

Основна термінологія Docker

Практичний довідник для тестиувальників ПЗ та QA-інженерів, які хочуть швидко освоїти ключові поняття Docker – від Dockerfile до Docker Compose. Цей документ веде вас логічним ланцюжком: **опис → збірка → запуск → реєстр → оркестрація**, допомагаючи зрозуміти, як кожна концепція пов'язана з іншою та як їх застосовувати у реальних тестових проектах.

Вступ

ОГЛЯД ДОКУМЕНТА

Docker став стандартом де-факто для ізоляції середовищ у сучасному розробленні та тестуванні. Для QA-інженера розуміння базових понять Docker – це не просто корисна навичка, а необхідна умова ефективної роботи в CI/CD-пайплайнах, де автотести виконуються у контейнерах. Цей документ охоплює п'ять ключових концепцій та їхній взаємозв'язок.

01

Dockerfile

Текстовий опис того, як зібрати середовище

02

Образ (Image)

Read-only шаблон, результат збірки

03

Контейнер

Запущений екземпляр образу, де виконуються тести

04

Docker Hub

Публічний реєстр для завантаження та публікації образів

05

Docker Compose

Оркестрація кількох контейнерів одночасно

Кожна секція цього документа детально розкриває одне поняття з прикладами команд, типовими помилками та порадами для тестувальників. Рекомендуємо читати послідовно, адже кожна концепція спирається на попередню.

Як пов'язані поняття

Щоб ефективно працювати з Docker, важливо розуміти не окремі команди, а логічний ланцюжок між усіма концепціями. Нижче показано, як кожне поняття витікає з попереднього і веде до наступного – від простого опису до повноцінного тестового середовища.



Цей ланцюжок відображає реальний робочий процес QA-інженера: ви описуєте середовище у Dockerfile, збираєте образ, запускаєте контейнер для виконання тестів, публікуєте образ на Docker Hub для спільноговикористання, а Docker Compose дозволяє піднімати повноцінне тестове оточення – застосунок, базу даних і тести – однією командою.

Dockerfile

КРОК 1 ІЗ 5

Що це таке і навіщо

Dockerfile – це звичайний текстовий файл без розширення, у якому по рядках записані інструкції для збірки **образу** Docker. Кожен рядок – це команда, яку Docker виконує по черзі під час `docker build`. Dockerfile не запускає контейнер сам по собі; він лише описує, як створити образ, з якого потім будуть створюватися контейнери.

- 💡 **Навіщо тестувальнику:** один Dockerfile дозволяє зібрати однакове середовище на будь-якій машині та в CI – однакові версії ОС, Node, залежностей. Це вирішує проблему «у мене працює, на CI – ні».

Основні інструкції

FROM	Базовий образ, від якого починається збірка (наприклад, <code>node:20</code>)
ENV	Встановлення змінних оточення всередині образу
RUN	Виконання команди під час збірки; кожен RUN створює новий шар образу
COPY / ADD	Копіювання файлів з хоста в образ; для початку краще COPY
WORKDIR	Встановлення робочої директорії для наступних інструкцій
EXPOSE	Документує порт контейнера (не відкриває порт на хості)
CMD / ENTRYPOINT	Команда при запуску контейнера; CMD можна перевизначити параметрами <code>docker run</code>

Приклад Dockerfile

Образ на базі Ubuntu для Node.js-додатку:

```
# Базовий образ
FROM ubuntu:20.04

# Змінні оточення
ENV APP_HOME /app

# Створення робочої директорії
RUN mkdir -p app

# Копіювання файлів проекту
COPY .

# Очищення зайвого
RUN rm -f package-lock.json \
&& rm -rf node_modules

# Робоча директорія
WORKDIR /app

# Встановлення залежностей
RUN npm install

# Порт додатку
EXPOSE 8080

# Команда запуску
CMD ["npm", "start"]
```

Важливо: у `ubuntu:20.04` немає Node.js за замовчуванням. У реальному проекті використовуйте `FROM node:20`.

Docker Image (образ)

КРОК 2 ІЗ 5

Образ (image) – це результат виконання інструкцій з Dockerfile: «знімок» файлової системи, залежностей і конфігурації.

Образ **тільки для читання**: його не змінюють під час роботи, з нього лише **створюють контейнери**. Один і той самий образ можна використати для запуску багатьох незалежних контейнерів.

Локальна збірка

Зберіть образ самостійно з Dockerfile командою `docker build`. Образ зберігається локально та доступний одразу.

Завантаження з реєстру

Завантажте готовий образ з Docker Hub командою `docker pull` `postgres:15`. Тисячі офіційних образів доступні безкоштовно.

Публікація образу

Після реєстрації на Docker Hub надішліть свій образ командою `docker push`, щоб ним могли користуватися колеги або CI-сервер.

