# ESTUDO COMPARATIVO DE DESEMPENHO ENTRE API DESENVOLVIDA COM SPRING WEBFLUX E NODE.JS

**Estêvão Henrique Cangussú de Souza[1], Rafael Isaías Reis[1], Fabrício Gustavo Henrique[2]**

[1]Faculdade de Tecnologia de Ribeirão Preto (FATEC)

Ribeirão Preto, SP – Brasil

estevao_souza16@hotmail.com, rafaelisaiasreis@gmail.com, fabricio.henrique@fatec.sp.gov.br

***Resumo.*** *Devido ao avanço das tecnologias, as APIs passaram a exercer um papel fundamental no processo de modernização e unificação dos sistemas. Com isso, surgiram novas ferramentas voltadas para o seu desenvolvimento, como o Ecossistema Spring e o Node.js. Embora ambas sejam capazes de estar envolvidas com a criação de APIs a nível comercial, as duas possuem suas particularidades quanto a forma de implementação e execução dos processos. Com isso, este trabalho teve como objetivo analisar e comparar o desempenho entre uma API desenvolvida com Spring WebFlux e Node.js, através do software medidor de desempenho Apache JMeter, tendo como resultado uma melhor performance a API desenvolvida com o Spring WebFlux.*

***Abstract.*** *Due to the advancement of technologies, APIs started to play a fundamental role in the process of modernization and unification of systems. With that, new tools have emerged focused on its development, such as the Ecosystem Spring and Node.js. Although both are able to be involved with the creation of APIs at a commercial level, both have their particularities as to how to implement and execute the processes. Thus, this work aimed to analyze and compare the performance between an API developed with Spring WebFlux and Node.js, through the Apache JMeter performance meter software, resulting in a better performance the API developed with the Spring Webflux.*

## 1. Introdução

Na história da computação, os anos 90 do século XX foram agitados, primeiramente com o surgimento da linguagem de programação Java, seguido pelo JavaScript, nomes que os faziam parecer gêmeos recém-nascidos, entretanto, não poderiam ser mais diferentes, sendo o Java compilado e digitado estaticamente e o JavaScript interpretado e digitado dinamicamente, sendo isto, apenas o começo das diferenças técnicas entre essas duas linguagens totalmente distintas que, desde então, entraram em uma espécie de curso de colisão devido as suas áreas de aplicações, como no desenvolvimento de APIs [Wayner 2019].

A sigla API refere-se ao termo inglês “Application Programming Interface” que traduzido para o português remete a “Interface de Programação de Aplicativos”. Uma API, é um aglomerado de rotinas e padrões de programação que servem para realizar o acesso a um aplicativo de software ou plataforma fundamentada na web [Canaltech 2020].

A criação de uma API ocorre quando uma empresa possui a intenção de que outras organizações desenvolvam produtos que se associem aos seus serviços [Canaltech 2020].

A linguagem Java foi concebida por James Gosling e o "Green Team" em 1991. Em muitos aspectos, o Java foi o pioneiro, trazendo o primeiro exemplo de uma implementação bem sucedida em diferentes dispositivos, demonstrando recursos de multiplataforma, estabilidade e eficiência. Hoje é uma das linguagens de programação mais conhecidas e utilizadas no mundo [Okuneu 2018].

O Ecossistema Spring representa uma completa plataforma de recursos para o desenvolvimento de aplicativos Java, criado com o intuito de simplificar as práticas de programação do Java EE, com vários módulos que buscam auxiliar a construção de sistemas para a redução do tempo gasto no desenvolvimento. Conta com recursos extremamente avançados que englobam várias áreas de uma aplicação com módulos prontos para serem utilizados, como o Spring Boot, Spring Data, Spring WebFlux, entre outros [Brito 2020].

Já o JavaScript foi criado em 1995 por Brendan Eich a serviço da Netscape [Luiz 2016] e hoje é tida como uma linguagem leve, interpretada e baseada em objetos, conhecida principalmente como uma linguagem de script para páginas web, entretanto utilizada não só neste ramo especificamente mas em vários outros ambientes sem browser, sendo isso graças a algumas ferramentas, como o Node.js [MDN Web Docs 2020].

O Node.js, escrito por Ryan Dahl em 2009, possui um ambiente poderoso que permite que o JavaScript saia dos navegadores da web e os iniciantes na programação iniciem sua caminhada sem muitas dificuldades [Okuneu 2018]. A primeira coisa a esclarecer é que o Node.js não é uma linguagem de programação e nem um framework, mas sim um ambiente de tempo de execução JavaScript multiplataforma de código aberto que executa código JavaScript fora de um navegador. Atualmente, esta ferramenta vem sendo amplamente utilizada para construir serviços back-end [Okuneu 2018].

Embora ambas as tecnologias sejam totalmente capazes de estarem envolvidas com a criação de APIs, elas possuem suas diferenças quanto a forma de execução e implementação de processos, afetando assim, o desempenho das aplicações [Okuneu 2018]. Pensando nisto, este trabalho foi desenvolvido com o objetivo de se analisar possíveis diferenças de desempenho entre uma API desenvolvida com estas duas ferramentas.

## 2. Materiais e Métodos

Para a realização deste projeto, foram criadas duas APIs, sendo uma delas desenvolvida com Spring WebFlux e a outra com Node.js. Ambas as aplicações foram projetadas dentro de um mesmo contexto, sendo disponibilizado uma lista de recursos persistidos em banco de dados não-relacional (MongoDB) através de requisições HTTP GET.

