Point de théorie · semaine 2

CI/CD

Un robot vérifie chaque Pull Request. Un autre livre en production. Vous dormez.

5IDEV · Projet d’intégration de développement · 1

« Ça marche sur ma machine »

- Votre binôme merge feature/notifications
- Une colonne renommée, une requête oubliée
- main = production → erreur 500 à 18h02
- Vous le découvrez dimanche soir, avant la démo

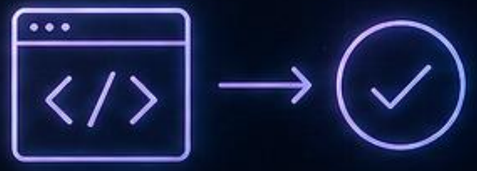
Plus il est trouvé tard, plus il coûte cher

Sur ma machine | **Dans la PR (CI)** | En production

secondes | **minutes** | une soirée + votre réputation

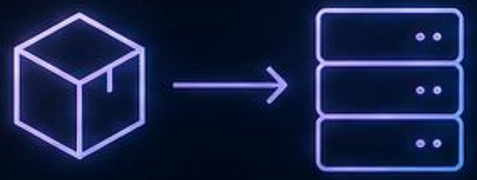
La **CI** attrape les bugs dans la PR. C'est là qu'on veut être.

