UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

FLÁVIO BORGES DE LIMA

IMPLANTAÇÃO DE CLUSTER KUBERNETES EM COMPUTADORES COM PODER COMPUTACIONAL LIMITADO

GUARAPUAVA

2025

FLÁVIO BORGES DE LIMA

IMPLANTAÇÃO DE CLUSTER KUBERNETES EM COMPUTADORES COM PODER COMPUTACIONAL LIMITADO

Deployment of Kubernetes Cluster for Computer Network with Limited Computational Power

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Hermano Pereira

Coorientador: Profª. Drª. Sediane Carmem Lunardi Hernandes

GUARAPUAVA

2025

FLÁVIO BORGES DE LIMA

IMPLANTAÇÃO DE CLUSTER KUBERNETES EM COMPUTADORES COM PODER COMPUTACIONAL LIMITADO

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 26/novembro/2025

Hermano Pereira
Doutor
Universidade Tecnológica Federal do Paraná

Sediane Carmem Lunardi Hernandes
Doutora
Universidade Tecnológica Federal do Paraná

William Alberto Cruz Castaneda
Doutor
Universidade Tecnológica Federal do Paraná

GUARAPUAVA

2025

AGRADECIMENTOS

Certamente estes parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase de minha vida. Portanto, desde já peço desculpas àquelas que não estão presentes entre essas palavras pois foram muitas, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão.

Agradeço ao meu orientador Prof. Dr. Hermano Pereira e a minha coorientadora Profª. Drª. Sediane Carmem Lunardi Hernandes pela sabedoria, conhecimento e paciência enorme com que me guiou nesta trajetória.

Agradeço aos meus pais (Jaqueline e Márcio), irmão (Ângelo Gabriel) e demais familiares.

A todos os alunos e professores da UTFPR que me apoiaram e incentivaram durante este percurso.

Em especial quero agradecer ao projeto de extensão Tecnolixo por fornecer o *hardware* utilizado no trabalho. Também gostaria de agradecer aos participantes do projeto de extensão Asimov, em especial ao Daniel Ahyub Lacerda, por ajudar na manutenção de alguns dos computadores.

Enfim, a todos os que por algum motivo contribuíram para a realização desta pesquisa.

RESUMO

Este trabalho propõe uma alternativa econômica, sustentável e funcional para a reutilização de *hardware* com recursos computacionais limitados, por meio da implantação de um *cluster* Kubernetes. Foram avaliadas três distribuições otimizadas da ferramenta — *k3s*, *k0s* e *MicroK8s* — instaladas em computadores antigos, com o objetivo de analisar o desempenho sob diferentes cargas de trabalho. Os experimentos incluíram aplicações com requisições concorrentes e tarefas sequenciais intensivas em CPU. Os resultados indicam que o uso dessas distribuições em ambientes com restrições de recursos é viável, especialmente em cenários com aplicações distribuídas leves. A proposta demonstrou potencial para uso em contextos educacionais, laboratoriais e projetos de baixo custo, destacando-se como uma solução prática para o reaproveitamento de equipamentos obsoletos.

Palavras-chave: kubernetes; clusters; recursos limitados; computação distribuída; otimização de desempenho.

ABSTRACT

This work presents an economical, sustainable, and functional proposal for reusing computers with limited computational power by deploying a Kubernetes cluster. Three lightweight distributions — *k3s*, *k0s*, and *MicroK8s* — were tested on outdated machines to assess their performance under different workloads. The experiments involved applications with concurrent requests and sequential, CPU-intensive tasks. The results showed that these distributions are viable for managing lightweight distributed applications in constrained environments. The proposed solution demonstrates potential for educational and experimental contexts, standing out as a practical and low-cost alternative for repurposing obsolete hardware.

Keywords: kubernetes; clusters; limited resources; distributed computing; performance optimization.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

API	Interface de Programação de Aplicações (API, do inglês *Application Programming Interface*)
ARP	Protocolo de Resolução de Endereços (ARP, do inglês *Address Resolution Protocol*)
BGP	Protocolo de Gateway de Fronteira (BGP, do inglês *Border Gateway Protocol*)
CPU	Unidade de Processamento Central (CPU, do inglês *Central Processing Unit*)
GB	Gigabyte (GB)
HA	Alta Disponibilidade (HA, do inglês *High Availability*)
HPC	Computação de Alta Performance (HPC, do inglês *High-Performance Computing*)
HTTP	Protocolo de Transferência de Hipertexto (HTTP, do inglês *Hypertext Transfer Protocol*)
IoT	Internet das Coisas (IoT, do inglês *Internet of Things*)
IP	Protocolo de Internet (IP, do inglês *Internet Protocol*)
JSON	Notação de Objetos JavaScript (JSON, do inglês *JavaScript Object Notation*)
L2	Camada 2 (L2, do inglês *Layer 2*)
LAN	Rede de Área Local (LAN, do inglês *Local Area Network*)
LB	Balanceador de Carga (LB, do inglês *Load Balancer*)
OS	Sistema Operacional (OS, do inglês *Operating System*)
OSI	Modelo de Interconexão de Sistemas Abertos (OSI, do inglês *Open Systems Interconnection*)
RAM	Memória de Acesso Aleatório (RAM, do inglês *Random Access Memory*)
RPS	Requisições por Segundo (RPS, do inglês *Requests Per Second*)
SSH	SSH, do inglês *Secure Shell*

Acrônimos

K8s	k8s é uma abreviação comum para Kubernetes, onde o ’8’ representa as oito letras entre o ’K’ e o ’s’ (APPVIA, 2024)

RESTful *Representational State Transfer*

SUMÁRIO

1 INTRODUÇÃO

O conceito de *cluster* tem se tornado fundamental em ambientes que exigem processamento distribuído (MONTEIRO *et al.*, 2021). Um *cluster* é formado por um conjunto de computadores que trabalham de forma colaborativa, aumentando a capacidade de processamento ao distribuir tarefas entre os computadores. (IBM, 2024). Isso permite que sistemas computacionais com recursos limitados, como computadores antigos, possam ser reaproveitados, sem custos adicionais consideráveis, em configurações que maximizam o desempenho do sistema como um todo.

Aplicações web podem ser distribuídas, permitindo que o armazenamento e o processamento de dados sejam divididos entre os vários computadores em uma rede ou *cluster* de computadores. De forma mais abrangente, aplicações que seguem o paradigma cliente-servidor ou peer-to-peer podem se beneficiar do uso de *clusters* de computadores para um sistema distribuído.

Com tecnologias como *containers*, que isolam aplicações e garantem a portabilidade entre diferentes sistemas, e o Kubernetes, que orquestra e gerencia esses *containers* (CNCF, 2024a), torna-se possível utilizar *clusters* Kubernetes para diversas finalidades, como mencionado em (MONTEIRO *et al.*, 2021), p. 154, "em seu *cluster* próprio (de forma local), em um sistema de arquitetura híbrida ou até mesmo em qualquer provedor de computação na nuvem pública.". Um *cluster* de computadores é considerado um sistema de arquitetura híbrido. Logo, um *cluster* Kubernetes pode ser implantado considerando essa arquitetura.

A principal motivação deste trabalho é oferecer uma alternativa econômica e ambientalmente sustentável ao reaproveitamento de computadores com poder computacional limitado, que normalmente seriam descartados. Ao implementar *clusters* Kubernetes nesses equipamentos, é possível reduzir custos com aquisição de *hardware* e minimizar o impacto ambiental do descarte de eletrônicos, promovendo o uso eficiente de recursos já disponíveis. Essa abordagem viabiliza a criação de ambientes acessíveis para testes e desenvolvimento de aplicações distribuídas, especialmente em contextos acadêmicos e experimentais.

Desta forma, este trabalho explorou *cluster* Kubernetes em computadores com capacidade computacional limitada para aproveitar ao máximo o sistema computacional como um todo.

1.1 Objetivos

1.1.1 Objetivo geral

Este trabalho teve como objetivo geral avaliar o desempenho de um *cluster* Kubernetes, para aplicações distribuídas stateless, o qual foi implantado em máquinas com poder computacional limitado.

1.1.2 Objetivos específicos

- Montar uma rede de computadores utilizando computadores com capacidade computacional limitada.
- Implantar um *cluster* Kubernetes na rede de computadores criada.
- Avaliar o desempenho do *cluster* Kubernetes criado, considerando suas limitações computacionais, utilizando aplicações stateless, como aplicações web, por exemplo.
- Documentar o processo de configuração e implantação do *cluster* Kubernetes.
- Montar os gráficos resultantes das avaliações de desempenho (por exemplo, uso de CPU, RAM e tempo de resposta) realizadas.

