

# **Étude Comparative d'Architectures API**

Modélisation d'un Système d'Information Retail Omnicanal  
Approche par Microservices : SOAP, REST, GraphQL et gRPC

Soutenance de Projet - Architecture Logicielle

# Plan de la présentation

- |                                                                          |       |
|--------------------------------------------------------------------------|-------|
| ① <b>Contexte Fonctionnel</b> – Définition des besoins métier            | 2 min |
| ② <b>Choix Architectural</b> – Justification de l'approche Microservices | 3 min |
| ③ <b>Analyse des Services et Protocoles</b> – SOAP, REST, GraphQL, gRPC  | 6 min |
| ④ <b>Étude Comparative</b> – Performance et cas d'usage                  | 5 min |
| ⑤ <b>Conclusion et Améliorations</b> – API Gateway                       | 2 min |

SOAP

REST

GraphQL

gRPC

# Modélisation du Système d'Information (Retail)

L'objectif est de modéliser le Système d'Information (SI) d'une entreprise opérant dans le commerce de détail (Retail), caractérisé par des flux de données hétérogènes impliquant **quatre domaines métiers distincts** :

## SOAP Domaine 1 : Approvisionnement (Procurement)

Service responsable de la création de bons de commande vers des fournisseurs industriels disposant de systèmes ERP historiques.

## REST Domaine 2 : Réseau de Boutiques (Marketplace)

Service permettant aux points de vente partenaires de synchroniser leurs niveaux de stock avec la plateforme centrale.

## GraphQL Domaine 3 : Reporting et Pilotage

Service exposant une vue consolidée (Tableau de bord) des stocks, commandes et de la logistique à destination de la direction.

## gRPC Domaine 4 : Logistique Entrepôt

Service traitant la télémétrie en temps réel issue de la flotte de robots d'automatisation d'entrepôt.

# Démarche d'Ingénierie : Approche "Contract-First"

La conception du SI a suivi une méthode stricte d'ingénierie dirigée par les contrats :

- ① **Spécification du Contrat** – Définition formelle de l'interface avant toute implémentation.
- ② **Génération/Implémentation** – Le serveur est contraint par les stubs et schémas générés à partir du contrat.
- ③ **Validation** – Vérification systématique de la conformité des échanges.

Module	Le fichier Contrat
SOAP	PurchaseOrder.wsdl
REST	openapi.yaml
GraphQL	schema.graphql
gRPC	warehouse.proto

## Bénéfice Architectural

Le découplage entre la spécification (contrat) et l'implémentation (code) favorise l'indépendance des équipes et la robustesse des systèmes distribués.

