Engineer

CI

att slå ihop minst dagligen, varje gång bevisat av ett bygge

Vad det är

CI är en förkortning av continuous integration, på svenska kontinuerlig integration: ett arbetssätt där alla i ett team lägger in sina ändringar på en och samma gemensamma kodlinje många gånger i veckan, och där en maskin bygger och testar resultatet varje gång, så att ett misstag syns medan den som gjorde det fortfarande minns att det hände. Det fulla namnet används lika ofta som förkortningen, både på engelska och i den svenska formen.

Ordet används om två olika saker, och att hålla isär dem sparar mycket förvirring. Den första är praktiken, som handlar om människor. Artikeln i ämnet av Martin Fowler, den källa som de flesta team till slut citerar, beskriver den som en uppsättning vanor: håll en enda huvudlinje, automatisera bygget, låt bygget testa sig självt, skicka till huvudlinjen minst en gång om dagen, bygg varje push, laga ett trasigt bygge direkt, håll bygget snabbt, göm halvfärdigt arbete i stället för att hålla det på en gren, och låt alla se hur bygget står till. Artikeln spårar namnet till Extreme Programming och noterar att Grady Booch använde uttrycket redan innan dess.

Den andra betydelsen är maskinen. När någon säger ”CI:n är röd” menar de tjänsten som körde bygget och rapporterade ett fel. GitHub Actions, GitLab CI/CD och deras likar har alla samma form. En fil i repot beskriver arbetet: `.github/workflows/*.yml` på GitHub, `.gitlab-ci.yml` i roten på GitLab. En händelse startar det, som en push, en pull request eller merge request, ett tidsschema eller en person som trycker på en knapp. En runner, alltså en maskin som är avsatt för ändamålet, checkar ut committen, kör kommandona i filen och rapporterar godkänt eller underkänt tillbaka mot committen.

En server ensam är ingen kontinuerlig integration. Ett team som håller arbete på grenar i en månad och låter en server bygga varje gren har automatiserat sina byggen, vilket är användbart, men själva integrationen sker fortfarande på en gång i slutet. Praktiken är de täta sammanslagningarna; maskinen är det som gör täta sammanslagningar tillräckligt säkra för att fortsätta med dem.

Kontinuerlig leverans och kontinuerlig driftsättning, continuous delivery och continuous deployment, är stegen som kommer efter. Leverans håller varje grönt bygge redo för release, och driftsättning släpper vart och ett automatiskt. Båda förutsätter att CI redan finns på plats, och därför skrivs de tre så ofta ihop som CI/CD.

Använd begreppet i ett beslut

Använd ”Vad är CI?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Testade, släppta och möjliga att backa, så att ändringar når kunderna en tisdagseftermiddag i stället för sent en fredagskväll.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Automatiserade GIS-driftsättning i GitHub Actions — driftsättning, data och rapportering, helt utan manuellt arbete

Automatiserade GIS-driftsättning i GitHub Actions Hör av er →

Varför det spelar roll

Kostnaden för att slå ihop två arbetslinjer växer snabbare än tiden de tillbringar isär. Två personer som båda ändrade samma modul i går kan reda ut konflikten på några minuter, eftersom båda minns vad de menade. Två personer som gick isär för sex veckor sedan rekonstruerar avsikter ur diffar. Att slå ihop små bitar ofta håller varje konflikt liten, och det är det mesta av nyttan innan någon maskin ens är inblandad.

Bygget vid varje push lägger till en sak till: en ren maskin som håller med. ”Det funkar på min dator” är ett påstående om en enda dator, med dess egna installerade bibliotek och kvarglömda filer. En runner som börjar från en färsk utcheckning varje gång återskapar antingen resultatet eller visar vad som var speciellt med datorn.

Snabbheten är där värdet finns. Artikeln av Fowler sätter ett mål på ungefär tio minuter för huvudbygget, och resonemanget håller i praktiken. Ett fel som rapporteras tio minuter efter en push går tillbaka till den som orsakade det

medan ändringen fortfarande står på skärmen. Ett fel som rapporteras nästa morgon går till den som råkar titta, och kostar först en timmes arkeologi.

Synligheten väger också tungt. När huvudlinjens status visas för alla börjar ingen nytt arbete ovanpå ett trasigt bygge utan att veta om det, och att laga det bygget blir teamets första uppgift i stället för någon annans problem.

