Engineer

CI

intégrer chaque jour, avec compilation et tests automatiques

Ce que c’est

CI abrège continuous integration, ou intégration continue : une façon de travailler où tous les membres d’une équipe intègrent leurs changements à une même ligne de développement partagée plusieurs fois par semaine. Une machine compile et teste le résultat à chaque fois, pour qu’une erreur apparaisse tant que son auteur se souvient encore de l’avoir introduite. Le nom complet, intégration continue, s’emploie aussi souvent que l’abréviation.

Le terme désigne deux choses différentes, qu’il vaut mieux distinguer pour éviter bien des confusions. La première est la pratique, qui concerne les personnes. L’article de Martin Fowler, source que la plupart des équipes finissent par citer, la décrit comme un ensemble d’habitudes : garder une ligne principale unique, automatiser la compilation, lui faire exécuter ses propres tests, pousser vers la ligne principale au moins chaque jour, compiler chaque envoi, réparer immédiatement une compilation cassée, conserver une compilation rapide, masquer le travail inachevé plutôt que le garder sur une branche et rendre l’état de la compilation visible à tous. L’article rattache le nom à l’Extreme Programming et précise que Grady Booch employait déjà l’expression auparavant.

Le second sens est la machine. Dire « la CI est rouge » désigne le service qui a exécuté la compilation et signalé un échec. GitHub Actions, GitLab CI/CD et leurs équivalents partagent le même schéma. Un fichier du dépôt décrit le travail : `.github/workflows/*.yml` sur GitHub et `.gitlab-ci.yml` à la racine sur GitLab. Un événement le déclenche : push, pull request ou demande de fusion, déclenchement planifié, ou personne appuyant sur un bouton. Un runner, machine réservée à cet usage, extrait le commit, exécute les commandes du fichier et associe au commit un résultat de réussite ou d’échec.

Un serveur seul ne constitue pas de l'intégration continue. Une équipe qui garde son travail un mois sur des branches et fait compiler chaque branche par un serveur a automatisé ses compilations, ce qui est utile, mais l'intégration se produit toujours d'un coup à la fin. La pratique est la fusion fréquente ; la machine la rend assez sûre pour qu'on puisse continuer à la pratiquer.

La livraison continue et le déploiement continu sont les étapes suivantes. La livraison garde chaque compilation réussie prête à être publiée, et le déploiement publie automatiquement chacune d'elles. Les deux exigent une CI déjà en place, d'où l'association fréquente des trois sous la forme CI/CD.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que la CI ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des changements testés, publiés et réversibles, pour livrer aux clients un mardi après-midi plutôt qu'un vendredi soir tard.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Déploiement SaaS GIS automatisé : GitHub Actions — déploiement, données et reporting automatisés

Déploiement SaaS GIS automatisé : GitHub Actions Contactez-nous →

Pourquoi c'est important

Le coût de fusion de deux lignes de développement augmente plus vite que le temps passé à les maintenir séparées. Deux personnes ayant modifié le même module hier peuvent résoudre leur conflit en quelques minutes, car elles se souviennent de leur intention. Deux personnes dont les travaux ont divergé six semaines plus tôt doivent reconstruire les intentions à partir des différences de code. Fusionner souvent de petits changements garde chaque conflit petit. C'est l'essentiel du bénéfice, avant même l'intervention d'une machine.

La compilation à chaque push ajoute un second apport : une machine propre qui confirme le résultat. « Cela fonctionne sur mon portable » est une affirmation sur un portable particulier, avec ses bibliothèques installées et ses fichiers résiduels. Un runner qui repart chaque fois d'une copie de travail propre reproduit le résultat ou révèle ce que le portable avait de particulier.

La valeur tient à la vitesse. Fowler fixe un objectif d'environ dix minutes pour la compilation principale, et ce raisonnement se vérifie en pratique. Un échec signalé dix minutes après un push revient à la personne qui l'a causé alors que le

changement est encore à l'écran. Un échec signalé le lendemain matin revient à la personne qui le découvre par hasard et exige d'abord une heure d'archéologie.

La visibilité compte aussi. Quand l'état de la ligne principale est visible de tous, personne ne commence un nouveau travail sur une compilation cassée sans le savoir. La réparation devient la première tâche de l'équipe plutôt que le problème de quelqu'un d'autre.

La CI n'implique pas non plus un service hébergé. pre-commit exécute les contrôles du projet comme hooks Git avant chaque commit ou push. Sa documentation montre les mêmes contrôles exécutés sur serveur avec `pre-commit run --all-files`. Task nomme chaque contrôle dans un seul `Taskfile.yml` pour qu'une personne et un runner appellent la même commande. Les contrôles de ce site suivent ce modèle. Ils s'exécutent localement, là où les pages sont écrites, et une vérification complète précède chaque push, sans service de compilation hébergé intermédiaire.

