# Архитектура Наблюдаемости Enterprise-уровня: Глубокий анализ OpenTelemetry и Метрик в экосистеме.NET

## 1. Эпистемологический Сдвиг: От Детерминизма к Стохастике в Инженерии ПО

### 1.1. Фундаментальная деконструкция ментальных моделей GameDev и Enterprise

Переход от разработки интерактивных приложений на Unity к проектированию распределенных систем на платформе.NET 8 требует не просто изучения новых синтаксических конструкций или библиотек, но радикальной смены эпистемологической парадигмы — того, как инженер познает состояние создаваемой им системы. Учебный план "Неделя 14" ставит задачу трансформации мышления разработчика, привыкшего к детерминированному игровому циклу, в мышление архитектора, оперирующего вероятностными процессами в распределенной среде.1

В контексте GameDev разработчик существует в "уютной", контролируемой вселенной одного процесса. Игровой цикл (Game Loop) — это жестко синхронизированная последовательность фаз: чтение ввода, обновление логики (Update, FixedUpdate), рендеринг кадра. Время здесь дискретно и измеряется кадрами. Если возникает аномалия — например, баг физики или некорректное состояние объекта — инженер обладает божественной властью остановить время. Используя отладчик Visual Studio, можно заморозить выполнение кода (Breakpoint), инспектировать любой байт в оперативной памяти (Heap Inspector) и даже изменить состояние "на лету", не разрушая целостность игрового мира.1

В противовес этому, экосистема Enterprise-бэкенда представляет собой стохастическую, асинхронную среду. Система бронирования переговорных комнат, которую мы проектируем, не живет в одном процессе. Она распределена между множеством экземпляров микросервисов, баз данных (PostgreSQL), кэшей (Redis) и внешних провайдеров идентификации (OpenIddict).1 В такой среде понятие "сейчас" размывается релятивистскими эффектами сетевых задержек. Попытка применить методы GameDev — например, подключить удаленный отладчик к продакшен-серверу — приведет к катастрофическим последствиям: остановка обработки запросов на одном узле вызовет каскадные тайм-ауты в зависимых системах, перегрузку очередей и отказ в обслуживании для тысяч пользователей.

Здесь классическая отладка (Debugging) умирает, и на её место приходит Наблюдаемость (Observability). Мы переходим от императивного вмешательства ("Остановись и покажи мне переменную X") к декларативному анализу следов ("Покажи мне агрегированную историю изменения переменной X за прошлый час"). Система должна быть спроектирована так, чтобы непрерывно излучать сигналы о своем внутреннем здоровье, позволяя инженеру реконструировать картину реальности постфактум, не касаясь работающих механизмов.2

### 1.2. Теоретические основы Наблюдаемости (Control Theory)

Термин "Наблюдаемость" пришел в программную инженерию из Теории автоматического управления (Control Theory), где он был формализован Рудольфом Калманом. В строгом инженерном смысле система считается наблюдаемой, если её текущее внутреннее состояние может быть полностью восстановлено за конечное время на основе знания только её внешних выходов (телеметрии).4

В приложении к разработке ПО это определение трансформируется в способность системы отвечать на *новые*, ранее не сформулированные вопросы, без необходимости внесения изменений в код и повторного развертывания. Это критическое отличие от традиционного Мониторинга.

| **Характеристика** | **Мониторинг (Monitoring)** | **Наблюдаемость (Observability)** |
| --- | --- | --- |
| **Фокус внимания** | Состояние инфраструктуры и компонентов. | Поведение системы и пользовательский опыт. |
| **Тип вопросов** | "Известные неизвестные" (Known-Unknowns). Мы знаем, что диск может заполниться, и ставим датчик. | "Неизвестные неизвестные" (Unknown-Unknowns). Почему транзакции тормозят только у пользователей iOS в Берлине? 7 |
| **Метафора** | Приборная панель автомобиля (горит лампочка "Check Engine"). | МРТ-сканирование и биохимический анализ крови пациента. |
| **Действие** | Реактивное (Alerting). | Проактивное (Exploration/Debugging).7 |
| **Природа данных** | Агрегаты, предопределенные метрики. | Сырые события, высокая кардинальность, контекст.9 |

