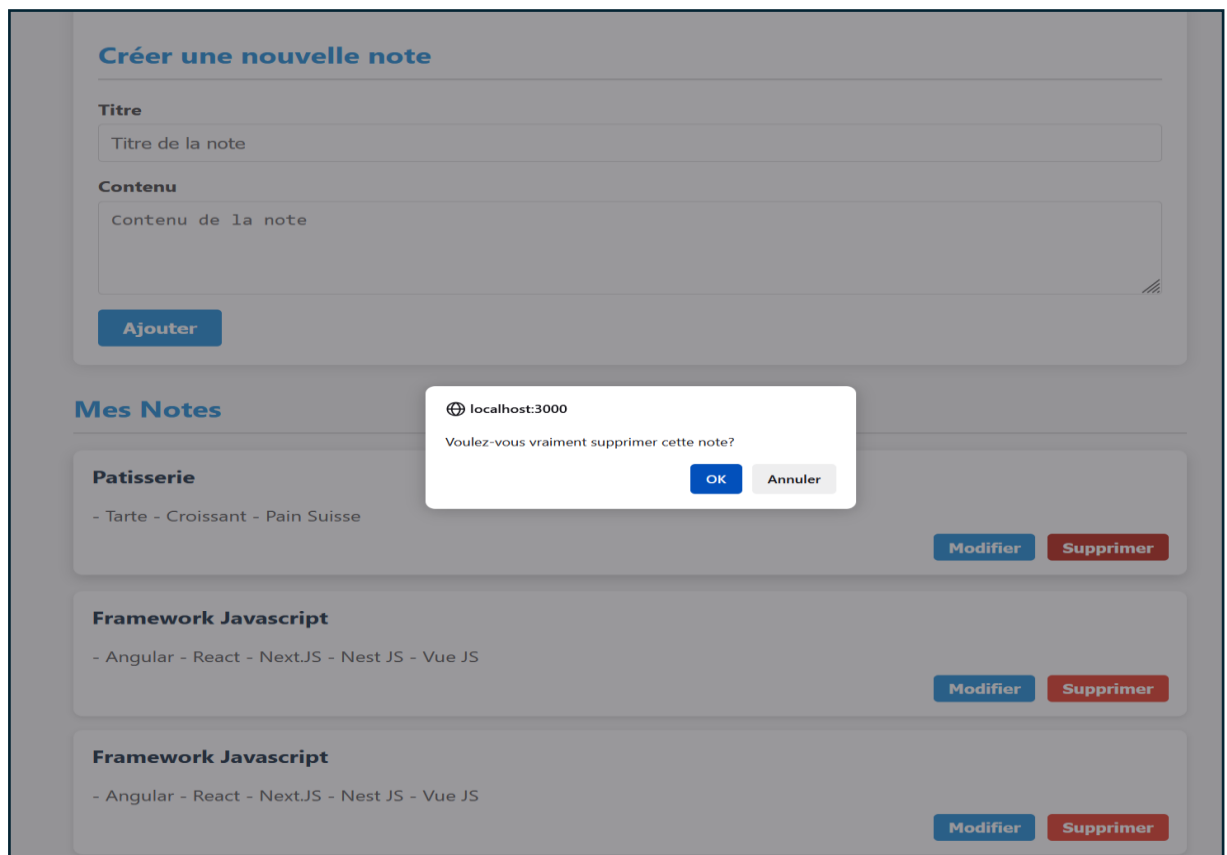


Rapport de Fonctionnement du Pipeline CI/CD pour Todolist-CICD-App



1. Tests automatiques

Configuration :

Le pipeline CI/CD exécute automatiquement les **tests unitaires** à chaque **push** ou **pull request**.

Les tests unitaires sont configurés avec Jest (jest --detectOpenHandles).

Le fichier package.json définit la commande de test: "test": "jest --detectOpenHandles".

Les test sont donc configuré avec Jest et permettent de vérifier :

Les routes API backend dans le fichier server.testjs :

Les fonctionnalités frontend en Javascript dans le fichier public/script.test.js

Résultats :

Grâce à une configuration adaptée, les tests s'exécutent dans des environnements appropriés :

- **Node.js** pour le backend
- **JSDOM** pour le frontend

Les principaux scénarios de test Jest incluent :

- La création, récupération, mise à jour et suppression des notes
- **La validation des données** pour éviter les entrées invalides

2. CI qui échoue en cas d'erreur

Configuration :

Le pipeline GitHub Actions est conçu pour **interrompre l'exécution** si un test échoue.

```
Run tests
7
8 jest-haste-map: Haste module naming collision: notes-app
9   The following files share their name; please adjust your hasteImpl:
10    * <rootDir>/package.json
11    * <rootDir>/creation_note/package.json
12
13 FAIL ./server.test.js
14   Notes API
15     ✓ GET /api/notes doit retourner toutes les notes (84 ms)
16     ✓ POST /api/notes doit créer une nouvelle note (19 ms)
17     ✓ PUT /api/notes/:id doit mettre à jour une note (11 ms)
18     ✓ DELETE /api/notes/:id doit supprimer une note (14 ms)
19     ✗ POST /api/notes doit échouer si le titre est vide (14 ms)
20
21   • Notes API > POST /api/notes doit échouer si le titre est vide
22
23     expect(received).not.toBe(expected) // Object.is equality
24
25     Expected: not ""
26
27     145 |   // Ce test devrait échouer si nous n'avons pas de validation côté serveur
28     146 |   // Mais pour simuler un échec dans la CI, on force l'échec ici
29     > 147 |     expect(invalidNote.title).not.toBe('');
30         |                                     ^
31     148 |   });
32     149 |
33     150 |   // Clean up resources after all tests
34
35     at Object.toBe (server.test.js:147:35)
36
37 FAIL public/script.test.js
38   • Test suite failed to run
39
40   • Validation Error:
41
42     Test environment jest-environment-jsdom cannot be found. Make sure the testEnvironment configuration option points to an existing node module.
43
44     Configuration Documentation:
45     https://jestjs.io/docs/configuration
46
```

La configuration est définie dans `.github/workflows/main.yml`, où le **job test** est exécuté en premier

Résultats :

Si un test échoue, l'erreur est détectée immédiatement. Par exemple :

POST /api/notes doit échouer si le titre est vide.

