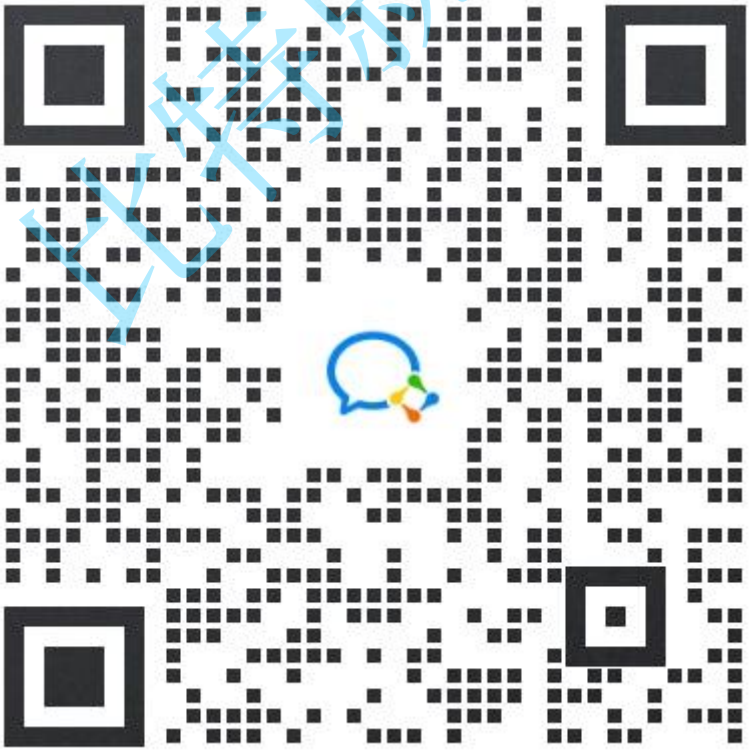
Dockerfile 搭建 mysql 主从集群

版权说明

对比特项目感兴趣，可以联系这个微信。



实战目的

通过 Dockerfile，配合 docker compose 完成 mysql 主从集群的搭建

基础知识

Docker compose 构建参数

build

- 功能

在 docker-compose.yml 文件中使用 build 选项编译镜像。

- 格式

```Shell
services:
  # 格式一
  frontend:
    image: awesome/webapp
    build: ./webapp

  # 格式二
  backend:
    image: awesome/database
    build:
      #构建上下文目录
      context: ./backend
      dockerfile: ./backend.Dockerfile
```

- 示例

```Bash
docker compose build
```

mysql 主从同步原理

什么是 mysql 主从同步

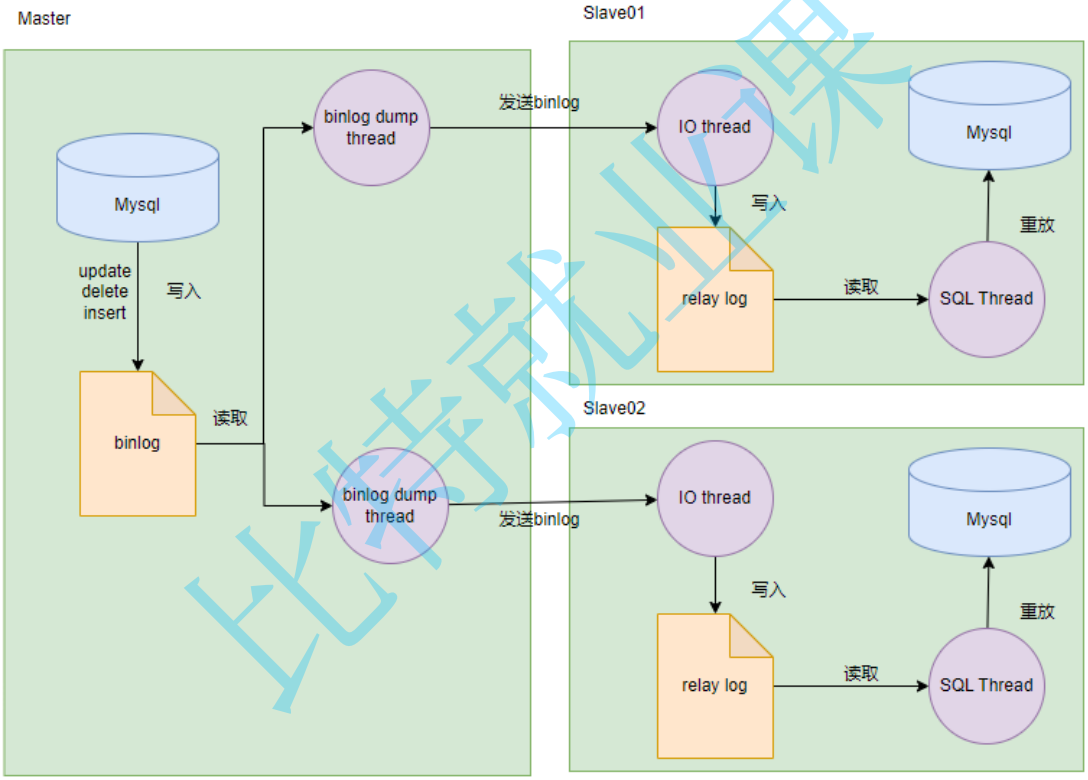
MySQL 主从复制是指数据可以从一个 MySQL 数据库服务器主节点复制到一个或多个从节点。MySQL 默认采用异步复制方式，这样从节点不用一直访问主服务器来更新自己的数据，数据的更新可以在远程连接上进行，从节点可以复制主数据库中的所有数据库或者特定的数据库，或者特定的表。

