MP-i

COLLECTION « CHAÎNE DE LA DONNÉE GÉOSPATIALE » — VOLUME 3

Exposer la donnée géospatiale sur le Web (1993–2026)

De MapServer à l'ère serverless — Serveurs, APIs, moteurs de rendu et architectures

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 — Mars 2026

On choisit les formats avec soin, on réfléchit longuement au moteur de stockage — et puis la donnée est là, propre, indexée, requêtable — et on règle la question de l'exposition en quelques heures. Un GeoServer posé rapidement, un endpoint WMS configuré par défaut, et on passe à autre chose. Le résultat est souvent fonctionnel. Il est rarement optimal. Ce troisième volume de la collection « Chaîne de la donnée géospatiale » s'attaque à cette couche trop souvent sous-estimée : comment servir la donnée géospatiale sur le web, architecturalement — pas seulement techniquement. En trente ans, la couche d'exposition a subi quatre ruptures aussi profondes que celles des formats et du stockage.

14
parties thématiques

6
familles d'exposition couvertes

4
architectures de référence 2026

1993
— 2026 couvert

Collection « Chaîne de la donnée géospatiale » — Volume 3

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

SOMMAIRE

Préface

Il existe un paradoxe récurrent dans les projets SIG : on choisit les formats avec soin, on réfléchit longuement au moteur de stockage, et puis — la donnée est là, propre, indexée, requêtable — on règle la question de l'exposition en quelques heures. Un GeoServer posé rapidement, un endpoint WMS configuré par défaut, et on passe à autre chose. Le résultat est souvent fonctionnel. Il est rarement optimal.

Ce troisième volume de la collection « Chaîne de la donnée géospatiale » s'attaque à cette couche trop souvent sous-estimée : comment servir la donnée géospatiale sur le web. Non pas techniquement — les outils sont documentés — mais architecturalement : pourquoi tel serveur plutôt qu'un autre, quelles sont les contraintes réelles de performance et de coût, et surtout, comment la couche d'exposition articule les décisions prises dans LB1 (les formats) et LB2 (le stockage) avec ce que le client JavaScript va réellement recevoir et rendre.

En trente ans, la couche d'exposition a subi quatre ruptures aussi profondes que celles des formats et du stockage. En 1994, exposer une carte sur le web signifiait faire tourner un script CGI en C sur un serveur Unix et envoyer une image GIF ou JPEG par requête HTTP. En 2026, la même carte peut être servie depuis un fichier unique hébergé sur un CDN Cloudflare, sans aucun serveur applicatif, avec un rendu GPU directement dans le navigateur du visiteur. La trajectoire entre ces deux points est l'objet de ce document.

Ce livre blanc suppose la lecture de LB1 et LB2. Les formats (substrat) et le stockage (backend) ne sont pas redécrits ici — des renvois précis permettent de retrouver le détail dans les volumes précédents. LB3 se concentre exclusivement sur les couches service, style et exposition.

Mattieu Pottier

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

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.

Introduction — Pourquoi l'exposition est une décision architecturale

Stocker la donnée géospatiale dans PostGIS ou GeoParquet résout le problème du backend. Mais la donnée n'a de valeur que lorsqu'elle est consommée — par un navigateur, une application mobile, un pipeline analytique ou un rapport PDF. La couche d'exposition est le pont entre le backend de stockage et le consommateur final. Et ce pont conditionne tout le reste.

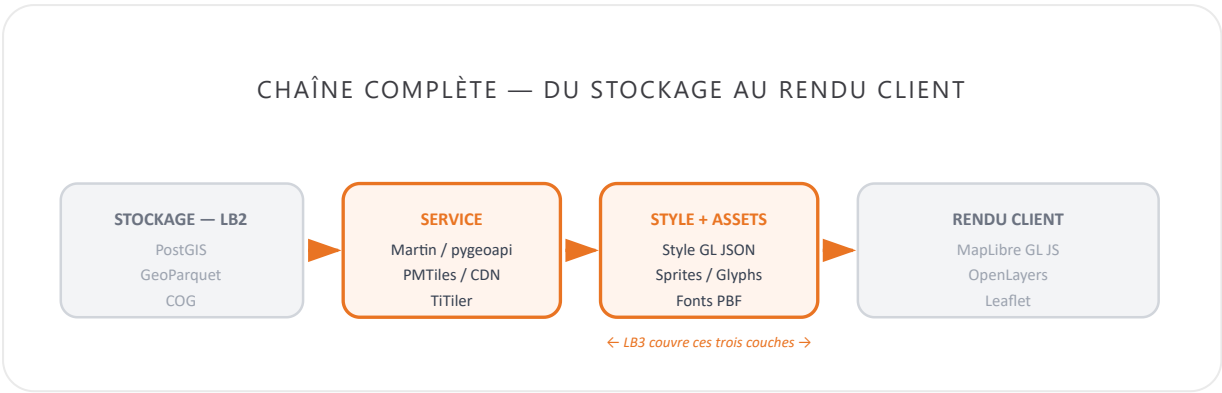
Choisir le mauvais mode d'exposition, c'est potentiellement :

- **Des performances dégradées** — un WMS générant des images PNG à la demande là où des tuiles vectorielles pré-calculées auraient rendu la même carte dix fois plus vite
- **Une scalabilité plafonnée** — un tile server dynamique qui sature à 50 requêtes simultanées là où PMTiles sur CDN aurait absorbé un million de requêtes/jour sans sueur
- **Un coût disproportionné** — un serveur dédié à 80 €/mois là où un fichier PMTiles sur Cloudflare R2 aurait coûté moins d'un euro
- **Une dette de souveraineté** — des dépendances silencieuses vers des CDN Mapbox ou Maptiler qui envoient les IPs de vos utilisateurs à des providers tiers, invisibles dans le code JavaScript
- **Une incompatibilité client** — servir du MVT à un client Leaflet qui ne le supporte pas nativement, ou configurer un tile server qui génère du WMS à un client MapLibre qui ne sait pas le rendre

> ⚠️ **Point fondamental**
>
> La couche d'exposition n'est pas un détail d'implémentation. C'est une décision architecturale qui se prend avant d'écrire le moindre code frontend.

La chaîne complète : de LB2 à LB3

LB1 a établi les formats — le substrat. LB2 a établi les moteurs de stockage — le backend. LB3 couvre les trois couches intermédiaires.



Articulation avec LB2 — de chaque scénario de stockage à son exposition

SCÉNARIO LB2	BACKEND DE STOCKAGE	CE QUE LB3 COMPLÈTE
S1 — SIG local GeoPackage	GeoPackage (.gpkg)	WMS via GeoServer ou MapServer, Leaflet, export web
S2 — Web collaboratif PostGIS	PostgreSQL + PostGIS	Martin ou pg_tileserv → MVT, MapLibre GL JS, pygeoapi
S3 — Pipeline analytique GeoParquet	GeoParquet sur S3/R2 + DuckDB	DuckDB → Tippecanoe → PMTiles → CDN, deck.gl
S4 — Fond de carte PMTiles	PMTiles sur Cloudflare R2	Cloudflare R2 + Workers, MapLibre GL JS, glyphs/sprites self-hosted
S5 — Lac de données enterprise	PostGIS + GeoParquet + Delta Lake	Spark → PMTiles + OGC API Features, stack analytique exposition

Les quatre questions fondamentales

01

Qui consomme ?

Un navigateur web grand public, un client SIG bureautique, une API machine-to-machine, une application mobile, un pipeline batch ? Le consommateur final détermine les formats acceptables, les contraintes de latence et les exigences d'interactivité.

02

Quel volume ?

Quelques centaines de POI ou plusieurs millions d'entités ? Des données statiques ou des mises à jour toutes les secondes ? La volumétrie et la fraîcheur orientent immédiatement vers des familles de solutions différentes.

