Análise de Auto-Scaling em Plataformas Serverless no Kubernetes

Maria Isadora G. Martins de Oliveira[1], **Michel Sales Bonfim**[1], **Emanuel F. Coutinho**[1], **Arthur de Castro Callado**[1], **Enyo José T. Gonçalves**[1]

[1]Programa de Pós-Graduação em Computação (PCOMP)
Universidade Federal do Ceará (UFC)
Quixadá – CE – Brasil

isadoramartins@alu.ufc.br,

{michelsb, emanuel.coutinho, arthur, enyo}@ufc.br

Abstract. *This study conducts a comparative analysis of the performance of the Knative and Apache OpenWhisk serverless platforms in a MicroK8s environment, focusing on the efficiency of auto-scaling. The platforms were subjected to load tests generated by an expanded benchmark tool based on hey, collecting detailed metrics such as CPU usage, memory, number of pods, latency, and requests per second. In parallel, a second experiment focuses exclusively on the behavior of Knative under three distinct load levels: 10, 50, and 100 concurrent requests. The data were analyzed in Google Colab using Python, with a total of 30 executions per scenario, ensuring robustness in the comparison of results. This study provides an in-depth analysis of auto-scaling and the performance of serverless platforms under different load conditions.*

Resumo. *Este estudo realiza uma análise comparativa do desempenho de plataformas serverless Knative e Apache OpenWhisk em ambiente MicroK8s, com foco na eficiência do auto-scaling. As plataformas foram submetidas a testes de carga gerados por uma ferramenta de benchmark expandida, baseada no hey, coletando métricas detalhadas como uso de CPU, memória, número de pods, latência e requisições por segundo. Em paralelo, um segundo experimento foca exclusivamente no comportamento do Knative sob três níveis de carga distintos, 10, 50 e 100 requisições concorrentes. Os dados foram analisados no Google Colab em Python, ao todo, foram realizadas 30 execuções por cenário, garantindo robustez na comparação dos resultados. Esse estudo fornece uma análise aprofundada sobre o auto-scaling e o desempenho das plataformas serverless em diferentes contextos de carga.*

1. Introdução

A computação *serverless* representa uma tendência recente na tecnologia, oferecendo elasticidade transparente como uma evolução natural da computação em nuvem. Essa abordagem permite que os desenvolvedores se concentrem exclusivamente na criação de funções, transferindo a responsabilidade pela gestão da infraestrutura para o provedor de serviços [Sousa 2020]. Aplicações baseadas em computação *serverless* são orientadas por eventos e executam funções apenas quando acionadas por esses eventos, como uma solicitação HTTP. Isso garante que os recursos de computação sejam utilizados de forma

eficiente, sendo provisionados apenas conforme necessário, o que resulta em custos diretamente proporcionais ao uso [Tari et al. 2024].

Uma característica fundamental dessa abordagem é o *auto-scaling*, que ajusta os recursos de computação conforme as demandas da carga de trabalho [Tari et al. 2024]. Quando a carga aumenta, novos recursos são alocados automaticamente, e, ao diminuir, são liberados. Isso elimina a necessidade de gerenciar recursos manualmente, permitindo que os desenvolvedores foquem no código, enquanto a infraestrutura se adapta dinamicamente às necessidades do aplicativo, otimizando o uso e reduzindo custos [Tari et al. 2024].

No contexto da computação *serverless*, onde a rapidez da inicialização de funções e a eficiência na escala-para-zero são fundamentais, a complexidade do Kubernetes pode impactar negativamente o tempo de inicialização dos *containers* e a experiência do usuário final. Para mitigar essas limitações, surgiram distribuições mais leves, como K3s[1] e MicroK8s[2], que eliminam componentes desnecessários e otimizam a implantação em infraestruturas bare-metal e de borda [Kjorveziroski and Filiposka 2022]. De acordo com [Koziolek and Eskandani 2023], o MicroK8s foi a primeira distribuição leve do Kubernetes, lançada em 2018. Desenvolvido para atender a ambientes de desenvolvimento e produção que necessitam de alta disponibilidade e baixo consumo de recursos. Sendo assim, essa será a distribuição usada no ambiente para execução das plataformas *serverless*.

Diferentes plataformas *serverless* oferecem mecanismos próprios de escalabilidade, permitindo que instâncias sejam criadas e removidas dinamicamente. No entanto, o comportamento desses *frameworks* em um ambiente Kubernetes, como o MicroK8s, ainda é pouco investigado, gerando incertezas sobre qual delas proporciona o melhor equilíbrio entre desempenho e eficiência. Para suprir essa lacuna, este estudo compara o desempenho do Knative[3] e do OpenWhisk[4], em relação ao *auto-scaling*, identificando suas limitações e vantagens. Após essa análise comparativa, o trabalho aprofunda a investigação no *framework* que se destacou na primeira análise, avaliando seu desempenho sob diferentes cargas. Para a realização da coleta das métricas, foi desenvolvida uma extensão do *benchmark* hey[5], que permite coletar métricas detalhadas, como uso de CPU, memória e número de pods ativos, fornecendo uma visão abrangente sobre a eficiência do *auto-scaling* no Knative.

