INTERFACE TECNOLÓGICA

# ANÁLISE DA ESCALABILIDADE DE APLICAÇÕES CONTAINERIZADAS EM KUBERNETES UTILIZANDO MINIKUBE

## *ANALYSIS OF THE SCALABILITY OF CONTAINERIZED APPLICATIONS IN KUBERNETES USING MINIKUBE*

Alexandre Betassa de Souza - alexandre.betassa.souza@hotmail.com
Faculdade de Tecnologia – FATEC – Taquaritinga- São Paulo- Brasil

Jederson Donizetie Luchi –jederson.zuchi@fatec.sp.gov.br
Faculdade de Tecnologia – FATEC – Taquaritinga- São Paulo- Brasil

DOI: 10.31510/infa.v22i2.2344
Data de submissão: 25/09/2025
Data do aceite: 01/12/2025
Data da publicação: 20/12/2025

**RESUMO**

O mercado tem observado o crescimento de aplicações baseadas em microsserviços, e a demanda por soluções escaláveis e resilientes impulsionou o uso de *containers* no desenvolvimento e implantação de sistemas. Nesse contexto, a orquestração de *containers* torna-se essencial para automatizar tarefas como escalonamento, balanceamento de carga, monitoramento e atualização de aplicações em ambientes distribuídos. Este estudo teve como objetivo investigar o conceito, funcionamento e benefícios da orquestração de *containers*, com foco em ferramentas modernas como Kubernetes. A metodologia combinou revisão de literatura e aplicação prática da tecnologia em uma aplicação demonstrativa. Os resultados evidenciam o crescimento do mercado global de orquestração de *containers* e a eficácia das ferramentas de *auto-scaling* na gestão eficiente de recursos e escalabilidade de aplicações.

**Palavras-chave**: Mercado. Crescimento. Gerenciamento. Kubernetes.

**ABSTRACT**

The market has witnessed the growth of microservices-based applications, and the demand for scalable and resilient solutions has driven the use of *containers* in system development and deployment. In this context, container orchestration becomes essential to automate tasks such as scaling, load balancing, monitoring, and updating applications in distributed environments. This study aimed to investigate the concept, operation, and benefits of container orchestration, focusing on modern tools such as Kubernetes. The methodology combined a literature review with a practical application of the technology in a demonstrative application. The results highlight the global growth of the container orchestration market and the effectiveness of auto-scaling tools in efficiently managing resources and ensuring application scalability.

**Keywords**: Market. Growth. Management. Kubernetes.

## 1 INTRODUÇÃO

Rosa e Mota (2021) explicam que o mundo vive há tempos a evolução das Tecnologias Digitais de Informação e Comunicação (TDICS), o que permitiu a sociedade ter acesso a infinitas informações. Diante desta constante evolução tecnológica, as organizações enfrentam desafios crescentes para manter aplicações eficientes e escaláveis. Ainda de acordo com os autores, muito se discute sobre a melhor forma de manter aplicações em funcionamento, sendo consideradas mais eficazes aquelas que atendem ao máximo de requisitos, com menores taxas de falha e maior excelência possível. Nesse contexto, a tecnologia de orquestração de *containers* tem se difundido, proporcionando maior portabilidade e escalabilidade das aplicações, além de garantir isolamento completo dos recursos essenciais à sua execução.

Segundo Freire (2021), em um mundo cada vez mais virtual torna-se difícil a escolha da melhor tecnologia a ser utilizada, visto que existem infinitas opções de programas, aplicações ou ferramentas dispostas no mercado. Dentro dessa realidade, é preciso que o consumidor dessa tecnologia realize um estudo de mercado para ver qual o *software* mais adequado para atender as suas necessidades.

O mercado global de orquestração de *containers* foi avaliado em aproximadamente US$ 0,69 bilhão em 2024, com projeção de atingir US$ 2,37 bilhões até 2033, registrando uma taxa de crescimento anual composta (CAGR) de 14,7% ao longo do período analisado. Esse avanço expressivo é impulsionado principalmente pela escalada na complexidade dos ambientes de aplicações modernas e pela crescente demanda por ferramentas robustas e escaláveis capazes de gerenciar cargas de trabalho containerizadas de forma eficiente, automatizada e resistente (Business Research Insights, 2024).