03

Quelle interactivité ?

Un fond de carte statique, une carte avec clic sur les entités, un filtre attributaire dynamique, ou une requête analytique à la volée ? Plus l'interactivité est élevée, plus la couche service doit être intelligente — et plus l'infrastructure est complexe.

04

Comment lire ce document ?

Lecture linéaire : section par section, des origines CGI de 1994 à l'ère serverless de 2026. **Lecture orientée décision :** directement vers la Partie 12 (Guide décisionnel) et la Partie 13 (Architectures de référence).

Partie 1 — Taxonomie des modes d'exposition

Avant d'entrer dans le détail historique, il est indispensable de disposer d'une grille de lecture commune. Les modes d'exposition géospatiale se répartissent en six grandes familles, chacune avec ses contraintes propres.

1.1 Les six grandes familles

FAMILLE 1

Fichier statique servi

PMTiles · COG · FlatGeobuf

La donnée est encapsulée dans un fichier unique servi via HTTP ou HTTP Range Requests. Aucune logique serveur — CDN ou stockage objet suffit. L'architecture la plus simple, la plus scalable et la moins coûteuse.

CDN · Serverless · Planet-scale

FAMILLE 2

API de features

OGC API Features · WFS · FastAPI

Un service HTTP répond à des requêtes avec filtres, pagination et sélection attributaire. La famille la plus interactive, mais aussi la plus sensible à la charge.

REST · Filtres · Attributs

FAMILLE 3

Tile server dynamique

Martin · pg_tileserv · TileServer GL

Le serveur génère des tuiles vectorielles ou raster à la volée depuis une base de données. Architecture de référence pour les données à forte volumétrie avec mise à jour fréquente.

MVT · PostGIS · Cache

FAMILLE 4

Service de rendu image

WMS · WMTS · OGC API Tiles

Le serveur génère une image cartographique côté serveur (PNG, JPEG). Architecture historique des standards OGC, encore dominante dans les contextes institutionnels INSPIRE.

OGC · INSPIRE · Legacy

FAMILLE 5

SaaS géré

Mapbox · Maptiler · Felt · Google Maps

Une plateforme cloud gère l'intégralité de la chaîne. Coût zéro ops contrebalancé par la dépendance à un provider et les implications RGPD. Deux rôles : fond de carte seul, ou plateforme complète avec données métier.

Zéro ops | RGPD ⚠️

FAMILLE 6

Big Data / analytique

DuckDB · BigQuery Geo · Spark Sedona

Le moteur de requête est directement exposé comme couche d'accès. SQL spatial massif sans tile server. Latence analytique (secondes), pas cartographique (millisecondes).

GeoParquet | OLAP | SQL

1.2 Les 5 critères de choix

Latence

Le temps de réponse acceptable varie radicalement. Un fond de carte doit répondre en moins de 100 ms. Une requête analytique peut tolérer 2 à 5 secondes. Les tuiles sur CDN (edge) offrent les latences les plus basses. Les APIs WMS avec rendu serveur ont les latences les plus élevées.

Volume

Quelques centaines de POI s'exposent en GeoJSON via API REST. Un million de bâtiments nécessite des tuiles vectorielles. Des données satellitaires à téraoctets exigent des formats cloud-native avec accès partiel (COG, GeoParquet).

Interactivité

Un fond raster (WMS, XYZ) ne permet aucune interactivité sur les entités. Les tuiles vectorielles (MVT, MLT) permettent le clic et le survol côté client. Une API de features permet le filtre serveur.

Coût

La plage va de zéro (PMTiles sur R2 pour un petit volume) à plusieurs centaines d'euros par mois (tile server dédié haute disponibilité). Inclure le coût d'opération et de maintenance, pas seulement la facture cloud.

Souveraineté

La question RGPD va au-delà du stockage de la donnée métier. Les fonts PBF (glyphs) et sprites servis depuis un CDN Mapbox ou Maptiler envoient l'IP de chaque visiteur à un tiers. L'hébergement complet — tuiles, style, sprites, glyphs — est la seule garantie d'autonomie réelle.

VOLUME D'ENTITÉS	GEOJSON (MS)	WFS (MS)	MVT DYNAMIQUE (MS)	PMTILES CDN (MS)
100	~20	~80	~25	~8
1 000	~30	~120	~30	~10
5 000	~80	~250	~40	~12
10 000	~200	~500	~50	~15
50 000	~1 500	~2 000	~60	~18
100 000	~4 000	~6 000	~80	~20
500 000	N/A	N/A	~90	~22
1 000 000	N/A	N/A	~110	~25
5 000 000+	N/A	N/A	~150	~28

💡 Valeurs indicatives — ordres de grandeur construits à partir des benchmarks qualitatifs de la littérature et de l'expérience terrain. GeoJSON devient inexploitable au-delà de 100 000 entités, WFS au-delà de 50 000 sans cache. PMTiles sur CDN edge reste sous les 30 ms quel que soit le volume.

1.3 Matrice synthétique famille × critères

FAMILLE	LATENCE	VOLUME	INTERACTIVITÉ	COÛT INFRA	SOUVERAINETÉ
Fichier statique (PMTiles/COG)	✅ Très faible (edge)	✅ Planet-scale	⚠️ Limitée	✅ Quasi nul	✅ Self-hosted
API de features (OGC, REST)	⚠️ Variable	⚠️ Limité sans pagination	✅ Totale	⚠️ Serveur requis	✅ Contrôle total
Tile server dynamique	⚠️ Variable (cache)	✅ Élevé	✅ Client complète	⚠️ Serveur requis	✅ Self-hosted
Service de rendu image (WMS)	❌ Élevée (CPU)	⚠️ Moyen avec cache	❌ Aucune sur entités	❌ Infrastructure lourde	✅ Self-hosted
SaaS géré	✅ Excellente (CDN)	✅ Illimité	⚠️ Dépend du service	✅ Zéro ops	❌ Dépendance provider
Big Data / analytique	⚠️ Latence analytique	✅ Téraoctets	✅ SQL à la volée	⚠️ Variable	✅ Self-hosted possible

→ **Point clé**

Le choix du mode d'exposition dépend du moteur de rendu client autant que du backend de stockage. Un client Leaflet ne bénéficiera pas d'un tile server MVT natif. Un client MapLibre GL JS exploite pleinement PMTiles, MVT et MLT. **Le choix du frontend doit précéder le choix du backend d'exposition.**

1993–2004	2005–2013	2014–2019	2020–2026	2026+ ↗
Ère 1 — Serveurs carto	**Ère 2 — Web carto**	**Ère 3 — APIs REST modernes**	**Ère 4 — Serverless / CDN**	**Tendances 2030**
MapServer · GeoServer · WMS · WFS	XYZ tiles · TileCache · Leaflet · OpenLayers	Martin · pg_tileserv · pygeoapi · OGC API	PMTiles · MLT · TiTiler · DuckDB-WASM	WebGPU · MLT dominant · STAC+OGC+PMTiles

Partie 2 — Ère 1 : Les serveurs cartographiques (1993–2004)

Scénario LB2 concerné : S1 (SIG local → exposition web institutionnelle)

Au milieu des années 1990, exposer une carte sur le web est une entreprise technique non triviale. Le web grand public est naissant, HTTP 1.0 date de 1996, et les navigateurs ne savent rien faire d'autre qu'afficher du HTML statique et des images. La cartographie web repose entièrement sur un paradigme simple : le serveur fait tout le travail de rendu, le navigateur reçoit une image.

