Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

OpenTelemetry: как мы строим единую наблюдаемость

OTel Collector, zero-code инструментация и корреляция трейсов, логов и метрик под ключ

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

У заказчика метрики в Prometheus, логи в ELK, трейсы — в лучшем случае у части сервисов. При инциденте инженеры вручную сопоставляют три системы, MTTR растягивается на часы, а смена бэкенда мониторинга требует переписывать инструментацию во всех сервисах. Мы внедряем OpenTelemetry: один SDK, один протокол OTLP, один Collector — и корреляция запроса по trace_id в один клик.

Почему это важно бизнесу

- Без корреляции лог→трейс диагностика по десяткам сервисов — ручной перебор: MTTR часы вместо минут, простой оплачивает бизнес
- Vendor lock-in: инструментация под конкретный APM зашита в код, смена бэкенда — перепись сервисов и недели работы разработчиков
- Хранение 100% трейсов раздувает бюджет хранилища; tail-sampling режет объём ~в 10 раз без потери ошибок
- Слепые зоны: неинструментированные сервисы всплывают в жалобах клиентов, а не в алертах

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

0.156.0

Версия otel/opentelemetry-collector-contrib: пиним релиз, обновляем по changelog, не latest

collector-contrib, июль 2026

4317/4318

Порты OTLP: gRPC 4317 внутри периметра, HTTP/protobuf 4318 для браузеров и legacy-клиентов

по докам opentelemetry.io

80% / 15%

memory_limiter: limit_percentage 80, spike_limit_percentage 15, check_interval 1s — защита Col...

по докам memory_limiter

30 с

decision_wait в tail_sampling (дефолт): окно, за которое собираем все спаны трейса перед решен...

tail_sampling README

10%

probabilistic для здоровых трейсов; ошибки и запросы дольше 1000 мс храним полностью

наш стандарт

2.29.0

Java-агент zero-code: -javaagent без правки кода, overhead единицы процентов CPU

opentelemetry-java-instrumentation, июнь 2026

РЕАЛИЗАЦИЯ 1 ИЗ 3

Разворачиваем Collector: agent + gateway

Что настраиваем

Все Docker/K8s-узлы заказчика: otelcol-contrib агентом на каждом хосте + центральный gateway

Как мы это делаем

1. Ставим otel/opentelemetry-collector-contrib:0.156.0 (systemd или DaemonSet), конфиг в /etc/otelcol-contrib/config.yaml
2. Пайплайны traces/metrics/logs: receiver otlp (4317/4318), процессоры строго memory_limiter → resource → batch
3. batch: send_batch_size 8192, timeout 5s; memory_limiter: check_interval 1s, limit_percentage 80, spike_limit_percentage 15
4. Экспортёры: prometheusremotewrite в VictoriaMetrics, otlp в Tempo, otlphttp в Loki 3.x (нативный /otlp)
5. Включаем extension health_check на :13133 — на него вешаем проверку живости в Zabbix

РЕЗУЛЬТАТ

Смена бэкенда — правка одного YAML на gateway, а не кода сервисов. Агент буферизирует телеметрию при недоступности gateway, gateway централизует семплирование и обогащение. Здоровье самого Collector видно через health_check и его метрики otelcol_*.

КЛЮЧЕВОЙ НЮАНС

memory_limiter всегда первый в цепочке процессоров — он должен видеть весь входящий поток до batch; иначе при всплеске нагрузки Collector уходит в OOM раньше, чем сработает ограничение.

РЕАЛИЗАЦИЯ 2 ИЗ 3

Инструментация без правки кода: Python, Java + обвязка Go

Что настраиваем

Прикладные сервисы заказчика: веб-приложения, API, фоновые воркеры — без правки исходников

Как мы это делаем

1. Python: pip install opentelemetry-distro, затем opentelemetry-bootstrap -a install, запуск через opentelemetry-instrument
2. Java: -javaagent:opentelemetry-javaagent.jar (2.29.0) в строку запуска — HTTP, JDBC, Kafka видны сразу
3. Go: обвязки otelhttp/otelgrpc из contrib + ручная инициализация TracerProvider с WithBatcher — zero-code агента у Go нет
4. Единые env при деплое: OTEL_SERVICE_NAME, OTEL_EXPORTER_OTLP_ENDPOINT (локальный агент :4317), OTEL_RESOURCE_ATTRIBUTES
5. Кастомные спаны добавляем только в бизнес-логику, которую автоинструментация не видит (расчёты, интеграции с 1С)

РЕЗУЛЬТАТ

Типовой сервис подключается к трассировке за 15–30 минут без релиза кода. HTTP, gRPC, SQL, Redis, Kafka инструментируются автоматически, у каждого сервиса корректный service.name вместо unknown_service.

КЛЮЧЕВОЙ НЮАНС

Перед стартом сверяем матрицу поддержки в доке javaagent и python-distro: EOL-версии фреймворков выпадают из автоинструментации — для них закладываем ручные спаны заранее.

