INSTITUTO FEDERAL
Pernambuco

UM ESTUDO COMPARATIVO ENTRE AS TECNOLOGIAS *SPRING BOOT* E *QUARKUS* NA IMPLEMENTAÇÃO DO *BACK-END* DE APLICAÇÕES *WEB* COM *MONGODB*

Matheus Albuquerque Montenegro
mam5@discente.ifpe.edu.br
Francisco do Nascimento Júnior
francisco.junior@jabaotao.ifpe.edu.br

RESUMO

O objetivo deste artigo é comparar o desempenho de duas tecnologias significantes para o desenvolvimento de aplicações *Java*: *Quarkus* e *Spring Boot*. Para isso, foram construídas duas aplicações com o mesmo escopo. Uma foi implementada no *Quarkus* e a outra no *Spring Boot*. Ambas aplicações acessam um banco de dados *MongoDB* armazenado em um contêiner *Docker* e atuam como uma *API* para interagir com o banco de dados usando o cliente *MongoClient*. O *MongoDB* é um banco de dados não relacional orientado a documentos amplamente utilizado em aplicações *web*, devido à sua alta escalabilidade e flexibilidade. Para avaliar o desempenho das duas aplicações, foram executados testes de carga utilizando *Jmeter*, para simular o comportamento do sistema com diferentes solicitações e tráfego de dados. Além dos testes de carga realizados para avaliar o desempenho da aplicação, também foram realizados testes unitários usando *JUnit* 5 e *Mockito*, duas das ferramentas essenciais para testes em *Java*. Os testes unitários destinam-se a testar a funcionalidade isoladamente para garantir a qualidade e a confiabilidade do código. Após a realização dos testes de carga e comparadas as métricas extraídas, foi constatado que o *Spring Boot* possui uma performance melhor que o *Quarkus*, em cenários específicos a diferença é mais substancial. Portanto, dentro do contexto proposto para esse trabalho, o *Spring Boot* demonstrou ter uma performance melhor que o *Quarkus*.

Palavras-chave: *Java*. *Quarkus*. *Spring Boot*. *MongoDB*. Tempo Médio de Resposta. Testes de Carga.

ABSTRACT

The objective of this article is to compare the performance of two significant technologies for Java application development: Quarkus and Spring Boot. To achieve this, two applications with the same scope were built. One was implemented in Quarkus and the other in Spring Boot. Both applications access a MongoDB database stored in a Docker container and act as an API to interact with the database using the MongoClient client. MongoDB is a widely used document-oriented NoSQL database in web applications due to its high scalability and flexibility. To evaluate the performance of the two applications, load tests were executed using JMeter to simulate the system behavior with different requests and data traffic. In addition to the load tests performed to assess the application's performance, unit tests were also conducted using JUnit 5 and Mockito, two essential testing tools in Java. Unit tests aim to test the functionality in isolation to ensure code quality and reliability. After conducting the load tests and comparing the extracted metrics, it was found that Spring Boot performs better than Quarkus, and the difference is more substantial in specific scenarios. Therefore, within the context proposed for this work, Spring Boot demonstrated better performance than Quarkus.

Keywords: *Java*. *Quarkus*. *Spring Boot*. *MongoDB*. Average Response Time. Load Tests.

1 INTRODUÇÃO

No desenvolvimento de aplicações *web* modernas, há diversas decisões a serem tomadas visando o sucesso do projeto. Dentre as principais escolhas, está a de qual *framework* será utilizado na aplicação, de acordo com Gamma et.al (2007, Gamma, p.41): “Um *framework* é um conjunto de classes cooperantes que constroem um projeto reutilizável para uma determinada categoria de *software*”. Sendo assim o *framework* é uma peça fundamental para tornar o processo de desenvolvimento mais ágil. Dentre os *frameworks* disponíveis do *Java*, o *Spring* e o *Quarkus* se destacam pelas suas robustez e confiabilidade. No entanto, um fator importante que pode determinar qual dos dois será escolhido para ser utilizado em um projeto, é a performance. Uma vez que muitas aplicações terão que lidar com muitas requisições simultâneas e ainda assim retornar os resultados com o melhor tempo possível, sempre visando a melhor experiência para o usuário.

Este trabalho tem como objetivo principal fornecer informações úteis para a escolha da tecnologia ideal para a implementação de aplicações web de alto desempenho que utilizam *Java* e necessitam acessar um banco de dados *MongoDB*. Segundo o próprio site oficial do *MongoDB*: “O *MongoDB* é um banco de dados de documentos projetado para facilitar o desenvolvimento de aplicativos e a escalabilidade” (*MONGODB*,2023).