MapServer (1994 — UMN ForNet / NASA)

MapServer naît en 1994 à l'Université du Minnesota (UMN), dans le cadre d'un projet de recherche financé par la NASA et le Minnesota Department of Natural Resources : le projet ForNet. Le premier programme est un script C qui génère des appels ArcPlot, avec des temps de rendu de l'ordre de 30 secondes par requête. En 1995, le passage à la bibliothèque GD pour la génération d'images marque un bond de performance. MapServer fonctionne selon le modèle CGI : chaque requête HTTP déclenche l'exécution d'un processus C qui lit un Mapfile, charge les données spatiales, effectue le rendu en mémoire, et retourne une image PNG ou JPEG.

MAPFILE

```
MAP
  NAME "exemple"
  STATUS ON
  SIZE 800 400
  EXTENT -5.5 41.0 10.0 51.5
  PROJECTION
    "init=epsg:4326"
  END
  LAYER
    NAME "departements"
    TYPE POLYGON
    DATA "/data/departements.shp"
    STATUS ON
    CLASS
      STYLE
        COLOR 200 200 200
        OUTLINECOLOR 100 100 100
      END
    END
  END
END
```

Statut 2026 : MapServer est toujours maintenu activement (version 8.4, janvier 2025). Il occupe une niche stable dans les portails cartographiques institutionnels et les projets legacy. Écrit en C, il reste l'un des moteurs de rendu cartographique les plus rapides disponibles.

GeoServer (2001 — The Open Planning Project)

GeoServer naît en 2001 à New York, au sein de The Open Planning Project (TOPP). Contrairement à MapServer, il est conçu comme un serveur de données géospatiales interopérable, dont la vocation est de publier des données en open access via des standards OGC. Application web Java (J2EE) déployable dans Tomcat, GeoServer dispose d'une interface d'administration web complète et est l'implémentation de référence du standard WFS de l'OGC.

CRITÈRE	MAPSERVER	GEOSERVER
Langage	C	Java
Configuration	Mapfile texte déclaratif	Interface web + SLD
Performance rendu	✅ Excellente (C natif)	⚠️ Correcte (JVM overhead)
Conformité OGC	✅ Bonne	✅ Excellente (reference impl. WFS)
OGC API moderne	⚠️ Support partiel	✅ Stable depuis GeoServer 2.19+
Interface admin	❌ Fichier texte uniquement	✅ Interface web complète
Cas d'usage 2026	Portails legacy, performance brute	Portails OGC, INSPIRE, interopérabilité

Les standards OGC — WMS, WFS, WMTS

L'OGC publie dans cette période les trois standards fondateurs de l'exposition cartographique web normalisée.

WMS (2000)	Web Map Service — protocole d'accès aux cartes en tant qu'images générées à la demande. Trois opérations : GetCapabilities, GetMap, GetFeatureInfo. Limite : couplage fort requête/rendu, mise en cache difficile.	**OGC std.**
WFS (2002)	Web Feature Service — expose les données vectorielles brutes (géométries + attributs). Version transactionnelle WFS-T pour l'édition. Peu scalable pour forte audience. Remplacé progressivement par OGC API Features.	**OGC std.**
WMTS (2010)	Web Map Tile Service — résout la limite WMS en précalculant les tuiles raster dans une grille fixe (TileMatrix). Permet la mise en cache complète. Supplanté progressivement par OGC API Tiles (2022).	**OGC std.**

💡 **PostGIS comme point d'entrée de l'exposition**

PostGIS (2001) n'est pas seulement un moteur de stockage spatial — deux fonctions constituent le pont direct vers la couche exposition : **ST_AsGeoJSON(geom)** pour les APIs REST et OGC API Features, et **ST_AsMVT(row, layer_name)** pour la génération directe de tuiles vectorielles MVT. Ces fonctions illustrent un principe fondamental : le moteur de base de données spatiale est aussi un moteur de sérialisation.

Partie 3 — Ère 2 : Le web cartographique et les tile servers (2005–2013)

🔗 Scénarios LB2 concernés : S1 (GeoPackage → exposition web), S2 (PostGIS → tile server)

Le 8 février 2005, Google Maps est lancé aux États-Unis. Ce n'est pas simplement un nouveau service de cartographie en ligne : c'est une rupture architecturale qui va transformer l'ensemble de l'écosystème géospatial web pour les vingt années suivantes.

Google Maps et la révolution du tuilage (2005)

Google Maps introduit deux innovations conjointes qui créent l'expérience de la "slippy map" : le **tuilage précalculé** (la carte mondiale découpée en millions de petites images 256×256 px organisées en niveaux de zoom XYZ) et le **chargement AJAX asynchrone** (les tuiles se chargent en arrière-plan sans rechargement de page). Le schéma d'adressage `/{z}/{x}/{y}.png` devient un standard de facto mondial — jamais formalisé par l'OGC, mais adopté par OpenStreetMap depuis 2007 et tous les frameworks cartographiques.

> ⚠️ **Impact architectural immédiat**
>
> Le WMS dynamique ne peut pas concurrencer un CDN distribuant des images statiques précalculées en termes de latence et de scalabilité. La question n'est plus "comment rendre la carte" mais "comment précalculer et distribuer les tuiles". C'est ce changement de paradigme qui donne naissance aux tile caches et, quelques années plus tard, aux tile servers modernes.

TileCache, TileStache, MapProxy — les premiers tile caches

TileCache (2006)	MetaCarta — proxy interposé devant un WMS existant. Le client demande une tuile XYZ, TileCache la génère depuis le WMS et la met en cache (disque, mémoire, S3). Permet de transformer n'importe quel WMS en source XYZ cachable sans modifier le serveur.
TileStache (2010)	Stamen Design — évolution de TileCache, plus flexible dans ses sources (WMS, PostGIS direct, raster). Intègre un prérendu asynchrone. Aujourd'hui en maintenance réduite.
MapProxy (2009)	Omniscale — proxy-cache mature acceptant WMS, WMTS, TMS, ArcGIS Server et produisant WMS, WMTS, TMS, XYZ — avec reprojection à la volée. Toujours actif en 2026 pour les architectures hybrides qui maintiennent des WMS institutionnels legacy.

TileServer GL — les tuiles vectorielles en production

TileServer GL, développé par Jiří Koníček (MapTiler), est publié en 2016. Il prend en entrée des fichiers MBTiles ou PMTiles et génère des tuiles MVT ou des images raster, avec une interface d'administration web intégrée. C'est le premier tile server open source à servir directement des tuiles vectorielles depuis des fichiers sans dépendance à une base de données. Il expose aussi un endpoint de rendu image statique — traité en Partie 9.

Leaflet (2011) et OpenLayers

Créé par Volodymyr Agafonkin en 2011, Leaflet est une bibliothèque cartographique JavaScript légère et accessible. Son architecture Canvas/SVG, sans WebGL, est à la fois sa force et sa limite : excellente pour les tuiles raster XYZ, WMS natif, et GeoJSON léger ; aucun support natif MVT, MLT, ou PMTiles. **Conséquence architecturale directe :** choisir Leaflet comme client rend un tile server MVT architecturalement inutile. OpenLayers, plus complet et plus verbeux, supporte WMS, WFS, WMTS, OGC API, et MVT via ol/format/MVT.

