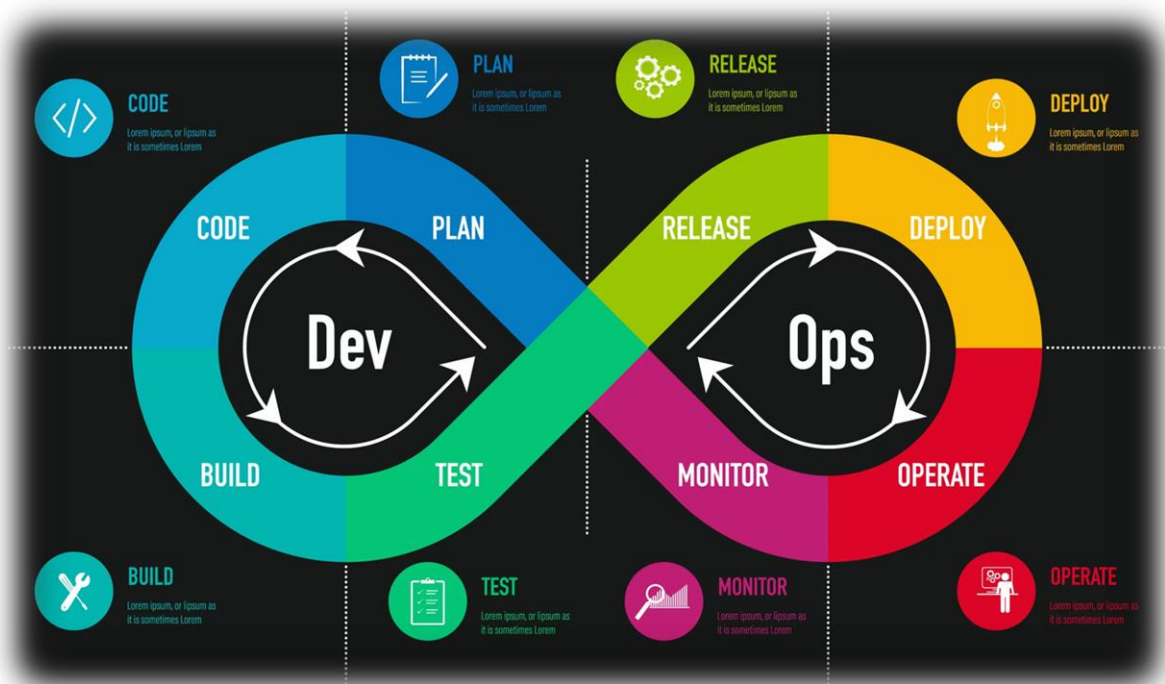


Rapport de Projet : IoT Monitor – Micro-Services DevOps



Étudiante : Celia MERABET

URL du dépôt GitHub : <https://github.com/celia-merabet/iot-monitoring-devops>

Table des matières :

Rapport de Projet : IoT Monitor – Micro-Services DevOps	1
1. Présentation du Projet	2
2. Architecture Technique	3
2.1 Schéma PlantUML	3
2.2 Description des services	3
3. Guide d'installation	4
3.1 Prérequis	4
3.2 Installation	4
4. Méthodologie & Organisation	5
5. Utilisation de l'IA	5
6. Difficultés rencontrées & Solutions	5
7. Résultats attendus	5
8. Conclusion	6
9. Annexes	6
10. Ressources & Liens Utiles	6
Documentation officielle	6
Tutoriels et guides utiles	6
Forums et communautés	6

1. Présentation du Projet

Le projet **IoT Monitor** est une plateforme de monitoring de capteurs IoT.
L'objectif est de mettre en place une **infrastructure micro-services conteneurisée avec Docker**, sécurisée et accessible via un reverse proxy et un tunnel Cloudflare.

Fonctionnalités principales :

- Dashboard web pour visualiser les données de capteurs IoT
- API backend pour la réception des données simulées
- Base de données PostgreSQL avec persistance des données
- Interface Adminer pour la gestion de la base de données
- Accès HTTPS public via Cloudflare Tunnel (optionnel)

L'accent est mis sur l'infrastructure et la robustesse des services, pas sur des fonctionnalités avancées IoT.

