

ТЕМА 14

Інфраструктура мікросервісів

*API Gateway · Service Discovery · конфігурація · кешування · масштабування
· балансування навантаження*

Бекенд-розробка, сервіси та мікросервіси

Комп'ютерна інженерія · 3 курс · Лекція 14 з 18

7 питань сьогоднішньої теми

01

API Gateway

02

Service Discovery

03

**Конфігурація
мікросервісів**

04

**Кешування в
розподіленій системі**

05

**Горизонтальне й
вертикальне
масштабування**

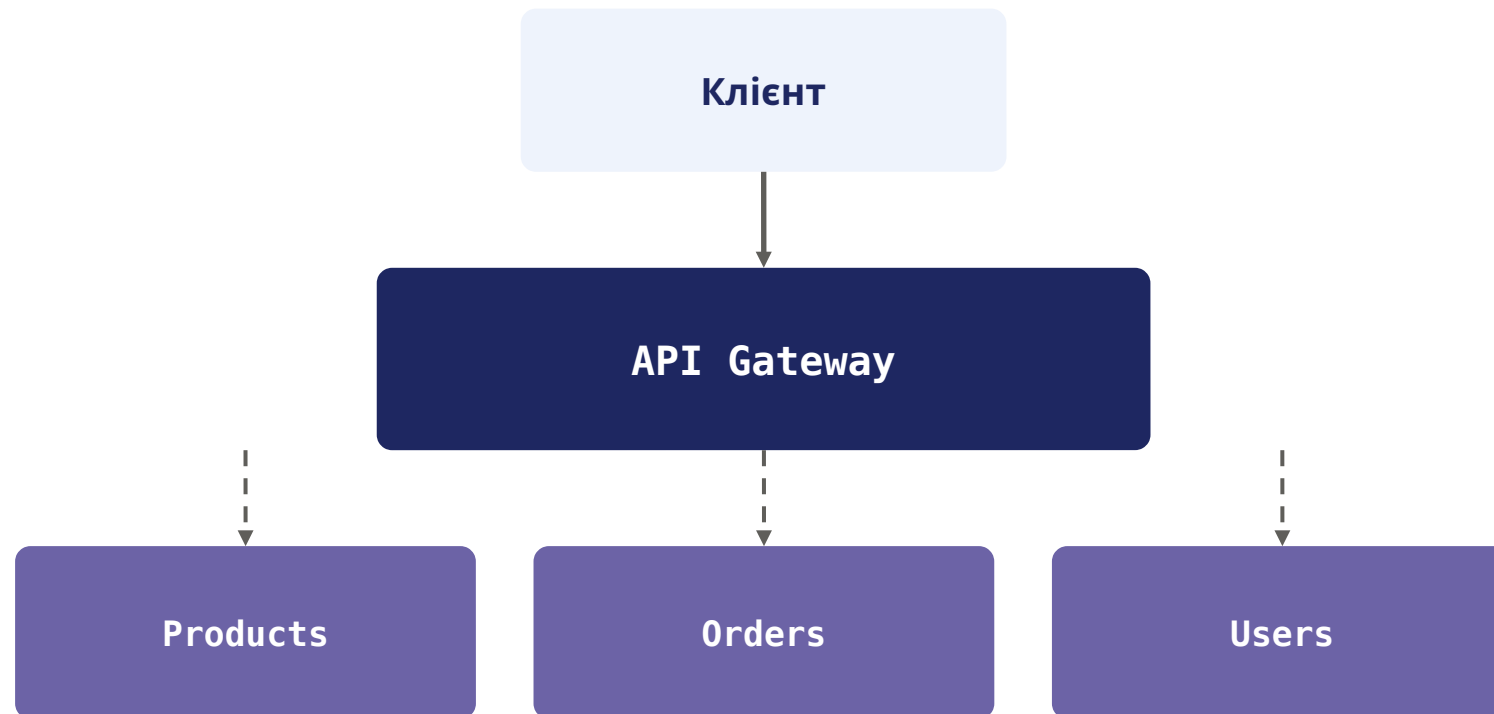
06

**Балансування
навантаження**

07

Як усе працює разом

Єдина точка входу для клієнтів



Клієнт знає лише про Gateway — внутрішня структура сервісів прихована й може змінюватись без впливу на клієнта.

Основні функції

Функція	Опис
Маршрутизація	/api/products → Products, /api/orders → Orders
Автентифікація	Перевіряє токен один раз, не дублюючи в кожному сервісі
Агрегація відповідей	Об'єднує дані з кількох сервісів в одну відповідь
Rate limiting	Обмежує кількість запитів від одного клієнта
Логування	Централізовано фіксує всі запити до системи

Як сервіси знаходять один одного

- 1 Products Service запускається → реєструється в реєстрі
- 2 Orders хоче звернутись до Products → запитує реєстр "де зараз products-service?"
- 3 Отримує актуальну адресу (може бути кілька)
- 4 Products падає → реєстр перестає отримувати heartbeat → видаляє адресу

Готові інструменти

Інструмент	Особливості
Consul	Реєстр сервісів із вбудованою перевіркою здоров'я
etcd	Розподілене сховище ключ-значення, часто в Kubernetes
Kubernetes DNS	Вбудований — сервіси звертаються за іменем

У Kubernetes достатньо звернутись за логічним іменем (`http://products-service`) — платформа сама направить запит на живу копію.