Para amparar o objetivo principal do trabalho, foram delimitados quatro

objetivos específicos:

• Determinar qual tecnologia possui a melhor performance diante de uma quantidade elevada de requisições concorrentes

• Determinar qual tecnologia possui a melhor performance diante de uma quantidade baixa de requisições concorrentes

• Determinar qual tecnologia possui a melhor performance transacionando arquivos que ocupam uma capacidade considerável de armazenamento

• Determinar qual tecnologia possui a melhor performance transacionando arquivos que ocupam pouco espaço de armazenamento

A escolha do *MongoDB* como banco de dados para a aplicação web, se deu por seu uso em aplicações modernas que demandam flexibilidade e uma performance superior a banco de dados relacionais na maioria dos cenários.

Este trabalho visa fornecer informações comparando o desempenho do *Quarkus* e do *Spring Boot* em uma aplicação que acessa um banco de dados *MongoDB*, utilizando o tempo médio de resposta como métrica de desempenho e testes de carga utilizando o *JMeter* que conforme o *site* oficial do *Jmeter*: “A aplicação *Apache JMeter* é um *software* de código aberto, uma aplicação 100% pura *Java* projetada para testar o comportamento funcional e medir o desempenho” (*JMETER*,2023). Além disso, os resultados deste trabalho ajudarão os desenvolvedores a escolher a melhor tecnologia para aplicações web *Java* que precisam de acesso a bancos de dados não relacionais orientados a documentos.

Este artigo está organizado da seguinte forma: A Seção 2 discorre sobre as tecnologias que são objetos de estudo deste trabalho. A Seção 3 descreve qual metodologia será aplicada. A Seção 4 descreve como ocorreu o desenvolvimento e execução das três primeiras etapas definidas na metodologia. A Seção 5 apresenta os resultados dos testes com as tecnologias e uma análise das informações obtidas. A seção final contém reflexões finais e conclusões.

1.1 Trabalhos Relacionados

Em sua tese, Matheus Santos de Almeida analisou o desempenho do *Spring Boot* e do *Quarkus* utilizando a *GraalVM*, chegando a conclusão que o *Quarkus* possui melhor desempenho nesse contexto, mas evidenciando que o *Spring Boot* não tem suporte à *GraalVM*, o que pode ter influenciado nos resultados. (ALMEIDA,2020).

Outro trabalho relevante é o de Assis e de Bavaresco, que busca comparar os desempenhos entre o *Spring Boot*, *Quarkus* e *Micronaut* no contexto de microsserviços que acessam um banco de dados relacional, chegando a conclusão que o *Spring Boot* possui o melhor desempenho. (ASSIS e BAVARESCO, 2022).

2 MATERIAIS E MÉTODOS

Esta seção tem como objetivo elucidar sobre as tecnologias que terão suas performances comparadas neste trabalho.

2.1 *Spring Boot*

Para Walls:

> "O *Spring* foi criado para lidar com a complexidade do desenvolvimento de aplicações empresariais e torna possível usar *Java*Beans simples para realizar coisas que anteriormente eram possíveis apenas com *EJB(Enterprise Java Beans)*. Mas a utilidade do *Spring* não se limita ao desenvolvimento do lado do servidor. Qualquer aplicação *Java* pode se beneficiar do *Spring* em termos de simplicidade, testabilidade e baixo acoplamento" (2014, WALLS, p.4).

Além dessas características, o gerenciamento de dependência, configuração de ambiente mais simples e a utilização de padrões de inversão de controle e injeção de dependências, tornaram o *Spring* um dos *frameworks* mais utilizados para a implementação de aplicações web que usam *Java* como sua linguagem de programação.

O *Spring Boot* por sua vez facilita a criação de aplicações no *Spring*, de acordo com o próprio *site* oficial do projeto *Spring*: "O *Spring Boot* torna fácil criar Aplicações baseadas em Spring autônomas e prontas para produção que você pode simplesmente executar"(SPRING BOOT, 2023).

2.2 *Quarkus*

De acordo com o site oficial do *Quarkus*: "O *Quarkus* foi criado para permitir que os desenvolvedores *Java* criem aplicações para um mundo moderno e nativo da nuvem. O *Quarkus* é um *framework Java* nativo do *Kubernetes* feito sob medida para o *GraalVM* e o *HotSpot*, criado a partir das melhores bibliotecas e padrões *Java*. O objetivo é tornar o *Java* a plataforma líder em ambientes *Kubernetes* e *serverless*, oferecendo aos desenvolvedores um *framework* para abordar uma ampla gama de arquiteturas de aplicativos distribuídos" (*QUARKUS*,2023).

O *Quarkus* foi concebido visando a implementação de aplicações modernas, para isso oferece diversos recursos comuns no desenvolvimento de aplicações *Java*, como a inversão de controle e injeção de dependências, além disso tem

compatibilidade com diversas tecnologias modernas como por exemplo o *Kubernetes*. Segundo o *site* oficial do *Kubernetes*: "*Kubernetes* é um produto Open Source utilizado para automatizar a implantação, o dimensionamento e o gerenciamento de aplicativos em contêiner" (KUBERNETES,2023).

