Engineer

CI

at flette ændringer ind dagligt og bygge og teste hver gang

Hvad det er

CI står for continuous integration, på dansk ofte kaldt kontinuerlig integration: en arbejdsform, hvor alle på et team lægger deres ændringer ind på én fælles hovedlinje mange gange om ugen, og en maskine bygger og tester resultatet hver gang, så en fejl dukker op, mens den, der lavede den, stadig husker hvorfor.

Ordet bruges om to forskellige ting, og det sparer mange misforståelser at holde dem adskilt. Den første er praksissen, og den handler om mennesker. Martin Fowlers artikel om emnet, som de fleste teams ender med at citere, beskriver den som en række vaner: hold én hovedlinje, automatisér buildet, lad buildet teste sig selv, skub til hovedlinjen mindst én gang om dagen, byg hvert skub, ret et brudt build med det samme, hold buildet hurtigt, skjul halvfærdigt arbejde i stedet for at parkere det på en gren, og lad alle se, hvordan buildet står. Artiklen fører navnet tilbage til Extreme Programming og nævner, at Grady Booch brugte udtrykket før det.

Den anden betydning er maskinen. Når nogen siger »CI'en er rød«, mener de den tjeneste, der kørte buildet og meldte en fejl. GitHub Actions, GitLab CI/CD og deres slægtninge har samme grundform. En fil i repositoriet beskriver arbejdet: `.github/workflows/*.yml` hos GitHub, `.gitlab-ci.yml` i roden hos GitLab. En hændelse sætter det i gang, for eksempel et push, en pull- eller merge-request, en tidsplan eller et tryk på en knap. En runner, altså en maskine sat af til formålet, henter commit'et, kører kommandoerne fra filen og melder bestået eller fejlet tilbage på commit'et.

En server alene er ikke continuous integration. Et team, der holder arbejdet på grene i en måned og lader en server bygge hver gren, har automatiseret sine builds, og det er nyttigt, men selve integrationen sker stadig på én gang til sidst.

Praksissen er den hyppige sammenfletning; maskinen er det, der gør den sikker nok til, at man bliver ved.

Continuous delivery og continuous deployment er trinene bagefter. Delivery holder hvert grønt build klar til udgivelse, og deployment udgiver hvert af dem automatisk. Begge forudsætter, at CI er på plads, og derfor skrives de tre så tit sammen som CI/CD.

Brug begrebet i en beslutning

Brug »Hvad er CI?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Testet, udgivet og til at rulle tilbage, så ændringer når kunderne en tirsdag eftermiddag i stedet for sent en fredag aften.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har automatiseret GIS-SaaS-udrulning med CI/CD — udrulning, data og rapportering, alt sammen uden manuelt arbejde

Har automatiseret GIS-SaaS-udrulning med CI/CD Kontakt os →

Hvorfor det betyder noget

Prisen for at flette to arbejdslinjer sammen vokser hurtigere end tiden, de har levet hver for sig. To udviklere, der begge ændrede det samme modul i går, kan løse konflikten på få minutter, fordi de husker, hvad de mente. To, der skiltes for seks uger siden, sidder og genskaber hensigter ud fra diffs. Små, hyppige sammenfletninger holder hver konflikt lille, og det er det meste af gevinsten, før nogen maskine overhovedet er med.

Buildet ved hvert push lægger en ting til: en ren maskine, der er enig. »Det virker på min bærbare« er en påstand om én bærbar med sine egne installerede biblioteker og efterladte filer. En runner, der starter fra et frisk checkout hver gang, gentager resultatet eller viser, hvor den bærbare var speciel.

Værdien ligger i hastigheden. Fowlers artikel sætter et mål på omkring ti minutter for hovedbuildet, og begrundelsen holder i praksis. En fejl, der meldes ti minutter efter et push, går tilbage til den, der lavede den, mens ændringen stadig står på skærmen. En fejl, der meldes næste morgen, lander hos den, der tilfældigvis kigger, og koster en times udgravning først.

Synlighed tæller også. Når hovedlinjens tilstand er synlig for alle, begynder ingen på nyt arbejde oven på et brudt build uden at vide det, og at rette buildet bliver teamets første opgave frem for en andens problem.

CI behøver heller ikke betyde en hostet tjeneste. pre-commit kører et projekts tjek som Git-hooks før hvert commit eller push, og dokumentationen viser det samme sæt køre på en server med `pre-commit run --all-files`. Task giver hvert tjek et navn i én `Taskfile.yml`, så et menneske og en runner kalder den samme kommando. Dette sites egne tjek kører netop sådan: på den maskine, sitet skrives på, med et fuldt tjek før hvert push og uden en hostet build-tjeneste imellem.

