

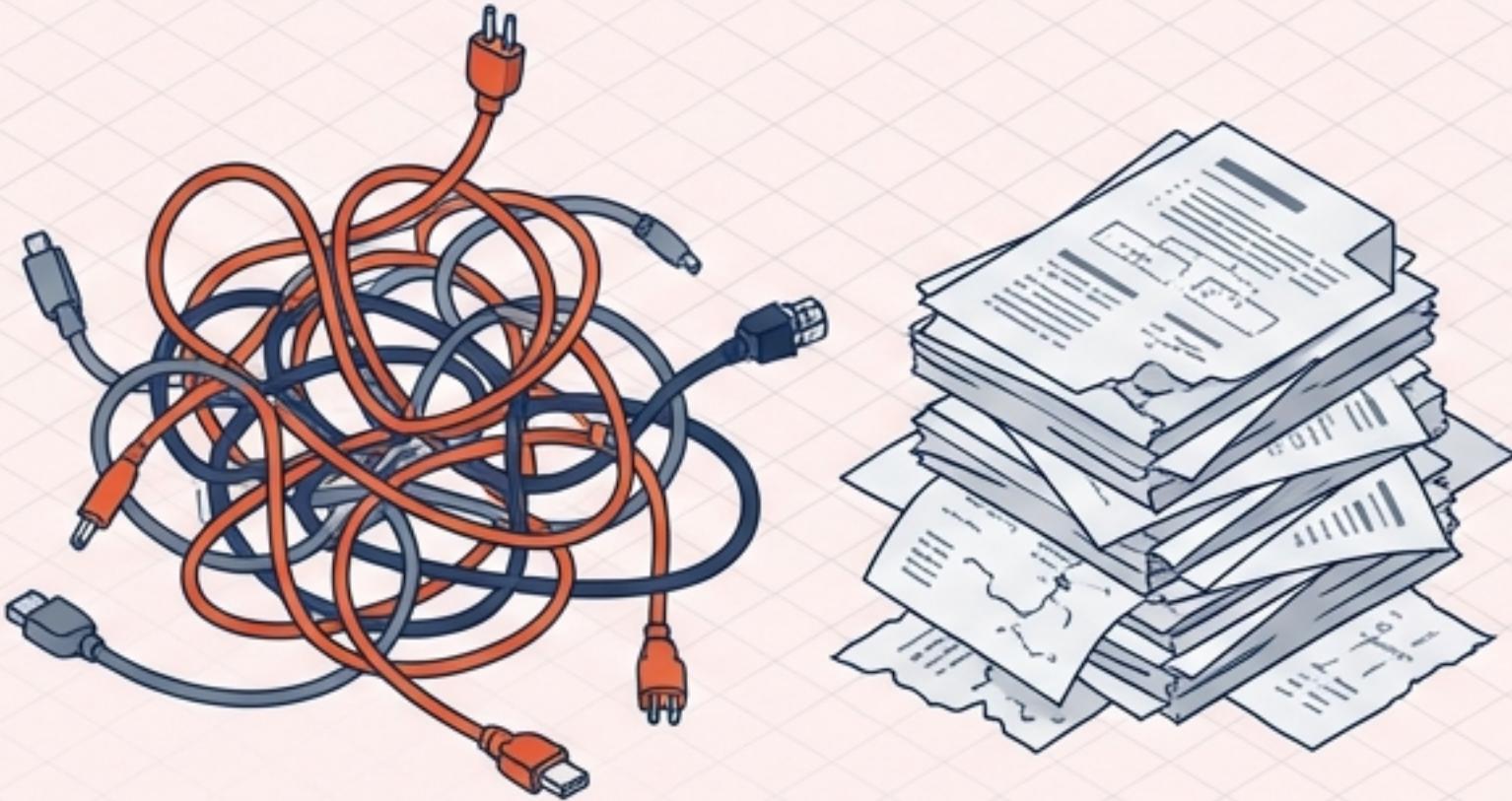
Інфраструктура для QA: Путівник по CI/CD, Docker та Tilt

Від хаосу локальних налаштувань до
автоматизованої досконалості.



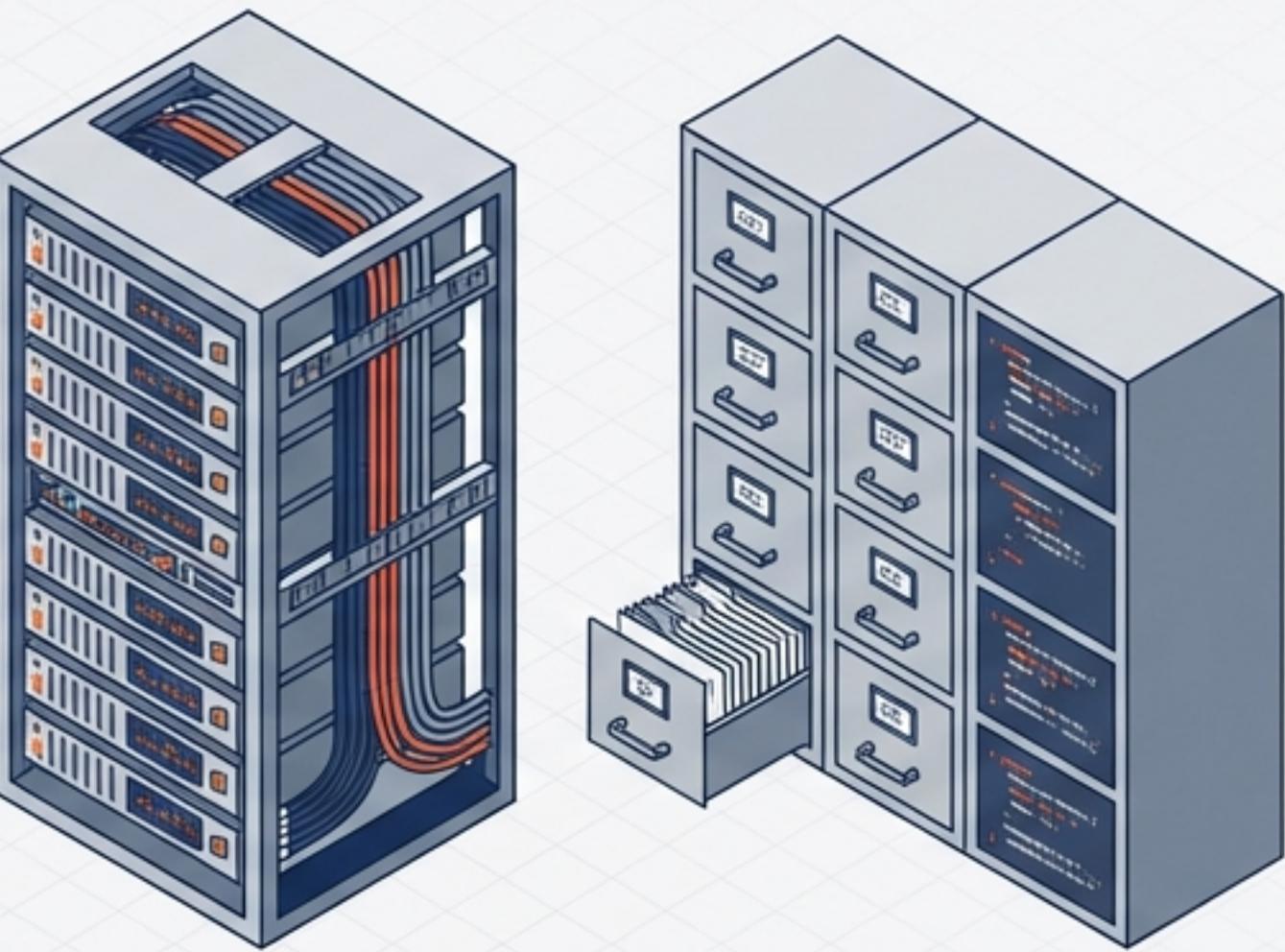
Посібник для сучасного інженера з автоматизації

