

Architecture logicielle (A2025)

Rapport de laboratoire

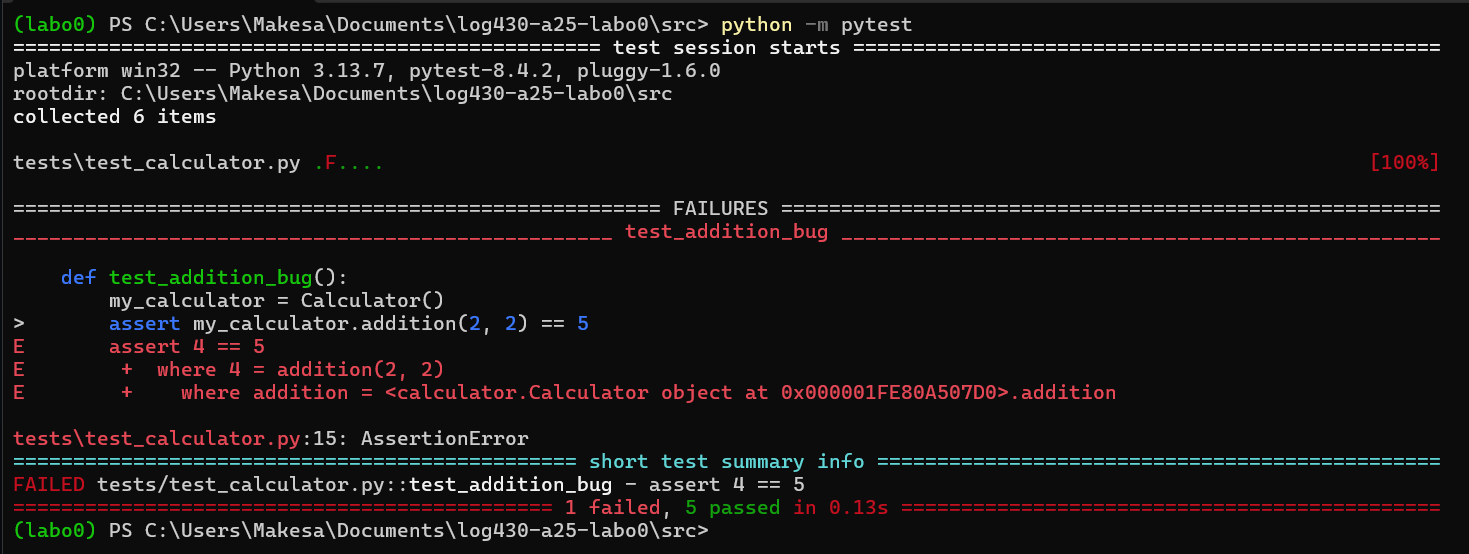
|  |  |
| --- | --- |
| **No de laboratoire** | Laboratoire 00 |
| **Étudiant(s)** | Mvuemba Gildor Makesa |
| **Cours** | LOG430-01-02 |
| **Session** | A2025 |
| **Professeur** | **Fabio Petrillo** |
| **Chargés de laboratoire** | **Gabriel C. Ullmann** |
| **Date de remise** | 13 sept.-25 |

💡 **Question 1** : Si l’un des tests échoue à cause d’un bug, comment pytest signale-t-il l’erreur et aide-t-il à la localiser ? Rédigez un test qui provoque volontairement une erreur, puis montrez la sortie du terminal obtenue.

Quand un test échoue, pytest fournit une sortie détaillée pour identifier l'erreur.

Il montre :

* **Le nom du fichier de test** et **le nom de la fonction de test** qui a échoué.
* **La ligne de code** exacte où l'échec s'est produit.
* **Une explication** de l'échec (par exemple, un AssertionError si l'assertion assert est fausse).

