Análise de Implementação de Servidor Web usando Kubernetes com Autoescalonamento Horizontal em Ambiente Local e na Nuvem AWS

Douglas Bellomo Cavalcante[1], **Elielder Belchior de Melo**[1]

[1]Escola Politécnica - Universidade de São Paulo (EP-USP)

douglas.b.cavalcante@gmail.com, elielder@usp.br

Resumo. *Este artigo analisa uma implementação de um servidor web usando Kubernetes com autoescalonamento horizontal (HPA) nos ambientes local bare-metal, virtualizado e em nuvem Amazon AWS utilizando o recurso Amazon Elastic Kubernetes Service (EKS). Foram realizados testes de carga para acionamento do autoescalamento horizontal e coletados os tempos para estabilização da carga de uso da CPU, bem como o número máximo de réplicas criadas. Foi observado que o ambiente baremetal apresentou melhor desempenho em relação ao virtualizado, tendo estabilizado com uso de um pod a menos. O ambiente em nuvem apresentou menor tempo para estabilização. Conclui-se que a decisão entre implementação local ou na nuvem deve considerar o equilíbrio entre a necessidade de controle e otimização de recursos oferecida pelas soluções locais, a flexibilidade e escalabilidade proporcionada pela nuvem e a relação custo-benefício da solução.*

Abstract. *This article analyzes the implementation of a web server using Kubernetes with horizontal pod autoscaling (HPA) in local bare-metal, virtualized, and Amazon AWS cloud environments, utilizing the Amazon Elastic Kubernetes Service (EKS) feature. Load tests were conducted to trigger horizontal autoscaling, and the times for CPU usage stabilization and the maximum number of replicas created were recorded. The bare-metal environment performed better than the virtualized one, stabilizing with one fewer pod. The cloud environment showed a shorter time for stabilization. It is concluded that the decision between local or cloud implementation should consider the balance between the need for control and resource optimization offered by local solutions, the flexibility and scalability provided by the cloud, and the cost-benefit of the solution.*

1. Introdução

Kubernetes é uma plataforma open-source projetada para automatizar a implantação, o gerenciamento e a escalabilidade de aplicações em contêineres. A escalabilidade pode ser realizada com o autoescalonamento horizontal de *pods* (*Horizontal Pod Autoscaling* – HPA), que é um mecanismo que ajusta dinamicamente o número de réplicas de uma aplicação com base em métricas de desempenho, como a utilização de CPU e memória [Nunes et al. 2021]. Esse processo permite que as aplicações mantenham uma alocação de recursos adequada à demanda em tempo real, respondendo eficientemente a picos de carga sem a necessidade de intervenção manual. O HPA monitora continuamente essas métricas por meio de componentes integrados ao cluster, como o *Metrics Server*, e, com

base em regras configuradas, decide se é necessário aumentar ou diminuir o número de *pods* em execução. O ajuste de réplicas é realizado periodicamente, garantindo que a aplicação possa lidar com aumentos repentinos de tráfego e, ao mesmo tempo, otimizar o uso dos recursos quando a demanda é menor [Kubernetes 2024a].

Para aplicações na nuvem, o recurso Kubernetes é encontrado no serviço Amazon Web Services (AWS) com o Amazon Elastic Kubernetes Service (EKS), que é um serviço gerenciado que simplifica a execução de aplicações em contêineres utilizando Kubernetes. O EKS oferece alta disponibilidade ao distribuir clusters por múltiplas zonas de disponibilidade, minimizando o risco de interrupções e garantindo a resiliência das aplicações. A escalabilidade, com suporte para escalonamento automático, permite que as aplicações ajustem sua capacidade conforme a demanda, de maneira eficiente [Amazon 2024a].

O balanceamento de carga é uma técnica fundamental na computação em nuvem, pois distribui eficientemente o tráfego e as cargas de trabalho entre múltiplos servidores ou recursos. Esta abordagem é essencial para garantir a escalabilidade e a resiliência das aplicações, mantendo alta disponibilidade e desempenho otimizado, mesmo diante de picos de demanda ou falhas de componentes [Gaur et al. 2019, Bundela et al. 2022].

