TiDB Data Migration 用户文档

PingCAP Inc.

20220929

Table of Contents

1 关于 DM

1.1 Data Migration 简介

TiDB Data Migration (DM) 是一体化的数据迁移任务管理工具，支持从与 MySQL 协议兼容的数据库（MySQL、MariaDB、Aurora MySQL）到 TiDB 的数据迁移。DM 工具旨在降低数据迁移的运维成本。使用 DM 进行数据迁移的时候，需要执行以下操作：

- 部署 DM 集群
- 创建上游数据源（source）对象，保存数据源访问信息
- 创建（多个）数据迁移任务从数据源迁移数据到 TiDB

数据迁移任务包含全量数据迁移、增量数据复制两个阶段：

- 全量数据迁移：从数据源迁移对应表的表结构到 TiDB，然后读取存量数据写入到 TiDB 集群；
- 增量数据复制：全量数据迁移完成后，从数据源读取对应的表变更然后写入到 TiDB 集群。

要快速了解 DM 的原理架构、适用场景，建议先观看下面的培训视频（时长 22 分钟）。注意本视频只作为学习参考，具体操作步骤和最新功能，请以文档内容为准。

下文介绍 DM 所具备的功能。

1.1.1 基本功能

本节介绍 DM 工具的核心功能模块。

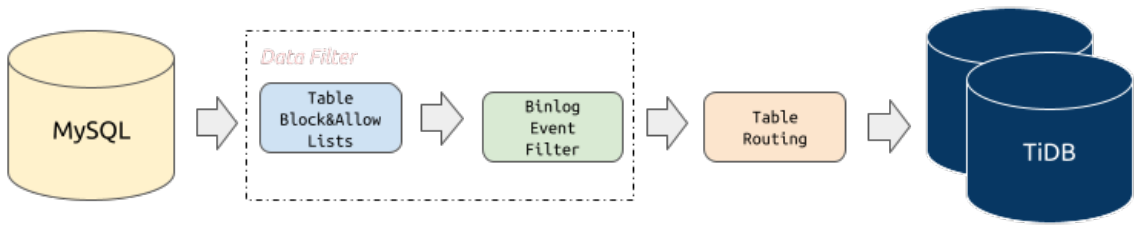


Figure 1: DM Core Features

1.1.1.1 Block & allow lists

Block & Allow Lists 的过滤规则类似于 MySQL `replication-rules-db/replication-rules-table`，用于过滤或指定只迁移某些数据库或某些表的所有操作。

1.1.1.2 Binlog event filter

Binlog Event Filter 用于过滤源数据库中特定表的特定类型操作，比如过滤掉表 `test.sbtest` 的 `INSERT` 操作或者过滤掉库 `test` 下所有表的 `TRUNCATE TABLE` 操作。

1.1.1.3 Table routing

Table Routing 是将源数据库的表迁移到下游指定表的路由功能，比如将源数据表 `test.sbtest1` 的表结构和数据迁移到 TiDB 的表 `test.sbtest2`。它也是分库分表合并迁移所需的一个核心功能。

1.1.2 高级功能

1.1.2.1 分库分表合并迁移

DM 支持对源数据的分库分表进行合并迁移，但有一些使用限制，详细信息请参考悲观模式分库分表合并迁移使用限制和乐观模式分库分表合并迁移使用限制。

1.1.2.2 对第三方 Online Schema Change 工具变更过程的同步优化

在 MySQL 生态中，gh-ost 与 pt-osc 等工具被广泛使用，DM 对其变更过程进行了特殊的优化，以避免对不必要的中间数据进行迁移。详细信息可参考online-ddl。

1.1.2.3 使用 SQL 表达式过滤某些行变更

在增量同步阶段，DM 支持配置 SQL 表达式过滤掉特定的行变更，以实现对同步数据的更精细控制。详细信息可参考使用 SQL 表达式过滤某些行变更。

1.1.3 使用限制

在使用 DM 工具之前，需了解以下限制：

- 数据库版本要求
  - MySQL 版本 > 5.5
  - MariaDB 版本 >= 10.1.2

> **注意：**
> 如果上游 MySQL/MariaDB servers 间构成主从复制结构，则需要 MySQL 版本高于 5.7.1 或者 MariaDB 版本等于或高于 10.1.3。

> **警告：**
>
> 支持从 MySQL v8.0 迁移数据是 DM v2.0 的实验特性，不建议在生产环境下使用。

- DDL 语法兼容性限制
  - 目前，TiDB 部分兼容 MySQL 支持的 DDL 语句。因为 DM 使用 TiDB parser 来解析处理 DDL 语句，所以目前仅支持 TiDB parser 支持的 DDL 语法。详见 TiDB DDL 语法支持。
  - DM 遇到不兼容的 DDL 语句时会报错。要解决此报错，需要使用 dmctl 手动处理，要么跳过该 DDL 语句，要么用指定的 DDL 语句来替换它。详见如何处理不兼容的 DDL 语句。
- 分库分表数据冲突合并
  - 如果业务分库分表之间存在数据冲突，可以参考自增主键冲突处理来解决；否则不推荐使用 DM 进行迁移，如果进行迁移则有冲突的数据会相互覆盖造成数据丢失。
  - 分库分表 DDL 同步限制，参见悲观模式下分库分表合并迁移使用限制以及乐观模式下分库分表合并迁移使用限制。
- 数据源 MySQL 实例切换
  - 当 DM-worker 通过虚拟 IP (VIP) 连接到 MySQL 且要切换 VIP 指向的 MySQL 实例时，DM 内部不同的 connection 可能会同时连接到切换前后不同的 MySQL 实例，造成 DM 拉取的 binlog 与从上游获取到的其他状态不一致，从而导致难以预期的异常行为甚至数据损坏。如需切换 VIP 指向的 MySQL 实例，请参考虚拟 IP 环境下的上游主从切换对 DM 手动执行变更。

1.2 DM 5.3.0 Release Notes

发版日期：2021 年 11 月 30 日

DM 版本：5.3.0

1.2.1 特别说明

在较早版本中 (v1.0 和 v2.0)，DM 采用独立于 TiDB 的版本号。从 DM v5.3 起，DM 采用与 TiDB 相同的版本号。DM v2.0 的下一个版本为 DM v5.3。DM v2.0 到 v5.3 无兼容性变更，升级过程与正常升级无差异，仅仅是版本号上的增加。

1.2.2 改进提升

- 开启 Relay Log 同步时大幅度降低延迟 #2225
- 增量同步时压缩/合并 DML 语句，大幅度降低同步延迟 #3162 #3167
- 支持通过 OpenAPI 运维管理 DM 集群（实验特性）#1928
- 优化 dmctl 的使用体验并增加一些子命令 #1746
- 支持停止或暂停同步任务时保持事务原子性 #1928
- 支持读取文件名大于 999999 Relay Log 文件 #1933
- load 和 sync 处理单元支持更多的监控指标 #1778
- 支持通过 dmctl 并发操控同步任务 #1995
- 优化增量同步时 DML 并发度 #2043
- 检测到 HTTP 代理相关的环境变量时提示用户 #1960
- 优化处理 RowEvent 时的日志显示 #2006
- 优化 SQL 执行过慢时的日志显示 #2024
- 优化获取数据源状态数据的逻辑，减少对上游的压力 #2076
- 遇到不支持的 binlog 格式时，报错并提示用户 #2099
- 支持通过 dmctl 批量操作数据源里的所有同步任务 #2166
- 通过下游表结构来生成 DML WHERE 语句 #3168
- 支持自动获取和配置上下游的时区 #3403

1.2.3 Bug 修复

- 修复上下游配置 SSL 证书时高可用调度失败的问题 #1910
- 修复暂停任务耗时过多的问题 #1945
- 修复 handle-error revert 返回错误信息不明确的问题 #1969
- 修复使用 binlog filter 跳过某些 DDL 时同步任务失败的问题 #1975
- 修复 evict-leader 在某些情况下失效的问题 #1986
- 修复 dmctl 返回错误信息不明确的问题 #2063
- 修复开启 Relay Log 时 DM-worker 调度失败的问题 #2199
- 修复开启 Relay Log 时 DM-worker 不能连接上游而启动失败的问题 #2227
- 修复开启 Relay Log 且上游发生切换时 meta 文件写入失败的问题 #3164

1.2.4 已知问题

GitHub issues

1.3 基本功能

1.3.1 主要特性

本文档介绍 DM 提供的数据迁移功能以及相关的配置选项与使用示例。

Table Routing、Block & Allow Lists、Binlog Event Filter 在匹配库表名时，有以下版本差异：

- 对于 v1.0.5 版及后续版本，以上功能均支持通配符匹配。但注意所有版本中通配符匹配中的 * 符号 只能有一个且必须在末尾。
- 对于 v1.0.5 以前的版本，Table Routing 和 Binlog Event Filter 支持通配符，但不支持 [...] 与 [!...] 表达式。Block & Allow Lists 仅支持正则表达式。

在简单任务场景下推荐使用通配符匹配。

1.3.1.1 Table routing

Table routing 提供将上游 MySQL/MariaDB 实例的某些表迁移到下游指定表的功能。

> 注意：
>
> - 不支持对同一个表设置多个不同的路由规则。
> - Schema 的匹配规则需要单独设置，用来迁移 CREATE/DROP SCHEMA xx，例如下面参数配置中的 rule-2。

1.3.1.1.1 参数配置

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

1.3.1.1.2 参数解释

将根据 schema-pattern/table-pattern 匹配上该规则的上游 MySQL/MariaDB 实例的表迁移到下游的 target-schema/target-table。

1.3.1.1.3 使用示例

下面展示了三个不同场景下的配置示例。

分库分表合并

假设存在分库分表场景，需要将上游两个 MySQL 实例的表 test_{1,2,3...}.t_{1,2,3...} 迁移到下游 TiDB 的一张表 test.t。

为了迁移到下游实例的表 test.t，需要创建以下 table routing 规则：

- rule-1 用来迁移匹配上 schema-pattern: "test_*" 和 table-pattern: "t_*" 的表的 DML/DDL 语句到下游的 test.t。
- rule-2 用来迁移匹配上 schema-pattern: "test_*" 的库的 DDL 语句，例如 CREATE /DROP SCHEMA xx。

注意：

- 如果下游 TiDB schema: test 已经存在，并且不会被删除，则可以省略 rule-2。
- 如果下游 TiDB schema: test 不存在，只设置了 rule_1，则迁移会报错 schema test doesn't exist。

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

分库合并

假设存在分库场景，将上游两个 MySQL 实例 test_{1,2,3...}.t_{1,2,3...} 迁移到下游 TiDB 的 test.t_{1,2,3...}，创建一条路由规则即可：

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

错误的 table routing

假设存在下面两个路由规则，test_1_bak.t_1_bak 可以匹配上 rule-1 和 rule-2，违反 table 路由的限制而报错。

```
rule-0:
  schema-pattern: "test_*"
  target-schema: "test"
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"
```

1.3.1.2 Block & Allow Table Lists

上游数据库实例表的黑白名单过滤规则，可以用来过滤或者只迁移某些 database/table 的所有操作。

1.3.1.2.1 参数配置

```
block-allow-list:          # 如果 DM 版本早于 v2.0.0-beta.2 则使用 black-white-list。
  rule-1:
    do-dbs: ["test*"]      # 非 ~ 字符开头，表示规则是通配符；v1.0.5 及后续版本支持通配符规则。
    do-tables:
    - db-name: "test[123]" # 匹配 test1、test2、test3。
      tbl-name: "t[1-5]"   # 匹配 t1、t2、t3、t4、t5。
    - db-name: "test"
      tbl-name: "t"
  rule-2:
    do-dbs: ["~^test.*"]   # 以 ~ 字符开头，表示规则是正则表达式。
    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"
```

1.3.1.2.2 参数解释

- do-dbs：要迁移的库的白名单，类似于 MySQL 中的 replicate-do-db。
- ignore-dbs：要迁移的库的黑名单，类似于 MySQL 中的 replicate-ignore-db。
- do-tables：要迁移的表的白名单，类似于 MySQL 中的 replicate-do-table。必须同时指定 db-name 与 tbl-name。
- ignore-tables：要迁移的表的黑名单，类似于 MySQL 中的 replicate-ignore-table。必须同时指定 db-name 与 tbl-name。

以上参数值以 ~ 开头时均支持使用正则表达式来匹配库名、表名。

1.3.1.2.3 过滤规则

do-dbs 与 ignore-dbs 对应的过滤规则与 MySQL 中的 Evaluation of Database-Level Replication and Binary Logging Options 类似，do-tables 与 ignore-tables 对应的过滤规则与 MySQL 中的 Evaluation of Table-Level Replication Options 类似。

> **注意：**
>
> DM 中黑白名单过滤规则与 MySQL 中相应规则存在以下区别：
>
> - MySQL 中存在 replicate-wild-do-table 与 replicate-wild-ignore-table 用于支持通配符，DM 中各配置参数直接支持以 ~ 字符开头的正则表达式。
> - DM 当前只支持 ROW 格式的 binlog，不支持 STATEMENT/MIXED 格式的 binlog，因此应与 MySQL 中 ROW 格式下的规则对应。
> - 对于 DDL，MySQL 仅依据默认的 database 名称（USE 语句显式指定的 database）进行判断，而 DM 优先依据 DDL 中的 database 名称部分进行判断，并当 DDL 中不包含 database 名称时再依据 USE 部分进行判断。假设需要判断的 SQL 为 USE test_db_2; CREATE TABLE test_db_1.test_table (c1 INT PRIMARY KEY)，且 MySQL 配置了 replicate-do-db=test_db_1、DM 配置了 do-dbs: ["test_db_1"]，则对于 MySQL 该规则不会生效，而对于 DM 该规则会生效。

判断 table test.t 是否应该被过滤的流程如下：

1. 首先进行 **schema** 过滤判断
   - 如果 do-dbs 不为空，判断 do-dbs 中是否存在一个匹配的 schema。
     - 如果存在，则进入 **table** 过滤判断。
     - 如果不存在，则过滤 test.t。
   - 如果 do-dbs 为空并且 ignore-dbs 不为空，判断 ignore-dbs 中是否存在一个匹配的 schema。
     - 如果存在，则过滤 test.t。
     - 如果不存在，则进入 **table** 过滤判断。
   - 如果 do-dbs 和 ignore-dbs 都为空，则进入 **table** 过滤判断。
2. 进行 **table** 过滤判断
   1. 如果 do-tables 不为空，判断 do-tables 中是否存在一个匹配的 table。
      - 如果存在，则迁移 test.t。

- 如果不存在，则过滤 test.t。

2. 如果 ignore-tables 不为空，判断 ignore-tables 中是否存在一个匹配的 table。
   - 如果存在，则过滤 test.t.
   - 如果不存在，则迁移 test.t。
3. 如果 do-tables 和 ignore-tables 都为空，则迁移 test.t。

> **注意：**
>
> 如果是判断 schema test 是否应该被过滤，则只进行 **schema** 过滤判断。

1.3.1.2.4 使用示例

假设上游 MySQL 实例包含以下表：

```
`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`
```

配置如下：

```
block-allow-list: # 如果 DM 版本早于 v2.0.0-beta.2 则使用 black-white-list。
  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.*"
```

应用 bw-rule 规则后：

table	是否过滤	过滤的原因
logs	是	schema logs 没有匹配到 do-dbs 任意一项
logs	是	schema logs 没有匹配到 do-dbs 任意一项
logs	是	schema logs 没有匹配到 do-dbs 任意一项
forum_backup_2016	是	schema forum_backup_2016 没有匹配到 do-dbs 任意一项
forum_backup_2017	是	schema forum_backup_2017 没有匹配到 do-dbs 任意一项
forum	是	1. schema forum 匹配到 do-dbs，进入 table 过滤判断 2. schema 和 table 没有匹配到 do-tables 和 ignore-tables 中任意一项，并且 do-tables 不为空，因此过滤

table	是否过滤	过滤的原因
forum ↪ .messages ↪	否	1. schema forum 匹配到 do-dbs，进入 table 过滤判断 2. schema 和 table 匹配到 do-tables 的 db-name: "~^ ↪ forum.*", ↪ tbl-name: ↪ "messages"
forum_backup_2018 ↪ .messages ↪	否	1. schema forum_backup_2018 ↪ 匹配到 do-dbs，进入 table 过滤判断 2. schema 和 table 匹配到 do-tables 的 db-name: "~^ ↪ forum.*", ↪ tbl-name: ↪ "messages"

1.3.1.3 Binlog event filter

Binlog event filter 是比迁移表黑白名单更加细粒度的过滤规则，可以指定只迁移或者过滤掉某些 `schema / table` 的指定类型 binlog，比如 `INSERT`、`TRUNCATE TABLE`。

注意：

- 同一个表匹配上多个规则，将会顺序应用这些规则，并且黑名单的优先级高于白名单，即如果同时存在规则 `Ignore` 和 `Do` 应用在某个 table 上，那么 `Ignore` 生效。
- 从 DM v2.0.2 开始，Binlog event filter 也可以在上游数据库配置文件中进行配置。见上游数据库配置文件介绍。

1.3.1.3.1 参数配置

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

1.3.1.3.2 参数解释

- schema-pattern/table-pattern：对匹配上的上游 MySQL/MariaDB 实例的表的 binlog events 或者 DDL SQL 语句通过以下规则进行过滤。
- events：binlog events 数组，仅支持从以下 Event 中选择一项或多项。

Event	分类	解释
all		代表包含下面所有的 events
all dml		代表包含下面所有 DML events
all ddl		代表包含下面所有 DDL events
none		代表不包含下面所有 events
none ddl		代表不包含下面所有 DDL events
none dml		代表不包含下面所有 DML events
insert	DML	insert DML event
update	DML	update DML event
delete	DML	delete DML event
create database	DDL	create database event
drop database	DDL	drop database event
create table	DDL	create table event
create index	DDL	create index event
drop table	DDL	drop table event
truncate table	DDL	truncate table event
rename table	DDL	rename table event
drop index	DDL	drop index event
alter table	DDL	alter table event

- sql-pattern：用于过滤指定的 DDL SQL 语句，支持正则表达式匹配，例如上面示例中的 "^DROP\\s+PROCEDURE"。
- action：string (Do / Ignore)；进行下面规则判断，满足其中之一则过滤，否则不过滤。
  - Do：白名单。binlog event 如果满足下面两个条件之一就会被过滤掉：

    * 不在该 rule 的 events 中。
    * 如果规则的 sql-pattern 不为空的话，对应的 SQL 没有匹配上 sql-pattern 中任意一项。
  - Ignore：黑名单。如果满足下面两个条件之一就会被过滤掉：
    * 在该 rule 的 events 中。
    * 如果规则的 sql-pattern 不为空的话，对应的 SQL 可以匹配上 sql-pattern 中任意一项。

1.3.1.3.3 使用示例

过滤分库分表的所有删除操作

需要设置下面两个 Binlog event filter rule 来过滤掉所有的删除操作：

- filter-table-rule 过滤掉所有匹配到 pattern test_*.t_* 的 table 的 turncate table、drop table、delete statement 操作。
- filter-schema-rule 过滤掉所有匹配到 pattern test_* 的 schema 的 drop database 操作。

```
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
```

只迁移分库分表的 DML 操作

需要设置下面两个 Binlog event filter rule 只迁移 DML 操作：

- do-table-rule 只迁移所有匹配到 pattern test_*.t_* 的 table 的 create table、insert、update、delete 操作。
- do-schema-rule 只迁移所有匹配到 pattern test_* 的 schema 的 create database 操作。

> 注意：
>
> 迁移 create database/table 的原因是创建库和表后才能迁移 DML。

```
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
```

过滤 TiDB 不支持的 SQL 语句

可设置如下规则过滤 TiDB 不支持的 PROCEDURE 语句：

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

过滤 TiDB parser 不支持的 SQL 语句

对于 TiDB parser 不支持的 SQL 语句，DM 无法解析获得 schema/table 信息，因此需要使用全局过滤规则：schema-pattern: "*"。

> 注意：
> 全局过滤规则的设置必须尽可能严格，以避免过滤掉需要迁移的数据。

可设置如下规则过滤某些版本的 TiDB parser 不支持的 PARTITION 语句：

```
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
```

1.3.1.4 online DDL 工具支持

在 MySQL 生态中，gh-ost 与 pt-osc 等工具较广泛地被使用，DM 对其提供了特殊的支持以避免对不必要的中间数据进行迁移。

有关 DM 对 online DDL 工具支持的原理、处理流程等，可参考online-ddl。

1.3.1.4.1 使用限制

- DM 仅针对 gh-ost 与 pt-osc 做了特殊支持。
- 在开启 `online-ddl` 时，增量复制对应的 checkpoint 应不处于 online DDL 执行过程中。如上游某次 online DDL 操作开始于 binlog `position-A`、结束于 `position-B`，则增量复制的起始点应早于 `position-A` 或晚于 `position-B`，否则可能出现迁移出错，具体可参考FAQ。

1.3.1.4.2 参数配置

在 v2.0.5 及之后的版本，请在 `task` 配置文件中使用 `online-ddl` 配置项。

如上游 MySQL/MariaDB （同时）使用 gh-ost 或 pt-osc 工具，则在 task 的配置文件中设置：

```
online-ddl: true
```

> **注意：**
>
> 自 v2.0.5 起，`online-ddl-scheme` 已被弃用，请使用 `online-ddl` 代替 `online-ddl-scheme`。如设置 `online-ddl: true` 会覆盖掉 `online-ddl-scheme`。如设置 `online-ddl-scheme: "pt"` 或 `online-ddl-scheme: "gh-ost"` 会被转换为 `online-ddl: true`。

在 v2.0.5 之前的版本（不含 v2.0.5），请在 `task` 配置文件中使用 `online-ddl-scheme` 配置项。

如上游 MySQL/MariaDB 使用的是 gh-ost 工具，则在 task 的配置文件中设置：

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

如上游 MySQL/MariaDB 使用的是 pt-osc 工具，则在 task 的配置文件中设置：

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

1.3.1.5 分库分表合并

DM 支持将上游 MySQL/MariaDB 各分库分表中的 DML、DDL 数据合并后迁移到下游 TiDB 的库表中。

1.3.1.5.1 使用限制

目前分库分表合并功能仅支持有限的场景，使用该功能前，请仔细阅读悲观模式分库分表合并迁移使用限制和乐观模式分库分表合并迁移使用限制。

1.3.1.5.2 参数配置

在 task 的配置文件中设置：

```
shard-mode: "pessimistic" # 默认值为 "" 即无需协调。如果为分库分表合并任务，
    ↪ 请设置为悲观协调模式 "pessimistic"。
    ↪ 在深入了解乐观协调模式的原理和使用限制后，也可以设置为乐观协调模式 "
    ↪ optimistic"
```

1.3.1.5.3 手动处理 Sharding DDL Lock

如果分库分表合并迁移过程中发生了异常，对于部分场景，可尝试参考手动处理 Sharding DDL Lock进行处理。

1.4 高级功能

1.4.1 分库分表合并迁移

1.4.1.1 分库分表合并迁移

本文介绍了 DM 提供的分库分表的合并迁移功能，此功能可用于将上游 MySQL/MariaDB 实例中结构相同/不同的表迁移到下游 TiDB 的同一个表中。DM 不仅支持迁移上游的 DML 数据，也支持协调迁移多个上游分表的 DDL 表结构变更。

1.4.1.1.1 简介

DM 支持对上游多个分表的数据合并迁移到 TiDB 的一个表中，在迁移过程中需要协调各个分表的 DDL，以及该 DDL 前后的 DML。针对用户的使用场景，DM 支持悲观协调和乐观协调两种不同的模式。

注意：

- 要执行分库分表合并迁移任务，必须在任务配置文件中设置 shard-mode。
- DM 对分库分表的合并默认使用悲观协调模式，在文档中如果没特殊说明则默认为悲观协调模式。
- 在没有深入了解乐观模式的原理和使用限制的情况下不建议使用该模式，否则可能造成迁移中断甚至数据不一致的严重后果。

悲观协调模式

当上游一个分表执行某一 DDL 后，这个分表的迁移会暂停，并等待其他所有分表都执行了同样的 DDL 才在下游执行该 DDL 并继续数据迁移。“悲观协调” 模式的优点是可以保证迁移到下游的数据不会出错，详细的介绍请参考悲观模式下分库分表合并迁移。

乐观协调模式

在一个分表上执行的 DDL，DM 会自动修改成兼容其他分表的语句，并立即迁移到下游，不会阻挡任何分表 DML 的迁移。“乐观协调” 模式的优点是处理 DDL 时不会阻塞数据的迁移，但是使用不当会有迁移中断甚至数据不一致的风险，详细的介绍请参考乐观模式下分库分表合并迁移。

悲观协调模式与乐观协调模式的对比

悲观协调模式	乐观协调模式
发起 DDL 的分表会暂停 DML 迁移	发起 DDL 的分表会继续 DML 迁移
每个分表的 DDL 执行次序和语句必须相同	每个分表只需保持表结构互相兼容即可
DDL 在整个分表群达成一致后才迁移到下游	每个分表的 DDL 会即时影响下游
错误的 DDL 操作在侦测到后可以被拦截	错误的 DDL 操作也会被迁移到下游，可能在侦测到之前

1.4.1.2 悲观模式下分库分表合并迁移

本文介绍了 DM 提供的悲观模式（默认模式）下分库分表合并迁移功能。此功能用于将上游 MySQL/MariaDB 实例中结构相同的表迁移到下游 TiDB 的同一个表中。

1.4.1.2.1 使用限制

DM 在悲观模式下进行分表 DDL 的迁移有以下几点使用限制：

- 对于一个逻辑 sharding group（需要合并迁移到下游同一个表的所有分表组成的 group），只能使用一个仅包含这些分表所在数据源的任务进行迁移。
- 在一个逻辑 sharding group 内，所有上游分表必须以相同的顺序执行相同的 DDL 语句（库名和表名可以不同），并且只有在所有分表执行完当前一条 DDL 语句后，下一条 DDL 语句才能执行。
  - 比如，如果在 table_1 表中先增加列 a 后再增加列 b，则在 table_2 表中就不能先增加列 b 后再增加列 a，因为 DM 不支持以不同的顺序来执行相同的 DDL 语句。
- 在一个逻辑 sharding group 内，所有上游分表都应该执行对应的 DDL 语句。
  - 比如，若 DM-worker-2 对应的一个或多个上游分表未执行 DDL 语句，则其他已执行 DDL 语句的 DM-worker 都会暂停迁移任务，直到等到 DM-worker-2 收到上游对应的 DDL 语句。
- sharding group 数据迁移任务不支持 `DROP DATABASE/TABLE` 语句。

  - DM-worker 中的 binlog 复制单元（sync）会自动忽略掉上游分表的 DROP DATABASE 和 DROP TABLE 语句。

- sharding group 数据迁移任务不支持 TRUNCATE TABLE 语句。

  - DM-worker 中的 binlog 复制单元（sync）会自动忽略掉上游分表的 TRUNCATE TABLE 语句。

- sharding group 数据迁移任务支持 RENAME TABLE 语句，但有如下限制（online DDL 中的 RENAME 有特殊方案进行支持）：

  - 只支持 RENAME TABLE 到一个不存在的表。
  - 一条 RENAME TABLE 语句只能包含一个 RENAME 操作。

- sharding group 数据迁移任务要求一个 DDL 语句仅包含对一张表的操作。
- 增量复制任务需要确认开始迁移的 binlog position 上各分表的表结构必须一致，才能确保来自不同分表的 DML 语句能够迁移到表结构确定的下游，并且后续各分表的 DDL 语句能够正确匹配与迁移。
- 如果需要变更table routing 规则，必须先等所有 sharding DDL 语句迁移完成。

  - 在 sharding DDL 语句迁移过程中，使用 dmctl 尝试变更 router-rules 会报错。

- 如果需要创建新表加入到一个正在执行 DDL 语句的 sharding group 中，则必须保持新表结构和最新更改的表结构一致。

  - 比如，原 table_1, table_2 表初始时有 (a, b) 两列，sharding DDL 语句执行后有 (a, b, c) 三列，则迁移完成后新创建的表也应当有 (a, b, c) 三列。

- 由于已经收到 DDL 语句的 DM-worker 会暂停任务以等待其他 DM-worker 收到对应的 DDL 语句，因此数据迁移延迟会增加。

1.4.1.2.2 背景

目前，DM 使用 ROW 格式的 binlog 进行数据迁移，且 binlog 中不包含表结构信息。在 ROW 格式的 binlog 迁移过程中，如果不需要将多个上游表合并迁移到下游的同一个表，则只存在一个上游表的 DDL 语句会更新对应下游表结构。ROW 格式的 binlog 可以认为是具有 self-description 属性。

分库分表合并迁移过程中，可以根据 column values 及下游的表结构构造出相应的 DML 语句，但此时若上游的分表执行 DDL 语句进行了表结构变更，则必须对该 DDL 语句进行额外迁移处理，以避免因为表结构和 binlog 数据不一致而造成迁移出错的问题。

以下是一个简化后的例子：

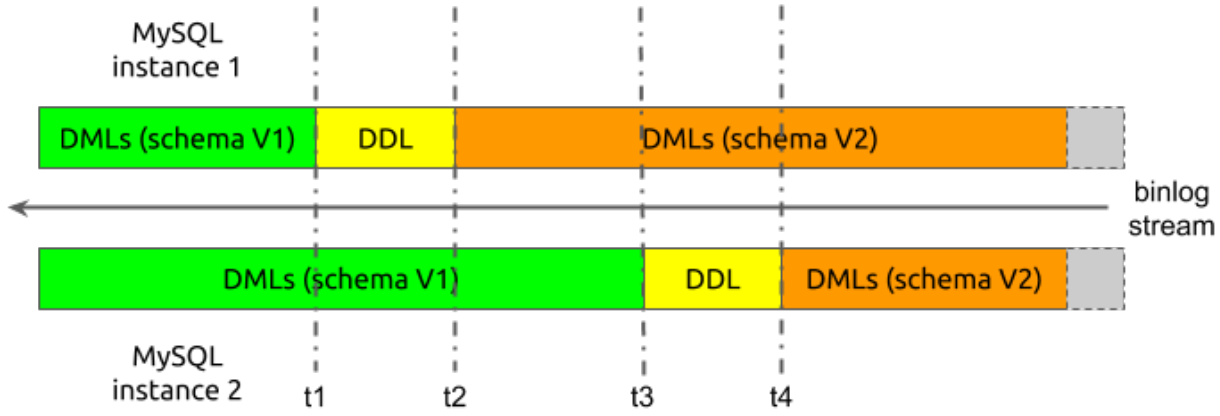


Figure 2: shard-ddl-example-1

在上图的例子中，分表的合库合表过程简化成了上游只有两个 MySQL 实例，每个实例内只有一个表。假设在数据迁移开始时，将两个分表的表结构版本记为 schema V1，将 DDL 语句执行完后的表结构版本记为 schema V2。

现在，假设数据迁移过程中，DM-worker 内的 binlog 复制单元（sync）从两个上游分表收到的 binlog 数据有如下时序：

1. 开始迁移时，sync 从两个分表收到的都是 schema V1 版本的 DML 语句。
2. 在 t1 时刻，sync 收到实例 1 上分表的 DDL 语句。
3. 从 t2 时刻开始，sync 从实例 1 收到的是 schema V2 版本的 DML 语句；但从实例 2 收到的仍是 schema V1 版本的 DML 语句。
4. 在 t3 时刻，sync 收到实例 2 上分表的 DDL 语句。
5. 从 t4 时刻开始，sync 从实例 2 收到的也是 schema V2 版本的 DML 语句。

假设在数据迁移过程中，不对分表的 DDL 语句进行额外处理。当实例 1 的 DDL 语句迁移到下游后，下游的表结构会变更成为 schema V2 版本。但在 t2 到 t3 这段时间内，sync 从实例 2 上收到的仍是 schema V1 版本的 DML 语句。当尝试把这些 schema V1 版本的 DML 语句迁移到下游时，就会由于 DML 语句与表结构的不一致而发生错误，从而无法正确迁移数据。

1.4.1.2.3 实现原理

基于上述例子，本部分介绍了 DM 在默认的悲观模式下合库合表过程中进行 DDL 迁移的实现原理。

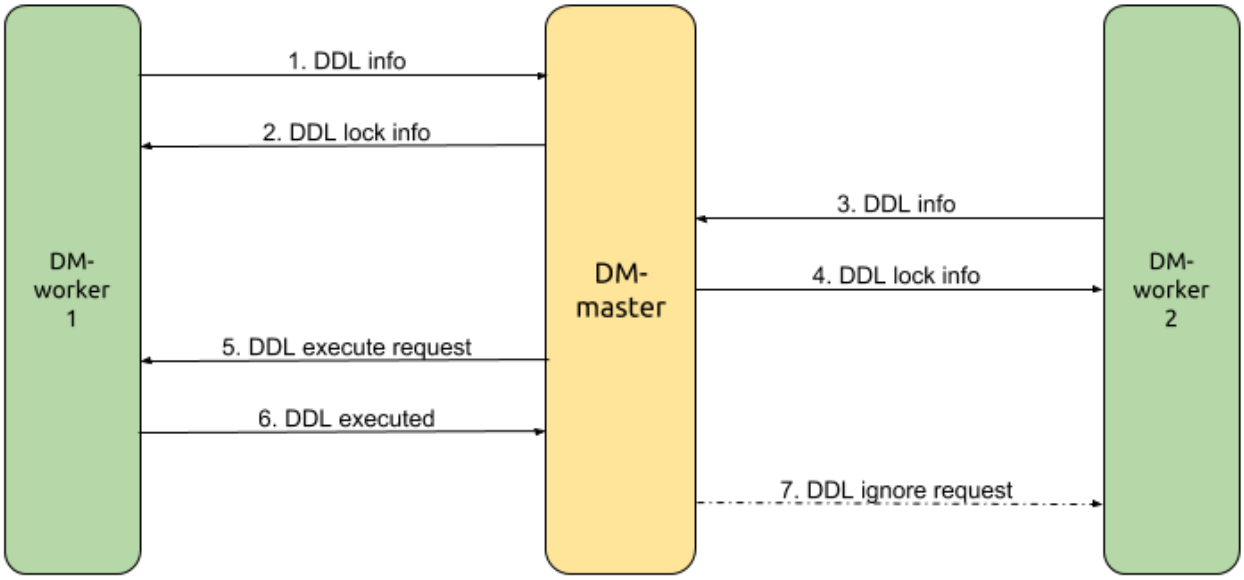


Figure 3: shard-ddl-flow

在这个例子中，DM-worker-1 负责迁移来自 MySQL 实例 1 的数据，DM-worker-2 负责迁移来自 MySQL 实例 2 的数据，DM-master 负责协调多个 DM-worker 间的 DDL 迁移。

从 DM-worker-1 收到 DDL 语句开始，简化后的 DDL 迁移流程为：

1. 在 t1 时刻，DM-worker-1 收到来自 MySQL 实例 1 的 DDL 语句，自身暂停该 DDL 语句对应任务的 DDL 及 DML 数据迁移，并将 DDL 相关信息发送给 DM-master。
2. DM-master 根据收到的 DDL 信息判断得知需要协调该 DDL 语句的迁移，于是为该 DDL 语句创建一个锁，并将 DDL 锁信息发回给 DM-worker-1，同时将 DM-worker-1 标记为这个锁的 owner。
3. DM-worker-2 继续进行 DML 语句的迁移，直到在 t3 时刻收到来自 MySQL 实例 2 的 DDL 语句，自身暂停该 DDL 语句对应任务的数据迁移，并将 DDL 相关信息发送给 DM-master。
4. DM-master 根据收到的 DDL 信息判断得知该 DDL 语句对应的锁信息已经存在，于是直接将对应锁信息发回给 DM-worker-2。
5. 根据任务启动时的配置信息、上游 MySQL 实例分表信息、部署拓扑信息等，DM-master 判断得知自身已经收到了来自待合表的所有上游分表的 DDL 语句，于是请求 DDL 锁的 owner（DM-worker-1）向下游迁移执行该 DDL。
6. DM-worker-1 根据第二步收到的 DDL 锁信息验证 DDL 语句执行请求；向下游执行 DDL，并将执行结果反馈给 DM-master；若 DDL 语句执行成功，则自身开始继续迁移后续的（从 t2 时刻对应的 binlog 开始的）DML 语句。

7. DM-master 收到来自 owner 执行 DDL 语句成功的响应，于是请求在等待该 DDL 锁的所有其他 DM-worker（DM-worker-2）忽略该 DDL 语句，直接继续迁移后续的（从 t4 时刻对应的 binlog 开始的）DML 语句。

根据上面的流程，可以归纳出 DM 协调多个 DM-worker 间 sharding DDL 迁移的特点：

- 根据任务配置与 DM 集群部署拓扑信息，DM-master 内部也会建立一个逻辑 sharding group 来协调 DDL 迁移，group 中的成员为负责处理该迁移任务拆解后的各子任务的 DM-worker。
- 各 DM-worker 从 binlog event 中收到 DDL 语句后，会将 DDL 信息发送给 DM-master。
- DM-master 根据来自 DM-worker 的 DDL 信息及 sharding group 信息创建或更新 DDL 锁。
- 如果 sharding group 的所有成员都收到了某一条相同的 DDL 语句，则表明上游分表在该 DDL 执行前的 DML 语句都已经迁移完成，此时可以执行该 DDL 语句，并继续后续的 DML 迁移。
- 上游所有分表的 DDL 在经过table router 转换后需要保持一致，因此仅需 DDL 锁的 owner 执行一次该 DDL 语句即可，其他 DM-worker 可直接忽略对应的 DDL 语句。

在上面的示例中，每个 DM-worker 对应的上游 MySQL 实例中只有一个待合并的分表。但在实际场景下，一个 MySQL 实例可能有多个分库内的多个分表需要进行合并，这种情况下，sharding DDL 的协调迁移过程将更加复杂。

假设同一个 MySQL 实例中有 table_1 和 table_2 两个分表需要进行合并：

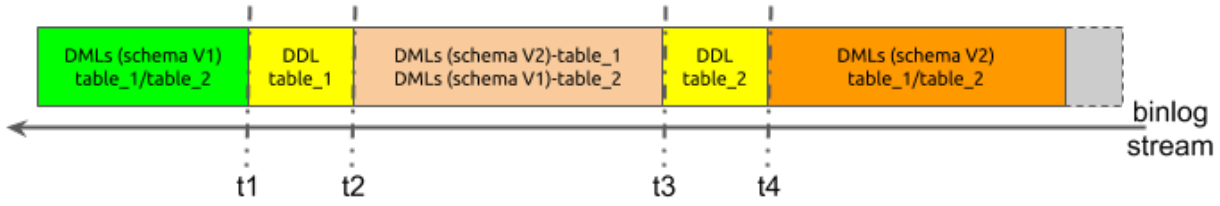


Figure 4: shard-ddl-example-2

在这个例子中，由于数据来自同一个 MySQL 实例，因此所有数据都是从同一个 binlog 流中获得，时序如下：

1. 开始迁移时，DM-worker 内的 sync 从两个分表收到的数据都是 schema V1 版本的 DML 语句。
2. 在 t1 时刻，sync 收到 table_1 分表的 DDL 语句。

3. 从 t2 到 t3 时刻，sync 收到的数据同时包含 table_1 的 DML 语句（schema V2 版本）及 table_2 的 DML 语句（schema V1 版本）。
4. 在 t3 时刻，sync 收到 table_2 分表的 DDL 语句。
5. 从 t4 时刻开始，sync 从两个分表收到的数据都是 schema V2 版本的 DML 语句。

假设在数据迁移过程中，不对分表的 DDL 语句进行额外处理。当 table_1 的 DDL 语句迁移到下游从而变更下游表结构后，table_2 的 DML 语句（schema V1 版本）将无法正常迁移。因此，在单个 DM-worker 内部，我们也构造了与 DM-master 内类似的逻辑 sharding group，但 group 的成员是同一个上游 MySQL 实例的不同分表。

DM-worker 内协调处理 sharding group 的迁移与 DM-master 处理 DM-worker 之间的迁移不完全一致，主要原因包括：

- 当 DM-worker 收到 table_1 分表的 DDL 语句时，迁移不能暂停，需要继续解析 binlog 才能获得后续 table_2 分表的 DDL 语句，即需要从 t2 时刻继续解析直到 t3 时刻。
- 在继续解析 t2 到 t3 时刻的 binlog 的过程中，table_1 分表的 DML 语句（schema V2 版本）不能向下游迁移；但当 sharding DDL 迁移并执行成功后，这些 DML 语句则需要迁移到下游。

DM-worker 内部 sharding DDL 迁移的简化流程为：

1. 在 t1 时刻，DM-worker 收到 table_1 的 DDL 语句，并记录 DDL 信息及此时的 binlog 位置点信息。
2. DM-worker 继续向前解析 t2 到 t3 时刻的 binlog。
3. 对于 table_1 的 DML 语句（schema V2 版本），忽略；对于 table_2 的 DML 语句（schema V1 版本），正常迁移到下游。
4. 在 t3 时刻，DM-worker 收到 table_2 的 DDL 语句，并记录 DDL 信息及此时的 binlog 位置点信息。
5. 根据迁移任务配置信息、上游库表信息等，DM-worker 判断得知该 MySQL 实例上所有分表的 DDL 语句都已收到；于是将该 DDL 语句迁移到下游执行并变更下游表结构。
6. DM-worker 设置 binlog 流的新解析起始位置点为第一步时保存的位置点。
7. DM-worker 重新开始解析从 t2 到 t3 时刻的 binlog。
8. 对于 table_1 的 DML 语句（schema V2 版本），正常迁移到下游；对于 table_2 的 DML 语句（schema V1 版本），忽略。

9. 解析到达第四步时保存的 binlog 位置点，可得知在第三步时被忽略的所有 DML 语句都已经重新迁移到下游。
10. DM-worker 继续从 t4 时刻对应的 binlog 位置点开始正常迁移。

综上可知，DM 在处理 sharding DDL 迁移时，主要通过两级 sharding group 来进行协调控制，简化的流程为：

1. 各 DM-worker 独立地协调对应上游 MySQL 实例内多个分表组成的 sharding group 的 DDL 迁移。
2. 当 DM-worker 收到所有分表的 DDL 语句时，向 DM-master 发送 DDL 相关信息。
3. DM-master 根据 DM-worker 发来的 DDL 信息，协调由各 DM-worker 组成的 sharing group 的 DDL 迁移。
4. 当 DM-master 收到所有 DM-worker 的 DDL 信息时，请求 DDL 锁的 owner（某个 DM-worker）执行该 DDL 语句。
5. DDL 锁的 owner 执行 DDL 语句，并将结果反馈给 DM-master；自身开始重新迁移在内部协调 DDL 迁移过程中被忽略的 DML 语句。
6. 当 DM-master 收到 owner 执行 DDL 成功的消息后，请求其他所有 DM-worker 继续开始迁移。
7. 其他所有 DM-worker 各自开始重新迁移在内部协调 DDL 迁移过程中被忽略的 DML 语句。
8. 在完成被忽略的 DML 语句的重新迁移后，所有 DM-worker 继续正常迁移。

1.4.1.3 乐观模式下分库分表合并迁移

本文介绍了 DM 提供的乐观模式下分库分表的合并迁移功能，此功能可用于将上游 MySQL/MariaDB 实例中结构相同/不同的表迁移到下游 TiDB 的同一个表中。

> **注意：**
>
> 在没有深入了解乐观模式的原理和使用限制的情况下不建议使用该模式，否则可能造成迁移中断甚至数据不一致的严重后果。

1.4.1.3.1 背景

DM 支持在线上执行分库分表的 DDL 语句（通称 Sharding DDL），默认使用“悲观模式”，即当上游一个分表执行某一 DDL 后，这个分表的迁移会暂停，等待其他所有分表都执行了同样的 DDL 才在下游执行该 DDL 并继续数据迁移。这种“悲观协调”模式的优点是可以保证迁移到下游的数据不会出错，缺点是会暂停数据迁移而不利于对上游进行灰度变更。有些用户可能会花较长时间在单一分表执行 DDL，验证一定时间后才会更改其他分表的结构。在悲观模式迁移的设定下，这些 DDL 会阻塞迁移，binlog 事件会大量积压。

因此，需要提供一种新的“乐观协调”模式，在一个分表上执行的 DDL，自动修改成兼容其他分表的语句后，立即迁移到下游，不会阻挡任何分表执行的 DML 的迁移。

1.4.1.3.2 乐观协调模式的配置

在任务的配置文件中指定 `shard-mode` 为 `optimistic` 则使用“乐观协调”模式，示例配置文件可以参考 DM 任务完整配置文件介绍。

1.4.1.3.3 使用限制

使用“乐观协调”模式有一定的风险，需要严格遵照以下方针：

- 执行每个批次的 DDL 前和后，要确保每个分表的结构达成一致。
- 进行灰度 DDL 时，只集中在一个分表上测试。
- 灰度完成后，在其他分表上尽量以最简单直接的 DDL 迁移到最终的 schema，而不要重新执行灰度测试中对或错的每一步。
  - 例如：在分表执行过 `ADD COLUMN A INT; DROP COLUMN A; ADD COLUMN A FLOAT;`，在其他分表直接执行 `ADD COLUMN A FLOAT` 即可，不需要三条 DDL 都执行一遍。
- 执行 DDL 时要注意观察 DM 迁移状态。当迁移报错时，需要判断这个批次的 DDL 是否会造成数据不一致。

“乐观协调”模式暂不支持以下语句：

- `ALTER TABLE table_name ADD COLUMN column_name datatype NOT NULL`（添加无默认值的 not null 的列）。
- `ALTER TABLE table_name ADD COLUMN column_name datetime DEFAULT NOW()`（增加的列默认值不固定）。
- `ALTER TABLE table_name ADD COLUMN col1 INT, DROP COLUMN col2`（在一个 DDL 语句中同时包含 `ADD COLUMN` 与 `DROP COLUMN`）。
- `ALTER TABLE table_name RENAME COLUMN column_1 TO column_2;`（重命名列）。
- `ALTER TABLE table_name RENAME INDEX index_1 TO index_2;`（重命名索引）。

此外，不论是使用“乐观协调”或“悲观协调”，DM 仍是有以下限制：

- 增量复制任务需要确保开始迁移的 binlog position 对应的各分表的表结构必须一致。

- 进入 sharding group 的新表必须与其他成员的表结构一致（正在执行一个 DDL 批次时禁止 `CREATE/RENAME TABLE`）。
- 不支持 `DROP TABLE/DROP DATABASE`。
- 不支持 `TRUNCATE TABLE`。
- 单条 DDL 语句要求仅包含对一张表的操作。
- TiDB 不支持的 DDL 语句在 DM 也不支持。
- 新增列的默认值不能包含 `current_timestamp`、`rand()`、`uuid()` 等，否则会造成上下游数据不一致。

1.4.1.3.4 风险

使用乐观模式迁移时，由于 DDL 会即时迁移到下游，若使用不当，可能导致上下游数据不一致。

使数据不一致的操作

- 各分表的表结构不兼容，例：
  - 两个分表各自添加相同名称的列，但其类型不同。
  - 两个分表各自添加相同名称的列，但其默认值不同。
  - 两个分表各自添加相同名称的生成列，但其生成表达式不同。
  - 两个分表各自添加相同名称的索引，但其键组合不同。
  - 其他同名异构的情况。
- 在分表上执行对数据具有破坏性的 DDL，然后尝试回滚，例：
  - 删除一列 X，之后又把 X 加回来。

例子

例如以下三个分表合并迁移到 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

在 tbl01 新增一列 Age，默认值定为 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

在 tbl00 新增一列 Age，但默认值定为 -1：

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

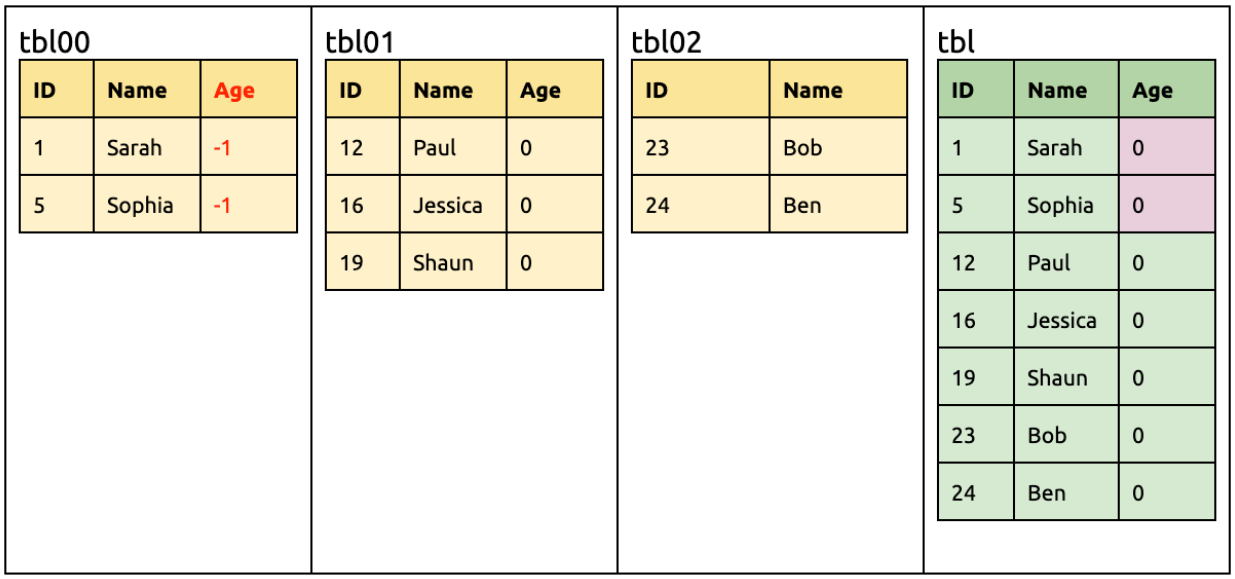


Figure 7: optimistic-ddl-fail-example-3

此时所有来自 `tbl00` 的 `Age` 都不一致了。这是由于 `DEFAULT 0` 和 `DEFAULT -1` 互不兼容。虽然 DM 遇到这种情况会报错，但上下游不一致的问题就需要手动去解决。

1.4.1.3.5 原理

在“乐观协调”模式下，DM-worker 接收到来自上游的 DDL 后，会把更新后的表结构转送给 DM-master。DM-worker 会追踪各分表当前的表结构，DM-master 合并成可兼容来自每个分表 DML 的合成结构，然后把与此对应的 DDL 迁移到下游；对于 DML 会直接迁移到下游。

