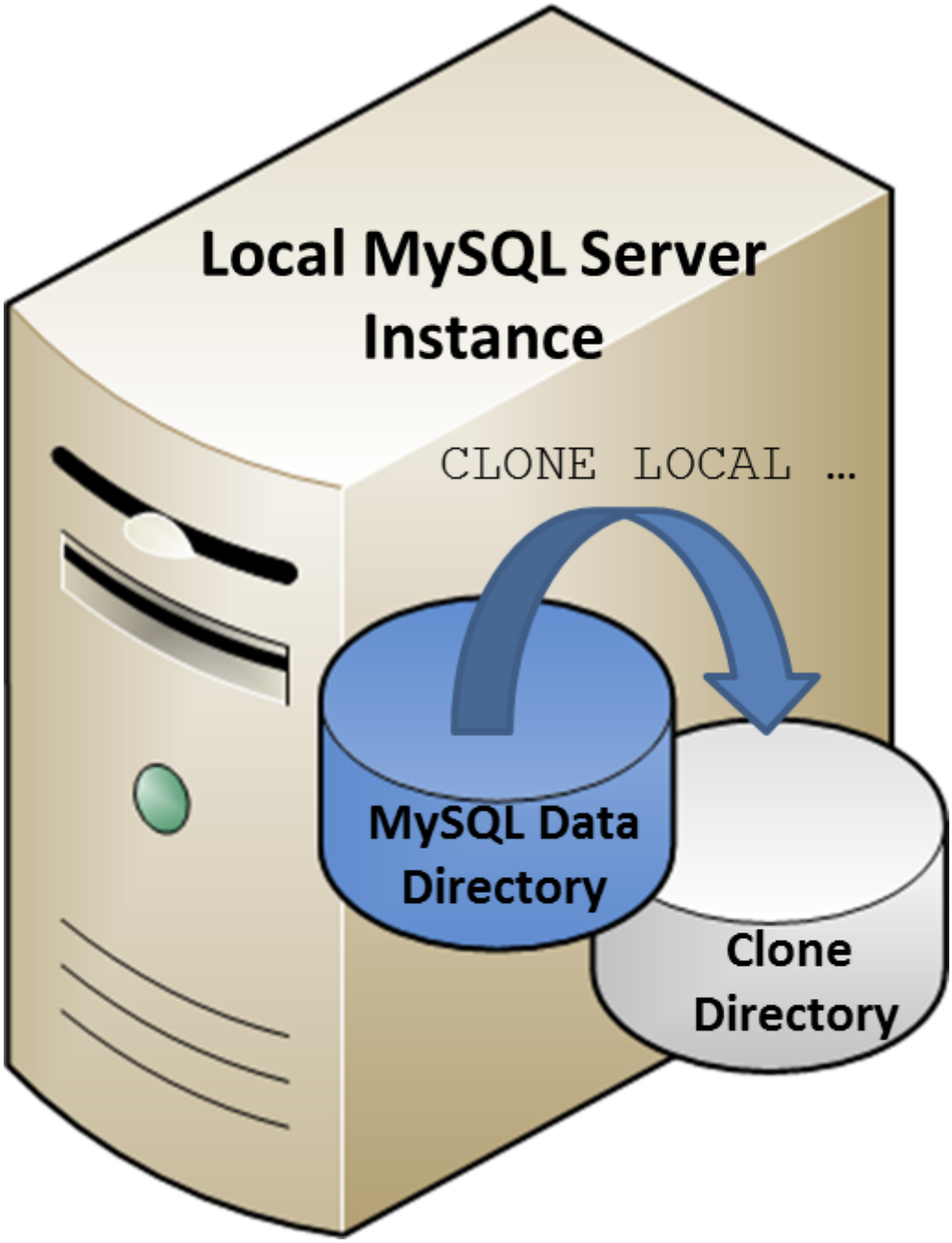
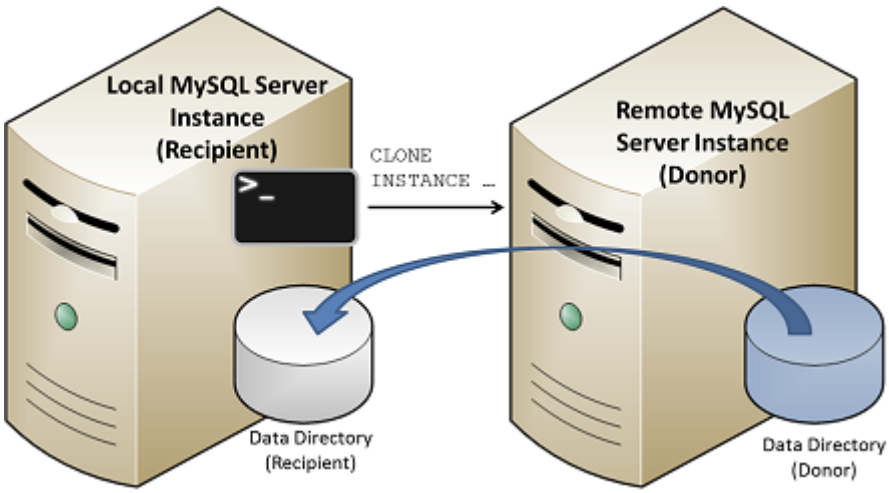
第二十二节：使用clone plugin搭建复制拓补---实战篇

1. 克隆插件简介

- 克隆插件允许从本地或远程的MySQL Server中克隆数据。克隆的数据是存储在InnoDB中的schema（database）、table（表）、tablespaces（表空间）和data dictionary metadata（数据字典元数据）的物理快照。该物理快照实际上是一个功能完整的数据目录，MySQL克隆插件可以使用该数据目录来配置并恢复一个MySQL Server。
- 本地克隆：指的是将数据从启动克隆操作的MySQL Server克隆到该MySQL Server的主机上的一个指定目录下。如下是本地克隆操作原理图：



远程克隆：涉及到启动克隆操作的本地MySQL Server（称为"recipient"，即，数据的接收者或接收方）和数据源所在的远程MySQL Server（称为"donor"，即，数据的提供者或发送方），在接收方上启动远程克隆操作时，克隆的数据会通过网络从发送方传输到接收方。默认情况下，远程克隆操作会删除接收方数据目录中的所有数据，并将其替换为克隆的新数据。如果不希望接收方中的现有数据被删除，也可以在接收方中执行克隆操作时将克隆数据指定存放在其他目录中。下图是远程克隆操作原理图：



