# Benchmark de performances des Web Services REST

Oussama Tahiri
Younes Fetouaki

Le 4 Novembre 2025

## T0 - Configuration matérielle & logicielle

| Élément | Valeur |
|---|---|
| Machine (CPU, cœurs, RAM) | GIGABYTE AORUS 16X ASG — Intel Core i7-14650HX (24 cœurs logiques), 16 GB DDR5 RAM, NVMe 1TB |
| OS / Kernel | Windows 11 Home 64-bit (Build 26200) |
| Java version | OpenJDK 17.0.2 |
| Docker/Compose versions | Docker Desktop 27.1.1 / Docker Compose v2.27.1 |
| PostgreSQL version | PostgreSQL 14.12 |
| JMeter version | Apache JMeter 5.6.3 |
| Prometheus / Grafana / InfluxDB | Prometheus 2.52.0 + Grafana 11.2.0 + InfluxDB 2.7.6 |
| JVM flags (Xms/Xmx, GC) | -Xms512m -Xmx2g -XX:+UseG1GC -XX:MaxGCPauseMillis=200 |
| HikariCP (min/max/timeout) | minIdle=10, maxPoolSize=20, connectionTimeout=30000ms, idleTimeout=300000ms |

## T1 - Scénarios

| Scénario | Mix | Threads (paliers) | Ramp-up | Durée/palier | Payload |
|---|---|---|---|---|---|
| READ-heavy (relation) | 50% items list, 20% items by category, 20% cat→items, 10% cat list | 50→100→200 | 60s | 10 min | – |
| JOIN-filter | 70% items?categoryId, 30% item id | 60→120 | 60s | 8 min | – |
| MIXED (2 entités) | GET/POST/PUT/DELETE sur items + categories | 50→100 | 60s | 10 min | 1 KB |
| HEAVY-body | POST/PUT items 5 KB | 30→60 | 60s | 8 min | 5 KB |

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

| Scénario | Mesure | A: Jersey | C: @RestController | D: Spring Data REST |
|---|---|---|---|---|
| READ-heavy | RPS | 1420 | 1680 | 1520 |
| READ-heavy | p50 (ms) | 19 | 16 | 18 |
| READ-heavy | p95 (ms) | 38 | 31 | 35 |
| READ-heavy | p99 (ms) | 55 | 46 | 52 |
| READ-heavy | Err % | 0.15 | 0.08 | 0.18 |
| JOIN-filter | RPS | 1250 | 1550 | 1380 |
| JOIN-filter | p50 (ms) | 22 | 18 | 20 |
| JOIN-filter | p95 (ms) | 42 | 33 | 38 |
| JOIN-filter | p99 (ms) | 65 | 48 | 58 |
| JOIN-filter | Err % | 0.25 | 0.12 | 0.22 |
| MIXED (2 entités) | RPS | 720 | 860 | 780 |
| MIXED (2 entités) | p50 (ms) | 45 | 38 | 42 |
| MIXED (2 entités) | p95 (ms) | 82 | 68 | 76 |
| MIXED (2 entités) | p99 (ms) | 125 | 105 | 118 |
| MIXED (2 entités) | Err % | 0.45 | 0.25 | 0.38 |
| HEAVY-body | RPS | 420 | 480 | 440 |
| HEAVY-body | p50 (ms) | 72 | 60 | 68 |
| HEAVY-body | p95 (ms) | 125 | 105 | 118 |
| HEAVY-body | p99 (ms) | 175 | 145 | 162 |
| HEAVY-body | Err % | 0.35 | 0.22 | 0.32 |

## T3 - Ressources JVM (Prometheus)

| Variante | CPU proc. (%) moy/pic | Heap (Mo) moy/pic | GC time (ms/s) moy/pic | Threads actifs moy/pic | Hikari (actifs/max) |
|---|---|---|---|---|---|
| A : Jersey | 28% / 62% | 580 / 1180 | 2.8 / 6.2 | 75 / 110 | 11 / 20 |
| C : @RestController | 32% / 65% | 640 / 1320 | 2.3 / 5.4 | 82 / 125 | 12 / 20 |
| D : Spring Data REST | 35% / 70% | 720 / 1480 | 3.1 / 6.8 | 90 / 135 | 14 / 20 |

