

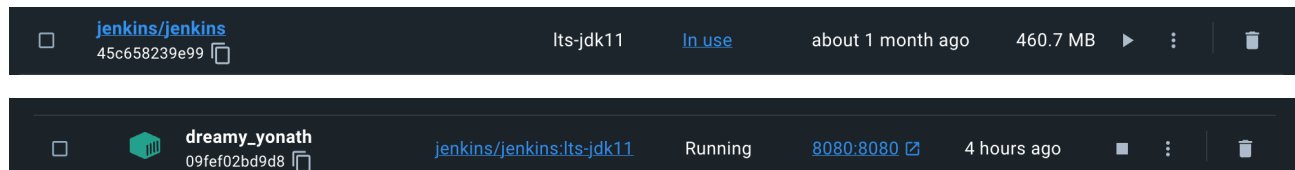
Endy Mehraein

B3-C1-TD Jenkins

1er Etape installation et Configuration

Tout d'abord j'ai fait le choix de faire ce TD via Docker, cela ma permis de monter mes compétences en docker de plus je trouve cela plus pratique.

Je suis donc aller récupéré l'image officielle de Jenkins sur le forum DockerHub
https://hub.docker.com/_/jenkins



Nous avons donc notre image et notre conteneur que nous pouvons lancé.

Suite à cela nous pouvons nous diriger sur le port <http://localhost:8080/>

Un mot de passe d'administrateur nous sera demandé, le mot de passe est dans le fichier initialAdminPassword il suffit de le copié et le renseigné

Une fois chose faite nous pouvons créer un compte administrateur via le formulaire Jenkins

Passons en suite à la création de l'item ou du projet, on renseigne un nom

Saisissez un nom

» Un job existe déjà avec le nom 'Python-Django'

**Construire un projet free-style**

Ceci est la fonction principale de Jenkins qui sert à builder (construire) votre projet. Vous pouvez intégrer tous les outils de gestion de version avec tous les systèmes de build. Il est même possible d'utiliser Jenkins pour tout autre chose qu'un build logiciel.

**Pipeline**

Organise des activités de longue durée qui peuvent s'étendre sur plusieurs agents de construction. Adapté pour la création des pipelines (anciennement connues comme workflows) et/ou pour organiser des activités complexes qui ne s'adaptent pas facilement à des tâches de type libre.

Et nous pouvons sélectionné « Pipeline » qui sera suffisant pour l'exécution de tests.

Nous retrouverons alors notre projet avec des métriques sur les précédent builds.

S	M	Nom du projet ↓	Dernier succès	Dernier échec	Dernière durée
✓	☀	Python-Django	54 mn #59	2 j 1 h #28	1 s ▶

Passons maintenant à la configuration de notre pipeline

Status

</> Changes

▶ Lancer un build

⚙ Configurer

🗑 Supprimer Pipeline

🔍 Full Stage View

🌐 GitHub

✎ Renommer

❓ Pipeline Syntax

🌞 Historique des builds tendance ▼

🔍 Filter builds... /

✓ #59 11 avr. 2023 12:05

✓ #58 11 avr. 2023 12:05

✓ #57 11 avr. 2023 12:04

Stage View

Average stage times:
(Average full run time: ~1s)

	Clonage du dépôt	Tests unitaires
#59 avr. 11 14:05 No Changes	532ms	319ms
#58 avr. 11 14:05 No Changes	568ms	328ms
#57 avr. 11 14:04 No Changes	516ms	338ms

Tout d'abord nous devons récupérer le projet dans laquelle nous voulons faire nos tests, Pour cela il faut spécifier à Jenkins l'adresse du repo cible.

General

Enabled

Description

Execution de tests dans un projet Python-Django stocké dans un repository git

[Plain text] Prévisualisation

☐ Ce build a des paramètres ?

☐ Do not allow concurrent builds

☐ Do not allow the pipeline to resume if the controller restarts

☒ GitHub project

Project url ?

https://github.com/Endy-sys32/B3-C1-Django-Endy-Mehraein.git/

Avancé ▼

✎ Edited

☐ Pipeline speed/durability override ?

Dans cet exemple le repo cible est <https://github.com/Endy-sys32/B3-C1-Django-Endy-Mehraein> qui est un projet Python-Django

Nous allons ensuite configurer la pipeline qui

Definition

Pipeline script

Script ?

```
1 pipeline {
2   agent any
3   stages {
4     stage('Clonage du dépôt') {
5       steps {
6         git branch: 'test', url: 'https://github.com/Endy-sys32/B3-C1-Django-Endy-Mehraein.git'
7       }
8     }
9     stage('Tests unitaires') {
10      steps {
11        dir('pilotage/src') {
12          sh 'python3 -m unittest tests.py'
13        }
14      }
15    }
16  }
17 }
18
```

