

# Docker Overview

Практичний вступ до Docker для тестувальників програмного забезпечення та інженерів з CI/CD. Дізнайтеся, як контейнеризація змінює підхід до тестування, усуває проблему «у мене працює», та інтегрується з сучасними пайплайнами безперервної інтеграції.

# Вступ

## Тема курсу

У цьому матеріалі ми розглянемо Docker з точки зору тестувальника: що це таке, навіщо це потрібно, і як це допомагає у щоденній роботі з CI/CD.

- Що таке Docker і навіщо він тестувальнику
- Зв'язок з тестуванням і CI/CD
- Відмінності від віртуальних машин

## Навіщо це тестувальнику

Одна з найпоширеніших проблем у тестуванні – розбіжність середовищ. Тест проходить локально, але «падає» на CI-сервері, або навпаки. Docker вирішує цю проблему радикально.

- Запуск тестів в **однаковому середовищі** локально, на CI та у колег – без зміни коду тестів. Менше ситуацій «у мене працює, у пайплайні – ні».

## План

01

### Що таке Docker

Основні концепції та термінологія

02

### Ізольоване середовище

Як Docker забезпечує консистентність

03

### Навіщо для тестування

Практичні переваги для QA

04

### Docker vs VM

Порівняння підходів до ізоляції

05

### Провайдери

Екосистема інструментів

06

### Висновки

Ключові тезиси та рекомендації

# Що таке Docker (простими словами)

## Docker

**Docker** – це інструмент для створення, розгортання та запуску програм у **контейнерах**. Він дозволяє розробникам і тестувальникам «упаковувати» програму разом із усіма її залежностями в єдиний портативний юніт, який однаково працює будь-де.

Docker з'явився у 2013 році і швидко став стандартом у сучасній розробці та DevOps-практиках. Сьогодні його використовують мільйони розробників і тисячі компаній по всьому світу.

## Контейнер

**Контейнер** – це упаковане середовище, в якому є все необхідне для роботи програми: код, бібліотеки, залежності, налаштування.

Аналогія: як посилка з товаром і інструкцією – відкриваєте в будь-якому місці і отримуєте одинаковий результат.

## Чому це важливо для тестувальника

На одному комп'ютері може стояти інша версія Node.js, браузера або ОС, ніж на сервері CI або на машині розробника. Тоді тест «у мене працює, а в пайплайні – ні» (або навпаки). Docker дозволяє упаковувати тести разом із потрібним середовищем у контейнер і запускати їх однаково скрізь: на вашому ноутбуці, на сервері CI/CD, у хмарі.

### Ваш ноутбук

Локальний запуск тестів

### CI/CD сервер

Автоматичний запуск у пайплайні

### Хмара

Масштабований запуск у продакшні

# Docker як ізольоване середовище

Docker дає змогу в ізольованому «контейнері» мати все необхідне для роботи вашого застосунку та тестів. Програма всередині контейнера не залежить від того, що встановлено на хост-машині – вона бачить лише те, що підготовлено в образі. Це і є **консистентність середовища** – однакові умови виконання незалежно від машини.



## Потрібна ОС

Мінімальний набір файлів операційної системи, необхідний для запуску вашого застосунку – без зайвого.



## Версія рантайму

Node.js, Python, Java або будь-який інший рантайм – точно та версія, яка потрібна вашим тестам.



## Змінні оточення

Усі необхідні ENV-zmінні та конфігурації прописані в образі і відтворюються автоматично.



## Залежності

Усі бібліотеки та пакети встановлені у потрібних версіях – без конфліктів з іншими проектами.

- ☐ **Ключова ідея:** Контейнер ізоляє середовище виконання на рівні файлової системи та процесів. Хост-машина не «забруднює» середовище тестів, а зміни всередині контейнера не впливають на хост.

# Навіщо Docker саме для тестування

Docker вирішує одразу кілька ключових проблем, з якими стикаються тестувальники в реальних проектах. Нижче – дев'ять конкретних переваг, які безпосередньо впливають на якість і стабільність тестування.



## Консистентність середовища

Однакове середовище на вашому ПК, на CI-сервері та у колег. Менше ситуацій «у мене працює» через різні версії ОС, мови чи бібліотек.



