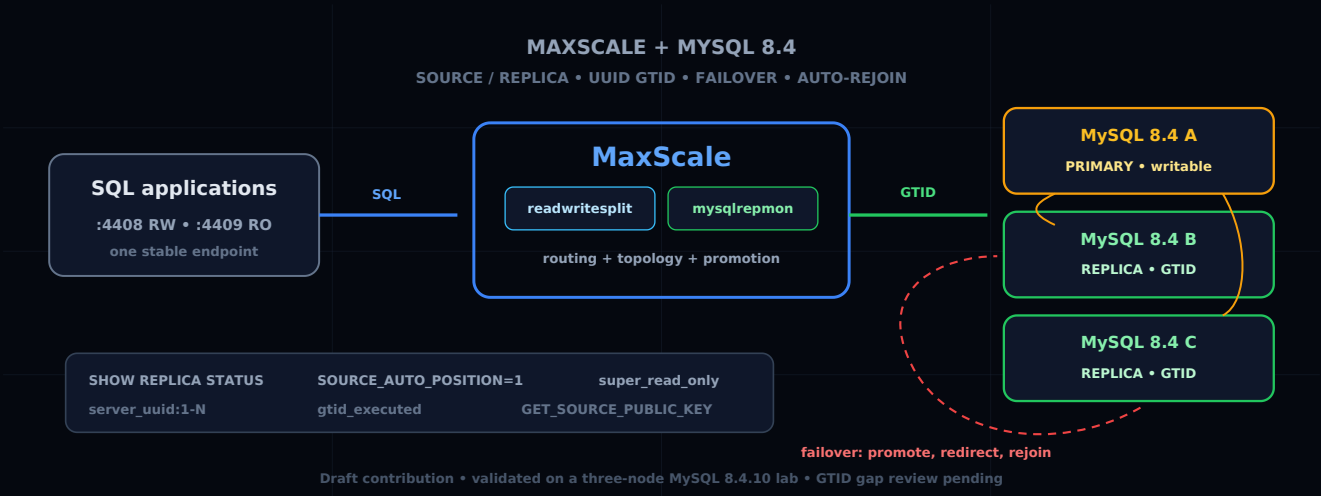
MaxScale et MySQL 8.4 : construire un failover GTID compatible Source/Replica

Aurélien LEQUOY · 27 juillet 2026

MAXSCALE MYSQL MYSQL-8.4 REPLICATION GTID FAILOVER HIGH-AVAILABILITY PROXY



> ***Statut du projet — 27 juillet 2026.*** *La PR upstream #421 propose pour MaxScale le module `mysqlrepmon` décrit dans cet article. La PR est volontairement en draft : le code a été compilé et testé sur un lab MySQL 8.4.10, mais il n'est pas encore inclus dans une version officielle de MaxScale. Ne le présentez pas comme une fonctionnalité supportée en production par l'éditeur.*

Le problème en une phrase

MaxScale sait historiquement superviser des topologies de réplication MariaDB / MySQL avec `mariadbmon`. Mais MySQL 8.4 a supprimé les anciennes commandes `SLAVE` / `MASTER`, utilise des GTID basés sur des UUID et exige une autre grammaire pour reconfigurer une réplication.

Le résultat est trompeur : un proxy peut continuer à accepter des connexions SQL alors que son moniteur ne sait plus lire correctement la topologie, promouvoir une replica ou rattacher l'ancien primary.

Il faut distinguer trois fonctions :

1. **observer** la topologie et l'état des threads de réplication ;
2. **modifier** la topologie lors d'un switchover, d'un failover ou d'un rejoin ;
3. **router** les sessions applicatives vers le primary courant et les replicas disponibles.

La compatibilité MySQL 8.4 doit être correcte sur les trois plans. Changer uniquement `SHOW SLAVE STATUS` en `SHOW REPLICA STATUS` ne suffit pas.

Pourquoi MySQL 8.4 casse l'hypothèse historique

MySQL 8.0.22 avait introduit la terminologie Source/Replica tout en conservant plusieurs alias historiques. MySQL 8.4 termine la transition : les commandes dépréciées ont été retirées.

