Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

Apache Kafka: потоковая обработка событий под ключ

Наша методология внедрения Kafka 4.x на KRaft: sizing, топики, exactly-once, мониторинг lag

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

Заказчику нужен единый поток событий (веб-аналитика, кассы, 1С, интеграции) без потерь и с пересчётом задним числом. Классические очереди теряют данные после ACK, не держат пики и не умеют replay. Мы строим платформу на Apache Kafka 4.x (KRaft): реплицируемый лог, exactly-once, масштабирование без остановки. Бездействие грозит потерей событий и расхождениями в отчётах.

Почему это важно бизнесу

- Потеря событий в очереди = расхождения в отчётах и биллинге; после ACK в RabbitMQ данные не восстановить, в Kafka есть replay за 30 дней
- Пики нагрузки x3–x4 к среднему кладут классические очереди — простой интеграций и деградация клиентских сервисов
- Без replay ошибку в логике обработки нельзя исправить задним числом: пересчёт агрегатов и метрик невозможен
- Kafka закрывает рост на порядок без переписывания архитектуры: брокеры и партиции добавляются без остановки потока

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

4.3.1

Актуальный релиз Kafka (25.06.2026); с ветки 4.0 — только KRaft, ZooKeeper удалён

kafka.apache.org

RF 3 / ISR 2

default.replication.factor=3, min.insync.replicas=2, acks=all: отказ брокера без потерь

наш стандарт

720 ч

log.retention.hours=720 (30 дней) на сыром топике — окно replay и пересчёта агрегатов

наш стандарт

5 мс

linger.ms: в Kafka 4.0 дефолт поднят с 0 до 5 мс (KIP-1030); мы ставим 50 мс под батчинг

по докам Kafka 4.0

6–8 ГБ

Heap JVM брокера; остальную RAM (из 32 ГБ) отдаём OS page cache — чтение сегментов идёт из него через sendfile (zero-copy)

Kafka ops docs

100K

Порог warning по kafka_consumergroup_lag_sum (5 мин); critical — 1M в течение 2 мин

наш стандарт

РЕАЛИЗАЦИЯ 1 ИЗ 3

Sizing и развёртывание кластера на KRaft

Что настраиваем

3–5 брокеров Kafka 4.3.1 + 3 выделенных KRaft-контроллера; NVMe JBOD, сеть 10 Gbps

Как мы это делаем

1. Считаем поток: 1 млрд соб./день ≈ 12K msg/s в среднем, 40K в пике; 320 Б на сжатое сообщение → ~29 ТБ хранилища на 30 дней при RF=3
2. Брокер: 8 vCPU, 32 ГБ RAM, 4×2 ТБ NVMe в JBOD (несколько путей в log.dirs, без RAID — реплицирует сама Kafka), сеть 10 Gbps
3. KRaft вместо ZooKeeper: 3 узла process.roles=controller, кворум через controller.quorum.bootstrap.servers (динамические voters, KIP-853)
4. Надёжность: default.replication.factor=3, min.insync.replicas=2, unclean.leader.election.enable=false
5. Хранение: log.retention.hours=720, log.segment.bytes=1073741824, compression.type=producer (lz4 задаёт продюсер)

РЕЗУЛЬТАТ

Кластер держит пик 40K msg/s на трети мощности; отказ любого брокера проходит без потери записи — ISR=2 сохраняет кворум, а page cache даёт p99 produce latency в единицы миллисекунд.

КЛЮЧЕВОЙ НЮАНС

RAM брокера — не для JVM: heap 6–8 ГБ, остальное ядру под page cache. Только NVMe/SSD — на HDD ловили latency spikes при compaction. С 4.0 брокеру нужна Java 17.