## Ізоляція залежностей

Різні проекти з різними версіями Node, Python тощо можуть співіснувати на одній машині в окремих контейнерах без конфліктів.



## Швидке розгортання

Контейнери можна швидко створити і видалити. Зручно для автоматизованих тестів: підняли середовище → прогнали тести → знишили контейнер.



## Відтворюваність тестів

Той самий образ дає той самий набір залежностей і ОС. Результати тестів стають передбачуваними і легше відтворюються при багах.

## Мікросервіси

Кожен сервіс у своєму контейнері; тести можна запускати в середовищі, близькому до реального.

## CI/CD інтеграція

Jenkins, GitLab CI, GitHub Actions легко запускають тести в Docker-контейнерах як крок пайплайну.

## Економія ресурсів

Порівняно з VM контейнери легші і швидше стартують, бо використовують ядро ОС хоста.

## Мобільність

Один і той самий образ можна переносити між локальною машиною, CI і хмарою без змін.

## Версіонування

Образ можна зберігати під тегом (як код у репозиторії) і відновлювати потрібну конфігурацію при потребі.

# Docker і віртуальні машини (VM)

Віртуалізація – це спосіб поділити ресурси комп'ютера (CPU, пам'ять, диск, мережа) так, що вони виглядають як кілька окремих машин. Однак Docker і традиційні VM реалізують цю ідею принципово різними способами.

## Віртуальна машина (VM)

Віртуалізує **всю** машину: «залізо» + повноцінна ОС. Кожна VM містить власне ядро, драйвери та повноцінну операційну систему. Це важкий і повільний старт, але повна ізоляція.

- Гіпервізор емулює апаратне забезпечення
- Кожна VM – окрема ОС (гігабайти)
- Старт займає хвилини
- Повна ізоляція на рівні заліза

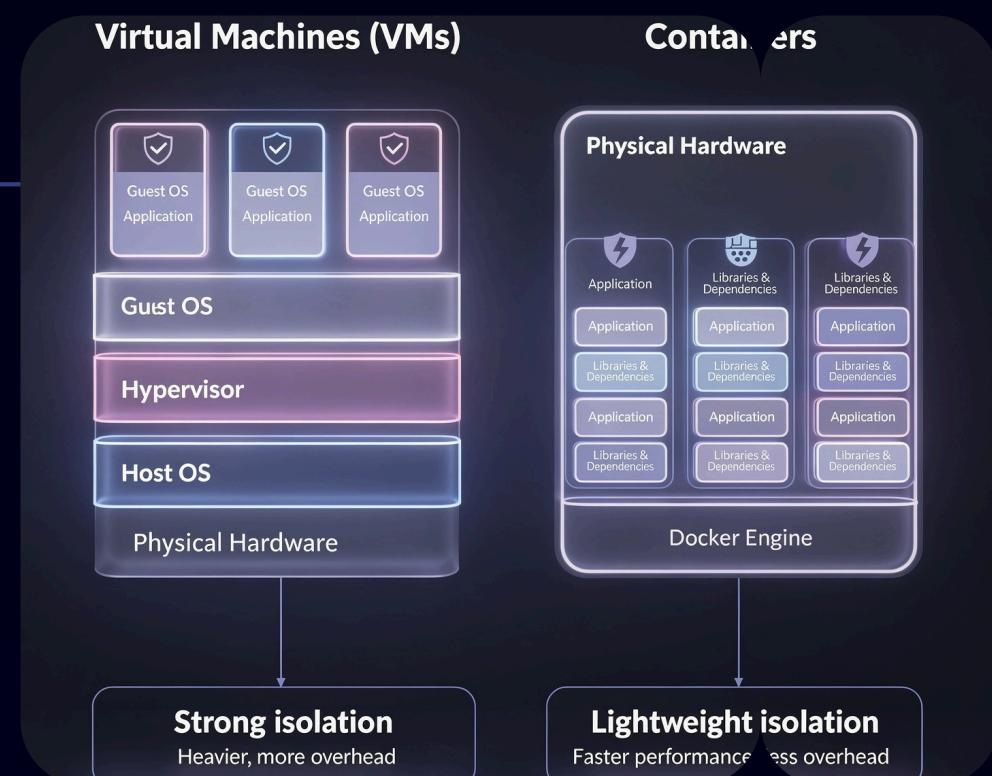
## Docker контейнер

Віртуалізує лише **програмний рівень** над ядром ОС хоста: файлова система, процеси, мережа ізольовані, але ядро одне. Тому контейнери легші і швидші.