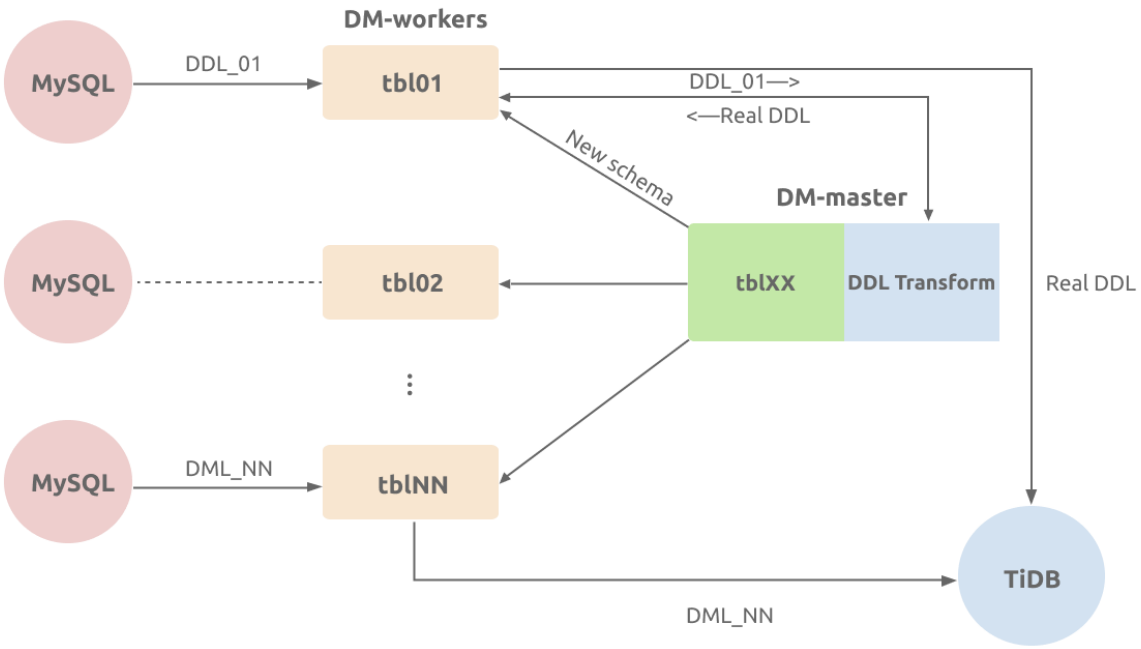


Figure 8: optimistic-ddl-flow

例子

例如上游 MySQL 有三个分表（`tbl00`, `tbl01` 以及 `tbl02`），使用 DM 迁移到下游 TiDB 的 `tbl` 表中，如下图所示：

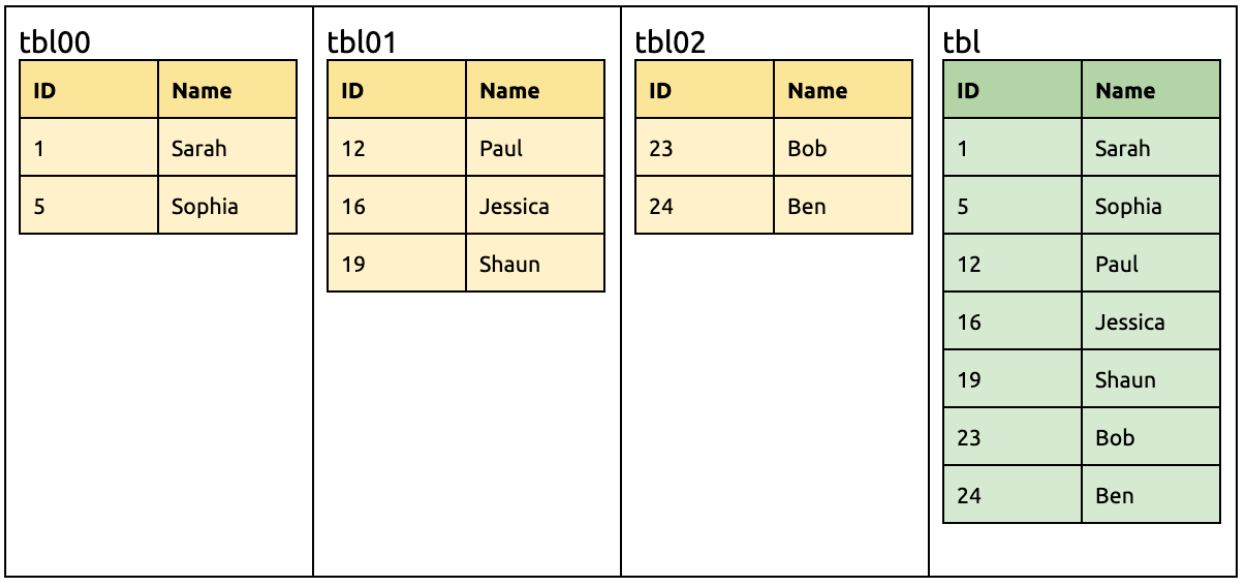


Figure 9: optimistic-ddl-example-1

在上游增加一列 `Level`：

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

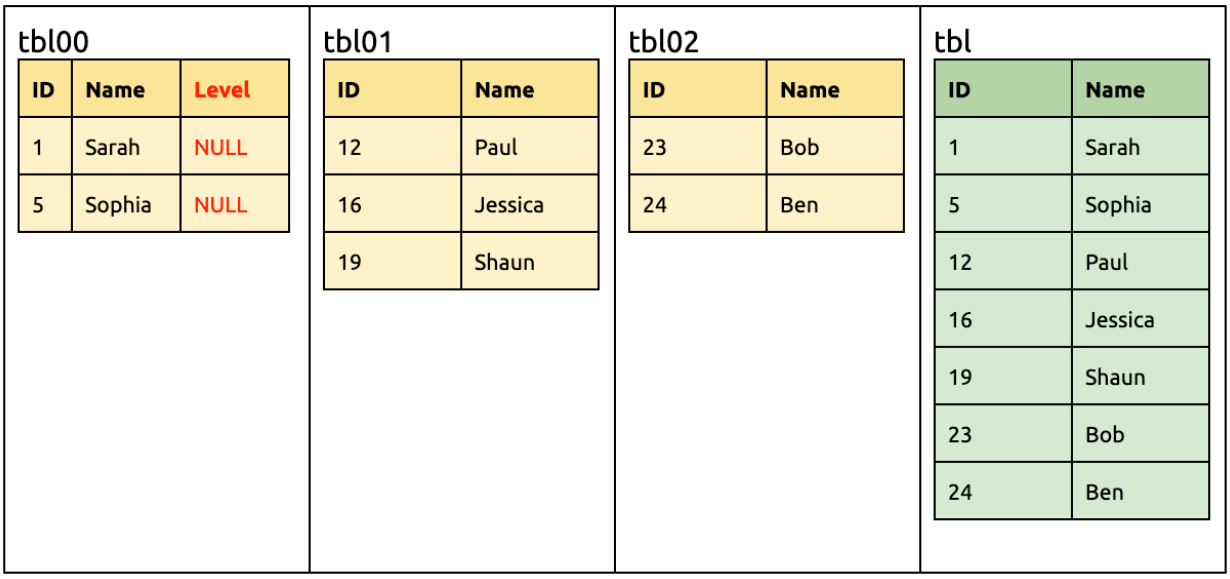


Figure 10: optimistic-ddl-example-2

此时下游 TiDB 要准备接受来自 `tbl00` 有 `Level` 的 DML、以及来自 `tbl01` 和 `tbl02` 没有 `Level` 的 DML。

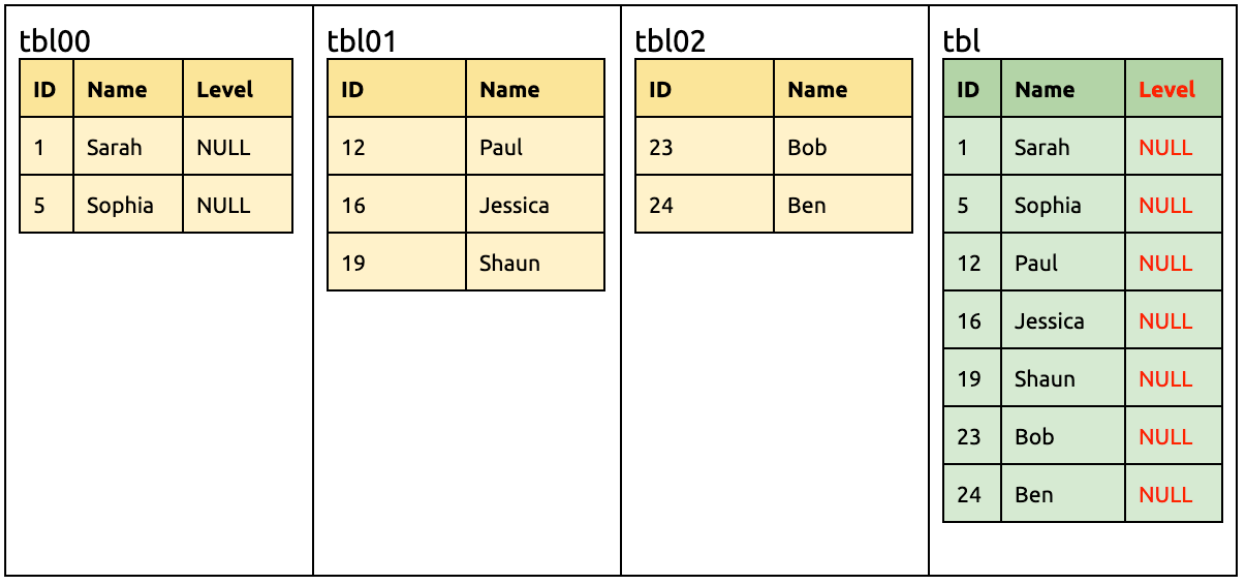


Figure 11: optimistic-ddl-example-3

这时候如下的 DML 无需修改就可以迁移到下游：

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

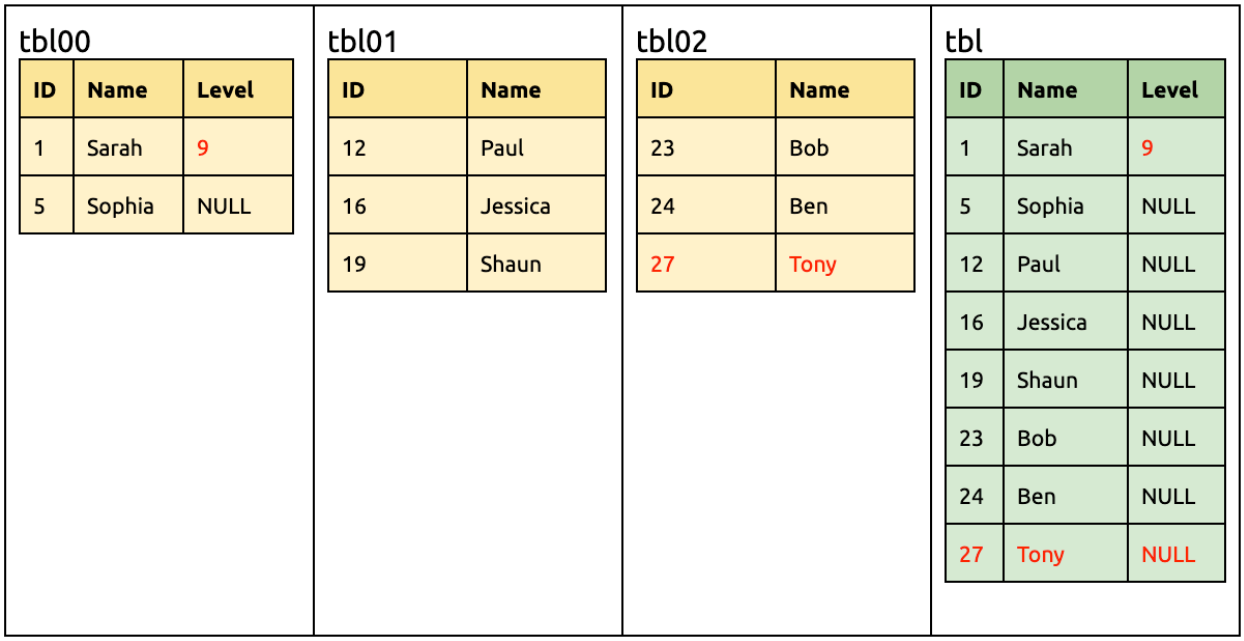


Figure 12: optimistic-ddl-example-4

在 tbl01 同样增加一列 Level：

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

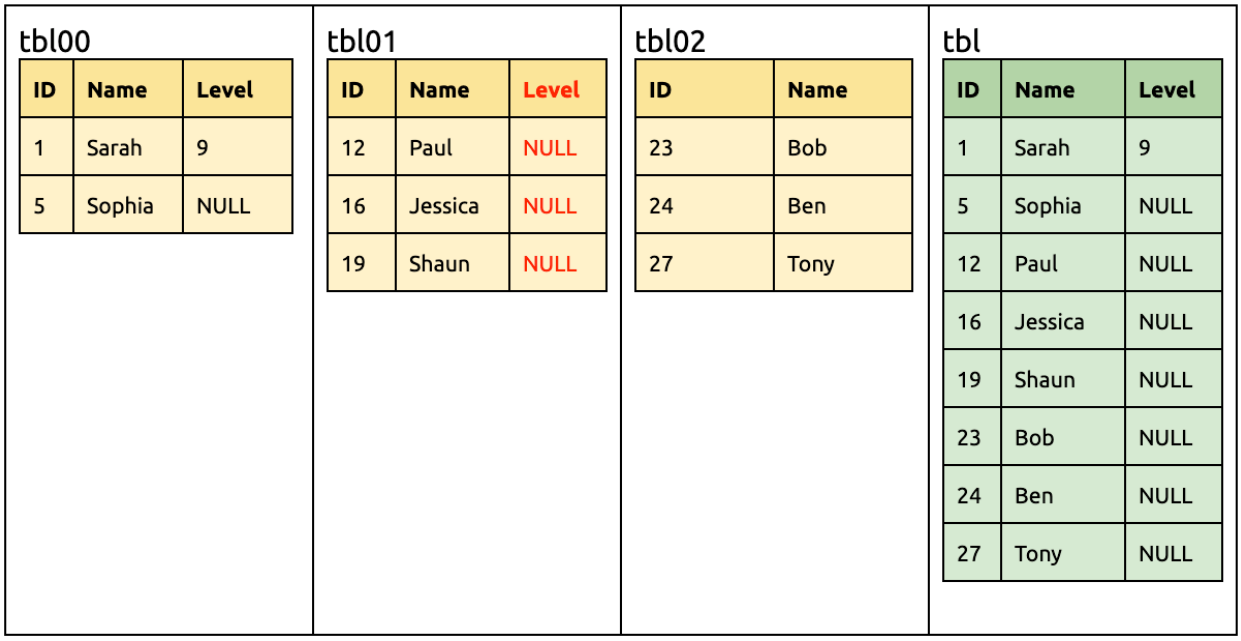


Figure 13: optimistic-ddl-example-5

此时下游已经有相同的 Level 列了，所以 DM-master 比较表结构之后不做任何操作。

在 tbl01 删除一列 Name：

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

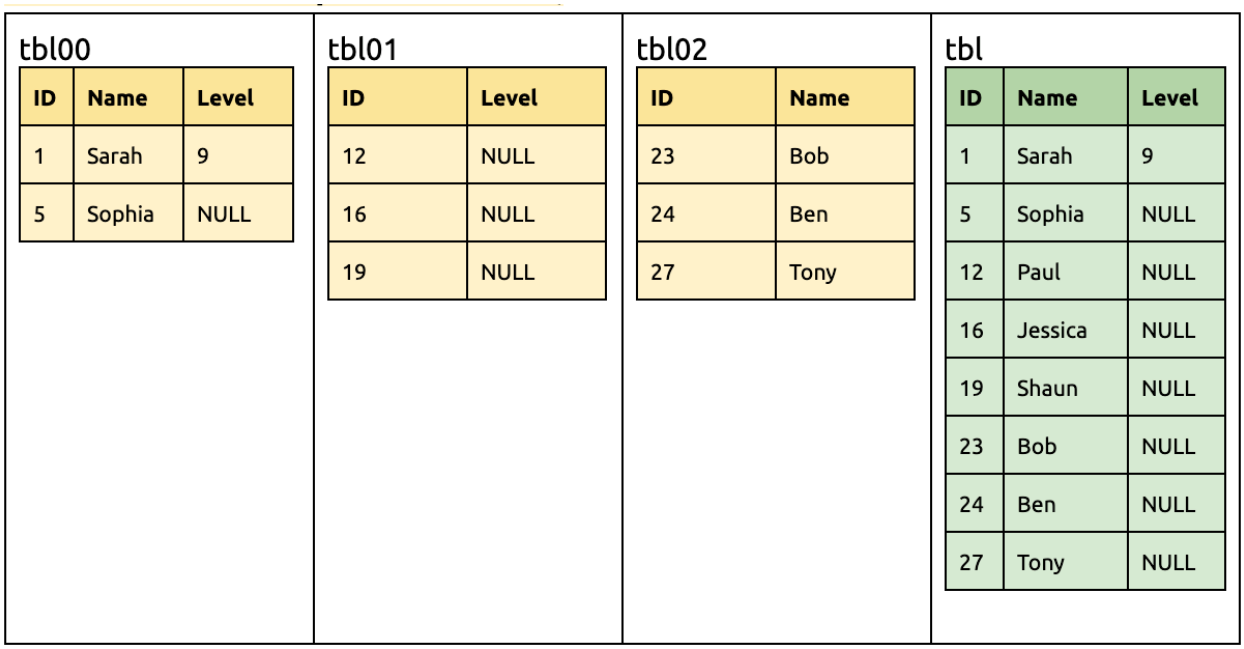


Figure 14: optimistic-ddl-example-6

此时下游仍需要接收来自 `tbl00` 和 `tbl02` 含 `Name` 的 DML 语句，因此不会立刻删除该列。

同样，各种 DML 仍可直接迁移到下游：

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

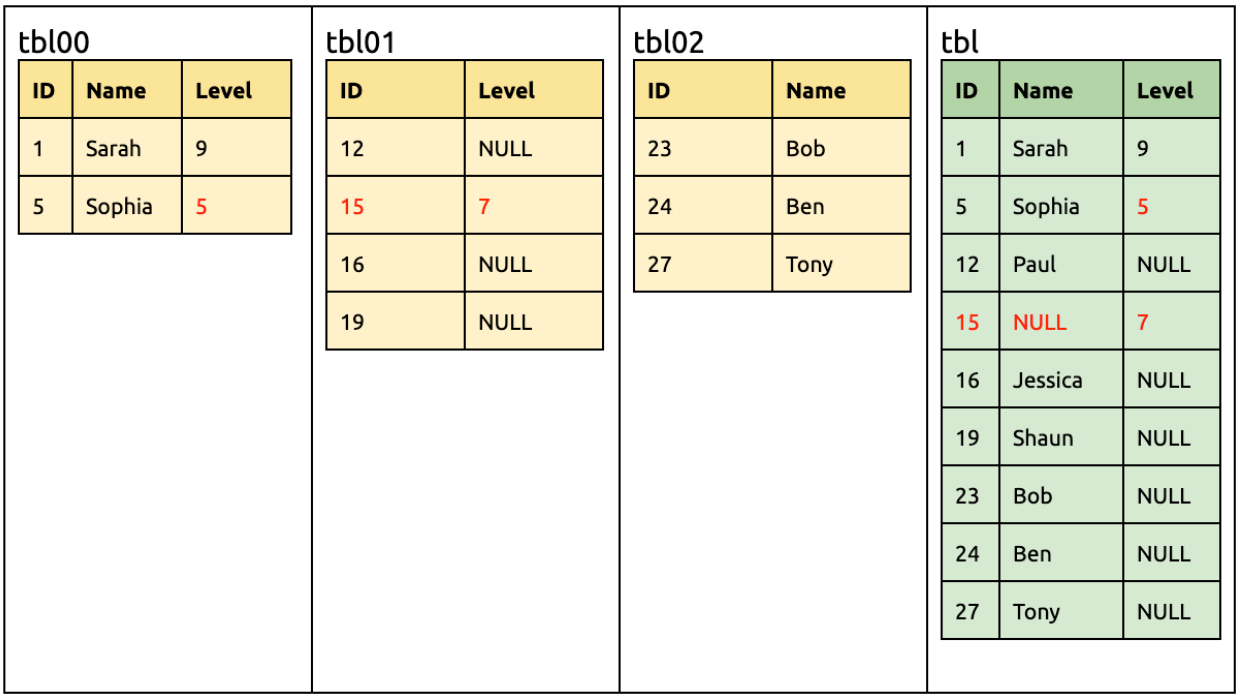


Figure 15: optimistic-ddl-example-7

在 `tbl02` 增加一列 `Level`：

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

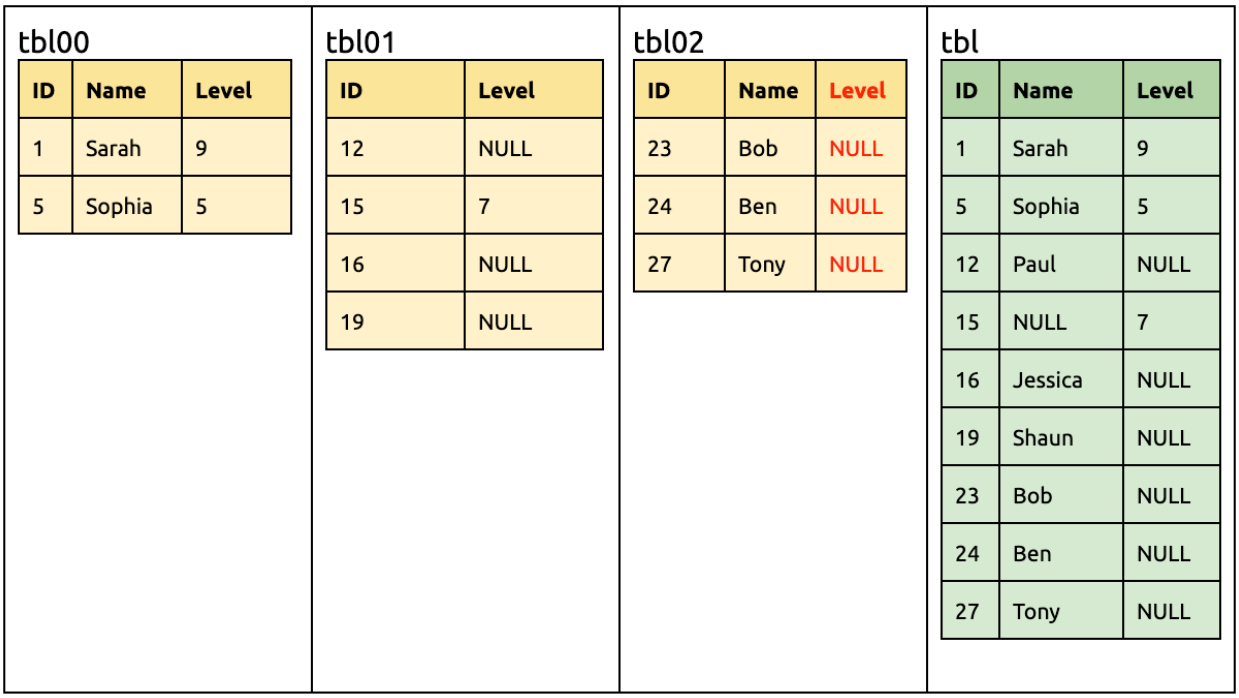


Figure 16: optimistic-ddl-example-8

此时所有分表都已有 `Level` 列。

在 `tbl00` 和 `tbl02` 各删除一列 `Name`：

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

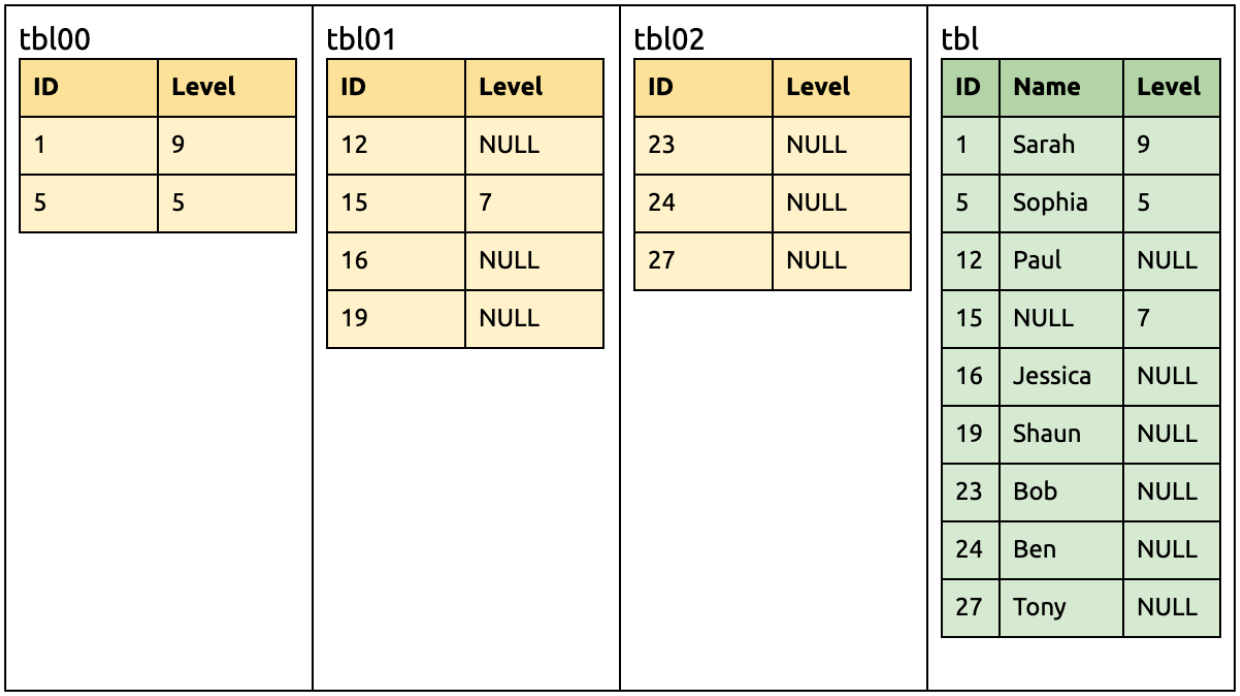
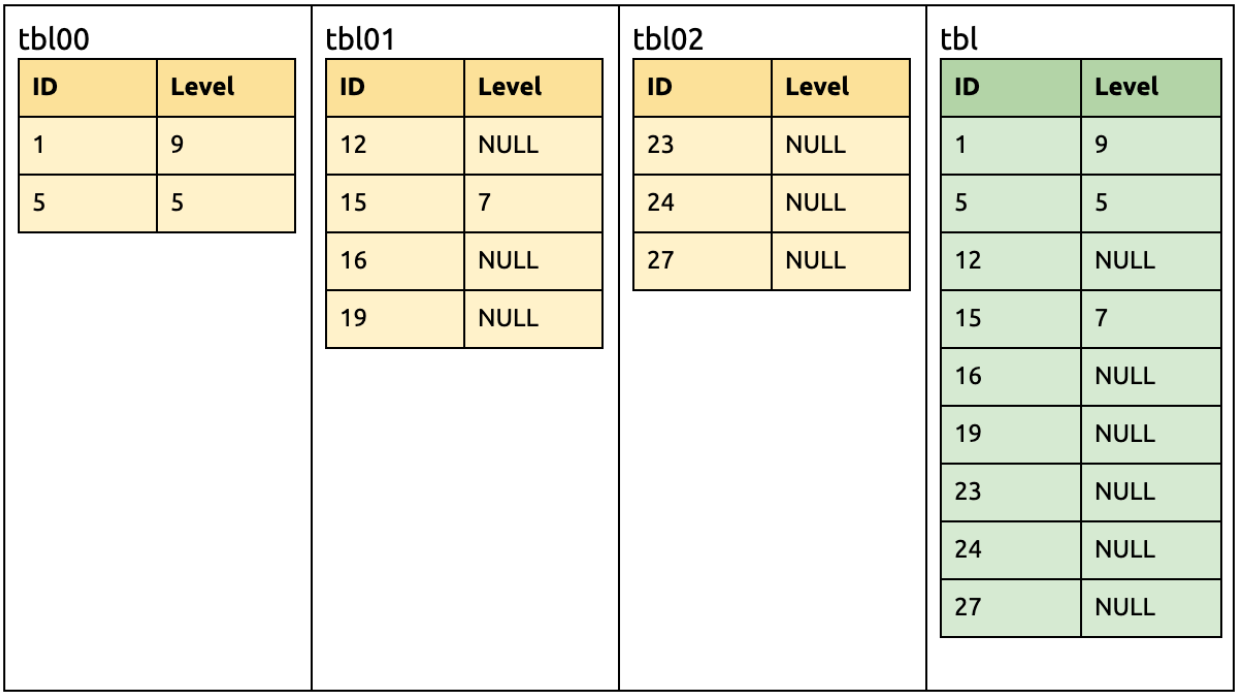


Figure 17: optimistic-ddl-example-9

到此步 `Name` 列也从所有分表消失了，所以可以安全从下游移除：

```
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

1.4.2 迁移使用 GH-ost/PT-osc 的源数据库

本文档介绍在使用 DM 进行从 MySQL 到 TiDB 的数据迁移时，如何配置 `online-ddl`，以及 DM 与 online DDL 工具的协作细节。

1.4.2.1 概述

DDL 是数据库应用中必然会使用的一类 SQL。MySQL 虽然在 5.6 的版本以后支持了 online-ddl 功能，但是也有或多或少的限制。比如某些时候执行 DDL 会获取 MDL 锁，造成锁表，生产环境中，锁表会一定程度阻塞数据库的读取或写入。另外，某些 DDL 需要复制整个表，从而影响数据库整体性能。

因此，用户往往会选择 online DDL 工具执行 DDL，把对读写的影响降到最低。常见的 Online DDL 工具有 gh-ost 和 pt-osc。

这些工具的工作原理可以概括为

1. 根据 DDL 目标表 (real table) 的表结构新建一张镜像表 (ghost table)；
2. 在镜像表上应用 DDL；
3. 将 DDL 目标表的数据同步到镜像表；
4. 在目标表与镜像表数据一致后，通过 `RENAME` 语句使镜像表替换掉目标表。

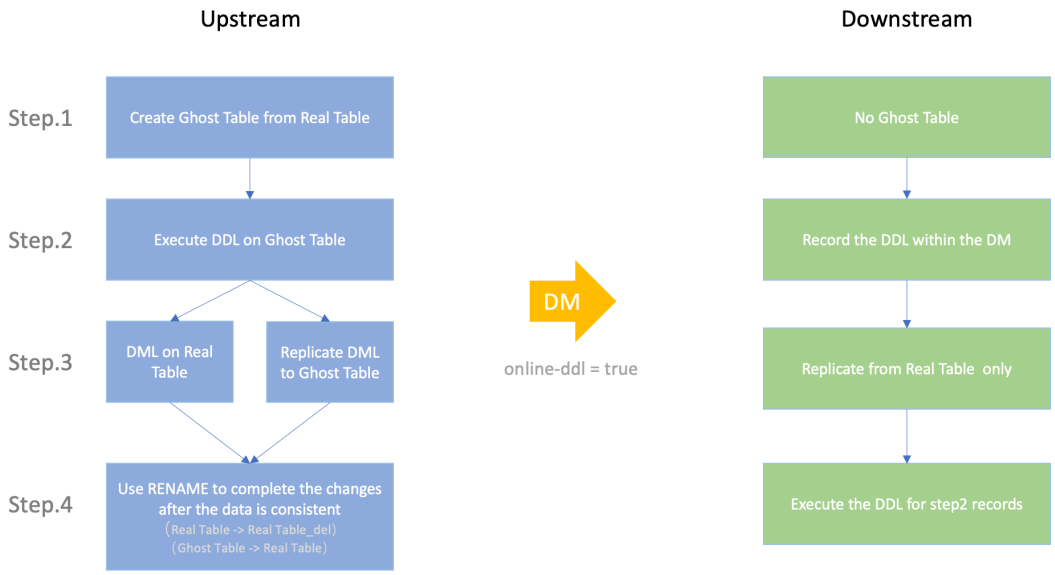


Figure 19: DM online-ddl

在使用 DM 完成 MySQL 到 TiDB 的数据迁移时，online-ddl 功能可以识别上述步骤 2 产生的 DDL，并在步骤 4 时向下游应用 DDL，从而降低镜像表的同步开销。

> **注意：**
>
> 如果希望从源码方面了解 DM online-ddl，可以参考 DM 源码阅读系列文章（八）Online Schema Change 迁移支持，以及 TiDB Online Schema Change 原理。

1.4.2.2 `online-ddl` 配置

一般情况下建议开启 DM 的 `online-ddl` 配置，将产生以下效果：

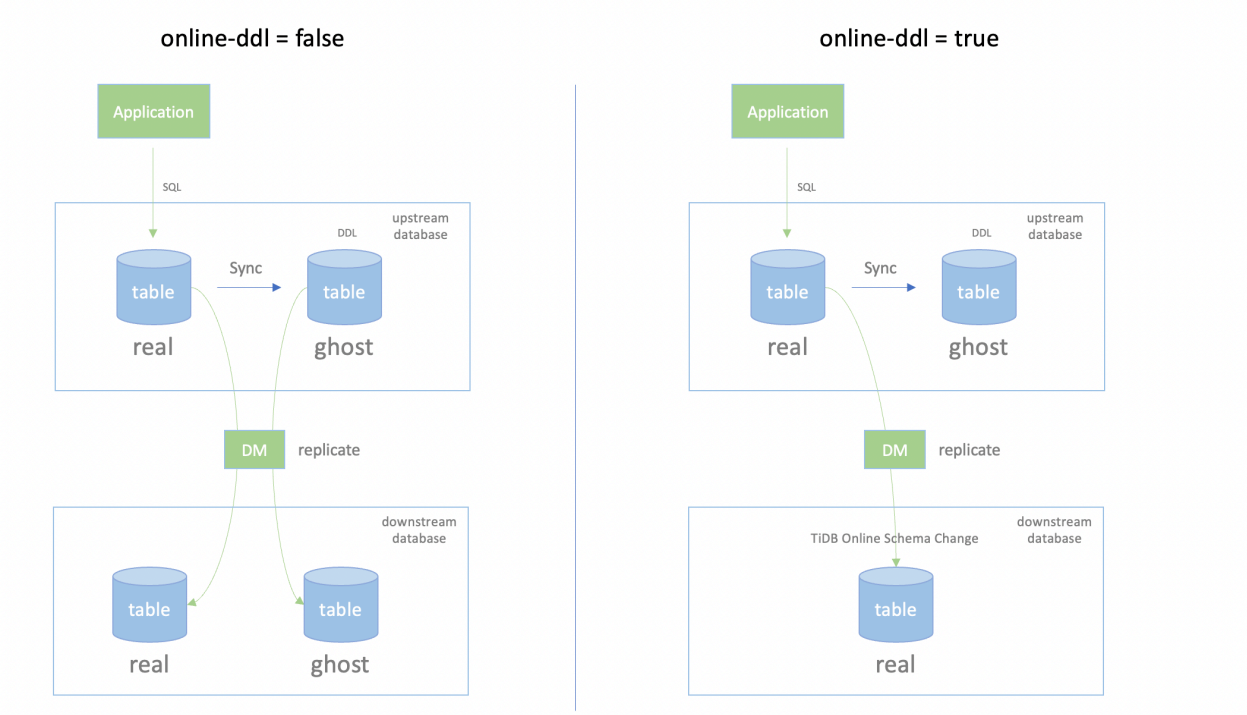


Figure 20: DM online-ddl

- 下游 TiDB 无需创建和同步镜像表，节约相应存储空间和网络传输等开销；
- 在分库分表合并场景下，忽略各分表镜像表的 RENAME 操作，保证同步正确性；
- 受目前 DM 实现限制，在向下游应用 DDL 时，该同步任务的其他 DML 会被阻塞直到 DDL 完成。我们会在后续优化该限制。

> 注意：
>
> 如果需要关闭 `online-ddl` 配置，需注意以下影响：
>
> - 下游 TiDB 将原样同步 gh-ost/pt-osc 等 online DDL 工具的行为；
> - 你需要手动将 online DDL 工具产生的各种临时表、镜像表等添加到任务配置白名单中；
> - 此场景下，无法与分库分表合并场景兼容使用。

1.4.2.3 配置

online-ddl 在 task 配置文件里面与 name 同级，例子详见下面配置 Example。完整的配置及意义，可以参考DM 完整配置文件示例：

```
### ----------- 全局配置 -----------
#### ********* 基本信息配置 *********
name: test                    # 任务名称，需要全局唯一
task-mode: all                # 任务模式，可设为 "full"、"incremental"、"all"
shard-mode: "pessimistic"     # 默认值为 "" 即无需协调。如果为分库分表合并任务
    ↪ ，请设置为悲观协调模式 "pessimistic"。
    ↪ 在深入了解乐观协调模式的原理和使用限制后，也可以设置为乐观协调模式 "
    ↪ optimistic"
meta-schema: "dm_meta"        # 下游储存 `meta` 信息的数据库
online-ddl: true              # 支持上游使用 gh-ost 、pt 两种工具的自动处理
online-ddl-scheme: "gh-ost" # `online-ddl-scheme` 在未来将被弃用，建议使用 `
    ↪ online-ddl`

target-database:              # 下游数据库实例配置
  host: "192.168.0.1"
  port: 4000
  user: "root"
  password: ""                # 如果密码不为空，则推荐使用经过 dmctl 加密的密文
```

在分库分表合并场景，迁移过程中需要协调各个分表的 DDL 语句，以及该 DDL 语句前后的 DML 语句。DM 支持悲观协调模式（pessimistic）和乐观协调模式（optimistic），关于二者的区别和适用场景可参考分库分表合并迁移。

1.4.2.4 DM 与 online DDL 工具协作细节

本小节介绍 DM 与 online DDL 工具 gh-ost 和 pt-osc 在实现 online-schema-change 过程中的协作细节。

1.4.2.4.1 online-schema-change: gh-ost

gh-ost 在实现 online-schema-change 的过程会产生 3 种 table：

- gho：用于应用 DDL，待 gho 表中数据迁移到与 origin table 一致后，通过 rename 的方式替换 origin table。
- ghc：用于存放 online-schema-change 相关的信息。
- del：对 origin table 执行 rename 操作而生成。

DM 在迁移过程中会把上述 table 分成 3 类：

- ghostTable：`\_\*\_gho`
- trashTable：`\_\*\_ghc`、`\_\*\_del`
- realTable：执行 online-ddl 的 origin table

gh-ost 涉及的主要 SQL 以及 DM 的处理：

1. 创建 _ghc 表：

```
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：不执行 _test4_ghc 的创建操作。

2. 创建 _gho 表：

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

DM：不执行 _test4_gho 的创建操作，根据 ghost_schema、ghost_table 以及 dm_worker 的 server_id，删除下游 dm_meta.{task_name}_onlineddl 的记录，清理内存中的相关信息。

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

3. 在 _gho 表应用需要执行的 DDL：

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

DM：不执行 _test4_gho 的 DDL 操作，而是把该 DDL 记录到 dm_meta.{task_name
↪ }_onlineddl 以及内存中。

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

4. 往 _ghc 表写入数据，以及往 _gho 表同步 origin table 的数据：

```
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：只要不是 **realtable** 的 DML 全部不执行。

5. 数据同步完成后 origin table 与 _gho 一起改名，完成 online DDL 操作：

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

DM 执行以下两个操作:

- 把 rename 语句拆分成两个 SQL：

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

- 不执行 `rename to _test4_del`。当要执行 `rename ghost_table to origin table` 的时候，并不执行 rename 语句，而是把步骤 3 记录在内存中的 DDL 读取出来，然后把 ghost_table、ghost_schema 替换为 origin_table 以及对应的 schema，再执行替换后的 DDL。

```
alter table test._test4_gho add column cl1 varchar(20) not null;
--替换为
alter table test.test4 add column cl1 varchar(20) not null;
```

注意：

具体 gh-ost 的 SQL 会根据工具执行时所带的参数而变化。本文只列出主要的 SQL，具体可以参考 gh-ost 官方文档。

1.4.2.4.2 online-schema-change: pt

pt-osc 在实现 online-schema-change 的过程会产生 2 种 table：

- `new`：用于应用 DDL，待表中数据同步到与 origin table 一致后，再通过 rename 的方式替换 origin table。
- `old`：对 origin table 执行 rename 操作后生成。
- 3 种 **trigger**：`pt_osc\_\*\_ins`、`pt_osc\_\*\_upd`、`pt_osc\_\*\_del`，用于在 pt_osc 过程中，同步 origin table 新产生的数据到 `new`。

DM 在迁移过程中会把上述 table 分成 3 类：

- ghostTable : _*_new
- trashTable : _*_old
- realTable : 执行的 online-ddl 的 origin table

pt-osc 主要涉及的 SQL 以及 DM 的处理：

1. 创建 _new 表：

```
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: 不执行 _test4_new 的创建操作。根据 ghost_schema、ghost_table 以及 dm_worker 的 server_id，删除下游 dm_meta.{task_name}_onlineddl 的记录，清理内存中的相关信息。

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

2. 在 _new 表上执行 DDL：

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

DM: 不执行 _test4_new 的 DDL 操作，而是把该 DDL 记录到 dm_meta.{task_name
↪ }_onlineddl 以及内存中。

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

3. 创建用于同步数据的 3 个 Trigger：

```
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: 不执行 TiDB 不支持的相关 Trigger 操作。

4. 往 _new 表同步 origin table 的数据：

```
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: 只要不是 **realTable** 的 DML 全部不执行。

5. 数据同步完成后 origin table 与 _new 一起改名，完成 online DDL 操作：

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

DM 执行以下两个操作:

- 把 rename 语句拆分成两个 SQL。

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

- 不执行 rename to _test4_old。当要执行 rename ghost_table to origin table 的时候，并不执行 rename，而是把步骤 2 记录在内存中的 DDL 读取出来，然后把 ghost_table、ghost_schema 替换为 origin_table 以及对应的 schema，再执行替换后的 DDL。

  ```
  sql  ALTER TABLE `test`.`_test4_new` add column c3 int; --替换为
  ↪ ALTER TABLE `test`.`test4` add column c3 int;
  ```

6. 删除 _old 表以及 online DDL 的 3 个 Trigger：

```
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: 不执行 _test4_old 以及 Trigger 的删除操作。

注意：

具体 pt-osc 的 SQL 会根据工具执行时所带的参数而变化。本文只列出主要的 SQL ，具体可以参考 pt-osc 官方文档。

1.4.3 使用 SQL 表达式过滤某些行变更

1.4.3.1 概述

在数据迁移的过程中，DM 提供了Binlog Event Filter 功能过滤某些类型的 binlog event，例如不向下游迁移 DELETE 事件以达到归档、审计等目的。但是 Binlog Event Filter 无法以更细粒度判断某一行的 DELETE 事件是否要被过滤。

为了解决上述问题，DM 支持使用 SQL 表达式过滤某些行变更。DM 支持的 ROW 格式的 binlog 中，binlog event 带有所有列的值。用户可以基于这些值配置 SQL 表达式。如果该表达式对于某条行变更的计算结果是 TRUE，DM 就不会向下游迁移该条行变更。

> **注意：**
>
> 该功能只会在增量复制阶段生效，并不会在全量迁移阶段生效。

1.4.3.2 配置示例

与Binlog Event Filter 类似，表达式过滤需要在数据迁移任务配置文件里配置，详见下面配置样例。完整的配置及意义，可以参考DM 完整配置文件示例：

