` configuration file.
stop-task test
```

6.2.7 Step 7: Monitor the task and check logs

Assuming that Prometheus, Alertmanager, and Grafana are successfully deployed along with the DM cluster deployment using DM-Ansible, and the Grafana address is `172.16.10.71`. To view the alert information related to DM, you can open

http://172.16.10.71:9093 in a browser and enter into Alertmanager; to check monitoring metrics, go to http://172.16.10.71:3000, and choose the DM dashboard.

While the DM cluster is running, DM-master, DM-worker, and dmctl output the monitoring metrics information through logs. The log directory of each component is as follows:

- DM-master log directory: It is specified by the `--log-file` DM-master process parameter. If DM is deployed using DM-Ansible, the log directory is `{ansible deploy}/log/dm-master.log` in the DM-master node.
- DM-worker log directory: It is specified by the `--log-file` DM-worker process parameter. If DM is deployed using DM-Ansible, the log directory is `{ansible deploy}/log/dm-worker.log` in the DM-worker node.
- dmctl log directory: It is the same as the binary directory of dmctl.

7 Configure

7.1 Data Migration Configuration File Overview

This document gives an overview of configuration files of DM (Data Migration).

7.1.1 DM process configuration files

- `inventory.ini`: The configuration file of deploying DM using DM-Ansible. You need to edit it based on your machine topology. For details, see Edit the `inventory.ini` file to orchestrate the DM cluster.
- `dm-master.toml`: The configuration file of running the DM-master process, including the topology information of the DM cluster and the corresponding relationship between the MySQL instance and DM-worker (must be one-to-one relationship). When you use DM-Ansible to deploy DM, `dm-master.toml` is generated automatically. Refer to DM-master Configuration File to see more details.
- `dm-worker.toml`: The configuration file of running the DM-worker process, including the upstream MySQL instance configuration and the relay log configuration. When you use DM-Ansible to deploy DM, `dm-worker.toml` is generated automatically. Refer to DM-worker Configuration File to see more details.

7.1.2 DM migration task configuration

7.1.2.1 DM task configuration file

When you use DM-Ansible to deploy DM, you can find the following task configuration file template in `<path-to-dm-ansible>/conf`:

- `task.yaml.exmaple`: The standard configuration file of the data migration task (a specific task corresponds to a `task.yaml`). For the introduction of the configuration file, see Task Configuration File.

7.1.2.2 Data migration task creation

You can perform the following steps to create a data migration task based on `task.yaml .example`:

1. Copy `task.yaml.example` as `your_task.yaml`.
2. Refer to the description in the Task Configuration File and modify the configuration in `your_task.yaml`.
3. Create your data migration task using dmctl.

7.1.2.3 Important concepts

This section shows description of some important concepts.

Concept	Description	Configuration File
`source -id`	Uniquely identifies a MySQL or MariaDB instance, or a replication group with the primary-secondary structure. The maximum length of `source-id` is 32.	`source_id` of `inventory. ini`; `source-id` of `dm-master. toml`; `source-id` of `task.yaml`
DM-worker ID	Uniquely identifies a DM-worker (by the `worker- addr` parameter of `dm-worker .toml`)	`worker-addr` of `dm-worker. toml`; the `-worker`/`-w` flag of the dmctl command line

7.2 DM-master Configuration File

This document introduces the configuration of DM-master, including the configuration file template and configurable items.

7.2.1 Configuration file template

The following is a configuration file template of DM-master.

```
## log configuration
log-file = "dm-master.log"

## DM-master listening address
master-addr = ":8261"

## DM-worker deployment. It will be refined when the new deployment function
    ↪ is available.
[[deploy]]
source-id = "mysql-replica-01"
dm-worker = "172.16.10.72:8262"

[[deploy]]
source-id = "mysql-replica-02"
dm-worker = "172.16.10.73:8262"
```

7.2.2 Configurable items

7.2.2.1 Global configuration

Name	Description
`log-file`	The log file. If not specified, the log is printed to the standard output.
`master-addr`	The address of DM-master which provides services. You can omit the IP address and specify the port number only, such as ":8261".

7.2.2.2 DM-worker configuration

Each DM-worker must be configured in separate `[deploy]` sections.

Name	Description
`source-id`	Uniquely identifies a MySQL or MariaDB instance, or a replication group with the primary-secondary structure, which needs to be consistent with the `source-id` of DM-worker.
`dm-worker`	The service address of DM-worker.

7.3 DM-worker Configuration File

This document introduces the basic configuration of DM worker, which provisions DM-worker's deployment in most scenarios. Refer to DM-worker Advanced Configuration File to see more parameters in detail.

7.3.1 Configuration file template

```
## Worker Configuration.

## Log configuration.
log-file = "dm-worker.log"

## DM-worker listen address.
worker-addr = ":8262"

## Represents a MySQL/MariaDB instance or a migration group.
source-id = "mysql-replica-01"

## Server ID of secondary library for binlog replication.
## Each instance (primary and secondary) in migration groups should have a
   ↪ different server ID.
server-id = 101

## flavor: mysql/mariadb
flavor = "mysql"

## The directory that used to store relay log.
relay-dir = "./relay_log"

[from]
host = "127.0.0.1"
user = "root"
password = "Up8156jArvIPymkVC+5LxkAT6rek"
port = 3306
```

7.3.2 Configuration parameters

7.3.2.1 Global

Parameter	Description	Default value
`log-file`	Specifies the log file directory. If not specified, the logs are printed onto the standard output.	
`worker-addr`	Specifies the address of DM-worker which provides services. You can omit the IP address and specify the port number only, such as ":8262".	
`source-id`	Uniquely identifies a MySQL or MariaDB instance, or a replication group	
`server-id`	Identifies the server ID of DM-worker as a MySQL or MariaDB secondary library, used when pulling binlogs from the upstream. In a replication group, each instance (primary and secondary included) must have a unique server ID. In v1.0.2 and later versions, the `server_id` is automatically generated by DM.	
`flavor`	Indicates the release type of MySQL: `"Percona"`, `"mysql"` or `"mariadb"`. In v1.0.2 and later versions, DM automatically detects and fills in the release type.	`"mysql"`
`relay-dir`	Specifies the relay log directory.	`"./relay_log"`

7.3.2.2 `[from]`

The `[from]` section contains parameters that affect the connection to the upstream database.

Parameter	Description
`host`	The host name of the upstream database.
`port`	The port number of the upstream database.
`user`	The username used to connect to the database.

Parameter	Description
password	The password used to connect to the database. Note: Use the password encrypted by dmctl.

7.4 DM-worker Advanced Configuration File

This document details the advanced configuration of DM-worker.

7.4.1 Configuration file template

```
## Worker Configuration.

## Log configuration.
log-level = "info"
log-file = "dm-worker.log"

## DM-worker listening address.
worker-addr = ":8262"

## Represents a MySQL/MariaDB instance or a replication group.
source-id = "mysql-replica-01"

## Server ID of secondary library for binlog replication.
## Each instance (primary and secondary) in the replication group should
   ↪ have a different server ID.
server-id = 101

## flavor: mysql/mariadb
flavor = "mysql"

## The directory used to store relay log.
relay-dir = "./relay_log"

## Enables gtid in the relay log unit.
enable-gtid = false

relay-binlog-name = ""
relay-binlog-gtid = ""

[from]
host = "127.0.0.1"
user = "root"
```

```
password = "Up8156jArvIPymkVC+5LxkAT6rek"
port = 3306

## Relay log purge strategy.
[purge]
interval = 3600
expires = 24
remain-space = 15

## Task status checker.
[checker]
check-enable = true
backoff-rollback = "5m"
backoff-max = "5m"
```

7.4.2 Configuration parameters

7.4.2.1 Global

Parameter	Description	Default value
`log-level`	Controls the log level: `"debug"`, `"info"`, `"warn"`, `"error"` or `"fatal"`. For troubleshooting purposes, set it to `"debug"`.	`"info"`
`log-file`	Specifies the log file. If not specified, the logs are printed onto the standard output.	
`worker-addr`	Specifies the address of DM-worker which provides services. You can omit the IP address and specify the port number only, such as ":8262".	
`source-id`	Uniquely identifies a MySQL or MariaDB instance, or a replication group	
`server-id`	Identifies the server ID of DM-worker as a MySQL or MariaDB secondary library. In a replication group, each instance (primary and secondary included) must have a unique server ID. In v1.0.2 and later versions, the `server_id` is automatically generated by DM.	

Parameter	Description	Default value
`flavor`	Indicates the release type of MySQL: `"Percona"`, `"mysql"` or `"mariadb"`. In v1.0.2 and later versions, DM automatically detects and fills in the release type.	`"mysql"`
`relay-dir`	Specifies the relay log directory.	`"./relay_log"`
`enable-gtid`	Determines whether DM-worker uses GTID to pull the binlog. If the upstream database has enabled the GTID mode and switching the DM-worker connection to another MySQL instance is needed, set it to `true`.	`false`
`relay-binlog-name`	Specifies the file name from which DM-worker starts to pull the binlog. For example, `"mysql-bin.000002"`. It only works when `enable_gtid` is `false`. If this parameter is not specified, DM-worker defaults to pulling the binlogs starting from the earliest one. But in v1.0.2 and later versions, DM-worker defaults to pulling the binlogs starting from the latest one.	
`relay-binlog-gtid`	Specifies the GTID from which DM-worker starts to pull the binlog. For example, `"e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849"`. It only works when `enable_gtid` is `true`. If this parameter is not specified, DM-worker defaults to pulling the binlogs starting from the earliest GTID. But in v1.0.2 and later versions, DM-worker defaults to pulling the binlogs starting from the latest GTID.	

7.4.2.2 `[from]`

The `[from]` section contains parameters that affect the connection to the upstream database.

Parameter	Description
`host`	The host name of the upstream database.
`port`	The port number of the upstream database.
`user`	The username used to connect to the database.
`password`	The password used to connect to the database. Note: Use the password encrypted by dmctl.

7.4.2.3 `[purge]`

The `[purge]` section contains parameters that affect the purge strategy of relay log.

Generally, there is no need to manually configure these parameters unless there is a large amount of relay logs and disk capacity is insufficient.

Parameter	Description	Default value
`interval`	Sets the time interval at which relay logs are regularly checked for expiration, in seconds.	`3600`
`expires`	Sets the expiration time for relay logs, in hours. The relay log that is not written by the relay processing unit, or does not need to be read by the existing data migration task will be deleted by DM if it exceeds the expiration time. If this parameter is not specified, the automatic purge is not performed.	`0`
`remain-space`	Sets the minimum amount of free disk space, in gigabytes. When the available disk space is smaller than this value, DM-worker tries to delete relay logs.	`15`

Note:

DM does not perform automatic purge when either of the following is true:

- `interval` is set to `0`
- Both `expires` and `remain-space` are set to `0`

7.4.2.4 `[checker]`

The [checker] section contains parameters that affect the task status checker.

Parameter	Description	Default value
check-enable	Determines whether to enable task status checker. If it is set to "true", DM tries to resume data migration task that is suspended due to errors.	true
backoff-rollback	Sets the time interval for adjusting the waiting time of the automatic recovery.	"5m0s"
backoff-max	Sets the longest time interval for the automatic recovery after errors are detected.	"5m0s"

Note:

Generally, you only need to determine whether to enable the task status checker through the check-enable parameter. It is not recommended to change the default values of backoff-rollback and backoff-max unless you have an in-depth understanding of these two parameters.

7.5 Data Migration Task Configuration File

This document introduces the basic task configuration file of Data Migration – task_basic.yaml, including global configuration and instance configuration.

DM also implements an advanced task configuration file which provides greater flexibility and more control over DM.

For the feature and configuration of each configuration item, see Data migration features.

7.5.1 Important concepts

For description of important concepts including source-id and the DM-worker ID, see Important concepts.

7.5.2 Task configuration file template (basic)

The following is a task configuration file template which allows you to perform basic data migration tasks.

7.6 “‘yaml

7.7 ———— Global configuration ————

7.7.1 ********** Basic configuration **********

```
name: test                    # The name of the task. Should be globally
   ↪ unique.
task-mode: all                # The task mode. Can be set to `full`/`
   ↪ incremental`/`all`.

target-database:              # Configuration of the downstream database
   ↪ instance.
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: ""                # The dmctl encryption is needed when the
     ↪ password is not empty.

### ******** Feature configuration set **********
## The filter rule set of the block and allow list of the matched table of
   ↪ the upstream database instance.
block-allow-list:     # Use black-white-list if the DM's version <= v1.0.6.
  bw-rule-1:          # The name of the block and allow lists filtering rule
     ↪  of the table matching the upstream database instance.
    do-dbs: ["all_mode"] # Allow list of upstream tables needs to be
       ↪ migrated
## ----------- Instance configuration -----------
mysql-instances:
  # The ID of the upstream instance or migration group. It can be configured
     ↪  by referring to the `source-id` in the `dm-master.toml` file.
  - source-id: "mysql-replica-01"
    block-allow-list: "bw-rule-1"  # Use black-white-list if the DM's
       ↪ version <= v1.0.6.
       mydumper-thread: 4         # The number of threads that the dump
          ↪ unit uses for dumping data, new in v1.0.2 and later versions
       loader-thread: 16          # The number of threads that the load
          ↪ unit uses for loading data, new in v1.0.2 and later versions
       syncer-thread: 16          # The number of threads that the sync
          ↪ unit uses for replicating incremental data, new in v1.0.2 and
          ↪ later versions
  - source-id: "mysql-replica-02"
    block-allow-list: "bw-rule-1"  # Use black-white-list if the DM's
       ↪ version <= v1.0.6.
    mydumper-thread: 4
```

```
  loader-thread: 16
  syncer-thread: 16
```

7.7.2 Configuration order

1. Edit the global configuration.
2. Edit the instance configuration based on the global configuration.

7.7.3 Global configuration

7.7.3.1 Basic configuration

Refer to the comments in the template to see more details. Specific instruction about `task-mode` are as follows:

- Description: the task mode that can be used to specify the data migration task to be executed.
- Value: string (`full`, `incremental`, or `all`).
  - `full` only makes a full backup of the upstream database and then imports the full data to the downstream database.
  - `incremental`: Only replicates the incremental data of the upstream database to the downstream database using the binlog. You can set the `meta` configuration item of the instance configuration to specify the starting position of incremental replication.
  - `all`: `full` + `incremental`. Makes a full backup of the upstream database, imports the full data to the downstream database, and then uses the binlog to make an incremental replication to the downstream database starting from the exported position during the full backup process (binlog position).

7.7.3.2 Feature configuration set

For basic applications, you only need to modify the block and allow lists filtering rule. Refer to the comments about `block-allow-list` in the template or Block & allow table lists to see more details.

7.7.4 Instance configuration

This part defines the subtask of data migration. DM supports migrating data from one or multiple MySQL instances to the same instance.

For more details, refer to the comments about `mysql-instances` in the template.

7.7.5 Modify the task configuration

In some cases, you might need to update the task configuration. For example, if you set `remove-meta` to `true` and `task-mode` to `all` when resetting the data migration task, you need to set `remove-meta` to `false` after the task is reset. This can prevent the task from being migrated the next time the task is started.

It is recommended to update the modified configuration to the DM cluster by executing the `stop-task` and `start-task` commands, since the DM cluster persists the task configuration. If the task configuration file is modified directly, without restarting the task, the configuration changes does not take effect. In this case, the DM cluster still reads the previous task configuration when the DM cluster is restarted.

To illustrate how to modify the task configuration, the following is an example of modifying `remove-meta`:

1. Modify the task configuration file and set `remove-meta` to `false`.
2. Stop the task by executing the `stop-task` command:

```
stop-task <task-name | task-file>
```

3. Start the task by executing the `start-task` command:

```
start-task <config-file>
```

7.8 DM Advanced Task Configuration File

This document introduces the advanced task configuration file of Data Migration – `task_advanced.yaml`, including global configuration and instance configuration.

For the feature and configuration of each configuration item, see Data migration features.

7.8.1 Important concepts

For description of important concepts including `source-id` and the DM-worker ID, see Important concepts.

7.8.2 Disable checking items

DM checks items according to the task type, see Disable checking items. You can use `ignore-checking-items` in the task configuration file to disable checking items.

7.8.3 Task configuration file template (advanced)

The following is the task configuration file template which allows you to perform **advanced** data migration tasks.

```
---

## ----------- Global setting -----------
### ********* Basic configuration *********
name: test                  # The name of the task. Should be globally
    ↪ unique.
task-mode: all              # The task mode. Can be set to `full`/`
    ↪ incremental`/`all`.
is-sharding: true           # Whether it is a task to merge shards.
meta-schema: "dm_meta"      # The downstream database that stores the `meta`
    ↪  information.
remove-meta: false          # Whether to remove the `meta` information (`
    ↪ checkpoint` and `onlineddl`) corresponding to the task name before
    ↪ starting the migration task.
enable-heartbeat: false     # Whether to enable the heartbeat feature.
online-ddl-scheme: "gh-ost" # Only "gh-ost" and "pt" are currently supported
    ↪ .
case-sensitive: false       # Whether schema/table is case-sensitive.
ignore-checking-items: []   # No element, which means not to disable any
    ↪ checking items.
clean-dump-file: true       # Whether to clean up the files generated during
    ↪  data dump. Note that these include `metadata` files. New in v1.0.7.

target-database:            # Configuration of the downstream database
    ↪ instance.
  host: "192.168.0.1"
  port: 4000
  user: "root"
  password: "/Q7B9DizNLLTTfiZHv9WoEAKamfpIUs=" # It is recommended to use a
      ↪ password encrypted with dmctl
  session:                                  # The session variables of TiDB,
      ↪ supported since v1.0.6. For details, go to `https://pingcap.com/docs
      ↪ /stable/system-variables`
    sql_mode: "ANSI_QUOTES,NO_ZERO_IN_DATE,NO_ZERO_DATE"
    tidb_skip_utf8_check: 1
    tidb_constraint_check_in_place: 0


### ******** Feature configuration set **********
## The routing mapping rule set between the upstream and downstream tables.
```

```
routes:
  route-rule-1:                # The name of the routing mapping rule
    schema-pattern: "test_*"   # The pattern of the upstream schema name,
       ↪ wildcard characters (*?) are supported
    table-pattern: "t_*"       # The pattern of the upstream table name,
       ↪ wildcard characters (*?) are supported
    target-schema: "test"      # The name of the downstream schema
    target-table: "t"          # The name of the downstream table
  route-rule-2:
    schema-pattern: "test_*"
    target-schema: "test"

## The binlog event filter rule set of the matched table of the upstream
   ↪ database instance.
filters:
  filter-rule-1:                           # The name of the filtering rule
    schema-pattern: "test_*"               # The pattern of the upstream
       ↪ schema name, wildcard characters (*?) are supported
    table-pattern: "t_*"                   # The pattern of the upstream
       ↪ schema name, wildcard characters (*?) are supported
    events: ["truncate table", "drop table"] # What event types to match
    action: Ignore                         # Whether to replicate (Do) or
       ↪ ignore (Ignore) the binlog that matches the filtering rule
  filter-rule-2:
    schema-pattern: "test_*"
    events: ["all dml"]
    action: Do

## The filter rule set of the block and allow list of the matched table of
   ↪ the upstream database instance.
block-allow-list:                # Use black-white-list if the DM's version
   ↪ <= v1.0.6.
  bw-rule-1:                     # The name of the block and allow list rule
    do-dbs: ["~^test.*", "user"] # The allow list of upstream schemas needs
       ↪ to be migrated
    ignore-dbs: ["mysql", "account"] # The block list of upstream schemas
       ↪ needs to be migrated
    do-tables:                   # The allow list of upstream tables needs
       ↪ to be migrated
    - db-name: "~^test.*"
      tbl-name: "~^t.*"
    - db-name: "user"
      tbl-name: "information"
    ignore-tables:               # The block list of upstream tables needs
       ↪ to be migrated
```

```
    - db-name: "user"
      tbl-name: "log"

## Configuration arguments of the dump processing unit
mydumpers:
  global:                       # The configuration name of the processing
     ↪ unit.
    # The binary file path of the dump unit ("./bin/mydumper" by default).
    mydumper-path: "./bin/mydumper"
    threads: 4                  # The number of the threads that the dump
       ↪ unit dumps from the upstream database (4 by default).
    chunk-filesize: 64          # The size of the file generated by the
       ↪ dump unit (64 in MB by default).
    skip-tz-utc: true           # Ignore timezone conversion for time type
       ↪ data (true by default).
    extra-args: "--no-locks"    # Other arguments of the dump unit. In v1
       ↪ .0.2 and later versions, DM automatically generates the table-list
       ↪  configurable items.

## Configuration arguments of the load processing unit
loaders:
  global:                       # The configuration name of the processing
     ↪ unit.
    pool-size: 16               # The number of threads that concurrently
       ↪ execute dumped SQL files in the load unit (16 by default).
    # The directory that the load unit reads from and the dump unit outputs
       ↪ SQL files to ("./dumped_data" by default). Directories for
       ↪ different tasks of the same instance must be different.
    dir: "./dumped_data"

## Configuration arguments of the sync processing unit
syncers:
  global:                       # The configuration name of the processing
     ↪ unit.
    worker-count: 16            # The number of threads that replicate
       ↪ binlog events concurrently in the sync unit.
    batch: 100                  # The number of SQL statements in a
       ↪ transaction batch that the sync unit replicates to the downstream
       ↪ database (100 by default).
    enable-ansi-quotes: true    # Enable this argument if `sql-mode: "
       ↪ ANSI_QUOTES"` is set in the `session`
    safe-mode: false            # If set to true, `INSERT` statements from
       ↪ upstream are rewritten to `REPLACE` statements, and `UPDATE`
       ↪ statements are rewritten to `DELETE` and `REPLACE` statements.
       ↪ This ensures that DML statements can be imported repeatedly during
```

```
      ↪  data migration when there is any primary key or unique index in
      ↪ the table schema. TiDB DM automatically enables safe mode within
      ↪ the first 5 minutes after starting or resuming migration tasks.

## ----------- Instance configuration -----------
mysql-instances:
  -
    source-id: "mysql-replica-01"                                 # The ID of
      ↪ the upstream instance or replication group. It can be configured
      ↪ by referring to the `source_id` in the `inventory.ini` file or the
      ↪  `source-id` in the `dm-master.toml` file.
    meta:                                                         # The
      ↪ position where the binlog replication starts when `task-mode` is `
      ↪ incremental` and the downstream database checkpoint does not exist
      ↪ . If the checkpoint exists, the checkpoint is used.

      binlog-name: binlog.000001
      binlog-pos: 4

    route-rules: ["route-rule-1", "route-rule-2"]                 # The name of
      ↪  the mapping rule between the table matching the upstream database
      ↪  instance and the downstream database.
    filter-rules: ["filter-rule-1"]                               # The name of
      ↪  the binlog event filtering rule of the table matching the
      ↪ upstream database instance.
    block-allow-list: "bw-rule-1"                                 # The name of
      ↪  the block and allow lists filtering rule of the table matching
      ↪ the upstream database instance. Use black-white-list if the DM's
      ↪ version <= v1.0.6.

    mydumper-config-name: "global"                                # The
      ↪ configuration name of the dump processing unit.
    loader-config-name: "global"                                  # The
      ↪ configuration name of the load processing unit.
    syncer-config-name: "global"                                  # The
      ↪ configuration name of the sync processing unit.

  -
    source-id: "mysql-replica-02"
    mydumper-config-name: "global"
    loader-config-name: "global"
    syncer-config-name: "global"
```

7.8.4 Configuration order

1. Edit the global configuration.
2. Edit the instance configuration based on the global configuration.

7.8.5 Global configuration

7.8.5.1 Basic configuration

Refer to the comments in the template to see more details. Detailed explanations about `task-mode` are as follows:

- Description: the task mode that can be used to specify the data migration task to be executed.
- Value: string (`full`, `incremental`, or `all`).
  - `full` only makes a full backup of the upstream database and then imports the full data to the downstream database.
  - `incremental`: Only replicates the incremental data of the upstream database to the downstream database using the binlog. You can set the `meta` configuration item of the instance configuration to specify the starting position of incremental replication.
  - `all`: `full` + `incremental`. Makes a full backup of the upstream database, imports the full data to the downstream database, and then uses the binlog to make an incremental replication to the downstream database starting from the exported position during the full backup process (binlog position).

7.8.5.2 Feature configuration set

Arguments in each feature configuration set are explained in the comments in the template.

Parameter	Description
`routes`	The routing mapping rule set between the upstream and downstream tables. If the names of the upstream and downstream schemas and tables are the same, this item does not need to be configured. See Table Routing for usage scenarios and sample configurations.
`filters`	The binlog event filter rule set of the matched table of the upstream database instance. If binlog filtering is not required, this item does not need to be configured. See Binlog Event Filter for usage scenarios and sample configurations.

Parameter	Description
`block-allow-list`	The filter rule set of the block and allow list of the matched table of the upstream database instance. It is recommended to specify the schemas and tables that need to be migrated through this item, otherwise all schemas and tables are migrated. See Binlog Event FilterBlock & Allow Lists for usage scenarios and sample configurations.
`mydumpers`	Configuration arguments of the dump processing unit. If the default configuration is sufficient for your needs, this item does not need to be configured. Or you can configure `thread` only using `mydumper-thread`.
`loaders`	Configuration arguments of the load processing unit. If the default configuration is sufficient for your needs, this item does not need to be configured. Or you can configure `pool-size` only using `loader-thread`.
`syncers`	Configuration arguments of the sync processing unit. If the default configuration is sufficient for your needs, this item does not need to be configured. Or you can configure `worker-count` only using `syncer-thread`.

7.8.6 Instance configuration

This part defines the subtask of data migration. DM supports migrating data from one or multiple MySQL instances in the upstream to the same instance in the downstream.

For the configuration details of the above options, see the corresponding part in Feature configuration set, as shown in the following table.

Option	Corresponding part
`route-rules`	`routes`
`filter-rules`	`filters`
`block-allow-list`	`block-allow-list`
`mydumper-config-name`	`mydumpers`
`loader-config-name`	`loaders`
`syncer-config-name`	`syncers`

8 Manage the DM Cluster

8.1 Data Migration Cluster Operations

This document introduces the DM cluster operations and considerations when you administer a DM cluster using DM-Ansible.

8.1.1 Start a cluster

Run the following command to start all the components (including DM-master, DM-worker and the monitoring component) of the whole DM cluster:

```
ansible-playbook start.yml
```

8.1.2 Stop a cluster

Run the following command to stop all the components (including DM-master, DM-worker and the monitoring component) of the whole DM cluster:

```
ansible-playbook stop.yml
```

8.1.3 Restart cluster components

You need to update the DM cluster components in the following cases:

- You want to upgrade the component version.
- A serious bug occurs and you have to restart the component for temporary recovery.
- The machine that the DM cluster is located in is restarted for certain reasons.

8.1.3.1 Restarting considerations

This sections describes the considerations that you need to know when you restart DM components.

8.1.3.1.1 Restarting DM-worker considerations

In the process of full data loading:

For the SQL files during full data import, DM uses the downstream database to record the checkpoint information, and DM-worker records the subtask information in the local meta file. When DM-worker is restarted, it checks the checkpoint information and the subtask information in the local record, and the running task before restarting recovers the data migration automatically.

In the process of incremental data replication:

For the binlog during incremental data import, DM uses the downstream database to record the checkpoint information, and enables the safe mode within the first 5 minutes after the replication task is started or recovered.

- Sharding DDL statements migration is not enabled

  If the sharding DDL statements migration is not enabled in the task running on DM-worker, when DM-worker is restarted, it checks the checkpoint information and the subtask information in the local record, and the running task before restarting recovers the data migration automatically.

- Sharding DDL statements migration is enabled

  - When DM is migrating the sharding DDL statements, if DM-worker successfully executes (or skips) the sharding DDL binlog event, then the checkpoints of all tables related to sharding DDL in the DM-worker are updated to the position after the binlog event corresponding to the DDL statement.
  - When DM-worker is restarted before or after migrating sharding DDL statements, it recovers the data migration automatically according to the checkpoint information and the subtask information in the local record.
  - When DM-worker is restarted during the process of migrating sharding DDL statements, the issue might occur that the owner (one of DM-worker instances) has executed the DDL statement and successfully changed the downstream database table schema, while other DM-worker instances are restarted but fail to skip the DDL statement and update the checkpoints.

    At this time, DM tries again to migrate these DDL statements that are not skipped. However, the restarted DM-worker instances will be blocked at the position of the binlog event corresponding to the DDL binlog event, because the DM-worker instance that is not restarted has executed to the place after this DDL binlog event.

    To resolve this issue, follow the steps described in Handle Sharding DDL Locks Manually.

Conclusion: Try to avoid restarting DM-worker in the process of sharding DDL migration.

8.1.3.1.2 Restarting DM-master considerations

The information maintained by DM-master includes the following two major types, and these data is not being persisted when you restart DM-master.

- The corresponding relationship between the task and DM-worker
- The sharding DDL lock related information

When DM-master is restarted, it automatically requests the task information from each DM-worker instance, rebuilds the corresponding relationship between the task and DM-worker, and also re-fetches the sharding DDL information from each DM-worker instance. So the corresponding DDL lock can be correctly rebuilt and the sharding DDL lock can be automatically resolved.

8.1.3.2 Restart DM-worker

> **Note:**
>
> Try to avoid restarting DM-worker during the process of migrating sharding DDL statements.

To restart the DM-worker component, you can use either of the following two approaches:

- Perform a rolling update on DM-worker

```
ansible-playbook rolling_update.yml --tags=dm-worker
```

- Stop DM-worker first and then restart it

```
ansible-playbook stop.yml --tags=dm-worker &&
ansible-playbook start.yml --tags=dm-worker
```

8.1.3.3 Restart DM-master

To restart the DM-master component, you can use either of the following two approaches:

- Perform a rolling update on DM-master

```
ansible-playbook rolling_update.yml --tags=dm-master
```

- Stop DM-master first and then restart it

```
ansible-playbook stop.yml --tags=dm-master &&
ansible-playbook start.yml --tags=dm-master
```

8.1.4 Upgrade the component version

1. Download the DM binary file.

    1. Delete the existing file in the `downloads` directory.

    ```
    cd /home/tidb/dm-ansible &&
    rm -rf downloads
    ```

    2. Use Playbook to download the version of DM binary file as specified in `inventory.ini`, and replace the existing binary in the `/home/tidb/dm-ansible` ↪ `/resource/bin/` directory with it automatically.

    ```
    ansible-playbook local_prepare.yml
    ```

2. Use DM-Ansible to perform the rolling update.

    1. Perform a rolling update on the DM-worker instance:

    ```
    ansible-playbook rolling_update.yml --tags=dm-worker
    ```

    2. Perform a rolling update on the DM-master instance:

    ```
    ansible-playbook rolling_update.yml --tags=dm-master
    ```

    3. Upgrade dmctl:

    ```
    ansible-playbook rolling_update.yml --tags=dmctl
    ```

    4. Perform a rolling update on DM-worker, DM-master and dmctl:

    ```
    ansible-playbook rolling_update.yml
    ```

8.1.5 Add a DM-worker instance

Assuming that you want to add a DM-worker instance on the `172.16.10.74` machine and the alias of the instance is `dm_worker3`, perform the following steps:

1. Configure the SSH mutual trust and sudo rules on the Control Machine.

    1. Refer to Configure the SSH mutual trust and sudo rules on the Control Machine, log in to the Control Machine using the `tidb` user account and add `172.16.10.74` to the `[servers]` section of the `hosts.ini` file.

    ```
    cd /home/tidb/dm-ansible &&
    vi hosts.ini
    ```

```
[servers]
172.16.10.74

[all:vars]
username = tidb
```

2. Run the following command and enter the `root` user password for deploying `172.16.10.74` according to the prompt.

