Análise de desempenho de estratégias de autoscaling vertical e horizontal: um estudo de caso com o Kubernetes

Kewyn Akshlley[1], **Marcus Carvalho**[1], **Raquel Lopes**[1]

[1]Departamento de Ciências Exatas (DCX)
Universidade Federal da Paraíba (UFPB) – Campus Litoral Norte, Rio Tinto – PB

{kewyn.akshlley, marcuswac, raquel}@dcx.ufpb.br

Abstract. *Scalable applications may adopt horizontal or vertical autoscaling to dynamically provision resources in the cloud. To help choosing the best strategy, this work aims to compare the performance of horizontal and vertical autoscaling in Kubernetes. Through mesurement experiments using synthetic load to a web application, the horizontal was shown more efficient, reacting faster to the load variation and resulting in a lower impact on the application response time.*

Resumo. *Aplicações escaláveis podem adotar estratégias de autoscaling horizontal ou o vertical para provisionar recursos na nuvem. Para ajudar na escolha da melhor estratégia, este trabalho visa comparar o desempenho do autoscaling horizontal e vertical no Kubernetes. Através de experimentos de medição usando carga sintética em uma aplicação web, o autoscaling horizontal se mostrou mais eficiente, reagindo mais rapidamente às variações da carga e tendo um menor impacto no tempo de resposta da aplicação.*

1. Introdução

A carga das aplicações na nuvem podem variar ao longo do tempo. Para serem escaláveis, pode-se adotar estratégias de *autoscaling*, que consistem em dimensionar automaticamente a capacidade de processamento das aplicações de acordo com métricas específicas (ex: utilização de CPU). Com isso, busca-se cumprir as metas de qualidade de serviço da aplicação e reduzir os custos da infraestrutura na nuvem, que aumentam de acordo com o uso dos recursos [Mao et al. 2010, Mao and Humphrey 2013, Netto et al. 2017].

Existem dois tipos de autoscaling: horizontal e vertical. No autoscaling horizontal, a quantidade de servidores é aumentada ou diminuída e a carga é distribuída entre todas as réplicas existentes através de um balanceador de carga. No autoscaling vertical, a quantidade de recursos computacionais dos servidores (ex: CPU e memória) são aumentados ou diminuídos conforme a carga [Yazdanov and Fetzer 2012, Sedaghat et al. 2013]. Apesar da existência das duas abordagens, não existem critérios bem definidos para decidir qual usar dependendo do cenário, e também há uma escassez de análises em relação ao provisionamento automático vertical e como ele se compara ao horizontal em relação a desempenho e custo-benefício [Liu et al. 2014, Jiang et al. 2013].

Para gerenciar aplicações em contêineres, uma opção é usar uma ferramenta de orquestração que dê suporte a estratégias de autoscaling [Casalicchio 2017]. O Kubernetes é uma das ferramentas mais populares de orquestração de contêineres, que dá suporte tanto ao autoscaling horizontal quanto ao vertical. Apesar do autoscaling horizontal já ter sido amplamente estudado, o autoscaling vertical é relativamente novo no contexto

de computação em nuvem e contêineres, existindo apenas alguns trabalhos focados em explorá-lo [Qu et al. 2018]. Além disso, não foi encontrada uma análise comparativa de mecanismos de autoscaling em ferramentas de orquestração de contêineres existentes.

O objetivo deste trabalho é analisar o desempenho das estratégias de autoscaling vertical e horizontal em orquestradores de contêineres. Para isto, foram realizados experimentos de medição com o Kubernetes com as duas abordagens em diferentes cenários, usando uma ferramenta de benchmark para geração de carga controlada e uma aplicação web escalável com base em sua demanda. Os dois mecanismos foram avaliados com base no tempo para as tomadas de decisão de autoscaling após mudanças de carga, na capacidade de CPU requisitada em cada decisão e no impacto no tempo de resposta da aplicação web. Com este estudo, pretende-se auxiliar tomadores de decisão na escolha entre as duas estratégias de autoscaling, orientando-os melhor a atingir as metas de desempenho das aplicações e minimizar os custos de infraestrutura.

