



Communications synchrones inter-applications

Objectifs

- Communications asynchrones
 - Fichiers
 - File de messages
 - Pub/Sub
- Communications synchrones
 - Accès direct à la BD
 - RPC (SOAP, JSON-RPC)
 - Par ressources (REST)
 - GraphQL
 - Temps interactif

Communications – Asynchrone – Fichiers 1/2

- Utilisations
 - Export quotidien de commandes vers l'ERP (CSV sur S3)
 - Import hebdomadaire de catalogue fournisseurs (XML/JSON)
 - Échanges EDI simples avec partenaires legacy
- Avantages
 - + Possibilité d'importations massives très efficaces (batch processing)
 - + Large éventail de moyens de transport (physique, réseau, cloud)
 - + Les formats textes et lisibles (CSV, JSON, XML, YAML, etc.)
 - + Certains formats sont standardisés (ASN.1, JSON, XML, YAML, etc.)
 - + Faible coût de mise en place (pas d'infrastructure complexe)
 - + Traitement quand les ressources sont disponibles (Tirer)
 - + Résilient par nature

Communications – Asynchrone – Fichiers 2/2

- Points de vigilance

- Tous les formats ne sont pas standardisés ou suffisamment spécifiés (CSV ambigu selon le séparateur, encodage, etc.)
- Problème d'encodage et de régionalisation pour les formats texte (UTF-8, ISO-8859, séparateur de champs, retour à la ligne, etc.)
- Maintenance difficile en cas de modification de schéma de données
- Forte dépendance au format partagé entre les applications
- Mise à l'échelle complexe
- Pas de traçabilité des accès
- Risques élevés pour la sécurité du transport (Injections, fichiers corrompus, transfert non chiffré)
- Non adapté aux intégrations en temps réel
- Limite l'intégration avec des applications tierces ou externes à l'entreprise

Communications – Asynchrone – File de messages 1/2

- Un producteur dépose un message dans une file, et un consommateur (ou plusieurs instances d'un même service) le traite (1 à 1)
 - Redimensionnement d'images, transcodage de vidéos, génération de PDF, envoi d'e-mails en arrière-plan
 - Pipeline d'enrichissement (enrichir puis stocker un document)
- Avantages
 - + Capable de gérer de gros volumes de messages en continu
 - + Rapide et efficace
 - + Répartition des traitements à plusieurs instances d'un consommateur
 - + Infrastructure relativement simple à mettre en place (Nombreuses solutions standards)
 - + Traitement quand les ressources sont disponibles (Tirer)
 - + Généralement sécurisé pour le producteur et le consommateur (Authentification/Autorisations/Chiffrement)
 - + Résilience pour les intégrations, notion de confirmation (ACK/NACK) de traitement

Communications – Asynchrone – File de messages 2/2

- Points de vigilance
 - Format des messages à négocier
 - Gestion des versions des formats de messages
 - Limité à un consommateur par message
 - Limite l'intégration avec des applications tierces ou externes à l'entreprise sans passerelle supplémentaire
 - Ajout de files de lettres mortes pour gérer les erreurs

Communications – Asynchrone – Pub/Sub 1/2

- Un producteur publie un message, et des abonnés le traitent
 - Propager le passage de commande (OrderCreated) vers facturation, stock, analytique en parallèle.
 - Synchroniser des caches et moteurs de recherche (ex : ProductUpdated → reindex)
- Avantages
 - + Capable de gérer de gros volumes de messages en continu
 - + Rapide et efficace pour distribuer des événements à plusieurs consommateurs en parallèle
 - + Répartition des traitements à plusieurs instances de plusieurs consommateurs
 - + Découplage fort : le producteur ne connaît pas le consommateur
 - + Infrastructure relativement simple à mettre en place (Nombreuses solutions standards)
 - + Traitement quand les ressources sont disponibles (Tirer)
 - + Généralement sécurisé pour le producteur et le consommateur (Authentification/Autorisations/Chiffrement)
 - + Résilience pour les intégrations, notion de confirmation (ACK/NACK) de traitement
 - + Garanties de livraison (at-least once 1+ et idempotent (Le plus fréquent), at-most once 0-1, exactly-once 1 (Rare))
 - + Certains brokers permettent la relecture des messages (replay) et peuvent servir de base pour un modèle Event Sourcing

