

# Architectures logicielles : services, microservices et BPM

Étude Comparative d'Architectures API : SOAP, REST, GraphQL et gRPC

Filière VAP DSI

27/02/2026



INSTITUT  
POLYTECHNIQUE  
DE PARIS

# Plan de la présentation

- |   |   |              |
|---|---|--------------|
| ❶ | <b>Synthèse Comparative Principale</b> – Tableau des 4 APIs   | <i>2 min</i> |
| ❷ | <b>Contexte Fonctionnel</b> – Définition des besoins métier   | <i>2 min</i> |
| ❸ | <b>Choix Architectural</b> – Approche Microservices           | <i>3 min</i> |
| ❹ | <b>Analyse des Services et Protocoles</b> – Plongée technique | <i>7 min</i> |
| ❺ | <b>Conclusion et Améliorations</b> – API Gateway              | <i>2 min</i> |

SOAP

REST

GraphQL

gRPC

# Tableau comparatif entre les 4 API

Attributs	SOAP	REST	GraphQL	gRPC
Format de Données	XML	JSON	JSON	Binaire (Protobuf)
Norme Transport	HTTP/1.1 (Action unique)	HTTP/1.1 (Verbes)	HTTP/1.1 (POST)	HTTP/2 (Multiplexé)
Typage & Contrat	Statique Fort (WSDL/XSD)	Faible (OpenAPI en fallback)	Fort (Schéma .graphql)	Statique Fort (Fichier .proto)
Modèle d'Interaction	Requête/Réponse synchrone	Orienté Ressources (CRUD)	Requête de Graphe	RPC & Streaming Bidirectionnel
Performance / Poids	Très lourd (Surcharge XML)	Moyen (Format verbeux)	Optimisé (Tri à la source)	Ultra léger & Rapide
Mécanique de Cache	Inexistante	HTTP Natif (GET)	Complexe (Payload POST)	Inexistante
Flexibilité Sécurité	Haute (Normes WS-Security)	Standard (OAuth, TLS)	Standard (Nécessite contrôles)	Standard (TLS imposé par HTTP/2)
Couplage Client	Élevé (Génération de stubs)	Faible (Indépendance)	Moyen (Agilité de requêtage)	Élevé (Stubs natifs compilés)
Cas d'Usage Idéal	Transactions B2B / ERP	APIs Publiques & Web	Agrégation Vue Frontend	Microservices & Temps-réel

*Ce tableau fait l'état des lieux macroscopique. La suite détaillera les implémentations pratiques.*

# Modélisation du Système d'Information (Retail)

Modélisation du SI d'une entreprise (Retail), caractérisé par des flux hétérogènes entre **quatre domaines métiers distincts** :

## **SOAP** Domaine 1 : Approvisionnement

Création de bons de commande vers des fournisseurs industriels (ERP historiques).

## **REST** Domaine 2 : Réseau de Boutiques

Boutiques partenaires synchronisant leurs niveaux de stock avec le siège central.

## **GraphQL** Domaine 3 : Pilotage

Vue consolidée (Tableau de bord) des stocks, commandes et logistique pour la direction.

## **gRPC** Domaine 4 : Logistique Entrepôt

Télémétrie très haute fréquence issue de la flotte de robots d'automatisation.

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

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

- ❶ **Spécification du Contrat** – Définition formelle de l'interface avant toute implémentation.
- ❷ **Génération/Implémentation** – Le serveur est contraint par les stubs et schémas générés à partir du contrat.
- ❸ **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

