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

SOAP REST GraphQL gRPC

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 :

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.

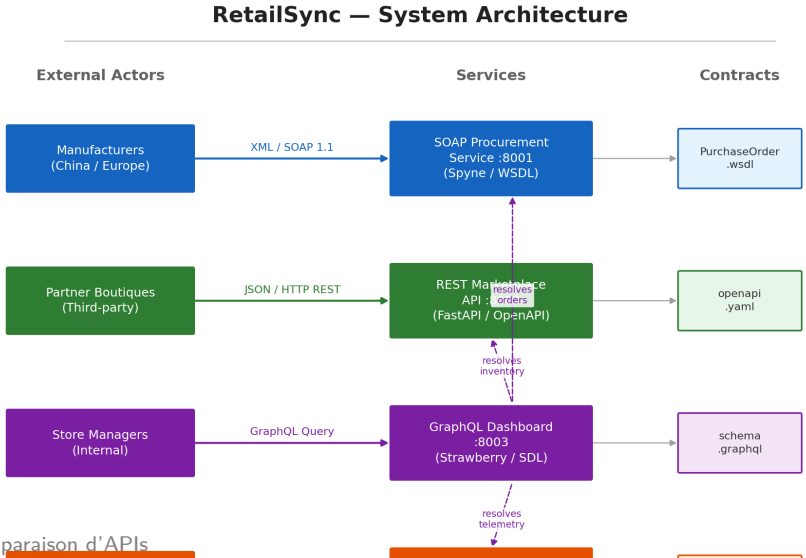
Justification de l'approche Microservices

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



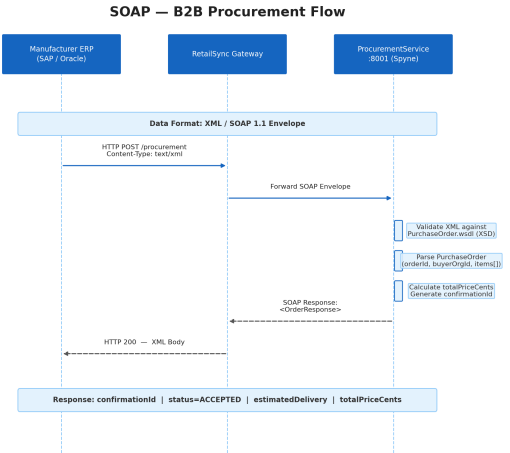
SOAP Service 1 : Approvisionnement (SOAP)

Service : Expose `SubmitOrder` pour centraliser les commandes.

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



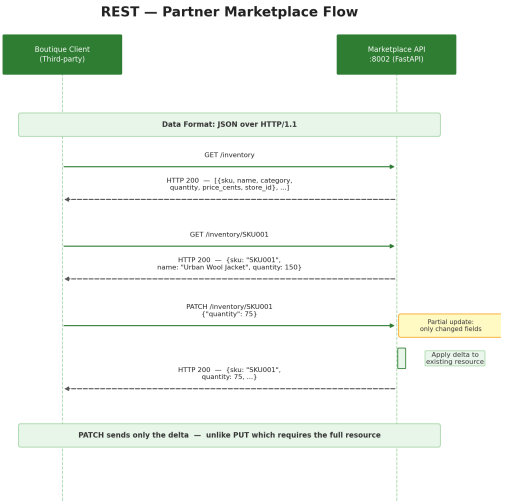
REST Service 2 : Stocks Routiques (REST)

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

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



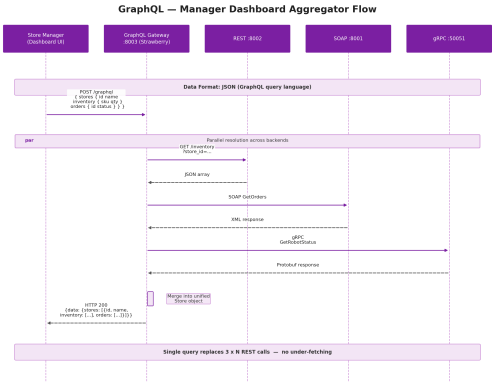
GraphQL Service 3 : Interface Reporting (GraphQL)

Service : Point d'entrée `/graphql` agrégeant magasins, employés et commandes.

