

Jérôme BOEBION

# GitHub Actions

[HTTPS://DOCS.GITHUB.COM/FR/ACTIONS](https://docs.github.com/fr/actions)



## GitHub Actions

# Vue d'ensemble

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.

- Créer des workflows qui testent chaque demande de pull request adressée à votre dépôt, ou déploie des demandes pull merge en production.
- 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 un problème dans votre dépôt.
- Pour cela, 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.

# Composants de GitHub Actions

Votre workflow contient **un ou plusieurs travaux** qui peuvent s'exécuter dans **un ordre séquentiel ou en parallèle**.

Syntaxe de Yaml :

<https://blog.stephane-robert.info/docs/developper/autres-langages/yaml/introduction/>

## ■ Workflows

- 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 GitHub** et s'exécutent lorsqu'ils sont déclenchés par un événement dans votre dépôt. *[ou ils peuvent être déclenchés manuellement ou selon une planification définie].*

## ■ Événements

- Un événement est une activité spécifique dans un dépôt qui déclenche l'exécution d'un workflow.
- <https://docs.github.com/fr/actions/using-workflows/events-that-trigger-workflows>

## ■ Travaux (Jobs)

- Ensemble d'étapes dans un workflow qui s'exécute sur le même **exécuteur**. Chaque étape est un **script d'interpréteur de commandes ou une action qui seront exécutée**.
  - Les étapes sont exécutées dans l'ordre et dépendent les unes des autres.
  - Comme chaque étape est exécutée sur le même exécuteur, vous pouvez partager des données d'une étape à une autre.

# Composants de GitHub Actions

Chaque **travail** s'exécute au sein de son propre **exécuteur** (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.

## ■ Actions

- Une action est une application personnalisée pour la plateforme GitHub Actions qui effectue une tâche complexe mais fréquemment répétée.
- Utilisez une action permet de réduire la quantité de code répétitif que vous écrivez dans vos fichiers de workflow.

## ■ Script d'interpréteur de commandes

- Selon l'environnement de l'exécuteur, il s'agit d'un ensemble de commandes réalisant une action (par exemple [mvn package](#))

## ■ Exécuteurs (environnement)

- Un exécuteur est un serveur qui exécute vos workflows quand ils sont déclenchés. Chaque exécuteur peut exécuter un seul travail à la fois.
- [GitHub fournit les exécuteurs](#) Ubuntu Linux, Microsoft Windows et macOS [pour exécuter vos workflows](#).
- Chaque exécution de workflow s'exécute sur une machine virtuelle nouvellement provisionnée.

# Composants de GitHub Actions

Parmi les actions, nous pouvons retrouver des **checkout** et utiliser des **variables secrètes** permettant de respecter la confidentialité de la configuration de nos workflows

## ■ Checkout

- L'action **Checkout** est l'une des actions les plus couramment utilisées dans les workflows GitHub Actions. Elle permet de cloner le code du projet dans l'environnement du workflow, rendant ainsi le code source disponible pour les étapes suivantes.
- Cela garantit que votre workflow dispose toujours de la dernière version du code source avant de commencer à exécuter les jobs.

## ■ Secrets

- Les secrets sont des variables sensibles stockées de manière sécurisée et utilisées dans les workflows.
- Ils sont généralement utilisés pour stocker des informations sensibles telles que des clés d'API, des jetons d'accès ou des mots de passe.
- GitHub Actions offre un espace de stockage sécurisé pour ces secrets.

# Composants de GitHub Actions

Enfin, nous allons pouvoir mettre en place des **conditions**, afin de réaliser nos workflows selon le comportement du dépôt.

Le **marketplace** est une ressource de partage de la communauté GitHub.

## ■ Conditions

- GitHub Actions permet d'utiliser des conditions pour déterminer si un job doit être exécuté en fonction d'une évaluation.
- Par exemple, vous pouvez définir des conditions pour exécuter un job uniquement si une certaine branche est modifiée, si un pull request a été ouvert, ou si d'autres critères spécifiques sont remplis.
- Cela vous permet de personnaliser davantage le comportement de vos workflows en fonction des événements et des circonstances..

## ■ Marketplace GitHub Actions

- Le Marketplace GitHub Actions est une ressource précieuse qui offre un vaste choix d'actions pré-construites que vous pouvez intégrer dans vos workflows. Ces actions sont créées par la communauté GitHub et couvrent une variété de cas d'usage.
- <https://github.com/marketplace?type=actions> : Vous y trouverez une liste d'actions prédéfinies, triées par catégorie et par popularité. Vous pouvez parcourir ces actions pour trouver celles qui correspondent le mieux à vos besoins.

# Les exécuteurs

**Runs-on** permet de définir le type de machine sur laquelle le travail doit être exécuté.

<https://docs.github.com/fr/actions/using-jobs/choosing-the-runner-for-a-job>

- Si vous utilisez un exécuteur hébergé par GitHub, chaque travail s'exécute sur une nouvelle instance d'une image d'exécuteur spécifiée par le mot-clé **runs-on**.
- Les étiquettes d'exécuteurs hébergés par GitHub disponibles sont les suivantes :
  - **Ubuntu-latest**, **ubuntu-22.04**, ubuntu-20.04
  - **Windows-latest**, **windows-2022**, windows-2019
  - **Macos-latest**, **macos-12**, macos-11
- Mais il est possible d'avoir ses propres exécuteurs. Ainsi, l'étiquette d'exécuteurs auto-hébergé est la suivante :
  - **Self-hosted**

# Création d'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).
- Exemple d'un workflow

```
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@v3
        with:
          node-version: '14'
      - run: npm install -g bats
      - run: bats -v
```



# Description du fichier de workflow

Pour vous aider à comprendre comment la syntaxe YAML est utilisée pour créer un fichier de workflow, cette section explique chaque ligne de l'exemple d'introduction.

```
# Optional - le nom du workflow qui apparaîtra dans le dépôt
name: learn-github-actions
# affiche dans le contexte du workflow, le nom de l'utilisateur déclencheur
run-name: ${github.actor} is learning GitHub Actions
# spécifie le déclencheur
on: [push]


# Groupes des travaux
jobs:

# Définie le nom du Job et ses propriétés
check-bats-version:

# Configure le job sur la dernière version de Ubuntu
runs-on: ubuntu-latest

# Les étapes du jobs
steps:

# Scripts ou Actions

# Le mot-clé « uses » indique que cette étape exécutera
# « v4 » de l'action « actions/checkout ».
#  s'agit d'une action qui vérifie votre référentiel sur le runner, vous permettant d'exécuter des scripts ou
# d'autres actions contre votre code (comme les outils de construction et de test).
# Vous devez utiliser l'action de vérification chaque fois que votre flux de travail utilisera le code du référentiel.
- uses: actions/checkout@v4

# Cette étape utilise l'action 'actions/setup-node@v3' pour installer la version spécifiée de Node.js.
#(Cet exemple utilise la version 14.) Ceci place les commandes 'node' et 'npm' dans votre 'PATH'.
- uses: actions/setup-node@v3
  with:
    node-version: '14'

# exécution d'une installation de bats
- run: npm install -g bats

# exécution de bats pour afficher la version
- run: bats -v
```

