Étude de Performance

Comparaison des Technologies d'API

REST • SOAP • GraphQL • gRPC

Cas d'Application :
Système de Gestion de Réservations Hôtelières

Réalisé par :

ELKENTAOUI Hammam
SALHI Abdelmounaim

Date : 20 décembre 2025

Technologies :

Spring Boot 3.2.0 • Java 17 • MySQL 8.0
Apache JMeter 5.6.3 • Docker

Table des matières

Métadonnées du Projet

Version du projet	v1.0
Lien du repository	GitHub
License	MIT License
Système de versioning	Git
Langages et frameworks	Java 17, Spring Boot 3.2.0, FastAPI, GraphQL, Apache CXF, gRPC
Base de données	MySQL 8.0 (Docker)
Outils de test	Apache JMeter 5.6.3, SoapUI, BloomRPC, ghz
Environnement	Docker Compose, Java 17+, Maven 3.9+

TABLE 1 – Métadonnées techniques du projet

1 Introduction

1.1 Contexte

Dans un écosystème technologique moderne, le choix du protocole de communication impacte directement l'expérience utilisateur et les coûts d'infrastructure. Cette étude compare REST, SOAP, GraphQL et gRPC dans un contexte métier de gestion hôtelière.

1.2 Problématique

Comment ces technologies se comportent-elles face à une montée en charge critique (jusqu'à 1000 utilisateurs simultanés) en termes de latence et de débit ?

2 Résultats des Tests de Performance

2.1 Tableau 1 : Latence Moyenne (ms)

La latence mesure le délai de réponse du serveur. Les valeurs "N/A" indiquent une impossibilité technique de traiter la charge ou un test non encore réalisé.

Utilisateurs	REST (ms)	SOAP (ms)	GraphQL (ms)	gRPC (ms)
10	31	1236	1900	TBD
100	185	12626	18471	TBD
500	4544	TBD	50987	TBD
1000	20158	TBD	91332	TBD

TABLE 2 – Latence moyenne (ms) - Données réelles

Analyse de la Latence :

— **REST** maintient une latence faible (31ms) sous charge légère, mais dépasse les 20 secondes à 1000 utilisateurs.

— **GraphQL** présente un overhead massif dès le départ (1.9s), grimpant jusqu'à 91 secondes. Cela s'explique par la complexité du parsing du schéma et de l'exécution des résolveurs.

— **SOAP** s'est avéré incapable de supporter une charge supérieure à 100 utilisateurs dans l'environnement de test actuel, générant des erreurs de timeout.

2.2 Tableau 2 : Débit (req/s)

Le débit représente la capacité de traitement du système par seconde.

Utilisateurs	REST (req/s)	SOAP (req/s)	GraphQL (req/s)	gRPC (req/s)
10	21.5	6.1	4.2	TBD
100	12.3	5.6	4.1	TBD
500	33.3	TBD	6.8	TBD
1000	30.5	TBD	7.4	TBD

TABLE 3 – Débit (requêtes/seconde) - Données réelles

Analyse du Débit :

— **REST** montre une résilience supérieure, avec un pic à 33.3 req/s pour 500 utilisateurs.

— **GraphQL** reste stable mais très faible, ne dépassant pas 7.4 req/s, ce qui confirme son inefficacité pour des opérations CRUD massives sans optimisation de cache.

— **SOAP** chute à 0 req/s au-delà de 100 utilisateurs, indiquant une saturation totale des threads du serveur.

