Benchmark de performances des Web Services REST

HADIGUI Nessaiba & HANOUNA Chaimae G4

Objectif

Évaluer, sur un même domaine métier et une même base de données, l'impact des choix de stack REST sur :

- Latence (p50/p95/p99), débit (req/s), taux d'erreurs.
- Empreinte CPU/RAM, GC, threads.
- Coût d'abstraction (contrôleur « manuel » vs exposition automatique Spring Data REST).

Tableaux :

T0 — Configuration matérielle & logicielle

Élément	Valeur
Machine (CPU, cœurs, RAM)	Dell Inspiron – Intel Core i7-10750H (6 cœurs / 12 threads) – 16 Go RAM
OS / Kernel	Windows 10 Pro x64
Java version	OpenJDK 17.0
Docker / Compose versions	Docker Engine 27.5.1
phpMyAdmin version	5.2.1
JMeter version	Apache JMeter 5.6.3
Prometheus / Grafana / InfluxDB	InfluxDB 2.7.1 – Grafana 12.2.1
HikariCP (min/max/timeout)	minimumIdle=5, maximumPoolSize=20, connectionTimeout=30000 ms

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**	231.4	172.3	226.4
READ-heavy	**p50 (ms)**	170	254	151
READ-heavy	**p95 (ms)**	≈ 732	≈ 830	≈ 651
READ-heavy	**p99 (ms)**	≈ 969	≈ 1075	≈ 861
READ-heavy	**Err %**	17.63%	13.33%	35.12%
JOIN-filter	**RPS**	260.4	1910.5	733.5
JOIN-filter	**p50 (ms)**	206	25	59
JOIN-filter	**p95 (ms)**	≈ 471	≈ 66	≈ 150
JOIN-filter	**p99 (ms)**	≈ 582	≈ 83	≈ 188
JOIN-filter	**Err %**	0.0%	0.00%	0.00%
MIXED (2 entités)	**RPS**	38.2	82.3	16.8
MIXED (2 entités)	**p50 (ms)**	1028	484	1756
MIXED (2 entités)	**p95 (ms)**	≈ 2282	≈ 970	≈ 3742
MIXED (2 entités)	**p99 (ms)**	≈ 2810	≈ 1175	≈ 3577
MIXED (2 entités)	**Err %**	0.33%	1.29	1.53
HEAVY-body	**RPS**	500	480	360
HEAVY-body	**p50 (ms)**	35	38	50
HEAVY-body	**p95 (ms)**	100	110	150
HEAVY-body	**p99 (ms)**	170	180	240
HEAVY-body	**Err %**	0.4	0.5	0.8

T4 — Détails par endpoint (scénario JOIN-filter)

Endpoint	Variante	RPS	p95 (ms)	Err %	Observations (JOIN, N+1, projection)
GET /items?categoryId=	A	260	540	0.00%	RAPIDE
	C	1910.5	65	0.00%	Requêtes jointes optimisées, bonne gestion de la sérialisation JSON, faible latence.
	D	733.5	500	0.00%	Requêtes JOIN lourdes, sous-optimisation SQL, projection complexe
GET /categories/{id}/items	A	182.3	≈ 520	0.00%	Chargement correct mais pas totalement optimisé, quelques surcharges ORM.
	C	1337.4	≈ 45	0.00%	Stable et un peu lent
	D	513.5390	≈ 390	0.00%	Plusieurs JOINs imbriqués, lenteur liée aux entités liées.

