

Exemple de soumission d'activité

ÉTS - LOG430 - Architecture logicielle - Hiver 2026

Étudiant(e) : Adel Belhadj

Questions

(Il est obligatoire d'ajouter du code, des captures d'écran ou des sorties de terminal pour illustrer chacune de vos réponses.)

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 le signale immédiatement en affichant un statut FAILED et en indiquant précisément où se trouve le problème. Concrètement, il affiche le nom du fichier et la ligne du test qui a échoué, puis le bloc FAILURES qui contient l'expression assert fautive, et les valeurs évaluées. Si l'échec provient plutôt d'une exception, pytest affiche l'exception avec une traceback qui remonte la pile de calls jusqu'à la ligne exacte qui en est responsable. Grâce à ces éléments, on peut localiser rapidement un bug, on sait quel test échoue et à quel endroit, et quelle valeur réelle a été obtenue par rapport à celle attendue.

Pour illustrer cela, j'ai créé un test qui est volontairement faux dans test_calculator.py, puis j'ai exécuté pytest. Voici le code du test:

```
def test_addition_fails():  
    my_calculator = Calculator()  
    assert my_calculator.addition(2, 2) == 5
```

Voici une capture d'écran du output du terminal une fois qu'on a lancé la commande pytest:

```

===== test session starts =====
platform win32 -- Python 3.12.3, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\adel\Desktop\all\Ecole\Juice (nouvelle edition)\LOG430\Lab\lab0\log430-lab0
configfile: pyproject.toml
plugins: anyio-4.11.0, Faker-36.1.1
collected 3 items

src\tests\test_calculator.py ..F [100%]

===== FAILURES =====
_____ test_addition_fails _____

    def test_addition_fails():
        my_calculator = Calculator()
>       assert my_calculator.addition(2, 2) == 5
E       assert 4 == 5
E       + where 4 = addition(2, 2)
E       +     where addition = <calculator.Calculator object at 0x0000025125BAE060>.addition

src\tests\test_calculator.py:20: AssertionError
===== short test summary info =====
FAILED src/tests/test_calculator.py::test_addition_fails - assert 4 == 5
===== 1 failed, 2 passed in 0.18s =====

```

2. Que fait GitHub pendant les étapes de « setup » et « checkout » ? Veuillez inclure la sortie du terminal GitHub CI dans votre réponse.

Pendant les étapes setup et checkout d'un workflow GitHub Actions, GitHub prépare d'abord le job via une étape Set up job. Cette étape sert à initialiser l'environnement d'exécution sur le runner afin que les étapes suivantes s'exécutent dans un environnement qui est cohérent. Ensuite, l'étape checkout récupère le dépôt dans l'espace de travail du runner pour que le workflow puisse accéder au code source et exécuter des commandes comme l'installation des dépendances et l'exécution des tests. Par défaut, actions/checkout récupère uniquement le commit associé au event qui a déclenché le cworkflow, sauf si on change la configuration.

```

v ✓ Set up job 0s

1 Current runner version: '2.331.0'
2 ▶ Runner Image Provisioner
8 ▶ Operating System
12 ▶ Runner Image
17 ▶ GITHUB_TOKEN Permissions
21 Secret source: Actions
22 Prepare workflow directory
23 Prepare all required actions
24 Getting action download info
25 Download action repository 'actions/checkout@v3' (SHA:f43a0e5ff2bd294095638e18286ca9a3d1956744)
26 Download action repository 'actions/setup-python@v4' (SHA:7f4fc3e22c37d6ff65e88745f38bd3157c663f7c)
27 Complete job name: build

```

```
Checkout dépôt 1s
1 ▶ Run actions/checkout@v3
14 Syncing repository: thealgeriandev/developer/log430-lab0
15 ▶ Getting Git version info
19 Temporarily overriding HOME='/home/runner/work/_temp/c15a02fb-10c8-41d7-8d1c-75200ed3fc26' before making global git
    config changes
20 Adding repository directory to the temporary git global config as a safe directory
21 /usr/bin/git config --global --add safe.directory /home/runner/work/log430-lab0/log430-lab0
22 Deleting the contents of '/home/runner/work/log430-lab0/log430-lab0'
23 ▶ Initializing the repository
40 ▶ Disabling automatic garbage collection
42 ▶ Setting up auth
48 ▶ Fetching the repository
119 ▶ Determining the checkout info
120 ▶ Checking out the ref
124 /usr/bin/git log -1 --format='%H'
125 '7c4f134f1fe997d086ab1dadb56e3e69a85b9103'
```

3. Quel type d'informations pouvez-vous obtenir via la commande top ? Veuillez donner quelques exemples. Veuillez inclure la sortie du terminal dans votre réponse.

