AVALIAÇÃO COMPARATIVA DE DESEMPENHO ENTRE AS TECNOLOGIAS *NODEJS* E *SPRING BOOT*

RAFAEL LUÍS SAVENHAGO[1], RICARDO RAMOS DE OLIVEIRA[2], RICARDO FERREIRA VILELA[3], VICTOR HUGO SANTIAGO COSTA PINTO[4], ROBERTO NAVES UNGARELLI[5]

[1] Graduando em Engenharia de Computação, IFSULDEMINAS, Câmpus Poços de Caldas, rafael.savenhago@gmail.com
[2] Professor EBTT, IFSULDEMINAS, Câmpus Poços de Caldas, ricardo.ramos@ifsuldeminas.edu.br
[3] Doutorando em Ciência da Computação, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo (ICMC-USP), São Carlos – SP, ricardovilela@usp.br
[4] Pesquisador, Zup Innovation, São Paulo – SP, victor.pinto@zup.com.br
[5] Empresário, Oceano Azul TI, Brasília – DF, roberto.ungarelli@oceanoazulti.com
Área de conhecimento (Tabela CNPq): 1.03.03.04-9 Sistemas de Informação

RESUMO: Recentemente, com a popularização de arquiteturas de serviços *web* e *Cloud Computing*, diversas empresas estão migrando suas principais soluções e aplicações para esse tipo de arquitetura. Essa mudança de paradigma fez com que os requisitos desses sistemas se concentrassem no desempenho (Tempo de resposta, vazão, disponibilidade, entre outros) e no desenvolvimento no lado do servidor para atender a uma maior demanda de usuários sem que a experiência dos mesmos seja afetada de forma perceptível. Essa nova visão incentivou o surgimento e melhoria de diversas plataformas e frameworks de desenvolvimento para esse tipo de aplicação, a maioria desses com foco na melhoria de desempenho, e facilidade de desenvolvimento e manutenção. Nesse artigo, é feita a comparação desses requisitos (principalmente de desempenho) através de duas baterias de testes em um sistema de duas telas desenvolvidos usando NodeJS e Spring Boot, simulando o comportamento das aplicações em momentos de carga. No final, os gráficos obtidos utilizando as ferramentas JMeter e Dynatrace foram comparados e foi possível ver que as diferenças são baixas, com apenas um pequeno *trade-off* entre desempenho e recursos, e portanto a escolha da tecnologia a ser utilizada depende em grande parte da consolidação da mesma no mercado, e/ou de seu potencial futuro.

PALAVRAS-CHAVE: Dynatrace; Testes; JMeter; Framework; *REST*.

PERFORMANCE COMPARISON BETWEEN NODEJS AND SPRING *BOOT*

ABSTRACT: Recently, with the popularization of web and *Cloud Computing* service architectures, several companies are migrating their main solutions and applications to this type of architecture. This paradigm shift meant that the requirements of these systems were focused on performance (response time, throughput, availability, among others) and on server-side development to meet greater demand from users without affecting their experience. noticeably. This new vision encouraged the emergence and improvement of several development platforms and frameworks for this type of application, most of them with a focus on improving performance, and ease of development and maintenance. In this article, the comparison of these requirements (mainly of performance) is made through two batteries of tests in a system of two screens developed using *NodeJS* and *Spring Boot*, simulating the behavior of the applications in moments of a load. In the end, the graphs obtained using the *JMeter* and *Dynatrace* tools were compared and it was possible to see that the differences are low, with only a small trade-off between Performance and Resources, and therefore the choice of technology to be used depends largely on the consolidation of the market, and/or its future potential.

KEYWORDS: Dynatrace; Tests; Jmeter; Framework; REST.

INTRODUÇÃO

Ao mesmo tempo em que ocorre o avanço das tecnologias de serviços *web*, novos obstáculos do ponto de vista de qualidade de software são manifestados. Um dos principais desafios recorrentes nesse sentido é o desempenho dessas aplicações. À medida que os serviços se tornam mais populares e um maior número de pessoas passa a usufruir de seus recursos de forma simultânea, o desempenho é fortemente impactado, e muitas vezes, a qualidade do serviço também.