Compreender os fundamentos, a arquitetura e os benefícios da orquestração de *containers* é essencial para profissionais e estudantes de tecnologia. Este artigo investiga o conceito, funcionamento e vantagens da orquestração de *containers*, com foco no Kubernetes, destacando sua relevância na automação da implantação, gerenciamento e escalabilidade de aplicações. Os objetivos incluem explicar o papel dos *containers* no desenvolvimento moderno, apresentar ferramentas de orquestração e analisar suas vantagens em relação à gestão manual. O artigo justifica-se pela necessidade de organizar e disseminar conceitos, ferramentas e boas práticas da tecnologia.

## 2 FUNDAMENTAÇÃO TEÓRICA

O conteúdo a seguir apresenta os principais conceitos teóricos envolvidos no contexto desse trabalho.

### 2.1 Computação em Nuvem

De acordo com Albuquerque (2021), o conceito de computação em nuvem (*Cloud Computing*) surgiu em 2006, quando Eric Schmidt, então CEO do Google, descreveu em uma palestra a forma como a empresa gerenciava seus data centers. O termo refere-se a recursos computacionais disponibilizados pela internet, organizados em três categorias principais: Infraestrutura como Serviço (IaaS), Plataforma como Serviço (PaaS) e *Software* como Serviço (SaaS). Entre suas características destacam-se o modelo de pagamento baseado no consumo, a facilidade de aquisição pelo próprio usuário e o amplo acesso por meio da rede.

O modelo PaaS oferece uma infraestrutura altamente integrada, facilitando a implementação e o teste de aplicações (Albuquerque, 2021).

Já de acordo com Santos (2001), a plataforma SaaS consiste em um modelo no qual os sistemas são acessíveis via internet, permitindo que o usuário configure a aplicação sem a necessidade de gerenciar infraestrutura ou instalações.

Por fim, o modelo IaaS é responsável por fornecer recursos de *hardware*, como máquinas virtuais, *data centers* e redes (Santos, 2001).

### 2.2 Máquinas Virtuais

As Máquinas Virtuais ou *Virtual Machines* (VMs) são ambientes computacionais simulados que utilizam recursos de um computador físico, como CPU, memória e armazenamento, para rodar sistemas operacionais de forma isolada (Microsoft Azure, 2024).

Susnjara e Smalley (2024) descrevem que as VMs possibilitam a execução de diversos sistemas operacionais em um único *hardware* físico, promovendo melhorias ao uso de recursos como CPU, memória, rede e armazenamento. Essas máquinas, denominadas "convidadas", operam dentro de uma máquina "hospedeira" e incluem tecnologias como servidores virtuais, instâncias de servidor virtual VSIs e servidores virtuais privados VPSs .

O mercado global de VMs superou US$ 9,5 bilhões em 2023, equivalente a uma taxa de crescimento de 12% entre 2024 e 2032, impulsionado pela crescente adoção da

computação em nuvem. O crescimento reflete a demanda por maior escalabilidade, flexibilidade e redução de custos, com provedores de nuvem incorporando VMs e *containers* para melhorar a infraestrutura de TI (Sunsjara; Smalley, 2024).

## 2.3 Arquitetura de Microsserviços

De acordo com Rosa e Mota (2021), com o tempo e segundo diversos conhecimentos adquiridos e praticados, tornou-se cada vez mais comum a utilização de arquitetura de *software* baseada em serviços distribuídos, com o objetivo de reduzir o *down time* (período em que a máquina não está em processo operacional) durante a entrega da aplicação em ambiente produtivo.

Os microsserviços são definidos como uma abordagem arquitetônica e organizacional que fragmenta o desenvolvimento de software em pequenos serviços autônomos, comunicando-se por meio de APIs bem definidas. Essa estrutura facilita a escalabilidade, acelera o desenvolvimento de aplicativos e reduz o tempo de entrega de novos recursos ao mercado, promovendo a inovação. Em contraste, arquiteturas monolíticas apresentam processos altamente acoplados e funcionam como um único serviço, como consequência, quando há aumento na demanda de um processo específico, toda a aplicação precisa ser escalada. Além disso, à medida que a base de código cresce, torna-se mais complexa a adição ou aprimoramento de funcionalidades, o que dificulta a experimentação e a implementação de novas ideias (Rosa; Mota, 2021).

Ainda segundo a AWS (2024), embora a alta dependência entre processos possa ampliar o risco de falhas, a arquitetura de microsserviços assegura que cada componente opere de forma independente, oferecendo maior flexibilidade, resiliência e eficiência. Entre seus principais benefícios estão a agilidade das equipes, a escalabilidade e implantação isoladas, a liberdade tecnológica e a modularização do código, que favorecem a reutilização e o desenvolvimento de novas funcionalidades sem comprometer o sistema como um todo.

