TiDB Data Migration 用户文档

PingCAP Inc.

20220809

Table of Contents

1 概述 **10**

1.1 Data Migration 简介 · · · · · 10

1.1.1 DM 架构 · · · · · 10

1.1.2 迁移功能介绍 · · · · · 11

1.1.3 使用限制 · · · · · 12

1.2 DM-worker 简介 · · · · · 13

1.2.1 DM-worker 处理单元 · · · · · 13

1.2.2 DM-worker 所需权限 · · · · · 14

1.3 DM Relay Log · · · · · 16

1.3.1 目录结构 · · · · · 16

1.3.2 初始迁移规则 · · · · · 18

1.3.3 数据清理 · · · · · 18

2 核心特性 **20**

2.1 数据迁移功能 · · · · · 20

2.1.1 Table routing · · · · · 20

2.1.2 Block & allow table lists · · · · · 23

2.1.3 Binlog event filter · · · · · 27

2.1.4 Column mapping · · · · · 30

2.1.5 迁移延迟监控 · · · · · 33

2.2 DM online-ddl-scheme 34
2.2.1 概述 34
2.2.2 配置 35
2.2.3 online-schema-change: gh-ost 35
2.2.4 online-schema-change: pt 37
2.3 Shard Support 39
2.3.1 分库分表合并迁移 39
2.3.2 手动处理 Sharding DDL Lock 46

3 Benchmark **57**
3.1 DM 1.0-GA 性能测试报告 57
3.1.1 测试目的 58
3.1.2 测试环境 58
3.1.3 测试场景 59
3.1.4 推荐迁移任务参数配置 62

4 使用场景 **62**
4.1 Data Migration 简单使用场景 62
4.1.1 上游实例 62
4.1.2 迁移要求 63
4.1.3 下游实例 63
4.1.4 迁移方案 64
4.1.5 迁移任务配置 65
4.2 DM 分库分表合并场景 68
4.2.1 上游实例 68
4.2.2 迁移需求 68
4.2.3 下游实例 69
4.2.4 迁移方案 69
4.2.5 迁移任务配置 70

4.3 分表合并数据迁移最佳实践 73
4.3.1 独立的数据迁移任务 73
4.3.2 手动处理 sharding DDL lock 73
4.3.3 自增主键冲突处理 73
4.3.4 合表迁移过程中在上游增/删表 75
4.3.5 数据迁移限速/流控 76
4.4 切换 DM-worker 与上游 MySQL 实例的连接 76
4.4.1 虚拟 IP 环境下切换 DM-worker 与 MySQL 实例的连接 77
4.4.2 变更 DM-worker 连接的上游 MySQL 实例地址 78

5 TiDB Data Migration 教程 78
5.1 Data Migration 架构 79
5.2 安装 80
5.3 迁移分片数据 81
5.3.1 启动 DM-master 和 DM-worker 83
5.4 结论 89

6 部署 89
6.1 部署 DM 集群 89
6.1.1 使用 DM-Ansible 部署 DM 集群 89
6.1.2 使用 DM binary 部署 DM 集群 107
6.1.3 使用 Kubernetes (实验特性) 115
6.2 使用 DM 迁移数据 115
6.2.1 第 1 步：部署 DM 集群 115
6.2.2 第 2 步：检查集群信息 115
6.2.3 第 3 步：配置任务 116
6.2.4 第 4 步：启动任务 117
6.2.5 第 5 步：查询任务 118
6.2.6 第 6 步：停止任务 119
6.2.7 第 7 步：监控任务与查看日志 119

7 配置 119

7.1 DM 配置简介 119
7.1.1 配置文件 119
7.1.2 迁移任务配置 120
7.2 DM-master 配置文件介绍 120
7.2.1 配置文件示例 120
7.2.2 配置项说明 121
7.3 DM-worker 配置文件介绍 121
7.3.1 配置文件示例 122
7.3.2 配置项说明 122
7.4 DM-worker 完整配置说明 123
7.4.1 配置文件示例 123
7.4.2 配置项说明 124
7.5 DM 任务配置文件介绍 127
7.5.1 关键概念 127
7.5.2 基础配置文件示例 127
7.5.3 配置顺序 128
7.5.4 全局配置 129
7.5.5 实例配置 129
7.5.6 修改任务配置 129
7.6 DM 任务完整配置文件介绍 130
7.6.1 关键概念 130
7.6.2 关闭检查项 130
7.6.3 完整配置文件示例 130
7.6.4 配置顺序 133
7.6.5 全局配置 133
7.6.6 实例配置 134

8 DM 集群管理 135

8.1 DM 集群操作 135
8.1.1 启动集群 135
8.1.2 下线集群 135
8.1.3 重启集群组件 135

8.1.4 更新组件版本 137
8.1.5 创建 DM-worker 实例 138
8.1.6 下线 DM-worker 实例 139
8.1.7 替换/迁移 DM-master 实例 140
8.1.8 替换/迁移 DM-worker 实例 141
8.2 TiDB Data Migration 版本升级 143
8.2.1 升级到 v1.0.5 144
8.2.2 升级到 v1.0.4 144
8.2.3 升级到 v1.0.3 145
8.2.4 升级到 v1.0.2 146
8.2.5 升级到 v1.0.1 147
8.2.6 升级到 v1.0.0-10-geb2889c9 (1.0 GA) 147
8.2.7 升级到 v1.0.0-rc.1-12-gaa39ff9 148

9 DM 迁移任务管理 149
9.1 管理数据迁移任务 149
9.1.1 dmctl 交互模式 149
9.1.2 管理数据迁移任务 151
9.1.3 管理 DDL lock 160
9.1.4 其他任务与集群管理命令 161
9.1.5 dmctl 命令模式 167
9.1.6 废弃或不推荐使用的命令 168
9.2 上游 MySQL 实例配置前置检查 168
9.2.1 使用命令 168
9.2.2 检查内容 168
9.3 DM 查询状态 170
9.3.1 查询结果 170
9.3.2 任务状态 171
9.3.3 详情查询结果 173
9.3.4 子任务状态 177
9.4 跳过或替代执行异常的 SQL 语句 179

10 DM 监控指标 **194**
10.1 Task · · · · · 194
10.1.1 overview · · · · · 194
10.1.2 task 状态 · · · · · 195
10.1.3 Relay log · · · · · 196
10.1.4 Dump/Load unit · · · · · 197
10.1.5 Binlog replication · · · · · 198
10.2 Instance · · · · · 201
10.2.1 Relay log · · · · · 201
10.2.2 task · · · · · 203

11 从与 MySQL 兼容的数据库迁移数据 **204**
11.1 从 MySQL 迁移数据 —— 以 Amazon Aurora MySQL 为例 · · · · · 204
11.1.1 第 1 步：在 Aurora 集群中启用 binlog · · · · · 204
11.1.2 第 2 步：部署 DM 集群 · · · · · 205
11.1.3 第 3 步：检查集群信息 · · · · · 206
11.1.4 第 4 步：配置任务 · · · · · 207
11.1.5 第 5 步：启动任务 · · · · · 208
11.1.6 第 6 步：查询任务 · · · · · 209

12 DM Portal 简介 **210**
12.1 功能描述 · · · · · 210
12.1.1 迁移模式配置 · · · · · 210
12.1.2 实例信息配置 · · · · · 210
12.1.3 binlog 过滤配置 · · · · · 210
12.1.4 配置文件生成 · · · · · 210
12.1.5 使用限制 · · · · · 210
12.2 部署使用 · · · · · 211
12.2.1 Binary 部署 · · · · · 211
12.2.2 DM Ansible 部署 · · · · · 211

12.3 使用说明 211
12.3.1 新建规则 211
12.3.2 基础信息配置 211
12.3.3 实例信息配置 212
12.3.4 binlog 过滤配置 213
12.3.5 库表路由配置 216

13 告警处理 225
13.1 DM 告警信息 225
13.2 告警处理 225
13.2.1 任务状态告警 226
13.2.2 relay log 告警 226
13.2.3 Dump/Load 告警 227
13.2.4 Binlog replication 告警 227

14 故障处理 227
14.1 故障及处理方法 227
14.1.1 DM 错误系统 227
14.1.2 DM 故障诊断 230
14.1.3 常见故障处理方法 230
14.2 性能问题及处理方法 233
14.2.1 relay log 模块的性能问题及处理方法 233
14.2.2 Load 模块的性能问题及处理方法 234
14.2.3 Binlog replication 模块的性能问题及处理方法 234

15 Data Migration 常见问题 236
15.1 DM 是否支持迁移阿里 RDS 以及其他云数据库的数据？ 236
15.2 task 配置中的黑白名单的正则表达式是否支持非获取匹配 (?!)? 236
15.3 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？ 236
15.4 如何处理不兼容的 DDL 语句？ 236
15.5 如何重置数据迁移任务？ 237
15.5.1 relay log 无异常时重置迁移任务 237
15.5.2 relay log 存在异常时重置迁移任务 237

15.6 设置了 online-ddl-scheme: "gh-ost"，gh-ost 表相关的 DDL 报错该如何处理？ 238
15.7 如何为已有迁移任务增加需要迁移的表？ 239
15.7.1 迁移任务当前处于 Dump 阶段 239
15.7.2 迁移任务当前处于 Load 阶段 239
15.7.3 迁移任务当前处于 Sync 阶段 239
15.8 DM v1.0 在任务出错时使用 sql-skip 命令无法跳过某些语句 240
15.9 DM 同步时下游长时间出现 REPLACE 语句 240
15.10 使用 dmctl 执行命令时无法连接 DM-master 241

16 版本发布历史 241

16.1 v1.0 241
16.1.1 DM 1.0.7 Release Notes 241
16.1.2 DM 1.0.6 Release Notes 241
16.1.3 DM 1.0.5 Release Notes 243
16.1.4 DM 1.0.4 Release Notes 243
16.1.5 DM 1.0.3 Release Notes 244
16.1.6 DM 1.0.2 Release Notes 245

17 TiDB Data Migration 术语表 246

17.1 B 246
17.1.1 Binlog 246
17.1.2 Binlog event 246
17.1.3 Binlog event filter 246
17.1.4 Binlog position 246
17.1.5 Binlog replication 处理单元 247
17.1.6 Block & allow table list 247
17.2 C 247
17.2.1 Checkpoint 247
17.3 D 247
17.3.1 Dump 处理单元 247
17.4 F 247
17.4.1 复制/增量复制 247

17.5 G 248
17.5.1 GTID 248
17.6 H 248
17.6.1 Heartbeat 248
17.7 L 248
17.7.1 Load 处理单元 248
17.8 Q 248
17.8.1 迁移/全量迁移 248
17.9 R 248
17.9.1 Relay log 248
17.9.2 Relay 处理单元 249
17.10 S 249
17.10.1 Safe mode 249
17.10.2 Shard DDL 249
17.10.3 Shard DDL lock 249
17.10.4 Shard group 249
17.10.5 Subtask 249
17.10.6 Subtask status 249
17.11 T 250
17.11.1 Table routing 250
17.11.2 Task 250
17.11.3 Task status 250

1 概述

1.1 Data Migration 简介

TiDB Data Migration (DM) 是一体化的数据迁移任务管理平台，支持从 MySQL 或 MariaDB 到 TiDB 的全量数据迁移和增量数据复制。使用 DM 工具有利于简化错误处理流程，降低运维成本。

> **注意：**
>
> DM 以 SQL 语句的形式将数据迁移到 TiDB 中，因此各个版本的 DM 都分别兼容所有版本的 TiDB。在生产环境中，推荐使用 DM 的最新已发布版本。已发布版本的下载方式参见 DM 下载链接。

1.1.1 DM 架构

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

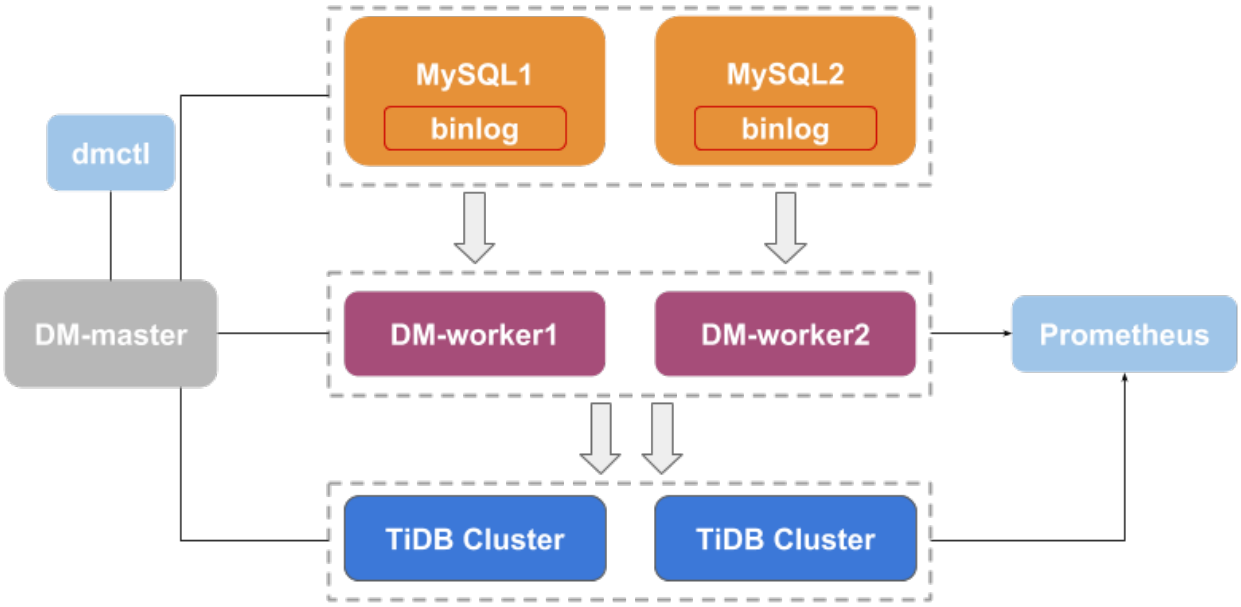


Figure 1: Data Migration architecture

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

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

DM-worker 启动后，会自动迁移上游 binlog 至本地配置目录（如果使用 DM-Ansible 部署 DM 集群，默认的迁移目录为 `<deploy_dir>/relay_log`）。关于 DM-worker，详见DM-worker 简介。关于 relay log，详见DM Relay Log。

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

1.1.2 迁移功能介绍

下面简单介绍 DM 数据迁移功能的核心特性。

1.1.2.1 Table routing

Table Routing 是指将上游 MySQL 或 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。

1.1.2.2 Block & allow table lists

Block & Allow Table Lists 是指上游数据库实例表的黑白名单过滤规则。其过滤规则类似于 MySQL `replication-rules-db/replication-rules-table`，可以用来过滤或只迁移某些数据库或某些表的所有操作。

1.1.2.3 Binlog event filter

Binlog Event Filter 是比库表迁移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 的指定类型的 binlog events，比如 INSERT，TRUNCATE TABLE。

1.1.2.4 Shard support

DM 支持对原分库分表进行合库合表操作，但需要满足一些使用限制。

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
  - 5.5 < MySQL 版本 < 8.0
  - MariaDB 版本 >= 10.1.2

> **注意：**
> 如果上游 MySQL/MariaDB server 间构成主从复制结构，则需要 5.7.1 < MySQL 版本 < 8.0 或者 MariaDB 版本 >= 10.1.3。

在使用 dmctl 启动任务时，DM 会自动对任务上下游数据库的配置、权限等进行前置检查。

- DDL 语法
  - 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
  - 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。
  - 关于分库分表合并场景的其它限制，参见使用限制。
- 操作限制

  - DM-worker 重启后不能自动恢复数据迁移任务，需要使用 dmctl 手动执行 `start-task`。详见管理数据迁移任务。
  - 在一些情况下，DM-worker 重启后不能自动恢复，需要手动处理。详见手动处理 Sharding DDL Lock。

- DM-worker 切换 MySQL

  - 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-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。

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。

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 slave relay log 类似，详见 Slave Relay Log。

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。

1.2.1.3 Load 处理单元

Load 处理单元读取 dump 处理单元的数据文件，然后加载到下游 TiDB。

1.2.1.4 Binlog replication/sync 处理单元

Binlog replication 处理单元（也就是 sync 处理单元）读取 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。

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。

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。

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';
```

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
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

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
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。

1.3 DM Relay Log

DM (Data Migration) 工具的 relay log 由一组有编号的文件和一个索引文件组成。这些有编号的文件包含了描述数据库更改的事件。索引文件包含所有使用过的 relay log 的文件名。

DM-worker 在启动后，会自动将上游 binlog 复制到本地配置目录 (若使用 DM-Ansible 部署 DM，则迁移目录默认为 `<deploy_dir> / relay_log`)。DM-worker 在运行过程中，会将上游 binlog 实时迁移到本地文件。DM-worker 的 sync 处理单元会实时读取本地 relay log 的 binlog 事件，将这些事件转换为 SQL 语句，再将 SQL 语句迁移到下游数据库。

本文档介绍 DM relay log 的目录结构、初始迁移规则和数据清理方法。

1.3.1 目录结构

Relay-log 本地存储的目录结构示例如下：

```
<deploy_dir>/relay_log/
```

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

- subdir：
  - DM-worker 把从上游数据库迁移到的 binlog 存储在同一目录下，每个目录都为一个 subdir。
  - subdir 的命名格式为 <上游数据库 UUID>.<本地 subdir 序列号>。
  - 在上游进行 master 和 slave 实例切换后，DM-worker 会生成一个序号递增的新 subdir 目录。
    - 在以上示例中，对于 7e427cc0-091c-11e9-9e45-72b7c59d52d7.000001 这一目录，7e427cc0-091c-11e9-9e45-72b7c59d52d7 是上游数据库的 UUID，000001 是本地 subdir 的序列号。
- server-uuid.index：记录当前可用的 subdir 目录。
- relay.meta：存储每个 subdir 中已迁移的 binlog 信息。例如，

```
cat c0149e17-dff1-11e8-b6a8-0242ac110004.000001/relay.meta
```

```
binlog-name = "mysql-bin.000010" # 当前迁移的 binlog 名
binlog-pos = 63083620            # 当前迁移的 binlog 位置
binlog-gtid = "c0149e17-dff1-11e8-b6a8-0242ac110004:1-3328" #
    ↪ 当前迁移的 binlog GTID
```

也可能包含多个 GTID：

```
cat 92acbd8a-c844-11e7-94a1-1866daf8accc.000001/relay.meta
```

```
binlog-name = "mysql-bin.018393"
binlog-pos = 277987307
binlog-gtid = "3ccc475b-2343-11e7-be21-6c0b84d59f30:1-14,406a3f61-690d
    ↪ -11e7-87c5-6c92bf46f384:1-94321383,53bfca22-690d-11e7-8a62-18
    ↪ ded7a37b78:1-495,686e1ab6-c47e-11e7-a42c-6c92bf46f384
    ↪ :1-34981190,03fc0263-28c7-11e7-a653-6c0b84d59f30:1-7041423,05474
    ↪ d3c-28c7-11e7-8352-203db246dd3d:1-170,10b039fc-c843-11e7-8f6a
    ↪ -1866daf8d810:1-308290454"
```

1.3.2 初始迁移规则

DM-worker 每次启动时（或在 DM-worker 暂停后 relay log 恢复迁移），迁移的起始位置会出现以下几种情况：

- 若是有效的本地 relay log（有效是指 relay log 具有有效的 `server-uuid.index`，`subdir` 和 `relay.meta` 文件），DM-worker 从 `relay.meta` 记录的位置恢复迁移。
- 若不存在有效的本地 relay log，而且 DM 配置文件中未指定 `relay-binlog-name` 或 `relay-binlog-gtid`：
  - 在非 GTID 模式下，DM-worker 从初始的上游 binlog 开始迁移，并将所有上游 binlog 文件连续迁移至最新。
  - 在 GTID 模式下，DM-worker 从初始上游 GTID 开始迁移。

> 注意：
> 若上游的 relay log 被清理掉，则会发生错误。在这种情况下，需设置 `relay-binlog-gtid` 来指定迁移的起始位置。

- 若不存在有效的本地 relay log：
  - 在非 GTID 模式下，若指定了 `relay-binlog-name`，则 DM-worker 从指定的 binlog 文件开始迁移。
  - 在 GTID 模式下，若指定了 `relay-binlog-gtid`，则 DM-worker 从指定的 GTID 开始迁移。

1.3.3 数据清理

因为存在文件读写的检测机制，所以 DM-worker 不会清理正在使用的 relay log，也不会清理当前已有数据迁移任务之后会使用到的 relay log。

Relay log 的数据清理包括自动清理和手动清理这两种方法。

1.3.3.1 自动数据清理

自动数据清理需对 DM-worker 命令行配置中的以下三项进行配置：

- `purge-interval`
  - 后台自动清理的时间间隔，以秒为单位。
  - 默认为 “3600”，表示每 3600 秒执行一次后台清理任务。
- `purge-expires`
  - 当前未由 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 在被后台清理前可保留的小时数。

  - 默认为 “0”，表示不按 relay log 的更新时间执行数据清理。

- purge-remain-space
  - 剩余磁盘空间，单位为 GB。若剩余磁盘空间小于该配置，则指定的 DM-worker 机器会在后台尝试自动清理可被安全清理的 relay-log。若这一数字被设为 “0”，则表示不按剩余磁盘空间来清理数据。
  - 默认为 “15”，表示可用磁盘空间小于 15GB 时，DM-master 会尝试安全地清理 relay log。

或者在 DM-woker 的配置文件中加入 purge 配置：

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

1.3.3.2 手动数据清理

手动数据清理是指使用 dmctl 提供的 purge-relay 命令，通过指定 subdir 和 binlog 文件名，来清理掉指定 **binlog** 之前的所有 relay log。若在命令中不填 -subdir 选项，则默认清理最新 **relay log** 子目录之前的所有 relay log。

假设当前 relay log 的目录结构如下：

```
tree .
```

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

```
cat server-uuid.index
```

```
deb76a2b-09cc-11e9-9129-5242cf3bb246.000001
e4e0e8ab-09cc-11e9-9220-82cc35207219.000002
deb76a2b-09cc-11e9-9129-5242cf3bb246.000003
```

在 dmctl 中使用 purge-relay 命令的示例如下：

- 以下命令指定的 relay log 子目录为 e4e0e8ab-09cc-11e9-9220-82cc35207219.000002，该子目录之前的 relay log 子目录为 deb76a2b-09cc-11e9-9129-5242cf3bb246.000001。所以该命令实际清空了 deb76a2b-09cc-11e9-9129-5242cf3bb246.000001 子目录，保留 e4e0e8ab-09cc-11e9-9220-82cc35207219.000002 和 deb76a2b-09cc-11e9-9129-5242cf3bb246.000003 子目录。

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

- 以下命令默认 --sub-dir 为最新的 deb76a2b-09cc-11e9-9129-5242cf3bb246.000003 子目录。该目录之前的 relay log 子目录为 deb76a2b-09cc-11e9-9129-5242cf3bb246.000001 和 e4e0e8ab-09cc-11e9-9220-82cc35207219.000002，所以该命令实际清空了这两个子目录，保留了 deb76a2b-09cc-11e9-9129-5242cf3bb246.000003。

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

2 核心特性

2.1 数据迁移功能

本文将详细介绍 DM 提供的数据迁移功能，以及相关的配置选项。

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。

2.1.1 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。

2.1.1.1 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"
```

2.1.1.2 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`。

2.1.1.3 使用示例

下面展示了三个不同场景下的配置示例。

2.1.1.3.1 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` 需要创建两个 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"
```

2.1.1.3.2 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"
```

2.1.1.3.3 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"
```

2.1.2 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。

2.1.2.1 参数配置

```
block-allow-list:          # 如果 DM 版本 <= v1.0.6 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"
```

2.1.2.2 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。
- ignore-tables：要迁移的表的黑名单，类似于 MySQL 中的 replicate-ignore-table。

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。

2.1.2.3 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` 是否应该被过滤的过滤流程如下：

1. 首先 **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：**
> 判断 schema test 是否被过滤，只进行 schema 过滤判断

2.1.2.4 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 <= v1.0.6 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 ↪ .messages_20 ↪	是	schema logs 没有匹配到 do-dbs 任意一项

table	是否过滤	过滤的原因
logs ↪ .messages_20[redacted]	是	schema logs 没有匹配到 do-dbs 任意一项
logs ↪ .messages_20[redacted]	是	schema logs 没有匹配到 do-dbs 任意一项
forum_backup_2016 ↪ .messages	是	1. schema forum_backup_2016 ↪ 没有匹配到 do-dbs 任意一项
forum_backup_2017 ↪ .messages	是	1. schema forum_backup_2017 ↪ 没有匹配到 do-dbs 任意一项
forum ↪ .users	是	1. schema forum 匹配到 do-dbs 进入 table 过滤 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
forum ↪ .messages	否	1. schema forum 匹配到 do-dbs 进入 table 过滤 2. schema 和 table 匹配到 do-tables 的 db-name: "~^ ↪ forum.*", ↪ tbl-name: ↪ "messages"

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

2.1.3 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。

2.1.3.1 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
```

2.1.3.2 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 语句进行以下规则过滤。

- `events`：binlog events 数组。

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 `"^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。

2.1.3.3 使用示例

2.1.3.3.1 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
```

2.1.3.3.2 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
```

2.1.3.3.3 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
```

2.1.3.3.4 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，以避免预期之外地过滤掉需要迁移的数据。

可设置如下规则过滤 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
```

2.1.4 Column mapping

> 注意：
>
> 由于 Column mapping 的使用限制较多，我们不推荐使用 Column mapping 功能作为首选方案。我们优先推荐的方案请参考自增主键冲突处理。

Column mapping 提供对表的列值进行修改的功能。可以根据不同的表达式对表的指定列做不同的修改操作，目前只支持 DM 提供的内置表达式。

注意：

- 不支持修改 column 的类型和表结构。
- 不支持对同一个表设置多个不同的列值转换规则。

2.1.4.1 参数配置

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

2.1.4.2 参数解释

- schema-pattern/table-pattern：对匹配上该规则的上游 MySQL/MariaDB 实例的表按照指定 expression 进行列值修改操作。
- source-column，target-column：对 source-column 列的值按照指定 expression 进行修改，将修改后的值赋值给 target-column。
- expression：对数据进行转换的表达式，目前只支持下面的内置计算表达式。

2.1.4.2.1 partition id 表达式

partition id 目的是为了解决分库分表合并迁移的自增主键的冲突。

partition id 限制

注意下面的限制：

- 只支持类型为 bigint 的列，通常为自增主键，联合主键或者联合唯一索引的其中一列
- 如果 schema 前缀 不为空，则库名的组成必须为 schema 前缀 或者 schema 前缀 + 分隔符 + 数字（即 schema ID），例如：支持 s 和 s_1，不支持 s_a

- 如果 `table 前缀` 不为空，则表名的组成必须为 `table 前缀` 或者 `table 前缀 + 分隔符 + 数字`（即 `table ID`）
- 如果库名/表名不包含 `… + 分隔符 + 数字` 部分，则对应的 ID 默认为 0
- 对分库分表的规模支持限制如下
  - 支持最多 16 个 MySQL/MariaDB 实例，且需要满足 0 <= instance ID <= 15。
  - 每个实例支持最多 128 个 schema，且需要满足 0 <= schema ID <= 127。
  - 每个实例的每个 schema 支持最多 256 个 table，且需要满足 0 <= table ID <= 255。
  - 进行 Column mapping 的列的范围需要满足 0 <= ID <= 17592186044415。
  - `{instance ID, schema ID, table ID}` 组合需要保持唯一。

`partition id` 参数配置

用户需要在 arguments 里面按顺序设置以下三个或四个参数：

- `instance_id`：客户指定的上游分库分表的 MySQL/MariaDB instance ID（0 <= instance ID <= 15）
- `schema 前缀`：用来解析库名并获取 `schema ID`
- `table 前缀`：用来解释表名并获取 `table ID`
- 分隔符：用来分隔前缀与 ID，可省略，省略时分隔符默认为空字符串

`instance_id`、`schema 前缀` 和 `table 前缀` 这三个参数均可被设置为空字符串（`""`），表示对应的部分不会被编码进 `partition id`。

`partition id` 表达式规则

`partition id` 会用 arguments 里面的数字来填充自增主键 ID 的首个比特位，计算出来一个 int64（即 MySQL bigint）类型的值，具体规则如下：

instance_id	schema 前缀	table 前缀					
已定义	已定义	已定义	[S: 1 比特位]	[I: 4 比特位]	[D: 7 比特位]	[T: 8 比特位]	[P: 44 比
空	已定义	已定义		[S: 1 比特位]	[D: 7 比特位]	[T: 8 比特位]	[P: 48 比
已定义	空	已定义		[S: 1 比特位]	[I: 4 比特位]	[T: 8 比特位]	[P: 51 比
已定义	已定义	空		[S: 1 比特位]	[I: 4 比特位]	[D: 7 比特位]	[P: 52 比
空	空	已定义			[S: 1 比特位]	[T: 8 比特位]	[P: 55 比
空	已定义	空			[S: 1 比特位]	[D: 7 比特位]	[P: 56 比
已定义	空	空			[S: 1 比特位]	[I: 4 比特位]	[P: 59 比

- `S`：符号位，保留
- `I`：instance ID，默认 4 比特位
- `D`：schema ID，默认 7 比特位
- `T`：table ID，默认 8 比特位
- `P`：自增主键 ID，占据剩下的比特位（ 44 比特位）

2.1.4.3 使用示例

假设存在分库分表场景：将上游两个 MySQL 实例的 `test_{1,2,3...}.t_{1,2,3...}` 迁移到下游 TiDB 的 `test.t`，并且这些表都有自增主键。

需要设置下面两个规则：

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

- MySQL instance 1 的表 `test_1.t_1` 的 `ID = 1` 的行经过转换后 ID = 1 变为 `1 << (64-1-4)| 1 << (64-1-4-7)| 1 << 44 | 1 = 580981944116838401`
- MySQL instance 2 的表 `test_1.t_2` 的 `ID = 1` 的行经过转换后 ID = 2 变为 `2 << (64-1-4)| 1 << (64-1-4-7)| 2 << 44 | 2 = 1157460288606306306`