Для разработчика, переходящего с Unity, принятие этой парадигмы означает отказ от Console.WriteLine или локальных логов, которые пишут неструктурированный текст "в никуда". В Enterprise-системе данные телеметрии становятся таким же важным продуктом работы сервиса, как и бизнес-функции (бронирование комнат). Если сервис работает, но мы не видим его метрик — он не работает.10

## 2. Архитектура Данных: Три Столпа Наблюдаемости

Индустриальный стандарт декомпозирует наблюдаемость на три фундаментальных типа данных, часто называемых "Тремя Столпами" (The Three Pillars). Однако для архитектора важно понимать не только их определения, но и экономические, физические и когнитивные компромиссы, связанные с каждым типом.11

### 2.1. Метрики (Metrics): Экономика сжатия и агрегации

Метрики — это числовые данные, измеренные в определенные моменты времени. Они представляют собой самый экономичный способ хранения информации о системе.

* **Природа:** Агрегация с потерями. Вместо того чтобы хранить запись о каждом из 10,000 HTTP-запросов, прошедших за минуту, метрика хранит одну запись: http\_requests\_total = 10000 и, возможно, avg\_latency = 50ms.
* **Преимущество:** Объем данных метрик зависит от *количества измеряемых параметров* и *кардинальности меток*, но **не зависит от нагрузки на систему**. Это делает метрики идеальным инструментом для хранения исторических трендов за месяцы и годы.14
* **Ограничение:** Потеря контекста. Глядя на метрику "50 ошибок", вы не знаете, *кто* их вызвал и *какие* данные были в запросах.
* **Аналог в Unity:** Счетчик FPS (Frames Per Second) или график Memory Profiler. Вы видите просадку, но не видите конкретную текстуру, вызвавшую её.

### 2.2. Логи (Logs): Дискретные события и контекст

Логи — это записи о дискретных событиях: "Пользователь X попытался забронировать комнату Y в 14:00".

* **Природа:** Высокая детализация. Логи могут быть структурированными (JSON) или неструктурированными (текст).
* **Стоимость:** Очень высокая. Объем логов растет **линейно** с ростом нагрузки. В высоконагруженных системах логирование каждого запроса (Access Logs) может генерировать терабайты данных в сутки, что делает их хранение экономически нецелесообразным без агрессивного сэмплирования или короткого срока хранения (Retention Policy).3
* **Роль:** Логи необходимы для расследования инцидентов ("Forensic analysis"), когда нужно понять *почему* произошло событие, зафиксированное метрикой.

### 2.3. Трассировка (Distributed Tracing): Каузальность и Связность

Трассировка — это клей, который соединяет события в распределенной системе. Если в Unity у вас есть Stack Trace, показывающий путь вызова функций внутри одного процесса, то в микросервисах этот путь разрывается сетевыми границами.

* **Механизм:** Каждому входящему запросу присваивается уникальный идентификатор (TraceId). Этот ID передается в заголовках (HTTP Headers, gRPC Metadata) при каждом вызове зависимого сервиса (Propagations).
* **Сущности:**
  + **Trace:** Полный путь запроса через всю систему.
  + **Span:** Единица работы внутри одного сервиса (например, "SQL Query" или "Redis Get"). Span содержит ссылки на родительский Span, время начала и конца, а также метаданные (Attributes).11
* **Ценность:** Единственный способ понять, *где* именно происходит задержка (Latency) в цепочке из 10 микросервисов.

### 2.4. Взаимосвязь и Корреляция

Сила системы наблюдаемости не в наличии этих трех столпов по отдельности, а в их **корреляции**.

* **Exemplars (Экземпляры):** Это современная техника (доступная в.NET 8), позволяющая прикрепить TraceId к конкретной точке в метрике (например, к бакету гистограммы). Видя на графике медленный запрос, инженер может кликнуть на точку и перейти сразу к трейсу этого запроса.17
* **Log Correlation:** Внедрение TraceId и SpanId в каждый лог-файл позволяет отфильтровать все логи всех микросервисов, относящиеся к одной конкретной бизнес-транзакции.

## 3. Архитектурный Стандарт: OpenTelemetry (OTel)

Долгое время индустрия страдала от фрагментации. Чтобы собрать метрики, разработчик должен был использовать SDK конкретного вендора (Datadog, NewRelic, Prometheus). Это создавало жесткий "Vendor Lock-in": смена системы мониторинга требовала переписывания кода всего приложения.