Старий підхід



- ✗ У мене все працює (але падає на продакшені)
- ✗ Ручний деплой та налаштування середовища
- ✗ Пекло мерджу (Merge Hell) в кінці спринту
- ✗ Повільний зворотний зв'язок

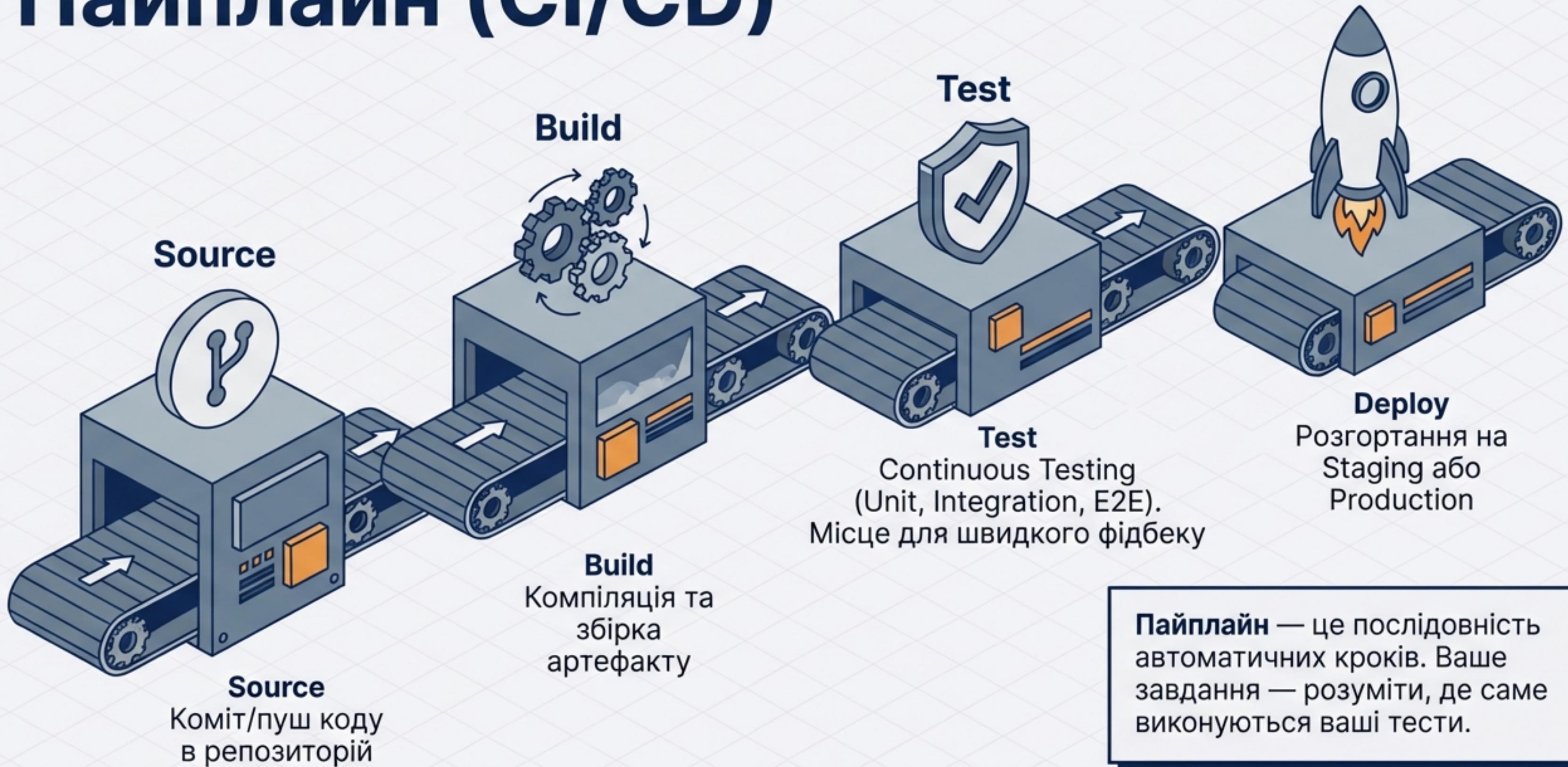
DevOps культура



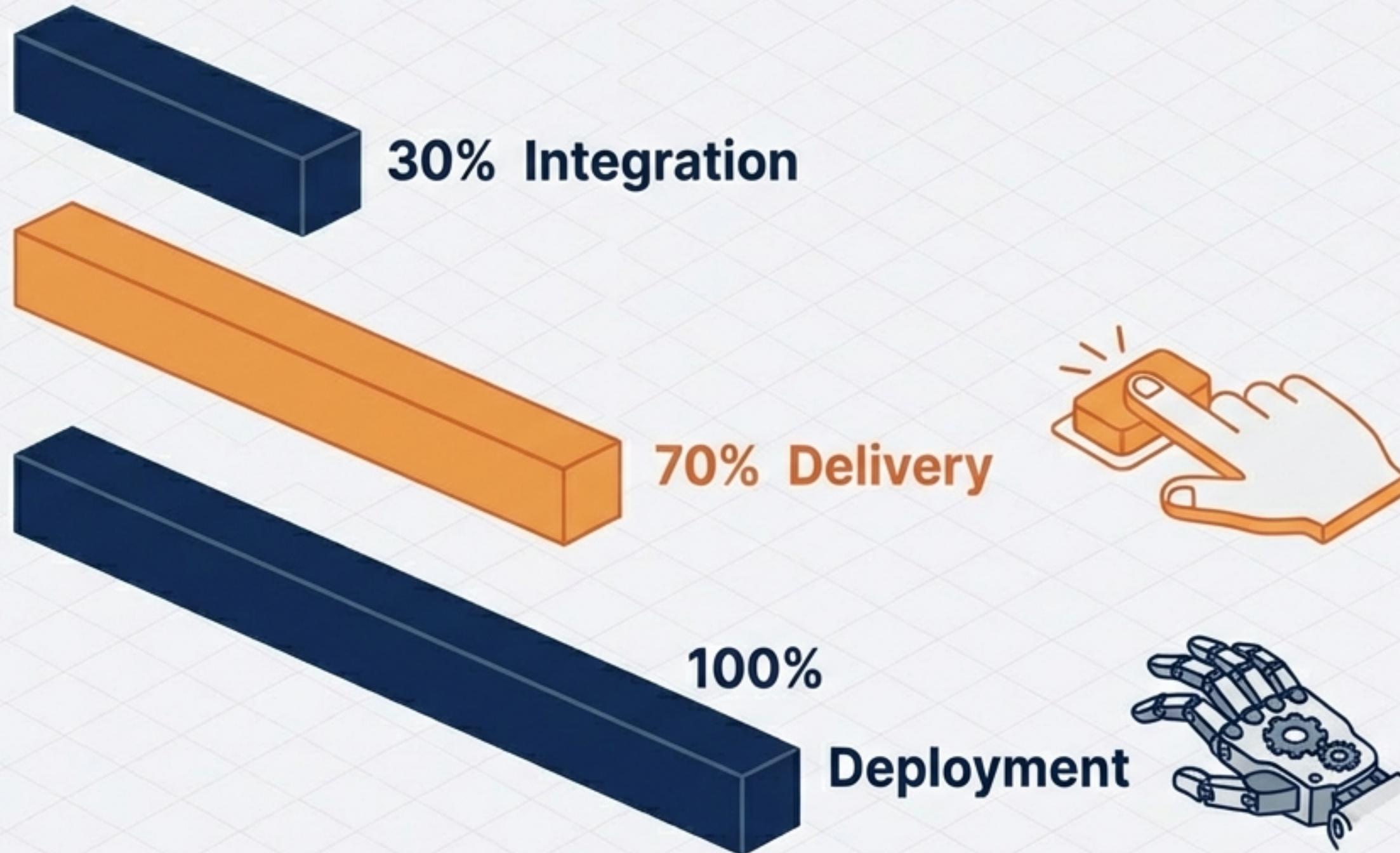
- ✓ CI: Часті злиття коду та автоматичні тести