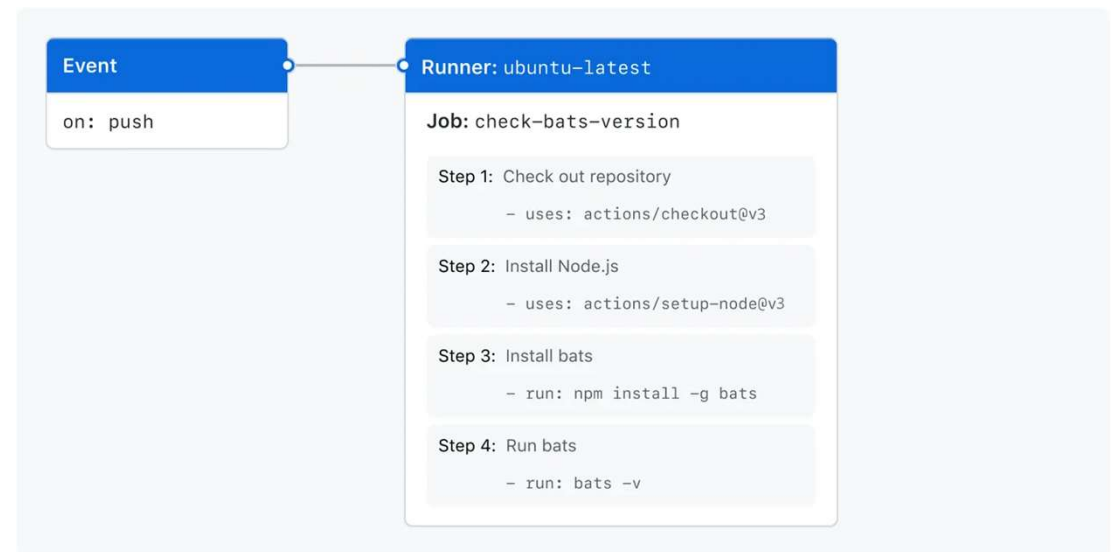
# Visualisation du fichier de workflow

Dans ce diagramme, vous pouvez voir le fichier de workflow que vous venez de créer et comment les composants GitHub Actions sont organisés dans une hiérarchie.

Chaque étape exécute une action ou un script d'interpréteur de commandes unique.

Les étapes 1 et 2 exécutent des actions, tandis que les étapes 3 et 4 exécutent des scripts d'interpréteur de commandes.

<https://docs.github.com/fr/actions/learn-github-actions/finding-and-customizing-actions>

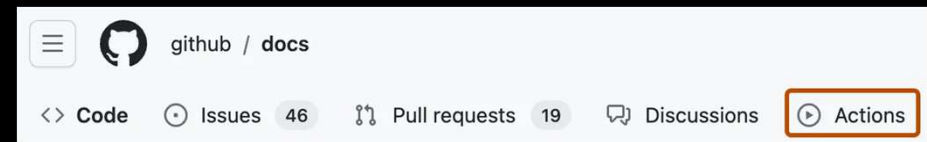


# Affichage de l'activité dans GitHub

Lorsque votre workflow est déclenché, une *exécution de workflow* est créée et exécute le workflow.

Une fois l'exécution de votre workflow démarrée, vous pouvez voir un graphe de visualisation de la progression de l'exécution, ainsi que l'activité de chaque étape sur GitHub.

1. Sur Github.com, accédez à la page principale du dépôt.
2. Sous le nom de votre dépôt, cliquez sur [Actions](#)



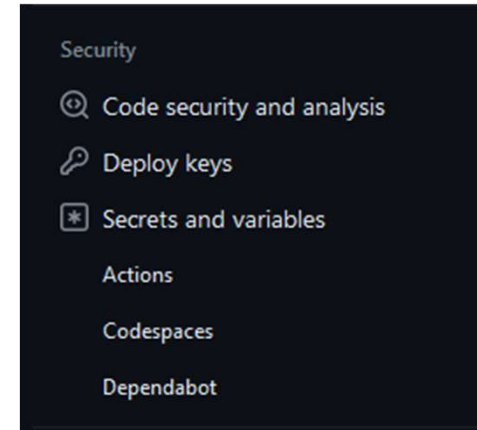
3. Dans la barre latérale cliquez sur le workflow que vous souhaitez afficher.
4. Dans la liste des exécutions de workflow, cliquez sur le nom de l'exécution pour voir le résumé de l'exécution du workflow
5. Dans la barre latérale à gauche ou dans le graphe de visualisation, cliquez sur le travail que vous souhaitez voir.
6. Pour voir les résultats d'une étape, cliquez sur l'étape.

# Création de variables secrètes

GitHub Actions vous permet de créer des variables pouvant contenir des données à protéger.

16/04/2025

- Depuis les [Settings](#) de votre dépôt et la rubrique [Security](#) :



- Allez dans la rubrique [Secrets and Variables > Actions](#)
- En cliquant sur [New repository secret](#), vous pouvez créer une variable secrète.
- Cette variable est utilisable dans le script via la syntaxe `${{ secrets.name }}`

# GitHub Actions artefacts

---

SAUVEGARDE DE DONNÉES DES WORKFLOWS

# Stockage des données de workflow en tant qu'artefacts

Les artefacts vous permettent de conserver des données une fois un travail terminé, et de partager ces données avec un autre travail du même workflow.

- Un artefact est un fichier ou une collection de fichiers générés pendant l'exécution d'un workflow.
- Par exemple, vous pouvez utiliser des artefacts pour enregistrer votre sortie de build et de test une fois l'exécution d'un workflow terminée.
- Toutes les actions et tous les workflows appelés dans une exécution disposent d'un accès en écriture aux artefacts de cette exécution.
  - Par défaut, GitHub Enterprise Server stocke les artefacts et journaux de build pendant 90 jours (paramétrable). La période de conservation d'une demande de tirage (pull request) redémarre chaque fois qu'un utilisateur pousse (par push) un nouveau commit à la demande de tirage.
- Attention : service limité voir payant s'il y a des abus.

# Un souci avec mon build ?

En effet, si on réfléchit à l'environnement de GitHub Actions, j'exécute mon workflow sur une machine dont je n'ai pas accès physiquement.

Qu'en est-il de mon build de maven ?

16/04/2025

Eh bien il est perdu après exécution. Le workflow s'exécute et en fin de workflow, il effectue un nettoyage et supprime le serveur utilisé.



La solution : les artefacts.

Les artefacts vous permettent de partager des données entre travaux dans un workflow et de stocker des données une fois ce workflow terminé.

Donc on va pouvoir stocker nos build dans un artefact.

PRÉSENTATION DE GITHUB ACTIONS

# Documentations GitHub Actions

---

TROUVER L'INFORMATION



# Liens

GitHub Actions peut vous aider à automatiser presque tous les aspects de vos processus de développement d'applications.

