InfluxDB avec Node.js
BUT3 - S5 INFO

Base de Données Time Series

Stocker et analyser des données temporelles

Joseph Azar

Maître de Conférences - Université Marie Louis Pasteur

Chercheur FEMTO-ST

🗺️ Le Paysage des Bases de Données

Avant InfluxDB, rappelons ce qu'on connait déjà...

Type	Exemple	Modèle	Cas d'usage
SQL (Relationnel)	PostgreSQL, MySQL	Tables + Relations	Données structurées, transactions
Document (NoSQL)	MongoDB	Collections + Documents JSON	Données semi-structurées, flexibles
Clé-Valeur	Redis	Key → Value en RAM	Cache, sessions, données temporaires
Time Series 📊	**InfluxDB**	Mesures + Temps	**Données temporelles, métriques, logs**

⏰ C'est Quoi des Données Time Series?

📈 Définition

Time Series = Série de valeurs indexées par le temps

Chaque point de donnée a :

- ⏰ Un **timestamp** (horodatage)
- 📊 Une ou plusieurs **valeurs**
- 🏷️ Des **métadonnées** optionnelles

🌍 Exemples Concrets

- 🌡️ **IoT** : Température toutes les 5min
- 📈 **Finance** : Prix actions chaque seconde
- 🏥 **Santé** : Battements cardiaques
- 🚗 **GPS** : Position véhicule en continu
- 💻 **Monitoring** : CPU/RAM serveur
- 👤 **Analytics** : Visites site web
- 🔐 **Sécurité** : Logs de connexion

🤔 Pourquoi Pas MongoDB ou PostgreSQL?

❌ Problèmes avec les bases traditionnelles pour Time Series

1. **Volume énorme** : Des millions de points par jour
2. **Requêtes lentes** : Agrégations temporelles complexes
3. **Espace disque** : Pas de compression optimisée
4. **Pas de TTL automatique** : Données anciennes s'accumulent

🐌 MongoDB/PostgreSQL

```
-- Requête "moyenne température par heure"
SELECT
  date_trunc('hour', timestamp) as hour,
  AVG(temperature) as avg_temp
FROM measurements
```

⚡ InfluxDB

```
-- Même requête en InfluxQL
SELECT MEAN(temperature)
FROM measurements
WHERE time > now() - 24h
GROUP BY time(1h)
```

🧩 Concepts Clés d'InfluxDB

SQL	MongoDB	Redis	InfluxDB
Database	Database	-	**Database**
Table	Collection	-	**Measurement**
Row	Document	Key-Value	**Point**
Column	Field	-	**Field / Tag**
Index	Index	-	**Tag (auto-indexed)**

📊 Anatomie d'un Point InfluxDB

```
measurement,tag1=value1,tag2=value2 field1=value1,field2=value2 timestamp

Exemple concret:
login_times,userId=123,googleId=abc displayName="Alice" 1704931200000000000
    |            |              |              |                       |
    |            |              |              |                       └─ Timestamp
    |            |              |              └─ Fields (valeurs mesurées)
    |            |              └─ Tag (métadonnée indexée)
```

🏷️ Tags vs Fields - La Différence Cruciale

🏷️ Tags

Métadonnées indexées

- ✅ **Indexés automatiquement**
- ✅ Parfait pour filtrer et grouper
- ✅ Stockés en string uniquement
- ⚠️ Cardinalité limitée (pas de valeurs uniques infinies)

Exemples :

- userId, region,

📊 Fields

Valeurs mesurées

- 📈 Les valeurs qu'on veut analyser
- 🔢 Peuvent être : string, float, int, bool
- ❌ Pas indexés (plus lents à filtrer)
- ✅ Cardinalité illimitée

Exemples :

- temperature, cpu_usage, price
- displayName,

🐋 Installation avec Docker

```
# Lancer InfluxDB 1.8 avec Docker
docker run -d --name influxdb-server \
  -p 8086:8086 \
  -v influxdb:/var/lib/influxdb \
  influxdb:1.8
```

🎯 Qu'est-ce que ça fait?

