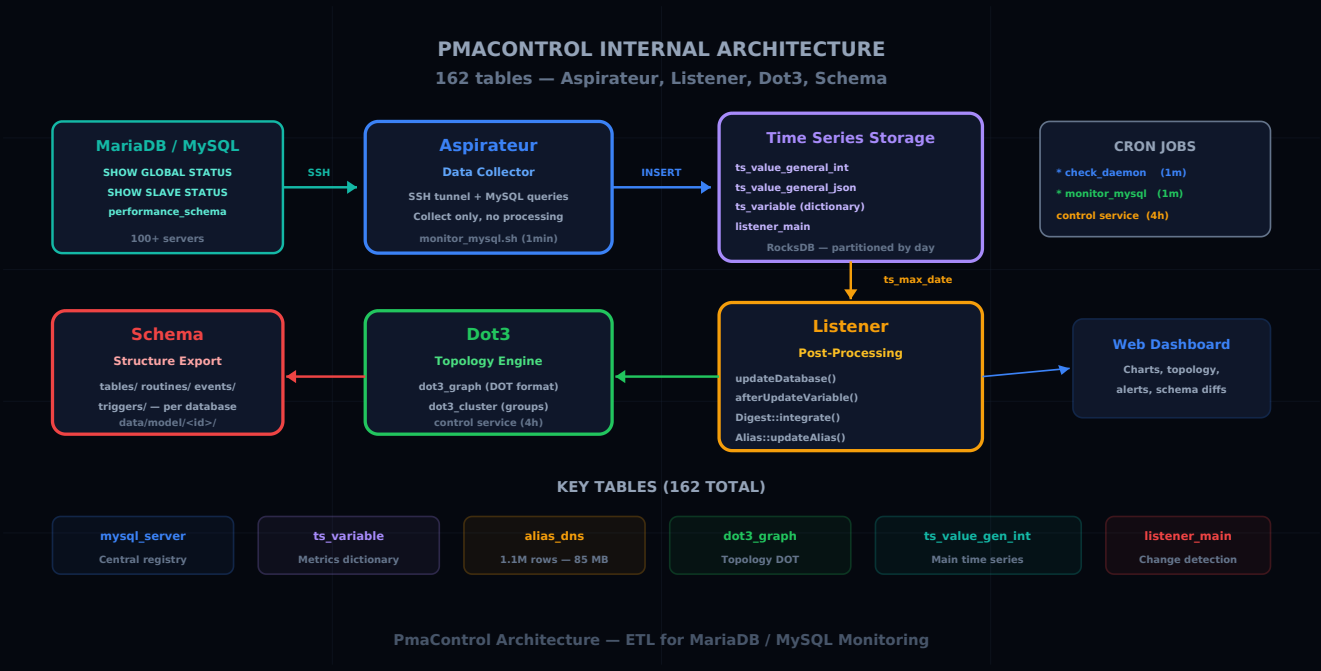
Architecture interne PmaControl : Aspirateur, Listener, Dot3 et 162 tables

Aurélien LEQUOY · 13 avril 2026

PMACONTROL ARCHITECTURE AGENTS CRON MONITORING



162 tables, pas une de trop

PmaControl n'est pas un simple tableau de bord. C'est un système distribué qui collecte, stocke, transforme et expose les métriques de centaines de serveurs MariaDB / MySQL en temps réel. La base de données interne contient **162 tables** — chacune avec un rôle précis dans le pipeline de données.

Cet article détaille l'architecture interne : les quatre composants principaux (Aspirateur, Listener, Dot3, Schema), le flux de données de bout en bout, les crons qui orchestrent le tout, et les tables clés à connaître pour déboguer ou étendre le système.

Les quatre piliers

Aspirateur : le collecteur de données

L'Aspirateur est le composant qui va chercher les métriques sur chaque serveur supervisé. Son fonctionnement est simple mais efficace :

1. Il se connecte au serveur via **SSH** (tunnel) puis ouvre une connexion **MySQL** locale
2. Il exécute une série de requêtes : `SHOW GLOBAL STATUS`, `SHOW GLOBAL VARIABLES`, `SHOW SLAVE STATUS`, `SHOW PROCESSLIST`, des requêtes sur `performance_schema`, etc.
3. Il écrit les résultats dans les tables `ts_value_*` (time series) de la base PmaControl

Le préfixe `ts_` est omniprésent : il signifie **time series**. Chaque métrique est horodatée et stockée avec l'identifiant du serveur source.

```
Aspirateur → SSH tunnel → MySQL local
           → SHOW GLOBAL STATUS
           → SHOW SLAVE STATUS
           → performance_schema queries
           → INSERT INTO ts_value_general_int (...)
           → INSERT INTO ts_value_general_json (...)
```

L'Aspirateur ne fait **aucun traitement**. Il collecte et écrit. C'est un principe de conception fondamental : séparer la collecte du traitement pour pouvoir les scaler indépendamment.

