UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE GRADUAÇÃO
CURSO DE SISTEMAS DE INFORMAÇÃO

GABRIEL FERREIRA DA ROSA

Análise comparativa de performance entre Spring Boot e Quarkus: Um estudo de caso

São Leopoldo
2022

GABRIEL FERREIRA DA ROSA

Análise comparativa de performance entre Spring Boot e Quarkus: Um estudo de caso

Artigo apresentado como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação, pelo Curso de Sistemas de Informação da Universidade do Vale do Rio dos Sinos - UNISINOS

Orientador: Prof. Ph.D. Kleinner Silva Farias de Oliveira

São Leopoldo

2022

Análise comparativa de performance entre Spring Boot e Quarkus: Um estudo de caso

Gabriel Ferreira da Rosa[1]
Kleinner Farias[2]

Resumo: Performance desempenha um papel fundamental em projetos de aplicações Web, sendo o tema amplamente pesquisado na literatura. Ao longo dos últimos anos, vários *frameworks* Java, tais como Spring Boot e Quarkus, buscam melhorar sua performance, a fim de se manterem relevantes no mercado. No entanto, a literatura atual carece de análises comparativas que ajudem os desenvolvedores no momento de escolha de um dos *frameworks*. Embora sejam bastante utilizados, poucos trabalhos buscam prover uma análise comparativa, o que leva muitos desenvolvedores a basearem-se nas suas experiências e não em um conhecimento empírico. Esse estudo, portanto, reporta uma análise comparativa de performance entre Spring Boot e Quarkus. Para isso, um estudo de caso foi realizado no contexto de cenários de comunicação via mensagens e a persistência das mesmas em uma base de dados. Para tanto, duas aplicações foram desenvolvidas utilizando os respectivos *frameworks*. Dados do consumo de CPU (Central Processing Unit), consumo de memória RAM (Randon Access Memory) e o tempo de processamento de mensagens foram utilizados para medir a performance de cada aplicação alvo. Os indicadores obtidos demonstraram, estatisticamente, que o Quarkus possui uma performance ligeiramente superior na maioria dos cenários analisados. Todavia, esse é um estudo inicial que busca explorar o tema e abrir caminho para futuras pesquisas com outros cenários e elementos.

Palavras-chave: Spring Boot; Quarkus; RabbitMQ; nativo; performance.

1 INTRODUÇÃO

A performance, um requisito não funcional, é de suma importância, sendo mais importante até mesmo que requisitos funcionais individuais. Uma performance ruim pode significar o descarte de um serviço inteiro ou até mesmo de um conjunto de serviços, consequentemente acarretando em prejuízos financeiros (SOMMERVILLE, 2011). Tendo como premissa, diversos *frameworks* Java buscaram melhorar sua performance, seja através de *refactoring* interno ou a implementação de novas tecnologias. Tecnologias como Ahead-of-time (AOT), a

[1] Graduando em Sistemas de Informação pela Unisinos. Email: gabrielfr97@gmail.com

[2] Doutor em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (2012). Mestre em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul (2008). Graduado em Ciência da Computação pela Universidade Federal de Alagoas (2006) e em Tecnologia da Informação pelo Instituto Federal de Alagoas (2006). Email: kleinnerfarias@unisinos.br

compilação do Java para código nativo, Hot Reload of Live Code, afinidade com container, entre outras levam a modificações dos *frameworks* já existentes e a criação de novos. No entanto, a literatura nestes casos não acompanha na mesma velocidade das alterações realizadas, criando assim questões como: “Qual framework devo utilizar?”.

Pode parecer uma questão simples. Todavia, como destaca Sommerville (2011, p. 302) “Eles são inerentemente complexos e podem demorar meses para alguém aprender a usá-los. Pode ser difícil e caro avaliar *frameworks* disponíveis para a escolha do *framework* mais adequado.”

Escolher a ferramenta que melhor atende às necessidades do cliente, neste caso um *framework*, traz um conjunto de benefícios como: a redução dos custos de infraestrutura, redução do tempo de resposta e uma melhor experiência do usuário (LARSSON, 2020). Diante disso, estudos de análise entre Spring Boot e Quarkus ajudam neste processo de avaliação, na medida em que são elaborados estudos sobre os mais diversos cenários.

Alguns trabalhos sobre o tema buscam realizar essa análise entre Spring Boot e Quarkus. O estudo realizado por Almeida (2020) analisa aplicações compiladas em código nativo e código a ser executado pela JVM (Java Virtual Machine) através de requisições HTTP (Hypertext Transfer Protocol). Entretanto, estudos de análise entre Spring Boot e Quarkus, ambos nativos, com a utilização do RabbitMQ como *message broker*, não foram encontrados. Logo, essa lacuna motivou o desenvolvimento deste trabalho, tomando como base o trabalho citado.

Portanto, este artigo reporta uma análise comparativa de performance entre os *frameworks* Java Spring Boot e Quarkus. Para isso, um estudo de caso foi realizado para avaliar o Spring Boot e Quarkus no contexto de uma aplicação e mensageria, em termos de consumo de CPU, consumo de memória, e tempo de processamento. Uma aplicação de mensageria foi desenvolvida com o Spring Boot e com o Quarkus. Buscou-se manter o código o mais próximo possível entre as aplicações, evitando funcionalidades específicas de cada *framework*. Tais aplicações foram submetidas a 3 cenários de cargas de mensagens sendo eles de 150 mil, 250 mil e 500 mil, cada um deles executado 10 vezes. Os resultados obtidos demonstram que há vantagem do Quarkus frente ao Spring Boot na maioria das variáveis analisadas. Os resultados obtidos podem beneficiar futuros

desenvolvedores a escolherem o *framework* que trará melhor resultado num contexto de mensageria.

O estudo está estruturado da seguinte forma: Seção 2 contém o referencial teórico, que traz conceitos que são utilizados na pesquisa; a Seção 3 trata de trabalhos relacionados, trazendo um breve resumo; a Seção 4 descreve a metodologia utilizada, elucidando os objetivos e hipóteses da pesquisa assim como suas variáveis e processo de captura dos dados e análise; a Seção 5 apresenta os resultados obtidos, através de tabelas e gráficos; e, por fim, a Seção 6 aborda conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção aborda os conceitos teóricos usados durante a construção e desenvolvimento do estudo.

2.1 Código nativo e GraalVM

Java é conhecido por ser uma linguagem interpretada, em que o Bytecode roda na JVM. Nas versões mais antigas, possuíam uma performance muito inferior a linguagens compiladas diretamente para código nativo. Todavia, foram lançadas versões que trouxeram técnicas para mitigar esse problema, como o JIT-compilers. Porém, novas tecnologias como a GraalVM buscam torná-la ainda mais rápida através da pré-compilação do Java em código nativo, permitindo, assim, aproximar-se das linguagens que compilam para código nativo como o C (LARSON, 2020).

A GraalVM surgiu com a proposta de ser a próxima geração de *Virtual Machines* de altíssima performance. Para tal propósito, reuniu um conjunto de recursos como *Graal Compiler*, *GraalVM Native Image Mechanism,* T*ruffle Language Implementation Framework* e *Sulong*, que são essenciais para garantir a capacidade poliglota (Šipek; Muharemagić; Mihaljević; Radovan, 2020).

