

# Laboratoire 0 - Infrastructure (Git, Docker, CI/CD)

---

## Table des matières

1. Introduction, description des fichiers et définition.
2. Réponse aux question.

## 1. Introduction, description des fichiers et définitions.

### Context et introduction

Ce laboratoire est le premier d'une suite de laboratoire donné dans le cadre du cour LOG430 - architecture logiciel. Il a pour objectif d'apprendre à l'étudiant les notions de base en terme d'infrastructure de développement logiciel, plus précisément concernant l'environnement GIT, la virtualisation avec Docker et le CI/CD typiquement utilisé en DevOps.

### Objectifs du laboratoire

Tel que décrit dans l'énoncé du laboratoire.

- Apprendre à créer un projet **Python** conteneurisé avec **Docker** à partir de zéro.
- Apprendre à écrire et exécuter des tests automatisés avec **pytest**.
- Mettre en place un pipeline **CI/CD** avec les ressources à notre disposition.

### Definitions

- LXD: Lightweight open source virtualisation. Permet de créer et gérer des composantes virtuelles sur une machine.
- LXC: Conteneur ou machine virtuelle permettant une virtualisation.
- Docker: Permet de déployer une application en faisant abstraction du système opératif d'un ordinateur. Docker crée un conteneur dans lequel toutes les dépendances d'une application sont insérées en passant du hardware jusqu'aux modules de dépendances.
- Choco : Gestionnaire de paquets pour windows.

### Description des fichiers

#### Fourni par le laboratoire :

- Dockerfile  
Sert à définir les attributs du conteneur Docker. Un conteneur contient les éléments et composantes essentiels au fonctionnement d'une application. Une application contenue dans un conteneur peut être déployée sur tout ordinateur sans considération du système opératif de celui-ci.
- docker-compose.yml # Indique à docker comment construire le conteneur Docker.

#### À produire :

- requirement.txt  
Définie la liste des dépendances python du projet.

- .env

Définie les variables dans une instance de l'application.

## 2. Réponses aux questions.

### 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.**

```
C:\Users\benja\Documents\ETS\H26\LOG430\GitHubTest\labo-00-infrastructure-git-docker-ci-cd-BeTardif>python -m pytest
=====
test session starts =====
platform win32 -- Python 3.13.9, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\benja\Documents\ETS\H26\LOG430\GitHubTest\labo-00-infrastructure-git-docker-ci-cd-BeTardif
configfile: pyproject.toml
plugins: anyio-4.10.0, cov-6.2.1
collected 6 items

src\tests\test_calculator.py .....
```

[100%]

```
=====
6 passed in 0.07s =====
```

  

```
C:\Users\benja\Documents\ETS\H26\LOG430\GitHubTest\labo-00-infrastructure-git-docker-ci-cd-BeTardif>python -m pytest
=====
test session starts =====
platform win32 -- Python 3.13.9, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\benja\Documents\ETS\H26\LOG430\GitHubTest\labo-00-infrastructure-git-docker-ci-cd-BeTardif
configfile: pyproject.toml
plugins: anyio-4.10.0, cov-6.2.1
collected 6 items

src\tests\test_calculator.py .....,F
```

[100%]

```
=====
FAILURES =====
test_failing
```

---

```
def test_failing():
    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 0x00000284A6CF1250>.addition
```

  

```
src\tests\test_calculator.py:37: AssertionError
=====
short test summary info =====
FAILED src\tests\test_calculator.py::test_failing - assert 4 == 5
===== 1 failed, 5 passed in 0.20s =====
```

**Réponse :** La capture d'écran présentée tient à démontrer la comparaison entre une suite de test qui échoue et une qui réussie. Elle tient aussi à expliquer comment pytest signale les erreurs des tests qui n'ont pas passés. Comme c'est représenté, Pytest démontre exactement à quelle ligne l'erreur se trouve en plus de montrer un code snippet qui indique le test qui a échoué.

### Question 2

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

**Réponse :** Dans le fichier `./.github/workflows/ci.yml` nous avons listé des actions à produire le *continuous integration (CI)* de notre programme. C'est dans ce fichier que nous intégrons une suite de tests à produire automatiquement lors des commandes indiquées. Dans notre cas, ce sont les commandes *Push* et *Pull\_Request*. La question demande ce que fait GitHub lors de deux étapes du processus que nous avons mis en place, le *setup* et le *checkout*. La première, le *setup*, est l'étape initiale par laquelle GitHub prépare le travail à faire. C'est lors de cette étape que les authentifications sont produites. Les ressources sont téléchargées et l'environnement est préparé pour le reste des actions mentionnées.