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



gRPC Service 4 : Télémétrie Automatisée (gRPC)

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

Choix de gRPC :

- **Efficience 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

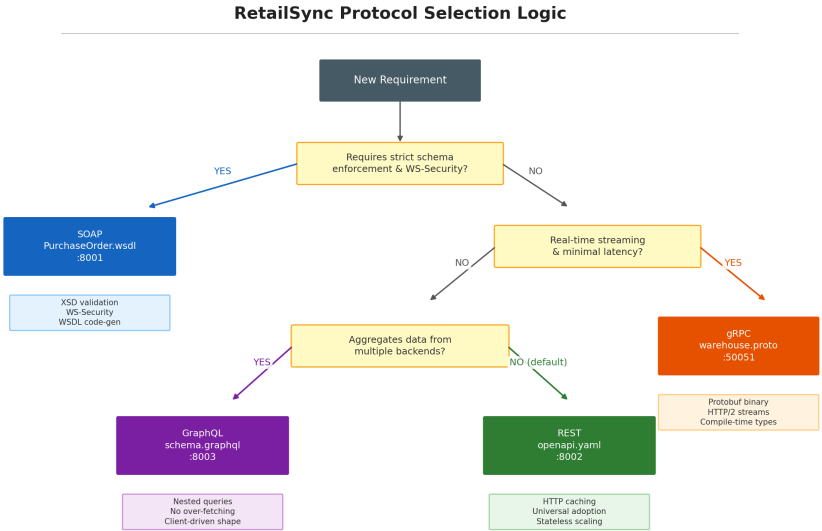
Validation Technique : Scénarios de Tests Pratiques

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).

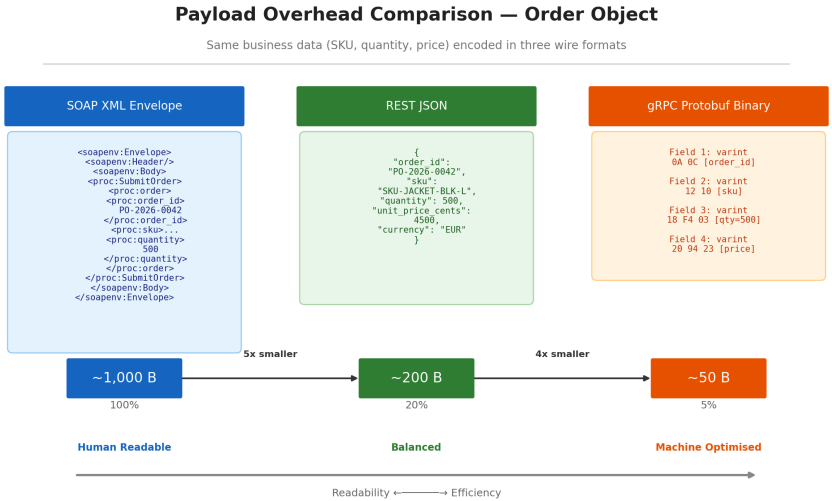
Périmètre de validation (Ce que nous allons exécuter et observer) :

- 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.

Matrice décisionnelle : Orientations technologiques



Analyse de surcharge d'encodage (Payload Overhead)



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

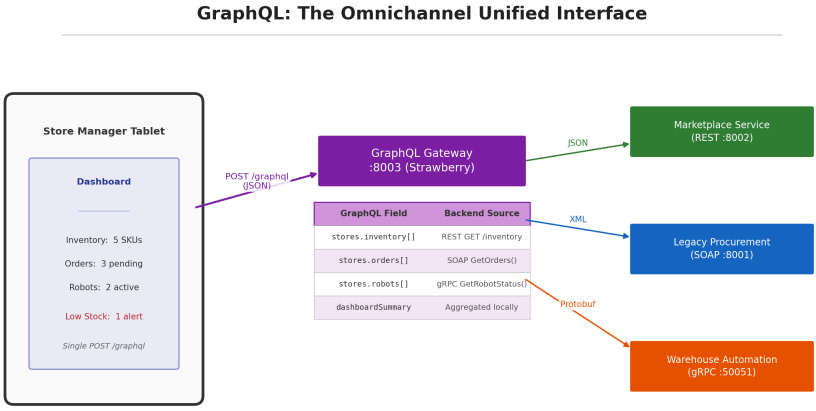


Tableau comparatif entre les 4 API

Attributs	SOAP	REST	GraphQL	gRPC
Couche Transport	HTTP/1.1	HTTP/1.1	HTTP/1.1	HTTP/2
Sérialisation	XML	JSON	JSON	Binaire (Protobuf)
Typage / Contrat	Statique Fort (XSD)	Faible (OpenAPI optionnel)	Fort (Schéma GraphQL)	Statique Fort (.proto)
Modèle d'Exécution	Requête/Réponse	Requête/Réponse	Requête/Réponse (Graph)	Streaming Natif RPC
Mécanique de Cache	Aucune	Native (GET HTTP)	Limité (POST HTTP)	Aucune
Couplage Client	Important (Stubs WSDL)	Faible	Moyen	Important (Stubs gRPC)
Cas d'Usage Cible	Transactions B2B	Opérations CRUD publiques	Agrégation de données UI	Télémétrie/Microservices internes

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 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

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