Esse artigo avalia o uso do HPA em três ambientes distintos e compara os resultados obtidos quanto ao desempenho dessa funcionalidade. A seção 2 demonstra a implementação dos serviços nos ambientes *baremetal*, virtualizado e em nuvem Amazon AWS, enquanto que a seção 3 apresenta a análise dos resultados obtidos. A seção 4 apresenta as conclusões obtidas a partir da análise da documentação, de literaturas relacionadas ao tema e dos resultados obtidos.

2. Implementação de Servidor Web com Kubernetes

As implementações Kubernetes de um servidor *web* com autoescalonamento horizontal, em ambientes local, virtualizado e em nuvem Amazon AWS estão descritas nas seções a seguir.

2.1. HPA Local

A implementação do HPA em máquina local foi baseada no tutorial oficial do Kubernetes [Kubernetes 2024a]. Foi utilizado o *minikube* para criação do cluster e o *kubectl* para gerenciamento. O *minikube* é uma ferramenta que permite executar um cluster Kubernetes localmente, facilitando o desenvolvimento e o teste de aplicações Kubernetes. Compatível com Linux, macOS e Windows, o Minikube cria um cluster com apenas um nó, suportando a maioria dos recursos do Kubernetes. Ele utiliza diferentes drivers para gerenciar máquinas virtuais (VMs) ou contêineres, e permite a ativação de diversos add-ons para adição novas de funcionalidades. O *kubectl* é uma ferramenta de linha de comando para interagir e gerenciar clusters Kubernetes.

A implementação passo-a-passo do processo pode ser vista a seguir, sendo que foram utilizados computadores Lenovo Thinkpad E-14 Gen 2 com processador Intel i7 e 16 GB de RAM com sistema operacional Linux Ubuntu.

2.1.1. Instalação do *minikube* e do *kubectl*

O *minikube* é uma ferramenta disponibilizada pelos desenvolvedores do Kubernetes, voltada para ambientes de aprendizado e desenvolvimento [Minikube 2024]. A ferramenta não está disponível nos repositórios oficiais do Debian ou do Ubuntu e, seguindo as recomendações de seus desenvolvedores, deve ser baixada diretamente do *site* do *minikube*. Tem como principal dependência ou um gerenciador de máquinas virtuais, como o VirtualBox ou o KVM, ou de contêineres, como o Docker.

Para utilizar o *minikube* sobre ambiente Docker deve-se executar o comando `$ minikube start --driver=docker`. Observa-se que o argumento `driver=` permite especificar o ambiente de execução. O minikube se responsabilizará por efetuar os downloads e configurações necessárias para iniciar um cluster Kubernetes local com um nó. O *minikube* disponibiliza, ainda, a possibilidade amplicar suas funcionalidades pela instalação de *add-ons*.

Assim como o *minikube*, o *kubectl* também não está presente nos repositórios oficiais do Debian ou do Ubuntu, sendo que a instalação pode feita baixando o executável a partir do *site* da Kubernetes [Kubernetes 2024b]. Dentre suas várias possibilidades, o gerenciador permite efetuar a implementação de aplicações configuradas em arquivos do tipo *.yaml* com o comando `$ kubectl apply -f ARQUIVO_APLICAÇÃO.yaml`. Permite também efetuar o apagamento de aplicações com o comando `$ kubectl delete NOME_APLICAÇÃO` ou obter informações de pods instalados com o comando `$ kubectl get NOME_POD`. A documentação completa está disponível no site do Kubernetes.

As implementações partiram da instalação das ferramentas *minikube* e *kubectl* em ambientes *baremetal*[1] e virtualizado[2], sendo que toda a documentação que demonstra as instalações e configurações está disponível no repositório do projeto no Github[3]. A aplicação *web server php-apache* foi utilizada para testes, sendo ela disponibilizada para fins de aprendizado pelos desenvolvedores do Kubernetes.