Listener : le moteur de post-traitement

Le Listener est le cerveau de PmaControl. Il surveille les tables de time series et déclenche des actions quand de nouvelles données arrivent. Son mécanisme repose sur une table pivot : `listener_main`.

La table `listener_main` contient :

Colonne	Rôle
`ts_file`	Le fichier source de données
`ts_max_date`	Le dernier timestamp traité
`ts_date_by_server`	Le dernier timestamp par serveur

Le Listener boucle en permanence. À chaque itération, il compare le `ts_max_date` enregistré avec le timestamp le plus récent dans les tables `ts_value_*`. Si une différence est détectée, cela signifie que l'Aspirateur a écrit de nouvelles données — le Listener déclenche alors la chaîne de post-traitement :

```
Listener loop:
  1. Check ts_max_date vs actual max(timestamp)
  2. If changed → trigger pipeline:
     a. updateDatabase()      — met à jour les métadonnées serveur
     b. afterUpdateVariable() — déclenche les règles conditionnelles
     c. Digest::integrate()   — agrège les métriques performance_schema
     d. Alias::updateAlias()  — rafraîchit les alias DNS
```

updateDatabase() synchronise les informations de base : version du serveur, état de la réplication, taille des bases, nombre de connexions actives.

afterUpdateVariable() est le moteur de règles. Il compare les nouvelles valeurs avec les seuils configurés et génère des alertes si nécessaire. Par exemple, si `Seconds_Behind_Master` passe au-dessus de 60, une alerte Warning est créée.

Digest::integrate() traite les données de `performance_schema`. Il agrège les statistiques de requêtes (temps d'exécution, lignes examinées, fréquence) et les stocke dans les tables de digest PmaControl. C'est ce qui alimente les dashboards de performance.

Alias::updateAlias() maintient la table `alias_dns`, qui mappe les noms conviviaux aux adresses IP réelles. Cette table est l'une des plus volumineuses : **1,1 million de lignes, 85 Mo de données**. Les alias sont utilisés partout dans l'interface pour afficher des noms lisibles au lieu d'adresses IP.

Dot3 : la topologie en temps réel

Dot3 est le composant de cartographie topologique. Il analyse les relations de réplication entre serveurs et génère un graphe dirigé au format DOT (Graphviz).

Le processus :

1. Dot3 lit les métadonnées de réplication de chaque serveur (master/slave, GTID, canal)
2. Il construit un graphe de dépendances : qui est master de qui, qui est slave de qui
3. Il génère une représentation visuelle avec des clusters (groupes de serveurs liés)

Les tables impliquées :

- **`dot3_graph`** : le graphe complet au format DOT, prêt à être rendu
- **`dot3_cluster`** : les clusters de serveurs (un cluster = un groupe de réplication)

Dot3 est particulièrement utile pour détecter les topologies cassées : un slave qui pointe vers un serveur inexistant, une boucle de réplication circulaire inattendue, ou un serveur isolé qui devrait être dans un cluster.

Schema : l'export de structure

Le composant Schema exporte la structure complète de chaque base de données supervisée. Pour chaque serveur, il crée une arborescence de fichiers :

```
data/model/<server_id>/databases/<db_name>/
├── schema/
│   └── tables/
│       ├── users.sql
│       ├── orders.sql
│       └── ...
├── routines/
│   ├── calculate_total.sql
│   └── ...
├── events/
│   ├── daily_cleanup.sql
│   └── ...
└── triggers/
    ├── before_insert_users.sql
    └── ...
```

Chaque fichier contient le `CREATE TABLE`, `CREATE PROCEDURE`, `CREATE EVENT` ou `CREATE TRIGGER` correspondant. Cela permet :

- De versionner la structure dans Git (diff entre deux exports)
- De comparer la structure entre production et staging
- De détecter les drifts de schéma (un index ajouté manuellement en production, une colonne modifiée sans migration)

Le CLI Glial

PmaControl est construit sur le framework Glial, qui fournit une interface en ligne de commande standardisée :

```
./glial <controller> <action> [params]
```

Exemples concrets :

```
# Vérifier l'état des démons
./glial agent check_daemon

# Forcer un cycle de collecte
./glial control service

# Exporter le schéma d'un serveur
./glial schema export 42

# Régénérer la topologie
./glial dot3 generate
```

Le CLI est utilisé à la fois manuellement (debugging, maintenance) et par les crons pour l'orchestration automatique.