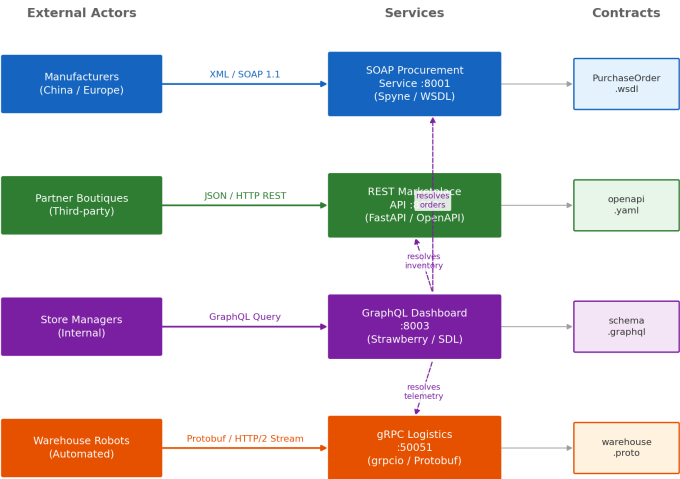
Face à l'hétérogénéité des besoins métier, une architecture monolithique a été rejetée au profit d'une **\*\*architecture orientée Microservices\*\***.

## 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 SubmitOrder pour centraliser les commandes.

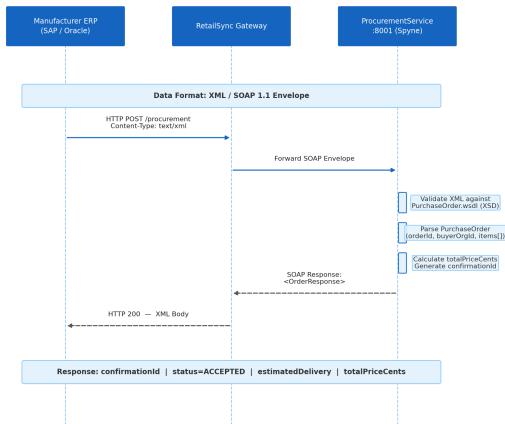
**Stack Technique** : Python (Framework Spyne, Serveur WSGI).

### Choix de SOAP :

- **Intégrité Stricte (XSD)** : Le WSDL force l'arborescence et les types ("Fail-fast").
- **Interopérabilité Legacy** : S'intègre aux ERP industriels sans couche de traduction.

- + Garantie structurelle
- + Sécurité (WS-Sec)
- Verbosité accrûe
- Bande passante

SOAP — B2B Procurement Flow





**Service** : Gère l'inventaire via HTTP (GET /inventory, PATCH /inventory/{sku}).

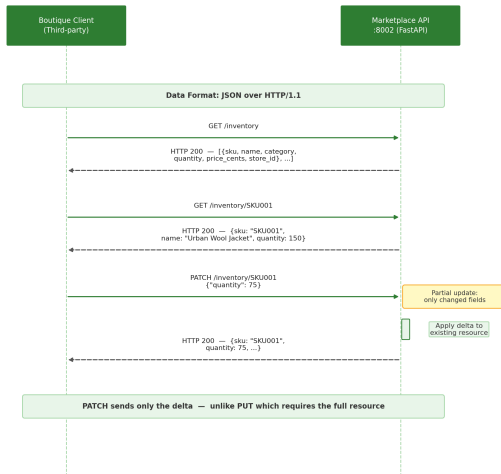
**Stack Technique** : Python (Framework FastAPI, Serveur Uvicorn ASGI).

## Motivations du choix de REST :

- **Accessibilité** : Standard du web fondé sur HTTP/JSON, facilitant l'intégration par les partenaires.
- **Notion de SKU** : Identifie un article unique en stock (ex: *Stock Keeping Unit*). Idéal pour cibler un PATCH précis.

- + Cache HTTP natif
- + Interface intuitive
- Typage dynamique
- Sur-récupération

### REST — Partner Marketplace Flow



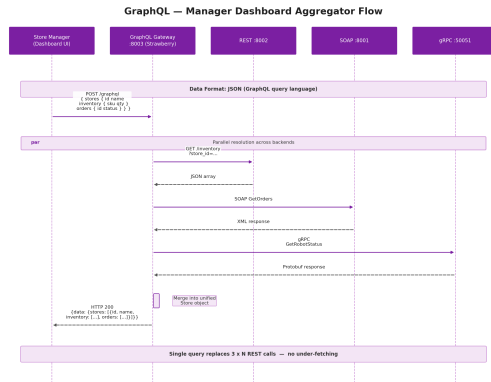
**Service** : Point d'entrée /graphql  
agrégant magasins, employés et commandes.