O resto do artigo está dividido da seguinte forma: a Seção 2 apresenta os trabalhos que abordaram problemas relacionados; a Seção 3 explica a metodologia, descrevendo como a avaliação foi realizada; a Seção 4 discute os resultados da avaliação; e, finalmente, a Seção 5 apresenta as conclusões do estudo.

2. Trabalhos relacionados

Nesta seção apresentamos pesquisas na área de autoscaling de aplicações em contêineres.

[Casalicchio 2017] apresentou problemas relacionados à orquestração automática de contêineres analisando as soluções existentes e seus desafios. Essas discussões motivaram o desenvolvimento deste trabalho. [Tosatto et al. 2015] analisaram o estado da arte e atuais desafios das pesquisas existentes sobre orquestração de contêineres na nuvem. São destacados fatores como de que forma a fragmentação das tecnologias está afetando a orquestração de contêiner e como os sistemas de monitoramento não estão aptos a atender as atuais demandas desta abordagem.

[Al-Dhuraibi et al. 2017] constataram que os estudos no campo de provisionamento automático são voltados mais para máquinas virtuais, sendo os trabalhos no contexto de contêineres mais focados no modelo de provisionamento horizontal. Com isto, foi proposto o sistema chamado “ELASTICDOCKER”, cujo objetivo é tornar o modelo de provisionamento automático vertical e horizontal mais eficiente. O foco foi promover uma proposta de um sistema comparando o seu desempenho em relação ao provisionamento automático horizontal e vertical de outros orquestradores de contêineres, o que o difere do objetivo deste trabalho que compara as abordagens de autoscaling vertical e horizontal de um mesmo orquestrador.

[Casalicchio and Perciballi 2017] trataram do problema de definir uma métrica ideal para a realização do autoscaling e concluíram que métricas absolutas (ex: CPU e memória) são mais precisas do que métricas relativas (ex: latência). Com base nisso, neste trabalho serão usadas métricas absolutas para a avaliação. [Tirmazi et al. 2020] realizaram um estudo onde compararam o histórico de informações sobre clusters gerenciados pela ferramenta Borg. Nessa análise foi possível perceber que o vertical autoscaling demonstra ser efetivo, pois através de históricos consegue fazer análises preditivas do uso de recursos de tarefas. Uma limitação é que o sistema analisado é apenas de uso interno em uma empresa, não estando disponível para uso externo e pesquisas.

Nenhum dos trabalhos analisados visa comparar empiricamente abordagens de autoscaling horizontal e vertical, que é o objetivo de pesquisa deste trabalho.

3. Metodologia

Nesta pesquisa foi adotado o método quantitativo através de experimentos de medição, com a avaliação de desempenho de abordagens de autoscaling vertical e horizontal do Kubernetes usando cargas de trabalho sintéticas. O ambiente de experimentação utilizado na análise está apresentado na Figura 1, com os componentes envolvidos na avaliação.

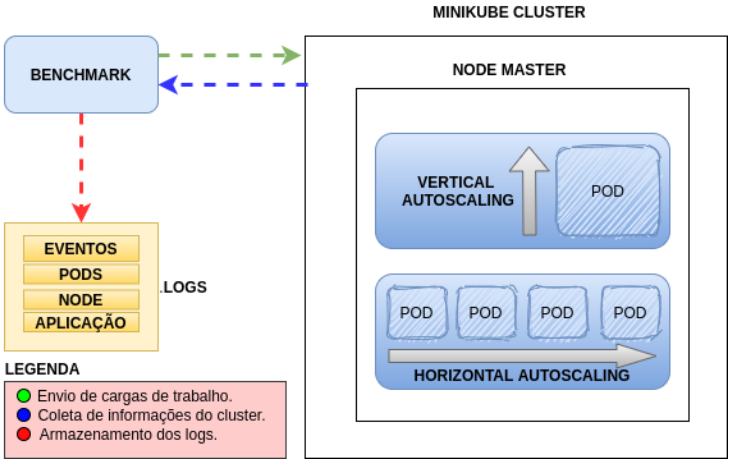

Figura 1. Arquitetura do ambiente de experimentação e seus componentes.

