



UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Investigação dos Efeitos do Envelhecimento de Software e Avaliação de Políticas de Rejuvenescimento em Sistema de Orquestração de Contêineres

Dissertação de Mestrado

José Marcelo dos Santos



São Cristóvão – Sergipe

2024

UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

José Marcelo dos Santos

**Investigação dos Efeitos do Envelhecimento de Software e
Avaliação de Políticas de Rejuvenescimento em Sistema de
Orquestração de Contêineres**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de mestre em Ciência da Computação.

Orientador(a): Prof. Dr. Jean Carlos Teixeira de Araujo
Coorientador(a): Prof. Dr. Rubens de Souza Matos Júnior

São Cristóvão – Sergipe

2024

**FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL
UNIVERSIDADE FEDERAL DE SERGIPE**

S237i Santos, José Marcelo dos.
Investigação dos efeitos do envelhecimento do software e avaliação de políticas de rejuvenescimento em sistema de orquestração de contêineres / João Marcelo dos Santos ; orientador Jean Carlos Teixeira de Araujo. – São Cristóvão, SE, 2024.

90 f. : il.

Dissertação (mestrado em Ciência da Computação) – Universidade Federal de Sergipe, 2024.

1. Computação. 2. Computação em nuvem. 3. Arquitetura de software. 4. Contêineres. I. Araújo, Jean Carlos Teixeira de, orient. II. Título.

CDU 004.05



UNIVERSIDADE FEDERAL DE SERGIPE
PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA
COORDENAÇÃO DE PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ata da Sessão Solene de Defesa da Dissertação do
Curso de Mestrado em Ciência da Computação-UFS.
Candidato: José Marcelo dos Santos

Em 27 dias do mês de fevereiro do ano de dois mil e vinte quatro, com início às 16h, realizou-se na Sala de Seminários do PROCC da Universidade Federal de Sergipe, na Cidade Universitária Prof. José Aloísio de Campos, a Sessão Pública de Defesa de Dissertação de Mestrado do candidato **José Marcelo dos Santos**, que desenvolveu o trabalho intitulado: *“Investigação dos Efeitos do Envelhecimento de Software e Avaliação de Políticas de Rejuvenescimento em Sistema de Orquestração de Contêineres”*, sob a orientação do Prof. Dr. Jean Carlos Teixeira de Araújo e coorientação do Prof. Dr. Rubens de Souza Matos Júnior. A Sessão foi presidida pelo Prof. Dr. Jean Carlos Teixeira de Araújo (PROCC/UFS), que após a apresentação da dissertação passou a palavra aos outros membros da Banca Examinadora, Prof. Dr. Paulo Romero Martins Maciel (UFPE), o Prof. Dr. Edward David Moreno Ordóñez (Procc/UFS) e, em seguida, o Prof. Dr. Rubens de Souza Matos Júnior (Procc/UFS). Após as discussões, a Banca Examinadora reuniu-se e considerou o mestrando (a) APROVADO. Atendidas as exigências da Instrução Normativa 05/2019/PROCC, do Regimento Interno do PROCC (Resolução 67/2014/CONEPE), e da Resolução no 04/2021/CONEPE que regulamentam a Apresentação e Defesa de Dissertação, e nada mais havendo a tratar, a Banca Examinadora elaborou esta Ata que será assinada pelos seus membros e pelo mestrando.

Documento assinado digitalmente
gov.br
JEAN CARLOS TEIXEIRA DE ARAUJO
Data: 28/02/2024 12:25:48-0300
Verifique em <https://validar.iti.gov.br>

Prof. Dr. Jean Carlos Teixeira de Araújo
(PROCC/UFS)
Presidente


Prof. Dr. Paulo Romero Martins Maciel
(UFPE)
Examinador Externo

Documento assinado digitalmente
gov.br
PAULO ROMERO MARTINS MACIEL
Data: 01/03/2024 13:27:56-0300
Verifique em <https://validar.iti.gov.br>

Documento assinado digitalmente
gov.br
RUBENS DE SOUZA MATOS JUNIOR
Data: 09/03/2024 18:21:10-0300
Verifique em <https://validar.iti.gov.br>

Prof. Dr. Rubens de Souza Matos Júnior
(PROCC/UFS)
Coorientador

Documento assinado digitalmente
gov.br
EDWARD DAVID MORENO ORDÓNEZ
Data: 01/03/2024 12:52:00-0300
Verifique em <https://validar.iti.gov.br>

Prof. Dr. Edward David Moreno Ordóñez
(PROCC/UFS)
Examinador Interno

Documento assinado digitalmente
gov.br
JOSE MARCELO DOS SANTOS
Data: 11/03/2024 08:31:56-0300
Verifique em <https://validar.iti.gov.br>

José Marcelo dos Santos
Candidato

Este trabalho é dedicado com profunda saudade e eterna gratidão aos meus amados pais (em memória), José Carlos e Josefa Odete. Suas vidas foram a inspiração para este esforço, e seu amor continua a guiar-me, mesmo na ausência física.

Agradecimentos

Agradeço primeiramente a Deus por sempre me guiar em todos os desafios da vida.

Quero expressar minha gratidão aos meus pais (em memória), José Carlos e Josefa Odete, pelo incentivo na minha formação educacional e por terem influenciado positivamente na formação do meu caráter. À minha esposa, Gabriella Caxico, agradeço pela parceria, paciência e por segurar a barra com nossos dois pequenos. A conquista de concluir esta etapa é tão minha quanto de vocês, pois foi com o apoio, amor e compreensão de vocês que consegui chegar até aqui. Agradeço também aos meus pequeninos, João e Marcele, pela paciência com o papai e por entenderem minha ausência em muitos momentos. Agradeço ainda aos meus irmãos, Carlos e Augusto, pelo apoio constante.

Expresso imensa gratidão também aos meus orientadores, Dr. Jean Carlos Teixeira de Araujo e Dr. Rubens de Souza Matos, por terem me aceitado como orientando, confiado a mim este desafio e me guiado nessa trajetória. Agradeço também ao grupo de Pesquisa *UNAME*, pela troca de experiências ao longo de toda esta desafiadora jornada.

Estendo meu agradecimento a todos os professores do PROCC/UFS que ministraram aulas ao longo deste período. Um agradecimento especial à Elaine por estar sempre disponível para esclarecer qualquer dúvida relacionada ao mestrado.

Agradeço profundamente aos meus coordenadores na SEFAZ/SE, Cristiano Menezes e Rodrigo Cavalcante, por tornarem possível conciliar, em várias ocasiões, as demandas profissionais com as atividades acadêmicas. Também expresso minha gratidão aos colegas de trabalho, Antônio Neto, Jacson Tavares, Alysson Anjos, Pedro Alexandria e Mauro Sandes, pela colaboração valiosa sempre que foi preciso. Ao amigo Gilvan Santos, meu especial agradecimento pela orientação inicial fornecida.

E a todos que me ajudaram diretamente ou indiretamente a finalizar mais uma etapa importante da minha vida.

Resumo

O envelhecimento e rejuvenescimento de software (SAR) têm recebido atenção nos últimos anos e foram extensivamente estudados em diversos cenários computacionais, especialmente em sistemas que demandam alta disponibilidade. Esse foco reflete a crescente importância de sistemas de software para a operação eficiente de várias organizações, tornando-se um tópico de pesquisa proeminente, impulsionado pela crescente demanda por confiabilidade e disponibilidade de serviços. Esta dissertação apresenta experimentos conduzidos em duas versões do Ubuntu Linux, simulando cenários de operação de uma nuvem privada. A plataforma de containerização utilizada é o *Containerd*, com o *Kubernetes* (K8s) como orquestrador, em 5 cenários distintos. Para cada um dos 5 cenários, as condições de hardware e softwares foram mantidas, incluindo cargas de trabalho e durações de testes. Durante a realização dos experimentos, métricas como utilização de CPU, uso de memória e uso de disco foram monitoradas em todo o sistema, além da análise dos serviços do *Containerd* e do *Kubelet*. Os experimentos também incluíram a medição do tempo de resposta de servidor web para solicitações HTTP externas. O cenário inicial concentrou-se na investigação dos efeitos do envelhecimento de software, enquanto cenários subsequentes exploraram a adoção de diferentes estratégias de rejuvenescimento. Efeitos do envelhecimento de software foram observados em todos os cenários, com vazamentos de recursos identificados, especialmente no uso de memória, mesmo quando o *cluster* estava sem carga após um período de estresse. Os problemas observados neste trabalho resultam na degradação do desempenho do sistema e comprometem a confiabilidade e disponibilidade quando o sistema falha devido à exaustão do uso de memória. O rejuvenescimento não surtiu efeito nos cenários de limpeza de memória *cached*, muito menos nos momentos de reinicialização dos serviços do *Containerd* e *Kubelet*. No entanto, o rejuvenescimento de software provou ser eficaz no cenário final, quando a reinicialização controlada dos nós *Workers* foi realizada, permitindo assim o cumprimento do tempo estipulado para o experimento.

Palavras-chave: Envelhecimento de Software, Rejuvenescimento de Software, Kubernetes (K8s), Orquestração, Contêiner.

Abstract

Software Aging and Rejuvenation (SAR) have garnered attention in recent years and have been extensively studied in various computational scenarios, especially in systems requiring high availability. This focus reflects the increasing importance of software systems for the efficient operation of various organizations, becoming a prominent research topic driven by the growing demand for reliability and service availability. This dissertation presents experiments conducted on two versions of Ubuntu Linux, simulating operation scenarios of a private cloud. The containerization platform used is Containerd, with Kubernetes (K8s) as the orchestrator, in 5 distinct scenarios. For each of the 5 scenarios, hardware and software conditions were maintained, including workloads and test durations. During the experiments, metrics such as CPU usage, memory usage, and disk usage were monitored throughout the system, in addition to analyzing the services of Containerd and Kubelet. The experiments also included measuring the web server response time for external HTTP requests. The initial scenario focused on investigating the effects of software aging, while subsequent scenarios explored the adoption of different rejuvenation strategies. Effects of software aging were observed in all scenarios, with resource leaks identified, especially in memory usage, even when the cluster was idle after a stress period. The issues observed in this work result in system performance degradation and compromise reliability and availability when the system fails due to memory usage exhaustion. Rejuvenation had no effect in scenarios involving memory cache cleaning, let alone during the restart of Containerd and Kubelet services. However, software rejuvenation proved effective in the final scenario when controlled restarts of Worker nodes were performed, thus allowing compliance with the stipulated experiment time.

Keywords: Software Aging, Software Rejuvenation, cluster, Kubernetes, Orchestration, Container.

Lista de ilustrações

Figura 1 – Arquitetura da Computação em Nuvem	21
Figura 2 – Arquitetura de Virtualização de VM à Esquerda e de Containerização à Direita	22
Figura 3 – Arquitetura do Cluster Kubernetes	24
Figura 4 – Técnicas de Rejuvenescimento de Software	26
Figura 5 – Metodologia do Experimento	28
Figura 6 – Fases do MSL.	33
Figura 7 – <i>String</i> de busca	34
Figura 8 – Processo de Seleção de Artigos.	35
Figura 9 – Gráfico de Participação nas Publicações (Autor/Ano)	41
Figura 10 – Gráfico da Participação das Instituições	42
Figura 11 – Gráfico das Publicações	43
Figura 12 – Cluster Kubernetes base dos Experimentos	48
Figura 13 – Cenário 1 - Carga de Estresse e Repouso da Aplicação (Ubuntu 20).	50
Figura 14 – Cenário 1 - Carga de Estresse e Repouso da Aplicação (Ubuntu 20).	51
Figura 15 – Cenário 1 - Carga de Estresse e Repouso da Aplicação (Ubuntu 20).	52
Figura 16 – Cenário 1 - Carga de Estresse e Descanso da Aplicação (Ubuntu 22).	53
Figura 17 – Cenário 1 - Carga de Estresse e Descanso da Aplicação (Ubuntu 22).	53
Figura 18 – Cenário 1 - Carga de Estresse e Descanso da Aplicação (Ubuntu 22).	54
Figura 19 – Cenário 2 - Limpeza de <i>caches</i> e <i>buffers</i> (Ubuntu 20).	55
Figura 20 – Cenário 2 - Limpeza de <i>caches</i> e <i>buffers</i> (Ubuntu 20).	56
Figura 21 – Cenário 2 - Limpeza de <i>caches</i> e <i>Buffers</i> (Ubuntu 20).	57
Figura 22 – Cenário 2 - Limpeza de <i>caches</i> e <i>Buffers</i> (Ubuntu 22).	57
Figura 23 – Cenário 2 - Limpeza de <i>caches</i> e <i>Buffers</i> (Ubuntu 22).	58
Figura 24 – Cenário 2 - Limpeza de <i>caches</i> e <i>Buffers</i> (Ubuntu 22).	59
Figura 25 – Cenário 3 - Reiniciar o Serviço <i>Containerd</i> (Ubuntu 20).	59
Figura 26 – Cenário 3 - Reiniciar o Serviço <i>Containerd</i> (Ubuntu 20).	60
Figura 27 – Cenário 3 - Reiniciar o Serviço <i>Containerd</i> (Ubuntu 20).	61
Figura 28 – Cenário 3 - Reiniciar o Serviço <i>Containerd</i> (Ubuntu 22).	61
Figura 29 – Cenário 3 - Reiniciar o Serviço <i>Containerd</i> (Ubuntu 22).	62
Figura 30 – Cenário 3 - Reiniciar o Serviço <i>Containerd</i> (Ubuntu 22).	63
Figura 31 – Cenário 4 - Reiniciando o Serviço <i>Kubelet</i> (Ubuntu 20).	64
Figura 32 – Cenário 4 - Reiniciando o Serviço <i>Kubelet</i> (Ubuntu 20).	65
Figura 33 – Cenário 4 - Reiniciando o Serviço <i>Kubelet</i> (Ubuntu 20).	66

Figura 34 – Cenário 4 - Reiniciando o Serviço <i>Kubelet</i> (Ubuntu 22).	66
Figura 35 – Cenário 4 - Reiniciando o Serviço <i>Kubelet</i> (Ubuntu 22).	67
Figura 36 – Cenário 4 - Reiniciando o Serviço <i>Kubelet</i> (Ubuntu 22).	68
Figura 37 – Cenário 5 - Reiniciando os Nós <i>Workers</i> (Ubuntu 20).	69
Figura 38 – Cenário 5 - Reiniciando os Nós <i>Workers</i> (Ubuntu 20).	70
Figura 39 – Cenário 5 - Reiniciando os Nós <i>Workers</i> (Ubuntu 20).	70
Figura 40 – Cenário 5 - Reiniciando os Nós <i>Workers</i> (Ubuntu 22).	71
Figura 41 – Cenário 5 - Reiniciando os Nós <i>Workers</i> (Ubuntu 22).	71
Figura 42 – Cenário 5 - Reiniciando os Nós <i>Workers</i> (Ubuntu 22).	72

Lista de tabelas

Tabela 1 – Resumo do Estado da Arte.	37
Tabela 2 – Orquestradores de Microserviços mais Utilizados nos Artigos.	43
Tabela 3 – Utilitários de Monitoramento.	45
Tabela 4 – Hardware Usado.	48
Tabela 5 – Tabela Comparativa de tempo de operação entre Cenários.	73

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
abnTeX	ABsurdas Normas para TeX
CE	Critérios de Exclusão
CI	Critérios de Inclusão
CPU	<i>Central Process Unit</i>
CVA	<i>Container Vertical Autoscaling</i>
DCOMP	Departamento de Computação
DDR	<i>Double Data Rate</i>
DRAM	<i>Dynamic Random-Access Memory</i>
HDD	<i>Hard disk Drive</i>
GB	<i>Gigabyte</i>
Gbps	<i>Gigabits per second</i>
GHz	<i>Gigahertz</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IaaS	<i>Infrastructure as a Service</i>
K8s	<i>Kubernetes</i>
LTS	<i>Long-Term Support</i>
LXC	<i>Linux Containers</i>
MB	<i>Megabyte</i>
MT/s	<i>Mega Transfers por segundo</i>
MSL	Mapeamento Sistemático da Literatura
OS	<i>Operating System</i>
PaaS	<i>Platform as a Service</i>
QoS	<i>Quality of Service</i>

QPs	Questões de Pesquisa
RAID	<i>Redundant Array of Independent Disks</i>
RAM	<i>Random Access Memory</i>
RPM	<i>Rotações Por Minuto</i>
RHEV	<i>Red Hat Enterprise Virtualization</i>
SaaS	<i>Software as a Service</i>
SAR	<i>Software Aging and Rejuvenation</i>
SAS	<i>Serial Attached SCSI</i>
SWARE	<i>Stress, Wait and Rejuvenation</i>
TI	Tecnologia da Informação
TVs	Televisores
UFS	Universidade Federal de Sergipe
VM	<i>Virtual Machine</i>
VSZ	<i>Virtual Set Size</i>

Sumário

1	Introdução	15
1.1	Contextualização	15
1.2	Justificativa	17
1.3	Objetivos	18
1.3.1	Objetivo Geral	18
1.3.2	Objetivos Específicos	19
1.4	Organização do Trabalho	19
2	Fundamentação Teórica	20
2.1	<i>Cloud Computing System</i>	20
2.2	Contêineres	21
2.3	Sistemas de Orquestração	22
2.4	Envelhecimento e Rejuvenescimento de Software	24
3	Metodologia	27
3.1	Visão Geral	27
3.2	Pré-avaliação	28
3.3	Avaliação	29
4	Mapeamento Sistemático da Literatura	32
4.1	Metodologia	32
4.1.1	Planejamento	33
4.1.2	Propósito e Questões de Pesquisa (QPs)	33
4.1.3	Base de Dados	33
4.1.4	Termos de Busca	34
4.1.5	Critérios de Inclusão e Exclusão	34
4.1.6	Condução	34
4.1.7	Relatório	35
4.2	Estado da Arte	36
4.3	Análise dos Resultados	41
4.3.1	QP1: Quais são os autores mais produtivos nesta área?	41
4.3.2	QP2: Quais universidades contribuem mais para os artigos mencionados?	42
4.3.3	QP3: Em quais continentes foram conduzidas pesquisas sobre o envelhecimento e rejuvenescimento de software em microsserviços?	42
4.3.4	QP4: Quais são as plataformas e orquestradores de microsserviços mais utilizados neste MSL?	42

4.3.5	QP5: Quais métodos ou técnicas são utilizados para forçar o envelhecimento e rejuvenescimento de microsserviços?	43
4.3.6	QP6: Do ponto de vista da monitorização, quais recursos foram monitorados nos artigos explorados?	44
4.4	Ameaças à Validade	45
4.5	Considerações Finais do Capítulo	46
5	SAR em Sistema de Orquestração de Contêineres	47
5.1	Ambiente Proposto para Avaliação	47
5.2	Análise dos Resultados	49
5.2.1	Cenário #1 - Efeitos do Envelhecimento do Software	49
5.2.2	Cenário #2 - Limpeza de <i>caches</i> e <i>Buffers</i> como Estratégia de Rejuvenescimento	55
5.2.3	Cenário #3 - Reinício do Serviço <i>Containerd</i> como Estratégia de Rejuvenescimento	58
5.2.4	Cenário #4 - Reinicialização do Serviço <i>Kubelet</i> como Estratégia de Rejuvenescimento	64
5.2.5	Cenário #5 - Reinicialização dos Nós de <i>Clusters</i> como Estratégia de Rejuvenescimento	68
6	Considerações Finais	74
6.1	Principais contribuições	76
6.2	Trabalhos futuros	76
	Referências	77
	Apêndices	82
	APÊNDICE A <i>Scripts de Monitoramento</i>	83
	APÊNDICE B <i>Scripts de Execução</i>	86

1

Introdução

Este Capítulo tem por objetivo introduzir o presente trabalho no contexto das estratégias para o rejuvenescimento de software em infraestruturas virtualizadas baseadas em contêineres. Na Seção 1.1, é apresentada uma breve contextualização sobre o crescimento tecnológico, aceitação da virtualização e seu contexto histórico, a base da computação em nuvem e suas entregas, a evolução da computação leve baseada em contêineres, a dependabilidade desses sistemas e uma breve definição sobre envelhecimento e rejuvenescimento de software, além de sua relevância no contexto de sistemas computacionais. Na Seção 1.2, é justificada a importância de tratar o tema proposto, uma vez que a containerização é usada por muitas empresas nos dias atuais e falhas que provoquem indisponibilidade precisam ser estudadas para não prejudicar os usuários finais. Na Seção 1.3, é dito através de objetivos gerais e específicos onde se deseja chegar com os experimentos realizados. Por fim, a Seção 1.4 descreve resumidamente o que é abordado em todos os capítulos sequenciais desse documento.

1.1 Contextualização

Em uma sociedade cada vez mais conectada à Internet, a demanda por serviços mais eficientes e disponíveis está cada vez maior. Neste cenário, a adoção do uso tradicional de servidores físicos tornou-se inviável, devido à incapacidade de acompanhar o crescimento dessas demandas. Neste contexto, surge a virtualização, uma tecnologia que permite a execução de vários sistemas operacionais em um único servidor físico e que trouxe benefícios significativos, proporcionando maior economia de espaço e compartilhamento de recursos (MARTIN, 2022; CHEN; NOBLE, 2001; BABCZYŃSKI et al., 2015). A computação em nuvem, um modelo computacional que tem a virtualização como tecnologia chave, expandiu-se rapidamente, oferecendo serviços como SaaS, PaaS e IaaS (PENG et al., 2009; GUPTA; MISHRA, 2022). Contudo, a alocação de uma aplicação por servidor virtual resultou em desperdício de recursos, impulsionando a adoção de containerização. Esta tecnologia, baseada no *kernel* de sistemas

Unix e Linux, proporciona isolamento de processos e recursos, sendo ágil e eficiente (AULIYA; NURDINSYAH; WULANDARI, 2019; MARTIN, 2022; CABODI; GIORGIO; MINÌ, 2020).

Os contêineres são uma tendência dominante nos sistemas de computação em nuvem, oferecendo uma série de opções como *LXC*, *LXD*, *Singularity*, *Docker*, *Containerd*, *Kata Containers* e *gVisor* (FLAUZAC; MAUHOURLAT; NOLOT, 2020). Para gerenciá-los eficientemente, foram desenvolvidos os orquestradores, responsáveis por automatizar operações e garantir alta disponibilidade, segurança e implantação contínua (NAYDENOV; RUSEVA, 2023). Estima-se que até 2025, cerca de 85% das empresas estarão adotando contêineres em seus ambientes de produção, utilizando plataformas de orquestração como o *Kubernetes* (Mia-Platform Team, 2018). No entanto, assim como outros softwares executados por longos períodos, os contêineres podem falhar devido a ocorrência do envelhecimento do software, um fenômeno de tempo de execução, que afeta um grande número de sistemas de software (PIETRANTUONO et al., 2022), e, que pode impactar diretamente a disponibilidade e o desempenho dos sistemas (OLIVEIRA et al., 2021).

O envelhecimento do software demanda a execução de ações proativas conhecidas como rejuvenescimento de software, as quais visam minimizar os efeitos negativos desse fenômeno. Essas ações se fundamentam em estratégias baseadas no tempo, limiar ou predição (ARAUJO et al., 2011; ARAUJO, 2012; VINÍCIUS et al., 2022). Embora o envelhecimento e rejuvenescimento de software sejam estudados extensivamente há mais de 25 anos em diversos softwares (COTRONEO et al., 2010; COSTA et al., 2021), o contexto da containerização carece de investigações mais profundas para otimizar métodos de previsão de falhas e aplicar o rejuvenescimento do software de maneira eficiente, aumentando ao máximo a disponibilidade dos sistemas.

Neste estudo, conduzimos experimentos em duas versões da distribuição Linux Ubuntu, 20.04.6 LTS e 22.04.2 LTS, simulando cenários de funcionamento de uma nuvem privada. Cada *Cluster* consistia em uma configuração com 1 nó *Master* e 3 nós *Workers*, utilizando o *Containerd* como *runtime* de contêineres e o *Kubernetes* (*K8s*) como orquestrador, em 5 cenários distintos. Mantivemos as mesmas condições de execução dos experimentos, como configurações, cargas de trabalho e a duração dos testes. Esses experimentos envolveram variações de carga de trabalho aplicada à infraestrutura de nuvem, monitorando métricas como CPU, memória RAM, disco, serviços do *Containerd* e *Kubelet*, além disso, foram realizadas medições de tempo de resposta de um servidor web por meio de requisições HTTP externas. O cenário inicial focou na investigação dos efeitos do envelhecimento do software, enquanto os demais cenários exploraram a adoção de diferentes estratégias de rejuvenescimento. Nossa análise destaca a detecção e tratamento do envelhecimento, oferecendo *insights* e métodos que podem auxiliar os administradores de ambientes em nuvem na tomada de decisões para mitigar problemas de envelhecimento de software nessas infraestruturas, através de monitoramento e ajustes contínuos.

Este estudo expande o campo de pesquisa sobre o envelhecimento e rejuvenescimento

de software em ambientes de computação em nuvem por meio das seguintes contribuições principais:

- Metodologia para medição e avaliação do envelhecimento e rejuvenescimento de software em um ambiente baseado em contêineres com *clusters Kubernetes*;
- Experimentos específicos com o intuito de observar de forma detalhada os fenômenos de envelhecimento de software;
- Realizar a implementação do rejuvenescimento de software em *Clusters Kubernetes (K8S)* com base nos recursos e serviços utilizados e monitorados, a fim de tratar o envelhecimento de forma eficaz.
- Constatações e impactos:
 - Os eventos de envelhecimento de software nos experimentos apontam para possíveis falhas e queda de desempenho nos sistemas, sendo influenciados pelas cargas de trabalho aplicadas e pelas especificações de software dos sistemas operacionais hospedeiros. A ocorrência desses eventos está diretamente ligada à intensidade da carga processada.
 - Se houvesse maior disponibilidade de recursos de hardware ou menores cargas de trabalho nos sistemas usados nos experimentos, o envelhecimento ocorreria de forma mais gradual, adiando as falhas devido ao esgotamento de recursos. No entanto, isso não reduz a importância de avaliar o envelhecimento do software nesses sistemas e planejar ações para mitigá-lo.

1.2 Justificativa

Muitos dos serviços acessados hoje pela Internet são oferecidos aos usuários finais por meio de microsserviços. Empresas como *Google*, *AWS*, *Netflix*, *Linkedin*, *eBay*, *Uber* e *Microsoft* já adotam essa plataforma devido à sua economia, resiliência e alta disponibilidade (PEREIRA; BACK; JÚNIOR, 2022), (VILLAMIZAR et al., 2017), (BLINOWSKI; OJDOWSKA; PRZYBYŁEK, 2022). Falhas nesses sistemas podem resultar em problemas de disponibilidade de serviços para os usuários, além de impactos financeiros, danos à reputação da empresa, custos elevados de manutenção, prejuízos à produtividade e obstáculos ao crescimento (MYINT; THEIN, 2010), (SANTOS et al., 2023).

Como é reconhecido, diversos softwares estão sujeitos ao envelhecimento, e há vários estudos na literatura que abordam o SAR (*Software Aging and Rejuvenation*). No trabalho de Costa et al. (2021), por exemplo, foi realizado um Mapeamento Sistemático da Literatura (MSL) sobre o envelhecimento de software e as tendências de pesquisa em rejuvenescimento. Identificou-se que a virtualização baseada em contêineres e outras tecnologias tornaram-se um

foco recente de alguns esforços de pesquisa em SAR, embora ainda possam existir questões em aberto que merecem investigação.

Para entender a motivação por trás desse estudo, o autor conduziu um Mapeamento Sistemático da Literatura (MSL) para avaliar o envolvimento da comunidade acadêmica em relação ao SAR em plataformas de microsserviços. Verificou-se que, quando se trata do comportamento de soluções baseadas na orquestração de contêineres, ainda existem várias lacunas. Algumas dessas lacunas incluem a investigação de possíveis efeitos colaterais do envelhecimento do software em contêineres, a identificação de ações de rejuvenescimento nos processos das plataformas de orquestração, a exploração de técnicas de previsão para degradação do envelhecimento de software em ambientes de microsserviços e, principalmente, abordagens relacionadas ao rejuvenescimento de softwares que hospedam os microsserviços.

Com base nas descobertas do Mapeamento Sistemático da Literatura (MSL) ([SANTOS et al., 2023](#)) e nas lacunas identificadas, tornou-se evidente que, apesar da proliferação de ambientes containerizados, é crucial explorar o desempenho, a disponibilidade e a confiabilidade dessas plataformas. Quando submetidas a cargas de trabalho intensas e prolongadas, essas plataformas podem sofrer com o envelhecimento. Portanto, é imperativo analisar esses cenários em um ambiente que forneça esses serviços em uma infraestrutura de computação em nuvem real, orquestrada pelo *Kubernetes* — a plataforma mais amplamente utilizada no mercado ([FLORA et al., 2021](#)). O objetivo é minimizar tais efeitos e propor soluções de rejuvenescimento de software, visando reduzir ao máximo a indisponibilidade do ambiente e, assim, contribuir para a comunidade acadêmica.

1.3 Objetivos

Esta seção apresenta o objetivo geral deste trabalho e os respectivos objetivos específicos desta pesquisa.

1.3.1 Objetivo Geral

O principal objetivo deste trabalho é identificar os efeitos do envelhecimento do software e propor estratégias de rejuvenescimento em uma infraestrutura de orquestração de contêineres. A identificação dos efeitos do envelhecimento do software é realizada por meio de abordagens já implementadas e difundidas na área. Além disso, durante o processo de testes, serão monitorados os recursos computacionais, bem como os serviços. Após a constatação do envelhecimento, serão aplicadas estratégias de rejuvenescimento de software.

1.3.2 Objetivos Específicos

Para atingir o objetivo geral deste trabalho, foram definidos os seguintes objetivos específicos:

- Desenvolvimento de estratégias de monitoramento de recursos computacionais e identificação do envelhecimento de software;
- Investigar os efeitos do envelhecimento de software através da adoção de cargas de stress em infraestruturas virtualizadas baseadas em contêineres;
- Avaliar o impacto de diferentes estratégias de rejuvenescimento de software na disponibilidade do sistema.

1.4 Organização do Trabalho

Esta seção apresenta de forma sucinta a divisão das demais partes deste trabalho e como as mesmas estão organizadas e distribuídas, da seguinte forma:

- Capítulo 2 - Fundamentação Teórica, aborda aspectos teóricos desta pesquisa, enfatizando conceitos básicos sobre os temas abordados neste trabalho, como por exemplo: computação em nuvem, contêineres, sistemas de orquestração de contêineres e envelhecimento e rejuvenescimento de software;
- Capítulo 3 - Metodologia descreve o processo metodológico abordado neste estudo e é apresentada em três seções: Visão Geral (Seção 3.1), Pré-avaliação (Seção 3.2) e Avaliação (Seção 3.3).
- Capítulo 4 - Mapeamento Sistemático da Literatura (MSL), todos os artigos que serviram como base para o desenvolvimento deste trabalho. Mostra a investigação e identificação de todos os estudos da literatura sobre envelhecimento e rejuvenescimento de software nos microsserviços, como estão sendo desenvolvidos os experimentos, e como os mesmos estão sendo validados e monitorados;
- Capítulo 5 - Avaliação Experimental, apresenta os detalhes do experimento controlado e resultados obtidos.
- Capítulo 6 - Conclusão, expõe considerações finais sobre as contribuições alcançadas, principais contribuições na Seção 6.1 e trabalhos futuros na Seção 6.2.

2

Fundamentação Teórica

Este Capítulo apresenta conceitos importantes para o entendimento deste trabalho, com o propósito de proporcionar maiores detalhes e trazer mais clareza sobre os temas abordados.

