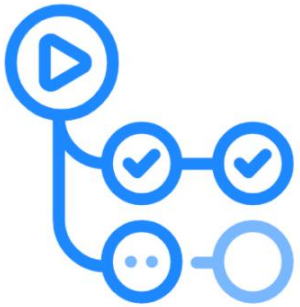
GITHUB ACTION

Quentin BIREAU

C'EST QUOI GITHUB ACTION ?

- GitHub Actions est une plateforme d'intégration continue et livraison continue (CI/CD) qui vous permet d'automatiser votre pipeline de génération, de test et de déploiement. Vous pouvez créer des workflows qui créent et testent chaque demande de tirage (pull request) adressée à votre dépôt, ou déployer des demandes de tirage fusionnées en production.

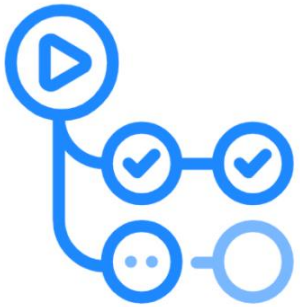
- GitHub Actions vous permet d'exécuter des workflows lorsque d'autres événements se produisent dans votre dépôt. Par exemple, vous pouvez exécuter un workflow pour ajouter automatiquement les étiquettes appropriées chaque fois que quelqu'un crée une issue dans votre dépôt.



GitHub Actions

C'EST QUOI GITHUB ACTION ?

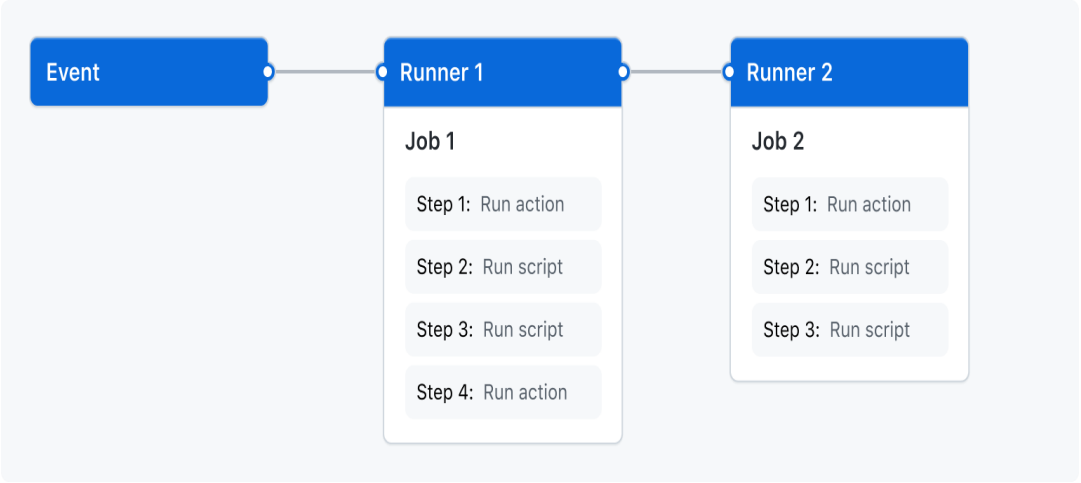
- GitHub fournit des machines virtuelles Linux, Windows et macOS pour exécuter vos workflows, ou vous pouvez héberger vos propres exécuteurs auto-hébergés dans votre propre centre de données ou infrastructure cloud.



GitHub Actions

LES DIFFÉRENTS COMPOSANTS

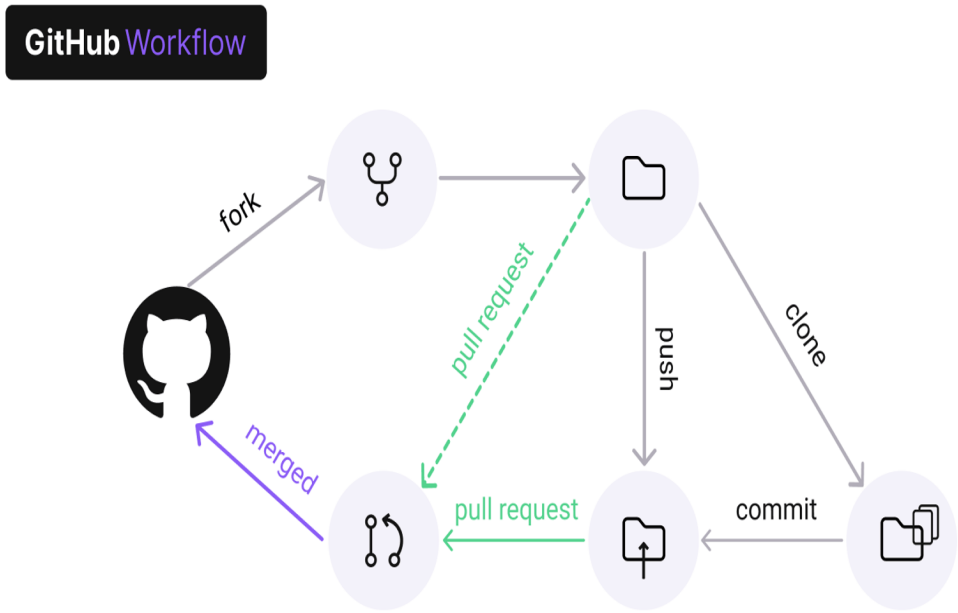
- Vous pouvez configurer un workflow GitHub Actions à déclencher quand un événement se produit dans votre dépôt, par exemple l'ouverture d'une demande de tirage (pull request) ou la création d'un problème. Votre workflow contient un ou plusieurs travaux qui peuvent s'exécuter dans un ordre séquentiel ou en parallèle. Chaque travail s'exécute au sein de son propre exécuteur de machine virtuelle, ou au sein d'un conteneur, et comporte une ou plusieurs étapes qui exécutent un script que vous définissez ou une action, qui est une extension réutilisable qui peut simplifier votre workflow.



- Mais c'est quoi un workflow du coup ?

WORKFLOW

- Un workflow est un processus automatisé configurable qui exécutera un ou plusieurs travaux. Les workflows sont définis par un fichier YAML archivé dans votre dépôt et s'exécutent lorsqu'ils sont déclenchés par un événement dans votre dépôt, ou ils peuvent être déclenchés manuellement ou selon une planification définie.
- Les workflows sont définis dans le répertoire .github/workflows d'un référentiel, et un référentiel peut avoir plusieurs workflows, chacun pouvant effectuer un ensemble différent de tâches. Par exemple, vous pouvez avoir un workflow pour générer et tester des demandes de tirage, un autre workflow pour déployer votre application chaque fois qu'une version est créée, et encore un autre workflow qui ajoute une étiquette chaque fois que quelqu'un ouvre un nouveau problème.



COMMENT METTRE EN PLACE UN WORKFLOW

- GitHub Actions utilise la syntaxe YAML pour définir le workflow. Chaque workflow est stocké en tant que fichier YAML distinct dans votre référentiel de code, dans un répertoire appelé .github/workflows.
- Vous pouvez créer un exemple de workflow dans votre dépôt qui déclenche automatiquement une série de commandes chaque fois que du code est poussé (push). Dans ce workflow, GitHub Actions extrait le code envoyé, installe le framework de test bats et exécute une commande de base pour générer la version de bats : bats -v.
- Validez ces modifications et poussez-les vers votre dépôt GitHub.
- Votre nouveau fichier de workflow GitHub Actions est maintenant installé dans votre dépôt et s'exécute automatiquement chaque fois que quelqu'un pousse (push) une modification vers le dépôt.

```yaml
name: learn-github-actions
run-name: ${{ github.actor }} is learning GitHub Actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '20'
      - run: npm install -g bats
      - run: bats -v
```

MODULE 1 : INTRODUCTION À GITHUB ACTIONS

- Structure et syntaxe
- UI

STRUCTURE ET SYNTAXE

- La structure et la syntaxe des fichiers de configuration GitHub Actions, généralement nommés workflow et situés dans le répertoire .github/workflows de votre repository, suivent un format bien défini utilisant YAML.