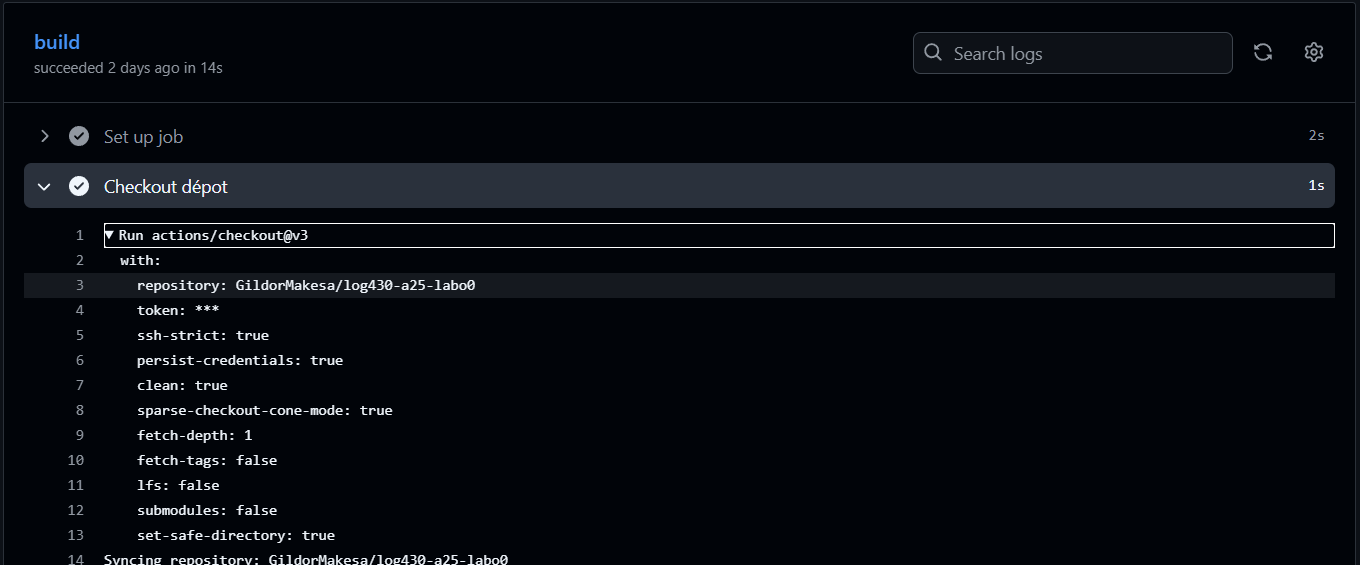


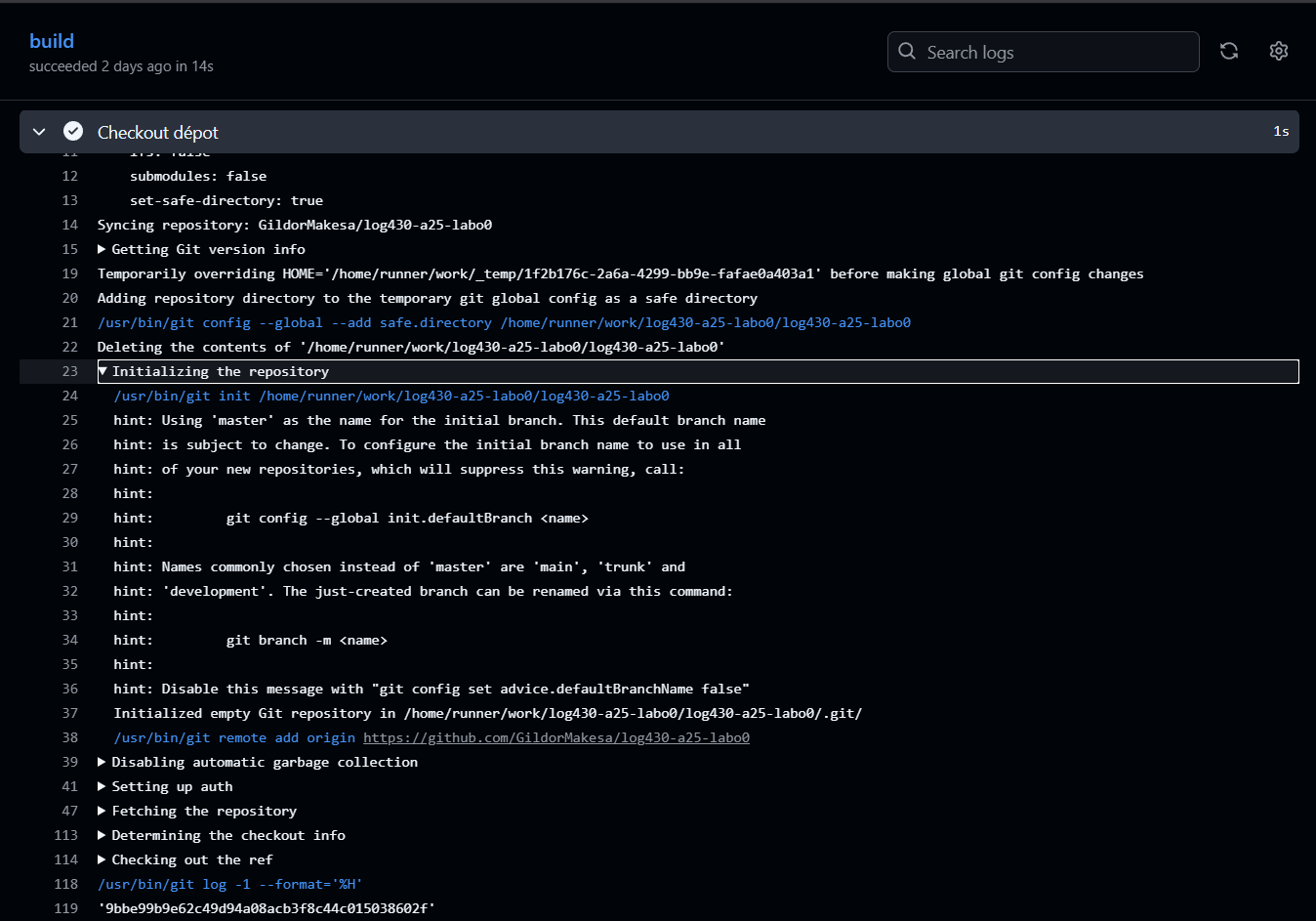
💡 **Question 2** : Que fait GitLab pendant les étapes de « setup » et « checkout » ? Veuillez inclure la sortie du terminal Gitlab CI dans votre réponse.

Explication des étapes de la pipeline

Les étapes checkout et setup dans votre fichier ci.yml (GitHub Actions) ont les rôles suivants :