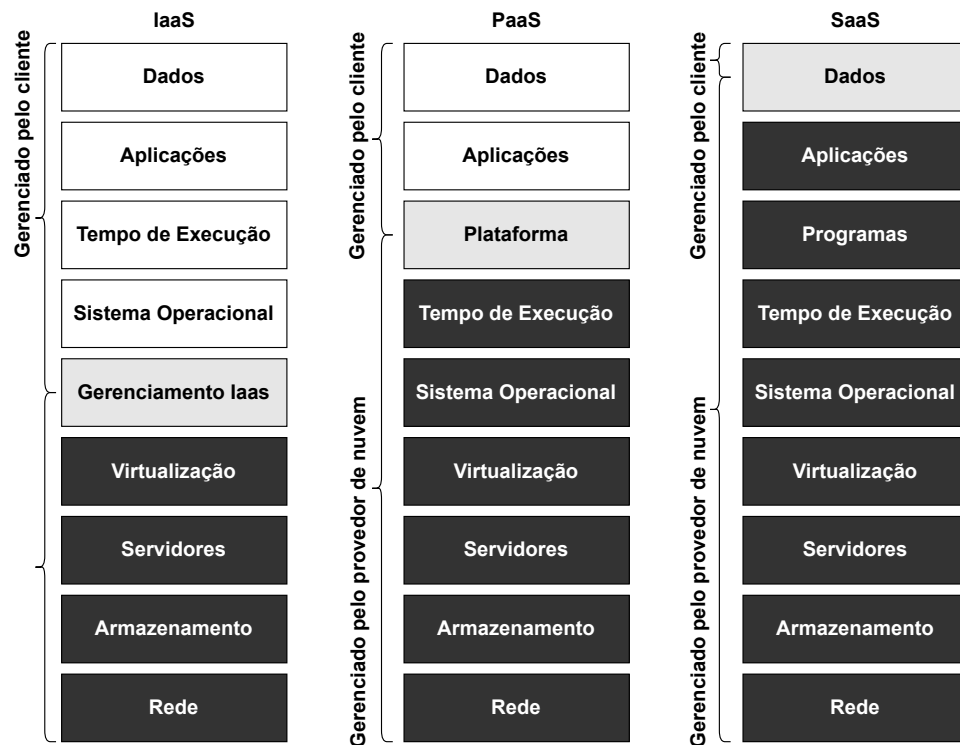
2.1 *Cloud Computing System*

Os sistemas de computação em nuvem oferecem serviços como *Software as a Service (SaaS)*, *Platform as a Service (PaaS)* e *Infrastructure as a Service (IaaS)*. No modelo IaaS, os serviços são disponibilizados como uma infraestrutura completa de computação; no PaaS, os clientes utilizam a plataforma para desenvolver aplicativos de software; enquanto no SaaS, os usuários acessam e usam o software pela Internet, interagindo com os provedores de serviços, conforme ilustrado na Figura 1.

As plataformas de nuvem gerenciam e utilizam recursos de *data center* sob demanda. Essas arquiteturas baseadas em nuvem têm crescido consideravelmente e já se tornaram uma realidade ([PAPAKOSTAS; RAMASUBRAMANIAN, 2022](#)). Isso se deve principalmente ao fato de oferecerem sistemas de computação altamente escaláveis, distribuídos e operacionais em ambientes compartilhados e automatizados ([MONDAL; SABYASACHI; MUPPALA, 2017](#); [OLIVEIRA et al., 2021](#)).

No contexto de computação em nuvem, essa escalabilidade é fornecida pela virtualização, onde as aplicações são executadas em máquinas virtuais (VMs) e entregues ao usuário final em um ambiente isolado. No entanto, uma considerável sobrecarga de virtualização tradicional de hardware parece ser inevitável ([MARTIN, 2022](#)). Dessa forma, os contêineres ganharam mercado por possuírem características de leveza em sua execução, com economia no consumo dos recursos de hardware e software.

Figura 1 – Arquitetura da Computação em Nuvem



Fonte: Adaptado de [Surbiryala e Rong \(2019\)](#)

2.2 Contêineres

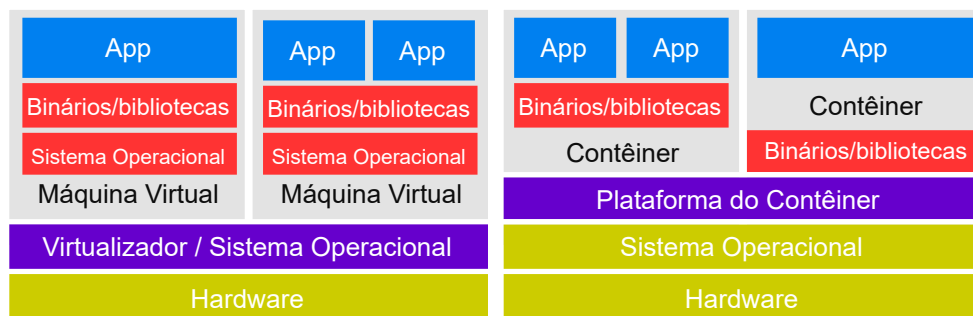
A história dos contêineres remonta a cerca de 1950 ([RANDAL, 2020](#)). Ao longo do tempo, várias iniciativas surgiram, e entre as principais soluções de contêineres estão o CoreOS rkt, Mesos Containerizer, OpenVZ, crio-d, LXC, LXD, Singularity, Kata Containers, gVisor, Containerd e Docker ([FLAUZAC; MAUHOURAT; NOLOT, 2020](#)). No entanto, é reconhecido que a era moderna dos contêineres teve início em 2013 com a introdução do Docker ([IBM, 2022](#)), um ecossistema de código aberto para criar, implantar e gerenciar aplicativos. Um contêiner representa uma unidade padronizada de software que empacota o código e todas as suas dependências para possibilitar a execução rápida e confiável de um ambiente de computação para outro ([Docker Inc., 2023](#)).

A arquitetura de containerização oferece uma abordagem leve e ágil à virtualização, dispensando a necessidade de um hypervisor. Isso resulta em maior agilidade no provisionamento de recursos e acesso mais rápido a novos softwares ([MARTIN, 2022](#)). Em suma, um contêiner é um ambiente de execução independente que compartilha o *kernel* do sistema *host* e pode ser isolado de outros contêineres no mesmo sistema ([CABODI; GIORGIO; MINÌ, 2020](#)).

Os contêineres solucionam os problemas inerentes à virtualização tradicional mencionados por ([MARTIN, 2022](#)), como a sobrecarga inevitável na virtualização tradicional de hardware. Além disso, a problemática de cada instância de VM usando seu próprio sistema operacional,

sistema de arquivos, binário e biblioteca é mitigada por essa plataforma. Com a containerização, não é mais necessário um virtualizador para intermediar a comunicação com o hardware. A camada de aplicativos é empacotada de forma independente e pronta para implantação, podendo compartilhar binários e bibliotecas ou não, eliminando a necessidade do papel do virtualizador. Dessa forma, uma camada é removida da arquitetura, permitindo que a comunicação ocorra diretamente entre o sistema operacional (instalado no *bare metal*) e o hardware. A Figura 2 ilustra a diferença entre essas duas arquiteturas mencionadas.

Figura 2 – Arquitetura de Virtualização de VM à Esquerda e de Containerização à Direita



Adaptado de Pahl (2015)

Quando comparado com a virtualização, a containerização tem como principais vantagens a simplificação de requisitos de recursos e a indiscutível eficiência na implantação dos contêineres (MORAVCIK et al., 2022). Dessa forma, quando criado, ele poderá ser implantado com muita facilidade. Além disso, o aplicativo executado dentro do contêiner não dependerá de um sistema operacional exclusivo para funcionar.

2.3 Sistemas de Orquestração

Com uma arquitetura virtualizada, leve e aprimorada, muitas empresas em todo o mundo têm implementado o uso de contêineres em suas infraestruturas, seja em ambientes on-premise ou na nuvem. A necessidade de agrupar várias tarefas automatizadas e suas configurações em grupos de sistemas ou máquinas está em ascensão. É nesse ponto que a orquestração pode ser valiosa. A orquestração é uma plataforma de código aberto altamente adaptável, voltada para gerenciar soluções e serviços em contêineres, simplificando a configuração por meio de automação e declarações concisas. Com um ecossistema abrangente em constante evolução, atende às demandas variadas de inúmeras organizações (MORAVCIK et al., 2022). Além disso, ela colabora com as equipes de TI ao supervisionar e gerenciar tarefas, transformando fluxos de trabalho complexos em atividades mais acessíveis e simplificadas.

Atualmente, existem várias plataformas de orquestração de contêineres amplamente utilizadas, incluindo Elastic Container Service, Mesosphere, Docker Swarm, AWS, OKD, Microsoft Azure, Rancher, Nomad, HashiCorp e Kubernetes (K8s). Segundo Flora et al. (2021),

este último é utilizado por 78% das empresas que adotaram microsserviços em seu cotidiano. A aceitação do Kubernetes forçou sua rápida maturação e estimulou a comunidade tecnológica a inovar rapidamente (Lawrence E Hecht, 2018).

O Kubernetes (K8s) é um produto de código aberto utilizado para automatizar a implantação, escalonamento e gerenciamento de aplicativos em contêineres. É uma plataforma facilmente portátil e expansível, usada para gerenciar soluções e serviços de contêineres. Facilita a configuração declarativa e a automação, possuindo um ecossistema amplo e em constante evolução que oferece suporte a diversas organizações (The Kubernetes, 2023).

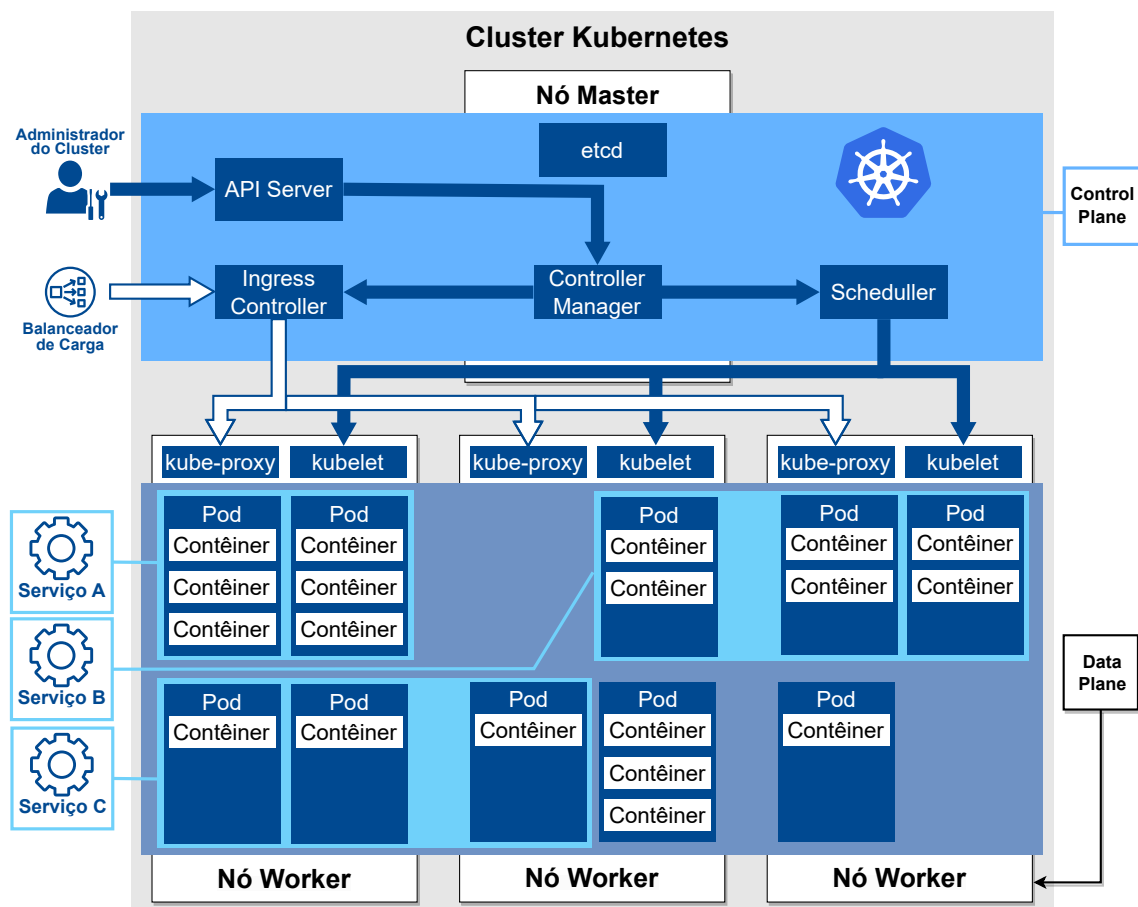
Em um breve histórico da evolução do Kubernetes, informações de (Risingstack, 2022) revelam que o desenvolvimento do Kubernetes remonta a 2003-2004, quando o Google introduziu o sistema Borg (colaborativo com um novo mecanismo de busca). Em 2013, o Google lançou o programa Omega (para expandir a capacidade se o *Cluster* consistisse em vários dispositivos). Em 2014, o Google avançou com o programa Borg, lançando o Kubernetes como uma versão de código aberto. A partir desse ponto, empresas como Microsoft, Red Hat, IBM e Docker juntaram-se à comunidade Kubernetes. No ano seguinte, foram lançadas as versões 1.0 e 1.1 do Kubernetes. Em 2016, o Kubernetes começou a se popularizar. Helm (um gerenciador de pacotes para Kubernetes) e Minikube (uma versão local tudo-em-um do Cluster Kubernetes) foram lançados, e a versão 1.5 foi introduzida. Em 2017, o Kubernetes foi implantado comercialmente e testado em várias organizações. Em 2018, a primeira versão beta do Kubernetes foi lançada, ganhando consistência e integração com diversas plataformas de nuvem. Hoje, o Kubernetes está na versão v1.27 e continua crescendo rapidamente.

Conforme pode ser visto na Figura 3, a arquitetura do *Cluster* Kubernetes é composta por:

- *Nó Master* - Responsável por executar os dados de controle e gerenciamento;
- *Control Plane* - Responsável por gerenciar o *cluster*, divide as tarefas com os seguintes componentes:
 - *API Server* - Responsável por estabelecer a comunicação entre pontos do Kubernetes;
 - *etcd*, banco de dados;
 - *Schedule* - Responsável por distribuir e definir em qual nó são executadas as requisições;
 - *Ingress Controller* - Responsável por cumprir o *Ingress*, geralmente com um balanceador de carga;
 - *Controller Manager* - Responsável por gerenciar todos os controladores do Kubernetes (*replicaset*, *admission controller*, *authentication controller*, *authorization controller*).
- *Nó Worker* - Responsável por executar as cargas de trabalho;

- *Data Plane* - Responsável pela parte operacional, funciona dentro dos *nodes workers* ;
 - *Kubelet* - Responsável por estabelecer a comunicação com o *API Server* para e gerenciar o *node*;
 - *Kube proxy* - Responsável por gerenciar a rede dos *nodes*;
 - *Pods* - Local onde os *containers* são hospedados e executados.

Figura 3 – Arquitetura do Cluster Kubernetes



Fonte: Adaptada de (Gartner, 2022)

2.4 Envelhecimento e Rejuvenescimento de Software

Em sistemas computacionais, a qualidade do serviço fornecido por uma aplicação e a confiança depositada no serviço fornecido são essenciais. A grande maioria dos sistemas acessados hoje em dia, são disponibilizados a partir de plataformas de computação de nuvem, onde essas aplicações precisam estar disponíveis 24 horas por dia e podem ser acessadas a qualquer momento. Essa disponibilidade é necessária já que as consequências de falhas de software podem causar grandes prejuízos econômicos e de reputação (AVIZIENIS et al., 2004).

Um dos maiores desafios para a disponibilidade dos sistemas é o fenômeno conhecido como envelhecimento do software. Geralmente, o envelhecimento do software é consequência de falhas de software (GROTTKE; MATIAS; TRIVEDI, 2008), sendo um processo inevitável em que os processos de aplicativos sofrem degradação de desempenho ao longo de sua utilização. Para Schroeder e Gibson (2009), Zheng, Okamura e Dohi (2020), esse fenômeno é desencadeado por falhas relacionadas ao envelhecimento, como acúmulo de erros durante a execução, levando eventualmente a falhas ou travamentos do sistema. Esse fenômeno pode afetar o sistema de diversas maneiras, como vazamento de recursos, fragmentação, corrupção e inconsistência de dados, ou erros numéricos (GROTTKE; MATIAS; TRIVEDI, 2008).

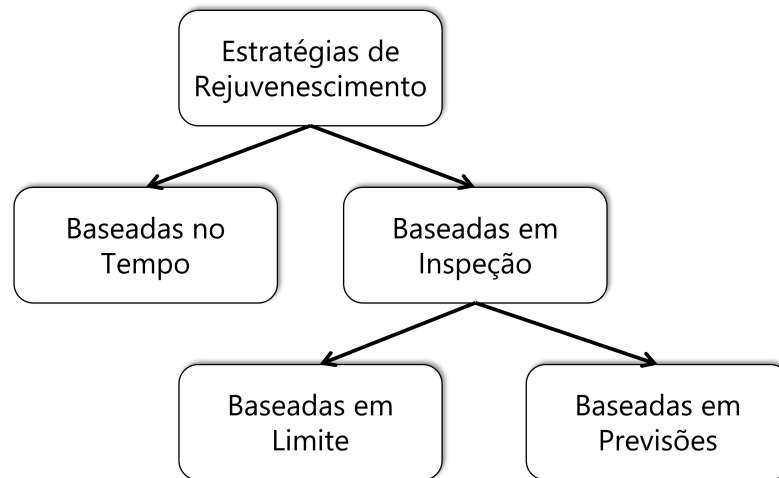
A fase de testes do software, embora detecte muitos erros ou *bugs*, pode não prever todas as falhas que surgem durante a produção, quando o software é executado continuamente por longos períodos, desencadeando o envelhecimento. Alguns softwares tendem a envelhecer à medida que é executado, acumulando potenciais falhas desde o início de sua operação (GARG et al., 1995). Este fenômeno afeta principalmente a disponibilidade e o desempenho dos sistemas (ARAUJO, 2017). Além disso, quanto mais componentes de software, maior o risco de falhas causadas pelo envelhecimento. O envelhecimento de software é um tema que vem sendo tratado na literatura nos últimos 25 anos (COSTA et al., 2021).

Para minimizar os efeitos do envelhecimento, existem ações proativas chamadas de rejuvenescimento de software. Essa prática consiste em um conjunto de ações ou técnicas aplicadas a um software afetado pelo envelhecimento (TORQUATO; ARAUJO; MACIEL, 2015), visando remover os efeitos acumulados do envelhecimento (ARAUJO, 2017). O rejuvenescimento de software é a maneira mais comum de combater o envelhecimento, podendo, em situações extremas, envolver até a reinicialização do hardware (TRIVEDI; VAIDYANATHAN; GOSEVA-POPSTOJANOVA, 2000). No entanto, essa prática pode ser custosa devido à possível perda de requisições ou à indisponibilidade temporária do serviço (ARAUJO, 2017).

Definir políticas de rejuvenescimento é crucial para maximizar a disponibilidade do sistema, porém, encontrar o momento ideal para essa ação é desafiador. Realizar o rejuvenescimento muito cedo ou tarde demais pode não ter o efeito desejado (TANG et al., 2020; LEVITIN; XING; BEN-HAIM, 2018). Diversas técnicas são estudadas, como descontinuar aplicações afetadas, reiniciar sistemas operacionais, máquinas virtuais ou aplicações de maneira independente (ALONSO et al., 2013). O rejuvenescimento ideal deve ser proativo, como reinicializações manuais ou automáticas em momentos específicos (COTRONEO et al., 2010). Alguns sistemas aplicam rejuvenescimento periódico aos seus componentes (CARNEVALI et al., 2021). Essas ações planejadas podem ser acionadas por estratégias baseadas no tempo, limiar ou predição (ARAUJO et al., 2011; ARAUJO, 2012), conforme ilustrado na Figura 4.

No contexto específico dos contêineres, há necessidade de estudos mais aprofundados sobre envelhecimento e rejuvenescimento de software, buscando métodos mais otimizados para antecipar falhas relacionadas ao envelhecimento. Isso viabilizaria a implementação de ações

Figura 4 – Técnicas de Rejuvenescimento de Software



Fonte: Adaptado de [Araujo et al. \(2011\)](#)

preventivas para aplicar o rejuvenescimento do software, melhorando a disponibilidade do sistema hospedeiro ou da aplicação.

Em relação aos microsserviços, o SAR foi explorado por 8 estudos experimentais. Estes serão discutidos no Capítulo 4.

3

Metodologia

Este Capítulo apresenta a metodologia que foi utilizada para o desenvolvimento desta dissertação. O processo metodológico está dividido nas seguintes seções: Visão Geral (Seção 3.1), Pré-Avaliação (Seção 3.2) e Avaliação (Seção 3.3).

3.1 Visão Geral

Esta seção oferece uma visão geral do processo metodológico empregado, baseada na metodologia de suporte estabelecida por [ARAUJO \(2017\)](#). Esse processo consiste em duas macro-atividades: **Pré-avaliação** e **Avaliação**.

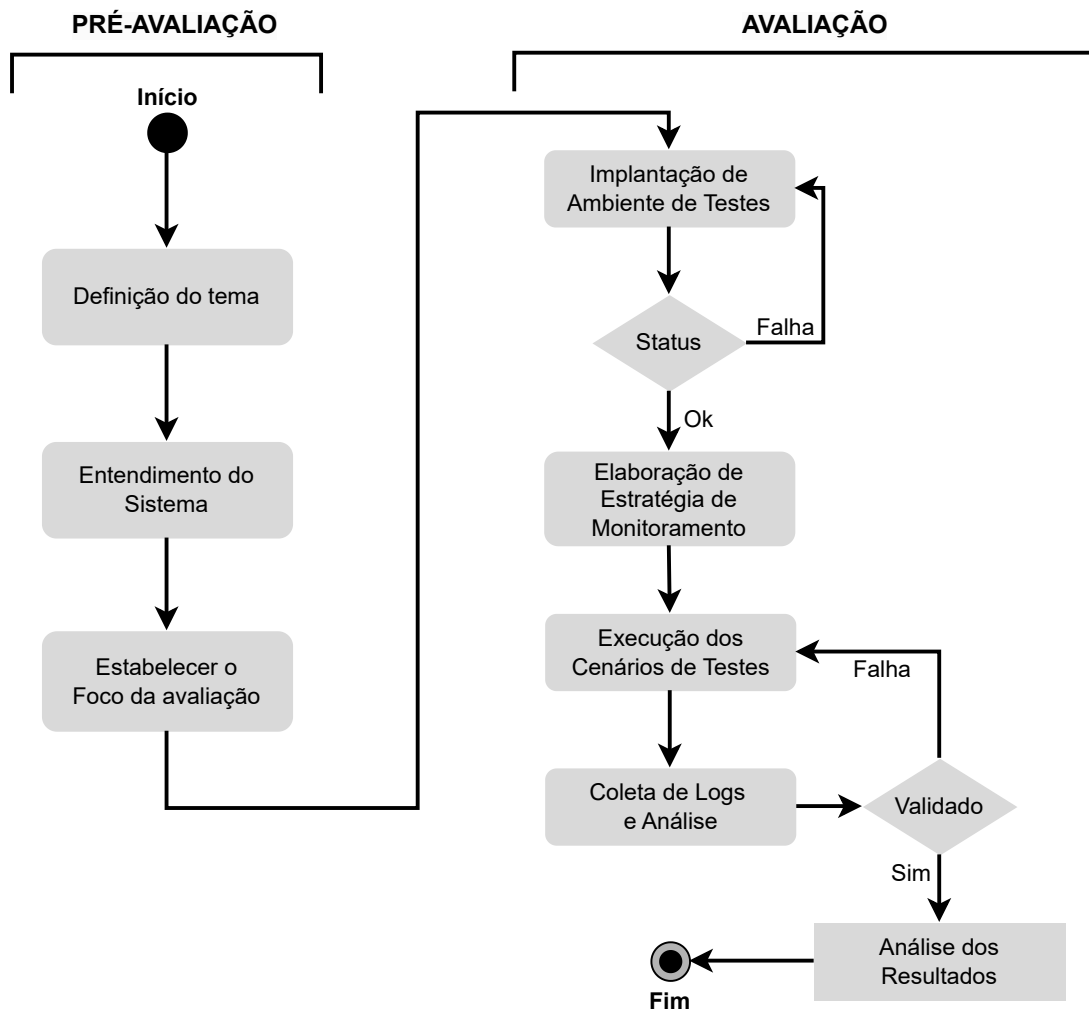
A **Pré-avaliação** está dividida em três fases, a saber: Definição do Tema; Entendimento do Sistema; Estabelecimento do Foco da Avaliação.

Já a **Avaliação** está dividida em: Implantação de Ambiente de Testes; Elaboração de Estratégias de Monitoramento; Execução dos Cenários de Testes; Coleta de *Logs* e Análise; Análise dos Resultados.

O diagrama de fluxo apresentado na Figura 5 oferece uma representação visual da metodologia de apoio adotada no planejamento. Cada etapa da metodologia é representada por retângulos, obedecendo à ordem de execução indicada pelas setas. O avaliador avança para a próxima etapa somente quando a etapa atual é concluída. O losango representa uma etapa que pode seguir por dois caminhos diferentes, dependendo do resultado obtido.

A seguir, cada atividade da metodologia de apoio é detalhadamente descrita.

Figura 5 – Metodologia do Experimento



Fonte: O autor

3.2 Pré-avaliação

Nesta seção, são apresentadas as quatro atividades que compõem a Pré-Avaliação: O propósito neste momento é apresentar os insumos necessários para a subsequente implementação da etapa de Avaliação.

- **Definição do tema**

A fase inicial, que consistiu na definição do tema, orientou a escolha da área de pesquisa. Um requisito fundamental foi optar por um tema que ainda não fosse extensivamente explorado, mas que também fosse de grande relevância, tanto em um contexto de mercado tecnológico quanto no contexto acadêmico.

- **Entendimento do Sistema**

Na etapa de entendimento do sistema, é essencial obter um conhecimento abrangente sobre a literatura. Para alcançar esse objetivo, procedemos com a contextualização por

meio da realização de um Mapeamento Sistemático da Literatura (MSL), seguindo o protocolo proposto por (KITCHENHAM, 2004a). Durante esse processo, diversas questões foram levantadas para evidenciar o envolvimento da comunidade acadêmica nos estudos sobre o envelhecimento e rejuvenescimento de software no contexto dos microsserviços. Essas questões foram respondidas e diversos dados foram coletados, conforme apresentado na Tabela 1, identificando, assim, lacunas ainda não exploradas. Após a conclusão do MSL, buscou-se um conhecimento mais aprofundado do sistema, abrangendo sua arquitetura, componentes, processos, interações e interoperabilidade, com um foco especial nas plataformas de containerização, operação dos contêineres e no *Kubernetes* (K8s). É fundamental possuir competências básicas em serviços de nuvem, uma sólida experiência em ambientes Linux e uma base sólida de conhecimento em plataformas de microsserviços e sistemas de orquestração. Esse conhecimento é crucial para orientar o planejamento da infraestrutura que foi implementada, proporcionando também uma oportunidade valiosa para pesquisar e adotar as melhores práticas na instalação e configuração dos *Clusters*.

- **Estabelecer o Foco da Avaliação**

Após uma análise aprofundada do sistema, crucial delinear o foco da avaliação, com base principalmente nas lacunas identificadas durante o MSL. Nessa etapa, torna-se imperativo estabelecer o objetivo central da avaliação experimental planejada. O avaliador precisa determinar quais cenários e métricas são prioritários, concentrando-se especialmente naqueles que exercem maior impacto nos níveis de qualidade do serviço prestado. Por exemplo, em um experimento contendo 5 cenários, no cenário 1, o foco será o envelhecimento de software investigando possíveis falhas e consumos excessivos quando utilizada a carga máxima que o *Cluster* suporta, enquanto nos 4 demais cenários, o foco está voltado para estratégias de rejuvenescimento tentando mitigar ou antecipar as possíveis falhas encontradas no envelhecimento. Além disso, é crucial planejar o hardware que será utilizado e determinar por quanto tempo estará disponível.

3.3 Avaliação

Nesta seção, são apresentadas as cinco atividades que compõem a avaliação. O propósito nesta etapa é empregar todo o conhecimento adquirido e o material desenvolvido nas fases anteriores para a realização uma avaliação efetiva do sistema.

- **Implantação de Ambiente de Testes**

Após a definição dos requisitos da **Pré-Avaliação**, inicia-se a fase de Instalação e Preparação do Ambiente, tendo como objetivo de configurar os *Clusters* do *Kubernetes* para a execução dos *scripts* e, assim, realizar o experimento propriamente dito. Nesta fase, ocorre um refinamento nos sistemas operacionais e no *container runtime* para implementar o *Cluster*.

Ao concluir a Pré-Avaliação, já se obtém conhecimento da arquitetura do *Kubernetes*, e qualquer erro que venha a ocorrer pode ser corrigido com a experiência adquirida, permitindo avançar para a conclusão desta etapa. Antes de prosseguir para a próxima fase, é essencial verificar o **status** de todo o ambiente, garantindo seu correto funcionamento e atestando a plena capacidade de operação das rotinas a serem programadas, além de compreender o limite de operação dos *clusters*. Em caso de detecção de qualquer comportamento inesperado, todo o processo de instalação seria corrigido ou refeito.

- **Elaboração de Estratégia de Monitoramento**

Com a conclusão da fase de instalação dos *Clusters*, inicia-se a etapa de elaboração dos *scripts* e monitoramento. Baseando-se nos artigos explorados no MSL e no conhecimento adquirido sobre o sistema, os *scripts* são desenvolvidos ou redesenhados e testados no ambiente, buscando aprimoramentos para otimizar a coleta de *logs*. Nessa fase, são testados tanto os *scripts* de carga e remoção de Pods quanto os recursos de hardware e serviços escolhidos para serem monitorados.

- **Execução dos Cenários de Testes**

Com as fases anteriores concluídas, nesta etapa, os cenários dos experimentos são iniciados, pois foram previamente planejados na seção 3.2, no item **estabelecer o foco da avaliação**. Cada um dos cenários planejados é executado, e os *logs* são salvos e armazenados em tempo real. Uma análise detalhada de cada cenário proposto nos experimentos será apresentada de forma mais aprofundada no Capítulo 5.

- **Coleta de *logs* e Análise**

Com o término dos cenários de teste, os *logs* são extraídos do ambiente, transferidos para fora do *cluster*, plotados em gráficos e analisados. Nessa etapa, compreende-se como os recursos, tanto de hardware quanto de software, influenciam o funcionamento do ambiente estudado. Antes de prosseguir para a próxima fase, é essencial verificar o **status** dos *logs* salvos, garantindo que ocorreu o perfeito funcionamento e atestando que os *logs* foram salvos. Em caso de detecção de qualquer comportamento inesperado, eventual falha ou problemas na coleta de *logs*, todo o processo de execução do cenário é refeito. Se tudo estiver em conformidade com o planejado, avança-se para a próxima fase.

- **Análise dos Resultados**

Ao verificar a abrangência correta dos resultados na validação da coleta e análise dos dados, faz-se necessário apresentar os dados coletados durante os experimentos de forma clara e organizada, descrevendo as principais tendências e padrões observados nos resultados, identificando quais métricas foram mais relevantes para avaliar o desempenho do sistema, e discutindo eventuais desvios ou anomalias nos resultados. Além disso, é necessário avaliar o sucesso das estratégias tanto no envelhecimento quanto no rejuvenescimento de

software que foram implementadas, destacando seus pontos fortes e áreas de melhoria. Também é importante considerar as limitações dos experimentos e suas implicações para a interpretação dos resultados, bem como oferecer insights sobre como os resultados podem orientar futuras pesquisas na área.

4

Mapeamento Sistemático da Literatura

Este Capítulo apresenta o Mapeamento Sistemático da Literatura (MSL) que serviu como base para o desenvolvimento deste trabalho. Este MSL foi realizado seguindo a metodologia definida por ([KITCHENHAM, 2004b](#)) (Figura 6), com o intuito de identificar estudos sobre o envelhecimento e rejuvenescimento de software em microserviços. Para isso, foram desenvolvidas questões de pesquisa (QPs) com o propósito de detalhar uma visão geral da área, afim de, evidenciar e selecionar lacunas a partir dos estudos primários encontrados.

4.1 Metodologia

Um Mapeamento Sistemático da Literatura (MSL) é um estudo secundário, baseado em evidências, que tem como proposta o levantamento, a identificação, avaliação e classificação da literatura para responder questões de pesquisa específicas ([KITCHENHAM, 2004b](#)). O objetivo do estudo secundário é prover a pesquisadores uma visão geral de uma área de pesquisa e ajudar a identificar lacunas nessa área ([WOHLIN et al., 2013](#)).