为什么要 mysql 主从同步

- 读写分离，性能提升：让主库负责写，从库负责读，这样，即使主库出现了锁表的情景，通过读从库也可以保证业务的正常运作。扩展架构提升读写能力。
- 数据实时备份：主库数据实时保存到存库，万一主库故障也有从库的数据备份
- 高可用 HA：当系统中某个节点发生故障时，可以方便的故障切换

主从同步的架构图

在主从复制的过程中，会基于三个线程来操作，一个是 binlog dump 线程，位于 master 节点上，另外两个线程分别是 I/O 线程和 SQL 线程，它们都分别位于 slave 节点上。



- 当 master 节点接收到一个写请求时，这个写请求可能是增删改操作，此时会把写请求的更新操作都记录到 binlog 日志中。
- master 节点会把数据复制给 slave 节点，如图中的 slave01 节点和 slave02 节点，这个过程，首先得要每个 slave 节点连接到 master 节点上，当 slave 节点连接到 master 节点上时，master 节点会为每一个 slave 节点分别创建一个 binlog dump 线程，用于向各个 slave 节点发送 binlog 日志。
- binlog dump 线程会读取 master 节点上的 binlog 日志，然后将 binlog 日志发送给 slave 节点上的 I/O 线程。当主库读取事件的时候，会在 Binglog 上加锁，读取完成之后，再将锁释放掉。

- slave 节点上的 I/O 线程接收到 binlog 日志后，会将 binlog 日志先写入到本地的 relaylog 中，relaylog 中就保存了 binlog 日志。
- slave 节点上的 SQL 线程，会来读取 relaylog 中的 binlog 日志，将其解析成具体的增删改操作，把这些在 master 节点上进行过的操作，重新在 slave 节点上也重做一遍，达到数据还原的效果，这样就可以保证 master 节点和 slave 节点的数据一致性了。

主从同步的数据内容其实是二进制日志（Binlog），它虽然叫二进制日志，实际上存储的是一个又一个的事件（Event），这些事件分别对应着数据库的更新操作，比如 INSERT、UPDATE、DELETE 等。