```
ansible-playbook -i hosts.ini create_users.yml -u root -k
```

This step creates a `tidb` user on the `172.16.10.74` machine, and configures sudo rules and the SSH mutual trust between the Control Machine and the `172.16.10.74` machine.

2. Edit the `inventory.ini` file and add the new DM-worker instance `dm_worker3`.

```
[dm_worker_servers]
dm_worker1 source_id="mysql-replica-01" ansible_host=172.16.10.72
    ↪ server_id=101 mysql_host=172.16.10.81 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306

dm_worker2 source_id="mysql-replica-02" ansible_host=172.16.10.73
    ↪ server_id=102 mysql_host=172.16.10.82 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306

dm_worker3 source_id="mysql-replica-03" ansible_host=172.16.10.74
    ↪ server_id=103 mysql_host=172.16.10.83 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
```

3. Deploy the new DM-worker instance.

```
ansible-playbook deploy.yml --tags=dm-worker -l dm_worker3
```

4. Start the new DM-worker instance.

```
ansible-playbook start.yml --tags=dm-worker -l dm_worker3
```

5. Configure and restart the DM-master service.

```
ansible-playbook rolling_update.yml --tags=dm-master
```

6. Configure and restart the Prometheus service.

```
ansible-playbook rolling_update_monitor.yml --tags=prometheus
```

8.1.6 Remove a DM-worker instance

Assuming that you want to remove the `dm_worker3` instance, perform the following steps:

1. Stop the DM-worker instance that you need to remove.

```
ansible-playbook stop.yml --tags=dm-worker -l dm_worker3
```

2. Edit the `inventory.ini` file and comment or delete the line where the `dm_worker3` instance exists.

```
[dm_worker_servers]
dm_worker1 source_id="mysql-replica-01" ansible_host=172.16.10.72
    ↪ server_id=101 mysql_host=172.16.10.81 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306

dm_worker2 source_id="mysql-replica-02" ansible_host=172.16.10.73
    ↪ server_id=102 mysql_host=172.16.10.82 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306

# dm_worker3 source_id="mysql-replica-03" ansible_host=172.16.10.74
    ↪ server_id=103 mysql_host=172.16.10.83 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
    ↪  # Comment or delete this line
```

3. Configure and restart the DM-master service.

```
ansible-playbook rolling_update.yml --tags=dm-master
```

4. Configure and restart the Prometheus service.

```
ansible-playbook rolling_update_monitor.yml --tags=prometheus
```

8.1.7 Replace/migrate a DM-master instance

Assuming that the `172.16.10.71` machine needs to be maintained or this machine breaks down, and you need to migrate the DM-master instance from `172.16.10.71` to `172.16.10.80`, perform the following steps:

1. Configure the SSH mutual trust and sudo rules on the Control machine.

    1. Refer to Configure the SSH mutual trust and sudo rules on the Control Machine, log in to the Control Machine using the `tidb` user account, and add `172.16.10.80` to the `[servers]` section of the `hosts.ini` file.

```
cd /home/tidb/dm-ansible &&
vi hosts.ini
```

```
[servers]
172.16.10.80

[all:vars]
username = tidb
```

2. Run the following command and enter the `root` user password for deploying `172.16.10.80` according to the prompt.

```
ansible-playbook -i hosts.ini create_users.yml -u root -k
```

This step creates the `tidb` user account on `172.16.10.80`, configures the sudo rules and the SSH mutual trust between the Control Machine and the `172.16.10.80` machine.

2. Stop the DM-master instance that you need to replace.

> **Note:**
>
> If the `172.16.10.71` machine breaks down and you cannot log in via SSH, ignore this step.

```
ansible-playbook stop.yml --tags=dm-master
```

3. Edit the `inventory.ini` file, comment or delete the line where the DM-master instance that you want to replace exists, and add the information of the new DM-master instance.

```
[dm_master_servers]
# dm_master ansible_host=172.16.10.71
dm_master ansible_host=172.16.10.80
```

4. Deploy the new DM-master instance.

```
ansible-playbook deploy.yml --tags=dm-master
```

5. Start the new DM-master instance.

```
ansible-playbook start.yml --tags=dm-master
```

6. Update the dmctl configuration file.

```
ansible-playbook rolling_update.yml --tags=dmctl
```

8.1.8 Replace/migrate a DM-worker instance

Assuming that the `172.16.10.72` machine needs to be maintained or this machine breaks down, and you need to migrate `dm_worker1` from `172.16.10.72` to `172.16.10.75`, perform the following steps:

1. Configure the SSH mutual trust and sudo rules on the Control Machine.

    1. Refer to Configure the SSH mutual trust and sudo rules on the Control Machine, log in to the Control Machine using the `tidb` user account, and add `172.16.10.75` to the `[servers]` section of the `hosts.ini` file.

    ```
    cd /home/tidb/dm-ansible &&
    vi hosts.ini
    ```

    ```
    [servers]
    172.16.10.75

    [all:vars]
    username = tidb
    ```

    2. Run the following command and enter the `root` user password for deploying `172.16.10.75` according to the prompt.

    ```
    ansible-playbook -i hosts.ini create_users.yml -u root -k
    ```

    This step creates the `tidb` user account on `172.16.10.75`, and configures the sudo rules and the SSH mutual trust between the Control Machine and the `172.16.10.75` machine.

2. Stop the DM-worker instance that you need to replace.

> **Note:**
>
> If the `172.16.10.72` machine breaks down and you cannot log in via SSH, ignore this step.

```
ansible-playbook stop.yml --tags=dm-worker -l dm_worker1
```

3. Edit the `inventory.ini` file and add the new DM-worker instance.

    Edit the `inventory.ini` file, comment or delete the line where the original `dm_worker1` instance (`172.16.10.72`) that you want to replace exists, and add the information for the new `dm_worker1` instance (`172.16.10.75`).

    To pull the relay log from a different binlog position or GTID Sets, you also need to update corresponding `{relay_binlog_name}` or `{relay_binlog_gtid}`.

```
[dm_worker_servers]
dm_worker1 source_id="mysql-replica-01" ansible_host=172.16.10.75
    ↪ server_id=101 mysql_host=172.16.10.81 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
# dm_worker1 source_id="mysql-replica-01" ansible_host=172.16.10.72
    ↪ server_id=101 mysql_host=172.16.10.81 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306

dm_worker2 source_id="mysql-replica-02" ansible_host=172.16.10.73
    ↪ server_id=102 mysql_host=172.16.10.82 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
```

4. Deploy the new DM-worker instance.

```
ansible-playbook deploy.yml --tags=dm-worker -l dm_worker1
```

5. Migrate the relay log.

    - If the `172.16.10.72` machine is still accessible, you can directly copy all data from the `{dm_worker_relay_dir}` directory to the corresponding directory of the new DM-worker instance.
    - If `172.16.10.72` machine is no longer accessible, you may need to manually recover data such as the relay log directories in Step 9.

6. Start the new DM-worker instance.

```
ansible-playbook start.yml --tags=dm-worker -l dm_worker1
```

7. Configure and restart the DM-master service.

```
ansible-playbook rolling_update.yml --tags=dm-master
```

8. Configure and restart the Prometheus service.

```
ansible-playbook rolling_update_monitor.yml --tags=prometheus
```

9. Start and verify data migration task.

    Execute `start-task` command to start data migration task. If no error is reported, then DM-worker migration completes successfully. If the following error is reported, you need to manually fix the relay log directory.

```
fail to initial unit Sync of subtask test-task : UUID suffix 000002
    ↪ with UUIDs [1ddbf6d3-d3b2-11e9-a4e9-0242ac140003.000001] not
    ↪ found
```

This error occurs because the upstream MySQL of the DM-worker instance to be replaced has been switched. You can fix this by following these steps:

1. Use `stop-task` to stop data migration task.
2. Use `ansible-playbook stop.yml --tags=dm-worker -l dm_worker1` to stop the DM-worker instance.
3. Update the suffix of the subdirectory of the relay log, such as renaming `1ddbf6d3-d3b2-11e9-a4e9-0242ac140003.000001` to `1ddbf6d3-d3b2-11e9-a4e9-0242ac140003.000002`.
4. Update the index file `server-uuid.index` in the subdirectory of the relay log, such as changing `1ddbf6d3-d3b2-11e9-a4e9-0242ac140003.000001` to `1ddbf6d3-d3b2-11e9-a4e9-0242ac140003.000002`.
5. Use `ansible-playbook start.yml --tags=dm-worker -l dm_worker1` to start the DM-worker instance.
6. Restart and verify data migration task.

8.2 Upgrade Data Migration

This document introduces how to upgrade your Data Migration (DM) version to an incompatible version.

Note:

- Unless otherwise stated, DM version upgrade means upgrading DM from the previous version with an upgrade procedure to the current version.
- Unless otherwise stated, all the following upgrade examples assume that you have downloaded the corresponding DM version and DM-Ansible version, and the DM binary exists in the corresponding directory of DM-Ansible. (For how to download the DM binary, see Upgrade the component version).
- Unless otherwise stated, all the following upgrade examples assume that all the data migration tasks have been stopped before the upgrade and all the migration tasks are restarted manually after DM upgrade is finished.
- The following shows the upgrade procedure of DM versions in reverse chronological order.

8.2.1 Upgrade to v1.0.3

8.2.1.1 Version information

```
Release Version: v1.0.3
Git Commit Hash: 41426af6cffcff9a325697a3bdebeadc9baa8aa6
Git Branch: release-1.0
UTC Build Time: 2019-12-13 07:04:53
Go Version: go version go1.13 linux/amd64
```

8.2.1.2 Main changes

- Add the command mode in dmctl
- Support migrating the `ALTER DATABASE` DDL statement
- Optimize the error message output
- Fix the panic-causing data race issue occurred when the full import unit pauses or exits
- Fix the issue that `stop-task` and `pause-task` might not take effect when retrying SQL operations to the downstream

8.2.1.3 Upgrade operation example

1. Download the new version of DM-Ansible, and confirm that there is `dm_version = v1.0.3` in the `inventory.ini` file.
2. Run `ansible-playbook local_prepare.yml` to download the new DM binary file to the local disk.
3. Run `ansible-playbook rolling_update.yml` to perform a rolling update for the DM cluster components.
4. Run `ansible-playbook rolling_update_monitor.yml` to perform a rolling update for the DM monitoring components.

> **Note:**
>
> When you upgrade DM to the 1.0.3 version, you must make sure that all DM cluster components (dmctl, DM-master, and DM-worker) are upgraded. Do not upgrade only a part of the components. Otherwise, an error might occur.

8.2.2 Upgrade to v1.0.2

8.2.2.1 Version information

```
Release Version: v1.0.2
Git Commit Hash: affc6546c0d9810b0630e85502d60ed5c800bf25
```

```
Git Branch: release-1.0
UTC Build Time: 2019-10-30 05:08:50
Go Version: go version go1.12 linux/amd64
```

8.2.2.2 Main changes

- Support automatically generating some configuration items for DM-worker to reduce manual configuration cost
- Support automatically generating the parameters of Mydumper database and tables to reduce manual configuration cost
- Optimize the default output of `query-status` to highlight important information
- Directly manage the DB connection to the downstream instead of using the built-in connection pool to optimize the handling of and retry for SQL errors
- Fix the panic that might occur when the DM-worker process is started or when the DML statement is failed to execute
- Fix the bug that the timeout of executing the sharding DDL statements (for example, `ADD INDEX`) might cause that the subsequent sharding DDL statements cannot be correctly coordinated
- Fix the bug that the `start-task` command cannot be executed when some DM-workers are inaccessible
- Improve the automatic retry policy for the `1105` error

8.2.2.3 Upgrade operation example

1. Download the new version of DM-Ansible, and confirm that there is `dm_version = v1.0.2` in the `inventory.ini` file.
2. Run `ansible-playbook local_prepare.yml` to download the new DM binary file to the local disk.
3. Run `ansible-playbook rolling_update.yml` to perform a rolling update for the DM cluster components.
4. Run `ansible-playbook rolling_update_monitor.yml` to perform a rolling update for the DM monitoring components.

> **Note:**
>
> When you upgrade DM to the 1.0.2 version, you must make sure that all DM cluster components (dmctl, DM-master, and DM-worker) are upgraded. Do not upgrade only a part of the components. Otherwise, an error might occur.

8.2.3 Upgrade to v1.0.1

8.2.3.1 Version information

```
Release Version: v1.0.1
Git Commit Hash: e63c6cdebea0edcf2ef8c91d84cff4aaa5fc2df7
Git Branch: release-1.0
UTC Build Time: 2019-09-10 06:15:05
Go Version: go version go1.12 linux/amd64
```

8.2.3.2 Main changes

- Fix the issue that DM frequently re-establishes the database connection in some situations
- Fix the panic that might occur when using the `query-status` command

8.2.3.3 Upgrade operation example

1. Download the new version of DM-Ansible, and confirm that there is `dm_version = v1.0.1` in the `inventory.ini` file.
2. Run `ansible-playbook local_prepare.yml` to download the new DM binary file to the local disk.
3. Run `ansible-playbook rolling_update.yml` to perform a rolling update for the DM cluster components.
4. Run `ansible-playbook rolling_update_monitor.yml` to perform a rolling update for the DM monitoring components.

> **Note:**
>
> When you upgrade DM to the 1.0.1 version, you must make sure that all DM cluster components (dmctl, DM-master, and DM-worker) are upgraded. Do not upgrade only a part of the components. Otherwise, an error might occur.

8.2.4 Upgrade to v1.0.0-10-geb2889c9 (1.0 GA)

8.2.4.1 Version information

```
Release Version: v1.0.0-10-geb2889c9
Git Commit Hash: eb2889c9dcfbff6653be9c8720a32998b4627db9
Git Branch: release-1.0
UTC Build Time: 2019-09-06 03:18:48
Go Version: go version go1.12 linux/amd64
```

8.2.4.2 Main changes

- Support automatically recovering migration tasks for some abnormal situations
- Improve compatibility with DDL syntaxes
- Fix the bug that the abnormal connection to the upstream database might cause data loss

8.2.4.3 Upgrade operation example

1. Download the new version of DM-Ansible, and confirm that there is `dm_version = v1.0.0` in the `inventory.ini` file.
2. Run `ansible-playbook local_prepare.yml` to download the new DM binary file to the local disk.
3. Run `ansible-playbook rolling_update.yml` to perform a rolling update for the DM cluster components.
4. Run `ansible-playbook rolling_update_monitor.yml` to perform a rolling update for the DM monitoring components.

> **Note:**
>
> When you upgrade DM to the 1.0 GA version, you must make sure that all DM cluster components (dmctl, DM-master, and DM-worker) are upgraded. Do not upgrade only a part of the components. Otherwise, an error might occur.

8.2.5 Upgrade to v1.0.0-rc.1-12-gaa39ff9

8.2.5.1 Version information

```
Release Version: v1.0.0-rc.1-12-gaa39ff9
Git Commit Hash: aa39ff981dfb3e8c0fa4180127246b253604cc34
Git Branch: dm-master
UTC Build Time: 2019-07-24 02:26:08
Go Version: go version go1.11.2 linux/amd64
```

8.2.5.2 Main changes

Starting from this release, DM checks all configurations strictly. Unrecognized configuration triggers an error. This is to ensure that users always know exactly what the configuration is.

8.2.5.3 Upgrade notes

Before starting the DM-master or DM-worker, ensure that the obsolete configuration information has been deleted and there are no redundant configuration items.

Otherwise, the starting might fail. In this situation, you can delete the deprecated configuration based on the failure information. These are two possible deprecated configurations:

- `meta-file` in `dm-worker.toml`
- `server-id` in `mysql-instances` in `task.yaml`

9 Manage Migration Tasks

9.1 Manage the Data Migration Task

This document describes how to manage and maintain the data migration task using the dmctl component. For the Data Migration cluster deployed using DM-Ansible, the dmctl binary file is in `dm-ansible/dmctl`.

The dmctl component supports the interactive mode for manual operations, and also supports the command mode for the script.

9.1.1 dmctl interactive mode

This section describes the basic use of dmctl commands in the interactive mode.

> **Note:**
>
> The interactive mode does not support Bash features. For example, you need to directly pass string flags instead of passing them in quotes.

9.1.1.1 dmctl help

```
$ ./dmctl --help
Usage of dmctl:
 # Prints the version information.
 -V prints version and exit
 -config string
      path to config file
 # Encrypts the database password according to the encryption method
    ↪ provided by DM; used in DM configuration files.
 -encrypt string
```

```
      encrypt plaintext to ciphertext
 # The DM-master access address. dmctl interacts with the DM-master to
    ↪ complete task management operations.
 -master-addr string
      master API server addr
 -rpc-timeout string
      rpc timeout ("10m" by default)
```

9.1.1.2 Database password encryption

In DM configuration files, you need to use the password encrypted using dmctl, otherwise an error occurs. For a same original password, the password is different after each encryption.

```
$ ./dmctl -encrypt 123456
VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=
```

9.1.1.3 Task management overview

Enter the interactive mode to interact with DM-master.

```
./dmctl -master-addr 172.16.30.14:8261
```

```
Welcome to dmctl
Release Version: v1.0.1
Git Commit Hash: e63c6cdebea0edcf2ef8c91d84cff4aaa5fc2df7
Git Branch: release-1.0
UTC Build Time: 2019-09-10 06:15:05
Go Version: go version go1.12 linux/amd64

» help
DM control

Usage:
  dmctl [command]

Available Commands:
  break-ddl-lock     forcefully break DM-worker's DDL lock
  check-task         check the config file of the task
  help               help about any command
  migrate-relay      migrate DM-worker's relay unit
  pause-relay        pause DM-worker's relay unit
  pause-task         pause a specified running task
  purge-relay        purge relay log files of the DM-worker according to the
     ↪  specified filename
  query-error        query task error
```

```
  query-status       query task status
  refresh-worker-tasks refresh worker -> tasks mapper
  resume-relay       resume DM-worker's relay unit
  resume-task        resume a specified paused task
  show-ddl-locks     show un-resolved DDL locks
  sql-inject         inject (limited) SQLs into binlog replication unit as
     ↪ binlog events
  sql-replace        replace SQLs matched by a specific binlog position (
     ↪ binlog-pos) or a SQL pattern (sql-pattern); each SQL must end with a
     ↪  semicolon
  sql-skip           skip the binlog event matched by a specific binlog
     ↪ position (binlog-pos) or a SQL pattern (sql-pattern)
  start-task         start a task as defined in the config file
  stop-task          stop a specified task
  switch-relay-master switch the master server of the DM-worker's relay unit
  unlock-ddl-lock    forcefully unlock DDL lock
  update-master-config update the config of the DM-master
  update-relay       update the relay unit config of the DM-worker
  update-task        update a task's config for routes, filters, or block-
     ↪ allow-list

Flags:
  -h, --help           help for dmctl
  -w, --worker strings DM-worker ID

## Use "dmctl [command] --help" for more information about a command.
```

9.1.2 Manage the data migration task

This section describes how to use the task management commands to execute corresponding operations.

9.1.2.1 Create the data migration task

You can use the `start-task` command to create the data migration task. Data Migration prechecks the corresponding privileges and configuration automatically while starting the data migration.

```
help start-task
```

```
start a task as defined in the config file

Usage:
 dmctl start-task [-w worker ...] <config-file> [flags]
```

```
Flags:
 -h, --help  help for start-task

Global Flags:
 -w, --worker strings DM-worker ID
```

9.1.2.1.1 Command usage example

```
start-task [ -w "172.16.30.15:8262"] ./task.yaml
```

9.1.2.1.2 Flags description

- `-w`: (Optional) Specifies the group of DM-workers to execute `task.yaml`. If it is set, only subtasks of the specified task on these DM-workers are started.
- `config-file`: (Required) Specifies the file path of `task.yaml`.

9.1.2.1.3 Returned results

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "172.16.30.15:8262",
            "msg": ""
        },
        {
            "result": true,
            "worker": "172.16.30.16:8262",
            "msg": ""
        }
    ]
}
```

9.1.2.2 Check the data migration task status

You can use the `query-status` task management command to check the status of the data migration task. For details about the query result and subtask status, see Query Status.

```
help query-status
```

```
query task status

Usage:
 dmctl query-status [-w worker ...] [task-name] [flags]

Flags:
 -h, --help help for query-status

Global Flags:
 -w, --worker strings DM-worker ID
```

9.1.2.2.1 Command usage example

```
query-status
```

9.1.2.2.2 Flags description

- `-w`: (Optional) Specifies the group of DM-workers where the subtasks of the migration task (that you want to query) run.
- `task-name`: (Optional) Specifies the task name. If it is not set, the results of all data migration tasks are returned.

9.1.2.2.3 Returned results

For detailed description of query parameters and a complete list of returned result, refer to Query status.

9.1.2.3 Check query errors

You can use `query-error` to check error information on migration tasks or relay units. Compared to `query-status`, `query-error` only retrieves information related to the error itself.

`query-error` is often used to obtain the binlog position information required by `sql-skip/sql-replace`. For details on the flags and results of `query-error`, refer to `query-error` in Skip or Replace Abnormal SQL Statements.

9.1.2.4 Pause the data migration task

You can use the `pause-task` command to pause a data migration task.

```
help pause-task
```

```
pause a specified running task

Usage:
 dmctl pause-task [-w worker ...] <task-name | task-file> [flags]

Flags:
 -h, --help help for pause-task

Global Flags:
 -w, --worker strings DM-worker ID
```

Note:

The differences between `pause-task` and `stop-task` are:

- `pause-task` only pauses a migration task, and the task information is retained in the memory, so that you can query using `query-status`. `stop-task` terminates a migration task and removes all task related information from the memory. This means you cannot use `query-status` to query. Data and the corresponding `dm_meta` like "checkpoint" that have been migrated to the downstream are not affected.
- `pause-task` is generally used to pause the task for troubleshooting, while `stop-task` is used to permanently end a migration task, or co-work with `start-task` to update the configuration information.

9.1.2.4.1 Command usage example

```
pause-task [-w "127.0.0.1:8262"] task-name
```

9.1.2.4.2 Flags description

- `-w`: (Optional) Specifies the group of DM-workers where the subtasks of the migration task (that you want to pause) run. If it is set, only subtasks on the specified DM-workers are paused.
- `task-name | task-file`: (Required) Specifies the task name or task file path.

9.1.2.4.3 Returned results

```
pause-task test
```

```
{
    "op": "Pause",
    "result": true,
    "msg": "",
    "workers": [
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.15:8262",
                "msg": ""
            },
            "op": "Pause",
            "logID": "2"
        },
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.16:8262",
                "msg": ""
            },
            "op": "Pause",
            "logID": "2"
        }
    ]
}
```


9.1.2.5 Resume the data migration task

You can use the `resume-task` command to resume the data migration task in the `Paused` state. This is generally used in scenarios where you want to manually resume a data migration task after you handle the errors that cause the task to pause.

```
help resume-task
```

```
resume a specified paused task

Usage:
 dmctl resume-task [-w worker ...] <task-name | task-file> [flags]

Flags:
 -h, --help help for resume-task

Global Flags:
 -w, --worker strings DM-worker ID
```

9.1.2.5.1 Command usage example

```
resume-task [-w "127.0.0.1:8262"] task-name
```

9.1.2.5.2 Flags description

- `-w`: (Optional) Specifies the group of DM-workers where the subtasks of the migration task (that you want to restart) run. If it is set, only subtasks on the specified DM-workers are restarted.
- `task-name | task-file`: (Required) Specifies the task name or task file path.

9.1.2.5.3 Returned results

```
resume-task test
```


```
{
    "op": "Resume",
    "result": true,
    "msg": "",
    "workers": [
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.15:8262",
                "msg": ""
            },
            "op": "Resume",
            "logID": "3"
        },
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.16:8262",
                "msg": ""
            },
            "op": "Resume",
            "logID": "3"
        }
    ]
}
```


9.1.2.6 Stop the data migration task

You can use the `stop-task` command to stop a data migration task. For differences between `stop-task` and `pause-task`, refer to Pause the data migration task.

```
help stop-task
```

```
stop a specified task

Usage:
 dmctl stop-task [-w worker ...] <task-name | task-file> [flags]

Flags:
 -h, --help help for stop-task

Global Flags:
 -w, --worker strings DM-worker ID
```

9.1.2.6.1 Command usage example

```
stop-task [-w "127.0.0.1:8262"] task-name
```

9.1.2.6.2 Flags description

- `-w`: (Optional) Specifies the group of DM-workers where the subtasks of the migration task (that you want to stop) run. If it is set, only subtasks on the specified DM-workers are stopped.
- `task-name | task-file`: (Required) Specifies the task name or task file path.

9.1.2.6.3 Returned results

```
stop-task test
```

```
{
    "op": "Stop",
    "result": true,
    "msg": "",
    "workers": [
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.15:8262",
                "msg": ""
            },
            "op": "Stop",
            "logID": "4"
        },
```

```
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.16:8262",
                "msg": ""
            },
            "op": "Stop",
            "logID": "4"
        }
    ]
}
```

9.1.2.7 Update the data migration task

You can use the `update-task` command to update the data migration task. The following items support online update, while all other items do not support online update.

- table route rules
- block allow list
- binlog filter rules

> **Note:**
>
> If you can make sure that the relay log required by the migration task will not be removed when the task is stopped, it is recommended that you use Update items that do not support online update to update task configurations.

9.1.2.7.1 Update items that support online update

1. Check the status of the corresponding data migration task using `query-status <task-name>`.

   If `stage` is not `Paused`, use `pause-task <task-name | task-file>` to pause the task.

2. Edit the `task.yaml` file to update the custom configuration that you need to modify and the incorrect configuration.

3. Update the task configuration using `update-task task.yaml`.

4. Resume the task using `<task-name | task-file>`.

9.1.2.7.2 Update items that do not support online update

1. Check the status of the corresponding data migration task using `query-status <task-name>`.

   If the task exists, use `stop-task <task-name | task-file>` to stop the task.

2. Edit the `task.yaml` file to update the custom configuration that you need to modify and the incorrect configuration.

3. Restart the task using `start-task <config-file>`.

9.1.2.7.3 Command usage help

```
help update-task
```

```
update a task's config for routes, filters, block-allow-list

Usage:
  dmctl update-task [-w worker ...] <config-file> [flags]

Flags:
  -h, --help help for update-task

Global Flags:
  -w, --worker strings DM-worker ID
```

9.1.2.7.4 Command usage example

```
update-task [-w "127.0.0.1:8262"] ./task.yaml
```

9.1.2.7.5 Flags description

- `-w`: (Optional) Specifies the group of DM-workers where the subtasks of the migration task (that you want to update) run. If it is set, only subtasks on the specified DM-workers are updated.
- `config-file`: (Required) Specifies the file path of `task.yaml`.

9.1.2.7.6 Returned results

```
update-task task.yaml
```

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "172.16.30.15:8262",
            "msg": ""
        },
        {
            "result": true,
            "worker": "172.16.30.16:8262",
            "msg": ""
        }
    ]
}
```


9.1.3 Manage DDL locks

Currently, DDL lock related commands mainly include `show-ddl-locks`, `unlock-ddl-lock`, `break-ddl-lock`, etc. For more information on their functions, usages, and applicable scenarios, refer to Handle Sharding DDL Locks Manually.

9.1.4 Other task and cluster management commands

In addition to the common task management commands above, DM also provides some other commands to manage data migration tasks and DM clusters.

9.1.4.1 Check the task configuration file

You can use the `check-task` command to check whether a specified configuration file (`task.yaml`) of the migration task is valid, or whether the configuration of upstream/downstream database, permission setting, and schema meet the migration requirements. For more details, refer to Precheck the upstream MySQL instance configuration.

When you use `start-task` to start a migration task, DM also executes all checks done by `check-task`.