# Justification de l'approche Microservices

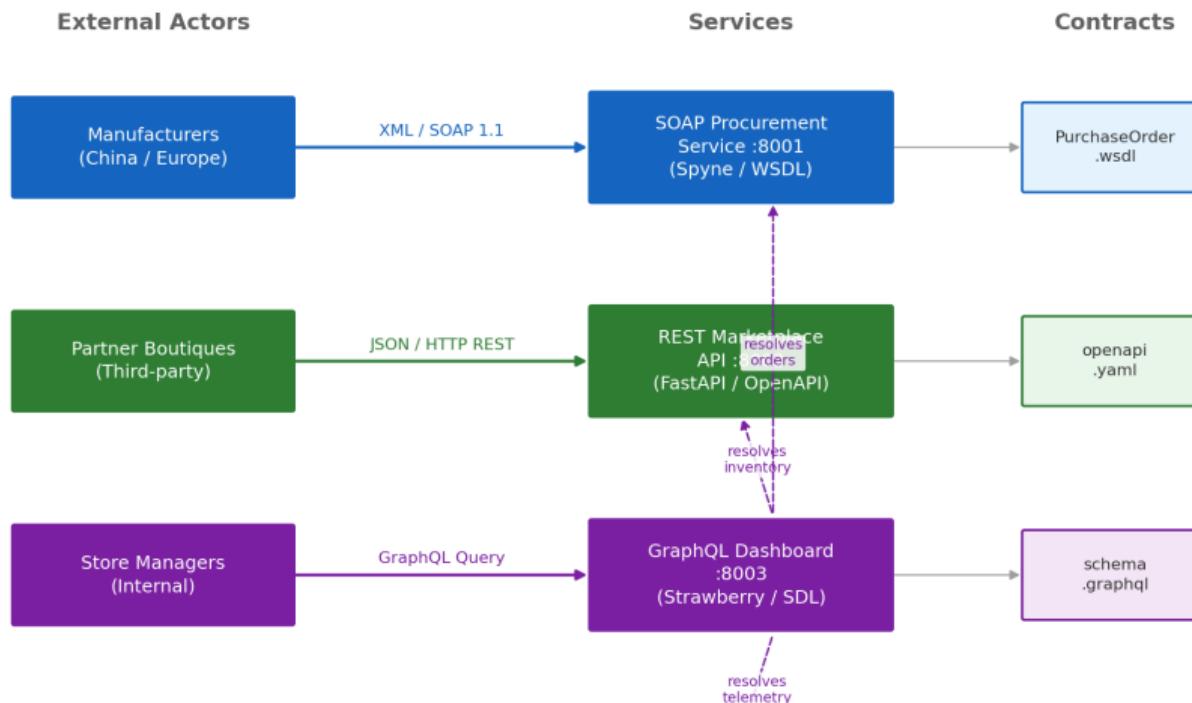
Face à l'hétérogénéité des besoins métier, une architecture monolithique a été rejetée au profit d'une \*\*architecture orientée Microservices\*\*.

## Raisonnement : Pourquoi segmenter en services distincts ?

- **Contraintes Technologiques Incompatibles** : gRPC (HTTP/2 binaire), SOAP (Spécifications XML lourdes) et REST (HTTP/1.1 classique) nécessitent des bibliothèques et des serveurs sous-jacents structurellement différents. Une fusion créerait une dépendance forte et complexe ("anti-pattern").
- **Isolation des Défaillances** : Une surcharge sur le service de télémétrie des robots (gRPC) ne doit pas impacter la création des bons de commande (SOAP). L'isolation par processus métier garantit la haute disponibilité.
- **Déploiement Indépendant** : Permet aux différentes équipes (ex: équipe Logistique vs équipe Partenaires) de déployer leurs évolutions sans risquer de générer des régressions sur les autres services.

