

Миграция очередей event_bus через RabbitMQ shovel

Runbook на случай, когда консьюмер микросервиса крашится с ошибкой вида:

```
PRECONDITION_FAILED - inequivalent arg 'x-max-priority' for queue '<name>'
in vhost '/': received the value '4' of type 'long' but current is none
```

Первый прецедент — 17.04.2026, очереди auth, location, otrs, social (562K сообщений суммарно, без консьюмеров в течение нескольких дней). [OLV-3520](#), [OLV-3521](#).

TL;DR

Если в код `AmqpService::createQueue()` добавили `$queue->setArgument('x-max-priority', N)`, а существующая очередь в RabbitMQ объявлена без этого аргумента — `declareQueue()` падает с `PRECONDITION_FAILED 406`. Consumer не стартует, supervisor рестартует его в цикле, сообщения копятсЯ.

Нельзя просто удалить очередь — все накопленные сообщения пропадут.

Правильный путь — shovel archive workflow (см. ниже). Минимизирует потери: сообщения идут в archive-очередь через binding, оттуда возвращаются в новую правильную очередь.

Workflow

На примере очереди foo с routing keys key1, key2 в exchange event_bus.

1. Создать архивную очередь с теми же bindings

Важно: bindings создаём до любых других действий. С этого момента любая новая публикация в event_bus по этим routing keys будет копироваться **и** в старую foo (без priority), **и** в новую foo_archive. Ни одно сообщение не пропадёт.

```
ssh root@srv-prod-postgre.olvr
rabbitmqadmin -u admin -p admin declare queue name=foo_archive durable=true
for rk in key1 key2; do
  rabbitmqadmin -u admin -p admin declare binding \
    source=event_bus destination=foo_archive routing_key=$rk
done
```

2. Shovel: слить старую очередь в archive

`src-delete-after=queue-length` фиксирует длину очереди на момент старта shovel и переносит ровно столько сообщений. Новые публикации уже идут параллельно в archive (через binding), поэтому «гонок» нет.

```
rabbitmqctl set_parameter shovel foo-drain '{
  "src-uri": "amqp://admin:admin@localhost:5672/%2F",
  "src-queue": "foo",
  "src-delete-after": "queue-length",
  "dest-uri": "amqp://admin:admin@localhost:5672/%2F",
  "dest-queue": "foo_archive"
}'
```

Отслеживаем прогресс через HTTP API (rabbitmqctl под нагрузкой таймаутится):

```
curl -s -u admin:admin \
  'http://localhost:15672/api/queues/%2F/foo?columns=messages'
curl -s -u admin:admin \
  'http://localhost:15672/api/queues/%2F/foo_archive?columns=messages'
```

Ждём пока foo дойдёт до 0.

3. Удалить старую очередь, создать новую

```
rabbitmqctl clear_parameter shovel foo-drain
rabbitmqadmin -u admin -p admin delete queue name=foo

# Создаём новую с нужными аргументами ДО рестарта консьюмера
# (чтобы не было гонки между declare консьюмера и нашим)
rabbitmqadmin -u admin -p admin declare queue name=foo durable=true \
  'arguments={"x-max-priority":4}'
for rk in key1 key2; do
  rabbitmqadmin -u admin -p admin declare binding \
    source=event_bus destination=foo routing_key=$rk
  # И сразу снимаем binding с архива, чтобы новые публикации шли только в foo
  rabbitmqadmin -u admin -p admin delete binding \
    source=event_bus destination_type=queue destination=foo_archive \
    properties_key=$rk
done
```

4. Рестарт консьюмера

```
ssh root@srv-prod-app.olvr \
  "docker exec tpark-it.back-foo.latest supervisorctl restart 'event-bus-
consume:*' "
```

Проверяем что консьюмер подключился:

```
curl -s -u admin:admin \
'http://localhost:15672/api/queues/%2F/foo?columns=name,consumers,arguments'
# Ожидаем: consumers=1, arguments={"x-max-priority":4}
```

5. Shovel archive → новая очередь

```
rabbitmqctl set_parameter shovel foo-restore '{
  "src-uri": "amqp://admin:admin@localhost:5672/%2F",
  "src-queue": "foo_archive",
  "src-delete-after": "queue-length",
  "dest-uri": "amqp://admin:admin@localhost:5672/%2F",
  "dest-queue": "foo"
}'
```

Консьюмер (если он шустрый) будет разгребать foo параллельно с заливкой из архива — foo может оставаться около нуля, а foo_archive уменьшаться. Это норма.

6. Чистка

```
rabbitmqctl clear_parameter shovel foo-restore
rabbitmqadmin -u admin -p admin delete queue name=foo_archive
```

Грабля 1: НЕ shovel через dest-exchange

Не использовать dest-exchange — он теряет routing_key исходного сообщения при republish. Сообщения уйдут в event_bus (direct) с пустым routing key и тихо отбросятся как unrouted, потому что:

- На event_bus не настроен alternate-exchange.
- DLX не ловит unrouted, только rejected/expired/queue-overflow.

17.04.2026 так потеряно 83 сообщения social.publication — безвозвратно.

Всегда использовать dest-queue (default exchange «» + routing_key = имя очереди).
Консьюмер при обработке не смотрит на routing_key из envelope, ему всё равно.

Грабля 2: hot-patch vendor пропадает при deploy

В инциденте 17.04 в контейнерах back-auth и back-otrs кто-то вручную закоментировал `$queue->setArgument(x-max-priority, ...)` в `/app/vendor/t-park/api-bundle/src/EventBus/AmqpService.php` — workaround без пересоздания очереди. Патч держится до первой пересборки образа и уйдёт с `composer install`.

Если вынуждены делать hot-patch:

- Обязательно заводить Jira-задачу на нормальный фикс (пересоздание очереди по этому runbook).
- Пометить в ~/work/olvery/olvery.ru/CLAUDE.md в разделе живого состояния, чтобы не потерять.

Грабля 3: rabbitmqctl может таймаутиться

Под нагрузкой shovel (особенно для больших очередей — 500K+ сообщений) rabbitmqctl list_queues уходит в 60-секундный таймаут. Для мониторинга использовать HTTP API ([http://localhost:15672/api/queues/...](http://localhost:15672/api/queues/)) — отвечает мгновенно.

Верификация после починки

- `docker exec tpark-it.back-foo.latest supervisorctl status` — процесс event-bus-consume должен быть RUNNING с uptime > пары минут (если uptime всё время 0:00:00 — consumer крашится, смотреть логи).
- `curl -s -u admin:admin http://localhost:15672/api/queues/%2F/foo` — consumers >= 1, arguments содержат ожидаемые.
- `docker exec tpark-it.back-foo.latest php /app/bin/console event-bus:consume --time-limit=3 -v` — запустить руками в foreground, увидеть Event Bus Message Consumed и отсутствие exceptions.

Связанные ссылки

- OLV-3520 — ИОЛЛА: не приходят уведомления о заказах (корневая причина — очередь auth)
- OLV-3521 — ИОЛЛА: двойная оплата переговорных (корневая причина — очередь otrs)
- [RabbitMQ Dynamic Shovel docs](#)
- AmqpService.php: olvery/back-api-bundle:src/EventBus/AmqpService.php — где живёт `setArgument('x-max-priority', ...)`

DISCUSSION

From:

<https://wiki.olvery.ru/> - Olvery

Permanent link:

<https://wiki.olvery.ru/doku.php?id=dev:ops:rabbitmq:queue-migration>

Last update: **2026/04/23 11:48**