### 2.1. Entrada e Saída de Dados

In-Out (I/O) ou Entrada-Saída (E/S) de dados refere-se à descrição do processo de inserção ou exibição de dados, ou seja, refere-se aos próprios dados inseridos ou exibidos pelo computador [Okuneu 2018]. Existem dois tipos de E/S encontrados na literatura:

I)E/S bloqueadoras/síncronas: neste caso, um thread não pode realizar mais nenhum procedimento até que a E/S seja totalmente processada, ou seja, um thread atende a uma solicitação e entra em um estado de espera, até que essa solicitação seja concluída [Okuneu 2018].

II)E/S não bloqueadoras/assíncronas: enquanto um thread auxilia uma solicitação de E/S, ele pode lidar com outras solicitações, sem a necessidade de aguardar a implementação de cada uma delas, ou seja, possibilita que um thread processe outra(s) solicitação(ões) até que a operação seja concluída, diminuindo o tempo de implementação. Caso necessário, a solicitação é enfileirada imediatamente e processada posteriormente [Okuneu 2018].

O Java possui uma forma de comunicação bloqueadora, com isso, tem de aguardar por cada solicitação de E/S ou então disparar um thread para cada solicitação. Entretanto, hoje com o uso do Spring WebFlux, é possível o Java se tornar assíncrono. Já os aplicativos Node são não bloqueadores, não sendo necessário nenhum esforço extra para se trabalhar com mais de uma solicitação de E/S [Okuneu 2018].

Resumindo: o bloqueio de E/S oferece uma programação linear e maior facilidade de codificação, porém, menos controle sobre o fluxo. Já o não bloqueio de E/S remete a programação paralela e maior dificuldade de codificação, porém, com mais controle sobre o fluxo [Okuneu 2018].

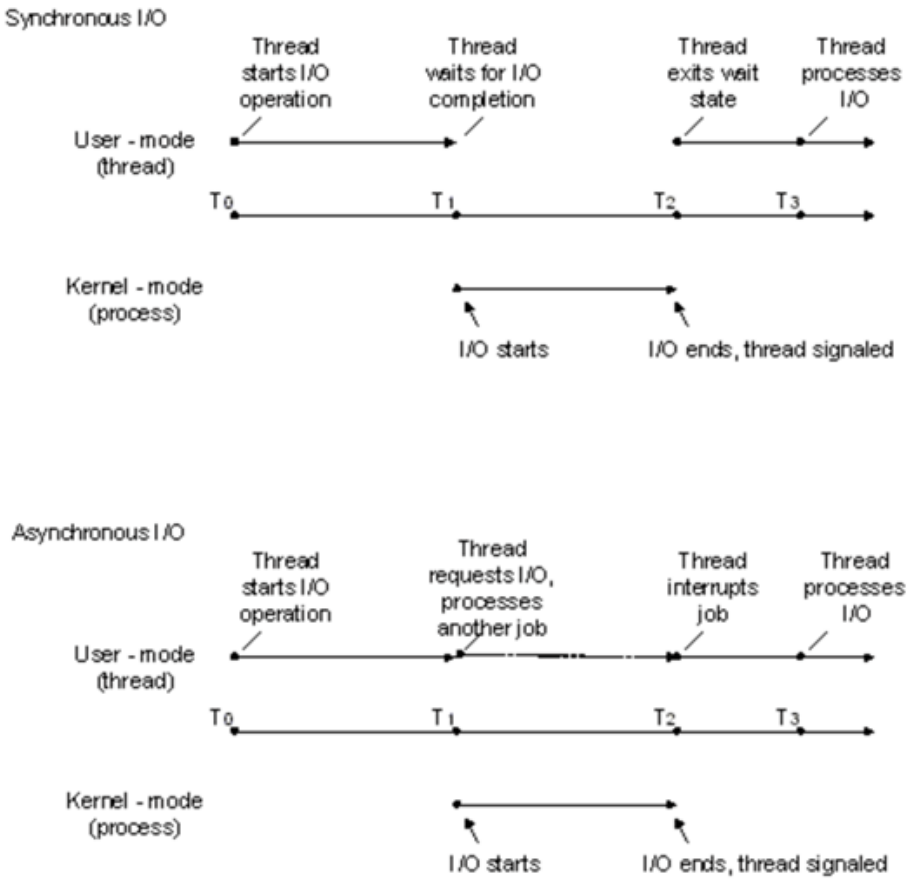


**Figura 1. E/S Síncrona e Assíncrona**

### 2.2. Spring WebFlux

O Spring WebFlux é um módulo que foi inserido no Spring Framework 5, que passou a possibilitar aplicações web com Spring em relação ao lado do servidor, proporcionando assim, trabalhar de forma reativa [Brito 2019]. Como definição da própria documentação do Spring (2020) sobre o WebFlux, temos: "Um framework web original incluído no Spring Framework Web MVC, foi desenvolvido especificamente para a API de Servlet e contêineres de Servlet. Um framework de desenvolvimento web de forma reativa, o Spring WebFlux foi adicionado posteriormente na versão 5.0. É totalmente não bloqueante, suporta contrapressão de streams reativos e é executado em servidores como Netty, Undertow e Servlet 3.1+".

Embora o Java seja multithread, o que significa que várias tarefas podem ser realizadas de forma simultânea dentro de um programa, até a implementação do Spring WebFlux as APIs utilizando o Ecossistema Spring lidavam com as requisições de forma síncrona. A programação multithread foi perfeitamente integrada ao Java, enquanto em outras linguagens, procedimentos específicos do sistema operacional devem ser chamados para habilitar tal comportamento [Okuneu 2018].