Opération	MariaDB / ancien dialecte	MySQL 8.4
Lire l'état d'une replica	`SHOW SLAVE STATUS`	`SHOW REPLICA STATUS`
Configurer la source	`CHANGE MASTER TO`	`CHANGE REPLICATION SOURCE TO`
Démarrer la réplication	`START SLAVE`	`START REPLICA`
Arrêter la réplication	`STOP SLAVE`	`STOP REPLICA`
Effacer la configuration	`RESET SLAVE`	`RESET REPLICA`
Lire le statut du binaire	`SHOW MASTER STATUS`	`SHOW BINARY LOG STATUS`
Réinitialiser les GTID	`RESET MASTER`	`RESET BINARY LOGS AND GTIDS`

Les colonnes retournées changent également :

Ancien nom	Nom MySQL 8.4
`Master_Host`	`Source_Host`
`Master_Port`	`Source_Port`
`Master_Server_Id`	`Source_Server_Id`
`Slave_IO_Running`	`Replica_IO_Running`
`Slave_SQL_Running`	`Replica_SQL_Running`
`Seconds_Behind_Master`	`Seconds_Behind_Source`

Ancien nom	Nom MySQL 8.4
`Master_Log_File`	`Source_Log_File`
`Read_Master_Log_Pos`	`Read_Source_Log_Pos`
`Exec_Master_Log_Pos`	`Exec_Source_Log_Pos`
`Channel_Name`	`Channel_Name`

Un moniteur qui envoie encore `SHOW SLAVE STATUS` échoue immédiatement avec une erreur de syntaxe. Un moniteur qui envoie la bonne requête mais continue à chercher `Slave_IO_Running` conclut à tort que la réplication n'est pas active.

Le vrai écart : le modèle GTID

La différence la plus structurante n'est pas le vocabulaire SQL. C'est la représentation des transactions.

GTID MariaDB

MariaDB représente un GTID sous la forme :

```
domain_id-server_id-sequence
```

Exemple :

```
0-101-7842
```

Le domaine fait partie du modèle. Le moniteur MariaDB sait comparer ces positions et utilise notamment `MASTER_USE_GTID`.

GTID MySQL

MySQL représente un ensemble de GTID par UUID de serveur et intervalles :

```
155fa720-7623-11f1-9a78-bc241174cab8:1-76
```

Un historique peut contenir plusieurs UUID et des intervalles discontinus :

```
155fa720-7623-11f1-9a78-bc241174cab8:1-10:20-30,
7fd8c42a-7630-11f1-99af-bc241174cab8:1-8
```

MySQL expose notamment :

- `@@global.server_uuid` pour identifier l'origine ;
- `@@global.gtid_executed` pour les transactions appliquées ;
- `@@global.gtid_purged` pour les GTID dont les binlogs ont été purgés ;
- `Retrieved_Gtid_Set` dans `SHOW REPLICA STATUS` pour ce qui a été reçu.

Le changement de source se fait avec l'auto-positionnement :

```
CHANGE REPLICATION SOURCE TO
    SOURCE_HOST = '10.0.0.12',
    SOURCE_PORT = 3306,
    SOURCE_USER = 'repl',
    SOURCE_PASSWORD = 'secret',
    SOURCE_AUTO_POSITION = 1,
    GET_SOURCE_PUBLIC_KEY = 1
FOR CHANNEL '';
```

Le serveur choisit alors le bon point de reprise à partir des ensembles GTID, sans fournir un fichier et une position de binlog.