Para criar um ambiente de testes foi usado o Minikube, solução nativa do Kubernetes que fornece facilidades na execução de experimentos. A máquina virtual usada nesta avaliação contém 4 CPU-cores e 2 GB de memória, que foi suficiente para atingir os objetivos deste trabalho. Apesar de ser um ambiente local, os resultados alcançados com o Minikube podem ser facilmente replicados em provedores de serviço em nuvem.

Para gerar a carga controlada para a aplicação foi usada a ferramenta *hey*[1], que é um benchmark web moderno, escrito em Go, capaz de enviar requisições simultâneas e configurar os seguintes parâmetros referentes à carga de trabalho:

- z: duração da geração da carga, em segundos.
- c: quantidade de workers (geradores de carga) executados de forma concorrente.
- q: taxa de requisições por segundo enviadas por cada worker.

Para obter a carga de trabalho desejada, a taxa de requisições por worker (q) foi fixada em 1 requisição por segundo (rps) por worker e a quantidade de workers (c) foi variada para os diferentes cenários. Após a geração da carga e finalização do tempo z, o *hey* gera um sumário com estatísticas sobre o desempenho das respostas às requisições para métricas como o tempo de resposta, a vazão e a taxa de erros das requisições.

Para processar as requisições geradas e consumir recursos computacionais, foi criada uma aplicação web que expõe uma API usando *Node.js*. A aplicação consome recursos através de uma função de espera ocupada visando atingir 100% de uso de um

[1] `https://github.com/rakyll/hey`

núcleo de CPU por um período de tempo determinado. A função recebe como entrada o tempo de serviço (S), que determina por quanto tempo ela ficará em espera ocupada.

Em todos os cenários avaliados, o tempo de serviço foi fixado em $S = 0,175$ segundos. A intensidade da carga em um cenário é controlada pela taxa de requisições (λ) que indica a quantidade de workers simultâneos do *hey* gerando 1 rps cada. Cada cenário de experimento foi dividido em 9 estágios de intensidade de carga. A taxa de requisições gerada pelo benchmark se mantém constante durante um estágio, mas pode ser alterada entre um estágio e outro. Cada estágio foi definido com duração de 2 minutos ($z = 120$), totalizando um tempo de execução de 18 minutos para cada cenário de teste.

Para cada abordagem de autoscaling serão realizados 4 cenários de experimento, onde cada cenário será composto por 9 estágios de carga. Para obter a utilização de CPU desejada em cada estágio, a taxa de requisições foi definida com base nas leis operacionais [Menasce et al. 2004], mais especificamente na lei da utilização: $\rho = \lambda * S$, onde ρ é a intensidade da carga, λ a taxa de chegada e S o tempo de serviço. Assim, a taxa de requisições foi definida com valores variando em: $\lambda = 2$ (que exige $\rho = 0,35$ CPU-cores); $\lambda = 4$ (que exige $\rho = 0,7$ CPU-cores); $\lambda = 6$ (que exige $\rho = 1,05$ CPU-cores); e $\lambda = 8$ (que exige $\rho = 1,4$ CPU-cores).

Os dois primeiros estágios de todos os cenários são de aquecimento (*warm-up*), abrangendo 4 minutos e com taxa fixa de $\lambda = 2$ rps, para que as abordagens de autoscaling se ajustem em uma configuração inicial. Os quatro cenários avaliados pretendem avaliar os mecanismos de autoscaling com as seguintes características:

1. $\lambda = [2, 2, 4, 6, 8, 6, 4, 2, 2]$ – a carga aumenta e diminui de forma mais gradual.
2. $\lambda = [2, 2, 8, 8, 8, 2, 2, 2, 2]$ – a carga aumenta de maneira abrupta para um valor alto e se mantém neste pico por 3 estágios; em seguida diminui de forma abrupta para o valor mínimo e se mantém assim até o fim do teste.
3. $\lambda = [2, 2, 8, 8, 2, 2, 2, 2, 2]$ – similar ao cenário 2, com aumento/diminuição de carga abruptos, mas se mantém no pico de carga durante menos tempo (2 estágios).
4. $\lambda = [2, 2, 8, 2, 8, 2, 8, 2, 2]$ – carga alterna entre muito alta e muita baixa com frequência, com apenas um estágio em cada intensidade de carga, para representar um cenário com alta variação.

