UNIVERSIDADE FEDERAL DE SERGIPE

PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Avaliação de Desempenho e Problemas de Envelhecimento de *Software* em *Clusters* de Contêineres

Dissertação de Mestrado

Jackson Tavares da Costa

São Cristóvão – Sergipe

2022

UNIVERSIDADE FEDERAL DE SERGIPE

PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Jackson Tavares da Costa

Avaliação de Desempenho e Problemas de Envelhecimento de *Software* em *Clusters* 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): Rubens de Souza Matos Júnior
Coorientador(a): Jean Carlos Teixeira de Araújo

São Cristóvão – Sergipe

2022

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL
UNIVERSIDADE FEDERAL DE SERGIPE

Costa, Jackson Tavares da
C837a Avaliação de desempenho e problemas de envelhecimento de software em clusters de contêineres / Jackson Tavares da Costa ; orientador Rubens de Souza Matos Júnior. - São Cristóvão, 2022.
79 f.; il.

Dissertação (mestrado em Ciência da Computação) – Universidade Federal de Sergipe, 2022.

1. Software. 2. Cluster (Sistema de computador). I. Matos Junior, Rubens de Souza orient. II. Título.

CDU 004.4

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: JACKSON TAVARES DA COSTA

Em 26 dias do mês de agosto do ano de dois mil e vinte dois, com início às 14h, realizou-se no Auditório 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 **Jackson Tavares da Costa**, que desenvolveu o trabalho intitulado: **" Avaliação de Desempenho e Problemas de Envelhecimento de Software em Clusters de Contêineres",** sob a orientação do Prof. Dr. **Rubens de Souza Matos Júnior**. A Sessão foi presidida pelo Prof. Dr. **Rubens de Souza Matos Júnior** (PROCC/UFS), que após a apresentação da dissertação passou a palavra aos outros membros da Banca Examinadora, Profª. Dra. **Rosangela Maria de Melo (Instituto Federal de Pernambuco - Campus Paulista)** e, em seguida, o Prof. Dr. **Admilson Ribamar Lima Ribeiro** (PROCC/UFS) e o Prof. Dr. **Jean Carlos Teixeira De Araujo** (PROCC/UFS). Após as discussões, a Banca Examinadora reuniu-se e considerou o mestrando (a) aprovado "*(aprovado/reprovado)*". 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.

Cidade Universitária "*Prof. José Aloísio de Campos*", 26 de agosto de 2022.

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

Prof. Dr. Jean Carlos Teixeira De Araujo
(PROCC/UFS)
Examinador Interno
Coorientador

Profª. Dra. Rosangela Maria de Melo
(IFPE)
Examinador Externo

Prof. Dr. Admilson de Ribamar Lima Ribeiro
(PROCC/UFS)
Examinador Interno

Jackson Tavares da Costa
Candidato

Dedico este trabalho para toda a minha família, amigos, professores e em especial à memória da melhor mãe que o mundo já viu e de que tive o privilégio de ser filho, Noélia Tavares da Costa!

Agradecimentos

Agradeço primeiramente a Deus, porque sempre tem me guiado em tudo o que faço e não foi diferente no mestrado, sempre me impulsionando a continuar, meio que de forma sobrenatural, quando tudo parecia desmoronar em minha volta.

Agradecer a minha mãe Noélia Tavares da Costa, minha razão de ser o que sou, razão pela qual sempre fiz o meu melhor para que tivesse orgulho de mim, como uma forma de retribuição do seu carinho e grande amor por mim. Agradeço aos meus irmãos Andréa Tavares da Costa, Jordão Tavares da Costa e Hércules Tavares da Costa pelo incondicional apoio desde o início até a conclusão deste trabalho.

Agradeço especialmente ao professor Rubens Matos por ter aceitado ser meu orientador neste trabalho e ter confiado em mim para este desafio e ao professor Jean Araújo por aceitar seu meu coorientador, contribuindo muito na condução.

Agradeço também ao professor Thiago Xavier por sempre estar disponível e a seu espirito colaborativo, sempre apto a ajudar. Agradeço também a todos os professores do Departamento de Computação da Universidade Federal de Sergipe, por terem passado um pouco do conhecimento e sempre estarem de forma direta ou indireta incentivando a superar nossos limites.

Agradecer também a Renata Grasiela, por sempre ter estado ao meu lado desde o início desta jornada, sempre confiando e incentivando na realização desse trabalho.

Ser derrotado muitas vezes
é uma condição temporária.
Desistir é o que o torna permanente.
(Marilyn vos Savant)

Resumo

Com o aumento das implementações em nuvem dos mais variados serviços por diversas companhias nos últimos anos, foram elevadas significativamente as demandas dos provedores por arquiteturas robustas, escaláveis e complexas, necessitando também de requisitos essenciais como confiabilidade, desempenho e disponibilidade. Com isso, tecnologias baseadas em virtualização de contêineres tem ganhado muito destaque. Em contrapartida, com o significativo aumento na utilização de contêineres comportando os serviços, surge também a necessidade de ferramentas de gerenciamento visando controlar as tarefas de automatização de implantação, processamento, escalonamento e operações gerais dos aplicativos. Para esta finalidade, o Kubernetes tem sido a ferramenta mais utilizada devido a sua eficiência na orquestração de contêineres, também no ambiente de tempo de execução, além da sua maturidade e robustez. Muitos trabalhos nos últimos anos vêm destacando a importância do estudo sobre o fenômeno de envelhecimento de *software* em sistemas de computação em nuvem, por se tratar de uma ameaça para a disponibilidade e desempenho dos serviços oferecidos. Diante disso, pretende-se avaliar o desempenho do Kubernetes, para verificar possíveis níveis de degradação ou falhas, visando detectar sinais oriundos do fenômeno de envelhecimento de *software* e propor abordagens de rejuvenescimento de *software*, com o objetivo de evitar repentinas interrupções no fornecimento dos serviços implantados no *Cluster* do Kubernetes.

Palavras-chave: Envelhecimento de Software, Rejuvenescimento de Software, Kubernetes, Orquestração, Contêineres.

Abstract

With the increase in cloud implementations of the most varied services by several companies in recent years, the demands of providers for robust, scalable and complex architectures have significantly increased, also requiring essential requirements such as reliability, performance and availability. As a result, technologies based on container virtualization have gained much prominence. On the other hand, with the significant increase in the use of containers supporting services, there is also a need for management tools to control the tasks of automating deployment, processing, scheduling and general application operations. For this purpose, Kubernetes has been the most used tool due to its efficiency in container orchestration, also in the runtime environment, in addition to its maturity and robustness. Many works in recent years have highlighted the importance of studying the phenomenon of software aging in cloud computing systems, as it is a threat to the availability and performance of the services offered. Therefore, we intend to evaluate the performance of Kubernetes, to verify possible levels of degradation or failures, in order to detect signs arising from the phenomenon of software aging and propose approaches to software rejuvenation, in order to avoid sudden interruptions in the provision of services deployed in the Kubernetes Cluster.

Keywords: Software Aging, Software Rejuvenation, Kubernetes, Orchestration, Containers.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
CNI	Container Network Interface
CPN	Coloured Petri Nets
CPU	Central Process Unit
DCOMP	Departamento de Computação
DNS	Domain Name System
DSPN	Deterministic and Stochastic Petri Nets
GQM	Goal Question Metric
HTTP	Hypertext Transfer Protocol
HPA	Horizontal Pod Autoscaling
ICMP	Internet Control Message Protocol
CLI	Interface de Linha de Comando
IoT	Internet das Coisas
IP	Internet Protocol
MS	Mapeamento Sistemático
MB	Megabytes
RAM	Random Access Memory
RTT	Round Trip Time
PaaS	Platform as a Service
SAR	Software Aging and Rejuvenation
SOs	Sistemas Operacionais
SPN	Stochastic Petri Nets
SRN	Stochastic Reward Nets
TI	Tecnologia da Informação

UFS	Universidade Federal de Sergipe
VMs	Virtual Machines
VMM	Virtual Machine Monitor
Wi-Fi	Wireless Fidelity

Sumário

1 Introdução

A computação comercial tem passado por diversas mudanças no âmbito da computação em nuvem (LIU et al., 2018; PAN; HU, 2014). A quantidade de companhias que têm hospedado e implementado aplicações em nuvem tem aumentado nos últimos anos. Dessa forma, desencadearam-se as demandas dos provedores em nuvem, surgindo a necessidade de projetar arquiteturas robustas e complexas (BRILHANTE et al., 2014). Garantir confiabilidade, desempenho e disponibilidade neste contexto se tornou um dos principais desafios (MONDAL; SABYASACHI; MUPPALA, 2017).

No contexto da computação em nuvem, as técnicas de virtualização fornecem escalabilidade de recursos que são compartilhados em tal ambiente. É dito por (PAHL, 2015) que as VMs (*Virtual Machines* – Máquinas Virtuais) constituíram-se na virtualização uma espécie de base fundamental de infraestrutura, disponibilizando de forma virtualizada os Sistemas Operacionais (SOs). Assim, as VMs são usadas para fornecer abstração a nível de SO e também como forma de disponibilizar Plataforma como Serviço (*Platform as a Service – PaaS*).

Desenvolvedores têm focado na produtividade das nuvens PaaS, em detrimento do gerenciamento de armazenamento, poder computacional e rede (DUA; RAJA; KAKADIA, 2014). Surgiu, dessa maneira, a necessidade da utilização de contêineres como solução para as nuvens PaaS no lugar da virtualização de *hardware* com um SO completo em uma VM, uma vez que os contêineres fornecem capacidade de gerenciamento de pacotes e interoperabilidade nas aplicações em nuvem, além de consumir menos tempo e recursos (PAHL, 2015).

Segundo (PORTWORX, 2018a), tecnologias de virtualização baseadas em contêineres têm ganhado muito destaque nos últimos anos nas plataformas de nuvem e provavelmente continuarão nos próximos anos (REDHAT, 2018). Os contêineres fornecem um ambiente isolado para os recursos como sistemas de arquivos, processos e redes, para serem executados a nível do SO do *host*, evitando assim a execução de uma VM completa com seu próprio SO no *hardware*

virtualizado (FERREIRA; SINNOTT, 2019a).

Isso significa que vários contêineres têm o mesmo *kernel* do SO compartilhado entre si, iniciando mais rápido e utilizando apenas fração da memória do sistema quando comparado com a inicialização de um SO virtualizado. Estudos como o de (MORABITO; KJäLLMAN; KOMU, 2015) e (JOY, 2015) demonstram que os custos gerais são menores, além dos aplicativos terem seu desempenho otimizado.

Contudo, com o aumento significativo da utilização de contêineres, surge também a necessidade de ferramentas para facilitar o gerenciamento dos contêineres com o intuito de fazer o controle das tarefas de automatização da implantação, processamento, escalonamento dos aplicativos e operações gerais dos aplicativos em contêineres (JOY, 2015).

Dessa forma, diversas soluções surgiram, principalmente em volta das duas principais ferramentas de gerenciamento, que são Kubernetes e Docker Swarm, que de forma rápida se tornaram as mais utilizadas nesse contexto devido ao fato da eficiência na orquestração de contêineres e também ao ambiente de tempo de execução (*runtime*) (DIAMANTI, 2018a), (PORTWORX, 2018a) e (BLOG, 2014a). Segundo (JOY, 2015), o Kubernetes tem se tornado a solução mais abrangente devido a seus recursos avançados, robustez e maturidade, fora o fato de que livra o usuário de ter que manter infraestruturas e ainda fazer configurações complexas.

Nesse cenário, em que o Kubernetes tem se tornado parte essencial de infraestruturas computacionais que precisam funcionar ininterruptamente, com períodos longos de cargas pesadas de trabalho, é importante monitorar os efeitos de possíveis degradações de desempenho e, mais especificamente, do fenômeno conhecido como envelhecimento de *software*.

Este tipo de problema já foi identificado anteriormente em sistemas de computação em nuvem baseados em VMs e pode configurar-se em uma grave ameaça à disponibilidade e desempenho dos serviços oferecidos. Portanto, neste trabalho foi realizada uma avaliação de desempenho de sistemas executando o Kubernetes, a fim de detectar tais efeitos nocivos e com isso aplicar uma abordagem de rejuvenescimento de *software*, evitando interrupções repentinas do fornecimento de serviços alocados no *Cluster* Kubernetes.

1.1 Motivação

A utilização de contêineres, devido a sua leveza, inicialização rápida, implantação e isolamento de dependências, tem sido boa solução nos cenários de ambientes virtualizados e computação em nuvem, tanto no contexto de mercado quanto no âmbito acadêmico. À medida que o uso de contêineres aumenta, acarreta na necessidade de um gerenciamento eficaz para controlar as tarefas de automação, dimensionamento e implantação nas mais variadas aplicações (FERREIRA; SINNOTT, 2019b).

A Google, criadora do Kubernetes, é um claro exemplo de como a orquestração é

indispensável neste contexto (KUBERNETES, 2021)(BLOG, 2014b), pois a própria empresa oferece todos os seus serviços através do uso de contêineres, como mencionado em (GOOGLE, 2021), que afirma que eles já chegaram a executar mais de dois bilhões de contêineres toda semana.

No relatório *Hype Cycle to Cloud Computing*, a empresa de consultoria Gartner fez a constatação de que tecnologias para o gerenciamento de contêineres atingiram o auge das expectativas (RESEARCH, 2018). Assim, é válido observar que o mercado de orquestração de contêineres está muito competitivo e muito provavelmente continuará a se disseminar à medida que organizações no âmbito global consideram a abordagem baseada em conteinerização como um caminho a ser seguido.

Tecnologias como Docker Swarm e o Kubernetes têm sido adotadas como padrões de solução para o ambiente de tempo de execução e orquestração de contêineres, respectivamente, afirmam (DIAMANTI, 2018b), (PORTWORX, 2018b) e (FORRESTER, 2018). Diante disso, este trabalho propõe uma análise do desempenho e do possível surgimento de sinais dos efeitos oriundos do envelhecimento de *software*, ao se avaliar o comportamento do mecanismo de dimensionamento dos recursos automaticamente através do componente *autoscaling* do Kubernetes, baseando-se nas métricas de desempenho e exaustão de recursos para utilizar abordagens que evitem uma possível degradação de desempenho ou até mesmo uma falha abrupta no fornecimento dos serviços.

1.2 Objetivos

O foco deste trabalho de pesquisa é analisar o desempenho de sistemas executando o Kubernetes para orquestração de contêineres, com a finalidade de identificar possíveis sinais de degradação do desempenho ou implicações decorrentes dos efeitos do envelhecimento de *software* e com isso utilizar técnicas de rejuvenescimento de *software*. Abaixo seguem alguns objetivos específicos, traçados para alcançar êxito na proposta:

1. Analisar o comportamento do Kubernetes durante atividades típicas do processo de *autoscaling*
2. Identificar possíveis efeitos de envelhecimento em *Clusters* Kubernetes
3. Analisar a aplicabilidade de técnicas de rejuvenescimento de *software* em *Clusters* Kubernetes
4. Caracterizar possíveis diferenças de desempenho e da utilização de recursos do sistema entre as implementações do Kubernetes conhecidas como Minikube e K3S

1.3 Escopo Negativo

Este trabalho teve como objetivo avaliar o comportamento de *Clusters* Kubernetes na orquestração de contêineres em cenários distintos ao ter seu componente *autoscaling* submetido a alta carga de estresse, a fim de verificar o desempenho e disponibilidade de seus serviços, como também os efeitos do envelhecimento de *software* através de experimentos controlados. Infelizmente não foram executados experimentos em infraestrutura computacional mais avançada com alto poder de processamento, tendo em vista o alto custo financeiro para conseguir tal infraestrutura computacional.

Vale ressaltar que não deve ser considerado como finalidade deste trabalho apresentar uma nova solução padrão de rejuvenescimento de *software* que possa ser utilizada como padrão para todos os casos de envelhecimento, considerando-se a limitação dos recursos utilizados nos experimentos contidos no escopo deste trabalho.

1.4 Justificativa

Para se ter luz sobre a motivação, foi realizado um Mapeamento Sistemático (MS) da literatura para revelar o engajamento da comunidade acadêmica sobre o tema, quais os principais efeitos de envelhecimento de *software* têm sido mensurados e quais as principais abordagens são sugeridas como solução para os problemas resultantes do envelhecimento de *software*.

A partir do que foi obtido no mapeamento sistemático, percebeu-se a existência de diversos domínios de aplicação para os estudos de SAR (*Software Aging and Rejuvenation*) e, apesar da disseminação de ambientes virtualizados, há ainda uma necessidade de investigação do comportamento de soluções baseadas em orquestração de contêineres. É importante analisar o desempenho, disponibilidade e confiabilidade destas, quando expostas a cargas de trabalho intensas e prolongadas, levando-se em consideração o grande impacto da quebra de um desses aspectos em um ambiente de produção de grande escala que oferte serviços do Kubernetes.

1.5 Organização do Documento

Para proporcionar o melhor entendimento, este documento está estruturado em capítulos, que são:

- Capítulo 2 - Fundamentação Teórica, aborda a teoria dos assuntos desta pesquisa.
- Capítulo 3 - Mapeamento Sistemático, apresenta o estado da arte sobre o tema de SAR.
- Capítulo 4 - Trabalhos Relacionados, apresenta alguns trabalhos relacionados ao proposto.
- Capítulo 5 - Metodologia de Avaliação de Problemas de Desempenho e Envelhecimento em *Clusters* de Contêineres.

- Capítulo 6 - Avaliação Experimental, apresenta os detalhes do experimento controlado e resultados obtidos.
- Capítulo 7 - Conclusão, expõe considerações finais sobre as contribuições alcançadas, dificuldades encontradas e possibilidades de trabalhos futuros.

2

Fundamentação Teórica

Neste capítulo são descritos os conceitos fundamentais para melhor entendimento do trabalho. Estes são os principais tópicos estudados para o desenvolvimento da pesquisa, incluindo conceitos de Virtualização por Contêiner, Sistemas de Orquestração, como também das principais técnicas de detecção de Envelhecimento e Rejuvenescimento de *Software*.

2.1 Virtualização por contêiner

A computação em nuvem, um paradigma fortemente baseado em tecnologias de virtualização, vem auxiliando no melhor aproveitamento de recursos físicos e do consumo de energia em *data centers*. De acordo com (KON et al., 2017), a virtualização também oferece um ambiente portátil e flexível para diversas áreas, além de possibilitar a integração de aplicativos de modelagem, compartilhamento e interconexão de ferramentas computacionais.

A virtualização em nível do SO, conhecida igualmente como virtualização baseada em contêiner (MERKEL, 2014) e (SOLTESZ et al., 2007), é uma forma alternativa de menor custo quando comparada aos sistemas tradicionais de virtualização em nível de *hardware*, ou seja, baseados em *hypervisor* (MORABITO; KJäLLMAN; KOMU, 2015) e (XAVIER; NEVES; ROSE, 2014). A virtualização por contêiner têm vários benefícios, como eficiência maior no uso dos recursos e alto desempenho, requisitos necessários para tornar satisfatório o gerenciamento de dispositivos, independente do contexto.

Utiliza-se comumente o SO GNU/Linux, executando-o em contêineres com a disponibilização de isolamento entre os diversos ambientes de execução (SOLTESZ et al., 2007). (TACHIBANA; KON; YAMAGUCHI, 2017) diz que com a flexibilidade a partir da alocação de recursos por meio da virtualização, tal tecnologia tem sido amplamente implantada. Em (KOOMEY, 2008), (SHARMA et al., 2016), (MORABITO; KJäLLMAN; KOMU, 2015), (XAVIER;

NEVES; ROSE, 2014), (RAHO et al., 2015) e (XAVIER et al., 2013), muito se discute a respeito do desempenho dos contêineres.

