Kubemon: extrator de métricas de desempenho de sistema operacional e aplicações conteinerizadas em ambientes de nuvem no domínio do provedor

Pedro Horchulhack[1], **Eduardo K. Viegas**[1], **Altair O. Santin**[1], **Felipe V. Ramos**[1]

[1]Programa de Pós-Graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná (PUCPR)
80.215-901 – Curitiba – PR

{pedro.horchulhack, eduardo.viegas, santin, felipe.ramos}@ppgia.pucpr.br

Abstract. *Cloud computing, mainly through containerization, has been wildly used in the last years. Generally, the deployed applications performance metrics are provided by the cloud provider, that may lead to conflict of interest. To retrieve those metrics without conflict of interests, as part of a work to the ICC2021, was proposed the Kubemon, that was able to collect* 14 *features in the first version and* 122 *in the second one, from the containerized operating system, Docker containers, and processes created by Docker, from the provider perspective. The tool is under development.*

Resumo. *A computação em nuvem, especialmente através das aplicações conteinerizadas, tem sido cada vez mais utilizada nos últimos anos. Em geral, as métricas de desempenho das aplicações implantadas são fornecidas pelos provedores de nuvem, o que pode levar a um conflito de interesses. Visando coletar métricas para avaliação sem tais conflitos de interesse, como parte de um trabalho para o ICC2021, é proposto o Kubemon que é capaz de coletar* 14 *métricas na primeira versão e* 122 *na segunda, considerando o sistema operacional, contêineres e os processos do contêiner, do ponto de vista do provedor. A ferramenta continua em desenvolvimento.*

1. Introdução

A computação em nuvem ganhou popularidade principalmente através do barateio da fabricação de computadores, aumento da capacidade de processamento e evolução das redes de comunicação. Tal paradigma levantou diversas possibilidades de pesquisa, bem como de utilização na indústria [Goyal 2014]. Nesse modelo os recursos físicos são virtualizados de modo que múltiplos usuários utilizem do mesmo hardware simultaneamente, onde é pago somente a quantia de recursos que foi consumida, estratégia esta denominada como *pay-as-you-go* [Vicentini et al. 2018]. No entanto, existem casos onde os usuários locatários não utilizam todos os recursos alocados, mantendo-os inutilizados e levando a problemas de consumo de recursos como energia, rede e processamento desnecessários.

Haja visto que a computação em nuvem foca principalmente no comércio de hardware virtualizado, existem duas principais estratégias de virtualização utilizadas na indústria: máquinas virtuais (*Virtual Machines*, VMs) e containers. A primeira técnica cria um sistema operacional (SO) sobrejacente à um software chamado *hypervisor*, cujo é responsável pelo isolamento, instanciação e justiça no compartilhamento de recursos

físicos entre demais máquinas virtuais [Arunarani et al. 2019]. Já os containers focam no isolamento de aplicações à nível de processo por meio do compartilhamento do núcleo do SO e alguns de seus mecanismos como *cgroups* e *namespaces*. Dentre as duas maneiras de virtualização as VMs é alvo de massiva utilização na indústria e foco de estudos, no entanto apresentam uma sobrecarga significativamente maior em comparação a containers justamente por instanciarem todo um SO sobrejacente à outro [dos Santos et al. 2023a]. Dessa forma, empresas têm migrado suas aplicações para containers, pois apresentam tempo de criação, consumo de memória e processamento e maior flexibilidade substancialmente menores que VMs [Xing et al. 2022]. Além disso, como a virtualização ocorre a nível de SO, todos os inquilinos compartilham do mesmo SO do provedor de nuvem e executando de forma isolada uns dos outros, porém, por serem processos, são gerenciados pelo escalonador de processos do SO. Tal fato pode incorrer à problemas de justeza no que tange à distribuição de recursos entre containers vizinhos [dos Santos et al. 2023b].