2.1.5 迁移延迟监控

DM 支持通过 heartbeat 真实迁移数据来计算每个迁移任务与 MySQL/MariaDB 的实时迁移延迟。

> **注意：**
>
> - 迁移延迟的估算的精度在秒级别。
> - heartbeat 相关的 binlog 不会迁移到下游，在计算延迟后会被丢弃。

2.1.5.1 系统权限

如果开启 heartbeat 功能，需要上游 MySQL/MariaDB 实例提供下面的权限：

- SELECT

- INSERT
- CREATE (databases, tables)
- DELETE

2.1.5.2 参数配置

在 task 的配置文件中设置：

```
enable-heartbeat: true
```

2.1.5.3 原理介绍

- DM-worker 在对应的上游 MySQL/MariaDB 创建库 `dm_heartbeat`（当前不可配置）
- DM-worker 在对应的上游 MySQL/MariaDB 创建表 `heartbeat`（当前不可配置）
- DM-worker 每秒钟（当前不可配置）在对应的上游 MySQL/MariaDB 的 `dm_heartbeat.heartbeat` 表中，利用 `replace statement` 更新当前时间戳 `TS_master`
- DM-worker 每个任务拿到 `dm_heartbeat.heartbeat` 的 binlog 后，更新自己的迁移时间 `TS_slave_task`
- DM-worker 每 10 秒在对应的上游 MySQL/MariaDB 的 `dm_heartbeat.heartbeat` 查询当前的 `TS_master`，并且对每个任务计算 `task_lag` = `TS_master` - `TS_slave_task`

可以在 metrics 的binlog replication 处理单元找到 replicate lag 监控项。

2.2 DM online-ddl-scheme

2.2.1 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，但是也有或多或少的限制。比如 MDL 锁的获取，某些 DDL 还是需要以 Copy 的方式来进行，在生产业务使用中，DDL 执行过程中的锁表会一定程度上阻塞数据库的读取或者写入。

因此，gh-ost 以及 pt-osc 可以更优雅地在 MySQL 上面执行 DDL，把对读写的影响降到最低。

TiDB 根据 Google F1 的在线异步 schema 变更算法实现，在 DDL 过程中并不会阻塞读写。因此，在 online-schema-change 过程中，gh-ost 和 pt-osc 所产生的大量中间表数据以及 binlog event，在 MySQL 与 TiDB 的数据迁移过程中并不需要。

DM 是 MySQL 到 TiDB 的数据迁移工具，online-ddl-scheme 功能就是对上述两个 online-schema-change 的工具进行特殊的处理，以便更快完成所需的 DDL 迁移。

如果想从源码方面了解 DM online-ddl-scheme，可以参考 DM 源码阅读系列文章（八） Online Schema Change 迁移支持

2.2.2 配置

online-ddl-scheme 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"
is-sharding: true           # 是否为分库分表合并任务
meta-schema: "dm_meta"      # 下游储存 `meta` 信息的数据库
remove-meta: false          # 是否在开始运行任务前移除该任务名对应的 `meta
    ↪ `（`checkpoint` 和 `onlineddl` 等）。
enable-heartbeat: false     # 是否开启 `heartbeat` 功能
online-ddl-scheme: "gh-ost" # 目前仅支持 gh-ost 、pt

target-database:            # 下游数据库实例配置
  host: "192.168.0.1"
  port: 4000
  user: "root"
  password: ""              # 如果不为空则需经过 dmctl 加密
```

2.2.3 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 表中数据同步到与 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。

2.2.4 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。

2.3 Shard Support

2.3.1 分库分表合并迁移

本文介绍了 DM 提供的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。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。

> 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 `is-sharding: true`。

2.3.1.1 使用限制

DM 进行分表 DDL 的迁移有以下几点使用限制：

- 在一个逻辑 sharding group（需要合并迁移到下游同一个表的所有分表组成的 group）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。
- 对于每个逻辑 sharding group，推荐使用一个独立的任务进行迁移。
  - 如果一个任务内存在多个 sharding group，则必须等待一个 sharding group 的 DDL 语句迁移完成后，才能开始对其他 sharding group 执行 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。
- 增量复制任务需要确认开始复制的 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 语句能复制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复制。

- 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。

2.3.1.2 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 复制过程中，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：

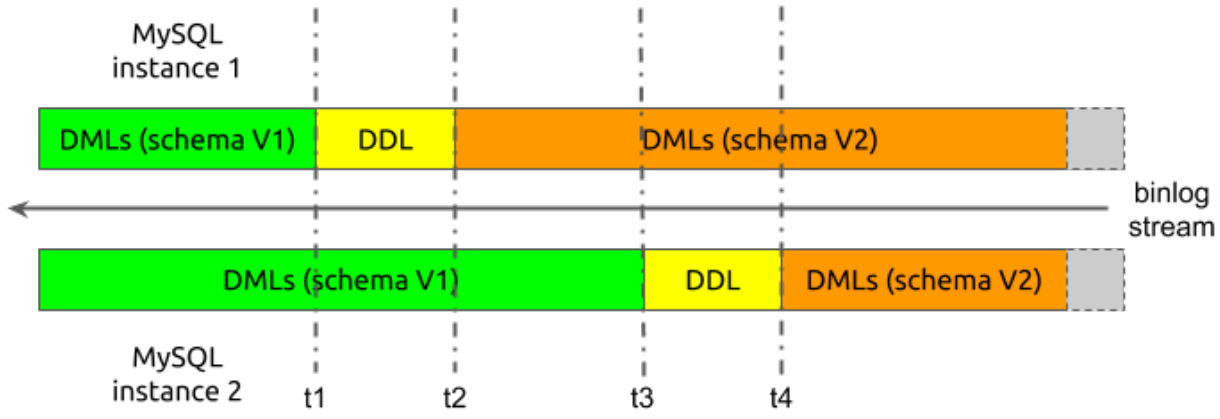


Figure 2: shard-ddl-example-1

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。

2.3.1.3 实现原理

基于上述例子，本部分介绍了 DM 在合库合表过程中进行 DDL 迁移的实现原理。

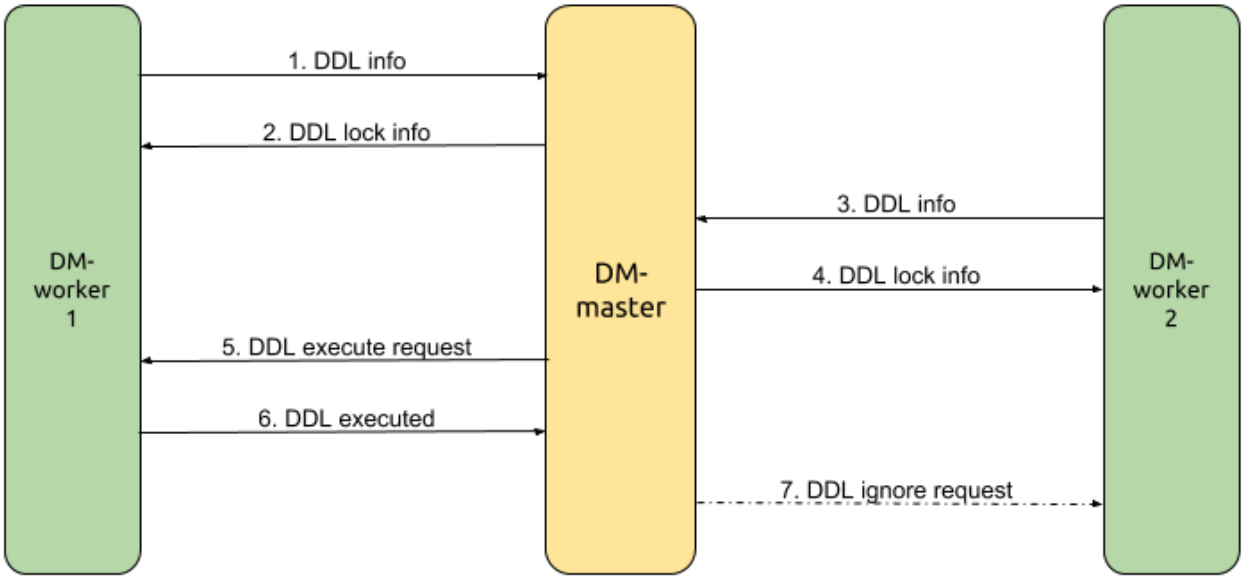


Figure 3: shard-ddl-flow

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：

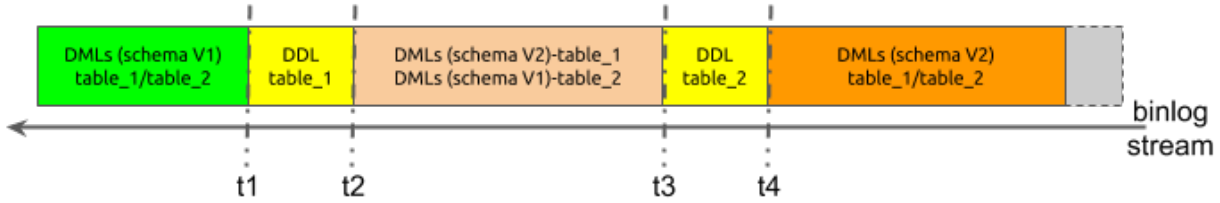


Figure 4: shard-ddl-example-2

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。

2.3.2 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/break-ddl-lock` 手动处理异常的 DDL lock。

> 注意：
>
> - 不要轻易使用 `unlock-ddl-lock/break-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。

2.3.2.1 命令介绍

2.3.2.1.1 `show-ddl-locks`

该命令用于查询当前 DM-master 上存在的 DDL lock 信息。

命令示例

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

参数解释

- `worker`：
  - flag 参数，string，`--worker`，可选
  - 不指定时，查询所有 DM-worker 相关的 lock 信息；指定时，仅查询与这组 DM-worker 相关的 lock 信息，可重复多次指定
- `task-name`：
  - 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

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": [                                          # DM-master 上存在的
        ↪ 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
            "owner": "127.0.0.1:8262",                  # lock 的 owner（
                ↪ 第一个遇到该 DDL 的 DM-worker）
            "DDLs": [                                   # lock 对应的 DDL 列表
                "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 DM-worker 列表
                "127.0.0.1:8262"
            ],
            "unsynced": [                               # 尚未收到对应 MySQL
                ↪ 实例内所有分表 DDL 的 DM-worker 列表
                "127.0.0.1:8263"
            ]
        }
    ]
}
```

2.3.2.1.2 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。

命令示例

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

参数解释

- worker：
  - flag 参数，string，--worker，可选

  - 不指定时，对所有已经在等待该 lock 的 DM-worker 发起跳过 DDL 操作请求；指定时，仅对这组 DM-worker 发起跳过 DDL 操作请求，可重复多次指定

- `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 DM-worker（替代默认的 owner）执行 DDL 操作

- `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`）

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
    "workers": [                                    # 各 DM-worker 执行/跳过
        ↪  DDL 操作结果信息列表
        {
            "result": true,                         # 该 DM-worker 执行/跳过
                ↪  DDL 操作是否成功
            "worker": "127.0.0.1:8262",             # DM-worker 地址 (DM-
                ↪ worker ID)
            "msg": ""                               # DM-worker 执行/跳过
                ↪ DDL 失败时的原因
        }
    ]
}
```

2.3.2.1.3 break-ddl-lock

该命令用于主动请求 DM-worker 强制打破当前正在等待 unlock 的 DDL lock，包括请求 DM-worker 执行或跳过 DDL 操作、移除该 DM-worker 上的 DDL lock 信息。

命令示例

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

参数解释

- worker：
  - flag 参数，string，--worker，必选
  - 指定需要执行 break 操作的 DM-worker
- remove-id：已废弃
- exec：
  - flag 参数，boolean，--exec，可选
  - 不可与 --skip 参数同时指定
  - 指定时，请求该 DM-worker 执行（execute）当前 lock 对应的 DDL
- skip：
  - flag 参数，boolean，--skip，可选
  - 不可与 --exec 参数同时指定
  - 指定时，请求该 DM-worker 跳过（skip）当前 lock 对应的 DDL
- task-name：
  - 非 flag 参数，string，必选
  - 指定要执行 break 操作的 lock 所在的 task 名称（要查看各 task 上是否存在 lock，可通过query-status 获得）

返回结果示例

```
break-ddl-lock -w 127.0.0.1:8262 --exec test
```

```
{
    "result": true,                                     # break lock
        ↪ 操作是否成功
    "msg": "",                                          # break lock
        ↪ 操作失败时的原因
    "workers": [                                        # 执行 break lock 操作的
        ↪  DM-worker 列表（当前单次操作仅支持对一个 DM-worker 执行 break lock
        ↪ ）
        {
            "result": false,                            # 该 DM-worker break
                ↪ lock 操作是否成功
```

```
            "worker": "127.0.0.1:8262",                 # 该 DM-worker 地址 (DM-
                ↪ worker ID)
            "msg": ""                                   # DM-worker break lock
                ↪ 失败时的原因
        }
    ]
}
```

2.3.2.2 支持场景

目前，使用 `unlock-ddl-lock/break-ddl-lock` 命令仅支持处理以下三种 sharding DDL lock 异常情况。

2.3.2.2.1 场景一：部分 DM-worker 下线

Lock 异常原因

在 DM-master 尝试自动 unlock sharding DDL lock 之前，需要等待所有 DM-worker 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，且有部分 DM-worker 下线，并且不再计划重启它们（按业务需求移除了这部分 DM-worker），则会由于永远无法等齐所有的 DDL 而造成 lock 无法自动 unlock。

手动处理示例

假设上游有 MySQL-1 和 MySQL-2 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-1 对应的 DM-worker-1 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-1 接受到两个分表的 DDL 之后，将对应 MySQL-1 相关的 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",
            "owner": "127.0.0.1:8262",
            "DDLs": [
                "USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` ADD
                    ↪  COLUMN `c2` int(11);"
            ],
            "synced": [
                "127.0.0.1:8262"
            ],
            "unsynced": [
                "127.0.0.1:8263"
            ]
        }
    ]
}
```

4. 由于业务需要，DM-worker-2 对应的 MySQL-2 的数据不再需要迁移到下游 TiDB，对 DM-worker-2 执行了下线处理。

5. DM-master 上 ID 为 `test-`shard_db`.`shard_table`` 的 lock 无法等到 DM-worker-2 的 DDL 操作信息。

   `show-ddl-locks` 返回的 `unsynced` 中一直包含 DM-worker-2 的信息 (`127.0.0.1:8263` ↪)。

6. 使用 `unlock-ddl-lock` 来请求 DM-master 主动 unlock 该 DDL lock。

   - 如果 DDL lock 的 owner 也已经下线，可以使用 `--owner` 参数指定其他 DM-worker 作为新 owner 来执行 DDL。

- 当存在任意 DM-worker 报错时，`result` 将为 `false`，此时请仔细检查各 DM-worker 的错误是否是预期可接受的。
  - 已下线的 DM-worker 会返回 `rpc error: code = Unavailable` 错误属于预期行为，可以忽略。
  - 如果其它未下线的 DM-worker 返回错误，则需要根据情况额外处理。

```
unlock-ddl-lock test-`shard_db`.`shard_table`
```

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

7. 使用 `show-ddl-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。

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 DM-worker，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 DM-worker 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 DM-worker 重新上线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。

2.3.2.2.2 场景二：unlock 过程中部分 DM-worker 重启

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 DM-worker 跳过 DDL 操作并更新对应分表的 checkpoint。

上述 unlock DDL lock 的操作不是原子的，当 owner 执行 DDL 操作成功后，请求其他 DM-worker 跳过 DDL 操作时，如果该 DM-worker 发生了重启，则会造成该 DM-worker 跳过 DDL 操作失败。

此时 DM-master 上的 lock 信息被移除，但该 DM-worker 重启后将尝试继续重新迁移该 DDL 操作。但是，由于其他 DM-worker（包括原 owner）已经迁移完该 DDL 操作，并已经在继续后续的迁移，该 DM-worker 将永远无法等待该 DDL 操作对应 lock 的自动 unlock。

手动处理示例

仍然假设是部分 DM-worker 下线 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 (DM-worker-1) 成功执行了 DDL 操作且开始继续进行后续迁移，并移除了 DM-master 上的 DDL lock 信息；但在请求 DM-worker-2 跳过 DDL 操作的过程中，由于 DM-worker-2 发生了重启而跳过 DDL 操作失败。

DM-worker-2 重启后，将尝试重新迁移重启前已经在等待的 DDL lock。此时将在 DM-master 上创建一个新的 lock，并且该 DM-worker 将成为 lock 的 owner（其他 DM-worker 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。

   应该仅有该重启的 DM-worker（`127.0.0.1:8263`）处于 `synced` 状态：

```
show-ddl-locks
```

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

```
    ]
}
```

2. 使用 `unlock-ddl-lock` 请求 DM-master unlock 该 lock。

    - 使用 `--worker` 参数限定操作仅针对该重启的 DM-worker（`127.0.0.1:8263`）。
    - Lock 过程中该 DM-worker 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 --worker=127.0.0.1:8263 test-`shard_db`.`
    ↪ shard_table`
```

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "127.0.0.1:8263",
            "msg": ""
        }
    ]
}
```

3. 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。

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。

2.3.2.2.3 场景三：unlock 过程中部分 DM-worker 临时不可达

Lock 异常原因

与unlock 过程中部分 DM-worker 重启 造成 lock 异常的原因类似。当请求 DM-worker 跳过 DDL 操作时，如果该 DM-worker 临时不可达，则会造成该 DM-worker 跳过 DDL 操作失败。此时 DM-master 上的 lock 信息被移除，但该 DM-worker 将处于等待一个不再存在的 DDL lock 的状态。

场景三与场景二的区别在于，场景三中 DM-master 没有 lock，而场景二中 DM-master 有一个新的 lock。

手动处理示例

仍然假设是部分 DM-worker 下线 示例中的上下游表结构及合表迁移需求。

在 DM-master 自动执行 unlock 操作的过程中，owner (DM-worker-1) 成功执行了 DDL 操作且开始继续进行后续迁移，并移除了 DM-master 上的 DDL lock 信息，但在请求 DM-worker-2 跳过 DDL 操作的过程中，由于网络原因等临时不可达而跳过 DDL 操作失败。

处理流程如下：

1. 使用 `show-ddl-locks` 确认 DM-master 上不再存在该 DDL 操作对应的 lock。
2. 使用 `query-status` 确认 DM-worker 仍在等待 lock 迁移。

```
query-status test
```

```
{
    "result": true,
    "msg": "",
    "workers": [
        ...
        {
            ...
            "worker": "127.0.0.1:8263",
            "subTaskStatus": [
                {
                    ...
                    "unresolvedDDLLockID": "test-`shard_db`.`shard_table
                        ↪ `",
                    "sync": {
                        ...
                        "blockingDDLs": [
                            "USE `shard_db`; ALTER TABLE `shard_db`.`
                                ↪ shard_table` ADD COLUMN `c2` int(11);"
                        ],
                        "unresolvedGroups": [
                            {
                                "target": "`shard_db`.`shard_table`",
                                "DDLs": [
                                    "USE `shard_db`; ALTER TABLE `shard_db
                                        ↪ `.`shard_table` ADD COLUMN `c2`
                                        ↪ int(11);"
                                ],
                                "firstPos": "(mysql-bin|000001.000003,
                                    ↪ 1752)",
                                "synced": [
                                    "`shard_db_2`.`shard_table_1`",
                                    "`shard_db_2`.`shard_table_2`"
                                ],
```

```
                        "unsynced": [
                        ]
                    }
                ],
                "synced": false
            }
        }
    ]
    ...
  }
 ]
}
```

3. 使用 `break-ddl-lock` 请求强制 break 该 DM-worker 当前正在等待的 DDL lock。由于 owner 已经向下游执行了 DDL 操作，因此在 break 时使用 `--skip` 参数。

```
break-ddl-lock --worker=127.0.0.1:8263 --skip test
```

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "127.0.0.1:8263",
            "msg": ""
        }
    ]
}
```

4. 使用 `query-status` 确认迁移任务是否正常且不再处于等待 lock 的状态。

手动处理后的影响

手动强制 break lock 后，后续 sharding DDL 将可以自动正常迁移。

3 Benchmark

3.1 DM 1.0-GA 性能测试报告

本报告记录了对 1.0-GA 版本的 DM 进行性能测试的目的、环境、场景和结果。

3.1.1 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。

3.1.2 测试环境

3.1.2.1 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
172.16.4.39	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
172.16.4.40	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
172.16.4.41	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
172.16.4.42	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
172.16.4.43	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4
172.16.4.44	CentOS Linux release 7.6.1810	3.10.0-957.1.3.el7.x86_64	ext4

硬件信息：

类别	指标
CPU	40 vCPUs, Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz
内存	192GB, 12 条 16GB DIMM DDR4 2133 MHz
磁盘	Intel DC P4510 4TB NVMe PCIe 3.0
网卡	万兆网卡

其他：

- 服务器间网络延迟：rtt min/avg/max/mdev = 0.074/0.088/0.121/0.019 ms

3.1.2.2 集群拓扑

机器 IP	部署的服务
172.16.4.39	PD1, DM-worker1, DM-master
172.16.4.40	PD2, MySQL1
172.16.4.41	PD3, TiDB
172.16.4.42	TiKV1
172.16.4.43	TiKV2
172.16.4.44	TiKV3

3.1.2.3 各服务版本信息

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

3.1.3 测试场景

3.1.3.1 迁移数据流

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

3.1.3.2 公共配置信息

3.1.3.2.1 迁移数据表结构

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

3.1.3.2.2 数据库配置

使用 TiDB Ansible 部署 TiDB 测试集群，所有配置使用 TiDB Ansible 提供的默认配置。

3.1.3.3 全量导入性能测试用例

3.1.3.3.1 测试过程

- 部署测试环境
- 使用 sysbench 在上游创建测试表，并生成全量导入的测试数据
- 在 full 模式下启动 DM 迁移任务

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
```

3.1.3.3.2 全量导入性能测试结果

在 mydumpers 中使用 --rows 选项可以开启单表多线程并发导出（当前 mydumpers 在 MySQL 的单表并发会优先选出一列做拆分基准，选择优先级为主键 > 唯一索引 > 普通索引，选出目标列后需保证该列为 INT 类型；针对 TiDB 的并发导出则会优先尝试 _tidb_rowid 列），以下的第一项测试用于验证开启单表并发导出可以提高数据导出性能。

测试项	导出线程数	mydumpers extra-args 参数	导出速度 (MB/s)
开启单表并发导出	32	“–rows 320000 –regex ’^sbtest.*’“	191.03
不开启单表并发导出	4	“–regex ’^sbtest.*’“	72.22

测试项	事务执行延迟 (s)	每条插入语句包含的行数	导入数据量 (GB)	导入时间 (s)	导入速度 (
load data	1.737	4878	38.14	2346.9	16.64

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

该测试中全量导入的数据量为 3.78 GB，使用以下 sysbench 命令生成：

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

load 处理单元 pool size	事务执行时间 (s)	导入时间 (s)	导入速度 (MB/s)	TiDB 99 duration (s)
2	0.250	425.9	9.1	0.23
4	0.523	360.1	10.7	0.41
8	0.986	267.0	14.5	0.93
16	2.022	265.9	14.5	2.68
32	3.778	262.3	14.7	6.39
64	7.452	281.9	13.7	8.00

3.1.3.3.4 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 dump 处理单元的 --statement-size 来控制。

每条语句中包含的行数	mydumpers extra-args 参数	事务执行时间 (s)	导入时间 (s)	导入速度 (MB/s)	TiDB 99 duration (s)
7426	-s 1500000 -r 320000	6.982	258.3	15.0	10.34
4903	-s 1000000 -r 320000	3.778	262.3	14.7	6.39
2470	-s 500000 -r 320000	1.962	271.36	14.3	2.00

每条语句中包含的行数	mydumpers extra-args 参数	事务执行时间 (s)	导入时间 (s)	导入速度 (MB/s)	TiDB 99 duration (s)
1236	-s 250000 -r 320000	1.911	283.3	13.7	1.50
618	-s 125000 -r 320000	0.683	299.9	12.9	0.73
310	-s 62500 -r 320000	0.413	322.6	12.0	0.49

3.1.3.4 增量复制性能测试用例

3.1.3.4.1 测试过程

- 部署测试环境
- 使用 sysbench 在上游创建测试表，并生成全量导入的测试数据
- 在 all 模式下启动 DM 迁移任务，等待迁移任务进入 sync 迁移阶段
- 使用 sysbench 在上游持续生成增量数据，通过 query-status 命令观测 DM 的复制状态，通过 Grafana 观测 DM 和 TiDB 的监控指标。

3.1.3.4.2 增量复制性能测试结果

上游 sysbench 生成增量数据命令

```
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
```

该性能测试中迁移任务 sync 处理单元 worker-count 设置为 32，batch 大小设置为 100。

组件	qps	tps	95
MySQL	42.79k	42.79k	1
DM relay log unit	-	11.3MB/s	45us (bi
DM binlog replication unit	22.97k (binlog event 接收 qps，不包括忽略的 event)	-	20ms (事
TiDB	31.30k (Begin/Commit 3.93k Insert 22.76k)	4.16k	95%: 6.4

3.1.3.4.3 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 内部 job tps	DM 事务执行时间 (ms)	TiDB qps	TiDB 99 dura
4	7074	63	7.1k	3
8	14684	64	14.9k	4
16	23486	56	24.9k	6

sync 处理单元 worker-count 数	DM 内部 job tps	DM 事务执行时间 (ms)	TiDB qps	TiDB 99 dura
32	23345	28	29.2k	10
64	23302	30	31.2k	16
1024	22225	70	56.9k	70

3.1.3.4.4 不同数据分布的增量复制性能测试对比

sysbench 语句类型	DM relay log 持久化速率 (MB/s)	DM 内部 job tps	DM 事务执行时间 (ms)	TiDB qps	TiDB 99 duration (ms)
insert_only	11.3	23345	28	29.2k	10
write_only	18.7	33470	129	34.6k	11

3.1.4 推荐迁移任务参数配置

3.1.4.1 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）。

3.1.4.2 load 处理单元

推荐 `pool-size` 设置为 16。

3.1.4.3 sync 处理单元

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

4 使用场景

4.1 Data Migration 简单使用场景

本文介绍了 DM 工具的一个简单使用场景（非分库分表合并场景）：将三个上游 MySQL 实例的数据迁移到一个下游 TiDB 集群中。

4.1.1 上游实例

假设上游结构为：

- 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
log	messages

4.1.2 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 `log` 表的任何数据。
2. 将上游 `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。

4.1.3 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

4.1.4 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:
    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
```

- 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:
    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 Table Lists 规则：

```
block-allow-list:  # 如果 DM 版本 <= v1.0.6 则使用 black-white-list
  log-ignored:
    ignore-dbs: ["log"]
```

4.1.5 迁移任务配置

以下是完整的迁移任务配置，详见配置介绍。

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

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

mysql-instances:
  -
```

```
    source-id: "instance-1"
    route-rules: ["instance-1-user-rule"]
    filter-rules: ["log-filter-rule", "user-filter-rule" , "store-filter-
      ↪ rule"]
    block-allow-list: "log-ignored" # 如果 DM 版本 <= v1.0.6 则使用 black-
      ↪ white-list
    mydumper-config-name: "global"
    loader-config-name: "global"
    syncer-config-name: "global"
  -
    source-id: "instance-2"
    route-rules: ["instance-2-user-rule", instance-2-store-rule]
    filter-rules: ["log-filter-rule", "user-filter-rule" , "store-filter-
      ↪ rule"]
    block-allow-list: "log-ignored" # 如果 DM 版本 <= v1.0.6 则使用 black-
      ↪ white-list
    mydumper-config-name: "global"
    loader-config-name: "global"
    syncer-config-name: "global"
  -
    source-id: "instance-3"
    route-rules: ["instance-3-user-rule", instance-3-store-rule]
    filter-rules: ["log-filter-rule", "user-filter-rule" , "store-filter-
      ↪ rule"]
    block-allow-list: "log-ignored" # 如果 DM 版本 <= v1.0.6 则使用 black-
      ↪ white-list
    mydumper-config-name: "global"
    loader-config-name: "global"
    syncer-config-name: "global"

## 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

block-allow-list:  # 如果 DM 版本 <= v1.0.6 则使用 black-white-list
  log-ignored:
    ignore-dbs: ["log"]

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

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

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

4.2 DM 分库分表合并场景

本文介绍如何在分库分表合并场景中使用 Data Migration (DM)。使用场景中，三个上游 MySQL 实例的分库和分表数据需要迁移至下游 TiDB 集群。

4.2.1 上游实例

假设上游库结构如下：

- 实例 1

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

- 实例 2

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

- 实例 3

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

4.2.2 迁移需求

1. 同名表合并场景，比如将三个实例中的 `user.information` 表合并至下游 TiDB 中的 `user.information` 表。
2. 不同名表合并场景，比如将三个实例中的 `user.log_{north|south|east}` 表合并至下游 TiDB 中的 `user.log_{north|south|east}` 表。
3. 分片表合并场景，比如将三个实例中的 `store_{01|02}.sale_{01|02}` 表合并至下游 TiDB 中的 `store.sale` 表。
4. 过滤删除操作场景，比如过滤掉三个实例中 `user.log_{north|south|east}` 表的所有删除操作。
5. 过滤删除操作场景，比如过滤掉三个实例中 `user.information` 表的所有删除操作。

6. 过滤删除操作场景，比如过滤掉三个实例中 store_{01|02}.sale_{01|02} 表的所有删除操作。
7. 使用通配符过滤特定表的场景，比如使用通配符 user.log_* 过滤掉三个实例的 user.log_bak 表。
8. 主键冲突处理场景，假设 store_{01|02}.sale_{01|02} 表带有 bigint 型的自增主键，将其合并至 TiDB 时会引发冲突，可以使用相应的方案来避免冲突。

4.2.3 下游实例

假设迁移后下游库结构如下：

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

4.2.4 迁移方案

- 要满足迁移需求 #1 和 #2，配置Table routing 规则 如下：

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

- 要满足迁移需求 #3，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 #4 和 #5，配置Binlog event filter 规则 如下：

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