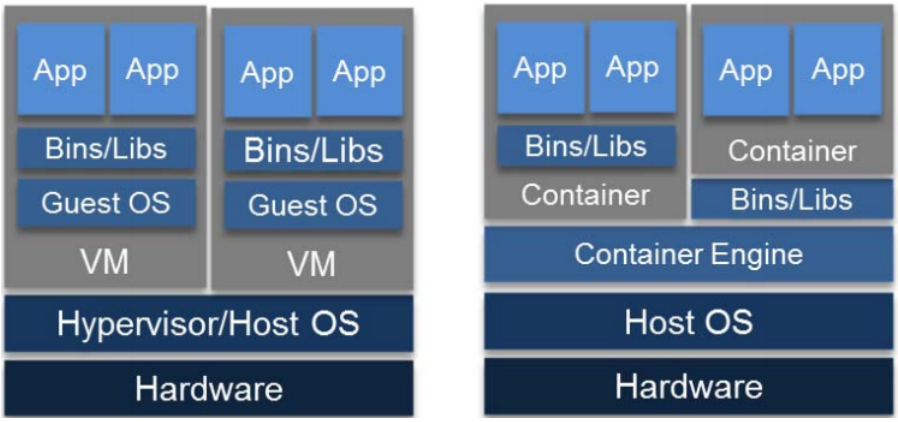
Sem dar ênfase aos servidores altamente consolidados, muitos trabalhos mostram que as implementações baseadas em contêineres são mais indicadas para atingir um bom desempenho em muitos contextos, entre eles o industrial, o de nuvem, de *data centers*, de dispositivos móveis e o de rede (VERMA et al., 2015) e (DRUTSKOY; KELLER; REXFORD, 2013).

Um contêiner consiste em todo um ambiente de execução: todas as dependências do aplicativo, binários, bibliotecas e arquivos de configuração necessários para a sua execução, agrupados em um pacote (MODAK et al., 2018). Os componentes de um aplicativo são empacotados como blocos de construção do aplicativo completo (MUDDINAGIRI; AMBAVANE; BAYAS, 2019).

Os contêineres provêm um ambiente similar a uma VM abstraindo o *hardware* por completo, porém sem a sobrecarga da execução do SO separado, isto é, executa o processo de forma discreta e leve. O SO da máquina hospedeira é compartilhado com outros contêineres diferentes instalados na mesma máquina (MUDDINAGIRI; AMBAVANE; BAYAS, 2019). Dessa forma, isolado do seu ambiente de origem, garante sua portabilidade para qualquer outro ambiente.

A execução de cada contêiner se equipara a um processo em nível de usuário no SO hospedeiro. Um contêiner tem na sua essência um agrupamento de partes de aplicativos autocontido e já pronto para ser implantado, capaz ainda de conter lógica de negócios, pacotes para rodar aplicações e *middleware* (SOLTESZ et al., 2007) e (PAHL; LEE, 2015). A Figura 1 expõe a diferença entre a tecnologia de VM e a de contêineres. A virtualização baseada em contêiner designa os recursos de *hardware* de acordo com a quantidade implementada de instâncias e com propriedades de isolamento, como observado em (XAVIER et al., 2013).

Figura 1 – Arquitetura de virtualização de VM à esquerda e de contêiner à direita.



Fonte: (PAHL; LEE, 2015)

A virtualização, com o objetivo de operar com diversos recursos, se dá através da

utilização de uma camada de *software* a mais com um nível acima do SO. Em (XAVIER et al., 2013), é visto que a virtualização baseada em *hypervisor* é uma técnica muito utilizada, que necessita de um monitor de VM do sistema físico implícito, porém situada em uma camada acima. As VMs, de forma individual, oferecem suporte para os sistemas operacionais executarem isoladamente e da mesma maneira é capaz de conter diversos sistemas operacionais convidados internamente no contexto de virtualização.

Em outras palavras, o contêiner faz a comunicação do SO do *hardware* de forma direta, focando no isolamento e na inicialização das aplicações com base na interoperabilidade nos ambientes oferecida pelo mesmo, enquanto que a VM comunica com o *hypervisor* primeiramente, para só depois se comunicar com a máquina criada. Após isso, o SO só iniciará quando se ativa de fato a VM, e só depois dessa etapa, inicializar então a execução das aplicações.

Ainda seguindo a linha de raciocínio de (XAVIER et al., 2013), com as tecnologias baseadas em contêiner, os processos convidados estão em um maior nível de abstração enquanto operam pela camada de virtualização no nível do SO. O uso de soluções com base em contêineres possibilita o uso de microsserviços em ambientes de hospedagem compartilhados, além também de uma implantação dos serviços de forma mais dinâmica, como afirma (KOZHIRBAYEV; SINNOTT, 2017).

2.2 Sistemas de Orquestração

(TOSATTO; RUIU; ATTANASIO, 2015) afirmam que a computação em nuvem tem sido revolucionada através da utilização de contêineres, por possibilitarem a diminuição significativa do tempo de início de atividade das instâncias e sobrecarga de armazenamento e processamento, além também de serem mais leves quando comparados às tradicionais VMs. (TOSATTO; RUIU; ATTANASIO, 2015) também diz que os sistemas de orquestração e automação avançada são as soluções valiosas que auxiliam os profissionais de TI (Tecnologia da Informação) a lidarem com *data centers* em nuvem de grande escala.

(TOSATTO; RUIU; ATTANASIO, 2015) dizem ainda que para interagir com pilhas de infraestrutura de TI, desde os serviços em nuvem até *data centers*, se faz necessário soluções de orquestração. Assim, os sistemas de orquestração precisam de infraestruturas escaláveis e elásticas que forneçam alocação dinâmica de recursos de forma rápida e no local certo.

Essas plataformas de orquestração de contêineres também podem ser definidas como uma estrutura que interage e gerencia os contêineres em escala. Tais plataformas, disponibilizam uma estrutura que define a implantação inicial do contêiner, facilitando o seu gerenciamento em larga escala, como uma entidade, com o objetivo de fornecer disponibilidade, rede e dimensionamento (KHAN, 2017).

Dessa forma, (LINK et al., 2019) afirmam ser necessário a utilização de orquestradores

de contêineres, como exemplo o Kubernetes (WIGGINS, 2017), para que o gerenciamento de contêineres seja automatizado, como também a replicação rápida quando houver falhas de sistema ou rede, além de seu balanceamento de carga. Com isso, utilizando as ferramentas de orquestração apropriadas, é plenamente possível que os usuários possam implementar e iniciar o uso dos serviços em qualquer plataforma em nuvem como também servidores (SUCHITRA, 2016).

É dito também em (SUCHITRA, 2016) que existem três particularidades na orquestração na nuvem que devem ser levadas em consideração, são elas: a orquestração de recursos, carga de trabalho e serviço. (TRUYEN et al., 2018) dizem que os sistemas de orquestração de contêineres em implementações em grande escala são fundamentais para atender a necessidade lógica de gerenciar importantes requisitos como custo-eficiência de recursos, elasticidade e resiliência. Ainda segundo (TRUYEN et al., 2018), os sistemas de orquestração de contêineres disponibilizam recursos como:

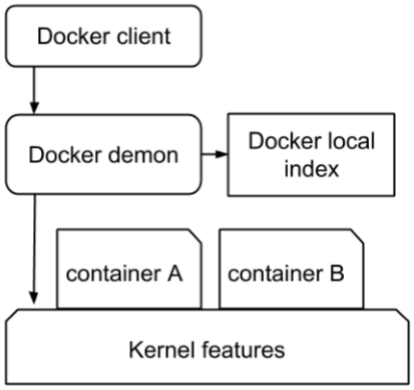
- Remoção automática de contêineres após a detecção automática de falhas de nó
- Um agendador central possibilitando a colocação de contêineres em nós, seguindo as restrições de posicionamento, especificadas pelo usuário
- Integração de serviços para a propagação persistente de volumes para armazenamento dos blocos de dados de disco com maior segurança, de maneira que possam ser recuperados após a falha de um nó
- *Proxy* de serviço de forma que cada serviço visível ao cliente possa ser acessado através do mesmo endereço IP *(Internet Protocol)*, mesmo após a migração de um contêiner para outro nó

Tais recursos são oferecidos por muitos sistemas de orquestração de contêineres, entre eles: Docker Swarm e Kubernetes. De acordo com uma pesquisa no meio dos administradores de nuvem OpenStack (OPENSTACK, 2017), para a execução de serviços a nível de produção em nuvens privadas OpenStack, os sistemas de orquestração de contêineres muito utilizados são: Kubernetes (DINESH, 2017), Docker Swarm (DOCKER, 2015b) e OpenShift 3.x (HAT, 2014).

O suporte à implantação e tarefas ou gerenciamento de cargas de trabalho orientadas a serviços, são oferecidos pelos sistemas Kubernetes e o Docker Swarm (HINDMAN et al., 2011). (TRUYEN et al., 2018) dizem que todos os sistemas de orquestração de contêineres incluem um serviço DNS (*Domain Name System*) interno, que possibilita a busca do endereço IP do *Cluster* de um serviço baseado no seu nome, que por sua vez é especificado na configuração do serviço.

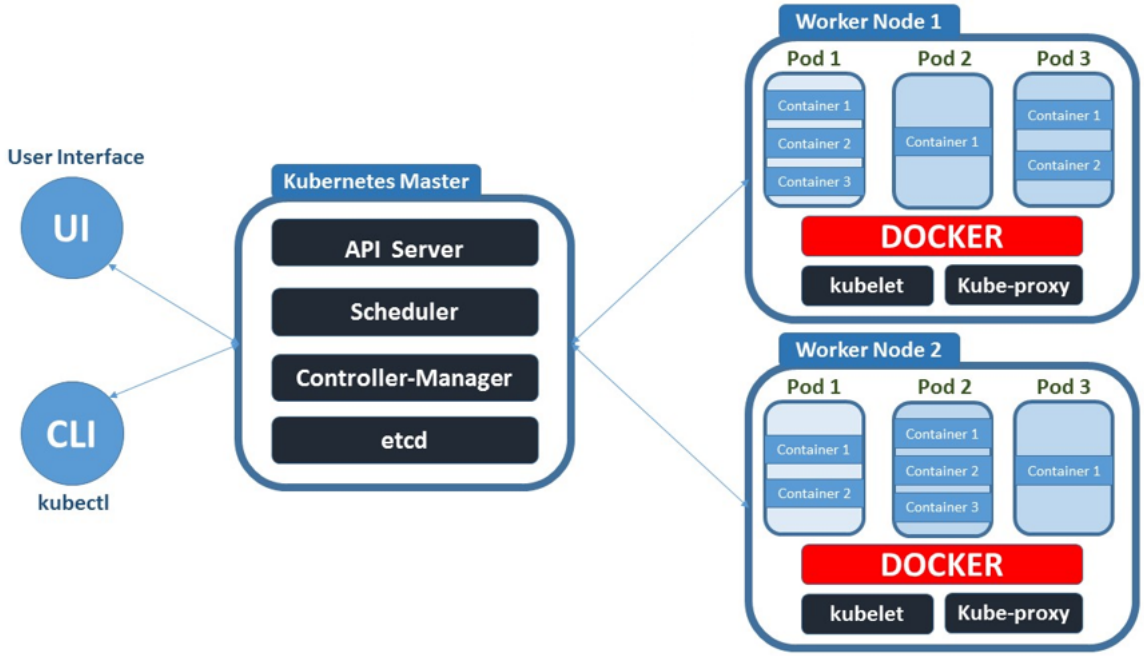
Nestes casos, um dos orquestradores de nuvem mais utilizados para soluções de virtualização em nível de sistema é o Docker Swarm (DOCKER, 2015a), que faz uso de contêineres Linux como meio de isolar dados, facilitar a implantação de aplicações de diferentes tipos e

Figura 2 – Arquitetura básica do Docker Swarm.



Fonte: (TOSATTO; RUIU; ATTANASIO, 2015)

Figura 3 – Arquitetura básica do Kubernetes.



Fonte: (LEARNITGUIDE, 2018)

também computação em recursos compartilhados. Quando um *daemon* do Docker Swarm, como representado na Figura 2, recebe uma solicitação de cliente para executar um contêiner, ele verifica a presença da imagem necessária. A imagem é então baixada do registro público se não já existir uma localmente para só então ser executada no contêiner (TOSATTO; RUIU; ATTANASIO, 2015).

A arquitetura básica do Kubernetes, ilustrada na Figura 3, fornece várias funcionalidades úteis e de refinamento para contêineres, como gerenciamento de rede, atualização contínua, registro de recursos, monitoramento, escalonamento automático, autocorreção, etc (WONG;

LEE, 2020). Ainda segundo (WONG; LEE, 2020), o padrão de arquitetura central aos sistemas baseados em Kubernetes e Docker Swarm é parecido: um *Cluster* de contêiner consiste em nós *Master* e *Workers*. O *Master* disponibiliza uma API *(Application Programming Interface)* onde é configurada de forma declarativa as aplicações distribuídas que se deseja, enquanto que no *Workers* é onde a aplicação realmente é executada no *Cluster* do kubernetes. A Figura 3, mostra a arquitetura básica do Kubernetes.

Contêineres que abrangem aplicativos de produção, devem ser implantados em diversos hospedeiros e através da orquestração por Kubernetes é possível gerenciar esses contêineres. Assim, é possível automatizar operações e remover muitos processos manuais (SHAH; DUBARIA, 2019).

Em (SHAH; DUBARIA, 2019), diz-se ainda que para trabalhar com Kubernetes é necessário conhecer alguns termos:

1) **Nó**: É uma máquina de operação. 2) ***Cluster***: Um conjunto de nós. 4) ***Pods***: são as unidades atômicas capazes de conter um ou mais recipientes. 5) ***Deployments***: fornecem a maneira mais comum de criar e gerenciar *Pods*. 6) ***Service***: Declara como os *Pods* podem ser acessados. No experimento serão executados diversos serviços propostos pelo Kubernetes, a fim de realizar análises sobre os mesmos e verificar alguns aspectos de degradação de desempenho, entre outros.

Em (KUBERNETES, 2020a) são disponibilizados os conceitos essenciais dos termos necessários para entendimento básico de alguns serviços ofertados pelo próprio Kubernetes, tais como:

- Descoberta de serviço e balanceamento de carga
- Orquestração de armazenamento
- Implementações e reversões automatizadas
- Gerenciamento de segurança e configuração

2.3 Minikube e K3S

O Kubernetes permite a implantação do seu *Cluster* de maneira física ou até mesmo virtual. Partindo desse entendimento, o Minikube cria uma VM localmente implantando o *Cluster* Kubernetes de maneira leve e simples com um único Nó. O Minikube está disponível para o SO Linux, macOS e Windows, fornecendo ainda operações como inicialização para trabalhar com o *Cluster* criado, como também a parada, exclusão e obtenção do *status* do mesmo para verificar a sua integridade e estado, entre outras coisas, tudo isso através da Interface de Linha de Comando (CLI) que é oferecida pelo Minikube (KUBERNETES, 2020b).

Como dito por (SOTOMAYOR et al., 2019), o Minikube encapsula os serviços dentro de Pods, disponibilizando esses mesmos serviços com resiliência ao ter o estado de saúde de cada um monitorado, enviando de forma contínua mensagens para verificar a disponibilidade do serviço, para acionar ações de estabilização da disponibilidade em caso de detecção de qualquer comportamento inesperado.

De maneira semelhante, O K3S é uma distribuição leve de Kubernetes, que opera com foco em áreas de aplicação de baixo custo, contendo todos os componentes que são padrões do Kubernetes visando disponibilizar um *Cluster* tolerante a falhas para um conjunto de nós, de maneira rápida, simples e eficiente, afirma (BÖHM; WIRTZ, 2021).

A distribuição do K3S é projetada para produção com cargas de trabalho em dispositivos autônomos ou dentro de dispositivos de Internet das Coisas (IoT) com recursos limitados. Com apenas um binário de tamanho menor que 50 MB, o K3S é encapsulado que diminui as dependências, além das etapas de instalação, execução e atualização do *Cluster* Kubernetes (K3S, 2021).

2.4 Envelhecimento e Rejuvenescimento de Software

Infraestruturas orientadas para a nuvem têm como requisitos fundamentais a confiabilidade e a disponibilidade, entretanto o aspecto de envelhecimento de *software* até então não tem sido tratado com atenção por muitos provedores de serviços (GROTTKE; JR; TRIVEDI, 2008). (GROTTKE; JR; TRIVEDI, 2008) e (BAO; SUN; TRIVEDI, 2005) analisaram e constataram que o papel do envelhecimento de *software* é importante na degradação desses requisitos e do desempenho de vários sistemas de *softwares*.

(GARG et al., 1998) afirma ainda que não somente os sistemas de *softwares* utilizados em grande escala, mas também os especializados utilizados em alta disponibilidade até aplicações críticas de segurança sofrem impactos do envelhecimento. (HUANG et al., 1995) diz que a partir do momento em que essa noção foi trazida a luz, despertou-se um grande interesse por esse fenômeno tanto do campo acadêmico quanto no industrial.

A literatura da área traz a documentação de ocorrência de envelhecimento de *software* em sistemas amplamente utilizados na indústria, a exemplo dos estudos realizados em (SYSTEMS, 2001), (GROTTKE et al., 2005) e (MATIAS; FILHO, 2006). Grande parte desses esforços de pesquisa têm como foco a compreensão dos efeitos empiricamente (GROTTKE et al., 2005), (MATIAS; FILHO, 2006) e (SILVA; MADEIRA; SILVA, 2006), mas também há importantes contribuições teóricas à compreensão deles, como em (BAO; SUN; TRIVEDI, 2005) e (SHERESHEVSKY et al., 2003). O *software* normalmente envelhece à medida que é executado, acumulando potenciais condições de falhas desde o início de sua operação, afirmam (GARG et al., 1995).

As interrupções em sistemas computacionais são mais devidas às falhas de *software* do que às falhas de *hardware*, afirmam (BAKER, 1995) e (XU; ZHANG, 2008). (ARAUJO et al., 2011a) também dizem que as falhas do sistema geralmente são mais frequentes do que as falhas causadas por componentes de *hardware*. A taxa com qual uma falha se torna ativa, depende da maneira e da forma como o sistema é utilizado, assim, o perfil operacional é uma caracterização quantitativa desse último aspecto (MUSA, 1993).

Erros numéricos, esgotamento dos recursos do SO e inconsistências de dados são alguns exemplos do acarretamento do envelhecimento de *software* (GROTTKE; JR; TRIVEDI, 2008). Seguindo essa linha, (GARG et al., 1998) dizem que corrupção de dados, fragmentação do espaço de armazenamento e bloqueios de arquivos são algumas características da degradação lenta do sistema. Essas falhas podem ser ditas como consequências naturais inerentes aos defeitos de *software* ou até mesmo uso de forma indevida pelos clientes (usuários). A discussão é abordada em (GROTTKE; JR; TRIVEDI, 2008), em termos de ameaças a patologia de uma falha, expondo as relações causais entre erro e falha.

O fenômeno de envelhecimento de *software* apareceu recentemente de forma que com o passar do tempo ou carga, as condições de erros aumentam, resultando dessa maneira na degradação do desempenho (MUSA, 1993). É dito por (AVIZIENIS et al., 2004) que o envelhecimento de *software* refere-se à degradação acometida pelo colapso do sistema ou acúmulo de erros nas aplicações, devido a erros inesperados no processo de operação contínua do sistema. Esse fenômeno se refere ao acúmulo de erros no momento da execução do *software*, resultando ocasionalmente em travamentos.

Técnicas de rejuvenescimento de *software* surgiram através da busca por mitigação das falhas (MATIAS; FILHO, 2006) e (VAIDYANATHAN; TRIVEDI, 2005). As técnicas pró-ativas de gerenciamento de falhas, como a de rejuvenescimento de *software* podem ser aplicadas de forma a neutralizar o envelhecimento, no caso de sua existência (HUANG et al., 1995). Segundo (GARG et al., 1998), uma maneira apropriada e muito utilizada no combate ao envelhecimento de *software* é o rejuvenescimento de *software*, aplicado por meio do reinício programado ou manual das aplicações ou até mesmo do dispositivo. Seguindo essa linha, (XIANG et al., 2018) também afirma que para diminuir as interrupções do serviço, o rejuvenescimento proativo é a técnica mais indicada, por tentar equilibrar o desempenho e o tempo de inatividade do sistema.

(HUANG et al., 1995) recomendam o rejuvenescimento de *software*, que ocasionalmente envolve a parada de execução do mesmo, para logo após eliminar as condições de erro acumuladas e então reiniciá-lo. A coleta de lixo, limpeza das tabelas do kernel do SO e a reinicialização das estruturas de dados, são alguns exemplos da limpeza do estado interno do *software*. A reinicialização do *hardware* é tido como um exemplo radical de rejuvenescimento (GARG et al., 1998).