## 2.4 *Containers*

De acordo com Vitalino e Castro (2016), os *containers* encapsulam aplicações juntamente com suas dependências, compartilhando o *kernel* do sistema operacional do *host*. Ao contrário das máquinas virtuais, os *containers* são mais leves e possuem uma integração

mais eficiente com o sistema, proporcionando melhor desempenho devido ao gerenciamento unificado dos recursos.

A virtualização em nível de sistema operacional, conhecida como conteinerização, possibilita a criação de várias instâncias isoladas dentro do mesmo núcleo do sistema operacional. Este modelo representa uma evolução significativa em relação à virtualização tradicional (Carey, 2021).

Segundo Sunsjara e Smalley (2024) os *containers*, por serem mais leves e eficientes em relação às máquinas virtuais, tornaram-se a base das aplicações nativas em nuvem e arquiteturas *multi cloud.* Sua principal vantagem é a abstração, que garante portabilidade, rapidez na inicialização e escalabilidade horizontal. Como também carregam todas as dependências, podem ser construídos uma vez e reutilizados em diferentes ambientes.

## 2.5 Orquestração de *Containers*

De acordo com a Red Hat (2024), a orquestração de *containers* automatiza o gerenciamento, a implantação e a escalabilidade dos *containers* ao longo de seu ciclo de vida. Esse processo permite que a mesma aplicação seja executada em diferentes ambientes sem necessidade de redesenho, sendo especialmente relevante para empresas que gerenciam centenas ou milhares de *containers* e *hosts* Linux. Ela também beneficia equipes de DevOps, que a integram aos fluxos de trabalho de integração e entrega contínua (CI/CD).

Segundo o site da AWS (2024), as ferramentas de orquestração de *containers* têm o objetivo de simplificar o gerenciamento da infraestrutura, automatizando todo o ciclo de vida dos *containers*, desde o provisionamento e agendamento até a implantação e exclusão. Com isso, as organizações podem se beneficiar da conteinerização em grande escala, sem incorrer em despesas adicionais de manutenção.

Dentro da orquestração de *containers* merece destaque a plataforma Kubernetes. O Kubernetes é um produto *Open Source* (código aberto) que tem como finalidade a automação da implantação, o dimensionamento e o gerenciamento de aplicativos de *containers* (Kubernetes, 2020).

Desenvolvido originalmente pela Google e posteriormente doado à *Cloud Native Computing Foundation CNCF*, o Kubernetes tornou-se o padrão de fato para orquestração de *containers* em ambientes de produção (Red Hat, 2024).

## 3 METODOLOGIA

Este estudo foi conduzido por meio de uma pesquisa aplicada, com abordagem descritiva e exploratória. A escolha dessa metodologia se justifica pela necessidade de compreender, descrever e demonstrar o funcionamento da orquestração de *containers* no contexto da computação em nuvem, com foco na ferramenta Kubernetes.

Segundo Gil (2008), a pesquisa exploratória tem como objetivo trazer maior familiaridade com o problema, tornando-o mais evidente, contribuindo para a formulação de hipóteses ou o aprimoramento de ideias. A pesquisa descritiva tem como finalidade observar, registrar, analisar e correlacionar fatos ou fenômenos, descrevendo suas características de forma precisa.

Como instrumento técnico complementar, realizou-se uma demonstração prática de orquestração de *containers* com Kubernetes, utilizando um cluster local via Minikube. Nessa simulação, uma aplicação foi implantada para ilustrar o processo de orquestração, abordando escalabilidade e gerenciamento de *pods*, além de evidenciar os benefícios e desafios da tecnologia, relacionando teoria e prática.

A metodologia experimental seguiu as seguintes etapas: configuração do ambiente, incluindo instalação do WSL, Docker, Kubernetes e Minikube; desenvolvimento de uma aplicação em NET 8 com *endpoints* para geração de carga de CPU; implementação de métricas por meio do *metrics-server*; configuração do HPA com políticas de escalonamento horizontal baseadas em CPU; execução de testes simulando cargas de trabalho e observação do comportamento do sistema; e análise dos resultados, com coleta e interpretação dos dados obtidos durante os experimentos.

Para o estudo, foi utilizado o Kubernetes, sistema de orquestração de *containers* que automatiza implantação, escalonamento e gerenciamento de aplicações em *clusters* distribuídos. O ambiente local foi configurado com o Minikube, ferramenta que cria e gerencia um cluster Kubernetes em uma única máquina, facilitando testes e experimentos.