  Voici une vue d'ensemble de la structure et des éléments clés ->

STRUCTURE ET SYNTAXE

Structure Générale d'un Fichier de Workflow :

- **Nom du Workflow**
- **Déclencheurs (Triggers)**
- **Jobs**
  - **Environnement d'exécution**
  - **Étapes (Steps)**
  - **Actions et Commandes**

STRUCTURE ET SYNTAXE

Nom du Workflow :

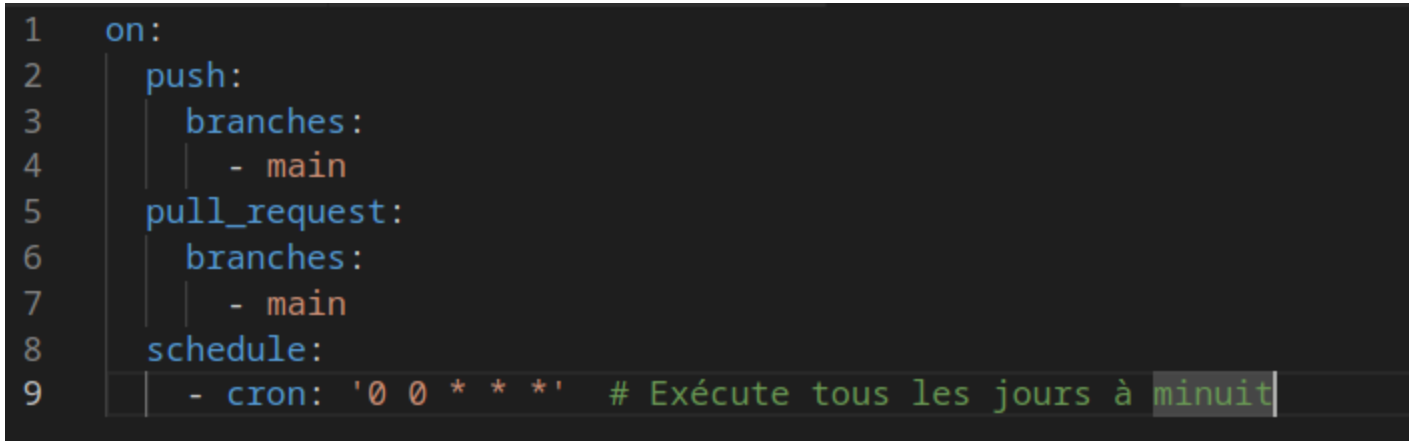
- Le nom du workflow est facultatif, mais il est utile pour l'identification dans l'interface GitHub Actions.

```
name: Nom du Workflow
```

STRUCTURE ET SYNTAXE

Déclencheurs (Triggers)

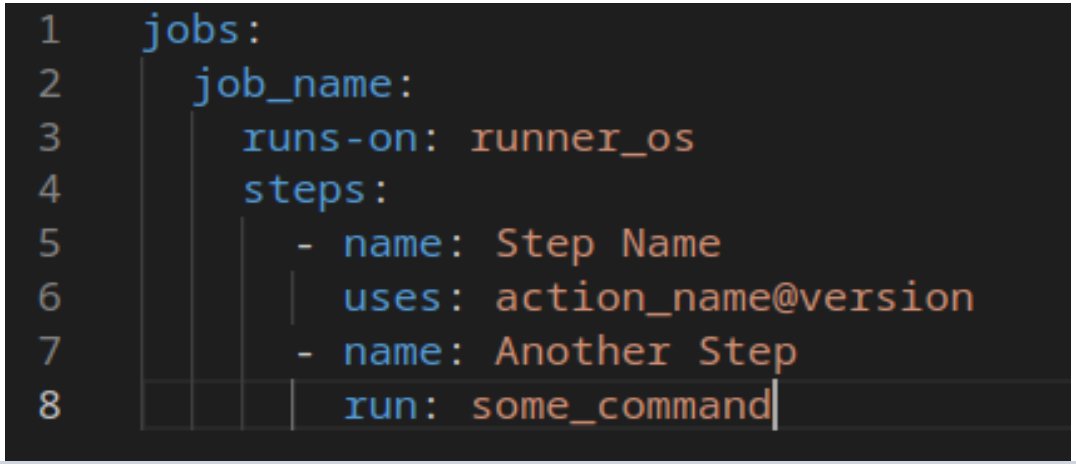
- Les déclencheurs spécifient les événements qui déclenchent l'exécution du workflow. Les déclencheurs courants incluent push, pull_request, schedule, workflow_dispatch, etc.



STRUCTURE ET SYNTAXE

Jobs

- Les jobs sont des ensembles d'étapes qui s'exécutent sur des runners. Un workflow peut avoir un ou plusieurs jobs. Chaque job a son propre environnement et ses propres étapes.



STRUCTURE ET SYNTAXE

Détails des Jobs

a. Nom du Job

Chaque job est identifié par un nom unique dans le workflow -> **job_name:**

b. Environnement d'Exécution (Runner)

Spécifie l'environnement sur lequel le job s'exécute. GitHub propose des runners hébergés tels que ubuntu-latest, windows-latest, et macos-latest -> **runs-on: ubuntu-latest**

STRUCTURE ET SYNTAXE

c. Étapes (Steps)

Les étapes à l'intérieur d'un job définissent les actions à exécuter. Chaque étape peut utiliser une action pré-existante ou exécuter une commande.

Utilisation d'une Action

Les actions sont des tâches réutilisables. L'utilisation d'une action se fait avec la directive uses ->

```
- name: Checkout Repository
  uses: actions/checkout@v4
```

STRUCTURE ET SYNTAXE

- Exécution d'une Commande
- Les commandes shell peuvent être exécutées directement dans une étape en utilisant la directive run.

```
- name: Install Dependencies

  run: npm install
```

STRUCTURE ET SYNTAXE

- Exemples Complet de Workflow
- Voici un exemple complet d'un fichier de workflow qui se déclenche sur les événements push et pull_request, et qui exécute un job pour installer des dépendances et exécuter des tests ->

STRUCTURE ET SYNTAXE

```yaml
name: CI Pipeline

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v4

      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '14'

      - name: Install Dependencies
        run: npm install

      - name: Run Tests
        run: npm test
```

STRUCTURE ET SYNTAXE

Détails "**Additionnels**"

Variables et Secrets

- Vous pouvez définir des variables d'environnement et accéder à des secrets sécurisés, exemple :

```yaml
env:
  NODE_ENV: production
  DATABASE_URL: ${{ secrets.DATABASE_URL }}
```

STRUCTURE ET SYNTAXE

Secrets

Les secrets sont définis dans les paramètres du repository et sont référencés dans les workflows.

```yaml
- name: Deploy
  env:
    DEPLOY_KEY: ${{ secrets.DEPLOY_KEY }}
  run: ./deploy.sh
```

STRUCTURE ET SYNTAXE

Utilisation des Expressions

GitHub Actions supporte les expressions pour accéder aux contextes et aux variables. Les expressions sont entourées de ${{ }}.

Exemple d'expression :

```
run: echo "This job is running on branch ${{ github.ref }}"
```

STRUCTURE ET SYNTAXE

Les expressions peuvent également être utilisées pour des conditions (if), des inputs, des outputs, etc.

Exemple d'utilisation d'une condition :

```yaml
- name: Deploy
  if: github.ref == 'refs/heads/main'
  run: ./deploy.sh
```

STRUCTURE ET SYNTAXE

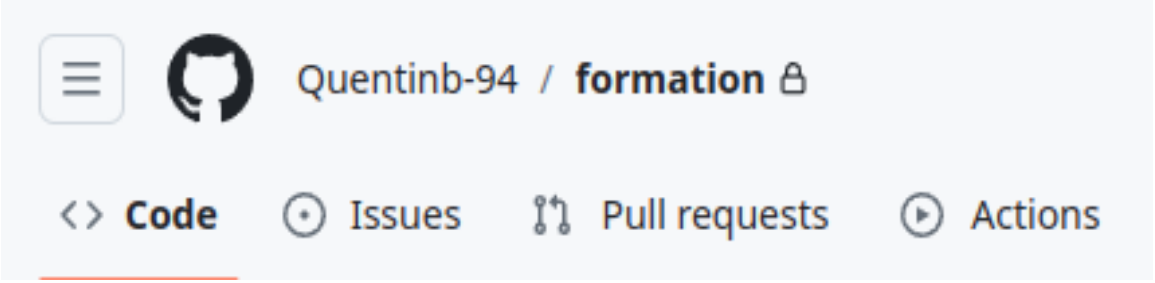
En résumé, la structure et la syntaxe de GitHub Actions permettent une grande flexibilité et puissance pour automatiser vos workflows CI/CD. En définissant des déclencheurs, des jobs, et des étapes de manière claire et concise, vous pouvez automatiser une large gamme de tâches directement dans votre repository GitHub.

UI

- L'interface utilisateur (UI) de GitHub Actions est intégrée directement dans GitHub, offrant une expérience utilisateur fluide pour la gestion et la visualisation des workflows CI/CD. Voici un aperçu des principales fonctionnalités et sections de l'UI GitHub Actions que nous allons voir.

