Relatório de Benchmark: Requisitos de Hardware para 1000 RPS

Escopo do Teste

O benchmark simula um fluxo de trabalho típico de backend através de dois endpoints:

1. `POST /bonus` **(Escrita)**: Exige validação de cliente em banco de dados, aplicação de regra de bônus condicional e persistência.
2. `GET /bonus/recents` **(Leitura + Processamento)**: Busca 100 registros do banco e realiza a ordenação por data **dentro da aplicação**. Este endpoint foi desenhado para medir a eficiência da linguagem em processamento de coleções e uso de memória sob carga.

Resumo Executivo

Este documento analisa a eficiência de hardware de diferentes stacks tecnológicas ao sustentar uma carga constante de **1000 requisições por segundo (RPS)**, mantendo a latência **P95 abaixo de 200ms**.

O diferencial deste teste foi observar quanto de recurso (CPU e Memória) cada stack alocou e efetivamente consumiu sob uma carga de trabalho idêntica e pré-definida.

Metodologia do Teste

Estratégia de Carga (k6)

O script de teste k6 seguiu um rigoroso processo de aquecimento e estabilização:

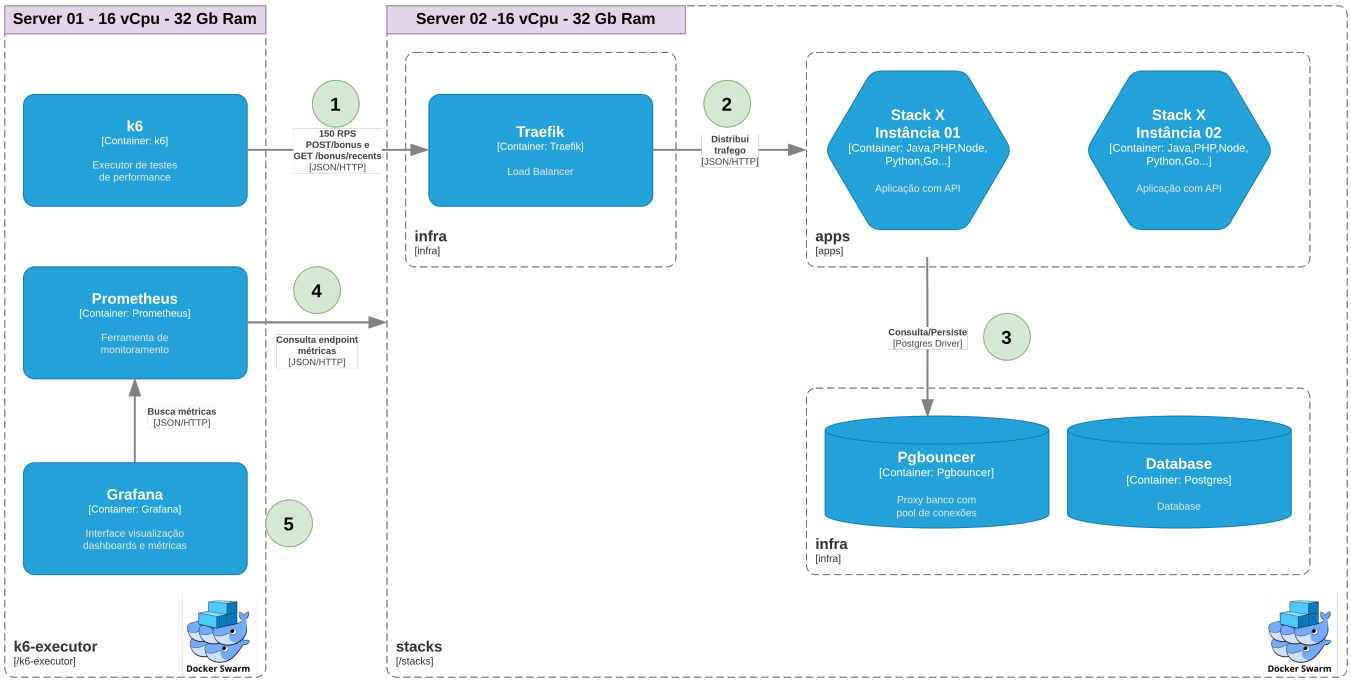
- **Aquecimento (Warm-up):** Uma fase inicial de **5 minutos** com carga progressiva (20 a 500 rate/s) para preparar o ambiente e as aplicações.
- **Carga Constante:** Após o aquecimento, foi aplicada uma carga fixa de **500 iterações/segundo** por **5 minutos**. Cada iteração realiza 2 chamadas (1 POST + 1 GET), totalizando exatamente **1000 RPS**.
- **Validação de SLA:** O threshold de sucesso foi definido como **P95 < 200ms** e taxa de erro inferior a **1%** durante a fase de carga real.