2. Architecture Technique

2.1 Schéma PlantUML

```
@startuml
!theme sunlust
title Architecture DevOps du Projet IoT Monitor

node "Docker Host" {
    component "Cloudflared" as Tunnel #Orange
    component "Caddy" as Proxy #Yellow

    package "Internal Network" {
        component "Frontend Nginx" as Frontend #LightBlue
        component "API Node.js" as API #LightGreen
        component "PostgreSQL DB" as DB #Pink
        component "Adminer" as Admin #LightGray
    }
}

cloud "Internet" {
    actor User
}

User --> Tunnel : HTTPS
Tunnel --> Proxy : HTTP
Proxy --> Frontend : /
Proxy --> API : /api/*
Proxy --> Admin : /admin
Frontend --> API : Requêtes API
API --> DB : Lecture / Écriture
Admin --> DB : Administration
@enduml
```

2.2 Description des services

Service	Image Docker	Rôle	Port Interne / Exposé
Caddy	caddy:alpine	Reverse Proxy & Routage	80 (host)
Frontend	nginx:alpine	Dashboard Web IoT	80 (container)
API	node:18-alpine	Backend Node.js	3000:3000
DB	postgres:15	Base de données persistante	5432 (container)
Adminer	adminer	Interface BDD	8080:8080
Cloudflared	cloudflare/cloudflared	Tunnel HTTPS public	N/A

Tous les conteneurs sont configurés avec `restart: unless-stopped` et `healthchecks` pour assurer la robustesse.

3. Guide d'installation

3.1 Prérequis

- Docker Desktop (Mac, Windows ou Linux)
- Docker Compose 3.9+
- Compte Cloudflare pour le tunnel (optionnel pour accès public)

3.2 Installation

1. Cloner le dépôt :

```
git clone https://github.com/celia-merabet/iot-monitoring-devops.git
cd iot-monitoring-devops
```

2. Lancer la stack (local) :

```
docker compose up -d --build
```

3. Vérifier les conteneurs :

```
docker ps
```

4. Accéder aux services :

Service	URL
Frontend	http://localhost
API	http://localhost:3000/api/status
Adminer	http://localhost:8080

5. Si le tunnel Cloudflare est configuré, obtenir l'URL publique :

```
docker compose logs -f cloudflared
```

4. Méthodologie & Organisation

- Projet réalisé en autonomie.
- Approche progressive : création du Dockerfile pour l'API → conteneurisation frontend → configuration BDD → reverse proxy Caddy → Cloudflare Tunnel.
- Tests réguliers via `curl` et navigateur pour vérifier la connectivité interne et externe.

5. Utilisation de l'IA

Outils utilisés : capilote

Usage :

- Conseils ponctuels de syntaxe et configuration Docker
- Débogage des conteneurs (API Node.js et Adminer)
- Suggestions pour l'architecture PlantUML

Apprentissage : L'IA a été utilisée comme assistant ; toutes les configurations ont été testées et adaptées manuellement par moi.

6. Difficultés rencontrées & Solutions

Problème	Solution
API Node.js non accessible depuis Mac	Ajout d'un mapping de port 3000:3000
Adminer inaccessible	Ajout d'un mapping de port 8080:8080 et choix du système PostgreSQL
Cloudflared restart en boucle	Token non configuré → tunnel commenté temporairement
Perte de données après redémarrage	Création d'un volume Docker pour la BDD

7. Résultats attendus

- Frontend accessible sur <http://localhost>
- API accessible sur <http://localhost:3000/api/status>
- Base PostgreSQL persistante via volume
- Adminer accessible sur <http://localhost:8080>
- Architecture robuste avec restart policies et isolation réseau

8. Conclusion

Ce projet montre la mise en place d'une **stack micro-services complète** avec Docker, intégrant :

- Frontend et backend séparés
- Base de données persistante
- Interface d'administration
- Reverse proxy Caddy