Communications – Asynchrone – Pub/Sub 2/2

- Points de vigilance
 - Format des messages à négocier
 - Gestion des versions des formats de messages
 - Complexité de monitoring
 - Limite l'intégration avec des applications tierces ou externes à l'entreprise sans passerelle supplémentaire

Communications – Synchrone – BD 1/2

- Accès direct aux bases de données : une application accède directement aux données d'autres applications par des requêtes SQL
 - Batch interne d'analyse ponctuelle par l'équipe BI *dans* le même domaine
 - Migration de données unique et contrôlée
- Avantages
 - + Très efficace et performant
 - + Très facile à mettre en œuvre

Communications – Synchrone – BD 2/2

Points de vigilance

- Nécessite de connaître en détail les structures de BD des autres applications
- Maintenance difficile en cas de modification de schéma
- Couplage très fort entre les applications
- Mise à l'échelle compliquée
- Peu ou pas de traçabilité des accès
- Risques élevés pour la sécurité
- Contourne les validations et règles métier
- Limite l'intégration avec des applications tierces ou externes à l'entreprise
- Résilience : nécessité de politiques de réessaie côté application
- Risque de surcharge des services (Pousser)

Communications – Synchrone – RPC 1/2

- RPC (Remote Procedure Call) : API de type « appel de procédure distante », i.e. appel de fonction distante
 - Autoriser un paiement
 - Valider un panier
 - Obtenir un devis
- Avantages
 - + Validation des règles métier par la logique applicative et non directement par la BD
 - + Découplage vis-à-vis des sources de données
 - + Supporte les architectures orientées services (SOA) et les microservices (MSA)
 - + Interopérabilité grâce à des protocoles souvent standardisés et technologiquement agnostiques
 - + Réutilisabilité : les mêmes services peuvent être consommés par plusieurs clients (web, mobile, etc.)
 - + Possibilité de versionnage et d'évolution contrôlée
 - + Mise à l'échelle facile

Communications – Synchrone – RPC 2/2

- Points de vigilance

- Moins efficace (en plus des accès BD, sérialisation / désérialisation)
- Dépendance contractuelle forte entre clients et service
- Nécessite une infrastructure supplémentaire (serveurs d'application, monitoring, etc.)
- Évolution complexe si la gestion de versions n'est pas rigoureuse
- Résilience : nécessité de politiques de réessaie côté application
- Risque de surcharge des services (Pousser)

Communications – Synchrone – RPC

- Exemple :
 - Guid EnregistrerClient(Client client)
 - decimal ConvertirMonnaie(decimal montant, string de, string a)
 - List<Message> GenereLaSuiteDuTexte(List<Message> conversation)
- Quelques protocoles communs :
 - **SOAP**
 - **JSON-RPC** (Dans la suite des modules)
 - gRPC (Google RPC)

Communications – Synchrone – RPC – SOAP

- SOAP (Simple Object Access Protocol)
 - Service web décrit en XML
 - Beaucoup d'extensions (WS-*)
 - Documentation WSDL (Web Service Description Language) : génération automatique du code client
 - Utilise une enveloppe composée d'une en-tête et d'un corps
 - Agnostique d'un point de vue technologique



Communications – Synchrone – RPC – JSON-RPC

- JSON-RPC
- Utilise des enveloppes JSON standardisées pour effectuer les appels et les réponses
- Utilise souvent HTTP mais peut fonctionner sur d'autres protocoles

