MP-i

LIVRE BLANC TECHNIQUE — DEV & AUTOMATISATION × SIG & CARTOGRAPHIE

IoT & Données Capteurs : de la réception à la visualisation géospatiale

Protocoles, brokers, traitement, stockage time series et cartographie web — Guide complet 2026

MQTT · Brokers | TimescaleDB + PostGIS | MapLibre GL JS · Martin | 3 architectures de référence

Mattieu Pottier

Consultant en transformation digitale — SIG, géomatique, Web, ERP Odoo
MP-i — Mattieu Pottier Indépendant · mp-i.pro

Version 1.0 — Avril 2026
Document distribué librement

Du capteur au tableau de bord : construire un pipeline IoT solide exige bien plus qu'un broker posé rapidement et un script Python qui tourne dans un coin. Le résultat est souvent fonctionnel — rarement pérenne. Ce livre blanc suit la donnée capteur à travers ses six maillons : protocole, broker, traitement, stockage time series géolocalisé et visualisation cartographique. L'angle est délibérément architectural, pas seulement technique : pourquoi tel choix plutôt qu'un autre, comment chaque maillon conditionne les suivants, et quelles architectures open source couvrent les besoins réels des PME, bureaux d'études et collectivités — pour 20 à 150 € par mois d'infrastructure.

6
maillons du pipeline couverts

9
parties thématiques

3
architectures de référence

1979
— 2026 couvert

🎯 **Périmètre de ce livre blanc**

Ce document couvre l'IoT industriel (IIoT) et les déploiements terrain : capteurs environnementaux, positionnement GPS, équipements d'infrastructure, supervision industrielle. Il ne traite pas du firmware, de l'électronique, ni de l'approvisionnement matériel. Il commence là où le capteur émet son premier message — et suit la donnée jusqu'au tableau de bord. La cible principale est PME, bureaux d'études et collectivités (jusqu'à ~5 000 devices) ; les architectures à très haute volumétrie sont évoquées en Partie 7 à titre de repère.

SOMMAIRE

Préface

Il existe un paradoxe récurrent dans les projets qui touchent à la donnée capteur : on passe des semaines à choisir le bon matériel, à calibrer les sondes, à optimiser la fréquence d'acquisition — et puis on règle la question du pipeline en quelques heures. Un broker MQTT posé rapidement, une base de données généraliste, un script Python qui tourne dans un coin. Le résultat est souvent fonctionnel. Il est rarement pérenne.

Ce livre blanc s'attaque à cette couche trop souvent sous-estimée : comment construire un pipeline IoT solide, de la réception des données à leur visualisation géospatiale. Non pas techniquement au sens étroit — les outils sont documentés — mais architecturalement : pourquoi tel broker plutôt qu'un autre, quelles sont les contraintes réelles de volumétrie et de coût, et surtout, comment chaque maillon du pipeline conditionne les suivants.

La convergence entre données capteurs et données géospatiales n'est plus une vision. C'est une réalité implémentable avec des outils open source matures. TimescaleDB + PostGIS dans une seule instance PostgreSQL permet de stocker des séries temporelles géolocalisées, de les requêter en SQL standard, de les afficher en temps réel sur une carte web, et de les archiver en GeoParquet pour l'analytique à long terme. Cette convergence est l'angle central de ce document.

Le périmètre est délibérément orienté vers les architectures ouvertes, maîtrisables en interne et économiquement raisonnables : PME, bureaux d'études, collectivités, opérateurs de terrain. Ce livre blanc ne traite pas du firmware, de l'électronique, ni de l'approvisionnement matériel. Il commence là où le capteur émet son premier message — et suit la donnée jusqu'au tableau de bord.

Mattieu Pottier — MP-i · Mattieu Pottier Indépendant

Introduction — Le pipeline IoT : 6 maillons

Un projet IoT n'est pas un problème de capteur. C'est un problème de pipeline : comment faire transiter une donnée produite par un device physique jusqu'à un tableau de bord décisionnel, de façon fiable, sécurisée, et à un coût maîtrisé.

```
PIPELINE IOT — VUE D'ENSEMBLE


[1] Capteur → [2] Protocole → [3] Broker → [4] Traitement → [5] Stockage →

[6] Visualisation
```

> ⚠️ **La règle fondamentale**
>
> Chaque maillon conditionne les choix en aval. Un broker mal dimensionné au maillon 3 imposera une refonte au moment où le volume de capteurs doublera. Un payload non normalisé au maillon 4 rendra les requêtes du maillon 5 impossibles sans retraitement coûteux. Un schéma de stockage conçu sans dimension géospatiale au maillon 5 coûtera plusieurs semaines de migration au moment d'ajouter la cartographie du maillon 6.

Trois profils de lecture

PROFIL	PARTIES PRIORITAIRES	OBJECTIF
Décideur / DSI	Introduction, Partie 1, Partie 7, Conclusion	Comprendre les enjeux, valider une architecture, estimer les coûts
Architecte SI	Document complet	Concevoir et documenter le pipeline de A à Z
Développeur	Parties 2 à 6, Partie 9, Glossaire	Implémenter chaque maillon du pipeline

Les trois questions fondamentales

Avant tout choix d'outil ou de technologie, trois questions méritent une réponse précise. Les réponses orientent directement vers une famille d'architectures.

01

Quelle fréquence d'acquisition ?
La réponse détermine le protocole, le broker et le moteur de stockage. Un capteur émettant une mesure par heure n'impose pas les mêmes contraintes qu'un capteur industriel à 1 000 Hz — c'est une différence d'architecture fondamentale.

02

Combien de capteurs, dans 3 ans ?
Passer de 50 à 500 capteurs peut sembler modeste, mais c'est souvent le seuil qui fait basculer un broker léger en goulot d'étranglement. La question des 3 ans n'est pas un luxe — c'est une nécessité d'architecture.

03

Temps réel strict ou analytique différée ?
Ces deux besoins ne partagent ni les mêmes outils, ni la même architecture, ni les mêmes sources de données. Les confondre dans une seule interface produit généralement un résultat médiocre pour les deux usages.

Partie 1 — Taxonomie : familles de données, protocoles, brokers

Avant de choisir un outil, il faut nommer précisément ce qu'on traite. L'IoT est un terme générique qui recouvre des réalités très différentes — un capteur de température en bâtiment, une balise GPS sur un véhicule et un accéléromètre industriel à haute fréquence ne partagent ni le même protocole, ni le même volume, ni la même architecture de traitement.

1.1 — Les cinq familles de données capteurs

Toutes les données capteurs ne se ressemblent pas. Les différences de fréquence, de volume et de nature géographique conditionnent directement l'architecture du pipeline.

FAMILLE 1

Capteurs environnementaux

1/min – 1/h · Position fixe · Volume faible

Température, humidité, CO_2, particules fines, pression. Surveillance qualité de l'air, conditions de stockage, monitoring de sites industriels. Stack légère suffisante (Mosquitto + TimescaleDB sur VPS).

MQTT LoRaWAN

FAMILLE 2

Capteurs de positionnement

1/s – 1/min · Mobile · Volume moyen

GPS/GNSS, BLE intérieur. Suivi de flotte, tracking d'actifs mobiles, monitoring d'agents terrain. **PostGIS est obligatoire dès la conception du schéma** — ne pas l'intégrer coûte une migration complète.

MQTT LTE-M

FAMILLE 3

Capteurs industriels

10 Hz – 1 kHz · Position fixe · Volume élevé

Vibration, pression, débit, courant. Maintenance prédictive, détection de déséquilibre. Fréquence élevée : pré-traitement edge obligatoire — agrégation locale réduit le débit d'un facteur 100 à 1 000.

MQTT post-edge | OPC-UA

FAMILLE 4

Capteurs d'infrastructure

1/h – 1/j · Position fixe stable · Volume très faible

Compteurs eau/élec/gaz, détecteurs d'ouverture, niveaux de cuve. Smart metering, réseaux d'eau, postes électriques. Fiabilité de transmission critique — longue durée de stockage requise.

LoRaWAN | NB-IoT

FAMILLE 5

Capteurs vidéo / audio

Continu · Fixe ou mobile · Volume massif

Caméras IP, microphones, LIDAR. Volumes massifs, traitement spécialisé (GStreamer, FFmpeg, computer vision). *Hors périmètre de ce livre blanc* — technologies distinctes du pipeline IoT généraliste.

HTTP | RTSP

FAMILLE	FRÉQUENCE TYPIQUE	VOLUME	GÉOLOCALISATION	PROTOCOLE ADAPTÉ
Environnementaux	1/min – 1/h	Faible	Fixe connue	MQTT, LoRaWAN
Positionnement	1/s – 1/min	Moyen	Mobile — c'est la donnée	MQTT, LTE-M
Industriels	10 Hz – 1 kHz	Élevé	Fixe	MQTT (post-agrégation edge)
Infrastructure	1/h – 1/j	Très faible	Fixe	LoRaWAN, NB-IoT, MQTT
Vidéo / Audio	Continu	Très élevé	Fixe ou mobile	HTTP, RTSP

1.2 — Panorama des protocoles de transport

MQTT — Message Queuing Telemetry Transport

MQTT a été conçu en 1999 par Andy Stanford-Clark (IBM) et Arlen Nipper pour connecter des systèmes de télémétrie de pipelines pétroliers via des liaisons satellite coûteuses. L'objectif initial était explicite : protocole le plus compact possible, consommation énergétique minimale, fiabilité sur des liens dégradés. Ces contraintes de départ sont exactement celles de l'IoT terrain — ce qui explique pourquoi MQTT est devenu le protocole de référence mondial.