Sådan ser det ud i praksis

Udgivelser hvilede på manuelle trin, som hver person udførte en smule forskelligt, indtil automatiserede build-, test- og udrulningspipelines afløste dem, så vejen fra en ændring til produktion blev standardiseret i stedet for improviseret. Siden gør den sparede tid op til omkring 4.000 timer, og udgivelserne blev både hurtigere og mere pålidelige.

På et projekt med en langtlevende database er skemaet beskrevet to gange: én gang af den initielle opsætning og én gang af migrationerne, som tæller 1.022, fra nummer 036 til nummer 1102. Et CI-job bygger databasen begge veje ved hver ændring, én gang fra det friske skema og én gang med alle migrationer afspillet i rækkefølge, og sammenligner både struktur og seedede data. Enhver uenighed får buildet til at fejle med forskellen printet ud, så et friskt miljø og et langtlevende er den samme database, fordi det bliver tjekket, ikke antaget.

CI kan også holde to halvdele af en applikation enige. Ét sted genereres API-beskrivelsen og frontendens klient i en kæde, der starter ved databasen og ender i en typet TypeScript-klient på 44.076 linjer. Et kontrakt-tjek i CI sammenligner det, frontenden sender, med det, API'et forventer, og får buildet til at fejle ved afvigelse. Resultatet, siden melder, er netop det, CI findes for: et felt, der ændres på kun den ene side, fejler i et build, minutter efter ændringen, i stedet for at nå en bruger som en fejl i browseren.

Hvor det går galt

De fleste CI-opsætninger forfalder snarere end at gå i stykker. Hvert af disse mønstre begynder småt og ender med et build, teamet har lært at arbejde udenom.

Buildet tager længere tid end arbejdet. Når et push er fyrre minutter om at melde tilbage, holder folk op med at vente, samler flere ændringer i ét push og pusher sjældnere, hvilket er det modsatte af praksissen. Mål, hvor tiden går, før der købes mere hardware. Cache afhængighederne, kør de hurtige tjek først, og flyt de langsomme suiter til et senere trin eller en tidsplan, så hovedbuildet holder sig omkring ti minutter.

Hovedlinjen står rød en hel dag. Et brudt build, som ingen retter, bliver normaltilstanden, og en ny fejl gemmer sig bag den gamle. Reglen, der forhindrer det, er enkel: den, der brød det, retter det nu eller ruller ændringen tilbage og retter den i ro. Et grønt build er det, alle andre bygger videre på.

Grene lever i ugevis, mens serveren bygger hver af dem. Hver gren lyser grønt, og problemerne kommer samlet den dag, de flettes. Det er automatiseret bygning uden integrationen. Flet ind i hovedlinjen mindst dagligt, og hold halvfærdige funktioner slået fra i stedet for at holde dem adskilt.

Serveren tjekker én liste og den bærbare en anden. To lister med kommandoer, der vedligeholdes i hånden, glider fra hinanden. Så består en ændring lokalt og fejler på serveren, eller omvendt, og ingen stoler på nogen af dem. Giv tjekkene ét indgangspunkt, og kald det fra begge steder. Med pre-commit kan serveren køre præcis de hooks, hver udvikler kører:

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

En ustabil test lærer alle at trykke på »kør igen«. En test, der fejler én gang ud af tyve af grunde, som intet har med ændringen at gøre, træner teamet i at genstarte frem for at læse. Bagefter får en ægte fejl samme behandling. Sæt

testen i karantæne, ret den eller slet den, og betragt en genstartsknap, der bliver trykket på tit, som et fund i sig selv.

Læs mere

- Continuous Integration — Martin Fowlers lange artikel: hver praksis, buildet på ti minutter, og hvor navnet kom fra.
- Understanding GitHub Actions — workflows, hændelser, jobs og runners, som GitHub selv definerer dem.
- GitLab CI/CD — filen `.gitlab-ci.yml`, pipelines, stages, jobs og runners med GitLabs egne ord.
- pre-commit — tjek erklæret én gang i en konfigurationsfil, kørt som Git-hooks og igen som et trin på en CI-server.
- Task — en `Taskfile.yml`, der giver hvert tjek ét navn, så et menneske og en runner skriver den samme kommando.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er CI?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/ci/