

# Analyse de la Scalabilité et des Performances des APIs Modernes

Étude de Cas : Gestion d'une Plateforme de Réservation d'Hôtel

Ahmed ELHAMRI

## Technologies comparées

REST • SOAP • GraphQL • gRPC

## Outils de test

- JMeter
- Prometheus

## Résumé

Cette étude présente une comparaison expérimentale de quatre technologies d'API modernes : **REST**, **SOAP**, **GraphQL** et **gRPC**, appliquées à un scénario réel de gestion de réservations hôtelières.

Les objectifs principaux sont :

- l'évaluation des performances (latence, débit),
- l'analyse de la scalabilité sous charge,
- la mesure de la consommation des ressources,
- l'étude de la simplicité d'implémentation,
- l'évaluation de la sécurité.

Les tests ont été réalisés à l'aide de **JMeter** pour la génération de charge et de **Prometheus** pour la collecte des métriques système.

## Contexte

Une plateforme de réservation d'hôtels doit gérer un grand nombre de requêtes concurrentes tout en garantissant :

- des temps de réponse faibles,
- une haute disponibilité,
- une sécurité renforcée.

Les opérations testées sont :

- Créer une réservation
- Consulter une réservation
- Modifier une réservation
- Supprimer une réservation

## Objectifs

- Comparer REST, SOAP, GraphQL et gRPC dans un scénario réaliste
- Mesurer la latence et le débit
- Analyser l'utilisation CPU et mémoire
- Évaluer la simplicité d'implémentation et la sécurité
- Fournir une synthèse pour le choix technologique

## Méthodologie de Test

### Outils

- **JMeter** : génération de charge et mesure des performances
- **Prometheus** : collecte CPU et mémoire

### Charges testées

- 10, 100, 500 et 1000 requêtes simultanées
- Opérations CRUD complètes

## Performances : Temps de Réponse (Latence)

Utilisateurs	Opération	REST	SOAP	GraphQL	gRPC
10	Créer	41	55	15	311
	Consulter	32	25	14	25
	Modifier	11	11	11	27
	Supprimer	8	7	9	12
100	Créer	26	34	175	105
	Consulter	14	19	274	12
	Modifier	6	7	165	9
	Supprimer	5	5	104	6
500	Créer	595	58	28	29
	Consulter	588	31	34	10
	Modifier	521	16	17	9
	Supprimer	418	11	11	5
1000	Créer	805	133	76	38
	Consulter	775	76	79	9
	Modifier	677	51	49	6
	Supprimer	573	35	39	5

## Performances : Débit (Throughput)

Requêtes	REST	SOAP	GraphQL	gRPC
10	1.3	4.4	4.4	1.8
100	1.9	40	40	1.6
500	5.7	26	200	0.4
1000	9.8	398	400	0.43

## Consommation des Ressources

### CPU

Requêtes	REST	SOAP	GraphQL	gRPC
10	9.7	0.04	15.48	12.98
100	9.6	12	13.51	11.72
500	13.25	11	10.22	11.87
1000	14.7	19.56	12.12	9.48

### Mémoire

Requêtes	REST	SOAP	GraphQL	gRPC
10	155.8	187	224	257
100	243	240	386	223
500	258	186	220	260
1000	303	356	310	239

## Simplicité d'Implémentation

Critère	REST	SOAP	GraphQL	gRPC
Temps d'implémentation	1–2 h	2–3 h	2–4 h	3–5 h
Lignes de code	Faible	Élevé	Moyen	Moyen
Outils disponibles	Très élevée	Élevée	Élevée	Moyenne
Courbe d'apprentissage	1–2 j	3–5 j	2–3 j	3–4 j

## Sécurité

Critère	REST	SOAP	GraphQL	gRPC
TLS/SSL	Oui	Oui	Oui	Oui
Authentification	OAuth2, JWT	WS-Security	OAuth2, JWT	JWT, mTLS
Résistance aux attaques	Moyenne	Élevée	Moyenne	Élevée

## Résumé Global

Critère	REST	SOAP	GraphQL	gRPC
Latence moyenne	Élevée	Moyenne	Faible	Très faible
Débit moyen	Faible	Élevé	Très élevé	Faible
CPU moyen	Moyen	Moyen	Moyen	Faible
Mémoire moyenne	Moyenne	Moyenne	Élevée	Moyenne
Sécurité	Bonne	Très élevée	Bonne	Très élevée
Simplicité	Très élevée	Faible	Moyenne	Moyenne

## Conclusion

Cette étude montre que :

- **GraphQL** offre le meilleur débit sous forte charge,
- **gRPC** présente une latence très faible et une bonne efficacité CPU,
- **SOAP** est robuste et très sécurisé mais complexe,
- **REST** reste le plus simple à implémenter mais moins scalable à grande échelle.

Le choix technologique dépend fortement du contexte applicatif, des exigences de performance et des contraintes de sécurité.