什么是 binlog

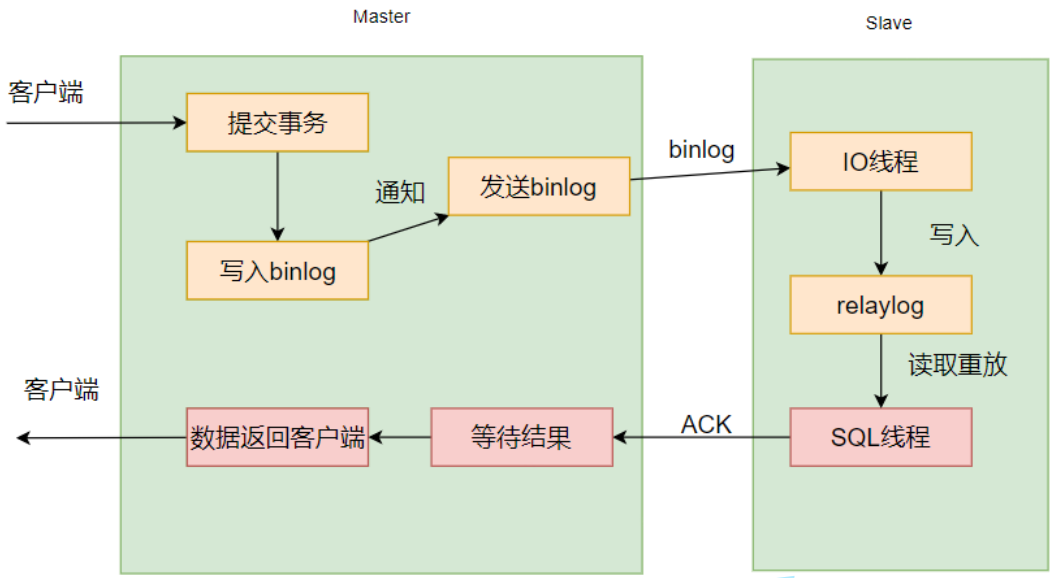
主库每提交一次事务，都会把数据变更，记录到一个二进制文件中，这个二进制文件就叫 binlog。需注意：只有写操作才会记录至 binlog，只读操作是不会的（如 select、show 语句）。

Bin Log 共有三种日志格式，可以 binlog_format 配置参数指定。

参数值	含义
Statement	记录原始 SQL 语句，会导致更新时间与原库不一致。 比如 update_time=now()
Row	记录每行数据的变化，保证了数据与原库一致，缺点是数据量较大。
Mixed	Statement 和 Row 的混合模式，默认采用 Statement 模式，涉及日期、函数相关的时候采用 Row 模式，既减少了数据量，又保证了数据一致性。

主从同步的方式

- 全同步方式：就是当主库执行完一个事务之后，要求所有的从库也都必须执行完该事务，才可以返回处理结果给客户端；因此，虽然全同步复制数据一致性得到保证了，但是主库完成一个事务需要等待所有从库也完成，性能就比较低了，从库如果挂了主库也受影响。

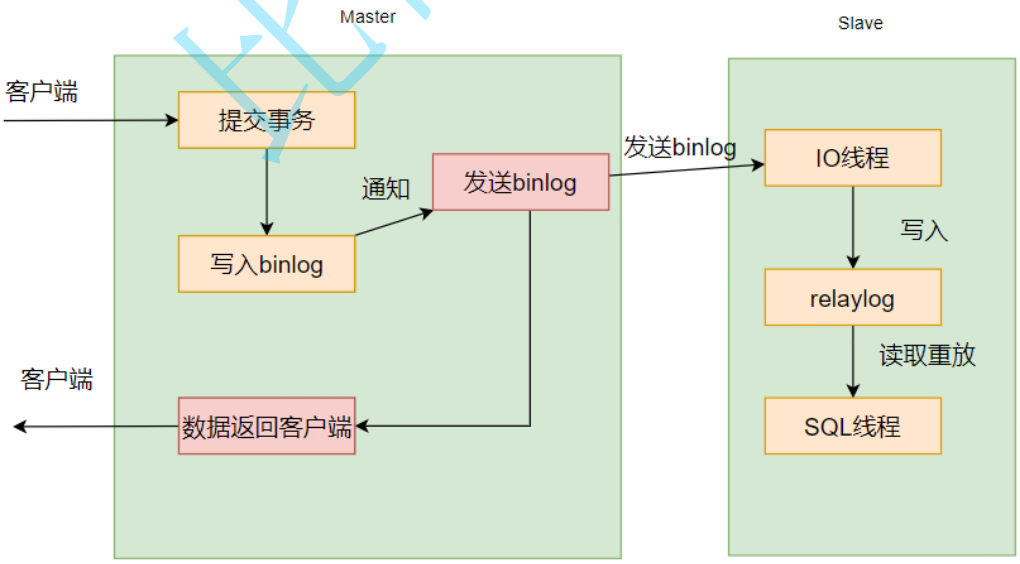


- 异步同步：默认方式，主库在执行完客户端提交的事务后会立即将结果返给给客户端，并不关心从库是否已经接收并处理。

因为主库只管自己执行完事务，就可以将处理结果返回给客户端，而不用关心从库是否执行完事务，这就可能导致短暂的主从数据不一致的问题了，比如刚在主库插入的新数据，如果马上在从库查询，就可能查询不到。

