Implantação de um servidor web em um cluster KUBERNETES com auto escalamento horizontal automático

Jean Carlos Mello Xavier Faria[1], **Jones Egydio**[1], **Ricardo Grobman**[1],

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

jeanmello@usp.br, jones.egydio@usp.br, rgrobman@usp.br

Abstract. *This article explores two different approaches to Kubernetes cluster implementation: Minikube, a local tool that enables running Kubernetes on a single machine, and AWS Elastic Kubernetes Service (EKS), a managed solution provided by Amazon Web Services. The article discusses the features, architectures, deployment, advantages, limitations, and use cases of each solution. Minikube is highlighted as a cost-effective and practical option for local development and testing, while AWS EKS offers a robust and scalable platform suitable for production environments. The comparative analysis aims to guide developers and DevOps engineers in choosing the most appropriate solution for their needs.*

Resumo. *Este artigo explora duas abordagens distintas para a implementação de clusters Kubernetes: Minikube, uma ferramenta local que permite a execução de Kubernetes em uma única máquina, e AWS Elastic Kubernetes Service (EKS), uma solução gerenciada pela Amazon Web Services. São discutidas as características, arquiteturas, implantação, vantagens, limitações e casos de uso de cada solução. Minikube se destaca como uma opção econômica e prática para desenvolvimento e testes locais, enquanto o AWS EKS oferece uma plataforma robusta e escalável adequada para ambientes de produção. A análise comparativa visa orientar desenvolvedores e engenheiros de DevOps na escolha da solução mais adequada às suas necessidades.*

1. Introdução

Nos últimos anos, a contêinerização de aplicações se tornou uma prática comum no desenvolvimento de software, permitindo maior flexibilidade e consistência no gerenciamento de ambientes de desenvolvimento, teste e produção. Kubernetes, originalmente desenvolvido pela Google, é uma plataforma de orquestração de contêineres de código aberto que automatiza a implantação, o dimensionamento e o gerenciamento de aplicações em contêineres.

Com o aumento da adoção de Kubernetes, surgiram diversas soluções para implementação de clusters, desde ambientes locais para desenvolvimento, como o Minikube, até plataformas de nuvem gerenciadas, como o AWS Elastic Kubernetes Service (EKS). Cada uma dessas abordagens oferece vantagens e desafios específicos, que devem ser considerados de acordo com as necessidades do projeto. Este artigo explora em detalhes duas dessas implantações, fornecendo uma análise comparativa que pode orientar a escolha da solução mais adequada para diferentes cenários.

2. Cluster Kubernetes usando Minikube

2.1. O que é Minikube?

Minikube é uma ferramenta de código aberto que permite executar Kubernetes localmente em uma única máquina. Ele é amplamente utilizado para desenvolvimento e testes de aplicações em Kubernetes, oferecendo uma maneira rápida e simples de criar um ambiente que simula um cluster Kubernetes real. Minikube suporta a maioria dos recursos que um cluster completo oferece, incluindo o gerenciamento de pods, serviços, volumes e ingressos [Poulton 2021].

2.2. Características e Arquitetura do Minikube

A arquitetura do Minikube é composta por um nó único que executa todos os componentes principais de um cluster Kubernetes, como o servidor API, etcd (o armazenamento de dados do Kubernetes), controlador e scheduler. Este nó é geralmente criado em uma máquina virtual utilizando drivers como VirtualBox, VMware, ou diretamente em containers com Docker. Essa simplicidade de configuração torna o Minikube ideal para experimentação e desenvolvimento local, permitindo que desenvolvedores testem suas aplicações em um ambiente Kubernetes antes de implantá-las em um cluster de produção.

Ademais, o Minikube oferece suporte para múltiplas versões do Kubernetes, permitindo que os desenvolvedores testem suas aplicações em diferentes versões da plataforma. Ele também fornece uma interface de linha de comando fácil de usar, que facilita a gestão do cluster, como iniciar, parar, reiniciar e alterar a configuração do Kubernetes [Poulton 2021].