Na época que os testes foram realizados, o vertical pod autoscaling (VPA) do Kubernetes estava em sua versão de testes beta, enquanto o horizontal pod autoscaling (HPA) já era considerado um produto mais estável. Um *pod* é a menor unidade de alocação em um nó, que pode empacotar um ou mais contêineres e definir regras de execução. Tanto a abordagem de VPA quanto a de horizontal pod autoscaling HPA do Kubernetes iniciam o teste com dois pods com capacidade de CPU de $0,15$ CPU-cores.

Para executar os experimentos em cada estágio, foi desenvolvido um script Shell que configura o ambiente e executa o benchmark.

Durante a execução dados são coletados pela API do Kubernetes: 1) utilização de CPU e memória do cluster; 2) recomendações de capacidade feitas pelo autoscaler; e 3) quantidades de CPU e memória requisitadas pelos pods. Essas coletas são realizadas a cada 10 segundos e salvas em arquivos de *logs* de forma estruturada. As métricas específicas analisadas foram:

- a capacidade de CPU requisitada por cada pod ao longo do tempo, de acordo com as decisões dos mecanismos de autoscaling;
- o tempo de resposta da aplicação ao longo do tempo, indicando o desempenho da mesma e a eficácia dos mecanismos de autoscaling em rapidamente redimensionar os pods para atender à demanda.

4. Resultados

Os quatro cenários de experimentos foram executados para cada mecanismo de autoscaling. Todos iniciam com 2 pods, cada um com $0,15$ CPU-cores, que vão sendo redimensionados pelo autoscaler ao longo do tempo. As Figuras 2 e 3 mostram a CPU requisitada por cada pod ao longo do tempo, para os cenários do autoscaling vertical (VPA) e horizontal (HPA), respectivamente. A linha tracejada indica a capacidade de CPU necessária para atingir 100% de utilização durante cada estágio de carga.

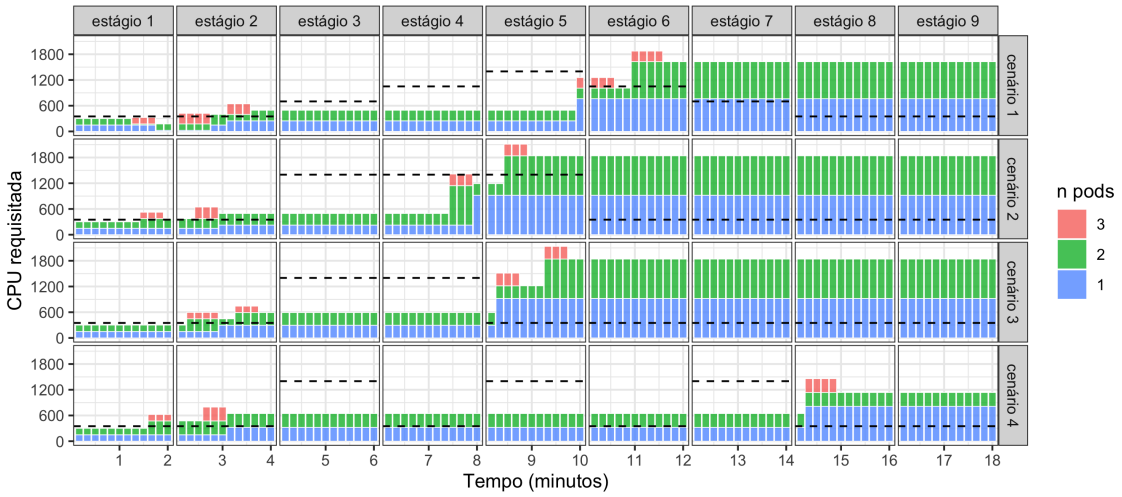

Figura 2. CPU requisitada por pods para cenários do autoscaling vertical.

