mysql backups using replication

if you want to create clean, consistent backups of a mysql (or mariadb) database without stopping the live database, consider using a replica server which you can then shut down, take a backup and then start again. besides producing nice and consistent backups, this will also have almost no performance impact on the live database, especially if myisam tables are used which would block writes with almost any other backup method.

in my case i had a docker-compose.yml which contains the php web app and a mysql server service. i first copied the service in the `docker-compose.yml` file to a `mysql-replica` service, so that the respective sections look like this:

```
  mysql:
#    image: mysql:8.0
    build:
      context: ./mysql/
      dockerfile: Dockerfile
    environment:
      - MYSQL_RANDOM_ROOT_PASSWORD=yes
      - TZ=Europe/Zurich
    restart: always
    volumes:
      - ./data/mysql:/var/lib/mysql
      - ./conf/mysql:/etc/mysql/conf.d
    user: '1002:1002'
    cap_add: [ SYS_NICE ]
    networks:
      - internal
  mysql-replica:
#    image: mysql:8.0
    build:
      context: ./mysql/
      dockerfile: Dockerfile
    environment:
      - MYSQL_RANDOM_ROOT_PASSWORD=yes
      - TZ=Europe/Zurich
    restart: always
    volumes:
      - ./data/mysql-replica:/var/lib/mysql
      - ./conf/mysql-replica:/etc/mysql/conf.d
    user: '1002:1002'
    cap_add: [ SYS_NICE ]
    networks:
      - internal
```

basically make sure both are the same and make sure they can use different data directories and different config files.

for the primary server, make sure the config contains the following settings:

```
[mysqld]
server-id = 1

# binary logging for replication
log_bin = mysql-bin
binlog_format = ROW

# GTID-based replication
gtid_mode = ON
enforce_gtid_consistency = ON
log_slave_updates = ON

# optional but recommended
binlog_expire_logs_seconds = 604800  # 7 days
```

and on the replica the settings should look like this:

```
[mysqld]
server-id = 2

relay_log = relay-bin

# keep binlog on replica too (useful for cascading replication, backups,
etc.)
log_bin = mysql-bin
binlog_format = ROW

gtid_mode = ON
enforce_gtid_consistency = ON
log_slave_updates = ON

# make it read-only for safety
read_only = ON
super_read_only = ON

# don't auto-start replication until we finish setup
skip_slave_start = ON
```

on the primary server which is still running, create a replica user. use a password of max. 32 charcters length.

```
docker-compose exec mysql mysql -uroot -psecretrootpassword

CREATE USER 'repl'@'%' IDENTIFIED BY 'replicationpassword';
GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'repl'@'%';
FLUSH PRIVILEGES;
```

now stop all containers and copa the mysql data directory from the primary to the replica container data storage path:

```
rsync -av data/mysql/ data/mysql-replica/
```

make sure new db server gets new uuid:

```
rm ./data/mysql-replica/auto.cnf
```

now start both database containers. make sure they are both started and running.

set up to replicatoin. connect to the **replica server** first

```
docker-compose exec mysql-replica mysql -uroot -psecretrootpassword

STOP REPLICA;  -- or STOP SLAVE;

CHANGE REPLICATION SOURCE TO
  SOURCE_HOST = 'mysql',
  SOURCE_PORT = 3306,
  SOURCE_USER = 'repl',
  SOURCE_PASSWORD = 'replicationpassword',
  SOURCE_AUTO_POSITION = 1,
  GET_SOURCE_PUBLIC_KEY = 1;

START REPLICA; -- or START SLAVE;
```

check the replication status:

```
SHOW REPLICA STATUS\G   -- or SHOW SLAVE STATUS\G;
```

If all went well, you should see:

```
Replica_IO_Running: Yes
Replica_SQL_Running: Yes
Seconds_Behind_Source: 0 (after it’s caught up)
```

you can also see the the executed and retrieved Gtid dataset, they should update as the master is writing data to the db.

now remove the `skip_slave_start = ON` line or comment it out in the `conf/mysql-replica/replica.cnf` file and restart the container, then re-check if it is still syncing.