РЕАЛИЗАЦИЯ 2 ИЗ 3

Топики, продюсеры/консюмеры и exactly-once

Что настраиваем

Топики events.raw → events.validated → metrics.realtime + DLQ; продюсеры Go, консюмеры Python

Как мы это делаем

1. Партиционируем по бизнес-ключу (site_id/договор): 48 партиций на raw = потолок параллелизма группы; DLQ отдельным топиком с retention 90 дней
2. Producer: acks=all и enable.idempotence=true (дефолты с Kafka 3.0), batch.size=1048576, linger.ms=50, compression.type=lz4
3. Consumer: enable.auto.commit=false, ручной commit только после обработки батча; fetch.min.bytes=50000, max.poll.interval.ms=300000
4. Exactly-once внутри Kafka: transactional.id на инстанс + send_offsets_to_transaction(); во внешние БД — идемпотентные upsert по event_id
5. Схемы: Avro + Schema Registry в режиме BACKWARD (дефолт) — новые поля только с default, продюсеры обновляются независимо от консюмеров

РЕЗУЛЬТАТ

Пайплайн переживает ретраи и рестарты без дублей и потерь: идемпотентный продюсер гасит дубли при retry, транзакции дают exactly-once между топиками, DLQ изолирует невалидные события для разбора.

КЛЮЧЕВОЙ НЮАНС

«Exactly-once» действует только в паттерне read-process-write внутри Kafka; при записи во внешние системы гарантию обеспечивает идемпотентность на стороне потребителя, а не брокер.

РЕАЛИЗАЦИЯ 3 ИЗ 3

Мониторинг lag и эксплуатация без простоев

Что настраиваем

Burrow + kafka-exporter → Prometheus/Grafana; алерты по lag, ISR и offline partitions

Как мы это делаем

1. Burrow оценивает динамику lag по окну из 10 коммитов оффсетов: статусы OK/WARNING/ERROR и пометка STALLED-партиций — без ручных порогов
2. kafka-exporter → Prometheus: kafka_consumergroup_lag_sum>100000 (warning, for 5m) и >1000000 (critical, for 2m)
3. UnderReplicatedPartitions>0 (1m) и OfflinePartitions>0 (30s) — critical; shrink ISR ловим по replica.lag.time.max.ms=30000 до потери партиций
4. Ребалансы: group.protocol=consumer (KIP-848, GA в Kafka 4.0) — инкрементальный протокол на стороне брокера вместо full-stop eager
5. Запас по retention без дисков: tiered storage (GA с Kafka 3.9) — remote.log.storage.system.enable на брокере + remote.storage.enable на топике

РЕЗУЛЬТАТ

Отставание потребителей видим за минуты до влияния на бизнес: WARNING по динамике lag приходит раньше жалоб, а ребаланс по KIP-848 сокращает простой группы с десятков секунд до единиц.

КЛЮЧЕВОЙ НЮАНС

Смотрим не абсолютный lag, а динамику: Burrow помечает STALLED, когда оффсеты не двигаются. Сжатие ISR — ранний сигнал деградации диска или сети, не ждём offline partitions.

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

- ✕ **Партиции «на вырост»**
  96 партиций вместо 48 дали медленный rebalance и нагрузку на контроллер. Считаем от параллелизма консюмеров ×2 — добавить партиции можно позже, убрать нельзя.

- ✕ **RAID вместо JBOD**
  Kafka реплицирует сама: RAID дублирует работу и режет I/O. Диски отдаём отдельными путями в log.dirs как JBOD (в KRaft поддержан, KIP-858).

- ✕ **Весь RAM — в JVM heap**
  Heap больше 8 ГБ отнимает page cache и удлиняет GC-паузы. Ставим 6–8 ГБ, остальную память оставляем ядру под кэш чтения сегментов.

- ✕ **auto.commit до обработки**
  enable.auto.commit=true коммитит оффсеты по таймеру до фактической обработки — потеря сообщений при падении. Только ручной commit после flush продюсера.

- ✕ **Eager-ребаланс всей группы**
  Классический eager-протокол останавливает всю группу на ребаланс. CooperativeStickyAssignor или group.protocol=consumer (Kafka 4.0) делают его инкрементальным.

- ✕ **Compaction без lag-порогов**
  cleanup.policy=compact без min.compaction.lag.ms уплотняет свежие записи. Ставим min.compaction.lag.ms=3600000 и delete.retention.ms=86400000 для tombstone.

- ✕ **Неуникальный transactional.id**
  Одинаковый transactional.id у двух инстансов вызывает ProducerFencedException и остановку пайплайна — кодируем в ID номер инстанса или партиции.