Les crons : l'orchestration

Trois crons essentiels font tourner PmaControl :

1. `./glial agent check_daemon` — toutes les minutes

C'est le cron le plus fréquent. Il vérifie que tous les processus agents sont vivants et les relance si nécessaire. Un agent mort signifie un trou dans les données — ce cron garantit la continuité de la collecte.

```
* * * * * cd /srv/www/pmacontrol && ./glial agent check_daemon >> /tmp/pmacontrol_agent.log 2>&1
```

Si un agent ne répond pas après 3 tentatives, une alerte Telegram est envoyée.

2. `./glial control service` — toutes les 4 heures

Ce cron effectue les tâches de maintenance lourde :

- Recalcul des agrégations journalières
- Nettoyage des données expirées (rétention configurable)
- Régénération de la topologie Dot3
- Synchronisation des métadonnées serveur

- Vérification de cohérence entre les tables

Quatre heures est un bon compromis entre fraîcheur et charge : ces opérations sont coûteuses et n'ont pas besoin d'être temps réel.

```
0 */4 * * * cd /srv/www/pmacontrol && ./glial control service >> /tmp/pmacontrol_control.log 2>&1
```

3. `./monitor_mysql.sh` — toutes les minutes

Ce script est le point d'entrée de l'Aspirateur. Il déclenche un cycle de collecte complet :

```
* * * * * cd /srv/www/pmacontrol && ./monitor_mysql.sh >> /tmp/pmacontrol_monitor.log 2>&1
```

Le script gère la parallélisation : si vous supervisez 200 serveurs, il ne les contacte pas séquentiellement. Il répartit le travail en lots parallèles, avec un nombre de workers configurable.

Les tables clés

Voici les tables les plus importantes à connaître pour comprendre ou déboguer PmaControl :

`mysql_server`

La table centrale. Chaque ligne représente **une instance supervisée** — pas uniquement des serveurs MariaDB / MySQL, mais aussi :

- **Serveurs MySQL / MariaDB** (le cas principal)
- **Proxies** : MaxScale, ProxySQL, HAProxy
- **VIP** (Virtual IP)

Les colonnes `is_proxy` et `is_vip` distinguent les types :

is_proxy	is_vip	Type
0	0	Serveur MySQL / MariaDB classique
1	0	Proxy (MaxScale, ProxySQL, HAProxy)
0	1	VIP (Virtual IP)

```
-- Serveurs MySQL/MariaDB uniquement
SELECT id, ip, port, name, display_name, id_environment
FROM mysql_server
WHERE is_deleted = 0 AND is_proxy = 0 AND is_vip = 0;

-- Proxies (MaxScale, ProxySQL, HAProxy)
SELECT id, ip, port, name, display_name
FROM mysql_server
WHERE is_deleted = 0 AND is_proxy = 1;

-- VIPs
SELECT id, ip, port, name, display_name
FROM mysql_server
WHERE is_deleted = 0 AND is_vip = 1;
```

Les proxies et VIPs sont stockés dans la même table que les serveurs MySQL pour simplifier les jointures et la topologie. Dot3 les utilise pour dessiner les connexions entre les couches réseau (VIP → Proxy → Master → Slave). La colonne `timeout` est calculée dynamiquement : 11 secondes pour les proxies (qui répondent plus lentement aux checks), 1 seconde pour les serveurs classiques.

Des tables dédiées complètent les détails spécifiques à chaque type de proxy :

- **`maxscale_server`** / **`maxscale_server__mysql_server`** — configuration MaxScale et ses backends
- **`proxysql_server`** — configuration ProxySQL
- **`haproxy_main`** / **`haproxy_main_input`** / **`haproxy_main_output`** / **`link__haproxy_main_output__mysql_server`** — configuration HAProxy (listeners, frontends, backends)
- **`vip_server`** — détails des VIPs

`ts_variable`

Le dictionnaire des métriques. Chaque variable collectée (par exemple `Threads_connected`, `Innodb_buffer_pool_pages_data`) a une entrée dans cette table avec son identifiant numérique.

```
SELECT id, name, source
FROM ts_variable
WHERE name LIKE 'Innodb%';
```

`ts_value_general_int`

Le stockage principal des métriques numériques. C'est la table la plus volumineuse — elle reçoit des milliers d'insertions par seconde, et peut atteindre plusieurs milliards de lignes par jour sur les plus grosses installations PmaControl.

```
SELECT server_id, variable_id, value, timestamp
FROM ts_value_general_int
WHERE server_id = 42
  AND variable_id = 107  -- Threads_connected
  AND timestamp > NOW() - INTERVAL 1 HOUR;
```

Cette table est partitionnée par jour pour permettre un nettoyage rapide des anciennes données (`ALTER TABLE ... DROP PARTITION`).