Pourquoi un module `mysqlrepmon`

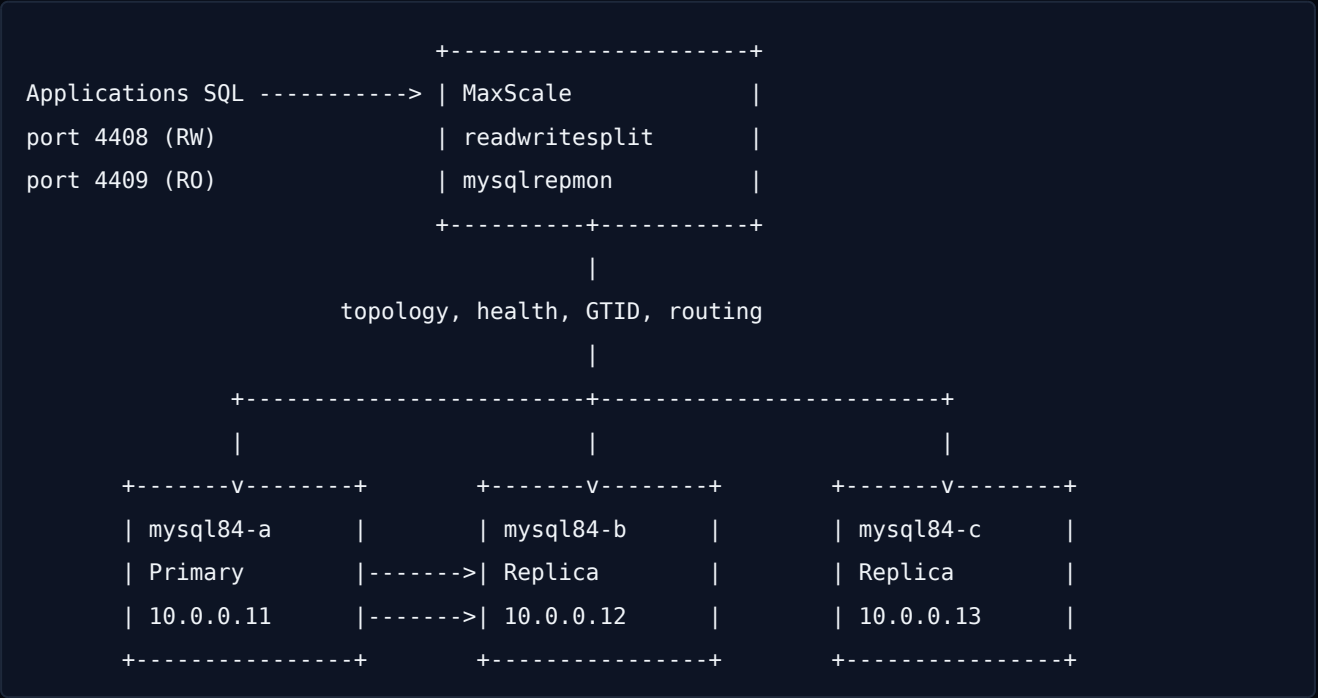
À côté de `mariadbmon`, la contribution ajoute un moniteur dédié à MySQL. Il réutilise le moteur éprouvé de décision et de manipulation de cluster, mais remplace les feuilles spécifiques au serveur :

- lecture de `SHOW REPLICA STATUS` et des colonnes `Source_*` / `Replica_*` ;
- lecture de `server_uuid`, `gtid_executed` et `gtid_purged` ;
- utilisation de `CHANGE REPLICATION SOURCE TO ... SOURCE_AUTO_POSITION=1` ;
- commandes `START`, `STOP` et `RESET REPLICA ... FOR CHANNEL` ;
- activation de `super_read_only=1` lors d'une rétrogradation ;
- utilisation de `RESET BINARY LOGS AND GTIDS` pour la commande manuelle de reset ;
- vérification de `enforce_gtid_consistency` et `log_replica_updates`.

Le module conserve les paramètres opérationnels connus de `mariadbmon` : `auto_failover`, `auto_rejoin`, `enforce_read_only_slaves`, commandes de switchover, failover et rejoin.

Architecture de référence

L'architecture testée comporte trois serveurs MySQL 8.4 et un nœud MaxScale :



Les applications ne connaissent jamais l'adresse du primary. Elles utilisent le listener MaxScale. Le moniteur décide quel serveur porte le rôle `Master` interne à MaxScale ; `readwritesplit` envoie les écritures vers ce serveur et distribue les lectures.

> *MaxScale doit lui-même être redondé. Un failover parfait des bases ne sert à rien si l'unique proxy SQL est un point de panne. En production, prévoyez au minimum deux MaxScale en actif/passif, un VIP ou un load balancer, et le mécanisme de verrouillage coopératif adapté.*

1. Préparer chaque serveur MySQL 8.4

Chaque serveur susceptible d'être promu doit pouvoir produire ses propres binlogs et relayer les transactions reçues.

Configuration de base :

```
[mysqld]
server_id=101
log_bin=mysql-bin
binlog_format=ROW

gtid_mode=ON
enforce_gtid_consistency=ON
log_replica_updates=ON

relay_log_recovery=ON
```

Utilisez un `server_id` différent sur chaque nœud : `101`, `102`, `103`, par exemple.