Voici quelques ressources utiles pour effectuer vos étapes suivantes avec GitHub Actions

- Pour obtenir un moyen rapide de créer un flux de travail GitHub Actions, consultez « Utilisation de workflows de démarrage ».
  - <https://docs.github.com/fr/actions/learn-github-actions/using-starter-workflows>
- Pour obtenir des workflows d'intégration continue (CI) permettant de générer et tester votre code, consultez « Automatisation des builds et des tests ».
  - <https://docs.github.com/fr/actions/automating-builds-and-tests>
- Pour générer et publier des packages, consultez « Publication de packages ».
  - <https://docs.github.com/fr/actions/publishing-packages>
- Guides
  - <https://docs.github.com/fr/actions/guides>

# Liens

Voici quelques ressources utiles pour effectuer vos étapes suivantes avec GitHub Actions

- Pour le déploiement de projets, consultez « Déploiement ».
  - <https://docs.github.com/fr/actions/deployment>
- Pour automatiser les tâches et les processus sur GitHub, consultez « Gestion des problèmes et demandes de tirage ».
  - <https://docs.github.com/fr/actions/managing-issues-and-pull-requests>
- Pour obtenir des exemples illustrant les fonctionnalités plus complexes de GitHub Actions, dont beaucoup des cas d'usage ci-dessus, consultez « Exemples ».
  - <https://docs.github.com/fr/actions/examples>

CRÉATION DE  
SON PREMIER  
WORKFLOW

# Travaux pratiques

**Attention :** ce qui est présenté dans les différents codes par la suite correspond à un de mes projets et par conséquent propre à son environnement. Chaque projet est différent et a son propre environnement.

# Création d'un 1<sup>er</sup> workflow

Essayez les fonctionnalités de GitHub Actions en 5 minutes ou moins.

Vous n'avez besoin que d'un dépôt GitHub pour créer et exécuter un workflow GitHub Actions. Dans ce guide, vous allez ajouter un workflow qui illustre certaines des fonctionnalités essentielles de GitHub Actions.

L'exemple suivant vous montre comment les travaux GitHub Actions peuvent être déclenchés automatiquement, où ils s'exécutent et comment ils peuvent interagir avec le code dans votre dépôt.

1. Créer un répertoire `.github/workflows` dans le dépôt de votre choix si ce répertoire n'existe pas.
2. Dans le répertoire nouvellement créé, ajouter un fichier `github-action-demo.yml`
3. Ce fichier va contenir le script ci-dessous :

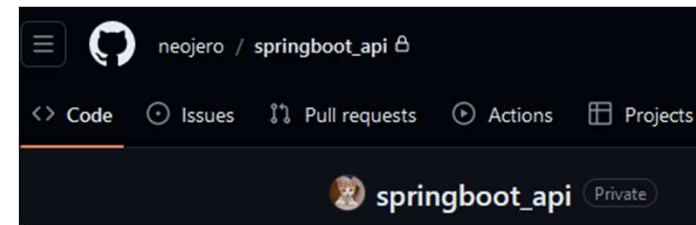
```
name: GitHub Actions Demo
run-name: ${ github.actor }} teste GitHub Actions 🚀
on: [push]
jobs:
  Explore-GitHub-Actions:
    runs-on: ubuntu-latest
    steps:
      - run: echo "🎉 The job was automatically triggered by a ${ github.event_name }} event."
      - run: echo "👤 This job is now running on a ${ runner.os }} server hosted by GitHub!"
      - run: echo "🔖 The name of your branch is ${ github.ref }} and your repository is ${ github.repository }}."
      - name: Check out repository code
        uses: actions/checkout@v4
      - run: echo "💡 The ${ github.repository }} repository has been cloned to the runner."
      - run: echo "📄 The workflow is now ready to test your code on the runner."
      - name: List files in the repository
        run: |
          ls ${ github.workspace }}
      - run: echo "🟢 This job's status is ${ job.status }}."
```

4. Ensuite, effectuez la mise à jour de votre dépôt distant sur GitHub.

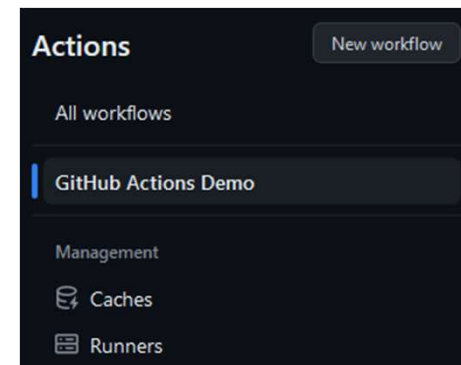
# Affichage des résultats de votre workflow

Désormais, la mise à jour du dépôt est effectuée, allons contrôler sur GitHub, notre workflow.

1. Dans GitHub, accédez à la page principale du dépôt.
2. Sous le nom du dépôt, cliquez [Actions](#)



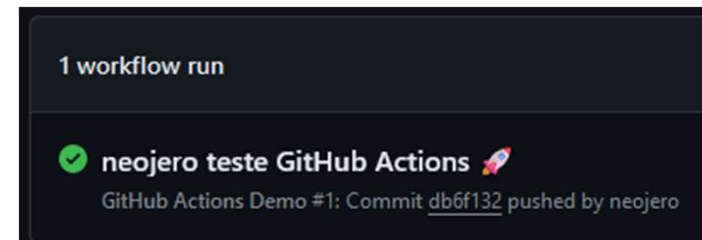
3. Dans la barre latérale gauche, vous avez l'ensemble des workflows. Vous pouvez cliquer sur celui qui vous intéresse.



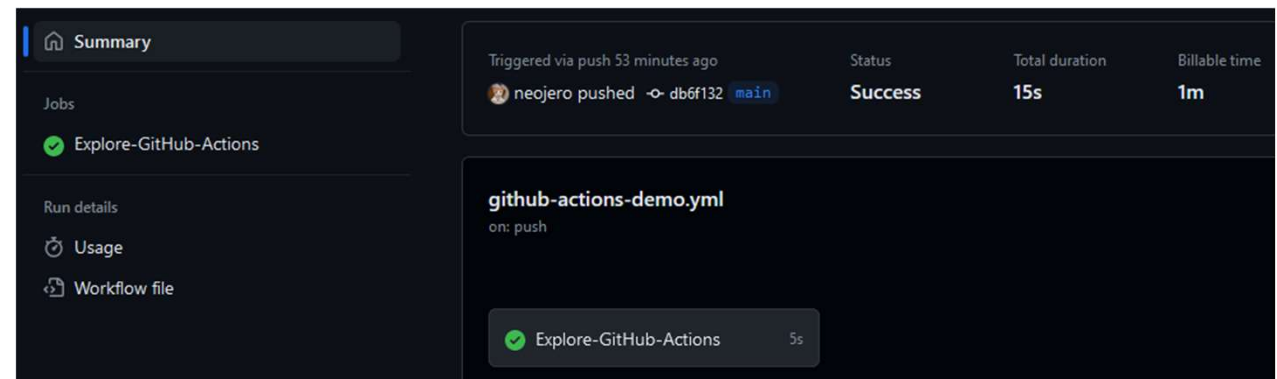
# Affichage des résultats de votre workflow

