Guide Déploiement Continu (CD) – Fintech-test

Ce document explique, en français et pas à pas, le pipeline de Déploiement Continu (CD) de ce dépôt. Il couvre les bases (CI vs CD), les runners GitHub (hébergés vs auto-hébergés), l'installation d'un runner self-hosted avec GitHub CLI, l'architecture de déploiement locale (UAT et PROD en blue/green), le fonctionnement du workflow de CD, et les problèmes résolus au cours de la mise en place.

Voir aussi le guide CI: `docs/CI-PIPELINE.md`.

---

1) CI vs CD – c'est quoi la différence ?

- Intégration Continue (CI): on s'assure automatiquement que l'application se construit, démarre, passe les tests et les scans (qualité/sécurité). Objectif: détecter vite.
- Déploiement Continu (CD): on publie automatiquement une version validée vers un environnement (ici local), avec vérifications de santé et possibilités de rollback.

Dans ce projet: CI construit et valide; CD déploie localement via Docker Compose sur un runner auto-hébergé.

---

2) Runners GitHub: hébergés vs auto-hébergés

- Runners hébergés (hosted): fournis par GitHub, éphémères (Ubuntu/Windows/macOS). Idéal CI.
- Runners auto-hébergés (self-hosted): vos machines (VM, serveur, PC, WSL2). Idéal quand on doit accéder au réseau/aux ports locaux ou à un environnement spécifique.

Ici, le CD est local: on lance des stacks Docker Compose sur la machine du runner. D'où l'usage d'un runner self-hosted avec Docker et les ports disponibles.

Pré-requis côté runner:

- Docker Engine + Docker Compose installés
- Accès aux ports: 5001/5002 (API), 5433/5434 (Postgres)
- Droits suffisants pour lancer des conteneurs

---

3) Installer un runner self-hosted (UI GitHub ou GitHub CLI)

Méthode UI (simple):

1. GitHub > dépôt > Settings > Actions > Runners > New self-hosted runner
2. Choisir l'OS (Linux recommandé; WSL2 sur Windows fonctionne très bien)
3. Suivre les commandes affichées (télécharger, extraire, configurer, démarrer en service)

Méthode GitHub CLI (avancée):

- Installer GitHub CLI: https://cli.github.com/
- Se connecter: `gh auth login`
- Générer un token d'enregistrement pour ce dépôt:

```
# Windows PowerShell
$token = gh api -X POST repos/:owner/:repo/actions/runners/registration-token -q
.token
```

Configurer le runner (exemples):

Linux/WSL2 (bash):

```
wget https://github.com/actions/runner/releases/download/v2.328.0/actions-runner-
linux-x64-2.328.0.tar.gz
mkdir -p ~/actions-runner && cd ~/actions-runner
tar xzf ~/actions-runner-linux-x64-2.328.0.tar.gz
./config.sh --url https://github.com/<owner>/<repo> --token "$token" --name
"local-runner" --labels "self-hosted,local"
./run.sh    # ou installez en service
```

Windows PowerShell (natif):

```
Invoke-WebRequest -Uri
https://github.com/actions/runner/releases/download/v2.328.0/actions-runner-win-
x64-2.328.0.zip -OutFile runner.zip
Expand-Archive runner.zip -DestinationPath C:\actions-runner
Set-Location C:\actions-runner
./config.cmd --url https://github.com/<owner>/<repo> --token $token --name "local-
win" --labels "self-hosted,local"
./run.cmd   # ou installez en service
```

Notes:

- Vous pouvez enregistrer le runner au niveau organisation via l'endpoint org.
- Ouvrez les ports nécessaires dans le firewall.

---

4) Architecture de déploiement locale

Deux environnements Docker Compose:

- UAT (recette locale):
  - Fichier: `docker/docker-compose.uat.yml`
  - Services: Postgres (port hôte 5433), API Flask (port hôte 5001)
  - Health check: l'API expose `/health` (attendu avant de continuer)
- PROD (locale, blue/green):
  - Fichier: `docker/docker-compose.prod.yml`
  - Services: Postgres (port hôte 5434), API en double (profils `blue` et `green`)

    - Seule la couleur active publie le port 5002; bascule contrôlée après validation santé

Smoke test (optionnel): un test k6 léger après chaque déploiement (ex: `/health`) pour vérifier la base.

---

5) Le workflow de CD

Fichier: `.github/workflows/cd-local.yml`

Déclencheurs:

- Push sur `main`: déclenche automatiquement un déploiement UAT local (runner self-hosted requis). Par défaut en `MODE=build` pour ne pas dépendre d'un registry.
- Tags `v*`: déploiement versionné avec gate d'approbation vers PROD (blue/green). Par défaut en `MODE=build` dans ce repo, mais conseillé en `MODE=image` si vous publiez vos images via la CI.
- Manuels: `workflow_dispatch` (inputs) pour lancer/paramétrer un déploiement ou un rollback à la demande.