```
    action: Ignore
```

> **注意：**
> 迁移需求 #4、#5 的操作意味着过滤掉所有对 user 库的删除操作，所以此处配置了库级别的过滤规则，user 库以后参与复制的表的所有删除操作也都会被过滤。

- 要满足迁移需求 #6，配置Binlog event filter 规则 如下：

```
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
```

- 要满足迁移需求 #7，配置Block & Allow Lists 如下：

```
block-allow-list:  # 如果 DM 版本 <= v1.0.6 则使用 black-white-list
  log-bak-ignored:
    ignore-tables:
    - db-name: "user"
      tbl-name: "log_bak"
```

- 要满足迁移需求 #8，首先参考自增主键冲突处理来解决冲突，保证在迁移到下游时不会因为分表中有相同的主键值而使迁移出现异常，然后需要配置 ignore-checking-items 来跳过自增主键冲突的检查：

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

4.2.5 迁移任务配置

迁移任务的完整配置如下。详情请参阅Data Migration 任务配置文件。

```
name: "shard_merge"
task-mode: all
meta-schema: "dm_meta"
remove-meta: false
```

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

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

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

  -
    source-id: "instance-2"
    route-rules: ["user-route-rule", "store-route-rule", "sale-route-rule"]
    filter-rules: ["user-filter-rule", "store-filter-rule", "sale-filter-
       ↪ rule"]
    block-allow-list: "log-bak-ignored" # 如果 DM 版本 <= v1.0.6 则使用
       ↪ black-white-list
    mydumper-config-name: "global"
    loader-config-name: "global"
    syncer-config-name: "global"
  -
    source-id: "instance-3"
    route-rules: ["user-route-rule", "store-route-rule", "sale-route-rule"]
    filter-rules: ["user-filter-rule", "store-filter-rule", "sale-filter-
       ↪ rule"]
    block-allow-list: "log-bak-ignored" # 如果 DM 版本 <= v1.0.6 则使用
       ↪ black-white-list
    mydumper-config-name: "global"
    loader-config-name: "global"
    syncer-config-name: "global"

## 所有实例共享的其他通用配置

routes:
  user-route-rule:
```

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

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

block-allow-list:    # 如果 DM 版本 <= v1.0.6 则使用 black-white-list
  log-bak-ignored:
    ignore-tables:
    - db-name: "user"
      tbl-name: "log_bak"

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

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

syncers:
  global:
```

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

4.3 分表合并数据迁移最佳实践

本文阐述了使用 TiDB Data Migration（以下简称 DM）对分库分表进行合并迁移的场景中，DM 相关功能的支持和限制，旨在给出一个业务的最佳实践。

4.3.1 独立的数据迁移任务

在分库分表合并迁移的实现原理部分，我们介绍了 sharding group 的概念，简单来说可以理解为需要合并到下游同一个表的所有上游表即组成一个 sharding group。

当前的 sharding DDL 算法为了能协调在不同分表执行 DDL 对 schema 变更的影响，加入了一些使用限制。而当这些使用限制由于某些异常原因被打破时，我们需要手动处理 Sharding DDL Lock 甚至是完整重做整个数据迁移任务。

因此，为了减小异常发生时对数据迁移的影响，我们推荐将每一个 sharding group 拆分成一个独立的数据迁移任务。这样当异常发生时，可能只有少部分迁移任务需要进行手动处理，其他数据迁移任务可以不受影响。

4.3.2 手动处理 sharding DDL lock

从分库分表合并迁移的实现原理部分我们可以知道，DM 中的 sharding DDL lock 是用于协调不同上游分表向下游执行 DDL 的一种机制，本身并不是异常。

因此，当通过 `show-ddl-locks` 查看到 DM-master 上存在 sharding DDL lock 时，或通过 `query-status` 查看到某些 DM-worker 有 `unresolvedGroups` 或 `blockingDDLs` 时，并不要急于使用 `unlock-ddl-lock` 或 `break-ddl-lock` 尝试去手动解除 sharding DDL lock。

只有在确认当前未能自动解除 sharding DDL lock 是文档中所列的支持场景之一时，才能参照对应支持场景中的手动处理示例进行处理。对于其他未被支持的场景，我们建议完整重做整个数据迁移任务，即清空下游数据库中的数据以及该数据迁移任务相关的 `dm_meta` 信息后，重新执行全量数据及增量数据的迁移。

4.3.3 自增主键冲突处理

在 DM 中，我们提供了Column mapping 用于处理 bigint 类型的自增主键在合并时可能出现冲突的问题，但我们强烈不推荐使用该功能。如果业务上允许，我们推荐使用以下两种处理方式。

4.3.3.1 去掉自增主键的主键属性

假设上游分表的表结构如下：

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

如果满足下列条件：

- auto_pk_c1 列对业务无意义，且不依赖该列的 PRIMARY KEY 属性。
- uk_c2 列有 UNIQUE KEY 属性，且能保证在所有上游分表间全局唯一。

则可以用以下步骤处理合表时可能由 auto_pk_c1 导致的 ERROR 1062 (23000): Duplicate entry '***' for key 'PRIMARY' 问题：

1. 在开始执行全量数据迁移前，在下游数据库创建用于合表迁移的表，但将 auto_pk_c1 的 PRIMARY KEY 属性修改为普通索引。

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

2. 在 task.yaml 文件中增加如下配置跳过自增主键冲突检查：

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

3. 启动数据迁移任务，执行全量与增量数据迁移。

4. 通过 query-status 验证数据迁移任务是否正常，在下游数据库中验证合表中是否已经存在了来自上游的数据。

4.3.3.2 使用联合主键

假设上游分表的表结构如下：

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

如果满足下列条件：

- 业务不依赖 auto_pk_c1 的 PRIMARY KEY 属性。
- auto_pk_c1 与 uuid_c2 的组合能确保全局唯一。
- 业务能接受将 auto_pk_c1 与 uuid_c2 组成联合 PRIMARY KEY。

则可以用以下步骤处理合表时可能由 auto_pk_c1 导致的 ERROR 1062 (23000): Duplicate entry '***' for key 'PRIMARY' 问题：

1. 在开始执行全量数据迁移前，在下游数据库创建用于合表迁移的表，但不为 auto_pk_c1 指定 PRIMARY KEY 属性，而是将 auto_pk_c1 与 uuid_c2 一起组成 PRIMARY KEY。

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

2. 启动数据迁移任务，执行全量与增量数据迁移。

3. 通过 query-status 验证数据迁移任务是否正常，在下游数据库中验证合表中是否已经存在了来自上游的数据。

4.3.4 合表迁移过程中在上游增/删表

从分库分表合并迁移的实现原理部分我们可以知道，sharding DDL lock 的协调依赖于是否已收到上游已有各分表对应的 DDL，且当前 DM 在合表迁移过程中暂时不支持在上游动态增加或删除分表。如果需要在合表迁移过程中，对上游执行增、删分表操作，推荐按以下方式进行处理。

4.3.4.1 在上游增加分表

如果需要在上游增加新的分表，推荐按以下顺序执行操作：

1. 等待在上游分表上执行过的所有 sharding DDL 协调同步完成。

2. 通过 `stop-task` 停止任务。
3. 在上游添加新的分表。
4. 确保 `task.yaml` 配置能使新添加的分表与其他已有的分表能合并到同一个下游表。
5. 通过 `start-task` 启动任务。
6. 通过 `query-status` 验证数据迁移任务是否正常，在下游数据库中验证合表中是否已经存在了来自上游的数据。

4.3.4.2 在上游删除分表

如果需要在上游删除原有的分表，推荐按以下顺序执行操作：

1. 在上游删除原有的分表，并通过 `SHOW BINLOG EVENTS` 获取该 `DROP TABLE` 语句在 binlog 中对应的 `End_log_pos`，记为 *Pos-M*。
2. 通过 `query-status` 获取当前 DM 已经处理完成的 binlog event 对应的 position（`syncerBinlog`），记为 *Pos-S*。
3. 当 *Pos-S* 比 *Pos-M* 更大后，则说明 DM 已经处理完 `DROP TABLE` 语句，且该表在被删除前的数据都已经迁移到下游，可以进行后续操作；否则，继续等待 DM 进行数据迁移。
4. 通过 `stop-task` 停止任务。
5. 确保 `task.yaml` 配置能忽略上游已删除的分表。
6. 通过 `start-task` 启动任务。
7. 通过 `query-status` 验证数据迁移任务是否正常。

4.3.5 数据迁移限速/流控

当将多个上游 MySQL/MariaDB 实例的数据合并迁移到下游同一个 TiDB 集群时，由于每个与上游 MySQL/MariaDB 对应的 DM-worker 都会并发地进行全量数据迁移与增量数据复制，默认的并发度（全量阶段的 `pool-size` 与增量阶段的 `worker-count`）通过多个 DM-worker 累加后，可能会给下游造成过大的压力，此时应根据 TiDB 监控及 DM 监控进行初步的性能分析后，适当地调整各并发度参数的大小。后续 DM 也将考虑支持部分自动的流控策略。

4.4 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 DM 通过 DM-Ansible 部署时必须指定 `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。

4.4.1 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 relay 处理单元已从原 MySQL 实例获取到的 binlog 对应的 GTID sets (`relayBinlogGtid`)，记为 `gtid-W`。
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-W` 包含 `gtid-P`（`gtid-P` 可以为空）
    - `gtid-E` 包含 `gtid-W`
5. 使用 `pause-relay` 命令暂停 relay 处理。
6. 使用 `pause-task` 命令暂停所有运行中的数据迁移任务。
7. 变更 VIP 以指向新的 MySQL 实例。
8. 使用 `switch-relay-master` 命令通知 relay 处理单元进行主从切换。
9. 使用 `resume-relay` 命令恢复 relay 处理，使其从新的 MySQL 实例读取 binlog。
10. 使用 `resume-task` 命令恢复之前的数据迁移任务。

4.4.2 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 relay 处理单元已从原 MySQL 实例获取到的 binlog 对应的 GTID sets (`relayBinlogGtid`)，记为 `gtid-W`。
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-W` 包含 `gtid-P`（`gtid-P` 可以为空）
    - `gtid-E` 包含 `gtid-W`
5. 使用 `stop-task` 命令停止所有运行中的数据迁移任务。
6. 更新 `inventory.ini` 中的 DM-worker 配置，并使用 DM-Ansible 对 DM-worker 进行滚动升级操作。
7. 使用 `start-task` 命令重新启动数据迁移任务。

5 TiDB Data Migration 教程

TiDB Data Migration (DM) 是一体化的数据迁移任务管理平台，支持将大量、复杂的生产环境中的数据从 MySQL 或 MariaDB 迁移到 TiDB。

DM 功能如下：

- 数据迁移
    - 支持导出与导入源数据库的初始全量数据，并在数据迁移过程中读取、应用来自源数据库存储的 binlog，从而保持数据的迁移。
    - 通过合并上游的多个 MySQL 或 MariaDB 实例或集群的表，DM 能迁移生产环境中的分库分表。
- 将 TiDB 作为 MySQL 或 MariaDB 的从库时，DM 能持续提高数据库水平扩展的能力，或在无需 ETL 作业的情况下，在 TiDB 上进行数据实时分析。

本教程主要介绍如何使用 DM 迁移上游多个 MySQL 实例的一个分片表。包括两种场景：

- 合并若干个互不冲突的表或分片，即这些表或分片的表结构并不会造成唯一键的冲突；
- 合并唯一键存在冲突的表。

本教程假设目前使用的是一台新的、纯净版 CentOS 7 实例，你能（使用 VMware、VirtualBox 及其他工具）在本地虚拟化或在供应商提供的平台上部署一台小型的云虚拟主机。因为需要运行多个服务，建议内存最好在 1 GB 以上。

> **警告：**
>
> 本教程中 TiDB 的部署方法并不适用于生产或开发环境。

5.1 Data Migration 架构

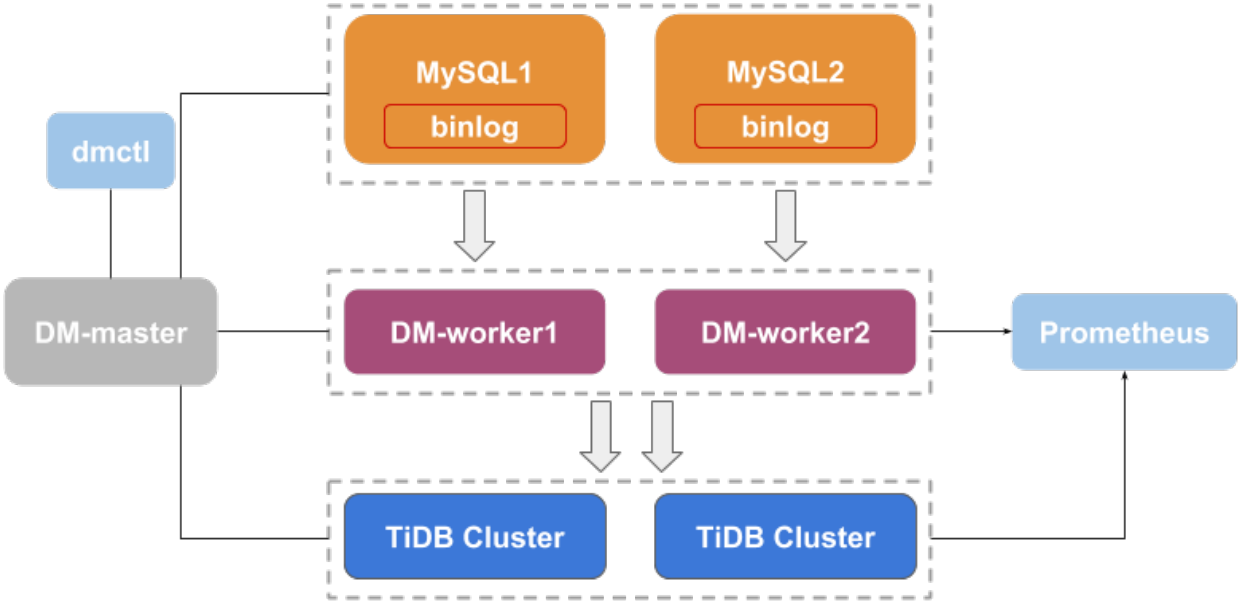


Figure 5: TiDB Data Migration 架构

TiDB Data Migration 平台由 3 部分组成：DM-master、DM-worker 和 dmctl。

- DM-master 负责管理和调度数据迁移任务的操作。
- DM-worker 负责执行特定的数据迁移任务。
- dmctl 是一个命令行工具，用于控制 DM 集群。

`.yaml` 文件中定义了各个数据迁移任务，dmctl 会读取这些文件，并且这些文件会被提交给 DM-master。DM-master 再将关于给定任务的相应职责告知每个 DM-worker 实例。

详情参见Data Migration 简介。

5.2 安装

本部分介绍如何部署 3 个 MySQL Server 实例及 pd-server、tikv-server 和 tidb-server 实例各 1 个，以及如何启动 1 个 DM-master 和 3 个 DM-worker 实例。

1. 安装 MySQL 5.7，下载或提取 TiDB v3.0 以及 DM v1.0.2 安装包：

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

2. 创建目录和符号链接：

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

3. 安装 3 个 MySQL Server 实例的配置：

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

```
server-id=3
port=3309
auto-increment-offset=3
EoCNF
```

4. 初始化并启动这些 MySQL 实例：

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

5. 执行 jobs 和/或 pgrep -a mysqld 以确保 MySQL Server 实例都在运行状态。

```
jobs
```

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

```
pgrep -a mysqld
```

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

5.3 迁移分片数据

本示例场景包含 3 个分片，这些分片表结构相同，但自增主键并不重叠。

在 .my.cnf 文件中设置 auto-increment-increment=5 和 auto-increment-offset 可以实现这种情况。将 auto-increment-increment 设置为 5，则这些实例的自增 ID 以 5 为单位递增；每个实例的 auto-increment-offset 都设置得不同，则这些实例的偏移为 0 到开始计数的值。例如，若一个实例的 auto-increment-increment 为 5，auto-increment-offset 为 2，则会生成自增 ID 序列 {2,7,12,17,22,...}。

1. 对于这 3 个 MySQL Server 实例，每个实例都分别创建数据库和表：

```
for i in 1 2 3
do
    mysql -h 127.0.0.1 -P "$((3306+i))" -u root <<EoSQL
        create database dmtest1;
```

```
        create table dmtest1.t1 (id bigint unsigned not null
            ↪ AUTO_INCREMENT primary key, c char(32), port int);
EoSQL
done
```

2. 在每个 MySQL 实例中插入几百行数据：

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

3. 查询上一步写入的所有行并排序，以确认写入数据是正确的：

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

注意返回的列表左侧是一列递增的、无重叠的 ID，右侧的端口编号显示这些数据插入到哪些实例以及从哪些实例中查询：

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

5.3.1 启动 DM-master 和 DM-worker

本小节介绍如何使用 DM 将来自不同的 MySQL 实例的数据合并到 TiDB 的一张表里。

配置文件包 `dm-cnf.tgz` 包含：

- TiDB 集群组件和 DM 组件的配置
- 本教程后文介绍的 2 个 DM 任务的配置

1. 启动单个 `tidb-server` 实例、每个 MySQL Server 实例（总共 3 个实例）的 DM-worker 进程和一个 DM-master 进程：

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

2. 执行 `jobs` 和/或 `ps -a`，确保这些进程都正在运行：

```
jobs
```

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

```
ps -a
```

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

每个上游的 MySQL Server 实例对应一个单独的 DM-worker 实例，每个 DM-worker 实例都有各自的配置文件。这些文件内容包括：

- 连接到上游 MySQL Server 的详细信息
- relay log（上游服务器的 binlog）的存储路径
- mydumper 的输出

各个 DM-worker 通过不同的端口监听（由 `worker-addr` 定义）。

以下为 `dm-worker1.toml` 的示例：

```
# DM-worker 配置

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

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

- 如果从 MySQL Server、Percona Server、Percona XtraDB Cluster、Amazon Aurora 或 RDS 迁移数据，则 `flavor` 配置项应设为 "mysql"（默认值，支持 5.5 < MySQL 版本 < 8.0）。
- 如果从 MariaDB Server 或 MariaDB (Galera) Cluster 迁移数据，则设置 `flavor = "mariadb"`（仅支持 10.1.2 以上 MariaDB 版本）。
- 从 DM 1.0.2 版本开始，`flavor`、`server-id` 项均会由 DM 自动生成，一般情况下不需要手动配置。
- `from` 中的 `password` 如果不为空，则需要使用 dmctl 进行加密，参见使用 dmctl 加密上游 MySQL 用户密码。

任务在 YAML 文件中定义。以下为一个 `dmtask1.yaml` 文件示例：

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

target-database:
  host: "127.0.0.1"
  port: 4000
```

```
  user: "root"
  password: ""

mysql-instances:
  - source-id: "mysql1"
    server-id: 1
    block-allow-list: "dmtest1" # 如果 DM 版本 <= v1.0.6 则使用 black-white-
      ↪ list
    loader-config-name: "loader1"
  - source-id: "mysql2"
    server-id: 2
    block-allow-list: "dmtest1" # 如果 DM 版本 <= v1.0.6 则使用 black-white-
      ↪ list
    loader-config-name: "loader2"
  - source-id: "mysql3"
    server-id: 3
    block-allow-list: "dmtest1" # 如果 DM 版本 <= v1.0.6 则使用 black-white-
      ↪ list
    loader-config-name: "loader3"

block-allow-list:              # 如果 DM 版本 <= v1.0.6 则使用 black-white-
  ↪ list
  dmtest1:
    do-dbs: ["dmtest1"]

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

以上文件包含一些全局配置项和几组定义各种行为的配置项。

- `task-mode: all`：DM 导入上游实例的全量备份，并使用上游 MySQL Server 的 binlog 进行增量复制。
  - 此外，可将 `task-mode` 设置为 `full` 或 `incremental` 以分别进行全量迁移或增量复制。
- `is-sharding: true`：多个 DM-worker 实例进行同一个任务，这些实例将上游的若干分片合并到一个下游的表中。
- `ignore-checking-items: ["auto_increment_ID"]`：关闭 DM 对上游实例中潜在的自增 ID 冲突的检测。DM 能检测出上游表结构相同、并包含自增列的分片间潜在的

列值冲突。通过配置 auto-increment-increment 和 auto-increment-offset 可使每个 MySQL Server 的 ID 都不重叠，从而避免不同表之间冲突的产生。因此，可以让 DM 关闭对自增 ID 冲突的检测。

- block-allow-list：将一个任务限制在数据库 dmtest 中。
- loaders：定义由各个 DM-worker 实例执行的每个 mydumper 实例的输出地址。
- target-database：定义目标数据库的链接信息，其中的 password 如果不为空，则需要使用 dmctl 进行加密，参见使用 dmctl 加密上游 MySQL 用户密码。

dmctl 是控制 DM 集群的命令行工具，用于启动任务、查询任务状态。执行 dmctl --master-addr :8261 获取如下交互提示，从而启动该工具：

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

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

»
```

执行 start-task dm-cnf/dmtask1.yaml 启动 dmtask1：

```
start-task dm-cnf/dmtask1.yaml
```

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "127.0.0.1:8262",
            "msg": ""
        },
        {
            "result": true,
            "worker": "127.0.0.1:8263",
            "msg": ""
        },
        {
            "result": true,
            "worker": "127.0.0.1:8264",
```

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

启动该任务意味着启动任务配置文件中定义的行为，包括执行 mydumper 和 loader 实例，加载初次 dump 的数据后，将 DM-worker 作为迁移任务的 slave 连接到上游的 MySQL Server。

所有的行数据都被迁移到 TiDB Server：

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

输出结果：

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

现在 DM 正作为每个 MySQL Server 的 slave，读取 MySQL Server 的 binlog，将更新的数据实时迁移到下游的 TiDB Server：

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

输出结果：

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

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

向上游 MySQL Server 插入几行数据，更新 MySQL 中的这些行，并再次查询这些行：

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

输出结果：

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

更新这些行，则可见更新的数据已迁移到 TiDB 中：

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

输出结果：

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

只要 DM-master 和 DM-worker 运行 `dmtest1` 任务，下游的 TiDB Server 将持续和上游的 MySQL Server 实例保持迁移的状态。

5.4 结论

本教程完成了上游 3 个 MySQL Server 实例的分片迁移，介绍了分片迁移中，DM 如何在集群中导入初始数据，以及如何读取 MySQL 的 binlog 来复制增量数据，从而使下游 TiDB 集群与上游实例保持迁移。

关于 DM 的更多详情，请参考Data Migration 简介，或加入 TiDB Community Slack channel 参与讨论。

6 部署

6.1 部署 DM 集群

6.1.1 使用 DM-Ansible 部署 DM 集群

DM-Ansible 是 PingCAP 基于 Ansible 的 Playbooks 研发的 DM (Data Migration) 集群部署工具。本文将介绍如何使用 DM-Ansible 快速部署 DM 集群。

6.1.1.1 准备工作

在开始之前，先确保您准备好了以下配置的机器：

1. 部署目标机器若干，配置如下：

    - CentOS 7.3 (64-bit) 或更高版本，x86_64 架构（AMD64）
    - 机器之间内网互通
    - 关闭防火墙，或开放服务端口

2. 一台中控机，配置如下：

    - 包含 Python 2.7 的 CentOS 7.3（64-bit）或更高版本
    - Ansible 2.5 或更高版本
    - 互联网访问

6.1.1.2 第 1 步：在中控机上安装依赖包

> **注意：**
>
> 请确保使用 `root` 账户登录中控机。

根据中控机的操作系统版本，运行相应命令如下：

- CentOS 7：

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

- Ubuntu：

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

6.1.1.3 第 2 步：在中控机上创建 `tidb` 用户，并生成 SSH 密钥

> **注意：**
>
> 请确保使用 `root` 账户登录中控机。

1. 创建 `tidb` 用户。

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

2. 为 tidb 用户设置密码。

```
passwd tidb
```

3. 在 sudo 文件尾部加上 tidb ALL=(ALL)NOPASSWD: ALL，为 tidb 用户设置免密使用 sudo。

```
visudo
```

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

4. 生成 SSH 密钥。

   执行以下 su 命令，将登录用户从 root 切换至 tidb。

```
su - tidb
```

   为 tidb 用户创建 SSH 密钥。当提示 Enter passphrase 时，按 Enter 键。命令成功执行后，生成的 SSH 私钥文件为 /home/tidb/.ssh/id_rsa，SSH 公钥文件为/home/tidb/.ssh/id_rsa.pub。

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

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

6.1.1.4 第 3 步：下载 DM-Ansible 至中控机

> **注意：**
> 请确保使用 `tidb` 账户登录中控机。

1. 打开 `/home/tidb` 目录。
2. 执行以下命令下载 DM-Ansible。

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

`{version}` 为期望下载的 DM 版本，如 `v1.0.1`、`v1.0.2` 等。可以通过 DM Release 查看当前已发布版本。

6.1.1.5 第 4 步：安装 DM-Ansible 及其依赖至中控机

> **注意：**
>
> - 请确保使用 `tidb` 账户登录中控机。
> - 您需要使用 `pip` 方式下载安装 TiDB Ansible 及其依赖，否则可能会遇到兼容性问题。DM-Ansible 当前与 Ansible 2.5 或更高版本兼容。

1. 在中控机上安装 DM-Ansible 及其依赖包：

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

Ansible 和相关依赖包含于 `dm-ansible/requirements.txt` 文件中。

2. 查看 Ansible 版本：

```
ansible --version
```

```
ansible 2.5.0
```

6.1.1.6 第 5 步：在中控机上配置 SSH 互信和 sudo 规则

> **注意：**
> 请确保使用 tidb 账户登录至中控机。

1. 将您部署的目标机器的 IP 地址加至 hosts.ini 文件中的 [servers] 部分。

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

```
[servers]
172.16.10.71
172.16.10.72
172.16.10.73

[all:vars]
username = tidb
```

2. 运行如下命令，然后输入部署目标机器的 root 用户密码。

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

该步骤将在部署目标机器上创建 tidb 用户，创建 sudo 规则，并为中控机和部署目标机器之间配置 SSH 互信。

6.1.1.7 第 6 步：下载 DM 及监控组件安装包至中控机

> **注意：**
> 请确保中控机接入互联网。

在中控机上，运行如下命令：

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

6.1.1.8 第 7 步：编辑 inventory.ini 配置文件

注意：

请确保使用 tidb 账户登录中控机。

打开并编辑 /home/tidb/dm-ansible/inventory.ini 文件如下，以管控 DM 集群。

```
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
```

根据场景需要，您可以在以下两种集群拓扑中任选一种：

- 单节点上单个 DM-worker 实例的集群拓扑
- 单节点上多个 DM-worker 实例的集群拓扑

通常情况下，我们推荐每个节点上部署单个 DM-Worker 实例。但如果您的机器拥有性能远超 TiDB 软件和硬件环境要求中推荐配置的 CPU 和内存，并且每个节点配置 2 块以上的硬盘或大于 2T 的 SSD，您可以在单个节点上部署不超过 2 个 DM-Worker 实例。

6.1.1.8.1 选项 1：使用单节点上单个 DM-Worker 实例的集群拓扑

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

```
### DM 模块
[dm_master_servers]
dm_master ansible_host=172.16.10.71

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

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

```
[dm_portal_servers]
dm_portal ansible_host=172.16.10.71

### 监控模块
[prometheus_servers]
prometheus ansible_host=172.16.10.71

[grafana_servers]
grafana ansible_host=172.16.10.71

[alertmanager_servers]
alertmanager ansible_host=172.16.10.71

### 全局变量
[all:vars]
cluster_name = test-cluster

ansible_user = tidb

dm_version = {version}

deploy_dir = /data1/dm

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

{version} 为当前 DM-Ansible 对应的版本号。关于 DM-worker 参数的更多信息，请参考DM-worker 配置及参数描述。

6.1.1.8.2 选项 2：使用单节点上多个 DM-worker 实例的集群拓扑

节点	主机 IP	服务
node1	172.16.10.71	DM-master, Prometheus, Grafana, Alertmanager, DM Portal
node2	172.16.10.72	DM-worker1-1, DM-worker1-2
node3	172.16.10.73	DM-worker2-1, DM-worker2-2

编辑 inventory.ini 文件时，请注意区分这些变量：server_id，deploy_dir，和 dm_worker_port。

```
### DM 模块
[dm_master_servers]
dm_master ansible_host=172.16.10.71
```

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

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

[dm_portal_servers]
dm_portal ansible_host=172.16.10.71

### 监控模块
[prometheus_servers]
prometheus ansible_host=172.16.10.71

[grafana_servers]
grafana ansible_host=172.16.10.71

[alertmanager_servers]
alertmanager ansible_host=172.16.10.71

### 全局变量
[all:vars]
cluster_name = test-cluster

ansible_user = tidb

dm_version = {version}

deploy_dir = /data1/dm

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

{version} 为当前 DM-Ansible 对应的版本号。

6.1.1.8.3 DM-worker 配置及参数描述

变量名称	描述
source_id	DM-worker 绑定到的一个数据库实例或是具有主从架构的复制组。当发生主从切换的时候，只需要更新 `mysql_host` ↪ 或 `mysql_port` ↪ 而不用更改该 ID 标识。

变量名称	描述
server_id	DM-worker 伪装成一个 MySQL slave，该变量即为这个 slave 的 server ID，在 MySQL 集群中需保持全局唯一。取值范围 0 ~ 4294967295。v1.0.2 及以上版本的 DM 会自动生成，不需要手动配置。
mysql_host	上游 MySQL 主机。
mysql_user	上游 MySQL 用户名，默认值为 “root”。