Resté propriétaire dix ans, MQTT est publié en libre accès en 2010, devient standard OASIS en 2013 (v3.1.1), puis standard ISO (ISO/IEC 20922) en 2016. La version 5.0 est publiée le **7 mars 2019**. Architecture publish/subscribe sur TCP/IP : les capteurs publient sur des *topics* (chaînes hiérarchiques, ex. `bâtiment/salle3/température`), un broker central redistribue aux abonnés. Ni le publisher ni le subscriber ne se connaissent directement.

NIVEAU	GARANTIE	MÉCANISME	USAGE RECOMMANDÉ
QoS 0	At-most-once	Fire and forget — pas d'accusé de réception	Données haute fréquence, perte acceptable (température toutes les secondes)
QoS 1	At-least-once	Accusé de réception, renvoi si non reçu — doublons possibles	Données importantes, dédoublonnage côté consommateur
QoS 2	Exactly-once	Protocole en 4 étapes — overhead réseau significatif	Données critiques à faible fréquence (compteurs, alertes, commandes)

→ **MQTT v5 — principales nouveautés (7 mars 2019)**

Reason codes sur chaque accusé de réception : le broker explique pourquoi il refuse une connexion. **Session Expiry Interval** : durée de vie configurable de la session après déconnexion. **Topic Aliases** : remplacement d'un topic string par un entier court — réduction du payload sur liens contraints. **Shared Subscriptions** : load balancing natif. **User Properties** : métadonnées key/value dans le header sans modifier le payload métier.

LoRaWAN — Long Range Wide Area Network

Protocole LPWAN sur bandes ISM non licenciées (868 MHz Europe, 915 MHz Amérique du Nord). La LoRa Alliance est fondée en 2015, la première spécification LoRaWAN v1.0 est publiée en janvier 2015. Portée > 10 km en zone rurale, consommation extrêmement faible (batterie AA pendant plusieurs années), débit très faible (250 bps à 5,5 kbps), duty cycle réglementaire de **1 % maximum** sur les canaux principaux en Europe (ETSI EN300.220).

```
ARCHITECTURE LORAWAN

Capteur LoRa → Gateway LoRa → Network Server (TTN / ChirpStack v4) → MQTT → Pipeline IoT
```

Le Network Server gère la déduplication, l'authentification des devices (AES-128) et le routage. En auto-hébergé : **ChirpStack v4** (MIT). En cloud community : **The Things Network (TTN)**. Recommandé pour : capteurs environnementaux outdoor, compteurs, détecteurs de niveau sur zone géographique étendue.

NB-IoT / LTE-M

Protocoles LPWAN cellulaires standardisés par le 3GPP. NB-IoT : optimisé pour les très faibles débits, excellente pénétration indoor, latence élevée. LTE-M : débit supérieur, latence plus faible, supporte le roaming international. Avantage sur LoRaWAN : couverture nationale garantie par l'opérateur, pas d'infrastructure gateway à déployer. Inconvénient : coût mensuel par SIM.

HTTP/REST, AMQP, CoAP

HTTP/REST : universel, mais overhead TCP+TLS à chaque requête — sous-optimal au-delà d'1 mesure/minute. Pertinent pour capteurs à faible fréquence sur connexion stable, ou intégrations avec APIs tierces existantes. **AMQP :** protocole enterprise (RabbitMQ), adapté aux intégrations IoT × SI complexes nécessitant un routage conditionnel avancé. **CoAP :** REST ultra-léger sur UDP pour réseaux très contraints (ZigBee, 6LoWPAN) — cas rares en contexte PME standard.

1.3 — Les brokers de messagerie

> ⚠️ **Point critique**
>
> Un broker n'est pas un système de stockage persistant — les messages non consommés ont une durée de vie limitée. Cette distinction est fondamentale : c'est l'une des erreurs d'architecture les plus fréquentes.

Mosquitto	Broker MQTT open source (Eclipse Foundation, EPL/EDL). Architecture mono-processus, très léger (~3 Mo RAM pour 1 000 clients). Supporte MQTT v3.1, v3.1.1 et v5.0. Pas de clustering natif — scaling vertical uniquement. **Recommandé pour :** projets à faible ou moyen débit (capteurs environnementaux, infrastructure émettant 1 mesure/min), développement, projets pilotes.	0 €
EMQX	Broker MQTT haute performance (EMQ Technologies). Depuis la v5.9.0 (mai 2025) : sous BSL 1.1. Nœud unique gratuit en production, cluster multi-nœuds = licence commerciale. Versions ≤ 5.8.x restent sous Apache 2.0 avec clustering gratuit. Interface d'administration web complète, monitoring intégré. **Recommandé pour :** projets nécessitant monitoring riche ou forte volumétrie mono-nœud.	0 € nœud unique
RabbitMQ	Broker AMQP (Broadcom/VMware, MPL-2.0). Files durables, routage conditionnel avancé, dead-letter queues natives. Pertinent pour les intégrations entre le pipeline IoT et d'autres systèmes SI (ERP, GMAO) nécessitant un routage complexe — pas le choix par défaut pour l'ingestion MQTT.	0 €

Partie 2 — Histoire : de Modbus 1979 à la convergence IoT × SIG 2026

Comprendre l'IoT actuel exige de comprendre sa trajectoire. Chaque génération de protocoles et d'outils a émergé pour répondre aux limites de la précédente, dans un contexte technique et économique en mutation permanente.

1979–1998	1999–2009	2010–2018	2019–2026	2026+ ↗
Automates en silo	**M2M & IP**	**Démocratisation**	**Convergence**	**IoT × SIG × IA**
Modbus (1979) · Profibus (1989)	MQTT v1 (1999) · OPC-UA (2006)	MQTT v3.1.1 (2014) · LoRaWAN (2015) · TimescaleDB (2017)	MQTT v5 (2019) · Edge computing · InfluxDB 3 GA (2025)	TimescaleDB+PostGIS · MCP · GeoParquet

1979–1998 — ÈRE 1

Les automates en silo : Modbus et Profibus

Les automates industriels (PLC) fonctionnent en vase clos. Modbus (Modicon, 1979) est le premier protocole de communication industriel standardisé. Profibus (1989) répond aux besoins de l'industrie de process. Ces protocoles ne sont pas conçus pour être connectés à Internet — la sécurité par isolation physique est le paradigme dominant. Pas de timestamp standardisé, pas d'horodatage natif, interopérabilité limitée.

1999–2009 — ÈRE 2

L'arrivée de l'IP et du M2M — MQTT naît en 1999

TCP/IP devient le substrat universel. Andy Stanford-Clark (IBM) et Arlen Nipper conçoivent MQTT en 1999 pour les pipelines pétroliers via liaisons satellite — trois contraintes fondatrices : compacité maximale, consommation minimale, fiabilité sur liens dégradés. OPC-UA (OPC Foundation, 2006) apporte la première architecture sécurisée et multiplateforme pour l'industrie 4.0, standardisée sous IEC 62541. Les premiers dataloggers IP font le pont entre capteurs propriétaires et réseaux TCP/IP.

2010–2018 — ÈRE 3

Démocratisation et standardisation — MQTT devient standard ISO

IBM publie MQTT en libre accès en 2010. Le standard OASIS MQTT v3.1.1 est publié le **29 octobre 2014**, puis adopté comme ISO/IEC 20922 en juillet 2016. Arduino (2005, explosion 2010) et Raspberry Pi (2012) transforment le prototypage IoT. La LoRa Alliance est fondée en 2015 — LoRaWAN v1.0 ouvre l'IoT aux capteurs outdoor sur batterie. AWS IoT Core (2015) et Azure IoT Hub lancent les plateformes managées. InfluxDB (2013) répond à l'impossibilité d'utiliser PostgreSQL standard pour ingérer des milliers de mesures/seconde. TimescaleDB (Timescale Inc., 2017) — première version stable — adopte l'angle inverse : rester dans l'écosystème PostgreSQL.

2019–2026 — ÈRE 4

Convergence IoT × Cloud × SIG × IA

MQTT v5 publié le **7 mars 2019** par l'OASIS — reason codes, session expiry, topic aliases, shared subscriptions. L'edge computing s'impose économiquement : Raspberry Pi 4, NVIDIA Jetson permettent pré-traitement local et fonctionnement dégradé sans réseau. InfluxDB 3 Core (IOx, Rust + Arrow + Parquet) atteint la disponibilité générale (GA) le **15 avril 2025** sous MIT/Apache 2.0 — rupture totale avec v1/v2, Flux abandonné. TimescaleDB + PostGIS coexistent dans la même instance PostgreSQL : la convergence time series × géospatial devient la référence open source. Node-RED v4 publié le **20 juin 2024** (OpenJS Foundation, Apache 2.0).

PÉRIODE	PARADIGME	PROTOCOLES EMBLÉMATIQUES	STOCKAGE
1979–1998	Automates en silo	Modbus (1979), Profibus (1989)	Pas de stockage centralisé
1999–2009	M2M, premiers réseaux IP	MQTT v1 (1999), OPC-UA (2006)	Bases relationnelles détournées
2010–2018	Démocratisation, standardisation	MQTT v3.1.1 (2014), LoRaWAN (2015)	InfluxDB (2013), TimescaleDB (2017)
2019–2026	Convergence Cloud × SIG × IA	MQTT v5 (2019), Edge computing	TimescaleDB+PostGIS, InfluxDB 3 Core (2025)

Partie 3 — Réception et ingestion : du capteur au broker

La réception est le premier maillon actif du pipeline — celui où la donnée physique devient une donnée numérique transmissible. C'est aussi le maillon où les erreurs de conception sont les plus difficiles à corriger après coup : un broker sous-dimensionné ou une architecture de réception inadaptée impose une refonte complète, pas un simple ajustement de configuration.

