

# Table of Contents

Introduction	1.1
0.概述	1.2
1.适用场景	1.3
1.0.MySQL的单向复制/聚合/分散	1.3.1
1.1.跨数据中心的双向复制	1.3.2
1.2.公有云间的数据同步	1.3.3
1.3.MySQL到Kafka的数据变更通知	1.3.4
2.快速开始	1.4
2.0.MySQL的单向复制	1.4.1
2.1.MySQL的聚合复制	1.4.2
2.2.MySQL的数据分散	1.4.3
2.3.MySQL的跨数据中心的双向复制	1.4.4
2.4.阿里云到京东云的MySQL复制	1.4.5
2.5.MySQL到Kafka的数据变更通知	1.4.6
3.功能说明	1.5
3.0.功能/场景的映射列表	1.5.1
3.1.使用限制	1.5.2
3.2.端口使用说明	1.5.3
3.3.对目标端数据库的影响	1.5.4
3.4.监控项说明	1.5.5
4.安装/配置说明	1.6
4.0.安装步骤	1.6.1
4.1.节点配置	1.6.2
4.2.命令说明	1.6.3
4.3.作业(job)配置	1.6.4
4.4.HTTP API说明	1.6.5
4.5.MySQL 用户权限说明	1.6.6
5.设计说明	1.7
5.1.时间/资源估算	1.7.1
5.2 基本架构	1.7.2
5.3 Kafka 消息格式	1.7.3
6.如何参与	1.8
7.路线图	1.9



# dtle 中文技术参考手册

## 目录

参考 gitbook 左侧目录区 或 [SUMMARY.md](#)

## PDF下载

[PDF下载](#)

## 官方技术支持

- 代码库 [github: `github.com/actiontech/dtle`](https://github.com/actiontech/dtle)
- 文档库 [github: `github.com/actiontech/dtle-docs-cn`](https://github.com/actiontech/dtle-docs-cn)
- 文档库 pages: [actiontech.github.io/dtle-docs-cn](https://actiontech.github.io/dtle-docs-cn)
- QQ group: 852990221

## 联系我们

如果想获得 dtle 的商业支持, 您可以联系我们:

- 全国支持: 400-820-6580
- 华北地区: 86-13718877200, 王先生
- 华南地区: 86-18503063188, 曹先生
- 华东地区: 86-18930110869, 梁先生
- 西南地区: 86-18328335660, 雷先生

## 概述

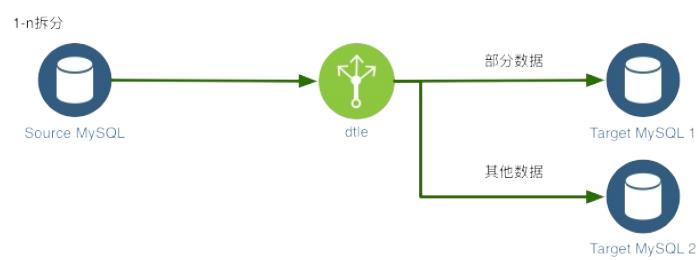
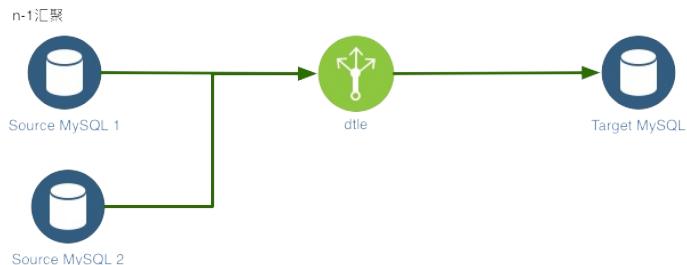
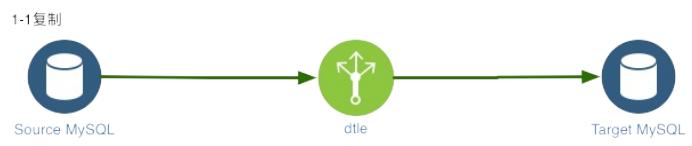
dtle (Data-Transformation-le) 是[上海爱可生信息技术股份有限公司](#) 开发并开源的 **CDC** 工具. 其功能特点是:

- 多种数据传输模式
  - 支持链路压缩
  - 支持同构传输和异构传输
  - 支持跨网络边际的传输
- 多种数据处理模式
  - 支持库/表/行级别 数据过滤
- 多种数据通道模式
  - 支持多对多的数据传输
  - 支持回环传输
- 多种源/目标端
  - 支持MySQL - MySQL的数据传输
  - 支持MySQL - Kafka的数据传输
- 集群模式
  - 提供可靠的元数据存储
  - 可进行自动任务分配
  - 支持自动故障转移

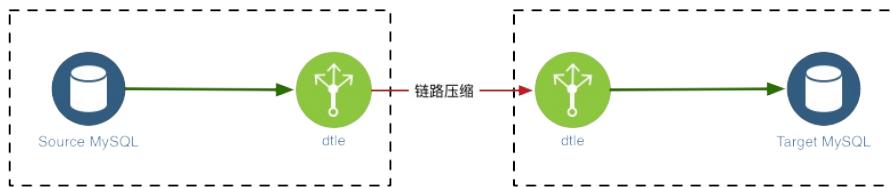
## 1.0 MySQL的单向复制/聚合/分散

如下图, dtle 支持 MySQL 单向数据复制的常见场景如下:

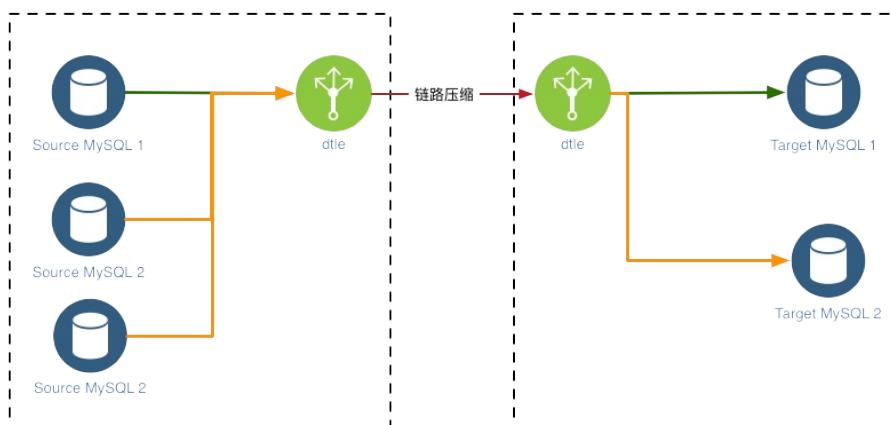
- 按数据源/数据目标的映射关系划分
  - 支持1:1的复制
  - 支持n:1的数据汇聚, 将多个数据源的数据 聚合到 同一个数据目标
  - 支持1:n的数据拆分, 将一个数据源的数据 拆分到 多个数据目标
- 按网络类型划分
  - 支持网络内的数据传输
  - 支持跨网络边际的数据传输 (可使用 链路压缩/链路限流 等功能)
- 按集群规模划分
  - 可配置 单一dtle实例 处理 单一数据通道
  - 可配置 dtle集群 处理 多个数据通道



跨网络边际的1-1复制



跨网络边际的多个复制通道

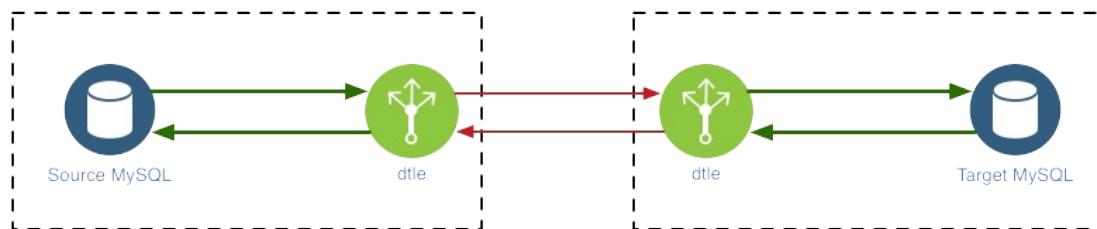




## 1.1 跨数据中心的双向复制

如下图, dtile 支持 MySQL 间的双向复制, 其普遍场景是用于跨数据中心的数据双向同步.

跨数据中心的双向复制



其中:

- dtile 会对数据的回环状况进行判断, 不会重复传输同一事务.
- dtile 在传输过程中维持数据的事务性, 对于数据源的事务产生的数据, 在数据目标端是以相同的事务方式进行回放. 对于双写的场景, 目标端不会受到不完整的事务的影响.
- dtile 在数据链路上, 可使用压缩/限速等功能, 更适合于跨数据中心的场景.

## 1.2 公有云间的数据同步

dtle 可用于公有云间的数据同步, 可支持的部署方式同 [1.0](#) 和 [1.1](#) 两节介绍的方式. 其中的不同之处在于:

- dtle 可部署于公有云的云主机服务上
- dtle 对公有云上RDS服务给予的权限进行了适配, 不需高权限即可实现数据复制
- dtle 对公有云的 MySQL 非官方版 进行了适配 (如阿里云RDS会增加隐式主键列, 导致 binlog中的数据与表结构不符)

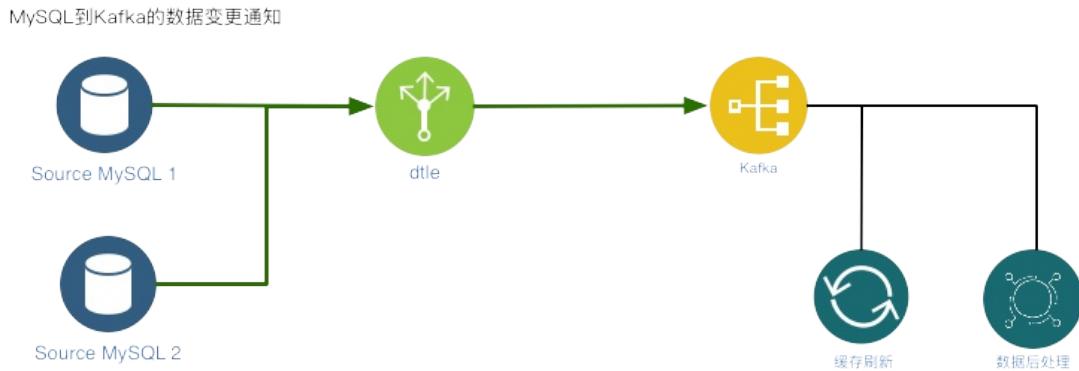
目前支持的公有云同步通道:

- 阿里云 -> 京东云

## 1.3 MySQL到Kafka的数据变更通知

如下图, dtle支持MySQL到Kafka的数据变更通知, 其普遍场景是:

- 当数据变更时, 通知 缓存件 进行缓存刷新
- 当数据变更时, 通知 数据后处理件 进行数据扫描



## MySQL 的单向复制

以下步骤以docker容器的方式快速演示如何搭建MySQL的单向复制环境.