Este trabalho apresenta como principais contribuições: (i) a comparação entre as plataformas serverless Knative e OpenWhisk em ambiente MicroK8s; (ii) o desenvolvimento de uma versão estendida da ferramenta de *benchmark* hey; (iii) a análise detalhada do desempenho do Knative sob diferentes cargas nesse ambiente; e (iv) o avanço do conhecimento na área de computação *serverless*, com ênfase na análise de *auto-scaling* e no comportamento dessas plataformas em distribuições leves do Kubernetes.

[1]https://k3s.io/
[2]https://microk8s.io/
[3]https://knative.dev/docs/
[4]https://openwhisk.apache.org/
[5]https://github.com/rakyll/hey

2. Fundamentação Teórica

A computação *serverless* é um paradigma emergente e promissor na computação em nuvem, cuja principal vantagem é liberar os desenvolvedores da atividade de gerenciar servidores, tarefas que são complexas e suscetíveis a erros. Ao abstrair essas responsabilidades, a computação *serverless* permite que os desenvolvedores se concentrem exclusivamente no desenvolvimento de suas aplicações, com acesso eficiente aos recursos necessários. Além disso, a computação *serverless* oferece *auto-scaling*, alocando recursos conforme a demanda, o que garante um uso otimizado e eficiente da infraestrutura [Wen et al. 2023].

O *auto-scaling* é um componente essencial da computação *serverless*, permitindo que a infraestrutura se ajuste dinamicamente às mudanças na carga de trabalho sem que haja necessidade de uma intervenção manual. Esse processo adapta os recursos computacionais conforme a demanda, ou seja, expande os recursos quando a carga aumenta e reduz quando a demanda diminui. Dessa forma, os desenvolvedores utilizam apenas os recursos necessários, otimizando o consumo e a eficiência [Tari et al. 2024].

Atualmente, os principais provedores de serviços em nuvem oferecem plataformas de computação *serverless*, mas essas plataformas geralmente exigem que as funções sejam escritas em formatos específicos, resultando em um bloqueio ao fornecedor. Em resposta a essa limitação, diversas infraestruturas privadas começaram a desenvolver suas próprias soluções para a execução de computação *serverless*, oferecendo maior flexibilidade e controle [Palade et al. 2019]. Alguns dos *frameworks serverless* de código aberto são: o Knative e o Apache OpenWhisk.

O Knative é um *framework* construído sobre o Kubernetes que simplifica o desenvolvimento de aplicações *serverless*, permitindo que os desenvolvedores foquem nas funcionalidades essenciais. Ele aproveita o poder do Kubernetes e componentes de código aberto para criar uma plataforma portátil, que facilita a execução de aplicações em *containers*, seja localmente ou em diferentes provedores de nuvem. Além disso, o Knative automatiza processos como a construção de *containers*, o roteamento de tráfego e a escalabilidade automática, incluindo a capacidade de reduzir a aplicação a zero quando não há demanda [Pereira 2019].

O Apache OpenWhisk é um *framework* que opera com três elementos principais: ações, gatilhos e regras. Ações são funções sem estado, gatilhos representam eventos e regras conectam gatilhos a ações. A plataforma permite combinar ações de diferentes linguagens em sequências para criar pipelines de processamento. Os componentes centrais incluem um servidor Nginx como proxy reverso, um controlador que autentica e roteia solicitações, um Apache Kafka que gerencia a comunicação entre o controlador e os Invokers, e um CouchDB que armazena dados e definições. As ações são executadas em contêineres Docker, e os resultados são armazenados no CouchDB [Palade et al. 2019].

3. Trabalhos Relacionados

Os trabalhos analisados apresentam diferentes abordagens para a avaliação de plataformas *serverless*, cada um com um foco específico. O estudo de [Li et al. 2021] investigou plataformas *serverless* de código aberto, como Knative, Kubeless, Nuclio e OpenFaaS, analisando as características e avaliando o desempenho dessas plataformas. De maneira

semelhante, o trabalho de [Mohanty et al. 2018] avaliou o tempo de resposta e a taxa de sucesso das plataformas Fission, Kubeless e OpenFaaS, utilizando o Google Kubernetes Engine (GKE) como ambiente de testes.

Já no estudo de [Copik et al. 2021], foi desenvolvido o SeBS, um *benchmark* abrangente para plataformas FaaS em provedores de nuvem, com métricas detalhadas, como tempo de execução, uso de CPU e desempenho de I/O. Por outro lado, o EdgeFaaS-Bench, proposto por [Rajput et al. 2022], analisou a execução de funções *serverless* em dispositivos de borda, utilizando hardware como o Raspberry Pi 4B e o Jetson Nano, explorando o impacto desses dispositivos no desempenho do OpenFaaS com Docker Swarm.