2.1.2. Habilitando o Autoescalonamento Horizontal – HPA

O HPA é uma funcionalidade do Kubernetes que permite o escalonamento automático de pods de uma aplicação por meio da criação ou deleção de pods baseados em métricas pré-determinadas, como uso de CPU ou de memória. Sua ativação pode ser feita pelo *kubectl* utilizando o comando `$ kubectl autoscale ARGUMENTOS`, ou ainda utilizando um arquivo de configuração do tipo *.yaml*, conforme documentação em [Kubernetes 2024a]. O comando `$ kubectl get hpa NOME_APLICAÇÃO` permite verificar o status do HPA criado para a aplicação, indicando a carga atual e o número de pods em execução.

Para o funcionamento correto do HPA há a necessidade de habilitar no *minikube* o *add-on* **Metrics Server**. Esse *add-on* instala uma aplicação que mo-

[1] https://github.com/douglasbcavalcante/cluster_kubernetes/blob/main/elielder.md
[2] https://github.com/douglasbcavalcante/cluster_kubernetes/blob/main/douglas.md
[3] https://github.com/douglasbcavalcante/cluster_kubernetes/tree/main

nitora os recursos dos pods do cluster, de maneira a informar ao HPA o estado atual do cluster para que ele tome a decisão de escalonamento. Seu comando de instalação é o `$ minikube addons enable metrics-server`. Em ambiente de produção, sem uso do *minikube*, esse add-on deve ser instalado manualmente conforme documentação do Kubernetes.

Ambos os ambientes de testes (*baremetal* e virtualizado) executaram a aplicação *php-apache* e habilitaram o HPA para escalonamento baseado em CPU com taxa-alvo de 50 %, número mínimo de 1 pod e máximo de 10 pods.

2.1.3. Teste de Escalonamento Horizontal

A documentação do Kubernetes disponibiliza uma forma simples de gerar carga para teste do HPA, que consiste em criar um pod que executa uma requisição http do tipo get a cada 0.01 s, em *loop* infinito [Kubernetes 2024a]. A grande quantidade de requisições faz com que a taxa de processamento da aplicação alvo (*php-apache*) cresça a valores que obriguem o HPA agir. Importante observar que o pod gerador de carga deve estar no mesmo cluster que a aplicação alvo, de forma que os mecanismos de proteção contra DoS (*deny-of-service*) não inviabilizem o teste. As Figuras 1 e 2 mostram o comportamento do HPA para os ambientes *baremetal* e virtualizado, respectivamente.

