

Développement front avancé

MMI 3 – TP#4 S6

Danielo JEAN-LOUIS

Au début du déploiement de sites web

- Planification du déploiement en amont
 - Parfois des mois en avance
- Création d'une archive avec les assets :
 - HTML, CSS, PHP, dépendances...
- Upload de fichiers divers : traductions...

Au début du déploiement de sites web

- Tests par la QA
 - Retour à la phase de dev si bug trouvé
- Actualisation d'un guide mise en prod pour les ops
 - Si nécessaire
- Mise en production

Avec la complexité des projets, chaque mise en production est une tâche fastidieuse, **risquée** et longue. Elle peut faire perdre beaucoup d'argent à des entreprises en cas d'erreur

Intégration continue / Livraison continue

- Appelé communément (pipeline) CI/CD
 - **Continuous Integration/Continuous Delivery ou Deployment**
 - Facilite le déploiement de projets (notamment en équipe)

Intégration continue / Livraison continue

- Automatisation de tâches sur un serveur :
 - Compilation, déploiement, tests unitaires, migrations...
- Chaque déploiement se passe de la même façon

Automatisation - Avantages

- Limite les risques d'erreurs et d'oublis
 - “Flemme d'exécuter les tests” *lol*
- Assure d'avoir le même environnement
- Permet de traquer et rejouer les erreurs aisément

La CI/CD consiste à créer une chaîne de commandes du développement au déploiement

Intégration continue / Livraison continue

- Prévient les bugs en production et lors du déploiement
 - Exécution régulière de tests
- Entre dans la logique de SCRUM : livraison régulière d'une nouvelle itération

Exemple : Knight Capital (08/2012)

- **Ancienne** entreprise de trading à haute fréquence
- Plus gros négociateur d'actions américaines
 - Représentait ~17 % des parts de marché sur le New York Stock Exchange (NYSE) et le NASDAQ

Source(s) :

- https://en.wikipedia.org/wiki/Knight_Capital_Group#2012_stock_trading_disruption - anglais
- <https://programmation.developpez.com/actu/361198/Knights-Capital-a-ete-victime-du-bogue-logiciel-le-plus-couteux-de-l-histoire-de-l-humanite-49-millions-de-dollars-par-seconde-8-6-milliards-de-dollars-en-28-minutes/>

Exemple : Knight Capital (08/2012)

- Perte de 440 millions de dollars à cause une mise à jour manuelle de serveurs incomplète
 - Un serveur avait été oublié → Exécution de 4 millions d'ordres d'achat non voulus et **non testés**
 - Fonction de test cassée lors de la dernière māj

L'intégration continue aurait évité cette catastrophe

Source(s) :

- https://en.wikipedia.org/wiki/Knight_Capital_Group#2012_stock_trading_disruption - anglais
- <https://programmation.developpez.com/actu/361198/Knights-Capital-a-ete-victime-du-bogue-logiciel-le-plus-couteux-de-l-histoire-de-l-humanite-49-millions-de-dollars-par-seconde-8-6-milliards-de-dollars-en-28-minutes/>

Intégration continue / Livraison continue

- S'articule autour d'un VCS (Version Control System)
 - Git, svn, Perforce...
- Existe dans toute typologie de projet : site web, application mobile...
- Géré par un(e) DevOps

Intégration continue / Livraison continue

- En résumé :
 - Intégration continue : Compile, teste et “emballe le code”
 - Livraison continue : Déploie le “paquet” sur un environnement externe

Le tout automatiquement

DevOps

- Métier combinant le développement (dev) et l'administration système (ops / it)
 - Réconcilie les deux domaines
 - 90 % d'expertise en développement / 10 % d'automatisme
- Profil très recherché

DevOps

- Est à l'aise avec la ligne de commandes Linux (ou Windows)
- Connaît un VCS (indispensable pour la CI/CD)
 - Version Control System : git, svn...
- Connaît les infrastructures Cloud : AWS, Azure...

DevOps

- Facilite le déploiement en production du code
 - Phase **critique** du développement logiciel
- Instaure « l'Infrastructure As Code »

Infrastructure as code

- Mouvance née avec DevOps
- Permet de configurer l'infrastructure serveur / machine via le code
 - Facilité de réplication de configuration
- Limite l'intervention humaine
 - Réduction d'erreurs / oublis

Un(e) DevOps priorise les processus avant les outils de déploiement. Autrement dit, il apporte une « culture » en entreprise.

Utiliser un outil de CI/CD sans comprendre le contexte ne fait pas de vous un(e) DevOps.

Intégration continue / Livraison continue

Grandes étapes

1) Compilation

- Ex : Suite à un push

2) Test : performances, unitaires, e2e, sécurité...

- Automatisés et manuels

3) Déploiement

Source(s) :

- <https://about.gitlab.com/fr-fr/topics/ci-cd/cicd-pipeline/>

Intégration Continue (CI)

- Intégration du code régulière dans un environnement accessible à tous
 - Évite les branches oubliées / abandonnées
- Vérifie le code à chaque modification du code source
 - Limite les régressions

Source(s) :

- <https://aws.amazon.com/fr/devops/continuous-integration/>

Intégration Continue (CI)

- Permet de détecter les problèmes en amont et plus rapidement
 - Un problème résolu en phase de développement coûte moins cher qu'en production

Source(s) :

- <https://aws.amazon.com/fr/devops/continuous-integration/>

Livraison / Déploiement Continu (CD)

- Gère les environnements intermédiaires :
 - Stage, preprod... (delivery)
- Déploie sur le serveur de production (deployment)
 - Permet un déploiement partiel (Canary release), Blue-Green deployment...