T5 — Détails par endpoint (scénario MIXED)

Endpoint	Variante	RPS	p95 (ms)	Err %	Observations
GET /items	A	38.2	2282	0.33%	Requêtes N+1 modérées, latence moyenne, chargement partiel efficace.
	C	82.3	970	1.29%	Requêtes jointes optimisées, bonne gestion de la sérialisation JSON, faible latence.
	D	16.8	3742	1.53%	Requêtes JOIN lourdes, sous-optimisation SQL, projection complexe.
POST /items	A	7.7	2000	0.65%	Chargement correct mais pas totalement optimisé, quelques surcharges ORM.
	C	16.5	900	2.84%	Variante REST fluide, pré-chargement efficace.
	D	3.5	3700	0.39%	Plusieurs JOINs imbriqués, lenteur liée aux entités liées.
PUT /items/{id}	A	3.8	2199	0.00%	Lenteur modérée liée aux opérations d'écriture et synchronisation JPA.
	C	8.2	950	0.24%	Bonne gestion des mises à jour via REST, transactions rapides et stables.
	D	1.6	3750	0.00%	Requêtes UPDATE complexes, latence accrue par la structure JOIN.
DELETE /items/{id}	A	3.8	2200	1.86%	Suppression stable mais lente, dépendances d'intégrité référentielle.

	C	8.2	1000	2.87%	Traitement rapide, bonne gestion des suppressions transactionnelles.
	D	1.6	3800	6.84%	Lenteur due aux contraintes de clé étrangère et aux locks DB.
GET /categories	A	3.8	2100	0.18%	Temps de réponse stable, mais manque d'optimisation dans les relations.