```
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"
```

上面的示例配置了 even_c 规则，并让 source ID 为 mysql-replica-01 的数据源引用了该规则。even_c 规则的含义是：

对于 expr_filter 库下的 tbl 表，当插入的 c 的值为偶数 (c % 2 = 0) 时，不将这条插入语句迁移到下游。

下面展示该规则的使用效果。

在上游数据源增量插入以下数据：

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

随后在下游查询 tbl 表，可见只有 c 的值为单数的行迁移到了下游：

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

1.4.3.3 配置参数及规则说明

- schema：要匹配的上游数据库库名，不支持通配符匹配或正则匹配。
- table: 要匹配的上游表名，不支持通配符匹配或正则匹配
- insert-value-expr：配置一个表达式，对 INSERT 类型的 binlog event (WRITE_ROWS_EVENT) 带有的值生效。不能与 update-old-value-expr、update-new-value-expr、delete-value-expr 出现在一个配置项中。
- update-old-value-expr：配置一个表达式，对 UPDATE 类型的 binlog event (UPDATE_ROWS_EVENT) 更新对应的旧值生效。不能与 insert-value-expr、delete-value-expr 出现在一个配置项中。
- update-new-value-expr：配置一个表达式，对 UPDATE 类型的 binlog event (UPDATE_ROWS_EVENT) 更新对应的新值生效。不能与 insert-value-expr、delete-value-expr 出现在一个配置项中。
- delete-value-expr：配置一个表达式，对 DELETE 类型的 binlog event (DELETE_ROWS_EVENT) 带有的值生效。不能与 insert-value-expr、update-old-value-expr、update-new-value-expr 出现在一个配置项中。

> 注意：
>
> update-old-value-expr 可以与 update-new-value-expr 同时配置。
>
> - 当二者同时配置时，会将更新旧值满足 update-old-value-expr 且更新新值满足 update-new-value-expr 的行变动过滤掉。
> - 当只配置一者时，配置的这条表达式会决定是否过滤整个行变更，即旧值的删除和新值的插入会作为一个整体被过滤掉。

SQL 表达式可以涉及一列或多列，也可使用 TiDB 支持的 SQL 函数，例如 `c % 2 = 0`、`a*a + b*b = c*c`、`ts > NOW()`。

TIMESTAMP 类型的默认时区是 UTC。可以使用 `c_timestamp = '2021-01-01 12:34:56.5678+08:00'` 的方式显式指定时区。

配置项 `expression-filter` 下可以定义多条过滤规则，上游数据源在其 `expression-filters` 配置项中引用需要的规则使其生效。当有多条规则生效时，匹配到任意一条规则即会导致某个行变更被过滤。

> **注意：**
>
> 为某张表设置过多的表达式过滤会增加 DM 的计算开销，可能导致数据迁移速度变慢。

1.5 Data Migration 架构

DM 主要包括三个组件：DM-master，DM-worker 和 dmctl。

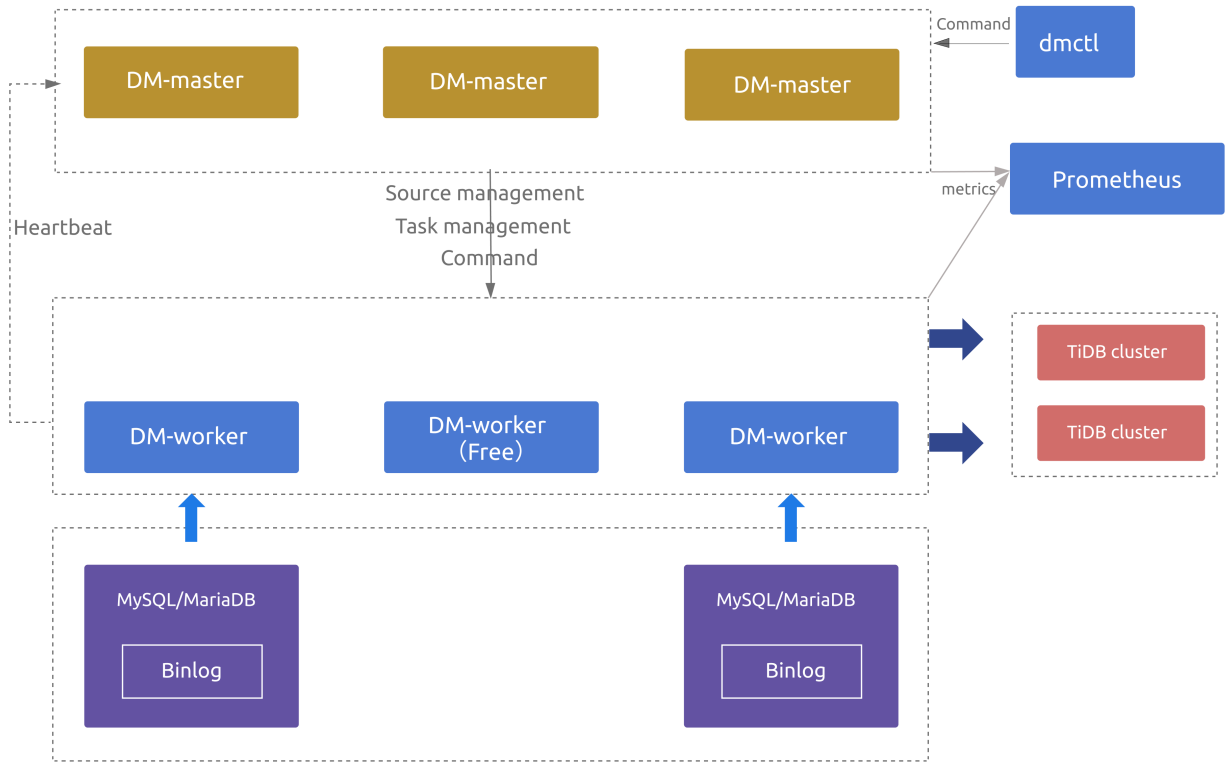


Figure 21: Data Migration architecture

1.5.1 架构组件

1.5.1.1 DM-master

DM-master 负责管理和调度数据迁移任务的各项操作。

- 保存 DM 集群的拓扑信息
- 监控 DM-worker 进程的运行状态
- 监控数据迁移任务的运行状态
- 提供数据迁移任务管理的统一入口
- 协调分库分表场景下各个实例分表的 DDL 迁移

1.5.1.2 DM-worker

DM-worker 负责执行具体的数据迁移任务。

- 将 binlog 数据持久化保存在本地
- 保存数据迁移子任务的配置信息
- 编排数据迁移子任务的运行
- 监控数据迁移子任务的运行状态

有关于 DM-worker 的更多介绍，详见DM-worker 简介。

1.5.1.3 dmctl

dmctl 是用来控制 DM 集群的命令行工具。

- 创建、更新或删除数据迁移任务
- 查看数据迁移任务状态
- 处理数据迁移任务错误
- 校验数据迁移任务配置的正确性

有关于 dmctl 的使用介绍，详见dmctl 使用。

1.5.2 架构特性

1.5.2.1 高可用

当部署多个 DM-master 节点时，所有 DM-master 节点将使用内部嵌入的 etcd 组成集群。该 DM-master 集群用于存储集群节点信息、任务配置等元数据，同时通过 etcd 选举出 leader 节点。该 leader 节点用于提供集群管理、数据迁移任务管理相关的各类服务。因此，若可用的 DM-master 节点数超过部署节点的半数，即可正常提供服务。

当部署的 DM-worker 节点数超过上游 MySQL/MariaDB 节点数时，超出上游节点数的相关 DM-worker 节点默认将处于空闲状态。若某个 DM-worker 节点下线或与 DM-master leader 发生网络隔离，DM-master 能自动将与原 DM-worker 节点相关的数据迁移任务调

度到其他空闲的 DM-worker 节点上（若原 DM-worker 节点为网络隔离状态，则其会自动停止相关的数据迁移任务）；若无空闲的 DM-worker 节点可供调度，则原 DM-worker 相关的数据迁移任务将暂时挂起，直到有空闲 DM-worker 节点后自动恢复。

> 注意：
>
> 当数据迁移任务处于全量导出或导入阶段时，该迁移任务暂不支持高可用，主要原因为：
>
> - 对于全量导出，MySQL 暂不支持指定从特定快照点导出，也就是说数据迁移任务被重新调度或重启后，无法继续从前一次中断时刻继续导出。
> - 对于全量导入，DM-worker 暂不支持跨节点读取全量导出数据，也就是说数据迁移任务被调度到的新 DM-worker 节点无法读取调度发生前原 DM-worker 节点上的全量导出数据。

1.6 DM 5.3.0 性能测试报告

本报告记录了对 5.3.0 版本的 DM 进行性能测试的目的、环境、场景和结果。

1.6.1 测试目的

该性能测试用于评估使用 DM 进行全量数据导入和增量数据复制的性能上限，并根据测试结果提供 DM 迁移任务的参考配置。

1.6.2 测试环境

1.6.2.1 测试机器信息

系统信息：

机器 IP	操作系统	内核版本	文件系统类型
172.16.6.1	CentOS Linux release 7.8.2003	3.10.0-957.el7.x86_64	ext4
172.16.6.2	CentOS Linux release 7.8.2003	3.10.0-957.el7.x86_64	ext4
172.16.6.3	CentOS Linux release 7.8.2003	3.10.0-957.el7.x86_64	ext4

硬件信息：

类别	指标
CPU	Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz, 48 Cores

类别	指标
内存	192G, 12 * 16GB DIMM DDR4 2133 MHz
磁盘	INTEL SSDPE2KX040T8 4TB
网卡	万兆网卡

其他：

- 服务器间网络延迟：rtt min/avg/max/mdev = 0.045/0.064/0.144/0.024 ms

1.6.2.2 集群拓扑

机器 IP	部署的服务
172.16.6.1	PD1, TiDB1, TiKV1, MySQL1, DM-master1
172.16.6.2	PD2, TiDB2, TiKV2, DM-worker1
172.16.6.3	PD3, TiDB3, TiKV3

1.6.2.3 各服务版本信息

- MySQL 版本：5.7.36-log
- TiDB 版本：v5.2.1
- DM 版本：v5.3.0
- Sysbench 版本：1.1.0

1.6.3 测试场景

可以参考性能测试中介绍的测试场景，测试单个 MySQL 实例到 TiDB 的数据迁移：MySQL1 (172.16.6.1) -> DM-worker(172.16.6.2) -> TiDB(load balance) (172.16.6.4)。

1.6.3.1 全量导入性能测试

可以参考全量导入性能测试用例中介绍的方法进行测试。

1.6.3.1.1 全量导入性能测试结果

在 `mydumper` 配置项中配置 `threads` 参数，可以通过 Dumpling 开启多线程并发导出，提高数据导出性能。

测试项	数据量 (G)	threads	rows	statement-size	导出时间 (s)	导出速度 (MB/s)
dump data	38.1	32	320000	1000000	45	846

测试项	数据量 (G)	pool size	每条插入语句包含的行数	事务执行最大延迟 (s)	导入时间 (s)	导
load data	38.1	32	4878	76	2740	

1.6.3.1.2 在 load 处理单元使用不同 pool size 的性能测试对比

该测试中使用 sysbench 全量导入的数据量为 3.78 GB，测试数据如下所示：

load 处理单元 pool size	事务执行最大延迟 (s)	导入时间 (s)	导入速度 (MB/s)	TiDB 99 duration (s
2	0.71	397	9.5	0.61
4	1.21	363	10.4	1.03
8	3.30	279	13.5	2.11
16	5.56	200	18.9	3.04
32	6.92	218	17.3	6.56
64	8.59	231	16.3	8.62

1.6.3.1.3 导入数据时每条插入语句包含行数不同的情况下的性能测试对比

该测试中全量导入的数据量为 3.78 GB，load 处理单元 pool-size 大小为 32。插入语句包含行数通过 mydumper 配置项中的 statement-size，rows 或者 extra-args 参数来控制。

每条语句中包含的行数	mydumper extra-args 参数	事务执行最大延迟 (s)	导入时间 (s)	导入速度 (MB/s)	TiDB 99 duration (s)
7506	-s 1500000 -r 320000	8.34	229	16.5	10.64
5006	-s 1000000 -r 320000	6.12	218	17.3	7.23
2506	-s 500000 -r 320000	4.27	232	16.2	3.24
1256	-s 250000 -r 320000	2.25	235	16.0	1.92
629	-s 125000 -r 320000	1.03	246	15.3	0.91
315	-s 62500 -r 320000	0.63	249	15.1	0.44

1.6.3.2 增量复制性能测试用例

使用增量复制性能测试用例中介绍的方法进行测试。

1.6.3.2.1 增量复制性能测试结果

该性能测试中复制任务 `sync` 处理单元 `worker-count` 设置为 32，`batch` 大小设置为 100。

组件	qps	tps	
MySQL	40.65k	40.65k	
DM binlog replication unit	29.1k (单位时间内接收到的不被忽略的 binlog event 数量)	-	92ms
TiDB	32.0k (Begin/Commit 1.5k Insert 29.1k)	3.72k	95%: 6

1.6.3.2.2 在 sync 处理单元使用不同并发度的性能测试对比

sync 处理单元 worker-count 数	DM qps	DM 事务执行最大延迟 (ms)	TiDB qps	TiDB 99 duration
4	10.2	40	10.5k	4
8	17.6k	64	18.9k	5
16	29.5k	80	30.5k	7
32	29.1k	92	32.0k	9
64	27.4k	88	37.7k	14
1024	22.9k	85	57.5k	25

1.6.3.2.3 不同数据分布的增量复制性能测试对比

sysbench 语句类型	DM qps	DM 事务执行最大延迟 (ms)	TiDB qps	TiDB 99 duration (ms)
insert_only	29.1k	64	32.0k	8
write_only	23.5k	296	24.2k	18

1.6.4 推荐迁移任务参数配置

1.6.4.1 dump 处理单元

推荐每一条插入语句的大小在 200KB ~ 1MB 之间，相应每条语句包含的行数大约在 1000-5000（具体包含的语句行数与实际场景中每行数据大小有关）。

1.6.4.2 load 处理单元

推荐 `pool-size` 设置为 16 ~ 32。

1.6.4.3 sync 处理单元

推荐将 `batch` 设置为 100，`worker-count` 设置为 16 ~ 32。

2 快速上手

2.1 TiDB Data Migration 快速上手指南

本文介绍如何快速体验使用数据迁移工具 TiDB Data Migration (DM) 从 MySQL 迁移数据到 TiDB。

如需在生产环境中部署 DM，请参考以下文档：

- 使用 TiUP 部署 DM 集群
- 创建数据源
- 创建数据迁移任务

2.1.1 使用样例

在本地部署的 DM 集群组件和访问的 MySQL 和 TiDB 节点的信息如下：

实例	服务器地址	端口使用
DM-master	127.0.0.1	8261, 8291（内部端口）
DM-worker	127.0.0.1	8262
MySQL-3306	127.0.0.1	3306
TiDB	127.0.0.1	4000

2.1.2 使用 binary 包部署 DM

2.1.2.1 准备 DM binary 包

首先需要下载 DM 最新的 binary 或者手动编译。

2.1.2.1.1 第一种方式：下载最新 DM binary 包

```
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 第二种方式：编译最新 DM binary 包

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

2.1.2.2 部署 DM-master

执行如下命令启动 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 部署 DM-worker

执行如下命令启动 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 检查 DM 集群部署是否正常

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

一个正常 DM 集群的范例返回结果如下所示：

```
{
    "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 从 MySQL 同步数据到 TiDB

2.1.3.1 准备数据

使用 DM 之前，先准备好数据，向 MySQL-3306 写入示例数据。

```
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 加载数据源 MySQL 配置

运行数据迁移任务前，需要加载数据源的配置，也就是 MySQL 的相关设置，到 DM。

2.1.3.2.1 对数据源 MySQL 访问密码进行加密

注意：

- 如果数据源没有设置密码，则可以跳过该步骤。
- 自 v2.0 起，DM 可以使用明文密码配置数据源的访问密码信息。

为了安全，可配置及使用加密后的 MySQL 访问密码，以密码为 “123456” 为例：

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

```
fCxfQ9XKCezSzuCD0Wf5dUD+LsKegSg=
```

记录该加密后的密码，用于下面新建 MySQL 数据源。

2.1.3.2.2 编写数据源 MySQL 配置

把以下配置文件内容写入到 `mysql-source-conf.yaml` 中。

MySQL1 的配置文件：

```
## MySQL Configuration.

source-id: "mysql-replica-01"

from:
  host: "127.0.0.1"
  user: "root"
  password: "fCxfQ9XKCezSzuCD0Wf5dUD+LsKegSg="
  port: 3306
```

2.1.3.2.3 加载数据源 MySQL 配置

在终端中执行下面的命令，使用 dmctl 将 MySQL 的数据源配置加载到 DM 集群中：

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

结果如下：

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
```

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

这样就成功将 MySQL-3306 数据源添加到了 DM 集群。

2.1.3.3 创建数据迁移任务

在导入准备数据后，进行以下操作将 MySQL 的 testdm.t1 和 testdm.t2 两张表迁移到 TiDB。

1. 创建任务的配置文件 testdm-task.yaml：

```
---
name: testdm
task-mode: all

target-database:
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: "" # 如果密码不为空，则推荐使用经过 dmctl 加密的密文

mysql-instances:
  - source-id: "mysql-replica-01"
    block-allow-list: "ba-rule1"

block-allow-list:
  ba-rule1:
    do-dbs: ["testdm"]
```

2. 使用 dmctl 创建任务：

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

结果如下：

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
```

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

这样就成功创建了一个将 MySQL-3306 数据迁移到 TiDB 的任务。

2.1.3.4 查看迁移任务状态

在创建迁移任务之后，可以用 `dmtcl query-status` 来查看任务的状态。

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

结果如下：

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

2.2 使用 TiUP 部署 DM 集群

TiUP 是 TiDB 4.0 版本引入的集群运维工具，TiUP DM 是 TiUP 提供的使用 Golang 编写的集群管理组件，通过 TiUP DM 组件就可以进行日常的运维工作，包括部署、启动、关闭、销毁、扩缩容、升级 DM 集群以及管理 DM 集群参数。

目前 TiUP 可以支持部署 v2.0 及以上版本的 DM。本文将介绍不同集群拓扑的具体部署步骤。

> **注意：**
>
> 如果部署机器的操作系统支持 SELinux，请确保 SELinux 处于关闭状态。

2.2.1 前提条件

当 DM 执行全量数据复制任务时，每个 DM-worker 只绑定一个上游数据库。DM-worker 首先在上游导出全部数据，然后将数据导入下游数据库。因此，DM-worker 的主机需要有足够的存储空间，具体存储路径在后续创建迁移任务时指定。

另外，部署 DM 集群需参照DM 集群软硬件环境需求，满足相应要求。

2.2.2 第 1 步：在中控机上安装 TiUP 组件

使用普通用户登录中控机，以 tidb 用户为例，后续安装 TiUP 及集群管理操作均通过该用户完成：

1. 执行如下命令安装 TiUP 工具：

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

安装完成后，~/.bashrc 已将 TiUP 加入到路径中，你需要新开一个终端或重新声明全局变量 source ~/.bashrc 来使用 TiUP。

2. 安装 TiUP DM 组件：

```
tiup install dm dmctl
```

2.2.3 第 2 步：编辑初始化配置文件

请根据不同的集群拓扑，编辑 TiUP 所需的集群初始化配置文件。

请根据配置文件模板，新建一个配置文件 topology.yaml。如果有其他组合场景的需求，请根据多个模板自行调整。

可以使用 tiup dm template > topology.yaml 命令快速生成配置文件模板。

部署 3 个 DM-master、3 个 DM-worker 与 1 个监控组件的配置如下：

```
#全局变量适用于配置中的其他组件。如果组件实例中缺少一个特定值，
    ↪ 则相应的全局变量将用作默认值。
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"
    # 下列配置项用于覆盖 `server_configs.master` 的值。
    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
## 如果不需要确保 DM 集群高可用，则可只部署 1 个 DM-master 节点，且部署的 DM-
    ↪ worker 节点数量不少于上游待迁移的 MySQL/MariaDB 实例数。
## 如果需要确保 DM 集群高可用，则推荐部署 3 个 DM-master 节点，且部署的 DM-
    ↪ worker 节点数量大于上游待迁移的 MySQL/MariaDB 实例数（如 DM-worker
    ↪ 节点数量比上游实例数多 2 个）。
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"
    # 下列配置项用于覆盖 `server_configs.worker` 的值。
    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"
```

注意：

- 不建议在一台主机上运行太多 DM-worker。每个 DM-worker 至少应有 2 核 CPU 和 4 GiB 内存。
- 需要确保以下组件间端口可正常连通：
  - 各 DM-master 节点间的 `peer_port`（默认为 8291）可互相连通。
  - 各 DM-master 节点可连通所有 DM-worker 节点的 `port`（默认为 8262）。
  - 各 DM-worker 节点可连通所有 DM-master 节点的 `port`（默认为 8261）。
  - TiUP 节点可连通所有 DM-master 节点的 `port`（默认为 8261）。
  - TiUP 节点可连通所有 DM-worker 节点的 `port`（默认为 8262）。

更多 master_servers.host.config 参数说明，请参考 master parameter；更多 worker_servers.host.config 参数说明，请参考 worker parameter。

2.2.4 第 3 步：执行部署命令

> 注意：
>
> 通过 TiUP 进行集群部署可以使用密钥或者交互密码方式来进行安全认证：
>
> - 如果是密钥方式，可以通过 -i 或者 --identity_file 来指定密钥的路径；
> - 如果是密码方式，可以通过 -p 进入密码交互窗口；
> - 如果已经配置免密登录目标机，则不需填写认证。

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

以上部署命令中：

- 通过 TiUP DM 部署的集群名称为 dm-test。
- 部署版本为 ${version}，可以通过执行 tiup list dm-master 来查看 TiUP 支持的最新版本。
- 初始化配置文件为 topology.yaml。
- --user root：通过 root 用户登录到目标主机完成集群部署，该用户需要有 ssh 到目标机器的权限，并且在目标机器有 sudo 权限。也可以用其他有 ssh 和 sudo 权限的用户完成部署。
- -i 及 -p：非必选项，如果已经配置免密登录目标机，则不需填写，否则选择其一即可。-i 为可登录到目标机的 root 用户（或 --user 指定的其他用户）的私钥，也可使用 -p 交互式输入该用户的密码。
- TiUP DM 使用内置的 SSH 客户端，如需使用系统自带的 SSH 客户端，请参考 TiUP DM 文档中使用中控机系统自带的 SSH 客户端连接集群章节进行设置。

预期日志结尾输出会有 Deployed cluster `dm-test` successfully 关键词，表示部署成功。

2.2.5 第 4 步：查看 TiUP 管理的集群情况

```
tiup dm list
```

TiUP 支持管理多个 DM 集群，该命令会输出当前通过 TiUP DM 管理的所有集群信息，包括集群名称、部署用户、版本、密钥信息等：

```
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 第 5 步：检查部署的 DM 集群情况

例如，执行如下命令检查 dm-test 集群情况：

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

预期输出包括 dm-test 集群中实例 ID、角色、主机、监听端口和状态（由于还未启动，所以状态为 Down/inactive）、目录信息。

2.2.7 第 6 步：启动集群

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

预期结果输出 Started cluster `dm-test` successfully 表示启动成功。

2.2.8 第 7 步：验证集群运行状态

通过以下 TiUP 命令检查集群状态：

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

在输出结果中，如果 Status 状态信息为 Up，说明集群状态正常。

2.2.9 第 8 步：使用 dmctl 管理迁移任务

dmctl 是用来控制集群运行命令的工具，推荐通过 TiUP 获取该工具。

dmctl 支持命令模式与交互模式，具体请见使用 dmctl 运维集群。

2.3 创建数据源

> 注意：
> 在创建数据源之前，你需要先使用 TiUP 部署 DM 集群。

本文档介绍如何为 TiDB Data Migration (DM) 的数据迁移任务创建数据源。

数据源包含了访问迁移任务上游所需的信息。数据迁移任务需要引用对应的数据源来获取访问配置信息。因此，在创建数据迁移任务之前，需要先创建任务的数据源。详细的数据源管理命令请参考管理上游数据源。

2.3.1 第一步：配置数据源

1. （可选）加密数据源密码

   在 DM 的配置文件中，推荐使用经 dmctl 加密后的密文密码。按照下面的示例可以获得数据源的密文密码，用于下一步编写数据源配置文件。

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

```
MKxn0Qo3m3XOyjCnhEMtsUCm83EhGQDZ/T4=
```

2. 编写数据源配置文件

   每个数据源需要一个单独的配置文件来创建数据源。按照下面示例创建 ID 为 “mysql-01” 的数据源，创建数据源配置文件 ./source-mysql-01.yaml：

```
source-id: "mysql-01" # 数据源 ID，在数据迁移任务配置和 dmctl
    ↪ 命令行中引用该 source-id 可以关联到对应的数据源

from:
  host: "127.0.0.1"
  port: 3306
  user: "root"
  password: "MKxn0Qo3m3XOyjCnhEMtsUCm83EhGQDZ/T4=" # 推荐使用 dmctl
      ↪ 对上游数据源的用户密码加密之后的密码
  security:                                    # 上游数据源 TLS 相关配置。
      ↪ 如果没有需要则可以删除
    ssl-ca: "/path/to/ca.pem"
    ssl-cert: "/path/to/cert.pem"
    ssl-key: "/path/to/key.pem"
```

2.3.2 第二步：创建数据源

使用如下命令创建数据源：

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

数据源配置文件的其他配置参考数据源配置文件介绍。

命令返回结果如下：

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

2.3.3 第三步：查询创建的数据源

创建数据源后，可以使用如下命令查看创建的数据源：

- 如果知道数据源的 source-id，可以通过 dmctl get-config source <source-id> 命令直接查看数据源配置：

```
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: '******'
}
```

- 如果不知道数据源的 source-id，可以先通过 dmctl operate-source show 命令查看源数据库列表，从中可以找到对应的数据源。

```
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 数据迁移场景

2.4.1 数据迁移场景概述

注意：

在创建数据迁移任务之前，需要先完成以下操作：

1. 使用 TiUP 部署 DM 集群。
2. 创建数据源。

本文介绍多个业务需求场景下如何配置数据迁移任务。你可以根据具体的场景介绍，选择参考适合的教程来创建对应的数据迁移任务。

除了业务需求场景导向的创建数据迁移任务教程之外：

- 完整的数据迁移任务配置示例，请参考 DM 任务完整配置文件介绍
- 数据迁移任务的配置向导，请参考数据迁移任务配置向导

2.4.1.1 多数据源汇总迁移到 TiDB

如果你需要将多个数据源的数据汇总迁移到 TiDB，此外还需要进行表重命名以防止多个数据源中相同表名在迁移过程中出现冲突，或者需要屏蔽掉某些表的某些 DDL/DML 操作，那么你可以参考多数据源汇总迁移到 TiDB。

2.4.1.2 分库分表合并迁移到 TiDB

如果你需要将使用分表方案的业务合并迁移到 TiDB，可以参考分表合并迁移到 TiDB。

2.4.1.3 只迁移数据源增量数据到 TiDB

如果你使用其他工具进行了全量数据迁移，例如使用 TiDB Lightning 迁移了全量数据，然后只使用 DM 进行增量数据迁移，那么该场景的数据迁移任务配置可以参考只迁移数据源增量数据到 TiDB。

2.4.1.4 TiDB 目标表比数据源表的列更多

如果你需要在 TiDB 中定制创建表结构，TiDB 的表结构包含数据源对应表的所有列，且比数据源的表结构有更多的列，那么该场景的数据迁移任务配置需要参考TiDB 表结构存在更多列场景的数据迁移。

2.4.2 多数据源合并迁移到 TiDB

本文介绍了 DM 工具的一个简单使用场景：将三个数据源 MySQL 实例的数据迁移到一个下游 TiDB 集群中。

2.4.2.1 数据源实例

假设数据源结构为：

- 实例 1

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

- 实例 2

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

- 实例 3

Schema	Tables
user	information, log
store	store_gz, store_sz

Schema	Tables
log	messages

2.4.2.2 迁移要求

1. 不合并 `user` 库。
    1. 将实例 1 中的 `user` 库迁移到下游 TiDB 的 `user_north` 库中。
    2. 将实例 2 中的 `user` 库迁移到下游 TiDB 的 `user_east` 库中。
    3. 将实例 3 中的 `user` 库迁移到下游 TiDB 的 `user_south` 库中。
    4. 任何情况下都不删除 `user.log` 表的任何数据。
2. 将数据源 `store` 库迁移到下游 `store` 库中，且迁移过程中不合并表。
    1. 实例 2 和实例 3 中都存在 `store_sz` 表，且这两个 `store_sz` 表分别被迁移到下游的 `store_suzhou` 表和 `store_shenzhen` 表中。
    2. 任何情况下都不删除 `store` 库的任何数据。
3. `log` 库需要被过滤掉。

2.4.2.3 下游实例

假设下游结构为：

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 迁移方案

- 为了满足迁移要求中第一点的前三条要求，需要配置以下table routing 规则：

```
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"
```

- 为了满足迁移要求中第二点的第一条要求，需要配置以下table routing 规则：

```
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"
```

- 为了满足迁移要求中第一点的第四条要求，需要配置以下binlog event filter 规则：

```
filters:
  ...
  log-filter-rule:              # 过滤掉 user.log 表的任何删除操作
    schema-pattern: "user"
    table-pattern: "log"
    events: ["truncate table", "drop table", "delete"]
    action: Ignore
  user-filter-rule:             # 过滤掉删除 user 库操作
    schema-pattern: "user"
    events: ["drop database"]
    action: Ignore
```

- 为了满足迁移要求中第二点的第二条要求，需要配置以下binlog event filter 规则：

```
filters:
  ...
  store-filter-rule:         # 过滤掉删除 store 库，以及 store
     ↪ 库下面任何表的所有删除操作
    schema-pattern: "store"
    events: ["drop database", "truncate table", "drop table", "delete"]
    action: Ignore
```

> **注意：**
>
> store-filter-rule 不同于 log-filter-rule 和 user-filter-rule。store-filter-rule 是针对整个 store 库的规则，而 log-filter-rule 和 user-filter-rule 是针对 user 库中 log 表的规则。

- 为了满足迁移要求中的第三点要求，需要配置以下Block & Allow Lists：

```
block-allow-list: # 通过黑白名单，过滤掉 log 库的所有操作
  log-ignored:
    ignore-dbs: ["log"]
```

2.4.2.5 迁移任务配置

以下是完整的迁移任务配置，更多详情请参阅数据迁移任务配置向导。

```
name: "one-tidb-slave"
task-mode: all                          # 进行全量数据迁移 + 增量数据迁移
meta-schema: "dm_meta"

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

mysql-instances:
  -
    source-id: "instance-1" # 数据源 ID，可以从数据源配置中获取
    route-rules: ["instance-1-user-rule"] # 应用于该数据源的 table route 规则
    filter-rules: ["log-filter-rule", "user-filter-rule" , "store-filter-
       ↪ rule"] # 应用于该数据源的 binlog event filter 规则
    block-allow-list: "log-ignored" # 应用于该数据源的 Block & Allow Lists
       ↪ 规则
  -
    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"
  -
    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"

### 所有实例的共有配置

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
```

2.4.3 分表合并迁移到 TiDB

本文介绍如何在分库分表合并场景中使用 Data Migration (DM) 将上游数据迁移至下游 TiDB 集群。

下面介绍了一个简单的场景，两个数据源 MySQL 实例的分库和分表数据需要迁移至下游 TiDB 集群。更多详情请参阅分表合并数据迁移最佳实践。

2.4.3.1 数据源实例

假设数据源结构如下：

- 实例 1

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

- 实例 2

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

2.4.3.2 迁移需求

1. `user.information` 需要合并到下游 TiDB 中的 `user.information` 表。
2. 实例中的 `store_{01|02}.sale_{01|02}` 表合并至下游 TiDB 中的 `store.sale` 表。
3. 同步 `user`，`store_{01|02}` 库，但不同步两个实例的 `user.log_bak` 表。
4. 过滤掉两个实例中 `store_{01|02}.sale_{01|02}` 表的所有删除操作，并过滤该库的 `drop database` 操作。

预期迁移后下游库结构如下：

Schema	Tables
user	information
store	sale

2.4.3.3 分表数据冲突检查

迁移需求 #1 和 #2 涉及合库合表，来自多张分表的数据可能引发主键或唯一索引的数据冲突。这需要我们检查这几组分表数据的业务特点，详情请见跨分表数据在主键或唯一索引冲突处理。在本示例中：

`user.information` 表结构为

```
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
```

其中 id 列为主键，uid 列为唯一索引。id 列具有自增属性，多个分表范围重复会引发数据冲突。uid 可以保证全局满足唯一索引，因此可以按照参考去掉自增主键的主键属性中介绍的操作绕过 id 列。

store_{01|02}.sale_{01|02} 的表结构为

```
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
```

其中 sid 是分片键，可以保证同一个 sid 只会划分到一个分表中，因此不会引发数据冲突，无需进行额外操作。

2.4.3.4 迁移方案

- 要满足迁移需求 #1，无需配置table routing 规则。按照去掉自增主键的主键属性的要求，在下游手动建表。

  ```
  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
  ```

  并在配置文件中跳过前置检查

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

- 要满足迁移需求 #2，配置table routing 规则如下：

```
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"
```

- 要满足迁移需求 #3，配置Block & Allow Lists 如下：

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

- 要满足迁移需求 #4，配置Binlog event filter 规则如下：

```
filters:
  ...
  sale-filter-rule:  # 过滤掉 store_* 库下面任何表的任何删除操作
    schema-pattern: "store_*"
    table-pattern: "sale_*"
    events: ["truncate table", "drop table", "delete"]
    action: Ignore
  store-filter-rule: # 过滤掉删除 store_* 库的操作
    schema-pattern: "store_*"
    events: ["drop database"]
    action: Ignore
```

2.4.3.5 迁移任务配置

迁移任务的完整配置如下，更多详情请参阅数据迁移任务配置向导。

```
name: "shard_merge"
task-mode: all                                  # 进行全量数据迁移 + 增量数据迁移
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"    # 数据源 ID，可以从数据源配置中获取
    route-rules: ["store-route-rule", "sale-route-rule"] # 应用于该数据源的
       ↪ table route 规则
    filter-rules: ["store-filter-rule", "sale-filter-rule"] #
       ↪ 应用于该数据源的 binlog event filter 规则
    block-allow-list: "log-bak-ignored" # 应用于该数据源的 Block & Allow
       ↪ Lists 规则
  -
    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"

### 所有实例共享的其他通用配置

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 增量迁移数据到 TiDB

本文介绍如何使用 DM 将源数据库从指定位置开始的 Binlog 同步到下游 TiDB。本文以迁移一个数据源 MySQL 实例为例。

2.4.4.1 数据源表

假设数据源实例为：

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

2.4.4.2 迁移要求

只将数据源 `log` 库从某个 binlog 位置起的数据变更同步到 TiDB 集群。

2.4.4.3 增量数据迁移操作

本节按顺序给出迁移操作步骤，指导如何使用 DM 将数据源的 `log` 库从某个时间点起的数据变动同步到 TiDB 集群。

2.4.4.3.1 确定增量同步起始位置

首先需要确定开始迁移的数据源 binlog 位置。如果你确定 binlog 的同步位置，那么可以跳过这一步。

你可以通过下面的方法获得对应数据源开启迁移的 binlog 位置点：

- 使用 Dumpling/Mydumper 进行全量数据导出，然后使用其他工具，如 TiDB Lightning，进行全量数据导入，则可以通过导出数据的 metadata 文件获取同步位置；

```
  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
```

- 使用 `SHOW BINLOG EVENTS` 语句，或者使用 `mysqlbinlog` 工具查看 binlog，选择合适的位置。

- 如果从当前时间点开始同步 binlog，则可以使用 SHOW MASTER STATUS 命令查看当前位置：

```
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)
```

本例将从 binlog position=(mysql-bin.000001, 2022), gtid=09bec856-ba95-11
↪ ea-850a-58f2b4af5188:1-9 这个位置开始同步。

2.4.4.3.2 手动在下游创建表

由于建表 SQL 语句在同步起始位置之前，本次增量同步任务并不会自动在下游创建表。因此需要手动在在下游 TiDB 创建数据源同步起始位置对应的表结构。该教程的示例如下：

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

2.4.4.3.3 创建同步任务

1. 创建任务配置文件 task.yaml，配置增量同步模式，以及每个数据源的同步起点。完整的任务配置文件示例如下：

“‘yaml name: task-test # 任务名称，需要全局唯一 task-mode: incremental # 任务模式，设为 “incremental” 即只进行增量数据迁移

配置下游 TiDB 数据库实例访问信息 target-database: # 下游数据库实例配置 host: “127.0.0.1” port: 4000 user: “root” password: “” # 如果密码不为空，则推荐使用经过 dmctl 加密的密文

使用黑白名单配置需要同步的表 block-allow-list: # 数据源数据库实例匹配的表的 block-allow-list 过滤规则集，如果 DM 版本早于 v2.0.0-beta.2 则使用 black-white-list bw-rule-1: # 黑白名单配置项 ID do-dbs: [“log”] # 迁移哪些库

【可选配置】如果增量数据迁移需要重复迁移已经在全量数据迁移中完成迁移的数据，则需要开启 safe mode 避免增量数据迁移报错 ## 该场景多见于，全量迁移的数据不属于数据源的一个一致性快照，随后从一个早于全量迁移数据之前的位置开始同步增量数据 syncers: # sync 处理单元的运行配置参数 global: # 配置名称 safe-mode: true # 设置为 true，则将来自数据源的 INSERT 改写为 REPLACE，将 UPDATE 改写为 DELETE 与 REPLACE。

配置数据源 mysql-instances: - source-id: “mysql-01” # 数据源 ID，可以从数据源配置中获取 block-allow-list: “bw-rule-1” # 引入上面黑白名单配置 syncer-config-name: “global” # 引用上面的 syncers 增量数据配置 meta: # task-mode 为 incremental 且下游数据库的 checkpoint 不存在时 binlog 迁移开始的位置; 如果 checkpoint 存在，以 checkpoint 为准 binlog-name: “mysql-bin.000001” binlog-pos: 2022 binlog-gtid: “09bec856-ba95-11ea-850a-58f2b4af5188:1-9” “‘

2. 使用 start-task 命令创建同步任务：

```
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. 使用 query-status 查看同步任务，确认无报错信息：

```
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 测试同步任务

在数据源数据库插入新增数据：

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

此时数据源数据为：

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

查询下游数据库，可以发现 (3, 'msg3') 之后的数据已同步成功：

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

2.4.5 下游 TiDB 表结构有更多列的迁移场景

本文介绍如何在下游 TiDB 表结构比上游存在更多列的情况下，使用 DM 对表进行迁移。

2.4.5.1 数据源表

本文档示例所用的数据源实例如下所示：

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

2.4.5.2 迁移要求

在 TiDB 中定制创建表 log.messages，其表结构包含数据源中 log.messages 表的所有列，且比数据源的表结构有更多的列。在此需求下，将数据源表 log.messages 迁移到 TiDB 集群的表 log.messages。

注意：

- 下游 TiDB 相比于数据源多出的列必须指定默认值或允许为 NULL。
- 对于已经通过 DM 在正常迁移的表，可直接在下游 TiDB 中新增指定了默认值或允许为 NULL 的列而不会影响 DM 正常的数据迁移。

2.4.5.3 只迁移数据源增量数据到 TiDB ，并且下游 TiDB 表结构有更多列的场景问题处理

如果该迁移任务包含全量数据迁移，迁移任务可以正常运行；但是如果你使用其他方式进行了全量迁移，只是用 DM 进行增量数据复制，可以参考只迁移数据源增量数据到 TiDB创建数据迁移任务，同时还需手动在 DM 中设置用于解析 MySQL binlog 的表结构。

否则，创建任务后，运行 query-status 会返回类似如下数据迁移错误：

```
"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": ""
    }
]
```

出现以上错误的原因是 DM 迁移 binlog event 时，如果 DM 内部没有维护对应于该表的表结构，则会尝试使用下游当前的表结构来解析 binlog event 并生成相应的 DML 语句。如果 binlog event 里数据的列数与下游表结构的列数不一致时，则会产生上述错误。

此时，我们可以使用operate-schema 命令来为该表指定与 binlog event 匹配的表结构。如果你在进行分表合并的数据迁移，那么需要为每个分表按照如下步骤在 DM 中设置用于解析 MySQL binlog 的表结构。具体操作为：

1. 为数据源中需要迁移的表 log.messages 指定表结构，表结构需要对应 DM 将要开始同步的 binlog event 的数据。将对应的 CREATE TABLE 表结构语句并保存到文件，例如将以下表结构保存到 log.messages.sql 中。

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

2. 使用operate-schema 命令设置表结构（此时 task 应该由于上述错误而处于 Paused 状态）。

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

3. 使用resume-task 命令恢复处于 Paused 状态的任务。

4. 使用query-status 命令确认数据迁移任务是否运行正常。

3 部署使用

3.1 DM 集群软硬件环境需求

DM 支持主流的 Linux 操作系统，具体版本要求见下表：

Linux 操作系统平台	版本
Red Hat Enterprise Linux	7.3 及以上
CentOS	7.3 及以上
Oracle Enterprise Linux	7.3 及以上
Ubuntu LTS	16.04 及以上

DM 可以在 Intel 架构服务器环境及主流虚拟化环境中部署和运行。

3.1.1 服务器建议配置

DM 支持部署和运行在 Intel x86-64 架构的 64 位通用硬件服务器平台。对于开发，测试，及生产环境的服务器硬件配置（不包含操作系统本身的占用）有以下要求和建议：

3.1.1.1 开发及测试环境

组件	CPU	内存	本地存储	网络	实例数量（最低要
DM-master	4 核 +	8 GB+	SAS，200 GB+	千兆网卡	1
DM-worker	8 核 +	16 GB+	SAS，200 GB+（大于迁移数据的大小）	千兆网卡	上游 MySQL 实例

注意：

- 在功能验证的测试环境中的 DM-master 和 DM-worker 可以部署在同一台服务器上。
- 如进行性能相关的测试，避免采用低性能存储和网络硬件配置，防止对测试结果的正确性产生干扰。
- 如果仅验证功能，可以单机部署一个 DM-master，DM-worker 部署的数量至少是上游 MySQL 实例的数量。为了保证高可用性，建议部署更多的 DM-worker。
- DM-worker 在 `dump` 和 `load` 阶段需要存储全量数据，因此 DM-worker 的磁盘空间需要大于需要迁移数据的总量；如果迁移任务开启了 relay log，DM-worker 也需要一定的磁盘空间来存储上游的 binlog 数据。

3.1.1.2 生产环境

组件	CPU	内存	硬盘类型	网络	实例数量（最低
DM-master	4 核 +	8 GB+	SAS，200 GB+	千兆网卡	3
DM-worker	16 核 +	32 GB+	SSD，200 GB+（大于迁移数据的大小）	万兆网卡	大于上游 MySQ
监控	8 核 +	16 GB+	SAS，200 GB+	千兆网卡	1

注意：

- 在生产环境中，不建议将 DM-master 和 DM-worker 部署和运行在同一个服务器上，以防 DM-worker 对磁盘的写入干扰 DM-master 高可用组件使用磁盘。
- 在遇到性能问题时可参照配置调优尝试修改任务配置。调优效果不明显时，可以尝试升级服务器配置。

3.2 部署 DM 集群

3.2.1 使用 TiUP 部署 DM 集群

TiUP 是 TiDB 4.0 版本引入的集群运维工具，TiUP DM 是 TiUP 提供的使用 Golang 编写的集群管理组件，通过 TiUP DM 组件就可以进行日常的运维工作，包括部署、启动、关闭、销毁、扩缩容、升级 DM 集群以及管理 DM 集群参数。

目前 TiUP 可以支持部署 v2.0 及以上版本的 DM。本文将介绍不同集群拓扑的具体部署步骤。

> **注意：**
>
> 如果部署机器的操作系统支持 SELinux，请确保 SELinux 处于关闭状态。

3.2.1.1 前提条件

当 DM 执行全量数据复制任务时，每个 DM-worker 只绑定一个上游数据库。DM-worker 首先在上游导出全部数据，然后将数据导入下游数据库。因此，DM-worker 的主机需要有足够的存储空间，具体存储路径在后续创建迁移任务时指定。

另外，部署 DM 集群需参照DM 集群软硬件环境需求，满足相应要求。

3.2.1.2 第 1 步：在中控机上安装 TiUP 组件

使用普通用户登录中控机，以 `tidb` 用户为例，后续安装 TiUP 及集群管理操作均通过该用户完成：

1. 执行如下命令安装 TiUP 工具：

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

安装完成后，`~/.bashrc` 已将 TiUP 加入到路径中，你需要新开一个终端或重新声明全局变量 `source ~/.bashrc` 来使用 TiUP。

2. 安装 TiUP DM 组件：

```
tiup install dm dmctl
```

3.2.1.3 第 2 步：编辑初始化配置文件

请根据不同的集群拓扑，编辑 TiUP 所需的集群初始化配置文件。

请根据配置文件模板，新建一个配置文件 `topology.yaml`。如果有其他组合场景的需求，请根据多个模板自行调整。

可以使用 `tiup dm template > topology.yaml` 命令快速生成配置文件模板。

部署 3 个 DM-master、3 个 DM-worker 与 1 个监控组件的配置如下：

```
#全局变量适用于配置中的其他组件。如果组件实例中缺少一个特定值，
    ↪ 则相应的全局变量将用作默认值。
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"
    # 下列配置项用于覆盖 `server_configs.master` 的值。
    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
### 如果不需要确保 DM 集群高可用，则可只部署 1 个 DM-master 节点，且部署的 DM
    ↪ -worker 节点数量不少于上游待迁移的 MySQL/MariaDB 实例数。
### 如果需要确保 DM 集群高可用，则推荐部署 3 个 DM-master 节点，且部署的 DM-
```

```
    ↪ worker 节点数量大于上游待迁移的 MySQL/MariaDB 实例数（如 DM-worker
    ↪ 节点数量比上游实例数多 2 个）。
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"
    # 下列配置项用于覆盖 `server_configs.worker` 的值。
    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"
```

注意：

- 不建议在一台主机上运行太多 DM-worker。每个 DM-worker 至少应有 2 核 CPU 和 4 GiB 内存。

- 需要确保以下组件间端口可正常连通：
  - 各 DM-master 节点间的 peer_port（默认为 8291）可互相连通。
  - 各 DM-master 节点可连通所有 DM-worker 节点的 port（默认为 8262）。
  - 各 DM-worker 节点可连通所有 DM-master 节点的 port（默认为 8261）。
  - TiUP 节点可连通所有 DM-master 节点的 port（默认为 8261）。
  - TiUP 节点可连通所有 DM-worker 节点的 port（默认为 8262）。

更多 master_servers.host.config 参数说明，请参考 master parameter；更多 worker_servers.host.config 参数说明，请参考 worker parameter。

3.2.1.4 第 3 步：执行部署命令

注意：

通过 TiUP 进行集群部署可以使用密钥或者交互密码方式来进行安全认证：

- 如果是密钥方式，可以通过 -i 或者 --identity_file 来指定密钥的路径；
- 如果是密码方式，可以通过 -p 进入密码交互窗口；
- 如果已经配置免密登录目标机，则不需填写认证。

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

以上部署命令中：

- 通过 TiUP DM 部署的集群名称为 dm-test。
- 部署版本为 ${version}，可以通过执行 tiup list dm-master 来查看 TiUP 支持的最新版本。
- 初始化配置文件为 topology.yaml。
- --user root：通过 root 用户登录到目标主机完成集群部署，该用户需要有 ssh 到目标机器的权限，并且在目标机器有 sudo 权限。也可以用其他有 ssh 和 sudo 权限的用户完成部署。
- -i 及 -p：非必选项，如果已经配置免密登录目标机，则不需填写，否则选择其一即可。-i 为可登录到目标机的 root 用户（或 --user 指定的其他用户）的私钥，也可使用 -p 交互式输入该用户的密码。

- TiUP DM 使用内置的 SSH 客户端，如需使用系统自带的 SSH 客户端，请参考 TiUP DM 文档中使用中控机系统自带的 SSH 客户端连接集群章节进行设置。

预期日志结尾输出会有 `Deployed cluster `dm-test` successfully` 关键词，表示部署成功。

3.2.1.5 第 4 步：查看 TiUP 管理的集群情况

```
tiup dm list
```

TiUP 支持管理多个 DM 集群，该命令会输出当前通过 TiUP DM 管理的所有集群信息，包括集群名称、部署用户、版本、密钥信息等：

```
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 第 5 步：检查部署的 DM 集群情况

例如，执行如下命令检查 `dm-test` 集群情况：

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

预期输出包括 `dm-test` 集群中实例 ID、角色、主机、监听端口和状态（由于还未启动，所以状态为 Down/inactive）、目录信息。

3.2.1.7 第 6 步：启动集群

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

预期结果输出 `Started cluster `dm-test` successfully` 表示启动成功。

3.2.1.8 第 7 步：验证集群运行状态

通过以下 TiUP 命令检查集群状态：

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

在输出结果中，如果 Status 状态信息为 `Up`，说明集群状态正常。

3.2.1.9 第 8 步：使用 dmctl 管理迁移任务

dmctl 是用来控制集群运行命令的工具，推荐通过 TiUP 获取该工具。

dmctl 支持命令模式与交互模式，具体请见使用 dmctl 运维集群。

3.2.2 使用 TiUP 离线镜像部署 DM 集群（实验特性）

> **警告：**
>
> 本文描述特性仍为实验特性，不建议在生产环境下使用 TiUP 离线镜像部署 DM 集群。

本文介绍如何使用 TiUP 离线部署 DM 集群，具体的操作步骤如下。

3.2.2.1 第 1 步：准备 TiUP 离线组件包

- 在线环境中安装 TiUP 包管理器工具。

  1. 执行如下命令安装 TiUP 工具：

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

  2. 重新声明全局环境变量：

     ```
     source .bash_profile
     ```

  3. 确认 TiUP 工具是否安装：

     ```
     which tiup
     ```

- 使用 TiUP 制作离线镜像。

  1. 在一台和外网相通的机器上拉取需要的组件：

     ```
     export version=v2.0.3 # 可修改成实际需要的版本
     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
           ↪ }')
     ```

     该命令会在当前目录下创建一个名叫 `tidb-dm-${version}-linux-amd64` 的目录，里面包含 TiUP 管理的组件包。

  2. 通过 tar 命令将该组件包打包然后发送到隔离环境的中控机：

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

此时，tidb-dm-${version}-linux-amd64.tar.gz 就是一个独立的离线环境包。

3.2.2.2 第 2 步：部署离线环境 TiUP 组件

将离线包发送到目标集群的中控机后，执行以下命令安装 TiUP 组件：

```
export version=v2.0.3 # 可修改成实际需要的版本
tar xzvf tidb-dm-${version}-linux-amd64.tar.gz
sh tidb-dm-${version}-linux-amd64/local_install.sh
source /home/tidb/.bash_profile
```

local_install.sh 脚本会自动执行 tiup mirror set tidb-dm-${version}-linux-amd64 命令将当前镜像地址设置为 tidb-dm-${version}-linux-amd64。

若需将镜像切换到其他目录，可以通过手动执行 tiup mirror set <mirror-dir> 进行切换，切换后如果再想切换成官方镜像，可执行 tiup mirror set https://tiup-mirrors.pingcap.com。

3.2.2.3 第 3 步：编辑初始化配置文件

请根据不同的集群拓扑，编辑 TiUP 所需的集群初始化配置文件。

请根据配置文件模板，新建一个配置文件 topology.yaml。如果有其他组合场景的需求，请根据多个模板自行调整。

部署 3 个 DM-master、3 个 DM-worker 与 1 个监控组件的配置如下：

```
---
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
```

注意：

- 如果不需要确保 DM 集群高可用，则可只部署 1 个 DM-master 节点，且部署的 DM-worker 节点数量不少于上游待迁移的 MySQL/MariaDB 实例数。
- 如果需要确保 DM 集群高可用，则推荐部署 3 个 DM-master 节点，且部署的 DM-worker 节点数量大于上游待迁移的 MySQL/MariaDB 实例数（如 DM-worker 节点数量比上游实例数多 2 个）。
- 对于需要全局生效的参数，请在配置文件中 `server_configs` 的对应组件下配置。
- 对于需要某个节点生效的参数，请在具体节点的 `config` 中配置。
- 配置的层次结构使用 `.` 表示。如：`log.slow-threshold`。更多格式说明，请参考 TiUP 配置参数模版。
- 更多参数说明，请参考 master `config.toml.example`、worker `config.toml.example`。
- 需要确保以下组件间端口可正常连通：
  - 各 DM-master 节点间的 `peer_port`（默认为 8291）可互相连通。
  - 各 DM-master 节点可连通所有 DM-worker 节点的 `port`（默认为 8262）。
  - 各 DM-worker 节点可连通所有 DM-master 节点的 `port`（默认为 8261）。
  - TiUP 节点可连通所有 DM-master 节点的 `port`（默认为 8261）。
  - TiUP 节点可连通所有 DM-worker 节点的 `port`（默认为 8262）。

3.2.2.4 第 4 步：执行部署命令

注意：

通过 TiUP 进行集群部署可以使用密钥或者交互密码方式来进行安全认证：

- 如果是密钥方式，可以通过 -i 或者 --identity_file 来指定密钥的路径；
- 如果是密码方式，可以通过 -p 进入密码交互窗口；
- 如果已经配置免密登录目标机，则不需填写认证。

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

以上部署命令中：

- 通过 TiUP DM 部署的集群名称为 dm-test。
- 部署版本为 ${version}，可以通过执行 tiup list dm-master 来查看 TiUP 支持的最新版本。
- 初始化配置文件为 topology.yaml。
- --user root：通过 root 用户登录到目标主机完成集群部署，该用户需要有 ssh 到目标机器的权限，并且在目标机器有 sudo 权限。也可以用其他有 ssh 和 sudo 权限的用户完成部署。
- -i 及 -p：非必选项，如果已经配置免密登录目标机，则不需填写，否则选择其一即可。-i 为可登录到目标机的 root 用户（或 --user 指定的其他用户）的私钥，也可使用 -p 交互式输入该用户的密码。
- TiUP DM 使用内置的 SSH 客户端，如需使用系统自带的 SSH 客户端，请参考 TiUP DM 文档中使用中控机系统自带的 SSH 客户端连接集群章节进行设置。

预期日志结尾输出会有 Deployed cluster `dm-test` successfully 关键词，表示部署成功。

3.2.2.5 第 5 步：查看 TiUP 管理的集群情况

```
tiup dm list
```

TiUP 支持管理多个 DM 集群，该命令会输出当前通过 TiUP DM 管理的所有集群信息，包括集群名称、部署用户、版本、密钥信息等：

```
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 第 6 步：检查部署的 DM 集群情况

例如，执行如下命令检查 dm-test 集群情况：

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

预期输出包括 dm-test 集群中实例 ID、角色、主机、监听端口和状态（由于还未启动，所以状态为 Down/inactive）、目录信息。

3.2.2.7 第 7 步：启动集群

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

预期结果输出 Started cluster `dm-test` successfully 表示启动成功。

3.2.2.8 第 8 步：验证集群运行状态

通过以下 TiUP 命令检查集群状态：

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

在输出结果中，如果 Status 状态信息为 Up，说明集群状态正常。

3.2.3 使用 DM binary 部署 DM 集群

本文将介绍如何使用 DM binary 快速部署 DM 集群。

> 注意：
> 对于生产环境，推荐使用 TiUP 部署 DM 集群及相关监控组件。

3.2.3.1 准备工作

使用下表中的链接下载官方 binary：

安装包	操作系统	架构	SHA256 校验和
https ↪ :// ↪ download ↪ . ↪ pingcap ↪ . ↪ org ↪ / ↪ dm ↪ -{ ↪ version ↪ }- ↪ linux ↪ - ↪ amd64 ↪ . ↪ tar ↪ . ↪ gz ↪	Linux	amd64	https ↪ :// ↪ download ↪ . ↪ pingcap ↪ . ↪ org ↪ / ↪ dm ↪ -{ ↪ version ↪ }- ↪ linux ↪ - ↪ amd64 ↪ . ↪ sha256 ↪

注意：

下载链接中的 `{version}` 为 DM 的版本号。例如，`v1.0.1` 版本的下载链接为 `https://download.pingcap.org/dm-v1.0.1-linux-amd64.tar.gz`。可以通过 DM Release 查看当前已发布版本。

下载的文件中包括子目录 bin 和 conf。bin 目录下包含 dm-master、dm-worker 以及 dmctl 的二进制文件。conf 目录下有相关的示例配置文件。

3.2.3.2 使用样例

假设在五台服务器上部署两个 DM-worker 实例和三个 DM-master 实例。各个节点的信息如下：

实例	服务器地址	端口
DM-master1	192.168.0.4	8261
DM-master2	192.168.0.5	8261

实例	服务器地址	端口
DM-master3	192.168.0.6	8261
DM-worker1	192.168.0.7	8262
DM-worker2	192.168.0.8	8262

下面以此为例，说明如何部署 DM。

注意：

- 在单机部署多个 DM-master 或 DM-worker 时，需要确保每个实例的端口以及运行命令的当前目录各不相同。
- 如果不需要确保 DM 集群高可用，则可只部署 1 个 DM-master 节点，且部署的 DM-worker 节点数量不少于上游待迁移的 MySQL/MariaDB 实例数。
- 如果需要确保 DM 集群高可用，则推荐部署 3 个 DM-master 节点，且部署的 DM-worker 节点数量大于上游待迁移的 MySQL/MariaDB 实例数（如 DM-worker 节点数量比上游实例数多 2 个）。
- 需要确保以下组件间端口可正常连通：
  - 各 DM-master 节点间的 8291 端口可互相连通。
  - 各 DM-master 节点可连通所有 DM-worker 节点的 8262 端口。
  - 各 DM-worker 节点可连通所有 DM-master 节点的 8261 端口。

3.2.3.2.1 部署 DM-master

DM-master 提供命令行参数和配置文件两种配置方式。

使用命令行参数部署 DM-master

DM-master 的命令行参数说明：