Partie 4 — Ère 3 : APIs REST et tile servers modernes (2014–2019)

Scénarios LB2 concernés : S2 (PostGIS → Martin/pg_tileserv), S1/S2 (GeoServer/pygeoapi)

2014 marque une rupture double : Mapbox publie les premières spécifications MVT et le client WebGL Mapbox GL JS. La combinaison tuile vectorielle + rendu GPU côté client change le paradigme : le serveur n'a plus besoin de styliser la carte — il se contente de fournir les données géométriques brutes, le style est appliqué côté client.

pg_tileserv et Martin — la génération ST_AsMVT

pg_tileserv	CrunchyData (Go) — tile server minimaliste qui expose automatiquement toutes les tables et vues PostGIS avec des colonnes géométriques. Zéro configuration : il détecte les couches disponibles via introspection PostgreSQL. Idéal pour le prototypage rapide.	
Martin v1.x (Rust)	Organisation MapLibre — tile server Rust haute performance. Supporte PostGIS (ST_AsMVT), PMTiles, MBTiles. Function layers paramétrés pour les tiles multi-tenant avec RLS. Version 1.0.0 publiée le 19 novembre 2025. Le choix recommandé en 2026 pour la production.	**v1.0 — 2025**

SQL — ST_ASMVT

```sql
-- PostGIS génère une tuile MVT depuis une table
SELECT ST_AsMVT(q, 'communes', 4096, 'geom')
FROM (
  SELECT id, nom, population,
         ST_AsMVTGeom(
           geom,
           ST_TileEnvelope(z, x, y),
           4096, 64, true
         ) AS geom
  FROM communes
  WHERE ST_Intersects(geom, ST_TileEnvelope(z, x, y))
    AND population > 0
) q;
```

pygeoapi — le serveur Python OGC API

pygeoapi implémente la famille complète des standards OGC API (Features, Tiles, Records, Coverages, EDR) en Python avec Flask/Starlette. Configuration YAML, certifié OGC Compliant pour OGC API Features depuis 2020. Son positionnement est complémentaire de Martin : pygeoapi gère les APIs features interactives (filtres, pagination, CQL2) là où Martin gère les tiles haute performance.

OGC API Features (2019) et OGC API Tiles (2022)

CRITÈRE	WFS / WMTS (LEGACY)	OGC API FEATURES / TILES (MODERNE)
Encodage	XML (GML / Capabilities)	JSON + OpenAPI
URLs	Paramètres GET complexes	URLs lisibles REST
Documentation	❌ Manuelle	✅ OpenAPI auto-générée
Filtrage	⚠️ OGC XML filters	✅ CQL2 expressif
Support vectoriel (tiles)	❌ Raster uniquement (WMTS)	✅ MVT, MLT, raster
Reco. nouveaux projets	❌ Legacy / INSPIRE existant	✅ Standard recommandé

✅ **L'architecture trois couches découplées**

L'émergence de pg_tileserv, Martin et pygeoapi consacre le passage du monolithe GeoServer à une architecture découplée : **[Stockage PostGIS]** → **[Service Martin/pygeoapi]** → **[Rendu MapLibre GL JS]**. Chaque couche peut être remplacée indépendamment. C'est la raison pour laquelle le choix du moteur de rendu client doit précéder le choix du tile server.

Partie 5 — Les moteurs de rendu client et leurs implications serveur

MOTEUR	RENDU	MVT NATIF	MLT NATIF	PMTILES NATIF	WMS NATIF	LICENCE 2026	RECO.
MapLibre GL JS	WebGL / WebGPU	✅	✅ v5.12+	✅ natif	❌	BSD-3 Open Source	✅ Nouveaux projets
Mapbox GL JS v2+	WebGL	✅	❌	✅ v3+	❌	Propriétaire	⚠️ Projets Mapbox existants
Leaflet	Canvas / SVG	⚠️ Plugin	❌	❌	✅	BSD-2 Open Source	⚠️ Données légères / WMS
OpenLayers	Canvas / WebGL	✅ ol/format/MVT	❌	❌	✅	BSD-2 Open Source	⚠️ Portails OGC / INSPIRE
deck.gl	WebGL / WebGPU	✅	❌	❌	❌	MIT Open Source	⚠️ Big Data / Analytique

💡 **MapLibre GL JS en 2026 — le moteur de référence self-hosted**

MapLibre GL JS naît en janvier 2021 comme fork communautaire de Mapbox GL JS 1.13 (dernière version BSD-3). Il supporte nativement PMTiles (sans plugin), MLT depuis la **version 5.12** (novembre 2025) via la propriété `"encoding": "mlt"` dans le style JSON, et ne dépend d'aucun provider tiers. Son backend naturel : Martin (tiles dynamiques), PMTiles sur CDN (tiles statiques), pygeoapi ou FastAPI (features interactives), TiTiler (raster COG).

Partie 6 — Le style cartographique : sprites, glyphs et autonomie complète

Le style GL JSON décrit l'intégralité du rendu cartographique (sources de données, couches, couleurs, fonts, icônes) dans un document JSON déclaratif. Il est consommé par MapLibre GL JS et Mapbox GL JS. Mais ce style référence deux ressources servies qui sont souvent oubliées dans l'architecture : les **sprites** (icônes vectorielles PNG + JSON descriptor) et les **glyphs** (fonts PBF pour le rendu des labels).

> 🚫 **Le piège RGPD — dépendances silencieuses**
>
> Un antipattern très fréquent et souvent invisible : le style JSON référence les glyphs via `"glyphs": "https://api.mapbox.com/fonts/v1/mapbox/..."`. Chaque fois qu'un utilisateur charge la carte, MapLibre envoie des requêtes HTTP vers Mapbox pour télécharger les fonts — transmettant l'adresse IP du visiteur à Mapbox, même si toutes les données métier sont auto-hébergées. Test de conformité RGPD : ouvrir les DevTools, onglet Network, vérifier que zéro requête part vers mapbox.com, maptiler.com, googleapis.com.

Checklist d'autonomie complète — zéro dépendance externe

RESSOURCE	TECHNOLOGIE RECOMMANDÉE	SOURCE OPEN
`Tuiles vectorielles`	PMTiles sur R2, ou Martin	OpenMapTiles via Planetiler
`Tuiles raster fond`	PMTiles raster, ou TiTiler + COG	Copernicus, SRTM
`Style JSON`	Fichier statique sur CDN ou Martin	OpenMapTiles styles
`Sprites (icônes)`	Fichiers statiques sur CDN	openmaptiles/maptiler-basic-gl-style
`Glyphs (fonts PBF)`	Fichiers statiques sur CDN	openmaptiles/fonts
`Terrain (optionnel)`	COG + TiTiler	SRTM / Copernicus DEM

Maputnik est l'éditeur visuel de référence pour les styles GL JSON — application React statique auto-hébergeable, avec prévisualisation temps réel, connexion à Martin ou PMTiles local, et export JSON en un clic. Élevé au rang de projet "supported" MapLibre en 2025.

Partie 7 — Ère 4 : Serverless, Cloud-Native et CDN (2020–2026)

Scénarios LB2 concernés : S3 (GeoParquet → PMTiles), S4 (PMTiles sur R2)

La quatrième rupture de l'exposition géospatiale ne vient pas d'un nouveau format ou d'un nouveau moteur de rendu — elle vient de l'élimination du serveur lui-même. L'idée que des tuiles cartographiques puissent être servies depuis un fichier statique sur un stockage objet, accessibles directement par le navigateur via HTTP Range Requests, sans aucune infrastructure applicative, représente un changement de paradigme aussi profond que le passage du WMS au tuilage XYZ en 2005.

PMTiles v3 — la tuile sans serveur

PMTiles est créé par Brandon Liu (projet Protomaps). La version 3, publiée en octobre 2022, stabilise l'approche HTTP Range Requests avec un système d'adressage en courbe de Hilbert et une déduplication interne des tuiles. Le principe : un fichier PMTiles est une archive binaire organisée en header fixe (127 octets), un répertoire d'index structuré selon la courbe de Hilbert, et les données de tuiles avec déduplication. Pour récupérer une tuile, MapLibre envoie 2 requêtes HTTP Range (3 max pour les fichiers planet-scale). PMTiles est format-agnostique : le conteneur peut héberger des tuiles MVT, MLT, ou raster PNG/JPEG/WebP.