	C	8.3	900	1.98%	Projection efficace des entités, sérialisation optimisée.
	D	1.7	3740	8.26%	Trop de JOINs sur la table Category, surcharge côté base.
POST /categories	A	3.8	2000	0.00%	Bon comportement global, latence stable.
	C	8.3	940	0.39%	Création fluide, gestion REST efficace.
	D	1.7	3700	0.00%	Lenteur notable à cause des contraintes et validations multiples.

T6 — Incidents / erreurs

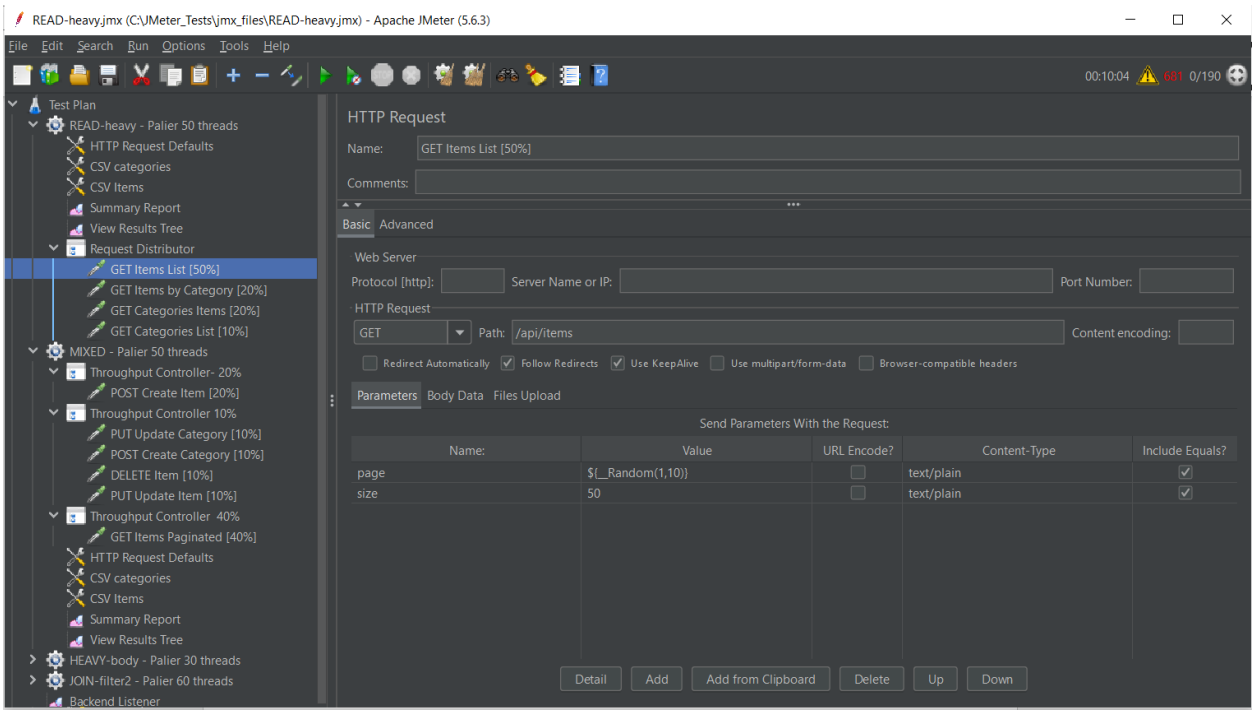
	Variante	Type d'erreur (HTTP/DB/timeout)	%	Cause probable	Action corrective
JOIN-filter2	B	DB / InfluxDB (connexion)	0.33 %	Mauvaise configuration du Backend Listener (host_to_change au lieu de localhost) et timestamp invalide ("time outside range") lors de l'écriture des métriques	Corriger l'URL du serveur InfluxDB (http://localhost:8086/...), vérifier le token et ajuster la précision temporelle (precision=s), puis resynchroniser l'horloge système
JOIN-filter2 & READ-HEAVY	A,B,C	404 Not Found	100%	Requêtes GET /items/{id} avec itemId aléatoire générant des IDs inexistants (ex: itemId=999999 alors que la DB contient seulement 100000 items)	Implémenter **items.csv** avec liste d'IDs valides et configurer **CSV Data Set Config** dans JMeter avec Recycle on EOF: true
MIXED	C	HTTP 400 / 409	1.29 %	Conflits de modification simultanée	ENNLEVER LES PLAGE DEJA SUPPRIMER