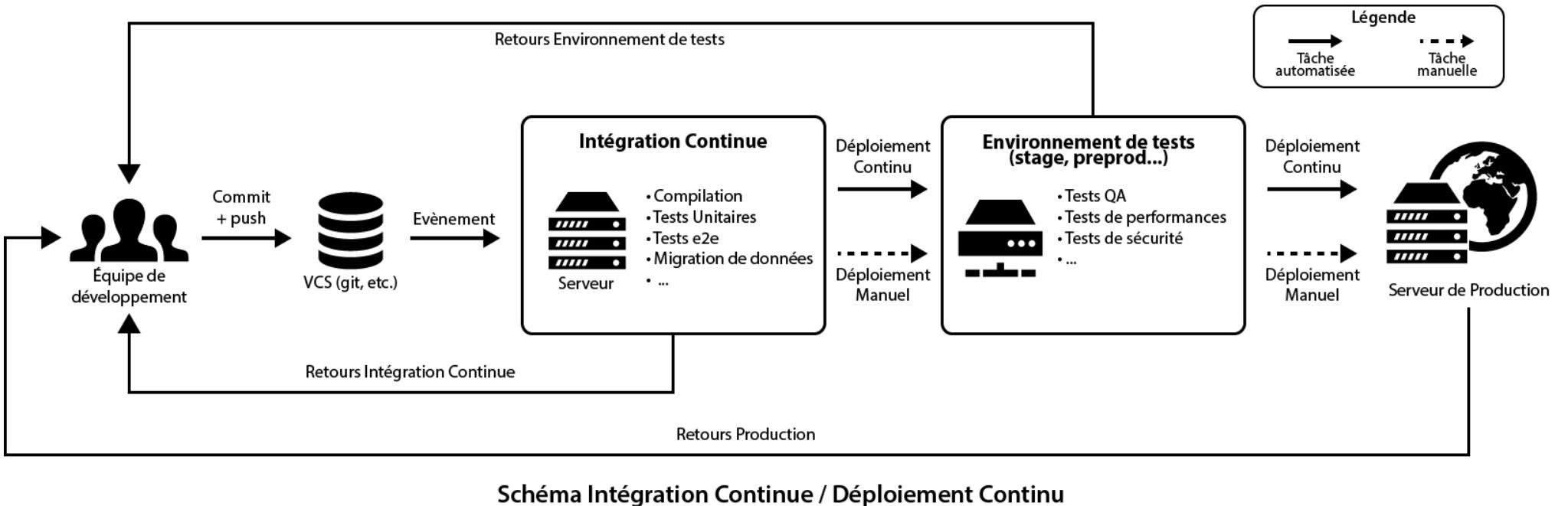
Source(s) :

- <https://github.com/WICG/import-maps?tab=readme-ov-file#installation>
- <https://geekflare.com/fr/blue-green-vs-canary-deployment/>

Livraison / Déploiement Continu (CD)

- Crédit à la création de versions (versioning)
- Possibilité de *rollback* prompt en cas de problème
- Permet de générer un build de production à tout moment
 - Livraison possible de petites mises à jour régulièrement et facilement

CI / CD - Schéma



Source(s) :

- <https://github.com/WICG/import-maps?tab=readme-ov-file#installation>

CD - Règles

- On évite de déployer le vendredi
 - En cas de problème, finir tard un vendredi, ce n'est pas génial
- Après chaque déploiement en production, on taggue le déploiement
 - Permet de “figer” le code source
 - On suit le *Semantic Versioning*

Source(s) :

- <https://semver.org/lang/fr/>

Semantic Versioning

- Standardisation de la gestion de version
- Prend la forme X.Y.Z
 - Trois entiers positifs
- X : majeur – Changement(s) non rétrocompatible(s)
- Y : mineur – Ajout de fonctionnalité(s)
- Z : correctif – Correction de bug(s)

Source(s) :

- <https://semver.org/lang/fr/>

Semantic Versioning

- Possibilité d'ajouter un préfixe pour les versions instables :
 - Ex : 11.42.0-rc1 ou 1.7.0-alpha2

Source(s) :

- <https://semver.org/lang/fr/>

CI/CD - Pre-prod(uction)

- Copie 1:1 de la production en terme d'environnement
 - A variances minimes
- Non accessible au public

CI/CD - Pre-prod(uction)

- Sert à tester le produit dans un environnement semblable à l'utilisateur final
- Le code validé en pré-prod est transféré en prod

git

- VCS le plus populaire
 - VCS : Version Control System
- Présent par défaut sous Linux
- Pierre angulaire du CI/CD
 - Indispensable

.gitignore

- Fichier excluant des fichiers du dépôt
- Permet d'alléger les dépôts
 - **Inutile de commiter vos dépendances**
- Préférable d'être présent à la racine
- Possibilité d'avoir plusieurs .gitignore

Source(s) :

- <https://github.com/github/gitignore>

GitHub Actions

- Solution freemium permettant la CI/CD sur n'importe quel dépôt sur github
- Permet de créer une pipeline CI/CD
 - Pipeline : Ensemble de tâches
 - Synonyme de pipeline : workflow

Source(s) :

- <https://docs.github.com/fr/actions>
- <https://github.com/actions>

GitHub Actions – Tarification / compte



Plan	Stockage	Minutes (par mois)
GitHub Free	500 Mo	2 000
GitHub Pro	1 Go	3 000
GitHub Free pour les organisations	500 Mo	2 000
GitHub Team	2 Go	3 000
GitHub Enterprise Cloud	50 Go	50 000

Passé ces limites, vous serez facturé(e) à la minute (temps d'exécution d'une tâche) – Carte bancaire non nécessaire pour l'utiliser gratuitement

Source(s) :

- <https://docs.github.com/fr/billing/managing-billing-for-your-products/managing-billing-for-github-actions/about-billing-for-github-actions>

GitHub Actions - Tarification

2 000 minutes, c'est très peu.
Évitez d'exécuter des tâches qui ne servent à rien dans votre CI/CD,
c'est du temps serveur.

Source(s) :

- <https://docs.github.com/fr/billing/managing-billing-for-your-products/managing-billing-for-github-actions/about-billing-for-github-actions>

GitHub Actions

- Repose sur un système d'évènements
- Tourne sur serveur macOS, Windows ou Linux
 - Linux coûte le moins cher
- Ne fonctionne qu'avec GitHub

Source(s) :

- <https://docs.github.com/fr/actions>
- <https://github.com/actions>

GitHub Actions

- Fonctionne avec des conteneurs Docker
 - Entre chaque pipeline les données ne sont pas conservées
- Envoie un e-mail, si échec
 - Désactivation : Settings > Notifications > Actions

Source(s) :

- <https://docs.github.com/fr/actions>
- <https://github.com/settings/notifications>

GitHub Actions

- Documentation en franglais
- Ne permet pas de remplir les prompts
 - Utilisation de fichiers de configuration ou de commandes pour passer/remplir les prompts

Source(s) :

- <https://docs.github.com/fr/actions>
- <https://github.com/settings/notifications>

Docker

- Exécute des applications cloisonnées
 - Appelée “conteneur”
 - Accède aux données de l'hôte
- Plus léger que la virtualisation
 - Un conteneur ne contient pas un OS complet (pas de GUI), seulement la CLI

Source(s) :

- https://cyber.gouv.fr/sites/default/files/2020/12/docker_fiche_technique.pdf
- <https://www.docker.com/>

Docker

- Résout le “ça marche chez moi”
 - Serveurs et développeurs ont le même environnement de travail
 - Versions, dépendances, plugins...
- Fonctionne aussi bien sur un serveur qu'un ordinateur (Windows/Linux/macOS)

Source(s) :

- https://cyber.gouv.fr/sites/default/files/2020/12/docker_fiche_technique.pdf
- <https://www.docker.com/>

Docker

- Possibilité de reproduction d'environnement et déploiement aisées grâce aux images
 - Point 10 du manifeste “Twelve-Factor”