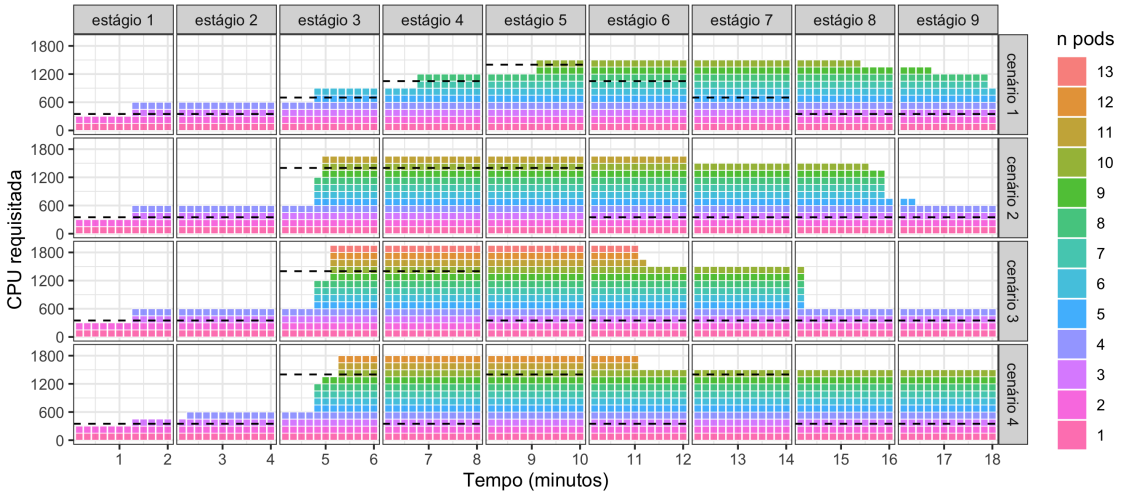

Figura 3. CPU requisitada por pods para cenários do autoscaling horizontal.

Para o VPA, observou-se um grande atraso no redimensionamento dos recursos, ficando parte do tempo com a CPU requisitada abaixo da capacidade necessária (barras coloridas abaixo da linha tracejada). A demora nas decisões foi maior no cenário 1, onde a carga cresce gradativamente, e menor nos cenários 2 e 3, onde a mudança de carga é mais abrupta. No cenário 4, como picos frequentes de curta duração, a provisão de recursos só foi efetivada no estágio 8. Ao aumentar os recursos (*upscale*) o VPA geralmente requisita um pouco mais de CPU do que a capacidade necessária para obter a utilização de 100%. Já ao diminuir os recursos (*downscale*) ele é mais conservador, só sendo ativado em um estágio de warm-up no cenário 1. Mesmo nos 5 últimos estágios seguidos de carga baixa constante do cenário 3 o VPA não realizou *downscale*, gerando super-provisão de recursos desnecessária. A demora para fazer o redimensionamento se deve a um fator de confiança deste mecanismo, que requer um tempo de execução maior para ter mais confiança nas recomendações. A existência de 3 pods em alguns momentos ocorre porque quando há um redimensionamento de recursos, ele cria um novo pod com a nova capacidade recomendada e só finaliza o anterior quando o novo estiver pronto para receber requisições.