变量名称	描述
mysql_password	上游 MySQL 用户密码，需使用 dmctl 工具加密。请参考使用 dmctl 加密上游 MySQL 用户密码。
mysql_port	上游 MySQL 端口，默认 3306。
enable_gtid	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。

变量名称	描述
relay_binlog_name	DM-worker 是否从指定 binlog 文件位置开始拉取 binlog。仅适用于本地无有效 relay log 的情况。v1.0.2 及以上版本的 DM 会默认从最新位置开始拉取 binlog，一般情况下不需要手动配置。

变量名称	描述
relay_binlog_gtid	DM-worker 是否从指定 GTID 位置开始拉取 binlog。仅适用于本地无有效 relay log，且 enable_gtid ↪ 设置为 true 的情况。v1.0.2 及以上版本的 DM 会默认从最新位置开始拉取 binlog，一般情况下不需要手动配置。

变量名称	描述
flavor	代表 MySQL 的版本发布类型。如果是官方版本，Percona 版，或 Cloud MySQL 版，其值为 “mysql”。如果是 MariaDB，其值为 “mariadb”。默认值是 “mysql”。v1.0.2 及以上版本的 DM 会自动判断上游版本，不需要手动配置。

关于 deploy_dir 配置的更多信息，请参考配置部署目录。

6.1.1.8.4 使用 dmctl 加密上游 MySQL 用户密码

假定上游 MySQL 的用户密码为 123456，运行以下命令，并将生成的字符串添加至 DM-worker 的 mysql_password 变量。

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

```
MKxn0Qo3m3XOyjCnhEMtsUCm83EhGQDZ/T4=
```

注意：

- 如果数据库没有设置密码，则可以跳过该步骤。
- DM v1.0.6 及其以后版本可以配置明文数据库密码。

6.1.1.9 第 8 步：编辑 inventory.ini 文件中的变量

此步介绍如何编辑部署目录中的变量，如何配置 relay log 迁移位置以及 relay log GTID 的迁移模式。此外，还会描述 inventory.ini 中的全局变量。

6.1.1.9.1 配置部署目录

编辑 deploy_dir 变量以配置部署目录。

- 全局变量默认设为 /home/tidb/deploy，适用于所有服务。如果数据盘挂载于 /data1 目录，您可以通过以下修改将其变更至 /data1/dm。

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

- 如果需要为某个服务创建单独的部署目录，您可以在 inventory.ini 中配置服务主机列表的同时设置 host 变量。此操作需要您添加第一列别名，以避免在混合服务部署场景下产生混淆。

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

6.1.1.9.2 配置 relay log 迁移位置

首次启动 DM-worker 时，您需要配置 relay_binlog_name 变量以指定 DM-worker 拉取上游 MySQL 或 MariaDB binlog 的起始位置。

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

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

> **注意：**
>
> 如未设定 `relay_binlog_name`，v1.0.2 之前版本的 DM-worker 将从上游 MySQL 或 MariaDB 现有最早时间点的 binlog 文件开始拉取 binlog，拉取到数据迁移任务需要的最新 binlog 可能需要很长时间；v1.0.2 及之后版本的 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。

6.1.1.9.3 开启 relay log GTID 迁移模式

在 DM 集群中，DM-worker 的 relay log 处理单元负责与上游 MySQL 或 MariaDB 通信，从而将 binlog 拉取至本地文件系统。

DM 目前支持 MySQL GTID 和 MariaDB GTID。您可以通过配置以下项目开启 relay log GTID 迁移模式：

- `enable_gtid`：打开 relay log GTID 迁移模式以处理 master 和 slave 易位的场景
- `relay_binlog_gtid`：指定 DM-worker 开始拉取对应上游 MySQL 或 MariaDB binlog 的起始位置

示例配置如下：

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

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

6.1.1.9.4 全局变量

变量名称	描述
cluster_name	集群名称，可调整
dm_version	DM 版本，默认已配置
grafana_admin_user	Grafana 管理员用户名称，默认值 admin
grafana_admin_password	Grafana 管理员账户的密码，用于通过 Ansible 导入 Dashboard。默认值为 admin。如果您在 Grafana 网页端修改了密码，请更新此变量。

6.1.1.10 第 9 步：部署 DM 集群

使用 ansible-playbook 运行 Playbook，默认并发数量是 5。如果部署目标机器较多，您可以使用 -f 参数增加并发数量，例如，ansible-playbook deploy.yml -f 10。

以下部署操作示例使用中运行服务的用户为 tidb：

1. 编辑 dm-ansible/inventory.ini 文件，确保 ansible_user = tidb。

```
ansible_user = tidb
```

> 注意：
> 请勿将 ansible_user 设为 root，因为 tidb-ansible 限制服务需以普通用户运行。

运行以下命令。如果所有服务都返回 tidb，则 SSH 互信配置成功。

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

运行以下命令。如果所有服务都返回 root，则 tidb 用户免密 sudo 操作配置成功。

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

2. 修改内核参数，并部署 DM 集群组件和监控组件。

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

> 注意：
> 目前 DM 和 TiDB 在部署与滚动升级时，均会覆盖监控组件原有的运行配置。因此，强烈建议为 DM 和 TiDB 部署独立的监控组件。

3. 启动 DM 集群。

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

此操作会按顺序启动 DM 集群的所有组件，包括 DM-master，DM-worker，以及监控组件。当一个 DM 集群被关闭后，您可以使用该命令将其开启。

6.1.1.11 第 10 步：关闭 DM 集群

如果您需要关闭一个 DM 集群，运行以下命令：

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

该操作会按顺序关闭整个 DM 集群中的所有组件，包括 DM-master，DM-worker，以及监控组件。

6.1.1.12 常见部署问题

6.1.1.12.1 默认服务端口

组件	端口变量	默认端口	描述
DM-master	dm_master_port	8261	DM-master 服务交流端口
DM-worker	dm_worker_port	8262	DM-worker 服务交流端口
Prometheus	prometheus_port	9090	Prometheus 服务交流端口
Grafana	grafana_port	3000	外部 Web 监控服务及客户端（浏览器）访问端口
Alertmanager	alertmanager_port	9093	Alertmanager 服务交流端口

6.1.1.12.2 自定义端口

编辑 inventory.ini 文件，将服务端口的相关主机变量添加在对应服务 IP 地址后：

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

6.1.1.12.3 更新 DM-Ansible

1. 使用 tidb 账户登录至中控机，进入 /home/tidb 目录，然后备份dm-ansible 文件夹。

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

2. 下载指定版本 DM-Ansible，解压。

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

```
mv dm-ansible-{version} dm-ansible
```

3. 迁移 inventory.ini 配置文件。

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

4. 迁移 dmctl 配置。

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

5. 用 Playbook 下载最新的 DM 二进制文件。此文件会自动替换 /home/tidb/dm-ansible/resource/bin/ 目录下的二进制文件。

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

6.1.2 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。

6.1.2.1 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、dmctl 以及 Mydumper 的二进制文件。conf 目录下有相关的示例配置文件。

6.1.2.2 使用样例

假设在两台服务器上部署 MySQL，在一台服务器上部署 TiDB (mocktikv 模式)，另外在三台服务器上部署两个 DM-worker 实例和一个 DM-master 实例。各个节点的信息如下：

实例	服务器地址
MySQL1	192.168.0.1

实例	服务器地址
MySQL2	192.168.0.2
TiDB	192.168.0.3
DM-master	192.168.0.4
DM-worker1	192.168.0.5
DM-worker2	192.168.0.6

MySQL1 和 MySQL2 中需要开启 binlog。DM-worker1 负责迁移 MySQL1 的数据，DM-worker2 负责迁移 MySQL2 的数据。下面以此为例，说明如何部署 DM。

6.1.2.2.1 DM-worker 的部署

DM-worker 需要连接上游 MySQL，且为了安全，强制用户配置加密后的密码。首先使用 dmctl 对 MySQL 的密码进行加密，以密码为 “123456” 为例：

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

```
fCxfQ9XKCezSzuCD0Wf5dUD+LsKegSg=
```

记录该加密后的值，用于下面部署 DM-worker 时的配置。

DM-worker 提供命令行参数和配置文件两种配置方式。

配置方式 1：命令行参数

查看 DM-worker 的命令行参数说明：

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

```
Usage of worker:
  -L string
        日志等级，值可以为 "debug", "info", "warn", "error" 或者 "fatal" (
            ↪ 默认值："info")
  -V    输出版本信息
  -checker-backoff-max duration
        任务检查模块中，检查出错误后等待自动恢复的最长时间间隔 (默认值："5m0s
            ↪ "，一般情况下不需要修改。如果对该参数的作用没有深入的了解，
            ↪ 不建议修改该参数)
  -checker-backoff-rollback duration
        任务检查模块中，调整自动恢复等待时间的间隔 (默认值："5m0s"，
            ↪ 一般情况下不需要修改，如果对该参数的作用没有深入的了解，
            ↪ 不建议修改该参数)
  -checker-check-enable
        是否开启任务状态检查。开启后 DM
            ↪ 会尝试自动恢复因错误而暂停的数据迁移任务 (默认值：true)
  -config string
        配置文件的路径
```

```
  -log-file string
        日志文件的路径
  -print-sample-config
        打印示例配置
  -purge-expires int
        relay log 的过期时间。DM-worker
            ↪ 会尝试自动删除最后修改时间超过了过期时间的 relay log（单位：小时
            ↪ ）
  -purge-interval int
        定期检查 relay log 是否过期的间隔时间（默认值：3600）（单位：秒）
  -purge-remain-space int
        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）
  -relay-dir string
        存储 relay log 的路径（默认值："./relay_log"）
  -worker-addr string
        DM-worker 的地址
```

> **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。

配置方式 2：配置文件

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.

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

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

### 作为 MySQL slave 的 server ID，用于获取 MySQL 的 binlog
### 在一个 replication group 中，每个实例（master 和 slave）都应该有唯一的
    ↪ server ID
### v1.0.2 及以上版本的 DM 会自动生成，不需要手动配置
```

```
server-id = 101

### 用于标识一个 replication group 或者 MySQL/MariaDB 实例
source-id = "mysql-replica-01"

### 上游实例类型，值可为 "mysql" 或者 "mariadb"
### v1.0.2 及以上版本的 DM 会自动识别上游实例类型，不需要手动配置
flavor = "mysql"

### MySQL 的连接地址
[from]
host = "192.168.0.1"
user = "root"
password = "fCxfQ9XKCezSzuCD0Wf5dUD+LsKegSg="
port = 3306
```

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 source-id 为 mysql-replica-02，并将 from 配置部分修改为 MySQL2 的地址即可。如果因为没有多余的机器，将 DM-worker2 与 DM-worker1 部署在一台机器上，需要把两个 DM-worker 实例部署在不同的路径下，否则保存元信息和 relay log 的默认路径会冲突。

6.1.2.2.2 DM-master 的部署

DM-master 提供命令行参数和配置文件两种配置方式。

配置方式 1：命令行参数

DM-master 的命令行参数说明：

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

```
Usage of dm-master:
  -L string
        日志等级，值可以为 "debug", "info", "warn", "error" 或者 "fatal" (
            ↪ 默认值为 "info")
  -V    输出版本信息
  -config string
        配置文件的路径
  -log-file string
        日志文件的路径
  -master-addr string
        DM-master 的地址
  -print-sample-config
```

打印出 DM-master 的示例配置

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。

配置方式 2：配置文件

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

DM-master 的配置文件：

```
### Master Configuration.

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

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

### DM-Worker 的配置
[[deploy]]
### 对应 DM-worker1 配置文件中的 source-id
source-id = "mysql-replica-01"
### DM-worker1 的服务地址
dm-worker = "192.168.0.5:8262"

[[deploy]]
### 对应 DM-worker2 配置文件中的 source-id
source-id = "mysql-replica-02"
### DM-worker2 的服务地址
dm-worker = "192.168.0.6:8262"
```

在终端中使用下面的命令运行 DM-master：

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

这样，DM 集群就部署成功了。下面创建简单的数据迁移任务来使用 DM 集群。

6.1.2.2.3 创建数据迁移任务

假设在 MySQL1 和 MySQL2 实例中有若干个分表，这些分表的结构相同，所在库的名称都以 “sharding” 开头，表名称都以 “t” 开头，并且主键或唯一键不存在冲突（即每张分表的主键或唯一键各不相同）。现在需要把这些分表迁移到 TiDB 中的 db_target.t_target 表中。

首先创建任务的配置文件：

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

target-database:
  host: "192.168.0.3"
  port: 4000
  user: "root"
  password: "" # 如果密码不为空，也需要配置 dmctl 加密后的密码

mysql-instances:
  - source-id: "mysql-replica-01"
    block-allow-list: "instance" # 如果 DM 版本 <= v1.0.6 则使用 black-white
      ↪ -list
    route-rules: ["sharding-route-rules-table", "sharding-route-rules-schema
      ↪ "]
    mydumper-thread: 4          # dump 处理单元用于导出数据的线程数量，在 v1
      ↪ .0.2 版本引入
    loader-thread: 16           # load 处理单元用于导入数据的线程数量，在 v1
      ↪ .0.2 版本引入
    syncer-thread: 16           # sync 处理单元用于复制增量数据的线程数量，在
      ↪ v1.0.2 版本引入

  - source-id: "mysql-replica-02"
    block-allow-list: "instance" # 如果 DM 版本 <= v1.0.6 则使用 black-white
      ↪ -list
    route-rules: ["sharding-route-rules-table", "sharding-route-rules-schema
      ↪ "]
    mydumper-thread: 4          # dump 处理单元用于导出数据的线程数量，在 v1
      ↪ .0.2 版本引入
    loader-thread: 16           # load 处理单元用于导入数据的线程数量，在 v1
      ↪ .0.2 版本引入
    syncer-thread: 16           # sync 处理单元用于复制增量数据的线程数量，在
      ↪ v1.0.2 版本引入

block-allow-list:               # 如果 DM 版本 <= v1.0.6 则使用 black-white-
  ↪ list
  instance:
```

```
    do-dbs: ["~^sharding[\\d]+"]
    do-tables:
    -  db-name: "~^sharding[\\d]+"
       tbl-name: "~^t[\\d]+"

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

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

将以上配置内容写入到 conf/task.yaml 文件中，使用 dmctl 创建任务：

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

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

```
» start-task conf/task.yaml
```

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

```
}
```

这样就成功创建了一个将 MySQL1 和 MySQL2 实例中的分表数据迁移到 TiDB 的任务。

6.1.3 使用 Kubernetes (实验特性)

6.2 使用 DM 迁移数据

本文介绍如何使用 DM (Data Migration) 迁移数据。

6.2.1 第 1 步：部署 DM 集群

目前推荐使用 DM-Ansible 部署 DM 集群，具体部署方法参照使用 DM-Ansible 部署 DM 集群；也可以使用 binary 部署 DM 集群用于体验或者测试，具体部署方法参照使用 DM binary 部署 DM 集群。

> 注意：
>
> - 在 DM 所有的配置文件中，数据库的密码要使用 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 加密上游 MySQL 用户密码。
> - 上下游数据库用户必须拥有相应的读写权限。

6.2.2 第 2 步：检查集群信息

使用 DM-Ansible 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=

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

- dm-master 进程配置文件 {ansible deploy}/conf/dm-master.toml 中的配置

```
# Master 配置

# 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

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

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

> **注意：**
>
> {ansible deploy}/conf/dm-master.toml 中的 {ansible deploy} 表示使用 DM-Ansible 部署 DM 时通过 deploy_dir 参数指定的目录。

6.2.3 第 3 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。

复制并编辑 {ansible deploy}/conf/task.yaml.example，生成如下任务配置文件 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: "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 <= v1.0.6 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 <= v1.0.6 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 <= v1.0.6 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:
    mydumper-path: "./bin/mydumper" # dump 处理单元二进制文件的路径。
    extra-args: "-B test_db -T test_table" # dump 处理单元的其他参数，从 DM
      ↪ 1.0.2 版本开始，DM 会自动生成 table-list 配置，
      ↪ 在其之前的版本仍然需要人工配置。
```

6.2.4 第 4 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。

1. 进入 dmctl 目录 `/home/tidb/dm-ansible/resources/bin/`。
2. 执行以下命令启动 dmctl。

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

3. 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。

```
» 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。

6.2.5 第 5 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，可在 dmctl 内使用以下命令进行查询：

```
» query-status
```

6.2.6 第 6 步：停止任务

如果不再需要进行数据迁移，可以在 dmctl 内使用以下命令停止迁移任务：

```
» 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。

6.2.7 第 7 步：监控任务与查看日志

如果使用 DM-Ansible 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 DM-Ansible 部署 DM，则日志目录位于 DM-master 节点的 `{ansible deploy}/log/dm-master.log`。
- DM-worker 日志目录：通过 DM-worker 进程参数 `--log-file` 设置。如果使用 DM-Ansible 部署 DM，则日志目录位于 DM-worker 节点的 `{ansible deploy}/log/dm-worker.log`。
- dmctl 日志目录：与其二进制文件目录相同。

7 配置

7.1 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。

7.1.1 配置文件

- `inventory.ini`：使用 DM-Ansible 部署 DM 集群的配置文件。需要根据所选用的集群拓扑来进行编辑。详见编辑 `inventory.ini` 配置文件。
- `dm-master.toml`：DM-master 进程的配置文件，包括 DM 集群的拓扑信息、MySQL 实例与 DM-worker 之间的关系（必须为一对一的关系）。使用 DM-Ansible 部署 DM 集群时，会自动生成 `dm-master.toml` 文件，各项配置说明详见DM-master 配置文件介绍
- `dm-worker.toml`：DM-worker 进程的配置文件，包括上游 MySQL 实例的配置和 relay log 的配置。使用 DM-Ansible 部署 DM 集群时，会自动生成 `dm-worker.toml` 文件，各项配置说明详见DM-worker 配置文件介绍

7.1.2 迁移任务配置

7.1.2.1 任务配置文件

使用 DM-Ansible 部署 DM 集群时，<path-to-dm-ansible>/conf 中提供了任务配置文件模板：task.yaml.exmaple 文件。该文件是 DM 迁移任务配置的标准文件，每一个具体的任务对应一个 task.yaml 文件。关于该配置文件的详细介绍，参见任务配置文件。

7.1.2.2 创建数据迁移任务

你可以基于 task.yaml.example 文件来创建数据迁移任务，具体步骤如下：

1. 复制 task.yaml.example 为 your_task.yaml。
2. 参考任务配置文件来修改 your_task.yaml 文件。
3. 使用 dmctl 创建数据迁移任务。

7.1.2.3 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	inventory.ini 的 source_id；dm-master.toml 的 source-id；task.yaml 的 source-id
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；dmctl 命令行的 -worker / -w flag

7.2 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。

7.2.1 配置文件示例

DM-master 的示例配置文件如下所示：

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

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

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

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

7.2.2 配置项说明

7.2.2.1 Global 配置

配置项	说明
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"。

7.2.2.2 DM-Worker 的配置

配置在 deploy 中，每一个 DM-worker 都需要设置一个 deploy。

配置项	说明
source-id	一个 replication group 或者 MySQL/MariaDB 实例的标识，需要和 DM-worker 中的 source-id 一致。
dm-worker	DM-worker 的服务地址。

7.3 DM-worker 配置文件介绍

本文档主要介绍 DM-worker 的基础配置文件。在一般场景中，用户只需要使用基础配置即可完成 DM-worker 的部署。

完整配置项参考DM-worker 完整配置说明。

7.3.1 配置文件示例

```
## Worker Configuration.

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

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

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

## Server id of slave for binlog replication.
## Each instance (master and slave) in replication group should have
   ↪ different server id.
server-id = 101

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

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

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

7.3.2 配置项说明

7.3.2.1 Global 配置

配置项	说明
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"。
source-id	标识一个 MySQL/MariaDB 实例或者 replication group。

配置项	说明
server-id	DM-worker 作为上游 MySQL/MariaDB slave 来获取 binlog 的 server id，该值在一个 replication group （包括 master 和 slave）中是唯一的。v1.0.2 及以上版本的 DM 会自动生成，不需要手动配置该项。
flavor	上游数据库的类型，目前值可以为 “mysql” 或者 “mariadb”。v1.0.2 及以上版本的 DM 会自动判断上游版本，不需要手动配置该项。
relay-dir	存储 relay log 的目录，默认值为 “./relay_log”。

7.3.2.2 数据库链接配置（from 配置项）

配置项	说明
host	上游数据库的 host。
port	上游数据库的端口。
user	连接数据库使用的用户名。
password	连接数据库使用的密码。注意：需要使用 dmctl 加密后的密码。

> 注意：
>
> 以上配置为部署 DM-worker 的基础配置，一般情况下使用基础配置即可，更多配置项参考DM-worker 完整配置说明。

7.4 DM-worker 完整配置说明

本文完整地介绍 DM-worker 的配置，包括配置文件示例与配置项说明。

7.4.1 配置文件示例

```
## Worker Configuration.

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

## DM-worker listening address.
```

```
worker-addr = ":8262"

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

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

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

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

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

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

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

## Relay log purge strategy.
[purge]
interval = 3600
expires = 24
remain-space = 15

## Task status checker.
[checker]
check-enable = true
backoff-rollback = "5m"
backoff-max = "5m"
```

7.4.2 配置项说明

7.4.2.1 Global 配置

配置项	说明
log-level	日志等级，值可以为 “debug”、“info”、“warn”、“error”、“fatal”，默认值为 “info”。一般情况下不需要手动配置，如果需要排查问题，可以将等级设置为 “debug”。
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”。
source-id	标识一个 MySQL/MariaDB 实例或者 replication group。
server-id	DM-worker 作为上游 MySQL/MariaDB slave 来获取 binlog 的 server id，该值在一个 replication group （包括 master 和 slave）中必须是唯一的。v1.0.2 及以上版本的 DM 会自动生成，不需要手动配置该项。
flavor	上游数据库的类型，目前值可以为 “mysql” 或者 “mariadb”。v1.0.2 及以上版本的 DM 会自动判断上游版本，不需要手动配置该项。
relay-dir	存储 relay log 的目录，默认值为 “./relay_log”。
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。
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，v1.0.2 之前版本的 DM-worker 将从上游 MySQL 或 MariaDB 现有最早时间点的 binlog 文件开始拉取 binlog，拉取到数据迁移任务需要的最新 binlog 可能需要很长时间；v1.0.2 及之后版本的 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。

配置项	说明
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，v1.0.2 之前版本的 DM-worker 将从上游 MySQL 或 MariaDB 现有最早时间点的 binlog GTID 开始拉取 binlog，拉取到数据迁移任务需要的最新 binlog 可能需要很长时间；v1.0.2 及之后版本的 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。

7.4.2.2 数据库链接配置（from 配置项）

配置项	说明
host	上游数据库的 host。
port	上游数据库的端口。
user	连接数据库使用的用户名。
password	连接数据库使用的密码。注意：需要使用 dmctl 加密后的密码。

7.4.2.3 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，则可以通过该配置项设置，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。
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。

7.4.2.4 任务检查模块配置（checker 配置项）

配置项	说明
check-enable	是否开启任务状态检查。开启后 DM 会尝试自动恢复因错误而暂停的数据迁移任务
backoff-rollback	任务检查模块中，定时调整恢复等待时间的间隔，默认值：“5m0s”。
backoff-max	任务检查模块中，检查出错误后等待自动恢复的最长时间间隔，默认值：“5m0s”。

注意：

用户只需要通过配置 check-enable 开启或者关闭任务状态检查功能。对于 backoff-rollback 和 backoff-max 一般情况下不需要修改，如果对该参数的作用没有深入的了解，不建议修改这两项参数。

7.5 DM 任务配置文件介绍

本文档主要介绍 Data Migration (DM) 的任务基础配置文件 task_basic.yaml，包含全局配置和实例配置两部分。

完整的任务配置参见DM 任务完整配置文件介绍。关于各配置项的功能和配置，请参阅数据迁移功能。

7.5.1 关键概念

关于包括 source-id 和 DM-worker ID 在内的关键概念的介绍，请参阅关键概念。

7.5.2 基础配置文件示例

下面是一个基础的配置文件示例，通过该示例可以完成简单的数据迁移功能。

```
---

## ----------- 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"
```

```
target-database:     # 下游数据库实例配置
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: ""       # 如果不为空则需经过 dmctl 加密

### ******** 功能配置集 **********
block-allow-list:     # 上游数据库实例匹配的表的 block & allow list
    ↪ 过滤规则集，如果 DM 版本 <= v1.0.6 则使用 black-white-list
  bw-rule-1:          # 黑白名单配置的名称
    do-dbs: ["all_mode"] # 迁移哪些库

## ----------- 实例配置 -----------
mysql-instances:
  - source-id: "mysql-replica-01" # 上游实例或者复制组 ID，参考 `dm-master.
      ↪ toml` 的 `source-id` 配置
    block-allow-list: "bw-rule-1" # 黑白名单配置名称，如果 DM 版本 <= v1.0.6
       ↪ 则使用 black-white-list
    mydumper-thread: 4         # dump 处理单元用于导出数据的线程数量，在 v1
       ↪ .0.2 版本引入
    loader-thread: 16          # load 处理单元用于导入数据的线程数量，在 v1
       ↪ .0.2 版本引入
    syncer-thread: 16          # sync 处理单元用于复制增量数据的线程数量，在
       ↪ v1.0.2 版本引入

  - source-id: "mysql-replica-02" # 上游实例或者复制组 ID，参考 `dm-master.
      ↪ toml` 的 `source-id` 配置
    block-allow-list: "bw-rule-1" # 黑白名单配置名称，如果 DM 版本 <= v1.0.6
       ↪ 则使用 black-white-list
    mydumper-thread: 4         # dump 处理单元用于导出数据的线程数量，在 v1
       ↪ .0.2 版本引入
    loader-thread: 16          # load 处理单元用于导入数据的线程数量，在 v1
       ↪ .0.2 版本引入
    syncer-thread: 16          # sync 处理单元用于复制增量数据的线程数量，在
       ↪ v1.0.2 版本引入
```

7.5.3 配置顺序

通过上面的配置文件示例，可以看出配置文件总共分为两个部分：全局配置和实例配置，其中全局配置又分为基础信息配置和功能配置集，配置顺序如下：

1. 编辑全局配置。
2. 根据全局配置编辑实例配置。

7.5.4 全局配置

7.5.4.1 任务基本信息配置

配置任务的基本信息，配置项的说明参见以上示例配置文件中的注释。关于 task-mode 的特殊说明如下：

- 描述：任务模式，可以通过任务模式来指定需要执行的数据迁移工作。
- 值为字符串（full，incremental 或 all）。
  - full：只全量备份上游数据库，然后将数据全量导入到下游数据库。
  - incremental：只通过 binlog 把上游数据库的增量修改复制到下游数据库，可以设置实例配置的 meta 配置项来指定增量复制开始的位置。
  - all：full + incremental。先全量备份上游数据库，将数据全量导入到下游数据库，然后从全量数据备份时导出的位置信息 (binlog position) 开始通过 binlog 增量复制数据到下游数据库。

7.5.4.2 功能配置集

对于一般的业务场景，只需要配置黑白名单过滤规则集，配置说明参见以上示例配置文件中 block-allow-list 的注释以及Block & Allow Table Lists

7.5.5 实例配置

本小节定义具体的数据迁移子任务，DM 支持从单个或者多个上游 MySQL 实例迁移数据到同一个下游数据库实例。配置项说明参见以上示例配置文件中 mysql-instances 的注释。

7.5.6 修改任务配置

在某些情况下需要更新任务配置内容，比如重置数据迁移任务时设置了 remove-meta 为 true 和 task-mode 为 all，当数据迁移任务重置完成后，需要更新任务配置文件中的 remove-meta 设置为 false，防止下一次启动任务的时候重新迁移任务。

因为 DM 集群会持久化保存任务配置，所以修改任务配置需要通过 stop-task、start-task 将修改后的配置更新到 DM 集群中，如果直接修改任务配置文件，但是不重启任务，配置变更不会生效，DM 集群重启时仍然会读取之前保存的任务配置。

这里以修改 remove-meta 配置为例，举例说明任务配置修改步骤：

1. 修改任务配置文件，将 remove-meta 设置为 false
2. 通过 stop-task 命令停止任务：stop-task <task-name | task-file>
3. 通过 start-task 命令启动任务：start-task <config-file>