Diferentemente dos demais, este trabalho foca especificamente na análise do *auto-scaling* das plataformas Knative e OpenWhisk em um ambiente MicroK8s, explorando uma distribuição leve do Kubernetes e incorporando a avaliação de um *framework* adicional, o OpenWhisk. Além disso, adotou-se uma ferramenta distinta de *benchmark* para a coleta de métricas, que também se diferenciam em relação as coletadas em estudos anteriores. Com isso, buscou-se oferecer uma contribuição relevante para a compreensão da escalabilidade e do desempenho dessas plataformas *serverless* no MicroK8s.

4. Metodologia

Este trabalho segue uma metodologia estruturada que inclui: seleção das plataformas, configuração do ambiente de testes, definição dos experimentos e métricas, implementação da ferramenta de *benchmark* e, por fim, uma seção de execução e análise dos testes em duas etapas experimentais distintas.

4.1. Seleção das Plataformas

Para a condução do estudo, foram definidas as plataformas Knative e Apache OpenWhisk, plataformas ativas no mercado *open-source*, além de apresentarem abordagens distintas e serem suportadas em ambientes Kubernetes. Ambas as plataformas *serverless* oferecem mecanismos de *auto-scaling*, característica essencial para os objetivos deste trabalho.

Como base de orquestração, optou-se pelo uso do MicroK8s, uma distribuição leve e otimizada do Kubernetes, capaz de facilitar a implantação das plataformas. O MicroK8s foi adequado para validar o comportamento dos *frameworks serverless* em um ambiente controlado, com foco na análise de desempenho e escalabilidade.

4.2. Ambiente de Execução

A Figura 1 apresenta a configuração do ambiente experimental, composto por duas máquinas. A primeira, com 24 núcleos de CPU e 32GB de RAM, hospedou o MicroK8s com as plataformas Knative, OpenWhisk e a função *serverless*. A segunda, um notebook ASUS com 8GB de RAM e SSD de 256GB, foi utilizada para os testes de carga, empregando uma versão aprimorada da ferramenta hey para geração de requisições e coleta de métricas.

4.3. Definição dos Experimentos e Métricas

O estudo teve como objetivo avaliar o desempenho do mecanismo de *auto-scaling* das plataformas *serverless* Knative e Apache OpenWhisk, implantadas em um cluster Kubernetes com MicroK8s. A partir dessa análise inicial, foi realizado um segundo conjunto de

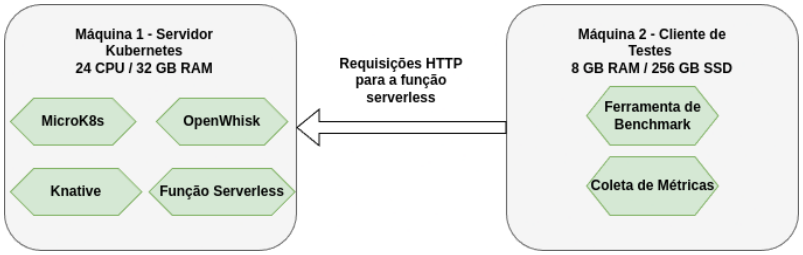


Figura 1. Ambiente de Execução

experimentos, voltado exclusivamente para a plataforma com melhor desempenho, considerando diferentes cargas de trabalho. Com base na abordagem de [Mohanty et al. 2018], que utilizou cinco níveis de carga, este estudo adotou três níveis de cargas, 10, 50 e 100 conexões concorrentes, correspondendo a situações de baixa, média e alta demanda. Essa variação possibilitou uma análise mais aprofundada sobre a escalabilidade e a alocação de recursos pela plataforma selecionada.

A análise de *auto-scaling* das plataformas *serverless* é baseada em métricas essenciais, como os percentis da latência, requisições por segundo, uso de CPU e memória, e número de pods, que fornecem uma visão detalhada do *auto-scaling* e do desempenho da plataforma sob diferentes níveis de carga. Essas métricas permitem avaliar como a plataforma responde à variação na demanda e ajusta seus recursos de forma eficiente.

4.4. Implementação do *benchmark*

Para comparar o desempenho das plataformas *serverless* e compreender o comportamento de uma plataforma sob condições variadas de concorrência, é essencial realizar *benchmarking*, processo que consiste na medição sistemática de métricas de desempenho entre diferentes configurações ou plataformas, permitindo identificar quais apresentam melhor eficiência. Além disso, o *benchmarking* possibilita a definição de parâmetros específicos para cada experimento, garantindo que os resultados sejam comparáveis e úteis para otimizar o desempenho das soluções analisadas [d Moura 2020].