Source(s) :

- https://cyber.gouv.fr/sites/default/files/2020/12/docker_fiche_technique.pdf
- <https://www.docker.com/>
- <https://12factor.net/fr/dev-prod-parity>

GitHub Actions

- Permet d'effectuer des cron
 - Cron : Tâches planifiées
- Configuré avec des fichiers YAML placés dans le dossier “.github/workflows”

Source(s) :

- <https://docs.github.com/fr/actions>

Fichier .yml / .yaml

- Format souvent utilisé pour la configuration
 - Utilisé notamment par Symfony
- Inspiré par le format CSV
 - YAML utilise des indentations pour structurer le contenu

Source(s) :

- <https://fr.wikipedia.org/wiki/YAML>

Fichier .yml / .yaml

- Permet la gestion de données complexes
 - Tout en gardant une lisibilité
- Deux espaces par indentation (par convention)

Source(s) :

- <https://fr.wikipedia.org/wiki/YAML>
- <https://marketplace.visualstudio.com/items?itemName=EditorConfig.EditorConfig>

Fichier .yml / .yaml



YAML

```
training: "MMI"
route: Développement Web et dispositifs interactifs
list_students:
  - firstname: Helena # Comment
    lastname: Despoux
  - { firstname: Roger, lastname: Gros }
```

Exemple de fichier YAML

Fichier .yml / .yaml

```
{  
    "training": "MMI",  
    "route": "Développement Web et Dispositifs interactifs",  
    "list_students": [  
        {"firstname": "Helena", "lastname": "Despoux"},  
        {"firstname": "Roger", "lastname": "Gros"}  
    ]  
}
```

Équivalent JSON du fichier YAML vu précédemment
(Rappel : les commentaires ne sont pas possibles en JSON)

.editorconfig

- Fichier de configuration permettant d'uniformiser les styles entre les fichiers
 - Indentation : nombre et type
 - Espaces en fin de ligne : oui ou non
 - ...
- Fichier partagé au sein de l'équipe de développement

Source(s) :

- <https://editorconfig.org/>
- <https://marketplace.visualstudio.com/items?itemName=EditorConfig.EditorConfig>

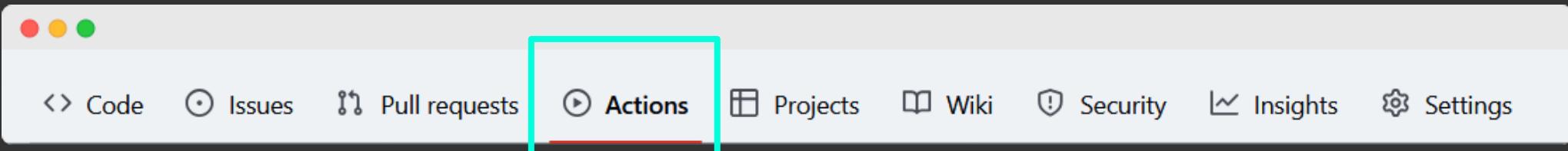
.editorconfig

- Évite des différences de fichiers à chaque commit
- Utilisation de l'extension “EditorConfig for VS Code” pour gérer le fichier

Source(s) :

- <https://editorconfig.org/>
- <https://marketplace.visualstudio.com/items?itemName=EditorConfig.EditorConfig>

GitHub Actions - Cr ation



Note : L'onglet “Actions” peut  tre d sactiv . Settings > Actions > General.



A screenshot of the 'Choose a workflow' page. It shows the heading 'Choose a workflow' and a descriptive text: 'Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.' Below this, there is a link 'Skip this and set up a workflow yourself →' which is highlighted with a red box.

Source(s) :

- <https://docs.github.com/fr/actions>

Pratiquons ! - GitHub Actions (Partie 1)

Pré-requis :

- Avoir la ressource `ressources/github-actions`

A télécharger ici : https://github.com/DanYellow/cours/blob/refs/heads/main/s6-developpement-web-et-dispositif-interactif/travaux-pratiques/numero-4/s6-developpement-web-et-dispositif-interactif_travaux-pratiques_numero-4.ressources.zip

Act

- Extension VS Code gratuite permettant de faire tourner vos GitHub Actions en local
- Nécessite Docker sur votre ordinateur

Source(s) :

- <https://github.com/nektos/act>

GitHub Actions - Gabarit



YAML

```
name: My first action
on: [my_event]
jobs:
  job_name:
    runs-on: volume_name
    steps:
      - name: step_name (optional)
      - run: command
```

Exemple de base d'un fichier de pipeline. Pour "on", le tableau n'est pas obligatoire.

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows>

GitHub Actions - Gabarit

- Chaque fichier d'actions doit contenir au moins deux clés racines :
 - on : Évènement qui va lancer la pipeline (pull, push...) - Valeurs définies (voir source)
 - jobs : Tâches à effectuer

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows>

GitHub Actions - Gabarit



YAML

```
name: My first action
on: [my_event]
jobs:
  job_name:
    runs-on: volume_name
    steps:
      - name: step_name (optional)
      - run: command
```

Pipeline / workflow

Fichier YAML exécuté quand un évènement a lieu

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows>

GitHub Actions - Gabarit



YAML

```
name: My first action
on: [my_event]
jobs:
  job_name:
    runs-on: volume_name
    steps:
      - name: step_name (optional)
      - run: command
```

Job

Ensemble de tâches exécutées dans un conteneur

- Note : les jobs sont exécutés en parallèle par défaut

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows>

GitHub Actions - Gabarit



YAML

```
name: My first action
on: [my_event]
jobs:
  job_name:
    runs-on: volume_name
    steps:
      - name: step_name (optional)
        run: command
```

Tâche

Exécution d'une commande (création de fichier, tirage de branche...)

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows>

GitHub Actions - Gabarit

- Le même fichier peut contenir plusieurs jobs
 - Il est préférable de séparer votre pipeline en plusieurs jobs
 - Un job : une grande tâche (déploiement, migration, tests...)

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows>

GitHub Actions – Action définie

- Ensemble de tâches personnalisées et complexes. Ex : Tirage de dépôt
 - Développés par la communauté et GitHub
- S'utilise avec la clé “uses” (à la place de “run”) dans le fichier yaml
 - Possibilité d'avoir plusieurs uses au sein du même job

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows>
- <https://github.com/sdras/awesome-actions?tab=readme-ov-file#official-actions>

GitHub Actions – Actions définies



YML

```
name: Node Continuous Integration

on:
  push:
    branches: [ master ]

jobs:
  create_build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v1
        with:
          node-version: 20
      - name: Install dependencies
        run: npm ci
      - name: Create build
        run: npm run build
```

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows>

Pratiquons ! - GitHub Actions (Partie 2)

Pré-requis :

- Avoir la ressource `ressources/github-actions`

A télécharger ici : https://github.com/DanYellow/cours/blob/refs/heads/main/s6-developpement-web-et-dispositif-interactif/travaux-pratiques/numero-4/s6-developpement-web-et-dispositif-interactif_travaux-pratiques_numero-4.ressources.zip

GitHub Actions – Fichier de workflow

- Seuls les fichiers de workflows de la branche par défaut sont accessibles depuis l'interface des Actions

GitHub Actions – Variables

d'environnement Permettent de réutiliser une valeur au sein d'un job

- Plusieurs portée possibles :
 - Globale, job, tâche
- Préfixée par “\$” pour être affichée
- Accessibles dans votre code (js, bash...)