```
help check-task
```

```
check the config file of the task

Usage:
 dmctl check-task <config-file> [flags]
```

```
Flags:
 -h, --help help for check-task

Global Flags:
 -w, --worker strings DM-worker ID
```

9.1.4.1.1 Command usage example

```
check-task task.yaml
```

9.1.4.1.2 Flags description

- `config-file`: (Required) Specifies the path of the `task.yaml` file

9.1.4.1.3 Returned results

```
check-task task-test.yaml
```

```
{
    "result": true,
    "msg": "check pass!!!"
}
```

9.1.4.2 Pause a relay unit

Relay units automatically run after the DM-worker thread starts. You can use the `pause-relay` command to pause the running relay units.

When you want to switch the DM-worker to connect to an upstream MySQL via a virtual IP, use `pause-relay` to make corresponding changes on DM.

```
help pause-relay
```

```
pause DM-worker's relay unit

Usage:
  dmctl pause-relay <-w worker ...> [flags]

Flags:
  -h, --help help for pause-relay

Global Flags:
  -w, --worker strings DM-worker ID
```

9.1.4.2.1 Command usage example

```
pause-relay -w "127.0.0.1:8262"
```

9.1.4.2.2 Flags description

- `-w`: (Required) Specifies the DM-worker for which to pause the relay unit

9.1.4.2.3 Returned results

```
pause-relay -w "172.16.30.15:8262"
```

```
{
    "op": "InvalidRelayOp",
    "result": true,
    "msg": "",
    "workers": [
        {
            "op": "PauseRelay",
            "result": true,
            "worker": "172.16.30.15:8262",
            "msg": ""
        }
    ]
}
```

9.1.4.3 Resume a relay unit

You can use the `resume-relay` command to resume a relay unit in `Paused` state.

When you want to switch the DM-worker to connect to an upstream MySQL via a virtual IP, use `resume-relay` to make corresponding changes on DM.

```
help resume-relay
```

```
resume DM-worker's relay unit

Usage:
  dmctl resume-relay <-w worker ...> [flags]

Flags:
  -h, --help help for resume-relay

Global Flags:
  -w, --worker strings DM-worker ID
```

9.1.4.3.1 Command usage example

```
resume-relay -w "127.0.0.1:8262"
```

9.1.4.3.2 Flags description

- `-w`: (Required) Specifies the DM-worker for which to resume the relay unit

9.1.4.3.3 Returned results

```
resume-relay -w "172.16.30.15:8262"
```

```
{
    "op": "InvalidRelayOp",
    "result": true,
    "msg": "",
    "workers": [
        {
            "op": "ResumeRelay",
            "result": true,
            "worker": "172.16.30.15:8262",
            "msg": ""
        }
    ]
}
```

9.1.4.4 Switch the sub-directory for relay logs

Relay units store the binlog data from upstream MySQL instances in sub-directories. You can use the `switch-relay-master` command to swith the relay unit to use a new sub-directory.

When you want to switch the DM-worker to connect to an upstream MySQL via a virtual IP, use `switch-relay-master` to make corresponding changes on DM.

```
help switch-relay-master
```

```
switch the master server of the DM-worker's relay unit

Usage:
  dmctl switch-relay-master <-w worker ...> [flags]

Flags:
  -h, --help help for switch-relay-master
```

```
Global Flags:
  -w, --worker strings DM-worker ID
```

9.1.4.4.1 Command usage example

```
switch-relay-master -w "127.0.0.1:8262"
```

9.1.4.4.2 Flags description

- `-w`: (Required) Specifies the DM-worker for which to switch the relay unit

9.1.4.4.3 Returned results

```
switch-relay-master -w "172.16.30.15:8262"
```

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "172.16.30.15:8262",
            "msg": ""
        }
    ]
}
```

9.1.4.5 Manually purge relay log

DM supports Automatic data purge. You can also use `purge-relay` to manually purge data.

```
help purge-relay
```

```
purge relay log files of the DM-worker according to the specified filename

Usage:
  dmctl purge-relay <-w worker> [--filename] [--sub-dir] [flags]

Flags:
  -f, --filename string name of the terminal file before which to purge
      ↪ relay log files. Sample format: "mysql-bin.000006"
  -h, --help           help for purge-relay
```

```
  -s, --sub-dir string specify relay sub directory for --filename. If not
      ↪ specified, the latest one will be used. Sample format: "2ae76434-
      ↪ f79f-11e8-bde2-0242ac130008.000001"

Global Flags:
  -w, --worker strings DM-worker ID
```

9.1.4.5.1 Command usage example

```
purge-relay -w "127.0.0.1:8262" --filename "mysql-bin.000003"
```

9.1.4.5.2 Flags description

- `-w`: (Required) Specifies the DM-worker for which to perform a clean operation
- `--filename`: (Required) Specifies the name of the terminal file before which to purge relay log files. For example, if the value is `mysql-bin.000100`, the clean operation stops at `mysql-bin.000099`.
- `--sub-dir`: (Optional) Specifies the relay log sub-directory corresponding to `--filename`. If not specified, the latest one is used.

9.1.4.5.3 Returned results

```
purge-relay -w "127.0.0.1:8262" --filename "mysql-bin.000003"
```

```
[warn] no --sub-dir specified for --filename; the latest one will be used
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "127.0.0.1:8262",
            "msg": ""
        }
    ]
}
```

9.1.4.6 Preset skip operation

You can use `sql-skip` to preset a skip operation to be executed when the position or the SQL statement of the binlog event matches with the specified `binlog-pos` or `sql-pattern`. For descriptions of related parameters and results, refer to sql-skip.

9.1.4.7 Preset replace operation

You can use `sql-replace` to preset a replace operation to be executed when the position or the SQL statement of the binlog event matches with the specified `binlog-pos` or `sql-pattern`. For descriptions of related parameters and results, refer to `sql-replace`.

9.1.4.8 Forcefully refresh the `task => DM-workers` mapping

You can use the `refresh-worker-tasks` command to forcefully refresh the `task => DM-workers` mapping cached in the memory of the DM-master.

> **Note:**
>
> Normally it is not necessary to use this command. Use it only when the `task => DM-workers` already exists and you are prompted to refresh it when executing other commands.

9.1.5 Refresh worker tasks

You can use the `refresh-worker-tasks` command to forcefully refresh the `task => DM-workers` mapping maintained in the DM-master memory.

> **Note:**
>
> Normally, you do not need to use this command. Use it only when you are sure that the `task => DM-workers` mapping exists, but you are still prompted to refresh while you are executing other commands.

9.1.6 dmctl command mode

The command mode differs from the interactive mode in that you need to append the task operation right after the dmctl command. The parameters of the task operation in the command mode are the same as those in the interactive mode.

> **Note:**
>
> - A dmctl command must be followed by only one task operation.
> - The task operation can be placed only at the end of the dmctl command.

```
./dmctl -master-addr 172.16.30.14:8261 start-task task.yaml
./dmctl -master-addr 172.16.30.14:8261 stop-task task
./dmctl -master-addr 172.16.30.14:8261 query-status
```

```
Available Commands:
  break-ddl-lock     break-ddl-lock <-w worker ...> <task-name> [--remove-id
     ↪ ] [--exec] [--skip]
  check-task         check-task <config-file>
  migrate-relay      migrate-relay <worker> <binlogName> <binlogPos>
  pause-relay        pause-relay <-w worker ...>
  pause-task         pause-task [-w worker ...] <task-name>
  purge-relay        purge-relay <-w worker> [--filename] [--sub-dir]
  query-error        query-error [-w worker ...] [task-name]
  query-status       query-status [-w worker ...] [task-name]
  refresh-worker-tasks refresh-worker-tasks
  resume-relay       resume-relay <-w worker ...>
  resume-task        resume-task [-w worker ...] <task-name>
  show-ddl-locks     show-ddl-locks [-w worker ...] [task-name]
  sql-inject         sql-inject <-w worker> <task-name> <sql1;sql2;>
  sql-replace        sql-replace <-w worker> [-b binlog-pos] [-s sql-pattern
     ↪ ] [--sharding] <task-name> <sql1;sql2;>
  sql-skip           sql-skip <-w worker> [-b binlog-pos] [-s sql-pattern]
     ↪ [--sharding] <task-name>
  start-task         start-task [-w worker ...] <config-file>
  stop-task          stop-task [-w worker ...] <task-name>
  switch-relay-master switch-relay-master <-w worker ...>
  unlock-ddl-lock    unlock-ddl-lock [-w worker ...] <lock-ID>
  update-master-config update-master-config <config-file>
  update-relay       update-relay [-w worker ...] <config-file>
  update-task        update-task [-w worker ...] <config-file>
```

9.1.7 Deprecated or unrecommended commands

The following commands are either deprecated or only used for debugging purposes. They might be completely removed or their semantics might be changed in future versions. **Strongly Not Recommended**.

- `migrate-relay`
- `sql-inject`
- `update-master-config`
- `update-relay`

9.2 Precheck the upstream MySQL instance configuration

This document introduces the precheck feature provided by DM. This feature is used to detect possible errors in the upstream MySQL instance configuration when the data migration task is started.

9.2.1 Command

`check-task` allows you to precheck whether the upstream MySQL instance configuration satisfies the DM requirements.

9.2.2 Checking items

Upstream and downstream database users must have the corresponding read and write privileges. DM checks the following privileges and configuration automatically while the data migration task is started:

- Database version
  - 5.5 < MySQL version < 8.0
  - MariaDB version >= 10.1.2
- MySQL binlog configuration
  - Whether the binlog is enabled (DM requires that the binlog must be enabled)
  - Whether `binlog_format=ROW` (DM only supports migration of the binlog in the ROW format)
  - Whether `binlog_row_image=FULL` (DM only supports `binlog_row_image=FULL`)
- The privileges of the upstream MySQL instance users

  MySQL users in DM configuration need to have the following privileges at least:
  - REPLICATION SLAVE
  - REPLICATION CLIENT
  - RELOAD
  - SELECT
- The compatibility of the upstream MySQL table schema

  TiDB differs from MySQL in compatibility in the following aspects:
  - TiDB does not support the foreign key.
  - Character set compatibility differs.

  DM will also check whether the primary key or unique key restriction exists in all upstream tables. This check is introduced in v1.0.7.

- The consistency of the sharded tables in the multiple upstream MySQL instances
  - The schema consistency of all sharded tables
    * Column size
    * Column name
    * Column position
    * Column type
    * Primary key
    * Unique index
  - The conflict of the auto increment primary keys in the sharded tables
    * The check fails in the following two conditions:
      · The auto increment primary key exists in the sharded tables and its column type *is not* bigint.
      · The auto increment primary key exists in the sharded tables and its column type *is* bigint, but column mapping *is not* configured.
    * The check succeeds in other conditions except the two above.

9.2.2.1 Disable checking items

DM checks items according to the task type, and you can use `ignore-checking-items` ↪ in the task configuration file to disable checking items. The list of element options for `ignore-checking-items` is as follows:

Element	Description
all	Disables all checks
dump_privilege	Disables checking dump-related privileges of the upstream MySQL instance user

Element	Description
replication_privilege	Disables checking replication-related privileges of the upstream MySQL instance user
version	Disables checking the upstream database version
binlog_enable	Disables checking whether the upstream database has binlog enabled

Element	Description
binlog_format	Disables checking whether the binlog format of the upstream database is ROW
binlog_row_image	Disables checking whether the binlog_row_image of the upstream database is FULL
table_schema	Disables checking the compatibility of the upstream MySQL table schema

Element	Description
schema_of_sharded_tables	Disables checking whether the schemas of upstream MySQL sharded tables are consistent in the multi-instance sharding scenario
auto_increment_ID	Disables checking the conflicts of auto-increment primary keys of the upstream MySQL shared tables in the multi-instance sharding scenario

9.3 Data Migration Query Status

This document introduces the query result, task status, and subtask status of Data Migration (DM).

9.3.1 Query result

```
» query-status
```

```
{
    "result": true, # Whether the query is successful.
    "msg": "",    # Describes the cause of the unsuccessful query.
    "tasks": [    # Migration task list.
        {
            "taskName": "test-1",       # The task name
            "taskStatus": "Running",    # The status of the task, including "
                ↪ New", "Running", "Paused", "Stopped", "Finished", and "
                ↪ Error".
            "workers": [                # The list of DM-workers that are
                ↪ used by the task.
                "127.0.0.1:8262"
            ]
        },
        {
            "taskName": "test-2",
            "taskStatus": "Error - Some error occurred in subtask", # A
                ↪ subtask encounters an error and is paused.
            "workers": [
                "127.0.0.1:8262",
                "127.0.0.1:8263"
            ]
        },
        {
            "taskName": "test-3",
            "taskStatus": "Error - Relay status is Error", # An error occurs
                ↪ in the Relay processing unit corresponding to a subtask
                ↪ that is in the Sync phase.
            "workers": [
                "127.0.0.1:8263",
                "127.0.0.1:8264"
            ]
        }
    ]
}
```

For detailed descriptions of `taskStatus` under the `tasks` section, refer to Task status.

It is recommended that you use `query-status` by the following steps:

1. Use `query-status` to check whether each on-going task is in the normal state.
2. If any error occurs in a task, use the `query-status <taskName>` command to see detailed error information. `<taskName>` in this command indicates the name of the task that encounters the error.

9.3.2 Task status

The status of a DM migration task depends on the status of each subtask assigned to DM-worker. For detailed descriptions of subtask status, see Subtask status. The table below shows how the subtask status is related to task status.

Subtask status in a task	Task status
One subtask is in the `paused` state and error information is returned.	`Error` - `Some paused error occurred in subtask`

Subtask status in a task	Task status
One subtask in the Sync phase is in the Running state but its Relay processing unit is not running (in the Error state).	`Error` `-` `Relay status` `is` `Error` `/` `Paused` `/` `Stopped`

Subtask status in a task	Task status
One subtask is in the `Paused` state and no error information is returned.	`Paused`
All subtasks are in the `New` state.	`New`
All subtasks are in the `Finished` state.	`Finished`

Subtask status in a task	Task status
All subtasks are in the Stopped state.	Stopped
Other situations	Running

9.3.3 Detailed query result

```
» query-status test
```

```
» query-status
{
    "result": true,   # Whether the query is successful.
    "msg": "",        # Describes the cause for the unsuccessful query.
    "workers": [                        # DM-worker list.
        {
            "result": true,
            "worker": "172.17.0.2:8262", # The `host:port` information of the
                ↪ DM-worker.
            "msg": "",
            "subTaskStatus": [          # The information of all the subtasks
                ↪ of the DM-worker.
                {
                    "name": "test",       # The name of the subtask.
                    "stage": "Running",  # The running status of the subtask,
                        ↪ including "New", "Running", "Paused", "Stopped", and
                        ↪  "Finished".
                    "unit": "Sync",       # The processing unit of DM,
                        ↪ including "Check", "Dump", "Load", and "Sync".
                    "result": null,       # Displays the error information if a
                        ↪  subtask fails.
```

```
                "unresolvedDDLLockID": "test-`test`.`t_target`", # The
                    ↪ sharding DDL lock ID, used for manually handling the
                    ↪  sharding DDL
                                                                 # lock in the
                                                                     ↪
                                                                     ↪ abnormal
                                                                     ↪
                                                                     ↪ condition
                                                                     ↪ .
                "sync": {                # The replication information of
                    ↪ the `Sync` processing unit. This information is
                    ↪ about the
                                         # same component with the current
                                             ↪ processing unit.
                    "totalEvents": "12", # The total number of binlog
                        ↪ events that are replicated in this subtask.
                    "totalTps": "1",     # The number of binlog events that
                        ↪  are replicated in this subtask per second.
                    "recentTps": "1",    # The number of binlog events that
                        ↪  are replicated in this subtask in the last one
                        ↪ second.
                    "masterBinlog": "(bin.000001, 3234)",
                        ↪                          # The binlog position in
                        ↪ the upstream database.
                    "masterBinlogGtid": "c0149e17-dff1-11e8-b6a8-0242
                        ↪ ac110004:1-14", # The GTID information in the
                        ↪ upstream database.
                    "syncerBinlog": "(bin.000001, 2525)",
                        ↪                          # The position of the
                        ↪ binlog that has been replicated
                                                                                #
                                                                                    ↪
                                                                                    ↪ i
                                                                                    ↪
                                                                                    ↪ t
                                                                                    ↪
                                                                                    ↪ `
                                                                                    ↪ S
                                                                                    ↪ `
                                                                                    ↪
                                                                                    ↪ p
                                                                                    ↪
                                                                                    ↪ u
                                                                                    ↪ .
                                                                                    ↪
```

```
"syncerBinlogGtid": "",                                    # It is
    ↪ always empty because `Sync` does not use GTID to
                                                                    #
                                                                        ↪
                                                                        ↪ m
                                                                        ↪
                                                                        ↪ d
                                                                        ↪ .
                                                                        ↪
"blockingDDLs": [    # The DDL list that is blocked
    ↪ currently. It is not empty only when all the
    ↪ upstream tables of this
                     # DM-worker are in the "synced"
                         ↪ status. In this case, it
                         ↪ indicates the sharding DDL
                         ↪ statements to be executed
                         ↪ or that are skipped.
    "USE `test`; ALTER TABLE `test`.`t_target` DROP
        ↪ COLUMN `age`;"
],
"unresolvedGroups": [ # The sharding group that is not
    ↪ resolved.
    {
        "target": "`test`.`t_target`",             # The
            ↪ downstream database table to be
            ↪ replicated.
        "DDLs": [
            "USE `test`; ALTER TABLE `test`.`t_target`
                ↪ DROP COLUMN `age`;"
        ],
        "firstPos": "(bin|000001.000001, 3130)",  # The
            ↪ starting position of the sharding DDL
            ↪ statement.
        "synced": [                                # The
            ↪ upstream sharded table whose executed
            ↪ sharding DDL statement has been read by
            ↪ the `Sync` unit.
            "`test`.`t2`"
            "`test`.`t3`"
            "`test`.`t1`"
        ],
        "unsynced": [                              # The
            ↪ upstream table that has not executed this
            ↪  sharding DDL
```

```
                                                                #
                                                                    ↪ statement
                                                                    ↪ .
                                                                    ↪
                                                                    ↪ If
                                                                    ↪
                                                                    ↪ any
                                                                    ↪
                                                                    ↪ upstream
                                                                    ↪
                                                                    ↪ tables
                                                                    ↪
                                                                    ↪ have
                                                                    ↪
                                                                    ↪ not
                                                                    ↪
                                                                    ↪ finished
                                                                    ↪
                                                                    ↪ replication
                                                                    ↪ ,
                                                                    ↪
                                                                # `
                                                                    ↪ blockingDDLs
                                                                    ↪ `
                                                                    ↪
                                                                    ↪ is
                                                                    ↪
                                                                    ↪ empty
                                                                    ↪ .
                                                                    ↪
                        ]
                    }
                ],
                "synced": false      # Whether the incremental
                    ↪ replication catches up with the upstream and has
                    ↪  the same binlog position as that in the
                                     # upstream. The save point is not
                                         ↪ refreshed in real time in
                                         ↪ the `Sync` background, so "
                                         ↪ false" of "synced"
                                     # does not always mean a
                                         ↪ replication delay exits.
            }
        }
    ],
```

```
        "relayStatus": {  # The replication status of the relay log.
            "masterBinlog": "(bin.000001, 3234)",                       #
                ↪ The binlog position of the upstream database.
            "masterBinlogGtid": "c0149e17-dff1-11e8-b6a8-0242ac110004
                ↪ :1-14", # The binlog GTID information of the upstream
                ↪ database.
            "relaySubDir": "c0149e17-dff1-11e8-b6a8-0242ac110004.000001",
                ↪ # The currently used subdirectory of the relay log.
            "relayBinlog": "(bin.000001, 3234)",                        #
                ↪ The position of the binlog that has been pulled to the
                ↪ local storage.
            "relayBinlogGtid": "c0149e17-dff1-11e8-b6a8-0242ac110004
                ↪ :1-14", # The GTID information of the binlog that has
                ↪ been pulled to the local
                                                                        #
                                                                      ↪ storage
                                                                      ↪ .
                                                                      ↪
            "relayCatchUpMaster": true, # Whether the progress of
                ↪ migrating the relay log in the local storage has been
                ↪ the same as that in
                                        # the upstream.
            "stage": "Running",         # The status of the `Sync`
                ↪ processing unit of the relay log.
            "result": null
        },
        "sourceID": "172.17.0.2:3306"    # ID of the upstream instance or
            ↪ replication group
    },
    {
        "result": true,
        "worker": "172.17.0.3:8262",
        "msg": "",
        "subTaskStatus": [
            {
                "name": "test",
                "stage": "Running",
                "unit": "Load",
                "result": null,
                "unresolvedDDLLockID": "",
                "load": {               # The replication information of
                    ↪ the `Load` processing unit.
                    "finishedBytes": "115", # The number of bytes that have
                        ↪  been loaded.
```

```
                "totalBytes": "452", # The total number of bytes that
                    ↪ need to be loaded.
                "progress": "25.44 %" # The progress of the loading
                    ↪ process.
            }
        }
    ],
    "relayStatus": {
        "masterBinlog": "(bin.000001, 28507)",
        "masterBinlogGtid": "c0149e17-dff1-11e8-b6a8-0242ac110004
            ↪ :1-96",
        "relaySubDir": "c0149e17-dff1-11e8-b6a8-0242ac110004.000001",
        "relayBinlog": "(bin.000001, 28507)",
        "relayBinlogGtid": "c0149e17-dff1-11e8-b6a8-0242ac110004
            ↪ :1-96",
        "relayCatchUpMaster": true,
        "stage": "Running",
        "result": null
    },
    "sourceID": "172.17.0.3:3306"
},
{
    "result": true,
    "worker": "172.17.0.6:8262",
    "msg": "",
    "subTaskStatus": [
        {
            "name": "test",
            "stage": "Paused",
            "unit": "Load",
            "result": {             # The error example.
                "isCanceled": false,
                "errors": [
                    {
                        "Type": "ExecSQL",
                        "msg": "Error 1062: Duplicate entry
                            ↪ '1155173304420532225' for key 'PRIMARY'\n
                            ↪ /home/jenkins/workspace/build_dm/go/src/
                            ↪ github.com/pingcap/tidb-enterprise-tools/
                            ↪ loader/db.go:160: \n/home/jenkins/
                            ↪ workspace/build_dm/go/src/github.com/
                            ↪ pingcap/tidb-enterprise-tools/loader/db.
                            ↪ go:105: \n/home/jenkins/workspace/
                            ↪ build_dm/go/src/github.com/pingcap/tidb-
                            ↪ enterprise-tools/loader/loader.go:138:
```

```
                                ↪ file test.t1.sql"
                        }
                    ],
                    "detail": null
                },
                "unresolvedDDLLockID": "",
                "load": {
                    "finishedBytes": "0",
                    "totalBytes": "156",
                    "progress": "0.00 %"
                }
            }
        ],
        "relayStatus": {
            "masterBinlog": "(bin.000001, 1691)",
            "masterBinlogGtid": "97b5142f-e19c-11e8-808c-0242ac110005
                ↪ :1-9",
            "relaySubDir": "97b5142f-e19c-11e8-808c-0242ac110005.000001",
            "relayBinlog": "(bin.000001, 1691)",
            "relayBinlogGtid": "97b5142f-e19c-11e8-808c-0242ac110005:1-9",
            "relayCatchUpMaster": true,
            "stage": "Running",
            "result": null,
            "sourceID": "172.17.0.6:3306"
        }
    }
]
}
```

For the status description and status switch relationship of "stage" of "subTaskStatus" of "workers", see Subtask status.

For operation details of "unresolvedDDLLockID" of "subTaskStatus" of "workers", see Handle Sharding DDL Locks Manually.

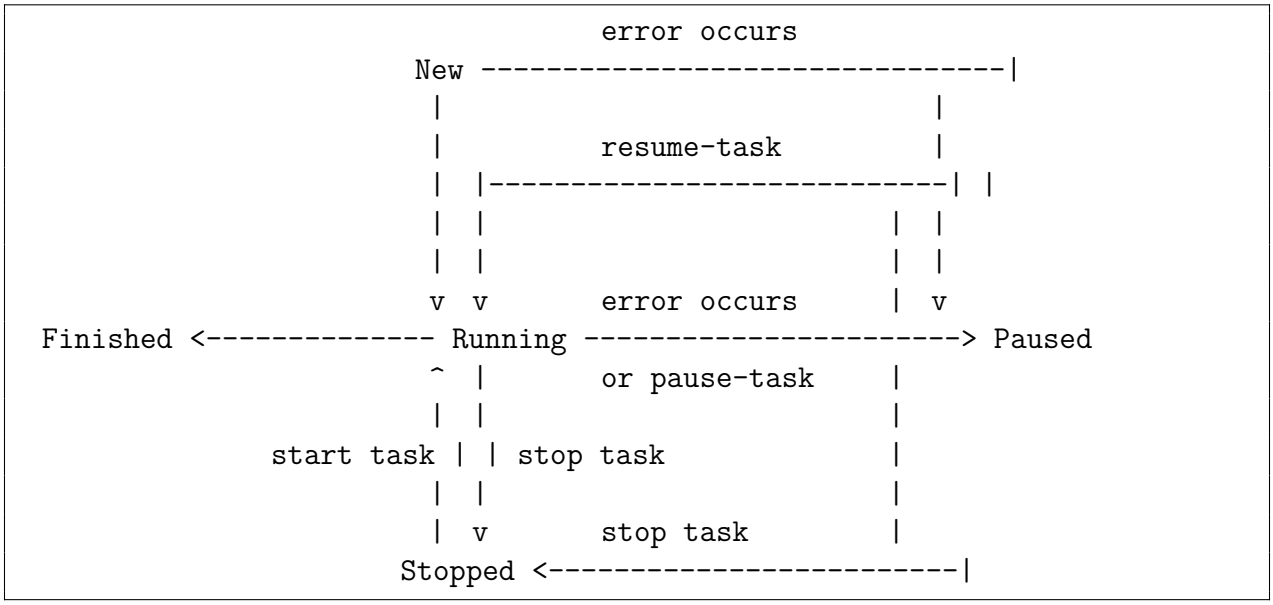
9.3.4 Subtask status

9.3.4.1 Status description

- `New`:
  - The initial status.
  - If the subtask does not encounter an error, it is switched to `Running`; otherwise it is switched to `Paused`.

- `Running`: The normal running status.
- `Paused`:
  - The paused status.
  - If the subtask encounters an error, it is switched to `Paused`.
  - If you run `pause-task` when the subtask is in the `Running` status, the task is switched to `Paused`.
  - When the subtask is in this status, you can run the `resume-task` command to resume the task.
- `Stopped`:
  - The stopped status.
  - If you run `stop-task` when the subtask is in the `Running` or `Paused` status, the task is switched to `Stopped`.
  - When the subtask is in this status, you cannot use `resume-task` to resume the task.
- `Finished`:
  - The finished subtask status.
  - Only when the full replication subtask is finished normally, the task is switched to this status.

9.3.4.2 Status switch diagram



9.4 Skip or Replace Abnormal SQL Statements

This document introduces how to handle abnormal SQL statements using Data Migration (DM).

Currently, TiDB is not completely compatible with all MySQL syntax (see the DDL statements supported by TiDB). Therefore, when DM is migrating data from MySQL to TiDB and TiDB does not support the corresponding SQL statement, an error might occur and break the migration process. In this case, there are two ways to resume the migration:

- Use dmctl to manually skip the binlog event to which this SQL statement corresponds
- Use dmctl to manually replace the corresponding binlog event with other specified SQL statements that should be executed to the downstream later

If you know in advance that an unsupported SQL statement is going to be migrated, you can also use dmctl to manually preset the skip or replace operation, which is automatically executed when DM migrates the corresponding binlog event into the downstream and thus avoid breaking the migration.

9.4.1 Restrictions

- The skip or replace operation is a one-time operation that is only used to skip or replace the SQL statement unsupported by the downstream TiDB. Do not handle other migration errors with this approach.
  - For other migration errors, try to handle them using Block and allow table lists or Binlog event filtering.
- If it is unacceptable in the actual production environment that the abnormal DDL statement is skipped in the downstream TiDB and it cannot be replaced with other DDL statements, then do not use this approach.
  - For example: `DROP PRIMARY KEY`
  - In this scenario, you can only create a new table in the downstream with the new table schema (after executing the DDL statement), and re-import all the data into this new table.
- A single skip or replace operation targets at a single binlog event.
- `--sharding` is only used to preset the operation to the sharding group. You must preset it before executing the DDL statement and presetting it after executing the DDL is not allowed.
  - `--sharding` only supports presetting operations, and in this mode, you can only use `--sql-pattern` to match the binlog event.
  - For the principles of migrating sharding DDL statements using DM, see Merge and migrate data from sharded tables

9.4.2 Match the binlog event

When the migration task gets interrupted because of the SQL execution error, you can obtain the position of the corresponding binlog event by using `query-error`. When you execute `sql-skip` or `sql-replace`, you can specify the position to match the binlog event.

However, when you try to avoid breaking the migration by actively handling unsupported SQL statements, you cannot know in advance the position of the binlog event, so you need another approach to match the subsequent binlog events.

In DM, two modes of matching the binlog event are supported (you can only choose one mode from below):

1. binlog position: the position information of the binlog event

    - The binlog position is given by `--binlog-pos` in the command, and the format is `binlog-filename:binlog-pos`, for example, `mysql-bin|000001.000003:3270`.
    - The format of the binlog filename in DM is not completely consistent with that in the upstream MySQL.
    - When the migration error occurs, the position can be directly obtained from `failedBinlogPosition` returned by `query-error`.