# Cartographie Logique du Système

## RetailSync — System Architecture

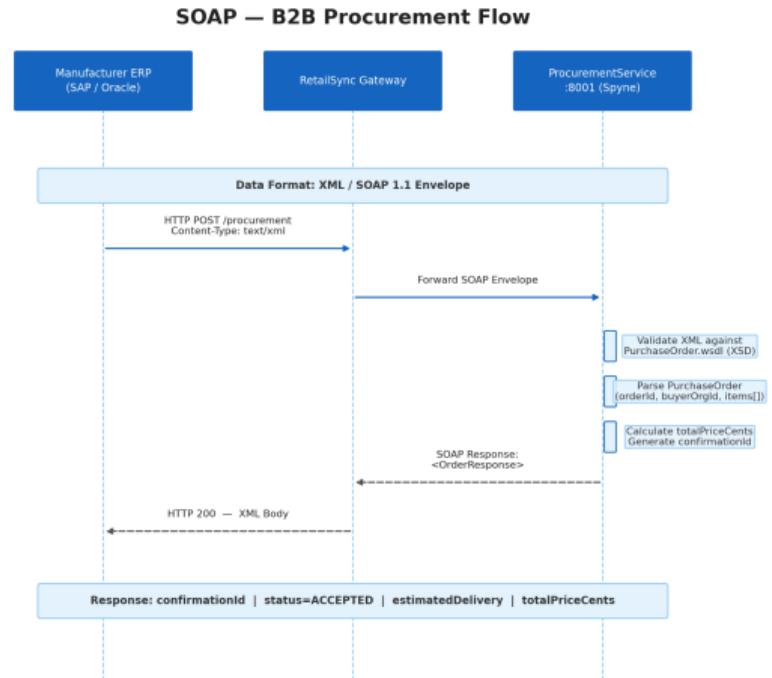


**Service :** Expose une méthode SubmitOrder recevant des bons de commande formatés.

### Motivations du choix de SOAP :

- **Intégrité Stricte (XSD) :** Le contrat WSDL inclut une définition XML Schema qui force l'arborescence, les types, et les contraintes de multiplicité ("Fail-fast").
- **Interopérabilité Legacy :** S'intègre nativement aux systèmes ERP monolithiques des partenaires industriels sans nécessiter de couche de traduction.

+ Garantie structurelle totale	- Verbosité (Parseurs lourds)
+ Sécurité applicative (WSSC)	- Bande passante



# REST Service 2 : Synchronisation des Stocks Boutiques (REST)

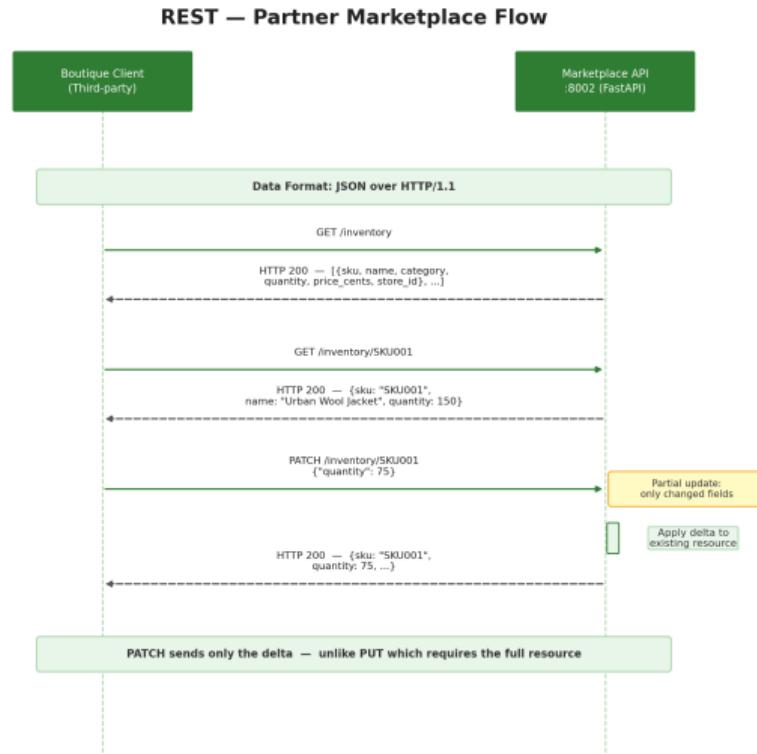
**Service :** Fournit des points de terminaison HTTP classiques (GET /inventory, PATCH /inventory/{sku}).

## Motivations du choix de REST :

- **Réduction de la barrière à l'entrée :**  
Standard du web fondé sur HTTP et JSON, permettant à toute entité tierce de s'intégrer rapidement.
- **Sémantique HTTP native :**  
Utilisation judicieuse de PATCH au lieu de PUT pour moduler uniquement le différentiel de stock, optimisant ainsi les requêtes.

