Protocole d'intégration et de déploiement continu (CI/CD)

Ce document décrit le protocole CI/CD utilisé pour les services du projet, en prenant `penpal-ai-db-service` comme exemple. L'infrastructure d'exécution se trouve chez Hetzner, orchestrée via Coolify. Les images Docker sont construites et poussées sur GHCR, puis déployées automatiquement selon la branche.

Vue d'ensemble

- **Branches → environnements**:
  - **develop → staging**: `staging."service".penpal-ai.maksou.dev`
  - **main → production**: `prod."service".penpal-ai.maksou.dev`
- **Registre d'images**: `ghcr.io/<OWNER>/<IMAGE_NAME>`
- **Déploiement**: déclenché par webhook Coolify après build & push de l'image Docker
- **Bases de données**: une instance MongoDB dédiée par environnement (staging et production)
- **Variables d'environnement**: gérées directement dans Coolify pour chaque service/environnement

Déclencheurs

- Push sur `develop` ou `main`
- Création d'un tag versionné `vX.Y.Z` (push de tag)

Pipeline CI (GitHub Actions)

Le workflow effectue d'abord tous les tests (lint, unitaires, couverture, E2E), puis construit et pousse l'image Docker, avant d'appeler le webhook Coolify pour déployer.

Extrait – Job de tests:

```
jobs:
  test:
    name: Unit & E2E Tests
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: npm

      - name: Install dependencies
        run: npm ci

      - name: Run lint
        run: npm run lint

      - name: Run unit tests
        run: npm test -- --runInBand
```

```
      - name: Run unit tests with coverage
        run: npm run test:cov -- --runInBand

      - name: Run E2E tests
        run: npm run test:e2e -- --runInBand
```

Extrait – Calcul des tags & build multi-arch:

```
      - name: Compute tags
        id: tags
        run: |
          if [[ "${GITHUB_REF}" == refs/tags/v*.*.* ]]; then
            # Version tag (v1.2.3)
            VERSION=${GITHUB_REF#refs/tags/v}
            echo "version_tag=${VERSION}" >> $GITHUB_OUTPUT
            echo "is_version=true" >> $GITHUB_OUTPUT
          else
            # Branch push
            BRANCH_TAG="${REF_SLUG}-latest"
            echo "branch_tag=${BRANCH_TAG}" >> $GITHUB_OUTPUT
            echo "is_version=false" >> $GITHUB_OUTPUT
          fi
          SHA_TAG="sha-${GITHUB_SHA::7}"
          echo "sha_tag=${SHA_TAG}" >> $GITHUB_OUTPUT

      - name: Build & push image (version)
        if: steps.tags.outputs.is_version == 'true'
        uses: docker/build-push-action@v6
        with:
          context: .
          file: ./Dockerfile
          platforms: linux/amd64,linux/arm64
          push: true
          tags: |
            ghcr.io/${{ env.OWNER }}/${{ env.IMAGE_NAME }}:${{
steps.tags.outputs.version_tag }}
            ghcr.io/${{ env.OWNER }}/${{ env.IMAGE_NAME }}:${{
steps.tags.outputs.sha_tag }}
            ghcr.io/${{ env.OWNER }}/${{ env.IMAGE_NAME }}:latest

      - name: Build & push image (branch)
        if: steps.tags.outputs.is_version == 'false'
        uses: docker/build-push-action@v6
        with:
          context: .
          file: ./Dockerfile
          platforms: linux/amd64,linux/arm64
          push: true
          tags: |
            ghcr.io/${{ env.OWNER }}/${{ env.IMAGE_NAME }}:${{
steps.tags.outputs.branch_tag }}
```

```
            ghcr.io/${{ env.OWNER }}/${{ env.IMAGE_NAME }}:${{
steps.tags.outputs.sha_tag }}
```

Extrait – Déclenchement Coolify:

```
      - name: Trigger Coolify deploy (staging)
        if: github.ref_name == 'develop'
        run: |
          curl -fsSL -X POST \
            -H "Authorization: Bearer ${{ secrets.COOLIFY_API_TOKEN }}" \
            "${{ secrets.COOLIFY_STAGING_WEBHOOK_URL }}"

      - name: Trigger Coolify deploy (production)
        if: github.ref_name == 'main'
        run: |
          curl -fsSL -X POST \
            -H "Authorization: Bearer ${{ secrets.COOLIFY_API_TOKEN }}" \
            "${{ secrets.COOLIFY_PROD_WEBHOOK_URL }}"
```

