Observability и SRE

В распределенных системах понятие «работоспособности» перестало быть бинарным (работает/не работает). Сервис может отвечать на health-check и иметь низкую нагрузку на CPU, но при этом отдавать ошибки 500 конкретной группе пользователей или медленно деградировать из-за утечки памяти. Именно здесь заканчивается зона ответственности классического мониторинга и начинается **Observability (Наблюдаемость)**.

Это архитектурное свойство системы, которое определяет, насколько точно можно понять её **внутреннее состояние**, анализируя только её **внешние выходные данные** (телеметрию). Если мониторинг говорит вам, *что* система сломалась (фиксирует известные симптомы), то Observability дает контекст, чтобы понять *почему* (позволяет расследовать первопричины). Для SRE это разница между пассивным наблюдением за красным дашбордом и активным расследованием «неизвестных неизвестных» (unknown unknowns) — сбоев, сценарии которых невозможно было предсказать заранее.

Четыре столпа observability

Полноценная наблюдаемость (Observability) строится не на одном инструменте, а на комбинации четырех типов данных. Каждый из них отвечает на свой вопрос о системе.

1. Метрики (Metrics): «Что происходит?»

Метрики — это числовые данные, агрегированные во времени. Они дешевы в хранении и идеально подходят для выявления трендов.

- **Плюс:** Экономичность. Хранить число `cpu_usage=80` за каждую секунду года намного дешевле, чем хранить тексты логов.
- **Минус:** Нет контекста. Метрика скажет, что CPU загружен, но не скажет, какой именно запрос это вызвал.

Золотые сигналы Google SRE (The Four Golden Signals) Это стандарт де-факто для мониторинга любого сервиса:

1. **Latency (Задержка):** Время обслуживания запроса.
   - *Важно:* Не смотрите на среднее значение (Average). Оно врет. Смотрите на **перцентили** (p95, p99). Если p99 = 2 сек, значит, 1% ваших пользователей (часто самых "жирных" клиентов с большими данными) ждут 2 секунды.
2. **Traffic (Трафик):** Нагрузка на систему.
   - Для веб-серверов: HTTP-запросы в секунду (RPS).
   - Для баз данных: Транзакции в секунду (TPS).
   - Для видео: Пропускная способность (Bandwidth).
3. **Errors (Ошибки):** Процент сбойных запросов.
   - Разделяйте явные ошибки (HTTP 500), логические ошибки (HTTP 200, но в теле JSON `error: true`) и мягкие сбои (HTTP 429 Rate Limited).
4. **Saturation (Насыщение):** Насколько система "полна".
   - Это метрика емкости: очередь задач, пул соединений к БД, утилизация диска. Насыщение предсказывает сбой *до* того, как он случится.

2. Логи (Logs): «Почему это происходит?»

Логи — это дискретные события. Если метрики — это "пульс" пациента, то логи — это его "история болезни". Они дают контекст.

Structured Logging (Структурированные логи) В 2024 году писать логи простым текстом (`Error: user failed`) — моветон. Логи должны быть машиночитаемыми (обычно JSON).

- **Плохо:** `2023-10-01 Payment failed for user 123`
- **Хорошо:** `{"level": "error", "event": "payment_failed", "user_id": 123, "amount": 500, "trace_id": "abc-123"}`

Уровни логирования:

- **ERROR:** Система сломалась, нужен человек (база упала).
- **WARN:** Ошибочное поведение, но система работает (404 ошибка, повторная попытка соединения).
- **INFO:** Штатная работа (пользователь вошел, заказ создан).
- **DEBUG:** Техническая "простыня" для разработчика (дамп переменных). *В продакшене обычно отключен.*

3. Распределенная трассировка (Distributed Tracing): «Где это происходит?»

В микросервисной архитектуре один клик пользователя может породить цепочку из 50 вызовов разных сервисов. Если "тормозит" страница, метрики покажут "медленно везде". Трассировка покажет конкретный **Span** (отрезок), который занял 90% времени.