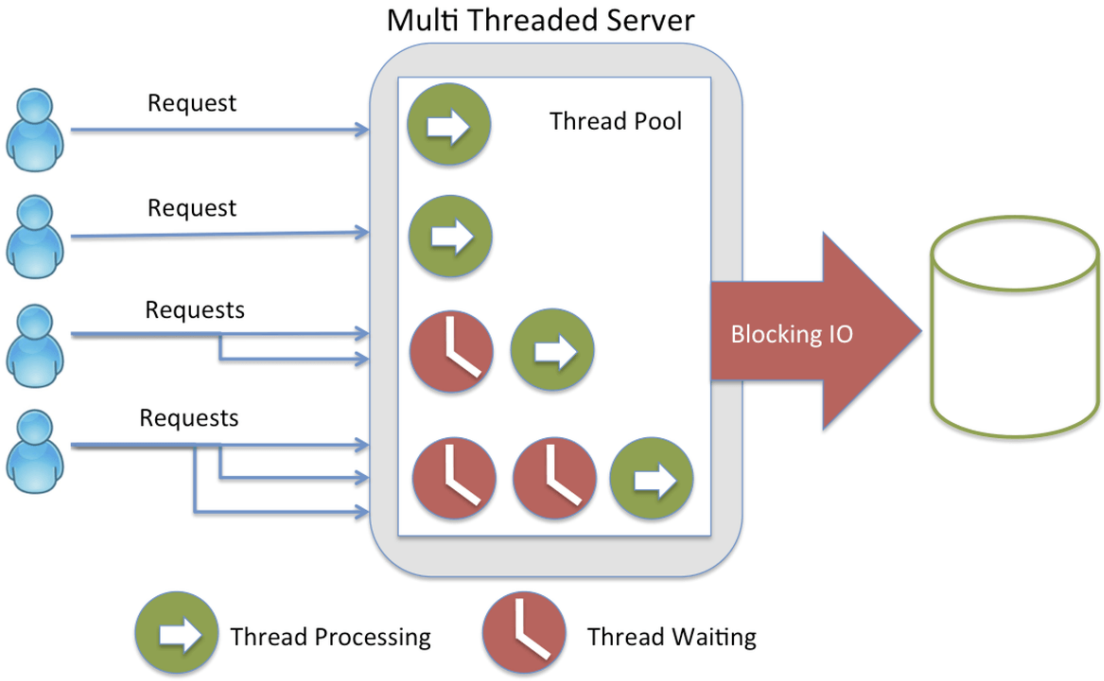


**Figura 2. Multithread**

Basicamente o multithreading auxilia os aplicativos a terem um melhor desempenho. Portanto, para projetos de grande escala que envolvem simultaneidade e uma alta performance do hardware, Java é altamente recomendado [Okuneu 2018].

### 2.3. Node.js

O Node.js é uma tecnologia que se baseia em uma plataforma que utiliza o JavaScript como linguagem de programação. Através dele, é possível desenvolver pequenas e

grandes aplicações. Uma de suas grandes vantagens é de ser de código aberto, além de possuir uma ampla comunidade. Uma das características principais do Node, é a utilização do NPM como gerenciador de pacotes e bibliotecas, que consiste no maior ecossistema de bibliotecas de código aberto do mundo [Brasil, 2019].

O Node.js é single-thread, o que significa que apresenta um único thread para lidar com todas as solicitações, ou seja, quando uma solicitação chega até a aplicação, esse thread é usado para tratá-la. Por exemplo, se for necessário realizar uma consulta ao bando de dados, um thread não precisa esperar que o banco de dados retorne os dados, pois durante a execução da consulta por parte do banco de dados, esse único thread será utilizado para atender a outra solicitação. Quando o banco de dados finaliza seu trabalho, ele coloca uma mensagem na fila de eventos, a qual é continuamente monitorada pelo Node, sendo que, após encontrar o evento, apenas o retira e o processa. Este tipo de arquitetura torna o Node ideal para aplicativos com E/S intensiva, os quais incluem muitas requisições ao banco ou acesso à rede, podendo assim, atender mais clientes sem a necessidade de solicitar mais do hardware. Esta é a principal característica que faz com que os aplicativos Node sejam altamente escalonáveis [Okuneu 2018].