2
requêtes HTTP Range pour récupérer une tuile

127
octets — header fixe PMTiles v3

0 €
egress Cloudflare R2 → réseau Cloudflare

Tippecanoe et Planetiler — les générateurs de tiles

Tippecanoe	Felt (fork de Mapbox) — outil CLI de génération de tuiles vectorielles MVT depuis GeoJSON ou GeoJSONSeq. Produit PMTiles ou MBTiles. Référence pour les jeux de données régionaux ou nationaux. Supporte les agrégations par densité, la simplification géométrique adaptative par zoom, et les expressions de filtre.
Planetiler	Java, OpenMapTiles — génère des PMTiles planet-scale depuis Overture Maps ou OpenStreetMap. Optimisé pour la génération de fond de carte mondial en quelques heures sur un VPS standard. Supporte MVT aujourd'hui, support MLT prévu dans les versions à venir.

TiTiler — les tuiles raster serverless

TiTiler (Development Seed, Python/FastAPI + Rasterio) génère des tuiles raster dynamiques depuis des COG (Cloud Optimized GeoTIFF) hébergés sur S3 ou R2, sans pré-génération. Chaque tuile est calculée à la demande par une HTTP Range Request sur le COG. Déployable en Lambda ou Cloud Run — réellement serverless. Cas d'usage : orthophotos, images satellites, cartes de chaleur, MNT.

FlatGeobuf — le vecteur cloud-native

FlatGeobuf est un format de fichier vectoriel binaire basé sur FlatBuffers, avec un index spatial R-tree intégré qui permet des HTTP Range Requests ciblées — similaire au principe COG/PMTiles mais pour les features vectorielles individuelles. Cas d'usage : distribution de données vectorielles statiques volumineuses sans tile server, avec accès bbox ciblé depuis un CDN.

Partie 8 — Big Data géospatial : DuckDB, Spark Sedona, BigQuery Geo

🔗 Scénarios LB2 concernés : S3 (GeoParquet + DuckDB), S5 (Lac de données enterprise)

Le Big Data géospatial change la couche d'exposition : le moteur de requête devient directement la couche d'accès. Pas de tile server intermédiaire, pas de sérialisation MVT préalable — le client envoie une requête SQL ou un filtre, le moteur retourne un agrégat GeoJSON ou un GeoParquet filtré en réponse.

MOTEUR	CAS D'USAGE PRINCIPAL	INTERFACE D'EXPOSITION	LATENCE TYPIQUE	RECO. 2026
DuckDB + extension spatial	Analytique interactive, ETL, requêtes GeoParquet	FastAPI (résultats GeoJSON), CLI	4 s / 1M entités (scan)	✅ PME, ETL, data journalism
DuckDB-WASM	Analytique directement dans le navigateur	JavaScript inline — GeoParquet sur R2	Variable (RAM navigateur)	✅ Dashboards légers 2026
Apache Spark + Sedona	Volumes nationaux / planétaires, batch	Spark REST API, JDBC, Delta Lake	Minutes	⚠️ Enterprise uniquement
BigQuery Geo	Analytique managée, volumes > 1 To	BigQuery REST API, Looker Studio	Secondes-minutes	⚠️ GCP / coût à surveiller

DuckDB — le moteur analytique in-process

DuckDB v1.0.0 est publié le 3 juin 2024. Son extension `spatial` supporte les types géométriques WKT/WKB, les fonctions ST_* usuelles (ST_Intersects, ST_Buffer, ST_Distance…), et la lecture directe de GeoParquet depuis S3 ou R2 via le connecteur HTTP. Benchmark indicatif sur 1M d'entités GeoParquet : agrégation spatiale en 4 secondes versus 45 secondes sur PostGIS sans index (requête de scan complet). PostGIS reprend l'avantage sur les requêtes indexées (<200 ms).

DUCKDB — SQL GÉOSPATIAL

```
-- Requête DuckDB sur GeoParquet hébergé sur Cloudflare R2
LOAD spatial;

SELECT
  departement,
  COUNT(*) AS nb_parcelles,
  SUM(superficie_ha) AS superficie_totale
FROM read_parquet('https://r2.example.com/parcelles_2026.parquet')
WHERE ST_Intersects(
  geometry,
  ST_GeomFromText('POLYGON((2.0 48.5, 3.0 48.5, 3.0 49.0, 2.0 49.0, 2.0 48.5))')
)
GROUP BY departement
ORDER BY superficie_totale DESC;
```

Partie 9 — Rendu serveur-side (SSR) : cartes PDF et exports automatisés

> 🔗 Cas d'usage typique MP-i : générer une carte PDF dans un rapport CDC ou un rapport Odoo

Le rendu serveur-side répond à quatre cas d'usage : export PDF avec carte intégrée (rapports clients, CDC), génération de thumbnails géographiques, pipelines automatisés (cartes de situation récurrentes), et environnements sans navigateur (scripts CLI, workers Lambda).

OUTIL	QUALITÉ VISUELLE	COMPLEXITÉ SETUP	SERVERLESS NATIF	RECO. PAR CAS D'USAGE
TileServer GL	✅ Haute (GL JSON natif)	⚠️ Moyenne (Docker)	❌ Node.js requis	Rapport PDF pipeline FastAPI
MapLibre Native headless	✅ Maximale	❌ Élevée (GPU/SwiftShader)	❌ Config GPU	Export haute qualité A3/impression
staticmap (Python)	⚠️ Basique (raster)	✅ Faible (pip install)	✅ Lambda compatible	Thumbnails, Lambda légère
Playwright headless	⚠️ Bonne (navigateur réel)	✅ Faible	❌ Chromium lourd (~300 Mo)	Prototype rapide
Mapbox Static Images API	✅ Haute	✅ Nulle	✅	Prototype sans infrastructure

Partie 10 — Les solutions SaaS cartographiques

⚠️ **Deux rôles à ne pas confondre**

Rôle 1 — Source de fond de carte uniquement : Maptiler Cloud ou Mapbox fournit les tuiles du fond de carte mondial. La donnée métier reste entièrement self-hosted. Le SaaS ne voit que les requêtes de tuiles du fond.
Rôle 2 — Plateforme complète : le SaaS gère stockage de la donnée métier, génération des tuiles, CDN. La donnée métier est confiée au provider. Les implications en termes de coût, de souveraineté et de dépendance sont radicalement différentes.

Mapbox	Fondateur de l'écosystème MVT/GL JS. Facturation au map load (50 000 gratuits/mois). Token obligatoire révélant l'usage à Mapbox. Forces : écosystème complet (Studio, SDK mobile, Navigation, géocodage). Dominant pre-2021, MapLibre capte une part croissante des nouveaux projets.	**$ map loads**
Maptiler	Organisation maintenant OpenMapTiles. Deux produits : Maptiler Cloud (SaaS, API key, CDN mondial, tier gratuit généreux) et Maptiler Server (on-premise, hébergement de PMTiles OpenMapTiles en self-hosted). Alternative open source propre à Mapbox pour le fond de carte.	**Freemium / On-prem**
Felt	Produit SaaS orienté collaboration et analyse spatiale. Utilise PMTiles en production pour les données uploadées. Financement Andreessen Horowitz. Positionné sur la collaboration asynchrone sur données géographiques.	**SaaS / équipes**

Partie 11 — Sécurité, gouvernance et RGPD

CORS — prérequis fondamental

NGINX — CORS TILES

```
location /tiles/ {
    add_header Access-Control-Allow-Origin "https://mon-app.example.com";
    add_header Access-Control-Allow-Methods "GET, HEAD, OPTIONS";
    add_header Access-Control-Allow-Headers "Range, Content-Type";
    add_header Access-Control-Expose-Headers "Content-Range, Content-Length, ETag";
    proxy_pass http://martin:3000/;
}
```

Rate limiting et protection des APIs

NGINX — RATE LIMITING

```
limit_req_zone $binary_remote_addr zone=tiles:10m rate=10r/s;

server {
    location /tiles/ {
        limit_req zone=tiles burst=50 nodelay;
        proxy_pass http://martin:3000/;
    }
}
```

Row-Level Security (RLS) et tiles multi-tenant

Martin supporte les function layers PostGIS — des fonctions SQL qui reçoivent les paramètres Z/X/Y et tout paramètre URL supplémentaire. Ce mécanisme permet des tiles filtrées par utilisateur via RLS : le proxy d'authentification valide le JWT, extrait le `user_id` comme paramètre URL, et PostGIS applique le RLS via `set_config('app.current_user_id', user_id, true)` avant d'exécuter ST_AsMVT.