- 对于克隆的数据本身来说，本地克隆操作与远程克隆操作没有太大区别。
- 克隆插件支持在复制拓扑中使用。除了克隆数据外，克隆操作还能够从发送方中提取和传输复制位点（二进制日志的位置），并将其应用于接收方，也就是说，我们可以使用克隆插件来在组复制中添加新的组成员，也可以在主从复制拓扑中添加新的从库。与通过二进制日志来复制大量事务相比，通过克隆插件要快得多，效率也更高。组复制成员还可以配置使用克隆插件来作为另一种恢复方法（如果不使用克隆插件，则必须使用基于二进制日志的状态传输进行数据恢复），当组成员和待加入组的Server都配置支持克隆插件时，待加入组的Server可以自行决定选择一个更加高效的方式从种子节点中获取数据：
- 克隆插件支持克隆数据加密和数据页压缩
- 要使用克隆功能，必须先安装克隆插件
- performance_schema中提供了用于监控克隆操作的一些性能事件采集器
- PS：在组复制拓扑中使用远程克隆操作时，为便于与非组复制拓扑做区分，我们这里也可以将"recipient"（即，接收方）称为"joiner"（即，加入方，表示将要加入组复制拓扑的MySQL Server）

本地克隆与远程克隆的语句用法示例：

```
# 本地克隆，使用克隆语句带上LOCAL关键字，并指定本地MySQL Server的数据存放目录，如果不指定存目录，则不允许执行本地克隆操作
CLONE LOCAL DATA DIRECTORY [=] 'clone_dir';
# 远程克隆，使用克隆语句带上INSTANCE关键字，并指定远程MySQL Server的连接信息，以及在本地的数据存放目录，如果不指定存放目录，则会使用远程MySQL Server的数据覆盖本地MySQL Server的数据
CLONE INSTANCE FROM 'user'@'host':port IDENTIFIED BY 'password' [DATA DIRECTORY [=] 'clone_dir'];
```

2.利用克隆插件快速搭建主从复制拓扑

- 假设有用于搭建主从复制的三台服务器，且在三台服务器中都各自初始化安装好了MySQL数据库，但没有配置主从复制拓扑，如下：
- - master：10.10.30.11
  - shadow：10.10.30.12
  - slave：10.10.30.13
- 其中，shadow节点的数据快照将利用克隆插件从master中获取，slave节点的数据快照将利用克隆插件从从shadow中获取。接下来，我们将分别介绍shadow节点和slave节点配置复制信息并加入复制拓扑的详细过程：

3.安装克隆插件

- 要使用克隆插件，必须要先安装克隆插件，由于对于远程克隆操作，克隆插件在发送方和接收方的MySQL Server上都必须安装（在本示例中，master、shadow和slave使用的是不同的服务器，因此，最方便快捷的方式是利用克隆插件执行远程克隆操作，而不是执行本地克隆操作之后，再进行数据拷贝）
- 要使插件可被Server使用，插件库文件必须位于MySQL Server程序目录下的plubin目录中（由系统变量plugin_dir指定的目录）。如果需要修改默认值，则需要在MySQL Server启动时通过系统变量plugin_dir进行指定新的位置
- 插件库的基本名是mysql_clone.so。文件名后缀因平台而异（例如，对于Unix和类Unix系统下库文件名后缀为.so，Windows系统下库文件名后缀为.dll)。
- 要在MySQL Server启动时加载插件，可以使用--plugin-load-add选项来指定需要加载的库文件名。使用这种插件加载方法，每次MySQL Server启动之前都必须设置好该选项。例如，将其写入my.cnf配置文件中（根据平台的不同调整库文件名后缀为.so或者.dll）：

```
[mysqld]
plugin-load-add=mysql_clone.so
```

- 编辑好my.cnf之后，重新启动MySQL Server以使新设置生效。请注意：在从旧版本升级到新版本过程中的重启操作，不能使用--plugin-load-add选项来加载克隆插件。例如，在将MySQL 5.7升级到MySQL 8.0的过程中，仅仅替换完成二进制程序文件之后，但还未完成其他升级步骤之前，尝试使用plugin-load-add=mysql_clone.so 重新启动MySQL Server。会导致报错：

  ```
  [ERROR] [MY-013238] [Server] Error installing plugin 'clone': Cannot install
  during upgrade。
  ```

- 在尝试使用--plugin-load-add=mysql_clone .so选项启动MySQL Server之前完成所有的升级操作可以避免该报错。
- 或者后续在需要时动态加载插件，可以使用以下语句（根据需要调整.so后缀）：

```
mysql> INSTALL PLUGIN clone SONAME 'mysql_clone.so';
```

- INSTALL PLUGIN语句可以加载插件，并将其注册到mysql系统库下的mysql.plugins表中，这样在后续重启MySQL Server时不需要重复使用--plugin-load-add选项来加载插件库。
- 要验证插件是否安装成功，可以查看INFORMATION_SCHEMA.plugins表或者使用SHOW PLUGINS语句查看。例如:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS FROM INFORMATION_SCHEMA.PLUGINS WHERE
PLUGIN_NAME = 'clone';
+-----------------------+---------------+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----------------------+---------------+
| clone | ACTIVE |
+-----------------------+---------------+
```

- 如果插件初始化失败，请检查MySQL错误日志以获取克隆或插件相关的错误日志。
- 如果插件之前已经通过INSTALL PLUGIIN语句或者--plugin-load-add选项成功注册过了，则可以在MySQL Server启动时使用--clone选项来控制克隆插件的激活状态。例如，要在启动时加载插件并防止它在运行时被删除，可以使用以下选项:

```
[mysqld]
plugin-load-add=mysql_clone.so
clone=FORCE_PLUS_PERMANENT
# 如果想要阻止MySQL Server在没有克隆插件的情况下运行，那么在插件初始化失败时，可以使用--clone选项设置FORCE或FORCE_PLUS_PERMANENT值强制MySQL Server启动失败
```

4. 快速搭建主从复制拓扑

- 首先，确保克隆插件在master、shadow、slave中都已经安装且处于激活状态:

```
root@localhost : (none) 03:10:09> SELECT PLUGIN_NAME, PLUGIN_STATUS FROM
INFORMATION_SCHEMA.PLUGINS WHERE PLUGIN_NAME = 'clone';
+-------------+---------------+
| PLUGIN_NAME | PLUGIN_STATUS |
+-------------+---------------+
| clone       | ACTIVE        |
+-------------+---------------+
1 row in set (0.00 sec)
```

在master中创建用于执行克隆操作的用户账户，使用mysql_native_password认证插件:

```
root@localhost : (none) 04:45:56> create user clone_user identified with
mysql_native_password by 'Bgview@2020';
Query OK, 0 rows affected (0.01 sec)
root@localhost : (none) 04:47:13> grant replication slave,backup_admin on *.* to
clone_user;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 04:47:18> show grants for clone_user;
+-----------------------------------------------------+
| Grants for clone_user@%                             |
+-----------------------------------------------------+
| GRANT REPLICATION SLAVE ON *.* TO `clone_user`@`%` |
| GRANT BACKUP_ADMIN ON *.* TO `clone_user`@`%`      |
+-----------------------------------------------------+
2 rows in set (0.00 sec)
```

在shadow中，使用如下克隆语句，执行远程克隆操作:

```
# 使用系统变量clone_valid_donor_list指定允许执行远程克隆操作的数据源实例对应的IP和端口信息
root@localhost : (none) 04:51:41> SET GLOBAL clone_valid_donor_list =
'10.10.30.11:3306';
Query OK, 0 rows affected (0.00 sec)
# 加载组复制插件（这里只是加载，不需要启用也不需要额外的配置）
root@localhost : (none) 10:11:48> INSTALL PLUGIN group_replication SONAME
'group_replication.so';
Query OK, 0 rows affected (0.01 sec)
# 执行远程克隆操作
root@localhost : (none) 04:53:03> CLONE INSTANCE FROM
'clone_user'@'10.10.30.11':3306 IDENTIFIED BY 'Bgview@2020';
Query OK, 0 rows affected (15.04 sec)
# 对于远程克隆操作，如果没有未远程克隆的数据副本指定一个在本地的存放路径，则会覆盖本地数据库实例的数据目录下的所有文件，在远程克隆操作执行结束后，会自动使用新的数据副本来重启本地数据库实例
root@localhost : (none) 04:53:38> show databases;
ERROR 2006 (HY000): MySQL server has gone away  # 从这里可以看到，之前执行远程克隆操作的会话，已经断开了（由于本地数据库实例重启，所以，本地数据库实例中的所有连接都会被断开）
```

```
No connection. Trying to reconnect...
Connection id:    7
Current database: *** NONE ***
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sbtest             |
| sbtest2            |
| sbtest3            |
| sbtest4            |
| sbtest5            |
| sys                |
+--------------------+
9 rows in set (0.01 sec)
```

现在，在master和shadow分别查看GTID信息（这里假设master没有任何数据写入）

```
# master中的GTID信息
root@localhost : (none) 05:05:07> show master status\G
*************************** 1. row ***************************
             File: mysql-bin.000054
         Position: 2076
     Binlog_Do_DB:
Binlog_Ignore_DB:
Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-454723:1386710-
1412416:2398244-2400644,
ee0905d8-a4a5-11ea-aa76-001c42789047:1-187
1 row in set (0.00 sec)
# shadow中的GTID信息
root@localhost : (none) 05:05:07> show master status\G
*************************** 1. row ***************************
             File: mysql-bin.000054
         Position: 2076
     Binlog_Do_DB:
Binlog_Ignore_DB:
Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-454723:1386710-
1412416:2398244-2400644,
ee0905d8-a4a5-11ea-aa76-001c42789047:1-187
1 row in set (0.00 sec)
# 通过比对master和shadow中的二进制日志文件和位置信息、GTID信息，可以发现，两者位置信息完全一致，说明克隆操作在获取的数据快照副本的同时，也获取了一个与数据保持一致的位置信息，而且该位置信息不是单独计量在某个文件中，不需要再单独在数据库中进行设置，这样一来，要在shadow中配置主从复制就变得非常简单，只需要使用CHANGE MASTER语句，带上master的连接信息即可，如下
root@localhost : (none) 05:50:43> change master to
master_host='10.10.30.11',master_user='clone_user',master_password='Bgview@2020'
,master_auto_position=1 for channel 'ms_replication_clone_test';
# 启动复制线程
root@localhost : (none) 05:51:43> start slave for channel
 'ms_replication_clone_test'\G
Query OK, 0 rows affected (0.06 sec)
# 查看复制状态
root@localhost : (none) 05:52:16> show slave status for channel
 'ms_replication_clone_test'\G
*************************** 1. row ***************************
```

```
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 10.10.30.11
                  Master_User: clone_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000054
          Read_Master_Log_Pos: 2076
               Relay_Log_File: mysql-relay-bin-ms_replication_clone_test.000002
                Relay_Log_Pos: 402
        Relay_Master_Log_File: mysql-bin.000054
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
......
          Exec_Master_Log_Pos: 2076
              Relay_Log_Space: 629
......
        Seconds_Behind_Master: 0
......
           Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-
454723:1386710-1412416:2398244-2400644,
ee0905d8-a4a5-11ea-aa76-001c42789047:1-187
                Auto_Position: 1
         Replicate_Rewrite_DB:
                 Channel_Name: ms_replication_clone_test
           Master_TLS_Version:
       Master_public_key_path:
        Get_master_public_key: 0
            Network_Namespace:
1 row in set (0.00 sec)
# 注意：这里我们假设master没有任何数据写入，目的是为了证明通过克隆插件获取的数据快照与获取的快照位置信息是一致的。正是因为这一特性，才使得在使用克隆插件执行克隆操作期间，master允许持续写入数据，不需要阻塞业务对数据库的读/写访问。
```

在master中写入一些测试数据，看看是否能够正常同步到shadow中:

```
# 主库中写入测试数据
root@localhost : (none) 06:35:35> create database sbtest6;
Query OK, 1 row affected (0.00 sec)
# shadow中查看数据是否已经同步
root@localhost : (none) 06:25:04> show databases;
+--------------------+
| Database           |
+--------------------+
......
| sbtest6            |
| sys                |
+--------------------+
10 rows in set (0.00 sec)
```

至此，shadow通过克隆插件从master中获取全量数据快照的方式，配置并启动好了复制线程，现在，我们继续通过克隆插件，从shadow中获取全量数据库快照的方式来配置slave，在slave中，执行如下操作:

```
# 通过克隆插件从shadow获取全量数据快照来配置slave，只需要执行如下语句即可，无需额外的其他配置步骤（因为复制配置信息，在shadow中）
```

```
root@localhost : (none) 04:51:41> SET GLOBAL clone_valid_donor_list =
'10.10.30.12:3306';
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 10:11:48> INSTALL PLUGIN group_replication SONAME
'group_replication.so';
Query OK, 0 rows affected (0.01 sec)
root@localhost : (none) 10:15:10> CLONE INSTANCE FROM
'clone_user'@'10.10.30.12':3306 IDENTIFIED BY 'Bgview@2020';
Query OK, 0 rows affected (10.83 sec)
# 等待克隆语句执行完成之后，我们直接使用如下语句查看复制配置信息，可以发现，slave的复制配置信息
已存在，且复制线程已经正常启动，说明slave已经成功加入了主从复制拓扑中
admin@localhost : (none) 10:15:54> show slave status\G
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    25
Current database: *** NONE ***
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 10.10.30.12
                  Master_User: clone_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000055
          Read_Master_Log_Pos: 232
               Relay_Log_File: mysql-relay-bin-ms_replication_clone_test.000002
                Relay_Log_Pos: 367
        Relay_Master_Log_File: mysql-bin.000055
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
......
          Exec_Master_Log_Pos: 232
              Relay_Log_Space: 602
......
        Seconds_Behind_Master: 0
......
            Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-
454723:1386710-1412416:2398244-2400644,
ee0905d8-a4a5-11ea-aa76-001c42789047:1-188
                Auto_Position: 1
         Replicate_Rewrite_DB:
                 Channel_Name: ms_replication_clone_test
......
1 row in set (0.00 sec)
```

- PS：为了最大限度地简化管理成本，建议在发送方上（数据源）设置master_info_repository=TABLE和relay_log_info_repository=TABLE时（这是MySQL 8.0的默认设置），在整个复制拓扑中启用GTID复制。
-
  - 克隆插件，要求数据的发送方（数据源）和数据的接收方（数据目标）使用相同的MySQL Server版本，否则拒绝执行克隆操作
  - 虽然在这一节里，我们介绍的是使用克隆插件来搭建主从复制拓扑，但是，克隆插件要求执行克隆操作必须加载组复制插件（只是加载组复制插件，并不需要启用），否则报错：

    ```
    ERROR 3870 (HY000): Clone Donor plugin group_replication is not active
    in Recipient
    ```

- 对于执行克隆操作时，从数据源实例中获取的位置信息，也可以在performance_schema.log_status表中查看（LOCAL字段记录了获取数据快照时的位置信息，存储内容为一个JSON数组，其中包含了二进制日志文件和位置信息、GTID等信息）

```
# master
root@localhost : (none) 05:05:10> select * from performance_schema.log_status\G
*************************** 1. row ***************************
   SERVER_UUID: ee0905d8-a4a5-11ea-aa76-001c42789047
         LOCAL: {"gtid_executed": "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-
454723:1386710-1412416:2398244-2400644,\nee0905d8-a4a5-11ea-aa76-001c42789047:1-
187", "binary_log_file": "mysql-bin.000054", "binary_log_position": 2076}
   REPLICATION: {"channels": [{"channel_name": "group_replication_applier",
"relay_log_file": "mysql-relay-bin-group_replication_applier.000023",
"relay_log_position": 152}, {"channel_name": "group_replication_recovery",
"relay_log_file": "mysql-relay-bin-group_replication_recovery.000006",
"relay_log_position": 152}]}
STORAGE_ENGINES: {"InnoDB": {"LSN": 3915886924, "LSN_checkpoint": 3915886924}}
1 row in set (0.01 sec)
# shadow
root@localhost : (none) 05:33:13> select * from performance_schema.log_status\G
*************************** 1. row ***************************
   SERVER_UUID: fdb2b4cd-a4a5-11ea-916d-001c42c65c10
         LOCAL: {"gtid_executed": "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-
454723:1386710-1412416:2398244-2400644,\nee0905d8-a4a5-11ea-aa76-001c42789047:1-
187", "binary_log_file": "mysql-bin.000002", "binary_log_position": 152}
   REPLICATION: {"channels": [{"channel_name": "group_replication_applier",
"relay_log_file": "mysql-relay-bin-group_replication_applier.000012",
"relay_log_position": 152}, {"channel_name": "group_replication_recovery",
"relay_log_file": "mysql-relay-bin-group_replication_recovery.000006",
"relay_log_position": 152}]}
STORAGE_ENGINES: {"InnoDB": {"LSN": 3915933877, "LSN_checkpoint": 3915933877}}
1 row in set (0.00 sec)
```

5.利用克隆插件快速搭建组复制拓扑

- 假设有用于搭建组复制的三台服务器，且在三台服务器中都各自初始化安装好了MySQL数据库，但没有配置组复制拓扑，如下：
-
  - 节点1：10.10.30.11
  - 节点2：10.10.30.12
  - 节点3：10.10.30.13
- 在组复制拓扑中，如果配置了克隆插件，则组复制插件会自动接管克隆插件，如果有新的节点尝试加入组复制拓扑时，复制组会尝试使用基于二进制日志的状态传输为新加入的节点提供数据快照，如果新加入的节点请求的数据位置，组复制拓扑中所有节点都无法提供基于二进制日志的状态传输，则会尝试使用基于全量数据克隆（即使用克隆插件，克隆插件需要事先安装且处于可用状态）的状态传输为新加入的节点提供数据快照
- 要利用克隆插件快速搭建组复制拓扑，首先，在三个节点的MySQL配置选项文件中，依次配置好如下内容，并重新启动MySQL Server

```
# 节点1
[root@localhost ~]# cat /etc/my.cnf
[mysqld]
......
super_read_only=1
```