Де зберігаються образи

Образи зберігаються локально після `docker build` або `docker pull`. Переглянути список локальних образів можна командою `docker images`. Їх також можна публікувати у **реєстр** (Docker Hub, GitLab Container Registry тощо), щоб інші могли їх завантажити і використовувати в будь-якому середовищі.

Для тестувальника

Образ – це готове середовище (ОС + залежності + код), в якому **однаково виконуються тести** на різних машинах і в CI. Замість налаштування середовища вручну на кожній машині – один образ, який гарантує ідентичну поведінку скрізь.

Команда docker build

КОМАНДА

```
docker build [OPTIONS] PATH | URL | -
```

Призначення: зібрати образ з Dockerfile. Це центральна команда, з якою починається будь-який процес контейнеризації. Docker читає Dockerfile крок за кроком, виконує кожну інструкцію і формує фінальний образ із шарів.

PATH – шлях до каталогу

Вказує, де знаходиться Dockerfile (за замовчуванням – поточний каталог .). Усі файли в цьому каталозі утворюють **контекст збірки**. Інструкції типу COPY можуть брати файли лише з контексту; файли поза ним недоступні.

URL – Git-репозиторій або архів

Docker завантажить контекст звідти і виконає збірку. Формат Git:
`https://github.com/user/repo.git#branch:subfolder`. Зручно для збірки безпосередньо з репозиторією без клонування.

Результат – новий образ

Після успішної збірки з'являється локальний образ з ім'ям і тегом, наприклад `app:latest`. Його можна одразу запустити або запушити на Docker Hub.

- ❑ **⚠ Поширення помилка:** виконувати docker build з каталогу, де дуже багато зайвих файлів. Вони всі потрапляють у контекст і можуть сповільнити збірку або потрапити в образ через COPY .. Краще тримати Dockerfile у окремому каталозі або використовувати `.dockerignore`, щоб виключити `node_modules`, `.git` та інші непотрібні файли.

Команда docker images

КОМАНДА

```
docker images [REPOSITORY[:TAG]]
```

Призначення: показати список образів на вашій машині. Це перша команда, яку варто виконати після збірки або завантаження образу – щоб переконатися, що образ існує і має правильний тег.

Колонки у виводі

REPOSITORY	Ім'я образу (репозиторій)
TAG	Тег версії (наприклад, latest, 20, v1.0). Разом REPOSITORY:TAG однозначно ідентифікують образ
IMAGE ID	Короткий унікальний ідентифікатор образу
CREATED	Коли образ було створено
SIZE	Розмір образу на диску

Приклади використання

```
docker images
```

Показує всі локальні образи незалежно від репозиторію та тегу

```
docker images java
```

Показує всі образи з репозиторію java (усі теги)

```
docker images java:8
```

Показує лише образ java з тегом 8. Якщо такого немає – список буде порожнім

Docker Container (контейнер)

КРОК 3 ІЗ 5

Контейнер – це запущений екземпляр образу. У ньому виконується ваш код і тести. Контейнер ізольований від хост-системи: має власну файлову систему, мережу та процеси. Всередині контейнера програма «бачить» лише те, що є в образі та що змонтовано ззовні (volumes, змінні оточення).

Образ vs Контейнер

Образ – шаблон (read-only). Він незмінний і може породжувати багато контейнерів.

Контейнер – «живий» екземпляр: його можна запускати, зупиняти, видаляти. Зміни всередині контейнера **не зберігаються** в образі, якщо не використовувати docker commit (що рідко потрібно).

Кілька контейнерів можна запустити з одного образу – вони будуть незалежними або пов'язаними через мережу/volumes, як у Docker Compose.

Для тестувальника

Контейнер – це те середовище, в якому реально виконуються автотести (наприклад, Cypress або Playwright). Ви запускаєте контейнер з потрібним образом і передаєте туди код тестів або монтуєте його з хоста через volume.

- Ізоляція гарантує, що тести не залежать від стану хост-машини
- Можна паралельно запускати кілька контейнерів з тестами
- Легко відтворити будь-яке середовище з точним набором залежностей
- Після завершення тестів контейнер видаляється, не залишаючи слідів

Docker Hub

КРОК 4 ІЗ 5

Docker Hub – це публічний реєстр образів Docker за адресою hub.docker.com. Це центральне місце, де зберігаються офіційні образи популярних інструментів і де команди можуть публікувати власні образи для спільноговикористання. Для QA-інженера Docker Hub – це насамперед джерело готових образів для баз даних, браузерів, тестових фреймворків та іншого.



Пошук образів

Шукайте готові образи для баз даних, мов програмування, інструментів – все доступно безкоштовно.



Завантаження

Завантажуйте образи командою `docker pull`. Наприклад:
`docker pull postgres:15`



Публікація

Після `docker login` надсилаєте свої образи командою `docker push` для спільногодоступу.



Реєстрація

Безкоштовний акаунт знімає обмеження на кількість `pull`-запитів і відкриває приватні репозиторії.