Pontos importantes

- Foram executadas repetitivas baterias de testes calibrando o hardware até encontrar o hardware minímo de cada stack para atender o teste.
- Stack PHP Laravel utilizando FPM foi removido do teste porque conseguiu alcançar somente 600 rps mesmo tendo 5 instâncias, 10 core alocado de cpu e 10gb de ram.
- Esse teste não é util para comparar tempos de resposta porque foi utilizado o menos hardware em cada um, ou seja, tem hardwares diferentes em cada stack.

Infraestrutura e Coleta

As aplicações rodaram no **Docker Swarm** com diferentes níveis de alocação de hardware para garantir a estabilidade do P95. As métricas foram coletadas via **Prometheus**, consolidando dados do Swarm (container) e do **Traefik** (edge router). Segue abaixo diagrama explicando a infraestrutura envolvida no teste:



- 1 - Na máquina 01 execução do teste pelo K6 enviando chamadas
- 2 - Na máquina 02 as aplicações de cada stack rodando de forma separada atendendo as requisições, tendo como ponto de entrega Traefik
- 3 - Na máquina 02 persiste e busca bônus no postgres
- 4 - Na máquina 01 ambiente de monitoramento com prometheus consulta os endpoints swarm e traefik para coletar métricas
- 5 - Na máquina 01 Grafana expoe dashboards para visualizar as métricas durante o teste

---

Resultados Consolidados: Eficiência de Hardware (1000 RPS)

Abaixo, os dados de infraestrutura e performance coletados durante a execução estável de 1000 RPS:

Infraestrutura e Consumo (Docker Swarm)

Stack	Instâncias	CPU Alocado (Total)	CPU Usado (Total)	Mem. Alocada (Total)	Mem. Usada (Total)
Rust Axum	2	0,52 core	0,39 core	512 MiB	16 MiB
Java Quarkus	2	1,04 core	0,59 core	512 MiB	471 MiB
Java MVC VT	2	1,04 core	0,74 core	512 MiB	504 MiB
Java WebFlux	2	2,00 core	1,35 core	512 MiB	479 MiB
Node.js (Fastify)	2	2,00 core	1,18 core	512 MiB	223 MiB
Java MVC Without VT	2	2,00 core	0,87 core	512 MiB	509 MiB

Stack	Instâncias	CPU Alocado (Total)	CPU Usado (Total)	Mem. Alocada (Total)	Mem. Usada (Total)
Node.js (Express)	2	3,00 core	1,41 core	512 MiB	232 MiB
.NET Core	2	3,00 core	1,70 core	512 MiB	187 MiB
Golang Gin	2	4,00 core	1,10 core	512 MiB	32 MiB
Python FastAPI	3	6,00 core	3,48 core	1536 MiB	749 MiB
PHP Laravel Octane	8	8,00 core	3,44 core	6048 MiB	2957 MiB

Performance de Rede (K6 & Traefik)

Todas as stacks listadas abaixo cumpriram o SLA de **P95 < 200ms** para 1000 RPS.

Stack	P95 K6 (ms)	P95 Traefik (ms)	Sucesso %	Status
Golang Gin	23,82	11,99	99,94%	✅
Java Quarkus	42,46	9,84	99,80%	✅
Rust Axum	46,25	36,45	99,64%	✅
Node.js (Express)	62,83	48,82	99,82%	✅
Python FastAPI	65,55	37,29	99,69%	✅
Java MVC Without VT	42,81	9,95	99,53%	✅
PHP Laravel Octane	117,40	46,57	99,77%	✅
Java WebFlux	137,05	71,12	99,74%	✅
.NET Core	136,52	9,77	99,54%	✅
Java MVC VT	138,25	20,30	99,72%	✅
Node.js (Fastify)	182,56	44,17	99,57%	✅

Materiais/Documentos

O código fonte das aplicações e teste escrito está disponível em https://github.com/crmbonus-oficial/benchmark-stacks/benchmark/benchmark-1000rps

- `load-all-swarm-1000rps.js`: script com cenário de teste k6.
- `graficos-grafana.md`: Links dos dashboards grafana com as métricas de performance obtidas ao longo do teste.
- `reports`: Relatórios gerados pelo k6 resumindo o teste executado.