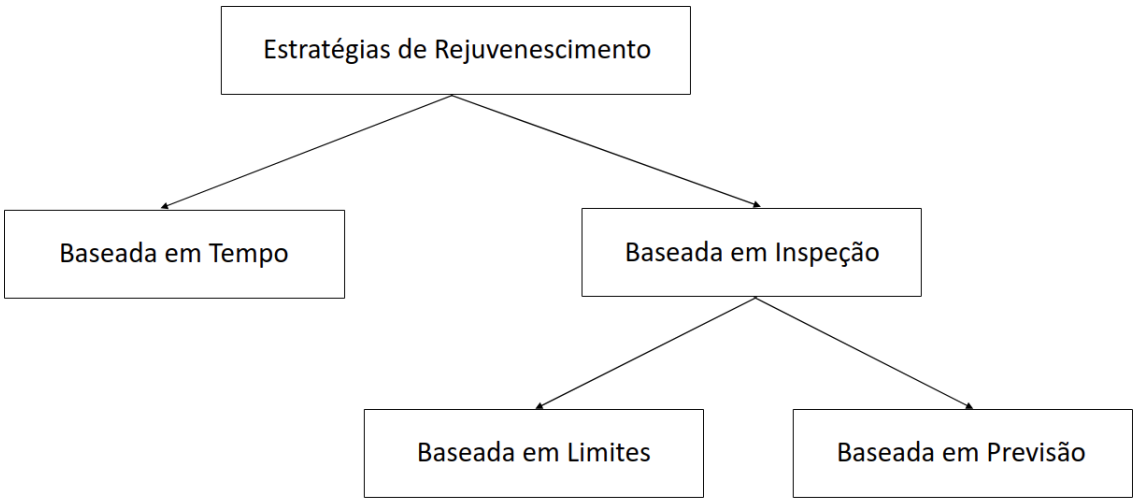
(COTRONEO et al., 2014) classifica o rejuvenescimento de *software* como técnica proativa ou reativa para evitar falhas relacionadas ao envelhecimento e também neutralizar os

seus efeitos, com técnicas de reinicialização automática ou manual em tempos determinados. (ALONSO et al., 2013) apresentaram seis técnicas distintas com diferentes níveis de granularidades e mostraram que o custo do desempenho do rejuvenescimento está relacionado a estes níveis.

(MACHIDA et al., 2012) propuseram uma abordagem para retardar o processo de envelhecimento, em contraste com as técnicas tradicionais de rejuvenescimento, que é a de alocar os recursos extras disponíveis para aplicações críticas, nomeada de extensão da vida do *software* (MACHIDA et al., 2017). Outro segmento, proposto por (ALONSO et al., 2007) e (ANDRZEJAK; MOSER; SILVA, 2007), faz uso da virtualização para mascarar o tempo inativo ocasionado pela reinicialização do SO ou até mesmo do próprio aplicativo.

Vários autores propuseram formas de representar o envelhecimento de *software* por meio de modelos analíticos e de simulação, aproveitando-os para verificar a eficácia de estratégias de rejuvenescimento. A Figura 4 demonstra a classificação de técnicas de rejuvenescimento de *software* proposta por (ARAUJO et al., 2011b).

Figura 4 – Classificação de técnicas de rejuvenescimento de *software*.



Fonte: (ARAUJO et al., 2011b)

Aqui, a abordagem Baseada em Tempo se dá quando uma ação é disparada em um tempo programado e de forma regular, enquanto a abordagem Baseada em Inspeção é definida como um monitoramento do estado do sistema e essa mesma abordagem é subdividida em duas outras, a Baseada em Limites e a Baseada em Previsão (ARAUJO et al., 2011b). A abordagem Baseada em Limites define limites a serem monitorados para quando algum deles for extrapolado, iniciar então uma ação de rejuvenescimento. Por fim, a abordagem Baseada em Previsão estima uma possível anomalia, através de modelos estatísticos ou também de aprendizagem de máquina (ALONSO; TORRES; GAVALDà, 2009)(ALONSO; BELANCHE; AVRESKY, 2011).

3

Mapeamento Sistemático

Neste capítulo é apresentado o modelo seguido para a realização de busca, seleção e requisitos utilizados no modo de filtragem dos trabalhos selecionados, visando identificar, analisar e interpretar os principais trabalhos disponibilizados de maior relevância na literatura a respeito do tópico de SAR.

No Mapeamento Sistemático (MS) há um procedimento de pesquisa consolidado possibilitando que outros pesquisadores, quando reproduzem o procedimento, consigam encontrar os mesmos resultados obtidos, por consistir em definir questões de pesquisa, realização de buscas, escolha de estudos de maior relevância, coleta dos dados e análise dos resultados, segundo (PETERSEN et al., 2008).

O MS é definido por (KITCHENHAM, 2004) como uma revisão geral dos estudos iniciais para um determinado problema específico de pesquisa, tendo como objetivo catalogar os estudos disponíveis e as principais abordagens utilizadas na literatura. Os estudos selecionados respeitaram os critérios pré-estabelecidos do MS, que estão descritos no decorrer deste capítulo.

3.1 Protocolo do mapeamento

Para a execução deste mapeamento, foi seguido o padrão definido por (KITCHENHAM, 2004) que implica em planejamento, condução e relatório do MS; essas são as três principais fases deste MS. Para auxiliar na condução do MS, haja vista as fases necessárias para seguir o caminho que se deseja, foi utilizada a ferramenta *online* Parsifal[1] (*Perform Systematic Literature Reviews*) por ser uma ferramenta que serve de suporte para diversos pesquisadores trabalharem e gerirem revisões ou MS ao mesmo tempo de forma *online*.

[1] Plataforma que oferece suporte total para pesquisadores geograficamente distribuídos para trabalhar na mesma Revisão Sistemática de Literatura.

Os objetivos atingidos e o protocolo que foi seguido durante o MS foram definidos nesta fase, concomitante com a forma de pesquisa como também a maneira de reprodução do estudo de forma transparente, sem esquecer de atentar para os resultados obtidos serem de alguma forma enviesados (STEINMACHER; CHAVES; GEROSA, 2013). Dessa forma, constituem o protocolo trabalhado neste MS: objetivos, questões de pesquisa, estratégia PICOC[2] (*Population, Intervention, Control, Outcome, Context*), *string* de busca, critérios de inclusão e exclusão dos trabalhos selecionados.

Vale ressaltar que as questões de buscas são um ponto fundamental ainda na parte de planejamento, devido ao fato delas serem o norte a ser seguido na realização da pesquisa. Uma vez bem definidas, possibilitam o direcionamento eficaz da pesquisa, sabendo onde se almeja chegar e qual problema tentar solucionar.

Com base no que diz (KITCHENHAM, 2004), os trabalhos primários selecionados para análises devem ir ao encontro e colaborar para a criação de respostas para as indagações de pesquisa do MS, evitando dessa maneira a dissonância dos mesmos com a estratégia de busca que foi planejada. A partir disso, foi utilizado a estratégia PICOC para definição das palavras-chaves que podem ser vistas na Tabela 1.

Tabela 1 – Método PIPOC

#	**Palavras-chave**
Population	Systems, Clouds, Orchestration, Clusters
Intervention	Container, Orchestration, Availability, Reliability Performance Degradation, Dependability
Control	Artigos da última década
Outcome	Aplicação de Domínio, Técnicas, Efeitos, Métricas
Context	SAR

Fonte: (COSTA et al., 2021)

Estes cinco componentes da estratégia PICOC e as palavras-chave, são elementos fundamentais que balizaram tanto na construção das questões de pesquisa contidas na Tabela 2, como na busca bibliográfica de evidências na literatura. Esta forma de mapeamento visando responder questões de pesquisa baseado nas evidências localizadas na literatura é definida por (KITCHENHAM, 2004) como um paradigma de engenharia de *software* baseada em evidências.

Tendo as questões elaboradas, foi possível então formular a *string* de busca, que foi constituída também com a utilização de operadores lógicos como *AND* e *OR* a fim de ter um melhor direcionamento nas buscas e obter resultados condizentes com o tema. A partir disso, foram definidas as principais fontes de pesquisa no âmbito da computação pensando-se estritamente na relevância para a área.

Através dos portais na *Internet* disponibilizados pelas bases, que estão listadas na Ta-

[2] Método usado para descrever os cinco elementos de uma pergunta de pesquisa.

Tabela 2 – Questões de pesquisa e levantamento do mapeamento sistemático

#	**Questões**
1	A quantidade de publicações por ano a partir de 2010 até 2020?
2	Quais os principais domínios de aplicação em SAR?
3	Quais as principais técnicas de modelagem adotadas?
4	Quais os principais efeitos de envelhecimento que foram avaliados?
5	Quais as principais abordagens de rejuvenescimento que foram utilizadas?

Fonte: O Autor

bela 3, foram realizadas as buscas considerando um intervalo de tempo, mais especificamente a última década para filtrar nos trabalhos mais recentes sobre o tema. A Tabela 3 contém a *string* de busca, como também as bases utilizada nas buscas deste MS.

Tabela 3 – String e fontes de buscas

#	**String**
	"("software aging") AND ("performance degradation "OR availability OR reliability OR dependability)"
Fonte	**URL**
Springer Link	http://link.springer.com
Science Direct	http://www.sciencedirect.com
IEEE Digital Library	http://ieeexplore.ieee.org
ACM Digital Library	http://portal.acm.org

Fonte: O Autor

Neste momento, também se fez necessário definir um dos passos mais importante nesta etapa, que foram os critérios de inclusão e exclusão que serviram para classificar os estudos primários para serem analisados no decorrer no MS, além disso, atentando-se ao fato de que o mesmo deve possibilitar uma replicação fidedigna. As Tabelas 4 e 5 a seguir apresentam respectivamente, 4 critérios de inclusão e 7 critérios de exclusão.

Tabela 4 – Critérios de Inclusão

#	**Critérios de Inclusão**
1	Artigos completos
2	Englobar ao menos duas palavras-chave
3	Possuir resumo contextualizando o trabalho
4	Ter sido publicado nos últimos 10 anos

Fonte: O Autor

Uma vez os critérios de inclusão definidos como constam na Tabela 4 , foi a vez então da definição dos critérios de exclusão expostos na Tabela 5, na tentativa de refinar o máximo possível a pesquisa almejando melhor qualidade nos resultados.

Tabela 5 – Critérios de Exclusão

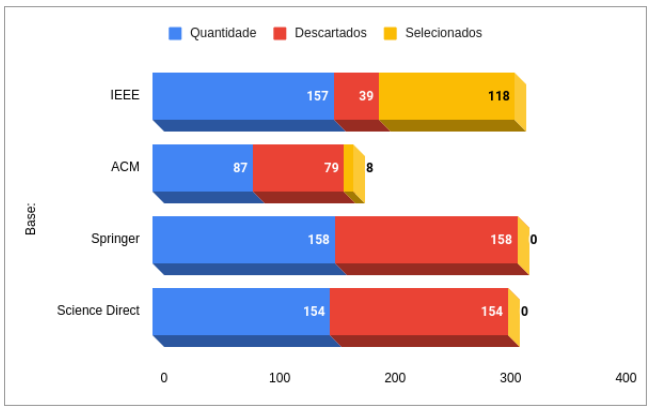
#	**Critérios de Exclusão**
1	Revisão Sistemática
2	Conferências abaixo da 5ª edição
3	Apenas resumo
4	Duplicados
5	Fora do tema
6	Artigos curtos
7	Survey

Fonte: O Autor

Vale ressaltar que conferências abaixo da quinta edição, foi tido como critério de exclusão visando a busca por conferências sólidas na área de Ciência da Computação. Outro aspecto que segue o mesmo princípio de critério de exclusão é o de fator de impacto com baixa relevância na literatura, as quais eram avaliadas abaixo de 1 ponto.

Com essas definições de protocolo já consolidadas, deu-se então início à fase de execução da *string* nas fontes de buscas já citadas a partir de junho de 2010, e para organizar a pesquisa realizada em cada fonte como também os resultados retornados de cada uma, foi utilizada a ferramenta de uso comum entre os pesquisadores em suas revisões e também em MS, a *Parsifal* para auxiliar neste caso, na importação de referências no padrão *BibTeX*, documentação dos resultados, entre outras funcionalidades.

Figura 5 – Trabalhos encontrados, selecionados e descartados.



(COSTA et al., 2021)

As buscas realizadas obtiveram um total de 556 artigos selecionados com potencial relevância para análises baseadas nos critérios de inclusão e exclusão estabelecidos, que culminaram na seleção de 126 artigos para leituras e estudos como podem ser observados na Figura 5 que apresenta os resultados obtidos em cada fonte de busca, a quantidade de trabalhos encontrados, selecionados e descartados.

Este MS abrangeu 556 trabalhos, dos quais 126 foram selecionados por atenderem aos

critérios estabelecidos durante o planejamento. De posse desses trabalhos, foi possível analisar e ter uma visão geral e principalmente sobre o atual estado da arte relacionada ao tema de pesquisa, possibilitando a identificação dos principais efeitos do envelhecimento e lacunas neste contexto, que estão descritas abaixo:

1. Domínio de aplicação (ambiente): A categorização dos trabalhos analisados culminou em 7 domínios principais até então pesquisados, entre eles, o de trabalhos teóricos foi o mais predominante com abordagens em modelos baseados em estados devido a necessidade da adequação das ferramentas matemáticas para os problemas do SAR e por serem as áreas de pesquisas originais dos principais grupos acadêmicos no campo do envelhecimento e rejuvenescimento de *software*. As categorias de computação em nuvem e ambientes virtualizados baseados em contêiner tornaram-se um foco recente de alguns esforços em pesquisas voltadas para o tema SAR, tendo ainda questões em aberto para aprofundamento.

2. Efeitos do envelhecimento: As métricas envolvidas nos sistemas que sofrem impactos do envelhecimento de *software* encontradas neste MS foram relacionadas à disponibilidade, confiabilidade, desempenho e exaustão de recursos. Sendo que as mais investigadas até então foram a de degradação de desempenho seguida por exaustão de recursos, ambas efeitos do envelhecimento do *software* que podem ser verificadas como o efeito acumulado da ativação de falhas de *software* durante o longo tempo de execução do sistema.

Os 126 trabalhos aqui selecionados atenderam de alguma forma a critérios de inclusão definidos na Tabela 4, e os demais trabalhos foram descartados por causa de pelo menos um dos itens dos critérios de exclusão definidos na Tabela 5, não tratando do tema de interesse. Todos esses critérios visam o refinamento da busca de modo a ter os trabalhos de maior relevância no MS.

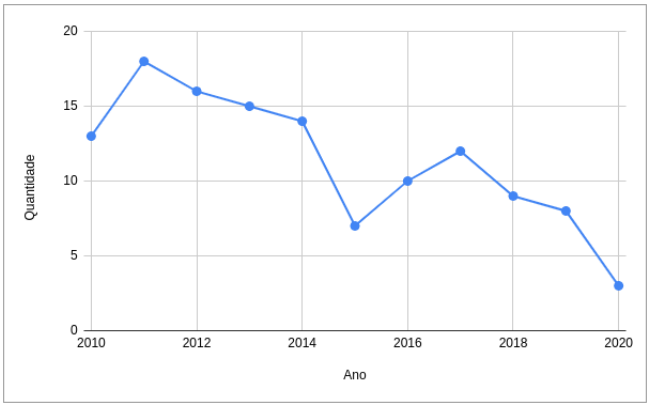
3.2 Análises quantitativas

A quantidade de trabalhos publicados por ano foi um dos resultados encontrados em resposta a primeira questão descrita na Tabela 2, com base na *string* de busca proposta com o intuito de se ter uma noção do engajamento da comunidade acadêmica no tema pesquisado. Neste caso a Figura 6 mostra a quantidade de trabalhos publicados entre 2010 e 2020.

Na Figura 6 pode ser visto que a quantidade de publicações por ano em 2011, foi o ano mais produtivo com 18 trabalhos publicados. Uma queda pode ser notada na quantidade de publicações sobre o tema principalmente em 2014, com uma retomada no ano seguinte em 2015. Entre 2016 a 2019 o número de publicações variou em torno de 10 por ano. Desses, apenas 3 artigos foram encontrados publicados em 2020, de janeiro a junho, embora não saibamos se essa queda se deu até o fim do ano de 2020 e também se foi impactada pela pandemia de COVID-19.

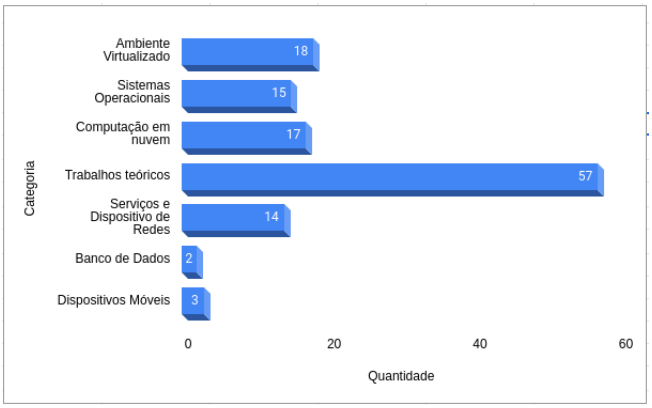
A categorização de domínio de aplicação pode ser vista na Figura 7, na qual podem ser

Figura 6 – Quantidade de trabalhos publicados entre 2010 até 2020.



Fonte: (COSTA et al., 2021)

Figura 7 – Quantidade por categoria de Domínio.



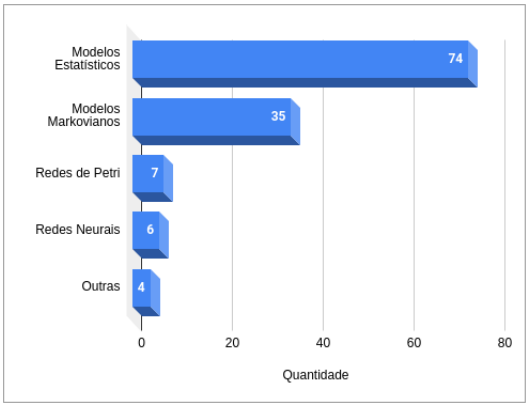
Fonte: (COSTA et al., 2021)

notadas as 7 categorias de aplicação encontradas nas buscas realizadas, entre elas: ambientes virtualizados; sistemas operacionais; computação em nuvem; trabalhos teóricos; dispositivos e serviços de rede; bancos de dados; e dispositivos móveis. Os trabalhos teóricos são predominantes, totalizando 57 trabalhos.

Trabalhos teóricos entende-se aqui como aqueles trabalhos que não realizam experimentos ou propõem modelos e estratégias para qualquer aplicação específica de domínio. Os outros 69 trabalhos focaram em aplicações de algum contexto especifico. Ambientes virtualizados e computação em nuvem captaram grande foco, com 18 e 17 artigos, respectivamente.

Aqui vale ressaltar que durante o processo de classificação, essas categorias foram separadas devido ao fato de muitos trabalhos estarem focados em *hypervisores* ou outras tecnologias em virtualizações sem uma relação direta com as infraestruturas de computação em nuvem. A terceira categoria mais pesquisada foi relacionada aos SOs, com 15 trabalhos. Sistemas de banco de dados e dispositivos móveis foram as categorias com as menores quantidades de trabalhos encontrados: 2 e 3, respectivamente. Respondendo dessa forma a questão 2, descrita na Tabela 2.

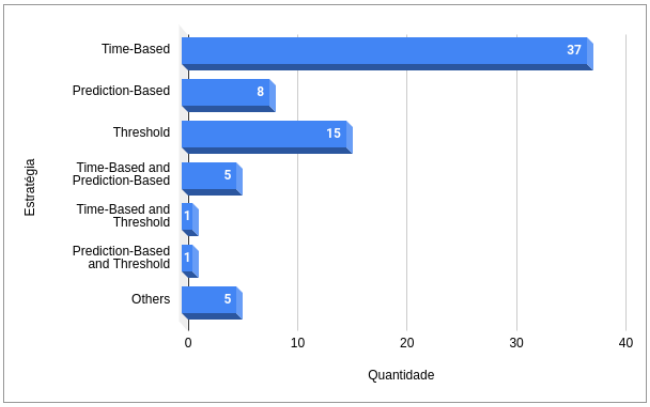
Figura 8 – Quantidade por tipo de técnicas de modelagem.