A execução ocorreu sobre o WSL (*Windows Subsystem for Linux*), que possibilita rodar um *kernel* Linux completo no Windows, garantindo compatibilidade com ferramentas nativas, como o kubectl, utilizado para gerenciamento do cluster por meio de comandos de criação, atualização e inspeção de recursos. Para monitoramento visual, empregou-se o k9s, que oferece uma interface interativa para acompanhar o estado dos recursos em tempo real.

A aplicação, desenvolvida em C# .NET 8, possui *endpoints* para gerar carga de CPU, permitindo que o *Horizontal Pod Autoscaler* (HPA) escalasse automaticamente conforme aumentava a demanda de processamento. O HPA, recurso do Kubernetes, ajusta dinamicamente o número de *pods* de um *deployment* ou *replicaSet* com base em métricas observáveis, garantindo o uso eficiente dos recursos.

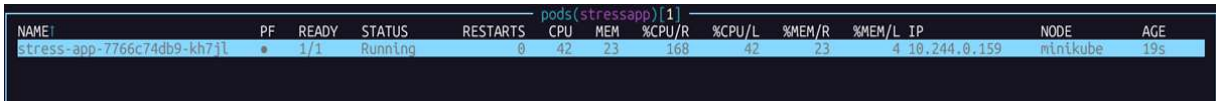
Para esta demonstração, o HPA foi configurado com mínimo de 1 pod e máximo de 25, monitorando a CPU com alvo de 60% de utilização. As políticas de comportamento incluíram *scale-up* sem janela de estabilização e *scale-down* com janela de 5 segundos, assegurando resposta rápida às variações de carga. Embora o HPA também possa monitorar memória, o foco principal foi CPU, por ser mais facilmente controlável nos testes. Para seu funcionamento, foi necessário instalar o *metrics-server*, componente do Kubernetes que coleta métricas de CPU e memória em tempo real, permitindo ao HPA ajustar automaticamente o número de *pods* conforme a utilização dos recursos.

## 4 RESULTADOS E DISCUSSÃO

### 4.1 Estado Inicial do Sistema

Ao iniciar a aplicação, observou-se na Figura 1 que ela foi implantada com apenas um pod, a menor unidade de execução do Kubernetes responsável por hospedar a aplicação. Esse cenário inicial serviu como referência para avaliar o comportamento da aplicação e a escalabilidade horizontal promovida pelo HPA durante a demonstração.

**Figura 1 – Quantidade de *pods* na inicialização**




**Fonte: Autor**

O HPA, associado ao *deployment*, utiliza métricas de CPU e memória para determinar a necessidade de aumentar ou reduzir o número de pods. Na observação inicial, a CPU estava em 4% de um alvo de 60% e a memória em 30% de utilização, indicando baixa carga sobre os recursos do pod, conforme mostrado na Figura 2. O número atual de réplicas era 1, correspondendo ao pod inicial implantado. Essa leitura permite compreender que, enquanto os

recursos utilizados estiverem abaixo dos limites definidos, o HPA não realizará escalonamento.

**Figura 2 – Métricas do HPA no momento da inicialização**

```
NAME            REFERENCE                TARGETS                       MINPODS   MAXPODS   REPLICAS   AGE
stress-app-hpa   Deployment/stress-app   cpu: 4%/90%, memory: 30%/60%   1         25        1          3m39s
```

**Fonte: Autor**

## 4.2 Processo de Escalonamento Horizontal

Quando a aplicação recebeu uma chamada para gerar carga, o HPA identificou o aumento da demanda, que ao atingir 67% de uso de CPU (Figura 3), acima do alvo de 60%, iniciou a criação de *pods* adicionais, aumentando rapidamente as réplicas de 1 para 11, demonstrando a eficiência do *auto-scaling*.

**Figura 3 – Métricas do HPA da aplicação com carga**

```
NAME            REFERENCE                TARGETS                       MINPODS   MAXPODS   REPLICAS   AGE
stress-app-hpa   Deployment/stress-app   cpu: 67%/60%, memory: 27%/60%   1         25        11         2m54s
```

**Fonte: Autor**

Na Figura 4, são exibidos os eventos de *SuccessfullRescale*, que refletem a resposta do HPA ao aumento da utilização de CPU. Conforme a CPU ultrapassava o limite definido, ele aumentou gradualmente o número de pods, inicialmente de 1 para 2, depois para 4, 7, 9, 11 e, finalmente, 13, atendendo à demanda crescente.