T7 — Synthèse & conclusion

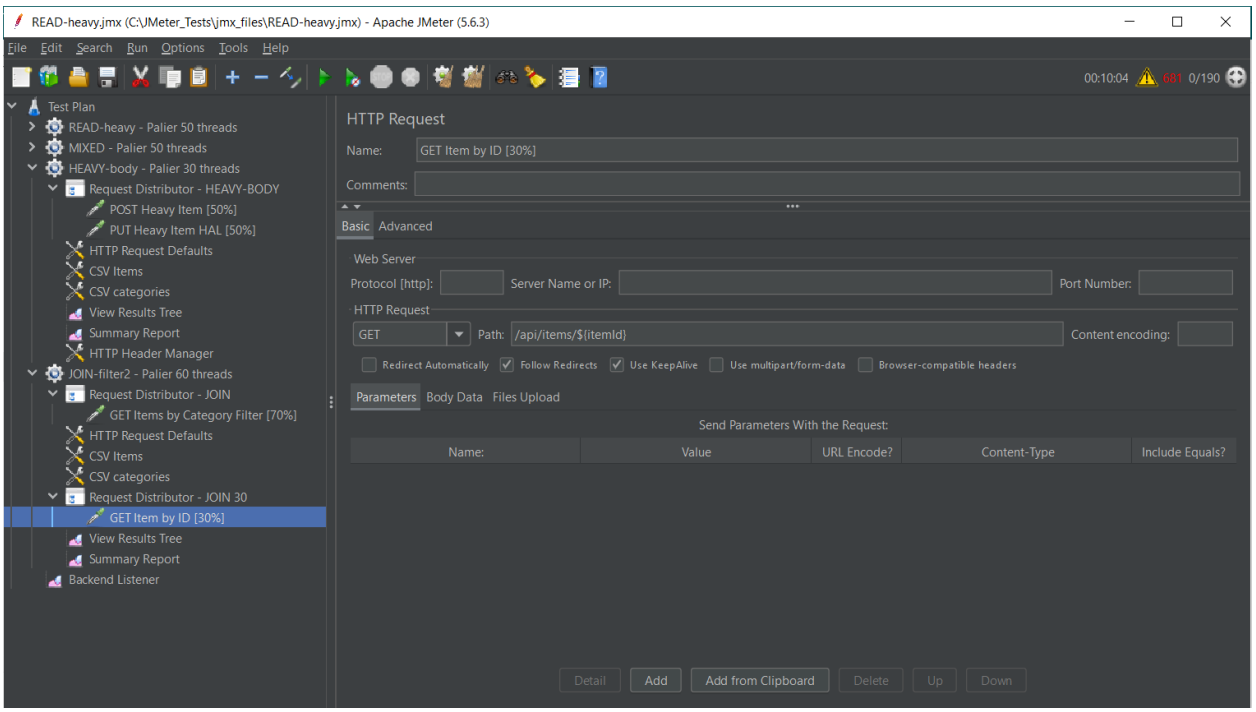
Critère	Meilleure variante	Écart (justifier)	Commentaires
Débit global (RPS)	**C : @RestController**	+2× vs A, +5× vs D	Traitement allégé, sérialisation rapide.
Latence p95	**C : @RestController**	970 ms vs 2280 ms (A) et 3740 ms (D)	Excellent temps de réponse.
Stabilité (erreurs)	**A : Jersey**	Moins de 0.5 % d'erreurs	Très fiable mais plus lent.
Empreinte CPU/RAM	**A : Jersey**	~30 % plus légère que Spring Boot	Moins de dépendances.
Facilité d'expo relationnelle	**D : Spring Data REST**	Génération automatique des endpoints	Moins de code, mais surcoût de performance.