Independente do modelo de virtualização escolhido, uma das tarefas mais importantes é garantir que os recursos contratados sejam efetivamente entregues. Uma das estratégias mais utilizadas são os acordos de níveis de serviço (*Service Level agreement* – SLA), que definem o que se espera que seja entregue, embora, via de regra, se encarreguem majoritariamente do tempo disponível. Para garantir a conformidade do SLA, provedores de nuvem fornecem indicadores de nível de serviço (*Service Level Indicators* – SLI), que são métricas a respeito do desempenho da aplicação na nuvem. No entando, possíveis conflitos de interesses residem no fato que o próprio provedor, que deve garantir a aderência ao SLA, comprova isso por meio do SLI. Um dos principais problemas é o *overbooking* de recursos, que significa a venda de mais recursos virtualizados do que os efetivamente disponíveis [Kholidy 2020], problema que já foi abordado por alguns autores como, por exemplo, [Tomás and Tordsson 2013, Tomás and Tordsson 2014, Son et al. 2017, Bashir et al. 2021, Abreu et al. 2017].

Nesse sentido, foi proposto o trabalho em [Ramos et al. 2021], que objetiva identificar *overbooking* de recursos de uma perspectiva do domínio do cliente. Para permitir a execução do método proposto, foi apresentado o programa *Kubemon*, descrito neste trabalho, que realiza o monitoramento de 14 métricas, sendo 8 do Sistema Operacional (SO) e 6 de uma aplicação Apache Spark. A ferramenta foi projetada para executar no ambiente de orquestração de contêineres *Kubernetes* para simular um cenário real de nuvem computacional. O programa, com versão operacional para a coleta dos dados para o artigo supracitado, continua passando por melhoras e, através do fornecimento de métricas dentro do domínio do cliente, garante a imparcialidade dos resultados, não incorrendo em conflito de interesses.

As principais contribuições desta ferramenta, no âmbito do trabalho para a qual foi desenvolvida, foram:

- Desenvolvimento de uma ferramenta livre de ingerências do provedor, para extração de métricas de sistema operacional e aplicação implantada no domínio do cliente;
- Extrator de característica focado em detecção de *overbooking* de recursos.

O artigo organiza-se como segue. Na Seção 2 é descrito o trabalho que originou esta ferramenta. Na Seção 3 a ferramenta é detalhada e finalmente a Seção 4 sumariza as conclusões deste trabalho e indica próximos passos possíveis no desenvolvimento do

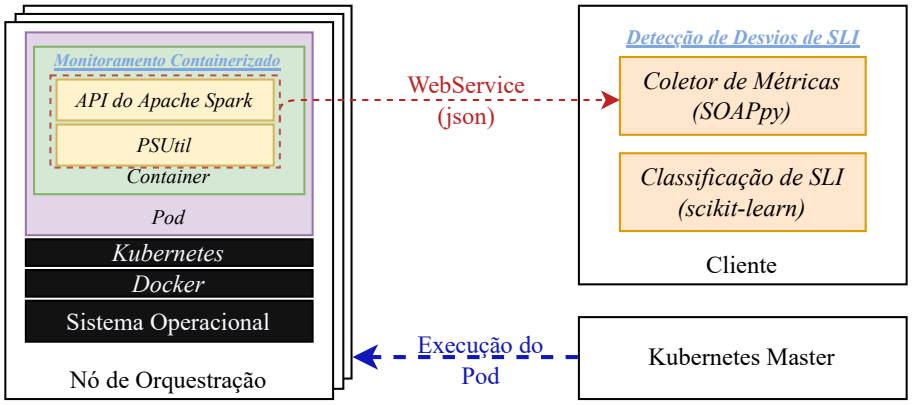

Figura 1. Protótipo da ferramenta. Fonte: [Ramos et al. 2021]

Kubemon.

2. Artigo Relacionado

Esta ferramenta foi desenvolvida no contexto do trabalho de Ramos *et al.* [Ramos et al. 2021], sendo descrita nas Seções IV e VI. O artigo foi publicado na *IEEE International Conference on Communications* (*ICC* – 2021). Além disso, foi utilizada no artigo de Horchulhack *et al.* [Horchulhack et al. 2022], publicada na conferência *IEEE Global Communications Conference* (*GLOBECOM* – 2022).