2. DDL pattern: the regular expression (only for the DDL statement) matching mode

    - The DDL pattern is given by `--sql-pattern` in the command, for example, to match `ALTER TABLE \`db2\`.\`tbl2\` DROP COLUMN \`c2\``, the corresponding regular expression should be `~(?i)ALTER\s+TABLE\s+\`db2\`.\`tbl2\`\s+DROP\s+COLUMN\s+\`c2\``.
    - The regular expression must be prefixed with `~` and cannot contain any common space (you can replace the space with `\s` or `\s+` in the string).

In the scenario of merging and migrating data from sharded tables, if you need DM to automatically select a DDL lock owner to execute the skip or replace operation, then you must use the DDL pattern matching mode because the binlog positions corresponding to the DDL statements on different DM-workers have no logical connection and are hard to confirm.

Note:

- You can only register one operator (specified by `--binlog-pos`) for one binlog event. The previous one can be overwritten by the newly registered operator.
- Do not specify an operator for one binlog event by using `--binlog-pos` and `--sql-pattern` at the same time.

- The operator is deleted once it successfully matches the binlog event (not after the execution succeeds). If you need to match again (using `--sql-pattern`) later, you have to register a new operator.

9.4.3 Supported scenarios

- Scenario 1: during the migration, the DDL statement unsupported by TiDB is executed in the upstream and migrated to the downstream, and as a result, the migration task gets interrupted.
  - If it is acceptable that this DDL statement is skipped in the downstream TiDB, then you can use `sql-skip` to resume the migration.
  - If it is acceptable that this DDL statement is replaced with other DDL statements, then you can use `sql-replace` to resume the migration.
- Scenario 2: during the migration, you know in advance that an unsupported SQL statement is going to be migrated, so you can handle it beforehand to avoid breaking the migration.
  - If it is acceptable that this DDL statement is skipped in the downstream TiDB, then you can use `sql-skip` to preset an operation to automatically skip this DDL statement when it needs to be executed.
  - If it is acceptable that this DDL statement is replaced with other DDL statements, then you can use `sql-replace` to preset an operation to automatically replace this DDL statement when it needs to be executed.

9.4.4 Implementation principles

In DM, simplified procedures of incremental data replication can be described as follows:

1. The relay unit is used as a secondary database of the upstream MySQL to fetch the binlog that is persisted in the local storage as the relay log.
2. The binlog replication unit (sync) reads the local relay log to obtain the binlog event.
3. The binlog replication unit parses the binlog event and builds the DDL/DML statements, and then replicates these statements to the downstream TiDB.

When the binlog replication unit is parsing the binlog event and replicating data to the downstream, the replication process might get interrupted because the corresponding SQL statement is not supported by TiDB.

In DM, you can register some skip or replace operators for the binlog event. Before migrating the SQL statements to the downstream, DM compares the current binlog event information(position, DDL statement) with registered operators. If the position or the DDL matches with a registered operator, it executes the operation corresponding to the operator and then remove this operator.

Use `sql-skip` / `sql-replace` to resume the migration

1. Use `sql-skip` or `sql-replace` to register an operator for the specified binlog position or DDL pattern.
2. Use `resume-task` to resume the migration task.
3. Regain and re-parse the binlog event that causes the migration error.
4. The binlog event successfully matches with the registered operator in step 1.
5. Execute the skip or replace operation corresponding to the operator and then the migration task continues.

Use `sql-skip` / `sql-replace` to preset operations to avoid breaking the migration

1. Use `sql-skip` or `sql-replace` to register an operator for the specified DDL pattern.
2. Parse the relay log to obtain the binlog event.
3. The binlog event (including the SQL statements unsupported by TiDB) successfully matches with the registered operator in step 1.
4. Execute the skip or replace operation corresponding to the operator and then the migration task continues and does not get interrupted.

Use `sql-skip` / `sql-replace` to preset operations to avoid breaking the migration in the scenario of merging and migrating data from sharded tables

1. Use `sql-skip` or `sql-replace` to register an operator (on DM-master) for the specified DDL pattern.
2. Each DM-worker parses the relay log to obtain the binlog event.
3. DM-master coordinates the DDL lock migration among DM-workers.
4. DM-master checks if the DDL lock migration succeeds, and sends the registered operator in step 1 to the DDL lock owner.
5. DM-master requests the DDL lock owner to execute the DDL statement.
6. The DDL statement that is to be executed by the DDL lock owner successfully matches with the received operator in step 4.
7. Execute the skip or replace operation corresponding to the operator and then the migration task continues.

9.4.5 Command

When you use dmctl to manually handle the SQL statements unsupported by TiDB, the commonly used commands include `query-status`, `query-error`, `sql-skip` and `sql-replace`.

9.4.5.1 query-status

`query-status` allows you to query the current status of items such as the subtask and the relay unit in each DM-worker. For details, see query status.

9.4.5.2 query-error

`query-error` allows you to query the existing errors of the running subtask and relay unit in DM-workers.

9.4.5.2.1 Command usage

```
query-error [--worker=127.0.0.1:8262] [task-name]
```

9.4.5.2.2 Arguments description

- `worker`:
  - Flag parameter, string, `--worker`, optional
  - If it is not specified, this command queries the errors in all DM-workers; if it is specified, this command queries the error of the specified DM-worker.
- `task-name`:
  - Non-flag parameter, string, optional
  - If it is not specified, this command queries the errors of all tasks; if it is specified, this command queries the error of the specified task.

9.4.5.2.3 Example of results

```
» query-error test
{
    "result": true,                             # The result of the error query.
    "msg": "",                                  # The additional message for the
        ↪  failure to the error query.
    "workers": [                                # The information list of DM-
        ↪ workers.
        {
            "result": true,                     # The result of the error query
                ↪ in this DM-worker.
```

```
            "worker": "127.0.0.1:8262",      # The IP:port (worker-id) of
               ↪ this DM-worker.
            "msg": "",                       # The additional message for the
               ↪  failure to the error query in this DM-worker.
            "subTaskError": [                # The error information of the
               ↪ running subtask in this DM-worker.
               {
                  "name": "test",            # The task name.
                  "stage": "Paused",         # The status of the current task
                     ↪ .
                  "unit": "Sync",            # The current processing unit of
                     ↪  the running task.
                  "sync": {                  # The error information of the
                     ↪ binlog replication unit (sync).
                     "errors": [             # The error information list of
                        ↪ the current processing unit.
                        {
                           // The error information description.
                           "msg": "exec sqls[[USE `db1`; ALTER TABLE `db1
                              ↪ `.`tbl1` CHANGE COLUMN `c2` `c2` decimal
                              ↪ (10,3);]] failed, err:Error 1105:
                              ↪ unsupported modify column length 10 is
                              ↪ less than origin 11",
                           // The position of the failed binlog event.
                           "failedBinlogPosition": "mysql-bin
                              ↪ |000001.000003:34642",
                           // The SQL statement that raises an error.
                           "errorSQL": "[USE `db1`; ALTER TABLE `db1`.`tbl1
                              ↪ ` CHANGE COLUMN `c2` `c2` decimal(10,3);]
                              ↪ "
                        }
                     ]
                  }
               }
            ],
            "RelayError": {                  # The error information of the
               ↪ relay processing unit in this DM-worker.
               "msg": ""                     # The error information
                  ↪ description.
            }
         }
      ]
}
```

9.4.5.3 sql-skip

`sql-skip` allows you to preset a skip operation that is to be executed when the position or the SQL statement of the binlog event matches with the specified `binlog-pos` or `sql-pattern`.

9.4.5.3.1 Command usage

```
sql-skip <--worker=127.0.0.1:8262> [--binlog-pos=mysql-bin
    ↪ |000001.000003:3270] [--sql-pattern=~(?i)ALTER\s+TABLE\s+`db1`.`tbl1
    ↪ `\s+ADD\s+COLUMN\s+col1\s+INT] [--sharding] <task-name>
```

9.4.5.3.2 Arguments description

- `worker`:
  - Flag parameter, string, `--worker`
  - If `--sharding` is not specified, `worker` is required; if `--sharding` is specified, `worker` is forbidden to use.
  - `worker` specifies the DM-worker in which the presetted operation is going to be executed.
- `binlog-pos`:
  - Flag parameter, string, `--binlog-pos`
  - You must specify `binlog-pos` or `--sql-pattern`, and you must not specify both.
  - If it is specified, the skip operation is executed when `binlog-pos` matches with the position of the binlog event. The format is `binlog-filename:binlog-pos`, for example, `mysql-bin|000001.000003:3270`.
  - When the migration error occurs, the position can be obtained from `failedBinlogPosition` ↪ returned by `query-error`.
- `sql-pattern`:
  - Flag parameter, string, `--sql-pattern`
  - You must specify `--sql-pattern` or `binlog-pos`, and you must not specify both.
  - If it is specified, the skip operation is executed when `sql-pattern` matches with the DDL statement (converted by the optional router-rule) of the binlog event. The format is a regular expression prefixed with `~`, for example, `~(?i)ALTER\s+` ↪ `TABLE\s+`db1`.`tbl1`\s+ADD\s+COLUMN\s+col1\s+INT`.
    * Common spaces are not supported in the regular expression temporarily. You can replace the space with `\s` or `\s+` if it is needed.
    * The regular expression must be prefixed with `~`. For details, see regular expression syntax.

    * The schema/table name in the regular expression must be converted by the optional router-rule, so the converted name is consistent with the target schema/table name in the downstream. For example, if there are `` `shard_db_1`.`shard_tbl_1` `` in the upstream and `` `shard_db`.`shard_tbl` `` in the downstream, then you should match `` `shard_db`.`shard_tbl` ``.
    * The schema/table/column name in the regular expression should be marked by `, for example, `` `db1`.`tbl1` ``.

- `sharding`:
  - Flag parameter, boolean, `--sharding`
  - If `--worker` is not specified, `sharding` is required; if `--worker` is specified, `sharding` is forbidden to use.
  - If `sharding` is specified, it indicates that the presetted operation is going to be executed in the DDL lock owner during the sharding DDL migration.
- `task-name`:
  - Non-flag parameter, string, required
  - `task-name` specifies the name of the task in which the presetted operation is going to be executed.

9.4.5.4 sql-replace

`sql-replace` allows you to preset a replace operation that is to be executed when the position or the SQL statement of the binlog event matches with the specified `binlog-pos` or `sql-pattern`.

9.4.5.4.1 Command usage

```
sql-replace <--worker=127.0.0.1:8262> [--binlog-pos=mysql-bin
    ↪ |000001.000003:3270] [--sql-pattern=~(?i)ALTER\s+TABLE\s+`db1`.`tbl1
    ↪ `\s+ADD\s+COLUMN\s+col1\s+INT] [--sharding] <task-name> <SQL-1;SQL-2>
```

9.4.5.4.2 Arguments description

- `worker`:
  - same with `--worker` of `sql-skip`
- `binlog-pos`:
  - same with `--binlog-pos` of `sql-skip`
- `sql-pattern`:
  - same with `--sql-pattern` of `sql-skip`

- `sharding`:
  - same with `--sharding` of `sql-skip`
- `task-name`:
  - same with `task-name` of `sql-skip`
- `SQLs`:
  - Non-flag parameter, string, required
  - `SQLs` specifies the new SQL statements that are going to replace the original binlog event. You should separate multiple SQL statements with `;`, for example, `ALTER TABLE shard_db.shard_table drop index idx_c2;ALTER TABLE shard_db.shard_table DROP COLUMN c2;`.

9.4.6 Usage examples

9.4.6.1 Passively skip after the migration gets interrupted

9.4.6.1.1 Application scenario

Assume that you need to migrate the upstream table `db1.tbl1` to the downstream TiDB (not in the scenario of merging and migrating data from sharded tables). The initial table schema is:

```
mysql> SHOW CREATE TABLE db1.tbl1;
+-------+-----------------------------------------------------+
| Table | Create Table                                        |
+-------+-----------------------------------------------------+
| tbl1 | CREATE TABLE `tbl1` (
  `c1` int(11) NOT NULL,
  `c2` decimal(11,3) DEFAULT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-------+-----------------------------------------------------+
```

Now, the following DDL statement is executed in the upstream to alter the table schema (namely, alter DECIMAL(11, 3) of c2 into DECIMAL(10, 3)):

```
ALTER TABLE db1.tbl1 CHANGE c2 c2 DECIMAL (10, 3);
```

Because this DDL statement is not supported by TiDB, the migration task of DM gets interrupted and reports the following error:

```
exec sqls[[USE `db1`; ALTER TABLE `db1`.`tbl1` CHANGE COLUMN `c2` `c2`
    ↪ decimal(10,3);]] failed,
err:Error 1105: unsupported modify column length 10 is less than origin 11
```

Now, if you query the status of the task using `query-status`, you can see that `stage` has changed into `Paused` and there is some related error description information in `errors`.

To obtain the details about the error, you should use `query-error`. For example, you can execute `query-error test` to get the position of the failed binlog event (`failedBinlogPosition`), which is `mysql-bin|000001.000003:34642`.

9.4.6.1.2 Passively skip the SQL statement

Assume that it is acceptable in the actual production environment that this DDL statement is not executed in the downstream TiDB (namely, the original table schema is retained). Then you can use `sql-skip` to skip this DDL statement to resume the migration. The procedures are as follows:

1. Use `query-error` to obtain the position of the failed binlog event.

    - You can get the position from `failedBinlogPosition` returned by `query-error`.
    - In this example, the position is `mysql-bin|000001.000003:34642`.

2. Use `sql-skip` to preset a skip operation that is to be executed when DM migrates this binlog event to the downstream after using `resume-task`.

```
» sql-skip --worker=127.0.0.1:8262 --binlog-pos=mysql-bin
   ↪ |000001.000003:34642 test
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "",
            "msg": ""
        }
    ]
}
```

   You can also view the following log in the corresponding DM-worker node:

```
2018/12/28 11:17:51 operator.go:121: [info] [sql-operator] set a new
   ↪ operator
uuid: 6bfcf30f-2841-4d70-9a34-28d7082bdbd7, pos: (mysql-bin
   ↪ |000001.000003, 34642), op: SKIP, args:
on migration unit
```

3. Use `resume-task` to resume the migration task

```
» resume-task --worker=127.0.0.1:8262 test
{
    "op": "Resume",
    "result": true,
    "msg": "",
    "workers": [
        {
            "op": "Resume",
            "result": true,
            "worker": "127.0.0.1:8262",
            "msg": ""
        }
    ]
}
```

You can also view the following log in the corresponding DM-worker node:

```
2018/12/28 11:27:46 operator.go:158: [info] [sql-operator] binlog-pos (
    ↪ mysql-bin|000001.000003, 34642) matched,
applying operator uuid: 6bfcf30f-2841-4d70-9a34-28d7082bdbd7, pos: (
    ↪ mysql-bin|000001.000003, 34642), op: SKIP, args:
```

4. Use `query-status` to guarantee that the `stage` of the task has changed into `Running`.
5. Use `query-error` to guarantee that no DDL execution error exists.

9.4.6.2 Actively replace before the migration gets interrupted

9.4.6.2.1 Application scenario

Assume that you need to migrate the upstream table `db2.tbl2` to the downstream TiDB (not in the scenario of merging and migrating data from sharded tables). The initial table schema is:

```
mysql> SHOW CREATE TABLE db2.tbl2;
+-------+-------------------------------------------------+
| Table | Create Table                                    |
+-------+-------------------------------------------------+
| tbl2 | CREATE TABLE `tbl2` (
  `c1` int(11) NOT NULL,
  `c2` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`),
  KEY `idx_c2` (`c2`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-------+-------------------------------------------------+
```

Now, the following DDL statement is executed in the upstream to alter the table schema (namely, `DROP COLUMN c2`):

```
ALTER TABLE db2.tbl2 DROP COLUMN c2;
```

Because this DDL statement is not supported by TiDB, the migration task of DM gets interrupted and reports the following error:

```
exec sqls[[USE `db2`; ALTER TABLE `db2`.`tbl2` DROP COLUMN `c2`;]] failed,
err:Error 1105: can't drop column c2 with index covered now
```

Assume that you know in advance that this DDL statement is not supported by TiDB before it is executed in the upstream. Then you can use `sql-skip` or `sql-replace` to preset a skip or replace operation for this DDL statement.

For this particular DDL statement, because dropping columns with the index is not temporarily supported by TiDB, you can use two new SQL statements to replace the original DDL, namely, DROP the index first and then DROP the column c2.

9.4.6.2.2 Actively replace the SQL statement

1. Design a matchable regular expression for the DDL statement (converted by the optional router-rule) to be executed in the upstream.

    - The DDL statement to be executed in the upstream is `ALTER TABLE db2.tbl2 ↪ DROP COLUMN c2;`.
    - Because its router-rule conversion does not exist, you can design the following regular expression:

    ```
    ~(?i)ALTER\s+TABLE\s+`db2`.`tbl2`\s+DROP\s+COLUMN\s+`c2`
    ```

2. Build new DDL statements that are used to replace this original DDL statement.

    ```
    ALTER TABLE `db2`.`tbl2` DROP INDEX idx_c2;ALTER TABLE `db2`.`tbl2`
       ↪ DROP COLUMN `c2`
    ```

3. Use `sql-replace` to preset a replace operation that is to be executed when DM migrates the corresponding binlog event to the downstream.

    ```
    » sql-replace --worker=127.0.0.1:8262 --sql-pattern=~(?i)ALTER\s+TABLE\
       ↪ s+`db2`.`tbl2`\s+DROP\s+COLUMN\s+`c2` test ALTER TABLE `db2`.`
       ↪ tbl2` DROP INDEX idx_c2;ALTER TABLE `db2`.`tbl2` DROP COLUMN `c2`
    {
        "result": true,
        "msg": "",
        "workers": [
    ```

```
        {
            "result": true,
            "worker": "",
            "msg": ""
        }
    ]
}
```

You can also view the following log in the corresponding DM-worker node:

```
2018/12/28 15:33:13 operator.go:121: [info] [sql-operator] set a new
    ↪ operator
uuid: c699a18a-8e75-47eb-8e7e-0e5abde2053c, pattern: ~(?i)ALTER\s+TABLE
    ↪ \s+`db2`.`tbl2`\s+DROP\s+COLUMN\s+`c2`,
op: REPLACE, args: ALTER TABLE `db2`.`tbl2` DROP INDEX idx_c2; ALTER
    ↪ TABLE `db2`.`tbl2` DROP COLUMN `c2`
on migration unit
```

4. Execute the DDL statements in the upstream MySQL.

5. Check if the downstream table schema is altered successfully, and you can view the following log in the corresponding DM-worker node:

```
2018/12/28 15:33:45 operator.go:158: [info] [sql-operator]
sql-pattern ~(?i)ALTER\s+TABLE\s+`db2`.`tbl2`\s+DROP\s+COLUMN\s+`c2`
    ↪ matched SQL
USE `db2`; ALTER TABLE `db2`.`tbl2` DROP COLUMN `c2`;,
applying operator uuid: c699a18a-8e75-47eb-8e7e-0e5abde2053c,
pattern: ~(?i)ALTER\s+TABLE\s+`db2`.`tbl2`\s+DROP\s+COLUMN\s+`c2`,
op: REPLACE, args: ALTER TABLE `db2`.`tbl2` DROP INDEX idx_c2; ALTER
    ↪ TABLE `db2`.`tbl2` DROP COLUMN `c2`
```

6. Use `query-status` to guarantee that the `stage` of the task has been sustained as `Running`.

7. Use `query-error` to guarantee that no DDL execution error exists.

9.4.6.3 Passively skip after the migration gets interrupted in the scenario of merging and migrating data from sharded tables

9.4.6.3.1 Application scenario

Assume that you need to merge and migrate multiple tables in multiple upstream MySQL instances to one same table in the downstream TiDB through multiple DM-workers. And the DDL statement unsupported by TiDB is executed to the upstream sharded tables.

After DM-master coordinates the DDL migration through the DDL lock and requests the DDL lock owner to execute the DDL statement to the downstream, the migration gets interrupted because this DDL statement is not supported by TiDB.

9.4.6.3.2 Passively skip the SQL statement

In the scenario of merging and migrating data from sharded tables, passively skipping the unsupported DDL statement has the similar steps with Passively skip after the migration gets interrupted.

There are two major differences between the two scenarios as follows. In the scenario of merging and migrating data from sharded tables:

1. You just need the DDL lock owner to execute `sql-skip` (`--worker={DDL-lock-owner}`).
2. You just need the DDL lock owner to execute `resume-task` (`--worker={DDL-lock-owner}`).

9.4.6.4 Actively replace before the migration gets interrupted in the scenario of merging and migrating data from sharded tables

9.4.6.4.1 Application scenario

Assume that you need to merge and migrate the following four tables in the upstream to one same table `` `shard_db`.`shard_table` `` in the downstream:

- In the MySQL instance 1, there is a schema `shard_db_1`, which has two tables `shard_table_1` and `shard_table_2`.
- In the MySQL instance 2, there is a schema `shard_db_2`, which has two tables `shard_table_1` and `shard_table_2`.

The initial table schema is:

```
mysql> SHOW CREATE TABLE shard_db_1.shard_table_1;
+---------------+-----------------------------------------+
| Table         | Create Table                            |
+---------------+-----------------------------------------+
| shard_table_1 | CREATE TABLE `shard_table_1` (
  `c1` int(11) NOT NULL,
  `c2` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`),
  KEY `idx_c2` (`c2`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+---------------+-----------------------------------------+
```

Now, the following DDL statement is executed to all upstream sharded tables to alter the table schemas (namely, `DROP COLUMN c2`):

```
ALTER TABLE shard_db_*.shard_table_* DROP COLUMN c2;
```

When DM coordinates the two DM-workers to migrate this DDL statement through the sharding DDL lock and requests the DDL lock owner to execute the DDL statement to the downstream, because this DDL statement is not supported by TiDB, the migration task gets interrupted and report the following error:

```
exec sqls[[USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` DROP COLUMN
    ↪  `c2`;]] failed,
err:Error 1105: can't drop column c2 with index covered now
```

Assume that you know in advance that this DDL statement is not supported by TiDB before it is executed in the upstream. Then you can use `sql-skip` or `sql-replace` to preset a skip or replace operation for this DDL statement.

For this particular DDL statement, because dropping columns with the index is not temporarily supported by TiDB, you can use two new SQL statements to replace the original DDL, namely, DROP the index first and then DROP the column c2.

9.4.6.4.2 Actively replace the SQL statement

1. Design a matchable regular expression for the DDL statement (converted by the optional router-rule) to be executed in the upstream.

    - The DDL statement to be executed in the upstream is `ALTER TABLE shard_db_ ↪ *.shard_table_* DROP COLUMN c2`.
    - Because the table name should be converted into `` `shard_db`.`shard_table` `` by the router-rule, you can design the following regular expression:

    ```
    ~(?i)ALTER\s+TABLE\s+`shard_db`.`shard_table`\s+DROP\s+COLUMN\s+`
        ↪ c2`
    ```

2. Build new DDL statements that are used to replace this original DDL statement.

    ```
    ALTER TABLE `shard_db`.`shard_table` DROP INDEX idx_c2;ALTER TABLE `
        ↪ shard_db`.`shard_table` DROP COLUMN `c2`
    ```

3. Because this is in the scenario of merging and migrating data from sharded tables, you can use `--sharding` to automatically guarantee that the replace operation is only executed in the DDL lock owner.

4. Use `sql-replace` to preset a replace operation that is to be executed when DM migrates the corresponding binlog event to the downstream.

```
» sql-replace --sharding --sql-pattern=~(?i)ALTER\s+TABLE\s+`shard_db
   ↪ `.`shard_table`\s+DROP\s+COLUMN\s+`c2` test ALTER TABLE `
   ↪ shard_db`.`shard_table` DROP INDEX idx_c2;ALTER TABLE `shard_db
   ↪ `.`shard_table` DROP COLUMN `c2`
{
    "result": true,
    "msg": "request with --sharding saved and will be sent to DDL lock
       ↪ 's owner when resolving DDL lock",
    "workers": [
    ]
}
```

You can also view the following log in the **DM-master** node:

```
2018/12/28 16:53:33 operator.go:105: [info] [sql-operator] set a new
   ↪ operator
uuid: eba35acd-6c5e-4bc3-b0b0-ae8bd1232351, request: name:"test"
op:REPLACE args:"ALTER TABLE `shard_db`.`shard_table` DROP INDEX idx_c2
   ↪ ;"
args:"ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`"
sqlPattern:"~(?i)ALTER\\s+TABLE\\s+`shard_db`.`shard_table`\\s+DROP\\s+
   ↪ COLUMN\\s+`c2`"
sharding:true
```

5. Execute the DDL statements to the sharded tables in the upstream MySQL instances.

6. Check if the downstream table schema is altered successfully, and you can also view the following log in the DDL lock **owner** node:

```
2018/12/28 16:54:35 operator.go:121: [info] [sql-operator] set a new
   ↪ operator
uuid: c959f2fb-f1c2-40c7-a1fa-e73cd51736dd,
pattern: ~(?i)ALTER\s+TABLE\s+`shard_db`.`shard_table`\s+DROP\s+COLUMN\
   ↪ s+`c2`,
op: REPLACE, args: ALTER TABLE `shard_db`.`shard_table` DROP INDEX
   ↪ idx_c2; ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`
on migration unit
```

```
2018/12/28 16:54:35 operator.go:158: [info] [sql-operator]
sql-pattern ~(?i)ALTER\s+TABLE\s+`shard_db`.`shard_table`\s+DROP\s+
   ↪ COLUMN\s+`c2` matched SQL
USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`;,
applying operator uuid: c959f2fb-f1c2-40c7-a1fa-e73cd51736dd,
pattern: ~(?i)ALTER\s+TABLE\s+`shard_db`.`shard_table`\s+DROP\s+COLUMN\
   ↪ s+`c2`,
```

```
op: REPLACE, args: ALTER TABLE `shard_db`.`shard_table` DROP INDEX
    ↪ idx_c2; ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`
```

In addition, you can view the following log in the **DM-master** node:

```
2018/12/28 16:54:35 operator.go:122: [info] [sql-operator] get an
    ↪ operator
uuid: eba35acd-6c5e-4bc3-b0b0-ae8bd1232351, request: name:"test" op:
    ↪ REPLACE
args:"ALTER TABLE `shard_db`.`shard_table` DROP INDEX idx_c2;"
args:"ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`"
sqlPattern:"~(?i)ALTER\\s+TABLE\\s+`shard_db`.`shard_table`\\s+DROP\\s+
    ↪ COLUMN\\s+`c2`"
sharding:true
with key ~(?i)ALTER\s+TABLE\s+`shard_db`.`shard_table`\s+DROP\s+COLUMN\
    ↪ s+`c2` matched SQL
USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`;
```

```
2018/12/28 16:54:36 operator.go:145: [info] [sql-operator] remove an
    ↪ operator
uuid: eba35acd-6c5e-4bc3-b0b0-ae8bd1232351, request: name:"test" op:
    ↪ REPLACE
args:"ALTER TABLE `shard_db`.`shard_table` DROP INDEX idx_c2;"
args:"ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`"
sqlPattern:"~(?i)ALTER\\s+TABLE\\s+`shard_db`.`shard_table`\\s+DROP\\s+
    ↪ COLUMN\\s+`c2`"
sharding:true
```

7. Use `query-status` to guarantee that the `stage` of the task has been sustained as `Running`, and there is no more DDL statement that is blocking the migration (`blockingDDLs`) and no more sharding group to be resolved (`unresolvedGroups`).
8. Use `query-error` to guarantee that no DDL execution error exists.
9. Use `show-ddl-locks` to guarantee that all DDL locks have been resolved.

10 Data Migration Monitoring Metrics

If your DM cluster is deployed using DM-Ansible, the monitoring system is also deployed at the same time. This document describes the monitoring metrics provided by DM-worker.

> **Note:**
>
> Currently, DM-master does not provide monitoring metrics yet.

10.1 Task

In the Grafana dashboard, the default name of DM is `DM-task`.

10.1.1 `overview`

`overview` contains some monitoring metrics of all the DM-worker instances in the currently selected task. The current default alert rule is only for a single DM-worker instance.

Metric name	Description	Alert	Severity level
task state	The state of subtasks for migration	N/A	N/A
storage capacity	The total storage capacity of the disk occupied by relay logs	N/A	N/A
storage remain	The remaining storage capacity of the disk occupied by relay logs	N/A	N/A
binlog file gap between master and relay	The number of binlog files by which the `relay` processing unit is behind the upstream master	N/A	N/A

Metric name	Description	Alert	Severity level
load progress	The percentage of the completed loading process of the load unit. The value is between 0%~100%	N/A	N/A
binlog file gap between master and syncer	The number of binlog files by which the binlog replication unit is behind the upstream master	N/A	N/A
shard lock resolving	Whether the current subtask is waiting for sharding DDL migration. A value greater than 0 means that the current subtask is waiting for sharding DDL migration	N/A	N/A

10.1.2 Task state

Metric name	Description	Alert	Severity level
task state	The state of subtasks	An alert occurs when the sub-task has been paused for more than 20 min-utes	critical

10.1.3 Relay log

Metric name	Description	Alert	Severity level
storage capac-ity	The storage capacity of the disk occupied by the relay log	N/A	N/A
storage re-main	The remaining storage capacity of the disk occupied by the relay log	An alert is needed once the value is smaller than 10G	critical

Metric name	Description	Alert	Severity level
process exits with error	The relay log encounters an error within the DM-worker and exits	Immediate alerts	critical
relay log data corruption	The number of corrupted relay log files	Immediate alerts	emergency
fail to read binlog from master	The number of errors encountered when the relay log reads the binlog from the upstream MySQL	Immediate alerts	critical
fail to write relay log	The number of errors encountered when the relay log writes the binlog to disks	Immediate alerts	critical
binlog file index	The largest index number of relay log files. For example, “value = 1” indicates “relay-log.000001”	N/A	N/A

Metric name	Description	Alert	Severity level
binlog file gap between master and relay	The number of binlog files in the relay log that are behind the upstream master	An alert occurs when the number of binlog files by which the `relay` processing unit is behind the upstream master exceeds one (>1) and the condition lasts over 10 minutes	critical
binlog pos	The write offset of the latest relay log file	N/A	N/A

Metric name	Description	Alert	Severity level
read binlog event duration	The duration that the relay log reads binlog from the upstream MySQL (in seconds)	N/A	N/A
write relay log duration	The duration that the relay log writes binlog into the disks each time (in seconds)	N/A	N/A
binlog event size	The size of a single binlog event that the relay log writes into the disks	N/A	N/A

10.1.4 Dump/Load unit

The following metrics show only when `task-mode` is in the `full` or `all` mode.