```
server_id=330611
sync_binlog=10000
innodb_flush_log_at_trx_commit = 2
binlog-checksum=NONE
binlog_row_image=full
log_slave_updates=ON
binlog_format=ROW
master_info_repository=TABLE
relay_log_info_repository=TABLE
gtid_mode=ON
enforce_gtid_consistency=ON
transaction-write-set-extraction=XXHASH64
auto_increment_increment=3
auto_increment_offset=1
plugin_load_add='group_replication.so;mysql_clone.so'
loose-group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
loose-group_replication_single_primary_mode=OFF
loose-group_replication_enforce_update_everywhere_checks=ON
loose-group_replication_start_on_boot=OFF
loose-group_replication_ip_whitelist='0.0.0.0/0'
loose-group_replication_local_address='10.211.55.11:24901'
loose-
group_replication_group_seeds='10.211.55.11:24901,10.211.55.12:24901,10.211.55.1
3:24901'
loose-group_replication_bootstrap_group=OFF
report_host='10.211.55.11'
# 节点2
[root@localhost ~]# cat /etc/my.cnf
[mysqld]
......
super_read_only=1
server_id=330612
sync_binlog=10000
innodb_flush_log_at_trx_commit = 2
binlog-checksum=NONE
binlog_row_image=full
log_slave_updates=ON
binlog_format=ROW
master_info_repository=TABLE
relay_log_info_repository=TABLE
gtid_mode=ON
enforce_gtid_consistency=ON
transaction-write-set-extraction=XXHASH64
auto_increment_increment=3
auto_increment_offset=2
plugin_load_add='group_replication.so;mysql_clone.so'
loose-group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
loose-group_replication_single_primary_mode=OFF
loose-group_replication_enforce_update_everywhere_checks=ON
loose-group_replication_start_on_boot=OFF
loose-group_replication_ip_whitelist='0.0.0.0/0'
loose-group_replication_local_address='10.211.55.12:24901'
loose-
group_replication_group_seeds='10.211.55.11:24901,10.211.55.12:24901,10.211.55.1
3:24901'
loose-group_replication_bootstrap_group=OFF
report_host='10.211.55.12'
# 节点3
```

```
[root@localhost ~]# cat /etc/my.cnf
[mysqld]
......
super_read_only=1
server_id=330613
sync_binlog=10000
innodb_flush_log_at_trx_commit = 2
binlog-checksum=NONE
binlog_row_image=full
log_slave_updates=ON
binlog_format=ROW
master_info_repository=TABLE
relay_log_info_repository=TABLE
gtid_mode=ON
enforce_gtid_consistency=ON
transaction-write-set-extraction=XXHASH64
auto_increment_increment=3
auto_increment_offset=3
plugin_load_add='group_replication.so;mysql_clone.so'
loose-group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
loose-group_replication_single_primary_mode=OFF
loose-group_replication_enforce_update_everywhere_checks=ON
loose-group_replication_start_on_boot=OFF
loose-group_replication_ip_whitelist='0.0.0.0/0'
loose-group_replication_local_address='10.211.55.13:24901'
loose-
group_replication_group_seeds='10.211.55.11:24901,10.211.55.12:24901,10.211.55.1
3:24901'
loose-group_replication_bootstrap_group=OFF
report_host='10.211.55.13'
# 在配置文件中设置好上述选项之后，依次重启三个节点的MySQL Server
[root@localhost ~]# service mysqld restart
Shutting down MySQL.. SUCCESS!
Starting MySQL... SUCCESS!
```

使用root用户登录到节点1的数据库实例中，创建复制用户，并引导集群启动：

```
# 首先查看组复制插件和克隆插件的状态信息，从下面的信息中可以看到，这两个插件当前在节点1中都处于
激活状态
root@localhost : (none) 11:12:55> SELECT PLUGIN_NAME, PLUGIN_STATUS FROM
INFORMATION_SCHEMA.PLUGINS WHERE PLUGIN_NAME in ('clone','group_replication');
+-------------------+---------------+
| PLUGIN_NAME       | PLUGIN_STATUS |
+-------------------+---------------+
| group_replication | ACTIVE        |
| clone             | ACTIVE        |
+-------------------+---------------+
2 rows in set (0.00 sec)
# 创建复制用户，并为复制用于赋权
root@localhost : (none) 11:08:11> set global read_only=0;
set global super_read_only=0;
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:08:11> set sql_log_bin=0;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:08:55> CREATE USER group_repl@'%' IDENTIFIED WITH
mysql_native_password BY 'password';
```

```
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:09:50> GRANT REPLICATION SLAVE,BACKUP_ADMIN ON *.* TO
group_repl@'%' ;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:10:23> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:08:11> set sql_log_bin=1;
Query OK, 0 rows affected (0.00 sec)
# 配置组复制并引导集群启动
root@localhost : (none) 11:11:27> set global
group_replication_bootstrap_group=ON;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:11:35> set global group_replication_start_on_boot=ON;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:11:40> CHANGE MASTER TO MASTER_USER='group_repl',
MASTER_PASSWORD='password' FOR CHANNEL 'group_replication_recovery';
Query OK, 0 rows affected, 2 warnings (0.01 sec)
root@localhost : (none) 11:11:50> start group_replication;
Query OK, 0 rows affected (3.18 sec)
root@localhost : (none) 11:12:43> set global
group_replication_bootstrap_group=OFF;
Query OK, 0 rows affected (0.00 sec)
# 引导集群启动成功之后，可通过performance_schema.replication_group_members表查看组成员的
状态信息，MEMBER_STATE字段为ONLINE，表示节点1已经成功加入到复制组中
root@localhost : (none) 11:12:49> SELECT * FROM
performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION |
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
| group_replication_applier | dfb1c0c1-d07e-11ea-98e7-001c42789047 | 10.10.30.11
|        3306 | ONLINE       | PRIMARY     | 8.0.20         |
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
1 row in set (0.01 sec)
```

在节点1中，使用sysbench造一些数据，完成之后，清理掉二进制日志。