O MSL foi utilizado como metodologia de pesquisa, tendo como finalidade específica de encontrar estudos primários, além de uma visão geral sobre esses estudos com foco em uma determinada área e com finalidade de identificação e classificação desses estudos ([PETERSEN; VAKKALANKA; KUZNIARZ, 2015](#)). Este MSL foi desenvolvida em três fases. São elas: Planejamento, Condução e Relatório Final da Revisão (Figura 6).

Na Seção a seguir são descritos os mecanismos metodológicos utilizados para nortear este estudo. Para isso, foram desenvolvidas questões de pesquisa (QPs) com proposito de detalhar uma visão geral da área, afim de, evidenciar e selecionar os estudos primários. Desta forma, torna-se possível a avaliação e repetição da revisão por outros pesquisadores.

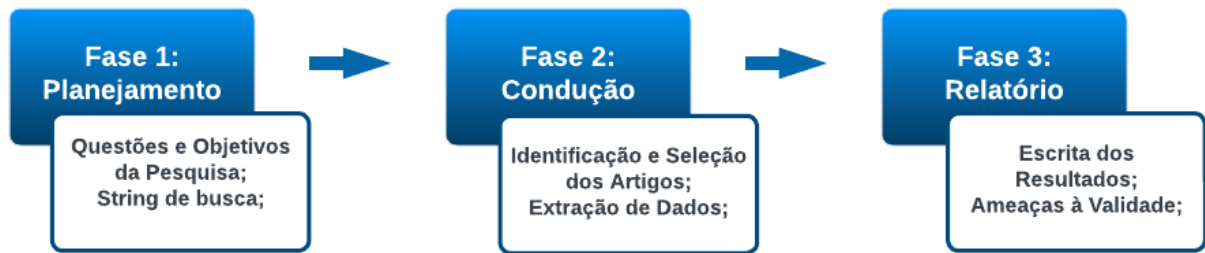


Figura 6 – Fases do MSL.

4.1.1 Planejamento

O planejamento se iniciou com a definição do objetivo e das questões de pesquisa. Em seguida, a estratégia de busca foi construída, determinando as bases de pesquisa, os termos de busca e os critérios de inclusão e exclusão para a seleção dos artigos.

4.1.2 Propósito e Questões de Pesquisa (QPs)

Esse mapeamento sistemático, tem como objetivo levantar e avaliar como estão sendo desenvolvidos os estudos sobre os efeitos do envelhecimento de software nos microsserviços, como os autores estão aplicando ações de rejuvenescimento, além de como tais efeitos estão sendo validados e monitorados. Para tanto, serão usados artigos pré-mapeados com a finalidade de responder as questões e os termos da pesquisa (JAMES; RANDALL; HADDAWAY, 2016).

Para alcançar esse objetivo foram definidas as seguintes questões de pesquisa:

- QP1: Quais são os autores mais produtivos nesta área?
- QP2: Quais universidades contribuem mais para os artigos mencionados?
- QP3: Em quais continentes foram conduzidas pesquisas sobre o envelhecimento e rejuvenescimento de software em microsserviços?
- QP4: Quais são as plataformas e orquestradores de microsserviços mais utilizados neste MSL?
- QP5: Quais métodos ou técnicas são utilizados para forçar o envelhecimento e rejuvenescimento de microsserviços?
- QP6: Do ponto de vista da monitorização, quais recursos foram monitorados nos artigos explorados?

4.1.3 Base de Dados

As bases de dados utilizadas nesse estudo foram a *ACM Digital Library*, *IEEE*, *MDPI*, *Scopus* e *SpringerLink*. As mesmas foram utilizadas pelo fato de indexarem as principais bases de

dados científicos, além de seus motores de busca automatizados possuírem uma maior eficiência em encontrar estudos relevantes.

4.1.4 Termos de Busca

Após a definição das bases de dados para encontrar os estudos primários, foi definida a *string* de busca, com finalidade de identificação e classificação desses estudos, conforme pode ser visto na Figura 7. Nas quatro bases de dados utilizadas foram empregados recursos de busca avançada, com o propósito de encontrar as palavras-chave pesquisadas contidas nos títulos e/ou nos resumos do material prospectado.

**(software aging OR software rejuvenation OR SAR)
AND
(container OR docker OR lxc OR openvz OR microservice
OR kubernetes OR k3s OR k8s OR minikube)**

Figura 7 – *String* de busca

4.1.5 Critérios de Inclusão e Exclusão

Os critérios de inclusão e exclusão são utilizados para delimitar estudos mais relevantes que respondam ou não as questões de pesquisa.

- CI1: Publicações focadas no estudo do envelhecimento e rejuvenescimento de software em microsserviços;
- CI2: Publicações que contribuem para responder pelo menos uma das QPs;
- CI3: Trabalhos publicados entre os anos de 2006 e 2023;
- CI4: Artigos escritos em português ou inglês;

4.1.6 Condução

A busca dos trabalhos foi executada no início do primeiro semestre de 2023. A quantidade de artigos identificados, o método de filtragem e a extração dos dados são detalhados a seguir.

Inicialmente foi buscado encontrar o maior número de artigos possíveis sobre o tema proposto, assim foram encontrados 39 na *ACM Digital Library*, 111 artigos na base do *IEEE*, 1 artigo no *MDPI*, 34 artigos na base do *Scopus* e 48 artigos na *SpringerLink*, totalizando 233 artigos. Após critérios de exclusão (CE), atentando-se ao objetivo do trabalho e à sua conclusão, foram realizadas leituras dos títulos dos artigos e resumos, além de filtragem dos que estavam em

duplicidade. Atentando-se aos sete artigos que sobraram, esses servirão de base para responder aos questionamentos deste mapeamento, descritos na Subseção 4.1.2.

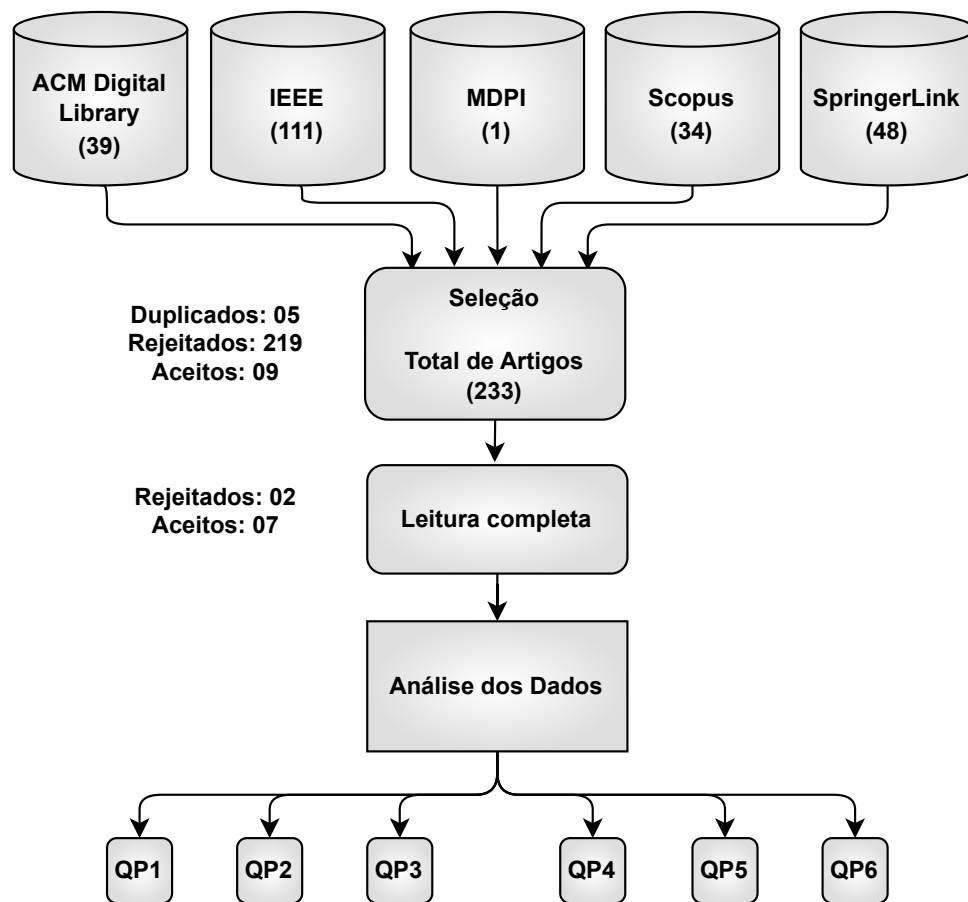


Figura 8 – Processo de Seleção de Artigos.

4.1.7 Relatório

Ao final de todo o processo metodológico do MSL, foram considerados sete artigos como relevantes. Uma síntese de cada um desses trabalhos é apresentada na Subseção 4.2.

Foram considerados 7 artigos relevantes ao final de todo o processo metodológico do MSL. A Seção 4.2 fornece um resumo de cada um dos estudos selecionados, delineando os passos experimentais realizados junto com seus respectivos cronogramas e resultados. Além disso, destaca como o monitoramento foi incorporado a cada experimento.

A pesquisa revelou a importância de discutir o SAR por meio da análise dos artigos. Também demonstrou até que ponto esse tópico tem sido estudado globalmente e identificou as lacunas restantes que precisam de uma exploração mais aprofundada.

4.2 Estado da Arte

A Tabela 1 resume as descobertas de nossa análise de sete artigos. Inclui informações como os autores e ano de publicação, as plataformas e orquestradores usados nos experimentos, os países onde os experimentos foram realizados, a duração dos experimentos, os métodos e técnicas empregados e o tipo de monitoramento utilizado.

Tabela 1 – Resumo do Estado da Arte.

AUTOR	PLATAFORMA/ORQUESTRADOR	PAÍS	TEMPO DO EXPERIMENTO	MÉTODOS E TÉCNICAS	MONITORAMENTO
(MATIAS; FILHO, 2006)	Docker / sem orquestrador	Brasil Portugal	26 dias	SWARE (Stress, Wait and Rejuvenation) i) S - ambiente de estresse com a carga de trabalho acelerada para induzir a ativação de bugs; (ii) W - interromper o envio da carga de trabalho para observar possíveis acumulados efeitos e; (iii) R - submeter uma ação de rejuvenescimento (reinicialização do SO) a perceber mudanças no estado interno do software.	mpstat, sar e pidstat
(OLIVEIRA et al., 2020)	Docker / sem orquestrador	Brasil	25 dias 23 dias	Compreensão do sistema; Planejamento de medição; Planejamento de carga de trabalho; Experimento estressante e análise de envelhecimento;	Comandos - free, ps, df Sistema de arquivos /pro Ferramenta - SystemTap
(YUE; WU; XUE, 2020)	Docker / sem orquestrador	China	30 dias	Utilização de recursos; Latência por camada; Número de solicitações pendentes;	cAdvisor
(FLORA et al., 2021)	Docker /Kubernetes	Portugal	02 dias 02 dias 01 dia	Operação normal; Operação com testes de estresse;; Operação com a utilização de uma falha relacionada ao envelhecimento;	Kubernetes Metrics; Events startup probe; Readiness probe; Liveness probe;
(OLIVEIRA et al., 2021)	Docker /Docker Swarm	Brasil	25 dias	Compreensão do sistema; Planejamento de medição; Planejamento de carga de trabalho; Experimento estressante e análise de envelhecimento;	Comandos - free, ps, df Sistema de arquivos /pro
(VINÍCIUS et al., 2022)	Docker / sem orquestrador	Brasil	30 dias	Investigação e avaliação de métricas em período específico; Stress-Wait-Rejuvenation (SWARE); Modelos de previsão usando os dados coletados com uma comparação;	Comandos free -mh, top e grep
(COSTA et al., 2023)	Docker / Kubernetes	Brazil Coreia do Sul	6 days 4 days	Full process monitoring, Cluster startup; High workload and orchestrator shutdown.	Commands: free e pidstat
Estudo atual	Containerd / Kubernetes	Brasil	10 days 10 days 10 days 10 days 10 days	Process monitoring, Software aging effects; Process monitoring, software aging, and software rejuvenation (cleaning caches and buffers); Process monitoring, software aging, and software rejuvenation (containerd service restart); Process monitoring, software aging, and software rejuvenation (kubelet service restart); Process monitoring, software aging, and Software rejuvenation (controlled reboots of Workers nodes).	Commands: free, /proc/meminfo, df, top, pidof, pidstat

An Experimental Study of Software Aging and Rejuvenation in dockerd

Nessa Pesquisa de [Torquato e Vieira \(2019\)](#), os autores empregaram a técnica *Stress, Wait e Rejuvenation (SWARE)* no *daemon* utilizando a imagem pública *httpd* para realizar cargas de trabalho simples com um *loop* de estresse enquanto criavam e excluíaam contêineres. Foram conduzidos dois experimentos com 50 e 2 contêineres, respectivamente. Para evitar picos de sobrecarga, foi imposto um atraso de 10 segundos após a criação dos contêineres e um atraso de 50 segundos após a exclusão deles. Ambos os experimentos observaram o envelhecimento e a falha catastrófica durante a fase de Estresse. Para recuperar e rejuvenescer o software, os autores deletaram o conteúdo da sobreposição, contêineres e pastas. Eles também purgaram quaisquer contêineres remanescentes e utilizaram o comando *docker system prune* para limpar o *Docker*. Por fim, reiniciaram o sistema operacional. Os testes duraram 26 dias, e o monitoramento foi feito usando comandos como *mpstat*, *sar* e *pidstat*.

Experimental evaluation of software aging effects in a container-based virtualization platform

No artigo de [Oliveira et al. \(2020\)](#), foi utilizado um processo de 5 etapas, que incluiu compreensão do sistema, planejamento de medição, planejamento da carga de trabalho, experimento de estresse e análise de envelhecimento. Os autores exploraram 2 cenários para o mesmo hardware. Para o cenário 1, os autores utilizaram o servidor *Apache* com um limite de uso de memória de 256 MB para simular o *IaaS*. Em contraste, para o cenário 2, eles padronizaram a imagem do *Debian* sem limitar a memória, visando analisar o desempenho de alguns dos principais componentes do *Docker*, como *dockerd*, *containerd* e *Network Manager*. No primeiro cenário, um *loop* foi gerado para criar 20 contêineres e verificar se o tempo de execução deles alcançava 120 segundos; caso contrário, eles eram reiniciados e aguardavam 5 segundos antes da próxima iteração. Quando os contêineres tinham sido executados por 120 segundos, eles eram parados e removidos. No segundo cenário, o experimento visava possíveis efeitos em contêineres com tempos de execução mais longos, e para isso, um *script* criava 5 contêineres persistentes a cada 5 segundos, e em cada contêiner, era utilizado o *docker exec* como recurso. Em ambos os cenários, foi observado o envelhecimento do software (*NetworkManager* e *dockerd*). Os testes foram executados continuamente por 25 dias no cenário 1 e 23 dias no cenário 2. O sistema foi monitorado usando *scripts bash* em ambos os cenários. Utilitários do Linux como *free*, *ps*, *df* e o sistema de arquivos */proc* foram utilizados, além da ferramenta *SystemTap*.

Microservice Aging and Rejuvenation

No Pesquisa de [Yue, Wu e Xue \(2020\)](#), foi proposto um método baseado em aprendizado profundo para prever o envelhecimento de microsserviços usando parâmetros de entrada como utilização de recursos, latência por camada e número de solicitações pendentes. A camada de saída gera uma probabilidade de violações de QoS (Qualidade de Serviço), definida de 0 a 9 (com uma variação de 10% para cada unidade), para cada microsserviço. Além disso, cada microsserviço possui uma fila de solicitações que armazena as solicitações do microsserviço

atual.

A fórmula utilizada, considerando que a taxa de envelhecimento é constante, é expressa como uma aproximação linear usando a equação $y = ax + c$. Além disso, foram estabelecidas fórmulas de função de distribuição acumulada com a função de densidade de probabilidade $c_i(t)$, como o total de contagem do código de recurso f_i , que esgota a unidade de recurso U_{fi} em sequência, para que a exaustão total sequencial causada pelos códigos de recurso possa ser determinada por $E_s(t)$. Usando as fórmulas descritas nas Equações 4.1 e 4.2, é possível identificar quando os indicadores excedem o limite e determinar qual microserviço está envelhecendo.

$$c_i(t) = \int_0^t c_i(t) dt \quad (4.1)$$

$$E_s(t) = \sum_{i=0}^m U_{fi} * \int_0^t c_i(t) dt \quad (4.2)$$

A opção de rejuvenescimento de software é implementada por meio da arquitetura CVA (*Container Vertical Autoscaling*). O experimento durou 30 dias e, para monitoramento, o *cAdvisor* foi utilizado para coletar memória livre e ferramentas de *RPC* (*Remote Procedure Call*) para coletar o número de solicitações atrasadas e pendentes.

My Services Got Old! Can Kubernetes Handle the Aging of Microservices?

Nesse experimento de [Flora et al. \(2021\)](#), Os autores examinaram um sistema de comércio eletrônico representativo composto por 5 microserviços, chamado *TeaStore* (desenvolvido como um *testbed* para artigos científicos), bem como um *framework* de teste de carga chamado *Locust*. Eles conduziram testes em 3 cenários diferentes. No cenário 1, eles testaram o funcionamento normal do sistema com nove usuários ativos exercendo-o por 48 horas sob uma carga de trabalho normal. No cenário 2, testaram a resposta do sistema ao estresse, simulando uma carga de 10 a 100 usuários ativos ao longo de 48 horas, o que causou um comportamento de envelhecimento. No cenário 3, combinaram os cenários 1 e 2, observando que o serviço *WebUI* do sistema falhou. O experimento durou 24 horas, tempo suficiente para demonstrar os efeitos do envelhecimento nos microserviços. Embora o envelhecimento de software tenha sido observado, o *Kubernetes* não demonstrou a capacidade de identificar ou resolver problemas relacionados ao envelhecimento em serviços monitorados. Os experimentos duraram 2 dias nos cenários 1 e 2, e 1 dia no cenário 3. Os autores utilizaram ferramentas do *Kubernetes*, como sondas de inicialização, prontidão e vivacidade, juntamente com métricas e eventos do próprio orquestrador, para fins de monitoramento.

Software Aging in Container-based Virtualization: An Experimental Analysis on Docker Platform

Os autores em [Oliveira et al. \(2021\)](#), elaboraram um trabalho baseando-se em um processo metodológico composto por 5 etapas: compreensão do sistema, planejamento de

medição, planejamento da carga de trabalho, experimento de estresse e análise de envelhecimento. Eles automatizaram a criação de grupos de 5 contêineres usando um *script*, com um atraso de 30 segundos entre a execução de cada rotina, e repetiram o processo até que 40 novos contêineres fossem criados. Da mesma forma, removeram a cada 5 contêineres com um atraso de 30 segundos entre cada remoção. Este ciclo de teste durou 25 dias, durante os quais um modelo de séries temporais, especificamente o modelo de tendência quadrática, foi ajustado. O modelo foi mais adequado para prever o uso de memória residente e virtual, com baixo *MAPE* (Erro Percentual Médio Absoluto) na previsão. Os autores descobriram que o processo *Dockerd* apresentava um crescimento linear no consumo de memória residente, juntamente com variações sazonais. As séries de modelos quadráticos previram que o uso de memória residente do processo *Dockerd* seria esgotado após 5 meses. Para monitorar e obter as métricas, os autores usaram utilitários do Linux como *free*, *ps* e *mpstat*, bem como arquivos no diretório */proc*. Cada medição foi realizada em intervalos de 60 segundos.

Docker platform aging: a systematic performance evaluation and prediction of resource consumption

Nesse artigo de [Vinícius et al. \(2022\)](#), foi realizado um experimento que consistiu em 3 etapas. A Etapa 1 durou 30 dias consecutivos, durante os quais 2 contêineres foram adicionados a cada 10 segundos até que o valor de 50 contêineres fosse alcançado. Após atingir o valor máximo, os contêineres foram removidos, e o processo foi repetido após uma espera de 50 segundos. A Etapa 2 durou 3 dias e utilizou a metodologia *SWARE*. Nesta etapa, os valores definidos na Etapa 1 foram aumentados, e foram criados 480 contêineres, incrementando-os de 16 em 16 até alcançar o valor máximo. Após atingir o valor máximo, os contêineres foram removidos, e o processo foi repetido após 50 segundos. Na etapa 3, foram utilizados 5 modelos de previsão, 4 baseados em séries temporais (*ARIMA*, *Drift*, *Holt* e *Holt-Winter*) e 1 baseado em aprendizado de máquina (*Long Short-Term Memory (LSTM)*). O conjunto de dados usado para comparação foi obtido da Etapa 1, onde 3 métricas de erro (*Mean Absolute Error (MAE)*, *Mean Absolute Percentage Error (MAPE)* e *Root Mean Squared Error (RMSE)*) foram usadas para avaliar a precisão. Após a realização do teste, o envelhecimento do software foi observado na etapa 1, nos Desktops 1 e 2. Na etapa 2, o envelhecimento do software foi observado em todos os 3 desktops. O rejuvenescimento foi eficiente em todos os 3 computadores. Nas etapas 1 e 2, o monitoramento foi realizado usando comandos (*free -mh*, *top* e *grep*). Na etapa 3, as métricas de erro (*RMSE*, *MAE* e *MAPE*) foram observadas e comparadas com o *LSTM*, que se mostrou mais adequado entre os analisados para prever a progressão do envelhecimento.

Software Aging Effects on Kubernetes in Container Orchestration Systems for Digital Twin Cloud Infrastructures of Urban Air Mobility

O artigo de [Costa et al. \(2023\)](#), investiga questões de envelhecimento de software em uma infraestrutura de nuvem com um gêmeo digital operacional (*ODT*) de uma infraestrutura típica de mobilidade aérea urbana (*UAM*). Um *Cluster Kubernetes* foi utilizado para uma abordagem

de simulação em nuvem neste sistema de nuvem privada. O processo metodológico ocorreu em 4 etapas, começando com a análise dos requisitos para sua realização nos ambientes *Minikube* e *K3S*, seguida pelo desenvolvimento e análise de *scripts* de monitoramento, execução do ambiente e seu estresse, e, finalmente, a geração de gráficos para análise dos resultados. O comportamento do *Kubernetes* foi avaliado usando ferramentas *Nginx* e *Kiali*. O ambiente foi monitorado por 2 horas sem carga de trabalho. Após a inicialização do *Cluster*, o ambiente foi monitorado por 2 horas durante a operação normal e, finalmente, uma carga de trabalho alta foi realizada, emulando o dimensionamento automático 420 vezes (5 Pods e 1 *Service*). Após o término desta etapa, o *Cluster* foi parado por 2 horas como uma possível ação de rejuvenescimento de software. Após o experimento, os efeitos de envelhecimento de software foram detectados. Indicou-se que um vazamento de memória não foi completamente eliminado quando o *Cluster* foi encerrado. Comandos como *pidstat*, *free* e *top* foram usados para fins de monitoramento.

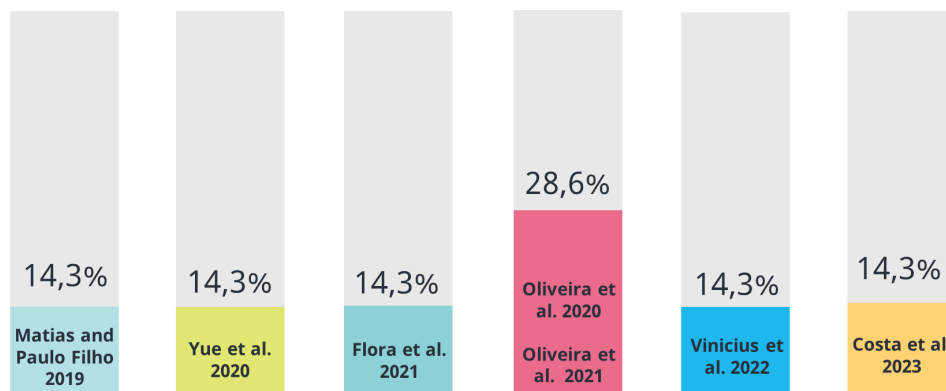
4.3 Análise dos Resultados

Nesta Seção são apresentados os resultados do MSL. Os resultados da extração de dados são expostos, analisados e discutidos e para um melhor entendimento e visualização dos mesmos, essa seção foi subdividida de acordo com as QPs.

4.3.1 QP1: Quais são os autores mais produtivos nesta área?

Conforme mostrado na Figura 9, os autores [Oliveira et al. \(2020\)](#) e [Oliveira et al. \(2021\)](#) participaram em 28,6% das publicações sobre envelhecimento e rejuvenescimento de software em microsserviços, com dois artigos publicados nos anos de 2020 e 2021. Os autores [Torquato e Vieira \(2019\)](#), [Yue, Wu e Xue \(2020\)](#), [Flora et al. \(2021\)](#), [Vinícius et al. \(2022\)](#), e [Costa et al. \(2023\)](#) contribuíram com 14,3% das publicações, com um artigo publicado por cada autor.

Figura 9 – Gráfico de Participação nas Publicações (Autor/Ano)

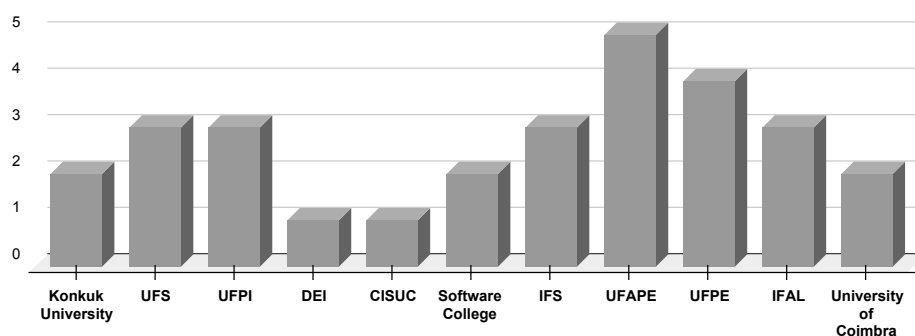


Fonte: O autor

4.3.2 QP2: Quais universidades contribuem mais para os artigos mencionados?

Na figura 10, é mostrado de forma gráfica todas as instituições que já abordaram o tema proposto. A universidade com a maior quantidade de participações nas publicações foi a **UFAPE**, (*Universidade Federal do Agreste de Pernambuco*), com cinco participações nas publicações. Em segundo lugar está a **UFPE**, (*Universidade Federal de Pernambuco*), com participação em quatro publicações. As universidades **UFS**, (*Universidade Federal de Sergipe*), e **UFPI**, (*Universidade Federal do Piauí*), com três participações nas publicações. Os institutos **IFS**, (*Instituto Federal de Sergipe*), e **IFAL**, (*Instituto Federal de Alagoas*), também com três participações nas publicações. Posteriormente estão as instituições *Konkuk University*, *Software College* e *University of Coimbra* com duas participações nas publicações e por fim as instituições **DEI** e **CISUC** com uma participação nas publicações para cada uma delas.

Figura 10 – Gráfico da Participação das Instituições



Fonte: O autor

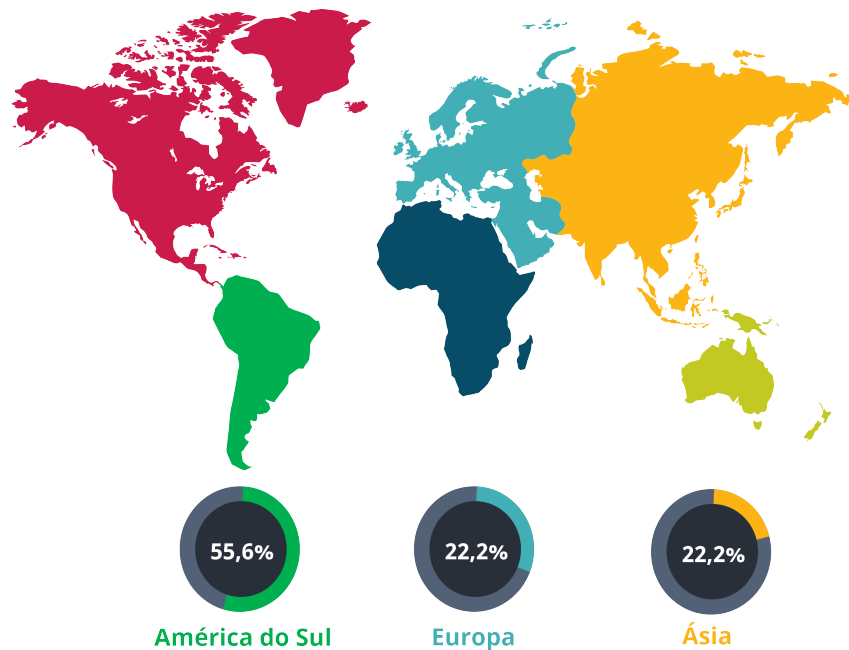
4.3.3 QP3: Em quais continentes foram conduzidas pesquisas sobre o envelhecimento e rejuvenescimento de software em microsserviços?

Na figura 11, é vislumbrado as pesquisas relacionadas com o tema proposto nesse MSL. Foram encontradas pesquisas em três continentes distintos, destacando a América do Sul com 55,6%, a Europa com 22,2% e a Ásia com 22,2%. Na América do Sul, o Brasil participou de cinco das publicações sobre o tema, na Europa, Portugal com duas participações nas publicações e na Ásia, a China e a Coreia do Sul com participação em uma publicação.

4.3.4 QP4: Quais são as plataformas e orquestradores de microsserviços mais utilizados neste MSL?

Nos sete artigos analisados, todos eles usaram como plataforma o *Docker*. Apenas os autores Flora et al. (FLORA et al., 2021), Oliveira et al. (OLIVEIRA et al., 2021), Yue et al. (YUE; WU; XUE, 2020) e Costa et al. (COSTA et al., 2023), exploraram o uso de orquestradores

Figura 11 – Gráfico das Publicações



Fonte: O autor

de microsserviços. Enquanto em (FLORA et al., 2021), (YUE; WU; XUE, 2020) e (COSTA et al., 2023) foram explorados testes com o *Kubernetes*, em (OLIVEIRA et al., 2021) foram realizados experimentos usando o *Docker Swarm*. A tabela 2, mostra o resumo de cada autor e o uso das Plataformas e Orquestradores explorados nos experimentos.

Tabela 2 – Orquestradores de Microsserviços mais Utilizados nos Artigos.

Autor	Plataforma	Orquestrador
Torquato e Vieira (2019)	<i>Docker</i>	<i>None</i>
Oliveira et al. (2020)	<i>Docker</i>	<i>None</i>
Yue, Wu e Xue (2020)	<i>Docker</i>	<i>Kubernetes</i>
Flora et al. (2021)	<i>Docker</i>	<i>Kubernetes</i>
Oliveira et al. (2021)	<i>Docker</i>	<i>Docker Swarm</i>
Vinícius et al. (2022)	<i>Docker</i>	<i>None</i>
Costa et al. (2023)	<i>Docker</i>	<i>Kubernetes</i>

4.3.5 QP5: Quais métodos ou técnicas são utilizados para forçar o envelhecimento e rejuvenescimento de microsserviços?

Dos sete artigos analisados, dois deles, (Matias and Paulo Filho 2019 (MATIAS; FILHO, 2006) e Vinicius et al. 2022(VINÍCIUS et al., 2022)), utilizaram a abordagem **SWARE** (*Stress, Wait and Rejuvenation*). Enquanto em (MATIAS; FILHO, 2006) foram estabelecidas três etapas: (i) *estresse* - ambiente de estresse com a carga de trabalho acelerada para induzir a ativação de bugs; (ii) *esperar* - interromper o envio da carga de trabalho para observar possíveis acumulados

efeitos e (iii) *rejuvenescer* - submeter uma ação de rejuvenescimento a perceber mudanças no estado interno do *software*, em (VINÍCIUS et al., 2022) foi inicialmente realizada uma investigação e avaliação de métricas em período específico, posteriormente aplicado o (SWARE) e por fim foram utilizados modelos de previsão usando os dados coletados com uma comparação.