CI · CD · CD



Intégration continue

À chaque push
Installer · Tester · Vert ou rouge



Livraison continue

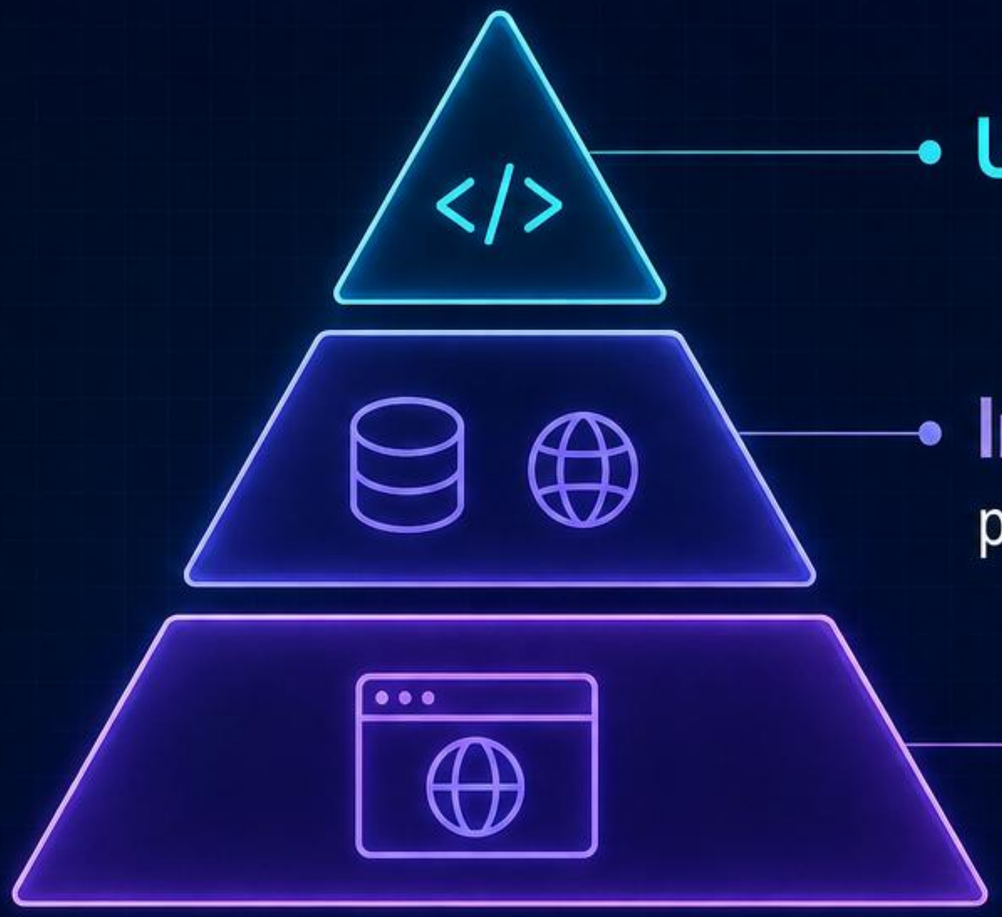
Version déployable
Un humain appuie

Déploiement continu

Vert sur `main`
En prod dans 3 minutes

Dans ce cours : `main` = production. **Déploiement continu.**

Testez le cœur, pas les getters



- **Unitaires** · une fonction isolée · ms · validateBody(””)
- **Intégration** · HTTP + base · POST /messages par un non-membre → 403
- **E2E** · un vrai navigateur · Alice envoie, le message apparaît

Pour votre chat : envoi, lecture, permissions, validation.

Préparer · Agir · Vérifier

Pest

```
it('refuse un non-membre (403)')
```

pytest

```
def test_un_non_membre_est_refuse()
```

Vitest

```
it("un non-membre est refusé (403)")
```

`php artisan test` · `pytest` · `npm test`

Une VM vierge, puis détruite

1. `>_` git push
2. GitHub lit `.github/workflows/*.yml`
3. Une VM Ubuntu vierge démarre
4. checkout → setup → install → tests
5. ✓ / ✗ Vert ou rouge, la VM est détruite

ⓘ Rien ne survit d'un run à l'autre. Tout doit être dans le YAML.

Workflow → Jobs → Steps

- **Workflow :** un fichier YAML, on: dit quand
- **Job :** sa propre machine, parallèle par défaut
- **Step :** une ligne de la to-do list, en séquence dans la machine
- **uses:** = action toute faite · **run:** = commande shell

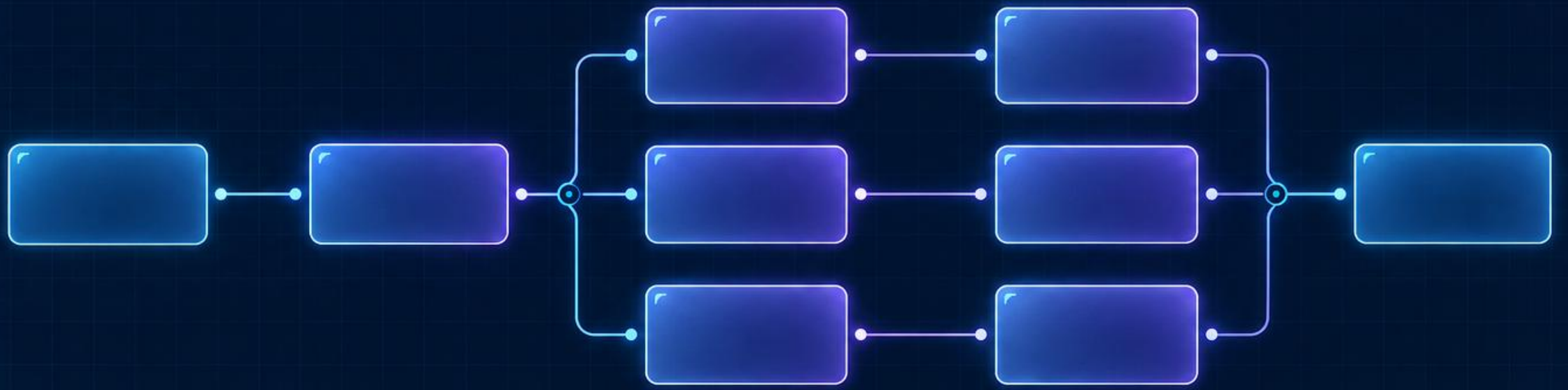
on:

- pull_request → vérifier avant le merge 
- push sur main → vérifier et déployer 
- workflow_dispatch → bouton « Run workflow » 
- schedule → chaque nuit 