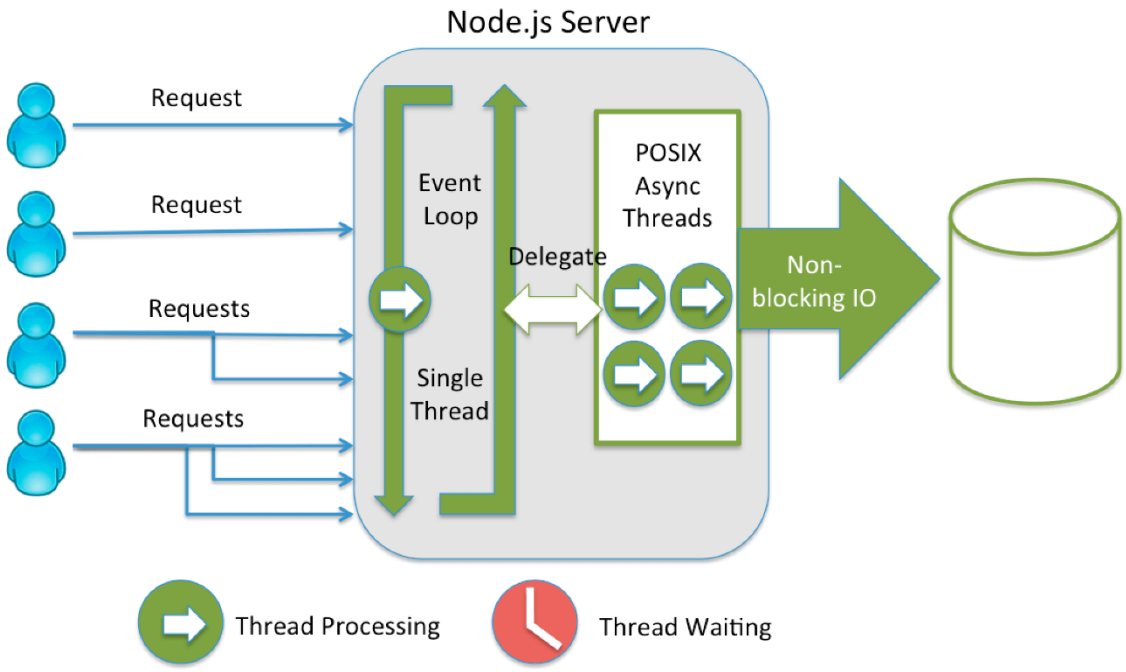


**Figura 3. Single-thread**

Entretanto, esta característica também possui seu lado negativo, pois ao realizar cálculos para atender a uma requisição, é necessário que outros clientes esperem sua finalização. Com isso, não é aconselhável a utilização do Node para aplicativos com intensivo uso de CPU [Okuneu 2018].

### 2.4. Apache JMeter

De acordo com a própria documentação do The Apache Software Fundation (2020), o Apache JMeter é um software de código aberto que originalmente foi criado para a

realização de testes de aplicações Web, mas que hoje apresenta outras funcionalidades, como, o teste de comportamento funcional de carga e a medição de desempenho de diversos serviços.

Este software permite a realização de diversos tipos de simulações, como por exemplo, o acesso simultâneo de usuários em diversas páginas web. O JMeter também fornece uma grande variedade de resultados, como relatórios, tabelas e gráficos que fornecem dados que auxiliam ao usuário visualizar e compreender como foi o correr da aplicação [Gomes 2008]. Esta ferramenta vem evoluindo e hoje possui a capacidade de carregar e testar o desempenho de diversos tipos de aplicativos, servidores e protocolos, como por exemplo: banco de dados via JDBC, FTP, LDAP, serviços web SOAP/REST, HTTP, conexões TCP genéricas e processos nativos do sistema operacional [The Apache Software Fundation 2020].

O JMeter oferece mecanismos que representam diversos componentes relacionados a aplicações web, permitindo assim testes deste tipo, como: grupos de threads, requisições, configurações básicas, sessões e cookies, entre muitos outros [Gomes 2008].

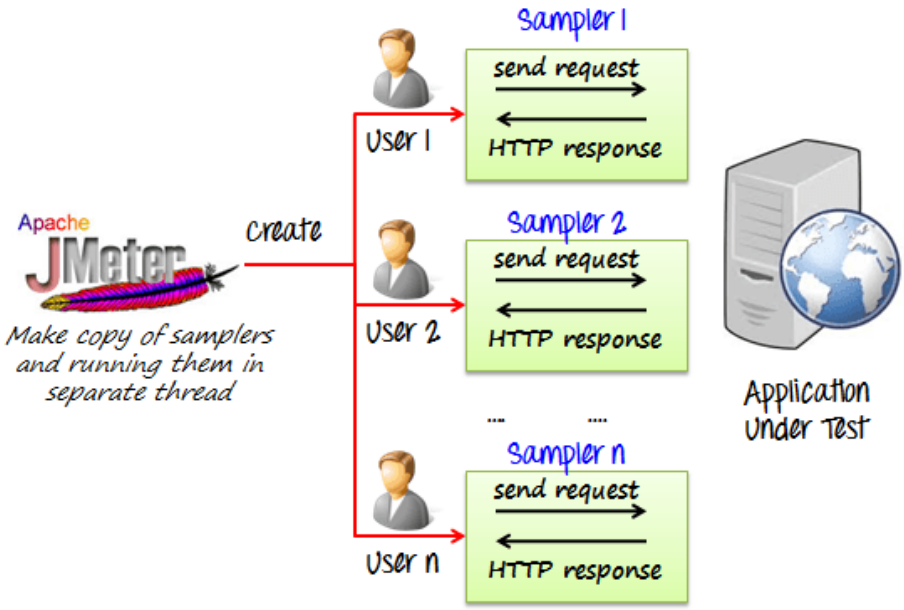


**Figura 4. Solicitações via Apache JMeter**

### 2.4.1. Configuração do JMeter

Para a execução do teste de carga, foram realizadas algumas configurações no JMeter com o intuito de fornecer uma carga pesada de requisições ao endpoint das APIs. Entre elas, temos alguns parâmetros que são essenciais para este tipo de teste, os quais são:

I) Número de threads (usuários): número de usuários que irão acessar a API durante a simulação;

II) Período de aceleração (em segundos): duração de tempo que o JMeter distribuirá o início dos threads;

III) Contagem de loop: número de vezes que o teste será executado.

Para análise dos dados, foram utilizados 100 (cem) usuários e uma contagem de loop igual a 10 (dez), tendo assim, 1.000 (mil) requisições dentro do tempo selecionado, o qual foi o único parametro alterado durante o teste, recebendo valores em segundos iguais a 60 (sessenta), 30 (trinta) e 10 (dez). Para cada uma das possibilidades foram realizadas 3 (três) baterias de testes.

### 2.5. MongoDB

O MongoDB é um banco de dados de documentos, ou seja, esta ferramenta tem como característica armazenar dados em documentos do tipo JSON. Esta nova forma de se lidar com os dados é muito expressiva e poderosa, sendo um grande concorrente aos bancos de dados relacionais [MongoDB 2020].