```
# 使用sysbench造数
root@localhost : (none) 11:14:55> create database sbtest;
Query OK, 1 row affected (0.01 sec)
[root@localhost ~]# sysbench --db-driver=mysql --time=180 --report-interval=1 --
mysql-host=10.10.30.11 --mysql-port=3306 --mysql-user=root --mysql-
password=password --mysql-db=sbtest --tables=8 --table-size=50000 --db-ps-
mode=disable oltp_read_write prepare --threads=8
......
# 清理二进制日志
root@localhost : (none) 11:32:19> flush binary logs;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:32:33> show binary logs;
+------------------+-----------+-----------+
| Log_name         | File_size | Encrypted |
+------------------+-----------+-----------+
| mysql-bin.000001 |       179 | No        |
| mysql-bin.000002 |       179 | No        |
```

```
| mysql-bin.000003 |       171 | No        |
| mysql-bin.000004 | 153942756 | No        |
| mysql-bin.000005 |       232 | No        |
+------------------+-----------+-----------+
5 rows in set (0.00 sec)
root@localhost : (none) 11:32:34> purge binary logs to 'mysql-bin.000005';
Query OK, 0 rows affected (0.02 sec)
root@localhost : (none) 11:32:44> show binary logs;
+------------------+-----------+-----------+
| Log_name         | File_size | Encrypted |
+------------------+-----------+-----------+
| mysql-bin.000005 |       232 | No        |
+------------------+-----------+-----------+
1 row in set (0.00 sec)
```

登录到节点2中，创建复制账号，配置组复制，并启动组复制，让节点2自动加入组复制拓扑中

```
# 检查组复制插件和克隆插件状态（略）
# 创建复制账号（这里要在会话级别关闭二进制日志的写入功能，我们故意让节点2中的数据与节点1不一致，在2个节点数据不一致的情况下，新加入节点是无法通过基于二进制日志的状态传输加入组复制拓扑的，只能通过基于数据克隆的状态传输加入组复制拓扑，即新加入节点中的数据会被克隆的数据副本全部覆盖，以便使得2个节点中的数据达到一致的状态）
root@localhost : (none) 11:18:58> set global read_only=0;set global
super_read_only=0;
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
# 在节点2中的操作，不能写入二进制日志，否则后续会由于节点2中存在复制组中不存在的数据而导致节点2无法加入节点1引导的复制组（这是出于数据的安全保护考虑，如果不加保护，如果节点2中的数据是有用的，被覆盖会导致数据丢失）
root@localhost : (none) 11:19:00> set sql_log_bin=0;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:19:00> CREATE USER group_repl@'%' IDENTIFIED WITH
mysql_native_password BY 'password';
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:19:08> GRANT REPLICATION SLAVE,BACKUP_ADMIN ON *.* TO
group_repl@'%' ;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:19:12> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:19:00> set sql_log_bin=1;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 11:36:21> start group_replication;
Query OK, 0 rows affected (3.94 sec)  # 从这里可以看到，直接启动节点2的组复制线程成功了
# 在节点2中启动组复制成功之后，可通过performance_schema.replication_group_members表查看组成员的状态信息，从下面的信息中可以看到，节点2也已经成功加入组复制拓扑
root@localhost : (none) 11:41:46> SELECT * FROM
performance_schema.replication_group_members;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    53
Current database: *** NONE ***
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION |
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
```

```
| group_replication_applier | cf7dd27b-d07e-11ea-b86f-001c42c65c10 | 10.10.30.12
|        3306 | ONLINE       | PRIMARY     | 8.0.20         |
| group_replication_applier | dfb1c0c1-d07e-11ea-98e7-001c42789047 | 10.10.30.11
|        3306 | ONLINE       | PRIMARY     | 8.0.20         |
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
2 rows in set (0.01 sec)
```

登录到节点3中，按照与节点2同样的操作步骤，创建复制账号，配置组复制，并启动组复制，让节点3自动加入组复制拓扑中

```
# 详细过程省略，当节点3操作完成之后，可通过performance_schema.replication_group_members
表查看组成员的状态信息，从下面的信息中可以看到，节点3也已经成功加入组复制拓扑
root@localhost : (none) 01:49:34> SELECT * FROM
performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION |
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
| group_replication_applier | 02497ad9-d094-11ea-b80b-001c421ee27e | 10.10.30.13
|        3306 | ONLINE       | PRIMARY     | 8.0.20         |
| group_replication_applier | aab4673f-d094-11ea-b441-001c42c65c10 | 10.10.30.12
|        3306 | ONLINE       | PRIMARY     | 8.0.20         |
| group_replication_applier | dfb1c0c1-d07e-11ea-98e7-001c42789047 | 10.10.30.11
|        3306 | ONLINE       | PRIMARY     | 8.0.20         |
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
3 rows in set (0.00 sec)
```

PS：在组复制专用通道中配置使用mysql_native_password认证插件的用户，有可能会在执行START GROUP_REPLICATION语句尝试加入复制拓扑时失败，如果加入复制组失败，则也可以使用克隆语句手工执行远程克隆，指定一个在线的组复制成员执行远程克隆，然后，直接使用START GROUP_REPLICATION语句就能够让新的节点加入组复制拓扑中，类似如下（也可以在组复制专用通道中配置使用caching_sha2_password认证插件的用户，这样组复制插件会为组复制专用通道启用加密连接，就不会碰到因为无法为克隆操作获取用户凭证而导致组复制插件自动执行远程克隆操作失败）：

```
# 停止组复制线程
root@localhost : (none) 01:44:57> stop group_replication;
Query OK, 0 rows affected (1.01 sec)
# 指定数据源实例的IP和端口
root@localhost : performance_schema 01:07:26> SET GLOBAL clone_valid_donor_list
= '10.10.30.11:3306';
Query OK, 0 rows affected (0.00 sec)
# 执行远程克隆
root@localhost : performance_schema 01:12:40> CLONE INSTANCE FROM
'group_repl'@'10.10.30.11':3306 IDENTIFIED BY 'password';
Query OK, 0 rows affected (2.48 sec)
# 远程克隆操作执行完成之后，直接通过performance_schema.replication_group_members表查看组
成员列表，下面的信息中可以看到节点3已经成功加入了组复制拓扑
root@localhost : (none) 01:49:34> SELECT * FROM
performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
```