1.2 Estrutura do trabalho

O trabalho está dividido como segue. A Seção 1 contextualiza o problema e apresenta os objetivos do trabalho e a justificativa para o seu desenvolvimento. A Seção 2 refere-se ao Referencial Teórico, o qual aborda os principais conceitos e tecnologias utilizados no trabalho, como *clusters*, *containers*, Kubernetes e Docker, além de discutir os desafios de uso de *hardware* ultrapassado em ambientes distribuídos. A Seção 3 descreve os trabalhos relacionados e na Seção 4 são citados o *software*, *hardware* e metodologias de testes que foram usados no decorrer do trabalho. Na Seção 4, o desenvolvimento deste trabalho é apresentado, e na Seção 5 os resultados são descritos. Por fim, as referências bibliográficas são apresentadas.

2 REFERENCIAL TEÓRICO

Nesta Seção, apresentaram-se conceitos fundamentais relacionados ao desenvolvimento deste trabalho, incluindo *clusters*, *containers*, orquestração de *containers* e Kubernetes, com foco na implantação de *clusters* Kubernetes em máquinas com poder computacional limitado.

2.1 Clusters

Segundo a IBM (2024), *clusters* são aglomerados de computadores que trabalham juntos para executar tarefas, balanceando a carga de trabalho. Cada computador em um *cluster* é chamado de nó (*node*), e cada nó é formado por um Sistema Operacional (OS, do inglês *Operating System*) e um *software* intermediário, que é responsável por gerenciar a comunicação e a execução de tarefas no *cluster*. A estrutura de um *cluster* pode variar entre um único nó até milhares de nós, geralmente conectados por uma Rede de Área Local (LAN, do inglês *Local Area Network*) (IBM, 2024).

Os *clusters* podem ser divididos em dois grupos principais:

- Computação de Alta Performance (HPC, do inglês *High-Performance Computing*): Projetados para executar tarefas que precisam de um grande poder computacional, como simulações científicas ou processamento de grande volume de dados.
- Alta Disponibilidade (HA, do inglês *High Availability*): Desenvolvidos para garantir que os serviços e recursos permaneçam acessíveis, mesmo em caso de falhas em um ou mais nós, geralmente sendo utilizados na hospedagem de aplicações.

Neste trabalho, foi utilizado um *cluster* de HA, gerenciado pelo Kubernetes, que foi explorado na Seção 4. A escolha do mesmo foi feita com base na facilidade de adicionar novos nós (CNCF, 2024a) e na flexibilidade fornecida pelos *containers* (GOOGLE, 2024a).

No caso do Kubernetes, o *cluster* é composto por um ou mais nós de plano de controle (do inglês, *control plane*) e um ou mais nós de trabalho (CNCF, 2024). Os nós de plano de controle são responsáveis pela administração centralizada do *cluster*, enquanto os nós de trabalho executam as cargas de trabalho, ou seja, os *pods* e *containers* das aplicações, mas em alguns casos, o plano de controle pode ser configurado para operar como um nó de trabalho também (ver Seção 2.3).

2.2 Containers

Segundo a Google (2024a) *containers* são "*pacotes de software que contêm todos os elementos necessários para serem executados em qualquer ambiente*". Os *containers* utilizam

o mesmo *kernel* que a máquina hospedeira e, além disso, podem compartilhar camadas de sistema de arquivos, o que permite que múltiplos *containers* usem as mesmas dependências sem a necessidade de duplicá-las, poupando assim recursos computacionais como memória e armazenamento.

Algumas características fundamentais dos *containers* incluem a fácil e confiável recriação de um mesmo *container*, o isolamento entre as aplicações que estão sendo executadas dentro de *containers* distintos em uma mesma máquina, e geralmente sendo mais leves em comparação com máquinas virtuais (GOOGLE, 2024a). O isolamento é uma grande vantagem dos *containers*, garantindo que as aplicações executadas em *containers* diferentes não interfiram uma nas outras, apesar de estarem rodando no mesmo OS e compartilhando o mesmo *kernel*. Isso oferece segurança, evita conflitos de dependências entre os diferentes ambientes de execução e gera uma facilidade em hospedar aplicações completamente distintas em um mesmo ambiente.

Essas características tornam os *containers* ideais para maximizar a utilização do *hardware* disponível. Devido à sua leveza em relação a máquinas virtuais e flexibilidade, é possível executar múltiplos *containers* em uma única máquina ou distribuir a carga de trabalho entre *containers* alocados em diferentes máquinas, otimizando a eficiência e a escalabilidade dos recursos computacionais.

2.2.1 Orquestração de *containers*

De acordo com o Google Cloud (GOOGLE, 2024b), a orquestração de *containers* refere-se à automação do gerenciamento de recursos e do ciclo de vida dos *containers*. Essa prática desempenha um papel crucial na garantia da eficiência e escalabilidade em ambientes com múltiplos *containers*, seja em sistemas independentes ou operando em *clusters*.

Neste trabalho, o Kubernetes foi empregado como a principal ferramenta de orquestração. Sua implantação permitiu a gestão eficiente dos recursos computacionais disponíveis, bem como a distribuição equilibrada da carga de trabalho entre as máquinas do *cluster*, maximizando a utilização de *hardware* limitado.

2.3 Kubernetes

O Kubernetes, também conhecido como K8s[1], é uma plataforma de orquestração de *containers* desenvolvida pela Google em 2014, projetada para automatizar a implantação, o dimensionamento e a operação de *containers* (CNCF, 2024). O Kubernetes permite a criação de *clusters* de HA, compostos por um ou mais nós de plano de controle e um ou mais nós

[1] O termo “K8s” é uma abreviação de Kubernetes, onde o número “8” representa as oito letras entre o “K” e o “s” na palavra “Kubernetes” (APPVIA, 2024). É uma prática comum em tecnologia para abreviar nomes longos e complexos.

de trabalho. Os nós de plano de controle são responsáveis pela administração centralizada do *cluster*, enquanto os nós de trabalho executam as cargas de trabalho, ou seja, os *pods* e *containers* das aplicações, mas em alguns casos, o plano de controle pode ser configurado para operar como um nó de trabalho também.

Os nós de plano de controle são responsáveis por manter o estado desejado do *cluster* dentro de parâmetros definidos pelo administrador, como a quantidade de cópias de um *pod* contendo um ou mais *containers* de uma aplicação ou a escolha do nó mais adequado para executar a mesma, avaliando os recursos disponíveis.

Os nós de trabalho são responsáveis por executar um ou mais *pods*. Os *pods* são as menores unidades de trabalho no Kubernetes, e cada um pode conter um ou mais *containers* que compartilham recursos como rede e armazenamento.

Segundo CNCF (2024b). Os *pods* são efêmeros, o que significa que estes podem ser criados, destruídos e recriados conforme necessário, dependendo das necessidades do *cluster*. Os *pods* são gerenciados por um ou mais nós de plano de controle do Kubernetes, que monitora o estado dos *pods* e garante que estes estejam sempre em conformidade com o estado desejado definido pelo administrador do *cluster*. Se um *pod* falhar, ser destruído ou seja alterada a quantidade de réplicas, o plano de controle pode automaticamente criar um ou mais *pods* para substituí-lo, garantindo a continuidade do serviço.

No contexto deste trabalho, a aplicação do Kubernetes foi fundamental no balanceamento de carga de trabalho entre as máquinas do *cluster* utilizando um LB, garantindo o uso eficiente do *hardware* disponível e maximizando o aproveitamento dos recursos computacionais.

2.3.1 Balanceamento de Carga

Dentro do Kubernetes, os *pods* possuem uma rede interna aonde *containers* podem se comunicar entre si, mas aonde a rede externa não pode acessar esses. Para clientes que desejam acessar aplicações existentes nesses *pods*, faz-se necessária a criação de *Services*. Os *Services* expõem um ou mais *pods* para a rede externa. Existem diversos tipos de *Services* com a mesma função, entretanto, neste trabalho o foco foi o *LB Service* (CNCF, 2024).

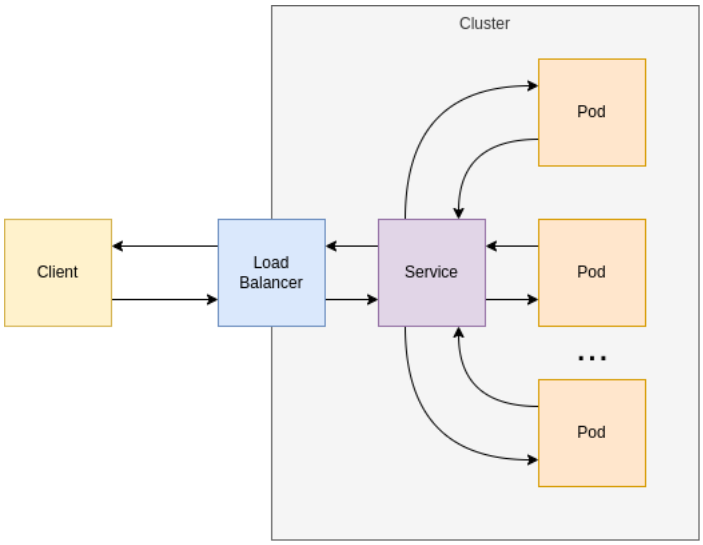
O *LB Service* no Kubernetes tem a capacidade de distribuir requisições entre um ou mais nós, facilitando o acesso externo às aplicações hospedadas nos *pods*. Existem diversas implementações de LB para o Kubernetes; normalmente, esse componente é um serviço externo que se integra ao *Service* do Kubernetes para encaminhar as requisições de forma equilibrada entre os *pods* disponíveis. Neste trabalho, foi utilizado o *MetalLB* [2] (CNCF, 2025a), uma solução de LB voltada para ambientes *bare metal*, ou seja, para infraestruturas que não estão hospedadas em provedores de nuvem.