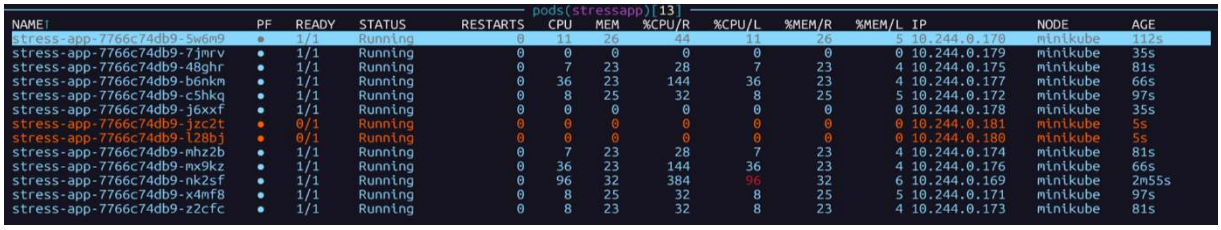
**Figura 4 – Eventos de escalonamento do HPA**

```
Events:
  Type     Reason                        Age                  From                       Message
  ----     ------                        ----                 ----                       -------
  Warning  FailedGetResourceMetric       3m46s (x2 over 4m1s)  horizontal-pod-autoscaler  failed to get cpu utilization: unable to get metrics for resource
cpu: no metrics returned from resource metrics API
  Warning  FailedGetResourceMetric       3m46s (x2 over 4m1s)  horizontal-pod-autoscaler  failed to get memory utilization: unable to get metrics for resour
ce memory: no metrics returned from resource metrics API
  Warning  FailedComputeMetricsReplicas  3m46s (x2 over 4m1s)  horizontal-pod-autoscaler  invalid metrics (2 invalid out of 2), first error is: failed to ge
t cpu resource metric value: failed to get cpu utilization: unable to get metrics for resource cpu: no metrics returned from resource metrics API
  Normal   SuccessfulRescale             3m                   horizontal-pod-autoscaler  New size: 2; reason: cpu resource utilization (percentage of reque
st) above target
  Normal   SuccessfulRescale             2m45s                horizontal-pod-autoscaler  New size: 4; reason: cpu resource utilization (percentage of reque
st) above target
  Normal   SuccessfulRescale             2m29s                horizontal-pod-autoscaler  New size: 7; reason: cpu resource utilization (percentage of reque
st) above target
  Normal   SuccessfulRescale             2m14s                horizontal-pod-autoscaler  New size: 9; reason: cpu resource utilization (percentage of reque
st) above target
  Normal   SuccessfulRescale             103s                 horizontal-pod-autoscaler  New size: 11; reason: cpu resource utilization (percentage of requ
est) above target
  Normal   SuccessfulRescale             73s                  horizontal-pod-autoscaler  New size: 13; reason: cpu resource utilization (percentage of requ
est) above target
```

**Fonte: Autor**

Cada incremento ocorreu automaticamente, distribuindo a carga entre os *pods* e mantendo desempenho e disponibilidade. Na Figura 5, é possível observar o número de *pods* em execução e os recursos utilizados em cada um. Essa sequência evidencia o funcionamento dinâmico do HPA, que aumenta automaticamente os *pods* quando os recursos monitorados ultrapassam os limites configurados, demonstrando a escalabilidade horizontal típica do Kubernetes.

**Figura 5 – Quantidade de pods após escalonamento**



pods(stressapp)[13]

