MP-i

COLLECTION « CHAÎNE DE LA DONNÉE GÉOSPATIALE » — VOLUME 2

LIVRE BLANC TECHNIQUE — N°2

Stocker la donnée géospatiale (1970–2026)

De ESRI Shapefile à GeoParquet Cloud-Native — Fichiers, bases de données, formats analytiques et architectures

Ce livre blanc couvre l'évolution complète des paradigmes de stockage géospatial sur 55 ans : du fichier propriétaire ESRI aux lacs de données columnar cloud-native. Il propose une taxonomie en cinq familles, une analyse technique par ère, un guide décisionnel complet avec matrice d'usage, arbre de décision, comparatif de coûts réels et quatre architectures de référence pour 2026.

LB1 — Les Formats › LB2 — Le Stockage ← › LB3 — L'Exposition

Mattieu Pottier

Consultant en transformation digitale — SIG, géomatique, Web, ERP Odoo

MP-i — Mattieu Pottier Indépendant · mp-i.pro

Version 1.0.3 — Mars 2026

Document distribué librement

On choisit un format avec soin, on construit une carte, on expose une API — et la question du stockage arrive en dernier, comme une contrainte technique à régler rapidement. C'est une erreur. Le *où stocker* n'est pas un détail d'infrastructure. C'est une décision architecturale qui conditionne les performances de lecture, la capacité d'analyse, la scalabilité, les coûts d'exploitation et la facilité d'exposition. Ce deuxième volume de la collection « Chaîne de la donnée géospatiale » couvre 55 ans d'évolution — du fichier propriétaire ESRI aux lacs de données columnar cloud-native — avec une taxonomie en cinq familles, des benchmarks réels et quatre architectures de référence pour 2026.

11
parties thématiques

5
familles de stockage couvertes

4
architectures de référence 2026

1970
— 2026 couvert

Collection « Chaîne de la donnée géospatiale » — Volume 2

LB1 — LE QUOI — Formats Géospatiaux 2026 — Structure, histoire, critères de choix.
LB2 — LE OÙ ← CE DOCUMENT — Stockage Géospatial — Moteurs, index, coûts, gouvernance.
LB3 — LE COMMENT — Exposition Web — Servir et consommer la donnée.
Ce document suppose la lecture de LB1. Il est le prérequis de LB3.

SOMMAIRE

Préface

La donnée géospatiale pose une question que l'on reporte trop souvent : *où la stocker ?* On choisit un format, on construit une carte, on expose une API — et la question du stockage arrive en dernier, comme une contrainte technique à régler rapidement. C'est une erreur.

Ce deuxième livre blanc de la série MP-i s'attaque à cette question centrale. Le *où stocker* n'est pas un détail d'infrastructure. C'est une décision architecturale qui conditionne les performances de lecture, la capacité d'analyse, la scalabilité, les coûts d'exploitation et la facilité d'exposition — tout ce qui vient ensuite.

Ce document est le maillon central d'une trilogie :

Livre Blanc Formats Géospatiaux 2026 (le QUOI) : la structure des formats géospatiaux, leur histoire, leurs critères de choix. La question des *contenants* de la donnée.

LB2 — Stockage (le OÙ) : ce document. Les moteurs, les index, les coûts, la gouvernance. La question de *l'endroit* où la donnée réside et comment on l'interroge.

LB3 — Exposition (le COMMENT) : servir et consommer la donnée sur le web. La question des *interfaces* entre le stockage et les clients.

> → **Règle de lecture**
>
> Chaque format déjà traité dans le *Livre Blanc Formats Géospatiaux 2026* (Shapefile, GeoTIFF, GeoPackage, FlatGeobuf, COG, GeoParquet, PMTiles) est abordé ici exclusivement sous l'angle du stockage — comportement moteur, index, coûts, maintenance. La structure du format n'est pas répétée.

Mattieu Pottier — Consultant en transformation digitale, spécialiste SIG et applications géospatiales

Introduction — Pourquoi le stockage géospatial est un sujet stratégique

Stocker de la donnée géospatiale n'est pas stocker de la donnée ordinaire. Une table de 10 millions de polygones de parcelles cadastrales ne se comporte pas comme une table de 10 millions de lignes de facturation. Les géométries sont des objets complexes, variables en taille, qui nécessitent des index spécialisés, des opérateurs dédiés et des moteurs capables de répondre à des questions fondamentalement différentes des requêtes attributaires classiques : *quelles entités intersectent cette zone ? quels points sont à moins de 500 mètres de cette route ? quelle est la superficie de l'union de ces polygones ?*

Choisir le mauvais moteur de stockage, c'est potentiellement :

- Des performances dégradées — un scan complet d'un GeoJSON de 2 Go pour trouver les entités dans une bbox, là où un index spatial GiST sur PostGIS répondrait en quelques millisecondes
- Une scalabilité impossible — un fichier GeoPackage verrouillé par un utilisateur QGIS pendant qu'un script Python tente d'écrire dessus
- Un coût d'infrastructure disproportionné — un stack PostGIS + tile server (Martin, pg_tileserv) pour servir des données qui changent une fois par mois, là où PMTiles sur un CDN suffirait pour dix fois moins cher
- Une dette technique — un Shapefile au cœur d'un pipeline de données moderne, avec sa limite de 2 Go, ses noms de colonnes tronqués à 10 caractères et son absence de transactions

Le fil conducteur : quatre ruptures technologiques

FIGURE 1 — LES QUATRE RUPTURES TECHNOLOGIQUES DU STOCKAGE GÉOSPATIAL (1970–2026)

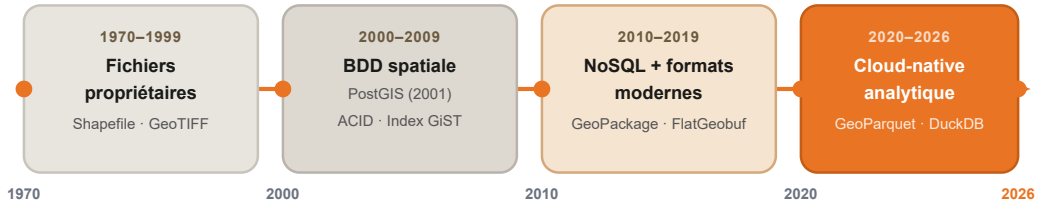

FIGURE 1 — LES QUATRE RUPTURES TECHNOLOGIQUES DU STOCKAGE GÉOSPATIAL (1970–2026)

PÉRIODE	RUPTURE	CE QUI CHANGE
1970–1999	Le fichier comme paradigme	La donnée géospatiale existe dans des fichiers propriétaires ou semi-ouverts. Pas de requête sans chargement complet. Pas de concurrence.
2000–2009	La base de données spatiale	SQL devient géographique. PostGIS (2001) donne à PostgreSQL des capacités spatiales complètes. La donnée devient requêtable, indexée, transactionnelle.
2010–2019	Le NoSQL et le cloud	De nouveaux paradigmes émergent : documents JSON géolocalisés (MongoDB), recherche spatiale (Elasticsearch), formats de fichier optimisés (GeoPackage, FlatGeobuf).
2020–2026	**Le cloud-native analytique**	**GeoParquet, COG, DuckDB et le stockage objet (S3/R2) redéfinissent ce que signifie « stocker » de la donnée géospatiale à grande échelle sans infrastructure dédiée.**

Les trois questions fondamentales

Avant de choisir un moteur de stockage, trois questions doivent être posées :

1 Quelle volumétrie ?

- ✓ Quelques milliers d'entités ou des milliards ?
- ✓ Un fichier de 10 Mo ou un lac de données de 500 Go ?
- ✓ La réponse oriente immédiatement vers des familles de solutions différentes

2 Quel usage dominant ?

- ✓ Édition interactive multi-utilisateurs ?
- ✓ Analyse spatiale à la volée ou pipeline batch ?
- ✓ Un même jeu de données peut nécessiter plusieurs moteurs selon les usages

3 Quelle infrastructure ?

- ✓ Serveur dédié auto-géré ou cloud managé ?
- ✓ Stockage objet serverless ou zéro infrastructure ?
- ✓ La contrainte budgétaire est souvent le premier filtre décisionnel

Comment lire ce document

Lecture linéaire — recommandée pour comprendre l'évolution des paradigmes de stockage géospatial et les raisons historiques des choix actuels. Suivre les parties 1 à 11 dans l'ordre.

Lecture orientée décision — pour un projet en cours, aller directement à la Partie 9 (guide décisionnel) et aux architectures de référence (Partie 10), puis remonter vers les parties techniques

selon les moteurs retenus.

Partie 1 — Taxonomie des modes de stockage géospatial

Avant d'entrer dans l'histoire et les détails techniques, il est utile de disposer d'une grille de lecture. Le paysage du stockage géospatial est vaste et hétérogène — des fichiers plats hérités des années 1990 aux lacs de données columnar cloud-native de 2026. Cinq grandes familles structurent cet espace.

1.1 Les cinq grandes familles

FAMILLE 1

Fichier vecteur plat

Sans serveur · Fichier unique

Format autodescriptif sur système de fichiers. Pas de mécanisme de requête interne — charger en mémoire pour interroger.

Shapefile · GeoJSON · GeoPackage · FlatGeobuf

FAMILLE 2

Fichier raster

Grille de valeurs · Référence spatiale

Grille de valeurs numériques avec référence spatiale. Performances gérées par pyramides, compression et tiling interne.

GeoTIFF · COG · MBTiles · NetCDF

FAMILLE 3

BDD relationnelle spatiale

Transactions ACID · Index GiST / R-Tree

Extension spatiale sur SGBD relationnel. Types géométriques natifs, index spatiaux, fonctions d'analyse via SQL standard.

PostGIS · SpatiaLite · Oracle Spatial · SQL Server

FAMILLE 4

NoSQL géospatial

Documents, IoT · Recherche hybride

Bases non relationnelles avec capacités géospatiales. Orientées cas d'usage spécifiques : IoT, recherche full-text, cache de proximité.

MongoDB · Elasticsearch · Redis GEO · TimescaleDB

FAMILLE 5

Analytique cloud-native

Columnar · S3/R2 · Serverless

Formats columnar partitionnés pour la lecture analytique à grande échelle sur stockage objet. Requêtables sans serveur dédié.