### 2.6. Ambiente de Desenvolvimento

Os testes foram realizados em um notebook da marca Samsung com a seguinte configuração: processador i5 – 5º geração (2.20GHz), SSD 240 Gb, 8 Gb de memória RAM, Sistema Operacional Windows 10 Pro com internet de 250 Mb.

Como IDE (Ambiente de Desenvolvimento Integrado) foi utilizado o IntelliJ Community Edition para o desenvolvimento da aplicação em Spring WebFlux. Já para a API desenvolvida em Node.js foi utilizado o editor de texto Visual Studio Code.

## 3. Resultados e Discussão

Embora será apresentado posteriormente a média de cada uma das 3 (três) baterias de testes para cada um dos 3 (três) tempos estabelecidos, a fim de não se exibir todas as imagens geradas, devido ao fato de apresentarem resultados muito parecidos, as baterias serão representadas através de um único gráfico para cada um dos ensaios propostos.

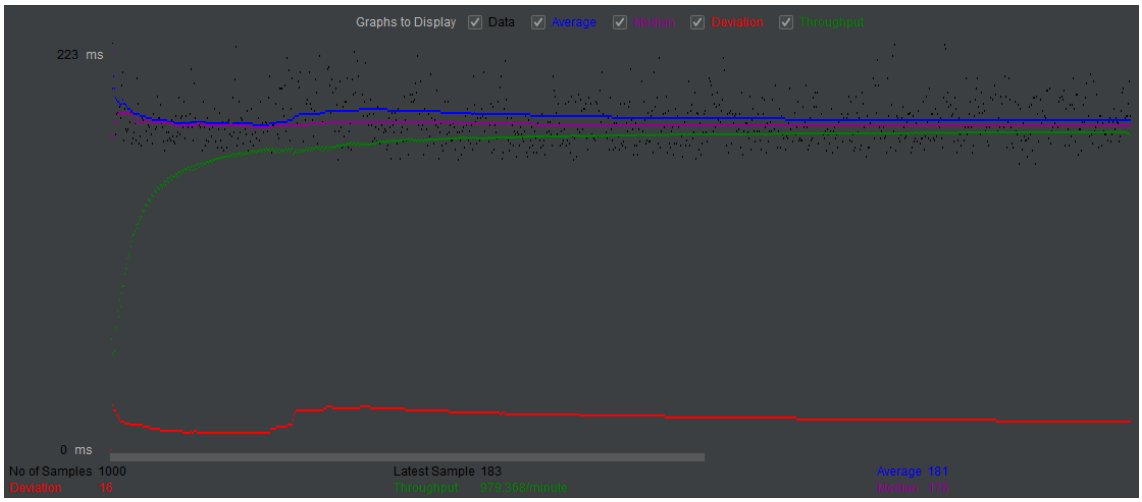


**Figura 5. API Node – 100 threads, 10 loops, 60 segundos**

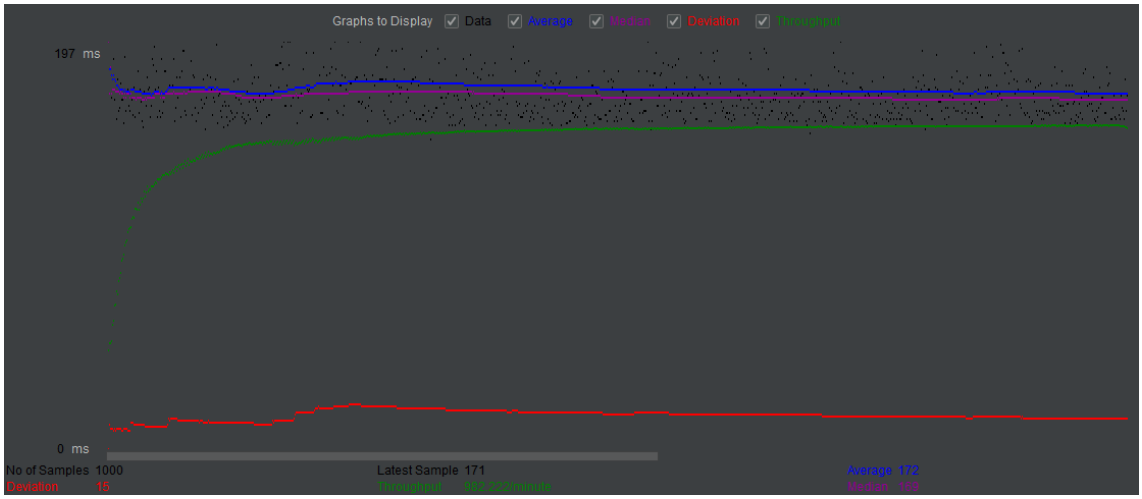


**Figura 6. API WebFlux – 100 threads, 10 loops, 60 segundos**

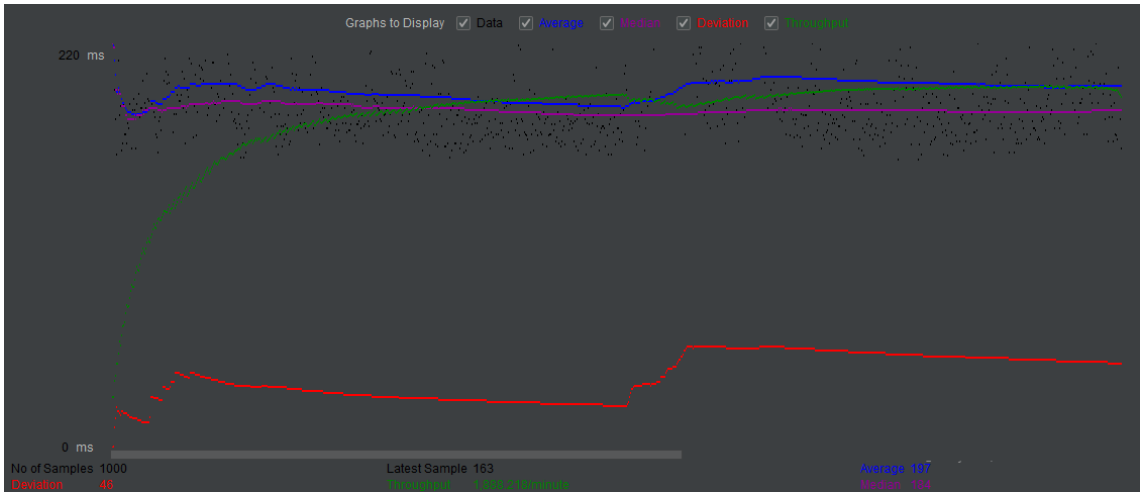


**Figura 7. API Node – 100 threads, 10 loops, 30 segundos**

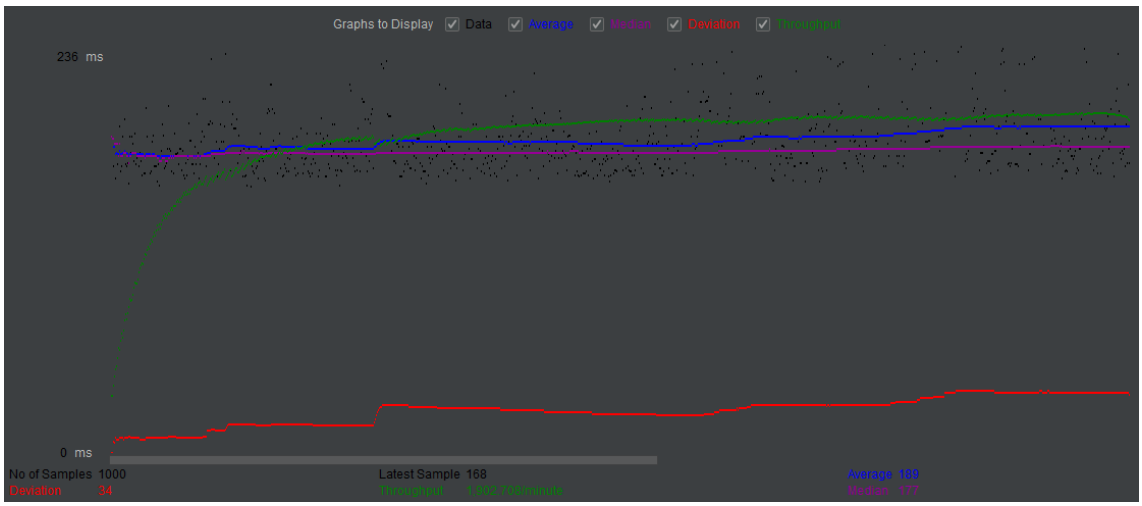


