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 retreived Gtid dataset, they should update as the master is writing data to the db.