Ответом индустрии стал OpenTelemetry (OTel) — проект Cloud Native Computing Foundation (CNCF), который объединил проекты OpenTracing и OpenCensus.

### 3.1. Философия OpenTelemetry

OpenTelemetry — это не бэкенд для хранения данных (как Prometheus или Elasticsearch). Это стандарт и набор инструментов для генерации, сбора и экспорта телеметрии.20

Его главная цель — отделить инструментацию кода (как мы измеряем) от инфраструктуры (куда мы сохраняем).

### 3.2. Компоненты Архитектуры OTel

1. **OTel API:** Интерфейсы, которые используются разработчиками в коде приложения. В.NET 8 ситуация уникальна тем, что Microsoft внедрила спецификации OTel API непосредственно в базовую библиотеку классов (BCL).
   * Для метрик используется System.Diagnostics.Metrics.21
   * Для трассировки используется System.Diagnostics.Activity.21
   * *Важнейший вывод:* Ваш бизнес-код не зависит от библиотек OpenTelemetry.\*. Он зависит только от System.Diagnostics, что делает его чистым и переносимым.
2. **OTel SDK:** Реализация, которая подключается на старте приложения (обычно в Program.cs). Она "подписывается" на сигналы от System.Diagnostics, агрегирует их, обогащает метаданными (Resource Attributes, например, имя пода Kubernetes) и готовит к отправке.
3. **OTLP (OpenTelemetry Protocol):** Универсальный бинарный протокол (обычно поверх gRPC или HTTP/Protobuf) для передачи телеметрии. Это "lingua franca" современной наблюдаемости.23

### 3.3. Паттерн OpenTelemetry Collector

В зрелой Enterprise-архитектуре приложения не должны знать, где хранятся метрики. Они просто "отдают" их локальному агенту или коллектору.

OpenTelemetry Collector — это независимый сервис ("швейцарский нож"), который стоит между приложением и бэкендами.25 Его конвейер (Pipeline) состоит из трех фаз:

1. **Receivers:** Принимают данные (например, по OTLP от приложения).
2. **Processors:** Обрабатывают данные. Примеры:
   * *Batch Processor:* Собирает пачки данных для эффективной отправки.
   * *Memory Limiter:* Защищает коллектор от переполнения памяти.
   * *Attributes Processor:* Добавляет теги (например, cloud.region, environment=production) или удаляет чувствительные данные (PII filtering).
3. **Exporters:** Отправляют данные в конечные системы (Prometheus, Jaeger, Datadog, AWS X-Ray).27

Использование Коллектора позволяет изменить систему мониторинга (например, переехать с Jaeger на Tempo), изменив только конфиг Коллектора, не трогая тысячи микросервисов.

## 4. Метрики в.NET 8: Глубокое Техническое Погружение

В рамках Дня 4 мы фокусируемся на реализации метрик. В.NET 8 это делается через API System.Diagnostics.Metrics, который является нативной реализацией стандарта OTel. Понимание семантики инструментов критически важно для корректного отображения реальности.22

### 4.1. Примитивы Измерения (Instruments)

| **Инструмент (.NET Type)** | **Поведение** | **Семантика** | **Пример использования** | **Unity Аналогия** |
| --- | --- | --- | --- | --- |
| **Counter<T>** | Монотонно возрастает. Только Add(+). | Кумулятивная сумма событий. Сброс только при рестарте процесса. | bookings\_created\_total, http\_requests\_received\_total. | Time.frameCount (всегда растет). |
| **UpDownCounter<T>** | Может расти и падать. Add(+), Add(-). | Текущее значение ресурса или состояния. | active\_connections, queue\_depth, memory\_usage\_bytes. | CurrentHealth игрока. |
| **ObservableGauge<T>** | Асинхронный опрос (Callback). | Измерение внешнего состояния в момент сбора (Scrape). Мы не меняем его явно, рантайм сам вызывает функцию измерения. | cpu\_usage\_percent, temperature\_celsius. | Опрос сенсоров ввода в Update(). |
| **Histogram<T>** | Запись распределения значений. Record(value). | Агрегация данных в "ведра" (buckets) для статистического анализа. | http\_request\_duration\_seconds, db\_query\_latency. | График профайлера с распределением времени кадра. |

### 4.2. Гистограммы: Математика и Ловушка Среднего

Наибольшую сложность и ценность представляют Гистограммы.