Le `server_uuid` est lui aussi obligatoirement unique. Il est conservé dans le fichier `auto.cnf` du datadir. Si vous clonez une machine ou un datadir, ne déployez jamais plusieurs serveurs avec le même `server_uuid`.

Sur les replicas :

```
read_only=ON
super_read_only=ON
```

Sur le primary :

```
read_only=OFF
super_read_only=OFF
```

Après redémarrage, contrôlez les invariants :

```
SELECT
    @@hostname,
    @@server_id,
    @@server_uuid,
    @@gtid_mode,
    @@enforce_gtid_consistency,
    @@log_bin,
    @@log_replica_updates,
    @@read_only,
    @@super_read_only\G

SELECT @@global.gtid_executed\G
```

```
SHOW BINARY LOG STATUS\G
SHOW REPLICA STATUS\G
```

Une replica promotable doit avoir `log_bin=1`, `log_replica_updates=1`, les deux threads de réplication actifs et un lag maîtrisé.

2. Créer des comptes séparés

Séparez au minimum :

- le compte de monitoring et de manipulation de topologie ;
- le compte utilisé par les replicas pour lire les binlogs ;
- les comptes applicatifs qui passent par le proxy.

Compte du moniteur

Pour l'observation seule :

```
CREATE USER 'maxscale_mon'@'10.0.0.10'
    IDENTIFIED BY 'a-long-random-password';

GRANT REPLICATION CLIENT
    ON *.* TO 'maxscale_mon'@'10.0.0.10';
```

Pour le failover, le switchover et le rejoin, le moniteur doit aussi pouvoir arrêter et reconfigurer la réplication, modifier les variables read-only et contrôler les connexions :

```
GRANT REPLICATION SLAVE,
      REPLICATION CLIENT,
      PROCESS,
      SUPER,
      SYSTEM_VARIABLES_ADMIN,
      REPLICATION_SLAVE_ADMIN,
      CONNECTION_ADMIN
    ON *.* TO 'maxscale_mon'@'10.0.0.10';
```

Le privilège historique `SUPER` reste demandé par certains chemins compatibles. Sur une version finalisée du module, réévaluez la liste exacte et retirez tout privilège devenu inutile.

Compte de réplication

```
CREATE USER 'repl'@'10.0.0.%'
    IDENTIFIED BY 'another-long-random-password'
    REQUIRE SSL;

GRANT REPLICATION SLAVE
    ON *.* TO 'repl'@'10.0.0.%';
```

Privilégiez TLS. Avec l'authentification `caching_sha2_password` par défaut de MySQL 8.4 et une connexion non chiffrée, la commande de changement de source doit fournir `GET_SOURCE_PUBLIC_KEY=1` ou un chemin vers la clé publique RSA. Le module ajoute cette option si TLS n'est pas activé.

Compte de service pour l'authentification des clients

Le `user` d'un service MaxScale n'est pas le compte sous lequel toutes les requêtes applicatives sont exécutées. Il permet au User Account Manager de MaxScale de charger les comptes et leurs droits depuis les backends :

```
CREATE USER 'maxscale_route'@'10.0.0.10'
    IDENTIFIED BY 'route-password';

GRANT SELECT ON mysql.user
    TO 'maxscale_route'@'10.0.0.10';
GRANT SELECT ON mysql.db
    TO 'maxscale_route'@'10.0.0.10';
GRANT SELECT ON mysql.tables_priv
    TO 'maxscale_route'@'10.0.0.10';
GRANT SELECT ON mysql.columns_priv
    TO 'maxscale_route'@'10.0.0.10';
GRANT SELECT ON mysql.procs_priv
    TO 'maxscale_route'@'10.0.0.10';
GRANT SELECT ON mysql.proxies_priv
    TO 'maxscale_route'@'10.0.0.10';
GRANT SHOW DATABASES ON *.*
    TO 'maxscale_route'@'10.0.0.10';
```

Les comptes applicatifs doivent toujours exister sur les serveurs MySQL avec le même secret et des grants cohérents. Comme les backends voient la connexion arriver depuis MaxScale, leur partie `@host` doit autoriser l'adresse du proxy.