而且，当主库提交事物后，如果宕机挂掉了，此时可能 binlog 还没来得及同步给从库，这时候如果为了恢复故障切换主从节点的话，就会出现数据丢失的问题，所以异步复制虽然性能高，但数据一致性上是最弱的。

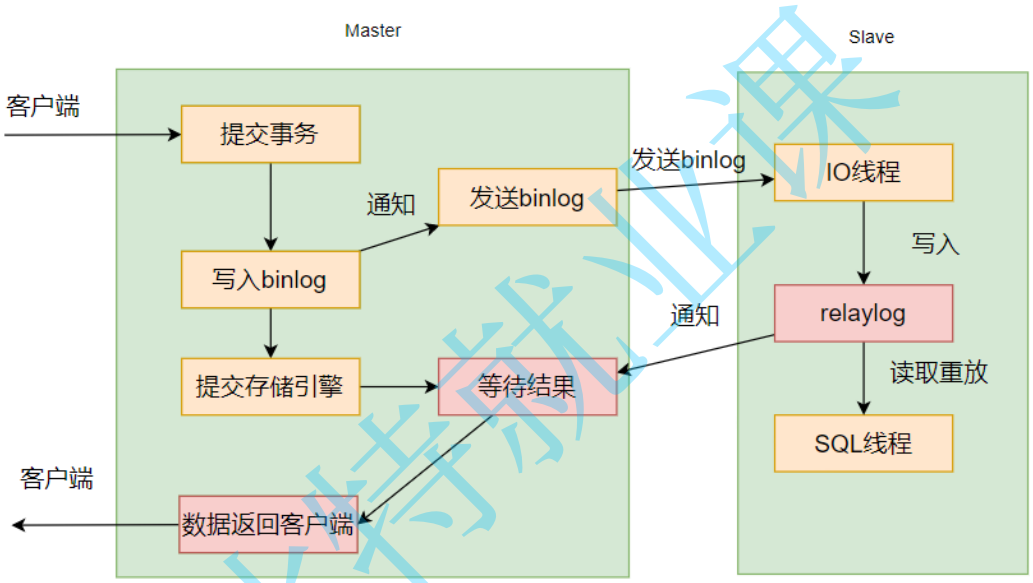
mysql 主从复制，默认采用的就是异步复制这种复制策略。



- 半同步：基于传统异步存在的缺陷，mysql 在 5.5 版本推出半同步复制。可以说半

同步复制是传统异步复制的改进，在 master 事务的 commit 之前，必须确保一个 slave 收到 relay log 并且响应给 master 以后，才能进行事务的 commit。相当于添加多了一个从库反馈机制。在 MySQL5.7 版本中还增加了一个 rpl_semi_sync_master_wait_for_slave_count 参数，我们可以对需要响应的从库数量进行设置，默认为 1，也就是说只要有一个从库进行了响应，就可以返回给客户端。如果将这个参数调大，可以提升数据一致性的强度，但也会增加主库等待从库响应的时间。

对应配置参数为 rpl_semi_sync_master_wait_point=after_commit。核心流程为，主库执行完事务后，主库提交 commit，同步 binlog 给从库，从库 ack 反馈接收到 binlog，反馈给客户端，释放会话；(主库生成 binlog，主库提交，再同步 binlog，等 ACK 反馈，返回客户端)



半同步复制存在以下几个问题:

- 半同步复制的性能，相比异步复制而言有所下降，相比于异步复制是不需要等待任何从库是否接收到数据的响应，而半同步复制则需要等待至少一个从库确认接收到 binlog 日志的响应，性能上是损耗更大的。

- 主库等待从库响应的最大时长是可以配置的，如果超过了配置的时间，半同步复制就会变成异步复制，那么，异步复制的问题同样也就会出现了。

- 半同步复制存在着幻读问题的:当主库成功提交事物并处于等待从库确认的过程中，这个时候，从库都还没来得及返回处理结果给客户端，但因为主库存储引擎内部已经提交事务了，所以，其他客户端是可以到从主库中读到数据的。但是，如果下一秒主库突然挂了，此时正好下一次请求过来，因为主库挂了，就只能把请求切换到从库中，因为从库还没从主库同步完数据，所以，从库中当然就读不到这条数据了，和上一秒读取数据的结果对比，就造成了幻读的现象了。