Разработчики часто полагаются на среднее значение (Arithmetic Mean). Однако в распределенных системах среднее значение лжет.

Пример: 99 запросов прошли за 10мс, а 1 запрос (из-за сборки мусора или сетевого лага) завис на 10 секунд.

Среднее: $(99 \times 0.01 + 10) / 100 \approx 0.11с$ (110мс).

Менеджер видит 110мс и думает, что все хорошо. Но пользователь, попавший на 10 секунд, в ярости ушел.

Это называется "Flaw of Averages".29

Для решения этой проблемы мы используем **Перцентили (Percentiles)**:

* **P50 (Медиана):** Время, быстрее которого прошло 50% запросов.
* **P95:** Время, быстрее которого прошло 95% запросов.
* **P99 (Tail Latency):** "Хвост" распределения. Это худший опыт, который получают 1% пользователей. В масштабах Google или Amazon 1% — это миллионы запросов, и именно P99 определяет доход (Revenue Impact).32

#### 4.2.1. Реализация Гистограмм и Бакетов (Buckets)

Гистограмма в Prometheus не хранит все значения. Она раскладывает их по "ведрам".

le (less or equal) — это верхняя граница ведра.

Если мы меряем время ответа (Latency), ведра могут быть: 0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2.5, 5, 10 секунд.35

Запрос длительностью 0.15с попадет в ведра 0.25, 0.5, 1 и так далее (кумулятивная гистограмма).

Проблема точности vs. Память:

Чем больше ведер, тем точнее расчет перцентилей через histogram\_quantile. Но каждое ведро — это отдельный временной ряд в базе данных. Слишком много ведер (High Cardinality) приведет к взрывному росту потребления памяти.36

В.NET 8 можно управлять границами ведер через InstrumentAdvice или конфигурацию View в OTel SDK.28

### 4.3. Опасность Высокой Кардинальности (Cardinality Explosion)

Это самая распространенная архитектурная ошибка при внедрении метрик.

Кардинальность метрики — это произведение количества возможных значений всех её меток (labels/tags).

Сценарий катастрофы: Разработчик решает добавить UserId в метрику http\_requests\_total, чтобы отслеживать активность конкретных пользователей.

http\_requests\_total{method="POST", user\_id="guid-1"}

http\_requests\_total{method="POST", user\_id="guid-2"}

Если в системе 1 миллион пользователей, Prometheus попытается создать 1 миллион новых временных рядов. Это приведет к OOM (Out Of Memory) и падению системы мониторинга.15

**Золотое правило:** В метках должны быть только значения с ограниченным набором вариантов (Bounded set):

* ✅ status\_code (200, 400, 500) — ок (сотни вариантов).
* ✅ method (GET, POST) — ок (единицы вариантов).
* ✅ region (US-East, EU-West) — ок.
* ❌ user\_id, trace\_id, session\_id, email — **запрещено**.

Для данных высокой кардинальности (High Cardinality) используются Логи или Трассировка, но не Метрики.

## 5. Инфраструктура Хранения: Prometheus и Модель Pull

### 5.1. Архитектурный спор: Pull vs. Push

В мире мониторинга существует два фундаментальных подхода к сбору данных:

1. **Push (Агенты, например, Datadog, StatsD):** Приложение само инициирует отправку данных на сервер мониторинга.
   * *Риск:* Если сервер мониторинга перегружен или упал, приложение может заблокироваться, пытаясь отправить метрики (если отправка синхронна), или потерять данные (если буфер переполнится). Приложение не знает, "живой" ли мониторинг.
2. **Pull (Prometheus):** Сервер мониторинга сам "приходит" к приложению и забирает данные.
   * *Преимущество:* Prometheus контролирует нагрузку. Если он перегружен, он просто замедлит опрос. Приложение ничего не знает о мониторинге, оно просто выставляет легковесный HTTP-эндпоинт (/metrics).
   * *Health Check:* Если Prometheus не может достучаться до сервиса, он точно знает, что сервис "Down". В Push-модели отсутствие данных может означать как смерть сервиса, так и проблемы сети.39

### 5.2. Реализация в.NET 8 (Prometheus Exporter)

Для нашего проекта мы используем модель Pull.