[2] O nome *MetalLB* é uma combinação de "*metal*" (*bare metal*) e LB, refletindo seu propósito de fornecer balanceamento de carga em ambientes *bare metal*. (CNCF, 2025a)

O *MetalLB* pode operar em dois modos principais (CNCF, 2025a): Protocolo de Gateway de Fronteira (BGP, do inglês *Border Gateway Protocol*) e Camada 2 (L2, do inglês *Layer 2*). No modo L2, correspondente à camada de enlace do modelo Modelo de Interconexão de Sistemas Abertos (OSI, do inglês *Open Systems Interconnection*), o *MetalLB* responde a requisições de Protocolo de Resolução de Endereços (ARP, do inglês *Address Resolution Protocol*) para um endereço Protocolo de Internet (IP, do inglês *Internet Protocol*) virtual e elege um nó do *cluster* para ser responsável por esse endereço, recebendo todas as requisições destinadas a esse. Caso esse nó falhe, outro nó é automaticamente escolhido para assumir a responsabilidade. Já no modo BGP, o *MetalLB* anuncia rotas para o endereço IP virtual utilizando o protocolo BGP, permitindo que o tráfego seja roteado para os nós do *cluster* por meio de roteadores externos. Neste trabalho, optou-se pelo modo L2 devido à sua simplicidade, facilidade de configuração em ambientes locais e não necessitar de *hardware* especializado, tornando-o ideal para clusters com recursos computacionais limitados.

No caso da utilização do *MetalLB* em modo L2, o balanceamento de carga é feito de forma automática pelo *Service* do Kubernetes, que utiliza o *kube-proxy* para balancear as requisições entre os *pods* disponíveis. O *kube-proxy* é um componente do Kubernetes que atua como um *proxy* de rede, roteando o tráfego para os *pods* corretos com base nas regras definidas no *Service* (CNCF, 2025b), a regra padrão é o *iptables*, que distribui as requisições entre os *pods* de forma randomizada.

Figura 1 – Balanceamento de carga usando o LB no Kubernetes.



Fonte: Adaptação de (GEEKSFORGEEKS, 2025).

Conforme pode ser observado na Figura 1, as requisições são recebidas pelo LB e depois distribuídas pelo *Service* entre os *pods*, que contêm a mesma aplicação, juntamente com a administração automática de qual nó de trabalho tem quais *pods* no *cluster* Kubernetes. O LB é responsável por balancear a carga de processamento das requisições entre as máquinas do *cluster*.

2.3.2 Distribuições do Kubernetes

Assim como outros diversos projetos de código aberto, o Kubernetes tem diversas variações que foram criadas com base no seu projeto inicial. E algumas dessas variações se especializaram na diminuição do consumo de recursos, como RAM e processamento da CPU, podendo ser citadas:

- O *k3s*, desenvolvido para ambientes de Internet das Coisas (IoT, do inglês *Internet of Things*), é otimizado para ser implantado em uma ampla gama de arquiteturas de sistemas (Rancher Labs, 2024).
- O *MicroK8s* é uma versão simplificada e otimizada do Kubernetes, a qual foi projetada para permitir a criação de *clusters* com facilidade e rapidez (CANONICAL, 2024).
- O *k0s* é uma variante *bare metal* do Kubernetes, ideal para ser utilizado em IoT, com flexibilidade para ser implantado em diversos ambientes (K0S PROJECT, 2024).

As distribuições são variações que podem ser utilizadas em diferentes ambientes e para vários propósitos.

Neste trabalho foram escolhidas distribuições do Kubernetes que são otimizadas para consumir poucos recursos computacionais, como RAM e processamento da CPU, visando maximizar o aproveitamento do *hardware* disponível. As distribuições escolhidas foram o *k3s*, o *MicroK8s* e o *k0s*.

2.4 Computadores com poder computacional limitado

O conceito de computadores com capacidade computacional limitada pode variar dependendo do contexto de uso. Para os propósitos deste trabalho, define-se como máquinas com recursos modestos em relação aos requisitos típicos de aplicações modernas. Nesta definição, incluem-se computadores que possuem 4 Gigabyte (GB) de RAM, processadores com 4 núcleos a 2 núcleos de CPU com arquitetura x86-64, características que restringem sua capacidade de executar tarefas intensivas em recursos. Os computadores que foram utilizados são descritos na Seção 4.1.1.

3 TRABALHOS RELACIONADOS

Diversos estudos abordam a utilização de Kubernetes em ambientes com recursos limitados, destacando-se pela adaptação de arquiteturas para permitir a operação eficiente em *hardware* de baixo custo. Esses trabalhos são relevantes para este, pois apresentam alternativas e abordagens que se alinham com a proposta de implantar um *cluster* Kubernetes em computadores com poder computacional limitado.

Um estudo realizado por Silva (2022) aborda a implementação de soluções computacionais em ambientes com *hardware* modesto utilizando tecnologias como o Kubernetes para otimização de recursos em sistemas com capacidade computacional limitada. Esse trabalho é particularmente interessante, pois explora as mesmas questões relacionadas ao uso de *clusters* Kubernetes em máquinas de baixo custo, oferecendo perspectivas relacionadas aos desafios de desempenho e eficiência.

O trabalho de *Programming Group* (GROUP, 2023) explora diferentes distribuições de Kubernetes otimizadas para ambientes com recursos limitados, como o *k3s*, *k0s* e *Microk8s*, destacando a importância dessas versões leves para a implementação de *clusters* em *hardware* modesto. Essa pesquisa complementa a proposta deste trabalho, já que o uso de distribuições otimizadas pode ser uma solução crucial para garantir o funcionamento eficiente do *cluster*, mesmo em máquinas com pouca memória RAM e capacidade de processamento.

Outro estudo relevante é o de *Learn Fast Make Things* (MORSE, 2023), que discute como transformar *hardware* antigo em *clusters* Kubernetes, utilizando recursos limitados de maneira eficiente. Esse trabalho se aproxima diretamente do objetivo deste, ao sugerir métodos de aproveitamento de *hardware* obsoleto para a criação de *clusters*, alinhando-se à proposta de utilizar máquinas com poder computacional limitado para implementação de soluções escaláveis.

Esses trabalhos forneceram uma base teórica e prática sólida, que orientou a escolha de tecnologias e estratégias para a implementação de *clusters* Kubernetes em *hardware* com recursos limitados, contribuindo significativamente para o desenvolvimento deste trabalho.

4 MATERIAIS E MÉTODOS

Esta Seção descreve os materiais e métodos utilizados para a implantação e avaliação de um *cluster* Kubernetes em computadores com poder computacional limitado. A Seção 4.1 detalha os componentes de *hardware* e *software* empregados, enquanto a Seção 4.2 apresenta os testes realizados, procedimentos adotados para realização desses e as métricas coletadas durante a execução.

4.1 Materiais

Os materiais utilizados estão organizados em duas categorias principais: *hardware*, que abrange os computadores e equipamentos de rede, e *software*, que compreende as distribuições de Kubernetes e ferramentas utilizadas para realizar os testes e monitoramento.

4.1.1 *Hardware*

As Especificações técnicas dos computadores que formaram o *cluster* pode ser observadas na tabela a seguir:

Tabela 1 – Especificações técnicas dos computadores.

Computador	Núcleos CPU	*Clock* CPU	*Cache* CPU	RAM	Velocidade de Rede
Nó Plano de Controle/Trabalho	4	2.95 GHz	8 MB	4 GB	100 Mbps
Nó de Trabalho 1	4	3.0 GHz	12 MB	4 GB	100 Mbps
Nó de Trabalho 2	2	3.0 GHz	3 MB	4 GB	100 Mbps

Fonte: Elaborado pelo autor.

E as peças que formaram os computadores são:

Tabela 2 – Peças que compuseram os computadores.