+ Mise en cache native - Typage dynamique

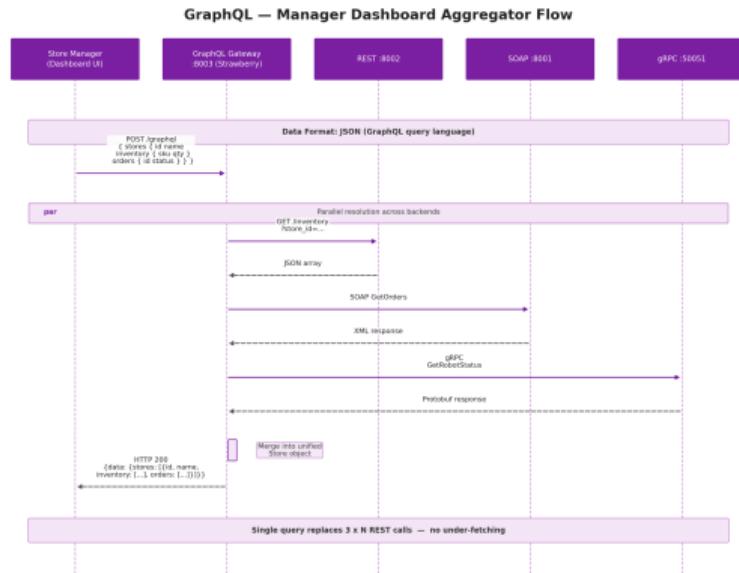
REST vs STPc – Comparaison d'APIs (faible)



**Service :** Point d'entrée unique (`/graphql`) capable d'agréger des graphes d'objets (Magasins, Employés, Commandes).

### Motivations du choix de GraphQL :

- **Résolution du problème du "N+1 Queries"** : Un frontend de reporting complexe évite de lancer N requêtes consécutives vers une API REST, en demandant un arbre de données entier via une unique trame POST.
- **Contrôle côté client (No Over-fetching)** : Le client précise les champs exacts requis pour chaque composant d'interface UI.



**Service :** Gère les communications

*Machine-to-Machine* à haute fréquence entre le serveur et des systèmes cyber-physiques (robots).

### Motivations du choix de gRPC :

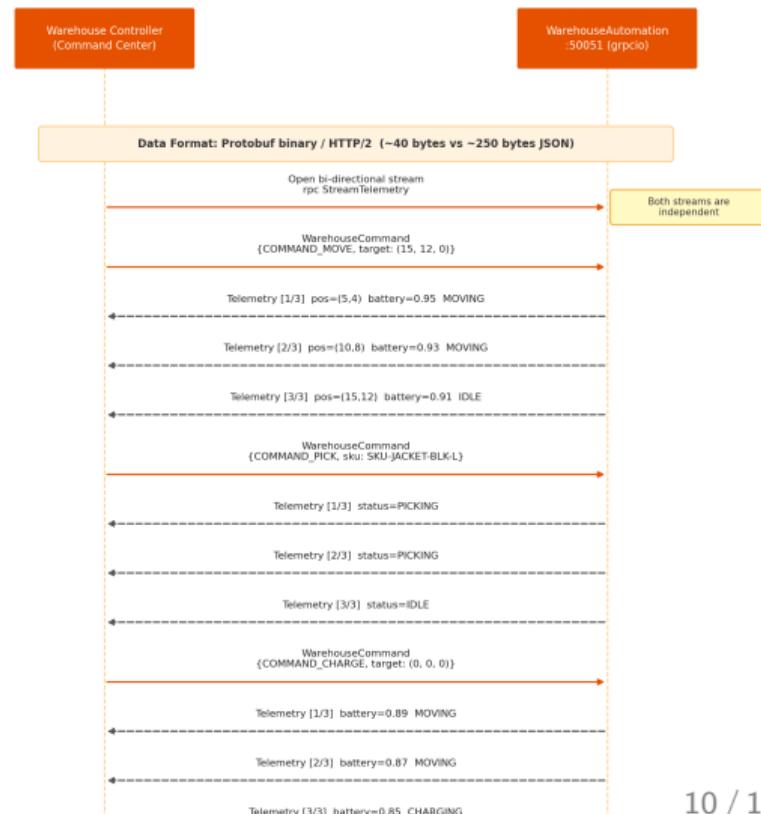
- **Efficience de sérialisation :**

L'encodage binaire Protocol Buffers divise par 6 (en moyenne) la taille du payload comparé au JSON, limitant la saturation du réseau sans-fil de l'entrepôt.