UI

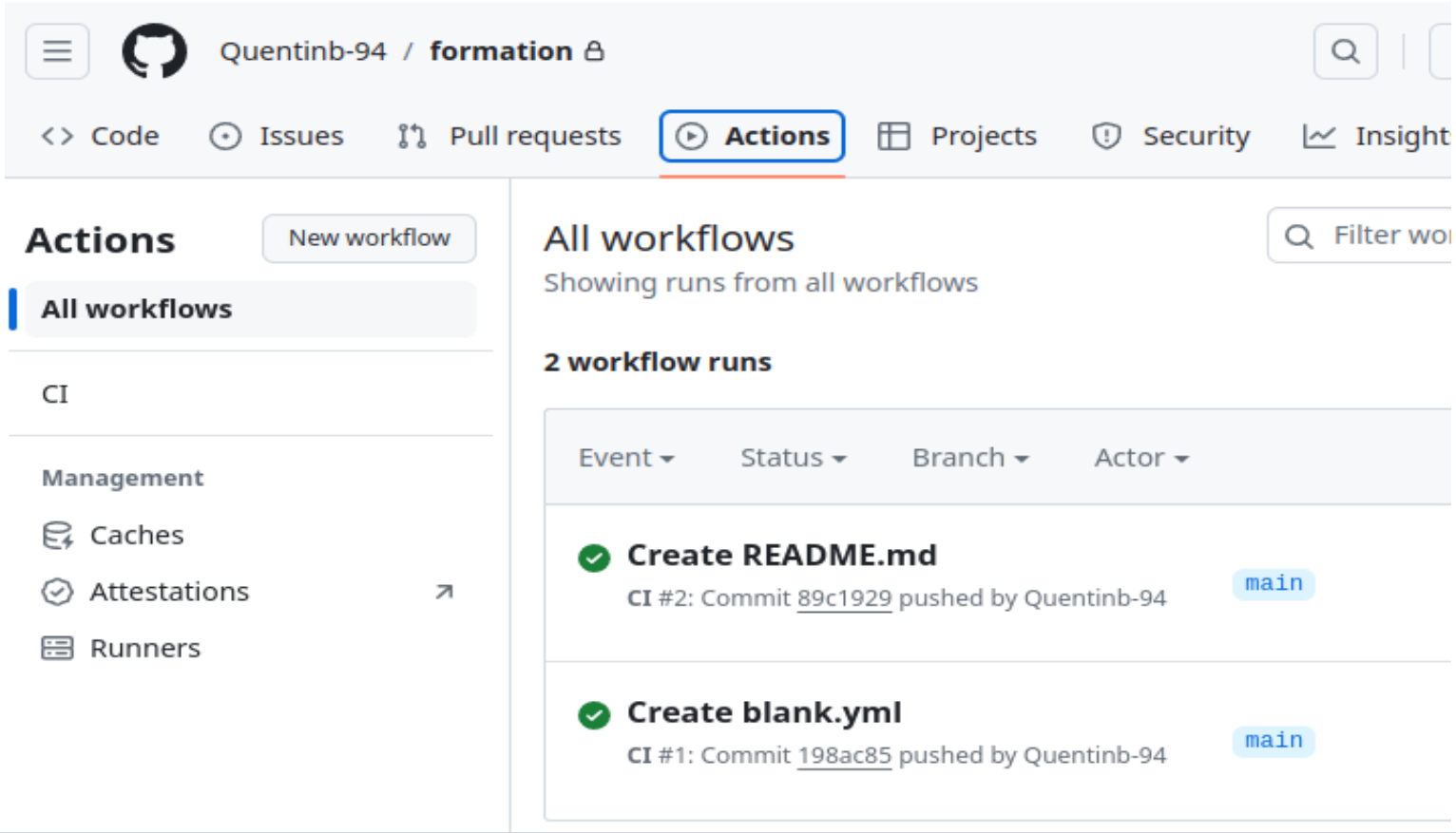
- Accès à l'UI GitHub Actions
- Pour accéder à l'interface GitHub Actions :
- Allez sur votre repository GitHub.
- Cliquez sur l'onglet "Actions" dans la barre de navigation supérieure.



UI

- Vue d'Ensemble des Workflows
- Dans la page principale des actions, vous pouvez voir tous les workflows configurés pour votre repository. Chaque workflow est listé avec les informations suivantes :
- **Nom du Workflow** : Le nom défini dans le fichier YAML.
- **Dernière Exécution** : La date et l'heure de la dernière exécution du workflow.
- **Statut** : Le statut de la dernière exécution (réussite, échec, en cours).

UI


Quentinb-94 / formation
Code
Issues
Pull requests
Actions
Projects
Security
Insight
Actions
New workflow
All workflows
CI
Management
Caches
Attestations
Runners
All workflows
Showing runs from all workflows
Filter wo
2 workflow runs
Event
Status
Branch
Actor
Create README.md
CI #2: Commit 89c1929 pushed by Quentinb-94
main
Create blank.yml
CI #1: Commit 198ac85 pushed by Quentinb-94
main

UI

- Détails d'un Workflow
- En cliquant sur un workflow spécifique, vous accédez à une vue détaillée montrant toutes les exécutions récentes de ce workflow :
- **Historique des Exécutions** : Liste des exécutions avec le statut, le commit associé, le déclencheur et l'utilisateur ayant initié l'action.
- **Filtrage** : Options pour filtrer les exécutions par statut (réussite, échec, en cours) ou par branche.

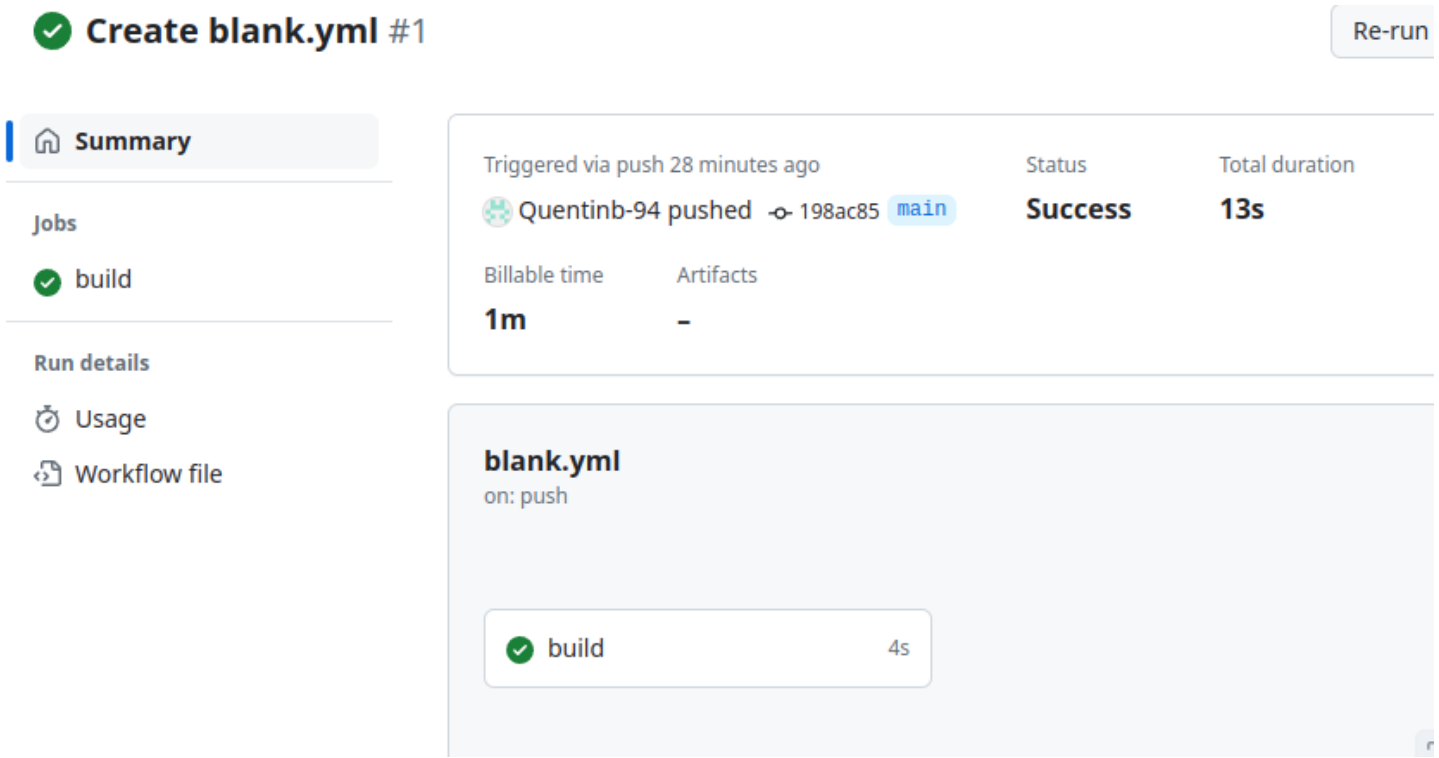
UI

Exécution d'un Workflow

En sélectionnant une exécution spécifique, vous obtenez une vue détaillée des jobs et des steps :

- **Jobs** : Liste des jobs définis dans le workflow. Chaque job affiche son statut et la durée d'exécution.
- **Steps** : Pour chaque job, vous pouvez voir les steps individuels avec des détails sur chaque commande exécutée, les logs de sortie et les erreurs éventuelles.