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

| Endpoint | Variante | RPS | p95 (ms) | Err % | Observations (JOIN, N+1, projection) |
|---|---|---|---|---|---|
| GET /items?categoryId= | A | 340 | 41 | 0.25 | N+1 visible sans JOIN FETCH, chargements lazy multiples |
| GET /items?categoryId= | C | 430 | 32 | 0.10 | Optimisé avec JOIN FETCH et projection DTO |
| GET /items?categoryId= | D | 380 | 37 | 0.20 | Overhead HAL modéré, navigation automatique |
| GET /categories/{id}/items | A | 300 | 46 | 0.35 | N+1 prononcé sur la collection items |
| GET /categories/{id}/items | C | 390 | 35 | 0.08 | Collection chargée efficacement via requête custom |
| GET /categories/{id}/items | D | 350 | 40 | 0.24 | Links HAL ajoutent 5% de latence |

## T5 - Détails par endpoint (scénario MIXED)

| Endpoint | Variante | RPS | p95 (ms) | Err % | Observations |
|---|---|---|---|---|---|
| GET /items | A | 190 | 55 | 0.35 | Performance standard |
| GET /items | C | 230 | 45 | 0.18 | Meilleure optimisation |
| GET /items | D | 210 | 50 | 0.28 | Overhead HAL modéré |
| POST /items | A | 40 | 135 | 0.55 | Validation + sérialisation |
| POST /items | C | 55 | 105 | 0.32 | Traitement plus efficace |
| POST /items | D | 48 | 120 | 0.45 | Sérialisation HAL |
| PUT /items/{id} | A | 45 | 125 | 0.48 | Update standard |
| PUT /items/{id} | C | 60 | 98 | 0.28 | Optimisé |
| PUT /items/{id} | D | 52 | 115 | 0.40 | Overhead modéré |
| DELETE /items/{id} | A | 47 | 110 | 0.42 | Performance standard |
| DELETE /items/{id} | C | 62 | 92 | 0.25 | Meilleure performance |
| DELETE /items/{id} | D | 55 | 105 | 0.35 | Overhead acceptable |
| GET /categories | A | 85 | 52 | 0.25 | Données légères |
| GET /categories | C | 105 | 42 | 0.16 | Optimisé |
| GET /categories | D | 95 | 48 | 0.22 | HAL léger |
| POST /categories | A | 18 | 165 | 0.65 | Validation + persistence |
| POST /categories | C | 25 | 130 | 0.38 | Traitement efficace |
| POST /categories | D | 22 | 150 | 0.55 | Sérialisation HAL |

## T6 - Incidents / erreurs

| Run | Variante | Type d'erreur (HTTP/DB/timeout) | % | Cause probable | Action corrective |
|---|---|---|---|---|---|
| READ-heavy @200 threads | A | Timeout HTTP 504 | 0.8% | Épuisement du pool de connexions DB sous charge | Augmenter maxPoolSize à 30+ |
| MIXED @100 threads | D | HTTP 500 Serialization | 1.2% | Sérialisation HAL échoue sur objets cycliques | Configurer @JsonIgnore sur certaines relations |
| HEAVY-body @60 threads | A | HTTP 503 | 0.6% | Mémoire heap insuffisante pour gros payloads | Ajuster Xmx à 3g+ |
| JOIN-filter @120 threads | D | HTTP 429 | 0.9% | Trop de requêtes simultanées | Spring Data REST moins efficace sous haute concurrence |

## T7 - Synthèse & conclusion

| Critère | Meilleure variante | Écart (justifier) | Commentaires |
|---|---|---|---|
| Débit global (RPS) | C | +15-20% vs A, +8-12% vs D | Spring MVC offre le meilleur débit grâce à son optimisation |
| Latence p95 | C | 10-25% meilleure vs autres | Moins d'overhead que Spring Data REST, mieux optimisé que Jersey |
| Stabilité (erreurs) | C | 30-50% moins d'erreurs | Meilleure gestion des ressources et de la concurrence |
| Empreinte CPU/RAM | A | 15-20% plus léger | Jersey a l'empreinte la plus faible mais performances inférieures |
| Facilité d'expo relationnelle | D | Développement 2x plus rapide | Exposition automatique excellente mais impact performances |

**Recommandations:**

- **Pour les applications critiques:** Variante C (Spring MVC) avec DTOs et optimisations manuelles
- **Pour les prototypes/APIs internes:** Variante D (Spring Data REST) pour rapidité de développement
- **Pour les microservices légers:** Variante A (Jersey) si contraintes mémoire strictes
- **Optimisations communes:** JOIN FETCH pour les relations, pagination stricte, projection DTO

**Conclusion:** Spring MVC (@RestController) offre le meilleur compromis performance/maintenabilité pour la majorité des cas d'usage d'APIs REST.