

Дисципліна

«Бекенд-розробка, сервіси та мікросервіси»

КОНСПЕКТ ЛЕКЦІЇ. ТЕМА 15

Контейнеризація: Docker та Docker Compose для PHP-мікросервісів

Спеціальність: Комп'ютерна інженерія, 3 курс

Обсяг: 1 год 20 хв (лекція 15 з 18)

План лекції

1. Навіщо потрібна контейнеризація
2. Основні поняття Docker: образи, контейнери, Dockerfile
3. Написання Dockerfile для PHP-застосунку
4. Основні команди Docker
5. Docker Compose: навіщо оркеструвати кілька контейнерів
6. docker-compose.yml для стеку PHP + MySQL + Redis
7. Volumes, мережі та змінні середовища на практиці

Мета лекції: зрозуміти, яку проблему вирішує контейнеризація, навчитися описувати PHP-застосунок у вигляді Docker-образу та запускати повний стек сервісів (застосунок, база даних, кеш) єдиною командою через Docker Compose.

1. Навіщо потрібна контейнеризація

Класична проблема, знайома кожному розробнику: код працює на комп'ютері розробника, але падає на сервері колеги чи в production — через іншу версію PHP, відсутнє розширення, іншу конфігурацію бази даних. Цю ситуацію жартома називають «у мене працює» (it works on my machine).

Контейнеризація вирішує цю проблему, пакуючи застосунок разом з усім необхідним оточенням (конкретна версія PHP, потрібні розширення, системні бібліотеки) в один переносний артефакт — контейнер, який гарантовано поводитиметься однаково на будь-якій машині, де встановлено Docker: на ноутбучі розробника, на сервері тестування чи на бойовому сервері.

Підхід	Проблема
Ручне встановлення на кожному сервері	Версії PHP/бібліотек можуть відрізнятися; налаштування губляться в документації
Віртуальні машини (VM)	Кожна VM містить повну копію операційної системи — важка, повільно стартує
Контейнери (Docker)	Спільне ядро ОС хоста; контейнер містить лише застосунок і його залежності — легкий і швидкий

Контейнер — не те саме, що віртуальна машина: VM віртуалізує апаратне забезпечення й запускає повноцінну гостьову ОС (важко, повільно), тоді як контейнер використовує ядро операційної системи хоста напям, ізолюючи лише процеси й файлову систему — це на порядки легше й дозволяє запускати десятки контейнерів там, де помістилося б лише кілька VM.

2. Основні поняття Docker: образи, контейнери, Dockerfile

Поняття	Означення
Образ (image)	Незмінний «знімок» файлової системи й налаштувань — шаблон для створення контейнерів
Контейнер (container)	Запущений (працюючий) екземпляр образу — те, що реально виконується
Dockerfile	Текстовий файл-інструкція, що описує, як побудувати образ крок за кроком
Реєстр образів (registry)	Сховище готових образів, звідки їх можна завантажити (напр. Docker Hub)

Аналогія, яка часто допомагає: образ — це як клас у програмуванні (тема 4), а контейнер — це об'єкт, створений з цього класу (new). З одного образу можна запустити скільки завгодно незалежних контейнерів — так само, як з одного класу можна створити багато екземплярів.

3. Написання Dockerfile для PHP-застосунку

Dockerfile складається з послідовності інструкцій, кожна з яких додає новий шар до образу. Розглянемо базовий Dockerfile для PHP-застосунку з нашого курсу (структура проєкту з теми 9).

```
# Dockerfile
FROM php:8.3-apache

# Встановлення розширень PHP, потрібних застосунку (напр. для роботи з MySQL)
RUN docker-php-ext-install pdo pdo_mysql

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

# Робоча тека всередині контейнера
WORKDIR /var/www/html

# Порт, який застосунок слухає всередині контейнера
EXPOSE 80
```