Para minimizar a queda na qualidade dos serviços, as empresas provedoras costumam seguir por dois caminhos, melhorar a infraestrutura (servidores), ou, desenvolver seus serviços utilizando tecnologias que apresentem melhor desempenho.

Apesar de existirem diversos trabalhos na literatura, a discussão permanece em aberto devido ao fato desse tipo de tecnologia estar em constante evolução, consequentemente existe maior complexidade para determinar um consenso sobre suas pontos positivos e negativos de cada tecnologia (Haro et al., 2019).

Considerando o contexto apresentado, o objetivo deste trabalho foi conduzir um estudo de caso sobre a comparação entre as tecnologias (NodeJS e Spring Boot). A condução dos testes de desempenho foi realizada para responder questões chave de pesquisa, sendo elas: **i)** Qual tecnologia apresenta maior performance; **ii)** Qual tecnologia demanda menor recurso computacional; **iii)** Qual a aderência no mercado; e **iv)** Qual a tendência tecnológica. Essas questões e demais aspectos são explorados ao longo do texto.

Em adição, o presente trabalho oferece uma visão sobre as tecnologias por meio do processo de teste e análise do comportamento em situações de sobrecarga de requisições. Além disso, pretende-se elucidar cenários onde o uso de cada *framework* é mais indicado.

FUNDAMENTAÇÃO TEÓRICA

Segundo Ihrig (2014), NodeJS é uma ferramenta (estrutura) de desenvolvimento assíncrona altamente escalável de aplicações JavaScript, escrita em C++ e JavaScript, tomando por base a *Engine V8-JavaScript* desenvolvida pela Google e utilizada pelo navegador Google Chrome. As principais características atribuídas ao NodeJS são seu uso de JavaScript como linguagem para desenvolvimento *back-end* (apesar de não ter sido a primeira implementação de tal conceito, acabou sendo a que mais se popularizou), e sua arquitetura não bloqueante (garante uma utilização mais eficiente dos recursos de máquina em comparação aos seus concorrentes).

Conforme Lecheta (2018) e Ihrig (2014) explicam, o NodeJS funciona com base em uma arquitetura assíncrona não bloqueante orientada à eventos, que trabalha de forma diferente de outras arquiteturas, tais como a arquitetura do Apache. Os servidores Apache primeiro tratam as requisições *HTTP* e deixam que a implementação da aplicação seja realizada em alguma linguagem. Já no NodeJS cada aplicação contém seu próprio servidor, ou seja, o servidor NodeJS é parte da própria aplicação.

Lecheta (2018) diz que, o Apache funciona criando vários grupos segmentos para tratar as requisições, ou seja, cria uma *thread* para tratar cada conexão e só finaliza a mesma quando a resposta do servidor é enviada. Essa arquitetura apresenta baixa escalabilidade, uma vez que fica muito dependente dos componentes de *hardware* para agilizar as operações de E/S (geralmente de alta latência).

Já o Spring Boot, segundo Gutierrez (2016), é uma extensão do *framework* Spring para *JEE*, que visa remover as complicadas configurações que são utilizadas em projetos Spring convencionais para que o programador possa focar no desenvolvimento da aplicação e deixar todas as dificuldades na mão do *framework*. Spring é um projeto *Open-Source* que através do incentivo ao uso de padrões de projeto (principalmente o padrão de Injeção de dependências, que permite que as classes tenham um baixo nível de acoplamento) facilita no desenvolvimento de aplicações *JEE* elegantes e confiáveis.