Análise Financeira: Estimativa de Custos

Para complementar a análise técnica, foi realizada uma simulação de custos utilizando:

- **Cloud:** Amazon Web Services (AWS)
- **Serviço:** AWS Fargate
- **Região:** US East (Ohio)
- **Data da consulta:** 24/02/2026
- **Horas consideradas:** 730 horas/mês (24x7)

Preços Utilizados

Recurso	Preço
vCPU / hora	US$ 0,040480
Memória GB / hora	US$ 0,004445

> ⚠️ Observação: Os valores abaixo consideram apenas custo de compute (containers). Não incluem banco de dados, tráfego de rede, NAT Gateway, storage ou observabilidade.

Custo Bruto (Consumo Exato Medido no Benchmark)

Simulação considerando exatamente o hardware mínimo necessário identificado no teste.

Stack	Instâncias	vCPU Por Instância	Memória Por Instância	Custo Mensal	Custo Anual
Rust Axum	2	0,256	0,256	R$ 16,79	R$ 201,49
Java Quarkus	2	0,512	0,256	R$ 31,92	R$ 383,05
Java MVC VT	2	0,512	0,256	R$ 31,92	R$ 383,05
Java WebFlux	2	1	0,256	R$ 60,76	R$ 729,15
Node.js (Fastify)	2	1	0,256	R$ 60,76	R$ 729,15
Java MVC Without VT	2	1	0,256	R$ 60,76	R$ 729,15
Node.js (Express)	2	1,5	0,256	R$ 90,31	R$ 1.083,75
.NET Core	2	1,5	0,256	R$ 90,31	R$ 1.083,75
Golang Gin	2	2	0,256	R$ 119,86	R$ 1.438,36
Python FastAPI	3	2	0,5	R$ 182,17	R$ 2.186,04

Stack	Instâncias	vCPU Por Instância	Memória Por Instância	Custo Mensal	Custo Anual
PHP Laravel Octane	8	1	0,756	R$ 256,03	R$ 3.072,34

Destaques Financeiros (Custo Bruto)

- Rust custa apresentou o custo mais baixo para sustentar os mesmos 1000 RPS.
- Java (Quarkus / MVC VT) apresenta excelente equilíbrio entre custo e previsibilidade.
- Python e PHP exigem investimento significativamente maior para manter o SLA.

Custo Ajustado aos Tamanhos Reais do Fargate

Como o Fargate possui combinações fixas de CPU/memória, foi realizada nova simulação respeitando os tamanhos válidos do serviço.

Stack	Instâncias	vCPU Total	Memória Total (GB)	Custo Mensal	Custo Anual
Rust Axum	2	0,256	0,5	**R$ 18,37**	**R$ 220,50**
Java Quarkus	2	0,512	1	R$ 36,75	R$ 440,99
Java MVC VT	2	0,512	1	R$ 36,75	R$ 440,99
Java WebFlux	2	1	2	R$ 72,08	R$ 864,96
Node.js (Fastify)	2	1	2	R$ 72,08	R$ 864,96
Java MVC Without VT	2	1	2	R$ 72,08	R$ 864,96
Node.js (Express)	2	2	4	R$ 144,16	R$ 1.729,92
.NET Core	2	2	4	R$ 144,16	R$ 1.729,92
Golang Gin	2	2	4	R$ 144,16	R$ 1.729,92
Python FastAPI	3	2	4	R$ 216,24	R$ 2.594,89
PHP Laravel Octane	8	1	2	R$ 288,32	R$ 3.459,85

Conclusões

1. **Eficiência Extrema:** O **Rust Axum** foi a stack mais eficiente, precisando alocar apenas **0,52 core** e **512 MiB** de RAM divididas em 2 instâncias para sustentar a carga total de 1000 RPS com excelente latência.
2. **Boa performance:** O **Java Quarkus** e **JAVA MVC VT** ficaram logo abaixo demonstrando boa eficiência com pouco hardware.

3. **Meio de tabela:** As stacks de meio de tabela — Java WebFlux, Node.js (Fastify), Java MVC sem Virtual Threads e .NET Core — conseguiram atingir 1000 RPS usando 2 cores no total, mostrando um equilíbrio entre desempenho e custo. Não são as mais eficientes em hardware, mas entregam performance consistente com infraestrutura simples e previsível.
4. **Interpretadas vs Compiladas:** Stacks como Python e PHP requerem significativamente mais instâncias e CPU total para entregar o mesmo throughput com a latência desejada em comparação a Rust, Go ou Java.