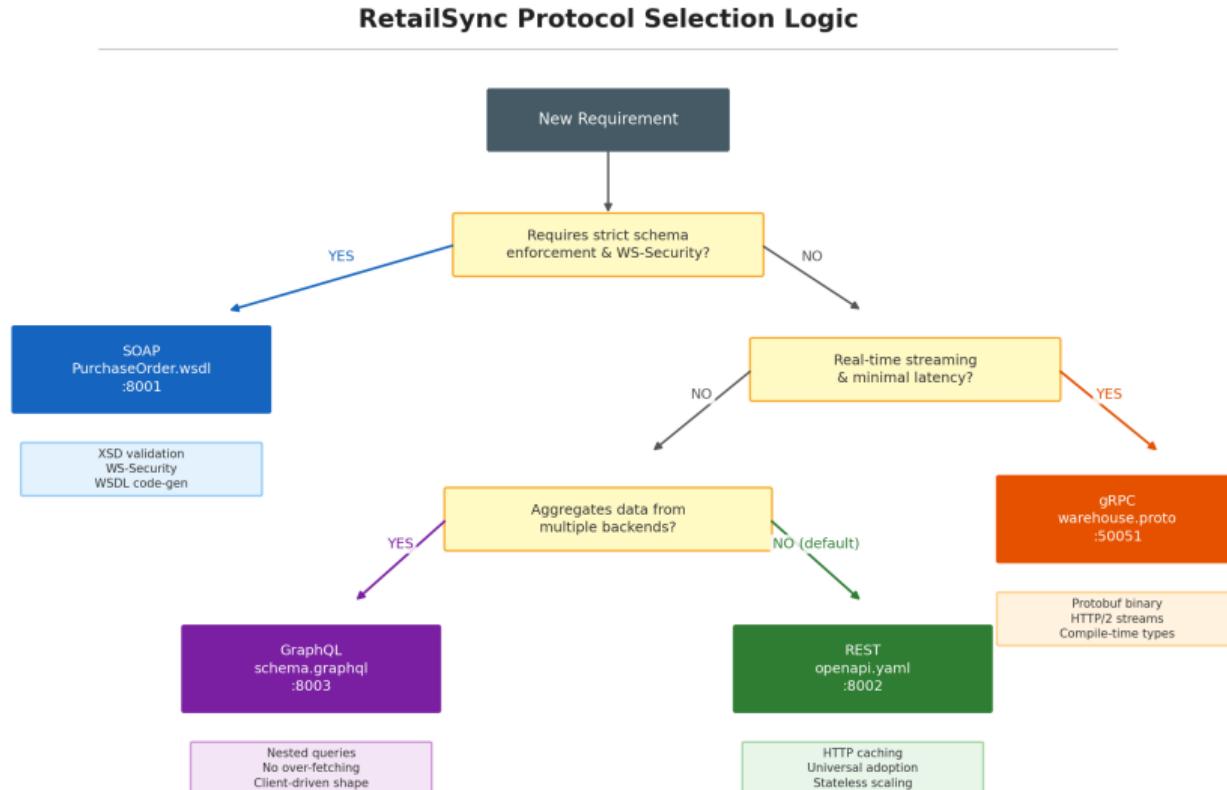
- **Streaming Bidirectionnel (HTTP/2)**

: La connexion persistante et la nature full-duplex annulent le coût d'ouverture répétée de connexions TCP ("Handshake