9. Annexes

- **Schéma PlantUML** : `architecture.puml`
- **Frontend HTML minimal** : `frontend/index.html`
- **API Node.js** : `api/index.js`
- **Docker Compose** : `docker-compose.yml`
- **Caddy configuration** : `Caddyfile`

10. Ressources & Liens Utiles

Documentation officielle

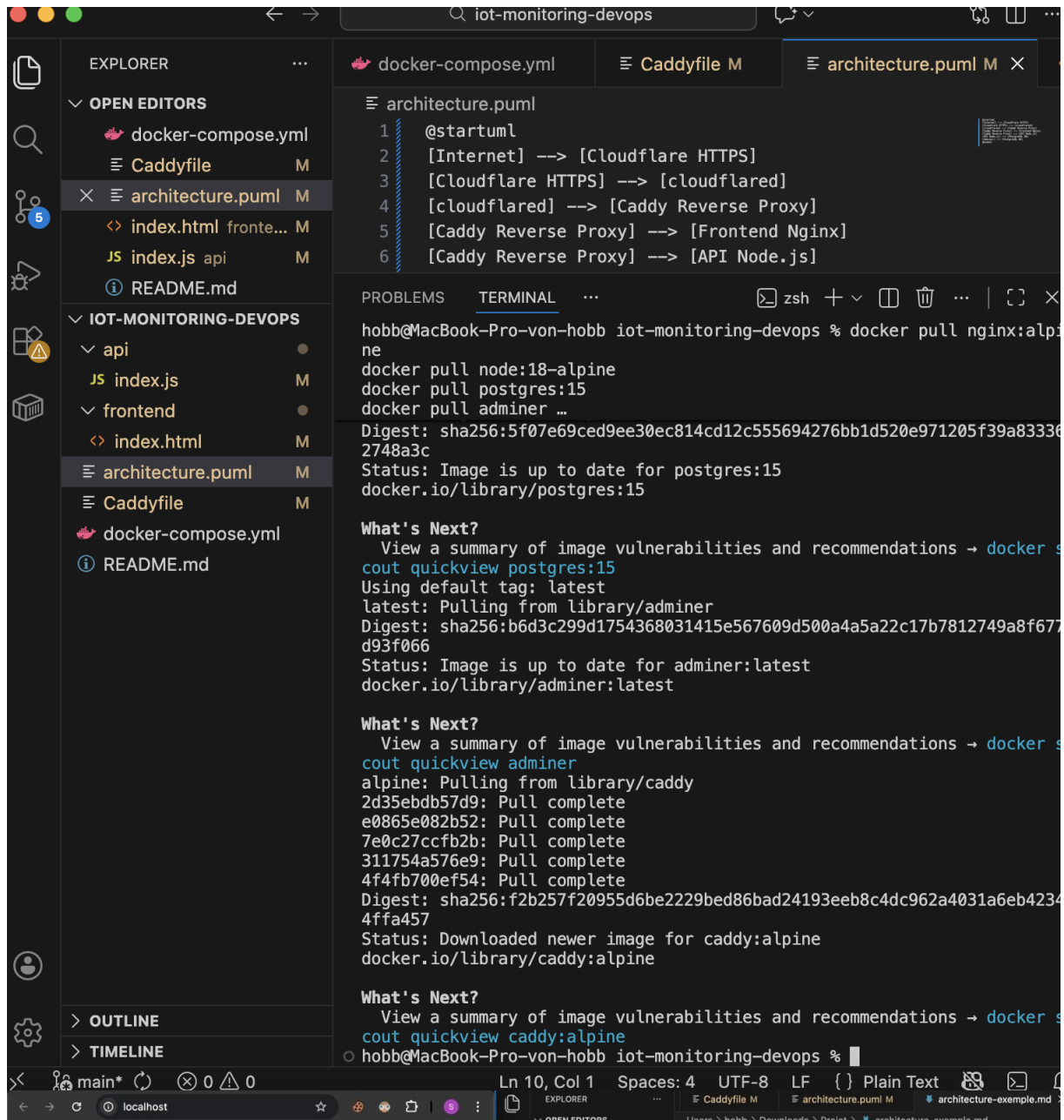
- **Docker** : <https://docs.docker.com/>
- **Docker Compose** : <https://docs.docker.com/compose/>
- **Node.js** : <https://nodejs.org/en/docs/>
- **PostgreSQL** : <https://www.postgresql.org/docs/>
- **Adminer** : <https://www.adminer.org/>
- **Caddy (reverse proxy)** : <https://caddyserver.com/docs/>
- **Cloudflare Tunnel (cloudflared)** : <https://developers.cloudflare.com/cloudflare-one/connections/connect-apps/>