- Спільне ядро ОС хоста
- Образ – мегабайти
- Старт займає секунди
- Ізоляція на рівні процесів

### Шлях VM

Hypervisor →  
Гостьова ОС →  
Додаток, повна  
ізоляція



### Шлях Контейнера

Docker Engine →  
Контейнер →  
Додаток, швидко  
та легковагово

- Коротко: VM = емуляція цілого комп'ютера; контейнер = ізольоване середовище для процесів на спільному ядрі ОС.

# Контейнери: плюси і мінуси

Контейнери – це легкі пакети з усім необхідним для запуску програми (системні бібліотеки, залежності, код).

Залежності всередині контейнера знаходяться вище рівня ядра ОС – це бібліотеки, рантайми, ваш код. Такий підхід забезпечує швидкість, але має свої обмеження, про які важливо знати.

## ✓ Плюси контейнерів

### → Швидкість

Контейнери швидко створюються, змінюються і перезапускаються. Старт займає секунди, що критично для автоматизованого тестування в CI/CD пайплайнах.

### → Готова екосистема

Публічні реєстри (наприклад, Docker Hub) містять готові образи баз даних, веб-серверів, брокерів повідомлень тощо, які можна швидко використовувати у тестах.

## ⚠ Мінуси контейнерів

### Спільний хост

Контейнери ділять ядро ОС і залізо. Теоретично вразливість в одному контейнері може вплинути на хост; на практиці це зменшують правильним налаштуванням і оновленнями.

### Безпека образів

Публічні образи можуть містити застарілі або вразливі пакети. Краще використовувати офіційні або перевірені образи і регулярно оновлювати їх.

# Віртуальні машини: плюси і мінуси

Віртуальна машина – це повна емуляція «заліза» (CPU, диск, мережа) і окремої ОС. Разом це дає повноцінний знімок обчислювальної системи, що забезпечує надійну ізоляцію, але за рахунок значних ресурсів.

## ✓ Плюси VM

### Сильна ізоляція

Одна VM не зачіпає інші на тому ж хості. Зламана або скомпрометована VM не виходить за межі своєї «машини», що забезпечує надійний захист.

### Гнучкість

Можна вручну встановлювати ПЗ, робити снепшоти стану і відновлюватися з них. Зручно для складних тестових сценаріїв з нестандартними конфігураціями.

## ⚠ Мінуси VM

### Повільний старт і зміни

Створення та зміна VM займає значно більше часу, ніж у контейнерів. Для швидких CI/CD пайплайнів це може бути критичним обмеженням.

### Великий об'єм

Образи VM часто займають багато гігабайт. Необхідно враховувати наявне місце на диску, особливо при зберіганні кількох образів одночасно.

# Провайдери контейнерів

Ринок контейнеризації пропонує кілька рішень, але для більшості тестувальників і DevOps-інженерів достатньо розуміти два ключові інструменти. Ось огляд основних гравців у цій екосистемі.



## Docker + Docker Hub

**Docker** – найпоширеніший рантайм контейнерів у світі. Простий у використанні, має величезну спільноту і чудову документацію.

**Docker Hub** – публічний реєстр образів. Звідти можна завантажувати готові образи (наприклад, Node, PostgreSQL, Selenium) і публікувати власні. Тисячі готових образів для будь-яких потреб тестування.

РЕКОМЕНДОВАНО ДЛЯ ПОЧАТКУ

## RKT (вимовляється «ракета»)

**RKT** – альтернативна контейнерна система від CoreOS з акцентом на безпеку та стандартизацію.

Використовує специфікацію App Container (appc) і підтримує запуск контейнерів без привілейованого демону.