Backup and Monitoring

now lets create a backup script. this script will first check if the replication is still running and updated before it shuts down the replica container and creates a tar.gz file of the database data directory. on success, it will notify a uptime kuma push monitor.

for this to work we first need to create a monitoring user **on the master database** which will then auto sync to the read-only replica database as well.

```
docker-compose exec mysql mysql -uroot -psecretrootpassword

CREATE USER 'monitor'@'%' IDENTIFIED BY 'monitor-password';
GRANT REPLICATION CLIENT ON *.* TO 'monitor'@'%';
FLUSH PRIVILEGES;
```

to test if it is working, run the following query **on the replica mysql server** this time:

```
docker-compose exec mysql-replica mysql -umonitor -pmonitor-password -e
"SHOW SLAVE STATUS\G"
```

this should show the same output we saw previously when setting up the replication.

and here is the script that does it all. it only sends a heartbeat update to the uptime kuma push monitor when the backup is successful AND the replica server was replicating prior to stopping it and backing it up. obviously you need to adjust the user parameters at the begining of the script:

```
#!/usr/bin/env bash
set -euo pipefail

########################################
# Settings
########################################

# Directory where your docker-compose.yml lives
COMPOSE_DIR="/path/to/your/project"   # <-- CHANGE THIS

# Name of the replica service in docker-compose
REPLICA_SERVICE="mysql-replica"

# Where the replica's data directory is on the host
REPLICA_DATA_DIR="${COMPOSE_DIR}/data/mysql-replica"

# Where to store backups
BACKUP_DIR="/backup"

# How many backups to keep
MAX_BACKUPS=24

# Uptime Kuma
UPTIME_SERVER="my.server.net"         # <-- CHANGE THIS
ENTITY_ID="YOUR_ENTITY_ID_HERE"       # <-- CHANGE THIS (create new monitor
of type "push" and copy the id from there)

# Monitoring DB user (created on the PRIMARY, replicated to replica)
MONITOR_USER="monitor"
MONITOR_PASSWORD="monitor-password"   # <-- CHANGE THIS or move to env

# State file to track last Executed_Gtid_Set
STATE_DIR="${BACKUP_DIR}/.state"
```

```
LAST_GTID_FILE="${STATE_DIR}/last_gtid"

########################################
# Globals
########################################

START_TS=$(date +%s)
REPL_WARN=0   # 0=OK, 1=replication issue detected

mkdir -p "${BACKUP_DIR}" "${STATE_DIR}"

########################################
# Functions
########################################

check_replication() {
  echo "[$(date)] Checking replication status on ${REPLICA_SERVICE}..."

  # Get SHOW SLAVE STATUS output
  local status
  status=$(docker-compose -f "${COMPOSE_DIR}/docker-compose.yml" exec -T
"${REPLICA_SERVICE}" \
    mysql -u"${MONITOR_USER}" -p"${MONITOR_PASSWORD}" -e "SHOW SLAVE
STATUS\G" 2>/dev/null || true)

  if [[ -z "${status}" ]]; then
    echo "[$(date)] WARNING: SHOW SLAVE STATUS returned nothing. Replication
may not be configured."
    REPL_WARN=1
    return
  fi

  # Parse key fields
  local io_running sql_running seconds_behind executed_gtid

  io_running=$(printf '%s\n' "${status}" | awk -F: '/Slave_IO_Running/
{gsub(/^[ \t]+/, "", $2); print $2}')
  sql_running=$(printf '%s\n' "${status}" | awk -F: '/Slave_SQL_Running/
{gsub(/^[ \t]+/, "", $2); print $2}')
  seconds_behind=$(printf '%s\n' "${status}" | awk -F:
'/Seconds_Behind_Master/ {gsub(/^[ \t]+/, "", $2); print $2}')
  executed_gtid=$(printf '%s\n' "${status}" | awk -F: '/Executed_Gtid_Set/
{sub(/^[ \t]+/, "", $2); print $2}')

  echo "[$(date)] Slave_IO_Running=${io_running},