À quoi cela ressemble en pratique

Environ 4 000 heures ont été rendues à une équipe lorsqu'elle a cessé de publier à la main. La livraison logicielle reposait sur des étapes manuelles que chacun exécutait un peu différemment, jusqu'à ce que la compilation, les tests et le déploiement automatisés les remplacent. Le chemin du changement à la production est devenu standard plutôt qu'improvisé, et les publications sont devenues à la fois plus rapides et plus fiables.

Sur un projet avec une base de données en service depuis longtemps, le schéma est décrit deux fois : par l'installation initiale et par 1 022 migrations numérotées de 036 à 1102. Une tâche CI construit la base des deux façons à chaque changement, une fois depuis le schéma neuf et une fois en rejouant toutes les migrations dans l'ordre, puis compare la structure et les données initiales. Si les deux constructions diffèrent, la tâche passe au rouge et affiche les différences. Un environnement neuf et un environnement ancien correspondent ainsi à la même base par vérification, plutôt que par hypothèse.

La CI peut aussi maintenir la cohérence de deux parties d'une application. Sur un autre projet, la description de l'API et le client du frontend sont générés par une chaîne qui commence à la base de données. Le client seul représente 44 076 lignes de TypeScript typé. Un contrôle CI confronte ce que le frontend envoie à ce que l'API attend et fait échouer la compilation en cas de divergence. C'est le

résultat pour lequel la CI existe : un champ modifié d'un seul côté fait échouer la compilation quelques minutes après le changement, au lieu de parvenir à l'utilisateur sous forme d'erreur dans le navigateur.

Où les problèmes surviennent

La plupart des configurations CI se dégradent plutôt qu'elles ne cassent. Chacun des cas suivants commence petit et finit par une compilation que l'équipe a appris à contourner.

La compilation dure plus longtemps que le travail. Dès qu'un push met quarante minutes à rendre son résultat, les personnes cessent d'attendre, regroupent plusieurs changements dans un push et poussent moins souvent : c'est l'inverse de la pratique. Mesurez où passe le temps avant d'ajouter du matériel. Mettez les dépendances en cache, exécutez d'abord les contrôles rapides et déplacez les suites lentes vers une seconde étape ou une exécution planifiée pour garder la compilation principale proche de dix minutes.

La ligne principale reste rouge toute une journée. Une compilation cassée que personne ne répare devient l'état normal, et un second échec arrive inaperçu derrière le premier. La règle est simple : la personne qui l'a cassée la répare maintenant, ou annule le changement et le corrige ensuite de son côté. Une ligne principale verte est la base sur laquelle tous les autres travaillent.

Les branches vivent des semaines pendant que le serveur compile chacune. Toutes sont vertes, et les problèmes arrivent d'un coup le jour de leur fusion. C'est de la compilation automatisée sans intégration. Fusionnez vers la ligne principale au moins chaque jour, et désactivez les fonctions inachevées au lieu de les garder séparées.

Le serveur contrôle une liste et le portable une autre. Deux listes de commandes maintenues à la main divergent. Un changement passe alors localement et échoue sur le serveur, ou l'inverse, et personne ne fait confiance à l'un ou à l'autre. Donnez un point d'entrée unique aux contrôles et appelez-le aux deux endroits. Avec pre-commit, le serveur peut exécuter exactement les hooks de chaque développeur :

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

Un test instable apprend à tous à cliquer sur Réessayer. Un test qui échoue une fois sur vingt pour une raison sans lien avec le changement apprend à l'équipe à relancer plutôt qu'à lire. Un véritable échec reçoit ensuite le même traitement. Isolez le test, réparez-le ou supprimez-le, et traitez un bouton de relance souvent utilisé comme un constat à part entière.

Pour aller plus loin

- Intégration continue — le long article de Martin Fowler : chaque pratique, la compilation en dix minutes et l'origine du nom.
- Comprendre GitHub Actions — workflows, événements, tâches et runners selon les définitions de GitHub.
- GitLab CI/CD — le fichier `.gitlab-ci.yml` , les pipelines, étapes, tâches et runners dans les termes de GitLab.
- pre-commit — des contrôles déclarés une fois dans un fichier, exécutés comme hooks Git, puis comme étape sur un serveur CI.
- Task — un `Taskfile.yml` donnant un nom à chaque contrôle, pour qu'une personne et un runner saisissent la même commande.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que la CI? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/ci/