- 📥 `-d` : Lance en arrière-plan
- 🔌 `-p 8086:8086` : Port HTTP API
- 💾 `-v influxdb:/var/lib/influxdb` : Persistance des données
- 📦 `influxdb:1.8` : Version stable avec InfluxQL

```
# Créer une base de données
curl -i -XPOST http://localhost:8086/query \
  --data-urlencode "q=CREATE DATABASE newsql_db"

# Se connecter au CLI InfluxDB
docker exec -it influxdb-server influx
```

Commandes InfluxQL de Base

```
-- Afficher les bases de données
SHOW DATABASES

-- Utiliser une base
USE newsql_db

-- Afficher les "tables" (measurements)
SHOW MEASUREMENTS

-- Afficher les séries (combinaisons tag)
SHOW SERIES

-- Afficher les tags d'une measurement
SHOW TAG KEYS FROM login_times

-- Afficher les valeurs d'un tag
SHOW TAG VALUES FROM login_times WITH KEY = "userId"

-- Afficher les fields
SHOW FIELD KEYS FROM login_times
```

Requêtes SELECT

```
-- Sélectionner tout (avec limite)
SELECT * FROM login_times LIMIT 10

-- Sélectionner avec filtre temporel
SELECT * FROM login_times WHERE time > now() - 1h

-- Compter les points
SELECT COUNT(displayName) FROM login_times

-- Grouper par temps (buckets de 5 minutes)
SELECT COUNT(displayName)
FROM login_times
WHERE time > now() - 1h
GROUP BY time(5m)
```

🔧 Intégration Node.js - Installation

```
# Installer le client InfluxDB
npm install influx

# Installer aussi pour les fuseaux horaires
npm install moment moment-timezone
```

📦 **Packages nécessaires :**

- `influx` : Client officiel InfluxDB pour Node.js
- `moment` et `moment-timezone` : Gestion des fuseaux horaires

🔧 Configuration du Client InfluxDB

```
const Influx = require('influx');

// Créer le client InfluxDB
const client = new Influx.InfluxDB({
    database: 'newsql_db',
    host: 'localhost',
    port: 8086,
    schema: [
        {
            measurement: 'login_times',
            fields: {
                displayName: Influx.FieldType.STRING
            },
            tags: [
                'userId',
                'googleId'
            ]
        }
    ]
});
```

✍️ Écrire des Données - Setup Client

Notre cas d'usage : Logger les connexions utilisateur

```
// middlewares/loginLogger.js
const Influx = require("influx");

module.exports = async (req, res, next) => {
    const client = new Influx.InfluxDB({
        database: "newsql_db",
        host: "localhost",
        port: 8086,
        schema: [
            {
                measurement: 'login_times',
                fields: {
                    displayName: Influx.FieldType.STRING,
                },
                tags: [
                    'userId',
                    'googleId'
                ]
            }
```

✍️ Écrire des Données - Écriture

```
    console.log('📊 Logging user login to InfluxDB');

    try {
        // Écrire le point de données
        await client.writePoints([
            {
                measurement: 'login_times',
                tags: {
                    userId: req.user._id,
                    googleId: req.user.googleId
                },
                fields: {
                    displayName: req.user.displayName
                },
                timestamp: new Date(), // Timestamp actuel
            },
        ]);
        console.log('✅ Data written to InfluxDB');
    } catch (error) {
        console.error('❌ Error writing to InfluxDB', error);
    }
```

📖 Lire des Données - Configuration

```
// routes/loginLogRoutes.js
const Influx = require("influx");
const moment = require('moment-timezone');

const client = new Influx.InfluxDB({
    database: "newsql_db",
    host: "localhost",
    port: 8086,
    schema: [
        {
            measurement: 'login_times',
            fields: {
                displayName: Influx.FieldType.STRING,
            },
            tags: ['userId', 'googleId']
        }
    ]
});
```

📖 Lire des Données - Requête Simple

```
module.exports = app => {
    // GET tous les logins (ordre décroissant)
    app.get("/api/logins", (req, res) => {
        client.query(`
            SELECT *
            FROM login_times
            ORDER BY time DESC
            tz('Europe/Paris')
        `)
        .then(results => {
            console.log('📊 Retrieved', results.length, 'login records
            res.status(200).json(results);
        })
        .catch(error => {
            console.error('❌ Error querying data:', error);
            res.status(500).send("Internal error");
        });
    });
};
```