UI

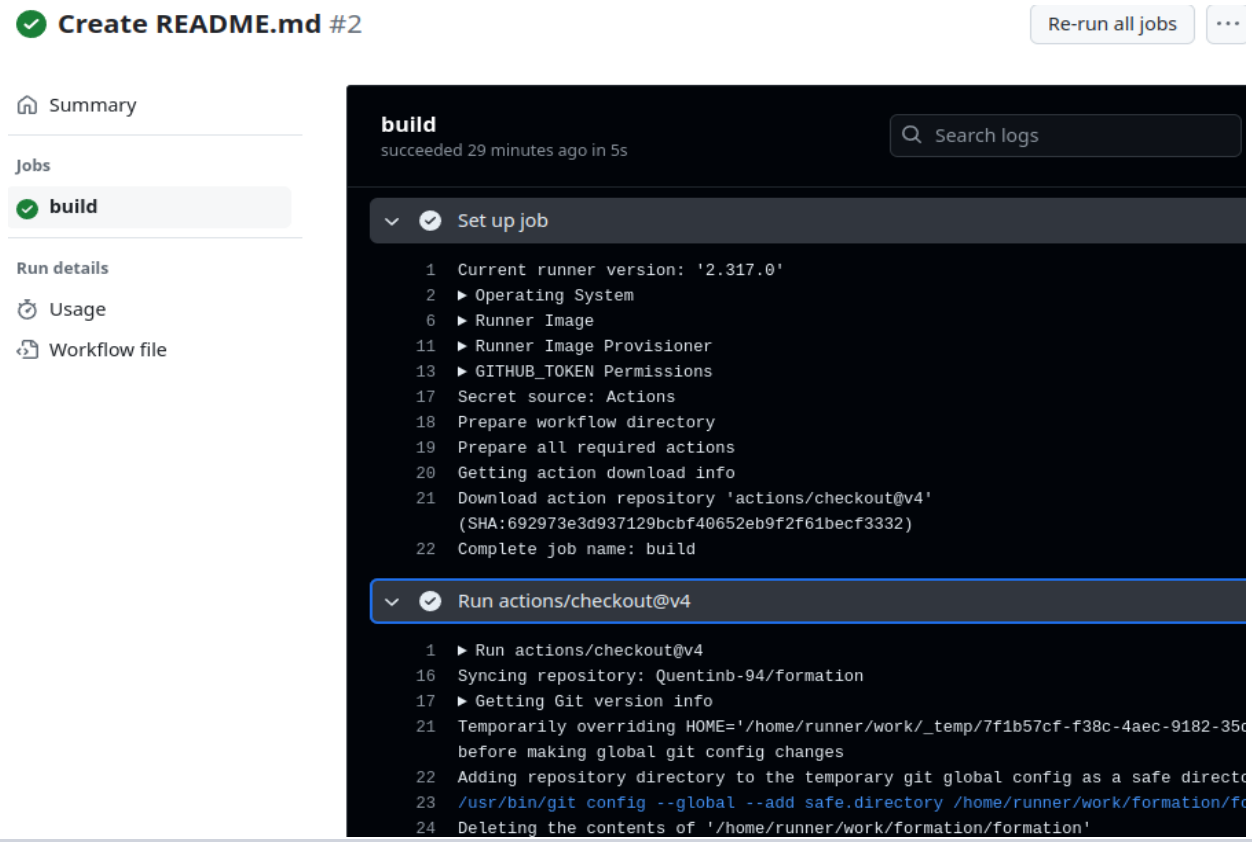

Create blank.yml #1
Re-run
Summary
Jobs
build
Run details
Usage
Workflow file
Triggered via push 28 minutes ago
Quentinb-94 pushed 198ac85 main
Status
Success
Total duration
13s
Billable time
1m
Artifacts
-
blank.yml
on: push
build
4s

UI

Visualisation des Logs

- Les logs de chaque step sont consultables directement dans l'interface :
- **Logs en Temps Réel** : Lorsqu'un workflow est en cours d'exécution, les logs se mettent à jour en temps réel.
- **Téléchargement des Logs** : Option pour télécharger les logs complets pour une analyse hors ligne.

UI


Create README.md #2
Re-run all jobs
Summary
Jobs
build
Run details
Usage
Workflow file
build
succeeded 29 minutes ago in 5s
Search logs
Set up job
1 Current runner version: '2.317.0'
2 ▶ Operating System
6 ▶ Runner Image
11 ▶ Runner Image Provisioner
13 ▶ GITHUB_TOKEN Permissions
17 Secret source: Actions
18 Prepare workflow directory
19 Prepare all required actions
20 Getting action download info
21 Download action repository 'actions/checkout@v4'
(SHA:692973e3d937129bcbf40652eb9f2f61becf3332)
22 Complete job name: build
Run actions/checkout@v4
1 ▶ Run actions/checkout@v4
16 Syncing repository: Quentinb-94/formation
17 ▶ Getting Git version info
21 Temporarily overriding HOME='/home/runner/work/_temp/7f1b57cf-f38c-4aec-9182-35c
before making global git config changes
22 Adding repository directory to the temporary git global config as a safe directo
23 /usr/bin/git config --global --add safe.directory /home/runner/work/formation/fo
24 Deleting the contents of '/home/runner/work/formation/formation'

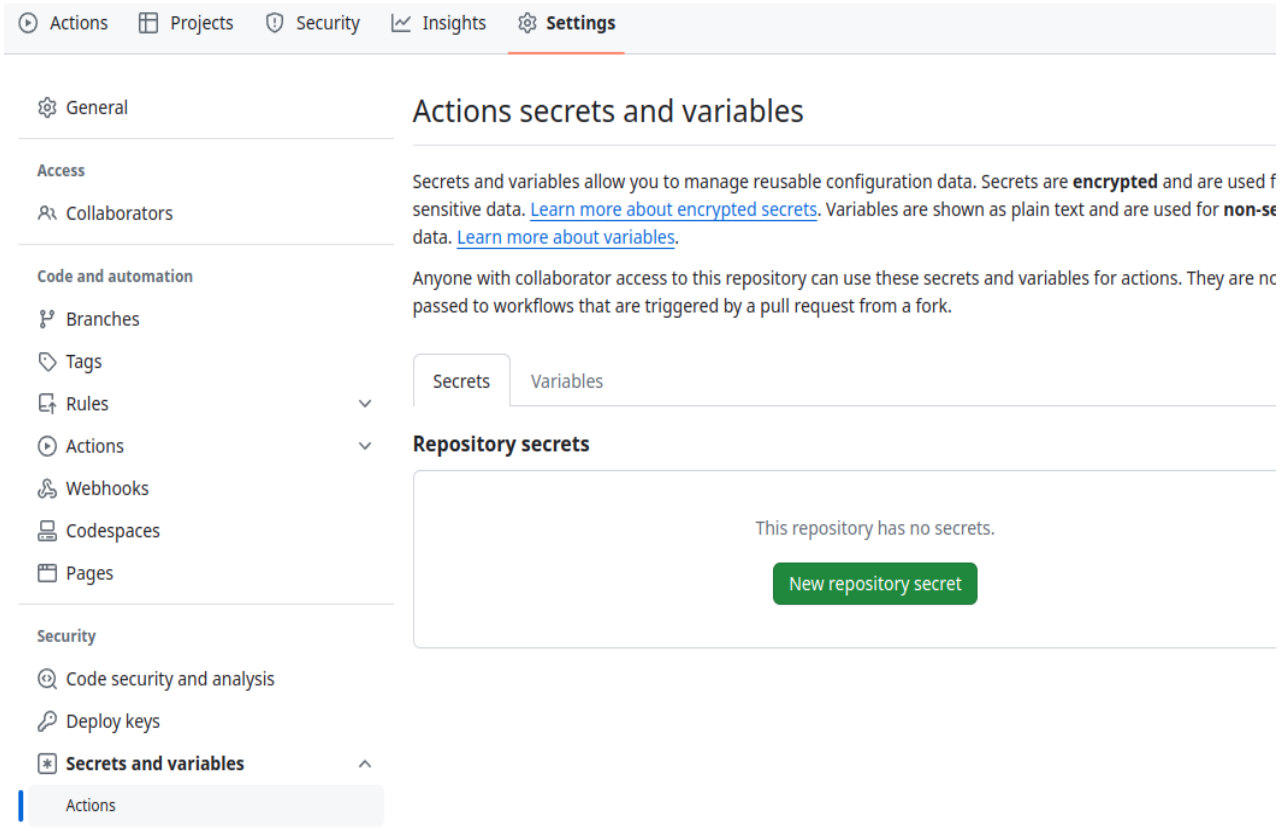
UI

Gestion des Secrets et Variables

Vous pouvez gérer les secrets et variables utilisés dans vos workflows depuis l'interface :

