Дисципліна

«Бекенд-розробка, сервіси та мікросервіси»

КОНСПЕКТ ЛЕКЦІЇ. ТЕМА 17

Моніторинг, логування та обробка помилок у розподілених системах

Спеціальність: Комп'ютерна інженерія, 3 курс
Обсяг: 1 год 20 хв (лекція 17 з 18)

План лекції

1. Навіщо потрібні моніторинг і логування
2. Логування в PHP: рівні важливості та структуровані логи
3. Централізоване логування: агрегація з багатьох сервісів
4. Метрики: що вимірювати і як
5. Розподілене трасування: відстеження запиту через кілька сервісів
6. Обробка помилок та алертинг
7. Практичний приклад: діагностика інциденту

> *Мета лекції: зрозуміти, чому в розподіленій системі неможливо просто «подивитись у консоль», щоб знайти проблему, і опанувати три стовпи спостережуваності (observability) — логи, метрики й трасування — які разом дозволяють діагностувати інциденти в системі з десятків мікросервісів.*

1. Навіщо потрібні моніторинг і логування

У монолітному застосунку (тема 12), що працює на одному сервері, діагностика проблеми відносно проста: подивитися в один файл логів чи консоль, побачити помилку, знайти рядок коду. У мікросервісній архітектурі, розглянутій із теми 12, той самий запит користувача може пройти через API Gateway (тема 14), потім через п'ять-десять різних сервісів, кожен з яких запущено в кількох екземплярах (тема 14) на різних серверах чи контейнерах (тема 15).

Питання «чому цей конкретний запит користувача виконувався 8 секунд замість звичних 200 мілісекунд?» чи «чому 2% запитів повертають помилку 500?» стає значно складнішим: потрібно зібрати інформацію з десятків різних джерел і відновити повну картину того, що відбувалося.

Стовп спостережуваності	Відповідає на запитання
Логи (Logs)	Що саме сталося в конкретний момент часу в конкретному сервісі?
Метрики (Metrics)	Як система поводиться загалом протягом часу (навантаження, швидкість, помилки)?
Трасування (Tracing)	Яким шляхом пройшов конкретний запит через усі сервіси системи?

> *Ці три стовпи (логи, метрики, трасування) прийнято називати observability (спостережуваність) — властивість системи, що дозволяє зрозуміти її внутрішній стан, спостерігаючи лише за зовнішніми сигналами, які вона видає, без необхідності "зупиняти й розбирати" систему для діагностики.*

2. Логування в PHP: рівні важливості та структуровані логи

Логування — запис подій, що відбуваються в застосунку, у постійне сховище (файл, базу даних, зовнішній сервіс) для подальшого аналізу. У PHP-екосистемі стандартом де-факто є бібліотека Monolog, яка реалізує інтерфейс PSR-3, спільний для більшості фреймворків.

2.1. Рівні важливості логів

Рівень	Коли використовувати
DEBUG	Детальна технічна інформація, потрібна лише під час розробки
INFO	Звичайні події в роботі застосунку (користувач увійшов, замовлення створено)
WARNING	Щось незвичне, але не критичне (застарілий метод API, повільний запит)
ERROR	Помилка, що завадила виконати конкретну операцію (не вдалося зберегти замовлення)
CRITICAL	Серйозна помилка, що загрожує роботі всього застосунку (втрачено з'єднання з БД)

```
<?php
use Monolog\Logger;
use Monolog\Handler\StreamHandler;

$logger = new Logger('shop');
$logger->pushHandler(new StreamHandler(__DIR__ . '/app.log',
Logger::DEBUG));

$logger->info('Замовлення створено', ['orderId' => 501, 'userId' => 42]);
$logger->error('Не вдалося списати кошти', ['orderId' => 501, 'reason' =>
'insufficient_funds']);
```

2.2. Структуровані логи

Другий аргумент методу info()/error() — масив контексту — це і є структуроване логування: замість одного довгого текстового рядка, який важко автоматично розібрати, лог записується як набір іменованих полів (orderId, userId, reason). Такий формат (найчастіше — JSON) дозволяє системам агрегації логів (розділ 3) фільтрувати й шукати за конкретними полями, а не за збігом тексту.

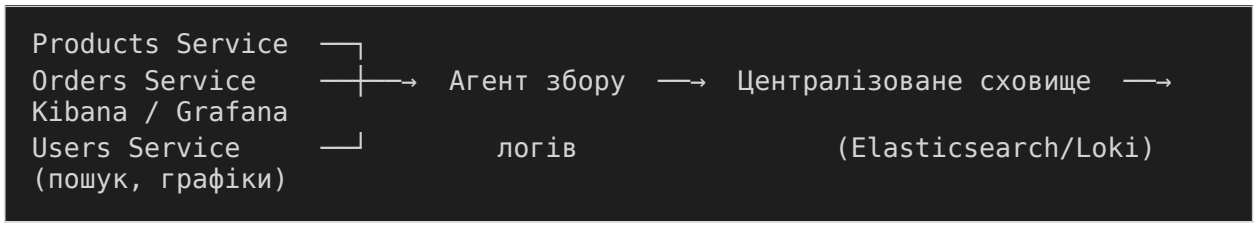
```
// Структурований лог у форматі JSON (як його збереже Monolog)
{
  "message": "Замовлення створено",
  "context": {"orderId": 501, "userId": 42},
  "level": "INFO",
  "datetime": "2026-01-15T14:32:07+02:00"
}
```