A ferramenta aprimorada no estudo, baseada no hey, foi extendida para fornecer métricas detalhadas na avaliação do *auto-scaling* em plataformas *serverless*. Além das métricas de latência e requisições por segundo já coletadas, agora também é possível coletar dados sobre uso de CPU, memória e número de pods. Para viabilizar essa coleta adicional, a ferramenta foi integrada ao Netdata, responsável por monitorar o ambiente MicroK8s. Essas melhorias permitem uma análise mais precisa do desempenho, facilitando a avaliação da eficiência do *auto-scaling* no Knative. A implementação da ferramenta de *benchmark* ocorreu na máquina 2, como mostra na Figura 1.

5. Execução dos Experimentos e Análise dos Resultados

5.1. Comparação do Desempenho do Knative e Openwhisk

A primeira etapa do estudo consistiu na execução da função *serverless* na máquina 1, enquanto a máquina 2 foi responsável pela aplicação dos testes de carga. Para isso, foi utilizada a ferramenta de *benchmark* expandida com os parâmetros -z 60s -c 100 -q 100, simulando 100 requisições por segundo, com 100 conexões concorrentes, durante 60 se-

gundos. Esse cenário permitiu avaliar como cada plataforma detecta picos de demanda, aciona o *auto-scaling* e mantém a estabilidade do serviço, por meio da análise de métricas.

Para garantir resultados consistentes, foram realizadas 30 execuções de teste por plataforma. Os dados coletados foram analisados estatisticamente em Python, no Google Colab, possibilitando a identificação de padrões de comportamento e diferenças de desempenho entre as soluções avaliadas.

A análise dos dados iniciou-se com a verificação da normalidade dos dados, utilizando-se o teste de Kolmogorov-Smirnov. Valores de p abaixo de 0,05 indicam que os dados não seguem uma distribuição normal.

Na Tabela 1, observa-se que, no caso do Knative, as distribuições normais foram identificadas apenas para as métricas de Utilização da Memória, sugerindo que as demais apresentam variações significativas. Indicando comportamentos não uniformes durante os testes, refletindo oscilações nas métricas de desempenho que podem ser influenciadas diretamente pela dinâmica do mecanismo de *auto-scaling*. Estas métricas apresentaram valores-p que indicam compatibilidade com a normalidade, conforme o teste de Kolmogorov-Smirnov.

Tabela 1. Teste de Kolmogorov-Smirnov para o Knative

Métrica	p-value	Distribuição Normal?
CPU Usage	0.0095	Não
Memory Utilization	0.7561	Sim
Requests/sec	0.0358	Não
Num Pods	0.0003	Não

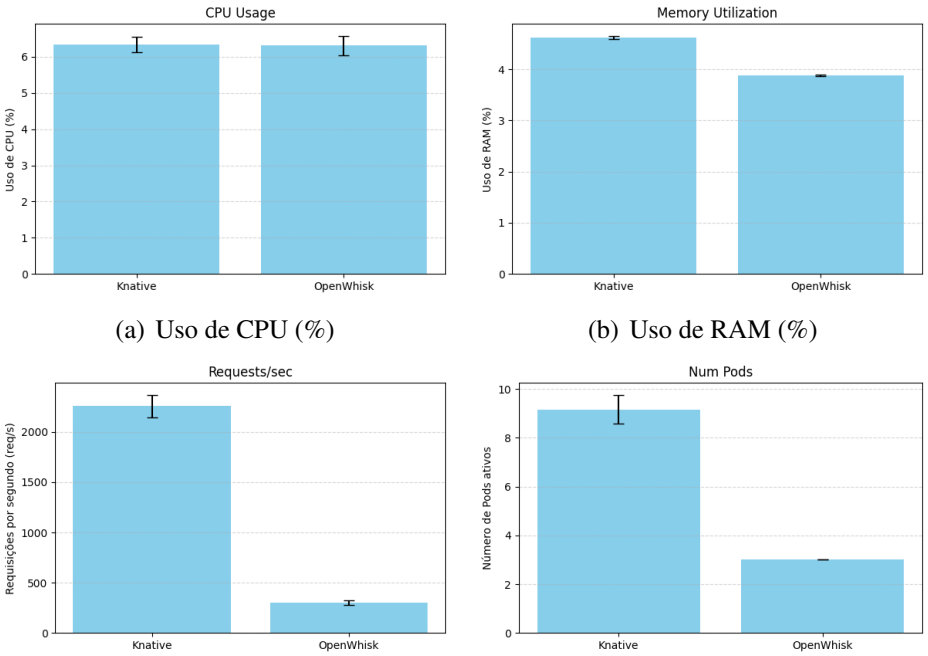
Na Tabela 2, para o OpenWhisk, as distribuições normais foram detectadas nas métricas de Utilização da Memória, Requisições por Segundo, o que indica um comportamento mais estável e previsível nesses aspectos ao longo das execuções. A normalidade nessas métricas sugere que o mecanismo de *auto-scaling* da plataforma respondeu de forma consistente à carga aplicada, mantendo variações dentro de um padrão esperado. O valor do *p-value* na métrica número de pods deu "nan" pelo valor baixo de números de pods ativos durante o teste.