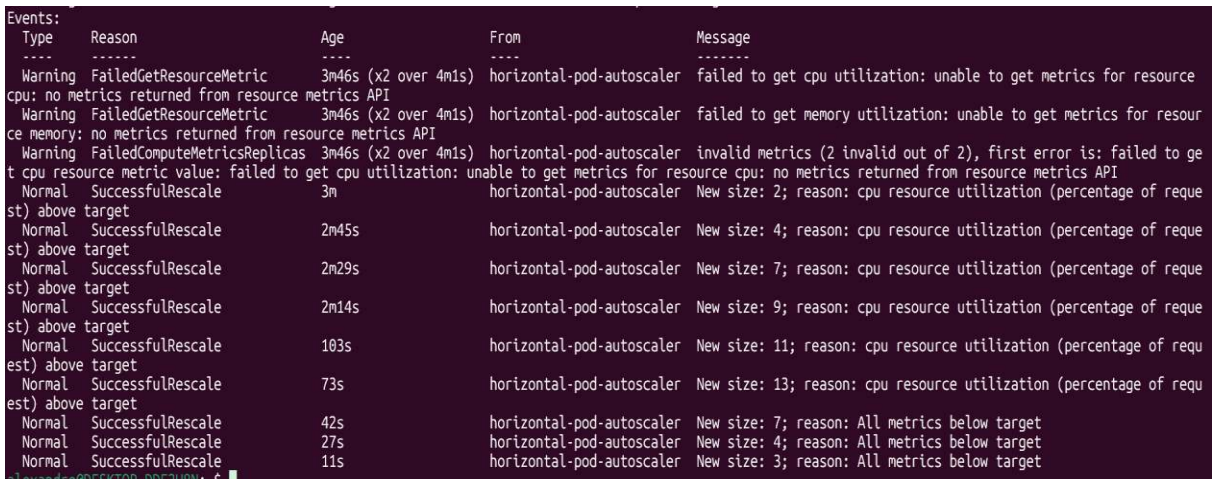
| NAME↑ | PF | READY | STATUS | RESTARTS | CPU | MEM | %CPU/R | %CPU/L | %MEM/R | %MEM/L | IP | NODE | AGE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| stress-app-7766c74db9-5w6m9 | • | 1/1 | Running | 0 | 11 | 26 | 44 | 11 | 26 | 5 | 10.244.0.170 | minikube | 112s |
| stress-app-7766c74db9-7jmrv | • | 1/1 | Running | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10.244.0.179 | minikube | 35s |
| stress-app-7766c74db9-48ghr | • | 1/1 | Running | 0 | 7 | 23 | 28 | 7 | 23 | 4 | 10.244.0.175 | minikube | 81s |
| stress-app-7766c74db9-b6nkm | • | 1/1 | Running | 0 | 36 | 23 | 144 | 36 | 23 | 4 | 10.244.0.177 | minikube | 66s |
| stress-app-7766c74db9-c5hkq | • | 1/1 | Running | 0 | 8 | 25 | 32 | 8 | 25 | 5 | 10.244.0.172 | minikube | 97s |
| stress-app-7766c74db9-j6xxf | • | 1/1 | Running | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10.244.0.178 | minikube | 35s |
| stress-app-7766c74db9-jzc2t | • | 0/1 | Running | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10.244.0.181 | minikube | 5s |
| stress-app-7766c74db9-l28bj | • | 0/1 | Running | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10.244.0.180 | minikube | 5s |
| stress-app-7766c74db9-mhz2b | • | 1/1 | Running | 0 | 7 | 23 | 28 | 7 | 23 | 4 | 10.244.0.174 | minikube | 81s |
| stress-app-7766c74db9-mx9kz | • | 1/1 | Running | 0 | 36 | 23 | 144 | 36 | 23 | 4 | 10.244.0.176 | minikube | 66s |
| stress-app-7766c74db9-nk2sf | • | 1/1 | Running | 0 | 96 | 32 | 384 | 96 | 32 | 6 | 10.244.0.169 | minikube | 2m55s |
| stress-app-7766c74db9-x4mf8 | • | 1/1 | Running | 0 | 8 | 25 | 32 | 8 | 25 | 5 | 10.244.0.171 | minikube | 97s |
| stress-app-7766c74db9-z2cfc | • | 1/1 | Running | 0 | 8 | 23 | 32 | 8 | 23 | 4 | 10.244.0.173 | minikube | 81s |

**Fonte: Autor**

## 4.3 Processo de Desescalonamento

Após o pico de utilização de CPU, a carga na aplicação começou a diminuir e o HPA, que monitora constantemente CPU e memória, identificou que os valores estavam abaixo dos limites configurados iniciando a redução gradual do número de pods. Conforme mostrado na Figura 6, ele primeiro diminuiu as réplicas de 13 para 7, respondendo rapidamente à queda de carga e a medida que a demanda continuou a cair, o número de pods foi ajustado para 4 e, por fim, 3, mantendo apenas o necessário para sustentar a aplicação de forma eficiente.