- ✕ **Апгрейд на 4.x вслепую**
  4.0 не стартует с невалидными legacy-конфигами (KIP-1030) и требует Java 17 на брокере; ZooKeeper-режим удалён — сначала миграция метаданных на KRaft.

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- 3 брокера KRaft, RF=3, min.insync.replicas=2, acks=all на продюсерах
- SSD/NVMe под log.dirs, heap JVM 6 ГБ, остальная RAM — под page cache
- kafka-exporter + Prometheus: алерты на lag и under-replicated partitions

НОРМАЛЬНО

- Schema Registry (BACKWARD), DLQ-топик и ручной commit оффсетов после обработки
- Идемпотентный продюсер + транзакции для read-process-write пайплайнов
- Burrow для оценки динамики lag, дашборды Grafana по consumer-группам
- Партиции = целевой параллелизм консюмеров ×2, ключ — по бизнес-сущности

ХОРОШО

- Kafka 4.3 (KRaft), group.protocol=consumer — инкрементальные ребалансы KIP-848
- Tiered storage под длинный retention без расширения локальных дисков
- Kafka Streams/ksqlDB для real-time агрегатов, compacted-топики под срезы
- DR: второй кластер через MirrorMaker 2 и регулярные учения replay/restore

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] acks=all и enable.idempotence=true заданы на всех продюсерах?
- [ ] min.insync.replicas=2 при RF=3 и unclean.leader.election.enable=false?
- [ ] Оффсеты коммитятся только после обработки и flush продюсера?
- [ ] Настроены алерты на consumer lag, under-replicated и offline partitions?
- [ ] Retention сырых топиков покрывает окно возможного пересчёта (replay)?
- [ ] Схемы событий под контролем Schema Registry с проверкой совместимости?
- [ ] DLQ настроен, невалидные события расследуются, а не теряются молча?
- [ ] Heap брокера 6–8 ГБ, а не половина RAM, page cache не задушен?
- [ ] План апгрейда на 4.x проверен: Java 17, KRaft, конфиги по KIP-1030?

Если хотя бы на два вопроса ответ «нет» или «не знаю» — тема требует внимания.

ЧТО МЫ ДЕЛАЕМ

Как поможет ITFresh

ITFresh — ИТ-аутсорсинг для юридических лиц до 50 рабочих мест в Москве и области. 15+ лет практики, собственная инфраструктура в дата-центре МТС (8 серверов Dell Xeon Platinum).

- Спроектируем и развернём кластер Kafka 4.x на KRaft: sizing под ваш поток, топология, безопасность
- Мигрируем очереди RabbitMQ и самописные обмены на Kafka без остановки интеграций
- Настроим мониторинг Burrow + Prometheus/Grafana с порогами lag и алертами в Telegram
- Возьмём эксплуатацию на сопровождение: апгрейды, ребалансировка, replay-пересчёты, DR

15+
лет в ИТ-поддержке

50
рабочих мест — наш профиль

МТС
дата-центр, Москва

КОНТАКТЫ

Обсудить вашу задачу

Сайт	**itfresh.ru**
Телефон	**+7 903 729-62-41**
Telegram	**@ITfresh_Boss**

Бесплатно посмотрим вашу инфраструктуру по этому чек-листу и скажем, где тонко — без обязательств.

itfresh.ru

ДОКУМЕНТАЦИЯ И НАШ СТЕК

Техническая база

01 Kafka Documentation — Broker & Topic Configs (kafka.apache.org — 4.3)

02 Consumer Rebalance Protocol (KIP-848) (kafka.apache.org — 4.1)

03 Operations — Tiered Storage (kafka.apache.org — 3.9+)

04 KIP-1030: Change constraints and default values (cwiki.apache.org — 4.0)

05 Schema Evolution and Compatibility (docs.confluent.io — 2026)

06 Burrow — Consumer Lag Evaluation Rules (github.com/linkedin — 2026)

07 librdkafka Configuration (confluent-kafka) (github.com/confluentinc — 2.x)

08 Наш шаблон server.properties и правил Prometheus (itfresh.ru — 2026)

Основано на официальной документации продуктов и нашей практике внедрения.