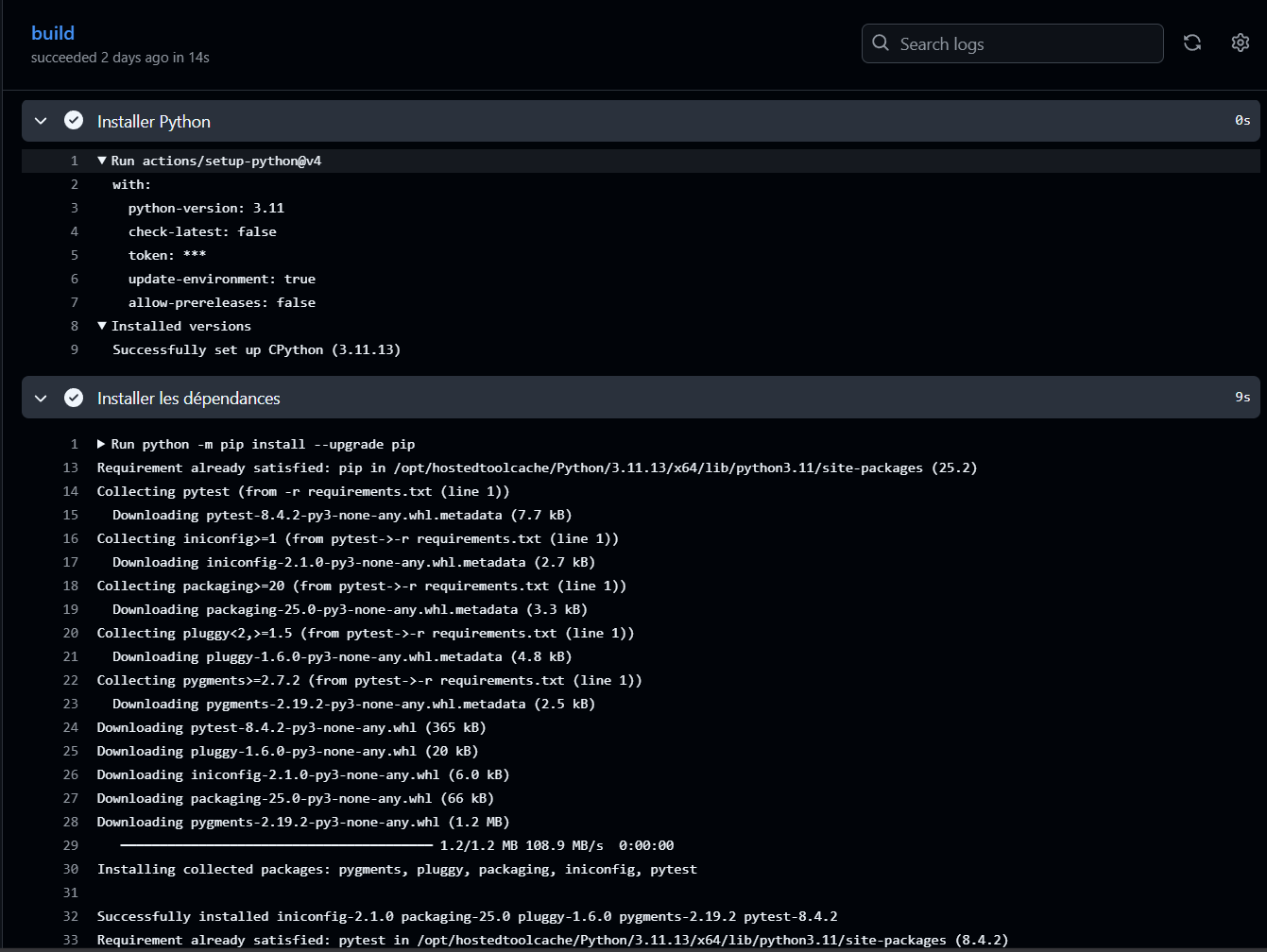
Checkout : Cette étape (uses: actions/checkout@v3) permet au runner de la pipeline (le conteneur sur les serveurs de GitHub) de récupérer votre code. Il télécharge l'intégralité du dépôt sur lequel l'événement a eu lieu (le push dans votre cas).





Setup : Les étapes qui suivent, comme Installer Python (uses: actions/setup-python@v4), permettent de préparer l'environnement. Elles installent les outils nécessaires (Python, les dépendances avec pip install) pour que le runner puisse exécuter votre code et vos tests.

