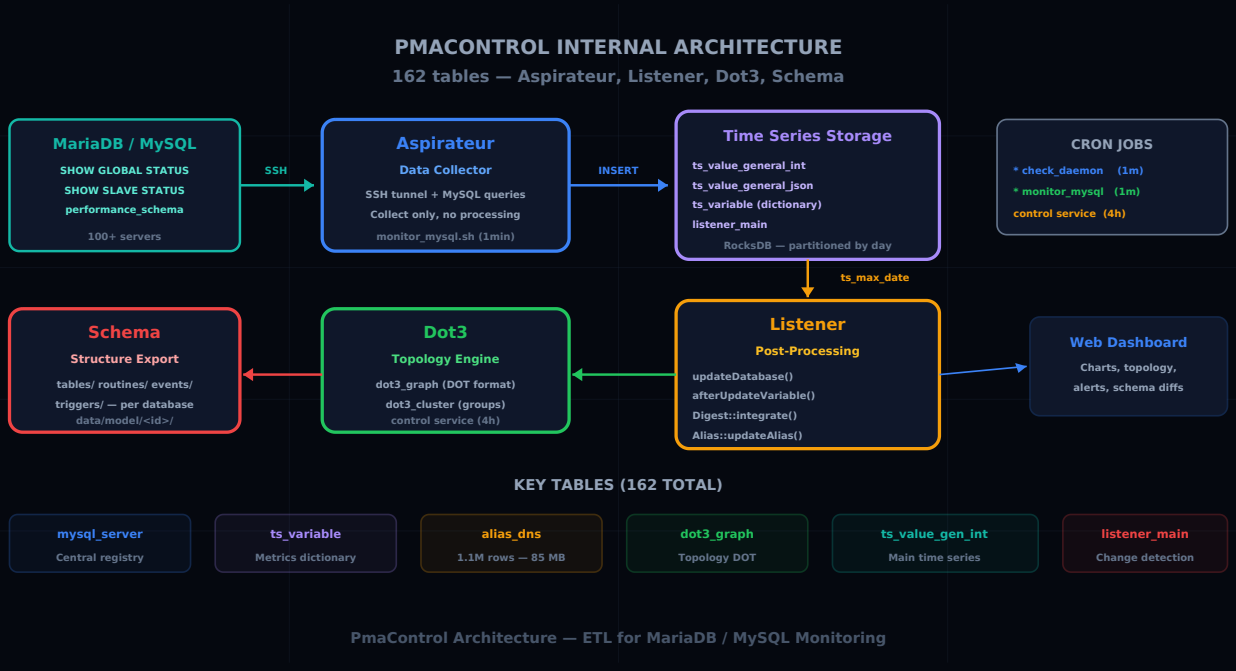
Внутренняя архитектура PmaControl: Aspirateur, Listener, Dot3 и 162 таблицы

Aurélien LEQUOY · April 13, 2026

PMACONTROL ARCHITECTURE AGENTS CRON MONITORING



162 таблицы, и ни одной лишней

PmaControl — это не просто панель мониторинга. Это распределённая система, которая собирает, хранит, преобразует и предоставляет метрики сотен серверов MariaDB / MySQL в реальном времени. Внутренняя база данных содержит **162 таблицы** — каждая с точной ролью в конвейере обработки данных.

В этой статье подробно разбирается внутренняя архитектура: четыре основных компонента (Aspirateur, Listener, Dot3, Schema), сквозной поток данных, cron-задачи, которые оркестрируют весь процесс, и ключевые таблицы, которые необходимо знать для отладки или расширения системы.