RKT менш поширена в навчальних матеріалах і у повсякденній практиці, але варто знати про її існування – особливо в контекстах, де безпека є пріоритетом.

для поглибленого вивчення

- Для початку **достатньо розуміти Docker і Docker Hub**. Саме ці інструменти ви зустрінете в більшості CI/CD пайплайнів, навчальних матеріалів і реальних проектів.

# Провайдери віртуальних машин (для контексту)

Хоча для тестування в CI/CD частіше використовують контейнери, важливо розуміти основні платформи віртуалізації – особливо для контексту і порівняння. Ці інструменти широко застосовуються в корпоративному середовищі та локальній розробці.

## VirtualBox

**Oracle VirtualBox** – безкоштовна платформа віртуалізації x86. Підтримує Windows, macOS і Linux як хост і гость. Зручна для локальної розробки та навчання.

- Безкоштовна та відкрита
- Підтримує знімки стану (snapshots)
- Широка підтримка гостьових ОС
- Ідеальна для навчання

## VMware

**VMware** – комерційні рішення з гіпервізором і зручним керуванням VM. Включає VMware Workstation, Fusion і vSphere для корпоративного використання.

- Надійний корпоративний стандарт
- Розширені можливості мережі
- Висока продуктивність
- Комерційна підтримка

■ Швидкість старту (бали)

■ Рівень ізоляції (бали)

Інструмент

Docker контейнер



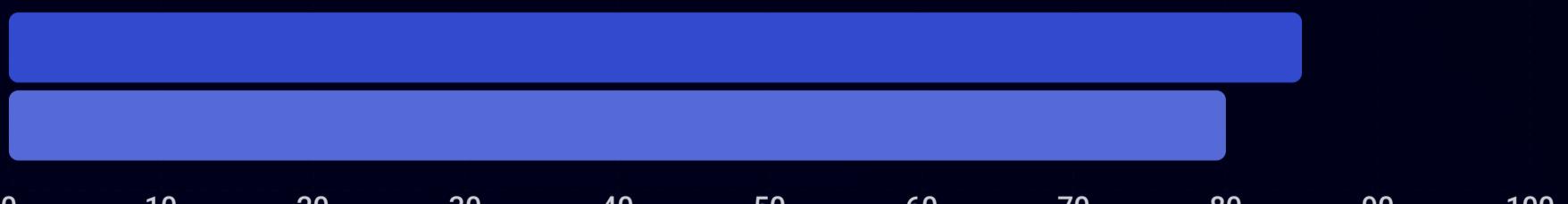
VirtualBox VM



VMware VM



RKT контейнер



Як видно з діаграмами, Docker контейнери значно випереджають VM за швидкістю старту, що робить їх ідеальним вибором для автоматизованого тестування в CI/CD пайплайнах. VM виграють за рівнем ізоляції, що може бути критично у специфічних сценаріях безпеки.

# Висновки

Docker став невід'ємним інструментом сучасного тестування. Розуміння його принципів дозволяє тестувальникам будувати надійні, відтворювані та ефективні процеси перевірки якості – від локального запуску до повноцінних CI/CD пайплайнів.



## Основа: однакове середовище

Docker дає **однакове середовище** для коду та тестів на будь-якій машині та в CI/CD. Це фундамент надійного тестування.



## Контейнер vs VM

**Контейнер** – легка упаковка (код + залежності) на спільному ядрі.  
**VM** – повна віртуальна машина з власною ОС. Кожен підхід має своє місце.



## Для тестувальника

Консистентність, відтворюваність, інтеграція з пайплайнами – головні переваги Docker у тестуванні. Врахуйте безпеку образів і спільне ядро хоста.

↓ 99%

## Менше «у мене працює»

Консистентне середовище усуває більшість проблем розбіжності оточень

~10x

## Швидший старт

Контейнери стартують у секунди порівняно з хвилинами для VM

1

## Образ – один для всіх

Один Docker-образ працює однаково локально, на CI і в хмарі

- ☐ **Наступний крок:** Встановіть Docker Desktop, завантажте офіційний образ Node.js з Docker Hub і спробуйте запустити свої тести всередині контейнера. Практика – найкращий шлях до розуміння!