**Figura 6 – Eventos do desescalonamento do HPA**



```
Events:
  Type     Reason                   Age                From                       Message
  ----     ------                   ----               ----                       -------
  Warning  FailedGetResourceMetric  3m46s (x2 over 4m1s)  horizontal-pod-autoscaler  failed to get cpu utilization: unable to get metrics for resource
cpu: no metrics returned from resource metrics API
  Warning  FailedGetResourceMetric  3m46s (x2 over 4m1s)  horizontal-pod-autoscaler  failed to get memory utilization: unable to get metrics for resour
ce memory: no metrics returned from resource metrics API
  Warning  FailedComputeMetricsReplicas  3m46s (x2 over 4m1s)  horizontal-pod-autoscaler  invalid metrics (2 invalid out of 2), first error is: failed to ge
t cpu resource metric value: failed to get cpu utilization: unable to get metrics for resource cpu: no metrics returned from resource metrics API
  Normal   SuccessfulRescale        3m                 horizontal-pod-autoscaler  New size: 2; reason: cpu resource utilization (percentage of reque
st) above target
  Normal   SuccessfulRescale        2m45s              horizontal-pod-autoscaler  New size: 4; reason: cpu resource utilization (percentage of reque
st) above target
  Normal   SuccessfulRescale        2m29s              horizontal-pod-autoscaler  New size: 7; reason: cpu resource utilization (percentage of reque
st) above target
  Normal   SuccessfulRescale        2m14s              horizontal-pod-autoscaler  New size: 9; reason: cpu resource utilization (percentage of reque
st) above target
  Normal   SuccessfulRescale        103s               horizontal-pod-autoscaler  New size: 11; reason: cpu resource utilization (percentage of requ
est) above target
  Normal   SuccessfulRescale        73s                horizontal-pod-autoscaler  New size: 13; reason: cpu resource utilization (percentage of requ
est) above target
  Normal   SuccessfulRescale        42s                horizontal-pod-autoscaler  New size: 7; reason: All metrics below target
  Normal   SuccessfulRescale        27s                horizontal-pod-autoscaler  New size: 4; reason: All metrics below target
  Normal   SuccessfulRescale        11s                horizontal-pod-autoscaler  New size: 3; reason: All metrics below target
```

**Fonte: Autor**

Esse comportamento mostra como o HPA consegue otimizar os recursos do cluster, reduzindo os pods de forma gradual e evitando desperdício, sem comprometer a estabilidade ou a disponibilidade da aplicação. Dessa maneira, mesmo durante períodos de menor demanda, a aplicação continua funcionando de forma confiável.

### 4.4 Análise dos Mecanismos de Monitoramento

A demonstração evidenciou o funcionamento do HPA no Kubernetes, mostrando como ele ajusta automaticamente o número de pods conforme a carga de CPU da aplicação, aumentando gradualmente as réplicas durante picos de uso e reduzindo-as quando a demanda diminui. Além disso, o Kubernetes conta com mecanismos adicionais de monitoramento, como *probes* e agentes nos nós do *cluster*, como o *kubelet*, que coleta métricas de CPU, memória e outros recursos, permitindo que o HPA e ferramentas de observabilidade acompanhem continuamente o desempenho da aplicação e do *cluster*, que não foram abordados nesse trabalho.

Os resultados mostram a eficácia do Kubernetes como plataforma de orquestração de *containers*, especialmente no escalonamento horizontal. O sistema respondeu rapidamente às variações de carga, otimizando recursos por meio do HPA, mantendo equilíbrio entre desempenho e utilização eficiente da infraestrutura. Observou-se também alta disponibilidade, com a aplicação estável durante todo o processo, e escalonamento automático sem intervenção manual.

Em termos práticos, os achados indicam que o Kubernetes é uma solução robusta para orquestração de *containers*, especialmente em aplicações com variações significativas de carga. O escalonamento automático demonstrado contribui para economia de recursos e melhora a experiência do usuário, consolidando-se como prática recomendada em arquiteturas modernas.

## 5 CONSIDERAÇÕES FINAIS

A orquestração de *containers* configura-se como um processo essencial de automatização da rede e do gerenciamento dos *containers* para que possam ser implantadas

aplicações em grandes escalas. Este estudo conseguiu demonstrar, tanto teoricamente quanto praticamente, a importância e eficácia dessa tecnologia no cenário atual de desenvolvimento de *software*.

De acordo com todo o fundamento evidenciado neste estudo, é possível afirmar que o objetivo proposto foi alcançado. A demonstração trazida no tópico resultados e discussão deixa evidente o funcionamento do HPA no Kubernetes, com resposta dinâmica, visto que as réplicas foram aumentadas de forma gradual no período onde a utilização foi maior e a redução aconteceu proporcionalmente quando a demanda sofreu queda. A combinação de escalonamento automático evidencia a capacidade do Kubernetes de gerenciar recursos de forma eficiente, garantindo disponibilidade, desempenho e escalabilidade horizontal da aplicação em tempo real.