Fonte: (COSTA et al., 2021)

A Figura 8 mostra as principais técnicas de modelagem adotadas nos trabalhos selecionados para avaliação, como resposta para a questão 3 descrita na Tabela 2. A sua maioria utiliza algum tipo de modelo estatístico, totalizando 74 artigos. Em seguida, 35 artigos adotam um modelo Markoviano, como processos Markovianos e Semi-Markovianos. As redes de Petri estão presentes em 7 trabalhos, abordando *Stochastic Petri Nets* (SPN), *Deterministic and Stochastic Petri Nets* (DSPN), *Coloured Petri Nets* (CPN) e *Stochastic Reward Nets* (SRN). Dos trabalhos encontrados, 6 adotam algum tipo de redes neurais artificiais, geralmente para prever o comportamento do sistema e em conjunto com alguma técnica de rejuvenescimento. Por fim, os quatro trabalhos restantes foram classificados em "Outras", por não estarem adotando nenhuma das técnicas mencionadas acima.

Figura 9 – Quantidade por tipo de técnicas de Rejuvenescimento.

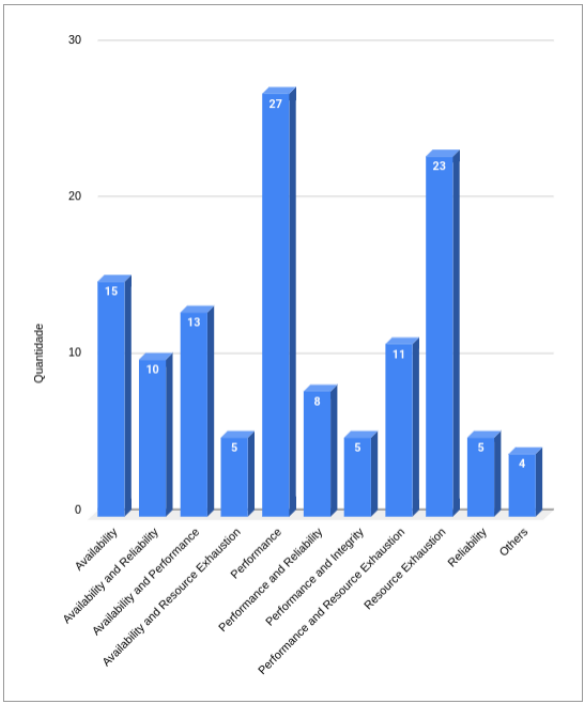


Fonte: (COSTA et al., 2021)

Na Figura 9 são apresentadas as estratégias de classificação de rejuvenescimento como proposto por (ARAUJO et al., 2011b), que segmenta as abordagens em *Time-Based e Prediction-Based*, sendo este último também subdividido em duas outras estratégias: *Threshold* e *Prediction-Based*. A estratégia mais adotada é a *Time-Based* com 37 artigos, seguida de *Threshold* com

15 artigos e, em seguida, *Prediction-Based* encontrada em 8 trabalhos dos selecionados. Há também as estratégias híbridas: *Time-Based* e *Prediction-Based* com 5 artigos, *Time-Based* e com *Threshold*, assim como *Prediction-Based* e *Threshold*, ambos sendo encontrados em apenas um trabalho. Tais estrategias respondem a questão de número 5, descrita na Tabela 2

Figura 10 – Quantidade por tipo de Efeitos de Envelhecimento.



Fonte: (COSTA et al., 2021)

A Figura 10 apresenta a resposta para a questão 4 descrita na Tabela 2, abordando o impacto do envelhecimento do *software* nas seguintes métricas do sistema: *availibility*, *reliability*, *performance*, e *resource exhaustion*. Os efeitos sobre o *performance* foram os mais investigados com 27 artigos, seguido de *resource exhaustion* com 23 e *availibility* com 15 trabalhos. Alguns trabalhos investigaram mais de uma métrica no mesmo estudo por exemplo *"availibility e performance"*; "*performance e resource exhaustion*"; e "*availibility e reliability*" foram os mais investigados, com 13, 11 e 10 artigos respectivamente. Por fim, 4 artigos foram classificados como *Others*.

4

Trabalhos Relacionados

Durante a realização de estudos voltados para o tema do MS como também do trabalho, foi constatada a existência de alguns trabalhos voltados para o tema de avaliação de sinais dos efeitos de envelhecimento e de desempenho após aplicação de técnicas de rejuvenescimento de *software*, que são abordados a seguir.

(BAI et al., 2020) analisa quantitativamente as técnicas de rejuvenescimento de *software* de provedores de serviços e visualizações dos usuários em um sistema virtualizado implantando técnicas de reinicialização do VMM (*Virtual Machine Monitor*) e migração de VM inteira rejuvenescida, sob a condição de que todo o tempo de envelhecimento, tempo de falha, tempo de correção do VMM e tempo de migração da VM seguem distribuições gerais. Assim, construíram um modelo analítico usando um processo semi-Markoviano e derivou fórmulas para calcular a disponibilidade do serviço da aplicação e o tempo de conclusão do trabalho.

Um estudo experimental de envelhecimento e rejuvenescimento de *software* com o foco no *daemon dockerd*, processo responsável pelo gerenciamento de contêineres do Docker Swarm, é apresentado por (TORQUATO; VIEIRA, 2019) que através da utilização da abordagem SWARE conduz o experimento, englobando três fases: i) estresse - ambiente de estresse com carga de trabalho acelerada para induzir a ativação de *bugs*; ii) aguardar - interrompe o envio da carga de trabalho para observar possíveis efeitos acumulados e por último; iii) rejuvenescimento - apresenta uma ação de rejuvenescimento para perceber mudanças no estado interno do *software*.

(TORQUATO; MACIEL; VIEIRA, 2019) em seu trabalho propõe uma abordagem de avaliação de segurança com base em um modelo de disponibilidade para sistemas virtualizados, com migração de VM para rejuvenescimento do VMM, a fim de encontrar o agendamento adequado do rejuvenescimento para atingir os níveis desejados de segurança e disponibilidade.

Um algoritmo de estimativa para distribuição do tempo de resposta é ponderado por (OKAMURA; LUO; DOHI, 2013) com uma aplicação em um ambiente que sofre os efeitos do

envelhecimento de *software*, através do desenvolvimento da cadeia de Markov de tempo contínuo, de maneira a representar o estágio da degradação do servidor para mostrar a distribuição do tempo de resposta caracterizada por um processo composto de Poisson.

(KHOSHMANESH; LUTZ, 2018) impulsiona o conhecimento das interações de recursos anteriores em uma linha de produtos já existentes, juntamente com medidas de similaridade com os novos recursos a serem criados, para detectar semelhanças em um novo produto já no processo de desenvolvimento, visando detectar interações problemáticas no intuito de reduzir o risco de envelhecimento de *software* no produto antes mesmo de ser lançado.

Uma abordagem sistemática para acelerar a manifestação do envelhecimento do *software*, com o intuito de reduzir o tempo de experimentação e assim estimar a distribuição do tempo de vida do sistema investigado é apresentada por (MATIAS et al., 2010), que introduzem o conceito de fator de envelhecimento. Essa abordagem oferece um controle dos efeitos do envelhecimento em nível experimental e estima, por meio de análises de sensibilidade, a distribuição do tempo de vida de um servidor web com sintomas de envelhecimento de *software*.

(REINECKE; WOLTER, 2010) verificam a eficácia da reinicialização do lado do cliente e do rejuvenescimento do lado do servidor em um sistema que sofre com o envelhecimento do *software*, com o foco nos vazamentos de memória, avaliando os modelos implícito e explícito para envelhecimento de *software*, considerando a disponibilidade do sistema e serviço.

(YANG et al., 2010) investigam em seu trabalho a política de rejuvenescimento de *software* para o sistema de computação em *Cluster* com os nós tendo uma relação de dependência, e reconstrói um modelo SRN para o rejuvenescimento de *software* para este cenário, considerando ambos os tipos de nós diferentes e a relação de dependência entre os nós, diminuindo a taxa de falhas e aumentando a disponibilidade do sistema.

A escalonabilidade de uma tarefa em tempo real é avaliada em (HUA et al., 2017), que define um recurso que apresenta degradação de desempenho ao longo do tempo com um padrão conhecido e usa rejuvenescimentos periódicos com abordagem Baseada no Tempo como contramedida, visando reduzir o impacto do envelhecimento de *software* na degradação de desempenho.

(TORQUATO et al., 2017) propõem em seu trabalho uma abordagem com base em experimentos e observação empírica para investigar o envelhecimento e rejuvenescimento em sistemas de *software*, através de três fases: (i) Fase de Estresse - ambiente de estresse com carga de trabalho acelerada induzindo a ativação de *bugs*; (ii) Aguardar - interrompe o envio da carga de trabalho para observar o impacto causado no status do *software*; (iii) Fase de Rejuvenescimento - apresenta uma ação de rejuvenescimento para observar mudanças que podem surgir no estado interno do *software*.

(SUKHWANI et al., 2017) investiga falhas causadas pelo envelhecimento de *software* que afetam um sistema controlador de nuvem e descreve um serviço desenvolvido para analisar

continuamente métricas do sistema/aplicativo dos sistemas do cliente, identificando possíveis cenários de falha relacionados ao envelhecimento com o passar de dois dias e gera uma lista de tarefas para a equipe de operações de desenvolvimento da IBM para mitigar as possíveis falhas.

Em (WENG et al., 2017), os autores realizam experimentos para simular o comportamento de um usuário no Android e encontrar fenômenos de envelhecimento de *software* no Android, utilizando aprendizado ativo baseado em floresta aleatória e quatro estados para construir o modelo e desenvolver uma estratégia de rejuvenescimento de *software* no sistema.

É apresentado por (TORQUATO; VIEIRA, 2018) um conjunto de modelos analíticos para avaliar a disponibilidade de um sistema virtualizado com VMM através do rejuvenescimento de *software* habilitado por agendamento com a migração de VM, utilizando modelos com aspectos de carga de trabalho variável adotando modelos interativos para reduzir o tempo e o custo de avaliação dos modelos, considerando a variação da carga de trabalho com dois estágios: i) pico - quando a carga de trabalho submetida é alta; e ii) fora de ponta - quando a carga de trabalho apresentada é baixa.

No trabalho de (DUA; RAJA; KAKADIA, 2014), a virtualização de contêineres de máquinas virtuais tem foco na comparação de desempenho utilizando cargas de trabalho de modo que os recursos de CPU (*Central Process Unit*) venham ser sobrecarregados, além do armazenamento, rede e memória.

(SOLTESZ et al., 2007) e (COREOS, 2015) propõem uma abordagem cíclica e interativa, partindo da especificação de algoritmos funcionais. (PENG et al., 2009) considera modelos formais no estudo de efeitos de envelhecimento de *software* sobre o desempenho em tecnologias de computação em nuvem, observando a execução de máquinas em nuvens, com o objetivo de avaliar o comportamento de seu desempenho.

(AMARAL et al., 2015) faz uma avaliação de desempenho de contêineres com Docker Swarm e evidencia a degradação de desempenho da rede e também seu impacto nos recursos de CPU. Discute-se uma comparação entre VMs e contêineres Linux, realizada por (JOY, 2015) observando os termos de escalabilidade e desempenho.

(FELTER et al., 2015) compara os contêineres do Linux com o desempenho obtido das implantações e dos aplicativos das VMs. Uma abordagem para a migração entre VMs de *hypervisor* para contêineres é feita por (LINTON D REKESH, 2017). (GUPTA, 2015) apresenta o desempenho no quesito segurança ao comparar VMs e contêineres Linux.

O desempenho de três redes populares do Kubernetes tiveram suas soluções examinadas por (ZENG et al., 2017). Os autores comparam a taxa de transferência média e latência dos plug-ins CNI (*Container Network Interface*) do Kubernetes.

Foram observados também problemas envolvendo envelhecimento de *software* em sistemas de *software* orientados para nuvem de código aberto (MACHIDA et al., 2012), também em sistemas de virtualização (MACHIDA; KIM; TRIVEDI, 2010) e em sistemas de *middleware*

(CARROZZA et al., 2010). No trabalho realizado por (ARAUJO et al., 2014) é relatada a detecção de envelhecimento de *software* na utilização da memória do controlador em um dos nós, além de outros componentes no sistema Eucalyptus, o gerenciador de nuvem.

(NAGARAJU; BASAVARAJ; FIONDELLA, 2016) em seu trabalho avalia o impacto de falhas correlacionadas à otimalidade da técnica rejuvenescimento por agendamento, em um sistema composto por dois servidores paralelos que podem sofrer falhas correlacionadas com essa abordagem, identificando objetivamente o período ideal de rejuvenescimento que maximiza a disponibilidade ou minimiza o custo do tempo de inatividade.

Uma estratégia de rejuvenescimento em dois níveis é apresentada por (GUO et al., 2015), intercalando um conjunto de n rejuvenescimentos quentes com um rejuvenescimento frio, para encontrar o n ótimo que maximize o desempenho médio do sistema, visando combater a degradação do desempenho dos recursos e, ao mesmo tempo, manter o desempenho médio dos recursos maximizado.

(YAKHCHI et al., 2015) propõe uma comparação entre diferentes técnicas de aprendizagem de máquina para prever com precisão o tempo de falha do *software* em diferentes cenários de envelhecimento, com o intuito de prever o tempo de falha devido a fatores de envelhecimento, como vazamentos de memória.

Diante das análises dos trabalhos verificados até aqui, por serem os mais próximos ao trabalho realizado nesta pesquisa, que estão relacionados no Apêndice A, observou-se uma lacuna quanto as avaliações próprias no *Cluster* do Kubernetes, voltadas para identificar a degradação do seu desempenho no decorrer de sua execução e a falta de uma abordagem a nível de mitigação da possível queda de desempenho através das técnicas de rejuvenescimento de *software*.

5

Metodologia de Avaliação de Problemas de Desempenho e Envelhecimento em *Clusters* de Contêineres

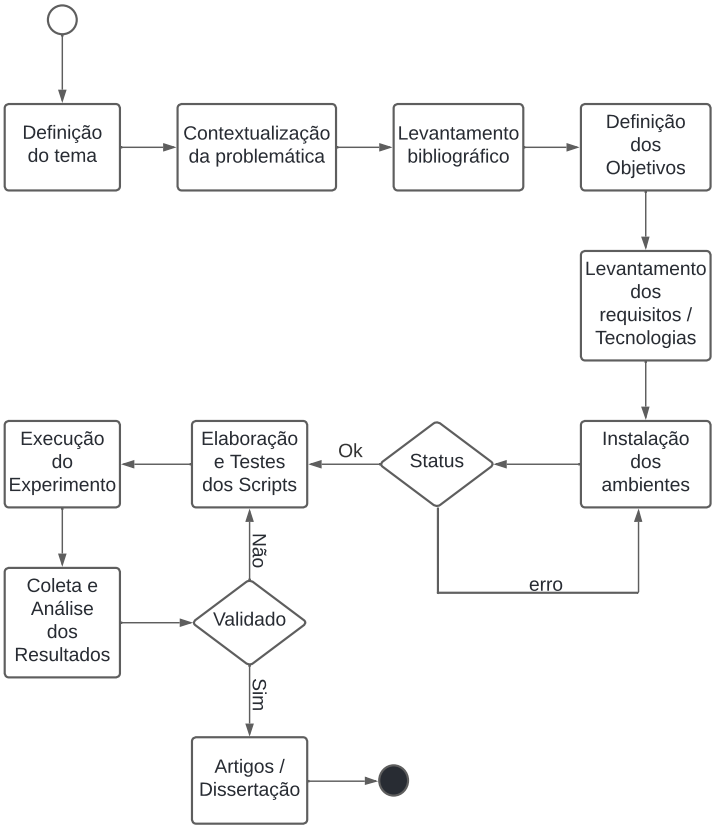
Nesta seção é apresentada a metodologia que foi seguida a fim de alcançar as respostas, para as questões levantadas como parte dos objetivos traçados deste trabalho, descrevendo a organização, cenários e ambientes para execução do experimento, bem como as tecnologias e algoritmos utilizados que viabilizaram a realização do experimento proposto neste trabalho.

A metodologia é baseada em uma avaliação experimental aplicada fazendo medição dos recursos utilizados do sistema e de seu desempenho em um ambiente conteinerizado com um *Cluster* Kubernetes, utilizando as ferramentas Minikube e K3S que executaram a mesma carga de trabalho.

Na Figura 11 são apresentadas todas as etapas seguidas neste trabalho, de modo a elucidar as 10 fases elaboradas visando lograr êxito nos objetivos. A fase inicial, que foi a definição do tema, levou ao direcionamento da área de pesquisa que teve como requisito ser um tema ainda não tanto explorado como também fosse de grande valia não só em um contexto de mercado tecnológico, mas também no contexto acadêmico.

Após a escolha do tema, foi a vez então de definir a contextualização do mesmo, isto é, em qual contexto poderia ser aplicado o tema de pesquisa de modo a ter maior relevância para o seu cenário, que pode ser vislumbrado através da execução da fase de levantamento bibliográfico, onde verificou-se o estado da arte sobre o tema em trabalhos filtrados pelos critérios de inclusão expostos na Tabela 4.

Figura 11 – Diagrama da metodologia.



Fonte: O Autor

De posse dos trabalhos, foi possível verificar quais aspectos envolvidos e lacunas ainda não exploradas sobre o tema em questão, permitindo com isso, a elaboração dos objetivos e metas a serem alcançados, de maneira a gerar uma contribuição não apenas a nível de mercado mas também na diminuição das lacunas ainda existentes na sua literatura, fase esta denominada de definição dos objetivos.

A partir disso, deu-se início à fase de levantamento dos requisitos necessários, como também o levantamento da tecnologia que melhor se enquadrasse para o desenvolvimento dos *scripts* visando a criação dos ambientes e a execução de todo o experimento. Uma vez definidos os requisitos e tecnologias, a fase de instalação e preparação do ambiente foi iniciada, com o intuito de preparar o ambiente do Minikube e do K3S para a execução dos *scripts* e experimento propriamente dito.

Mas antes de seguir para a próxima fase, se fez necessário a verificação do *status* de todo o ambiente, a fim de verificar o seu correto funcionamento, para atestar a plena capacidade de operação das rotinas que seriam programadas. Em caso de detecção de qualquer comportamento inesperado, todo o processo de instalação seria refeito.

Com todo o ambiente funcionando corretamente, foi iniciada a fase de elaboração e testes dos *scripts* com o desenvolvimento dos mesmos, tanto os *scripts* de carga quanto os de monitoramento dos ambientes visando a medição de métricas sugeridas pelo MS. A fase de execução do experimento começou logo após a construção dos *scripts*. Nela, eles foram executados repetidas vezes conforme o planejamento, iniciando as ferramentas com os *Clusters* para que o estresse derivado da carga ocorresse, além de seu monitoramento.

Concluída a fase de execução do experimento, foi possível fazer a coleta e análise dos resultados advindos da execução para fazer a validação dos mesmos, verificando a correta execução das rotinas programadas na fase de elaboração dos *scripts*. Em caso de alguma ocorrência de resultados fora do escopo do projeto, refez-se então todo o caminho de elaboração e testes dos *scripts* até a fase de coleta e análise de resultados até que os resultados estivessem de acordo com o escopo esperado.

Por fim, após verificar o correto escopo dos resultados na validação da coleta e análise dos resultados, iniciaram as escritas tanto dos artigos como também da dissertação para submissão em revistas e conferências, finalizando assim a fase de artigos e dissertação.

5.1 Quais aspectos/componentes/cenários do sistema pretende-se avaliar?

Os experimentos foram conduzidos com a utilização de *scripts* desenvolvidos para contextos que simulassem requisições de serviço, enviadas ao *Cluster* Kubernetes, onde o mesmo pode ser acessado de forma remota através da *Internet*, possibilitando a recepção de altas cargas de requisições, sobrecarregando o sistema.

Neste cenário, espera-se que a grande quantidade de requisições recebidas resulte em muita demanda de processamento por parte do sistema sem que o mesmo perca a capacidade de permanecer com o serviço sempre disponível, de modo a manter também a sua confiabilidade, pré-requisitos fundamentais a cada dia que passa em qualquer sistema de computação.