> *Правило, якого варто дотримуватись із самого початку проєкту: ніколи не записувати змінні дані (ID користувача, ID замовлення) просто конкатенацією в текст повідомлення ("Order $orderId failed"). Завжди виносити їх у масив контексту — коли знадобиться знайти всі логи для orderId=501 серед мільйонів записів, структуровані поля роблять це миттєвим, а пошук підрядка в тексті — повільним і ненадійним.*

3. Централізоване логування: агрегація з багатьох сервісів

Якщо кожен із двадцяти мікросервісів (тема 12) пише логи у власний локальний файл на власному сервері чи контейнері, знайти всі записи, що стосуються одного інциденту, означає вручну підключатися до двадцяти різних машин. Централізоване логування вирішує це, збираючи логи з усіх сервісів в одне спільне сховище, доступне для пошуку через єдиний інтерфейс.

Компонент	Роль
Агент збору (напр. Filebeat, Fluentd)	Запущений поряд із кожним сервісом; читає локальні логи й пересилає їх далі
Сховище логів (напр. Elasticsearch, Loki)	Централізовано зберігає й індексує логи з усіх сервісів
Інтерфейс пошуку (напр. Kibana, Grafana)	Дозволяє шукати, фільтрувати й візуалізувати логи через веброзсерфер



> *Популярний набір інструментів для централізованого логування називають ELK-стеком (Elasticsearch, Logstash, Kibana) чи його легшою альтернативою — Loki у поєднанні з Grafana. Для навчальних і невеликих проєктів такий стек часто надлишковий (принцип MonolithFirst із теми 12 застосовний і тут) — проте розуміти принцип важливо для роботи з реальними production-системами.*

4. Метрики: що вимірювати і як

Метрика — числове значення, що вимірюється регулярно протягом часу (наприклад, щосекунди) і дозволяє побачити тенденції та аномалії, яких не видно в окремих записах логів. Якщо лог розповідає про одну конкретну подію, метрика розповідає про загальну поведінку системи.

Категорія метрик	Приклад
Продуктивність (latency)	Середній і 95-й перцентиль часу відповіді API-ендпоінта
Пропускна здатність (throughput)	Кількість запитів за секунду до кожного сервісу
Частота помилок (error rate)	Відсоток запитів, що завершились кодом 5xx (тема 10)
Використання ресурсів	Завантаження CPU, пам'яті, місця на диску кожного контейнера (тема 15)
Бізнес-метрики	Кількість оформлених замовлень за годину, середній чек

Prometheus — популярний інструмент для збору й зберігання метрик: він періодично "опитує" (scrape) кожен сервіс через спеціальний ендпоінт /metrics, де сервіс публікує свої поточні показники у стандартизованому текстовому форматі.

```
# Приклад відповіді ендпоінта /metrics (формат Prometheus)
http_requests_total{method="GET",status="200"} 15420
http_requests_total{method="GET",status="500"} 37
http_request_duration_seconds{quantile="0.95"} 0.240
```

> *95-й перцентиль (p95) часто інформативніший за середнє значення: якщо 95% запитів обробляються швидко, а 5% — дуже повільно, середнє значення може виглядати прийнятним, хоча значна частина користувачів насправді має поганий досвід. p95 і p99 явно показують, наскільки поганим є "найгірший типовий" випадок.*

5. Розподілене трасування: відстеження запиту через кілька сервісів

Навіть маючи логи й метрики з кожного сервісу окремо, складно відповісти на запитання "який саме сервіс сповільнив цей конкретний запит користувача, що пройшов через API Gateway, Products, Orders і Payments?". Розподілене трасування вирішує це, присвоюючи кожному вхідному запиту унікальний ідентифікатор (trace ID), який передається від сервісу до сервіса разом із запитом (тема 13) і додається до кожного логу й виміру, пов'язаного з обробкою саме цього запиту.

```
<?php
// API Gateway генерує trace ID для нового запиту (чи бере вже наявний)
$traceId = $_SERVER['HTTP_X_TRACE_ID'] ?? bin2hex(random_bytes(8));

// Trace ID передається в заголовку до кожного внутрішнього виклику (тема
13)
curl_setopt($ch, CURLOPT_HTTPHEADER, ["X-Trace-Id: $traceId"]);

// І додається до кожного запису логу в цьому сервісі
$logger->info('Перевірка наявності товару', ['traceId' => $traceId,
'productId' => 42]);
```

Маючи один trace ID, інженер може знайти в централізованому сховищі логів (розділ 3) усі записи від усіх сервісів, що стосуються саме цього запиту, і відновити повну хронологію: скільки часу запит провів у Gateway, скільки — очікуючи на відповідь Products, скільки — на Orders, і де саме виникла затримка чи помилка.