Четыре столпа

Aspirateur: сборщик данных

Aspirateur — это компонент, который собирает метрики с каждого наблюдаемого сервера. Его работа проста, но эффективна:

1. Он подключается к серверу через **SSH** (туннель), затем открывает локальное соединение **MySQL**
2. Выполняет серию запросов: `SHOW GLOBAL STATUS`, `SHOW GLOBAL VARIABLES`, `SHOW SLAVE STATUS`, `SHOW PROCESSLIST`, запросы к `performance_schema` и т.д.
3. Записывает результаты в таблицы `ts_value_*` (временные ряды) базы PmaControl

Префикс `ts_` встречается повсюду: он означает **time series** (временные ряды). Каждая метрика сохраняется с меткой времени и идентификатором исходного сервера.

```
Aspirateur → SSH tunnel → локальный MySQL
           → SHOW GLOBAL STATUS
           → SHOW SLAVE STATUS
           → запросы к performance_schema
           → INSERT INTO ts_value_general_int (...)
           → INSERT INTO ts_value_general_json (...)
```

Aspirateur **не выполняет никакой обработки**. Он собирает и записывает. Это фундаментальный принцип проектирования: разделение сбора и обработки для возможности их независимого масштабирования.

Listener: движок пост-обработки

Listener — это мозг PmaControl. Он наблюдает за таблицами временных рядов и запускает действия при поступлении новых данных. Его механизм основан на сводной таблице `listener_main`.

Таблица `listener_main` содержит:

Колонка	Назначение
`ts_file`	Файл-источник данных
`ts_max_date`	Последняя обработанная метка времени
`ts_date_by_server`	Последняя метка времени по каждому серверу

Listener работает в непрерывном цикле. На каждой итерации он сравнивает записанное значение `ts_max_date` с самой свежей меткой времени в таблицах `ts_value_*`. Если

обнаружено расхождение, значит Aspirateur записал новые данные — Listener запускает цепочку пост-обработки:

```
Цикл Listener:
  1. Проверка ts_max_date vs реальный max(timestamp)
  2. Если изменился → запуск пайплайна:
     a. updateDatabase()      — обновление метаданных сервера
     b. afterUpdateVariable() — условные правила
     c. Digest::integrate()   — агрегация метрик performance_schema
     d. Alias::updateAlias()  — обновление DNS-алиасов
```

updateDatabase() синхронизирует базовую информацию: версию сервера, состояние репликации, размеры баз данных, количество активных соединений.

afterUpdateVariable() — это движок правил. Он сравнивает новые значения с настроенными пороговыми значениями и генерирует оповещения при необходимости. Например, если `Seconds_Behind_Master` превышает 60, создаётся предупреждение уровня Warning.

Digest::integrate() обрабатывает данные `performance_schema`. Он агрегирует статистику запросов (время выполнения, количество просмотренных строк, частоту) и сохраняет их в таблицы дайджестов PmaControl. Именно эти данные питают дашборды производительности.

Alias::updateAlias() поддерживает таблицу `alias_dns`, которая сопоставляет удобочитаемые имена с реальными IP-адресами. Эта таблица — одна из самых объёмных: **1,1 миллиона строк, 85 МБ данных**. Алиасы используются повсюду в интерфейсе для отображения читаемых имён вместо IP-адресов.

Dot3: топология в реальном времени

Dot3 — это компонент картографии топологии. Он анализирует связи репликации между серверами и генерирует ориентированный граф в формате DOT (Graphviz).

Процесс:

1. Dot3 считывает метаданные репликации каждого сервера (master/slave, GTID, канал)
2. Строит граф зависимостей: кто является мастером кого, кто является слейвом кого
3. Генерирует визуальное представление с кластерами (группами связанных серверов)

Задействованные таблицы:

- `dot3_graph` : полный граф в формате DOT, готовый к рендерингу
- `dot3_cluster` : кластеры серверов (кластер = группа репликации)

Dot3 особенно полезен для обнаружения нарушенных топологий: слейв, указывающий на несуществующий сервер, неожиданная циклическая петля репликации или изолированный сервер, который должен входить в кластер.