No primeiro artigo mencionado, foi proposto um modelo de aprendizado de máquina para detecção de *overbooking* de recursos em ambientes de nuvem de aplicações conteinerizadas. Nesta proposta, a ferramenta aqui apresentada foi usada para extrair as métricas utilizadas no treinamento de classificadores que avaliam a existência ou não de degradação causada por *overbooking* de recursos nas aplicações. Provou-se que tal degradação pode elevar substancialmente o tempo necessário para a execução de tarefas, façam elas uso intensivo apenas de processamento (*CPU*) ou de diversos recursos de uma vez (memória, *CPU*, E/S em disco e rede). Sem esta ferramenta, não seria possível extrair os dados necessários para validar a proposta no artigo em questão.

Já no segundo artigo, foi proposto um modelo de aprendizagem profunda utilizando Redes Neurais Recorrentes (*Recurrent Neural Networks*, RNNs) para a classificação temporal da presença de *overbooking* ou não. A proposta do estudo foi avaliar, da perspectiva do provedor de serviço, o impacto causado por aplicações vizinhas executarem tarefas de processamento de Big Data simultaneamente e paralelamente. Além disso, a ferramenta foi utilizada e complementada de modo à coletar mais métricas de dois subconjuntos distintos além de SO: métricas de containers e processos dos containers. Tais subconjuntos permitiram que a classificação dos eventos fosse mais granular, permitindo que fosse avaliado quais containers estavam sofrendo degradação sem ter acesso direto ao container.

3. Descrição da Ferramenta

A ferramenta proposta, chamada de Kubemon, permite a extração de métricas do SO virtualizado e da aplicação conteinerizada distribuída via a API REST do Apache Spark. Esta Seção detalha a ferramenta e suas versões.

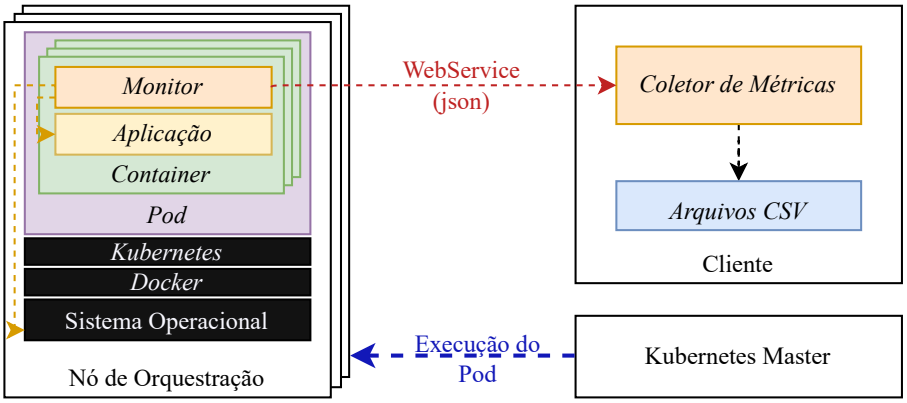

Figura 2. Diagrama da Ferramenta, V1

A primeira versão da ferramenta, utilizada no trabalho [Ramos et al. 2021], garantia a extração das métricas de dentro do ambiente do inquilino, ou seja, do contêiner. Isto afasta a possibilidade de conflitos de interesses, inclusive a distorção de informações repassadas pelo provedor de nuvem computacional. Já a versão mais recente considera um conjunto de métricas mais abrangente, considerando métricas de SO, contêineres e processos criados pelos contêineres.

Ela foi desenvolvida na linguagem Python 3.8, utilizando das bibliotecas `psutil v5.9.0` para a coleta das métricas e `requests v2.27.1` para requisições HTTP. A ferramenta possui dois componentes principais: o *monitor* e o *coletor*. A arquitetura adotada para a ferramenta foi a de microsserviços, onde os componentes responsáveis pela coleta de métricas (*monitores*) e o componente responsável pela persistência dos dados coletados (*coletor*) sejam fracamente acoplados. O fluxo de provisionamento da ferramenta se dá através da instanciação dos componentes *monitores*, portanto no mínimo um *monitor* deve estar presente em cada nó de um cluster *Kubernetes*. Após isso, deve ser realizada a instanciação do componente *coletor*, preferencialmente no nó *Kubernetes* responsável pelo agendamento dos *pods*. A Figura 1 representa a relação entre os *monitores* e o *coletor*, sendo uma versão adaptada de [Ramos et al. 2021]. Também, a figura em questão demonstra uma das possíveis aplicações da ferramenta.