Reddy (2017) afirma que, Spring Boot traz consigo várias adições que visam facilitar a vida de um desenvolvedor que utiliza o *framework* Spring. Esse *framework*, apesar de ser robusto, requer certa carga de atividades para conseguir ser utilizado corretamente, fato este que torna seus desenvolvedores dependentes de ferramentas para auxiliá-los durante este processo. Gutierrez (2018) diz que, para reduzir esta perca de tempo, *Spring Boot* possui certas características, como por exemplo seu servidor *Tomcat* imbutido que reduz o tempo gasto configurando o servidor para uso e com *debugs*, ou o fato de não necessitar de configurações em *XML*. Outras características incluem: Simplicidade de configurar e usar

logs, mais formas de configurar eventos e *listeners*, permite o uso de perfis para auxiliar a aplicação a trabalhar em diferentes ambientes, entre inúmeras outras.

MATERIAL E MÉTODOS

A metodologia desse trabalho foi realizada conforme as etapas a seguir:

Os microserviços Item tela e Menu foram desenvolvidos nas tecnologias NodeJS e Spring Boot utilizando uma mesma aplicação.

As aplicações foram implantadas em *containers* Docker autodimensionáveis para evitar problemas com compatibilidade e escalonamento de recursos para os serviços. Desta forma, espera-se que os resultados sejam mais concisos, e que hajam menos ruídos (variância) nos dados coletados.

Foi escrito um *script* para avaliar o desempenho das aplicações com a ferramenta JMeter que realiza requisições por um tempo determinado, desta forma imitando o perfil dos usuários dos serviços, como definido no cenário. As requisições foram realizadas com um pequeno intervalo de tempo aleatório entre cada repetição para melhor simular o processo. O desempenho das tecnologias foi medido através da ferramenta JMeter que monitorou os serviços durante a execução do *script*.

Para conduzir o estudo de caso comparativo entre NodeJS e Spring Boot utilizou-se duas baterias de teste de desempenho para simular o comportamento entre dois perfis de usuários conforme o tempo entre suas ações (usuário ágil – 1 segundo e usuário lento – 3 segundos). Ambas as baterias de testes avaliaram sobrecarga nas respectivas aplicações. O procedimento de cada bateria é semelhante, variando apenas o tempo entre ações, sendo o número de requisições uma consequência do mesmo. A primeira bateria consta a realização do teste com o parâmetro *Think Time* referente à 3 segundos, enquanto a segunda bateria tem por parâmetro 1 segundo.

As baterias de teste foram configuradas da seguinte forma:

- Número de Usuários: 100
- *Ramp-up* (Tempo até o JMeter carregar todas as *threads*): 10 segundos
- *Think Time* (Tempo entre uma ação e outra): 1 à 3 segundos
- Duração: 15 minutos

Com base nestas configurações pode-se perceber que durante 15 minutos, 100 usuários virtuais irão realizar o maior número possível de requisições, com um espaço de tempo de 1 à 3 segundos entre cada requisição. Após os 15 minutos os resultados serão avaliados de forma a tentar generalizar o acontecimento durante altas cargas nos microsserviços.

A simulação consiste no seguinte cenário: um usuário acessa uma Tela de Login (*endpoint* Menu), aguarda por um período determinado para o carregamento da tela, realiza o login, e então é redirecionado para uma tela de Menu (*endpoint* Item Tela), que realiza operações de alta latência no servidor (consultas à banco de dados). Novamente existe um tempo pré-determinado para o carregamento das funcionalidades e posteriormente finaliza a conexão.

Em seguida, o *script* do JMeter foi executado juntamente com os serviços em *containers* Docker pelas mesmas razões citadas na implantação das aplicações sob testes nos containers Docker.

Os resultados foram análisados por meio dos gráficos gerados pelo agente Dynatrace implatado nos *containers* Docker.

O *script* do JMeter foi executado para cada uma das tecnologias (NodeJS e Spring Boot).

Por fim, os microserviços foram monitorados.

RESULTADOS E DISCUSSÃO

A Figura 1 ilustra os resultados obtidos. A partir das requisições realizadas em cada *endpoint,* percebe-se que o Spring Boot aparentemente é capaz de processar um pouco mais de requisições, embora a diferença entre as amostras, tanto para a tela de Menu, quanto para Item Tela, seja pequena. A primeira

bateria obteve cerca de 20 mil requisições durante a execução da simulação, enquanto a segunda bateria de teste obteve cerca de 25 mil requisições.