As principais contribuições deste trabalho incluem a apresentação conceitual de forma organizada dos fundamentos da orquestração de *containers*; a demonstração prática, por meio da aplicação experimental dos conceitos teóricos; a análise crítica, destacando vantagens da tecnologia; e a orientação técnica, oferecendo diretrizes para a implementação de soluções semelhantes.

É importante destacar algumas limitações deste estudo: o ambiente experimental restringiu-se a um *cluster* local via Minikube; a análise concentrou-se principalmente em métricas de CPU, sem explorar outras possibilidades; não foram feitas comparações com outras ferramentas de orquestração; e o período de observação foi relativamente curto.

Com base nos resultados obtidos e nas limitações identificadas, sugerem-se as seguintes direções para trabalhos futuros: comparação entre diferentes orquestradores; estudos de desempenho em ambientes de produção de grande escala; análise de custos operacionais da orquestração de *containers*; investigação de métricas customizadas para escalonamento.

A orquestração de *containers*, especialmente através do Kubernetes, representa uma evolução significativa na forma como desenvolvemos, implantamos e gerenciamos aplicações modernas, oferecendo não apenas eficiência operacional, mas também a base para arquiteturas mais resilientes e escaláveis.

## REFERÊNCIAS

ALBUQUERQUE, F. S. **Computação em Nuvem**: Conceitos, Tecnologias e Desafios. Revista de Tecnologia da Informação, v. 15, n. 2, p. 45 62, 2021.

AWS. **O que são microsserviços?** Disponível em: https://aws.amazon.com/pt/microservices/. Acesso em: 1 mar. 2025.

BUSINESS RESEARCH INSIGHTS. Container Orchestration Market Size, Share, Growth, and Industry Analysis, By Type Cloud, On-Premises), By Application BFSI, Healthcare, Government, Others), **Regional Forecast** 2024 2033. Business Research Insights, 2024. Disponível em: https://www.businessresearchinsights.com/pt/market-reports/container-orchestration-market-112752. Acesso em: 1 mar. 2025.

CAREY, M. Understanding Container Virtualization. **Tech Publishers,** 2021.
FREIRE, J. E. L. **Orquestração de *Containers* Usando Kubernetes e Docker Swarm**. 2021. Disponível em: https://ubibliorum.ubi.pt/bitstream/10400.6/11091/1/7913_17373.pdf. Acesso em: 2 mar. 2025.

GIL, A. C. **Como elaborar projetos de pesquisa**. 4. ed. São Paulo: Atlas, 2008.

IBM. **Máquinas virtuais VMs** . 2024. Disponível em: https://www.ibm.com/br-pt/topics/virtual-machines. Acesso em: 18 mar. 2025.

_____. **O que são *containers*?** 2024. Disponível em: https://www.ibm.com/br-pt/topics/*containers*. Acesso em: 1 mar. 2025.

KUBERNETES. **Configure Liveness, Readiness and Startup Probes**. Disponível em: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/. Acesso em: 30 ago. 2025.

_____. **Horizontal Pod Autoscaler**. Disponível em: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale. Acesso em: 30 ago. 2025.

_____. **Orquestração de *containers* prontos para produção**. 2020. Disponível em: https://kubernetes.io/pt/. Acesso em: 20 maio 2025.

MICROSOFT AZURE. **O que é uma máquina virtual VM ?** 2025. Disponível em: https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-a-virtual-machine. Acesso em: 18 mar. 2025.

OLIVEIRA, R. S.; LIMA, P. C. **Arquiteturas de Microsserviços**: Implementação e Melhores Práticas. São Paulo: Novatec, 2021.

RED HAT. **O que é orquestração de *containers*?** 2022. Disponível em: https://www.redhat.com/pt-br/topics/*containers*/what-is-container-orchestration. Acesso em: 20 mai 2025.

ROSA, J. A. C.; MOTA, J. dos R. Utilização e orquestração de *containers* em aplicações web. **Revista do Fórum Gerencial**, v. 1, n. 2, p. 78 95, 2021.

SANTOS, M. A. **Computação em Nuvem**: Fundamentos e Aplicações. Rio de Janeiro: Brasport, 2001.

SUSNJARA, K.; SMALLEY, J. **O que é uma Máquina Vitual?** 2024. Disponivel em: https://www.ibm.com/br-pt/think/topics/virtual-machines. Acesso em: 2 mai. 2025.

VITALINO, J. F.; CASTRO, M. A. S. **Virtualização e *Containers***: Tecnologias para Computação em Nuvem. São Paulo: Novatec, 2016.