- ✓ CD: Код завжди готовий до релізу
- ✓ Docker: Єднакове середовище всюди

Автоматизація — це не лише про швидкість. Це про безпеку та передбачуваність.

Пайпайн (CI/CD)



CI vs. CD vs. Deployment

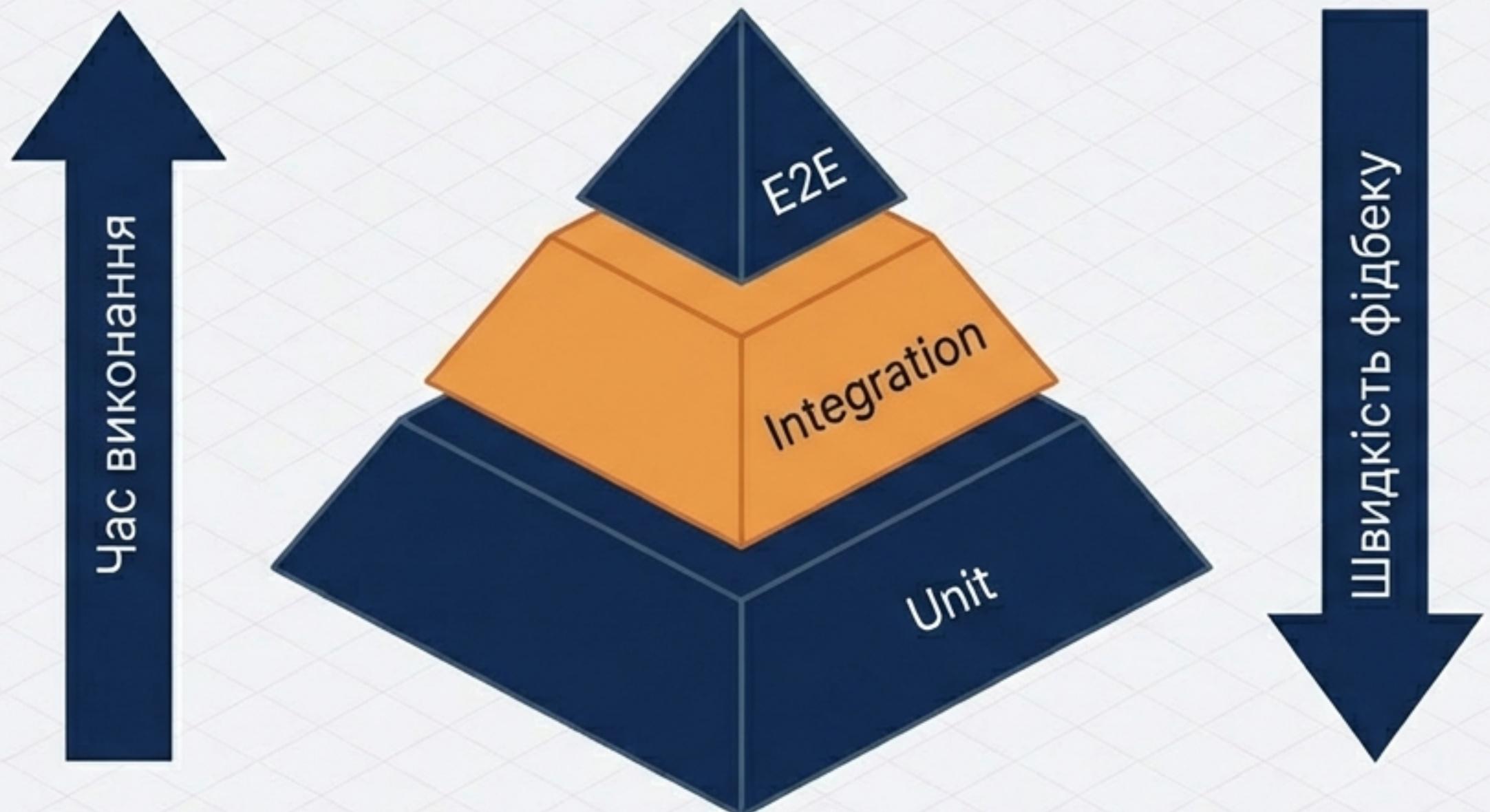


CI: Короткочасні гілки + Автоматична збірка + Тести. Фокус на якості коду.