2.3. Implementação de um Cluster Kubernetes com Minikube

A implementação de um cluster Kubernetes com Minikube é um processo direto que envolve, resumidamente, as seguintes etapas:

- **Instalação do Minikube e Driver de Virtualização:** Minikube pode ser instalado em várias plataformas, como Windows, macOS e Linux. O driver de virtualização (como VirtualBox ou Docker) também deve ser instalado para permitir que o Minikube crie e gerencie a máquina virtual.
- **Iniciação do Cluster:** com um simples comando `minikube start`, o Minikube inicializa uma máquina virtual, configura o nó único e instala os componentes necessários do Kubernetes.
- **Implantação de aplicações:** após a configuração do cluster, os desenvolvedores podem usar o comando `kubectl` para implantar aplicações, criar serviços e gerenciar os recursos do Kubernetes diretamente no cluster Minikube.
- **Monitoramento e debugging:** Minikube oferece ferramentas de monitoramento, como o dashboard do Kubernetes, que permite visualizar o estado dos recursos do cluster e diagnosticar problemas.

A implementação seguiu o roteiro disponibilizado pelo Kubernetes.io [Kubernetes 2024]. Além disso, o passo a passo dos comandos para a implementação bem sucedida pode ser visto no README do repositório no github criado pelos autores, além dos arquivos YAML utilizados para deployment [Jean Mello 2024]. Nas imagens a seguir, será visto á sequência dos comandos utilizandos com as suas respectivas saídas.

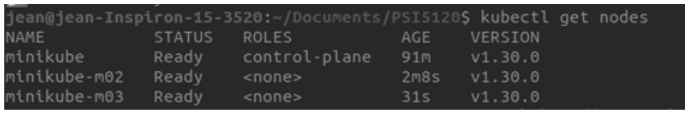

Figure 1. Enable metrics-server

Na Figura 1 é possível observar os nodes do minikube rodando na máquina local, executando a versão 1.30 do Kubernetes.

O servidor web executado foi o php-apache. Na Figura 2 é realizado o deploy dele no cluster.

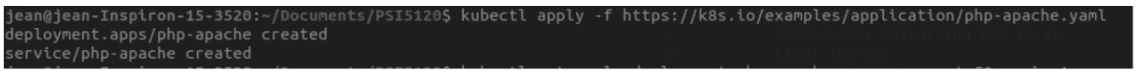

Figure 2. Deploy php-apache server

Após a execução do servidor web, é necessário criar o Horizontal Pod Autoscaler através do comando

```
$ kubectl autoscale deployment php-apache --cpu-percent=50
   --min=1 --max=10
```

Na Figura 3 é visto qual é a porcentagem de processamento do servidor web sem carga.

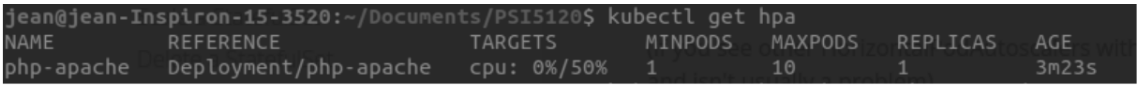

Figure 3. Status without load

Simulando uma carga no servidor Figura 4 ocorre um grande aumento do processamento do servidor web. Inicialmente, esse aumento ocorre na porcentagem de utilização da CPU do servidor e para conseguir reduzir esse processamento, ele cria réplicas, conseguindo reduzir essa porcentagem. Após finalizar o processo de carga, é possível observar uma redução no processamento, chegando até 0% e o número de réplicas chegando a 1, igual inicialmente. Esse processo pode ser visto na Figura 5

Na Figura 6 é possível observar a descrição do servidor web php-apache criado e os eventos ocorridos nele.

```
jean@jean-Inspiron-15-3520:~$ kubectl run -i     --tty load-generator     --rm --image=busybox     --rest
art=Never     -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
If you don't see a command prompt, try pressing enter.
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
```