Entrons dans le détail de notre workflow

4. On peut déjà constater que notre workflow affiche comme nom de workflow **USERNAME teste GitHub Actions** 🚀



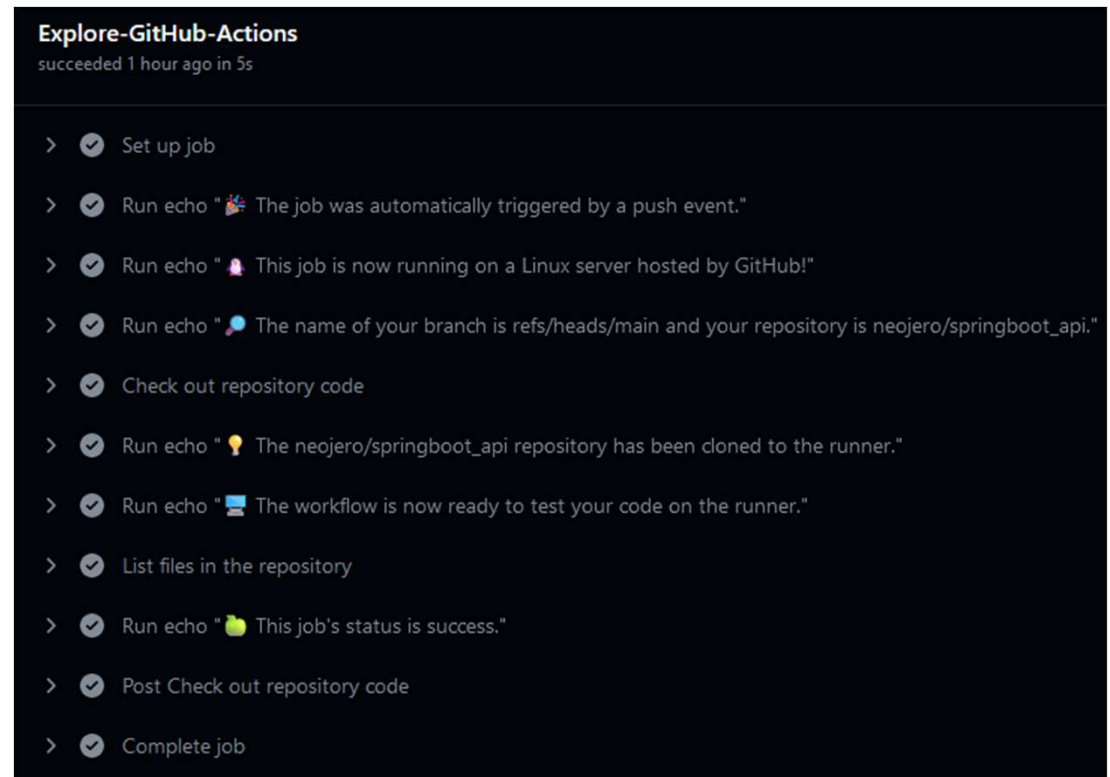
5. En cliquant dessus, on entre dans le détail



# Afficher le journal du workflow

Le journal montre comment chacune des étapes a été traitée

16/04/2025



PRÉSENTATION DE GITHUB ACTIONS

# Afficher le détail d'une étape

On peut ensuite afficher le détail de chaque étape du workflow.

Ici, le détail lors de l'action checkout.

```
✓ Check out repository code

1  ▶ Run actions/checkout@v4
15 Syncing repository: neojero/springboot_api
16 ▶ Getting Git version info
20 Temporarily overriding HOME='/home/runner/work/_temp/3959cd2a-1144-43ea-89b7-5574dd8c25a0'
21 Adding repository directory to the temporary git global config as a safe directory
22 /usr/bin/git config --global --add safe.directory /home/runner/work/springboot_api/springboot_api
23 Deleting the contents of '/home/runner/work/springboot_api/springboot_api'
24 ▶ Initializing the repository
38 ▶ Disabling automatic garbage collection
40 ▶ Setting up auth
46 ▶ Fetching the repository
50 ▶ Determining the checkout info
51 ▶ Checking out the ref
55 /usr/bin/git log -1 --format='%H'
56 'db6f132efaad822517f4eead5ebf49b7c45fb19e'
```



# Suivi des workflows

On peut suivre nos workflows depuis GitHub.

En cas d'échec d'un workflow, un email est envoyé informant le propriétaire du dépôt.

16/04/2025

The image shows a screenshot of the GitHub Actions interface. On the left, a list of workflow runs is displayed under the heading "52 workflow runs". The list includes:

- test2 workflow maven** (status: success) - Java CI with Maven #25: Commit [348b739](#) pushed by neojero
- neojero teste GitHub Actions** (status: success) - GitHub Actions Demo #26: Commit [348b739](#) pushed by neojero
- test2 workflow maven** (status: failure) - Java CI with Maven #24: Commit [348b739](#) pushed by neojero
- neojero teste GitHub Actions** (status: success) - GitHub Actions Demo #25: Commit [348b739](#) pushed by neojero
- test2 workflow maven** (status: failure) - Java CI with Maven #23: Commit [348b739](#) pushed by neojero

On the right, an email notification is shown from "neojero" to "neojero/apiSpringboot2025". The subject is "[neojero/apiSpringboot2025] Run failed at startup: SonarQube, Attempt #2 - main (670e771)". The email body states: "[neojero/apiSpringboot2025] SonarQube workflow run, Attempt #2". Below the text, there is a SonarQube logo and the message "SonarQube, Attempt #2: No jobs were run". A green button labeled "View workflow run" is at the bottom. The email also includes a "Traduire en français" link and a "Corbeille" icon.

PRÉSENTATION DE GITHUB ACTIONS

# Conclusion

L'exemple de workflow que vous venez d'ajouter est déclenché chaque fois que du code est poussé vers la branche, et montre comment GitHub Actions peut fonctionner avec le contenu de votre dépôt.

- GitHub fournit des workflows de démarrage préconfiguré que vous pouvez personnaliser pour créer votre propre workflow d'intégration continue.
- GitHub analyse votre code et affiche des workflows de démarrage CI qui pourraient être utiles pour votre référentiel.
- Vous pouvez parcourir la liste complète des workflows de démarrage dans le référentiel actions/starter-workflows.
  - <https://github.com/actions/starter-workflows>