Schema: экспорт структуры

Компонент Schema экспортирует полную структуру каждой наблюдаемой базы данных. Для каждого сервера он создаёт дерево файлов:

```
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
    └── ...
```

Каждый файл содержит соответствующий `CREATE TABLE` , `CREATE PROCEDURE` , `CREATE EVENT` или `CREATE TRIGGER` . Это позволяет:

- Версионировать структуру в Git (diff между двумя экспортами)
- Сравнивать структуру между production и staging
- Обнаруживать дрейф схемы (индекс, добавленный вручную в production, колонка, изменённая без миграции)

CLI Glial

PmaControl построен на фреймворке Glial, который предоставляет стандартизированный интерфейс командной строки:

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

Практические примеры:

```
# Проверить состояние демонов
./glial agent check_daemon

# Принудительно запустить цикл сбора
./glial control service

# Экспортировать схему сервера
./glial schema export 42

# Перегенерировать топологию
./glial dot3 generate
```

CLI используется как вручную (отладка, обслуживание), так и cron-задачами для автоматической оркестрации.

Cron-задачи: оркестрация

Три основные cron-задачи обеспечивают работу PmaControl:

1. `./glial agent check_daemon` — каждую минуту

Это самая частая cron-задача. Она проверяет, что все процессы агентов активны, и перезапускает их при необходимости. Остановившийся агент означает пробел в данных — эта задача гарантирует непрерывность сбора.

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

Если агент не отвечает после 3 попыток, отправляется оповещение в Telegram.

2. `./glial control service` — каждые 4 часа

Эта cron-задача выполняет тяжёлые задачи обслуживания:

- Пересчёт ежедневных агрегаций
- Очистка устаревших данных (настраиваемый период хранения)
- Перегенерация топологии Dot3
- Синхронизация метаданных серверов
- Проверка консистентности между таблицами

Четыре часа — хороший компромисс между актуальностью и нагрузкой: эти операции ресурсоёмкие и не нуждаются в выполнении в реальном времени.

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

3. `./monitor_mysql.sh` — каждую минуту

Этот скрипт является точкой входа для Aspirateur. Он запускает полный цикл сбора данных:

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

Скрипт управляет параллелизацией: если вы наблюдаете за 200 серверами, он не опрашивает их последовательно. Он распределяет работу на параллельные пакеты с настраиваемым числом воркеров.

Ключевые таблицы

Вот наиболее важные таблицы, которые нужно знать для понимания или отладки PmaControl:

`mysql_server`

Центральная таблица. Каждая строка представляет **наблюдаемый экземпляр** — не только серверы MariaDB / MySQL, но также:

- **Серверы MariaDB / MySQL** (основной случай)
- **Прокси**: MaxScale, ProxySQL, HAProxy
- **VIP** (Virtual IP)

Колонки `is_proxy` и `is_vip` различают типы:

is_proxy	is_vip	Тип
0	0	Обычный сервер MariaDB / MySQL
1	0	Прокси (MaxScale, ProxySQL, HAProxy)
0	1	VIP (Virtual IP)

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

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

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

Прокси и VIP хранятся в той же таблице, что и серверы MySQL, для упрощения JOIN-ов и построения топологии. Dot3 использует их для отрисовки связей между сетевыми уровнями (VIP → Proxy → Master → Slave). Колонка `timeout` вычисляется динамически: 11 секунд для прокси (которые медленнее отвечают на проверки), 1 секунда для обычных серверов.

Отдельные таблицы дополняют специфические детали каждого типа прокси:

- **`maxscale_server`** / **`maxscale_server__mysql_server`** — конфигурация MaxScale и его бэкенды
- **`proxysql_server`** — конфигурация ProxySQL
- **`haproxy_main`** / **`haproxy_main_input`** / **`haproxy_main_output`** / **`link__haproxy_main_output__mysql_server`** — конфигурация HAProxy (listeners, frontends, backends)
- **`vip_server`** — детали VIP

`ts_variable`

Словарь метрик. Каждая собираемая переменная (например, `Threads_connected`, `Innodb_buffer_pool_pages_data`) имеет запись в этой таблице с числовым идентификатором.

