Engineer

CI

å slå sammen arbeid minst daglig, hver gang bevist ved bygg

Hva det er

CI er kort for *continuous integration*, på norsk kontinuerlig integrasjon: en arbeidsmåte der alle på et team lander endringene sine på én felles kodelinje mange ganger i uka, og en maskin bygger og tester resultatet hver gang, så en feil kommer fram mens den som gjorde den, ennå husker det. Det fulle navnet, kontinuerlig integrasjon, brukes like ofte som forkortelsen.

Ordet brukes om to ulike ting, og det sparer mye forvirring å holde dem fra hverandre. Den første betydningen er praksisen, og den handler om mennesker. Martin Fowlers artikkel om emnet, kilden de fleste team ender med å sitere, beskriver den som et sett vaner: hold én hovedlinje (*mainline*), automatiser byggingen, la byggingen teste seg selv, send endringer til hovedlinjen minst daglig, bygg hver push, reparer en ødelagt bygging med en gang, hold byggingen rask, skjul uferdig arbeid i stedet for å holde det på en gren, og la alle se hvordan byggingen står. Artikkelen fører navnet tilbake til Extreme Programming og nevner at Grady Booch brukte uttrykket før det.

Den andre betydningen er maskinen. Når noen sier «CI-en er rød», mener de tjenesten som kjørte byggingen og meldte en feil. GitHub Actions, GitLab CI/CD og lignende tjenester har alle samme form. En fil i repoet beskriver arbeidet: `.github/workflows/*.yml` på GitHub, `.gitlab-ci.yml` i roten på GitLab. En hendelse starter det, f.eks. en push, en pull request eller merge request, en tidsplan eller en person som trykker på en knapp. En runner, altså en maskin satt av til formålet, sjekker ut commiten, kjører kommandoene i filen og melder bestått eller feilet tilbake på commiten.

En server alene er ikke kontinuerlig integrasjon. Et team som holder arbeid på grener i en måned og lar en server bygge hver gren, har automatisert byggingen, og det er nyttig, men integreringen skjer fortsatt på én gang til slutt. Praksisen er

den hyppige sammenslåingen. Maskinen er det som gjør hyppig sammenslåing trygg nok til at teamet fortsetter med den.

Kontinuerlig levering (*continuous delivery*) og kontinuerlig produksjonssetting (*continuous deployment*) er trinnene som kommer etter. Levering holder hver grønn bygging klar til utgivelse, og produksjonssetting gir ut hver av dem automatisk. Begge forutsetter at CI er på plass først, og derfor skrives de tre så ofte sammen som CI/CD.

Bruk begrepet i en beslutning

Bruk «Hva er CI?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Testet, utgitt og mulig å reversere, slik at endringer når kundene en tirsdag ettermiddag i stedet for sent en fredagskveld.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Automatiserte GIS-SaaS-utrulling i GitHub Actions — produksjonssetting, data og rapportering, alt uten manuelle grep

Automatiserte GIS-SaaS-utrulling i GitHub Actions Ta kontakt →

Hvorfor det er viktig

Kostnaden ved å slå sammen to arbeidslinjer vokser raskere enn tiden de er adskilt. To personer som begge endret den samme modulen i går, kan løse konflikten på noen minutter, fordi begge husker hva de mente. To personer som gikk hver sin vei for seks uker siden, må rekonstruere hensikten ut fra diffene. Å slå sammen små biter ofte holder hver konflikt liten, og det er det meste av gevinsten før noen maskin er involvert.

Byggingen ved hver push legger til en ting til: en ren maskin som er enig. «Det virker på min laptop» er en påstand om én laptop, med sine egne installerte biblioteker og sine etterlatte filer. En runner som starter fra en fersk utsjekk hver gang, gjenskaper enten resultatet eller viser hvor laptopen var spesiell.

Farten er der verdien ligger. Fowlers artikkel setter et mål på omtrent ti minutter for hovedbyggingen, og begrunnelsen holder i praksis. En feil som meldes ti minutter etter en push, går tilbake til personen som forårsaket den, mens endringen ennå står på skjermen. En feil som meldes neste morgen, går til den som tilfeldigvis ser, og koster først en time med arkeologi.

Synlighet teller også. Når tilstanden til hovedlinjen vises for alle, starter ingen nytt arbeid oppå en ødelagt bygging uten å vite det, og å reparere den byggingen blir teamets første jobb og ikke noen andres problem.