3.1 — Les quatre architectures de réception

ARCHITECTURE A

Connexion directe broker

Réseau IP stable · WiFi / Ethernet / cellulaire · Latence minimale

Capteur WiFi / Ethernet

↓

MQTT Broker (TLS 8883)

↓

Pipeline

Bâtiments connectés, salles informatiques, équipements en réseau d'entreprise, capteurs WiFi en zone de couverture garantie.

ARCHITECTURE B

Edge gateway

Réseau intermittent · Capteurs non-IP · Store-and-forward

Capteurs série / radio

↓

Gateway (RPi / ESP32 / routeur industriel) + buffer local SQLite

↓

Broker Cloud / SI

Chantiers, sites isolés, capteurs Modbus, zones à couverture réseau intermittente. La gateway buffer localement et transmet à la reconnexion.

ARCHITECTURE C

LoRaWAN

Outdoor longue portée · Batterie · Faible consommation

Capteur LoRa

↓

Gateway LoRa → Network Server (TTN / ChirpStack v4)

↓

MQTT → Pipeline

Compteurs eau/élec/gaz, capteurs environnementaux outdoor, surveillance de bâti sur zone étendue. Duty cycle 1 % — pas adapté au temps réel.

ARCHITECTURE D

HTTP Push

Capteurs IP · Faible fréquence · Connexion stable

Capteur IP

↓

Endpoint REST (FastAPI / Express)

↓

File de traitement → Broker ou TSDB

< 1 mesure/minute, intégration avec APIs tierces existantes, webhooks depuis plateformes SaaS. Overhead TCP+TLS à chaque requête — inefficace à haute fréquence.

→ **Règle de décision principale**

Si le réseau est fiable et les capteurs ont une connectivité IP, l'Architecture A est préférable pour sa simplicité. Si l'un de ces deux critères fait défaut, l'Architecture B avec buffer local est la bonne réponse. LoRaWAN s'impose dès que la portée et la durée de vie sur batterie priment sur la fréquence et la latence.

3.2 — Qualité de service et mécanismes de fiabilité MQTT

Au-delà des trois niveaux de QoS (couverts en Partie 1), MQTT fournit des mécanismes complémentaires de fiabilité indispensables en production.

Retained message	Le broker conserve le dernier message publié sur un topic. Tout nouveau subscriber reçoit immédiatement ce message retained lors de sa connexion, sans attendre la prochaine publication. Utile pour connaître l'état courant d'un capteur instantanément.
Last Will & Testament	Le client déclare à la connexion un message qui sera automatiquement publié par le broker si le client se déconnecte de façon anormale (coupure réseau, crash, timeout). Mécanisme de détection de panne capteur natif — le broker publie `{"statut":"offline"}` sur `appareils/{id}/statut` si le capteur disparaît sans se déconnecter proprement.
Dead Letter Queue	Les messages non traitables (payload invalide, consommateur défaillant) doivent être dirigés vers une file spéciale plutôt qu'être perdus. Non natif MQTT — à implémenter au niveau du pipeline de traitement (Partie 4) ou via les règles de routage EMQX.
Store-and-forward edge	La gateway doit implémenter une file locale persistante sur disque (SQLite, pas en mémoire) pour absorber les déconnexions réseau. La persistance sur disque est indispensable — une file en mémoire est perdue en cas de redémarrage. La file est vidée dans l'ordre chronologique à la reconnexion.

3.3 — Sécurité à l'ingestion

> ⚠️ **Principe fondamental**
>
> La sécurité du pipeline IoT ne peut pas être ajoutée après coup. Elle doit être conçue dès l'architecture initiale, au niveau de chaque maillon. L'ingestion est le maillon le plus exposé — c'est là que les devices communiquent depuis l'extérieur du SI.

TLS : toute connexion MQTT en production doit utiliser TLS (port 8883 vs 1883 non chiffré). TLS garantit la confidentialité des données en transit et l'authenticité du broker. TLS 1.3 recommandé — TLS 1.0 et 1.1 dépréciés depuis 2021 (RFC 8996).

mTLS — authentification mutuelle : en standard, TLS authentifie le serveur auprès du client. Le mTLS va plus loin : le client s'authentifie également via son propre certificat X.509. Chaque device dispose d'un certificat unique signé par la CA interne. Avantages : pas de mot de passe sur le device, révocation granulaire (un device compromis = un certificat révoqué sans affecter les autres), résistance au rejeu.

ACL MOSQUITTO — CONTRÔLE D'ACCÈS PAR TOPIC

```
# acl.conf — principe de moindre privilège
# Device capteur-001 : publication uniquement sur son topic
user capteur-001
topic write capteurs/+/capteur-001/#

# Backend applicatif : lecture sur tous les topics capteurs
user backend-app
topic read capteurs/#
```

Segmentation réseau : les devices IoT doivent être isolés dans un VLAN dédié, séparé du réseau SI principal. Le firewall entre VLAN IoT et VLAN SI ne laisse passer que les flux strictement nécessaires : le broker MQTT (port 8883) depuis le VLAN IoT. Le broker est le seul point d'entrée autorisé pour les données capteurs.

Partie 4 — Traitement : de la donnée brute à la donnée utile

La donnée brute qui sort d'un capteur n'est pas directement exploitable. Elle arrive sans horodatage normalisé, avec des champs mal nommés, des valeurs aberrantes non filtrées, dans un format qui dépend du firmware du device. Le traitement est la couche qui transforme cette donnée brute en donnée métier : normalisée, enrichie, agrégée, prête à être stockée. C'est aussi la couche la plus sous-estimée — les problèmes surgissent des mois plus tard si elle est bâclée.

4.1 — Les trois niveaux de traitement

Niveau 1	**Edge — traitement local avant transmission** Agrégation temporelle (capteur 1 kHz → 1 valeur/seconde = ÷1 000 du volume réseau), filtrage des valeurs aberrantes, compression payload (MessagePack, CBOR sur liaisons contraintes), buffering local en cas de déconnexion. Outils : Python/Go sur RPi, MicroPython sur ESP32. **Règle :** logique simple et robuste uniquement — un edge qui tombe coupe tous les capteurs qu'il agrège.
Niveau 2	**Stream — traitement en flux temps réel** Normalisation du payload au format canonique, enrichissement géographique (rattachement de coordonnées aux capteurs fixes), dédoublonnage QoS 1, détection d'anomalies simples (dépassement de seuil), routage conditionnel (alertes vers file prioritaire). Outils : Node-RED v4, Python asyncio + paho-mqtt, faust-streaming sur Kafka.
Niveau 3	**Batch — retraitement analytique différé** Recalcul de Continuous Aggregates sur des plages historiques, détection d'anomalies complexes (ML sur fenêtres longues), migration de schéma ou de format, génération de rapports périodiques. Outils : dbt (transformations SQL sur TimescaleDB), DuckDB (analytique locale sur GeoParquet), Apache Spark (grand volume distribué).

4.2 — Node-RED v4 : l'orchestrateur visuel terrain

Node-RED est un outil d'orchestration visuel par flows, développé initialement par IBM et transféré à l'**OpenJS Foundation** (Apache 2.0). La version 4.0 a été publiée le **20 juin 2024** et requiert Node.js 18+. Son modèle repose sur des flows : des chaînes de nœuds connectés visuellement, chaque nœud effectuant une transformation, une lecture ou une écriture. Un flow MQTT complet tient en quelques nœuds configurés en quelques minutes sans écrire une ligne de code.

FORCES

Ce que Node-RED fait bien

- ✓ MQTT natif : nœuds `mqtt-in` / `mqtt-out` intégrés, TLS inclus
- ✓ Parsing et transformation JSON sans code
- ✓ Enrichissement géographique via appel API externe
- ✓ Routage conditionnel : nœud `switch` selon valeur capteur
- ✓ Intégration PostgreSQL/TimescaleDB native (nœud communauté)
- ✓ Débogage en temps réel à n'importe quel point du flow

LIMITES

Ce qu'il faut anticiper

- → Mono-processus Node.js : pas de parallélisme natif
- → Gestion d'erreurs explicite : câbler des nœuds `catch` sur chaque partie critique
- → Pas de versioning natif : export Git régulier indispensable
- → Monitoring limité sans outil tiers (FlowFuse, Prometheus)
- → Seuil recommandé : quelques milliers de messages/minute max

4.3 — Normalisation : la fondation non négociable

Un payload mal normalisé en entrée de TSDB rend toutes les requêtes métier impossibles sans retraitement coûteux. C'est une décision d'architecture, pas une tâche technique

mineure.

JSON — PAYLOAD CANONIQUE RECOMMANDÉ

```
{
  "device_id": "env-cayenne-003",
  "timestamp": "2026-04-01T08:30:00.000Z",
  "location": { "lat": 4.9222, "lon": -52.3135 },
  "measurements": {
    "temperature_c": 28.4,
    "humidity_pct": 82.1,
    "co2_ppm": 412
  },
  "quality": "ok"
}
```

CHAMP	RÈGLE	RAISON
`device_id`	Identifiant stable, unique, non réutilisable	Clé de jointure avec le référentiel devices
`timestamp`	UTC, ISO 8601 avec milliseconde	Jamais l'heure locale — source de bugs au premier changement d'heure
`location`	Coordonnées WGS84 (EPSG:4326)	Interopérable avec PostGIS sans conversion
Noms de mesures	snake_case avec unité suffixée	`temperature_c` est sans ambiguïté, `temp` ne l'est pas
`quality`	Flag optionnel (`ok` , `degraded` , `error`)	Filtre les mesures dégradées sans supprimer les données

4.4 — Downsampling : la décision la plus importante pour la pérennité

Sans politique de downsampling, une infrastructure IoT réaliste atteint des volumes ingérables. 200 capteurs émettant 1 mesure/minute = **105 millions de lignes/an**. Le downsampling remplace les données brutes anciennes par des agrégats à résolution inférieure — compromis explicite entre précision historique et coût de stockage.