Figure 4. Sending load to php-apache server

```
ricardo@LAPTOP-I2UUK1OM:~$ kubectl get hpa php-apache --watch
NAME         REFERENCE               TARGETS        MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   cpu: 0%/50%    1         10        1          18m
php-apache   Deployment/php-apache   cpu: 16%/50%   1         10        1          18m
php-apache   Deployment/php-apache   cpu: 250%/50%  1         10        1          19m
php-apache   Deployment/php-apache   cpu: 250%/50%  1         10        4          19m
php-apache   Deployment/php-apache   cpu: 250%/50%  1         10        5          19m
php-apache   Deployment/php-apache   cpu: 116%/50%  1         10        5          20m
php-apache   Deployment/php-apache   cpu: 70%/50%   1         10        5          21m
php-apache   Deployment/php-apache   cpu: 70%/50%   1         10        7          21m      Interrupção da carga
php-apache   Deployment/php-apache   cpu: 54%/50%   1         10        7          22m
php-apache   Deployment/php-apache   cpu: 3%/50%    1         10        7          23m
php-apache   Deployment/php-apache   cpu: 0%/50%    1         10        7          24m
php-apache   Deployment/php-apache   cpu: 0%/50%    1         10        7          28m
php-apache   Deployment/php-apache   cpu: 0%/50%    1         10        1          28m
php-apache   Deployment/php-apache   cpu: 0%/50%    1         10        1          29m
```

Figure 5. Interrupção de carga

```
ricardo@LAPTOP-I2UUK1OM:~$ kubectl describe hpa php-apache
Name:                                                  php-apache
Namespace:                                             default
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Thu, 01 Aug 2024 11:04:24 -0300
Reference:                                             Deployment/php-apache
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  0% (1m) / 50%
Min replicas:                                          1
Max replicas:                                          10
Deployment pods:                                       1 current / 1 desired
Conditions:
  Type            Status  Reason            Message
  ----            ------  ------            -------
  AbleToScale     True    ReadyForNewScale  recommended size matches current size
  ScalingActive   True    ValidMetricFound  the HPA was able to successfully calculate a replica count from cpu resource utilization (percentage of request)
  ScalingLimited  True    TooFewReplicas    the desired replica count is less than the minimum replica count
Events:
  Type    Reason             Age    From                       Message
  ----    ------             ----   ----                       -------
  Normal  SuccessfulRescale  18m    horizontal-pod-autoscaler  New size: 4; reason: cpu resource utilization (percentage of request) above target
  Normal  SuccessfulRescale  17m    horizontal-pod-autoscaler  New size: 5; reason: cpu resource utilization (percentage of request) above target
  Normal  SuccessfulRescale  16m    horizontal-pod-autoscaler  New size: 7; reason: cpu resource utilization (percentage of request) above target
  Normal  SuccessfulRescale  9m19s  horizontal-pod-autoscaler  New size: 1; reason: All metrics below target
```

Figure 6. Descrição do HPA e dos eventos

2.4. Vantagens e Limitações do Minikube

Vantagens:

- **Facilidade de uso:** Minikube é fácil de instalar e configurar, permitindo que desenvolvedores iniciem rapidamente um ambiente Kubernetes local.
- **Custo-benefício:** como o Minikube roda localmente, ele elimina custos de infraestrutura, sendo ideal para desenvolvimento e testes.
- **Simulação realista:** embora seja uma solução de nó único, Minikube oferece uma simulação realista do ambiente Kubernetes, permitindo que desenvolvedores testem configurações antes de moverem para produção.

Limitações:

- **Escalabilidade limitada:** Minikube é limitado a um único nó, o que não permite testes de alta disponibilidade ou escalabilidade horizontal.
- **Desempenho dependente do hardware:** o desempenho do cluster Minikube está diretamente ligado aos recursos da máquina host, o que pode limitar a sua eficácia para aplicações mais pesadas.
- **Ausência de algumas funcionalidades de produção:** alguns recursos avançados de Kubernetes, como volumes persistentes baseados em rede ou balanceadores de carga avançados, podem não estar totalmente disponíveis ou funcionar de maneira limitada no Minikube.

3. Cluster Kubernetes usando AWS EKS

3.1. O que é AWS EKS?

O AWS Elastic Kubernetes Service (EKS) é um serviço gerenciado que facilita a execução de Kubernetes na infraestrutura da Amazon Web Services (AWS). O EKS elimina grande parte da complexidade associada à configuração e manutenção de clusters Kubernetes, permitindo que as equipes de DevOps se concentrem em construir e operar aplicações, em vez de gerenciar a infraestrutura subjacente [Chhabra 2023].

3.2. Características e Arquitetura do AWS EKS

A arquitetura de um cluster EKS é composta por dois componentes principais: o plano de controle (control plane) gerenciado pela AWS e os nós de trabalho (worker nodes) que são executados em instâncias EC2. O plano de controle gerenciado inclui o servidor API do Kubernetes, etcd, e outros componentes críticos, garantindo alta disponibilidade e segurança através de zonas de disponibilidade múltiplas (Multi-AZ).