Computador	CPU	Placa mãe	RAM	Disco	Placa de
Plano de Controle/Nó de Trabalho	i7-870	bpc-hm55	kvr16n11/4	wd5000lpvx	Onbo
Nó de Trabalho 1	e5450	g41m	BMD34096M1333C9	wd5000lpvx	Onbo
Nó de Trabalho 2	g2030	0xfwhv	m378b5273eb0-ck0	st500dm002-1bd142	Onbo

Fonte: Elaborado pelo autor.

Sendo o nó de plano de controle atuando também como nó de trabalho e dois computadores adicionais atuando apenas como nós de trabalho.

Além dos computadores, foi utilizado um *switch* de rede Encore do modelo enh916p-nwy com *fast ethernet* (até 100 Megabits por segundo de velocidade) para interligar os dispositivos e um computador configurado para servir como roteador.

4.1.2 *Software*

O *software* utilizado no desenvolvimento deste trabalho pode ser dividido em cinco categorias principais: distribuições de Kubernetes, componentes adicionais instalados no Kubernetes, pacotes instalados nas máquinas, linguagem e tecnologias utilizadas na aplicação de teste e ferramentas para desenvolvimento do *script* de testes.

As distribuições de Kubernetes utilizadas foram: *k3s*, *MicroK8s* e *k0s* que são melhores descritas na Seção 2.3.2.

Componentes adicionais instalados no Kubernetes:

- **MetalLB**: Implementação de LB para ambientes *bare metal*, permitindo a distribuição de tráfego entre os *pods* do *cluster* Kubernetes (CNCF, 2025a).

Os pacotes instalados nas máquinas que compuseram o *cluster* Kubernetes são:

- **Debian 11**: Sistema operacional baseado em Linux, conhecido por sua estabilidade e segurança, utilizado como base para a instalação das distribuições de Kubernetes (DEBIAN, 2024).
- **sysstat**: Conjunto de ferramentas para monitoramento de desempenho do sistema, utilizado para coletar métricas de uso da CPU e memória (SYSSTAT, 2024).
- **OpenSSH**: Conjunto de ferramentas para acesso remoto seguro via SSH, do inglês *Secure Shell*, utilizado para administração e monitoramento dos nós do *cluster* (OPENSSH, 2025).

A aplicação de teste foi desenvolvida utilizando as seguintes tecnologias:

- **Docker**: Plataforma de *containers* utilizada para empacotar a aplicação de teste e suas dependências, garantindo portabilidade e consistência entre os ambientes (DOCKER, 2024).
- **Node.js v22**: Ambiente de execução *JavaScript OpenSource* utilizado no desenvolvimento da aplicação de teste (NODE.JS, 2024).
- **Express**: Framework para Node.js que facilita a criação de aplicações web e Interface de Programação de Aplicações (API, do inglês *Application Programming Interface*)s RESTful (EXPRESS, 2024).
- **Axios**: Biblioteca Protocolo de Transferência de Hipertexto (HTTP, do inglês *Hypertext Transfer Protocol*) utilizada para realizar as requisições entre as partes da aplicação de teste (AXIOS, 2024).

As ferramentas utilizadas para o desenvolvimento do *script* são:

- **Python 3.9**: Linguagem utilizada na implementação do *script*, responsável tanto pela coleta de dados via SSH quanto pela execução das requisições à aplicação (PYTHON SOFTWARE FOUNDATION, 2025).
- **paramiko**: Biblioteca *Python* para conexão SSH, utilizada no *script* para acessar remotamente os servidores e coletar métricas de uso dos recursos (PARAMIKO, 2024).
- **httpx**: Biblioteca HTTP assíncrona para *Python*, utilizada no *script* para gerar requisições concorrentes com controle preciso de tempo e volume (ENCODE, 2024).

4.2 Métodos

Nesta Seção são descritos os testes e métricas executados para avaliar o desempenho do *cluster* Kubernetes implementado em máquinas com poder computacional limitado.

4.2.1 Testes de Interação do *cluster*

Os testes descritos nesta Seção tiveram como objetivo avaliar o desempenho geral do *cluster* Kubernetes em diferentes cenários de uso. Os algoritmos implementados para os testes foram:

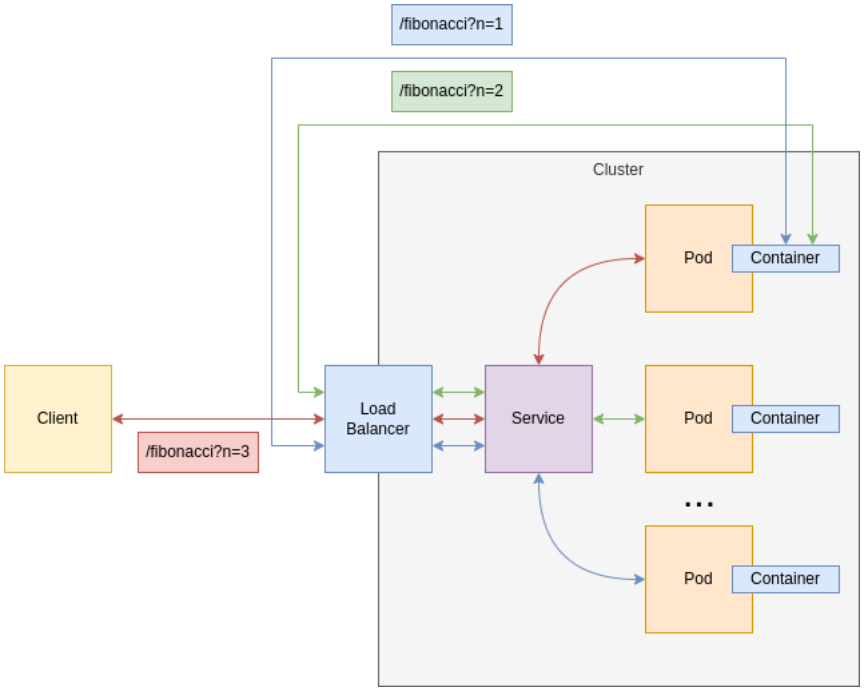
- Implementação do algoritmo do cálculo do n-ésimo número da série de *Fibonacci* (CORMEN *et al.*, 2024) usando requisições HTTP para formar uma recursão, assim simulando uma grande quantidade de requisições simultâneas e distribuindo a carga entre os nós do *cluster* ao passar as requisições pelo LB.

  A Figura 2 ilustra o ciclo de vida de uma requisição para calcular o dígito n da sequência de *Fibonacci*. O processo inicia-se com o cliente enviando a requisição ao LB, que a encaminha para um dos *pods* do *cluster*. O *pod* recebe a requisição e, por meio do *container* responsável pelo algoritmo, realiza chamadas recursivas ao próprio *Service* exposto, solicitando os valores de $n-1$ e $n-2$ — cada chamada passando novamente pelo LB assim distribuindo a carga entre os nós. Esse fluxo se repete até atingir os casos base ($n = 1$ ou $n = 2$), que retornam 1. Por fim, os resultados das chamadas recursivas são somados e o valor final é devolvido ao cliente.

  A abordagem utilizada baseia-se na definição recursiva da sequência de *Fibonacci*:

$$Fibonacci(n) = \begin{cases} 1, & \text{se } n \leq 2 \\ Fibonacci(n-1) + Fibonacci(n-2), & \text{senão} \end{cases} \quad (1)$$

Figura 2 – Ciclo de vida de um requisição do número de Fibonacci.



Fonte: Autoria própria (2025).

Essa definição gera uma grande quantidade de requisições simultâneas, simulando um cenário de alto tráfego no *cluster*. O objetivo desse teste é avaliar como o sistema se comporta sob cargas intensas, identificando possíveis gargalos e limitações de desempenho.

- Desenvolvimento de um algoritmo de ordenação utilizando o método de ordenação *Bubble Sort* (BIGGAR; GREGG, 2005) para processar listas invertidas de diferentes tamanhos. Esse teste visou um processo computacional relativamente pesado e não distribuído, assim sendo cada requisição era processada em um dos nós do *cluster* sem distribuição de carga.

Esses experimentos permitem avaliar o desempenho do *cluster* em situações diversas, desde alto tráfego de requisições até cargas computacionalmente intensivas, identificando possíveis gargalos e oportunidades para otimização.

A aplicação de teste foi implantada no *cluster* Kubernetes utilizando *pods*. Cada nó continha um *pod* que continha a aplicação de teste, permitindo a distribuição das requisições entre os nós do *cluster*.

4.2.2 Ferramenta de automação dos testes

Para padronizar os testes e garantir reprodutibilidade, foi desenvolvido um *script* em *Python 3.9*. Esse *script* teve como propósito automatizar a descoberta da carga máxima supor-

tada por cada distribuição Kubernetes, considerando o tempo médio de resposta como critério de aceitação, coletar métricas dos computadores usando SSH e armazenar os dados em um arquivo de Notação de Objetos JavaScript (JSON, do inglês *JavaScript Object Notation*) dos computadores e do tempo de resposta das requisições para posterior analise.