1. Un fichier PMTiles sur R2 est accessible publiquement sans authentification par défaut — utiliser Cloudflare Workers pour ajouter une validation JWT avant de servir les tiles si les données sont sensibles.

2. Loguer les coordonnées Z/X/Y avec l'IP de l'utilisateur — les logs de requêtes tiles révèlent la zone géographique consultée. Ne loguer que des métriques agrégées (nombre de requêtes par niveau de zoom).

3. Oublier de self-héberger les glyphs et sprites — chaque chargement de carte transmet l'IP du visiteur à Mapbox ou Maptiler si les fonts sont référencées depuis leurs CDN.

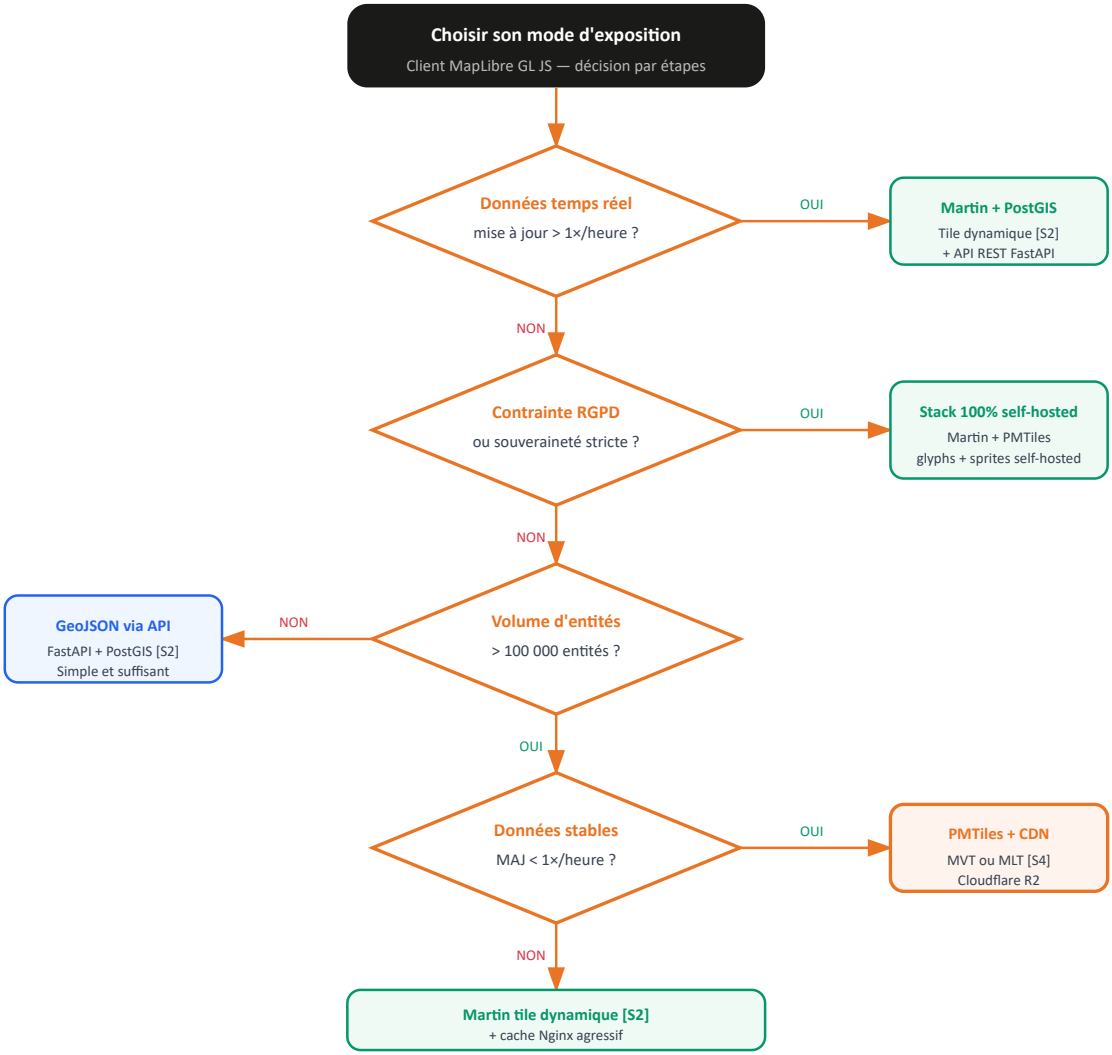
Partie 12 — Guide décisionnel complet

12.1 Matrice d'usage — cas d'usage → solution

CAS D'USAGE	VOLUME DONNÉES	SOLUTION RECOMMANDÉE	SCÉNARIO LB2
POI simples, interactivité basique	< 5 000 entités	API REST GeoJSON (FastAPI + PostGIS)	S2
Application web interactive, données dynamiques	< 100 000 entités	Martin (MVT dynamique) + PostGIS	S2
Portail institutionnel OGC / INSPIRE	Variable	GeoServer + pygeoapi (OGC API Tiles)	S1/S2
Cartographie haute perf., données stables	> 100 000 entités	PMTiles + MVT sur CDN	S4
Haute perf. WebGPU / planet-scale	Tout volume	PMTiles + MLT	S4
Fond de carte mondial self-hosted	Planet-scale	OpenMapTiles PMTiles auto-hébergé	S4
Analytics spatiale sur gros volumes	> 1M entités	DuckDB + GeoParquet → PMTiles	S3
Prototype sans infrastructure	Tout	Mapbox ou Maptiler Cloud	—
Raster cloud (orthophotos, satellite)	Tout	COG + TiTiler serverless	S3/S5
Export PDF avec carte / SSR	Tout	TileServer GL endpoint statique	S2
Stack 100% self-hosted, RGPD strict	Tout	PMTiles + Martin + glyphs/sprites self-hosted	S2 + S4

12.2 Arbre de décision

ARBRE DE DÉCISION — CHOISIR SON MODE D'EXPOSITION (MAPLIBRE GL JS COMME CLIENT)



12.3 Comparatif coûts réels — exposition et stockage combinés

PROFIL	TILE REQUESTS / JOUR	STOCKAGE (LB2)	PMTILES SUR R2	MARTIN VPS	TOTAL PMTILES	TOTAL MARTIN
Petit projet	10 000	~5 €/mois	~0 €	~10 €/mois	~5 €/mois	~15 €/mois
Projet moyen	1 000 000	~30–40 €/mois	~3–5 €/mois	~20–40 €/mois	~35–45 €/mois	~50–80 €/mois
Grand projet	100 000 000	~200 $/mois	~200–400 $/mois	~200 $ + scaling	~400–600 $/mois	~400 $/mois

À fort volume, Martin avec cache agressif peut égaler PMTiles en coût total. La différence se joue sur la latence (edge CDN vs serveur centralisé) et les mises à jour (Martin = temps réel, PMTiles = régénération). Tarification R2 2026 : ~0,015 $/Go/mois, tier gratuit : 10 Go + 10M opérations/mois.

