

# Intégration et livraison continues

# Plan

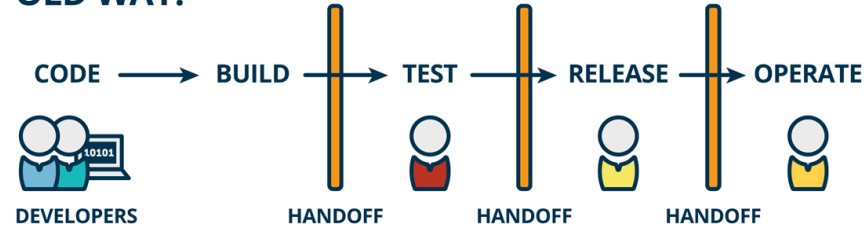
---

1. Introduction
2. Intégration continue
  - 2.1. CI Provider
  - 2.2. GitHub Actions
    - 2.2.1. Workflows
      - 2.2.1.1. Basique
      - 2.2.1.2. Utilisation de matrices
      - 2.2.1.3. Utilisation des cronjobs
      - 2.2.1.4. Documentation
    - 2.2.2. Runners
    - 2.2.3. Exercices
  - 2.3. Flow de CI
    - 2.3.1. Utilisation de PRs
3. Livraison continue
  - 3.1. Flow de CD
  - 3.2. PaaS et IaaS
  - 3.3. Exercice
4. Ressources

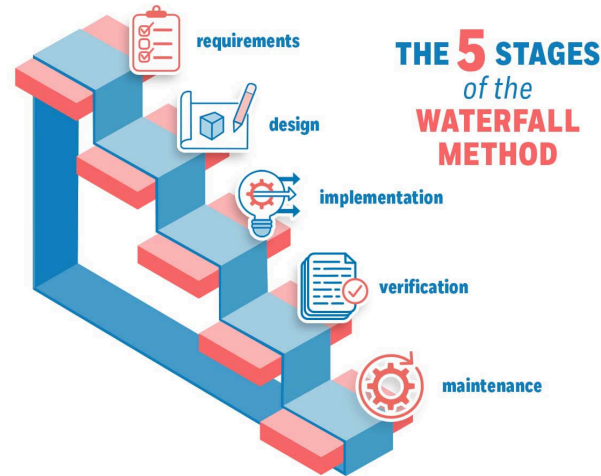
# Introduction

Il y a fort, fort longtemps...l'intégration et la livraison d'un software étaient assignées à des équipes dédiées.

## OLD WAY:



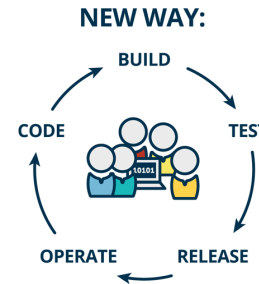
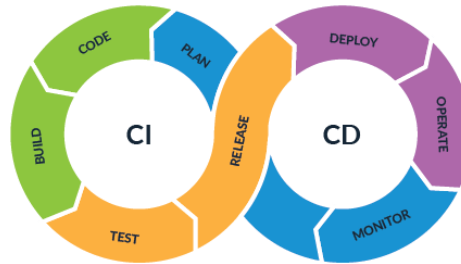
Cette approche convenait parfaitement aux logiciels critiques (avions, centrales nucléaires, armements, etc.) qui suivent un cycle de développement en **cascade**.



Cette approche était moins adaptée aux entreprises agiles (startups) qui souhaitent:

- **accélérer le time-to-market** (le temps de développement et de mise en production d'une fonctionnalité)
- **accroître la stabilité** en permettant la reproductibilité du processus de livraison
- assurer une **continuité de service** des applications

De ces besoins découlent les pratiques DevOps connues sous le nom de **CI/CD**.



# Intégration continue

---

L'intégration continue fait référence à plusieurs pratiques:

- Construire une version fonctionnelle du système chaque jour
- Exécuter les tests tous les jours
- Committer ses changements régulièrement sur un **dépôt partagé** (Exemple: Git)
- Un système qui observe les changements sur le dépôt et si il détecte un changement:
  - Récupère une copie du logiciel depuis le dépôt
  - **Compile et exécute les tests**
  - Si les tests passent, possibilité de créer une nouvelle release du logiciel
  - Sinon averti le développeur concerné

Une série d'étapes à réaliser lors de la CI se nomme **pipeline**:



Mais comment et où faire tourner ces pipelines?