GeoParquet · DuckDB · Delta Lake · BigQuery Geo

Figure 2 — Les cinq familles de stockage géospatial

1.2 Les 5 critères de choix d'un mode de stockage

CRITÈRE	SEUILS / RÈGLES	SOLUTIONS INDICATIVES
Volume	< 500 Mo → fichier plat · 500 Mo–10 Go → BDD · > 10 Go → analytique ou BDD partitionnée	GeoPackage · PostGIS · GeoParquet
Usage dominant	Édition interactive → transactions, concurrence · Analytique → columnar · Exposition statique → CDN	PostGIS · GeoParquet + DuckDB · PMTiles
Requêtabilité	Filtres attributaires → tout moteur · Intersections → PostGIS/DuckDB · Full-text + spatial → Elasticsearch · Agrégats massifs → GeoParquet + Sedona	PostGIS · Elasticsearch · Spark
Infrastructure	Serveur dédié → PostGIS · Cloud managé → Cloud SQL/AlloyDB/RDS · Zéro infra → DuckDB + GeoParquet S3/R2	PostGIS · AlloyDB · DuckDB
Interopérabilité	PostGIS → universel SIG · GeoParquet → GDAL 3.5+, GeoPandas, DuckDB, Sedona · SpatiaLite → QGIS + GDAL	PostGIS · GeoParquet · SpatiaLite

1.3 Matrice synthétique famille × critères

ℹ️ ✅ = natif / optimal · ⚠️ = partiel / conditionnel · ❌ = absent / incompatible

FAMILLE	VOL. MAX RECO.	REQUÊTE SPATIALE	CONC.	COÛT INFRA	INTEROP.
Fichier vecteur plat	~ 500 Mo	⚠️ Scan complet (sauf FlatGeobuf)	❌	Nul	✅ Universelle
Fichier raster	Illimité (COG)	⚠️ Partielle (HTTP Range / COG)	❌	Nul–faible	✅ Bonne
BDD relationnelle spatiale	Multi-Go (> 100M avec index)	✅ Native + index GiST	✅	Moyen–élevé	✅ Excellente
NoSQL géospatial	Multi-Go	⚠️ Partielle (points + géom. simples)	✅	Moyen–élevé	⚠️ Variable
Analytique cloud-native	Illimité	✅ Columnar (scan massif)	❌ lecture seule	Très faible	✅ Croissante

Partie 2 — Ère 1 : Le fichier comme paradigme (1970–1999)

> 🗂️ **1970–1999 — Paradigme fichier**
>
> Toute la donnée géographique vit dans des formats propriétaires ou semi-ouverts, conçus pour des logiciels SIG desktop spécifiques. Cette ère a posé des habitudes de travail qui persistent encore en 2026 — pour le meilleur et surtout pour le pire.

Shapefile (1993 — ESRI)

→ Architecture du format détaillée dans le *Livre Blanc Formats Géospatiaux 2026*. Ce chapitre traite exclusivement des implications de stockage.

Le Shapefile est introduit par ESRI en 1993 avec ArcView GIS 2. Conçu comme format d'interopérabilité entre produits ESRI, il devient le format d'échange universel de la géomatique par défaut d'inertie, faute de concurrent ouvert viable pendant plus d'une décennie.

Problèmes de stockage spécifiques au Shapefile

LIMITATION	IMPACT OPÉRATIONNEL
Architecture multi-fichiers (.shp, .dbf, .shx, .prj…)	Risque de corruption par copie partielle, versionnement incomplet, archivage incomplet
Limite de 2 Go par fichier .shp (architecture 32 bits, 1993)	Fragmentation artificielle imposée pour les données nationales ou LiDAR
Pas de transactions	Écriture interrompue = état incohérent sans rollback possible
Noms de colonnes tronqués à 10 caractères	Perte de sémantique, conflits de noms, nettoyage obligatoire à l'import
Un seul type géométrique par fichier	Multiplication des fichiers pour les données géométriques mixtes
Encodage texte non standardisé	Problèmes récurrents d'accents et caractères spéciaux en échange inter-systèmes

⚠️ **Statut 2026 : migration recommandée**

Le Shapefile reste omniprésent dans les administrations et les échanges entre systèmes legacy. Mais pour tout nouveau projet, **GeoPackage** constitue le remplacement naturel : même philosophie de fichier autodescriptif, sans aucune des limitations du Shapefile.

Coverage ESRI et formats propriétaires historiques

Avant le Shapefile, ESRI utilisait le format **Coverage** (ARC/INFO, années 1980) — une structure de répertoire complexe stockant géométries, topologie et tables attributaires dans une arborescence propriétaire. Il encodait la topologie explicitement (relations entre arcs, nœuds, polygones), puissant pour l'analyse topologique mais opaque pour l'interopérabilité.

Les formats **DXF** (AutoCAD, 1982) et **DGN** (MicroStation, Intergraph / Bentley) viennent du monde de la CAO et ont été massivement utilisés pour stocker des données géographiques dans les collectivités et bureaux d'études. Ils partagent les mêmes limitations : pas de projection standardisée, pas d'attributs structurés, sémantique géométrique CAO incompatible avec les SIG.

⚠️ Leur présence dans les systèmes actuels est le signe d'une dette technique à migrer. GDAL/OGR peut les lire en lecture seule uniquement.

GeoTIFF (1994–1995) — le raster de référence

→ Architecture du format détaillée dans le *Livre Blanc Formats Géospatiaux 2026*. Ce chapitre traite exclusivement du stockage.

La spécification GeoTIFF est rédigée par **Niles Ritter** (NASA Jet Propulsion Laboratory) et **Mike Ruth** (SPOT Image Corp). La version **1.0 officielle** est publiée en **novembre 1995**. Il étend le format TIFF en ajoutant des balises géoréférencées (système de coordonnées, transformation affine) dans les métadonnées du fichier.

Sans pyramides (*overviews*), tout accès à une vue dézoomée oblige à lire l'intégralité du fichier. Les overviews sont des versions sous-échantillonnées précalculées — internes au fichier ou dans un fichier `.ovr` séparé. Le tiling interne (`TILED=YES` dans GDAL) organise les données en blocs carrés (256×256 ou 512×512 px), réduisant les lectures pour les accès à une zone géographique spécifique.

La limite fondamentale

Un GeoTIFF classique n'est pas conçu pour être lu partiellement depuis un stockage distant. Accéder à une bbox sur un GeoTIFF hébergé sur S3 implique de télécharger le fichier complet. C'est précisément ce que résout le **COG** (Cloud Optimized GeoTIFF), traité en Partie 6.

Le problème fondamental du fichier plat

LIMITATION STRUCTURELLE	CONSÉQUENCE
Pas de concurrence d'accès	Écriture = blocage de tous les lecteurs ; corruption silencieuse sur NAS réseau
Pas d'index spatial natif (hors FlatGeobuf)	Toute requête spatiale = scan complet en mémoire
Pas de requête sans chargement complet	GDAL parcourt le fichier entièrement même pour un filtre spatial
Pas de transactions	Écriture partielle = état incohérent non détectable automatiquement

Partie 3 — Ère 2 : La base de données spatiale (2000–2009)

2000–2009 — BDD Spatiale

L'arrivée des extensions spatiales dans les SGBD relationnels constitue la rupture technologique la plus importante de l'histoire du stockage géospatial : requêtes indexées, transactions, concurrence, fonctions d'analyse intégrées — en une seule transition.

PostGIS (2001 — Refractions Research)

→ PostGIS est présenté dans le *Livre Blanc Formats Géospatiaux 2026* comme socle d'architecture. Ce chapitre traite le moteur de stockage en profondeur.

PostGIS est créé par **Refractions Research** (Victoria, Colombie-Britannique, Canada). Première version publiée en **mai 2001** par **Dave Blasby**, avec Paul Ramsey comme co-fondateur. Version stable **1.0** : 19 avril 2005. Version actuelle 2026 : **PostGIS 3.5.x**, maintenu par la communauté OSGeo.

Types géométriques et géographiques

`GEOMETRY`	Coordonnées planes, calculs euclidiens. Adapté aux données projetées (Lambert 93, UTM). Distances et superficies en unités de la projection.
`GEOGRAPHY`	Coordonnées sphériques (longitude/latitude). Calculs géodésiques sur la surface de la Terre. Adapté aux données mondiales et aux distances précises sur grandes distances. Plus coûteux en calcul que GEOMETRY.
`RASTER`	Extension postgis_raster — stockage raster dans PostgreSQL. En déclin en 2026 au profit du COG sur stockage objet (voir Partie 6–7).

Index spatial : GiST, SP-GiST, BRIN

INDEX	STRUCTURE	USAGE RECOMMANDÉ
`GiST`	R-Tree sur bounding boxes	✅ Choix par défaut — ST_Intersects, ST_DWithin, ST_Contains
`SP-GiST`	Partitionnement quadtree / kd-tree	Distributions très non-uniformes (zones très denses)
`BRIN`	Index léger par plages de blocs	Tables très larges avec import séquentiel géographique

> ✅ **Règle pratique**
>
> Toujours créer un index GiST sur les colonnes géométriques d'une table opérationnelle. Un `VACUUM ANALYZE` régulier maintient les statistiques à jour pour l'optimiseur de requêtes.

Fonctions clés PostGIS

FONCTION	USAGE
`ST_Intersects(a, b)`	Test d'intersection — utilise l'index GiST
`ST_Contains(a, b)`	Test de contenance
`ST_DWithin(a, b, distance)`	Entités dans un rayon — très optimisé avec index
`ST_Buffer(geom, distance)`	Zone tampon
`ST_Union(geom)`	Fusion de géométries (agrégat)
`ST_Transform(geom, srid)`	Reprojection à la volée
`ST_AsMVT(row, layer)`	Export Mapbox Vector Tile (exposition web — LB3)
`ST_AsGeoJSON(geom)`	Export GeoJSON
`ST_IsValid(geom)`	Vérification intégrité — à exécuter après import

EXPLAIN ANALYZE sur les requêtes PostGIS

SQL

```
-- Vérifier que l'index GiST est bien utilisé
EXPLAIN ANALYZE
SELECT id, nom
FROM parcelles
WHERE ST_Intersects(geom, ST_MakeEnvelope(2.3, 48.8, 2.4, 48.9, 4326));

-- Un plan optimal montre un Bitmap Index Scan ou Index Scan
-- sur l'index GiST, suivi d'un Recheck Cond.
-- Un Seq Scan sur une grande table = index absent ou désactivé.
```

> **Statut 2026 : référence absolue pour le transactionnel**
>
> PostGIS reste en 2026 la solution de référence pour tout projet nécessitant édition collaborative, requêtes spatiales indexées et intégrité transactionnelle. Sa maturité (25 ans), son écosystème (QGIS, GeoServer, GDAL, MapLibre, toutes les bibliothèques Python géospatiales) et sa performance sur les cas d'usage courants n'ont pas d'équivalent open source.

SpatiaLite (2008)

SpatiaLite est une extension spatiale pour **SQLite**, développée par **Alessandro Furieri** à partir de 2008. Il apporte à SQLite des capacités comparables à PostGIS : types géométriques OGC, index spatial R-Tree natif, fonctions d'analyse géospatiale (basées sur GEOS et PROJ).

Positionnement : le fichier de base de données spatiale transportable. Un fichier `.sqlite` peut être copié, ouvert dans QGIS sur n'importe quelle machine — sans configurer un serveur PostgreSQL. Idéal pour les applications QGIS hors ligne, les échanges entre techniciens SIG sans infrastructure partagée, les bases embarquées.

