Engineer

CI

щоденне злиття змін, кожне з яких перевіряє збірка

Що це таке

CI — скорочення від continuous integration, українською «безперервна інтеграція»: спосіб роботи, за якого кожен у команді вливає свої зміни в одну спільну основну гілку багато разів на тиждень, а машина щоразу збирає й тестує результат, щоб помилка проявилася, поки її автор ще пам'ятає, навіщо вніс зміну.

Цим словом називають дві різні речі, і якщо їх розрізняти, плутанини стає значно менше. Перша — практика, і вона про людей. Стаття Мартіна Фаулера на цю тему, яку зрештою цитує більшість команд, описує її як набір звичок: одна основна гілка, автоматизована збірка, збірка, що сама себе тестує, злиття в основну гілку щонайменше раз на день, збірка на кожне надсилання змін, негайне виправлення зламаної збірки, швидка збірка, приховування незавершеної роботи замість відкладання її в окрему гілку і стан збірки, який бачать усі. Стаття виводить назву з Extreme Programming і згадує, що Грейді Буч уживав цей вислів ще раніше.

Друге значення — машина. Коли кажуть «CI червоний», мають на увазі сервіс, який запустив збірку й повідомив про помилку. GitHub Actions, GitLab CI/CD та подібні до них мають однакову будову. Файл у репозиторії описує роботу: `.github/workflows/*.yml` на GitHub, `.gitlab-ci.yml` у корені на GitLab. Подія запускає її: push, pull request чи merge request, розклад або натискання кнопки. Runner, тобто виділена для цього машина, отримує коміт, виконує команди з файлу і повертає до коміту результат: пройшло чи впало.

Сам лише сервер ще не є безперервною інтеграцією. Команда, яка місяць тримає роботу в гілках і дає серверу збирати кожну з них, автоматизувала збірки, і це корисно, але інтеграція все одно відбувається разом і наприкінці.

Практика — це часте злиття, а машина робить його достатньо безпечним, щоб від нього не відмовлялися.

Continuous delivery і continuous deployment — наступні кроки. Delivery тримає кожну зелену збірку готовою до релізу, а deployment випускає кожну з них автоматично. Обидва спираються на вже налагоджений CI, тому всі три так часто пишуть разом як CI/CD.

Застосуйте поняття до рішення

Застосуйте «Що таке CI?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Перевірено, випущено і з можливістю відкату, тож зміни доходять до клієнтів у вівторок удень, а не пізно в п'ятницю.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Автоматизувала розгортання GIS SaaS через CI/CD — розгортання, дані та звітність — усе без ручної роботи

Автоматизувала розгортання GIS SaaS через CI/CD Зв'язатися з нами →

Чому це важливо

Ціна злиття двох ліній роботи зростає швидше, ніж час, який вони провели окремо. Двоє розробників, що вчора змінили один і той самий модуль, розв’яжуть конфлікт за кілька хвилин, бо обоє пам’ятають свій задум. Двоє, чиї гілки розійшлися шість тижнів тому, відновлюють задум із diff-ів. Невеликі часті злиття тримають кожен конфлікт маленьким, і саме в цьому більша частина користі ще до того, як з’являється будь-яка машина.

Збірка на кожен push додає ще одне: чисту машину, яка погоджується. «На моєму ноутбуці працює» — твердження про один ноутбук із його власними бібліотеками й забутими файлами. Runner, що щоразу починає зі свіжого checkout, або відтворює результат, або показує, чим ноутбук був особливим.

Цінність — у швидкості. Стаття Фаулера ставить за мету близько десяти хвилин на основну збірку, і на практиці це обґрунтування витримує перевірку. Помилка, про яку повідомлено через десять хвилин після push, повертається до того, хто її спричинив, поки зміна ще в нього на екрані. Помилка, про яку дізнаються наступного ранку, дістається тому, хто випадково подивиться, і спершу коштує години розкопок.

Важить і видимість. Коли стан основної гілки бачать усі, ніхто не починає нову роботу поверх зламаної збірки, не знаючи про це, а виправлення збірки стає першою справою команди, а не чиєюсь чужою проблемою.

CI також не обов'язково означає хмарний сервіс. pre-commit запускає перевірки проєкту як Git-хуки перед кожним комітом або push, а його документація показує, як той самий набір виконується на сервері через `pre-commit run --all-files`. Task дає кожній перевірці ім'я в одному `Taskfile.yml`, тож людина й runner викликають ту саму команду. Перевірки цього сайту працюють саме так: на машині, де пишеться сайт, із повною перевіркою перед кожним push і без хмарного сервісу збірки посередині.