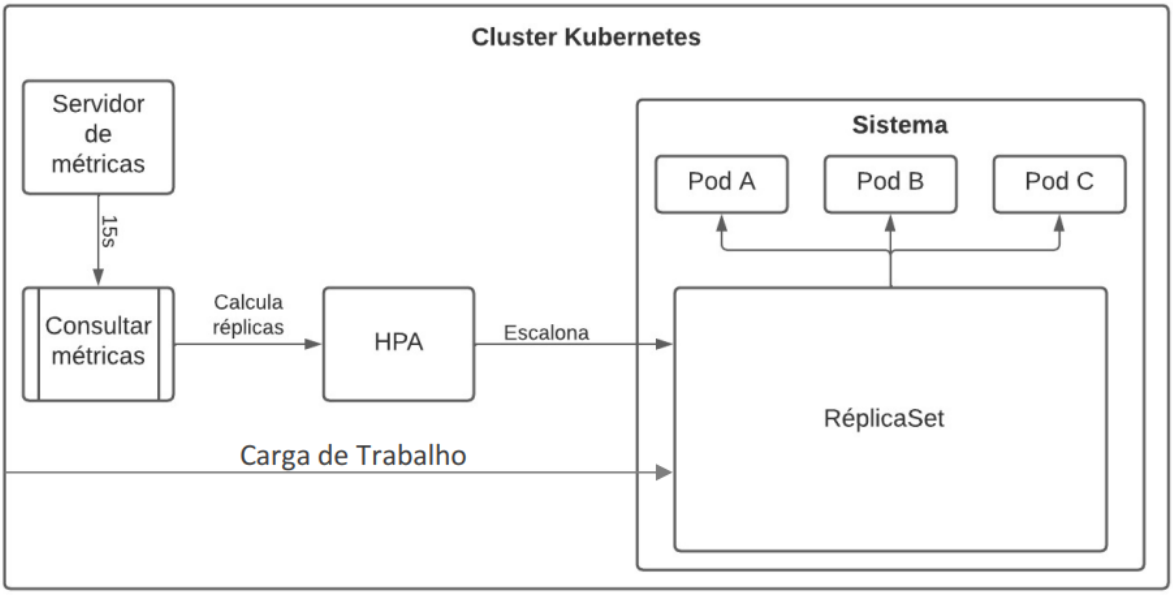
Para viabilizar tal contexto, foram utilizadas as ferramentas de gerenciamento de *Clusters* de contêineres Kubernetes: Minikube e K3S. Ambos tiveram suas configurações feitas de forma semelhante para a disponibilização do serviço, neste caso um servidor leve de HTTP (*Hypertext Transfer Protocol*) para receber as requisições e realizar as respostas.

Com a geração de demanda por processamento, neste cenário foram mensuradas a utilização de CPU, consumo de memória, consumo de disco e tempo de resposta. Essas métricas foram bem definidas com o claro objetivo de investigar a capacidade de processamento e resposta às requisições recebidas em cada um dos ambientes, tanto no Minikube como no K3S, possibilitando dessa forma também vislumbrar a ocorrência dos efeitos do envelhecimento de *software*.

O *autoscaling* foi o componente escolhido para essa avaliação neste trabalho por ser ele um dos principais recursos do Kubernetes, pelo fato de possibilitar a automatização de diversas tarefas de provisionamento, dimensionamento, como também de gerenciamento de *Clusters*, permitindo a criação de procedimentos automatizados sem ter que gastar tempo ao fazer alocações de recursos de forma manual.

Essa automatização possibilita respostas mais ágeis às altas demandas e também reduz custos dos recursos quando estes não são mais necessários em casos de baixa demanda. O processo de *autoscaling* como pode ser observado na Figura 12, se dá através do *Horizontal Pod Autoscaler* (HPA) que é acionado quando existe uma necessidade de criar ou remover réplicas de um *Pod*.

Figura 12 – Processo do *autoscaling* do Kubernetes



Fonte: O Autor

O HPA gerencia automaticamente o dimensionamento da carga de trabalho recebida, que executa um *loop* de controle por um período de 15 segundos através de um sinalizador fornecido pelo gerenciador do controlador, que por sua vez, verifica o uso dos recursos com as métricas definidas, podendo ser útil para sistemas com e sem estados.

Com isso, a partir do processo de estresse através de altas cargas de requisições no cenário proposto na execução do Minikube e K3S para gerenciar o *Cluster* Kubernetes em ambos, tendo o procedimento de *autoscaling* como um componente fundamental na manutenibilidade do sistema como um todo. Para manter a sua disponibilidade e confiabilidade, o *Cluster* Kubernetes passou por procedimentos de rejuvenescimento de *software*, tentando conter qualquer sinal de envelhecimento observado.

5.2 De que forma foram realizados os experimentos?

Os ambientes do Minikube e K3S utilizados neste trabalho são representados tanto na Figura 13 como na Figura 14. Na Figura 14, pode ser observado que cada um deles possui 5 *Pods* e 1 *Service* possibilitando a comunicação interna entre os mesmos, como também a externa ao ambiente. Um deles, foi configurado com *Deployment* de um servidor Web Nginx.

Isso permitiu a realização de testes de desempenho de uma aplicação armazenada no *Cluster* Kubernetes através do Minikube e K3S, ao ter a simulação de respostas as requisições de usuários de qualquer lugar a partir da conexão com a *Internet*, representando dessa forma um contexto de uma aplicação de qualquer serviço hospedado também na *Internet*.

A partir disso, os objetivos deste trabalho se deram com base no modelo proposto por (BASILI; CALDIERA; ROMBACH, 1994), pela utilização do método *Goal Question Metric* (GQM) para se obter respostas as questões a partir dos objetivos definidos parar verificar a ocorrência dos primeiros sinais dos efeitos do envelhecimento de *software* com base nas métricas monitoradas, como também o desempenho do *Cluster* Kubernertes no Minikube e K3S por meio das respostas do serviço as requisições recebidas.

Logo, baseado nas métricas utilizadas por (FERREIRA; SINNOTT, 2019a) e (XAVIER et al., 2013), este trabalho também mensurou métricas como consumo de memória, utilização de CPU, utilização do disco, como também o tempo de respostas as requisições feitas por um usuário externo ao *Cluster*. Com isso, os resultados das medições de cada métrica avaliada foram obtidos por *scripts* desenvolvidos para o monitoramento das mesmas com o propósito de terem seus comportamentos analisados.

No experimento, as variáveis independentes foram as quantidades da replicação de *Pods* simultaneamente as solicitações recebidas ao serviço oferecido, sobrecarregando o servidor *Web* Nginx que dessa maneira faz a emulação do componente de *autoscaling* do *Cluster* Kubernetes, para tentar fazer com o que servidor da aplicação permaneça disponível mesmo sob altas cargas de trabalho, mantendo a sua confiabilidade.

Por outro lado, foram consideradas variáveis dependentes como a utilização de CPU, consumo de memória, utilização do disco e o tempo de respostas as requisições. E com isso, os objetivos contidos na seção 1.2 a serem alcançados foram elaborados seguindo o método GQM visando a plena cobertura do tema proposto neste trabalho.

Para a execução completa do experimento, as etapas listadas abaixo foram desenvolvidas:

1. Levantamento e avaliação dos requisitos para sua realização no ambiente Minikube e K3S;
2. Desenvolvimento de *scripts* de monitoramento, execução do ambiente e suas altas cargas de trabalho e análises dos mesmos;

3. O *script* de execução do experimento em ambos os ambientes seguiram o seguinte o roteiro:

   a O monitoramento do ambiente é executado 2 horas mesmo sem ter o *Cluster* inicializado como também sem qualquer carga de trabalho;

   b O *Cluster* do Kubernetes é inciado após 2 horas da execução do monitoramento, seguindo ainda sem qualquer tipo de carga;

   c Executa após as 2 horas de iniciado o *Cluster* Kubernetes, a alta carga de trabalho por 420 vezes em um *loop*, emulando então o *autoscaling*;

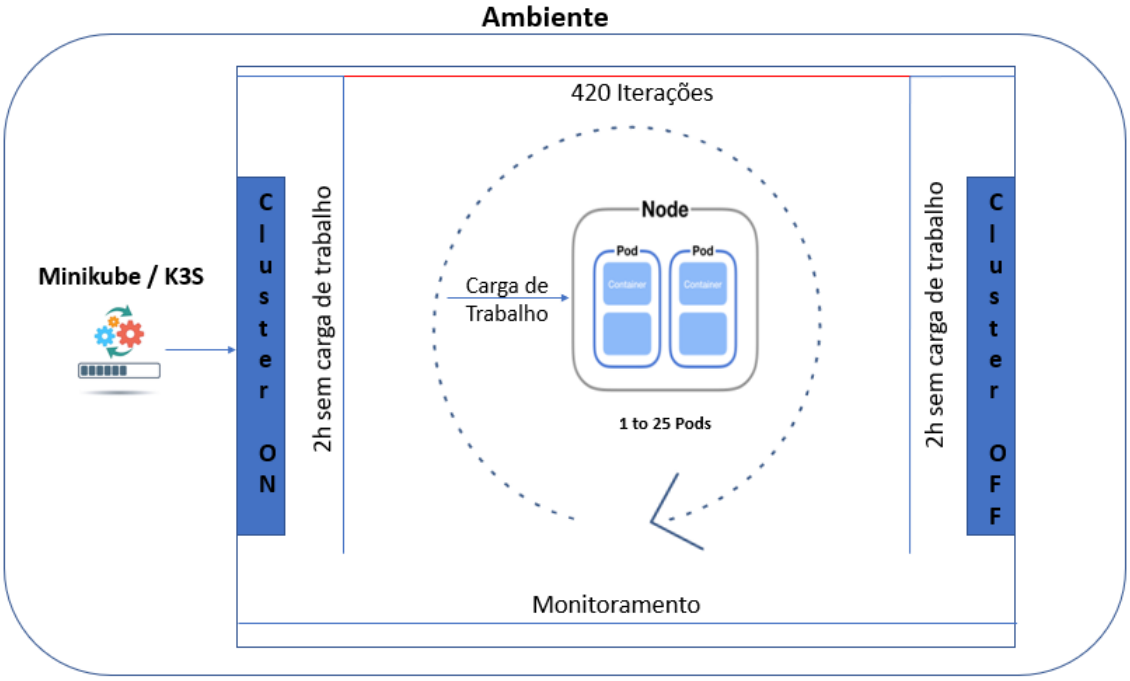
   d Depois da execução da alta carga de trabalho, então é mantido novamente o ambiente sem qualquer carga de trabalho por mais 2 horas para então encerrar o *Cluster* como uma aplicabilidade de ação de rejuvenescimento de *software*, sendo assim uma abordagem baseada no tempo;

   e Repete-se então as 4 etapas anteriores até completar o ciclo de 5 execuções;

4. Analisar e gerar os gráficos dos resultados obtidos.

A Figura 13 apresenta um diagrama representando a sequência das operações realizadas pelo *script* de execução do experimento, descrito acima, para reforçar o entendimento do experimento.

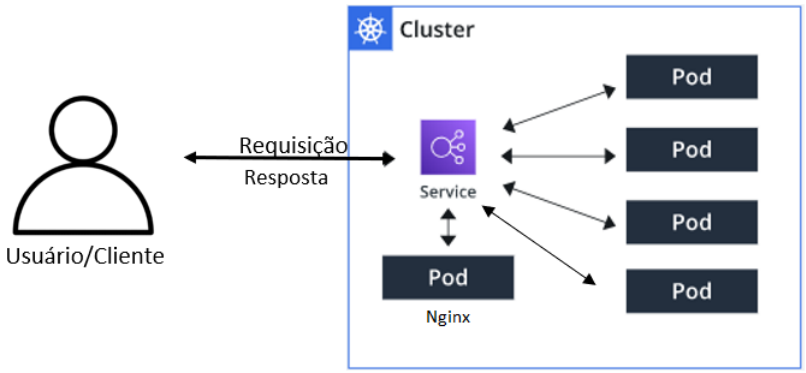
Figura 13 – Diagrama dos ciclos do experimento.



Fonte: O Autor

Durante a execução das etapas 3a, 3b, 3c e 3d outro *script* de envio de requisições ao servidor de aplicação *Web* Nginx que está alocado no *cluster* Kubernetes é executado, sendo que tais requisições podem ser respondida por quaisquer dos *Pods* criados durante o procedimento de sua replicação ou remoção enquanto processa a alta carga de trabalho.

Figura 14 – Cenário de interação entre cliente e servidor.



Fonte: O Autor

A interação entre cliente e o serviço alocado no *Cluster* Kubernetes, pode ser observada na Figura 14, representando o cenário tanto no ambiente Minikube quanto no K3S, onde em ambos, a arquitetura é configurada como na Figura 14, com um grupo lógico de *Pods*.

Além dos *Pods*, o cenário contou também com uma abstração do *Service* que é definido, possibilitando a exposição externa do serviço gerando tráfego de pacotes, balanceamento de carga e descoberta de serviços para esses *Pods*, que possuem como serviço de *Web* Nginx. Esse cenário do experimento foi realizado utilizando a seguinte configuração de *hardware*:

- Um computador com 8GB de RAM;
- Processador Core i3 com um *clock* de 3,1 GHz;
- Um módulo Wi-Fi;
- SO Linux Ubuntu de Versão 20.04 64 bits.

A configuração na parte de tecnologia de *softwares* foram utilizadas:

- *Shell script*, para a implementação das rotinas tanto do experimento, como também de monitoramento e obtenção dos dados resultantes do experimento executados através do interpretador de comando *bash*;
- *GNU bash* versão 5.0.17(1): *release* (x86_64pc-linux-gnu)
- Nginx versão *stable*

- Contêiner ID: docker://9ac4a6bb3a7c550e4e0fce6f3a75d8bf19919106101cb107b1170cfb8f5c7339
- Minikube de versão 1.15.1 e commit: 3f40a012abb52eff365ff99a709501a61ac5876;
- k3s de versão v1.22.5+k3s1;
- Kubernetes de versão v1.19.4 no Docker 19.03.13 para a execução do *cluster* do Kubernetes.

As métricas de utilização de CPU e consumo de memória foram coletadas cada uma com um intervalo de 60 segundos, enquanto a métrica de uso de disco foi coletada com um intervalo de 5 segundos, intervalos entendidos como necessários para ter amostras satisfatórias a partir das primeiras amostras obtidas de acordo com os primeiros resultados dos experimentos iniciais. Todos os *scripts* desenvolvidos durante este trabalho, encontram-se disponíveis no Anexo A através de Pseudocódigo.

6 Avaliação Experimental

Este capítulo apresenta os detalhes dos resultados obtidos dos experimentos realizados, de maneira a demonstrar o comportamento do *Cluster* Kubernetes no ambiente do Minikube e K3S, como também o desempenho individual de cada um deles a partir das métricas de tempo de reinicialização de cada *Pod*, do tempo de estimativa do RTT (*Round Trip Time*) do serviço Nginx obtido com a utilização do *ping*, de utilização de CPU, consumo de memória e por fim, do uso de disco.

6.1 Resultados

É importante destacar que o tempo total do experimento realizado no Minikube difere do tempo do experimento realizado no ambiente do K3S, apesar de executar as mesmas rotinas programadas nos algoritmos, devido ao diferente tempo total gasto para reinicializar o *Pod* durante o processo de *autoscaling*.

Tabela 6 – Tempo médio de reinicialização do *Pod*

Ambiente	**Tempo (s)**
K3S	72,80
Minikube	97,56

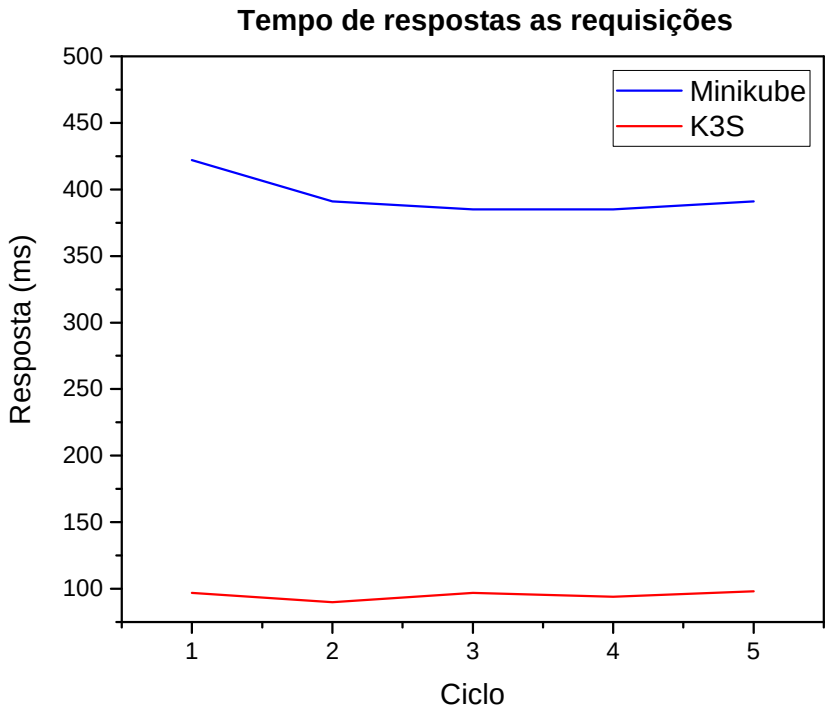
Fonte: O Autor

A Tabela 6 apresenta o tempo médio em segundos de cada ambiente para realizar a reinicialização do *Pod*, evidenciando a execução mais rápida no ambiente do K3S do que o Minikube, com diferença de 25,4%, demonstrando maior eficiência no processo de *autoscaling* do K3S em relação ao Minikube.

Na Figura 15 é apresentada a mediana do tempo de estimativa do RTT (*Round Trip Time*)

às requisições recebidas pelo serviço do Nginx tanto no ambiente Minikube como no ambiente K3S, representando o tempo de estimativa do RTT na metade de cada ciclo onde era exercida alta carga de trabalho no *Cluster* do Kubernetes.

Figura 15 – Tempo de estimativa do RTT.



Fonte: O Autor

Durante a execução de todo o experimento, a métrica foi monitorada através da ferramenta *ping* que utiliza o protocolo ICMP (*Internet Control Message Protocol*) para atestar a conexão entre dispositivos, que disparava requisições a cada 60 segundos. Dessa forma, como apresentado na Figura 15 , percebe-se a vantagem no desempenho na estimativa do RTT no ambiente K3S.

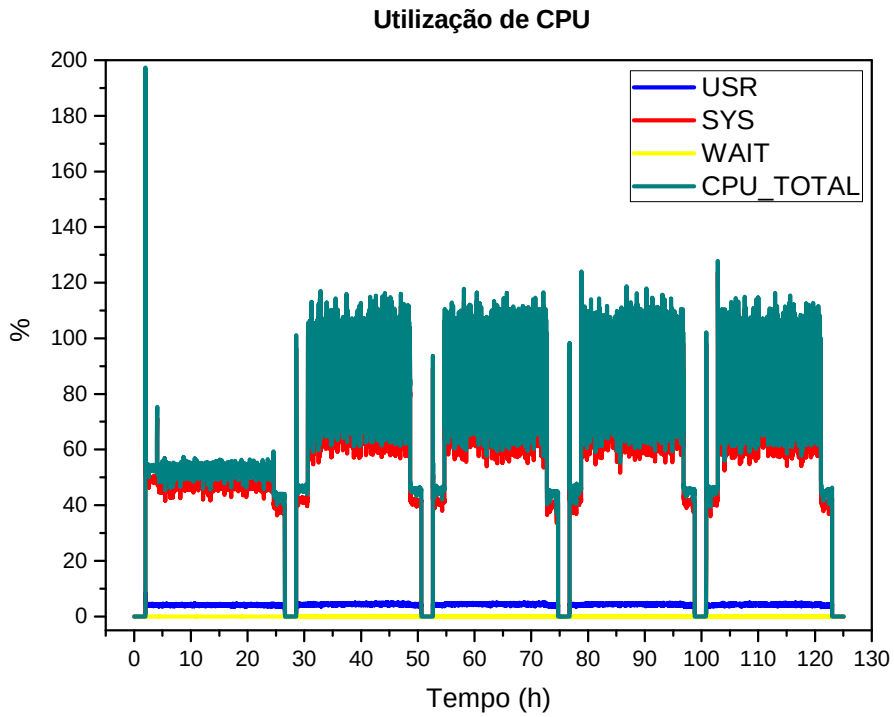
O *Cluster* Kubernetes no ambiente do Minikube, mantém ao longo da execução do experimento um tempo de estimativa do RTT sempre acima dos 350 ms, enquanto no ambiente do K3S, o tempo de de estimativa do RTT se manteve próximo a marca dos 100 ms, uma redução de 72% em relação ao do Minikube.

6.1.1 Utilização de CPU

Partindo desse entendimento, pode-se então apresentar os resultados coletados para a métrica de utilização de CPU através da ferramenta Pidstat, que monitora as tarefas de modo individual que são gerenciadas pelo *kernel* do SO Linux, disponibilizando relatórios estatísticos das tarefas, neste caso a do experimento, levando em consideração os parâmetros USR, SYS, *WAIT* e CPU_TOTAL.

