

# MMQ: MariaDB как очередь сообщений

Sylvain ARBAUDIE · November 5, 2024

MARIADB

MESSAGE-QUEUE

BLACKHOLE

REPLICATION

## MMQ — MARIADB AS A MESSAGE QUEUE

Blackhole engine + binlog replication + InnoDB consumers



Sometimes the best architecture uses what you already have

## Безумная идея

А что если построить очередь сообщений (message queue), используя только MariaDB? Без Kafka, без RabbitMQ, без Redis Streams. Только MariaDB, её движки хранения и репликация binlog.

Это мысленный эксперимент, proof-of-concept. Цель — не заменить проверенные решения для обмена сообщениями, а продемонстрировать гибкость архитектуры MariaDB / MySQL и исследовать малоизвестные паттерны.

## Архитектура MMQ

MMQ (MariaDB Message Queue) основана на трёх нативных компонентах:

1. **Движок Blackhole**: движок хранения, который принимает INSERT, но ничего не сохраняет. Данные «исчезают» — за исключением того, что они записываются в binlog.
2. **Репликация binlog**: нативный механизм репликации MariaDB, который передаёт события с одного сервера на другой.
3. **Таблицы InnoDB + триггеры**: для потребления и отслеживания сообщений.

## Продюсер (Publisher)

Сервер-продюсер имеет таблицу Blackhole, служащую точкой входа:

```
CREATE TABLE message_queue (  
  msg_id BIGINT NOT NULL,  
  topic VARCHAR(255) NOT NULL,  
  payload JSON NOT NULL,  
  created_at DATETIME(6) DEFAULT NOW(6)  
) ENGINE=Blackhole;
```

Когда приложение публикует сообщение:

```
INSERT INTO message_queue (msg_id, topic, payload)  
VALUES (  
  NEXT VALUE FOR msg_sequence,  
  'order.created',  
  '{"order_id": 12345, "customer": "acme", "total": 99.99}'  
);
```

Движок Blackhole не записывает ничего на диск. Но INSERT записывается в binlog сервера. Вот в чём магия Blackhole: он участвует в binlog без потребления хранилища.

## Последовательности для идентификаторов

MariaDB поддерживает последовательности (с версии 10.3), которые предоставляют уникальные идентификаторы без стоимости AUTO\_INCREMENT с блокировкой:

```
CREATE SEQUENCE msg_sequence  
  START WITH 1  
  INCREMENT BY 1  
  CACHE 1000;
```

CACHE 1000 предварительно выделяет 1 000 значений в памяти, уменьшая обращения к диску и блокировки.

## Брокер (Relay)

Брокер — это сервер MariaDB, настроенный как слейв продюсера. Он получает события binlog и реплицирует их. Это механизм распределения.

Для fanout (одно сообщение нескольким потребителям) можно иметь несколько слейвов одного мастера — каждый слейв получает независимую копию всех сообщений.

```
Продюсер (Blackhole) → binlog → Брокер 1 (слейв)
                               → Брокер 2 (слейв)
                               → Брокер 3 (слейв)
```

## Потребитель (Consumer)

Каждый потребитель имеет таблицу InnoDB для хранения полученных сообщений и механизм отслеживания потребления:

```
CREATE TABLE consumed_messages (
  msg_id BIGINT PRIMARY KEY,
  topic VARCHAR(255),
  payload JSON,
  created_at DATETIME(6),
  consumed_at DATETIME(6) DEFAULT NULL,
  consumer_id VARCHAR(100) DEFAULT NULL
) ENGINE=InnoDB;
```

Триггер преобразует реплицированные INSERT в рабочие строки:

```
CREATE TRIGGER trg_message_arrived
BEFORE INSERT ON message_queue
FOR EACH ROW
BEGIN
  INSERT INTO consumed_messages (msg_id, topic, payload, created_at)
  VALUES (NEW.msg_id, NEW.topic, NEW.payload, NEW.created_at);
END;
```

Потребление осуществляется атомарным запросом:

```
UPDATE consumed_messages
SET consumed_at = NOW(6),
    consumer_id = 'worker-01'
WHERE consumed_at IS NULL
  AND topic = 'order.created'
ORDER BY msg_id ASC
LIMIT 1;
```

**LIMIT 1** в сочетании с атомарным **UPDATE** гарантирует, что только один потребитель обработает каждое сообщение (без двойного потребления).

## Сообщения в формате JSON

---

Нативный формат JSON в MariaDB (с версии 10.2) позволяет структурировать сообщения с богатыми payload:

```
-- Публикация сложного события
INSERT INTO message_queue (msg_id, topic, payload) VALUES (
  NEXT VALUE FOR msg_sequence,
  'user.profile.updated',
  JSON_OBJECT(
    'user_id', 42,
    'changes', JSON_ARRAY(
      JSON_OBJECT('field', 'email', 'old', 'old@mail.com', 'new', 'new@mail.com'),
      JSON_OBJECT('field', 'name', 'old', 'John', 'new', 'Jonathan')
    ),
    'timestamp', NOW(6)
  )
);
```

## Ограничения (и их много)

---

Будем ясны: MMQ — это концепция, а не production-ready решение.

**Нет гарантии надёжной доставки.** Если репликация упадёт, сообщения теряются (или задерживаются). Нет нативного механизма повторных попыток.

**Нет партиционирования.** Все сообщения проходят через один binlog. Нет распределения по topic, как в Kafka.

**Нет повторного воспроизведения.** После потребления сообщение не может быть легко воспроизведено (если только не сохранять binlog на продюсере).

**Задержка репликации.** Задержка репликации добавляет промежуток между публикацией и доступностью сообщения. Это допустимо для асинхронного режима, но не для реального времени.

**Нет распределённого подтверждения.** Продюсер не знает, обработал ли потребитель сообщение.

## Почему это всё же интересно

---

Несмотря на ограничения, этот паттерн демонстрирует важные концепции:

1. **Binlog как поток событий.** Binlog MariaDB / MySQL — это упорядоченный, долговечный и реплицируемый поток событий. Концептуально он близок к логу Kafka.
2. **Движок Blackhole как адаптер.** Blackhole позволяет «публиковать» без хранения, используя binlog как канал транспорта.
3. **Репликация как механизм распределения.** Multi-slave репликация предлагает нативный fanout без дополнительной конфигурации.
4. **База данных как универсальная инфраструктура.** Если у вас уже есть MariaDB в production, у вас уже есть инфраструктура для простого обмена сообщениями.

Для простых случаев использования — внутренние уведомления между сервисами, событийный аудит, репликация событий между площадками — MMQ может быть достаточной без добавления дополнительного компонента инфраструктуры.

## Заключение

---

MariaDB как очередь сообщений: безумная идея, забавный proof-of-concept и демонстрация гибкости связки движок Blackhole + репликация binlog. Не используйте это в production для критически важного обмена сообщениями. Но держите концепцию в голове — иногда лучшая архитектура использует то, что у вас уже есть.

---

Эта статья была первоначально опубликована на [Medium](#).