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/workflow-commands-for-github-actions#environment-files>

GitHub Actions – Variables d'environnement

```
name: Display a variable

on:
  workflow_dispatch

env:
  UNIVERSITY: CY Paris Université # Global scope

jobs:
  display_student_infos:
    runs-on: ubuntu-latest
    env:
      FORMATION: BUT MMI # Job scope
    steps:
      - name: "Presentation"
        run: echo "I'm $FIRST_NAME, I'm a student in $FORMATION at $UNIVERSITY"
    env:
      FIRST_NAME: John Doe # Step scope
```

Ici, nous avons trois variables avec trois portées différentes

GitHub Actions – Variables d'environnement

Ne permettent pas d'interpréter des expressions...

```
● ● ● XML  
env:  
  MY_VAR: ls -al
```

```
jobs:  
  my_job:  
    #[ ... ]  
    steps:  
      - run: echo "$MY_VAR"  
    #[ ... ]
```

MY_VAR contient “ls -al” et non le résultat de la commande. En l'occurrence le contenu du dossier.

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/workflow-commands-for-github-actions#environment-files>

GitHub Actions – Variables

d'environnement

...mais il est possible d'en créer à la volée

- Utilisation de la variable \$GITHUB_ENV



YAML

```
jobs:  
  my_job:  
    #[ ... ]  
    steps:  
      - run: echo "MY_VAR=$(date +'%Y/%m/%d')" >> "$GITHUB_ENV"  
    #[ ... ]
```

MY_VAR est accessible dans l'objet “env”

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/workflow-commands-for-github-actions#environment-files>

GitHub Actions – Variables

Les variables d'environnement créées à la volée ne sont pas accessible au sein de l'étape qui les crée

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/workflow-commands-for-github-actions#environment-files>

GitHub Actions – Context

- Objets par défaut permettant d'accéder à diverses informations : état du job, nom de l'utilisateur courant...
- S'affiche "\${{ <context> }}"

Source(s) :

- <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/accessing-contextual-information-about-workflow-runs>

GitHub Actions – Secrets

- Variables d'environnement privées
 - Valeurs qui ne doivent pas être publiques...
 - ...mais qu'on veut utiliser dans ses pipelines. Ex : mot de passe
- Données chiffrées

Source(s) :

- <https://docs.github.com/fr/actions/security-for-github-actions/security-guides/using-secrets-in-github-actions>

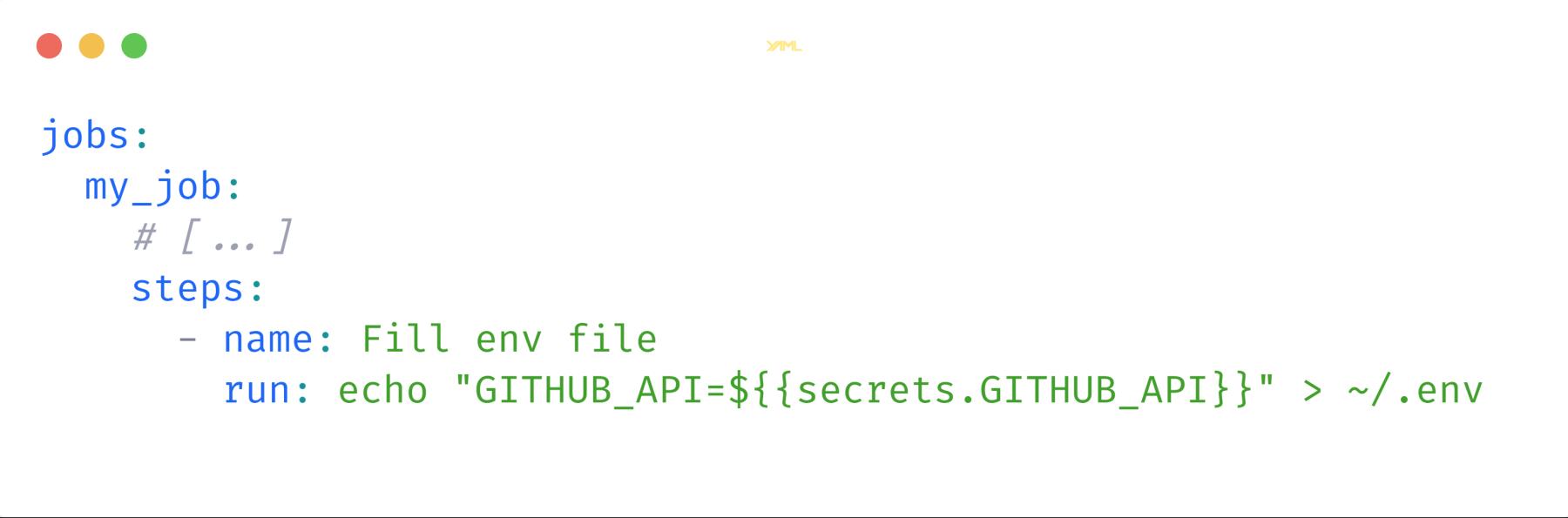
GitHub Actions – Secrets

- Ne doivent pas commencer par un nombre
 - Caractères alphanumériques et underscore seulement
 - Non sensibles à la casse
- Peuvent être écrit dans un fichier
 - Ex : Fichier .env

Source(s) :

- <https://docs.github.com/fr/actions/security-for-github-actions/security-guides/using-secrets-in-github-actions>

GitHub Actions – Secrets



```
jobs:
  my_job:
    # [ ... ]
    steps:
      - name: Fill env file
        run: echo "GITHUB_API=${{secrets.GITHUB_API}}" > ~/.env
```

Notre secret “GITHUB_API” est écrit en clair dans le fichier .env

Source(s) :

- <https://docs.github.com/fr/actions/security-for-github-actions/security-guides/using-secrets-in-github-actions>