Détails d'implémentation

- **Tests**: la qualité est garantie avant toute image Docker via `npm run lint`, tests unitaires, couverture, et tests E2E.
- **Build multi-architecture**: `linux/amd64` et `linux/arm64` via Buildx/QEMU.
- **Tagging des images**:
  - Sur un tag `vX.Y.Z`: pousse `vX.Y.Z`, `sha-<7>`, et `latest`.
  - Sur un push de branche (`develop`/`main`): pousse `<branch>-latest` et `sha-<7>`.
- **Déploiement**:
  - Push sur `develop` → webhook `staging` (Coolify) → déploiement sur `staging."service".penpal-ai.maksou.dev`.
  - Push sur `main` → webhook `production` (Coolify) → déploiement sur `prod."service".penpal-ai.maksou.dev`.
  - NB: Un tag `vX.Y.Z` publie les images mais ne déclenche pas, à lui seul, le déploiement en production (le webhook prod est lié à `main`). Pour livrer en prod, merge/push sur `main`.

Secrets requis (GitHub)

- `COOLIFY_API_TOKEN`: jeton d'API pour authentifier l'appel webhook Coolify
- `COOLIFY_STAGING_WEBHOOK_URL`: URL du webhook de l'application staging dans Coolify
- `COOLIFY_PROD_WEBHOOK_URL`: URL du webhook de l'application production dans Coolify
- `GITHUB_TOKEN`: fourni par GitHub Actions, utilisé pour GHCR (login/push)

Variables d'environnement (Coolify)

Définies par environnement au niveau de l'application :

- `NODE_ENV` (`staging` | `production`)
- `PORT`

- `MONGODB_URI` (staging et prod pointent vers des bases distinctes)
- Toute clé/API tierce requise par le service

Les variables ne sont pas stockées dans le dépôt mais gérées dans Coolify. Chaque service possède son propre jeu de variables pour `staging` et `production`.

Bases de données

- **Staging**: MongoDB dédiée pour l'environnement de test
- **Production**: MongoDB dédiée pour l'environnement live

Stratégie de release

1. Préparer la release sur `develop` (tests verts, validations fonctionnelles)
2. Taguer la version: `git tag vX.Y.Z && git push --tags` (pousse les images versionnées et `latest`)
3. Merger sur `main` pour déclencher le déploiement production via webhook

Rollback

- Option 1 (rapide): dans Coolify, sélectionner/redéployer une image précédente (ex. tag `sha-abcdef0` ou un tag versionné `vX.Y.Z`).
- Option 2: revert Git sur `main` puis push (déclenche un nouveau build/push et un déploiement du dernier bon état).

Onboarding d'un nouveau service

1. Créer un workflow CI similaire (`.github/workflows/ci.yml`) en adaptant `IMAGE_NAME`, chemins et versions Node.
2. Publier une image Docker multi-arch sur GHCR (login via `GITHUB_TOKEN`).
3. Créer deux applications dans Coolify (staging et production) pointant vers l'image GHCR.
4. Configurer les variables d'environnement dans chaque application (staging/prod), y compris les URI MongoDB.
5. Récupérer les URLs de webhook Coolify et les déposer dans les secrets GitHub du dépôt (`COOLIFY_*`).
6. Vérifier le mapping domaine ↔ application :
   - `staging."service".penpal-ai.maksou.dev`
   - `prod."service".penpal-ai.maksou.dev`

Vérification post-déploiement

- Vérifier l'endpoint de santé (ex. `/health`) du service déployé
- Contrôler les logs dans Coolify
- S'assurer que les métriques/alertes (si configurées) ne remontent aucune anomalie

Dépannage (quick checklist)

- Pipeline rouge avant build ? Corriger lint/tests unitaires/E2E.
- Image non disponible ? Vérifier le login GHCR et les tags produits.

- Déploiement non déclenché ? Vérifier `github.ref_name` et la cible (`develop` vs `main`), ainsi que les secrets `COOLIFY_*`.
- Variables manquantes ? Vérifier la configuration dans Coolify (staging/prod).

Référence

- Workflow CI: `penpal-ai-db-service/.github/workflows/ci.yml`