No artigo de Oliveira et al. 2020(OLIVEIRA et al., 2020) e Oliveira et al. 2021(OLIVEIRA et al., 2021), foi inicialmente realizado uma análise compreensiva do sistema, posteriormente um planejamento de medição, um planejamento de carga de trabalho e por fim foi realizado um estresse no software com a finalidade de analisar seu envelhecimento. No trabalho de Yue et al. 2020 (YUE; WU; XUE, 2020) foi realizada a utilização de recursos do sistema operacional, analisada a latência por camada e verificado o número de solicitações pendentes após aplicação de carga máxima do experimento.

Em Flora et al. 2021 (FLORA et al., 2021), foram inicialmente utilizados testes com operação normal (baixa carga de trabalho), em um segundo momento testes de estresse específicos por fim a utilização de uma falha relacionada ao envelhecimento. Na pesquisa de Costa et al. 2023 (COSTA et al., 2023), foi realizado o monitoramento de todo o processo e em paralelo inicializado o *Cluster* kubernetes, aplicando baixa carga de trabalho, posteriormente aumentado gradativamente a carga, verificado tempo de resposta das aplicações e constatação do envelhecimento.

4.3.6 QP6: Do ponto de vista da monitorização, quais recursos foram monitorados nos artigos explorados?

Todos os autores monitoraram os recursos computacionais do sistema operacional para verificar o ponto em que ocorre o envelhecimento do software. A Tabela 3 mostra os comandos Linux conforme síntese dos artigos na Seção 4.2, todos os autores monitoraram os recursos computacionais do sistema operacional com a finalidade de constatar o ponto em que o sistema envelhecia.

Na Tabela 3 são expostos os comandos utilizados do Linux para coletar informações e assim validar o envelhecimento de software.

Em (MATIAS; FILHO, 2006) foram coletadas informações de estatísticas do processador com o comando (*mpstat*), métricas de carga do sistema, atividade da CPU e memória com o comando (*sar*), além de monitorar tarefas individuais gerenciadas pelo kernel do Linux usando o comando (*pidstat*), esse último também usado por (COSTA et al., 2023).

Em (OLIVEIRA et al., 2020; OLIVEIRA et al., 2021; VINÍCIUS et al., 2022; COSTA et al., 2023) foi utilizado o comando (*free*) para gerar resumo do uso de RAM, memória total, usada, livre, compartilhada e disponível.

Nos trabalhos de (OLIVEIRA et al., 2020; OLIVEIRA et al., 2021) investigou-se recursos dos processos, o comando (*ps*), além de informações relacionadas aos sistemas de arquivos, o

espaço total e o espaço disponível utilizando o comando (*df*).

Em (FLORA et al., 2021), os autores se utilizaram de recursos do próprio orquestrador, analisando *Kubernetes metrics, events startup probe, readiness probe, liveness probe*.

Ainda em (VINÍCIUS et al., 2022) os autores utilizaram os comandos *top* (como uma forma de ver os processos em execução no Linux, com atualização ordenados pelos processos que utilizam mais CPU, no topo da tela) e *grep* (uma das ferramentas mais úteis e versáteis disponível no Unix para procurar por padrões especificados pelo usuário dentro de arquivos de texto).

Tabela 3 – Utilitários de Monitoramento.

Autor	Utilitários de Monitoramento
Torquato e Vieira (2019)	<i>mpstat, sar e pidstat</i>
Oliveira et al. (2020)	<i>free, ps, df</i>
Yue, Wu e Xue (2020)	<i>cAdvisor</i>
Flora et al. (2021)	<i>Kubernetes Metrics / Events, startup probe readiness probe / liveness probe</i>
Oliveira et al. (2021)	<i>free, ps, pdf</i>
Vinícius et al. (2022)	<i>free -mh, top e grep</i>
Costa et al. (2023)	<i>free e pidstat</i>

4.4 Ameaças à Validade

Para que um trabalho seja aceito como contribuição ao conhecimento científico, o pesquisador precisa convencer os leitores de que as conclusões tiradas de um estudo empírico são válidas (EASTERBROOK et al., 2008). Nesta seção estão expostas todas as ameaças à validade encontradas neste estudo. São elas:

- A string de busca e as questões de pesquisa podem não cobrir todos os estudos relevantes da área. Como forma de mitigação, as palavras-chave e as questões de pesquisa foram elaboradas a partir de uma pesquisa prévia do assunto.
- **Validade Interna:** Caso seja executada novamente, esse MSL poderia apresentar um resultado diferente do obtido. Para mitigar essa ameaça, a metodologia definida por (KITCHENHAM, 2004a) foi usada nesse trabalho. Conflitos na seleção de artigos foram discutidos entre os autores, para mitigar o viés pessoal na seleção dos estudos;
- **Validade Externa:** O resultado da busca pode não conter todos os artigos relevantes para o estudo. Como forma de mitigar essa ameaça, a busca foi realizada em 5 das principais bases de dados científicas *ACM Digital Library, IEEE, MDPI, Scopus* e *textitSpringerLink*.

4.5 Considerações Finais do Capítulo

Esta MSL identificou e analisou artigos que investigam o envelhecimento e rejuvenescimento de software baseado em microsserviços. Os resultados desta revisão fornecem *insights* valiosos sobre o estado atual da pesquisa nesse campo e identificam lacunas na literatura que justificam uma exploração mais aprofundada. Pesquisadores podem usar essas informações para orientar o desenvolvimento de novos estudos e avançar nossa compreensão desta área importante de pesquisa. No geral, este MSL representa uma contribuição importante para informar futuras pesquisas e práticas no desenvolvimento de software baseado em microsserviços.

Para trabalhos futuros, um *Cluster Kubernetes* será implantado para investigar o impacto das estratégias de rejuvenescimento de software na infraestrutura de orquestração de contêineres. Testes específicos serão realizados tanto nos nós mestre quanto nos nós de trabalho.

5

SAR em Sistema de Orquestração de Con- têineres

Este Capítulo apresenta em detalhes os resultados obtidos a partir da experimentação em cinco cenários, visando demonstrar o comportamento de dois *Clusters Kubernetes* operando nos sistemas Ubuntu 20.04.6 LTS e Ubuntu 22.04.2 LTS. O objetivo é oferecer uma análise abrangente do desempenho individual de cada cenário, destacando métricas essenciais de recursos e serviços monitorados, tais como utilização de CPU, utilização de disco, consumo de memória RAM, consumo de memória *cached*, e a performance dos serviços do *containerd* e *kubelet* em todos os cinco cenários. Adicionalmente, nos cenários de 2 a 5, é apresentado o comportamento do tempo de resposta de um cliente externo desses *Clusters* ao acessar um site interno em ambos os ambientes. Essa abordagem permite uma compreensão mais completa e contextualizada do desempenho, considerando não apenas as métricas internas, mas também a experiência de usuário externo ao interagir com os serviços fornecidos pelos *Clusters Kubernetes*.

5.1 Ambiente Proposto para Avaliação

Com o propósito de consolidar a operação de vários servidores em um único sistema, esta seção apresenta a configuração dos *Clusters*, que incluem tanto os aspectos de hardware e software, como o provisionamento de máquinas virtuais para composição dos mesmos.

- **Hardware**

Para simular cenários comuns no mercado, foi utilizado um servidor dedicado do modelo *DELL PowerEdge M640*, no qual foram criadas as máquinas virtuais. Uma visão mais detalhada da configuração do hardware do servidor, está descrita na Tabela 4.

- **Softwares e *Clusters***

Na infraestrutura detalhada na Tabela 4, foram realizadas a instalação e configuração do software *VMware ESXi 7*, no servidor *DELL PowerEdge M640*. Em seguida, foram

Tabela 4 – Hardware Usado.

Função	Hardware
Storage	2 x Dell 300GB 15K RPM SAS 12Gbps 2.5 (RAID 1)
CPU	1 x Intel(R) Xeon(R) Silver 4112 CPU @ 2.60GHz
Memória RAM	8 x DRAM, DDR-4, 32 GB, Dual Rank, 2400 MT/s 256 GB
Network	1 x QLogic 577xx/578xx 10 Gb Ethernet BCM57840

criadas oito máquinas virtuais, cada uma equipada com 6 vCPUs, 10 GB de RAM e 50 GB de armazenamento. Quatro dessas máquinas virtuais foram configuradas para executar o sistema operacional Ubuntu 20.04.3 LTS, enquanto as outras quatro foram provisionadas com o Ubuntu 22.04.2 LTS. Em todas as máquinas virtuais, o "Containerd" foi implementado como o *runtime* de contêiner, na versão 1.6.12, e a função de swap desativada. Posteriormente, procedeu-se com a instalação e inicialização do *Cluster Kubernetes*, utilizando a versão v1.27.2, que inclui 1 nó *Master* e 3 nós *Workers* para coordenar e gerenciar a solução. É relevante notar que a capacidade máxima de pods suportada por cada *Cluster* é de 450.

Além disso, foi criada uma nona máquina virtual denominada "acesso cliente" com o propósito de acessar os pods (portais de notícias) implementados em cada um dos *Clusters*. É importante ressaltar que essa máquina virtual está localizada fora de ambos os *Clusters*. Uma representação visual de toda essa configuração pode ser visualizada na Figura 12.

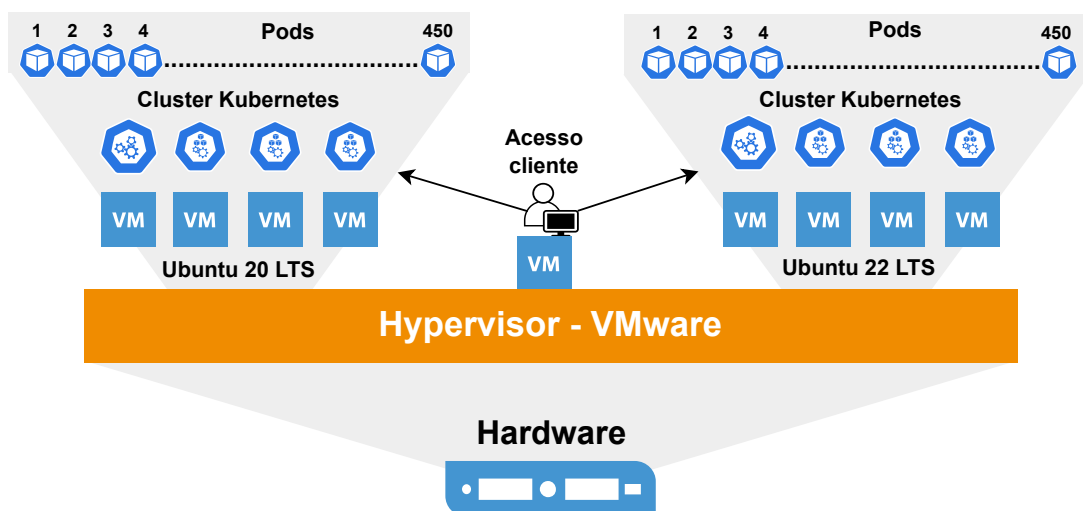


Figura 12 – Cluster Kubernetes base dos Experimentos

Durante este estudo, realizamos experimentos em dois sistemas operacionais distintos: Ubuntu 20.04.6 LTS e Ubuntu 22.04.2 LTS. Utilizamos o *Containerd* como *runtime* de contêiner e o *Kubernetes* como orquestrador de *Clusters* em 5 cenários distintos. Mantivemos consistentes as configurações dos *Clusters*, as cargas de trabalho aplicadas e os períodos de teste. Os experimentos envolveram um aumento gradual da carga hospedada,

começando com 150 pods “A”, 150 pods “B” e 150 pods “C” de um servidor web (*nginx*), seguido pela remoção na mesma proporção.

É importante salientar que o servidor foi reiniciado sempre ao fim da execução de cada cenário dos experimentos.

5.2 Análise dos Resultados

Esta seção apresenta os resultados obtidos a partir dos experimentos realizados. Ao todo, foram cinco cenários avaliados, onde um deles tem por objetivo investigar os efeitos do envelhecimento de software no *Kubernetes*, enquanto os outros quatro avaliam diferentes estratégias de rejuvenescimento de software. Foram analisadas métricas como utilização da CPU, consumo de memória RAM, consumo de memória *cached*, uso de disco, desempenho dos serviços do *containerd* e *kubelet*, além das requisições feitas ao portal de notícias.

Para cada um dos 5 cenários, serão exibidos gráficos representativos de apenas 1 dos 3 *Workers* onde os experimentos foram executados, uma vez que o comportamento dos nós foi bastante semelhante devido às configurações dos *Clusters* instalados. Todos os gráficos mostram o limite final atingido por cada experimento, representado no plano cartesiano (x, y) e destacado em vermelho, onde x representa sempre o tempo em minutos, e y representa a quantidade de recursos utilizada. Também serão apresentados dados em azul durante as tentativas de rejuvenescimentos nos cenários de 2 a 5. Os gráficos dos nós *Masters* não serão incluídos, já que sua função primária era administrar o *Cluster* e o mesmo não demonstrou utilização de recursos significativa.

5.2.1 Cenário #1 - Efeitos do Envelhecimento do Software

O objetivo do *Cenário #1* foi investigar os efeitos do envelhecimento de software, observando o comportamento de sistemas de longa duração, com o intuito de identificar quais serviços dos sistemas operacionais consomem mais recursos, contribuindo para a degradação gradual da sua eficiência, desempenho e confiabilidade. Foi necessário criar e executar um *script* para adicionar e remover pods de um servidor web (*nginx*) em ambos os *Clusters Kubernetes*.

O ciclo de testes *Cluster* foi baseado em 12 dias e ocorreu da seguinte maneira: Nos primeiros 2 dias, o *Cluster* não realizou nenhuma atividade de carga. Já nos 10 dias seguintes, o mesmo executou os *scripts* de carga, descritos anteriormente. Este ciclo foi executado 3 vezes.

Neste cenário, buscou-se demonstrar por meio dos gráficos, gerados pelo *Worker3* no *Cluster* com Ubuntu 20, a utilização da CPU, consumo de memória, utilização disco, além do consumo de memória virtual (VSZ) dos processos *Containerd* e *Kubelet* (ver Figura 15). Como pode ser observado nos gráficos da Figura 13(a), o consumo de CPU permaneceu inferior a 5% durante aproximadamente 2 dias em que não houve carga de processamento. Após esse período, foi aplicada uma carga de stress ao *Cluster*, onde foi possível observar picos de utilização de

CPU que ultrapassam 55%. Concluindo o período da carga de estresse (cerca de 10 dias), houve uma pausa de aproximadamente 2 dias sem atividade no *Cluster*; nesse período é verificado que, mesmo sem carga, a utilização de CPU não volta ao valor inicial observado antes do período de carga, atingindo picos acima de 22,5%. Quando iniciou-se novamente a carga de stress, o *Worker3* atingiu picos de mais de 50% de processamento até o momento de falha total do *Cluster* com 19979 minutos de funcionamento. O *Worker3* parou de funcionar, não sendo possível coletar mais informações.

A Figura 13(b) ilustra o consumo de disco ao longo do tempo. No início do período observado, o consumo de disco era cerca de 15000 MB. Durante os primeiros dois dias, não houve atividade significativa, mas depois houve flutuações de aumento e diminuição do consumo. Essas variações podem ser atribuídas às criações e remoções dos pods no *Cluster*. Após um ciclo de aproximadamente 10 dias com carga de estresse, o consumo de disco apresentou uma tendência de queda, embora não tenha retornado ao nível inicial registrado quando o *Cluster* foi iniciado. Quando um novo ciclo de criação e remoção de pods foi iniciado, é observada uma interrupção no funcionamento do *Worker3*, que não conseguiu continuar coletando dados de consumo de disco a partir do minuto 19979.

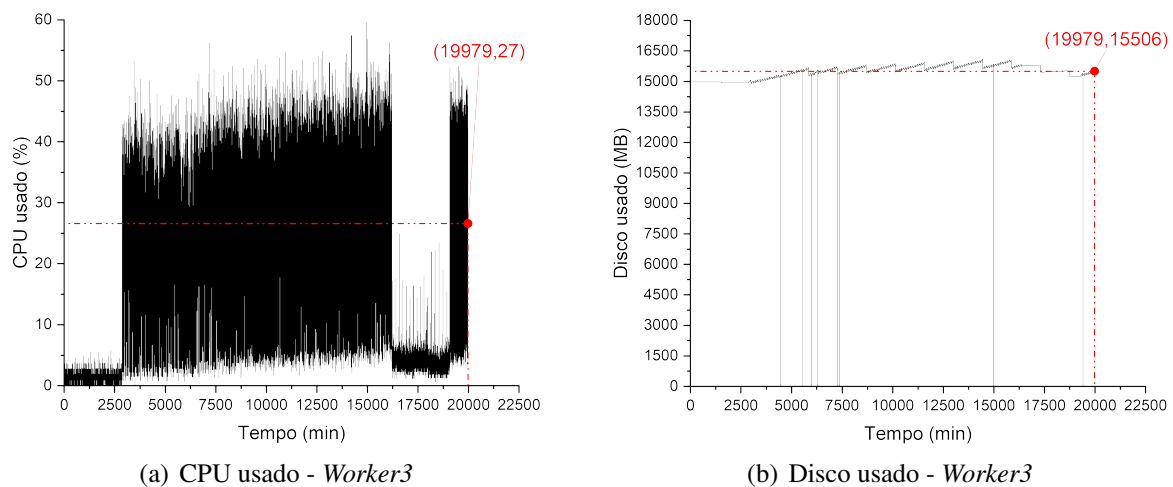


Figura 13 – Cenário 1 - Carga de Estresse e Repouso da Aplicação (Ubuntu 20).

Na Figura 14(a), foi analisado o uso de memória. Durante os 2 primeiros dias, quando não havia carga de trabalho, o consumo de memória RAM permaneceu abaixo de 1000 MB. No entanto, quando o *Cluster* foi submetido a uma carga total devido à criação e remoção de pods, o consumo de memória aumentou significativamente, ultrapassando 9000 MB. Após esse período de carga, o *Cluster* passou por aproximadamente 2 dias de inatividade, e durante esse intervalo, o uso de memória diminuiu para cerca de 7766 MB, mantendo-se relativamente constante e não voltando para o ponto inicial, demonstrando vazamento de memória e envelhecimento de software. Ao iniciar um novo período de carga com a criação e remoção de pods, observamos que o *Worker3* parou de funcionar no minuto 19979.

Foi avaliado o comportamento do uso de memória *cached*, como pode ser visto na

Figura 14(b). Nos primeiros 2 dias, quando não havia carga de trabalho, o consumo de memória permaneceu ligeiramente acima dos 1000 MB. No entanto, quando submetido o *Cluster* a uma carga total devido à criação e remoção de pods, o uso de memória *cached* aumentou significativamente, atingindo quase 4500 MB. À medida que a memória utilizada aumentava, a memória *cached* diminuía, chegando a aproximadamente 250 MB durante o período de carga de estresse. Após esse período de carga, o *Cluster* passou por cerca de 2 dias de inatividade, e nesse intervalo, o uso de memória *cached* não ultrapassou os 1000 MB. Iniciando um novo período de carga com a criação e remoção de pods, foi observado que o *Worker3* parou de funcionar nos minutos 19979, quando a memória *cached* consumia 283 MB de memória.

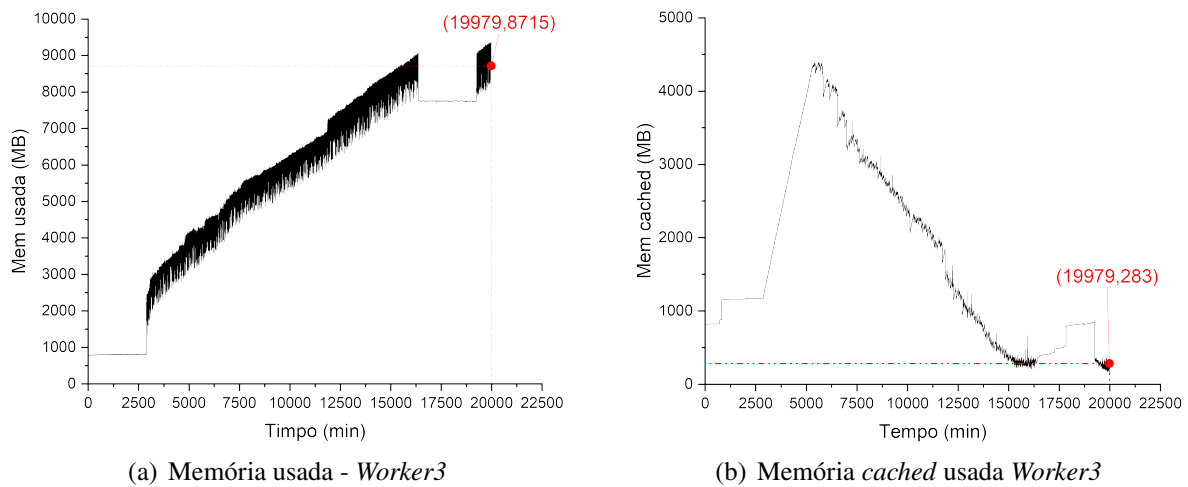


Figura 14 – Cenário 1 - Carga de Estresse e Repouso da Aplicação (Ubuntu 20).

Na Figura 15(a), foi examinado o comportamento do consumo de memória virtual (VSZ) do *Containerd*. Nos primeiros dois dias sem carga, o consumo de memória virtual estava consistentemente abaixo de 2000 MB. No entanto, durante o período de carga subsequente, observou-se picos de consumo de memória que ultrapassaram os 5000 MB. Quando iniciou-se o segundo período sem carga, o consumo de memória virtual permaneceu relativamente constante em torno de 4399 MB, mas voltou a subir quando iniciou o novo período de carga, atingindo seu pico no minuto 19978, quando o *Worker3* parou de funcionar.

O comportamento do consumo de memória virtual (VSZ) no *Kubelet* pode ser observado no gráfico na Figura 15(b) revelando que nos 2 primeiros dias sem carga, o consumo não ultrapassou 2500 MB. No entanto, ao iniciar o período de carga, o consumo subiu para aproximadamente 60000 MB e permaneceu constante até o momento em que o *Worker3* parou. Esse consumo se mostrou intrigante, uma vez que as VMs utilizadas neste experimento possuem 10000 MB de memória e o *swap* não está ativo no *Cluster*, como mencionado na seção 5.1. Esse comportamento será analisado nos trabalhos futuros.

Após a investigação do comportamento no Ubuntu 20, os testes permaneceram no **Cenário #1**, agora com o Ubuntu 22. A partir do gráfico da Figura 16(a), é possível observar que o consumo de CPU permaneceu abaixo de 50%, durante os aproximadamente 2 dias em

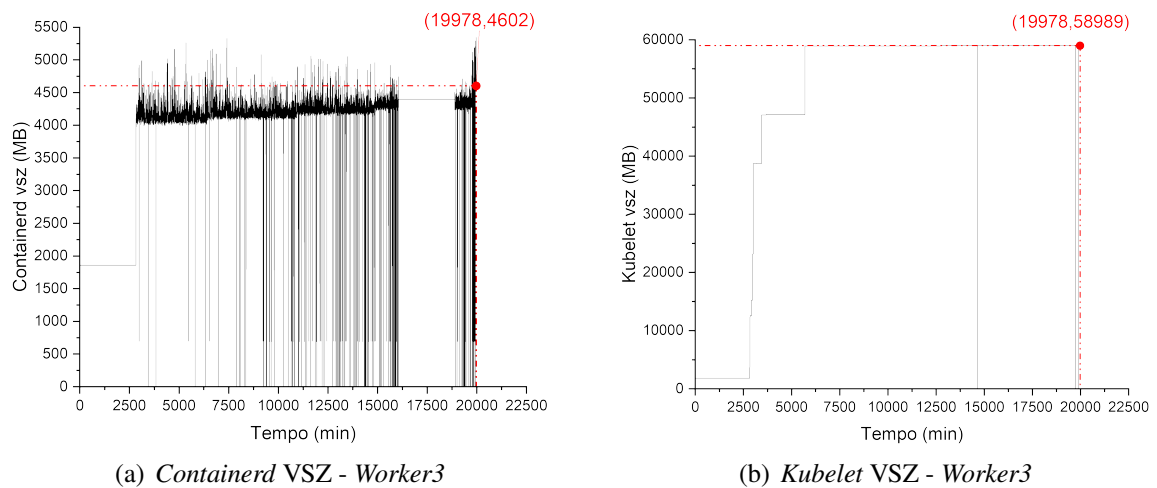


Figura 15 – Cenário 1 - Carga de Estresse e Repouso da Aplicação (Ubuntu 20).

que não houve carga de trabalho. Entretanto, quando aplicada uma carga de estresse ao *Cluster*, após esse período notou-se picos de consumo que ultrapassaram os 57.5% de utilização de CPU. É importante destacar que o *Cluster* não conseguiu manter a operação ao longo do ciclo de aproximadamente 10 dias com essa carga intensa e parou de funcionar após 8149 minutos de operação. O *Worker3* continuou coletando informações, mesmo quando o ciclo de criação e remoção de recursos já não estava mais funcionando.

No que diz respeito ao consumo de disco, o gráfico na Figura 16(b) ilustra a utilização de disco ao longo do tempo. O consumo inicial de disco foi registrado em 23750 MB. Nos 2 primeiros dias, observou-se que o consumo aumentou para além de 25000 MB e manteve-se relativamente estável nesse patamar. Porém, ao iniciar o período de carga, houve um aumento notável no uso de disco, atingindo um pico antes de apresentar uma queda por volta do minuto 4819. Posteriormente, o consumo de disco voltou a aumentar até o minuto 8149, quando o ciclo de criação e remoção de pods falhou. Vale salientar que, mesmo com o término do ciclo de criação e remoção, o *Worker3* continuou coletando informações, uma vez que o sistema operacional permaneceu funcional.

Na Figura 17(a), foi analisado o uso de memória. Durante os aproximadamente 2 primeiros dias, quando não havia carga de trabalho, o consumo de memória se manteve abaixo de 3000 MB. No entanto, ao submeter o *Cluster* a uma carga de estresse, pode-se observar um aumento significativo no consumo de memória, ultrapassando os 6000 MB. É importante destacar que o *Cluster* não conseguiu completar o ciclo previsto de 10 dias, falhando no minuto 8149. Mesmo após a interrupção do ciclo de criação e remoção de pods, o consumo de memória permaneceu elevado, registrando 5337 MB, não retornando ao nível inicial e indicando um possível vazamento de memória.

Ao analisar o comportamento do uso de memória *cached*, conforme evidenciado na Figura 17(b). Nos primeiros dois dias, mesmo sem carga de trabalho, foi constatado um consumo

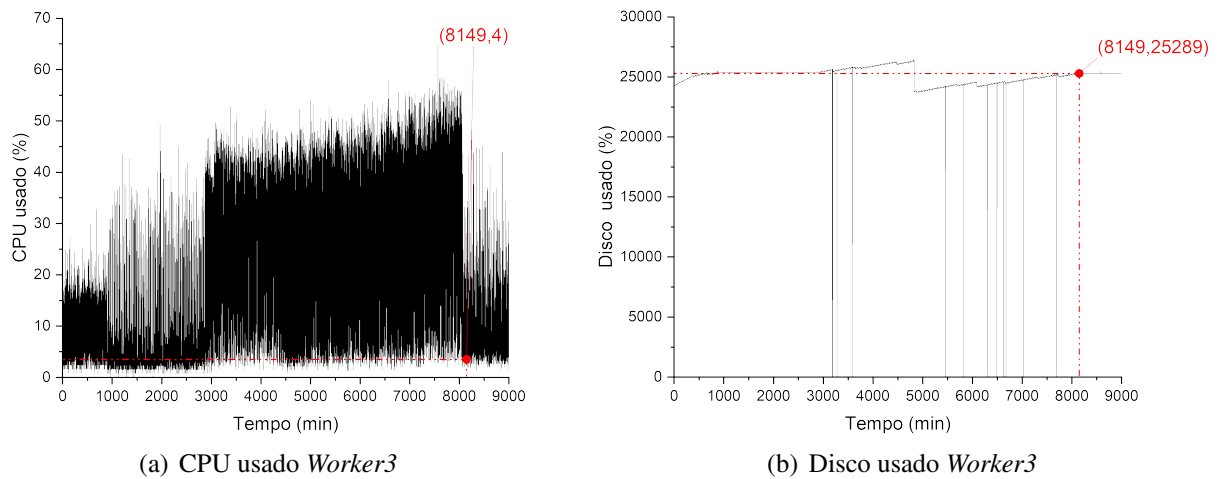


Figura 16 – Cenário 1 - Carga de Estresse e Descanso da Aplicação (Ubuntu 22).

de memória *cached* que ultrapassou os 4500 MB. Entretanto, ao submeter o *Cluster* a uma carga de estresse, resultante da criação e remoção de pods, notou-se uma diminuição no uso de memória *cached* à medida que a memória total utilizada aumentava. O *Cluster* não conseguiu concluir o ciclo planejado de 10 dias, falhando no minuto 8149, quando a memória *cached* consumia 2515 MB.

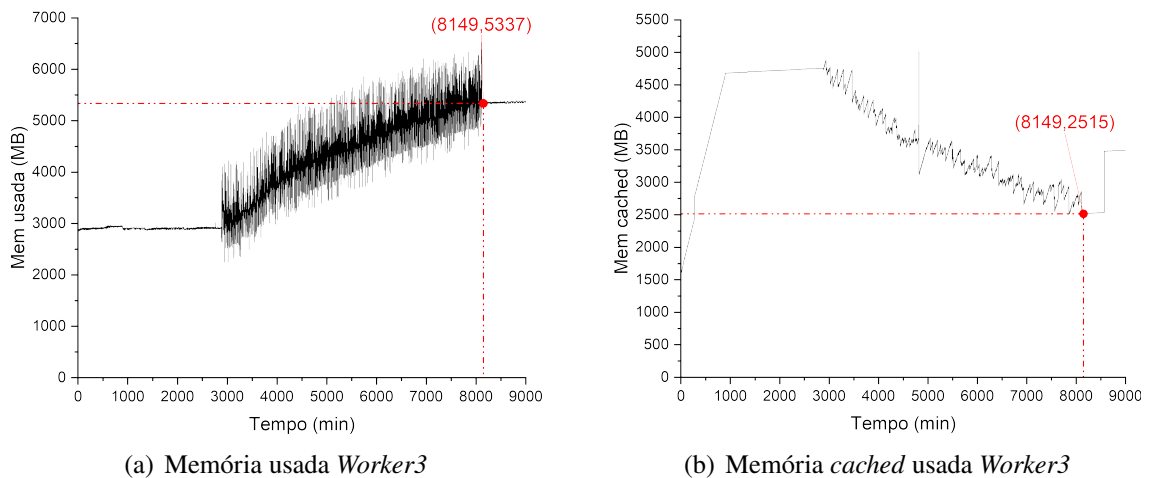


Figura 17 – Cenário 1 - Carga de Estresse e Descanso da Aplicação (Ubuntu 22).

Na análise do serviço *Containerd*, o uso de memória virtual foi examinado (ver Figura 18(a)). Nos primeiros 2 dias de experimento, quando não havia carga de trabalho, o consumo de memória virtual utilizou cerca de 4678 MB. No entanto, durante o período de carga, picos no consumo de memória que ultrapassaram os 5000 MB foram notados, persistindo até o minuto 7990. Após esse ponto, o serviço estabilizou seu consumo em 4527 MB, mesmo quando o *Cluster* não conseguiu prosseguir com o ciclo de criação e remoção dos pods.

Ao analisar o comportamento do consumo de memória virtual no *Kubelet*, a Figura 18(b) revela que, durante os 2 primeiros dias sem carga, o consumo de memória se manteve abaixo de 3000 MB. Porém, ao iniciar o período de carga, identificou-se um aumento significativo no

consumo, atingindo aproximadamente 5677 MB, e essa quantidade se manteve constante mesmo após o *Cluster* não conseguir continuar com o ciclo de criação e remoção dos pods.