- **Secrets** : Accédez à cette section depuis les paramètres du repository (Settings > Secrets and variables > Actions).
- **Variables** : Définissez des variables d'environnement pour vos workflows (Settings > Secrets and variables > Actions).

UI


Actions
Projects
Security
Insights
Settings
General
Access
Collaborators
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Codespaces
Pages
Security
Code security and analysis
Deploy keys
Secrets and variables
Actions
Actions secrets and variables
Secrets and variables allow you to manage reusable configuration data. Secrets are encrypted and are used f
sensitive data. Learn more about encrypted secrets. Variables are shown as plain text and are used for non-se
data. Learn more about variables.
Anyone with collaborator access to this repository can use these secrets and variables for actions. They are nc
passed to workflows that are triggered by a pull request from a fork.
Secrets
Variables
Repository secrets
This repository has no secrets.
New repository secret

UI

Re-lancer un Workflow

L'interface permet de relancer facilement une exécution de workflow :

- **Re-run Job** : Option pour relancer un job spécifique en cas d'échec.
- **Re-run all jobs** : Option pour relancer tous les jobs d'un workflow.

UI

Interface de Configuration Visuelle (Beta)

GitHub propose une interface de configuration visuelle pour créer et modifier des workflows sans écrire de YAML :

- **Visual Editor** : Accédez à l'éditeur visuel via l'onglet Actions, puis cliquez sur "New workflow" ou modifiez un workflow existant.
- **Drag-and-Drop** : Créez des workflows en glissant-déposant des actions et en configurant leurs paramètres via une interface graphique.

UI

Marketplace des Actions

L'UI inclut un accès direct à GitHub Marketplace, où vous pouvez rechercher et intégrer des actions tierces :

- **Recherche** : Utilisez la barre de recherche pour trouver des actions spécifiques.
- **Catégories** : Parcourez les actions par catégorie (CI/CD, DevOps, etc.).
- **Détails des Actions** : Voir la documentation, les versions et les exemples d'utilisation pour chaque action.

UI

Deployment

View all

Deploy Node.js to Azure Web App
By Microsoft Azure

Build a Node.js project and deploy it to an Azure Web App.

Configure

Deployment

Deploy to Amazon ECS
By Amazon Web Services

Deploy a container to an Amazon ECS service powered by AWS Fargate or Amazon EC2.

Configure

Deployment

Build and Deploy to GKE
By Google Cloud

Build a docker container, publish it to Google Container Registry, and deploy to GKE.

Configure

Deployment

Terraform
By HashiCorp

Set up Terraform CLI in your GitHub Actions workflow.

Configure

Deployment

Deploy to Alibaba Cloud ACK
By Alibaba Cloud

Deploy a container to Alibaba Cloud Container Service for Kubernetes (ACK).

Configure

Deployment

Deploy to IBM Cloud Kubernetes Service
By IBM

Build a docker container, publish it to IBM Cloud Container Registry, and deploy to IBM Cloud Kubernetes Service.

Configure

Deployment

ATELIER : VOTRE PREMIER WORKFLOW

- Objectif :

Dans cet atelier, vous allez créer un workflow GitHub Actions simple pour automatiser des tests unitaires chaque fois que du code est poussé sur la branche principale (main) de votre repository. Nous allons utiliser un projet Node.js comme exemple, mais les concepts peuvent être adaptés à n'importe quel type de projet.

- Prérequis :

Un compte GitHub.

Un repository GitHub avec un projet Node.js. Si vous n'en avez pas, vous pouvez en créer un rapidement avec une simple application Node.js.

TP1_Workflow_1

ATELIER : VOTRE PREMIER WORKFLOW

- Si vous ne pouvez avoir de ressources à disposition, on va chercher plus simple, mettre en place un workflow qui vérifiera simplement que le code se compile sans erreur chaque fois que vous effectuez un push sur la branche principale (main).

- Prérequis :

Un compte GitHub.

Un repository GitHub

TP1_Workflow_2

ATELIER : VOTRE PREMIER WORKFLOW

- Et encore mieux, mettre en place un Workflow de votre propre conception.

- Prérequis :

Un compte GitHub.

Un repository GitHub

Votre talent et imagination

MODULE 2 : INTERAGIR AVEC L'ENVIRONNEMENT GITHUB

- Les variables d'environnement
- Les Secrets
- Github Runner

LES VARIABLES D'ENVIRONNEMENT

Les variables d'environnement sous GitHub Actions permettent de personnaliser et de sécuriser les workflows en stockant des valeurs utilisées lors de l'exécution des jobs. Il existe plusieurs façons de définir et d'utiliser ces variables dans vos workflows.

LES VARIABLES D'ENVIRONNEMENT

- **Types de Variables d'Environnement**

1. **Variables d'Environnement Définies Globalement** : Affectées pour l'ensemble du workflow.
2. **Variables d'Environnement au Niveau du Job** : Spécifiques à un job particulier.
3. **Variables d'Environnement au Niveau de l'Étape** : Spécifiques à une étape particulière.
4. **Secrets** : Valeurs sensibles sécurisées.
5. **Variables Contextuelles** : Informations fournies automatiquement par GitHub.

LES VARIABLES D'ENVIRONNEMENT

- **Définir des Variables d'Environnement**
- **Variables Globales**
- Les variables globales sont définies dans le bloc env au niveau du workflow.

LES VARIABLES D'ENVIRONNEMENT

```yaml
name: CI Workflow

on: [push]

env:
  GLOBAL_VAR: "vars Global"

jobs:
  example-job:
    runs-on: ubuntu-latest
    steps:
      - name: Print global variable
        run: echo $GLOBAL_VAR
```

LES VARIABLES D'ENVIRONNEMENT

- **Variables au Niveau du Job**
- Les variables spécifiques à un job sont définies dans le bloc env du job.

```yaml
jobs:
  example-job:
    runs-on: ubuntu-latest
    env:
      JOB_VAR: "This is a job-specific variable"
    steps:
      - name: Print job variable
        run: echo $JOB_VAR
```

LES VARIABLES D'ENVIRONNEMENT

- Variables au Niveau de l'Étape (Step)
- Les variables spécifiques à une étape sont définies directement dans le bloc env de l'étape.

```yaml
jobs:
  example-job:
    runs-on: ubuntu-latest
    steps:
      - name: Print step variable
        env:
          STEP_VAR: "This is a step-specific variable"
        run: echo $STEP_VAR
```

LES VARIABLES D'ENVIRONNEMENT

- Variables Contextuelles
- GitHub Actions fournit un ensemble de variables contextuelles accessibles via ${{ }}.
- Exemples de Contextes Courants
- **github** : Informations sur le repository, l'événement et l'acteur.
- **env** : Variables d'environnement définies dans le workflow.
- **job** : Informations sur le job en cours.
- **steps** : Informations sur les étapes précédentes.
- **runner** : Informations sur le runner.

LES VARIABLES D'ENVIRONNEMENT

```yaml
jobs:
  example-job:
    runs-on: ubuntu-latest
    steps:
      - name: Print GitHub context
        run: echo "This run was triggered by ${{ github.actor }} on ${{ github.ref }}"
```

LES VARIABLES D'ENVIRONNEMENT

Petit résumé sur ce que nous avons vu avant de passer à l'étape suivante.