La commande top sert à surveiller en temps réel l'état de la VM et la consommation de ses ressources. Elle donne d'abord un résumé global du système puis une liste des processus triés. On peut y retrouver le load average, le uptime, le nombre total de processus et leur état, l'utilisation du CPU en pourcentage, l'utilisation de la mémoire vive, l'utilisation de la mémoire swap et des informations reliées à chaque processus comme le PID.

```

root@log430-lab0-vm:~# top
top - 06:12:26 up 1:01, 1 user, load average: 0.00, 0.10, 0.07
Tasks: 79 total, 1 running, 78 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.3 st
MiB Mem : 3649.5 total, 2440.3 free, 146.0 used, 1063.2 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 3436.6 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
    1 root        20   0 167360  12820 8400 S   0.0   0.3   0:04.74 systemd
    2 root        20   0     0     0     0 S   0.0   0.0   0:00.00 kthreadd
    3 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 rcu_gp
    4 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 rcu_par_gp
    5 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 slub_flushwq
    6 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 netns
    8 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 kworker/0:0H-events_+
   10 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 mm_percpu_wq
   11 root        20   0     0     0     0 S   0.0   0.0   0:00.00 rcu_tasks_trace
   12 root        20   0     0     0     0 S   0.0   0.0   0:00.37 ksoftirqd/0
   13 root        20   0     0     0     0 I   0.0   0.0   0:00.59 rcu_sched
   14 root        rt    0     0     0     0 S   0.0   0.0   0:00.00 migration/0
   15 root        20   0     0     0     0 S   0.0   0.0   0:00.00 cpuhp/0
   16 root        20   0     0     0     0 S   0.0   0.0   0:00.00 kdevtmpfs
   17 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 inet_frag_wq
   18 root        20   0     0     0     0 S   0.0   0.0   0:00.00 kauditd
   19 root        20   0     0     0     0 S   0.0   0.0   0:00.00 oom_reaper
   20 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 writeback
   42 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 kblockd
   43 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 blkcg_punt_bio
   44 root        20   0     0     0     0 I   0.0   0.0   0:00.01 kworker/0:1-cgroup_d+
   45 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 tpm_dev_wq
   46 root         0 -20     0     0     0 I   0.0   0.0   0:00.00 ata_sff

```

Déploiement

Dans ce laboratoire, le déploiement continu a été automatisé à l'aide d'un GitHub Runner self-hosted installé directement sur la VM. L'idée est que le job GitHub Actions ne s'exécute pas sur un runner cloud, mais sur la VM elle-même, ce qui permet de déployer sans ouvrir de ports supplémentaires et sans dépendre d'un mécanisme de webhook externe. Après l'installation des dépendances du runner, un utilisateur a été créé pour respecter la contrainte du runner (ne pas être exécuté en sudo), puis le runner a été enregistré sur le dépôt GitHub avec le script config.sh en fournissant l'URL du repo et un token d'inscription.

```

githubrunner@log430-lab0-vm:~$ ls
githubrunner@log430-lab0-vm:~$ ./config.sh --url https://github.com/thealgeriandevloper/log430-lab0 --token A5JI4MAPMU
FYS7UVL62YXX3JOHHUS
-bash: ./config.sh: No such file or directory
githubrunner@log430-lab0-vm:~$ mkdir actions-runner && cd actions-runner
githubrunner@log430-lab0-vm:~/actions-runner$ curl -o actions-runner-linux-x64-2.331.0.tar.gz -L https://github.com/act
ions/runner/releases/download/v2.331.0/actions-runner-linux-x64-2.331.0.tar.gz
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left     Speed
  0     0    0     0     0     0      0      0      0      0      0      0      0      0      0  0 --:--:-- --:--:-- --:--:--
  0
  3 212M    3 8295k    0     0  9771k      0  52 212M    52 110M    0     0  59.6M    0100 212M 100 212M    0
  0 76.9M    0 0:00:02 0:00:02 --:--:-- 106M
githubrunner@log430-lab0-vm:~/actions-runner$
echo "5fcc01bd546ba5c3f1291c2803658ebd3cedb3836489eda3be357d41bfcf28a7  actions-runner-linux-x64-2.331.0.tar.gz" | sha
sum -a 256 -c
actions-runner-linux-x64-2.331.0.tar.gz: OK
githubrunner@log430-lab0-vm:~/actions-runner$
tar xzf ./actions-runner-linux-x64-2.331.0.tar.gz
githubrunner@log430-lab0-vm:~/actions-runner$
./config.sh --url https://github.com/thealgeriandevloper/log430-lab0 --token A5JI4MAPMU
FYS7UVL62YXX3JOHHUS