В код добавляется библиотека OpenTelemetry.Exporter.Prometheus.AspNetCore. Она регистрирует Middleware, который перехватывает запросы к пути /metrics.

Когда Prometheus делает запрос GET /metrics, библиотека:

1. Собирает текущие значения всех счетчиков и гистограмм из памяти процесса (очень быстро, без блокировок, используя Interlocked операции).
2. Сериализует их в текстовый формат Prometheus Exposition Format.
3. Отдает ответ.

Это гарантирует минимальное влияние на производительность основного приложения (Overhead < 1%).42

**Важное уточнение:** Для продакшен-среды с Kubernetes часто используется гибридный подход. Приложения отдают метрики OTel Collector'у по протоколу OTLP (Push внутри кластера), а Prometheus забирает их уже с Коллектора (Pull с централизованной точки).44 Но для обучения (Day 4) мы используем прямой Pull с приложения.

## 6. Прикладная Наблюдаемость: Метод RED и PromQL

Собрать метрики — полдела. Нужно уметь их читать. Универсальным языком запросов для Prometheus является **PromQL**, а методологией визуализации для микросервисов — **RED**.46

### 6.1. Rate (Частота запросов)

Показывает объем трафика. Это "пульс" сервиса.

* **PromQL:** sum(rate(http\_request\_duration\_seconds\_count[1m]))
  + rate(...[1m]): Вычисляет производную (запросов в секунду) на основе данных за последнюю минуту. Обрабатывает сброс счетчиков при перезапуске сервиса.
  + sum(...): Агрегирует данные со всех инстансов сервиса (подов), чтобы получить общий трафик.

### 6.2. Errors (Ошибки)

Показывает корректность работы.

* **PromQL:** sum(rate(http\_request\_duration\_seconds\_count{http\_response\_status\_code=~"5.."}[1m]))
  + Фильтр status\_code=~"5.." выбирает только серверные ошибки (Internal Server Error). Ошибки клиента (4xx) обычно мониторят отдельно, так как они могут быть вызваны некорректными действиями пользователей, а не поломкой системы.

### 6.3. Duration (Длительность/Задержка)

Самый сложный и важный компонент.

* **PromQL (P99):** histogram\_quantile(0.99, sum(rate(http\_request\_duration\_seconds\_bucket[5m])) by (le))
  + rate(...bucket[5m]): Вычисляет скорость наполнения каждого ведра гистограммы.
  + sum(...) by (le): Суммирует ведра от разных инстансов, сохраняя разбивку по границам (le). Это критически важно: **нельзя усреднять перцентили**. Перцентиль от агрегата гистограмм — это математически корректная операция.37
  + histogram\_quantile(0.99,...): Аппроксимирует значение, ниже которого находятся 99% запросов.

### 6.4. Проектирование Дашбордов в Grafana

При создании визуализации (Grafana) следует избегать "Alert Fatigue" (усталости от тревог).

* **Best Practice:** Дашборд должен рассказывать историю. Сверху — светофор RED (общий статус). Ниже — разбивка по эндпоинтам (POST /booking vs GET /rooms). Еще ниже — ресурсы инфраструктуры (CPU/Memory - метод USE).
* Визуализация P99 и P50 на одном графике позволяет сразу оценить разброс (jitter) задержки.50

## 7. Практическое Задание Дня 4: Интеграция и Анализ

На основе изученной теории, практическая реализация в проекте "Meeting Room Booking System" включает следующие шаги:

1. **Инструментация:** Внедрение Meter и ActivitySource в слой бизнес-логики. Создание кастомных метрик:
   * bookings\_created\_total (Counter) — бизнес-метрика успеха.
   * booking\_conflict\_total (Counter) — метрика коллизий (Double Booking).
2. **Middleware:** Подключение стандартной инструментации ASP.NET Core (AddAspNetCoreInstrumentation), которая автоматически даст RED-метрики для всех HTTP-эндпоинтов без написания кода.52
3. **Экспорт:** Настройка эндпоинта /metrics для Prometheus.
4. **Визуализация:** Построение дашборда в Grafana, который отвечает на вопрос: "Может ли пользователь забронировать комнату прямо сейчас?".