12.4 Les 7 scénarios architecturaux

S1 **Application web légère (< 5 000 entités)**

PostGIS → FastAPI (ST_AsGeoJSON) → GeoJSON → MapLibre GL JS + PMTiles fond de carte OpenMapTiles self-hosted

S2 **Portail institutionnel OGC**

GeoServer 2.x + pygeoapi (OGC API Features + Tiles) → OpenLayers. Environnements INSPIRE avec contrainte d'interopérabilité OGC formelle.

S3 **Application haute performance, données dynamiques**

Martin v1.x (ST_AsMVT → MVT dynamique) + PMTiles fond de carte OpenMapTiles → MapLibre GL JS. Cache Nginx + Redis optionnel.

S4 **Application haute performance, données statiques (MLT + WebGPU)**

PostGIS → Planetiler (MLT à venir) → PMTiles (MLT) → Cloudflare R2 → MapLibre GL JS v5.12+. Architecture cible 2027–2028.

S5 **Pipeline Big Data géospatial**

DuckDB → Tippecanoe → PMTiles vectoriel + TiTiler (raster COG) → deck.gl / MapLibre GL JS. Volumes > 1M entités.

S6 **Stack 100% self-hosted, zéro dépendance externe**

PMTiles OpenMapTiles auto-hébergé + Martin v1.x + glyphs/sprites/style fichiers statiques + MapLibre GL JS. Maximum RGPD.

S7

Exports PDF / rendu SSR automatisé

PostGIS + PMTiles + TileServer GL (endpoint /static/...) + FastAPI (pipeline rapport PDF). Cas d'usage type MP-i : CDC, rapports Odoo géolocalisés.

Partie 13 — Architectures de référence 2026

Stack Standard — PME / Startup

30–60 € / mois

Application cartographique web avec données métier dans PostGIS, sans contrainte OGC stricte.

- Stockage : LB2 Stack PME (PostGIS + R2)
- Tile server dynamique : Martin v1.x (Rust)
- Distribution tiles statiques : PMTiles sur Cloudflare R2
- API features : FastAPI + PostGIS
- Style + sprites + glyphs : fichiers statiques sur R2
- Frontend : MapLibre GL JS
- Fond de carte : OpenMapTiles PMTiles self-hosted

Stack Institutionnelle OGC-compliant

Infrastructure dédiée

Organisation publique soumise à des exigences INSPIRE, avec portails de données OGC formels.

- Stockage : LB2 Stack Institutionnelle (PostGIS self-hosted UE)
- Serveur OGC principal : GeoServer 2.x
- API OGC moderne : pygeoapi (Features + Tiles + Records)
- Cache WMS legacy : MapProxy
- Portail données : GeoNode ou CKAN géo
- Frontend : OpenLayers
- Métadonnées : GeoNetwork (CSW/STAC)

Stack Cloud-native haute performance

200–800 $ / mois

Volumes importants, données stables ou lentement évolutives, priorité performance et coût à l'échelle.

- Stockage : LB2 Stack Cloud-native (GeoParquet R2 + PostGIS)
- ETL vers tiles : DuckDB → Tippecanoe / Planetiler
- Tiles vectorielles : PMTiles (MVT → MLT à venir) sur R2
- Tiles raster : COG + TiTiler serverless (Lambda/Cloud Run)
- API features : pygeoapi
- Frontend : MapLibre GL JS v5.12+ (WebGPU natif)
- CDN : Cloudflare R2 + Workers

Stack Big Data analytique

Coût variable (compute)

Données à l'échelle nationale ou mondiale : Overture Maps, données de mobilité, LiDAR, observations satellites.

- Stockage : LB2 Stack Big Data (Spark + GeoParquet + PostGIS)
- Pipeline tiles : Spark Sedona → GeoParquet → Tippecanoe → PMTiles
- SQL interactif : DuckDB (ad hoc) ou BigQuery Geo (managé)
- API résultats : FastAPI + GeoJSON agrégé
- Visualisation : deck.gl ou MapLibre GL JS

Partie 14 — Tendances 2030

Cette partie projette les trajectoires identifiables à partir de l'état de l'écosystème en mars 2026. Les horizons indiqués sont des estimations fondées sur les signaux observables — roadmaps publiées, communiqués officiels, rythme d'adoption — pas des prédictions.

2026–2027 — CONSOLIDATION

PMTiles standard de facto · Martin v1.x en production large · TiTiler raster cloud-native

PMTiles a atteint un niveau d'adoption qui en fait le standard de facto pour la distribution de tuiles statiques. Felt l'utilise en production, Overture Maps publie ses tilesets en PMTiles, la spécification v3 est enregistrée comme type MIME IANA (`application/vnd.pmtiles`). Martin v1.0 (novembre 2025) entre en production large.

2027–2028 — TRANSITION

OGC API Tiles/Maps remplace WMS/WMTS · DuckDB-WASM maturité production · AlloyDB/pg_duckdb

OGC API Tiles (nov. 2022) et OGC API Maps (Candidate Standard) sont les successeurs REST de WMTS et WMS. GeoServer, pygeoapi et ldproxy les implémentent déjà. DuckDB-WASM atteint la maturité pour les cas d'usage de production (extension spatiale WASM complète). AlloyDB et pg_duckdb convergent vers un seul point d'entrée OLTP+OLAP géospatial.

2028–2029 — RUPTURE WEBGPU

MLT dominant haute performance · MapLibre GL JS v6+

MLT est conçu pour WebGPU : son encodage column-oriented produit des buffers chargés directement dans les tampons GPU sans transformation intermédiaire, là où MVT nécessite un décodage Protocol Buffers → JavaScript → GPU. WebGPU est disponible dans Chrome 113 (mai 2023), Safari 26 (automne 2025), Firefox 141 (juillet 2025).

2027–2028 — CONVERGENCE

Architecture unifiée : STAC + OGC API + PMTiles

STAC API pour cataloguer les assets (COG, GeoParquet, PMTiles), OGC API Features/Tiles pour l'accès programmatique standardisé, PMTiles pour la distribution edge. Les briques : stac-fastapi, pygeoapi, TiTiler, GeoNetwork. Architecture de référence des portails de données géospatiales cloud à l'horizon 2027–2028.

SOLUTION	TRAJECTOIRE	HORIZON
MLT	Standard WebGPU tile en construction	2028–2029 : dominant haute perf.
PMTiles	Standard de facto serverless — déjà en production large	Consolidation 2026–2027
MapLibre GL JS	Moteur de référence open source self-hosted	Déjà dominant en 2026
Martin v1.x	Tile server de référence production (v1.0 nov. 2025)	Production large dès 2026
OGC API Tiles/Maps	Successeurs institutionnels de WMTS/WMS	Remplacement progressif 2027–2028
DuckDB-WASM géo	Analytique navigateur sans backend	Maturité production 2027–2028
TiTiler serverless	Standard raster cloud — déjà en production	Consolidation 2026
AlloyDB / pg_duckdb	OLTP+OLAP PostGIS unifié	Production large 2027
GeoServer	Déclin lent sur nouveaux projets, stable institutionnel	Niche institutionnelle stable

Conclusion

Trente ans d'exposition géospatiale sur le web se résument en quatre ruptures architecturales successives. Chacune a rendu obsolètes les outils de la précédente, sans pour autant les éliminer complètement — les WMS institutionnels existent encore en 2026 aux côtés des PMTiles edge-native.

1993–2004

Le serveur génère tout

MapServer (1994) construit des images cartographiques à la demande. GeoServer le rejoint en 2001. WMS normalise ce paradigme. Limite : le rendu côté serveur est coûteux, non cachable, non interactif.

2005–2013

Le tuilage découple génération et distribution

Google Maps impose le modèle XYZ. TileCache, TileStache, Mapbox popularisent la génération offline de tuiles et leur distribution via CDN. Leaflet et OpenLayers rendent la cartographie web accessible à tous les développeurs. Limite : les tuiles sont des images mortes, sans interactivité sur les entités.