A lógica implementada foi dividida em duas etapas, executadas em rodadas. Cada rodada é definida por uma carga C, uma taxa de requisições por segundo R e pela função MédiaTempoDeResposta(C,R), que retorna o tempo médio de resposta (em segundos) para a combinação (C,R). Cada rodada consistiu de 30 segundos de geração de requisições seguidos por 30 segundos de pausa.

- **Etapa 1 – Descoberta da carga máxima para uma requisição por segundo:** O *script* incrementava progressivamente a carga C começando em 1 (número da sequência de *Fibonacci* sendo $DigitoDeFibonacci(C) = C$ ou itens para ordenação com *Bubble Sort* sendo $TamanhoDoVetor(C) = 2^C$) em cada rodada, inicialmente mantendo a taxa de requisições constante em $R = 1$. O processo continuava até que o tempo médio de resposta T ultrapassasse 2 segundos. Sendo a carga que ultrapassou o limite considerada C para a próxima fase. Essa etapa pode ser formalizada da seguinte maneira:

$$\text{MaiorCarga}(C) = \begin{cases} C, & \text{se MédiaTempoDeResposta}(C,1) > 2\,\text{s} \\ \text{MaiorCarga}(C+1), & \text{senão} \end{cases} \tag{2}$$

- **Etapa 2 – Determinação da taxa máxima de requisições para diferentes cargas:** Para os valores de carga $C-1$, $C-2$ e $C-3$, determinado na Etapa 1, o *script* definia a quantidade de requisições R com R começando em 1 e dobrando a cada rodada até que o tempo médio de resposta T excedesse 2 segundos. Em seguida, era realizada uma busca binária para identificar o maior valor de R que mantivesse o tempo de resposta médio abaixo do limite estabelecido. Essa etapa pode ser formalizada da seguinte maneira:

$$\text{MaiorTaxa}(C,R) = \begin{cases} \text{BuscaBinária}(R), & \text{se MédiaTempoDeResposta}(C,R) > 2\,\text{s} \\ \text{MaiorTaxa}(C,2R), & \text{senão} \end{cases} \tag{3}$$

O tempo de resposta médio de 2 segundos foi escolhido como critério de aceitação para garantir que o sistema mantivesse um desempenho aceitável sob carga, evitando tempos de espera excessivos que poderiam comprometer a experiência do usuário como descrito por (Jakob Nielsen, 2025).

Durante a execução dos testes, após a determinação de $C-1$, $C-2$ e $C-3$ e de suas respectivas taxas máximas de requisições, as métricas dos servidores — como uso de CPU e RAM — foram coletadas a intervalos regulares de 0,5 segundos. Paralelamente, os dados das requisições, incluindo o tempo de resposta de cada chamada, foram registrados e armazenados para posterior análise, permitindo uma visão detalhada e contínua do comportamento do sistema sob carga.

Esse procedimento de testes foi inicialmente executado em uma distribuição de Kubernetes escolhida como referência. Os parâmetros de carga (C) e de taxa de requisições máxima (R) determinados nessa execução foram reproduzidos nas demais distribuições, mantendo constantes a duração das rodadas e os períodos de pausa, mas ignorando o critério de aceitação (tempo médio de resposta maior ou igual à 2 segundos). Para cada distribuição, o *script* automatizou as etapas descritas, coletando e registrando de forma padronizada as métricas de tempo de resposta e o uso de recursos (CPU e RAM), o que permitiu comparações diretas e isentas de variabilidade humana.

Os testes foram executados por um computador externo ao *cluster*, conectado diretamente ao *switch*, de modo a não interferir nos recursos computacionais dos nós do sistema. Esse *script* também foi responsável pela coleta e armazenamento das métricas de tempo de resposta e, em conjunto com o *sysstat*, pelo monitoramento de uso de CPU e RAM via SSH.

Essa automação aumentou o rigor experimental ao padronizar a geração de carga, a coleta de métricas e o registro temporal dos eventos. O script controla de forma determinística as rodadas de teste (duração, períodos de pausa e valores de carga), realiza a amostragem das métricas de CPU e RAM em intervalos regulares de 0,5 s via SSH e grava todos os resultados em arquivos JSON juntamente com metadados (data e hora, nó, carga e parâmetros de execução). Essa padronização reduz variações introduzidas pela execução manual, facilita a reprodutibilidade dos experimentos e possibilita análises estatísticas sobre os dados brutos. A implementação completa do script, instruções de execução e exemplos dos dados coletados estão disponíveis no Apêndice A.

4.2.3 Métricas

Durante os testes, foram coletadas as seguintes métricas para avaliar o desempenho do sistema:

- Consumo de memória RAM: Foi monitorada a utilização de memória pelas máquinas individualmente e pelo *cluster* como um todo, identificando o impacto das diferentes cargas de trabalho.
- Utilização de processamento (CPU): Acompanhamento da carga de processamento em cada máquina e no *cluster* com o objetivo de detectar possíveis gargalos.

- Tempo de resposta: Medição do tempo necessário para as aplicações implantadas no *cluster* processarem as requisições enviadas, avaliando a eficiência do sistema.

5 RESULTADOS EXPERIMENTAIS

Esta seção apresenta os resultados obtidos durante a execução dos testes no *cluster* Kubernetes implementado com *hardware* de poder computacional limitado. O *cluster* operou com três nós (um nó de plano de controle/trabalho e dois nós somente de trabalho). As cargas e requisições máximas foram determinadas com a metodologia explicada na Seção 4.2.1 e foi escolhida arbitrariamente a distribuição *k3s* como referência. Adotou-se um critério de aceitação de tempo de resposta médio inferior a 2 s para definir o ponto em que o sistema ainda é considerado aceitável em termos de usabilidade e capacidade de atendimento; esse limite baseia-se em estudos de percepção de latência do usuário (Jakob Nielsen, 2025) e funciona como um parâmetro prático para comparar distribuições sob as mesmas restrições. Nas análises, os resultados foram avaliados considerando as métricas de CPU, RAM e desempenho de rede, de modo a correlacionar a experiência de resposta com a eficiência no uso de recursos do *cluster*.

5.1 Resultados por Tipo de Teste

Os dados coletados durante os testes de desempenho do *cluster* estão organizados nas tabelas a seguir de acordo com o tipo de carga computacional. Na Tabela 3, estão dispostos os resultados obtidos com o algoritmo da Série de *Fibonacci*, onde o campo "n" indica o número da sequência a ser calculado. Já a Tabela 4 apresenta os dados referentes ao teste de ordenação com o algoritmo *Bubble Sort*, sendo o campo "Tamanho do vetor" o expoente aplicado na geração das listas inversamente ordenadas (2^n).

Tabela 3 – Dados coletados durante a execução do teste de Fibonacci.

Distribuição	n	Tempo de resposta médio	Requisições/segundo
k0s	18	2.870s	1
k3s	18	1.413s	1
microk8s	18	3.303s	1
k0s	17	2.049s	2
k3s	17	1.482s	2
microk8s	17	3.705s	2
k0s	16	3.024s	3
k3s	16	2.021s	3
microk8s	16	3.044s	3

Fonte: Elaborado pelo autor

A Tabela 4 apresenta os dados coletados durante a execução do algoritmo de ordenação *Bubble Sort*.

Tabela 4 – Dados coletados durante execução do teste de Bubble Sort.

Distribuição	Tamanho do vetor	Tempo de resposta médio	Requisições/segundo
k0s	2^{15}	1.456s	4
k3s	2^{15}	1.472s	4
microk8s	2^{15}	1.480s	4
k0s	2^{14}	0.459s	22
k3s	2^{14}	0.497s	22
microk8s	2^{14}	0.438s	22
k0s	2^{13}	0.102s	38
k3s	2^{13}	0.111s	38
microk8s	2^{13}	0.107s	38

Fonte: Elaborado pelo autor

5.2 Análise de Desempenho

A análise dos resultados demonstra diferenças significativas no desempenho e na eficiência de recursos das distribuições Kubernetes sob cargas de trabalho distintas.

5.2.1 Série de *Fibonacci* (Processamento distribuído)

O teste de *Fibonacci* (Tabela 3), que é intensivo em CPU e demanda paralelismo, evidenciou que o desempenho está diretamente ligado à capacidade da distribuição de balancear a carga entre os nós.