Por fim, a comunicação realizada entre os *monitores* e o *coletor*, cada *monitor* instanciado envia ao *coletor* as métricas coletadas através do protocolo SOAP (Simple Object Access Protocol), onde ele converte para uma representação JSON (JavaScript Object Notation) e finalmente as salva em um arquivo CSV. As métricas são coletadas em um intervalo de tempo de 5 segundos.

3.1. Primeira Versão

A principal abordagem considerada nesta versão é a coleta de informações no domínio do cliente, isto é, de dentro de um contêiner. A ferramenta em tal versão é dividida em três componentes: dois *monitores* para cada nó e um *coletor*. Tangente aos *monitores* que coletam o total de 14 métricas, listadas na Tabela 1, o primeiro *monitor* coleta métricas de sistema operacional, totalizando 8 métricas. Já o segundo *monitor* coleta o total de 6 métricas de aplicação do Apache Spark, que são provenientes de sua *API REST*. A Figura 2, análoga a Figura 1, representa o protótipo implementado na versão em questão.

Conjunto de Métricas	*Métricas Extraídas*
Métricas de APP (API do Apache Spark)	Uso de CPU da JVM, Escritas Shuffle, Leituras Shuffle, uso de memória na execução, tempo do GC da JVM
Métricas de SO (PSUtil)	Uso de CPU, uso de memória, setores escritos no disco, setores lidos do disco, bytes enviados, bytes recebidos, pacotes enviados, pacotes recebidos

Tabela 1. Conjunto de características extraídas da etapa de monitoramento *conteinerizado* no intervalo de 5 segundos.

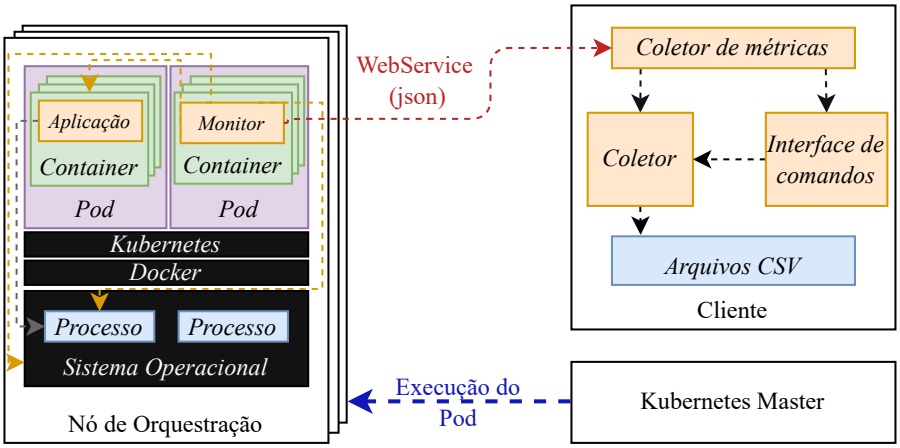

Figura 3. Diagrama da Ferramenta, V2

Ela explicita as relações entre os componentes, bem como o fluxo de coleta de dados e a persistência deles.

Ainda, considerando que a coleta das métricas é realizada por consultas periódicas à APIs de acesso a objetos como REST e SOAP, a ferramenta se demonstra flexível no que tange a modularização. Portanto, contanto que um *monitor* forneça uma API REST ou SOAP, é possível que tais dados sejam coletados com sucesso.

O protótipo do trabalho apresentado em [Ramos et al. 2021], presente na Figura 1, mostra um dos possíveis usos desta ferramenta. É possível compreender com esse protótipo como a ferramenta pode ser implantada na prática.