- Il faut savoir que les variables d'environnement sous GitHub Actions permettent une flexibilité et une sécurité accrues lors de l'automatisation des workflows. En les définissant globalement, au niveau des jobs ou des étapes, et en utilisant des secrets pour les données sensibles, vous pouvez adapter vos workflows aux besoins spécifiques de votre projet tout en maintenant la sécurité et la maintenabilité du code. Les variables contextuelles fournissent des informations dynamiques sur le workflow et l'environnement d'exécution, rendant les actions encore plus puissantes et adaptables.

LES SECRETS

Les secrets dans GitHub Actions sont des valeurs sensibles comme des tokens API, des clés de déploiement, des mots de passe, etc., qui sont stockées de manière sécurisée et peuvent être utilisées dans les workflows sans être exposées dans le code source. Voici une explication détaillée sur la manière de gérer et d'utiliser les secrets dans GitHub Actions.

LES SECRETS

Pour ajouter un secret à un repository GitHub :

1. **Accédez aux paramètres du repository** :
1. Allez sur la page de votre repository sur GitHub.
2. Cliquez sur l'onglet "Settings" (Paramètres).
2. **Naviguez vers les secrets** :
1. Dans le menu latéral gauche, trouvez et cliquez sur "Secrets and variables" sous la section "Security".
2. Sélectionnez "Actions".
3. **Ajouter un nouveau secret** :
1. Cliquez sur "New repository secret".
2. Entrez un nom pour votre secret dans le champ "Name".
3. Entrez la valeur du secret dans le champ "Value".
4. Cliquez sur "Add secret" pour enregistrer le secret.

LES SECRETS

Utiliser des Secrets dans un Workflow

- Une fois le secret ajouté, vous pouvez l'utiliser dans votre workflow GitHub Actions en accédant à la variable de secrets avec ${{ secrets.<secret_name> }}.

Exemple de Workflow utilisant des Secrets

- Voici un exemple simple montrant comment utiliser un secret pour déployer une application ou exécuter une commande sécurisée

```yaml
name: Deploy Application

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Use deployment secret
        env:
          DEPLOY_KEY: ${{ secrets.DEPLOY_KEY }}
        run: |
          echo "Using the deploy key: $DEPLOY_KEY"
          # Exemple d'utilisation d'un secret pour exécuter une commande sécurisée
          ./deploy.sh --key $DEPLOY_KEY
```

LES SECRETS

Dans cet exemple :

- Un secret nommé DEPLOY_KEY est utilisé.
- La valeur du secret est accessible via ${{ secrets.DEPLOY_KEY }} et assignée à une variable d'environnement DEPLOY_KEY.
- La commande de déploiement utilise cette variable pour effectuer une opération sécurisée.

LES SECRETS

Best pratices :

- **Limiter l'accès** :
  - Ne partagez pas les secrets entre trop de personnes ou de services.
  - Utilisez des permissions pour restreindre l'accès aux secrets seulement aux membres nécessaires de l'équipe.
- **Utiliser des Secrets de l'Organisation** :
  - Si vous avez plusieurs repositories nécessitant le même secret, utilisez des secrets au niveau de l'organisation pour centraliser la gestion des secrets.
- **Rotation des Secrets** :
  - Changez régulièrement les valeurs des secrets pour améliorer la sécurité.
  - Utilisez des secrets temporaires quand c'est possible.
- **Ne pas exposer les Secrets** :
  - Évitez d'imprimer les secrets dans les logs ou de les exposer dans le code source.
  - Utilisez des vérifications pour s'assurer que les secrets ne sont pas inclus accidentellement dans les messages de commit ou les fichiers de configuration.

LES SECRETS

Secrets de l'Organisation

Pour les projets au niveau de l'organisation, vous pouvez également gérer les secrets au niveau de l'organisation, ce qui permet de les partager entre plusieurs repositories

LES SECRETS

Accédez aux paramètres de l'organisation :

- Allez sur la page de votre organisation sur GitHub.
- Cliquez sur "Settings" (Paramètres).

Naviguez vers les secrets :

- Dans le menu latéral gauche, trouvez et cliquez sur "Secrets and variables" sous la section "Security".
- Sélectionnez "Actions".

Ajouter un nouveau secret d'organisation :

- Cliquez sur "New organization secret".
- Entrez un nom et une valeur pour le secret.
- Configurez les permissions pour spécifier quels repositories peuvent accéder à ce secret.
- Cliquez sur "Add secret" pour enregistrer le secret.

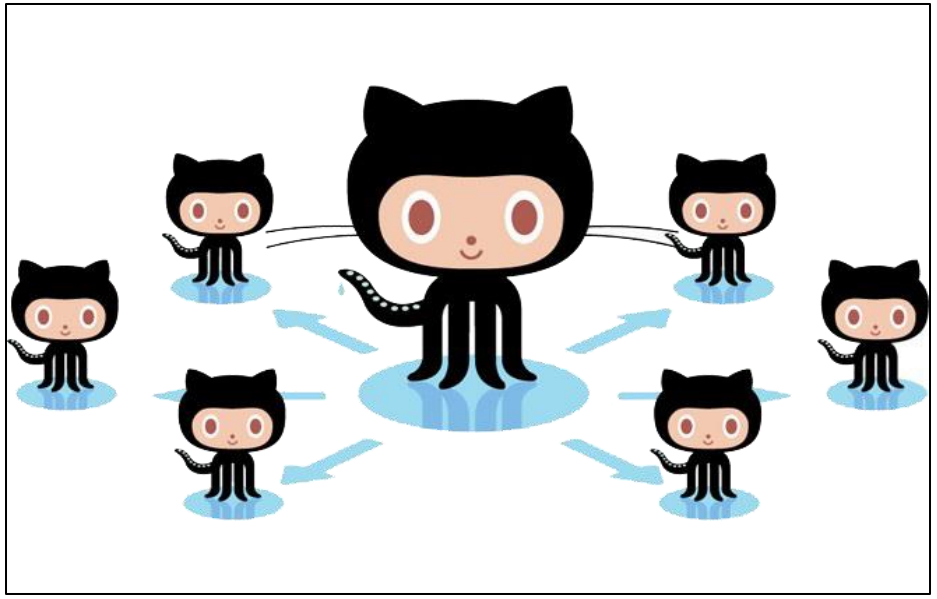
LES SECRETS

```yaml
name: Advanced Deploy Workflow

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '14'

      - name: Install dependencies
        run: npm install

      - name: Run tests
        run: npm test

      - name: Build project
        run: npm run build

      - name: Deploy to server
        env:
          DEPLOY_KEY: ${{ secrets.DEPLOY_KEY }}
          API_TOKEN: ${{ secrets.API_TOKEN }}
        run: |
          echo "Deploying with key: $DEPLOY_KEY and token: $AP
          ./deploy.sh --key $DEPLOY_KEY --token $API_TOKEN
```

GITHUB RUNNER

Les GitHub Runners sont des serveurs sur lesquels s'exécutent les commandes spécifiées dans vos workflows.

GitHub propose deux types de runners : les runners hébergés par GitHub et les runners auto-hébergés.



GITHUB RUNNER

1 - Runners Hébergés par GitHub

Les runners hébergés par GitHub sont des machines virtuelles fournies et gérées par GitHub. Ils sont prêts à l'emploi et configurés pour exécuter vos jobs sans nécessiter de configuration supplémentaire de votre part. Voici quelques caractéristiques des runners hébergés par GitHub :

1) Praticité :

- **Pré-configurés** : Ils sont pré-configurés avec des environnements populaires et couramment utilisés comme Ubuntu, Windows, et macOS, incluant divers outils de développement (par exemple, Node.js, Python, Java).
- **Maintenance** : GitHub s'occupe de la maintenance, des mises à jour de sécurité, et de la gestion des ressources des runners.

GITHUB RUNNER

2) Scalabilité :

- **Disponibilité** : GitHub garantit une disponibilité élevée et une montée en charge automatique selon la demande de vos workflows.
- **Isolation** : Chaque job s'exécute dans une machine virtuelle isolée, assurant ainsi un environnement propre et sécuritaire à chaque exécution.

3) Coût :

- **Gratuité limitée** : Les runners hébergés par GitHub sont gratuits jusqu'à un certain quota mensuel. Au-delà de ce quota, des frais supplémentaires peuvent s'appliquer selon votre plan GitHub (gratuit, Pro, Team, ou Enterprise).