Tabela 2. Teste de Kolmogorov-Smirnov para o OpenWhisk

Métrica	p-value	Distribuição Normal?
CPU Usage	0.0101	Não
Memory Utilization	0.1237	Sim
Requests/sec	0.3183	Sim
Num Pods	nan	Não

Após a verificação da normalidade das métricas na primeira análise, foi possível aplicar análise de estatística inferencial para obter estimativas mais robustas do comportamento das plataformas, como mostra a Figura 4.

Analisando a Figura 2(a) referente a média e intervalo de confiança da métricam Uso da CPU os valores foram muito próximos e o intervalo de confiança se encontram,



(a) Uso de CPU (%)

(b) Uso de RAM (%)

(c) Requisições por segundo (req/s)

(d) Número de Pods ativos

Figura 2. Análise Estatística Inferencial - Knative x OpenWhisk

sendo assim estatisticamente não se pode inferir sobre qual plataforma *serverless* manteve um melhor desempenho.

Em relação à métrica de Utilização da Memória, conforme apresentado na Figura 2(b), observa-se que o OpenWhisk apresentou menor consumo desse recurso em comparação ao Knative. Sugerindo que o OpenWhisk manteve uma maior eficiência na alocação de memória durante o processo de *auto-scaling*, indicando um uso mais econômico e potencialmente otimizado da infraestrutura disponível. Na Figura 2(c), é ilustrado o resultado da análise da métrica requisições por segundo, o Knative manteve um melhor desempenho. Isso se deve à capacidade do Knative de gerenciar melhor o escalonamento de recursos sob alta carga, resultando em uma maior quantidade de requisições atendidas.

A Figura 2(d) apresenta uma eficiência no uso de pods e a boa capacidade de *auto-scaling* do Knative permitiram que ele mantivesse um desempenho superior, mesmo sob maior demanda, contrastando com o comportamento do OpenWhisk que não conseguiu atender tantas requisições por segundo por justamente não ter escalado bem.

Na Figura 3, é apresentada a análise dos percentis por meio de um gráfico que evidencia a variação da latência conforme o aumento dos percentis, permitindo a identificação de picos e padrões de desempenho. Verifica-se que o Knative manteve uma latência mais baixa em praticamente todos os percentis quando comparado ao OpenWhisk. No entanto, a partir do percentil 95%, é possível notar o surgimento de diversos picos de latência, principalmente no OpenWhisk, o que indica dificuldade em lidar com a carga de requisições mais elevada e sugere um comportamento menos estável sob maior demanda.

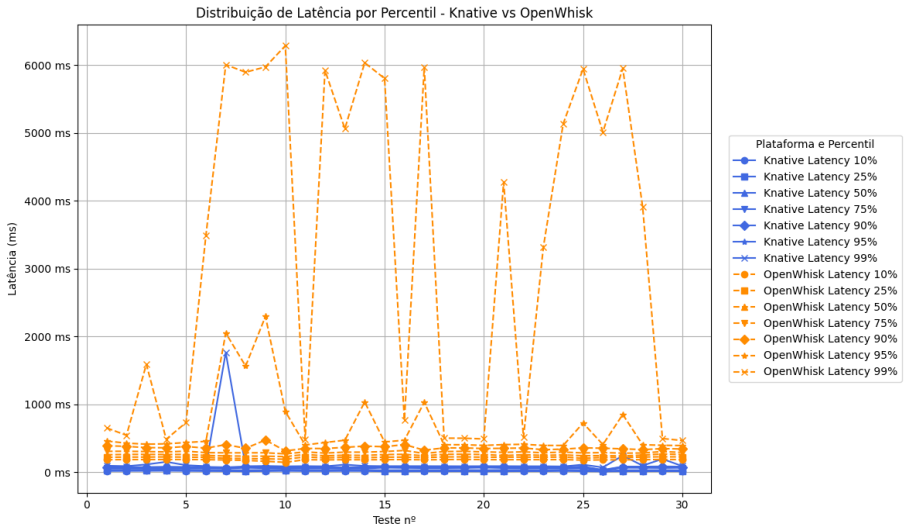


Figura 3. Percentis da Latência - Knative x OpenWhisk

A análise de *auto-scaling* em plataformas *serverless* em ambiente MicroK8s se torna necessária para a garantia de um bom desempenho e eficiência. O *auto-scaling* permite que a infraestrutura possa ser ajustada automaticamente o número de réplicas de uma aplicação conforme haja necessidade para aumento do tráfego. Em relação a plataformas *serverless*, isso significa que os recursos são alocados de maneira eficiente, garantindo que eficiência e escalabilidade, diminuindo até mesmo os custos.