2.2 RabbitMQ

O RabbitMQ é uma mensageria desenvolvida pela Rabbit Technologies, que tem por objetivo gerenciar as mensagens em sistemas distribuídos. Logo, consegue integrar diferentes sistemas com diferentes linguagens, provendo o gerenciamento de carga, de falhas e de mensagens (IONESCU, 2015). O RabbitMQ utiliza o protocolo *Advanced Message Queuing Protocol* (AMQP), que permite uma comunicação assíncrona entre as entidades que interagem com ele. Ademais, as entidades não precisam estar funcionando ao mesmo tempo para se comunicarem, o RabbitMQ tem a capacidade de guardar as mensagens até que um consumidor esteja disponível e encaixa-se nas regras de processamento (SHARVARI; NAG, 2019).

As mensagens são recebidas e enviadas de acordo com regras que são definidas no momento da configuração. Quando uma mensagem é publicada, deve possuir um *routing key* que endereçará para a fila correta. Quando há o processamento, ou consumo como também é conhecido, a entidade que processa essa mensagem recebe a mensagem, realiza todo o processamento sobre ela e ao final envia ao RabbitMQ um *acknowledgement* (ACK), que indica que a mensagem foi processada sem a ocorrência de problemas (SHARVARI; NAG, 2019).

2.3 Frameworks

O Spring Boot é um framework que surgiu de uma simplificação do Spring Framework. O objetivo foi de tornar o tempo de configuração o menor possível, ou seja, a partir do momento em que o projeto foi criado em poucos passos a aplicação está pronta para subir para no ambiente de produção (GUTIERREZ, 2019). Outra grande vantagem é a sua integração com aplicações de mercado de terceiros, onde o Spring Boot oferece um processo rápido de integração através de suas anotações e um gerenciamento de configuração (MOHAN, 2022).

O Quarkus é um framework Java versátil, ideal para *serverless*, microsserviços e containers. Para tanto oferece um tempo de inicialização baixo, baixo consumo de memória RSS (*Resident Set Size*) e uma boa escalabilidade (PLESSIS; MENDES; CORREIA, 2021). O Quarkus apresenta um tratamento diferente no que tange Java Reflection, por esse modo ser o contrário da premissa do modo nativo. No modo

nativo busca-se conhecer todas as informações das classes no momento da compilação, já o Java Reflection obtém as informações quando a aplicação está em execução. Para obter o melhor dos dois, o Quarkus oferece arquivos de configuração e anotações para fazer uso do Java Reflection (KOLEOSO, 2020).

2.4 Estudo de caso

Estudo de caso é uma forma de investigação empírica que busca investigar uma situação dentro de um determinado contexto, utilizando métodos quantitativos e qualitativos. É uma estratégia de pesquisa abrangente, que pode ser efetuada de diversas maneiras, a depender dos procedimentos escolhidos pelo pesquisador de acordo com a situação. No entanto, os procedimentos devem ser seguidos para garantir a qualidade da pesquisa (YIN, 2001). De acordo com Wohlin (2012), trata-se de várias fontes de evidência a fim investigar fenômenos em um determinado contexto, principalmente quando não há clareza entre fenômeno e contexto.
Em função da característica tanto do estudo de caso, assim como, dos frameworks, citados na seção 2.3, de estarem alinhados a contextos específicos e também por não se conseguir realizar uma separação clara entre os fenômenos e o contexto do objeto em estudo. O estudo de caso foi escolhido porque consiste em uma investigação para avaliar como os dois frameworks se comportam em um determinado contexto.

3 TRABALHOS RELACIONADOS

A pesquisa pelos trabalhos relacionados foi realizada no repositório digital *Google Scholar*. O principal termo utilizado para realizar a seleção dos trabalhos foi “spring boot quarkus performance”.

3.1 Análise dos trabalhos relacionados

(ALMEIDA, 2020). O trabalho apresenta uma comparação entre Spring Boot nativo, Spring Boot JVM, Quarkus nativo e Quarkus JVM. Através dessas formas de compilação, execução e frameworks Java, foram realizados testes para comparar o desempenho das aplicações. Levou-se em conta, principalmente, o número de

requisições HTTP (*Hyper Text Transfer Protocol*) e o tempo de resposta obtido para cada aplicação. O autor apresenta as dificuldades encontradas inerentes ao uso de tecnologias novas que estão sendo amadurecidas. Verificou-se neste estudo uma superioridade do Quarkus nativo em quase todos os cenários executados. Por fim, traz uma importante observação notada tanto pelo autor como no estudo de Larsson (2020, p. 1).

> "[...] comparamos o desempenho das versões community edition e enterprise edition do GraalVM ao OpenJDK e OracleJDK, usando Java 8 e Java 11 […]. Descobrimos que o desempenho dos diferentes JDKs variam significativamente dependendo do teste, o que torna difícil fazer qualquer conclusão definitiva." Larsson, R. [Tradução nossa].

(BACK, 2016). Apresenta uma análise comparativa entre os métodos de integração de serviços REST (*Representational State Transfer*) e AMQP (*Advanced Message Queuing Protocol*). Foram realizados diversos testes no ambiente local e Heroku onde foi medida a latência e vazão das duas abordagens. Reportando, ainda, a complexidade observada em cada método.

(BUONO; PETROVIC, 2016). Neste estudo, os autores propõem uma nova forma de comunicação entre os microsserviços, substituindo a comunicação baseada em texto por uma comunicação baseada em binários, através do Protocol Buffers. Para tanto, utilizam uma arquitetura moderna com Quarkus e GraalVM. Destaca, também, sua integração com o Kubernetes que permite que durante um momento de tráfego de dados intenso e a necessidade de mais instâncias, o Quarkus propicia a subida rápida de uma nova instância. Nesse sentido, os autores concluem que a utilização de uma comunicação binária leva a uma redução considerável no tempo de resposta.

(RITZAL, 2020). Apresenta as tecnologias que a GraalVM utiliza e como elas se tornam mais rápidas que outras *Virtual Machines* para Java. Além disso, há uma breve introdução aos principais *frameworks* Java, como Spring, Micronaut e Quarkus. Com a utilização da GraalVM é possível uma otimização desses *frameworks*. O trabalho apresenta uma comparação entre a JVM e a GraalVM no que tange o tempo de inicialização da aplicação, o uso de memória e o tempo de execução.

(SOUZA, 2020). Apresenta uma visão geral das tecnologias RabbitMQ e Apache Kafka. Faz uma análise tanto em termos qualitativos como o

desacoplamento por tempo e garantia de entrega, como em termos quantitativos através da latência e teste de ambiente controlado. Traz também uma análise em quais cenários cada tecnologia possui melhor performance.

(FONG; RAED, 2021). O trabalho apresenta um teste de performance Java Development Kits (JDKs), GraalVM Enterprise Edition (EE) e Community Edition (CE) contra Oracle JDK e OpenJDK para Java 8 e 11. Para tanto, foi utilizado uma coleção de casos de teste onde cada JDK foi testada. Os resultados obtidos demonstram que GraalVM EE 11 obteve uma performance melhor na maioria dos testes, entretanto também foi verificado que o hardware desempenha um papel fundamental na performance da mesma.