Uma das vantagens do EKS é sua integração nativa com outros serviços da AWS, como IAM (Identity and Access Management) para gerenciamento de permissões, VPC (Virtual Private Cloud) para rede, e CloudWatch para monitoramento. Isso permite que as empresas construam soluções altamente integradas e seguras, com suporte para práticas de DevOps como Continuous Integration/Continuous Deployment (CI/CD) e infraestrutura como código (IaC).

3.3. Implementação de um Cluster Kubernetes com AWS EKS

Para a configuração do EKS Cluster, foi utilizado como referência o material da matéria PSI5120, ministrada pelo professor Sérgio Kofuji [Kofuji 2024].

A implementação do HorizontalPod Autoscaler seguiu o procedimento fornecido pelo guia do usuário, fornecedio pela Amazon [AWS 2024]. Além disso, no README do repositório [Jean Mello 2024], também é possível encontrar o passo a passo para conseguir realizar a implementação dessa parte.

A implementação de um cluster Kubernetes com AWS EKS envolve, de forma geral, as etapas:

- **Criação do plano de controle:** O primeiro passo na criação de um cluster EKS é configurar o plano de controle, o que pode ser feito através do console AWS, AWS CLI ou ferramentas como Terraform. A AWS cuida de toda a gestão do plano de controle, incluindo atualizações automáticas e patches de segurança. Na Figura 7 contém a implementação na plataforma AWS.
- **Configuração da rede:** antes de adicionar os nós de trabalho, é necessário configurar a rede, incluindo a criação de uma VPC com sub-redes públicas e privadas, tabelas de roteamento, e grupos de segurança. Esta configuração é necessária para garantir que o cluster tenha conectividade adequada e esteja protegido contra acessos não autorizados.
- **Provisionamento dos nós de trabalho:** os nós de trabalho podem ser configurados utilizando o AWS EC2, com suporte para escalonamento automático. O EKS permite a adição de nós com diferentes tamanhos e tipos de instâncias, dependendo da carga de trabalho. As instâncias podem ser gerenciadas manualmente ou através de Grupos de Auto Scaling. Na Figura 8 contém dois nós criados dentro do EKS Cluster.
- **Integração com ferramentas AWS:** após a configuração básica, o EKS pode ser integrado com serviços como Elastic Load Balancing (ELB) para balanceamento de carga, Amazon RDS para bancos de dados gerenciados, e Amazon S3 para armazenamento de objetos.
- **Gerenciamento e operação:** o AWS EKS fornece suporte nativo para ferramentas de monitoramento, como o CloudWatch, bem como para práticas de CI/CD com o uso de serviços como o CodePipeline e o CodeBuild. Além disso, o EKS suporta atualizações automáticas do Kubernetes, garantindo que o cluster esteja sempre atualizado com as últimas melhorias e patches de segurança.

Tendo configurado o AWS CLI no computador local é possível configurar o Cluster EKS criado na plataforma da Amazon. Contudo, para ser possível a comunicação entre o computador local e o Cluster é necessário fazer o deploy do nginx.

Após essa primeira configuração, para testar a comunicação pode se tentar observar os nodes que estão sendo executados, através do comando `kubectl get nodes -o wide`. Assim, será observado os dois nós criados dentro do cluster EKS. Com essas configurações iniciais, pode se executar o mesmo procedimento realizado para configurar o servidor web php-apache do capítulo 2.3 deste trabalho. Na Figura 9 é possível ver os comandos executados com as suas respectivas saídas e a carga no servidor web durante o procedimento de carga, o mesmo processo executado no minikube.

