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: mysgl: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:

```
[mysald]
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
qtid mode = ON
enforce_gtid_consistency = ON
log slave updates = ON
# make it read-only for safety
read only = 0N
super read only = 0N
# don't auto-start replication until we finish setup
skip slave start = 0N
```

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:

Printed on 13.11.2025 12:22 http://wiki.psuter.ch/

```
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.

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
```

GRANT REPLICATION CLIENT ON *.* TO 'monitor'@'%';

FLUSH PRIVILEGES;

```
CREATE USER 'monitor'@'%' IDENTIFIED BY 'monitor-password';
```

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 beginning 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="${COMPOSE DIR}/data/mysql-replica"
# Where to store backups
BACKUP DIR="/backup"
# How many backups to keep
MAX BACKUPS=24
# Uptime Kuma
                                  # <-- CHANGE THIS
UPTIME SERVER="my.server.net"
ENTITY_ID="YOUR ENTITY ID HERE"
                                 # <-- CHANGE THIS (create new monitor
of type "push" and copy the id from there)
# Monitoring DB user (created on the PRIMARY, replicated to replica)
MONITOR USER="monitor"
MONITOR PASSWORD="monitor-password" # <-- CHANGE THIS or move to env
# State file to track last Executed Gtid Set
STATE DIR="${BACKUP DIR}/.state"
LAST GTID FILE="${STATE DIR}/last gtid"
```

http://wiki.psuter.ch/ Printed on 13.11.2025 12:22

```
# Globals
START TS=$(date +%s)
REPL WARN=0 # 0=0K, 1=replication issue detected
mkdir -p "${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}")</pre>
      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=${m
sq}&ping=${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}"
```

http://wiki.psuter.ch/ Printed on 13.11.2025 12:22

```
# 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</pre>
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."
```

From:

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

Permanent link:

http://wiki.psuter.ch/doku.php?id=mysql_backups_using_replication&rev=1762989188

Last update: **13.11.2025 00:13**



http://wiki.psuter.ch/ Printed on 13.11.2025 12:22