3.2 Análise comparativa dos trabalhos relacionados

Critérios de Comparação. Os Critérios de Comparação (CC) são utilizados para realizar a comparação entre o trabalho proposto e os trabalhos relacionados selecionados.

- **Estudo Empírico (CC1):** as conclusões são baseadas em evidências empíricas concretas e que podem ser validadas por meio de dados obtidos através de experimentos e testes.
- **Análise de consumo de memória (CC2):** a capacidade de verificar o consumo de memória da aplicação testada.
- **Análise de consumo de CPU (CC3):** a capacidade de verificar o consumo de processamento da aplicação testada.
- **Análise do tempo de processamento das mensagens (CC4):** a capacidade de verificar o tempo de processamento das mensagens.

O resultado da comparação dos trabalhos relacionados e do trabalho proposto é apresentado no Quadro 1. A partir da análise, os seguintes pontos foram observados: apenas 3 trabalhos analisam o consumo de recursos; apenas 1 trabalho verifica o tempo de processamento das mensagens; nenhum trabalho apresenta uma análise do tempo de processamento das mensagens aliado ao consumo de recursos.

Quadro 1 – Análise comparativa dos Trabalhos Relacionados selecionados

Trabalho Relacionado	Critério de Comparação			
	CC1	CC2	CC3	CC4
Trabalho Proposto	X	X	X	X
(ALMEIDA, 2020)	X	X	X	0
(BACK, 2016)	X	0	0	X
(BUONO; PETROVIC, 2016)	X	X	0	0
(RITZAL, 2020)	X	X	0	0
(SOUZA, 2020)	X	0	0	X
(FONG; RAED, 2021)	X	0	0	0

Fonte: Elaborado pelo autor.

Oportunidades de pesquisa. Com base nos pontos destacados no Quadro 1, foi identificada a seguinte oportunidade de pesquisa: execução de um estudo de caso de forma empírica sobre a performance de Spring Boot e Quarkus, nativos, em um contexto de mensageria utilizando os critérios explicitados acima. A oportunidade de pesquisa é explorada nas seções abaixo.

4 METODOLOGIA

A seção relata a metodologia utilizada na execução da pesquisa empírica. A seção 4.1 apresenta o objetivo e a questão da pesquisa proposta. A seção 4.2 apresenta as hipóteses. A seção 4.3 apresenta as variáveis e métricas. A seção 4.4 os artefatos e ferramentas que foram utilizados para a realização do estudo. A seção 4.5 apresenta as aplicações alvo. A seção 4.6 descreve o processo experimental. A seção 4.7 descreve o processo de análise. A metodologia é baseada em estudos anteriores: (LAZZARI, 2021), Evaluating the effort of composing design models: a controlled experiment (FARIAS; GARCIA; WHITTLE; CHAVEZ; LUCENA, 2013) e Wohlin (2012).

4.1 Objetivo e Questão de Pesquisa

O **objetivo** primário desta pesquisa é fazer uma análise comparativa de performance entre os *frameworks* Spring Boot e Quarkus, no contexto de aplicações de mensageria. Em particular, este objetivo busca entender os efeitos dos frameworks sobre três diferentes variáveis: consumo de CPU, consumo de memória RAM e tempo de processamento de mensagens. Tais frameworks são analisados através de cenários sintéticos de processamento de mensagens (Seção 4.6), utilizando como plataforma de mensagens RabbitMQ. A seguir, o objetivo desta pesquisa é formulado seguindo no modelo GQM (BASILI, 1992):

Analisar os *frameworks* Java
com o propósito de investigar seus efeitos
em relação à performance
através da perspectiva de desenvolvedores
no contexto de aplicações de mensageria.

O artigo, em particular, almeja produzir evidências empíricas sobre a performance do Spring Boot e do Quarkus, ambos utilizando a GraalVM, em cenários de processamento de mensagens. Neste sentido, a seguinte questão de pesquisa (QP) foi formulada:

QP: Será que a performance do Quarkus é superior ao do Spring Boot no contexto de aplicações de mensageria e compiladas em código nativo com a GraalVM?

4.2 Hipóteses

Hipótese 1. Conjectura-se que o Quarkus pode apresentar uma performance superior ao Spring Boot, já que a pesquisa realizada por Almeida (2020) indica essa vantagem em um contexto de requisições HTTP. O Quarkus busca outros mecanismos para otimização, como o desestímulo do uso da API *Reflection* que tem elevado custo, ao contrário do Spring Boot que faz uso extenso (KOLEOSO, 2020). Outro ponto a ser observado é que o Spring Boot nativo encontra-se, ainda, em fase

experimental, diferentemente do Quarkus que possui uma versão final. Apesar disso, o Spring Boot possui bibliotecas de integração com RabbitMQ e MongoDB desenvolvidas especificamente para ele, o que pode traduzir-se em uma performance melhor.

Desta forma, a performance será uma síntese da utilização de recursos de hardware e tempo de processamento das mensagens, portanto, declarando as hipóteses nula e alternativa da seguinte maneira:

> **Hipótese Nula, H_{nula}:** Spring Boot utiliza menos (ou igual) recursos de hardware e possui menor (ou igual) tempo de processamento das mensagens que o Quarkus.
>
> $H1_{nula}$ *: Performance(Quarkus) ≤ Performance(Spring Boot)*
>
> **Hipótese Alternativa, H_{alt}:** Quarkus utiliza menos recursos de hardware e possui menor tempo de processamento das mensagens que o Spring Boot.
>
> $H1_{alt}$ *: Performance(Quarkus) > Performance(Spring Boot)*

Testando essa hipótese, possibilitará verificar o uso de recursos de *hardware* e tempo de consumo das mensagens. Comparando assim através destas variáveis as duas aplicações alvo. Os dados colhidos serviram de insumo para a tomada de decisão de futuros usuários desses *frameworks*.

4.3 Variáveis e Métricas

Variável independente. A variável independente nesta hipótese é a de *frameworks* e a arquitetura orientada a mensagens.

Variável dependente. A variável dependente nesta hipótese é a de métricas de performance definidas no Quadro 1. O conhecimento dessa variável permite a realização de análises para entender o impacto de cada *framework* nos testes. A variável está divida em três facetas: consumo de CPU, consumo de memória RAM e tempo de processamento das mensagens.

Consumo de CPU. Dado representa a porcentagem do uso de CPU do sistema, sendo esse capturado a cada cinco segundos pelo Prometheus e exibido

nos gráficos do Grafana. O dado é obtido através de uma média de todos os valores capturados durante o período do teste.

Consumo de memória RAM. Dado representa em MebiByte o uso de memória RAM, sendo esse capturado a cada cinco segundos pelo Prometheus e exibido nos gráficos do Grafana. O dado é obtido através de uma média de todos os valores capturados durante o período do teste.

Tempo de Processamento de mensagem. Obtido através da diferença de tempo entre o comando de início da aplicação alvo e a última mensagem gravada no banco de dados MongoDB. Essa informação está contida no campo *date*, no corpo da mensagem. Quando a aplicação alvo em teste processa a mensagem, esse campo recebe a data e horário atual e grava a mensagem no banco de dados.