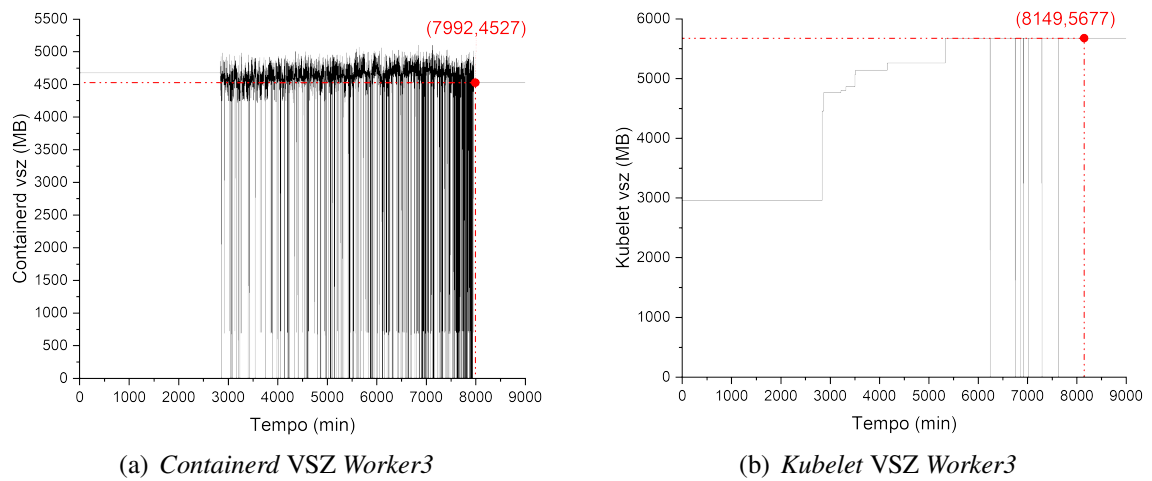


Figura 18 – Cenário 1 - Carga de Estresse e Descanso da Aplicação (Ubuntu 22).

Neste cenário, descobertas significativas sobre o envelhecimento de software nos *Clusters* Ubuntu 20 e 22 foram reveladas. No Ubuntu 20, observamos um aumento consistente no uso de recursos, sugerindo principalmente vazamentos de memória, persistindo mesmo após períodos de inatividade. Sob cargas de trabalho intensas, o *Cluster* apresentou picos significativos no consumo de CPU. O uso de memória mostrou uma tendência crescente e, durante a inatividade, uma diminuição, mas sem retornar ao ponto inicial, indicando vazamento de memória. O uso de disco mostrou um leve aumento. O serviço *Containerd* apresentou variações. Enquanto isso, a análise do serviço *Kubelet* mostrou um comportamento peculiar, consumindo cerca de 6 vezes mais memória vsz do que o *cluster* poderia fornecer. Além disso, há sinais claros de que o *Cluster* não se recupera totalmente após cargas de trabalho. Os consumos dos recursos como memória usada, memória cached, disco usado e serviços não voltam ao ponto inicial, indicando envelhecimento de software. No Ubuntu 22, o *Cluster* não conseguiu sustentar operações durante o primeiro ciclo planejado sob carga intensa, indicando instabilidade. Na análise do consumo de CPU, observou-se um comportamento mais agressivo ao longo de todo o período, mesmo durante os períodos de inatividade. Os padrões de uso de disco se assemelhavam aos do Ubuntu 20, mostrando uma leve tendência de aumento. O uso de memória permaneceu estável durante os períodos de ociosidade e aumentou durante a carga. O consumo de memória VSZ no serviço *Containerd* mostrou semelhanças com o Ubuntu 20, enquanto o serviço *Kubelet* exibiu padrões distintamente diferentes. No Ubuntu 22, o consumo do serviço não excedeu a capacidade do *Cluster*.

5.2.2 Cenário #2 - Limpeza de *caches* e *Buffers* como Estratégia de Rejuvenescimento

O propósito do **Cenário #2** foi avaliar a eficácia da limpeza de memória *cache* e *buffer* como estratégia de rejuvenescimento de software. Neste cenário, foram implementadas as mesmas cargas do **Cenário #1**, sem pausas de inatividade no *Cluster*. A limpeza dos *caches* e *buffers* do sistema foram realizados através da execução de um *script*, em intervalos de aproximadamente 2 dias. O objetivo foi liberar a memória RAM anteriormente alocada para armazenar os dados em *caches* e *buffers*, conforme destacado nas Figuras 14(b) e 17(b) do **Cenário #1**. Esse processo visou otimizar a utilização da memória útil, com o intuito de estender a disponibilidade dos *Clusters*.

No **Cenário #2**, examinando os gráficos do *Worker2* com Ubuntu 20, é possível observar que o *Cluster* não conseguiu atingir a meta de 10 dias de carga. O uso da CPU mostrou uma tendência gradual de aumento, atingindo picos de cerca de 55% (ver Figura 19(a)). O monitoramento foi interrompido no minuto 7839. Quanto ao uso de disco, notou-se uma tendência ascendente, com um consumo inferior a 1600 MB no início (minuto zero) e um aumento para 16470 MB no minuto 7839, quando o *Worker2* interrompeu o consumo (ver Figura 19(b)).

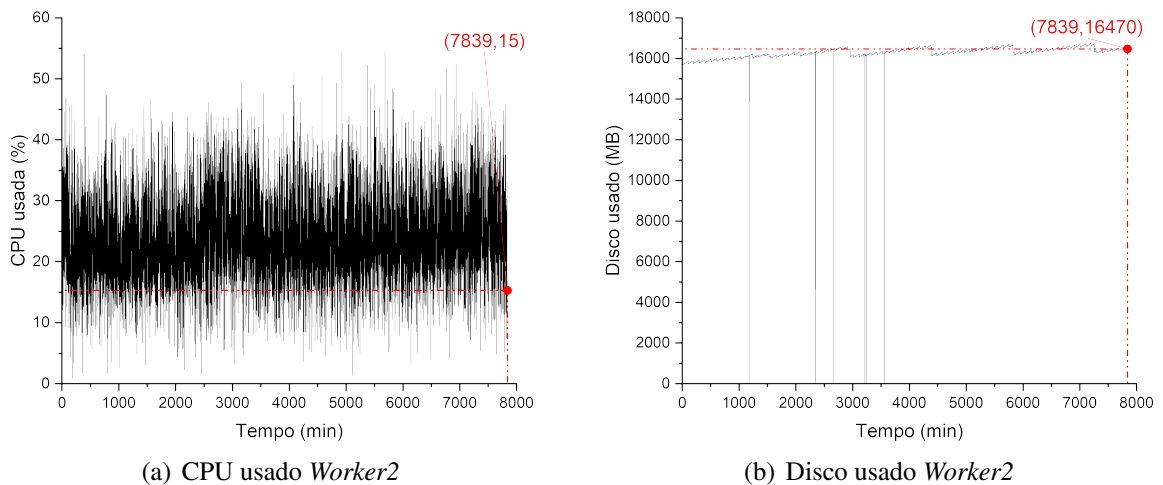


Figura 19 – Cenário 2 - Limpeza de *caches* e *buffers* (Ubuntu 20).

A Figura 20(a) apresenta o uso de memória, exibindo uma trajetória de aumento contínuo, sem oscilações significativas nos minutos 2873 e 5726, quando ocorreu a limpeza do *cached* do sistema. A coleta de dados de memória cessou no minuto 7839, com um consumo total de 9277 MB. No Gráfico de consumo de memória *cached*, apresentado na Figura 20(b), foi possível detectar um pico de consumo de memória *cached* acima de 4000 MB, com quedas acentuadas nos momentos de limpeza de *cached*, ou seja, nos minutos 2873 e 5726, quando os valores caíram para 564 MB e 539 MB, respectivamente. O monitoramento do consumo de *cached* parou no minuto 7839.

Com relação consumo de VSZ no serviço do *Containerd*, o mesmo permaneceu acima

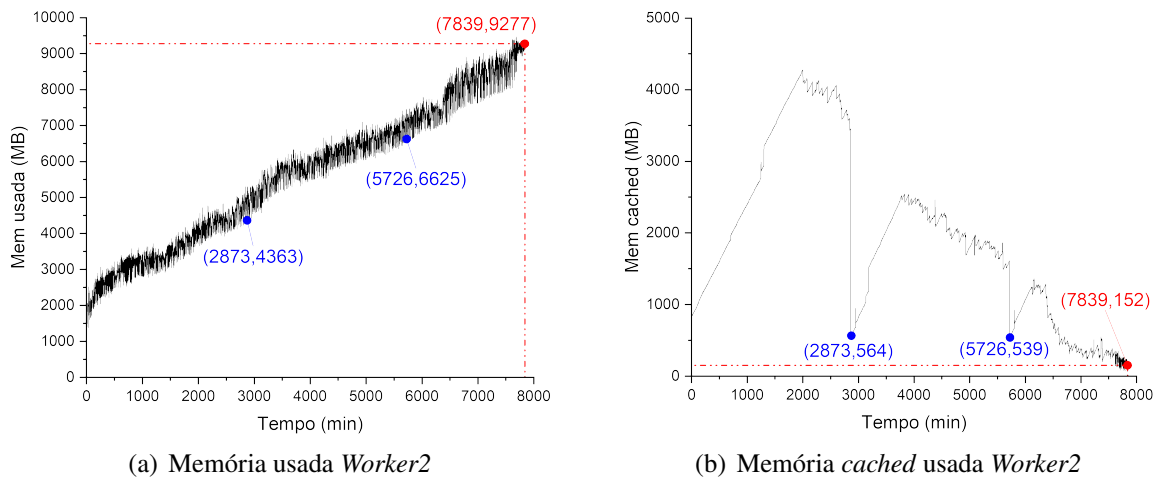


Figura 20 – Cenário 2 - Limpeza de *caches* e *buffers* (Ubuntu 20).

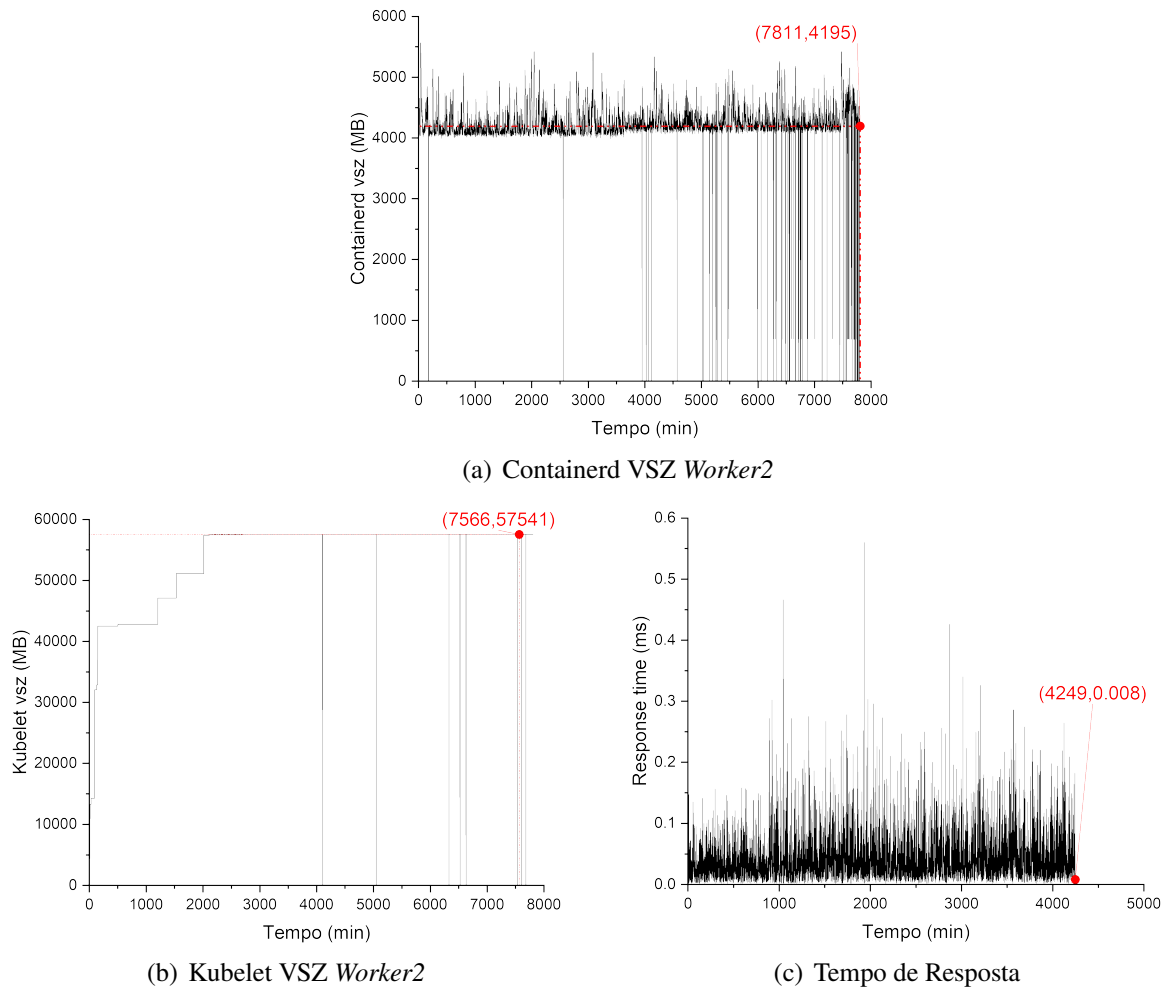
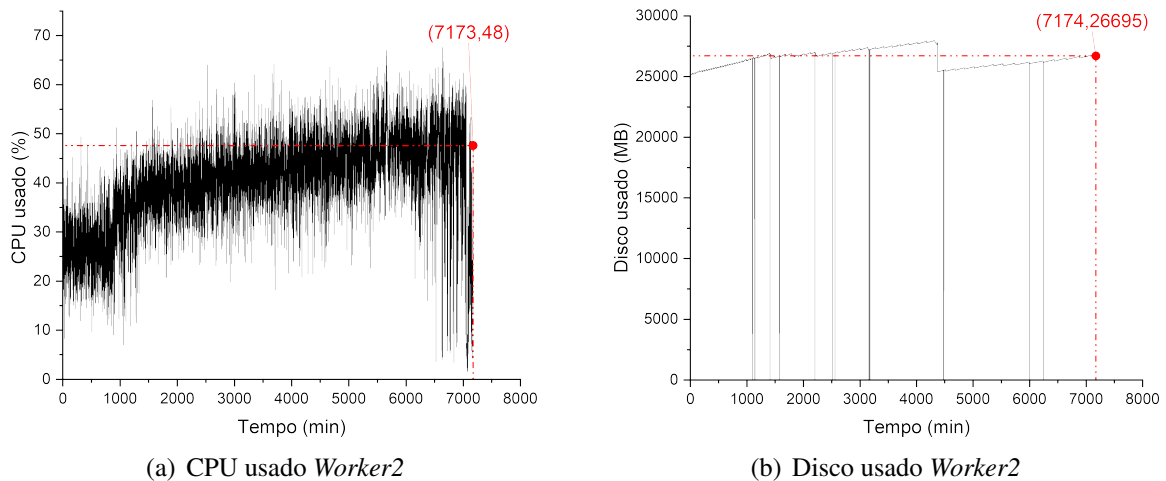
dos 4000 MB, com variações que chegaram a zero em alguns pontos e picos acima de 5500 MB, conforme apresentado na Figura 21(a). De maneira semelhante ao que foi analisado no Cenário #1, a Figura 21(b) demonstra que o consumo do *Kubelet* atingiu picos que excederam a capacidade do *Cluster*, mantendo uma constância de consumo em torno de 57541 MB.

Ao se analisar o tempo de resposta para o acesso externo ao *Cluster*, o gráfico da Figura 21(c) revela picos que ultrapassaram 0,50 ms e também indica que o pod “portal de notícias” deixou de responder no minuto 4249 à medida que o tempo avançava.

No **Cenário #2**, os gráficos relacionados ao uso do Ubuntu 22, especificadamente no *Worker2*, foram analisados, constatando um comportamento distinto, embora o *Cluster* também tenha parado de funcionar antes do previsto. A Figura 22(a) mostra um consumo de CPU mais elevado do que o gráfico da Figura 19(a), com picos que ultrapassaram os 65%, encerrando o monitoramento no minuto 7174. O consumo de disco foi ainda mais acentuado em comparação com o mesmo cenário no Ubuntu 20, como demonstrado na Figura 22(b). O *Worker2* iniciou o consumo de disco ligeiramente acima dos 25000 MB e, ao cessar no minuto 7174, o consumo havia atingido 26695 MB.

No que diz respeito ao consumo de memória, conforme pode ser observado na Figura 23(a), o *Worker2* apresentou pouca oscilação nos estágios iniciais. Pouco antes dos 1000 minutos, o uso de memória aumentou consideravelmente. Nos momentos 2828 e 5439 minutos, ocorreu a limpeza da memória *cached*. No primeiro momento, houve até um aumento no consumo após a limpeza, enquanto no segundo momento não houve oscilações significativas. O *Worker2* encerrou seu monitoramento no minuto 7174, com um consumo de memória de 9506 MB.

A Figura 23(b) apresenta o consumo de memória *cached*, com um pico de mais de 5500 MB. O consumo tende a diminuir devido à memória usada pelo *Worker2*, e nos minutos 2828 e 5439, ocorreu a limpeza da memória *cached*. O consumo de memória *cached* continuou a diminuir, e o monitoramento cessou no minuto 7174.

Figura 21 – Cenário 2 - Limpeza de *caches* e *Buffers* (Ubuntu 20).Figura 22 – Cenário 2 - Limpeza de *caches* e *Buffers* (Ubuntu 22).

O consumo de VSZ no serviço do *Containerd* ultrapassou os 8500 MB e o monitoramento do processo foi interrompido no minuto 6110, conforme apresentado na Figura 24(a). Enquanto isso, o monitoramento do serviço do *Kubelet* apresentou um aumento significativo no consumo

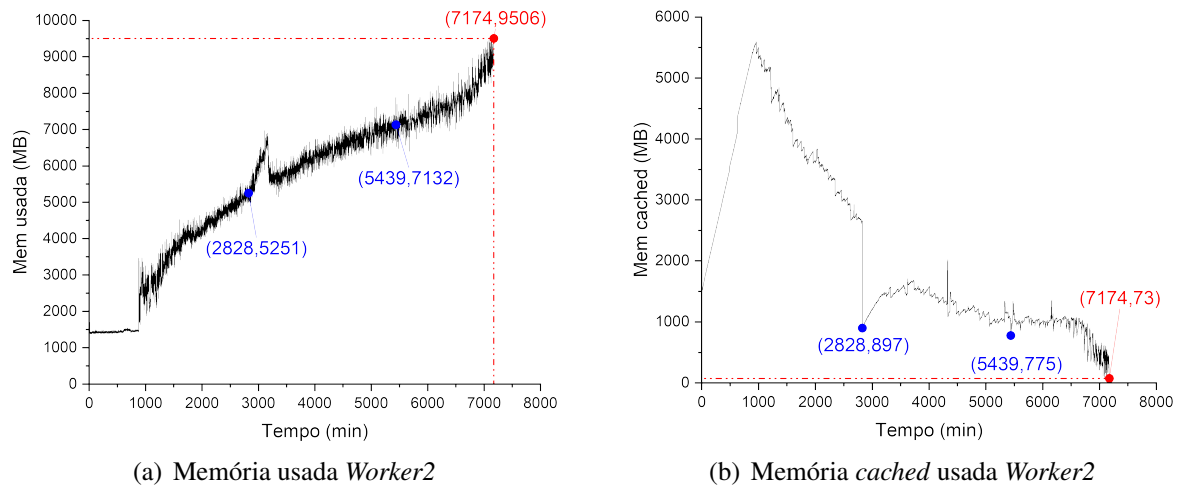


Figura 23 – Cenário 2 - Limpeza de *caches* e *Buffers* (Ubuntu 22).

de memória virtual (VSZ), e o processo foi encerrado no minuto 7123, com um consumo de memória de 6192 MB, conforme pode ser visualizado na Figura 24(b).

No tocante ao tempo de resposta para o acesso externo ao *Cluster*, o gráfico 24(c) revelou picos que ultrapassaram 0,50 ms e indicou que o pod “portal de notícias” parou de responder no minuto 4234 à medida que o tempo avançava.

O **Cenário #2** buscou avaliar a eficácia da limpeza de memória *cache* e *buffer* como estratégia de rejuvenescimento de software, aplicando-a durante a execução da carga de estresse. No *Worker2* com Ubuntu 20, o *Cluster* não atingiu os 10 dias de carga, apresentando um aumento gradual no uso de CPU e disco. A limpeza de *cached* não reverteu completamente o aumento no uso de memória, resultando em falha no minuto 7839. No Ubuntu 22, o *Worker2* também falhou antes do previsto, destacando os desafios na estratégia de limpeza de memória. O estudo indica a necessidade de aprimoramentos para rejuvenescimento de software em ambientes de longa duração.

5.2.3 Cenário #3 - Reinício do Serviço *Containerd* como Estratégia de Rejuvenescimento

O **Cenário #3** é voltado para o *Containerd*. Conforme está evidenciado nas Figuras 15(a) e 18(a) do Cenário #1, o *Containerd* é um dos principais processos envolvidos no funcionamento do sistema e está suscetível ao envelhecimento. Para avaliar seu rejuvenescimento, foi executada a reinicialização do serviço do *Containerd*, sem pausas de inatividade no *Cluster*. Foi desenvolvido um *script* para reinicializar o serviço em intervalos de aproximadamente 2 dias. Essa abordagem visa otimizar o uso da memória útil, com o propósito de prolongar a disponibilidade do *Cluster*.

É possível observar na Figura 25(a) um significativo aumento no consumo de CPU, com picos que ultrapassaram os 50%, encerrando o monitoramento no minuto 6725. O consumo de disco iniciou ligeiramente acima de 16000 MB e aumentou para 17123 MB no minuto 6724,

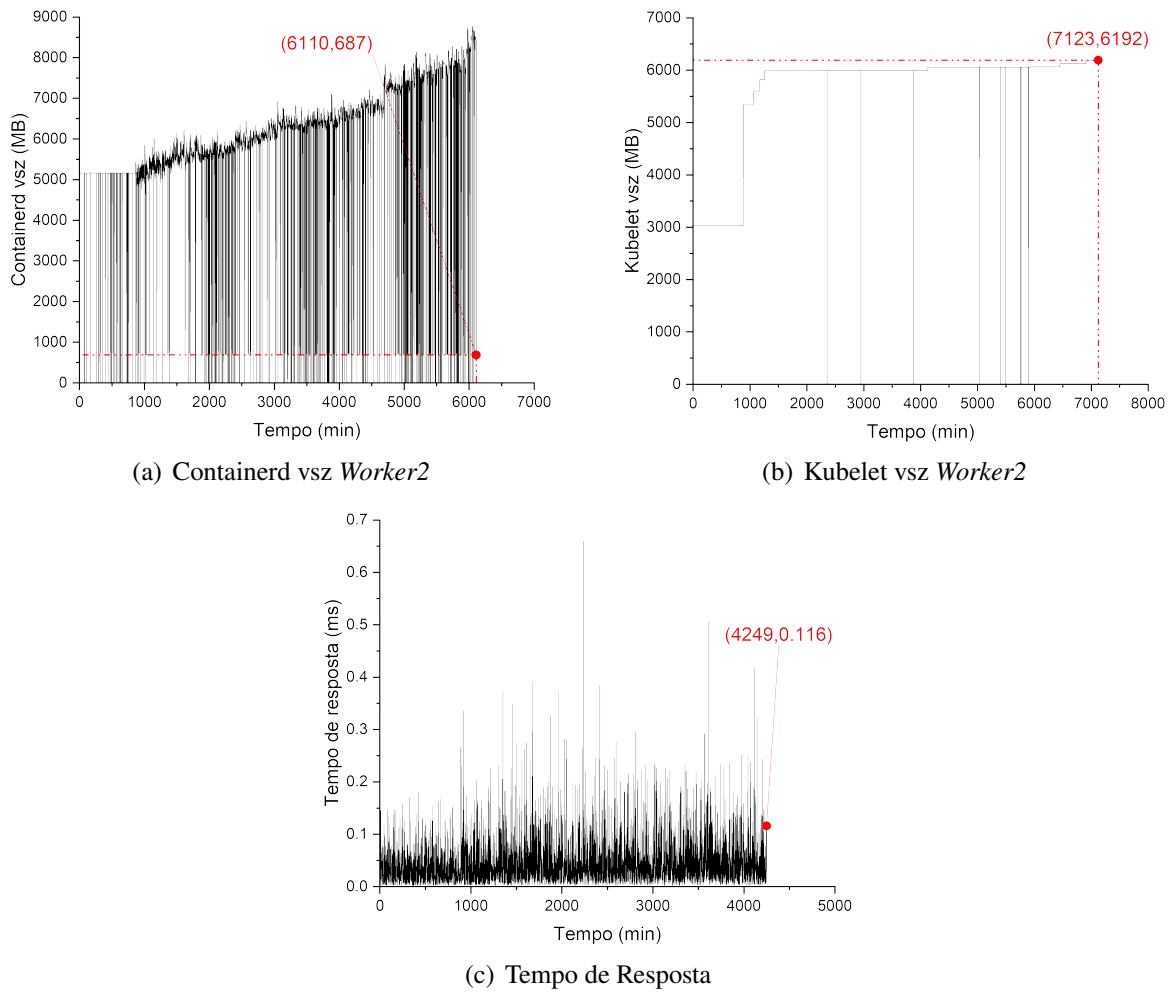


Figura 24 – Cenário 2 - Limpeza de *caches* e *Buffers* (Ubuntu 22).

conforme ilustrado na Figura 25(b), marcando o término do monitoramento.

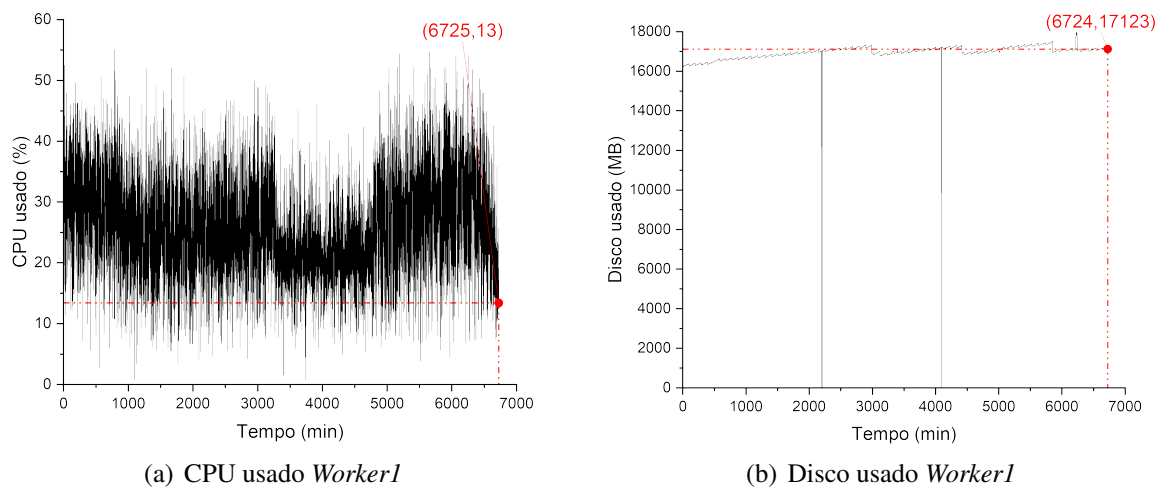


Figura 25 – Cenário 3 - Reiniciar o Serviço *Containerd* (Ubuntu 20).

O *Worker1* iniciou o consumo de memória com pouco mais de 2000 MB, e esse consumo aumentou à medida que o experimento era executado. Nos momentos de reinicialização do

serviço *Containerd* (minutos 2810 e 5538), não foram observadas diminuições na memória utilizada. O *Worker1* encerrou o monitoramento de memória no minuto 6724, consumindo 9343 MB, conforme apresentado na Figura 26(a).

No gráfico referente ao consumo de memória *cached*, apresentado na Figura 26(b), foi evidenciada uma elevação no consumo de memória *cached* que ultrapassou os 4000 MB nos primeiros 2000 minutos. Após esse período, há uma tendência de queda, chegando a 171 MB no minuto 6615, momento em que o *Worker1* encerra sua execução.

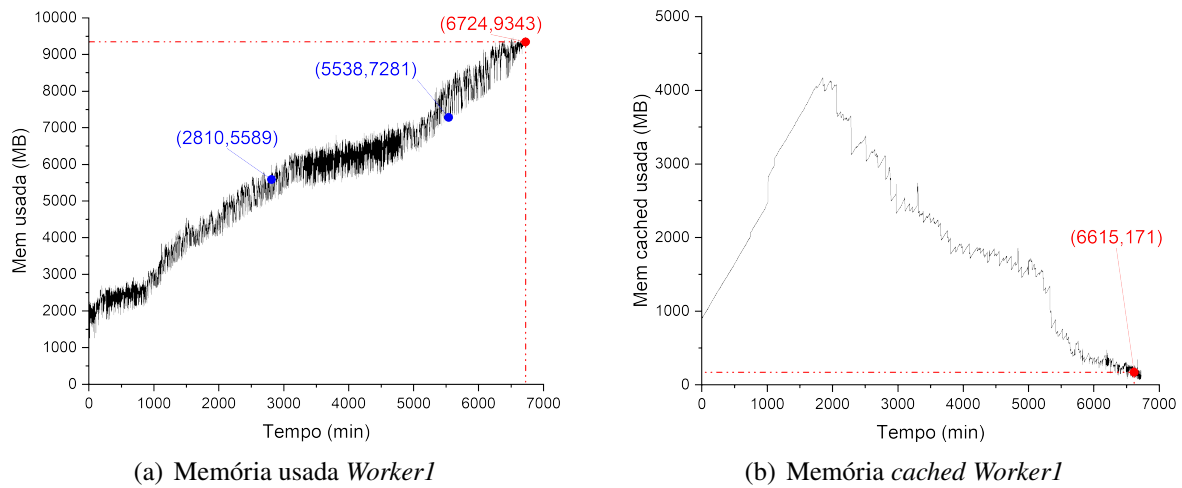


Figura 26 – Cenário 3 - Reiniciar o Serviço *Containerd* (Ubuntu 20).

Ao analisar o reinício do serviço do *Containerd*, foi monitorado o consumo de VSZ no serviço e observou-se um comportamento de consumo uniforme variando entre 4000 e 5000 MB, com alguns picos ultrapassando os 5000 MB e algumas quedas chegando a zero. O *Worker1*, onde o processo foi monitorado, interrompeu sua coleta no minuto 6658, com um consumo de 4295 MB, conforme mostrado na Figura 27(a).

Durante o processo de reinicialização do serviço do *Containerd*, não foram observados comportamentos distintos no *Kubelet* durante esse período, conforme evidenciado na Figura 27(b). De maneira similar ao que foi analisado no *Cenário #1*, o consumo do *Kubelet* atingiu picos que ultrapassaram a capacidade do *Cluster*, mantendo uma constância de consumo em torno de 64124 MB de memória.

Quanto ao tempo de resposta para acessar externamente o *Cluster*, a Figura 27(c) apresenta picos que excederam 1,50 ms, indicando que o pod “portal de notícias” cessou sua capacidade de resposta no minuto 2964, à medida que o tempo progredia.