Metric name	Description	Alert	Severity level
load progress	The percentage of the completed loading process of the load unit. The value range is 0%~100%	N/A	N/A

Metric name	Description	Alert	Severity level
data file size	The total size of the data files (includes the `INSERT INTO` statement) in the full data imported by the load unit	N/A	N/A
dump process exits with error	The dump unit encounters an error within the DM-worker and exits	Immediate alerts	critical
load process exits with error	The load unit encounters an error within the DM-worker and exits	Immediate alerts	critical
table count	The total number of tables in the full data imported by the load unit	N/A	N/A
data file count	The total number of data files (includes the `INSERT INTO` statement) in the full data imported by the load unit	N/A	N/A

Metric name	Description	Alert	Severity level
transaction execution latency	The latency of executing a transaction by the load unit (in seconds)	N/A	N/A
statement execution latency	The duration of executing a statement by the load unit (in seconds)	N/A	N/A

10.1.5 Binlog replication

The following metrics show only when `task-mode` is in the `incremental` or `all` mode.

Metric name	Description	Alert	Severity level
remaining time to sync	The predicted remaining time it takes `syncer` to be completely migrated with the master (in minutes)	N/A	N/A
replicate lag	The latency time it takes to replicate the binlog from master to `syncer` (in seconds)	N/A	N/A

Metric name	Description	Alert	Severity level
process exist with error	The binlog replication unit encounters an error within the DM-worker and exits	Immediate alerts	critical

Metric name	Description	Alert	Severity level
binlog file gap between master and syncer	The number of binlog files by which the `syncer` processing unit is behind the master	An alert occurs when the number of binlog files by which the `syncer` ↪ processing unit is behind the master exceeds one (>1) and the condition lasts over 10 minutes	critical

Metric name	Description	Alert	Severity level
binlog file gap between relay and syncer	The number of binlog files by which `syncer` is behind `relay`	An alert occurs when the number of binlog files by which the `syncer` ↪ processing unit is behind the `relay` processing unit exceeds one (>1) and the condition lasts over 10 minutes	critical

Metric name	Description	Alert	Severity level
binlog event QPS	The number of binlog events received per unit of time (this number does not include the events that need to be skipped)	N/A	N/A
skipped binlog event QPS	The number of binlog events received per unit of time that need to be skipped	N/A	N/A
read binlog event duration	The duration that the binlog replication unit reads the binlog from the relay log or the upstream MySQL (in seconds)	N/A	N/A
transform binlog event duration	The duration that the binlog replication unit parses and transforms the binlog into SQL statements (in seconds)	N/A	N/A

Metric name	Description	Alert	Severity level
dispatch binlog event duration	The duration that the binlog replication unit dispatches a binlog event (in seconds)	N/A	N/A
transaction execution latency	The duration that the binlog replication unit executes the transaction to the downstream (in seconds)	N/A	N/A
binlog event size	The size of a binlog event that the binlog replication unit reads from the relay log or the upstream MySQL	N/A	N/A
DML queue remain length	The length of the remaining DML job queue	N/A	N/A
total sqls jobs	The number of newly added jobs per unit of time	N/A	N/A
finished sqls jobs	The number of finished jobs per unit of time	N/A	N/A

Metric name	Description	Alert	Severity level
statement execution latency	The duration that the binlog replication unit executes the statement to the downstream (in seconds)	N/A	N/A
add job duration	The duration tht the binlog replication unit adds a job to the queue (in seconds)	N/A	N/A
DML conflict detect duration	The duration that the binlog replication unit detects the conflict in DML (in seconds)	N/A	N/A
skipped event duration	The duration that the binlog replication unit skips a binlog event (in seconds)	N/A	N/A
unsynced tables	The number of tables that have not received the shard DDL statement in the current subtask	N/A	N/A

Metric name	Description	Alert	Severity level
shard lock resolving	Whether the current subtask is waiting for the shard DDL lock to be resolved. A value greater than 0 indicates that it is waiting for the shard DDL lock to be resolved	N/A	N/A

10.2 Instance

In the Grafana dashboard, the default name of an instance is `DM-instance`.

10.2.1 Relay log

Metric name	Description	Alert	Severity level
storage capacity	The total storage capacity of the disk occupied by the relay log	N/A	N/A
storage remain	The remaining storage capacity within the disk occupied by the relay log	An alert occurs once the value is smaller than 10G	critical

Metric name	Description	Alert	Severity level
process exits with error	The relay log encounters an error in DM-worker and exits	Immediate alerts	critical
relay log data corruption	The number of corrupted relay logs	Immediate alerts	emergency
fail to read binlog from master	The number of errors encountered when relay log reads the binlog from the upstream MySQL	Immediate alerts	critical
fail to write relay log	The number of errors encountered when the relay log writes the binlog to disks	Immediate alerts	critical
binlog file index	The largest index number of relay log files. For example, “value = 1” indicates “relay-log.000001”	N/A	N/A

Metric name	Description	Alert	Severity level
binlog file gap between master and relay	The number of binlog files by which the `relay` processing unit is behind the upstream master	An alert occurs when the number of binlog files by which the `relay` processing unit is behind the upstream master exceeds one (>1) and the condition lasts over 10 minutes	critical
binlog pos	The write offset of the latest relay log file	N/A	N/A

Metric name	Description	Alert	Severity level
read binlog duration	The duration that the relay log reads the binlog from the upstream MySQL (in seconds)	N/A	N/A
write relay log duration	The duration that the relay log writes the binlog into the disk each time (in seconds)	N/A	N/A
binlog size	The size of a single binlog event that the relay log writes into the disks	N/A	N/A

10.2.2 Task

Metric name	Description	Alert	Severity level
task state	The state of subtasks for migration	An alert occurs when the sub-task has been paused for more than 10 min-utes	critical

Metric name	Description	Alert	Severity level
load progress	The percentage of the completed loading process of the load unit. The value range is 0%~100%	N/A	N/A
binlog file gap between master and syncer	The number of binlog files by which the binlog replication unit is behind the upstream master	N/A	N/A
shard lock resolving	Whether the current subtask is waiting for sharding DDL migration. A value greater than 0 means that the current subtask is waiting for sharding DDL migration	N/A	N/A

11 Migrate from MySQL-compatible Database

11.1 Migrate from a MySQL-compatible Database - Taking Amazon Aurora MySQL as an Example

This document describes how to migrate from Amazon Aurora MySQL to TiDB by using TiDB Data Migration (DM).

11.1.1 Step 1: Enable binlog in the Aurora cluster

Assuming that you want to migrate data from two Aurora clusters to TiDB, the information of the Aurora clusters is listed in the following table. The Aurora-1 cluster contains a separate reader endpoint.

Cluster	Endpoint	Port	Role
Aurora-1	pingcap-1.h8emfqdptyc4.us-east-2.rds.amazonaws.com	3306	Writer
Aurora-1	pingcap-1-us-east-2a.h8emfqdptyc4.us-east-2.rds.amazonaws.com	3306	Reader
Aurora-2	pingcap-2.h8emfqdptyc4.us-east-2.rds.amazonaws.com	3306	Writer

DM relies on the `ROW` format of binlog during the incremental replication process, so you need to set the binlog format as `ROW`. If binlog is not enabled or is incorrectly configured, DM cannot migrate data normally. For more details, see Checking items.

> **Note:**
>
> Because binlog cannot be enabled in the Aurora reader, it cannot be taken as the upstream master server when you use DM to migrate data.

If you need to migrate data based on GTID (Global Transaction Identifier), enable GTID for the Aurora cluster.

> **Note:**
>
> GTID-based data migration requires MySQL 5.7 (Aurora 2.04.1) version or later.

11.1.1.1 Modify binlog related parameters in the Aurora cluster

In the Aurora cluster, binlog related parameters are cluster level parameters among cluster parameter groups. For more information about binlog in the Aurora cluster, see Enable Binary Logging on the Replication Master. You need to set the `binlog_format` to `ROW` when using DM for data migration.

To migrate data based on GTID, set both `gtid-mode` and `enforce_gtid_consistency` to `ON`. See Configuring GTID-Based Replication for an Aurora MySQL Cluster for more information about enabling GTID-based migration for Aurora cluster.

> **Note:**
>
> In the AWS Management Console, the `gtid_mode` parameter appears as `gtid -mode`.

11.1.2 Step 2: Deploy the DM cluster

It is recommended to use DM-Ansible to deploy a DM cluster. See Deploy Data Migration Using DM-Ansible.

> **Note:**
>
> - Use password encrypted with dmctl in all the DM configuration files. If the database password is empty, it is unnecessary to encrypt it. For how to use dmctl to encrypt a cleartext password, see Encrypt the upstream MySQL user password using dmctl.
> - Both the upstream and downstream users must have the corresponding read and write privileges.

11.1.3 Step 3: Check the cluster informtaion

After a DM cluster is deployed using DM-Ansible, the configuration information is as follows:

- DM cluster components

Component	Host	Port
dm_worker1	172.16.10.72	8262
dm_worker2	172.16.10.73	8262
dm_master	172.16.10.71	8261

- Upstream and downstream database instances

Database instance	Host	Port	Username	Encrypted password
Upstream Aurora-1	pingcap-1.h8emfqdptyc4.us-east-2.rds.amazonaws.com	3306	root	VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=
Upstream Aurora-2	pingcap-2.h8emfqdptyc4.us-east-2.rds.amazonaws.com	3306	root	VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=
Downstream TiDB	172.16.4.108	4000	root	

- Configuration in the `{ansible deploy}/conf/dm-master.toml` DM-master process configuration file

```
# DM-Master Configuration

[[deploy]]
source-id = "mysql-replica-01"
dm-worker = "172.16.10.72:8262"

[[deploy]]
source-id = "mysql-replica-02"
dm-worker = "172.16.10.73:8262"
```

11.1.4 Step 4: Configure the task

This section assumes that you need to migrate data of the `test_table` table in the `test_db` schema of Aurora-1 and Aurora-2 instances, in both full data migration and incremental replication modes, to the `test_table` table of the `test_db` schema in one downstream TiDB instance.

Copy and edit `{ansible deploy}/conf/task.yaml.example` to generate the following `task.yaml` configuration file:

```
## The task name. You need to use a different name for each of the multiple
   ↪ tasks that run simultaneously.
name: "test"
## The full data migration plus incremental replication task mode.
task-mode: "all"
## The downstream TiDB configuration information.
```

```
target-database:
  host: "172.16.10.83"
  port: 4000
  user: "root"
  password: ""

## Configuration of all the upstream MySQL instances required by the current
   ↪  data migration task.
mysql-instances:
-
  # ID of the upstream instance or the migration group. Refer to the
     ↪ configuration of `source_id` in the `inventory.ini` file or
     ↪ configuration of `source-id` in the `dm-master.toml` file.
  source-id: "mysql-replica-01"
  # The configuration item name of the block and allow lists of the schema
     ↪ or table to be migrated, used to quote the global block and allow
     ↪ lists configuration. For global configuration, see the `block-allow-
     ↪ list` below.
  block-allow-list: "global" # Use black-white-list if the DM's version <=
     ↪ v1.0.6.
  # The configuration item name of the dump unit, used to quote the global
     ↪ dump unit configuration.
  mydumper-config-name: "global"

-
  source-id: "mysql-replica-02"
  block-allow-list: "global" # Use black-white-list if the DM's version <=
     ↪ v1.0.6.
  mydumper-config-name: "global"

## The global configuration of block and allow lists. Each instance can
   ↪ quote it by the configuration item name.
block-allow-list:                 # Use black-white-list if the DM's version
   ↪  <= v1.0.6.
  global:
    do-tables:                    # The allow list of the upstream table to
       ↪ be migrated
    - db-name: "test_db"          # The database name of the table to be
       ↪ migrated
      tbl-name: "test_table"      # The name of the table to be migrated

## The global configuration of dump unit. Each instance can quote it by the
   ↪ configuration item name.
mydumpers:
  global:
```

```
  extra-args: "-B test_db -T test_table" # Extra arguments of the dump
     ↪ unit. Since DM 1.0.2, DM automatically generates the "--tables-
     ↪ list" configuration. For versions earlier than 1.0.2, you need to
     ↪ configure this option manually.
```

11.1.5 Step 5: Start the task

1. Go to the dmctl directory: `/home/tidb/dm-ansible/resources/bin/`.
2. Start dmctl using the following command:

```
./dmctl --master-addr 172.16.10.71:8261
```

3. Start data migration task using the following command:

```
# `task.yaml` is the previously edited configuration file.
start-task ./task.yaml
```

- If the returned results do not contain any error, it indicates the task is successfully started.
- If the returned results contain the following error information, it indicates the upstream Aurora user might have privileges unsupported by TiDB:

```
{
    "id": 4,
    "name": "source db dump privilege chcker",
    "desc": "check dump privileges of source DB",
    "state": "fail",
    "errorMsg": "line 1 column 285 near \"LOAD FROM S3, SELECT INTO
       ↪  S3 ON *.* TO 'root'@'%' WITH GRANT OPTION\" ...",
    "instruction": "",
    "extra": "address of db instance - pingcap-1.h8emfqdptyc4.us-
       ↪ east-2.rds.amazonaws.com"
},
{
    "id": 5,
    "name": "source db replication privilege chcker",
    "desc": "check replication privileges of source DB",
    "state": "fail",
    "errorMsg": "line 1 column 285 near \"LOAD FROM S3, SELECT INTO
       ↪  S3 ON *.* TO 'root'@'%' WITH GRANT OPTION\" ...",
    "instruction": "",
    "extra": "address of db instance - pingcap-1.h8emfqdptyc4.us-
       ↪ east-2.rds.amazonaws.com"
}
```

To resolve this issue, use either of the following two solutions to handle it and then use the `start-task` command to restart the task:

1. Remove the unnecessary privileges unsupported by TiDB for the Aurora user that is used to migrate data.
2. If you can make sure that the Aurora user has the privileges required by DM, add the following configuration item to the `task.yaml` configuration file to skip the privileges precheck when starting the task.

```
ignore-checking-items: ["dump_privilege", "
    ↪ replication_privilege"]
```

11.1.6 Step 6: Query the task

To view the on-going data migration task(s) in the DM cluster or the task status, run the following command in dmctl to query:

```
query-status
```

Note:

If the following error message is in the returned results of the above query command, it indicates the corresponding lock cannot be obtained during the phase of the full data migration.

```
bash  Couldn't acquire global lock, snapshots will not be
↪  consistent: Access denied for user 'root'@'%' (using
↪ password: YES)
```

If it is acceptable to not use FTWL to guarantee that the dump file is consistent with metadata or the upstream can pause writing data, you can skip the above error by adding the `--no-locks` argument for `extra-args` under `mydumpers`. The steps are as follows:

1. Use the `stop-task` command to stop the paused task caused by the failure of nomarl dumping.
2. In the `task.yaml` file, modify `extra-args: "-B test_db -T
   ↪ test_table"` to `extra-args: "-B test_db -T test_table --no
   ↪ -locks"`.
3. Use the `start-task` command to restart the task.

12 DM Portal Overview

Data Migration (DM) provides a variety of features, including table routing, block & allow table lists, and binlog event filter. However, these features also increase the complexity of using DM, especially when users are modifying DM task configurations.

To address this problem, DM provides a simple web application, DM Portal. DM Portal enables users to visually configure the required migration tasks, and generates a `task.yaml` file that can be directly executed by DM.

12.1 Features

This sections describes the features of DM Portal.

12.1.1 Configure the migration mode

DM Portal supports three migration modes:

- Full migration
- Incremental replication
- All (full + incremental)

12.1.2 Configure the instance information

DM Portal supports configuring table routing, and merging sharded schemas and tables in DM.

12.1.3 Configure the binlog event filter

DM Portal supports configuring the binlog event filter in schemas and tables.

12.1.4 Generate the configuration file

DM Portal supports generating configuration files and downloading these files to your local computer. Meanwhile, it automatically creates a file in the `/tmp/` directory on the dm-portal server.

12.2 Restrictions

Currently, DM Portal's visualized pages cover most DM configuration scenarios, but with the following restrictions:

- The SQL pattern of binlog event filter is not supported.

- The editing feature does not support parsing the `task.yaml` file created by the user. The user can only edit the `task.yaml` file generated by the page.
- The editing feature does not support modifying the instance configuration. If the user need to adjust the instance configuration, the `task.yaml` file has to be regenerated.
- The upstream instance configuration on the page can only be used to obtain the upstream table schema. The related upstream instance information still needs to be configured in DM-worker.
- In the generated `task.yaml` file, mydumper-path is `./bin/mydumper` by default. If you use another path, modify the generated `task.yaml` file manually.

12.3 Deploy

This section describes how to deploy DM Portal in two ways: using binary or DM Ansible.

12.3.1 Deploy using binary

Download DM Portal at dm-portal-latest-linux-amd64.tar.gz. To start DM Portal, run the `./dm-portal` command.

- If you run DM Portal locally, visit `127.0.0.1:8280` in your browser.
- If you run DM Portal on a server, configure a proxy on the server.

12.3.2 Deploy using DM Ansible

To deploy DM Portal using DM Ansible, refer to Deploy Data Migration Using DM-Ansible for details.

12.4 Usage

This section describes how to use DM Portal.

12.4.1 Create rules

This feature is used to create a `task.yaml` file.

12.4.1.1 Operation steps

Access the dm-portal page, and click **Create New Rule**.

12.4.2 Configure the basic information

This feature is used to fill in the task name and select a task type.

12.4.2.1 Prerequisites

Create New Sync Rule is already selected.

12.4.2.2 Operation steps

1. Fill in the task name.
2. Choose a task type.

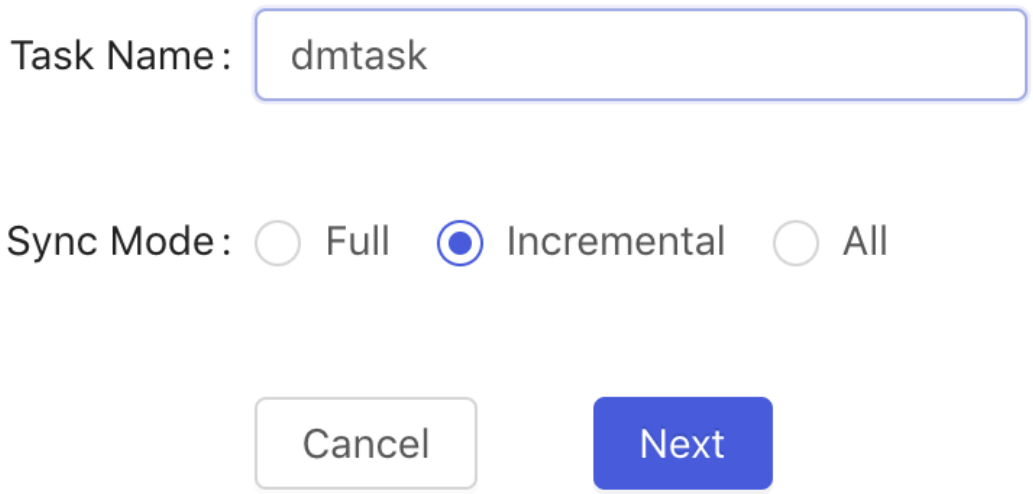


Figure 7: DM Portal BasicConfig

12.4.3 Configure the instance information

This feature is used to configure the upstream and downstream instance information, including Host, Port, Username, and Password.

12.4.3.1 Prerequisites

Task Name and **Task Type** are already filled in.

Note:

If you choose **Incremental** or **All** in Task Type, you need to configure binlog-file and binlog-pos when configuring the upstream instance information.

12.4.3.2 Operation steps

1. Fill in the upstream instance information.
2. Fill in the downstream instance information.
3. Click **Next**.

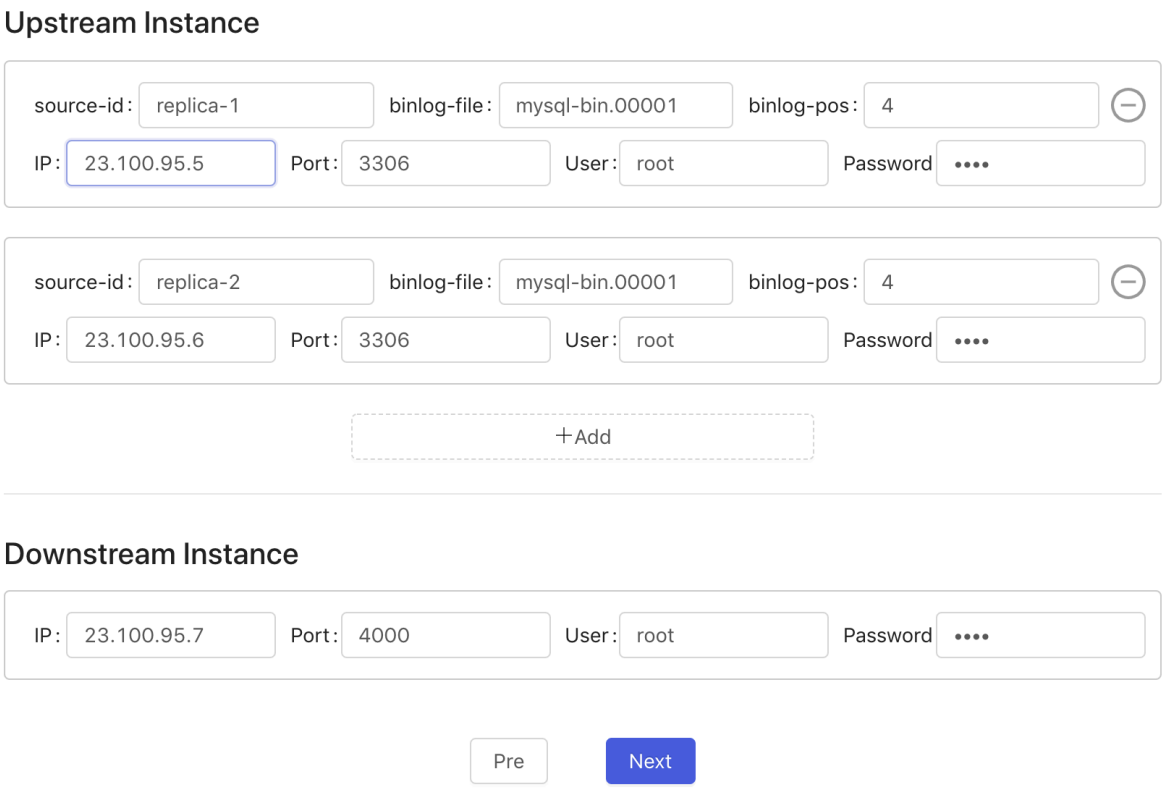


Figure 8: DM Portal InstanceConfig

12.4.4 Configure the binlog filter

This feature is used to filter the upstream binlog. You can choose the DDL or DML that needs to be filtered. The filter configured on the database is automatically inherited by tables in that database.

12.4.4.1 Prerequisites

- The upstream and downstream instance information is configured.
- The connection is verified.

> **Note:**
>
> - The binlog filter configuration can only be modified in the upstream instance. Once the database or table is moved to the downstream instance, the configuration cannot be modified.
> - The binlog filter configured on the database is automatically inherited by tables in that database.

12.4.4.2 Operation steps

1. Select the databases or tables that need to be configured.
2. Click the **Edit** button, and select the binlog types to be filtered.

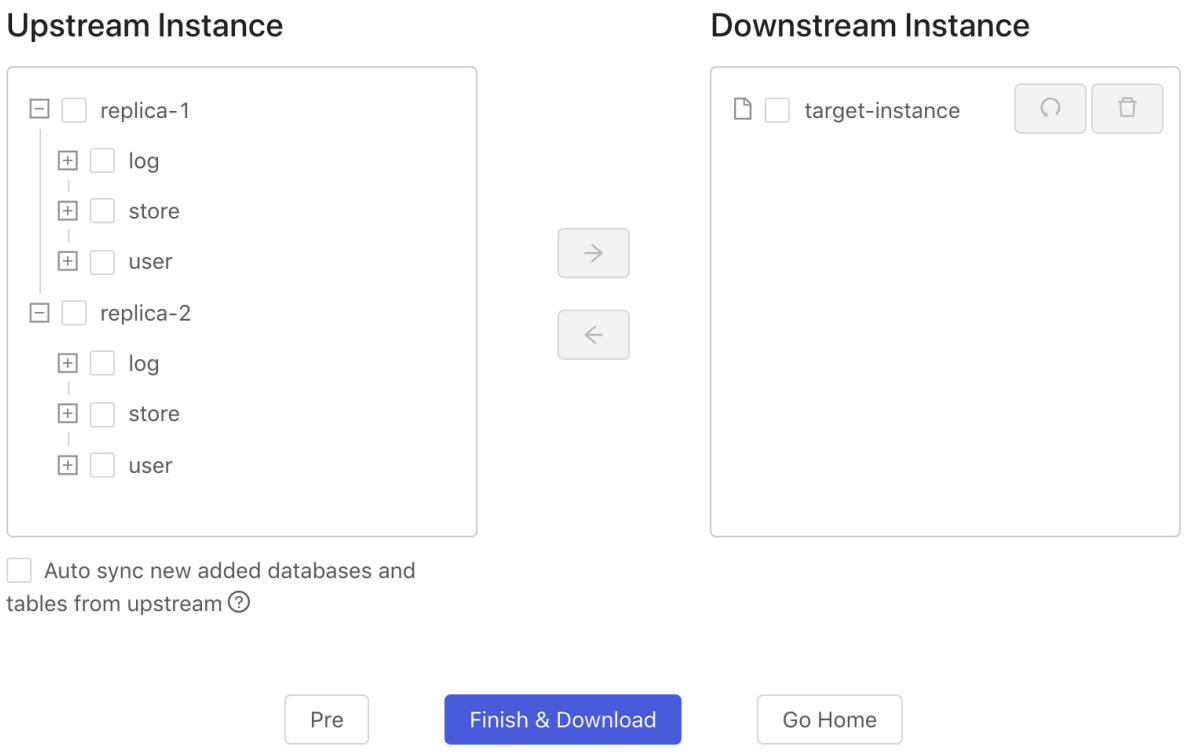


Figure 9: DM Portal InstanceShow

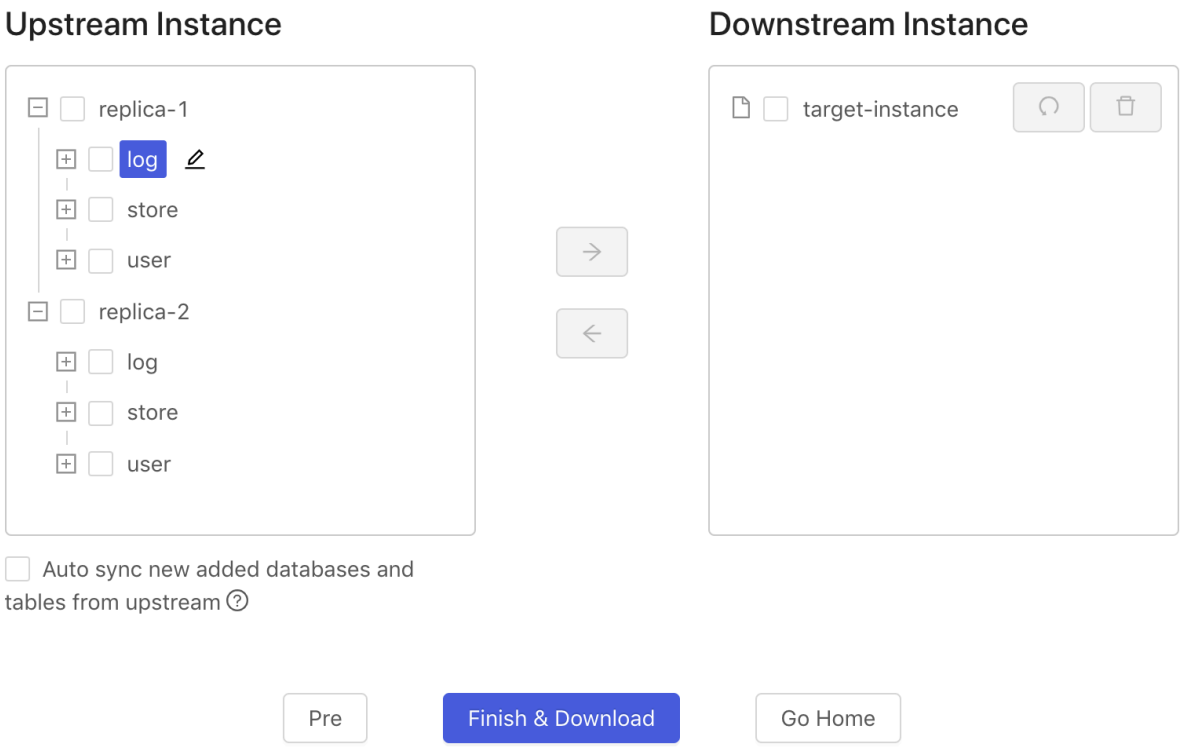


Figure 10: DM Portal BinlogFilter 1

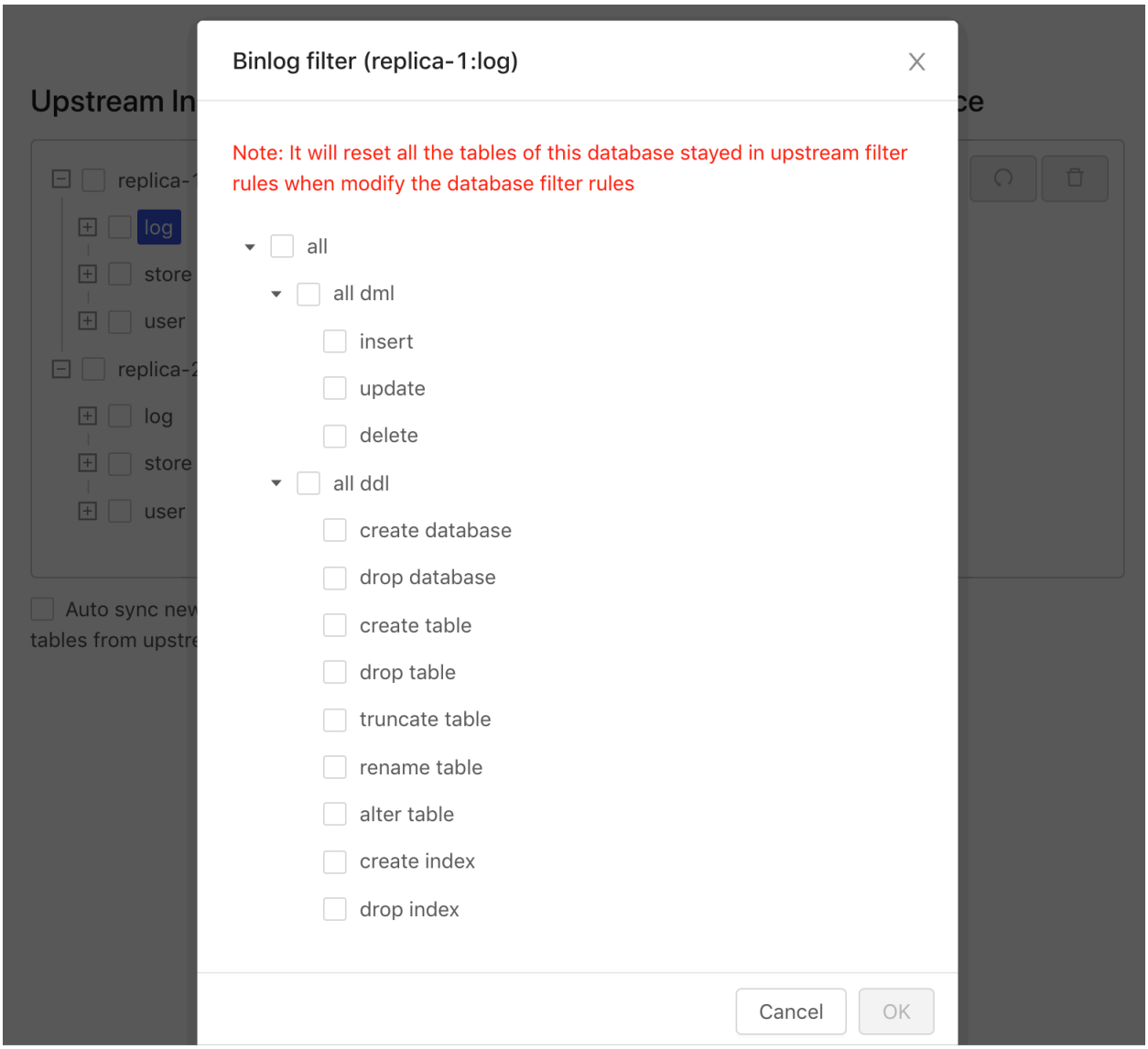


Figure 11: DM Portal BinlogFilter 2

12.4.5 Configure table routing

This feature is used to perform the following operations:

- Select the databases and tables that need to be synced, modify their names, and merge databases and tables
- Revert the last operation
- Reset all configurations of table routing

After the task configuration is completed, DM Portal generates the corresponding `task.yaml` file.