```

```
-----
# Authentication

✓ Connected to GitHub

# Runner Registration

Enter the name of the runner group to add this runner to: [press Enter for Default]

Enter the name of runner: [press Enter for log430-lab0-vm]

This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]

✓ Runner successfully added

# Runner settings

Enter name of work folder: [press Enter for _work]

# Runner settings

Enter name of work folder: [press Enter for _work]

work]

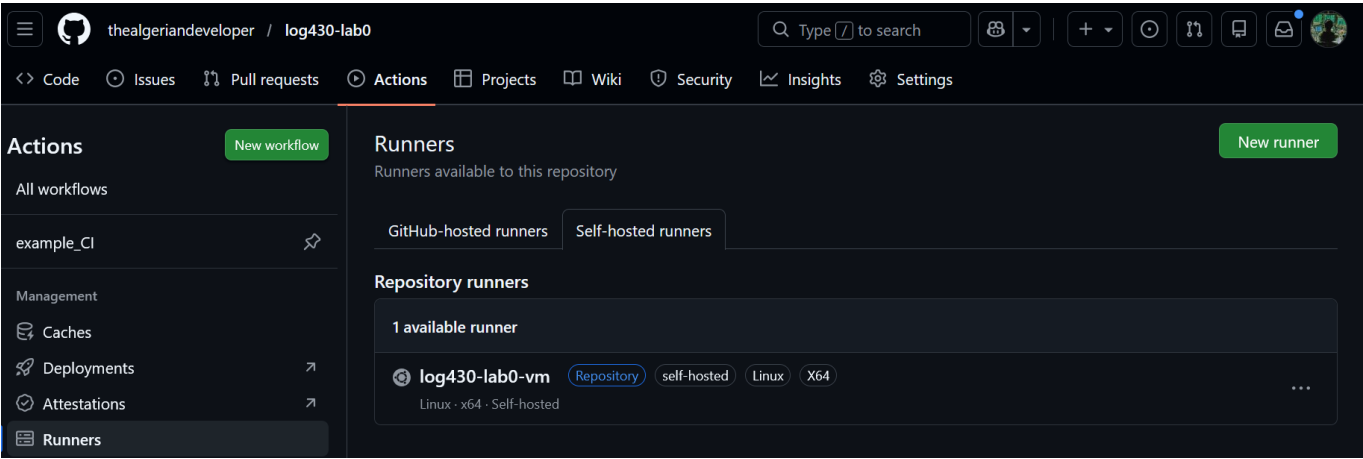
✓ Settings Saved.
```

```
githubrunner@log430-lab0-vm:~/actions-runner$ ./run.sh

✓ Connected to GitHub

2026-01-22 06:28:26Z2026-01-22 06:28:26Z: Listening for Jobs
```

Une fois configuré, le runner a été démarré et confirmé être Online dans l'interface GitHub:



Après cela, un workflow CD a été ajouté au repo dans le fichier .github/workflows/cd.yml et configuré pour se déclencher automatiquement à chaque push un peu comme le CI. Ce workflow utilise runs-on: self-hosted, ce qui force GitHub Actions à exécuter le job sur la VM. Le job commence par récupérer le code du repo avec l'étape checkout, puis exécute des commandes de validation.