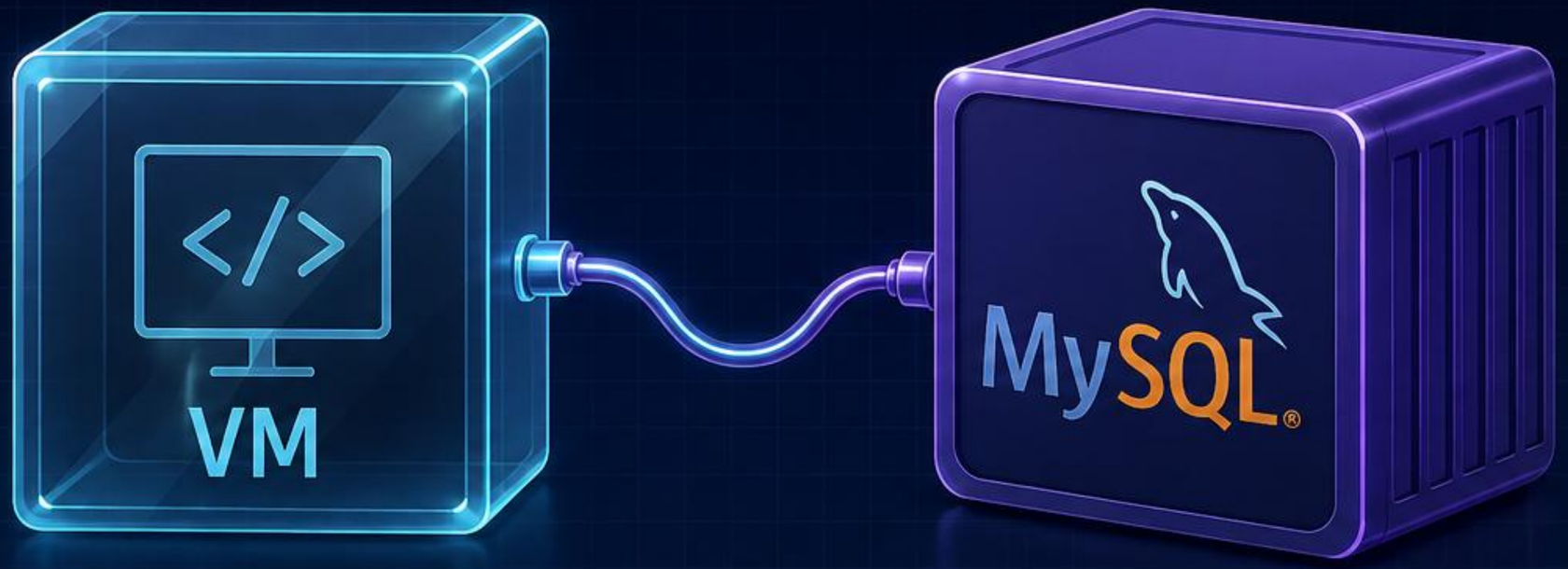
⚠ Pas de on: push tout nu : ciblez main + pull_request.

needs · matrix · concurrency

- **needs:** tests → ordre garanti, pas de E2E si les unitaires sont rouges
- **matrix:** node: [22, 24, 26] → 3 machines en parallèle
- **cancel-in-progress:** true → un nouveau push annule le run précédent



SQLite pour la vitesse, MySQL pour la fidélité




```yaml
services:
  mysql:
    image: mysql:8.4
    options: --health-cmd="mysqladmin ping"
```

Le conteneur démarre à côté du job, sur 127.0.0.1:3306. Le health check n'est pas optionnel.

Jamais de .env commité

- Valeurs jetables (DB_PASSWORD: root d'un MySQL de 3 minutes) → en clair dans env:
- Vraies valeurs → Settings → Secrets → ${{ secrets.NOM }}, masqué dans les logs
- Ce que le framework attend → cp .env.example .env + key:generate

Votre .env.example est lu par la CI : gardez-le complet.

Votre hébergeur déploie déjà

- PaaS managé (Vercel, Render, Railway, Fly…) : 3 clics, auto-deploy + previews
- VPS + Coolify / Dokploy : votre serveur, le même confort push-to-deploy
- Piège : il déploie dès le push, sans attendre la CI