Эта работа завершает трансформацию инженерного подхода: от "слепой" веры в работоспособность кода к доказательной базе, основанной на телеметрии. Для Middle+ разработчика это переход от написания кода к владению системой в продакшене.

**Использованные источники:**

* 1 - Контекст учебного плана.
* 2 - Теория: Мониторинг vs Наблюдаемость.
* 20 - Архитектура OpenTelemetry.
* 21 - Реализация Metrics API в.NET.
* 46 - Метод RED и PromQL.
* 31 - Статистика, перцентили, "Flaw of Averages".
* 15 - Кардинальность и гистограммы.
* 39 - Pull vs Push модели.
* 17 - Exemplars и корреляция.

#### Источники

1. Неделя 14
2. Observability vs. monitoring: What's the difference? - Dynatrace, дата последнего обращения: декабря 10, 2025, <https://www.dynatrace.com/news/blog/observability-vs-monitoring/>
3. Monitoring and Observability - Cindy Sridharan - Medium, дата последнего обращения: декабря 10, 2025, <https://copyconstruct.medium.com/monitoring-and-observability-8417d1952e1c>
4. What Is Observability? | IBM, дата последнего обращения: декабря 10, 2025, <https://www.ibm.com/think/topics/observability>
5. What is observability? An In-Depth Guide - InfluxData, дата последнего обращения: декабря 10, 2025, <https://www.influxdata.com/what-is-observability/>
6. Observability in Software Engineering – Metrics, Logs, Traces - Mia-Platform, дата последнего обращения: декабря 10, 2025, <https://mia-platform.eu/blog/observability-software-engineering/>
7. Observability vs Monitoring - Difference Between Data-Based Processes - AWS, дата последнего обращения: декабря 10, 2025, <https://aws.amazon.com/compare/the-difference-between-monitoring-and-observability/>
8. Observability vs. Monitoring: What's the Difference? - IBM, дата последнего обращения: декабря 10, 2025, <https://www.ibm.com/think/topics/observability-vs-monitoring>
9. Observability vs monitoring: Understanding the difference - Graphite, дата последнего обращения: декабря 10, 2025, <https://graphite.com/guides/observability-vs-monitoring>
10. Monitoring, Logging And Observability, дата последнего обращения: декабря 10, 2025, <https://ntombizakhona.medium.com/monitoring-logging-and-observability-760cb718dbe1>
11. The Three Pillars of Observability: Logs, Metrics, and Traces - CrowdStrike, дата последнего обращения: декабря 10, 2025, <https://www.crowdstrike.com/en-us/cybersecurity-101/observability/three-pillars-of-observability/>
12. What Is Observability? | Datadog, дата последнего обращения: декабря 10, 2025, <https://www.datadoghq.com/knowledge-center/observability/>
13. Three Pillars of Observability: Logs, Metrics and Traces | IBM, дата последнего обращения: декабря 10, 2025, <https://www.ibm.com/think/insights/observability-pillars>
14. The Three Pillars Of Observability Logs Metrics And Traces - Netdata, дата последнего обращения: декабря 10, 2025, <https://www.netdata.cloud/academy/pillars-of-observability/>
15. Three Pillars of Observability Explained: Metrics, Logs, Traces - StrongDM, дата последнего обращения: декабря 10, 2025, <https://www.strongdm.com/blog/three-pillars-of-observability>
16. How to get the answers you deserve using the three pillars of observability - Dynatrace, дата последнего обращения: декабря 10, 2025, <https://www.dynatrace.com/news/blog/how-to-get-the-answers-you-deserve-using-the-three-pillars-of-observability/>
17. Using exemplars | OpenTelemetry, дата последнего обращения: декабря 10, 2025, <https://opentelemetry.io/docs/languages/dotnet/metrics/exemplars/>
18. OpenTelemetry Metrics, дата последнего обращения: декабря 10, 2025, <https://opentelemetry.io/docs/specs/otel/metrics/>
19. Connecitng Metrics ↔ Traces with Exemplars in OpenTelemetry - OneUptime, дата последнего обращения: декабря 10, 2025, <https://oneuptime.com/blog/post/2025-09-22-connecting-metrics-to-traces-with-exemplars/view>