PÉRIODE	RÉSOLUTION CONSERVÉE	MÉTHODE
0 → 48h	Native (1/min)	Données brutes en TimescaleDB
48h → 30j	5 min (÷5)	Continuous Aggregate TimescaleDB
30j → 1 an	1 heure (÷60)	Continuous Aggregate TimescaleDB
> 1 an	1 jour ou archive Parquet	Tiering cold sur Cloudflare R2

Cette politique réduit le volume de **99 %** sur les données de plus d'un an, tout en conservant la granularité fine pour l'analyse récente.

4.5 — Données GPS : traitement spécifique et geofencing

Les capteurs de positionnement mobile produisent des données qui nécessitent un traitement spécifique avant persistance. Trois problèmes sont systématiques.

1. **Lacunes GPS :** en milieu urbain dense, sous couvert forestier ou en tunnel, le signal est interrompu. Stratégie : interpolation linéaire (acceptable pour vitesses lentes) ou marquage explicite de l'interruption sans interpolation (préférable si la vitesse peut être élevée).

2. **Points aberrants :** un GPS peut ponctuellement renvoyer une position erronée très éloignée de la trajectoire. Filtre simple : calculer la vitesse impliquée entre deux mesures consécutives — si physiquement impossible (> 200 km/h pour un capteur piéton), rejeter le point.

3. **Précision variable (HDOP) :** le champ HDOP (Horizontal Dilution of Precision) indique la qualité du fix GPS. HDOP < 2 : excellent. HDOP 2–5 : bon. HDOP > 5 : dégradé. Stocker le HDOP avec chaque position permet de filtrer les mesures dégradées lors des requêtes analytiques sans les supprimer.

Le **geofencing en temps réel** détecte l'entrée ou la sortie d'un device dans une zone géographique prédéfinie. Implémentation avec PostGIS :

SQL — GEOFENCING POSTGIS

```sql
-- Vérifier si un point est dans une zone d'alerte
SELECT z.nom, z.type_alerte
FROM zones_surveillance z
WHERE ST_Within(
  ST_SetSRID(ST_MakePoint(-52.3135, 4.9222), 4326),
  z.geometrie
);
```

Partie 5 — Stockage : time series, géospatial et stratégie tiering

Le stockage est le maillon qui transforme un flux de données éphémères en historique exploitable. Un mauvais choix de moteur, un schéma mal pensé ou l'absence de politique de rétention peuvent ruiner plusieurs mois de données collectées.

5.1 — Pourquoi le stockage time series est fondamentalement différent

Une table PostgreSQL sans extension spécialisée peut stocker des données IoT — mais elle dégradera rapidement en performance dès que le volume dépasse quelques millions de lignes. Les données time series ont quatre caractéristiques qui exigent un traitement spécialisé.

Append-only
Les mesures passées ne se modifient jamais. L'écriture est séquentielle dans le temps. Ce comportement permet d'optimiser le stockage d'une façon impossible avec les données transactionnelles classiques.

Volume croissant prévisible
500 capteurs à 1 mesure/minute = 720 000 lignes/jour, 263 millions/an. Ce calcul doit être fait avant de choisir le moteur de stockage.

Requêtes temporelles dominantes
95 % des requêtes portent sur une plage de temps + filtre device_id. L'index temporel est primordial — un index B-tree standard sur 500 millions de lignes sans partitionnement est un gouffre de performance.

Compression spécialisée
Les séries IoT ont une forte corrélation temporelle. Les algorithmes delta encoding et run-length encoding atteignent 90 à 95 % de compression — inaccessible avec un stockage orienté lignes standard.

5.2 — TimescaleDB + PostGIS : la stack recommandée

TimescaleDB est une extension PostgreSQL open source (**Apache 2.0**), développée par Timescale Inc. (fondée avril 2017). Première version stable : **novembre 2018**. Son principe fondamental : partitionner automatiquement les données en **hypertables** par intervalles temporels (chunks). De l'extérieur, une hypertable se comporte comme une table PostgreSQL standard — les requêtes SQL sont identiques.

SQL — CRÉATION HYPERTABLE TIMESCALEDB

```
-- Table standard PostgreSQL
CREATE TABLE measurements (
  time        TIMESTAMPTZ NOT NULL,
  device_id   TEXT        NOT NULL,
  temperature FLOAT,
  humidity    FLOAT
);

-- Conversion en hypertable (une seule commande)
SELECT create_hypertable('measurements', 'time');

-- TimescaleDB crée automatiquement un nouveau chunk par intervalle de 7 jours.
-- Les requêtes sur une plage temporelle ne lisent que les chunks couverts — chunk exclusion.
```

SQL — CONTINUOUS AGGREGATE POUR DOWNSAMPLING AUTOMATIQUE

```
-- Vue agrégée par heure — se met à jour automatiquement en arrière-plan
CREATE MATERIALIZED VIEW measurements_hourly
WITH (timescaledb.continuous) AS
SELECT
  time_bucket('1 hour', time) AS bucket,
  device_id,
  AVG(temperature)            AS temp_avg,
  MAX(temperature)            AS temp_max,
  MIN(temperature)            AS temp_min,
  COUNT(*)                    AS nb_mesures
FROM measurements
GROUP BY bucket, device_id
WITH NO DATA;

SELECT add_continuous_aggregate_policy('measurements_hourly',
  start_offset => INTERVAL '3 hours',
  end_offset   => INTERVAL '1 hour',
  schedule_interval => INTERVAL '1 hour'
);
```

SQL — HYPERTABLE AVEC COLONNE GÉOSPATIALE POSTGIS

```sql
-- Table positions géolocalisées — TimescaleDB + PostGIS dans la même instance
CREATE TABLE positions (
  time      TIMESTAMPTZ NOT NULL,
  device_id TEXT        NOT NULL,
  geom      GEOMETRY(Point, 4326),  -- type PostGIS dès le départ
  speed_kmh FLOAT,
  hdop      FLOAT
);
SELECT create_hypertable('positions', 'time');
CREATE INDEX ON positions USING GIST (geom);  -- index spatial

-- Requête spatio-temporelle : positions dans une zone sur les 24 dernières heures
SELECT device_id, time, geom, speed_kmh
FROM positions
WHERE time > NOW() - INTERVAL '24 hours'
  AND ST_Within(geom, ST_GeomFromText(
    'POLYGON((-52.35 4.90, -52.30 4.90, -52.30 4.95, -52.35 4.95, -52.35 4.90))',
    4326
  ))
ORDER BY device_id, time;
```

💡 **Note sur le tiering cloud natif**

Le tiering natif de chunks vers Amazon S3 est une fonctionnalité **exclusive à Timescale Cloud** — elle n'est pas disponible sur un TimescaleDB auto-hébergé (VPS). Sur un déploiement auto-hébergé, le tiering cold vers Cloudflare R2 se fait via un export batch périodique en GeoParquet (script Python ou `pg_dump` + upload R2), puis suppression des chunks anciens via la politique de rétention. Ce processus est entièrement automatisable via `pg_cron` ou un job systemd.

5.3 — InfluxDB 3 Core — panorama et points de vigilance 2026

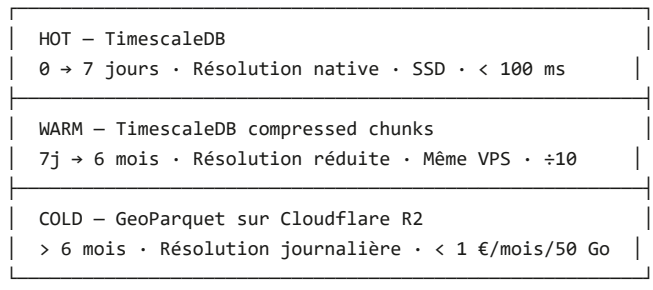
InfluxDB est la référence historique des TSDB. Sa trajectoire sur 12 ans est instructive pour comprendre les risques d'un choix de moteur propriétaire.

VERSION	DATE	MOTEUR	LANGAGE QUERY	REMARQUES
v1.x	2013	TSM (maison)	InfluxQL	Référence, encore très déployé
v2.x	2020	TSM (maison)	Flux (propre)	Remplacement de InfluxQL par Flux
v3 Cloud	Avril 2023	IOx (Rust, Arrow)	SQL + InfluxQL	Cloud uniquement d'abord
v3 Core OSS	**GA 15 avril 2025**	**IOx**	**SQL (Flux abandonné)**	**MIT/Apache 2.0**

> 🚫 **Points de vigilance critiques 2026**
>
> **Rupture totale** entre v2 et v3 : modèle de données, langage de requête (Flux abandonné) et moteur de stockage entièrement différents — pas de migration automatique. **Flux abandonné :** les équipes ayant investi dans Flux v2 voient leur investissement déprécié. **Recommandation :** pour un nouveau projet IoT sans héritage InfluxDB, évaluer TimescaleDB en premier. Si un InfluxDB 1.x est en production, ne pas migrer vers v3 sans analyse approfondie.