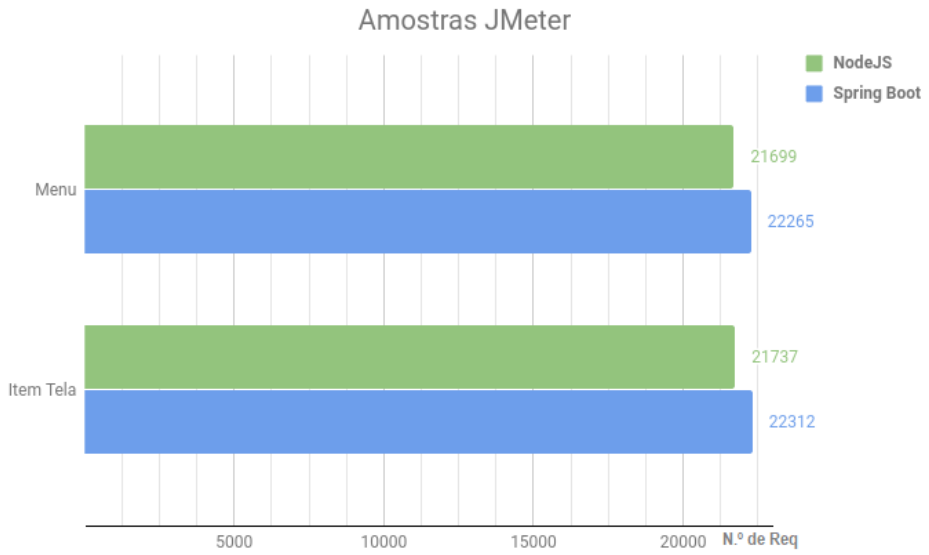

Figura 1 - JMeter- Requisições realizadas em cada endpoint (Média entre baterias).

Ao realizar as analíses dos resultados de uso de CPU, tempo de resposta, consumo de memória e *Throughput* (Quantidade de requisições por unidade de tempo) entre as tecnologias avaliadas, pode-se verificar que NodeJS obteve uma performance melhor no uso de CPU, tempo de resposta e consumo de memória. O consumo de CPU do NodeJS foi 42% do total utilizado pelo Spring Boot, conforme pode ser visto na Tabela 1, já o tempo de resposta e o consumo de memória do NodeJS foi de aproximadamente 50% e 71% respectivamento comparado ao total do Spring Boot, conforme os resultados da Figura 2. Já a vantagem do Spring Boot quanto ao *Throughput* foi somente de 2% comparado com o NodeJS, conforme ilustrado na Figura 2.

	NodeJS	Spring Boot
CPU (ms/req)	0,47	1,12

Tabela 1 – Uso de CPU por Tecnologia.

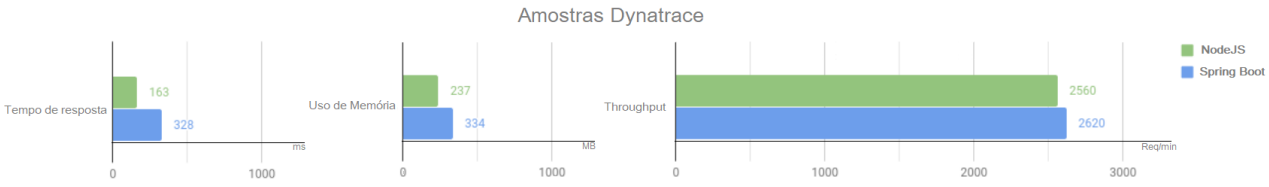

Figura 2 – Dynatrace – Dados de CPU, Memória e Throughput (Média entre baterias).