Le pipeline s'arrête automatiquement, empêchant ainsi le déploiement d'un code défectueux.

Un retour visuel est affiché directement sur **GitHub** avec un indicateur :

✅ **Vert** si les tests réussissent

❌ **Rouge** si un échec est détecté

3. Déploiement automatique après merge

Configuration :

Le **déploiement est conditionnel** et ne s'exécute que si un **push** est effectué sur `main` ou `master`.

Définition dans le workflow YAML :

```
deploy:
  needs: docker-build
  runs-on: ubuntu-latest
  # S'exécute uniquement lors d'un push sur la branche principale (merge) et non lors des pull requests
  if: github.event_name == 'push' && (github.ref == 'refs/heads/main' || github.ref == 'refs/heads/master')
```

Le processus simulé pour démonstration est appliqué sur le `server.js` à la racine du projet.

Résultats :

Ce processus **ne se déclenche pas sur les pull requests**, garantissant ainsi une stabilité avant mise en production.

La séquence complète du pipeline suit donc ces étapes :

Tests → Build → Docker Build → Déploiement

Une notification est générée avec **les informations du dernier commit** et un journal détaillé des étapes exécutées.

12 workflow runs		Event	Status	Branch	Actor
✓	modif server delete et update une note qd on deploie notre app avec d...	main	11 hours ago 4m 23s	...	
Notes App CI #12: Commit 675579d pushed by CStalenes					
✓	switch docker-compose en docker compse, add docker ps et modif mdp en...	main	12 hours ago 4m 46s	...	
Notes App CI #11: Commit b1cfaed9 pushed by CStalenes					
✗	ajout job docker-build avec image app, reseau , conteneur puis maj jo...	main	12 hours ago 4m 34s	...	
Notes App CI #10: Commit 9aa15d1 pushed by CStalenes					
✓	ajout fic style et index pour le test dans script et modif test sante...	main	yesterday 2m 5s	...	
Notes App CI #9: Commit 4f53983 pushed by CStalenes					
✗	amelioration server pour resoudre connection et amelioration workflow...	main	yesterday 1m 26s	...	
Notes App CI #8: Commit 63166ec pushed by CStalenes					
✗	creation script dans public, modif du jest.config, modif script.test.js	main	yesterday 3m 22s	...	
Notes App CI #7: Commit 0089e7a pushed by CStalenes					
✗	modif du jest.conf add dependance jest-env-jsdom, modif expect(invali...	main	yesterday 25s	...	
Notes App CI #6: Commit 9a5f7c1 pushed by CStalenes					
✗	modi du fic server pour docker et creation dokerfile dockercompose et...	main	yesterday 29s	...	
Notes App CI #5: Commit ba80edf pushed by CStalenes					
✗	ajout job deploy et pipelin cd	main	2 days ago 27s	...	
Notes App CI #4: Commit b031bb5 pushed by CStalenes					
✗	modification package.json script.test.js et ajout server.test.js	main	2 days ago 28s	...	
Notes App CI #3: Commit 93a979b pushed by CStalenes					
✗	ajout pack-lock.json	main	2 days ago 28s	...	
Notes App CI #2: Commit 83b7c32 pushed by CStalenes					
✗	first commit	main	2 days ago	...	

4. Déploiement via Docker

Configuration :

Maintenant que nous avons fini la configuration du pipeline en CI/CD on va utiliser docker pour déployer notre application avec une image.

On va donc procéder en finalisant la construction de notre fichier Dockerfile pour la construction de l'image de l'application à la racine du projet puis un fichier docker-compose.yml pour orchestrer le lancement des service Node.js et MySQL.

L'application est **conteneurisée avec Docker**, permettant un **déploiement stable et reproductible**.

On a donc :

- Job docker-build qui construit et teste l'image
- Job deploy qui utilise Docker Compose pour le déploiement

Test d'intégration docker :

- ✓ Création d'un **réseau Docker dédié**
- ✓ Lancement des **conteneurs MySQL et Node.js**
- ✓ Vérification du bon fonctionnement des **routes API**

L'image Docker est correctement construite et donc on va procéder au Déploiement complet via docker compose sur server.js avec commande :

docker compose up -d

Les **logs** des conteneurs permettent de vérifier le bon fonctionnement de l'application.

Conclusion :

Le pipeline CI/CD implémente toutes les étapes demandées et fonctionne correctement :

1. ✓ Tests automatiques avec Jest qui exécute tous les tests unitaires
2. ✓ CI avec détection d'erreur qui implique un échec du pipeline en cas de test non passant
3. ✓ Déploiement automatique après merge déclenché uniquement sur main/master après validation des modification.
4. ✓ Déploiement en production avec Docker pour la construction et déploiement de conteneurs

Cette structure assure ainsi la qualité du code, l'intégrité de l'application et un déploiement sans friction après validation des modifications.