CI trenger heller ikke å være en tjeneste noen andre driver. pre-commit kjører et prosjekts sjekker som hooks i Git før hver commit eller push, og dokumentasjonen viser det samme settet kjørt på en server med `pre-commit run --all-files`. Task gir hver sjekk et navn i én `Taskfile.yml`, så en person og en runner kaller den samme kommandoen. Dette nettstedets egne sjekker kjører slik: på maskinen nettstedet skrives på, med en full sjekk før hver push og uten noen driftet byggetjeneste imellom.

Slik ser det ut i praksis

Å få programvare ut døren hvilte i ett prosjekt på manuelle steg som hver person gjorde litt annerledes. Automatiserte leveransepipeliner for bygging, testing og produksjonssetting erstattet dem, så veien fra en endring til produksjon ble standardisert i stedet for improvisert. Siden anslår tiden som ble gitt tilbake, til omtrent 4 000 timer, og utgivelsene ble både raskere og mer pålitelige.

I et prosjekt med en database som har levd lenge, er skjemaet beskrevet to ganger: én gang av det første oppsettet og én gang av migreringsrekken fra nummer 036 til 1102, 1 022 migreringer i alt. En CI-jobb bygger databasen begge veier ved hver endring, én gang fra det ferske skjemaet og én gang med alle migreringene spilt av i rekkefølge, og sammenligner strukturen og startdataene. Ethvert avvik får byggingen til å feile med forskjellen skrevet ut, så et nytt miljø og et som har levd lenge, er den samme databasen fordi det er sjekket, og ikke fordi noen antar det.

I et tredje prosjekt genereres API-beskrivelsen og klienten i frontenden i en kjede som starter i databasen og ender i en typet TypeScript-klient på 44 076 linjer. En kontraktskontroll i CI sammenligner det frontenden sender, med det API-et forventer, og får byggingen til å feile når de spriker. Resultatet siden rapporterer, er det CI finnes for: et felt som endres på bare én side, feiler i en bygging minutter etter endringen, i stedet for å nå en bruker som en feil i nettleseren.

Hvor det går galt

De fleste CI-oppsett forfaller i stedet for å bryte sammen. Hvert av disse problemene starter smått og ender med en bygging teamet har lært seg å gå rundt.

Byggingen tar lengre tid enn arbeidet. Når en push bruker førti minutter på å komme tilbake, slutter folk å vente på den, samler flere endringer i én push og sender sjeldnere, som er det motsatte av praksisen. Mål hvor tiden går, før du kjøper mer maskinvare. Cache avhengigheter, kjør de raske sjekkene først, og flytt de trege testsuitene til et andre trinn eller en tidsplan, så hovedbyggingen holder seg nær timinuttersmerket.

Hovedlinjen står rød i et døgn. En ødelagt bygging som ingen reparerer, blir normaltilstanden, og en ny feil kommer ubemerket bak den første. Regelen som hindrer dette, er enkel: den som ødela byggingen, reparerer den nå, eller tilbakestiller endringen og reparerer den på egen tid. En grønn hovedlinje er det alle andre bygger på.

Grener lever i ukevis mens serveren bygger hver av dem. Hver gren viser grønt, og problemene kommer den dagen de slås sammen, alle på en gang. Dette er automatisert bygging uten integrasjonen. Slå sammen til hovedlinjen minst daglig, og hold uferdige funksjoner slått av i stedet for adskilt.

Serveren sjekker én liste og laptopen en annen. To lister med kommandoer som holdes ved like for hånd, glir fra hverandre, og så består en endring lokalt og feiler på serveren, eller omvendt, og til slutt stoler ingen på noen av dem. Gi sjekkene ett inngangspunkt og kall det fra begge steder. Med pre-commit kan serveren kjøre nøyaktig de hookene hver utvikler kjører:

```
# .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 å trykke på «prøv igjen». En test som feiler én kjøring av tjue, av grunner som ikke har noe med endringen å gjøre, lærer teamet å kjøre på nytt i stedet for å lese. Etter det får en ekte feil samme behandling. Sett testen i karantene, reparer den eller slett den, og regn en prøv igjen-knapp som trykkes ofte, som et funn i seg selv.

Les mer

- Continuous Integration — Martin Fowlers lange artikkel: hver praksis, bygging på ti minutter og hvor navnet kom fra.
- Understanding GitHub Actions — arbeidsflyter, hendelser, jobber og runnere slik GitHub definerer dem.
- GitLab CI/CD — filen `.gitlab-ci.yml`, pipeliner, trinn, jobber og runnere med GitLabs egne ord.
- pre-commit — sjekker som erklæres én gang i en konfigurasjonsfil og kjøres som hooks i Git og igjen som et steg på en CI-server.
- Task — en `Taskfile.yml` som gir hver sjekk ett navn, så en person og en runner skriver den samme kommandoen.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er CI?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/ci/