7.6 DM 任务完整配置文件介绍

本文档主要介绍 Data Migration (DM) 的任务完整的配置文件 task_advanced.yaml，包含全局配置 和实例配置 两部分。

关于各配置项的功能和配置，请参阅数据迁移功能。

7.6.1 关键概念

关于包括 source-id 和 DM-worker ID 在内的关键概念的介绍，请参阅关键概念。

7.6.2 关闭检查项

DM 会根据任务类型进行相应检查。可以参考关闭检查项，在任务配置文件中使用 ignore-checking-items 配置关闭相应检查。

7.6.3 完整配置文件示例

下面是一个完整的配置文件示例，通过该示例可以完成复杂的数据迁移功能。

```
---

## ----------- 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"
is-sharding: true             # 是否为分库分表合并任务
meta-schema: "dm_meta"        # 下游储存 `meta` 信息的数据库
remove-meta: false            # 是否在开始运行任务前移除该任务名对应的 `meta
    ↪ `（`checkpoint` 和 `onlineddl` 等）。
enable-heartbeat: false       # 是否开启 `heartbeat` 功能
online-ddl-scheme: "gh-ost" # 目前仅支持 "gh-ost" 、"pt"
case-sensitive: false         # schema/table 是否大小写敏感
ignore-checking-items: []     # 不关闭任何检查项
clean-dump-file: true         # 是否清理 dump 阶段产生的文件，包括 metadata
    ↪ 文件、建库建表 SQL 文件以及数据导入 SQL 文件。v1.0.7 新增

target-database:              # 下游数据库实例配置
  host: "192.168.0.1"
  port: 4000
  user: "root"
  password: "/Q7B9DizNLLTTfiZHv9WoEAKamfpIUs=" # 推荐使用经 dmctl
      ↪ 加密后的密码
  session:                                    # 设置 TiDB 的 session 变量，在 v1
      ↪ .0.6 版本引入。更多变量及解释参见 `https://docs.pingcap.com/zh/tidb/
      ↪ stable/system-variables`
```

```
    sql_mode: "ANSI_QUOTES,NO_ZERO_IN_DATE,NO_ZERO_DATE"
    tidb_skip_utf8_check: 1
    tidb_constraint_check_in_place: 0


### ******** 功能配置集 **********

routes:                         # 上游和下游表之间的路由 table routing 规则集
  route-rule-1:                 # 配置名称
    schema-pattern: "test_*"    # 库名匹配规则，支持通配符 "*" 和 "?"
    table-pattern: "t_*"        # 表名匹配规则，支持通配符 "*" 和 "?"
    target-schema: "test"       # 目标库名称
    target-table: "t"           # 目标表名称
  route-rule-2:
    schema-pattern: "test_*"
    target-schema: "test"

filters:                                    # 上游数据库实例匹配的表的 binlog
    ↪ event filter 规则集
  filter-rule-1:                            # 配置名称
    schema-pattern: "test_*"                # 库名匹配规则，支持通配符 "*" 和
        ↪ "?"
    table-pattern: "t_*"                    # 表名匹配规则，支持通配符 "*" 和
        ↪ "?"
    events: ["truncate table", "drop table"] # 匹配哪些 event 类型
    action: Ignore                          # 对与符合匹配规则的 binlog 复制 (
        ↪ Do ) 还是忽略(Ignore)
  filter-rule-2:
    schema-pattern: "test_*"
    events: ["all dml"]
    action: Do

block-allow-list:                   # 上游数据库实例匹配的表的 block & allow
    ↪ list 过滤规则集，如果 DM 版本 <= v1.0.6 则使用 black-white-list
  bw-rule-1:                        # 配置名称
    do-dbs: ["~^test.*", "user"] # 迁移哪些库
    ignore-dbs: ["mysql", "account"] # 忽略哪些库
    do-tables:                      # 迁移哪些表
    - db-name: "~^test.*"
      tbl-name: "~^t.*"
    - db-name: "user"
      tbl-name: "information"
    ignore-tables:                  # 忽略哪些表
    - db-name: "user"
      tbl-name: "log"
```

```
mydumpers:                          # dump 处理单元运行配置参数
  global:                           # 配置名称
    mydumper-path: "./bin/mydumper" # dump 处理单元 binary 文件地址，默认值为
       ↪  "./bin/mydumper"
    threads: 4                      # dump
       ↪ 处理单元从上游数据库实例导出数据的线程数量，默认值为 4
    chunk-filesize: 64              # dump 处理单元生成的数据文件大小，默认值为
       ↪ 64，单位为 MB
    skip-tz-utc: true               # 忽略对时间类型数据进行时区转化，默认值为
       ↪ true
    extra-args: "--no-locks"        # dump 处理单元的其他参数，在 v1.0.2 版本中
       ↪ DM 会自动生成 table-list 配置，在其之前的版本仍然需要人工配置

loaders:                            # load 处理单元运行配置参数
  global:                           # 配置名称
    pool-size: 16                   # load 处理单元并发执行 dump 处理单元导出的
       ↪ SQL 文件的线程数量，默认值为 16
    dir: "./dumped_data"            # load 处理单元读取 dump
       ↪ 处理单元输出文件的地址，同实例对应的不同任务必须不同（dump
       ↪ 处理单元会根据这个地址输出 SQL 文件），默认值为 "./dumped_data"

syncers:                            # sync 处理单元运行配置参数
  global:                           # 配置名称
    worker-count: 16                # sync 处理单元并发迁移 binlog event
       ↪ 的线程数量，默认值为 16
    batch: 100                      # sync
       ↪ 处理单元迁移到下游数据库的一个事务批次 SQL 语句数，默认值为 100
    enable-ansi-quotes: true        # 若 `session` 中设置 `sql-mode: "
       ↪ ANSI_QUOTES"`，则需开启此项
    safe-mode: false                # 设置为 true，则将来自上游的 `INSERT`
       ↪ 改写为 `REPLACE`，将 `UPDATE` 改写为 `DELETE` 与 `REPLACE`，
       ↪ 保证在表结构中存在主键或唯一索引的条件下迁移数据时可以重复导入 DML。
       ↪ 在启动或恢复增量复制任务的前 5 分钟内 TiDB DM 会自动启动 safe mode

## ----------- 实例配置 -----------
mysql-instances:
  -
    source-id: "mysql-replica-01"      # 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
    meta:                              # `task-mode` 为 `incremental`
       ↪ 且下游数据库的 `checkpoint` 不存在时 binlog 复制开始的位置；如果
       ↪ checkpoint 存在，则以 `checkpoint` 为准
      binlog-name: binlog.000001
```

```
    binlog-pos: 4

  route-rules: ["route-rule-1", "route-rule-2"] #
     ↪ 该上游数据库实例匹配的表到下游数据库的 table routing 规则名称
  filter-rules: ["filter-rule-1"]          # 该上游数据库实例匹配的表的
     ↪ binlog event filter 规则名称
  block-allow-list: "bw-rule-1"            # 该上游数据库实例匹配的表的
     ↪ block & allow list 过滤规则名称，如果 DM 版本 <= v1.0.6 则使用
     ↪ black-white-list

  mydumper-config-name: "global"     # dump 处理单元配置名称
  loader-config-name: "global"       # load 处理单元配置名称
  syncer-config-name: "global"       # sync 处理单元配置名称

-
  source-id: "mysql-replica-02" # 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
  mydumper-thread: 4         # dump 处理单元用于导出数据的线程数量，等同于
     ↪ dump 处理单元配置中的 `threads`，在 v1.0.2 版本引入
  loader-thread: 16          # load 处理单元用于导入数据的线程数量，等同于
     ↪ load 处理单元配置中的 `pool-size`，在 v1.0.2 版本引入
  syncer-thread: 16          # sync 处理单元用于复制增量数据的线程数量，
     ↪ 等同于 sync 处理单元配置中的 `worker-count`，在 v1.0.2 版本引入
```

7.6.4 配置顺序

通过上面的配置文件示例，可以看出配置文件总共分为两个部分：全局配置和实例配置，其中全局配置又分为基本信息配置和实例配置，配置顺序如下：

1. 编辑全局配置。
2. 根据全局配置编辑实例配置。

7.6.5 全局配置

7.6.5.1 任务基本信息配置

配置任务的基本信息，配置项的说明参见以上示例配置文件中的注释。其中 task-mode 需要特殊说明：

task-mode

- 描述：任务模式，可以通过任务模式来指定需要执行的数据迁移工作。
- 值为字符串（full，incremental 或 all）。
  - full：只全量备份上游数据库，然后将数据全量导入到下游数据库。

- `incremental`：只通过 binlog 把上游数据库的增量修改复制到下游数据库，可以设置实例配置的 `meta` 配置项来指定增量复制开始的位置。
- `all`：`full` + `incremental`。先全量备份上游数据库，将数据全量导入到下游数据库，然后从全量数据备份时导出的位置信息 (binlog position) 开始通过 binlog 增量复制数据到下游数据库。

7.6.5.2 功能配置集

全局配置主要包含下列功能配置集：

配置项	说明
`routes`	上游和下游表之间的路由 table routing 规则集。如果上游与下游的库名、表名一致，则不需要配置该项。使用场景及示例配置参见Table Routing
`filters`	上游数据库实例匹配的表的 binlog event filter 规则集。如果不需要对 binlog 进行过滤，则不需要配置该项。使用场景及示例配置参见Binlog Event Filter
`block-allow-list`	该上游数据库实例匹配的表的 block & allow list 过滤规则集。建议通过该项指定需要迁移的库和表，否则会迁移所有的库和表。使用场景及示例配置参见Block & Allow Lists
`mydumpers`	dump 处理单元运行配置参数。如果默认配置可以满足需求，则不需要配置该项，也可以只使用 `mydumper-thread` 对 `thread` 配置项单独进行配置。
`loaders`	load 处理单元运行配置参数。如果默认配置可以满足需求，则不需要配置该项，也可以只使用 `loader-thread` 对 `pool-size` 配置项单独进行配置。
`syncers`	sync 处理单元运行配置参数。如果默认配置可以满足需求，则不需要配置该项，也可以只使用 `syncer-thread` 对 `worker-count` 配置项单独进行配置。

各个功能配置集的参数及解释参见完整配置文件示例中的注释说明。

7.6.6 实例配置

本小节定义具体的数据迁移子任务，DM 支持从单个或者多个上游 MySQL 实例迁移数据到同一个下游数据库实例。

在该项配置中设置数据迁移子任务中各个功能对应的配置集中的配置名称，关于这些配置项的更多配置细节，参见功能配置集的相关配置项，对应关系如下：

配置项	相关配置项
route-rules	routes
filter-rules	filters
block-allow-list	block-allow-list
mydumper-config-name	mydumpers
loader-config-name	loaders
syncer-config-name	syncers

8 DM 集群管理

8.1 DM 集群操作

本文介绍 DM 集群操作以及使用 DM-Ansible 管理 DM 集群时需要注意的事项。

8.1.1 启动集群

运行以下命令以启动整个集群的所有组件（包括 DM-master、DM-worker 和监控组件）：

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

8.1.2 下线集群

运行以下命令以下线整个集群的所有组件（包括 DM-master、DM-worker 和监控组件）：

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

8.1.3 重启集群组件

在以下情况下，需要更新 DM 集群组件：

- 您想要更新组件版本。
- 发生了严重的错误，您需要重启组件完成临时恢复。
- DM 集群所在的机器由于某种原因重启。

8.1.3.1 重启注意事项

该部分描述重启 DM 各组件时需要了解的事项。

8.1.3.1.1 DM-worker 重启事项

全量数据导入过程中：

对于全量数据导入时的 SQL 文件，DM 使用下游数据库记录断点信息，DM-worker 会在本地 meta 文件记录子任务信息。DM-worker 重启时会检查断点信息和本地记录的子任务信息，重启前处于运行中状态的任务会自动恢复数据迁移。

增量数据复制过程中：

对于增量数据导入过程中的 binlog，DM 使用下游数据库记录断点信息，并会在复制任务开始或恢复后的第一个五分钟之内开启安全模式。

- 未启用 sharding DDL 迁移

  如果 DM-worker 上运行的任务未启用 sharding DDL 迁移功能，DM-worker 重启时会检查断点信息和本地记录的子任务信息，重启前处于运行中状态的任务会自动恢复数据迁移。

- 已启用 sharding DDL 迁移

  - DM 迁移 sharding DDL 语句时，如果 DM-worker 成功执行（或跳过）sharding DDL 的 binlog event，与 DM-worker 中的 sharding DDL 语句相关的所有表的断点信息都会被更新至 DDL 语句对应的 binlog event 之后的位置。
  - 当 DM-worker 重启发生在 sharding DDL 语句迁移开始前或完成后，DM-worker 会根据断点信息和本地记录的子任务信息自动恢复数据迁移。
  - 当 DM-worker 重启发生在 sharding DDL 语句迁移过程中，可能会出现作为 DDL lock owner 的 DM-worker 实例已执行了 DDL 语句并成功变更了下游数据库表结构，但其他 DM-worker 实例重启而无法跳过 DDL 语句也无法更新断点的情况。

  此时 DM 会再次尝试迁移这些未跳过执行的 DDL 语句。然而，由于未重启的 DM-worker 实例已经执行到了此 DDL 对应的 binlog event 之后，重启的 DM-worker 实例会被阻滞在重启前 DDL binlog event 对应的位置。

  要解决这个问题，请按照手动处理 Sharding DDL Lock 中描述的步骤操作。

总结：尽量避免在 sharding DDL 迁移过程中重启 DM-worker。

8.1.3.1.2 DM-master 重启事项

由 DM-master 维护的信息包括以下两种。重启 DM-master 不会持久化保存这些信息的相关数据。

- 任务信息
- Sharding DDL lock 信息

DM-master 重启时会自动向每个 DM-worker 实例请求任务信息，重建任务与 DM-worker 之间的对应关系，并从每个 DM-worker 实例获取 sharding DDL 信息。这样，就可以准确重建相应的 DDL lock，也可以自动解除 sharding DDL lock。

8.1.3.2 重启 DM-worker

> **注意：**
> 尽量避免在 sharding DDL 迁移过程中重启 DM-worker。

使用以下两种方法中任一种重启 DM-worker 组件：

- 对 DM-worker 执行滚动升级。

```
ansible-playbook rolling_update.yml --tags=dm-worker
```

- 先停止 DM-worker，然后重启。

```
ansible-playbook stop.yml --tags=dm-worker &&
ansible-playbook start.yml --tags=dm-worker
```

8.1.3.3 重启 DM-master

在以下两种方法中任选一种，重启 DM-master 组件：

- 对 DM-master 执行滚动升级。

```
ansible-playbook rolling_update.yml --tags=dm-master
```

- 停止 DM-master，然后重启。

```
ansible-playbook stop.yml --tags=dm-master &&
ansible-playbook start.yml --tags=dm-master
```

8.1.4 更新组件版本

1. 下载 DM 二进制文件。

    1. 从 `downloads` 目录删除已有文件。

    ```
    cd /home/tidb/dm-ansible &&
    rm -rf downloads
    ```

    2. 用 Playbook 下载 inventory.ini 文件中指定版本的最新 DM 二进制文件。这会自动替换 `/home/tidb/dm-ansible/resource/bin/` 中已有文件。

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

2. 使用 TiDB Ansible 执行滚动升级。

    1. 对 DM-worker 实例执行滚动升级：

    ```
    ansible-playbook rolling_update.yml --tags=dm-worker
    ```

    2. 对 DM-master 实例执行滚动升级：

    ```
    ansible-playbook rolling_update.yml --tags=dm-master
    ```

    3. 升级 dmctl：

    ```
    ansible-playbook rolling_update.yml --tags=dmctl
    ```

    4. 对 DM-worker，DM-master，以及 dmctl 整体执行滚动升级：

    ```
    ansible-playbook rolling_update.yml
    ```

8.1.5 创建 DM-worker 实例

假设您想要在机器 172.16.10.74 上创建一个名为 dm_worker3 的 DM-worker 实例，按以下步骤操作：

1. 为中控机设置 SSH 互信以及 sudo 规则。

    1. 参考在中控机上配置 SSH 互信和 sudo 规则，使用 tidb 用户登录至中控机，并将 172.16.10.74 添加至 hosts.ini 文件中的 [servers] 部分。

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

    ```
    [servers]
    172.16.10.74

    [all:vars]
    username = tidb
    ```

    2. 运行以下命令。根据屏幕提示，输入 root 用户密码以部署 172.16.10.74。

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

    该步在 172.16.10.74 机器上创建了一个 tidb 用户，设置了 sudo 规则，并为中控机与该机器配置了 SSH 互信。

2. 修改 inventory.ini 文件，创建新 DM-worker 实例 dm_worker3。

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

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

dm_worker3 source_id="mysql-replica-03" ansible_host=172.16.10.74
    ↪ server_id=103 mysql_host=172.16.10.83 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
```

3. 部署新 DM-worker 实例。

```
ansible-playbook deploy.yml --tags=dm-worker -l dm_worker3
```

4. 启用新 DM-worker 实例。

```
ansible-playbook start.yml --tags=dm-worker -l dm_worker3
```

5. 配置并重启 DM-master 服务。

```
ansible-playbook rolling_update.yml --tags=dm-master
```

6. 配置并重启 Prometheus 服务。

```
ansible-playbook rolling_update_monitor.yml --tags=prometheus
```

8.1.6 下线 DM-worker 实例

假设您想要下线的 DM-worker 实例为 `dm_worker3`。按以下步骤操作：

1. 关闭您想要下线的 DM-worker 实例。

```
ansible-playbook stop.yml --tags=dm-worker -l dm_worker3
```

2. 修改 `inventory.ini` 文件，注释或删除 `dm_worker3` 实例所在行。

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

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

# dm_worker3 source_id="mysql-replica-03" ansible_host=172.16.10.74
    ↪ server_id=103 mysql_host=172.16.10.83 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
    ↪  # Comment or delete this line
```

3. 配置并重启 DM-master 服务。

```
ansible-playbook rolling_update.yml --tags=dm-master
```

4. 配置并重启 Prometheus 服务。

```
ansible-playbook rolling_update_monitor.yml --tags=prometheus
```

8.1.7 替换/迁移 DM-master 实例

假设机器 172.16.10.71 需要进行维护或者已崩溃，需要将 DM-master 实例从 172.16.10.71 迁移至 172.16.10.80。按以下步骤操作：

1. 为中控机设置 SSH 互信以及 sudo 规则。

    1. 参考在中控机上配置 SSH 互信和 sudo 规则，使用 tidb 账户登录至中控机，并将 172.16.10.80 添加至 hosts.ini 文件中的 [servers] 部分。

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

    ```
    [servers]
    172.16.10.80

    [all:vars]
    username = tidb
    ```

    2. 运行以下命令。根据屏幕提示，输入 root 用户密码以部署 172.16.10.80。

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

    该步在 172.16.10.80 机器上创建了一个 tidb 用户，设置了 sudo 规则，并为中控机与该机器配置了 SSH 互信。

2. 关闭待替换的 DM-master 实例。

> **注意：**
> 如果机器 172.16.10.71 宕机，无法通过 SSH 登录，请忽略此步。

```
ansible-playbook stop.yml --tags=dm-master
```

3. 修改 inventory.ini 文件。注释或删除待替换实例所在行，同时为新 DM-master 实例添加相关信息。

```
[dm_master_servers]
# dm_master ansible_host=172.16.10.71
dm_master ansible_host=172.16.10.80
```

4. 部署新 DM-master 实例。

```
ansible-playbook deploy.yml --tags=dm-master
```

5. 启用新 DM-master 实例。

```
ansible-playbook start.yml --tags=dm-master
```

6. 更新 dmctl 配置文件。

```
ansible-playbook rolling_update.yml --tags=dmctl
```

8.1.8 替换/迁移 DM-worker 实例

假设机器 172.16.10.72 需要进行维护或者已崩溃，您需要将 dm_worker1 实例从 172.16.10.72 迁移至 172.16.10.75。按以下步骤操作：

1. 为中控机设置 SSH 互信以及 sudo 规则。

    1. 参考在中控机上配置 SSH 互信和 sudo 规则，使用 tidb 账户登录至中控机，并将 172.16.10.75 添加至 hosts.ini 文件中的 [servers] 部分。

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

```
[servers]
172.16.10.75

[all:vars]
username = tidb
```

2. 运行以下命令。根据屏幕提示，输入 root 用户密码以部署 172.16.10.85。

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

该步在 172.16.10.75 上创建了一个 tidb 用户，设置了 sudo 规则，并为中控机与该机器配置了 SSH 互信。

2. 下线待替换 DM-worker 实例。

> **注意：**
> 如果机器 172.16.10.71 宕机，无法通过 SSH 登录，请忽略此步。

```
ansible-playbook stop.yml --tags=dm-worker -l dm_worker1
```

3. 修改 inventory.ini 文件，为新 DM-worker 实例添加相关信息。

修改 inventory.ini 文件。注释或删除旧 dm_worker1 实例所在行；同时为新 dm_worker1 实例添加相关信息。

如果希望从不同的 binlog position 或 GTID Sets 拉取 relay log，则也需要更新对应的 {relay_binlog_name} 或 {relay_binlog_gtid}。

```
[dm_worker_servers]
dm_worker1 source_id="mysql-replica-01" ansible_host=172.16.10.75
    ↪ server_id=101 mysql_host=172.16.10.81 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
# dm_worker1 source_id="mysql-replica-01" ansible_host=172.16.10.72
    ↪ server_id=101 mysql_host=172.16.10.81 mysql_user=root
    ↪ mysql_password='VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306

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

4. 部署新 DM-worker 实例。

```
ansible-playbook deploy.yml --tags=dm-worker -l dm_worker1
```

5. 迁移 relay log 数据。

- 如果待替换 DM-worker 实例所在机器仍能访问，则可直接将该实例的 {↪ dm_worker_relay_dir} 目录下的所有数据复制到新 DM-worker 实例的对应目录。
- 如果待替换 DM-worker 实例所在机器已无法访问，可能需在第 9 步中手动恢复 relay log 目录等信息。

6. 启动新 DM-worker 实例。

```
ansible-playbook start.yml --tags=dm-worker -l dm_worker1
```

7. 配置并重启 DM-master 服务。

```
ansible-playbook rolling_update.yml --tags=dm-master
```

8. 配置并重启 Prometheus 服务。

```
ansible-playbook rolling_update_monitor.yml --tags=prometheus
```

9. 启动并验证数据迁移任务。

   使用 `start-task` 命令启动数据迁移任务，如果任务运行正常，则表示 DM-worker 迁移顺利完成；如果报类似如下错误，则需要对 relay log 目录进行手动修复。

```
fail to initial unit Sync of subtask test-task : UUID suffix 000002
    ↪ with UUIDs [1ddbf6d3-d3b2-11e9-a4e9-0242ac140003.000001] not
    ↪ found
```

   如果待替换 DM-worker 所连接的上游 MySQL 已发生过切换，则会产生如上错误。此时可通过如下步骤手动修复：

   1. 使用 `stop-task` 命令停止数据迁移任务。
   2. 通过 `$ ansible-playbook stop.yml --tags=dm-worker -l dm_worker1` 停止 DM-worker 实例。
   3. 更新 relay log 子目录的后缀，例如将 `1ddbf6d3-d3b2-11e9-a4e9-0242ac140003 ↪ .000001` 重命名为 `1ddbf6d3-d3b2-11e9-a4e9-0242ac140003.000002`。
   4. 更新 relay log 子目录索引文件 `server-uuid.index`，例如将其中的内容由 `1ddbf6d3-d3b2-11e9-a4e9-0242ac140003.000001` 变更为 `1ddbf6d3-d3b2-11 ↪ e9-a4e9-0242ac140003.000002`。
   5. 通过 `$ ansible-playbook start.yml --tags=dm-worker -l dm_worker1` 启动 DM-worker 实例。
   6. 再次启动并验证数据迁移任务。

8.2 TiDB Data Migration 版本升级

本文档主要介绍各 TiDB Data Migration (DM) 版本间的升级操作步骤以及各版本的版本信息和主要变更。

> 注意：

- 若无特殊说明，各版本的升级操作均为从前一个有升级指引的版本向当前版本升级。
- 若无特殊说明，各升级操作示例均假定已经下载了对应版本的 DM 和 DM-Ansible 且 DM binary 存在于 DM-Ansible 的相应目录中（下载 DM binary 可以参考更新组件版本）。
- 若无特殊说明，各升级操作示例均假定升级前已停止所有迁移任务，升级完成后手动重新启动所有迁移任务。
- 以下版本升级指引逆序展示。

8.2.1 升级到 v1.0.5

8.2.1.1 版本信息

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

8.2.1.2 主要变更

- 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
- 修复了全量数据导入过程中有可能触发 `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

8.2.1.3 升级操作示例

1. 下载新版本 DM-Ansible，确认 `inventory.ini` 文件中 `dm_version = v1.0.5`
2. 执行 `ansible-playbook local_prepare.yml` 下载新的 DM binary 到本地
3. 执行 `ansible-playbook rolling_update.yml` 滚动升级 DM 集群组件
4. 执行 `ansible-playbook rolling_update_monitor.yml` 滚动升级 DM 监控组件

8.2.2 升级到 v1.0.4

8.2.2.1 版本信息

```
Release Version: v1.0.4-1-gd681c67
Git Commit Hash: d681c6731d3432f4d8f38ea651f44d49d6860269
Git Branch: release-1.0
```

```
UTC Build Time: 2020-03-16 09:45:29
Go Version: go version go1.13 linux/amd64
```

8.2.2.2 主要变更

- 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
- 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 异常返回的 query-error 可能导致 DM-worker panic 的问题
- 修复 relay.meta 写入完成前，DM-worker 进程异常停止，导致重启 DM-worker 时可能无法正确恢复 relay log 文件与 relay.meta 的问题

8.2.2.3 升级操作示例

1. 下载新版本 DM-Ansible，确认 inventory.ini 文件中 dm_version = v1.0.4
2. 执行 ansible-playbook local_prepare.yml 下载新的 DM binary 到本地
3. 执行 ansible-playbook rolling_update.yml 滚动升级 DM 集群组件
4. 执行 ansible-playbook rolling_update_monitor.yml 滚动升级 DM 监控组件

8.2.3 升级到 v1.0.3

8.2.3.1 版本信息

```
Release Version: v1.0.3
Git Commit Hash: 41426af6cffcff9a325697a3bdebeadc9baa8aa6
Git Branch: release-1.0
UTC Build Time: 2019-12-13 07:04:53
Go Version: go version go1.13 linux/amd64
```

8.2.3.2 主要变更

- 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
- 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

8.2.3.3 升级操作示例

1. 下载新版本 DM-Ansible，确认 `inventory.ini` 文件中 `dm_version = v1.0.3`
2. 执行 `ansible-playbook local_prepare.yml` 下载新的 DM binary 到本地
3. 执行 `ansible-playbook rolling_update.yml` 滚动升级 DM 集群组件
4. 执行 `ansible-playbook rolling_update_monitor.yml` 滚动升级 DM 监控组件

> **注意：**
>
> 更新至 DM 1.0.3 版本时，需要确保 DM 所有组件 (dmctl/DM-master/DM-worker) 同时升级。不支持部分组件升级使用。

8.2.4 升级到 v1.0.2

8.2.4.1 版本信息

```
Release Version: v1.0.2
Git Commit Hash: affc6546c0d9810b0630e85502d60ed5c800bf25
Git Branch: release-1.0
UTC Build Time: 2019-10-30 05:08:50
Go Version: go version go1.12 linux/amd64
```

8.2.4.2 主要变更

- 支持自动为 DM-worker 生成部分配置项，减少人工配置成本
- 支持自动生成 mydumper 库表参数，减少人工配置成本
- 优化 `query-status` 默认输出，突出重点信息
- 直接管理到下游的 DB 连接而不是使用内置连接池，优化 SQL 错误处理与重试
- 修复 DM-worker 进程启动时、执行 DML 失败时可能 panic 的 bug
- 修复执行 sharding DDL（如 `ADD INDEX`）超时后可能造成后续 sharding DDL 无法正确协调的 bug
- 修复了有部分 DM-worker 不可访问时无法 `start-task` 的 bug
- 完善了对 1105 错误的自动重试策略

8.2.4.3 升级操作示例

1. 下载新版本 DM-Ansible, 确认 `inventory.ini` 文件中 `dm_version = v1.0.2`
2. 执行 `ansible-playbook local_prepare.yml` 下载新的 DM binary 到本地
3. 执行 `ansible-playbook rolling_update.yml` 滚动升级 DM 集群组件
4. 执行 `ansible-playbook rolling_update_monitor.yml` 滚动升级 DM 监控组件

注意：

更新至 DM 1.0.2 版本时，需要确保 DM 所有组件 (dmctl/DM-master/DM-worker) 同时升级。不支持部分组件升级使用。

8.2.5 升级到 v1.0.1

8.2.5.1 版本信息

```
Release Version: v1.0.1
Git Commit Hash: e63c6cdebea0edcf2ef8c91d84cff4aaa5fc2df7
Git Branch: release-1.0
UTC Build Time: 2019-09-10 06:15:05
Go Version: go version go1.12 linux/amd64
```

8.2.5.2 主要变更

- 修复某些情况下 DM 会频繁重建数据库连接的问题
- 修复使用 query-status 时潜在的 panic 问题

8.2.5.3 升级操作示例

1. 下载新版本 DM-Ansible, 确认 `inventory.ini` 文件中 `dm_version = v1.0.1`
2. 执行 `ansible-playbook local_prepare.yml` 下载新的 DM binary 到本地
3. 执行 `ansible-playbook rolling_update.yml` 滚动升级 DM 集群组件
4. 执行 `ansible-playbook rolling_update_monitor.yml` 滚动升级 DM 监控组件

注意：

更新至 DM 1.0.1 版本时，需要确保 DM 所有组件 (dmctl/DM-master/DM-worker) 同时升级。不支持部分组件升级使用。

8.2.6 升级到 v1.0.0-10-geb2889c9 (1.0 GA)

8.2.6.1 版本信息

```
Release Version: v1.0.0-10-geb2889c9
Git Commit Hash: eb2889c9dcfbff6653be9c8720a32998b4627db9
Git Branch: release-1.0
UTC Build Time: 2019-09-06 03:18:48
Go Version: go version go1.12 linux/amd64
```

8.2.6.2 主要变更

- 常见的异常场景支持自动尝试恢复迁移任务
- 提升 DDL 语法兼容性
- 修复上游数据库连接异常时可能丢失数据的 bug

8.2.6.3 升级操作示例

1. 下载新版本 DM-Ansible, 确认 `inventory.ini` 文件中 `dm_version = v1.0.0`
2. 执行 `ansible-playbook local_prepare.yml` 下载新的 DM binary 到本地
3. 执行 `ansible-playbook rolling_update.yml` 滚动升级 DM 集群组件
4. 执行 `ansible-playbook rolling_update_monitor.yml` 滚动升级 DM 监控组件

> **注意：**
>
> 更新至 DM 1.0 GA 版本时，需要确保 DM 所有组件 (dmctl/DM-master/DM-worker) 同时升级。不支持部分组件升级使用。

8.2.7 升级到 v1.0.0-rc.1-12-gaa39ff9

8.2.7.1 版本信息