# CI Provider

Il existe de nombreuses plate-formes d'intégration continue:



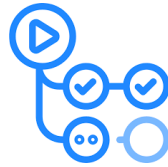
CircleCI



TravisCI



Netlify BUILD



GitHub Actions



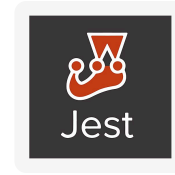
Gitlab CI/CD



# GitHub Actions

Pour illustrer ce cours, nous utiliserons l'outil GitHub Actions sur un projet d'application Web.

**À vous de jouer:** Faites un fork du projet <https://github.com/isfates-l3-outils-et-methodes-de-dev/webapp> et clonez-le.



# Workflows

- L'automatisation des tâches d'une CI se fait via des **workflows**.
- Un workflow est un processus automatisé configurable qui exécutera un ou plusieurs travaux (**jobs**).
- Les workflows sont définis par un fichier YAML archivé dans votre dépôt.

```
.github
├── workflows
│   ├── ci-pipeline.yaml
│   ├── release.yaml
│   └── label.yaml
```

Un workflow doit contenir les 3 composants de base suivants:

1. Un ou plusieurs **événements** qui déclenchent le workflow.
2. Un ou plusieurs **jobs**, dont chacun s'exécute sur un ordinateur exécutateur et exécute une série d'une ou plusieurs étapes (**steps**).
3. Chaque étape peut exécuter un script que vous définissez ou une action, qui est une extension réutilisable qui peut simplifier votre workflow.

## Basique

```
name: Linting, Testing, Building
on: push

jobs:
  Linting:
    runs-on: ubuntu-latest
    steps:
      - name: Get Code
        uses: actions/checkout@v3
      - name: Setup environment
        uses: actions/setup-node@v4
      - name: Run Linting Rules
        run: exec lint

  Testing:
    needs: Linting
    runs-on: ubuntu-latest
    steps:
      # ...
      - name: Run Unit Tests
        run: exec test
```

- **name** : Le nom du workflow. GitHub affiche les noms de vos workflows sous l'onglet **Actions** de votre dépôt.
- **on** : Définit des événements uniques ou multiples qui peuvent déclencher le workflow ou définir un horaire (cronjob).
- **runs-on** : Définit le type de machine sur laquelle le job doit être exécuté.
- **uses** : Sélectionne une action à exécuter dans le cadre d'une step.
  - Une action est une unité de code réutilisable.
  - Vous pouvez utiliser une action définie dans le même dépôt, dans un dépôt public ou dans une image Docker publiée.

## Introduction

The screenshot displays the GitHub Actions interface for a workflow named 'tetest #8'. The top navigation bar includes links for Code, Issues, Pull requests, Actions (selected), Projects, Wiki, Security, Insights, and Settings. The workflow is categorized under 'Linting, Testing, Building'. The left sidebar shows the 'Summary' tab selected, with a list of jobs: Linting, Testing, and Building, all marked as successful. Below the jobs list are links for 'Run details', 'Usage', and 'Workflow file'. The main content area shows the workflow was triggered via a push 15 hours ago by user 'johannchopin' on the 'main' branch. The status is 'Success' with a total duration of '3m 26s'. Below this, the workflow file 'ci-pipeline.yaml' is shown with the trigger 'on: push'. A visual pipeline diagram shows two steps: 'Linting' (2m 32s) and 'Testing' (17s), both completed successfully.

Code Issues Pull requests **Actions** Projects Wiki Security Insights Settings

← Linting, Testing, Building

✓ **tetest #8**

**Summary**

Jobs

- ✓ Linting
- ✓ Testing
- ✓ Building

Run details

- Usage
- Workflow file

Triggered via push 15 hours ago

✖️ johannchopin pushed → fdf8923 **main**

Status	Total duration	Artifacts
<b>Success</b>	<b>3m 26s</b>	—

**ci-pipeline.yaml**  
on: push

✓ **Linting** 2m 32s — ✓ **Testing** 17s

# Utilisation de matrices

Une stratégie matricielle vous permet d'utiliser des variables lors de la définition d'un job afin de créer automatiquement plusieurs jobs basées sur les combinaisons des variables.

```
jobs:
  example_matrix:
    strategy:
      matrix:
        os: [ubuntu-22.04, ubuntu-20.04]
        nodeVersion: [10, 12, 14]
    runs-on: ${ matrix.os }
    steps:
      - uses: actions/setup-node@v4
        with:
          node-version: ${ matrix.nodeVersion }
```

## Utilisation des cronjobs


Vous pouvez utiliser `on.schedule` pour définir un calendrier pour vos workflow en le programmant à l'aide de la syntaxe **POSIX cron**:

```
on:  
  schedule:  
    # déclenche le workflow tous les jours à 5:30 et 17:30 UTC  
    - cron: '30 5,17 * * *'
```

💡 GitHub Actions accepte un cronjob de maximum 5 minutes d'intervalle.