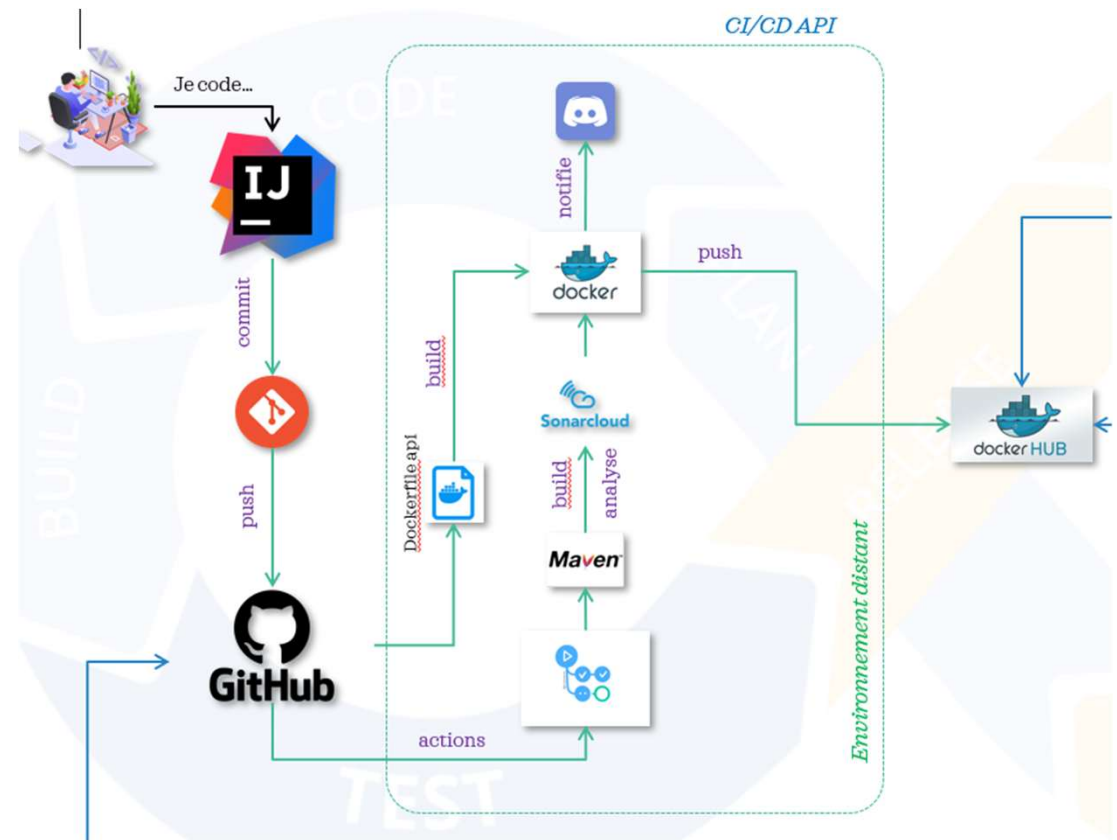
MISE EN PLACE D'UN WORKFLOW POUR LE PROJET  
SPRINGBOOT API AVEC GITHUB ACTIONS

---

# Travaux pratiques *ECF*

# 1ère partie.

Depuis notre environnement local (développement) lors d'un **push** sur la branche **main** vers le dépôt GitHub, c'est à ce moment que notre workflow GitHub Actions pour l'API démarre... !



# Objectifs

Effectuer une recherche depuis la documentation de GitHub Actions pour la mise en place d'un workflow de l'un de nos projets Java.

Ce projet contient un test unitaire, un test d'intégration sur une base de données.

- Ce workflow devra sur un push :
  - Copier notre dépôt dans l'exécuteur (**checkout**)
  - Lancer la phase de tests de notre projet (**CI**).
    - Tests unitaires
    - Tests d'intégrations (par exemple test avec la BDD)
    - Tests fonctionnels [utilisation avec selenium]
    - Lancer l'analyse du code avec SonarCloud.
  - Lancer le build de notre projet.
    - Stocker le build dans un artefact.
  - Lancer la phase de construction de l'image de l'API.  
(**Livraison continue pour CD final**)
    - Récupérer l'artefact stocké
    - Construire l'image docker de l'API depuis un Dockerfile
    - Stocker l'image Docker de l'API dans notre Docker Hub.
    - Informer de la fin du workflow *optionnel*

# Commençons...

Pour créer notre workflow, nous devons dans un exécuteur, effectuer la copie du dépôt, puis effectuer la construction du projet avec Maven.

Pour cela, nous devons installer la version de la JDK de notre projet.

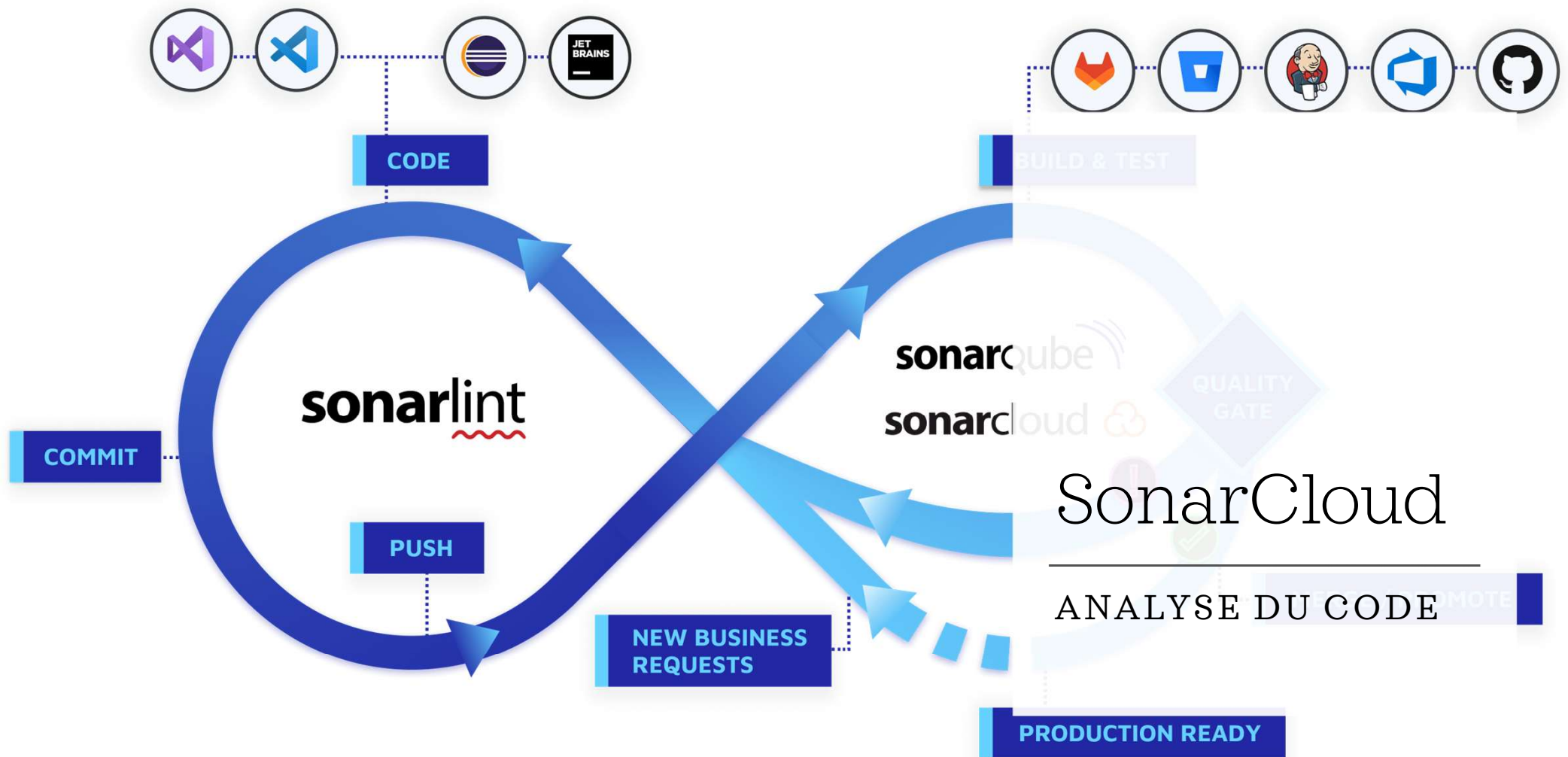
<https://docs.github.com/fr/actions/automating-builds-and-tests>

```
name: myCICD
run-name: ${ github.actor } is testing out GitHub Actions 🚀
on: [push]
jobs:
  build-projet:
    runs-on: ubuntu-latest
    steps:
      # copie de mon dépôt
      - name: Check out repository code
        uses: actions/checkout@v4
      - run: echo "💡 The ${ github.repository } repository has been cloned to the runner."
      - run: echo "🔖 The workflow is now ready to build your code on the runner."
      - name: List files in the repository
        run: |
          ls ${ github.workspace }

      # installation de la JDK
      - name: Set up JDK 21
        uses: actions/setup-java@v4
        with:
          java-version: 21
          distribution: 'zulu' # Alternative distribution options are available.

      # construction du projet
      - name: Build projet
        run: mvn -B verify

      - run: echo "🍏 This job's status is ${ job.status }."
```





# SonarCloud

Un outil comme **SonarCloud** peut vous aider. Il va vous permettre de :

- Analyser votre code.
- Envoyer son analyse au serveur **SonarCloud**.
- Faire un rapport web dynamique pour lister les problèmes, les prioriser et vous expliquer comment améliorer votre code.