20. What is OpenTelemetry?, дата последнего обращения: декабря 10, 2025, <https://opentelemetry.io/docs/what-is-opentelemetry/>
21. .NET Observability with OpenTelemetry - .NET | Microsoft Learn, дата последнего обращения: декабря 10, 2025, <https://learn.microsoft.com/en-us/dotnet/core/diagnostics/observability-with-otel>
22. System.Diagnostics.Metrics Namespace - Microsoft Learn, дата последнего обращения: декабря 10, 2025, <https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.metrics?view=net-10.0>
23. OpenTelemetry Collector vs Exporter - Key Differences Explained | SigNoz, дата последнего обращения: декабря 10, 2025, <https://signoz.io/guides/opentelemetry-collector-vs-exporter/>
24. A Practical Guide to the OpenTelemetry OTLP Receiver - Dash0, дата последнего обращения: декабря 10, 2025, <https://www.dash0.com/guides/opentelemetry-otlp-receiver>
25. Components - OpenTelemetry, дата последнего обращения: декабря 10, 2025, <https://opentelemetry.io/docs/concepts/components/>
26. Collector - OpenTelemetry, дата последнего обращения: декабря 10, 2025, <https://opentelemetry.io/docs/collector/>
27. Architecture | OpenTelemetry, дата последнего обращения: декабря 10, 2025, <https://opentelemetry.io/docs/collector/architecture/>
28. Creating Metrics - .NET - Microsoft Learn, дата последнего обращения: декабря 10, 2025, <https://learn.microsoft.com/en-us/dotnet/core/diagnostics/metrics-instrumentation>
29. Limit Work in Progress without Work In Progress Limits | by Benjamin Huser-Berta | Medium, дата последнего обращения: декабря 10, 2025, <https://medium.com/@benjihuser/limit-work-in-progress-without-work-in-progress-limits-33ee889f661d>
30. Metrics — Home — Nicolas Brown, дата последнего обращения: декабря 10, 2025, <https://www.nicolasbrown.co.uk/blog/tag/Metrics>
31. Why average latency is a terrible way to track website performance ... and how to fix it., дата последнего обращения: декабря 10, 2025, <https://mvolo.com/why-average-latency-is-a-terrible-way-to-track-website-performance-and-how-to-fix-it/>
32. What Is P99 Latency? Understanding the 99th Percentile of Performance | Aerospike, дата последнего обращения: декабря 10, 2025, <https://aerospike.com/blog/what-is-p99-latency/>
33. Why is tail latency (p95/p99) often more important than average latency for evaluating the performance of a vector search in user-facing applications? - Milvus, дата последнего обращения: декабря 10, 2025, <https://milvus.io/ai-quick-reference/why-is-tail-latency-p95p99-often-more-important-than-average-latency-for-evaluating-the-performance-of-a-vector-search-in-userfacing-applications>
34. Why is tail latency (p95/p99) often more important than average latency for evaluating the performance of a vector search in user-facing applications? - Zilliz Vector Database, дата последнего обращения: декабря 10, 2025, <https://zilliz.com/ai-faq/why-is-tail-latency-p95p99-often-more-important-than-average-latency-for-evaluating-the-performance-of-a-vector-search-in-userfacing-applications>
35. ASP.NET Core built-in metrics - Microsoft Learn, дата последнего обращения: декабря 10, 2025, <https://learn.microsoft.com/en-us/aspnet/core/log-mon/metrics/built-in?view=aspnetcore-10.0>
36. Understanding Buckets in Prometheus: A Comprehensive Guide with Real-Time Examples, дата последнего обращения: декабря 10, 2025, <https://www.atatus.com/blog/what-is-a-bucket-in-prometheus/>
37. A Deep Dive Into the Four Types of Prometheus Metrics | Tiger Data, дата последнего обращения: декабря 10, 2025, <https://www.tigerdata.com/blog/four-types-prometheus-metrics-to-collect>
38. OBI metrics cardinality | OpenTelemetry, дата последнего обращения: декабря 10, 2025, <https://opentelemetry.io/docs/zero-code/obi/cardinality/>
39. Is Prometheus Monitoring Push or Pull? - SigNoz, дата последнего обращения: декабря 10, 2025, <https://signoz.io/guides/is-prometheus-monitoring-push-or-pull/>