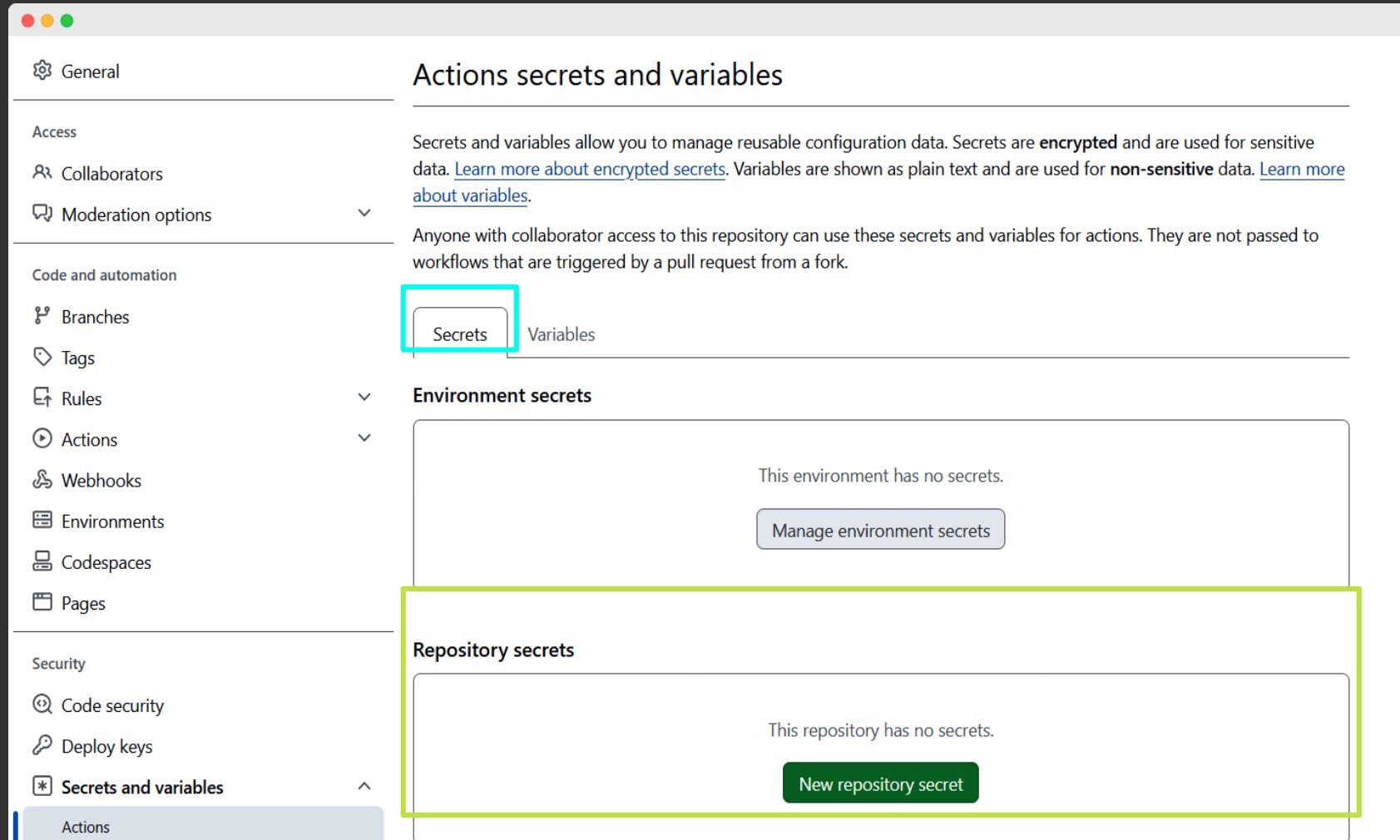
GitHub Actions – Secrets

- Chargés depuis les paramètres du dépôt
 - Settings > Secrets and variables > Actions
- S'utilisent comme les variables de contexte
 - `${{ secrets.SECRET_KEY }}`

Source(s) :

- <https://docs.github.com/fr/actions/security-for-github-actions/security-guides/using-secrets-in-github-actions>

GitHub Actions – Secrets



The screenshot shows the 'Actions secrets and variables' section of the GitHub repository settings. On the left, there's a sidebar with various repository management options like General, Access, Collaborators, and Moderation options. Below that is a 'Code and automation' section with Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, and Pages. At the bottom of the sidebar is a 'Secrets and variables' option, which is currently selected and highlighted with a blue border. The main content area has two tabs: 'Secrets' (which is active and highlighted with a red border) and 'Variables'. Under the 'Secrets' tab, there are two sections: 'Environment secrets' and 'Repository secrets'. Both sections show a message indicating they have no secrets ('This environment has no secrets.' and 'This repository has no secrets.') and contain a 'Manage environment secrets' button and a 'New repository secret' button respectively.

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Rules

Actions

Webhooks

Environments

Codespaces

Pages

Secrets and variables

Actions

Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. [Learn more about encrypted secrets](#). Variables are shown as plain text and are used for **non-sensitive** data. [Learn more about variables](#).

Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.

Secrets

Variables

Environment secrets

This environment has no secrets.

Manage environment secrets

Repository secrets

This repository has no secrets.

New repository secret

GitHub Actions – Artifact

- Représente le résultat d'un build **persistant** sur le serveur de CI/CD
 - Durée de vie par défaut : 90 jours
- Peut être lu par d'autres jobs
 - Ex : Job de déploiement

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/storing-and-sharing-data-from-a-workflow>

GitHub Actions – Artifact (upload)

- Nécessite l'action “action/upload-artifact@master” pour être partagé

YAML

```
steps:  
  - name: Generate artifact  
    uses: actions/upload-artifact@master  
    with:  
      name: bundle # artifact / directory name on the server  
      path: ./dist # source directory
```

On copie le contenu du dossier “dist” dans un artifact nommé “bundle”

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/storing-and-sharing-data-from-a-workflow>

GitHub Actions – Artifact (upload)

- Les artifacts générés peuvent être téléchargés manuellement depuis la page “summary” d'une pipeline

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/storing-and-sharing-data-from-a-workflow>

GitHub Actions – Artifact (download)

- Nécessite l'action “action/download-artifact@master” pour être récupéré



yml

steps:

```
- name: Download artifact
  uses: actions/download-artifact@v4
  with:
    name: my_artifact # Artifact to download
    path: ./build # Destination path
```

On récupère le contenu de notre artifact “my_artifact” dans le dossier build

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/storing-and-sharing-data-from-a-workflow>
- <https://github.com/actions/download-artifact/blob/main/README.md>

GitHub Actions – Inter-dépendances

- Permet d'attendre l'exécution d'un job avant l'exécution d'un autre
 - Multiple dépendances possibles
 - Utilisation de la clé “needs”

Source(s) :

- <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/using-jobs-in-a-workflow#define-prerequisite-jobs>

