



ТЕМА 17

Моніторинг та логування

Логи · метрики · розподілене трасування · обробка помилок у розподілених системах

Бекенд-розробка, сервіси та мікросервіси

Комп'ютерна інженерія · 3 курс · Лекція 17 з 18

7 питань сьогоднішньої теми

01

Навіщо моніторинг і
логування

02

Логування в PHP

03

Централізоване
логування

04

Метрики: що
вимірювати

05

Розподілене трасування

06

Обробка помилок та
алертинг

07

Діагностика інциденту

Три стовпи спостережуваності

Логи

Що саме сталося в конкретний момент?

Метрики

Як система поводить ся загалом?

Трасування

Яким шляхом пройшов запит?

Рівні важливості логів

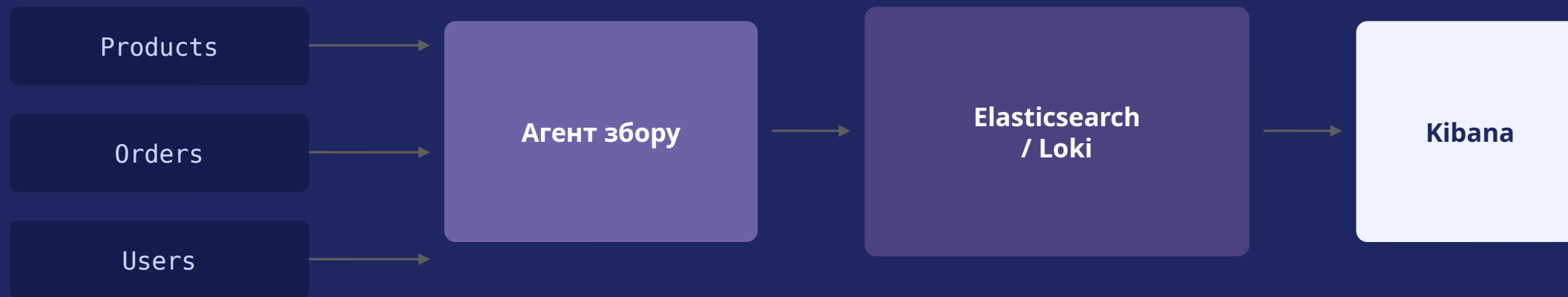
Рівень	Коли використовувати
DEBUG	Технічні деталі лише під час розробки
INFO	Звичайні події (вхід, замовлення створено)
WARNING	Незвичне, але не критичне
ERROR	Помилка, що завадила конкретній операції
CRITICAL	Загроза роботі всього застосунку

Structured logging з Monolog

```
$logger = new Logger('shop');  
$logger->pushHandler(new StreamHandler(__DIR__ . '/app.log'));  
  
$logger->info('Замовлення створено', ['orderId' => 501, 'userId' => 42]);  
$logger->error('Не вдалося списати кошти', ['orderId' => 501]);
```

Ніколи: "Order \$orderId failed" (конкатенація в текст). **Завжди:** ['orderId' => \$orderId] — окреме поле контексту.

Збір логів з усіх сервісів



Замість підключення до двадцяти різних серверів — один інтерфейс пошуку по логах усіх сервісів одночасно.

П'ять категорій метрик

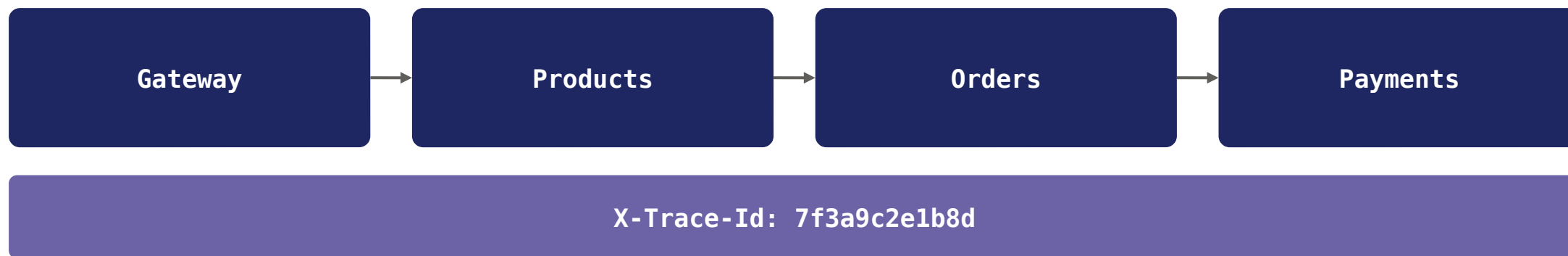
Категорія	Приклад
Продуктивність	Час відповіді API (p95)
Пропускна здатність	Запитів за секунду
Частота помилок	Відсоток відповідей 5xx
Ресурси	CPU, пам'ять контейнера
Бізнес-метрики	Замовлень за годину

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
```

p95 часто інформативніший за середнє: якщо 95% запитів швидкі, а 5% — дуже повільні, середнє виглядає прийнятним, хоча частина користувачів має поганий досвід.

Один trace ID через усі сервіси



Той самий ідентифікатор передається в заголовку запиту (тема 13) і додається до кожного логу — дозволяє відновити повну хронологію одного запиту через усі сервіси.

Trace ID у коді PHP

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

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

// Додається до кожного логу в цьому сервісі
$logger->info('Перевірка товару', ['traceId' => $traceId]);
```

Jaeger чи Zipkin візуалізують шлях запиту як "водоспадну" діаграму (waterfall).

Алертинг: сповіщення вчасно

```
alert: HighErrorRate  
condition: error_rate > 0.05 за останні 5 хвилин  
action: надіслати повідомлення в Slack команді чергування
```

Alert fatigue: якщо система надсилає десятки несуттєвих сповіщень щодня, команда почне їх ігнорувати — і справді критичний алерт загубиться серед шуму.

Розслідування: від алерту до причини

- 1 Алертинг: p95 часу відповіді POST /api/orders перевищив поріг
- 2 Дашборд метрик: уповільнення збігається зі сплеском запитів
- 3 Трасування: найбільше часу займає виклик до Payments Service
- 4 Централізовані логи: WARNING про повільні відповіді від платіжного шлюзу
- 5 Висновок: додати таймаут і Circuit Breaker (тема 13) для зовнішнього виклику

Підсумок теми 17

- Три стовпи спостережуваності: логи, метрики, трасування
- Структуровані логи (поля контексту) шукати й фільтрувати значно легше за текст
- Централізоване логування збирає логи всіх сервісів в одне місце для пошуку
- p95/p99 показують "найгірший типовий" досвід краще за середнє значення
- Trace ID дозволяє відновити шлях одного запиту через усі сервіси
- Якісний алертинг сповіщає лише про значущі відхилення — без "втоми від сповіщень"