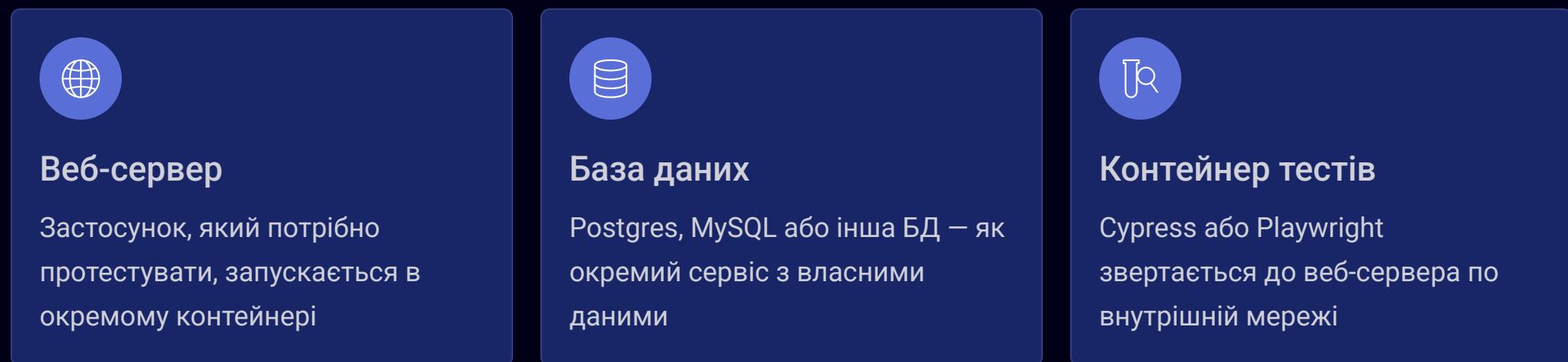
- ☐ **⚠ Важливо:** без акаунта в деяких випадках неможливо або обмежено завантажувати образи з Docker Hub через `rate limit`. Для навчання та роботи з офіційними образами варто створити безкоштовний акаунт.
Інструкція: [Docker ID – документація](#).

Docker Compose

КРОК 5 ІЗ 5

Docker Compose – інструмент для опису та запуску **багатоконтейнерних** застосунків. Конфігурація пишеться у файлі **YAML** (зазвичай `docker-compose.yml`). Однією командою (`docker compose up`) можна підняти всі сервіси, мережі та томи, описані у файлі. Compose підтримують більшість CI-провайдерів, що робить його ідеальним інструментом для тестових середовищ.

Навіщо тестиwalнику Docker Compose



Приклад `docker-compose.yml`

```
version: '3'  
services:  
  web:  
    image: apache  
    build: ./webapp  
    container_name: apache  
    restart: always  
    ports:  
      - "8080:80"  
  
  e2e:  
    image: cypress  
    build: ./e2e  
    container_name: cypress  
    depends_on:  
      - web  
    environment:  
      - CYPRESS_baseUrl=http://web  
    command: npx cypress run  
    volumes:  
      - ./e2e/cypress:/app/cypress  
      - ./e2e/cypress.config.js:/app/cypress.config.js
```

Пояснення прикладу

- **web** – сервіс на базі Apache, слухає порт 8080 на хості (проброс з порту 80 контейнера)
- **e2e** – сервіс з Cypress, залежить від `web` через `depends_on`
- Cypress звертається до веб-сервера по імені `web` у внутрішній мережі
- Каталог з тестами монтується з хоста через `volumes` – не потрібно перезбирати образ при зміні тестів

Порада: у Compose V2 поле `version` більше не потрібне. Ключові елементи для розуміння: `services`, `ports`, `depends_on`, `volumes`, `environment`.

Висновки

Ви пройшли повний ланцюжок ключових понять Docker – від опису середовища до оркестрації повноцінного тестового стенду. Ось ключові тези для запам'ятовування та швидкого довідника у щоденній роботі QA-інженера.

Dockerfile → Образ → Контейнер

Dockerfile – текстовий опис збірки; образ – read-only шаблон, результат docker build; контейнер – запущений екземпляр образу, де виконується код і тести.

Docker Hub – реєстр образів

Центральне сховище офіційних і користувачьких образів. Для повноцінної роботи, особливо в CI, потрібен безкоштовний обліковий запис, щоб уникнути обмежень на pull-запити.

Ключові команди

docker build – збирає образ із Dockerfile та контексту; docker images – показує список локальних образів із тегами, ID та розміром.

Docker Compose – для e2e та інтеграційного тестування

Дозволяє описати та запустити кілька контейнерів разом (застосунок + тести + залежності) однією командою. Ідеальний інструмент для автоматизованих тестів у CI/CD.

5

КЛЮЧОВИХ ПОНЯТЬ

Dockerfile, Image, Container, Docker Hub, Compose

2

ОСНОВНІ КОМАНДИ

docker build та docker images

1

ФАЙЛ КОНФІГУРАЦІЇ

docker-compose.yml для всього оточення