TiDB Data Migration Documentation

PingCAP Inc.

20220809

Table of Contents

1 About DM

1.1 Data Migration Overview

TiDB Data Migration (DM) is an integrated data migration task management platform, which supports the full data migration and the incremental data replication from MySQL-compatible databases (such as MySQL, MariaDB, and Aurora MySQL) into TiDB. It can help to reduce the operation cost of data migration and simplify the troubleshooting process. When using DM for data migration, you need to perform the following operations:

- Deploy a DM Cluster
- Create upstream data source and save data source access information
- Create data migration tasks to migrate data from data sources to TiDB

The data migration task includes two stages: full data migration and incremental data replication:

- Full data migration: Migrate the table structure of the corresponding table from the data source to TiDB, and then read the data stored in the data source and write it to the TiDB cluster.
- Incremental data replication: After the full data migration is completed, the corresponding table changes from the data source are read and then written to the TiDB cluster.

The following describes the features of DM.

1.1.1 Basic features

This section describes the basic data migration features provided by DM.

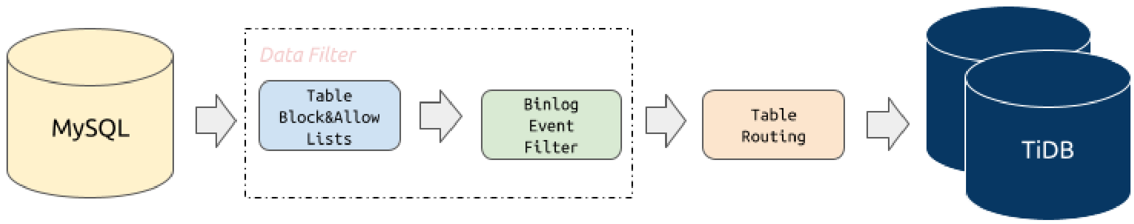

Figure 1: DM Core Features

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

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

1.1.1.2 Binlog event filtering

The binlog event filtering feature means that DM can filter certain types of SQL statements from certain tables in the source database. For example, you can filter all `INSERT` statements in the table `test.sbtest` or filter all `TRUNCATE TABLE` statements in the schema `test`.

1.1.1.3 Schema and table routing

The schema and table routing feature means that DM can migrate a certain table of the source database to the specified table in the downstream. For example, you can migrate the table structure and data from the table `test.sbtest1` in the source database to the table `test.sbtest2` in TiDB. This is also a core feature for merging and migrating sharded databases and tables.

1.1.2 Advanced features

1.1.2.1 Shard merge and migration

DM supports merging and migrating the original sharded instances and tables from the source databases into TiDB, but with some restrictions. For details, see Sharding DDL usage restrictions in the pessimistic mode and Sharding DDL usage restrictions in the optimistic mode.

1.1.2.2 Optimization for third-party online-schema-change tools in the migration process

In the MySQL ecosystem, tools such as gh-ost and pt-osc are widely used. DM provides support for these tools to avoid migrating unnecessary intermediate data. For details, see Online DDL Tools

1.1.2.3 Filter certain row changes using SQL expressions

In the phase of incremental replication, DM supports the configuration of SQL expressions to filter out certain row changes, which lets you replicate the data with a greater granularity. For more information, refer to Filter Certain Row Changes Using SQL Expressions.

1.1.3 Usage restrictions

Before using the DM tool, note the following restrictions:

- Database version requirements
  - MySQL version > 5.5
  - 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.
> - MySQL version > 5.7.1
> - MariaDB version >= 10.1.3

> **Warning:**
> Support for MySQL 8.0 is an experimental feature of TiDB Data Migration v2.0. It is **NOT** recommended that you use it in a production environment.

- DDL syntax compatibility

- 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 merge with conflicts
  - 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 DDL migration restrictions, see Sharding DDL usage restrictions in the pessimistic mode and Sharding DDL usage restrictions in the optimistic mode.

- Switch of MySQL instances for data sources

  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, see Switch DM-worker connection via virtual IP.

1.2 Basic Features

1.2.1 Key Features

This document describes the data migration features provided by TiDB Data Migration (DM) and introduces appropriate parameter configurations.

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.

1.2.1.1 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.

1.2.1.1.1 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"
```

1.2.1.1.2 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`.

1.2.1.1.3 Usage examples

This section shows the usage examples in different scenarios.

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 the following 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 is not to 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"
```

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"
```

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"
```

1.2.1.2 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.

1.2.1.2.1 Parameter configuration

```
block-allow-list:          # This configuration applies to DM versions
   ↪ higher than v2.0.0-beta.2. Use black-white-list otherwise.
  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"
```

1.2.1.2.2 Parameter explanation

- `do-dbs`: allow lists of the schemas to be migrated, similar to `replicate-do-db` in MySQL
- `ignore-dbs`: block lists of the schemas to be migrated, similar to `replicate-ignore-db` in MySQL
- `do-tables`: allow lists of the tables to be migrated, similar to `replicate-do-table` in MySQL. Both `db-name` and `tbl-name` must be specified
- `ignore-tables`: block lists of the tables to be migrated, similar to `replicate-ignore-table` in MySQL. Both `db-name` and `tbl-name` must be specified

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.

1.2.1.2.3 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 a 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` should be filtered, you only need to filter at the schema level.

1.2.1.2.4 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: # This configuration applies to DM versions higher than v2
    ↪ .0.0-beta.2. Use black-white-list otherwise.
  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`.

Table	Whether to filter	Why filter
`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.
`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	1. 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`.

1.2.1.3 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 the 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 table, the `Ignore` rule takes effect.
> - Starting from DM v2.0.2, you can configure binlog event filters in the source configuration file. For details, see Upstream Database Configuration File.

1.2.1.3.1 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
```

1.2.1.3.2 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. You can only select one or more `Events` from the following table:

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

Events	Type	Description
`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 is filtered; otherwise, the binlog is not filtered.
  - `Do`: the allow list. The binlog is 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 is 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.

1.2.1.3.3 Usage examples

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

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
```

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
```

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.

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 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 (of some version) 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
```

1.2.1.4 Online DDL tools

In the MySQL ecosystem, tools such as gh-ost and pt-osc are widely used. DM provides supports for these tools to avoid migrating unnecessary intermediate data.

1.2.1.4.1 Restrictions

- DM only supports gh-ost and pt-osc.
- When `online-ddl` is enabled, the checkpoint corresponding to incremental replication should not be in the process of online DDL execution. For example, if an upstream online DDL operation starts at `position-A` and ends at `position-B` of the binlog, the starting point of incremental replication should be earlier than `position-A` or later than `position-B`; otherwise, an error occurs. For details, refer to FAQ.

1.2.1.4.2 Parameter configuration

In v2.0.5 and later versions, you need to use the `online-ddl` configuration item in the `task` configuration file.

- If the upstream MySQL/MariaDB (at the same time) uses the gh-ost or pt-osc tool, set `online-ddl` to `true` in the task configuration file:

```
online-ddl: true
```

> **Note:**
>
> Since v2.0.5, `online-ddl-scheme` has been deprecated, so you need to use `online-ddl` instead of `online-ddl-scheme`. That means that setting `online-ddl: true` overwrites `online-ddl-scheme`, and setting `online-ddl-scheme: "pt"` or `online-ddl-scheme: "gh-ost"` is converted to `online-ddl: true`.

Before v2.0.5 (not including v2.0.5), you need to use the `online-ddl-scheme` configuration item in the `task` configuration file.

- If the upstream MySQL/MariaDB uses the gh-ost tool, set it in the task configuration file:

```
online-ddl-scheme: "gh-ost"
```

- If the upstream MySQL/MariaDB uses the pt tool, set it in the task configuration file:

```
online-ddl-scheme: "pt"
```

1.2.1.5 Shard merge

DM supports merging the DML and DDL data in the upstream MySQL/MariaDB sharded tables and migrating the merged data to the downstream TiDB tables.

1.2.1.5.1 Restrictions

Currently, the shard merge feature is supported only in limited scenarios. For details, refer to Sharding DDL usage Restrictions in the pessimistic mode and Sharding DDL usage Restrictions in the optimistic mode.

1.2.1.5.2 Parameter configuration

Set `shard-mode` to `pessimistic` in the task configuration file:

```
shard-mode: "pessimistic" # The shard merge mode. Optional modes are ""/"
    ↪ pessimistic"/"optimistic". The "" mode is used by default which means
    ↪  sharding DDL merge is disabled. If the task is a shard merge task,
    ↪ set it to the "pessimistic" mode. After getting a deep understanding
    ↪ of the principles and restrictions of the "optimistic" mode, you can
    ↪ set it to the "optimistic" mode.
```

1.2.1.5.3 Handle sharding DDL locks manually

In some abnormal scenarios, you need to handle sharding DDL Locks manually.

1.3 Advanced Features

1.3.1 Merge and Migrate Data from Sharded Tables

1.3.1.1 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 or different table schemas 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.

1.3.1.1.1 Overview

DM supports merging and migrating the data of multiple upstream sharded tables into one table in TiDB. During the migration, the DDL of each sharded table, and the DML before and after the DDL need to be coordinated. For the usage scenarios, DM supports two different modes: pessimistic mode and optimistic mode.

> **Note:**
>
> - To merge and migrate data from sharded tables, you must set `shard-↪ mode` in the task configuration file.
> - DM uses the pessimistic mode by default for the merge of the sharding support feature. (If there is no special description in the document, use the pessimistic mode by default.)

- It is not recommended to use this mode if you do not understand the principles and restrictions of the optimistic mode. Otherwise, it may cause serious consequences such as migration interruption and even data inconsistency.

The pessimistic mode

When an upstream sharded table executes a DDL statement, the migration of this sharded table will be suspended. After all other sharded tables execute the same DDL, the DDL will be executed in the downstream and the data migration task will restart. The advantage of this mode is that it can ensure that the data migrated to the downstream will not go wrong. For details, refer to shard merge in pessimistic mode. ###### The optimistic mode

DM will automatically modify the DDL executed on a sharded table into a statement compatible with other sharded tables, and then migrate to the downstream. This will not block the DML migration of any sharded tables. The advantage of this mode is that it will not block data migration when processing DDL. However, improper use will cause migration interruption or even data inconsistency. For details, refer to shard merge in optimistic mode.

Contrast

Pessimistic mode	Optimistic mode
Sharded tables that executes DDL suspend DML migration	Sharded tables that executes DDL continue DML migration
The DDL execution order and statements of each sharded table must be the same	Each sharded table only needs to keep the table schema compatible with each other

Pessimistic mode	Optimistic mode
The DDL is migrated to the downstream after the entire shard group is consistent	The DDL of each sharded table immediately affects the downstream
Wrong DDL operations can be intercepted after the detection	Wrong DDL operations will be migrated to the downstream, which may cause inconsistency between the upstream and downstream data before the detection

1.3.1.2 Merge and Migrate Data from Sharded Tables in the Pessimistic Mode

This document introduces the sharding support feature provided by Data Migration (DM) in the pessimistic mode (the default mode). 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.

1.3.1.2.1 Restrictions

DM has the following sharding DDL usage restrictions in the pessimistic mode:

- For a logical **sharding group** (composed of all sharded tables that need to be merged and migrated into one same downstream table), it is limited to use one task containing exactly the sources of sharded tables to perform the migration.
- In a logical **sharding group**, 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.

- 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 does not support `TRUNCATE TABLE`.
  - The sync unit in DM-worker automatically ignores the `TRUNCATE 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 sharding group migration task requires each DDL statement to involve operations on only one table.
- 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.

1.3.1.2.2 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:

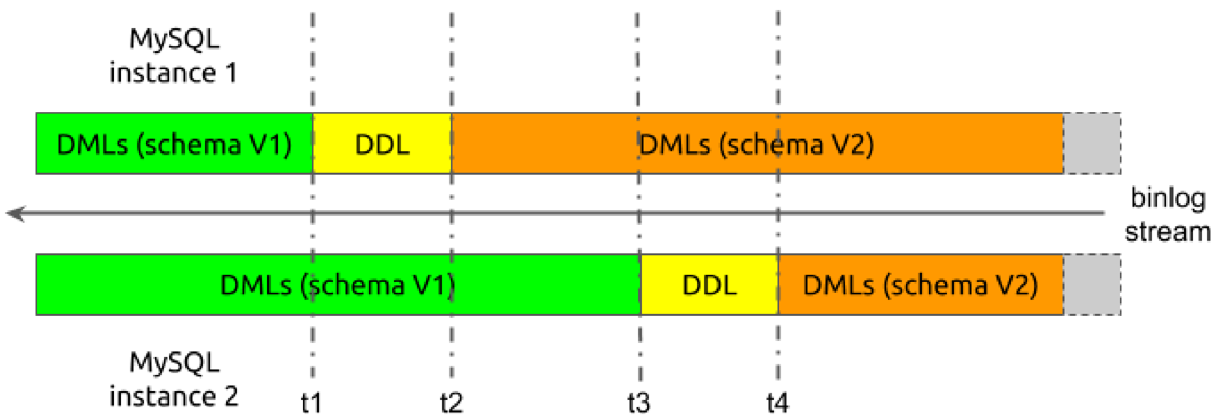

Figure 2: shard-ddl-example-1

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.

1.3.1.2.3 Principles

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

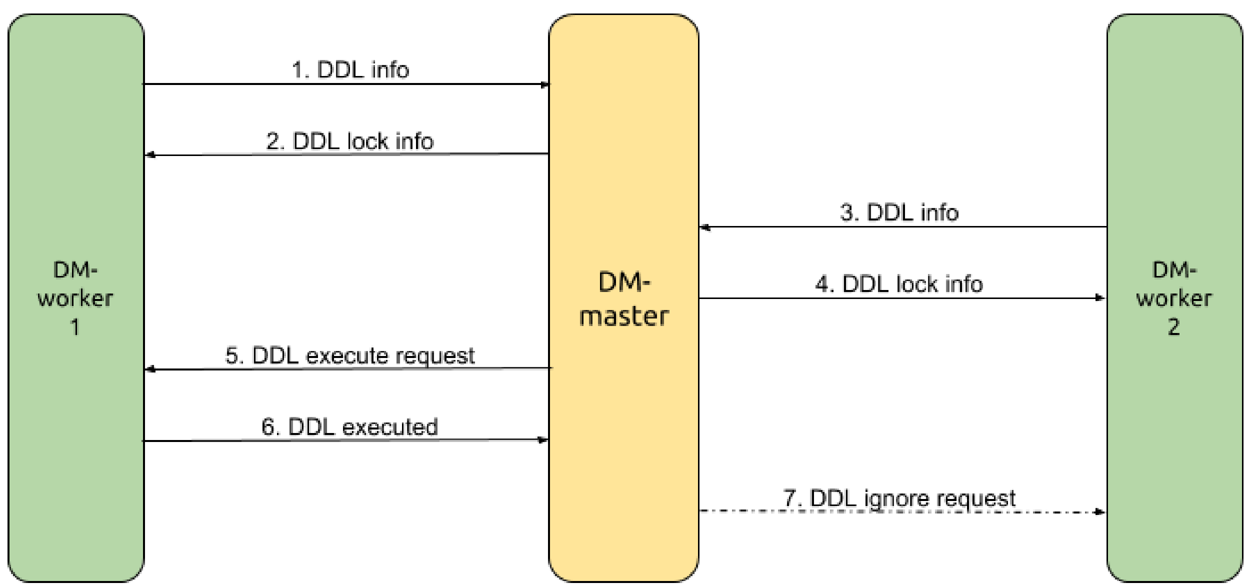

Figure 3: shard-ddl-flow

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:

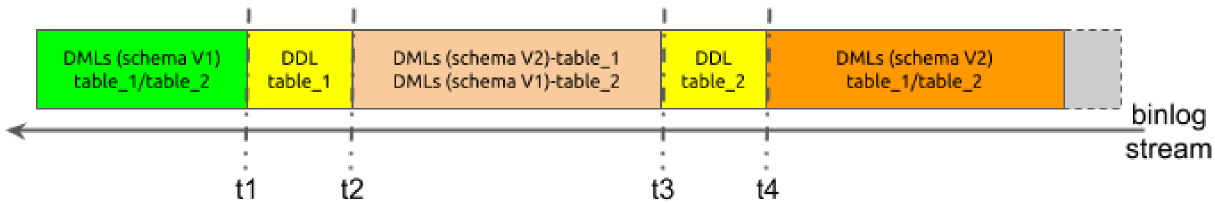

Figure 4: shard-ddl-example-2

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.

1.3.1.3 Merge and Migrate Data from Sharded Tables in Optimistic Mode

This document introduces the sharding support feature provided by Data Migration (DM) in the optimistic mode. This feature allows you to merge and migrate the data of tables with the same or different table schema(s) in the upstream MySQL or MariaDB instances into one same table in the downstream TiDB.

> **Note:**
>
> If you do not have an in-depth understanding of the optimistic mode and its restrictions, it is **NOT** recommended to use this mode. Otherwise, migration interruption or even data inconsistency might occur.

1.3.1.3.1 Background

DM supports executing DDL statements on sharded tables online, which is called sharding DDL, and uses the "pessimistic mode" by default. In this mode, when a DDL statement is executed in an upstream sharded table, data migration of this table is paused until the same DDL statement is executed in all other sharded tables. Only by then this DDL statement is executed in the downstream and data migration resumes.

The pessimistic mode guarantees that the data migrated to the downstream is always correct, but it pauses the data migration, which is bad for making A/B changes in the upstream. In some cases, users might spend a long time executing DDL statements in a single sharded table and change the schemas of other sharded tables only after a period of validation. In the pessimistic mode, these DDL statements block data migration and cause many binlog events to pile up.

Therefore, an "optimistic mode" is needed. In this mode, a DDL statement executed on a sharded table is automatically converted to a statement that is compatible with other sharded tables, and then immediately migrated to the downstream. In this way, the DDL statement does not block any sharded table from executing DML migration.

1.3.1.3.2 Configuration of the optimistic mode

To use the optimistic mode, specify the `shard-mode` item in the task configuration file as `optimistic`. For the detailed sample configuration file, see DM Advanced Task Configuration File.

1.3.1.3.3 Restrictions

It takes some risks to use the optimistic mode. Follow these rules when you use it:

- Ensure that the schema of every sharded table is consistent with each other before and after you execute a batch of DDL statements.
- If you perform an A/B test, perform the test **ONLY** on one sharded table.
- After the A/B test is finished, migrate only the most direct DDL statement(s) to the final schema. Do not re-execute every right or wrong step of the test.

  For example, if you have executed `ADD COLUMN A INT; DROP COLUMN A; ADD COLUMN ↪ A FLOAT;` in a sharded table, you only need to execute `ADD COLUMN A FLOAT` in other sharded tables. You do not need to executed all of the three DDL statements again.
- Observe the status of the DM migration when executing the DDL statement. When an error is reported, you need to determine whether this batch of DDL statements will cause data inconsistency.

Currently, the following statements are not supported in the optimistic mode:

- `ALTER TABLE table_name ADD COLUMN column_name datatype NOT NULL` (To add a `NOT NULL` column without a default value).
- `ALTER TABLE table_name ADD COLUMN column_name datetime DEFAULT NOW()` (To add a column with a varying value).
- `ALTER TABLE table_name ADD COLUMN col1 INT, DROP COLUMN col2` (Contains both `ADD COLUMN` and `DROP COLUMN` in one DDL statement).
- `ALTER TABLE table_name RENAME COLUMN column_1 TO column_2;` (To rename a column).
- `ALTER TABLE table_name RENAME INDEX index_1 TO index_2;` (To rename an index).

In addition, the following restrictions apply to both the optimistic mode and the pessimistic mode:

- In an incremental replication task, ensure that each sharded table's schema that corresponds to the binlog position at the start of the task is consistent with each other.
- The new table added to a sharding group must have a consistent table schema with that of other members. The `CREATE/RENAME TABLE` statement is forbidden when a batch of DDL statements is being executed.
- `DROP TABLE` or `DROP DATABASE` is not supported.
- `TRUNCATE TABLE` is not supported.
- Each DDL statement must involve operations on only one table.
- The DDL statement that is not supported in TiDB is also not supported in DM.
- The default value of a newly added column must not contain `current_timestamp ↪ `, `rand()`, `uuid()`; otherwise, data inconsistency between the upstream and the downstream might occur.

1.3.1.3.4 Risks

When you use the optimistic mode for a migration task, a DDL statement is migrated to the downstream immediately. If this mode is misused, data inconsistency between the upstream and the downstream might occur.

Operations that cause data inconsistency

- The schema of each sharded table is incompatible with each other. For example:
  - Two columns of the same name are added to two sharded tables respectively, but the columns are of different types.
  - Two columns of the same name are added to two sharded tables respectively, but the columns have different default values.
  - Two generated columns of the same name are added to two sharded tables respectively, but the columns are generated using different expressions.
  - Two indexes of the same name are added to two sharded tables respectively, but the keys are different.
  - Other different table schemas with the same name.
- Execute the DDL statement that can corrupt data in the sharded table and then try to roll back.

  For example, drop a column `X` and then add this column back.

Example

Merge and migrate the following three sharded tables to TiDB:

tbl00

ID	Name
1	Sarah
5	Sophia

tbl01

ID	Name
12	Paul
16	Jessica
19	Shaun

tbl02

ID	Name
23	Bob
24	Ben

tbl

ID	Name
1	Sarah
5	Sophia
12	Paul
16	Jessica
19	Shaun
23	Bob
24	Ben

Figure 5: optimistic-ddl-fail-example-1

Add a new column `Age` in `tbl01` and set the default value of the column to `0`:

```
ALTER TABLE `tbl01` ADD COLUMN `Age` INT DEFAULT 0;
```

tbl00

ID	Name
1	Sarah
5	Sophia

tbl01

ID	Name	Age
12	Paul	0
16	Jessica	0
19	Shaun	0

tbl02

ID	Name
23	Bob
24	Ben

tbl

ID	Name	Age
1	Sarah	0
5	Sophia	0
12	Paul	0
16	Jessica	0
19	Shaun	0
23	Bob	0
24	Ben	0

Figure 6: optimistic-ddl-fail-example-2

Add a new column `Age` in `tbl00` and set the default value of the column to `-1`:

```
ALTER TABLE `tbl00` ADD COLUMN `Age` INT DEFAULT -1;
```

tbl00

ID	Name	Age
1	Sarah	-1
5	Sophia	-1

tbl01

ID	Name	Age
12	Paul	0
16	Jessica	0
19	Shaun	0

tbl02

ID	Name
23	Bob
24	Ben

tbl

ID	Name	Age
1	Sarah	0
5	Sophia	0
12	Paul	0
16	Jessica	0
19	Shaun	0
23	Bob	0
24	Ben	0

Figure 7: optimistic-ddl-fail-example-3

By then, the `Age` column of `tbl00` is inconsistent because `DEFAULT 0` and `DEFAULT -1` are incompatible with each other. In this situation, DM will report the error, but you have to manually fix the data inconsistency.

1.3.1.3.5 Implementation principle

In the optimistic mode, after DM-worker receives the DDL statement from the upstream, it forwards the updated table schema to DM-master. DM-worker tracks the current schema of each sharded table, and DM-master merges these schemas into a composite schema that is compatible with DML statements of every sharded table. Then DM-master migrates the corresponding DDL statement to the downstream. DML statements are directly migrated to the downstream.

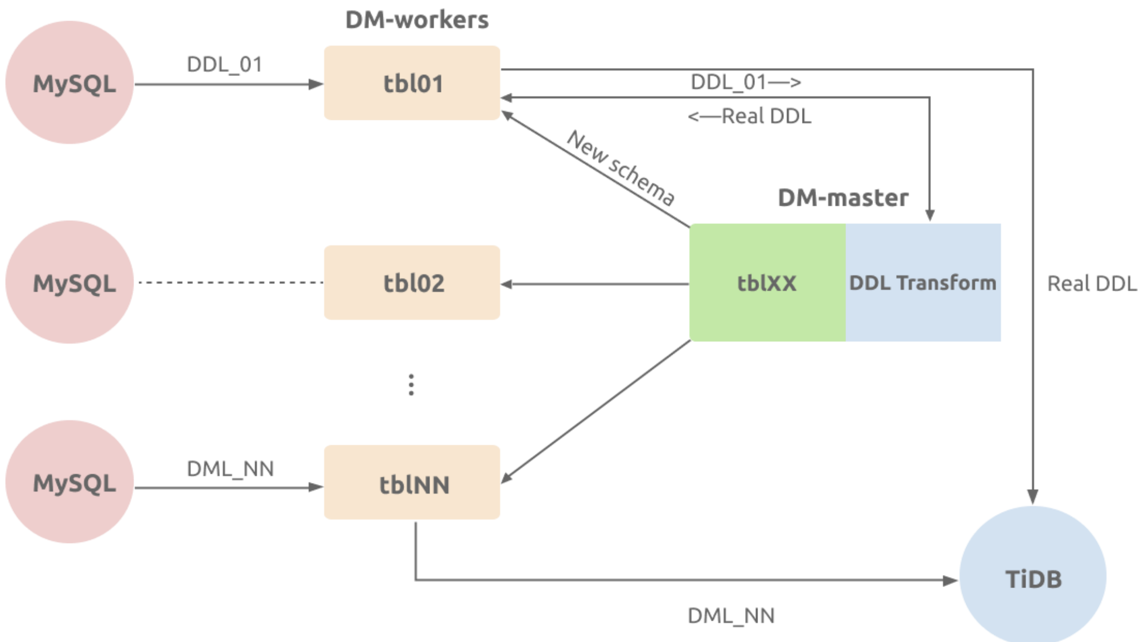

Figure 8: optimistic-ddl-flow

Examples

Assume the upstream MySQL has three sharded tables (`tbl00`, `tbl01`, and `tbl02`). Merge and migrate these sharded tables to the `tbl` table in the downstream TiDB. See the following image:

tbl00

ID	Name
1	Sarah
5	Sophia

tbl01

ID	Name
12	Paul
16	Jessica
19	Shaun

tbl02

ID	Name
23	Bob
24	Ben

tbl

ID	Name
1	Sarah
5	Sophia
12	Paul
16	Jessica
19	Shaun
23	Bob
24	Ben

Figure 9: optimistic-ddl-example-1

Add a `Level` column in the upstream:

```
ALTER TABLE `tbl00` ADD COLUMN `Level` INT;
```

tbl00

ID	Name	Level
1	Sarah	NULL
5	Sophia	NULL

tbl01

ID	Name
12	Paul
16	Jessica
19	Shaun

tbl02

ID	Name
23	Bob
24	Ben

tbl

ID	Name
1	Sarah
5	Sophia
12	Paul
16	Jessica
19	Shaun
23	Bob
24	Ben

Figure 10: optimistic-ddl-example-2

Then TiDB will receive the DML statement from `tbl00` (with the `Level` column) and the DML statement from the `tbl01` and `tbl02` tables (without the `Level` column).

tbl00

ID	Name	Level
1	Sarah	NULL
5	Sophia	NULL

tbl01

ID	Name
12	Paul
16	Jessica
19	Shaun

tbl02

ID	Name
23	Bob
24	Ben

tbl

ID	Name	Level
1	Sarah	NULL
5	Sophia	NULL
12	Paul	NULL
16	Jessica	NULL
19	Shaun	NULL
23	Bob	NULL
24	Ben	NULL

Figure 11: optimistic-ddl-example-3

The following DML statements can be migrated to the downstream without any modification:

```
UPDATE `tbl00` SET `Level` = 9 WHERE `ID` = 1;
INSERT INTO `tbl02` (`ID`, `Name`) VALUES (27, 'Tony');
```

tbl00

ID	Name	Level
1	Sarah	9
5	Sophia	NULL

tbl01

ID	Name
12	Paul
16	Jessica
19	Shaun

tbl02

ID	Name
23	Bob
24	Ben
27	Tony

tbl

ID	Name	Level
1	Sarah	9
5	Sophia	NULL
12	Paul	NULL
16	Jessica	NULL
19	Shaun	NULL
23	Bob	NULL
24	Ben	NULL
27	Tony	NULL

Figure 12: optimistic-ddl-example-4

Also add a `Level` column in `tbl01`:

```sql
ALTER TABLE `tbl01` ADD COLUMN `Level` INT;
```

tbl00

ID	Name	Level
1	Sarah	9
5	Sophia	NULL

tbl01

ID	Name	Level
12	Paul	NULL
16	Jessica	NULL
19	Shaun	NULL

tbl02

ID	Name
23	Bob
24	Ben
27	Tony

tbl

ID	Name	Level
1	Sarah	9
5	Sophia	NULL
12	Paul	NULL
16	Jessica	NULL
19	Shaun	NULL
23	Bob	NULL
24	Ben	NULL
27	Tony	NULL

Figure 13: optimistic-ddl-example-5

At this time, the downstream already have had the same `Level` column, so DM-master performs no operation after comparing the table schemas.

Drop a `Name` column in `tbl01`:

```sql
ALTER TABLE `tbl01` DROP COLUMN `Name`;
```

tbl00

ID	Name	Level
1	Sarah	9
5	Sophia	NULL

tbl01

ID	Level
12	NULL
16	NULL
19	NULL

tbl02

ID	Name
23	Bob
24	Ben
27	Tony

tbl

ID	Name	Level
1	Sarah	9
5	Sophia	NULL
12	Paul	NULL
16	Jessica	NULL
19	Shaun	NULL
23	Bob	NULL
24	Ben	NULL
27	Tony	NULL

Figure 14: optimistic-ddl-example-6

Then the downstream will receive the DML statements from `tbl00` and `tbl02` with the `Name` column, so this column is not immediately dropped.

In the same way, all DML statements can still be migrated to the downstream:

```
INSERT INTO `tbl01` (`ID`, `Level`) VALUES (15, 7);
UPDATE `tbl00` SET `Level` = 5 WHERE `ID` = 5;
```

tbl00

ID	Name	Level
1	Sarah	9
5	Sophia	5

tbl01

ID	Level
12	NULL
15	7
16	NULL
19	NULL

tbl02

ID	Name
23	Bob
24	Ben
27	Tony

tbl

ID	Name	Level
1	Sarah	9
5	Sophia	5
12	Paul	NULL
15	NULL	7
16	Jessica	NULL
19	Shaun	NULL
23	Bob	NULL
24	Ben	NULL
27	Tony	NULL

Figure 15: optimistic-ddl-example-7

Add a `Level` column in `tbl02`:

```
ALTER TABLE `tbl02` ADD COLUMN `Level` INT;
```

tbl00

ID	Name	Level
1	Sarah	9
5	Sophia	5

tbl01

ID	Level
12	NULL
15	7
16	NULL
19	NULL

tbl02

ID	Name	Level
23	Bob	NULL
24	Ben	NULL
27	Tony	NULL

tbl

ID	Name	Level
1	Sarah	9
5	Sophia	5
12	Paul	NULL
15	NULL	7
16	Jessica	NULL
19	Shaun	NULL
23	Bob	NULL
24	Ben	NULL
27	Tony	NULL

Figure 16: optimistic-ddl-example-8

By then, all sharded tables have the `Level` column.

Drop the `Name` columns in `tbl00` and `tbl02` respectively:

```
ALTER TABLE `tbl00` DROP COLUMN `Name`;
ALTER TABLE `tbl02` DROP COLUMN `Name`;
```

tbl00

ID	Level
1	9
5	5

tbl01

ID	Level
12	NULL
15	7
16	NULL
19	NULL

tbl02

ID	Level
23	NULL
24	NULL
27	NULL

tbl

ID	Name	Level
1	Sarah	9
5	Sophia	5
12	Paul	NULL
15	NULL	7
16	Jessica	NULL
19	Shaun	NULL
23	Bob	NULL
24	Ben	NULL
27	Tony	NULL

Figure 17: optimistic-ddl-example-9

By then, the `Name` columns are dropped from all sharded tables and can be safely dropped in the downstream:

```
ALTER TABLE `tbl` DROP COLUMN `Name`;
```

tbl00

ID	Level
1	9
5	5

tbl01

ID	Level
12	NULL
15	7
16	NULL
19	NULL

tbl02

ID	Level
23	NULL
24	NULL
27	NULL

tbl

ID	Level
1	9
5	5
12	NULL
15	7
16	NULL
19	NULL
23	NULL
24	NULL
27	NULL

Figure 18: optimistic-ddl-example-10

1.3.2 Migrate from Databases that Use GH-ost/PT-osc

This document introduces the `online-ddl` feature of DM when DM is used to migrate data from MySQL to TiDB and how online DDL tools perform during the data migration process.

1.3.2.1 Overview

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

Therefore, online DDL tools are often used to execute DDLs to reduce the impact on reads and writes. Common DDL tools are gh-ost and pt-osc.

Generally, these tools work by the following steps:

1. Create a new ghost table according to the table schema of the DDL real table;
2. Apply DDLs on the ghost table;
3. Replicate the data of the DDL real table to the ghost table;
4. After the data are consistent between the two tables, use the `rename` statement to replace the real table with the ghost table.

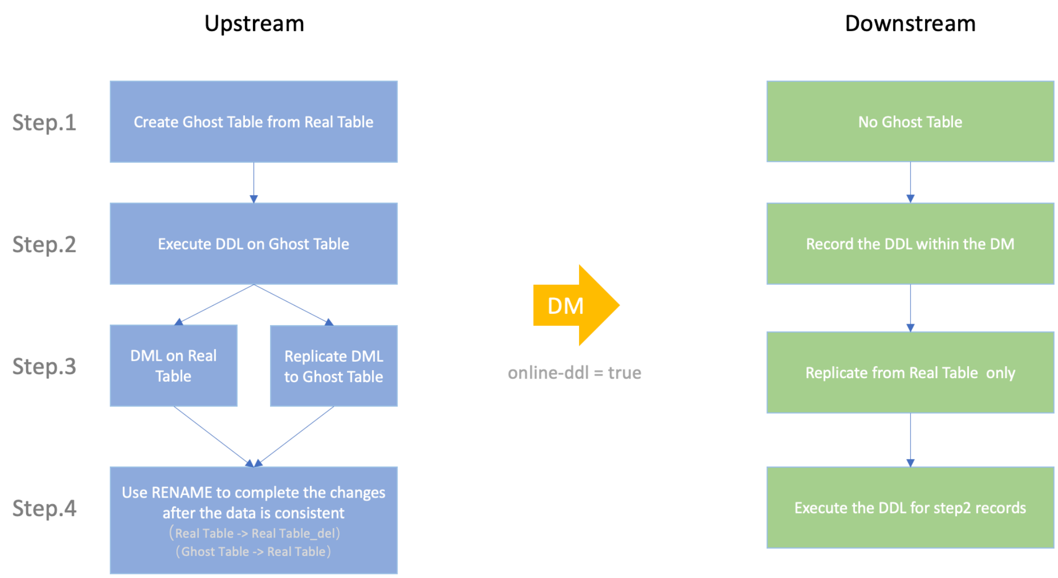

Figure 19: DM online-ddl

When you migrate data from MySQL to TiDB using DM, online DDL tools can identify the DDLs in the above step 2 and apply them downstream in step 4, which can reduce the replication workload for the ghost table.

1.3.2.2 `online-ddl` Configuration

Generally, it is recommended to enbale the `online-ddl` configuration and you can see the following effects:

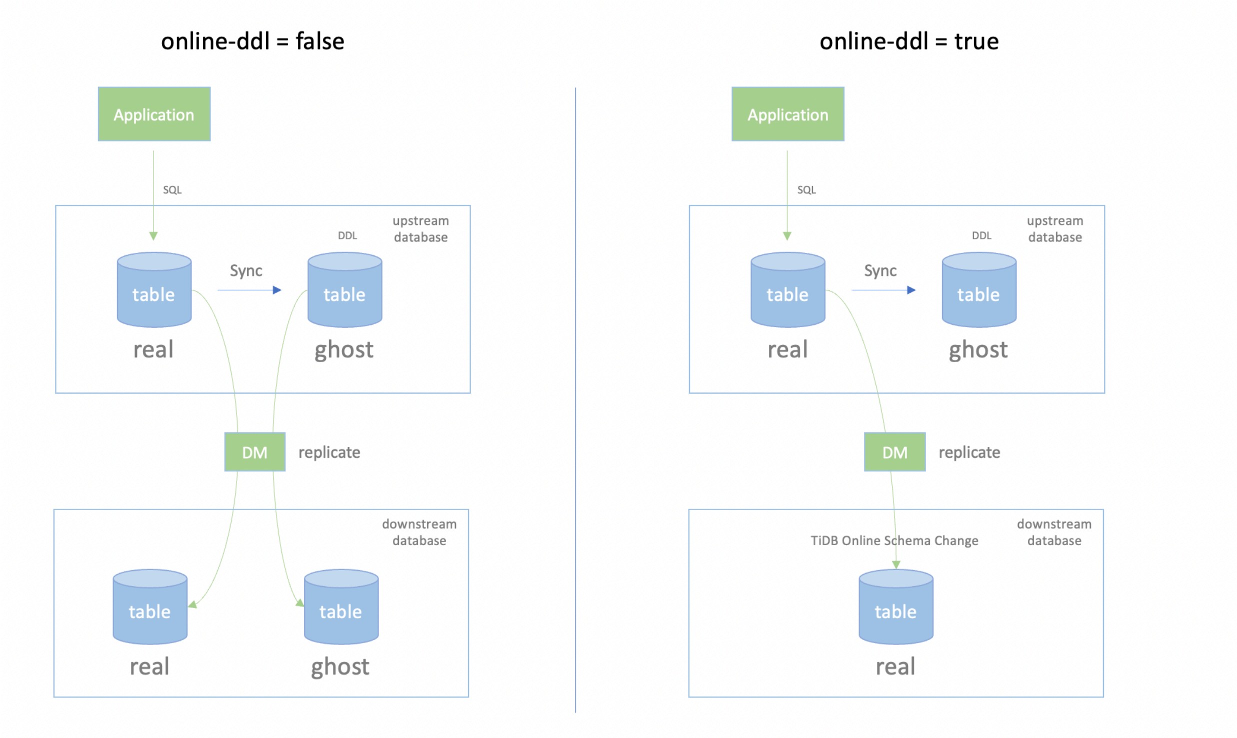

Figure 20: DM online-ddl

- The downstream TiDB does not need to create and replicate the ghost table, saving the storage space and network transmission overhead;
- When you merge and migrate data from sharded tables, the RENAME operation is ignored for each sharded ghost tables to ensure the correctness of the replication;
- Currently, one limitation for DM is that DMLs in this task are blocked until DDL operation is finished when you apply DDL operation to the downstream TiDB. This limitation will be removed later.