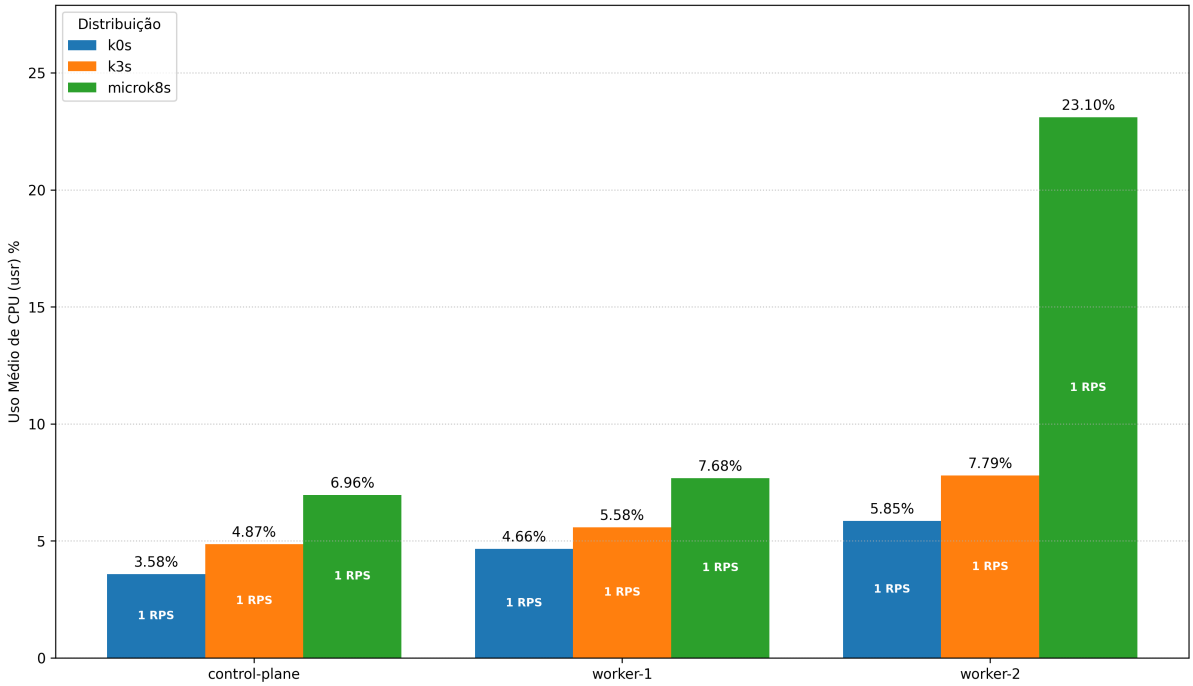
- Quanto ao desempenho a distribuição *k3s* para a carga de $n = 18$, o *k3s*(1,413s) foi o único capaz de atender a 1 Requisições por Segundo (RPS, do inglês *Requests Per Second*) dentro do limite aceitável de 2s, enquanto *k0s* e *MicroK8s* não conseguiram atender a essa carga dentro do limite.
- Para a carga de $n = 17$, o *k3s* (1,482s) novamente superou as outras distribuições, atendendo a 2 RPS dentro do limite aceitável, enquanto o *k0s* (2,049s) ficou ligeiramente acima do limite e o *MicroK8s* (3,705s) não conseguiu atender a essa carga.
- Na carga de $n = 16$, o *k3s* (2,021s) manteve a liderança em desempenho, atendendo a 3 RPS próximo ao limite aceitável, seguido pelo *k0s* (3,024s) e pelo *MicroK8s* (3,044s), ambos incapazes de atender a essa carga dentro do limite.

A distribuição *k3s*, provou ser a opção mais eficiente para um grande número de requisições simultâneas de baixa complexidade no contexto deste trabalho, especialmente em cenários de alta concorrência.

A análise integrada dos recursos de CPU e RAM (Figuras 3 e 4) revela a correlação direta entre o padrão de uso de recursos e o desempenho observado no teste de *Fibonacci*. O *k3s*, que obteve o melhor tempo de resposta (1,413s para $n = 18$), demonstrou uma estratégia equilibrada: consumo intermediário de CPU em todos os nós e maior alocação de RAM especialmente no nó de plano de controle/trabalho. Essa maior alocação de memória permite

que o *k3s* mantenha estruturas de dados otimizadas para o escalonamento e gerenciamento de tarefas paralelas, resultando em distribuição mais eficiente da carga entre os nós do *cluster*.

Figura 3 – Comparação de Uso de CPU por Nó (Fibonacci número 18).



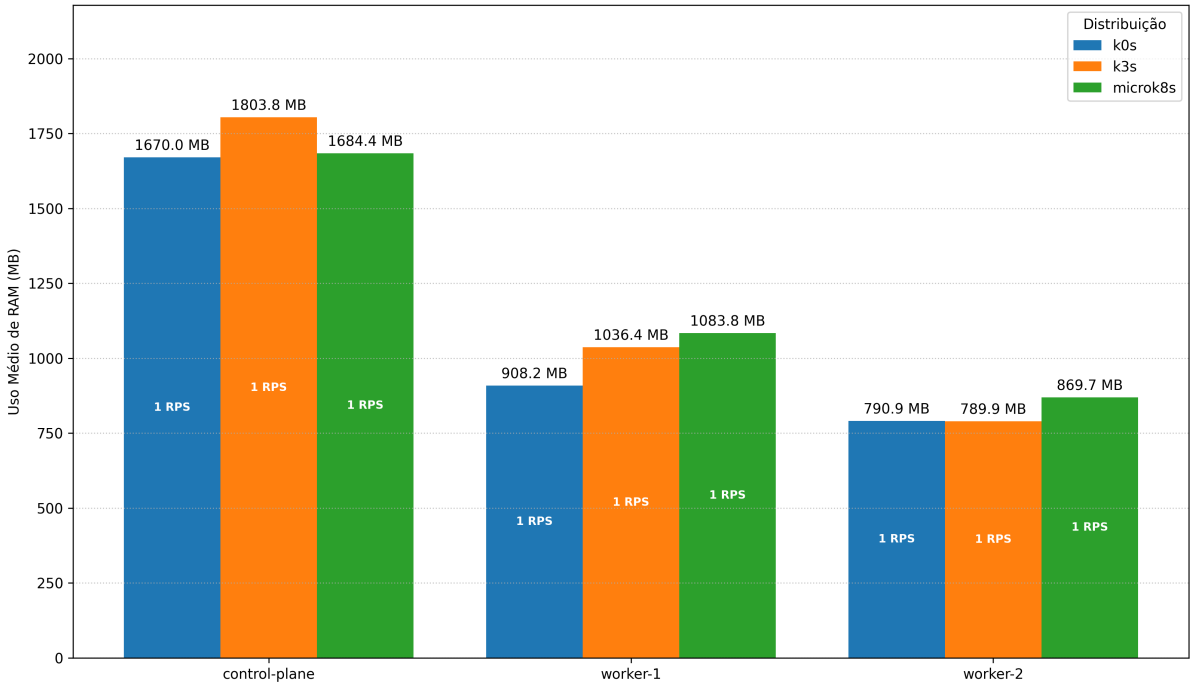
Fonte: Elaborado pelo autor

Em contraste, o *k0s* apresentou o menor consumo tanto de CPU quanto de RAM em todos os nós, mas obteve tempo de resposta 103% maior que o *k3s* (2,870s versus 1,413s). Essa economia de recursos indica uma abordagem conservadora na distribuição de carga: ao utilizar menos memória para gerenciamento, o *k0s* limita sua capacidade de paralelizar eficientemente as requisições, resultando em menor aproveitamento da capacidade computacional disponível no *cluster*. Essa característica torna o *k0s* adequado para ambientes com severas restrições de RAM, porém às custas de desempenho em cargas paralelas.

A distribuição *MicroK8s* exibiu o padrão mais ineficiente: maior consumo de CPU em todos os nós e alto consumo de RAM nos nós de trabalho, porém com o pior desempenho (3,303s — 134% mais lento que o *k3s*). Esse comportamento sugere sobrecarga excessiva de gerenciamento, onde o uso elevado de recursos não se traduz em melhor distribuição de tarefas. A combinação de alta utilização de CPU com baixo desempenho indica ineficiências arquiteturais, possivelmente relacionadas a processos de orquestração que competem com as aplicações pelos recursos computacionais.

Portanto, para cargas de trabalho paralelas intensivas em CPU, o investimento de memória RAM no plano de controle para gerenciamento eficiente do escalonamento é determinante para o desempenho. O *k3s* demonstra o melhor equilíbrio entre uso de recursos e capacidade de resposta, enquanto o *k0s* prioriza economia de recursos em detrimento do desempenho, e o *MicroK8s* apresenta ineficiência na conversão de recursos em ganhos de desempenho.

Figura 4 – Comparação de Uso de RAM por Nó (Fibonacci número 18).



Fonte: Elaborado pelo autor

5.2.2 Bubble Sort (Processamento Sequencial)

Os resultados do algoritmo *Bubble Sort* (Tabela 4), de natureza sequencial, indicaram um cenário diferente, dominado pela limitação do *hardware* e otimizações do consumo de recursos para processar cada tarefa individualmente.