`ts_value_general_json`

Pour les métriques complexes qui ne tiennent pas dans un entier : résultats de `SHOW PROCESSLIST` , tables de `performance_schema` (digest de requêtes, locks, table I/O), `SHOW ENGINE INNODB STATUS` , etc. Le format JSON permet de stocker des structures arbitraires. Les métriques de réplication (`SHOW SLAVE STATUS`) ont leurs tables dédiées (`ts_value_slave_*`).

`alias_dns`

La table d'alias DNS — 1,1 million de lignes, 85 Mo. Elle mappe les adresses IP aux noms lisibles et est utilisée dans toute l'interface.

```
SELECT ip, alias, source, updated_at
FROM alias_dns
WHERE ip = '10.0.1.42';
```

`dot3_graph` et `dot3_cluster`

Les tables de topologie. `dot3_graph` contient le graphe DOT complet, `dot3_cluster` les groupes logiques de serveurs.

Le flux de données complet

Récapitulons le parcours d'une métrique, de la source à l'écran :

Étape	Composant	Action

1	**CRON** `monitor_mysql.sh` (toutes les minutes)	Lance l'Aspirateur
2	**ASPIRATEUR**	SSH tunnel → MySQL → `SHOW GLOBAL STATUS`
		Écrit dans `ts_value_general_int` / `ts_value_general_json`
3	**LISTENER** (détecte `ts_max_date` changé)	`updateDatabase()` — met à jour les métadonnées serveur
		`afterUpdateVariable()` — alertes si seuils dépassés
		`Digest::integrate()` — agrégation `performance_schema`
		`Alias::updateAlias()` — rafraîchit `alias_dns`
4	**DOT3** (boucle toutes les ~3s)	Régénère la topologie de réplication en temps réel
5	**CRON** `control service` (toutes les 4h)	Nettoyage et agrégation journalière
6	**INTERFACE WEB**	Lit les tables agrégées → dashboards, graphiques, topologie

Dimensionnement

Pour un déploiement typique de 100 serveurs MariaDB / MySQL :

- **Base PmaControl** : environ 15 Go de données (dominé par les tables `ts_value_*`)
- **CPU** : 2-4 cores suffisent (le Listener est le plus gourmand)
- **RAM** : 4 Go minimum, 8 Go recommandé (pour le buffer pool de la base PmaControl elle-même)
- **Disque** : SSD obligatoire — les tables time series font beaucoup d'I/O aléatoire

Le moteur de stockage recommandé pour les tables `ts_value_*` est **RocksDB** (via MyRocks) : meilleure compression, meilleures performances en écriture séquentielle, et partitionnement natif par jour.

Debugging

Quand quelque chose ne fonctionne pas, voici la checklist :

1. **Les agents tournent-ils ?** `./glial agent check_daemon` — si un agent est mort, les données ne sont plus collectées pour les serveurs qu'il gère
2. **Le Listener tourne-t-il ?** Vérifier `ts_max_date` dans `listener_main` — s'il ne progresse plus, le Listener est bloqué
3. **Les crons s'exécutent-ils ?** Vérifier `/tmp/pmacontrol_*.log` pour les erreurs
4. **La connectivité SSH est-elle OK ?** Tester manuellement `ssh -p <port> <user>@<host>` avec la clé configurée
5. **La base PmaControl est-elle saine ?** Vérifier l'espace disque, les locks, les slow queries sur la base PmaControl elle-même

Conclusion

L'architecture de PmaControl suit un modèle classique ETL (Extract-Transform-Load) adapté au monitoring :

- **Extract** : l'Aspirateur collecte sans transformer
- **Transform** : le Listener applique les règles et agrège
- **Load** : les dashboards lisent les données transformées

Les 162 tables ne sont pas un accident de complexité — elles reflètent la richesse des données collectées sur chaque serveur MariaDB / MySQL. Comprendre cette architecture est essentiel pour quiconque veut déboguer un problème de collecte, étendre PmaControl avec un nouveau type de métrique, ou optimiser les performances du système de supervision lui-même.