### 创建网络

```
docker network create dtle-net
```

### 创建源端/目标端 MySQL

```
docker run --name mysql-src -e MYSQL_ROOT_PASSWORD=pass -p 33061:3306 --network=dtl
e-net -d mysql:5.7 --gtid-mode=ON --enforce-gtid-consistency=1 --log-bin=bin --serv
er-id=1
```

```
docker run --name mysql-dst -e MYSQL_ROOT_PASSWORD=pass -p 33062:3306 --network=dtl
e-net -d mysql:5.7 --gtid-mode=ON --enforce-gtid-consistency=1 --log-bin=bin --serv
er-id=2
```

检查是否联通:

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "select @@version\G"
< *****
@@version: 5.7.23-log

> mysql -h 127.0.0.1 -P 33062 -uroot -ppass -e "select @@version\G"
< *****
@@version: 5.7.23-log
```

### 创建 dtle

```
docker run --name dtle -p 8190:8190 --network=dtle-net -d actiontech/dtle
```

检查是否正常:

```
> curl -XGET "127.0.0.1:8190/v1/nodes" -s | jq
< [
  {
    "CreateIndex": 4,
    "Datacenter": "dc1",
    "HTTPAddr": "0.0.0.0:8190",
    "ID": "f7051e01-b323-5056-de5d-4958949a7bc2",
    "ModifyIndex": 99,
```

```

    "Name": "7539b7887eb2",
    "Status": "ready",
    "StatusDescription": ""
}
]

```

## 准备作业定义文件

准备文件job.json, 内容如下:

```

{
    "Name": "dtle-demo",
    "Tasks": [
        {
            "Type": "Src",
            "Config": {
                "Gtid": "",
                "ReplicateDoDb": [
                    {
                        "TableSchema": "demo",
                        "Tables": [
                            {
                                "TableName": "demo_tbl"
                            }
                        ]
                    }
                ],
                "ConnectionConfig": {
                    "Host": "mysql-src",
                    "Port": "3306",
                    "User": "root",
                    "Password": "pass"
                }
            }
        },
        {
            "Type": "Dest",
            "Config": {
                "ConnectionConfig": {
                    "Host": "mysql-dst",
                    "Port": "3306",
                    "User": "root",
                    "Password": "pass"
                }
            }
        }
    ]
}

```

其中定义了：

- 源端/目标端的连接字符串
- 要复制的表为 `demo.demo_tbl`
- GTID点位为空, 表示此复制是 全量+增量 的复制. 如只测试增量复制, 可指定合法的GTID

## 准备测试数据

可在源端准备提前建表 `demo.demo_tbl`, 并插入数据, 以体验全量复制过程. 也可不提前建表.

## 创建复制任务

```
> curl -H "Accept:application/json" -XPOST "http://127.0.0.1:8190/v1/jobs" -d @job.json -s | jq
< {
  "Index": 234,
  "KnownLeader": false,
  "LastContact": 0,
  "Success": true
}
```

查看作业ID:

```
> curl -XGET "127.0.0.1:8190/v1/jobs" -s | jq '.[].ID'
< "67b32f86-2c65-0a4b-8101-6301e4dc41ff"
```

查看作业状态

```
> curl -XGET "127.0.0.1:8190/v1/job/67b32f86-2c65-0a4b-8101-6301e4dc41ff" -s | jq '.Status'
< "running"
```

## 测试

此时可在源端对表 `demo.demo_tbl` 进行DDL/DML等各种操作, 查看目标端数据是否一致

## MySQL 的汇聚复制

以下步骤以docker容器的方式快速演示如何搭建MySQL的汇聚复制环境.

### 创建网络

```
docker network create dtle-net
```

### 创建源端(2个)和目标端(1个) MySQL

```
docker run --name mysql-src1 -e MYSQL_ROOT_PASSWORD=pass -p 33061:3306 --network=dtle-net -d mysql:5.7 --gtid-mode=ON --enforce-gtid-consistency=1 --log-bin=bin --server-id=1

docker run --name mysql-src2 -e MYSQL_ROOT_PASSWORD=pass -p 33062:3306 --network=dtle-net -d mysql:5.7 --gtid-mode=ON --enforce-gtid-consistency=1 --log-bin=bin --server-id=2

docker run --name mysql-dst -e MYSQL_ROOT_PASSWORD=pass -p 33063:3306 --network=dtle-net -d mysql:5.7 --gtid-mode=ON --enforce-gtid-consistency=1 --log-bin=bin --server-id=3
```

检查是否联通:

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "select @@version\G"
< *****
@@version: 5.7.23-log

> mysql -h 127.0.0.1 -P 33062 -uroot -ppass -e "select @@version\G"
< *****
@@version: 5.7.23-log

> mysql -h 127.0.0.1 -P 33063 -uroot -ppass -e "select @@version\G"
< *****
@@version: 5.7.23-log
```

### 在源端MySQL中创建表结构, 获取GTID点位, 并插入数据

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "CREATE DATABASE demo; CREATE TABLE demo.demo_tbl(a int primary key)"
< ...

> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "show master status\G" | grep "Execu
```

```

ted_Gtid_Set"
< Executed_Gtid_Set: f6def853-cbaa-11e8-8aeb-0242ac120003:1-7

> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "insert into demo.demo_tbl values(1)
,(2),(3)"
< ...

---

> mysql -h 127.0.0.1 -P 33062 -uroot -ppass -e "CREATE DATABASE demo; CREATE TABLE
demo.demo_tbl(a int primary key)"
< ...

> mysql -h 127.0.0.1 -P 33062 -uroot -ppass -e "show master status\G" | grep "Execu
ted_Gtid_Set"
< Executed_Gtid_Set: f74aacb5-cbaa-11e8-bdd1-0242ac120004:1-7

> mysql -h 127.0.0.1 -P 33062 -uroot -ppass -e "insert into demo.demo_tbl values(4)
,(5),(6)"
< ...

```

## 在目标端MySQL中创建表结构

```

> mysql -h 127.0.0.1 -P 33063 -uroot -ppass -e "CREATE DATABASE demo; CREATE TABLE
demo.demo_tbl(a int primary key)"
< ...

```

## 创建 dtle

```
docker run --name dtle -p 8190:8190 --network=dtle-net -d actiontech/dtle
```

检查是否正常:

```

> curl -XGET "127.0.0.1:8190/v1/nodes" -s | jq
< [
  {
    "CreateIndex": 4,
    "Datacenter": "dc1",
    "HTTPAddr": "0.0.0.0:8190",
    "ID": "f7051e01-b323-5056-de5d-4958949a7bc2",
    "ModifyIndex": 99,
    "Name": "7539b7887eb2",
    "Status": "ready",
    "StatusDescription": ""
  }
]

```

]

## 准备作业定义文件

### src1到dst的复制定义文件

准备src1\_dst.json, 内容如下:

```
{
  "Name": "dtle-demo-src1-dst",
  "Tasks": [
    {
      "Type": "Src",
      "Config": {
        "Gtid": "f6def853-cbaa-11e8-8aeb-0242ac120003:1-7",
        "ReplicateDoDb": [
          {
            "TableSchema": "demo",
            "Tables": [
              {
                "TableName": "demo_tbl"
              }
            ]
          }
        ],
        "ConnectionConfig": {
          "Host": "mysql-src1",
          "Port": "3306",
          "User": "root",
          "Password": "pass"
        }
      }
    },
    {
      "Type": "Dest",
      "Config": {
        "ConnectionConfig": {
          "Host": "mysql-dst",
          "Port": "3306",
          "User": "root",
          "Password": "pass"
        }
      }
    }
  ]
}
```

其中定义了:

- 源端/目标端的连接字符串
- 要复制的表为 `demo.demo_tbl`
- GTID点位为 准备数据阶段 插入数据之前的src1上的GTID点位

## src2到dst的复制定义文件

准备src2\_dst.json, 内容如下:

```
{
  "Name": "dtle-demo-src2-dst",
  "Tasks": [
    {
      "Type": "Src",
      "Config": {
        "Gtid": "f74aacb5-cbaa-11e8-bdd1-0242ac120004:1-7",
        "ReplicateDoDb": [
          {
            "TableSchema": "demo",
            "Tables": [
              {
                "TableName": "demo_tbl"
              }
            ]
          }
        ],
        "ConnectionConfig": {
          "Host": "mysql-src2",
          "Port": "3306",
          "User": "root",
          "Password": "pass"
        }
      }
    },
    {
      "Type": "Dest",
      "Config": {
        "ConnectionConfig": {
          "Host": "mysql-dst",
          "Port": "3306",
          "User": "root",
          "Password": "pass"
        }
      }
    }
  ]
}
```

其中与 `src1_dst.json` 不同的是:

- 源端的连接字符串
- GTID点位为 准备数据阶段 插入数据之前的src2上的GTID点位

## 创建复制任务

```
> curl -H "Accept:application/json" -XPOST "http://127.0.0.1:8190/v1/jobs" -d @src1
_dst.json -s | jq
< {
  "Index": 328,
  "KnownLeader": false,
  "LastContact": 0,
  "Success": true
}

> curl -H "Accept:application/json" -XPOST "http://127.0.0.1:8190/v1/jobs" -d @src2
_dst.json -s | jq
< {
  "Index": 341,
  "KnownLeader": false,
  "LastContact": 0,
  "Success": true
}
```

查看作业ID和状态:

```
> curl -XGET "127.0.0.1:8190/v1/jobs" -s | jq '.[] | .ID, .Status'
< "484c0099-c45c-397b-6c76-13e4c5efe7a1"
"running"
"cab19f6a-fe44-497c-e03d-0bf3755163a7"
"running"
```

## 测试

在src1和src2中分别插入数据, 查看dst中的数据, 验证全量和增量的数据均存在

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "insert into demo.demo_tbl values(11)
)"
< ...

> mysql -h 127.0.0.1 -P 33062 -uroot -ppass -e "insert into demo.demo_tbl values(12
)"
< ...

> mysql -h 127.0.0.1 -P 33063 -uroot -ppass -e "select * from demo.demo_tbl"
<
+---+
| a  |
| a  |
```

```
+----+  
| 1 |  
| 2 |  
| 3 |  
| 4 |  
| 5 |  
| 6 |  
| 11 |  
| 12 |  
+----+
```

## MySQL 的数据分散

以下步骤以docker容器的方式快速演示如何搭建MySQL的数据分散环境. 数据分散复制, 将源表中的数据中, 主键<5 的行复制到目标库1, 主键>=5 的行复制到目标库2.

### 创建网络

```
docker network create dtle-net
```

### 创建源端(1个)和目标端(2个) MySQL

```
docker run --name mysql-src -e MYSQL_ROOT_PASSWORD=pass -p 33061:3306 --network=dtle-net -d mysql:5.7 --gtid-mode=ON --enforce-gtid-consistency=1 --log-bin=bin --server-id=1

docker run --name mysql-dst1 -e MYSQL_ROOT_PASSWORD=pass -p 33062:3306 --network=dtle-net -d mysql:5.7 --gtid-mode=ON --enforce-gtid-consistency=1 --log-bin=bin --server-id=2

docker run --name mysql-dst2 -e MYSQL_ROOT_PASSWORD=pass -p 33063:3306 --network=dtle-net -d mysql:5.7 --gtid-mode=ON --enforce-gtid-consistency=1 --log-bin=bin --server-id=3
```

检查是否联通:

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "select @@version\G"
< *****
@@version: 5.7.23-log

> mysql -h 127.0.0.1 -P 33062 -uroot -ppass -e "select @@version\G"
< *****
@@version: 5.7.23-log

> mysql -h 127.0.0.1 -P 33063 -uroot -ppass -e "select @@version\G"
< *****
@@version: 5.7.23-log
```

### 在源端MySQL中创建表结构, 获取GTID点位, 并插入数据

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "CREATE DATABASE demo; CREATE TABLE demo.demo_tbl(a int primary key)"
< ...
```

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "show master status\G" | grep "Executed_Gtid_Set"
< Executed_Gtid_Set: 167dd42f-d076-11e8-8104-0242ac120003:1-7

> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "insert into demo.demo_tbl values(1),(2),(3)"
< ...


```

## 在目标端MySQL中创建表结构

```
> mysql -h 127.0.0.1 -P 33062 -uroot -ppass -e "CREATE DATABASE demo; CREATE TABLE demo.demo_tbl(a int primary key)"
< ...