- **SonarCloud** se base sur des règles qui ont été prédéfinies.
  - Ces dernières ont été créées pour pouvoir émettre un jugement sur des risques potentiels.
- Il prend en charge tous les principaux langages de programmation.
- **SonarCloud** propose ses propres serveurs pour mener à bien toutes ces tâches. **C'est gratuit pour du code open source, hébergé par exemple sur GitHub.**
  - **Dans le cadre privé d'un projet, un service payant est proposé.**
- Dans la famille **Sonar**, on y trouve
  - **SonarCloud** : version web,
  - **SonarLint** : plugin pour IDE,
  - **SonarQube** : version intégrée à votre projet.

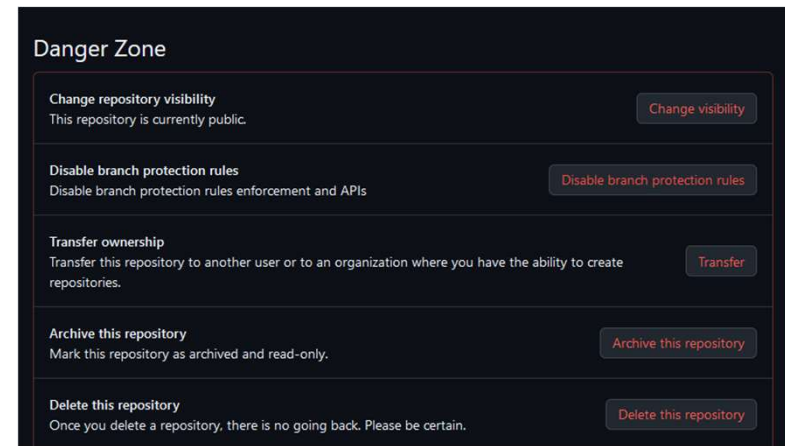


# 1ère étape : création d'un compte SonarCloud

Il faut tout d'abord créer un compte chez **SonarCloud**, en le liant avec son compte GitHub.

Et s'assurer que le projet que l'on souhaite analyser soit public.

- Dès lors, vous pouvez analyser un de vos dépôts GitHub et obtenir l'analyse de votre travail.
  - *Attention, pour rappel, le dépôt doit être **public** pour pouvoir bénéficier de la gratuité.*
  - Dans le cas où votre dépôt est en privée, depuis l'interface GitHub, vous avez la possibilité de changer la visibilité du dépôt.
- Depuis GitHub, pour changer la visibilité du projet, aller dans les **Settings** du dépôt :



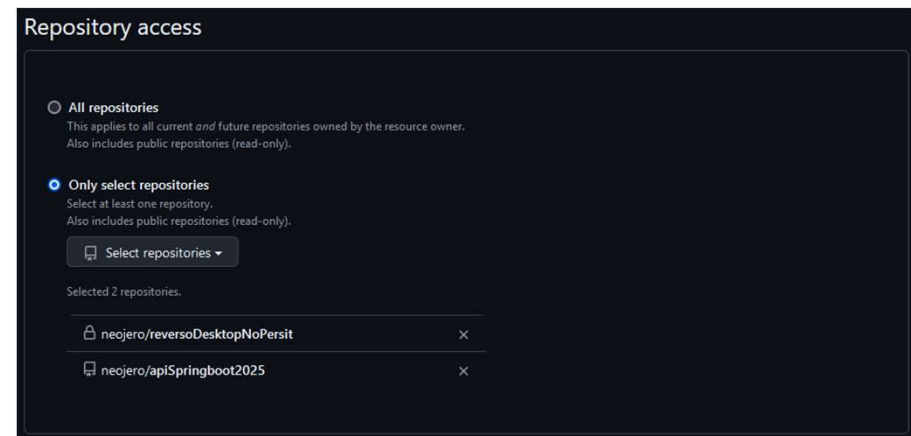
## 2ème étape : sélectionner les dépôts pour SonarCloud

Après avoir lié le compte GitHub avec **SonarCloud**, il faut penser à configurer les dépôts que l'on souhaite faire analyser.

En effet, par défaut, aucun dépôt est sélectionné.

Pour cela, il faut se rendre dans les settings de votre compte Github (en haut à droite, en cliquant sur la photo de votre profil).

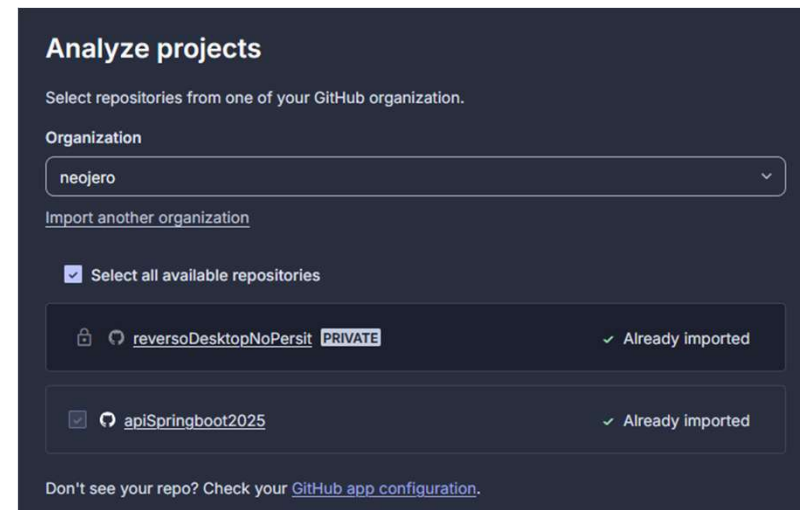
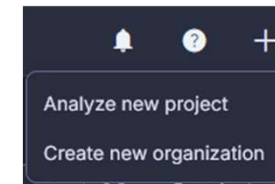
Dans les [settings](#), cliquer sur applications, puis [Configure](#) de [SonarQubeCloud](#). Dès lors, vous allez pouvoir soit autoriser tous les dépôts, ou seulement les dépôts concernés.



## 3ème étape : Ajouter votre dépôt dans SonarCloud

Maintenant qu'au niveau de GitHub, on a autorisé nos dépôts à être visible pour **SonarCloud**, on peut dès lors mettre en place l'analyse du projet.

Avant tout, sur le site de SonarCloud, on va créer un nouveau projet d'analyse et sélectionner le dépôt.