#### gRPC — Warehouse Robot Bi-directional Streaming



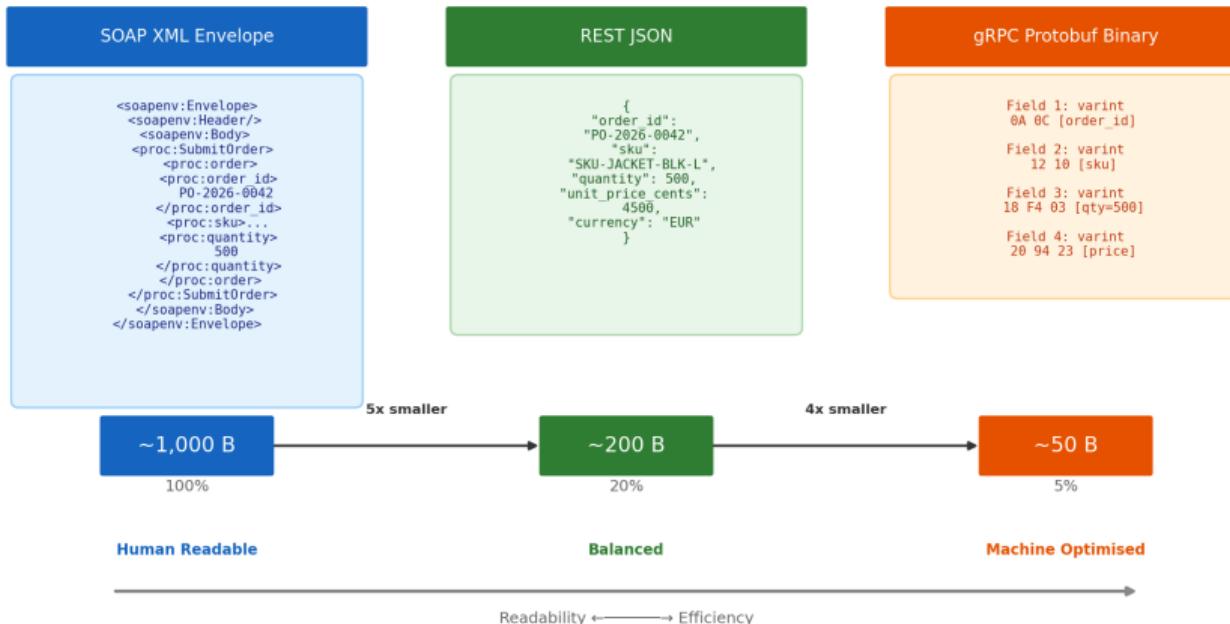
# Matrice décisionnelle : Orientations technologiques



# Analyse de surcharge d'encodage (Payload Overhead)

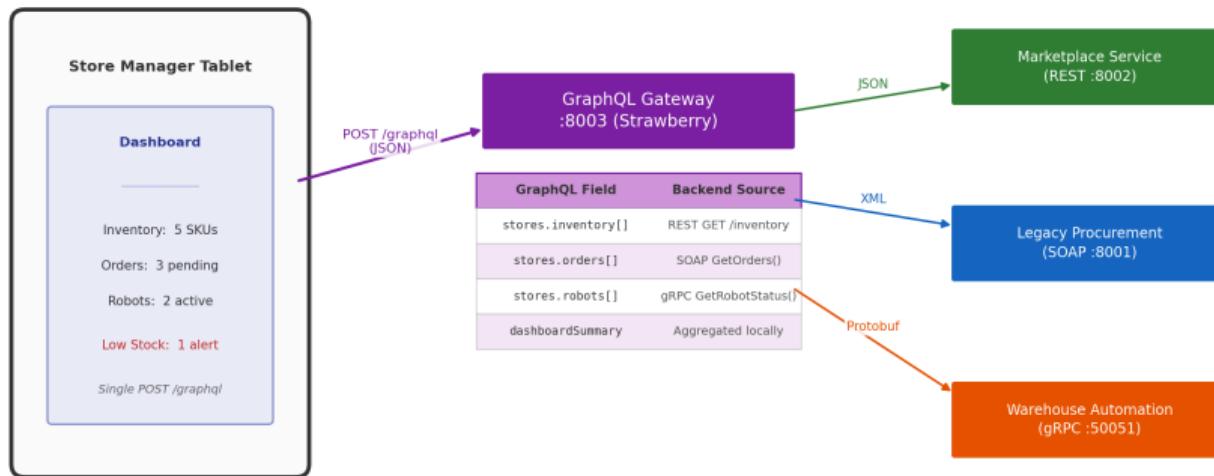
## Payload Overhead Comparison — Order Object