> ⚠️ **Limite de concurrence**
>
> SQLite utilise un mécanisme de verrouillage fichier : une seule écriture à la fois, toutes les lectures bloquées pendant l'écriture. Acceptable pour un utilisateur unique. Rédhibitoire dès que plusieurs utilisateurs tentent d'écrire simultanément. En 2026, pour les échanges inter-systèmes, GeoPackage est préférable à SpatiaLite.

Oracle Spatial & Graph

Oracle intègre des capacités spatiales depuis **Oracle 7 Spatial Data Option** (milieu des années 1990), avec le composant **Oracle Spatial** stabilisé à partir d'**Oracle 9i** (2001). Il a dominé le marché du SIG d'entreprise pendant les années 2000, porté par la présence Oracle déjà établie dans les grandes infrastructures publiques.

Statut 2026 : Oracle Spatial reste présent dans les grandes structures publiques (ministères, opérateurs d'infrastructures critiques) standardisées Oracle depuis les années 2000. Les nouveaux projets choisissent systématiquement PostGIS. La migration Oracle Spatial → PostGIS est un chantier courant dans les DSI en transformation.

SQL Server Spatial (2008)

Microsoft introduit les types spatiaux dans **SQL Server 2008** avec deux types distincts : `geometry` (coordonnées planes) et `geography` (coordonnées sphériques). SQL Server Spatial est le choix naturel dans les organisations standardisées sur l'écosystème Microsoft (Azure, Power BI, SSRS). Hors cet écosystème, PostGIS reste plus performant, plus complet et sans coût de licence.

PostGIS vs les alternatives relationnelles — comparatif

CRITÈRE	POSTGIS	SPATIALITE	ORACLE SPATIAL	SQL SERVER SPATIAL
Coût	Gratuit (OSS)	Gratuit (OSS)	Licence Oracle	Licence MS
Fonctions spatiales	800+	~300	~400	~200
Concurrence	✅ Complète	⚠️ Limitée	✅	✅
Scalabilité	✅ Multi-Go	⚠️ Quelques Go	✅	✅
Interopérabilité SIG	✅ Excellente	✅ Bonne	⚠️ Correcte	⚠️ Correcte
Cas d'usage dominant	Tout projet SIG	Offline, portable	Legacy entreprise	Stack Microsoft

L'index spatial : le cœur de la performance

Le R-Tree (Rectangle Tree) est la structure d'index spatial dominante. Il organise les géométries dans une hiérarchie de bounding boxes imbriquées. Une requête spatiale parcourt l'arbre depuis la racine, éliminant les branches dont la bounding box n'intersecte pas la zone de recherche.

PostGIS implémente le R-Tree via le **GiST** (Generalized Search Tree), un framework d'index extensible de PostgreSQL. L'index GiST est bidimensionnel par défaut (2D) mais supporte les index nD pour les géométries 3D.

Limite de l'index spatial

L'index GiST travaille sur les **bounding boxes** des géométries, pas sur les géométries elles-mêmes. Une requête `ST_Intersects` commence par filtrer via l'index (phase "index scan" sur les bounding boxes), puis valide les candidats par calcul géométrique exact (phase "recheck"). Pour des géométries complexes (polygones à 10 000 sommets), le recheck peut être coûteux même avec un bon index.

Partie 4 — Formats de fichier modernes (2010–2019)

📦 **2010–2019 — Formats modernes**

GeoPackage, FlatGeobuf, MBTiles et PMTiles partagent une philosophie commune : être utilisables sans infrastructure serveur, tout en offrant des capacités de requête ou d'accès partiel que le Shapefile ne permettait pas.

GeoPackage (2014 — OGC)

→ Architecture du format détaillée dans le *Livre Blanc Formats Géospatiaux 2026*. Ce chapitre traite exclusivement des comportements de stockage.

Le GeoPackage est adopté comme standard OGC en **février 2014** (OGC 12-128r12). Basé sur SQLite — fichier unique contenant une base de données relationnelle complète — avec un schéma standardisé OGC pour stocker vecteur, raster et métadonnées.

Pourquoi GeoPackage remplace Shapefile

LIMITATION SHAPEFILE	SOLUTION GEOPACKAGE
Multi-fichiers (.shp, .dbf, .shx…)	Fichier unique `.gpkg`
Limite 2 Go par fichier	Pas de limite pratique (SQLite ~140 To théorique)
Noms de colonnes tronqués à 10 caractères	Noms illimités (SQLite)
Pas de NULL vrai	NULL natif SQLite
Un seul type géométrique par fichier	Plusieurs couches dans un fichier
Encodage texte problématique	UTF-8 natif
Pas de métadonnées structurées	Tables de métadonnées OGC intégrées

GeoPackage hérite des caractéristiques de stockage de SQLite : index spatial R-Tree (`rtree`), moins performant que GiST PostGIS sur grands volumes mais suffisant pour des millions d'entités en lecture. Concurrence limitée : SQLite utilise un verrou global en écriture (mode WAL améliore légèrement la lecture concurrente).

⚠️ **Limite multi-utilisateurs**

GeoPackage reste un fichier. Un GeoPackage sur un NAS partagé avec plusieurs utilisateurs QGIS écrivant simultanément produit des erreurs de verrouillage. Pour un usage multi-utilisateurs intensif en écriture, PostGIS reste le seul choix.

2026 **Cas d'usage recommandés**

Échange de données entre systèmes SIG · Stockage local dans QGIS (couches de travail, projets offline) · Livraison de données à un client sans infrastructure partagée · Applications terrain (QGIS sur tablette)

FlatGeobuf (2018–2019)

→ Architecture du format et index Hilbert R-Tree détaillés dans le *Livre Blanc Formats Géospatiaux 2026*. Ce chapitre traite des stratégies de stockage sur CDN et objet.

FlatGeobuf est développé par **Björn Harrtell** à partir de **2018–2019**. Il utilise **FlatBuffers** (format de sérialisation binaire Google) pour encoder les géométries et intègre un **index Hilbert R-Tree** pour l'accès spatial sans serveur.

Stockage sur CDN et S3 : l'accès HTTP Range

La caractéristique distinctive de FlatGeobuf est son support natif des **HTTP Range Requests**. Un fichier `.fgb` hébergé sur S3, R2 ou un CDN peut être requêté spatialement sans serveur intermédiaire. Le flux d'accès partiel :

1 **Lecture de l'en-tête**

Quelques Ko — métadonnées et position de l'index dans le fichier.

2 **Lecture de l'index R-Tree**

Uniquement la portion de l'index correspondant à la bbox souhaitée.

3 **Lecture des données**

Uniquement les octets des entités sélectionnées. Fichier statique transformé en ressource quasi-requêtable sans infrastructure serveur.

⚠️ **Pas d'écriture incrémentale**

Toute modification nécessite une régénération complète du fichier (reconstruction de l'index Hilbert R-Tree). Adapté aux données en lecture majoritaire — couches de référence géographique, données de fond de carte à mettre à jour mensuellement.

MBTiles — le tile store en SQLite

MBTiles est une spécification développée par **Mapbox** à partir de **2010** pour stocker des tuiles cartographiques (PNG, WebP, ou MVT) dans une base de données SQLite. Structure simple : une table `tiles` avec colonnes `zoom_level`, `tile_column`, `tile_row` et un blob `tile_data`.

Avantages pour le stockage local : compatible **TileServer GL**, supporté par QGIS pour les fonds de carte offline, utilisé dans les applications mobiles hors ligne (OsmAnd, Organic Maps). Limite pour le cloud : SQLite ne supporte pas les HTTP Range Requests nativement — un serveur (TileServer GL) reste nécessaire pour servir les tuiles HTTP. C'est précisément ce que PMTiles élimine.

PMTiles (2021 — Protomaps, V3 stable 2022)

> → Architecture du format détaillée dans le *Livre Blanc Formats Géospatiaux 2026*. Ce chapitre traite du stockage objet, de la génération et des coûts.

PMTiles est développé par **Brandon Liu** (Protomaps LLC). Première version annoncée en **mars 2021**. Version 3 stable publiée en **octobre 2022**. Il résout le problème central de MBTiles pour le cloud : servir des tuiles depuis un stockage objet sans serveur. Chaque tuile est accessible via une unique requête HTTP Range, permettant à MapLibre GL JS de la consommer directement depuis Cloudflare R2 ou AWS S3 — sans aucun serveur intermédiaire.

Workflow de génération

Tippecanoe Génère des tuiles vectorielles MVT depuis GeoJSON ou FlatGeobuf. Très configurable (simplification, fusion, filtrage par zoom). Commande typique : `tippecanoe -o output.pmtiles -z14 input.geojson`

Planetiler Génère des fonds de carte monde entier depuis OpenStreetMap ou Overture Maps en quelques heures sur un serveur standard. Optimisé pour la génération de fonds de carte complets.

Comparatif coûts de stockage : S3 vs R2

INFRASTRUCTURE	STOCKAGE (GO/MOIS)	EGRESS (GO)	REMARQUE
AWS S3	~0,023 $	~0,09 $	Egress coûteux à grande échelle
Cloudflare R2	~0,015 $	Gratuit	Avantage décisif pour la distribution géospatiale
Scaleway (Paris)	~0,015 €	75 Go offerts/mois	Datacenter EU, conformité RGPD
Hetzner (nbg1)	~0,0059 €	Inclus (1 To/mois)	⚠️ Pas d'API S3 native — proxy Minio requis pour HTTP Range

> ✅ **Quand PMTiles remplace PostGIS + tile server**
>
> Pour les données statiques ou peu fréquemment mises à jour (fond de carte administratif, données de référence annuelles), le pipeline **PostGIS → Tippecanoe → PMTiles → R2** est économiquement et opérationnellement supérieur à un tile server dynamique. La mise à jour implique une régénération complète et un upload — un pipeline automatisable par script.

Partie 5 — Le NoSQL géospatial (2010–2019)

2010–2019 — NoSQL

Les moteurs NoSQL n'ont pas émergé pour remplacer PostGIS. Ils répondent à des besoins complémentaires : documents semi-structurés, ingestion massive, recherche full-text. Leurs capacités géospatiales sont généralement limitées aux points et géométries simples.

MongoDB + GeoJSON

MongoDB (lancé en 2009 par 10gen) intègre des capacités géospatiales significatives à partir de la version **2.4** (mars 2013) avec l'introduction de l'index `2dsphere` et le support natif de GeoJSON. Les types supportés couvrent les géométries OGC Simple Features : `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, `GeometryCollection`.