# Intégration avec GitHub Actions

Bien évidemment l'objectif d'un tel outil est de l'intégrer dans notre CI/CD afin d'avoir régulièrement une analyse de notre codage.

- A partir de l'interface SonarCloud, en sélectionnant votre dépôt, vous pouvez configurer la méthode de l'analyse (automatique ou manuelle).
- Ensuite, vous pouvez sélectionner la méthode. Comme nous utilisons GitHub Actions alors...

## Choose your Analysis Method

 With GitHub Actions

- Ensuite, SonarCloud vous indique la marche à suivre :
  - Créer une variable secrète contenant le [token Sonar dans GitHub](#)
  - Ensuite en sélectionnant votre outil de build

## Analyze a project with a GitHub Action

### 1 Create a GitHub Secret

In your GitHub repository, go to [Settings > Secrets > Actions](#) and create a new secret with the following details:

1 In the Name field, enter `SONAR_TOKEN`

2 In the Value field, enter

### 2 Create or update a build file

What option best describes your build?

☐ Maven ☐ Gradle ☐ C, C++ or ObjC ☐ .NET ☐ Other (for JS, TS, Go, Python, PHP, ...)

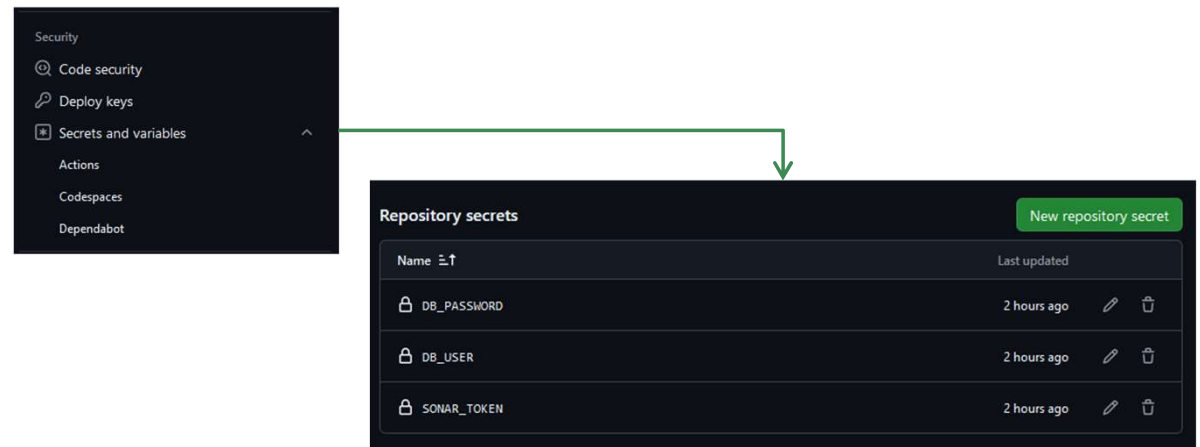
# Ajout du token Sonar dans le Dépôt GitHub

Afin de lier le dépôt GitHub avec le projet d'analyse Sonar, il faut comme Sonar vous l'indique mettre la variable en place.

C'est ce qui permettra aux deux outils de communiquer.

Dans les [Settings](#) du dépôt GitHub, on a un [Secrets and variables](#), qui permet de créer des variables secrètes (mot de passe, login etc...) qu'on pourra reprendre au sein de notre workflow, évitant de mettre en clair des informations confidentielles.

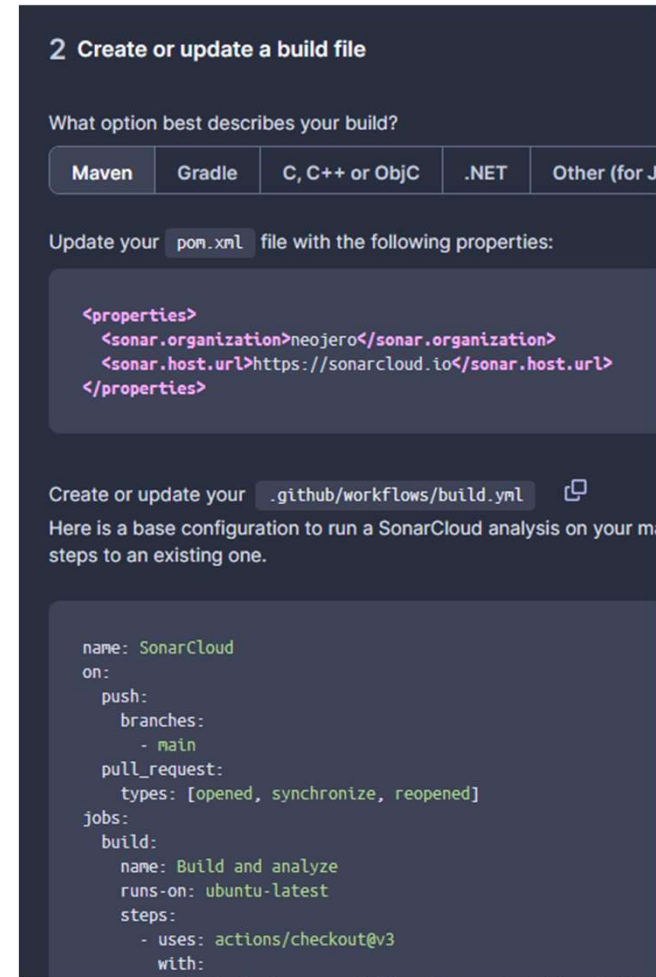
Ces variables secrètes seront alors exclusivement liées au dépôt sur lequel on les place.



# Intégration avec GitHub Actions

Enfin, en sélectionnant Maven, vous obtenez les dernières instructions pour mettre en place votre Actions au sein de GitHub

1. Ajouter dans le pom.xml , la propriété pour Sonar
2. Créer le workflow en le copiant et éventuellement le corriger.



The screenshot shows the '2 Create or update a build file' step of the SonarCloud integration wizard. It asks 'What option best describes your build?' with buttons for Maven, Gradle, C, C++ or ObjC, .NET, and Other (for J). The 'Maven' button is selected. Below, it says 'Update your pom.xml file with the following properties:' and shows a code block with XML properties for sonar.organization, sonar.host.url, and sonar.organization. At the bottom, it says 'Create or update your .github/workflows/build.yml' and provides a base configuration for a SonarCloud analysis workflow.

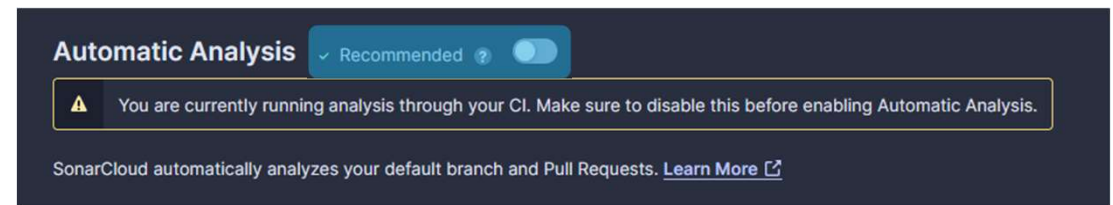
```
name: SonarCloud
on:
  push:
    branches:
      - main
  pull_request:
    types: [opened, synchronize, reopened]
jobs:
  build:
    name: Build and analyze
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - with:
```

# Intégration avec GitHub Actions

**Tips** : après avoir intégré votre projet dans sonarCloud, il va effectuer une première analyse du projet.

Souvent à la mise en place de GitHub Actions, il peut y avoir conflit.