Same business data (SKU, quantity, price) encoded in three wire formats



# Motif d'Architecture : Le Backend-For-Frontend (BFF) Aggregator

## GraphQL: The Omnichannel Unified Interface



# Tableau de Synthèse Technique

Attributs	SOAP	REST	GraphQL	Blockchain
Couche Transport	HTTP/1.1	HTTP/1.1	HTTP/1.1	Blockchain
Sérialisation	XML	JSON	JSON	Blockchain
Typage / Contrat	Statique Fort (XSD)	Faible (OpenAPI optionnel)	Fort (Schéma GraphQL)	Statique
Modèle d'Exécution	Requête/Réponse	Requête/Réponse	Requête/Réponse (Graph)	Structuré
Mécanique de Cache	Aucune	Native (GET HTTP)	Limité (POST HTTP)	Structuré
Couplage Client	Important (Stubs WSDL)	Faible	Moyen	Important
Cas d'Usage Cible	Transactions B2B	Opérations CRUD publiques	Agrégation de données UI	Télémétrique

Il n'existe pas de standard API universel ("Silver Bullet"), uniquement des modélisations optimales pour des contraintes spécifiques.

# Point d'Amélioration : Implémentation d'une API Gateway

## Le périmètre actuel de l'étude (Scope Pédagogique) :

- Exposition directe de 4 serveurs sur **4 ports disparates** (8001, 8002, 8003, 50051).
- Permet l'étude isolée de chaque serveur, mais viole les bonnes pratiques de sécurité et de routage en environnement d'intégration.

## Évolution cible en production :

- Déploiement d'une **Passerelle d'API (API Gateway)**, comme Kong ou AWS Gateway.
- La passerelle agit comme reverse proxy de niveau **L7 (Couche Application)**, exposant un port d'entrée unifié (ex: 443 pour TLS).
  - /api/procurement → Routé vers le cluster SOAP interne.
  - /api/inventory → Routé vers le cluster REST interne.
- Résultat : Abstraction totale de l'architecture distribuée sous-jacente pour les consommateurs finaux, et centralisation des fonctions de sécurité (Authentification, Rate-Limiting).

# Synthèse Finale

- ① **Cohérence Fonctionnelle/Technique** : L'approche protocolaire doit toujours être subordonnée aux exigences fonctionnelles du domaine (ex. haute-fréquence logistique vs accessibilité partenaire).
- ② **Primauté de REST** : Demeure le standard *de facto* pour l'interopérabilité large, particulièrement pour des architectures orientées ressources (CRUD).
- ③ **Design Contract-Driven** : Établir des contrats (OpenAPI, WSDL, Protobuf) est un prérequis indispensable à la décentralisation des développements en architecture microservices.
- ④ **Synergie REST/GraphQL** : GraphQL ne déprécie pas REST. Il l'enrichit en se positionnant comme une couche d'agrégation d'expérience (*Backend-For-Frontend*) optimisée réseau.
- ⑤ **Avantage comparatif gRPC** : L'utilisation de protocoles multiplexés (HTTP/2) compacts (Protobuf) s'avère indispensable en contexte IOT et M2M pour prévenir la saturation réseau.

# Merci pour votre attention !

Avez-vous des questions ?

SOAP

:8001

REST

:8002

GraphQL

:8003

gRPC

:50051

[github.com/AyaMor/omnichain-retail-mesh](https://github.com/AyaMor/omnichain-retail-mesh)