Set up job 2s

```

1 Current runner version: '2.331.0'
2 ▶ Runner Image Provisioner
3 ▶ Operating System
4 ▶ Runner Image
5 ▶ GITHUB_TOKEN Permissions
6 Secret source: Actions
7 Prepare workflow directory
8 Prepare all required actions
9 Getting action download info
10 Download action repository 'actions/checkout@v4' (SHA:34e114876b0b11c390a56381ad16ebd13914f8d5)
11 Download action repository 'classroom-resources/autograding-python-grader@v1'
  (SHA:e7b43bb6c646f533fde044cce6d8cc5d1e0e9568)
12 Download action repository 'classroom-resources/autograding-io-grader@v1' (SHA:059171be2cf09310e36a4be355b8ee0d2728fca0)
13 Download action repository 'classroom-resources/autograding-grading-reporter@v1'
  (SHA:d13b5912daf5042ac1b71bc38bdb4a6bcd61d8ca)
14 Complete job name: run-autograding-tests

```

> Build classroom-resources/autograding-python-grader@v1 13s

Ensuite, GitHub construit l'environnement pour arriver à la deuxième étape demandée dans la question, le checkout :

Checkout code 1s

```

1 ▶ Run actions/checkout@v4
2 Syncing repository: guteacher/lab0-00-infrastructure-git-docker-ci-cd-BeTardif
3 ▶ Getting Git version info
4 Temporarily overriding HOME='/home/runner/work/_temp/bbe84f08-c957-4d70-a0c2-f429de843daa' before making global git
  config changes
5 Adding repository directory to the temporary git global config as a safe directory
6 /usr/bin/git config --global --add safe.directory /home/runner/work/lab0-00-infrastructure-git-docker-ci-cd-
  BeTardif/lab0-00-infrastructure-git-docker-ci-cd-BeTardif
7 Deleting the contents of '/home/runner/work/lab0-00-infrastructure-git-docker-ci-cd-BeTardif/lab0-00-infrastructure-git-
  docker-ci-cd-BeTardif'
8 ▶ Initializing the repository
9 ▶ Disabling automatic garbage collection
10 ▶ Setting up auth
11 ▶ Fetching the repository
12 ▶ Determining the checkout info
13 /usr/bin/git sparse-checkout disable
14 /usr/bin/git config --local --unset-all extensions.worktreeConfig
15 ▶ Checking out the ref
16 /usr/bin/git log -1 --format=%H
17 660b94304e1a12545c0408eb173c4ae8f80284f3

```

Dans ce *Checkout* github regarde les modifications à apportées à l'application afin qu'elle soit à jour avec code nouvellement publié. On peut le voir à partir de la ligne 52 de la capture d'écran présentée.

### Question 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` nous permet d'obtenir l'information concernant l'utilisation du CPU de la VM. Tel qu'il est démontré dans la capture d'écran ci-dessous, les lignes décrivent les éléments qui utilisent le CPU. Par exemple, la deuxième ligne est celle du conteneur docker nommé `containerd`, elle utilise 0.3% de l'attribution émise lors de l'étape cinq du laboratoire quand nous avons initié la machine virtuelle. Un autre exemple serait la première ligne qui indique l'agent LXD qui permet la virtualisation. Celle command utilise aussi 0.3% de la capacité du CPU.

```
Administrator: Command Prompt - lxc exec fiware-1:LOG430-VM1-BTardif -- bash

op - 15:13:55 up 1 day, 11:48, 0 users, load average: 0.00, 0.00, 0.00
Tasks: 79 total, 1 running, 78 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.3 us, 0.0 sy, 0.0 ni, 99.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.3 st
MiB Mem : 3649.5 total, 1813.6 free, 180.4 used, 1655.5 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 3402.3 avail Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
229 root 20 0 1239392 16448 11644 S 0.3 0.4 0:24.11 lxd-agent
10659 root 20 0 1654660 46176 33208 S 0.3 1.2 2:38.77 containerd
12951 root 20 0 10768 3804 3208 R 0.3 0.1 0:00.06 top
1 root 20 0 167460 12844 8260 S 0.0 0.3 0:06.85 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
7 root 20 0 0 0 0 I 0.0 0.0 0:00.31 kworker/0:0-events
8 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 kworker/0:0H-events_highpri
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 0:00.40 ksoftirqd/0
13 root 20 0 0 0 0 I 0.0 0.0 0:03.48 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
21 root 0 0 0 0 0 T 0.0 0.0 0:00.00 idle
```