Централізоване сховище конфігурації

```
class ConfigClient {  
    public function __construct(private string $configServiceUrl) {}  
  
    public function get(string $key): mixed {  
        $response = file_get_contents("${this->configServiceUrl}/config/{$key}");  
        return json_decode($response, true)['value'];  
    }  
}  
  
$config = new ConfigClient('http://config-service');  
$dbHost = $config->get('database.host');
```

Секрети (паролі, API-ключі) — у спеціалізовані сховища (Vault), не в звичайну конфігурацію.

Рівні середовища

Середовище	Приклад
Development	Локальна база, вимкнене зовнішнє логування
Staging	Копія бойових даних, увімкнені всі логи
Production	Реальна база, суворі ліміти, моніторинг

Чотири рівні кешування

Рівень	Приклад
На клієнті	Браузер кешує CSS/зображення за Cache-Control
На API Gateway	Кешує відповідь GET /products на кілька секунд
Усередині сервіса	Redis-кеш результату запиту до БД
Бази даних	СУБД кешує часто використовувані сторінки в пам'яті

Read-through кеш із Redis

```
function getProduct(int $id, Redis $redis, PDO $pdo): array {  
    $cacheKey = "product:{$id}";  
  
    $cached = $redis->get($cacheKey);  
    if ($cached !== false) {  
        return json_decode($cached, true);    // швидко: дані в пам'яті  
    }  
  
    $stmt = $pdo->prepare('SELECT * FROM products WHERE id = :id');  
    $stmt->execute(['id' => $id]);  
    $product = $stmt->fetch(PDO::FETCH_ASSOC);  
  
    $redis->setex($cacheKey, 300, json_encode($product));  
    return $product;  
}
```

Найскладніше: інвалідація кешу

```
function updateProductPrice(int $id, float $newPrice, Redis $redis, PDO $pdo) {  
    $stmt = $pdo->prepare('UPDATE products SET price = :price WHERE id = :id');  
    $stmt->execute(['price' => $newPrice, 'id' => $id]);  
    $redis->del("product:{$id}");    // видалити застарілий кеш  
}
```

*"У комп'ютерних науках є лише дві по-справжньому складні речі: іменування змінних та інвалідація кешу."
Занадто довгий термін дії кешу — застарілі дані; занадто короткий — немає користі.*

Горизонтальне vs вертикальне

Вертикальне

Збільшити потужність ОДНОГО сервера (більше CPU, RAM)

Обмеження: фізична межа заліза; єдина точка відмови

Горизонтальне

Запустити КІЛЬКА копій сервіса на різних серверах

Потребує балансування навантаження й stateless-сервісів

Мікросервіси створені для горизонтального масштабування — кожен незалежний, stateless-процес.

Алгоритми розподілу запитів

Алгоритм	Принцип
Round Robin	По черзі: 1-й, 2-й, 3-й, знову 1-й...
Least Connections	До екземпляра з найменшою кількістю з'єднань
IP Hash	Клієнт завжди потрапляє на той самий екземпляр
Weighted Round Robin	Round Robin з урахуванням "ваги" екземпляра

NGINX як балансувальник

```
upstream products_service {  
    server 10.0.1.15:8080;  
    server 10.0.1.16:8080;  
    server 10.0.1.17:8080;  
}  
  
server {  
    location /api/products {  
        proxy_pass http://products_service;  
    }  
}
```

IP Hash дає "липку" сесію, але кращою практикою вважається робити сервіс повністю stateless.

ЖИТТЄВИЙ ЦИКЛ одного запиту

- 1 Клієнт → GET /api/products/42 на API Gateway
- 2 Gateway перевіряє токен автентифікації
- 3 Gateway питає Service Discovery: де живі копії Products?
- 4 Балансувальник обирає один із доступних екземплярів
- 5 Екземпляр перевіряє Redis-кеш, при потребі йде в БД
- 6 Відповідь повертається клієнту через Gateway

Підсумок теми 14

- API Gateway ховає внутрішню структуру системи від клієнта
- Service Discovery дозволяє сервісам знаходити одне одного динамічно
- Централізована конфігурація замінює розкидані по серверах .env файли
- Кешування знижує навантаження, але інвалідація — найскладніша частина
- Мікросервіси масштабують горизонтально: багато stateless-копій
- Жоден компонент не обов'язковий одразу — додавайте за реальної потреби (MonolithFirst)