12.4.5.1 Prerequisites

The required binlog filter rules are configured.

> **Note:**
>
> - Batch operation is not supported when you merge databases and tables. You can only drag them one by one.
> - You can only drag tables when you merge databases and tables. You cannot drag databases.

12.4.5.2 Operation steps

1. Select the databases and tables that need to be synced from **Upstream Instance**.
2. Click the **Move** button and move the selected databases and tables to **Downstream Instance**.

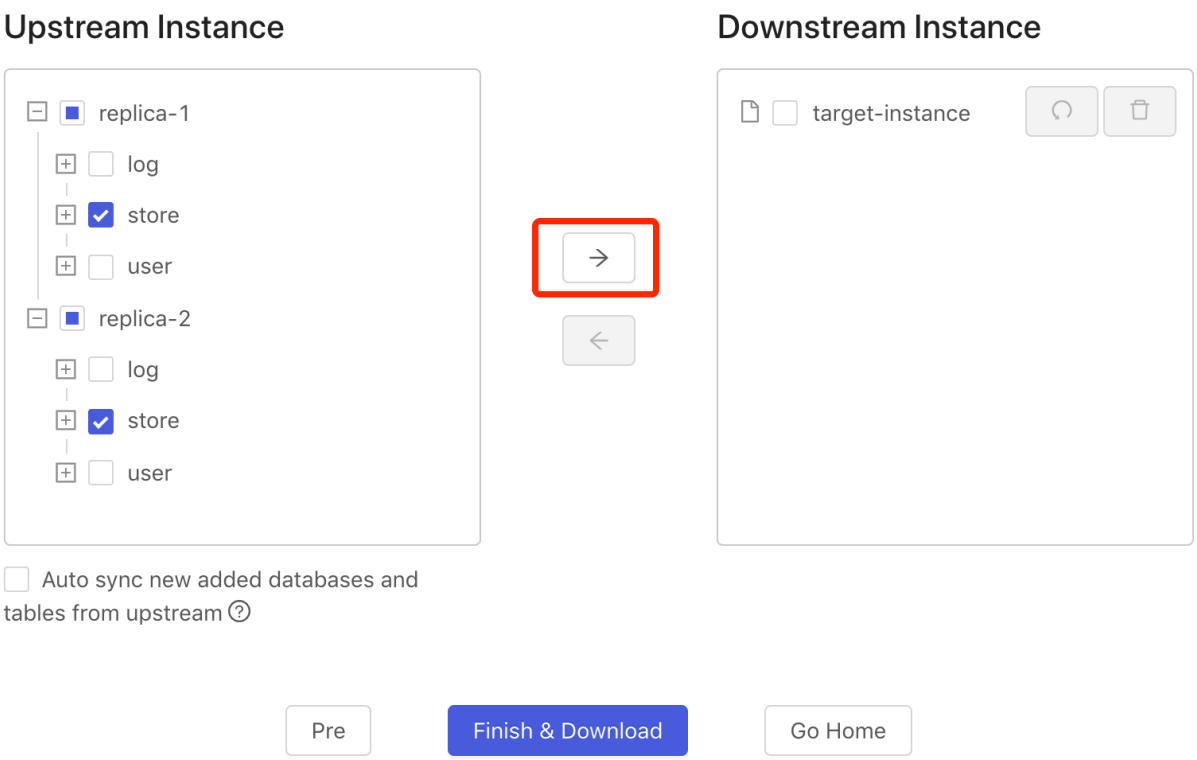


Figure 12: DM Portal TableRoute 1

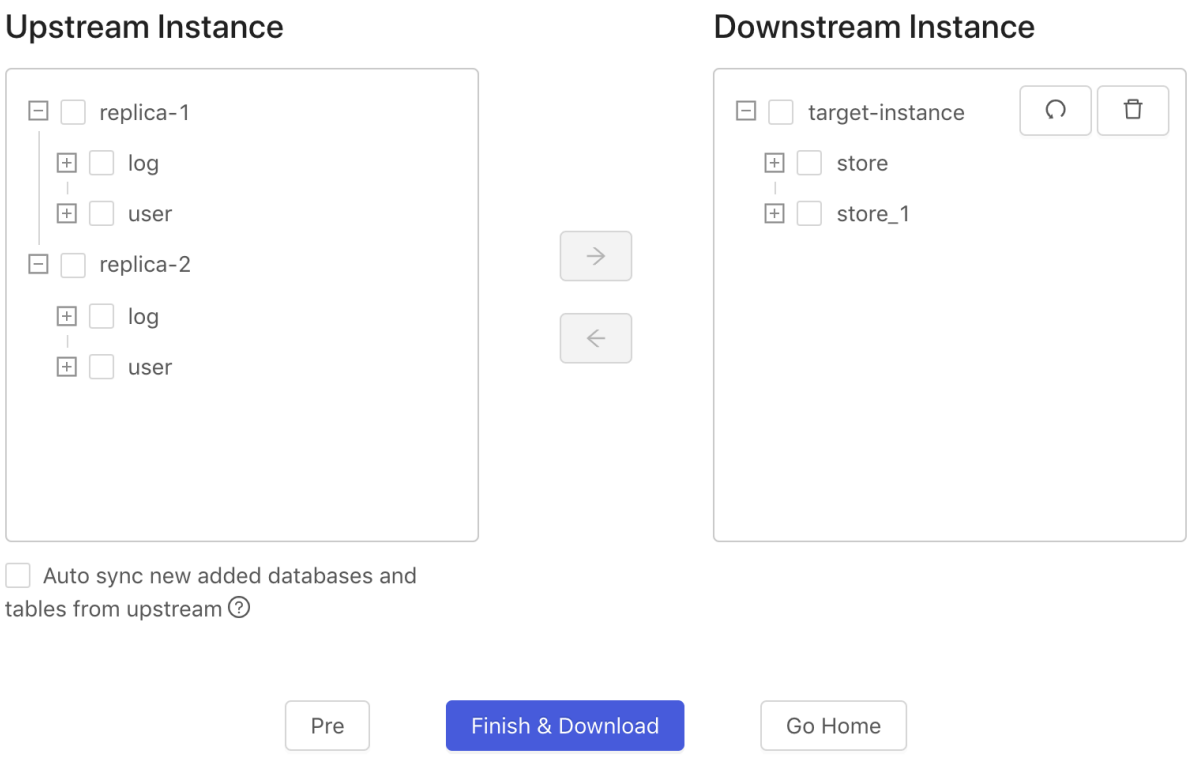


Figure 13: DM Portal TableRoute 2

3. Right click the databases and tables to rename them.

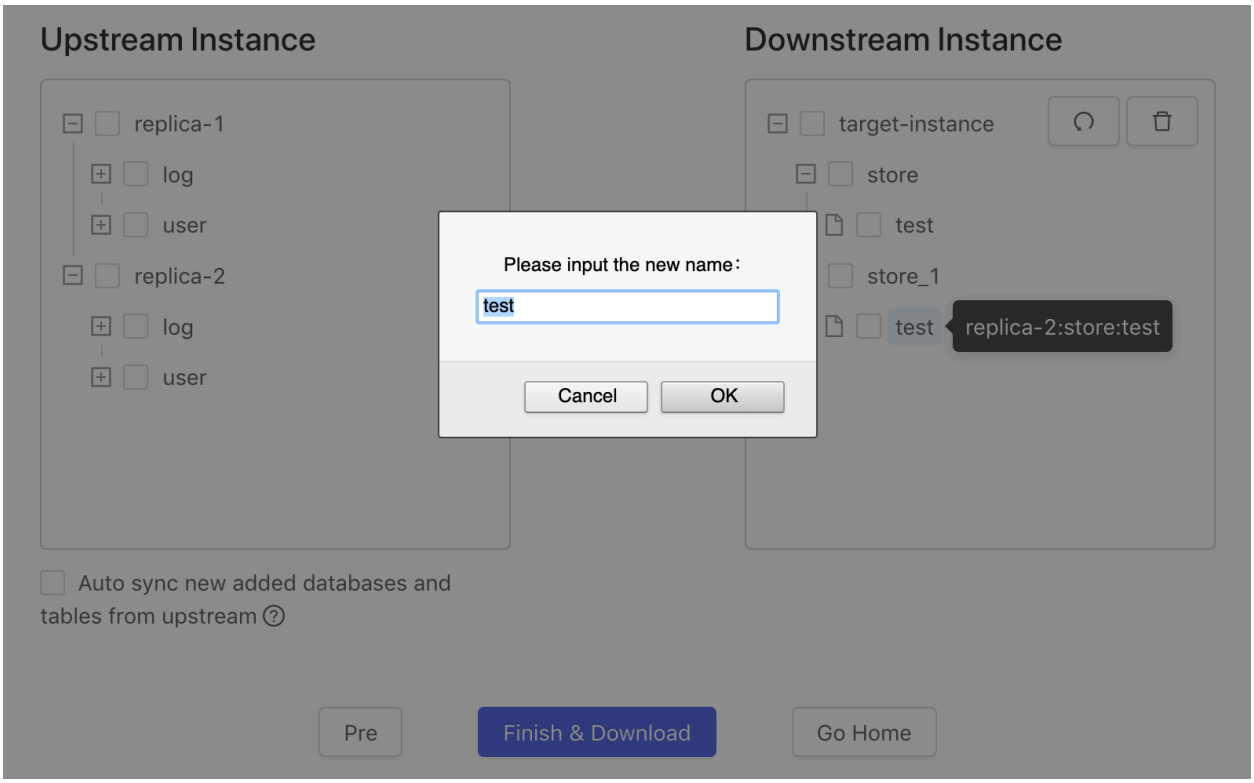


Figure 14: DM Portal ChangeTableName

4. Select the required table to perform the following operation:

    - To merge two tables, drag the table onto another table

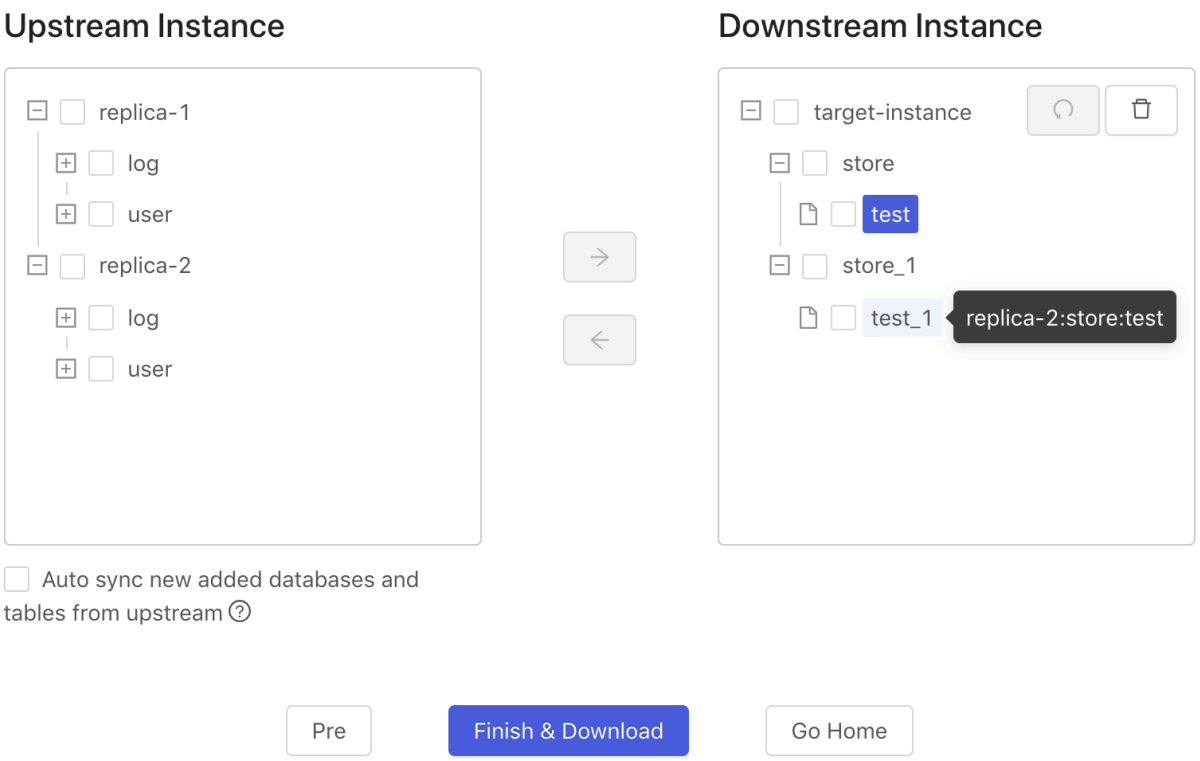


Figure 15: DM Portal MergeTable 1

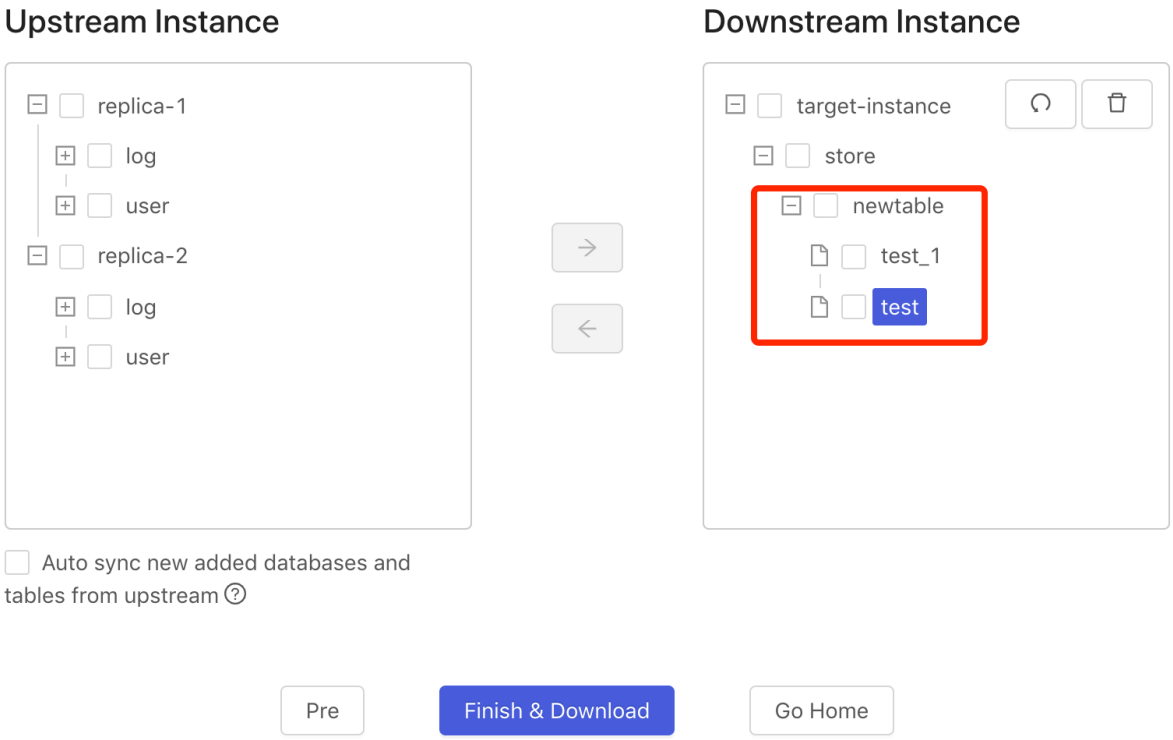


Figure 16: DM Portal MergeTable 2

- To move the table to an existing database, drag the table onto the database

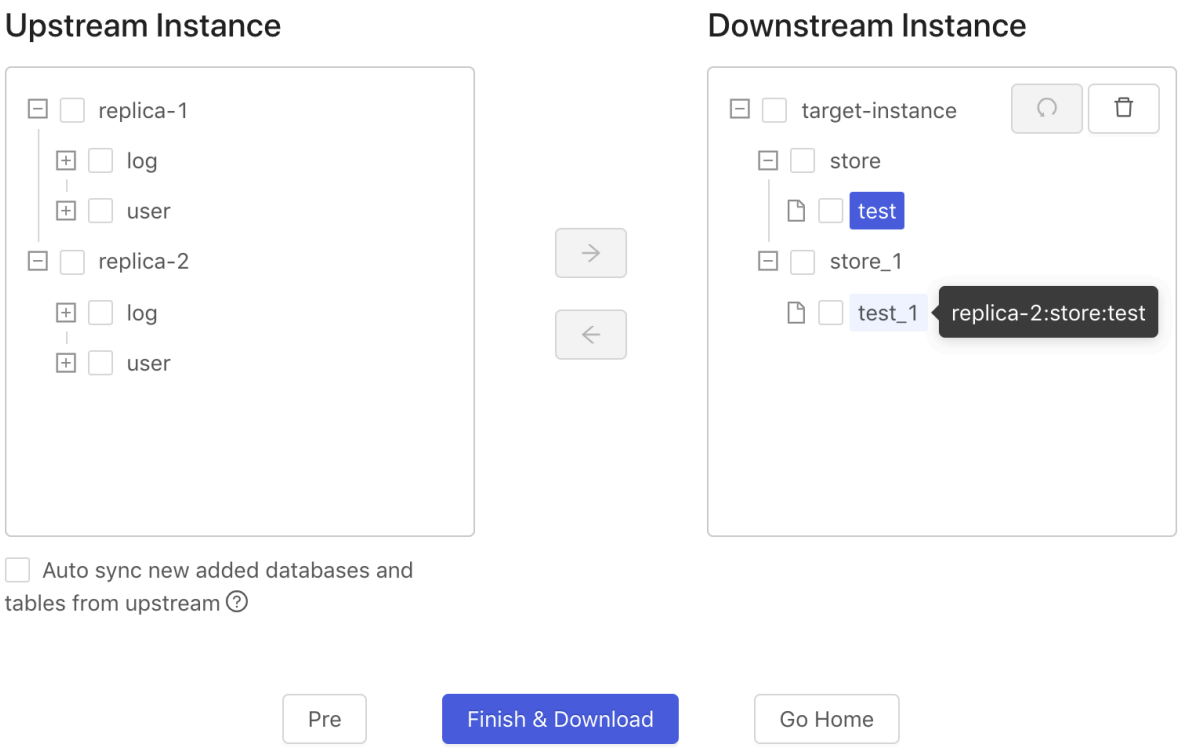


Figure 17: DM Portal MoveToDB 1

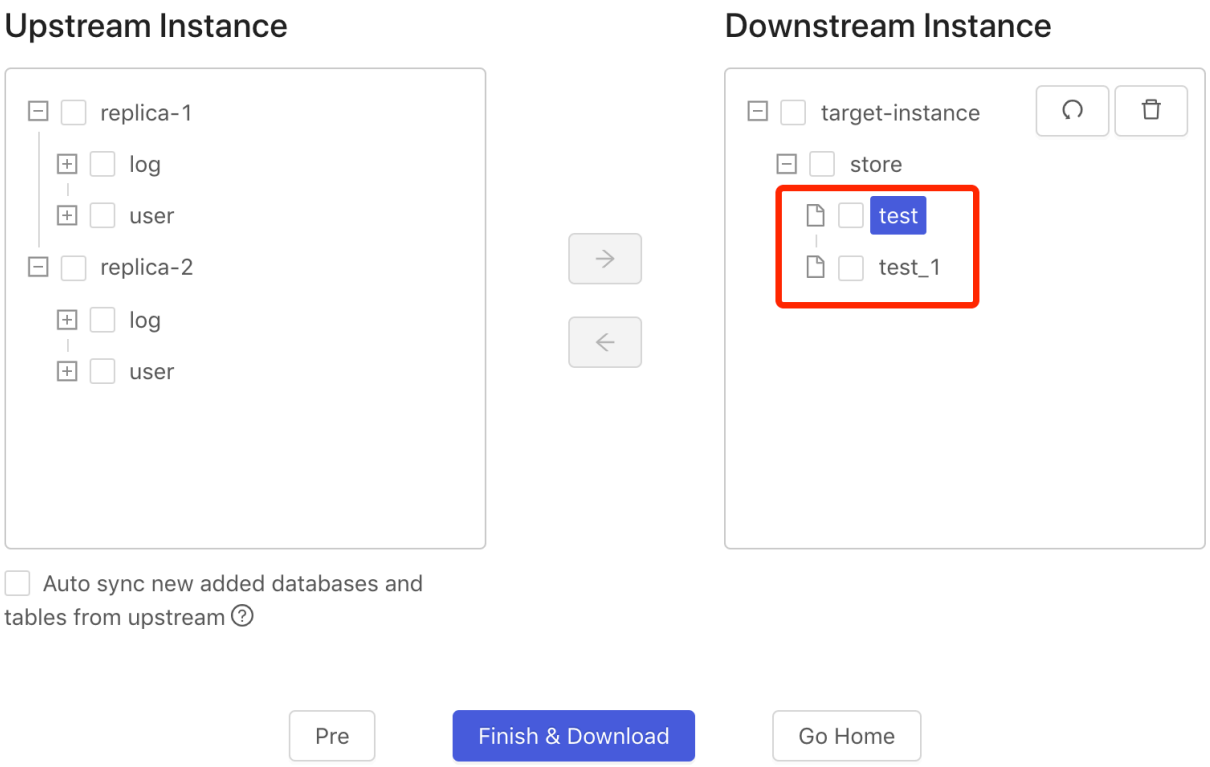


Figure 18: DM Portal MoveToDB 2

- To move the table to a new database, drag the table onto the `target-instance` icon