- **Как это работает:** Каждому входящему запросу присваивается уникальный `Trace ID`. Этот ID передается в заголовках (Headers) при каждом внутреннем вызове (между сервисами, к базе, к кэшу).
- **OpenTelemetry:** Сейчас это индустриальный стандарт. Используйте его вместо проприетарных агентов, чтобы избежать Vendor Lock-in.
- **Сэмплирование (Sampling):** Трассировать 100% запросов очень дорого. Обычно сохраняют 1–10% трафиков или только ошибочные запросы (Tail-based sampling), чего достаточно для статистики.

4. Алертниг (Alerting): «Когда звать человека?»

Алерты связывают Observability с реальностью. Главный враг SRE — **Alert Fatigue** (Усталость от алертов). Если телефон звонит каждые 5 минут из-за ерунды, настоящий сбой будет пропущен.

Правила здоровых алертов:

1. **Алертить на симптомы (Symptoms), а не на причины (Causes).**
   - *Плохой алерт:* "CPU > 90%". (Может, это штатный процесс индексации?)
   - *Хороший алерт:* "Latency p99 > 2 сек" или "Error Rate > 1%". (Пользователю плохо — значит, надо чинить).
2. **Actionable (Призыв к действию).** Каждый алерт должен иметь Playbook (инструкцию): что делать инженеру, получившему это уведомление. Если инструкции нет — алерт не нужен.

Технологический стек (Примеры)

Не обязательно покупать дорогие Enterprise-решения. Рынок делится на Open Source и SaaS:

Категория	Open Source (Self-hosted)	SaaS (Managed)
Метрики	Prometheus, VictoriaMetrics, Grafana	Datadog Metrics, AWS CloudWatch
Логи	ELK Stack (Elasticsearch, Logstash, Kibana), Loki	Splunk, Datadog Logs
Трассировка	Jaeger, Zipkin, Grafana Tempo	Honeycomb, Dynatrace
Визуализация	Grafana	Datadog Dashboards

Итог: Observability стоит денег и ресурсов CPU. Начинайте с Метрик (дешево и сердито) и Структурированных логов. Добавляйте Трассировку, когда количество микросервисов перевалит за 5–10.

Site Reliability Engineering (SRE): Инженерный подход к эксплуатации

SRE — это дисциплина, зародившаяся в Google, суть которой можно описать одной фразой: **«Что происходит, когда к задачам эксплуатации (Operations) подходит софтверный инженер»**. Вместо ручного администрирования SRE предлагает использовать программный код для построения надежных и масштабируемых систем.

Главная задача SRE — найти баланс между **стабильностью** (надежностью) и **скоростью** (выкаткой новых фич).

Три кита метрик: SLI, SLO и Бюджет ошибок

В основе SRE лежат три ключевых аббревиатуры, которые превращают абстрактное понятие «надежность» в конкретные цифры.

1. SLI (Service Level Indicator) — Индикатор Это «градусник», измеряющий здоровье сервиса.

- *Что это:* Количественная метрика того, что действительно важно для пользователя.
- *Формула:* Обычно это отношение «Хорошие события / Все события».
- *Примеры:*
  - **Доступность:** (Успешные запросы / Всего запросов) * 100%.
  - **Latency:** (Запросы быстрее 200мс / Всего запросов) * 100%.
  - **Data Pipeline:** Свежесть данных (Freshness), пропускная способность (Throughput) и корректность (Correctness).
- *Совет:* Не измеряйте CPU или RAM здесь. Пользователю плевать на загрузку вашего процессора, ему важно, открылась ли страница.

2. SLO (Service Level Objective) — Цель Это целевое значение вашего индикатора. Граница между «мы молодцы» и «у нас проблемы».

- *Принцип:* SLO всегда меньше 100%. Идеальных систем не бывает, а стремление к 100% стоит бесконечно дорого.
- *Пример:* «99.9% запросов к API за календарный месяц должны быть успешными».
- *Важно:* SLO — это внутренняя цель команды. Не путайте с **SLA** (внешний контракт с финансовыми штрафами).