```
Release Version: v1.0.0-rc.1-12-gaa39ff9
Git Commit Hash: aa39ff981dfb3e8c0fa4180127246b253604cc34
Git Branch: dm-master
UTC Build Time: 2019-07-24 02:26:08
Go Version: go version go1.11.2 linux/amd64
```

8.2.7.2 主要变更

从此版本开始，将对所有的配置进行严格检查，遇到无法识别的配置会报错，以确保用户始终准确地了解自己的配置。

8.2.7.3 升级操作示例

启动 DM-master 或 DM-worker 前，必须确保已经删除废弃的配置信息，且没有多余的配置项，否则会启动失败。可根据失败信息删除多余的配置。

可能遗留的废弃配置：

- `dm-worker.toml` 中的 `meta-file`
- `task.yaml` 中的 `mysql-instances` 中的 `server-id`

9 DM 迁移任务管理

9.1 管理数据迁移任务

本文介绍了如何使用dmctl 组件来进行数据迁移任务的管理和维护。对于用 DM-Ansible 部署的 DM 集群，dmctl 二进制文件路径为 `dm-ansible/dmctl`。

dmctl 支持交互模式用于人工操作，同时也支持命令模式用于脚本。

9.1.1 dmctl 交互模式

本部分描述了在交互模式下一些 dmctl 命令的基本用法。

> **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。

9.1.1.1 dmctl 使用帮助

```
./dmctl --help
```

```
Usage of dmctl:
 -V prints version and exit
 -config string
      path to config file
 # 按照 DM 提供的加密方法加密数据库密码，用于 DM 的配置文件
 -encrypt string
      encrypt plaintext to ciphertext
 # DM-master 访问地址，dmctl 与 DM-master 交互以完成任务管理操作
 -master-addr string
      master API server addr
```

```
 -rpc-timeout string
      rpc timeout, default is 10m (default "10m")
```

9.1.1.2 加密数据库密码

在 DM 相关配置文件中，要求必须使用经 dmctl 加密后的密码，否则会报错。对于同一个原始密码，每次加密后密码不同。

```
./dmctl -encrypt 123456
```

```
VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=
```

9.1.1.3 任务管理概览

进入交互模式，与 DM-master 进行交互：

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

```
Welcome to dmctl
Release Version: v1.0.1
Git Commit Hash: e63c6cdebea0edcf2ef8c91d84cff4aaa5fc2df7
Git Branch: release-1.0
UTC Build Time: 2019-09-10 06:15:05
Go Version: go version go1.12 linux/amd64

» help
DM control

Usage:
  dmctl [command]

Available Commands:
  break-ddl-lock     forcefully break DM-worker's DDL lock
  check-task         check the config file of the task
  help               help about any command
  migrate-relay      migrate DM-worker's relay unit
  pause-relay        pause DM-worker's relay unit
  pause-task         pause a specified running task
  purge-relay        purge relay log files of the DM-worker according to the
     ↪  specified filename
  query-error        query task error
  query-status       query task status
  refresh-worker-tasks refresh worker -> tasks mapper
  resume-relay       resume DM-worker's relay unit
  resume-task        resume a specified paused task
```

```
  show-ddl-locks     show un-resolved DDL locks
  sql-inject         inject (limited) SQLs into binlog replication unit as
     ↪ binlog events
  sql-replace        replace SQLs matched by a specific binlog position (
     ↪ binlog-pos) or a SQL pattern (sql-pattern); each SQL must end with a
     ↪  semicolon
  sql-skip           skip the binlog event matched by a specific binlog
     ↪ position (binlog-pos) or a SQL pattern (sql-pattern)
  start-task         start a task as defined in the config file
  stop-task          stop a specified task
  switch-relay-master switch the master server of the DM-worker's relay unit
  unlock-ddl-lock    forcefully unlock DDL lock
  update-master-config update the config of the DM-master
  update-relay       update the relay unit config of the DM-worker
  update-task        update a task's config for routes, filters, or block-
     ↪ allow-list

Flags:
  -h, --help           help for dmctl
  -w, --worker strings DM-worker ID

## 使用 `dmctl [command] --help` 来获取某个命令的更多信息
```

9.1.2 管理数据迁移任务

本部分描述了如何使用不同的任务管理命令来执行相应操作。

9.1.2.1 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
```

```
start a task as defined in the config file

Usage:
 dmctl start-task [-w worker ...] <config-file> [flags]

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

Global Flags:
 -w, --worker strings DM-worker ID
```

9.1.2.1.1 命令用法示例

```
start-task [ -w "172.16.30.15:8262"] ./task.yaml
```

9.1.2.1.2 参数解释

- `-w`:
  - 可选
  - 指定在特定的一组 DM-workers 上执行 `task.yaml`
  - 如果设置，则只启动指定任务在该组 DM-workers 上的子任务
- `config-file`:
  - 必选
  - 指定 `task.yaml` 的文件路径

9.1.2.1.3 返回结果示例

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

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "172.16.30.15:8262",
            "msg": ""
        },
        {
            "result": true,
            "worker": "172.16.30.16:8262",
            "msg": ""
        }
    ]
}
```

9.1.2.2 查询数据迁移任务状态

`query-status` 命令用于查询数据迁移任务状态。有关查询结果及子任务状态，详见查询状态。

```
help query-status
```

```
query task status

Usage:
 dmctl query-status [-w worker ...] [task-name] [flags]

Flags:
 -h, --help help for query-status

Global Flags:
 -w, --worker strings DM-worker ID
```

9.1.2.2.1 命令用法示例

```
query-status
```

9.1.2.2.2 参数解释

- `-w`:
  - 可选
  - 查询在指定的一组 DM-workers 上运行的数据迁移任务的子任务
- `task-name`:
  - 可选
  - 指定任务名称
  - 如果未设置，则返回全部数据迁移任务的查询结果

9.1.2.2.3 返回结果示例

有关查询结果中各参数的意义，详见查询状态结果。

9.1.2.3 查询运行错误

`query-error` 可用于查询数据迁移任务与 relay 处理单元的错误信息。相比于 `query-status`，`query-error` 一般不用于获取除错误信息之外的其他信息。

`query-error` 常用于获取 `sql-skip`/`sql-replace` 所需的 binlog position 信息，有关 `query-error` 的参数与结果解释，请参考“跳过或替代执行异常的 SQL 语句” 文档中的 query-error。

9.1.2.4 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 [-w worker ...] <task-name | task-file> [flags]

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

Global Flags:
 -w, --worker strings DM-worker ID
```

9.1.2.4.1 命令用法示例

```
pause-task [-w "127.0.0.1:8262"] task-name
```

9.1.2.4.2 参数解释

- -w:

  - 可选
  - 指定在特定的一组 DM-workers 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 DM-workers 上的子任务

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

9.1.2.4.3 返回结果示例

```
pause-task test
```

```
{
    "op": "Pause",
    "result": true,
    "msg": "",
    "workers": [
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.15:8262",
                "msg": ""
            },
            "op": "Pause",
            "logID": "2"
        },
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.16:8262",
                "msg": ""
            },
            "op": "Pause",
            "logID": "2"
        }
    ]
}
```

9.1.2.5 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 [-w worker ...] <task-name | task-file> [flags]

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

Global Flags:
 -w, --worker strings DM-worker ID
```

9.1.2.5.1 命令用法示例

```
resume-task [-w "127.0.0.1:8262"] task-name
```

9.1.2.5.2 参数解释

- `-w`:
  - 可选
  - 指定在特定的一组 DM-workers 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 DM-workers 上的子任务
- `task-name | task-file`:
  - 必选
  - 指定任务名称或任务文件路径

9.1.2.5.3 返回结果示例

```
resume-task test
```

```
{
    "op": "Resume",
    "result": true,
    "msg": "",
    "workers": [
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.15:8262",
                "msg": ""
            },
            "op": "Resume",
```

```
            "logID": "3"
        },
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.16:8262",
                "msg": ""
            },
            "op": "Resume",
            "logID": "3"
        }
    ]
}
```

9.1.2.6 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 [-w worker ...] <task-name | task-file> [flags]

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

Global Flags:
 -w, --worker strings DM-worker ID
```

9.1.2.6.1 命令用法示例

```
stop-task [-w "127.0.0.1:8262"] task-name
```

9.1.2.6.2 参数解释

- -w:
  - 可选
  - 指定在特定的一组 DM-workers 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 DM-workers 上的子任务

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

9.1.2.6.3 返回结果示例

```
stop-task test
```


```
{
    "op": "Stop",
    "result": true,
    "msg": "",
    "workers": [
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.15:8262",
                "msg": ""
            },
            "op": "Stop",
            "logID": "4"
        },
        {
            "meta": {
                "result": true,
                "worker": "172.16.30.16:8262",
                "msg": ""
            },
            "op": "Stop",
            "logID": "4"
        }
    ]
}
```


9.1.2.7 更新数据迁移任务

`update-task` 命令用于更新数据迁移任务。

支持的更新项包括：

- Table routing 规则
- Block & allow table lists 规则
- Binlog event filter 规则

其余项均不支持更新。

> **注意：**
> 如果能确保迁移任务所需的 relay log 在任务停止期间不会被清理，则推荐使用不支持更新项的更新步骤来以统一的方式更新任务配置信息。

9.1.2.7.1 支持更新项的更新步骤

1. 使用 `query-status <task-name>` 查询对应数据迁移任务的状态。
    - 若 `stage` 不为 `Paused`，则先使用 `pause-task <task-name | task-file>` 暂停任务。
2. 在 `task.yaml` 文件中更新需要修改的自定义配置或者错误配置。
3. 使用 `update-task task.yaml` 更新任务配置。
4. 使用 `resume-task <task-name | task-file>` 恢复任务。

9.1.2.7.2 不支持更新项的更新步骤

1. 使用 `query-status <task-name>` 查询对应数据迁移任务的状态。
    - 若任务存在，则通过 `stop-task <task-name | task-file>` 停止任务。
2. 在 `task.yaml` 文件中更新需要修改的自定义配置或者错误配置。
3. 使用 `start-task <config-file>` 重启恢复任务。

```
help update-task
```

```
update a task's config for routes, filters, or block-allow-list

Usage:
  dmctl update-task [-w worker ...] <config-file> [flags]

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

Global Flags:
  -w, --worker strings DM-worker ID
```

9.1.2.7.3 命令用法示例

```
update-task [-w "127.0.0.1:8262"] ./task.yaml
```

9.1.2.7.4 参数解释

- `-w`:
  - 可选
  - 指定在特定的一组 DM-workers 上更新数据迁移任务的子任务
  - 如果设置，则只更新指定 DM-workers 上的子任务配置
- `config-file`:
  - 必选
  - 指定 `task.yaml` 的文件路径

9.1.2.7.5 返回结果示例

```
update-task task.yaml
```

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "172.16.30.15:8262",
            "msg": ""
        },
        {
            "result": true,
            "worker": "172.16.30.16:8262",
            "msg": ""
        }
    ]
}
```

9.1.3 管理 DDL lock

目前与 DDL lock 相关的命令主要包括 `show-ddl-locks`、`unlock-ddl-lock`、`break-ddl-lock` 等。有关它们的功能、用法以及适用场景等，请参考手动处理 sharding DDL lock。

9.1.4 其他任务与集群管理命令

除上述常用的任务管理命令外，DM 还提供了其他一些命令用于管理数据迁移任务或 DM 集群本身。

9.1.4.1 检查任务配置文件

`check-task` 命令用于检查指定的数据迁移任务配置文件（`task.yaml`）是否合法以及上下游数据库的配置、权限、表结构等是否满足迁移需要。具体可参考上游 MySQL 实例配置前置检查。

在使用 `start-task` 启动迁移任务时，DM 也会执行 `check-task` 所做的全部检查。

```
help check-task
```

```
check the config file of the task

Usage:
 dmctl check-task <config-file> [flags]

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

Global Flags:
 -w, --worker strings DM-worker ID
```

9.1.4.1.1 命令用法示例

```
check-task task.yaml
```

9.1.4.1.2 参数解释

- `config-file`：
  - 必选
  - 指定 `task.yaml` 的文件路径

9.1.4.1.3 返回结果示例

```
check-task task-test.yaml
```

```
{
    "result": true,
    "msg": "check pass!!!"
}
```

9.1.4.2 暂停 relay 处理单元

relay 处理单元在 DM-worker 进程启动后即开始自动运行。通过使用 pause-relay 命令，我们可以暂停 relay 处理单元的运行。

当需要切换 DM-worker 通过虚拟 IP 连接的上游 MySQL 时，我们需要使用 pause-relay 对 DM 执行变更。具体变更步骤请参考虚拟 IP 环境下切换 DM-worker 与 MySQL 实例的连接。

```
help pause-relay
```

```
pause DM-worker's relay unit

Usage:
  dmctl pause-relay <-w worker ...> [flags]

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

Global Flags:
  -w, --worker strings DM-worker ID
```

9.1.4.2.1 命令用法示例

```
pause-relay -w "127.0.0.1:8262"
```

9.1.4.2.2 参数解释

- -w:
  - 必选
  - 指定需要暂停 relay 处理单元的 DM-worker

9.1.4.2.3 返回结果示例

```
pause-relay -w "172.16.30.15:8262"
```

```
{
    "op": "InvalidRelayOp",
    "result": true,
    "msg": "",
    "workers": [
        {
            "op": "PauseRelay",
            "result": true,
```

```
            "worker": "172.16.30.15:8262",
            "msg": ""
        }
    ]
}
```

9.1.4.3 恢复 relay 处理单元

resume-relay 用于恢复处于 Paused 状态的 relay 处理单元。

当需要切换 DM-worker 通过虚拟 IP 连接的上游 MySQL 时，我们需要使用 resume-relay 对 DM 执行变更。具体变更步骤请参考虚拟 IP 环境下切换 DM-worker 与 MySQL 实例的连接。

```
help resume-relay
```

```
resume DM-worker's relay unit

Usage:
  dmctl resume-relay <-w worker ...> [flags]

Flags:
  -h, --help help for resume-relay

Global Flags:
  -w, --worker strings DM-worker ID
```

9.1.4.3.1 命令用法示例

```
resume-relay -w "127.0.0.1:8262"
```

9.1.4.3.2 参数解释

- -w:
  - 必选
  - 指定需要恢复 relay 处理单元的 DM-worker

9.1.4.3.3 返回结果示例

```
resume-relay -w "172.16.30.15:8262"
```

```
{
    "op": "InvalidRelayOp",
    "result": true,
    "msg": "",
    "workers": [
        {
            "op": "ResumeRelay",
            "result": true,
            "worker": "172.16.30.15:8262",
            "msg": ""
        }
    ]
}
```

9.1.4.4 切换 relay log 到新的子目录

relay 处理单元通过使用不同的子目录来存储来自上游不同 MySQL 实例的 binlog 数据。通过使用 `switch-relay-master` 命令，我们可以变更 relay 处理单元以开始使用一个新的子目录。

当需要切换 DM-worker 通过虚拟 IP 连接的上游 MySQL 时，我们需要使用 `switch-relay-master` 对 DM 执行变更。具体变更步骤请参考虚拟 IP 环境下切换 DM-worker 与 MySQL 实例的连接。

```
help switch-relay-master
```

```
switch the master server of the DM-worker's relay unit

Usage:
  dmctl switch-relay-master <-w worker ...> [flags]

Flags:
  -h, --help help for switch-relay-master

Global Flags:
  -w, --worker strings DM-worker ID
```

9.1.4.4.1 命令用法示例

```
switch-relay-master -w "127.0.0.1:8262"
```

9.1.4.4.2 参数解释

- `-w`:

- 必选
- 指定需要切换 relay 处理单元使用子目录的 DM-worker

9.1.4.4.3 返回结果示例

```
switch-relay-master -w "172.16.30.15:8262"
```

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "172.16.30.15:8262",
            "msg": ""
        }
    ]
}
```

9.1.4.5 手动清理 relay log

DM 支持自动清理 relay log，但同时 DM 也支持使用 purge-relay 命令手动清理 relay log。

```
help purge-relay
```

```
purge relay log files of the DM-worker according to the specified filename

Usage:
  dmctl purge-relay <-w worker> [--filename] [--sub-dir] [flags]

Flags:
  -f, --filename string name of the terminal file before which to purge
      ↪ relay log files. Sample format: "mysql-bin.000006"
  -h, --help           help for purge-relay
  -s, --sub-dir string specify relay sub directory for --filename. If not
      ↪ specified, the latest one will be used. Sample format: "2ae76434-
      ↪ f79f-11e8-bde2-0242ac130008.000001"

Global Flags:
  -w, --worker strings DM-worker ID
```

9.1.4.5.1 命令用法示例

```
purge-relay -w "127.0.0.1:8262" --filename "mysql-bin.000003"
```

9.1.4.5.2 参数解释

- `-w`:
  - 必选
  - 指定需要执行 relay log 清理操作的 DM-worker
- `--filename`:
  - 必选
  - 指定标识 relay log 将要停止清理的文件名。如指定为 `mysql-bin.000100`，则只尝试清理到 `mysql-bin.000099`
- `--sub-dir`:
  - 可选
  - 指定 `--filename` 对应的 relay log 子目录，如果不指定则会使用当前最新的子目录

9.1.4.5.3 返回结果示例

```
purge-relay -w "127.0.0.1:8262" --filename "mysql-bin.000003"
```

```
[warn] no --sub-dir specified for --filename; the latest one will be used
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "127.0.0.1:8262",
            "msg": ""
        }
    ]
}
```

9.1.4.6 预设跳过 DDL 操作

`sql-skip` 命令用于预设一个跳过操作。当 binlog event 的 position 或 SQL 语句与指定的 `binlog-pos` 或 `sql-pattern` 匹配时，执行该跳过操作。相关参数与结果解释，请参考sql-skip。

9.1.4.7 预设替换 DDL 操作

`sql-replace` 命令用于预设一个替换执行操作。当 binlog event 的 position 或 SQL 语句与指定的 `binlog-pos` 或 `sql-pattern` 匹配时，执行该替换执行操作。相关参数与结果解释，请参考sql-replace。

9.1.4.8 强制刷新 task => DM-workers 映射关系

refresh-worker-tasks 命令用于强制刷新 DM-master 内存中维护的 task => DM-workers 映射关系。

> **注意：**
>
> 一般不需要使用此命令。仅当已确定 task => DM-workers 映射关系存在，但执行其它命令时仍提示必须刷新它时，你才需要使用此命令。

9.1.5 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
> - 任务操作只能放在 dmctl 命令的最后

```
./dmctl -master-addr 172.16.30.14:8261 start-task task.yaml
./dmctl -master-addr 172.16.30.14:8261 stop-task task
./dmctl -master-addr 172.16.30.14:8261 query-status
```

```
Available Commands:
  break-ddl-lock     break-ddl-lock <-w worker ...> <task-name> [--remove-id
     ↪ ] [--exec] [--skip]
  check-task         check-task <config-file>
  migrate-relay      migrate-relay <worker> <binlogName> <binlogPos>
  pause-relay        pause-relay <-w worker ...>
  pause-task         pause-task [-w worker ...] <task-name>
  purge-relay        purge-relay <-w worker> [--filename] [--sub-dir]
  query-error        query-error [-w worker ...] [task-name]
  query-status       query-status [-w worker ...] [task-name]
  refresh-worker-tasks refresh-worker-tasks
  resume-relay       resume-relay <-w worker ...>
  resume-task        resume-task [-w worker ...] <task-name>
  show-ddl-locks     show-ddl-locks [-w worker ...] [task-name]
  sql-inject         sql-inject <-w worker> <task-name> <sql1;sql2;>
```

```
sql-replace         sql-replace <-w worker> [-b binlog-pos] [-s sql-pattern
    ↪ ] [--sharding] <task-name> <sql1;sql2;>
sql-skip            sql-skip <-w worker> [-b binlog-pos] [-s sql-pattern]
    ↪ [--sharding] <task-name>
start-task          start-task [-w worker ...] <config-file>
stop-task           stop-task [-w worker ...] <task-name>
switch-relay-master switch-relay-master <-w worker ...>
unlock-ddl-lock     unlock-ddl-lock [-w worker ...] <lock-ID>
update-master-config update-master-config <config-file>
update-relay        update-relay [-w worker ...] <config-file>
update-task         update-task [-w worker ...] <config-file>
```

9.1.6 废弃或不推荐使用的命令

以下命令已经被废弃或仅用于 debug，在接下来的版本中可能会被移除或修改其语义，强烈不推荐使用。

- migrate-relay
- sql-inject
- update-master-config
- update-relay

9.2 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。

9.2.1 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。

9.2.2 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
  - 5.5 < MySQL 版本 < 8.0
  - MariaDB 版本 >= 10.1.2
- 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 复制）
  - 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

9.2.2.1 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

取值	含义
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
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

9.3 DM 查询状态

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

9.3.1 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-1",        # 任务名称
            "taskStatus": "Running",     # 任务运行状态，包括 “New”，“
                ↪ Running”，“Paused”，“Stopped”，“Finished” 以及 “
                ↪ Error”。
            "workers": [                 # 该任务所使用的 DM-workers 列表
                "127.0.0.1:8262"
            ]
        },
        {
            "taskName": "test-2",
            "taskStatus": "Error - Some error occurred in subtask", #
                ↪ 该任务的子任务存在运行错误并暂停的现象
            "workers": [
                "127.0.0.1:8262",
                "127.0.0.1:8263"
            ]
        },
        {
```

```
            "taskName": "test-3",
            "taskStatus": "Error - Relay status is Error", # 该任务的某个处于
                ↪ Sync 阶段的子任务对应的 Relay 处理单元出错
            "workers": [
                "127.0.0.1:8263",
                "127.0.0.1:8264"
            ]
        }
    ]
}
```

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。

9.3.2 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

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

任务对应的所有子任务的状态	任务状态
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

9.3.3 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。
    "workers": [                        # DM-worker 列表。
        {
            "result": true,
            "worker": "172.17.0.2:8262", # DM-worker ID。
            "msg": "",
            "subTaskStatus": [          # DM-worker 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": "",
        ↪                                   #
        ↪ 当前版本总是为空（因为 `Sync` 处理单元暂不使用
        ↪ GTID 迁移数据）。
    "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。
            }
        }
    ],
    "relayStatus": {  # relay 单元的迁移状态.
        "masterBinlog": "(bin.000001, 3234)",                       #
            ↪ 上游数据库的 binlog position。
        "masterBinlogGtid": "c0149e17-dff1-11e8-b6a8-0242ac110004
            ↪ :1-14", # 上游数据库的 binlog GTID 信息。
        "relaySubDir": "c0149e17-dff1-11e8-b6a8-0242ac110004.000001",
            ↪ # 当前使用的 relay log 子目录。
        "relayBinlog": "(bin.000001, 3234)",                        #
            ↪ 已被拉取至本地存储的 binlog position。
        "relayBinlogGtid": "c0149e17-dff1-11e8-b6a8-0242ac110004
            ↪ :1-14", # 已被拉取至本地存储的 binlog GTID 信息。
        "relayCatchUpMaster": true, # 本地 relay log
            ↪ 迁移进度是否与上游一致。
        "stage": "Running",          # relay 处理单元状态
        "result": null
    },
    "sourceID": "172.17.0.2:3306"    # 上游实例或者复制组 ID
},
{
    "result": true,
    "worker": "172.17.0.3:8262",
    "msg": "",
```

```
        "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。
                }
            }
        ],
        "relayStatus": {
            "masterBinlog": "(bin.000001, 28507)",
            "masterBinlogGtid": "c0149e17-dff1-11e8-b6a8-0242ac110004
                ↪ :1-96",
            "relaySubDir": "c0149e17-dff1-11e8-b6a8-0242ac110004.000001",
            "relayBinlog": "(bin.000001, 28507)",
            "relayBinlogGtid": "c0149e17-dff1-11e8-b6a8-0242ac110004
                ↪ :1-96",
            "relayCatchUpMaster": true,
            "stage": "Running",
            "result": null
        },
        "sourceID": "172.17.0.3:3306"
    },
    {
        "result": true,
        "worker": "172.17.0.6:8262",
        "msg": "",
        "subTaskStatus": [
            {
                "name": "test",
                "stage": "Paused",
                "unit": "Load",
                "result": {              # 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 %"
                    }
                }
            ],
            "relayStatus": {
                "masterBinlog": "(bin.000001, 1691)",
                "masterBinlogGtid": "97b5142f-e19c-11e8-808c-0242ac110005
                    ↪ :1-9",
                "relaySubDir": "97b5142f-e19c-11e8-808c-0242ac110005.000001",
                "relayBinlog": "(bin.000001, 1691)",
                "relayBinlogGtid": "97b5142f-e19c-11e8-808c-0242ac110005:1-9",
                "relayCatchUpMaster": true,
                "stage": "Running",
                "result": null
            },
            "sourceID": "172.17.0.6:3306"
        }
    ]
}
```

关于 workers 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。

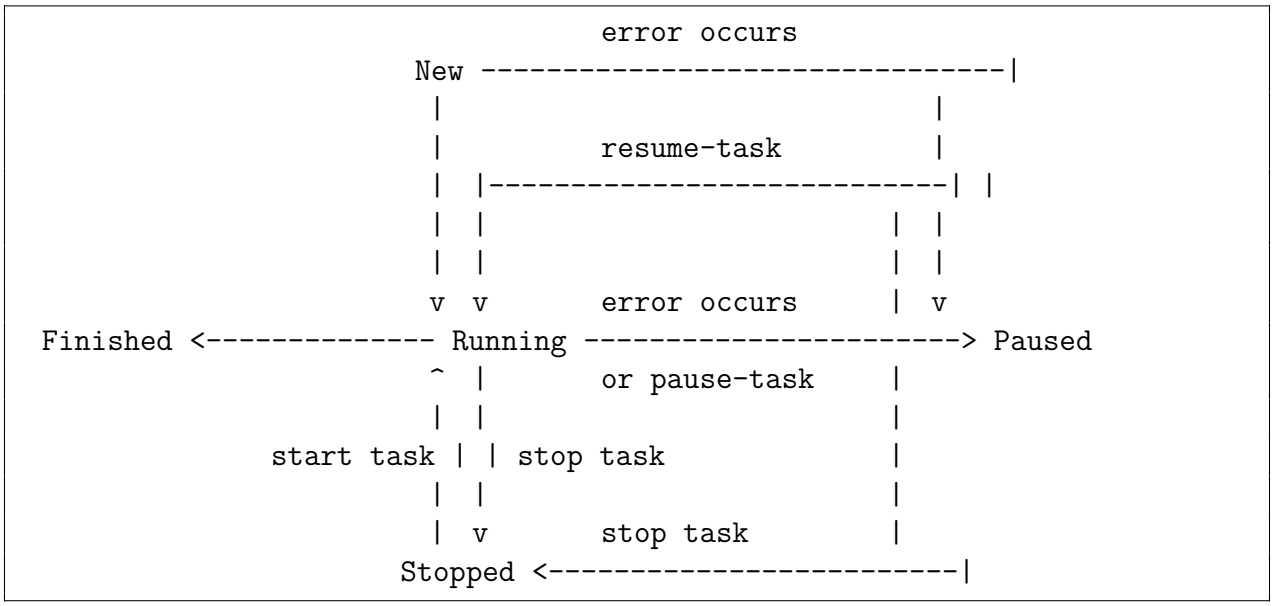
关于 workers 下 subTaskStatus 中 unresolvedDDLLockID的操作细节，请参阅手动处理 Sharding DDL Lock。

9.3.4 子任务状态

9.3.4.1 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。

9.3.4.2 状态转换图



9.4 跳过或替代执行异常的 SQL 语句

本文介绍了如何使用 DM 来处理异常的 SQL 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 SQL 语句，可能会造成错误并中断迁移任务。在这种情况下，DM 提供以下两种方式来恢复迁移：

- 使用 dmctl 来手动跳过 (skip) 该 SQL 语句对应的 binlog event。
- 使用 dmctl 来手动指定其他 SQL 语句来替代 (replace) 该 SQL 语句对应的 binlog event，并向下游执行。

如果提前预知将要迁移 TiDB 不支持的 SQL 语句，也可以使用 dmctl 来手动预设跳过/替代执行操作。当 DM 尝试将该 SQL 语句对应的 binlog event 迁移到下游时，该预设的操作将自动执行，从而避免迁移过程被中断。

9.4.0.0.1 使用限制

- 跳过/替代执行操作只适合用于一次性跳过/替代执行下游 **TiDB** 不支持执行的 **SQL** 语句，其它迁移错误请不要使用此方式进行处理。
  - 其它迁移错误可尝试使用Block & Allow Table Lists 或binlog event filter。
- 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。
- 单次跳过/替代执行操作都是针对单个 binlog event 的。
- `--sharding` 仅用于对 sharding group 预设一些操作，并且必须在 DDL 语句执行之前预设，不能在 DDL 语句已经执行后预设。
  - `--sharding` 模式下只支持预设，并只能使用 `--sql-pattern` 来匹配 binlog event。
  - 有关使用 DM 处理 sharding DDL 迁移的原理，请参阅分库分表合并迁移原理。

9.4.0.0.2 匹配 binlog event

当迁移任务由于执行 SQL 语句出错而中断时，可以使用 `query-error` 命令获取对应 binlog event 的 position 信息。在执行 `sql-skip` / `sql-replace` 时，通过指定该 position 信息，即可与对应的 binlog event 进行匹配。

然而，在迁移中断前主动处理不被支持的 SQL 语句的情况下，由于无法提前预知 binlog event 的 position 信息，则需要使用其他方式来确保与后续将到达的 binlog event 进行匹配。

在 DM 中，支持如下两种 binlog event 匹配模式（两种模式只能选择其中一种）：

1. binlog position：binlog event 的 position 信息

    - binlog position 在命令中使用 `--binlog-pos` 参数传入，格式为 `binlog-filename :binlog-pos`，如 `mysql-bin|000001.000003:3270`。
    - DM 中 binlog filename 的格式与上游 MySQL 中 binlog filename 的格式不完全一致。
    - 在迁移执行出错后，binlog position 可直接从 `query-error` 返回的 `failedBinlogPosition` 中获得。

2. DDL pattern：（仅限于 DDL 语句的）正则表达式匹配模式

    - DDL pattern 在命令中使用 `--sql-pattern` 参数传入，如要匹配 `ALTER TABLE `db2`.`tbl2` DROP COLUMN `c2``，则对应的正则表达式为 `~(?i)ALTER\s+TABLE\s+`db2`.`tbl2`\s+DROP\s+COLUMN\s+`c2``。
    - 正则表达式必须以 `~` 为前缀，且不包含任何原始空格（正则表达式字符串中的空格均以 `\s` 或 `\s+` 表示）。