Ao comparar o desempenho de *auto-scaling* das plataformas *serverless* Knative e Apache OpenWhisk no ambiente MicroK8s, foram identificadas diferenças significativas em termos de latência, uso de recursos e escalabilidade. O Knative apresentou o melhor desempenho, isso pode ser atribído à sua arquitetura menos complexa e a integração nativa com componentes Kubernetes, podendo facilitar maior eficiência no escalonamento de recursos. Por outro lado, o Apache OpenWhisk, apesar de ser uma plataforma mais robusta e flexível, possui uma arquitetura mais complexa e dependente de múltiplos serviços, o que pode ter introduzido maior sobrecarga no gerenciamento dos recursos e atrasos no processo de *auto-scaling*.

5.2. Análise de Desempenho do Knative sob Diferentes Cargas

A partir da primeira análise, o estudo foca para uma análise de desempenho do *auto-scaling* da plataforma *serverless* Knative no ambiente MicroK8s. Para isso, foi utilizada uma ferramenta de *benchmark* expandida, desenvolvida no contexto deste trabalho, com o objetivo de coletar métricas detalhadas sobre o comportamento da plataforma sob diferentes cargas de trabalho. Conforme mencionado anteriormente, foram realizadas variações na quantidade de requisições concorrentes, utilizando os seguintes parâmetros: -z 60s -c 10, -z 60s -c 50 e -z 60s -c 100, permitindo assim observar o comportamento do *auto-scaling* em diferentes níveis de concorrência.

Para inicar a análise, foi realizado um teste de normalidade para verificar se os dados seguem uma distribuição normal em todas as métricas e cenários. Foi aplicado o teste de Shapiro-Wilk, no qual um *p-value* menor que 0,05 indica que os dados não seguem uma distribuição normal. Nesse caso, determina-se que os testes subsequentes

devem ser não paramétricos.

No cenário com 10 requisições concorrentes (Tabela 3), apenas a métrica de Uso da Memória apresentou distribuição normal. No cenário com 50 requisições concorrentes (Tabela 4), foi o Uso de CPU que seguiu uma distribuição normal. Por fim, no cenário com 100 requisições concorrentes (Tabela 5), as métricas de Uso de CPU, Uso de Memória e Requisições por segundo exibiram distribuição normal.

Essas variações indicam que, conforme a carga aumenta, diferentes métricas apresentam comportamentos estatísticos distintos, o que pode influenciar a interpretação dos resultados de desempenho. Além disso, pode-se inferir que, nas cargas de 10 e 50 requisições, houve maior variabilidade dos dados. Já no cenário com 100 requisições, a não normalidade observada na métrica número de pods pode ser atribuída a variações causadas por problemas de *cold start*.

Tabela 3. Teste Shapiro-Wilk - cenário de 10 concorrentes

Métrica	p-value	Distribuição Normal?
Uso de CPU	9.187568e-03	Não
Uso de Memória	2.667708e-01	Sim
Número de Pods	5.981490e-10	Não
Requisições por Segundo	3.359333e-04	Não

Tabela 4. Teste Shapiro-Wilk - cenário de 50 concorrentes

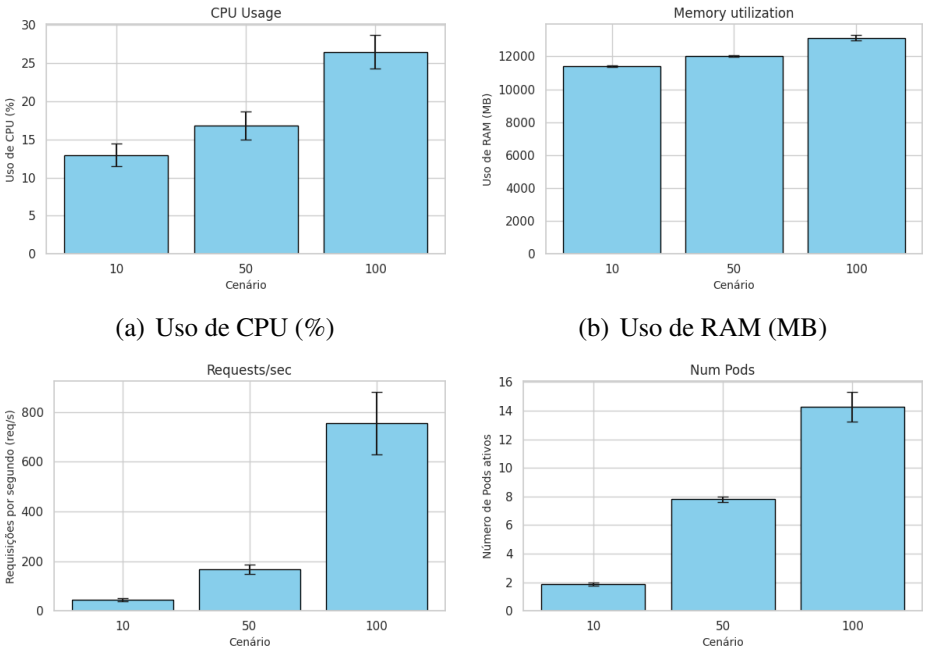
Métrica	p-value	Distribuição Normal?
Uso de CPU	1.291951e-01	Sim
Uso de Memória	2.402037e-02	Não
Número de Pods	2.824241e-07	Não
Requisições por Segundo	4.336038e-03	Não

Tabela 5. Teste Shapiro-Wilk - cenário de 100 concorrentes

Métrica	p-value	Distribuição Normal?
Uso de CPU	9.493429e-01	Sim
Uso de Memória	1.218137e-01	Sim
Número de Pods	1.102838e-07	Não
Requisições por Segundo	2.200498e-01	Sim

Para seguir a análise, foram elaborados gráficos com a média e o intervalo de confiança, a fim de fornecer uma representação clara e quantitativa da variabilidade e precisão dos dados, como mostra a Figura 4.

