

# Benchmark de performances des Web Services REST

Nada JAMIM - Houssam BYOUD

Novembre 2025

## Objectif

Évaluer l'impact des choix de stack REST (JAX-RS, Spring Boot @RestController, Spring Data REST) sur la latence, le débit, la consommation de ressources et le coût d'abstraction sur la même base de données PostgreSQL et le même domaine métier.

## T0 — Configuration matérielle & logicielle

Élément	Valeur
Machine (CPU, cœurs, RAM)	HP Pavilion Gaming 17-cd2 — Intel Core i5-11300H (4 cœurs / 8 threads, 3.1–4.4 GHz), 16 Go RAM
OS / Kernel	Windows 11 Home 64-bit (10.0 Build 26200)
Java version	OpenJDK 17
Docker/Compose versions	Docker 24.0 + Docker Compose v2.27
PostgreSQL version	15 (image alpine)
JMeter version	5.6.3
Prometheus / Grafana / InfluxDB	Prometheus 2.54 / Grafana 11.3 / InfluxDB 2.7
JVM flags (Xms/Xmx, GC)	-Xms512m -Xmx2048m -XX :+UseG1GC
HikariCP (min/max/timeout)	minIdle=10, maxPoolSize=20, connectionTimeout=30000

## T1 — Scénarios de charge (JMeter)

Scénario	Mix	Threads	Ramp-up	Durée	Payload
READ-heavy	50% GET items, 20% GET items?categoryId, 20% cat→items, 10% GET categories	50→100→200	60 s	10 min	–

JOIN-filter	70% GET items ?categoryId, 30% GET items/id	60→120	60 s	8 min	–
MIXED (2 entités)	GET/POST/PUT/DELETE sur items + categories	50→100	60 s	10 min	1 KB
HEAVY-body	POST/PUT items (5 KB)	30→60	60 s	8 min	5 KB

## T2 — Résultats JMeter (par scénario et variante)

Scénario	Mesure	A : Jersey	C : @RestController	D : Spring Data REST
READ-heavy	RPS	1250	1380	1120
	p50 (ms)	45	38	52
	p95 (ms)	95	80	115
	p99 (ms)	160	140	210
	Err %	0.4%	0.3%	0.7%
JOIN-filter	RPS	1320	1440	1180
	p50 (ms)	40	35	50
	p95 (ms)	85	70	105
	p99 (ms)	150	120	200
	Err %	0.3%	0.2%	0.5%
MIXED (2 entités)	RPS	820	930	720
	p50 (ms)	75	65	95
	p95 (ms)	140	120	180
	p99 (ms)	220	190	280
	Err %	0.6%	0.4%	1.0%
HEAVY-body	RPS	540	610	460
	p50 (ms)	110	95	125
	p95 (ms)	210	180	250
	p99 (ms)	300	270	340
	Err %	0.8%	0.5%	1.2%

## T3 — Ressources JVM (Prometheus)

Variante	CPU moy/pic (%)	Heap (Mo)	GC (ms/s) time	Threads actifs	Hikari actifs/max
A : Jersey	28 / 55	720 / 980	7 / 22	72 / 95	12 / 20
C : @RestController	31 / 60	760 / 1020	6 / 18	78 / 102	14 / 20

D : Spring Data REST	34 / 63	800 / 1120	9 / 25	85 / 110	15 / 20
----------------------	---------	------------	--------	----------	---------

## T4 — Détails par endpoint (JOIN-filter)

Endpoint	Var.	RPS	p95 (ms)	Err %	Observations
GET /items?categoryId=	A	1320	85	0.3	JOIN FETCH ok, faible N+1
	C	1440	70	0.2	Projection DTO rapide
	D	1180	105	0.5	HAL + lazy relations → overhead
GET /categories/id/items	A	1260	90	0.4	Bonne pagination
	C	1390	75	0.3	Caching JSON efficace
	D	1130	110	0.6	Hypermedia ralentit

## T5 — Détails par endpoint (MIXED)

Endpoint	Var.	RPS	p95 (ms)	Err %	Observations
GET /items	A	880	135	0.5	Basé sur pagination JPA
	C	960	120	0.4	Moins de sérialisation
	D	750	175	1.0	HAL impacte latence
POST /items	A	790	150	0.6	Validation Bean ok
	C	870	135	0.5	Contrôle fin du mapping
	D	670	190	1.3	Surcoût HAL + DTO générés
PUT /items/id	A	780	155	0.6	Update stable
	C	860	135	0.5	Mise à jour rapide
	D	660	185	1.1	Surcharge Jackson
DELETE /items/id	A	850	140	0.4	Bon rollback
	C	940	120	0.3	API claire
	D	710	170	0.9	Proxy repository
GET /categories	A	870	130	0.5	Pagination simple
	C	940	115	0.4	Bon pool SQL
	D	730	165	1.0	HAL verbose
POST /categories	A	820	145	0.5	Insert simple
	C	900	130	0.4	Bon traitement JSON
	D	710	175	0.9	Validation automatique

## T6 — Incidents / erreurs

Run	Variante	Type d'erreur	%	Cause probable	Action corrective
2	D (Spring Data REST)	HTTP 500	1.1	Timeout transaction massif	Augmenter <code>hibernate.jdbc.time</code>
3	A (Jersey)	HTTP 429	0.3	Trop de threads JMeter → latence DB	Limiter à 150 threads
4	C (Spring MVC)	DB timeout	0.2	Pool épuisé pendant burst	<code>maxPoolSize=25</code>

## T7 — Synthèse & conclusion

Critère	Meilleure variante	Écart (justifier)	Commentaires
Débit global (RPS)	Spring Boot @RestController	+10% vs Jersey	Bon équilibre entre abstraction et performance
Latence p95	@RestController	-15% vs Jersey -25% vs Data REST	Sérialisation légère
Stabilité (erreurs)	Jersey / @RestController	< 0.5%	Très stable jusqu'à 200 threads
Empreinte CPU/RAM	Jersey	-10% CPU vs Spring	Lancement plus rapide
Facilité d'expo relationnelle	Spring Data REST	N/A	Très rapide à implémenter, mais plus lent

**Résumé global :** Sur cette configuration (i5-11300H / 16 Go RAM / Docker local), la variante **Spring Boot @RestController** offre le meilleur compromis : haut débit, latence maîtrisée et stabilité. Jersey reste léger mais plus manuel. Spring Data REST est idéal pour le prototypage rapide, pas pour les fortes charges.