# Documentation

---

 <https://docs.github.com/fr/actions/writing-workflows/workflow-syntax-for-github-actions>



# Runners

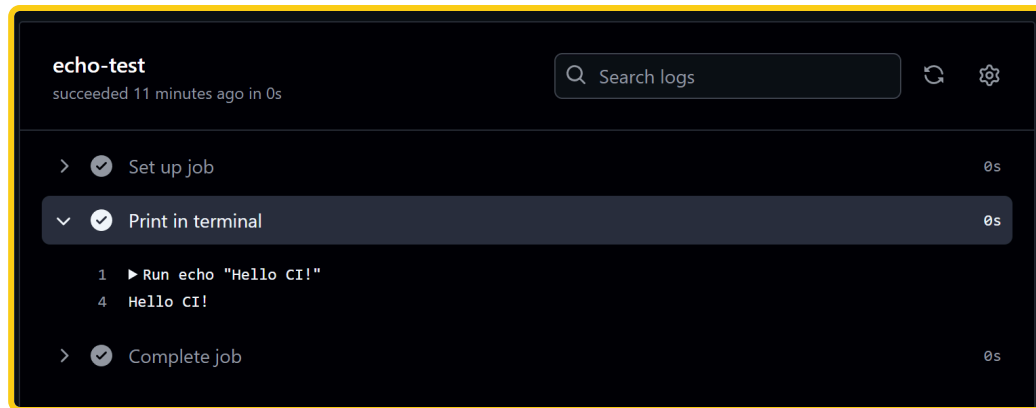
- ⚠ Les jobs sur les GitHub-hosted runners sont indépendants, ils démarrent donc dans une image de runner propre et doivent télécharger les dépendances à chaque fois.
- L'utilisation de ces runners sur des dépôts publics est gratuite et illimitée.

Virtual Machine	Processor (CPU)	Memory (RAM)	Storage (SSD)	Workflow label
Linux	4	16 GB	14 GB	<a href="#">ubuntu-latest</a> , <a href="#">ubuntu-24.04</a> , <a href="#">ubuntu-22.04</a> , <a href="#">ubuntu-20.04</a>
Windows	4	16 GB	14 GB	<a href="#">windows-latest</a> , <a href="#">windows-2022</a> , <a href="#">windows-2019</a>
macOS	3	14 GB	14 GB	<a href="#">macos-12</a>
macOS	4	14 GB	14 GB	<a href="#">macos-13</a>
macOS	3 (M1)	7 GB	14 GB	<a href="#">macos-latest</a> , <a href="#">macos-14</a> , <a href="#">macos-15</a> [Beta]

## Exercices

À vous de jouer: Ajoutez dans votre projet **webapp** un workflow nommé `Hello CI` :

- Celui-ci sera exécuté **lors de chaque push** de code sur le dépôt.
- Il aura un unique job basé sur la dernière version du runner `ubuntu` qui exécutera un simple `echo "Hello CI!"`.
- Inspectez le résultat de votre Action de sorte à avoir un résultat similaire:



## Introduction

#/ Correction

#

Modifiez maintenant ce workflow afin de lancer à chaque `push` les jobs `Lint`, `Test` et `Build`.

Chaque job devra:

- Récupérer le code du repo ( `action/checkout` )
- Setup l'environnement nodejs ( `action/setup-node` )
- Installer les dependencies
- Exécuter la commande appropriée

Utilisez cette steps pour installer les dependencies:

```
- name: Load & Cache Dependencies  
  uses: ./github/actions/cached-deps
```