РЕАЛИЗАЦИЯ 3 ИЗ 3

Корреляция и tail-sampling в production

Что настраиваем

Gateway-Collector + Grafana: связка Tempo (трейсы), Loki (логи), VictoriaMetrics (метрики)

Как мы это делаем

1. Прокидываем trace_id/span_id в формат логов (logging.Filter в Python, MDC в Java) — лог ищется по трейсу и обратно
2. tail_sampling на gateway: policies status_code=ERROR (всё), latency threshold_ms=1000 (всё), probabilistic 10%
3. В Grafana настраиваем derived field в Loki: клик по trace_id в строке лога открывает трейс в Tempo
4. Контекст между сервисами — W3C Trace Context (traceparent), пропагатор по умолчанию во всех OTel SDK

РЕЗУЛЬТАТ

Инцидент разбирается от алерта до конкретного вызова за минуты: метрика → exemplar → трейс → логи одного запроса. Объём хранимых трейсов падает ~в 10 раз, при этом 100% ошибок и медленных запросов сохранены.

КЛЮЧЕВОЙ НЮАНС

tail_sampling работает, только если все спаны трейса приходят на один экземпляр Collector — при нескольких gateway ставим перед ними loadbalancing-экспортёр с routing_key: traceID.

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

- ✕ **batch раньше memory_limiter**
  Неверный порядок процессоров ведёт к OOM: memory_limiter обязан стоять первым в пайплайне, batch — последним перед экспортом

- ✕ **SDK напрямую в бэкенд**
  Без Collector нет буферизации и смены бэкенда без релиза; ставим локальный агент, SDK шлёт только на localhost:4317

- ✕ **Устаревший loki-экспортёр**
  Старый loki exporter удалён из contrib; логи в Loki 3.x отправляем через otlphttp на нативный OTLP-эндпоинт /otlp

- ✕ **Тег latest у Collector**
  Между релизами contrib меняются поля конфигов и удаляются компоненты; пиним версию образа и читаем changelog перед обновлением

- ✕ **unknown_service в трейсах**
  Забытый OTEL_SERVICE_NAME превращает карту сервисов в кашу; имя и версию сервиса задаём через env на этапе деплоя

- ✕ **Head-sampling теряет ошибки**
  Вероятностное семплирование на SDK отбрасывает трейсы до того, как известен их статус; решение переносим на Collector (tail_sampling)

- ✕ **Взрыв кардинальности метрик**
  user_id и URL с параметрами в атрибутах метрик кладут хранилище; чистим атрибуты процессорами attributes/transform на gateway

- ✕ **OTLP наружу без TLS**
  tls.insecure допустим только внутри периметра; между площадками — сертификаты или туннель, иначе телеметрия идёт открытым текстом

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- Один Collector (contrib, пиновая версия), пайплайны traces+logs по OTLP
- Zero-code автоинструментация ключевых сервисов, service.name у каждого
- memory_limiter + batch в каждом пайплайне, health_check под мониторинг

НОРМАЛЬНО

- Схема agent+gateway: агент на каждом узле, gateway для обработки и экспорта
- trace_id в логах всех сервисов, derived fields Loki→Tempo в Grafana
- tail_sampling: 100% ошибок и медленных запросов, 10% остального
- TLS на OTLP между площадками, пиновые версии SDK и агентов

ХОРОШО

- Несколько gateway за loadbalancing-экспортёром (routing_key: traceID)
- Exemplars: переход метрика→трейс прямо из графика Grafana
- Контроль кардинальности: attributes/transform-процессоры, лимит атрибутов
- Алерты на метрики самого Collector: otelcol_exporter_send_failed_*, очереди

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] Есть ли у каждого сервиса корректный service.name и версия в ресурсных атрибутах?
- [ ] Стоит ли memory_limiter первым процессором во всех пайплайнах Collector?
- [ ] Можно ли из ошибки в логах за один клик открыть трейс конкретного запроса?
- [ ] Сохраняются ли 100% трейсов с ошибками при включённом семплировании?
- [ ] Зафиксированы ли версии Collector и SDK (не latest), читается ли changelog перед апдейтом?
- [ ] Мониторится ли сам Collector: health_check :13133, метрики otelcol_*, очереди экспортёров?
- [ ] Закрыт ли OTLP-трафик TLS и фаерволом на портах 4317/4318 между площадками?
- [ ] Переживёт ли телеметрия 10-минутную недоступность бэкенда (буферы агента)?
- [ ] Вычищены ли высококардинальные атрибуты из метрик до записи в хранилище?

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).

- Разворачиваем стек наблюдаемости под ключ: Collector agent+gateway, Tempo, Loki, VictoriaMetrics, Grafana
- Инструментируем сервисы без правки кода (Python/Java) и с минимальной обвязкой (Go)
- Настраиваем tail-sampling, корреляцию лог↔трейс↔метрика и алерты по SLO
- Берём стек на поддержку: обновления Collector/SDK по changelog, контроль объёмов и кардинальности

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 Collector: configuration, memory_limiter и batch processors (opentelemetry.io — 2026)

02 tail_sampling processor README (collector-contrib) (github.com — v0.156.0)

03 Zero-code instrumentation: Java agent, Python distro (opentelemetry.io — 2.29.0)

04 OTLP specification и W3C Trace Context (opentelemetry.io — 1.x)

05 Tempo и Loki: нативный приём OTLP (grafana.com — 2026)

06 Наш шаблон otelcol-config.yaml (agent+gateway) (itfresh.ru — 2026)

Основано на официальной документации продуктов и нашей практике внедрения.