**Stack Technique** : Python (Bibliothèque Strawberry sur FastAPI).

### Choix de GraphQL :

- **Résout "N+1 Queries"** : Demande un arbre de données entier via une seule requête POST, évitant de multiplier les appels réseau HTTP.
- **"No Over-fetching"** : Le client spécifie uniquement les champs nécessaires pour l'interface UI.

- + Optimisation réseau
- + API Introspective
- Implémentation dure
- Cache inopérant



**Service** : Communication robotique  
*Machine-to-Machine* à très haute fréquence.

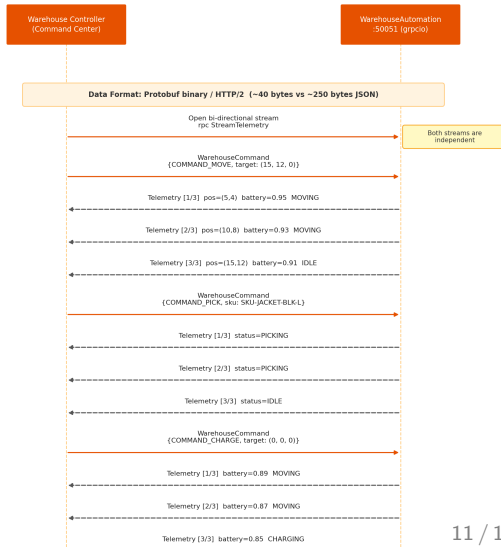
**Stack Technique** : Python (Lib. native  
grpcio & Protobuf).

### Choix de gRPC :

- **Efficiences binaire** : Protobuf divise par 6 la taille du payload par rapport au JSON. Vital pour réseaux sans-fil (entrepôt).
- **Streaming Bidirectionnel** : HTTP/2 maintient la connexion ouverte, annulant la latence TCP ("Handshake") pour du temps-réel pur.

- + Faible latence pure
- + Génération de Stubs
- Format opaque
- Browsers inadaptés

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



# Validation Technique : Scénarios de Tests Pratiques

**Objectif de la phase de test :** Valider l'implémentation et l'intégration des 4 protocoles en conditions réelles, en sollicitant les serveurs déployés localement (via Postman et terminaux).

**Périmètre de validation (Ce que nous allons exécuter et observer) :**

- **SOAP Approvisionnement (SOAP)** : Envoi et validation d'une commande B2B via une enveloppe XML stricte (WSDL).
- **REST Stocks Boutiques (REST)** : Récupération d'un inventaire produit via une méthode HTTP GET classique en JSON.
- **GraphQL Pilotage (GraphQL)** : Agrégation de données complexes (Magasins + Commandes) en une seule trame POST ciblée.
- **gRPC Logistique (gRPC)** : Établissement d'un flux de streaming bidirectionnel (HTTP/2) affichant la télémétrie robotique en temps réel.

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

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

- Exposition direct 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

- ❶ **Cohérence Fonctionnelle/Technique** : L'approche protocolaire doit toujours être subordonnée aux exigences fonctionnelles du domaine (ex. haute-fréquence logistique vs accessibilité partenaire).
- ❷ **Primauté de REST** : Demeure le standard *de facto* pour l'interopérabilité large, particulièrement pour des architectures orientées ressources (CRUD).
- ❸ **Design Contract-Driven** : Établir des contrats (OpenAPI, WSDL, Protobuf) est un prérequis indispensable à la décentralisation des développements en architecture microservices.
- ❹ **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.
- ❺ **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`