```
./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
```

注意：

某些情况下，无法使用命令行参数来配置 DM-master，因为有的配置并未暴露给命令行。

使用配置文件部署 DM-master

推荐使用配置文件，把以下配置文件内容写入到 `conf/dm-master1.toml` 中。

DM-master 的配置文件：

```
### Master Configuration.

name = "master1"

### 日志配置
log-level = "info"
log-file = "dm-master.log"

### DM-master 监听地址
master-addr = "192.168.0.4:8261"
```

```
### DM-master 节点的对等 URL
peer-urls = "192.168.0.4:8291"

### 初始集群中所有 DM-master 的 advertise-peer-urls 的值
initial-cluster = "master1=http://192.168.0.4:8291,master2=http
    ↪ ://192.168.0.5:8291,master3=http://192.168.0.6:8291"
```

在终端中使用下面的命令运行 DM-master：

> **注意：**
>
> 执行该命令后控制台不会输出日志，可以通过 `tail -f dm-master.log` 查看运行日志。

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

对于 DM-master2 和 DM-master3，修改配置文件中的 `name` 为 `master2` 和 `master3`，并将 `peer-urls` 的值改为 `192.168.0.5:8291` 和 `192.168.0.6:8291` 即可。

3.2.3.2.2 部署 DM-worker

DM-worker 提供命令行参数和配置文件两种配置方式。

使用命令行参数部署 DM-worker

查看 DM-worker 的命令行参数说明：

```
./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
```

> **注意：**
>
> 某些情况下，无法使用命令行参数的方法来配置 DM-worker，因为有的配置并未暴露给命令行。

使用配置文件部署 DM-worker

推荐使用配置文件来配置 DM-worker，把以下配置文件内容写入到 `conf/dm-worker1.toml` 中。

DM-worker 的配置文件：

```
### Worker Configuration.

name = "worker1"

### 日志配置
log-level = "info"
log-file = "dm-worker.log"

### DM-worker 的地址
worker-addr = ":8262"

### 对应集群中 DM-master 配置中的 master-addr
join = "192.168.0.4:8261,192.168.0.5:8261,192.168.0.6:8261"
```

在终端中使用下面的命令运行 DM-worker：

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

对于 DM-worker2，修改配置文件中的 `name` 为 `worker2` 即可。

这样，DM 集群就部署成功了。

3.2.4 使用 Kubernetes

3.3 使用 DM 迁移数据

本文介绍如何使用 DM 工具迁移数据。

3.3.1 第 1 步：部署 DM 集群

推荐使用 TiUP 部署 DM 集群；也可以使用 binary 部署 DM 集群用于体验或测试。

> 注意：
>
> - 在 DM 所有的配置文件中，对于数据库密码推荐使用 dmctl 加密后的密文。如果数据库密码为空，则不需要加密。关于如何使用 dmctl 加密明文密码，参考使用 dmctl 加密数据库密码。
> - 上下游数据库用户必须拥有相应的读写权限。

3.3.2 第 2 步：检查集群信息

使用 TiUP 部署 DM 集群后，相关配置信息如下：

- DM 集群相关组件配置信息

组件	主机	端口
dm_worker1	172.16.10.72	8262
dm_worker2	172.16.10.73	8262
dm_master	172.16.10.71	8261

- 上下游数据库实例相关信息

数据库实例	主机	端口	用户名	加密密码
上游 MySQL-1	172.16.10.81	3306	root	VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=
上游 MySQL-2	172.16.10.82	3306	root	VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=
下游 TiDB	172.16.10.83	4000	root	

上游 MySQL 数据库实例用户所需权限参见上游 MySQL 实例配置前置检查介绍。

3.3.3 第 3 步：创建数据源

1. 将 MySQL-1 的相关信息写入到 conf/source1.yaml 中：

```
# MySQL1 Configuration.

source-id: "mysql-replica-01"

# DM-worker 是否使用全局事务标识符（GTID）拉取 binlog。使用前提是在上游
    ↪ MySQL 已开启 GTID 模式。
enable-gtid: false

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

2. 在终端中执行下面的命令，使用 tiup dmctl 将 MySQL-1 的数据源配置加载到 DM 集群中：

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

3. 对于 MySQL-2，修改配置文件中的相关信息，并执行相同的 dmctl 命令。

3.3.4 第 4 步：配置任务

假设需要将 MySQL-1 和 MySQL-2 实例的 test_db 库的 test_table 表以全量 + 增量的模式迁移到下游 TiDB 的 test_db 库的 test_table 表。

编辑任务配置文件 task.yaml：

```
## 任务名，多个同时运行的任务不能重名。
name: "test"
## 全量+增量（all）迁移模式。
task-mode: "all"
## 下游 TiDB 配置信息。
target-database:
  host: "172.16.10.83"
  port: 4000
  user: "root"
  password: ""

## 当前数据迁移任务需要的全部上游 MySQL 实例配置。
mysql-instances:
```

```
-
  # 上游实例或者复制组 ID，参考 `inventory.ini` 的 `source_id` 或者 `dm-
    ↪ master.toml` 的 `source-id 配置`。
  source-id: "mysql-replica-01"
  # 需要迁移的库名或表名的黑白名单的配置项名称，用于引用全局的黑白名单配置，
    ↪ 全局配置见下面的 `block-allow-list` 的配置。
  block-allow-list: "global"     # 如果 DM 版本早于 v2.0.0-beta.2 则使用
    ↪ black-white-list。
  # dump 处理单元的配置项名称，用于引用全局的 dump 处理单元配置。
  mydumper-config-name: "global"

-
  source-id: "mysql-replica-02"
  block-allow-list: "global"     # 如果 DM 版本早于 v2.0.0-beta.2 则使用
    ↪ black-white-list。
  mydumper-config-name: "global"

## 黑白名单全局配置，各实例通过配置项名引用。
block-allow-list:                # 如果 DM 版本早于 v2.0.0-beta.2 则使用
  ↪ black-white-list。
  global:
    do-tables:                   # 需要迁移的上游表的白名单。
    - db-name: "test_db"         # 需要迁移的表的库名。
      tbl-name: "test_table"     # 需要迁移的表的名称。

## dump 处理单元全局配置，各实例通过配置项名引用。
mydumpers:
  global:
    extra-args: ""
```

3.3.5 第 5 步：启动任务

为了提前发现数据迁移任务的一些配置错误，DM 中增加了前置检查功能：

- 启动数据迁移任务时，DM 自动检查相应的权限和配置。
- 也可使用 `check-task` 命令手动前置检查上游的 MySQL 实例配置是否符合 DM 的配置要求。

> **注意：**
>
> 第一次启动数据迁移任务时，必须确保上游数据库已配置。否则，启动任务时会报错。

使用 tiup dmctl 执行以下命令启动数据迁移任务。其中，task.yaml 是之前编辑的配置文件。

```
tiup dmctl --master-addr 172.16.10.71:8261 start-task ./task.yaml
```

- 如果执行该命令后返回的结果如下，则表明任务已成功启动。

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

- 如果任务启动失败，可根据返回结果的提示进行配置变更后执行 start-task task.yaml 命令重新启动任务。

3.3.6 第 6 步：查询任务

如需了解 DM 集群中是否存在正在运行的迁移任务及任务状态等信息，可使用 tiup dmctl 执行以下命令进行查询：

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

3.3.7 第 7 步：停止任务

如果不再需要进行数据迁移，可以使用 tiup dmctl 执行以下命令停止迁移任务：

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

其中的 test 是 task.yaml 配置文件中 name 配置项设置的任务名。

3.3.8 第 8 步：监控任务与查看日志

如果使用 TiUP 部署 DM 集群时，正确部署了 Prometheus、Alertmanager 与 Grafana，且其地址均为 172.16.10.71。可在浏览器中打开 http://172.16.10.71:9093 进入 Alertmanager 查看 DM 告警信息；可在浏览器中打开 http://172.16.10.71:3000 进入 Grafana，选择 DM 的 dashboard 查看 DM 相关监控项。

DM 在运行过程中，DM-worker, DM-master 及 dmctl 都会通过日志输出相关信息。各组件的日志目录如下：

- DM-master 日志目录：通过 DM-master 进程参数 `--log-file` 设置。如果使用 TiUP 部署 DM，则日志目录位于 `{log_dir}`。
- DM-worker 日志目录：通过 DM-worker 进程参数 `--log-file` 设置。如果使用 TiUP 部署 DM，则日志目录位于 `{log_dir}`。

3.4 DM 集群性能测试

本文档介绍如何构建测试场景对 DM 集群进行性能测试，包括数据迁移速度、延迟等。

3.4.1 迁移数据流

可以使用简单的迁移数据流来测试 DM 集群的数据迁移性能，即单个 MySQL 实例到 TiDB 的数据迁移：MySQL -> DM -> TiDB。

3.4.2 部署测试环境

- 使用 TiUP 部署 TiDB 测试集群，所有配置使用 TiUP 提供的默认配置。
- 部署 MySQL 服务，开启 ROW 模式 binlog，其他配置项使用默认配置。
- 部署 DM 集群，部署一个 DM-worker 和一个 DM-master 即可。

3.4.3 性能测试

3.4.3.1 迁移数据表结构

使用如下结构的表进行性能测试：

```
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 全量导入性能测试用例

3.4.3.2.1 生成测试数据

使用 sysbench 在上游创建测试表，并生成全量导入的测试数据。sysbench 生成数据的命令如下所示：

```
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 创建数据迁移任务

1. 创建上游 MySQL 的 source，将 source-id 配置为 source-1。详细操作方法参考：加载数据源配置。
2. 创建 full 模式的 DM 迁移任务，示例任务配置文件如下：

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

使用实际测试环境中 TiDB 的信息配置 target-database: host: “192.168.0.1” port: 4000 user: “root” password: “”

mysql-instances: - source-id: “source-1” block-allow-list: “instance” # 如果 DM 版本早于 v2.0.0-beta.2 则使用 black-white-list mydumper-config-name: “global” loader-thread: 16

配置 sysbench 生成数据所在的库的名称 block-allow-list: # 如果 DM 版本早于 v2.0.0-beta.2 则使用 black-white-list instance: do-dbs: [“dm_benchmark”]

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

创建数据迁移任务的详细操作参考创建数据迁移任务。

注意：

- 在 mydumpers 配置项中使用 rows 选项，可以开启单表多线程并发导出，加快数据导出速度。
- mysql-instances 配置中的 loader-thread 以及 mydumpers 配置项中的 rows 和 threads 可以做适当调整，测试在不同配置下对性能的影响。

3.4.3.2.3 获取测试结果

观察 DM-worker 日志，当出现 `all data files have been finished` 时，表示全量数据导入完成，此时可以看到消耗时间。示例日志如下：

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

根据测试数据的数据量和导入消耗时间，可以算出全量数据的迁移速度。

3.4.3.3 增量复制性能测试用例

3.4.3.3.1 初始化表

使用 `sysbench` 在上游创建测试表。

3.4.3.3.2 创建数据迁移任务

1. 创建上游 MySQL 的 source, source-id 配置为 `source-1`（如果在全量迁移性能测试中已经创建，则不需要再次创建）。详细操作方法参考：加载数据源配置。
2. 创建 `all` 模式的 DM 迁移任务，示例任务配置文件如下：

“‘yaml
name: test-all
task-mode: all

使用实际测试环境中 TiDB 的信息配置 target-database: host: “192.168.0.1” port: 4000 user: “root” password: “”

mysql-instances: - source-id: “source-1” block-allow-list: “instance” # 如果 DM 版本早于 v2.0.0-beta.2 则使用 black-white-list syncer-config-name: “global”

配置 sysbench 生成数据所在的库的名称 block-allow-list: # 如果 DM 版本早于 v2.0.0-beta.2 则使用 black-white-list instance: do-dbs: [“dm_benchmark”]

syncers: global: worker-count: 16 batch: 100 “‘

创建数据迁移任务的详细操作参考创建数据迁移任务。

> **注意：**
>
> `syncers` 配置项中的 `worker-count` 和 `batch` 可以做适当调整，测试在不同配置下性能的差异。

3.4.3.3.3 生成增量数据

执行 sysbench 命令在上游持续生成增量数据：

```
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
```

> 注意：
>
> 可以通过调整 sysbench 的语句类型，测试在不同业务场景下 DM 的数据迁移性能。

3.4.3.3.4 获取测试结果

通过 query-status 命令观测 DM 的迁移状态，通过 Grafana 观测 DM 的监控指标。主要包括单位时间内完成的 job 数量 finished sqls jobs 等，详细的监控指标说明参考Binlog replication 监控指标。

4 运维操作

4.1 集群运维工具

4.1.1 使用 TiUP 运维 DM 集群

本文介绍如何使用 TiUP 的 DM 组件运维 DM 集群。

如果你还未部署 DM 集群，可参考使用 TiUP 部署 DM 集群。

> 注意：
>
> - 需要确保以下组件间端口可正常连通：
>   - 各 DM-master 节点间的 peer_port（默认为 8291）可互相连通。
>   - 各 DM-master 节点可连通所有 DM-worker 节点的 port（默认为 8262）。
>   - 各 DM-worker 节点可连通所有 DM-master 节点的 port（默认为 8261）。
>   - TiUP 节点可连通所有 DM-master 节点的 port（默认为 8261）。
>   - TiUP 节点可连通所有 DM-worker 节点的 port（默认为 8262）。

TiUP DM 组件的帮助信息如下：

```
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 查看集群列表

集群部署成功后，可以通过 `tiup dm list` 命令在集群列表中查看该集群：

```
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 启动集群

集群部署成功后，可以执行以下命令启动该集群。如果忘记了部署的集群名字，可以使用 `tiup dm list` 命令查看。

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

4.1.1.3 检查集群状态

如果想查看集群中每个组件的运行状态，逐一登录到各个机器上查看显然很低效。因此，TiUP 提供了 `tiup dm display` 命令，用法如下：

```
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
```

Status 列用 Up 或者 Down 表示该服务是否正常。对于 DM-master 组件，同时可能会带有 |L 表示该 DM-master 是 Leader, 对于 DM-worker 组件，Free 表示当前 DM-worker 没有与上游绑定。

4.1.1.4 缩容节点

缩容即下线服务，最终会将指定的节点从集群中移除，并删除遗留的相关数据文件。

缩容操作进行时，内部对 DM-master、DM-worker 组件的操作流程为：

1. 停止组件进程
2. 调用 DM-master 删除 member 的 API
3. 清除节点的相关数据文件

缩容命令的基本用法：

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

它需要指定至少两个参数，一个是集群名字，另一个是节点 ID。节点 ID 可以参考上一节使用 tiup dm display 命令获取。

比如想缩容 172.16.5.140 上的 DM-worker 节点 (DM-master 的缩容类似)，可以执行：

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

4.1.1.5 扩容节点

扩容的内部逻辑与部署类似，TiUP DM 组件会先保证节点的 SSH 连接，在目标节点上创建必要的目录，然后执行部署并且启动服务。

例如，在集群 prod-cluster 中扩容一个 DM-worker 节点 (DM-master 的扩容类似)：

1. 新建 scale.yaml 文件，添加新增的 woker 节点信息：

> 注意：
> 需要新建一个拓扑文件，文件中只写入扩容节点的描述信息，不要包含已存在的节点。
> 其他更多配置项 (如：部署目录等) 请参考 TiUP 配置参数模版。

```
---

worker_servers:
  - host: 172.16.5.140
```

2. 执行扩容操作。TiUP DM 根据 scale.yaml 文件中声明的端口、目录等信息在集群中添加相应的节点：

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

执行完成之后可以通过 `tiup dm display prod-cluster` 命令检查扩容后的集群状态。

4.1.1.6 滚动升级

> 注意：
>
> 从 v2.0.5 版本开始，dmctl 支持导出和导入集群的数据源和任务配置。
>
> 升级前，可使用 `config export` 命令导出集群的配置文件，升级后如需降级回退到旧版本，可重建旧集群后，使用 `config import` 导入之前的配置。
>
> 对于 v2.0.5 之前版本的集群，可使用 v2.0.5 及之后版本的 dmctl 导出和导入集群配置。
>
> 对于 v2.0.2 之后的版本，导入集群配置时暂不支持自动恢复 relay worker 相关配置，可手动执行 `start-relay` 命令开启 relay log。

滚动升级过程中尽量保证对前端业务透明、无感知，其中对不同节点有不同的操作。

4.1.1.6.1 升级操作

可使用 `tiup dm upgrade` 命令来升级集群。例如，把集群升级到 v2.0.1 的命令为：

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

4.1.1.7 更新配置

如果想要动态更新组件的配置，TiUP DM 组件为每个集群保存了一份当前的配置，如果想要编辑这份配置，则执行 `tiup dm edit-config <cluster-name>` 命令。例如：

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

然后 TiUP DM 组件会使用 vi 打开配置文件供编辑（如果你想要使用其他编辑器，请使用 `EDITOR` 环境变量自定义编辑器，例如 `export EDITOR=nano`），编辑完之后保存即可。此时的配置并没有应用到集群，如果想要让它生效，还需要执行：

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

该操作会将配置发送到目标机器，滚动重启集群，使配置生效。

4.1.1.8 更新组件

常规的升级集群可以使用 upgrade 命令，但是在某些场景下（例如 Debug），可能需要用一个临时的包替换正在运行的组件，此时可以用 patch 命令：

```
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'
```

例如，有一个 DM-master 的 hotfix 包放在 /tmp/dm-master-hotfix.tar.gz，如果此时想要替换集群上的所有 DM-master，则可以执行：

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

或者只替换其中一个 DM-master：

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

4.1.1.9 查看操作日志

操作日志的查看可以借助 audit 命令，其用法如下：

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

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

在不使用 [audit-id] 参数时，该命令会显示执行的命令列表，如下：

```
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
```

第一列为 audit-id，如果想看某个命令的执行日志，则传入这个 audit-id：

```
tiup dm audit 4D5kQY
```

4.1.1.10 在集群节点机器上执行命令

exec 命令可以很方便地到集群的机器上执行命令，使用方式如下：

```
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)
```

例如，如果要到所有的 DM 节点上执行 ls /tmp，则可以执行：

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

4.1.1.11 集群控制工具 (dmctl)

TiUP 集成了 DM 的控制工具 dmctl：

运行命令如下：

```
tiup dmctl [args]
```

指定 dmctl 版本：

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

例如，以前添加 source 命令为 dmctl --master-addr master1:8261 operate-source
↪ create /tmp/source1.yml，集成到 TiUP 中的命令为：

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

4.1.1.12 使用中控机系统自带的 SSH 客户端连接集群

在以上所有操作中，涉及到对集群机器的操作都是通过 TiUP 内置的 SSH 客户端连接集群执行命令，但是在某些场景下，需要使用系统自带的 SSH 客户端来对集群执行操作，比如：

- 使用 SSH 插件来做认证
- 使用定制的 SSH 客户端

此时可以通过命令行参数 `--native-ssh` 启用系统自带命令行：

- 部署集群: `tiup dm deploy <cluster-name> <version> <topo> --native-ssh`
- 启动集群: `tiup dm start <cluster-name> --native-ssh`
- 升级集群: `tiup dm upgrade ... --native-ssh`

所有涉及集群操作的步骤都可以加上 `--native-ssh` 来使用系统自带的客户端。

也可以使用环境变量 `TIUP_NATIVE_SSH` 来指定是否使用本地 SSH 客户端，避免每个命令都需要添加 `--native-ssh` 参数：

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

若环境变量和 `--native-ssh` 同时指定，则以 `--native-ssh` 为准。

> 注意：
>
> 在部署集群的步骤中，若需要使用密码的方式连接 (-p)，或者密钥文件设置了 passphrase，则需要保证中控机上安装了 sshpass，否则连接时会报错。

4.1.2 使用 dmctl 运维集群

> 注意：
>
> 对于用 TiUP 部署的 DM 集群，推荐直接使用`tiup dmctl` 命令。

dmctl 是用来运维 DM 集群的命令行工具，支持交互模式和命令模式。

4.1.2.1 dmctl 交互模式

进入交互模式，与 DM-master 进行交互：

> **注意：**
>
> 交互模式下不具有 bash 的特性，比如不需要通过引号传递字符串参数而应当直接传递。

```
./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          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`/`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 dmctl 命令模式

命令模式跟交互模式的区别是，执行命令时只需要在 dmctl 命令后紧接着执行任务操作，任务操作同交互模式的参数一致。

> **注意：**
>
> - 一条 dmctl 命令只能跟一个任务操作
> - 从 v2.0.4 版本开始，支持从环境变量 (DM_MASTER_ADDR) 里读取 -master-addr 参数

```
./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 [-s source ...] <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 address.
  --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
     ↪ .
```

4.1.3 使用 OpenAPI 运维集群

> **警告：**
> 当前该功能为实验特性，默认关闭，不建议在生产环境中使用。

DM 提供 OpenAPI 功能，你可以通过 OpenAPI 对 DM 集群进行查询和运维操作。OpenAPI 的总体功能和 dmctl 工具类似。如需开启该功能，请在 DM-master 的配置文件中增加如下配置项：

```
[experimental]
openapi = true
```

> 注意：
>
> - DM 提供符合 OpenAPI 3.0.0 标准的 Spec 文档，其中包含了所有 API 的请求参数和返回体，你可自行复制到如 Swagger Editor 等工具中在线预览文档。
> - 部署 DM-master 后，你可访问 `http://{master-addr}/api/v1/docs` 在线预览文档。

你可以通过 OpenAPI 完成 DM 集群的如下运维操作：

4.1.3.1 集群相关 API

- 获取 DM-master 节点信息
- 下线 DM-master 节点
- 获取 DM-worker 节点信息
- 下线 DM-worker 节点

4.1.3.2 数据源相关 API

- 创建数据源
- 获取数据源列表
- 删除数据源
- 获取数据源状态
- 对数据源开启 relay-log 功能
- 对数据源停止 relay-log 功能
- 对数据源暂停 relay-log 功能
- 对数据源恢复 relay-log 功能
- 更改数据源和 DM-worker 的绑定关系
- 获取数据源的数据库名列表
- 获取数据源的指定数据库的表名列表

4.1.3.3 同步任务相关 API

- 创建同步任务
- 获取同步任务列表
- 停止同步任务
- 获取同步任务状态
- 暂停同步任务
- 恢复同步任务
- 获取同步任务关联数据源的数据库名列表
- 获取同步任务关联数据源的数据表名列表
- 获取同步任务关联数据源的数据表的创建语句
- 更新同步任务关联数据源的数据表的创建语句
- 删除同步任务关联数据源的数据表

本文档以下部分描述当前提供的 API 的具体使用方法。

4.1.3.4 API 统一错误格式

对 API 发起的请求后，如发生错误，返回错误信息的格式如下所示：

```
{
    "error_msg": "",
    "error_code": ""
}
```

如上所示，`error_msg` 描述错误信息，`error_code` 则是对应的错误码。

4.1.3.5 获取 DM-master 节点信息

该接口是一个同步接口，请求成功会返回对应节点的状态信息。

4.1.3.5.1 请求 URI

`GET /api/v1/cluster/masters`

4.1.3.5.2 使用样例

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/cluster/masters' \
  -H 'accept: application/json'
```

```
{
  "total": 1,
  "data": [
    {
      "name": "master1",
```

```
      "alive": true,
      "leader": true,
      "addr": "127.0.0.1:8261"
    }
  ]
}
```

4.1.3.6 下线 DM-master 节点

该接口是一个同步接口，请求成功后返回体的 Status Code 是 204。

4.1.3.6.1 请求 URI

```
DELETE /api/v1/cluster/masters/{master-name}
```

4.1.3.6.2 使用样例

```
curl -X 'DELETE' \
  'http://127.0.0.1:8261/api/v1/cluster/masters/master1' \
  -H 'accept: */*'
```

4.1.3.7 获取 DM-worker 节点信息

该接口是一个同步接口，请求成功会返回对应节点的状态信息。

4.1.3.7.1 请求 URI

```
GET /api/v1/cluster/workers
```

4.1.3.7.2 使用样例

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/cluster/workers' \
  -H 'accept: application/json'
```

```
{
  "total": 1,
  "data": [
    {
      "name": "worker1",
      "addr": "127.0.0.1:8261",
      "bound_stage": "bound",
      "bound_source_name": "mysql-01"
    }
  ]
```

```
}
```

4.1.3.8 下线 DM-worker 节点

该接口是一个同步接口，请求成功后返回体的 Status Code 是 204。

4.1.3.8.1 请求 URI

```
DELETE /api/v1/cluster/workers/{worker-name}
```

4.1.3.8.2 使用样例

```
curl -X 'DELETE' \
  'http://127.0.0.1:8261/api/v1/cluster/workers/worker1' \
  -H 'accept: */*'
```

4.1.3.9 创建数据源

该接口是一个同步接口，请求成功会返回对应数据源信息。

4.1.3.9.1 请求 URI

```
POST /api/v1/sources
```

4.1.3.9.2 使用样例

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/sources' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "source_name": "mysql-01",
  "host": "127.0.0.1",
  "port": 3306,
  "user": "root",
  "password": "123456",
  "enable_gtid": false,
  "security": {
    "ssl_ca_content": "",
    "ssl_cert_content": "",
    "ssl_key_content": "",
    "cert_allowed_cn": [
      "string"
    ]
  },
```

```
  "purge": {
    "interval": 3600,
    "expires": 0,
    "remain_space": 15
  }
}'
```

```
{
  "source_name": "mysql-01",
  "host": "127.0.0.1",
  "port": 3306,
  "user": "root",
  "password": "123456",
  "enable_gtid": false,
  "security": {
    "ssl_ca_content": "",
    "ssl_cert_content": "",
    "ssl_key_content": "",
    "cert_allowed_cn": [
      "string"
    ]
  },
  "purge": {
    "interval": 3600,
    "expires": 0,
    "remain_space": 15
  },
  "status_list": [
    {
      "source_name": "mysql-replica-01",
      "worker_name": "worker-1",
      "relay_status": {
        "master_binlog": "(mysql-bin.000001, 1979)",
        "master_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
        "relay_dir": "./sub_dir",
        "relay_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
        "relay_catch_up_master": true,
        "stage": "Running"
      },
      "error_msg": "string"
    }
  ]
}
```

4.1.3.10 获取数据源列表

该接口是一个同步接口，请求成功会返回数据源列表信息。

4.1.3.10.1 请求 URI

```
GET /api/v1/sources
```

4.1.3.10.2 使用样例

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/sources?with_status=true' \
  -H 'accept: application/json'
```


```
{
  "data": [
    {
      "enable_gtid": false,
      "host": "127.0.0.1",
      "password": "******",
      "port": 3306,
      "purge": {
        "expires": 0,
        "interval": 3600,
        "remain_space": 15
      },
      "security": null,
      "source_name": "mysql-01",
      "user": "root"
    },
    {
      "enable_gtid": false,
      "host": "127.0.0.1",
      "password": "******",
      "port": 3307,
      "purge": {
        "expires": 0,
        "interval": 3600,
        "remain_space": 15
      },
      "security": null,
      "source_name": "mysql-02",
      "user": "root"
    }
  ],
  "total": 2
```

```
}
```

4.1.3.11 删除数据源

该接口是一个同步接口，请求成功后返回体的 Status Code 是 204。

4.1.3.11.1 请求 URI

```
DELETE /api/v1/sources/{source-name}
```

4.1.3.11.2 使用样例

```
curl -X 'DELETE' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01?force=true' \
  -H 'accept: application/json'
```

4.1.3.12 获取数据源状态

该接口是一个同步接口，请求成功会返回对应节点的状态信息。

4.1.3.12.1 请求 URI

```
GET /api/v1/sources/{source-name}/status
```

4.1.3.12.2 使用样例

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-replica-01/status' \
  -H 'accept: application/json'
```

```
{
  "total": 1,
  "data": [
    {
      "source_name": "mysql-replica-01",
      "worker_name": "worker-1",
      "relay_status": {
        "master_binlog": "(mysql-bin.000001, 1979)",
        "master_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
        "relay_dir": "./sub_dir",
        "relay_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
        "relay_catch_up_master": true,
        "stage": "Running"
      },
      "error_msg": "string"
```

```
    }
  ]
}
```

4.1.3.13 对数据源开启 relay-log 功能

这是一个异步接口，请求成功的 Status Code 是 200，可通过获取数据源状态接口获取最新的状态。

4.1.3.13.1 请求 URI

```
PATCH /api/v1/sources/{source-name}/start-relay
```

4.1.3.13.2 使用样例

```
curl -X 'PATCH' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01/start-relay' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "worker_name_list": [
    "worker-1"
  ],
  "relay_binlog_name": "mysql-bin.000002",
  "relay_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
  "relay_dir": "./relay_log"
}'
```

4.1.3.14 对数据源停止 relay-log 功能

这是一个异步接口，请求成功的 Status Code 是 200，可通过获取数据源状态接口获取最新的状态。

4.1.3.14.1 请求 URI

```
PATCH /api/v1/sources/{source-name}/stop-relay
```

4.1.3.14.2 使用样例

```
curl -X 'PATCH' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01/stop-relay' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "worker_name_list": [
```

```
    "worker-1"
  ]
}'
```

4.1.3.15 对数据源暂停 relay-log 功能

这是一个异步接口，请求成功的 Status Code 是 200，可通过获取数据源状态接口获取最新的状态。

4.1.3.15.1 请求 URI

```
PATCH /api/v1/sources/{source-name}/pause-relay
```

4.1.3.15.2 使用样例

```
curl -X 'PATCH' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01/pause-relay' \
  -H 'accept: */*'
```

4.1.3.16 对数据源恢复 relay-log 功能

这是一个异步接口，请求成功的 Status Code 是 200，可通过获取数据源状态接口获取最新的状态。

4.1.3.16.1 请求 URI

```
PATCH /api/v1/sources/{source-name}/resume-relay
```

4.1.3.16.2 使用样例

```
curl -X 'PATCH' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01/resume-relay' \
  -H 'accept: */*'
```

4.1.3.17 更改数据源和 DM-worker 的绑定关系

这是一个异步接口，请求成功的 Status Code 是 200，可通过获取 DM-worker 节点信息接口获取最新的状态。

4.1.3.17.1 请求 URI

```
PATCH /api/v1/sources/{source-name}/transfer
```

4.1.3.17.2 使用样例

```
curl -X 'PATCH' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01/transfer' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "worker_name": "worker-1"
}'
```

4.1.3.18 获取数据源的数据库名列表

该接口是一个同步接口，请求成功会返回对应的列表。

4.1.3.18.1 请求 URI

```
GET /api/v1/sources/{source-name}/schemas
```

4.1.3.18.2 使用样例

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/sources/source-1/schemas' \
  -H 'accept: application/json'
```

```
[
  "db1"
]
```

4.1.3.19 获取数据源的指定数据库的表名列表

该接口是一个同步接口，请求成功会返回对应的列表。

4.1.3.19.1 请求 URI

```
GET /api/v1/sources/{source-name}/schemas/{schema-name}
```

4.1.3.19.2 使用样例

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/sources/source-1/schemas/db1' \
  -H 'accept: application/json'
```

```
[
  "table1"
]
```

4.1.3.20 创建同步任务

这是一个异步接口，请求成功的 Status Code 是 200，可通过获取同步任务状态接口获取最新的状态。

4.1.3.20.1 请求 URI

```
POST /api/v1/tasks
```

4.1.3.20.2 使用样例

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/tasks' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "remove_meta": false,
  "task": {
    "name": "task-1",
    "task_mode": "all",
    "shard_mode": "pessimistic",
    "meta_schema": "dm-meta",
    "enhance_online_schema_change": true,
    "on_duplicate": "overwrite",
    "target_config": {
      "host": "127.0.0.1",
      "port": 3306,
      "user": "root",
      "password": "123456",
      "security": {
        "ssl_ca_content": "",
        "ssl_cert_content": "",
        "ssl_key_content": "",
        "cert_allowed_cn": [
          "string"
        ]
      }
    },
    "binlog_filter_rule": {
      "rule-1": {
        "ignore_event": [
          "all dml"
        ],
        "ignore_sql": [
          "^Drop"
        ]
```

```
    },
    "rule-2": {
      "ignore_event": [
        "all dml"
      ],
      "ignore_sql": [
        "^Drop"
      ]
    },
    "rule-3": {
      "ignore_event": [
        "all dml"
      ],
      "ignore_sql": [
        "^Drop"
      ]
    }
  },
  "table_migrate_rule": [
    {
      "source": {
        "source_name": "source-name",
        "schema": "db-*",
        "table": "tb-*"
      },
      "target": {
        "schema": "db1",
        "table": "tb1"
      },
      "binlog_filter_rule": [
        "rule-1",
        "rule-2",
        "rule-3",
      ]
    }
  ],
  "source_config": {
    "full_migrate_conf": {
      "export_threads": 4,
      "import_threads": 16,
      "data_dir": "./exported_data",
      "consistency": "auto"
    },
    "incr_migrate_conf": {
      "repl_threads": 16,
```

```
      "repl_batch": 100
    },
    "source_conf": [
      {
        "source_name": "mysql-replica-01",
        "binlog_name": "binlog.000001",
        "binlog_pos": 4,
        "binlog_gtid": "03fc0263-28c7-11e7-a653-6c0b84d59f30
            ↪ :1-7041423,05474d3c-28c7-11e7-8352-203db246dd3d:1-170"
      }
    ]
  }
 },
 "source_name_list": [
  "source-1"
 ]
}'
```

```
{
 "name": "task-1",
 "task_mode": "all",
 "shard_mode": "pessimistic",
 "meta_schema": "dm-meta",
 "enhance_online_schema_change": true,
 "on_duplicate": "overwrite",
 "target_config": {
  "host": "127.0.0.1",
  "port": 3306,
  "user": "root",
  "password": "123456",
  "security": {
    "ssl_ca_content": "",
    "ssl_cert_content": "",
    "ssl_key_content": "",
    "cert_allowed_cn": [
      "string"
    ]
  }
 },
 "binlog_filter_rule": {
  "rule-1": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
```

```
      "^Drop"
    ]
  },
  "rule-2": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-3": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  }
},
"table_migrate_rule": [
  {
    "source": {
      "source_name": "source-name",
      "schema": "db-*",
      "table": "tb-*"
    },
    "target": {
      "schema": "db1",
      "table": "tb1"
    },
    "binlog_filter_rule": [
      "rule-1",
      "rule-2",
      "rule-3",
    ]
  }
],
"source_config": {
  "full_migrate_conf": {
    "export_threads": 4,
    "import_threads": 16,
    "data_dir": "./exported_data",
    "consistency": "auto"
  },
```

```
    "incr_migrate_conf": {
      "repl_threads": 16,
      "repl_batch": 100
    },
    "source_conf": [
      {
        "source_name": "mysql-replica-01",
        "binlog_name": "binlog.000001",
        "binlog_pos": 4,
        "binlog_gtid": "03fc0263-28c7-11e7-a653-6c0b84d59f30:1-7041423,05474
            ↪ d3c-28c7-11e7-8352-203db246dd3d:1-170"
      }
    ]
  }
}
```

4.1.3.21 获取同步任务列表

该接口是一个同步接口，请求成功会返回对应的同步任务信息。

4.1.3.21.1 请求 URI

```
GET /api/v1/tasks
```

4.1.3.21.2 使用样例

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks' \
  -H 'accept: application/json'
```

```
{
  "total": 2,
  "data": [
    {
      "name": "task-1",
      "task_mode": "all",
      "shard_mode": "pessimistic",
      "meta_schema": "dm-meta",
      "enhance_online_schema_change": true,
      "on_duplicate": "overwrite",
      "target_config": {
        "host": "127.0.0.1",
        "port": 3306,
        "user": "root",
        "password": "123456",
```

```
  "security": {
    "ssl_ca_content": "",
    "ssl_cert_content": "",
    "ssl_key_content": "",
    "cert_allowed_cn": [
      "string"
    ]
  }
},
"binlog_filter_rule": {
  "rule-1": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-2": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-3": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  }
},
"table_migrate_rule": [
  {
    "source": {
      "source_name": "source-name",
      "schema": "db-*",
      "table": "tb-*"
    },
    "target": {
      "schema": "db1",
      "table": "tb1"
```

```
        },
        "binlog_filter_rule": [
          "rule-1",
          "rule-2",
          "rule-3",
        ]
      }
    ],
    "source_config": {
      "full_migrate_conf": {
        "export_threads": 4,
        "import_threads": 16,
        "data_dir": "./exported_data",
        "consistency": "auto"
      },
      "incr_migrate_conf": {
        "repl_threads": 16,
        "repl_batch": 100
      },
      "source_conf": [
        {
          "source_name": "mysql-replica-01",
          "binlog_name": "binlog.000001",
          "binlog_pos": 4,
          "binlog_gtid": "03fc0263-28c7-11e7-a653-6c0b84d59f30
              ↪ :1-7041423,05474d3c-28c7-11e7-8352-203db246dd3d:1-170"
        }
      ]
    }
  }
]
}
```

4.1.3.22 停止同步任务

这是一个异步接口，请求成功的 Status Code 是 204，可通过获取同步任务状态接口获取最新的状态。

4.1.3.22.1 请求 URI

```
DELETE /api/v1/tasks/{task-name}
```

4.1.3.22.2 使用样例

```
curl -X 'DELETE' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1' \
  -H 'accept: */*'
```

4.1.3.23 获取同步任务状态

该接口是一个同步接口，请求成功会返回对应节点的状态信息。

4.1.3.23.1 请求 URI

```
GET /api/v1/tasks/task-1/status
```

4.1.3.23.2 使用样例

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/status?stage=running' \
  -H 'accept: application/json'
```

```
{
  "total": 1,
  "data": [
    {
      "name": "string",
      "source_name": "string",
      "worker_name": "string",
      "stage": "runing",
      "unit": "sync",
      "unresolved_ddl_lock_id": "string",
      "load_status": {
        "finished_bytes": 0,
        "total_bytes": 0,
        "progress": "string",
        "meta_binlog": "string",
        "meta_binlog_gtid": "string"
      },
      "sync_status": {
        "total_events": 0,
        "total_tps": 0,
        "recent_tps": 0,
        "master_binlog": "string",
        "master_binlog_gtid": "string",
        "syncer_binlog": "string",
        "syncer_binlog_gtid": "string",
        "blocking_ddls": [
```

```
        "string"
      ],
      "unresolved_groups": [
        {
          "target": "string",
          "ddl_list": [
            "string"
          ],
          "first_location": "string",
          "synced": [
            "string"
          ],
          "unsynced": [
            "string"
          ]
        }
      ],
      "synced": true,
      "binlog_type": "string",
      "seconds_behind_master": 0
    }
  }
 ]
}
```

4.1.3.24 暂停同步任务

这是一个异步接口，请求成功的 Status Code 是 200，可通过获取同步任务状态接口获取最新的状态。

4.1.3.24.1 请求 URI

```
PATCH /api/v1/tasks/task-1/pause
```

4.1.3.24.2 使用样例

```
curl -X 'PATCH' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/pause' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '[
  "source-1"
]'
```

4.1.3.25 恢复同步任务

这是一个异步接口，请求成功的 Status Code 是 200，可通过获取同步任务状态接口获取最新的状态。

4.1.3.25.1 请求 URI

```
PATCH /api/v1/tasks/task-1/resume
```

4.1.3.25.2 使用样例

```
curl -X 'PATCH' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/resume' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '[
  "source-1"
]'
```

4.1.3.26 获取同步任务关联数据源的数据库名列表

该接口是一个同步接口，请求成功会返回对应的列表。

4.1.3.26.1 请求 URI

```
GET /api/v1/tasks/{task-name}/sources/{source-name}/schemas
```

4.1.3.26.2 使用样例

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/sources/source-1/schemas' \
  -H 'accept: application/json'
```

```
[
  "db1"
]
```

4.1.3.27 获取同步任务关联数据源的数据表名列表

该接口是一个同步接口，请求成功会返回对应的列表。

4.1.3.27.1 请求 URI

```
GET /api/v1/tasks/{task-name}/sources/{source-name}/schemas/{schema-name
↪ }
```

4.1.3.27.2 使用样例

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/sources/source-1/schemas/db1' \
  -H 'accept: application/json'
```

```
[
  "table1"
]
```

4.1.3.28 获取同步任务关联数据源的数据表的创建语句

该接口是一个同步接口，请求成功会返回对应的创建语句。

4.1.3.28.1 请求 URI

```
GET /api/v1/tasks/{task-name}/sources/{source-name}/schemas/{schema-name
↪ }/{table-name}
```

4.1.3.28.2 使用样例

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/sources/source-1/schemas/db1/
     ↪ table1' \
  -H 'accept: application/json'
```

```
{
  "schema_name": "db1",
  "table_name": "table1",
  "schema_create_sql": "CREATE TABLE `t1` (`id` int(11) NOT NULL
     ↪ AUTO_INCREMENT,PRIMARY KEY (`id`) /*T![clustered_index] CLUSTERED
     ↪ */) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin"
}
```

4.1.3.29 更新同步任务关联数据源的数据表的创建语句

该接口是一个同步接口，返回体的 Status Code 是 200。

4.1.3.29.1 请求 URI

```
PATCH /api/v1/tasks/{task-name}/sources/{source-name}/schemas/{schema-
↪ name}/{table-name}
```

4.1.3.29.2 使用样例

```
curl -X 'PUT' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/sources/task-1/schemas/db1/
     ↪ table1' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "sql_content": "CREATE TABLE `t1` ( `c1` int(11) DEFAULT NULL, `c2` int
     ↪ (11) DEFAULT NULL, `c3` int(11) DEFAULT NULL) ENGINE=InnoDB DEFAULT
     ↪ CHARSET=utf8mb4 COLLATE=utf8mb4_bin;",
  "flush": true,
  "sync": true
}'
```

4.1.3.30 删除同步任务关联数据源的数据表

该接口是一个同步接口，返回体的 Status Code 是 200。

4.1.3.30.1 请求 URI

```
DELETE /api/v1/tasks/{task-name}/sources/{source-name}/schemas/{schema-
↪ name}/{table-name}
```

4.1.3.30.2 使用样例

```
curl -X 'DELETE' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/sources/source-1/schemas/db1/
     ↪ table1' \
  -H 'accept: */*'
```

4.2 升级版本

4.2.1 TiDB Data Migration 1.0.x 到 2.0+ 手动升级

本文档主要介绍如何手动从 DM v1.0.x 升级到 v2.0+，主要思路为利用 v1.0.x 时的全局 checkpoint 信息在 v2.0+ 集群中启动一个新的增量数据复制任务。

注意：

- DM 当前不支持在数据迁移任务处于全量导出或全量导入过程中从 v1.0.x 升级到 v2.0+。

- 由于 DM 各组件间用于交互的 gRPC 协议进行了较大变更，因此需确保升级前后 DM 集群各组件（包括 dmctl）使用相同的版本。
- 由于 DM 集群的元数据存储（如 checkpoint、shard DDL lock 状态及 online DDL 元信息等）发生了较大变更，升级到 v2.0+ 后无法自动复用 v1.0.x 的元数据，因此在执行升级操作前需要确保：
  - 所有数据迁移任务不处于 shard DDL 协调过程中。
  - 所有数据迁移任务不处于 online DDL 协调过程中。

下面是手动升级的具体步骤。

4.2.1.1 第 1 步：准备 v2.0+ 的配置文件

准备的 v2.0+ 的配置文件包括上游数据库的配置文件以及数据迁移任务的配置文件。

4.2.1.1.1 上游数据库配置文件

在 v2.0+ 中将上游数据库 source 相关的配置从 DM-worker 的进程配置中独立了出来，因此需要根据 v1.0.x 的 DM-worker 配置拆分得到 source 配置。

> **注意：**
>
> 当前从 v1.0.x 升级到 v2.0+ 时，如在 source 配置中启用了 `enable-gtid`，则后续需要通过解析 binlog 或 relay log 文件获取 binlog position 对应的 GTID sets。

从 DM-Ansible 部署的 v1.0.x 升级

如果 v1.0.x 是使用 DM-Ansible 部署的，且假设在 `inventory.ini` 中有如下 `dm_worker_servers` 配置：

```
[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
```

则可以转换得到如下两个 source 配置文件：

```
### 原 dm_worker1 对应的 source 配置，如命名为 source1.yaml
server-id: 101                                  # 对应原 `server_id`
source-id: "mysql-replica-01"                   # 对应原 `source_id`
from:
  host: "172.16.10.81"                          # 对应原 `mysql_host`
  port: 3306                                    # 对应原 `mysql_port`
  user: "root"                                  # 对应原 `mysql_user`
  password: "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=" # 对应原 `mysql_password`
```

```
### 原 dm_worker2 对应的 source 配置，如命名为 source2.yaml
server-id: 102                                  # 对应原 `server_id`
source-id: "mysql-replica-02"                   # 对应原 `source_id`
from:
  host: "172.16.10.82"                          # 对应原 `mysql_host`
  port: 3306                                    # 对应原 `mysql_port`
  user: "root"                                  # 对应原 `mysql_user`
  password: "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=" # 对应原 `mysql_password`
```

从 Binary 部署的 v1.0.x 升级

如果 v1.0.x 是使用 Binary 部署的，且对应的 DM-worker 配置如下：

```
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
```

则可转换得到如下的一个 source 配置文件：

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

4.2.1.1.2 数据迁移任务配置文件

对于数据迁移任务配置向导，v2.0+ 基本与 v1.0.x 保持兼容，可直接复制 v1.0.x 的配置。

4.2.1.2 第 2 步：部署 v2.0+ 集群

> **注意：**
> 如果已有其他可用的 v2.0+ 集群，可跳过此步。

使用 TiUP 按所需要节点数部署新的 v2.0+ 集群。

4.2.1.3 第 3 步：下线 v1.0.x 集群

如果原 v1.0.x 集群是使用 DM-Ansible 部署的，则使用 DM-Ansible 下线 v1.0.x 集群。

如果原 v1.0.x 集群是使用 Binary 部署，则直接停止 DM-worker 与 DM-master 进程。

4.2.1.4 第 4 步：升级数据迁移任务

1. 使用`operate-source` 命令将准备 v2.0+ 的配置文件 中得到的上游数据库 source 配置加载到 v2.0+ 集群中。
2. 在下游 TiDB 中，从 v1.0.x 的数据复制任务对应的增量 checkpoint 表中获取对应的全局 checkpoint 信息。
   - 假设 v1.0.x 的数据迁移配置中未额外指定 `meta-schema`（或指定其值为默认的`dm_meta`），且对应的任务名为 `task_v1`，则对应的 checkpoint 信息在下游 TiDB 的 `` `dm_meta`.`task_v1_syncer_checkpoint` `` 表中。
   - 使用以下 SQL 语句分别获取该数据迁移任务对应的所有上游数据库 source 的全局 checkpoint 信息。

```
> 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. 更新 v1.0.x 的数据迁移任务配置文件以启动新的 v2.0+ 数据迁移任务。

- 如 v1.0.x 的数据迁移任务配置文件为 task_v1.yaml，则将其复制一份为 task_v2.yaml。
- 对 task_v2.yaml 进行以下修改：
  - 将 name 修改为一个新的、不存在的名称，如 task_v2
  - 将 task-mode 修改为 incremental
  - 根据 step.2 中获取的全局 checkpoint 信息，为各 source 设置增量复制的起始点，如：

```
mysql-instances:
  - source-id: "mysql-replica-01"    # 对应 checkpoint 信息所属的
    ↪  `id`
    meta:
      binlog-name: "mysql-bin.000123" # 对应 checkpoint 信息中的
        ↪  `binlog_name`，但不包含 `|000001` 部分
      binlog-pos: 15847              # 对应 checkpoint 信息中的 `
        ↪ binlog_pos`

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

> 注意：
> 如在 source 配置中启动了 enable-gtid，当前需要通过解析 binlog 或 relay log 文件获取 binlog position 对应的 GTID sets 并在 meta 中设置为 binlog-gtid。

4. 使用start-task 命令以 v2.0+ 的数据迁移任务配置文件启动升级后的数据迁移任务。
5. 使用query-status 命令确认数据迁移任务是否运行正常。

如果数据迁移任务运行正常，则表明 DM 升级到 v2.0+ 的操作成功。

4.3 管理上游数据源配置

本文介绍了如何使用dmctl 组件来管理数据源配置，包括如何加密数据库密码，数据源操作，查看数据源配置，改变数据源与 DM-worker 的绑定关系。

4.3.1 加密数据库密码

在 DM 相关配置文件中，推荐使用经 dmctl 加密后的密码。对于同一个原始密码，每次加密后密码不同。

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

```
MKxn0Qo3m3XOyjCnhEMtsUCm83EhGQDZ/T4=
```

4.3.2 数据源操作

operate-source 命令向 DM 集群加载、列出、移除数据源。

```
help operate-source
```

```
`create`/`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 参数解释

- create：创建一个或多个上游的数据库源。创建多个数据源失败时，会尝试回滚到执行命令之前的状态
- stop：停止一个或多个上游的数据库源。停止多个数据源失败时，可能有部分数据源已成功停止
- show：显示已添加的数据源以及对应的 DM-worker
- config-file：
  - 指定 source.yaml 的文件路径
  - 可传递多个文件路径
- --print-sample-config：打印示例配置文件。该参数会忽视其余参数

4.3.2.2 命令用法示例

使用 operate-source 命令创建数据源配置：

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

其中 source.yaml 的配置参考上游数据库配置文件介绍。

结果如下：

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

4.3.3 查看数据源配置

> **注意：**
>
> get-config 命令仅在 DM v2.0.1 及其以后版本支持。

如果知道 source-id，可以通过 dmctl --master-addr <master-addr> get-config source <source-id> 命令直接查看数据源配置。