CD (Delivery): Після тестів код готовий до релізу. Деплой на прод — це ручна кнопка.

CD (Deployment): Успішний білд автоматично летить на прод без втручання людини.

Роль тестувальника у пайплайні



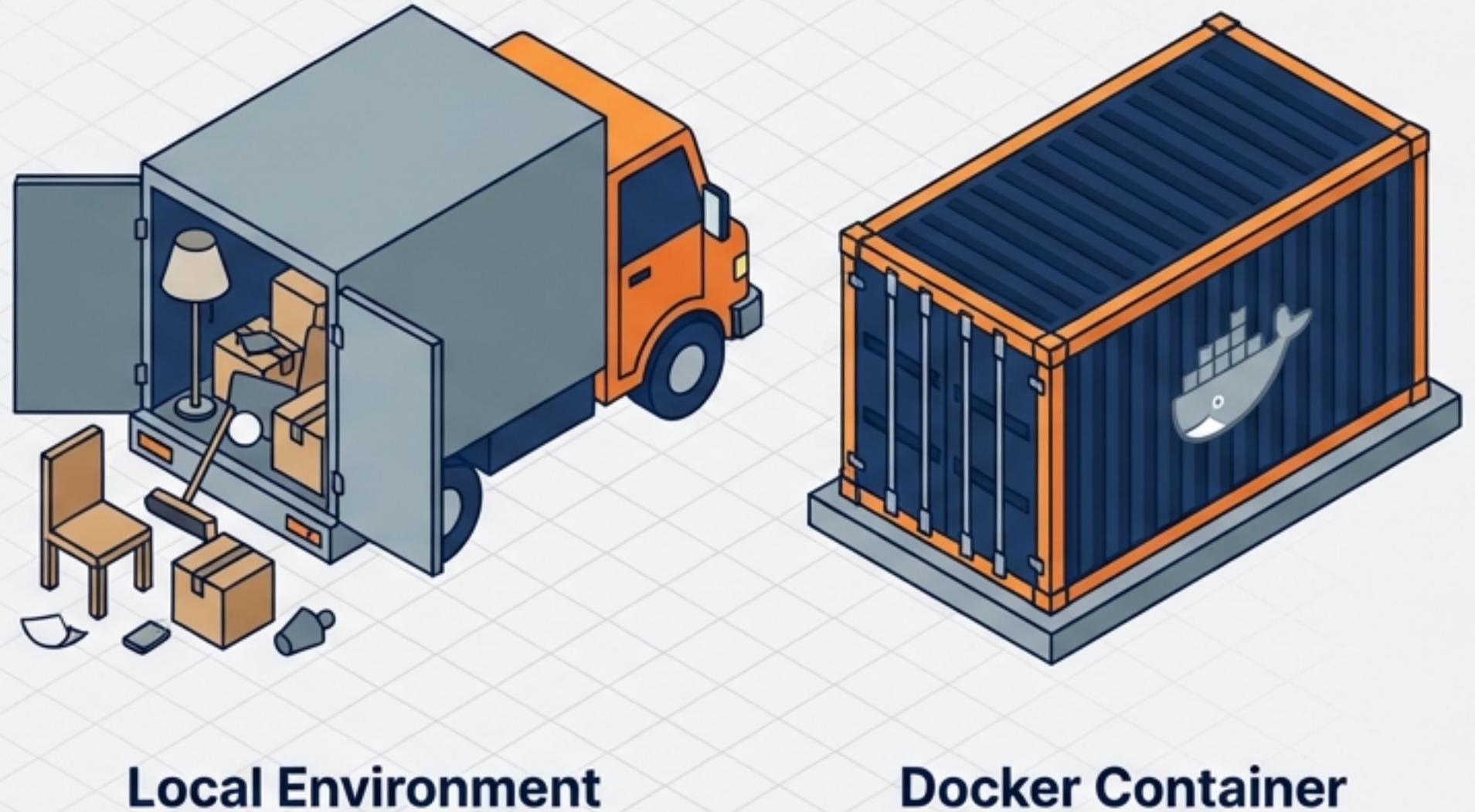
E2E (End-to-End): Повільні, імітують користувача. Запускаються рідше або паралельно.

Integration (Інтеграційні): Перевірка взаємодії компонентів (API + DB).

Unit (Юніт): Швидкі, дешеві, перевіряють ізольовані блоки. Основа CI.

Continuous Testing — це інтеграція тестів у пайплайн. Не обов'язково мати всі рівні одразу, але автоматичний запуск хоча б одного рівня — це вже перемога.

Docker: Кінець епохи "It works on my machine"