Utilizando o Ubuntu 22 no *Cenário 3*, durante a reinicialização do serviço do *Containerd*, ficou notável que o consumo de CPU no *Worker1* teve uma oscilação inicial entre 0% e 50%. Posteriormente, houve um aumento nessa porcentagem, com picos que se aproximaram de 65%, encerrando seu monitoramento no minuto 7727 com 57% de processamento, conforme apresentado na Figura 28(a). Em comparação com o mesmo cenário no Ubuntu 20, o consumo

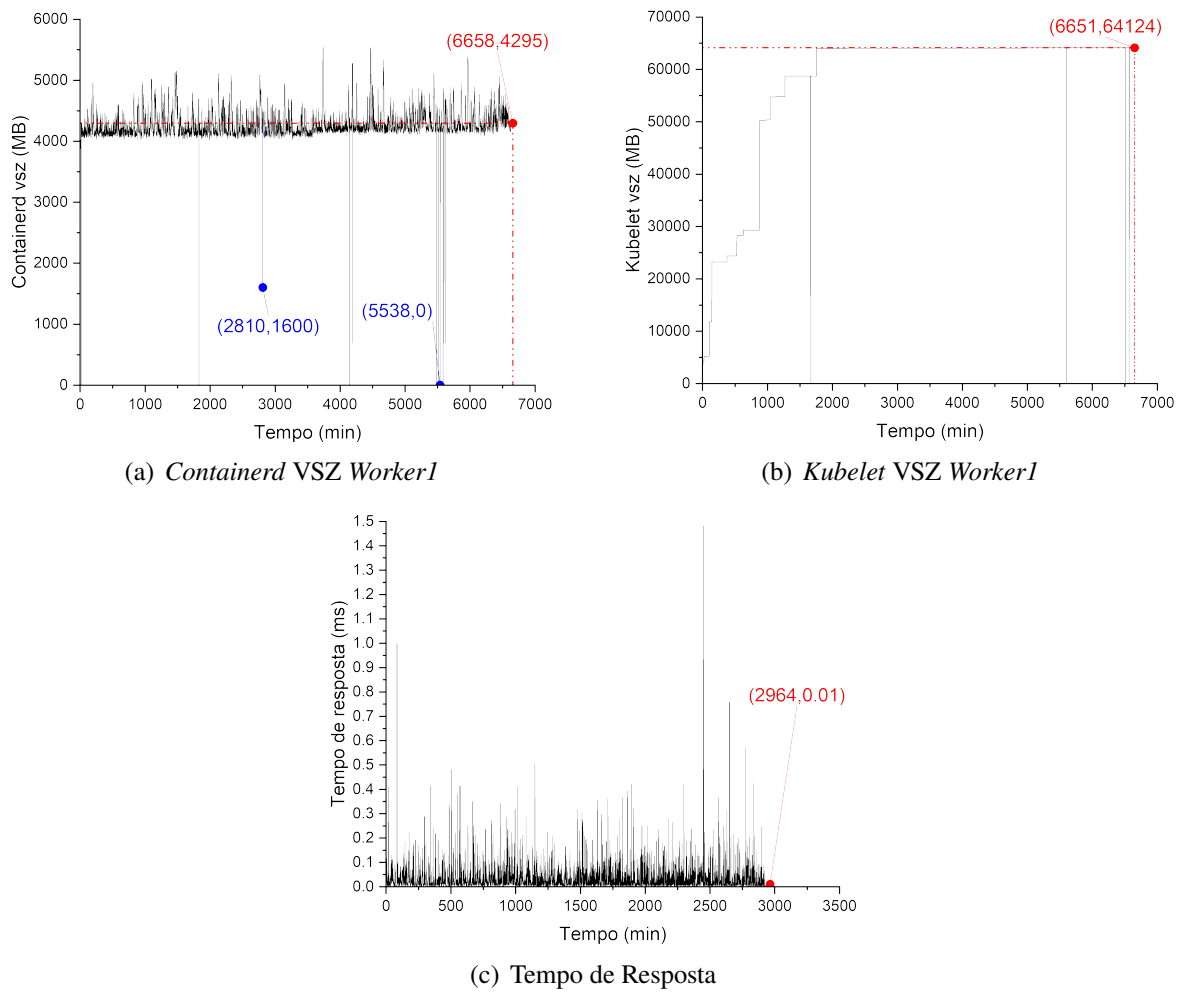


Figura 27 – Cenário 3 - Reiniciar o Serviço *Containerd* (Ubuntu 20).

de disco foi ainda mais acentuado no consumo, como evidenciado na Figura 28(b). O *Worker1* iniciou o consumo de disco abaixo dos 25000 MB e, ao cessar no minuto 7827, o consumo havia atingido 27548 MB.

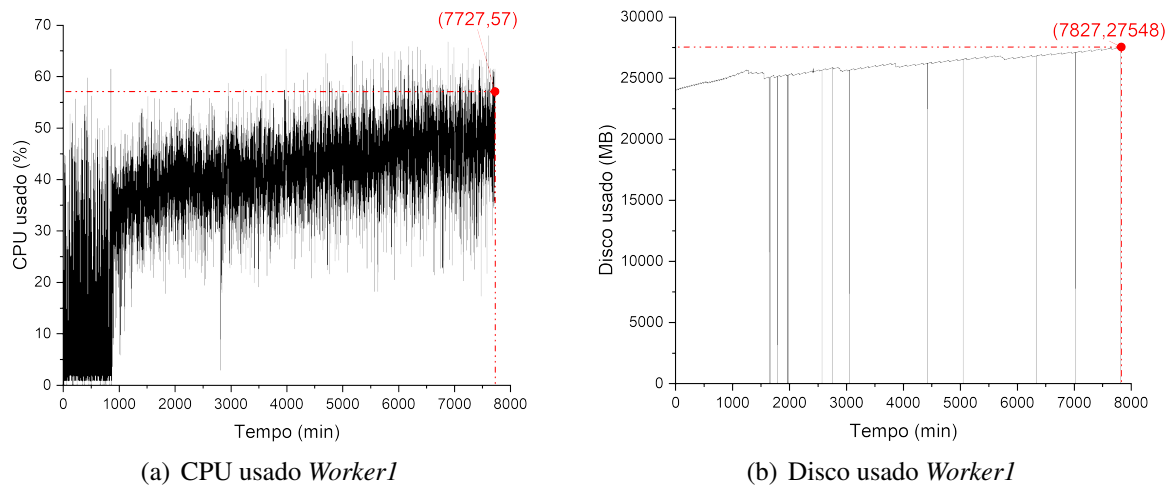


Figura 28 – Cenário 3 - Reiniciar o Serviço *Containerd* (Ubuntu 22).

Em relação ao consumo de memória, nos estágios iniciais, o *Worker1* apresentou pouca oscilação, conforme apresentado na Figura 29(a). Pouco antes dos 1000 minutos, o uso de memória aumentou consideravelmente. Nos momentos 2835 e 5650 minutos, ocorreu a reinicialização do serviço do *Containerd*, havendo uma pequena variação no uso de memória. Contudo, não impedindo o crescimento do uso, muito menos a parada do *Worker1* no minuto 7727 com um consumo de 7232 MB de memória.

No gráfico de consumo de memória *cached*, apresentado na Figura 29(b), foi evidenciado um aumento no consumo de memória *cached* que ultrapassou os 6000 MB pouco depois dos 1000 primeiros minutos. Após esse período, há uma tendência de queda, chegando a 1238 MB no minuto 7727, momento em que o *Worker1* encerra sua execução.

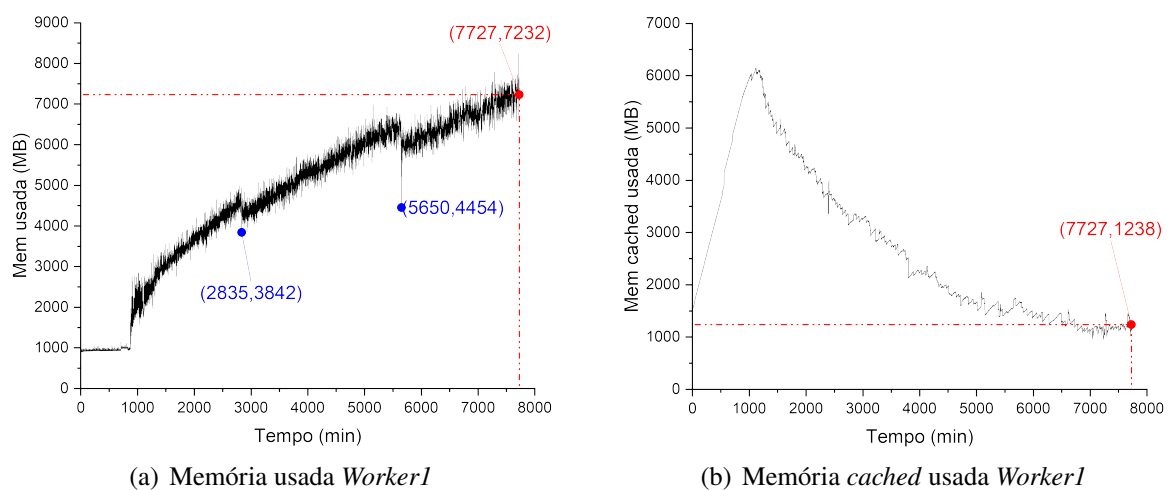


Figura 29 – Cenário 3 - Reiniciar o Serviço *Containerd* (Ubuntu 22).

A Figura 30(a) apresenta o consumo do serviço do *Containerd* em relação ao uso de memória VSZ. Nele, é mostrado um comportamento até pouco antes dos 1000 minutos, com consumo próximo de 5500 MB. Posteriormente, ocorreram variações de subidas e descidas, e nos minutos 2835 e 5650, uma queda no uso de memória VSZ, indicando algum efeito na reinicialização do processo, fazendo com que o processo postergue o consumo mais rápido de memória, porém não impedindo que o *Cluster* falhe no minuto 7668 com 9031 MB de uso de memória VSZ.

Durante o processo de reinicialização do serviço do *Containerd*, o consumo de memória do processo *Kubelet* não apresentou oscilações significativas. Ele manteve-se pouco acima dos 2000 MB, experimentando uma elevação até atingir 6000 MB e variando posteriormente entre 6000 MB e 6395 MB. O monitoramento foi encerrado no minuto 7667, conforme evidenciado na Figura 30(b).

No contexto do tempo de resposta para o acesso externo ao *Cluster*, a Figura 30(c) revelou picos que ultrapassaram 0,50 ms, indicando uma interrupção na capacidade de resposta do pod “portal de notícias” no minuto 2993, à medida que o tempo avançava.

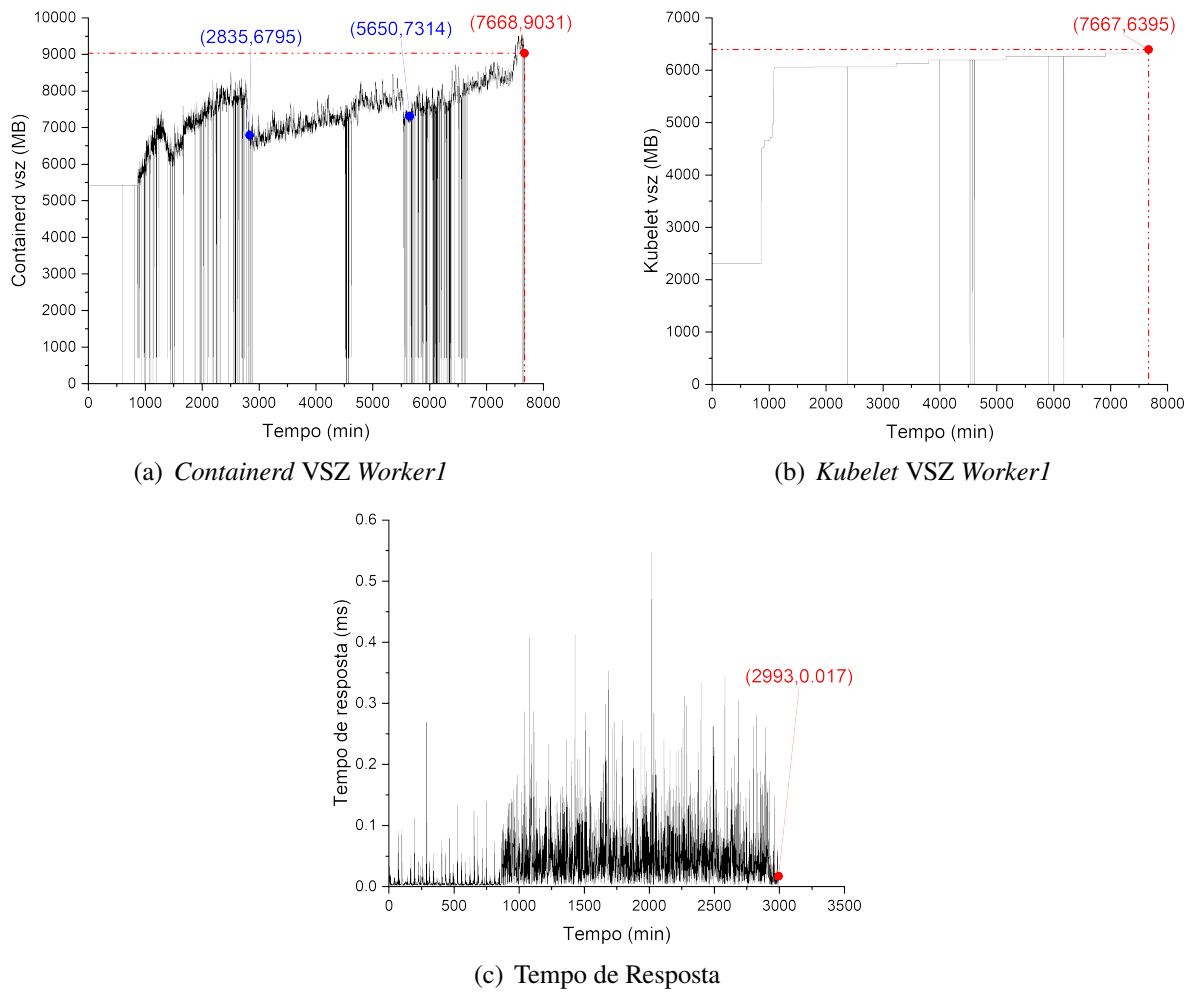


Figura 30 – Cenário 3 - Reiniciar o Serviço *Containerd* (Ubuntu 22).

Os resultados do **Cenário #3**, focados no reinício do *Containerd*, revelaram desafios, pois não conseguiram conter o aumento no uso de CPU, memória e disco ao longo do tempo para ambos os *clusters*. No *Cluster* Ubuntu 20, o comportamento do *Kubelet* continuou a consumir uma quantidade excessiva de memória VSZ. Mesmo com tentativas de rejuvenescimento através dos reinícios do *Containerd*, as ações não foram totalmente eficazes em evitar falhas. No Ubuntu 22, observamos flutuações no uso de memória durante os reinícios do *Containerd*, no entanto, seu crescimento consistente, mesmo com reinícios, levou à interrupção do *Worker1*, destacando os desafios em mitigar o envelhecimento de software. Ambos os ambientes indicam que o *Containerd* desempenha um papel crítico no sistema e é suscetível ao envelhecimento. A estratégia de reinicialização do serviço, embora tenha efeitos variados, não resolve completamente os problemas.

5.2.4 Cenário #4 - Reinicialização do Serviço *Kubelet* como Estratégia de Rejuvenescimento

No **Cenário #4**, o ponto principal foi *Kubelet*, sendo suscetível ao envelhecimento, como evidenciado nas Figuras 15(b) e 18(b). Responsável pela comunicação entre *Master* e *Workers*, o mesmo ciclo de testes do **Cenário #1** sem pausas no *Cluster*. Um *script* foi desenvolvido com a finalidade de reiniciar periodicamente o serviço do *Kubelet*, configurado para operar a cada aproximadamente 2 dias, com a intenção de promover rejuvenescimento.

No **Cenário #4**, ao reiniciar o serviço do *Kubelet*, o *Cluster* não conseguiu atingir o objetivo de carga de 10 dias, conforme pode ser observado nas Figuras 31, 32 proveniente do *Worker3*. A Figura 31(a) apresenta o comportamento da CPU durante o experimento, com picos que ultrapassaram os 50%, e a conclusão da atividade no minuto 6413. O consumo de disco começou acima de 16000 MB, alcançando 16969 MB no minuto 6413, marcando o fim do monitoramento, como apresentado na Figura 31(b).

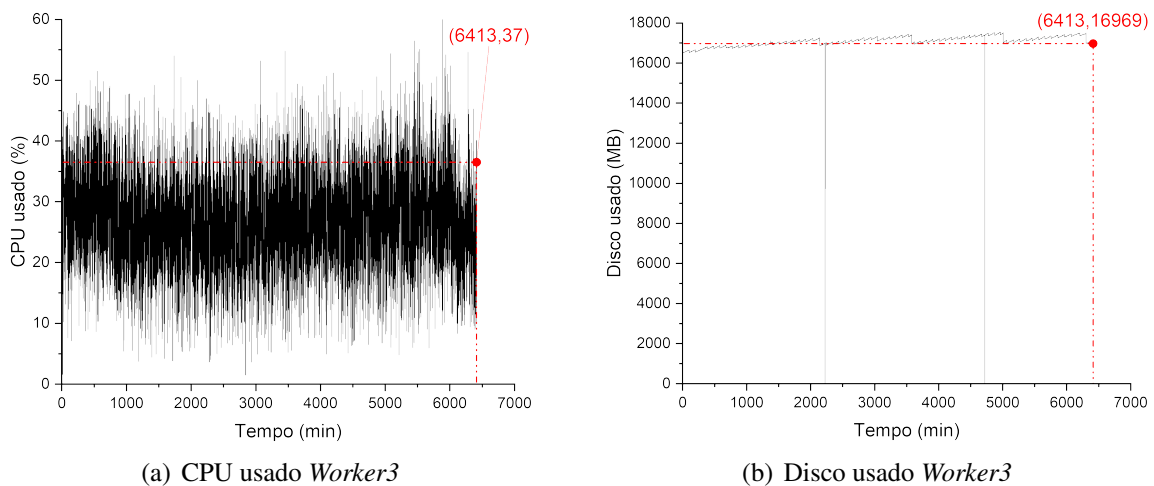


Figura 31 – Cenário 4 - Reiniciando o Serviço *Kubelet* (Ubuntu 20).

O *Worker3* iniciou o consumo de memória com pouco mais de 2000 MB, experimentou um aumento progressivo e, nos momentos de reinicialização do serviço do *Kubelet* (minutos 2819 e 5618), houve uma leve diminuição na memória utilizada. A Figura 32(a) demonstra que o consumo de memória no *Worker3* continuou subindo até o momento de parada do *Cluster* no minuto 6412, consumindo 9236 MB de memória.

No gráfico que representa o consumo de memória *cached* (Figura 32(b)), nota-se um crescimento no uso de memória *cached*, que ultrapassou os 4000 MB pouco após os 2000 minutos iniciais. Posteriormente, há uma tendência de declínio, chegando a 156 MB no minuto 6412, quando o *Worker3* encerra sua execução.

Em relação ao consumo de memória no serviço do *Kubelet*, o consumo de VSZ foi monitorado e observado, evidenciando um comportamento que excedia o valor máximo fornecido pelo *Worker3* (10000 MB), ultrapassando valores de 60000 MB. No momento do reinício do

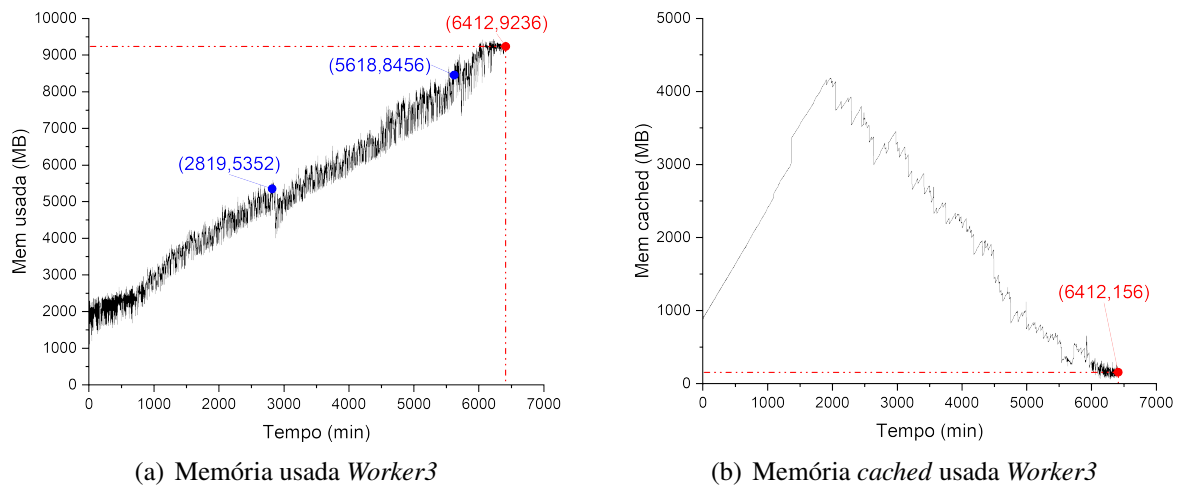


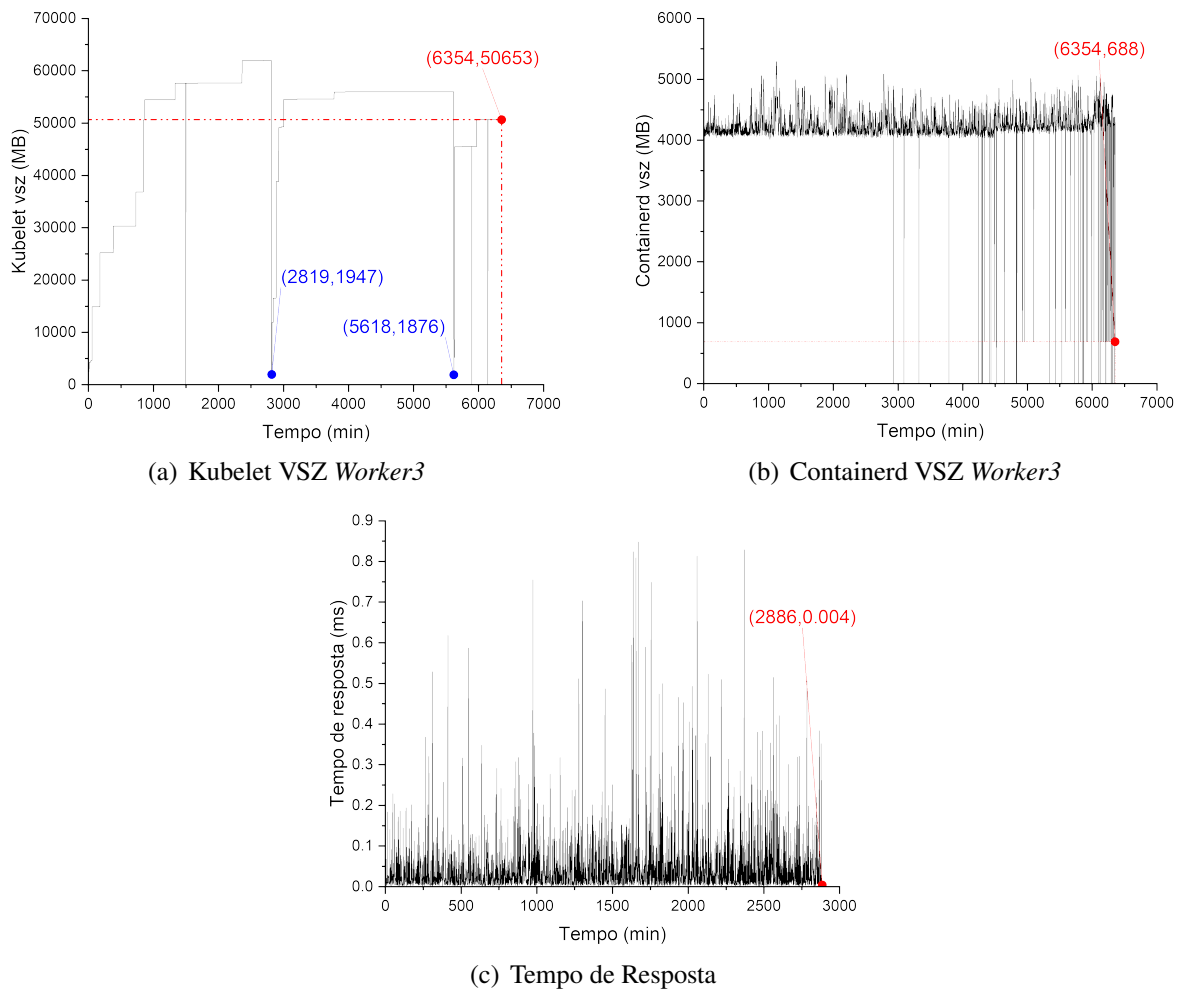
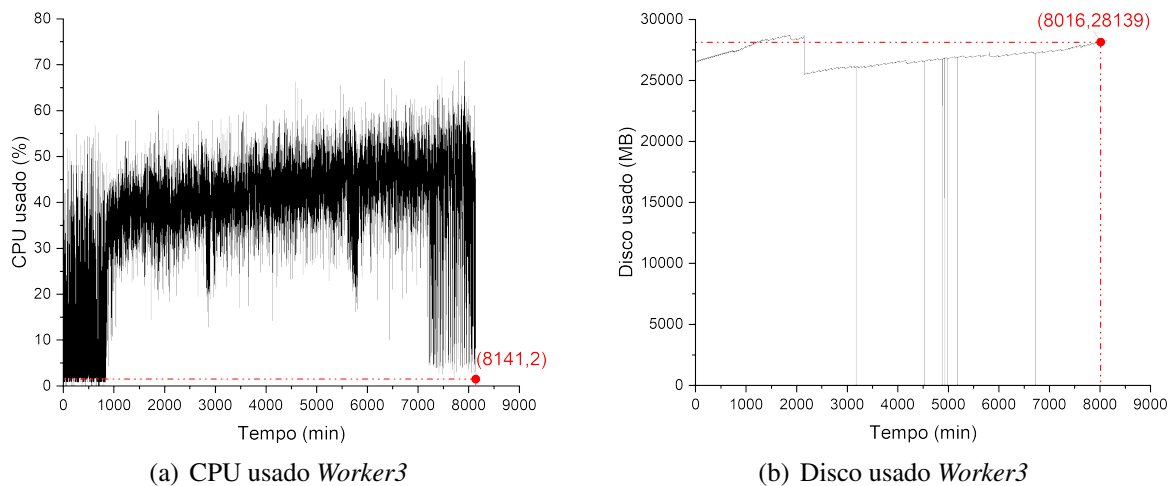
Figura 32 – Cenário 4 - Reiniciando o Serviço *Kubelet* (Ubuntu 20).

serviço (minutos 2819 e 5618), houve uma diminuição do uso de memória VSZ, que voltou a aumentar em questão de minutos. Houve também algumas quedas na utilização da memória VSZ, com seu consumo chegando a 0 MB. No *Worker3*, onde o processo foi monitorado, a coleta de *logs* foi interrompida no minuto 6354, com um consumo de 50653 MB. Esse processo pode ser visto no gráfico presente Figura 33(a).

Durante a reinicialização do serviço do *Kubelet*, o processo do *Containerd* apresentou uma consistência acima dos 4000 MB, com algumas oscilações que ultrapassaram os 5000 MB, conforme destacado na Figura 33(b). Em relação ao tempo de resposta para acesso externo do *Cluster*, a partir da Figura 33(c), verificou-se picos que excederam 0.8 ms. A partir da Figura 33(b) pode-se constatar também que o pod “portal de notícias” cessou sua capacidade de resposta no minuto 2886, à medida que o tempo progredia.

Com o Ubuntu 22 no **Cenário #4**, durante a reinicialização do serviço do *Kubelet*, executou-se uma análise dos gráficos (34, 35 e 36), gerados a partir do *Worker3*. No âmbito do consumo de CPU, visualizou-se uma oscilação inicial entre 0% e 50%. Posteriormente, essa porcentagem aumentou, atingindo picos próximos a 60%, encerrando seu monitoramento no minuto 8141 com 2% de processamento, conforme evidenciado no gráfico 34(a). No que tange ao consumo de disco, visualizado no gráfico 34(b), o *Worker3* iniciou o consumo acima dos 25000 MB e, ao cessar no minuto 8016, o consumo havia atingido 28139 MB.

Em relação à memória, nos estágios iniciais, o *Worker3* apresentou uma utilização reduzida, conforme demonstrado no gráfico 35(a). Pouco antes dos 1000 minutos, o uso de memória aumentou consideravelmente. Nos momentos 2804 e 5549 minutos, ocorreu a reinicialização do serviço do *Kubelet*, resultando em uma pequena queda no uso de memória. No entanto, isso não impediu o crescimento do uso, e o *Worker3* parou de funcionar no minuto 8141 com um consumo de 8891 MB de memória. No gráfico que ilustra o consumo de memória *cached* (Figura 35(b)), notou-se um incremento no uso de memória *cached* que ultrapassou os

Figura 33 – Cenário 4 - Reiniciando o Serviço *Kubelet* (Ubuntu 20).Figura 34 – Cenário 4 - Reiniciando o Serviço *Kubelet* (Ubuntu 22).

5000 MB pouco depois dos 1000 minutos iniciais. Subsequentemente, observa-se uma tendência de queda, atingindo 428 MB no minuto 8141, quando o *Worker3* conclui sua execução.

A Figura 36(a) apresenta o consumo de memória VSZ do serviço do *Kubelet*, podendo

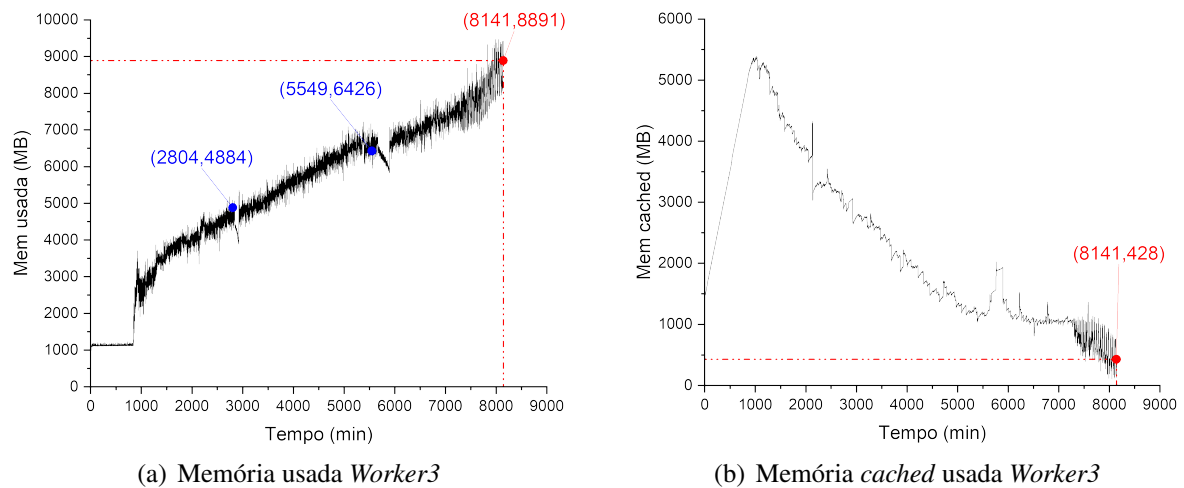


Figura 35 – Cenário 4 - Reiniciando o Serviço *Kubelet* (Ubuntu 22).

observar um comportamento até pouco antes dos 1000 minutos, com um consumo próximo de 2000 MB. Posteriormente, houve um aumento que ultrapassou os 5000 MB, e nos minutos 2804 e 5549 houve uma queda no uso de memória VSZ, indicando algum impacto na reinicialização do processo. No entanto, o consumo rapidamente voltou a subir, não impedindo a falha do *Cluster* no minuto 8073 com 5893 MB de uso de memória VSZ.

Durante a reinicialização do serviço do *Kubelet*, o processo do *Containerd* não apresentou nenhuma mudança em seu comportamento, sempre com tendência de subida e interrompendo o monitoramento no minuto 8066, com consumo de 14066 MB de Memória VSZ, conforme destacado na Figura 36(b).

No concerne ao tempo de resposta para o acesso externo ao *Cluster*, a Figura 36(c) mostra picos que ultrapassaram 0,4 ms, indicando uma interrupção na capacidade de resposta do pod “portal de notícias” no minuto 2955, à medida que o tempo avançava.

No **Cenário #4**, a ênfase principal estava no reinício do serviço *Kubelet*. Reinicializações programadas a cada 2 dias não permitiram que os *Clusters* atingissem a carga contínua desejada de 10 dias em ambos os sistemas operacionais. No Ubuntu 20, *Worker3* apresentou picos de CPU, um aumento significativo no consumo de disco e memória. O *Kubelet* continuou consumindo acima do que o cluster poderia fornecer, e o *Containerd* não foi afetado pelo reinício do serviço. No Ubuntu 22, *Worker3* exibiu flutuações no consumo de CPU, memória e disco. O *Kubelet* se comportou de forma semelhante aos outros cenários, embora os reinícios aumentassem o tempo de atividade do cluster. Embora a estratégia de reiniciar o serviço tenha efeitos diversos, ela não aborda totalmente os problemas.