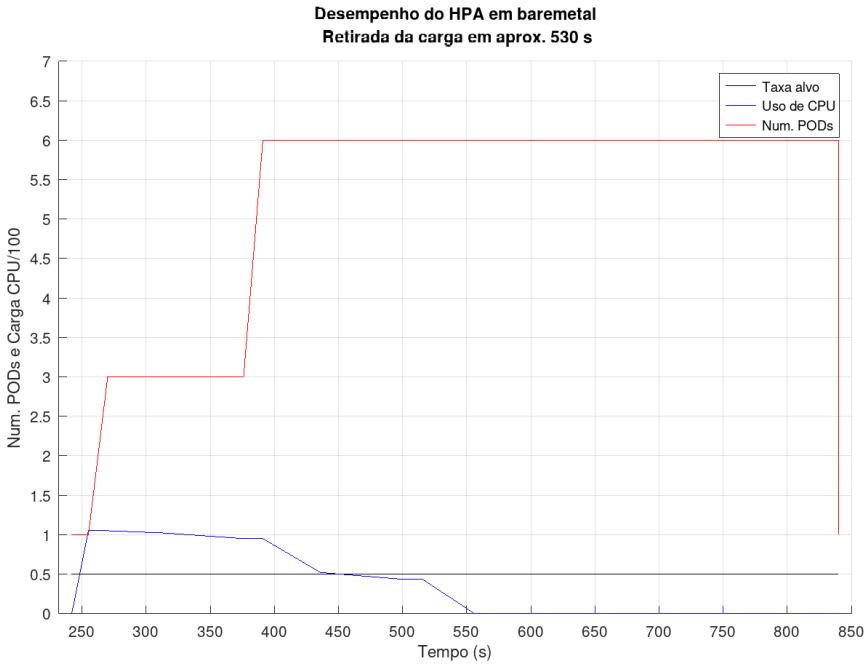

Figura 1. Resultados do teste de carga com kubernetes em ambiente *baremetal*

2.2. HPA na Amazon EKS

O Amazon Elastic Kubernetes Service é o serviço de nuvem da Amazon AWS para implementação de clusters Kubernetes escaláveis, abstraindo a necessidade de criação e gerenciamento da infraestrutura de rede e servidor. A documentação de criação e gerenciamento dos clusters no Amazon EKS está disponível no *site* da AWS [Amazon 2024b].

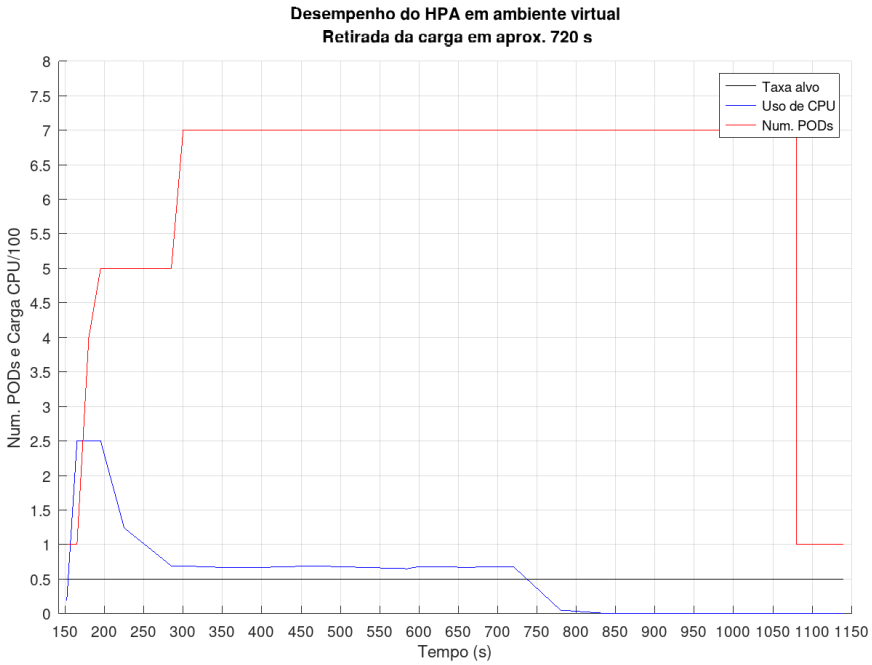

Figura 2. Resultado do teste de carga com kubernetes em ambiente virtualizado

A nuvem Amazon AWS permite a criação de clusters no EKS de diferentes maneiras. Pode-se criá-los usando a interface *web* de gerenciamento da Amazon EKS (Figura 3, o utilitário *eksctl*, disponibilizado pela Amazon AWS, ou ainda o utilitário de gerenciamento em nuvem *aws cli*. Os testes realizados utilizaram a ferramenta *aws cli* e a interface de gerenciamento em nuvem.