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

`ts_value_general_int`

Основное хранилище числовых метрик. Это самая объёмная таблица — она получает тысячи вставок в секунду и на крупнейших инсталляциях PmaControl может достигать нескольких миллиардов строк в день.

```
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;
```

Эта таблица партиционирована по дням для быстрой очистки устаревших данных (`ALTER TABLE ... DROP PARTITION`).

`ts_value_general_json`

Для сложных метрик, которые не укладываются в целое число: результаты `SHOW PROCESSLIST`, таблицы `performance_schema` (дайджесты запросов, блокировки, table I/O), `SHOW ENGINE INNODB STATUS` и т.д. Формат JSON позволяет хранить произвольные структуры. Метрики репликации (`SHOW SLAVE STATUS`) имеют собственные выделенные таблицы (`ts_value_slave_*`).

`alias_dns`

Таблица DNS-алиасов — 1,1 миллиона строк, 85 МБ. Она сопоставляет IP-адреса с читаемыми именами и используется по всему интерфейсу.

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

`dot3_graph` И `dot3_cluster`

Таблицы топологии. `dot3_graph` содержит полный граф в формате DOT, `dot3_cluster` — логические группы серверов.

Полный поток данных

Подведём итог пути метрики от источника до экрана:

Этап	Компонент	Действие
1	**CRON** `monitor_mysql.sh` (каждую минуту)	Запускает Aspirateur
2	**ASPIRATEUR**	SSH-туннель → MySQL → `SHOW GLOBAL STATUS`
		Записывает в `ts_value_general_int` / `ts_value_general_json`
3	**LISTENER** (обнаруживает изменение `ts_max_date`)	`updateDatabase()` — обновляет метаданные сервера
		`afterUpdateVariable()` — оповещения при превышении порогов
		`Digest::integrate()` — агрегация `performance_schema`
		`Alias::updateAlias()` — обновление `alias_dns`
4	**DOT3** (цикл каждые ~3 сек)	Перегенерация топологии репликации в реальном времени
5	**CRON** `control service` (каждые 4 ч)	Очистка и ежедневная агрегация
6	**ВЕБ-ИНТЕРФЕЙС**	Чтение агрегированных таблиц → дашборды, графики, топология

Размерность

Для типового развёртывания из 100 серверов MariaDB / MySQL:

- **База PmaControl**: около 15 ГБ данных (основная часть — таблицы `ts_value_*`)
- **CPU**: 2-4 ядра достаточно (Listener — самый требовательный)

- **RAM**: минимум 4 ГБ, рекомендуется 8 ГБ (для buffer pool самой базы PmaControl)
- **Диск**: SSD обязателен — таблицы временных рядов создают интенсивный случайный I/O

Рекомендуемый движок хранения для таблиц `ts_value_*` — **RocksDB** (через MyRocks): лучшее сжатие, лучшая производительность последовательной записи и нативное партиционирование по дням.

Отладка

Когда что-то не работает, вот чек-лист:

1. **Агенты запущены?** `./glial agent check_daemon` — если агент остановлен, данные больше не собираются для серверов, которыми он управляет
2. **Listener работает?** Проверьте `ts_max_date` в `listener_main` — если значение больше не увеличивается, Listener завис
3. **Cron-задачи выполняются?** Проверьте `/tmp/pmacontrol_*.log` на наличие ошибок
4. **Связь по SSH работает?** Протестируйте вручную `ssh -p <port> <user>@<host>` с настроенным ключом
5. **База PmaControl в порядке?** Проверьте свободное место на диске, блокировки, медленные запросы на самой базе PmaControl

Заключение

Архитектура PmaControl следует классической модели ETL (Extract-Transform-Load), адаптированной для мониторинга:

- **Extract**: Aspirateur собирает без преобразования
- **Transform**: Listener применяет правила и агрегирует
- **Load**: дашборды читают преобразованные данные

162 таблицы — не случайность избыточной сложности, а отражение богатства данных, собираемых с каждого сервера MariaDB / MySQL. Понимание этой архитектуры необходимо для любого, кто хочет отладить проблему сбора данных, расширить PmaControl новым типом метрик или оптимизировать производительность самой системы мониторинга.