O parâmetro USR afere a porcentagem do tempo de CPU utilizada pela tarefa durante a execução em nível do usuário. O parâmetro SYS por sua vez, faz a medição da porcentagem do tempo de CPU usada durante a execução da tarefa no nível do *kernel* do SO, enquanto que o parâmetro *WAIT* mede a porcentagem do tempo de CPU que a tarefa gastou esperando para ser executada, isto é, em filas de I/O ou aguardando outros processos liberarem a CPU. Por fim, o parâmetro CPU_TOTAL é a medida da porcentagem total do tempo da CPU utilizada pela tarefa.

Figura 16 – Utilização de CPU no ambiente Minikube.



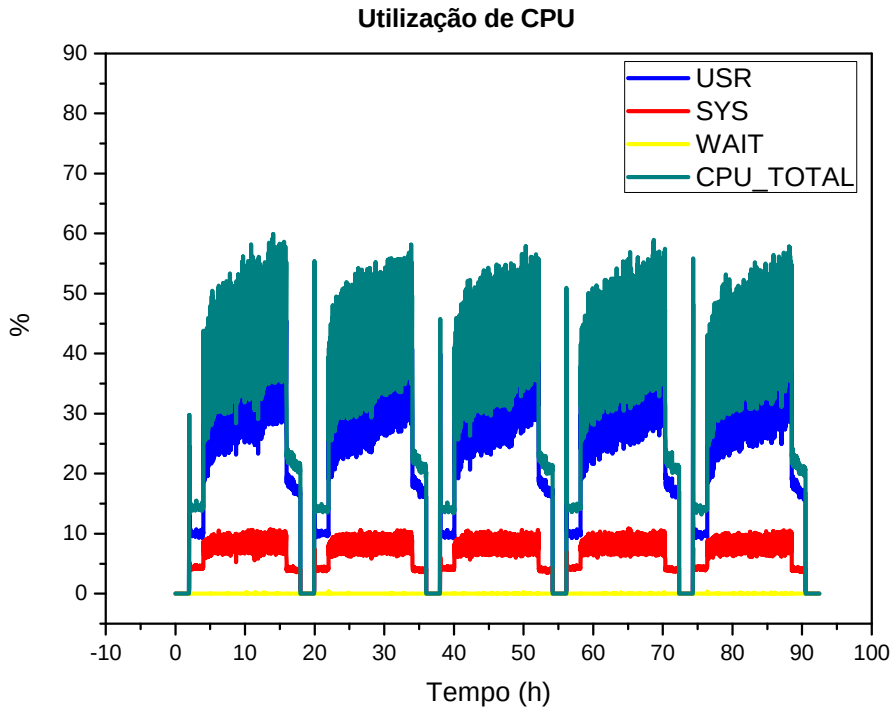
Fonte: O Autor

Na Figura 16 percebe-se um pico de consumo da utilização de CPU de 180% de CPU_TOTAL na inicialização do *Cluster*, mantendo entretanto uma média um pouco acima de 100% durante todo o processo de experimento no ambiente do Minikube.

Também é possível perceber, através da Figura 16, um ambiente controlado sob os limites na métrica avaliada, devido a sua extrapolação exclusivamente ao iniciar o seu ambiente. É importante também notar, que neste contexto, os valores acima de 100% estão diretamente relacionados com a utilização de mais de um núcleo do processador pelo processo específico da tarefa monitorada.

Na Figura 17 é possível observar um comportamento diferente do K3S com relação ao Minikube na utilização da CPU quando o parâmetro de CPU_TOTAL é analisado, que ao contrário do Minikube aponta um aumento na utilização da CPU_TOTAL juntamente com o parâmetro USR com o passar do tempo. Este comportamento só é interrompido após a

Figura 17 – Utilização de CPU no ambiente K3S.



Fonte: O Autor

reinicialização do *Cluster*, que funciona como uma ação de rejuvenescimento de *software* baseada no tempo para esta situação, mesmo que durante todo o experimento, o parâmetro CPU_TOTAL não tenha ultrapassado 60% de utilização.

6.1.2 Uso de disco

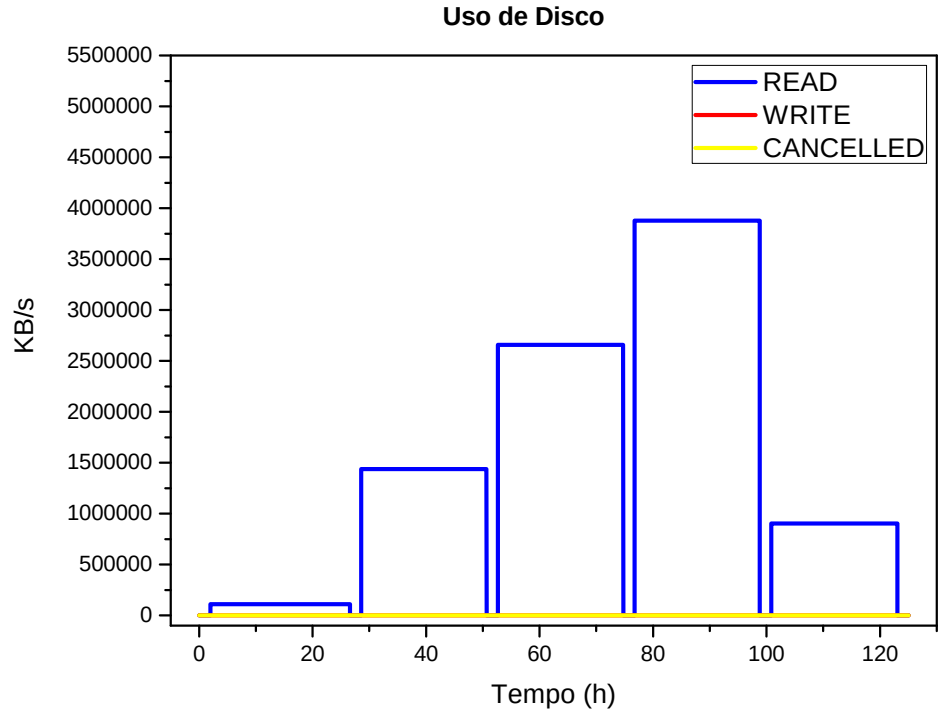
Para a avaliação da métrica uso de disco, foram coletados os parâmetros relacionados como *READ*, *WRITE* e *CANCELLED*. O parâmetro *READ*, representa a quantidade de kilobytes por segundos que a tarefa solicitou para leitura. Já o parâmetro *WRITE*, afere a quantidade de *kilobytes* por segundo que a tarefa enviou para escrita no disco.

Por fim, o parâmetro *CANCELLED* mede a quantidade de kilobytes por segundo cuja a gravação no disco tenha sido cancelada pela tarefa, que pode ocorrer possivelmente quando a tarefa encontra cache de página sujo. Os parâmetros foram coletados do ambiente do Minikube e do K3S, utilizando a ferramenta Pidstat de monitoramento.

A Figura 18 mostra um comportamento estável e inalterado para as métricas *WRITE* e *CANCELLED* durante todo o tempo de execução do experimento, sempre estando muito próximo de 0 KB/s. O comportamento do parâmetro *READ* diverge em relação aos outros, ao manter o mesmo valor ao longo de cada ciclo da alta carga de trabalho executada no *Cluster*, apresentando

um crescimento linear de um ciclo para outro, até o quarto ciclo de execução.

Figura 18 – Uso de Disco no ambiente Minikube.



Fonte: O Autor

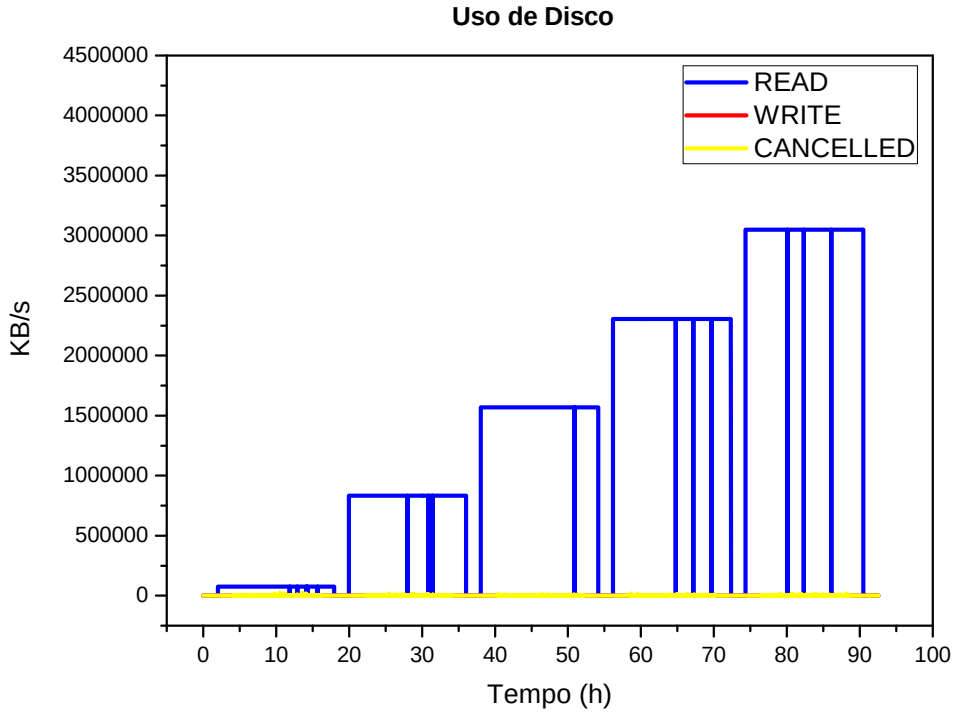
Como pode ser observado na Figura 18, há uma interrupção de forma abrupta ao atingir a marca de 4.000.000 KB/s, por ser o valor limitante pelo ambiente do Minikube. Esse tipo de comportamento pode ser plenamente visto como um indicativo de alta relevância na observância dos sinais dos efeitos de envelhecimento de *software* presentes neste ambiente.

Na Figura 19 são apresentados os resultados obtidos da métrica de utilização de disco no ambiente K3S, evidenciando que os parâmetros *WRITE* E *CANCELLED* assim como no ambiente Minikube permanecem muito próximos de 0 KB/s durante a execução de todo o experimento.

Entretanto, o parâmetro *READ* neste cenário diferente do exposto no ambiente do Minikube, não apresentou uma redução de forma abrupta após o quarto ciclo, apesar de apresentar um comportamento de crescimento linear a cada novo ciclo, do início até o fim do experimento executado no ambiente K3S. Também pode-se observar quedas do parâmetro *READ* dentro de cada ciclo, porém seguidas imediatamente por retomadas aos níveis anteriores.

Apesar disso, é importante salientar que os valores lidos de *bytes* por segundo apresentados no ambiente do K3S são menores quando comparados aos valores retornados do ambiente Minikube, o que pode ter impedido dessa maneira a existência de uma possível interrupção de forma abrupta. O que não impede totalmente a ocorrência de um efeito mais severo do

Figura 19 – Uso de Disco no ambiente K3S.



Fonte: O Autor

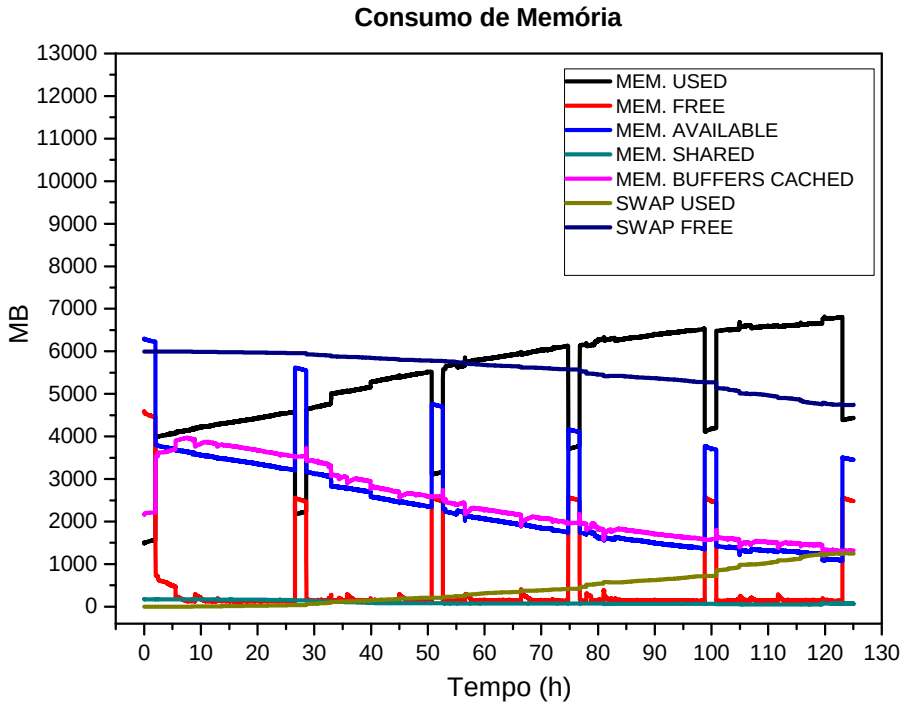
envelhecimento de *software*, uma vez que é plausível supormos que caso o experimento tivesse sido executado por mais tempo, a interrupção pudesse ocorrer assim que se atingisse valores limítrofes do parâmetro para este ambiente.

6.1.3 Consumo de Memória

Na avaliação dos resultados obtidos das métricas de consumo de memória, foi utilizada a ferramenta de monitoramento *free* do próprio SO Linux para medir o parâmetro MEM_USED que representa o cálculo da memória total utilizada, o parâmetro MEM_FREE que implica na quantidade de memória não utilizada, o parâmetro MEM_AVAILABLE que por sua vez estipula o quanto de memória está disponível para possibilitar a inicialização de uma nova aplicação sem trocas de páginas de/para o disco, isto é, possibilidade de incluir o espaço de memória que está sendo utilizado para cache ou *buffers*.

Além destes, também foi coletado o parâmetro de MEM_SHARED que é a memória usada pelo TMPFs (temporary filesystem) - sistema de arquivos que mantém os arquivos na memória virtual. O parâmetro MEM_BUFFERS_CACHED que é a soma da memória *cache* e *buffers*. Por fim, os parâmetros SWAP_FREE e SWAP_USED que representam respectivamente a quantidade livre e usada da memória virtual no espaço de trocas, que permite a utilização do disco rígido por parte do sistema, como uma memória física.

Figura 20 – Consumo de memória no ambiente Minikube.



Fonte: O Autor

A Figura 20 apresenta os resultados obtidos da métrica consumo de memória no ambiente Minikube, onde mostra que o parâmetro MEM_USED tem um comportamento espelhado ao do parâmetro MEM_AVAILABLE, pois enquanto que o parâmetro MEM_USED aumenta com o passar do tempo durante o experimento, o parâmetro MEM_AVAILABLE por sua vez diminui proporcionalmente.

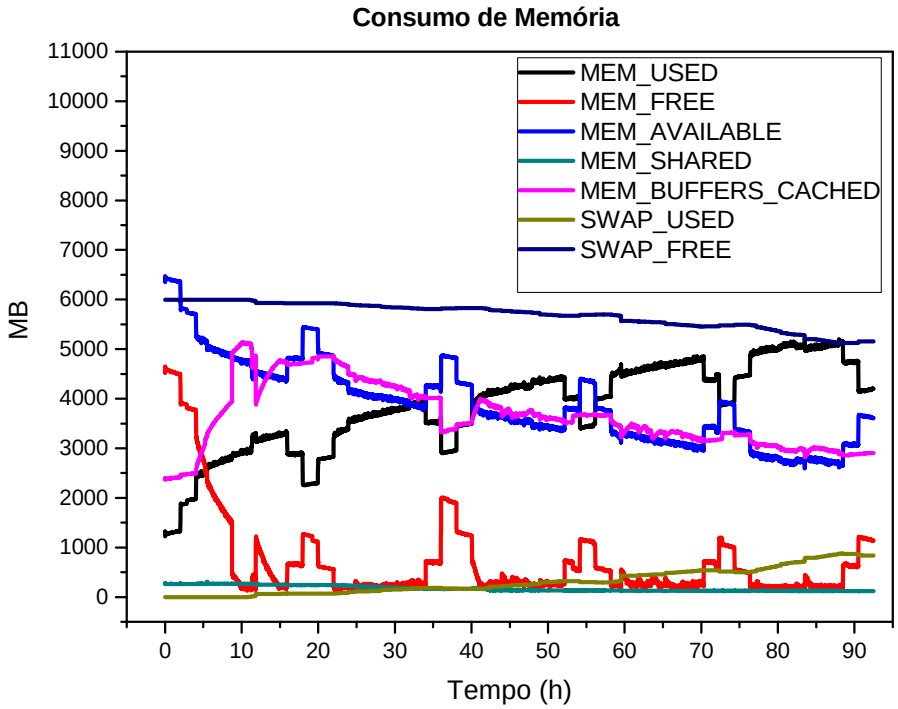
O parâmetro MEM_USED tem um aumento de consumo em torno de 70% no término do experimento mesmo após a aplicação da ação de rejuvenescimento de *software* ao encerrar o *Cluster* ao fim de cada ciclo de carga e reiniciá-lo de forma programada. Nota-se que essa ação diminui o uso da memória de forma temporária, mas quando o sistema é iniciado novamente volta a ter um consumo de memória no mesmo nível imediatamente anterior à de quando passou pela ação de rejuvenescimento.

Por sua vez, o parâmetro MEM_FREE teve uma queda em termos percentuais em torno de 48% e seguindo também essa linha, o parâmetro MEM_BUFFERS_CACHED também apresenta uma queda de aproximadamente de 41%. De forma semelhante aos parâmetros MEM_USED e MEM_AVAILABLE, o parâmetro SWAP_USED se comportou de forma inversa ao SWAP_FREE.

O SWAP_USED teve um aumento de 20% ao fim do experimento enquanto o SWAP_FREE tem uma queda de 11% e por fim, o parâmetro MEM_SHARED tanto no ambiente Minikube e

K3S teve um comportamento semelhante, tendo uma regularidade no seu consumo entre 40 a 179 MB.

Figura 21 – Consumo de memória no ambiente K3S.



Fonte: O Autor

Analisando a métrica de consumo de memória no ambiente K3S apresentado na Figura 21, que apresenta o comportamento do parâmetro MEM_USED similar ao obtido no ambiente do Minikube. O MEM_USED teve um aumento no consumo de 61% ao fim do experimento executado, mesmo sob ação do rejuvenescimento de *software* aplicada como no Minikube.

O parâmetro MEM_FREE apresentou uma queda de aproximadamente 79%, diferente do parâmetro MEM_BUFFERS_CACHED que teve um aumento de consumo em torno de 12% ao fim do experimento, mas que aumentou seu consumo nas primeiras 10 horas de execução e logo após apresentou uma queda até chegar esse patamar no final do experimento. Já o parâmetro SWAP_USED teve um aumento de 8%, enquanto o parâmetro SWAP_FREE reduziu o seu consumo em 8,5%.