```
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: '******'
}
```

如果不知道 source-id，可以先通过 `dmctl --master-addr <master-addr> operate-source show` 查看源数据库列表。

```
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"
        }
    ]
}
```

4.3.4 改变数据源与 DM-worker 的绑定关系

`transfer-source` 用于改变数据源与 DM-worker 的绑定关系。

```
help transfer-source
```

```
Transfers a 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.
```

在改变绑定关系前，DM 会检查待解绑的 worker 是否正在运行同步任务，如果正在运行则需要先暂停任务，并在改变绑定关系后恢复任务。

4.3.4.1 命令用法示例

如果不清楚 DM-worker 的绑定关系，可以通过 `dmctl --master-addr <master-addr > list-member --worker` 查看。

```
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": ""
                    }
                ]
            }
        }
    ]
}
```

在本示例中 `mysql-replica-01` 绑定到了 `dm-worker-1` 上。使用如下命令可以将该数据源绑定到 `dm-worker-2` 上

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

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

再次通过 `dmctl --master-addr <master-addr> list-member --worker` 查看，检查命令已生效。

```
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 管理迁移任务

4.4.1 数据迁移任务配置向导

本文档介绍如何配置 Data Migration (DM) 的数据迁移任务。

4.4.1.1 配置需要迁移的数据源

配置需要迁移的数据源之前，首先应该确认已经在 DM 创建相应数据源：

- 查看数据源可以参考查看数据源配置
- 创建数据源可以参考在 DM 创建数据源
- 数据源配置可以参考数据源配置文件介绍

仿照下面的 mysql-instances: 示例定义数据迁移任务需要同步的单个或者多个数据源。

```
---

#### ********* 任务信息配置 *********
name: test           # 任务名称，需要全局唯一

#### ******** 数据源配置 **********
mysql-instances:
  - source-id: "mysql-replica-01" # 从 source-id = mysql-replica-01
    ↪ 的数据源迁移数据
  - source-id: "mysql-replica-02" # 从 source-id = mysql-replica-02
    ↪ 的数据源迁移数据
```

4.4.1.2 配置迁移的目标 TiDB 集群

仿照下面的 target-database: 示例定义迁移的目标 TiDB 集群。

```
---

#### ********* 任务信息配置 *********
name: test           # 任务名称，需要全局唯一

#### ******** 数据源配置 **********
mysql-instances:
  - source-id: "mysql-replica-01" # 从 source-id = mysql-replica-01
    ↪ 的数据源迁移数据
  - source-id: "mysql-replica-02" # 从 source-id = mysql-replica-02
    ↪ 的数据源迁移数据

#### ******** 目标 TiDB 配置 **********
target-database:     # 目标 TiDB 配置
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: ""       # 如果密码不为空，则推荐使用经过 dmctl 加密的密文
```

4.4.1.3 配置需要迁移的表

如果不需要过滤或迁移特定表，可以跳过该项配置。

配置从数据源迁移表的黑白名单，则需要添加两个定义，详细配置规则参考Block & Allow Lists：

1. 定义全局的黑白名单规则

```
block-allow-list:
  bw-rule-1:                          # 规则名称
    do-dbs: ["test.*", "user"]        # 迁移哪些库，支持通配符 "*" 和 "?"，
       ↪ do-dbs 和 ignore-dbs 只需要配置一个，如果两者同时配置只有 do-
       ↪ dbs 会生效
    # ignore-dbs: ["mysql", "account"] # 忽略哪些库，支持通配符 "*" 和
       ↪ "?"
    do-tables:                        # 迁移哪些表，do-tables 和 ignore-
       ↪ tables 只需要配置一个，如果两者同时配置只有 do-tables 会生效
    - db-name: "test.*"
      tbl-name: "t.*"
    - db-name: "user"
      tbl-name: "information"
  bw-rule-2:                          # 规则名称
    ignore-tables:                    # 忽略哪些表
    - db-name: "user"
      tbl-name: "log"
```

2. 在数据源配置中引用黑白名单规则，过滤该数据源需要迁移的表

```
mysql-instances:
    - source-id: "mysql-replica-01" # 从 source-id = mysql-replica-01
       ↪ 的数据源迁移数据
      block-allow-list: "bw-rule-1" # 黑白名单配置名称，如果 DM 版本早于
         ↪  v2.0.0-beta.2 则使用 black-white-list
    - source-id: "mysql-replica-02" # 从 source-id = mysql-replica-02
       ↪ 的数据源迁移数据
      block-allow-list: "bw-rule-2" # 黑白名单配置名称，如果 DM 版本早于
         ↪  v2.0.0-beta.2 则使用 black-white-list
```

4.4.1.4 配置需要过滤的操作

如果不需要过滤特定库或者特定表的特定操作，可以跳过该项配置。

配置过滤特定操作，则需要添加两个定义，详细配置规则参考Binlog Event Filter：

1. 定义全局的数据源操作过滤规则

```
filters:                                # 定义过滤数据源特定操作的规则
   ↪ ，可以定义多个规则
  filter-rule-1:                        # 规则名称
    schema-pattern: "test_*"            # 匹配数据源的库名，支持通配符
       ↪  "*" 和 "?"
    table-pattern: "t_*"                # 匹配数据源的表名，支持通配符
       ↪  "*" 和 "?"
```

```
    events: ["truncate table", "drop table"] # 匹配上 schema-pattern 和
      ↪ table-pattern 的库或者表的操作类型
    action: Ignore                          # 迁移(Do)还是忽略(Ignore)
  filter-rule-2:
    schema-pattern: "test"
    events: ["all dml"]
    action: Do
```

2. 在数据源配置中引用数据源操作过滤规则，过滤该数据源的指定库或表的指定操作

```
mysql-instances:
  - source-id: "mysql-replica-01" # 从 source-id = mysql-replica-01
      ↪ 的数据源迁移数据
    block-allow-list: "bw-rule-1" # 黑白名单配置名称，如果 DM 版本早于
      ↪ v2.0.0-beta.2 则使用 black-white-list
    filter-rules: ["filter-rule-1"] # 过滤数据源特定操作的规则，
      ↪ 可以配置多个过滤规则
  - source-id: "mysql-replica-02" # 从 source-id = mysql-replica-02
      ↪ 的数据源迁移数据
    block-allow-list: "bw-rule-2" # 黑白名单配置名称，如果 DM 版本早于
      ↪ v2.0.0-beta.2 则使用 black-white-list
    filter-rules: ["filter-rule-2"] # 过滤数据源特定操作的规则，
      ↪ 可以配置多个过滤规则
```

4.4.1.5 配置需要数据源表到目标 TiDB 表的映射

如果不需要将数据源表路由到不同名的目标 TiDB 表，可以跳过该项配置。分库分表合并迁移的场景必须配置该规则。

配置数据源表迁移到目标 TiDB 表的路由规则，则需要添加两个定义，详细配置规则参考Table Routing：

1. 定义全局的路由规则

```
routes:                             # 定义数据源表迁移到目标 TiDB 表的路由规则
  ↪ ，可以定义多个规则
  route-rule-1:                     # 规则名称
    schema-pattern: "test_*"        # 匹配数据源的库名，支持通配符 "*" 和 "?"
    table-pattern: "t_*"            # 匹配数据源的表名，支持通配符 "*" 和 "?"
    target-schema: "test"           # 目标 TiDB 库名
    target-table: "t"               # 目标 TiDB 表名
  route-rule-2:
    schema-pattern: "test_*"
    target-schema: "test"
```

2. 在数据源配置中引用路由规则，过滤该数据源需要迁移的表

```
mysql-instances:
  - source-id: "mysql-replica-01"                # 从 source-id = mysql-
    ↪ replica-01 的数据源迁移数据
    block-allow-list: "bw-rule-1"                # 黑白名单配置名称，如果
       ↪  DM 版本早于 v2.0.0-beta.2 则使用 black-white-list
    filter-rules: ["filter-rule-1"]              #
       ↪ 过滤数据源特定操作的规则，可以配置多个过滤规则
    route-rules: ["route-rule-1", "route-rule-2"] # 数据源表迁移到目标
       ↪ TiDB 表的路由规则，可以定义多个规则
  - source-id: "mysql-replica-02"                # 从 source-id = mysql-
    ↪ replica-02 的数据源迁移数据
    block-allow-list: "bw-rule-2"                # 黑白名单配置名称，如果
       ↪  DM 版本早于 v2.0.0-beta.2 则使用 black-white-list
    filter-rules: ["filter-rule-2"]              #
       ↪ 过滤数据源特定操作的规则，可以配置多个过滤规则
```

4.4.1.6 配置是否进行分库分表合并

如果是分库分表合并的数据迁移场景，并且需要同步分库分表的 DDL，则必须显式配置 `shard-mode`，否则不要配置该选项。

分库分表 DDL 同步问题特别多，请确认了解 DM 同步分库分表 DDL 的原理和限制后，谨慎使用。

```
---

#### ********* 任务信息配置 *********
name: test                    # 任务名称，需要全局唯一
shard-mode: "pessimistic"     # 默认值为 "" 即无需协调。如果为分库分表合并任务
   ↪ ，请设置为悲观协调模式 "pessimistic"。
   ↪ 在深入了解乐观协调模式的原理和使用限制后，也可以设置为乐观协调模式 "
   ↪ optimistic"
```

4.4.1.7 其他配置

下面是本数据迁移任务配置向导的完整示例。完整的任务配置参见 DM 任务完整配置文件介绍，其他各配置项的功能和配置也可参阅数据迁移功能。

```
---

#### ********* 任务信息配置 *********
name: test                    # 任务名称，需要全局唯一
shard-mode: "pessimistic"     # 默认值为 "" 即无需协调。如果为分库分表合并任务
   ↪ ，请设置为悲观协调模式 "pessimistic"。
```

```
    ↪ 在深入了解乐观协调模式的原理和使用限制后，也可以设置为乐观协调模式 "
    ↪ optimistic"
task-mode: all              # 任务模式，可设为 "full" - "只进行全量数据迁移
    ↪ "、"incremental" - "Binlog 实时同步"、"all" - "全量 + Binlog 迁移"

#### ******** 数据源配置 **********
mysql-instances:
  - source-id: "mysql-replica-01"            # 从 source-id = mysql-replica
      ↪ -01 的数据源迁移数据
    block-allow-list: "bw-rule-1"            # 黑白名单配置名称，如果 DM
        ↪ 版本早于 v2.0.0-beta.2 则使用 black-white-list
    filter-rules: ["filter-rule-1"]          # 过滤数据源特定操作的规则，
        ↪ 可以配置多个过滤规则
    route-rules: ["route-rule-1", "route-rule-2"] # 数据源表迁移到目标 TiDB
        ↪ 表的路由规则，可以定义多个规则
  - source-id: "mysql-replica-02"            # 从 source-id = mysql-replica
      ↪ -02 的数据源迁移数据
    block-allow-list: "bw-rule-2"            # 黑白名单配置名称，如果 DM
        ↪ 版本早于 v2.0.0-beta.2 则使用 black-white-list
    filter-rules: ["filter-rule-2"]          # 过滤数据源特定操作的规则，
        ↪ 可以配置多个过滤规则
    route-rules: ["route-rule-2"]            # 数据源表迁移到目标 TiDB
        ↪ 表的路由规则，可以定义多个规则

#### ******** 目标 TiDB 配置 **********
target-database:    # 目标 TiDB 配置
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: ""      # 如果密码不为空，则推荐使用经过 dmctl 加密的密文

#### ******** 功能配置 **********
block-allow-list:                   # 定义数据源迁移表的过滤规则，
    ↪ 可以定义多个规则。如果 DM 版本早于 v2.0.0-beta.2 则使用 black-white-
    ↪ list
  bw-rule-1:                        # 规则名称
    do-dbs: ["test.*", "user"]      # 迁移哪些库，支持通配符 "*" 和 "?"，do-
        ↪ dbs 和 ignore-dbs 只需要配置一个，如果两者同时配置只有 do-dbs
        ↪ 会生效
    # ignore-dbs: ["mysql", "account"] # 忽略哪些库，支持通配符 "*" 和 "?"
    do-tables:                      # 迁移哪些表，do-tables 和 ignore-tables
        ↪ 只需要配置一个，如果两者同时配置只有 do-tables 会生效
    - db-name: "test.*"
      tbl-name: "t.*"
    - db-name: "user"
```

```
    tbl-name: "information"
  bw-rule-2:                    # 规则名称
    ignore-tables:              # 忽略哪些表
    - db-name: "user"
      tbl-name: "log"

filters:                                    # 定义过滤数据源特定操作的规则，
    ↪ 可以定义多个规则
  filter-rule-1:                            # 规则名称
    schema-pattern: "test_*"                # 匹配数据源的库名，支持通配符 "*"
       ↪  和 "?"
    table-pattern: "t_*"                    # 匹配数据源的表名，支持通配符 "*"
       ↪  和 "?"
    events: ["truncate table", "drop table"] # 匹配上 schema-pattern 和
       ↪ table-pattern 的库或者表的操作类型
    action: Ignore                          # 迁移 ( Do ) 还是忽略(Ignore)
  filter-rule-2:
    schema-pattern: "test"
    events: ["all dml"]
    action: Do

routes:                         # 定义数据源表迁移到目标 TiDB 表的路由规则，
    ↪ 可以定义多个规则
  route-rule-1:                 # 规则名称
    schema-pattern: "test_*"    # 匹配数据源的库名，支持通配符 "*" 和 "?"
    table-pattern: "t_*"        # 匹配数据源的表名，支持通配符 "*" 和 "?"
    target-schema: "test"       # 目标 TiDB 库名
    target-table: "t"           # 目标 TiDB 表名
  route-rule-2:
    schema-pattern: "test_*"
    target-schema: "test"
```

4.4.2 上游 MySQL 实例配置前置检查

本文介绍了 DM 提供的前置检查功能，此功能用于在数据迁移任务启动时提前检测出上游 MySQL 实例配置中可能存在的一些错误。

4.4.2.1 使用命令

`check-task` 命令用于对上游 MySQL 实例配置是否满足 DM 要求进行前置检查。

4.4.2.2 检查内容

上下游数据库用户必须具备相应读写权限。当数据迁移任务启动时，DM 会自动检查下列权限和配置：

- 数据库版本
  - MySQL 版本 > 5.5
  - MariaDB 版本 >= 10.1.2

> **警告：**
> 支持从 MySQL v8.0 迁移数据是 DM v2.0 的实验特性，不建议在生产环境下使用。

- 数据库配置
  - 是否设置 `server_id`
- MySQL binlog 配置
  - binlog 是否开启（DM 要求 binlog 必须开启）
  - 是否有 `binlog_format=ROW`（DM 只支持 ROW 格式的 binlog 迁移）
  - 是否有 `binlog_row_image=FULL`（DM 只支持 `binlog_row_image=FULL`）
- 上游 MySQL 实例用户的权限

  DM 配置中的 MySQL 用户至少需要具备以下权限：
  - REPLICATION SLAVE
  - REPLICATION CLIENT
  - RELOAD
  - SELECT
- 上游 MySQL 表结构的兼容性

  TiDB 和 MySQL 的兼容性存在以下一些区别：
  - TiDB 不支持外键
  - 字符集的兼容性不同，详见 TiDB 支持的字符集

  DM 还会检查上游表中是否存在主键或唯一键约束，在 v1.0.7 版本引入。
- 上游 MySQL 多实例分库分表的一致性
  - 所有分表的表结构是否一致，检查内容包括：
    * Column 数量
    * Column 名称
    * Column 位置
    * Column 类型
    * 主键

    * 唯一索引
  - 分表中自增主键冲突检查
    * 在两种情况下会造成检查失败：
      · 分表存在自增主键，且自增主键 column 类型不为 bigint
      · 分表存在自增主键，自增主键 column 类型为 bigint，但没有为其配置列值转换
    * 其他情况下检查将成功

4.4.2.2.1 关闭检查项

DM 会根据任务类型进行相应检查，用户可以在任务配置文件中使用 `ignore-checking-items` 配置关闭检查。`ignore-checking-items` 是一个列表，其中可能的取值包括：

取值	含义
all	关闭所有检查
dump_privilege	关闭检查上游 MySQL 实例用户的 dump 相关权限
replication_privilege	关闭检查上游 MySQL 实例用户的 replication 相关权限
version	关闭检查上游数据库版本
server_id	关闭检查上游数据库是否设置 server_id
binlog_enable	关闭检查上游数据库是否已启用 binlog
binlog_format	关闭检查上游数据库 binlog 格式是否为 ROW
binlog_row_image	关闭检查上游数据库 binlog_row_image 是否为 FULL
table_schema	关闭检查上游 MySQL 表结构的兼容性
schema_of_shard_tables	关闭检查上游 MySQL 多实例分库分表的表结构一致性
auto_increment_ID	关闭检查上游 MySQL 多实例分库分表的自增主键冲突

4.4.3 创建数据迁移任务

`start-task` 命令用于创建数据迁移任务。当数据迁移任务启动时，DM 将自动对相应权限和配置进行前置检查。

```
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 命令用法示例

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

4.4.3.2 参数解释

- `-s`:
  - 可选
  - 指定在特定的一个 MySQL 源上执行 `task.yaml`
  - 如果设置，则只启动指定任务在该 MySQL 源上的子任务
- `config-file`:
  - 必选
  - 指定 `task.yaml` 的文件路径
- `remove-meta`:
  - 可选
  - 如果设置，则在启动指定任务时会移除该任务之前存在的 metadata

4.4.3.3 返回结果示例

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

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

4.4.4 TiDB Data Migration 查询状态

本文介绍 TiDB Data Migration (DM) `query-status` 命令的查询结果、任务状态与子任务状态。

4.4.4.1 查询结果

```
» query-status
```

```
{
    "result": true,   # 查询是否成功
    "msg": "",        # 查询失败原因描述
    "tasks": [        # 迁移 task 列表
        {
            "taskName": "test",      # 任务名称
            "taskStatus": "Running", # 任务运行状态
            "sources": [             # 该任务的上游 MySQL 列表
                "mysql-replica-01",
                "mysql-replica-02"
            ]
        },
        {
            "taskName": "test2",
            "taskStatus": "Paused",
            "sources": [
                "mysql-replica-01",
                "mysql-replica-02"
            ]
        }
    ]
}
```

关于 tasks 下的 taskStatus 状态的详细定义，请参阅任务状态。

推荐的 query-status 使用方法是：

1. 首先使用 query-status 查看各个 task 的运行状态是否正常。
2. 如果发现其中某一 task 状态有问题，通过 query-status <出错任务的 taskName> 来得到更详细的错误信息。

4.4.4.2 任务状态

DM 的迁移任务状态取决于其分配到 DM-worker 上的子任务状态，定义见下表：

任务对应的所有子任务的状态	任务状态
任一子任务处于“Paused”状态且返回结果有错误信息	Error - Some error occurred in sub-task
任一处于 Sync 阶段的子任务处于“Running”状态但其 Relay 处理单元未运行 (处于 Error/-Paused/Stopped 状态)	Error - Relay status is Error/-Paused/Stopped

任务对应的所有子任务的状态	任务状态
任一子任务处于“Paused”状态且返回结果没有错误信息	Paused
所有子任务处于“New”状态	New
所有子任务处于“Finished”状态	Finished
所有子任务处于“Stopped”状态	Stopped
其他情况	Running

4.4.4.3 详情查询结果

```
» query-status test
```

```
{
    "result": true,   # 查询是否成功
    "msg": "",        # 查询失败原因描述
    "sources": [      # 上游 MySQL 列表
        {
            "result": true,
            "msg": "",
            "sourceStatus": {               # 上游 MySQL 的信息
                "source": "mysql-replica-01",
                "worker": "worker1",
                "result": null,
                "relayStatus": null
            },
            "subTaskStatus": [          # 上游 MySQL 所有子任务的信息
                {
                    "name": "test",       # 子任务名称
                    "stage": "Running",  # 子任务运行状态，包括 “New”，“
                        ↪ Running”，“Paused”，“Stopped” 以及 “Finished”
                    "unit": "Sync",       # DM 的处理单元，包括 “Check”，“
                        ↪ Dump“，“Load” 以及 “Sync”
                    "result": null,       # 子任务失败时显示错误信息
                    "unresolvedDDLLockID": "test-`test`.`t_target`", #
                        ↪ sharding DDL lock ID，可用于异常情况下手动处理
                        ↪ sharding DDL lock
                    "sync": {               # 当前 `Sync` 处理单元的迁移信息
                        "totalEvents": "12", # 该子任务中迁移的 binlog event
                            ↪ 总数
                        "totalTps": "1",      # 该子任务中每秒迁移的 binlog
                            ↪ event 数量
                        "recentTps": "1",     # 该子任务中最后一秒迁移的 binlog
                            ↪ event 数量
                        "masterBinlog": "(bin.000001, 3234)",
                            ↪                       # 上游数据库当前的 binlog
                            ↪ position
                        "masterBinlogGtid": "c0149e17-dff1-11e8-b6a8-0242
                            ↪ ac110004:1-14", # 上游数据库当前的 GTID 信息
                        "syncerBinlog": "(bin.000001, 2525)",
                            ↪                       # 已被 `Sync`
                            ↪ 处理单元迁移的 binlog position
                        "syncerBinlogGtid": "",
                            ↪                                       # 使用 GTID
                            ↪ 迁移的 binlog position
                        "blockingDDLs": [     # 当前被阻塞的 DDL 列表。
                            ↪ 该项仅在当前 DM-worker 所有上游表都处于 “synced
```

```
                    ↪ “ 状态时才有数值，
                    ↪ 此时该列表包含的是待执行或待跳过的 sharding DDL
                    ↪ 语句
                    "USE `test`; ALTER TABLE `test`.`t_target` DROP
                        ↪ COLUMN `age`;"
                ],
                "unresolvedGroups": [ # 没有被解决的 sharding group
                    ↪ 信息
                    {
                        "target": "`test`.`t_target`",           #
                            ↪ 待迁移的下游表
                        "DDLs": [
                            "USE `test`; ALTER TABLE `test`.`t_target`
                                ↪ DROP COLUMN `age`;"
                        ],
                        "firstPos": "(bin|000001.000001, 3130)",  #
                            ↪ sharding DDL 语句起始 binlog position
                        "synced": [                                 # `
                            ↪ Sync` 处理单元已经读到该 sharding DDL
                            ↪ 的上游分表
                            "`test`.`t2`"
                            "`test`.`t3`"
                            "`test`.`t1`"
                        ],
                        "unsynced": [                               # `
                            ↪ Sync` 处理单元未读到该 sharding DDL
                            ↪ 的上游分表。如有上游分表未完成同步，`
                            ↪ blockingDDLs` 为空
                        ]
                    }
                ],
                "synced": false      # 增量复制是否已追上上游。由于后台
                    ↪ `Sync` 单元并不会实时刷新保存点，当前值为 “false
                    ↪ “ 并不一定代表发生了迁移延迟
            }
        }
    ]
},
{
    "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": {                # `Load` 处理单元的迁移信息
                    "finishedBytes": "115", # 已全量导入字节数
                    "totalBytes": "452", # 总计需要导入的字节数
                    "progress": "25.44 %" # 全量导入进度
                }
            }
        ]
    },
    {
        "result": true,
        "sourceStatus": {
            "source": "mysql-replica-03",
            "worker": "worker3",
            "result": null,
            "relayStatus": null
        },
        "subTaskStatus": [
            {
                "name": "test",
                "stage": "Paused",
                "unit": "Load",
                "result": {              # 错误示例
                    "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 %"
                }
            }
        ]
    }
]
}
```

关于 sources 下 subTaskStatus 中 stage 状态和状态转换关系的详细信息，请参阅子任务状态。

关于 sources 下 subTaskStatus 中 unresolvedDDLLockID的操作细节，请参阅手动处理 Sharding DDL Lock。

4.4.4.4 子任务状态

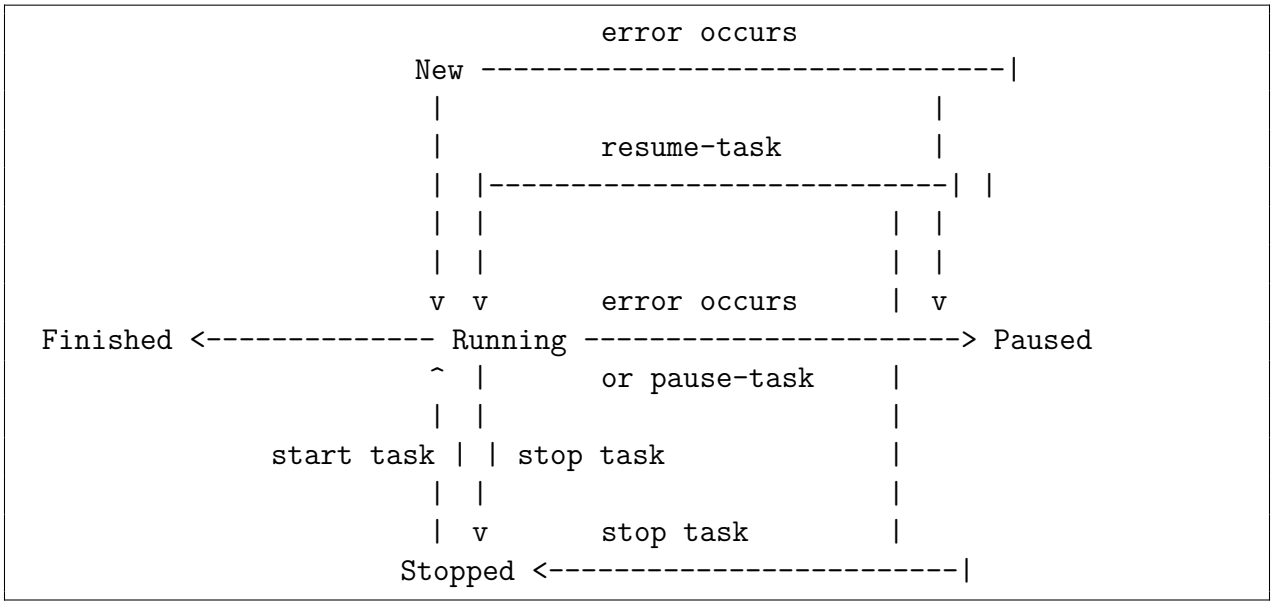
4.4.4.4.1 状态描述

- New:
  - 初始状态。
  - 如果子任务没有发生错误，状态切换为 Running，其他情况则切换为 Paused。
- Running: 正常运行状态。
- Paused:
  - 暂停状态。
  - 子任务发生错误，状态切换为 Paused。
  - 如在子任务为 Running 状态下执行 pause-task 命令,任务状态会切换为 Paused ↪ 。
  - 如子任务处于该状态，可以使用 resume-task 命令恢复任务。
- Stopped:
  - 停止状态。

- 如在子任务为 Running 或 Paused 状态下执行 stop-task 命令，任务状态会切换为 Stopped。
- 如子任务处于该状态，不可使用 resume-task 命令恢复任务。

- Finished:
  - 任务完成状态。
  - 只有 task-mode 为 full 的任务正常完成后，任务才会切换为该状态。

4.4.4.4.2 状态转换图



4.4.5 暂停数据迁移任务

pause-task 命令用于暂停数据迁移任务。

> **注意：**
>
> 有关 pause-task 与 stop-task 的区别如下：
>
> - 使用 pause-task 仅暂停迁移任务的执行，但仍然会在内存中保留任务的状态信息等，且可通过 query-status 进行查询；使用 stop-task 会停止迁移任务的执行，并移除内存中与该任务相关的信息，且不可再通过 query-status 进行查询，但不会移除已经写入到下游数据库中的数据以及其中的 checkpoint 等 dm_meta 信息。
> - 使用 pause-task 暂停迁移任务期间，由于任务本身仍然存在，因此不能再启动同名的新任务，且会阻止对该任务所需 relay log 的清理；使用 stop-task 停止任务后，由于任务不再存在，因此可以再启动同名的新任务，且不会阻止对 relay log 的清理。

- pause-task 一般用于临时暂停迁移任务以排查问题等；stop-task 一般用于永久删除迁移任务或通过与 start-task 配合以更新配置信息。

```
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 命令用法示例

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

4.4.5.2 参数解释

- -s:
  - 可选
  - 指定在特定的一个 MySQL 源上暂停数据迁移任务的子任务
  - 如果设置，则只暂停该任务在指定 MySQL 源上的子任务
- task-name| task-file:
  - 必选
  - 指定任务名称或任务文件路径

4.4.5.3 返回结果示例

```
pause-task test
```

```
{
    "op": "Pause",
    "result": true,
    "msg": "",
    "sources": [
```

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

4.4.6 恢复数据迁移任务

resume-task 命令用于恢复处于 Paused 状态的数据迁移任务，通常用于在人为处理完造成迁移任务暂停的故障后手动恢复迁移任务。

```
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 命令用法示例

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

4.4.6.2 参数解释

- `-s`:
  - 可选
  - 指定在特定的一个 MySQL 源上恢复数据迁移任务的子任务
  - 如果设置，则只恢复该任务在指定 MySQL 源上的子任务
- `task-name | task-file`:
  - 必选
  - 指定任务名称或任务文件路径

4.4.6.3 返回结果示例

```
resume-task test
```

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

4.4.7 停止数据迁移任务

stop-task 命令用于停止数据迁移任务。有关 stop-task 与 pause-task 的区别，请参考暂停数据迁移任务中的相关说明。

```
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 命令用法示例

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

4.4.7.2 参数解释

- -s:

  - 可选
  - 指定在特定的一个 MySQL 源上停止数据迁移任务的子任务
  - 如果设置，则只停止该任务在指定 MySQL 源上的子任务

- `task-name | task-file`:
  - 必选
  - 指定任务名称或任务文件路径

4.4.7.3 返回结果示例

```
stop-task test
```

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

4.4.8 导出和导入集群的数据源和任务配置

`config` 命令用于导出和导入集群的数据源和任务配置。

> **注意：**
>
> 对于 v2.0.5 版本之前的集群，可使用 v2.0.5 版本及之后的 dmctl 导出和导入集群的数据源和任务配置文件。

```
» 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 导出集群的数据源和任务配置

使用 export 子命令导出集群的数据源和任务配置到指定文件夹中。

```
config export [--dir directory]
```

4.4.8.1.1 参数解释

- dir:
  - 可选
  - 指定导出文件夹路径
  - 默认值为 ./configs

4.4.8.1.2 返回结果示例

```
config export -d /tmp/configs
```

```
export configs to directory `/tmp/configs` succeed
```

4.4.8.2 导入集群的数据源和任务配置

使用 import 子命令从指定文件夹中导入集群的数据源和任务配置。

```
config import [--dir directory]
```

> **注意:**
>
> 对于 v2.0.2 版本之后的集群，暂不支持自动导入 relay worker 的相关配置，可以手动使用 start-relay 命令开启 relay log。

4.4.8.2.1 参数解释

- dir:
  - 可选
  - 指定导入文件夹路径
  - 默认值为 ./configs

4.4.8.2.2 返回结果示例

```
config import -d /tmp/configs
```

```
start creating sources
start creating tasks
import configs from directory `/tmp/configs` succeed
```

4.4.9 处理出错的 DDL 语句

本文介绍了如何使用 DM 来处理出错的 DDL 语句。

目前，TiDB 并不完全兼容所有的 MySQL 语法（详见 TiDB 已支持的 DDL 语句）。当使用 DM 从 MySQL 迁移数据到 TiDB 时，如果 TiDB 不支持对应的 DDL 语句，可能会造成错误并中断迁移任务。在这种情况下，DM 提供 handle-error 命令来恢复迁移。

4.4.9.1 使用限制

如果业务不能接受下游 TiDB 跳过异常的 DDL 语句，也不接受使用其他 DDL 语句作为替代，则不适合使用此方式进行处理。

比如：DROP PRIMARY KEY，这种情况下，只能在下游重建一个（DDL 执行完后的）新表结构对应的表，并将原表的全部数据重新导入该新表。

4.4.9.2 支持场景

迁移过程中，上游执行了 TiDB 不支持的 DDL 语句并迁移到了 DM，造成迁移任务中断。

- 如果业务能接受下游 TiDB 不执行该 DDL 语句，则使用 handle-error <task-name> skip 跳过对该 DDL 语句的迁移以恢复迁移任务。
- 如果业务能接受下游 TiDB 执行其他 DDL 语句来作为替代，则使用 handle-error <task-name> replace 替代该 DDL 的迁移以恢复迁移任务。

4.4.9.3 命令介绍

使用 dmctl 手动处理出错的 DDL 语句时，主要使用的命令包括 query-status、handle-error。

4.4.9.3.1 query-status

query-status 命令用于查询当前 MySQL 实例内子任务及 relay 单元等的状态和错误信息，详见查询状态。

4.4.9.3.2 handle-error

handle-error 命令用于处理错误的 DDL 语句。

4.4.9.3.3 命令用法

```
» 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 参数解释

- task-name：
  - 非 flag 参数，string，必选；
  - 指定预设的操作将生效的任务。
- source：
  - flag 参数，string，--source；
  - source 指定预设操作将生效的 MySQL 实例。
- skip：跳过该错误
- replace：替代错误的 DDL 语句
- revert：重置该错误先前的 skip/replace 操作，仅在先前的操作没有最终生效前执行
- binlog-pos：

- flag 参数，string，--binlog-pos；
- 若不指定，DM 会自动处理当前出错的 DDL 语句
- 在指定时表示操作将在 binlog-pos 与 binlog event 的 position 匹配时生效，格式为 binlog-filename:binlog-pos，如 mysql-bin|000001.000003:3270。
- 在迁移执行出错后，binlog position 可直接从 query-status 返回的 startLocation 中的 position 获得；在迁移执行出错前，binlog position 可在上游 MySQL 中使用 SHOW BINLOG EVENTS 获得。

4.4.9.4 使用示例

4.4.9.4.1 迁移中断执行跳过操作

非合库合表场景

假设现在需要将上游的 db1.tbl1 表迁移到下游 TiDB，初始时表结构为：

```
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 |
+-------+-----------------------------------------------------+
```

此时，上游执行以下 DDL 操作修改表结构（将列的 DECIMAL(11, 3) 修改为 DECIMAL(10, 3)）：

```
ALTER TABLE db1.tbl1 CHANGE c2 c2 DECIMAL (10, 3);
```

则会由于 TiDB 不支持该 DDL 语句而导致 DM 迁移任务中断，使用 query-status <task-name> 命令可看到如下错误：

```
ERROR 8200 (HY000): Unsupported modify column: can't change decimal column
    ↪ precision
```

假设业务上可以接受下游 TiDB 不执行此 DDL 语句（即继续保持原有的表结构），则可以通过使用 handle-error <task-name> skip 命令跳过该 DDL 语句以恢复迁移任务。操作步骤如下：

1. 使用 handle-error <task-name> skip 跳过当前错误的 DDL 语句

```
» handle-error test skip
```

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

2. 使用 query-status <task-name> 查看任务状态

```
» query-status test
```

执行结果

```
{
    "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"
                }
            }
        ]
    }
]
}
```

可以看到任务运行正常，错误的 DDL 被跳过。

合库合表场景

假设现在存在如下四个上游表需要合并迁移到下游的同一个表 `shard_db`.`shard_table`，任务模式为悲观协调模式：

- MySQL 实例 1 内有 shard_db_1 库，包括 shard_table_1 和 shard_table_2 两个表。
- MySQL 实例 2 内有 shard_db_2 库，包括 shard_table_1 和 shard_table_2 两个表。

初始时表结构为：

```
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 |
+-------+------------------------------------------------------------------------------
    ↪
```

此时，在上游所有分表上都执行以下 DDL 操作修改表字符集

```
ALTER TABLE `shard_db_*`.`shard_table_*` CHARACTER SET LATIN1 COLLATE
    ↪ LATIN1_DANISH_CI;
```

则会由于 TiDB 不支持该 DDL 语句而导致 DM 迁移任务中断，使用 query-status ↪ 命令可以看到 MySQL 实例 1 的 shard_db_1.shard_table_1 表和 MySQL 实例 2 的 shard_db_2.shard_table_1 表报错：

```
{
    "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"
}
```

假设业务上可以接受下游 TiDB 不执行此 DDL 语句（即继续保持原有的表结构），则可以通过使用 handle-error <task-name> skip 命令跳过该 DDL 语句以恢复迁移任务。操作步骤如下：

1. 使用 handle-error <task-name> skip 跳过 MySQL 实例 1 和实例 2 当前错误的 DDL 语句

```
» 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. 使用 `query-status <task-name>` 查看任务状态，可以看到 MySQL 实例 1 的 `shard_db_1.shard_table_2` 表和 MySQL 实例 2 的 `shard_db_2.shard_table_2` 表报错：

```
{
    "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. 继续使用 `handle-error <task-name> skip` 跳过 MySQL 实例 1 和实例 2 当前错误的 DDL 语句

```
» 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. 使用 query-status <task-name> 查看任务状态

```
» query-status test
```

执行结果

```
{
    "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"
                }
            }
        ]
    }
]
```

```
}
```

可以看到任务运行正常，无错误信息。四条 DDL 全部被跳过。

4.4.9.4.2 迁移中断执行替代操作

非合库合表场景

假设现在需要将上游的 db1.tbl1 表迁移到下游 TiDB，初始时表结构为：

```
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 |
+-------+-----------------------------------------------------------------------------
    ↪
```

此时，上游执行以下 DDL 操作增加新列，并添加 UNIQUE 约束

```
ALTER TABLE `db1`.`tbl1` ADD COLUMN new_col INT UNIQUE;
```

则会由于 TiDB 不支持该 DDL 语句而导致 DM 迁移任务中断，使用 query-status 命令可看到如下错误：

```
{
    "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'",
}
```

我们将该 DDL 替换成两条等价的 DDL。操作步骤如下：

1. 使用如下命令替换错误的 DDL 语句

```
» 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. 使用 query-status <task-name> 查看任务状态

```
» query-status test
```

执行结果


```
{
    "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"
                }
            }
        ]
    }
]
}
```

可以看到任务运行正常，错误的 DDL 已被替换且执行成功。

合库合表场景

假设现在存在如下四个上游表需要合并迁移到下游的同一个表 `shard_db`.`shard_table`，任务模式为悲观协调模式：

- MySQL 实例 1 内有 shard_db_1 库，包括 shard_table_1 和 shard_table_2 两个表。
- MySQL 实例 2 内有 shard_db_2 库，包括 shard_table_1 和 shard_table_2 两个表。

初始时表结构为：

```
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 |
+-------+------------------------------------------------------------------------------
    ↪
```

此时，在上游所有分表上都执行以下 DDL 操作增加新列，并添加 UNIQUE 约束：

```
ALTER TABLE `shard_db_*`.`shard_table_*` ADD COLUMN new_col INT UNIQUE;
```

则会由于 TiDB 不支持该 DDL 语句而导致 DM 迁移任务中断，使用 query-status ↪ 命令可以看到 MySQL 实例 1 的 shard_db_1.shard_table_1 表和 MySQL 实例 2 的 shard_db_2.shard_table_1 表报错：

```
{
    "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'",
}
```

我们将该 DDL 替换成两条等价的 DDL。操作步骤如下：

1. 使用如下命令分别替换 MySQL 实例 1 和实例 2 中错误的 DDL 语句

```
» 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. 使用 `query-status <task-name>` 查看任务状态，可以看到 MySQL 实例 1 的 `shard_db_1.shard_table_2` 表和 MySQL 实例 2 的 `shard_db_2.shard_table_2` 表报错：

```
{
    "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. 使用如下命令继续分别替换 MySQL 实例 1 和实例 2 中错误的 DDL 语句

```
» 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. 使用 query-status <task-name> 查看任务状态

```
» query-status test
```

执行结果

```
{
    "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"
                }
            }
        ]
    }
]
}
```


可以看到任务运行正常，无错误信息。四条 DDL 全部被替换。

4.5 手动处理 Sharding DDL Lock

DM (Data Migration) 使用 sharding DDL lock 来确保分库分表的 DDL 操作可以正确执行。绝大多数情况下，该锁定机制可自动完成；但在部分异常情况发生时，需要使用 `unlock-ddl-lock` 手动处理异常的 DDL lock。

注意：

- 本文档只适用于悲观协调模式下 sharding DDL lock 的处理。

- 本文档的命令在交互模式中进行，因此在以下命令示例中未添加转义字符。在命令行模式中，你需要添加转义字符，防止报错。
- 不要轻易使用 unlock-ddl-lock 命令，除非完全明确当前场景下使用这些命令可能会造成的影响，并能接受这些影响。
- 在手动处理异常的 DDL lock 前，请确保已经了解 DM 的分库分表合并迁移原理。

4.5.1 命令介绍

4.5.1.1 show-ddl-locks

该命令用于查询当前 DM-master 上存在的 DDL lock 信息。

4.5.1.1.1 命令示例

```
show-ddl-locks [--source=mysql-replica-01] [task-name | task-file]
```

4.5.1.1.2 参数解释

- source:
  - flag 参数，string，--source，可选
  - 不指定时，查询所有 MySQL source 相关的 lock 信息；指定时，仅查询与该 MySQL source 相关的 lock 信息，可重复多次指定
- task-name | task-file:
  - 非 flag 参数，string，可选
  - 不指定时，查询与所有任务相关的 lock 信息；指定时，仅查询特定任务相关的 lock 信息

4.5.1.1.3 返回结果示例

```
show-ddl-locks test
```

```
{
    "result": true,                                     # 查询 lock
        ↪ 操作本身是否成功
    "msg": "",                                          # 查询 lock
        ↪ 操作失败时的原因或其它描述信息（如不存在任务 lock）
    "locks": [                                          # 当前存在的 lock
        ↪ 信息列表
        {
```

```
            "ID": "test-`shard_db`.`shard_table`",   # lock 的 ID 标识，
               ↪ 当前由任务名与 DDL 对应的 schema/table 信息组成
            "task": "test",                          # lock 所属的任务名
            "mode": "pessimistic"                    # shard DDL 协调模式，
               ↪ 可为悲观模式 "pessimistic" 或乐观模式 "optimistic"
            "owner": "mysql-replica-01",             # lock 的 owner（
               ↪ 在悲观模式时为第一个遇到该 DDL 的 source ID），
               ↪ 在乐观模式时总为空
            "DDLs": [                                # 在悲观模式时为 lock
               ↪ 对应的 DDL 列表，在乐观模式时总为空
               "USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` DROP
                  ↪ COLUMN `c2`;"
            ],
            "synced": [                              # 已经收到对应 MySQL
               ↪ 实例内所有分表 DDL 的 source 列表
               "mysql-replica-01"
            ],
            "unsynced": [                            # 尚未收到对应 MySQL
               ↪ 实例内所有分表 DDL 的 source 列表
               "mysql-replica-02"
            ]
        }
    ]
}
```

4.5.1.2 unlock-ddl-lock

该命令用于主动请求 DM-master 解除指定的 DDL lock，包括的操作：请求 owner 执行 DDL 操作，请求其他非 owner 的 DM-worker 跳过 DDL 操作，移除 DM-master 上的 lock 信息。

> **注意：**
>
> unlock-ddl-lock 当前仅对悲观协调模式 (pessimistic) 下产生的 lock 有效。

4.5.1.2.1 命令示例

```
unlock-ddl-lock [--owner] [--force-remove] <lock-ID>
```

4.5.1.2.2 参数解释

- `owner`:
  - flag 参数，string，`--owner`，可选
  - 不指定时，请求默认的 owner（`show-ddl-locks` 返回结果中的 `owner`）执行 DDL 操作；指定时，请求该 MySQL source（替代默认的 owner）执行 DDL 操作
  - 除非原 owner 已经从集群中移除，否则不应该指定新的 owner
- `force-remove`:
  - flag 参数，boolean，`--force-remove`，可选
  - 不指定时，仅在 owner 执行 DDL 成功时移除 lock 信息；指定时，即使 owner 执行 DDL 失败也强制移除 lock 信息（此后将无法再次查询或操作该 lock）
- `lock-ID`:
  - 非 flag 参数，string，必选
  - 指定需要执行 unlock 操作的 DDL lock ID（即 `show-ddl-locks` 返回结果中的 `ID`）

4.5.1.2.3 返回结果示例

```
unlock-ddl-lock test-`shard_db`.`shard_table`
```

```
{
    "result": true,                                  # unlock lock
        ↪ 操作是否成功
    "msg": "",                                       # unlock lock
        ↪ 操作失败时的原因
}
```

4.5.2 支持场景

目前，使用 `unlock-ddl-lock` 命令仅支持处理以下两种 sharding DDL lock 异常情况。

4.5.2.1 场景一：部分 MySQL source 被移除

4.5.2.1.1 Lock 异常原因

在 DM-master 尝试自动 unlock sharding DDL lock 之前，需要等待所有 MySQL source 的 sharding DDL events 全部到达（详见分库分表合并迁移原理）。如果 sharding DDL 已经在迁移过程中，同时有部分 MySQL source 被移除，且不再计划重新加载它们（按业务需求移除了这部分 MySQL source），则会由于永远无法等齐所有的 DDL 而造成 lock 无法自动 unlock。

4.5.2.1.2 手动处理示例

假设上游有 MySQL-1（mysql-replica-01）和 MySQL-2（mysql-replica-02）两个实例，其中 MySQL-1 中有 shard_db_1.shard_table_1 和 shard_db_1.shard_table_2 两个表，MySQL-2 中有 shard_db_2.shard_table_1 和 shard_db_2.shard_table_2 两个表。现在需要将这 4 个表合并后迁移到下游 TiDB 的 shard_db.shard_table 表中。

初始表结构如下：

```
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 |
+--------------+----------------------------------------+
```

上游分表将执行以下 DDL 语句变更表结构：

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

MySQL 及 DM 操作与处理流程如下：

1. mysql-replica-01 对应的两个分表执行了对应的 DDL 操作进行表结构变更。

   ```
   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 接受到 mysql-replica-01 两个分表的 DDL 之后，将对应的 DDL 信息发送给 DM-master，DM-master 创建相应的 DDL lock。

3. 使用 show-ddl-lock 查看当前的 DDL lock 信息。

   ```
   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. 由于业务需要，mysql-replica-02 对应的数据不再需要迁移到下游 TiDB，对 mysql-replica-02 执行了移除操作。

5. DM-master 上 ID 为 test-`shard_db`.`shard_table` 的 lock 无法等到 mysql-replica-02 的 DDL 操作信息。

   show-ddl-locks 返回的 unsynced 中一直包含 mysql-replica-02 的信息。

6. 使用 unlock-dll-lock 来请求 DM-master 主动 unlock 该 DDL lock。

   - 如果 DDL lock 的 owner 也已经被移除，可以使用 --owner 参数指定其他 MySQL source 作为新 owner 来执行 DDL。
   - 当存在任意 MySQL source 报错时，result 将为 false，此时请仔细检查各 MySQL source 的错误是否是预期可接受的。

   ```
   unlock-ddl-lock test-`shard_db`.`shard_table`
   ```

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

7. 使用 show-dd-locks 确认 DDL lock 是否被成功 unlock。

```
show-ddl-locks test
```

```
{
    "result": true,
    "msg": "no DDL lock exists",
    "locks": [
    ]
}
```

8. 查看下游 TiDB 中的表结构是否变更成功。

```
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. 使用 query-status 确认迁移任务是否正常。

4.5.2.1.3 手动处理后的影响

使用 unlock-ddl-lock 手动执行 unlock 操作后，由于该任务的配置信息中仍然包含了已下线的 MySQL source，如果不进行处理，则当下次 sharding DDL 到达时，仍会出现 lock 无法自动完成迁移的情况。

因此，在手动解锁 DDL lock 后，需要再执行以下操作：

1. 使用 stop-task 停止运行中的任务。
2. 更新任务配置文件，将已下线 MySQL source 对应的信息从配置文件中移除。
3. 使用 start-task 及新任务配置文件重新启动任务。

> **注意：**
>
> 在 unlock-ddl-lock 之后，如果已下线的 MySQL source 重新加载并尝试对其中的分表进行数据迁移，则会由于数据与下游的表结构不匹配而发生错误。

4.5.2.2 场景二：unlock 过程中部分 DM-worker 异常停止或网络中断

4.5.2.2.1 Lock 异常原因

在 DM-master 收到所有 DM-worker 的 DDL 信息后，执行自动 unlock DDL lock 的操作主要包括以下步骤：

1. 请求 lock owner 执行 DDL 操作，并更新对应分表的 checkpoint。

2. 在 owner 执行 DDL 操作成功后，移除 DM-master 上保存的 DDL lock 信息。
3. 在 owner 执行 DDL 操作成功后，请求其他所有非 owner 跳过 DDL 操作并更新对应分表的 checkpoint。
4. DM-master 在所有 owner/非 owner 操作成功后，移除对应的 DDL lock 信息。

上述 unlock DDL lock 的操作不是原子的。如果非 owner 跳过 DDL 操作成功后，所在的 DM-worker 异常停止或与下游 TiDB 发生网络异常，造成无法成功更新 checkpoint。

当非 owner 对应的 MySQL source 恢复数据迁移时，会尝试请求 DM-master 重新协调异常发生前已经协调过的 DDL、且永远无法等到其他 MySQL source 的对应 DDL，造成该 DDL 操作对应 lock 的自动 unlock。

4.5.2.2.2 手动处理示例

仍然假设是部分 MySQL source 被移除 示例中的上下游表结构及合表迁移需求。

当在 DM-master 自动执行 unlock 操作的过程中，owner（`mysql-replica-01`）成功执行了 DDL 操作且开始继续进行后续迁移，但在请求非 owner（`mysql-replica-02`）跳过 DDL 操作的过程中，由于对应的 DM-worker 发生了重启在跳过 DDL 后未能更新 checkpoint。

`mysql-replica-02` 对应的数据迁移子任务恢复后，将在 DM-master 上创建一个新的 lock，但其他 MySQL source 此时已经执行或跳过 DDL 操作并在进行后续迁移。

处理流程如下：

1. 使用 `show-ddl-locks` 确认 DM-master 上存在该 DDL 操作对应的 lock。

   应该仅有 `mysql-replica-02` 处于 `synced` 状态：

```
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. 使用 `unlock-ddl-lock` 请求 DM-master unlock 该 lock。

    - Lock 过程中会尝试再次向下游执行该 DDL 操作（重启前的原 owner 已向下游执行过该 DDL 操作），需要确保该 DDL 操作可被多次执行。