Note:

If you need to disable the `online-ddl` configuration, pay attention to the following effects:

- The downstream TiDB replicates the behaviors of online DDL tools like gh-ost/pt-osc;
- You need to manually add various temporary tables and ghost tables generated by the online DDL tools to the task configuration white list;
- You cannot merge and migrate data from sharded tables.

1.3.2.3 Configuration

In the task configuration file, `online-ddl` 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`.
shard-mode: "pessimistic"   # The shard merge mode. Optional modes are ""/"
   ↪ pessimistic"/"optimistic". The "" mode is used by default which means
   ↪  sharding DDL merge is disabled. If the task is a shard merge task,
   ↪ set it to the "pessimistic" mode. After understanding the principles
   ↪ and restrictions of the "optimistic" mode, you can set it to the "
   ↪ optimistic" mode.
meta-schema: "dm_meta"      # The downstream database that stores the `meta`
   ↪  information.
online-ddl: true            # Supports automatic processing of "gh-ost" and
   ↪ "pt" for the upstream database.
online-ddl-scheme: "gh-ost" # `online-ddl-scheme` will be deprecated in the
   ↪ future, so it is recommended to use `online-ddl`.
target-database:            # Configuration of the downstream database
   ↪ instance.
  host: "192.168.0.1"
  port: 4000
  user: "root"
  password: ""              # It is recommended to use password encrypted
     ↪ with dmctl if the password is not empty.
```

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

When you merge and migrate data from sharded tables, you need to coordinate the DDL of each sharded table, and the DML before and after the DDL. DM supports two different modes: pessimistic mode and optimistic mode. For the differences and scenarios between the two modes, refer to Merge and Migrate Data from Sharded Tables.

1.3.2.4 Working details for DM with online DDL tools

This section describes the working details for DM with the online DDL tools gh-ost and pt-osc when implementing online-schema-change.

1.3.2.4.1 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 migration, 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.

```sql
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:

```sql
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 migration is completed, both the origin table and `_gho` table are renamed, and the online DDL operation is completed:

```sql
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.

```sql
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

```sql
alter table test._test4_gho add column cl1 varchar(20) not null;
-- Replaced with:
alter table test.test4 add column cl1 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.

1.3.2.5 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 migration, 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 migration:

```
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 migration 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

```sql
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:

```sql
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.

1.3.3 Filter Certain Row Changes Using SQL Expressions

1.3.3.1 Overview

In the process of data migration, DM provides the Binlog Event Filter feature to filter certain types of binlog events. For example, for archiving or auditing purposes, `DELETE` event might be filtered when data is migrated to the downstream. However, Binlog Event Filter cannot judge with a greater granularity whether the `DELETE` event of a certain row should be filtered.

To solve the above issue, DM supports filtering certain row changes using SQL expressions. The binlog in the `ROW` format supported by DM has the values of all columns in binlog events. You can configure SQL expressions according to these values. If the SQL expressions evaluate a row change as `TRUE`, DM will not migrate the row change downstream.

> **Note:**
>
> This feature only takes effect in the phase of incremental replication, not in the phase of full migration.

1.3.3.2 Configuration example

Similar to Binlog Event Filter, you also need to configure the expression-filter feature in the configuration file of the data migration task, as shown below. For complete configuration and its descriptions, refer to DM Advanced Task Configuration File:

```yaml
name: test
task-mode: all

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

mysql-instances:
  - source-id: "mysql-replica-01"
    expression-filters: ["even_c"]

expression-filter:
  even_c:
    schema: "expr_filter"
    table: "tbl"
    insert-value-expr: "c % 2 = 0"
```

The above example configures `even_c` rule, and allows the data source whose ID is `mysql-replica-01` to refer this rule. The meaning of `even_c` is:

For the `tbl` table in the `expr_filter` shema, when the value of the inserted `c` is even (`c % 2 = 0`), the inserted statement will not be migrated downstream.

The usage result of this rule is shown below.

Insert the following data in the upstream data source:

```sql
INSERT INTO tbl(id, c) VALUES (1, 1), (2, 2), (3, 3), (4, 4);
```

Then query the `tbl` table downstream and you can find that only rows with an odd value of `c` are migrated downstream:

```
MySQL [test]> select * from tbl;
+------+------+
| id   | c    |
+------+------+
|    1 |    1 |
|    3 |    3 |
+------+------+
2 rows in set (0.001 sec)
```

1.3.3.3 Configuration parameters and rule descriptions

- `schema`: The name of the upstream database to be matched. Wildcard match or regular match is not supported.
- `table`: The name of the upstream table to be matched. Wildcard match or regular match is not supported.
- `insert-value-expr`: Specifies an expression which takes effect on the value of binlog event (WRITE_ROWS_EVENT) of INSERT type. Do not use it with `update-old-value-expr`, `update-new-value-expr`, or `delete-value-expr` in the same configuration item.
- `update-old-value-expr:` Specifies an expression which takes effect on the old value of binlog event (UPDATE_ROWS_EVENT) of UPDATE type. Do not use it with `insert-value-expr` or `delete-value-expr` in the same configuration item.
- `update-new-value-expr`: Specifies an expression which takes effect on the new value of binlog event (UPDATE_ROWS_EVENT) of UPDATE type. Do not use it with `insert-value-expr` or `delete-value-expr` in the same configuration item.
- `delete-value-expr:` Specifies an expression which takes effect on the value of binlog event (DELETE_ROWS_EVENT) of DELETE type. Do not use it with`insert-value-expr`, `update-old-value-expr`, or `update-new-value-expr` in the same configuration item.

Note:

You can configure `update-old-value-expr` and `update-new-value-expr` at the same time.

- When you configure `update-old-value-expr` and `update-new-value-expr` at the same time, the row changes where updated old value meets the rule of `update-old-value-expr` **and** the updated new value meets the rule of `update-new-value-expr` will be filtered out.

- When you only configure one parameter, the statement you configure will decide whether to filter **the whole row changes**, which means the delete event of an old value and the insert event of a new value will be filtered out as a whole.

SQL expressions can involve one or more columns. You can also use the SQL functions TiDB supports, such as `c % 2 = 0`, `a*a + b*b = c*c`, and `ts > NOW()`.

The timezone of TIMESTAMP is UTC by default. You can use `c_timestamp = '2021-01-01 12:34:56.5678+08:00'` to specify the timezone explicitly.

You can define multiple filter rules under the configuration item `expression-filter`. By refering the rules you need in the configuration item of `expression-filters` in the upstream data source, the rules can take effect. When multiple rules take effect, matching **any** of the rules causes a row change to be filtered.

> **Note:**
>
> Setting too many expression filters for a table increases the computing overhead of DM, which might impede data migration.

1.4 Data Migration Architecture

This document introduces the architecture of Data Migration (DM).

DM consists of three components: DM-master, DM-worker, and dmctl.

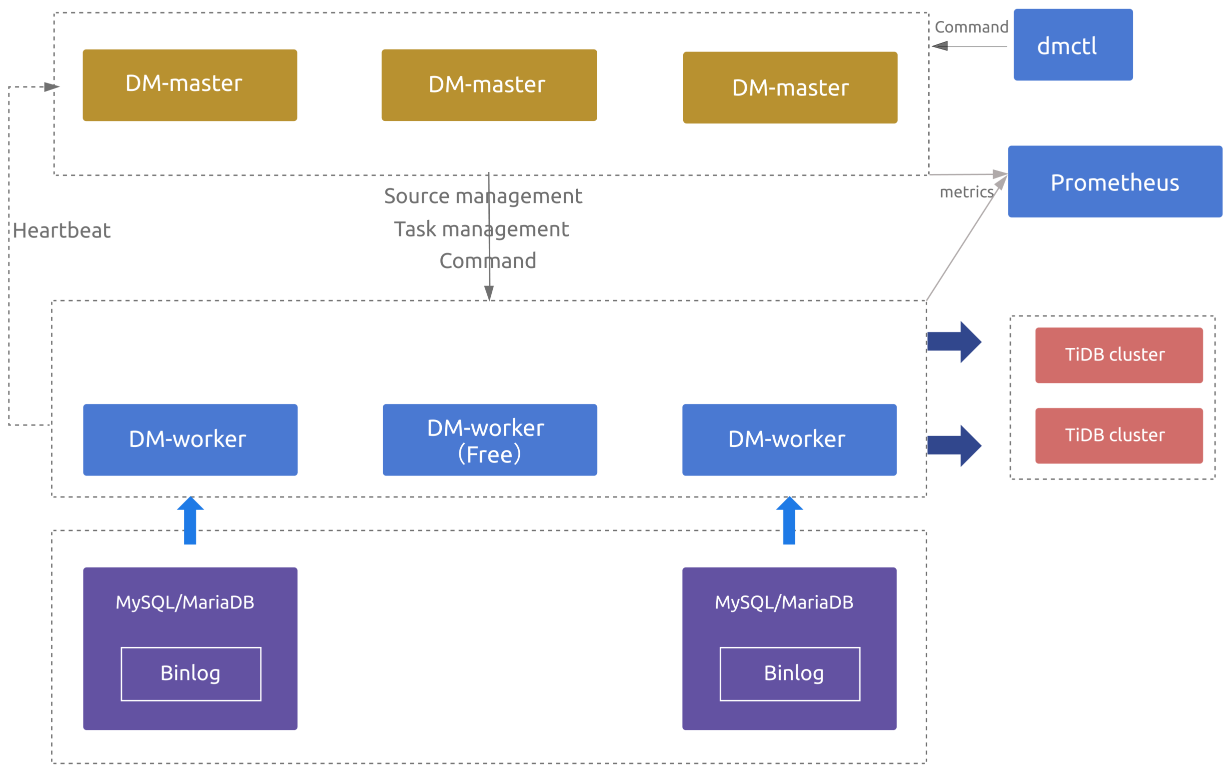

Figure 21: Data Migration architecture

1.4.1 Architecture components

1.4.1.1 DM-master

DM-master manages and schedules the operations 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

1.4.1.2 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

For more details of DM-worker, see DM-worker Introduction.

1.4.1.3 dmctl

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

- Creating, updating, or dropping data migration tasks
- Checking the state of data migration tasks
- Handling errors of data migration tasks
- Verifying the configuration correctness of data migration tasks

1.4.2 Architecture features

1.4.2.1 High availability

When you deploy multiple DM-master nodes, all DM-master nodes use the embedded etcd to form a cluster. The DM-master cluster is used to store metadata such as cluster node information and task configuration. The leader node elected through etcd is used to provide services such as cluster management and data migration task management. Therefore, if the number of available DM-master nodes exceeds half of the deployed nodes, the DM cluster can normally provide services.

When the number of deployed DM-worker nodes exceeds the number of upstream MySQL/MariaDB nodes, the extra DM-worker nodes are idle by default. If a DM-worker node goes offline or is isolated from the DM-master leader, DM-master automatically schedules data migration tasks of the original DM-worker node to other idle DM-worker nodes. (If a DM-worker node is isolated, it automatically stops the data migration tasks on it); if there are no available idle DM-worker nodes, the data migration tasks of the original DM-worker are temporarily hung until one DM-worker node becomes idle, and then the tasks are automatically resumed.

> **Note:**
>
> When the data migration task is in the process of full export or import, the migration task does not support high availability. Here are the main reasons:
>
> - For the full export, MySQL does not support exporting from a specific snapshot point yet. This means that after the data migration task is rescheduled or restarted, the export cannot resume from the previous interruption point.

- For the full import, DM-worker does not support reading exported full data across the nodes yet. This means that after the data migration task is scheduled to a new DM-worker node, you cannot read the exported full data on the original DM-worker node before the scheduling happens.

1.5 DM 2.0-GA Benchmark Report

This benchmark report describes the test purpose, environment, scenario, and results for DM 2.0-GA.

1.5.1 Test purpose

The purpose of this test is to evaluate the performance of DM full import and incremental replication and to conclude recommended configurations for DM migration tasks based on the test results.

1.5.2 Test environment

1.5.2.1 Machine information

System information:

Machine IP	Operating System	Kernel version	File system type
172.16.5.32	CentOS Linux release 7.8.2003	3.10.0-957.el7.x86_64	ext4
172.16.5.33	CentOS Linux release 7.8.2003	3.10.0-957.el7.x86_64	ext4
172.16.5.34	CentOS Linux release 7.8.2003	3.10.0-957.el7.x86_64	ext4
172.16.5.35	CentOS Linux release 7.8.2003	3.10.0-957.el7.x86_64	ext4
172.16.5.36	CentOS Linux release 7.8.2003	3.10.0-957.el7.x86_64	ext4
172.16.5.37	CentOS Linux release 7.8.2003	3.10.0-957.el7.x86_64	ext4

Hardware information:

Type	Specification
CPU	Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, 40 Cores
Memory	128G, 8 * 16GB DIMM DDR4 2133 MHz
Disk	Intel SSD DC P4800X 375G NVMe * 2
Network card	10 Gigabit Ethernet

Others:

- Network rtt between servers: rtt min/avg/max/mdev = 0.074/0.116/0.158/0.042 ms

1.5.2.2 Cluster topology

Machine IP	Deployed instance
172.16.5.32	PD1, DM-worker1, DM-master
172.16.5.33	PD2, MySQL1
172.16.5.34	PD3, TiDB
172.16.5.35	TiKV1(nvme0n1), TiKV2(nvme1n1)
172.16.5.36	TiKV3(nvme0n1), TiKV4(nvme1n1)
172.16.5.37	TiKV5(nvme0n1), TiKV6(nvme1n1)

1.5.2.3 Version information

- MySQL version: 5.7.31-log
- TiDB version: v4.0.7
- DM version: v2.0.0
- Sysbench version: 1.0.17

1.5.3 Test scenario

You can use a simple data migration flow, that is, MySQL1 (172.16.5.33) -> DM-worker(172.16.5.32) -> TiDB (172.16.5.34), to do the test. For detailed test scenario description, see performance test.

1.5.3.1 Full import benchmark case

For detailed full import test method, see Full Import Benchmark Case.

1.5.3.1.1 Full import benchmark results

To enable multi-thread concurrent data export via Dumpling, you can configure the `threads` parameter in the `mydumpers` configuration item. This speeds up data export.

Item	Data size (GB)	Threads	Rows	Statement-size	Time (s)	Dump speed (MB/s)
dump data	38.1	32	320000	1000000	106.73	359.43

Item	Data size (GB)	Pool size	Statement per TXN	Max latency of TXN execution (s)	Time (s)	Import speed (MB/s)
load data	38.1	32	4878	20.95	1580.54	24.11

Item	Data size (GB)	Pool size	Statement per TXN	Max latency of TXN execution (s)	Time (s)	Import speed (MB/s)

1.5.3.1.2 Benchmark results with different pool sizes in load unit

In this test, the full amount of data imported using `sysbench` is 3.78 GB. The following is detailed information of the test data:

load unit pool size	Max latency of TXN execution (s)	Import time (s)	Import Speed (MB/s)	TiDB 99 duration (s)
2	0.35	438	8.63	0.32
4	0.65	305	12.30	0.55
8	1.82	231	16.36	2.26
16	3.46	228	16.57	3.04
32	5.92	208	18.17	6.56
64	8.59	221	17.10	9.62

1.5.3.1.3 Benchmark results with different row count per statement

In this test, the full amount of imported data is 3.78 GB and the `pool-size` of load unit is set to 32. The statement count is controlled by `statement-size`, `rows`, or `extra-args` parameters in the `mydumpers` configuration item.

Row count per statement	mydumpers extra-args	Max latency of TXN execution (s)	Import time (s)	Import speed (MB/s)	TiDB 99 duration (s)
7506	-s 1500000 -r 320000	8.74	218	17.3	10.49
5006	-s 1000000 -r 320000	5.92	208	18.1	6.56
2506	-s 500000 -r 320000	3.07	222	17.0	2.32
1256	-s 250000 -r 320000	2.01	230	16.4	1.87
629	-s 125000 -r 320000	0.98	241	15.6	0.94
315	-s 62500 -r 320000	0.51	245	15.4	0.45

1.5.3.2 Incremental replication benchmark case

For detailed incremental replication test method, see Incremental Replication Benchmark Case.

1.5.3.2.1 Incremental replication benchmark result

In this test, the `worker-count` of sync unit is set to 32 and `batch` is set to 100.

Items	QPS	TPS	95% latency
MySQL	38.65k	38.65k	1.10ms
DM binlog replication unit	21.33k (The number of binlog events received per unit of time, not including skipped events)	-	66.75ms (txn execution time)
TiDB	21.90k (Begin/Commit 2.32k Insert 21.35k)	3.52k	95%: 5.2ms 99%: 8.3ms

1.5.3.2.2 Benchmark results with different sync unit concurrency

sync unit worker-count	DM QPS	Max DM execution latency (ms)	TiDB QPS	TiDB 99 duration (ms)
4	11.83k	56	12.1k	4
8	18.34k	58	18.9k	5
16	20.85k	60	21.6k	6
32	21.33k	66	21.9k	8
64	21.52k	68	22.1k	10
1024	20.45k	85	50.5k	52

1.5.3.2.3 Benchmark results with different SQL distribution

Sysbench type	DM QPS	Max DM execution latency (ms)	TiDB QPS	TiDB 99 duration (ms)
insert_only	21.33k	66	21.9k	8
write_only	10.2k	87	11.2k	8

1.5.4 Recommended parameter configuration

1.5.4.1 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.

1.5.4.2 load unit

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

1.5.4.3 sync unit

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

2 Quick Start

2.1 Quick Start Guide for TiDB Data Migration

This document describes how to migrate data from MySQL to TiDB using TiDB Data Migration (DM).

If you need to deploy DM in the production environment, refer to the following documents:

- Deploy a DM cluster Using TiUP
- Create a Data Source
- Create a Data Migration Task

2.1.1 Sample scenario

Suppose you deploy DM-master and DM-worker instances in an on-premise environment, and migrate data from an upstream MySQL instance to a downstream TiDB instance.

The detailed information of each instance is as follows:

Instance	Server Address	Port
DM-master	127.0.0.1	8261, 8291 (Internal port)
DM-worker	127.0.0.1	8262
MySQL-3306	127.0.0.1	3306
TiDB	127.0.0.1	4000

2.1.2 Deploy DM using the binary package

2.1.2.1 Download DM binary package

Download DM v2.0 binary package or compile the package manually.

2.1.2.1.1 Method 1: Download the latest version of binary package

```
wget http://download.pingcap.org/dm-nightly-linux-amd64.tar.gz
tar -xzvf dm-nightly-linux-amd64.tar.gz
```

```
cd dm-nightly-linux-amd64
```

2.1.2.1.2 Method 2: Compile the latest version of binary package

```
git clone https://github.com/pingcap/dm.git
cd dm
make
```

2.1.2.2 Deploy DM-master

Execute the following command to start the DM-master:

```
nohup bin/dm-master --master-addr='127.0.0.1:8261' --log-file=/tmp/dm-
    ↪ master.log --name="master1" >> /tmp/dm-master.log 2>&1 &
```

2.1.2.3 Deploy DM-worker

Execute the following command to start the DM-worker:

```
nohup bin/dm-worker --worker-addr='127.0.0.1:8262' --log-file=/tmp/dm-
    ↪ worker.log --join='127.0.0.1:8261' --name="worker1" >> /tmp/dm-worker
    ↪ .log 2>&1 &
```

2.1.2.4 Check deployment status

To check whether the DM cluster has been deployed successfully, execute the following command:

```
bin/dmctl --master-addr=127.0.0.1:8261 list-member
```

A normal DM cluster returns the following information:


```
{
    "result": true,
    "msg": "",
    "members": [
        {
            "leader": {
                "msg": "",
                "name": "master1",
                "addr": "127.0.0.1:8261"
            }
        },
        {
            "master": {
```

```
                "msg": "",
                "masters": [
                    {
                        "name": "master1",
                        "memberID": "11007177379717700053",
                        "alive": true,
                        "peerURLs": [
                            "http://127.0.0.1:8291"
                        ],
                        "clientURLs": [
                            "http://127.0.0.1:8261"
                        ]
                    }
                ]
            }
        },
        {
            "worker": {
                "msg": "",
                "workers": [
                    {
                        "name": "worker1",
                        "addr": "127.0.0.1:8262",
                        "stage": "free",
                        "source": ""
                    }
                ]
            }
        }
    ]
}
```

2.1.3 Migrate data from MySQL to TiDB

2.1.3.1 Prepare sample data

Before using DM, insert the following sample data to `MySQL-3306`:

```
drop database if exists `testdm`;
create database `testdm`;
use `testdm`;
create table t1 (id bigint, uid int, name varchar(80), info varchar(100),
    ↪ primary key (`id`), unique key(`uid`)) DEFAULT CHARSET=utf8mb4;
create table t2 (id bigint, uid int, name varchar(80), info varchar(100),
    ↪ primary key (`id`), unique key(`uid`)) DEFAULT CHARSET=utf8mb4;
```

```
insert into t1 (id, uid, name) values (1, 10001, 'Gabriel García Márquez'),
    ↪  (2, 10002, 'Cien años de soledad');
insert into t2 (id, uid, name) values (3, 20001, 'José Arcadio Buendía'),
    ↪ (4, 20002, 'Úrsula Iguarán'), (5, 20003, 'José Arcadio');
```

2.1.3.2 Load data source configurations

Before running a data migration task, you need to first load the configuration file of the corresponding data source (that is, `MySQL-3306` in the example) to DM.

2.1.3.2.1 Encrypt the data source password

Note:

- You can skip this step if the data source does not have a password.
- You can use the plaintext password to configure the data source information in DM v2.0 and later versions.

For safety reasons, it is recommended to configure and use encrypted passwords for data sources. Suppose the password is "123456":

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

```
fCxfQ9XKCezSzuCD0Wf5dUD+LsKegSg=
```

Save this encrypted value, and use it for creating a MySQL data source in the following steps.

2.1.3.2.2 Edit the source configuration file of the MySQL instance

Write the following configurations to `conf/source1.yaml`.

```
## MySQL1 Configuration.
source-id: "mysql-replica-01"
from:
  host: "127.0.0.1"
  user: "root"
  password: "fCxfQ9XKCezSzuCD0Wf5dUD+LsKegSg="
  port: 3306
```

2.1.3.2.3 Load data source configuration file

To load the data source configuration file of MySQL to the DM cluster using dmctl, run the following command in the terminal:

```
./bin/dmctl --master-addr=127.0.0.1:8261 operate-source create conf/source1
    ↪ .yaml
```

The following is an example of the returned results:

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

Now you successfully add the data source `MySQL-3306` to the DM cluster.

2.1.3.3 Create a data migration task

After inserting the sample data into `MySQL-3306`, take the following steps to migrate the tables `testdm.t1` and `testdm.t2` to the downstream TiDB instance:

1. Create a task configuration file `testdm-task.yaml`, and add the following configurations to the file.

```
---
name: testdm
task-mode: all
target-database:
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: "" # If the password is not null, it is recommended to use
      ↪ password encrypted with dmctl.
mysql-instances:
  - source-id: "mysql-replica-01"
    block-allow-list: "ba-rule1"
block-allow-list:
  ba-rule1:
    do-dbs: ["testdm"]
```

2. Load the task configuration file using dmctl:

```
./bin/dmctl --master-addr 127.0.0.1:8261 start-task testdm-task.yaml
```

The following is an example of the returned results:

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

Now you successfully create a data migration task that migrates data from `MySQL-3306` to the downstream TiDB instance.

2.1.3.4 Check status of the data migration task

After the data migration task is created, you can use `dmtcl query-status` to check the status of the task. See the following example:

```
./bin/dmctl --master-addr 127.0.0.1:8261 query-status
```

The following is an example of the returned results:

```
{
    "result": true,
    "msg": "",
    "tasks": [
        {
            "taskName": "testdm",
            "taskStatus": "Running",
            "sources": [
                "mysql-replica-01"
            ]
        }
    ]
}
```

2.2 Deploy a DM Cluster Using TiUP

TiUP is a cluster operation and maintenance tool introduced in TiDB 4.0. TiUP provides TiUP DM, a cluster management component written in Golang. By using TiUP DM, you can easily perform daily TiDB Data Migration (DM) operations, including deploying, starting, stopping, destroying, scaling, and upgrading a DM cluster, and manage DM cluster parameters.

TiUP supports deploying DM v2.0 or later DM versions. This document introduces how to deploy DM clusters of different topologies.

> **Note:**
>
> If your target machine's operating system supports SELinux, make sure that SELinux is **disabled**.

2.2.1 Prerequisites

When DM performs a full data replication task, the DM-worker is bound with only one upstream database. The DM-worker first exports the full amount of data locally, and then imports the data into the downstream database. Therefore, the worker's host needs sufficient storage space (The storage path is specified later when you create the task).

In addition, you need to meet the hardware and software requirements when deploying a DM cluster.

2.2.2 Step 1: Install TiUP on the control machine

Log in to the control machine using a regular user account (take the `tidb` user as an example). All the following TiUP installation and cluster management operations can be performed by the `tidb` user.

1. Install TiUP by executing the following command:

   ```
   curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
       ↪ install.sh | sh
   ```

   After the installing, `~/.bashrc` has been modified to add TiUP to PATH, so you need to open a new terminal or redeclare the global environment variables `source ~/.bashrc` to use it.

2. Install the TiUP DM component:

   ```
   tiup install dm dmctl
   ```

2.2.3 Step 2: Edit the initialization configuration file

According to the intended cluster topology, you need to manually create and edit the cluster initialization configuration file.

You need to create a YAML configuration file (named `topology.yaml` for example) according to the configuration file template. For other scenarios, edit the configuration accordingly.

You can use the command `tiup dm template > topology.yaml` to generate a configuration file template quickly.

The configuration of deploying three DM-masters, three DM-workers, and one monitoring component instance is as follows:

```
## The global variables apply to all other components in the configuration.
   ↪ If one specific value is missing in the component instance, the
   ↪ corresponding global variable serves as the default value.
global:
  user: "tidb"
  ssh_port: 22
  deploy_dir: "/dm-deploy"
  data_dir: "/dm-data"

server_configs:
  master:
    log-level: info
    # rpc-timeout: "30s"
    # rpc-rate-limit: 10.0
    # rpc-rate-burst: 40
  worker:
    log-level: info

master_servers:
  - host: 10.0.1.11
    name: master1
    ssh_port: 22
    port: 8261
    # peer_port: 8291
    # deploy_dir: "/dm-deploy/dm-master-8261"
    # data_dir: "/dm-data/dm-master-8261"
    # log_dir: "/dm-deploy/dm-master-8261/log"
    # numa_node: "0,1"
    # The following configs are used to overwrite the `server_configs.master
       ↪ ` values.
    config:
      log-level: info
```

```
    # rpc-timeout: "30s"
    # rpc-rate-limit: 10.0
    # rpc-rate-burst: 40
  - host: 10.0.1.18
    name: master2
    ssh_port: 22
    port: 8261
  - host: 10.0.1.19
    name: master3
    ssh_port: 22
    port: 8261
## If you do not need to ensure high availability of the DM cluster, deploy
   ↪ only one DM-master node, and the number of deployed DM-worker nodes
   ↪ must be no less than the number of upstream MySQL/MariaDB instances
   ↪ to be migrated.
## To ensure high availability of the DM cluster, it is recommended to
   ↪ deploy three DM-master nodes, and the number of deployed DM-worker
   ↪ nodes must exceed the number of upstream MySQL/MariaDB instances to
   ↪ be migrated (for example, the number of DM-worker nodes is two more
   ↪ than the number of upstream instances).
worker_servers:
  - host: 10.0.1.12
    ssh_port: 22
    port: 8262
    # deploy_dir: "/dm-deploy/dm-worker-8262"
    # log_dir: "/dm-deploy/dm-worker-8262/log"
    # numa_node: "0,1"
    # The following configs are used to overwrite the `server_configs.worker
       ↪ ` values.
    config:
      log-level: info
  - host: 10.0.1.19
    ssh_port: 22
    port: 8262

monitoring_servers:
  - host: 10.0.1.13
    ssh_port: 22
    port: 9090
    # deploy_dir: "/tidb-deploy/prometheus-8249"
    # data_dir: "/tidb-data/prometheus-8249"
    # log_dir: "/tidb-deploy/prometheus-8249/log"

grafana_servers:
  - host: 10.0.1.14
```

```
    port: 3000
    # deploy_dir: /tidb-deploy/grafana-3000

alertmanager_servers:
  - host: 10.0.1.15
    ssh_port: 22
    web_port: 9093
    # cluster_port: 9094
    # deploy_dir: "/tidb-deploy/alertmanager-9093"
    # data_dir: "/tidb-data/alertmanager-9093"
    # log_dir: "/tidb-deploy/alertmanager-9093/log"
```

> **Note:**
>
> - It is not recommended to run too many DM-workers on one host. Each DM-worker should be allocated at least 2 core CPU and 4 GiB memory.
> - Make sure that the ports among the following components are interconnected:
>   - The `peer_port` (`8291` by default) among the DM-master nodes are interconnected.
>   - Each DM-master node can connect to the `port` of all DM-worker nodes (`8262` by default).
>   - Each DM-worker node can connect to the `port` of all DM-master nodes (`8261` by default).
>   - The TiUP nodes can connect to the `port` of all DM-master nodes (`8261` by default).
>   - The TiUP nodes can connect to the `port` of all DM-worker nodes (`8262` by default).

For more `master_servers.host.config` parameter description, refer to master parameter. For more `worker_servers.host.config` parameter description, refer to worker parameter.

2.2.4 Step 3: Execute the deployment command

> **Note:**
>
> You can use secret keys or interactive passwords for security authentication when you deploy TiDB using TiUP:

- If you use secret keys, you can specify the path of the keys through `-i` or `--identity_file`;
- If you use passwords, add the `-p` flag to enter the password interaction window;
- If password-free login to the target machine has been configured, no authentication is required.

```
tiup dm deploy ${name} ${version} ./topology.yaml -u ${ssh_user} [-p] [-i /
    ↪ home/root/.ssh/gcp_rsa]
```

The parameters used in this step are as follows.

Parameter	Description
`${name}`	The name of the DM cluster, eg: dm-test

Parameter	Description
`${version}`	The version of the DM cluster. You can see other supported versions by running `tiup list dm-master`.
`./topology.yaml`	The path of the topology configuration file.

Parameter	Description
`-u` or `--user`	Log in to the target machine as the root user or other user account with ssh and sudo privileges to complete the cluster deployment.

Parameter	Description
`-p` or `--password`	The password of target hosts. If specified, password authentication is used.

Parameter	Description
`-i` or `--identity_file`	The path of the SSH identity file. If specified, public key authentication is used (default "/root/.ssh/id_rsa").

At the end of the output log, you will see `Deployed cluster `dm-test` successfully`. This indicates that the deployment is successful.

2.2.5 Step 4: Check the clusters managed by TiUP

```
tiup dm list
```

TiUP supports managing multiple DM clusters. The command above outputs information of all the clusters currently managed by TiUP, including the name, deployment user, version, and secret key information:

```
Name  User  Version Path                                      PrivateKey
----  ----  ------- ----                                      ----------
dm-test tidb v2.0.3 /root/.tiup/storage/dm/clusters/dm-test /root/.tiup/
    ↪ storage/dm/clusters/dm-test/ssh/id_rsa
```

2.2.6 Step 5: Check the status of the deployed DM cluster

To check the status of the `dm-test` cluster, execute the following command:

```
tiup dm display dm-test
```

Expected output includes the instance ID, role, host, listening port, and status (because the cluster is not started yet, so the status is `Down`/`inactive`), and directory information.

2.2.7 Step 6: Start the TiDB cluster

```
tiup dm start dm-test
```

If the output log includes ``Started cluster `dm-test` successfully``, the start is successful.

2.2.8 Step 7: Verify the running status of the TiDB cluster

Check the DM cluster status using TiUP:

```
tiup dm display dm-test
```

If the `Status` is `Up` in the output, the cluster status is normal.

2.2.9 Step 8: Managing migration tasks using dmctl

dmctl is a command-line tool used to control DM clusters. You are recommended to use dmctl via TiUP.

dmctl supports both the command mode and the interactive mode. For details, see Maintain DM Clusters Using dmctl.

2.3 Create a Data Source

> **Note:**
>
> Before creating a data source, you need to Deploy a DM Cluster Using TiUP。

The document describes how to create a data source for the data migration task of TiDB Data Migration (DM).

A data source contains the information for accessing the upstream migration task. Because a data migration task requires referring its corresponding data source to obtain the configuration information of access, you need to create the data source of a task before creating a data migration task. For specific data source management commands, refer to Manage Data Source Configurations.

2.3.1 Step 1: Configure the data source

1. (optional) Encrypt the data source password

   In DM configuration files, it is recommended to use the password encrypted with dmctl. You can follow the example below to obtain the encrypted password of the data source, which can be used to write the configuration file later.

   ```
   tiup dmctl encrypt 'abc!@#123'
   ```

   ```
   MKxn0Qo3m3XOyjCnhEMtsUCm83EhGQDZ/T4=
   ```

2. Write the configuration file of the data source

   For each data source, you need an individual configuration file to create it. You can follow the example below to create a data source whose ID is "mysql-01". First create the configuration file `./source-mysql-01.yaml`:

   ```
   source-id: "mysql-01" # The ID of the data source, you can refer this
       ↪ source-id in the task configuration and dmctl command to
       ↪ associate the corresponding data source.

   from:
     host: "127.0.0.1"
     port: 3306
     user: "root"
     password: "MKxn0Qo3m3XOyjCnhEMtsUCm83EhGQDZ/T4=" # The user password
         ↪ of the upstream data source. It is recommended to use the
         ↪ password encrypted with dmctl.
     security:                                  # The TLS configuration of
         ↪  the upstream data source. If not necessary, it can be deleted.
       ssl-ca: "/path/to/ca.pem"
       ssl-cert: "/path/to/cert.pem"
       ssl-key: "/path/to/key.pem"
   ```

2.3.2 Step 2: Create a data source

You can use the following command to create a data source:

```
tiup dmctl --master-addr <master-addr> operate-source create ./source-mysql
    ↪ -01.yaml
```

For other configuration parameters, refer to Upstream Database Configuration File.

The returned results are as follows:

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-01",
            "worker": "dm-worker-1"
        }
    ]
}
```

2.3.3 Step 3: Query the data source you created

After creating a data source, you can use the following command to query the data source:

- If you konw the source-id of the data source, you can use the `dmctl get-config ↪ source <source-id>` command to directly check the configuration of the data source:

```
tiup dmctl --master-addr <master-addr> get-config source mysql-01
```

```
{
  "result": true,
  "msg": "",
  "cfg": "enable-gtid: false
    flavor: mysql
    source-id: mysql-01
    from:
      host: 127.0.0.1
      port: 3306
      user: root
      password: '******'
}
```

- If you do not know the `source-id`, you can use the `dmctl operate-source show` command to check the source database list, from which you can find the corresponding data source.

```
tiup dmctl --master-addr <master-addr> operate-source show
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "source is added but there is no free worker to bound ",
            "source": "mysql-02",
            "worker": ""
        },
        {
            "result": true,
            "msg": "",
            "source": "mysql-01",
            "worker": "dm-worker-1"
        }
    ]
}
```

2.4 Data Migration Scenarios

2.4.1 Data Migration Scenario Overview

> **Note:**
>
> Before creating a data migration task, you need to perform the following operations:
>
> 1. Deploy a DM Cluster Using TiUP。
> 2. Create a Data Source。

This document introduces how to configure a data migration task in different scenarios. You can choose suitable documents to create your data migration task according to the specific scenario.

In addition to scenario-based documents, you can also refer to the following ones:

- For a complete example of data migration task configuration, refer to DM Advanced Task Configuration File.
- For a data migration task configuration guide, refer to Data Migration Task Configuration Guide.

2.4.1.1 Migrate Data from Multiple Data Sources to TiDB

If you need to migrate data from multiple data sources to TiDB, and to rename tables to avoid migration conflicts caused by duplicate table names in different data sources, or to disable some DDL/DML operations in some tables, refer to Migrate Data from Multiple Data Sources to TiDB.

2.4.1.2 Migrate Sharded Schemas and Sharded Tables to TiDB

If you need to migrate sharded schemas and sharded tables to TiDB, refer to Data Migration Shard Merge Scenario.

2.4.1.3 Migrate Incremental Data to TiDB

If you have already migrated full data using other tools like TiDB Lightning and you need to migrate incremental data, refer to Migrate Incremental Data to TiDB.

2.4.1.4 Migration when the Downstream Table Has More Columns

If you need to customize your table schema in TiDB to include not only all the columns from the source but also additional columns, refer to Migration when the Downstream Table Has More Columns.

2.4.2 Using Migrate Data from Multiple Data Sources to TiDB

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

2.4.2.1 Data source instances

Assume that the data sources 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

Schema	Tables
log	messages

- Instance 3

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

2.4.2.2 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.