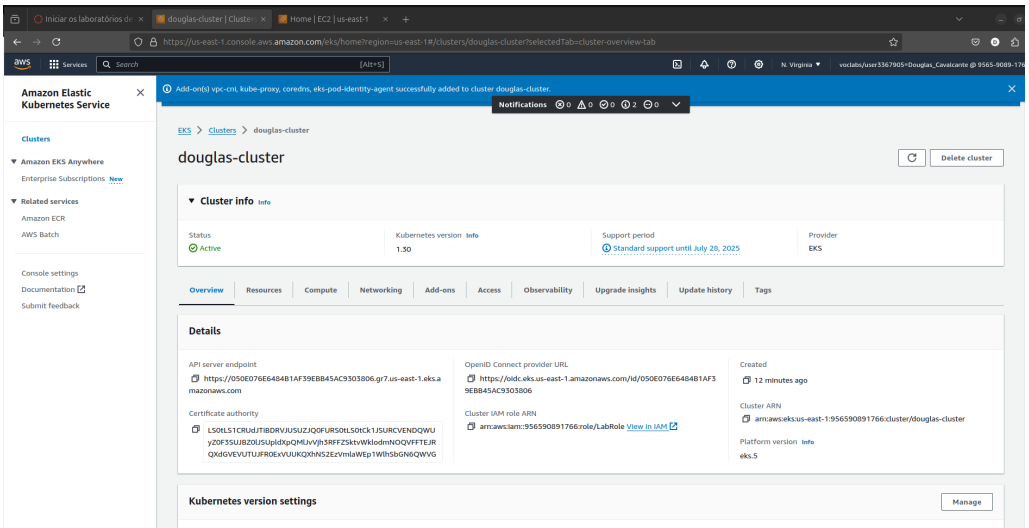

Figura 3. Interface de gerenciamento e criação de clusters na Amazon EKS

A conta utilizada para criação do cluster na Amazon EKS, bem como para execução dos testes, foi a disponibilizada pela Amazon para os alunos da disciplina PSI5120-2024. Essa conta possui limitações de uso e não permite a criação de usuários e perfis (*roles*), sendo que para os testes foram utilizados os usuários disponibilizados aos

autores e a *role* IAM LabRole.

2.2.1. Criação do Cluster na Amazon EKS

A criação do cluster se inicia com a formação de uma nuvem privada virtual (VPC) com subredes privadas que atendem os requisitos da Amazon EKS. Para isso, executa-se o comando `$ aws cloudformation create-stack` com argumentos de região (utilizada a *us-east-1*), nome da VPC e *template* (utilizado o padrão sugerido no tutorial [Amazon 2024b]).

Uma vez criada a VPC, cria-se o cluster utilizando a interface *web* de gerenciamento da nuvem Amazon AWS, conforme mostrado na Figura 3. Sugere-se, então, configurar o ambiente para operação a partir da máquina local, criando um contexto de configuração do *kubectl* para operação remota do cluster. Faz-se isso utilizando o comando `$ aws eks update-kubeconfig`, informando a região e o nome do cluster.

2.2.2. Criação dos Nós do Cluster na Amazon EC2

A Amazon EKS fornece duas opções para criação de nós para o cluster, uma utilizando máquinas virtuais criadas no serviço Amazon Elastic Compute Cloud (EC2), e outra utilizando o Amazon Fargate. O Amazon Fargate é um serviço de computação *serverless* que permite o *deploy* de aplicações Kubernetes sem necessidade de uso de instânicas do EC2. Para os testes realizados nesse artigo foram utilizadas instâncias do EC2. Os nós do cluster são criados a partir do seu próprio ambiente de configuração na interface de gerenciamento (aba *Compute* do Amazon EKS). Para os testes em questão foram criadas duas instâncias do tipo *t3.medium* com quantidades padrões de cpu e memória.

2.2.3. Deploy da Aplicação e Configuração do HPA

O *deploy* da aplicação de testes *php-apache* e a configuração do HPA seguiram exatamente os procedimentos relatados na Seção 2.1.2, depois de criados os nós e configurado o *kubectl*, conforme explicado na subseção anterior.

2.2.4. Teste de Escalonamento Horizontal

Da mesma forma que descrito na Seção 2.1.3, o teste de escalonamento horizontal consistiu na criação de um pod que efetuava requisições http em intervalos de 0,01 ms, sendo que esse pod foi criado no cluster na Amazon EKS conforme o procedimento do *site* da Amazon AWS [Amazon 2024c]. A Figura 4 mostra o comportamento do HPA obtido o ambiente Amazon EKS de um dos autores[4].