```
| CHANNEL_NAME               | MEMBER_ID                             | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION |
+----------------------------+---------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
| group_replication_applier | 02497ad9-d094-11ea-b80b-001c421ee27e |
10.211.55.13 |        3306 | ONLINE       | PRIMARY     | 8.0.20         |
| group_replication_applier | aab4673f-d094-11ea-b441-001c42c65c10 |
10.211.55.12 |        3306 | ONLINE       | PRIMARY     | 8.0.20         |
| group_replication_applier | dfb1c0c1-d07e-11ea-98e7-001c42789047 |
10.211.55.11 |        3306 | ONLINE       | PRIMARY     | 8.0.20         |
+----------------------------+---------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
3 rows in set (0.00 sec)
```

6.克隆操作启用连接加密

- 要对数据克隆操作启用连接加密
- - 需要一个使用caching_sha2_password认证插件的用户
  - 需要在所有节点中配置好SSL证书（在MySQL 8中，默认情况下会自动创建自签证书。也可以自行创建自签证书或使用付费证书）
  - 对于手工执行克隆语句，需要在克隆语句中使用REQUIRE SSL子句启用加密；对于组复制插件对克隆插件的自动接管，需要配置系统变量clone_ssl_ca、clone_ssl_cert、clone_ssl_key，如果不配置这些系统变量，则会自动从组复制系统变量group_replication_recovery_ssl_ca、group_replication_recovery_ssl_cert、group_replication_recovery_ssl_key中获取值（如果组复制系统变量也没有配置值，则组复制系统变量会从MySQL Server提供的系统变量ssl_ca、ssl_cert、ssl_key中获取值）
- 下面，我们将分别介绍在克隆语句中启用连接加密和在组复制拓扑中对数据克隆启用连接加密

在克隆语句中启用连接加密

- 假设节点1已经引导了集群启动，现在，需要使用克隆语句将节点2加入组复制拓扑中，且需要在克隆语句中启用连接加密
- 在节点1中创建新的复制用户，使用caching_sha2_password认证插件

```
root@localhost : (none) 03:35:47> create user group_repl_ssl identified with
caching_sha2_password by 'password';
Query OK, 0 rows affected (0.01 sec)
root@localhost : (none) 03:36:19> grant replication slave,backup_admin on *.* to
group_repl_ssl;
Query OK, 0 rows affected (0.01 sec)
```

在节点2中，使用如下克隆语句执行远程克隆操作:

```
root@localhost : performance_schema 01:07:26> SET GLOBAL clone_valid_donor_list
= '10.10.30.11:3306';
Query OK, 0 rows affected (0.00 sec)
root@localhost : performance_schema 01:12:40> CLONE INSTANCE FROM
'group_repl_ssl'@'10.10.30.11':3306 IDENTIFIED BY 'password' REQUIRE SSL;
Query OK, 0 rows affected (2.48 sec)
```

远程克隆操作执行完成之后，直接通过performance_schema.replication_group_members表查看组成员列表，下面的信息中可以看到节点2已经成功加入了组复制拓扑

```
root@localhost : (none) 03:51:14> SELECT * FROM
performance_schema.replication_group_members;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    7
Current database: *** NONE ***  # 远程克隆操作完成之后，重启了MySQL Server，且由于我们在配置选项文件中关闭了系统变量group_replication_start_on_boot，因此，组复制线程不会跟随MySQL Server一并启动
+---------------------------+-----------+-------------+-------------+------------
--+-------------+----------------+
| CHANNEL_NAME              | MEMBER_ID | MEMBER_HOST | MEMBER_PORT |
MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION |
+---------------------------+-----------+-------------+-------------+------------
--+-------------+----------------+
| group_replication_applier |           |             |        NULL | OFFLINE
   |             |                |
+---------------------------+-----------+-------------+-------------+------------
--+-------------+----------------+
1 row in set (0.01 sec)
# 手工启动组复制线程
root@localhost : (none) 03:51:45> start group_replication;
Query OK, 0 rows affected (3.79 sec)
# 通过performance_schema.replication_group_members查看组成员状态，从下面的信息中可以看到，节点2已经成功加入了组复制拓扑
root@localhost : (none) 03:52:05> SELECT * FROM
performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION |
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
| group_replication_applier | dfb1c0c1-d07e-11ea-98e7-001c42789047 | 10.10.30.11
|        3306 | ONLINE       | PRIMARY     | 8.0.20         |
| group_replication_applier | ffea7eb6-d0a6-11ea-a9c6-001c42c65c10 | 10.10.30.12
|        3306 | ONLINE       | PRIMARY     | 8.0.20         |
+---------------------------+--------------------------------------+-------------
-+-------------+--------------+-------------+----------------+
2 rows in set (0.00 sec)
```

在performance_schema.clone_status表中，可以查看到本次克隆操作的一些状态和操作时间信息

```
root@localhost : (none) 03:52:58> select * from
performance_schema.clone_status\G
*************************** 1. row ***************************
            ID: 1
           PID: 0
         STATE: Completed
    BEGIN_TIME: 2020-07-28 15:51:11.885
      END_TIME: 2020-07-28 15:51:19.099
        SOURCE: 10.211.55.11:3306
   DESTINATION: LOCAL INSTANCE
      ERROR_NO: 0
 ERROR_MESSAGE:
   BINLOG_FILE: mysql-bin.000005
BINLOG_POSITION: 10833
 GTID_EXECUTED: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-199,
```

```
dfb1c0c1-d07e-11ea-98e7-001c42789047:1-4
1 row in set (0.00 sec)
```

在组复制拓扑中对数据数据克隆启用连接加密

- 创建一个使用caching_sha2_password认证插件的用户
- 由于MySQL 8在启动MySQL Server启动时会自动创建自签证书，且会自动使用自签证书、秘钥等文件来配置MySQL Server提供的系统变量ssl_ca、ssl_cert、ssl_key。如果组复制的系统变量group_replication_recovery_ssl_ca、group_replication_recovery_ssl_cert、group_replication_recovery_ssl_key未配置，则会从MySQL Server提供的系统变量中获取值。如果克隆插件的系统变量clone_ssl_ca、clone_ssl_cert、clone_ssl_key未配置，则会从组复制的系统变量红获取值。为了演示方便，这里我们直接使用MySQL Server启动时会自动创建自签证书（即不需要再单独进行SSL相关的配置）
- 假设在节点3中，配置组复制时，在CHANGE MASTER语句中指定使用了认证插件的用户，组复制插件即可自动启动组复制恢复专用通道的连接加密