2.4.2.3 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

2.4.2.4 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 version is earlier
    ↪ than or equal to v2.0.0-beta.2.
  log-ignored:
    ignore-dbs: ["log"]
```

2.4.2.5 Migration task configuration

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

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

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 version
       ↪  is earlier than or equal to v2.0.0-beta.2.
    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 version
        ↪  is earlier than or equal to v2.0.0-beta.2.
    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 version
        ↪  is earlier than or equal to v2.0.0-beta.2.
    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 version is earlier than
    ↪ or equal to v2.0.0-beta.2.
  log-ignored:
    ignore-dbs: ["log"]

mydumpers:
  global:
    threads: 4
    chunk-filesize: 64

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

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

2.4.3 Data Migration Shard Merge Scenario

This document shows how to use Data Migration (DM) to migrate data to the downstream TiDB in the shard merge scenario.

The example used in this document is a simple scenario where sharded schemas and sharded tables of two data source MySQL instances need to be migrated to a downstream TiDB cluster.

For other scenarios, you can refer to Best Practices of Data Migration in the Shard Merge Scenario.

2.4.3.1 Data source instances

Assume that the data source structures are as follows:

- Instance 1

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

- Instance 2

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

2.4.3.2 Migration requirements

1. Merge the `user.information` tables to the downstream `user.information` table in TiDB.
2. Merge the `store_{01|02}.sale_{01|02}` tables in the above instances to the downstream `store.sale` table in TiDB.
3. Replicate `user` and `store_{01|02}` schemas but do not replicate the `user.log_bak` tables in the above instances.
4. Filter out all the delete operations in the `store_{01|02}.sale_{01|02}` table of the above instances and filter out the `drop database` operation in shemas.

The expected downstream schema after migration is as follows:

Schema	Tables
user	information
store	sale

2.4.3.3 Conflict check across sharded tables

Because migration requirements #1 and #2 involve the DM Shard Merge feature, data from multiple tables might cause conflicts between the primary keys or the unique keys. You need to check these sharded tables. For details, refer to Handle conflicts between primary keys or unique indexes across multiple sharded tables. In this example:

The table schema of `user.information` is

```
CREATE TABLE `information` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `uid` bigint(20) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `data` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `uid` (`uid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

In the above structure, column `id` is the primary key and column `uid` is the unique index. Column `id` has auto-increment attribute and if the ranges of tables overlap, data conflicts might occur. Column `uid` can ensure only a unique index exists globally. So, you can avoid column `id` by following the steps in the section Remove the `PRIMARY KEY` attribute from the column.

The table schema of `store_{01|02}.sale_{01|02}` is

```
CREATE TABLE `sale_01` (
  `sid` bigint(20) NOT NULL,
  `pid` bigint(20) NOT NULL,
  `comment` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`sid`),
  KEY `pid` (`pid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

In the above structure, `sid` is the shard key, which can ensure that the same `sid` only exists in one sharded table. So no data conflict is caused and you do not need to perform extra operations.

2.4.3.4 Migration solution

- To satisfy the migration requirements #1, you do not need to configure the table routing rule. You need to manually create a table based on the requirements in the section Remove the `PRIMARY KEY` attribute from the column:

```
CREATE TABLE `information` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `uid` bigint(20) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `data` varchar(255) DEFAULT NULL,
  INDEX (`id`),
  UNIQUE KEY `uid` (`uid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

And skip precheck in the configuration file:

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

- To satisfy the migration requirement #2, 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 #3, configure the Block and allow table lists as follows:

```
block-allow-list:
  log-bak-ignored:
    do-dbs: ["user", "store_*"]
    ignore-tables:
    - db-name: "user"
      tbl-name: "log_bak"
```

- To satisfy the migration requirement #4, configure the binlog event filter rule as follows:

```
filters:
  ...
  sale-filter-rule:  # filter out all deletion operations of all tables
      ↪  under store_* schema
    schema-pattern: "store_*"
    table-pattern: "sale_*"
    events: ["truncate table", "drop table", "delete"]
    action: Ignore
  store-filter-rule: # filter out the deletion operation of store_*
      ↪ schema
    schema-pattern: "store_*"
    events: ["drop database"]
    action: Ignore
```

2.4.3.5 Migration task configuration

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

```
name: "shard_merge"
task-mode: all                    # full data migration + incremental data
    ↪ migration
meta-schema: "dm_meta"
ignore-checking-items: ["auto_increment_ID"]

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

mysql-instances:
  -
    source-id: "instance-1" # The ID of the data source and can be obtained
        ↪ from the data source configuration
    route-rules: ["store-route-rule", "sale-route-rule"] # Applies to the
        ↪ table route rules of this data source
    filter-rules: ["store-filter-rule" , "sale-filter-rule"] # Applies to
        ↪ the binlog event filter rules of this data source
    block-allow-list: "log-bak-ignored" # Applies to the block and allow
        ↪ lists of this data source
  -
    source-id: "instance-2"
    route-rules: ["store-route-rule", "sale-route-rule"]
    filter-rules: ["store-filter-rule", "sale-filter-rule"]
    block-allow-list: "log-bak-ignored"

### Other common configs shared by all instances

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"

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

block-allow-list:
  log-bak-ignored:
    do-dbs: ["user", "store_*"]
    ignore-tables:
    - db-name: "user"
      tbl-name: "log_bak"
```

2.4.4 Incremental Data Migration Scenario

This document describes how to use Data Migration (DM) to replicate the Binlog from a specified position in the source database to the downstream TiDB. The scenario is based on an example of migrating a data source MySQL instance to TiDB.

2.4.4.1 Data source table

Assume the data source instance is:

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

2.4.4.2 Migration requirements

Only replicate the data change from a specified position in the source database `log` to the TiDB cluster.

2.4.4.3 Incremental data migration operations

This section provides you data migration steps, which helps you use DM to replicate data changes from the `log` database to the TiDB cluster.

2.4.4.3.1 Determines the start position of incremental replication

First you need to determine the replication position of the binlog where you start to migrate data. If you have determined the position of binlog, skip this step.

By following the steps below, you can obtain the position of binlog where you start migrating data in the source data:

- Use Dumpling/Mydumper for full data export. Then use other tools, such as TiDB Lightning, for full data import. After that, you can obtain the replication position by inspecting the metadata files.

```
file  Started dump at: 2020-11-10 10:40:19 SHOW MASTER STATUS:   Log:
↪ mysql-bin.000001     Pos: 2022       GTID: 09bec856-ba95-11ea-850a-58
↪ f2b4af5188:1-9 Finished dump at: 2020-11-10 10:40:20
```

- Use `SHOW BINLOG EVENTS`, or use the `mysqlbinlog` tool to check binlog and select an appropriate position.
- If you want to start replicating binlog at the current time, use `SHOW MASTER STATUS` to check the current position:

```
sql  MySQL [log]> SHOW MASTER STATUS; +------------------+----------+--------------+-
↪  | File | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set
↪         |   +------------------+----------+--------------+------------------+--------
↪  | mysql-bin.000001 | 2022 | |                 | 09bec856-ba95-11ea-850a-58
↪ f2b4af5188:1-9 | +------------------+----------+--------------+------------------+---
↪  1 row in set (0.000 sec)
```

This example starts replicating data changes from `binlog position=(mysql-bin ↪ .000001, 2022), gtid=09bec856-ba95-11ea-850a-58f2b4af5188:1-9`.

2.4.4.3.2 Create tables manually downstream

Because the table SQL statements are created before replication starting point, this incremental replication task does not automatically create tables downstream. So you need to manually create a table schema at the corresponding starting point in the downstream TiDB. The detailed steps are as follows:

```sql
CREATE TABLE `messages` (
  `id` int(11) NOT NULL,
  `message` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
)
```

2.4.4.3.3 Create a replication task

1. Create task configuration `task.yaml` to configure incremental replication mode and replication starting point for each data source. The complete task configuration example is as follows:

"'yaml name: task-test # The name of the task. Should be globally unique. task-mode: incremental # The mode of the task. For "incremental", only incremental data is migrated.

Configure the access information of TiDB database instance: target-database: # Downstream database instance configuration. host: "127.0.0.1" port: 4000 user: "root" password: "" # If password is not empty, it is recommended to use dmctl encrypted password.

Use block-allow-list to configure tables that require sync: block-allow-list: # The filter rule set of the matched table of the data source database instance. Use black-white-list if the DM version is earlier than or equal to v2.0.0-beta.2. bw-rule-1: # The name of the block and allow list rule. do-dbs: ["log"]# The databases to be migrated.

(Optional) If incremental data migration needs to remigrate the data that has already been migrated during full data migration process, you need to enable safe mode to avoid incremental migration errors. ## This scenario usually happens when the full migrated data is not a consistent snapshot of the data source. You need to start migrating incremental data at a position before the full data migration starting point. syncers: # The configuration parameters of sync unit. global: # The name of the configuration. safe-mode: true # If you set `safe-mode` to `true`, `INSERT`` from data sources is rewritten ↪ to`REPLACE`and`UPDATE`is rewritten to`DELETE`and`REPLACE'. This is to ensure that when primary keys or unique keys exist in table structure, you can re-import DML when migrating data. TiDB DM automatically enables the safe mode within 1 minute immediately after the incremental replication task is started or resumed.

Configure the data source mysql-instances: - source-id: "mysql-01" # The ID of data source. You can obtain it from the configuration of the data source. block-allow-list: "bw-rule-1" # To import the block-allow-list configuration above. syncer-config-name: "global" # To import the incremental data migration configuration of syncers. meta: # If `task-mode` is `incremental` and the `checkpoint` in the downstream database does not exist, `meta` is the starting point of binlog; If `checkpoint` exists, base it on `checkpoint`. binlog-name: "mysql-bin. 00001" binlog-pos: 2022 binlog-gtid: "09bec856-ba95-11ea-850a-58f2b4af5188:1-9" "'

2. Create a replication task using the `start-task` command:

```
bash   tiup dmctl --master-addr <master-addr> start-task task.yaml
```

```
{       "result": true,     "msg": "",      "sources": [         {
↪              "result": true,             "msg": "",              "source
↪ ": "mysql-01",            "worker": "127.0.0.1:8262"       }       ]    }
```

3. Check the replication task using the `query-status` command to ensure that no error message occurs:

```
bash   tiup dmctl --master-addr <master-addr> query-status test
```

```
{       "result": true,     "msg": "",      "sources": [         {
↪             "result": true,             "msg": "",              "sourceStatus
↪ ": {              "source": "mysql-01",              "worker": "127.0.0.1:8262",
↪               "result": null,              "relayStatus": null          },
↪               "subTaskStatus": [              {                "
↪ name": "task-test",               "stage": "Running",               "
↪ unit": "Sync",             "result": null,              "
↪ unresolvedDDLLockID": "",             "sync": {               "
↪ totalEvents": "0",            "totalTps": "0",                "
↪ recentTps": "0",              "masterBinlog": "(mysql-bin.000001,
↪  2022)",            "masterBinlogGtid": "09bec856-ba95-11ea-850a
↪ -58f2b4af5188:1-9",            "syncerBinlog": "(mysql-bin.000001,
↪ 2022)",          "syncerBinlogGtid": "",              "
↪ blockingDDLs": [              ],             "unresolvedGroups
↪ ": [             ],             "synced": true,
↪               "binlogType": "remote"              }              }
↪             ]        }      ]   }
```

2.4.4.4 Test replication tasks

Insert new data in the source database:

```
MySQL [log]> INSERT INTO messages VALUES (4, 'msg4'), (5, 'msg5');
Query OK, 2 rows affected (0.010 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

Currently, the source data is:

```
MySQL [log]> SELECT * FROM messages;
+----+---------+
| id | message |
+----+---------+
|  1 | msg1    |
|  2 | msg2    |
|  3 | msg3    |
|  4 | msg4    |
|  5 | msg5    |
+----+---------+
5 rows in set (0.001 sec)
```

If you query data in the downstream, you can find that the data after `(3, 'msg3')` is replicated successfully:

```
MySQL [log]> SELECT * FROM messages;
+----+---------+
| id | message |
+----+---------+
```

```
|  4 | msg4   |
|  5 | msg5   |
+----+---------+
2 rows in set (0.001 sec)
```

2.4.5 Migration when There Are More Columns in the Downstream TiDB Table

This document describes how to migrate tables using DM when there are more columns in the downstream TiDB table schema than the upstream table schema.

2.4.5.1 The table shcema of the data source

This document uses the follwing data source example:

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

2.4.5.2 Migration requirements

Create a customized table `log.messages` in TiDB. Its schema contains not only all the columns in the `log.messages` table of the data source, but also additional columns. In this case, migrate the table `log.messages` of the data source to the table `log.messages` of the TiDB cluster.

> **Note:**
>
> - The columns that only exist in the downstream TiDB must be given a default value or allowed to be `NULL`.
> - For tables that are being migrated by DM, you can directly add new columns in the downstream TiDB that are given a default value or allowed to be `NULL`. Adding such new columns does not affect the data migration.

2.4.5.3 Only migrate incremental data to TiDB and the downstream TiDB table has more columns

If your migration task contains full data migration, the task can operate normally. If you have already used other tools to do full data migration and this migration task only uses

DM to replicate incremental data, refer to Migrate Incremental Data to TiDB to create a data migration task. At the same time, you need to manually configure the table schema in DM for MySQL binlog parsing.

Otherwise, after creating the task, the following data migration errors occur when you execute the query-status' command:

```
"errors": [
    {
        "ErrCode": 36027,
        "ErrClass": "sync-unit",
        "ErrScope": "internal",
        "ErrLevel": "high",
        "Message": "startLocation: [position: (mysql-bin.000001, 2022), gtid-
            ↪ set:09bec856-ba95-11ea-850a-58f2b4af5188:1-9 ], endLocation: [
            ↪ position: (mysql-bin.000001, 2022), gtid-set: 09bec856-ba95-11
            ↪ ea-850a-58f2b4af5188:1-9]: gen insert sqls failed, schema: log
            ↪ , table: messages: Column count doesn't match value count: 3 (
            ↪ columns) vs 2 (values)",
        "RawCause": "",
        "Workaround": ""
    }
]
```

The reason for the above errors is that when DM migrates the binlog event, if DM has not maintained internally the table schema corresponding to that table, DM tries to use the current table schema in the downstream to parse the binlog event and generate the corresponding DML statement. If the number of columns in the binlog event is inconsistent with the number of columns in the downstream table schema, the above error might occur.

In such cases, you can execute the `operate-schema` command to specify for the table a table schema that matches the binlog event. If you are migrating sharded tables, you need to configure the table schema in DM for parsing MySQL binlog for each sharded tables according to the following steps:

1. Specify the table schema for the table `log.messages` to be migrated in the data source. The table schema needs to correspond to the data of the binlog event to be replicated by DM. Then save the `CREATE TABLE` table schema statement in a file. For example, save the following table schema in the `log.messages.sql` file:

   ```sql
   CREATE TABLE `messages` (
     `id` int(11) NOT NULL,
     `message` varchar(255) DEFAULT NULL,
     PRIMARY KEY (`id`)
   )
   ```

2. Execute the `operate-schema` command to set the table schema. At this time, the task should be in the `Paused` state because of the above error.

```
tiup dmctl --master-addr <master-addr> operate-schema set -s mysql-01
   ↪ task-test -d log -t message log.message.sql
```

3. Execute the `resume-task` command to resume the `Paused` task.

4. Execute the `query-status` command to check whether the data migration task is running normally.

3 Deploy

3.1 Software and Hardware Requirements

TiDB Data Migration (DM) supports mainstream Linux operating systems. See the following table for specific version requirements:

Linux OS Platform	Version
Red Hat Enterprise Linux	7.3 or later
CentOS	7.3 or later
Oracle Enterprise Linux	7.3 or later
Ubuntu LTS	16.04 or later

DM can be deployed and run on Intel architecture servers and mainstream virtualization environments.

3.1.1 Recommended server requirements

DM can be deployed and run on a 64-bit generic hardware server platform (Intel x86-64 architecture). For servers used in the development, testing, and production environments, this section illustrates recommended hardware configurations (these do not include the resources used by the operating system).

3.1.1.1 Development and test environments

Component	CPU	Memory	Local Storage	Network	Number of Instances (Minimum Requirement)
DM-master	4 core+	8 GB+	SAS, 200 GB+	Gigabit network card	1
DM-worker	8 core+	16 GB+	SAS, 200 GB+ (Greater than the size of the migrated data)	Gigabit network card	The number of upstream MySQL instances

Note:

- In the test environment, DM-master and DM-worker used for functional verification can be deployed on the same server.
- To prevent interference with the accuracy of the performance test results, it is **not recommended** to use low-performance storage and network hardware configurations.
- If you need to verify the function only, you can deploy a DM-master on a single machine. The number of DM-worker deployed must be greater than or equal to the number of upstream MySQL instances. To ensure high availability, it is recommended to deploy more DM-workers.
- DM-worker stores full data in the `dump` and `load` phases. Therefore, the disk space for DM-worker needs to be greater than the total amount of data to be migrated. If the relay log is enabled for the migration task, DM-worker needs additional disk space to store upstream binlog data.

3.1.1.2 Production environment

Component	CPU	Memory	Hard Disk Type	Network	Number of Instances (Minimum Requirement)
DM-master	4 core+	8 GB+	SAS, 200 GB+	Gigabit network card	3
DM-worker	16 core+	32 GB+	SSD, 200 GB+ (Greater than the size of the migrated data)	10 Gigabit network card	Greater than the number of upstream MySQL instances
Monitor	8 core+	16 GB+	SAS, 200 GB+	Gigabit network card	1

Note:

- In the production environment, it is not recommended to deploy and run DM-master and DM-worker on the same server, because when DM-worker writes data to disks, it might interfere with the use of disks by DM-master's high availability component.
- If a performance issue occurs, you are recommended to modify the task configuration file according to the Optimize Configuration of DM document. If the performance is not effectively optimized by tuning the

configuration file, you can try to upgrade the hardware of your server.

3.2 Deploy a DM Cluster

3.2.1 Deploy a DM Cluster Using TiUP

TiUP is a cluster operation and maintenance tool introduced in TiDB 4.0. TiUP provides TiUP DM, a cluster management component written in Golang. By using TiUP DM, you can easily perform daily TiDB Data Migration (DM) operations, including deploying, starting, stopping, destroying, scaling, and upgrading a DM cluster, and manage DM cluster parameters.

TiUP supports deploying DM v2.0 or later DM versions. This document introduces how to deploy DM clusters of different topologies.

Note:

If your target machine's operating system supports SELinux, make sure that SELinux is **disabled**.

3.2.1.1 Prerequisites

When DM performs a full data replication task, the DM-worker is bound with only one upstream database. The DM-worker first exports the full amount of data locally, and then imports the data into the downstream database. Therefore, the worker's host needs sufficient storage space (The storage path is specified later when you create the task).

In addition, you need to meet the hardware and software requirements when deploying a DM cluster.

3.2.1.2 Step 1: Install TiUP on the control machine

Log in to the control machine using a regular user account (take the `tidb` user as an example). All the following TiUP installation and cluster management operations can be performed by the `tidb` user.

1. Install TiUP by executing the following command:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
```

After the installing, `~/.bashrc` has been modified to add TiUP to PATH, so you need to open a new terminal or redeclare the global environment variables `source ~/.bashrc` to use it.

2. Install the TiUP DM component:

```
tiup install dm dmctl
```

3.2.1.3 Step 2: Edit the initialization configuration file

According to the intended cluster topology, you need to manually create and edit the cluster initialization configuration file.

You need to create a YAML configuration file (named `topology.yaml` for example) according to the configuration file template. For other scenarios, edit the configuration accordingly.

You can use the command `tiup dm template > topology.yaml` to generate a configuration file template quickly.

The configuration of deploying three DM-masters, three DM-workers, and one monitoring component instance is as follows:

```
### The global variables apply to all other components in the configuration.
    ↪  If one specific value is missing in the component instance, the
    ↪ corresponding global variable serves as the default value.
global:
  user: "tidb"
  ssh_port: 22
  deploy_dir: "/dm-deploy"
  data_dir: "/dm-data"

server_configs:
  master:
    log-level: info
    # rpc-timeout: "30s"
    # rpc-rate-limit: 10.0
    # rpc-rate-burst: 40
  worker:
    log-level: info

master_servers:
  - host: 10.0.1.11
    name: master1
    ssh_port: 22
    port: 8261
    # peer_port: 8291
    # deploy_dir: "/dm-deploy/dm-master-8261"
```

```
    # data_dir: "/dm-data/dm-master-8261"
    # log_dir: "/dm-deploy/dm-master-8261/log"
    # numa_node: "0,1"
    # The following configs are used to overwrite the `server_configs.master
       ↪ ` values.
    config:
      log-level: info
      # rpc-timeout: "30s"
      # rpc-rate-limit: 10.0
      # rpc-rate-burst: 40
  - host: 10.0.1.18
    name: master2
    ssh_port: 22
    port: 8261
  - host: 10.0.1.19
    name: master3
    ssh_port: 22
    port: 8261
### If you do not need to ensure high availability of the DM cluster, deploy
   ↪  only one DM-master node, and the number of deployed DM-worker nodes
   ↪ must be no less than the number of upstream MySQL/MariaDB instances
   ↪ to be migrated.
### To ensure high availability of the DM cluster, it is recommended to
   ↪ deploy three DM-master nodes, and the number of deployed DM-worker
   ↪ nodes must exceed the number of upstream MySQL/MariaDB instances to
   ↪ be migrated (for example, the number of DM-worker nodes is two more
   ↪ than the number of upstream instances).
worker_servers:
  - host: 10.0.1.12
    ssh_port: 22
    port: 8262
    # deploy_dir: "/dm-deploy/dm-worker-8262"
    # log_dir: "/dm-deploy/dm-worker-8262/log"
    # numa_node: "0,1"
    # The following configs are used to overwrite the `server_configs.worker
       ↪ ` values.
    config:
      log-level: info
  - host: 10.0.1.19
    ssh_port: 22
    port: 8262

monitoring_servers:
  - host: 10.0.1.13
    ssh_port: 22
```

```
    port: 9090
    # deploy_dir: "/tidb-deploy/prometheus-8249"
    # data_dir: "/tidb-data/prometheus-8249"
    # log_dir: "/tidb-deploy/prometheus-8249/log"

grafana_servers:
  - host: 10.0.1.14
    port: 3000
    # deploy_dir: /tidb-deploy/grafana-3000

alertmanager_servers:
  - host: 10.0.1.15
    ssh_port: 22
    web_port: 9093
    # cluster_port: 9094
    # deploy_dir: "/tidb-deploy/alertmanager-9093"
    # data_dir: "/tidb-data/alertmanager-9093"
    # log_dir: "/tidb-deploy/alertmanager-9093/log"
```

Note:

- It is not recommended to run too many DM-workers on one host. Each DM-worker should be allocated at least 2 core CPU and 4 GiB memory.
- Make sure that the ports among the following components are interconnected:
  - The `peer_port` (`8291` by default) among the DM-master nodes are interconnected.
  - Each DM-master node can connect to the `port` of all DM-worker nodes (`8262` by default).
  - Each DM-worker node can connect to the `port` of all DM-master nodes (`8261` by default).
  - The TiUP nodes can connect to the `port` of all DM-master nodes (`8261` by default).
  - The TiUP nodes can connect to the `port` of all DM-worker nodes (`8262` by default).

For more `master_servers.host.config` parameter description, refer to master parameter. For more `worker_servers.host.config` parameter description, refer to worker parameter.

3.2.1.4 Step 3: Execute the deployment command

Note:

You can use secret keys or interactive passwords for security authentication when you deploy TiDB using TiUP:

- If you use secret keys, you can specify the path of the keys through `-i` or `--identity_file`;
- If you use passwords, add the `-p` flag to enter the password interaction window;
- If password-free login to the target machine has been configured, no authentication is required.

```
tiup dm deploy ${name} ${version} ./topology.yaml -u ${ssh_user} [-p] [-i /
    ↪ home/root/.ssh/gcp_rsa]
```

The parameters used in this step are as follows.

Parameter	Description
`${name}`	The name of the DM cluster, eg: dm-test

Parameter	Description
`${version}`	The version of the DM cluster. You can see other supported versions by running `tiup list dm-master`.
`./topology.yaml`	The path of the topology configuration file.

Parameter	Description
`-u` or `--user`	Log in to the target machine as the root user or other user account with ssh and sudo privileges to complete the cluster deployment.

Parameter	Description
`-p` or `--password`	The password of target hosts. If specified, password authentication is used.

Parameter	Description
`-i` or `--identity_file`	The path of the SSH identity file. If specified, public key authentication is used (default "/root/.ssh/id_rsa").

At the end of the output log, you will see ``Deployed cluster `dm-test` successfully``. This indicates that the deployment is successful.

3.2.1.5 Step 4: Check the clusters managed by TiUP

```
tiup dm list
```

TiUP supports managing multiple DM clusters. The command above outputs information of all the clusters currently managed by TiUP, including the name, deployment user, version, and secret key information:

```
Name  User Version Path                                  PrivateKey
----  ---- ------- ----                                  ----------
dm-test tidb v2.0.3 /root/.tiup/storage/dm/clusters/dm-test /root/.tiup/
    ↪ storage/dm/clusters/dm-test/ssh/id_rsa
```

3.2.1.6 Step 5: Check the status of the deployed DM cluster

To check the status of the `dm-test` cluster, execute the following command:

```
tiup dm display dm-test
```

Expected output includes the instance ID, role, host, listening port, and status (because the cluster is not started yet, so the status is `Down`/`inactive`), and directory information.

3.2.1.7 Step 6: Start the TiDB cluster

```
tiup dm start dm-test
```

If the output log includes ``Started cluster `dm-test` successfully``, the start is successful.

3.2.1.8 Step 7: Verify the running status of the TiDB cluster

Check the DM cluster status using TiUP:

```
tiup dm display dm-test
```

If the `Status` is `Up` in the output, the cluster status is normal.

3.2.1.9 Step 8: Managing migration tasks using dmctl

dmctl is a command-line tool used to control DM clusters. You are recommended to use dmctl via TiUP.

dmctl supports both the command mode and the interactive mode. For details, see Maintain DM Clusters Using dmctl.

3.2.2 Deploy a DM Cluster Offline Using TiUP (Experimental)

> **Warning:**
>
> Using TiUP to deploy a DM cluster offline is still an experimental feature. It is **NOT** recommended to use this feature in production.

This document describes how to deploy a DM cluster offline using TiUP.

3.2.2.1 Step 1: Prepare the TiUP offline component package

- Install the TiUP package manager online.

  1. Install the TiUP tool:

     ```
     curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.
         ↪ com/install.sh | sh
     ```

  2. Redeclare the global environment variables:

     ```
     source .bash_profile
     ```

  3. Confirm whether TiUP is installed:

     ```
     which tiup
     ```

- Pull the mirror using TiUP

  1. Pull the needed components on a machine that has access to the Internet:

     ```
     export version=v2.0.3 # You can modify it to the needed version.
     tiup mirror clone tidb-dm-${version}-linux-amd64 --os=linux --arch=
         ↪ amd64 \
         --dm-master=${version} --dm-worker=${version} --dmctl=${version
             ↪ } \
         --alertmanager=v0.17.0 --grafana=v4.0.3 --prometheus=v4.0.3 \
         --tiup=v$(tiup --version|grep 'tiup'|awk -F ' ' '{print $1}')
             ↪ --dm=v$(tiup --version|grep 'tiup'|awk -F ' ' '{print $1
             ↪ }')
     ```

     The command above creates a directory named `tidb-dm-${version}-linux-↪ amd64` in the current directory, which contains the component package managed by TiUP.

  2. Pack the component package by using the `tar` command and send the package to the control machine in the isolated environment:

     ```
     tar czvf tidb-dm-${version}-linux-amd64.tar.gz tidb-dm-${version}-
         ↪ linux-amd64
     ```

     `tidb-dm-${version}-linux-amd64.tar.gz` is an independent offline environment package.

3.2.2.2 Step 2: Deploy the offline TiUP component

After sending the package to the control machine of the target cluster, install the TiUP component by running the following command:

```
export version=v2.0.3 # You can modify it to the needed version.
tar xzvf tidb-dm-${version}-linux-amd64.tar.gz
sh tidb-dm-${version}-linux-amd64/local_install.sh
source /home/tidb/.bash_profile
```

The `local_install.sh` script automatically executes the `tiup mirror set tidb-dm-↪ ${version}-linux-amd64` command to set the current mirror address to `tidb-dm-${↪ version}-linux-amd64`.

To switch the mirror to another directory, manually execute the `tiup mirror set ↪ <mirror-dir>` command. If you want to switch back to the official mirror, execute `tiup mirror set https://tiup-mirrors.pingcap.com`.

3.2.2.3 Step 3: Edit the initialization configuration file

You need to edit the cluster initialization configuration file according to different cluster topologies.

For the full configuration template, refer to the TiUP configuration parameter template. Create a configuration file `topology.yaml`. In other combined scenarios, edit the configuration file as needed according to the templates.

The configuration of deploying three DM-masters, three DM-workers, and one monitoring component instance is as follows:

```
---
global:
  user: "tidb"
  ssh_port: 22
  deploy_dir: "/home/tidb/dm/deploy"
  data_dir: "/home/tidb/dm/data"
  # arch: "amd64"

master_servers:
  - host: 172.19.0.101
  - host: 172.19.0.102
  - host: 172.19.0.103

worker_servers:
  - host: 172.19.0.101
  - host: 172.19.0.102
  - host: 172.19.0.103
```

```
monitoring_servers:
  - host: 172.19.0.101

grafana_servers:
  - host: 172.19.0.101

alertmanager_servers:
  - host: 172.19.0.101
```

Note:

- If you do not need to ensure high availability of the DM cluster, deploy only one DM-master node, and the number of deployed DM-worker nodes must be no less than the number of upstream MySQL/MariaDB instances to be migrated.
- To ensure high availability of the DM cluster, it is recommended to deploy three DM-master nodes, and the number of deployed DM-worker nodes must be greater than the number of upstream MySQL/MariaDB instances to be migrated (for example, the number of DM-worker nodes is two more than the number of upstream instances).
- For parameters that should be globally effective, configure these parameters of corresponding components in the `server_configs` section of the configuration file.
- For parameters that should be effective on a specific node, configure these parameters in `config` of this node.
- Use `.` to indicate the subcategory of the configuration, such as `log.slow` ↪ `-threshold`. For more formats, see TiUP configuration template.
- For more parameter description, see master `config.toml.example` and worker `config.toml.example`.
- Make sure that the ports among the following components are interconnected:
  - The `peer_port` (`8291` by default) among the DM-master nodes are interconnected.
  - Each DM-master node can connect to the `port` of all DM-worker nodes (`8262` by default).
  - Each DM-worker node can connect to the `port` of all DM-master nodes (`8261` by default).
  - The TiUP nodes can connect to the `port` of all DM-master nodes (`8261` by default).
  - The TiUP nodes can connect to the `port` of all DM-worker nodes (`8262` by default).

3.2.2.4 Step 4: Execute the deployment command

> **Note:**
>
> You can use secret keys or interactive passwords for security authentication when you deploy DM using TiUP:
>
> - If you use secret keys, you can specify the path of the keys through `-i` or `--identity_file`;
> - If you use passwords, add the `-p` flag to enter the password interaction window;
> - If password-free login to the target machine has been configured, no authentication is required.

```
tiup dm deploy dm-test ${version} ./topology.yaml --user root [-p] [-i /home
    ↪ /root/.ssh/gcp_rsa]
```

In the above command:

- The name of the deployed DM cluster is `dm-test`.
- The version of the DM cluster is `${version}`. You can view the latest versions supported by TiUP by running `tiup list dm-master`.
- The initialization configuration file is `topology.yaml`.
- `--user root`: Log in to the target machine through the `root` key to complete the cluster deployment, or you can use other users with `ssh` and `sudo` privileges to complete the deployment.
- `[-i]` and `[-p]`: optional. If you have configured login to the target machine without password, these parameters are not required. If not, choose one of the two parameters. `[-i]` is the private key of the `root` user (or other users specified by `--user`) that has access to the target machine. `[-p]` is used to input the user password interactively.
- TiUP DM uses the embedded SSH client. If you want to use the SSH client native to the control machine system, edit the configuration according to using the system's native SSH client to connect to the cluster.

At the end of the output log, you will see ``Deployed cluster `dm-test` successfully``. This indicates that the deployment is successful.

3.2.2.5 Step 5: Check the clusters managed by TiUP

```
tiup dm list
```

TiUP supports managing multiple DM clusters. The command above outputs information of all the clusters currently managed by TiUP, including the name, deployment user, version, and secret key information:

```
Name  User  Version Path                                             PrivateKey
----  ----  ------- ----                                             ----------
dm-test tidb v2.0.3 /root/.tiup/storage/dm/clusters/dm-test /root/.tiup/
    ↪ storage/dm/clusters/dm-test/ssh/id_rsa
```

3.2.2.6 Step 6: Check the status of the deployed DM cluster

To check the status of the `dm-test` cluster, execute the following command:

```
tiup dm display dm-test
```

Expected output includes the instance ID, role, host, listening port, and status (because the cluster is not started yet, so the status is `Down`/`inactive`), and directory information of the `dm-test` cluster.

3.2.2.7 Step 7: Start the cluster

```
tiup dm start dm-test
```

If the output log includes ``Started cluster `dm-test` successfully``, the start is successful.

3.2.2.8 Step 8: Verify the running status of the cluster

Check the DM cluster status using TiUP:

```
tiup dm display dm-test
```

If the `Status` is `Up` in the output, the cluster status is normal.

3.2.3 Deploy Data Migration Using DM Binary

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

> **Note:**
>
> In the production environment, it is recommended to use TiUP to deploy a DM cluster.

3.2.3.1 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, and dmctl. The `conf` directory contains the sample configuration files.

3.2.3.2 Sample scenario

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

Two DM-worker nodes and three DM-master nodes are deployed on five servers.

Here is the address of each node:

Instance	Server address	Port
DM-master1	192.168.0.4	8261
DM-master2	192.168.0.5	8261
DM-master3	192.168.0.6	8261
DM-worker1	192.168.0.7	8262
DM-worker2	192.168.0.8	8262

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

Note:

- If you deploy multiple DM-master or DM-worker instances in a single server, the port and working directory of each instance must be unique.
- If you do not need to ensure high availability of the DM cluster, deploy only one DM-master node, and the number of deployed DM-worker nodes must be no less than the number of upstream MySQL/MariaDB instances to be migrated.
- To ensure high availability of the DM cluster, it is recommended to deploy three DM-master nodes, and the number of deployed DM-worker nodes must be greater than the number of upstream MySQL/MariaDB instances to be migrated (for example, the number of DM-worker nodes is two more than the number of upstream instances).
- Make sure that the ports among the following components are interconnected:
  - The `8291` ports among the DM-master nodes are interconnected.
  - Each DM-master node can connect to the `8262` ports of all DM-worker nodes.
  - Each DM-worker node can connect to the `8261` port of all DM-master nodes.

3.2.3.2.1 Deploy DM-master

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

DM-master command-line parameters

The following is the description of DM-master command-line parameters:

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

```
Usage of dm-master:
  -L string
        log level: debug, info, warn, error, fatal (default "info")
  -V    prints version and exit
  -advertise-addr string
        advertise address for client traffic (default "${master-addr}")
  -advertise-peer-urls string
        advertise URLs for peer traffic (default "${peer-urls}")
  -config string
        path to config file
  -data-dir string
        path to the data directory (default "default.${name}")
  -initial-cluster string
        initial cluster configuration for bootstrapping, e.g. dm-master=http
            ↪ ://127.0.0.1:8291
  -join string
        join to an existing cluster (usage: cluster's "${master-addr}" list,
            ↪ e.g. "127.0.0.1:8261,127.0.0.1:18261"
  -log-file string
        log file path
  -master-addr string
        master API server and status addr
  -name string
        human-readable name for this DM-master member
  -peer-urls string
        URLs for peer traffic (default "http://127.0.0.1:8291")
  -print-sample-config
        print sample config file of dm-worker
```

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

DM-master configuration file

The following is the configuration file of DM-master. It is recommended that you configure DM-master by using this method.

1. Write the following configuration to `conf/dm-master1.toml`:

   "'toml # Master Configuration. name = "master1"

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

   # The listening address of DM-master. master-addr = "192.168.0.4:8261"

   # The peer URLs of DM-master. peer-urls = "192.168.0.4:8291"

   # The value of `initial-cluster` is the combination of the `advertise-peer-urls` value of all DM-master nodes in the initial cluster. initial-cluster = "master1=http://192.168.0.4:8291,master2=http://192.168.0.5:8291,master3=http://192.168.0.6:8291" "'

2. Execute the following command in the terminal to run DM-master:

   `bash ./bin/dm-master -config conf/dm-master1.toml`

> **Note:**
> The console does not output logs after this command is executed. If you want to view the runtime log, you can execute `tail -f dm-master.log`.

3. For DM-master2 and DM-master3, change `name` in the configuration file to `master2` and `master3` respectively, and change `peer-urls` to `192.168.0.5:8291` and `192.168.0.6:8291` respectively. Then repeat Step 2.

3.2.3.2.2 Deploy DM-worker

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

DM-worker command-line parameters

The following is the description of the DM-worker command-line parameters:

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

```
Usage of worker:
  -L string
        log level: debug, info, warn, error, fatal (default "info")
  -V    prints version and exit
  -advertise-addr string
        advertise address for client traffic (default "${worker-addr}")
  -config string
        path to config file
  -join string
        join to an existing cluster (usage: dm-master cluster's "${master-
           ↪ addr}")
```

```
-keepalive-ttl int
      dm-worker's TTL for keepalive with etcd (in seconds) (default 10)
-log-file string
      log file path
-name string
      human-readable name for DM-worker member
-print-sample-config
      print sample config file of dm-worker
-worker-addr string
      listen address for client traffic
```

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

DM-worker configuration file

The following is the DM-worker configuration file. It is recommended that you configure DM-worker by using this method.