Tanto no ambiente Minikube como no K3S foram desenvolvidos para os parâmetros MEM_USED e SWAP_USED avaliados, cálculos de regressão linear a fim de estimar em que momento o sistema poderia atingir o seu limite superior no consumo de memória RAM, que neste caso é de 8 GB para o parâmetro MEM_USED e de 5.8 GB para o SWAP_USED. Dessa forma, é possível ter um indicativo de quando pode ocorrer o esgotamento destes recursos no sistema, que por sua vez pode implicar na interrupção inesperada no fornecimento do serviço.

$$MU_{Minikube} = 3900,84 + 23,98072 \times T_{stress} \tag{6.1}$$

Na Equação 6.1 de regressão linear, temos que a MU_Minikube é a quantidade de memória, em KB, usada pelo Minikube, e T_stress é o tempo, em segundos, decorrido desde o início da carga de *stress* no sistema. A mesma pode ser a resultante para o parâmetro MEM_USED no ambiente Minikube, apresentado na Figura 20, que possibilita observar essa equação como uma função de y, onde o limite superior de 8 GB é alcançado depois de 170 horas, aproximadamente 7 dias e 2 horas, de execução da carga de trabalho utilizada de forma contínua no experimento.

$$SU_{Minikube} = -221,43413 + 10,37255 \times T_{stress} \tag{6.2}$$

Na Equação 6.2 temos que SU_Minikube, por sua vez, também é a quantidade de memória em KB, usada pelo Minikube, tendo o T_stress em segundos, para o parâmetro de SWAP_USED. Nessa equação é possível observar que o limite superior de 5.8 GB é alcançado depois de 551 horas, aproximadamente 22 dias, de execução da carga de trabalho utilizada no experimento, tenso assim o seu recurso esgotado.

$$MU_{K3S} = 2482,70 + 29,67105 \times T_{stress} \tag{6.3}$$

A Equação 6.3 se refere ao parâmetro de MEM_USED para o ambiente do K3S, tendo sua memória em KB onde é possível vislumbrar no ambiente do K3S que tal parâmetro, com o limite superior de 8 GB, tem o esgotamento deste recurso no decorrer de 187 horas ou no sétimo dia de execução da carga de trabalho no *Cluster* do Kubernetes que é apresentado na Figura 21, tempo maior que o apresentado pelo mesmo parâmetro no ambiente do Minikube.

$$SU_{K3S} = -120,24857 + 9,30782 \times T_{stress} \tag{6.4}$$

Já a Equação 6.4 se trata do parâmetro de SWAP_USED também no ambiente K3S, com o limite superior de 5.8 GB, possibilita observar o esgotamento de tal recurso com o passar de 603 horas, aproximadamente 25 dias, de execução da carga de trabalho no *Cluster* do Kubernetes apresentado na Figura 21, levando mais tempo para ser esgotado do que no ambiente Minikube.

É importante mencionar que essas equações são dependentes do cenário de carga aqui utilizado, e portanto em ambientes com qualquer outra característica de carga, novas equações deveriam ser obtidas, para somente assim estimar com precisão o tempo no qual o limite do uso de memoria seria alcançado.

Vale ressaltar que os dados coletados de cada resultado tanto do ambiente Minikube quanto do K3S estão disponibilizados para consulta no link <https://bit.ly/3SXY0I8>. Essa disponibilização de forma separada se deu, por conta de que os dados coletados são muito extensos para estarem descritos aqui.

O gerenciamento e orquestração de contêineres têm ganhado cada vez mais relevância e destaque no contexto em que os contêineres têm sido uma boa opção tecnológica às VMs, por ter melhor eficiência e flexibilidade no gerenciamento dos seus recursos, como também a facilidade na implantação de serviços.

Em consequência, há uma necessidade também por soluções no âmbito de garantia na manutenibilidade do serviço ofertado nos *Clusters* de contêineres, através de automatização de rotinas como gestão das aplicações, segurança, implantação, redimensionamento e escalonamento dos recursos, de modo a manter sua confiabilidade e disponibilidade.

Diante disso, diminuir as possibilidades de uma possível interrupção de maneira inesperada é fundamental neste contexto. Pensando nisso, este trabalho investigou uma possível existência dos sinais de envelhecimento de *software* no *Cluster* Kubernetes, tema que tem ganhado atenção e por se manifestar de forma silenciosa, é imprescindível a sua observância.

7

Conclusão

Este trabalho analisou possíveis efeitos de envelhecimento de *software* no processo de *autoscaling* de *Clusters* Kubernetes, executados por meio dos ambientes Minikube e K3S. Para isso, as métricas de tempo de reinicialização de cada *Pod*, tempo de estimativa do RTT às requisições recebidas, utilização de CPU, uso de disco e consumo de memória foram monitoradas no experimento para serem analisadas e discutidas.

A partir disso, quando observa-se a Tabela 6, é apresentado um desempenho melhor do ambiente K3S quando se avalia o tempo médio de reinicialização do *Pod* quando comparado ao mesmo comportamento no ambiente do Minikube, ao fazer o processo de *autoscaling* dos recursos. Além disso, neste mesmo contexto, o RTT no ambiente do K3S teve melhor desempenho em relação ao Minikube como mostra a Figura 15.

Os resultados apresentados nas Figuras 16 e 17 indicam que a maior parte da utilização de CPU acontece com código em nível de usuário no ambiente do K3S, diferente do ambiente Minikube que tem a maior utilização em nível de *kernel*, portanto com instruções privilegiadas de acesso ao *hardware*. Este comportamento de consumo crescente no parâmetro USR no ambiente K3S, ocorreu mesmo após a aplicação de uma ação de rejuvenescimento de *software* periódica, que seria o encerramento e posterior inicialização do *Cluster*. No ambiente Minikube, o parâmetro SYS se manteve estável durante todo o experimento, uma estabilidade que destoa do comportamento apresentado no ambiente do K3S.

Nas Figuras 18 e 19 são apresentados os resultados, que evidenciam semelhante comportamento da métrica uso de disco no ambiente K3S e Minikube, diferindo apenas o parâmetro *READ* no Minikube, que apresenta uma interrupção abrupta quando atinge 4.000.000 KB/s, retornando no quinto ciclo com uma utilização total perto de 10% do valor anterior.

No ambiente K3S, o parâmetro *READ* não apresenta qualquer interrupção de forma inesperada, porém há um comportamento de uso crescente de maneira linear, desde o primeiro

ciclo de carga de trabalho até o último executado no experimento. Vale destacar que a cada ciclo executado no experimento, o parâmetro *READ* demonstra uma necessidade de leitura maior a cada ciclo.

Comportamento semelhante é apresentado na métrica de consumo de memória no ambiente Minikube e no K3S, como pode ser notado nas Figuras 20 e 21. Nos dois ambientes, é evidenciado um crescimento linear do uso de memória RAM e swap (espaço em disco para memória virtual).

Diante disso, foi possível realizar cálculos de regressão linear para estimar o efeito de envelhecimento de *software* referente ao uso de memória do Kubernetes. Foram geradas equações que permitem saber quando um determinado valor limite de uso de memória será alcançado, a partir do tempo decorrido desde o início da carga de *stress* no sistema.

Esses sinais observados nos resultados obtidos, podem indicar as ameaças de desempenho do sistema como também a sua falha, devido os efeitos sofridos do envelhecimento de *software*. Contudo, é importante destacar que o momento em que os eventos decorrentes dos efeitos do envelhecimento poderão ocorrer dependendo das características da carga de trabalho e sua intensidade, como também das especificidades do *hardware* e sistema do *Cluster* Kubernetes em questão.

Também é válido observar, que se o sistema tiver mais recursos à sua disposição ou não receber um quantidade elevada de carga de trabalho, os efeitos de envelhecimento de *software* podem se manifestar mais lentamente, tendo como consequência uma ocorrência muito mais demorada de eventos inesperados como falhas pela exaustão de recursos.

7.1 Principais contribuições

Este trabalho demonstrou a possibilidade de ocorrência de eventos inesperados devido os efeitos do envelhecimento de *software* através de experimentos controlados emulando cenários reais, mesmo que em ambientes experimentais como Minikube e K3S através do processo de *autoscaling* do *Cluster* Kubernetes.

A pesquisa relatada, também apresentou o comportamento de desempenho do ambiente Minikube e do K3S ao aplicar cargas de trabalhos semelhantes em ambos, para validar a hipótese de envelhecimento de *software*, além de medir a eficiência de cada um ao responder requisições para o serviço disponibilizado.

Esta dissertação também contribuiu com a literatura sobre o tema envelhecimento e rejuvenescimento de *software* ao publicar um artigo científico com um mapeamento sistemático e a confecção e submissão de mais dois artigos sobre o tema envelhecimento de *software* aplicado ao *Cluster* Kubernetes, que estão aguardando aprovação.

7.2 Dificuldades e limitações

Durante o desenvolvimento deste trabalho surgiram várias dificuldades, entre elas, o dimensionamento do escopo do trabalho, haja vista a enorme gama de possibilidades a serem exploradas neste contexto, dificuldades de desenvolver de fato os algoritmos necessários para levantamento de outras métricas e todas as rotinas para execução do experimento.

Também houve dificuldades na compreensão do ambiente do Kubernetes, pelo fato de pouca clareza no funcionamento dos seus componentes, mesmo consultando a sua documentação. A falta de acessibilidade a laboratórios, como também à universidade, além de psicológicas devido ao enfrentamento de algo totalmente inesperado como uma pandemia, fazendo com que todo e qualquer contato fosse feito de modo virtual.

Além disso, houve também as limitações a nível de recursos computacionais propriamente ditos, dificultando a execução do experimento em nível de escala maior, simulando a execução de um ambiente de produção processando uma quantidade muito maior de cargas de trabalho, para deixar o experimento o mais próximo possível de um cenário real.

Até mesmo limitação de acesso a outro computador, o mais simples que fosse, de maneira que pudesse trabalhar em pesquisas, estudos conceituais e até de escrita de artigos ou dissertação paralelamente à execução do experimento. Assim, o fato de ter disponível um único computador para a realização de estudos, escritas e execução de experimento culminaram em alguns atrasos no cumprimento do cronograma.

7.3 Trabalhos futuros

Aplicação de outras abordagens de rejuvenescimento de *software*, juntamente com a avaliação de outras métricas podem ser os próximos desafios para trabalhos futuros. Outro potencial tema a ser abordado como continuidade a esta pesquisa é a investigação das falhas de programação ou projeto que levam ao crescente uso de memória e também crescente taxa de leitura do disco pelo Kubernetes.

Novas cargas de trabalho, estressando diferentes partes do sistema, além do mecanismo de *autoscaling*, também podem ser aplicadas, em busca de outros sinais de envelhecimento de *software*.

Referências

ALONSO, J.; BELANCHE, L.; AVRESKY, D. R. Predicting software anomalies using machine learning techniques. In: *2011 IEEE 10th International Symposium on Network Computing and Applications*. [S.l.: s.n.], 2011. p. 163–170. Citado na página 28.

ALONSO, J. et al. A comparative experimental study of software rejuvenation overhead. *Performance Evaluation*, v. 70, n. 3, p. 231 – 250, 2013. ISSN 0166-5316. Special Issue on Software Aging and Rejuvenation. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0166531612000922>. Citado na página 28.

ALONSO, J. et al. High-available grid services through the use of virtualized clustering. In: *2007 8th IEEE/ACM International Conference on Grid Computing*. [S.l.: s.n.], 2007. p. 34–41. Citado na página 28.

ALONSO, J.; TORRES, J.; GAVALDà, R. Predicting web server crashes: A case study in comparing prediction algorithms. In: *2009 Fifth International Conference on Autonomic and Autonomous Systems*. [S.l.: s.n.], 2009. p. 264–269. Citado na página 28.

AMARAL, M. et al. Performance evaluation of microservices architectures using containers. In: . [S.l.: s.n.], 2015. p. 27–34. Citado na página 39.

ANDRZEJAK, A.; MOSER, M.; SILVA, L. Managing performance of aging applications via synchronized replica rejuvenation. In: . [S.l.: s.n.], 2007. v. 4785, p. 98–109. Citado na página 28.

ARAUJO, J. et al. Software aging in the eucalyptus cloud computing infrastructure: Characterization and rejuvenation. *ACM Journal on Emerging Technologies in Computing Systems*, v. 10, 01 2014. Citado na página 40.

ARAUJO, J. et al. Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure. *Proceedings of the ACM/IFIP/Usenix Middleware Conference 2011 Industry Track Workshop*, 12 2011. Citado na página 27.

ARAUJO, J. et al. Software rejuvenation in eucalyptus cloud computing infrastructure: A method based on time series forecasting and multiple thresholds. In: . [S.l.: s.n.], 2011. Citado 2 vezes nas páginas 28 e 35.

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 27.

BAI, J. et al. Analyzing software rejuvenation techniques in a virtualized system: Service provider and user views. *IEEE Access*, v. 8, p. 6448–6459, 2020. Citado na página 37.

BAKER, H. G. *Memory Management*: International workshop iwmm 95. [S.l.]: Springer, 1995. Citado na página 27.

BAO, Y.; SUN, X.; TRIVEDI, K. A workload-based analysis of software aging, and rejuvenation. *Reliability, IEEE Transactions on*, v. 54, p. 541 – 548, 10 2005. Citado na página 26.

BASILI, G.; CALDIERA, V. R.; ROMBACH, H. D. The goal question metric approach. *Encyclopedia of software engineering*, p. 528–532, 1994. Citado na página 45.

BLOG, G. C. P. *An update on container support on google cloud platform*. 2014. Accessed: 2020-12-14. Citado na página 16.

BLOG, G. C. P. *An update on container support on google cloud platform*. 2014. <https://cloudplatform.googleblog.com/2014/06/an-update-on-container-support-on-google-cloud-platform.html>. Accessed: 2022-01-19. Citado na página 17.

BÖHM, S.; WIRTZ, G. Profiling lightweight container platforms: Microk8s and k3s in comparison to kubernetes. In: *ZEUS*. [S.l.: s.n.], 2021. p. 65–73. Citado na página 26.

BRILHANTE, J. et al. Dependability models for eucalyptus infrastructure clouds considering vm life-cycle. In: *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.: s.n.], 2014. p. 1336–1341. Citado na página 15.

CARROZZA, G. et al. Memory leak analysis of mission-critical middleware. *Journal of Systems and Software*, v. 83, n. 9, p. 1556 – 1567, 2010. ISSN 0164-1212. Software Dependability. Disponível em: <http://www.sciencedirect.com/science/article/pii/S016412121000110X>. Citado na página 40.

COREOS. *Rkt, a securuty-minded,standards-based container engine*. 2015. Accessed: 2020-12-19. Citado na página 39.

COSTA, J. T. da et al. Systematic mapping of literature on software aging and rejuvenation research trends. In: *2021 Annual Reliability and Maintainability Symposium (RAMS)*. [S.l.: s.n.], 2021. p. 1–6. Citado 5 vezes nas páginas 30, 32, 34, 35 e 36.

COTRONEO, D. et al. A survey of software aging and rejuvenation studies. *J. Emerg. Technol. Comput. Syst.*, Association for Computing Machinery, New York, NY, USA, v. 10, n. 1, jan. 2014. ISSN 1550-4832. Disponível em: <https://doi.org/10.1145/2539117>. Citado na página 27.

DIAMANTI. *2018 container adoption benchmark survey*. 2018. Accessed: 2020-12-14. Citado na página 16.

DIAMANTI. *2018 container adoption benchmark survey*. 2018. <https://diamanti.com/wp-content/uploads/2018/07/WP_Diamanti_End-User_Survey_072818.pdf>. Accessed: 2022-01-28. Citado na página 17.

DINESH, S. *Running MongoDB on Kubernetes with StatefulSets*. 2017. <http://blog.kubernetes.io/2017/01/running-mongodb-on-kubernetes-with-statefulsets.html>. Accessed: 2020-12-15. Citado na página 23.

DOCKER. *Concepts*. 2015. <https://www.docker.com/>. Accessed: 2020-12-15. Citado na página 23.

DOCKER. *Swarm mode overview*. 2015. <https://docs.docker.com/engine/swarm/>. Accessed: 2020-12-15. Citado na página 23.

DRUTSKOY, D.; KELLER, E.; REXFORD, J. Scalable network virtualization in software-defined networks. *IEEE Internet Computing*, v. 17, n. 2, p. 20–27, 2013. Citado na página 21.

DUA, R.; RAJA, A. R.; KAKADIA, D. Virtualization vs containerization to support paas. In: *2014 IEEE International Conference on Cloud Engineering*. [S.l.: s.n.], 2014. p. 610–614. Citado 2 vezes nas páginas 15 e 39.

FELTER, W. et al. An updated performance comparison of virtual machines and linux containers. In: *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. [S.l.: s.n.], 2015. p. 171–172. Citado na página 39.

FERREIRA, A. P.; SINNOTT, R. A performance evaluation of containers running on managed kubernetes services. In: *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. [S.l.: s.n.], 2019. p. 199–208. Citado 2 vezes nas páginas 16 e 45.

FERREIRA, A. P.; SINNOTT, R. A performance evaluation of containers running on managed kubernetes services. In: *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. [S.l.: s.n.], 2019. p. 199–208. Citado na página 16.

FORRESTER. *The forrester new waveTM: Enterprise container platform software suites, q4 2018*. 2018. <https://info.rancher.com/hubfs/eBooks,%20reports,%20and%20whitepapers/The%20Forrester%20New%20Wave%20-%20Enterprise%20Container%20Platform%20Software%20Suites,%20Q4%202018.pdf>. Accessed: 2022-01-29. Citado na página 17.

GARG, S. et al. A methodology for detection and estimation of software aging. In: *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No.98TB100257)*. [S.l.: s.n.], 1998. p. 283–292. Citado 2 vezes nas páginas 26 e 27.

GARG, S. et al. Analysis of software rejuvenation using markov regenerative stochastic petri net. In: *Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95*. [S.l.: s.n.], 1995. p. 180–187. Citado na página 26.

GOOGLE. *Containers at google*. 2021. <https://cloud.google.com/containers/>. Accessed: 2022-01-23. Citado na página 17.

GROTTKE, M.; JR, R. M.; TRIVEDI, K. The fundamentals of software aging. In: . [S.l.: s.n.], 2008. p. 1 – 6. Citado 2 vezes nas páginas 26 e 27.

GROTTKE, M. et al. Analysis of software aging in a web server. *Friedrich-Alexander-University Erlangen-Nuremberg, Chair of Statistics and Econometrics, Discussion Papers*, v. 55, 11 2005. Citado na página 26.

GUO, C. et al. Use two-level rejuvenation to combat software aging and maximize average resource performance. In: *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. [S.l.: s.n.], 2015. p. 1160–1165. Citado na página 40.

GUPTA, U. Comparison between security majors in virtual machine and linux containers. *CoRR*, abs/1507.07816, 2015. Disponível em: <http://arxiv.org/abs/1507.07816>. Citado na página 39.

HAT, R. *OpenShift Container Platform 3.7 Documentation*. 2014. <https://docs.openshift.com/container-platform/3.7/welcome/index.html>. Accessed: 14 2020-12-16. Citado na página 23.

HINDMAN, B. et al. Mesos: A platform for fine-grained resource sharing in the data center. In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. USA: USENIX Association, 2011. (NSDI'11), p. 295–308. Citado na página 23.

HUA, X. et al. Schedulability analysis for real-time task set on resource with performance degradation and dual-level periodic rejuvenations. *IEEE Transactions on Computers*, v. 66, n. 3, p. 553–559, 2017. Citado na página 38.

HUANG, Y. et al. Software rejuvenation: analysis, module and applications. In: *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*. [S.l.: s.n.], 1995. p. 381–390. Citado 2 vezes nas páginas 26 e 27.

JOY, A. M. Performance comparison between linux containers and virtual machines. In: *2015 International Conference on Advances in Computer Engineering and Applications*. [S.l.: s.n.], 2015. p. 342–346. Citado 2 vezes nas páginas 16 e 39.

K3S. *Why Use K3s*. 2021. <https://k3s.io/>. Accessed: 2021-12-21. Citado na página 26.

KHAN, A. Key characteristics of a container orchestration platform to enable a modern application. *IEEE Cloud Computing*, v. 4, n. 5, p. 42–48, 2017. Citado na página 22.

KHOSHMANESH, S.; LUTZ, R. R. The role of similarity in detecting feature interaction in software product lines. In: *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. [S.l.: s.n.], 2018. p. 286–292. Citado na página 38.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele Univ.*, v. 33, 08 2004. Citado 2 vezes nas páginas 29 e 30.

KON, J. et al. Highly consolidated servers with container-based virtualization. In: *2017 IEEE International Conference on Big Data (Big Data)*. [S.l.: s.n.], 2017. p. 2472–2479. Citado na página 20.

KOOMEY, J. Estimating regional power consumption by servers: A technical note. 01 2008. Citado na página 20.

KOZHIRBAYEV, Z.; SINNOTT, R. A performance comparison of container-based technologies for the cloud. *Future Generation Computer Systems*, v. 68, 03 2017. Citado na página 22.

KUBERNETES. *Concepts*. 2020. <https://kubernetes.io/docs/concepts/>. Accessed: 2020-12-18. Citado na página 25.

KUBERNETES. *Usando Minikube para criar um cluster*. 2020. <https://kubernetes.io/pt-br/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/>. Accessed: 2020-12-19. Citado na página 25.

KUBERNETES. *Production-grade container orchestration*. 2021. <https://kubernetes.io/>. Accessed: 2021-07-15. Citado na página 17.

LEARNITGUIDE. *What is Kubernetes – Learn Kubernetes from Basics*. 2018. <https://www.learnitguide.net/2018/08/what-is-kubernetes-learn-kubernetes.html>. Accessed: 2021-12-18. Citado na página 24.