## Introduction

#/ Correction

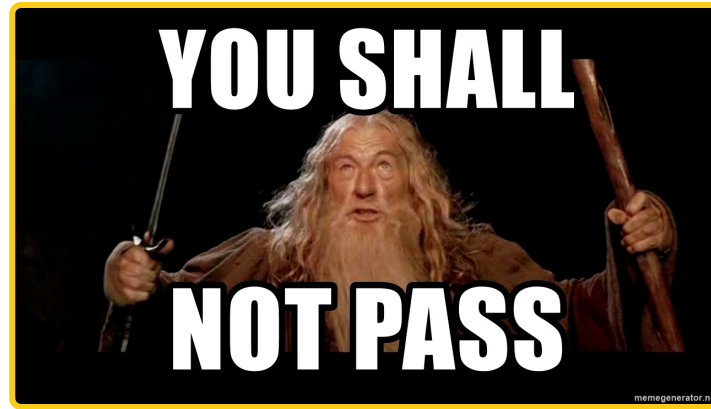
#

# Flow de CI

---

En l'état, la CI ne nous alerte qu'en cas de besoin et ne bloque pas nos actions.

Comment s'assurer que le code envoyé au dépôt est valide?



## Utilisation de PRs

Il est très simple avec les PRs de sécuriser la qualité du code.

Voici un exemple dans un flow de trunk-based development:

```
Error: Cannot merge branch  
'main' into itself.
```

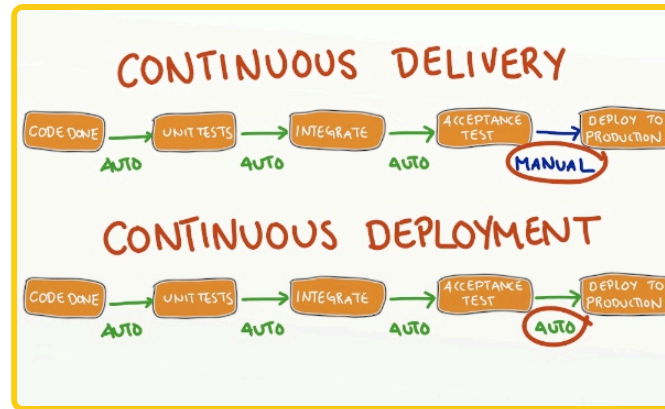
- La branch trunk est sécurisée et ne peut recevoir de commits en dehors d'une PR
- Chaque PRs doivent exécuter la CI
  - soit à chaque push
  - soit lors d'une action manuelle
- Une PR ne peut être mergée dans la branch trunk sans avoir une CI validée

# Livraison continue

La livraison continue est la suite logique de l'intégration continue.

Il s'agit d'une discipline où l'application est construite de manière à pouvoir être mise en production à n'importe quel moment.

Livraison continue  $\neq$  déploiement continu





# Flow de CD

---

- **Une simple action manuelle**
  - Une interface avec un bouton de mise en production
  - Une commande à lancer sur un serveur
- **Une automation sur le dépôt**
  - Lors d'un merge sur une branche spécifique
  - Lors de la création d'un nouveau tag
  - Lors d'une Release (exemple de GitHub Releases)

# PaaS et IaaS

---

La livraison continue se met généralement en place à l'aide d'une **Platform as a Service** ou même une **Infrastructure as a Service**.

- **Rentabilité:** Pas besoin d'acheter du matériel ou de payer des frais pendant les temps d'arrêt
- **Gain de temps:** Pas besoin de passer du temps à mettre en place/maintenir la pile de base
- **Rapidité de mise sur le marché:** Accélération de la création d'applications
- **Future-Proof:** Accès à un datacenter et à des systèmes d'exploitation toujours à jours
- **Amélioration de la sécurité:** Les fournisseurs de PaaS investissent massivement dans la technologie et l'expertise en matière de sécurité.
- **Évolution dynamique:** Augmentation rapide de la capacité en période de pointe
- **Flexibilité:** Permet aux employés de se connecter et de travailler sur les applications depuis n'importe où.

## Exemples de PaaS:



## Exemples de IaaS:



# Exercice

**À vous de jouer:** Nous allons dans un premier temps isoler le workflow `Build` dans un fichier dédié de sorte à pouvoir facilement le réutiliser.

1. Isoler le workflow dans un fichier `build.yaml`.
2. Il s'exécutera lors d'un simple appel ( `workflow_call` ).
3. Il devra rendre accessible le contenu du dossier généré `dist` aux autres workflow (regardez voir l'action `actions/upload-artifact` ).
4. Réutilisez le workflow dans votre CI.