Інструкція	Призначення
FROM	Базовий образ, від якого будується новий (тут — офіційний образ PHP з Apache)
RUN	Виконує команду під час побудови образу (встановлення пакетів, розширень)
COPY	Копіює файли з машини розробника всередину образу
WORKDIR	Встановлює робочу теку для наступних інструкцій і команд у контейнері
EXPOSE	Документує, який порт застосунок використовує всередині контейнера
CMD	Команда, що виконується при запуску контейнера (якщо не задана — використовується команда з базового образу)

FROM php:8.3-apache використовує офіційний, готовий образ PHP з підтримки спільноти — не потрібно самостійно встановлювати операційну систему й компілювати PHP з нуля. Вибір конкретної версії (8.3, а не просто php) — важлива практика: це гарантує, що образ не «зміниться сам по собі» при новому запуску, коли вийде PHP 8.4.

4. Основні команди Docker

Команда	Призначення
<code>docker build -t myapp .</code>	Побудувати образ з Dockerfile у поточній теці, назвавши його myapp
<code>docker run -p 8080:80 myapp</code>	Запустити контейнер, прокинувши порт 8080 хоста на порт 80 контейнера
<code>docker ps</code>	Показати список запущених контейнерів
<code>docker logs <container></code>	Переглянути вивід (логи) контейнера
<code>docker exec -it <container> bash</code>	Відкрити командний рядок усередині запущеного контейнера
<code>docker stop <container></code>	Зупинити контейнер

```
$ docker build -t php-shop-app .
$ docker run -p 8080:80 php-shop-app

# Застосунок тепер доступний у браузері за адресою:
# http://localhost:8080
```

Прокидання портів (-p 8080:80) — ключовий механізм доступу ззовні до контейнера: перше число — порт на машині хоста, друге — порт усередині контейнера (той самий, що вказаний у EXPOSE). Це дозволяє запускати кілька контейнерів з однаковим внутрішнім портом 80, прокидаючи кожен на свій зовнішній порт хоста (8080, 8081, 8082...).

5. Docker Compose: навіщо оркеструвати кілька контейнерів

Реальний застосунок рідко складається з одного контейнера: потрібна база даних (тема 7), можливо — Redis для кешування (тема 14), окремий контейнер для самого PHP-застосунку. Запускати кожен контейнер вручну окремою командою `docker run` незручно й важко відтворити point-in-time — Docker Compose дозволяє описати весь стек сервісів в одному декларативному файлі та керувати ним як єдиним цілим.

Без Docker Compose	З Docker Compose
Кожен контейнер запускається окремою командою <code>docker run</code> з довгим списком параметрів	Уся конфігурація описана один раз у файлі <code>docker-compose.yml</code>
Мережу між контейнерами потрібно налаштовувати вручну	Всі сервіси автоматично потрапляють в одну мережу й бачать одне одного за іменем
Важко відтворити той самий стек на іншій машині	<code>docker compose up</code> піднімає весь стек ідентично на будь-якій машині

6. docker-compose.yml для стеку PHP + MySQL + Redis

Розглянемо повний приклад docker-compose.yml для типового стеку нашого курсового застосунку: PHP-застосунок, база даних MySQL і Redis для кешування (тема 14).

```
# docker-compose.yml
services:
  app:
    build: .
    ports:
      - "8080:80"
    environment:
      - DB_HOST=db
      - DB_NAME=shop
      - REDIS_HOST=cache
    depends_on:
      - db
      - cache

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

  cache:
    image: redis:7-alpine

volumes:
  db_data:
```

Один-єдиний запуск команди `docker compose up` піднімає всі три сервіси одночасно, у правильному порядку (завдяки `depends_on`), у спільній мережі, де кожен сервіс звертається до інших за іменем (наприклад, PHP-застосунок звертається до бази даних просто за іменем `db`, а не за IP-адресою) — це прямий практичний приклад Service Discovery, розглянутого концептуально в темі 14.

```
$ docker compose up -d      # підняти весь стек у фоновому режимі
$ docker compose logs app   # переглянути логи конкретного сервісу
$ docker compose down       # зупинити й видалити всі контейнери стеку
```

Змінні середовища (*environment*) у прикладі вище — це той самий принцип централізованої конфігурації мікросервісів (тема 14), лише в найпростішій формі: замість звернення до окремого сервісу конфігурації значення передаються напряму при запуску контейнера. Для *production*-середовищ значення секретів (паролі) зазвичай виносять у окремий *.env* файл, який не потрапляє в систему контролю версій.