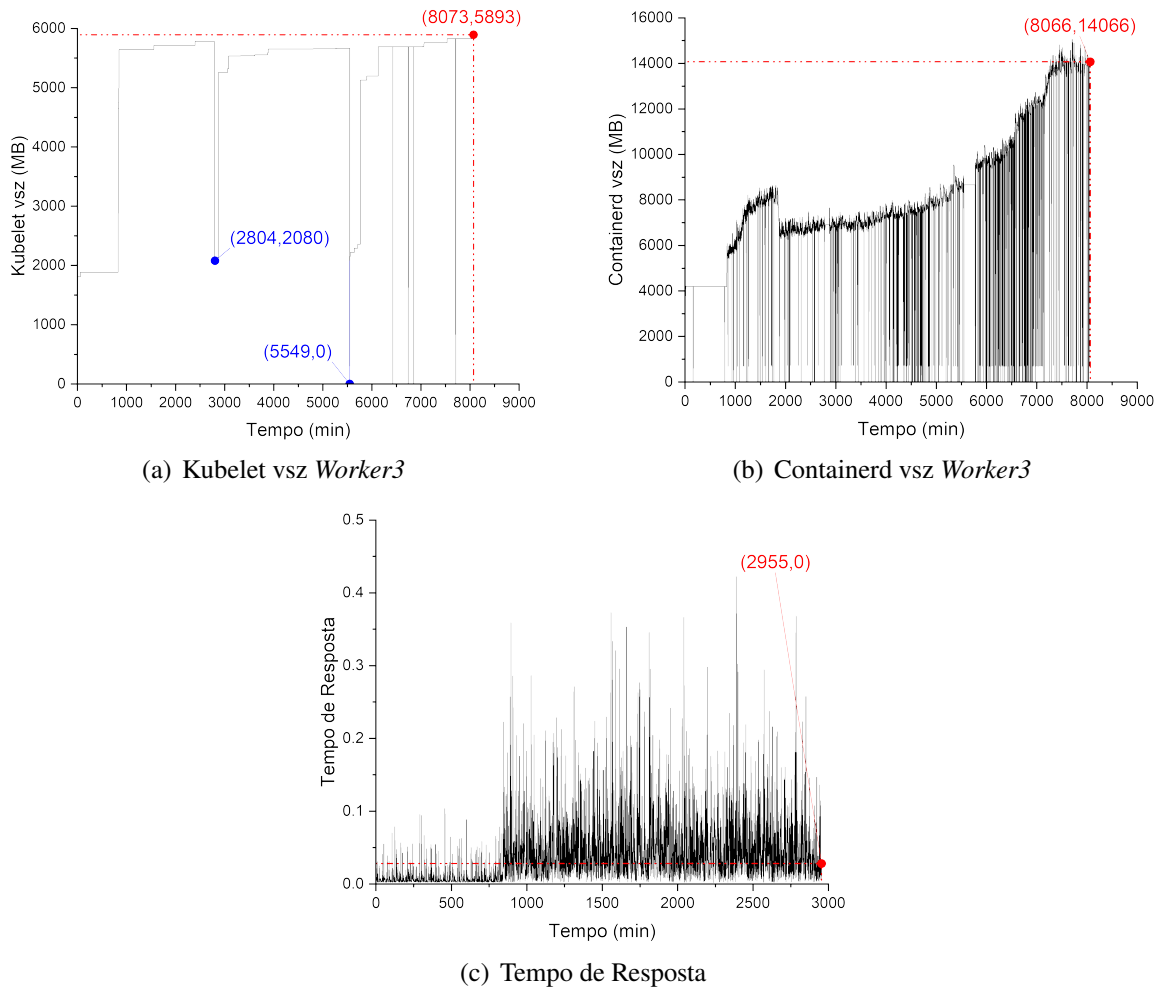


Figura 36 – Cenário 4 - Reiniciando o Serviço *Kubelet* (Ubuntu 22).

5.2.5 Cenário #5 - Reinicialização dos Nós de *Clusters* como Estratégia de Rejuvenescimento

As estratégias de rejuvenescimento de software anteriores não alcançaram os resultados desejados. Portanto, no **Cenário #5**, foi proposto realizar as reinicializações controladas dos nós *Workers*, tanto para o *Cluster* operando com Ubuntu 20 quanto para o com Ubuntu 22. Cada um dos três nós (*Worker1*, *Worker2* e *Worker3*) passou por reinicializações controladas, sendo executadas em intervalos de 5 minutos entre elas. Essas reinicializações replicaram a mesma carga de estresse do **Cenário #1** sem interromper as atividades do *Cluster*. Utilizando o sistema de agendamento de tarefas do Linux, o *crontab*, foram configuradas tarefas específicas para realizar reinicializações individuais em cada nó *Worker*. O padrão de intervalos segue aproximadamente 1 dia para a primeira reinicialização, em torno de 1,5 dia para a segunda, cerca de 2 dias para a terceira e, por fim, aproximadamente 2 dias para a quarta reinicialização. Esta estratégia foi selecionada com base em testes prévios conduzidos nos *Clusters*, com a finalidade de avaliar os impactos desse processo de rejuvenescimento de software em diferentes estágios de tempo no envelhecimento de software.

No **Cenário #5**, será apresentado o comportamento apenas do *Worker2*, devido à sua semelhança com os demais nós (*Worker1* e *Worker3*), conforme realizado nos cenários anteriores. No Ubuntu 20 (ver Figuras 37, 38, 39) o monitoramento da utilização da CPU, apresentado na Figura 37(a), revelou picos de processamento que excederam os 55%. Em relação ao uso do disco, apresentado na Figura 37(b), observou-se um leve aumento, começando em aproximadamente 17500 MB no minuto zero e chegando a 17586 MB no minuto 11239, quando o experimento foi concluído.

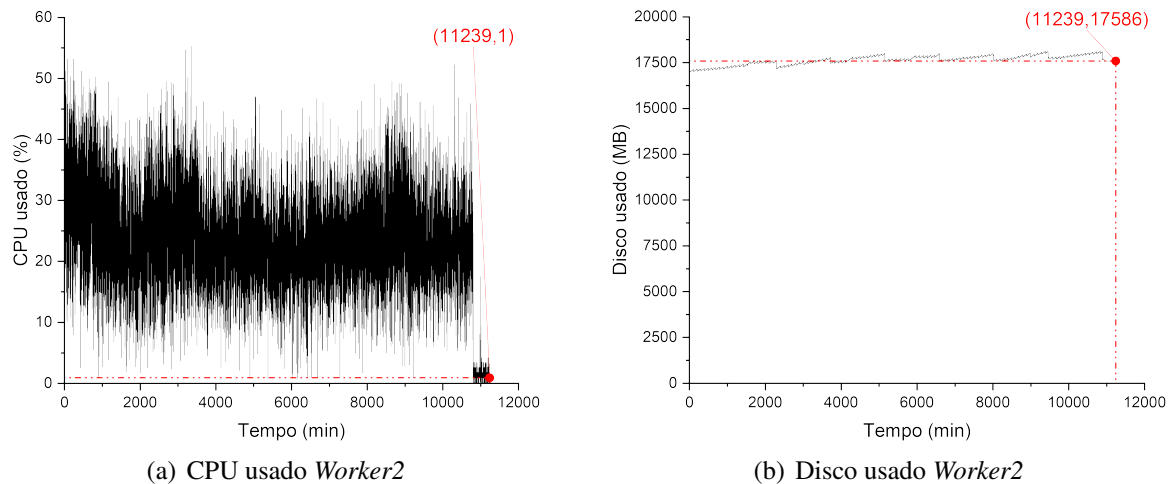


Figura 37 – Cenário 5 - Reiniciando os Nós *Workers* (Ubuntu 20).

A utilização de memória, apresentada na Figura 38(a), mostra uma trajetória ascendente, com quedas nos minutos 1432, 3583, 6440 e 9296, coincidindo com os momentos de reinicialização do sistema operacional. A coleta de dados de memória foi interrompida no minuto 11239, com um consumo total de 2825 MB de memória utilizada, momento em que o *Cluster* finalizou o experimento.

O consumo de memória *cached*, conforme apresentado no gráfico (Figura 38(b)), revela aumentos no uso de memória *cached* que são interrompidos nos momentos de reinicialização do *Worker2*. Esse comportamento cessa no minuto 11239, consumindo 3582 MB quando o *Worker2* conclui o cenário.

No que diz respeito ao consumo de VSZ no serviço do *Containerd*, manteve-se acima dos 4000 MB, com variações que atingiram zero em alguns picos acima de 5500 MB, conforme apresentado na Figura 39(a). De maneira análoga ao que foi analisado no **Cenário #1**, o consumo do *Kubelet* atingiu picos que ultrapassaram a capacidade do *Cluster*, mantendo um padrão de consumo acima de 50000 MB, conforme apresentado na Figura 39(b).

Em relação ao tempo de resposta para o acesso externo ao *Cluster*, a Figura 39(c) revela picos que ultrapassaram 1,4 ms e também indica que o pod “portal de notícias” continuou respondendo até o minuto 11489, mesmo após a conclusão do experimento pelo *Cluster* no minuto 11239.

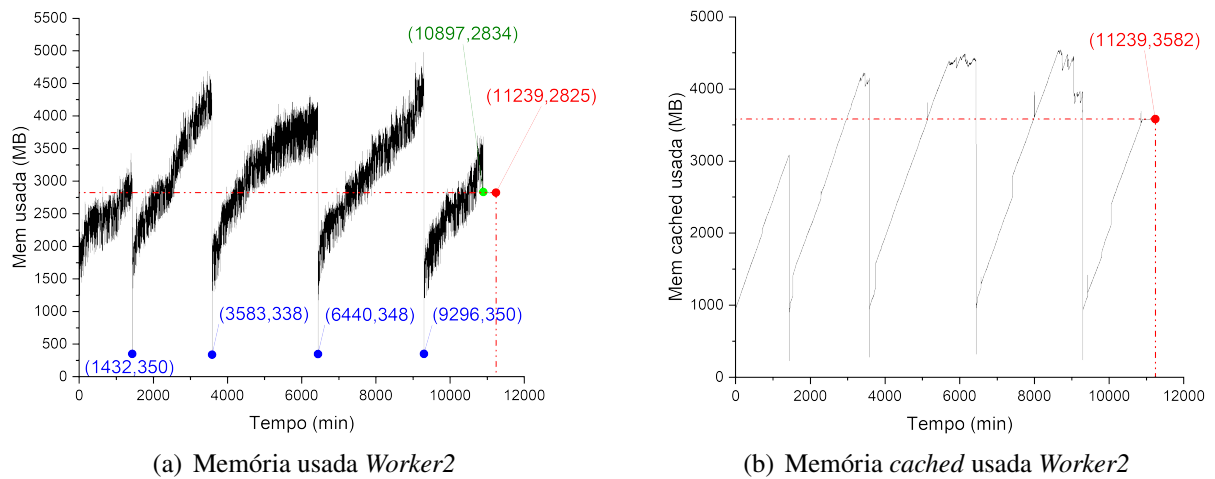


Figura 38 – Cenário 5 - Reiniciando os Nós Workers (Ubuntu 20).

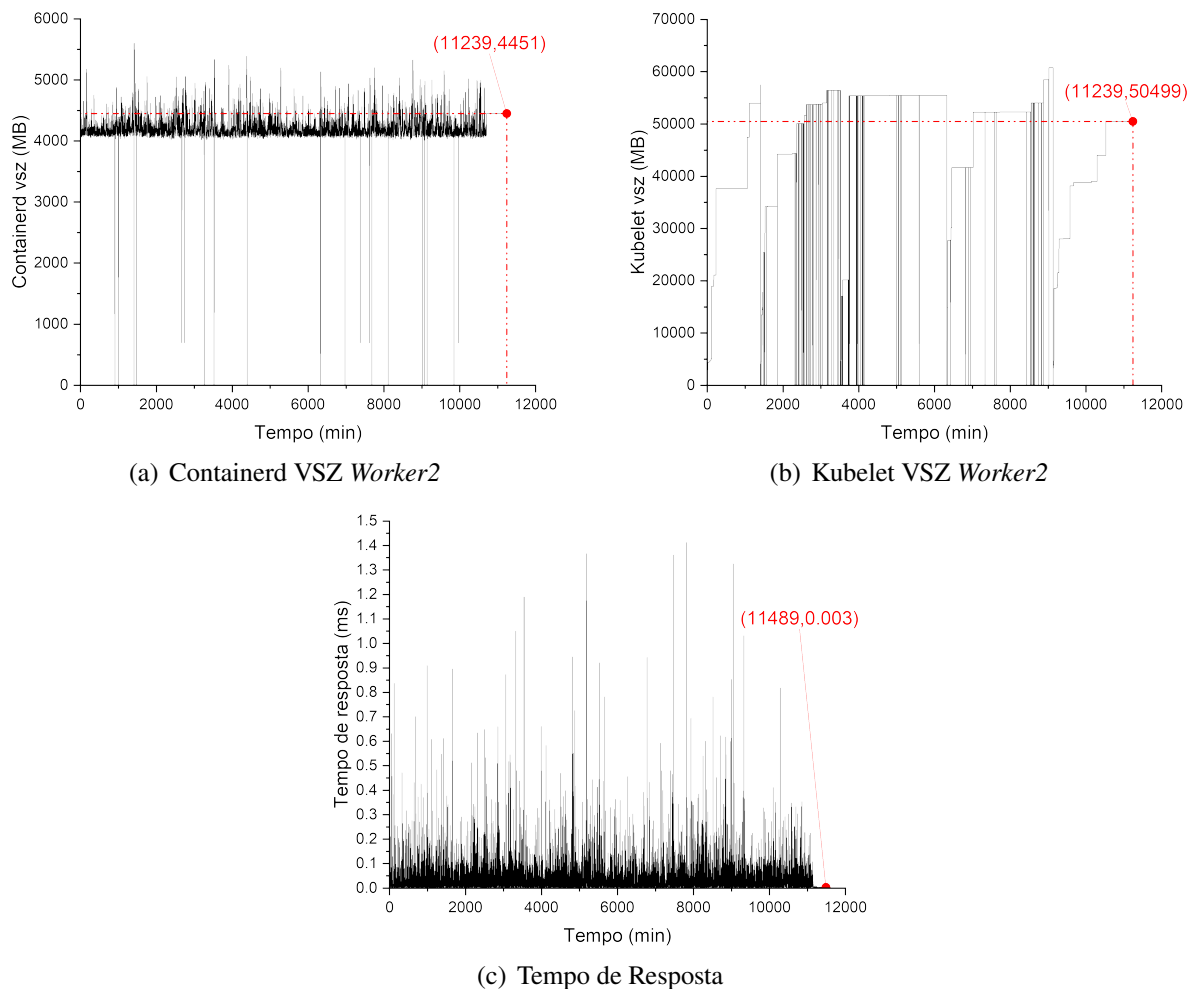


Figura 39 – Cenário 5 - Reiniciando os Nós Workers (Ubuntu 20).

Adotando a mesma abordagem de reinicialização controlada dos *Workers*, agora direcionando a atenção para o comportamento do *Worker2* no Ubuntu 22 (ver Figuras 40, 41, 42). A análise da utilização da CPU, apresentada na Figura 40(a), revelou picos de processamento que ultrapassaram os 80%. No que diz respeito ao uso do disco, conforme representado na Figura

40(b), observou-se um aumento significativo, iniciando acima de 22500 MB no minuto zero e atingindo 27571 MB no minuto 11187, quando o experimento foi concluído sem falhas.

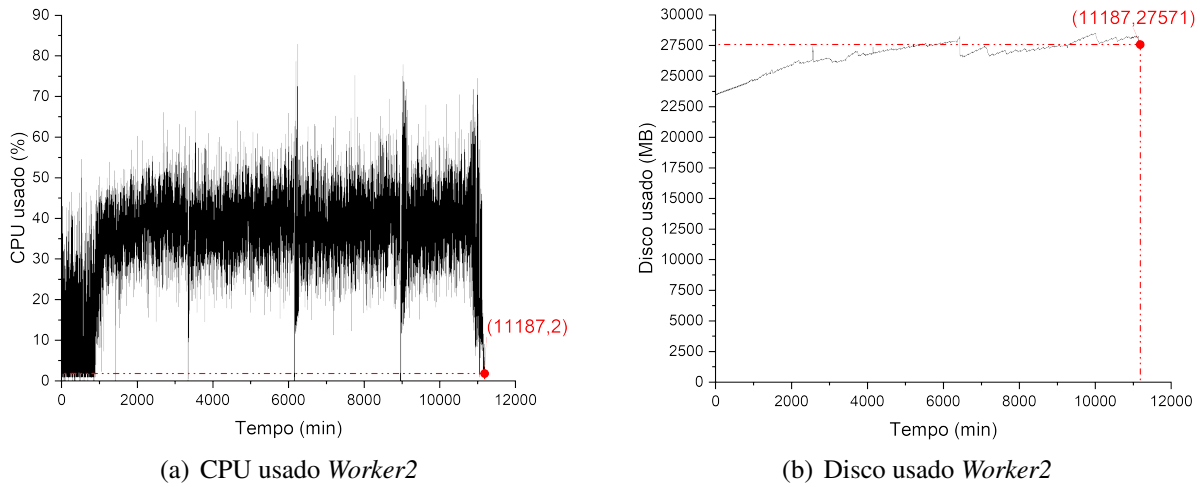


Figura 40 – Cenário 5 - Reiniciando os Nós Workers (Ubuntu 22).

Já a Figura 41(a), exibe uma trajetória ascendente do uso de memória, com quedas notáveis nos minutos 1435, 3368, 6200 e 9034, coincidindo com os momentos de reinicialização do sistema operacional. A coleta de dados de memória foi interrompida no minuto 11187, com um consumo total de 3746 MB de memória usada, pouco depois do *Cluster* finalizar o experimento.

O gráfico de consumo de memória *cached*, apresentado na Figura 41(b), evidencia um aumento no uso de memória *cached* que ultrapassa os 5000 MB, sendo interrompido no momento da reinicialização do Worker2, tendendo a zero. Após esse comportamento, o consumo se manteve mais estável, não ultrapassando os 4000 MB e sempre tendendo a zero nos momentos de reinicialização do Worker2. O Worker2 encerra o monitoramento no minuto 11187, consumindo 2508 MB quando conclui o experimento.

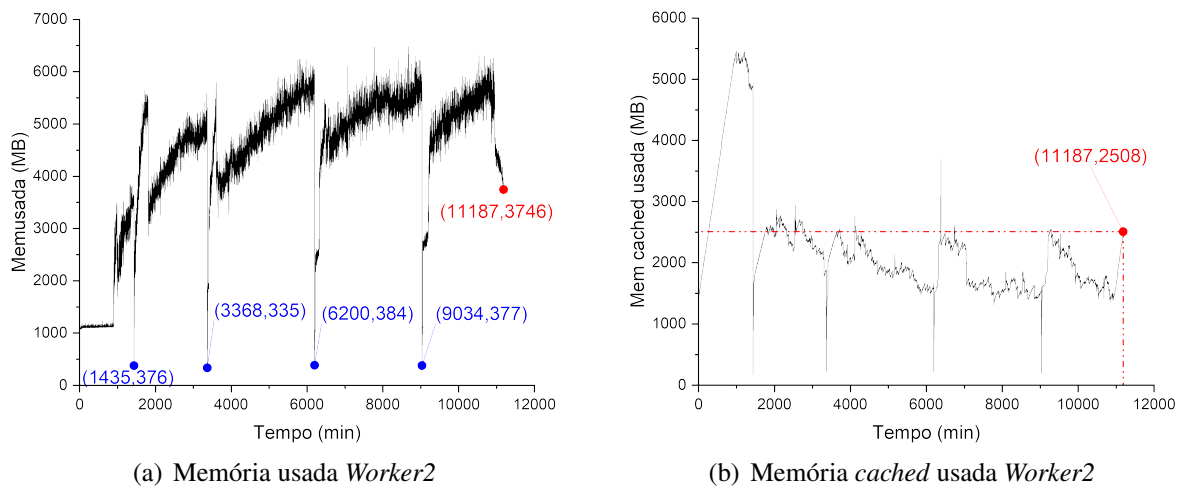


Figura 41 – Cenário 5 - Reiniciando os Nós Workers (Ubuntu 22).

Quanto ao consumo de VSZ no serviço do *Containerd*, iniciou-se com um consumo superior a 4000 MB, continuando a crescer até se aproximar de 10000 MB, conforme pode ser observado na Figura 42(a). Houve variações de redução de consumo quando o *Worker2* foi reinicializado, concluindo o experimento com um consumo de memória VSZ de 9430 MB no minuto 11187. O consumo do serviço *Kubelet* atingiu picos que ultrapassaram os 6000 MB, com variações nos momentos de reinicialização. O serviço consumia 6364 MB no minuto 11187, quando o experimento foi concluído, conforme apresentado na Figura 42(b).

No tocante ao tempo de resposta para o acesso externo ao *Cluster*, a Figura 42(c) apresenta picos que ultrapassaram 0,8 ms e também indica que o pod “portal de notícias” continuou respondendo até o minuto 11489, mesmo após a conclusão do experimento pelo *Cluster* no minuto 11187.

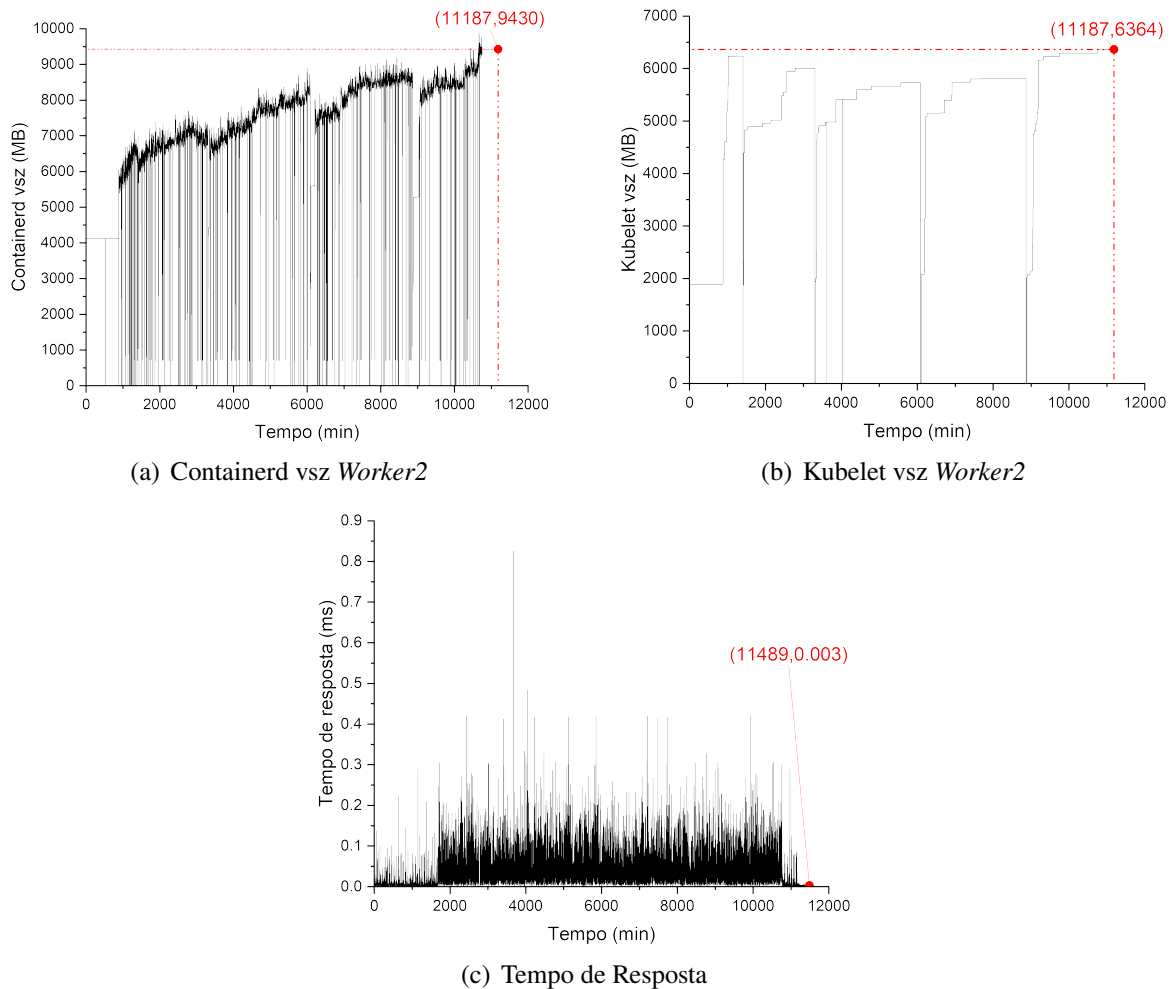


Figura 42 – Cenário 5 - Reiniciando os Nós Workers (Ubuntu 22).

No **Cenário #5**, uma nova abordagem foi implementada, com foco em reinicializações controladas dos nós *Worker* em ambos os *Clusters* Ubuntu 20 e Ubuntu 22. O objetivo era manter uma carga contínua sem interromper as atividades do *Cluster*. No Ubuntu 20, o *Worker2* apresentou picos significativos de CPU e um leve aumento no uso do disco. O uso de memória mostrou flutuações, coincidindo com momentos de reinicializações do sistema operacional. O

serviço *Containerd* manteve um padrão de consumo relativamente estável, enquanto o serviço *Kubelet* consistentemente excedia a capacidade do *Cluster*. Por outro lado, no Ubuntu 22, o *Worker2* apresentou alto uso de CPU e um aumento notável no uso do disco. O consumo de memória exibiu flutuações correspondentes às reinicializações do sistema operacional. O serviço *Containerd* mostrou uma tendência de consumo crescente, enquanto o serviço *Kubelet* atingiu picos dentro da capacidade do *Cluster* durante as reinicializações. Além disso, na análise do tempo de resposta ao pod "portal de notícias", este cenário foi o único a permanecer responsivo durante todo o experimento, e o acesso externo aos *Clusters* permaneceu disponível mesmo após a conclusão dos *scripts* de carga no *Cluster*.

A Tabela 5 apresenta de maneira consolidada os tempos de operação para cada um dos 5 cenários propostos.

Tabela 5 – Tabela Comparativa de tempo de operação entre Cenários.

Versão Ubuntu	Recursos Monitorados	Cenário#1 <i>Worker3</i>	Cenário#2 <i>Worker2</i>	Cenário#3 <i>Worker1</i>	Cenário#4 <i>Worker3</i>	Cenário#5 <i>Worker2</i>
20	CPU	19979	7839	6725	6413	11239
	Disco	19979	7839	6724	6413	11239
	Memória	19979	7839	6724	6412	11239
	Mem Cached	19979	7839	6615	6412	11239
	Containerd	19978	7811	6658	6354	11239
	Kubelet	19978	7566	6651	6354	11239
	Tempo de Resposta	X	4249	2964	2886	11489
22	CPU	8149	7173	7727	8141	11187
	Disco	8149	7174	7827	8016	11187
	Memória	8149	7174	7727	8141	11187
	Mem Cached	8149	7174	7727	8141	11187
	Containerd	8149	6110	7668	8066	11187
	Kubelet	7992	7123	7667	8073	11187
	Tempo de Resposta	X	4249	2993	2955	11489

6

Considerações Finais

Este trabalho analisou o envelhecimento de software em dois *Clusters Kubernetes* hospedados em diferentes versões do Ubuntu (20.04.6 LTS e 22.04.2 LTS) e propôs estratégias de rejuvenescimento de software para ambos os *Clusters*. Na sua totalidade, foram realizados experimentos em cinco cenários, e, em todos eles, tanto no de envelhecimento de software, (**Cenário #1**), quanto nos de rejuvenescimento de software, (**Cenários #2 a #5**), foram monitoradas métricas como: uso de CPU, uso de disco, consumo de memória RAM, uso de memória cached e operação dos serviços do *containerd* e *kubelet*. Adicionalmente aos cenários em que as estratégias de rejuvenescimento foram abordadas, foi implementado um pod “portal de notícias” em um servidor web *nginx* para cada um dos *Clusters*, e requisições, originadas de fora do domínio desses *Clusters*, foram realizadas para verificar a disponibilidade dos acessos. Através dos monitoramentos realizados durante os experimentos, foram obtidos os dados que serviram para plotagem gráfica, análise e discussão.

Nos experimentos do **Cenário #1**, em ambos os sistemas operacionais (Ubuntu 20 e Ubuntu 22), foram identificados vazamentos de recursos indicando o envelhecimento de software. O Ubuntu 20 mostrou-se mais eficiente no gerenciamento de recursos, resultando em uma maior disponibilidade do *Cluster* em comparação com o Ubuntu 22 sob carga de estresse. Observaram-se aumentos persistentes no uso de CPU, disco e memória RAM, sem retorno aos valores iniciais após os períodos de descanso, além da variação no comportamento dos serviços *Containerd* e *Kubelet* entre ambos os sistemas operacionais. Assim, pode-se concluir a presença de indícios de envelhecimento de software nos serviços analisados, manifestando-se de maneira mais proeminente no **Cenário #1**. Essa constatação orientou os experimentos subsequentes (**Cenários #2 a #5**) na direção de esforços voltados para o rejuvenescimento de software.

No **Cenário #2**, ao limpar cache e buffers para reverter o envelhecimento do software, não houve melhorias nos consumos de CPU e disco nos sistemas Ubuntu 20 e Ubuntu 22. Apesar da liberação de memória observada nos gráficos de limpeza de caches, os *Clusters* ainda falharam

por falta de memória disponível. O comportamento ascendente dos serviços do *Containerd* não foi afetado pelas limpezas, semelhante aos padrões observados nos gráficos de CPU, disco e *Kubelet*. Ambos os *Clusters* também apresentaram interrupção no acesso ao “portal de notícias”.

Ao reiniciar o serviço do *Containerd*, no **Cenário #3**, não houve melhorias nos consumos de CPU e disco nos sistemas Ubuntu 20 e Ubuntu 22. No Ubuntu 20, houve uma leve variação na memória usada, enquanto a memória virtual do *Containerd* mostrou apenas quedas momentâneas durante a reinicialização. No Ubuntu 22, foram observadas variações perceptíveis nas métricas de memória. No entanto, essas ações não trataram o envelhecimento de software nos sistemas, levando ao encerramento dos *Clusters* por volta dos minutos 6700 no Ubuntu 20 e 7700 no Ubuntu 22. Ambos os *Clusters* também experimentaram interrupções no acesso ao “portal de notícias”, aproximadamente nos minutos 2960 (Ubuntu 20) e 2990 (Ubuntu 22).

No experimento de reinicialização do serviço do *Kubelet* durante o **Cenário #4**, não foram observadas melhorias substanciais nos consumos de CPU e disco nos sistemas Ubuntu 20 e Ubuntu 22. No Ubuntu 20, observou-se uma pequena variação na memória usada, enquanto a memória virtual do *Kubelet* mostrou apenas quedas momentâneas durante a reinicialização, retornando a um consumo aproximado de 6000 MB. No Ubuntu 22, foram observadas pequenas variações nas métricas de memória. Contudo, essa ação não resultou no rejuvenescimento dos softwares, fazendo com que os *Clusters* parassem por volta dos minutos 6400 no Ubuntu 20 e 8100 no Ubuntu 22. Ambos os *Clusters* também enfrentaram interrupções no acesso ao “portal de notícias”, por volta dos minutos 2900 (Ubuntu 20) e 2950 (Ubuntu 22).

