Créer une nouvelle webapp Node.js — Workflow complet

Ce document décrit le workflow complet pour créer, déployer et maintenir une nouvelle application web Node.js/Hono sur l'infrastructure homelab. Il couvre la création du dépôt Gitea, le pipeline CI/CD via runner, et le déploiement sur Unraid.

---

1. Stack technique

Couche	Technologie	Rôle
Runtime	Node.js 22+ (ESM)	Exécution du serveur
Framework	Hono v4	Routing HTTP, middlewares
Base de données	PostgreSQL 17	Persistance des données
Migrations BDD	node-pg-migrate	Versionning du schéma BDD
Frontend	Alpine.js + Tailwind CSS	Interactivité et style (CDN, pas de build)
Versionning	Gitea (gitea.thymon.fr)	Dépôt de code + registry Docker
CI/CD	Gitea Runner	Build image Docker + push registry
Déploiement	Unraid + Docker	Hébergement des containers
Reverse Proxy	Nginx Proxy Manager	HTTPS + routing vers les containers

---

2. Prérequis (déjà en place)

- Gitea accessible sur **https://gitea.thymon.fr** avec registry Docker activé (`[packages] ENABLED = true` dans `app.ini`)
- Runner Gitea connecté avec Docker et Node.js installés
- Unraid connecté au registry Gitea (`docker login gitea.thymon.fr` déjà fait)
- Instance PostgreSQL existante accessible en réseau local
- VM Linux de dev avec Node.js 22+ et git installés

3. Étape 1 — Créer le dépôt Gitea

1. Aller sur **https://gitea.thymon.fr**
2. **+ → New Repository**
3. Remplir :
   - Repository name : `nom-de-lapp`
   - Visibility : **Private**
   - ⚠ Ne pas cocher "*Initialize this repository*"
4. Cliquer **Create Repository**

Ajouter les secrets CI/CD

Dans **Settings → Actions → Secrets** du dépôt :

Nom du secret	Valeur
`REGISTRY_USER`	thymon
`REGISTRY_PASSWORD`	token Gitea (généré dans User Settings → Applications)

Ajouter la variable CI/CD

Dans **Settings → Actions → Variables** :

Nom	Valeur
`REGISTRY_URL`	gitea.thymon.fr

> ⚠ Le nom de variable ne doit pas commencer par `GITEA_` (réservé).

---

4. Étape 2 — Initialiser le projet sur la VM de dev

```
# Créer le dossier projet
mkdir -p ~/projets/nom-de-lapp && cd ~/projets/nom-de-lapp

# Copier le socle de base (zip disponible dans Gitea thymon/webapp-node-socle)
# puis dézipper et initialiser git

git init
git checkout -b main
git remote add origin https://gitea.thymon.fr/thymon/nom-de-lapp.git

# Configurer la mémorisation des credentials (une seule fois)
git config --global credential.helper store

# Installer les dépendances et générer le package-lock.json
npm install

# Premier commit
git add .
git commit -m "init: socle Node.js/Hono"
git push -u origin main
```

> ⚠ Le `package-lock.json` doit être committé — le Dockerfile utilise `npm ci` qui l'exige.

---

5. Étape 3 — Tester en local sur la VM

```
# Copier et configurer le .env
cp .env.example .env
# Éditer .env avec les valeurs locales

# Lancer PostgreSQL local (pour le dev uniquement)
docker compose -f docker-compose.dev.yml up -d
```

```
# Démarrer le serveur avec hot-reload
npm run dev
```

Le serveur doit afficher :

```
□□Vérification des migrations...
□ Migrations OK
□ Serveur démarré sur http://localhost:3000
```

Vérifier sur **http://localhost:3000/api/health** — doit retourner `{"status":"ok","db":"ok"}`.

---

6. Étape 4 — Pipeline CI/CD

Le fichier `.gitea/workflows/build.yml` est inclus dans le socle. Il se déclenche à chaque push sur `main` et :

1. Checkout du code
2. Setup Docker Buildx
3. Login sur le registry `gitea.thymon.fr`
4. Build de l'image Docker
5. Push de l'image avec les tags `latest` et `SHA du commit`

Vérifier le résultat dans **https://gitea.thymon.fr/thymon/nom-de-lapp/actions**.

L'image buildée est visible dans **https://gitea.thymon.fr/thymon/nom-de-lapp/packages**.