```
unlock-ddl-lock test-`shard_db`.`shard_table`
```

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

3. 使用 `show-ddl-locks` 确认 DDL lock 是否被成功 unlock。

4. 使用 `query-status` 确认迁移任务是否正常。

4.5.2.2.3 手动处理后的影响

手动 unlock sharding DDL lock 后，后续的 sharding DDL 将可以自动正常迁移。

4.6 管理迁移表的表结构

本文介绍如何使用dmctl 组件来管理通过 DM 迁移的表在 DM 内部的表结构。

4.6.1 原理介绍

在使用 DM 迁移数据表时，DM 对于表结构主要包含以下相关处理。

对于全量导出与导入，DM 直接导出当前时刻上游的表结构到 SQL 格式的文件中，并将该表结构直接应用到下游。

对于增量复制，在整个数据链路中则包含以下几类可能相同或不同的表结构。

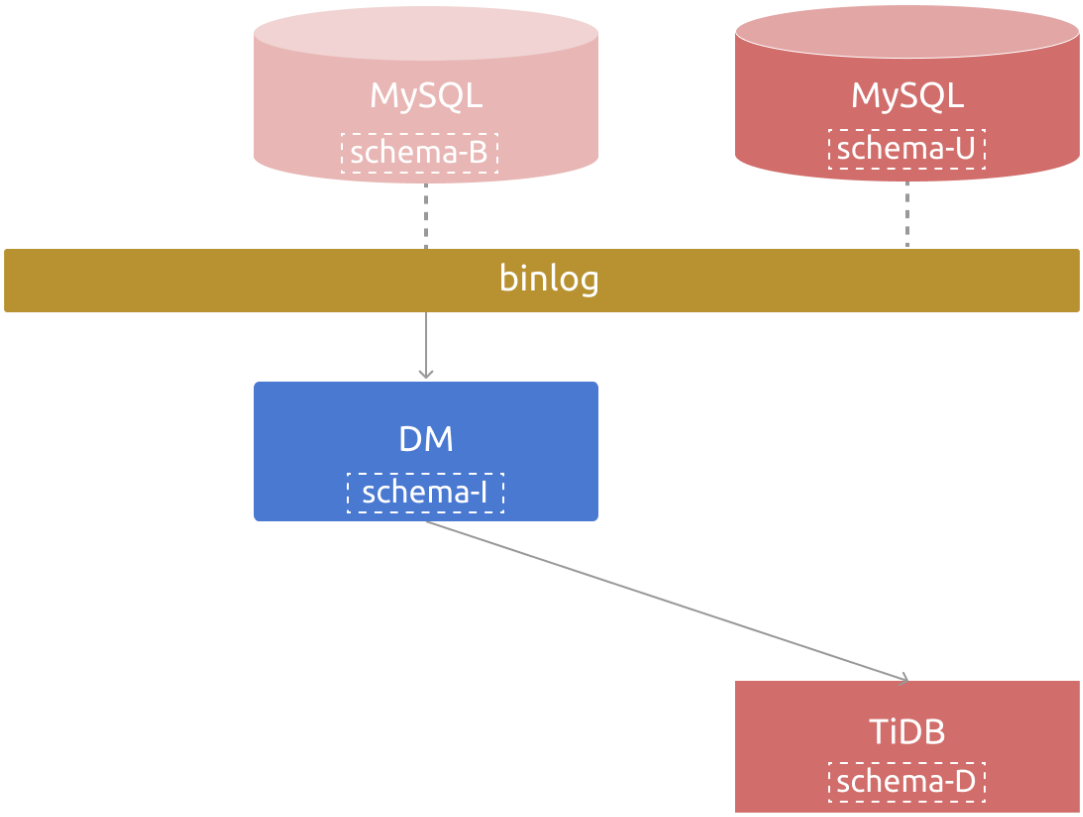


Figure 22: 表结构

- 上游当前时刻的表结构（记为 `schema-U`）。
- 当前 DM 正在消费的 binlog event 的表结构（记为 `schema-B`，其对应于上游某个历史时刻的表结构）。
- DM 内部（schema tracker 组件）当前维护的表结构（记为 `schema-I`）。
- 下游 TiDB 集群中的表结构（记为 `schema-D`）。

在大多数情况下，以上 4 类表结构一致。

当上游执行 DDL 变更表结构后，`schema-U` 即会发生变更；DM 通过将该 DDL 应用于内部的 schema tracker 组件及下游 TiDB，会先后更新 `schema-I`、`schema-D` 以与 `schema-U` 保持一致，因而随后能正常消费 binlog 中在 DDL 之后对应表结构为 `schema-B` 的 binlog event。即当 DDL 被复制成功后，仍能保持 `schema-U`、`schema-B`、`schema-I` 及 `schema-D` 的一致。

但在开启乐观 shard DDL 支持 的数据迁移过程中，下游合并表的 `schema-D` 可能与部分分表对应的 `schema-B` 及 `schema-I` 并不一致，但 DM 仍保持 `schema-I` 与 `schema-B` 的一致以确保能正常解析 DML 对应的 binlog event。

此外，在其他一些场景下（如：下游比上游多部分列），schema-D 也可能会与 schema-B 及 schema-I 并不一致。

为了支持以上的特殊场景及处理其他可能的由于 schema 不匹配导致的迁移中断等问题，DM 提供了 operate-schema 命令来获取、修改、删除 DM 内部维护的表结构 schema-I。

4.6.2 命令介绍

```
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.
```

> 注意：
>
> - 由于表结构在数据迁移过程中可能会发生变更，为获取确定性的表结构，当前 operate-schema 命令仅能在数据迁移任务处于 Paused 状态时可用。
> - 强烈建议在修改表结构前，首先获取并备份表结构，以免误操作导致数据丢失。

4.6.3 参数解释

- operate-type:
  - 必选
  - 指定对于 schema 的操作类型，可以为 get、set 或 remove

- `-s`:
  - 必选
  - 指定操作将应用到的 MySQL 源
- `task-name | task-file`:
  - 必选
  - 指定任务名称或任务文件路径
- `-d`:
  - 必选
  - 指定数据表所属的上游数据库名
- `-t`:
  - 必选
  - 指定数据表对应的上游表名
- `schema-file`:
  - `set` 操作时必选，其他操作不需指定
  - 将被设置的表结构文件，文件内容应为一个合法的 `CREATE TABLE` 语句
- `--flush`:
  - 可选
  - 同时将表结构写入 checkpoint，从而在任务重启后加载到该表结构
  - 默认值为 `true`
- `--sync`:
  - 可选
  - 仅在乐观协调 DDL 出错时使用。使用该表结构更新乐观协调元数据中的表结构

4.6.4 使用示例

4.6.4.1 获取表结构

假设要获取 `db_single` 任务对应于 `mysql-replica-01` MySQL 源的 `` `db_single`.`t1` `` 表的表结构，则执行如下命令：

```
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 设置表结构

假设要设置 db_single 任务对应于 mysql-replica-01 MySQL 源的 `db_single`.`t1` 表的表结构为

```
CREATE TABLE `t1` (
    `c1` int(11) NOT NULL,
    `c2` bigint(11) DEFAULT NULL,
    PRIMARY KEY (`c1`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin
```

则将上述 CREATE TABLE 语句保存为文件（如 db_single.t1-schema.sql）后执行如下命令：

```
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 删除表结构

注意：

删除 DM 内部维护的表结构后，如果后续有该表的 DDL/DML 需要复制到下游，则 DM 会依次尝试从 checkpoint 表里 `table_info` 字段、乐观 shard DDL 协调中的元信息以及下游 TiDB 中对应的该表获取表结构。

假设要删除 `db_single` 任务对应于 `mysql-replica-01` MySQL 源的 `` `db_single`.`t1` `` 表的表结构，则执行如下命令：

```
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 处理告警

本文档介绍 DM 中各主要告警信息的处理方法。

4.7.1 高可用告警

4.7.1.1 DM_master_all_down

当全部 DM-master 离线时触发该告警。发生该错误时，需要检查集群环境，并通过各节点日志排查错误。

4.7.1.2 DM_worker_offline

存在离线的 DM-worker 超过一小时会触发该告警。在高可用架构下，该告警可能不会直接中断任务，但是会提升任务中断的风险。处理告警可以查看对应 DM-worker 节点的工作状态，检查是否连通，并通过日志排查错误。

4.7.1.3 DM_DDL_error

处理 shard DDL 时出现错误，此时需要参考DM 故障诊断进行处理。

4.7.1.4 DM_pending_DDL

存在未完成的 shard DDL 并超过一小时会触发该告警。在某些应用场景下，存在未完成的 shard DDL 可能是用户所期望的。在用户预期以外的场景下，可以通过手动处理 Sharding DDL Lock解决。

4.7.2 任务状态告警

4.7.2.1 DM_task_state

当 DM-worker 内有子任务处于 Paused 状态超过 20 分钟时会触发该告警，此时需要参考DM 故障诊断进行处理。

4.7.3 relay log 告警

4.7.3.1 DM_relay_process_exits_with_error

当 relay log 处理单元遇到错误时，会转为 Paused 状态并立即触发该告警，此时需要参考DM 故障诊断进行处理。

4.7.3.2 DM_remain_storage_of_relay_log

当 relay log 所在磁盘的剩余可用容量小于 10G 时会触发该告警，对应的处理方法包括：

- 手动清理该磁盘上其他无用数据以增加可用容量。
- 尝试调整 relay log 的自动清理策略或执行手动清理。
- 使用 pause-relay 命令暂停 relay log 的拉取，并在磁盘空间合适之后使用 resume-relay 命令恢复。需要注意上游数据源不要清理尚未拉取的 binlog。

4.7.3.3 DM_relay_log_data_corruption

当 relay log 处理单元在校验从上游读取到的 binlog event 且发现 checksum 信息异常时会转为 Paused 状态并立即触发告警，此时需要参考DM 故障诊断进行处理。

4.7.3.4 DM_fail_to_read_binlog_from_master

当 relay log 处理单元在尝试从上游读取 binlog event 发生错误时，会转为 Paused 状态并立即触发该告警，此时需要参考DM 故障诊断进行处理。

4.7.3.5 DM_fail_to_write_relay_log

当 relay log 处理单元在尝试将 binlog event 写入 relay log 文件发生错误时，会转为 Paused 状态并立即触发该告警，此时需要参考DM 故障诊断进行处理。

4.7.3.6 DM_binlog_file_gap_between_master_relay

当 relay log 处理单元已拉取到的最新的 binlog 文件个数落后于当前上游 MySQL/-MariaDB 超过 1 个（不含 1 个）且持续 10 分钟时会触发该告警，此时需要参考性能问题及处理方法对 relay log 处理单元相关的性能问题进行排查与处理。

4.7.4 Dump/Load 告警

4.7.4.1 DM_dump_process_exists_with_error

当 Dump 处理单元遇到错误时，会转为 Paused 状态并立即触发该告警，此时需要参考DM 故障诊断进行处理。

4.7.4.2 DM_load_process_exists_with_error

当 Load 处理单元遇到错误时，会转为 Paused 状态并立即触发该告警，此时需要参考DM 故障诊断进行处理。

4.7.5 Binlog replication 告警

4.7.5.1 DM_sync_process_exists_with_error

当 Binlog replication 处理单元遇到错误时，会转为 Paused 状态并立即触发该告警，此时需要参考DM 故障诊断进行处理。

4.7.5.2 DM_binlog_file_gap_between_master_syncer

当 Binlog replication 处理单元已处理到的最新的 binlog 文件个数落后于当前上游 MySQL/MariaDB 超过 1 个（不含 1 个）且持续 10 分钟时 DM 会触发该告警，此时需要参考性能问题及处理方法对 Binlog replication 处理单元相关的性能问题进行排查与处理。

4.7.5.3 DM_binlog_file_gap_between_relay_syncer

当 Binlog replication 处理单元已处理到的最新的 binlog 文件个数落后于当前 relay log 处理单元超过 1 个（不含 1 个）且持续 10 分钟时 DM 会触发该告警，此时需要参考性能问题及处理方法对 Binlog replication 处理单元相关的性能问题进行排查与处理。

4.8 TiDB Data Migration 日常巡检

本文总结了 TiDB Data Migration (DM) 工具日常巡检的方法：

- 方法一：执行 query-status 命令查看任务运行状态以及相关错误输出。详见查询状态。

- 方法二：如果使用 TiUP 部署 DM 集群时正确部署了 Prometheus 与 Grafana，如 Grafana 的地址为 172.16.10.71，可在浏览器中打开 http://172.16.10.71:3000 进入 Grafana，选择 DM 的 Dashboard 即可查看 DM 相关监控项。具体监控指标参照监控与告警设置。
- 方法三：通过日志文件查看 DM 运行状态和相关错误。
  - DM-master 日志目录：通过 DM-master 进程参数 `--log-file` 设置。如果使用 TiUP 部署 DM，则日志目录位于 `{log_dir}`。
  - DM-worker 日志目录：通过 DM-worker 进程参数 `--log-file` 设置。如果使用 TiUP 部署 DM，则日志目录位于 `{log_dir}`。

5 使用场景

5.1 从兼容 MySQL 的数据库迁移数据 ——以 Amazon Aurora MySQL 为例

本文以 Amazon Aurora MySQL 为例介绍如何使用 DM 从 MySQL 兼容的数据库迁移数据到 TiDB。

示例使用的 Aurora 集群信息如下：

集群	终端节点	端口	角色	版本
Aurora-1	test-dm-2-0.cluster-czrtqco96yc6.us-east-2.rds.amazonaws.com	3306	写入器	Aurora (MySQL)-5.7.12
Aurora-1	test-dm-2-0.cluster-ro-czrtqco96yc6.us-east-2.rds.amazonaws.com	3306	读取器	Aurora (MySQL)-5.7.12

集群	终端节点	端口	角色	版本
Aurora-2	test-dm-2-0-2.cluster-czrtqco96yc6.us-east-2.rds.amazonaws.com	3306	写入器	Aurora (MySQL)-5.7.12
Aurora-2	test-dm-2-0-2.cluster-ro-czrtqco96yc6.us-east-2.rds.amazonaws.com	3306	读取器	Aurora (MySQL)-5.7.12

Aurora 集群数据与迁移计划如下：

集群	数据库	表	是否迁移
Aurora-1	migrate_me	t1	是
Aurora-1	ignore_me	ignore_table	否
Aurora-2	migrate_me	t2	是
Aurora-2	ignore_me	ignore_table	否

迁移使用的 Aurora 集群用户如下：

集群	用户	密码
Aurora-1	root	12345678
Aurora-2	root	12345678

示例使用的 TiDB 集群信息如下。该集群使用 TiDB Cloud 服务一键部署：

节点	端口	版本
tidb.6657c286.23110bc6.us-east-1.prod.aws.tidbcloud.com	4000	v4.0.2

迁移使用的 TiDB 集群用户如下：

用户	密码
root	87654321

预期迁移后，TiDB 集群中存在表`migrate_me`.`t1`与`migrate_me`.`t2`，其中数据与 Aurora 集群一致。

> **注意：**
>
> 本次迁移不涉及合库合表，如需使用合库合表，参见DM 合库合表场景。

5.1.1 第 1 步：数据迁移前置条件

为了保证迁移成功，在开始迁移之前需要进行前置条件的检查。本文在此列出了需要的检查以及与 DM、Aurora 组件相关的解决方案。

5.1.1.1 DM 部署节点

DM 作为数据迁移的核心，需要正常连接上游 Aurora 集群与下游 TiDB 集群，因此通过 MySQL client 等方式检查部署 DM 的节点是否能连通上下游。除此以外，关于 DM 节点数目、软硬件等要求，参见DM 集群软硬件环境需求。

5.1.1.2 Aurora

DM 在增量复制阶段依赖 `ROW` 格式的 binlog，参见为 Aurora 实例启用 binlog 进行配置。

如果 Aurora 已开启了 GTID，则可以基于 GTID 进行数据迁移。GTID 的启用方式参见为 Aurora 集群启用 GTID 支持。基于 GTID 进行数据迁移，需要将第 3 步数据源配置文件中的 `enable-gtid` 设置为 `true`。

> **注意：**
>
> - 基于 GTID 进行数据迁移需要 MySQL 5.7 (Aurora 2.04) 或更高版本。
> - 除上述 Aurora 特有配置以外，上游数据库需满足迁移 MySQL 的其他要求，例如表结构、字符集、权限等，参见上游 MySQL 实例检查内容。

5.1.2 第 2 步：部署 DM 集群

DM 可以通过多种方式进行部署，目前推荐使用 TiUP 部署 DM 集群。具体部署方法，参见使用 TiUP 部署 DM 集群。示例有两个数据源，因此需要至少部署两个 DM-worker 节点。

部署完成后，需要记录任意一台 DM-master 节点的 IP 和服务端口（默认为 8261），以供 dmctl 连接。本示例使用 127.0.0.1:8261。通过 TiUP 使用 dmctl 检查 DM 状态：

> **注意：**
>
> 使用其他方式部署 DM 可以用类似的方式调用 dmctl，参见dmctl 简介。

```
tiup dmctl --master-addr 127.0.0.1:8261 list-member
```

返回值中的 master 与 worker 与部署数目一致：

```
{
    "result": true,
    "msg": "",
    "members": [
        {
            "leader": {
                ...
            }
        },
        {
            "master": {
                "msg": "",
                "masters": [
                    ...
                ]
            }
        },
        {
            "worker": {
                "msg": "",
                "workers": [
                    ...
                ]
            }
        }
    ]
}
```

5.1.3 第 3 步：配置数据源

> **注意：**
>
> DM 所使用的配置文件支持明文或密文数据库密码，推荐使用密文数据库密码确保安全。如何获得密文数据库密码，参见使用 dmctl 加密数据库密码。

根据示例信息保存如下的数据源配置文件，其中 source-id 的值将在第 4 步配置任务时被引用。

文件 source1.yaml：

```
## Aurora-1
source-id: "aurora-replica-01"

## 基于 GTID 进行数据迁移时，需要将该项设置为 true
enable-gtid: false

from:
  host: "test-dm-2-0.cluster-czrtqco96yc6.us-east-2.rds.amazonaws.com"
  user: "root"
  password: "12345678"
  port: 3306
```

文件 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
```

参见使用 DM 迁移数据：创建数据源，通过 TiUP 使用 dmctl 添加两个数据源。

```
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
```

添加数据源成功时，每个数据源的返回信息中包含了一个与之绑定的 DM-worker。

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "aurora-replica-01",
            "worker": "one-dm-worker-ID"
        }
    ]
}
```

5.1.4 第 4 步：配置任务

> **注意：**
>
> 由于 Aurora 不支持 FTWRL，仅使用全量模式导出数据时需要暂停写入，参见 AWS 官网说明。在示例的全量 + 增量模式下，DM 将自动启用safe mode 解决这一问题。在其他模式下如需保证数据一致，参见 AWS 官网说明操作。

本示例选择迁移 Aurora 已有数据并将新增数据实时迁移给 TiDB，即全量 + 增量模式。根据上文的 TiDB 集群信息、已添加的 source-id、要迁移的表，保存如下任务配置文件 task.yaml：

```
## 任务名，多个同时运行的任务不能重名
name: "test"
## 全量+增量 (all) 迁移模式
task-mode: "all"
## 下游 TiDB 配置信息
target-database:
  host: "tidb.6657c286.23110bc6.us-east-1.prod.aws.tidbcloud.com"
  port: 4000
  user: "root"
  password: "87654321"

## 当前数据迁移任务需要的全部上游 MySQL 实例配置
mysql-instances:
- source-id: "aurora-replica-01"
```

```
  # 需要迁移的库名或表名的黑白名单的配置项名称，用于引用全局的黑白名单配置，
    ↪ 全局配置见下面的 `block-allow-list` 的配置
  block-allow-list: "global"
  mydumper-config-name: "global"

- source-id: "aurora-replica-02"
  block-allow-list: "global"
  mydumper-config-name: "global"

## 黑白名单配置
block-allow-list:
  global:                           # 被上文 block-allow-list: "global" 引用
    do-dbs: ["migrate_me"]          # 需要迁移的上游数据库白名单。
      ↪ 白名单以外的库表不会被迁移

## Dump 单元配置
mydumpers:
  global:                           # 被上文 mydumper-config-name: "global"
    ↪ 引用
    extra-args: "--consistency none" # Aurora 不支持 FTWRL，配置此项以绕过
```

5.1.5 第 5 步：启动任务

通过 TiUP 使用 dmctl 启动任务。

> **注意：**
>
> 目前通过 TiUP 使用 dmctl 时，需要使用 task.yaml 绝对路径。TiUP 将会在后续更新中正确支持相对路径。

```
tiup dmctl --master-addr 127.0.0.1:8261 start-task /absolute/path/to/task.
  ↪ yaml --remove-meta
```

启动成功时的返回信息是：

```
{
    "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"
        }
    ]
}
```

如果返回信息中有 source db replication privilege checker、source db dump privilege checker 错误，请检查 errorMsg 字段是否存在不能识别的权限。例如：

```
line 1 column 287 near \"INVOKE LAMBDA ON *.* TO...
```

以上返回信息说明 INVOKE LAMBDA 权限导致报错。如果该权限是 Aurora 特有的，请在配置文件中添加如下内容跳过检查。DM 会在版本更新中增强对 Aurora 权限的自动处理。

```
ignore-checking-items: ["replication_privilege","dump_privilege"]
```

5.1.6 第 6 步：查询任务并验证数据

通过 TiUP 使用 dmctl 查询正在运行的迁移任务及任务状态等信息。

```
tiup dmctl --master-addr 127.0.0.1:8261 query-status
```

任务正常运行的返回信息是：

```
{
    "result": true,
    "msg": "",
    "tasks": [
        {
            "taskName": "test",
            "taskStatus": "Running",
            "sources": [
                "aurora-replica-01",
                "aurora-replica-02"
            ]
        }
    ]
}
```

用户也可以在下游查询数据，在 Aurora 中修改数据并验证到 TiDB 的数据复制。

5.2 下游 TiDB 表结构有更多列的迁移场景

本文介绍如何在下游 TiDB 表结构比上游存在更多列的情况下，使用 DM 对表进行迁移。

5.2.1 数据源表

本文档示例所用的数据源实例如下所示：

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

5.2.2 迁移要求

在 TiDB 中定制创建表 `log.messages`，其表结构包含数据源中 `log.messages` 表的所有列，且比数据源的表结构有更多的列。在此需求下，将数据源表 `log.messages` 迁移到 TiDB 集群的表 `log.messages`。

> 注意：
>
> - 下游 TiDB 相比于数据源多出的列必须指定默认值或允许为 `NULL`。
> - 对于已经通过 DM 在正常迁移的表，可直接在下游 TiDB 中新增指定了默认值或允许为 `NULL` 的列而不会影响 DM 正常的数据迁移。

5.2.3 只迁移数据源增量数据到 TiDB ，并且下游 TiDB 表结构有更多列的场景问题处理

如果该迁移任务包含全量数据迁移，迁移任务可以正常运行；但是如果你使用其他方式进行了全量迁移，只是用 DM 进行增量数据复制，可以参考只迁移数据源增量数据到 TiDB创建数据迁移任务，同时还需手动在 DM 中设置用于解析 MySQL binlog 的表结构。

否则，创建任务后，运行 `query-status` 会返回类似如下数据迁移错误：

```
"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": ""
    }
]
```

出现以上错误的原因是 DM 迁移 binlog event 时，如果 DM 内部没有维护对应于该表的表结构，则会尝试使用下游当前的表结构来解析 binlog event 并生成相应的 DML 语句。如果 binlog event 里数据的列数与下游表结构的列数不一致时，则会产生上述错误。

此时，我们可以使用operate-schema 命令来为该表指定与 binlog event 匹配的表结构。如果你在进行分表合并的数据迁移，那么需要为每个分表按照如下步骤在 DM 中设置用于解析 MySQL binlog 的表结构。具体操作为：

1. 为数据源中需要迁移的表 log.messages 指定表结构，表结构需要对应 DM 将要开始同步的 binlog event 的数据。将对应的 CREATE TABLE 表结构语句并保存到文件，例如将以下表结构保存到 log.messages.sql 中。

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

2. 使用operate-schema 命令设置表结构（此时 task 应该由于上述错误而处于 Paused 状态）。

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

3. 使用resume-task 命令恢复处于 Paused 状态的任务。

4. 使用query-status 命令确认数据迁移任务是否运行正常。

5.3 切换 DM-worker 与上游 MySQL 实例的连接

当需要对 DM-worker 所连接的上游 MySQL 实例进行停机维护或该实例意外宕机时，需要将 DM-worker 的连接切换到同一个主从复制集群内的另一个 MySQL 实例上。本文介绍如何将 DM-worker 的连接从一个 MySQL 实例切换到另一个 MySQL 实例上。

注意：

- 仅支持在同一个主从复制集内的 MySQL 实例间进行切换。
- 将要切换到的 MySQL 实例必须拥有 DM-worker 所需的 binlog。
- DM-worker 必须以 GTID sets 模式运行，即对应 source 配置文件中指定 `enable-gtid: true`。
- DM 仅支持以下两种切换场景，且必须严格按照各场景的步骤执行操作，否则可能需要根据切换后的 MySQL 实例重新搭建 DM 集群并完整重做数据迁移任务。

有关 GTID sets 的概念解释，请参考 MySQL 文档。

5.3.1 虚拟 IP 环境下切换 DM-worker 与 MySQL 实例的连接

如果 DM-worker 通过虚拟 IP (VIP) 连接上游的 MySQL 实例，更改 VIP 所指向的 MySQL 实例，即是在 DM-worker 对应上游连接地址不变的情况下切换 DM-worker 实际所连接的 MySQL 实例。

注意：

如果不对 DM 执行必要变更，当切换 VIP 所指向的 MySQL 实例时，DM 内部不同的 connection 可能会同时连接到切换前后不同的 MySQL 实例，造成 DM 拉取的 binlog 与从上游获取到的其他状态不一致，从而导致难以预期的异常行为甚至数据损坏。

如果 DM-worker 连接的 VIP 需要指向新的 MySQL 实例，需要按以下步骤进行操作：

1. 使用 `query-status` 命令获取当前 binlog replication 处理单元已复制到下游的 binlog 对应的 GTID sets (`syncerBinlogGtid`)，记为 `gtid-S`。
2. 在将要切换到的 MySQL 实例上使用 `SELECT @@GLOBAL.gtid_purged;` 获取已经被 purged 的 binlog 对应的 GTID sets，记为 `gtid-P`。
3. 在将要切换到的 MySQL 实例上使用 `SELECT @@GLOBAL.gtid_executed;` 获取所有已经执行成功的事务对应的 GTID sets，记为 `gtid-E`。
4. 确保满足以下关系，否则不支持将 DM-worker 连接切换到相应的 MySQL 实例：
   - `gtid-S` 包含 `gtid-P`（`gtid-P` 可以为空）
   - `gtid-E` 包含 `gtid-S`
5. 使用 `pause-task` 命令暂停所有运行中的数据迁移任务。
6. 变更 VIP 以指向新的 MySQL 实例。
7. 使用 `resume-task` 命令恢复之前的数据迁移任务。

5.3.2 变更 DM-worker 连接的上游 MySQL 实例地址

若要变更 DM-worker 的配置信息来使 DM-worker 连接到新的上游 MySQL 实例，需要按以下步骤进行操作：

1. 使用 `query-status` 命令获取当前 binlog replication 处理单元已复制到下游的 binlog 对应的 GTID sets (`syncerBinlogGtid`)，记为 `gtid-S`。
2. 在将要切换到的 MySQL 实例上使用 `SELECT @@GLOBAL.gtid_purged;` 获取已经被 purged 的 binlog 对应的 GTID sets，记为 `gtid-P`。
3. 在将要切换到的 MySQL 实例上使用 `SELECT @@GLOBAL.gtid_executed;` 获取所有已经执行成功的事务对应的 GTID sets，记为 `gtid-E`。
4. 确保满足以下关系，否则不支持将 DM-worker 连接切换到相应的 MySQL 实例：
   - `gtid-S` 包含 `gtid-P`（`gtid-P` 可以为空）
   - `gtid-E` 包含 `gtid-S`
5. 使用 `stop-task` 命令停止所有运行中的数据迁移任务。
6. 使用 `operate-source stop` 命令从 DM 集群中移除原 MySQL 实例地址对应的 source 配置。
7. 更新 source 配置文件中的 MySQL 实例地址并使用 `operate-source create` 命令将新的 source 配置重新加载到 DM 集群中。
8. 使用 `start-task` 命令重新启动数据迁移任务。

6 故障处理

6.1 故障及处理方法

本文档介绍 DM 的错误系统及常见故障的处理方法。

6.1.1 DM 错误系统

在 DM 的错误系统中，对于一条特定的错误，通常主要包含以下信息：

- `code`：错误码。

  同一种错误都使用相同的错误码。错误码不随 DM 版本改变。

  在 DM 迭代过程中，部分错误可能会被移除，但错误码不会。新增的错误会使用新的错误码，不会复用已有的错误码。

- `class`：发生错误的类别。

  用于标记出现错误的系统子模块。

  下表展示所有的错误类别、错误对应的系统子模块、错误样例：

  |

  错误类别

错误对应的系统子模块	错误样例
database	执行数据库操作出现错误	[code=10003:class=database:scope=downstream:level=medium] database driver: invalid connection
functional	系统底层的基础函数错误	[code=11005:class=functional:scope=internal:level=high] not allowed operation: alter multiple tables in one statement
config	配置错误	[code=20005:class=config:scope=internal:level=medium] empty source-id not valid
binlog-op	binlog 操作出现错误	[code=22001:class=binlog-op:scope=internal:level=high] empty UUIDs not valid
checkpoint	checkpoint 相关操作出现错误	[code=24002:class=checkpoint:scope=internal:level=high] save point bin.1234 is older than current pos bin.1371
task-check	进行任务检查时出现错误	[code=26003:class=task-check:scope=internal:level=medium] new table router error
relay-event-lib| 执行 relay 模块基础功能时出现错误	[code=28001:class=relay-event-lib:scope=internal:level=high] parse server-uuid.index
relay-unit	relay 处理单元内出现错误	[code=30015:class=relay-unit:scope=upstream:level=high] TCPReader get event: ERROR 1236 (HY000): Could not open log file
dump-unit	dump 处理单元内出现错误	[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 处理单元内出现错误	[code=34002:class=load-unit:scope=internal:level=high] corresponding ending of sql: ')' not found
sync-unit	sync 处理单元内出现错误	[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 服务内部出现错误	[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 服务内部出现错误	[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 服务内部出现错误	[code=42004:class=dm-tracer:scope=internal:level=medium] trace event test.1 not found
schema-tracker	增量复制时记录 schema 变更出现错误	[code=44006:class=

↪ schema-tracker:scope=internal:level=high],"cannot track DDL: ALTER
↪ TABLE test DROP COLUMN col1" |
scheduler | 数据迁移任务调度相关操作出错操作 | [code=46001:class=scheduler
↪ :scope=internal:level=high],"the scheduler has not started" |
dmctl | dmctl 内部或与其他组件交互出现错误 | [code=48001:class=dmctl:scope=
↪ internal:level=high],"can not create grpc connection" |

- scope：错误作用域。

  用于标识错误发生时 DM 作用对象的范围和来源，包括未设置 (not-set)、上游数据库 (upstream)、下游数据库 (downstream)、内部 (internal) 四种类型。

  如果错误发生的逻辑直接涉及到上下游数据库请求，作用域会设置为 upstream 或 downstream，其他出错场景目前都设置为 internal。

- level：错误级别。

  错误的严重级别，包括低级别 (low)、中级别 (medium)、高级别 (high) 三种。

  低级别通常是用户操作、输入错误，不影响正常迁移任务；中级别通常是用户配置等错误，会影响部分新启动服务，不影响已有系统迁移状态；高级别通常是用户需要关注的一些错误，可能存在迁移任务中断等风险，需要用户进行处理。

- message：错误描述。

  错误的详细描述信息。对于错误调用链上每一层额外增加的错误 message，采用 errors.Wrap 的模式来叠加和保存错误 message。wrap 最外层的 message 是 DM 内部对该错误的描述，wrap 最内层的 message 是该错误最底层出错位置的错误描述。

- workaround: 错误处理方法（可选）。

  对该错误的处理方法。对于部分明确的错误（如：配置信息错误等），DM 会在 workaround 中给出相应的人为处理方法。

- 错误堆栈信息（可选）。

  DM 根据错误的严重程度和必要性来选择是否输出错误堆栈。错误堆栈记录了错误发生时完整的堆栈调用信息。如果用户通过错误基本信息和错误 message 描述不能完全诊断出错误发生的原因，可以通过错误堆栈进一步跟进出错时代码的运行路径。

可在 DM 代码仓库 已发布的错误码 中查询完整的错误码列表。

6.1.2 DM 故障诊断

如果在运行 DM 工具时出现了错误，请尝试以下解决方案：

1. 执行 query-status 命令查看任务运行状态以及相关错误输出。
2. 查看与该错误相关的日志文件。日志文件位于 DM-master、DM-worker 部署节点上，通过DM 错误系统 获取错误的关键信息，然后查看常见故障处理方法以寻找相应的解决方案。

3. 如果该错误还没有相应的解决方案，并且你无法通过查询日志或监控指标自行解决此问题，你可以联系相关技术支持人员。

4. 一般情况下，错误处理完成后，只需使用 dmctl 重启任务即可。

```
resume-task ${task name}
```

但在某些情况下，你还需要重置数据迁移任务。有关何时需要重置以及如何重置，详见重置数据迁移任务。

6.1.3 常见故障处理方法

|

错误码

错误说明	解决方法
code=10001	数据库操作异常	进一步分析错误信息和错误堆栈	
code=10002	数据库底层的 bad connection 错误，通常表示 DM 到下游 TiDB 的数据库连接出现了异常（如网络故障、TiDB 重启等）且当前请求的数据暂时未能发送到 TiDB。	DM 提供针对此类错误的自动恢复。如果长时间未恢复，需要用户检查网络或 TiDB 状态。
code=10003	数据库底层 invalid connection 错误，通常表示 DM 到下游 TiDB 的数据库连接出现了异常（如网络故障、TiDB 重启、TiKV busy 等）且当前请求已有部分数据发送到了 TiDB。	DM 提供针对此类错误的自动恢复。如果未能正常恢复，需要用户进一步检查错误信息并根据具体场景进行分析。
code=10005	数据库查询类语句出错		
code=10006	数据库 EXECUTE 类型语句出错，包括 DDL 和 INSERT/UPDATE/DELETE 类型的 DML。更详细的错误信息可通过错误 message 获取。错误 message 中通常包含操作数据库所返回的错误码和错误信息。		
code=11006	DM 内置的 parser 解析不兼容的 DDL 时出错	可参考Data Migration 故障诊断 -处理不兼容的 DDL 语句 提供的解决方案
code=20010	处理任务配置时，解密数据库的密码出错	检查任务配置中提供的下游数据库密码是否有使用 dmctl 正确加密
code=26002	任务检查创建数据库连接失败。更详细的错误信息可通过错误 message 获取。错误 message 中包含操作数据库所返回的错误码和错误信息。	检查 DM-master 所在的机器是否有权限访问上游
code=32001	dump 处理单元异常	如果报错 msg 包含 mydumper: argument list too long.，则需要用户根据 block-allow-list，在 task.yaml 的 dump 处理单元的 extra-args 参数中手动加上 --regex 正则表达式设置要导出的库表。例如，如果要导出所有库中表名字为 hello 的表，可加上 --regex '.*\\.hello$'，如果要导出所有表，可加上 --regex '.*'。

code=38008 | DM 组件间的 gRPC 通信出错 | 检查 class，定位错误发生在哪些组件的交互环节，根据错误 message 判断是哪类通信错误。如果是 gRPC 建立连接出错，可检查通信服务端是否运行正常。|

6.1.3.1 迁移任务中断并包含 invalid connection 错误

6.1.3.1.1 原因

发生 invalid connection 错误时，通常表示 DM 到下游 TiDB 的数据库连接出现了异常（如网络故障、TiDB 重启、TiKV busy 等）且当前请求已有部分数据发送到了 TiDB。

6.1.3.1.2 解决方案

由于 DM 中存在迁移任务并发向下游复制数据的特性，因此在任务中断时可能同时包含多个错误（可通过 query-status 查询当前错误）。

- 如果错误中仅包含 invalid connection 类型的错误且当前处于增量复制阶段，则 DM 会自动进行重试。
- 如果 DM 由于版本问题等未自动进行重试或自动重试未能成功，则可尝试先使用 stop-task 停止任务，然后再使用 start-task 重启任务。

6.1.3.2 迁移任务中断并包含 driver: bad connection 错误

6.1.3.2.1 原因

发生 driver: bad connection 错误时，通常表示 DM 到下游 TiDB 的数据库连接出现了异常（如网络故障、TiDB 重启等）且当前请求的数据暂时未能发送到 TiDB。

6.1.3.2.2 解决方案

当前版本 DM 会自动进行重试，如果由于版本问题等未自动重试，可先使用 stop-task 停止任务后再使用 start-task 重启任务。

6.1.3.3 relay 处理单元报错 event from * in * diff from passed-in event * 或迁移任务中断并包含 get binlog error ERROR 1236 (HY000)、binlog checksum mismatch, data may be corrupted 等 binlog 获取或解析失败错误

6.1.3.3.1 原因

在 DM 进行 relay log 拉取与增量复制过程中，如果遇到了上游超过 4GB 的 binlog 文件，就可能出现这两个错误。

原因是 DM 在写 relay log 时需要依据 binlog position 及文件大小对 event 进行验证，且需要保存迁移的 binlog position 信息作为 checkpoint。但是 MySQL binlog position 官方定义使用 uint32 存储，所以超过 4G 部分的 binlog position 的 offset 值会溢出，进而出现上面的错误。

6.1.3.3.2 解决方案

对于 relay 处理单元，可通过以下步骤手动恢复：

1. 在上游确认出错时对应的 binlog 文件的大小超出了 4GB。
2. 停止 DM-worker。
3. 将上游对应的 binlog 文件复制到 relay log 目录作为 relay log 文件。
4. 更新 relay log 目录内对应的 `relay.meta` 文件以从下一个 binlog 开始拉取。如果 DM worker 已开启 `enable_gtid`，那么在修改 `relay.meta` 文件时，同样需要修改下一个 binlog 对应的 GTID。如果未开启 `enable_gtid` 则无需修改 GTID。

   例如：报错时有 `binlog-name = "mysql-bin.004451"` 与 `binlog-pos = 2453`，则将其分别更新为 `binlog-name = "mysql-bin.004452"` 和 `binlog-pos = 4`，同时更新 `binlog-gtid = "f0e914ef-54cf-11e7-813d-6c92bf2fa791:1-138218058"`。
5. 重启 DM-worker。

对于 binlog replication 处理单元，可通过以下步骤手动恢复：

1. 在上游确认出错时对应的 binlog 文件的大小超出了 4GB。
2. 通过 `stop-task` 停止迁移任务。
3. 将下游 `dm_meta` 数据库中 global checkpoint 与每个 table 的 checkpoint 中的 `binlog_name` 更新为出错的 binlog 文件，将 `binlog_pos` 更新为已迁移过的一个合法的 position 值，比如 4。

   例如：出错任务名为 `dm_test`，对应的 `source-id` 为 `replica-1`，出错时对应的 binlog 文件为 `mysql-bin|000001.004451`，则执行 `UPDATE dm_test_syncer_checkpoint ↪ SET binlog_name='mysql-bin|000001.004451', binlog_pos = 4 WHERE id=' ↪ replica-1';`。
4. 在迁移任务配置中为 `syncers` 部分设置 `safe-mode: true` 以保证可重入执行。
5. 通过 `start-task` 启动迁移任务。
6. 通过 `query-status` 观察迁移任务状态，当原造成出错的 relay log 文件迁移完成后，即可还原 `safe-mode` 为原始值并重启迁移任务。

6.1.3.4 执行 `query-status` 或查看日志时出现 `Access denied for user 'root'@'172.31.43.27' (using password: YES)`

在所有 DM 配置文件中，数据库相关的密码都推荐使用经 dmctl 加密后的密文（若数据库密码为空，则无需加密）。有关如何使用 dmctl 加密明文密码，参见使用 dmctl 加密数据库密码。

此外，在 DM 运行过程中，上下游数据库的用户必须具备相应的读写权限。在启动迁移任务过程中，DM 会自动进行相应权限的前置检查，详见上游 MySQL 实例配置前置检查。

6.1.3.5 load 处理单元报错 packet for query is too large. Try adjusting the 'max_allowed_packet' variable

6.1.3.5.1 原因

- MySQL client 和 MySQL/TiDB Server 都有 max_allowed_packet 配额的限制，如果在使用过程中违反其中任何一个 max_allowed_packet 配额，客户端程序就会收到对应的报错。目前最新版本的 DM 和 TiDB Server 的默认 max_allowed_packet 配额都为 64M。
- DM 的全量数据导入处理模块不支持对 dump 处理模块导出的 SQL 文件进行切分。

6.1.3.5.2 解决方案

- 推荐在 DM 的 dump 处理单元提供的配置 extra-args 中设置 statement-size:

  依据默认的 --statement-size 设置，DM 的 dump 处理单元默认生成的 Insert Statement 大小一般会在 1M 左右，使用默认值就可以确保绝大部分情况 load 处理单元不会报错 packet for query is too large. Try adjusting the 'max_allowed_packet' variable。

  有时候在 dump 过程中会出现下面的 WARN log。这个 WARN log 不影响 dump 的过程，只是说明 dump 的表可能是宽表。

```
Row bigger than statement_size for xxx
```

- 如果宽表的单行超过了 64M，需要修改以下两项配置，并且确保其生效。
  - 在 TiDB Server 执行 set @@global.max_allowed_packet=134217728 (134217728 = 128M)
  - 根据实际情况为 DM 的任务配置文件中的 target-database 增加配置 max-allowed-packet: 134217728 (128M)，执行 stop-task 后再重新 start-task。

6.2 性能问题及处理方法

本文档介绍 DM 中可能存在的、常见的性能问题及其处理方法。

在诊断与处理性能问题时，请确保已经正确配置并安装 DM 的监控组件，并能在 Grafana 监控面板查看DM 的监控指标。

在诊断性能问题时，请先确保对应组件正在正常运行，否则可能出现监控指标异常的情况，对性能问题的诊断造成干扰。

在诊断问题前，也可以先了解 DM 的性能测试报告。

当数据迁移过程存在较大延迟时，若需快速定位瓶颈是在 DM 组件内部还是在 TiDB 集群，可先排查写入 SQL 到下游 部分的 DML queue remain length。

6.2.1 relay log 模块的性能问题及处理方法

在relay log 的监控部分，可以主要通过 `binlog file gap between master and relay` 监控项确认是否存在性能问题。如果该指标长时间大于 1，通常表明存在性能问题；如果该指标基本为 0，一般表明没有性能问题。

如果 `binlog file gap between master and relay` 基本为 0，但仍怀疑存在性能问题，则可以继续查看 `binlog pos`，如果该指标中 `master` 远大于 `relay`，则表明可能存在性能问题。

如果存在性能问题，则继续根据 relay log 模块的主要处理流程分别进行诊断与处理。

6.2.1.1 读取 binlog 数据

与 relay log 模块从上游读取 binlog 数据相关的主要性能指标是 `read binlog event duration`，该指标表示从上游 MySQL/MariaDB 读取到单个 binlog event 所需要的时间，理想情况下应接近于 DM-worker 与 MySQL/MariaDB 实例间的网络延迟。

对于同机房的数据迁移，这部分一般不会成为性能瓶颈；如果该值过大，请排查 DM-worker 与 MySQL/MariaDB 间的网络连通情况。

对于跨机房的数据迁移，可尝试将 DM-worker 与 MySQL/MariaDB 部署在同一机房，而仍将 TiDB 集群部署在目标机房。

从上游读取 binlog 数据这一流程细分后包括以下三个子流程：

- 上游 MySQL/MariaDB 从本地读取 binlog 数据并通过网络进行发送。上游 MySQL/MariaDB 负载无异常时，该子流程通常不会成为瓶颈。
- binlog 数据通过网络从 MySQL/MariaDB 所在机器传输到 DM-worker 所在机器。该子流程主要由 DM-worker 与上游 MySQL/MariaDB 的网络连通情况决定。
- DM-worker 从网络数据流中读取 binlog 数据，并构造成 binlog event。当 DM-worker 负载无异常时，该子流程通常不会成为瓶颈。

> **注意：**
>
> 如果 `read binlog event duration` 的值较大，另一个可能的原因是上游 MySQL/MariaDB 负载较低，一段时间内暂时没有需要发送给 DM 的 binlog event，relay log 模块处于等待状态，导致该值包含了额外的等待时间。

6.2.1.2 binlog 数据解码与验证

将 binlog event 读取到 DM 内存后，会进行必要的数据解码与验证，这部分通常不会存在性能瓶颈，因此监控面板上默认无对应性能指标。如果需要查看相应指标，可手动为 Prometheus 中的 `dm_relay_read_transform_duration` 添加相应的监控。

6.2.1.3 写入 relay log 文件

在将 binlog event 写入 relay log 文件时，相关的主要性能指标是 `write relay log duration`，该指标在 `binlog event size` 不是特别大时，值应在微秒级别。如果该值过大，需排查磁盘写入性能，如尽量优先为 DM-worker 使用本地 SSD 等。

6.2.2 Load 模块的性能问题及处理方法

Load 模块主要操作为从本地读取 SQL 文件数据并写入到下游，对应的主要性能指标是 `transaction execution latency`，如果该值过大，则通常需要根据下游数据库的监控对下游性能进行排查。

另外，也可以查看是否 DM 到下游数据库间的网络存在较大的延迟。

6.2.3 Binlog replication 模块的性能问题及处理方法

在Binlog replication 的监控部分，可以主要通过 `binlog file gap between master and syncer` 监控项确认是否存在性能问题，如果该指标长时间大于 1，则通常表明存在性能问题；如果该指标基本为 0，则一般表明没有性能问题。

如果 `binlog file gap between master and syncer` 长时间大于 1，则可以再通过 `binlog file gap between relay and syncer` 判断延迟主要存在于哪个模块，如果该值基本为 0，则延迟可能存在于 relay log 模块，请先参考relay log 模块的性能问题及处理方法 进行处理；否则继续对 Binlog replication 进行排查。

6.2.3.1 读取 binlog 数据

Binlog replication 模块会根据配置选择从上游 MySQL/MariaDB 或 relay log 文件中读取 binlog event，对应的主要性能指标是 `read binlog event duration`，该值的范围一般是几微秒至几十微秒。

- 如果是从上游 MySQL/MariaDB 读取 binlog event，则可参考 relay log 模块下的读取 binlog 数据进行排查与处理。
- 如果是从 relay log 文件中读取，则在 `binlog event size` 不是特别大时，`read binlog event duration` 的值应在微秒级别。如果 `read binlog event duration` 过大，则需排查磁盘读取性能，如尽量优先为 DM-worker 使用本地 SSD 等。

6.2.3.2 binlog event 转换

Binlog replication 模块从 binlog event 数据中尝试构造 DML、解析 DDL 以及进行table router 转换等，主要的性能指标是 `transform binlog event duration`。

这部分的耗时受上游写入的业务特点影响较大，如对于 `INSERT INTO` 语句，转换单个 `VALUES` 的时间和转换大量 `VALUES` 的时间差距很多，其波动范围可能从几十微秒至上百微秒，但一般不会成为系统的瓶颈。

6.2.3.3 写入 SQL 到下游

Binlog replication 模块将转换后的 SQL 写入到下游时，涉及到的性能指标主要包括 `DML queue remain length` 与 `transaction execution latency`。

DM 在从 binlog event 构造出 SQL 后，会使用 `worker-count` 个队列尝试并发写入到下游。但为了避免监控条目过多，会将并发队列编号按 8 取模，即所有并发队列在监控上会对应到 `q_0` 到 `q_7` 的某一项。

`DML queue remain length` 用于表示并发处理队列中尚未取出并开始用于向下游写入的 DML 语句数，理想情况下，各 `q_*` 对应的曲线应该基本一致，如果极不一致则表明并发的负载极不均衡。

如果负载不均衡，请确认需要迁移的所有表结构中都有主键或唯一键，如没有主键或唯一键则请尝试为其添加主键或唯一键；如果存在主键或唯一键时仍存在该问题，可尝试升级 DM 到 v1.0.5 及以上的版本。

当整个数据迁移链路无明显延迟时，`DML queue remain length` 对应曲线应基本为 0，且最大通常应不超过任务配置文件中的 `batch` 值。

如果确认数据迁移链路存在明显延迟，且 `DML queue remain length` 中各 `q_*` 对应的曲线基本一致且基本为 0，则表明 DM 未能及时地从上游读取数据、进行转换或进行并行分发（如瓶颈存在于 relay log 模块等），请参考本文档前述各节进行排查。

如果 `DML queue remain length` 对应曲线不为 0（最大一般不超过 1024），则通常表明向下游写入 SQL 时存在瓶颈，可通过 `transaction execution latency` 查看向下游执行单个事务的耗时情况。

`transaction execution latency` 一般应在几十毫秒。如果该值过高，则通常需要根据下游数据库的监控对下游性能进行排查，另外也可以关注是否 DM 到下游数据库间的网络存在较大的延迟。

此外，也可通过 `statement execution latency` 查看向下游写入 `BEGIN`、`INSERT` /`UPDATE`/`DELETE`、`COMMIT` 等单条语句的耗时情况。

7 性能调优

7.1 DM 配置优化

本文档介绍如何对迁移任务的配置进行优化，从而提高 DM 的数据迁移性能。

7.1.1 全量导出

全量导出相关的配置项为 `mydumpers`，下面介绍和性能相关的参数如何配置。

7.1.1.1 `rows`

设置 `rows` 选项可以开启单表多线程并发导出，值为导出的每个 chunk 包含的最大行数。开启后，DM 会在 MySQL 的单表并发导出时，优先选出一列做拆分基准，选择的

优先级为主键 > 唯一索引 > 普通索引，选出目标列后需保证该列为整数类型（如 INT、MEDIUMINT、BIGINT 等）。

rows 的值可以设置为 10000，具体设置的值可以根据表中包含数据的总行数以及数据库的性能做调整。另外也需要设置 threads 来控制并发线程数量，默认值为 4，可以适当做些调整。

7.1.1.2 chunk-filesize

DM 全量备份时会根据 chunk-filesize 参数的值把每个表的数据划分成多个 chunk，每个 chunk 保存到一个文件中，大小约为 chunk-filesize。根据这个参数把数据切分到多个文件中，这样就可以利用 DM Load 处理单元的并行处理逻辑提高导入速度。该参数默认值为 64（单位为 MB），正常情况下不需要设置，也可以根据全量数据的大小做适当的调整。