JSON

```
{
  "_id": ObjectId("..."),
  "nom": "Tour Eiffel",
  "localisation": {
    "type": "Point",
    "coordinates": [2.2945, 48.8584]
  },
  "categorie": "monument"
}
```

Index géospatiaux et opérateurs

STANDARD

Index `2dsphere`

- ✓ Surface sphérique (WGS84)
- ✓ Requêtes géodésiques précises
- ✓ Intersections et containments
- ✓ Choix par défaut — données géographiques réelles

VS

LEGACY

Index `2d`

- → Plan euclidien uniquement
- → Héritage des premières versions
- → Limité aux points
- → À éviter pour les nouveaux projets

OPÉRATEUR	USAGE
`$near`	Documents proches d'un point, triés par distance
`$nearSphere`	Idem, calcul sphérique géodésique
`$geoWithin`	Documents dont la géométrie est dans une zone
`$geoIntersects`	Documents dont la géométrie intersecte une zone

Cas d'usage : données IoT/GPS (ingestion massive, attributs variables par capteur), documents géolocalisés riches (fiches de commerces, POI avec photos et horaires), logs géolocalisés. MongoDB brille sur les cas que PostGIS gère mal ou surdimensionne.

🚫 **Limites structurelles**

Pas d'équivalent de `ST_Buffer`, `ST_Union`, `ST_Intersection`. Pas de topologie. Travail exclusivement en WGS84 — pas de reprojection. Performance dégradée sur polygones complexes (milliers de sommets). Pas de fonctions d'agrégation spatiale.

Elasticsearch Geo

Elasticsearch (lancé le **8 février 2010** par Shay Banon) intègre des capacités géospatiales dès ses premières versions. En 2026, la vraie valeur d'Elasticsearch pour le stockage géospatial n'est pas la performance spatiale — PostGIS est meilleur sur ce point — c'est la **combinaison requête textuelle + filtre spatial** dans un seul moteur distribué.

`geo_point`	Stocke un point (latitude/longitude). Très performant pour les requêtes de proximité et de filtrage spatial sur des points. Cas d'usage : POI, positions utilisateurs, adresses.
`geo_shape`	Stocke n'importe quelle géométrie GeoJSON. Basé sur un index BKD-tree (ES 7.x+). Supporte les requêtes INTERSECTS, WITHIN, CONTAINS, DISJOINT. Beaucoup plus coûteux en ressources que geo_point.

JSON

```
// "Toutes les boulangeries artisanales dans un rayon de 500m"
{
  "query": {
    "bool": {
      "must": {
        "match": { "nom": "boulangerie artisanale" }
      },
      "filter": {
        "geo_distance": {
          "distance": "500m",
          "localisation": { "lat": 48.8584, "lon": 2.2945 }
        }
      }
    }
  }
}
```

⚠️ **Coût et licence 2026**

Pas de transactions, pas d'intégrité référentielle, pas de fonctions d'analyse spatiale avancée. Cluster minimum 3 nœuds recommandé en production. Elastic a réintroduit une licence open source (AGPL) depuis fin 2024. **OpenSearch** (fork Apache 2.0) reste l'alternative à privilégier sans restriction d'usage cloud.

Redis Geo

Redis intègre les commandes `GEO*` à partir de la version **3.2** (avril 2016). Les données GEO sont stockées comme scores dans un sorted set Redis, encodées en Geohash 52 bits — précision sub-centimétrique, largement suffisante pour toutes les applications de géolocalisation.

COMMANDE	USAGE
`GEOADD clé lon lat membre`	Ajouter un point
`GEODIST clé m1 m2 [unité]`	Distance entre deux points
`GEOSEARCH clé FROMMEMBER m BYRADIUS r km`	Points dans un rayon (Redis 6.2+)
`GEOPOS clé membre`	Coordonnées d'un membre
`GEOHASH clé membre`	Geohash d'un membre

Redis Geo = cache spatial, pas stockage primaire

Ses cas d'usage sont exclusivement ceux qui nécessitent une latence sub-milliseconde sur des données de points : classement de proximité temps réel (< 1ms), géofencing léger, compteurs spatiaux. Les données ont vocation à être éphémères ou régénérées depuis une source de vérité (PostGIS, MongoDB). **Points uniquement** — pas de lignes, pas de polygones.

TimescaleDB — le stockage spatio-temporel

TimescaleDB est une extension PostgreSQL lancée par **Timescale Inc.** en **2017**, conçue pour les séries temporelles. Elle ne remplace pas PostGIS — elle le **complète** pour les données à forte dimension temporelle. Elle organise les données en **hypertables**, partitionnées automatiquement par intervalle de temps en **chunks** internes (compressibles, archivables, supprimables automatiquement).

SQL

```sql
-- Table de positions GPS de véhicules
CREATE TABLE positions (
  temps        TIMESTAMPTZ NOT NULL,
  vehicule_id  INTEGER NOT NULL,
  geom         GEOMETRY(Point, 4326) NOT NULL,
  vitesse      FLOAT,
  cap          FLOAT
);

-- Conversion en hypertable (partitionnement par jour)
SELECT create_hypertable('positions', 'temps');

-- Index spatial sur la géométrie
CREATE INDEX ON positions USING GIST(geom);
```

Cas d'usage : suivi GPS de flottes (positions toutes les secondes), données de capteurs IoT géolocalisés (qualité de l'air, météo, trafic), trajectoires et analyses de mobilité, télémétrie géolocalisée.

CRITÈRE	TIMESCALEDB + POSTGIS	INFLUXDB
Stockage géospatial	✅ PostGIS complet	⚠️ Tags lat/lon seulement
Analyse spatiale	✅ Toutes fonctions PostGIS	❌ Aucune
SQL standard	✅ PostgreSQL natif	⚠️ Flux (langage propriétaire)
Cas d'usage dominant	IoT géospatial, mobilité	Monitoring, métriques IT

NoSQL géospatial vs PostGIS — quand choisir quoi

SITUATION	CHOIX RECOMMANDÉ
Édition collaborative de données géographiques	PostGIS
Analyse spatiale avancée (buffer, union, overlay)	PostGIS
Intégrité topologique, contraintes géographiques	PostGIS exclusivement
Données IoT avec attributs variables par type de capteur	MongoDB
Recherche locale (full-text + proximité)	Elasticsearch
Cache de proximité temps réel (< 1ms)	Redis GEO
Séries temporelles géolocalisées haute fréquence	TimescaleDB + PostGIS
Documents géolocalisés riches hétérogènes	MongoDB

Partie 6 — Stockage cloud-native et analytique (2020–2026)

2020–2026 — Cloud-native

GeoParquet, COG et DuckDB ne sont pas des améliorations incrémentales. Ils représentent une remise en question fondamentale : stocker la donnée géospatiale sur du stockage objet (S3, R2) au format columnar, et la requêter **sans serveur dédié**.

COG — Cloud Optimized GeoTIFF

→ Architecture interne du format détaillée dans le *Livre Blanc Formats Géospatiaux 2026*. Ce chapitre traite du stockage objet, de la génération et des stratégies de déploiement.

Le COG est un GeoTIFF réorganisé pour que toutes les métadonnées soient en tête de fichier et que les overviews + tuiles soient géographiquement ordonnées. Cela permet à un client HTTP de requêter uniquement les octets nécessaires — souvent 1 à 5 % du fichier total pour une vue typique.

1 **Lecture du header**

Quelques Ko — métadonnées, SRS, position des IFDs (overviews). Une seule requête HTTP Range.

2 **Calcul des offsets**

Le client calcule la position des tuiles correspondant au zoom et à la zone souhaitée.

3 **Requête partielle**

Uniquement les octets utiles — 1 à 5 % du fichier total pour une vue cartographique typique.

Workflow de génération

BASH

```
# Conversion GeoTIFF classique → COG avec GDAL
gdal_translate \
  -of COG \
  -co COMPRESS=ZSTD \
  -co OVERVIEW_RESAMPLING=LANCZOS \
  -co OVERVIEW_COMPRESS=ZSTD \
  input.tif \
  output_cog.tif

# Validation avec rio-cogeo (pip install rio-cogeo)
rio cogeo validate output_cog.tif
```

Stratégies d'overviews selon la volumétrie

TAILLE RASTER SOURCE	STRATÉGIE
< 100 Mo	4–5 niveaux d'overview internes suffisent
100 Mo – 10 Go	6–8 niveaux, compression ZSTD agressive
> 10 Go	Mosaïque de COG via `cogeo-mosaic` + catalog STAC
Données scientifiques (NetCDF, HDF5)	Conversion GDAL → COG avec LERC compression (précision contrôlable)

GeoParquet (spécification v1.0 stable — septembre 2023)

→ Architecture du format et encodage WKB/GeoArrow détaillés dans le *Livre Blanc Formats Géospatiaux 2026*. Ce chapitre traite du partitionnement, du stockage S3 et des stratégies de requête.

GeoParquet 1.0.0-beta.1 est publié en décembre 2022. La version stable **GeoParquet 1.0.0** est publiée en **septembre 2023**. Contributeurs fondateurs : GeoPandas, GeoTrellis, Voltron Data, Microsoft, CARTO, Planet.

Apache Parquet stocke les données d'une même colonne de manière contiguë sur disque. Pour une requête analytique typique — "superficie totale de toutes les parcelles dans une région" — un moteur columnar ne lit que deux colonnes (`geom` et `surface`), pas l'intégralité de chaque ligne. Sur 10 millions d'entités avec 50 attributs, l'économie est considérable.

Partitionnement spatial

Par bbox Diviser en tuiles géographiques régulières. Simple, mais partitions déséquilibrées si les données sont concentrées géographiquement.

Par Geohash Coder chaque entité avec son Geohash et partitionner par valeur. Meilleure répartition pour les données irrégulièrement distribuées.

Z-order Ordonner les entités selon une courbe de Morton avant de partitionner. Optimise les requêtes de range spatial sur Delta Lake / Iceberg.

Lire GeoParquet depuis DuckDB, GeoPandas, Spark Sedona

PYTHON

```python
# GeoPandas
import geopandas as gpd
gdf = gpd.read_parquet("s3://mon-bucket/parcelles.parquet",
                       storage_options={"anon": True})

# DuckDB
import duckdb
conn = duckdb.connect()
conn.execute("INSTALL spatial; LOAD spatial;")
result = conn.execute("""
  SELECT ST_Area(geometry) as superficie, commune
  FROM read_parquet('s3://mon-bucket/parcelles.parquet')
  WHERE bbox.xmin > 2.0 AND bbox.xmax < 3.0
    AND bbox.ymin > 48.0 AND bbox.ymax < 49.0
""").fetchdf()
```

Comparatif GeoParquet vs PostGIS pour l'analytique

CRITÈRE	POSTGIS	GEOPARQUET + DUCKDB
Lecture analytique (scan complet, 10M entités)	⚠️ 15–60 s	✅ 1–5 s
Requête spatiale indexée (ST_Intersects + index)	✅ < 200 ms	⚠️ 500 ms–2 s
Écriture / édition interactive	✅ Native, ACID	❌ Immutable (réécriture)
Coût infra mensuel (50 Go)	⚠️ 30–80 € (VPS)	✅ < 1 € (S3/R2)
Mise à jour incrémentale	✅ INSERT/UPDATE/DELETE	❌ Delta Lake requis
Concurrence d'écriture	✅ Multi-utilisateurs	❌ Mono-processus
Compatibilité QGIS, GeoServer	✅ Excellente	⚠️ Croissante (GDAL 3.5+)

→

Le verdict

GeoParquet n'est pas un remplacement de PostGIS. C'est un format **complémentaire** pour les cas d'usage analytiques — là où PostGIS est surdimensionné en infrastructure et sous-performant en lecture massive. La bonne architecture 2026 combine les deux.