对于合库合表场景，如果需要由 DM 自动选择 DDL lock owner 来执行跳过/替代执行操作，则由于不同 DM-worker 上 DDL 语句对应的 binlog position 无逻辑关联且难以确定，因此只能使用 DDL pattern 匹配模式。

> 注意：
>
> - 一个 binlog event 只能注册一个使用 `--binlog-pos` 指定的 operator，后注册的 operator 会覆盖之前已经注册的 operator。
> - 不要尝试为一个 binlog event 同时使用 `--binlog-pos` 和 `--sql-pattern` 指定 operator。
> - operator 在与 binlog event 匹配成功后（而非执行成功后）即会被删除，后续如果需要再进行（`--sql-pattern`）匹配则必须重新注册。

9.4.0.1 支持场景

- 场景一：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 `sql-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 `sql-replace` 替代该 DDL 的迁移以恢复迁移任务。
- 场景二：迁移过程中，预先知道了上游将执行 TiDB 不支持的 DDL 语句，则需要提前处理以避免迁移任务中断。
  - 如果业务能接受下游 TiDB 不执行该 DDL 语句，则使用 `sql-skip` 预设一个跳过该 DDL 语句的操作，当执行到该 DDL 语句时即自动跳过。
  - 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 `sql-replace` 预设一个替代该 DDL 语句的操作，当执行到该 DDL 语句时即自动替代。

9.4.0.2 实现原理

DM 在进行增量数据复制时，简化后的流程大致为：

1. relay 处理单元从上游拉取 binlog 存储在本地作为 relay log。
2. binlog 复制单元（sync）读取本地 relay log，获取其中的 binlog event。
3. binlog 复制单元解析该 binlog event 并构造 DDL/DML 语句，然后将这些语句向下游 TiDB 执行。

在 binlog 复制单元解析完 binlog event 并向下游 TiDB 执行时，可能会由于 TiDB 不支持对应的 SQL 语句而报错并造成迁移中断。

在 DM 中，可以为 binlog event 注册一些跳过/替代执行操作（operator）。在向下游 TiDB 执行 SQL 语句前，将当前的 binlog event 信息（position、DDL 语句）与注册的 operator 进行比较。如果 position 或 DDL 语句与注册的某个 operator 匹配，则执行该 operator 对应的操作并将该 operator 移除。

迁移中断后使用 `sql-skip` 或 `sql-replace` 恢复迁移的流程

1. 使用 `sql-skip` 或 `sql-replace` 为指定的 binlog position 或 DDL pattern 注册 operator。
2. 使用 `resume-task` 恢复之前由于迁移出错导致中断的任务。
3. 重新解析获得之前造成迁移出错的 binlog event。

4. 该 binlog event 与第一步注册的 operator 匹配成功。
5. 执行 operator 对应的操作（跳过/替代执行）后，继续执行迁移任务。

迁移中断前使用 `sql-skip` 或 `sql-replace` 预设操作以避免迁移中断的流程

1. 使用 `sql-skip` 或 `sql-replace` 为指定的 DDL pattern 注册 operator。
2. 从 relay log 中解析获得 binlog event。
3. （包含 TiDB 不支持 SQL 语句的）binlog event 与第一步注册的 operator 匹配成功。
4. 执行 operator 对应的操作（跳过/替代执行）后，继续执行迁移任务，任务不发生中断。

合库合表迁移中断前使用 `sql-skip` 或 `sql-replace` 预设操作以避免迁移中断的流程

1. 使用 `sql-skip` 或 `sql-replace`（在 DM-master 上）为指定的 DDL pattern 注册 operator。
2. 各 DM-worker 从 relay log 中解析获得 binlog event。
3. DM-master 协调各个 DM-worker 进行 DDL lock 同步。
4. DM-master 判断得知 DDL lock 迁移成功后，将第一步注册的 operator 发送给 DDL lock owner。
5. DM-master 请求 DDL lock owner 执行 DDL 语句。
6. DDL lock owner 将要执行的 DDL 语句与第四步收到的 operator 匹配成功。
7. 执行 operator 对应的操作（跳过/替代执行）后，继续执行迁移任务。

9.4.0.3 命令介绍

使用 dmctl 手动处理 TiDB 不支持的 SQL 语句时，主要使用的命令包括 `query-status`、`query-error`、`sql-skip` 和 `sql-replace`。

9.4.0.3.1 query-status

`query-status` 命令用于查询当前 DM-worker 内子任务及 relay 单元等的状态，详见查询状态。

9.4.0.3.2 query-error

query-error 命令用于查询 DM-worker 内子任务及 relay 单元当前在运行中存在的错误。

命令用法

```
query-error [--worker=127.0.0.1:8262] [task-name]
```

参数解释

- worker：
  - flag 参数，string，--worker，可选；
  - 不指定时查询所有 DM-worker 上的错误，指定时仅查询特定一组 DM-worker 上的错误。
- task-name：
  - 非 flag 参数，string，可选；
  - 不指定时查询所有任务内的错误，指定时仅查询特定任务内的错误。

结果示例

```
» query-error test
```

```
{
    "result": true,                             # query-error 操作本身是否成功
    "msg": "",                                  # query-error 操作失败的说明信息
    "workers": [                                # DM-worker 信息列表
        {
            "result": true,                     # 该 DM-worker 上 query-error
                ↪ 操作是否成功
            "worker": "127.0.0.1:8262",         # 该 DM-worker 的 IP:port(worker
                ↪ -id)
            "msg": "",                          # 该 DM-worker 上 query-error
                ↪ 操作失败的说明信息
            "subTaskError": [                   # 该 DM-worker
                ↪ 上运行子任务的错误信息
                {
                    "name": "test",             # 任务名
                    "stage": "Paused",          # 当前任务的状态
                    "unit": "Sync",             # 当前正在处理任务的处理单元
                    "sync": {                   # binlog 复制单元(sync)
                        ↪ 的错误信息
                        "errors": [             # 当前处理单元的错误信息列表
                            {
                                // 错误信息描述
```

```
                            "msg": "exec sqls[[USE `db1`; ALTER TABLE `db1
                                ↪ `.`tbl1` CHANGE COLUMN `c2` `c2` decimal
                                ↪ (10,3);]] failed, err:Error 1105:
                                ↪ unsupported modify column length 10 is
                                ↪ less than origin 11",
                            // 发生错误的 binlog event 的 position
                            "failedBinlogPosition": "mysql-bin
                                ↪ |000001.000003:34642",
                            // 发生错误的 SQL 语句
                            "errorSQL": "[USE `db1`; ALTER TABLE `db1`.`tbl1
                                ↪ ` CHANGE COLUMN `c2` `c2` decimal(10,3)
                                ↪ ;]"
                        }
                    ]
                }
            }
        ],
        "RelayError": {                 # 该 DM-worker 上 relay
            ↪ 处理单元的错误信息
            "msg": ""                   # 错误信息描述
        }
    }
  ]
}
```

9.4.0.3.3 sql-skip

`sql-skip` 命令用于预设一个跳过操作，当 binlog event 的 position 或 SQL 语句与指定的 `binlog-pos` 或 `sql-pattern` 匹配时，执行该跳过操作。

命令用法

```
sql-skip <--worker=127.0.0.1:8262> [--binlog-pos=mysql-bin
    ↪ |000001.000003:3270] [--sql-pattern=~(?i)ALTER\s+TABLE\s+`db1`.`tbl1
    ↪ `\s+ADD\s+COLUMN\s+col1\s+INT] [--sharding] <task-name>
```

参数解释

- `worker`:
  - flag 参数，string，`--worker`;
  - 未指定 `--sharding` 时必选，指定 `--sharding` 时禁止使用;
  - `worker` 指定预设操作将生效的 DM-worker。
- `binlog-pos`:
  - flag 参数，string，`--binlog-pos`;

  - binlog-pos 与 --sql-pattern 必须指定其中一个，且只能指定其中一个。
  - 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-error 返回的 failedBinlogPosition 中获得。

- sql-pattern:
  - flag 参数，string，--sql-pattern;
  - --sql-pattern 与 binlog-pos 必须指定其中一个，且只能指定其中一个。
  - 在指定时表示操作将在 sql-pattern 与 binlog event 对应的（经过可选的 router-rule 转换后的）DDL 语句匹配时生效。格式为以 ~ 为前缀的正则表达式，如 ~(?i)ALTER\s+TABLE\s+`db1`.`tbl1`\s+ADD\s+COLUMN\s+col1\s+INT。
    * 暂时不支持正则表达式中包含原始空格，需要使用 \s 或 \s+ 替代空格。
    * 正则表达式必须以 ~ 为前缀，详见正则表达式语法。
    * 正则表达式中的库名和表名必须是经过可选的 router-rule 转换后的名字，即对应下游的目标库名和表名。如上游为 `shard_db_1`.`shard_tbl_1`，下游为 `shard_db`.`shard_tbl`，则应该尝试匹配 `shard_db`.`shard_tbl`。
    * 正则表达式中的库名、表名及列名需要使用 ` 标记，如 `db1`.`tbl1`。

- sharding:
  - flag 参数，boolean，--sharding;
  - 未指定 --worker 时必选，指定 --worker 时禁止使用;
  - 在指定时表示预设的操作将在 sharding DDL 迁移过程中的 DDL lock owner 内生效。

- task-name:
  - 非 flag 参数，string，必选;
  - 指定预设的操作将生效的任务。

9.4.0.3.4 sql-replace

sql-replace 命令用于预设一个替代执行操作，当 binlog event 的 position 或 SQL 语句与指定的 binlog-pos 或 sql-pattern 匹配时，执行该替代执行操作。

命令用法

```
sql-replace <--worker=127.0.0.1:8262> [--binlog-pos=mysql-bin
    ↪ |000001.000003:3270] [--sql-pattern=~(?i)ALTER\s+TABLE\s+`db1`.`tbl1
    ↪ `\s+ADD\s+COLUMN\s+col1\s+INT] [--sharding] <task-name> <SQL-1;SQL-2>
```

参数解释

- worker:
  - 与 sql-skip 命令的 --worker 参数解释一致。

- binlog-pos:

  - 与 sql-skip 命令的 --binlog-pos 参数解释一致。

- sql-pattern：
  - 与 sql-skip 命令的 --sql-pattern 参数解释一致。
- sharding：
  - 与 sql-skip 命令的 --sharding 参数解释一致。
- task-name：
  - 与 sql-skip 命令的 task-name 参数解释一致。
- SQLs：
  - 非 flag 参数，string，必选；
  - SQLs 指定将用于替代原 binlog event 的新的 SQL 语句。多条 SQL 语句间以 ; 分隔，如 ALTER TABLE shard_db.shard_table drop index idx_c2;ALTER TABLE shard_db.shard_table DROP COLUMN c2;。

9.4.0.4 使用示例

9.4.0.4.1 迁移中断后被动执行跳过操作

应用场景

假设现在需要将上游的 db1.tbl1 表迁移到下游 TiDB（非合库合表迁移场景），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 迁移任务中断且报如下错误：

```
exec sqls[[USE `db1`; ALTER TABLE `db1`.`tbl1` CHANGE COLUMN `c2` `c2`
    ↪ decimal(10,3);]] failed,
err:Error 1105: unsupported modify column length 10 is less than origin 11
```

此时使用 query-status 查询任务状态，可看到 stage 转为了 Paused，且 errors 中有相关的错误描述信息。

使用 query-error 可以获取该错误的详细信息。比如，执行 query-error test ↪ 可获得出错的 binlog event 的 position（failedBinlogPosition）为 mysql-bin ↪ |000001.000003:34642。

被动跳过 SQL 语句

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 sql-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 query-error 获取迁移出错的 binlog event 的 position 信息。

    - position 信息可直接由 query-error 返回的 failedBinlogPosition 获得。
    - 本示例中的 position 为 mysql-bin|000001.000003:34642。

2. 使用 sql-skip 预设一个 binlog event 跳过操作，该操作将在使用 resume-task 后迁移该 binlog event 到下游时生效。

```
» sql-skip --worker=127.0.0.1:8262 --binlog-pos=mysql-bin
    ↪ |000001.000003:34642 test
```

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

对应 DM-worker 节点中也可以看到类似如下日志：

```
2018/12/28 11:17:51 operator.go:121: [info] [sql-operator] set a new
    ↪ operator
uuid: 6bfcf30f-2841-4d70-9a34-28d7082bdbd7, pos: (mysql-bin
    ↪ |000001.000003, 34642), op: SKIP, args:
on replication unit
```

3. 使用 resume-task 恢复之前出错中断的迁移任务。

```
» resume-task --worker=127.0.0.1:8262 test
```

```
{
    "op": "Resume",
    "result": true,
    "msg": "",
    "workers": [
        {
            "op": "Resume",
            "result": true,
            "worker": "127.0.0.1:8262",
            "msg": ""
        }
    ]
}
```

对应 DM-worker 节点中也可以看到类似如下日志：

```
2018/12/28 11:27:46 operator.go:158: [info] [sql-operator] binlog-pos (
    ↪ mysql-bin|000001.000003, 34642) matched,
applying operator uuid: 6bfcf30f-2841-4d70-9a34-28d7082bdbd7, pos: (
    ↪ mysql-bin|000001.000003, 34642), op: SKIP, args:
```

4. 使用 query-status 确认该任务的 stage 已经转为 Running。

5. 使用 query-error 确认原错误信息已不再存在。

9.4.0.4.2 迁移中断前主动执行替代执行操作

应用场景

假设现在需要将上游的 db2.tbl2 表迁移到下游 TiDB（非合库合表迁移场景），初始时表结构为：

```
SHOW CREATE TABLE db2.tbl2;
```

```
+-------+-------------------------------------------------+
| Table | Create Table                                    |
+-------+-------------------------------------------------+
| tbl2 | CREATE TABLE `tbl2` (
  `c1` int(11) NOT NULL,
  `c2` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`),
  KEY `idx_c2` (`c2`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-------+-------------------------------------------------+
```

此时，上游执行以下 DDL 操作修改表结构（即 DROP 列 c2）：

```
ALTER TABLE db2.tbl2 DROP COLUMN c2;
```

当迁移该 DDL 语句对应的 binlog event 到下游时，会由于 TiDB 不支持该 DDL 语句而导致 DM 迁移任务中断且报如下错误：

```
exec sqls[[USE `db2`; ALTER TABLE `db2`.`tbl2` DROP COLUMN `c2`;]] failed,
err:Error 1105: can't drop column c2 with index covered now
```

但如果在上游实际执行该 DDL 语句前，你已提前知道该 DDL 语句不被 TiDB 所支持。则可以使用 `sql-skip` 或 `sql-replace` 为该 DDL 语句预设一个跳过（skip）或替代执行（replace）操作。

对于本示例中的 DDL 语句，由于 TiDB 暂时不支持 DROP 存在索引的列，因此可以使用两条新的 SQL 语句来进行替代执行操作，即可以先 DROP 索引、然后再 DROP 列 c2。

主动替代执行该 SQL 语句

1. 为将要在上游执行的 DDL 语句（经过可选的 router-rule 转换后的 DDL 语句）设计一个能匹配上的正则表达式。

   - 上游将执行的 DDL 语句为 `ALTER TABLE db2.tbl2 DROP COLUMN c2;`。
   - 由于该 DDL 语句不存在 router-rule 转换，可设计如下正则表达式：

     ```
     ~(?i)ALTER\s+TABLE\s+`db2`.`tbl2`\s+DROP\s+COLUMN\s+`c2`
     ```

2. 为该 DDL 语句设计将用于替代执行的新的 DDL 语句：

   ```
   ALTER TABLE `db2`.`tbl2` DROP INDEX idx_c2;ALTER TABLE `db2`.`tbl2`
       ↪ DROP COLUMN `c2`;
   ```

3. 使用 `sql-replace` 预设一个 binlog event 替代执行操作，该操作将在迁移该 binlog event 到下游时生效。

   ```
   » sql-replace --worker=127.0.0.1:8262 --sql-pattern=~(?i)ALTER\s+TABLE\
       ↪ s+`db2`.`tbl2`\s+DROP\s+COLUMN\s+`c2` test ALTER TABLE `db2`.`
       ↪ tbl2` DROP INDEX idx_c2;ALTER TABLE `db2`.`tbl2` DROP COLUMN `c2`
   ```

   ```
   {
       "result": true,
       "msg": "",
       "workers": [
           {
               "result": true,
               "worker": "",
   ```

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

对应 DM-worker 节点中也可以看到类似如下日志：

```
2018/12/28 15:33:13 operator.go:121: [info] [sql-operator] set a new
    ↪ operator
uuid: c699a18a-8e75-47eb-8e7e-0e5abde2053c, pattern: ~(?i)ALTER\s+TABLE
    ↪ \s+`db2`.`tbl2`\s+DROP\s+COLUMN\s+`c2`,
op: REPLACE, args: ALTER TABLE `db2`.`tbl2` DROP INDEX idx_c2; ALTER
    ↪ TABLE `db2`.`tbl2` DROP COLUMN `c2`
on replication unit
```

4. 在上游 MySQL 执行该 DDL 语句。
5. 观察下游表结构是否变更成功，对应 DM-worker 节点中也可以看到类似如下日志：

```
2018/12/28 15:33:45 operator.go:158: [info] [sql-operator]
sql-pattern ~(?i)ALTER\s+TABLE\s+`db2`.`tbl2`\s+DROP\s+COLUMN\s+`c2`
    ↪ matched SQL
USE `db2`; ALTER TABLE `db2`.`tbl2` DROP COLUMN `c2`;,
applying operator uuid: c699a18a-8e75-47eb-8e7e-0e5abde2053c,
pattern: ~(?i)ALTER\s+TABLE\s+`db2`.`tbl2`\s+DROP\s+COLUMN\s+`c2`,
op: REPLACE, args: ALTER TABLE `db2`.`tbl2` DROP INDEX idx_c2; ALTER
    ↪ TABLE `db2`.`tbl2` DROP COLUMN `c2`
```

6. 使用 `query-status` 确认该任务的 `stage` 持续为 `Running`。
7. 使用 `query-error` 确认不存在 DDL 执行错误。

9.4.0.4.3 合库合表场景下迁移中断后被动执行跳过操作

应用场景

假设现在通过多个 DM-worker 将上游多个 MySQL 实例内的多个表进行合库合表迁移到下游 TiDB 的同一个表，并且上游各分表执行了 TiDB 不支持的 DDL 语句。

DM-master 通过 DDL lock 协调 DDL 迁移、并请求 DDL lock owner 向下游 TiDB 执行该 DDL 语句后，由于 TiDB 不支持该 DDL 语句，迁移任务会报错并中断。

被动跳过 SQL 语句

合库合表场景下，被动跳过 TiDB 不支持的 DDL 语句的处理方式与非合库合表场景下的迁移中断后被动执行跳过操作基本一致。

但在合库合表场景下，只需要 DDL lock owner 向下游迁移该 DDL 语句，因此在两种场景下的处理过程主要存在以下区别：

1. 合库合表场景下，仅需要对 DDL lock owner 执行 `sql-skip`（`--worker={DDL-lock-owner}`）。
2. 合库合表场景下，仅需要对 DDL lock owner 执行 `resume-task`（`--worker={DDL-lock-owner}`）。

9.4.0.4.4 合库合表场景下迁移中断前主动执行替代执行操作

应用场景

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` ``：

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

初始时表结构为：

```
SHOW CREATE TABLE shard_db_1.shard_table_1;
```

```
+--------------+-----------------------------------------+
| Table        | Create Table                            |
+--------------+-----------------------------------------+
| shard_table_1 | CREATE TABLE `shard_table_1` (
  `c1` int(11) NOT NULL,
  `c2` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`),
  KEY `idx_c2` (`c2`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+--------------+-----------------------------------------+
```

此时，在上游所有分表上都执行以下 DDL 操作修改表结构（即 DROP 列 c2）：

```
ALTER TABLE shard_db_*.shard_table_* DROP COLUMN c2;
```

则当 DM 通过 sharding DDL lock 协调两个 DM-worker 迁移该 DDL 语句、并请求 DDL lock owner 向下游执行该 DDL 语句时，会由于 TiDB 不支持该 DDL 语句而导致迁移任务中断且报如下错误：

```
exec sqls[[USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` DROP COLUMN
    ↪ `c2`;]] failed,
err:Error 1105: can't drop column c2 with index covered now
```

但如果在上游实际执行该 **DDL** 语句前，你已提前知道该 **DDL** 语句不被 **TiDB** 所支持。则可以使用 `sql-skip` 或 `sql-replace` 命令为该 DDL 语句预设一个跳过/替代执行操作。

对于本示例中的 DDL 语句，由于 TiDB 暂时不支持 DROP 存在索引的列，因此可以使用两条新的 SQL 语句来进行替代执行操作，即可以先 DROP 索引、然后再 DROP c2 列。

主动替代执行该 SQL 语句

1. 为将要在上游执行的（经过可选的 router-rule 转换后的）DDL 语句设计一个能匹配上的正则表达式。

   - 上游将执行的 DDL 语句为 `ALTER TABLE shard_db_*.shard_table_* DROP COLUMN c2`。
   - 由于存在 router-rule 会将表名转换为 `` `shard_db`.`shard_table` ``，可设计如下正则表达式：

   ```
   ~(?i)ALTER\s+TABLE\s+`shard_db`.`shard_table`\s+DROP\s+COLUMN\s+`c2`
   ```

2. 为该 DDL 语句设计将用于替代执行的新的 DDL 语句：

   ```
   ALTER TABLE `shard_db`.`shard_table` DROP INDEX idx_c2;ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`;
   ```

3. 由于这是合库合表场景，因此使用 `--sharding` 参数来由 DM 自动确定替代执行操作只发生在 DDL lock owner 上。

4. 使用 `sql-replace` 预设一个 binlog event 替代执行操作，该操作将在迁移该 binlog event 到下游时生效。

   ```
   » sql-replace --sharding --sql-pattern=~(?i)ALTER\s+TABLE\s+`shard_db`.`shard_table`\s+DROP\s+COLUMN\s+`c2` test ALTER TABLE `shard_db`.`shard_table` DROP INDEX idx_c2;ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`
   ```

   ```
   {
       "result": true,
       "msg": "request with --sharding saved and will be sent to DDL lock's owner when resolving DDL lock",
       "workers": [
       ]
   }
   ```

   DM-master 节点中也可以看到类似如下日志：

```
2018/12/28 16:53:33 operator.go:105: [info] [sql-operator] set a new
    ↪ operator
uuid: eba35acd-6c5e-4bc3-b0b0-ae8bd1232351, request: name:"test"
op:REPLACE args:"ALTER TABLE `shard_db`.`shard_table` DROP INDEX idx_c2
    ↪ ;"
args:"ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`"
sqlPattern:"~(?i)ALTER\\s+TABLE\\s+`shard_db`.`shard_table`\\s+DROP\\s+
    ↪ COLUMN\\s+`c2`"
sharding:true
```

5. 在上游 MySQL 实例的分表上执行 DDL 语句。

6. 观察下游表结构是否变更成功，对应的 DDL lock **owner** 节点中也可以看到类似如下日志：

```
2018/12/28 16:54:35 operator.go:121: [info] [sql-operator] set a new
    ↪ operator
uuid: c959f2fb-f1c2-40c7-a1fa-e73cd51736dd,
pattern: ~(?i)ALTER\s+TABLE\s+`shard_db`.`shard_table`\s+DROP\s+COLUMN\
    ↪ s+`c2`,
op: REPLACE, args: ALTER TABLE `shard_db`.`shard_table` DROP INDEX
    ↪ idx_c2; ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`
on replication unit
```

```
2018/12/28 16:54:35 operator.go:158: [info] [sql-operator]
sql-pattern ~(?i)ALTER\s+TABLE\s+`shard_db`.`shard_table`\s+DROP\s+
    ↪ COLUMN\s+`c2` matched SQL
USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`;,
applying operator uuid: c959f2fb-f1c2-40c7-a1fa-e73cd51736dd,
pattern: ~(?i)ALTER\s+TABLE\s+`shard_db`.`shard_table`\s+DROP\s+COLUMN\
    ↪ s+`c2`,
op: REPLACE, args: ALTER TABLE `shard_db`.`shard_table` DROP INDEX
    ↪ idx_c2; ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`
```

另外，**DM-master** 节点中也可以看到类似如下日志：

```
2018/12/28 16:54:35 operator.go:122: [info] [sql-operator] get an
    ↪ operator
uuid: eba35acd-6c5e-4bc3-b0b0-ae8bd1232351, request: name:"test" op:
    ↪ REPLACE
args:"ALTER TABLE `shard_db`.`shard_table` DROP INDEX idx_c2;"
args:"ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`"
sqlPattern:"~(?i)ALTER\\s+TABLE\\s+`shard_db`.`shard_table`\\s+DROP\\s+
    ↪ COLUMN\\s+`c2`"
sharding:true
```

```
with key ~(?i)ALTER\s+TABLE\s+`shard_db`.`shard_table`\s+DROP\s+COLUMN\
    ↪ s+`c2` matched SQL
USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`;
```

```
2018/12/28 16:54:36 operator.go:145: [info] [sql-operator] remove an
    ↪ operator
uuid: eba35acd-6c5e-4bc3-b0b0-ae8bd1232351, request: name:"test" op:
    ↪ REPLACE
args:"ALTER TABLE `shard_db`.`shard_table` DROP INDEX idx_c2;"
args:"ALTER TABLE `shard_db`.`shard_table` DROP COLUMN `c2`"
sqlPattern:"~(?i)ALTER\\s+TABLE\\s+`shard_db`.`shard_table`\\s+DROP\\s+
    ↪ COLUMN\\s+`c2`"
sharding:true
```

7. 使用 `query-status` 确认任务的 `stage` 持续为 `Running`，且不存在正阻塞迁移的 DDL 语句（`blockingDDLs`）与待解决的 sharding group（`unresolvedGroups`）。
8. 使用 `query-error` 确认不存在 DDL 执行错误。
9. 使用 `show-ddl-locks` 确认不存在待解决的 DDL lock。

10 DM 监控指标

使用 DM-Ansible 部署 DM 集群的时候，会默认部署一套监控系统。

> **注意：**
> 目前只有 DM-worker 提供了 metrics，DM-master 暂未提供。

10.1 Task

在 Grafana dashboard 中，DM 默认名称为 `DM-task`。

10.1.1 overview

overview 下包含运行当前选定 task 的所有 DM-worker instance 的部分监控指标。当前默认告警规则只针对于单个 DM-worker instance。

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
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

10.1.2 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

10.1.3 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 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

10.1.4 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

metric 名称	说明
statement execution latency	load unit 执行语句的耗时，单位：秒

10.1.5 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 master 完全同步，单位：分钟	N/A	N/A
replicate lag	master 到 Syncer 的 binlog 复制延迟时间，单位：秒	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

10.2 Instance

在 Grafana dashboard 中，instance 的默认名称为 `DM-instance`。

10.2.1 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

10.2.2 task

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

11 从与 MySQL 兼容的数据库迁移数据

11.1 从 MySQL 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 协议数据库迁移数据到 TiDB。

11.1.1 第 1 步：在 Aurora 集群中启用 binlog

假设有两个 Aurora 集群需要迁移数据到 TiDB，其集群信息如下，其中 Aurora-1 包含一个独立的读取器节点。

集群	终端节点	端口	角色
Aurora-1	pingcap-1.h8emfqdptyc4.us-east-2.rds.amazonaws.com	3306	写入器
Aurora-1	pingcap-1-us-east-2a.h8emfqdptyc4.us-east-2.rds.amazonaws.com	3306	读取器
Aurora-2	pingcap-2.h8emfqdptyc4.us-east-2.rds.amazonaws.com	3306	写入器

DM 在增量复制阶段依赖 ROW 格式的 binlog，如果未启用 binlog 及设置正确的 binlog 格式，则不能正常使用 DM 进行数据迁移，具体可参见检查内容。

> **注意：**
>
> Aurora 读取器不能开启 binlog，因此不能作为 DM 数据迁移时的上游 master server。

如果需要基于 GTID 进行数据迁移，还需要为 Aurora 集群启用 GTID 支持。

> **注意：**
>
> 基于 GTID 的数据迁移需要 MySQL 5.7 (Aurora 2.04.1) 或更高版本。

11.1.1.1 为 Aurora 集群修改 binlog 相关参数

在 Aurora 集群中，binlog 相关参数是集群参数组中的集群级参数，有关如何为 Aurora 集群启用 binlog 支持，请参考在复制主实例上启用二进制日志记录。在使用 DM 进行数据迁移时，需要将 `binlog_format` 参数设置为 `ROW`。

如果需要基于 GTID 进行数据迁移，需要将 `gtid-mode` 与 `enforce_gtid_consistency` 均设置为 `ON`。有关如何为 Aurora 集群启用基于 GTID 的数据迁移支持，请参考 Configuring GTID-Based Replication for an Aurora MySQL Cluster。

> **注意：**
>
> 在 Aurora 管理后台中，`gtid_mode` 参数表示为 `gtid-mode`。

11.1.2 第 2 步：部署 DM 集群

目前推荐使用 DM-Ansible 部署 DM 集群，具体部署方法参照使用 DM-Ansible 部署 DM 集群。

注意：

- 在 DM 所有的配置文件中，数据库的密码要使用 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 加密上游 MySQL 用户密码。
- 上下游数据库用户必须拥有相应的读写权限。

11.1.3 第 3 步：检查集群信息

使用 DM-Ansible 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 Aurora-1	pingcap-1.h8emfqdptyc4.us-east-2.rds.amazonaws.com	3306	root	VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=
上游 Aurora-2	pingcap-2.h8emfqdptyc4.us-east-2.rds.amazonaws.com	3306	root	VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=
下游 TiDB	172.16.4.83	4000	root	

