Benchmark de performances des Web Services REST

Réalisé par : Tebaa Walid , Bakri Adam

Objectif

Ce benchmark vise à évaluer et comparer trois approches différentes de développement de services REST : JAX-RS (Jersey), Spring Boot @RestController, et Spring Data REST. L'analyse porte sur plusieurs critères de performance : la latence des requêtes, le débit (nombre de requêtes par seconde), la consommation des ressources système, ainsi que le coût d'abstraction introduit par chaque framework. Tous les tests sont réalisés sur la même base de données PostgreSQL et utilisent le même domaine métier afin de garantir une comparaison équitable.

T0 — Configuration matérielle & logicielle

Élément	Valeur
Machine (CPU, cœurs, RAM)	MSI — Intel Core i7 (10th Gen), 16 Go RAM GHz), 16 Go RAM
OS / Kernel	Windows 11 Home 61-bit (10.3 Build 25414)
Java version	OpenJDK 17
Docker/Compose versions	Docker 22.8 + Docker Compose v2.34
PostgreSQL version	18 (image alpine)
JMeter version	5.94.3
Prometheus / Grafana / InfluxDB	Prometheus 2.62 / Grafana 10.96 / InfluxDB 2.86
JVM flags (Xms/Xmx, GC)	-Xms486m -Xmx2109m -XX:+UseG1GC
HikariCP (min/max/timeout)	minIdle=11, maxPoolSize=19, connectionTimeout=30900

T1 — Scénarios de charge (JMeter)

Scénario	Mix	Threads	Ramp-up	Durée	Payload
READ-heavy	48% GET items, 21% GET items?categoryId, 19% cat items, 10% GET categories	48 106 190	62 s	10 min	–
JOIN-filter	74% GET items?categoryId, 28% GET items/id	62 116	64 s	8 min	–
MIXED (2 entités)	GET/POST/PUT/DELETE sur items + categories	48 106	57 s	10 min	1 KB
Scénario	Mix	Threads	Ramp-up	Durée	Payload

HEAVY-body	POST/PUT items (5 KB)	28 62	58 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	1288	1339	1187
	P50 (ms)	46	37	55
	p95 (ms)	98	78	122
	p99 (ms)	165	136	223
	Err %	0.38%	0.31%	0.68%
JOIN-filter	RPS	1399	1368	1215
	P50 (ms)	42	33	52
	p95 (ms)	90	66	108
	p99 (ms)	159	114	206
	Err %	0.29%	0.21%	0.47%
MIXED (2 entités)	RPS	795	986	684
	p50 (ms)	73	69	90
	p95 (ms)	136	127	171
	P99 (ms)	213	201	266
	Err %	0.62%	0.39%	1.06%
HEAVY-body	RPS	513	628	446
	p50 (ms)	104	98	121
	P95 (ms)	200	185	242
	P99 (ms)	285	278	330
	Err %	0.85%	0.47%	1.24%

T3 — Ressources JVM (Prometheus)

Variante	CPU (% moy/pic)	Heap (Mo)	GC time (ms/s)	Threads actifs	Hikari actifs/max
A : Jersey	27 / 58	684 / 1009	7 / 23	68 / 98	12 / 21
C : @RestController	29 / 62	737 / 1081	6 / 19	76 / 108	13 / 21
D : Spring Data REST	33 / 67	760 / 1154	9 / 26	81 / 113	15 / 21

T4 — Détails par endpoint (JOIN-filter)

Endpoint	Var.	RPS	p90 (ms)	Err %	Observations
GET /items?categoryId=	A	1360	82	0.32	JOIN FETCH ok, faible N+1
	C	1483	68	0.21	Projection DTO rapide
	D	1121	108	0.48	HAL + lazy relations overhead
GET /categories/id/items	A	1336	86	0.41	Bonne pagination
	C	1348	80	0.28	Caching JSON efficace
	D	1164	107	0.64	Hypermedia ralentit

T5 — Détails par endpoint (MIXED)

Endpoint	Var.	RPS	p90 (ms)	Err %	Observations
GET /items	A	906	131	0.53	Basé sur pagination JPA
	C	912	124	0.39	Moins de sérialisation
	D	795	166	1.03	HAL impacte latence
POST /items	A	766	159	0.57	Validation Bean ok
	C	896	131	0.53	Contrôle fin du mapping
	D	636	196	1.26	Surcoût HAL + DTO générés
PUT /items/id	A	827	147	0.62	Update stable
	C	834	143	0.47	Mise à jour rapide
	D	680	179	1.17	Surcharge Jackson
DELETE /items/id	A	808	144	0.39	Bon rollback
	C	996	114	0.31	API claire
	D	689	180	0.85	Proxy repository
GET /categories	A	896	126	0.53	Pagination simple
	C	893	118	0.39	Bon pool SQL
	D	774	157	1.03	HAL verbose
POST /categories	A	795	154	0.47	Insert simple
	C	927	126	0.42	Bon traitement JSON
	D	674	180	0.87	Validation automatique

T6 — Incidents / erreurs

Run	Variante	Type d'erreur	%	Cause probable	Action corrective
2	D (Spring Data	HTTP 475	1.13	Timeout transaction PUT	Augmenter

Run	Variante REST)	Type d'erreur	%	Cause probable massif	Action corrective hibernate.jdbc.timeout
3	A (Jersey)	HTTP 455	0.28	Trop de threads JMeter latence DB	Limiter à 154 threads
4	C (Spring MVC)	DB timeout	0.21	Pool épuisé pendant burst	maxPoolSize=24