Do conjunto de cenários propostos, apenas o **Cenário #5**, manteve os *Clusters* ativos até o fim dos experimentos. Essa estratégia de rejuvenescimento de software foi a única eficaz, empregando reinicializações controladas nos *Workers* a cada 5 minutos e programando reinicializações em intervalos específicos. No Ubuntu 20, o *Worker2* exibiu picos de CPU, aumento no consumo de disco e memória. No Ubuntu 22, o *Worker2* apresentou comportamento semelhante, mas com picos mais elevados, demonstrando robustez sob carga de estresse prolongada. Ambos os sistemas mantiveram a funcionalidade do pod “portal de notícias” até o final do experimento, indicando que essa estratégia promoveu a estabilidade dos *Clusters* em cenários de estresse prolongado, superando desafios anteriores no processo de rejuvenescimento de software.

Fazendo uma relação entre os dois *Clusters*, na Tabela 5 o Ubuntu 20 teve um melhor desempenho nos **Cenários #1 e #2**, enquanto o Ubuntu 22 se saiu melhor nos **Cenários 3 e 4**. Apenas no **Cenário 5**, em ambos os *Clusters*, foi possível concluir o experimento. A reinicialização dos *Workers* foi a única a ter um efeito efetivo de rejuvenescimento em ambos os sistemas nos 5 cenários propostos.

6.1 Principais contribuições

Este trabalho explorou os efeitos do envelhecimento de software por meio de experimentos controlados em cinco cenários reais, utilizando dois *Clusters Kubernetes*. Do segundo ao quinto cenário, foram implementadas práticas de rejuvenescimento de software. Além disso, avaliou-se a eficiência de um serviço disponibilizado em cada um desses *Clusters* ao responder requisições HTTP de clientes externos.

Esta dissertação desempenhou um papel significativo para a literatura sobre o envelhecimento e rejuvenescimento de software. Tal contribuição é evidenciada pela publicação e aceitação de um artigo científico que apresenta um mapeamento sistemático, ocorrido na *CISTI - 18ª Iberian Conference on Information Systems and Technologies*, em 2023. Além disso, destaca-se a elaboração e submissão de outro artigo, intitulado "Software Aging Issues and Rejuvenation Strategies for a Container Orchestration System", destinado à revista *IEEE Transactions on Cloud Computing (SJR Q1)*, em análise na revista. Esses feitos consolidam a relevância e impacto desta pesquisa no tocante ao envelhecimento e rejuvenescimento de software.

6.2 Trabalhos futuros

Aplicar outras abordagens de rejuvenescimento de software e avaliar métricas adicionais podem ser os próximos desafios para trabalhos futuros. Além disso, como continuidade desta pesquisa, é necessário investigar a memória *Vsz* utilizada, pois foram detectados comportamentos anormais, especialmente nas métricas do serviço do *kubelet*, especificamente no Ubuntu 20, conforme apresentado nas Figuras [15\(b\)](#), [21\(b\)](#), [27\(b\)](#), [33\(a\)](#), [39\(b\)](#).

Outros temas para futuras investigações podem incluir a análise de falhas de programação ou projeto que levam ao aumento no uso de memória pelo *Kubernetes* e o emprego de mecanismos para a criação horizontal e automática de novos nós *Workers* à medida que os nós em produção envelhecem.

Referências

- ALONSO, J. et al. A comparative experimental study of software rejuvenation overhead. *Performance Evaluation*, Elsevier, v. 70, n. 3, p. 231–250, 2013. Citado na página 25.
- ARAUJO, J. et al. Software rejuvenation in eucalyptus cloud computing infrastructure: A method based on time series forecasting and multiple thresholds. In: IEEE. *2011 IEEE Third International Workshop on Software Aging and Rejuvenation*. [S.l.], 2011. p. 38–43. Citado 3 vezes nas páginas 16, 25 e 26.
- ARAUJO, J. C. T. d. *Planejamento de infraestruturas de mobile cloud computing baseado em modelos estocásticos*. [S.l.]: Universidade Federal de Pernambuco, 2017. Citado 2 vezes nas páginas 25 e 27.
- ARAUJO, L. C. *Configuração: uma perspectiva de Arquitetura da Informação da Escola de Brasília*. Dissertação (Mestrado) — Universidade de Brasília, Brasília, mar. 2012. Citado 2 vezes nas páginas 16 e 25.
- AULIYA, Y.; NURDINSYAH, Y.; WULANDARI, D. Performance comparison of docker and lxd with apachebench. In: IOP PUBLISHING. *Journal of Physics: Conference Series*. [S.l.], 2019. v. 1211, n. 1, p. 012042. Citado na página 16.
- AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, v. 1, n. 1, p. 11–33, 2004. Citado na página 24.
- BABCZYŃSKI, T. et al. Virtualization from university point of view. In: _____. *Innovative Technologies in Management and Science*. Cham: Springer International Publishing, 2015. p. 153–169. ISBN 978-3-319-12652-4. Disponível em: <https://doi.org/10.1007/978-3-319-12652-4_9>. Citado na página 15.
- BLINOWSKI, G.; OJDOWSKA, A.; PRZYBYŁEK, A. Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, v. 10, p. 20357–20374, 2022. Citado na página 17.
- CABODI, G.; GIORGIO, D. A.; MINÌ, G. *OS-level virtualization with Linux containers: process isolation mechanisms and performance analysis of last generation container runtimes*. Tese (Doutorado) — Doctoral dissertation, Politecnico di Torino, 2020. Citado 2 vezes nas páginas 16 e 21.
- CARNEVALI, L. et al. Compositional analysis of hierarchical uml statecharts. *IEEE Transactions on Software Engineering*, IEEE, v. 48, n. 12, p. 4762–4788, 2021. Citado na página 25.
- CHEN, P. M.; NOBLE, B. D. When virtual is better than real [operating system relocation to virtual machines]. In: IEEE. *Proceedings eighth workshop on hot topics in operating systems*. [S.l.], 2001. p. 133–138. Citado na página 15.
- COSTA, J. et al. Systematic mapping of literature on software aging and rejuvenation research trends. In: *Reliability and Maintainability Symposium (RAMS)*. [S.l.: s.n.], 2021. p. 1–6. Citado 3 vezes nas páginas 16, 17 e 25.

COSTA, J. et al. Software aging effects on kubernetes in container orchestration systems for digital twin cloud infrastructures of urban air mobility. *Drones*, v. 7, n. 1, 2023. ISSN 2504-446X. Citado 7 vezes nas páginas 37, 40, 41, 42, 43, 44 e 45.

COTRONEO, D. et al. Software aging analysis of the linux operating system. In: *2010 IEEE 21st International Symposium on Software Reliability Engineering*. [S.l.: s.n.], 2010. p. 71–80. Citado 2 vezes nas páginas 16 e 25.

Docker Inc. *container*. 2023. Available at: <<https://www.docker.com/resources/what-container/>>. Accessed on: January 13, 2023. Citado na página 21.

EASTERBROOK, S. et al. Selecting Empirical Methods for Software Engineering Research. In: _____. *Guide to Advanced Empirical Software Engineering*. London: Springer London, 2008. p. 285–311. ISBN 978-1-84800-044-5. Citado na página 45.

FLAUZAC, O.; MAUHOURLAT, F.; NOLOT, F. A review of native container security for running applications. *Procedia Computer Science*, Elsevier, v. 175, p. 157–164, 2020. Citado 2 vezes nas páginas 16 e 21.

FLORA, J. et al. My services got old! can kubernetes handle the aging of microservices? In: *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. [S.l.: s.n.], 2021. p. 40–47. Citado 9 vezes nas páginas 18, 22, 37, 39, 41, 42, 43, 44 e 45.

GARG, S. et al. Analysis of software rejuvenation using markov regenerative stochastic petri net. In: IEEE. *Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95*. [S.l.], 1995. p. 180–187. Citado na página 25.

Gartner. *Microservice*. 2022. Available at: <<https://www.gartner.com/en/information-technology/glossary/microservice>>. Accessed on: January 13, 2024. Citado na página 24.

GROTTKE, M.; MATIAS, R.; TRIVEDI, K. S. The fundamentals of software aging. In: IEEE. *2008 IEEE International conference on software reliability engineering workshops (ISSRE Wksp)*. [S.l.], 2008. p. 1–6. Citado na página 25.

GUPTA, B.; MISHRA, N. Optimized deep learning-based attack detection framework for secure virtualized infrastructures in cloud. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, Wiley Online Library, v. 35, n. 1, p. e2945, 2022. Citado na página 15.

IBM. *What are containers?* 2022. Available at: <<https://www.ibm.com/topics/containers>>. Accessed on: January 13, 2024. Citado na página 21.

JAMES, K.; RANDALL, N.; HADDAWAY, N. A methodology for systematic mapping in environmental sciences. *Environmental Evidence*, SEI: Stockholm Environment Institute, v. 5, 2016. Citado na página 33.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele University Technical Report TR/SE-0401*, v. 33, 08 2004. ISSN 1353-7776. Citado 2 vezes nas páginas 29 e 45.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004. Citado na página 32.

Lawrence E Hecht. What the data says about kubernetes deployment patterns. 2018. Citado na página 23.

LEVITIN, G.; XING, L.; BEN-HAIM, H. Optimizing software rejuvenation policy for real time tasks. *Reliability Engineering & System Safety*, Elsevier, v. 176, p. 202–208, 2018. Citado na página 25.

MARTIN, E. *Virtualization and containerization: a new concept for data center management to optimize resources distribution*. [S.l.]: Università Ca' Foscari Venezia, 2022. Citado 4 vezes nas páginas 15, 16, 20 e 21.

MATIAS, R.; FILHO, P. An experimental study on software aging and rejuvenation in web servers. In: IEEE. *30th Int. Computer Software and Applications Conference (COMPSAC'06)*. [S.l.], 2006. v. 1, p. 189–196. Citado 3 vezes nas páginas 37, 43 e 44.

Mia-Platform Team. *Top 4 cloud native trends in 2022 shaping the future of business*. 2018. Available at: <<https://www.cncf.io/blog/2022/01/05/top-4-cloud-native-trends-in-2022-shaping-the-future-of-business/>>. Accessed on: January 13, 2024. Citado na página 16.

MONDAL, S. K.; SABYASACHI, A. S.; MUPPALA, J. K. On dependability, cost and security trade-off in cloud data centers. In: *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. [S.l.: s.n.], 2017. p. 11–19. Citado na página 20.

MORAVCIK, M. et al. Kubernetes-evolution of virtualization. In: IEEE. *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. [S.l.], 2022. p. 454–459. Citado na página 22.

MYINT, M. T. H.; THEIN, T. Availability improvement in virtualized multiple servers with software rejuvenation and virtualization. In: *4th IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI 2010)*. [S.l.: s.n.], 2010. p. 156 – 162. Citado na página 17.

NAYDENOV, N.; RUSEVA, S. Cloud container orchestration architectures, models and methods: a systematic mapping study. In: *2023 22nd International Symposium INFOTEH-JAHORINA (INFOTEH)*. [S.l.: s.n.], 2023. p. 1–8. Citado na página 16.

OLIVEIRA, F. et al. Experimental evaluation of software aging effects in a container-based virtualization platform. In: IEEE. *Int. Conf. on Systems, Man, and Cybernetics (SMC)*. [S.l.], 2020. p. 414–419. Citado 6 vezes nas páginas 37, 38, 41, 43, 44 e 45.

OLIVEIRA, F. et al. Software aging in container-based virtualization: An experimental analysis on docker platform. In: *16th Iberian Conf. on Information Systems and Technologies (CISTI)*. [S.l.: s.n.], 2021. p. 1–7. Citado 9 vezes nas páginas 16, 20, 37, 39, 41, 42, 43, 44 e 45.

PAHL, C. Containerization and the paas cloud. *IEEE Cloud Computing*, IEEE, v. 2, n. 3, p. 24–31, 2015. Citado na página 22.

PAPAKOSTAS, N.; RAMASUBRAMANIAN, A. K. Digital technologies as a solution to complexity caused by mass personalization. In: *Design and Operation of Production Networks for Mass Personalization in the Era of Cloud Technology*. [S.l.]: Elsevier, 2022. p. 153–180. Citado na página 20.

PENG, J. et al. Comparison of several cloud computing platforms. In: IEEE. *2009 Second international symposium on information science and engineering*. [S.l.], 2009. p. 23–27. Citado na página 15.

PEREIRA, M. M.; BACK, G.; JÚNIOR, N. W. *Arquitetura baseada em microserviço*. 2022. Acessado em 13 de Janeiro de 2024 Obtido de <https://www.researchgate.net/profile/Murillo-De-Miranda-Pereira> Citado na página 17.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and software technology*, Elsevier, v. 64, p. 1–18, 2015. Citado na página 32.

PIETRANTUONO, R. et al. An empirical study on software aging of long-running object detection algorithms. In: *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*. [S.l.: s.n.], 2022. p. 1091–1102. Citado na página 16.

RANDAL, A. The ideal versus the real: Revisiting the history of virtual machines and containers. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 53, n. 1, p. 1–31, 2020. Citado na página 21.

Risingstack. What is orchestration? redhat 2019. 2022. Citado na página 23.

SANTOS, M. et al. Software aging and rejuvenation in microservices: A systematic literature mapping. In: IEEE. *2023 18th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.], 2023. p. 1–8. Citado 2 vezes nas páginas 17 e 18.

SCHROEDER, B.; GIBSON, G. A. A large-scale study of failures in high-performance computing systems. *IEEE transactions on Dependable and Secure Computing*, Ieee, v. 7, n. 4, p. 337–350, 2009. Citado na página 25.

SURBIRYALA, J.; RONG, C. Cloud computing: History and overview. In: IEEE. *2019 IEEE Cloud Summit*. [S.l.], 2019. p. 1–7. Citado na página 21.

TANG, M. et al. A task execution framework based on aging indicator and sarimi. In: IEEE. *2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. [S.l.], 2020. p. 18–25. Citado na página 25.

The Kubernetes. Kubernetes documentation/concepts/overview. 2023. Citado na página 23.

TORQUATO, M.; ARAUJO, J.; MACIEL, P. Estudo experimental de envelhecimento de software em nuvens kvm/opennebula: Live migration como mecanismo de suporte ao rejuvenescimento de software. In: *XIII Workshop em Clouds e Aplicacoes in conjunction with 33rd Brazilian Symposium on Computer Networks and Distributed Systems (SBRC2015)*. Vitoria, ES, Brazil: Universidade Federal do Espirito Santo (UFES). [S.l.: s.n.], 2015. p. 1–14. Citado na página 25.

TORQUATO, M.; VIEIRA, M. An experimental study of software aging and rejuvenation in dockerd. In: *2019 15th European Dependable Computing Conference (EDCC)*. [S.l.: s.n.], 2019. p. 1–6. Citado 4 vezes nas páginas 38, 41, 43 e 45.

TRIVEDI, K.; VAIDYANATHAN, K.; GOSEVA-POPSTOJANOVA, K. Modeling and analysis of software aging and rejuvenation. In: *Proceedings 33rd Annual Simulation Symposium (SS 2000)*. [S.l.: s.n.], 2000. p. 270–279. Citado na página 25.

VILLAMIZAR, M. et al. Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures. *Service Oriented Computing and Applications*, Springer, v. 11, p. 233–247, 2017. Citado na página 17.

VINÍCIUS, L. et al. Docker platform aging: a systematic performance evaluation and prediction of resource consumption. *The Journal of Supercomputing*, Springer, p. 1–31, 2022. Citado 7 vezes nas páginas 16, 37, 40, 41, 43, 44 e 45.

WOHLIN, C. et al. On the reliability of mapping studies in software engineering. *Journal of Systems and Software*, Elsevier, v. 86, n. 10, p. 2594–2610, 2013. Citado na página 32.

YUE, J.; WU, X.; XUE, Y. Microservice aging and rejuvenation. In: IEEE. *2020 World Conference on Computing and Communication Technologies (WCCCT)*. [S.l.], 2020. p. 1–5. Citado 7 vezes nas páginas 37, 38, 41, 42, 43, 44 e 45.

ZHENG, J.; OKAMURA, H.; DOHI, T. Optimal rejuvenation policies for non-markovian availability models with aperiodic checkpointing. *IEICE Transactions on Information and Systems*, The Institute of Electronics, Information and Communication Engineers, v. 103, n. 10, p. 2133–2142, 2020. Citado na página 25.

Apêndices

APÊNDICE A – *Scripts de Monitoramento*

Neste apêndice, estão contidos todos os *scripts* de monitoramento desenvolvidos e utilizados para a monitoramento dos experimentos deste trabalho, são eles: monitoramento de CPU (Código 1), monitoramento de disco (Código 2), monitoramento de memória (Código 3), monitoramento do serviço do *Containerd* (Código 4), e, por fim, monitoramento do serviço do *Kubelet* (Código 5).

Código 1 – *Script* de Monitoramento de CPU.

```
1 #!/bin/bash
2 LOG_FILE="/logs/monitor_cpu.log"
3 function timestamp() {date +"%Y-%m-%d %H:%M:%S"}
4 while true
5 do
6     cpu_usage=$(top -b -n 1 | grep "Cpu(s)" | awk '{print $2}')
7     cpu=$(mpstat | grep all)
8     usr=$(echo $cpu | awk '{print $3}')
9     nice=$(echo $cpu | awk '{print $4}')
10    sys=$(echo $cpu | awk '{print $5}')
11    iowait=$(echo $cpu | awk '{print $6}')
12    soft=$(echo $cpu | awk '{print $8}')
13    echo "$(timestamp) CPU: $cpu_usage $usr $nice $sys $iowait $soft" >> $LOG_FILE
14    sleep 60
15 done
```

Código 2 – Script de Monitoramento de Disco.

```
1
2 #!/bin/bash
3 LOG_FILE="/logs/monitor_disco.log"
4 function timestamp() {date +"%Y-%m-%d %H:%M:%S"}
5 while true
6 do
7     disk=$(df | grep /dev/sda3)
8     disk_usage=$(echo $disk | awk '{print $3}')
9     disk_usage2=$(echo $disk | awk '{print $5}')
10    echo "$(timestamp) Disk: $disk_usage $disk_usage2" >> $LOG_FILE
11    sleep 60
12 done
13
14
```

Código 3 – Script de Monitoramento de Memória.

```
1
2 #!/bin/bash
3 LOG_FILE="/logs/monitor_memoria.log"
4 function timestamp() {date +"%Y-%m-%d %H:%M:%S"}
5 while true
6 do
7     mem=$(free | grep Mem)
8     mem_total=$(echo $mem | awk '{printf $2}')
9     mem_usage=$(echo $mem | awk '{printf $3}')
10    mem_free=$(echo $mem | awk '{printf $4}')
11    mem_shared=$(echo $mem | awk '{printf $5}')
12    mem_cache=$(echo $mem | awk '{printf $6}')
13    mem_available=$(echo $mem | awk '{printf $7}')
14    cached=$(cat /proc/meminfo | grep -i Cached | sed -n '1p' | awk '{print $2}')
15    buffer=$(cat /proc/meminfo | grep -i Buffers | awk '{print $2}')
16    swap=$(cat /proc/meminfo | grep -i Swap | grep -i Free | awk '{print $2}')
17    echo "$(timestamp) Memória: $mem_total $mem_usage $mem_free $mem_shared
18    ↵ $mem_cache $mem_available $cached $buffer $swap" >> $LOG_FILE
19    sleep 60
20 done
```

Código 4 – Script de Monitoramento de *Containerd*.

```

1  #!/bin/bash
2  LOG_FILE="/logs/monitor_containerd.log"
3  function timestamp() {date +"%d-%m-%Y %H:%M:%S"}
4  while true
5  do
6      pidHead=$(pidof -s containerd)
7      dados=$(pidstat -u -h -p $pidHead -T ALL -r 1 1 | sed -n '4p')
8      thread=$(cat /proc/"$pidHead"/status | grep Threads | awk '{print $2}')
9      system=$(echo "$dados" | awk '{print $5}')
10     cpu=$(echo "$dados" | awk '{print $8}')
11     mem=$(echo "$dados" | awk '{print $14}')
12     vmrss=$(echo "$dados" | awk '{print $13}')
13     vsz=$(echo "$dados" | awk '{print $12}')
14     echo "$(timestamp); $system; $cpu; $mem; $vmrss; $vsz; $thread" >>
15         ↪ $LOG_FILE
16     sleep 60
17 done
18

```

Código 5 – Script de Monitoramento de *Kubelet*.

```

1  #!/bin/bash
2  LOG_FILE="/logs/monitor_kubelet.log"
3  function timestamp() {date +"%d-%m-%Y %H:%M:%S"}
4  while true
5  do
6      pidHead=$(pidof -s kubelet)
7      dados=$(pidstat -u -h -p $pidHead -T ALL -r 1 1 | sed -n '4p')
8      thread=$(cat /proc/"$pidHead"/status | grep Threads | awk '{print $2}')
9      system=$(echo "$dados" | awk '{print $5}')
10     cpu=$(echo "$dados" | awk '{print $8}')
11     mem=$(echo "$dados" | awk '{print $14}')
12     vmrss=$(echo "$dados" | awk '{print $13}')
13     vsz=$(echo "$dados" | awk '{print $12}')
14     echo "$(timestamp); $system; $cpu; $mem; $vmrss; $vsz; $thread" >>
15         ↪ $LOG_FILE
16     sleep 60
17 done

```

APÊNDICE B – *Scripts* de Execução

Neste apêndice, estão contidos todos os *scripts* de execução desenvolvidos e utilizados para a monitoramento dos experimentos deste trabalho. São eles: *script* de criação e remoção dos pods (Código 6 e Código 7), *script* de execução de limpeza de *caches* e *buffers* (Código 8), *script* de execução para reinicialização do *containerd* (Código 9), *script* de execução para reinicialização do *kubelet* (Código 11) e por fim, o *script* de execução para reinicialização dos *Workers* (Código 12).

Código 6 – *Script* de Criação e Remoção dos pods.

```

1      #!/bin/bash
2      function timestamp() {date +"%Y-%m-%d %H:%M:%S"}
3      echo $(timestamp) "Iniciando aproximadamente 10 dia de carga com reboot dos nodes a
   ↪   cada 2 dias"
4      echo $(timestamp) "status dos nodes"
5      kubectrl get nodes -o wide
6      contador1=0
7      while [ $contador1 -lt 1140 ]; do
8          echo $(timestamp) "Contador 1: $contador1"
9          echo $(timestamp) "status dos nodes"
10         kubectrl get nodes -o wide
11         echo $(timestamp) "Iniciando 150 Pods A..."
12         kubectrl apply -f 150pods_a.yaml
13         echo $(timestamp) "Esperando 120 segundos"
14         sleep 120s
15         echo $(timestamp) "Pods em funcionamento"
16         kubectrl get pods --all-namespaces | grep -c Running
17         echo $(timestamp) "status dos nodes"
18         kubectrl get nodes -o wide
19         echo $(timestamp) "Iniciando 150 Pods B..."
20         kubectrl apply -f 150pods_b.yaml
21         echo $(timestamp) "Esperando 120 segundos"
22         sleep 120s
23         echo $(timestamp) "Pods em funcionamento"
24         kubectrl get pods --all-namespaces | grep -c Running
25         echo $(timestamp) "status dos nodes"
26         kubectrl get nodes -o wide
27         echo $(timestamp) "Iniciando 150 Pods C..."
28         kubectrl apply -f 150pods_c.yaml
29         echo $(timestamp) "Esperando 120 segundos"

```

Código 7 – Continuação do *Script* de Criação e Remoção dos pods.

```
1 sleep 120s
2   echo $(timestamp) "Pods em funcionamento*"
3   kubectl get pods --all-namespaces | grep -c Running
4   echo $(timestamp) "Pods em funcionamento"
5   kubectl get pods --all-namespaces | grep -c Running
6   echo $(timestamp) "status dos nodes"
7   kubectl get nodes -o wide
8   echo $(timestamp) "Deletando 150 Pods A..."
9   kubectl delete -f 150pods_a.yaml
10  echo $(timestamp) "Esperando 60 segundos"
11 sleep 60s
12  echo $(timestamp) "Pods em funcionamento"
13  kubectl get pods --all-namespaces | grep -c Running
14  echo $(timestamp) "status dos nodes"
15  kubectl get nodes -o wide
16  echo $(timestamp) "Deletando 150 Pods B..."
17  kubectl delete -f 150pods_b.yaml
18  echo $(timestamp) "Esperando 60 segundos"
19 sleep 60s
20  echo $(timestamp) "Pods em funcionamento"
21  kubectl get pods --all-namespaces | grep -c Running
22  echo $(timestamp) "status dos nodes"
23  kubectl get nodes -o wide
24  echo $(timestamp) "Deletando 150 Pods C..."
25  kubectl delete -f 150pods_c.yaml
26  echo $(timestamp) "Esperando 60 segundos"
27 sleep 90s
28  echo $(timestamp) "Pods em funcionamento"
29  kubectl get pods --all-namespaces | grep -c Running
30  echo $(timestamp) "status dos nodes"
31  kubectl get nodes -o wide
32  contador=$((contador+1))
33  restart_loop=$(echo $(timestamp) "Reiniciando loop de criação e remoção dos pods")
34 done
35
36 echo $(timestamp) "concluído aproximadamente 10 dia de carga"
37 echo $(timestamp) "status dos nodes"
38 kubectl get nodes -o wide
39 echo $(timestamp) "FIM"
40
```

Código 8 – Script de Execução de Limpeza de Caches e Buffers.

```
1  #!/bin/bash
2  function timestamp() {date +"%Y-%m-%d %H:%M:%S"}
3  echo $(timestamp) "Status memoria"
4  free -h
5  echo $(timestamp) "Esperando 2 dias"
6  sleep 172800s
7  echo $(timestamp) "Status memoria"
8  free -h
9  echo $(timestamp) "limpando cache"
10 sudo sync && echo 3 | sudo tee /proc/sys/vm/drop_caches
11 echo $(timestamp) "cache limpo"
12 echo $(timestamp) "Status memoria"
13 free -h
14 echo $(timestamp) "Esperando +2 dias"
15 echo $(timestamp) "Status memoria"
16 free -h
17 sleep 172800s
18 echo $(timestamp) "Status memoria"
19 free -h
20 echo $(timestamp) "limpando cache"
21 sudo sync && echo 3 | sudo tee /proc/sys/vm/drop_caches
22 echo $(timestamp) "cache limpo"
23 echo $(timestamp) "Status memoria"
24 free -h
25 echo $(timestamp) "Esperando +2 dias"
26 sleep 172800s
27 echo $(timestamp) "Status memoria"
28 free -h
29 echo $(timestamp) "limpando cache"
30 sudo sync && echo 3 | sudo tee /proc/sys/vm/drop_caches
31 echo $(timestamp) "cache limpo"
32 echo $(timestamp) "Status memoria"
33 free -h
34 echo $(timestamp) "Esperando +2 dias"
35 sleep 172800s
36 echo $(timestamp) "Status memoria"
37 free -h
38 echo $(timestamp) "limpando cache"
39 sudo sync && echo 3 | sudo tee /proc/sys/vm/drop_caches
40 echo $(timestamp) "cache limpo"
41 echo $(timestamp) "FIM"
```

Código 9 – Script de Execução para Reinicialização do *Containerd*.

```
1  #!/bin/bash
2  function timestamp() {date +"%Y-%m-%d %H:%M:%S"}
3  echo $(timestamp) "Status containerd.service WOR1"
4  sudo systemctl status containerd.service >> /logs/restart_containerd.csv
5  echo $(timestamp) "Esperando 2 dias"
6
7  sleep 172800s
8  echo $(timestamp) STATUS
9  sudo systemctl status containerd.service >> /logs/restart_containerd.csv
10 sudo systemctl restart containerd.service >> /logs/restart_containerd.csv
11 sudo systemctl status containerd.service >> /logs/restart_containerd.csv
12 echo $(timestamp) "Esperando 2 dias"
13
14 sleep 172800s
15 echo $(timestamp) STATUS
16 sudo systemctl status containerd.service >> /logs/restart_containerd.csv
17 echo $(timestamp) RESTART
18 sudo systemctl restart containerd.service >> /logs/restart_containerd.csv
19 echo $(timestamp) STATUS
20 sudo systemctl status containerd.service >> /logs/restart_containerd.csv
21 echo $(timestamp) "Esperando 2 dias"
22
23 sleep 172800s
24 echo $(timestamp) STATUS
25 sudo systemctl status containerd.service >> /logs/restart_containerd.csv
26 echo $(timestamp) RESTART
27 sudo systemctl restart containerd.service >> /logs/restart_containerd.csv
28 echo $(timestamp) STATUS
29 sudo systemctl status containerd.service >> /logs/restart_containerd.csv
30 echo $(timestamp) "Esperando 2 dias"
31
32 sleep 172800s
33 echo $(timestamp) STATUS
34 sudo systemctl status containerd.service >> /logs/restart_containerd.csv
35 echo $(timestamp) RESTART
36 sudo systemctl restart containerd.service >> /logs/restart_containerd.csv
37 echo $(timestamp) STATUS
38 sudo systemctl status containerd.service >> /logs/restart_containerd.csv
39 echo $(timestamp) "Esperando 2 dias"
```

Código 10 – Continuação do *Script* de Execução para Reinicialização do *Containerd*.

```
1 sleep 172800s
2 echo $(timestamp) STATUS
3 sudo systemctl status containerd.service >> /logs/restart_containerd.csv
4 echo $(timestamp) RESTART
5 sudo systemctl restart containerd.service >> /logs/restart_containerd.csv
6 echo $(timestamp) STATUS
7 sudo systemctl status containerd.service >> /logs/restart_containerd.csv
8
9 echo $(timestamp) "FIM"
```

Código 11 – *Script* de Execução para Reinicialização do *Kubelet*.

```
1 #!/bin/bash
2 function timestamp() {date +"%Y-%m-%d %H:%M:%S"}
3 echo $(timestamp) "Status kubelet WOR1" >> /logs/restart_kubelet.csv
4 sudo systemctl status kubelet >> /logs/restart_kubelet.csv
5 echo $(timestamp) "Esperando 2 dias" >> /logs/restart_kubelet.csv
6 sleep 172800s
7 echo $(timestamp) STATUS kubelet >> /logs/restart_kubelet.csv
8 sudo systemctl status kubelet >> /logs/restart_kubelet.csv
9 sudo systemctl restart kubelet >> /logs/restart_kubelet.csv
10 sudo systemctl status kubelet >> /logs/restart_kubelet.csv
11 echo $(timestamp) "Esperando 2 dias" >> /logs/restart_kubelet.csv
12 sleep 172800s
13 echo $(timestamp) STATUS kubelet >> /logs/restart_kubelet.csv
14 sudo systemctl status kubelet >> /logs/restart_kubelet.csv
15 sudo systemctl restart kubelet >> /logs/restart_kubelet.csv
16 sudo systemctl status kubelet >> /logs/restart_kubelet.csv
17 echo $(timestamp) "Esperando 2 dias" >> /logs/restart_kubelet.csv
18 sleep 172800s
19 echo $(timestamp) STATUS kubelet >> /logs/restart_kubelet.csv
20 sudo systemctl status kubelet >> /logs/restart_kubelet.csv
21 sudo systemctl restart kubelet >> /logs/restart_kubelet.csv
22 sudo systemctl status kubelet >> /logs/restart_kubelet.csv
23 echo $(timestamp) "Esperando 2 dias" >> /logs/restart_kubelet.csv
24 sleep 172800s
25 echo $(timestamp) STATUS kubelet >> /logs/restart_kubelet.csv
26 sudo systemctl status kubelet >> /logs/restart_kubelet.csv
27 sudo systemctl restart kubelet >> /logs/restart_kubelet.csv
28 sudo systemctl status kubelet >> /logs/restart_kubelet.csv
29 echo $(timestamp) "Esperando 2 dias" >> /logs/restart_kubelet.csv
30 echo $(timestamp) "FIM"
```


Código 12 – Script de Execução para Reinicialização dos *Workers*.

```
1  #!/bin/bash
2
3  function timestamp() {date +"%Y-%m-%d %H:%M:%S"}
4  echo $(timestamp) "Status WOR1" >> /logs/restart.csv
5  free -h >> /logs/restart.csv
6  echo $(timestamp) "Esperando 2 dias" >> /logs/restart.csv
7  sleep 10s
8  echo $(timestamp) STATUS UBUNTU >> /logs/restart.csv
9  systemctl status containerd.service >> /logs/restart.csv
10 free -h >> /logs/restart.csv
11 echo $(timestamp) RESTART >> /logs/restart.csv
12 /sbin/reboot
13
```