Agrégation Temporelle

Compter les logins par intervalles de temps (fenêtres temporelles)

```
// Compter les logins par fenêtre de temps (ex: toutes les 5 minutes)
app.get("/api/logins/minutes", (req, res) => {
    // Paramètre optionnel : taille de la fenêtre en minutes
    let mins = req.query.mins || 5;
    let timeWindow = mins + "m";

    client.query(`
        SELECT COUNT(displayName)
        FROM login_times
        WHERE time > now() - 60m
        GROUP BY time(${timeWindow})
        ORDER BY time DESC
        tz('Europe/Paris')
    `)
    .then(results => {
        // Formater les timestamps pour l'affichage
        const adjustedResults = results.map(result => {
            const date = moment(result.time).tz('Europe/Paris');
            const formattedDate = date.format('HH:mm');
```

Grouper par Tags (Par Utilisateur)

```
// Compter les logins par utilisateur sur une période
app.get('/api/logins/count', async (req, res) => {
    const { start, end } = req.query;

    if (!start || !end) {
        return res.status(400).send('Both start and end datetime are required');
    }

    try {
        const results = await client.query(`
            SELECT COUNT(displayName) as count, last(displayName) as displayName
            FROM login_times
            WHERE time >= '${start}' AND time <= '${end}'
            GROUP BY userId
        `);

        const adjustedResults = results.map(result => {
            const time = moment(result.time).tz('Europe/Paris').format();
            return {
                time,
                displayName: result.displayName,
                count: result.count
            };
```

📈 Agrégation Avancée - Requête

```
// Calculer la moyenne de logins par heure pour chaque utilisateur
app.get('/api/logins/average', async (req, res) => {
    const { start, end } = req.query;

    if (!start || !end) {
        return res.status(400).send('Both start and end datetime are require
    }

    try {
        // Grouper par userId ET par heure
        const results = await client.query(`
            SELECT COUNT(displayName) as count,
                   last(displayName) as displayName
            FROM login_times
            WHERE time >= '${start}' AND time <= '${end}'
            GROUP BY time(1h), userId
        `);
```

📈 Agrégation Avancée - Traitement

```
        // Map: Créer un mapping userId -> array de counts
        const userLoginCounts = results.reduce((acc, result) => {
            const userId = result.userId;
            if (!acc[userId]) {
                acc[userId] = [];
            }
            acc[userId].push(result.count);
            return acc;
        }, {});

        // Reduce: Calculer la moyenne pour chaque user
        const averageLoginsPerUser = Object.keys(userLoginCounts).map(userI
            const counts = userLoginCounts[userId];
            const total = counts.reduce((sum, count) => sum + count, 0);
            const average = total / counts.length;

            return {
                userId,
                displayName: results.find(result => result.userId === userI
                averageLogins: average.toFixed(2)
            };
```

📦 Écriture en Masse - Préparation

Pour tester, créons des fausses données de connexion

```
// Endpoint pour générer des données de test
app.post("/api/logins/fake", async (req, res) => {
    // Profils utilisateurs manuels
    const users = [
        { displayName: "Alice", googleId: "alice123", _id: "1" },
        { displayName: "Bob", googleId: "bob123", _id: "2" },
        { displayName: "Charlie", googleId: "charlie123", _id: "3" }
    ];

    const points = [];

    // Créer 100 points de données aléatoires
    for (let i = 0; i < 100; i++) {
        const user = users[Math.floor(Math.random() * users.length)];
        const time = new Date(Date.now() - 3600000 * Math.random());

        points.push({
            measurement: 'login_times',
            tags: { userId: user._id, googleId: user.googleId },
```

📦 Écriture en Masse - Insertion

```
    try {
        // Écrire tous les points en une seule opération (efficace!)
        await client.writePoints(points);
        console.log('✅ Inserted', points.length, 'fake login records')
        res.status(200).send(
          `Fake login data inserted successfully! (${points.length} poi
        );
    } catch (error) {
        console.error('❌ Error writing to InfluxDB', error);
        res.status(500).send('Failed to insert fake login data.');
    }
});

// Test: curl -X POST http://localhost:5000/api/logins/fake
```

⚡ **Performance :** writePoints() accepte un array - toujours écrire en batch quand possible!