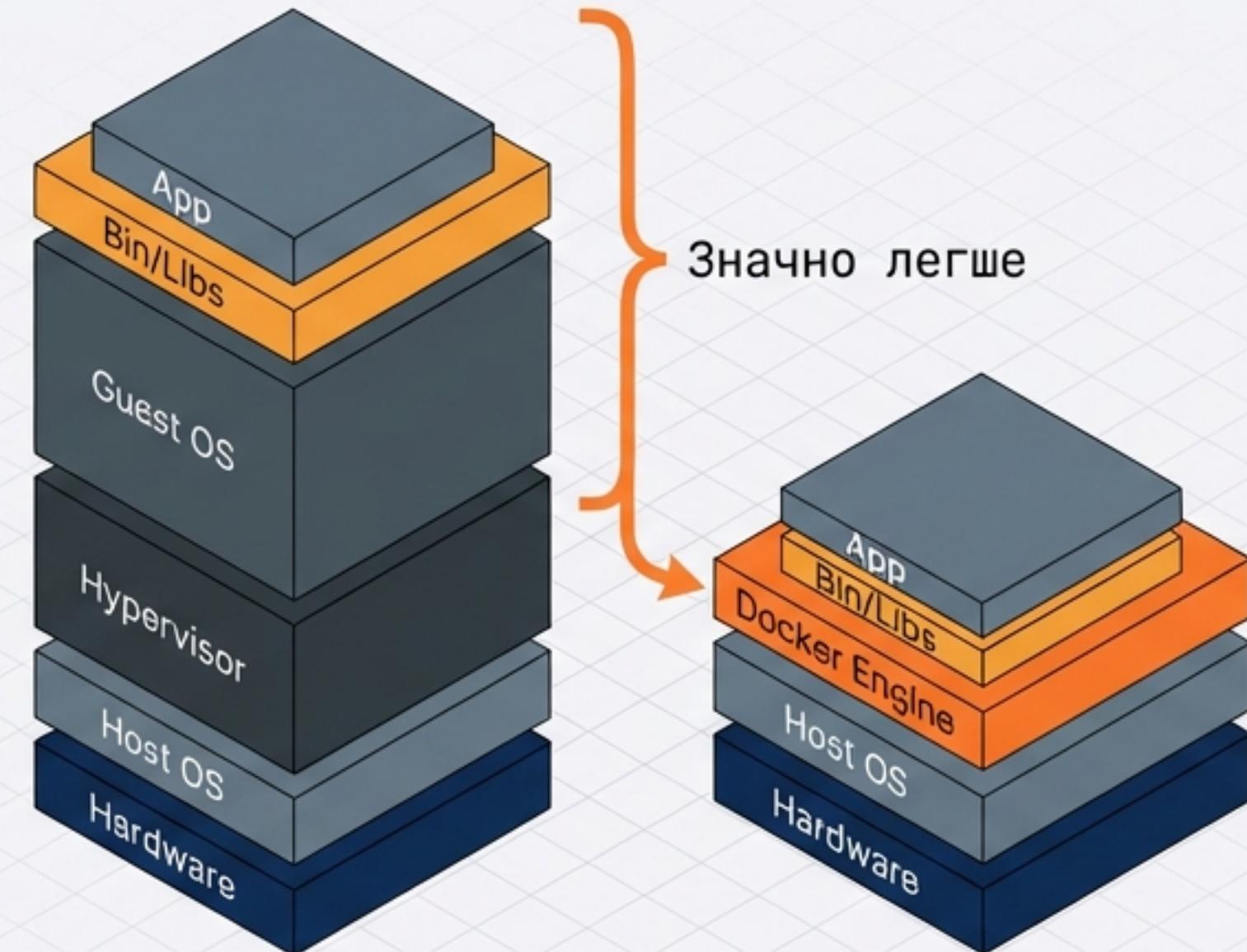


- **Контейнер** — це упаковане середовище (код, бібліотеки, налаштування).
- **Гарантія:** Якщо тест проходить у контейнері локально, він пройде і в пайплайні CI/CD.
- **Ізоляція:** Різні проекти з різними версіями Node.js не конфліктуватимуть на одній машині.

Контейнери vs. Віртуальні машини

VM (Віртуальна машина)

Емуляція цілого комп'ютера. Важка, повільний старт.

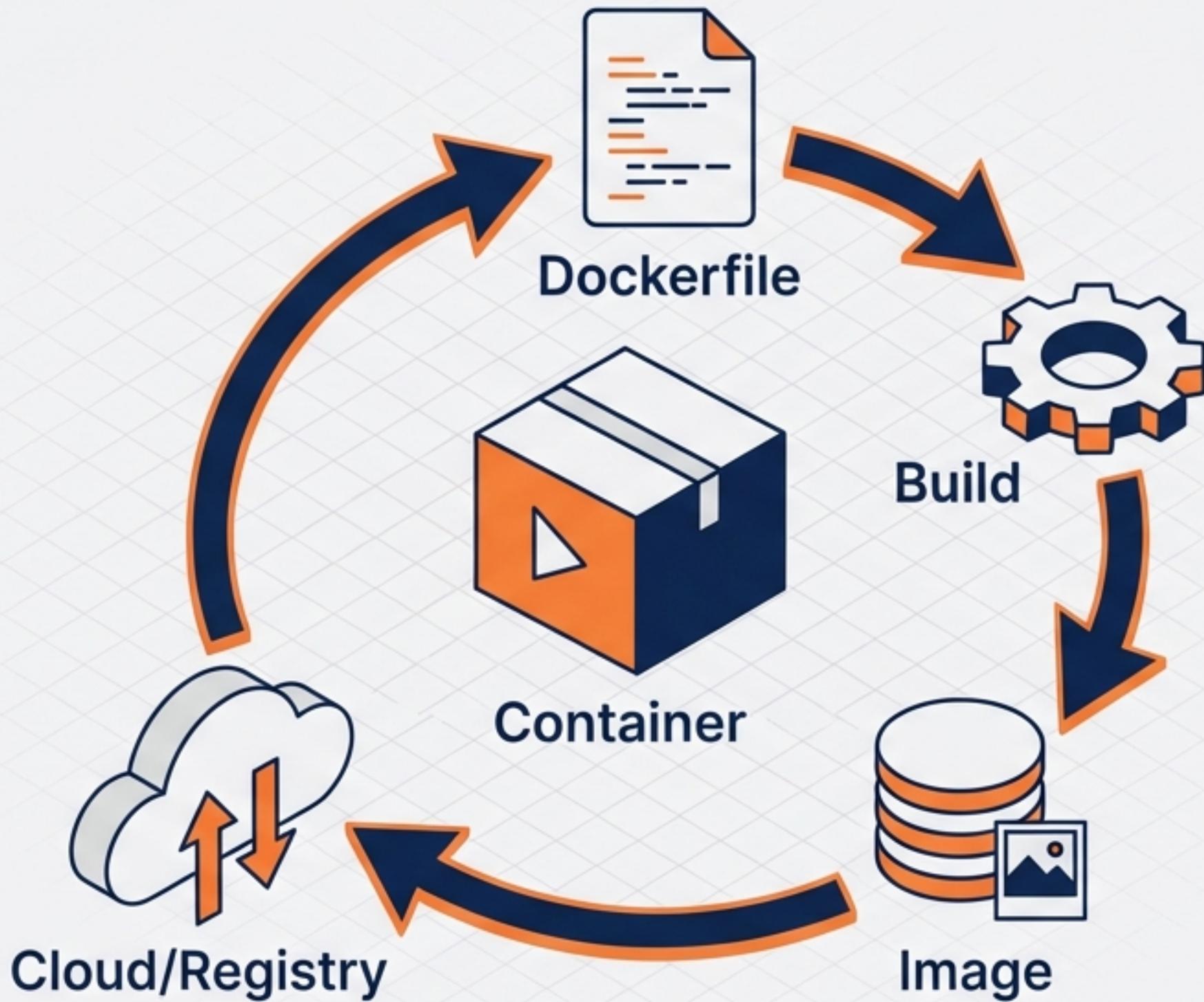


Container
(Docker)

Віртуалізація
програмного рівня.
Використовує ядро хоста.
Легкий, миттєвий старт.

Для CI/CD контейнери ідеальні — вони запускаються за секунди і споживають мінімум ресурсів.

Основні поняття екосистеми



Dockerfile:

Інструкція (рецепт). Текстовий файл, що описує, як зібрати образ.

Image (Образ):

Шаблон (Read-only). Результат збірки. "Знімок" файлової системи.

Container (Контейнер):

Живий процес. Запущений екземпляр образу.