1. Write the following configuration to `conf/dm-worker1.toml`:

   "'toml # Worker Configuration. name = "worker1"

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

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

   # The master-addr configuration of the DM-master nodes in the cluster. join = "192.168.0.4:8261,192.168.0.5:8261,192.168.0.6:8261" "'

2. Execute the following command in the terminal to run DM-worker:

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

3. For DM-worker2, change `name` in the configuration file to `worker2`. Then repeat Step 2.

Now, a DM cluster is successfully deployed.

3.2.4 Use Kubernetes

3.3 Migrate Data Using Data Migration

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

3.3.1 Step 1: Deploy the DM cluster

It is recommended to deploy the DM cluster using TiUP. You can also deploy the DM cluster using binary for trial or test.

Note:

- For database 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. See Encrypt the database password using dmctl.
- The user of the upstream and downstream databases must have the corresponding read and write privileges.

3.3.2 Step 2: Check the cluster information

After the DM cluster is deployed using TiUP, 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=
Downstream TiDB	172.16.10.83	4000	root	

The list of privileges needed on the MySQL host can be found in the precheck documentation.

3.3.3 Step 3: Create data source

1. Write MySQL-1 related information to `conf/source1.yaml`:

```
# MySQL1 Configuration.

source-id: "mysql-replica-01"
# 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

from:
  host: "172.16.10.81"
  user: "root"
  password: "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU="
  port: 3306
```

2. Execute the following command in the terminal, and use `tiup dmctl` to load the MySQL-1 data source configuration to the DM cluster:

```
tiup dmctl --master-addr 172.16.10.71:8261 operate-source create conf/
    ↪ source1.yaml
```

3. For MySQL-2, modify the relevant information in the configuration file and execute the same `dmctl` command.

3.3.4 Step 4: 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.

Edit 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 version is
     ↪ earlier than or equal to v2.0.0-beta.2.
  # The configuration item name of the dump processing unit, used to quote
     ↪ the global configuration of the dump unit.
  mydumper-config-name: "global"

-
  source-id: "mysql-replica-02"
  block-allow-list: "global" # Use black-white-list if the DM version is
     ↪ earlier than or equal to v2.0.0-beta.2.
  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 version
   ↪ is earlier than or equal to v2.0.0-beta.2.
  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:
    extra-args: ""
```

3.3.5 Step 5: 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.

Run the `tiup dmctl` command to start the data migration tasks. `task.yaml` is the configuration file that is edited above.

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

3.3.6 Step 6: 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 `tiup dmctl`:

```
tiup dmctl --master-addr 172.16.10.71:8261 query-status
```

3.3.7 Step 7: Stop the data migration task

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

```
tiup dmctl --master-addr 172.16.10.71:8261 stop-task test
```

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

3.3.8 Step 8: Monitor the task and check logs

Assuming that Prometheus, Alertmanager, and Grafana are successfully deployed along with the DM cluster deployment using TiUP, 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 TiUP, the log directory is `{log_dir}` 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 TiUP, the log directory is `{log_dir}` in the DM-worker node.

3.4 DM Cluster Performance Test

This document describes how to build a test scenario to do a performance test on the DM cluster, including the speed test and latency test regarding data migration.

3.4.1 Migration data flow

You can use a simple migration data flow, that is, MySQL -> DM -> TiDB, to test the data migration performance of the DM cluster.

3.4.2 Deploy test environment

- Deploy the TiDB test cluster using TiUP, with all default configurations.
- Deploy the MySQL service. Enable the `ROW` mode for binlog, and use default configurations for other configuration items.
- Deploy a DM cluster, with a DM-worker and a DM-master.

3.4.3 Performance test

3.4.3.1 Table schema

Use tables with the following schema for the performance 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
```

3.4.3.2 Full import benchmark case

3.4.3.2.1 Generate test data

Use `sysbench` to create test tables upstream and generate test data for full import. Execute the following `sysbench` command to generate test 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
```

3.4.3.2.2 Create a data migration task

1. Create an upstream MySQL source and set `source-id` to `source-1`. For details, see Load the Data Source Configurations.

2. Create a migration task (in `full` mode). The following is a task configuration template:

“‘yaml
name: test-full
task-mode: full

Configure the migration task using the TiDB information of your actual test environment. target-database: host: “192.168.0.1” port: 4000 user: “root” password: “”

mysql-instances: - source-id: “source-1” block-allow-list: “instance” mydumper-config-name: “global” loader-thread: 16

Configure the name of the database where sysbench generates data. block-allow-list: instance: do-dbs: [“dm_benchmark”]

mydumpers: global: rows: 32000 threads: 32 “‘

For details about how to create a migration task, see Create a Data Migration Task.

> **Note:**
>
> - To enable concurrently exporting data from a single table using multi-thread, you can use the `rows` option in the `mydumpers` configuration item. This speeds up data export.
> - To test the performance under different configurations, you can tune `loader-thread` in the `mysql-instances` configuration, as well as `rows` and `threads` in the `mydumpers` configuration item.

3.4.3.2.3 Get test results

Observe the DM-worker log. When you see `all data files have been finished`, it means that full data has been imported. In this case, you can see the time consumed to import data. The sample log is as follows:

```
[INFO] [loader.go:604] ["all data files have been finished"] [task=test] [
    ↪ unit=load] ["cost time"=52.439796ms]
```

According to the size of the test data and the time consumed to import data, you can calculate the migration speed of the full data.

3.4.3.3 Incremental replication benchmark case

3.4.3.3.1 Initialize tables

Use `sysbench` to create test tables in the upstream.

3.4.3.3.2 Create a data migration task

1. Create the source of the upstream MySQL. Set `source-id` to `source-1` (if the source has been created in the full import benchmark case, you do not need to create it again). For details, see Load the Data Source Configurations.

2. Create a DM migration task (in `all` mode). The following is an example of the task configuration file:

"'yaml
name: test-all task-mode: all

Configure the migration task using the TiDB information of your actual test environment. target-database: host: "192.168.0.1" port: 4000 user: "root" password: ""

mysql-instances: - source-id: "source-1" block-allow-list: "instance" syncer-config-name: "global"

Configure the name of the database where sysbench generates data. block-allow-list: instance: do-dbs: ["dm_benchmark"]

syncers: global: worker-count: 16 batch: 100 "'

For details about how to create a data migration task, see Create a Data Migration Task.

> **Note:**
>
> To test the performance under different configurations, you can tune `worker`
> ↪ `-count` and `batch` in the `syncers` configuration item.

3.4.3.3.3 Generate incremental data

To continuously generate incremental data in the upstream, run the `sysbench` command:

```
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
```

> **Note:**

You can test the data migration performance under different scenarios by using different `sysbench` statements.

3.4.3.3.4 Get test results

To observe the migration status of DM, you can run the `query-status` command. To observe the monitoring metrics of DM, you can use Grafana. Here the monitoring metrics refer to `finished sqls jobs` (the number of jobs finished per unit time), etc. For more information, see Binlog Migration Monitoring Metrics.

4 Maintain

4.1 Tools

4.1.1 Maintain a DM Cluster Using TiUP

This document introduces how to maintain a DM cluster using the TiUP DM component.

If you have not deployed a DM cluster yet, you can refer to Deploy a DM Cluster Using TiUP for instructions.

Note:

- Make sure that the ports among the following components are interconnected
  - The `peer_port` (`8291` by default) among the DM-master nodes are interconnected.
  - Each DM-master node can connect to the `port` of all DM-worker nodes (`8262` by default).
  - Each DM-worker node can connect to the `port` of all DM-master nodes (`8261` by default).
  - The TiUP nodes can connect to the `port` of all DM-master nodes (`8261` by default).
  - The TiUP nodes can connect to the `port` of all DM-worker nodes (`8262` by default).

For the help information of the TiUP DM component, run the following command:

```
tiup dm --help
```

```
Deploy a DM cluster for production

Usage:
  tiup dm [flags]
  tiup dm [command]

Available Commands:
  deploy      Deploy a DM cluster for production
  start       Start a DM cluster
  stop        Stop a DM cluster
  restart     Restart a DM cluster
  list        List all clusters
  destroy     Destroy a specified DM cluster
  audit       Show audit log of cluster operation
  exec        Run shell command on host in the dm cluster
  edit-config Edit DM cluster config
  display     Display information of a DM cluster
  reload      Reload a DM cluster's config and restart if needed
  upgrade     Upgrade a specified DM cluster
  patch       Replace the remote package with a specified package and restart
     ↪   the service
  scale-out   Scale out a DM cluster
  scale-in    Scale in a DM cluster
  import      Import an exist DM 1.0 cluster from dm-ansible and re-deploy
     ↪  2.0 version
  help        Help about any command

Flags:
  -h, --help              help for tiup-dm
      --native-ssh        Use the native SSH client installed on local system
         ↪  instead of the build-in one.
      --ssh-timeout int   Timeout in seconds to connect host via SSH, ignored
         ↪  for operations that don't need an SSH connection. (default 5)
  -v, --version           version for tiup-dm
      --wait-timeout int  Timeout in seconds to wait for an operation to
         ↪  complete, ignored for operations that don't fit. (default 60)
  -y, --yes               Skip all confirmations and assumes 'yes'
```

4.1.1.1 View the cluster list

After the cluster is successfully deployed, view the cluster list by running the following command:

```
tiup dm list
```

```
Name User Version Path                                     PrivateKey
---- ---- ------- ----                                     ----------
prod-cluster tidb v2.0.3 /root/.tiup/storage/dm/clusters/test /root/.tiup/
    ↪ storage/dm/clusters/test/ssh/id_rsa
```

4.1.1.2 Start the cluster

After the cluster is successfully deployed, start the cluster by running the following command:

```
tiup dm start prod-cluster
```

If you forget the name of your cluster, view the cluster list by running `tiup dm list`.

4.1.1.3 Check the cluster status

TiUP provides the `tiup dm display` command to view the status of each component in the cluster. With this command, you do not have to log in to each machine to see the component status. The usage of the command is as follows:

```
tiup dm display prod-cluster
```

```
dm Cluster: prod-cluster
dm Version: v2.0.3
ID                Role         Host         Ports      OS/Arch      Status
    ↪ Data Dir                          Deploy Dir
--                ----         ----         -----      -------      ------
    ↪ --------                          ----------
172.19.0.101:9093 alertmanager 172.19.0.101 9093/9094 linux/x86_64 Up    /
    ↪ home/tidb/data/alertmanager-9093 /home/tidb/deploy/alertmanager-9093
172.19.0.101:8261 dm-master  172.19.0.101 8261/8291 linux/x86_64 Healthy|L /
    ↪ home/tidb/data/dm-master-8261 /home/tidb/deploy/dm-master-8261
172.19.0.102:8261 dm-master  172.19.0.102 8261/8291 linux/x86_64 Healthy /
    ↪ home/tidb/data/dm-master-8261 /home/tidb/deploy/dm-master-8261
172.19.0.103:8261 dm-master  172.19.0.103 8261/8291 linux/x86_64 Healthy /
    ↪ home/tidb/data/dm-master-8261 /home/tidb/deploy/dm-master-8261
172.19.0.101:8262 dm-worker  172.19.0.101 8262     linux/x86_64 Free     /
    ↪ home/tidb/data/dm-worker-8262 /home/tidb/deploy/dm-worker-8262
172.19.0.102:8262 dm-worker  172.19.0.102 8262     linux/x86_64 Free     /
    ↪ home/tidb/data/dm-worker-8262 /home/tidb/deploy/dm-worker-8262
172.19.0.103:8262 dm-worker  172.19.0.103 8262     linux/x86_64 Free     /
    ↪ home/tidb/data/dm-worker-8262 /home/tidb/deploy/dm-worker-8262
172.19.0.101:3000 grafana    172.19.0.101 3000     linux/x86_64 Up       -
    ↪                                /home/tidb/deploy/grafana-3000
```

```
172.19.0.101:9090 prometheus 172.19.0.101 9090     linux/x86_64 Up       /
    ↪ home/tidb/data/prometheus-9090 /home/tidb/deploy/prometheus-9090
```

The `Status` column uses `Up` or `Down` to indicate whether the service is running normally.

For the DM-master component, `|L` might be appended to a status, which indicates that the DM-master node is a Leader. For the DM-worker component, `Free` indicates that the current DM-worker node is not bound to an upstream.

4.1.1.4 Scale in a cluster

Scaling in a cluster means making some node(s) offline. This operation removes the specified node(s) from the cluster and deletes the remaining data files.

When you scale in a cluster, DM operations on DM-master and DM-worker components are performed in the following order:

1. Stop component processes.
2. Call the API for DM-master to delete the `member`.
3. Clean up the data files related to the node.

The basic usage of the scale-in command:

```
tiup dm scale-in <cluster-name> -N <node-id>
```

To use this command, you need to specify at least two arguments: the cluster name and the node ID. The node ID can be obtained by using the `tiup dm display` command in the previous section.

For example, to scale in the DM-worker node on `172.16.5.140` (similar to scaling in DM-master), run the following command:

```
tiup dm scale-in prod-cluster -N 172.16.5.140:8262
```

4.1.1.5 Scale out a cluster

The scale-out operation has an inner logic similar to that of deployment: the TiUP DM component first ensures the SSH connection of the node, creates the required directories on the target node, then executes the deployment operation, and starts the node service.

For example, to scale out a DM-worker node in the `prod-cluster` cluster, take the following steps (scaling out DM-master has similar steps):

1. Create a `scale.yaml` file and add information of the new worker node:

> **Note:**
>
> You need to create a topology file, which includes only the description of the new nodes, not the existing nodes. For more configuration items (such as the deployment directory), refer to this TiUP configuration parameter example.

```
---

worker_servers:
  - host: 172.16.5.140
```

2. Perform the scale-out operation. TiUP DM adds the corresponding nodes to the cluster according to the port, directory, and other information described in `scale.yaml`.

```
tiup dm scale-out prod-cluster scale.yaml
```

   After the command is executed, you can check the status of the scaled-out cluster by running `tiup dm display prod-cluster`.

4.1.1.6 Rolling upgrade

> **Note:**
>
> Since v2.0.5, dmctl support Export and Import Data Sources and Task Configuration of Clusters。
>
> Before upgrading, you can use `config export` to export the configuration files of clusters. After upgrading, if you need to downgrade to an earlier version, you can first redeploy the earlier cluster and then use `config import` to import the previous configuration files.
>
> For clusters earlier than v2.0.5, you can use dmctl v2.0.5 or later to export and import the data source and task configuration files.
>
> For clusters later than v2.0.2, currently, it is not supported to automatically import the configuration related to relay worker. You can use `start-relay` command to manually start relay log.

The rolling upgrade process is made as transparent as possible to the application, and does not affect the business. The operations vary with different nodes.

4.1.1.6.1 Upgrade command

You can run the `tiup dm upgrade` command to upgrade a DM cluster. For example, the following command upgrades the cluster to v2.0.1:

```
tiup dm upgrade prod-cluster v2.0.1
```

4.1.1.7 Update configuration

If you want to dynamically update the component configurations, the TiUP DM component saves a current configuration for each cluster. To edit this configuration, execute the `tiup dm edit-config <cluster-name>` command. For example:

```
tiup dm edit-config prod-cluster
```

TiUP DM opens the configuration file in the vi editor. If you want to use other editors, use the `EDITOR` environment variable to customize the editor, such as `export EDITOR=nano` ↪ . After editing the file, save the changes. To apply the new configuration to the cluster, execute the following command:

```
tiup dm reload prod-cluster
```

The command sends the configuration to the target machine and restarts the cluster to make the configuration take effect.

4.1.1.8 Update component

For normal upgrade, you can use the `upgrade` command. But in some scenarios, such as debugging, you might need to replace the currently running component with a temporary package. To achieve this, use the `patch` command:

```
tiup dm patch --help
```

```
Replace the remote package with a specified package and restart the service

Usage:
  tiup dm patch <cluster-name> <package-path> [flags]

Flags:
  -h, --help                 help for patch
  -N, --node strings         Specify the nodes
      --overwrite            Use this package in the future scale-out
        ↪ operations
  -R, --role strings         Specify the role
      --transfer-timeout int Timeout in seconds when transferring dm-master
        ↪ leaders (default 300)
```

```
Global Flags:
     --native-ssh      Use the native SSH client installed on local system
        ↪ instead of the build-in one.
     --ssh-timeout int Timeout in seconds to connect host via SSH, ignored
        ↪ for operations that don't need an SSH connection. (default 5)
     --wait-timeout int Timeout in seconds to wait for an operation to
        ↪ complete, ignored for operations that don't fit. (default 60)
  -y, --yes            Skip all confirmations and assumes 'yes'
```

If a DM-master hotfix package is in `/tmp/dm-master-hotfix.tar.gz` and you want to replace all the DM-master packages in the cluster, run the following command:

```
tiup dm patch prod-cluster /tmp/dm-master-hotfix.tar.gz -R dm-master
```

You can also replace only one DM-master package in the cluster:

```
tiup dm patch prod-cluster /tmp/dm--hotfix.tar.gz -N 172.16.4.5:8261
```

4.1.1.9 Import and upgrade a DM 1.0 cluster deployed using DM-Ansible

Note:

- TiUP does not support importing the DM Portal component in a DM 1.0 cluster.
- You need to stop the original cluster before importing.
- Don't run `stop-task` for tasks that need to be upgraded to 2.0.
- TiUP only supports importing to a DM cluster of v2.0.0-rc.2 or a later version.
- The `import` command is used to import data from a DM 1.0 cluster to a new DM 2.0 cluster. If you need to import DM migration tasks to an existing DM 2.0 cluster, refer to Manually Upgrade TiDB Data Migration from v1.0.x to v2.0.x.
- The deployment directories of some components are different from those of the original cluster. You can execute the `display` command to view the details.
- Run `tiup update --self && tiup update dm` before importing to make sure that the TiUP DM component is the latest version.
- Only one DM-master node exists in the cluster after importing. Refer to Scale out a cluster to scale out the DM-master.

Before TiUP is released, DM-Ansible is often used to deploy DM clusters. To enable TiUP to take over the DM 1.0 cluster deployed by DM-Ansible, use the `import` command.

For example, to import a cluster deployed using DM Ansible:

```
tiup dm import --dir=/path/to/dm-ansible --cluster-version v2.0.3
```

Execute `tiup list dm-master` to view the latest cluster version supported by TiUP.

The process of using the `import` command is as follows:

1. TiUP generates a topology file `topology.yml` based on the DM cluster previously deployed using DM-Ansible.
2. After confirming that the topology file has been generated, you can use it to deploy the DM cluster of v2.0 or later versions.

After the deployment is completed, you can execute the `tiup dm start` command to start the cluster and begin the process of upgrading the DM kernel.

4.1.1.10 View the operation log

To view the operation log, use the `audit` command. The usage of the `audit` command is as follows:

```
Usage:
  tiup dm audit [audit-id] [flags]

Flags:
  -h, --help help for audit
```

If the `[audit-id]` argument is not specified, the command shows a list of commands that have been executed. For example:

```
tiup dm audit
```

```
ID      Time                  Command
--      ----                  -------
4D5kQY 2020-08-13T05:38:19Z tiup dm display test
4D5kNv 2020-08-13T05:36:13Z tiup dm list
4D5kNr 2020-08-13T05:36:10Z tiup dm deploy -p prod-cluster v2.0.3 ./examples
    ↪ /dm/minimal.yaml
```

The first column is `audit-id`. To view the execution log of a certain command, pass the `audit-id` argument as follows:

```
tiup dm audit 4D5kQY
```

4.1.1.11 Run commands on a host in the DM cluster

To run commands on a host in the DM cluster, use the `exec` command. The usage of the `exec` command is as follows:

```
Usage:
  tiup dm exec <cluster-name> [flags]

Flags:
      --command string the command run on cluster host (default "ls")
  -h, --help           help for exec
  -N, --node strings   Only exec on host with specified nodes
  -R, --role strings   Only exec on host with specified roles
      --sudo           use root permissions (default false)
```

For example, to execute `ls /tmp` on all DM nodes, run the following command:

```
tiup dm exec prod-cluster --command='ls /tmp'
```

4.1.1.12 dmctl

TiUP integrates the DM cluster controller `dmctl`.

Run the following command to use dmctl:

```
tiup dmctl [args]
```

Specify the version of dmctl:

```
tiup dmctl:v2.0.3 [args]
```

The previous dmctl command to add a source is `dmctl --master-addr master1:8261 ↪ operate-source create /tmp/source1.yml`. After dmctl is integrated into TiUP, the command is:

```
tiup dmctl --master-addr master1:8261 operate-source create /tmp/source1.
    ↪ yml
```

4.1.1.13 Use the system's native SSH client to connect to cluster

All operations above performed on the cluster machine use the SSH client embedded in TiUP to connect to the cluster and execute commands. However, in some scenarios, you might also need to use the SSH client native to the control machine system to perform such cluster operations. For example:

- To use a SSH plug-in for authentication
- To use a customized SSH client

Then you can use the `--native-ssh` command-line flag to enable the system-native command-line tool:

- Deploy a cluster: `tiup dm deploy <cluster-name> <version> <topo> --native-ssh`
- Start a cluster: `tiup dm start <cluster-name> --native-ssh`
- Upgrade a cluster: `tiup dm upgrade ... --native-ssh`

You can add `--native-ssh` in all cluster operation commands above to use the system's native SSH client.

To avoid adding such a flag in every command, you can use the `TIUP_NATIVE_SSH` system variable to specify whether to use the local SSH client:

```
export TIUP_NATIVE_SSH=true
### or
export TIUP_NATIVE_SSH=1
### or
export TIUP_NATIVE_SSH=enable
```

If you specify this environment variable and `--native-ssh` at the same time, `--native-ssh` has higher priority.

> **Note:**
>
> During the process of cluster deployment, if you need to use a password for connection or `passphrase` is configured in the key file, you must ensure that `sshpass` is installed on the control machine; otherwise, a timeout error is reported.

4.1.2 Maintain DM Clusters Using dmctl

> **Note:**
>
> For DM clusters deployed using TiUP, you are recommended to directly use `tiup dmctl` to maintain the clusters.

dmctl is a command line tool used to maintain DM clusters. It supports both the interactive mode and the command mode.

4.1.2.1 Interactive mode

Enter the interactive mode to interact with DM-master:

> **Note:**
>
> The interactive mode does not support Bash features. For example, you need to directly pass string flags instead of passing them in quotes.

```
./dmctl --master-addr 172.16.30.14:8261
```

```
Welcome to dmctl
Release Version: v2.0.3
Git Commit Hash: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Git Branch: release-2.0
UTC Build Time: yyyy-mm-dd hh:mm:ss
Go Version: go version go1.13 linux/amd64

» help
DM control

Usage:
  dmctl [command]

Available Commands:
  check-task     Checks the configuration file of the task.
  config         Commands to import/export config.
  get-config     Gets the configuration.
  handle-error   `skip`/`replace`/`revert` the current error event or a
     ↪ specific binlog position (binlog-pos) event.
  help           Gets help about any command.
  list-member    Lists member information.
  offline-member Offlines member which has been closed.
  operate-leader `evict`/`cancel-evict` the leader.
  operate-schema `get`/`set`/`remove` the schema for an upstream table.
  operate-source `create`/`update`/`stop`/`show` upstream MySQL/MariaDB
     ↪ source.
  pause-relay    Pauses DM-worker's relay unit.
  pause-task     Pauses a specified running task.
  purge-relay    Purges relay log files of the DM-worker according to the
     ↪ specified filename.
  query-status   Queries task status.
  resume-relay   Resumes DM-worker's relay unit.
```

```
  resume-task   Resumes a specified paused task.
  show-ddl-locks Shows un-resolved DDL locks.
  start-task    Starts a task as defined in the configuration file.
  stop-task     Stops a specified task.
  unlock-ddl-lock Unlocks DDL lock forcefully.

Flags:
  -h, --help           Help for dmctl.
  -s, --source strings MySQL Source ID.

Use "dmctl [command] --help" for more information about a command.
```

4.1.2.2 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.
> - Starting from v2.0.4, DM supports reading the `-master-addr` parameter from the environment variable `DM_MASTER_ADDR`.

```bash
./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

export DM_MASTER_ADDR="172.16.30.14:8261"
./dmctl query-status
```

```
Available Commands:
  check-task          check-task <config-file> [--error count] [--warn count]
  config              commands to import/export config
  get-config          get-config <task | master | worker | source> <name> [--
    ↪ file filename]
  handle-error        handle-error <task-name | task-file> [-s source ...] [-
    ↪ b binlog-pos] <skip/replace/revert> [replace-sql1;replace-sql2;]
  list-member         list-member [--leader] [--master] [--worker] [--name
    ↪ master-name/worker-name ...]
```

```
  offline-member     offline-member <--master/--worker> <--name master-name/
    ↪ worker-name>
  operate-leader     operate-leader <operate-type>
  operate-schema     operate-schema <operate-type> <-s source ...> <task-
    ↪ name | task-file> <-d database> <-t table> [schema-file]
  operate-source     operate-source <operate-type> [config-file ...] [--
    ↪ print-sample-config]
  pause-relay        pause-relay <-s source ...>
  pause-task         pause-task [-s source ...] <task-name | task-file>
  purge-relay        purge-relay <-s source> <-f filename> [--sub-dir
    ↪ directory]
  query-status       query-status [-s source ...] [task-name | task-file]
    ↪ [--more]
  resume-relay       resume-relay <-s source ...>
  resume-task        resume-task [-s source ...] <task-name | task-file>
  show-ddl-locks     show-ddl-locks [-s source ...] [task-name | task-file]
  start-task         start-task [-s source ...] [--remove-meta] <config-file
    ↪ >
  stop-task          stop-task [-s source ...] <task-name | task-file>
  unlock-ddl-lock    unlock-ddl-lock <lock-ID>

Special Commands:
  --encrypt Encrypts plaintext to ciphertext.
  --decrypt Decrypts ciphertext to plaintext.

Global Options:
  --V Prints version and exit.
  --config Path to configuration file.
  --master-addr Master API server addr.
  --rpc-timeout RPC timeout, default is 10m.
  --ssl-ca Path of file that contains list of trusted SSL CAs for connection
    ↪ .
  --ssl-cert Path of file that contains X509 certificate in PEM format for
    ↪ connection.
  --ssl-key Path of file that contains X509 key in PEM format for connection
    ↪ .
```

4.2 Cluster Upgrade

4.2.1 Manually Upgrade TiDB Data Migration from v1.0.x to v2.0.x

This document introduces how to manually upgrade the TiDB DM tool from v1.0.x to v2.0.x. The main idea is to use the global checkpoint information in v1.0.x to start a new data migration task in the v2.0.x cluster.

For how to automatically upgrade the TiDB DM tool from v1.0.x to v2.0.x, refer to Using TiUP to automatically import the 1.0 cluster deployed by DM-Ansible.

> **Note:**
>
> - Currently, upgrading DM from v1.0.x to v2.0.x is not supported when the data migration task is in the process of full export or full import.
> - As the gRPC protocol used for interaction between the components of the DM cluster is updated greatly, you need to make sure that the DM components (including dmctl) use the same version before and after the upgrade.
> - Because the metadata storage of the DM cluster (such as checkpoint, shard DDL lock status and online DDL metadata, etc.) is updated greatly, the metadata of v1.0.x cannot be reused automatically in v2.0.x. So you need to make sure the following requirements are satisfied before performing the upgrade operation:
>   - All data migration tasks are not in the process of shard DDL coordination.
>   - All data migration tasks are not in the process of online DDL coordination.

The steps for manual upgrade are as follows.

4.2.1.1 Step 1: Prepare v2.0.x configuration file

The prepared configuration files of v2.0.x include the configuration files of the upstream database and the configuration files of the data migration task.

4.2.1.1.1 Upstream database configuration file

In v2.0.x, the upstream database configuration file is separated from the process configuration of the DM-worker, so you need to obtain the source configuration based on the v1.0.x DM-worker configuration.

> **Note:**
>
> If `enable-gtid` in the source configuration is enabled during the upgrade from v1.0.x to v2.0.x, you need to parse the binlog or relay log file to obtain the GTID sets corresponding to the binlog position.

Upgrade a v1.0.x cluster deployed by DM-Ansible

Assume that the v1.0.x DM cluster is deployed by DM-Ansible, and the following `dm_worker_servers` configuration is in the `inventory.ini` file:

```
[dm_master_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
```

Then you can convert it to the following two source configuration files:

```
### The source configuration corresponding to the original dm_worker1. For
    ↪ example, it is named as source1.yaml.
server-id: 101                              # Corresponds to the original `
    ↪ server_id`.
source-id: "mysql-replica-01"               # Corresponds to the original `
    ↪ source_id`.
from:
  host: "172.16.10.81"                      # Corresponds to the original `
      ↪ mysql_host`.
  port: 3306                                # Corresponds to the original `
      ↪ mysql_port`.
  user: "root"                              # Corresponds to the original `
      ↪ mysql_user`.
  password: "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=" # Corresponds to the original
      ↪  `mysql_password`.
```

```
### The source configuration corresponding to the original dm_worker2. For
    ↪ example, it is named as source2.yaml.
server-id: 102                              # Corresponds to the original `
    ↪ server_id`.
source-id: "mysql-replica-02"               # Corresponds to the original `
    ↪ source_id`.
from:
  host: "172.16.10.82"                      # Corresponds to the original `
      ↪ mysql_host`.
  port: 3306                                # Corresponds to the original `
      ↪ mysql_port`.
  user: "root"                              # Corresponds to the original `
      ↪ mysql_user`.
  password: "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=" # Corresponds to the original
      ↪  `mysql_password`.
```

Upgrade a v1.0.x cluster deployed by binary

Assume that the v1.0.x DM cluster is deployed by binary, and the corresponding DM-worker configuration is as follows:

```
log-level = "info"
log-file = "dm-worker.log"
worker-addr = ":8262"
server-id = 101
source-id = "mysql-replica-01"
flavor = "mysql"
[from]
host = "172.16.10.81"
user = "root"
password = "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU="
port = 3306
```

Then you can convert it to the following source configuration file:

```
server-id: 101                                # Corresponds to the original `
   ↪ server-id`.
source-id: "mysql-replica-01"                 # Corresponds to the original `
   ↪ source-id`.
flavor: "mysql"                               # Corresponds to the original `
   ↪ flavor`.
from:
  host: "172.16.10.81"                        # Corresponds to the original `
     ↪ from.host`.
  port: 3306                                  # Corresponds to the original `
     ↪ from.port`.
  user: "root"                                # Corresponds to the original `
     ↪ from.user`.
  password: "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=" # Corresponds to the original
     ↪  `from.password`.
```

4.2.1.1.2 Data migration task configuration file

For data migration task configuration guide, v2.0.x is basically compatible with v1.0.x. You can directly copy the configuration of v1.0.x.

4.2.1.2 Step 2: Deploy the v2.0.x cluster

Note:

Skip this step if you have other v2.0.x clusters available.

Use TiUP to deploy a new v2.0.x cluster according to the required number of nodes.

4.2.1.3 Step 3: Stop the v1.0.x cluster

If the original v1.0.x cluster is deployed by DM-Ansible, you need to use DM-Ansible to stop the v1.0.x cluster.

If the original v1.0.x cluster is deployed by binary, you can stop the DM-worker and DM-master processes directly.

4.2.1.4 Step 4: Upgrade data migration task

1. Use the `operate-source` command to load the upstream database source configuration from step 1 into the v2.0.x cluster.

2. In the downstream TiDB cluster, obtain the corresponding global checkpoint information from the incremental checkpoint table of the v1.0.x data migration task.

   - Assume that the v1.0.x data migration configuration does not specify `meta-schema` (or specify its value as the default `dm_meta`), and the corresponding task name is `task_v1`, the corresponding checkpoint information is in the `` `dm_meta`.`task_v1_syncer_checkpoint` `` table of the downstream TiDB.
   - Use the following SQL statements to obtain the global checkpoint information of all upstream database sources corresponding to the data migration task.

   ```
   > SELECT `id`, `binlog_name`, `binlog_pos` FROM `dm_meta`.`
       ↪ task_v1_syncer_checkpoint` WHERE `is_global`=1;
   +------------------+-------------------------+------------+
   | id               | binlog_name             | binlog_pos |
   +------------------+-------------------------+------------+
   | mysql-replica-01 | mysql-bin|000001.000123 | 15847      |
   | mysql-replica-02 | mysql-bin|000001.000456 | 10485      |
   +------------------+-------------------------+------------+
   ```

3. Update the v1.0.x data migration task configuration file to start a new v2.0.x data migration task.

   - If the data migration task configuration file of v1.0.x is `task_v1.yaml`, copy it and rename it to `task_v2.yaml`.
   - Make the following changes to `task_v2.yaml`:
     - Modify `name` to a new name, such as `task_v2`.
     - Change `task-mode` to `incremental`.
     - Set the starting point of incremental replication for each source according to the global checkpoint information obtained in step 2. For example:

```
mysql-instances:
  - source-id: "mysql-replica-01"   # Corresponds to the `id`
      ↪ of the checkpoint information.
    meta:
      binlog-name: "mysql-bin.000123" # Corresponds to the `
          ↪ binlog_name` in the checkpoint information,
          ↪ excluding the part of `|000001`.
      binlog-pos: 15847               # Corresponds to `
          ↪ binlog_pos` in the checkpoint information.

  - source-id: "mysql-replica-02"
    meta:
      binlog-name: "mysql-bin.000456"
      binlog-pos: 10485
```

Note:
If `enable-gtid` is enabled in the source configuration, currently you need to parse the binlog or relay log file to obtain the GTID sets corresponding to the binlog position, and set it to `binlog-gtid` in the `meta`.

4. Use the `start-task` command to start the upgraded data migration task through the v2.0.x data migration task configuration file.

5. Use the `query-status` command to confirm whether the data migration task is running normally.

If the data migration task runs normally, it indicates that the DM upgrade to v2.0.x is successful.

4.2.2 Upgrade TiDB Data Migration Between 1.0.x Versions

This document introduces how to upgrade a TiDB DM cluster from a lower 1.0.x version to a higher 1.0.x version, and the major changes and other information about the upgraded 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.
- 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.

4.2.2.1 Upgrade to v1.0.5

4.2.2.1.1 Version information

```
Release Version: v1.0.5
Git Commit Hash: a8e9f53f91e29756b09a22cdc37a6a6efcdfe55b
Git Branch: release-1.0
UTC Build Time: 2020-04-27 06:56:31
Go Version: go version go1.13 linux/amd64
```

4.2.2.1.2 Main changes

- 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
- Fix the issue that the `Duplicate entry` error might occur during the full data import
- Fix the issue that the `stop-task`/`pause-task` command may not work when no data written upstream after the full import is completed
- Fix the issue that the monitoring metrics still display data after the migration task is stopped

4.2.2.1.3 Upgrade operation example

1. Download the new version of DM-Ansible, and confirm that there is `dm_version =` ↪ `v1.0.5` 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.

4.2.2.2 Upgrade to v1.0.4

4.2.2.2.1 Version information

```
Release Version: v1.0.4-1-gd681c67
Git Commit Hash: d681c6731d3432f4d8f38ea651f44d49d6860269
Git Branch: release-1.0
UTC Build Time: 2020-03-16 09:45:29
Go Version: go version go1.13 linux/amd64
```

4.2.2.2.2 Main changes

- Add English UI for DM Portal
- Add the `--more` parameter in the `query-status` command to show complete migration status information
- 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

4.2.2.2.3 Upgrade operation example

1. Download the new version of DM-Ansible, and confirm that there is `dm_version = ↪ v1.0.4` 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.

4.2.2.3 Upgrade to v1.0.3

4.2.2.3.1 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
```

4.2.2.3.2 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

4.2.2.3.3 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.

4.2.2.4 Upgrade to v1.0.2

4.2.2.4.1 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
```

4.2.2.4.2 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

4.2.2.4.3 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.

4.2.2.5 Upgrade to v1.0.1

4.2.2.5.1 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
```

4.2.2.5.2 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

4.2.2.5.3 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.

4.2.2.6 Upgrade to v1.0.0-10-geb2889c9 (1.0 GA)

4.2.2.6.1 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
```

4.2.2.6.2 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

4.2.2.6.3 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.

4.2.2.7 Upgrade to v1.0.0-rc.1-12-gaa39ff9

4.2.2.7.1 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
```

4.2.2.7.2 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.

4.2.2.7.3 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`

4.3 Manage Data Source Configurations

This document introduces how to manage data source configurations, including encrypting the MySQL password, operating the data source, and changing the bindings between upstream MySQL instances and DM-workers using dmctl.

4.3.1 Encrypt the database password

In DM configuration files, it is recommended to use the password encrypted with dmctl. For one original password, the encrypted password is different after each encryption.

```
./dmctl -encrypt 'abc!@#123'
```

```
MKxn0Qo3m3XOyjCnhEMtsUCm83EhGQDZ/T4=
```

4.3.2 Operate data source

You can use the `operate-source` command to load, list or remove the data source configurations to the DM cluster.

```
help operate-source
```

```
`create`/`update`/`stop`/`show` upstream MySQL/MariaDB source.

Usage:
  dmctl operate-source <operate-type> [config-file ...] [--print-sample-
    ↪ config] [flags]
```

```
Flags:
  -h, --help                 help for operate-source
  -p, --print-sample-config print sample config file of source

Global Flags:
  -s, --source strings MySQL Source ID
```

4.3.2.1 Flags description

- `create`: Creates one or more upstream database source(s). When creating multiple data sources fails, DM rolls back to the state where the command was not executed.
- `update`: Updates an upstream database source.
- `stop`: Stops one or more upstream database source(s). When stopping multiple data sources fails, some data sources might be stopped.
- `show`: Shows the added data source and the corresponding DM-worker.
- `config-file`: Specifies the file path of `source.yaml` and can pass multiple file paths.
- `--print-sample-config`: Prints the sample configuration file. This parameter ignores other parameters.