**Figura 8. API WebFlux – 100 threads, 10 loops, 30 segundos**

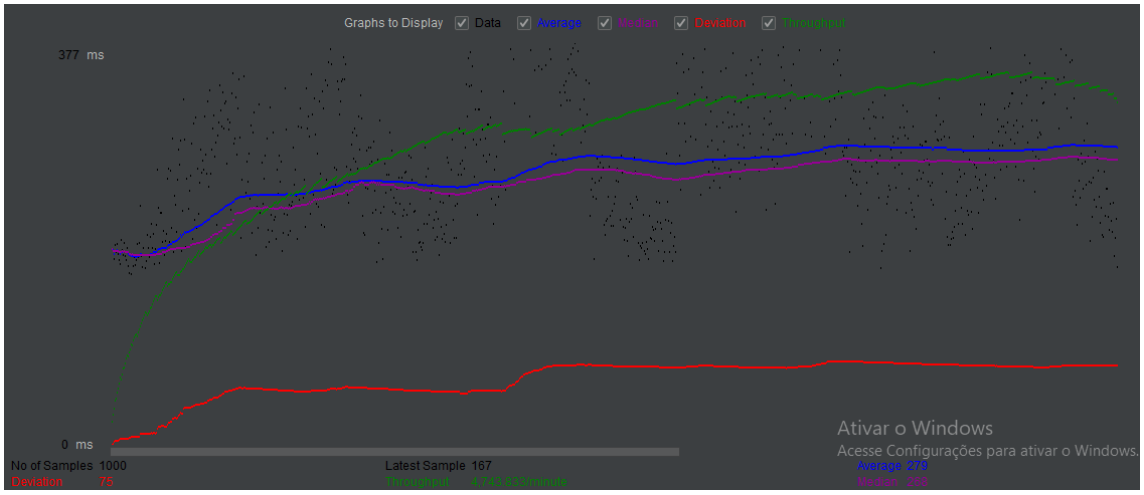


**Figura 9. API Node – 100 threads, 10 loops, 10 segundos**

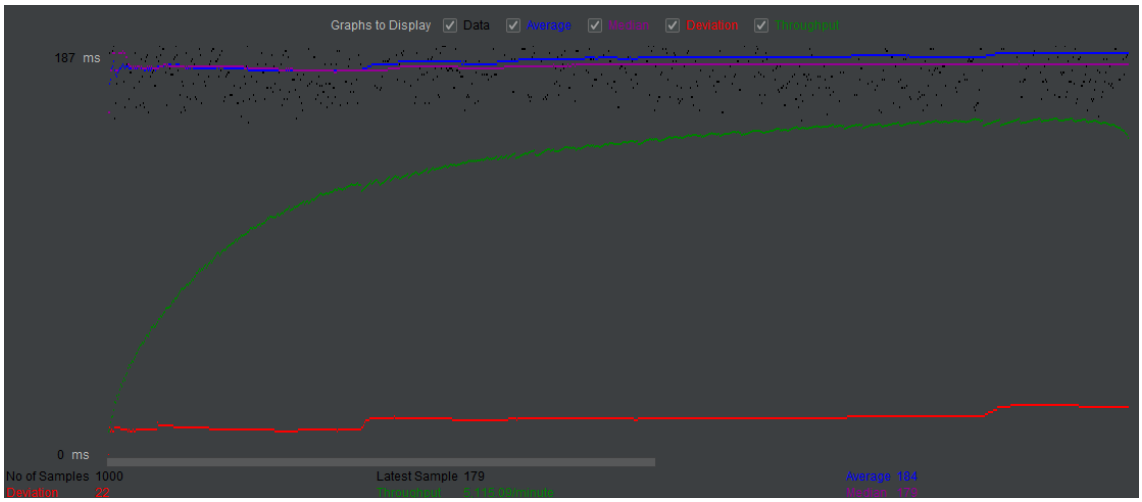


**Figura 10. API WebFlux – 100 threads, 10 loops, 10 segundos**

No rodapé das imagens pode-se observar informações referentes ao teste de carga. Os parâmetros na cor preta, "No of Samples" e "Latest Sample", fazem referência ao número de requisições feitas e a última amostra que foi processada, respectivamente. O parâmetro em azul ("Average") se refere a média de requisições efetuadas diante do tempo. O parâmetro em vermelho ("Deviation") representa o desvio padrão. E por fim, o de cor verde ("Throughput") refere-se ao número de solicitações por minuto tratadas pelo servidor.

Para análise do desempenho do servidor web em teste de carga, o foco deve ser concentrado em 2 (dois) parâmetros específicos: deviation e throughput, sendo este último, o mais importante. O throughput representa a capacidade do servidor em lidar com uma carga alta de requisições. Quanto maior for o throughput, melhor será o desempenho do servidor. Já o deviation ou desvio padrão, indica que quanto menor seu valor, melhor é o desempenho.