```
# 创建复制用户
root@localhost : (none) 04:22:58> set sql_log_bin=0;
Query OK, 0 rows affected (0.00 sec)
root@localhost : (none) 04:23:16> set global super_read_only=0;
Query OK, 0 rows affected (0.00 sec)
# 用于配置组复制专用通道的用户group_repl_ssl可以使用REQUIRE SSL子句，也可以不使用，如果需要使用，在CREATE USER语句的末尾添加REQUIRE SSL子句即可
root@localhost : (none) 03:35:47> create user group_repl_ssl identified with caching_sha2_password by 'password';
Query OK, 0 rows affected (0.01 sec)
root@localhost : (none) 03:36:19> grant replication slave,backup_admin on *.* to group_repl_ssl;
Query OK, 0 rows affected (0.01 sec)
# 使用 CHANGE MASTER语句指定group_repl_ssl用户配置组复制
root@localhost : (none) 04:25:52> CHANGE MASTER TO MASTER_USER='group_repl_ssl', MASTER_PASSWORD='password' FOR CHANNEL 'group_replication_recovery';
Query OK, 0 rows affected, 2 warnings (0.01 sec)
root@localhost : (none) 04:25:55> start group_replication;
Query OK, 0 rows affected (3.78 sec)
root@localhost : (none) 04:26:27> SELECT * FROM performance_schema.replication_group_members;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    7
Current database: *** NONE ***  # 远程克隆数据之后，节点3自动重启了MySQL Server，因此这里可以看到连接断开了
+---------------------------+-----------+-------------+-------------+------------+-------------+----------------+
| CHANNEL_NAME              | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION |
+---------------------------+-----------+-------------+-------------+------------+-------------+----------------+
| group_replication_applier |           |             |        NULL | OFFLINE    |             |                |
+---------------------------+-----------+-------------+-------------+------------+-------------+----------------+
1 row in set (0.01 sec)
# 手动启动组复制线程
root@localhost : (none) 04:26:41> start group_replication;
Query OK, 0 rows affected (3.54 sec)
```

```
# 通过performance_schema.replication_group_members查看组成员状态，从下面的信息中可以看到，节点3已经成功加入了组复制拓扑
root@localhost : (none) 04:26:58> SELECT * FROM performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------+-------------+----------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION |
+---------------------------+--------------------------------------+-------------+-------------+----------------+
| group_replication_applier | 703f5576-d0ab-11ea-bd41-001c421ee27e | 10.10.30.13 |        3306 | ONLINE       | PRIMARY     | 8.0.20         |
| group_replication_applier | dfb1c0c1-d07e-11ea-98e7-001c42789047 | 10.10.30.11 |        3306 | ONLINE       | PRIMARY     | 8.0.20         |
| group_replication_applier | ffea7eb6-d0a6-11ea-a9c6-001c42c65c10 | 10.10.30.12 |        3306 | ONLINE       | PRIMARY     | 8.0.20         |
+---------------------------+--------------------------------------+-------------+-------------+----------------+
3 rows in set (0.00 sec)
```

在错误日志中，我们可以看到类似如下信息

```
2020-07-28T16:26:24.076123+08:00 9 [System] [MY-010597] [Repl] 'CHANGE MASTER TO FOR CHANNEL 'group_replication_applier' executed'. Previous state master_host='', master_port= 3306, master_log_file='', master_log_pos= 4, master_bind=''. New state master_host='<NULL>', master_port= 0, master_log_file='', master_log_pos= 4, master_bind=''.
# 下面一行错误日志信息表明节点3没有从组复制拓扑中的任何存活组成员中找到能够提供基于二进制日志的快照传输来恢复数据的组成员，将使用基于克隆的状态传输
2020-07-28T16:26:27.850181+08:00 0 [Warning] [MY-013470] [Repl] Plugin group_replication reported: 'As no ONLINE member has the missing data for recovering in its binary logs, this member will start distributed recovery using clone.'
# 执行远程克隆
2020-07-28T16:26:28.907533+08:00 34 [Warning] [MY-013460] [InnoDB] Clone removing all user data for provisioning: Started
2020-07-28T16:26:28.990207+08:00 34 [Warning] [MY-013460] [InnoDB] Clone removing all user data for provisioning: Finished
2020-07-28T16:26:32.975660+08:00 0 [Warning] [MY-010909] [Server] /usr/local/mysql/bin/mysqld: Forcing close of thread 7  user: 'root'.
# 远程克隆操作执行完成之后，自动关闭MySQL Server
2020-07-28T16:26:37.144956+08:00 0 [System] [MY-010910] [Server] /usr/local/mysql/bin/mysqld: Shutdown complete (mysqld 8.0.20)  MySQL Community Server - GPL.
2020-07-28T16:26:37.613107+08:00 0 [Warning] [MY-011068] [Server] The syntax 'expire-logs-days' is deprecated and will be removed in a future release. Please use binlog_expire_logs_seconds instead.
# 自动启动MySQL Server
2020-07-28T16:26:37.613416+08:00 0 [System] [MY-010116] [Server] /usr/local/mysql/bin/mysqld (mysqld 8.0.20) starting as process 28172
2020-07-28T16:26:37.624573+08:00 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2020-07-28T16:26:38.689777+08:00 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2020-07-28T16:26:38.909297+08:00 0 [System] [MY-011323] [Server] X Plugin ready for connections. Socket: '/home/mysql/data/mysqldata1/sock/mysqlx.sock' bind-address: '::' port: 33060
```

```
2020-07-28T16:26:38.949965+08:00 0 [ERROR] [MY-011947] [InnoDB] Cannot open
'/data/mysqldata1/innodb_ts/ib_buffer_pool' for reading: No such file or
directory
2020-07-28T16:26:38.976043+08:00 0 [Warning] [MY-010068] [Server] CA certificate
ca.pem is self signed.
2020-07-28T16:26:39.014293+08:00 0 [System] [MY-010931] [Server]
/usr/local/mysql/bin/mysqld: ready for connections. Version: '8.0.20'  socket:
'/home/mysql/data/mysqldata1/sock/mysql.sock'  port: 3306  MySQL Community
Server - GPL.
```