Engineer

CI

Änderungen täglich integrieren, automatisch bauen und testen

Was es ist

CI steht für Continuous Integration, auf Deutsch kontinuierliche Integration: eine Arbeitsweise, bei der alle Teammitglieder ihre Änderungen mehrmals pro Woche in einen gemeinsamen Entwicklungsstand einbringen. Eine Maschine baut und testet jedes Mal das Ergebnis, damit ein Fehler auffällt, solange sich die Person, die ihn verursacht hat, noch daran erinnert. Die ausgeschriebene Bezeichnung Continuous Integration wird ebenso häufig verwendet wie die Abkürzung.

Das Wort bezeichnet zwei verschiedene Dinge. Sie auseinanderzuhalten erspart viel Verwirrung. Das erste ist die Praxis, bei der es um Menschen geht. Martin Fowlers Artikel dazu, die letztlich von den meisten Teams zitierte Quelle, beschreibt eine Reihe von Gewohnheiten: mit einem einzigen Hauptentwicklungszweig arbeiten, den Build automatisieren, den Build sich selbst testen lassen, mindestens täglich in den Hauptzweig pushen, jeden Push bauen, einen defekten Build sofort reparieren, den Build schnell halten, unfertige Arbeit verbergen statt auf einem eigenen Branch zu halten und den Buildstatus für alle sichtbar machen. Der Artikel führt den Namen auf Extreme Programming zurück und merkt an, dass Grady Booch den Ausdruck schon vorher verwendete.

Die zweite Bedeutung ist die Maschine. Sagt jemand „die CI ist rot“, meint die Person den Dienst, der den Build ausgeführt und einen Fehler gemeldet hat. GitHub Actions, GitLab CI/CD und vergleichbare Dienste folgen demselben Muster. Eine Datei im Repository beschreibt die Arbeit: `.github/workflows/*.yml` bei GitHub, `.gitlab-ci.yml` im Wurzelverzeichnis bei GitLab. Ein Ereignis startet sie, etwa ein Push, ein Pull Request oder Merge Request, ein Zeitplan oder eine Person, die eine Schaltfläche betätigt. Ein Runner, eine dafür vorgesehene Maschine, checkt den Commit aus, führt die Befehle aus der Datei aus und meldet Erfolg oder Fehlschlag für diesen Commit zurück.

Ein Server allein ist keine kontinuierliche Integration. Ein Team, das Arbeit einen Monat lang auf Branches hält und jeden Branch von einem Server bauen lässt, hat seine Builds automatisiert. Das ist nützlich, doch die Integration geschieht weiterhin am Ende auf einmal. Die Praxis ist das häufige Zusammenführen. Die Maschine macht es sicher genug, das dauerhaft zu tun.

Continuous Delivery und Continuous Deployment sind die nachfolgenden Schritte. Delivery hält jeden grünen Build zur Veröffentlichung bereit. Deployment veröffentlicht jeden davon automatisch. Beide setzen CI voraus. Deshalb werden die drei so oft gemeinsam als CI/CD geschrieben.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist CI?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Geprüft, veröffentlicht und rückgängig zu machen: Änderungen erreichen Ihre Kunden am Dienstagnachmittag statt spät am Freitagabend.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. GIS-SaaS-Deployments mit GitHub automatisiert — Bereitstellung, Daten und Berichte ohne Handarbeit

GIS-SaaS-Deployments mit GitHub automatisiert Kontakt aufnehmen →

Warum es wichtig ist

Die Kosten für das Zusammenführen zweier Arbeitsstränge wachsen überproportional zur Dauer ihrer Trennung. Zwei Personen, die gestern dasselbe Modul geändert haben, können den Konflikt in wenigen Minuten klären, weil sich beide an ihre Absicht erinnern. Zwei Personen, deren Arbeit vor sechs Wochen auseinanderging, rekonstruieren die Absicht aus Diffs. Kleine Stücke häufig zusammenzuführen hält jeden Konflikt klein. Darin liegt der größte Teil des Nutzens, noch bevor eine Maschine beteiligt ist.

Der Build bei jedem Push ergänzt etwas Zweites: eine saubere Maschine, die das Ergebnis bestätigt. „Auf meinem Laptop funktioniert es“ ist eine Aussage über einen Laptop mit seinen installierten Bibliotheken und zurückgebliebenen Dateien. Ein Runner, der jedes Mal mit einem frischen Checkout beginnt, reproduziert entweder das Ergebnis oder zeigt, was den Laptop zum Sonderfall machte.

Der Wert liegt in der Geschwindigkeit. Fowlers Artikel setzt etwa zehn Minuten als Ziel für den Hauptbuild, und die Begründung bewährt sich in der Praxis. Ein zehn Minuten nach dem Push gemeldeter Fehler erreicht die verursachende Person, während die Änderung noch auf ihrem Bildschirm steht. Ein erst am nächsten Morgen gemeldeter Fehler erreicht die Person, die gerade hinsieht, und kostet zunächst eine Stunde Spurensuche.

Auch Sichtbarkeit zählt. Ist der Zustand des Hauptzweigs für alle sichtbar, beginnt niemand unwissentlich neue Arbeit auf einem defekten Build. Ihn zu reparieren wird zur ersten Aufgabe des Teams statt zum Problem anderer.