Registry (Hub):

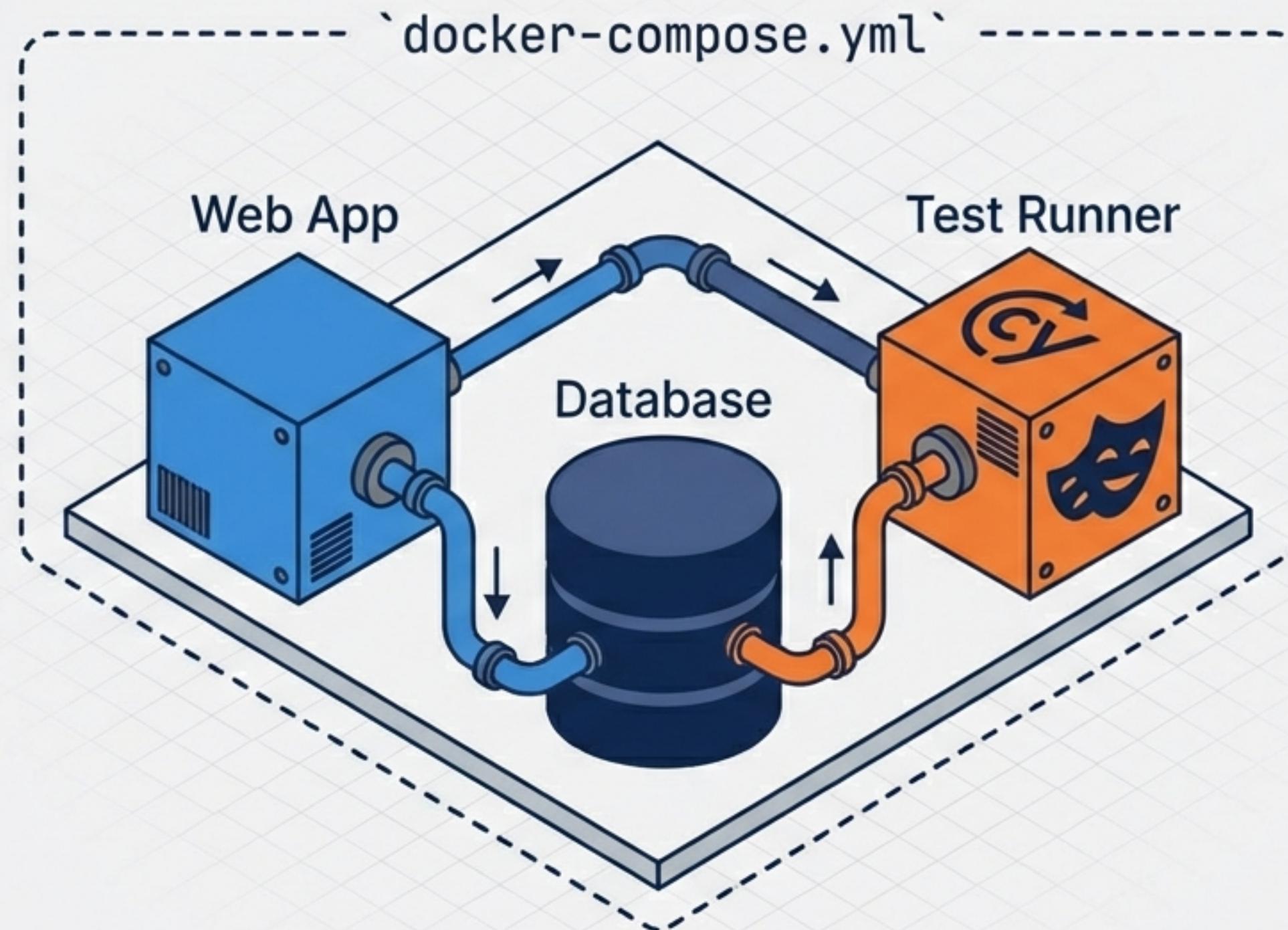
Склад образів (Docker Hub), звідки ми робимо pull і куди робимо push.

Docker Cheat Sheet

```
$ docker build .      → Зібрати образ з Dockerfile у поточній папці  
$ docker images       → Показати список доступних локальних образів  
$ docker pull [image] → Завантажити готовий образ з Docker Hub  
$ docker run -it [image] → Створити та запустити контейнер  
$ docker ps           → Показати запущені контейнери
```

 **Pro Tip:** Використовуйте `.dockerignore`, щоб не тягнути зайві файли в контекст збірки.

Docker Compose: Оркестрація для тестувальника



- Дозволяє описати багатоконтейнерну систему в одному файлі.
- Одна команда `docker compose up` піднімає все оточення.

Ідеально для інтеграційних тестів: підняли базу, підняли бекенд, запустили контейнер з Cypress, отримали звіт.

Docker для Test Automation



Cypress

- Використовуйте `cypress/included` (містить OS, Cypress, браузери).
- Фіксуйте теги (напр., `:13.6.0`), уникайте `latest` для стабільності.