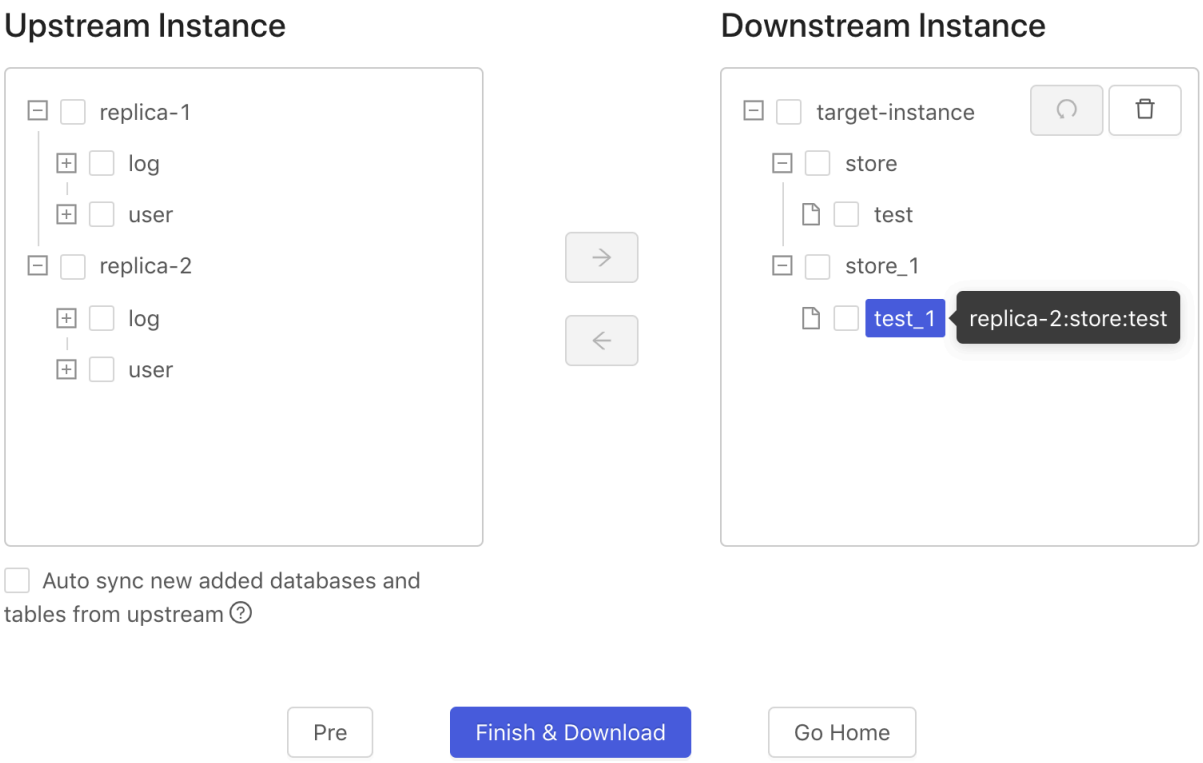


Figure 19: DM Portal MoveToNewDB 1

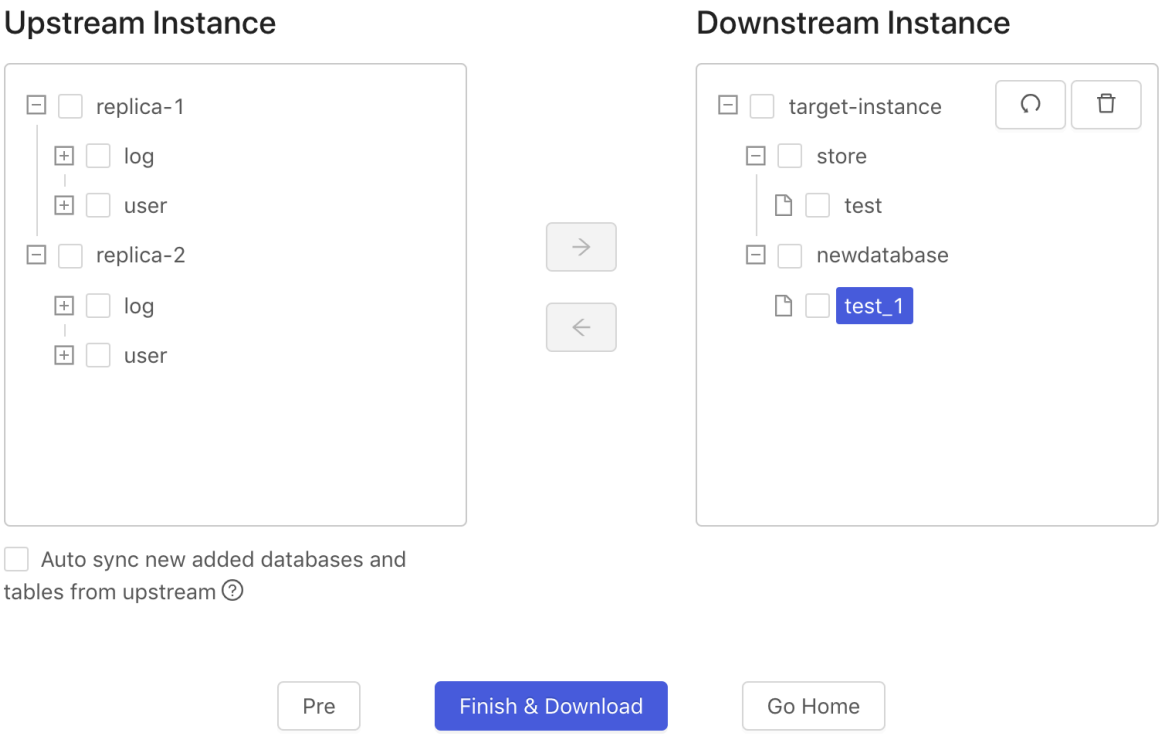


Figure 20: DM Portal MoveToNewDB 2

5. Click **Go Back** to undo the last operation.

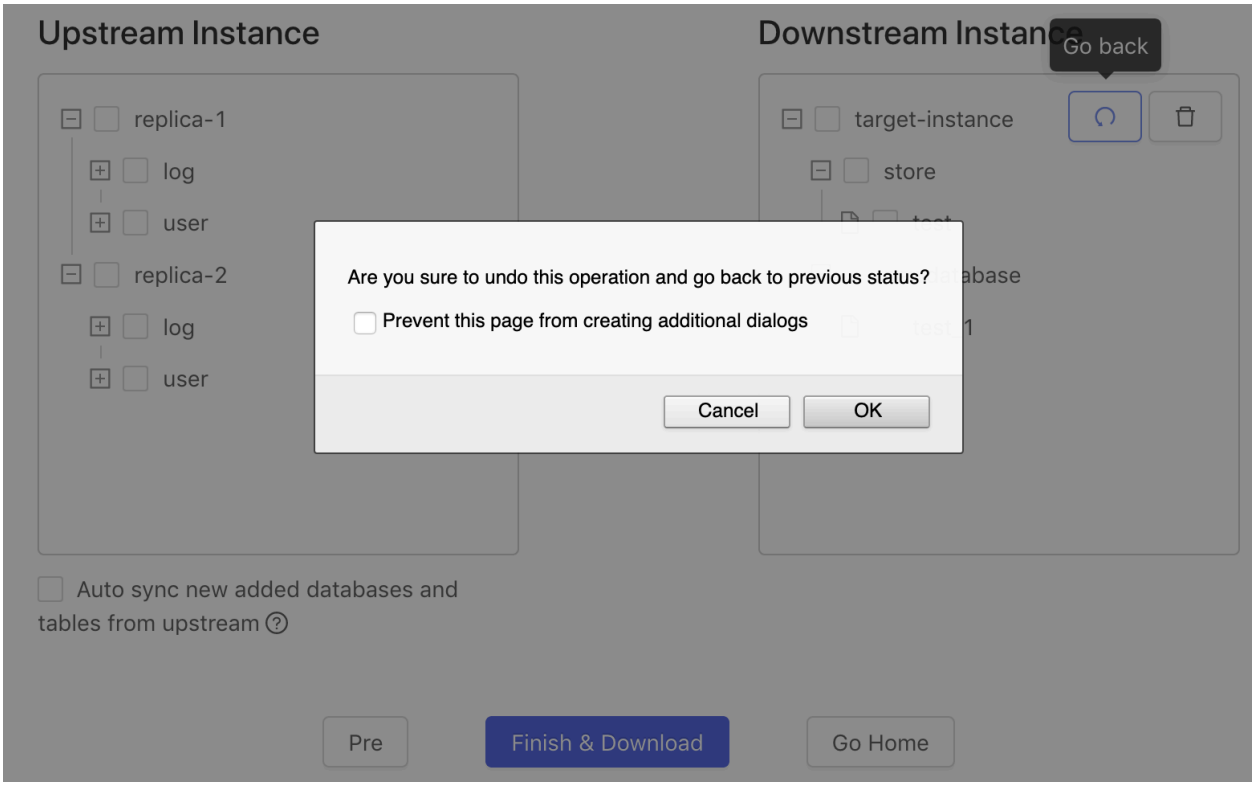


Figure 21: DM Portal Revert

6. Click **Reset** to clear the downstream instance.

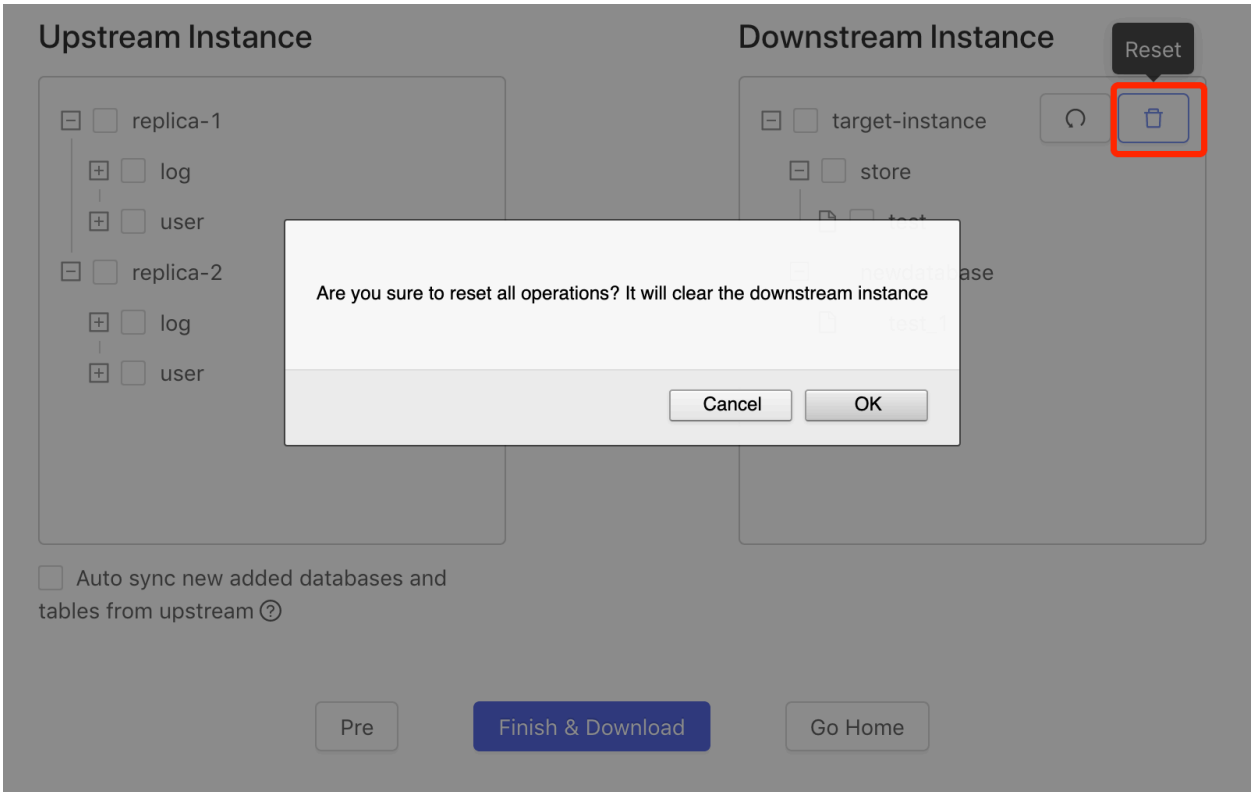


Figure 22: DM Portal Reset

7. Click **Finish & Download**. DM Portal automatically downloads the `task.yaml` file to the local computer, and creates a `task.yaml` configuration file in the `/tmp/` directory on the DM Portal server.

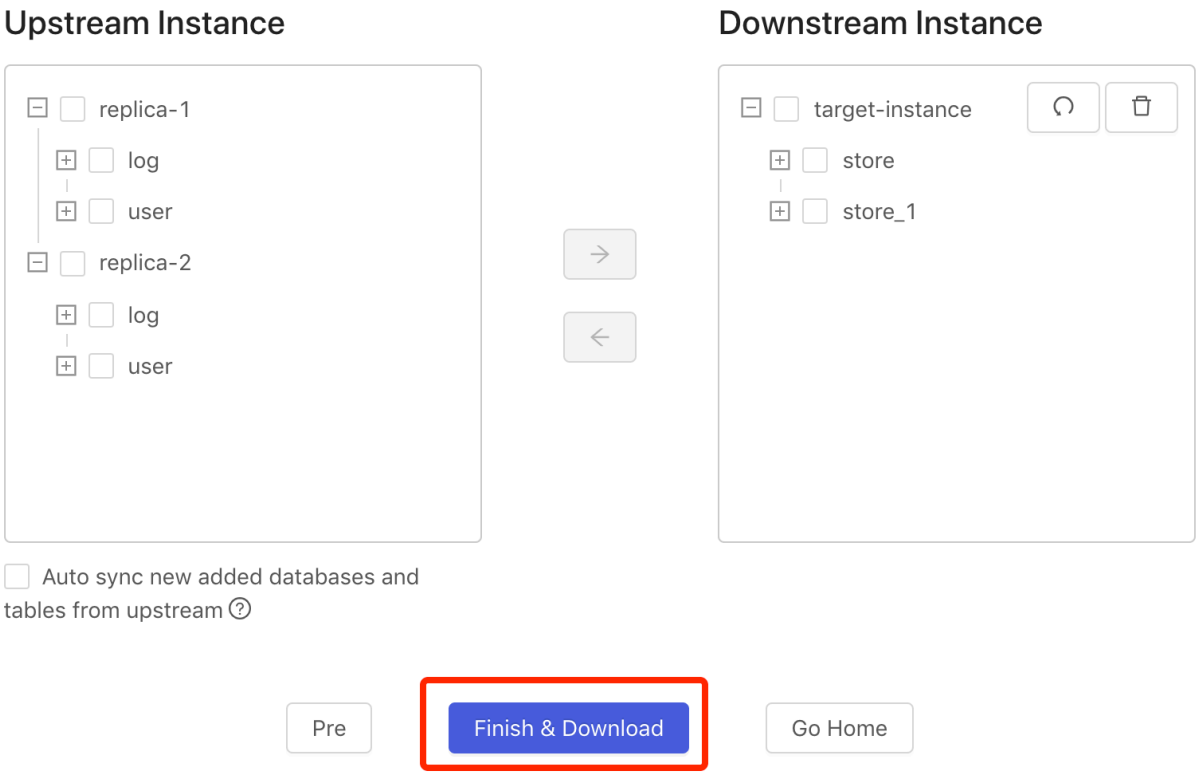


Figure 23: DM Portal GenerateConfig

13 Alert

13.1 DM Alert Information

The alert system is deployed by default when you deploy a DM cluster using DM-Ansible.

> **Note:**
>
> There are alert rules provided with DM-worker only.

For more information about DM alert rules and the solutions, refer to handle alerts.

Both DM alert information and monitoring metrics are based on Prometheus. For more information about their relationship, refer to DM monitoring metrics.

13.2 Handle Alerts

This document introduces how to deal with the alert information in DM.

13.2.1 Alert rules related to task status

13.2.1.1 `DM_task_state`

- Description:

  When a sub-task of DM-worker is in the `Paused` state for over 20 minutes, an alert is triggered.

- Solution:

  Refer to Troubleshoot DM.

13.2.2 Alert rules related to relay log

13.2.2.1 `DM_relay_process_exits_with_error`

- Description:

  When the relay log processing unit encounters an error, this unit moves to `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

13.2.2.2 `DM_remain_storage_of_relay_log`

- Description:

  When the free space of the disk where the relay log is located is less than 10G, an alert is triggered.

- Solutions:

  You can take the following methods to handle the alert:

  - Delete unwanted data manually to increase free disk space.
  - Reconfigure the automatic data purge strategy of the relay log or purge data manually.
  - Migrate the DM-worker instance to a disk with enough free space.

13.2.2.3 `DM_relay_log_data_corruption`

- Description:

  When the relay log processing unit validates the binlog event read from the upstream and detects abnormal checksum information, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

13.2.2.4 `DM_fail_to_read_binlog_from_master`

- Description:

  If an error occurs when the relay log processing unit tries to read the binlog event from the upstream, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

13.2.2.5 `DM_fail_to_write_relay_log`

- Description:

  If an error occurs when the relay log processing unit tries to write the binlog event into the relay log file, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

13.2.2.6 `DM_binlog_file_gap_between_master_relay`

- Description:

  When the number of the binlog files in the current upstream MySQL/MariaDB exceeds that of the latest binlog files pulled by the relay log processing unit by **more than** 1 for 10 minutes, and an alert is triggered.

- Solution:

  Refer to Troubleshoot DM.

13.2.3 Alert rules related to Dump/Load

13.2.3.1 `DM_dump_process_exists_with_error`

- Description:

  When the Dump processing unit encounters an error, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

13.2.3.2 `DM_load_process_exists_with_error`

- Description:

  When the Load processing unit encounters an error, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

13.2.4 Alert rules related to binlog replication

13.2.4.1 `DM_sync_process_exists_with_error`

- Description:

  When the binlog replication processing unit encounters an error, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

13.2.4.2 `DM_binlog_file_gap_between_master_syncer`

- Description:

  When the number of the binlog files in the current upstream MySQL/MariaDB exceeds that of the latest binlog files processed by the relay log processing unit by **more than** 1 for 10 minutes, an alert is triggered.

- Solution:

  Refer to Handle Performance Issues.

13.2.4.3 DM_binlog_file_gap_between_relay_syncer

- Description:

  When the number of the binlog files in the current relay log processing unit exceeds that of the latest binlog files processed by the binlog replication processing unit by **more than** 1 for 10 minutes, an alert is triggered.

- Solution:

  Refer to Handle Performance Issues.

14 Troubleshoot

14.1 Handle Errors

This document introduces the error system and how to handle common errors when you use DM.

14.1.1 Error system

In the error system, usually, the information of a specific error is as follows:

- `code`: error code.

  DM uses the same error code for the same error type. An error code does not change as the DM version changes.

  Some errors might be removed during the DM iteration, while the error codes are not. DM uses a new error code instead of an existing one for a new error.

- `class`: error type.

  It is used to mark the component where an error occurs (error source).

  The following table displays all error types, error sources, and error samples.

  |

  Error Type

  ```
  |   Error Source          | Error Sample
  ↪                                                        |
  ```

  :———— | :——————————— | :————————————————————— |

  `database` | Database operations | `[code=10003:class=database:scope=downstream ↪ :level=medium] database driver: invalid connection` |

functional | Underlying functions of DM | [code=11005:class=functional:scope
↪ =internal:level=high] not allowed operation: alter multiple tables
↪ in one statement |
config | Incorrect configuration | [code=20005:class=config:scope=internal:
↪ level=medium] empty source-id not valid |
binlog-op | Binlog operations | [code=22001:class=binlog-op:scope=internal:
↪ level=high] empty UUIDs not valid |
checkpoint | checkpoint operations | [code=24002:class=checkpoint:scope=
↪ internal:level=high] save point bin.1234 is older than current pos
↪ bin.1371 |
task-check | Performing task check | [code=26003:class=task-check:scope=
↪ internal:level=medium] new table router error |
relay-event-lib| Executing the basic functions of the relay module | [code=28001:
↪ class=relay-event-lib:scope=internal:level=high] parse server-uuid.
↪ index |
relay-unit | relay processing unit | [code=30015:class=relay-unit:scope=
↪ upstream:level=high] TCPReader get event: ERROR 1236 (HY000): Could
↪ not open log file |
dump-unit | dump processing unit | [code=32001:class=dump-unit:scope=internal
↪ :level=high] mydumper runs with error: CRITICAL **: 15:12:17.559:
↪ Error connecting to database: Access denied for user 'root'@'172.17.0.1'
↪ (using password: NO) |
load-unit | load processing unit | [code=34002:class=load-unit:scope=internal
↪ :level=high] corresponding ending of sql: ')' not found |
sync-unit | sync processing unit | [code=36027:class=sync-unit:scope=internal
↪ :level=high] Column count doesn't match value count: 9 (columns)vs
↪ 10 (values) |
dm-master | DM-master service | [code=38008:class=dm-master:scope=internal
↪ :level=high] grpc request error: rpc error: code = Unavailable desc
↪ = all SubConns are in TransientFailure, latest connection error:
↪ connection error: desc = "transport: Error while dialing dial tcp
↪ 172.17.0.2:8262: connect: connection refused" |
dm-worker | DM-worker service | [code=40066:class=dm-worker:scope=internal
↪ :level=high] ExecuteDDL timeout, try use query-status to query
↪ whether the DDL is still blocking |
dm-tracer | DM-tracer service | [code=42004:class=dm-tracer:scope=internal:
↪ level=medium] trace event test.1 not found |
schema-tracker | schema-tracker (during incremental data replication) | [code
↪ =44006:class=schema-tracker:scope=internal:level=high],"cannot track
↪ DDL: ALTER TABLE test DROP COLUMN col1" |
scheduler | Scheduling operations (of data migration tasks) | [code=46001:class=
↪ scheduler:scope=internal:level=high],"the scheduler has not started"
|
dmctl | An error occurs within dmctl or when it interacts with other components |
[code=48001:class=dmctl:scope=internal:level=high],"can not create grpc

```
↪ connection" |
```

- `scope`: Error scope.

  It is used to mark the scope and source of DM objects when an error occurs. `scope` includes four types: `not-set`, `upstream`, `downstream`, and `internal`.

  If the logic of the error directly involves requests between upstream and downstream databases, the scope is set to `upstream` or `downstream`; otherwise, it is currently set to `internal`.

- `level`: Error level.

  The severity level of the error, including `low`, `medium`, and `high`.

  - The `low` level error usually relates to user operations and incorrect inputs. It does not affect migration tasks.
  - The `medium` level error usually relates to user configurations. It affects some newly started services; however, it does not affect the existing DM migration status.
  - The `high` level error usually needs your attention, since you need to resolve it to avoid the possible interruption of a migration task.

- `message`: Error descriptions.

  Detailed descriptions of the error. To wrap and store every additional layer of error message on the error call chain, the errors.Wrap mode is adopted. The message description wrapped at the outermost layer indicates the error in DM and the message description wrapped at the innermost layer indicates the error source.

- `workaround`: Error handling methods (optional)

  The handling methods for this error. For some confirmed errors (such as configuration errors), DM gives the corresponding manual handling methods in `workaround`.

- Error stack information (optional)

  Whether DM outputs the error stack information depends on the error severity and the necessity. The error stack records the complete stack call information when the error occurs. If you cannot figure out the error cause based on the basic information and the error message, you can trace the execution path of the code when the error occurs using the error stack.

For the complete list of error codes, refer to the error code lists.

14.1.2 Troubleshooting

If you encounter an error while running DM, take the following steps to troubleshoot this error:

1. Execute the `query-status` command to check the task running status and the error output.
2. Check the log files related to the error. The log files are on the DM-master and DM-worker nodes. To get key information about the error, refer to the error system. Then check the Handle Common Errors section to find the solution.
3. If the error is not covered in this document, and you cannot solve the problem by checking the log or monitoring metrics, you can contact the R&D.
4. After the error is resolved, restart the task using dmctl.

```
resume-task ${task name}
```

However, you need to reset the data migration task in some cases. For details, refer to Reset the Data Migration Task.

14.1.3 Handle common errors

|

Error Code

Error Description	How to Handle
code=10001	Abnormal database operation.	Further analyze the error message and error stack.
code=10002	The `bad connection` error from the underlying database. It usually indicates that the connection between DM and the downstream TiDB instance is abnormal (possibly caused by network failure, TiDB restart and so on) and the currently requested data is not sent to TiDB.	DM provides automatic recovery for such error. If the recovery is not successful for a long time, check the network or TiDB status.
code=10003	The `invalid connection` error from the underlying database. It usually indicates that the connection between DM and the downstream TiDB instance is abnormal (possibly caused by network failure, TiDB restart and so on) and the currently requested data is partly sent to TiDB.	DM provides automatic recovery for such error. If the recovery is not successful for a long time, further check the error message and analyze the information based on the actual situation.
code=10005	Occurs when performing the `QUERY` type SQL statements.	
code=10006	Occurs when performing the `EXECUTE` type SQL statements, including DDL statements and DML statements of the `INSERT`, `UPDATE`or `DELETE` type. For more detailed error information, check the error message which usually includes the error code and error information returned for database operations.	

|
code=11006 | Occurs when the built-in parser of DM parses the incompatible DDL statements. | Refer to Data Migration - incompatible DDL statements for solution. |
code=20010 | Occurs when decrypting the database password that is provided in task configuration. | Check whether the downstream database password provided in the configuration task is correctly encrypted using dmctl. |
code=26002 | The task check fails to establish database connection. For more detailed error information, check the error message which usually includes the error code and error information returned for database operations. | Check whether the machine where DM-master is located has permission to access the upstream. |
code=32001 | Abnormal dump processing unit | If the error message contains `mydumper: argument list too long.`, configure the table to be exported by manually adding the `--regex` regular expression in the Mydumper argument `extra-args` in the `task.yaml` file according to the block-allow list. For example, to export all tables named `hello`, add `--regex '.*\\.hello$'`; to export all tables, add `--regex '.*'`. |
code=38008 | An error occurs in the gRPC communication among DM components. | Check `class`. Find out the error occurs in the interaction of which components. Determine the type of communication error. If the error occurs when establishing gRPC connection, check whether the communication server is working normally. |

14.1.3.1 What can I do when a migration task is interrupted with the `invalid connection` error returned?

The `invalid connection` error indicates that anomalies have occurred in the connection between DM and the downstream TiDB database (such as network failure, TiDB restart, TiKV busy and so on) and that a part of the data for the current request has been sent to TiDB.

Because DM has the feature of concurrently migrating data to the downstream in migration tasks, several errors might occur when a task is interrupted. You can check these errors by using `query-status` or `query-error`.

- If only the `invalid connection` error occurs during the incremental replication process, DM retries the task automatically.
- If DM does not or fails to retry automatically because of version problems, use `stop-task` to stop the task and then use `start-task` to restart the task.

14.1.3.2 A migration task is interrupted with the `driver: bad connection` error returned

The `driver: bad connection` error indicates that anomalies have occurred in the connection between DM and the upstream TiDB database (such as network failure, TiDB restart and so on) and that the data of the current request has not yet been sent to TiDB at that moment.

The current version of DM automatically retries on error. If you use the previous version which does not support automatically retry, you can execute the `stop-task` command to

stop the task. Then execute `start-task` to restart the task.

14.1.3.3 The relay unit throws error `event from * in * diff from passed-in event *` or a migration task is interrupted with failing to get or parse binlog errors like `get binlog error ERROR 1236 (HY000)` and `binlog checksum mismatch, data may be corrupted` returned

During the DM process of relay log pulling or incremental replication, this two errors might occur if the size of the upstream binlog file exceeds **4 GB**.

Cause: When writing relay logs, DM needs to perform event verification based on binlog positions and the size of the binlog file, and store the replicated binlog positions as checkpoints. However, the official MySQL uses `uint32` to store binlog positions. This means the binlog position for a binlog file over 4 GB overflows, and then the errors above occur.

For relay units, manually recover migration using the following solution:

1. Identify in the upstream that the size of the corresponding binlog file has exceeded 4GB when the error occurs.
2. Stop the DM-worker.
3. Copy the corresponding binlog file in the upstream to the relay log directory as the relay log file.
4. In the relay log directory, update the corresponding `relay.meta` file to pull from the next binlog file. If you have specified `enable_gtid` to `true` for the DM-worker, you need to modify the GTID corresponding to the next binlog file when updating the `relay.meta` file. Otherwise, you don't need to modify the GTID.

   Example: when the error occurs, `binlog-name = "mysql-bin.004451"` and `binlog-pos = 2453`. Update them respectively to `binlog-name = "mysql-bin.004452"` and `binlog-pos = 4`, and update `binlog-gtid` to `f0e914ef-54cf-11e7-813d-6c92bf2fa791:1-138218058`.
5. Restart the DM-worker.

For binlog replication processing units, manually recover migration using the following solution:

1. Identify in the upstream that the size of the corresponding binlog file has exceeded 4GB when the error occurs.
2. Stop the migration task using `stop-task`.
3. Update the `binlog_name` in the global checkpoints and in each table checkpoint of the downstream `dm_meta` database to the name of the binlog file in error; update

`binlog_pos` to a valid position value for which migration has completed, for example, 4.

Example: the name of the task in error is `dm_test`, the corresponding s`source-id` is `replica-1`, and the corresponding binlog file is `mysql-bin|000001.004451`. Execute the following command:

```
UPDATE dm_test_syncer_checkpoint SET binlog_name='mysql-bin
    ↪ |000001.004451', binlog_pos = 4 WHERE id='replica-1';
```

4. Specify `safe-mode: true` in the `syncers` section of the migration task configuration to ensure re-entrant.
5. Start the migration task using `start-task`.
6. View the status of the migration task using `query-status`. You can restore `safe-` ↪ `mode` to the original value and restart the migration task when migration is done for the original error-triggering relay log files.

14.1.3.4 `Access denied for user 'root'@'172.31.43.27' (using password: YES)` shows when you query the task or check the log

For database related passwords in all the DM configuration files, it is recommended to use the passwords encrypted by `dmctl`. If a database password is empty, it is unnecessary to encrypt it. For how to encrypt the plaintext password, see Encrypt the upstream MySQL user password using dmctl.

In addition, the user of the upstream and downstream databases must have the corresponding read and write privileges. Data Migration also prechecks the corresponding privileges automatically while starting the data migration task.

14.2 Handle Performance Issues

This document introduces common performance issues that might exist in DM and how to deal with them.

Before diagnosing an issue, you can refer to the DM 1.0-GA Benchmark Report.

When diagnosing and handling performance issues, make sure that:

- The DM monitoring component is correctly configured and installed.
- You can view monitoring metrics on the Grafana monitoring dashboard.
- The component you diagnose works well; otherwise, possible monitoring metrics exceptions might interfere with the diagnosis of performance issues.

In the case of a large latency in the data migration, to quickly figure out whether the bottleneck is inside the DM component or in the TiDB cluster, you can first check `DML` ↪ `queue remain length` in Write SQL Statements to Downstream.

14.2.1 relay log unit

To diagnose performance issues in the relay log unit, you can check the `binlog file gap between master and relay` monitoring metric. For more information about this metric, refer to monitoring metrics of the relay log. If this metric is greater than 1 for a long time, it usually indicates that there is a performance issue; if this metric is 0, it usually indicates that there is no performance issue.

If the value of `binlog file gap between master and relay` is 0, but you suspect that there is a performance issue, you can check `binlog pos`. If `master` in this metric is much larger than `relay`, a performance issue might exist. In this case, diagnose and handle this issue accordingly.

14.2.1.1 Read binlog data

`read binlog event duration` refers to the duration that the relay log reads binlog from the upstream database (MySQL/MariaDB). Ideally, this metric is close to the network latency between DM-worker and MySQL/MariaDB instances.

- For data migration in one data center, reading binlog data is not a performance bottleneck. If the value of `read binlog event duration` is too large, check the network connection between DM-worker and MySQL/MariaDB.
- For data migration in the geo-distributed environment, try to deploy DM-worker and MySQL/MariaDB in one data center, while deploying the TiDB cluster in the target data center.

The process of reading binlog data from the upstream database includes the following sub-processes:

- The upstream MySQL/MariaDB reads the binlog data locally and sends it through the network. When no exception occurs in the MySQL/MariaDB load, this sub-process usually does not become a bottleneck.
- The binlog data is transferred from the machine where MySQL/MariaDB is located to the machine where DM-worker is located via the network. Whether this sub-process becomes a bottleneck mainly depends on the network connection between DM-worker and the upstream MySQL/MariaDB.
- DM-worker reads binlog data from the network data stream and constructs it as a binlog event. When no exception occurs in the DM-worker load, this sub-process usually does not become a bottleneck.

Note:

If the value of `read binlog event duration` is large, another possible reason is that the upstream MySQL/MariaDB has a low load. This means that no binlog event needs to be sent to DM for a period of time, and the relay log unit stays in a wait state, thus this value includes additional waiting time.

14.2.1.2 binlog data decoding and verification

After reading the binlog event into the DM memory, DM's relay processing unit decodes and verifies data. This usually does not lead to performance bottleneck; therefore, there is no related performance metric on the monitoring dashboard by default. If you need to view this metric, you can manually add a monitoring item in Grafana. This monitoring item corresponds to `dm_relay_read_transform_duration`, a metric from Prometheus.

14.2.1.3 Write relay log files

When writing a binlog event to a relay log file, the relevant performance metric is `write relay log duration`. This value should be microseconds when `binlog event size` is not too large. If `write relay log duration` is too large, check the write performance of the disk. To avoid low write performance, use local SSDs for DM-worker.