> mysql -h 127.0.0.1 -P 33063 -uroot -ppass -e "CREATE DATABASE demo; CREATE TABLE demo.demo_tbl(a int primary key)"
< ...


```

## 创建 dtle

```
docker run --name dtle -p 8190:8190 --network=dtle-net -d actiontech/dtle
```

检查是否正常:

```
> curl -XGET "127.0.0.1:8190/v1/nodes" -s | jq
< [
  {
    "CreateIndex": 4,
    "Datacenter": "dc1",
    "HTTPAddr": "0.0.0.0:8190",
    "ID": "f7051e01-b323-5056-de5d-4958949a7bc2",
    "ModifyIndex": 99,
    "Name": "7539b7887eb2",
    "Status": "ready",
    "StatusDescription": ""
  }
]
```

## 准备作业定义文件

### src到dst1的复制定义文件

准备src\_dst1.json, 内容如下:

```
{
  "Name": "dtle-demo-src-dst1",
  "Tasks": [
    {
      "Type": "Src",
      "Config": {
        "ReplicateDoDb": [
          {
            "TableSchema": "demo",
            "Tables": [
              {
                "TableName": "demo_tbl",
                "Where": "a<5"
              }
            ]
          }
        ],
        "ConnectionConfig": {
          "Host": "mysql-src",
          "Port": "3306",
          "User": "root",
          "Password": "pass"
        }
      }
    },
    {
      "Type": "Dest",
      "Config": {
        "ConnectionConfig": {
          "Host": "mysql-dst1",
          "Port": "3306",
          "User": "root",
          "Password": "pass"
        }
      }
    }
  ]
}
```

其中定义了：

- 源端/目标端的连接字符串
- 要复制的表为 demo.demo\_tbl
- demo\_tbl 的复制数据条件为 a<5

## src到dst2的复制定义文件

准备src\_dst2.json, 内容如下：

```
{
  "Name": "dtle-demo-src-dst2",
  "Tasks": [
    {
      "Type": "Src",
      "Config": {
        "ReplicateDoDb": [
          {
            "TableSchema": "demo",
            "Tables": [
              {
                "TableName": "demo_tbl",
                "Where": "a>=5"
              }
            ]
          }
        ],
        "ConnectionConfig": {
          "Host": "mysql-src",
          "Port": "3306",
          "User": "root",
          "Password": "pass"
        }
      }
    },
    {
      "Type": "Dest",
      "Config": {
        "ConnectionConfig": {
          "Host": "mysql-dst2",
          "Port": "3306",
          "User": "root",
          "Password": "pass"
        }
      }
    }
  ]
}
```

其中定义了：

- 源端/目标端的连接字符串
- 要复制的表为 demo.demo\_tbl
- demo\_tbl 的复制数据条件为 a>=5

其中与 src1\_dst.json 不同的是：

- 源端的连接字符串
- demo\_tbl 的复制数据条件

## 创建复制任务

```
> curl -H "Accept:application/json" -XPOST "http://127.0.0.1:8190/v1/jobs" -d @src_dst1.json -s | jq
< {
  "Index": 19663,
  "KnownLeader": false,
  "LastContact": 0,
  "Success": true
}

> curl -H "Accept:application/json" -XPOST "http://127.0.0.1:8190/v1/jobs" -d @src_dst2.json -s | jq
< {
  "Index": 19908,
  "KnownLeader": false,
  "LastContact": 0,
  "Success": true
}
```

查看作业ID和状态:

```
> curl -XGET "127.0.0.1:8190/v1/jobs" -s | jq '.[] | .ID, .Status'
< "4544d894-368f-a9d7-7b06-37330980e935"
"running"
"62059ef6-0b7e-b7d8-b386-80aa86791f4a"
"running"
```

## 测试

在src中插入数据, 查看dst1/dst2中的数据, 验证全量和增量的数据均存在

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "insert into demo.demo_tbl values(0), (10)"
< ...
>
> mysql -h 127.0.0.1 -P 33062 -uroot -ppass -e "select * from demo.demo_tbl"
<
+---+
| a |
+---+
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
```

```
+----+  
> mysql -h 127.0.0.1 -P 33063 -uroot -ppass -e "select * from demo.demo_tbl"  
<  
+----+  
| a |  
+----+  
| 5 |  
| 6 |  
| 7 |  
| 8 |  
| 9 |  
| 10 |  
+----+
```

## MySQL的跨数据中心的双向复制

以下步骤以docker容器的方式快速演示如何搭建MySQL的跨数据中心的双向复制.

### 创建两个网络

```
docker network create dtle-net-dc1
docker network create dtle-net-dc2
```

### 在两个网络中分别创建MySQL

```
docker run --name mysql-dc1 -e MYSQL_ROOT_PASSWORD=pass -p 33061:3306 --network=dtle-net-dc1 -d mysql:5.7 --gtid-mode=ON --enforce-gtid-consistency=1 --log-bin=bin --server-id=1

docker run --name mysql-dc2 -e MYSQL_ROOT_PASSWORD=pass -p 33062:3306 --network=dtle-net-dc2 -d mysql:5.7 --gtid-mode=ON --enforce-gtid-consistency=1 --log-bin=bin --server-id=2
```

检查MySQL是否启动成功:

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "select @@version\G"
< *****
      1. row *****

@@version: 5.7.23-log

> mysql -h 127.0.0.1 -P 33062 -uroot -ppass -e "select @@version\G"
< *****
      1. row *****

@@version: 5.7.23-log
```

### 在两个网络中分别创建dtle

```
docker run --name dtle-dc1 -p 8191:8190 --network=dtle-net-dc1 -d actiontech/dtle
docker run --name dtle-dc2 -p 8192:8190 --network=dtle-net-dc2 -d actiontech/dtle
```

### 将两个dtle通过公网连通

```
docker network create dtle-net-public
docker network connect dtle-net-public dtle-dc1
docker network connect dtle-net-public dtle-dc2
```

## 修改dtle的配置

### 修改容器dtle-dc1内的配置并重启

修改容器dtle-dc1内的配置并重启:

```
docker exec -u root -it dtle-dc1 vi /etc/dtle/dtle.conf
...
docker exec -u root -it dtle-dc1 rm -rf /dtle/data
docker restart dtle-dc1
```

配置 /etc/dtle/dtle.conf 修改的内容如下:

```
# Setup data dir
data_dir = "/dtle/data"

bind_addr = "172.22.0.2"
advertise {
    rpc = "172.22.0.2"
}

# Modify our port to avoid a collision with server
ports {
    http = 8190
}

# Enable the manager
manager {
    enabled = true
    join = [ "dtle-dc1" ]
}

# Enable the agent
agent {
    enabled = true
    managers = ["dtle-dc1:8191"]
}
```

其中:

- 由于dtle-dc1容器存在两个网络 (与MySQL通信的内网 `dtle-net-dc1` , 和与dtle-dc2通信的公网 `dtle-net-public` ), 需要指定 `bind_addr` 和 `advertise.rpc` 为本机的 `dtle-net-public` 的网络地址, 此处为 `172.22.0.2`

### 修改容器dtle-dc2内的配置并重启

修改容器dtle-dc2内的配置并重启:

```
docker exec -u root -it dtle-dc2 vi /etc/dtle/dtle.conf
...
docker exec -u root -it dtle-dc2 rm -rf /dtle/data
docker restart dtle-dc2
```

配置 `/etc/dtle/dtle.conf` 修改的内容如下:

```
# Setup data dir
data_dir = "/dtle/data"

bind_addr = "172.22.0.3"
advertise {
    rpc = "172.22.0.3"
}

# Modify our port to avoid a collision with server
ports {
    http = 8190
}

# Enable the manager
manager {
    enabled = false
}

# Enable the agent
agent {
    enabled = true
    managers = ["dtle-dc1:8191"]
}
```

其中:

- 由于`dtle-dc2`容器存在两个网络 (与MySQL通信的内网 `dtle-net-dc2` , 和与`dtle-dc1`通信的公网 `dtle-net-public` ), 需要指定 `bind_addr` 和 `advertise.rpc` 为本机的 `dtle-net-public` 的网络地址, 此处为 `172.22.0.3`

## 检查是否正常

```
> curl -XGET "127.0.0.1:8191/v1/nodes" -s | jq
< [
{
    "CreateIndex": 5,
    "Datacenter": "dc1",
    "HTTPAddr": "172.22.0.2:8190",
    "ID": "2d1f92a8-c7cd-35b8-86ec-f81867f98c41",
    "ModifyIndex": 13,
    "Name": "910e07e7389d",
```

```

    "Status": "ready",
    "StatusDescription": ""

},
{
    "CreateIndex": 4,
    "Datacenter": "dc1",
    "HTTPAddr": "172.22.0.3:8190",
    "ID": "79741bfc-2815-389b-9f0e-f6c327063ae5",
    "ModifyIndex": 14,
    "Name": "8898434cbc95",
    "Status": "ready",
    "StatusDescription": ""
}
]

```

## 配置dc1到dc2的复制

获取mysql-dc1的GTID:

```

> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "show master status\G" | grep "Executed_Gtid_Set"
< Executed_Gtid_Set: 41f102d4-d29f-11e8-8de7-0242ac130002:1-5

```

准备文件job-dc1-dc2.json, 内容如下:

```

{
    "Name": "dtle-demo-dc1-2-dc2",
    "Tasks": [
        {
            "Type": "Src",
            "NodeId": "2d1f92a8-c7cd-35b8-86ec-f81867f98c41",
            "Config": {
                "Gtid": "41f102d4-d29f-11e8-8de7-0242ac130002:1-5",
                "ReplicateDoDb": [
                    {
                        "TableSchema": "demo",
                        "Tables": [
                            {
                                "TableName": "demo_tbl"
                            }
                        ]
                    }
                ],
                "ConnectionConfig": {
                    "Host": "mysql-dc1",
                    "Port": "3306",
                    "User": "root",
                    "Password": "pass"
                }
            }
        }
    ]
}

```

```

        }
    }
},
{
    "Type": "Dest",
    "NodeId": "79741bfc-2815-389b-9f0e-f6c327063ae5",
    "Config": {
        "ConnectionConfig": {
            "Host": "mysql-dc2",
            "Port": "3306",
            "User": "root",
            "Password": "pass"
        }
    }
}
]
}

```

其中定义了：

- 源端/目标端的连接字符串
- 要复制的表为 demo.demo\_tbl
- GTID点位, 表示此复制是 增量复制 (双向复制 只支持增量复制)
- 源任务(Src)配置在dc1的dtle节点上执行 (通过NodeId指定)
- 目标任务(Dest)配置在dc2的dtle节点上执行 (通过NodeId指定)

## 创建dc1到dc2的复制任务

```

> curl -H "Accept:application/json" -XPOST "http://127.0.0.1:8191/v1/jobs" -d @job-
dc1-dc2.json -s | jq
< {
    "Index": 234,
    "KnownLeader": false,
    "LastContact": 0,
    "Success": true
}

```

查看作业ID：

```

> curl -XGET "127.0.0.1:8191/v1/jobs" -s | jq '.[].ID'
< "f0d4b250-40b3-060f-a096-ae6c7896ba72"

```

查看作业状态

```

> curl -XGET "127.0.0.1:8191/v1/job/f0d4b250-40b3-060f-a096-ae6c7896ba72" -s | jq '.
    .Status'
< "running"

```

## 配置dc2到dc1的复制

获取mysql-dc2的GTID:

```
> mysql -h 127.0.0.1 -P 33062 -uroot -ppass -e "show master status\G"
< *****
      File: bin.000003
      Position: 537
      Binlog_Do_DB:
      Binlog_Ignore_DB:
Executed_Gtid_Set: 41f102d4-d29f-11e8-8de7-0242ac130002:6-7,
42158e2f-d29f-11e8-b322-0242ac150002:1-5
```

准备文件job-dc2-dc1.json, 内容如下:

```
{
  "Name": "dtle-demo-dc2-2-dc1",
  "Tasks": [
    {
      "Type": "Src",
      "NodeId": "79741bfc-2815-389b-9f0e-f6c327063ae5",
      "Config": {
        "Gtid": "41f102d4-d29f-11e8-8de7-0242ac130002:6-7,42158e2f-d29f-11e8
-b322-0242ac150002:1-5",
        "ReplicateDoDb": [
          {
            "TableSchema": "demo",
            "Tables": [
              {
                "TableName": "demo_tbl"
              }
            ]
          }
        ],
        "ConnectionConfig": {
          "Host": "mysql-dc2",
          "Port": "3306",
          "User": "root",
          "Password": "pass"
        }
      }
    },
    {
      "Type": "Dest",
      "NodeId": "2d1f92a8-c7cd-35b8-86ec-f81867f98c41",
      "Config": {
        "ConnectionConfig": {
```

```

        "Host":"mysql-dc1",
        "Port":3306,
        "User":"root",
        "Password":"pass"
    }
}
]
}

```

其中与 dc1到dc2的复制任务 不同的是:

- 源端/目标端的连接字符串
- GTID点位
- 源任务(Src)配置在dc2的dtle节点上执行 (通过NodeId指定)
- 目标任务(Dest)配置在dc1的dtle节点上执行 (通过NodeId指定)

## 创建dc2到dc1的复制任务

```

> curl -H "Accept:application/json" -XPOST "http://127.0.0.1:8191/v1/jobs" -d @job-
dc2-dc1.json -s | jq
< {
  "Index": 234,
  "KnownLeader": false,
  "LastContact": 0,
  "Success": true
}

```

查看作业ID:

```

> curl -XGET "127.0.0.1:8191/v1/jobs" -s | jq '.[].ID'
< "6e32b339-33d2-9531-4bf7-fbc425f87388"
"f0d4b250-40b3-060f-a096-ae6c7896ba72"

```

查看作业状态

```

> curl -XGET "127.0.0.1:8191/v1/job/6e32b339-33d2-9531-4bf7-fbc425f87388" -s | jq '.
>Status'
< "running"

```

## 测试

此时可在任一端对表 demo.demo\_tbl 进行DDL/DML等各种操作, 查看目标端数据是否一致



## 阿里云到京东云的MySQL复制

以下步骤演示如何搭建从阿里云RDS到京东云RDS的MySQL复制.

### 检查阿里云RDS的环境

MySQL版本为5.7.18

检查权限:

```
mysql> select user();
+-----+
| user()          |
+-----+
| root@180.169.60.146 |
+-----+
1 row in set (0.02 sec)

mysql> show grants for 'root'@'%';
***** 1. row *****
Grants for root@%: GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROC
ESS, REFERENCES, INDEX, ALTER, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLI
CATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROU
TINE, CREATE USER, EVENT, TRIGGER ON *.* TO 'root'@'%' WITH GRANT OPTION
1 row in set (0.02 sec)
```

### 检查京东云RDS的环境

MySQL版本为5.7.21

**注意: 京东云RDS实例的用户权限是以schema为基础的. 需要在创建迁移job前, 通过京东云RDS为该MySQL实例创建两个schema: dtle(存储dtle元数据) 和 迁移的目标库**

```
mysql> select user();
+-----+
| user()          |
+-----+
| actiontech@180.169.60.146 |
+-----+
1 row in set (0.00 sec)

mysql> show grants for 'actiontech'@'%';
+-----+
| Grants for actiontech@%           |
+-----+
```

```
+-----+
| GRANT USAGE ON *.* TO 'actiontech'@'%'          |
| GRANT ALL PRIVILEGES ON `actiontech`.* TO 'actiontech'@'%' |
+-----+
2 rows in set (0.00 sec)
```

## 申请京东云ECS

需要申请京东云ECS, 用于

示例主机IP为 192.168.0.17 , 规格是1c4g40g

## 安装并配置dtle

安装dtle:

```
rpm -ivh --prefix /opt/dtle dtle.rpm
```

配置/etc/dtle/dtle.conf:

```
# Setup data dir
data_dir = "/opt/dtle/data"
log_file = "/opt/dtle/var/log/dtle/dtle.log"
log_level = "INFO"
bind_addr = "192.168.0.17"

# Modify our port to avoid a collision with server
ports {
    http = 8190
}

# Enable the manager
manager {
    enabled = true

    # Self-elect, should be 3 or 5 for production,
    # Addresses to attempt to join when the server starts.
    join = [ "192.168.0.17" ]
}

# Enable the agent
agent {
    enabled = true
    managers = ["192.168.0.17:8191"]
}
```

```

addresses {
    http = "192.168.0.17"
    rpc = "192.168.0.17"
    serf = "192.168.0.17"
}
advertise {
    http = "192.168.0.17"
    rpc = "192.168.0.17"
    serf = "192.168.0.17"
}

```

启动dtle:

```
systemctl start dtle
```

## 增加复制任务

复制配置文件 job.json 内容如下:

```
{
    "Name": "ali-jd-demo",
    "Tasks": [
        {
            "Type": "Src",
            "Config": {
                "Gtid": "",
                "ReplicateDoDb": [
                    {
                        "TableSchema": "actiontech",
                        "Tables": []
                    }
                ],
                "ConnectionConfig": {
                    "Host": "rm-xxxx.mysql.rds.aliyuncs.com",
                    "Port": "3306",
                    "User": "root",
                    "Password": "Acti0nTech"
                }
            }
        },
        {
            "Type": "Dest",
            "Config": {
                "ConnectionConfig": {
                    "Host": "mysql-cn-east-2-yyyy.public.jcloud.com",
                    "Port": "3306",
                    "User": "actiontech",
                    "Password": "Acti0nTech"
                }
            }
        }
    ]
}
```

```
        "Password": "Acti0ntech"
    }
}
]
}
```

向dtile发布任务:

```
curl -H "Accept:application/json" -XPOST "192.168.0.17:8190/v1/jobs" -d @job.json
```

检查任务运行状态:

```
curl -XGET "192.168.0.17:8190/v1/jobs" -s | jq '.[] | .ID, .Status'
```

## 其他

如要使用链路压缩等功能, 可参照[MySQL的跨数据中心的双向复制](#)

## MySQL到Kafka的数据变更通知

以下步骤以docker容器的方式快速演示如何搭建MySQL的单向复制环境.

### 创建网络

```
docker network create dtle-net
```

### 创建源端 MySQL

```
docker run --name mysql-src -e MYSQL_ROOT_PASSWORD=pass -p 33061:3306 --network=dtle-net -d mysql:5.7 --gtid-mode=ON --enforce-gtid-consistency=1 --log-bin=bin --server-id=1
```

检查是否联通:

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "select @@version\G"
< *****
@version: 5.7.23-log
```

### 创建源端表结构

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "CREATE DATABASE demo; CREATE TABLE demo.demo_tbl(a int primary key)"
```

### 创建目标端 Kafka

```
docker run --name kafka-zookeeper -p 2181:2181 -e ALLOW_ANONYMOUS_LOGIN=yes --network=dtle-net -d bitnami/zookeeper
docker run --name kafka-dst -p 9092:9092 -e KAFKA_ZOOKEEPER_CONNECT=kafka-zookeeper:2181 -e ALLOW_PLAINTEXT_LISTENER=yes --network=dtle-net -d bitnami/kafka
```

检查是否联通:

```
> docker run -it --rm \
--network dtle-net \
-e KAFKA_ZOOKEEPER_CONNECT=kafka-zookeeper:2181 \
bitnami/kafka:latest kafka-topics.sh --list --zookeeper kafka-zookeeper:2181
< Welcome to the Bitnami kafka container
Subscribe to project updates by watching https://github.com/bitnami/bitnami-docker-
```

kafka

Submit issues and feature requests at <https://github.com/bitnami/bitnami-docker-kafka/issues>

## 创建 dtle

```
docker run --name dtle -p 8190:8190 --network=dtle-net -d actiontech/dtle
```

检查是否正常:

```
> curl -XGET "127.0.0.1:8190/v1/nodes" -s | jq
< [
  {
    "CreateIndex": 4,
    "Datacenter": "dc1",
    "HTTPAddr": "0.0.0.0:8190",
    "ID": "f7051e01-b323-5056-de5d-4958949a7bc2",
    "ModifyIndex": 99,
    "Name": "7539b7887eb2",
    "Status": "ready",
    "StatusDescription": ""
  }
]
```

## 准备作业定义文件

准备文件job.json, 内容如下:

```
{
  "Name": "dtle-demo",
  "Tasks": [
    {
      "Type": "Src",
      "Config": {
        "Gtid": "",
        "ReplicateDoDb": [
          {
            "TableSchema": "demo",
            "Tables": [
              {
                "TableName": "demo_tbl"
              }
            ]
          }
        ],
        "ConnectionConfig": {
          "Host": "127.0.0.1",
          "Port": 3306,
          "User": "root",
          "Password": "password"
        }
      }
    }
  ]
}
```

```

        "Host": "mysql-src",
        "Port": "3306",
        "User": "root",
        "Password": "pass"
    }
}
},
{
    "Type": "Dest",
    "Driver": "Kafka",
    "Config": {
        "Topic": "demo-topic",
        "Brokers": ["kafka-dst:9092"],
        "Converter": "json"
    }
}
]
}

```

其中定义了：

- 源端 MySQL 的连接字符串
- 目标端 Kafka 的 broker 访问地址
- 要复制的表为 demo.demo\_tbl
- GTID点位为空, 表示此复制是 全量+增量 的复制. 如只测试增量复制, 可指定合法的GTID

## 创建复制任务

```

> curl -H "Accept:application/json" -XPOST "http://127.0.0.1:8190/v1/jobs" -d @job.json -s | jq
< {
    "Index": 25476,
    "KnownLeader": false,
    "LastContact": 0,
    "Success": true
}

```

查看作业ID:

```

> curl -XGET "127.0.0.1:8190/v1/jobs" -s | jq '.[].ID'
< "eb2f440d-916b-0c30-30ca-72aef1868cc0"

```

查看作业状态:

```

> curl -XGET "127.0.0.1:8190/v1/job/eb2f440d-916b-0c30-30ca-72aef1868cc0" -s | jq '.Status'
< "running"

```

## 测试

在源端写入数据:

```
> mysql -h 127.0.0.1 -P 33061 -uroot -ppass -e "INSERT INTO demo.demo_tbl values(1)
"
...
...
```

验证相关的topic存在:

```
> docker run -it --rm \
--network dtle-net \
-e KAFKA_ZOOKEEPER_CONNECT=kafka-zookeeper:2181 \
bitnami/kafka:latest kafka-topics.sh --list --zookeeper kafka-zookeeper:2181
< Welcome to the Bitnami kafka container
Subscribe to project updates by watching https://github.com/bitnami/bitnami-docker-
kafka
Submit issues and feature requests at https://github.com/bitnami/bitnami-docker-kaf
ka/issues

demo-topic.demo.demo_tbl
```

验证数据:

```
> docker run -it --rm \
--network dtle-net \
-e KAFKA_ZOOKEEPER_CONNECT=kafka-zookeeper:2181 \
bitnami/kafka:latest kafka-console-consumer.sh --bootstrap-server kafka-dst:909
2 --topic demo-topic.demo.demo_tbl --from-beginning
< ...
{"schema": {"type": "struct", "optional": false, "fields": [{"type": "struct", "optional": true, "field": "before", "fields": [{"type": "int32", "optional": false, "field": "a"}], "name": "demo-topic.demo.demo_tbl.Value"}, {"type": "struct", "optional": true, "field": "after", "fields": [{"type": "int32", "optional": false, "field": "a"}], "name": "demo-topic.demo.demo_tbl.Value"}, {"type": "struct", "optional": false, "field": "source", "fields": [{"type": "string", "optional": true, "field": "version"}, {"type": "string", "optional": false, "field": "name"}, {"type": "int64", "optional": false, "field": "server_id"}, {"type": "int64", "optional": false, "field": "ts_sec"}, {"type": "string", "optional": true, "field": "gtid"}, {"type": "string", "optional": false, "field": "file"}, {"type": "int64", "optional": false, "field": "pos"}, {"type": "int32", "optional": false, "field": "row"}, {"type": "boolean", "optional": true, "field": "snapshot"}, {"type": "int64", "optional": true, "field": "thread"}, {"type": "string", "optional": true, "field": "db"}, {"type": "string", "optional": true, "field": "table"}, {"name": "io.debezium.connector.mysql.Source"}, {"type": "string", "optional": false, "field": "op"}, {"type": "int64", "optional": true, "field": "ts_ms"}], "name": "demo-topic.demo.demo_tbl.Envelope", "version": 1}, "payload": {"before": null, "after": {"a": 11}, "source": {"version": "0.0.1", "name": "demo-topic", "server_id": 0, "ts_sec": 0, "gtid": null, "file": "", "pos": 0, "row": 1, "snapshot": true, "thread": null, "db": "demo", "ta
```

```
ble:"demo_tbl"}, "op":"c", "ts_ms":1539760682507}]}
```

此时可在源端对表 `demo.demo_tbl` 进行DDL/DML等各种操作, 查看目标端数据是否一致

关于Kafka的消息格式, 参看[5.3 Kafka 消息格式](#)

## 功能/场景的映射列表

场景	复制手段 (binlog-binlog)	复制手段 (binlog-sql)	复制模式 (全量+增量)	复制模式(增量)
单个MySQL 单向复制到 单个MySQL	支持	支持	支持	支持
单个MySQL 双向复制到 单个MySQL	支持	支持	-	支持
多个MySQL的表 合并到 单个MySQL	支持	支持	支持	支持
单个MySQL的不同表 分发 到 多个MySQL	支持	支持	支持	支持
单个MySQL的同一表的不同记录 分发 到 多个 MySQL	支持按主键分发; 不支持按函数分发	支持	支持	支持按主键分发; 不支持按函数分发
公有云间的数据同步	不支持	支持	支持	支持
单个MySQL复制到Kafka	-	-	支持	支持
多个MySQL复制到Kafka	-	-	支持	支持

场景	复制对象(整库复制)	复制对象 (整表复制)	复制对象(按条件 复制部分记录)
单个MySQL 单向复制到 单个MySQL	支持	支持	支持
单个MySQL 双向复制到 单个MySQL	支持; 同名库的双向复制存在缺陷 #312	支持	支持
多个MySQL的表 合并到 单个MySQL	-	支持	支持
单个MySQL的不同表 分发到 多个MySQL	-	支持	支持
单个MySQL的同一表的不同记录 分发 到 多个MySQL	-	支持	支持
公有云间的数据同步	支持	支持	支持
单个MySQL复制到Kafka	支持	支持	支持
多个MySQL复制到Kafka	支持	支持	支持

场景	复制链路(链路压缩)	复制链路(跨网络边际)	回访模式(并行回放)
单个MySQL 单向复制到 单个MySQL	支持	支持	支持
单个MySQL 双向复制到 单个MySQL	支持	支持	支持

多个MySQL的表 合并到 单个MySQL	支持	支持	支持
单个MySQL的不同表 分发到 多个MySQL	支持	支持	支持
单个MySQL的同一表的不同记录 分发到 多个MySQL	支持	支持	支持
公有云间的数据同步	支持	支持	支持
单个MySQL复制到Kafka	支持	支持	-
多个MySQL复制到Kafka	支持	支持	-

场景	自动创建表结构	支持DDL	Agent 水平扩展
单个MySQL 单向复制到 单个MySQL	支持	支持	支持
单个MySQL 双向复制到 单个MySQL	-	支持	支持
多个MySQL的表 合并到 单个MySQL	支持	支持	支持
单个MySQL的不同表 分发到 多个MySQL	支持	支持	支持
单个MySQL的同一表的不同记录 分发 到 多个MySQL	支持	支持按主键分发; 不支持按函数分发	支持按主键分发; 不支持按函数分发
公有云间的数据同步	支持	支持	支持
单个MySQL复制到Kafka	支持	不支持	支持
多个MySQL复制到Kafka	支持	不支持	支持

场景	高可用(故障转移)	高可用(断点续做)	任务暂停/恢复	监控
单个MySQL 单向复制到 单个MySQL	支持	支持	支持	支持
单个MySQL 双向复制到 单个MySQL	支持; 同名库表的双向复制存在缺陷	支持; 同名库表的双向复制存在缺陷	支持; 同名库表的双向复制存在缺陷	支持
多个MySQL的表 合并到 单个MySQL	支持	支持	支持	支持
单个MySQL的不同表 分发 到 多个MySQL	支持	支持	支持	支持
单个MySQL的同一表的不同记录 分发 到 多个MySQL	支持按主键分发; 不支持按函数分发	支持按主键分发; 不支持按函数分发	支持按主键分发; 不支持按函数分发	支持
公有云间的数据同步	支持	支持	支持	支持
单个MySQL复制到Kafka	支持	不支持	支持	支

单个MySQL复制到Kafka	支持	不支持	支持	持
多个MySQL复制到Kafka	支持	不支持	支持	支 持

# 使用限制

## 限制

- 仅支持 MySQL 5.6/5.7 版本
- 仅支持 InnoDB 引擎
- 仅支持以下字符集:
  - latin1
  - latin2
  - gbk
  - utf8
  - utf8mb4
  - binary
- binlog 仅支持 row 模式
- binlog image 仅支持 FULL 模式
- 源端和目标端大小写敏感配置 (`lower_case_table_names`) 需保持一致
- 需要开启 GTID
- 不支持 Trigger
- 只支持 MySQL 认证方式 `mysql_native_password`, 不支持其他类型的 `default_authentication_plugin`

## 源端 MySQL 需配置如下参数

参数	值	检查方式
<code>log_bin</code>	ON	<code>show global variables like 'log_bin';</code>
<code>binlog_format</code>	ROW	<code>show global variables like 'binlog_format';</code>
<code>binlog_row_image</code>	FULL	<code>show global variables like 'binlog_row_image';</code>
<code>log_slave_updates</code>	ON	<code>show global variables like 'log_slave_updates';</code>
<code>gtid_mode</code>	ON	<code>show global variables like 'gtid_mode';</code>

## 端口使用说明

默认情况下, dtle的传输/通信会使用如下端口:

端口号	说明
8190	HTTP 服务的端口号
8191	服务端与客户端的通信端口
8192	服务端与服务端的通信端口
8193	数据传输的端口

## 如何修改

端口配置可在[dtle.conf](#)中修改

## 对目标端数据库的影响

### 表 dtle.gtid\_executed\_v2

当目标端是MySQL数据库时, dtle会在目标端自动创建表 `dtle.gtid_executed_v2`, 目标端的用于回放数据的数据库用户需要对这张表有[相应权限](#).

表 `dtle.gtid_executed_v2` 的作用是存储已经回放的事务的GTID, 用作断点续传/数据检查等.

使用表 `dtle.gtid_executed_v2` 模仿GTID机制, 而不使用MySQL原生GTID机制的原因是: 在回放时, `set GTID_NEXT=...` 语句需要 SUPER 权限, 而云环境下, 数据库用户可能无法拥有 SUPER 权限.

`dtle.gtid_executed_v2` 的建表语句如下:

```
CREATE TABLE IF NOT EXISTS actiontech_udup.gtid_executed_v2 (
    job_uuid binary(16) NOT NULL COMMENT 'unique identifier of job',
    source_uuid binary(16) NOT NULL COMMENT 'uuid of the source where the transaction was originally executed.',
    interval_gtid text NOT NULL COMMENT 'number of interval.'
);
```

表结构说明:

- `job_uuid`: 执行同步的任务编号
- `source_uuid`: 源端数据库UUID号
- `interval_gtid`: 同步数据的GTID值

## 监控项说明

类别	监控项
网络流量状态	-
-	network.in_msgs
-	network.out_msgs
-	network.in_bytes
-	network.out_bytes
缓存/队列状态	-
-	buffer.src_queue_size
-	buffer.dest_group_queue_size
-	buffer.dest_queue_size
-	buffer.send_by_timeout
-	buffer.send_by_size_full
表统计	-
-	table.insert
-	table.update
-	table.delete
延迟统计	-
-	delay.num
-	delay.time
吞吐统计	-
-	throughput.num
-	throughput.time
Nomad client 监控项 <a href="#">参考</a>	-
-	client.allocations.migrating.{nodeID}
-	client.allocations.blocked.{nodeID}
-	client.allocations.pending.{nodeID}
-	client.allocations.running.{nodeID}
-	client.allocations.terminal.{nodeID}
Nomad server 监控项 <a href="#">参考</a>	-
-	server.blocked_evals.total_blocked
-	server.blocked_evals.total_escaped

-	server.broker.total_ready
-	server.broker.total_unacked
-	server.broker.total_blocked
-	server.broker.total_waiting
-	server.broker.{sched}.ready
-	server.broker.{sched}.unacked
-	server.heartbeat.active
-	server.plan.queue_depth
-	server.rpc.accept_conn
-	server.rpc.raft_handoff
-	server.rpc.request_error
-	server.rpc.request
-	server.rpc.cross-region.{region}
-	server.rpc.query

## 安装步骤

### 基于容器使用

直接运行 `docker pull actiontech/dtle` 可使用最新版本的dtle, 使用方法参见 快速开始 一节  
容器的版本列表参看[docker hub](#)

### 基于rpm包的安装

[从此处](#) 下载dtle的 rpm 安装包, 并执行以下命令可安装dtle

```
rpm -ivh --prefix /opt/dtle dtle-xxx.rpm
```

配置文件位于 `/opt/dtle/etc/dtle/dtle.conf`

服务启动命令:

```
systemctl start dtle
```

日志文件位于 `/opt/dtle/var/log/dtle/dtle.log`

## 节点配置

节点即一个dtle进程，可运行 dtle manager 和/或 agent。配置文件一般为 `dtle.conf`。

参数类别	参数名	默认值	取值范围	说明
全局参数	data_dir	"/opt/dtle/data"	-	dtle运行数据的存放路径
全局参数	bind_addr	"0.0.0.0"	-	对外服务的ip地址，参考 <a href="#">说明</a>
全局参数	log_level	"INFO"	"DEBUG"/"INFO"/"WARN"/"ERR"	日志级别
全局参数	log_file	"/var/log/dtle/dtle.log"	-	日志路径
全局参数	log_to_stdout	false	true/false	是否将日志输出到标准流
端口参数(ports)	http	8190	-	http服务的端口号
端口参数(ports)	rpc	8191	-	服务端与客户端的通信端口
端口参数(ports)	serf	8192	-	服务端与服务端的通信端口
端口参数(ports)	nats	8193	-	数据传输的端口
服务端参数(manager)	enabled	true	true/false	本节点是否作为服务端
服务端参数(manager)	join	["127.0.0.1"]	可配置多值，格式为「"ip1", "ip2", "ip3", ...」	当节点作为服务端时，启动时加入的集群地址
服务端参数(manager)	bootstrap_expect	1	1/3/5	服务端节点集群的预期数量。集群启动时，在达成预期数量前，服务

				端对外不提供服务
服务端参数 (manager)	heartbeat_grace	"30s"	时间描述字符串	心跳检查的容忍时间, 时间内若心跳检查均失败, 则判定节点为异常状态, 将进行任务切换
服务端参数 (manager)	retry_max	3	-	节点启动时, 加入服务端集群的重试次数
服务端参数 (manager)	retry_interval	"15s"	时间描述字符串	节点启动时, 加入服务端集群的重试间隔
客户端参数 (agent)	enabled	true	true/false	本节点是否作为客户端
客户端参数 (agent)	managers	["127.0.0.1:8191"]	可配置多值, 格式为「 "ip1", "ip2", "ip3", ... ]	当节点作为客户端时, 启动时加入的服务端集群的地址
网络绑定地址 (addresses)	http	"0.0.0.0"	-	http服务的绑定地址
网络绑定地址 (addresses)	rpc	"0.0.0.0"	-	服务端与客户端的通信的绑定地址
网络绑定地址 (addresses)	serf	"0.0.0.0"	-	服务端与服务端的通信的绑定地址
外部访问地址 (advertise)	http	"0.0.0.0"	-	http服务的外部访问地址, 可用于配置NAT
外部访问地址	rpc	"0.0.0.0"	-	服务端与客户端的通信的外部访问地

(advertise)				址, 可用于配置NAT
外部访问地址 (advertise)	serf	"0.0.0.0"	-	服务端与服务端的通信的外部访问地址, 可用于配置NAT
网络配置 (network)	max_payload	"100MB"	-	网络传输的单个消息的大小上限
监控 (metric)	collection_interval	""	-	监控采集的周期. 若不配置, 则禁用监控采集.
监控 (metric)	publish_allocation_metrics	true	-	是否收集任务监控信息.
监控 (metric)	publish_node_metrics	true	-	是否收集节点监控信息.

## 命令说明

### **dtle server**

格式: `dtle server [options]`

作用: 启动dtle节点.

参数说明:

参数名	说明
<code>-config=</code>	可指定配置文件路径, 或配置文件目录路径

### **dtle job-status**

格式: `dtle job-status [options] <job-id>`

作用: 查看作业状态. 不指定 `job-id` 时, 将列出所有作业

参数说明:

参数名	说明
<code>-address=</code>	指定 服务端 地址
<code>-all-allocs</code>	列出作业下的所有执行任务

### **dtle members**

格式: `dtle members [options]`

作用: 查看服务端集群的节点状态

参数说明:

参数名	说明
<code>-address=</code>	指定 服务端 地址
<code>-detailed</code>	列出详细信息

### **dtle node-status**

格式: `dtle node-status [options] <node-id>`

作用: 查看节点状态. 不指定 `node-id` 时, 将列出所有节点

参数说明:


参数名	说明
-address=	指定 服务端 地址
-allocs	列出节点下的所有执行任务

## **dtle version**

格式: `dtle version`

作用: 查看dtle的版本, 与 `dtle --version` 作用相同

## 作业(job)配置

作业配置一般采用json文件, 有如下字段:

参数名	必填?	类型	源端/目标端可配	默认值	说明
ID	否	String	-	自动生成的UUID	作业ID. 若不指定, 则自动生成UUID
Name	是	String	-	-	作业名. 助记用, 可重
Failover	否	Bool	-	false	是否允许故障时进行业转移
Tasks	是	Array	-	-	作业中的任务集合. 元素类型为Task
Task.Type	是	String	-	-	任务的类型. 可取值: Src (从源数据库抽取 Dest (向目标数据库回放)
Task.Driver	否	String	-	"MySQL"	数据库的类型
Task.NodeId	否	String	-	随机选取执行节点	任务的执行节点的节ID (可通过 GET /nodes 获取). 若不指定则随机选取
Task.NodeName	否	String	-	随机选取执行节点	任务的执行节点的节名 (可通过 GET /nodes 获取). 若不指定则随机选取
Task.Config.Gtid	否	String	源端	默认为 全量+增量任务	MySQL的GTID点位, 取值: 1. 默认为空, 则为 <全量+增量> 复制任务 2. 正确的GTID, 则从定的GTID的下一个事开始增量复制 3. 若配置不存在的 GTID, 且SID是源端 MySQL, 则报错 4. 若配置不存在的 GTID, 且SID不是源端 MySQL, 则增量从源复制所有事务
Task.Config.			源		全量复制时, 在目标端删除参与复制的表, 之后由dtle自动创建表结构 (相关参数:

DropTableIfExists	否	Bool	端	false	SkipCreateDbTable 如果开启此选项, 目标端数据库用户需要有应表的 DROP 权限.
Task.Config.ApproveHeterogeneous	否	Bool	源端	true	增量复制时, 是否将数据转为SQL执行. 以下场景需设置为 true 1. 目标端不是MySQL 2. 目标MySQL没有super权限
Task.Config.SkipCreateDbTable	否	Bool	源端	false	不为目标库创建复制和复制表. 如果关闭此选项, 目标端数据库用户需要有相应表的 CREATE 权限.
Task.Config.ParallelWorkers	否	Int	目标端	1	回放端的并发数. 当值大于1, 且源端MySQL支持 MTS时, 目标端进行并行回放
Task.Config.ReplChanBufferSize	否	Int	源端	600	复制任务缓存的大小 单位为事务数
Task.Config.ChunkSize	否	Int	源端	2000	全量复制时, 每次读取传输-写入的行数
Task.Config.ExpandSyntaxSupport	否	Bool	源端	false	支持复制 用户权限/存储过程DDL/函数DDL
Task.Config.MsgBytesLimit	否	Int	源端	20480	单个消息大小限制, 单位是字节
Task.Config.MsgsLimit	否	Int	源端	65536	消息数量限制
Task.Config.BytesLimit	否	Int	源端	67108864	消息大小限制
Task.Config.ReplicateDoDb	否	Array	源端	-	如为空, 则复制整个数据库实例. 可填写多值
Task.Config.ReplicateDoDb.TableSchema	否	String	源端	-	数据库名
Task.Config.ReplicateDoDb.Tables	否	Array	源端	-	可配置多张表, 类型为Table. 若不配置, 则复制指定数据库中的所有表
Task.Config.ReplicateDoDb.Table.TableName	否	String	源端	-	表名
Task.Config.ReplicateDoDb.Table.Where	否	String	源端	-	只复制满足该条件的数据行. 语法为SQL表达式, 返回值应为布尔值 可以引用表中的列名
Task.Config.ConnectionConfig.Host	是	String	两端	-	数据源地址

Task.Config.ConnectionConfig.Port	是	String	两端	-	数据源端口
Task.Config.ConnectionConfig.User	是	String	两端	-	数据源用户名
Task.Config.ConnectionConfig.Password	是	String	两端	-	数据源密码
Task.Config.ConnectionConfig.Charset	否	String	两端	"utf8"	数据源的字符集
Task.Config.GroupMaxSize	否	int	源端	1	源端发送数据时, 等待数据包达到一定大小 ( GroupMaxSize 字节后发送该包. 单位为字节. 默认值1表示即刻送数据)
Task.Config.GroupTimeout	否	int	源端	100	源端发送数据时, 等待数据包达到超时时间 ( GroupTimeout 毫秒发送该包. 单位为毫秒)

## HTTP API 说明

### 列出所有作业

API: GET /v1/jobs

样例:

```
> curl -XGET "172.17.5.6:8190/v1/jobs" | jq
< [
  {
    "CreateIndex": 59,
    "ID": "8ce4b408-8c64-41b9-04d7-baf451348e89",
    "JobModifyIndex": 469,
    "JobSummary": {
      "Constraints": null,
      "CreateIndex": 59,
      "Datacenters": [
        "dc1"
      ],
      "EnforceIndex": false,
      "Failover": false,
      "ID": "8ce4b408-8c64-41b9-04d7-baf451348e89",
      "JobModifyIndex": 469,
      "ModifyIndex": 469,
      "Name": "test1-2",
      "Orders": [],
      "Region": "global",
      "Status": "running",
      "StatusDescription": "",
      "Tasks": [
        {
          "Config": {
            "ExpandSyntaxSupport": false,
            "NatsAddr": "127.0.0.1:8193",
            "RepChanBufferSize": "600",
            "ReplicateDoDb": [
              {
                "TableSchema": "db1",
                "Tables": [
                  {
                    "TableName": "tb1"
                  }
                ]
              }
            ],
            "ChunkSize": "2000",
            "ApproveHeterogeneous": true,
            "ReplicateFromDb": [
              {
                "TableSchema": "db1",
                "Tables": [
                  {
                    "TableName": "tb1"
                  }
                ]
              }
            ],
            "ReplicateIndex": 59
          }
        }
      ]
    }
  }
]
```

```
        "DropTableIfExists": false,
        "ConnectionConfig": {
            "Password": "*",
            "Host": "172.100.9.3",
            "Port": "3306",
            "User": "lx1"
        },
        "TrafficAgainstLimits": 0,
        "Gtid": "8868d98f-af5e-11e8-9aa9-0242ac110002:1-171",
        "SkipCreateDbTable": false
    },
    "ConfigLock": {},
    "Constraints": null,
    "Driver": "MySQL",
    "Leader": false,
    "NodeID": "",
    "NodeName": "dtle",
    "Type": "Src"
},
{
    "Config": {
        "NatsAddr": "127.0.0.1:8193",
        "Gtid": "8868d98f-af5e-11e8-9aa9-0242ac110002:1-171",
        "SkipCreateDbTable": false,
        "DropTableIfExists": false,
        "ExpandSyntaxSupport": false,
        "ReplChanBufferSize": "600",
        "ApproveHeterogeneous": true,
        "ConnectionConfig": {
            "User": "test1",
            "Password": "*",
            "Host": "172.100.9.6",
            "Port": "3306"
        }
    },
    "ConfigLock": {},
    "Constraints": null,
    "Driver": "MySQL",
    "Leader": true,
    "NodeID": "",
    "NodeName": "dtle",
    "Type": "Dest"
}
],
    "Type": "synchronous"
},
"ModifyIndex": 469,
"Name": "test1-2",
>Status": "running",
>StatusDescription": "",
>Type": "synchronous"
```

```

},
{
  "CreateIndex": 66,
  "ID": "ae2a74ac-1ee7-7f82-4b52-2f1ca759eab5",
  "JobModifyIndex": 469,
  "JobSummary": {
    "Constraints": null,
    "CreateIndex": 66,
    "Datacenters": [
      "dc1"
    ],
    "EnforceIndex": false,
    "Failover": false,
    "ID": "ae2a74ac-1ee7-7f82-4b52-2f1ca759eab5",
    "JobModifyIndex": 469,
    "ModifyIndex": 469,
    "Name": "test1-2",
    "Orders": [],
    "Region": "global",
    "Status": "running",
    "StatusDescription": "",
    "Tasks": [
      {
        "Config": {
          "ConnectionConfig": {
            "Host": "172.100.9.3",
            "Port": "3306",
            "User": "lx1",
            "Password": "*"
          },
          "TrafficAgainstLimits": 0,
          "NatsAddr": "127.0.0.1:8193",
          "ReplicateDoDb": [
            {
              "Tables": [
                {
                  "TableName": "tb1"
                }
              ],
              "TableSchema": "db1"
            }
          ],
          "DropTableIfExists": false,
          "ExpandSyntaxSupport": false,
          "ChunkSize": "2000",
          "SkipCreateDbTable": false,
          "Gtid": "8868d98f-af5e-11e8-9aa9-0242ac110002:1-171",
          "ApproveHeterogeneous": true,
          "ReplChanBufferSize": "600"
        },
        "ConfigLock": {}
      }
    ]
  }
}

```