Tutoriels et guides utiles

- **Docker + Node.js + PostgreSQL** : <https://node-postgres.com/>
- **Créer un Dockerfile pour Node.js** : <https://nodejs.org/en/docs/guides/nodejs-docker-webapp/>
- **PlantUML pour diagrammes d'architecture** : <https://plantuml.com/fr/>
- **Exemples Docker Compose multi-services** : <https://docs.docker.com/compose/gettingstarted/>

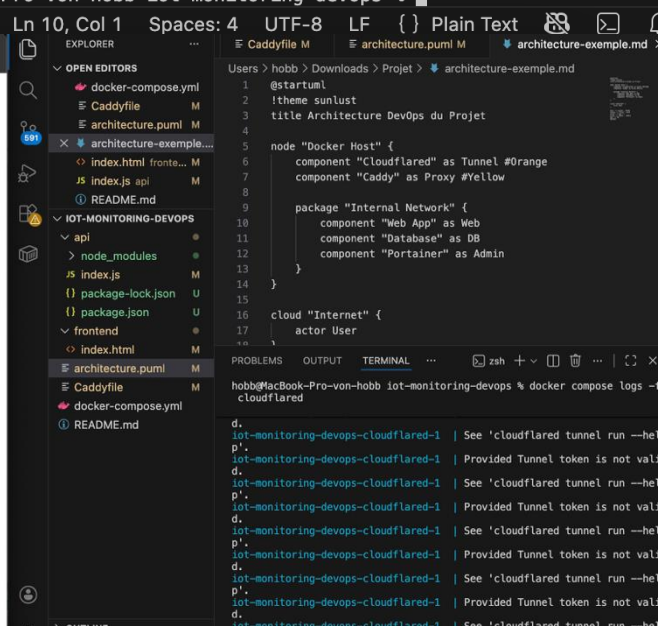
Forums et communautés

- **Stack Overflow** : <https://stackoverflow.com/>



IoT Monitor – DevOps Project

Bienvenue sur le dashboard de supervision des capteurs IoT.



localhost:3000/api/status

```
{
  "status": "OK",
  "message": "API IoT Monitor en fonctionnement"
}
```

localhost:8080

Adminer 5.4.1

Langue: Français

Système: MySQL / MariaDB

Serveur: db

Utilisateur:

Mot de passe:

Base de données:

Authentication ☐ Authentification permanente

EXPLORER

OPEN EDITORS

- docker-compose.yml
- Caddyfile
- architecture.puml
- architecture-exemple...
- index.html frontend
- index.js api
- README.md
- IOT-MONITORING-DEVOPS
- api
- node_modules
- index.js
- package-lock.json
- package.json
- frontend
- index.html
- architecture.puml
- Caddyfile
- docker-compose.yml
- README.md

docker-compose.yml

```
services:
  api:
    environment:
      - DB_HOST=db
      - DB_USER=postgres
      - DB_PASSWORD=postgres
      - DB_NAME=iot
    restart: unless-stopped
    ports:
      - "3000:3000"

  db:
    image: postgres:15
    environment:
      POSTGRES_DB: iot
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
```

PROBLEMS OUTPUT TERMINAL

hobbMacBook-Pro-von-hobb iot-monitoring-devops % docker compose down

```
docker compose up -d
```