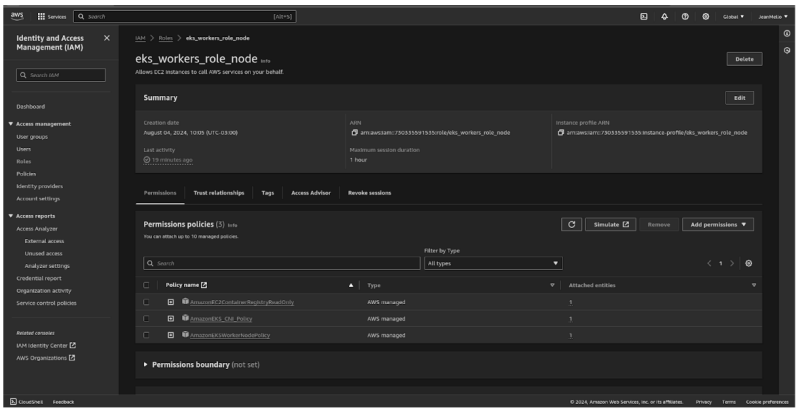

Figure 7. Configurações do Role na plataforma do AWS

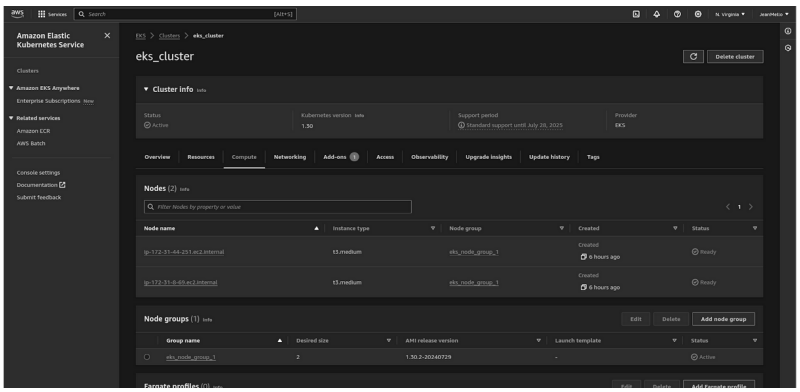

Figure 8. Nós criados no cluster EKS na plataforma do AWS

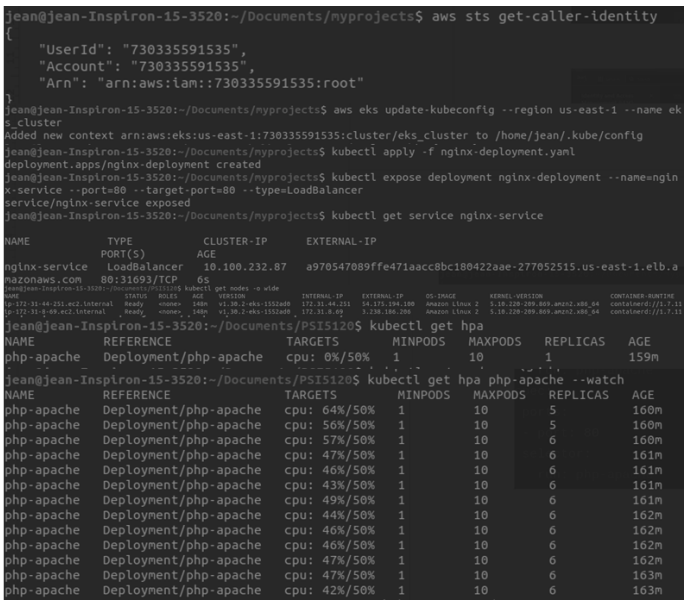

Figure 9. Sequência de comandos até a carga

3.4. Vantagens e limitações do AWS EKS

Vantagens:

- **Alta disponibilidade:** o EKS é executado em várias zonas de disponibilidade, garantindo resiliência contra falhas de infraestrutura.
- **Escalabilidade automática:** com o suporte para Grupos de Auto Scaling, o EKS pode ajustar automaticamente o número de nós de trabalho com base na demanda da aplicação, garantindo desempenho consistente sem desperdício de recursos.
- **Integração nativa com serviços AWS:** a integração com a vasta gama de serviços AWS permite que as empresas criem soluções altamente eficientes e seguras, aproveitando o melhor da infraestrutura AWS.
- **Suporte para Clusters Multi-AZ:** o EKS permite a criação de clusters que se estendem por várias zonas de disponibilidade, garantindo alta disponibilidade e distribuição geográfica das cargas de trabalho.

Limitações:

- **Complexidade de configuração:** embora a AWS forneça diversas ferramentas para simplificar a criação de clusters EKS, a configuração inicial pode ser complexa e requer conhecimento aprofundado da infraestrutura AWS.
- **Custo elevado:** o uso do EKS envolve custos não apenas com as instâncias EC2, mas também com o plano de controle gerenciado e outros serviços AWS associados. Isso pode tornar o EKS uma opção cara para pequenas empresas ou para cenários de desenvolvimento e teste.
- **Dependência da infraestrutura AWS:** embora a integração com os serviços da AWS seja uma vantagem, também cria uma dependência da infraestrutura da AWS, o que pode limitar a flexibilidade para migração ou uso de outras plataformas.