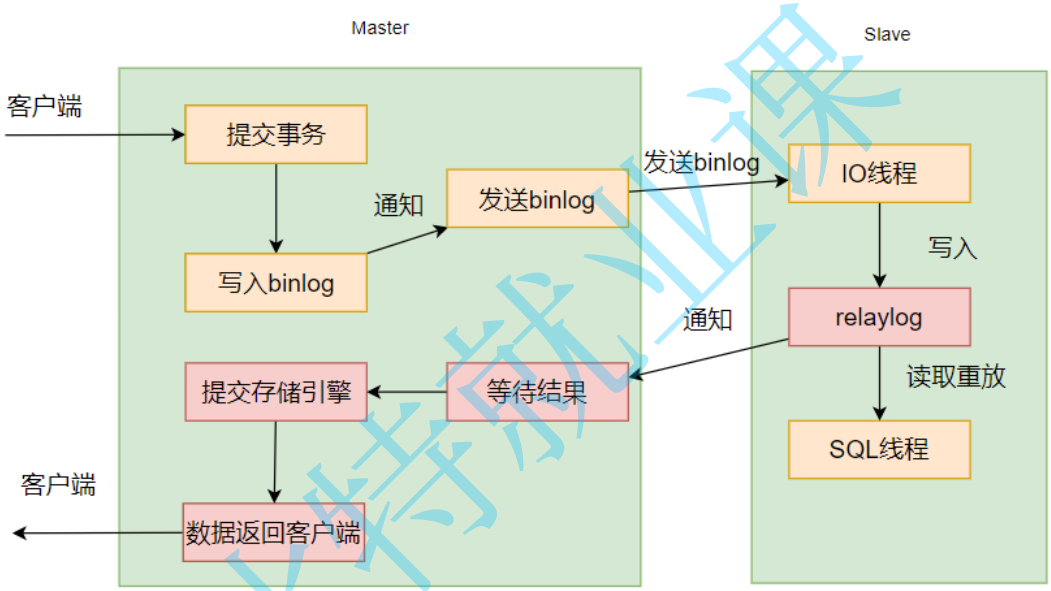
- 增强半同步复制

增强半同步复制，是 对半同步复制做的一个改进，原理上几乎是一样的，主要是解决幻读的问题。

主库配置了参数 rpl_semi_sync_master_wait_point = AFTER_SYNC 后，主库在存储引擎提交事务前，必须先收到从库数据同步完成的确认信息后，才能提交事务，以此来解决幻读问题。

核心流程为主库执行完事务后，同步 binlog 给从库，从库 ack 反馈接收到 binlog，主库提交 commit，反馈给客户端，释放会话；(主库生成 binlog，再同步 binlog，等 ACK 反馈，主库提交，返回客户端)

但是 slave 对于 relay log 的应用仍然是异步进行的.



- 组复制:

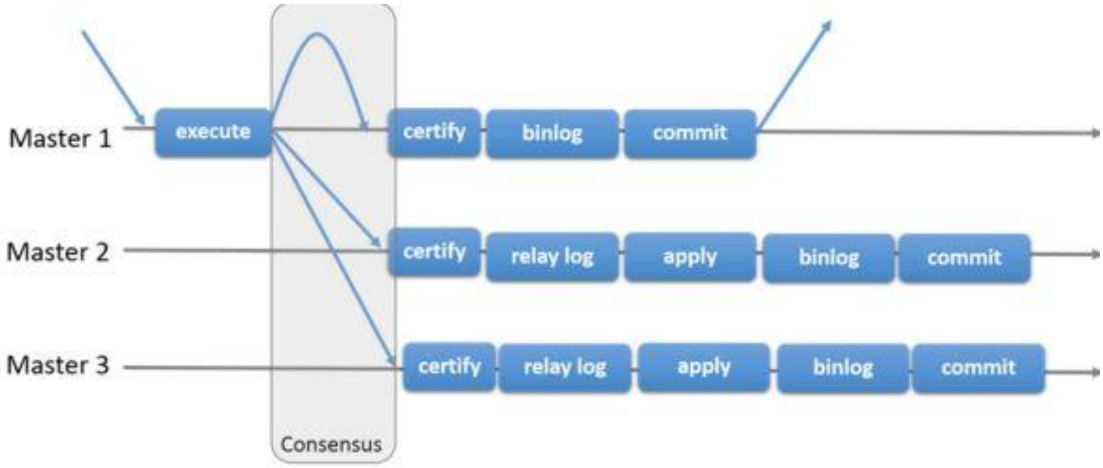
MySQL 官方在 5.7.17 版本正式推出组复制（MySQL Group Replication，简称 MGR）