```

    "Constraints": null,
    "Driver": "MySQL",
    "Leader": false,
    "NodeID": "",
    "NodeName": "dtle",
    "Type": "Src"
},
{
    "Config": {
        "DropTableIfExists": false,
        "ExpandSyntaxSupport": false,
        "NatsAddr": "127.0.0.1:8193",
        "Rep1ChanBufferSize": "600",
        "ApproveHeterogeneous": true,
        "ConnectionConfig": {
            "Port": "3306",
            "User": "test1",
            "Password": "*",
            "Host": "172.100.9.6"
        },
        "Gtid": "8868d98f-af5e-11e8-9aa9-0242ac110002:1-171",
        "SkipCreateDbTable": false
    },
    "ConfigLock": {},
    "Constraints": null,
    "Driver": "MySQL",
    "Leader": true,
    "NodeID": "",
    "NodeName": "dtle",
    "Type": "Dest"
}
],
{
    "Type": "synchronous"
},
{
    "ModifyIndex": 469,
    "Name": "test1-2",
    "Status": "running",
    "StatusDescription": "",
    "Type": "synchronous"
}
]

```

## 获取某个作业的信息

API: GET /v1/job/{ID}

样例:

```
> curl -XGET "172.17.5.6:8190/v1/job/8ce4b408-8c64-41b9-04d7-baf451348e89" | jq
```

```

< {
  "Constraints": null,
  "CreateIndex": 59,
  "Datacenters": [
    "dc1"
  ],
  "EnforceIndex": false,
  "Failover": false,
  "ID": "8ce4b408-8c64-41b9-04d7-baf451348e89",
  "JobModifyIndex": 471,
  "ModifyIndex": 471,
  "Name": "test1-2",
  "Orders": [],
  "Region": "global",
  "Status": "running",
  "StatusDescription": "",
  "Tasks": [
    {
      "Config": {
        "ConnectionConfig": {
          "Host": "172.100.9.3",
          "Port": "3306",
          "User": "lx1",
          "Password": "111111"
        },
        "ChunkSize": "2000",
        "ApproveHeterogeneous": true,
        "DropTableIfExists": false,
        "ReplChanBufferSize": "600",
        "ReplicateDoDb": [
          {
            "TableSchema": "db1",
            "Tables": [
              {
                "TableName": "tb1"
              }
            ]
          }
        ],
        "NatsAddr": "127.0.0.1:8193",
        "TrafficAgainstLimits": 0,
        "Gtid": "8868d98f-af5e-11e8-9aa9-0242ac110002:1-171",
        "SkipCreateDbTable": false,
        "ExpandSyntaxSupport": false
      },
      "ConfigLock": {},
      "Constraints": null,
      "Driver": "MySQL",
      "Leader": false,
      "NodeID": "",
      "NodeName": "dtle",
    }
  ]
}

```

```

    "Type": "Src"
},
{
  "Config": {
    "Gtid": "8868d98f-af5e-11e8-9aa9-0242ac110002:1-171",
    "SkipCreateDbTable": false,
    "DropTableIfExists": false,
    "ExpandSyntaxSupport": false,
    "ReplChanBufferSize": "600",
    "ApproveHeterogeneous": true,
    "ConnectionConfig": {
      "Host": "172.100.9.6",
      "Port": "3306",
      "User": "test1",
      "Password": "111111"
    },
    "NatsAddr": "127.0.0.1:8193"
  },
  "ConfigLock": {},
  "Constraints": null,
  "Driver": "MySQL",
  "Leader": true,
  "NodeID": "",
  "NodeName": "dtle",
  "Type": "Dest"
}
],
"Type": "synchronous"
}

```

## 获取服务端集群的leader节点信息

API: GET /v1/leader

样例:

```

> curl -XGET "172.17.5.6:8190/v1/leader"
< "172.17.5.6:8191"

```

## 列出服务端集群的节点

API: GET /v1/members

样例:

```

> curl -XGET "172.17.5.6:8190/v1/members" | jq
< [
  "Members": [

```

```
{
    "Addr": "172.17.5.6",
    "DelegateCur": 4,
    "DelegateMax": 5,
    "DelegateMin": 2,
    "Name": "udp-6.global",
    "Port": 8192,
    "ProtocolCur": 2,
    "ProtocolMax": 5,
    "ProtocolMin": 1,
    "Status": "alive",
    "Tags": {
        "port": "8191",
        "bootstrap": "1",
        "role": "server",
        "region": "global",
        "dc": "dc1",
        "build": "9.9.9.9"
    }
},
],
"ServerDC": "dc1",
"ServerName": "udp-6",
"ServerRegion": "global"
}
```

## 列出所有节点

API: GET /v1/nodes

样例:

```
> curl -XGET "172.17.5.6:8190/v1/nodes" | jq
< [
    {
        "CreateIndex": 4,
        "Datacenter": "dc1",
        "HTTPAddr": "172.17.5.6:8190",
        "ID": "8f05de6f-34ef-5989-9a20-c9f8bc9817b0",
        "ModifyIndex": 43,
        "Name": "udp-6",
        "Status": "ready",
        "StatusDescription": ""
    }
]
```

## 列出某作业的所有任务执行

API: GET /job/< jobId >/allocations

样例:

```
> curl -XGET "172.17.5.6:8190/v1/job/8ce4b408-8c64-41b9-04d7-baf451348e89/allocations" | jq
< [
  {
    "ClientDescription": "",
    "ClientStatus": "running",
    "CreateIndex": 61,
    "CreateTime": 1538993511593515300,
    "DesiredDescription": "",
    "DesiredStatus": "run",
    "EvalID": "f67f2020-a326-6d12-2239-6da8f9dd3e4e",
    "ID": "33a7a1bb-8eb4-11a2-bd2c-0c7e0457dfc2",
    "JobID": "8ce4b408-8c64-41b9-04d7-baf451348e89",
    "ModifyIndex": 63,
    "Name": "test1-2.Src",
    "NodeID": "b32743aa-5c69-0853-239c-a69237e97c45",
    "Task": "Src",
    "TaskStates": {
      "Src": {
        "Events": [
          {
            "DiskLimit": 0,
            "DriverError": "",
            "DriverMessage": "",
            "ExitCode": 0,
            "FailedSibling": "",
            "FailsTask": false,
            "KillError": "",
            "KillReason": "",
            "KillTimeout": 0,
            "Message": "",
            "RestartReason": "",
            "SetupError": "",
            "StartDelay": 0,
            "TaskSignal": "",
            "TaskSignalReason": "",
            "Time": "2018-10-08T10:11:51.603015093Z",
            "Type": "Received"
          },
          {
            "DiskLimit": 0,
            "DriverError": "",
            "DriverMessage": "",
            "ExitCode": 0,
            "FailedSibling": "",
            "FailsTask": false,
            "KillError": "",
            "KillReason": "",
            "KillTimeout": 0,
            "Message": "",
            "RestartReason": "",
            "SetupError": "",
            "StartDelay": 0,
            "TaskSignal": "",
            "TaskSignalReason": "",
            "Time": "2018-10-08T10:11:51.603015093Z",
            "Type": "Received"
          }
        ]
      }
    }
  }
]
```

```

        "KillError": "",
        "KillReason": "",
        "KillTimeout": 0,
        "Message": "",
        "RestartReason": "",
        "SetupError": "",
        "StartDelay": 0,
        "TaskSignal": "",
        "TaskSignalReason": "",
        "Time": "2018-10-08T10:11:51.603322601Z",
        "Type": "Started"
    }
],
"Failed": false,
"FinishedAt": null,
"StartedAt": "2018-10-08T10:11:51.603998869Z",
"State": "running"
}
}
},
{
"ClientDescription": "",
"ClientStatus": "running",
"CreateIndex": 61,
"CreateTime": 1538993511593515300,
"DesiredDescription": "",
"DesiredStatus": "run",
"EvalID": "f67f2020-a326-6d12-2239-6da8f9dd3e4e",
"ID": "83a46402-2c98-ffa1-b953-1cc038d91a7c",
"JobID": "8ce4b408-8c64-41b9-04d7-baf451348e89",
"ModifyIndex": 63,
"Name": "test1-2.Dest",
"NodeID": "b32743aa-5c69-0853-239c-a69237e97c45",
"Task": "Dest",
"TaskStates": {
    "Dest": {
        "Events": [
            {
                "DiskLimit": 0,
                "DriverError": "",
                "DriverMessage": "",
                "ExitCode": 0,
                "FailedSibling": "",
                "FailsTask": false,
                "KillError": "",
                "KillReason": "",
                "KillTimeout": 0,
                "Message": "",
                "RestartReason": "",
                "SetupError": "",
                "StartDelay": 0,

```

```

    "TaskSignal": "",
    "TaskSignalReason": "",
    "Time": "2018-10-08T10:11:51.603156346Z",
    "Type": "Received"
  },
  {
    "DiskLimit": 0,
    "DriverError": "",
    "DriverMessage": "",
    "ExitCode": 0,
    "FailedSibling": "",
    "FailsTask": false,
    "KillError": "",
    "KillReason": "",
    "KillTimeout": 0,
    "Message": "",
    "RestartReason": "",
    "SetupError": "",
    "StartDelay": 0,
    "TaskSignal": "",
    "TaskSignalReason": "",
    "Time": "2018-10-08T10:11:51.603388746Z",
    "Type": "Started"
  }
],
"Failed": false,
"FinishedAt": null,
"StartedAt": "2018-10-08T10:11:51.603690587Z",
"State": "running"
}
]

```

## 查看某个任务执行的状态

API: GET /agent/allocation/<allocId>/stats

样例:

```

> curl -XGET "172.17.5.6:8190/v1/agent/allocation/33a7a1bb-8eb4-11a2-bd2c-0c7e0457d
fc2/stats" | jq
< {
  "Tasks": [
    "Src": {
      "Backlog": "0/600",
      "BufferStat": {
        "ApplierGroupTxQueueSize": 0,
        "ApplierTxQueueSize": 0,
        "TxQueueSize": 0
      }
    }
  ]
}

```

```

    "ExtractorTxQueueSize": 0,
    "SendBySizeFull": 0,
    "SendByTimeout": 0
},
"CurrentCoordinates": {
    "ExecutedGtidSet": "",
    "File": "1.000002",
    "GtidSet": "8868d98f-af5e-11e8-9aa9-0242ac110002:171",
    "Position": 44918,
    "ReadMasterLogPos": 0,
    "RelayMasterLogFile": "",
    "RetrievedGtidSet": ""
},
"DelayCount": null,
"ETA": "0s",
"ExecMasterRowCount": 14,
"ExecMasterTxCount": 111939,
"MsgStat": {
    "InBytes": 0,
    "InMsgs": 4,
    "OutBytes": 3413,
    "OutMsgs": 4,
    "Reconnects": 0
},
"ProgressPct": "100.0",
"ReadMasterRowCount": 14,
"ReadMasterTxCount": 111939,
"Stage": "Master has sent all binlog to slave; waiting for more updates",
"TableStats": null,
"ThroughputStat": null,
"Timestamp": 1539004705449136600
}
}
}
}

```

## 创建/更新一个作业

API: POST /v1/jobs

样例: job1.json的内容说明参看[作业\(job\)配置](#)