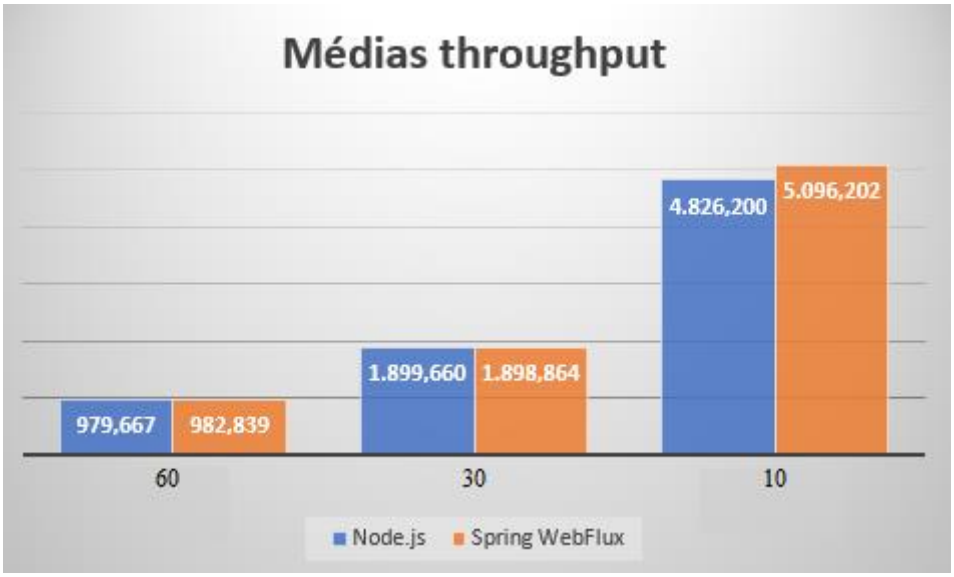


**Figura 11. Médias do parâmetro throughput para cada uma das baterias de teste**

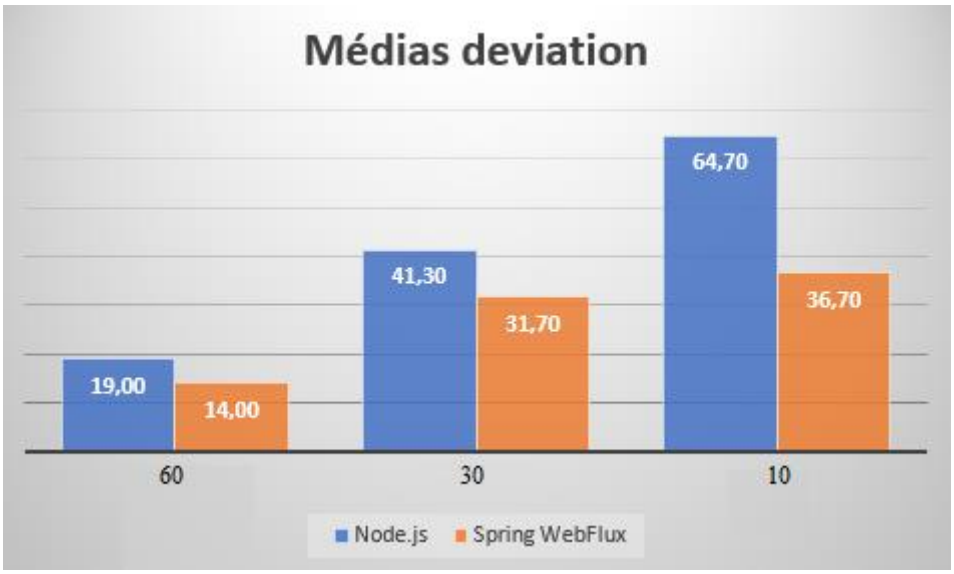


**Figura 12. Médias do parâmetro deviation para cada uma das baterias de teste**

Com base nos gráficos exibidos, é possível dizer que para os ensaios com o tempo parametrizado em 60 (sessenta) e 30 (trinta) segundos, o throughput praticamente foi idêntico entre as APIs, tendo uma média de 979,667/minuto na API Node e 982,839/minuto na API WebFlux para o tempo de 60 segundos, e 1.899,660/minuto na API Node e 1.898,864/minuto na API WebFlux para o tempo de 30 segundos. Já o desvio padrão, é visivelmente maior para a API Node em ambos os cenários, o que neste caso, indica um possível melhor desempenho por parte da API WebFlux.

Já os resultados apurados nos testes com o tempo de 10 (dez) segundos, demonstraram uma tendência no aumento da diferença de throughput entre as APIs conforme a quantidade de requisições também aumenta. Houve uma diferença significativa neste parâmetro, com uma média de 4.826,200/minuto na API Node e

5.096,202/minuto para a API WebFlux. Além disso, a diferença do desvio padrão entre as APIs foi ainda maior, com uma média de 64,70 para a API Node e 36,70 para a API WebFlux. Ou seja, ambos os cenários foram favoráveis a API WebFlux com o aumento das requisições.

## 4. Conclusão

Antes da conclusão com base nos resultados encontrados, é valido lembrar que o desempenho não é a única consideração que deve ser feita ao se analisar uma tecnologia relacionada ao desenvolvimento de aplicações. Também deve ser levado em conta uma arquitetura bem projetada, bem como o contexto em que a aplicação irá se encontrar. Ou seja, não existem verdades absolutas e vários outros aspectos devem ser considerados em relação a este assunto.

Agora, focando nos resultados nos quais este trabalho teve como objetivo analisar, temos: o Spring evoluiu bastante com a implementação do WebFlux, sendo que o paradigma de programação reativa conseguiu equiparar e até mesmo superar o Node em alguns cenários, além de se mostrar mais estável durante uma alta carga de requisições. Conclui-se que quanto maior a demanda do servidor e consequentemente maior o número de requisições simultâneas, mais eficiente é o Spring WebFlux em relação ao Node.js.