7. Volumes, мережі та змінні середовища на практиці

7.1. Volumes: збереження даних поза контейнером

Контейнери за задумом одноразові (ephemeral): якщо контейнер видалити, усі дані, записані всередині нього, зникають. Для бази даних це неприйнятно — дані повинні зберігатися між перезапусками. Volume (том) — механізм Docker для збереження даних на диску хоста, незалежно від життєвого циклу контейнера.

У прикладі `docker-compose.yml` вище рядок `db_data:/var/lib/mysql` означає: тека `/var/lib/mysql` усередині контейнера (де MySQL зберігає файли бази даних) фізично зберігається в іменованому томі `db_data` на диску хоста. Якщо контейнер `db` буде видалено й створено заново, дані бази залишаться недоторканими.

7.2. Мережі: як сервіси знаходять один одного

Docker Compose автоматично створює окрему віртуальну мережу для всіх сервісів, описаних в одному файлі, і реєструє кожен сервіс під його іменем із `docker-compose.yml` — тому PHP-застосунок може підключатися до бази даних за `host=db`, а не за конкретною (і змінною) IP-адресою контейнера. Це мініатюрна, готова «з коробки» реалізація Service Discovery (тема 14) для локальної розробки.

7.3. Змінні середовища у PHP-коді

Значення, передані через `environment` у `docker-compose.yml`, доступні всередині PHP-коду через функцію `getenv()` — це і є той механізм, яким застосунок «дізнається» адресу бази даних, не маючи її захардкодженою в самому коді.

```
<?php
$dbHost = getenv('DB_HOST') ?: 'localhost';
$dbName = getenv('DB_NAME') ?: 'shop';

$pdo = new PDO("mysql:host=$dbHost;dbname=$dbName;charset=utf8mb4",
    'root', 'secret');
```

Оператор `?:` (елвіс-оператор, коротка форма тернарного) забезпечує запасне значення `localhost`, якщо змінна середовища `DB_HOST` не встановлена — це дозволяє тому самому коду коректно працювати як усередині Docker Compose (де `DB_HOST=db`), так і при локальному запуску без контейнерів (де використовується значення за замовчуванням).

Питання для самоперевірки

- У чому суть проблеми «у мене працює» і як контейнеризація її вирішує?
- Чим контейнер принципово відрізняється від віртуальної машини?
- Проведіть аналогію між образом/контейнером Docker і класом/об'єктом у програмуванні.
- Що означають числа в команді `docker run -p 8080:80` — яке з них відповідає хосту, а яке контейнеру?
- Чому Docker Compose зручніший за запуск кожного контейнера окремою командою `docker run`?
- Як сервіси, описані в одному `docker-compose.yml`, знаходять один одного за іменем, а не за IP-адресою?
- Для чого потрібні `volumes` і що станеться з даними бази без них?
- Як значення, задані через `environment` у `docker-compose.yml`, потрапляють у PHP-код застосунку?

Короткий глосарій

Термін	Означення
Образ (image)	Незмінний шаблон файлової системи й налаштувань для створення контейнерів
Контейнер (container)	Запущений ізольований екземпляр образу
Dockerfile	Інструкція для побудови образу крок за кроком
Docker Compose	Інструмент для опису й запуску кількох контейнерів як єдиного стеку
Volume (том)	Механізм збереження даних контейнера на диску хоста поза його життєвим циклом
Прокидання порту (port mapping)	Зв'язування порту хоста з портом усередині контейнера
getenv()	Функція PHP для читання значення змінної середовища