5.4 — Stratégie hot / warm / cold : le tiering en pratique

ARCHITECTURE DE STOCKAGE — TIERING 3 NIVEAUX



NIVEAU	VOLUME BRUT ANNUEL	VOLUME APRÈS COMPRESSION	COÛT STOCKAGE
Hot (7j)	~5 Go	~0,5 Go compressé	Inclus VPS (30–40 €/mois)
Warm (6 mois)	~130 Go	~13 Go compressé	Inclus VPS
Cold (> 6 mois)	~260 Go/an	~30 Go/an (GeoParquet)	~0,45 €/mois (R2)

Base de calcul : 500 capteurs / 1 mesure/minute. Cloudflare R2 : 0,015 $/Go/mois, sans frais d'egress vers le réseau Cloudflare.

5.5 — Comparatif moteurs de stockage IoT 2026

CRITÈRE	TIMESCALEDB + POSTGIS	INFLUXDB 3 CORE	GEOPARQUET + DUCKDB
Requêtes temps réel	✅ < 100 ms	✅ < 50 ms	❌ Batch uniquement
SQL natif	✅ Standard PostgreSQL	✅	✅
Géospatial natif	✅ PostGIS complet	❌	⚠️ Extension spatiale limitée
Downsampling automatique	✅ Continuous Aggregates	✅	❌ ETL manuel
Édition / UPDATE	✅ ACID complet	❌ Append-only	❌ Lecture seule
Compression	✅ 90–95 %	✅ Parquet	✅ Parquet colonnes
Licence	Apache 2.0	MIT / Apache 2.0	Apache 2.0
Coût infra	Faible (VPS 30–40 €)	Moyen	Très faible (< 1 €/50 Go)
Maturité 2026	⭐⭐⭐	⭐⭐	⭐⭐⭐ (analytique)
Recommandé pour	Stack principale PME/BE	Analytics haute cardinalité	Archive longue durée

5.6 — Les trois erreurs de schéma à éviter absolument

1. **Stocker latitude et longitude en colonnes FLOAT séparées.** Sans type géospatial `GEOMETRY`, il est impossible d'utiliser les fonctions PostGIS (`ST_Within`, `ST_Distance`, `ST_MakeLine`). Un index sur deux colonnes float ne peut pas remplacer un index GiST spatial. Migrer rétroactivement sur des centaines de millions de lignes est coûteux. Solution : `geom GEOMETRY(Point, 4326)` dès la création de la table.

2. **Une table par device** (`mesures_capteur_001`, `mesures_capteur_002`…). Anti-pattern classique : rend toutes les requêtes multi-devices impossibles sans `UNION`, empêche les Continuous Aggregates, explose le catalogue PostgreSQL dès quelques centaines de capteurs. Solution : une seule hypertable avec `device_id` comme colonne de segmentation dans la politique de compression.

3. **Pas de politique de rétention dès le départ.** "On verra plus tard" devient "on a 500 Go de données brutes non agrégées et les requêtes mettent 30 secondes". La politique de rétention se définit et se teste avant le premier déploiement en production — elle est triviale à configurer à l'avance et coûteuse à corriger rétroactivement.

Partie 6 — Visualisation : Grafana, MapLibre GL JS, Kepler.gl

La visualisation est le maillon final du pipeline — celui qui transforme des millions de lignes en base de données en une information compréhensible et exploitable. C'est aussi la couche la plus visible : un pipeline techniquement impeccable dont la visualisation est confuse ou inadaptée ne remplit pas sa promesse. Trois outils complémentaires couvrent la quasi-totalité des besoins IoT géolocalisés.

6.1 — Grafana : le tableau de bord opérationnel de référence

Grafana est la plateforme de visualisation open source de référence pour les séries temporelles. Développée par Grafana Labs, la version OSS est distribuée sous licence **AGPLv3**. La version courante au moment de la rédaction est la **12.x** (avril 2026). Son modèle repose sur des *data sources* (connecteurs vers les bases de données) et des *panels* organisés en dashboards. Un dashboard Grafana complet pour un déploiement IoT géolocalisé se configure sans écrire une seule ligne de code.

SQL — REQUÊTE GRAFANA POUR SÉRIE TEMPORELLE MULTI-DEVICES

```
SELECT
  time_bucket('5 minutes', time) AS "time",
  device_id,
  AVG(temperature)               AS temperature
FROM measurements
WHERE
  $__timeFilter(time)       -- macro Grafana : filtre sur la plage sélectionnée
  AND device_id IN ($devices) -- variable de dashboard : sélection multi-capteurs
GROUP BY 1, 2
ORDER BY 1;
```

PANEL	USAGE TYPIQUE IOT
Time series	Courbe de température, humidité, tension sur le temps
Stat	Valeur courante d'un capteur, dernière mesure reçue
Gauge	Jauge visuelle pour un indicateur de niveau (cuve, batterie)
Table	Liste des dernières mesures par device avec colonnes triables
Geomap	Cartographie des devices avec markers colorés selon valeur
Bar chart	Comparaison d'agrégats entre devices ou zones
Logs	Flux d'événements bruts (erreurs, alertes, changements d'état)
Canvas	Visualisation sur schéma métier (plan d'usine, synoptique)

> ⚠️ **Limites de Grafana pour la cartographie avancée**
>
> Le panel Geomap ne consomme pas nativement des tuiles vectorielles MVT — il affiche des points GeoJSON. Au-delà de 10 000 points, les performances se dégradent. Pas de trajectoires, pas de heatmaps spatiales H3, pas de filtrage spatial interactif. Pour ces cas d'usage, combiner Grafana (monitoring opérationnel) avec MapLibre GL JS (analyse spatiale).

6.2 — MapLibre GL JS : la bibliothèque cartographique open source

MapLibre GL JS est une bibliothèque TypeScript open source pour la publication de cartes interactives dans un navigateur web. Distribuée sous licence **BSD-3-Clause**, c'est un fork de mapbox-gl-js effectué en décembre 2020 lorsque Mapbox a abandonné la licence open source. La version courante est la **5.x**. Son rendu repose sur **WebGL** : les tuiles vectorielles sont rendues entièrement côté client (GPU), permettant l'affichage fluide de millions d'entités sans aller-retour serveur.

JAVASCRIPT — PATTERN GEOJSON DYNAMIQUE (VOLUMES FAIBLES, < 10 000 ENTITÉS)

```
map.addSource('capteurs', {
  type: 'geojson',
  data: '/api/capteurs/positions?since=1h'
});
map.addLayer({
  id: 'capteurs-layer',
  type: 'circle',
  source: 'capteurs',
  paint: {
    'circle-radius': 8,
    'circle-color': ['interpolate', ['linear'], ['get', 'temperature'],
      20, '#2196F3',   // bleu — froid
      30, '#FF9800',   // orange — chaud
      40, '#F44336'    // rouge — critique
    ]
  }
});
```

JAVASCRIPT — PATTERN MVT VIA MARTIN + ACTUALISATION TEMPS RÉEL

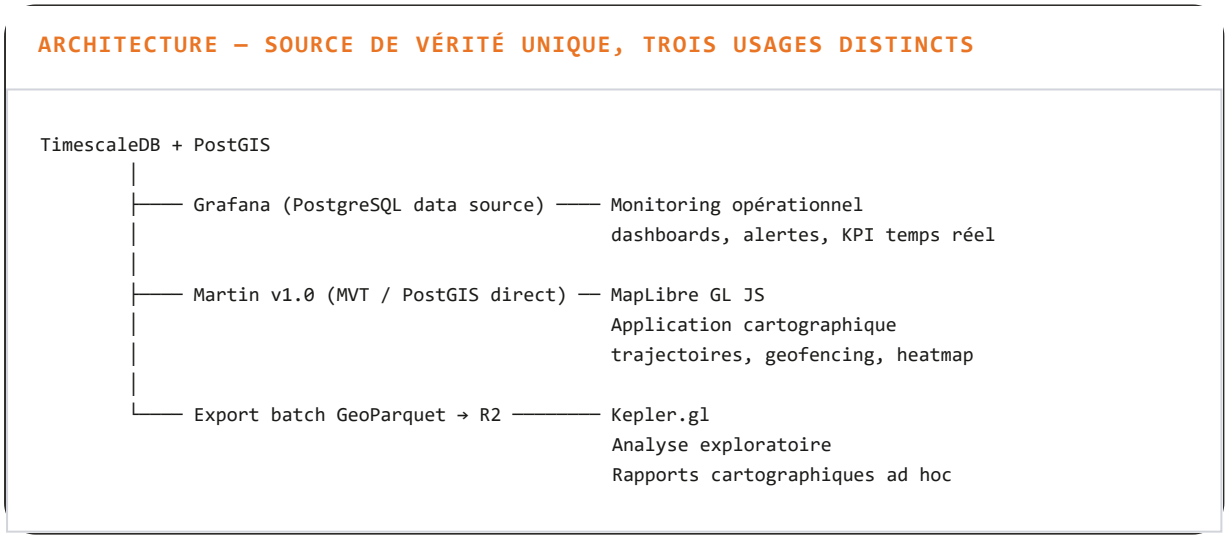
```
// Martin v1.0 (MIT, nov. 2025) — serveur MVT Rust, connecté directement à PostGIS
map.addSource('positions-historiques', {
  type: 'vector',
  tiles: ['https://tiles.example.com/positions/{z}/{x}/{y}']
});
map.addLayer({
  id: 'trajectoires', type: 'line', source: 'positions-historiques',
  'source-layer': 'positions',
  paint: { 'line-color': '#e97423', 'line-width': 2 }
});

// Polling GeoJSON toutes les 30 secondes pour les positions temps réel
async function refreshPositions() {
  const geojson = await fetch('/api/capteurs/positions?since=5min').then(r => r.json());
  map.getSource('capteurs').setData(geojson);
}
setInterval(refreshPositions, 30_000);
refreshPositions();
```

6.3 — Kepler.gl et architecture de visualisation complète

Kepler.gl est une bibliothèque open source développée par Uber, distribuée sous licence **MIT**. Construite sur MapLibre GL JS (depuis la v3) et deck.gl, elle permet de créer des visualisations géospatiales avancées — heatmaps H3, flux animés, arcs, couches 3D — via une interface drag-and-drop ou une API React. Cas d'usage IoT : analyse exploratoire de GeoParquet historique, visualisation de heatmaps H3, flux entre points d'origine/destination, mode playback temporel.