A métrica de performance está representada no Quadro 2.

Quadro 2 - Métrica de performance

Nome	Descrição	Ferramenta
Consumo de memória	Consumo do recurso de memória durante os testes de carga em MebiByte.	Grafana
Consumo de CPU	Consumo do recurso de CPU durante os testes de carga em porcentagem.	Grafana
Tempo de processamento de mensagem	Tempo decorrido entre o comando de início da aplicação alvo e a gravação da última mensagem no banco de dados.	MongoDB

Fonte: Elaborado pelo autor.

4.4 Artefatos e ferramentas utilizados nesta pesquisa

O estudo necessita de ferramentas e artefatos para que seja efetuado. Portanto, o Quadro 3 apresenta a lista com todos os elementos utilizados e suas respectivas versões.

Quadro 3 - Artefatos e ferramentas utilizados na pesquisa

Artefato	Versão
Maven	3.6.3
Spring Boot	2.6.4
Spring Boot Actuator	2.6.4
Spring Boot Web	2.6.4
Spring Boot AMQP	2.6.4
Spring Boot Data Mongodb	2.6.4
Spring Boot Native (experimental)	0.11.3
GraalVM	22.0.0.2
MongoDB	5.0.6
RabbitMQ	3.8.4
Grafana	8.5.0
Prometheus	2.35.0
Micrometer Registry Prometheus	1.8.3
Quarkus	2.7.4
Quarkus Camel Bom	2.7.4
Quarkus RabbitMQ Client	0.5.0
Quarkus Resteasy	2.7.4
Quarkus Mongo Client	2.7.4
Jackson Databind	2.12.4

Fonte: Elaborado pelo autor.

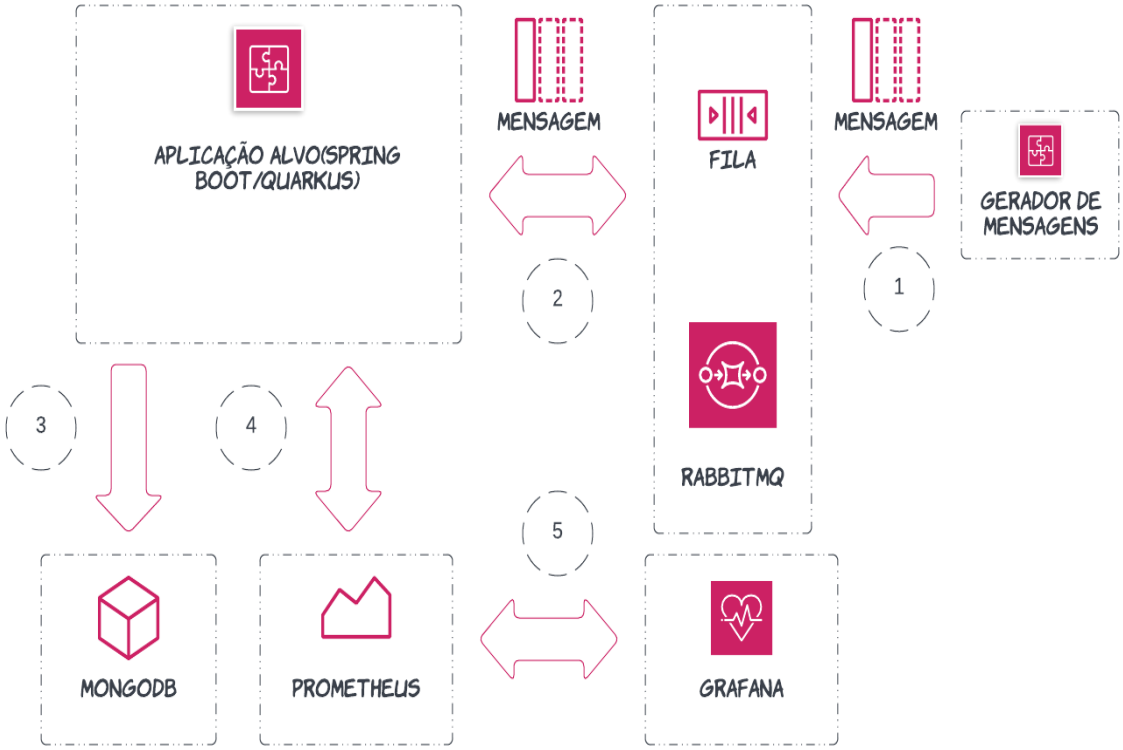
A máquina utilizada para a execução dos testes foi um notebook Lenovo com 16GB de memória RAM DDR4, SSD (Solid State Drive) 256GB, processador Intel Core i7 10610U 1.8GHz e o sistema operacional Ubuntu.

4.5 Aplicações alvo

Duas aplicações foram desenvolvidas para a pesquisa. Sendo uma desenvolvida utilizando o Spring Boot e outra utilizando o Quarkus, ambas utilizam a linguagem Java e a GraalVM para realizar a compilação para código nativo. As aplicações são de pequeno porte, realizam apenas o processamento das mensagens que consiste em receber a mensagem, definir o campo *date*, salvar no banco e retornar o ACK para o RabbitMQ. Sendo assim, é possível evitar possíveis interferências durante o processamento de uma mensagem e os dados obtidos são referentes, realmente, ao *framework* em teste.

As aplicações conectam-se com três ferramentas, sendo elas: message broker RabbitMQ, banco de dados MongoDB e com a ferramenta de monitoramento Prometheus. A Figura 1 possui uma ilustração esquemática do fluxo de dados das aplicações alvo. No passo 1 uma aplicação geradora de mensagens é acionada; no passo 2 a aplicação alvo, em teste, conecta-se ao *message broker* RabbitMQ para receber as mensagens e realizar o envio da confirmação de leitura; no passo 3 salva as mensagens no banco de dados MongoDB; no passo 4 ocorre envio de informações contendo consumo de recursos para o Prometheus e por último no passo 5 há a comunicação entre Prometheus e Grafana para que os gráficos sejam gerados no Grafana. Importante destacar que o passo 2 começa somente quando o passo 1 terminar, ou seja, quando terminar de enviar o lote definido de mensagens. O fluxo de dados do passo 4 acontece de forma independente dos passos 2 e 3.

Figura 1 – Ilustração do fluxo de dados



Fonte: Elaborada pelo autor.

4.5.1 Testes de carga

Os testes de carga iniciam quando o gerador de mensagens é acionado, recebendo como parâmetro o número de mensagens que devem ser geradas. A Figura 2 apresenta a estrutura da mensagem. Assim que o envio de todas as mensagens é efetuado, a aplicação alvo é inicializada e passa a consumir as mensagens. Ao inicializar, a comunicação com o Prometheus é inicializada e logo em seguida os gráficos começam a ser formados no Grafana, permitindo assim acompanhar o teste e iniciar a captura das métricas.

Alguns pontos importantes:

- O período de inicialização da aplicação, onde ainda não há consumo de mensagens, é levado em consideração nas métricas.
- A cada teste todos os dados do teste anterior são removidos do banco de dados.