Inputs (principaux):

- `action`: `deploy` (déployer) ou `rollback`
- `mode`: `build` (build local) ou `image` (pull GHCR)
- `image_name`, `image_tag` (si mode=image)
- `run_smoke`: `true/false` – lancer k6 après le déploiement
- `prune_uat`: `true/false` – purger volume DB UAT (rare, migrations majeures)
- `target`, `image_tag_previous`: pour rollback

Jobs:

- `setup-runner-check`: vérifie Docker/Compose
- `deploy-uat`: prépare `.env.uat`, déploie UAT, attend `/health`, lance k6 si demandé
- `approval`: gate de prod (approbation manuelle)
- `deploy-prod` (blue/green): déploie sur le slot inactif, vérifie santé interne, bascule le port 5002, vérifie santé externe, lance k6 si demandé
- `rollback`: redéploie une version précédente (UAT ou PROD)

Diagnostics en cas d'échec:

- Dump `docker compose ps` et `docker compose logs` (API)
- `--remove-orphans` utilisé pour éviter les conteneurs résiduels

---

6) Lancer le CD (UI et gh CLI)

UI GitHub Actions:

- Actions > "CD Local (UAT & Prod)" > Run workflow
- Exemple d'inputs:
    - action: deploy
    - mode: build (ou image)
    - run_smoke: true

    - prune_uat: false (sauf cas particuliers)

GitHub CLI:

```
gh workflow run "CD Local (UAT & Prod)" --ref main -f action=deploy -f mode=build
-f run_smoke=true -f prune_uat=false
```

Accès:

- UAT: http://localhost:5001 (santé: `/health`)
- PROD: http://localhost:5002 (santé: `/health`)

Vérifier l'application (UAT et PROD):

- URL d'accueil: `http://localhost:5001` (UAT) ou `http://localhost:5002` (PROD)
- Auth:
    - S'inscrire: `/register` (formulaire)
    - Se connecter: `/login` (formulaire)
    - Tableau de bord (protégé): `/dashboard`
- Comptes de test (semés automatiquement si DB vide): `admin`, `alice`, `bob`, `charlie` avec mot de passe `password123`.
- Base de données:
    - UAT Postgres: port hôte `5433`
    - PROD Postgres: port hôte `5434`
    - Exemple psql UAT: `psql -h localhost -p 5433 -U bankuser -d bankdb`

---

7) Blue/Green expliqué simplement

- Deux slots: `blue` (actif), `green` (inactif)
- On déploie sur la couleur inactive
- On vérifie la santé
- On bascule le port public (5002) sur la nouvelle couleur
- Avantage: rollback instantané si problème (il suffit de rebascule)

---

8) Rollback (retour arrière)

- Inputs: `action=rollback`, `target=uat|prod`, `image_tag_previous=<tag|sha>`
- Permet de revenir à une image connue rapidement

---

9) Problèmes rencontrés et solutions

1. Scripts ignorés / exécutabilité

    - Raison: patterns d'ignore, fins de lignes Windows
    - Solution: ajuster `.gitignore`, ajouter `.gitattributes` (LF pour .sh), forcer `chmod +x` dans le workflow

2. Timeouts santé UAT (API qui redémarre)
    - Cause: exceptions au démarrage → `/health` ne répond pas
    - Solution: diagnostics étendus dans le script, délais plus longs, correctifs app

3. Erreur Postgres `value too long for type character varying(128)`
    - Cause: colonne `password_hash` trop courte pour des hash modernes (scrypt)
    - Solutions complémentaires appliquées:
        - Modèle: `db.String(255)`
        - Option de CD: `prune_uat=true` pour reset propre du volume lors d'un changement de schéma
        - Auto-migration au démarrage: si Postgres, on élargit en `VARCHAR(255)` automatiquement pour éviter les crashs sur volumes existants

4. Conteneurs orphelins
    - Solution: `docker compose up --remove-orphans`

5. Avertissement Compose `version` obsolète
    - Solution: retirer le champ `version` des fichiers Compose

---

10) Bonnes pratiques et pistes d'évolution

- Secrets: n'en commitez pas; utilisez des environnements GitHub et des secrets d'environnement
- Reproductibilité: mode=image + GHCR pour déployer l'image construite par la CI
- Observabilité: conservez les logs d'échec; ajoutez de la télémétrie si besoin
- Migrations: adopter Alembic pour gérer l'évolution du schéma proprement

---

11) Fichiers utiles

- Workflow CD: `.github/workflows/cd-local.yml`
- Compose UAT: `docker/docker-compose.uat.yml`
- Compose PROD: `docker/docker-compose.prod.yml`
- Scripts: `scripts/deploy_blue_green_local.sh`, `scripts/wait_for_http.sh`
- Exemples d'env: `docker/.env.uat.example`, `docker/.env.prod.example`