3 METODOLOGIA

Este tópico descreve a metodologia aplicada para realizar a comparação de desempenho entre as tecnologias *Spring Boot* e *Quarkus* na implementação de aplicações web que utilizem o *MongoDB* como seu banco de dados.

3.1 Questões da pesquisa

Foram selecionadas quatro questões buscando retratar o desempenho do *Spring Boot* e do *Quarkus* baseado em variados cenários possíveis que uma aplicação web do lado do servidor pode estar exposta. Logo, a questão de pesquisa (**QP**) geral que estas análises visam investigar é:

QP: Dentre as tecnologias *Spring Boot* e *Quarkus*, qual apresenta o melhor desempenho?

Para auxiliar na resposta dessa pergunta geral, foram desenvolvidas outras quatro questões de pesquisa (**QP**) específicas:

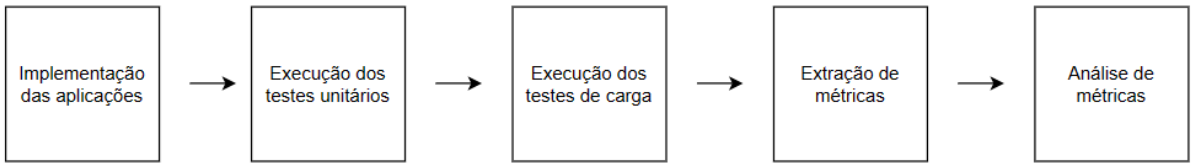
- **QP1:** Qual tecnologia possui melhor a performance para uma alta quantidade de requisições realizadas por usuários concorrentes?
- **QP2:** Qual tecnologia possui melhor a performance para uma quantidade baixa de requisições realizadas por usuários concorrentes?
- **QP3:** Qual tecnologia possui a melhor performance para transacionar arquivos que possuem 500 *Megabytes*?
- **QP4:** Qual tecnologia possui a melhor performance para transacionar arquivos que possuem 5 *Kilobytes*?

Dentro do escopo deste estudo, iremos considerar que um arquivo de 500 *Megabytes* é grande, devido à sua capacidade significativa de armazenamento e de transferência de dados, o que pode afetar diretamente a eficiência de muitos cenários. Por outro lado, iremos considerar um arquivo de 5 Kilobytes como pequeno, devido à sua baixa capacidade de armazenamento e de fácil transacionamento desse volume de dados. Essa distinção é importante para definirmos a avaliação de desempenho entre os cenários da **QP3** e **QP4**.

No intuito de realizar a comparação de desempenho de forma a ajudar a responder as questões analisadas neste trabalho, as seguintes etapas foram

seguidas:

Figura 1 - Fluxograma de etapas propostas na metodologia



Fonte: Elaborado pelo próprio autor.

3.1.1 Implementação das aplicações

Será desenvolvida uma aplicação utilizando *Spring Boot* e outra utilizando *Quarkus*, implementando o mesmo conjunto de requisitos, dentro do escopo de um sistema de gestão de recursos humanos, desse modo restringindo as diferenças entre as aplicações somente às peculiaridades de cada *framework*. Os seguintes *endpoints* foram criados nas aplicações:

Tabela 1 – Lista de *endpoints* criados em ambas aplicações

Endpoints
Retorno de lista de todos os funcionários
Cadastro de Funcionários
Marcação de ponto
Envio de atestado médico

Fonte: Elaborado pelo próprio autor.

3.1.2 Execução dos testes unitários

Serão desenvolvidos testes unitários com a finalidade de demonstrar a qualidade e confiabilidade do código-fonte das aplicações. Sendo possível verificar através dos testes e de suas coberturas que as aplicações estão funcionando conforme as regras de negócio estabelecidas.

3.1.3 Execução dos testes de carga

Scripts de testes de carga serão desenvolvidos com a finalidade de simular possíveis situações que uma aplicação sofreria em um ambiente real e com um número alto de requisições simultâneas. A aplicação e os *scripts* dos testes de cargas serão executados para ambas tecnologias no mesmo ambiente de teste, possuindo as mesmas configurações de hardware e sistema operacional.

3.1.4 Extração de métricas

Após a execução dos testes de carga, será possível extrair as seguintes métricas que auxiliarão a responder as questões deste trabalho:

• **Tempo médio de resposta**: A soma de todos tempos de respostas dividido pela quantidade de requisições realizadas em milissegundos.

• **Tempo mínimo de resposta**: Dentre todas as requisições realizadas, qual durou a menor quantidade de tempo até a resposta em milissegundos.

• **Tempo máximo de resposta**: Dentre todas as requisições realizadas, qual durou a maior quantidade de tempo até a resposta em milissegundos.

• **Taxa média de transferência de dados**: A soma de todas as taxas de transferência medidas em KB/segundos dividido pela quantidade de requisições

3.1.5 Análise de métricas

Com os dados obtidos da extração de métricas, uma análise comparativa será feita, para determinar qual *framework* se saiu melhor em cada cenário analisado.