3. Construire la contribution à un SHA reproductible

La PR étant encore en draft, le paquet MaxScale standard ne contient pas `mysqlrepmon`. Pour un lab uniquement, construisez le SHA public exact :

```
git clone https://github.com/mariadb-corporation/MaxScale.git
cd MaxScale

git fetch https://github.com/Esysteme/MaxScale.git \
  aurelien/mysql-8.4-replication-support
git checkout --detach f61776a5b19cf295636c94a8a3dea6d81fcd61fb

cmake -S . -B build \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_INSTALL_PREFIX=/usr/local/maxscale-mysql84

cmake --build build --target mariadbmon mysqlrepmon test_cycle_find -j2
ctest --test-dir build --output-on-failure \
  -R '^test_mariadbmon_cycle_find$'

cmake --build build -j2
sudo cmake --install build
```

Le build ciblé doit produire `libmysqlrepmon.so`. La correction CMake de la contribution réutilise explicitement `LIBSSH_LIBRARY` et `LIBSSH_INCLUDE_DIR` détectés par MaxScale au lieu de supposer l'existence d'une cible interne `libssh`.

Démarrer le build isolé

Le préfixe `/usr/local/maxscale-mysql84` évite d'écraser le paquet installé. En contrepartie, le service `maxscale.service` du paquet continue de lancer `/usr/bin/maxscale` et ne chargera pas le nouveau module. Pour le lab, créez `/etc/systemd/system/maxscale-mysql84.service` :

```
[Unit]
Description=MaxScale MySQL 8.4 compiled build
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
```

```
User=maxscale
Group=maxscale
RuntimeDirectory=maxscale
RuntimeDirectoryMode=0755
ExecStart=/usr/local/maxscale-mysql84/bin/maxscale --nodaemon --config=/etc/maxscale.cnf --
logdir=/var/log/maxscale --datadir=/var/lib/maxscale --cachedir=/var/cache/maxscale --
libdir=/usr/local/maxscale-mysql84/lib/maxscale --piddir=/run/maxscale
Restart=on-failure
RestartSec=5s
LimitNOFILE=65535

[Install]
WantedBy=multi-user.target
```

Le compte système `maxscale` et les répertoires de données doivent exister ; ils sont normalement créés par le paquet MaxScale installé pour fournir l'environnement d'exécution. Rechargez ensuite systemd :

```
sudo install -d -o maxscale -g maxscale -m 0755 \
  /var/lib/maxscale /var/cache/maxscale /var/log/maxscale
sudo systemctl daemon-reload
```

4. Configurer MaxScale

Exemple complet minimal :

```
[maxscale]
threads=auto
admin_host=127.0.0.1
admin_port=8989

[mysql84-a]
type=server
address=10.0.0.11
port=3306
protocol=MariaDBBackend

[mysql84-b]
type=server
address=10.0.0.12
```

```
port=3306
protocol=MariaDBBackend

[mysql84-c]
type=server
address=10.0.0.13
port=3306
protocol=MariaDBBackend

[MySQL84-Monitor]
type=monitor
module=mysqlrepmon
servers=mysql84-a,mysql84-b,mysql84-c
user=maxscale_mon
password=a-long-random-password
monitor_interval=2000ms

assume_unique_hostnames=true
auto_failover=safe
auto_rejoin=true
enforce_read_only_slaves=true

replication_user=repl
replication_password=another-long-random-password
replication_master_ssl=true

[MySQL84-RW-Service]
type=service
router=readwritesplit
servers=mysql84-a,mysql84-b,mysql84-c
user=maxscale_route
password=route-password

master_reconnection=true
master_failure_mode=fail_on_write

[MySQL84-RW-Listener]
type=listener
service=MySQL84-RW-Service
protocol=MariaDBClient
port=4408
```

```
[MySQL84-RO-Service]
type=service
router=readconnroute
servers=mysql84-a,mysql84-b,mysql84-c
user=maxscale_route
password=route-password
router_options=slave

[MySQL84-RO-Listener]
type=listener
service=MySQL84-RO-Service
protocol=MariaDBClient
port=4409
```

Les mots de passe sont laissés en clair ici uniquement pour rendre l'exemple lisible. En exploitation, utilisez le chiffrement de secrets proposé par MaxScale, des permissions strictes sur le fichier de configuration et une rotation documentée.