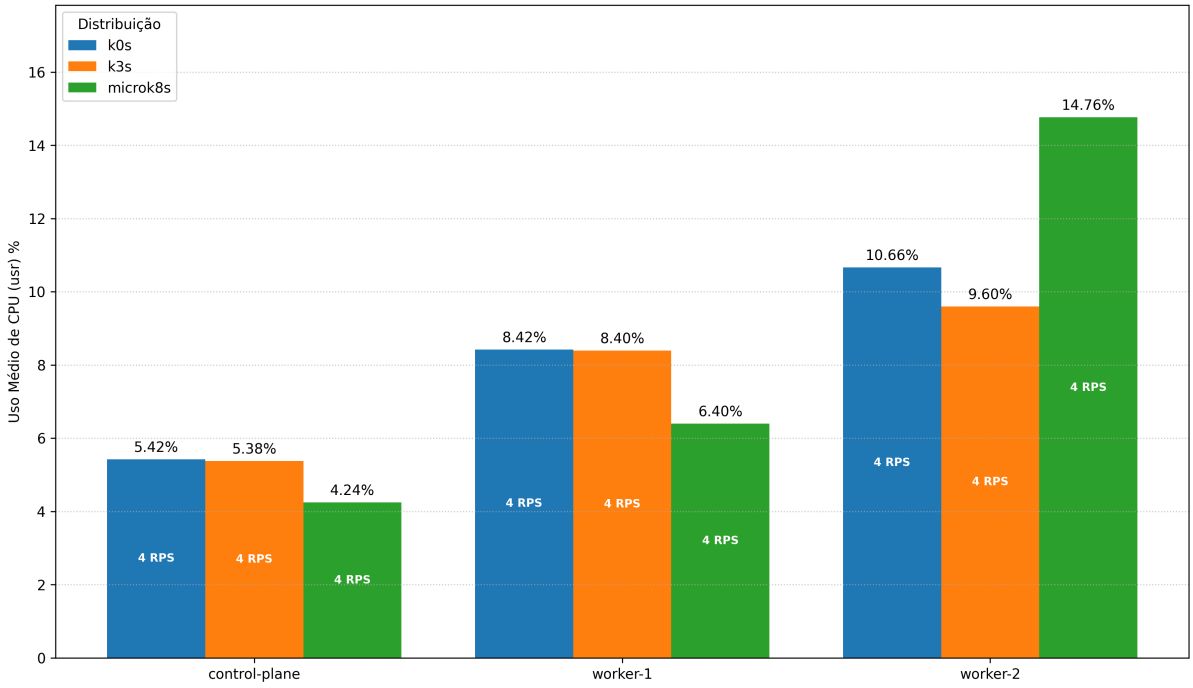
- Na carga mais pesada (2^{15} elementos), todas as distribuições apresentaram desempenho semelhante, com o *k0s* (1,456s), *k3s* (1,472s) e *MicroK8s* (1,480s) respondendo a 4 RPS.
- Na carga intermediária (2^{14} elementos), o *MicroK8s* (0,438s) superou o *k0s* (0,459s) e o *k3s* (0,497s), todos respondendo a 22 RPS. Porém todas as distribuições tiveram tempos de resposta muito próximos.
- Na carga mais leve (2^{13} elementos), o *k0s* (0,102s) liderou novamente, seguido pelo *MicroK8s* (0,107s) e pelo *k3s* (0,111s), todos respondendo a 38 RPS. Novamente, os tempos de resposta foram muito próximos entre as distribuições.

Para algoritmos computacionalmente intensivos e sequenciais, como o *Bubble Sort*, o *k0s* demonstrou ser ligeiramente mais eficiente, embora as diferenças de desempenho entre as distribuições fossem mínimas.

A análise conjunta dos recursos de CPU (Figura 5) e RAM (Figura 6) durante a execução do *Bubble Sort* com carga máxima (2^{15} elementos) revela padrões distintos que explicam por que, apesar dos tempos de resposta similares (*k0s*: 1,456s; *k3s*: 1,472s; *MicroK8s*: 1,480s

— diferença máxima de apenas 1,6%), as estratégias de uso de recursos diferem significativamente.

Figura 5 – Comparação de Uso de CPU por Nó (*Bubble Sort* 2^{15} items no vetor).



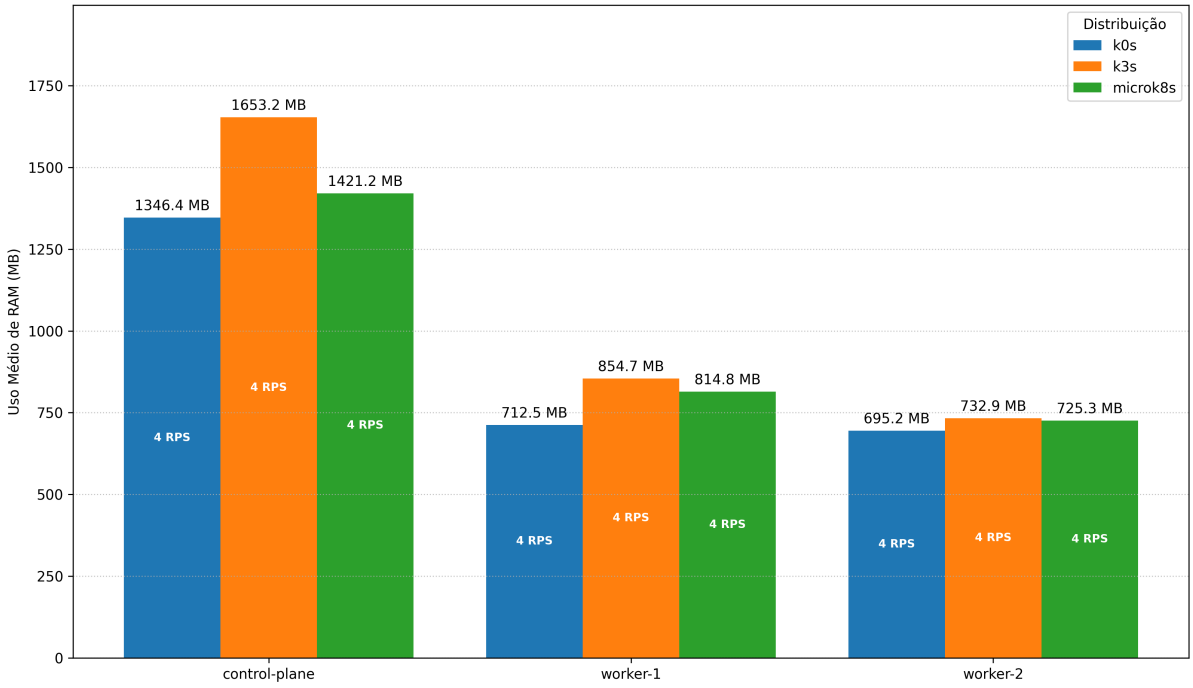
Fonte: Elaborado pelo autor

O *k0s* apresentou o perfil mais eficiente em termos de recursos, combinando o menor consumo de RAM em todos os nós com uso de CPU equivalente ao *k3s*, resultando no melhor tempo de resposta. Essa eficiência demonstra que, para cargas sequenciais onde não há benefício em paralelização, a economia de memória não prejudica o desempenho. O *k0s* otimiza a alocação de recursos ao evitar estruturas de gerenciamento supérfluas para este tipo de carga, tornando-se a opção ideal para ambientes com restrições severas de RAM que executam predominantemente tarefas sequenciais.

Em contraste, o *k3s* exibiu o maior consumo de RAM em todos os nós (especialmente no plano de controle/trabalho), apesar de manter uso de CPU similar ao *k0s*. Essa alocação adicional de memória, que foi vantajosa no teste de *Fibonacci*, não gerou benefícios no *Bubble Sort* — o tempo de resposta foi apenas 1,1% superior ao *k0s*. Isso indica que a estratégia do *k3s* de investir em memória para otimizar distribuição de carga é eficaz apenas em cenários com paralelismo real, representando sobrecarga desnecessária em processamento sequencial.

A distribuição *MicroK8s* apresentou padrão intermediário de consumo de RAM e uso desbalanceado de CPU, com menor utilização no plano de controle/trabalho porém maior no nó de trabalho 2. Esse desbalanceamento, embora não tenha impactado significativamente o tempo de resposta devido à natureza não-paralela da carga, evidencia ineficiências no escalonamento: idealmente, em cargas sequenciais, a distribuição de uso de CPU deveria ser homogênea entre os nós. O tempo de resposta ligeiramente inferior (1,480s) apesar do uso

Figura 6 – Comparação de Uso de RAM por Nó (*Bubble Sort* 2^{15} items no vetor).