Na Figura 4(a), verifica-se que o uso de CPU cresce proporcionalmente à carga de trabalho, passando de 12,98% no cenário de 10 requisições por segundo para 26,48% no cenário de 100 requisições. Esse aumento reflete a maior demanda computacional e a eficiência do sistema de *auto-scaling* do Knative, que ajusta os recursos de CPU conforme o aumento nas requisições.



(a) Uso de CPU (%)

(b) Uso de RAM (MB)

(c) Requisições por segundo (req/s)

(d) Número de Pods ativos

Figura 4. Análise Estatística Inferencial - Knative sob diferentes cargas

Em relação à utilização de memória, como mostra a Figura 4(b), intentifica-se uma variação similar à do uso de CPU com o aumento da carga de trabalho. Esse aumento reflete a maior demanda de memória para suportar o crescimento do número de pods e o processamento das requisições, evidenciando a eficiência do sistema no ajuste proporcional da alocação de recursos.

Na Figura 4(c), constata-se um aumento nas requisições por segundo, refletindo a eficácia do Knative em lidar com tráfego crescente. O sistema passou de 45,60 requisições por segundo no cenário de 10 requisições para 755,56 no cenário de 100 requisições, evidenciando a capacidade do Knative em escalar automaticamente e manter o desempenho sob alta carga.

O número de pods aumentou proporcionalmente à carga como indicado na Figura 4(d), com uma média de 1,87 pods no cenário de 10 requisições, 7,80 pods no cenário de 50 requisições e 14,27 pods no cenário de 100 requisições. Este comportamento demonstra a eficácia do mecanismo de *auto-scaling* do Knative, que ajusta automaticamente a quantidade de pods disponíveis conforme a demanda cresce. A resposta rápida do Knative a diferentes cenários de carga mostra a capacidade da plataforma em dimensionar recursos de forma dinâmica e eficiente, garantindo que a infraestrutura de pods seja suficiente para suportar as requisições sem causar gargalos.

A análise dos percentis de latência, conforme apresentado na Figura 5, revela que a maior variabilidade ocorre no percentil 99%, especialmente nas cargas mais baixas e médias. Essa variação mais expressiva pode ser atribuída ao comportamento do sistema durante momentos de baixa carga, onde há uma maior probabilidade de o tempo de resposta ser afetado por fatores como inicialização de novos pods. Em contrapartida, nas cargas mais altas, o sistema tende a manter uma latência mais constante, possivelmente

devido ao maior número de pods em operação e ao ajuste mais eficiente dos recursos, melhorando o seu desempenho.

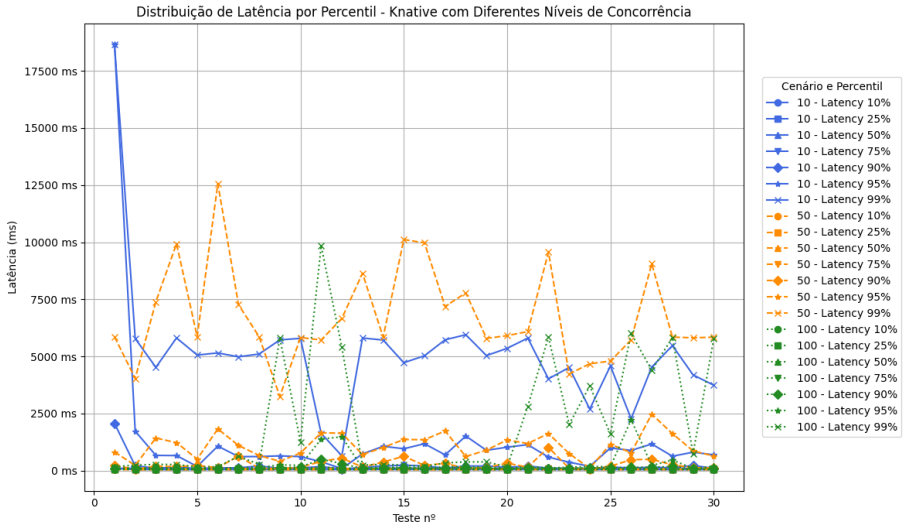


Figura 5. Percentis da Latência - Knative sob diferentes cargas

Os resultados mostram que o Knative ajusta dinamicamente o número de pods conforme a carga de trabalho, mantendo a operação sob alta concorrência, mas afetando latência e uso de recursos. A análise estatística indicou que algumas métricas, especialmente latências em percentis elevados, não seguiram distribuição normal, sugerindo maior variabilidade do tempo de resposta sob carga intensa, possivelmente devido ao tempo de inicialização de novos pods.