A Figura 2 apresenta a estrutura de mensagem utilizada nos testes. O gerador de mensagens preenche com dados aleatórios os campos: *id*, *name*,

description e *status*. O campo *date* é preenchido na aplicação alvo, em teste, antes da persistência da mensagem no banco de dados.

Figura 2 – Estrutura da mensagem

```
{
  "id": "e8204258-c995-11ec-9d64-0242ac120002",
  "name": "B4vCh5T56r",
  "description: "jGlqeXibSI",
  "status": "qGSr9sWJYx",
  "date": "2022-02-02T03:18:43Z"
}
```

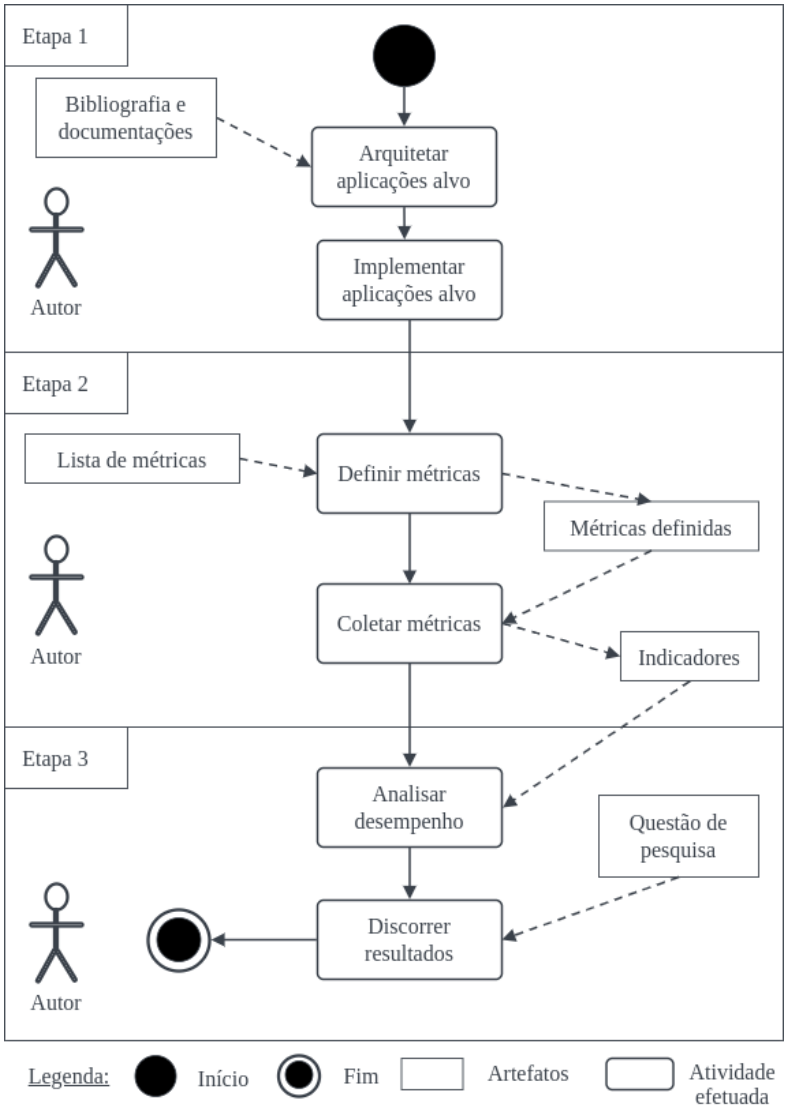
Fonte: Elaborada pelo autor.

A estratégia adotada para essa pesquisa foi de produzir um determinado número de mensagens, enviá-las para o RabbitMQ e depois consumi-las. O motivo é que não haja a interferência do agente produtor sobre as métricas coletadas da aplicação alvo, ou seja, caso o produtor tenha uma capacidade menor de produção que a capacidade de consumo do consumidor, não impactará no resultado.

4.6 Processo experimental

Na Figura 3 é possível ver o processo experimental adotado, o qual é formado por três etapas: desenvolver as aplicações alvo (1), definir e coletar métricas (2) e analisar os dados coletados (3). Cada etapa é elucidada a seguir.

Figura 3 – Processo experimental



Fonte: Elaborada pelo autor.

Etapa 1: Desenvolver as aplicações alvo. Nesta etapa a pesquisa foca na busca de estudos e documentações dos *frameworks* que servem de insumo para o desenvolvimento das aplicações alvo. O resultado foi a implementação de duas aplicações que estão alinhadas com os mais recentes estudos e tecnologias utilizadas pela comunidade. Assim como um código que utiliza boas práticas e segue as documentações disponibilizadas.

Etapa 2: Definir e coletar métricas. As métricas são definidas nesta etapa de acordo com a lista de métricas de consumo de recursos disponibilizada pelo Java e a métrica de tempo de consumo das mensagens. A partir da definição, iniciam-se os testes de carga definidos na Seção 4.5.1, ao mesmo tempo as métricas são coletadas e agrupadas. O resultado são indicadores que serviram de insumo para a Etapa 3.

Etapa 3: Analisar os dados coletados. O estudo, agora, busca por meio dos indicadores coletados na Etapa 2 e aplicação do teste de Wilcoxon, analisar e confirmar ou refutar as hipóteses propostas na Seção 4.2. Para tanto, os indicadores são exibidos em tabelas e gráficos que permitem a identificação visual dos resultados do estudo.

4.7 Processo de análise

Análise quantitativa. A inferência estatística é utilizada para a realização do teste da hipótese proposta. O teste não paramétrico de Wilcoxon é aplicado, já que não requer dois conjuntos separados de amostras distribuídas identicamente. Uma diferença significativa é encontrada quando o *p-value* $\leqslant 0.05$.
Análise qualitativa. Os dados qualitativos são coletados a partir do que é observado durante o desenvolvimento da pesquisa. Sendo assim, evidências não tangíveis, tão somente, pelos dados quantitativos podem ser apresentadas e enriquecer a pesquisa.

5 RESULTADOS

Esta seção tem como objetivo apresentar os resultados referentes às questões de pesquisas formuladas na Seção 2. A Seção 5.1 apresenta a análise descritiva dos dados coletados. A seção 5.2 apresenta uma discussão sobre a hipótese elaborada na seção 4. A seção 5.3 apresenta uma discussão sobre os resultados das variáveis analisadas. A seção 5.4 apresenta as limitações do estudo. Por último, a seção 5.5 relata algumas observações feitas durante o estudo.

5.1 Estatística descritiva

A seção aborda os aspectos dos dados coletados referentes a performance dos *frameworks* estudados. Para tanto, utiliza-se de tendências como médias e medianas e as dispersões por meio do desvio padrão para a realização de uma análise sobre a distribuição dos dados das variáveis observadas.