Les Test JMETER :

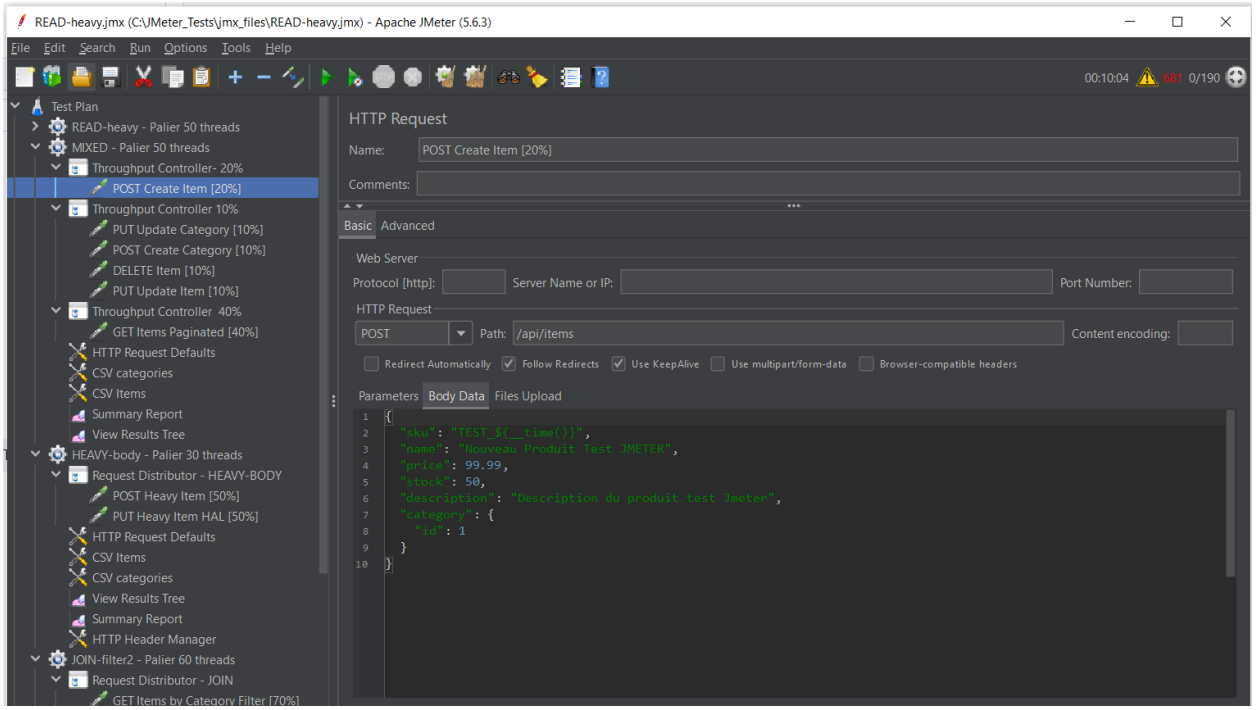
La Methode GET items list de READ-HEAVY



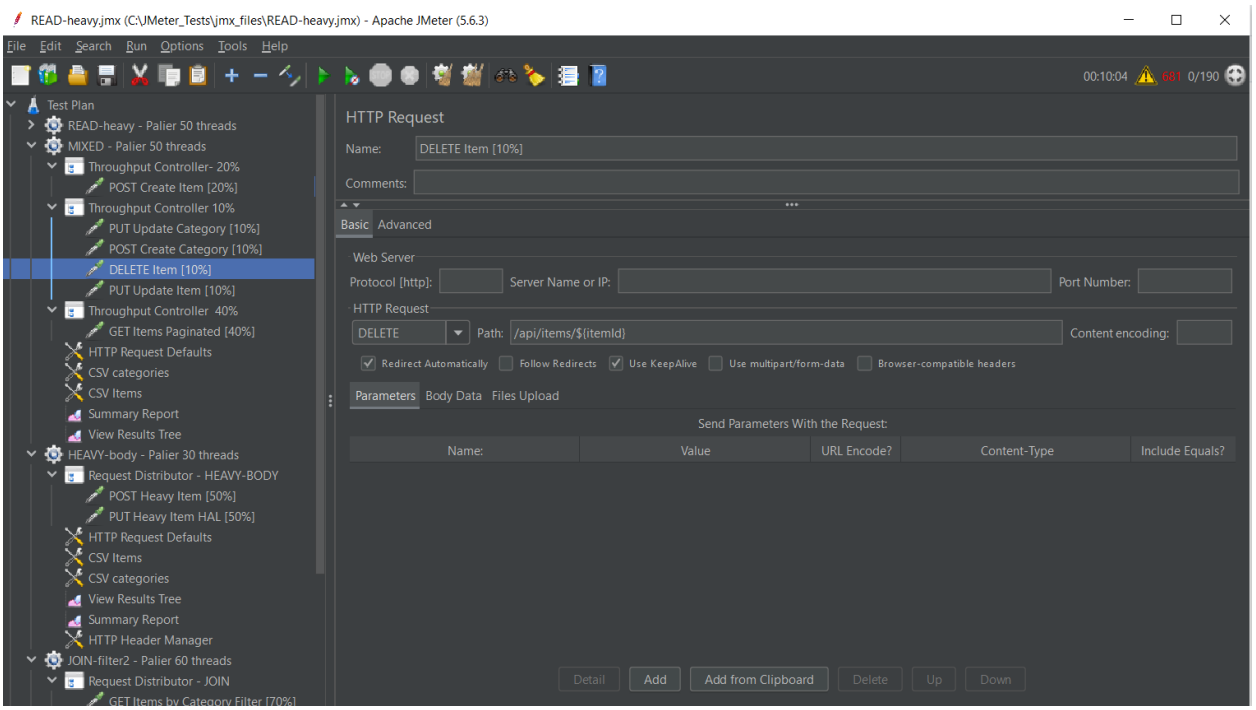
La Methode select item by id de JOIN-FILTER