CI muss auch keinen gehosteten Dienst bedeuten. pre-commit führt die Prüfungen eines Projekts als Git-Hooks vor jedem Commit oder Push aus. Seine Dokumentation zeigt dieselbe Auswahl auf einem Server mit `pre-commit run --all-files`. Task gibt jeder Prüfung einen Namen in einer `Taskfile.yml`, sodass ein Mensch und ein Runner denselben Befehl aufrufen. Die Prüfungen hinter dieser Website folgen diesem Modell. Sie laufen lokal dort, wo die Seiten geschrieben werden. Vor jedem Push läuft eine vollständige Prüfung, ohne gehosteten Builddienst dazwischen.

Wie es in der Praxis aussieht

Rund 4.000 Stunden wurden frei, als ein Team nicht mehr von Hand veröffentlichte. Die Softwareauslieferung hatte auf manuellen Schritten beruht, die jede Person etwas anders ausführte, bis automatisierte Builds, Tests und Bereitstellungen sie ersetzten. Der Weg von einer Änderung in Produktion wurde standardisiert statt improvisiert. Die Veröffentlichungen wurden schneller und zuverlässiger.

In einem Projekt mit einer lange bestehenden Datenbank wird das Schema zweimal beschrieben: durch die Ersteinrichtung und durch 1.022 Migrationen mit den Nummern 036 bis 1102. Ein CI-Job baut die Datenbank bei jeder Änderung auf beiden Wegen: einmal aus dem frischen Schema und einmal durch geordnetes Wiederholen jeder Migration. Er vergleicht Struktur und vorbelegte Daten. Unterscheiden sich die beiden Aufbauten, wird der Job rot und gibt die Unterschiede aus. Dass eine frische und eine lange bestehende Umgebung dieselbe Datenbank ergeben, ist damit geprüft statt angenommen.

CI kann auch zwei Hälften einer Anwendung in Übereinstimmung halten. In einem anderen Projekt entstehen API-Beschreibung und Frontend-Client in einer Generierungskette, die bei der Datenbank beginnt. Allein der Client umfasst 44.076 Zeilen typisierten TypeScript-Codes. Eine CI-Prüfung vergleicht, was das Frontend sendet, mit dem, was die API erwartet, und lässt den Build bei Abweichungen fehlschlagen. Dafür gibt es CI: Ein nur auf einer Seite geändertes Feld fällt Minuten nach der Änderung im Build auf, statt als Browserfehler einen Nutzer zu erreichen.

Wo es schiefgeht

Die meisten CI-Systeme werden allmählich schlechter, statt plötzlich auszufallen. Jeder dieser Fälle beginnt klein und endet bei einem Build, den das Team zu umgehen gelernt hat.

Der Build dauert länger als die Arbeit. Braucht ein Push vierzig Minuten bis zur Rückmeldung, warten Menschen nicht mehr darauf. Sie bündeln mehrere Änderungen in einem Push und pushen seltener, das Gegenteil der Praxis. Messen Sie den Zeitverbrauch, bevor Sie Hardware ergänzen. Cachen Sie Abhängigkeiten, führen Sie schnelle Prüfungen zuerst aus und verlegen Sie langsame Testsammlungen in eine zweite Stufe oder einen zeitgesteuerten Lauf, damit der Hauptbuild nahe bei zehn Minuten bleibt.

Der Hauptzweig bleibt einen Tag rot. Ein von niemandem reparierter defekter Build wird zum Normalzustand. Ein zweiter Fehler kommt unbemerkt hinter dem ersten hinzu. Die Gegenregel ist klar: Wer ihn kaputtgemacht hat, repariert ihn jetzt oder nimmt die Änderung zurück und repariert sie in der eigenen Zeit. Auf einem grünen Hauptzweig bauen alle anderen auf.

Branches leben wochenlang, während der Server jeden einzeln baut. Jeder Branch ist grün. Am Tag des Zusammenführens kommen alle Probleme zugleich. Das ist automatisiertes Bauen ohne Integration. Führen Sie Änderungen mindestens täglich im Hauptzweig zusammen und halten Sie unfertige Funktionen ausgeschaltet statt getrennt.

Server und Laptop prüfen verschiedene Listen. Zwei von Hand gepflegte Befehlslisten entwickeln sich auseinander. Eine Änderung besteht dann lokal und scheitert auf dem Server oder umgekehrt, und niemand vertraut noch einer der

beiden Prüfungen. Geben Sie den Prüfungen einen Einstiegspunkt und rufen Sie ihn an beiden Orten auf. Mit pre-commit kann der Server genau die Hooks ausführen, die jeder Entwickler ausführt:

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

Ein instabiler Test lehrt alle, erneut zu starten. Ein Test, der bei einem von zwanzig Durchläufen aus Gründen scheitert, die nichts mit der Änderung zu tun haben, gewöhnt das Team ans Wiederholen statt ans Lesen. Ein echter Fehler wird danach genauso behandelt. Isolieren Sie den Test, reparieren oder löschen Sie ihn und behandeln Sie einen häufig gedrückten Wiederholungsbutton als eigenständigen Befund.

Weiterführende Lektüre

- Continuous Integration — Martin Fowlers ausführlicher Artikel: jede Gewohnheit, der Zehn-Minuten-Build und die Herkunft des Namens.
- GitHub Actions verstehen — Workflows, Ereignisse, Jobs und Runner nach GitHubs Definition.
- GitLab CI/CD — die Datei `.gitlab-ci.yml`, Pipelines, Stufen, Jobs und Runner in GitLabs eigenen Begriffen.
- pre-commit — einmal in einer Konfigurationsdatei festgelegte Prüfungen, die als Git-Hooks und nochmals als Schritt auf einem CI-Server laufen.
- Task — eine `Taskfile.yml`, die jeder Prüfung einen Namen gibt, damit Mensch und Runner denselben Befehl eingeben.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist CI?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/ci/