2014–2019

WebGL délègue le rendu au client

Mapbox GL JS confie le rendu GPU au navigateur. Le serveur ne génère plus des images mais des données vectorielles compactes (MVT via ST_AsMVT). Martin, pg_tileserv, pygeoapi émergent. L'architecture trois couches (stockage / service / rendu) s'impose.

2020–2026

Le serveur lui-même est éliminé pour les cas statiques

PMTiles rend les tuiles accessibles directement depuis un stockage objet via HTTP Range Requests. MLT optimise cette architecture pour WebGPU. DuckDB-WASM déplace l'analytique dans le navigateur. Pour la majorité des cas d'usage statiques, un CDN remplace avantageusement un tile server.

> **La question architecturale fondamentale reste inchangée depuis trente ans : qui consomme, quel volume, quelle interactivité ? Les réponses à ces trois questions déterminent la stack optimale mieux que n'importe quelle tendance technologique.**
>
> — Mattieu Pottier, LB3 — Exposer la donnée géospatiale sur le Web, Mars 2026

Ce document clôt la collection « Chaîne de la donnée géospatiale ». LB1 a établi le substrat : les formats géospatiaux et leurs compromis. LB2 a construit le backend : comment stocker la donnée géospatiale à chaque échelle. LB3 a couvert la couche finale : comment exposer cette donnée au web, avec quels outils, à quel coût, et avec quelles garanties de souveraineté.

L'écosystème continue d'évoluer rapidement — les sections sur MLT, DuckDB-WASM et WebGPU seront les premières à vieillir. Les principes architecturaux (découplage, séparation des fréquences de mise à jour, choix du client avant le serveur) sont, eux, durables.

Références

Tile servers	Martin v1.x : maplibre.org/martin/ · pg_tileserv : github.com/CrunchyData/pg_tileserv · TileServer GL : github.com/maptiler/tileserver-gl · pygeoapi : pygeoapi.io · TiTiler : developmentseed.org/titiler/
Formats & distribution	PMTiles v3 spec : github.com/protomaps/PMTiles · MLT spec : github.com/maplibre/maplibre-tile-spec · Tippecanoe : github.com/felt/tippecanoe · Planetiler : github.com/onthegomap/planetiler
Moteurs de rendu	MapLibre GL JS : maplibre.org/maplibre-gl-js/docs/ · MapLibre Native : github.com/maplibre/maplibre-native · OpenLayers : openlayers.org · Leaflet : leafletjs.com · deck.gl : deck.gl
Style & assets	MapLibre Style Spec : maplibre.org/maplibre-style-spec/ · Maputnik : maputnik.github.io · OpenMapTiles fonts : github.com/openmaptiles/fonts · build_pbf_glyphs : github.com/stadiamaps/build_pbf_glyphs
Standards OGC	OGC API Features : ogcapi.ogc.org/features/ · OGC API Tiles : ogcapi.ogc.org/tiles/ · GeoServer : geoserver.org · MapServer : mapserver.org · MapProxy : mapproxy.org
Big Data géospatial	DuckDB spatial : duckdb.org/docs/extensions/spatial · Apache Sedona : sedona.apache.org · pg_duckdb : github.com/duckdb/pg_duckdb · Cloudflare R2 : developers.cloudflare.com/r2/pricing/
Collection « Chaîne de la donnée géospatiale »	LB1 — Formats géospatiaux 2026 : mp-i.pro · LB2 — Stocker la donnée géospatiale (1970–2026) : mp-i.pro · LB3 — Exposer la donnée géospatiale sur le Web : ce document

Glossaire

COG (Cloud Optimized GeoTIFF)

Format GeoTIFF réorganisé pour des lectures partielles via HTTP Range Requests. Permet de ne télécharger que la portion nécessaire d'une image depuis un stockage objet distant.

CORS

Cross-Origin Resource Sharing — mécanisme HTTP contrôlant les requêtes entre domaines. À configurer sur tout serveur de tiles accessible depuis un navigateur.

Glyphs

Fonts vectorielles encodées en Protocol Buffers (PBF) par plages de 256 caractères Unicode. Utilisées par MapLibre GL JS pour le rendu des labels via la technique SDF.

HTTP Range Requests

Mécanisme HTTP (RFC 9110) permettant de demander une plage d'octets précise d'un fichier distant. Fondement technique de PMTiles et COG.

Martin

Tile server Rust open source de l'organisation MapLibre. Génère des tuiles MVT depuis PostGIS et sert des fichiers PMTiles/MBTiles. v1.0.0 — 19 novembre 2025.

MLT (MapLibre Tile)

Format de tuile vectorielle successeur de MVT, à encodage column-oriented, optimisé pour WebGPU. Spécification stable octobre 2025, annonce publique janvier 2026.

MVT (Mapbox Vector Tile)

Format de tuile vectorielle basé sur Protocol Buffers. Standard de facto pour les tuiles vectorielles depuis 2015.

OGC API Features

Standard OGC REST (2019) pour l'accès aux features géospatiales via JSON. Successeur de WFS.

OGC API Tiles

Standard OGC REST (2022) pour l'accès aux tuiles vectorielles ou raster. Successeur de WMTS.

PMTiles

Format d'archive binaire pour pyramides de tuiles, créé par Brandon Liu (Protomaps). Repose sur HTTP Range Requests pour servir des tuiles depuis un stockage objet sans serveur applicatif. Version 3 — octobre 2022.

R2 (Cloudflare)

Service de stockage objet S3-compatible de Cloudflare. Particulièrement adapté à la distribution de PMTiles grâce à l'absence de frais d'egress vers le réseau Cloudflare.

SDF (Signed Distance Field)

Technique de rendu de texte sur GPU. Permet de rendre du texte à toute taille et rotation sans pixellisation. Utilisée par MapLibre GL JS pour les labels cartographiques.

STAC

SpatioTemporal Asset Catalog — spécification de catalogage d'assets géospatiaux (images, COG, GeoParquet).

ST_AsMVT

Fonction aggregate PostGIS (v2.4, 2017) qui sérialise des lignes de table en tuile MVT binaire. Point de jonction entre le stockage PostGIS et la couche service.

TiTiler

Serveur de tuiles raster dynamiques en Python (FastAPI + Rasterio), Development Seed. Sert des tuiles depuis des COG hébergés sur S3 ou R2.

Tippecanoe

Outil de génération de tuiles vectorielles (MVT), maintenu par Felt. Prend des fichiers GeoJSON en entrée et produit des PMTiles ou MBTiles.

WebGPU

API graphique bas niveau pour navigateurs, successeur de WebGL. Chrome 113 (mai 2023), Safari 26 (automne 2025), Firefox 141 (juillet 2025).

WFS (Web Feature Service)

Standard OGC (2002) pour l'accès aux features géospatiales en GML. Remplacé progressivement par OGC API Features.

WMS (Web Map Service)

Standard OGC (2000) pour la génération d'images cartographiques côté serveur. Encore dominant dans les contextes institutionnels INSPIRE.

Historique des révisions

VERSION	DATE	AUTEUR	MODIFICATIONS
v1.0.0	21 mars 2026	Mattieu Pottier	Version initiale — publication

À propos de l'auteur

Mattieu Pottier

Consultant indépendant en transformation digitale — MP-i · Mattieu Pottier Indépendant

Expert SIG, architecture système et Odoo. Spécialisé dans les ERP (Odoo V18/V19), les systèmes d'information géographique (QGIS, développement Python), et les applications web (WordPress, FastAPI, JavaScript). La collection « Chaîne de la donnée géospatiale » 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. mp-i.pro (https://www.mp-i.pro)