Fonte: Elaborado pelo autor

equilibrado de recursos sugere que a sobrecarga de gerenciamento do *MicroK8s* permanece presente mesmo em cenários sequenciais.

Em síntese, para cargas sequenciais intensivas em CPU, a eficiência no uso de RAM torna-se mais relevante que a capacidade de distribuição, pois não há benefício em paralelização. O *k0s* demonstra ser a distribuição mais adequada ao combinar menor footprint de memória com desempenho equivalente, enquanto o *k3s* apresenta sobrecarga de RAM sem ganhos proporcionais. Esses resultados contrastam diretamente com o teste de *Fibonacci*, evidenciando que a escolha da distribuição deve considerar primordialmente a natureza da carga de trabalho (paralela versus sequencial) e as restrições de recursos do ambiente.

6 CONSIDERAÇÕES FINAIS

Este Trabalho teve como objetivo principal avaliar a viabilidade e o desempenho de um *cluster* Kubernetes em computadores com poder computacional limitado. Os resultados confirmaram a viabilidade técnica da proposta, demonstrando que é possível obter um desempenho satisfatório em aplicações *stateless* ao reutilizar *hardware* obsoleto.

O desempenho superior foi alcançado em aplicações que permitem o uso intensivo de requisições paralelas e distribuem a carga de trabalho de forma eficiente (Teste *Fibonacci*), com destaque para o *k3s*. Em contraste, em tarefas sequenciais e dependentes da CPU (Teste *Bubble Sort*), o desempenho foi limitado pelas características do *hardware*, e as diferenças entre as distribuições foram mínimas em termos de RPS, mas notáveis na eficiência de uso dos recursos.

Os testes verificaram empiricamente que *clusters* Kubernetes dentro dos testes realizados neste trabalho são mais eficazes ao lidar com múltiplas requisições de baixa complexidade do que com poucas requisições de alta carga computacional. Isso ocorre porque o Kubernetes é projetado para distribuir requisições entre os nós, mas não para paralelizar o processamento interno de uma única requisição, salvo em aplicações com chamadas internas entre *containers*.

Conclui-se que a proposta de reutilizar *hardware* limitado em ambientes de *cluster* representa uma alternativa promissora do ponto de vista econômico e sustentável. Como sugestão para trabalhos recomenda-se a validação do desempenho em cenários mais complexos, como o uso de bancos de dados distribuídos nos testes ou a implementação de *scripts* de teste em linguagens compiladas para reduzir a interferência do tempo de execução do *script* na análise de desempenho.

REFERÊNCIAS

APPVIA. **Why is Kubernetes Called K8s?** 2024. Disponível em: https://www.appvia.io/blog/why-is-kubernetes-called-k8s. Acesso em: 27 set. 2025.

AXIOS. **Axios - Promise based HTTP client for the browser and node.js**. 2024. Disponível em: https://axios-http.com/. Acesso em: 10 set. 2025.

BIGGAR, P.; GREGG, D. **Sorting in the Presence of Branch Prediction and Caches**. Dublin, Ireland, 2005. Disponível em: https://www.scss.tcd.ie/publications/tech-reports/reports.05/TCD-CS-2005-57.pdf.

CANONICAL. **MicroK8s Documentation**. 2024. Disponível em: https://microk8s.io/docs. Acesso em: 18 nov. 2024.

CNCF. **Cluster Architecture**. Online, 2024. Disponível em: https://kubernetes.io/docs/concepts/architecture/. Acesso em: 18 nov. 2024.

CNCF. **Service**. 2024. Disponível em: https://kubernetes.io/docs/concepts/services-networking/service/. Acesso em: 8 set. 2025.

CNCF. **MetalLB**. 2025. Disponível em: https://metallb.io/. Acesso em: 10 set. 2025.

CNCF. **Virtual IPs and Service Proxies**. 2025. Disponível em: https://kubernetes.io/docs/reference/networking/virtual-ips/. Acesso em: 10 set. 2025.

CNCF . **Overview**. Online, 2024. Disponível em: https://kubernetes.io/docs/concepts/overview/. Acesso em: 18 nov. 2024.

CNCF . **Pods**. Online, 2024. Disponível em: https://kubernetes.io/docs/concepts/workloads/pods/. Acesso em: 20 nov. 2024.

CORMEN, T. H. *et al.* **Algoritmos**. 4. ed.. ed. Rio de Janeiro: GEN LTC, 2024. ISBN 9788595159914. Disponível em: https://app.minhabiblioteca.com.br/reader/books/9788595159914/. Acesso em: 27 set. 2025.

DEBIAN. **Debian – Reasons to use Debian**. 2024. Accessed: 20 Nov. 2024. Disponível em: https://www.debian.org/intro/why_debian.

DOCKER. **Docker**. 2024. Disponível em: https://www.docker.com/. Acesso em: 20 nov. 2024.

ENCODE. **HTTPX**. 2024. Disponível em: https://www.python-httpx.org/. Acesso em: 10 set. 2025.

EXPRESS. **Express - Node.js web application framework**. 2024. Disponível em: https://expressjs.com/. Acesso em: 10 set. 2025.

GEEKSFORGEEKS. **What is Load Balancer?** 2025. Disponível em: https://www.geeksforgeeks.org/system-design/what-is-load-balancer-system-design/. Acesso em: 10 set. 2025.

GOOGLE . **What Are Containers?** 2024. Disponível em: https://cloud.google.com/learn/what-are-containers. Acesso em: 6 nov. 2024.

GOOGLE . **What is container orchestration?** 2024. Disponível em: https://cloud.google.com/discover/what-is-container-orchestration?hl=en. Acesso em: 18 nov. 2024.

GROUP, P. **Lightweight Kubernetes Distributions**. 2023. Disponível em: https://programming-group.com/assets/pdf/papers/2023_Lightweight-Kubernetes-Distributions.pdf. Acesso em: 20 nov. 2024.

IBM. **What is Cluster Computing?** 2024. Disponível em: https://www.ibm.com/think/topics/cluster-computing. Acesso em: 18 nov. 2024.

Jakob Nielsen. **Response Times: The Three Important Limits**. 2025. Disponível em: https://www.nngroup.com/articles/response-times-3-important-limits/. Acesso em: 16 out. 2025.

K0S PROJECT. **K0s Documentation**. 2024. Disponível em: https://k0sproject.io/. Acesso em: 20 nov. 2024.

MONTEIRO, E. R. *et al.* **DevOps**. SAGAH, 2021. 154 p. ISBN 9786556901725. Disponível em: https://app.minhabiblioteca.com.br/reader/books/9786556901725/. Acesso em: 6 nov. 2024.

MORSE, G. **How to turn your old hardware into a Kubernetes cluster**. 2023. Disponível em: https://learnfastmakethings.com/p/how-to-turn-your-old-hardware-into-a-kubernetes-cluster-129d17aa8704. Acesso em: 20 nov. 2024.

NODE.JS. **Node.js v22.19.0 Documentation**. 2024. Disponível em: https://nodejs.org/docs/latest-v22.x/api/documentation.html. Acesso em: 10 set. 2025.

OPENSSH. **OpenSSH: OpenSSH**. 2025. Disponível em: https://www.openssh.com/. Acesso em: 10 set. 2025.

PARAMIKO. **Welcome to Paramiko!** 2024. Disponível em: https://www.paramiko.org/. Acesso em: 10 set. 2025.

PYTHON SOFTWARE FOUNDATION. **Python Documentation**. 2025. Disponível em: https://docs.python.org/3/. Acesso em: 10 set. 2025.

Rancher Labs . **K3s Documentation**. 2024. Disponível em: https://k3s.io/. Acesso em: 20 nov. 2024.

SILVA, J. **Implementando um sistema de conteinerização com Kubernetes usando GitOps**. 2022. Trabalho de Conclusão de Curso (Curso de Ciência da Computação) – Universidade Federal de Pernambuco, Recife. Disponível em: https://www.cin.ufpe.br/~tg/2022-1/tg_CC/tg_pgrr.pdf. Acesso em: 20 nov. 2024.

SYSSTAT. **sysstat**. 2024. Disponível em: https://github.com/sysstat/sysstat. Acesso em: 10 set. 2025.

GLOSSÁRIO

stateless termo em inglês que significa "sem estado". Em computação, refere-se a sistemas ou aplicações que não mantêm informações sobre o estado ou contexto entre diferentes interações ou sessões. Cada requisição é tratada de forma independente, sem depender de dados armazenados de interações anteriores. 11

APÊNDICES

APÊNDICE A – *Script* de teste e aplicação de teste

Script de teste que implementa a metodologia de teste e aplicação de teste que implementa os algoritmos de teste podem ser acessados em: https://github.com/FaboBorgesLima/cluster-tester/releases/tag/v2