---

7. Étape 5 — Créer le template Unraid

Sur Unraid, créer le fichier :

```
/boot/config/plugins/dockerMan/templates-user/my-nom-de-lapp_thymon.xml
```

Structure du template XML :

```
<?xml version="1.0"?>
<Container version="2">
  <Name>nom-de-lapp</Name>
  <Repository>gitea.thymon.fr/thymon/nom-de-lapp:latest</Repository>
  <Registry>https://gitea.thymon.fr</Registry>
  <Network>bridge</Network>
  <MyIP/>
  <Shell>sh</Shell>
  <Privileged>false</Privileged>
  <Overview>Description de l'application.</Overview>
  <Category>Tools:Utilities</Category>
  <WebUI>http://[IP]:[PORT:3000]/</WebUI>
  <Config Name="Port: Web UI" Target="3000" Default="3000" Mode="tcp" Type="Port"
Display="always" Required="true" Mask="false">3000</Config>
  <Config Name="Variable: DB_HOST" Target="DB_HOST" Default="192.168.10.xxx" Mode=""
Type="Variable" Display="always" Required="true" Mask="false">192.168.10.xxx</Config>
  <Config Name="Variable: DB_PORT" Target="DB_PORT" Default="5432" Mode="" Type="Variable"
Display="always" Required="true" Mask="false">5432</Config>
  <Config Name="Variable: DB_NAME" Target="DB_NAME" Default="appdb" Mode="" Type="Variable"
Display="always" Required="true" Mask="false">appdb</Config>
  <Config Name="Variable: DB_USER" Target="DB_USER" Default="appuser" Mode="" Type="Variable"
Display="always" Required="true" Mask="false">appuser</Config>
  <Config Name="Variable: DB_PASSWORD" Target="DB_PASSWORD" Default="" Mode="" Type="Variable"
Display="always" Required="true" Mask="true"></Config>
  <Config Name="Variable: PORT" Target="PORT" Default="3000" Mode="" Type="Variable"
Display="advanced" Required="false" Mask="false">3000</Config>
</Container>
```

---

8. Étape 6 — Déployer sur Unraid

1. Créer la base de données sur l'instance PostgreSQL existante :

   ```
   CREATE DATABASE appdb;
   CREATE USER appuser WITH PASSWORD 'motdepasse';
   GRANT ALL PRIVILEGES ON DATABASE appdb TO appuser;
   ```

2. Dans Unraid : **Docker → Add Container** → sélectionner le template `my-nom-de-lapp_thymon`
3. Renseigner les variables (IP PostgreSQL, BDD, user, password)

4. Cliquer **Apply** — le container démarre, les migrations tournent automatiquement
5. Vérifier sur **http://IP-UNRAID:3000/api/health**

---

9. Étape 7 — Configurer NPM

Dans Nginx Proxy Manager → **Add Proxy Host** :

Champ	Valeur
Domain Names	monapp.thymon.fr
Scheme	http
Forward Hostname/IP	IP Unraid
Forward Port	3000
Websockets Support	On
Block Common Exploits	On
Force SSL	On
HTTP/2 Support	On

---

10. Workflow de mise à jour

```
# 1. Développer sur la VM
npm run dev

# 2. Pusher les changements
git add .
git commit -m "feat: nouvelle fonctionnalité"
git push
# → le runner build et push automatiquement la nouvelle image

# 3. Sur Unraid
# Docker → cliquer sur le container → "Check for updates" → "Update"
# Les migrations BDD tournent automatiquement au redémarrage
# Les données PostgreSQL (volume nommé) ne sont jamais touchées
```

---

11. Commandes utiles

Migrations BDD

Action	Commande
Créer une migration	`npm run migrate:create -- nom_migration`
Appliquer les migrations	`npm run migrate:up`
Rollback	`npm run migrate:down`
Statut	`npm run migrate:status`

Dev local

Action	Commande
Démarrer PostgreSQL local	`docker compose -f docker-compose.dev.yml up -d`
Arrêter PostgreSQL local	`docker compose -f docker-compose.dev.yml down`
Démarrer le serveur	`npm run dev`

---

12. Règles absolues

- ⚠ **Ne jamais modifier** une migration déjà appliquée en production — toujours créer une nouvelle migration
- ⚠ **Ne jamais faire de SQL direct** en prod — toujours passer par une migration
- ⚠ **Le `package-lock.json` doit être committé** — requis par `npm ci` dans le Dockerfile
- ⚠ **Les variables sensibles** uniquement dans `.env` (jamais dans le code)
- □ Tout le code est en **ESM** (`import/export`, pas de `require`)
- □ Préférer **jsonb** à **json** pour les colonnes JSON en PostgreSQL

Revision #1
Created 2026-03-19 21:56:49 UTC by thymon
Updated 2026-03-19 21:57:03 UTC by thymon