5.1.1 Consumo de Memória

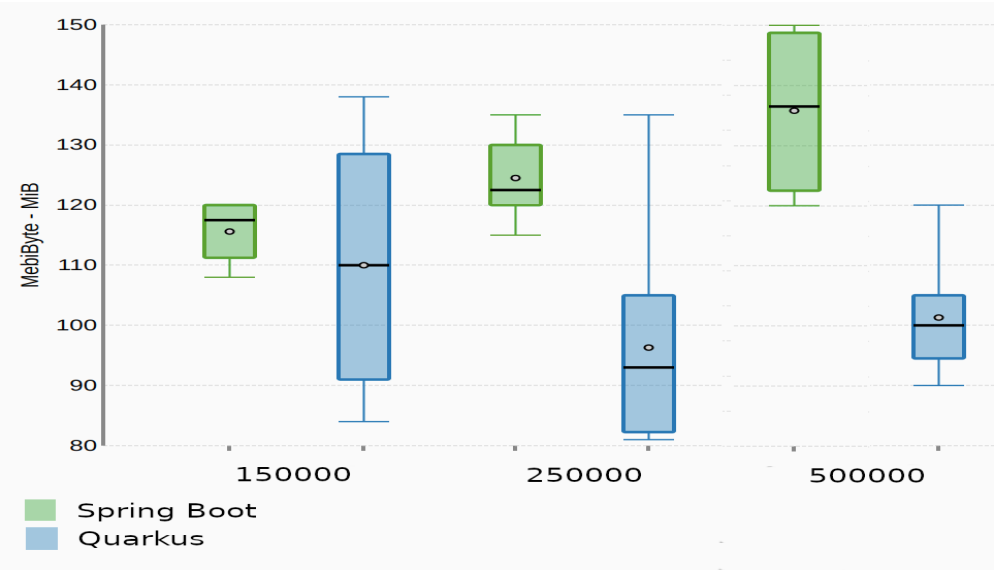
Estatística descritiva. As médias de uso de memória RAM para cada cenário com Spring Boot foram de 115,6MiB, 124,5MiB e 135,8MiB, já para os cenários com Quarkus foram de 110MiB, 96,3MiB e 101,3MiB. Portanto, verificou-se que na média o Quarkus utiliza menos memória RAM, Verificou-se, também, que o Quarkus apresenta um desvio padrão mais elevado, principalmente nos cenários de 150 mil e 250 mil mensagens. A Tabela 1 evidencia melhor esses comportamentos.

Tabela 1 – Resultado do consumo de Memória (medidas em MebiByte – MiB)

Aplicação alvo	Número de mensagens	N	Menor	Q1	Mediana	Q3	Maior	Média	Desvio Padrão
Spring Boot	150000	10	108	111,25	117,5	120	120	115,6	5,21
Quarkus	150000	10	84	91	110	128,5	138	110	21,5
Spring Boot	250000	10	115	120	122,5	130	135	124,5	7,61
Quarkus	250000	10	81	82,25	93	105	135	96,3	17,04
Spring Boot	500000	10	120	122,5	136,5	148,75	150	135,8	12,81
Quarkus	500000	10	90	94,5	100	105	120	101,3	9,03

Fonte: Elaborada pelo autor.

Figura 4 – Diagrama de box-plot do consumo de memória.



Fonte: Elaborada pelo autor.

5.1.2 Consumo de CPU

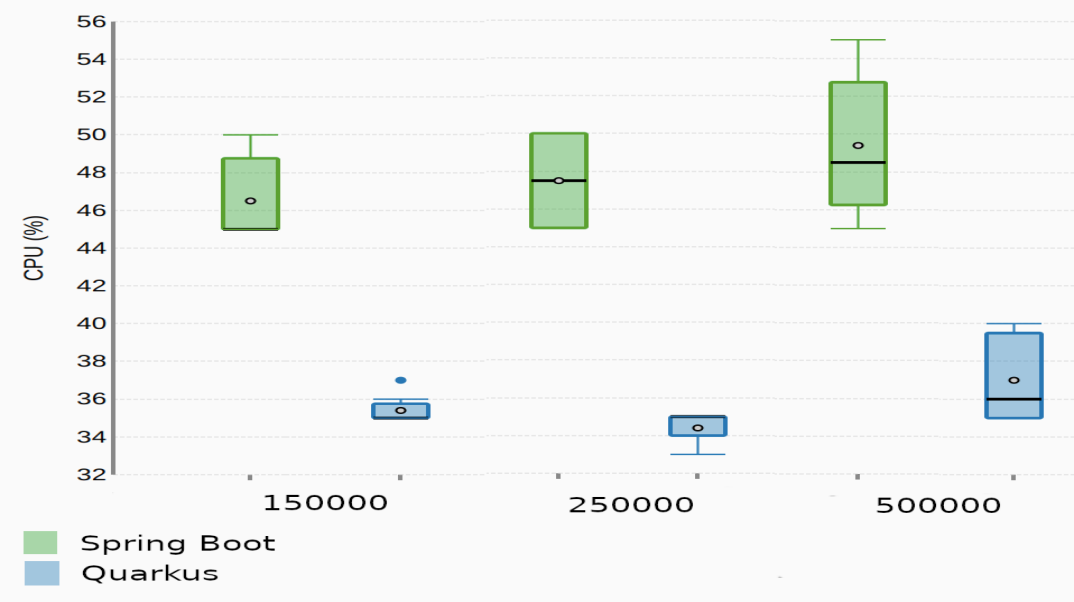
Estatística descritiva. As médias de uso de CPU para cenários com o Spring Boot foram 46,5%, 47,5% e 49,4%, já para os cenários com o Quarkus foram 35,4%, 34,4% e 37%. Logo, verificou-se que o Quarkus consome menos recursos da CPU que o Spring Boot. Além disso, o Spring Boot apresentou um desvio padrão superior ao do Quarkus em todos os cenários, destacando que, nos cenários em que há menos mensagens (150 mil e 250mil), essa diferença é maior.

Tabela 2 – Resultados consumo de CPU

Aplicação alvo	Número de mensagens	N	Menor	Q1	Mediana	Q3	Maior	Média	Desvio Padrão
Spring Boot	150000	10	45	45	45	48,75	50	46,5	2,41
Quarkus	150000	10	35	35	35	35,75	37	35,4	0,69
Spring Boot	250000	10	45	45	47,5	50	50	47,5	2,63
Quarkus	250000	10	33	34	35	35	35	34,4	0,84
Spring Boot	500000	10	45	46,25	48,5	52,75	55	49,4	3,8
Quarkus	500000	10	35	35	36	39,5	40	37	2,3

Fonte: Elaborada pelo autor.

Figura 5 – Diagrama de box-plot do consumo de CPU.



Fonte: Elaborada pelo autor.