由若干个节点共同组成一个复制组，一个事务的提交，必须经过组内大多数节点（N / 2 + 1）决议并通过，才能得以提交。如上图所示，由 3 个节点组成一个复制组，Consensus 层为一致性协议层，在事务提交过程中，发生组间通讯，由 2 个节点决议(certify)通过这个事务，事务才能够最终得以提交并响应。

引入组复制，主要是为了解决传统异步复制和半同步复制可能产生数据不一致的问题。组复制依靠分布式一致性协议(Paxos 协议的变体)，实现了分布式下数据的最终一致性，提供了真正的数据高可用方案(是否真正高可用还有待商榷)。其提供的多写方案，给我们实现多活方案带来了希望。

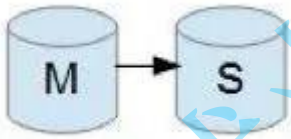
MGR 的解决方案有一定的局限性，如仅支持 InnoDB 表，并且每张表一定要有一

个主键，用于做 write set 的冲突检测；开启 GTID 特性等

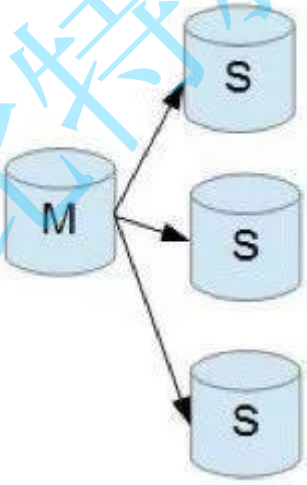


MySQL 主从形式

一主一从

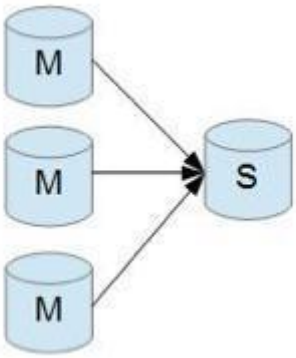


一主多从，提高系统的读性能



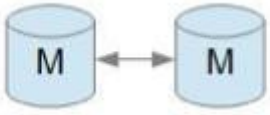
一主一从和一主多从是最常见的主从架构，实施起来简单并且有效，不仅可以实现 HA，而且还能读写分离，进而提升集群的并发能力。

多主一从



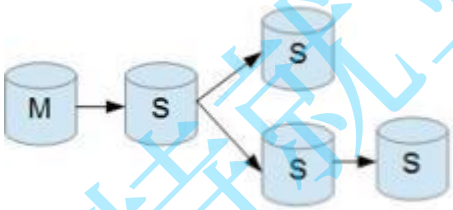
多主一从可以将多个 mysql 数据库备份到一台存储性能比较好的服务器上。

双主复制



双主复制，也就是互做主从复制，每个 master 既是 master，又是另外一台服务器的 slave。这样任何一方所做的变更，都会通过复制应用到另外一方的数据库中。

级联复制



级联复制模式下，部分 slave 的数据同步不连接主节点，而是连接从节点。因为如果主节点有太多的从节点，就会损耗一部分性能用于 replication，那么我们可以让 3~5 个从节点连接主节点，其它从节点作为二级或者三级与从节点连接，这样不仅可以缓解主节点的压力，并且对数据一致性没有负面影响。

mysql 主从集群搭建步骤

1. 创建主库，并在主库中创建单独的 mysql 用户用于同步数据,授予该用户数据同步权限
2. 创建从库，配置从库的数据同步的主库信息
3. 启动从库，开始同步

实战操作

创建一个一主二从的 mysql 集群

1. 创建目录

```Shell
mkdir -p /data/maxhou/mysqlcluster/
mkdir -p /data/maxhou/mysqlcluster/master/
mkdir -p /data/maxhou/mysqlcluster/slave/
```

2. 进入目录/data/maxhou/mysqlcluster/master,创建主库 Dockerfile 文件为 Dockerfile-master

```Shell
FROM mysql:5.7
RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
COPY ./master/master.sql /docker-entrypoint-initdb.d
```

3. 创建主库配置脚本 master.sql

```Shell
CREATE USER 'root'@'%' IDENTIFIED BY 'root';
grant replication slave, replication client on *.* to 'root'@'%';
flush privileges;
```