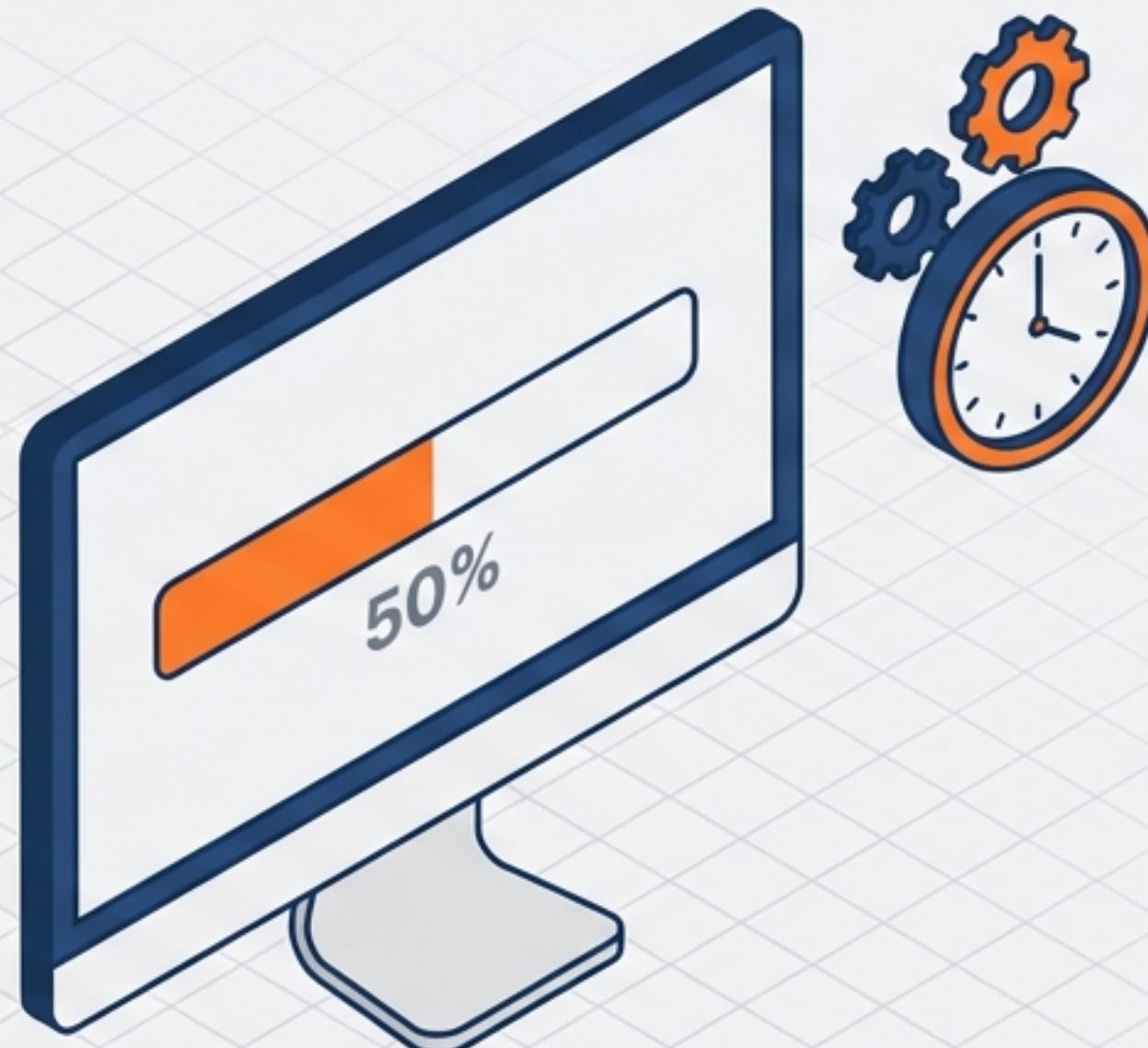


Playwright

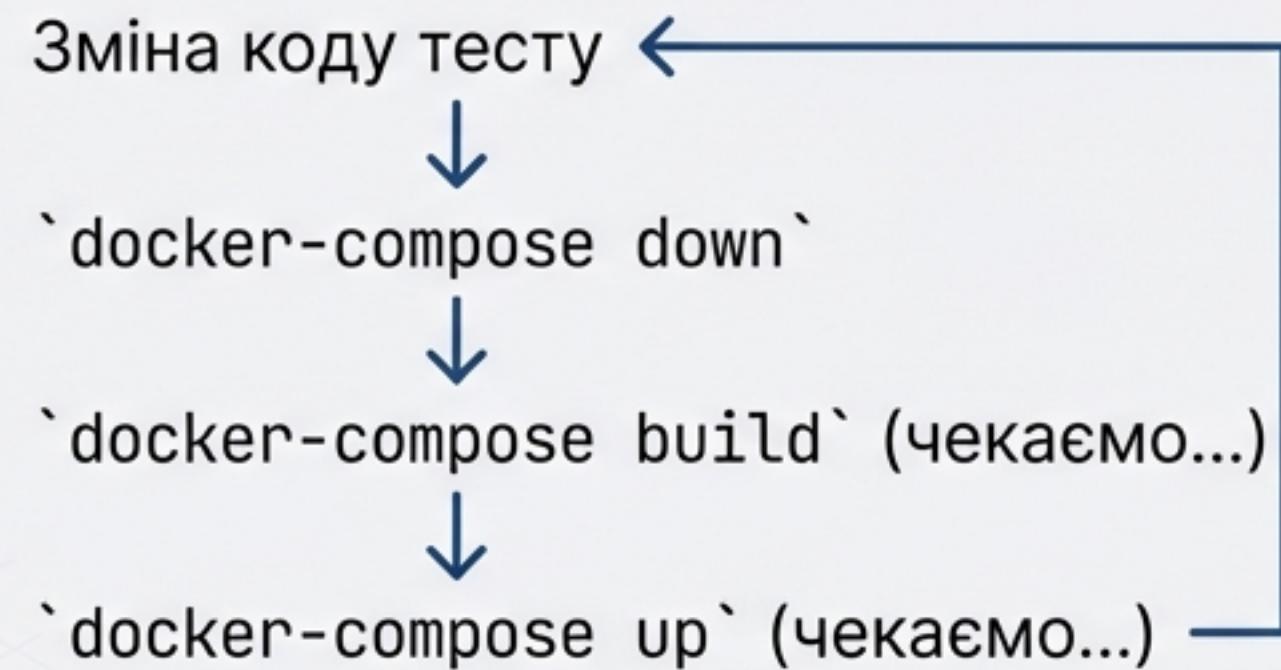
- Образи в Microsoft Container Registry (MCR).
- Для Chrome обов'язково: `--ipc=host` (унікнення крашу пам'яті).
- Теги `jammy/focal` вказують на версію Ubuntu.

```
docker run -v $PWD:/e2e -w /e2e ...  
(Монтування папки з тестами)
```