4 DESENVOLVIMENTO

Nesta seção está a descrição das três primeiras etapas que estão definidas no capítulo de Metodologia, demonstrando em detalhes como foi executada cada uma dessas etapas.

4.1 Implementação das aplicações

Na execução dessa etapa foi realizada a implementação de duas aplicações que possuem o mesmo escopo, sendo uma implementação realizada em *Quarkus* na versão 2.2.3 e a outra em *Spring Boot* na versão 3.0.5. Ambas aplicações estão utilizando a *Open JDK* na versão 17.0.3 para suas execuções e utilizando o banco de dados *MongoDB* na versão 6.0.3.

O *MongoClient* foi a ferramenta escolhida para realizar as interações entre as aplicações e o banco de dados *MongoDB* por ser possível utilizar os mesmos algoritmos para implementar tais interações tanto no *Spring Boot* quanto *Quarkus*. Para armazenar os arquivos enviados em alguns dos *endpoints* criados nas duas aplicações, foi utilizada o *GridFS* que é uma especificação do *MongoDB* que torna possível armazenar e retornar arquivos que excedam 16 *Megabytes* de tamanho.

Ambas aplicações funcionam como uma *API (Application Program Interface)* e possuem o escopo de um sistema básico de gerenciamento de recursos humanos, onde é possível realizar a inserção, consulta, atualização e deleção de dados de um funcionário, além de realizar a marcação de ponto e o envio de atestados médicos.

Foram criados métodos *HTTP* para que os usuários pudessem interagir com as aplicações, sendo dois métodos *GET* dos quais um retorna a consulta de todos os funcionários existentes no banco de dados e o outro retorna um funcionário específico baseado no seu código enviado na requisição, três métodos *POST* para a realização de inserção de dados no banco de dados, dos quais um está inserindo dados para o cadastro de um funcionário, o segundo tem o objetivo de realizar a marcação de ponto de um funcionário e o terceiro que realiza o envio de atestados médicos, além de um método *PUT* para a atualização dos dados de um funcionário e do *DELETE* para a deleção dos dados de um funcionário. Para cada método *HTTP* foi criado um respectivo *endpoint*:

- *GET* /rh/*api*/*employee* (*Endpoint* que retorna a lista de todos os funcionários)
- *POST* /rh/*api*/*employee* (*Endpoint* para cadastrar funcionários)
- *POST* /rh/*api*/*clocking* (*Endpoint* responsável pela marcação de ponto)
- *POST* rh/*api*/*sickNote* (*Endpoint* que salva os atestados médicos no banco de dados)

As implementações dos códigos podem ser encontradas em repositórios no *GitHub*. A implementação do *Spring Boot* está disponível no seguinte *link*: https://github.com/matheusmontenegro97/rh-admin-spring. Por outro lado, a implementação do *Quarkus* pode ser acessada no *link*: https://github.com/matheusmontenegro97/rh-admin-quarkus. Sendo possível verificar o código-fonte de ambas aplicações e comparar as nuances de cada implementação.

As aplicações seguiram uma estrutura de classes separadas por *Codecs*, *Controller*, *Exceptions*, *Model* e *Repository*. Cada estrutura comporta classes com finalidades específicas.

4.1.1 *Codecs*

No escopo da estrutura dos *Codecs* estão as classes responsáveis por realizar a conversão dos dados dispostos nas classes *Java* em documentos *BSON (Binary JSON)* que é o formato utilizado pelo *MongoDB*. Também realizando a conversão de *BSON* para uma classe *Java*.

4.1.2 *Controller*

Na estrutura de *Controller* estão todas as classes que recebem as requisições *HTTP* e tem a função de realizar as requisições necessárias para outras classes na aplicação que tratarão os dados conforme as regras de negócios e retornará as

respostas para a requisição *HTTP* realizada.

4.1.3. *Exceptions*

A estrutura de *Exceptions* contempla todas as classes responsáveis por customizar exceções que serão lançadas quando algo de não esperado ocorrer durante o fluxo da aplicação.

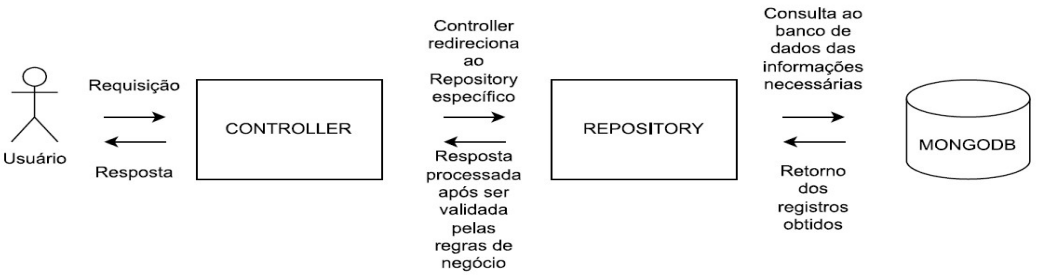
4.1.4 *Model*

No escopo da estrutura *Model* estão as classes que contém os *model*os dos objetos utilizados pela aplicação. Responsável também por gerenciar os elementos dos dados, utilizando as melhores práticas de orientação a objetos.