[4]Consultar o repositório do projeto para visualização dos testes realizados por ambos os autores.

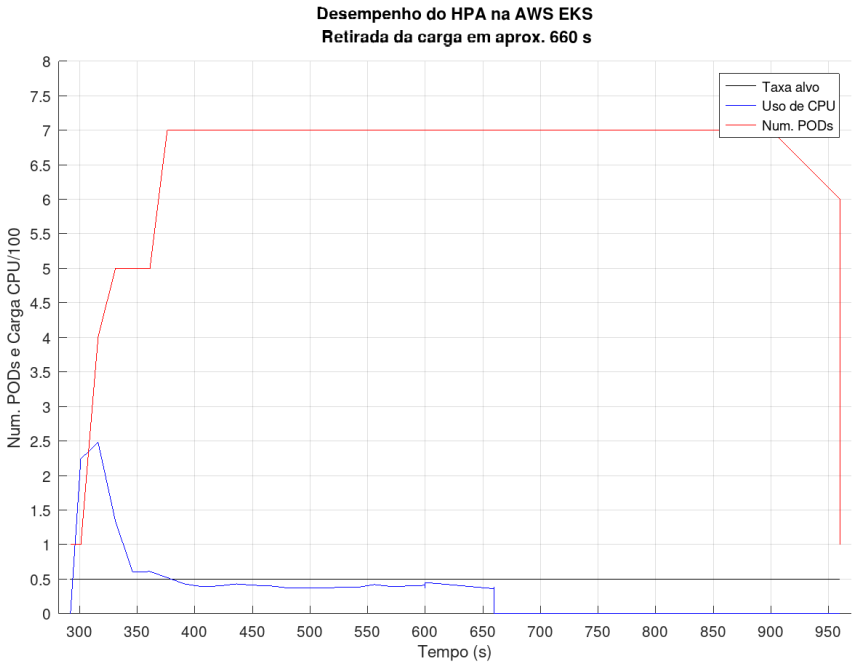

Figura 4. Resultados do teste de carga com Kubernetes na Amazon EKS

3. Análise de Resultados

A Tabela 1 mostra os resultados obtidos para os três ambientes implementados, contendo a média de carga percentual da CPU durante o período de maior número de réplicas e o tempos de criação e de remoção dos pods, dados esses também representados nas Figuras 1, 2 e 4.

Tabela 1. Respostas do HPA nas três condições testadas

Ambiente	Média % CPU	Criação pod (s)	Rem. pods	Máx réplicas
Baremetal	46,7	149	325	6
Ambiente virtual	67,4	148	240	7
Amazon AWS/EKS	40,7	84	300	7

Observa-se que, para as implementações em ambiente local, a média da carga de CPU em *baremetal* é menor em relação ao ambiente virtualizado, que, por sua vez, tem um tempo remanescente de réplicas menor quando a carga no servidor *web* é removida. A implementação em nuvem Amazon AWS com EKS manteve cargas menores em relação aos ambientes locais e apresentou tempo para criação das réplicas 43,2 % menor que os obtidos nos ambientes locais, porém o tempo de remoção foi similar (cabe ressaltar que a documentação da Amazon EKS ressalta que a resposta à redução de carga pode levar até 5 minutos [Amazon 2024c]). Ressalta-se, ainda, que o ambiente em *baremetal* conseguiu estabilizar a carga de CPU perto da meta de 50 % com uma réplica a menos.

Embora a plataforma Kubernetes tenha o mesmo processo de instalação para as implementações locais, que se utilizam de ambiente *baremetal* e virtualizado, o comportamento de evolução da carga e das réplicas pode ser distinto. A implementação de um

servidor *web* em Kubernetes em uma infraestrutura *baremetal* tende a oferecer desempenho superior, pois elimina a camada de virtualização e permite acesso direto aos recursos de hardware, podendo resultar em menor latência e maior *throughput*. Esse comportamento se mostrou presente nos testes realizados, conforme pode ser visto na Tabela 1, onde o mesmo desempenho foi obtido com um pod a menos. No entanto, abordagem *baremetal* pode ser mais complexa na configuração inicial e no gerenciamento contínuo do hardware. Por outro lado, ao utilizar um ambiente virtualizado tende-se haver uma perda de desempenho devido à virtualização, mas ganha-se em flexibilidade e isolamento. A virtualização facilita a criação, destruição e migração de máquinas virtuais, o que pode ser vantajoso em ambientes dinâmicos.