CRITÈRE	GRAFANA GEOMAP	MAPLIBRE GL JS	KEPLER.GL
Contrôle développeur	Faible (UI Grafana)	Élevé — tout est API	Moyen — UI préconfigurée
Visualisations avancées (H3, arcs, flux)	❌	⚠️ Manuelle	✅ Natif drag-and-drop
Intégration pipeline IoT	✅ Direct TimescaleDB	✅ Facile	⚠️ Nécessite Redux
Temps réel	✅ Alerting natif	✅ Polling / WebSocket	❌ Statique
Analyse exploratoire ad hoc	❌	❌	✅
Usage recommandé	Points fixes simples, alertes	Application web IoT sur mesure	Analyse, rapports cartographiques



Partie 7 — Guide décisionnel : choisir son architecture IoT géolocalisé

Cette partie synthétise les éléments des parties précédentes en un guide décisionnel opérationnel. L'objectif n'est pas de proposer une architecture universelle — il n'en existe pas. C'est de structurer les questions à poser en amont, et d'associer à chaque combinaison de réponses une recommandation architecturale concrète et justifiée.

7.1 — Les 8 questions fondamentales avant tout choix

Q1

Quel est le volume de devices ?

< 50 : architecture légère, broker simple. 50–500 : stack standard, TimescaleDB VPS, Martin. 500–5 000 : monitoring dédié, réplication. > 5 000 : architecture distribuée, évaluer Kafka.

Q2

Quelle fréquence d'émission par device ?

La combinaison Volume × Fréquence détermine le débit d'ingestion. Au-delà de 500 messages/seconde, un broker intermédiaire (EMQX ou Kafka bridge) est nécessaire. TimescaleDB VPS absorbe confortablement jusqu'à ~10 000 insertions/s avec batching.

Q3

Les données sont-elles géolocalisées ?

Fixes : coordonnées dans le référentiel devices, enrichissement stream. Mobiles (GPS) : colonne `GEOMETRY(Point, 4326)`, index GiST, pipeline GPS. Non géolocalisées : PostGIS non nécessaire, TimescaleDB seul suffit.

Q4

Quel besoin de latence pour la visualisation ?

Monitoring opérationnel (10–60 s) : Grafana + polling. Alerte réactive (5–10 s) : Grafana alerting + webhook. Temps réel strict (< 2 s) : WebSocket + cache Redis. La grande majorité des cas IoT est dans "monitoring opérationnel" — valider le besoin réel avant le temps réel strict.

Q5

Contraintes de connectivité réseau ?

Stable : MQTT direct, QoS 1 ou 2. Dégradée / intermittente : buffer local obligatoire, store-and-forward. Très contraints en énergie : LoRaWAN ou NB-IoT selon couverture opérateur. Hors ligne prolongé : synchronisation batch à la reconnexion.

Q6

Qui accède aux données et comment ?

Technicien terrain → Grafana mobile. Manager → Grafana lecture seule. Analyste SIG → Kepler.gl ou QGIS via GeoParquet. Application web cliente → MapLibre + API REST + MVT. Système tiers → API REST ou Webhook avec token.

Q7

Contraintes de souveraineté et sécurité ?

Données sensibles / réglementées : hébergement France ou UE, chiffrement TLS + au repos, traçabilité des accès. Secteur critique (eau, énergie, transport) : se référer à NIS2 et réglementation OIV. Données de positionnement de personnes physiques : RGPD.

Q8

Quelle maturité technique de l'équipe ?

Faible → Node-RED + Grafana Cloud + TimescaleDB managé. Moyen → Node-RED ou Python + TimescaleDB VPS + Grafana OSS. Élevé (DevOps + Dev) → architecture complète auto-hébergée, Martin. Expert → Kafka bridge, cluster PostGIS.

Le piège le plus courant

Dimensionner l'architecture pour la charge maximale théorique, sans tenir compte de la maturité réelle de l'équipe. Une architecture complexe mal opérée est plus risquée qu'une architecture simple bien maintenue.

7.2 — Trois architectures de référence

ARCHITECTURE 1

Starter — IoT léger

~20 €/mois · 1–2 semaines déploiement

< 50 devices · ≤ 1 mesure/min · équipe limitée · budget < 30 €/mois.

Device → MQTT TLS (8883)

↓

Mosquitto

↓

Node-RED v4

↓

TimescaleDB

↓

Grafana OSS (panel Geomap)

→ Un seul VPS, tous services en containers Docker

→ Downsampling via Continuous Aggregates

→ Sauvegarde pg_dump quotidienne → R2

→ Archive cold GeoParquet → Cloudflare R2 < 1 €/mois

ARCHITECTURE 2

Standard — IoT géolocalisé opérationnel

~50 €/mois · 3–4 semaines déploiement

50–500 devices · données GPS ou capteurs fixes · ≤ 1 mesure/min · dev backend disponible.

Device → MQTT TLS / mTLS

↓

EMQX (nœud unique BSL 1.1 — gratuit)

↓

Python asyncio + paho-mqtt

↓

TimescaleDB + PostGIS

↓

Grafana OSS + Martin MVT + MapLibre GL JS

→ Séparation broker / BDD sur 2 VPS distincts

→ Authentification MQTT par certificats X.509 (mTLS)

ARCHITECTURE 3

Production — Haute disponibilité

~150 €/mois · 6–10 semaines déploiement

500–5 000 devices · SLA requis · données critiques · ≤ 1 mesure/10 s · équipe DevOps.

Devices → MQTT / LoRaWAN / HTTP

↓

EMQX cluster → Kafka bridge

↓

Python faust-streaming

↓

TimescaleDB primary + replica + PostGIS

↓

Grafana + Martin + MapLibre + Kepler.gl

→ Réplication PostgreSQL — primary écritures, replica lectures

→ EMQX cluster 3 nœuds : **licence commerciale requise** (BSL 1.1 v5.9+)

→ Alternative libre : Mosquitto + load balancer + sessions persistantes

→ Redis (déduplication), Prometheus + Grafana (monitoring infra)

→ Runbook restauration testé · RTO cible : 1h

Martin expose
→ PostGIS en MVT pour
MapLibre

Rétention : 30j
→ bruts · 1 an agrégé
· archive cold R2

7.3 — Tableau décisionnel synthétique

CRITÈRE	ARCHITECTURE 1 — STARTER	ARCHITECTURE 2 — STANDARD	ARCHITECTURE 3 — PRODUCTION
Devices	< 50	50–500	500–5 000
Fréquence max	1/min	1/min	1/10s
Débit max	~1 msg/s	~10 msg/s	~500 msg/s
Haute disponibilité	Non	Non	Oui (réplication)
Budget infra	~20 €/mois	~50 €/mois	~150 €/mois
Maturité requise	Faible	Moyenne	Élevée
Délai déploiement	1–2 semaines	3–4 semaines	6–10 semaines
Broker	Mosquitto	EMQX nœud unique	EMQX cluster
Pipeline	Node-RED	Python asyncio	Faust-streaming
Stockage	TimescaleDB	TimescaleDB + PostGIS	TimescaleDB HA + PostGIS
Carte	Geomap Grafana	MapLibre + Martin	MapLibre + Martin + Kepler.gl

7.4 — Les 7 erreurs d'architecture les plus fréquentes

1. **Dimensionner pour la charge théorique maximale.** "On pourrait avoir 10 000 capteurs un jour" ne justifie pas une Architecture 3 dès le départ. Une Architecture 1 bien conçue est extensible. Commencer simple, mesurer, puis évoluer.

2. **Choisir le broker avant de connaître les volumes.** EMQX cluster pour 20 devices, Mosquitto pour 2 000 devices : les deux sont des erreurs. Le broker doit correspondre au débit réel, pas à la notoriété de l'outil.

3. **Ignorer la politique de rétention au démarrage.** "On verra plus tard" devient "on a 500 Go de données brutes non agrégées et les requêtes mettent 30 secondes". La politique de rétention se définit avant le premier déploiement en production.

4. **Confondre besoin de carte et besoin de SIG.** Grafana Geomap suffit souvent. Déployer QGIS Server ou GeoServer pour afficher 200 capteurs fixes est un surdimensionnement coûteux.

5. **"Les données doivent être en temps réel" sans définition précise.** Demander : "dans combien de temps après la mesure l'information doit-elle être visible ?" La réponse est rarement < 5 secondes pour un usage opérationnel standard.

6. **Une table par device.** Anti-pattern documenté (section 5.6) encore répandu. Requêtes multi-devices impossibles sans UNION, Continuous Aggregates bloquées, catalogue PostgreSQL surchargé.

7 **Pas de séparation lecture / écriture.** Grafana, l'API REST et Martin lisent en permanence la base pendant que le pipeline ingère. En Architecture 3, la replica dédiée aux lectures résout cette contention.

