Engineer

CI

merging work at least daily, each merge proven by a build

What it is

CI is short for continuous integration: a way of working in which everyone on a team lands their changes on one shared line of code many times a week, and a machine builds and tests the result each time so that a mistake surfaces while its author still remembers making it. The full name, continuous integration, is used as often as the abbreviation.

The word is used for two different things, and keeping them apart saves a lot of confusion. The first is the practice, which is about people. Martin Fowler's article on the subject, the source most teams end up quoting, describes it as a set of habits: keep one mainline, automate the build, make the build test itself, push to the mainline at least daily, build every push, fix a broken build at once, keep the build fast, hide unfinished work rather than keeping it on a branch, and let everyone see the state of the build. The article traces the name to Extreme Programming, and notes that Grady Booch used the phrase before that.

The second meaning is the machine. When someone says "the CI is red", they mean the service that ran the build and reported a failure. GitHub Actions, GitLab CI/CD and their peers all share one shape. A file in the repository describes the work: `.github/workflows/*.yml` on GitHub, `.gitlab-ci.yml` at the root on GitLab. An event starts it, such as a push, a pull or merge request, a schedule or a person pressing a button. A runner, which is a machine set aside for the purpose, checks out the commit, runs the commands in that file, and reports a pass or a fail back against the commit.

A server on its own is not continuous integration. A team that keeps work on branches for a month and has a server build each branch has automated its builds, which is useful, but the integrating still happens all at once at the end.

The practice is the frequent merging; the machine is what makes frequent merging safe enough to keep doing.

Continuous delivery and continuous deployment are the steps that come after. Delivery keeps every green build ready to release, and deployment releases each one automatically. Both depend on CI being in place first, which is why the three are so often written together as CI/CD.

Use the concept in a decision

Put “What is CI?” to work in a real decision. We can help relate the concept to the system you are building or buying. Tested, released and reversible, so changes reach customers on a Tuesday afternoon rather than late on a Friday night.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Automated GIS SaaS deployment with GitHub Actions — deployment, data and reporting, all off the hands

Automated GIS SaaS deployment with GitHub Actions Get in touch →

Why it matters

The cost of merging two lines of work grows faster than the time they spend apart. Two people who each changed the same module yesterday can sort out the conflict in a few minutes, because both remember what they meant. Two people who diverged six weeks ago are reconstructing intent from diffs. Merging small pieces often keeps every conflict small, and that is most of the benefit before any machine is involved.

The build on every push adds a second thing: a clean machine that agrees. “It works on my laptop” is a claim about one laptop, with its own installed libraries and leftover files. A runner that starts from a fresh checkout every time either reproduces the result or shows where the laptop was special.

Speed is where the value lives. Fowler’s article sets a target of about ten minutes for the main build, and the reasoning holds up in practice. A failure reported ten minutes after a push goes back to the person who caused it while the change is still on their screen. A failure reported the next morning goes to whoever happens to look, and costs an hour of archaeology first.

Visibility matters as well. When the state of the mainline is shown to everyone, nobody starts new work on top of a broken build without knowing it, and fixing that build becomes the team's first job rather than someone else's problem.

CI also need not mean a hosted service. pre-commit runs a project's checks as Git hooks before each commit or push, and its documentation shows the same set run on a server with `pre-commit run --all-files`. Task gives each check a name in one `Taskfile.yml`, so a person and a runner call the same command. The checks behind this site follow that model. They run locally, where the pages are written, and a full check runs before every push with no hosted build service in between.

What it looks like in practice

Roughly 4,000 hours were given back once a team stopped releasing by hand. Getting software out had relied on manual steps that each person did slightly differently, until automated build, test and deployment replaced them. The path from a change to production became a standard one rather than an improvised one, and the releases became both faster and more reliable.

On a project with a long-lived database, the schema is described twice: once by the initial setup and once by 1,022 migrations, numbered 036 through 1102. A CI job builds the database both ways on every change, once from the fresh schema and once with every migration replayed in order, and compares the structure and the seeded data. Where the two builds differ, the job goes red and prints what differs, so a fresh environment and a long-lived one are the same database by check rather than by assumption.

CI can also keep two halves of one application in agreement. On another project, the API description and the frontend's client are generated in a chain that starts at the database, and the client alone is 44,076 lines of typed TypeScript. A CI check holds what the frontend sends against what the API expects and fails the build when they diverge. That is the result CI exists for: a field that changes on one side only fails in a build, minutes after the change, instead of reaching a user as an error in the browser.

Where it goes wrong

Most CI setups decay rather than break. Each of these starts small and ends with a build the team has learned to work around.

The build takes longer than the work. Once a push takes forty minutes to come back, people stop waiting for it, batch several changes into one push and push less often, which is the opposite of the practice. Measure where the time goes before adding hardware. Cache dependencies, run the fast checks first, and move the slow suites to a second stage or a schedule, so that the main build stays near the ten-minute mark.

The mainline stays red for a day. A broken build that nobody fixes becomes the normal state, and a second failure arrives unnoticed behind the first. The rule that keeps this from happening is plain: whoever broke it fixes it now, or reverts the change and fixes it on their own time. A green mainline is the thing everyone else is building on.

Branches live for weeks while the server builds each one. Every branch shows green, and the trouble arrives on the day they merge, all at once. This is automated building without the integration. Merge to the mainline at least daily, and keep unfinished features switched off rather than kept apart.

The server checks one list and the laptop checks another. Two lists of commands kept by hand drift apart, and then a change passes locally and fails on the server, or the reverse, and nobody trusts either. Give the checks one entry point and call it from both places. With pre-commit, the server can run exactly the hooks every developer runs:

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

A flaky test teaches everyone to press retry. A test that fails one run in twenty for reasons unrelated to the change trains the team to rerun instead of read. After that, a real failure gets the same treatment. Quarantine the test, fix it or delete it, and treat a retry button that gets pressed often as a finding in its own right.

Further reading

- Continuous Integration — Martin Fowler's long article: each practice, the ten-minute build, and where the name came from.
- Understanding GitHub Actions — workflows, events, jobs and runners as GitHub defines them.
- GitLab CI/CD — the `.gitlab-ci.yml` file, pipelines, stages, jobs and runners in GitLab's own terms.
- pre-commit — checks declared once in a config file, run as Git hooks and again as a step on a CI server.
- Task — a `Taskfile.yml` that gives each check one name, so a person and a runner type the same command.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how "What is CI?" applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

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

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/ci/