🗑️ Retention Policies - Auto-suppression

Gérer automatiquement la durée de vie des données

> 🎯 **Pourquoi?**
>
> Les données time series s'accumulent rapidement. Les retention policies suppriment automatiquement les vieilles données.

```
-- Créer une retention policy de 12 heures
CREATE RETENTION POLICY "twelve_hours"
ON "newsql_db"
DURATION 12h
REPLICATION 1
DEFAULT

-- Voir les retention policies
SHOW RETENTION POLICIES ON "newsql_db"

-- Modifier une retention policy
ALTER RETENTION POLICY "twelve_hours"
```

🗑️ Retention Policies - Paramètres

Paramètre	Description	Exemple
DURATION	Combien de temps garder les données	12h, 7d, 52w, INF (infini)
REPLICATION	Nombre de copies (cluster)	1 (single node)
SHARD DURATION	Taille des shards (optimisation)	1h (pour DURATION < 7d)
DEFAULT	Policy par défaut de la DB	DEFAULT ou non

🎯 Quand Utiliser InfluxDB?

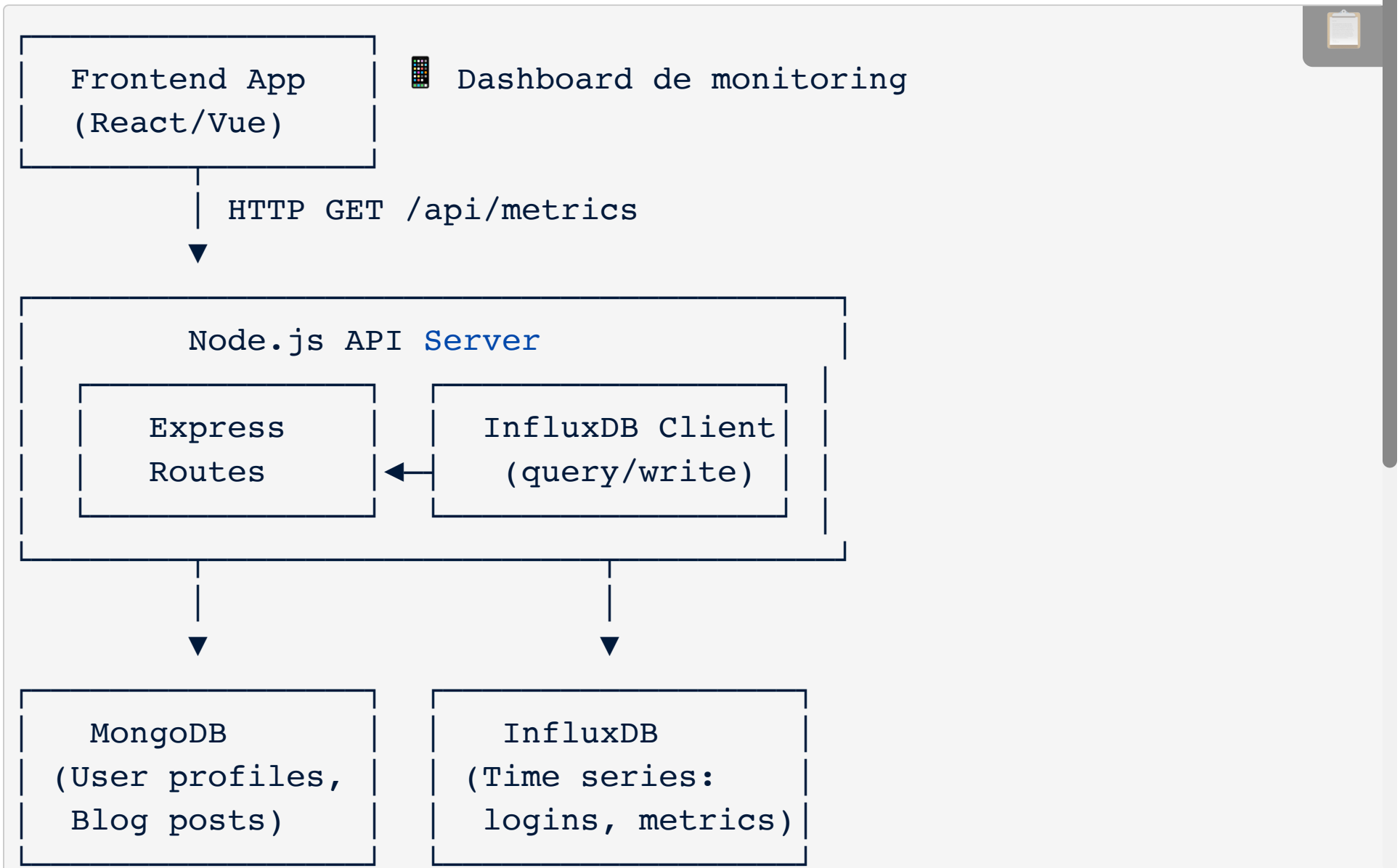
✅ Cas d'Usage Parfaits

- 📊 **Monitoring** : CPU, RAM, réseau
- 🏥 **IoT/Sensors** : Température, humidité
- 📈 **Analytics** : Page views, événements
- 💹 **Finance** : Prix, volumes trading
- 🚗 **Logs** : Application logs, accès
- 🔐 **Sécurité** : Authentification, firewall
- 🌐 **Performance** : Latence, response time

❌ Cas où NE PAS Utiliser

- ❌ **Relations complexes** → PostgreSQL
- ❌ **Documents flexibles** → MongoDB
- ❌ **Cache rapide** → Redis
- ❌ **Transactions ACID** → SQL
- ❌ **Full-text search** → Elasticsearch
- ❌ **Graphes** → Neo4j

Architecture Typique avec InfluxDB



📊 Comparaison Rapide

Fonctionnalité	PostgreSQL	MongoDB	Redis	InfluxDB
Modèle	Relationnel	Document	Key-Value	Time Series