- dm-master 进程配置文件 `{ansible deploy}/conf/dm-master.toml` 中的配置

```
# Master 配置。

[[deploy]]
```

```
source-id = "mysql-replica-01"
dm-worker = "172.16.10.72:8262"

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

11.1.4 第 4 步：配置任务

假设需要将 Aurora-1 和 Aurora-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。

复制并编辑 {ansible deploy}/conf/task.yaml.example，生成如下任务配置文件 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: "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 <= v1.0.6 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 <= v1.0.6 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 <= v1.0.6 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: "-B test_db -T test_table" # dump 处理单元的其他参数，从 DM
       ↪ 1.0.2 版本开始，DM 会自动生成 table-list 配置，
       ↪ 在其之前的版本仍然需要人工配置。
```

11.1.5 第 5 步：启动任务

1. 进入 dmctl 目录 /home/tidb/dm-ansible/resources/bin/
2. 执行以下命令启动 dmctl

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

3. 执行以下命令启动数据迁移任务（task.yaml 是之前编辑的配置文件）

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

- 如果执行命令后的返回结果中不包含错误信息，则表明任务已经成功启动
- 如果包含以下错误信息，则表明上游 Aurora 用户可能拥有 TiDB 不支持的权限类型

```
{
    "id": 4,
    "name": "source db dump privilege chcker",
    "desc": "check dump privileges of source DB",
    "state": "fail",
    "errorMsg": "line 1 column 285 near \"LOAD FROM S3, SELECT INTO
       ↪  S3 ON *.* TO 'root'@'%' WITH GRANT OPTION\" ...",
    "instruction": "",
    "extra": "address of db instance - pingcap-1.h8emfqdptyc4.us-
       ↪ east-2.rds.amazonaws.com"
},
{
    "id": 5,
    "name": "source db replication privilege chcker",
```

```
    "desc": "check replication privileges of source DB",
    "state": "fail",
    "errorMsg": "line 1 column 285 near \"LOAD FROM S3, SELECT INTO
       ↪  S3 ON *.* TO 'root'@'%' WITH GRANT OPTION\" ...",
    "instruction": "",
    "extra": "address of db instance - pingcap-1.h8emfqdptyc4.us-
       ↪ east-2.rds.amazonaws.com"
}
```

此时可以选择以下两种处理方法中的任意一种进行处理后，再使用 `start-task` 尝试重新启动任务：

1. 为用于进行数据迁移的 Aurora 用户移除不被 TiDB 支持的不必要的权限
2. 如果能确保 Aurora 用户拥有 DM 所需要的权限，可以在 `task.yaml` 配置文件中添加如下顶级配置项以跳过启用任务时的前置权限检查

```
ignore-checking-items: ["dump_privilege", "
   ↪ replication_privilege"]
```

11.1.6 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，可在 dmctl 内使用以下命令进行查询：

```
query-status
```

注意：

如果查询命令的返回结果中包含以下错误信息，则表明在全量迁移的 dump 阶段不能获得相应的 lock：

```
Couldn't acquire global lock, snapshots will not be consistent:
   ↪  Access denied for user 'root'@'%' (using password: YES)
```

此时如果能接受不使用 FTWL 来确保 dump 文件与 metadata 的一致或上游能暂时停止写入，可以通过为 `mydumpers` 下的 `extra-args` 添加 `--no-locks` 参数来进行绕过，具体方法为：

1. 使用 `stop-task` 停止当前由于不能正常 dump 而已经转为 paused 的任务
2. 将原 `task.yaml` 中的 `extra-args: "-B test_db -T test_table"` 更新为 `extra-args: "-B test_db -T test_table --no-locks"`
3. 使用 `start-task` 重新启动任务

12 DM Portal 简介

当前版本的 DM 提供了丰富多样的功能特性，包括Table Routing，Block & Allow Lists，Binlog Event Filter 等。但这些功能特性同时也增加了用户使用 DM 的复杂度，尤其在编写DM 任务配置的时候。

针对这个问题，DM 提供了一个精简的网页程序 DM Portal，能够帮助用户以可视化的方式去配置需要的迁移任务，并且生成可以直接让 DM 直接执行的 `task.yaml` 文件。

12.1 功能描述

本节简要介绍 DM Portal 的各项功能。

12.1.1 迁移模式配置

支持 DM 的三种迁移模式：

- 全量迁移
- 增量复制
- All (全量 + 增量)

12.1.2 实例信息配置

支持配置库表迁移路由方式，能够支持 DM 中分库分表合并的配置方式。

12.1.3 binlog 过滤配置

支持对数据库、数据表配置 binlog event 过滤。

12.1.4 配置文件生成

支持配置文件创建，能够将配置文件下载到本地并且同时会在 dm-portal 服务器的 `/tmp/` 目录下自动创建。

12.1.5 使用限制

当前的 DM 配置可视化生成页面能够覆盖绝大部分的 DM 配置场景，但也有一定的使用限制：

- 不支持Binlog event filter 的 SQL pattern 方式
- 编辑功能不支持解析用户之前写的 `task.yaml` 文件，页面只能编辑由页面生成的 `task.yaml` 文件

- 编辑功能不支持修改实例配置信息，如果用户需要调整实例配置，需要重新生成 `task.yaml` 文件
- 页面的上游实例配置仅用于获取上游库表结构，DM-worker 里依旧需要配置对应的上游实例信息
- 生成的 `task.yaml` 中，默认 mydumper-path 为 `./bin/mydumper`，如果实际使用其他路径，需要在生成的配置文件中进行手动修改。

12.2 部署使用

本节介绍两种部署方式：Binary 部署和 DM Ansible 部署。

12.2.1 Binary 部署

DM Portal 可以在 dm-portal-latest-linux-amd64.tar.gz 下载，通过 `./dm-portal` 的命令即可直接启动。

- 如果在本地启动，浏览器访问 `127.0.0.1:8280` 即可使用。
- 如果在服务器上启动，需要为服务器上配置访问代理。

12.2.2 DM Ansible 部署

可以使用 DM Ansible 部署 DM Portal，具体部署方法参照使用 DM Ansible 部署 DM 集群。

12.3 使用说明

本节介绍如何使用 DM Portal 各个功能。

12.3.1 新建规则

12.3.1.1 功能描述

新建一个 `task.yaml` 文件。

12.3.1.2 操作步骤

登录 dm-portal 页面，点击新建任务规则。

12.3.2 基础信息配置

12.3.2.1 功能描述

用于填写任务名称，以及选择任务类型。

12.3.2.2 前置条件

已选择新建迁移规则。

12.3.2.3 操作步骤

1. 填写任务名称。
2. 选择任务类型。

Figure 6: DM Portal BasicConfig

12.3.3 实例信息配置

12.3.3.1 功能描述

用于配置上下游实例信息，包括 Host、Port、Username、Password。

12.3.3.2 前置条件

已填写任务名称和选择任务类型。

12.3.3.3 注意事项

如果任务类型选择增量或者 All，在配置上游实例信息时候，还需要配置 binlog-file 和 binlog-pos。

12.3.3.4 操作步骤

1. 填写上游实例信息。
2. 填写下游实例信息。
3. 点击下一步。

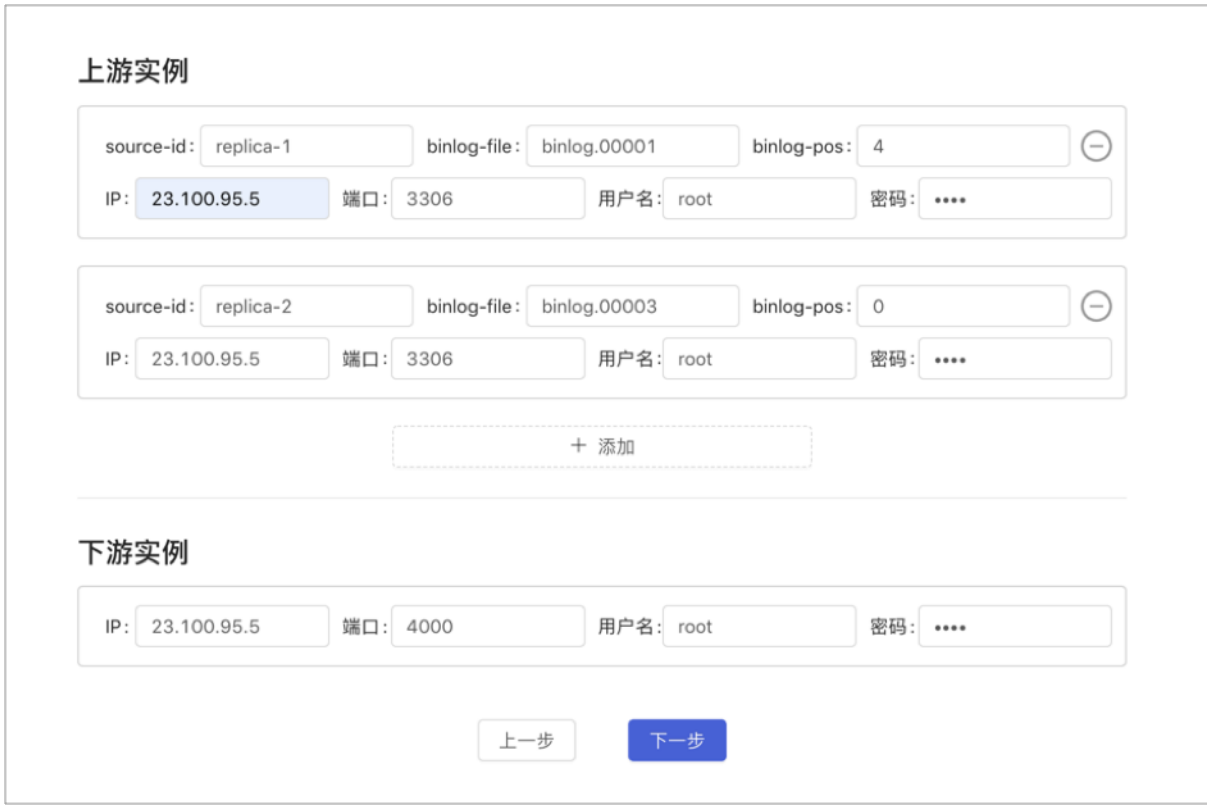


Figure 7: DM Portal InstanceConfig

12.3.4 binlog 过滤配置

12.3.4.1 功能描述

用于配置上游的 binlog 过滤，可以选择需要过滤的 DDL/DML，并且在数据库上配置的 filter 后会自动给其下的数据表继承。

12.3.4.2 前置条件

已经配置好上下游实例信息并且连接验证没问题。

12.3.4.3 注意事项

- binlog 过滤配置只能在上游实例处进行修改编辑，一旦数据库或者数据表被移动到下游实例后，就不可以进行修改编辑。
- 在数据库上配置的 binlog 过滤会自动被其下的数据表继承。

12.3.4.4 操作步骤

1. 点击需要配置的数据库或者数据表。
2. 点击编辑按钮，选择需要过滤的 binlog 类型。

Figure 8: DM Portal InstanceShow

Figure 9: DM Portal BinlogFilter 1

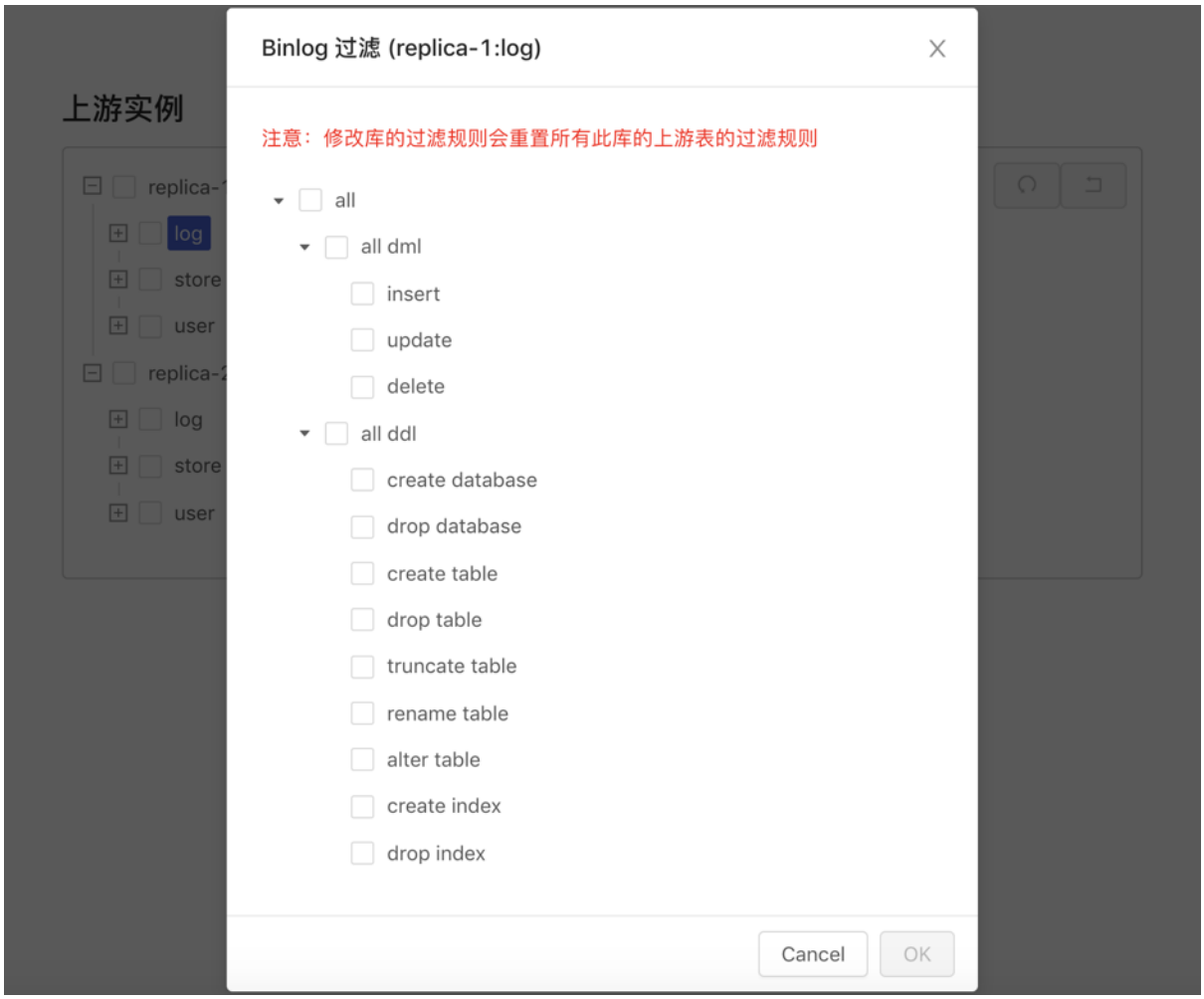


Figure 10: DM Portal BinlogFilter 2

12.3.5 库表路由配置

12.3.5.1 功能描述

可以选择需要迁移的数据库和数据表，并且进行修改名称、合并库、合并表等操作。可以对上一步操作进行撤销，可以对库表路由配置进行全部重置。在完成任务配置后，DM Portal 会帮忙生成对应的 `task.yaml` 文件。

12.3.5.2 前置条件

- 已经配置好需要的 binlog 过滤规则。

12.3.5.3 注意事项

- 在合并库表操作的时候，不允许批量操作，只能逐一拖动。

- 在合表库表操作的时候，只能对数据表进行拖动操作，不能对数据库进行数据库进行拖动操作。

12.3.5.4 操作步骤

1. 在上游实例处，选择需要迁移的数据库和数据表。
2. 点击移动按钮，将需要迁移的库表移动至下游实例处。
3. 点击右键按钮，可以对库表进行改名操作。
4. 选中需要操作的数据表，可以拖动至别的数据表图标上创建出合并表；可以拖动到数据库图标上移动至该库下；可以拖动到 target-instance 图标上移动到一个新的数据库下。
5. 点击完成，自动下载 `task.yaml` 到本地，并且在 DM Portal 服务器上的 `/tmp/` 目录下自动创建一份 `task.yaml` 配置文件。

12.3.5.4.1 移动迁移库表

Figure 11: DM Portal TableRoute 1

Figure 12: DM Portal TableRoute 2

12.3.5.4.2 右键修改库表名称

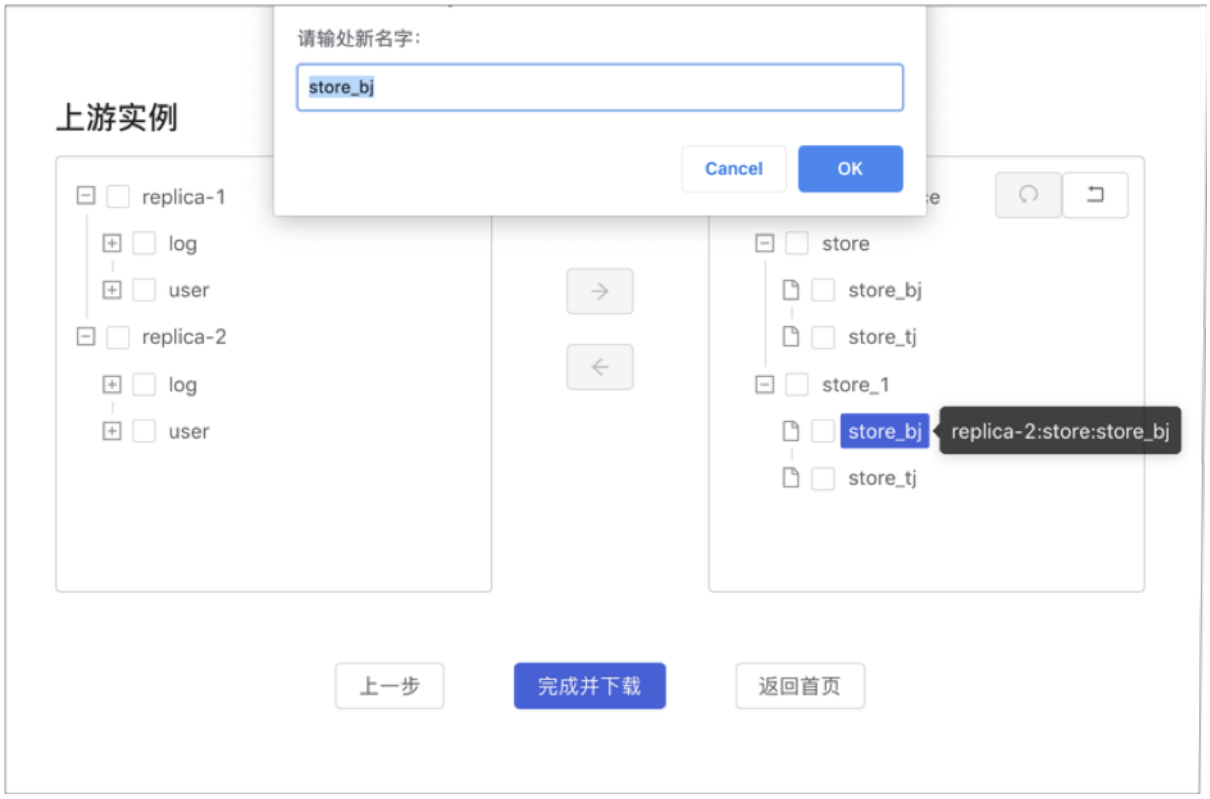


Figure 13: DM Portal ChangeTableName

12.3.5.4.3 合并数据表操作

Figure 14: DM Portal MergeTable 1

Figure 15: DM Portal MergeTable 2

12.3.5.4.4 移动数据表至其他数据库

Figure 16: DM Portal MoveToDB 1

Figure 17: DM Portal MoveToDB 2

12.3.5.4.5 移动数据表至新建默认数据库

Figure 18: DM Portal MoveToNewDB 1

Figure 19: DM Portal MoveToNewDB 2

12.3.5.4.6 撤销本次操作

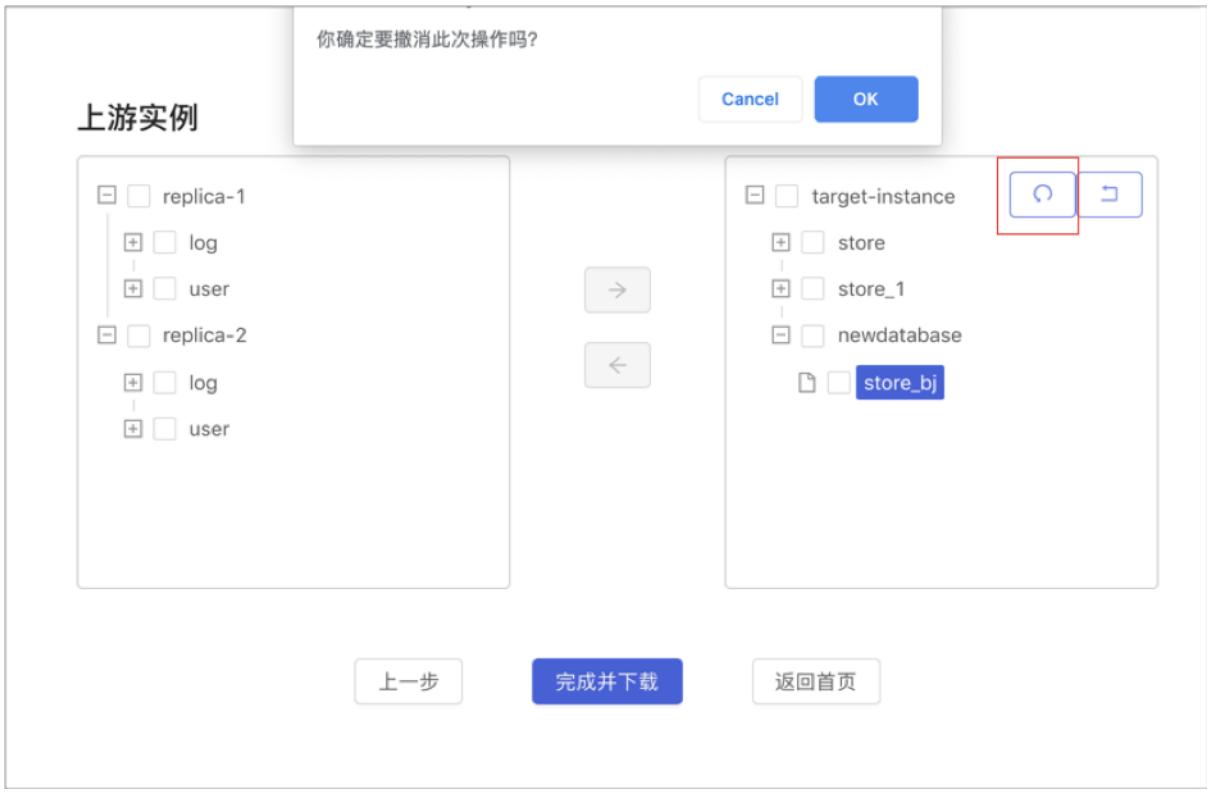


Figure 20: DM Portal Revert

12.3.5.4.7 清空下游实例

Figure 21: DM Portal Reset

12.3.5.4.8 完成并下载

Figure 22: DM Portal GenerateConfig

13 告警处理

13.1 DM 告警信息

使用 DM-Ansible 部署 DM 集群的时候，会默认部署一套告警系统。

> **注意：**
> 目前只有 DM-worker 提供了 alert rules，DM-master 暂未提供。

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。

13.2 告警处理

本文档介绍 DM 中各主要告警信息的处理方法。

13.2.1 任务状态告警

13.2.1.1 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。

13.2.2 relay log 告警

13.2.2.1 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。

13.2.2.2 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。
- 尝试迁移 DM-worker 实例到其他可用容量充足的磁盘上。

13.2.2.3 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。

13.2.2.4 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。

13.2.2.5 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。

13.2.2.6 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。

13.2.3 Dump/Load 告警

13.2.3.1 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。

13.2.3.2 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。

13.2.4 Binlog replication 告警

13.2.4.1 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。

13.2.4.2 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。

13.2.4.3 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。

14 故障处理

14.1 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。

14.1.1 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。

14.1.2 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。

14.1.3 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 mydumper 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。|

14.1.3.1 迁移任务中断并包含 invalid connection 错误

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。

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 或 query-error 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。

14.1.3.2 迁移任务中断并包含 driver: bad connection 错误

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。

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。

14.1.3.3 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

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。

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。

14.1.3.4 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 加密上游 MySQL 用户密码。

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。

14.2 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`。

14.2.1 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。

14.2.1.1 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。

14.2.1.2 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。

14.2.1.3 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。

14.2.2 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。

14.2.3 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。

14.2.3.1 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。

14.2.3.2 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。

14.2.3.3 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。

15 Data Migration 常见问题

15.1 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。

15.2 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。

15.3 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。

15.4 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，详见跳过或替代执行异常的 SQL 语句)。

> **注意：**
>
> TiDB 目前并不兼容 MySQL 支持的所有 DDL 语句。

15.5 如何重置数据迁移任务？

15.5.1 relay log 无异常时重置迁移任务

如果数据迁移任务所需要的 relay log 不存在异常，可使用如下步骤重置数据迁移任务以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` 重启迁移任务。
    - 修改任务配置文件以设置 `remove-meta` 为 `true`，然后使用 `start-task` 重启迁移任务。

15.5.2 relay log 存在异常时重置迁移任务

15.5.2.1 所需的 relay log 在上游 MySQL 中存在

如果迁移任务所需要的 relay log 在 DM-worker 中存在异常，但仍然正常存在于上游 MySQL 中时，可使用如下步骤恢复数据迁移任务：

1. 使用 `stop-task` 停止当前正在运行的所有迁移任务。
2. 参考重启 DM-worker文档，停止存在异常的 DM-worker 节点。
3. 从上游 MySQL 中复制正常的 binlog 文件以替换 DM-worker 的relay log 目录内的对应文件。
    - 如果是使用 DM-Ansible 部署，relay log 目录即 `<deploy_dir>/relay_log` 目录。
    - 如果是使用二进制文件手动部署，relay log 目录即 `relay-dir` 参数设置的目录。
4. 修改 DM-worker 的 relay log 目录内的 `relay.meta` 的信息为下一个 binlog 文件对应的信息。
    - 如果未启用 `enable-gtid`，则将 `binlog-name` 设置为下一个 binlog 文件的文件名，并将 `binlog-pos` 设置为 4。如从上游 MySQL 复制了 `mysq-bin.000100` 到 relay 目录，并期望之后从 `mysql-bin.000101` 开始继续 binlog 的拉取，则将 `binlog-name` 设置为 `mysql-bin.000101`。
    - 如果启用了 `enable-gtid`，则将 `binlog-gtid` 设置为下一个 binlog 文件起始处的 `Previous_gtids` 对应的值（可通过在上游 MySQL 执行 SHOW BINLOG EVENTS 获得）。
5. 参考重启 DM-worker文档，启动存在异常的 DM-worker 节点。
6. 使用 `start-task` 恢复已停止的迁移任务。

15.5.2.2 所需的 relay log 在上游 MySQL 已清理

如果迁移任务所需要的 relay log 在 DM-worker 中存在异常、且在上游已被清理，可使用如下步骤重置数据迁移任务以对数据重新进行迁移：

1. 使用 `stop-task` 命令停止当前正在运行的所有迁移任务。
2. 使用 DM-Ansible 停止整个 DM 集群。
3. 手动清理掉与 binlog event 被重置的 MySQL 相对应的 DM-worker 的 relay log 目录。
    - 如果是使用 DM-Ansible 部署，relay log 目录即 `<deploy_dir>/relay_log` 目录。
    - 如果是使用二进制文件手动部署，relay log 目录即 `relay-dir` 参数设置的目录。
4. 清理掉下游已迁移的数据。
5. 使用 DM-Ansible 启动整个 DM 集群。
6. 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` 重启迁移任务。
    - 修改任务配置文件以设置 `remove-meta` 为 `true`，然后使用 `start-task` 重启迁移任务。

15.6 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。

15.7 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。

15.7.1 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。

15.7.2 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。

15.7.3 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。

15.8 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: "*"。

15.9 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，会满足“启动或恢复任务的前 5 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。

15.10 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。

16 版本发布历史

16.1 v1.0

16.1.1 DM 1.0.7 Release Notes

发版日期：2021 年 6 月 21 日

DM 版本：1.0.7

16.1.1.1 Bug 修复

- 修复同步任务中断重启后可能丢数据的问题 #1783

16.1.2 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

16.1.2.1 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

16.1.2.2 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`

16.1.2.3 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

16.1.3 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

16.1.3.1 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

16.1.3.2 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

16.1.3.3 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

16.1.4 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

16.1.4.1 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

16.1.4.2 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

16.1.4.3 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

16.1.5 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

16.1.5.1 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

16.1.5.2 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

16.1.5.3 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

16.1.6 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

16.1.6.1 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

16.1.6.2 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

16.1.6.3 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

17 TiDB Data Migration 术语表

本文档介绍 TiDB Data Migration (TiDB DM) 相关术语。

17.1 B

17.1.1 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。

17.1.2 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。

17.1.3 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。

17.1.4 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`。

17.1.5 Binlog replication 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。

17.1.6 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。

17.2 C

17.2.1 Checkpoint

TiDB DM 在全量迁移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。

- 对于全量迁移，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。

17.3 D

17.3.1 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。

17.4 F

17.4.1 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。

17.5 G

17.5.1 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。

17.6 H

17.6.1 Heartbeat

在增量数据复制过程中，用于估算数据从在上游写入后到达 Binlog replication 处理单元延迟时间的机制，具体可参考迁移延迟监控。

17.7 L

17.7.1 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。

17.8 Q

17.8.1 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。

17.9 R

17.9.1 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。

17.9.2 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。

17.10 S

17.10.1 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：

- 启动或恢复增量复制任务的前 5 分钟保持启用
- 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

17.10.2 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。

17.10.3 Shard DDL lock

用于协调 Shard DDL 迁移的锁机制，具体原理可查看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。

17.10.4 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。

17.10.5 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。

17.10.6 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。

17.11 T

17.11.1 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。

17.11.2 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。

17.11.3 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。