Як це виглядає на практиці

Випуск програм на одному проєкті тримався на ручних кроках, які кожен виконував трохи по-своєму. Їх замінили автоматизовані пайплайни збирання, тестування та розгортання, і шлях від зміни до production став стандартизованим, а не імпровізованим. Сторінка оцінює повернений час приблизно в 4 000 годин, а релізи стали водночас швидшими й надійнішими.

Для довгоживучої бази даних CI стає охороною. На проєкті з такою базою схему описано двічі: один раз початковим налаштуванням і ще раз — 1 022 міграціями, пронумерованими від 036 до 1102. CI-задача на кожну зміну збирає базу обома шляхами: зі свіжої схеми і з усіма міграціями, відтвореними по черзі, — а потім порівнює і структуру, і початкові дані. Будь-яка розбіжність валить білд із виведеною різницею, тож свіже середовище й довгоживуче — одна й та сама база, і це перевіряється, а не припускається.

Дві половини застосунку CI тримає в згоді на ще одному проєкті, де опис API і клієнт фронтенду генеруються ланцюгом, що починається з бази даних і закінчується типізованим клієнтом TypeScript на 44 076 рядків. Перевірка контракту в CI порівнює те, що надсилає фронтенд, із тим, що очікує API, і завалює білд за розбіжності. Результат, який описує сторінка, — саме те, заради чого існує CI: поле, змінене лише з одного боку, падає у білді за кілька хвилин після зміни, а не доходить до користувача помилкою в браузері.

Де виникають проблеми

Більшість налаштувань CI радше поступово занепадають, ніж ламаються. Кожен із цих сценаріїв починається з дрібниці, а закінчується збіркою, яку команда навчилася обходити.

Збірка триває довше, ніж сама робота. Коли відповідь на push приходить через сорок хвилин, люди перестають чекати, складають кілька змін в один push і надсилають рідше, тобто роблять протилежне до практики. Спершу варто виміряти, куди йде час, а вже потім купувати залізо. Кешуйте залежності, запускайте швидкі перевірки першими, а повільні набори тестів виносьте в окремий етап або за розкладом, щоб основна збірка трималася біля десяти хвилин.

Основна гілка цілий день стоїть червоною. Зламана збірка, яку ніхто не лагодить, стає звичним станом, і нова помилка ховається за старою. Правило, що цьому запобігає, просте: хто зламав, той виправляє зараз або відкочує зміну й виправляє її спокійно. Зелена основна гілка — те, на чому будують усі інші.

Гілки живуть тижнями, а сервер збирає кожну з них. Кожна гілка світиться зеленим, а проблеми приходять разом у день злиття. Це автоматизована збірка без інтеграції. Зливайте в основну гілку щонайменше щодня, а незавершені функції тримайте вимкненими, а не відокремленими.

Сервер перевіряє один список, а ноутбук — інший. Два списки команд, які ведуть вручну, розходяться. Тоді зміна проходить локально й падає на сервері або навпаки, і довіри не лишається до жодного з них. Дайте перевіркам одну точку входу й викликайте її з обох місць. З pre-commit сервер може запускати саме ті хуки, які запускає кожен розробник:

```
# .gitlab-ci.yml
check:
    image: python:3.13
    script:
        - pip install pre-commit
        - pre-commit run --all-files --show-diff-on-failure
```

Нестабільний тест привчає всіх тиснути «перезапустити». Тест, що падає раз на двадцять запусків із причин, які до зміни стосунку не мають, привчає команду перезапускати замість читати. Після цього справжня помилка отримує те саме ставлення. Ізолюйте такий тест, виправте або видаліть його, а кнопку перезапуску, яку натискають часто, вважайте окремою знахідкою.

Що почитати далі

- Continuous Integration — довга стаття Мартіна Фаулера: кожна практика, збірка на десять хвилин і походження назви.
- Understanding GitHub Actions — робочі процеси, події, завдання й виконавці (runners) так, як їх визначає сам GitHub.
- GitLab CI/CD — файл `.gitlab-ci.yml` , пайплайни, етапи, завдання й виконавці у термінах самого GitLab.
- pre-commit — перевірки, оголошені один раз у конфігураційному файлі, що працюють як Git-хуки і ще раз як крок на CI-сервері.
- Task — `Taskfile.yml` , що дає кожній перевірці одне ім’я, тож людина й runner вводять ту саму команду.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке CI?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/ci/