Já o HPA, na maior parte do tempo, conseguiu reagir de forma efetiva à mudança na carga e manteve a CPU requisitada um pouco acima da linha da capacidade de CPU necessária. Ele teve em média uma demora de 40 segundos para decidir redimensionar os recursos quando a carga aumenta. Apenas nas transições de aumento de carga do estágio 3 para todos os cenários e nos estágios 4 e 5 do cenário 1 que a CPU requisitada ficou abaixo do necessário por aproximadamente 1 minuto. Apesar do *downscale* ser mais conservador e demorar um pouco mais para ser efetivado (aproximadamente 5 minutos), o HPA chegou a reduzir a quantidade de pods em todos os cenários, ao contrário do VPA que não fez *downscale*. Por fazer o *downscale* mais conservador, no cenário 4 que tem uma grande variação de carga, o HPA tende a manter a CPU requisitada super-provisionada. Manter a requisição de CPU sempre acima do esperado é algo positivo por conseguir lidar com cargas de trabalho que mudam com frequência e em curtos períodos de tempo, porém, a longo prazo, pode impactar nos custos da infraestrutura.

A Figura 4 mostra boxplots comparando o tempo de resposta das requisições feitas ao servidor web ao longo dos estágios de carga de cada cenário. A linha no meio de cada caixa representa a mediana, enquanto o ponto representa a média do tempo de resposta em cada estágio. Observa-se que o autoscaling horizontal apresentou tempos de resposta muito próximos do tempo de serviço (0,175 segundos) em quase todos os cenários, tendo apenas a média e o 3o quartil um pouco maiores em alguns estágios de aumento de carga, mas mantendo a mediana sempre baixa. Por outro lado, o autoscaling vertical apresentou tempos de resposta muito acima do tempo de serviço em muitos estágios, tanto para a média quanto para os quartis, devido à demora em redimensionar os pods.

Percebe-se então que, no estágio atual de desenvolvimento do Kubernetes e com as configurações padrão dos mecanismos de autoscaling, o HPA se mostrou mais efetivo em responder rapidamente às mudanças na carga com uma quantidade de pods adequada para atender as requisições, enquanto o VPA foi impactado negativamente pela demora em redimensionar os seus pods.

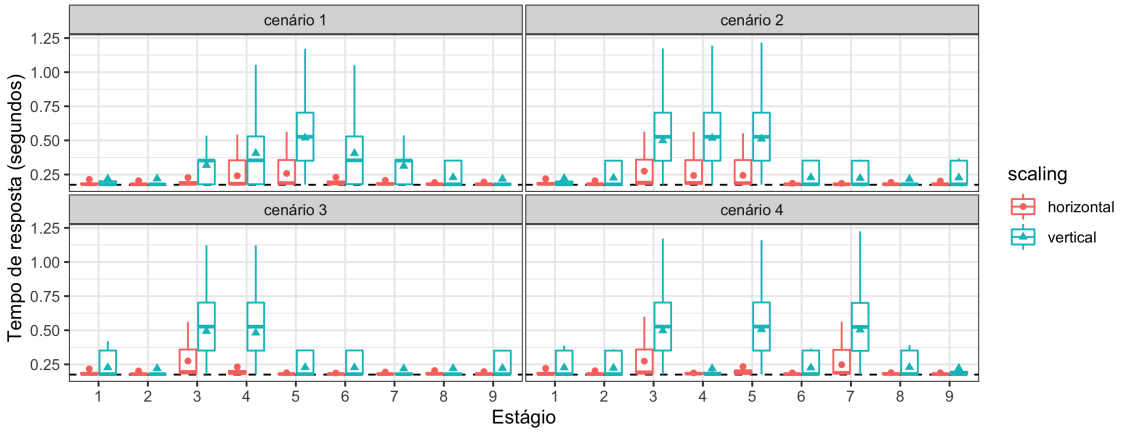

Figura 4. Boxplot do tempo de resposta do autoscaling vertical e horizontal.

5. Conclusão

Este trabalho teve como objetivo realizar uma análise de desempenho comparativa entre os mecanismos de autoscaling vertical e horizontal do Kubernetes, através de experimentos de medição. Para isto, foi necessário criar uma forma de geração e controle de cargas de trabalho através de uma ferramenta de benchmarking e de uma aplicação web, investigar detalhes de implementação das funcionalidades de autoscaling do Kubernetes e criar cenários de experimentos para analisar o comportamento do autoscaling vertical e horizontal levando em consideração métricas de tempo de resposta, requisição de CPU dos Pods e tempo entre eventos de autoscaling.