`assume_unique_hostnames=true` est requis pour autoriser `auto_failover` et `auto_rejoin`. Chaque backend doit donc exposer une adresse ou un hostname unique, stable et identique à celui que les replicas enregistrent comme source ; utilisez `private_address` si le réseau de réplication emploie une autre adresse.

Pourquoi `auto_failover=safe` pour commencer ? Ce mode refuse l'opération si le moniteur détecte qu'une promotion entraînerait clairement une perte de transactions. Il ne transforme pas une réplication asynchrone en réplication synchrone, mais ajoute une barrière utile pendant l'évaluation.

Pourquoi `master_reconnection=true` et `master_failure_mode=fail_on_write` ? Une session `readwritesplit` peut conserver son contexte et se reconnecter au nouveau primary tant qu'aucune écriture n'arrive pendant la fenêtre sans primary et qu'aucune transaction n'est ouverte. Sans ces paramètres, la base peut réussir son failover tandis que toutes les sessions applicatives sont quand même coupées.

5. Vérifier avant le premier test

Validez la configuration puis démarrez MaxScale :

```
sudo /usr/local/maxscale-mysql84/bin/maxscale \
  --config=/etc/maxscale.cnf \
  --libdir=/usr/local/maxscale-mysql84/lib/maxscale \
  --config-check

sudo systemctl disable --now maxscale.service 2>/dev/null || true
sudo systemctl enable --now maxscale-mysql84.service
systemctl --no-pager --full status maxscale-mysql84.service
```

Vérifiez que le module et la topologie sont visibles :

```
maxctrl list modules | grep -E 'mariadbmon|mysqlrepmon'
maxctrl list servers
maxctrl list monitors
maxctrl show monitor MySQL84-Monitor
```

État attendu :

- un serveur `Master, Running` ;
- deux serveurs `Slave, Running` ;
- aucun serveur `Down` ;
- le monitor actif ;
- les listeners `4408` et `4409` ouverts.

Contrôlez aussi directement MySQL :

```
mysql -h 10.0.0.12 -e "SHOW REPLICA STATUS\\G"
mysql -h 10.0.0.13 -e "SHOW REPLICA STATUS\\G"
```

Les champs essentiels sont :

```
Replica_IO_Running: Yes
Replica_SQL_Running: Yes
Seconds_Behind_Source: 0
Auto_Position: 1
```

6. Comprendre le déroulement d'un failover

Quand le primary disparaît, le moniteur ne se contente pas de changer une étiquette :

1. il attend le nombre de cycles défini par `failcount` et vérifie autant que possible que la panne est réelle ;
2. il compare l'état des replicas et choisit une candidate promotable ;
3. il arrête la réplication sur la candidate ;
4. il désactive `read_only` sur le nouveau primary ;
5. il redirige les autres replicas avec `CHANGE REPLICATION SOURCE TO ... SOURCE_AUTO_POSITION=1` ;
6. il redémarre leurs threads de réplication ;
7. `readwritesplit` détecte le nouveau rôle et route les prochaines écritures vers le nouveau primary.

Pour rétrograder un primary, `mysqlrepmon` utilise :

```
SET GLOBAL super_read_only = 1;
```

C'est plus fort que `read_only=1` : même un compte doté de privilèges administratifs ne doit pas continuer à écrire accidentellement sur l'ancien primary.

7. Tester un switchover contrôlé

Avant de simuler une panne brutale, validez un switchover :

```
maxctrl call command mysqlrepmon switchover \
  MySQL84-Monitor mysql84-b mysql84-a
```

Puis contrôlez :

```
maxctrl list servers

mysql -h 10.0.0.11 -e \
  "SELECT @@hostname, @@read_only, @@super_read_only, @@global.gtid_executed\\G"
mysql -h 10.0.0.12 -e \
  "SELECT @@hostname, @@read_only, @@super_read_only, @@global.gtid_executed\\G"
mysql -h 10.0.0.13 -e \
  "SELECT @@hostname, @@read_only, @@super_read_only, @@global.gtid_executed\\G"
```

Le nouveau primary doit être writable. Les deux autres nœuds doivent être en `super_read_only=1` et répliquer depuis lui.