Slave_SQL_Running=${sql_running}, Seconds_Behind_Master=${seconds_behind}"

  # Basic health check
  if [[ "${io_running}" != "Yes" || "${sql_running}" != "Yes" ]]; then
    echo "[$(date)] WARNING: Replication threads not running correctly."
    REPL_WARN=1
```

```
  fi

  # Check if replication advanced since last backup (via Executed_Gtid_Set)
  if [[ -n "${executed_gtid}" ]]; then
    if [[ -f "${LAST_GTID_FILE}" ]]; then
      local prev_gtid
      prev_gtid=$(<"${LAST_GTID_FILE}")
      if [[ "${prev_gtid}" == "${executed_gtid}" ]]; then
        echo "[$(date)] WARNING: Executed_Gtid_Set unchanged since last
backup — replication may be stuck."
        REPL_WARN=1
      else
        echo "[$(date)] Replication has advanced since last backup."
      fi
    else
      echo "[$(date)] No previous GTID file — this is probably the first
backup."
    fi
    # Update GTID state for next run
    printf '%s\n' "${executed_gtid}" > "${LAST_GTID_FILE}"
  else
    echo "[$(date)] WARNING: Executed_Gtid_Set is empty."
    REPL_WARN=1
  fi
}

send_uptime_ping() {
  local end_ts
  end_ts=$(date +%s)
  local walltime=$((end_ts - START_TS))

  local status="up"
  local msg="OK"

  msg="${msg// /_}"

  local
url="https://${UPTIME_SERVER}/api/push/${ENTITY_ID}?status=${status}&msg=${m
sg}&ping=${walltime}"

  echo "[$(date)] Sending uptime ping: ${url}"
  curl -fsS "${url}" >/dev/null 2>&1 || echo "[$(date)] WARNING: failed to
send uptime ping"
}

on_exit() {
  local exit_code="$1"
  echo "[$(date)] Starting ${REPLICA_SERVICE} container again..."
  docker-compose -f "${COMPOSE_DIR}/docker-compose.yml" start
"${REPLICA_SERVICE}" || \
```

```
    echo "[$(date)] WARNING: failed to start ${REPLICA_SERVICE}"

  # Only send ping if:
  #  - script exited successfully (exit_code == 0)
  #  - replication checks reported no issues (REPL_WARN == 0)
  if [[ "${exit_code}" -eq 0 && "${REPL_WARN}" -eq 0 ]]; then
    send_uptime_ping
  else
    echo "[$(date)] Not sending uptime ping (exit_code=${exit_code},
REPL_WARN=${REPL_WARN})."
  fi
}

trap 'on_exit $?' EXIT

########################################
# Main
########################################

echo "[$(date)] Starting MySQL replica backup..."

cd "${COMPOSE_DIR}"

# 1) Check replication health *before* stopping the replica
check_replication

# 2) Stop replica container (no password needed here)
echo "[$(date)] Stopping ${REPLICA_SERVICE} container..."
docker-compose stop "${REPLICA_SERVICE}"

# 3) Create tar.bz2 backup
TIMESTAMP="$(date +%Y%m%d-%H%M%S)"
BACKUP_FILE="${BACKUP_DIR}/mysql-replica-${TIMESTAMP}.tar.bz2"

echo "[$(date)] Creating tar.bz2 from ${REPLICA_DATA_DIR} ->
${BACKUP_FILE}..."
tar -cjf "${BACKUP_FILE}" -C "${REPLICA_DATA_DIR}" .

echo "[$(date)] Backup archive created: ${BACKUP_FILE}"

# 4) Rotate old backups
echo "[$(date)] Rotating old backups (keeping last ${MAX_BACKUPS})..."
mapfile -t BACKUPS < <(ls -1t "${BACKUP_DIR}"/mysql-replica-*.tar.bz2
2>/dev/null || true)

if (( ${#BACKUPS[@]} > MAX_BACKUPS )); then
  for f in "${BACKUPS[@]:MAX_BACKUPS}"; do
    echo "[$(date)] Deleting old backup: ${f}"
    rm -f -- "${f}"
  done
fi
```

```
echo "[$(date)] Backup finished successfully."
```

NOTE you may want to use gz instead of bz2 in case it takes too long

here is the cron job:

```
0 */2 * * * /opt/dbbackup.sh >> /var/log/dbbackup.log 2>&1
```

add the dbbackup.log to your logrotate config

From:
http://wiki.psuter.ch/ - **pswiki**

Permanent link:
http://wiki.psuter.ch/doku.php?id=mysql_backups_using_replication&rev=1762996119

Last update: **13.11.2025 02:08**