4.1.5 *Repository*

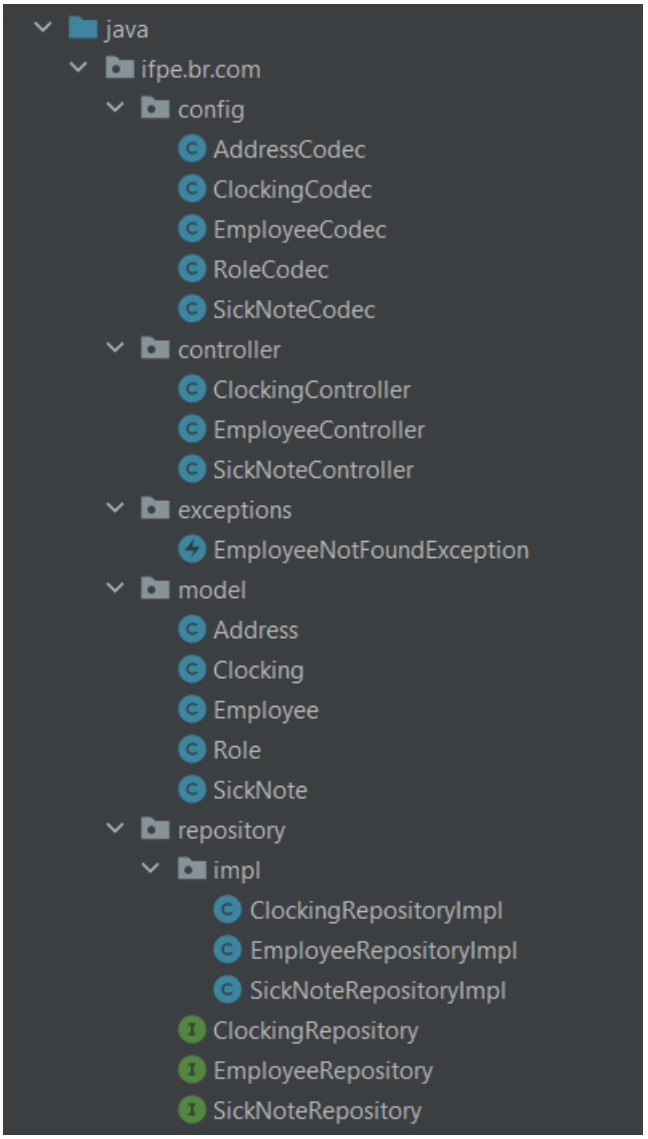
Por fim, a estrutura *Repository* é responsável por todas as classes que interagem com o banco de dados. E no caso desta aplicação, por conter poucas regras de negócio, dentro dessa estrutura estão contidas todas as regras de negócio. Há uma estrutura impl dentro da *Repository*, para fazer a implementação das interfaces declaradas na raíz da estrutura, dessa forma atende a uma característica muito importante da programação orientada a objetos que é o Polimorfismo.

Figura 2 – Arquitetura de soluções utilizada em ambos projetos



Fonte: Elaborado pelo próprio autor.

Figura 3 – Estrutura de classes criadas em ambos projetos



Fonte: Elaborado pelo próprio autor.

4.2 Execução dos Testes Unitários

No desenvolvimento dessa etapa, testes unitários foram criados nas duas aplicações utilizando os *frameworks Junit5* e *Mockito* na versão 5.3.1. O *Junit5* se utiliza-se da plataforma *Junit* para compor um framework de testes completo para o ecossistema do *Java* (JUNIT5,2023). Enquanto, o Mockito é um framework com uma *API* simples e limpa para simulação (MOCKITO,2023). Com o *Junit5* foi possível criar os casos de testes para as funcionalidades e regras de negócio da aplicação e o *Mockito* foi responsável por simular o comportamento de outras classes que necessitam compor o contexto dos testes, mas não era o alvo do teste em si. Para ambas aplicações foram criados 16 cenários de testes para as classes dentro dos domínios de *Controller* e de *Repository*, pois são as classes que concentravam as

regras de negócio e o fluxo de dados consumido pela aplicação.

Além disso, foi utilizado o *plugin JaCoCo (Java Code Coverage)* na versão 0.8.7 para verificar a cobertura dos testes unitários, que totalizou 100% de cobertura nas classes testadas, atestando a qualidade e a confiabilidade dos testes unitários produzidos.