14.2.2 Load unit

The main operations of the Load unit are to read the SQL file data from the local and write it to the downstream. The related performance metric is `transaction execution latency`. If this value is too large, check the downstream performance by checking the monitoring of the downstream database. You can also check whether there is a large network latency between DM and the downstream database.

14.2.3 Binlog replication unit

To diagnose performance issues in the Binlog replication unit, you can check the `binlog file gap between master and syncer` monitoring metric. For more information about this metric, refer to monitoring metrics of the Binlog replication.

- If this metric is greater than 1 for a long time, it usually indicates that there is a performance issue.
- If this metric is 0, it usually indicates that there is no performance issue.

When `binlog file gap between master and syncer` is greater than 1 for a long time, check `binlog file gap between relay and syncer` to figure out which unit the latency

mainly exists in. If this value is usually 0, the latency might exist in the relay log unit. Then you can refer to relay log unit to resolve this issue; otherwise, continue checking the Binlog replication unit.

14.2.3.1 Read binlog data

The Binlog replication unit decides whether to read the binlog event from the upstream MySQL/MariaDB or from the relay log file according to the configuration. The related performance metric is `read binlog event duration`, which generally ranges from a few microseconds to tens of microseconds.

- If DM's Binlog replication processing unit reads the binlog event from upstream MySQL/MariaDB, to locate and resolve the issue, refer to read binlog data in the "relay log unit" section.
- If DM's Binlog replication processing unit reads the binlog event from the relay log file, when `binlog event size` is not too large, the value of `read binlog event duration` should be microseconds. If `read binlog event duration` is too large, check the read performance of the disk. To avoid low write performance, use local SSDs for DM-worker.

14.2.3.2 binlog event conversion

The Binlog replication unit constructs DML, parses DDL, and performs table router conversion from binlog event data. The related metric is `transform binlog event duration`.

The duration is mainly affected by the write operations upstream. Take the `INSERT INTO` statement as an example, the time consumed to convert a single `VALUES` greatly differs from that to convert a lot of `VALUES`. The time consumed might range from tens of microseconds to hundreds of microseconds. However, usually this is not a bottleneck of the system.

14.2.3.3 Write SQL statements to downstream

When the Binlog replication unit writes the converted SQL statements to the downstream, the related performance metrics are `DML queue remain length` and `transaction execution latency`.

After constructing SQL statements from binlog event, DM uses `worker-count` queues to concurrently write these statements to the downstream. However, to avoid too many monitoring entries, DM performs the modulo `8` operation on the IDs of concurrent queues. This means that all concurrent queues correspond to one item from `q_0` to `q_7`.

`DML queue remain length` indicates in the concurrent processing queue, the number of DML statements that have not been consumed and have not started to be written downstream. Ideally, the curves corresponding to each `q_*` are almost the same. If not, it indicates that the concurrent load is extremely unbalanced.

If the load is not balanced, confirm whether tables need to be migrated have primary keys or unique keys. If these keys do not exist, add the primary keys or the unique keys; if these keys do exist while the load is not balanced, upgrade DM to v1.0.5 or later versions.

- When there is no noticeable latency in the entire data migration link, the corresponding curve of `DML queue remain length` is almost always 0, and the maximum does not exceed the value of `batch` in the task configuration file.
- If you find a noticeable latency in the data migration link, and the curve of `DML queue remain length` corresponding to each `q_*` is almost the same and is almost always 0, it means that DM fails to read, convert, or concurrently write the data from the upstream in time (the bottleneck might be in the relay log unit). For troubleshooting, refer to the previous sections of this document.

If the corresponding curve of `DML queue remain length` is not 0 (usually the maximum is not more than 1024), it indicates that there is a bottleneck when writing SQL statements to the downstream. You can use `transaction execution latency` to view the time consumed to execute a single transaction to the downstream.

`transaction execution latency` is usually tens of milliseconds. If this value is too large, check the downstream performance based on the monitoring of the downstream database. You can also check whether there is a large network latency between DM and the downstream database.

To view the time consumed to write a single statement such as `BEGIN`, `INSERT`, `UPDATE`, `DELETE`, or `COMMIT` to the downstream, you can also check `statement execution latency`.

15 TiDB Data Migration FAQ

This document collects the frequently asked questions (FAQs) about TiDB Data Migration (DM).

15.1 Does DM support migrating data from Alibaba RDS or other cloud databases?

Currently, DM only supports decoding the standard version of MySQL or MariaDB binlog. It has not been tested for Alibaba Cloud RDS or other cloud databases. If you are confirmed that its binlog is in standard format, then it is supported.

Here are some known incompatible issues:

- In **Alibaba Cloud RDS**, for an upstream table with no primary key, its binlog still contains a hidden primary key column, which is inconsistent with the original table structure.
- In **HUAWEI Cloud RDS**, directly reading binlog files is not supported. For more details, see Can HUAWEI Cloud RDS Directly Read Binlog Backup Files?

15.2 Does the regular expression of the block and allow list in the task configuration support `non-capturing (?!)`?

Currently, DM does not support it and only supports the regular expressions of the Golang standard library. See regular expressions supported by Golang via re2-syntax.

15.3 If a statement executed upstream contains multiple DDL operations, does DM support such migration?

DM will attempt to split a single statement containing multiple DDL change operations into multiple statements containing only one DDL operation, but might not cover all cases. It is recommended to include only one DDL operation in a statement executed upstream, or verify it in the test environment. If it is not supported, you can file an issue to the DM repository.

15.4 How to handle incompatible DDL statements?

When you encounter a DDL statement unsupported by TiDB, you need to manually handle it using dmctl (skipping the DDL statement or replacing the DDL statement with a specified DDL statement). For details, see Skip or replace abnormal SQL statements.

> **Note:**
>
> Currently, TiDB is not compatible with all the DDL statements that MySQL supports. See MySQL Compatibility.

15.5 How to reset the data migration task?

15.5.1 Reset the data migration task when the relay log is in the normal state

If the relay log required by the data migration task is normal, you can use the following steps to reset the data migration task:

1. Use `stop-task` to stop abnormal data migration tasks.
2. Clean up the downstream migrated data.
3. Choose one of the following methods to restart the data migration task:
    - Modify the task configuration file to specify a new task name, and then use `start-task` to restart the migration task.
    - Modify the task configuration file to set `remove-meta` to `true`, and then use `start-task` to restart the migration task.

15.5.2 Reset the data migration task when the relay log is in the abnormal state

15.5.2.1 The required relay log exists in upstream MySQL

If the relay log required by the migration task is abnormal in the DM-worker, but is normal in the upstream MySQL, you can use the following steps to restore the data migration task:

1. Use the `stop-task` command to stop all the migration tasks that are currently running.
2. Refer to restart DM-worker to **stop** the abnormal DM-worker node.
3. Copy the normal binlog file from the upstream MySQL to replace the corresponding file in the relay log directory of DM-worker.
    - If the cluster is deployed using DM-Ansible, the relay log is in the `<deploy_dir ↪ >/relay_log` directory.
    - If the cluster is manually deployed using the binary, the relay log is in the directory set by the `relay-dir` parameter.
4. Modify the information of `relay.meta` in the relay log directory of DM-worker to the information corresponding to the next binlog file.
    - If `enable-gtid` is not enabled, set `binlog-name` to the file name of the next binlog file, and set `binlog-pos` to `4`. If you copy `mysq-bin.000100` from the upstream MySQL to the relay directory, and want to continue to pull binlog from `mysql-bin.000101` later, set `binlog-name` to `mysql-bin.000101`.
    - If `enable-gtid` is enabled, set `binlog-gtid` to the value corresponding to `Previous_gtids` at the beginning of the next binlog file. You can obtain the value by executing SHOW BINLOG EVENTS.
5. Refer to restart DM-worker to **start** the abnormal DM-worker node.
6. Use `start-task` to resume all stopped migration tasks.

15.5.2.2 The required relay log has been purged in upstream MySQL

If the relay log required by the migration task is abnormal in the DM-worker, and has been purged in the upstream MySQL, you can use the following steps to reset the data migration task:

1. Use the `stop-task` command to stop all the migration tasks that are currently running.
2. Use DM-Ansible to stop the entire DM cluster.
3. Manually clean up the relay log directory of the DM-worker corresponding to the MySQL cluster whose binlog is reset.

- If the cluster is deployed using DM-Ansible, the relay log is in the `<deploy_dir >/relay_log` directory.
- If the cluster is manually deployed using the binary, the relay log is in the directory set of the `relay-dir` parameter.

4. Clean up downstream migrated data.
5. Use DM-Ansible to start the entire DM cluster.
6. Choose one of the following methods to restart the data migration task:

    - Modify the task configuration file to specify a new task name, and then use `start -task` to restart the migration task.
    - Modify the task configuration file to set `remove-meta` to `true`, and then use `start-task` to restart the migration task.

15.6 How to handle the error returned by the DDL operation related to the gh-ost table, after `online-ddl-scheme: "gh-ost"` is set?

```
[unit=Sync] ["error information"="{\"msg\":\"[code=36046:class=sync-unit:
    ↪ scope=internal:level=high] online ddls on ghost table `xxx`.`
    ↪ _xxxx_gho`\\ngithub.com/pingcap/dm/pkg/terror.(*Error).Generate
    ↪ ......
```

The above error can be caused by the following reason:

In the last `rename ghost_table to origin table` step, DM reads the DDL information in memory, and restores it to the DDL of the origin table.

However, the DDL information in memory is obtained in either of the two ways:

- DM processes the gh-ost table during the `alter ghost_table` operation and records the DDL information of `ghost_table`;
- When DM-worker is restarted to start the task, DM reads the DDL from `dm_meta.{ task_name}_onlineddl`.

Therefore, in the process of incremental replication, if the specified Pos has skipped the `alter ghost_table` DDL but the Pos is still in the online-ddl process of gh-ost, the ghost_table is not written into memory or `dm_meta.{task_name}_onlineddl` correctly. In such cases, the above error is returned.

You can avoid this error by the following steps:

1. Remove the `online-ddl-scheme` configuration of the task.

2. Configure `_{table_name}_gho`, `_{table_name}_ghc`, and `_{table_name}_del` in `block-allow-list.ignore-tables`.
3. Execute the upstream DDL in the downstream TiDB manually.
4. After the Pos is replicated to the position after the gh-ost process, re-enable the `online-ddl-scheme` and comment out `block-allow-list.ignore-tables`.

15.7 How to add tables to the existing data migration tasks?

If you need to add tables to a data migration task that is running, you can address it in the following ways according to the stage of the task.

> **Note:**
>
> Because adding tables to an existing data migration task is complex, it is recommended that you perform this operation only when necessary.

15.7.1 In the `Dump` stage

Since MySQL cannot specify a snapshot for export, it does not support updating data migration tasks during the export and then restarting to resume the export through the checkpoint. Therefore, you cannot dynamically add tables that need to be migrated at the `Dump` stage.

If you really need to add tables for migration, it is recommended to restart the task directly using the new configuration file.

15.7.2 In the `Load` stage

During the export, multiple data migration tasks usually have different binlog positions. If you merge the tasks in the `Load` stage, they might not be able to reach consensus on binlog positions. Therefore, it is not recommended to add tables to a data migration task in the `Load` stage.

15.7.3 In the `Sync` stage

When the data migration task is in the `Sync` stage, if you add additional tables to the configuration file and restart the task, DM does not re-execute full export and import for the newly added tables. Instead, DM continues incremental replication from the previous checkpoint.

Therefore, if the full data of the newly added table has not been imported to the downstream, you need to use a separate data migration task to export and import the full data to the downstream.

Record the position information in the global checkpoint (`is_global=1`) corresponding to the existing migration task as `checkpoint-T`, such as `(mysql-bin.000100, 1234)`. Record the position information of the full export `metedata` (or the checkpoint of another data migration task in the `Sync` stage) of the table to be added to the migration task as `checkpoint-S`, such as `(mysql-bin.000099, 5678)`. You can add the table to the migration task by the following steps:

1. Use `stop-task` to stop an existing migration task. If the table to be added belongs to another running migration task, stop that task as well.
2. Use a MySQL client to connect the downstream TiDB database and manually update the information in the checkpoint table corresponding to the existing migration task to the smaller value between `checkpoint-T` and `checkpoint-S`. In this example, it is `(mysql- bin.000099, 5678)`.
    - The checkpoint table to be updated is `{task-name}_syncer_checkpoint` in the `{dm_meta}` schema.
    - The checkpoint rows to be updated match `id=(source-id)` and `is_global=1`.
    - The checkpoint columns to be updated are `binlog_name` and `binlog_pos`.
3. Set `safe-mode: true` for the `syncers` in the task to ensure reentrant execution.
4. Start the task using `start-task`.
5. Observe the task status through `query-status`. When `syncerBinlog` exceeds the larger value of `checkpoint-T` and `checkpoint-S`, restore `safe-mode` to the original value and restart the task. In this example, it is `(mysql-bin.000100, 1234)`.

15.8 In DM v1.0, why does the command `sql-skip` fail to skip some statements when the task is in error?

You need to first check whether the binlog position is still advancing after you execute `sql-skip`. If so, it means that `sql-skip` has taken effect. The reason why this error keeps occurring is that the upstream sends multiple unsupported DDL statements. You can use `sql-skip -s <sql-pattern>` to set a pattern to match these statements.

Sometimes, the error message contains the `parse statement` information, for example:

```
if the DDL is not needed, you can use a filter rule with \"*\" schema-
    ↪ pattern to ignore it.\n\t : parse statement: line 1 column 11 near \"
    ↪ EVENT `event_del_big_table` \r\nDISABLE\" %!(MISSING)(EXTRA string=
    ↪ ALTER EVENT `event_del_big_table` \r\nDISABLE
```

The reason for this type of error is that the TiDB parser cannot parse DDL statements sent by the upstream, such as `ALTER EVENT`, so `sql-skip` does not take effect as expected. You can add binlog event filters in the configuration file to filter those statements and set `schema-pattern: "*"`.

15.9 Why do `REPLACE` statements keep appearing in the downstream when DM is replicating?

You need to check whether the safe mode is automatically enabled for the task. If the task is automatically resumed after an error, or if there is high availability scheduling, then the safe mode is enabled because it is within 5 minutes after the task is started or resumed.

You can check the DM-worker log file and search for a line containing `change count`. If the `new count` in the line is not zero, the safe mode is enabled. To find out why it is enabled, check when it happens and if any errors are reported before.

16 Releases

16.1 v1.0

16.1.1 DM 1.0.7 Release Notes

Release date: June 21, 2021

DM version: 1.0.7

16.1.1.1 Bug fixes

- Fix the issue that data may be lost after a task restarts from interruption #1783

16.1.2 DM 1.0.6 Release Notes

Release date: June 17, 2020

DM version: 1.0.6

DM-Ansible version: 1.0.6

16.1.2.1 Improvements

- Support the original plaintext passwords for upstream and downstream databases
- Support configuring session variables for DM's connections to upstream and downstream databases

- Remove the call stack information in some error messages returned by the `query-status` command when the data migration task encounters an exception
- Filter out the items that pass the precheck from the message returned when the precheck of the data migration task fails

16.1.2.2 Bug fixes

- Fix the issue that the data migration task is not automatically paused and the error cannot be identified by executing the `query-status` command if an error occurs when the load unit creates a table
- Fix possible DM-worker panics when data migration tasks run simultaneously
- Fix the issue that the existing data migration task cannot be automatically restarted when the DM-worker process is restarted if the `enable-heartbeat` parameter of the task is set to `true`
- Fix the issue that the shard DDL conflict error may not be returned after the task is resumed
- Fix the issue that the `replicate lag` information is displayed incorrectly for an initial period of time when the `enable-heartbeat` parameter of the data migration task is set to `true`
- Fix the issue that `replicate lag` cannot be calculated using the heartbeat information when `lower_case_table_names` is set to `1` in the upstream database
- Disable the meaningless auto-resume tasks triggered by the `unsupported collation` error during data migration

16.1.2.3 Detailed bug fixes and changes

- Support the original plaintext passwords for upstream and downstream databases #676
- Support configuring session variables for DM's connections to upstream and downstream databases #692
- Remove the call stack information in some error messages returned by the `query-status` command when the data migration task encounters an exception #733 #747
- Filter out the items that pass the precheck from the message returned when the precheck of the data migration task fails #730
- Fix the issue that the data migration task is not automatically paused and the error cannot be identified by executing the `query-status` command if an error occurs when the load unit creates a table #747
- Fix possible DM-worker panics when data migration tasks run simultaneously #710
- Fix the issue that the existing data migration task cannot be automatically restarted when the DM-worker process is restarted if the `enable-heartbeat` parameter of the task is set to `true` #739
- Fix the issue that the shard DDL conflict error may not be returned after the task is resumed #739 #742

- Fix the issue that the `replicate lag` information is displayed incorrectly for an initial period of time when the `enable-heartbeat` parameter of the data migration task is set to `true` #704
- Fix the issue that `replicate lag` cannot be calculated using the heartbeat information when `lower_case_table_names` is set to `1` in the upstream database #704
- Disable the meaningless auto-resume tasks triggered by the `unsupported collation` error during data migration #735
- Optimize some logs #660 #724 #738

16.1.3 DM 1.0.5 Release Notes

Release date: April 27, 2020

DM version: 1.0.5

DM-Ansible version: 1.0.5

16.1.3.1 Improvements

- Improve the incremental replication speed when the `UNIQUE KEY` column has the `NULL` value
- Add retry for the `Write conflict` (9007 and 8005) error returned by TiDB

16.1.3.2 Bug fixes

- Fix the issue that the `Duplicate entry` error might occur during the full data import
- Fix the issue that the migration task cannot be stopped or paused when the full data import is completed and the upstream has no written data

- Fix the issue the monitoring metrics still display data after the migration task is stopped

16.1.3.3 Detailed bug fixes and changes

- Improve the incremental replication speed when the `UNIQUE KEY` column has the `NULL` value #588 #597
- Add retry for the `Write conflict` (9007 and 8005) error returned by TiDB #632
- Fix the issue that the `Duplicate entry` error might occur during the full data import #554
- Fix the issue that the migration task cannot be stopped or paused when the full data import is completed and the upstream has no written data #622
- Fix the issue the monitoring metrics still display data after the migration task is stopped #616

- Fix the issue that the `Column count doesn't match value count` error might be returned during the sharding DDL migration #624
- Fix the issue that some metrics such as `data file size` are incorrectly displayed when the paused task of full data import is resumed #570
- Add and fix multiple monitoring metrics #590 #594

16.1.4 DM 1.0.4 Release Notes

Release date: March 13, 2020

DM version: 1.0.4

DM-Ansible version: 1.0.4

16.1.4.1 Improvements

- Add English UI for DM-portal
- Add the `--more` parameter in the `query-status` command to show complete migration status information

16.1.4.2 Bug fixes

- Fix the issue that `resume-task` might fail to resume the migration task which is interrupted by the abnormal connection to the downstream TiDB server
- Fix the issue that the online DDL operation cannot be properly migrated after a failed migration task is restarted because the online DDL meta information has been cleared after the DDL operation failure
- Fix the issue that `query-error` might cause the DM-worker to panic after `start-task` goes into error
- Fix the issue that the relay log file and `relay.meta` cannot be correctly recovered when restarting an abnormally stopped DM-worker process before `relay.meta` is successfully written

16.1.4.3 Detailed bug fixes and changes

- Add English UI for DM-portal #480
- Add the `--more` parameter in the `query-status` command to show complete migration status information #533
- Fix the issue that `resume-task` might fail to resume the migration task which is interrupted by the abnormal connection to the downstream TiDB server #436
- Fix the issue that the online DDL operation cannot be properly migrated after a failed migration task is restarted because the online DDL meta information is cleared after the DDL operation failure #465

- Fix the issue that `query-error` might cause the DM-worker to panic after `start-task` goes into error #519
- Fix the issue that the relay log file and `relay.meta` cannot be correctly recovered when restarting an abnormally stopped DM-worker process before `relay.meta` is successfully written #534
- Fix the issue that the `value out of range` error might be reported when getting `server-id` from the upstream #538
- Fix the issue that when Prometheus is not configured DM-Ansible prints the wrong error message that DM-master is not configured #438

16.1.5 DM 1.0.3 Release Notes

Release date: December 13, 2019

DM version: 1.0.3

DM-Ansible version: 1.0.3

16.1.5.1 Improvements

- Add the command mode in dmctl
- Support migrating the `ALTER DATABASE` DDL statement
- Optimize the error message output

16.1.5.2 Bug fixes

- Fix the panic-causing data race issue occurred when the full import unit pauses or exits
- Fix the issue that `stop-task` and `pause-task` might not take effect when retrying SQL operations to the downstream

16.1.5.3 Detailed bug fixes and changes

- Add the command mode in dmctl #364
- Optimize the error message output #351
- Optimize the output of the `query-status` command #357
- Optimize the privilege check for different task modes #374
- Support checking the duplicate quoted route-rules or filter-rules in task config #385
- Support migrating the `ALTER DATABASE` DDL statement #389
- Optimize the retry mechanism for anomalies #391
- Fix the panic issue caused by the data race when the import unit pauses or exits #353
- Fix the issue that `stop-task` and `pause-task` might not take effect when retrying SQL operations to the downstream #400
- Upgrade Golang to v1.13 and upgrade the version of other dependencies #362

- Filter the error that the context is canceled when a SQL statement is being executed #382
- Fix the issue that the error occurred when performing a rolling update to DM monitor using DM-ansible causes the update to fail #408

16.1.6 DM 1.0.2 Release Notes

Release date: October 30, 2019

DM version: 1.0.2

DM-Ansible version: 1.0.2

16.1.6.1 Improvements

- Generate some config items for DM-worker automatically
- Generate some config items for migration task automatically
- Simplify the output of `query-status` without arguments
- Manage DB connections directly for downstream

16.1.6.2 Bug fixes

- Fix some panic when starting up or executing SQL statements
- Fix abnormal sharding DDL migration on DDL execution timeout
- Fix starting task failure caused by the checking timeout or any inaccessible DM-worker
- Fix SQL execution retry for some error

16.1.6.3 Detailed bug fixes and changes

- Generate random `server-id` for DM-worker config automatically #337
- Generate `flavor` for DM-worker config automatically #328
- Generate `relay-binlog-name` and `relay-binlog-gtid` for DM-worker config automatically #318
- Generate the name list of tables to be dumped in task config from block & allow table lists automatically #326
- Add concurrency items (`mydumper-thread`, `loader-thread` and `syncer-thread`) for task config #314
- Simplify the output of `query-status` without arguments #340
- Fix abnormal sharding DDL migration on DDL execution timeout #338
- Fix potential DM-worker panic when restoring subtask from local meta #311
- Fix DM-worker panic when committing a DML transaction failed #313
- Fix DM-worker or DM-master panic when the listening port is being used #301
- Fix retry for error code `1105` #321, #332
- Fix retry for `Duplicate entry` and `Data too long for column` #313

- Fix task check timeout when having large amounts of tables in upstream #327
- Fix starting task failure when any DM-worker is not accessible #319
- Fix potential DM-worker startup failure in GTID mode after being recovered from corrupt relay log #339
- Fix in-memory TPS count for sync unit #294
- Manage DB connections directly for downstream #325
- Improve the error system by refining error information passed between components #320

17 TiDB Data Migration Glossary

This document lists the terms used in the logs, monitoring, configurations, and documentation of TiDB Data Migration (DM).

17.1 B

17.1.1 Binlog

In TiDB DM, binlogs refer to the binary log files generated in the TiDB database. It has the same indications as that in MySQL or MariaDB. Refer to MySQL Binary Log and MariaDB Binary Log for details.

17.1.2 Binlog event

Binlog events are information about data modification made to a MySQL or MariaDB server instance. These binlog events are stored in the binlog files. Refer to MySQL Binlog Event and MariaDB Binlog Event for details.

17.1.3 Binlog event filter

Binlog event filter is a more fine-grained filtering feature than the block and allow lists filtering rule. Refer to binlog event filter for details.

17.1.4 Binlog position

The binlog position is the offset information of a binlog event in a binlog file. Refer to MySQL `SHOW BINLOG EVENTS` and MariaDB `SHOW BINLOG EVENTS` for details.

17.1.5 Binlog replication processing unit

Binlog replication processing unit is the processing unit used in DM-worker to read upstream binlogs or local relay logs, and to replicate these logs to the downstream. Each

subtask corresponds to a binlog replication processing unit. In the current documentation, the binlog replication processing unit is also referred to as the sync processing unit.

17.1.6 Block & allow table list

Block & allow table list is the feature that filters or only migrates all operations of some databases or some tables. Refer to block & allow table lists for details. This feature is similar to MySQL Migration Filtering and MariaDB Migration Filters.

17.2 C

17.2.1 Checkpoint

A checkpoint indicates the position from which a full data import or an incremental replication task is paused and resumed, or is stopped and restarted.

- In a full import task, a checkpoint corresponds to the offset and other information of the successfully imported data in a file that is being imported. A checkpoint is updated synchronously with the data import task.
- In an incremental replication, a checkpoint corresponds to the binlog position and other information of a binlog event that is successfully parsed and migrated to the downstream. A checkpoint is updated after the DDL operation is successfully migrated or 30 seconds after the last update.

In addition, the `relay.meta` information corresponding to a relay processing unit works similarly to a checkpoint. A relay processing unit pulls the binlog event from the upstream and writes this event to the relay log, and writes the binlog position or the GTID information corresponding to this event to `relay.meta`.

17.3 D

17.3.1 Dump processing unit

The dump processing unit is the processing unit used in DM-worker to export all data from the upstream. Each subtask corresponds to a dump processing unit.

17.4 G

17.4.1 GTID

The GTID is the global transaction ID of MySQL or MariaDB. With this feature enabled, the GTID information is recorded in the binlog files. Multiple GTIDs form a GTID set. Refer to MySQL GTID Format and Storage and MariaDB Global Transaction ID for details.

17.5 H

17.5.1 Heartbeat

The heartbeat is a mechanism that calculates the delay from the time data is written in the upstream to the time data is processed by the binlog replication processing unit. Refer to migration delay monitoring for details.

17.6 L

17.6.1 Load processing unit

The load processing unit is the processing unit used in DM-worker to import the fully exported data to the downstream. Each subtask corresponds to a load processing unit. In the current documentation, the load processing unit is also referred to as the import processing unit.

17.7 M

17.7.1 Migrate/migration

The process of using the TiDB Data Migration tool to copy the **full data** of the upstream database to the downstream database.

In the case of clearly mentioning "full", not explicitly mentioning "full or incremental", and clearly mentioning "full + incremental", use migrate/migration instead of replicate/replication.

17.8 R

17.8.1 Relay log

The relay log refers to the binlog files that DM-worker pulls from the upstream MySQL or MariaDB, and stores in the local disk. The format of the relay log is the standard binlog file, which can be parsed by tools such as mysqlbinlog of a compatible version.

For more details such as the relay log's directory structure, initial migration rules, and data purge in TiDB DM, see TiDB DM relay log.

17.8.2 Relay processing unit

The relay processing unit is the processing unit used in DM-worker to pull binlog files from the upstream and write data into relay logs. Each DM-worker instance has only one relay processing unit.

17.8.3 Replicate/replication

The process of using the TiDB Data Migration tool to copy the **incremental data** of the upstream database to the downstream database.

In the case of clearly mentioning "incremental", use replicate/replication instead of migrate/migration.

17.9 S

17.9.1 Safe mode

Safe mode is the mode in which DML statements can be imported more than once when the primary key or unique index exists in the table schema. In this mode, some statements from the upstream are migrated to the downstream only after they are re-written. The `INSERT` statement is re-written as `REPLACE`; the `UPDATE` statement is re-written as `DELETE` and `REPLACE`.

This mode is enabled in any of the following situations:

- TiDB DM automatically enables the safe mode within 5 minutes immediately after the incremental replication task is started or resumed.
- The safe mode remains enabled when the `safe-mode` parameter in the task configuration file is set to `true`.
- In shard merge scenarios, the safe mode remains enabled before DDL statements are replicated in all sharded tables.

17.9.2 Shard DDL

The shard DDL is the DDL statement that is executed on the upstream sharded tables. It needs to be coordinated and migrated by TiDB DM in the process of merging the sharded tables. In the current documentation, the shard DDL is also referred to as the sharding DDL.

17.9.3 Shard DDL lock

The shard DDL lock is the lock mechanism that coordinates the migration of shard DDL. Refer to the implementation principles of merging and migrating data from sharded tables for details. In the current documentation, the shard DDL lock is also referred to as the sharding DDL lock.

17.9.4 Shard group

A shard group is all the upstream sharded tables to be merged and migrated to the same table in the downstream. Two-level shard groups are used for implementation of TiDB DM.

Refer to the implementation principles of merging and migrating sharded tables for details. In the current documentation, the shard group is also referred to as the sharding group.

17.9.5 Subtask

The subtask is a part of a data migration task that is running on each DM-worker instance. In different task configurations, a single data migration task might have one subtask or multiple subtasks.

17.9.6 Subtask status

The subtask status is the status of a data migration subtask. The current status options include `New`, `Running`, `Paused`, `Stopped`, and `Finished`. Refer to subtask status for more details about the status of a data migration task or subtask.

17.10 T

17.10.1 Table routing

The table routing feature enables DM to migrate a certain table of the upstream MySQL or MariaDB instance to the specified table in the downstream, which can be used to merge and migrate sharded tables. Refer to table routing for details.

17.10.2 Task

The data migration task, which is started after you successfully execute a `start-task` ↪ command. In different task configurations, a single migration task can run on a single DM-worker instance or on multiple DM-worker instances at the same time.

17.10.3 Task status

The task status refers to the status of a data migration task. The task status depends on the statuses of all its subtasks. Refer to subtask status for details.