- Pour régler ce problème, désactiver Automatic Analysis



# GITHUB\_TOKEN

Au début de chaque workflow, **GitHub** crée automatiquement un `GITHUB_TOKEN` unique à utiliser dans votre workflow.

Vous pouvez utiliser `GITHUB_TOKEN` pour vous authentifier dans un travail de workflow.

Lorsque vous activez GitHub Actions, GitHub installe une variable d'environnement nommé `GITHUB_TOKEN` sur votre dépôt.

Le `GITHUB_TOKEN` est un jeton d'accès.

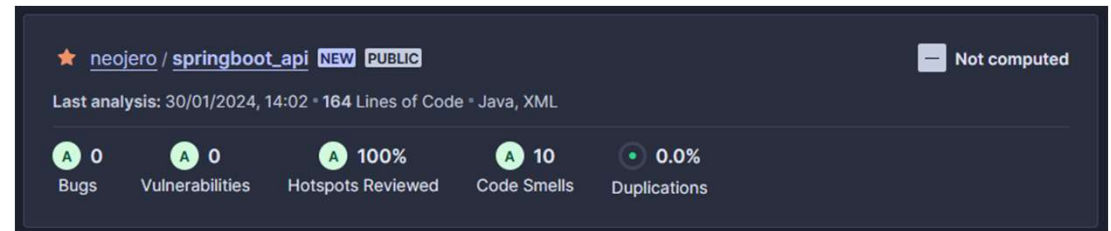
Vous pouvez utiliser le jeton d'accès pour vous authentifier au nom de GitHub sur votre dépôt :

- Les autorisations du jeton sont limitées au dépôt qui contient votre workflow.
- Avant que chaque travail ne commence, GitHub récupère un jeton d'accès d'installation pour le workflow.
- Le `GITHUB_TOKEN` expire à la fin d'un travail ou après un délai maximal de 24 heures.



# Résultats

Une foule d'informations sont disponibles.



- des informations sur la branche construite, les nombre de lignes de code, le langage
- le nombre d'avertissements relevés par thèmes :
  - [Reliability - Bugs](#) → bugs potentiels de votre application,
  - [Security - Vulnerabilities](#) → problèmes potentiels de sécurité,
  - [Maintenability - Code smells](#) → problèmes de syntaxe pouvant rendre votre code moins lisible et moins maintenable (littéralement "mauvaises odeurs de code")
- la couverture du code par les tests et le nombre de tests unitaires
- la duplication de code : proportion de code qui semble être du copier-coller. Plus le pourcentage est élevé, moins bonne est la qualité, car hormis les tests, il faut toujours regrouper du code commun et ne pas faire du copier-coller.

# GitHub Actions artefacts

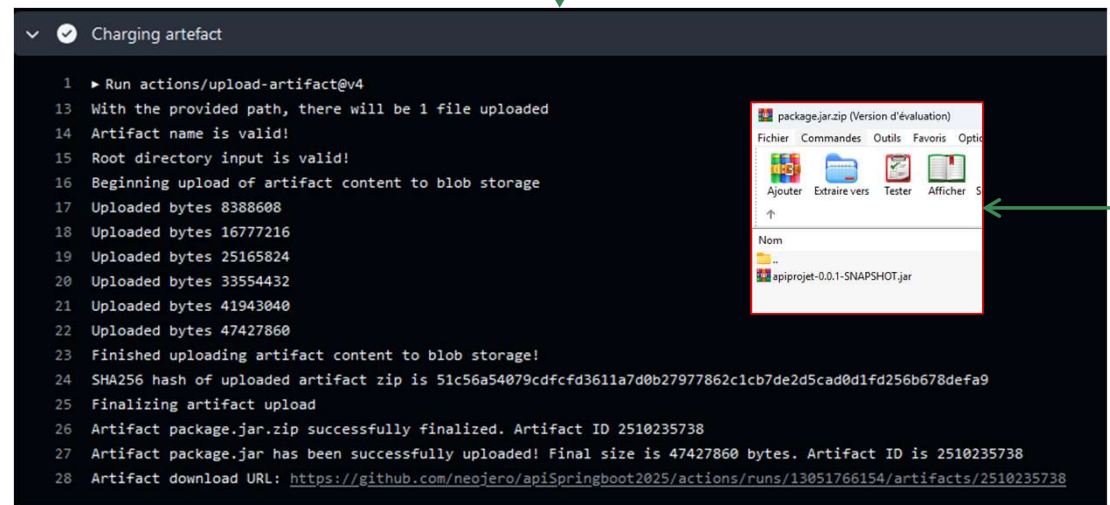
---

SAUVEGARDE DE DONNÉES DES WORKFLOWS

# Les artefacts

Les **artefacts** sont une solution de stockage temporaire proposée par GitHub des données que nous souhaiterions conserver soit au sein du même workflow, soit pour un autre workflow.

```
- name: Charging artefact # permet de charger un artefact
  uses: actions/upload-artifact@v4
  with:
    name: package.jar
    path: target/*.jar
    retention-days: 1
```





# Discord

---

AJOUT D'UNE  
NOTIFICATION DANS LE  
WORKFLOW

# Notification Discord

Dans le cadre d'un travail collaboratif, il est toujours bien d'informer l'équipe des actions réalisées.

Ainsi en cas d'échec ou de réussites, les actions suivantes peuvent être appliquées.

Voici, un exemple de mise en place d'une notification Discord au travers d'un webhook.

Le token est fourni dans le lien webhook que Discord crée :

[https://discord.com/api/webhooks/\\*\\*/QCG5pmXJv8shssdO4UGfSwn99OJyIfUixcmv5T4dXXbztAtdGfBiEkSIxbU5J-odH79q](https://discord.com/api/webhooks/**/QCG5pmXJv8shssdO4UGfSwn99OJyIfUixcmv5T4dXXbztAtdGfBiEkSIxbU5J-odH79q)

```
# Notification
Discord:
  name: Discord
  runs-on: ubuntu-latest
  needs: docker-build-image-push

  steps:
    - name: Discord notification
      env:
        DISCORD_WEBHOOK: ${ secrets.DISCORD_WEBHOOK_TOKEN }
      uses: sarisia/actions-status-discord@v1
      if: always()
      with:
        description: 'The project Reverso has been deployed.'
```



Github APP 07/05/2024 11:46

## Success: Java CI/CD with SonarCloud and Artifact

The project Reverso has been deployed.

<b>Repository</b>	<b>Ref</b>
neojero/reversoDesktopNoPersit	refs/heads/main

<b>Event - push</b>
8777827 modification Main

<b>Triggered by</b>	<b>Workflow</b>
neojero	Java CI/CD with SonarCloud and Artifact
07/05/2024 11:46	

# Fin du CI...

---

...DÉCOUVERTE DE  
DOCKER...AVANT DE  
POURSUIVRE SUR LE  
DÉPLOIEMENT.

4/16/2025



PRÉSENTATION DE GITHUB ACTIONS

46