Partie 8 — Sécurité et gouvernance des pipelines IoT

La sécurité d'un pipeline IoT est souvent traitée comme une couche ajoutée après le déploiement. C'est une erreur structurelle. Un broker MQTT compromis peut injecter de fausses mesures. Une base de données exposée peut exfiltrer des données industrielles ou de positionnement. Cette partie traite la sécurité comme une exigence d'architecture — pas un paramètre à régler en fin de projet.

8.1 — Quatre surfaces d'attaque d'un pipeline IoT géolocalisé

Surface 1	**Les devices :** physiquement accessibles en zones non surveillées. Menaces : remplacement physique, extraction des certificats stockés en clair, injection de données falsifiées via un device compromis.
Surface 2	**Le canal MQTT :** transport entre devices et broker. Menaces : interception si TLS absent, usurpation d'identité si authentification faible (connexion anonyme, mot de passe par défaut), rejeu de messages.
Surface 3	**Le broker et le pipeline :** concentrateur de tous les flux IoT. Menaces : accès non autorisé à l'interface d'administration, déni de service (flood de connexions), injection de messages via un client malveillant.
Surface 4	**Base de données et visualisation :** actifs les plus sensibles. Menaces : port PostgreSQL exposé publiquement, injection SQL via requêtes Grafana mal paramétrées, accès Grafana en écriture non contrôlé.

8.2 — TLS, mTLS et ACL : les fondations de sécurité

TLS : toute connexion MQTT en production doit utiliser TLS (port 8883). TLS 1.3 recommandé — TLS 1.0 et 1.1 dépréciés (RFC 8996). Let's Encrypt fournit des certificats valides gratuitement via ACME — la rotation automatique (certbot, Caddy) est indispensable.

CONFIG — MOSQUITTO MTLS + ACL MOINDRE PRIVILÈGE

```
# mosquitto.conf — mTLS complet
listener 8883
cafile   /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/server.crt
keyfile  /etc/mosquitto/certs/server.key
tls_version  tlsv1.3
require_certificate true
use_identity_as_username true   # Common Name du cert = username pour les ACL

# acl_file — moindre privilège par device
pattern write devices/%u/telemetry   # chaque device publie sur son propre topic
pattern read  devices/%u/commands    # et souscrit à ses propres commandes
user pipeline-service
topic read  devices/#
topic write alerts/#

# PostgreSQL — rôle lecture seule pour Grafana
CREATE ROLE grafana_reader WITH LOGIN PASSWORD '...';
GRANT CONNECT ON DATABASE iotdb TO grafana_reader;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO grafana_reader;
```

Règle absolue — PostgreSQL

Le port PostgreSQL (5432) ne doit jamais être exposé publiquement. PostgreSQL écoute sur `localhost` ou un réseau privé uniquement. Aucune ligne `host all all 0.0.0.0/0` dans `pg_hba.conf`. L'accès depuis Grafana, l'API REST et le pipeline se fait en réseau interne (VPN, réseau Docker privé, VPC privé).

8.3 — Cadre réglementaire : NIS2, CRA et RGPD IoT

RÈGLEMENT	PÉRIMÈTRE	OBLIGATIONS CLÉS POUR L'IOT	ÉCHÉANCE
NIS2	Entités essentielles (>250 sal. ou >50 M€) et importantes (>50 sal.) dans 18 secteurs	Politique de gestion des risques, mesures de sécurité techniques, notification incidents ANSSI < 24h (alerte) / 72h (rapport complet)	Transposition FR en cours (avril 2026)
Cyber Resilience Act	Fabricants de produits connectés commerciaux	Sécurité by design, mises à jour de sécurité, signalement des vulnérabilités exploitées activement	Obligations principales : 11 déc. 2027. Signalement vulnérabilités : 11 sept. 2026
RGPD	Données GPS de personnes physiques identifiables (livreurs, techniciens terrain)	Base légale explicite, information de la personne, durée de conservation limitée, droit d'accès et d'effacement	En vigueur

Recommandation pratique NIS2

Vérifier en amont si l'entité porteuse est dans le périmètre NIS2 via le portail MonEspaceNIS2 (monespacenis2.cyber.gouv.fr). Les mesures de sécurité décrites dans cette partie — TLS, mTLS, ACL, isolation réseau PostgreSQL, rôles à privilèges minimaux — constituent une base de conformité solide, quel que soit le calendrier définitif de la transposition française.

Partie 9 — Tendances 2027–2030 : ce qui va changer

L'architecture IoT géolocalisé de 2026 est stable et mature. Les outils documentés dans ce livre blanc sont tous activement maintenus. Mais plusieurs tendances de fond vont transformer progressivement la façon dont les pipelines IoT sont conçus et opérés d'ici 2030. Cette partie n'expose pas des certitudes — elle identifie les évolutions les plus documentées et les plus susceptibles d'avoir un impact concret sur les choix d'architecture.

ÉVOLUTION	HORIZON PROBABLE	IMPACT ARCHITECTURE	MATURITÉ 2026
WASM à l'edge	2027–2029	Traitement distribué sans recompilation dédiée (ARM, x86, RISC-V)	Émergente
Inférence ML edge (TFLite, ONNX)	2026–2028	Détection anomalies embarquée, autonomie sans réseau	En développement
MQTT over QUIC	2027–2028	Meilleure résilience mobile, 0-RTT handshake (EMQX)	En développement
GeoParquet + Apache Iceberg cold	2027–2028	Tables versionnées sur R2, time travel, ACID sur stockage objet	Bêta (R2 Data Catalog)
DuckDB spatial analytique	Disponible maintenant	Analyse cold sans serveur, GeoParquet distants requêtables (v0.10+)	Stable
LoRaWAN 2.4 GHz	2027–2028	Débits supérieurs (~250 kbit/s), couverture mondiale, pas de duty cycle réglementaire	Émergente
MCP + IoT	2027–2029	Requêtes en langage naturel sur TimescaleDB / brokers MQTT via protocole Anthropic	Émergente
ACME pour PKI IoT interne	2026–2027	Rotation automatique des certifs devices (step-ca / Smallstep)	En développement
Cyber Resilience Act	Déc. 2027+	Qualification obligatoire des devices commerciaux connectés	Adoption progressive

Conclusion — Ce que ce livre blanc a construit

Ce document a suivi la chaîne IoT géolocalisé dans l'ordre, du capteur au tableau de bord. Neuf parties, une architecture complète, une logique cohérente. Cinq principes structurent ce parcours.

1 Partir du besoin, pas du catalogue éditeur

- ✓ La première question n'est pas "quel broker choisir ?"
- ✓ mais "combien de devices, à quelle fréquence, pour quel usage ?"
- ✓ La réponse détermine l'architecture — le catalogue vient ensuite

2 Dimensionner pour la réalité

- ✓ "On pourrait avoir 10 000 devices" ne justifie pas une Architecture 3
- ✓ Une Architecture 1 bien conçue est extensible
- ✓ Commencer simple, mesurer, puis évoluer

3 Séparer ingestion, stockage et lecture

- ✓ Pipeline rapide et résilient pour l'ingestion
- ✓ TimescaleDB primary pour le stockage transactionnel
- ✓ Replica ou Continuous Aggregates pour Grafana / Martin / API

4 La politique de rétention avant le déploiement

- ✓ Compression, downsampling, archivage cold sur R2
- ✓ Se configure une fois, en amont, gère automatiquement le volume
- ✓ Ignorer = 500 Go de données brutes non agrégées en 6 mois

> **La stack open source couvre l'intégralité des besoins opérationnels. Mosquitto, EMQX, Node-RED, TimescaleDB, PostGIS, Martin, Grafana, MapLibre GL JS — tous actifs, maintenus, documentés. Une Architecture 2 complète tient sur deux VPS pour 40 à 60 euros par mois. Aucune licence commerciale requise pour les déploiements de taille petite à moyenne.**
>
> — Mattieu Pottier, MP-i · Mattieu Pottier Indépendant

Références

Protocoles et brokers

MQTT v5.0 — Spécification OASIS : docs.oasis-open.org (https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html)

Eclipse Mosquitto : mosquitto.org (https://mosquitto.org)

EMQX v5 : docs.emqx.com (https://docs.emqx.com/en/emqx/latest/)

Node-RED : nodered.org/docs (https://nodered.org/docs/)

RFC 8996 — Deprecating TLS 1.0 and 1.1, mars 2021 : rfc-editor.org (https://www.rfc-editor.org/rfc/rfc8996)

RFC 9000 — QUIC, mai 2021 : rfc-editor.org (https://www.rfc-editor.org/rfc/rfc9000)

ChirpStack v4 (Network Server LoRaWAN) : chirpstack.io/docs (https://www.chirpstack.io/docs/)

Stockage et bases de données

TimescaleDB : docs.timescale.com (https://docs.timescale.com)

PostGIS : postgis.net/docs (https://postgis.net/docs/)

DuckDB spatial : duckdb.org (https://duckdb.org/docs/extensions/spatial/overview)

GeoParquet — Spécification 1.0 : geoparquet.org (https://geoparquet.org/)

h3-pg — Extension H3 pour PostgreSQL : github.com/zachasme/h3-pg (https://github.com/zachasme/h3-pg)

Visualisation

Grafana : grafana.com/docs (https://grafana.com/docs/)

MapLibre GL JS : maplibre.org (https://maplibre.org/maplibre-gl-js/docs/)

Martin — Serveur MVT Rust : maplibre.org/martin (https://maplibre.org/martin/)

Kepler.gl : docs.kepler.gl (https://docs.kepler.gl/)

H3 — Grille hexagonale Uber : h3geo.org (https://h3geo.org/)

Sécurité et réglementation

NIS2 (UE) 2022/2555 : eur-lex.europa.eu (https://eur-lex.europa.eu/legal-content/FR/TXT/?uri=CELEX:32022L2555)

MonEspaceNIS2 — ANSSI : monespacenis2.cyber.gouv.fr (https://monespacenis2.cyber.gouv.fr)

Cyber Resilience Act (UE) 2024/2847 : eur-lex.europa.eu (https://eur-lex.europa.eu/legal-content/FR/TXT/?uri=CELEX:32024R2847)

step-ca — PKI IoT interne : smallstep.com/docs/step-ca (https://smallstep.com/docs/step-ca/)

Glossaire

ACID

Atomicity, Consistency, Isolation, Durability. Les quatre propriétés garantissant la fiabilité des transactions en base de données relationnelle. PostgreSQL/TimescaleDB est conforme ACID.