DuckDB + extension spatial

DuckDB est créé par **Mark Raasveldt** et **Hannes Mühleisen** au CWI Amsterdam. Version stable **1.0.0** ("Snow Duck") publiée le **3 juin 2024**. Extension `spatial` disponible depuis la version 0.7 (2023). Moteur analytique in-process — il s'exécute dans le processus appelant, sans serveur séparé. Lit directement depuis S3, R2, fichiers locaux, HTTP.

SQL

```
INSTALL spatial; LOAD spatial;

-- Lire GeoParquet S3, filtrer spatialement, écrire en GeoParquet
COPY (
  SELECT geom, commune, surface_ha
  FROM read_parquet('s3://mon-bucket/france/parcelles.parquet')
  WHERE ST_Within(
    geom,
    ST_GeomFromText('POLYGON((2.2 48.8, 2.4 48.8, 2.4 48.9, 2.2 48.9, 2.2 48.8))')
  )
  AND surface_ha > 1.0
) TO 'paris_grandes_parcelles.parquet' (FORMAT PARQUET);
```

PYTHON

```
import duckdb

conn = duckdb.connect()
conn.execute("INSTALL spatial; LOAD spatial;")

# ETL : Shapefile → reprojection Lambert 93 → WGS84 → GeoParquet
conn.execute("""
  COPY (
    SELECT
      ST_Transform(geom_src, 'EPSG:2154', 'EPSG:4326') as geom,
      nom_commune, code_insee,
      ST_Area(geom_src) / 10000 as surface_ha   -- en Lambert 93 = m²
    FROM (
      SELECT geom as geom_src, nom_commune, code_insee
      FROM ST_Read('communes.shp')
      WHERE code_insee LIKE '75%'
    )
  ) TO 'paris_communes.parquet' (FORMAT PARQUET)
""")
```

Benchmark Overture Maps France (~30M entités, 8 Go GeoParquet)

REQUÊTE	POSTGIS (INDEX GIST)	DUCKDB + GEOPARQUET
Bâtiments département 75 (requête indexée)	~800 ms ✅	~2,5 s ⚠️
Superficie totale France (scan complet)	~45 s ⚠️	~4 s ✅

Serveur Linux 16 Go RAM, disque NVMe SSD. PostGIS gagne sur les requêtes indexées précises. DuckDB gagne sur les scans analytiques complets. Les deux sont complémentaires.

> ⚠️ **Limites DuckDB**
>
> Mono-nœud — au-delà de 500 Go ou pour les requêtes distribuées, Spark + Sedona est nécessaire. Pas de concurrence d'écriture. Pas de mise à jour incrémentale native. Index R-Tree moins mature que le GiST PostGIS.

Delta Lake / Apache Iceberg + géospatial

Delta Lake est lancé par **Databricks** en **2019** pour apporter les propriétés ACID aux lacs de données Parquet. **Apache Iceberg** est créé par **Ryan Blue et Daniel Weeks** chez Netflix (Apache Incubator 2018, projet top-level 2020). GeoParquet est immutable — toute modification implique de réécrire le fichier complet. Delta Lake résout ce problème via un journal de transactions (`_delta_log`) qui enregistre chaque opération.

PYTHON

```python
from delta import DeltaTable
from pyspark.sql import SparkSession
from sedona.spark import SedonaContext

spark = SedonaContext.create(SparkSession.builder
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension")
    .getOrCreate())

# Écriture géospatiale en Delta
parcelles_df.write.format("delta").save("s3://mon-bucket/delta/parcelles")

# Mise à jour ACID : corriger la géométrie d'une parcelle
deltaTable = DeltaTable.forPath(spark, "s3://mon-bucket/delta/parcelles")
deltaTable.update(
    condition = "id_parcelle = '75001-001'",
    set = {"geom": "ST_GeomFromText('POLYGON(...)')"}
)

# Time travel : état du dataset il y a 7 jours
spark.read.format("delta") \
    .option("timestampAsOf", "2026-03-10") \
    .load("s3://mon-bucket/delta/parcelles")
```

CRITÈRE	DELTA LAKE	APACHE ICEBERG
Origine	Databricks (2019)	Netflix / Apache (2018)
Time travel	✅	✅
Support DuckDB	✅ delta-rs	✅ iceberg-rust
Écosystème dominant	Databricks / Azure Synapse	AWS Athena / Glue, multi-cloud

Overture Maps + GeoParquet

Overture Maps Foundation est lancée en **décembre 2022** par Amazon, Microsoft, Meta et TomTom. Première release publique : **juillet 2023**. Overture publie ses données exclusivement en GeoParquet partitionné sur AWS S3 — la plus grande validation publique du format à ce jour.

BASH

```
# Télécharger les bâtiments de France depuis Overture Maps avec DuckDB
duckdb -c "
INSTALL spatial; LOAD spatial; INSTALL httpfs; LOAD httpfs;

COPY (
  SELECT id, geometry, height, num_floors
  FROM read_parquet('s3://overturemaps-us-west-2/release/{RELEASE_DATE}/theme=buildings/type=building/
*',
                    filename=true, hive_partitioning=1)
  -- Remplacer {RELEASE_DATE} par la date courante : overturemaps.org/download/
  WHERE bbox.xmin > -5.1 AND bbox.xmax < 9.6
    AND bbox.ymin > 41.3 AND bbox.ymax < 51.1
) TO 'france_buildings.parquet' (FORMAT PARQUET);
"
```

Partie 7 — Stockage raster : du fichier à l'objet cloud

Le stockage raster géospatial a suivi une trajectoire parallèle au vecteur : du fichier local isolé vers le stockage objet cloud requêtable à distance. GeoTIFF pyramidé reste le standard local ; COG est devenu le standard cloud ; STAC en est le catalogue.

PostGIS Raster (postgis_raster)

L'extension `postgis_raster` permet de stocker des données raster dans PostgreSQL sous forme de tuiles. L'outil CLI `raster2pgsql` génère les instructions SQL de chargement depuis un GeoTIFF.

BASH

```
# Chargement d'un GeoTIFF en base PostGIS — tuiles 100×100 px
raster2pgsql -s 4326 -I -C -M -t 100x100 mnt_france.tif public.mnt_france | psql -d geodata
```

FONCTION	USAGE
`ST_Value(rast, pt)`	Valeur du pixel à un point donné
`ST_Clip(rast, geom)`	Découpe d'un raster par une géométrie vecteur
`ST_Resample(rast, srid, scale)`	Rééchantillonnage spatial
`ST_Union(rast)`	Fusion de tuiles raster (agrégat)
`ST_MapAlgebra(rast1, rast2, expr)`	Algèbre raster (addition, soustraction de bandes)

⚠️ **Statut 2026 : usage en déclin**

Base de données qui explose en volume, performance dégradée vs COG pour les accès distants, maintenance coûteuse. Pour les nouveaux projets, **COG sur stockage objet est systématiquement préférable**. PostGIS Raster reste pertinent uniquement pour les petits rasters à interroger via SQL en combinaison avec des données vecteur.

GeoTIFF pyramidé — le raster local classique

RECOMMANDÉ

Overviews internes

- ✓ Stockées dans le même `.tif` fichier
- ✓ Fichier unique, pas de synchronisation
- ✓ `gdal_addo -r lanczos input.tif 2 4 8 16 32 64`

VS

CAS PARTICULIERS

Overviews externes (.ovr)

- → Fichier `.ovr` séparé
- → Utile si la source est en lecture seule
- → Synchronisation à gérer manuellement

Compression : quel choix pour quel usage

COMPRESSION	RATIO	VITESSE DÉCOMPRESSION	USAGE RECOMMANDÉ
LZW	Bon	Rapide	Rasters thématiques, classification, MNA entier
DEFLATE	Meilleur	Moyen	Usage général
ZSTD	Excellent	Très rapide	Cloud, performances prioritaires — recommandé pour COG
LERC	Excellent (lossy possible)	Rapide	Données scientifiques (précision contrôlable)
WebP	Excellent (lossy)	Moyen	Ortho-photos (perte acceptable)

COG sur stockage objet — le raster cloud

Un COG hébergé sur Cloudflare R2 ou AWS S3 peut être consommé directement par QGIS (via GDAL/VSI), par des scripts Python (`rasterio` , `rio-cogeo`), par des APIs (TiTiler — exposition traitée dans LB3) ou par des navigateurs.

INFRASTRUCTURE	STOCKAGE/MOIS	REQUÊTES 1M/MOIS	TOTAL ESTIMÉ
AWS S3	~46 $	~0,40 $	~46 $
Cloudflare R2	~30 $	Gratuit (egress)	~30 $
Scaleway Object Storage	~23 €	Inclus	~23 €
Hetzner Storage Box	Forfait fixe	Inclus	~10 € ⚠️ proxy Minio requis pour HTTP Range

STAC — cataloguer le raster cloud

Le **SpatioTemporal Asset Catalog (STAC)** est une spécification communautaire dont le premier sprint a lieu en **octobre 2017** (Radiant Earth Foundation). Version **1.0.0** publiée le **25 mai 2021**. STAC répond à un problème simple : comment découvrir et requêter des COG stockés sur S3 sans les connaître individuellement ?

Item	Un asset géospatial unique (scène satellite, COG de département, tuile LiDAR). Contient métadonnées spatiales (bbox, empreinte), temporelles (date d'acquisition) et liens vers les fichiers.
Collection	Groupe d'Items partageant les mêmes caractéristiques (même capteur, même résolution, même périmètre géographique).
Catalog	Point d'entrée racine. Liste les Collections disponibles. Implémentations : stac-fastapi (Python/FastAPI), pystac (Python), plugin QGIS STAC Browser.

JSON

```
{
  "type": "Feature",
  "stac_version": "1.0.0",
  "id": "idf-ortho-2024-75",
  "bbox": [2.224, 48.815, 2.470, 48.902],
  "geometry": { "type": "Polygon", "coordinates": [[...]] },
  "properties": {
    "datetime": "2024-06-15T00:00:00Z",
    "resolution": 0.2,
    "crs": "EPSG:2154"
  },
  "assets": {
    "image": {
      "href": "s3://mon-bucket/ortho/idf-2024-75.tif",
      "type": "image/tiff; application=geotiff; profile=cloud-optimized"
    }
  }
}
```

Stratégies de pyramidage et de compression — tableau récapitulatif

FORMAT	COMPRESSION RECOMMANDÉE	PYRAMIDES	USAGE PRINCIPAL
GeoTIFF local (SIG desktop)	LZW ou DEFLATE	Externes (.ovr) ou internes	Flux QGIS, GRASS, ArcGIS
COG cloud (S3/R2)	ZSTD	Internes (obligatoires)	Exposition web, accès distant
COG scientifique	LERC (précision contrôlée)	Internes	Données météo, alti, scientifiques
MBTiles raster	PNG / WebP	Tiles fixes par zoom	Fond de carte offline, TileServer GL
PostGIS Raster	N/A (compressé par PG)	Overviews SQL	Requêtes SQL vecteur+raster combinées

Partie 8 — Sécurité, accès et gouvernance du stockage

Contrôle d'accès à PostGIS

La bonne pratique est de créer des rôles métier correspondant aux profils d'utilisation, plutôt que de gérer les droits par utilisateur.