Проблема локальної розробки: Повільний цикл фідбеку



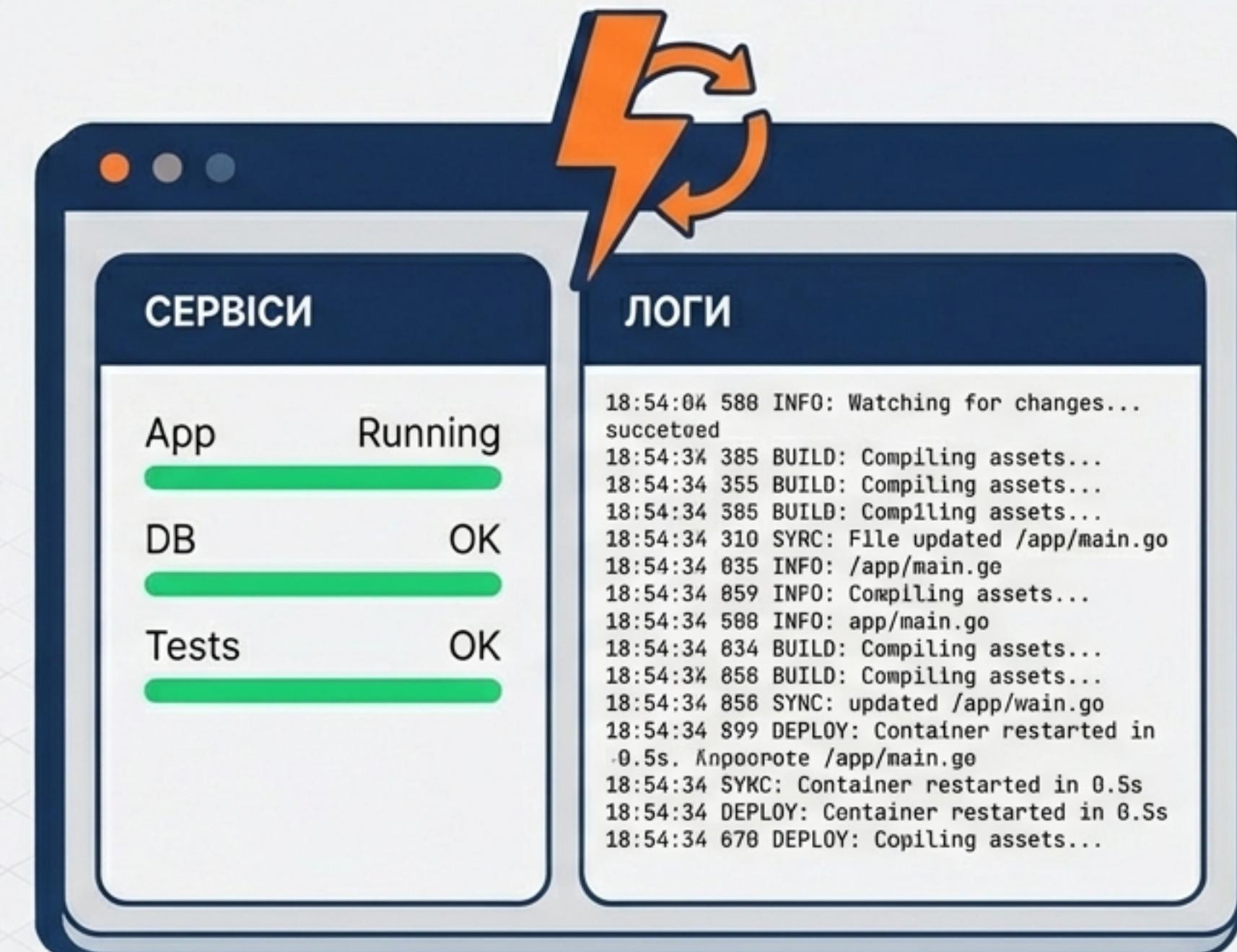
Цикл фідбеку:



Стандартні Docker-команди чудові для CI,
але занадто повільні для активного
написання тестів.

Tilt: Від хаосу до порядку

- **Live Update:** Синхронізація змінених файлів у запущений контейнер без повного ребілду.
- **Dashboard:** Веб-інтерфейс для моніторингу всіх сервісів та логів в одному вікні.
- **Automation:** Ви пишете код — Tilt оновлює середовище миттєво.



Як працює Tilt (Tiltfile)

Tiltfile

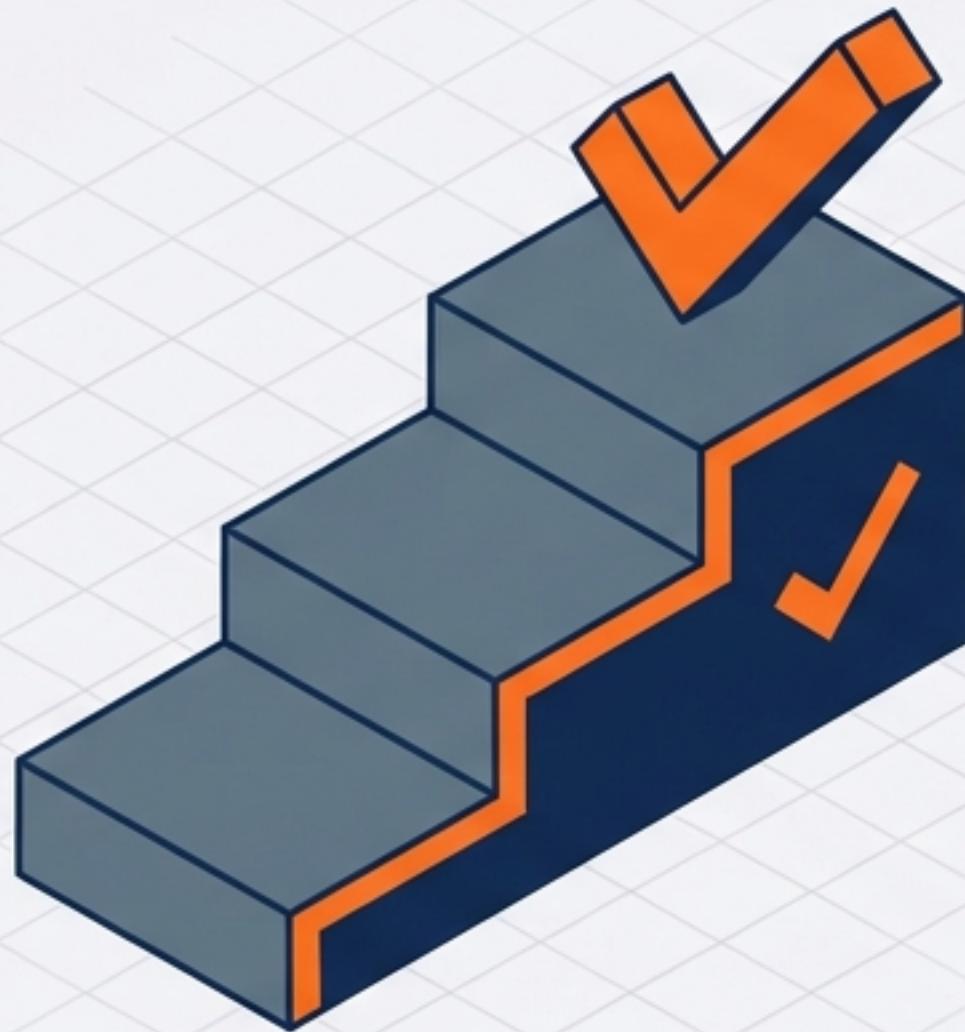
```
docker_build('my-image', '.', live_update=[  
    sync('.','/app')  
])  
docker_compose('docker-compose.yml')  
local_resource('test', 'npm test')
```

Вказує, як збирати образ і налаштовує live_update.

Підключає ваш існуючий docker-compose.yml.

Автоматичний запуск локальних команд при зміні файлів.

Підсумки та поради



- 1. **Пайплайн (CI/CD)** — це фундамент.
Розумійте процес.
- 2. **Docker** — це стандарт. Забезпечте
консистентність середовища.
- 3. **Tilt** — це прискорювач.
Автоматизуйте локальну рутину.

Почніть з малого. Спочатку запустіть тест у контейнері, потім додайте його в Compose, і лише потім оптимізуйте через Tilt.