A partir dos experimentos realizados neste trabalho, percebe-se que o autoscaling horizontal é menos conservador e mais efetivo no redimensionamento de recursos nos cenários avaliados. É importante lembrar que essa precisão deve ser motivada pela objetividade do algoritmo usado para acionar o autoscaling, que é basicamente manter a requisição de recursos na média do limite de utilização de recursos definido.

Em contrapartida, o auto scaling vertical é mais conservador nas decisões de provisão de recursos, pois conta com a lógica de aumentar o fator de confiança de acordo com o tempo. Acredita-se que em experimentos mais longos, onde seja possível manter um histórico maior de execução dos pods, o autoscaling vertical será mais efetivo em suas decisões de provisão de recursos.

Com base nos cenários e variáveis analisadas neste trabalho, a abordagem horizontal se mostrou mais eficiente em relação à vertical devido à sua precisão nas decisões de autoscaling, agilidade para redimensionamento de recursos e consequentemente ao baixo tempo de resposta da aplicação na maior parte do tempo. Vale salientar que o autoscaling vertical ainda está em fase beta no Kubernetes e que está recebendo constantes melhorias, o que pode torná-lo mais efetivo no futuro. Também é importante destacar que foram usadas as configurações padrão do Kubernetes para os dois mecanismos de autoscaling e um melhor ajuste de parâmetros pode gerar melhores resultados.

Como trabalhos futuros, pretende-se avaliar versões mais recentes do Kubernetes, com cargas reais de datasets públicos e em infraestruturas de maior escala, além de investigar uma estratégia que escolhe o melhor tipo de autoscaling em diferentes situações.

Referências

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., and Merle, P. (2017). Autonomic vertical elasticity of docker containers with elasticdocker. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 472–479.

Casalicchio, E. (2017). Autonomic orchestration of containers: Problem definition and research challenges. ACM.

Casalicchio, E. and Perciballi, V. (2017). Auto-scaling of containers: The impact of relative and absolute metrics. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 207–214.

Jiang, J., Lu, J., Zhang, G., and Long, G. (2013). Optimal cloud resource auto-scaling for web applications. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 58–65.

Liu, C., Shie, M., Lee, Y., Lin, Y., and Lai, K. (2014). Vertical/horizontal resource scaling mechanism for federated clouds. In *2014 International Conference on Information Science Applications (ICISA)*, pages 1–4.

Mao, M. and Humphrey, M. (2013). Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, page 67–78, USA. IEEE Computer Society.

Mao, M., Li, J., and Humphrey, M. (2010). Cloud auto-scaling with deadline and budget constraints. In *2010 11th IEEE/ACM International Conference on Grid Computing*.

Menasce, D. A., Dowdy, L. W., and Almeida, V. A. F. (2004). *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, USA.

Netto, H. V., Lung, L. C., Correia, M., Luiz, A. F., and Souza, L. M. (2017). State machine replication in containers managed by Kubernetes. *Journal of Systems Architecture*.

Qu, C., Calheiros, R. N., and Buyya, R. (2018). Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Comput. Surv.*, 51(4).

Sedaghat, M., Hernandez-Rodriguez, F., and Elmroth, E. (2013). A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC '13, New York, NY, USA. Association for Computing Machinery.

Tirmazi, M., Barker, A., Deng, N., Haque, M. E., Qin, Z. G., Hand, S., Harchol-Balter, M., and Wilkes, J. (2020). Borg: the next generation. In *EuroSys'20*, Heraklion, Crete.

Tosatto, A., Ruiu, P., and Attanasio, A. (2015). Container-based orchestration in cloud: State of the art and challenges. In *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 70–75.

Yazdanov, L. and Fetzer, C. (2012). Vertical scaling for prioritized vms provisioning. In *2012 Second International Conference on Cloud and Green Computing*.