40. Pull doesn't scale - or does it? - Prometheus, дата последнего обращения: декабря 10, 2025, <https://prometheus.io/blog/2016/07/23/pull-does-not-scale-or-does-it/>
41. Pull or Push: How to Select Monitoring Systems? - Alibaba Cloud Community, дата последнего обращения: декабря 10, 2025, <https://www.alibabacloud.com/blog/pull-or-push-how-to-select-monitoring-systems_599007>
42. OpenTelemetry.Exporter.Prometheus.AspNetCore 1.14.0-beta.1 - NuGet, дата последнего обращения: декабря 10, 2025, <https://www.nuget.org/packages/OpenTelemetry.Exporter.Prometheus.AspNetCore>
43. Exporters - OpenTelemetry, дата последнего обращения: декабря 10, 2025, <https://opentelemetry.io/docs/languages/dotnet/exporters/>
44. Metrics: Difference between OpenTelemetry.Exporter.Prometheus.AspNetCore and OpenTelemetry.Exporter.OpenTelemetryProtocol · open-telemetry opentelemetry-dotnet · Discussion #4495 - GitHub, дата последнего обращения: декабря 10, 2025, <https://github.com/open-telemetry/opentelemetry-dotnet/discussions/4495>
45. Export to Prometheus and Grafana - OpenTelemetry, дата последнего обращения: декабря 10, 2025, <https://opentelemetry.io/docs/languages/dotnet/metrics/getting-started-prometheus-grafana/>
46. RED and USE Metrics for Monitoring and Observability | Better Stack Community, дата последнего обращения: декабря 10, 2025, <https://betterstack.com/community/guides/monitoring/red-use-metrics/>
47. Microservices Monitoring with the RED Method - Last9, дата последнего обращения: декабря 10, 2025, <https://last9.io/blog/monitoring-with-red-method/>
48. Mastering Observability in SRE: Golden Signals, RED & USE Metrics - Medium, дата последнего обращения: декабря 10, 2025, <https://medium.com/@farhanramzan799/mastering-observability-in-sre-golden-signals-red-use-metrics-005656c4fe7d>
49. Understanding histogram\_quantile based on rate in Prometheus - Stack Overflow, дата последнего обращения: декабря 10, 2025, <https://stackoverflow.com/questions/55162093/understanding-histogram-quantile-based-on-rate-in-prometheus>
50. Getting started with Grafana: best practices to design your first dashboard, дата последнего обращения: декабря 10, 2025, <https://grafana.com/blog/2024/07/03/getting-started-with-grafana-best-practices-to-design-your-first-dashboard/>
51. Choose a metric for your investigation | Grafana documentation, дата последнего обращения: декабря 10, 2025, <https://grafana.com/docs/grafana/latest/visualizations/simplified-exploration/traces/investigate/choose-red-metric/>
52. Getting started with metrics - ASP.NET Core - OpenTelemetry, дата последнего обращения: декабря 10, 2025, <https://opentelemetry.io/docs/languages/dotnet/metrics/getting-started-aspnetcore/>
53. Example: Use OpenTelemetry with Prometheus, Grafana, and Jaeger - .NET, дата последнего обращения: декабря 10, 2025, <https://learn.microsoft.com/en-us/dotnet/core/diagnostics/observability-prgrja-example>
54. The RED Method: How to Instrument Your Services | Grafana Labs, дата последнего обращения: декабря 10, 2025, <https://grafana.com/blog/2018/08/02/the-red-method-how-to-instrument-your-services/>
55. Essential Prometheus Queries: Simple to Advanced - Last9, дата последнего обращения: декабря 10, 2025, <https://last9.io/blog/prometheus-query-examples/>
56. P50 vs P95 vs P99 Latency: What These Percentiles Actually Mean (And How to Use Them), дата последнего обращения: декабря 10, 2025, <https://www.reddit.com/r/sre/comments/1nhlltm/p50_vs_p95_vs_p99_latency_what_these_percentiles/>
57. OpenTelemetry Exponential Histograms - New Relic, дата последнего обращения: декабря 10, 2025, <https://newrelic.com/blog/best-practices/opentelemetry-histograms>
58. Opentelemetry vs Prometheus | IBM, дата последнего обращения: декабря 10, 2025, <https://www.ibm.com/think/topics/opentelemetry-vs-prometheus>