4. Comparação entre Minikube e AWS EKS

4.1. Características principais

Minikube e AWS EKS representam dois extremos do espectro de soluções Kubernetes. Enquanto Minikube é uma solução leve e local, projetada para desenvolvimento e testes, o EKS é uma plataforma robusta e escalável, ideal para ambientes de produção em larga escala. Minikube permite a criação rápida de um cluster Kubernetes local em uma máquina virtual, enquanto o EKS oferece uma infraestrutura gerenciada que pode ser distribuída por várias zonas de disponibilidade.

4.2. Casos de uso

Minikube é adequado para desenvolvedores individuais ou pequenas equipes que precisam de um ambiente Kubernetes para desenvolvimento, testes unitários ou validação de conceitos. É útil em cenários onde a simplicidade e o baixo custo são fatores decisivos.

AWS EKS, por outro lado, é ideal para grandes empresas e organizações que operam aplicações críticas em produção. A escalabilidade, resiliência e integração com outros serviços AWS fazem do EKS uma escolha poderosa para ambientes onde a alta disponibilidade e o desempenho são cruciais. Além disso, o EKS é adequado para equipes de DevOps que desejam aproveitar práticas de CI/CD, monitoramento avançado e gestão centralizada.

4.3. Desempenho e escalabilidade

Minikube oferece desempenho suficiente para testes e desenvolvimento local, mas é limitado pela capacidade da máquina host. Ele não é projetado para cargas de trabalho de produção ou escalabilidade horizontal.

AWS EKS, em contraste, oferece suporte para escalabilidade horizontal automática, permitindo que o número de nós do cluster se ajuste dinamicamente com base na carga de trabalho. A infraestrutura da AWS permite que o EKS suporte grandes volumes de tráfego e aplicações distribuídas, com balanceamento de carga e failover automático

4.4. Custos Envolvidos

Minikube é uma solução econômica, já que sua execução ocorre localmente e não envolve custos com infraestrutura de nuvem. No entanto, os recursos da máquina local limitam sua capacidade e escalabilidade.

AWS EKS envolve custos associados ao uso da infraestrutura AWS, incluindo instâncias EC2, balanceadores de carga, armazenamento, e o plano de controle gerenciado. Embora os custos sejam mais elevados, eles são justificados pelo nível de serviço, alta disponibilidade, e escalabilidade oferecida pela plataforma.

5. Conclusão

Minikube e AWS EKS são ferramentas poderosas para a implementação de clusters Kubernetes, cada uma atendendo a diferentes necessidades e cenários. Minikube é uma escolha excelente para desenvolvimento local, oferecendo uma maneira rápida e eficiente de testar aplicações em Kubernetes sem custos adicionais. Por outro lado, o AWS EKS se destaca em ambientes de produção que exigem alta disponibilidade, escalabilidade e integração com serviços de nuvem.

A escolha entre essas duas soluções deve ser baseada nos requisitos específicos do projeto. Para desenvolvimento e testes, Minikube oferece simplicidade e economia. Para aplicações em produção que exigem desempenho e resiliência, o AWS EKS é a escolha mais adequada, oferecendo uma solução gerenciada que facilita a operação em larga escala.

References

AWS (2024). Scale pod deployments with horizontal pod autoscaler.

Chhabra, N. K. (2023). *Mastering Elastic Kubernetes Service on AWS: Deploy and manage EKS clusters to support cloud-native applications in AWS*. Packt Publishing, english edition edition. Formato: eBook Kindle.

Jean Mello, Jones Egydio, R. G. (2024). Repositório para o trabalho da matéria psi5120. https://github.com/JeanMX/PSI5120.

Kofuji, S. (2024). Notas de aula da disciplina psi5120 - tópicos em computação em nuvem. Escola Politécnica da Universidade de São Paulo. Autor: Prof. Dr. Sérgio Kofuji.

Kubernetes (2024). Horizontalpodautoscaler walkthrough.

Poulton, N. (2021). *Guia Rápido Kubernetes*. Amazon Kindle. Formato: eBook Kindle.