4.3 Execução dos Testes de Carga

O Apache *Jmeter* na versão 5.5 foi utilizado para desenvolver os *scripts* de testes de carga, para cada *endpoint* foi criado um plano de testes no *Jmeter* que recebe uma variável *users*, para identificar a quantidade de requisições simultâneas. Serão utilizados os seguintes comandos no terminal para a execução dos planos de testes:

Tabela 2 – Lista de comandos para execução dos *scripts* do *Jmeter*

Comandos do *Jmeter* que serão executados
jmeter -n -t .\templates*GET_Employee_Test*.jmx -J*users*={*users*} -f -l *output*.csv -e o *dashboard*
jmeter -n -t .\templates*POST_Employee_Test*.jmx -J*users*={*users*} -f -l *output*.csv -e o *dashboard*
jmeter -n -t .\templates*POST_Clocking_Test*.jmx -J*users*={*users*} -f -l *output*.csv -e o *dashboard*
jmeter -n -t .\templates*POST_SickNote_Test*.jmx -J*users*={*users*} -f -l *output*.csv -e o *dashboard*

Fonte: Elaborado pelo próprio autor.

Algumas *flags* foram adicionadas aos comandos demonstrados acima, com algumas finalidades específicas:

- **-n**: responsável por indicar ao *Jmeter* que o teste de carga será executado no modo sem interface gráfica
- **-t**: indica ao *Jmeter* qual arquivo de teste jmx será executado
- **-f**: aponta que se houver algum arquivo de saída de resultado dos testes com o mesmo nome, ele será substituído pelo arquivo gerado pela execução do teste atual
- **-l**: especifica qual será o nome e a localização dos arquivos de resultado de saída
- **-e**: indica ao *Jmeter* para gerar um relatório *HTML* após a execução do teste
- **-o**: aponta em qual diretório será salvo o relatório gerado em *HTML*

Para a execução dos testes de carga será utilizado uma instância c6i.8xlarge de com*put*ação em nuvem da *AWS* (*Amazon Web Services)* que conforme o *site* oficial da *AWS*: “A *AWS* foi projetada para ser um dos ambientes de computação em nuvem mais flexíveis e seguros atualmente disponíveis” (*AWS*,2023). Esta instância

possui um *hardware* com 32 *vCPUs* provenientes de um processador físico *Intel Xeon* 8375C (Ice Lake) e memória *RAM* de 64GB, o sistema operacional utilizado foi o *Ubuntu* 22.04 LTS. Por fim, os testes serão realizados com apenas a aplicação a ser testada e o *Jmeter* executando no sistema operacional, de modo a garantir que haja o mínimo de interferência no desempenho. Ainda assim, vale ressaltar que há limitações quanto ao controle da utilização dos recursos da máquina durante os testes, uma vez que há processos do próprio sistema operacional que podem interferir nos resultados que serão obtidos.

5 RESULTADOS E ANÁLISE

Este capítulo aborda as duas últimas etapas mencionadas na seção da Metodologia, demonstrando de qual modo as métricas foram extraídas e realizando a comparação entre as mesmas.

5.1 Extração das Métricas

Para extrair as métricas necessárias, comandos serão executados sequencialmente no terminal para que os planos de testes construídos no *Jmeter* realizem as requisições aos respectivos *endpoints* conforme cada cenário definido.

Visando responder a **QP1** será realizada a execução dos testes de carga no *Jmeter* com 2000 requisições simultâneas, assim extraindo as métricas e comparando-as entre as duas tecnologias, os seguintes comandos serão executados no terminal:

Tabela 3 – Lista de comandos para executados para o cenário proposto na **QP1**

Comandos executados para a **QP1**
.\j*meter* -n -t .\templates*POST_Employee*_Test.jmx -J*users*=2000 -f -l *output*.csv -e -o *dashboard*
.\j*meter* -n -t .\templates*GET_Employee*_Test.jmx -J*users*=2000 -f -l *output*.csv -e -o *dashboard*
.\j*meter* -n -t .\templates*POST_Clocking*_Test.jmx -J*users*=2000 -f -l *output*.csv -e -o *dashboard*

Fonte: Elaborado pelo próprio autor.

Os seguintes resultados foram obtidos da execução dos comandos da tabela acima:

Tabela 4 – Resultados obtidos após a execução dos comandos da Tabela 3

Execução	Tecnologia	Tempo médio de Resposta (ms)	Tempo mínimo de resposta (ms)	Tempo máximo de resposta (ms)	Taxa média de transferência de dados (KB/s)
Primeira Execução	*Spring Boot*	121,91	1	620	694,33
	Quarkus	145,14	1	658	693,24
Segunda Execução	*Spring Boot*	2805,74	1574	3745	55,27
	Quarkus	3391,35	1723	4874	48,42
Terceira Execução	*Spring Boot*	360,15	1	1392	407,96
	Quarkus	423,75	2	1381	408,82

Fonte: Elaborado pelo próprio autor.