GitHub Actions – Inter-dépendances



yml

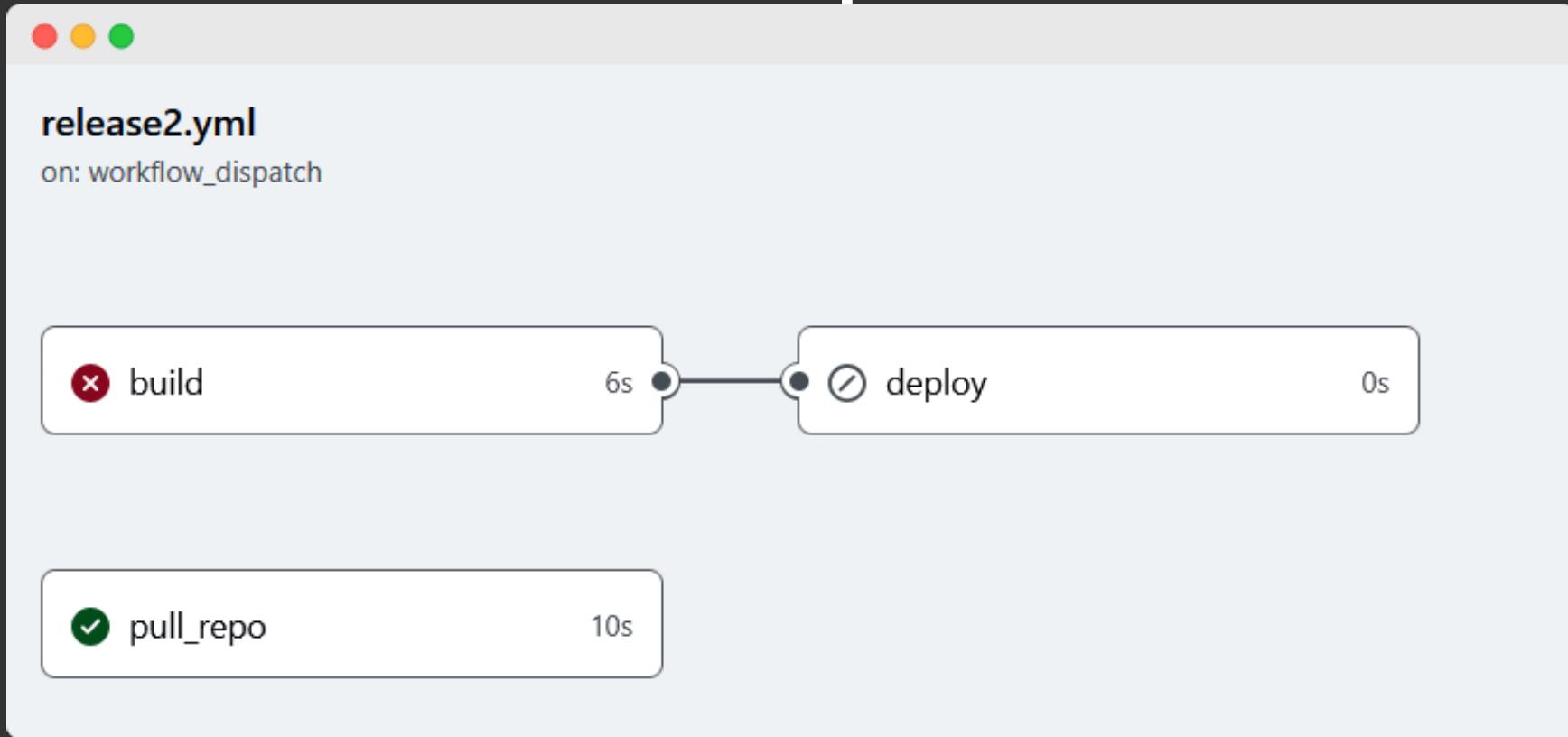
```
deploy:  
  runs-on: ubuntu-latest  
  needs: [build]  
  
steps:  
  - name: # ...
```

Notre job “deploy” ne peut s’exécuter que si et seulement si le job “build” s’est terminé avec succès

Source(s) :

- <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/using-jobs-in-a-workflow#define-prerequisite-jobs>

GitHub Actions – Inter-dépendances



L'interface de GitHub Actions nous indique l'interdépendance de jobs

Source(s) :

- <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/using-jobs-in-a-workflow#define-prerequisite-jobs>

Pratiquons ! - GitHub Actions (Partie 3)

Pré-requis :

- Avoir la ressource `ressources/github-actions`

A télécharger ici : https://github.com/DanYellow/cours/blob/refs/heads/main/s6-developpement-web-et-dispositif-interactif/travaux-pratiques/numero-4/s6-developpement-web-et-dispositif-interactif_travaux-pratiques_numero-4.ressources.zip

GitHub Actions – Inputs

- Permet de définir des valeurs depuis github qui seront utilisées dans votre workflow
 - Ex : Définir le serveur de stage
- Plusieurs types de données possibles : choice (équivalent <select> en HTML), boolean, string et environnement

Source(s) :

- <https://github.blog/changelog/2021-11-10-github-actions-input-types-for-manual-workflows/>

GitHub Actions – Inputs

- Ne fonctionne qu'avec les workflows lancés manuellement
 - on : workflow_dispatch
- Valeurs accessibles depuis la variable “github.event.inputs.VALUE”

Source(s) :

- <https://github.blog/changelog/2021-11-10-github-actions-input-types-for-manual-workflows/>

GitHub Actions – Inputs



YAML

```
name: Github inputs

on:
  workflow_dispatch:
    stage:
      type: choice
      description: Select stage env
      default: stage2
      options:
        - stage1
        - stage2

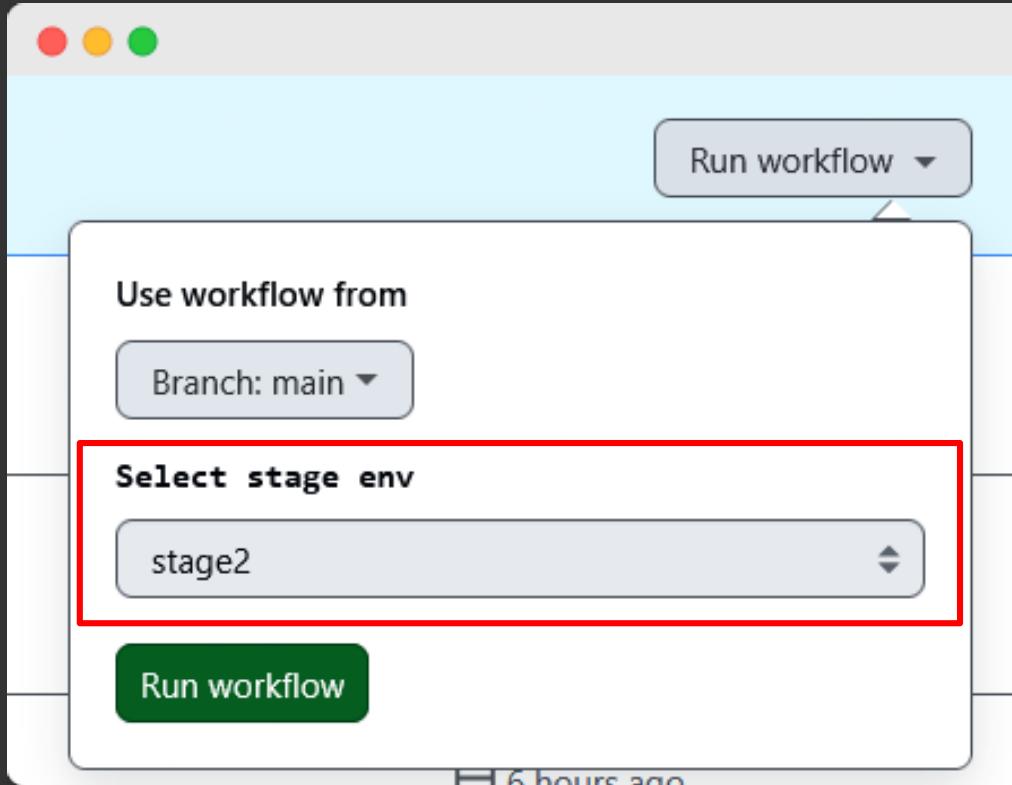
jobs:
  my_input:
    # [ ... ]
    steps:
      - name: Use my input value
        run: echo "${{ github.event.inputs.stage }}"
```