4.3.2.2 Usage example

Use the following `operate-source` command to create a source configuration file:

```
operate-source create ./source.yaml
```

For the configuration of `source.yaml`, refer to Upstream Database Configuration File Introduction.

The following is an example of the returned result:

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "dm-worker-1"
        }
    ]
}
```

4.3.2.3 Check data source configurations

> **Note:**
>
> The `get-config` command is only supported in DM v2.0.1 and later versions.

If you know the `source-id`, you can run `dmctl --master-addr <master-addr> get-config source <source-id>` to get the data source configuration.

```
get-config source mysql-replica-01
```

```
{
  "result": true,
    "msg": "",
    "cfg": "enable-gtid: false
      flavor: mysql
      source-id: mysql-replica-01
      from:
        host: 127.0.0.1
        port: 8407
        user: root
        password: '******'
}
```

If you don't know the `source-id`, you can run `dmctl --master-addr <master-addr> operate-source show` to list all data sources first.

```
operate-source show
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "source is added but there is no free worker to bound",
            "source": "mysql-replica-02",
            "worker": ""
        },
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
```

```
            "worker": "dm-worker-1"
        }
    ]
}
```

4.3.3 Change the bindings between upstream MySQL instances and DM-workers

You can use the `transfer-source` command to change the bindings between upstream MySQL instances and DM-workers.

```
help transfer-source
```

```
Transfers an upstream MySQL/MariaDB source to a free worker.
Usage:
  dmctl transfer-source <source-id> <worker-id> [flags]
Flags:
  -h, --help help for transfer-source
Global Flags:
  -s, --source strings MySQL Source ID.
```

Before transferring, DM checks whether the worker to be unbound still has running tasks. If the worker has any running tasks, you need to pause the tasks first, change the binding, and then resume the tasks.

4.3.3.1 Usage example

If you do not know the bindings of DM-workers, you can run `dmctl --master-addr <master-addr> list-member --worker` to list the current bindings of all workers.

```
list-member --worker
```

```
{
    "result": true,
    "msg": "",
    "members": [
        {
            "worker": {
                "msg": "",
                "workers": [
                    {
                        "name": "dm-worker-1",
                        "addr": "127.0.0.1:8262",
                        "stage": "bound",
                        "source": "mysql-replica-01"
```

```
                },
                {
                    "name": "dm-worker-2",
                    "addr": "127.0.0.1:8263",
                    "stage": "free",
                    "source": ""
                }
            ]
        }
    }
  ]
}
```

In the above example, mysql-replica-01 is bound to dm-worker-1. The below command transfers the binding worker of mysql-replica-01 to dm-worker-2.

```
transfer-source mysql-replica-01 dm-worker-2
```

```
{
    "result": true,
    "msg": ""
}
```

Check whether the command takes effect by running `dmctl --master-addr <master-addr> list-member --worker`.

```
list-member --worker
```

```
{
    "result": true,
    "msg": "",
    "members": [
        {
            "worker": {
                "msg": "",
                "workers": [
                    {
                        "name": "dm-worker-1",
                        "addr": "127.0.0.1:8262",
                        "stage": "free",
                        "source": ""
                    },
                    {
                        "name": "dm-worker-2",
                        "addr": "127.0.0.1:8263",
                        "stage": "bound",
```

```
                        "source": "mysql-replica-01"
                    }
                ]
            }
        }
    ]
}
```

4.4 Manage a Data Migration Task

4.4.1 Data Migration Task Configuration Guide

This document introduces how to configure a data migration task in Data Migration (DM).

4.4.1.1 Configure data sources to be migrated

Before configuring the data sources to be migrated for the task, you need to first make sure that DM has loaded the configuration files of the corresponding data sources. The following are some operation references:

- To view the data source, you can refer to Check the data source configuration.
- To create a data source, you can refer to Create data source.
- To generate a data source configuration file, you can refer to Source configuration file introduction.

The following example of `mysql-instances` shows how to configure data sources that need to be migrated for the data migration task:

```
---

#### ********* Basic configuration *********
name: test          # The name of the task. Should be globally unique.

#### ******** Data source configuration **********
mysql-instances:
  - source-id: "mysql-replica-01" # Migrate data from the data source whose
    ↪ `source-id` is `mysql-replica-01`.
  - source-id: "mysql-replica-02" # Migrate data from the data source whose
    ↪ `source-id` is `mysql-replica-02`.
```

4.4.1.2 Configure the downstream TiDB cluster

The following example of `target-database` shows how to configure the target TiDB cluster to be migrated to for the data migration task:

```
---

#### ********* Basic configuration *********
name: test          # The name of the task. Should be globally unique.

#### ******** Data source configuration **********
mysql-instances:
  - source-id: "mysql-replica-01" # Migrate data from the data source whose
     ↪ `source-id` is `mysql-replica-01`.
  - source-id: "mysql-replica-02" # Migrate data from the data source whose
     ↪ `source-id` is `mysql-replica-02`.

#### ******** Downstream TiDB database configuration **********
target-database:    # Configuration of target TiDB database.
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: ""      # If the password is not null, it is recommended to use
     ↪  a password encrypted with dmctl.
```

4.4.1.3 Configure tables to be migrated

> **Note:**
>
> If you do not need to filter specific tables or migrate specific tables, skip this configuration.

To configure the block and allow list of data source tables for the data migration task, perform the following steps:

1. Configure a global filter rule set of the block and allow list in the task configuration file.

```
block-allow-list:
  bw-rule-1:                        # The name of the block and allow
     ↪ list rule.
```

```
    do-dbs: ["test.*", "user"]      # The allow list of upstream schemas
       ↪ to be migrated. Wildcard characters (*?) are supported. You
       ↪ only need to configure either `do-dbs` or `ignore-dbs`. If
       ↪ both fields are configured, only `do-dbs` takes effect.
    # ignore-dbs: ["mysql", "account"] # The block list of upstream
       ↪ schemas to be migrated. Wildcard characters (*?) are
       ↪ supported.
    do-tables:                      # The allow list of upstream tables
       ↪ to be migrated. You only need to configure either `do-tables`
       ↪  or `ignore-tables`. If both fields are configured, only `do-
       ↪ tables` takes effect.
    - db-name: "test.*"
      tbl-name: "t.*"
    - db-name: "user"
      tbl-name: "information"
  bw-rule-2:                       # The name of the block allow list
     ↪ rule.
    ignore-tables:                 # The block list of data source tables
       ↪  needs to be migrated.
    - db-name: "user"
      tbl-name: "log"
```

For detailed configuration rules, see Block and allow table lists.

2. Reference the block and allow list rules in the data source configuration to filter tables to be migrated.

```
mysql-instances:
  - source-id: "mysql-replica-01" # Migrate data from the data source
     ↪ whose `source-id` is `mysql-replica-01`.
    block-allow-list: "bw-rule-1" # The name of the block and allow
       ↪ list rule. If the DM version is earlier than v2.0.0-beta.2,
       ↪ use `black-white-list` instead.
  - source-id: "mysql-replica-02" # Migrate data from the data source
     ↪ whose `source-id` is `mysql-replica-02`.
    block-allow-list: "bw-rule-2" # The name of the block and allow
       ↪ list rule. If the DM version is earlier than v2.0.0-beta.2,
       ↪ use `black-white-list` instead.
```

4.4.1.4 Configure binlog events to be migrated

Note:

If you do not need to filter specific binlog events of certain schemas or tables, skip this configuration.

To configure the filters of binlog events for the data migration task, perform the following steps:

1. Configure a global filter rule set of binlog events in the task configuration file.

```
filters:                                      # The filter rule set of data
    ↪  source binlog events. You can set multiple rules at the same
    ↪ time.
  filter-rule-1:                              # The name of the filtering
      ↪ rule.
    schema-pattern: "test_*"                  # The pattern of the data
        ↪ source schema name. Wildcard characters (*?) are supported.
    table-pattern: "t_*"                      # The pattern of the data
        ↪ source table name. Wildcard characters (*?) are supported.
    events: ["truncate table", "drop table"] # The event types to be
        ↪ filtered out in schemas or tables that match the `schema-
        ↪ pattern` or the `table-pattern`.
    action: Ignore                            # Whether to migrate (Do) or
        ↪ ignore (Ignore) the binlog that matches the filtering rule.
  filter-rule-2:
    schema-pattern: "test"
    events: ["all dml"]
    action: Do
```

   For detailed configuration rules, see Binlog event filter.

2. Reference the binlog event filtering rules in the data source configuration to filter specified binlog events of specified tables or schemas in the data source.

```
mysql-instances:
  - source-id: "mysql-replica-01" # Migrate data from the data source
      ↪ whose `source-id` is `mysql-replica-01`.
    block-allow-list: "bw-rule-1" # The name of the block and allow
        ↪ list rule. If the DM version is earlier than v2.0.0-beta.2,
        ↪ use `black-white-list` instead.
    filter-rules: ["filter-rule-1"] # The name of the rule that filters
        ↪  specific binlog events of the data source. You can configure
        ↪  multiple rules here.
  - source-id: "mysql-replica-02" # Migrate data from the data source
      ↪ whose `source-id` is `mysql-replica-01`.
```

```
    block-allow-list: "bw-rule-2" # The name of the block and allow
        ↪ list rule. If the DM version is earlier than v2.0.0-beta.2,
        ↪ use `black-white-list` instead.
    filter-rules: ["filter-rule-2"] # The name of the rule that filters
        ↪  specific binlog events of the data source. You can configure
        ↪  multiple rules here.
```

4.4.1.5 Configure the mapping of data source tables to downstream TiDB tables

Note:

- If you do not need to migrate a certain table of the data source to the table with a different name in the downstream TiDB instance, skip this configuration.
- If it is a shard merge task, you **must** set mapping rules in the task configuration file.

To configure the routing mapping rules for migrating data source tables to specified downstream TiDB tables, perform the following steps:

1. Configure a global routing mapping rule set in the task configuration file.

```
routes:                         # The routing mapping rule set between
    ↪ the data source tables and downstream TiDB tables. You can set
    ↪ multiple rules at the same time.
  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 TiDB schema.
    target-table: "t"           # The name of the downstream TiDB table.
  route-rule-2:
    schema-pattern: "test_*"
    target-schema: "test"
```

   For detailed configuration rules, see Table Routing.

2. Reference the routing mapping rules in the data source configuration to filter tables to be migrated.

```
mysql-instances:
  - source-id: "mysql-replica-01"              # Migrate data from the
    ↪ data source whose `source-id` is `mysql-replica-01`.
    block-allow-list: "bw-rule-1"              # The name of the block
      ↪ and allow list rule. If the DM version is earlier than v2
      ↪ .0.0-beta.2, use `black-white-list` instead.
    filter-rules: ["filter-rule-1"]            # The name of the rule
      ↪ that filters specific binlog events of the data source. You
      ↪ can configure multiple rules here.
    route-rules: ["route-rule-1", "route-rule-2"] # The name of the
      ↪ routing mapping rule. You can configure multiple rules here.
  - source-id: "mysql-replica-02"              # Migrate data from the
    ↪ data source whose `source-id` is `mysql-replica-02`.
    block-allow-list: "bw-rule-2"              # The name of the block
      ↪ and allow list rule. If the DM version is earlier than v2
      ↪ .0.0-beta.2, use `black-white-list` instead.
    filter-rules: ["filter-rule-2"]            # The name of the rule
      ↪ that filters specific binlog events of the data source. You
      ↪ can configure multiple rules here.
```

4.4.1.6 Configure a shard merge task

Note:

- If you need to migrate sharding DDL statements in a shard merge scenario, you **must** explicitly configure the `shard-mode` field. Otherwise, **DO NOT** configure `shard-mode` at all.
- Migrating sharding DDL statements is likely to cause many issues. Make sure you understand the principles and restrictions of DM migrating DDL statements before using this feature, and you **must** use this feature with caution.

The following example shows how to configure the task as a shard merge task:

```
---

#### ********* Basic information *********
name: test                    # The name of the task. Should be globally
  ↪ unique.
```

```
shard-mode: "pessimistic"   # The shard merge mode. Optional modes are ""/"
    ↪ pessimistic"/"optimistic". The "" mode is used by default which means
    ↪  sharding DDL merge is disabled. If the task is a shard merge task,
    ↪ set it to the "pessimistic" mode. After getting a deep understanding
    ↪ of the principles and restrictions of the "optimistic" mode, you can
    ↪ set it to the "optimistic" mode.
```

4.4.1.7 Other configurations

The following is an overall task configuration example of this document. The complete task configuration template can be found in DM task configuration file full introduction. For the usage and configuration of other configuration items, refer to Features of Data Migration.

```
---

#### ********* Basic configuration *********
name: test                  # The name of the task. Should be globally
    ↪ unique.
shard-mode: "pessimistic"   # The shard merge mode. Optional modes are ""/"
    ↪ pessimistic"/"optimistic". The "" mode is used by default which means
    ↪  sharding DDL merge is disabled. If the task is a shard merge task,
    ↪ set it to the "pessimistic" mode. After getting a deep understanding
    ↪ of the principles and restrictions of the "optimistic" mode, you can
    ↪ set it to the "optimistic" mode.
task-mode: all              # The task mode. Can be set to `full`(only
    ↪ migrates full data)/`incremental`(replicates binlog synchronously)/`
    ↪ all` (replicates both full and incremental binlogs).

#### ******** Data source configuration **********
mysql-instances:
  - source-id: "mysql-replica-01"              # Migrate data from the data
      ↪ source whose `source-id` is `mysql-replica-01`.
    block-allow-list: "bw-rule-1"              # The name of the block and
        ↪ allow list rule. If the DM version is earlier than v2.0.0-beta.2,
        ↪ use `black-white-list` instead.
    filter-rules: ["filter-rule-1"]            # The name of the rule that
        ↪ filters specific binlog events of the data source. You can
        ↪ configure multiple rules here.
    route-rules: ["route-rule-1", "route-rule-2"] # The name of the routing
        ↪ mapping rule. You can configure multiple rules here.
  - source-id: "mysql-replica-02"              # Migrate data from the data
      ↪ source whose `source-id` is `mysql-replica-02`.
    block-allow-list: "bw-rule-2"              # The name of the block and
        ↪ allow list rule. If the DM version is earlier than v2.0.0-beta.2,
        ↪ use `black-white-list` instead.
```

```
    filter-rules: ["filter-rule-2"]           # The name of the rule that
       ↪ filters specific binlog events of the data source. You can
       ↪ configure multiple rules here.
    route-rules: ["route-rule-2"]             # The name of the routing
       ↪ mapping rule. You can configure multiple rules here.

#### ******** Downstream TiDB instance configuration **********
target-database:    # Configuration of the downstream database instance.
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: ""      # If the password is not null, it is recommended to use
     ↪  a password encrypted with dmctl.

#### ******** Feature configuration set **********
### The filter rule set of tables to be migrated from the upstream database
   ↪ instance. You can set multiple rules at the same time.
block-allow-list:                 # Use black-white-list if the DM version
   ↪ is earlier than v2.0.0-beta.2.
  bw-rule-1:                      # The name of the block and allow list
     ↪ rule.
    do-dbs: ["test.*", "user"]    # The allow list of upstream schemas to
       ↪ be migrated. Wildcard characters (*?) are supported. You only need
       ↪  to configure either `do-dbs` or `ignore-dbs`. If both fields are
       ↪ configured, only `do-dbs` takes effect.
    # ignore-dbs: ["mysql", "account"] # The block list of upstream schemas
       ↪ to be migrated. Wildcard characters (*?) are supported.
    do-tables:                    # The allow list of upstream tables to be
       ↪  migrated. You only need to configure either `do-tables` or `
       ↪ ignore-tables`. If both fields are configured, only `do-tables`
       ↪ takes effect.
    - db-name: "test.*"
      tbl-name: "t.*"
    - db-name: "user"
      tbl-name: "information"
  bw-rule-2:                     # The name of the block allow list rule.
    ignore-tables:               # The block list of data source tables
       ↪ needs to be migrated.
    - db-name: "user"
      tbl-name: "log"

### The filter rule set of data source binlog events.
filters:                                  # You can set multiple rules at
   ↪ the same time.
  filter-rule-1:                          # The name of the filtering rule.
```

```
    schema-pattern: "test_*"              # The pattern of the data source
       ↪ schema name. Wildcard characters (*?) are supported.
    table-pattern: "t_*"                  # The pattern of the data source
       ↪ table name. Wildcard characters (*?) are supported.
    events: ["truncate table", "drop table"] # The event types to be
       ↪ filtered out in schemas or tables that match the `schema-pattern`
       ↪ or the `table-pattern`.
    action: Ignore                        # Whether to migrate (Do) or
       ↪ ignore (Ignore) the binlog that matches the filtering rule.
  filter-rule-2:
    schema-pattern: "test"
    events: ["all dml"]
    action: Do

### The routing mapping rule set between the data source and target TiDB
   ↪ instance tables.
routes:                           # You can set multiple rules at the same time.
  route-rule-1:                   # The name of the routing mapping rule.
    schema-pattern: "test_*"      # The pattern of the data source schema name.
       ↪ Wildcard characters (*?) are supported.
    table-pattern: "t_*"          # The pattern of the data source table name.
       ↪ Wildcard characters (*?) are supported.
    target-schema: "test"         # The name of the downstream TiDB schema.
    target-table: "t"             # The name of the downstream TiDB table.
  route-rule-2:
    schema-pattern: "test_*"
    target-schema: "test"
```

4.4.2 Precheck the Upstream MySQL Instance Configurations

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.

4.4.2.1 Command

`check-task` allows you to precheck whether the upstream MySQL instance configuration satisfies the DM requirements.

4.4.2.2 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
    - MySQL version > 5.5
    - MariaDB version >= 10.1.2

> **Warning:**
> Support for MySQL 8.0 is an experimental feature of TiDB Data Migration v2.0. It is **NOT** recommended that you use it in a production environment.

- Database configuration
    - Whether `server_id` is configured
- 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.

4.4.2.2.1 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
server_id	Disables checking the upstream database server_id
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_shard_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

4.4.3 Create a Data Migration Task

You can use the `start-task` command to create a data migration task. When the data migration task is started, DM prechecks privileges and configurations.

```
help start-task
```

```
Starts a task as defined in the configuration file
Usage:
  dmctl start-task [-s source ...] [--remove-meta] <config-file> [flags]
Flags:
  -h, --help        Help for start-task
      --remove-meta Whether to remove task's metadata
Global Flags:
  -s, --source strings MySQL Source ID
```

4.4.3.1 Usage example

```
start-task [ -s "mysql-replica-01"] ./task.yaml
```

4.4.3.2 Flags description

- `-s`: (Optional) Specifies the MySQL source to execute `task.yaml`. If it is set, the command only starts the subtasks of the specified task on the MySQL source.
- `config-file`: (Required) Specifies the file path of `task.yaml`.
- `remove-meta`: (Optional) Specifies whether to remove the task's previous metadata when starting the task.

4.4.3.3 Returned results

```
start-task task.yaml
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

4.4.4 Query Status

This document introduces how to use the `query-status` command to query the task status, and the subtask status of DM.

4.4.4.1 Query result

```
» query-status
```

```
{
    "result": true,   # Whether the query is successful.
    "msg": "",        # Describes the reason for the unsuccessful query.
    "tasks": [        # Migration task list.
        {
            "taskName": "test",      # The task name.
            "taskStatus": "Running", # The status of the task.
            "sources": [             # The upstream MySQL list.
                "mysql-replica-01",
                "mysql-replica-02"
            ]
        },
        {
            "taskName": "test2",
            "taskStatus": "Paused",
            "sources": [
                "mysql-replica-01",
                "mysql-replica-02"
            ]
        }
    ]
}
```

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.

4.4.4.2 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 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`/`Paused`/`Stopped` 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`

4.4.4.3 Detailed query result

```
» query-status test
```

```
» query-status
{
    "result": true,   # Whether the query is successful.
    "msg": "",        # Describes the cause for the unsuccessful query.
    "sources": [                         # The upstream MySQL list.
        {
            "result": true,
            "msg": "",
            "sourceStatus": {                 # The information of the upstream
                ↪ MySQL databases.
                "source": "mysql-replica-01",
                "worker": "worker1",
                "result": null,
                "relayStatus": null
            },
            "subTaskStatus": [           # The information of all subtasks of
                ↪ upstream MySQL databases.
                {
                    "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
```

```
"syncerBinlogGtid": "",
    ↪                                                        # The binlog
    ↪ position replicated using GTID.
"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.
            }
        }
    ]
},
{
    "result": true,
```

```
        "msg": "",
        "sourceStatus": {
            "source": "mysql-replica-02",
            "worker": "worker2",
            "result": null,
            "relayStatus": null
        },
        "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.
                }
            }
        ]
    },
    {
        "result": true,
        "sourceStatus": {
            "source": "mysql-replica-03",
            "worker": "worker3",
            "result": null,
            "relayStatus": null
        },
        "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 %"
        }
      }
    ]
  }
 ]
}
```


For the status description and status switch relationship of "stage" of "subTaskStatus" of "sources", see the subtask status.

For operation details of "unresolvedDDLLockID" of "subTaskStatus" of "sources", see Handle Sharding DDL Locks Manually.

4.4.4.4 Subtask status

4.4.4.4.1 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.

4.4.4.4.2 Status switch diagram

```
                                  error occurs
                       New --------------------------------|
                        |                              |
                        |         resume-task          |
                        |  |----------------------------| |
                        |  |                        |  |
                        |  |                        |  |
                        v  v      error occurs      |  v
Finished <-------------- Running -----------------------> Paused
                        ^  |      or pause-task     |
                        |  |                        |
              start task | | stop task              |
                        |  |                        |
                        |  v      stop task         |
                      Stopped <-------------------------|
```

4.4.5 Pause a Data Migration Task

You can use the `pause-task` command to pause a data migration task.

`pause-task` differs from `stop-task` in that:

- `pause-task` only pauses a migration task. You can query the status information (retained in the memory) of the task using `query-status`. `stop-task` terminates a migration task and removes all information related to this task from the memory. This means you cannot use `query-status` to query the status information. `dm_meta` like "checkpoint" and data that have been migrated to the downstream are not removed.
- If `pause-task` is executed to pause the migration task, you cannot start a new task with the same name, neither can you get the relay log of the paused task removed, since this task does exist. If `stop-task` is executed to stop a task, you can start a new task with the same name, and you can get the relay log of the stopped task removed, since this task no longer exists.
- `pause-task` is usually used to pause a task for troubleshooting, while `stop-task` is to permanently remove a migration task, or to co-work with `start-task` to update the configuration information.

```
help pause-task
```

```
pause a specified running task

Usage:
 dmctl pause-task [-s source ...] <task-name | task-file> [flags]

Flags:
 -h, --help help for pause-task

Global Flags:
 -s, --source strings MySQL Source ID
```

4.4.5.1 Usage example

```
pause-task [-s "mysql-replica-01"] task-name
```

4.4.5.2 Flags description

- `-s`: (Optional) Specifies the MySQL source where you want to pause the subtasks of the migration task. If it is set, this command pauses only the subtasks on the specified MySQL source.
- `task-name| task-file`: (Required) Specifies the task name or task file path.

4.4.5.3 Returned results

```
pause-task test
```

```
{
    "op": "Pause",
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

4.4.6 Resume a Data Migration Task

You can use the `resume-task` command to resume a data migration task in the `Paused` ↪ state. This is generally used in scenarios where you want to manually resume a data migration task after handling the error that get the task paused.

```bash
help resume-task
```

```
resume a specified paused task

Usage:
 dmctl resume-task [-s source ...] <task-name | task-file> [flags]

Flags:
 -h, --help help for resume-task

Global Flags:
 -s, --source strings MySQL Source ID
```

4.4.6.1 Usage example

```bash
resume-task [-s "mysql-replica-01"] task-name
```

4.4.6.2 Flags description

- `-s`: (Optional) Specifies the MySQL source where you want to resume the subtask of the migration task. If it is set, the command resumes only the subtasks on the specified MySQL source.
- `task-name | task-file`: (Required) Specifies the task name or task file path.

4.4.6.3 Returned results

```
resume-task test
```


```
{
    "op": "Resume",
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```


4.4.7 Stop a 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 a Data Migration Task.

```
help stop-task
```

```
stop a specified task

Usage:
 dmctl stop-task [-s source ...] <task-name | task-file> [flags]

Flags:
 -h, --help help for stop-task

Global Flags:
 -s, --source strings MySQL Source ID
```

4.4.7.1 Usage example

```
stop-task [-s "mysql-replica-01"] task-name
```

4.4.7.2 Flags description

- `-s`: (Optional) Specifies the MySQL source where the subtasks of the migration task (that you want to stop) run. If it is set, only subtasks on the specified MySQL source are stopped.

- `task-name | task-file`: (Required) Specifies the task name or task file path.

4.4.7.3 Returned results

```
stop-task test
```

```
{
    "op": "Stop",
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

4.4.8 Export and Import Data Sources and Task Configuration of Clusters

`config` command is used to export and import data sources and task configuration of clusters.

> **Note:**
>
> For clusters earlier than v2.0.5, you can use dmctl v2.0.5 or later to export and import the data source and task configuration files.

```
» help config
Commands to import/export config
Usage:
  dmctl config [command]
Available Commands:
  export     Export the configurations of sources and tasks.
  import     Import the configurations of sources and tasks.
Flags:
  -h, --help help for config
Global Flags:
  -s, --source strings MySQL Source ID.
Use "dmctl config [command] --help" for more information about a command.
```

4.4.8.1 Export the data source and task configuration of clusters

You can use `export` command to export the data source and task configuration of clusters to specified files.

```
config export [--dir directory]
```

4.4.8.1.1 Parameter explanation

- `dir`:
  - optional
  - specifies the file path for exporting
  - the default value is `./configs`

4.4.8.1.2 Returned results

```
config export -d /tmp/configs
```

```
export configs to directory `/tmp/configs` succeed
```

4.4.8.2 import the data source and task configuration of clusters

You can use `import` command to import the data source and task configuration of clusters from specified files.

```
config import [--dir directory]
```

> **Note:**
>
> For clusters later than v2.0.2, currently, it is not supported to automatically import the configuration related to relay worker. You can use `start-relay` command to manually start relay log.

4.4.8.2.1 Parameter explanation

- `dir`:
  - optional
  - specifies the file path for importing
  - the default value is `./configs`

4.4.8.2.2 Returned results

```
config import -d /tmp/configs
```

```
start creating sources
start creating tasks
import configs from directory `/tmp/configs` succeed
```

4.4.9 Handle Failed DDL Statements

This document introduces how to handle failed DDL statements when you're using the TiDB Data Migration (DM) tool to migrate data.

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 DDL statement, an error might occur and break the migration process. In this case, you can use the `handle-error` command of DM to resume the migration.

4.4.9.1 Restrictions

If it is unacceptable in the actual production environment that the failed DDL statement is skipped in the downstream TiDB and it cannot be replaced with other DDL statements, then do not use this command.

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.

4.4.9.2 Supported scenarios

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 `handle-error <task-name> skip` to skip migrating this DDL statement and resume the migration.
- If it is acceptable that this DDL statement is replaced with other DDL statements, then you can use `handle-error <task-name> replace` to replace this DDL statement and resume the migration.

4.4.9.3 Command

When you use dmctl to manually handle the failed DDL statements, the commonly used commands include `query-status` and `handle-error`.

4.4.9.3.1 query-status

The `query-status` command is used to query the current status of items such as the subtask and the relay unit in each MySQL instance. For details, see query status.

4.4.9.3.2 handle-error

The `handle-error` command is used to handle the failed DDL statements.

4.4.9.3.3 Command usage

```
» handle-error -h
```

```
Usage:
  dmctl handle-error <task-name | task-file> [-s source ...] [-b binlog-pos]
    ↪  <skip/replace/revert> [replace-sql1;replace-sql2;] [flags]

Flags:
  -b, --binlog-pos string position used to match binlog event if matched the
    ↪  handler-error operation will be applied. The format like "mysql-bin
    ↪ |000001.000003:3270"
  -h, --help             help for handle-error

Global Flags:
  -s, --source strings MySQL Source ID
```

4.4.9.3.4 Flags descriptions

- `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.
- `source`:
  - Flag parameter, string, `--source`
  - `source` specifies the MySQL instance in which the preset operation is to be executed.
- `skip`: Skip the error
- `replace`: Replace the failed DDL statement
- `revert`: Reset the previous skip/replace operation before the error occurs (only reset it when the previous skip/replace operation has not finally taken effect)

- binlog-pos:
  - Flag parameter, string, `--binlog-pos`
  - If it is not specified, DM automatically handles the currently failed DDL statement.
  - 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`.
  - After the migration returns an error, the binlog position can be obtained from `position` in `startLocation` returned by `query-status`. Before the migration returns an error, the binlog position can be obtained by using `SHOW BINLOG EVENTS` in the upstream MySQL instance.

4.4.9.4 Usage examples

4.4.9.4.1 Skip DDL if the migration gets interrupted

Non-shard-merge scenario

Assume that you need to migrate the upstream table `db1.tbl1` to the downstream TiDB. The initial table schema is:

```sql
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)):

```sql
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. Execute the `query-status <task-name>` command, and you can see the following error:

```
ERROR 8200 (HY000): Unsupported modify column: can't change decimal column
    ↪ precision
```

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 `handle-error <task-name> skip` to skip this DDL statement to resume the migration. The procedures are as follows:

1. Execute `handle-error <task-name> skip` to skip the currently failed DDL statement:

   ```
   » handle-error test skip
   ```

   ```
   {
       "result": true,
       "msg": "",
       "sources": [
           {
               "result": true,
               "msg": "",
               "source": "mysql-replica-01",
               "worker": "worker1"
           }
       ]
   }
   ```

2. Execute `query-status <task-name>` to view the task status:

   ```
   » query-status test
   ```

   See the execution result.

   ```
   {
       "result": true,
       "msg": "",
       "sources": [
           {
               "result": true,
               "msg": "",
               "sourceStatus": {
                   "source": "mysql-replica-01",
                   "worker": "worker1",
                   "result": null,
                   "relayStatus": null
               },
               "subTaskStatus": [
                   {
                       "name": "test",
                       "stage": "Running",
   ```

```
                "unit": "Sync",
                "result": null,
                "unresolvedDDLLockID": "",
                "sync": {
                    "totalEvents": "4",
                    "totalTps": "0",
                    "recentTps": "0",
                    "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                        ↪ 2388)",
                    "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                        ↪ de45f57:1-10",
                    "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                        ↪ 2388)",
                    "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                        ↪ de45f57:1-4",
                    "blockingDDLs": [
                    ],
                    "unresolvedGroups": [
                    ],
                    "synced": true,
                    "binlogType": "remote"
                }
            }
        ]
    }
  ]
}
```

You can see that the task runs normally and the wrong DDL is skipped.

Shard merge 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. The task mode is "pessimistic".

- MySQL instance 1 contains the `shard_db_1` schema, which includes the `shard_table_1` ↪ and `shard_table_2` tables.
- MySQL instance 2 contains the `shard_db_2` schema, which includes the `shard_table_1` ↪ and `shard_table_2` tables.

The initial table schema is:

```
SHOW CREATE TABLE shard_db.shard_table;
```

```
+--
    ↪ -----+------------------------------------------------------------
    ↪
| Table | Create Table
    ↪
    ↪ |
+--
    ↪ -----+------------------------------------------------------------
    ↪
| tb    | CREATE TABLE `shard_table` (
  `id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin |
+--
    ↪ -----+------------------------------------------------------------
    ↪
```

Now, execute the following DDL statement to all upstream sharded tables to alter their character set:

```
ALTER TABLE `shard_db_*`.`shard_table_*` CHARACTER SET LATIN1 COLLATE
    ↪ LATIN1_DANISH_CI;
```

Because this DDL statement is not supported by TiDB, the migration task of DM gets interrupted. Execute the `query-status` command, and you can see the following errors reported by the `shard_db_1.shard_table_1` table in MySQL instance 1 and the `shard_db_2.shard_table_1` table in MySQL instance 2:

```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_1`.`shard_table_1`
        ↪ CHARACTER SET UTF8 COLLATE UTF8_UNICODE_CI",
    "RawCause": "[ddl:8200]Unsupported modify charset from latin1 to utf8"
}
```

```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_2`.`shard_table_1`
        ↪ CHARACTER SET UTF8 COLLATE UTF8_UNICODE_CI",
    "RawCause": "[ddl:8200]Unsupported modify charset from latin1 to utf8"
}
```

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 `handle-error <task-name> skip` to skip this DDL statement to resume the migration. The procedures are as follows:

1. Execute `handle-error <task-name> skip` to skip the currently failed DDL statements in MySQL instance 1 and 2:

```
» handle-error test skip
```


```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        },
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-02",
            "worker": "worker2"
        }
    ]
}
```


2. Execute the `query-status` command, and you can see the errors reported by the `shard_db_1.shard_table_2` table in MySQL instance 1 and the `shard_db_2` ↪ `.shard_table_2` table in MySQL instance 2:


```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_1`.`
        ↪ shard_table_2` CHARACTER SET UTF8 COLLATE UTF8_UNICODE_CI",
    "RawCause": "[ddl:8200]Unsupported modify charset from latin1 to
        ↪ utf8"
}
```

```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_2`.`
        ↪ shard_table_2` CHARACTER SET UTF8 COLLATE UTF8_UNICODE_CI",
    "RawCause": "[ddl:8200]Unsupported modify charset from latin1 to
        ↪ utf8"
}
```


3. Execute `handle-error <task-name> skip` again to skip the currently failed DDL statements in MySQL instance 1 and 2:

```
» handle-error test skip
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        },
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-02",
            "worker": "worker2"
        }
    ]
}
```

4. Use `query-status <task-name>` to view the task status:

```
» query-status test
```

See the execution result.

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "sourceStatus": {
                "source": "mysql-replica-01",
                "worker": "worker1",
                "result": null,
                "relayStatus": null
            },
            "subTaskStatus": [
                {
                    "name": "test",
                    "stage": "Running",
                    "unit": "Sync",
                    "result": null,
                    "unresolvedDDLLockID": "",
```

```
                "sync": {
                    "totalEvents": "4",
                    "totalTps": "0",
                    "recentTps": "0",
                    "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                      ↪ 2388)",
                    "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                      ↪ de45f57:1-10",
                    "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                      ↪ 2388)",
                    "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                      ↪ de45f57:1-4",
                    "blockingDDLs": [
                    ],
                    "unresolvedGroups": [
                    ],
                    "synced": true,
                    "binlogType": "remote"
                }
            }
        ]
    },
    {
        "result": true,
        "msg": "",
        "sourceStatus": {
            "source": "mysql-replica-02",
            "worker": "worker2",
            "result": null,
            "relayStatus": null
        },
        "subTaskStatus": [
            {
                "name": "test",
                "stage": "Running",
                "unit": "Sync",
                "result": null,
                "unresolvedDDLLockID": "",
                "sync": {
                    "totalEvents": "4",
                    "totalTps": "0",
                    "recentTps": "0",
                    "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                      ↪ 2388)",
                    "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
```

```
                    ↪ de45f57:1-10",
                "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                    ↪ 2388)",
                "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                    ↪ de45f57:1-4",
                "blockingDDLs": [
                ],
                "unresolvedGroups": [
                ],
                "synced": true,
                "binlogType": "remote"
            }
        }
    ]
  }
 ]
}
```

You can see that the task runs normally with no error and all four wrong DDL statements are skipped.

4.4.9.4.2 Replace DDL if the migration gets interrupted

Non-shard-merge scenario

Assume that you need to migrate the upstream table `db1.tbl1` to the downstream TiDB. The initial table schema is:

```sql
SHOW CREATE TABLE db1.tbl1;
```

```
+--
    ↪ -----+------------------------------------------------------------------
    ↪
| Table | Create Table
    ↪
    ↪ |
+--
    ↪ -----+------------------------------------------------------------------
    ↪
| tb    | CREATE TABLE `tbl1` (
  `id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin |
+--
    ↪ -----+------------------------------------------------------------------
    ↪
```

Now, perform the following DDL operation in the upstream to add a new column with the UNIQUE constraint:

```sql
ALTER TABLE `db1`.`tbl1` ADD COLUMN new_col INT UNIQUE;
```

Because this DDL statement is not supported by TiDB, the migration task gets interrupted. Execute the `query-status` command, and you can see the following error:

```
{
    "Message": "cannot track DDL: ALTER TABLE `db1`.`tbl1` ADD COLUMN `
        ↪ new_col` INT UNIQUE KEY",
    "RawCause": "[ddl:8200]unsupported add column 'new_col' constraint
        ↪ UNIQUE KEY when altering 'db1.tbl1'",
}
```

You can replace this DDL statement with two equivalent DDL statements. The steps are as follows:

1. Replace the wrong DDL statement by the following command:

   ```
   » handle-error test replace "ALTER TABLE `db1`.`tbl1` ADD COLUMN `
       ↪ new_col` INT;ALTER TABLE `db1`.`tbl1` ADD UNIQUE(`new_col`)";
   ```

   ```
   {
       "result": true,
       "msg": "",
       "sources": [
           {
               "result": true,
               "msg": "",
               "source": "mysql-replica-01",
               "worker": "worker1"
           }
       ]
   }
   ```

2. Use `query-status <task-name>` to view the task status:

   ```
   » query-status test
   ```

   See the execution result.