A fim de responder a **QP2** será realizada a execução dos testes de carga no *Jmeter* com 200 requisições simultâneas, desse modo as métricas serão extraídas e comparadas entre as duas tecnologias, os seguintes comandos serão executados no terminal:

Tabela 5 – Lista de comandos para executados para o cenário proposto na **QP2**

Comandos executados para a **QP2**
.\j*meter* -n -t .\templates*POST_Employee*_Test.jmx -J*users*=200 -f -l *output*.csv -e -o *dashboard*
.\j*meter* -n -t .\templates*GET_Employee*_Test.jmx -J*users*=200 -f -l *output*.csv -e -o *dashboard*
.\j*meter* -n -t .\templates*POST_Clocking*_Test.jmx -J*users*=200 -f -l *output*.csv -e -o *dashboard*

Fonte: Elaborado pelo próprio autor.

Após a execução dos comandos citados na tabela acima, os seguintes resultados foram obtidos:

Tabela 6 – Resultados obtidos após a execução dos comandos da Tabela 5

Execução	Tecnologia	Tempo médio de Resposta (ms)	Tempo mínimo de resposta (ms)	Tempo máximo de resposta (ms)	Taxa média de transferência de dados (KB/s)
Primeira Execução	*Spring Boot*	3,65	1	30	143,84
	Quarkus	4,30	1	29	143,06
Segunda Execução	*Spring Boot*	19,93	13	56	27,51
	Quarkus	66,19	45	145	26,27
Terceira Execução	*Spring Boot*	4,79	1	32	84,28
	Quarkus	5,45	2	31	84,28

Fonte: Elaborado pelo próprio autor.

Com o propósito de trazer uma resposta para a **QP3** será realizada a execução de testes de carga usando o *Jmeter* com 100 requisições simultâneas no *endpoint POST* rh/*api*/*sickNote*, passando o caminho para um arquivo de 500 *Megabytes* no corpo da requisição, o seguinte comando será executado no terminal:

Tabela 7 – Lista de comandos para executados para o cenário proposto na **QP3**

Comandos executados para a **QP3**
.*jmeter* -n -t .\templates*POST_SickNote*_Test.jmx -J*users*=100 -f -l *output*.csv -e -o *dashboard*

Fonte: Elaborado pelo próprio autor.

Os consecutivos resultados foram extraídos após a execução do comando na tabela acima:

Tabela 8 - Resultados obtidos após a execução dos comandos da Tabela 7

Framework	**Tempo médio de resposta (ms)**	**Tempo mínimo de resposta (ms)**	**Tempo máximo de resposta (ms)**	**Taxa média de transferência de dados (KB/s)**
Spring Boot	366445,73	365708	367209	0,08
Quarkus	776382,68	773966	777146	0,04

Fonte: Elaborado pelo próprio autor.

Por fim para responder a **QP4** será realizada a execução de testes de carga usando o *Jmeter* com 1000 requisições simultâneas no *endpoint* *POST* rh/*api*/*sickNote*, com o caminho para um arquivo de 5KB no corpo da requisição, o seguinte comando será executado no terminal:

Tabela 9 – Lista de comandos para executados para o cenário proposto na **QP4**

Comandos executados para a **QP4**
.*jmeter* -n -t .\templates*POST_SickNote*_Test.jmx -J*users*=1000 -f -l *output*.csv -e -o *dashboard*

Fonte: Elaborado pelo próprio autor.

Posteriormente à execução do comando exposto na tabela a cima, os seguintes resultados foram coletados:

Tabela 10 - Resultados obtidos após a execução dos comandos da Tabela 9

Tecnologia	**Tempo médio de resposta (ms)**	**Tempo mínimo de resposta (ms)**	**Tempo máximo de resposta (ms)**	**Taxa média de transferência de dados (ms)**
Spring Boot	882,56	557	1529	187,43
Quarkus	1384,70	369	1944	138,64

Fonte: Elaborado pelo próprio autor.

5.2 Análise de Métricas Extraídas

Com as métricas extraídas após a execução dos planos de teste do *Jmeter*

através dos comandos no terminal, podemos realizar as devidas análises baseadas nas questões de pesquisa específicas estipuladas na Seção 2 deste trabalho.

5.2.1 QP1: Qual tecnologia possui melhor a performance para uma alta quantidade de requisições realizadas por usuários concorrentes?

Para o primeiro e o terceiro cenário, o *Spring Boot* conseguiu métricas de Tempo Médio de Resposta e Tempo Máximo de Resposta levemente melhores em comparação ao *Quarkus*. No entanto, no segundo cenário em que as requisições ocorrem a um *endpoint* com verbo *HTTP GET*, essas duas métricas foram significativamente superiores ao *Quarkus*. Portanto, o *Spring Boot* consegue se sair um pouco melhor que o *Quarkus* em um contexto de várias requisições concorrentes, podendo se tornar bem superior caso a requisição seja realizada um *endpoint* com método de requisição *GET*.

5.2.2 QP2: Qual *tecnologia* possui melhor a performance para uma quantidade baixa de requisições realizadas por usuários concorrentes?