4. 进入目录/data/maxhou/mysqlcluster/slave/，创建从库 Dockerfile，文件为 Dockerfile-slave

```Shell
FROM mysql:5.7
RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
COPY ./slave/slave.sql /docker-entrypoint-initdb.d
```

5. 创建从库配置脚本 slave.sql

```Shell
change master to master_host='mysql-master',
master_user='root',master_password='root',master_port=3306;
start slave;
```

6. 进入/data/maxhou/mysqlcluster 目录，创建 docker-compose.yml 配置文件

- server-id:
- log-bin:打开二进制日志功能,配置 binlog 文件名
- binlog-ignore-db: 配置忽略的数据库
- binlog_cache_size:在一个事务中 binlog 为了记录 SQL 状态所持有的 cache 大小，如果经常使用大事务，可以增加此值来获取更大的性能。

- binlog_format:ROW/STATEMENT/MIXED
- lower_case_table_names:表采用小写
- character-set-server:配置字符集
- collation-server:配置比较规则

```Shell
version: "3"
services:
  mysql-master:
    build:
      context: ./
      dockerfile: ./master/Dockerfile-master
    image: mysqlmaster:v1.0
    restart: always
    container_name: mysql-master
    volumes:
      - ./mastervarlib:/var/lib/mysql
    ports:
      - 9306:3306
    environment:
      MYSQL_ROOT_PASSWORD: root
    privileged: true
    command: ['--server-id=1',
              '--log-bin=master-bin',
              '--binlog-ignore-db=mysql',
              '--binlog_cache_size=256M',
              '--binlog_format=mixed',
              '--lower_case_table_names=1',
              '--character-set-server=utf8',
              '--collation-server=utf8_general_ci']

  mysql-slave:
    build:
      context: ./
      dockerfile: ./slave/Dockerfile-slave
    image: mysqlslave:v1.0
    restart: always
    container_name: mysql-slave
    volumes:
      - ./slavevarlib:/var/lib/mysql
    ports:
      - 9307:3306
    environment:
```

```
      MYSQL_ROOT_PASSWORD: root
    privileged: true
    command: ['--server-id=2',
              '--relay_log=slave-relay',
              '--lower_case_table_names=1',
              '--character-set-server=utf8',
              '--collation-server=utf8_general_ci']
    depends_on:
      - mysql-master
  mysql-slave2:
    build:
      context: ./
      dockerfile: ./slave/Dockerfile-slave
    image: mysqlslave:v1.0
    restart: always
    container_name: mysql-slave2
    volumes:
      - ./slavevarlib2:/var/lib/mysql
    ports:
      - 9308:3306
    environment:
      MYSQL_ROOT_PASSWORD: root
    privileged: true
    command: ['--server-id=3',
              '--relay_log=slave-relay',
              '--lower_case_table_names=1',
              '--character-set-server=utf8',
              '--collation-server=utf8_general_ci']
    depends_on:
      - mysql-master
```

7. 构建镜像

```
Shell
root@139-159-150-152:/data/maxhou/mysqlcluster# docker compose
build
[+] Building 0.1s (8/8) FINISHED
 => [internal] load build definition from Dockerfile-master
0.1s
 => => transferring dockerfile: 174B
0.1s
 => [internal] load .dockerignore
0.0s
 => => transferring context: 2B
```

```
0.0s
 => [internal] load metadata for docker.io/library/mysql:5.7
0.0s
 => [1/3] FROM docker.io/library/mysql:5.7
0.0s
 => [internal] load build context
0.0s
 => => transferring context: 225B
0.0s
 => CACHED [2/3] RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai
/etc/localtime
0.0s
 => CACHED [3/3] COPY ./master/master.sql /docker-entrypoint-
initdb.d
0.0s
 => exporting to image
0.0s
 => => exporting layers
0.0s
 => => writing image
sha256:411d6631ca9872e44d3488761ca8865e8def82adc720d36db279014130b
b9462
0.0s
 => => naming to docker.io/library/mysqlmaster:v1.0
0.0s
[+] Building 0.1s (8/8) FINISHED
 => [internal] load build definition from Dockerfile-slave
0.0s
 => => transferring dockerfile: 169B
0.0s
 => [internal] load .dockerignore
0.0s
 => => transferring context: 2B
0.0s
 => [internal] load metadata for docker.io/library/mysql:5.7
0.0s
 => [1/3] FROM docker.io/library/mysql:5.7
0.0s
 => [internal] load build context
0.0s
 => => transferring context: 189B
0.0s
 => CACHED [2/3] RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai
/etc/localtime
```

```
0.0s
 => CACHED [3/3] COPY ./slave/slave.sql /docker-entrypoint-
initdb.d
0.0s
 => exporting to image
0.0s
 => => exporting layers
0.0s
 => => writing image
sha256:8747faa2258bf0ade83942724bd0800533bee02cf28195a82fd089eb000
ec1df
0.0s
 => => naming to docker.io/library/mysqlslave:v1.0
```