Container	Status	Size
Container iot-monitoring-devops-api-1	Removed	10.3s
Container iot-monitoring-devops-cloudflared-1	Removed	0.0s
Container iot-monitoring-devops-caddy-1	Removed	0.7s
Container iot-monitoring-devops-db-1	Removed	0.6s
Container iot-monitoring-devops-adminer-1	Removed	0.4s
Network iot-monitoring-devops_default	Removed	0.2s
[+] Building 0.0s (0/0) docker:desktop-linux		
[+] Running 7/7		
Network iot-monitoring-devops_default	Created	0.1s
Container iot-monitoring-devops-db-1	Started	0.2s
Container iot-monitoring-devops-adminer-1	Started	0.2s
Container iot-monitoring-devops-caddy-1	Started	0.2s
Container iot-monitoring-devops-cloudflared-1	Started	0.2s
Container iot-monitoring-devops-frontend-1	Started	0.2s
Container iot-monitoring-devops-api-1	Started	0.2s

hobbMacBook-Pro-von-hobb iot-monitoring-devops % curl http://localhost:3000/api/status

docker-compose.yml

```
services:
  db:
    volumes:
      - db_data:/var/lib/postgresql/data
    restart: unless-stopped

  adminer:
    image: adminer
    restart: unless-stopped
    ports:
      - "8080:8080"

  caddy:
    image: caddy:alpine
    ports:
      - "80:80"
```

PROBLEMS OUTPUT TERMINAL

hobbMacBook-Pro-von-hobb iot-monitoring-devops % docker compose down

```
docker compose up -d
```

Container	Status	Size
Container iot-monitoring-devops-api-1	Removed	10.2s
Container iot-monitoring-devops-frontend-1	Removed	0.6s
Container iot-monitoring-devops-db-1	Removed	0.7s
Container iot-monitoring-devops-cloudflared-1	Removed	0.0s
Container iot-monitoring-devops-adminer-1	Removed	0.7s
Container iot-monitoring-devops-caddy-1	Removed	0.5s
Network iot-monitoring-devops_default	Removed	0.1s
[+] Building 0.0s (0/0) docker:desktop-linux		
[+] Running 7/7		
Network iot-monitoring-devops_default	Created	0.1s
Container iot-monitoring-devops-db-1	Started	0.1s
Container iot-monitoring-devops-cloudflared-1	Started	0.1s
Container iot-monitoring-devops-api-1	Started	0.1s
Container iot-monitoring-devops-adminer-1	Started	0.1s
Container iot-monitoring-devops-frontend-1	Started	0.1s

The image is a composite of three screenshots from a development environment.

Top Left: PostgreSQL Schema Page
 The browser shows the PostgreSQL interface at `localhost:8080/?pgsql=db&us...`. The page title is "Schéma: public". It lists options: "Modifier le schéma", "Schéma de la base de données", "Routines", "Séquences", "Types utilisateur". Under "Tables et vues", it says "Aucune table." and provides links "Créer une table" and "Créer une vue". Under "Routines", it provides links "Créer une procédure" and "Créer une fonction". Under "Séquences", it provides a link "Créer une séquence". Under "Types utilisateur", it provides a link "Créer un type".

Bottom Left: PostgreSQL Create Type Page
 The browser shows the "Créer un type" page. It has a form with "Nom:" followed by a text input field containing "AS". Below the input is an "Enregistrer" button.

Right: Code Editor and Terminal
 The code editor shows a project structure with files like `docker-compose.yml`, `Caddyfile`, `architecture.puml`, `index.html`, `index.js`, `package-lock.json`, `package.json`, `frontent`, `index.html`, `architecture.puml`, `Caddyfile`, and `docker-compose.yml`. The `docker-compose.yml` file is open, showing a configuration for a service named `api` using `node:18-alpine` image. The terminal window shows the output of the command `docker compose down` and `docker compose up -d`, listing the status of various containers (e.g., `iot-monitoring-devops-api-1`, `iot-monitoring-devops-frontend-1`, etc.) and their states (Removed, Created, Started).