CI behöver inte heller betyda en tjänst som någon annan driver. pre-commit kör ett projekts kontroller som git-hookar före varje commit eller push, och dess dokumentation visar samma uppsättning körd på en server med `pre-commit run --all-files`. Task ger varje kontroll ett namn i en enda `Taskfile.yml`, så att en människa och en runner anropar samma kommando. Den här webbplatsens egna kontroller körs på det sättet: på maskinen där webbplatsen skrivs, med en fullständig kontroll före varje push och ingen extern byggtjänst däremellan.

Så ser det ut i praktiken

Tre problem får här sin lösning i CI: manuella releaser, ett databasschema som beskrivs på två sätt och en applikation vars två halvor kan glida isär.

I det första hängde utleveransen av mjukvara på manuella steg som var och en gjorde lite olika. Automatiserade pipelines för bygge, test och driftsättning ersatte dem, så att vägen från en ändring till produktion blev standardiserad i stället för improviserad. Sidan anger att ungefär 4 000 timmar gavs tillbaka, och releaserna blev både snabbare och mer tillförlitliga.

En långlivad databas har här två beskrivningar av samma schema: den initiala uppsättningen och de 1 022 migreringarna ovanpå den, med nummer som börjar på 036 och slutar på 1102. Ett CI-jobb bygger databasen på båda sätten vid varje ändring, en gång från det färska schemat och en gång med varje migrering uppspelad i ordning, och jämför sedan strukturen och grunddatan. Varje avvikelse fäller bygget, med skillnaden utskriven, så en ny miljö och en långlivad miljö är samma databas genom kontroll i stället för genom antagande.

I det tredje genereras API-beskrivningen och frontendens klient i en kedja som börjar i databasen och slutar i en typad TypeScript-klient på 44 076 rader. En kontraktskontroll i CI jämför det frontenden skickar med det API:et förväntar sig och stoppar bygget när de skiljer sig åt. Resultatet som sidan redovisar

är det som CI finns till för: ett fält som ändras på bara ena sidan fäller ett bygge, minuter efter ändringen, i stället för att nå en användare som ett fel i webbläsaren.

Var det går fel

De flesta CI-uppsättningar förfaller snarare än går sönder. Var och en av de här börjar smått och slutar med ett bygge som teamet har lärt sig att arbeta runt.

Bygget tar längre tid än arbetet. När en push tar fyrtio minuter att komma tillbaka slutar folk vänta på den, samlar flera ändringar i en push och pushar mer sällan, vilket är motsatsen till praktiken. Mät var tiden går innan du lägger till hårdvara. Cacha beroenden, kör de snabba kontrollerna först och flytta de långsamma testsviterna till ett andra steg eller till en schemalagd körning, så att huvudbygget håller sig nära tio minuter.

Huvudlinjen förblir röd i ett dygn. Ett trasigt bygge som ingen lagar blir det normala, och ett andra fel kommer obemärkt bakom det första. Regeln som hindrar det är enkel: den som bröt bygget lagar det nu, eller backar ändringen och lagar den på egen tid. En grön huvudlinje är det som alla andra bygger vidare på.

Grenar lever i veckor medan servern bygger var och en. Varje gren visar grönt, och problemen kommer på dagen då de slås ihop, allihop på en gång. Det här är automatiserade byggen utan själva integrationen. Slå ihop till huvudlinjen minst en gång om dagen, och håll ofärdiga funktioner avstängda i stället för isärhållna.

Servern kontrollerar en lista och datorn en annan. Två listor med kommandon som underhålls för hand glider isär, och sedan går en ändring igenom lokalt och fallerar på servern, eller tvärtom, och ingen litar på någon av dem. Ge kontrollerna en enda ingång och anropa den från båda ställena. Med pre-commit kan servern köra exakt de hookar som varje utvecklare kör:

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

Ett instabilt test lär alla att trycka på kör igen. Ett test som fallerar en körning av tjugo, av skäl som inte har med ändringen att göra, tränar teamet att köra om i stället för att läsa. Efter det får ett verkligt fel samma behandling. Sätt testet i karantän, laga det eller radera det, och behandla en knapp för omkörning som trycks ofta som ett fynd i sig.

Läs mer

- Continuous Integration — den långa artikeln av Martin Fowler: varje praktik, tiominutersbygget och var namnet kom ifrån.
- Understanding GitHub Actions — arbetsflöden, händelser, jobb och runners så som GitHub definierar dem.
- GitLab CI/CD — filen `.gitlab-ci.yml`, pipelines, steg, jobb och runners med de begrepp som GitLab själv använder.
- pre-commit — kontroller som deklareras en gång i en konfigurationsfil, körs som git-hookar och en gång till som ett steg på en CI-server.
- Task — en `Taskfile.yml` som ger varje kontroll ett namn, så att en människa och en runner skriver samma kommando.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är CI?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/ci/