Stockage	Disque	Disque	RAM	Disque (compressé)
Requêtes temps	Lent	Lent	N/A	⚡ Très rapide
Agrégations	Complexe	Framework aggr.	Limité	⚡ Optimisé
TTL auto	❌ Non	✅ Oui	✅ Oui	✅ Retention policies
Compression	Limitée	Limitée	N/A	⚡ Excellente
Cas d'usage	Données métier	Documents JSON	Cache	Métriques/Logs

✅ Bonnes Pratiques InfluxDB

🏷️ Tags

✅ **DO**

- Utiliser pour **filtrer et grouper**
- Cardinalité limitée (< 100k valeurs uniques)
- Exemples : userId, region, status

❌ **DON'T**

- Pas de valeurs uniques infinies (UUID, email)
- Pas de données mesurées

📊 Fields

✅ **DO**

- Valeurs qu'on veut **agréger**
- Cardinalité illimitée OK
- Exemples : temperature, count, displayName

❌ **DON'T**

- Pas pour filtrer (lent, non-indexé)
- Au moins 1 field requis par point

⚠️ Pièges Courants

🌍 Exemple Concret : Notre Système de Login

📊 Ce qu'on track avec InfluxDB

```
Measurement: login_times
├── Tags (indexés, pour filtrer)
│   ├── userId      : Identifiant MongoDB unique
│   └── googleId    : Identifiant Google OAuth
│
├── Fields (valeurs mesurées)
│   └── displayName : Nom affiché de l'utilisateur
│
└── Timestamp       : Moment exact de la connexion
```

📝 Données Écrites

```
// Dans le middleware loginLogger
await client.writePoints([{
```

📊 Requêtes Possibles

```
-- Logins par heure
SELECT COUNT(*)
```

📈 Visualisation des Données

🎨 Outils de Visualisation

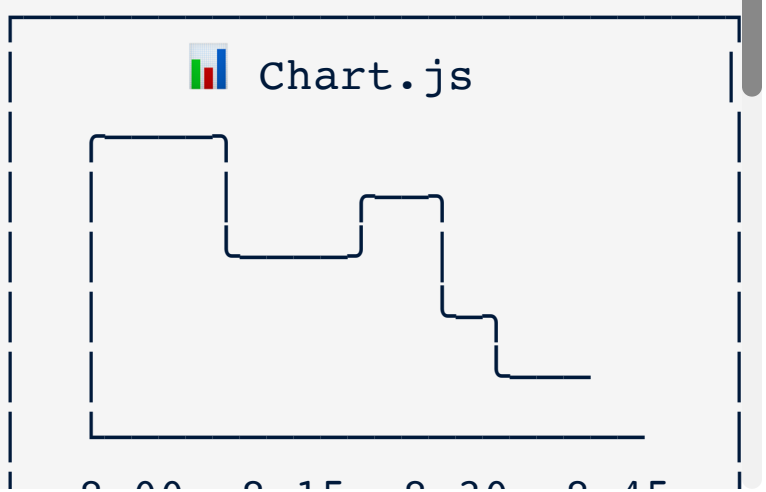
- 📊 **Grafana** : Le plus populaire (dashboards puissants)
- 🖥️ **Chronograf** : Interface officielle InfluxDB
- 📈 **Custom Dashboard** : Chart.js, D3.js avec API

