# Groupe G2

## Data Engineering & Ingestion Temps Réel

Mise en place de la base PostgreSQL Dockerisée
et du pipeline d'ingestion


Matricules : 23623 / 23639 / 23647

23 janvier 2026

# Table des matières

# 1 Objectif de la partie G2

L'objectif de cette partie est de mettre en place une base de données PostgreSQL fiable et reproductible, capable de recevoir un flux de données simulant un temps réel à partir du dataset *Individual Household Electric Power Consumption* (UCI).

**Information**

Cette base constitue le socle central du projet : elle alimente les modules d'analyse (G3), de détection d'anomalies (G4) et de visualisation (G5).

# 2 Pré-requis pour les autres groupes

Avant d'utiliser la base de données fournie par G2, chaque membre ou groupe doit disposer des éléments suivants :

## 2.1 Logiciels requis

— **Git** : pour cloner et mettre à jour le dépôt
— **Docker et Docker Compose** : pour déployer la base de données
— **Python 3.9+** : si exécution des scripts Python en local

# 3 Récupération du projet

Dans un terminal, cloner ou mettre à jour le dépôt GitHub :

```
1 git pull origin main
```

S'assurer d'avoir la dernière version du projet, notamment :

— `docker-compose.yaml`
— le dossier `sql/`
— les scripts Python (`producer.py`, `db.py`)

# 4 Création et lancement de la base de données avec Docker

## 4.1 Principe général

La base PostgreSQL est déployée dans un conteneur Docker, ce qui garantit :

— un environnement identique pour tous
— une installation sans configuration manuelle
— une reproductibilité totale

## 4.2 Lancement de la base

Depuis le dossier racine du projet :

```
docker compose up -d
```

Cette commande :

— télécharge l'image PostgreSQL si nécessaire

— crée et démarre le conteneur

— initialise automatiquement la base de données

— exécute le script SQL d'initialisation

# 5 Explication du fichier docker-compose.yaml

Le fichier `docker-compose.yaml` décrit la configuration du service PostgreSQL.

## 5.1 Service db

```
services:
  db:
    image: postgres:15-alpine
```

⇒ Utilisation de l'image officielle PostgreSQL (version 15), légère et stable.

```
    container_name: sdid_postgres
```

⇒ Donne un nom explicite au conteneur pour faciliter les commandes (`docker exec`).

```
    environment:
      POSTGRES_USER: sdid_user
      POSTGRES_PASSWORD: sdid_password
      POSTGRES_DB: sdid_db
```

⇒ Paramètres utilisés uniquement lors du premier démarrage pour :

— créer l'utilisateur

— définir le mot de passe

— créer la base de données

```
    ports:
      - "5432:5432"
```

⇒ Expose PostgreSQL vers la machine locale, permettant :

— la connexion via scripts Python

— l'utilisation de clients SQL externes

```
    volumes:
      - ./sql/init_db.sql:/docker-entrypoint-initdb.d/init_db.sql
```

⇒ Exécute automatiquement le script SQL d'initialisation (création des tables).

```
- postgres_data:/var/lib/postgresql/data
```

⇒ Volume Docker assurant la persistance des données
⇒ Les données ne sont pas perdues lors de l'arrêt du conteneur.

# 6 Schéma de la base de données

La table principale est `power_consumption` et contient notamment :

— un timestamp (`ts`)
— les mesures électriques
— des champs réservés à la détection d'anomalies (`is_anomaly`, `anomaly_score`)

**Information**

Ce schéma est volontairement extensible pour les besoins des groupes G3 et G4.

# 7 Vérification de la connexion à la base

Avant de lancer l'ingestion, il est recommandé de tester la connexion.

## 7.1 Script de test (exemple)

```
python test.py
```

Ce script vérifie :
— l'accès à PostgreSQL
— la validité des paramètres de connexion
— l'existence de la table

# 8 Ingestion des données — producer.py

## 8.1 Rôle du script

Le script `producer.py` est responsable de :
— lire le fichier source `.txt`
— nettoyer les valeurs manquantes (`?`)
— convertir les dates et heures en timestamps
— insérer les données ligne par ligne
— simuler un flux temps réel grâce à un délai fixe

## 8.2 Fonctionnement général

1. Lecture du fichier `household_power_consumption.txt`
2. Nettoyage et conversion des données
3. Insertion dans la base PostgreSQL
4. Pause de 2 secondes entre chaque insertion

**Information**

Cette approche permet de simuler un flux continu, proche d'un cas réel industriel.

## 8.3 Lancement du producer

```
python producer.py
```

Une fois lancé, le script alimente la base automatiquement.

# 9 Ordre recommandé d'exécution (IMPORTANT)

**Important**

Pour une utilisation correcte par les autres groupes :

1. `git pull` (dernière version)
2. Installation de Docker
3. Lancer la base :

```
docker compose up -d
```

4. Tester la connexion :

```
python test.py
```

5. Lancer l'ingestion :

```
python producer.py
```

# 10 Conclusion

La partie G2 fournit une infrastructure de données robuste, reproductible et prête à l'emploi. Grâce à Docker, chaque groupe peut déployer localement la base sans configuration complexe et se concentrer sur son cœur de métier (analyse, détection, visualisation).

Cette approche garantit :

— la cohérence des données
— la compatibilité entre modules
— une intégration fluide du pipeline complet