O uso de CPU e memória cresce proporcionalmente à concorrência, mas uma carga excessiva pode impactar significativamente o sistema. O aumento no número de pods confirma o funcionamento do *auto-scaling*, mas o sistema pode atingir limitações em cenários de carga extrema, indicando que aumentar apenas os pods não é suficiente para manter o desempenho. Isso destaca a necessidade de otimizações no escalonamento, como melhorias no *cold start* e *warm start*, para reduzir a variabilidade da latência.

6. Conclusão

Este estudo conclui que a plataforma *serverless* que apresentou o melhor desempenho foi o Knative. Esse desempenho superior pode ser atribuído à sua menor complexidade de configuração em comparação com o Apache OpenWhisk. A configuração mais complexa do OpenWhisk, especialmente no que diz respeito ao seu mecanismo de *auto-scaling*, introduziu desafios adicionais, afetando a performance e a facilidade de manutenção da plataforma. Esse fator deve ser considerado ao escolher uma plataforma *serverless*, já que a complexidade pode impactar não apenas o desempenho, mas também a eficiência operacional.

A análise de desempenho do Knative sob diferentes níveis de carga revelou que, enquanto o sistema se ajusta dinamicamente ao aumento da carga de trabalho, esse escalonamento impacta tanto o uso de recursos quanto a latência. O melhor desempenho foi observado no cenário com 10 requisições concorrentes, onde a latência foi baixa e o

uso de recursos eficiente. Em cenários com 50 e 100 requisições concorrentes, a latência aumentou, e o sistema começou a mostrar variações significativas, indicando possíveis limitações de escalabilidade em cenários de carga extrema. Essas observações fornecem *insights* importantes sobre os limites do Knative em situações de alta carga.

Para trabalhos futuros, seria interessante explorar o desempenho do Knative em diferentes distribuições Kubernetes, além de investigar a resiliência a falhas, escalabilidade horizontal e a integração com ferramentas de DevOps. Essas análises poderiam oferecer uma visão mais completa sobre a adaptabilidade e o desempenho das plataformas *serverless* em diferentes ambientes e condições de carga.

Referências

Copik, M., Kwasniewski, G., Besta, M., Podstawski, M., and Hoefler, T. (2021). Sebs: A serverless benchmark suite for function-as-a-service computing. In *Proceedings of the 22nd International Middleware Conference*, pages 64–78.

d Moura, E. S. (2020). Em direção a uma ferramenta para a realização de benchmarking de funções serverless em ambientes possivelmente multi-cloud.

Kjorveziroski, V. and Filiposka, S. (2022). Kubernetes distributions for the edge: serverless performance evaluation. *The Journal of Supercomputing*, 78(11):13728–13755.

Koziolek, H. and Eskandani, N. (2023). Lightweight kubernetes distributions: A performance comparison of microk8s, k3s, k0s, and microshift. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, pages 17–29.

Li, J., Kulkarni, S. G., Ramakrishnan, K., and Li, D. (2021). Analyzing open-source serverless platforms: Characteristics and performance. *arXiv preprint arXiv:2106.03601*.

Mohanty, S. K., Premsankar, G., and Di Francesco, M. (2018). An evaluation of open source serverless computing frameworks. In *IEEE International Conference on Cloud Computing Technology and Science*, pages 115–120. IEEE.

Palade, A., Kazmi, A., and Clarke, S. (2019). An evaluation of open source serverless computing frameworks support at the edge. In *2019 IEEE World Congress on Services (SERVICES)*, volume 2642, pages 206–211. IEEE.

Pereira, G. S. (2019). Implantação do knative em um cluster de kubernetes utilizando a google cloud.

Rajput, K. R., Kulkarni, C. D., Cho, B., Wang, W., and Kim, I. K. (2022). Edgefaasbench: Benchmarking edge devices using serverless computing. In *2022 IEEE International Conference on Edge Computing and Communications (EDGE)*, pages 93–103. IEEE.

Sousa, F. R. (2020). Computação serverless e gerenciamento de dados. In *Anais do XXXV Simpósio Brasileiro de Bancos de Dados*, pages 199–204. SBC.

Tari, M., Ghobaei-Arani, M., Pouramini, J., and Ghorbian, M. (2024). Auto-scaling mechanisms in serverless computing: A comprehensive review. *Computer Science Review*, 53:100650.

Wen, J., Chen, Z., Jin, X., and Liu, X. (2023). Rise of the planet of serverless computing: A systematic review. *ACM Transactions on Software Engineering and Methodology*, 32(5):1–61.