8. Tester une panne réelle sans tricher

Créez d'abord une donnée via MaxScale :

```
CREATE DATABASE maxscale_ha_test;
CREATE TABLE maxscale_ha_test.events (
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    source_hostname VARCHAR(255) NOT NULL,
    created_at TIMESTAMP(6) NOT NULL DEFAULT CURRENT_TIMESTAMP(6),
    PRIMARY KEY (id)
) ENGINE=InnoDB;

INSERT INTO maxscale_ha_test.events(source_hostname)
VALUES (@@hostname);
```

Vérifiez qu'elle existe sur les replicas, puis arrêtez le primary au niveau service ou machine. Ne mettez pas simplement le serveur en maintenance dans MaxScale : cela teste une décision administrative, pas la détection d'une panne.

Pendant le test :

```
watch -n 1 'maxctrl list servers'
```

Dans un autre terminal :

```
journalctl -u maxscale -f
```

Critères de réussite :

- un seul nouveau primary est élu ;
- les autres replicas pointent vers lui ;
- une écriture via le listener `4408` réussit après la promotion ;
- les lectures via `4409` restent cohérentes ;
- l'ancien primary rejoint comme replica après son retour ;
- les ensembles `gtid_executed` convergent.

Ce que le lab MySQL 8.4.10 a validé

Le scénario a été exécuté sur trois nœuds MySQL 8.4.10 :

- détection initiale du primary et de deux replicas ;
- arrêt du primary ;
- promotion d'une replica ;
- redirection de la replica restante vers la nouvelle source ;
- retour de l'ancien primary et rejoin automatique comme replica ;
- second failover dans l'autre sens ;
- convergence finale du même ensemble `gtid_executed` sur les trois serveurs ;
- routage lecture/écriture valide via MaxScale ;
- refus des écritures sur le listener read-only.

Ce test valide le chemin fonctionnel sur des historiques GTID continus. Il ne prouve pas encore tous les cas limites nécessaires à une intégration upstream.

Authentification : ne pas confondre deux pannes

Le build MaxScale déployé dans le lab rejetait les comptes applicatifs utilisant le hash `caching_sha2_password` de MySQL 8.4 :

```
Stored password hash length is 70 when 40 was expected
```

Ce message provient de l'authenticator du protocole MaxScale utilisé dans ce build, pas du moniteur de réplication. La réplication et le routage peuvent être sains alors que l'authentification client échoue.

Dans le lab, un compte `mysql_native_password` explicitement dédié au probe a permis d'isoler le problème. Ce n'est pas une recommandation générale : MySQL 8.4 désactive ce plugin historique par défaut. En production, utilisez un build et un authenticator MaxScale compatibles avec `caching_sha2_password`, ou une méthode d'authentification officiellement supportée, au lieu d'affaiblir globalement la configuration MySQL.

Même règle pour :

```
401 Unauthorized
```

sur le port REST : cela signifie que `maxctrl` n'a pas les bons identifiants d'administration. Ce n'est pas une preuve que le monitor ou le routage SQL est en panne.

Tableau de diagnostic

Symptôme	Cause probable	Vérification
Erreur de syntaxe sur `SHOW SLAVE STATUS`	ancien module ou dialecte MariaDB utilisé contre MySQL 8.4	log MaxScale, requête réellement envoyée
Serveur affiché sans rôle replica	colonnes `Replica_*` non mappées	`SHOW REPLICA STATUS\G` direct
Replica non promotable	`log_bin`, `log_replica_updates` ou GTID désactivé	variables globales et log monitor
Rejoin refusé	transactions errantes ou historique incompatible	comparer `gtid_executed` / `gtid_purged`
`Stored password hash length is 70...`	authenticator incompatible avec `caching_sha2_password`	log MaxScale, plugin du compte
`401 Unauthorized` avec `maxctrl`	identifiants REST incorrects	configuration admin MaxScale
Session coupée malgré la promotion	reconnexion router impossible, transaction ouverte ou historique de session dépassé	paramètres `master_reconnection`, `master_failure_mode`, log router
Deux primaries writable	fencing/read-only insuffisant ou monitors concurrents	`super_read_only`, locks coopératifs, état réseau

Limite connue : les gaps dans les ensembles GTID

Pour réutiliser le moteur interne de `mariadbmon`, la proposition associe chaque UUID MySQL à un domaine synthétique et convertit ses intervalles vers la représentation interne existante.