Já na implementação do mesmo servidor *web* em Kubernetes utilizando Amazon AWS/EKS sobre instâncias EC2 combina a flexibilidade e o poder da infraestrutura em nuvem com o gerenciamento simplificado do Kubernetes, o que permite mitigar possíveis perdas de desempenho. O Amazon AWS/EKS gerencia a configuração e o controle da camada de Kubernetes, permitindo que as equipes se concentrem no desenvolvimento e na operação das aplicações sem se preocupar com a complexidade da gestão do cluster. As instâncias EC2 fornecem a capacidade de escalar automaticamente com base na demanda, que simplifica a implementação no geral.

4. Conclusão

Considerando as implementações realizadas, a escolha da infraestrutura para um servidor *web* em Kubernetes envolve uma análise cuidadosa das necessidades de desempenho, flexibilidade, escalabilidade e custos. A implementação em *baremetal* proporciona desempenho máximo com acesso direto ao hardware, sendo ideal para cargas de trabalho que exigem latência mínima. Contudo, essa abordagem pode ser mais complexa de gerenciar e menos flexível em termos de escalabilidade rápida. Por outro lado, a virtualização oferece maior flexibilidade e isolamento, permitindo um gerenciamento mais ágil das máquinas virtuais, embora com uma penalidade de desempenho devido ao *overhead* da virtualização.

Por outro lado, a implementação em nuvem, utilizando o Amazon AWS/EKS em instâncias EC2 destaca-se por permitir que os recursos se ajustem dinamicamente à demanda de forma tecnicamente ilimitada, o que é essencial para aplicações com cargas de trabalho variáveis. Percebeu-se pelos resultados dos testes que as variações de desempenho necessariamente levam a análises de custo-benefício para a melhor escolha.

A decisão entre implementação local ou na nuvem deve considerar o equilíbrio entre a necessidade de controle e otimização de recursos oferecida pelas soluções locais e a flexibilidade e escalabilidade proporcionada pela nuvem.

Referências

[Amazon 2024a] Amazon (2024a). Deploy a sample application - Amazon EKS. https://docs.aws.amazon.com/eks/latest/userguide/sample-deployment.html.

[Amazon 2024b] Amazon (2024b). Getting started with amazon EKS – AWS management console and AWS CLI - amazon EKS. https://docs.aws.amazon.com/eks/latest/userguide/getting-started-console.html.

[Amazon 2024c] Amazon (2024c). Scale pod deployments with horizontal pod autoscaler - amazon EKS. https://docs.aws.amazon.com/eks/latest/userguide/horizontal-pod-autoscaler.html.

[Bundela et al. 2022] Bundela, R., Dhanda, N., and Verma, R. (2022). Load Balanced Web Server on AWS Cloud.

[Gaur et al. 2019] Gaur, M., Kumar, A., and Dadheech, P. (2019). Enhancement of Auto Scaling and Load Balancing using AWS. 9.

[Kubernetes 2024a] Kubernetes (2024a). Horizontal Pod Autoscaler Walkthrough. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/.

[Kubernetes 2024b] Kubernetes (2024b). Install and Set Up kubectl on Linux. https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/.

[Minikube 2024] Minikube (2024). Minikube start. https://minikube.sigs.k8s.io/docs/start/.

[Nunes et al. 2021] Nunes, J. P. K. S., Bianchi, T., Iwazaki, A. Y., and Nakagawa, E. Y. (2021). State of the Art on Microservices Autoscaling: An Overview.