O *Spring Boot* conseguiu obter resultados significativos em comparação ao *Quarkus* nas requisições realizadas ao *endpoint* que retorna os funcionários. Já nas requisições realizadas aos demais *endpoints*, o *Spring Boot* se sobressaiu levemente. Dessa forma, podemos determinar que o *Spring Boot* possui uma performance levemente melhor para uma quantidade baixa de requisições simultâneas, caso o *endpoint* tenha o verbo *HTTP GET*, essa diferença de performance pode se tornar mais substancial.

5.2.3 QP3: Qual tecnologia possui a melhor performance para transacionar arquivos que possuem 500 *Megabytes*?

O *Spring Boot* exibiu resultados mais satisfatórios na performance nesse cenário, conseguindo o dobro de Taxa Média de Transferência de Dados que o *Quarkus*, além de ter tido um tempo mais que duas vezes menor nas outras métricas. Portanto, é notória a superioridade de performance do *Spring Boot* para transacionar arquivos grandes em face do *Quarkus*.

5.2.4 QP4: Qual tecnologia possui a melhor performance para transacionar arquivos que possuem 5 *Kilobytes*?

Para transações de arquivos pequenos, o *Spring Boot* se demonstrou superior ao *Quarkus* em quase todas as métricas, com a exceção do Tempo Mínimo de Resposta. Consequentemente, é possível determinarmos que para esse cenário em específico, o *Spring Boot* possui uma performance mais robusta que o *Quarkus*.

6 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo final realizar uma comparação de desempenho entre duas tecnologias utilizadas no desenvolvimento de aplicações *Java* que se

conectem a um banco de dados não relacional baseado em documentos como o *MongoDB*. Assim tornando mais claro para quem deseja implementar aplicações nesse contexto e que necessitem de alta performance, possa decidir qual é a melhor tecnologia para o seu caso de uso.

Após a obtenção dos resultados ao executar as etapas definidas no capítulo de Metodologia, foi possível notar que o *Spring Boot* demonstrou ter uma performance superior ao *Quarkus*, em todos cenários apresentados dentro do contexto proposto por este trabalho. No entanto, a melhoria de performance é mais notória para *endpoints* com método de requisição *HTTP GET* e também para *endpoints* que transacionem arquivos grandes. Apesar de ter sido verificado para esse contexto que o *Spring* tem uma performance melhor, seriam interessantes trabalhos futuros que abordassem outros contextos como a utilização de banco de dados relacionais, assim como banco de dados não relacionais orientados a grafos ou do tipo chave-valor, além de verificar se a utilização de um contexto aplicado ao *Kubernetes* pode afetar o desempenho dos *frameworks*, especialmente do *Quarkus* que possui suporte nativo à essa ferramenta.

REFERÊNCIAS

ALMEIDA, Matheus. **UMA ANÁLISE COMPARATIVA DE DESEMPENHO ENTRE DIFERENTES TECNOLOGIAS DE EXECUÇÃO DE APLICAÇÕES WEB DO LADO DO SERVIDOR.** Monografia (Bacharelado em Engenharia da Computação). Universidade Federal de São Carlos. 2020.

Amazon Web Services. **AWS**, 2023. Computação em nuvem com a AWS. Disponível em: https://aws.amazon.com/pt/what-is-aws/. Acesso em: 06 jul. 2023.

Apache. **JMETER**, 2023. Apache JMeter. Disponível em: https://jmeter.apache.org/. Acesso em: 06 jul. 2023.

GAMMA, Erich et al. **PADRÕES DE PROJETO: SOLUÇÕES REUTILIZÁVEIS DE SOFTWARES ORIENTADOS A OBJETO**. Porto Alegre: Bookman, *2007.*

Junit. **JUNIT 5**, 2023. JUnit 5 User Guide. Disponível em: https://junit.org/junit5/docs/current/user-guide/. Acesso em: 06 jul. 2023.

Kubernetes. **KUBERNETES**, 2023. Orquestração de contêineres prontos para produção. Disponível em: https://kubernetes.io/pt-br/. Acesso em: 06 jul. 2023.

Mockito. **MOCKITO**, 2023. Why drink it? Disponível em: https://site.mockito.org/. Acesso em: 06 jul. 2023.

MongoDB. **MONGODB**, 2023. O que é o MongoDB? Disponível em: https://www.mongodb.com/pt-br/what-is-mongodb. Acesso em: 07 mai. 2023.

ASSIS, Roniê e Bavaresco, Jorge. **ESTUDO COMPARATIVO DOS FRAMEWORKS SPRING BOOT, MICRONAUT E QUARKUS PARA O DESENVOLVIMENTO DE MICROSSERVIÇOS EM JAVA.** Trabalho de conclusão de curso (Bacharelado em Ciência da Computação). Instituto Federal de Educação, Ciência e Tecnologia Sul-rio-grandense. 2022.

VMware Tanzu. **SPRING BOOT**, 2023. Spring Boot Disponível em: https://spring.io/projects/spring-boot. Acesso em: 07 mai. 2023.

WALLS, Craig. **SPRING IN ACTION**. New York: Manning, 2014.