SQL

```
-- Rôle lecture seule SIG
CREATE ROLE sig_lecteur;
GRANT CONNECT ON DATABASE geodata TO sig_lecteur;
GRANT USAGE ON SCHEMA public TO sig_lecteur;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO sig_lecteur;

-- Rôle éditeur (lecture + écriture sur tables métier)
CREATE ROLE sig_editeur;
GRANT sig_lecteur TO sig_editeur;
GRANT INSERT, UPDATE ON TABLE parcelles, voirie, zonage TO sig_editeur;

-- Rôle admin
CREATE ROLE sig_admin;
GRANT ALL ON ALL TABLES IN SCHEMA public TO sig_admin;
GRANT ALL ON ALL SEQUENCES IN SCHEMA public TO sig_admin;
```

Row-Level Security (RLS) sur des tables géographiques

SQL

```
-- Activer RLS sur la table parcelles
ALTER TABLE parcelles ENABLE ROW LEVEL SECURITY;

-- Politique : un utilisateur ne voit que les parcelles de sa commune
CREATE POLICY acces_commune ON parcelles
  USING (
    code_commune IN (
      SELECT code_commune
      FROM utilisateurs_communes
      WHERE utilisateur = current_user
    )
  );
```

Chiffrement au repos et en transit

LUKS / LUKS2	Chiffrement du volume système de fichiers sur Linux — transparent pour PostgreSQL. Option la plus simple et robuste pour un VPS PostGIS.
pgcrypto	Extension PostgreSQL — chiffre des valeurs individuelles (`pgp_sym_encrypt` , `pgp_sym_decrypt`). Utile pour chiffrer des colonnes sensibles (noms, adresses) dans des tables géospatiales.
S3 / R2 SSE	AWS S3 SSE-S3 (AES-256, activé par défaut depuis 2023) ou SSE-KMS (rotation automatique des clés). Cloudflare R2 : chiffrement AES-256 au repos par défaut.
TLS PostgreSQL	Paramètre `ssl = on` dans `postgresql.conf` avec certificat serveur valide. Toutes les connexions doivent utiliser TLS en production.

Sauvegarde et restauration d'une base PostGIS

BASH

```
# Sauvegarde standard (format custom compressé)
pg_dump -Fc -d geodata -f geodata_backup_$(date +%Y%m%d).dump

# Sauvegarde parallèle pour bases > 50 Go (format directory, 4 workers)
pg_dump -Fd -j 4 -d geodata -f geodata_backup_$(date +%Y%m%d)/

# Restauration complète
pg_restore -d geodata_restore geodata_backup_20260317.dump

# Restauration d'une table spécifique
pg_restore -d geodata -t parcelles geodata_backup_20260317.dump
```

MÉTHODE	RPO	CAS D'USAGE
`pg_dump` quotidien	24h	Bases < 50 Go, mises à jour peu fréquentes
`pg_dump -j N` (parallèle)	24h	Bases 50–500 Go
WAL archiving + PITR (pgBackRest / Barman)	Quelques secondes	Bases en production avec mises à jour fréquentes

Note archive_command

La commande `archive_command = 'cp %p ...'` est illustrative. Elle ne gère ni les erreurs ni les retentatives. En production, utiliser **pgBackRest** ou **Barman** qui gèrent la rétention, les erreurs et la restauration automatique.

Vérification de l'intégrité après restauration

SQL

```sql
-- Vérifier les géométries invalides
SELECT id, ST_IsValidReason(geom) as raison
FROM parcelles
WHERE NOT ST_IsValid(geom);

-- Compter les entités par type pour détecter une corruption
SELECT ST_GeometryType(geom), COUNT(*)
FROM parcelles
GROUP BY ST_GeometryType(geom);
```

RGPD et données de géolocalisation

⚠️ **Les coordonnées GPS comme données personnelles**

La position géographique d'une personne est une donnée personnelle au sens du RGPD dès qu'elle permet d'identifier directement ou indirectement l'individu : adresses domiciliaires géocodées, traces GPS d'appareils mobiles liés à un utilisateur identifié, positions de capteurs IoT en espaces privatifs, parcelles cadastrales nominatives. Les données purement administratives (limites communales, réseaux routiers, zones PLU) ne sont pas des données personnelles.

Techniques d'anonymisation spatiale

Floutage — Déplacement aléatoire dans un rayon donné. Opérer en Lambert 93 (EPSG:2154) pour un résultat métrique correct, puis reprojeter en WGS84 : `ST_Transform(ST_Translate(ST_Transform(geom, 2154), (random()-0.5)*rayon_m, (random()-0.5)*rayon_m), 4326)`. Ne pas opérer directement en degrés décimaux.

Agrégation — Remplacer des positions individuelles par des comptages par maille (IRIS, carreaux 200m). Utilisé pour la publication de données de mobilité.

k-anonymat	Garantir que chaque point est indiscernable d'au moins k-1 autres points dans un voisinage donné.

SQL

```sql
-- Purge automatique des positions GPS > 90 jours
CREATE OR REPLACE FUNCTION purge_positions_anciennes() RETURNS void AS $$
BEGIN
  DELETE FROM positions_gps
  WHERE horodatage < NOW() - INTERVAL '90 days';
END;
$$ LANGUAGE plpgsql;

-- Planification via pg_cron
SELECT cron.schedule('purge-positions', '0 3 * * *', 'SELECT purge_positions_anciennes()');
```

Versioning et historique des données géospatiales

SQL

```sql
-- Schéma de versionning géospatial manuel
CREATE TABLE parcelles_historique (
  id              SERIAL PRIMARY KEY,
  parcelle_id     INTEGER NOT NULL,
  geom            GEOMETRY(Polygon, 4326) NOT NULL,
  attributs       JSONB,
  valid_from      TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  valid_to        TIMESTAMPTZ,
  modifie_par     TEXT NOT NULL DEFAULT current_user,
  operation       CHAR(1) CHECK (operation IN ('I', 'U', 'D'))
);

-- État des parcelles à une date donnée
SELECT * FROM parcelles_historique
WHERE valid_from <= '2025-01-01'
  AND (valid_to IS NULL OR valid_to > '2025-01-01');
```

BASH

```
# Versionner un fichier GeoParquet avec DVC
dvc init
dvc remote add -d s3remote s3://mon-bucket/dvc-storage

dvc add france_parcelles_2026.parquet
git add france_parcelles_2026.parquet.dvc .gitignore
git commit -m "feat: mise à jour parcelles 2026"
dvc push
```

Partie 9 — Guide décisionnel complet

Ce guide synthétise les enseignements des parties précédentes en outils directement utilisables : matrice d'usage, arbre de décision, comparatif de coûts réels et cinq scénarios architecturaux bout en bout.

9.1 Matrice d'usage

CAS D'USAGE	VOLUME INDICATIF	SOLUTION RECOMMANDÉE	ALTERNATIVE
SIG desktop, échange de fichiers	< 500 Mo	**GeoPackage**	Shapefile (legacy uniquement)
Application web interactive, édition multi-users	Toute volumétrie	**PostGIS**	SpatiaLite (offline uniquement)
Analyse spatiale avancée (buffer, union, overlay)	Variable	**PostGIS**	DuckDB + GeoParquet (analytique)
ETL / transformation géospatiale batch	Grand volume	**DuckDB + GeoParquet**	GDAL/OGR + PostGIS
Raster local, SIG desktop	Variable	**GeoTIFF pyramidé (= COG)**	GeoTIFF sans overviews (déconseillé)
Raster cloud, exposition web	Variable	**COG sur S3/R2**	PostGIS Raster (petits volumes)
Données IoT géolocalisées haute fréquence	Volume élevé, temps réel	**TimescaleDB + PostGIS**	MongoDB
Recherche locale (full-text + proximité)	Variable	**Elasticsearch + geo_shape**	PostGIS + pg_trgm
Cache de proximité temps réel	Petits volumes, < 1ms	**Redis GEO**	PostGIS (si latence tolérable)
Analytics massif (> 50M entités, scan complet)	Très grand	**GeoParquet + DuckDB**	BigQuery Geo (cloud managé)
Fond de carte vectoriel statique	Planet-scale	**GeoParquet → Tippecanoe → PMTiles**	MBTiles + TileServer GL
Lac de données géospatiales avec mises à jour	Grand volume	**Delta Lake + GeoParquet + Sedona**	Apache Iceberg + GeoParquet
Catalogue raster cloud	Variable	**STAC + COG**	GeoServer (OGC WCS)

9.2 Arbre de décision

FIGURE 3 — ARBRE DE DÉCISION DU STOCKAGE GÉOSPATIAL

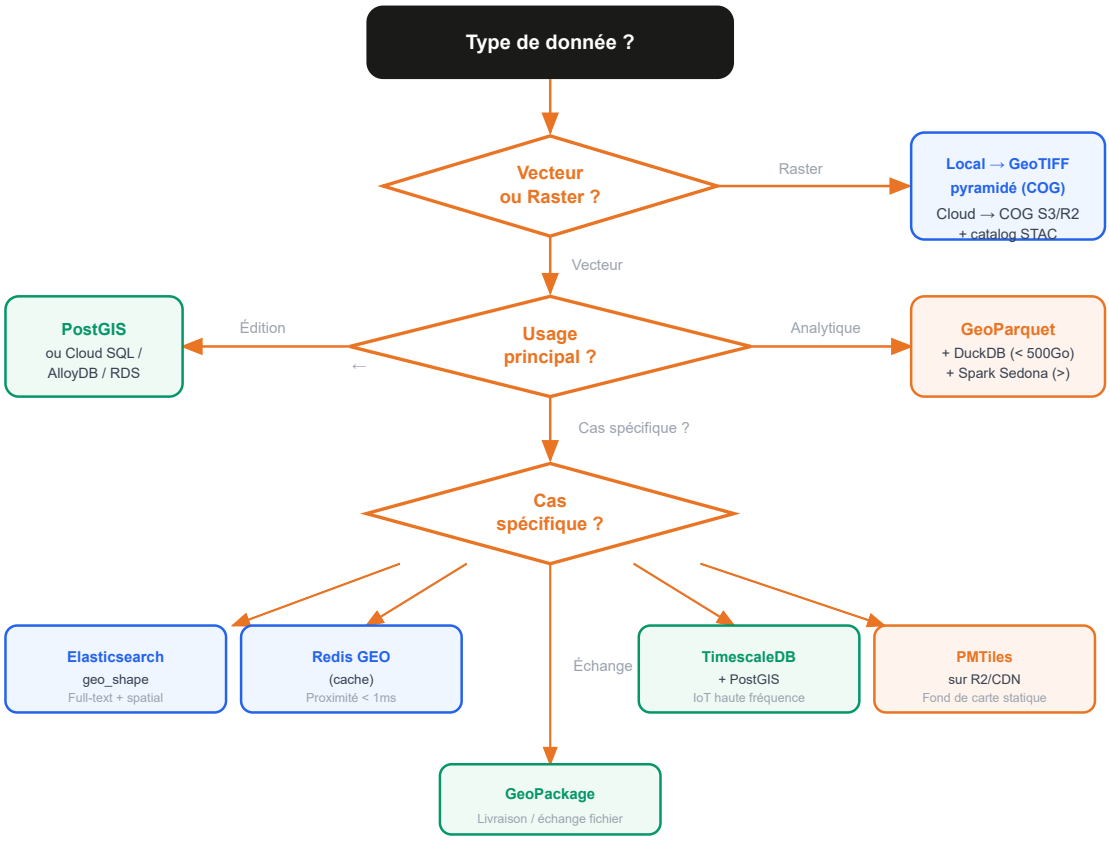

FIGURE 3 — ARBRE DE DÉCISION DU STOCKAGE GÉOSPATIAL

9.3 Comparatif coûts réels 2026

⚠️ Tarifs vérifiés en mars 2026 — hors coûts de développement, maintenance et services périphériques. Consulter les calculateurs officiels AWS et Cloudflare R2 pour des projections précises en production.