   ```
   {
       "result": true,
       "msg": "",
       "sources": [
           {
   ```

```
            "result": true,
            "msg": "",
            "sourceStatus": {
                "source": "mysql-replica-01",
                "worker": "worker1",
                "result": null,
                "relayStatus": null
            },
            "subTaskStatus": [
                {
                    "name": "test",
                    "stage": "Running",
                    "unit": "Sync",
                    "result": null,
                    "unresolvedDDLLockID": "",
                    "sync": {
                        "totalEvents": "4",
                        "totalTps": "0",
                        "recentTps": "0",
                        "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
                        "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                            ↪ de45f57:1-10",
                        "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
                        "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                            ↪ de45f57:1-4",
                        "blockingDDLs": [
                        ],
                        "unresolvedGroups": [
                        ],
                        "synced": true,
                        "binlogType": "remote"
                    }
                }
            ]
        }
    ]
}
```

You can see that the task runs normally and the wrong DDL statement is replaced by new DDL statements that execute successfully.

Shard merge 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. The task mode is "pessimistic".

- 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:

```
SHOW CREATE TABLE shard_db.shard_table;
```

```
+--
    ↪ -----+--------------------------------------------------------------------
    ↪
| Table | Create Table
    ↪
    ↪ |
+--
    ↪ -----+--------------------------------------------------------------------
    ↪
| tb    | CREATE TABLE `shard_table` (
  `id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin |
+--
    ↪ -----+--------------------------------------------------------------------
    ↪
```

Now, perform the following DDL operation to all upstream sharded tables to add a new column with the UNIQUE constraint:

```
ALTER TABLE `shard_db_*`.`shard_table_*` ADD COLUMN new_col INT UNIQUE;
```

Because this DDL statement is not supported by TiDB, the migration task gets interrupted. Execute the `query-status` command, and you can see the following errors reported by the `shard_db_1.shard_table_1` table in MySQL instance 1 and the `shard_db_2` ↪ `.shard_table_1` table in MySQL instance 2:

```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_1`.`shard_table_1`
        ↪ ADD COLUMN `new_col` INT UNIQUE KEY",
    "RawCause": "[ddl:8200]unsupported add column 'new_col' constraint
        ↪ UNIQUE KEY when altering 'shard_db_1.shard_table_1'",
}
```

```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_2`.`shard_table_1`
       ↪ ADD COLUMN `new_col` INT UNIQUE KEY",
    "RawCause": "[ddl:8200]unsupported add column 'new_col' constraint
       ↪ UNIQUE KEY when altering 'shard_db_2.shard_table_1'",
}
```

You can replace this DDL statement with two equivalent DDL statements. The steps are as follows:

1. Replace the wrong DDL statements respectively in MySQL instance 1 and MySQL instance 2 by the following commands:

```
» handle-error test -s mysql-replica-01 replace "ALTER TABLE `
   ↪ shard_db_1`.`shard_table_1` ADD COLUMN `new_col` INT;ALTER TABLE
   ↪ `shard_db_1`.`shard_table_1` ADD UNIQUE(`new_col`)";
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

```
» handle-error test -s mysql-replica-02 replace "ALTER TABLE `
   ↪ shard_db_2`.`shard_table_1` ADD COLUMN `new_col` INT;ALTER TABLE
   ↪ `shard_db_2`.`shard_table_1` ADD UNIQUE(`new_col`)";
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-02",
            "worker": "worker2"
        }
```

```
    ]
}
```

2. Use `query-status <task-name>` to view the task status, and you can see the following errors reported by the `shard_db_1.shard_table_2` table in MySQL instance 1 and the `shard_db_2.shard_table_2` table in MySQL instance 2:

```
{
    "Message": "detect inconsistent DDL sequence from source ... ddls:
        ↪ [ALTER TABLE `shard_db`.`tb` ADD COLUMN `new_col` INT UNIQUE
        ↪ KEY] source: `shard_db_1`.`shard_table_2`], right DDL
        ↪ sequence should be ..."
}
```

```
{
    "Message": "detect inconsistent DDL sequence from source ... ddls:
        ↪ [ALTER TABLE `shard_db`.`tb` ADD COLUMN `new_col` INT UNIQUE
        ↪ KEY] source: `shard_db_2`.`shard_table_2`], right DDL
        ↪ sequence should be ..."
}
```

3. Execute `handle-error <task-name> replace` again to replace the wrong DDL statements in MySQL instance 1 and 2:

```
» handle-error test -s mysql-replica-01 replace "ALTER TABLE `
    ↪ shard_db_1`.`shard_table_2` ADD COLUMN `new_col` INT;ALTER TABLE
    ↪ `shard_db_1`.`shard_table_2` ADD UNIQUE(`new_col`)";
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

```
» handle-error test -s mysql-replica-02 replace "ALTER TABLE `
    ↪ shard_db_2`.`shard_table_2` ADD COLUMN `new_col` INT;ALTER TABLE
    ↪ `shard_db_2`.`shard_table_2` ADD UNIQUE(`new_col`)";
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-02",
            "worker": "worker2"
        }
    ]
}
```

4. Use `query-status <task-name>` to view the task status:

```
» query-status test
```

See the execution result.

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "sourceStatus": {
                "source": "mysql-replica-01",
                "worker": "worker1",
                "result": null,
                "relayStatus": null
            },
            "subTaskStatus": [
                {
                    "name": "test",
                    "stage": "Running",
                    "unit": "Sync",
                    "result": null,
                    "unresolvedDDLLockID": "",
                    "sync": {
                        "totalEvents": "4",
                        "totalTps": "0",
                        "recentTps": "0",
                        "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                        ↪ 2388)",
```

```
                "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                    ↪ de45f57:1-10",
                "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                    ↪ 2388)",
                "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                    ↪ de45f57:1-4",
                "blockingDDLs": [
                ],
                "unresolvedGroups": [
                ],
                "unresolvedGroups": [
                ],
                "synced": true,
                "binlogType": "remote"
            }
        }
    ]
},
{
    "result": true,
    "msg": "",
    "sourceStatus": {
        "source": "mysql-replica-02",
        "worker": "worker2",
        "result": null,
        "relayStatus": null
    },
    "subTaskStatus": [
        {
            "name": "test",
            "stage": "Running",
            "unit": "Sync",
            "result": null,
            "unresolvedDDLLockID": "",
            "sync": {
                "totalEvents": "4",
                "totalTps": "0",
                "recentTps": "0",
                "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                    ↪ 2388)",
                "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                    ↪ de45f57:1-10",
                "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                    ↪ 2388)",
                "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
```

```
                        ↪ de45f57:1-4",
                    "blockingDDLs": [
                    ],
                    "unresolvedGroups": [
                    ],
                    "unresolvedGroups": [
                    ],
                    "synced": try,
                    "binlogType": "remote"
                }
            }
        ]
    }
  ]
}
```

You can see that the task runs normally with no error and all four wrong DDL statements are replaced.

4.5 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` command to manually handle the abnormal DDL locks in some abnormal scenarios.

Note:

- This document only applies to the processing of sharding DDL lock in pessimistic coordination mode.
- The commands in the Command usage sections in this document are in interactive mode. In command-line mode, you need to add the escape characters to avoid an error report.
- 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.

4.5.1 Command

4.5.1.1 show-ddl-locks

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

4.5.1.1.1 Command usage

```
show-ddl-locks [--source=mysql-replica-01] [task-name | task-file]
```

4.5.1.1.2 Arguments description

- `source`:
  - Flag; string; `--source`; optional
  - It can be specified repeatedly multiple times.
  - If it is not specified, this command queries the lock information related to all MySQL sources; if it is specified, this command queries the lock information related only to the specified MySQL source.
- `task-name | task-file`:
  - 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.

4.5.1.1.3 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 existing lock
        ↪ information list.
        {
            "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.
```

```
            "mode": "pessimistic"                    # The shard DDL mode.
                ↪ Can be set to "pessimistic" or "optimistic".
            "owner": "mysql-replica-01",             # The owner of the lock
                ↪  (the ID of the first source that encounters this DDL
                ↪ operation in the pessimistic mode), which is always empty
                ↪ in the optimistic mode.
            "DDLs": [                                # The list of DDL
                ↪ operations corresponding to the lock in the pessimistic
                ↪ mode, which is always empty in the optimistic mode.
                "USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` DROP
                    ↪ COLUMN `c2`;"
            ],
            "synced": [                              # The list of sources
                ↪ that have received all sharding DDL events in the
                ↪ corresponding MySQL instance.
                "mysql-replica-01"
            ],
            "unsynced": [                            # The list of sources
                ↪ that have not yet received all sharding DDL events in the
                ↪ corresponding MySQL instance.
                "mysql-replica-02"
            ]
        }
    ]
}
```

4.5.1.2 `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`.

> **Note:**
>
> Currently, `unlock DDL lock` takes effect only for the lock in the `pessimistic` mode.

4.5.1.2.1 Command usage

```
unlock-ddl-lock [--owner] [--force-remove] <lock-ID>
```

4.5.1.2.2 Arguments description

- `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 MySQL source (the alternative of the default owner) to execute the DDL statement.
  - The new owner should not be specified unless the original owner is already removed from the cluster.
- `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`).

4.5.1.2.3 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.
}
```

4.5.2 Supported scenarios

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

4.5.2.1 Scenario 1: Some MySQL sources are removed

4.5.2.1.1 The reason for the abnormal lock

Before DM-master tries to automatically unlock the sharding DDL lock, all the MySQL sources 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 MySQL sources have been removed and are not to be reloaded (these MySQL sources 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.

4.5.2.1.2 Manual solution

Suppose that there are two instances MySQL-1 (mysql-replica-01) and MySQL-2 (mysql-replica-02) 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:

```
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 mysql-↪ replica-01 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 sends the received DDL information of the two sharded tables of mysql-↪ replica-01 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",
            "mode": "pessimistic"
            "owner": "mysql-replica-01",
            "DDLs": [
                "USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` ADD
                    ↪  COLUMN `c2` int(11);"
            ],
            "synced": [
                "mysql-replica-01"
            ],
            "unsynced": [
                "mysql-replica-02"
            ]
        }
    ]
}
```

4. Due to the application demand, the data corresponding to mysql-replica-02 is no longer needed to be migrated to the downstream TiDB, and mysql-replica-02 is removed.

5. The lock whose ID is test-`shard_db`.`shard_table` on DM-master cannot receive the DDL information of mysql-replica-02.

    - The returned result unsynced by show-ddl-locks has always included the information of mysql-replica-02.

6. Use `unlock-dll-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 MySQL source reports an error, `result` will be set to `false`, and at this point you should check carefully if the errors of each MySQL source is acceptable and within expectations.

    ```
    unlock-ddl-lock test-`shard_db`.`shard_table`
    ```

    ```
    {
        "result": true,
        "msg": ""
    ```

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.

4.5.2.1.3 Impact

After you have manually unlocked the lock by using `unlock-ddl-lock`, if you don't deal with the offline MySQL sources 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 MySQL source 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 MySQL source that went offline is reloaded and the DM-worker tries to migrate the data of the sharded tables, a match error between the data and the downstream table structure might occur.

4.5.2.2 Scenario 2: Some DM-workers stop abnormally or the network failure occurs during the DDL unlocking process

4.5.2.2.1 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 non-owners to skip the DDL and update the checkpoints of corresponding sharded tables after the owner successfully executes the DDL.
4. DM-master removes the corresponding DDL lock information after all the owners or non-owners' operations are successful.

Currently, the above unlocking process is not atomic. If the non-owner skips the DDL operation successfully, the DM-worker where the non-owner is located stops abnormally or a network anomaly occurs with the downstream TiDB, which can cause the checkpoint updating to fail.

When the MySQL source corresponding to the non-owner restores data migration, the non-owner tries to request the DM-master to re-coordinate the DDL operation that has been coordinated before the exception occurs and will never receives the corresponding DDL operation from other MySQL sources. This can cause the DDL operation to automatically unlock the corresponding lock.

4.5.2.2.2 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 MySQL sources are removed.

When `DM-master` automatically executes the unlocking process, the owner (`mysql-↪ replica-01`) successfully executes the DDL and continues the migration process. However, in the process of requesting the non-owner (`mysql-replica-02`) to skip the DDL operation, the checkpoint fails to update after the DM-worker skips the DDL operation because the corresponding DM-worker was restarted.

After the data migration subtask corresponding to `mysql-replica-02` restores, a new lock is created on the DM-master, but other MySQL sources have executed or skipped DDL operations and are performing subsequent migration.

The operation processes are:

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

   Only `mysql-replica-02` is at the `synced` state.

```
» show-ddl-locks
{
    "result": true,
    "msg": "",
    "locks": [
        {
            "ID": "test-`shard_db`.`shard_table`",
            "task": "test",
            "mode": "pessimistic"
            "owner": "mysql-replica-02",
            "DDLs": [
                "USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` ADD
                    ↪  COLUMN `c2` int(11);"
            ],
            "synced": [
                "mysql-replica-02"
            ],
            "unsynced": [
                "mysql-replica-01"
            ]
        }
    ]
}
```

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

- During the unlocking process, the owner tries to execute the DDL operation to the downstream again (the original owner before restarting has executed the DDL operation to the downstream once). Make sure that the DDL operation can be executed multiple times.

```
unlock-ddl-lock test-`shard_db`.`shard_table`
{
    "result": true,
    "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.

4.5.2.2.3 Impact

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

4.6 Manage Table Schemas of Tables to be Migrated

This document describes how to manage the schema of the table in DM during migration using dmctl.

4.6.1 Implementation principles

When you migrate tables using DM, DM performs the following operations on the table schema:

- For full export and import, DM directly exports the upstream table schema of the current time to SQL files and applies the table schema to the downstream.

- For incremental replication, the whole data link contains the following table schemas, which might be the same or different:

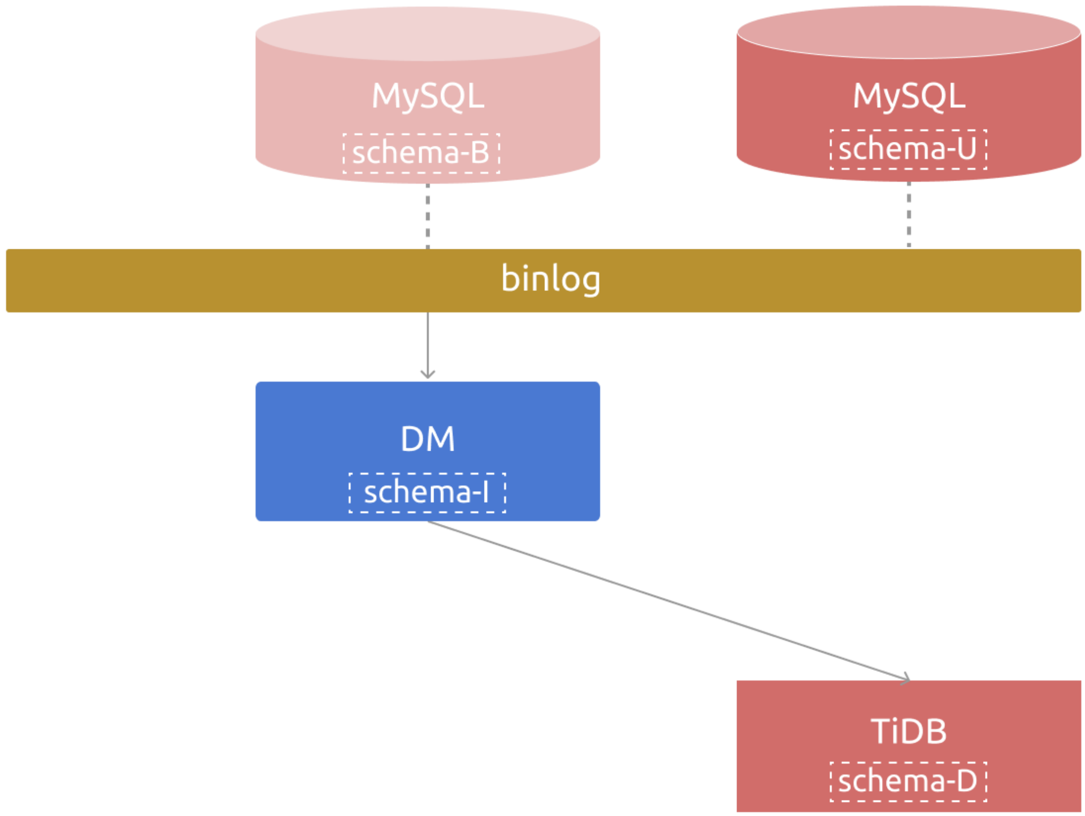

Figure 22: schema

- The upstream table schema at the current time, identified as `schema-U`.
- The table schema of the binlog event currently being consumed by DM, identified as `schema-B`. This schema corresponds to the upstream table schema at a historical time.
- The table schema currently maintained in DM (the schema tracker component), identified as `schema-I`.
- The table schema in the downstream TiDB cluster, identified as `schema-D`.

In most cases, the four table schemas above are the same.

When the upstream database performs a DDL operation to change the table schema, `schema-U` is changed. By applying the DDL operation to the internal schema tracker component and the downstream TiDB cluster, DM updates `schema-I` and `schema-D` in an orderly manner to keep them consistent with `schema-U`. Therefore, DM can then normally consume the binlog event corresponding to the `schema-B` table schema. That is, after the DDL operation is successfully migrated, `schema-U`, `schema-B`, `schema-I`, and `schema-D` are still consistent.

However, during the migration with optimistic mode sharding DDL support enabled, the `schema-D` of the downstream table might be inconsistent with the `schema-B` and `schema-I` of some upstream sharded tables. In such cases, DM still keeps `schema-I` and `schema-B` consistent to ensure that the binlog event corresponding to DML can be parsed normally.

In addition, in some scenarios (such as when the downstream table has more columns than the upstream table), `schema-D` might be inconsistent with `schema-B` and `schema-I`.

To support the scenarios mentioned above and handle other migration interruptions caused by schema inconsistency, DM provides the `operate-schema` command to obtain, modify, and delete the `schema-I` table schema maintained in DM.

4.6.2 Command

```
help operate-schema
```

```
`get`/`set`/`remove` the schema for an upstream table.

Usage:
  dmctl operate-schema <operate-type> <-s source ...> <task-name | task-file
    ↪ > <-d database> <-t table> [schema-file] [--flush] [--sync] [flags]

Flags:
  -d, --database string database name of the table
      --flush           flush the table info and checkpoint immediately
  -h, --help            help for operate-schema
      --sync            sync the table info to master to resolve shard ddl
        ↪ lock, only for optimistic mode now
  -t, --table string    table name

Global Flags:
  -s, --source strings MySQL Source ID.
```

Note:

- Because a table schema might change during data migration, to obtain a predictable table schema, currently the `operate-schema` command can be used only when the data migration task is in the `Paused` state.
- To avoid data loss due to mishandling, it is **strongly recommended** to get and backup the table schema firstly before you modify the schema.

4.6.3 Parameters

- `operate-type`:
  - Required.
  - Specifies the type of operation on the schema. The optional values are `get`, `set`, and `remove`.
- `-s`:
  - Required.
  - Specifies the MySQL source that the operation is applied to.
- `task-name | task-file`:
  - Required.
  - Specifies the task name or task file path.
- `-d`:
  - Required.
  - Specifies the name of the upstream database the table belongs to.
- `-t`:
  - Required.
  - Specifies the name of the upstream table corresponding to the table.
- `schema-file`:
  - Required when the operation type is `set`. Optional for other operation types.
  - The table schema file to be set. The file content should be a valid `CREATE TABLE` statement.
- `--flush`:
  - Optional.
  - Writes the schema to the checkpoint so that DM can load it after restarting the task.
  - The default value is `true`.
- `--sync`:
  - Optional. Only used when an error occurs in the optimistic sharding DDL mode.
  - Updates the optimistic sharding metadata with this schema.

4.6.4 Usage example

4.6.4.1 Get the table schema

If you want to get the table schema of the `` `db_single`.`t1` `` table corresponding to the `mysql-replica-01` MySQL source in the `db_single` task, run the following command:

```
operate-schema get -s mysql-replica-01 task_single -d db_single -t t1
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "CREATE TABLE `t1` ( `c1` int(11) NOT NULL, `c2` int(11)
                ↪ DEFAULT NULL, PRIMARY KEY (`c1`)) ENGINE=InnoDB DEFAULT
                ↪ CHARSET=latin1 COLLATE=latin1_bin",
            "source": "mysql-replica-01",
            "worker": "127.0.0.1:8262"
        }
    ]
}
```

4.6.4.2 Set the table schema

If you want to set the table schema of the `` `db_single`.`t1` `` table corresponding to the `mysql-replica-01` MySQL source in the `db_single` task as follows:

```sql
CREATE TABLE `t1` (
    `c1` int(11) NOT NULL,
    `c2` bigint(11) DEFAULT NULL,
    PRIMARY KEY (`c1`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin
```

Save the `CREATE TABLE` statement above as a file (for example, `db_single.t1-schema.` ↪ `sql`), and run the following command:

```
operate-schema set -s mysql-replica-01 task_single -d db_single -t t1
    ↪ db_single.t1-schema.sql
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "127.0.0.1:8262"
        }
    ]
```

```
}
```

4.6.4.3 Delete table schema

> **Note:**
>
> After the table schema maintained in DM is deleted, if a DDL/DML statement related to this table needs to be migrated to the downstream, DM will try to get the table schema from the following three sources in an orderly manner:
>
> - The `table_info` field in the checkpoint table
> - The meta information in the optimistic sharding DDL
> - The corresponding table in the downstream TiDB

If you want to delete the table schema of the `` `db_single`.`t1` `` table corresponding to the `mysql-replica-01` MySQL source in the `db_single` task, run the following command:

```
operate-schema remove -s mysql-replica-01 task_single -d db_single -t t1
```


```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "127.0.0.1:8262"
        }
    ]
}
```


4.7 Handle Alerts

This document introduces how to deal with the alert information in DM.

4.7.1 Alerts related to high availability

4.7.1.1 `DM_master_all_down`

- Description:

  If all DM-master nodes are offline, this alert is triggered.

- Solution:

  You can take the following steps to handle the alert:

  1. Check the environment of the cluster.
  2. Check the logs of all DM-master nodes for troubleshooting.

4.7.1.2 `DM_worker_offline`

- Description:

  If a DM-worker node is offline for more than one hour, this alert is triggered. In a high-availability architecture, this alert might not directly interrupt the task but increases the risk of interruption.

- Solution:

  You can take the following steps to handle the alert:

  1. View the working status of the corresponding DM-worker node.
  2. Check whether the node is connected.
  3. Troubleshoot errors through logs.

4.7.1.3 `DM_DDL_error`

- Description:

  This error occurs when DM is processing the sharding DDL operations.

- Solution:

  Refer to Troubleshoot DM.

4.7.1.4 `DM_pending_DDL`

- Description:

  If a sharding DDL operation is pending for more than one hour, this alert is triggered.

- Solution:

  In some scenarios, the pending sharding DDL operation might be what users expect. Otherwise, refer to Handle Sharding DDL Locks Manually in DM for solution.

4.7.2 Alert rules related to task status

4.7.2.1 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.

4.7.3 Alert rules related to relay log

4.7.3.1 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.

4.7.3.2 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.
  - Execute the command `pause-relay` to pause the relay log pulling process. After there is enough free disk space, resume the process by running the command `resume-relay`. Note that you must not purge upstream binlog files that have not been pulled after the relay log pulling process is paused.

4.7.3.3 `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.

4.7.3.4 `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.

4.7.3.5 `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.

4.7.3.6 `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.

4.7.4 Alert rules related to Dump/Load

4.7.4.1 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.

4.7.4.2 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.

4.7.5 Alert rules related to binlog replication

4.7.5.1 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.

4.7.5.2 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.

4.7.5.3 `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.

4.8 Daily Check

This document summarizes how to perform a daily check on TiDB Data Migration (DM).

- Method 1: Execute the `query-status` command to check the running status of the task and the error output (if any). For details, see Query Status.
- Method 2: If Prometheus and Grafana are correctly deployed when you deploy the DM cluster using TiUP, you can view DM monitoring metrics in Grafana. For example, suppose that the Grafana's address is `172.16.10.71`, go to http://172.16.10.71:3000, enter the Grafana dashboard, and select the DM Dashboard to check monitoring metrics of DM. For more information of these metrics, see DM Monitoring Metrics.
- Method 3: Check the running status of DM and the error (if any) using the log file.
  - DM-master log directory: It is specified by the `--log-file` DM-master process parameter. If DM is deployed using TiUP, the log directory is `{log_dir}` 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 TiUP, the log directory is `{log_dir}` in the DM-worker node.

5 Usage Scenarios

5.1 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).

The information of the Aurora cluster in the example is as follows:

Cluster	Endpoint	Port	Role	Version
Aurora-1	test-dm-2-0.cluster-czrtqco96yc6.us-east-2.rds.amazonaws.com	3306	Writer	Aurora (MySQL)-5.7.12
Aurora-1	test-dm-2-0.cluster-ro-czrtqco96yc6.us-east-2.rds.amazonaws.com	3306	Reader	Aurora (MySQL)-5.7.12
Aurora-2	test-dm-2-0-2.cluster-czrtqco96yc6.us-east-2.rds.amazonaws.com	3306	Writer	Aurora (MySQL)-5.7.12
Aurora-2	test-dm-2-0-2.cluster-ro-czrtqco96yc6.us-east-2.rds.amazonaws.com	3306	Reader	Aurora (MySQL)-5.7.12

The data and migration plan of the Aurora cluster are as follows:

Cluster	Database	Table	Migration
Aurora-1	migrate_me	t1	Yes
Aurora-1	ignore_me	ignore_table	No
Aurora-2	migrate_me	t2	Yes
Aurora-2	ignore_me	ignore_table	No

The Aurora users in this migration are as follows:

Cluster	User	Password
Aurora-1	root	12345678
Aurora-2	root	12345678

The TiDB cluster information in the example is as follows. The TiDB cluster is deployed using TiDB Cloud.

Node	Port	Version
tidb.6657c286.23110bc6.us-east-1.prod.aws.tidbcloud.com	4000	v4.0.2

The TiDB users in this migration are as follows:

User	Password
root	87654321

After migration, the `` `migrate_me`.`t1` `` and `` `migrate_me`.`t2` `` tables are expected to exist in the TiDB cluster. The data of these tables is consistent with that of the Aurora cluster.

> **Note:**
>
> This migration does not involve the DM Shard Merge feature. To use this feature, see DM Shard Merge Scenario.

5.1.1 Step 1: Precheck

To ensure a successful migration, you need to do prechecks before starting the migration. This section provides the precheck list and solutions to DM and Aurora components.

5.1.1.1 DM nodes deployment

As the hub of data migration, DM needs to connect to the upstream Aurora cluster and the downstream TiDB cluster. Therefore, you need to use the MySQL client to check whether the nodes in which DM is to be deployed can connect to the upstream and downstream. In addition, for details of DM requirements on hardware, software, and the node number, see DM Cluster Software and Hardware Recommendations.

5.1.1.2 Aurora

DM relies on the `ROW`-formatted binlog for incremental replication. See Enable binary for an Aurora Cluster for the configuration instruction.

If GTID is enabled in Aurora, you can migrate data based on GTID. For how to enable it, see Configuring GTID-Based Replication for an Aurora MySQL Cluster. To migrate data based on GTID, you need to set `enable-gtid` to `true` in the configuration file of data source in step 3.

> **Note:**
>
> - GTID-based data migration requires MySQL 5.7 (Aurora 2.04) version or later.
> - In addition to the Aurora-specific configuration above, the upstream database must meet other requirements for migrating from MySQL, such as table schemas, character sets, and privileges. See Checking Items for details.

5.1.2 Step 2: Deploy the DM cluster

DM can be deployed in multiple ways. Currently, it is recommended to use TiUP to deploy a DM cluster. For the specific deployment method, see Deploy DM cluster using TiUP. This example has two data sources, so at least two DM-worker nodes need to be deployed.

After deployment, you need to record the IP and service port of any DM-master node (`8261` by default) for `dmctl` to connect. This example uses `127.0.0.1:8261`. Check the DM status through TiUP using `dmctl`:

> **Note:**
>
> When using other methods to deploy DM, you can call `dmctl` in a similar way. See Introduction to dmctl.

```
tiup dmctl --master-addr 127.0.0.1:8261 list-member
```

The number of `masters` and `workers` in the returned result is consistent with the number of deployed nodes:

```
{
    "result": true,
```

```
    "msg": "",
    "members": [
        {
            "leader": {
                ...
            }
        },
        {
            "master": {
                "msg": "",
                "masters": [
                    ...
                ]
            }
        },
        {
            "worker": {
                "msg": "",
                "workers": [
                    ...
                ]
            }
        }
    ]
}
```

5.1.3 Step 3: Configure the data source

> **Note:**
>
> The configuration file used by DM supports database passwords in plaintext or ciphertext. It is recommended to use password encrypted using dmctl. To obtain the ciphertext password, see Encrypt the database password using dmctl.

Save the following configuration files of data source according to the example, in which the value of `source-id` will be used in the task configuration in step 4.

The content of `source1.yaml`:

```
## Aurora-1
source-id: "aurora-replica-01"
```

```
## To migrate data based on GTID, you need to set this item to true.
enable-gtid: false

from:
  host: "test-dm-2-0.cluster-czrtqco96yc6.us-east-2.rds.amazonaws.com"
  user: "root"
  password: "12345678"
  port: 3306
```

The content of source2.yaml:

```
## Aurora-2
source-id: "aurora-replica-02"

enable-gtid: false

from:
  host: "test-dm-2-0-2.cluster-czrtqco96yc6.us-east-2.rds.amazonaws.com"
  user: "root"
  password: "12345678"
  port: 3306
```

See Migrate Data Using Data Migration - Create Data Source, and use `dmctl` to add two data sources through TiUP.

```
tiup dmctl --master-addr 127.0.0.1:8261 operate-source create dm-test/
    ↪ source1.yaml
tiup dmctl --master-addr 127.0.0.1:8261 operate-source create dm-test/
    ↪ source2.yaml
```

When the data sources are successfully added, the return information of each data source includes a DM-worker bound to it.

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "aurora-replica-01",
            "worker": "one-dm-worker-ID"
        }
    ]
}
```

5.1.4 Step 4: Configure the task

> **Note:**
>
> Because Aurora does not support FTWRL, write operations have to be paused when you only perform the full data migration to export data. See AWS documentation for details. In this example, both full data migration and incremental replication are performed, and DM automatically enables the `safe mode` to solve this pause issue. To ensure data consistency in other combinations of task mode, see AWS documentation.

This example migrates the existing data in Aurora and replicates incremental data to TiDB in real time, which is the **full data migration plus incremental replication** mode. According to the TiDB cluster information above, the added `source-id`, and the table to be migrated, save the following task configuration file `task.yaml`:

```
## 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: "tidb.6657c286.23110bc6.us-east-1.prod.aws.tidbcloud.com"
  port: 4000
  user: "root"
  password: "87654321"

## Configuration of all the upstream MySQL instances required by the current
   ↪  data migration task.
mysql-instances:
- source-id: "aurora-replica-01"
  # The configuration items 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"
  mydumper-config-name: "global"

- source-id: "aurora-replica-02"
  block-allow-list: "global"
  mydumper-config-name: "global"
```

```
## The configuration of block and allow lists.
block-allow-list:
  global:                         # Quoted by block-allow-list: "global"
      ↪ above
    do-dbs: ["migrate_me"]        # The allow list of the upstream table to
        ↪ be migrated. Database tables that are not in the allow list will
        ↪ not be migrated.

## The configuration of the dump unit.
mydumpers:
   global:                        # Quoted by mydumper-config-name: "global"
      ↪  above
    extra-args: "--consistency none" # Aurora does not support FTWRL, you
        ↪ need to configure this option to bypass FTWRL.
```

5.1.5 Step 5: Start the task

Start the task using `dmctl` through TiUP.

> **Note:**
>
> Currently, when using `dmctl` in TiUP, you need to use the absolute path of `task.yaml`. TiUP will support the relative path in later versions.

```
tiup dmctl --master-addr 127.0.0.1:8261 start-task /absolute/path/to/task.
    ↪ yaml --remove-meta
```

If the task is successfully started, the following information is returned:

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "aurora-replica-01",
            "worker": "one-dm-worker-ID"
        },
        {
            "result": true,
```

```
            "msg": "",
            "source": "aurora-replica-02",
            "worker": "another-dm-worker-ID"
        }
    ]
}
```

If `source db replication privilege checker` and `source db dump privilege ↪ checker` errors are in the returned information, check whether unrecognized privileges exsit in the `errorMsg` field. For example:

```
line 1 column 287 near \"INVOKE LAMBDA ON *.* TO...
```

The returned information above shows that the `INVOKE LAMBDA` privilege causes an error. If the privilege is Aurora-specific, add the following content to the configuration file to skip the check. DM will improve the automatic handling of Aurora privileges in later versions.

```
ignore-checking-items: ["replication_privilege","dump_privilege"]
```

5.1.6 Step 6: Query the task and validate the data

Use `dmctl` through TiUP to query information of the on-going migration task and the task status.

```
tiup dmctl --master-addr 127.0.0.1:8261 query-status
```

If the task is running normally, the following information is returned.

```
{
    "result": true,
    "msg": "",
    "tasks": [
        {
            "taskName": "test",
            "taskStatus": "Running",
            "sources": [
                "aurora-replica-01",
                "aurora-replica-02"
            ]
        }
    ]
}
```

You can query data in the downstream, modify data in Aurora, and validate the data migrated to TiDB.

5.2 Migration when There Are More Columns in the Downstream TiDB Table

This document describes how to migrate tables using DM when there are more columns in the downstream TiDB table schema than the upstream table schema.

5.2.1 The table shcema of the data source

This document uses the follwing data source example:

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

5.2.2 Migration requirements

Create a customized table `log.messages` in TiDB. Its schema contains not only all the columns in the `log.messages` table of the data source, but also additional columns. In this case, migrate the table `log.messages` of the data source to the table `log.messages` of the TiDB cluster.

Note:

- The columns that only exist in the downstream TiDB must be given a default value or allowed to be `NULL`.
- For tables that are being migrated by DM, you can directly add new columns in the downstream TiDB that are given a default value or allowed to be `NULL`. Adding such new columns does not affect the data migration.

5.2.3 Only migrate incremental data to TiDB and the downstream TiDB table has more columns

If your migration task contains full data migration, the task can operate normally. If you have already used other tools to do full data migration and this migration task only uses DM to replicate incremental data, refer to Migrate Incremental Data to TiDB to create a data migration task. At the same time, you need to manually configure the table schema in DM for MySQL binlog parsing.

Otherwise, after creating the task, the following data migration errors occur when you execute the query-status‘ command:

```
"errors": [
    {
        "ErrCode": 36027,
        "ErrClass": "sync-unit",
        "ErrScope": "internal",
        "ErrLevel": "high",
        "Message": "startLocation: [position: (mysql-bin.000001, 2022), gtid-set:09bec856-ba95-11ea-850a-58f2b4af5188:1-9 ], endLocation: [position: (mysql-bin.000001, 2022), gtid-set: 09bec856-ba95-11ea-850a-58f2b4af5188:1-9]: gen insert sqls failed, schema: log, table: messages: Column count doesn't match value count: 3 (columns) vs 2 (values)",
        "RawCause": "",
        "Workaround": ""
    }
]
```

The reason for the above errors is that when DM migrates the binlog event, if DM has not maintained internally the table schema corresponding to that table, DM tries to use the current table schema in the downstream to parse the binlog event and generate the corresponding DML statement. If the number of columns in the binlog event is inconsistent with the number of columns in the downstream table schema, the above error might occur.

In such cases, you can execute the `operate-schema` command to specify for the table a table schema that matches the binlog event. If you are migrating sharded tables, you need to configure the table schema in DM for parsing MySQL binlog for each sharded tables according to the following steps:

1. Specify the table schema for the table `log.messages` to be migrated in the data source. The table schema needs to correspond to the data of the binlog event to be replicated by DM. Then save the `CREATE TABLE` table schema statement in a file. For example, save the following table schema in the `log.messages.sql` file:

   ```sql
   CREATE TABLE `messages` (
     `id` int(11) NOT NULL,
     `message` varchar(255) DEFAULT NULL,
     PRIMARY KEY (`id`)
   )
   ```

2. Execute the `operate-schema` command to set the table schema. At this time, the task should be in the `Paused` state because of the above error.

   ```shell
   tiup dmctl --master-addr <master-addr> operate-schema set -s mysql-01 task-test -d log -t message log.message.sql
   ```

3. Execute the `resume-task` command to resume the `Paused` task.

4. Execute the `query-status` command to check whether the data migration task is running normally.

5.3 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` in the corresponding source configuration file.
> - 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.

5.3.1 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 (`syncerBinlogGtid`) corresponding to the binlog that the current processing unit of binlog replication has replicated to the downstream. Mark the sets as `gtid-S`.
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-S` contains `gtid-P`. `gtid-P` can be empty.
    - `gtid-E` contains `gtid-S`.
5. Use `pause-task` to pause all running tasks of data migration.
6. Change the VIP for it to direct at the new MySQL instance.
7. Use `resume-task` to resume the previous migration task.

5.3.2 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 (`syncerBinlogGtid`) corresponding to the binlog that the current processing unit of binlog replication has replicated to the downstream. Mark this sets as `gtid-S`.
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-S` contains `gtid-P`. `gtid-P` can be empty.

- gtid-E contains gtid-S.

5. Use stop-task to stop all running tasks of data migration.
6. Use the operator-source stop command to remove the source configuration corresponding to the address of the old MySQL instance from the DM cluster.
7. Update the address of the MySQL instance in the source configuration file and use the operate-source create command to reload the new source configuration in the DM cluster.
8. Use start-task to restart the migration task.

6 Troubleshoot

6.1 Handle Errors

This document introduces the error system and how to handle common errors when you use DM.