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": {"subtrahend": 23, "minuend": 42}, "id": 3}  
<-- {"jsonrpc": "2.0", "result": 19, "id": 3}
```

Communications – Synchrone – Par ressources

- REST (Representational State Transfer)
 - Portail public développeurs : /clients, /commandes, /factures
 - Intégration SaaS partenaires (standard de facto HTTP+JSON)
- Basé sur HTTP et expose des ressources via des URI
- Échanges généralement en JSON ou XML
- Même + et – que pour les RPCs et :
 - + Opérations sont normalisées (CRUD)
 - + Standard largement adopté et bien supporté dans l'écosystème web
 - + Basé sur des protocoles ouverts (HTTP, TLS, OAuth2, etc.)
 - Risque de renvoyer trop ou pas assez de données => possibilité de sur-requêtes
 - Simple à mettre en place techniquement mais difficile de définir les bonnes ressources
 - Pas de contrat strict, dépend de la rigueur des développeurs
 - Pas optimal pour les communications en temps interactif
 - Résilience : nécessité de politiques de réessaie côté application
 - Risque de surcharge des services (Pousser)

Communications – Synchrone – GraphQL

- Langage de requête pour API : choix des données attendues
 - App mobile avec écrans composites (profils + commandes + recommandations)
 - UI admin avec exploration ad-hoc et pagination affinée
- Même + et – que pour les RPCs et :
 - + Très flexible (Requêtes, paramètres, etc.)
 - + Réduit les risques de sur-requêtes
 - + Il suffit d'un seul point de terminaison
 - + Typage fort via un schéma GraphQL
 - + Adapté aux applications web et mobiles
 - Courbe d'apprentissage élevée
 - Complexe à sécuriser et à optimiser
 - Peut entraîner des surcharges des ressources si les requêtes ne sont pas limitées
 - Résilience : nécessité de politiques de réessaie côté application
 - Risque de surcharge des services (Pousser)

Operation	Response
<pre>query HeroNameAndFriends(\$episode: Episode) { hero(episode: \$episode) { name friends { name } } }</pre>	<pre>{ "data": { "hero": { "name": "R2-D2", "friends": [{ "name": "Luke Skywalker" }, { "name": "Han Solo" }, { "name": "Leia Organa" }] } } }</pre>
Variables	
<pre>{ "episode": "JEDI" }</pre>	

Communications – Synchrone – Temps interactif

- Échanges quasi immédiats (latence faible) entre les clients et le serveur, souvent bidirectionnel
 - Chat, tableau de bord, co-édition, jeux, enchères, etc.
 - Notifications de messages, de fin d'une opération, de tour dans une file d'attente, etc.
- Même + et – que pour les RPCs et :
 - + Expérience client
 - + Réduction des requêtes périodiques : le serveur peut communiquer avec les clients
 - + Envoi progressif des résultats
 - Complexité : état de la connexion, reconnexion, reprise, ordre des messages
 - Sécurité : authentification en continu, contrôle de fin de communication
 - Difficile à tester
- Transports : WebSocket (Bidirectionnel), Server-Sent Events (SSE, unidirectionnel), Long-polling, etc.
- Cadriciels : SignalR (.NET), Socket.IO (Node.JS)

Résumé

Approche	Type com.	Couplage	Performance	Scalabilité	Sécurité	Résilience	Cas d'usage typiques	Limites / risques
Fichiers (batch)	Asynchrone	Faible (format partagé)	Élevée en lot	Complexe (fenêtres, orchestration)	Variable (transport/stockage)	Naturellement asynchrone	Import/export périodiques, intégrations legacy, EDI simple	Pas temps réel, schémas fragiles, traçabilité faible
File de messages (queue)	Asynchrone	Faible	Élevée en lot	Bonne (workers parallèles)	Bonne	Naturellement asynchrone ACK/NACK	Traitements 1→1, tâches lourdes différées	Un seul consommateur par message, versioning formats
Pub/Sub	Asynchrone	Très faible	Élevée en lot	Excellente (n consommateurs)	Bonne	Naturellement asynchrone ACK/NACK, replays (selon broker)	Événements de domaine, broadcast de changements	Monitoring/observabilité plus complexes
Accès direct BD	Synchrone	Très fort	Très élevée	Difficile	Risquée (contourne règles métier)	À la charge du client	Lecture analytique interne, migration ponctuelle	Surcharges, sécurité/traçabilité faibles, couplage dur
RPC (SOAP/gRPC/JSON-RPC)	Synchrone	Contractuel	Bonne	Bonne	Bonne	À implémenter côté client	Microservices orientés opérations, logique métier	Contrats stricts, versionnage à gérer
REST (par ressources)	Synchrone	Contractuel léger	Bonne	Bonne	Bonne	À implémenter côté client	APIs web & mobiles, CRUD, intégration large	Sur/sous-requêtes, design des ressources délicat
GraphQL	Synchrone	Schéma fort	Bonne	Bonne	Bonne	À implémenter côté client	UIs complexes, mobiles, agrégation de sources	Requêtes coûteuses si non bornées, gouvernance requise
Temps interactif (push/bidirectionnel)	Synchrone	Contractuel (canaux/hubs)	Très bonne	Bonne mais nécessite un backplane (bus de diffusion) entre les instances	Bonne	Reconnexion, heartbeats, reprise d'offset	Chat, co-édition, dashboards live, notifications, streaming IA	Complexité (état, ordre, backpressure), sticky sessions , tests/monitoring plus difficiles