3. Error Budget — Бюджет на ошибки Это самый мощный инструмент SRE. Это «валюта», которой вы платите за инновации.

- *Расчет:* `100% - SLO`. Если цель доступности 99.9%, ваш бюджет на ошибки — **0.1%**.
- *Как это работает:* Если у вас 1 миллион запросов в месяц, вы имеете право «уронить» 1000 из них.
- *Философия:* Пока бюджет не исчерпан, команда может рисковать, экспериментировать и релизить быстро. Как только бюджет исчерпан (0.1% ошибок набралось), приоритет сменяется с разработки новых фич на стабилизацию системы.

Внедрение культуры SRE

SRE — это не только метрики, но и процессы.

1. Борьба с Рутиной (Toil) В SRE есть понятие **Toil** (Рутина). Это операционная работа, которая:

- Ручная и повторяющаяся.
- Не требует творческого подхода.
- Линейно растет с ростом сервиса.
- **Не приносит долгосрочной ценности.**

SRE-инженеры стремятся тратить на рутину не более 50% времени. Остальные 50% должны уходить на инжиниринг — написание кода и автоматизацию, которая эту рутину уничтожит.

2. Error Budget Policy (Политика реагирования) Что делать, если бюджет ошибок кончился? Это должно быть решено «на берегу», а не во время пожара. Пример политики эскалации:

- **Потрачено 50%:** Обсуждаем причины на стендапе.
- **Потрачено 75%:** Отменяем рискованные эксперименты.
- **Потрачено 100%:** Вводится **Code Freeze** (Заморозка релизов). Мы не катим новые фичи, пока система не стабилизируется или пока не начнется новый период (месяц). Весь бэклог замораживается, команда чинит техдолг и баги.

3. Blameless Post-mortem (Разбор полетов без вины) Когда случается инцидент, пишется Post-mortem (постмортем).

- **Главное правило:** Мы не ищем виноватых («Вася уронил прод»). Мы ищем системные причины («Почему система позволила Васе уронить прод одной командой?»).
- **Цель:** Извлечь урок и изменить процессы или архитектуру так, чтобы этот конкретный сбой больше никогда не повторился. Если постмортем написан, но в бэклоге не появилось задач на исправление — время потрачено зря.

Заключение: Культура надежности

Observability и SRE — это не просто набор инструментов, это фундамент современной эксплуатации. В распределенных системах, где сбои неизбежны, эти практики меняют сам подход к надежности: от реактивного «тушения пожаров» к проактивному управлению рисками.

Главные выводы (Key Takeaways):

1. **Полная картина (Observability):** Четыре столпа — **Метрики** (Что?), **Логи** (Почему?), **Трейсинг** (Где?) и **Алертинг** (Когда?) — дают вам «рентгеновское зрение». Без них вы управляете сложной системой вслепую.
2. **Надежность как цифра (SRE):** Фреймворк **SLI/SLO** и **Бюджеты ошибок** превращает надежность из субъективного ощущения («вроде работает нормально») в измеримую инженерную величину. Это позволяет принимать объективные решения: когда можно релизиться, а когда нужно остановиться и чинить техдолг.
3. **Эволюция, а не революция:** Не пытайтесь внедрить всё сразу.
    - *Начните с малого:* Настройте сбор «Золотых сигналов» (Latency, Traffic, Errors, Saturation) для критических путей пользователя.
    - *Добавьте контекст:* Внедрите структурированные логи и базовый трейсинг.
    - *Определите правила:* Договоритесь о первом SLO для самого важного сервиса.

Итог: Инвестиции в Observability всегда окупаются. Сначала вы тратите время на настройку, но затем экономите сотни часов на дебаггинге и устранении инцидентов (MTTR). По мере роста сложности системы эти практики перестают быть «опцией» и становятся единственным способом сохранить контроль над масштабируемым продуктом.