Dans l'état actuel, elle conserve le numéro de transaction le plus élevé par UUID. Ainsi :

```
uuid:1-10:20-30
```

est résumé comme si la position atteignait `30`. L'information « 11 à 19 sont absents » n'est pas représentée fidèlement.

Sur le lab, les historiques étaient continus. Sur une infrastructure ayant des transactions injectées, filtrées, purgées ou des GTID errants, cette approximation peut fausser la comparaison des candidates.

C'est la raison principale du statut draft de la PR. Avant de considérer le module comme production-ready, il faut :

- représenter ou comparer correctement les intervalles discontinus ;
- ajouter des tests unitaires avec plusieurs UUID et des gaps ;
- couvrir les GTID errants et les historiques purgés ;
- exécuter la CI upstream complète ;
- obtenir la revue des mainteneurs MaxScale.

Ce que le failover ne garantit pas

Un moniteur n'abolit pas les propriétés de la réplication asynchrone.

- **RPO nul non garanti** : une transaction confirmée sur l'ancien primary peut ne pas avoir atteint une replica.
- **Transactions ouvertes** : une session en transaction ne peut pas toujours être déplacée sans erreur.
- **Split-brain** : si l'ancien primary reste accessible à une partie des applications, il faut un vrai fencing réseau ou système.
- **Topologies complexes** : multi-source, réplication circulaire et relais demandent une stratégie spécifique.
- **Données divergentes** : `auto_rejoin` ne doit pas écraser silencieusement des transactions errantes.
- **Proxy unique** : MaxScale doit être redondé séparément.

Un bon test de haute disponibilité mesure donc quatre choses distinctes : détection, promotion, convergence des données et continuité applicative.

Checklist avant production

- [ ] sauvegarde restaurable testée ;
- [ ] trois `server_id` et `server_uuid` uniques ;
- [ ] GTID et `log_replica_updates` activés partout ;
- [ ] TLS entre MaxScale et les backends ;
- [ ] comptes monitor, replication et application séparés ;
- [ ] `super_read_only=1` vérifié sur toutes les replicas ;
- [ ] switchover contrôlé validé avant le failover brutal ;
- [ ] test d'écriture via MaxScale après promotion ;
- [ ] test de rejoin de l'ancien primary ;
- [ ] comparaison finale des ensembles `gtid_executed` ;
- [ ] comportement des transactions applicatives documenté ;
- [ ] fencing et redondance de MaxScale testés ;
- [ ] alerte sur toute promotion et toute divergence GTID ;
- [ ] module upstream finalisé, relu et supporté avant production.

Pourquoi publier cette contribution

MySQL 8.4 est une LTS. Les commandes historiques ne reviendront pas. Maintenir indéfiniment des alias dans les outils de supervision ne résout pas l'écart de modèle GTID ni la grammaire de reconfiguration.

Un module dédié rend la frontière explicite :

- `mariadbmon` reste natif pour le dialecte et les GTID MariaDB ;
- `mysqlrepmon` porte les règles propres à MySQL 8.x ;
- le moteur commun de failover reste partagé ;
- les tests peuvent vérifier chaque famille sans multiplier les conditions dispersées.

La PR MaxScale #421 contient le code, la documentation, les commandes de validation, le résultat du lab et la limite GTID connue. L'objectif du draft est précisément d'obtenir une revue sur cette représentation avant de prétendre que tous les historiques MySQL sont sûrs.

Pour aller plus loin

- MySQL 8.4 — nouveautés et commandes de réplication supprimées
- MySQL 8.4 — configuration de la réplication
- MySQL 8.4 — changement de source et `GET_SOURCE_PUBLIC_KEY`
- MaxScale — paramètres de MariaDB Monitor
- MaxScale — routeur readwritesplit
- Installer MySQL 8.4 sur Debian 13
- Comprendre le passage de Master/Slave à Source/Replica

Vous voulez voir ce support dans MaxScale ?

Si cette compatibilité MySQL 8.4 vous serait utile, venez lire la proposition, tester la branche et apporter votre soutien directement sur la PR. Les retours de terrain, les cas GTID complexes et les revues techniques aideront à transformer ce draft en une contribution réellement intégrable :

Soutenir la PR MaxScale #421