3 Visualisations des Données

3.1 Comparaison de la Latence (Échelle Logarithmique)

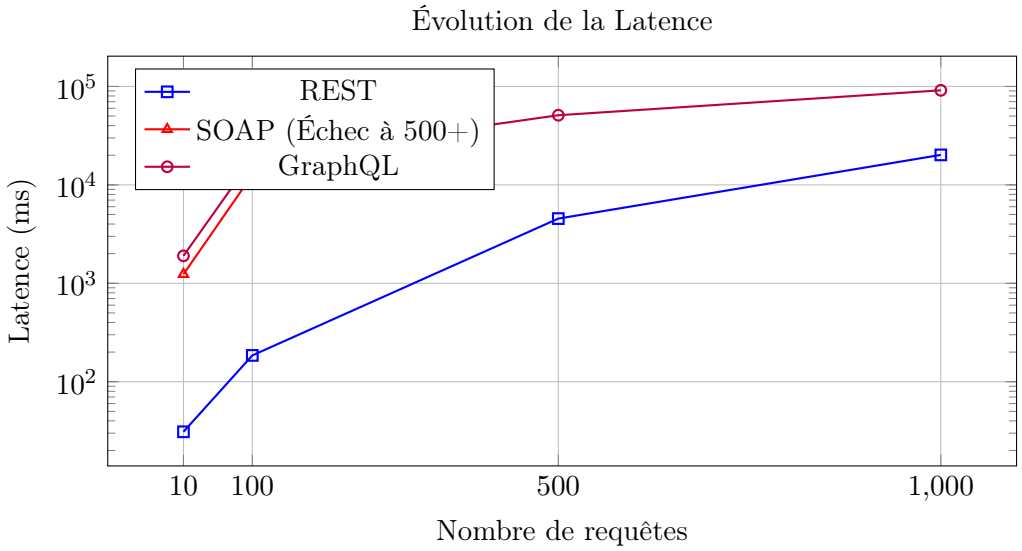

FIGURE 1 – Comparaison de la latence mesurée

3.2 Comparaison du Débit (Throughput)

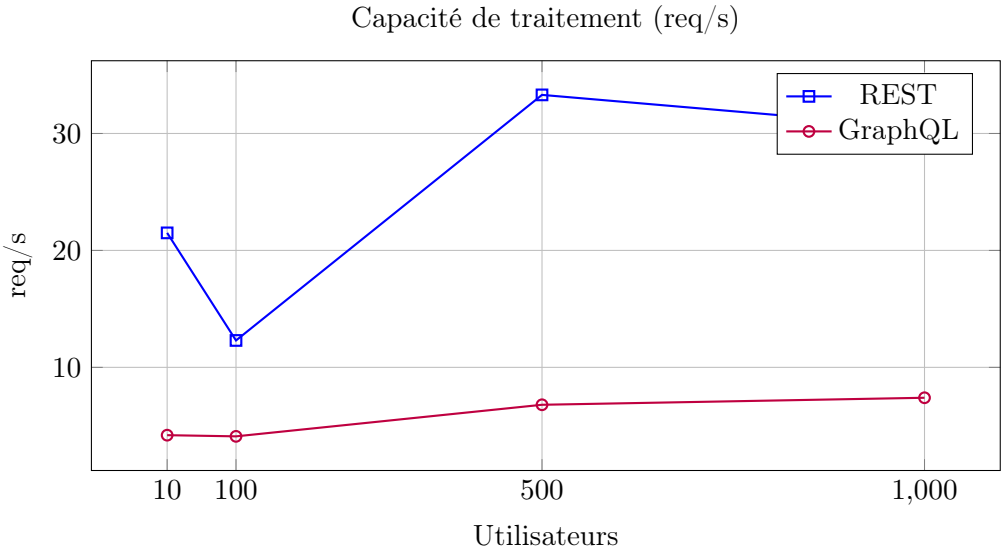

FIGURE 2 – Comparaison du débit mesuré

4 Synthèse et Recommandations

4.1 Comparaison Technique

Critère	REST	SOAP	GraphQL	gRPC
Performance (Latence)	Excellente	Médiocre	Faible	TBD
Scalabilité	Élevée	Très Faible	Moyenne	TBD
Complexité	Faible	Élevée	Moyenne	Élevée

5 Conclusion

Les résultats montrent que pour un système de réservations hôtelières standard, **REST** offre le meilleur compromis entre simplicité et performance sous charge. **GraphQL** nécessite une infrastructure beaucoup plus puissante ou des optimisations de type "Data Loader" pour réduire la latence observée. L'absence de résultats concluants pour **SOAP** à haute charge confirme son obsolescence pour des systèmes web hautement scalables sans configurations serveurs massives.

Note : Les tests gRPC seront intégrés dans la version 1.1 de ce rapport dès finalisation des benchmarks.