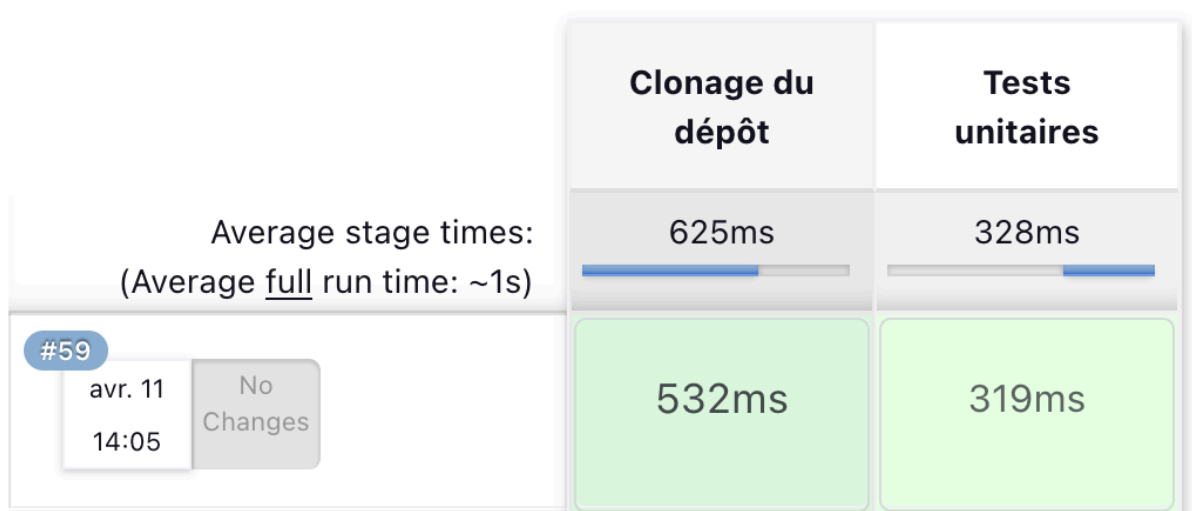
☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

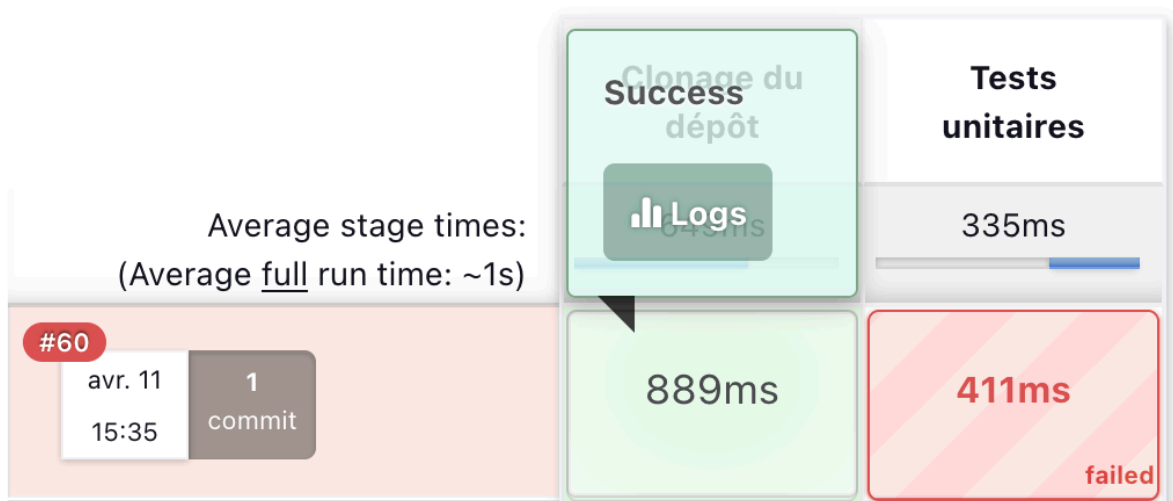
Nous pouvons voir deux stages différents, Clonage du dépôt et Tests unitaires

Dans la phase clonage du dépôt nous récupérons le code du repo et nous nous positionnons sur la branche test

Dans la phase Tests unitaires nous nous déplaçons dans l'arborescence pour accéder au dossier contenant les tests puis nous les exécutons.



Lors de l'exécution des tests nous pouvons donc voir les deux phases avec des logs sur le déroulement de la pipeline.



En cas de fail nous pouvons voir la phases à l'issue du fail
Et nous avons accès à des logs sur les causes de l'erreur

Stage Logs (Tests unitaires)

Shell Script -- python3 -m unittest tests.py (self time 282ms)

```
+ python3 -m unittest tests.py
.....F.....
=====
FAIL: test_division5 (tests.Test)
-----
Traceback (most recent call last):
  File "/var/jenkins_home/workspace/Python-Django/pilotage/src/tests.py", line 42, in test_division5
    self.assertEqual(division(4, 4), 6)
AssertionError: 1.0 != 6

-----

Ran 17 tests in 0.000s

FAILED (failures=1)
```

Ici nous pouvons voir que la fonction `test_division5` qui a pour paramètre 4 et 4 attend la réponse 6 alors que la réponse est 1 il nous précise alors que 1 est différent de 6 donc erreur.

Pour exécuter le build à distance nous pouvons utiliser un trigger qui se déclenchera lors d'un push sur le repo choisi

Construire des déclencheurs

- ☐ Construire après le build sur d'autres projets ?
 - ☐ Construire périodiquement ?
 - ☒ Déclencheur de crochet GitHub pour l'interrogation GITScm ?
 - ☐ Scrutation de l'outil de gestion de version ?
 - ☐ Période d'attente ?
 - ☐ Déclencher les builds à distance (Par exemple, à partir de scripts) ?
-

Et pour finir en cas d'échec ou du succès du build nous pouvons configurer la pipeline afin d'envoyer un mail.

```
    }  
    success {  
        steps {  
            mail to: 'ensy.mehraein@epsi.fr',  
                subject: 'Construction réussie'  
                body: "Le build s'est terminé avec succès"  
        }  
    }  
    failure {  
        steps {  
            mail to: 'ensy.mehraein@epsi.fr',  
                subject: 'Échec de la construction Jenkins',  
                body: "Il y a eu une erreur lors de la construction du projet Jenkins. Veuillez vérifier les journaux de build pour plus d'informations."  
        }  
    }  
}
```