Atualmente entre os *frameworks* mais populares e utilizados no desenvolvimento de API's REST está o NodeJS, permitindo criar soluções tecnologicamente competitivas com a linguagem JavaScript (SOUSA, 2015).

O Spring Boot permite simplificar o processo de configuração e implantação de aplicações *web,* além de oferecer um servidor Tomcat embarcado, essas caracteristicas consolidaram o *framework* Java no mercado. Além disso, o Spring Boot está sendo utilizado para a criação de arquiteturas baseadas em microsserviços (GEEKHUNTER, 2019).

Por fim, o profissional com conhecimentos sólidos no ambiente de execução NodeJS e do *framework* Spring Boot estão alinhados com o mercado de trabalho, essas tecnologias são indispensáveis e amplamente utilizadas em diversas aplicações empresáriais (GEEKHUNTER, 2019) (SOUSA, 2015).

CONCLUSÕES

Com base nos resultados dos testes, nos quais a aplicação faz acesso à banco de dados, percebe-se que as diferenças de desempenho existem dentro do cenário proposto. Sendo assim, pode-se melhorar o desempenho de ambas as tecnologias utilizando-se de técnicas que não foram contempladas no estudo de caso, como *Cache* e *Tuning*.

O Spring Boot possui a vantagem de fazer uso da plataforma Java, que está bem inserida no mercado e conta com diversos *frameworks* e biliotecas (GEEKHUNTER, 2019). No entanto, juntamente com a arquitetura de microserviços, as tecnologias mais flexíveis começaram a se destacar, entre elas o NodeJS, como cita Lecheta (2018). Ao considerar o fato de que o NodeJS é mais recente e ainda não possui tanto suporte e aderência por parte da comunidade pode-se dizer que a expectativa para o mesmo é otimista. Por ser leve e multiplataforma esse é muito utilizado em projetos *Open-Source*, e por fazer uso de JavaScript, o mesmo se torna cada vez mais popular. Lecheta (2018) também diz que, atualmente essa é a linguagem que mais cresce entre os desenvolvedores.

Como via de sugestão, é interessante que os serviços categorizados como complexos sejam escritos com a utilização do *framework* Spring Boot e aqueles de baixa complexidade sejam escritos em *NodeJS* de modo que sejam aproveitados os benefícios que as duas tecnologias disponibilizam.

Portanto, conclui-se com esse estudo de caso que o NodeJS foi melhor em relação tempo de resposta, consumo de memória e uso de CPU, enquanto o Spring Boot foi ligeiramente melhor no *Throughput.*

REFERÊNCIAS

GEEKHUNTER. **Spring Boot: Tudo que você precisa saber!** 2019. Disponível em: https://blog.geekhunter.com.br/tudo-o-que-voce-precisa-saber-sobre-o-spring-boot/. Acesso em: 20 out. 2020.

GUTIERREZ, F. (2018). **Spring boot internals and features.** InPro Spring Boot 2, pages 45–85. Apress.

HARO, E., Guarda, T., PEÑAHERRERA, Alex, O. Z., and QUIÑA, G. N. (2019). **Desarrollo backend para aplicaciones web, servicios web restful: Node.js vs spring boot.**Revista Ibérica de Sistemas e Tecnologias de Informação, pages 309–321. Nome - YahooInc; Facebook Inc; Google Inc; Copyright © 2019. Última atualizaçã em - 2019-08-19.

IHRIG, C. J. (2014).**Pro node.js para desenvolvedores.** Ciência Moderna.

LECHETA, R. R. (2018).**Node essencial.** Novatec.

REDDY, K. S. P. (2017). **Spring boot autoconfiguration.** In Beginning Spring Boot 2, pages 35–45. Apress.

SOUSA, Filipe Perdigão de. **Criação de framework REST/HATEOAS Open Source para desenvolvimento de APIs em Node.js**. 2015. 105 f. Dissertação (Mestrado) - Curso de Mestrado Integrado em Engenharia Informática e Computação, Faculdade de Engenharia da Universidade do Porto, Porto - Portugal, 2015.