6.1.1 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.

6.1.2 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.

6.1.3 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. |

6.1.3.1 What can I do when a migration task is interrupted with the `invalid connection` error returned?

6.1.3.1.1 Reason

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.

6.1.3.1.2 Solutions

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`.

- 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.

6.1.3.2 A migration task is interrupted with the `driver: bad connection` error returned

6.1.3.2.1 Reason

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.

6.1.3.2.2 Solution

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.

6.1.3.3 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

6.1.3.3.1 Reason

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.

6.1.3.3.2 Solutions

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 `ssource-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.

6.1.3.4 `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 database 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.

6.1.3.5 The `load` processing unit reports the error `packet for query is too large. Try adjusting the 'max_allowed_packet' variable`

6.1.3.5.1 Reasons

- Both MySQL client and MySQL/TiDB server have the quota limits for `max_allowed_packet`. If any `max_allowed_packet` exceeds a limit, the client receives the error message. Currently, for the latest version of DM and TiDB server, the default value of `max_allowed_packet` is `64M`.
- The full data import processing unit in DM does not support splitting the SQL file exported by the Dump processing unit in DM.

6.1.3.5.2 Solutions

- It is recommended to set the `statement-size` option of `extra-args` for the Dump processing unit:

  According to the default `--statement-size` setting, the default size of `Insert Statement` generated by the Dump processing unit is about `1M`. With this default setting, the load processing unit does not report the error `packet for query is too large. Try adjusting the 'max_allowed_packet' variable` in most cases.

  Sometimes you might receive the following `WARN` log during the data dump. This `WARN` log does not affect the dump process. This only means that wide tables are dumped.

  ```
  Row bigger than statement_size for xxx
  ```

- If the single row of the wide table exceeds `64M`, you need to modify the following configurations and make sure the configurations take effect.
  - Execute `set @@global.max_allowed_packet=134217728` (`134217728` = 128 MB) in the TiDB server.
  - First add the `max-allowed-packet: 134217728` (128 MB) to the `target-database` section in the DM task configuration file. Then, execute the `stop-task` command and execute the `start-task` command.

6.2 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 2.0-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.

6.2.1 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.

6.2.1.1 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.

6.2.1.2 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.

6.2.1.3 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.

6.2.2 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.

6.2.3 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.

6.2.3.1 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.

6.2.3.2 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.

6.2.3.3 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`.

7 Performance Tuning

7.1 Optimize Configuration of DM

This document introduces how to optimize the configuration of the data migration task to improve the performance of data migration.

7.1.1 Full data export

`mydumpers` is the configuration item related to full data export. This section describes how to configure performance-related options.

7.1.1.1 `rows`

Setting the `rows` option enables concurrently exporting data from a single table using multi-thread. The value of `rows` is the maximum number of rows contained in each exported chunk. After this option is enabled, DM selects a column as the split benchmark when the data of a MySQL single table is concurrently exported. This column can be one of the following columns: the primary key column, the unique index column, and the normal index column (ordered from highest priority to lowest). Make sure this column is of integer type (for example, `INT`, `MEDIUMINT`, `BIGINT`).

The value of `rows` can be set to 10000. You can change this value according to the total number of rows in the table and the performance of the database. In addition, you need to set `threads` to control the number of concurrent threads. By default, the value of `threads` is 4. You can adjust this value as needed.

7.1.1.2 `chunk-filesize`

During full backup, DM splits the data of each table into multiple chunks according to the value of the `chunk-filesize` option. Each chunk is saved in a file with a size of about `chunk-filesize`. In this way, data is split into multiple files and you can use the parallel processing of the DM Load unit to improve the import speed. The default value of this option is 64 (in MB). Normally, you do not need to set this option. If you set it, adjust the value of this option according to the size of the full data.

> **Note:**
>
> - You cannot update the value of `mydumpers` after the migration task is created. Be sure about the value of each option before creating the task. If you need to update the value, stop the task using dmctl, update the configuration file, and re-create the task.
> - `mydumpers.threads` can be replaced with the `mydumper-thread` configuration item for simplicity.
> - If `rows` is set`,` DM ignores the value of `chunk-filesize`.

7.1.2 Full data import

`loaders` is the configuration item related to full data import. This section describes how to configure performance-related options.

7.1.2.1 `pool-size`

The `pool-size` option determines the number of threads in the DM Load unit. The default value is 16. Normally, you do not need to set this option. If you set it, adjust the value of this option according to the size of the full data and the performance of the database.

Note:

- You cannot update the value of `loaders` after the migration task is created. Be sure about the value of each option before creating the task. If you need to update the value, stop the task using dmctl, update the configuration file, and re-create the task.
- `loaders.pool-size` can be replaced with the `loader-thread` configuration item for simplicity.

7.1.3 Incremental data replication

`syncers` is the configuration item related to incremental data replication. This section describes how to configure performance-related options.

7.1.3.1 `worker-count`

`worker-count` determines the number of threads for concurrent replication of DMLs in the DM Sync unit. The default value is 16. To speed up data replication, increase the value of this option appropriately.

7.1.3.2 `batch`

`batch` determines the number of DMLs included in each transaction when the data is replicated to the downstream database during the DM Sync unit. The default value is 100. Normally, you do not need to change the value of this option.

Note:

- You cannot update the value of `syncers` after the replication task is created. Be sure about the value of each option before creating the task. If you need to update the value, stop the task using dmctl, update the configuration file, and re-create the task.
- `syncers.worker-count` can be replaced with the `syncer-thread` configuration item for simplicity.
- You can change the values of `worker-count` and `batch` according to the actual scenario. For example, if there is a high network delay between DM and the downstream database, you can increase the value of `worker-count` and decrease the value of `batch` appropriately.

8 Reference

8.1 Architecture

8.1.1 Data Migration Overview

TiDB Data Migration (DM) is an integrated data migration task management platform, which supports the full data migration and the incremental data replication from MySQL-compatible databases (such as MySQL, MariaDB, and Aurora MySQL) into TiDB. It can help to reduce the operation cost of data migration and simplify the troubleshooting process. When using DM for data migration, you need to perform the following operations:

- Deploy a DM Cluster
- Create upstream data source and save data source access information
- Create data migration tasks to migrate data from data sources to TiDB

The data migration task includes two stages: full data migration and incremental data replication:

- Full data migration: Migrate the table structure of the corresponding table from the data source to TiDB, and then read the data stored in the data source and write it to the TiDB cluster.
- Incremental data replication: After the full data migration is completed, the corresponding table changes from the data source are read and then written to the TiDB cluster.

The following describes the features of DM.

8.1.1.1 Basic features

This section describes the basic data migration features provided by DM.

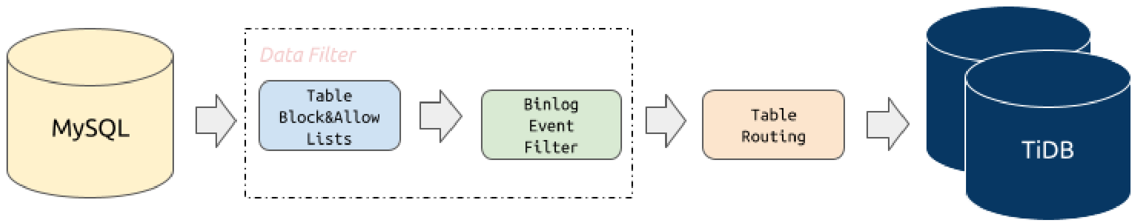

Figure 23: DM Core Features

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

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

8.1.1.1.2 Binlog event filtering

The binlog event filtering feature means that DM can filter certain types of SQL statements from certain tables in the source database. For example, you can filter all `INSERT` statements in the table `test.sbtest` or filter all `TRUNCATE TABLE` statements in the schema `test`.

8.1.1.1.3 Schema and table routing

The schema and table routing feature means that DM can migrate a certain table of the source database to the specified table in the downstream. For example, you can migrate the table structure and data from the table `test.sbtest1` in the source database to the table `test.sbtest2` in TiDB. This is also a core feature for merging and migrating sharded databases and tables.

8.1.1.2 Advanced features

8.1.1.2.1 Shard merge and migration

DM supports merging and migrating the original sharded instances and tables from the source databases into TiDB, but with some restrictions. For details, see Sharding DDL usage restrictions in the pessimistic mode and Sharding DDL usage restrictions in the optimistic mode.

8.1.1.2.2 Optimization for third-party online-schema-change tools in the migration process

In the MySQL ecosystem, tools such as gh-ost and pt-osc are widely used. DM provides support for these tools to avoid migrating unnecessary intermediate data. For details, see Online DDL Tools

8.1.1.2.3 Filter certain row changes using SQL expressions

In the phase of incremental replication, DM supports the configuration of SQL expressions to filter out certain row changes, which lets you replicate the data with a greater granularity. For more information, refer to Filter Certain Row Changes Using SQL Expressions.

8.1.1.3 Usage restrictions

Before using the DM tool, note the following restrictions:

- Database version requirements
    - MySQL version > 5.5
    - 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.
> - MySQL version > 5.7.1
> - MariaDB version >= 10.1.3

> **Warning:**
> Support for MySQL 8.0 is an experimental feature of TiDB Data Migration v2.0. It is **NOT** recommended that you use it in a production environment.

- DDL syntax compatibility
    - 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 merge with conflicts
  - 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 DDL migration restrictions, see Sharding DDL usage restrictions in the pessimistic mode and Sharding DDL usage restrictions in the optimistic mode.

- Switch of MySQL instances for data sources

  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, see Switch DM-worker connection via virtual IP.

8.1.2 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

8.1.2.1 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.

8.1.2.1.1 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 relay log of MySQL. For details, see MySQL Relay Log.

8.1.2.1.2 Dump processing unit

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

8.1.2.1.3 Load processing unit

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

8.1.2.1.4 Binlog replication/sync processing unit

Binlog replication/sync processing unit reads the binlog events of the upstream MySQL/-MariaDB or the binlog events of the relay log, transforms these events to SQL statements, and then applies these statements to the downstream TiDB.

8.1.2.2 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.

8.1.2.2.1 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.

8.1.2.2.2 Downstream database user privileges

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

Privilege	Scope
`SELECT`	Tables

Privilege	Scope
`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';
```

8.1.2.2.3 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
Dump	`SELECTRELOAD` (flushes tables with Read lock and unlocks tables)	NULL	Write local files

Processing unit	Minimal upstream (MySQL/MariaDB) privilege	Minimal downstream (TiDB) privilege	Minimal system privilege
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.

8.2 Command-line Flags

This document introduces DM's command-line flags.

8.2.1 DM-master

8.2.1.1 `--advertise-addr`

- The external address of DM-master used to receive client requests
- The default value is `"{master-addr}"`
- Optional flag. It can be in the form of `"domain-name:port"`

8.2.1.2 `--advertise-peer-urls`

- The external address for communication between DM-master nodes
- The default value is `"{peer-urls}"`
- Optional flag. It can be in the form of `"http(s)://domain-name:port"`

8.2.1.3 `--config`

- The configuration file path of DM-master
- The default value is `""`
- Optional flag

8.2.1.4 `--data-dir`

- The directory used to store data of DM-master
- The default value is `"default.{name}"`
- Optional flag

8.2.1.5 `--initial-cluster`

- The `"{node name}={external address}"` list used to bootstrap DM-master cluster
- The default value is `"{name}={advertise-peer-urls}"`
- This flag needs to be specified if the `join` flag is not specified. A configuration example of a 3-node cluster is `"dm-master-1=http://172.16.15.11:8291,dm-master-2=http ↪ ://172.16.15.12:8291,dm-master-3=http://172.16.15.13:8291"`

8.2.1.6 `--join`

- The existing cluster's `advertise-addr` list when a DM-master node joins this cluster
- The default value is `""`
- This flag needs to be specified if the `initial-cluster` flag is not specified. Suppose a new node joins a cluster that has 2 nodes, a configuration example is `"172.16.15.11:8261,172.16.15.12:8261"`

8.2.1.7 `--log-file`

- The output file name of the log
- The default value is `""`
- Optional flag

8.2.1.8 `-L`

- The log level
- The default value is `"info"`
- Optional flag

8.2.1.9 `--master-addr`

- The address on which DM-master listens to the client's requests
- The default value is `""`
- Required flag

8.2.1.10 `--name`

- The name of a DM-master node
- The default value is `"dm-master-{hostname}"`
- Required flag

8.2.1.11 `--peer-urls`

- The listening address for communications between DM-master nodes
- The default value is `"http://127.0.0.1:8291"`
- Required flag

8.2.2 DM-worker

8.2.2.1 `--advertise-addr`

- The external address of DM-worker used to receive client requests
- The default value is `"{worker-addr}"`
- Optional flag. It can be in the form of `"domain-name:port"`

8.2.2.2 `--config`

- The configuration file path of DM-worker
- The default value is `""`
- Optional flag

8.2.2.3 `--join`

- The `{advertise-addr}` list of DM-master nodes in a cluster when a DM-worker registers to this cluster
- The default value is `""`
- Required flag. A configuration example of 3-node (DM-master node) cluster is `"172.16.15.11:8261,172.16.15.12:8261,172.16.15.13:8261"`

8.2.2.4 `--log-file`

- The output file name of the log
- The default value is `""`
- Optional flag

8.2.2.5 `-L`

- The log level
- The default value is `"info"`
- Optional flag

8.2.2.6 `--name`

- The name of a DM-worker node
- The default value is `"{advertise-addr}"`
- Required flag

8.2.2.7 `--worker-addr`

- The address on which DM-worker listens to the client's requests
- The default value is `""`
- Required flag

8.2.3 dmctl

8.2.3.1 `--config`

- The configuration file path of dmctl
- The default value is `""`
- Optional flag

8.2.3.2 `--master-addr`

- The `{advertise-addr}` of any DM-master node in the cluster to be connected by dmctl
- The default value is `""`
- It is a required flag when dmctl interacts with DM-master

8.2.3.3 `--encrypt`

- Encrypts the plaintext database password into ciphertext
- The default value is `""`
- When this flag is specified, it is only used to encrypt the plaintext without interacting with the DM-master

8.2.3.4 `--decrypt`

- Decrypts ciphertext encrypted with dmctl into plaintext
- The default value is `""`
- When this flag is specified, it is only used to decrypt the ciphertext without interacting with the DM-master

8.3 Configuration

8.3.1 Data Migration Configuration File Overview

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

8.3.1.1 DM process configuration files

- `dm-master.toml`: The configuration file of running the DM-master process, including the topology information and the logs of the DM-master. For more details, refer to DM-master Configuration File.
- `dm-worker.toml`: The configuration file of running the DM-worker process, including the topology information and the logs of the DM-worker. For more details, refer to DM-worker Configuration File.
- `source.yaml`: The configuration of the upstream database such as MySQL and MariaDB. For more details, refer to Upstream Database Configuration File.

8.3.1.2 DM migration task configuration

8.3.1.2.1 Data migration task creation

You can take the following steps to create a data migration task:

1. Load the data source configuration into the DM cluster using dmctl.
2. Refer to the description in the Task Configuration Guide and create the configuration file `your_task.yaml`.
3. Create the data migration task using dmctl.

8.3.1.2.2 Important concepts

This section shows description of some important concepts.

Concept	Description	Configuration File
`source-id`	Uniquely represents a MySQL or MariaDB instance, or a migration group with the primary-secondary structure. The maximum length of `source-id` is 32.	`source_id` of `source.yaml`; `source-id` of `task.yaml`
DM-master ID	Uniquely represents a DM-master (by the `master-addr` parameter of `dm-master.toml`)	`master-addr` of `dm-master.toml`

Concept	Description	Configuration File
DM-worker ID	Uniquely represents a DM-worker (by the `worker-addr` parameter of `dm-worker.toml`)	`worker-addr` of `dm-worker.toml`

8.3.2 DM-master Configuration File

This document introduces the configuration of DM-master, including a configuration file template and a description of each configuration parameter in this file.

8.3.2.1 Configuration file template

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

```
name = "dm-master"

### log configuration
log-level = "info"
log-file = "dm-master.log"

### DM-master listening address
master-addr = ":8261"
advertise-addr = "127.0.0.1:8261"

### URLs for peer traffic
peer-urls = "http://127.0.0.1:8291"
advertise-peer-urls = "http://127.0.0.1:8291"

### cluster configuration
initial-cluster = "master1=http://127.0.0.1:8291,master2=http
    ↪ ://127.0.0.1:8292,master3=http://127.0.0.1:8293"
join = ""

ssl-ca = "/path/to/ca.pem"
ssl-cert = "/path/to/cert.pem"
ssl-key = "/path/to/key.pem"
```

```
cert-allowed-cn = ["dm"]
```

8.3.2.2 Configuration parameters

This section introduces the configuration parameters of DM-master.

8.3.2.2.1 Global configuration

Parameter	Description
`name`	The name of the DM-master.
`log-level`	Specifies a log level from `debug`, `info`, `warn`, `error`, and `fatal`. The default log level is `info`.
`log-file`	Specifies the log file directory. If the parameter is not specified, the logs are printed onto the standard output.
`master-addr`	Specifies the address of DM-master which provides services. You can omit the IP address and specify the port number only, such as ":8261".
`advertise-addr`	Specifies the address that DM-master advertises to the outside world.
`peer-urls`	Specifies the peer URL of the DM-master node.
`advertise-peer-urls`	Specifies the peer URL that DM-master advertises to the outside world. The value of `advertise-peer-urls` is by default the same as that of `peer-urls`.
`initial-cluster`	The value of `initial-cluster` is the combination of the `advertise-peer-urls` value of all DM-master nodes in the initial cluster.
`join`	The value of `join` is the combination of the `advertise-peer-urls` value of the existed DM-master nodes in the cluster. If the DM-master node is newly added, replace `initial-cluster` with `join`.
`ssl-ca`	The path of the file that contains list of trusted SSL CAs for DM-master to connect with other components.
`ssl-cert`	The path of the file that contains X509 certificate in PEM format for DM-master to connect with other components.

Parameter	Description
`ssl-key`	The path of the file that contains X509 key in PEM format for DM-master to connect with other components.
`cert-allowed` ↪ `-cn`	Common Name list.

8.3.3 DM-worker Configuration File

This document introduces the configuration of DM worker, including a configuration file template and a description of each configuration parameter in this file.

8.3.3.1 Configuration file template

The following is a configuration file template of the DM-worker:

```
### Worker Configuration.
name = "worker1"

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

### DM-worker listen address.
worker-addr = ":8262"
advertise-addr = "127.0.0.1:8262"
join = "http://127.0.0.1:8261,http://127.0.0.1:8361,http://127.0.0.1:8461"

keepalive-ttl = 60
relay-keepalive-ttl = 1800 # New in DM v2.0.2.

ssl-ca = "/path/to/ca.pem"
ssl-cert = "/path/to/cert.pem"
ssl-key = "/path/to/key.pem"
cert-allowed-cn = ["dm"]
```

8.3.3.2 Configuration parameters

8.3.3.2.1 Global

Parameter	Description
`name`	The name of the DM-worker.

Parameter	Description
`log-level`	Specifies a log level from `debug`, `info`, `warn`, `error`, and `fatal`. The default log level is `info`.
`log-file`	Specifies the log file directory. If this parameter is 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".
`advertise-` ↪ `addr`	Specifies the address that DM-worker advertises to the outside world.
`join`	Corresponds to one or more `master-addrs` in the DM-master configuration file.
`keepalive-` ↪ `ttl`	The keepalive time (in seconds) of a DM-worker node to the DM-master node if the upstream data source of the DM-worker node does not enable the relay log. The default value is 60s.
`relay-` ↪ `keepalive` ↪ `-ttl`	The keepalive time (in seconds) of a DM-worker node to the DM-master node if the upstream data source of the DM-worker node enables the relay log. The default value is 1800s. This parameter is added since DM v2.0.2.
`ssl-ca`	The path of the file that contains list of trusted SSL CAs for DM-worker to connect with other components.
`ssl-cert`	The path of the file that contains X509 certificate in PEM format for DM-worker to connect with other components.
`ssl-key`	The path of the file that contains X509 key in PEM format for DM-worker to connect with other components.
`cert-allowed` ↪ `-cn`	Common Name list.

8.3.4 Upstream Database Configuration File

This document introduces the configuration file of the upstream database, including a configuration file template and the description of each configuration parameter in this file.

8.3.4.1 Configuration file template

The following is a configuration file template of the upstream database:

```
source-id: "mysql-replica-01"

### Whether to enable GTID.
enable-gtid: false

### Whether to enable relay log.
enable-relay: false    # Since DM v2.0.2, this configuration item is
   ↪ deprecated. To enable the relay log feature, use the `start-relay`
   ↪ command instead.
relay-binlog-name: ""  # The file name from which DM-worker starts to pull
   ↪ the binlog.
relay-binlog-gtid: ""  # The GTID from which DM-worker starts to pull the
   ↪ binlog.
relay-dir: "relay-dir" # The directory used to store relay log. The default
   ↪ value is "relay-dir".


from:
  host: "127.0.0.1"
  port: 3306
  user: "root"
  password: "ZqMLjZ2j5khNelDEfDoUhkD5aV5fIJOe0fiog9w=" # The user password
     ↪ of the upstream database. It is recommended to use the password
     ↪ encrypted with dmctl.
  security:                   # The TLS configuration of the upstream
     ↪ database
    ssl-ca: "/path/to/ca.pem"
    ssl-cert: "/path/to/cert.pem"
    ssl-key: "/path/to/key.pem"

### purge:
###   interval: 3600
###   expires: 0
###   remain-space: 15

### checker:
###   check-enable: true
###   backoff-rollback: 5m0s
###   backoff-max: 5m0s    # The maximum value of backoff, should be larger
   ↪ than 1s

### Configure binlog event filters. New in DM v2.0.2
```

```
### case-sensitive: false
### filters:
### - schema-pattern: dmctl
###   table-pattern: t_1
###   events: []
###   sql-pattern:
###   - alter table .* add column `aaa` int
###   action: Ignore
```

Note:

In DM v2.0.1, DO NOT set `enable-gtid` and `enable-relay` to `true` at the same time. Otherwise, it may cause loss of incremental data.

8.3.4.2 Configuration parameters

This section describes each configuration parameter in the configuration file.

8.3.4.2.1 Global configuration

Parameter	Description
`source-id`	Represents a MySQL instance ID.
`enable-gtid`	Determines whether to pull binlog from the upstream using GTID. The default value is `false`. In general, you do not need to configure `enable-gtid` manually. However, if GTID is enabled in the upstream database, and the primary/secondary switch is required, you need to set `enable-gtid` to `true`.
`enable-relay`	Determines whether to enable the relay log feature. The default value is `false`. Since DM v2.0.2, this configuration item is deprecated. To enable the relay log feature, use the `start-relay` command instead.
`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 will pull the binlogs starting from the latest one.

Parameter	Description
`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 will pull the binlogs starting from the latest GTID.
`relay-dir`	Specifies the relay log directory.
`host`	Specifies the host of the upstream database.
`port`	Specifies the port of the upstream database.
`user`	Specifies the username of the upstream database.
`password`	Specifies the user password of the upstream database. It is recommended to use the password encrypted with dmctl.
`security`	Specifies the TLS config of the upstream database. The configured file paths of the certificates must be accessible to all nodes. If the configured file paths are local paths, then all the nodes in the cluster need to store a copy of the certificates in the same path of each host.

8.3.4.2.2 Relay log cleanup strategy configuration (`purge`)

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`

Parameter	Description	Default value
`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:**
>
> The automatic data purge strategy only takes effect when `interval` is not 0 and at least one of the two configuration items `expires` and `remain-space` is not 0.

8.3.4.2.3 Task status checker configuration (`checker`)

DM periodically checks the current task status and error message to determine if resuming the task will eliminate the error. If needed, DM automatically retries to resume the task. DM adjusts the checking interval using the exponential backoff strategy. Its behaviors can be adjusted by the following configuration.

Parameter	Description
`check-enable`	Whether to enable this feature.
`backoff-rollback`	If the current checking interval of backoff strategy is larger than this value and the task status is normal, DM will try to decrease the interval.
`backoff-max`	The maximum value of checking interval of backoff strategy, must be larger than 1 second.

8.3.4.2.4 Binlog event filter

Starting from DM v2.0.2, you can configure binlog event filters in the source configuration file.

Parameter	Description
`case-sensitive`	Determines whether the filtering rules are case-sensitive. The default value is `false`.
`filters`	Sets binlog event filtering rules. For details, see Binlog event filter parameter explanation.

9 Secure

9.1 Enable TLS for DM Connections

This document describes how to enable encrypted data transmission for DM connections, including connections between the DM-master, DM-worker, and dmctl components, and connections between DM and the upstream or downstream database.

9.1.1 Enable encrypted data transmission between DM-master, DM-worker, and dmctl

This section introduces how to enable encrypted data transmission between DM-master, DM-worker, and dmctl.

9.1.1.1 Configure and enable encrypted data transmission

1. Prepare certificates.

   It is recommended to prepare a server certificate for DM-master and DM-worker separately. Make sure that the two components can authenticate each other. You can choose to share one client certificate for dmctl.

   To generate self-signed certificates, you can use `openssl`, `cfssl` and other tools based on `openssl`, such as `easy-rsa`.

   If you choose `openssl`, you can refer to generating self-signed certificates.

2. Configure certificates.

> **Note:**
> You can configure DM-master, DM-worker, and dmctl to use the same set of certificates.

- DM-master
  Configure in the configuration file or command-line arguments:

```
ssl-ca = "/path/to/ca.pem"
ssl-cert = "/path/to/master-cert.pem"
ssl-key = "/path/to/master-key.pem"
```

- DM-worker
  Configure in the configuration file or command-line arguments:

```
ssl-ca = "/path/to/ca.pem"
ssl-cert = "/path/to/worker-cert.pem"
ssl-key = "/path/to/worker-key.pem"
```

- dmctl

  After enabling encrypted transmission in a DM cluster, if you need to connect to the cluster using dmctl, specify the client certificate. For example:

```
./dmctl --master-addr=127.0.0.1:8261 --ssl-ca /path/to/ca.pem --ssl
    ↪ -cert /path/to/client-cert.pem --ssl-key /path/to/client-key
    ↪ .pem
```

9.1.1.2 Verify component caller's identity

The Common Name is used for caller verification. In general, the callee needs to verify the caller's identity, in addition to verifying the key, the certificates, and the CA provided by the caller. For example, DM-worker can only be accessed by DM-master, and other visitors are blocked even though they have legitimate certificates.

To verify component caller's identity, you need to mark the certificate user identity using `Common Name` (CN) when generating the certificate, and to check the caller's identity by configuring the `Common Name` list for the callee.

- DM-master

  Configure in the configuration file or command-line arguments:

```
cert-allowed-cn = ["dm"]
```

- DM-worker

  Configure in the configuration file or command-line arguments:

```
cert-allowed-cn = ["dm"]
```

9.1.1.3 Reload certificates

To reload the certificates and the keys, DM-master, DM-worker, and dmctl reread the current certificates and the key files each time a new connection is created.

When the files specified by `ssl-ca`, `ssl-cert` or `ssl-key` are updated, restart DM components to reload the certificates and the key files and reconnect with each other.

9.1.2 Enable encrypted data transmission between DM components and the upstream or downstream database

This section introduces how to enable encrypted data transmission between DM components and the upstream or downstream database.

9.1.2.1 Enable encrypted data transmission for upstream database

1. Configure the upstream database, enable the encryption support, and set the server certificate. For detailed operations, see Using encrypted connections.

2. Set the MySQL client certificate in the source configuration file:

> **Note:**
> Make sure that all DM-master and DM-worker components can read the certificates and the key files via specified paths.

```
from:
    security:
        ssl-ca: "/path/to/mysql-ca.pem"
        ssl-cert: "/path/to/mysql-cert.pem"
        ssl-key: "/path/to/mysql-key.pem"
```

9.1.2.2 Enable encrypted data transmission for downstream TiDB

1. Configure the downstream TiDB to use encrypted connections. For detailed operatons, refer to Configure TiDB to use encrypted connections.

2. Set the TiDB client certificate in the task configuration file:

> **Note:**
> Make sure that all DM-master and DM-worker components can read the certificates and the key files via specified paths.

```
target-database:
    security:
        ssl-ca: "/path/to/tidb-ca.pem"
        ssl-cert: "/path/to/tidb-client-cert.pem"
        ssl-key: "/path/to/tidb-client-key.pem"
```

9.2 Generate Self-signed Certificates

This document provides an example of using `openssl` to generate a self-signed certificate. You can also generate certificates and keys that meet requirements according to your demands.

Assume that the topology of the instance cluster is as follows:

Name	Host IP	Services
node1	172.16.10.11	DM-master1
node2	172.16.10.12	DM-master2
node3	172.16.10.13	DM-master3
node4	172.16.10.14	DM-worker1
node5	172.16.10.15	DM-worker2
node6	172.16.10.16	DM-worker3

9.2.1 Install OpenSSL

- For Debian or Ubuntu OS:

  ```
  apt install openssl
  ```

- For RedHat or CentOS OS:

  ```
  yum install openssl
  ```

You can also refer to OpenSSL's official download document for installation.

9.2.2 Generate the CA certificate

A certificate authority (CA) is a trusted entity that issues digital certificates. In practice, contact your administrator to issue the certificate or use a trusted CA. CA manages multiple certificate pairs. Here you only need to generate an original pair of certificates as follows.

1. Generate the CA key:

   ```
   openssl genrsa -out ca-key.pem 4096
   ```

2. Generate the CA certificates:

   ```
   openssl req -new -x509 -days 1000 -key ca-key.pem -out ca.pem
   ```

3. Validate the CA certificates:

   ```
   openssl x509 -text -in ca.pem -noout
   ```

9.2.3 Issue certificates for individual components

9.2.3.1 Certificates that might be used in the cluster

- The `master` certificate used by DM-master to authenticate DM-master for other components.
- The `worker` certificate used by DM-worker to authenticate DM-worker for other components.
- The `client` certificate used by dmctl to authenticate clients for DM-master and DM-worker.

9.2.3.2 Issue certificates for DM-master

To issue a certificate to a DM-master instance, perform the following steps:

1. Generate the private key corresponding to the certificate:

```
openssl genrsa -out master-key.pem 2048
```

2. Make a copy of the OpenSSL configuration template file (Refer to the actual location of your template file because it might have more than one location):

```
cp /usr/lib/ssl/openssl.cnf .
```

   If you do not know the actual location, look for it in the root directory:

```
find / -name openssl.cnf
```

3. Edit `openssl.cnf`, add `req_extensions = v3_req` under the `[ req ]` field, and add `subjectAltName = @alt_names` under the `[ v3_req ]` field. Finally, create a new field and edit the information of `Subject Alternative Name` (SAN) according to the cluster topology description above.

```
[ alt_names ]
IP.1 = 127.0.0.1
IP.2 = 172.16.10.11
IP.3 = 172.16.10.12
IP.4 = 172.16.10.13
```

   The following checking items of SAN are currently supported:

   - `IP`
   - `DNS`
   - `URI`

> **Note:**
>
> If a special IP such as `0.0.0.0` is to be used for connection or communication, you must also add it to `alt_names`.

4. Save the `openssl.cnf` file, and generate the certificate request file: (When giving input to `Common Name (e.g. server FQDN or YOUR name)[]:`, you assign a Common Name (CN) to the certificate, such as `dm`. It is used by the server to validate the identity of the client. Each component does not enable the validation by default. You can enable it in the configuration file.)

```
openssl req -new -key master-key.pem -out master-cert.pem -config
    ↪ openssl.cnf
```

5. Issue and generate the certificate:

```
openssl x509 -req -days 365 -CA ca.pem -CAkey ca-key.pem -
    ↪ CAcreateserial -in master-cert.pem -out master-cert.pem -
    ↪ extensions v3_req -extfile openssl.cnf
```

6. Verify that the certificate includes the SAN field (optional):

```
openssl x509 -text -in master-cert.pem -noout
```

7. Confirm that the following files exist in your current directory:

```
ca.pem
master-cert.pem
master-key.pem
```

> **Note:**
>
> The process of issuing certificates for the DM-worker instance is similar and will not be repeated in this document.

9.2.3.3 Issue certificates for the client (dmctl)

To issue a certificate to the client (dmctl), perform the following steps:

1. Generate the private key corresponding to the certificate:

```
openssl genrsa -out client-key.pem 2048
```

2. Generate the certificate request file (in this step, you can also assign a Common Name to the certificate, which is used to allow the server to validate the identity of the client. Each component does not enable the validation by default, and you can enable it in the configuration file):

```
openssl req -new -key client-key.pem -out client-cert.pem
```

3. Issue and generate the certificate:

```
openssl x509 -req -days 365 -CA ca.pem -CAkey ca-key.pem -
    ↪ CAcreateserial -in client-cert.pem -out client-cert.pem
```

9.3 Data Migration Monitoring Metrics

If your DM cluster is deployed using TiUP, the monitoring system is also deployed at the same time. This document describes the monitoring metrics provided by DM-worker.

9.3.1 Task

In the Grafana dashboard, the default name of DM is `DM-task`.

9.3.1.1 `overview`

`Overview` contains some monitoring metrics of all the DM-worker and DM-master instances or sources in the currently selected task. The current default alert rule is only for a single DM-worker/DM-master instance/source.

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

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	N/A	N/A
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

Metric name	Description	Alert	Severity level
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

9.3.1.2 Operation errors

Metric name	Description	Alert	Severity level
before any operate error	The number of errors before any operation	N/A	N/A
source bound error	The number of errors of data source binding operations	N/A	N/A
start error	The number of errors during the start of a subtask	N/A	N/A
pause error	The number of errors during the pause of a subtask	N/A	N/A

Metric name	Description	Alert	Severity level
resume error	The number of errors during the resuming of a subtask	N/A	N/A
auto-resume error	The number of errors during the auto-resuming of a subtask	N/A	N/A
update error	The number of errors during the update of a subtask	N/A	N/A
stop error	The number of errors during the stop of a subtask	N/A	N/A

9.3.1.3 High availability

Metric name	Description	Alert	Severity level
number of dm-masters start leader components per minute	The number of DM-master attempts to enable leader related components per minute	N/A	N/A

Metric name	Description	Alert	Severity level
number of workers in different state	The number of DM-workers in different states	Some DM-worker(s) has (have) been offline for more than one hour	critical
workers' state	The state of the DM-worker	N/A	N/A
number of worker event error	The number of different types of DM-worker errors	N/A	N/A
shard ddl error per minute	The number of different types of sharding DDL errors per minute	Any sharding DDL error occurs	critical
number of pending shard ddl	The number of pending sharding DDL operations	Any pending sharding DDL operation has existed for more than one hour	critical

9.3.1.4 Task state

Metric name	Description	Alert	Severity level
task state	The state of subtasks	An alert occurs when the subtask has been in the `Paused` ↪ state for more than 20 minutes	critical

9.3.1.5 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
remaining time	The remaining time of replicating data by the load unit (in seconds)	N/A	N/A

9.3.1.6 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 for `syncer` to be completely migrated with the upstream master (in minutes)	N/A	N/A

Metric name	Description	Alert	Severity level
replicate lag gauge	The latency time it takes to replicate the binlog from upstream to downstream (in seconds)	N/A	N/A
replicate lag histogram	The histogram of replicating the binlog from upstream to downstream (in seconds). Note that due to different statistical mechanisms, the data might be inaccurate	N/A	N/A
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 upstream master	An alert occurs when the number of binlog files by which the `syncer` processing unit is behind the upstream 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
ideal QPS	The highest QPS that can be achieved when the running time of DM is 0	N/A	N/A
binlog event row	The number of rows in a binlog event	N/A	N/A
finished transaction total	The number of finished transactions in total	N/A	N/A
replication transaction batch	The number of sql rows in the transaction executed to the downstream	N/A	N/A
flush checkpoints time interval	The time interval for flushing the checkpoints (in seconds)	N/A	N/A

9.3.1.7 Relay log

> **Note:**
>
> Currently, DM v2.0 does not support enabling the relay log feature.

Metric name	Description	Alert	Severity level
storage capacity	The storage capacity of the disk occupied by the relay log	N/A	N/A
storage remain	The remaining storage capacity of the disk occupied by the relay log	An alert is needed once the value is smaller than 10G	critical
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

Metric name	Description	Alert	Severity level
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

9.3.2 Instance

In the Grafana dashboard, the default name of an instance is `DM-instance`.

9.3.2.1 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

Metric name	Description	Alert	Severity level
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
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

Metric name	Description	Alert	Severity level
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

9.3.2.2 Task

Metric name	Description	Alert	Severity level
task state	The state of subtasks for migration	An alert occurs when the subtask has been paused for more than 10 minutes	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

9.4 DM Alert Information

The alert system is deployed by default when you deploy a DM cluster using TiUP.

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.

10 TiDB Data Migration FAQ

This document collects the frequently asked questions (FAQs) about TiDB Data Migration (DM).

10.1 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.

It is a known issue that for an upstream table with no primary key in Alibaba Cloud RDS, its binlog still contains a hidden primary key column, which is inconsistent with the original table structure.

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?

10.2 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.

10.3 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.

10.4 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 Handle failed DDL statements.

> **Note:**
>
> Currently, TiDB is not compatible with all the DDL statements that MySQL supports. See MySQL Compatibility.

10.5 How to reset the data migration task?

When an exception occurs during data migration and the data migration task cannot be resumed, you need to reset the task and re-migrate the data:

1. Execute the `stop-task` command to stop the abnormal data migration task.
2. Purge the data migrated to the downstream.
3. Use one of the following ways to restart the data migration task.

- Specify a new task name in the task configuration file. Then execute `start-task {task-config-file}`.
- Execute `start-task --remove-meta {task-config-file}`.

10.6 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`.

10.7 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.

10.7.1 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.

10.7.2 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.

10.7.3 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)`.

10.8 How to handle the error `packet for query is too large. Try adjusting the 'max_allowed_packet' variable` that occurs during the full import?

Set the parameters below to a value larger than the default 67108864 (64M).

- The global variable of the TiDB server: `max_allowed_packet`.
- The configuration item in the task configuration file: `target-database.max-allowed` ↪ `-packet`. For details, refer to DM Advanced Task Configuration File.

For details, see Loader solution.

10.9 How to handle the error `Error 1054: Unknown column 'binlog_gtid' in 'field list'` that occurs when existing DM migration tasks of an DM 1.0 cluster are running on a DM 2.0 cluster?

DM 2.0 introduces more fields to metadata tables such as checkpoint. In DM 2.0, if you directly run the `start-task` command with the task configuration file of the DM 1.0 cluster to continue the incremental data replication, the error `Error 1054: Unknown column '` ↪ `binlog_gtid' in 'field list'` occurs.

This error can be handled in any of the following ways:

- Import a DM 1.0 cluster into a new DM 2.0 cluster using TiUP.
- Manually import DM migration tasks of a DM 1.0 cluster to a DM 2.0 cluster。

10.10 Why does TiUP fail to deploy some versions of DM (for example, v2.0.0-hotfix)？

You can use the `tiup list dm-master` command to view the DM versions that TiUP supports to deploy. TiUP does not manage DM versions which are not shown by this command.

10.11 How to handle the error `parse mydumper metadata error: EOF` that occurs when DM is replicating data？

You need to check the error message and log files to further analyze this error. The cause might be that the dump unit does not produce the correct metadata file due to a lack of permissions.

10.12 Why does DM report no fatal error when replicating sharded schemas and tables, but downstream data is lost?

Check the configuration items `block-allow-list` and `table-route`:

- You need to configure the names of upstream databases and tables under `block-allow` ↪ `-list`. You can add "~" before `do-tables` to use regular expressions to match names.
- `table-route` uses wildcard characters instead of regular expressions to match table names. For example, `table_parttern_[0-63]` only matches 7 tables, from `table_parttern_0` to `table_pattern_6`.

10.13 Why does the `replicate lag` monitor metric show no data when DM is not replicating from upstream?

In DM 1.0, you need to enable `enable-heartbeat` to generate the monitor data. In DM 2.0, it is expected to have no data in the monitor metric `replicate lag` because this feature is not supported.

10.14 How to handle the error `fail to initial unit Sync of subtask` when DM is starting a task, with the `RawCause` in the error message showing `context deadline exceeded`?

This is a known issue in DM 2.0.0 version and will be fixed in DM 2.0.1 version. It is likely to be triggered when a replication task has a lot of tables to process. If you use TiUP to deploy DM, you can upgrade DM to the nightly version to fix this issue. Or you can download the 2.0.0-hotfix version from the release page of DM on GitHub and manually replace the executable files.

10.15 How to handle the error `duplicate entry` when DM is replicating data?

You need to first check and confirm the following things:

- `disable-detect` is not configured in the replication task (in v2.0.7 and earlier versions).
- The data is not inserted manually or by other replication programs.
- No DML filter associated with this table is configured.

To facilitate troubleshooting, you can first collect general log files of the downstream TiDB instance and then ask for technical support at TiDB Community slack channel. The following example shows how to collect general log files:

```
# Enable general log collection
curl -X POST -d "tidb_general_log=1" http://{TiDBIP}:10080/settings
# Disable general log collection
curl -X POST -d "tidb_general_log=0" http://{TiDBIP}:10080/settings
```

When the `duplicate entry` error occurs, you need to check the log files for the records that contain conflict data.

10.16 Why do some monitoring panels show `No data point`?

It is normal for some panels to have no data. For example, when there is no error reported, no DDL lock, or the relay log feature is not enabled, the corresponding panels show `No data point`. For detailed description of each panel, see DM Monitoring Metrics.

10.17 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: "*"`. Starting from DM v2.0.1, DM pre-filters statements related to `EVENT`.

In DM v2.0, `handle-error` replaces `sql-skip`. You can use `handle-error` instead to avoid this issue.

10.18 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 1 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.

10.19 In DM v2.0, why does the full import task fail if DM restarts during the task?

In DM v2.0.1 and lower versions, if DM restarts before the full import completes, the bindings between upstream data sources and DM-worker nodes might change. For example, it is possible that the intermediate data of the dump unit is on DM-worker node A but the load unit is run by DM-worker node B, thus causing the operation to fail.

The following are two solutions to this issue:

- If the data volume is small (less than 1 TB) or the task merges sharded tables, take these steps:
    1. Clean up the imported data in the downstream database.
    2. Remove all files in the directory of exported data.
    3. Delete the task using dmctl and run the command `start-task --remove-meta` to create a new task.

  After the new task starts, it is recommended to ensure that there is no redundant DM worker node and avoid restarting or upgrading the DM cluster during the full import.
- If the data volume is large (more than 1 TB), take these steps:
    1. Clean up the imported data in the downstream database.
    2. Deploy TiDB-Lightning to the DM worker nodes that process the data.
    3. Use the Local-backend mode of TiDB-Lightning to import data that DM dump units export.
    4. After the full import completes, edit the task configuration file in the following ways and restart the task:
        - Change `task-mode` to `incremental`.
        - Set the value of `mysql-instance.meta.pos` to the position recorded in the metadata file that the dump unit outputs.

10.20 Why does DM report the error `ERROR 1236 (HY000): The slave is connecting using CHANGE MASTER TO MASTER_AUTO_POSITION = 1, but the master has purged binary logs containing GTIDs that the slave requires.` if it restarts during an incremental task?

This error indicates that the upstream binlog position recorded in the metadata file output by the dump unit has been purged during the full migration.

If this issue occurs, you need to pause the task, delete all migrated data in the downstream database, and start a new task with the `--remove-meta` option.

You can avoid this issue in advance by configuring in the following ways:

1. Increase the value of `expire_logs_days` in the upstream MySQL database to avoid wrongly purging needed binlog files before the full migration task completes. If the data volume is large, it is recommended to use dumpling and TiDB-Lightning at the same time to speed up the task.
2. Enable the relay log feature for this task so that DM can read data from relay logs even though the binlog position is purged.

10.21 Why does the Grafana dashboard of a DM cluster display `failed to fetch dashboard` if the cluster is deployed using TiUP v1.3.0 or v1.3.1?

This is a known bug of TiUP, which is fixed in TiUP v1.3.2. The following are two solutions to this issue:

- Solution one:
  1. Upgrade TiUP to a later version using the command `tiup update --self && ↪ tiup update dm`.
  2. Scale in and then scale out Grafana nodes in the cluster to restart the Grafana service.
- Solution two:
  1. Back up the `deploy/grafana-$port/bin/public` folder.
  2. Download the TiUP DM offline package and unpack it.
  3. Unpack the `grafana-v4.0.3-**.tar.gz` in the offline package.
  4. Replace the folder `deploy/grafana-$port/bin/public` with the `public` folder in `grafana-v4.0.3-**.tar.gz`.
  5. Execute `tiup dm restart $cluster_name -R grafana` to restart the Grafana service.

10.22 In DM v2.0, why does the query result of the command `query-status` show that the Syncer checkpoint GTIDs are inconsecutive if the task has `enable-relay` and `enable-gtid` enabled at the same time?

This is a known bug in DM, which is fixed in DM v2.0.2. The bug is triggered when the following two conditions are fully met at the same time:

1. Parameters `enable-relay` and `enable-gtid` are set to `true` in the source configuration file.
2. The upstream database is a **MySQL secondary database**. If you execute the command `show binlog events in '<newest-binlog>' limit 2` to query the `previous_gtids` of the database, the result is inconsecutive, such as the following example:

```
mysql> show binlog events in 'mysql-bin.000005' limit 2;
+------------------+------+----------------+-----------+-------------+------------------
    ↪
| Log_name         | Pos  | Event_type     | Server_id | End_log_pos | Info
    ↪                                                               |
+------------------+------+----------------+-----------+-------------+------------------
    ↪
| mysql-bin.000005 | 4 | Format_desc |    123452 |        123 | Server ver:
    ↪ 5.7.32-35-log, Binlog ver: 4                              |
| mysql-bin.000005 | 123 | Previous_gtids | 123452 |      194 | d3618e68
    ↪ -6052-11eb-a68b-0242ac110002:6-7                              |
+------------------+------+----------------+-----------+-------------+------------------
    ↪
```

The bug occurs if you run `query-status <task>` in dmctl to query task information and find that `subTaskStatus.sync.syncerBinlogGtid` is inconsecutive but `subTaskStatus ↪ .sync.masterBinlogGtid` is consecutive. See the following example:

```
query-status test
{
    ...
    "sources": [
       {
          ...
          "sourceStatus": {
             "source": "mysql1",
             ...
             "relayStatus": {
                "masterBinlog": "(mysql-bin.000006, 744)",
                "masterBinlogGtid": "f8004e25-6067-11eb-9fa3-0242ac110003
                   ↪ :1-50",
                ...
             }
          },
          "subTaskStatus": [
             {
                ...
                "sync": {
```

```
                    ...
                    "masterBinlog": "(mysql-bin.000006, 744)",
                    "masterBinlogGtid": "f8004e25-6067-11eb-9fa3-0242
                       ↪ ac110003:1-50",
                    "syncerBinlog": "(mysql-bin|000001.000006, 738)",
                    "syncerBinlogGtid": "f8004e25-6067-11eb-9fa3-0242
                       ↪ ac110003:1-20:40-49",
                    ...
                    "synced": false,
                    "binlogType": "local"
                }
            }
        ]
    },
    {
        ...
        "sourceStatus": {
            "source": "mysql2",
            ...
            "relayStatus": {
                "masterBinlog": "(mysql-bin.000007, 1979)",
                "masterBinlogGtid": "ddb8974e-6064-11eb-8357-0242ac110002
                   ↪ :1-25",
                ...
            }
        },
        "subTaskStatus": [
            {
                ...
                "sync": {
                    "masterBinlog": "(mysql-bin.000007, 1979)",
                    "masterBinlogGtid": "ddb8974e-6064-11eb-8357-0242
                       ↪ ac110002:1-25",
                    "syncerBinlog": "(mysql-bin|000001.000008, 1979)",
                    "syncerBinlogGtid": "ddb8974e-6064-11eb-8357-0242
                       ↪ ac110002:1-25",
                    ...
                    "synced": true,
                    "binlogType": "local"
                }
            }
        ]
    }
]
}
```

In the example, the `syncerBinlogGtid` of the data source `mysql1` is inconsecutive. In this case, you can do one of the following to handle the data loss:

- If upstream binlogs from the current time to the position recorded in the metadata of the full export task have not been purged, you can take these steps:
  1. Stop the current task and delete all data sources with inconsecutive GTIDs.
  2. Set `enable-relay` to `false` in all source configuration files.
  3. For data sources with inconsecutive GTIDs (such as `mysql1` in the above example), change the task to an incremental task and configure related `mysql-instances.meta` with metadata information of each full export task, including the `binlog-name`, `binlog-pos`, and `binlog-gtid` information.
  4. Set `syncers.safe-mode` to `true` in `task.yaml` of the incremental task and restart the task.
  5. After the incremental task replicates all missing data to the downstream, stop the task and change `safe-mode` to `false` in the `task.yaml`.
  6. Restart the task again.
- If upstream binlogs have been purged but local relay logs remain, you can take these steps:
  1. Stop the current task.
  2. For data sources with inconsecutive GTIDs (such as `mysql1` in the above example), change the task to an incremental task and configure related `mysql-instances.meta` with metadata information of each full export task, including the `binlog-name`, `binlog-pos`, and `binlog-gtid` information.
  3. In the `task.yaml` of the incremental task, change the previous value of `binlog-gtid` to the previous value of `previous_gtids`. For the above example, change `1-y` to `6-y`.
  4. Set `syncers.safe-mode` to `true` in the `task.yaml` and restart the task.
  5. After the incremental task replicates all missing data to the downstream, stop the task and change `safe-mode` to `false` in the `task.yaml`.
  6. Restart the task again.
  7. Restart the data source and set either `enable-relay` or `enable-gtid` to `false` in the source configuration file.
- If none of the above conditions is met or if the data volume of the task is small, you can take these steps:
  1. Clean up imported data in the downstream database.
  2. Restart the data source and set either `enable-relay` or `enable-gtid` to `false` in the source configuration file.
  3. Create a new task and run the command `start-task task.yaml --remove-meta` to migrate data from the beginning again.

For data sources that can be replicated normally (such as `mysql2` in the above example) in the first and second solutions above, configure related `mysql-instances.meta` with `syncerBinlog` and `syncerBinlogGtid` information from `subTaskStatus.sync` when setting the incremental task.

10.23 In DM v2.0, how do I handle the error "heartbeat config is different from previous used: serverID not equal" when switching the connection between DM-workers and MySQL instances in a virtual IP environment with the `heartbeat` feature enabled?

The `heartbeat` feature is disabled by default in DM v2.0. If you enable the feature in the task configuration file, it interferes with the high availability feature. To solve this issue, you can disable the `heartbeat` feature by setting `enable-heartbeat` to `false` in the task configuration file, and then reload the task configuration file. DM will forcibly disable the `heartbeat` feature in subsequent releases.

10.24 Why does a DM-master fail to join the cluster after it restarts and DM reports the error "fail to start embed etcd, RawCause: member xxx has already been bootstrapped"?

When a DM-master starts, DM records the etcd information in the current directory. If the directory changes after the DM-master restarts, DM cannot get access to the etcd information, and thus the restart fails.

To solve this issue, you are recommended to maintain DM clusters using TiUP. In the case that you need to deploy using binary files, you need to configure `data-dir` with absolute paths in the configuration file of the DM-master, or pay attention to the current directory where you run the command.

10.25 Why DM-master cannot be connected when I use dmctl to execute commands?

When using dmctl execute commands, you might find the connection to DM master fails (even if you have specified the parameter value of `--master-addr` in the command), and the error message is like `RawCause: context deadline exceeded, Workaround: please ↪ check your network connection.`. But afer checking the network connection using commands like `telnet <master-addr>`, no exception is found.

In this case, you can check the environment variable `https_proxy` (note that it is **https**). If this variable is configured, dmctl automatically connects the host and port specified by `https_proxy`. If the host does not have a corresponding `proxy` forwarding service, the connection fails.

To solve this issue, check whether `https_proxy` is mandatory. If not, cancel the setting. Otherwise, add the environment variable setting `https_proxy="" ./dmctl --master-addr "x.x.x.x:8261"` before the oringial dmctl commands.

> **Note:**
>
> The environment variables related to `proxy` include `http_proxy`, `https_proxy`, and `no_proxy`. If the connection error persists after you perform the above steps, check whether the configuration parameters of `http_proxy` and `no_proxy` are correct.

10.26 How to handle the returned error when executing start-relay command for DM versions from 2.0.2 to 2.0.6?

```
flush local meta, Rawcause: open relay-dir/xxx.000001/relay.metayyyy: no
  ↪ such file or directory
```

The above error might be made in the following cases:

- DM has been upgraded from v2.0.1 and earlier to v2.0.2 - v2.0.6, and relay log is started before the upgrade and restarted after the upgrade.
- Execute the stop-relay command to pause the relay log and then restart it.

You can avoid this error by the following options:

- Restart relay log:

  ```
  » stop-relay -s sourceID workerName
  » start-relay -s sourceID workerName
  ```

- Upgrade DM to v2.0.7 or later versions.

11 TiDB Data Migration Glossary

This document lists the terms used in the logs, monitoring, configurations, and documentation of TiDB Data Migration (DM).

11.1 B

11.1.1 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.

11.1.2 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.

11.1.3 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.

11.1.4 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.

11.1.5 Binlog replication processing unit/sync unit

Binlog replication processing unit is the processing unit used in DM-worker to read upstream binlogs or local relay logs, and to migrate 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.

11.1.6 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 Replication Filtering and MariaDB Replication Filters.

11.2 C

11.2.1 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`.

11.3 D

11.3.1 Dump processing unit/dump 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.

11.4 G

11.4.1 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.

11.5 L

11.5.1 Load processing unit/load 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.

11.6 M

11.6.1 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.

11.7 R

11.7.1 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. Its role is similar to MySQL Relay Log and MariaDB Relay Log.

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.

11.7.2 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.

11.7.3 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.

11.8 S

11.8.1 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 1 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.
- If the argument `--consistency none` is configured for the dump processing unit of a full migration task, it cannot be determined whether the binlog changes at the beginning of the export affect the exported data or not. Therefore, the safe mode remains enabled for the incremental replication of these binlog changes.

11.8.2 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.

11.8.3 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 in the pessimistic mode for details. In the current documentation, the shard DDL lock is also referred to as the sharding DDL lock.

11.8.4 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 data from sharded tables in the pessimistic mode for details. In the current documentation, the shard group is also referred to as the sharding group.

11.8.5 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.

11.8.6 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.

11.9 T

11.9.1 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.

11.9.2 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.

11.9.3 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.

12 Release Notes

12.1 v2.0

12.1.1 DM 2.0.7 Release Notes

Release date: September 29, 2021

DM version: 2.0.7

12.1.1.1 Bug fixes

- Fix the error that binlog event is purged when switching `enable-gtid` in source configuration from `false` to `true` #2094
- Fix the memory leak problem of schema-tracker #2133

12.1.1.2 Improvements

- Disable background statistic job in schema tracker to reduce CPU consumption #2065
- Support regular expressions for online DDL shadow and trash tables #2139

12.1.1.3 Known issues

GitHub issues

12.1.2 DM 2.0.6 Release Notes

Release date: August 13, 2021

DM version: 2.0.6

12.1.2.1 Bug fixes

- Fix the issue that the metadata inconsistency between DDL infos and upstream tables in the optimistic sharding DDL mode causes DM-master panic #1971

12.1.2.2 Known issues

GitHub issues

12.1.3 DM 2.0.5 Release Notes

Release date: July 30, 2021

DM version: 2.0.5

12.1.3.1 Improvements

- Support for filtering certain DML using SQL expressions #1832
- Add `config import/export` command to import and export cluster sources and tasks configuration files for downgrade #1921
- Optimize safe-mode to improve replication efficiency #1920
- Maximize compatibility with upstream SQL_MODE #1894
- Support upstream using both pt and gh-ost online DDL modes in one task #1918
- Improve the efficiency of replication of DECIMAL types #1841
- Support for automatic retry of transaction-related retryable errors #1916

12.1.3.2 Bug fixes

- Fix the issue that the inconsistency of upstream and downstream primary keys might lead to data loss #1919
- Fix the issue that too many upstream sources cause cluster upgrade failure and DM-master OOM #1868
- Fix the issue of the configuration item `case-sensitive` #1886
- Fix the issue that the default value of `tidb_enable_change_column_type` inside DM is wrong #1843
- Fix the issue that the `auto_random` column in downstream may causes task interruption #1847
- Fix the issue that `operate-schema set -flush` command causes DM-worker panic #1829

- Fix the issue that DDL fails to coordinate within DM-worker due to repeated execution of the same DDL in pessimistic mode #1816
- Fix the issue that wrong configuration causes DM-worker panic #1842
- Fix the issue that redoing tasks causes loader panic #1822
- Fix the issue that DM binlog file name is not timely updated after upstream master-slave switch #1874
- Fix the issue of incorrect value of replication delay monitoring #1880
- Fix the issue that block-allow-list fails to filter online DDL in some cases #1867
- Fix the issue that the task cannot be stopped manually due to the error after automatic resuming #1917

12.1.3.3 Known issues

GitHub issues

12.1.4 DM 2.0.4 Release Notes

Release date: June 18, 2021

DM version: 2.0.4

12.1.4.1 Improvements

- Support rescheduling and automatically resuming tasks after a DM-worker goes offline first and then comes back online during the full import #1784
- Add the metric `replicationLagGauge` to monitor replication delay #1759
- Restore schemas in parallel during the full import #1701
- Support automatically adjusting the time_zone settings of both the upstream and downstream databases #1714
- Improve the speed of rolling back incremental replication tasks after the tasks meet errors #1705
- Automatically adjust GTID according to checkpoints when GTID is enabled during the incremental replication #1745
- Detect the versions of upstream and downstream databases and record the versions in log files #1693
- Use the schema from the dump stage of the full export as the initial schema for the incremental replication task of the same data source #1754
- Decrease the time that the safe mode lasts after the incremental task is restarted to one minute to improve the replication speed #1779
- Improve the usability of dmctl
  - Support setting the address of DM-master as an environment variable #1726
  - Support specifying the `master-addr` parameter anywhere in a dmctl command #1771
  - Use the `encrypt/decrypt` command instead of the `--decrypt/-encrypt` parameter to encrypt or decrypt the database password #1771

12.1.4.2 Bug fixes

- Fix the issue that data may be lost after a non-GTID task restarts from interruption #1781
- Fix the issue that the data source binding information may be lost after upgrading a DM cluster which has been downgraded before #1713
- Fix the issue that etcd reports that the wal directory does not exist when DM-master restarts #1680
- Fix the issue that the number of error messages reported from precheck exceeds the grpc limit #1688
- Fix the issue that DM-worker panics when replicating unsupported statements from a MariaDB database of an earlier version #1734
- Fix the issue that DM does not update the metric of relay log disk capacity #1753
- Fix the issue that DM may panic when getting the master status of the upstream database binlog #1774

12.1.4.3 Known issues

GitHub issues

12.1.5 DM 2.0.3 Release Notes

Release date: May 11, 2021

DM version: 2.0.3

12.1.5.1 Improvements

- Support deleting residual DDL locks using the command `unlock-ddl-lock` after the migration task is stopped #1612
- Support limiting the number of errors and warnings that DM reports during the precheck process #1621
- Optimize the behavior of the command `query-status` to get the status of upstream binlogs #1630
- Optimize the format of sharded tables' migration status output by the command `query-status` in the pessimistic mode #1650
- Print help message first when dmtcl processes commands with the `--help` input #1637
- Automatically remove the related information from monitoring panels after a DDL lock is deleted #1631
- Automatically remove the related task status from monitoring panels after a task is stopped or completed #1614

12.1.5.2 Bug fixes

- Fix the issue that DM-master becomes out of memory after DM is updated to v2.0.2 in the process of shard DDL coordination using the optimistic mode #1643 #1649
- Fix the issue that the source binding information is lost when DM is started for the first time after updated to v2.0.2 #1649
- Fix the issue that the flag in the command `operate-source show -s` does not take effect #1587
- Fix the issue that the command `operate-source stop <config-file>` fails because DM cannot connect to the source #1587
- Fix the finer-grained issue that some migration errors might be wrongly ignored #1599
- Fix the issue that the migration is interrupted when DM filters online DDL statements according to binlog event filtering rules that are configured #1668

12.1.5.3 Known issues

GitHub issues

12.1.6 DM 2.0.2 Release Notes

Release date: April 9, 2021

DM version: 2.0.2

12.1.6.1 Improvements

- Relay log GA
  - The relay log feature is no longer enabled by setting the source configuration file. Now, the feature is enabled by running commands in dmctl for specified DM-workers #1499
  - DM sends the commands `query-status -s` and `purge-relay` to all DM-workers that pull relay logs #1533
  - Align the relay unit's behavior of pulling and sending binlogs with that of the secondary MySQL database #1390
  - Reduce the scenarios where relay logs need to be purged #1400
  - Support sending heartbeat events when the relay log feature is enabled to display task progress with regular updates #1404
- Optimistic sharding DDL mode
  - Optimize operations for resolving DDL conflicts #1496 #1506 #1518 #1551
  - Adjust the DDL coordination behavior in the optimistic mode to avoid data inconsistency in advance #1510 #1512

- Support automatically recognizing the switching of upstream data sources when the source configuration needs no update, for example, when the IP address does not change #1364
- Precheck the privileges of the upstream MySQL instance at a finer granularity #1336
- Support configuring binlog event filtering rules in the source configuration file #1370
- When binding an idle upstream data source to an idle DM-worker node, DM-master nodes firstly choose the most recent binding of that DM-worker node #1373
- Improve the stability of DM automatically getting the SQL mode from the binlog file #1382 #1552
- Support automatically parsing GTIDs of different formats in the source configuration file #1385
- Extend DM-worker's TTL for keepalive to reduce scheduling caused by poor network #1405
- Support reporting an error when the configuration file contains configuration items that are not referenced #1410
- Improve the display of a GTID set by sorting it in dictionary order #1424
- Optimize monitoring and alerting rules #1438
- Support manually transferring an upstream data source to a specified DM-worker #1492
- Add configurations of etcd compaction and disk quota #1521

12.1.6.2 Bug fixes

- Fix the issue of data loss during the full data migration occurred because DM frequently restarts the task #1378
- Fix the issue that an incremental replication task fails to start when the binlog position is not specified together with GTID in the task configuration #1393
- Fix the issue that DM-worker's binding relationships become abnormal when the disk and network environments are poor #1396
- Fix the issue that enabling the relay log feature might cause data loss when the GTIDs specified in upstream binlog `previous_gtids` events are not consecutive #1390 #1430
- Disable the heartbeat feature of DM v1.0 to avoid the failure of high availability scheduling #1467
- Fix the issue that the migration fails if the upstream binlog sequence number is larger than 999999 #1476
- Fix the issue that DM commands hang when DM gets stuck in pinging the upstream and downstream databases #1477
- Fix the issue that the full import fails when the upstream database enables the `ANSI_QUOTES` mode #1497
- Fix the issue that DM might duplicate binlog events when the GTID and the relay log are enabled at the same time #1525

12.1.6.3 Known issues

GitHub issues

12.1.7 DM 2.0.1 Release Notes

Release date: December 25, 2020

DM version: 2.0.1

12.1.7.1 Improvements

- Support the relay log feature in high availability scenarios #1353
  - DM-worker supports storing relay logs only locally.
  - In scenarios where a DM-worker node is down or is offline due to network fluctuations, the newly scheduled DM-worker pulls the upstream binlog again.
- Restrict the `handle-error` command to only handle DDL errors to avoid misuse #1303
- Support simultaneously connecting multiple DM-master nodes and automatically switching connected nodes in dmctl #1349
- Add the `get-config` command to get the configuration of migration tasks and DM components #1348
- Support migrating SQL statements like `ALTER TABLE ADD COLUMN (xx, xx)` #1345
- Support automatically filtering SQL statements like `CREATE/ALTER/DROP EVENT` #1343
- Support checking whether `server-id` is set for the upstream MySQL/MariaDB instance before the incremental replication task starts #1315
- Support replicating schemas and tables with `sql` in their names during the full import #1259

12.1.7.2 Bug fixes

- Fix the issue that restarting a task might cause `fail to initial unit Sync of` ↪ `subtask` error #1274
- Fix the issue that the `pause-task` command might be blocked when it is executed during the full import #1269 #1277
- Fix the issue that DM fails to create a data source for a MariaDB instance when `enable-gtid: true` is configured #1344
- Fix the issue that the `query-status` command might be blocked when it is executed #1293
- Fix the issue that concurrently coordinating multiple DDL statements in the pessimistic shard DDL mode might block the task #1263
- Fix the issue that running the `pause-task` command might get the meaningless `sql:` ↪ `connection is already closed` error #1304
- Fix the issue that the full migration fails when the upstream instance does not have the `REPLICATION` privilege #1326
- Fix the issue that the `route-rules` configuration of a shard merge task does not take effect in the full import when the `SQL_MODE` of the task contains `ANSI_QUOTES` #1314

- Fix the issue that DM fails to automatically apply the `SQL_MODE` of the upstream database during the incremental replication #1307
- Fix the issue that DM logs the `fail to parse binlog status_vars` warning when automatically parsing the `SQL_MODE` of the upstream database #1299

12.1.8 DM 2.0 GA Release Notes

Release date: October 30, 2020

DM version: 2.0.0

12.1.8.1 Improvements

- Optimize the setting of `safe-mode` to ensure the eventual consistency of data when the upstream database, such as Amazon Aurora and Aliyun RDS, does not support FTWRL in the full export #981 #1017
- Support automatic configuration of `sql_mode` for data migration based on the global `sql_mode` of upstream and downstream databases and `sql_mode` of binlog events #1005 #1071 #1137
- Support automatic configuration of the `max_allowed_packet` from DM to the downstream TiDB, based on the global `max_allowed_packet` value of the downstream TiDB #1071
- Optimize the incremental replication speed compared with DM 2.0 RC version #1203
- Improve performance by using optimistic transaction to migrate data to TiDB by default #1107
- Support DM-worker automatically fetching and using the list of DM-master nodes in the cluster #1180
- Disable auto-resume behavior for more errors that cannot be automatically recovered #979 #1085 #1216

12.1.8.2 Bug fixes

- Fix the issue that failure to automatically set the default value of `statement-size` for full export might cause the `packet for query is too large` error or the OOM issue in TiDB #1133
- Fix DM-worker panic when there are concurrent checkpoint operations during the full import #1182
- Fix the issue that the migration task might have `table checkpoint position less than global checkpoint position` error and be interrupted after the upstream MySQL/MariaDB instance is restarted #1041
- Fix the issue that migration tasks might be interrupted when the upstream database does not enable GTID #1123
- Fix the issue that the DM-master node does not start properly after conflicts occur during the shard DDL coordination #1199

- Fix the issue that the incremental replication might be too slow when there are multiple common indexes in the table to be migrated #1063
- Fix the issue that the progress display is abnormal after restarting the migration task during the full import #1043
- Fix the issue that paused migration subtasks cannot be obtained by `query-status` after being scheduled to another DM-worker #1183
- Fix the issue that `FileSize` might not take effect during the full export #1191
- Fix the issue that the `-s` parameter in `extra-args` does not take effect during the full export #1196
- Fix the issue that enabling the online DDL feature might cause `not allowed ↪ operation: alter multiple tables in one statement` error #1192
- Fix the issue that during the incremental replication, the migration task might be interrupted when the DDL statements to be migrated are associated with other tables, such as DDL statements related to foreign keys #1101 #1108
- Fix the issue that database names and table names with character `/` are not correctly parsed during the full migration #991
- Fix the issue that after failing to migrate DDL statements to the downstream TiDB database during the incremental replication, migration tasks might not be paused and the corresponding error cannot be obtained from `query-status` #1059
- Fix the issue that concurrently coordinating multiple DDL statements in the optimistic shard DDL mode might block the task #1051
- Fix the issue that a DM-master might try to forward requests to other DM-master nodes after it becomes the leader #1157
- Fix the issue that DM cannot parse `GRANT CREATE TABLESPACE` during the precheck #1113
- Fix the issue that migration tasks are interrupted when migrating `DROP TABLE` statements but corresponding tables don't exist #990
- Fix the issue that `operate-schema` might not work properly when the `--source` parameter is specified #1106
- Fix the issue that `list-member` cannot be executed correctly after enabling TLS #1050
- Fix the issue that mixing `https` and `http` in the config items might cause the cluster to not work properly after enabling TLS #1220
- Fix the issue that the HTTP API cannot work properly after configuring the `cert- ↪ allowed-cn` parameter for DM-masters #1036
- Fix the issue that for incremental replication tasks, the configuration check fails when `binlog-gtid` is only specified in the `meta` of the task configuration #987
- Fix the issue that in the interactive mode, dmctl cannot correctly execute some commands starting or ending with blank characters #1202
- Fix the issue that the `converting NULL to string is unsupported` error is output to the log file during the full export #1014
- Fix the issue that the progress might be displayed as `NaN` during the full import #1209

12.1.9 DM 2.0 RC.2 Release Notes

Release date: September 1, 2020

DM version: 2.0.0-rc.2

12.1.9.1 Improvements

- Support more AWS Aurora-specific privileges when pre-checking the data migration task #950
- Check whether GTID is enabled for the upstream MySQL/MariaDB when configuring `enable-gtid: true` and creating a data source #957

12.1.9.2 Bug fixes

- Fix the `Column count doesn't match value count` error that occurs in the running migration task after automatically upgrading the DM cluster from v1.0.x to v2.0.0-rc #952
- Fix the issue that the DM-worker or DM-master component might not correctly exit #963
- Fix the issue that the `--no-locks` argument does not take effect on the dump processing unit in DM v2.0 #961
- Fix the `field remove-meta not found in type config.TaskConfig` error that occurs when using the task configuration file of the v1.0.x cluster to start the task of a v2.0 cluster #965
- Fix the issue that when the domain name is used as the connection address of each component, the component might not be correctly started #955
- Fix the issue that the connection between the upstream and downstream might not be released after the migration task is stopped #943
- Fix the issue that in the optimistic sharding DDL mode, concurrently executing the DDL statement on multiple sharded tables might block the sharding DDL coordination #944
- Fix the issue that the newly started DM-master might cause the `list-member` to panic #970

12.1.10 DM 2.0 RC Release Notes

Release date: August 21, 2020

DM version: 2.0.0-rc

12.1.10.1 Improvements

- Support high availability for data migration tasks

- Add an optimistic mode for sharding DDL statements
- Add the `handle-error` command to handle errors during DDL incremental replication
- Add a `workaround` field in the error returned by `query-status` to suggest the error handling method
- Improve the monitoring dashboards and alert rules
- Replace Mydumper with Dumpling as the full export unit
- Support the GTID mode when performing incremental replication to the downstream
- Support TLS connections between upstream and downstream databases, and between DM components
- Support the incremental replication scenarios where the table of the downstream has more columns than that of the upstream
- Add a `--remove-meta` option to the `start-task` command to clean up metadata related to data migration tasks
- Support dropping columns with single-column indices
- Support automatically cleaning up temporary files after a successful full import
- Support checking whether the table to be migrated has a primary key or a unique key before starting a migration task
- Support connectivity check between dmctl and DM-master while starting dmctl
- Support connectivity check for downstream TiDB during the execution of `start-task` ↪ `/check-task`
- Support replacing task names with task configuration files for some commands such as `pause-task`
- Support logs in `json` format for DM-master and DM-worker components
- Remove the call stack information and redundant fields in the error message returned by `query-status`
- Improve the binlog position information of the upstream database returned by `query` ↪ `-status`
- Improve the processing of `auto resume` when an error is encountered during the full export

12.1.10.2 Bug fixes

- Fix the issue of goroutine leak after executing `stop-task`
- Fix the issue that the task might not be paused after executing `pause-task`
- Fix the issue that the checkpoint might not be saved correctly in the initial stage of incremental replication
- Fix the issue that the `BIT` data type is incorrectly handled during incremental replication

12.1.10.3 Detailed bug fixes and changes

- Support high availability for data migration tasks #473
- Add an optimistic mode for sharding DDL statements #568

- Add the `handle-error` command to handle errors during DDL incremental replication #850
- Add a `workaround` field in the error returned by `query-status` to suggest the error handling method #753
- Improve the monitoring dashboards and alert rules #853
- Replace Mydumper with Dumpling as the full export unit #540
- Support the GTID mode when performing incremental replication to the downstream #521
- Support TLS connections between upstream and downstream databases, and between DM components #569
- Support the incremental replication scenarios where the table of the downstream has more columns than that of the upstream #379
- Add a `--remove-meta` option to the `start-task` command to clean up metadata related to data migration tasks #651
- Support dropping columns with single-column indices #801
- Support automatically cleaning up temporary files after a successful full import #770
- Support checking whether the table to be migrated has a primary key or a unique key before starting a migration task #870
- Support connectivity check between dmctl and DM-master while starting dmctl #786
- Support connectivity check for downstream TiDB during the execution of `start-task` ↪ `/check-task` #769
- Support replacing task names with task configuration files for some commands such as `pause-task` #854
- Support logs in `json` format for DM-master and DM-worker components #808
- Remove the call stack information in the error message returned by `query-status` #746
- Remove the redundant fields in the error message returned by `query-status` #771
- Improve the binlog position information of the upstream database returned by `query` ↪ `-status` #830
- Improve the processing of `auto resume` when an error is encountered during the full export #872
- Fix the issue of goroutine leak after executing `stop-task` #731
- Fix the issue that the task might not be paused after executing `pause-task` #644
- Fix the issue that the checkpoint might not be saved correctly in the initial stage of incremental replication #758
- Fix the issue that the `BIT` data type is incorrectly handled during incremental replication #876

12.2 v1.0

12.2.1 DM 1.0.7 Release Notes

Release date: June 21, 2021

DM version: 1.0.7

12.2.1.1 Bug fixes

- Fix the issue that data may be lost after a task restarts from interruption #1783

12.2.2 DM 1.0.6 Release Notes

Release date: June 17, 2020

DM version: 1.0.6

DM-Ansible version: 1.0.6

12.2.2.1 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

12.2.2.2 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

12.2.2.3 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

12.2.3 DM 1.0.5 Release Notes

Release date: April 27, 2020

DM version: 1.0.5

DM-Ansible version: 1.0.5

12.2.3.1 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

12.2.3.2 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

12.2.3.3 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

12.2.4 DM 1.0.4 Release Notes

Release date: March 13, 2020

DM version: 1.0.4

DM-Ansible version: 1.0.4

12.2.4.1 Improvements

- Add English UI for DM-portal
- Add the `--more` parameter in the `query-status` command to show complete migration status information

12.2.4.2 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

12.2.4.3 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

12.2.5 DM 1.0.3 Release Notes

Release date: December 13, 2019

DM version: 1.0.3

DM-Ansible version: 1.0.3

12.2.5.1 Improvements

- Add the command mode in dmctl
- Support migrating the `ALTER DATABASE` DDL statement
- Optimize the error message output

12.2.5.2 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

12.2.5.3 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

12.2.6 DM 1.0.2 Release Notes

Release date: October 30, 2019

DM version: 1.0.2

DM-Ansible version: 1.0.2

12.2.6.1 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

12.2.6.2 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

12.2.6.3 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 black & white 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