```
> curl -H "Accept:application/json" -XPOST "172.17.5.6:8190/v1/jobs" -d @job1.json
```

## 删除一个作业

API: POST /v1/job/{ID}

样例:

```
> curl -H "Accept:application/json" -XDELETE "172.17.5.6:8190/v1/job/8ce4b408-8c64-41b9-04d7-baf451348e89"
```

## 暂停一个作业

API: POST /v1/job/{ID}/pause

样例:

```
> curl -H "Accept:application/json" -XPOST "172.17.5.6:8190/v1/job/8ce4b408-8c64-41b9-04d7-baf451348e89/pause"
```

## 继续一个作业

API: POST /v1/job/{ID}/resume

样例:

```
> curl -H "Accept:application/json" -XPOST "172.17.5.6:8190/v1/job/8ce4b408-8c64-41b9-04d7-baf451348e89/resume"
```

# MySQL 用户权限说明

dtle配置的MySQL用户, 在使用不同功能时, 需具有以下权限

## 源端用户

权限	功能说明
select	全量复制时, 对目标表需要 select 权限
replication client	全量/增量复制时, 需执行 show master status 获取binlog信息
replication slave	增量复制时, 需要模拟 MySQL 复制

## 目标端用户

权限	功能说明
alter	复制时处理DDL语句
create	复制时处理DDL语句; 自动创建表结构功能; 自动创建目标端的GTID元数据表
drop	复制时处理DDL语句
index	复制时处理DDL语句
reference	复制时处理DDL语句
insert	复制时处理DML语句; 修改目标端的GTID元数据表
delete	复制时处理DML语句; 修改目标端的GTID元数据表
update	复制时处理DML语句
select	查询目标端的GTID元数据表
trigger	进行目标端触发器检查

## 时间/资源估算

### ETA (预计完成时间) 估算

#### 源端

- 全量过程, 公式为:

```
总时间 = 已用时间 / 发送到目标端的行数 * 总行数
其中, 总行数 = (select count(*) ...)
预计完成时间 = 总时间 - 已用时间
即: 预计完成时间 = 剩余行数 / 当前发送速率
```

- 增量过程, ETA 一直为 0s

#### 目标端

- 全量过程. 公式为:

```
总时间 = 已用时间 / 已写入目标端的行数 * 总行数
预计完成时间 = 总时间 - 已用时间
即: 预计完成时间 = 剩余行数 / 当前写入速率
```

- 增量过程, ETA 一直为 0s

## 内存占用估算

```
内存占用估算 = RowSize * ChunkSize * QueueSize * 内存占用系数
```

其中:

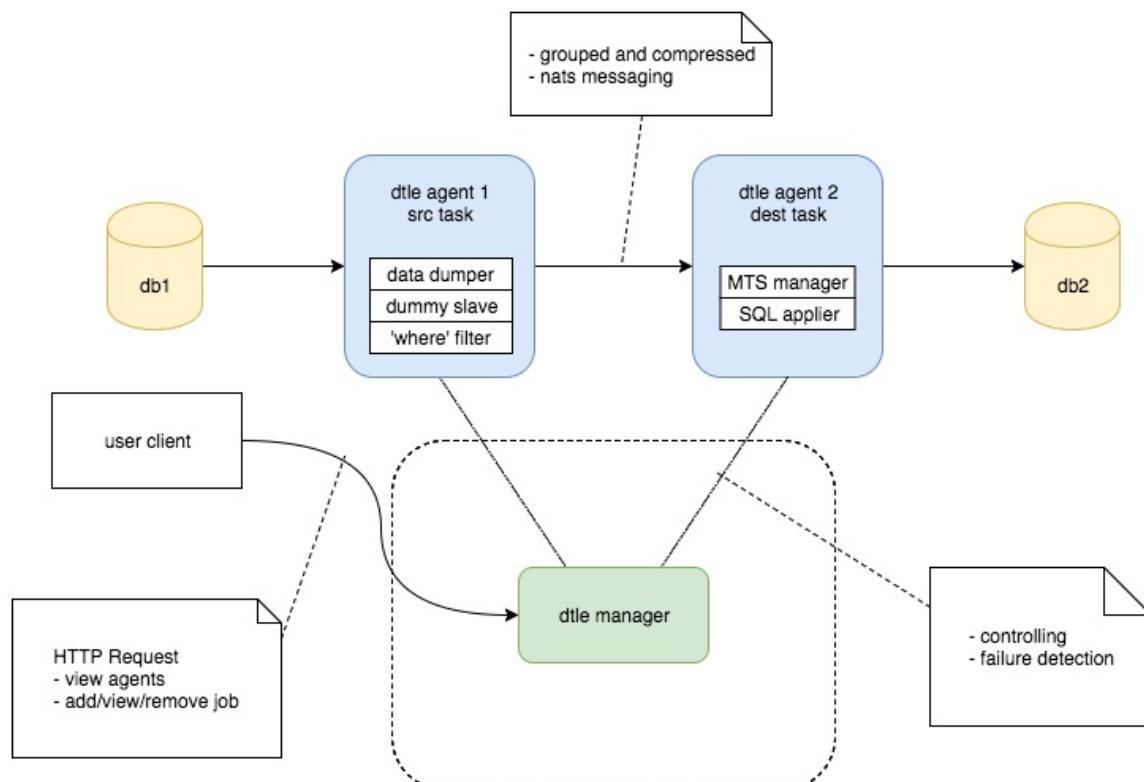
- RowSize为数据行的平均大小 (字节)
- ChunkSize为[配置项](#)
- QueueSize为传输队列长度, 硬编码为24
- 内存占用系数 测量约为 常量3.2

## dtle 架构

dtle角色分为manager、agent.

- manager数量应为1、3或5个
- agent数量不限
- 至少需要1个manager和1个agent
- 一个dtle进程可同时扮演manager和agent

任务分为源端任务和目标端任务, 各由agent执行. 通过网络压缩传输数据.



# Kafka 消息格式

dtile Kafka 输出, 消息格式兼容 Debezium

其消息格式具体可参考 <https://debezium.io/docs/tutorial/#viewing-the-change-events>

此处概要说明

- 每行数据变更会有一个消息
- 每个消息分为key和value
  - key是该次变更的主键
  - value是该次变更的整行数据
- key和value各自又有schema和payload
  - payload是具体的数据
  - schema指明了数据的格式, 即payload的解读方式, 可以理解为“类定义”
    - 注意和SQL schema含义不同
    - 表结构会包含在 Kafka Connect schema 中

## Key

以下是一个消息的key. 只是简单的包含了主键.

```
{
  "schema": {
    "type": "struct",
    "name": "dbserver1.inventory.customers.Key"
    "optional": false,
    "fields": [
      {
        "field": "id",
        "type": "int32",
        "optional": false
      }
    ]
  },
  "payload": {
    "id": 1004
  }
}
```

## Value

以下是一个消息的value, 其类型为 `topic.schema.table.Envelope`, 拥有5个字段

- `before` , 复杂类型 `topic.schema.table.Value` , 为该表的表结构.
- `after` , 复杂类型, 同上

- `source` , 复杂类型, 为该次变更的元数据
- `op : string` . 用"i", "d", "u" 分别表达操作类型: 增、删、改
- `ts_ms : int64` . 表示处理该行变更的时间.

```
{
  "schema": {
    "type": "struct",
    "fields": [
      {
        "type": "struct",
        "fields": [
          {
            "type": "int32",
            "optional": false,
            "field": "id"
          },
          {
            "type": "string",
            "optional": false,
            "field": "first_name"
          },
          {
            "type": "string",
            "optional": false,
            "field": "last_name"
          },
          {
            "type": "string",
            "optional": false,
            "field": "email"
          }
        ],
        "optional": true,
        "name": "dbserver1.inventory.customers.Value",
        "field": "before"
      },
      {
        "type": "struct",
        "fields": [
          {
            "type": "int32",
            "optional": false,
            "field": "id"
          },
          {
            "type": "string",
            "optional": false,
            "field": "first_name"
          }
        ]
      }
    ]
  }
}
```

```
        "type": "string",
        "optional": false,
        "field": "last_name"
    },
    {
        "type": "string",
        "optional": false,
        "field": "email"
    }
],
"optional": true,
"name": "dbserver1.inventory.customers.Value",
"field": "after"
},
{
    "type": "struct",
    "fields": [
        {
            "type": "string",
            "optional": true,
            "field": "version"
        },
        {
            "type": "string",
            "optional": false,
            "field": "name"
        },
        {
            "type": "int64",
            "optional": false,
            "field": "server_id"
        },
        {
            "type": "int64",
            "optional": false,
            "field": "ts_sec"
        },
        {
            "type": "string",
            "optional": true,
            "field": "gtid"
        },
        {
            "type": "string",
            "optional": false,
            "field": "file"
        },
        {
            "type": "int64",
            "optional": false,
            "field": "pos"
        }
    ]
}
```

```
        },
        {
            "type": "int32",
            "optional": false,
            "field": "row"
        },
        {
            "type": "boolean",
            "optional": true,
            "field": "snapshot"
        },
        {
            "type": "int64",
            "optional": true,
            "field": "thread"
        },
        {
            "type": "string",
            "optional": true,
            "field": "db"
        },
        {
            "type": "string",
            "optional": true,
            "field": "table"
        }
    ],
    "optional": false,
    "name": "io.debezium.connector.mysql.Source",
    "field": "source"
},
{
    "type": "string",
    "optional": false,
    "field": "op"
},
{
    "type": "int64",
    "optional": true,
    "field": "ts_ms"
}
],
"optional": false,
"name": "dbserver1.inventory.customers.Envelope",
"version": 1
},
"payload": {
    "before": null,
    "after": {
        "id": 1004,
        "first_name": "Anne",
```

```
    "last_name": "Kretchmar",
    "email": "annek@noanswer.org"
},
"source": {
    "version": "0.8.3.Final",
    "name": "dbserver1",
    "server_id": 0,
    "ts_sec": 0,
    "gtid": null,
    "file": "mysql-bin.000003",
    "pos": 154,
    "row": 0,
    "snapshot": true,
    "thread": null,
    "db": "inventory",
    "table": "customers"
},
"op": "c",
"ts_ms": 1486500577691
}
}
```

## MySQL数据类型到“Kafka Connect schema types”的转换

见 <https://debezium.io/docs/connectors/mysql/#data-types>

## 如何参与

### 提交缺陷

可直接在[github issues页面](#) 新建 issue, 选择 `Bug Report` 模板, 按格式填写完成后提交即可

### 提交功能

可直接在[github issues页面](#) 新建 issue, 选择 `Feature request` 模板, 按格式填写完成后提交即可

### 提交代码

按照github的[pull request流程](#)即可

## 如何全职参与

本项目的维护方([上海爱可生信息技术股份有限公司](#))一直在招聘 靠谱的研发工程师/靠谱的测试工程师.  
如果通过dtile, 您对全职参与类似的项目有兴趣, 请联系[我们的研发团队](#).

## 路线图

- 支持更多种类的公有云间的数据迁移
- WHERE 过滤条件 支持更丰富的函数 (目前仅支持关系符和简单函数)
- 对于 MySQL 分布式中间件 (如 dble) 提供数据扩容方案
- 对链路提供限流参数
- 提供告警功能
- 复制到 Kafka 的数据格式支持 Avro
- 表名变换/列名变换
- 数据变换
- 线路加密
- 免一致性快照事务的全量复制
  - 全量复制也可断点续传
- 一致性 DDL 元数据