> *Інструменти на кшталт Jaeger чи Zipkin автоматизують цей процес, візуалізуючи весь шлях запиту у вигляді "водоспадної" діаграми (waterfall), де наочно видно тривалість обробки в кожному сервісі та паралельні чи послідовні виклики між ними — це значно швидше, ніж вручну зіставляти окремі логи за trace ID.*

6. Обробка помилок та алертинг

Зібрані логи, метрики й трасування марні, якщо ніхто не дізнається про проблему вчасно. Алертинг (alerting) — механізм, що автоматично сповіщає відповідальну команду, коли метрики виходять за встановлені межі (наприклад, частота помилок перевищує 5% чи час відповіді перевищує 2 секунди).

```
# Приклад правила алерту (спрощено, у стилі Prometheus Alertmanager)
alert: HighErrorRate
condition: error_rate > 0.05  за останні 5 хвилин
action: надіслати повідомлення в Slack команді чергування
```

Важливий принцип якісного алертингу — уникати "втоми від сповіщень" (alert fatigue): якщо система надсилає десятки несуттєвих сповіщень щодня, команда неминуче почне їх ігнорувати, і справді критичний алерт може загубитися серед шуму. Алерти повинні налаштовуватись на дійсно значущі відхилення, що вимагають людського втручання.

> *Обробку помилок на рівні коду варто поєднувати зі спостережуваністю: коли catch-блок (тема 7) перехоплює виняток, недостатньо просто показати користувачу узагальнене повідомлення (тема 7) — деталі помилки обов'язково логуються з повним контекстом (стек викликів, вхідні параметри, trace ID), інакше сама наявність проблеми залишиться непоміченою.*

7. Практичний приклад: діагностика інциденту

Розглянемо, як три стовпи спостережуваності разом допомагають розслідувати типовий інцидент: користувачі скаржаться, що оформлення замовлень стало повільним.

- Алертинг (розділ 6) спрацьовує: метрика p95 часу відповіді ендпоінта POST /api/orders перевищила поріг 2 секунди — команда отримує сповіщення в Slack.
- Інженер відкриває дашборд метрик (розділ 4) і бачить, що уповільнення почалося о 14:32 і збігається зі сплеском кількості запитів — можливо, причина в навантаженні.
- Інженер переглядає кілька трасувань (розділ 5) повільних запитів за цей період і бачить, що в кожному з них найбільше часу займає виклик до Payments Service, а не сама операція збереження замовлення.
- Інженер переходить до централізованих логів (розділ 3), відфільтрованих за іменем сервіса Payments і часовим проміжком 14:30–14:35, і знаходить повторювані записи WARNING про повільні відповіді від зовнішнього платіжного шлюзу.
- Висновок: проблема не в коді самого застосунку, а в тимчасовому уповільненні стороннього платіжного провайдера — рішення: додати таймаут і Circuit Breaker (тема 13) для цього зовнішнього виклику, щоб уповільнення одного провайдера не блокувало весь процес оформлення замовлення.

> *Цей приклад демонструє головну цінність спостережуваності: без метрик команда взагалі не дізналася б про проблему вчасно; без трасування довелося б вручну перевіряти кожен сервіс по черзі; без структурованих логів (розділ 2) не вдалося б швидко знайти причину серед мільйонів записів. Разом усі три інструменти скоротили розслідування з годин до хвилин.*

Питання для самоперевірки

- Чому діагностика проблеми в мікросервісній архітектурі складніша, ніж у монолітному застосунку?
- На яке з трьох запитань (що сталося? як система поводиться загалом? яким шляхом пройшов запит?) відповідає кожен зі стовпів спостережуваності?
- Чому структуровані логи (з контекстом у вигляді полів) кращі за звичайний текстовий рядок з конкатенованими змінними?
- Яку проблему вирішує централізоване логування порівняно з логами, розкиданими по окремих серверах?
- Чому 95-й перцентиль часу відповіді часто інформативніший за середнє значення?
- Як trace ID дозволяє зібрати логи одного запиту з кількох різних сервісів?
- Що таке "втома від сповіщень" і чому вона небезпечна для якісного алертингу?
- У наведеному прикладі діагностики інциденту — яку саме інформацію дав кожен із трьох стовпів спостережуваності (метрики, трасування, логи)?

Короткий глосарій

Термін	Означення
Observability (спостережуваність)	Властивість системи, що дозволяє зрозуміти її стан через зовнішні сигнали
Структуровані логи	Записи логів у вигляді іменованих полів, а не вільного тексту
Централізоване логування	Збір логів з усіх сервісів в одне спільне сховище для пошуку
Метрика	Числове значення, що регулярно вимірюється протягом часу
Перцентиль (p95, p99)	Значення, нижче якого потрапляє задана частка вимірювань
Trace ID	Унікальний ідентифікатор запиту, що передається між сервісами для трасування
Розподілене трасування	Відстеження шляху одного запиту через кілька сервісів
Алертинг (alerting)	Автоматичне сповіщення команди при виході метрик за встановлені межі