Protection de branche

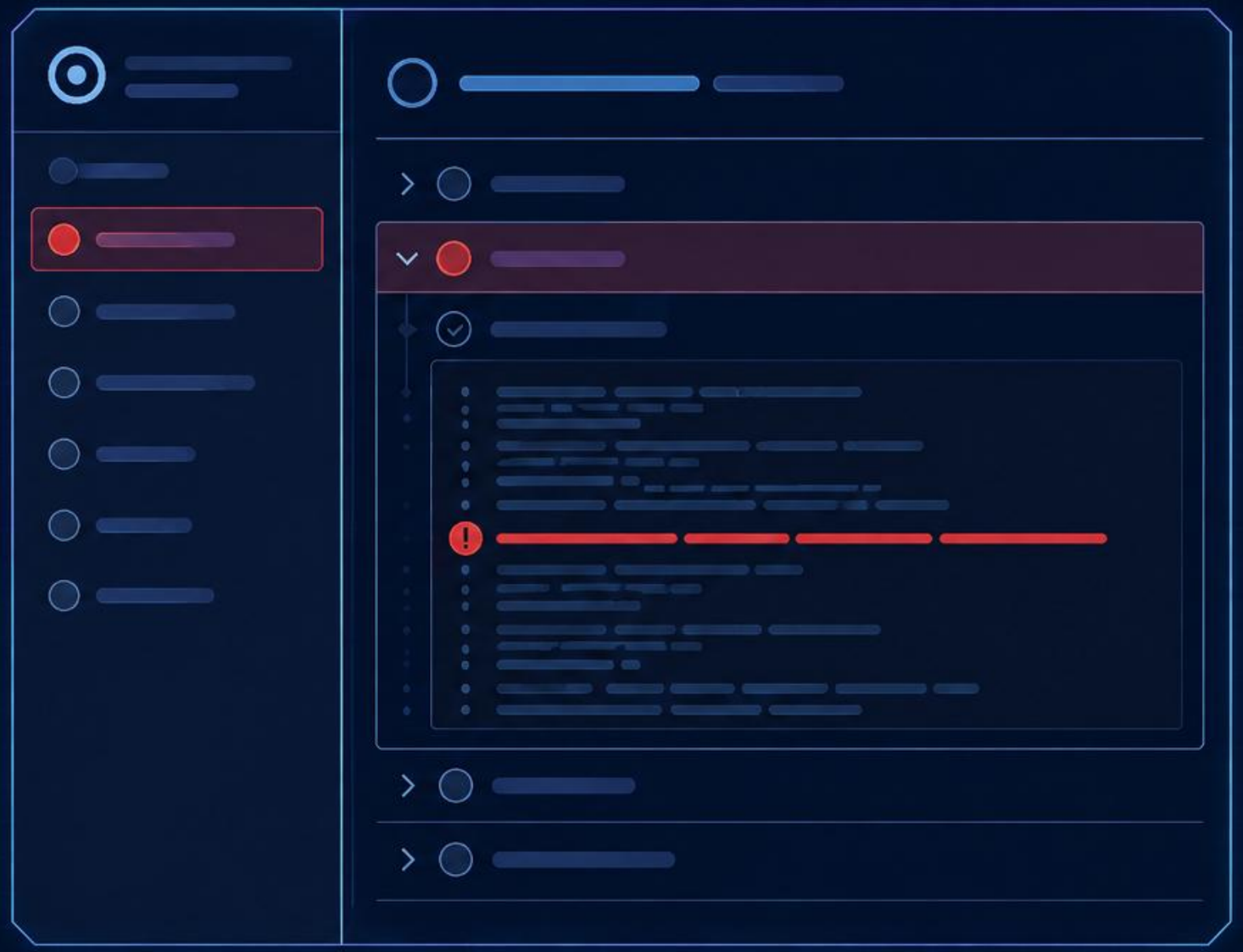
Settings → Branches → main

- [x] Require a pull request before merging
- [x] Require status checks to pass
- [x] Require approvals : 1

Sans ça, votre CI est décorative.

Lire un run, sans paniquer

1. Le job rouge, à gauche
2. Le step rouge, déjà déplié
3. Le log : la première erreur, pas la dernière
4. Reproduire en local avec la même commande
5. Re-run seulement si c'est aléatoire



Avant la revue d’avancement n°1

- Workflow sur pull_request + push vers main
- Un test d’intégration sur le cœur, vert en CI
- Job de style en parallèle
- Protection de branche activée
- Déploiement auto vérifié
- Badge dans le README

Repo d’exemple : github.com/opmvpc/cicd26