注意：

- mydumpers 的参数值不支持在迁移任务创建后更新，所以需要在创建任务前确定好各个参数的值。如果需要更新，则需要使用 dmctl stop 任务后更新配置文件，然后再重新创建任务。
- mydumpers.threads 可以使用配置项 mydumper-thread 替代来简化配置。
- 如果设置了 rows，DM 会忽略 chunk-filesize 的值。

7.1.2 全量导入

全量导入相关的配置项为 loaders，下面介绍和性能相关的参数如何配置。

7.1.2.1 pool-size

pool-size 为 DM Load 阶段线程数量的设置，默认值为 16，正常情况下不需要设置，也可以根据全量数据的大小以及数据库的性能做适当的调整。

注意：

- loaders 的参数值不支持在迁移任务创建后更新，所以需要在创建任务前确定好各个参数的值。如果需要更新，则需要使用 dmctl stop 任务后更新配置文件，然后再重新创建任务。
- loaders.pool-size 可以使用配置项 loader-thread 替代来简化配置。

7.1.3 增量复制

增量复制相关的配置为 syncers，下面介绍和性能相关的参数如何配置。

7.1.3.1 worker-count

worker-count 为 DM Sync 阶段并发迁移 DML 的线程数量设置，默认值为 16，如果对迁移速度有较高的要求，可以适当调高改参数的值。

7.1.3.2 batch

batch 为 DM Sync 阶段迁移数据到下游数据库时，每个事务包含的 DML 的数量，默认值为 100，正常情况下不需要调整。

注意：

- syncers 的参数值不支持在迁移任务创建后更新，所以需要在创建任务前确定好各个参数的值。如果需要更新，则需要使用 dmctl stop 任务后更新配置文件，然后再重新创建任务。
- syncers.worker-count 可以使用配置项 syncer-thread 替代来简化配置。
- worker-count 和 batch 的设置需要根据实际的场景进行调整，例如：DM 到下游数据库的网络延迟较高，可以适当调高 worker-count，调低 batch。

8 参考指南

8.1 架构

8.1.1 Data Migration 架构

DM 主要包括三个组件：DM-master，DM-worker 和 dmctl。

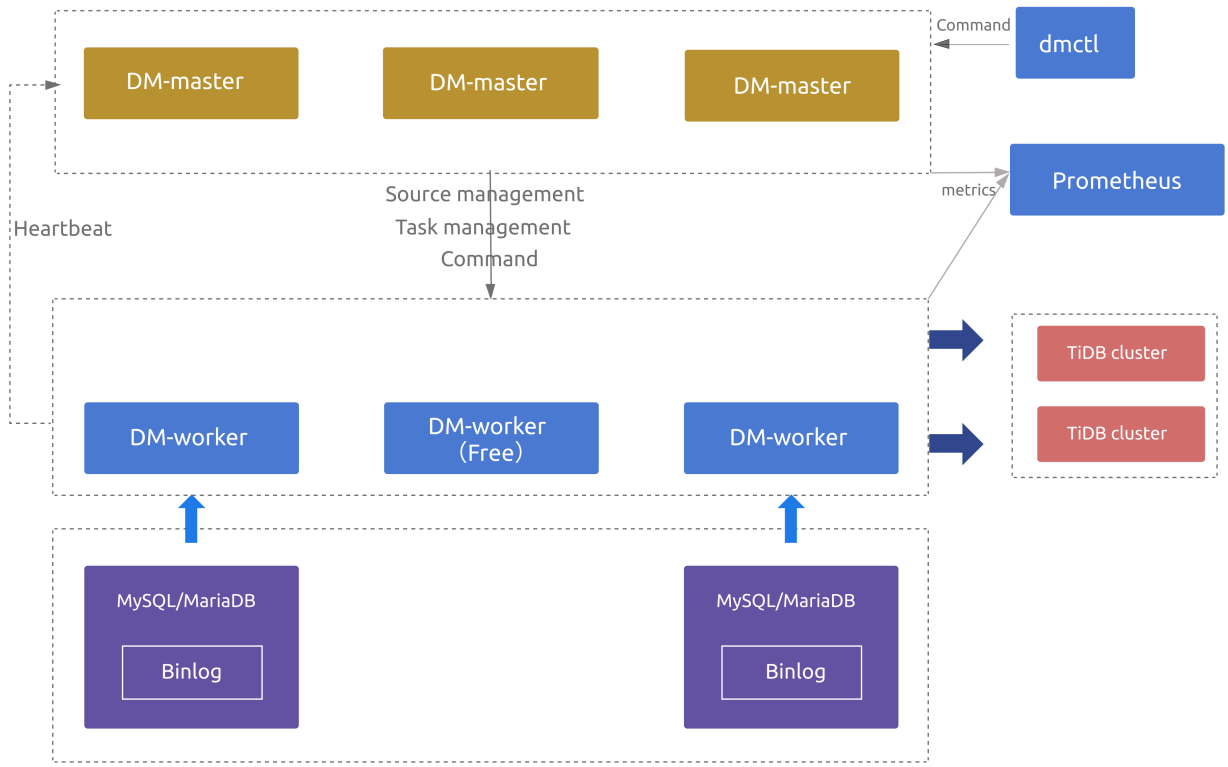


Figure 23: Data Migration architecture

8.1.1.1 架构组件

8.1.1.1.1 DM-master

DM-master 负责管理和调度数据迁移任务的各项操作。

- 保存 DM 集群的拓扑信息
- 监控 DM-worker 进程的运行状态
- 监控数据迁移任务的运行状态
- 提供数据迁移任务管理的统一入口
- 协调分库分表场景下各个实例分表的 DDL 迁移

8.1.1.1.2 DM-worker

DM-worker 负责执行具体的数据迁移任务。

- 将 binlog 数据持久化保存在本地
- 保存数据迁移子任务的配置信息
- 编排数据迁移子任务的运行
- 监控数据迁移子任务的运行状态

有关于 DM-worker 的更多介绍，详见DM-worker 简介。

8.1.1.1.3 dmctl

dmctl 是用来控制 DM 集群的命令行工具。

- 创建、更新或删除数据迁移任务
- 查看数据迁移任务状态
- 处理数据迁移任务错误
- 校验数据迁移任务配置的正确性

有关于 dmctl 的使用介绍，详见dmctl 使用。

8.1.1.2 架构特性

8.1.1.2.1 高可用

当部署多个 DM-master 节点时，所有 DM-master 节点将使用内部嵌入的 etcd 组成集群。该 DM-master 集群用于存储集群节点信息、任务配置等元数据，同时通过 etcd 选举出 leader 节点。该 leader 节点用于提供集群管理、数据迁移任务管理相关的各类服务。因此，若可用的 DM-master 节点数超过部署节点的半数，即可正常提供服务。

当部署的 DM-worker 节点数超过上游 MySQL/MariaDB 节点数时，超出上游节点数的相关 DM-worker 节点默认将处于空闲状态。若某个 DM-worker 节点下线或与 DM-master leader 发生网络隔离，DM-master 能自动将与原 DM-worker 节点相关的数据迁移任务调度到其他空闲的 DM-worker 节点上（若原 DM-worker 节点为网络隔离状态，则其会自动停止相关的数据迁移任务）；若无空闲的 DM-worker 节点可供调度，则原 DM-worker 相关的数据迁移任务将暂时挂起，直到有空闲 DM-worker 节点后自动恢复。

> 注意：
>
> 当数据迁移任务处于全量导出或导入阶段时，该迁移任务暂不支持高可用，主要原因为：
>
> - 对于全量导出，MySQL 暂不支持指定从特定快照点导出，也就是说数据迁移任务被重新调度或重启后，无法继续从前一次中断时刻继续导出。
> - 对于全量导入，DM-worker 暂不支持跨节点读取全量导出数据，也就是说数据迁移任务被调度到的新 DM-worker 节点无法读取调度发生前原 DM-worker 节点上的全量导出数据。

8.1.2 DM-worker 简介

DM-worker 是 DM (Data Migration) 的一个组件，负责执行具体的数据迁移任务。

其主要功能如下：

- 注册为一台 MySQL 或 MariaDB 服务器的 slave。
- 读取 MySQL 或 MariaDB 的 binlog event，并将这些 event 持久化保存在本地 (relay log)。
- 单个 DM-worker 支持迁移一个 MySQL 或 MariaDB 实例的数据到下游的多个 TiDB 实例。
- 多个 DM-Worker 支持迁移多个 MySQL 或 MariaDB 实例的数据到下游的一个 TiDB 实例。

8.1.2.1 DM-worker 处理单元

DM-worker 任务包含如下多个逻辑处理单元。

8.1.2.1.1 Relay log

Relay log 持久化保存从上游 MySQL 或 MariaDB 读取的 binlog，并对 binlog replication 处理单元提供读取 binlog event 的功能。

其原理和功能与 MySQL relay log 类似，详见 MySQL Relay Log。

8.1.2.1.2 dump 处理单元

dump 处理单元从上游 MySQL 或 MariaDB 导出全量数据到本地磁盘。

8.1.2.1.3 load 处理单元

load 处理单元读取 dump 处理单元导出的数据文件，然后加载到下游 TiDB。

8.1.2.1.4 Binlog replication/sync 处理单元

Binlog replication/sync 处理单元读取上游 MySQL/MariaDB 的 binlog event 或 relay log 处理单元的 binlog event，将这些 event 转化为 SQL 语句，再将这些 SQL 语句应用到下游 TiDB。

8.1.2.2 DM-worker 所需权限

本小节主要介绍使用 DM-worker 时所需的上下游数据库用户权限以及各处理单元所需的用户权限。

8.1.2.2.1 上游数据库用户权限

上游数据库 (MySQL/MariaDB) 用户必须拥有以下权限：

权限	作用域
SELECT	Tables
RELOAD	Global
REPLICATION SLAVE	Global
REPLICATION CLIENT	Global

如果要迁移 db1 的数据到 TiDB，可执行如下的 GRANT 语句：

```
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';
```

如果还要迁移其他数据库的数据到 TiDB，请确保已赋予这些库跟 db1 一样的权限。

8.1.2.2.2 下游数据库用户权限

下游数据库 (TiDB) 用户必须拥有以下权限：

权限	作用域
SELECT	Tables
INSERT	Tables
UPDATE	Tables
DELETE	Tables
CREATE	Databases，tables
DROP	Databases，tables
ALTER	Tables
INDEX	Tables

对要执行迁移操作的数据库或表执行下面的 GRANT 语句：

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

8.1.2.2.3 处理单元所需的最小权限

处理单元	最小上游 (MySQL/MariaDB) 权限	最小下游 (TiDB) 权限	最小系统权限
Relay log	REPLICATION SLAVE (读取 binlog) REPLICATION CLIENT (show master ↪ status, show slave status)	无	本地读/写磁盘
Dump	SELECTRELOAD (获取读锁将表数据刷到磁盘，进行一些操作后，再释放读锁对表进行解锁)	无	本地写磁盘

处理单元	最小上游 (MySQL/MariaDB) 权限	最小下游 (TiDB) 权限	最小系统权限
Load	无	SELECT（查询 checkpoint 历史） CREATE（创建数据库或表）DELETE（删除 checkpoint） INSERT（插入 dump 数据）	读/写本地文件
Binlog replication	REPLICATION SLAVE（读 binlog） REPLICATION CLIENT (show master ↪ status, show slave status)	SELECT（显示索引和列） INSERT (DML)UPDATE ↪ (DML)DELETE ↪ (DML)CREATE ↪ （创建数据库或表） DROP（删除数据库或表） ALTER（修改表）INDEX（创建或删除索引）	本地读/写磁盘

注意：

这些权限并非一成不变。随着需求改变，这些权限也可能会改变。

8.2 DM 命令行参数

本文档介绍 DM 中各组件的主要命令行参数。

8.2.1 DM-master

8.2.1.1 --advertise-addr

- DM-master 用于接收客户端请求的外部地址
- 默认值为 "{master-addr}"
- 可选参数，可以为 "域名:port" 的形式

8.2.1.2 --advertise-peer-urls

- DM-master 节点间通信的外部连接地址
- 默认值为 "{peer-urls}"
- 可选参数，可以为 "http(s)://域名:port" 的形式

8.2.1.3 --config

- DM-master 配置文件路径
- 默认值为 ""
- 可选参数

8.2.1.4 --data-dir

- DM-master 用于存储自身数据的目录
- 默认值为 "default.{name}"
- 可选参数

8.2.1.5 --initial-cluster

- 用于 bootstrap DM-master 集群的 "{节点名}={外部地址}" 列表
- 默认值为 "{name}={advertise-peer-urls}"
- 在未指定 join 参数时需要指定该参数。一个 3 节点集群的配置示例为 "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

- DM-master 节点加入到已有集群时，已有集群的 advertise-addr 地址列表
- 默认值为 ""
- 未指定 initial-cluster 参数时需要指定该参数。一个新节点加入到一个已有 2 个节点的集群的示例为 "172.16.15.11:8261,172.16.15.12:8261"

8.2.1.7 --log-file

- log 输出文件名
- 默认值为 ""
- 可选参数

8.2.1.8 -L

- log 级别
- 默认值为 "info"
- 可选参数

8.2.1.9 --master-addr

- DM-master 监听客户端请求的地址
- 默认值为 ""
- 必选参数

8.2.1.10 --name

- DM-master 节点名称
- 默认值为 "dm-master-{hostname}"
- 必选参数

8.2.1.11 --peer-urls

- DM-master 节点间通信的监听地址
- 默认值为 "http://127.0.0.1:8291"
- 必选参数

8.2.2 DM-worker

8.2.2.1 --advertise-addr

- DM-worker 用于接受客户端请求的外部地址
- 默认值为 "{worker-addr}"
- 可选参数，可以为 "域名:port" 的形式

8.2.2.2 --config

- DM-worker 配置文件路径
- 默认值为 ""
- 可选参数

8.2.2.3 --join

- DM-worker 注册到集群时，相应集群的 DM-master 节点的 {advertise-addr} 列表
- 默认值为 ""
- 必选参数，一个 3 DM-master 节点的集群配置示例为 "172.16.15.11:8261,172.16.15.12:8261,17
  ↪

8.2.2.4 --log-file

- log 输出文件名
- 默认值为 ""
- 可选参数

8.2.2.5 -L

- log 级别
- 默认值为 "info"
- 可选参数

8.2.2.6 --name

- DM-worker 节点名称
- 默认值为 "{advertise-addr}"
- 必选参数

8.2.2.7 --worker-addr

- DM-worker 监听客户端请求的地址
- 默认值为 ""
- 必选参数

8.2.3 dmctl

8.2.3.1 --config

- dmctl 配置文件路径
- 默认值为 ""
- 可选参数

8.2.3.2 --master-addr

- dmctl 要连接的集群的任意 DM-master 节点的 {advertise-addr}
- 默认值为 ""
- 需要与 DM-master 交互时为必选参数

8.2.3.3 --encrypt

- 将明文数据库密码加密成密文
- 默认值为 ""
- 指定该参数时，仅用于加密明文而不会与 DM-master 交互

8.2.3.4 --decrypt

- 将使用 dmctl 加密过的密文解密为明文
- 默认值为 ""
- 指定该参数时，仅用于解密密文而不会与 DM-master 交互

8.3 配置

8.3.1 DM 配置简介

本文档简要介绍 DM (Data Migration) 的配置文件和数据迁移任务的配置。

8.3.1.1 配置文件

- dm-master.toml：DM-master 进程的配置文件，包括 DM-master 的拓扑信息、日志等各项配置。配置说明详见DM-master 配置文件介绍。
- dm-worker.toml：DM-worker 进程的配置文件，包括 DM-worker 的拓扑信息、日志等各项配置。配置说明详见DM-worker 配置文件介绍。
- source.yaml：上游数据库 MySQL/MariaDB 相关配置。配置说明详见上游数据库配置文件介绍。

8.3.1.2 迁移任务配置

8.3.1.2.1 创建数据迁移任务

具体步骤如下：

1. 使用 dmctl 将数据源配置加载到 DM 集群；
2. 参考数据任务配置向导来创建 your_task.yaml；
3. 使用 dmctl 创建数据迁移任务。

8.3.1.2.2 关键概念

DM 配置的关键概念如下：

概念	解释	配置文件
source-id	唯一确定一个 MySQL 或 MariaDB 实例，或者一个具有主从结构的复制组，字符串长度不大于 32	`source.yaml` 的 `source-id`；`task.yaml` 的 `source-id`
DM-master ID	唯一确定一个 DM-master（取值于 `dm-master.toml` 的 `master-addr` 参数）	`dm-master.toml` 的 `master-addr`
DM-worker ID	唯一确定一个 DM-worker（取值于 `dm-worker.toml` 的 `worker-addr` 参数）	`dm-worker.toml` 的 `worker-addr`

8.3.2 DM-master 配置文件介绍

本文介绍 DM-master 的配置文件，包括配置文件示例与配置项说明。

8.3.2.1 配置文件示例

DM-master 的示例配置文件如下所示：

```
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 配置项说明

8.3.2.2.1 Global 配置

配置项	说明
name	标识一个 DM-master。
log-level	日志级别：debug、info、warn、error、fatal。默认为 info。
log-file	日志文件，如果不配置，日志会输出到标准输出中。
master-addr	DM-master 服务的地址，可以省略 IP 信息，例如：":8261"。
advertise-addr	DM-master 向外界宣告的地址。
peer-urls	DM-master 节点的对等 URL。
advertise-peer-urls	DM-master 向外界宣告的对等 URL。默认为 peer-urls 的值。
initial-cluster	初始集群中所有 DM-master 的 advertise-peer-urls 的值。
join	集群里已有的 DM-master 的 advertise-peer-urls 的值。如果是新加入的 DM-master 节点，使用 join 替代 initial-cluster。
ssl-ca	DM-master 组件用于与其它组件连接的 SSL CA 证书所在的路径
ssl-cert	DM-master 组件用于与其它组件连接的 PEM 格式的 X509 证书所在的路径
ssl-key	DM-master 组件用于与其它组件连接的 PEM 格式的 X509 密钥所在的路径
cert-allowed-cn	证书检查 Common Name 列表

8.3.3 DM-worker 配置文件介绍

本文介绍 DM-worker 的配置文件，包括配置文件示例与配置项说明。

8.3.3.1 配置文件示例

```
### 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 # 版本 2.0.2 新增

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 配置项说明

8.3.3.2.1 Global 配置

配置项	说明
name	标识一个 DM-worker。
log-level	日志级别：debug、info、warn、error、fatal。默认为 info。
log-file	日志文件，如果不配置日志会输出到标准输出中。
worker-addr	DM-worker 服务的地址，可以省略 IP 信息，例如：":8262"。
advertise-addr	DM-worker 向外界宣告的地址。
join	对应一个或多个 DM-master 配置中的master-addr。

配置项	说明
keepalive- ↪ ttl	当绑定的上游数据源没有启用 relay log 时，DM-worker 向 DM-master 保持存活的周期，单位为秒。默认是 60 秒。
relay- ↪ keepalive ↪ -ttl	当绑定的上游数据源启用 relay log 时，DM-worker 向 DM-master 保持存活的周期，单位为秒。默认是 1800 秒。在版本 2.0.2 新增。
ssl-ca	DM-worker 组件用于与其它组件连接的 SSL CA 证书所在的路径
ssl-cert	DM-worker 组件用于与其它组件连接的 PEM 格式的 X509 证书所在的路径
ssl-key	DM-worker 组件用于与其它组件连接的 PEM 格式的 X509 密钥所在的路径
cert-allowed ↪ -cn	证书检查 Common Name 列表

8.3.4 上游数据库配置文件介绍

本文介绍上游数据库的配置文件，包括配置文件示例与配置项说明。

8.3.4.1 配置文件示例

上游数据库的示例配置文件如下所示：

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

### 是否开启 GTID
enable-gtid: false

### 是否开启 relay log
enable-relay: false    # 该配置项从 DM v2.0.2 版本起弃用，使用 `start-relay`
   ↪ 命令开启 relay log
relay-binlog-name: ""  # 拉取上游 binlog 的起始文件名
relay-binlog-gtid: ""  # 拉取上游 binlog 的起始 GTID
relay-dir: "relay-dir" # 存储 relay log 的目录，默认值为 "relay-dir"

from:
  host: "127.0.0.1"
  port: 3306
  user: "root"
  password: "ZqMLjZ2j5khNelDEfDoUhkD5aV5fIJOe0fiog9w=" # 推荐使用 dmctl
     ↪ 对上游数据库的用户密码加密之后的密码
  security:                    # 上游数据库 TLS 相关配置
```

```
    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    # backoff 的最大值，不能小于 1s

### 从 DM v2.0.2 开始，Binlog event filter
    ↪ 也可以在上游数据库配置文件中进行配置
### case-sensitive: false
### filters:
### - schema-pattern: dmctl
###   table-pattern: t_1
###   events: []
###   sql-pattern:
###   - alter table .* add column `aaa` int
###   action: Ignore
```

注意：

在 DM v2.0.1 版本中，请勿同时配置 `enable-gtid` 与 `enable-relay` 为 `true`，否则可能引发增量数据丢失问题。

8.3.4.2 配置项说明

8.3.4.2.1 Global 配置

配置项	说明
`source-id`	标识一个 MySQL 实例。
`enable-gtid`	是否使用 GTID 方式从上游拉取 binlog，默认值为 false。一般情况下不需要手动配置，如果上游数据库启用了 GTID 支持，且需要做主从切换，则将该配置项设置为 true。

配置项	说明
enable-relay	是否开启 relay log，默认值为 false。自 DM v2.0.2 版本起，该配置项弃用，需使用 start-relay 命令开启 relay log。
relay-binlog-name	拉取上游 binlog 的起始文件名，例如 “mysql-bin.000002”，该配置在 enable-gtid 为 false 的情况下生效。如果不配置该项，DM-worker 将从最新时间点的 binlog 文件开始拉取 binlog，一般情况下不需要手动配置。
relay-binlog-gtid	拉取上游 binlog 的起始 GTID，例如 “e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849”，该配置在 enable-gtid 为 true 的情况下生效。如果不配置该项，DM-worker 将从最新时间点的 binlog GTID 开始拉取 binlog，一般情况下不需要手动配置。
relay-dir	存储 relay log 的目录，默认值为 “./relay_log”。
host	上游数据库的 host。
port	上游数据库的端口。
user	上游数据库使用的用户名。
password	上游数据库的用户密码。注意：推荐使用 dmctl 加密后的密码。
security	上游数据库 TLS 相关配置。配置的证书文件路径需能被所有节点访问。若配置为本地路径，则集群所有节点需要将证书文件拷贝一份放在各节点机器相同的路径位置上。

8.3.4.2.2 relay log 清理策略配置（purge 配置项）

一般情况下不需要手动配置，如果 relay log 数据量较大，磁盘空间不足，则可以通过设置该配置项来避免 relay log 写满磁盘。

配置项	说明
interval	定期检查 relay log 是否过期的间隔时间，默认值：3600，单位：秒。
expires	relay log 的过期时间，默认值为 0，单位：小时。未由 relay 处理单元进行写入、或已有数据迁移任务当前或未来不需要读取的 relay log 在超过过期时间后会被 DM 删除。如果不设置则 DM 不会自动清理过期的 relay log。

配置项	说明
remain-space	设置最小的可用磁盘空间。当磁盘可用空间小于这个值时，DM-worker 会尝试删除 relay log，默认值：15，单位：GB。

> 注意：
>
> 仅在 interval 不为 0 且 expires 和 remain-space 两个配置项中至少有一个不为 0 的情况下 DM 的自动清理策略才会生效。

8.3.4.2.3 任务状态检查配置（checker 配置项）

DM 会定期检查当前任务状态以及错误信息，判断恢复任务能否消除错误，并自动尝试恢复任务进行重试。DM 会使用指数回退策略调整检查间隔。这些行为可以通过如下配置进行调整：

配置项	说明
check-enable	启用自动重试功能。
backoff-rollback	如果指数回退策略的间隔大于该值，且任务处于正常状态，尝试减小间隔。
backoff-max	指数回退策略的间隔的最大值，该值必须大于 1 秒。

8.3.4.2.4 Binlog event filter

从 DM v2.0.2 开始，Binlog event filter 也可以在上游数据库配置文件中进行配置。

配置项	说明
case-sensitive	Binlog event filter 标识符是否大小写敏感。默认值：false。
filters	配置 Binlog event filter，含义见Binlog event filter 参数解释。

8.4 安全

8.4.1 为 DM 的连接开启加密传输

本文介绍如何为 DM 的连接开启加密传输，包括 DM-master，DM-worker，dmctl 组件之间的连接以及 DM 组件与上下游数据库之间的连接。

8.4.1.1 为 DM-master，DM-worker，dmctl 组件之间的连接开启加密传输

本节介绍如何为 DM-master，DM-worker，dmctl 组件之间的连接开启加密传输。

8.4.1.1.1 配置开启加密传输

1. 准备证书。

   推荐为 DM-master、DM-worker 分别准备一个 Server 证书，并保证可以相互验证，而 dmctl 工具则可选择共用 Client 证书。

   有多种工具可以生成自签名证书，如 `openssl`，`cfssl` 及 `easy-rsa` 等基于 `openssl` 的工具。

   这里提供一个使用 `openssl` 生成证书的示例：生成自签名证书。

2. 配置证书。

> **注意：**
> DM-master、DM-worker 与 dmctl 三个组件可使用同一套证书。

- DM-master

  在 DM-master 配置文件或命令行参数中设置：

```
ssl-ca = "/path/to/ca.pem"
ssl-cert = "/path/to/master-cert.pem"
ssl-key = "/path/to/master-key.pem"
```

- DM-worker

  在 DM-worker 配置文件或命令行参数中设置：

```
ssl-ca = "/path/to/ca.pem"
ssl-cert = "/path/to/worker-cert.pem"
ssl-key = "/path/to/worker-key.pem"
```

- dmctl

  若 DM 集群各个组件间开启加密传输后，在使用 dmctl 工具连接集群时，需要指定 Client 证书，示例如下：

```
./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
```

8.4.1.1.2 认证组件调用者身份

通常被调用者除了校验调用者提供的密钥、证书和 CA 有效性外，还需要校验调用者身份以防止拥有有效证书的非法访问者进行访问（例如：DM-worker 只能被 DM-master 访问，需阻止拥有合法证书但非 DM-master 的其他访问者访问 DM-worker）。

如希望进行组件调用者身份认证，需要在生成证书时通过 `Common Name` (CN) 标识证书使用者身份，并在被调用者配置检查证书 `Common Name` 列表时检查调用者身份。

- DM-master

  在 config 文件或命令行参数中设置：

```
cert-allowed-cn = ["dm"]
```

- DM-worker

  在 config 文件或命令行参数中设置：

```
cert-allowed-cn = ["dm"]
```

8.4.1.1.3 证书重加载

DM-master、DM-worker 和 dmctl 都会在每次新建相互通讯的连接时重新读取当前的证书和密钥文件内容，实现证书和密钥的重加载。

当 ssl-ca、ssl-cert 或 ssl-key 的文件内容更新后，可通过重启 DM 组件使其重新加载证书与密钥内容并重新建立连接。

8.4.1.2 DM 组件与上下游数据库之间的连接开启加密传输

本节介绍如何为 DM 组件与上下游数据库之间的连接开启加密传输。

8.4.1.2.1 为上游数据库连接开启加密传输

1. 配置上游数据库，启用加密连接支持并设置 Server 证书，具体可参考 Using encrypted connections
2. 在 source 配置文件中设置 MySQL Client 证书：

> **注意：**
> 请确保所有 DM-master 与 DM-worker 组件能通过指定路径读取到证书与密钥文件的内容。

```
from:
    security:
        ssl-ca: "/path/to/mysql-ca.pem"
        ssl-cert: "/path/to/mysql-client-cert.pem"
        ssl-key: "/path/to/mysql-client-key.pem"
```

8.4.1.2.2 为下游 TiDB 连接开启加密传输

1. 配置下游 TiDB 启用加密连接支持，具体可参考 配置 TiDB 启用加密连接支持
2. 在 task 配置文件中设置 TiDB Client 证书：

> **注意：**
> 请确保所有 DM-master 与 DM-worker 组件能通过指定路径读取到证书与密钥文件的内容。

```
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"
```

8.4.2 生成自签名证书

本文档提供使用 openssl 生成自签名证书的一个示例，用户也可以根据自己的需求生成符合要求的证书和密钥。

假设实例集群拓扑如下：

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

8.4.2.1 安装 OpenSSL

对于 Debian 或 Ubuntu 操作系统：

```
apt install openssl
```

对于 RedHat 或 CentOS 操作系统：

```
yum install openssl
```

也可以参考 OpenSSL 官方的下载文档进行安装。

8.4.2.2 生成 CA 证书

CA 的作用是签发证书。实际情况中，请联系你的管理员签发证书或者使用信任的 CA 机构。CA 会管理多个证书对，这里只需生成原始的一对证书，步骤如下：

1. 生成 CA 密钥：

```
openssl genrsa -out ca-key.pem 4096
```

2. 生成 CA 证书：

```
openssl req -new -x509 -days 1000 -key ca-key.pem -out ca.pem
```

3. 验证 CA 证书：

```
openssl x509 -text -in ca.pem -noout
```

8.4.2.3 签发各个组件的证书

8.4.2.3.1 集群中可能使用到的证书

- master certificate 由 DM-master 使用，为其他组件验证 DM-master 身份。
- worker certificate 由 DM-worker 使用，为其他组件验证 DM-worker 身份。
- client certificate 由 dmctl 使用，用于 DM-master、DM-worker 验证客户端。

8.4.2.3.2 为 DM-master 签发证书

给 DM-master 实例签发证书的步骤如下：

1. 生成该证书对应的私钥：

```
openssl genrsa -out master-key.pem 2048
```

2. 拷贝一份 OpenSSL 的配置模板文件。

   模板文件可能存在多个位置，请以实际位置为准：

```
cp /usr/lib/ssl/openssl.cnf .
```

   如果不知道实际位置，请在根目录下查找：

```
find / -name openssl.cnf
```

3. 编辑 `openssl.cnf`，在 `[ req ]` 字段下加入 `req_extensions = v3_req`，然后在 `[ v3_req ]` 字段下加入 `subjectAltName = @alt_names`。最后新建一个字段，根据前述的集群拓扑编辑 `Subject Alternative Name` (SAN) 的信息：

```
[ 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
```

目前支持以下 SAN 检查项：

- IP
- DNS
- URI

> **注意：**
> 如果要使用 0.0.0.0 等特殊 IP 用于连接通讯，也需要将其加入到 alt_names 中。

4. 保存 openssl.cnf 文件后，生成证书请求文件（在这一步中提供 Common Name (e.g. server FQDN or YOUR name)[]：输入时，可以为该证书指定 Common Name (CN)，如 dm。其作用是让服务端验证接入的客户端的身份，各个组件默认不会开启验证，需要在配置文件中启用该功能才生效）：

```
openssl req -new -key master-key.pem -out master-cert.pem -config
    ↪ openssl.cnf
```

5. 签发生成证书：

```
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. 验证证书携带 SAN 字段信息（可选）：

```
openssl x509 -text -in master-cert.pem -noout
```

7. 确认在当前目录下得到如下文件：

```
ca.pem
master-cert.pem
master-key.pem
```

> **注意：**
> 为 DM-worker 组件签发证书的过程类似，此文档不再赘述。

8.4.2.3.3 为 dmctl 签发证书

为客户端签发证书的步骤如下。

1. 生成该证书对应的私钥：

```
openssl genrsa -out client-key.pem 2048
```

2. 生成证书请求文件（在这一步也可以为该证书指定 Common Name，其作用是让服务端验证接入的客户端的身份，默认不会开启对各个组件的验证，需要在配置文件中启用该功能才生效）

```
openssl req -new -key client-key.pem -out client-cert.pem
```

3. 签发生成证书：

```
openssl x509 -req -days 365 -CA ca.pem -CAkey ca-key.pem -
    ↪ CAcreateserial -in client-cert.pem -out client-cert.pem
```

8.5 DM 监控指标

使用 TiUP 部署 DM 集群的时候，会默认部署一套监控系统。

8.5.1 Task

在 Grafana dashboard 中，DM 默认名称为 `DM-task`。

8.5.1.1 Overview

overview 下包含运行当前选定 task 的所有 DM-worker/master instance/source 的部分监控指标。当前默认告警规则只针对于单个 DM-worker/master instance/source。

metric 名称	说明	告警说明	告警级别
task state	迁移子任务的状态	N/A	N/A
storage capacity	relay log 占有的磁盘的总容量	N/A	N/A
storage remain	relay log 占有的磁盘的剩余可用容量	N/A	N/A

metric 名称	说明	告警说明	告警级别
binlog file gap between master and relay	relay 与上游 master 相比落后的 binlog file 个数	N/A	N/A
load progress	load unit 导入过程的进度百分比，值变化范围为：0% - 100%	N/A	N/A
binlog file gap between master and syncer	与上游 master 相比 binlog replication unit 落后的 binlog file 个数	N/A	N/A
shard lock resolving	当前子任务是否正在等待 shard DDL 迁移，大于 0 表示正在等待迁移	N/A	N/A

8.5.1.2 Operate error

metric 名称	说明	告警说明	告警级别
before any operate error	在进行操作之前出错的次数	N/A	N/A
source bound error	数据源绑定操作出错次数	N/A	N/A
start error	子任务启动的出错次数	N/A	N/A
pause error	子任务暂停的出错次数	N/A	N/A
resume error	子任务恢复的出错次数	N/A	N/A
auto-resume error	子任务自动恢复的出错次数	N/A	N/A
update error	子任务更新的出错次数	N/A	N/A

metric 名称	说明	告警说明	告警级别
stop error	子任务停止的出错次数	N/A	N/A

8.5.1.3 HA 高可用

metric 名称	说明	告警说明	告警级别
number of dm-masters start leader components per minute	每分钟内 DM-master 尝试启用 leader 相关组件次数	N/A	N/A
number of workers in different state	不同状态下有多少个 DM-worker	存在离线的 DM-worker 超过一小时	critical
workers' state	DM-worker 的状态	N/A	N/A
number of worker event error	不同类型的 DM-worker 错误出现次数	N/A	N/A
shard ddl error per minute	每分钟内不同类型的 shard DDL 错误次数	发生 shard DDL 错误	critical

metric 名称	说明	告警说明	告警级别
number of pending shard ddl	未完成的 shard DDL 数目	存在未完成的 shard DDL 数目超过一小时	critical

8.5.1.4 Task 状态

metric 名称	说明	告警说明	告警级别
task state	迁移子任务的状态	当子任务状态处于 Paused 超过 20 分钟时	critical

8.5.1.5 Dump/Load unit

下面 metrics 仅在 task-mode 为 full 或者 all 模式下会有值。

metric 名称	说明
load progress	load unit 导入过程的进度百分比，值变化范围为：0% - 100%
data file size	load unit 导入的全量数据中数据文件（内含 INSERT INTO 语句）的总大
dump process exits with error	dump unit 在 DM-worker 内部遇到错误并且退出了
load process exits with error	load unit 在 DM-worker 内部遇到错误并且退出了
table count	load unit 导入的全量数据中 table 的数量总和
data file count	load unit 导入的全量数据中数据文件（内含 INSERT INTO 语句）的数量
transaction execution latency	load unit 在执行事务的时延，单位：秒
statement execution latency	load unit 执行语句的耗时，单位：秒
remaining time	load unit 完成同步的剩余时间，单位：秒

8.5.1.6 Binlog replication

下面 metrics 仅在 task-mode 为 incremental 或者 all 模式下会有值。

metric 名称	说明	告警说明	告警级别
remaining_time_to_sync	预计 Syncer 还需要多少分钟可以和上游 master 完全同步，单位：分钟	N/A	N/A
replicate_lag_gauge	上游 master 到下游的 binlog 复制延迟时间，单位：秒	N/A	N/A
replicate_lag_histogram	上游 master 到下游的 binlog 复制延迟分布，单位：秒。注意由于统计机制不同，数据会有误差	N/A	N/A
process_exist_with_error	binlog replication unit 在 DM-worker 内部遇到错误并且退出了	立即告警	critical
binlog_file_gap_between_master_and_syncer	与上游 master 相比落后的 binlog file 个数	落后 binlog file 个数超过 1 个 (不含 1 个) 且持续 10 分钟时	critical

metric 名称	说明	告警说明	告警级别
binlog file gap between relay and syncer	与 relay 相比落后的 binlog file 个数	落后 binlog file 个数超过 1 个 (不含 1 个) 且持续 10 分钟时	critical
binlog event QPS	单位时间内接收到的 binlog event 数量 (不包含需要跳过的 event)	N/A	N/A
skipped binlog event QPS	单位时间内接收到的需要跳过的 binlog event 数量	N/A	N/A
read binlog event duration	binlog replication unit 从 relay log 或上游 MySQL 读取 binlog 的耗时，单位：秒	N/A	N/A
transform binlog event duration	binlog replication unit 解析 binlog 并将 binlog 转换成 SQL 语句的耗时，单位：秒	N/A	N/A

metric 名称	说明	告警说明	告警级别
dispatch binlog event dura-tion	binlog replication unit 调度一条 binlog event 的耗时，单位：秒	N/A	N/A
transaction execu-tion la-tency	binlog replication unit 执行事务到下游的耗时，单位：秒	N/A	N/A
binlog event size	binlog replication unit 从 relay log 或上游 MySQL 读取的单条 binlog event 的大小	N/A	N/A
DML queue re-main length	剩余 DML job 队列的长度	N/A	N/A
total sqls jobs	单位时间内新增的 job 数量	N/A	N/A
finished sqls jobs	单位时间内完成的 job 数量	N/A	N/A
statement execu-tion la-tency	binlog replication unit 执行语句到下游的耗时，单位：秒	N/A	N/A

metric 名称	说明	告警说明	告警级别
add job duration	binlog replication unit 增加一条 job 到队列的耗时，单位：秒	N/A	N/A
DML conflict detect duration	binlog replication unit 检测 DML 间冲突的耗时，单位：秒	N/A	N/A
skipped event duration	binlog replication unit 跳过 binlog event 的耗时，单位：秒	N/A	N/A
unsynced tables	当前子任务内还未收到 shard DDL 的分表数量	N/A	N/A
shard lock resolving	当前子任务是否正在等待 shard DDL 迁移，大于 0 表示正在等待迁移	N/A	N/A
ideal QPS	在 DM 运行耗时为 0 时可以达到的最高 QPS	N/A	N/A
binlog event row	一个 binlog 事件中的行数	N/A	N/A
finished transaction total	执行完毕的事务数量	N/A	N/A

metric 名称	说明	告警说明	告警级别
replication transaction batch	执行到下游的事务里中 sql 行数	N/A	N/A
flush checkpoints time interval	检查点刷新时间间隔，单位：秒	N/A	N/A

8.5.1.7 Relay log

metric 名称	说明	告警说明	告警级别
storage capacity	relay log 占有的磁盘的总容量	N/A	N/A
storage remain	relay log 占有的磁盘的剩余可用容量	小于 10G 的时候需要告警	critical
process exits with error	relay log 在 DM-worker 内部遇到错误并且退出了	立即告警	critical
relay log data corruption	relay log 文件损坏的个数	立即告警	emergency
fail to read binlog from master	relay 从上游的 MySQL 读取 binlog 时遇到的错误数	立即告警	critical

metric 名称	说明	告警说明	告警级别
fail to write relay log	relay 写 binlog 到磁盘时遇到的错误数	立即告警	critical
binlog file index	relay log 最大的文件序列号。如 value = 1 表示 relay-log.000001	N/A	N/A
binlog file gap between master and relay	relay 与上游 master 相比落后的 binlog file 个数	落后 binlog file 个数超过 1 个 (不含 1 个) 且持续 10 分钟时	critical
binlog pos	relay log 最新文件的写入 offset	N/A	N/A
read binlog event duration	relay log 从上游的 MySQL 读取 binlog 的时延，单位：秒	N/A	N/A
write relay log duration	relay log 每次写 binlog 到磁盘的时延，单位：秒	N/A	N/A
binlog event size	relay log 写到磁盘的单条 binlog 的大小	N/A	N/A

8.5.2 Instance

在 Grafana dashboard 中，instance 的默认名称为 `DM-instance`。

8.5.2.1 Relay log

metric 名称	说明	告警说明	告警级别
storage capacity	relay log 占有的磁盘的总容量	N/A	N/A
storage remain	relay log 占有的磁盘的剩余可用容量	小于 10G 的时候需要告警	critical
process exits with error	relay log 在 DM-worker 内部遇到错误并且退出了	立即告警	critical
relay log data corruption	relay log 文件损坏的个数	立即告警	emergency
fail to read binlog from master	relay 从上游的 MySQL 读取 binlog 时遇到的错误数	立即告警	critical
fail to write relay log	relay 写 binlog 到磁盘时遇到的错误数	立即告警	critical
binlog file index	relay log 最大的文件序列号。如 value = 1 表示 relay-log.000001	N/A	N/A

metric 名称	说明	告警说明	告警级别
binlog file gap between master and relay	relay 与上游 master 相比落后的 binlog file 个数	落后 binlog file 个数超过 1 个（不含 1 个）且持续 10 分钟时	critical
binlog pos	relay log 最新文件的写入 offset	N/A	N/A
read binlog duration	relay log 从上游的 MySQL 读取 binlog 的时延，单位：秒	N/A	N/A
write relay log duration	relay log 每次写 binlog 到磁盘的时延，单位：秒	N/A	N/A
binlog size	relay log 写到磁盘的单条 binlog 的大小	N/A	N/A

8.5.2.2 task

metric 名称	说明	告警说明	告警级别

metric 名称	说明	告警说明	告警级别
task state	迁移子任务的状态	当子任务状态处于 paused 超过 10 分钟时	critical
load progress	load unit 导入过程的进度百分比，值变化范围为：0% - 100%	N/A	N/A
binlog file gap between master and syncer	与上游 master 相比 binlog replication unit 落后的 binlog file 个数	N/A	N/A
shard lock resolving	当前子任务是否正在等待 shard DDL 迁移，大于 0 表示正在等待迁移	N/A	N/A

8.6 DM 告警信息

使用 TiUP 部署 DM 集群的时候，会默认部署一套告警系统。

DM 的告警规则及其对应的处理方法可参考告警处理。

DM 的告警信息与监控指标均基于 Prometheus，告警规则与监控指标的对应关系可参考DM 监控指标。

9 Data Migration 常见问题

9.1 DM 是否支持迁移阿里 RDS 以及其他云数据库的数据？

DM 仅支持解析标准版本的 MySQL/MariaDB 的 binlog，对于阿里云 RDS 以及其他云数据库没有进行过测试，如果确认其 binlog 为标准格式，则可以支持。

已知问题的兼容情况：

- 阿里云 RDS
  - 即使上游表没有主键，阿里云 RDS 的 binlog 中也会包含隐藏的主键列，与上游表结构不一致。
- 华为云 RDS
  - 不支持，详见：华为云数据库 RDS 是否支持直接读取 Binlog 备份文件。

9.2 task 配置中的黑白名单的正则表达式是否支持非获取匹配 (?!)?