Ici nous déclarons un <select> avec deux choix possibles

Source(s) :

- <https://github.blog/changelog/2021-11-10-github-actions-input-types-for-manual-workflows/>

GitHub Actions – Inputs



Depuis l'interface des actions, je peux changer à la volée des variables de mon workflow

Source(s) :

- <https://github.blog/changelog/2021-11-10-github-actions-input-types-for-manual-workflows/>

GitHub Actions – workflow_call

- Évènement permettent à une pipeline d'être appelée dans une autre
 - Réutilisation de la pipeline
 - Utilisation d'un chemin relatif à la racine du dépôt pour être appelé

Source(s) :

- https://docs.github.com/fr/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows#workflow_call
- <https://docs.github.com/fr/actions/sharing-automations/reusing-workflows>

GitHub Actions – pull_request

- Évènement permettent à une pipeline d'être appelée quand une pull_request est faite
 - Plusieurs sous évènements possibles
- La pipeline est exécutée sur la branche qui effectue la pull request

Source(s) :

- <https://docs.github.com/fr/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/managing-a-branch-protection-rule>

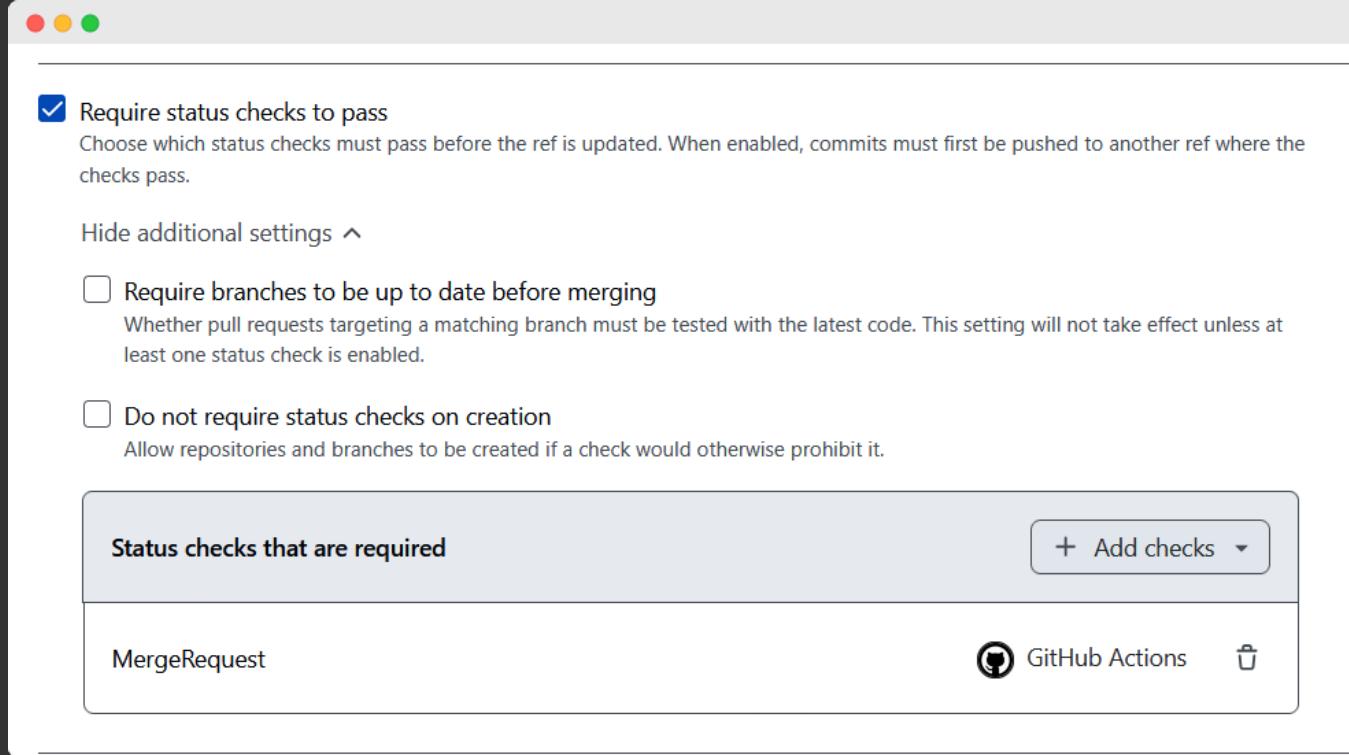
GitHub Actions – pull_request

- Possibilité de bloquer toute fusion si la branche ne valide pas la pipeline
 - Menu : Settings > Branches > Branch protection rules

Source(s) :

- <https://docs.github.com/fr/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/managing-a-branch-protection-rule>

GitHub Actions – pull_request



Nous avons défini notre pipeline “MergeRequest” comme étant une condition sine quo non pour fusionner une branche

Source(s) :

- <https://docs.github.com/fr/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/managing-a-branch-protection-rule>

GitHub Actions – Condition

- Permet de lancer une tâche / job si une condition est remplie
- Deux niveaux possibles : Job et tâche
 - Note : si un job est conditionnel et qu'un autre job en dépend, ce dernier ne sera pas exécuté