Tarifs de référence stockage objet 2026

SERVICE	STOCKAGE (PAR GO/MOIS)	EGRESS (PAR GO)	REQUÊTES
AWS S3 Standard (us-east-1)	0,023 $	0,09 $ (>100 Go)	GET : 0,0004 $/1 000
Cloudflare R2 Standard	0,015 $	Gratuit	Class A : 4,50 $/M · Class B : 0,36 $/M
Scaleway Object Storage (Paris)	~0,015 €	75 Go offerts/mois	Inclus
Hetzner Object Storage (nbg1)	~0,0059 €	Inclus (1 To/mois)	Inclus ⚠️ proxy Minio requis

FIGURE 4 — COMPARATIF COÛTS MENSUELS ESTIMÉS PAR PROFIL PROJET (MARS 2026)

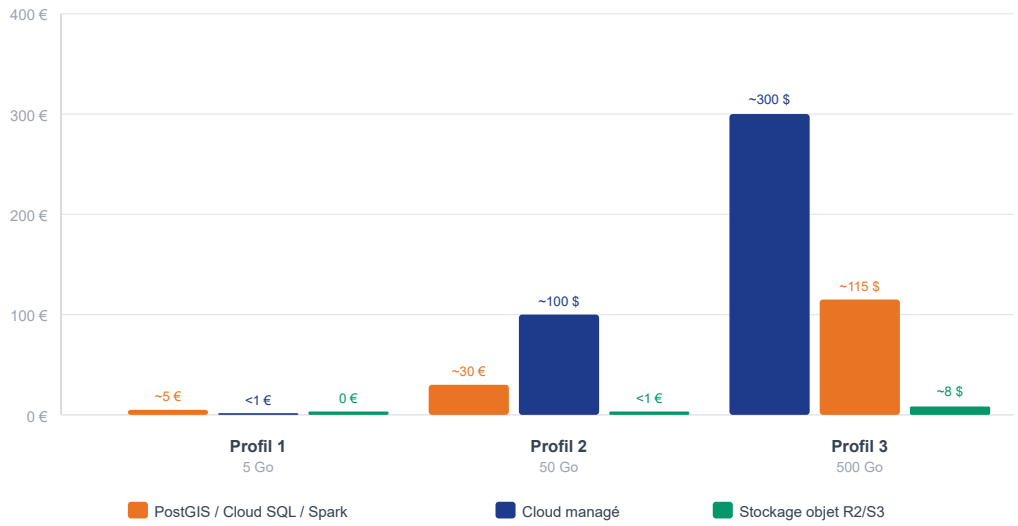

FIGURE 4 — COMPARATIF COÛTS MENSUELS ESTIMÉS PAR PROFIL PROJET (MARS 2026)

9.4 Cinq scénarios architecturaux

S1

Application SIG locale (bureau, hors-ligne)

Stockage principal : GeoPackage (.gpkg) · Stockage référence : GeoTIFF pyramidé (COG) local · Échange : GeoPackage ou Shapefile (legacy) · Sauvegarde : copie du .gpkg sur NAS ou Nextcloud. Critères : portabilité maximale, aucune dépendance serveur, compatibilité QGIS native.

S2

Application web SIG collaborative

PostgreSQL 16 + PostGIS 3.5 · Accès QGIS direct · API web FastAPI/Django + GeoJSON · Tuiles : pg_tileserv ou Martin → MapLibre GL JS · Raster : COG sur S3/R2 + TiTiler · Sauvegarde : pg_dump + WAL archiving (pgBackRest).

S3

Pipeline analytique géospatial

Source : Overture Maps S3 ou PostGIS export · ETL : DuckDB + extension spatial · Stockage intermédiaire : GeoParquet partitionné sur S3/R2 · Stockage résultat : GeoParquet agrégé ou PostGIS · Orchestration : Python + cron ou Apache Airflow.

S4

Fond de carte vectoriel haute performance

Source de vérité : PostGIS · Pipeline : PostGIS → export GeoJSON → Tippecanoe → PMTiles → upload R2 · Distribution : Cloudflare R2 (egress gratuit) + Workers pour CORS/cache · Client : MapLibre GL JS.

S5

Lac de données géospatiales d'entreprise

Couche transactionnelle : PostGIS (source de vérité) · Couche analytique : GeoParquet + Delta Lake sur S3 (historique, ACID) · Traitement distribué : Spark + Sedona (> 100M entités) · Exposition : Martin + TiTiler · Orchestration : Apache Airflow ou Dagster.

Partie 10 — Architectures de référence 2026

Quatre stacks types, calibrées par profil de projet. Chaque stack est opérationnelle telle quelle et peut être adaptée selon les contraintes d'infrastructure.

Stack PME / Startup

`20–50 € / mois`

1–5 personnes · projet SIG web ou data · budget infrastructure modéré

- → PostgreSQL 16 + PostGIS 3.5
- → GeoParquet sur Cloudflare R2
- → COG sur R2
- → DuckDB + GDAL/OGR (ETL)
- → pg_dump + WAL archiving

Stack Institutionnelle

`50–200 € / mois`

Collectivité · secteur public · contrainte souveraineté RGPD

- → PostGIS self-hosted (datacenter EU)
- → GeoPackage + Shapefile (legacy)
- → GeoNetwork (ISO 19115)
- → RLS PostgreSQL + rôles métier
- → LUKS + WAL archiving

Stack Cloud-native

200–800 $ / mois

Volumétrie élevée · millions req/mois · scale-up

- → GeoParquet sur Cloudflare R2
- → DuckDB + Spark Sedona
- → AlloyDB / RDS + PostGIS
- → COG + STAC sur R2
- → PMTiles sur R2 + Cloudflare CDN

Stack Big Data

500–2 000 $ / mois

Data engineering · > 500 Go · analytique distribuée

- → Spark + Sedona (distribué)
- → GeoParquet + Delta Lake (S3)
- → DuckDB (< 200 Go) / Athena (> 200 Go)
- → PostGIS (source de vérité)
- → Airflow / Prefect (orchestration)

Figure 5 — Les quatre architectures de référence 2026

Partie 11 — Tendances 2030

Les technologies décrites dans ce livre blanc ne sont pas figées. Cinq dynamiques structurantes dessinent le paysage du stockage géospatial à l'horizon 2030.

GeoParquet comme standard universel du vecteur cloud

GeoParquet 1.0 est stable depuis septembre 2023. Overture Maps publie l'intégralité de ses données en GeoParquet, GDAL 3.5+ supporte le format nativement en lecture/écriture, GeoPandas, DuckDB et Spark Sedona en font un format de première classe. Deux limitations actuelles freinent l'adoption totale : pas de transactions natives (Delta Lake et Apache Iceberg adressent ce point, mais ajoutent de la complexité) ; pas d'index spatial dédié (GeoParquet 1.1 introduit la covering column bbox, un index spatial complet reste à standardiser). Horizon 2027 : GeoParquet probable standard OGC officiel, adoption généralisée dans QGIS 4.x et GRASS GIS.

DuckDB comme ETL géospatial de référence

DuckDB 1.x (stable depuis juin 2024) s'impose comme le moteur ETL géospatial sans infrastructure. Son extension `spatial` couvre l'essentiel des fonctions GDAL/OGR en SQL pur. Deux limites à surveiller : mono-nœud (au-delà de 500 Go, Spark Sedona reste nécessaire) ; DuckDB-WASM + spatial non encore pleinement supporté. Horizon 2027–2028 : DuckDB-WASM spatial mature, analytique géospatiale navigateur viable sur des datasets < 1 Go stockés sur S3/R2.

La convergence PostGIS / analytique

PostGIS reste irremplaçable sur le transactionnel, mais la pression des cas d'usage analytiques pousse à des hybridations : **AlloyDB** (Google Cloud, moteur columnar adaptatif + PostGIS, hybride OLTP/OLAP, en production générale depuis 2023) ; **pg_duckdb** (intégration DuckDB dans PostgreSQL, en forte croissance en 2026) ; **PostGIS + Parquet FDW** (requêtes SQL PostGIS lisant directement des GeoParquet S3 via Foreign Data Wrapper, expérimental en 2026). L'objectif : un seul moteur PostgreSQL capable de gérer transactions géospatiales ET analytique columnar.

Delta Lake et l'ACID sur les lacs de données géospatiaux

L'immutabilité de GeoParquet est un blocage pour les pipelines avec mises à jour fréquentes. En 2026, Delta Lake et Apache Iceberg coexistent avec des bases d'adoption différentes : Delta Lake dominant dans les environnements Databricks et Azure Synapse ; Apache Iceberg en progression rapide sur AWS (Athena, Glue, S3 Tables) et les stacks multi-cloud. La question ouverte pour 2030 :

l'un des deux s'imposera-t-il comme standard universel, ou les deux coexisteront-ils via des couches d'interopérabilité (Apache XTable) ?

Le stockage objet comme nouveau paradigme central

S3, R2 et Azure Blob Storage deviennent les backends de référence de tout le stockage géospatial analytique. Coût au Go 10 à 50× inférieur à un disque attaché à une VM, scalabilité infinie, egress gratuit sur Cloudflare R2, intégration native avec les moteurs analytiques. Conséquence pour les architectures SIG : la séparation stockage / calcul devient le modèle dominant — le stockage objet stocke, le moteur analytique calcule à la demande, sans serveur dédié permanent.