GITHUB RUNNER

Exemple d'utilisation de runners hébergés par GitHub :

```yaml
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2

      - name: Set up Node.js
        uses: actions/setup-node@v2

        with:
          node-version: '14'

      - name: Install dependencies
        run: npm install

      - name: Run tests
        run: npm test
```

2 - Runners Auto-Hébergés

Les runners auto-hébergés sont des machines que vous configurez et gérez vous-même. Vous avez la flexibilité de personnaliser l'environnement d'exécution selon vos besoins spécifiques. Voici quelques avantages et caractéristiques des runners auto-hébergés

1) **Contrôle :**

- **Personnalisation** : Vous avez un contrôle total sur l'environnement, y compris le système d'exploitation, les logiciels installés, et la configuration matérielle.
- **Configuration** : Vous pouvez installer des dépendances spécifiques, configurer des outils personnalisés, et ajuster les paramètres système selon les exigences de vos workflows.

GITHUB RUNNER

2) Performance :

- **Ressources dédiées** : Les runners auto-hébergés utilisent les ressources de vos propres serveurs, permettant une exécution plus rapide et plus fiable pour des tâches intensives.
- **Pas de quotas** : Contrairement aux runners hébergés, il n'y a pas de quotas imposés par GitHub, ce qui est idéal pour des projets nécessitant une exécution fréquente ou intensive.

3) Sécurité :

- **Isolation** : Vous pouvez isoler les runners au sein de votre propre réseau ou infrastructure de sécurité, assurant ainsi une gestion plus stricte des données sensibles et des accès réseau.
- **Conformité** : Les runners auto-hébergés permettent de se conformer à des politiques internes ou réglementaires spécifiques en matière de sécurité et de confidentialité.

Exemple de configuration d'un runner auto-hébergé :

1) Téléchargez le package du runner depuis GitHub.

curl -o actions-runner-linux-x64-2.277.1.tar.gz -L
https://github.com/actions/runner/releases/download/v2.277.1/actions-runner-linux-x64-2.277.1.tar.gz

GITHUB RUNNER

2) Décompressez le package.

tar xzf ./actions-runner-linux-x64-2.277.1.tar.gz

3) Configurez le runner avec le token fourni par GitHub.

./config.sh --url https://github.com/mon-org/mon-repo --token MON_TOKEN

4) Démarrez le runner.

./run.sh

GITHUB RUNNER

- **Utilisation d'un runner auto-hébergé dans un workflow :**

Pour utiliser un runner auto-hébergé dans un workflow, spécifiez `runs-on: self-hosted`

```yaml
jobs:
  build:
    runs-on: self-hosted
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2
      - name: Build project
        run: make build
```

Cela permet d'exécuter le job sur une machine que vous contrôlez complètement.

GITHUB RUNNER

En résumé, le choix entre runners hébergés et auto-hébergés dépend de vos besoins en termes de personnalisation, de performance, de sécurité, et de coûts. Les runners hébergés par GitHub offrent une solution clé en main pratique pour la plupart des cas d'utilisation, tandis que les runners auto-hébergés fournissent une flexibilité maximale pour des environnements de travail spécifiques.

Atelier Pratique

- **Exercice 1 :** Définissez et utilisez des variables d'environnement dans un workflow.
- **Exercice 2 :** Ajoutez des secrets à votre repository et utilisez-les dans un workflow.
- **Exercice 3 :** Configurez un runner auto-hébergé et exécutez un workflow sur ce runner.
- **Discussion :** Partagez les erreurs rencontrées et résolvez-les en groupe si besoin est.

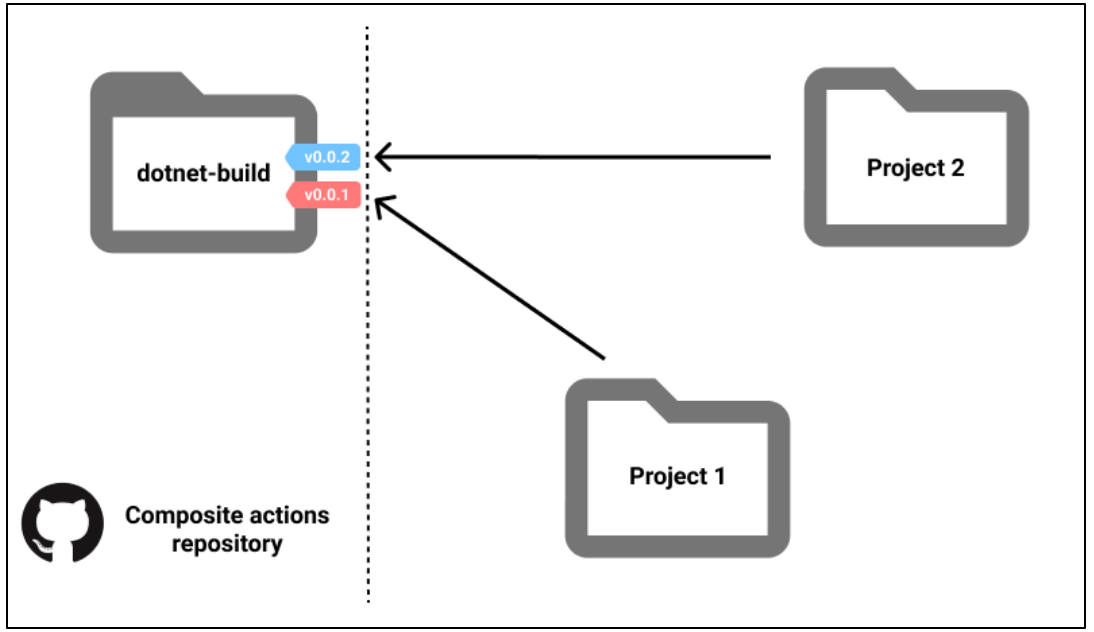
MODULE 3 : WORKFLOWS RÉUTILISABLES

- Concept
- Utilisation

CONCEPT

Les Workflows Réutilisables :

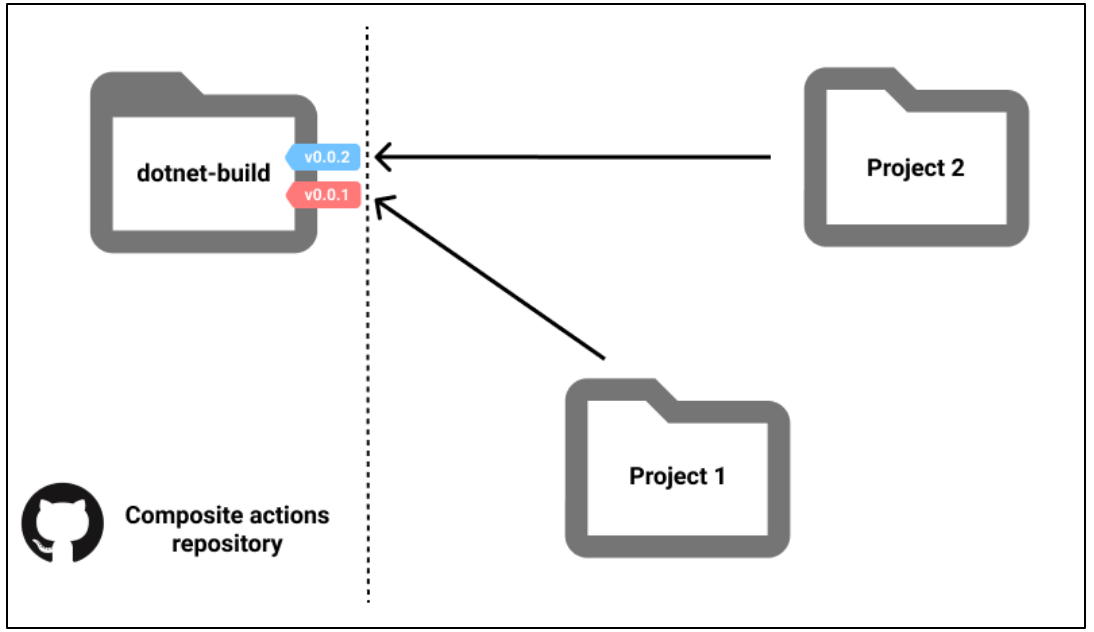
Les workflows réutilisables permettent de centraliser des logiques de workflow répétitives dans un seul fichier et de les réutiliser dans différents projets, ce qui simplifie la maintenance et réduit la duplication du code. Ils sont particulièrement utiles pour des tâches communes comme les tests, la compilation et le déploiement.