## Introduction

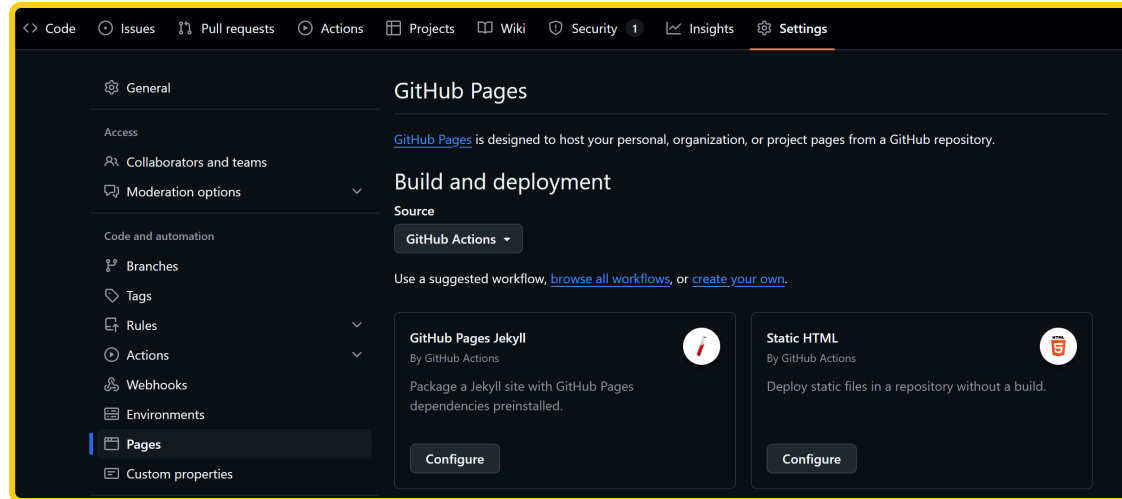
#/ Correction

#

## Introduction

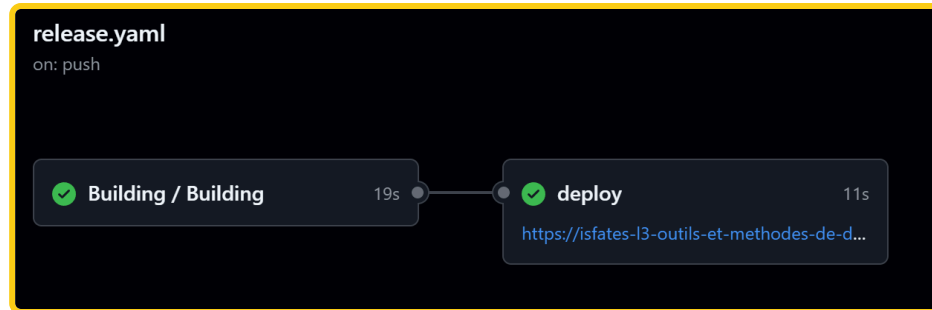
Ajoutons maintenant un Workflow de **Release** qui déploiera automatiquement notre Web Application.

Pour ce faire, nous utiliserons le service d'hébergement statique **GitHub Pages** et sa GitHub Actions dédiée:



**À vous de jouer:** Ajoutez un nouveau workflow `Release` :

1. Il devra s'exécuter lors de la création d'un tag suivant la syntaxe SemVer ( `v[0-9]+.[0-9]+.[0-9]+` )
2. Il sera composé de 2 jobs `Build` et `Deploy`
3. Le job `Deploy`
  1. fera appel à l'action `actions/download-artifact` pour récupérer le dossier `dist`
  2. enverra son contenu à l'aide de l'action `actions/upload-pages-artifact` et
  3. déploiera le site statique à l'aide de `actions/deploy-page`



## Introduction

#/ Correction

#



# Ressources

---

Ce cours a été élaboré à partir des ressources suivantes :

- <https://openclassrooms.com/fr/courses/2035736-mettez-en-place-lintegration-et-la-livraison-continues-avec-la-demarche-devops>
- [https://tropars.github.io/downloads/lectures/DevOps/devops\\_11\\_Integration\\_Continue.pdf](https://tropars.github.io/downloads/lectures/DevOps/devops_11_Integration_Continue.pdf)
- <https://fr.parasoft.com/blog/implementing-qa-in-a-ci-cd-pipeline/>
- <https://www.mindtheproduct.com/what-the-hell-are-ci-cd-and-devops-a-cheatsheet-for-the-rest-of-us/>
- <https://docs.github.com/en/actions/writing-workflows/about-workflows>

