



ТЕМА 15

# Контейнеризація

*Docker та Docker Compose для PHP-мікросервісів*

---

Бекенд-розробка, сервіси та мікросервіси

Комп'ютерна інженерія · 3 курс · Лекція 15 з 18

# 7 питань сьогоднішньої теми

**01**

Навіщо потрібна  
контейнеризація

**02**

Образи, контейнери,  
Dockerfile

**03**

Dockerfile для PHP-  
застосунку

**04**

Основні команди Docker

**05**

Навіщо Docker Compose

**06**

docker-compose.yml: PHP  
+ MySQL + Redis

**07**

Volumes, мережі, змінні  
середовища

# Віртуальна машина vs контейнер

## Віртуальна машина

Застосунок

Гостьова ОС (повна)

Гіпервізор

Апаратне забезпечення

## Контейнер

Застосунок

Docker Engine

Ядро ОС хоста (спільне)

Апаратне забезпечення

# Основні поняття

| Поняття               | Означення                                                   |
|-----------------------|-------------------------------------------------------------|
| Образ (image)         | Незмінний шаблон файлової системи для створення контейнерів |
| Контейнер (container) | Запущений (працюючий) екземпляр образу                      |
| Dockerfile            | Інструкція, що описує, як побудувати образ крок за кроком   |
| Registry              | Сховище готових образів (напр. Docker Hub)                  |

*Образ — це як клас (тема 4), а контейнер — як об'єкт (new). З одного образу можна запустити скільки завгодно контейнерів.*

## Приклад Dockerfile

```
FROM php:8.3-apache

# Встановлення розширень PHP
RUN docker-php-ext-install pdo pdo_mysql

# Копіювання коду застосунку в образ
COPY . /var/www/html/

WORKDIR /var/www/html
EXPOSE 80
```

*Використання готового офіційного образу `php:8.3-apache` — не потрібно встановлювати ОС і компілювати PHP з нуля.*

# Основні інструкції

| Інструкція | Призначення                                    |
|------------|------------------------------------------------|
| FROM       | Базовий образ для побудови нового              |
| RUN        | Виконує команду під час побудови образу        |
| COPY       | Копіює файли з хоста всередину образу          |
| WORKDIR    | Робоча тека для наступних команд               |
| EXPOSE     | Порт, який застосунок слухає в контейнері      |
| CMD        | Команда, що виконується при запуску контейнера |

## Побудова та запуск

| Команда                                             | Призначення                          |
|-----------------------------------------------------|--------------------------------------|
| <code>docker build -t myapp .</code>                | Побудувати образ з Dockerfile        |
| <code>docker run -p 8080:80 myapp</code>            | Запустити контейнер, прокинувши порт |
| <code>docker ps</code>                              | Список запущених контейнерів         |
| <code>docker logs &lt;container&gt;</code>          | Переглянути логи контейнера          |
| <code>docker exec -it &lt;container&gt; bash</code> | Командний рядок усередині контейнера |

## Прокидання портів

```
$ docker build -t php-shop-app .  
$ docker run -p 8080:80 php-shop-app  
  
# http://localhost:8080
```

**-p 8080:80** — перше число: порт хоста, друге: порт усередині контейнера (той самий, що EXPOSE).  
Дозволяє запускати кілька контейнерів з однаковим внутрішнім портом.

## Без Compose vs з Compose

| Без Docker Compose                                             | З Docker Compose                                               |
|----------------------------------------------------------------|----------------------------------------------------------------|
| Кожен контейнер — окрема довга команда <code>docker run</code> | Уся конфігурація в одному <code>docker-compose.yml</code>      |
| Мережу між контейнерами треба налаштовувати вручну             | Сервіси автоматично бачать одне одного за іменем               |
| Важко відтворити той самий стек на іншій машині                | <code>docker compose up</code> піднімає стек ідентично будь-де |

## Повний стек в одному файлі

```
services:
  app:
    build: .
    ports: ["8080:80"]
    environment:
      - DB_HOST=db
      - REDIS_HOST=cache
    depends_on: [db, cache]

  db:
    image: mysql:8.0
    environment:
      - MYSQL_ROOT_PASSWORD=secret
    volumes:
      - db_data:/var/lib/mysql

  cache:
    image: redis:7-alpine

volumes:
  db_data:
```

## Керування стеком

```
$ docker compose up -d      # підняти стек у фоні  
$ docker compose logs app   # логи конкретного сервісу  
$ docker compose down       # зупинити й видалити все
```

PHP-застосунок звертається до бази даних за іменем db, а не IP-адресою — **прямий практичний приклад Service Discovery (тема 14).**

## Volumes: дані поза контейнером

Контейнери — одноразові (ephemeral): якщо контейнер видалити, дані всередині зникають. Для бази даних це неприйнятно.

```
db_data:/var/lib/mysql
```

файли бази даних фізично зберігаються на диску хоста — контейнер можна видалити й перестворити, дані залишаться.

## Мережі: сервіси за іменем



Docker Compose автоматично створює спільну мережу для всіх сервісів одного файлу — app звертається до db за іменем, не за IP-адресою. Готова "з коробки" реалізація Service Discovery для локальної розробки.

## Змінні середовища в PHP-кодi

```
$dbHost = getenv('DB_HOST') ?: 'localhost';  
$dbName = getenv('DB_NAME') ?: 'shop';  
  
$pdo = new PDO("mysql:host=$dbHost;dbname=$dbName", 'root', 'secret');
```

**?:** (елвіс-оператор) дає запасне значення localhost, якщо DB\_HOST не встановлено — той самий код працює і в Docker Compose, і локально без контейнерів.

# Підсумок теми 15

- Контейнер вирішує проблему "у мене працює", пакуючи застосунок з усім оточенням
- Контейнер легший за VM — спільне ядро ОС хоста замість повної гостьової системи
- Dockerfile описує, як побудувати образ; образ — шаблон, контейнер — його запущений екземпляр
- Docker Compose описує весь стек сервісів в одному декларативному файлі
- Сервіси в Compose бачать одне одного за іменем — це Service Discovery "з коробки"
- Volumes зберігають дані бази поза життєвим циклом контейнера