C'est un peu comme si un robot-testeur recevait une copie de votre projet, l'ouvrait, installait tous les logiciels nécessaires, puis lançait les tests.



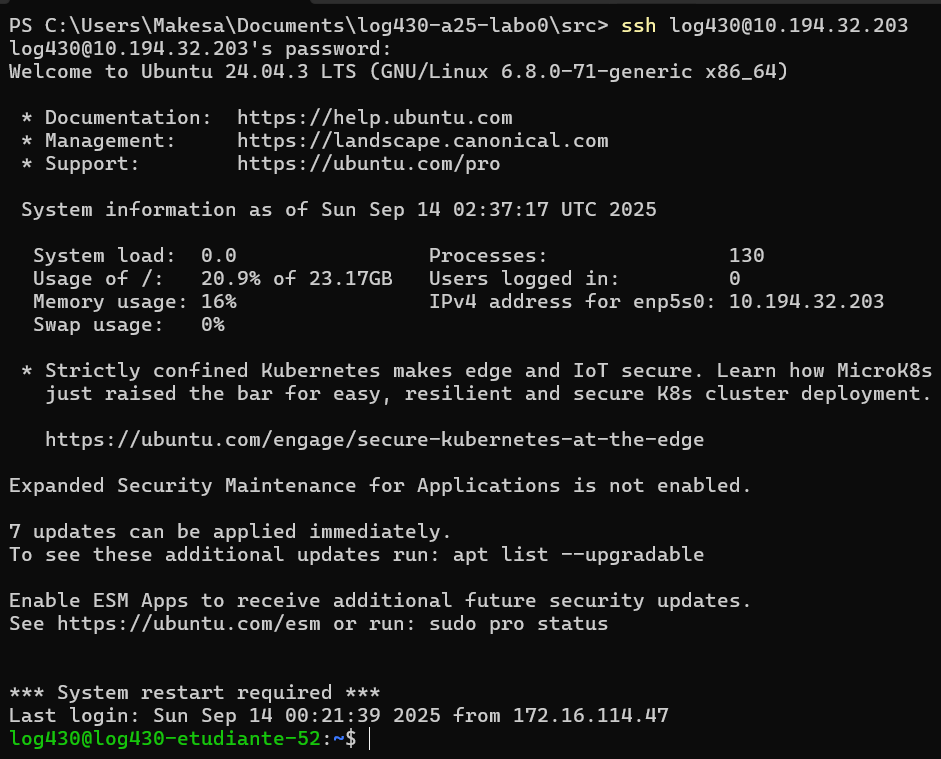
💡 **Question 3** : Quel approche et quelles commandes avez-vous exécutées pour automatiser le déploiement continu de l'application dans la machine virtuelle ? Veuillez inclure les sorties du terminal et les scripts bash dans votre réponse.