ACL (Access Control List)

Mécanisme de contrôle d'accès par liste de règles. Dans Mosquitto, les ACL définissent quels clients peuvent publier ou souscrire sur quels topics. La variable `%u` est remplacée par le username du client authentifié.

ACME

Automatic Certificate Management Environment (RFC 8555). Protocole automatisant l'obtention et le renouvellement de certificats TLS. Utilisé par Let's Encrypt. Adaptable pour la PKI IoT interne via step-ca (Smallstep).

AMQP

Advanced Message Queuing Protocol. Protocole de messagerie enterprise (OASIS) : files persistantes, routage conditionnel, transactions. Moins adapté que MQTT pour les devices IoT contraints. Implémenté par RabbitMQ.

Broker MQTT

Serveur central qui reçoit les messages des publishers et les distribue aux subscribers. Implémentations de référence : Mosquitto (EPL/EDL, open source), EMQX (BSL 1.1, nœud unique gratuit).

Chunk (TimescaleDB)

Partition physique d'une hypertable, organisée par intervalle de temps. TimescaleDB crée automatiquement de nouveaux chunks à chaque nouvel intervalle. La compression et la rétention s'appliquent chunk par chunk.

Continuous Aggregate

Vue matérialisée incrémentale gérée automatiquement par TimescaleDB. Précalcule des agrégats sur des fenêtres temporelles définies, se maintient en arrière-plan à mesure que de nouvelles données arrivent. Clé du downsampling.

CRA (Cyber Resilience Act)

Règlement (UE) 2024/2847 imposant des obligations de sécurité aux fabricants de produits connectés. Entré en vigueur le 10 décembre 2024. Obligations principales applicables à partir du 11 décembre 2027.

CRL (Certificate Revocation List)

Liste de certificats X.509 révoqués, publiée par l'autorité de certification. Sur Mosquitto, le fichier CRL est rechargé via `kill -HUP` sans redémarrage du broker.

DLQ (Dead Letter Queue)

File de messages ayant échoué à être traités. Permet d'inspecter et rejouer les messages problématiques sans les perdre. Non natif MQTT — implémenté au niveau du pipeline de traitement ou via les règles de routage EMQX.

DuckDB

Moteur analytique SQL embarqué (sans serveur). Depuis la v0.10 (mars 2024), inclut une extension spatiale permettant les requêtes géospatiales sur GeoParquet distants. Idéal pour l'analyse cold sans infrastructure dédiée.

Edge Device

Dispositif intermédiaire entre les capteurs et le cloud. Capable de pré-traiter, agréger et mettre en mémoire tampon les données avant transmission. Exemples : Raspberry Pi, passerelles industrielles, modules ESP32.

EMQX

Broker MQTT haute performance (EMQ Technologies). Sous BSL 1.1 depuis la v5.9.0 (mai 2025) — nœud unique gratuit en production, cluster multi-nœuds = licence commerciale. Versions ≤ 5.8.x sous Apache 2.0 avec clustering gratuit.

GeoJSON

Format d'échange de données géospatiales basé sur JSON (RFC 7946, 2016). Simple et nativement supporté par JavaScript. Limité en production au-delà de ~10 000 entités simultanées côté client.

GeoParquet

Spécification de stockage de données vectorielles géospatiales dans Apache Parquet. Version stable 1.0 publiée en 2023. Format de référence pour l'archivage cold géospatial cloud-native.

H3

Système d'indexation géospatiale hexagonale à plusieurs résolutions (Uber, open source). Permet l'agrégation spatiale efficace — heatmaps, densité. Extension PostgreSQL : h3-pg v4.

HDOP

Horizontal Dilution of Precision. Indicateur de précision horizontale des données GPS. HDOP < 2 : excellent. HDOP 2–5 : bon. HDOP > 5 : dégradé. À stocker avec chaque position pour le filtrage qualité.

Hypertable (TimescaleDB)

Table PostgreSQL partitionnée automatiquement en chunks par dimension temporelle. Créée via `SELECT create_hypertable(...)`. Supporte nativement la compression, les Continuous Aggregates et les politiques de rétention.

Index GiST

Generalized Search Tree. Framework d'index extensible de PostgreSQL, utilisé par PostGIS pour les index spatiaux R-Tree. Permet les requêtes spatiales rapides. Commande : `CREATE INDEX ON table USING GIST(geom)`.

IoT / IIoT

Internet of Things / Industrial IoT. Ce livre blanc couvre l'IIoT : capteurs industriels, environnementaux et de positionnement. Exclut domotique grand public et wearables.

Martin

Serveur de tuiles vectorielles MVT open source écrit en Rust (licence MIT). Se connecte directement à PostGIS et expose les données géospatiales via endpoint `/tiles/{z}/{x}/{y}`. Version 1.0 publiée en novembre 2025.

MCP (Model Context Protocol)

Protocole développé par Anthropic (novembre 2024) standardisant l'accès des applications d'IA à des sources de données et outils externes. Des serveurs MCP pour TimescaleDB et brokers MQTT émergent en 2026.

Mosquitto

Broker MQTT open source (Eclipse Foundation, EPL/EDL). Très léger (< 1 Mo en mémoire). Architecture mono-processus, pas de clustering natif. Supporte MQTT v3.1, v3.1.1 et v5.0.

MQTT

Message Queuing Telemetry Transport. Protocole publish/subscribe optimisé pour les réseaux à faible bande passante. Standard OASIS / ISO (ISO/IEC 20922). v3.1.1 : très répandu. v5.0 : publié 7 mars 2019.

mTLS (Mutual TLS)

Authentification TLS mutuelle : le broker vérifie le certificat client, le client vérifie le certificat serveur. Mécanisme le plus robuste pour les devices IoT — pas de mot de passe stocké sur le device, révocation granulaire.

MVT (Mapbox Vector Tile)

Format binaire de tuiles vectorielles basé sur Protocol Buffers, organisé en pyramide XYZ. Standard de facto du web cartographique. Affichage fluide de millions d'entités via rendu WebGL côté client.

NB-IoT

Narrowband IoT. Protocole LPWAN cellulaire (3GPP Release 13, 2016). Très faible consommation, excellente pénétration indoor, couverture opérateur nécessaire. Adapté aux capteurs statiques en zone urbaine couverte.

NIS2

Directive (UE) 2022/2555 sur la sécurité des réseaux et systèmes d'information. Oblige les entités essentielles et importantes dans 18 secteurs à mettre en place des mesures de cybersécurité. Transposition française en cours (avril 2026).

OIV

Opérateur d'Importance Vitale. Désignation française (LPM 2014) attribuée par l'ANSSI. Obligations de sécurité renforcées, articulées avec NIS2 pour les entités concernées.

PostGIS

Extension spatiale pour PostgreSQL. Version stable 2026 : 3.6.x. Ajoute les types géométriques (`GEOMETRY`, `GEOGRAPHY`), les index GiST spatiaux, et 800+ fonctions géospatiales.

QoS MQTT

Quality of Service. Niveau de garantie de délivrance : QoS 0 (at-most-once), QoS 1 (at-least-once, doublons possibles), QoS 2 (exactly-once, handshake 4 étapes). Impact sur le débit et la consommation.

RGPD

Règlement Général sur la Protection des Données (UE) 2016/679. S'applique aux données de positionnement d'une personne physique identifiable. Implique base légale, information, durée de conservation limitée, droit d'accès/effacement.

Tiering (stockage)

Stratégie de stockage multi-niveaux. Hot (0–7 jours, TimescaleDB brut SSD), Warm (7j–6 mois, Continuous Aggregates compressés), Cold (> 6 mois, GeoParquet sur Cloudflare R2 à 0,015 $/Go/mois).

TimescaleDB

Extension PostgreSQL open source (Apache 2.0) optimisée pour les séries temporelles. Hypertables, compression native (90–95 %), Continuous Aggregates, politiques de rétention. Première version stable : novembre 2018.

TLS

Transport Layer Security. Protocole de chiffrement du transport. TLS 1.2 minimum acceptable en 2026, TLS 1.3 recommandé. TLS 1.0 et 1.1 dépréciés (RFC 8996, mars 2021). Port MQTT standard TLS : 8883.

Topic MQTT

Chaîne hiérarchique d'adressage MQTT. Format recommandé : `devices/{device_id}/telemetry`. Séparateur `/`, wildcard un niveau `+`, wildcard multi-niveaux `#`.

TSDB

Time Series Database. Base de données optimisée pour les données horodatées : append-only, compression temporelle, downsampling natif, politique de rétention automatique. Exemple phare : TimescaleDB.

Mattieu Pottier

Consultant SI indépendant — MPI Ingénierie & Numérique

Expert SIG, architecture système et Odoo. J'aide les collectivités, PME et bureaux d'études à construire des systèmes d'information calibrés sur leurs besoins réels — pipelines IoT, infrastructures géospatiales, intégration ERP. Sans surcoût ni dépendance éditeur.