目前不支持，DM 仅支持 golang 标准库的正则，可以通过 re2-syntax 了解 golang 支持的正则表达式。

9.3 如果在上游执行的一个 statement 包含多个 DDL 操作，DM 是否支持迁移？

DM 会尝试将包含多个 DDL 变更操作的单条语句拆分成只包含一个 DDL 操作的多条语句，但是可能没有覆盖所有的场景。建议在上游执行的一条 statement 中只包含一个 DDL 操作，或者在测试环境中验证一下，如果不支持，可以给 DM 提 issue。

9.4 如何处理不兼容的 DDL 语句？

你需要使用 dmctl 手动处理 TiDB 不兼容的 DDL 语句 (包括手动跳过该 DDL 语句或使用用户指定的 DDL 语句替换原 DDL 语句，详见处理出错的 DDL 语句)。

> **注意：**
>
> TiDB 目前并不兼容 MySQL 支持的所有 DDL 语句。

9.5 如何重置数据迁移任务？

当数据迁移过程中发生异常且无法恢复时，需要重置数据迁移任务，对数据重新进行迁移：

1. 使用 `stop-task` 停止异常的数据迁移任务。
2. 清理下游已迁移的数据。
3. 从下面两种方式中选择其中一种重启数据迁移任务：
    - 修改任务配置文件以指定新的任务名，然后使用 `start-task {task-config-file}` 重启迁移任务。
    - 使用 `start-task --remove-meta {task-config-file}` 重启数据迁移任务。

9.6 设置了 online-ddl-scheme: "gh-ost"，gh-ost 表相关的 DDL 报错该如何处理？

```
[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
    ↪ ......
```

出现上述错误可能有以下原因：

DM 在最后 `rename ghost_table to origin table` 的步骤会把内存的 DDL 信息读出，并且还原为 origin table 的 DDL。而内存中的 DDL 信息是在 `alter ghost_table` 的时候进行处理，记录 ghost_table DDL 的信息；或者是在重启 dm-worker 启动 task 的时候，从 `dm_meta.{task_name}_onlineddl` 中读取出来。

因此，如果在增量复制过程中，指定的 Pos 跳过了 `alter ghost_table` 的 DDL，但是该 Pos 仍在 gh-ost 的 online-ddl 的过程中，就会因为 ghost_table 没有正确写入到内存以及 `dm_meta.{task_name}_onlineddl`，而导致该问题。

可以通过以下方式绕过这个问题：

1. 取消 task 的 `online-ddl-schema` 的配置。
2. 把 `_{table_name}_gho`、`_{table_name}_ghc`、`_{table_name}_del` 配置到 `block-allow-list.ignore-tables` 中。
3. 手工在下游的 TiDB 执行上游的 DDL。
4. 待 Pos 复制到 gh-ost 整体流程后的位置，再重新启用 `online-ddl-schema` 以及注释掉 `block-allow-list.ignore-tables`。

9.7 如何为已有迁移任务增加需要迁移的表？

假如已有数据迁移任务正在运行，但又有其他的表需要添加到该迁移任务中，可根据当前数据迁移任务所处的阶段按下列方式分别进行处理。

> **注意：**
>
> 向已有数据迁移任务中增加需要迁移的表操作较复杂，请仅在确有强烈需求时进行。

9.7.1 迁移任务当前处于 Dump 阶段

由于 MySQL 不支持指定 snapshot 来进行导出，因此在导出过程中不支持更新迁移任务并重启以通过断点继续导出，故无法支持在该阶段动态增加需要迁移的表。

如果确实需要增加其他的表用于迁移，建议直接使用新的配置文件重新启动迁移任务。

9.7.2 迁移任务当前处于 Load 阶段

多个不同的数据迁移任务在导出时，通常对应于不同的 binlog position，如将它们在 Load 阶段合并导入，则无法就 binlog position 达成一致，因此不建议在 Load 阶段向数据迁移任务中增加需要迁移的表。

9.7.3 迁移任务当前处于 Sync 阶段

当数据迁移任务已经处于 Sync 阶段时，在配置文件中增加额外的表并重启任务，DM 并不会为新增的表重新执行全量导出与导入，而是会继续从之前的断点进行增量复制。

因此，如果需要新增的表对应的全量数据尚未导入到下游，则需要先使用单独的数据迁移任务将其全量数据导出并导入到下游。

将已有迁移任务对应的全局 checkpoint （`is_global=1`）中的 position 信息记为 `checkpoint-T`，如 `(mysql-bin.000100, 1234)`。将需要增加到迁移任务的表在全量导出的 `metedata`（或另一个处于 Sync 阶段的数据迁移任务的 checkpoint）的 position 信息记为 `checkpoint-S`，如 `(mysql-bin.000099, 5678)`。则可通过以下步骤将表增加到迁移任务中：

1. 使用 `stop-task` 停止已有迁移任务。如果需要增加的表属于另一个运行中的迁移任务，则也将其停止。
2. 使用 MySQL 客户连接到下游 TiDB 数据库，手动更新已有迁移任务对应的 checkpoint 表中的信息为 `checkpoint-T` 与 `checkpoint-S` 中的较小值（在本例中，为 `(mysql-bin.000099, 5678)`）。

- 需要更新的 checkpoint 表为 {dm_meta} 库中的 {task-name}_syncer_checkpoint ↪ 。
- 需要更新的 checkpoint 行为 id={source-id} 且 is_global=1。
- 需要更新的 checkpoint 列为 binlog_name 与 binlog_pos。

3. 在迁移任务配置中为 syncers 部分设置 safe-mode: true 以保证可重入执行。
4. 通过 start-task 启动迁移任务。
5. 通过 query-status 观察迁移任务状态，当 syncerBinlog 超过 checkpoint-T 与 checkpoint-S 中的较大值后（在本例中，为 (mysql-bin.000100, 1234)），即可还原 safe-mode 为原始值并重启迁移任务。

9.8 全量导入过程中遇到报错 packet for query is too large. Try adjusting the 'max_allowed_packet' variable

尝试将

- TiDB Server 的全局变量 max_allowed_packet
- 任务配置文件中的配置项 target-database.max-allowed-packet（详情参见 DM 任务完整配置文件介绍）

设置为比默认 67108864 (64M) 更大的值。详见 Loader 解决方案。

9.9 2.0+ 集群运行 1.0 已有数据迁移任务时报错 Error 1054: Unknown column 'binlog_gtid' in 'field list'

在 DM 2.0 之后，为 checkpoint 等元信息表引入了更多的字段。如果通过 start-task 直接使用 1.0 集群的任务配置文件从增量复制阶段继续运行，则会出现 Error 1054: ↪ Unknown column 'binlog_gtid' in 'field list' 错误。

对于此错误，可手动将 DM 1.0 的数据迁移任务导入到 2.0+ 集群。

9.10 TiUP 无法部署 DM 的某个版本（如 v2.0.0-hotfix）

你可以通过 tiup list dm-master 命令查看 TiUP 支持部署的 DM 版本。该命令未展示的版本不能由 TiUP 管理。

9.11 DM 同步报错 parse mydumper metadata error: EOF

该错误需要查看报错信息以及日志进一步分析。报错原因可能是 dump 单元由于缺少权限没有产生正确的 metadata 文件。

9.12 DM 分库分表同步中没有明显报错，但是下游数据丢失

需要检查配置项 block-allow-list 和 table-route：

- block-allow-list 填写的是上游数据库表，可以在 do-tables 前通过加 “~” 来进行正则匹配。
- table-route 不支持正则，采用的是通配符模式，所以 table_parttern_[0-63] 只会匹配 table_parttern_0 到 table_pattern_6 这 7 张表。

9.13 DM 上游无写入，replicate lag 监控无数据

在 DM v1.0 中，需要开启 enable-heartbeat 才会产生该监控数据。v2.0 及以后版本中，尚未启用该功能，replicate lag 监控无数据是预期行为。

9.14 DM v2.0.0 启动任务时出现 fail to initial unit Sync of subtask，报错信息的 RawCause 显示 context deadline exceeded

该问题是 DM v2.0.0 的已知问题，在同步任务的表数目较多时触发，将在 v2.0.1 修复。使用 TiUP 部署的用户可以升级到开发版 nightly 解决该问题，或者访问 GitHub 上 DM 仓库的 release 页面下载 v2.0.0-hotfix 版本手动替换可执行文件。

9.15 DM 同步中报错 duplicate entry

用户需要首先确认任务中没有配置 disable-detect（v2.0.7 及之前版本），没有其他同步程序或手动插入数据，表中没有配置相关的 DML 过滤器。

为了便于排查问题，用户收集到下游 TiDB 相关 general log 后可以在 AskTUG 社区联系专家进行排查。收集 general log 的方式如下：

```
# 开启 general log
curl -X POST -d "tidb_general_log=1" http://{TiDBIP}:10080/settings
# 关闭 general log
curl -X POST -d "tidb_general_log=0" http://{TiDBIP}:10080/settings
```

在发生 duplicate entry 报错时，确认日志中包含冲突数据的记录。

9.16 监控中部分面板显示 No data point

请参照DM 监控指标查看各面板含义，部分面板没有数据是正常现象。例如没有发生错误、不存在 DDL lock、没有启用 relay 功能等情况，均可能使得对应面板没有数据。

9.17 DM v1.0 在任务出错时使用 sql-skip 命令无法跳过某些语句

首先需要检查执行 sql-skip 之后 binlog 位置是否在推进，如果是的话表示 sql-skip 已经生效。重复出错的原因是上游发送了多个不支持的 DDL，可以通过 sql-skip -s <sql-pattern> 进行模式匹配。

对于类似下面这种报错（报错中包含 parse statement）：

```
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
```

出现报错的原因是 TiDB parser 无法解析上游的 DDL，例如 ALTER EVENT，所以 sql-skip 不会按预期生效。可以在任务配置文件中添加Binlog 过滤规则进行过滤，并设置 schema-pattern: "*"。从 DM 2.0.1 版本开始，已预设过滤了 EVENT 相关语句。

在 DM v2.0 版本之后 sql-skip 已经被 handle-error 替代，handle-error 可以跳过该类错误。

9.18 DM 同步时下游长时间出现 REPLACE 语句

请检查是否符合safe mode 触发条件。如果任务发生错误并自动恢复，或者发生高可用调度，会满足“启动或恢复任务的前 1 分钟”这一条件，因此启用 safe mode。

可以检查 DM-worker 日志，在其中搜索包含 change count 的行，该行的 new count 非零时会启用 safe mode。检查 safe mode 启用时间以及启用前是否有报错，以定位启用原因。

9.19 使用 DM v2.0 同步数据时重启 DM 进程，出现全量数据导入失败错误

在 DM v2.0.1 及更早版本中，如果全量导入操作未完成时发生重启，重启后的上游数据源与 DM worker 的绑定关系可能会发生变化。例如，可能出现 dump 单元的中间数据在 DM worker A 机器上，但却由 DM worker B 进行 load 单元的情况，进而导致操作失败。

该情况有两种解决方案：

- 如果数据量较小（TB 级以下）或任务有合库合表：清空下游数据库的已导入数据，同时清空导出数据目录，使用 dmctl 删除并 start-task --remove-meta 重建任务。后续尽量保证全量导出导入阶段 DM 没有冗余 worker 以及避免在该时段内重启或升级 DM 集群。
- 如果数据量较大（数 TB 或更多）：清空下游数据库的已导入数据，将 lightning 部署到数据所在的 DM worker 节点，使用 lightning local backend 模式 导入 DM dump 单元导出的数据。全量导入完成后，修改任务的 task-mode 为 incremental，修改 mysql-instance.meta.pos 为 dump 单元导出数据 metadata 中记录的位置，启动一个增量任务。

9.20 使用 DM 同步数据时重启 DM 进程，增量任务出现 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. 错误

该错误表明全量迁移期间，dump 单元记录 metadata 中的 binlog 位置已经被上游清理。

解决方案：出现该问题时只能清空下游数据库已同步数据，并在停止任务后加上 `--remove-meta` 参数重建任务。

如要提前避免该问题，需要进行以下配置：

1. 在 DM 全量迁移未完成时调大上游 MySQL 的 `expire_logs_days` 变量，保证全量进行结束时 metadata 中的 binlog 位置到当前时间的 binlog 都还没有被清理掉。如果数据量较大，应该使用 dumpling + lightning 的方式加快全量迁移速度。
2. DM 任务开启 relay log 选项，保证 binlog 被清理后 DM 仍有 relay log 可读取。

9.21 使用 TiUP v1.3.0, v1.3.1 部署 DM 集群，DM 集群的 grafana 监控报错显示 failed to fetch dashboard

该问题为 TiUP 已知 bug，在 TiUP v1.3.2 中已进行修复。可采取以下任一方法解决：

- 方法一：使用 `tiup update --self && tiup update dm` 升级 TiUP 到更新版本，随后先缩容再扩容集群中的 grafana 节点，重建 grafana 服务。
- 方法二：
    1. 备份 `deploy/grafana-$port/bin/public` 文件夹。
    2. 下载 TiUP DM 离线镜像包，并进行解压，将其中的 grafana-v4.0.3-**.tar.gz 文件解压后，用解压出的 public/ 文件夹替换前面所描述的文件夹，运行 `tiup dm restart $cluster_name -R grafana` 重启 grafana 服务监控。

9.22 在 DM v2.0 中，同时开启 relay 与 gtid 同步 MySQL 时，运行 query-status 发现 syncer checkpoint 中 GTID 不连续

该问题为 DM 已知 bug，在完全满足以下两个条件时将会触发，DM 将在 v2.0.2 修复该问题：

1. DM 配置的 source 同时设置了 `enable-relay` 与 `enable-gtid` 为 `true`
2. DM 同步上游为 MySQL 从库，并且该从库通过 `show binlog events in '<newest-binlog>' limit 2` 查询出的 `previous_gtids` 区间不连续，例如：

```
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                  |
+------------------+------+---------------+----------+-------------+-------------------
   ↪
```

使用 dmctl 的 query-status <task> 指令查询任务信息，如果已经出现 subTaskStatus ↪ .sync.syncerBinlogGtid 不连续但 subTaskStatus.sync.masterBinlogGtid 连续的情况，例如下述例子：

```
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"
                }
            }
        ]
    }
]
}
```

其中 mysql1 的 `syncerBinlogGtid` 不连续，已有数据丢失需要按下述方案之一处理：

- 如果全量导出任务 metadata 中的 position 到当前时间的上游数据库的 binlog 仍未被清理：

    1. 停止当前任务并删除所有 GTID 不连续的 source

2. 设置所有 source 的 `enable-relay` 为 `false`
3. 针对 GTID 不连续的 source（上例 mysql1），在对应的任务配置文件 `task.yaml` 中，把 `task-mode` 修改为 `incremental` 并配置增量任务起始点 `mysql-instances.meta` 为各个全量导出任务 metadata 的 binlog name，position 和 gtid 信息
4. 配置 `task.yaml` 中的 `syncers.safe-mode` 为 `true` 并重启任务
5. 待增量同步追上后，停止任务并在任务配置文件中设置 `safe-mode` 为 `false`
6. 再次重启任务

- 如果上游数据库 binlog 已被清理但是本地 relay log 仍未被清理：

1. 停止当前任务
2. 针对 GTID 不连续的 source（上例 mysql1），在对应的任务配置文件 `task.yaml` 中，把 `task-mode` 修改为 `incremental` 并配置增量任务起始点 `mysql-instances.meta` 为各个全量导出任务 metadata 的 binlog name，position 和 gtid 信息
3. 修改其中的 GTID 信息的 `1-y` 为 `previous_gtids` 的前段值，例如，上述例子需要改为 `6-y`
4. 配置 `task.yaml` 中的 `syncers.safe-mode` 为 `true` 并重启任务
5. 待增量同步追上后，停止任务并在任务配置文件中设置 `safe-mode` 为 `false`
6. 再次重启任务
7. 重启 source 并关闭 gtid 或 relay

- 如果上述条件均不满足或任务同步数据量较小：

1. 清空下游数据库中数据
2. 重启 source 并关闭 gtid 或 relay
3. 重建任务并通过 `start-task task.yaml --remove-meta` 重新同步

上述处理方案中，针对正常同步的 source（如上例 mysql2），重设增量任务时起始点需设置 `mysql-instances.meta` 为 `subTaskStatus.sync` 的 `syncerBinlog` 与 `syncerBinlogGtid`。

9.23 在 DM 2.0 中开启 heartbeat，虚拟 IP 环境下切换 DM-worker 与 MySQL 实例的连接，遇到 “heartbeat config is different from previous used: serverID not equal” 错误

`heartbeat` 功能在 DM v2.0 及之后版本已经默认关闭，如果用户在同步任务配置文件中开启会干扰高可用特性，在配置文件中关闭该项（通过设置 `enable-heartbeat: false`，然后更新任务配置）即可解决。DM 将会在后续版本强制关闭该功能。

9.24 DM-master 在重启后无法加入集群，报错信息为 “fail to start embed etcd, RawCause: member xxx has already been bootstrapped”

DM-master 会在启动时将 etcd 信息记录在当前目录。如果重启后当前目录发生变化，会导致 DM 缺失 etcd 信息，从而启动失败。

推荐使用 TiUP 运维 DM 避免这一问题。在需要使用二进制部署的场合，需要在 DM-master 配置文件中使用绝对路径配置 data-dir 项，或者注意运行命令的当前目录。

9.25 使用 dmctl 执行命令时无法连接 DM-master

在使用 dmctl 执行相关命令时，发现连接 DM-master 失败（即使已在命令中指定 `--master-addr` 的参数值），报错内容类似 `RawCause: context deadline exceeded, Workaround: please check your network connection.`，但使用 `telnet <master-addr>` 之类的命令检查网络却没有发现异常。

这种情况可以检查下环境变量 `https_proxy`（注意，这里是 **https** ）。如果配置了该环境变量，dmctl 会自动去连接 `https_proxy` 指定的主机及端口，而如果该主机没有相应的 `proxy` 转发服务，则会导致连接失败。

解决方案：确认 `https_proxy` 是否必须要配置，如果不是必须的，取消该设置即可。如果环境必须，那么在原命令前加环境变量设置 `https_proxy="" ./dmctl --master-addr "x.x.x.x:8261"` 即可。

> **注意：**
>
> 关于 `proxy` 的环境变量有 `http_proxy`，`https_proxy`，`no_proxy` 等。如果依据上述解决方案处理后仍无法连接，可以考虑检查 `http_proxy` 和 `no_proxy` 的参数配置是否有影响。

9.26 v2.0.2 - v2.0.6 版本执行 start-relay 命令报错该如何处理？

```
flush local meta, Rawcause: open relay-dir/xxx.000001/relay.metayyyy: no such file or directory
```

上述报错在以下情况下有可能会被触发：

- DM 从 v2.0.1 及之前的版本升级到 v2.0.2 - v2.0.6 版本，且升级之前曾开启过 relay log，升级完后重新开启。
- 使用 stop-relay 命令暂停 relay log 后重新开启 relay log。

可以通过以下方式解决该问题：

- 重启 relay log:

```
» stop-relay -s sourceID workerName
» start-relay -s sourceID workerName
```

- 升级 DM 至 v2.0.7 或之后版本。

10 TiDB Data Migration 术语表

本文档介绍 TiDB Data Migration (TiDB DM) 相关术语。

10.1 B

10.1.1 Binlog

在 TiDB DM 中，Binlog 通常指 MySQL/MariaDB 生成的 binary log 文件，具体请参考 MySQL Binary Log 与 MariaDB Binary Log。

10.1.2 Binlog event

MySQL/MariaDB 生成的 Binlog 文件中的数据变更信息，具体请参考 MySQL Binlog Event 与 MariaDB Binlog Event。

10.1.3 Binlog event filter

比 Block & allow table list 更加细粒度的过滤功能，具体可参考Binlog Event Filter。

10.1.4 Binlog position

特定 Binlog event 在 Binlog 文件中的位置偏移信息，具体请参考 MySQL `SHOW BINLOG EVENTS` 与 MariaDB `SHOW BINLOG EVENTS`。

10.1.5 Binlog replication 处理单元/ sync 处理单元

DM-worker 内部用于读取上游 Binlog 或本地 Relay log 并迁移到下游的处理单元，每个 Subtask 对应一个 Binlog replication 处理单元。在当前文档中，有时也称作 Sync 处理单元。

10.1.6 Block & allow table list

针对上游数据库实例表的黑白名单过滤功能，具体可参考Block & Allow Table Lists。该功能与 MySQL Replication Filtering 及 MariaDB Replication Filters 类似。

10.2 C

10.2.1 Checkpoint

TiDB DM 在全量导入与增量复制过程中的断点信息，用于在重新启动或恢复任务时从之前已经处理过的位置继续执行。

- 对于全量导入，Checkpoint 信息对应于每个数据文件已经被成功导入的数据对应的文件内偏移量等信息，其在每个导入数据的事务中迁移更新；
- 对于增量复制，Checkpoint 信息对应于已经成功解析并导入到下游的Binlog event 对应的Binlog position 等信息，其在 DDL 导入成功后或距上次更新时间超过 30 秒等条件下更新。

另外，Relay 处理单元 对应的 `relay.meta` 内记录的信息也相当于 Checkpoint，其对应于 Relay 处理单元已经成功从上游拉取并写入到Relay log 的Binlog event 对应的Binlog position 或GTID 信息。

10.3 D

10.3.1 Dump 处理单元

DM-worker 内部用于从上游导出全量数据的处理单元，每个 Subtask 对应一个 Dump 处理单元。

10.4 F

10.4.1 复制/增量复制

使用 TiDB Data Migration 工具将上游数据库的增量数据复制到下游数据库的过程。

本用户手册中，在明确提到是 “增量” 的情况下，将使用 “复制” 或 “增量复制” 进行文档描述。

10.5 G

10.5.1 GTID

MySQL/MariaDB 的全局事务 ID，当启用该功能后会在 Binlog 文件中记录 GTID 相关信息，多个 GTID 即组成为 GTID Set，具体请参考 MySQL GTID Format and Storage 与 MariaDB Global Transaction ID。

10.6 L

10.6.1 Load 处理单元

DM-worker 内部用于将全量导出数据导入到下游的处理单元，每个 Subtask 对应一个 Load 处理单元。在当前文档中，有时也称作 Import 处理单元。

10.7 Q

10.7.1 迁移/全量迁移

使用 TiDB Data Migration 工具将上游数据库的全量数据迁移到下游数据库的过程。

本用户手册中，在明确提到是 “全量” 的情况下，将使用 “迁移” 或 “全量迁移” 进行文档描述；在明确提到是 “全量 + 增量” 的情况下，也将统一使用 “迁移” 进行文档描述。

10.8 R

10.8.1 Relay log

DM-worker 从上游 MySQL/MariaDB 拉取 Binlog 后存储在本地的文件，当前其格式为标准的 Binlog 格式，可使用版本兼容的 mysqlbinlog 等工具进行解析。其作用与 MySQL Relay Log 及 MariaDB Relay Log 相近。

有关 TiDB DM 内 Relay log 的目录结构、初始迁移规则、数据清理等内容，可参考 TiDB DM Relay Log。

10.8.2 Relay 处理单元

DM-worker 内部用于从上游拉取 Binlog 并写入数据到 Relay log 的处理单元，每个 DM-worker 实例内部仅存在一个该处理单元。

10.9 S

10.9.1 Safe mode

指增量复制过程中，用于支持在表结构中存在主键或唯一索引的条件下可重复导入 DML 的模式。该模式的主要特点是：将来自上游的 `INSERT` 改写为 `REPLACE`，将 `UPDATE` 改写为 `DELETE` 与 `REPLACE` 后再向下游执行。

该模式会在满足如下任一条件时启用：

- 任务配置文件中设置 `safe-mode: true` 时会始终启用
- 合库合表模式下，DDL 尚未在所有分表完成同步时保持启用

- 在全量迁移任务中的 dump 处理单元配置 `--consistency none` 后，不能确定导出开始时的 binlog 变动是否影响了导出数据。Safe mode 会在增量复制这部分 binlog 时保持启用
- 任务出错停止并恢复后，对有些数据的操作可能会被执行两次时保持启用

10.9.2 Shard DDL

指合库合表迁移过程中，在上游各分表 (shard) 上执行的需要 TiDB DM 进行协调迁移的 DDL。在当前文档中，有时也称作 Sharding DDL。

10.9.3 Shard DDL lock

用于协调 Shard DDL 迁移的锁机制，具体原理可查看悲观模式下分库分表合并迁移实现原理。在当前文档中，有时也称作 Sharding DDL lock。

10.9.4 Shard group

指合库合表迁移过程中，需要合并迁移到下游同一张表的所有上游分表 (shard)，TiDB DM 内部具体实现时使用了两级抽象的 Shard group，具体可查看悲观模式下分库分表合并迁移实现原理。在当前文档中，有时也称作 Sharding group。

10.9.5 Subtask

数据迁移子任务，即数据迁移任务运行在单个 DM-worker 实例上的部分。根据任务配置的不同，单个数据迁移任务可能只有一个子任务，也可能有多个子任务。

10.9.6 Subtask status

数据迁移子任务所处的状态，目前包括 `New`、`Running`、`Paused`、`Stopped` 及 `Finished` 5 种状态。有关数据迁移任务、子任务状态的更多信息可参考任务状态。

10.10 T

10.10.1 Table routing

用于支持将上游 MySQL/MariaDB 实例的某些表迁移到下游指定表的路由功能，可以用于分库分表的合并迁移，具体可参考Table routing。

10.10.2 Task

数据迁移任务，执行 `start-task` 命令成功后即启动一个数据迁移任务。根据任务配置的不同，单个数据迁移任务既可能只在单个 DM-worker 实例上运行，也可能同时在多个 DM-worker 实例上运行。

10.10.3 Task status

数据迁移子任务所处的状态，由Subtask status 整合而来，具体信息可查看任务状态。

11 版本发布历史

11.1 v5.3

11.1.1 DM 5.3.0 Release Notes

发版日期：2021 年 11 月 30 日

DM 版本：5.3.0

11.1.1.1 特别说明

在较早版本中（v1.0 和 v2.0），DM 采用独立于 TiDB 的版本号。从 DM v5.3 起，DM 采用与 TiDB 相同的版本号。DM v2.0 的下一个版本为 DM v5.3。DM v2.0 到 v5.3 无兼容性变更，升级过程与正常升级无差异，仅仅是版本号上的增加。

11.1.1.2 改进提升

- 开启 Relay Log 同步时大幅度降低延迟 #2225
- 增量同步时压缩/合并 DML 语句，大幅度降低同步延迟 #3162 #3167
- 支持通过 OpenAPI 运维管理 DM 集群（实验特性）#1928
- 优化 dmctl 的使用体验并增加一些子命令 #1746
- 支持停止或暂停同步任务时保持事务原子性 #1928
- 支持读取文件名大于 999999 Relay Log 文件 #1933
- load 和 sync 处理单元支持更多的监控指标 #1778
- 支持通过 dmctl 并发操控同步任务 #1995
- 优化增量同步时 DML 并发度 #2043
- 检测到 HTTP 代理相关的环境变量时提示用户 #1960
- 优化处理 RowEvent 时的日志显示 #2006
- 优化 SQL 执行过慢时的日志显示 #2024
- 优化获取数据源状态数据的逻辑，减少对上游的压力 #2076
- 遇到不支持的 binlog 格式时，报错并提示用户 #2099
- 支持通过 dmctl 批量操作数据源里的所有同步任务 #2166
- 通过下游表结构来生成 DML WHERE 语句 #3168
- 支持自动获取和配置上下游的时区 #3403

11.1.1.3 Bug 修复

- 修复上下游配置 SSL 证书时高可用调度失败的问题 #1910

- 修复暂停任务耗时过多的问题 #1945
- 修复 handle-error revert 返回错误信息不明确的问题 #1969
- 修复使用 binlog filter 跳过某些 DDL 时同步任务失败的问题 #1975
- 修复 evict-leader 在某些情况下失效的问题 #1986
- 修复 dmctl 返回错误信息不明确的问题 #2063
- 修复开启 Relay Log 时 DM-worker 调度失败的问题 #2199
- 修复开启 Relay Log 时 DM-worker 不能连接上游而启动失败的问题 #2227
- 修复开启 Relay Log 且上游发生切换时 meta 文件写入失败的问题 #3164

11.1.1.4 已知问题

GitHub issues

11.2 v2.0

11.2.1 DM 2.0.7 Release Notes

发版日期：2021 年 9 月 29 日

DM 版本：2.0.7

11.2.1.1 Bug 修复

- 修复 source 配置中的 `enable-gtid` 从 `false` 切换到 `true` 时，报错 binlog 被清除的问题 #2094
- 修复 schema tracker 的内存泄漏问题 #2133

11.2.1.2 改进提升

- 禁用 schema tracker 的后台统计线程，以减少 CPU 消耗 #2065
- 支持配置 online DDL shadow/trash 表的正则表达式规则 #2139

11.2.1.3 已知问题

GitHub issues

11.2.2 DM 2.0.6 Release Notes

发版日期：2021 年 8 月 13 日

DM 版本：2.0.6

11.2.2.1 Bug 修复

- 修复乐观协调模式下 DDL infos 和 upstream table 的元数据不一致导致 DM-master panic 的问题#1971

11.2.2.2 已知问题

GitHub issues

11.2.3 DM 2.0.5 Release Notes

发版日期：2021 年 7 月 30 日

DM 版本：2.0.5

11.2.3.1 改进提升

- 支持使用 SQL 表达式过滤某些 DML #1832
- 支持使用 `config import/export` 命令导入和导出集群上游和任务相关配置文件，用于降级回退 #1921
- 优化 safe-mode 提升同步效率 #1920
- 最大程度兼容上游 SQL_MODE #1894
- 单个任务同时支持上游使用 pt 和 gh-ost 两种 online DDL 模式 #1918
- 提升 DECIMAL 类型的同步效率 #1841
- 支持自动重试事务相关的可重试的错误 #1916
- 监控展示相关优化 #1808
- 错误信息相关优化 #1861
- 升级 Golang 版本到 v1.16 #1922

11.2.3.2 Bug 修复

- 修复上下游主键不一致可能导致数据丢失的问题 #1919
- 修复上游 source 过多导致集群升级失败并且 DM-master OOM 的问题 #1868
- 修复 `case-sensitive` 配置项不生效的问题 #1886
- 修复 DM 内部 `tidb_enable_change_column_type` 默认值错误的问题 #1843
- 修复下游表结构存在 `auto_random` 列导致任务中断的问题 #1847
- 修复 `operate-schema set -flush` 命令导致 DM-worker panic 的问题 #1829
- 修复悲观模式下相同 DDL 重复执行导致 DM-worker 内 DDL 协调失败的问题 #1816
- 修复配置错误导致 DM-worker panic 的问题 #1842
- 修复重做任务导致 loader panic 的问题 #1822
- 修复上游主从切换后 DM binlog 文件名更新不及时的问题 #1874
- 修复同步延迟监控数值错误的问题 #1880
- 修复 block-allow-list 某些情况下无法过滤 online DDL 的问题 #1867
- 修复任务自动恢复时报错导致无法手动暂停的问题 #1917

11.2.3.3 已知问题

GitHub issues

11.2.4 DM 2.0.4 Release Notes

发版日期：2021 年 6 月 18 日

DM 版本：2.0.4

11.2.4.1 改进提升

- 支持全量导入过程中 DM-worker 下线后重新上线时，任务重新调度并自动恢复 #1784
- 增加增量同步时延监控 #1759
- 全量导入阶段并发创建表结构 #1701
- 支持自动调整上下游数据库 time_zone 设置 #1714
- 提升增量阶段任务出错暂停后回滚的速度 #1705
- 增量迁移过程中开启 GTID 时，自动从断点处设置 GTID #1745
- 检测上下游数据库版本并输出到日志文件中 #1693
- 使用全量导出的表结构作为增量阶段的初始表结构 #1754
- 减小增量任务重启时 safe mode 的时间到 1 分钟，提升任务重启后的同步速度 #1779
- 提升 dmctl 的易用性
  - 支持在环境变量中设置 DM-master 的地址 #1726
  - 支持在命令的任意位置指定 master-addr 参数 #1771
  - 使用 `encrypt/decrypt` 命令代替原有的 `--decrypt/--encrypt` 参数方式加密/解密数据库密码 #1771

11.2.4.2 Bug 修复

- 修复非 GTID 同步任务中断重启后可能丢数据的问题 #1781
- 修复 DM 降级后重新升级可能丢失数据源绑定关系的问题 #1713
- 修复 DM-master 重启时 etcd 报 wal 目录不存在的问题 #1680
- 修复前置检查错误消息过多超过 grpc 限制的问题 #1688
- 修复同步旧版本 MariaDB 时，遇到不支持的语句导致 DM-worker panic 的问题 #1734
- 修复 relay log 磁盘容量监控不更新的问题 #1753
- 修复 DM 获取上游数据库 binlog 状态时出错导致 panic 的问题 #1774

11.2.4.3 已知问题

GitHub issues

11.2.5 DM 2.0.3 Release Notes

发版日期：2021 年 5 月 11 日

DM 版本：2.0.3

11.2.5.1 改进提升

- 支持任务停止后使用 `unlock-ddl-lock` 命令删除残留的 ddl-lock #1612
- 支持限制前置检查返回错误和警告的数量 #1621
- 优化 `query-status` 获取上游 binlog 状态的行为 #1630
- 优化悲观模式下 `query-status` 命令对分表同步状态的展示 #1650
- dmctl 优先显示帮助信息 #1637
- ddl-lock 删除后，自动删除监控中残留的相关信息 #1631
- 任务停止或完成后，自动删除监控中残留的任务状态 #1614

11.2.5.2 Bug 修复

- 修复乐观协调过程中滚动升级到 v2.0.2 后 DM-master OOM 的问题 #1643 #1649
- 修复滚动升级到 v2.0.2 后首次启动 source 绑定信息丢失的问题 #1649
- 修复 `operate-source show -s` flag 不生效的问题 #1587
- 修复 `operate-source stop <config-file>` 因上游无法连接而失败的问题 #1587
- 减小错误忽略的粒度，修复同步错误可能被错误跳过的问题 #1599
- 修复 online DDL 被配置的 binlog event 过滤时同步中断的问题 #1668

11.2.5.3 已知问题

GitHub issues

11.2.6 DM 2.0.2 Release Notes

发版日期：2021 年 4 月 9 日

DM 版本：2.0.2

11.2.6.1 改进提升

- Relay log GA
  - Relay log 不再通过设置上游数据源配置文件来开启，而是通过 dmctl 为指定的 DM-worker 开启 #1499
  - `query-status -s` 与 `purge-relay` 会发送到所有拉取 relay log 的 DM-worker #1533
  - 调整 relay 单元拉取、发送 binlog 的行为与 MySQL 从库一致 #1390
  - 减少清理 relay log 的场景 #1400
  - 为启用 relay 功能时增加心跳 binlog，以定时更新显示进度 #1404
- 乐观 DDL 协调模式
  - 优化修正 DDL 冲突的操作 #1496 #1506 #1518 #1551
  - 调整乐观 DDL 协调行为，预先避免进入数据不一致的状态 #1510 #1512

- 在 IP 不变等无需更新上游数据源配置的前提下，支持自动识别上游数据源切换 #1364
- 任务前置检查会以更细粒度检查某些权限 #1366
- 支持在上游数据源配置中配置 binlog 过滤 #1370
- DM-master 在绑定空闲上游数据源和 DM-worker 时，会优先使用 DM-worker 最近一次的绑定关系 #1373
- 提升了从 binlog 中自动获取 SQL mode 的稳定性 #1382 #1552
- 支持在上游数据源配置中，自动尝试解析不同格式的 GTID #1385
- 增加了 DM-worker keepalive 间隔，减少网络环境较差时产生的调度 #1405
- 配置文件中存在没有被引用的配置项时会报错 #1410
- 按字典顺序显示 GTID set，提升显示效果 #1424
- 优化了监控、告警规则 #1438
- 支持手动调度上游数据源到指定 DM-worker #1492
- 新增 etcd 压缩与磁盘配额配置 #1521

11.2.6.2 Bug 修复

- 修复了 DM 频繁重启任务导致全量阶段同步数据缺失的问题 #1378
- 修复了增量任务起始点仅指定 GTID、不指定 binlog position 无法启动的问题 #1393
- 修复了在较差磁盘、网络环境下，DM-worker 绑定关系异常的问题 #1396
- 修复了上游 binlog previous_gtids 事件的 GTID 不连续时，开启 relay 功能同步可能丢失数据的问题 #1390 #1430
- 屏蔽 DM 1.0 版本的心跳功能，避免高可用调度失效 #1467
- 修复了上游 binlog 序号超过 999999 同步失败的问题 #1476
- 修复了上下游数据库 ping 卡住导致命令卡住的问题 #1477
- 修复了上游开启 ANSI_QUOTES 时，全量导入失败的问题 #1497
- 修复了同时启用 GTID 和 relay 时可能重复同步 binlog 的问题 #1525

11.2.6.3 已知问题

GitHub issues

11.2.7 DM 2.0.1 Release Notes

发版日期：2020 年 12 月 25 日

DM 版本：2.0.1

11.2.7.1 改进提升

- 增加高可用场景下对 relay log 的支持 #1353
  - relay log 只支持 DM-worker 本地存储。
  - 在 DM-worker 节点宕机、网络波动导致节点下线等场景中，新调度的 DM-worker 会重新向上游拉取 binlog。

- 限制 handle-error 命令只能处理 DDL 错误以避免误用 #1303
- 支持 dmctl 同时连接多个 DM-master 节点及自动切换连接节点 #1349
- 增加 get-config 命令用于获取迁移任务和 DM 组件的配置 #1348
- 支持迁移 ALTER TABLE ADD COLUMN (xx, xx) 语句 #1345
- 支持自动过滤 CREATE/ALTER/DROP EVENT 语句 #1343
- 增加增量复制任务开始前对上游 MySQL/MariaDB 是否设置 server-id 的检查 #1315
- 支持全量导入名字中包含 sql 的库和表 #1259

11.2.7.2 Bug 修复

- 修复任务重启时出现错误 fail to initial unit Sync of subtask 的问题 #1274
- 修复全量数据导入时正在执行的 pause-task 命令阻塞的问题 #1269 #1277
- 修复配置 enable-gtid: true 时，DM 无法为 MariaDB 实例创建数据源的问题 #1344
- 修复 query-status 命令执行时阻塞的问题 #1293
- 修复悲观 shard DDL 模式下，并发协调多条 DDL 语句时可能造成阻塞的问题 #1263
- 修复 pause-task 命令执行时可能出现错误 sql: connection is already closed 的问题 #1304
- 修复上游无 REPLICATION 权限时全量同步失败的问题 #1326
- 修复 SQL_MODE 包含 ANSI_QUOTES 时，合库合表任务 route-rules 配置在全量导入时无法生效的问题 #1314
- 修复增量同步时没有自动应用上游 SQL_MODE 的问题 #1307
- 修复自动解析上游 SQL_MODE 时日志出现 WARN fail to parse binlog status_vars 的问题 #1299

11.2.8 DM 2.0 GA Release Notes

发版日期：2020 年 10 月 30 日

DM 版本：2.0.0

11.2.8.1 改进提升

- 对于 Amazon Aurora、阿里云 RDS 等不能使用 FTWRL 进行全量导出的场景，优化 safe-mode 的开启以确保数据的最终一致性 #981 #1017
- 支持根据上下游的 global sql_mode 及 binlog event 中的 sql_mode，自动配置数据迁移过程中所需要的 sql_mode #1005 #1071 #1137
- 支持根据下游 TiDB 的 global max_allowed_packet，自动设置 DM 连接到下游 client 的 max_allowed_packet #1071
- 相对 DM 2.0 RC 版本优化了增量复制的速度 #1203
- 默认使用乐观事务向 TiDB 迁移数据以提升性能 #1107
- DM-worker 支持自动获取并使用集群中的 DM-master 节点列表 #1180
- 禁用对更多不可自动恢复错误的 auto-resume 行为 #979 #1085 #1216

11.2.8.2 Bug 修复

- 修复未自动设置全量导出的 `statement-size` 默认值而可能造成 `packet for query is too large` 或 TiDB OOM 的问题 #1133
- 修复了全量导入过程中，可能由于并发 checkpoint 操作而造成 DM-worker panic 的问题 #1182
- 修复上游 MySQL/MariaDB 实例重启后，数据迁移任务可能触发 `table checkpoint position * less than global checkpoint position` 而中断的问题 #1041
- 修复上游未开启 GTID 时可能导致数据迁移任务中断的问题 #1123
- 修复 shard DDL 协调遇到冲突后，DM-master 节点无法正常启动的问题 #1199
- 修复待迁移的表中存在多个普通索引时，可能导致增量复制速度过慢的问题 #1063
- 修复全量导入过程中，重启数据迁移任务后进度显示异常的问题 #1043
- 修复处于暂停状态的数据迁移子任务被调度到其他 DM-worker 后无法被 `query-status` 获取的问题 #1183
- 修复全量导出时，`FileSize` 参数可能不生效的问题 #1191
- 修复全量导出时，`extra-args` 中 `-s` 参数不生效的问题 #1196
- 修复启用 online DDL 支持时，可能触发 `not allowed operation: alter multiple tables in one statement` 错误的问题 #1192
- 修复增量复制时，待迁移的 DDL 语句有关联其他表时（如 foreign key 相关的 DDL）可能导致任务中断的问题 #1101 #1108
- 修复全量迁移过程中，不能正确处理数据库名、表名中包含 / 字符的问题 #991
- 修复增量复制过程中，向下游 TiDB 执行 DDL 失败后数据迁移任务可能未暂停且无法通过 `query-status` 获取到相应错误的问题 #1059
- 修复乐观 shard DDL 模式下，并发协调多条 DDL 语句时可能造成阻塞的问题 #1051
- 修复 DM-master 成为 leader 后仍可能尝试转发请求给其他 DM-master 节点的问题 #1157
- 修复任务前置检查时，无法解析 `GRANT CREATE TABLESPACE` 的问题 #1113
- 修复迁移 `DROP TABLE` 语句而对应数据表不存在时，数据迁移任务中断的问题 #990
- 修复 `operate-schema` 在指定 `--source` 参数时可能无法正确工作的问题 #1106
- 修复开启 TLS 后，无法正确执行 `list-member` 的问题 #1050
- 修复开启 TLS 后，配置项中混合使用 `https` 与 `http` 可能导致集群无法正常工作的问题 #1220
- 修复 DM-master 启用 `cert-allowed-cn` 后，HTTP API 无法正常使用的问题 #1036
- 修复对于增量数据复制任务，仅在任务配置的 `meta` 中指定 `binlog-gtid` 时，配置检查不能通过的问题 #987
- 修复 dmctl 在交互模式下，部分命令首尾包含空白字符时执行出错的问题 #1202
- 修复全量导出时，日志中输出 `converting NULL to string is unsupported` 错误的问题 #1014
- 修复全量导入时，进度可能显示为 `NaN` 的问题 #1209

11.2.9 DM 2.0 RC.2 Release Notes

发版日期：2020 年 9 月 1 日

DM 版本：2.0.0-rc.2

11.2.9.1 改进提升

- 在数据迁移任务前置检查时，增加对更多 AWS Aurora 特有权限的兼容 #950
- 配置 `enable-gtid: true` 并创建数据源时，增加对上游 MySQL/MariaDB 是否启用 GTID 的检查 #957

11.2.9.2 Bug 修复

- 修复从 1.0.x 自动升级到 2.0.0-rc 后，数据迁移任务运行时出现 `Column count doesn't match value count` 错误的问题 #952
- 修复 DM-worker/DM-master 组件可能无法正确退出的问题 #963
- 修复 2.0 版本中 `--no-locks` 参数对 dump 处理单元不生效的问题 #961
- 修复使用 1.0.x 的数据迁移任务配置文件在 2.0 集群中启动任务时可能出现 `field remove-meta not found in type config.TaskConfig` 错误的问题 #965
- 修复使用域名作为各组件的连接地址时可能无法正确启动相关组件的问题 #955
- 修复在停止数据迁移任务后，到上下游数据库的连接可能未被释放的问题 #943
- 修复乐观 shard DDL 模式下，多表并发执行 DDL 时可能阻塞 shard DDL 协调的问题 #944
- 修复了新启动的 DM-master 可能导致 `list-member` panic 的问题 #970

11.2.10 DM 2.0 RC Release Notes

发版日期：2020 年 8 月 21 日

DM 版本：2.0.0-rc

11.2.10.1 改进提升

- 增加对数据迁移任务的高可用支持
- 增加 sharding DDL 乐观协调模式
- 增加 `handle-error` 命令用于处理增量过程中 DDL 复制相关的错误
- 为 `query-status` 返回的错误增加了 `workaround` 字段用于描述错误处理方法
- 增加与完善监控面板及告警规则
- 使用 Dumpling 替换 Mydumper 作为全量导出模块
- 增加向下游执行增量复制时的 GTID 模式支持
- 增加对上下游数据库及 DM 各组件间 TLS 连接的支持
- 增加下游表结构比上游存在更多列的增量复制场景支持
- 为 `start-task` 增加 `--remove-meta` 参数用于清理数据迁移任务相关的元信息
- 支持 `DROP` 存在单列索引的列
- 增加在全量导入成功后自动清理临时文件的支持
- 增加启动任务前检查待迁移表是否存在主键或唯一键的支持
- 增加对 dmctl 在启动时与 DM-master 的连通性检查
- 增加在 `start-task`/`check-task` 时对下游 TiDB 的连通性检查
- 增加对 `pause-task` 等部分命令以任务配置文件取代任务名的支持

- 为 DM-master、DM-worker 组件增加了 json 格式 log 的支持
- 优化移除了 query-status 返回错误信息中的程序调用栈信息与冗余字段
- 优化了 query-status 返回的上游数据库 binlog position 信息
- 优化了对全量导出过程中遇到错误时的 auto resume 处理

11.2.10.2 问题修复

- 修复了执行 stop-task 后部分 goroutine 泄露的问题
- 修复了执行 pause-task 后任务可能仍未暂停的问题
- 修复了增量复制初始阶段可能未正确保存 checkpoint 的问题
- 修复了增量复制时未能正确处理 BIT 类型的问题

11.2.10.3 详细变更及问题修复

- 增加对数据迁移任务的高可用支持 #473
- 增加 sharding DDL 乐观协调模式 #568
- 增加 handle-error 命令用于处理增量过程中 DDL 迁移相关的错误 #850
- 为 query-status 返回的错误增加了 workaround 字段用于描述错误处理方法 #753
- 增加与完善监控面板及告警规则 #853
- 使用 Dumpling 替换 Mydumper 作为全量导出模块 #540
- 增加向下游执行增量复制时的 GTID 模式支持 #521
- 增加对上下游数据库及 DM 各组件间 TLS 连接的支持 #569
- 增加下游表结构比上游存在更多列的增量复制场景支持 #379
- 为 start-task 增加 --remove-meta 参数用于清理数据迁移任务相关的元信息 #651
- 支持 DROP 存在单列索引的列 #801
- 增加在全量导入成功后自动清理临时文件的支持 #770
- 增加启动任务前检查待迁移表是否存在主键或唯一键的支持 #870
- 增加对 dmctl 在启动时与 DM-master 的连通性检查 #786
- 增加在 start-task/check-task 时对下游 TiDB 的连通性检查 #769
- 增加对 pause-task 等部分命令以任务配置文件取代任务名的支持 #854
- 为 DM-master、DM-worker 组件增加了 json 格式 log 的支持 #808
- 优化移除了 query-status 返回错误信息中的程序调用栈信息 #746
- 优化移除了 query-status 返回错误信息中的冗余字段 #771
- 优化了 query-status 返回的上游数据库 binlog position 信息 #830
- 优化了对全量导出过程中遇到错误时的 auto resume 处理 #872
- 修复了执行 stop-task 后部分 goroutine 泄露的问题 #731
- 修复了执行 pause-task 后任务可能仍未暂停的问题 #644
- 修复了增量复制初始阶段可能未正确保存 checkpoint 的问题 #758
- 修复了增量复制时未能正确处理 BIT 类型的问题 #876

11.3 v1.0

11.3.1 DM 1.0.7 Release Notes

发版日期：2021 年 6 月 21 日

DM 版本：1.0.7

11.3.1.1 Bug 修复

- 修复同步任务中断重启后可能丢数据的问题 #1783

11.3.2 DM 1.0.6 Release Notes

发版日期：2020 年 6 月 17 日

DM 版本：1.0.6

DM-Ansible 版本：1.0.6

11.3.2.1 改进提升

- 增加对上下游数据库原始明文密码的支持
- 为 DM 到上下游数据库的连接增加配置 session 变量的支持
- 移除了数据迁移任务异常时通过 `query-status` 返回的部分错误提示中的程序调用栈信息
- 移除了数据迁移任务前置检查失败时，返回的提示消息中的成功项信息

11.3.2.2 问题修复

- 修复 load 单元在创建表结构遇到错误后，数据迁移任务未自动暂停且 `query-status` 无法查询到对应错误的问题
- 修复了多个数据迁移任务同时运行时 DM-worker 有低概率 panic 的问题
- 修复了数据迁移任务设置 `enable-heartbeat: true` 后，重启 DM-worker 进程时已有数据迁移任务无法自动恢复的问题
- 修复了 `resume-task` 后可能无法正常显示 shard DDL 冲突错误的问题
- 修复了数据迁移任务设置 `enable-heartbeat: true` 后，初始一段时间内 `replicate lag` 可能显示异常的问题
- 修复了上游数据库设置 `lower_case_table_names=1` 时，可能无法通过 heartbeat 计算 `replicate lag` 的问题
- 禁用了数据迁移过程中对 `unsupported collation` 错误的无意义 `auto resume`

11.3.2.3 详细变更及问题修复

- 增加对上下游数据库原始明文密码的支持 #676
- 为 DM 到上下游数据库的连接增加配置 session 变量的支持 #692
- 移除了数据迁移任务异常时通过 `query-status` 返回的部分错误提示中的程序调用栈信息 #733 #747
- 移除了数据迁移任务前置检查失败时，返回的提示消息中的成功项信息 #730
- 修复 load 单元在创建表结构遇到错误后，数据迁移任务未自动暂停且 `query-status` 无法查询到对应错误的问题 #747
- 修复了多个数据迁移任务同时运行时 DM-worker 有低概率 panic 的问题 #710
- 修复了数据迁移任务设置 `enable-heartbeat: true` 后，重启 DM-worker 进程时已有数据迁移任务无法自动恢复的问题 #739
- 修复了 `resume-task` 后可能无法正常显示 shard DDL 冲突错误的问题 #739 #742
- 修复了数据迁移任务设置 `enable-heartbeat: true` 后，初始一段时间内 `replicate lag` 可能显示异常的问题 #704
- 修复了上游数据库设置 `lower_case_table_names=1` 时，可能无法通过 heartbeat 计算 `replicate lag` 的问题 #704
- 禁用了数据迁移过程中对 `unsupported collation` 错误的无意义 `auto resume` #735
- 优化了部分 log #660 #724 #738

11.3.3 DM 1.0.5 Release Notes

发版日期：2020 年 4 月 27 日

DM 版本：1.0.5

DM-Ansible 版本：1.0.5

11.3.3.1 改进提升

- 优化了 `UNIQUE KEY` 对应列含 `NULL` 值时的增量复制速度
- 增加对 TiDB 返回的 `Write conflict`（9007 与 8005）错误的重试

11.3.3.2 问题修复

- 修复了全量数据导入过程中有概率触发 `Duplicate entry` 错误的问题
- 修复了全量导入完成后上游无数据写入时可能无法 `stop-task/pause-task` 的问题
- 修复 `stop-task` 后监控 metrics 仍有数据显示的问题

11.3.3.3 详细变更及问题修复

- 优化了 `UNIQUE KEY` 对应列含 `NULL` 值时的增量复制速度 #588 #597
- 增加对 TiDB 返回的 `Write conflict`（9007 与 8005）错误的重试 #632
- 修复了全量数据导入过程中有概率触发 `Duplicate entry` 错误的问题 #554

- 修复了全量导入完成后上游无数据写入时可能无法 `stop-task/pause-task` 的问题 #622
- 修复 `stop-task` 后监控 metrics 仍有数据显示的问题 #616
- 修复迁移过程中有概率出现 `Column count doesn't match value count` 的问题 #624
- 修复了全量导入阶段从 paused 状态 `resume-task` 后 `data file size` 等部分 metrics 显示错误的问题 #570
- 添加与修复了多个 metrics 监控项 #590 #594

11.3.4 DM 1.0.4 Release Notes

发版日期：2020 年 03 月 13 日

DM 版本：1.0.4

DM-Ansible 版本：1.0.4

11.3.4.1 改进提升

- DM-portal 新增英文 UI 的支持
- `query-status` 命令增加 `--more` 参数用于显示完整的迁移状态信息

11.3.4.2 问题修复

- 修复到下游 TiDB 连接异常导致迁移暂停后，resume-task 可能无法正常恢复迁移的问题
- 修复 online DDL 执行失败后错误清理了 online DDL meta 信息而导致重启任务后无法继续正确处理 online DDL 迁移的问题
- 修复 `start-task` 异常后 `query-error` 可能导致 DM-worker panic 的问题
- 修复 relay.meta 写入成功前 DM-worker 进程异常停止后，重启 DM-worker 时可能无法正确 recover relay log 文件与 relay.meta 的问题

11.3.4.3 详细变更及问题修复

- DM-portal 增加支持英文 UI #480
- `query-status` 命令增加 `--more` 参数用于显示完整的迁移状态信息 #533
- 修复到下游 TiDB 连接异常导致迁移暂停后，`resume-task` 可能无法正常恢复迁移的问题 #436
- 修复 online DDL 执行失败后错误清理了 online DDL meta 信息而导致重启任务后无法继续正确处理 online DDL 迁移的问题 #465
- 修复 `start-task` 异常后 `query-error` 可能导致 DM-worker panic 的问题 #519
- 修复 relay.meta 写入成功前 DM-worker 进程异常停止后，重启 DM-worker 时可能无法正确恢复 relay log 文件与 relay.meta 的问题 #534
- 修复获取上游 `server-id` 时可能报 `value out of range` 错误的问题 #538
- 修复 DM-Ansible 在未配置 Prometheus 时错误提示未配置 dm-master 的问题 #438

11.3.5 DM 1.0.3 Release Notes

发版日期：2019 年 12 月 13 日

DM 版本：1.0.3

DM-Ansible 版本：1.0.3

11.3.5.1 改进提升

- dmctl 支持命令式使用
- 支持迁移 `ALTER DATABASE` DDL 语句
- 优化 DM 错误提示信息

11.3.5.2 问题修复

- 修复全量导入模块在暂停或退出时 data race 导致 panic 的问题
- 修复对下游进行重试操作时，`stop-task` 和 `pause-task` 可能不生效的问题

11.3.5.3 详细变更及问题修复

- dmctl 支持命令式使用 #364
- 优化 DM 错误提示信息 #351
- 优化 `query-status` 命令输出内容 #357
- 优化 DM 不同任务类型的权限检查 #374
- 支持对重复引用的路由配置和过滤配置进行检查 #385
- 支持迁移 `ALTER DATABASE` DDL 语句 #389
- 优化 DM 异常重试机制 #391
- 修复全量导入模块在暂停或退出时 data race 导致 panic 的问题 #353
- 修复对下游进行重试操作时，`stop-task` 和 `pause-task` 可能不生效的问题 #400
- 更新 Golang 版本至 1.13 以及其他依赖包版本 #362
- 过滤 SQL 执行时出现的 `context canceled` 错误 #382
- 修复使用 DM-ansible 滚动升级 DM 监控过程中出错导致升级失败的问题 #408

11.3.6 DM 1.0.2 Release Notes

发版日期：2019 年 10 月 30 日

DM 版本：1.0.2

DM-Ansible 版本：1.0.2

11.3.6.1 改进提升

- 支持自动为 DM-worker 生成部分配置项
- 支持自动为数据迁移任务生成部分配置项
- 简化 `query-status` 在无参数时的默认输出
- DM 直接管理到下游数据库的连接

11.3.6.2 问题修复

- 修复在进程启动过程中以及执行 SQL 失败时可能 panic 的问题
- 修复 DDL 执行超时后可能造成 sharding DDL 协调异常的问题
- 修复由于前置检查超时或部分 DM-worker 不可访问而不能启动数据迁移任务的问题
- 修复 SQL 执行失败后可能错误重试的问题

11.3.6.3 详细变更及问题修复

- 支持自动为 DM-worker 生成随机的 `server-id` 配置项 #337
- 支持自动为 DM-worker 生成 `flavor` 配置项 #328
- 支持自动为 DM-worker 生成 `relay-binlog-name` 与 `relay-binlog-gtid` 配置项 #318
- 支持根据黑白名单生成 mydumper 需要导出的表名配置项 #326
- 为数据迁移任务增加并发度配置项 (`mydumper-thread`、`loader-thread` 与 `syncer-thread`) #314
- 简化 `query-status` 在无参数时的默认输出 #340
- 修复 DDL 执行超时后可能造成 sharding DDL 协调异常的问题 #338
- 修复 DM-worker 从本地 meta 数据恢复数据迁移任务时可能 panic 的问题 #311
- 修复提交事务失败时可能造成 DM-worker panic 的问题 #313
- 修复监听端口被占用时 DM-worker 或 DM-master 启动过程中可能 panic 的问题 #301
- 修复对 1105 错误码的部分重试问题 #321, #332
- 修复对 `Duplicate entry` 与 `Data too long for column` 错误的重试问题 #313
- 修复在上游存在大量需要迁移的表时可能造成启动任务前置检查超时中断的问题 #327
- 修复部分 DM-worker 不可访问时无法启动数据迁移任务的问题 #319
- 修复从损坏的 relay log 恢复时可能错误更新 GTID sets 信息的问题 #339
- 修复 sync 处理单元计算 TPS 错误的问题 #294
- DM 直接管理到下游数据库的连接 #325
- 提升组件内错误信息的传递方式 #320