GitHub Actions – Condition



YAML

```
jobs:  
  my_condition_job:  
    # [...]  
    steps:  
      - name: Run bash file  
        run: |  
          chmod +x ./my-bash-file.sh  
          ./my-bash-file.sh  
          if: ${{ github.ref = 'refs/heads/main' }}  
    # [...]
```

Le fichier bash sera exécuté si et seulement si la branche est “main”

GitHub Actions – Condition

```
# [...]
if: ${{ github.ref == 'github.ref_name' && github.actor != "Gentoo" }}
# [...]
```

Condition avec de multiples critères

GitHub Actions – Condition

- Fonctions d'état : permettent de lancer un job / tâche en fonction de l'état de la pipeline
 - `${{ always() }}` : Toujours exécuté
 - `${{ cancelled() }}` : En cas d'annulation
 - `${{ failure() }}` : En cas d'échec
 - `${{ success() }}` : Si succès

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/evaluate-expressions-in-workflows-and-actions#status-check-functions>

GitHub Actions – Annotation

- Affiche un message dans le résumé d'une pipeline
- Quatre types de messages possibles



Un exemple d'annotation sous un build

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/workflow-commands-for-github-actions#setting-a-debug-message>

GitHub Actions – Annotation



YAML

```
jobs:  
  my_condition_job:  
    # [...]  
    steps:  
      - name: Start mysql service  
        run: echo "::notice My message"  
        if: ${{ hashFiles("mon-fichier.html") = '' }}  
      # [...]
```

Notre annotation s'affichera si la condition est remplie

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/workflow-commands-for-github-actions#setting-a-debug-message>

GitHub Actions – Shell (terminal)

- Paramètre de job permettant de changer de terminal
 - shell par défaut : celui du runner
- Shells possibles : python, bash, cmd...
- Certains shells sont exclusifs au runner
 - Ex : cmd sur Linux est impossible

Source(s) :

- https://docs.github.com/en/actions/writing-workflows/workflow-syntax-for-github-actions#jobsjob_idstepsshell

GitHub Actions – Shell (terminal)



YAML

```
# [...]
steps:
  - name: Display the modules
    shell: python
    run: |
      print(help('modules'))
```

Cette étape permet d'exécuter du code Python depuis le fichier de pipeline

Source(s) :

- https://docs.github.com/en/actions/writing-workflows/workflow-syntax-for-github-actions#jobsjob_idstepsshell
- <https://docs.github.com/en/actions/writing-workflows/workflow-syntax-for-github-actions#example-running-an-inline-python-script>

GitHub Actions – Outputs

- Variables partageables entre job
 - Les jobs qui en ont besoin doivent être interdépendants
- Ne peuvent pas contenir un secret
 - Si c'est le cas, la valeur sera nulle

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/passing-information-between-jobs>

GitHub Actions – Outputs



YAML

```
my_job_1:
  # [...]
  outputs:
    output_1: ${{ steps.id_1.outputs.lastname }}
  steps:
    - id: id_1
      run: echo "lastname=Thomas" >> "$GITHUB_OUTPUT"

my_job_2:
  # [...]
  needs: my_job_1
  steps:
    - run: echo "${{ needs.define_env.outputs.output_1 }}"
```

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/passing-information-between-jobs>

GitHub Actions – Outputs – A noter

- La tâche qui définit un output doit avoir une clé “id” avec une valeur unique
- L’output peut être stocké directement au niveau de la clé env (portée de job ou de tâche)
- Certaines actions (ex : actions/upload) retournent un output

Source(s) :

- <https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/passing-information-between-jobs>

GitHub Actions – API

- Permet de manipuler la pipeline :
 - Récupération d'artifact, suppression de secret...
- API gratuite
- Utilisable aussi bien sur un site web qu'un workflow

Source(s) :

- <https://docs.github.com/en/rest/actions?apiVersion=2022-11-28>
- <https://cli.github.com/>

GitHub Actions – API

- Accessible depuis la cli GitHub
 - CLI accessible depuis la pipeline
- Nécessite un token qui ne doit pas être public
 - GitHub bloquera votre push s'il est commisé
- GitHub propose d'autres types d'API
 - Collaborateurs, commits...
- Plusieurs SDK disponibles : C#, js...

Source(s) :

- <https://docs.github.com/en/rest/actions?apiVersion=2022-11-28>
- <https://cli.github.com/>

GitHub Actions – API



YAML

```
const headers = {
  'Accept': 'application/vnd.github+json',
  'Authorization': 'Bearer MY-TOKEN',
  'X-GitHub-Api-Version': '2022-11-28',
  'User-Agent': 'curl'
}

const req = await fetch(
  "https://api.github.com/repos/{owner}/{repo}/collaborators",
  { headers }
);
```

Utilisation de `fetch` pour récupérer les collaborateurs d'un dépôt

Source(s) :

- <https://docs.github.com/en/rest/actions?apiVersion=2022-11-28>
- <https://cli.github.com/>

GitHub Actions – API

```
list_collaborators=$(gh api /repos/{OWNER}/{REPOSITORY}/collaborators)

result='[]'
for collaborator in `echo $list_collaborators | jq --raw-output -c '.[]'`; do
    login=`echo $collaborator | jq '.login'`
    user_request=$(gh api /users/$(echo $login | jq --raw-output))

    collaborator={}
    avatar_url=`echo $user_request | jq '.avatar_url'`
    name=`echo $user_request | jq '.name'`

    collaborator=$(jq ".login=${login}" <<< "$collaborator")
    collaborator=$(jq ".name=${name}" <<< "$collaborator")
    collaborator=$(jq ".avatar_url=${avatar_url}" <<< "$collaborator")

    result=$(jq --argjson val "$collaborator" '. += [$val]' <<< "$result")
done

# We cast to string for ENV Var
str_result=$(jq .'| tostring' <<< "$result")
echo "VITE_LIST_COLLABORATORS=$str_result" >> "$GITHUB_ENV"
```

Même chose, mais cette fois ci, on utilise la CLI GitHub en bash et le résultat est passé en env var (code source dans la ressource, fichier .sh)

Source(s) :

- <https://docs.github.com/en/rest/actions?apiVersion=2022-11-28>
- <https://cli.github.com/>

jq (JSON query)

- Commande non native permettant de manipuler le JSON en ligne de commandes
 - Installé par défaut dans un conteneur GitHub Actions
 - Fonctionne sous macOS, Windows et Linux

Source(s) :

- <https://jqplay.org/>
- <https://jqlang.github.io/jq/>

GitHub Actions – Alternatives – Liste non exhaustive

- Circle CI
- Gitlab - https://docs.gitlab.com/ee/ci/quick_start/
 - Fonctionnement très proche de GitHub
- Azure DevOps
- Jenkins : Nécessite **beaucoup** de configuration
- TeamCity
- ...

DevOps – Pour aller plus loin

- <https://roadmap.sh/devops>
- <https://github.com/actions/toolkit>

Questions ?