8. 启动服务进行测试

```
Shell
root@139-159-150-152:/data/maxhou/mysqlcluster# docker compose up
-d
[+] Running 4/4
 ✔ Network mysqlcluster_default   Created
0.1s
 ✔ Container mysql-master         Started
0.8s
 ✔ Container mysql-slave2         Started
2.2s
 ✔ Container mysql-slave          Started
2.2s
```

9. 状态查看正常

```
Shell
root@139-159-150-152:/data/maxhou/mysqlcluster# docker compose ps
NAME                IMAGE               COMMAND
SERVICE             CREATED             STATUS              PORTS
mysql-master        mysqlmaster:v1.0    "docker-entrypoint.s…"
mysql-master        18 seconds ago      Up 17 seconds
33060/tcp, 0.0.0.0:9306->3306/tcp, :::9306->3306/tcp
mysql-slave         mysqlslave:v1.0     "docker-entrypoint.s…"
mysql-slave         18 seconds ago      Up 15 seconds
33060/tcp, 0.0.0.0:9307->3306/tcp, :::9307->3306/tcp
mysql-slave2        mysqlslave:v1.0     "docker-entrypoint.s…"
mysql-slave2        18 seconds ago      Up 15 seconds
33060/tcp, 0.0.0.0:9308->3306/tcp, :::9308->3306/tcp
```

10. 连上主库，查看数据库可以看到运行正常

```
Shell
root@139-159-150-152:/data/maxhou/mysqlcluster# docker exec -it
mysql-master bash
root@bd3f9f453628:/# mysql -p
Enter password:
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

11. 查看数据库角色，和同步状态，连接上从库，在主库和从库上都可以执行这 2 个命令

```
Shell
#查看命令
SHOW MASTER STATUS\G
SHOW SLAVE STATUS\G


mysql> show slave status \G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: mysql-master
                  Master_User: root
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: master-bin.000003
          Read_Master_Log_Pos: 1070
               Relay_Log_File: slave-relay.000005
                Relay_Log_Pos: 1285
        Relay_Master_Log_File: master-bin.000003
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
```

```
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 1070
              Relay_Log_Space: 2336
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
                  Master_UUID: de6f3fbe-f167-11ed-a7c8-
0242c0a82002
             Master_Info_File: /var/lib/mysql/master.info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay log;
waiting for more updates
           Master_Retry_Count: 86400
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set:
            Executed_Gtid_Set:
                Auto_Position: 0
```

```
         Replicate_Rewrite_DB:
                 Channel_Name:
           Master_TLS_Version:
1 row in set (0.00 sec)
```

12. 主库上创建数据库

```
Shell
mysql> create database test;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
```

13. 连接上任意一个从库，查看从库上数据库自动创建

```
Shell
root@139-159-150-152:/data/maxhou/mysqlcluster# docker exec -it
mysql-slave bash
root@d7cedf19e729:/# mysql -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.36 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current
input statement.
```

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
```

14. 主库上创建表，插入数据

```
Shell
mysql> create table users(sno int,sname varchar(20));
Query OK, 0 rows affected (0.03 sec)

mysql> insert into users values (1,'pony');
Query OK, 1 row affected (0.02 sec)

mysql> insert into users values (2,'maxhou');
Query OK, 1 row affected (0.01 sec)
```

15. 查看从库上数据自动写入

```
Shell
mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| users          |
+----------------+
1 row in set (0.00 sec)


mysql> select *from users;
+------+--------+
| uid  | uname  |
+------+--------+
|    1 | pony   |
|    2 | maxhou |
```

```
+------+--------+
2 rows in set (0.00 sec)
```

至此可以看到我们搭建的 mysql 集群已经能够正常进行工作