Pour automatiser le déploiement continu de mon application dans la machine virtuelle (VM), j’ai choisi l’approche **GitHub Actions avec un self-hosted runner**.  
En effet, comme ma VM n’est accessible que via **VPN** et ne peut pas être déployée sur un runner GitHub hébergé dans le cloud, la solution la plus adaptée était d’installer et de configurer un runner directement dans la VM.

**Étapes suivies**

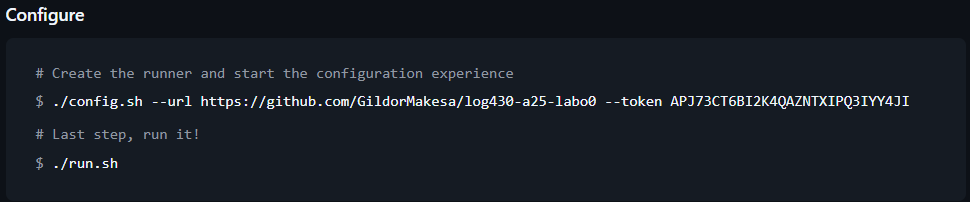
**1. Installation du self-hosted runner dans la VM**

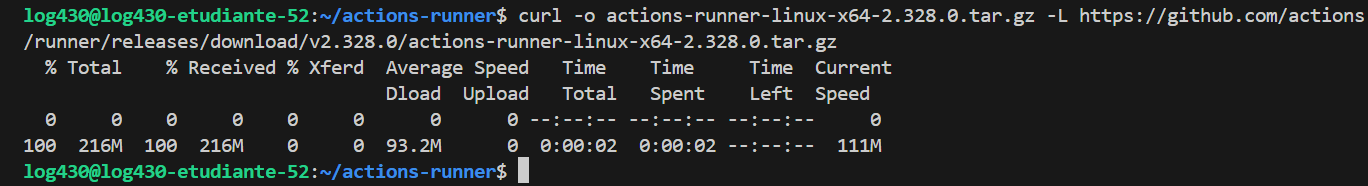
* Connexion à la VM (sous VPN) :



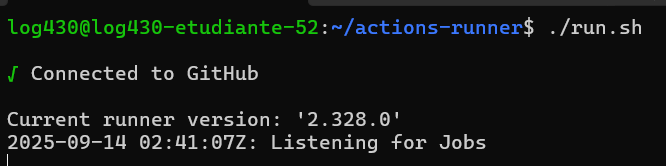
* Téléchargement et configuration du runner (suivant les instructions de GitHub) :







* Démarrage du service :



À ce stade, ma VM est enregistrée comme **runner self-hosted** dans GitHub.

2. Création du workflow CI/CD

Dans le dépôt GitHub (.github/workflows/ci.yml), j’ai défini deux jobs :

* build → exécute les tests unitaires
* deploy → déploie l’application dans Docker sur la VM

💡 **Question 4** : Quel type d'informations pouvez-vous obtenir via la commande « top » ? Veuillez inclure la sortie du terminal dans votre réponse.