3.2. Segunda Versão

Com o objetivo de melhorias de implementação, a ferramenta continua em desenvolvimento e já foi utilizada no artigo [Horchulhack et al. 2022]. O principal objetivo da segunda versão da ferramenta, além dos aprimoramentos, é a de coleta no contexto do provedor de nuvem computacional. Em comparação a versão anterior do *Kubemon*, algumas melhorias foram incorporadas a ferramenta. Dentre elas, está um conjunto mais abrangente de métricas, podendo coletar até 122 características de três visões distintas: métricas de SO (49 métricas), de contêineres (39 métricas) e de processos criados pelos contêineres (34 métricas).

A Figura 3 ilustra a relação entre *monitores* e o *coletor*. Ainda, houve a adição

Comando	*Argumentos*	*Descrição*
`start`	`DIR_NAME`	Inicia a coleta das métricas, onde serão salvas no argumento `DIR_NAME` especificado.
`stop`	–	Para a coleta de métricas.
`instances`	–	Lista, por visão e nó, a quantidade de instâncias cujas métricas serão coletadas.
`daemons`	–	Lista todos os *monitores* existentes.
`alive`	–	Verifica se o *coletor* está inicializado.
`help`	–	Lista todos os comandos disponíveis da ferramenta.

Tabela 2. Lista de comandos e seus argumentos da segunda versão da ferramenta.

de um novo componente, denominado *Interface de Comandos*, com o intuito de melhorar o controle entre os demais componentes. A Tabela 2 lista os comandos implementados, bem como a descrição de cada um e seus argumentos. A comunicação realizada entre os componentes *coletor* e *inteface de comandos* é através do protocolo TCP.

A instanciação de cada *monitor* é feita através da implantação de um *DaemonSet* do *Kubernetes*. Um *DaemonSet* garante que um conjunto de *Pods* seja instanciado em todos os nós do cluster. O *coletor* é instanciado como um *Pod* no *Kubernetes Master*. Cada *monitor* instanciado, o *coletor* cria uma thread para permitir coleta de dados de diversos contêineres, processos e nós simultaneamente. Ainda, a *interface de comandos* pode ser executada localmente desde que se saiba o endereço IP do *coletor*.

A comunicação entre os *monitores* e o *coletor* é realizada de forma semelhante a primeira versão. No entanto, cada *monitor* fornece uma API REST implementada através das bibliotecas `flask v2.0.3`, `flask_restful v0.2.12` e `gunicorn v20.1.0`, onde as métricas são fornecidas através de um JSON. A Tabela 3 lista por conjunto de métricas (SO, contêineres, processos) em comum.

3.3. Principais Funcionalidades

Ambas as versões citadas no trabalho possuem funcionalidades como a coleta de métricas e a instanciação dos componentes dela em um cluster *Kubernetes*. No entanto, considerando a versão mais recente, as principais funcionalidades e características da ferramenta são:

- As métricas são coletadas no domínio do provedor;
- As métricas são coletadas de dentro de contêineres Docker;
- Existência de comunicação via interface de linha de comando;
- Os componentes da ferramenta podem ser implantados em um cluster *Kubernetes*;
- A configuração dos componentes pode ser realizada através de variáveis de ambiente.

Embora ainda não existam trabalhos publicados utilizando a versão 2.0 da ferramenta, as novas funcionalidades permitem realizar abordagens distintas, posto que extraem informações no domínio do provedor. Muito embora não seja, a priori, exequível em ambientes de produção, é uma importante fonte de estudos.

Conjunto de Métricas	*Subconjunto*	*Métricas*
Sistema Operacional, Contêiner, Processo	Rede	Bytes Enviados
		Bytes Recebidos
		Pacotes Enviados
		Pacotes Recebidos
	Disco	Bytes Escritos
		Bytes Lidos
		Bytes Descartados
		Caracteres Lidos
		Caracteres Escritos
		Quantidade de Leituras no Disco
		Quantidade de Escritas no Disco
	CPU	Tempo de Uso do CPU do Usuário
		Tempo de Uso do CPU do Sistema
	Memória	Quantidade de Falhas de Paginação
		Quantidade de Páginas Mapeadas
		Quantidade de Páginas Compartilhadas
		Quantidade de Páginas Ativas
		Quantidade de Páginas Inativas

Tabela 3. Tabela de métricas em comum entre os módulos coletadas pela segunda versão do *Kubemon*.