CONCEPT

Les Workflows Réutilisables :

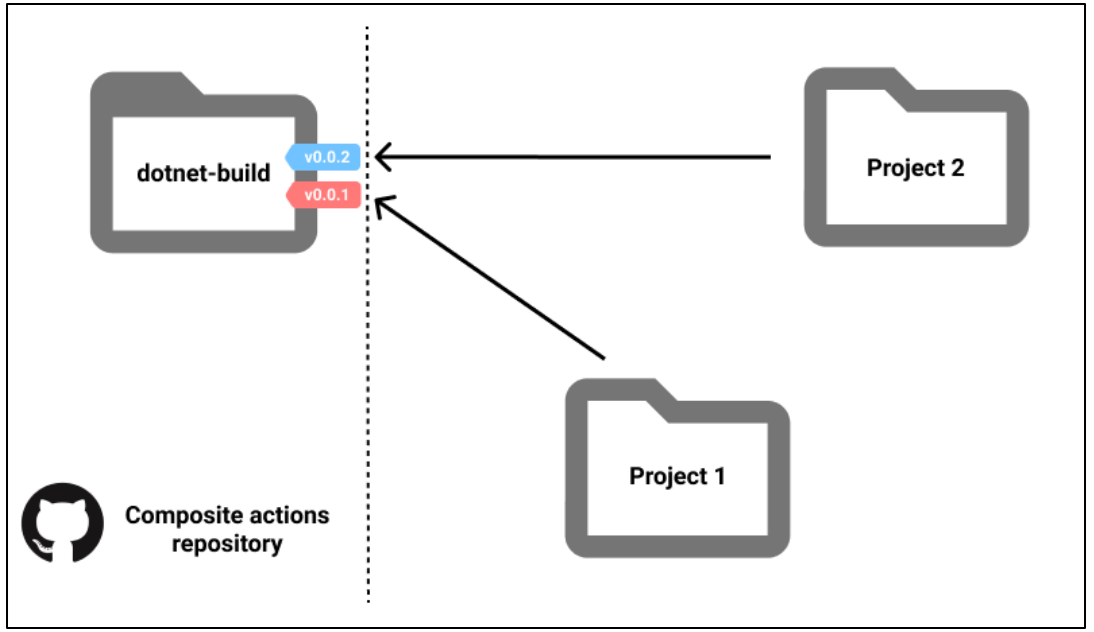
Pour résumé au lieu de copier et de coller d'un workflow vers un autre, vous pouvez rendre les workflows réutilisables. Vous et toute personne ayant accès au workflow réutilisable peut alors appeler le workflow réutilisable à partir d'un autre workflow



CONCEPT

Les Workflows Réutilisables :

La réutilisation de workflows évite la duplication. Cela facilite la maintenance des workflows et vous permet de créer plus rapidement de nouveaux workflows en s'appuyant sur le travail des autres, comme vous le faites avec des actions. La réutilisation des workflows favorise également les bonnes pratiques en vous aidant à utiliser des workflows bien conçus, qui ont déjà été testés et qui se sont avérés efficaces. Votre organisation peut créer une bibliothèque de workflows réutilisables qui peuvent être gérés de manière centralisée.



Avantages

- **Réduction de la Duplication :** Un workflow peut être utilisé par plusieurs dépôts, ce qui évite de réécrire les mêmes étapes à chaque fois.
- **Facilité de Maintenance :** Les modifications apportées à un workflow réutilisable sont appliquées automatiquement à tous les dépôts qui l'utilisent, facilitant la mise à jour.
- **Cohérence :** Assure que les mêmes étapes et configurations sont utilisées dans différents projets, garantissant des processus uniformes.

Définir un Workflow

Un workflow réutilisable est défini avec le déclencheur `workflow_call`, ce qui permet à d'autres workflows de l'appeler. Voici un exemple :

```yaml
name: Reusable Workflow
on: workflow_call
jobs:
  example-job:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2
      - name: Installer les dépendances
        run: npm install
      - name: Lancer les tests
        run: npm test
```

Ce workflow peut être appelé depuis d'autres repositories.

Utiliser un Workflow Réutilisable

Pour utiliser un workflow réutilisable, il suffit de l'appeler avec la directive `uses` dans un autre workflow :

```yaml
jobs:
  call-reusable-workflow:
    uses: organisation/repository/.github/workflows/reusable-workflow.yml@main
    with:
      example-input: 'valeur'
```

Cela intègre le workflow défini dans `reusable-workflow.yml` au sein du workflow appelant.

Configurer les Entrées et Sorties

Pour rendre les workflows réutilisables plus flexibles, vous pouvez définir des entrées (**inputs**) et des sorties (**outputs**). Voici un exemple :

```yaml
on: workflow_call:
  inputs:
    example-input:
      description: 'Un exemple d'entrée'
      required: true
jobs:
  example-job:
```

ATELIER : UTILISER UN REUSABLE WORKFLOW

- Objectif :

Les workflows réutilisables dans GitHub Actions permettent de centraliser et de réutiliser les définitions de workflows complexes à travers plusieurs repositories. Cela améliore la maintenabilité et la cohérence des workflows CI/CD

- Créer un workflow réutilisable.
- Appeler ce workflow réutilisable depuis un autre workflow.
- Passer des paramètres au workflow réutilisable.

- Prérequis :

Un compte GitHub.

Deux repositories GitHub (ou un repository avec deux branches) pour démontrer l'utilisation du workflow réutilisable

TP2_Workflow_reusable

ATELIER : UTILISER UN REUSABLE WORKFLOW

- Vous pouvez aussi partir sur votre propre façon de faire afin de tester.

- https://docs.github.com/fr/actions/using-workflows/reusing-workflows

MODULE 4 : MARKETPLACE GITHUB ACTIONS

- Utilisation
- Les indispensables
- Les risques

UTILISATION

La Marketplace GitHub Actions est une plateforme où vous pouvez trouver et utiliser des actions créées par la communauté GitHub. Elle vous permet d'intégrer facilement des fonctionnalités dans vos workflows sans avoir à les coder vous-même. Vous pouvez rechercher des actions par mots-clés, vérifier leur popularité, et consulter les avis des utilisateurs.

UTILISATION

- **Exemple**: Pour utiliser une action de la marketplace :
- Recherchez l'action désirée sur la Marketplace GitHub Actions.
- Consultez la documentation de l'action pour comprendre son utilisation.
- Ajoutez l'action dans votre workflow YAML :
- .

```yaml
jobs:
  example-job:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v2

      - name: Utiliser une action de la marketplace
        uses: actions/setup-node@v2

        with:
          node-version: '14'
```

LES INDISPENSABLES

Certaines actions de la marketplace sont particulièrement utiles et largement utilisées dans les workflows GitHub Actions :

- **actions/checkout** : Clone votre dépôt dans le runner.
- **actions/setup-node** : Configure l'environnement Node.js.
- **actions/cache** : Met en cache les dépendances pour accélérer les builds.
- **actions/upload-artifact** : Permet de sauvegarder et de partager les artefacts produits par les workflows.
- **actions/download-artifact** : Télécharge les artefacts stockés.

LES INDISPENSABLES

Exemple :

```yaml
jobs:
  buid:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2

      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14'

      - name: Install dependencies
        run: npm install

      - name: Cache dependencies
        uses: actions/cache@v2
        with:
          path: ~/.npm
          key: ${{ runner.os }}-node-${{ hashFiles('**/package-lock.jso
          restore-keys: |
            ${{ runner.os }}-node-

      - name: Run tests
        run: npm test

      - name: Upload test results
        uses: actions/upload-artifact@v2
        with:
          name: test-results
          path: ./test-results
```

LES RISQUES

L'utilisation d'actions tierces comporte certains risques :

- **Sécurité** : Les actions provenant de sources non vérifiées peuvent contenir du code malveillant.
- **Maintenance** : Certaines actions peuvent ne plus être maintenues, entraînant des incompatibilités avec les nouvelles versions de GitHub Actions.
- **Fiabilité** : Les actions créées par des développeurs tiers peuvent ne pas offrir le même niveau de fiabilité que les actions officielles.

LES RISQUES

Bonnes pratiques:

- **Vérifiez l'authenticité** des actions en utilisant celles provenant de sources réputées.
- **Lisez les avis** et vérifiez la fréquence des mises à jour.
- **Testez les actions** dans un environnement contrôlé avant de les intégrer dans vos workflows de production.

ATELIER PRATIQUE

- **Exercice 1** : Recherchez et utilisez une action de la marketplace pour configurer un environnement de test.
- **Exercice 2** : Utilisez une action pour mettre en cache les dépendances dans votre workflow.
- **Exercice 3** : Téléchargez et utilisez une action pour déployer une application.
- **Discussion** : Partagez vos expériences et les défis rencontrés avec l'utilisation des actions de la marketplace.

FIN DE LA 1ERE PARTIE

Merci et à demain !