5.1.3 Tempo de execução

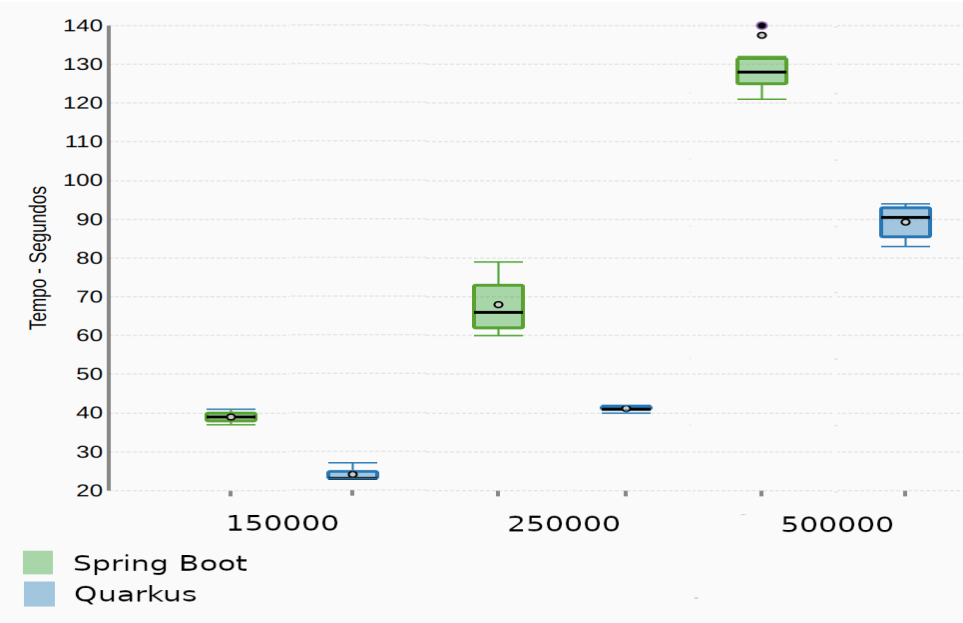
Estatística descritiva. As médias de tempo de processamento para o cenário com Spring Boot foram 39s, 68s e 137,5, já para os cenários com Quarkus foram 24s, 41,2s e 89,3s. Através destes valores foi possível notar que o Quarkus consegue processar as mensagens mais rapidamente em todos os cenários. Com o aumento no número de mensagens o desvio padrão teve um grande aumento nos cenários com o Spring Boot, saltando de 1,49s com 150 mil mensagens para 28,2s com 500 mil mensagens, conforme exposto pela tabela 3.

Tabela 3 – Resultados do tempo de consumo das mensagens (em segundos)

Aplicação alvo	Número de mensagens	N	Menor	Q1	Mediana	Q3	Maior	Média	Desvio Padrão
Spring Boot	150000	10	37	38	39	40	41	39	1,49
Quarkus	150000	10	23	23	23	24,75	27	24	1,49
Spring Boot	250000	10	60	62	66	73	79	68	6,99
Quarkus	250000	10	40	41	41	41,75	42	41,2	0,63
Spring Boot	500000	10	121	125	128	131,5	214	137,5	28,2
Quarkus	500000	10	83	85,5	90,5	93	94	89,3	4,19

Fonte: Elaborada pelo autor.

Figura 6 – Diagrama de box-plot do tempo de processamento das mensagens.



Fonte: Elaborada pelo autor.

5.2 Teste da Hipótese

Testes estatísticos foram executados a fim de verificar se há diferença de performance entre os *frameworks*. A performance é dada através de três variáveis: consumo de CPU, consumo de memória RAM e tempo de processamento das mensagens. A análise conjugada das três indica se a hipótese que o Quarkus possui uma performance superior é verdadeira. Foi aplicado o teste Wilcoxon com a significância em 0,05 (*p-value* $\leqslant 0.05$).

Verifica-se que a $\mathbf{H_{nula}}$ pode ser descartada já que, conforme os dados apresentados na seção 5.1, o Quarkus apresenta resultados melhores. Logo, pode-se afirmar que, levando em conta todos os cenários executados e analisados, $\mathbf{H_{alt}}$ é verdadeira na medida que somente o cenário de 150mil mensagens na variável de consumo de memória RAM, ao analisar estatisticamente o Quarkus, não é superior ao Spring Boot, conforme a Tabela 4.

Tabela 4 – Teste estatístico Wilcoxon

Cenário	Estatística	CPU	Memória Ram	Tempo de Processamento
150000	p-value	0,005	0,374	0,005
250000	p-value	0,005	0,009	0,006
500000	p-value	0,006	0,006	0,006

Fonte: Elaborada pelo autor.

Consumo de memória RAM. Foi hipotetizado que o Quarkus possui um consumo menor, logo uma performance superior ao do framework Spring Boot. Conforme a Tabela 4, a $\mathbf{H_{alt}}$ pode ser confirmada para os cenários de 250 mil e 500mil, entretanto, para o cenário de 150 mil essa hipótese, estatisticamente, não pode ser confirmada onde o *p-value* é 0,374.

Consumo de CPU. Foi hipotetizado que o Quarkus possui um consumo menor de CPU. Conforme a Tabela 4, a $\mathbf{H_{nula}}$ pode ser descartada já que em todos os cenários, estatisticamente, o Quarkus consome menos, logo obtém uma performance melhor.

Tempo de processamento. Foi hipotetizado que o Quarkus possui um tempo de processamento menor que o Spring Boot. Conforme a Tabela 4, a $\mathbf{H_{nula}}$ pode ser

descartada já que em todos os cenários, estatisticamente, o Quarkus processa mais rapidamente que o Spring Boot, logo obtém uma performance melhor.

5.3 Discussão

Análise de consumo de memória. Ao analisar o consumo de memória, de ambas as aplicações, foi possível notar que o Quarkus possui um consumo significativamente menor em comparação ao Spring Boot. Entretanto, o Quarkus possui também uma variação de consumo significativamente maior que o Spring Boot, porém, o consumo de memória não aumentou, ainda que tenha aumentando o número de mensagens. Especula-se diante dos dados que o Spring Boot apresenta um melhor gerenciamento dos recursos, portanto, apesar de consumir mais, não se verifica grandes variações como no Quarkus.

Análise de consumo de CPU. Ao analisar o consumo de CPU, de ambas as aplicações, nota-se que as duas mantiveram o consumo linear, independentemente do tamanho do lote de mensagens. Assim como, evidenciou-se que o Quarkus apresentou um consumo ligeiramente menor que o Spring Boot. Especula-se que esse consumo melhor ocorre em função das técnicas que o Quarkus utiliza para obter um consumo melhor de recursos, como por exemplo: a utilização restrita do Java Reflection.

Análise do tempo de processamento. Ao analisar o consumo de CPU, de ambas as aplicações, nota-se que o Quarkus conseguiu processar mais rapidamente as mensagens que o Spring Boot. Um ponto de destaque é a variação elevada observada em uma amostra com 500 mil mensagens no Spring Boot, todavia essa variação pode ser explicada por possíveis processamentos paralelos, não intencionais, sendo executados na máquina em que se realizou o teste.

5.4 Limitações do estudo

Trata-se de uma pesquisa inicial que busca trazer uma luz sobre o tema, explora tecnologias novas e que ainda estão sendo desenvolvidas e amadurecidas. Portanto, a pesquisa possui algumas limitações que devem ser levadas em consideração. Os dados coletados são referentes ao cenário e configurações

específicas, qualquer alteração, mesmo que aplicada para ambas, pode levar a dados completamente distintos.

5.5 Observações

Alterar as configurações de interação entre MongoDB e Quarkus pode aumentar ou diminuir o consumo de mensagens em mais de 120 vezes por segundo. Nesse sentido, há uma carência de documentações sobre as configurações referente a conexão entre os dois e no Quarkus como um todo. A carência deixa o desenvolvimento moroso e a aplicação suscetível a erros.