3.4. Disponibilização da Ferramenta

O *Kubemon*, incluindo sua documentação, código fonte, manuais de uso e demonstrativos está disponível em `https://hrchlhck.github.io/kubemon/pt-br/`.

4. Conclusão

Neste trabalho, foi apresentada a ferramenta *Kubemon*. O objetivo desta ferramenta é a extração de métricas de uso de aplicações conteinerizadas e orquestradas em nuvens computacionais. Continuando em desenvolvimento, o *Kubemon* conta com duas versões, que abordam tanto a coleta das métricas do ponto de vista do cliente quanto do provedor de nuvem computacional.

Melhorias futuras incluem a implantação de mecanismos de instanciação dinâmica da ferramenta, de modo similar a ferramenta de orquestração de contêineres *Kubernetes* e implementar novo mecanismo de coleta de métricas de rede de modo a aumentar a quantidade de informações.

Referências

Abreu, V., Santin, A. O., Viegas, E. K., and Stihler, M. (2017). A multi-domain role activation model. In *2017 IEEE International Conference on Communications (ICC)*. IEEE.

Arunarani, A., Manjula, D., and Sugumaran, V. (2019). Task scheduling techniques in cloud computing: A literature survey. *Future Generation Computer Systems*, 91:407–415.

Bashir, N., Deng, N., Rzadca, K., Irwin, D., Kodak, S., and Jnagal, R. (2021). Take it to the limit: Peak prediction-driven resource overcommitment in datacenters. In *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, page 556–573, New York, NY, USA. Association for Computing Machinery.

dos Santos, R. R., Viegas, E. K., Santin, A. O., and Cogo, V. V. (2023a). Reinforcement learning for intrusion detection: More model longness and fewer updates. *IEEE Transactions on Network and Service Management*, 20(2):2040–2055.

dos Santos, R. R., Viegas, E. K., Santin, A. O., and Tedeschi, P. (2023b). Federated learning for reliable model updates in network-based intrusion detection. *Computers &amp Security*, 133:103413.

Goyal, S. (2014). Public vs private vs hybrid vs community - cloud computing: A critical review. pages 20–29. International Journal of Computer Network and Information Security.

Horchulhack, P., Viegas, E. K., and Santin, A. O. (2022). Detection of service provider hardware over-commitment in container orchestration environments. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 6354–6359.

Kholidy, H. A. (2020). An intelligent swarm based prediction approach for predicting cloud computing user resource needs. *Computer Communications*, 151:133–144.

Ramos, F., Viegas, E., Santin, A. O., Horchulhack, P., dos Santos, R., and Espindola, A. (2021). A machine learning model for detection of docker-based APP overbooking on kubernetes. In *2021 IEEE International Conference on Communications (ICC): SAC Cloud Computing, Networking and Storage Track (IEEE ICC'21 - SAC-02 CCNS Track)*, Montreal, Canada.

Son, J., Dastjerdi, A. V., Calheiros, R. N., and Buyya, R. (2017). Sla-aware and energy-efficient dynamic overbooking in sdn-based cloud data centers. *IEEE Transactions on Sustainable Computing*, 2(2):76–89.

Tomás, L. and Tordsson, J. (2013). Improving cloud infrastructure utilization through overbooking. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing conference*, pages 1–10.

Tomás, L. and Tordsson, J. (2014). An autonomic approach to risk-aware data center overbooking. *IEEE Transactions on Cloud Computing*, 2(3):292–305.

Vicentini, C., Santin, A., Viegas, E., and Abreu, V. (2018). A machine learning auditing model for detection of multi-tenancy issues within tenant domain. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE.

Xing, T., Barbalace, A., Olivier, P., Karaoui, M. L., Wang, W., and Ravindran, B. (2022). H-container: Enabling heterogeneous-isa container migration in edge computing. *ACM Trans. Comput. Syst.*, 39(1–4).