📊 Exemple avec Chart.js

```
// Frontend fetch
fetch('/api/logins/minutes?mins=5')
  .then(res => res.json())
  .then(data => {
    const ctx = document.getElementById('chart');
    new Chart(ctx, {
      type: 'line',
      data: {
        labels: data.map(d => d.time),
        datasets: [{
          label: 'Logins per 5min',
          data: data.map(d => d.count),
```

📈 Résultat Visuel

Logins par 5 minutes



🔧 Fonctions InfluxQL Utiles

Fonction	Description	Exemple
COUNT()	Compter les points	`SELECT COUNT(displayName) FROM login_times`
MEAN()	Moyenne	`SELECT MEAN(temperature) FROM sensors`
SUM()	Somme	`SELECT SUM(visitors) FROM analytics`
MIN() / MAX()	Min/Max	`SELECT MIN(price), MAX(price) FROM stocks`
FIRST() / LAST()	Premier/Dernier	`SELECT FIRST(value), LAST(value) FROM metrics`
MEDIAN()	Médiane	`SELECT MEDIAN(response_time) FROM requests`

🔧 Fonctions InfluxQL - Exemple Combiné

```
-- Exemple combiné : Stats complètes de température
SELECT
  COUNT(temperature) as total_readings,
  MEAN(temperature) as avg_temp,
  MIN(temperature) as min_temp,
  MAX(temperature) as max_temp,
  STDDEV(temperature) as std_deviation
FROM room_sensors
WHERE time > now() - 24h
GROUP BY room
```

> 📊 **Résultat :** Statistiques complètes par pièce pour les dernières 24h

🚀 Pour Aller Plus Loin

📚 Concepts Avancés

- 📊 **Continuous Queries** : Auto-agrégation
- 🔄 **Downsampling** : Réduire la résolution
- 🌐 **InfluxDB 2.x** : Flux language
- 🔐 **Authentication** : Users & privileges
- 📈 **Kapacitor** : Alerting & processing
- ☁️ **InfluxDB Cloud** : Solution managée

🛠️ Outils Recommandés

- 📊 **Grafana** : Dashboards pro
- 🖥️ **Chronograf** : GUI InfluxDB
- ⚙️ **Telegraf** : Agent de collecte
- 🔔 **Kapacitor** : Alerting système
- 🐋 **Docker Compose** : Stack TICK

🎯 Projets d'Entraînement

1. 🌡️ **IoT Dashboard** : Capteurs température/humidité en

✅ Ce Qu'on a Appris

🎯 Concepts Clés

- 📊 Time Series = Données + Temps
- 🏷️ Tags vs Fields (crucial!)
- 📈 Measurements ≈ Tables
- ⏰ Retention Policies
- 🔍 InfluxQL pour requêtes
- 📦 Bulk writes pour performance

💻 Code Pratique

- 🐋 Installation Docker
- 🔌 Client Node.js
- ✍️ writePoints() pour écrire
- 📖 query() pour lire
- 📊 Agrégations temporelles
- 🏷️ GROUP BY tags et time

🎉 Bravo!

Tu peux maintenant stocker et analyser des données

🎯 Messages Clés à Retenir

1️⃣ Choisir la Bonne Base

- 📊 **InfluxDB** : Données temporelles, métriques, logs
- 🗄️ **PostgreSQL** : Données métier, relations
- 📄 **MongoDB** : Documents flexibles
- ⚡ **Redis** : Cache, sessions

2️⃣ Tags vs Fields = Performance

Tags : Pour filtrer et grouper (indexés) 🏷️

Fields : Pour mesurer et agréger (valeurs) 📊

3️⃣ Penser "Temps"

❓ Questions ?

N'hésitez pas à poser vos questions!



Joseph Azar

Maître de Conférences - Université Marie Louis Pasteur

Chercheur FEMTO-ST

joseph.azar@univ-fcomte.fr