Synthèse — Solutions à surveiller d'ici 2030

SOLUTION	TRAJECTOIRE	HORIZON
GeoParquet	Standard vecteur cloud, adoption généralisée	2027 : standard OGC probable
DuckDB spatial	ETL sans infrastructure, croissance rapide	2026 : déjà en production large
DuckDB-WASM spatial	Analytique navigateur, encore limité	2027–2028 : maturité attendue
Delta Lake géo	ACID sur lac de données, dominant Databricks/Azure	2026–2027 : consolidation
Apache Iceberg géo	Croissance AWS / multi-cloud	2027 : possible standard universel
AlloyDB (PostGIS columnar)	Hybride OLTP/OLAP managé	2026 : en production (Google Cloud)
pg_duckdb	Columnar PostgreSQL + DuckDB dans PG	2026 : actif, forte croissance
PMTiles	Standard tiles statiques, adoption forte	2026–2027 : vers le standard de facto
STAC	Standard catalogue cloud, généralisé	2026 : déjà en production large
SpatiaLite	Stable, niche offline	Pas de croissance attendue
Oracle Spatial	Déclin sur nouveaux projets	Legacy fort, maintenance

Conclusion

Le stockage géospatial a traversé en 25 ans une transformation radicale. Du Shapefile multi-fichiers de 1993, limité par une architecture 32 bits et conçu pour un usage desktop mono-utilisateur, au GeoParquet columnar hébergé sur Cloudflare R2 sans aucune infrastructure dédiée — le chemin est considérable.

> **PostGIS reste la colonne vertébrale. Vingt-cinq ans de maturité, un index GiST éprouvé, des transactions ACID, un écosystème SIG universel. Pour tout projet nécessitant édition collaborative, intégrité spatiale et requêtes indexées, PostGIS n'a pas d'équivalent open source.**
>
> **Le stockage columnar redéfinit l'analytique géospatiale. GeoParquet + DuckDB sur S3/R2 ne remplace pas PostGIS. Il complète l'écosystème pour les cas d'usage que PostGIS gère mal : scans analytiques complets, pipelines batch sans serveur, coût proche de zéro. La bonne architecture 2026 combine les deux.**
>
> **Le fichier plat n'est pas mort — il évolue. Shapefile recule, GeoPackage le remplace. PMTiles redéfinit la distribution de tuiles. FlatGeobuf ouvre le stockage vectoriel cloud sans serveur. COG transforme GeoTIFF en ressource cloud requêtable.**
>
> — Mattieu Pottier, MP-i — Livre Blanc LB2, mars 2026

Trilogie MP-i — Chaîne de la donnée géospatiale

Le **Livre Blanc Formats Géospatiaux 2026** a posé les formats (le QUOI). Le **LB2** a posé le stockage (le OÙ). Le **LB3** ferme la boucle en traitant l'exposition : comment servir et consommer efficacement les données stockées selon les architectures décrites ici.

Références et ressources

Bases de données spatiales

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

SpatiaLite : gaia-gis.it (https://www.gaia-gis.it/gaia-sins/)

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

AlloyDB (Google Cloud) : cloud.google.com/alloydb (https://cloud.google.com/alloydb)

pg_duckdb : github.com/duckdb/pg_duckdb (https://github.com/duckdb/pg_duckdb)

Formats cloud-native

GeoParquet spec : geoparquet.org (https://geoparquet.org/)

COG spec : cogeo.org (https://cogeo.org/)

PMTiles spec : github.com/protomaps/PMTiles (https://github.com/protomaps/PMTiles)

FlatGeobuf : flatgeobuf.org (https://flatgeobuf.org/)

GeoPackage OGC : geopackage.org (https://www.geopackage.org/)

Moteurs analytiques

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

Apache Sedona : sedona.apache.org (https://sedona.apache.org/)

GDAL/OGR : gdal.org (https://gdal.org/)

Stockage objet et cloud

Cloudflare R2 : developers.cloudflare.com/r2 (https://developers.cloudflare.com/r2/)

Delta Lake : delta.io (https://delta.io/)

Apache Iceberg : iceberg.apache.org (https://iceberg.apache.org/)

Catalogues et standards

STAC Spec : stacspec.org (https://stacspec.org/)

Overture Maps : overturemaps.org (https://overturemaps.org/)

OGC Standards : ogc.org/standards (https://www.ogc.org/standards/)

Standards historiques et formats classiques

GeoTIFF spec : geotiff.maptools.org/spec/geotiffhome.html
(https://geotiff.maptools.org/spec/geotiffhome.html)

ESRI Shapefile tech description : esri.com — shapefile.pdf
(https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf)

Outils ETL et génération

cogeo-mosaic : github.com/developmentseed/cogeo-mosaic
(https://github.com/developmentseed/cogeo-mosaic)

Tippecanoe (Felt) : github.com/felt/tippecanoe (https://github.com/felt/tippecanoe)

Planetiler : github.com/onthegomap/planetiler (https://github.com/onthegomap/planetiler)

DVC : dvc.org (https://dvc.org/)

Glossaire

ACID
Atomicity, Consistency, Isolation, Durability. Les quatre propriétés garantissant la fiabilité des transactions en base de données.

BKD-tree
Block KD-tree. Index k-dimensionnel utilisé par Elasticsearch pour les `geo_shape` depuis la v7.x. Meilleure précision (~1e-7°) et meilleures performances mémoire que le prefix tree.

COG (Cloud Optimized GeoTIFF)
GeoTIFF dont la structure interne permet un accès partiel via HTTP Range Requests depuis un stockage objet distant.

Columnar
Mode de stockage où les données d'une même colonne sont contiguës sur disque (opposé au row-oriented). Avantageux pour l'analytique — lecture de colonnes sélectives sur grands volumes.

EPSG / IOGP
Registre de systèmes de référence de coordonnées géographiques maintenu par l'IOGP. EPSG:4326 = WGS84, EPSG:2154 = Lambert 93 France métropolitaine.

Egress
Données sortant d'un datacenter vers Internet. Coût caché sur AWS S3 (0,09 $/Go), gratuit sur Cloudflare R2.

FlatGeobuf
Format binaire vecteur basé sur FlatBuffers, avec index Hilbert R-Tree intégré. Support natif des HTTP Range Requests pour l'accès spatial partiel sans serveur.

GeoArrow
Encodage natif des géométries dans Apache Arrow et Parquet. Alternative à WKB, plus efficace pour l'analytique vectorisée. Intégré dans GeoParquet v1.1.

GeoPackage
Format SQLite standardisé OGC (v1.0 : février 2014) pour stocker vecteur, raster et métadonnées dans un fichier unique. Successeur naturel du Shapefile.

GeoParquet
Spécification de stockage de données vecteur géospatiales dans Apache Parquet. Version stable 1.0 : septembre 2023.

Geohash
Encodage d'une position géographique en chaîne alphanumérique. Utilisé par Redis GEO (52 bits) et comme stratégie de partitionnement GeoParquet.

GiST (Generalized Search Tree)
Framework d'index extensible de PostgreSQL. Utilisé par PostGIS pour construire les index R-Tree spatiaux sur les colonnes géométriques.

HTTP Range Requests
Mécanisme HTTP permettant de requêter un sous-ensemble d'octets d'une ressource distante. Fondement des formats COG, FlatGeobuf et PMTiles.

PITR (Point-In-Time Recovery)
Capacité à restaurer une base de données dans l'état exact à un instant précis dans le passé, grâce au WAL archiving.

PMTiles
Format d'archive de tuiles cartographiques dans un fichier unique, conçu pour le stockage objet (S3/R2). V3 stable : octobre 2022.

PostGIS
Extension spatiale pour PostgreSQL. Première version : mai 2001 (Refractions Research, Dave Blasby et Paul Ramsey). Version stable 2026 : 3.5.x.

R-Tree
Structure d'index spatial organisant les géométries dans une hiérarchie de bounding boxes imbriquées. Base de l'index GiST PostGIS et de l'extension `rtree` SQLite.

RLS (Row-Level Security)
Mécanisme PostgreSQL filtrant les lignes visibles d'une table selon des politiques basées sur l'utilisateur courant. Clé pour la sécurité géospatiale multi-zones.

Shapefile
Format vecteur multi-fichiers ESRI (ArcView 2, 1993). Omniprésent par inertie malgré ses limitations : limite 2 Go, noms de colonnes tronqués à 10 caractères, pas de transactions.

STAC (SpatioTemporal Asset Catalog)
Spécification ouverte pour cataloguer les assets géospatiaux (COG, GeoParquet, LiDAR). Sprint fondateur : octobre 2017. Version 1.0 : 25 mai 2021.

WAL (Write-Ahead Log)
Journal des transactions PostgreSQL. Son archivage continu permet le PITR et la réplication logique.

WKB (Well-Known Binary)
Format binaire standardisé OGC pour encoder les géométries. Utilisé par PostGIS, GeoParquet (encodage par défaut) et les APIs géospatiales.

Z-order (Courbe de Morton)
Fonction mappant des coordonnées 2D à une courbe 1D en préservant la localité spatiale. Utilisé pour ordonner et partitionner les GeoParquet afin d'optimiser les requêtes spatiales.

Historique des révisions

VERSION	DATE	AUTEUR	MODIFICATIONS
v1.0.0	17 mars 2026	Mattieu Pottier	Version initiale — publication
v1.0.1	18 mars 2026	Mattieu Pottier	Corrections post-révision : Tippecanoe attribution, Hetzner note P7, matrice Famille 1 FlatGeobuf, glossaire EPSG, classification TimescaleDB, nettoyage typographique
v1.0.2	18 mars 2026	Mattieu Pottier	Corrections révision approfondie : code DuckDB ST_Area sous-requête, GRANT PG14+, QGIS 4.x, glossaire Tippecanoe, ST_AsMVT LB3, note Famille 1 GeoPackage/FlatGeobuf
v1.0.3	18 mars 2026	Mattieu Pottier	Corrections finales : raster2pgsql clarification P7, note archive_command P8, cogeo-mosaic ajouté aux références

À propos de l'auteur

Mattieu Pottier est consultant indépendant en transformation digitale, spécialisé dans les ERP (Odoo V18/V19), les systèmes d'information géographique (QGIS, développement QGIS), et les applications web (WordPress, FastAPI, Python, JavaScript). Il intervient sous la marque **MP-i — Mattieu Pottier Indépendant** auprès d'organisations souhaitant intégrer des outils open source dans leurs processus métier.

La trilogie de livres blancs techniques MP-i répond à un besoin terrain récurrent : disposer d'une référence synthétique, factuellement vérifiable et architecturalement actionnable sur la chaîne complète des données géospatiales — du format de stockage à l'exposition web.

→ **Contact et publications**

mp-i.pro — Ce document est le deuxième volet de la trilogie MP-i. LB1 couvre les formats géospatiaux (le *quoi*). LB2 se concentre sur les comportements de stockage, les index, les coûts et la gouvernance. LB3 traite la couche d'exposition web.

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 — sans surcoût ni dépendance éditeur.