LINK, C. et al. Container orchestration by kubernetes for rdma networking. In: *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. [S.l.: s.n.], 2019. p. 1–2. Citado na página 22.

LINTON D REKESH, B. R. J. Virtual machine migration between hypervisor virtual machines and containers[j]. 2017. Citado na página 39.

LIU, B. et al. Model-based sensitivity analysis of iaas cloud availability. *Future Generation Computer Systems*, v. 83, 01 2018. Citado na página 15.

MACHIDA, F.; KIM, D. S.; TRIVEDI, K. Modeling and analysis of software rejuvenation in a server virtualized system with live vm migration. In: . [S.l.: s.n.], 2010. v. 70, p. 1 – 6. Citado na página 39.

MACHIDA, F. et al. Aging-related bugs in cloud computing software. In: *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*. [S.l.: s.n.], 2012. p. 287–292. Citado 2 vezes nas páginas 28 e 39.

MACHIDA, F. et al. Lifetime extension of software execution subject to aging. *IEEE Transactions on Reliability*, v. 66, n. 1, p. 123–134, 2017. Citado na página 28.

MATIAS, R. et al. Accelerated degradation tests applied to software aging experiments. *IEEE Transactions on Reliability*, v. 59, n. 1, p. 102–114, 2010. Citado na página 38.

MATIAS, R.; FILHO, P. J. F. An experimental study on software aging and rejuvenation in web servers. In: *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*. [S.l.: s.n.], 2006. v. 1, p. 189–196. Citado 2 vezes nas páginas 26 e 27.

MERKEL, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, Belltown Media, Houston, TX, v. 2014, n. 239, mar. 2014. ISSN 1075-3583. Citado na página 20.

MODAK, A. et al. Techniques to secure data on cloud: Docker swarm or kubernetes? In: *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. [S.l.: s.n.], 2018. p. 7–12. Citado na página 21.

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 15.

MORABITO, R.; KJäLLMAN, J.; KOMU, M. Hypervisors vs. lightweight virtualization: A performance comparison. In: *2015 IEEE International Conference on Cloud Engineering*. [S.l.: s.n.], 2015. p. 386–393. Citado 2 vezes nas páginas 16 e 20.

MUDDINAGIRI, R.; AMBAVANE, S.; BAYAS, S. Self-hosted kubernetes: Deploying docker containers locally with minikube. In: *2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET)*. [S.l.: s.n.], 2019. p. 239–243. Citado na página 21.

MUSA, J. D. Operational profiles in software-reliability engineering. *IEEE Software*, v. 10, n. 2, p. 14–32, 1993. Citado na página 27.

NAGARAJU, V.; BASAVARAJ, V. V.; FIONDELLA, L. Software rejuvenation of a fault-tolerant server subject to correlated failure. In: *2016 Annual Reliability and Maintainability Symposium (RAMS)*. [S.l.: s.n.], 2016. p. 1–6. Citado na página 40.

OKAMURA, H.; LUO, C.; DOHI, T. Estimating response time distribution of server application in software aging phenomenon. In: *2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. [S.l.: s.n.], 2013. p. 281–284. Citado na página 37.

OPENSTACK. *OpenStack User Survey Report November 2017*. 2017. <https://www.openstack.org/assets/survey/OpenStack-User-Survey-Nov17.pdf>. Accessed: 2020-12-15. Citado na página 23.

PAHL, C. Containerization and the paas cloud. *IEEE Cloud Computing*, v. 2, n. 3, p. 24–31, 2015. Citado na página 15.

PAHL, C.; LEE, B. Containers and clusters for edge cloud architectures – a technology review. In: *2015 3rd International Conference on Future Internet of Things and Cloud*. [S.l.: s.n.], 2015. p. 379–386. Citado na página 21.

PAN, Y.; HU, N. Research on dependability of cloud computing systems. In: *2014 10th International Conference on Reliability, Maintainability and Safety (ICRMS)*. [S.l.: s.n.], 2014. p. 435–439. Citado na página 15.

PENG, J. et al. Comparison of several cloud computing platforms. In: *2009 Second International Symposium on Information Science and Engineering*. [S.l.: s.n.], 2009. p. 23–27. Citado na página 39.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*. Swindon, GBR: BCS Learning & Development Ltd., 2008. (EASE'08), p. 68–77. Citado na página 29.

PORTWORX. *2018 container adoption survey*. 2018. Accessed: 2020-12-14. Citado 2 vezes nas páginas 15 e 16.

PORTWORX. *2018 container adoption survey*. 2018. <https://portworx.com/wp-content/uploads/2018/12/Portworx-Container-Adoption-Survey-Report-2018.pdf>. Accessed: 2022-01-28. Citado na página 17.

RAHO, M. et al. Kvm, xen and docker: A performance analysis for arm based nfv and cloud computing. In: *2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. [S.l.: s.n.], 2015. p. 1–8. Citado na página 21.

REDHAT. *Red hat global customer tech outlook 2019: Automation, cloud, security lead funding priorities*. 2018. Accessed: 2020-12-14. Citado na página 15.

REINECKE, P.; WOLTER, K. A simulation study on the effectiveness of restart and rejuvenation to mitigate the effects of software ageing. In: *2010 IEEE Second International Workshop on Software Aging and Rejuvenation*. [S.l.: s.n.], 2010. p. 1–6. Citado na página 38.

RESEARCH, G. *Hype cycle for cloud computing*. 2018. <https://www.gartner.com/en/documents/3884671>. Accessed: 2022-01-28. Citado na página 17.

SHAH, J.; DUBARIA, D. Building modern clouds: Using docker, kubernetes google cloud platform. In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. [S.l.: s.n.], 2019. p. 0184–0189. Citado na página 25.

SHARMA, P. et al. Containers and virtual machines at scale: A comparative study. In: *Proceedings of the 17th International Middleware Conference*. New York, NY, USA: Association for Computing Machinery, 2016. (Middleware '16). ISBN 9781450343008. Disponível em: <https://doi.org/10.1145/2988336.2988337>. Citado na página 20.

SHERESHEVSKY, M. et al. Software aging and multifractality of memory resources. In: *2003 International Conference on Dependable Systems and Networks, 2003. Proceedings.* [S.l.: s.n.], 2003. p. 721–730. Citado na página 26.

SILVA, L.; MADEIRA, H.; SILVA, J. G. Software aging and rejuvenation in a soap-based server. In: *Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)*. [S.l.: s.n.], 2006. p. 56–65. Citado na página 26.

SOLTESZ, S. et al. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. New York, NY, USA: Association for Computing Machinery, 2007. (EuroSys '07), p. 275–287. ISBN 9781595936363. Disponível em: <https://doi.org/10.1145/1272996.1273025>. Citado 3 vezes nas páginas 20, 21 e 39.

SOTOMAYOR, J. P. et al. Comparison of runtime testing tools for microservices. In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. [S.l.: s.n.], 2019. v. 2, p. 356–361. Citado na página 26.

STEINMACHER, I.; CHAVES, A. P.; GEROSA, M. A. Awareness support in distributed software development: A systematic review and mapping of the literature. *Comput. Supported Coop. Work*, Kluwer Academic Publishers, USA, v. 22, n. 2–3, p. 113–158, apr 2013. ISSN 0925-9724. Disponível em: <https://doi.org/10.1007/s10606-012-9164-4>. Citado na página 30.

SUCHITRA. *Tecnologias de orquestração de nuvem: explore suas opções*. 2016. <https://developer.ibm.com/br/depmodels/cloud/articles/cl-cloud-orchestration-technologies-trs/>. Accessed: 2020-12-15. Citado na página 23.

SUKHWANI, H. et al. Monitoring and mitigating software aging on ibm cloud controller system. In: *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. [S.l.: s.n.], 2017. p. 266–272. Citado na página 38.

SYSTEMS, I. C. *Cisco security advisory: Cisco Catalyst memory leak vulnerability*. 2001. <http://www.cisco.com/warp/public/707/cisco-sa-20001206-catalystmemleak.shtml>. Accessed: 2020-12-16. Citado na página 26.

TACHIBANA, Y.; KON, J.; YAMAGUCHI, S. A study on the performance of web applications based on ror in a highly consolidated server with container-based virtualization. In: *2017 Fifth International Symposium on Computing and Networking (CANDAR)*. [S.l.: s.n.], 2017. p. 580–583. Citado na página 20.

TORQUATO, M. et al. An approach to investigate aging symptoms and rejuvenation effectiveness on software systems. In: *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.: s.n.], 2017. p. 1–6. Citado na página 38.

TORQUATO, M.; MACIEL, P.; VIEIRA, M. A model for availability and security risk evaluation for systems with vmm rejuvenation enabled by vm migration scheduling. *IEEE Access*, v. 7, p. 138315–138326, 2019. Citado na página 37.

TORQUATO, M.; VIEIRA, M. Interacting srn models for availability evaluation of vm migration as rejuvenation on a system under varying workload. In: *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. [S.l.: s.n.], 2018. p. 300–307. Citado na página 39.

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 na página 37.

TOSATTO, A.; RUIU, P.; ATTANASIO, A. Container-based orchestration in cloud: State of the art and challenges. In: *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*. [S.l.: s.n.], 2015. p. 70–75. Citado 2 vezes nas páginas 22 e 24.

TRUYEN, E. et al. Evaluation of container orchestration systems for deploying and managing nosql database clusters. In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. [S.l.: s.n.], 2018. p. 468–475. Citado na página 23.

VAIDYANATHAN, K.; TRIVEDI, K. S. A comprehensive model for software rejuvenation. *IEEE Trans. Dependable Secur. Comput.*, IEEE Computer Society Press, Washington, DC, USA, v. 2, n. 2, p. 124–137, abr. 2005. ISSN 1545-5971. Disponível em: <https://doi.org/10.1109/TDSC.2005.15>. Citado na página 27.

VERMA, A. et al. Large-scale cluster management at Google with Borg. In: *Proceedings of the European Conference on Computer Systems (EuroSys)*. Bordeaux, France: [s.n.], 2015. Citado na página 21.

WENG, C. et al. A rejuvenation strategy in android. *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, p. 273–279, 2017. Citado na página 39.

WIGGINS, A. *The twelve-factor app*. [S.l.: s.n.], 2017. Citado na página 23.

WONG, W. K.; LEE, C. S. An implementation of face recognition with deep learning based on a container-orchestration platform. In: *2020 Indo – Taiwan 2nd International Conference on Computing, Analytics and Networks (Indo-Taiwan ICAN)*. [S.l.: s.n.], 2020. p. 325–329. Citado na página 25.

XAVIER, M. G.; NEVES, M. V.; ROSE, C. A. F. D. A performance comparison of container-based virtualization systems for mapreduce clusters. In: *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. [S.l.: s.n.], 2014. p. 299–306. Citado 2 vezes nas páginas 20 e 21.

XAVIER, M. G. et al. Performance evaluation of container-based virtualization for high performance computing environments. In: *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. [S.l.: s.n.], 2013. p. 233–240. Citado 3 vezes nas páginas 21, 22 e 45.

XIANG, J. et al. A new software rejuvenation model for android. In: *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. [S.l.: s.n.], 2018. p. 293–299. Citado na página 27.

XU, Z.; ZHANG, J. Path and context sensitive inter-procedural memory leak detection. In: *2008 The Eighth International Conference on Quality Software*. [S.l.: s.n.], 2008. p. 412–420. Citado na página 27.

YAKHCHI, M. et al. Neural network based approach for time to crash prediction to cope with software aging. *Journal of Systems Engineering and Electronics*, v. 26, n. 2, p. 407–414, 2015. Citado na página 40.

YANG, M. et al. Analysis of software rejuvenation in clustered computing system with dependency relation between nodes. In: *2010 10th IEEE International Conference on Computer and Information Technology*. [S.l.: s.n.], 2010. p. 46–53. Citado na página 38.

ZENG, H. et al. Measurement and evaluation for docker container networking. In: *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. [S.l.: s.n.], 2017. p. 105–108. Citado na página 39.

Apêndices

APÊNDICE A – Tabela de Trabalhos Relacionados

Título	Autor(es)
Analyzing Software Rejuvenation Techniques in a Virtualized System: Service Provider and User Views	BAI, J. et al., 2020
An Experimental Study of Software Aging and Rejuvenation in dockerd	TORQUATO, M.; VIEIRA, M. A, 2019
A Model for Availability and Security Risk Evaluation for Systems With VMM Rejuvenation Enabled by VM Migration Scheduling	TORQUATO, M.; MACIEL, P.; VIEIRA, M. A, 2019
Estimating response time distribution of server application in software aging phenomenon	OKAMURA, H.; LUO, C.; DOHI, T., 2013
The Role of Similarity in Detecting Feature Interaction in Software Product Lines	KHOSHMANESH, S.; LUTZ, R. R., 2018
Accelerated Degradation Tests Applied to Software Aging Experiments	MATIAS, R. et al., 2010
A simulation study on the effectiveness of restart and rejuvenation to mitigate the effects of software ageing	REINECKE, P.; WOLTER, K. A, 2010
Analysis of Software Rejuvenation in Clustered Computing System with Dependency Relation between Nodes	YANG, M. et al., 2010
Schedulability Analysis for Real-Time Task Set on Resource with Performance Degradation and Dual-Level Periodic Rejuvenations	HUA, X. et al., 2017
An approach to investigate aging symptoms and rejuvenation effectiveness on software systems	TORQUATO, M. et al., 2017
Monitoring and Mitigating Software Aging on IBM Cloud Controller System	SUKHWANI, H. et al., 2017
A Rejuvenation Strategy in Android	WENG, C. et al., 2017
Interacting SRN Models for Availability Evaluation of VM Migration as Rejuvenation on a System under Varying Workload	TORQUATO, M.; VIEIRA, M. I, 2018
Virtualization vs Containerization to Support PaaS	DUA, R.; RAJA, A. R.; KAKADIA, D., 2014

Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007	SOLTESZ, S. et al., 2007
Rkt, a securuty-minded,standards-based container engine	COREOS, 2015
Comparison of Several Cloud Computing Platforms	PENG, J. et al., 2009
Performance Evaluation of Microservices Architectures Using Containers	AMARAL, M. et al., 2015
Performance comparison between Linux containers and virtual machines	JOY, A. M., 2015
An updated performance comparison of virtual machines and Linux containers	FELTER, W. et al., 2015
Virtual machine migration between hypervisor virtual machines and containers	LINTON D REKESH, B. R. J., 2017
Comparison between security majors in virtual machine and linux containers	GUPTA, U., 2015
Measurement and Evaluation for Docker Container Networking	ZENG, H. et al., 2017
Aging-Related Bugs in Cloud Computing Software	MACHIDA, F. et al., 2012
Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration	MACHIDA, F.; KIM, D. S.; TRIVEDI, K. M, 2010
Memory leak analysis of mission-critical middleware	CARROZZA, G. et al., 2010
Software Aging in the Eucalyptus Cloud Computing Infrastructure: Characterization and Rejuvenation	ARAUJO, J. et al., 2014
Software rejuvenation of a fault-tolerant server subject to correlated failure	NAGARAJU, V.; BASAVARAJ, V. V.; FIONDELLA, L. S, 2016
Use Two-Level Rejuvenation to Combat Software Aging and Maximize Average Resource Performance	GUO, C. et al., 2015
Neural network based approach for time to crash prediction to cope with software aging	YAKHCHI, M. et al., 2015

Tabela 7 – Trabalhos Relacionados

Anexos

ANEXO A – Algoritmos

Neste Anexo, estão contidos todos os algoritmos desenvolvidos e utilizados para a realização do experimento deste trabalho.

Tabela 8 – Algoritmo de execução do Minikube

INÍCIO – SCRIPT EXPERIMENTO MINIKUBE	
1:	iniciar minikube
2:	iniciar cluster minikube
3:	iniciar script de requisições e monitoramento requisições
4:	iniciar script monitoramento disco
5:	iniciar script monitoramento memoria
6:	iniciar script monitoramento cpu
7:	contador1=0
8:	**ENQUANTO (contador1 <5)**
9:	esperar 2m
10:	iniciar 5 Pods A
11:	iniciar 1 Pod service
12:	esperar 2h
13:	iniciar 25 Pods B
14:	**PARA (contador2 = 0; contador2 <420; contador2++)**
15:	deletar 25 Pods B
16:	esperar 45s
17:	**FIM - PARA**
18:	encerrar 5 Pods A
19:	encerrar 1 Pod service
20:	deletar 25 Pods B
21:	esperar 2h
22:	parar minikube
23:	fim cluster minikube
24:	esperar 2h
25:	contador1++
26:	**FIM - ENQUANTO**
27:	esperar 2h
28:	excluir todos os pods
29:	parar minikube
FIM	

Fonte: O Autor.

Tabela 9 – Algoritmo de execução do K3S

INÍCIO – SCRIPT EXPERIMENTO K3S	
1:	iniciar k3s
2:	iniciar cluster k3s
3:	iniciar script de requisições e monitoramento requisições
4:	iniciar script monitoramento disco
5:	iniciar script monitoramento memoria
6:	iniciar script monitoramento cpu
7:	contador1=0
8:	**ENQUANTO (contador1 <5)**
9:	esperar 2m
10:	iniciar 5 Pods A
11:	iniciar 1 Pod service
12:	esperar 2h
13:	iniciar 25 Pods B
14:	**PARA (contador2 = 0; contador2 <420; contador2++)**
15:	deletar 25 Pods B
16:	esperar 45s
17:	**FIM - PARA**
18:	encerrar 5 Pods A
19:	encerrar 1 Pod service
20:	deletar 25 Pods B
21:	esperar 2h
22:	parar k3s
23:	fim cluster k3s
24:	esperar 2h
25:	contador1++
26:	**FIM - ENQUANTO**
27:	esperar 2h
28:	excluir todos os pods
29:	parar k3s
FIM	

Fonte: O Autor.

Tabela 10 – Algoritmo de monitoramento da métrica de Utilização de CPU.

```
INÍCIO – SCRIPT DE MONITORAMENTO DA CPU
1:      execucao = 0
2:      ENQUANTO
3:          cpu = mpstat a cada 60s (usr %sys %iowait %idle)
4:          imprima cpu em arquivo
5:          execucao = execucao + 60
6:          SE (execucao = 400000)
7:              entao
8:                  pare
9:          FIM - SE
10:     FIM - ENQUANTO
FIM
```

Fonte: O Autor.

Tabela 11 – Algoritmo de monitoramento da métrica de uso de Disco.

```
INÍCIO – SCRIPT DE MONITORAMENTO DO DISCO
1:      execucao = 0
2:      contador = 0
3:      ENQUANTO (VERDADE)
4:          aux = 0
5:              ENQUANTO (aux =< 60)
6:                  esperar 5s
7:                  pid = pidof minikube ou k3s
8:                  disco = pidstat do pid (read(kB/s) write(kB/s) cancelled(kB/s))
9:                  imprima disco contador
10:                 contador = contador + 5
11:             FIM - ENQUANTO
12:         execucao = execucao + 60
13:         SE (execucao = 400000)
14:             entao
15:                 pare
16:         FIM - SE
17:     FIM - ENQUANTO
FIM
```

Fonte: O Autor.

Tabela 12 – Algoritmo de monitoramento da métrica de consumo de Memória.

INÍCIO – SCRIPT DE MONITORAMENTO DA MEMÓRIA	
1:	execucao = 0
3:	**ENQUANTO (VERDADE)**
4:	swap = free –mega (swap)
4:	memfree = free –mega (free)
4:	memused = free –mega (used)
4:	membuff = free –mega (buff/cached)
4:	memshared = free –mega (shared)
4:	memavailable = free –mega (available)
4:	swap_free = swap (free)
4:	swap_used = swap (used)
4:	imprima memused memfree memavailable memshared
1:	membuff swap_used swap_free em arquivo
4:	espera 60s
4:	execucao = execucao + 60
5:	**SE (execucao = 400000)**
6:	entao
7:	pare
9:	**FIM - SE**
17:	**FIM - ENQUANTO**
FIM	

Fonte: O Autor.

Tabela 13 – Algoritmo de requisições e monitoramento.

INÍCIO – SCRIPT DE MONITORAMENTO E REQUISIÇÕES	
1:	pingar por 400000s a cada 60s endereco Pod (Serviço Nginx)
2:	**PARA(contador=0; contador < 420; contador++)**
3:	**ENQUANTO LER linha**
4:	imprima data() - linha no ARQUIVO
5:	**FIM - ENQUANTO**
6:	**FIM - PARA**
FIM	

Fonte: O Autor.