## Referências

BRASIL, André. “O que é o Node.js e quais são as suas características”. KINGHOST, 2019. Disponível em: <https://king.host/wiki/artigo/o-que-e-o-node-js-e-quais-sao-as-suas-caracteristicas/>. Acesso em: 23 de nov. de 2020.

BRITO, Michelli. “SPRING BOOT: DA API REST AOS MICROSERVICES”. 1º Edição, 2020.

BRITO, Michelli. “Spring Webflux”. Medium, 2019. Disponível em: <https://medium.com/@michellibrito/spring-webflux-f611c8256c53>. Acesso em: 21 de nov. de 2020.

CANALTECH. “O que é API?”, 2020. Disponível em: <https://canaltech.com.br/software/o-que-e-api/#:~:text=API%20%C3%A9%20um%20conjunto%20de,Interface%20de%20Programa%C3%A7%C3%A3o%20de%20Aplicativos%22>. Acesso em: 23 de nov. De 2020.

GOMES, Handerson Ferreira. “Artigo Java Magazine 11 – Teste de carga com JMeter”. DEVMEDIA, 2008. Disponível em: <https://www.devmedia.com.br/artigo-java-magazine-11-testes-de-carga-com-jmeter/9087>. Acesso em: 26 de nov. de 2020.

GURU99. “How To Use JMeter for Performance & Load Testing”, 2020. Disponível em: <https://www.guru99.com/jmeter-performance-testing.html>. Acesso em: 23 de nov. de 2020.

LUIZ, Andrey. “JavaScript #1 – Uma breve história da linguagem”. Ship it!, 2016. Disponível em: <http://shipit.resultadosdigitais.com.br/blog/javascript-1-uma-breve-historia-da-linguagem/>. Acesso em: 20 de nov. de 2020.

MDN WEB DOCS. “JavaScript”, c2020. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em: 20 de nov. de 2020.

MONGODB. “The database for modern applications”, 2020. Disponível em: <https://www.mongodb.com/>. Acesso em: 27 de nov. de 2020.

OKUNEU, Barys. “Java vs Nodejs”. Belitsoft, 2018. Disponível em: <https://belitsoft.com/java-development-services/java-vs-nodejs>. Acesso em: 22 de nov. de 2020.

SPRING. “Spring Framework Documentation”, 2020. Disponível em: <https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html>. Acesso em: 21 de nov. de 2020.

THE APACHE SOFTWARE FUNDATION. “Apache JMeter”, 2020. Disponível em: <https://jmeter.apache.org/>. Acesso em: 24 de nov. de 2020.

WAYNER, Peter. “Node.js vs. Java: An epic battle for developer mindshare”, 2019. Disponível em: <https://www.infoworld.com/article/2883328/nodejs-vs-java-an-epic-battle-for-developer-mindshare.html>. Acesso em: 20 de nov. de 2020.