A compilação do Spring Boot no modo nativo mostrou-se extremamente demorada em comparação à compilação normal, além de consumir todos os recursos da máquina e, muitas vezes, provocando sua paralisação devido a falta de memória RAM.

Como o Quarkus é um *framework* relativamente novo, a quantidade de documentações, artigos, trabalhos acadêmicos e discussões na Internet são menores em comparação ao Spring Boot. Isso se traduz em um desenvolvimento mais demorado e propenso a realização de códigos menos performáticos, já que não foram amplamente testados.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho realizou um estudo empírico para analisar a performance Spring Boot e Quarkus em um ambiente de mensageria. As aplicações foram submetidas a um conjunto de testes, a fim de coletar dados previamente selecionados. Dados como consumo de CPU, consumo de memória RAM e tempo de processamento de mensagens foram observados e analisados. Os resultados demonstram que na maioria dos testes o Quarkus apresentou resultados melhores que o Spring Boot.

Os resultados obtidos neste estudo, apontam para um consumo de recursos de hardware menores com Quarkus em comparação ao Spring Boot. Assim, como o tempo de processamento foi menor com Quarkus. Através desse estudo é possível tomar decisões baseada em conhecimentos empíricos no momento de escolha de um *framework* Java. Importante destacar que as conclusões desse estudo estão

ligadas ao contexto onde os testes foram executados e uma generalização não é possível.

Como trabalhos futuros, pretende-se realizar: (1) outros cenários de teste, com outros protocolos de comunicação; (2) realização de testes de cargas intermitentes; (3) considerar mais parâmetros para a avaliação. Esse trabalho busca incrementar os trabalhos já existentes na área de análise de performance entre os frameworks, para tanto trouxe um novo cenário com outras tecnologias envolvidas com o RabbitMQ. Serve, também, como estudo inicial envolvendo as tecnologia Java, GraalVM, Spring Boot, Quarkus, RabbitMQ e MongoDB, provendo espaço para estudos relacionados ou mais profundos.

REFERÊNCIAS

ALMEIDA, Matheus Santos De. **Uma Análise Comparativa De Desempenho Entre Diferentes Tecnologias De Execução De Aplicações Web Do Lado Do Servidor.** Monografia**.** Universidade Federal de São Carlos. (Engenharia da Computação). São Carlos 2020. Disponível em: <https://bit.ly/3N8IztG>. Acesso em: 30 maio 2022.

BACK, Renato Pereira. **Análise Comparativa De Técnicas De Integração Entre Microsserviços.** Universidade Federal De Santa Catarina. Florianópolis, 2016. Disponível em: <https://bit.ly/3N3qxIW>. Acesso em: 30 maio 2022.

BASILI, V. R. **Software modeling and measurement**: the Goal/Question/Metric paradigm. [S.l.], 1992.

BUONO, Vincenzo; PETROVIC Petar. **Enhance Inter-service Comminication In Supersonic K-Native REST-based Java Microservice Architectures.** Kristianstad University Sweden, 2021. Disponível em: <https://bit.ly/3wWshhJ>. Acesso em: 30 maio 2022.

FARIAS, Kleinner; GARCIA, Alessandro; WHITTLE, Jon; CHAVEZ, Christina von Flach Garcia; LUCENA, Carlos. Evaluating the effort of composing design models: a controlled experiment. **Model Driven Engineering Languages and Systems.** Berlin, pp. 676–691, vol. 7590, 2013. Disponível em: <https://bit.ly/3PQJNv9>. Acesso em: 30 maio 2022.

FARIAS, Kleinner; GARCIA, Alessandro; LUCENA, Carlos. Effects of Stability on Model Composition Effort: An Exploratory Study. **Journal on Software and Systems Modeling**, vol. 13, Issue 4, pp. 1473–1494, 2014.

FONG, Fredric; RAED, Mustafa. **Performance comparison of GraalVM, Oracle JDK and OpenJDK for optimization of test suite execution time.** 2021. Disponível em: <https://bit.ly/3wYwW2t>. Acesso em: 30 maio 2022.

GUTIERREZ, Felipe. **Pro Spring Boot 2 An Authoritative Guide to Building Microservices, Web and Enterprise Applications, and Best Practices**. Library of Congress Centrol. 2019.

IONESCU, Valeriu Manuel. **The Analysis of the Performance of RabbitMQ and ActiveMQ.** IEEE 2015. Disponível em: <https://bit.ly/38wo8aX>. Acesso em: 30 maio 2022.

KOLEOSO, Tayo. **Beginning Quarkus Framework**: Build Cloud-Native Enterprise Java Applications and Microservices. 2020.

LARSSON, Robin. **Evaluation of GraalVM Performance for Java Programs.** 2020. Disponível em: <https://bit.ly/3LUtcnb>. Acesso em: 25 maio 2022.

LAZZARI, Luan; FARIAS, Kleinner. **Os efeitos da arquitetura dirigida a eventos na modularidade de software: Um estudo exploratório.** 2021. Disponível em: <https://bit.ly/3t7w8Gl>. Acesso em: 15 maio 2022.

MOHAN, Anand. "**Kafka Streaming Application using Java Spring Boot**." (2022). Disponível em: <https://bit.ly/3lUqDXy>. Acesso em: 15 maio 2022.

PLESSIS, Shani du; MENDES, Bruno; CORREIA, Noélia. **A Comparative Study of Microservices Frameworks in IoT Deployments**. 2021 International Young Engineers Forum (YEF-ECE), pp. 86-91, 2021. Disponível em: <https://bit.ly/3GwOXbq>. Acesso em: 15 maio 2022.

RITZAL, Roman. **Optimizing Java For Serverless Applications.** (Dissertação de mestrado) University of Applied Sciences, FH Campus Wien 2020. Disponível em: <https://bit.ly/3wVRXK4>. Acesso em: 30 maio 2022.

SHARVARI, T.; NAG, K. Sowmya. **A study on Modern Messaging Systems-Kafka, RabbitMQ and NATS Streaming.** Cornell University, 2019. Disponível em: <https://arxiv.org/abs/1912.03715>. Acesso em: 30 maio 2022.

ŠIPEK M.; MUHAREMAGIĆ, D.; MIHALJEVIĆ, B.; RADOVAN, A. **Enhancing Performance of Cloud-based Software Applications with GraalVM and Quarkus**. 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), 2020, pp. 1746-1751, 2020. Disponível em: <https://bit.ly/3azWLxr>. Acesso em: 30 maio 2022.

SOMMERVILLE, Ian. **Engenharia de Software.** 9.ed. Tradução Ivan Bosnic e Kalinka G O Gonçalves. São Paulo: Pearson Prentice \hall, 2011.

SOUZA, Ronan de Araújo. **Perfomance Analysis Between Apache Kafka And RabbitMQ**. (Trabalho de Conclusão de Curso) Universidade Federal de Campina Grande 2020. Disponível em: <https://bit.ly/3t4y4zA>. Acesso em: 30 maio 2022.

WOHLIN, Claes, et al. **Experimentation in software engineering**. Springer Science & Business Media, 2012.

YIN, Robert. **Estudo de caso** Planejamento e Métodos 2.ed. Porto Alegre: Bookman, 2001.