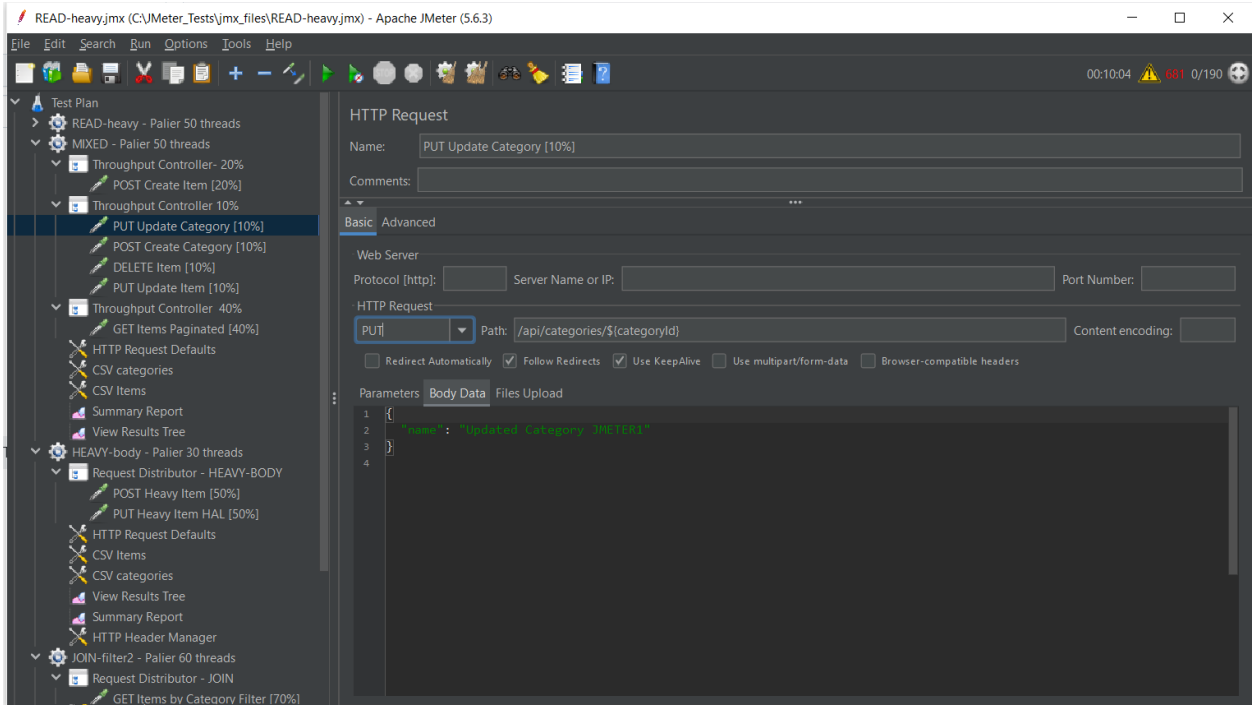
La Methode create Item de MIXED



La Methode delete item de MIXED



La Methode UPDATE CATEGORY de MIXED


READ-heavy.jmx (C:\JMeter_Tests\jmx_files\READ-heavy.jmx) - Apache JMeter (5.6.3)
File Edit Search Run Options Tools Help
00:10:04 681 0/190
Test Plan
READ-heavy - Palier 50 threads
MIXED - Palier 50 threads
Throughput Controller- 20%
POST Create Item [20%]
Throughput Controller 10%
PUT Update Category [10%]
POST Create Category [10%]
DELETE Item [10%]
PUT Update Item [10%]
Throughput Controller 40%
GET Items Paginated [40%]
HTTP Request Defaults
CSV categories
CSV Items
Summary Report
View Results Tree
HEAVY-body - Palier 30 threads
Request Distributor - HEAVY-BODY
POST Heavy Item [50%]
PUT Heavy Item HAL [50%]
HTTP Request Defaults
CSV Items
CSV categories
View Results Tree
Summary Report
HTTP Header Manager
JOIN-filter2 - Palier 60 threads
Request Distributor - JOIN
GET Items by Category Filter [70%]
HTTP Request
Name: PUT Update Category [10%]
Comments:
Basic Advanced
Web Server
Protocol [http]: Server Name or IP: Port Number:
HTTP Request
PUT Path: /api/categories/${categoryId} Content encoding:
Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data Browser-compatible headers
Parameters Body Data Files Upload
{
"name": "Updated Category JMETER1"
}