CURSO DE CIÊNCIA DA COMPUTAÇÃO – TCC	
(X) PRÉ-PROJETO ( ) PROJETO	ANO/SEMESTRE: 2022/2

ORQUESTRAÇÃO DE CONTÊINERES PARA ESCALA AUTOMÁTICA DE SERVIÇOS EM UM AMBIENTE DISTRIBUÍDO

Éliton Lunardi

Prof. Aurélio Faustino Hoppe - Orientador

1 INTRODUÇÃO

Atualmente, as organizações estão migrando as aplicações monolíticas que necessitam de escala para ambientes de computação em nuvem (FRITZSCH *et al*., 2018). Neste sentido, segundo os autores, elas normalmente optam por uma abordagem baseada em microsserviços, ao qual possibilita a criação de aplicações que sejam desacopladas e que possuam benefícios como alta disponibilidade, escala automática, fácil gerenciamento de infraestrutura, combinando fluxo de desenvolvimento e operação.

Newman (2021) define microsserviços como sendo serviços modelados a partir de um domínio de negócio, que são capazes de serem implantados nos ambientes de forma independente. Ainda segundo o autor, conceitos de modelagem a partir de um domínio, estado único não compartilhado, tamanho de abrangência e flexibilidade são as chaves para compreender como os microsserviços funcionam. Newman (2021) também destaca que a implantação independente é essencial, pois garante o baixo acoplamento, ao qual permite alterar um serviço sem interferir em nenhum outro. Para o autor, o grande benefício da arquitetura de microsserviços é a escala pois, em um serviço monolito, torna-se necessário escalar toda a estrutura, sendo que em alguns momentos, apenas uma parte específica esteja recebendo uma carga excessiva.

Ainda de acordo com Newman (2021), o provisionamento de recursos de infraestrutura (servidores, redes, usuários, serviços) de forma automática (escala automática) é essencial para aproveitar ao máximo a estrutura baseada em microsserviços, justamente pela possibilidade de provisionar mais máquinas em horários de picos, conforme carga recebida nos serviços. Contudo, ao reduzir a carga, as instâncias provisionadas podem ser removidas, poupando dinheiro.

Vayghan *et al.* (2019) ressaltam que, com o aumento do uso de arquiteturas de microsserviços, torna-se necessário a utilização de uma ferramenta de orquestração de contêineres para gerenciamento dos *deploys.* Ainda segundo os autores, a conteinerização deixa o processo de execução leve e isolado, sendo assim adequado para a abordagem de microsserviços. Lehtinen (2022) destaca que para gerenciar a carga de trabalho ou a quantidade de processamento que o computador recebe (*workloads)* em um determinado momento sem a utilização de uma ferramenta de orquestração de contêineres é possível, entretanto, quanto maior for a escala ou a quantidade de servidores necessários para suportar a carga recebida, o gerenciamento em si se torna mais complexo, fazendo com que inevitavelmente tem-se a necessidade de utilização de uma ferramenta de orquestração de contêineres para minimizar o problema. Neste sentido, segundo Vayghan *et al*. (2019), o Kubernetes apresenta-se como uma ferramenta de orquestração de contêineres disponível e de código *open-source*, que permite os benefícios do *deploy* automatizado, escala automática de recursos e o gerenciamento do ciclo de vida das aplicações microsserviços que estão executando através de conteinerizações. Além disso, também permite verificar a saúde dos mesmos, reiniciando contêineres que estão em estado de falha ou executar novos contêineres caso algum dos *hosts* do *cluster* falhe e assim por diante.

Em relação a escala automática, segundo Lehtinen (2022), o Kubernetes possui o *Horizontal Pod Autoscaling* (HPA), que é responsável por efetuar a escala horizontal, ou seja, incrementar ou diminuir o número de contêineres (*pods*). Neste sentido, a partir de métricas coletadas pelo Kubernetes como CPU e RAM de forma nativa, ele pode efetuar a decisão de escala nas configurações efetuadas, ao qual denomina-se estado desejado. Com isso, ainda de acordo com Lehtinen (2022), torna-se possível efetuar o *deploy* de uma aplicação com baixa configuração e, à medida que mais recursos tornam-se necessários, o HPA incrementa ou decrementa a quantidade de *pods* até que a demanda da aplicação seja suprida.

Nguyen *et al*. (2020), salientam que apesar do dimensionamento automático ser uma necessidade inerente para realizar o provisionamento adequado em uma arquitetura de microsserviços, as ferramentas e métricas existentes para análise de desempenho de serviços que utilizam contêineres não são suficientes para determinados casos de uso. Por esse motivo, segundo os autores, novas formas de monitoramento precisam ser estudadas e apresentadas no intuito de aumentar a efetividade da escala automática do Kubernetes.

Diante deste contexto, este trabalho tem como objetivo o desenvolvimento de uma ferramenta que realize a configuração de um *cluster* de Kubernetes em um provedor de computação em nuvem, coletando resultados através de testes de carga em uma aplicação de microsserviços, aprofundando-se na questão de escala automática

que o Kubernetes disponibiliza. Além disso, também serão implementadas e testadas métricas externas ao *cluster* (além de CPU e RAM), buscando atender os cenários de uso ao qual tais métricas podem ser mais efetivas.

1.1 OBJETIVOS

O objetivo deste trabalho é disponibilizar uma ferramenta que seja capaz de escalar automaticamente serviços de uma aplicação de computação em nuvem.

Os objetivos específicos são:

a) criação de *scripts* e arquivos de configuração para provisionar os recursos computacionais da arquitetura distribuída baseada em Kubernetes;
b) estudar e avaliar técnicas que possam auxiliar no processo de obtenção de métricas internas e externas quanto ao uso do *cluster*;
c) avaliar a capacidade de aumentar e reduzir o tamanho da infraestrutura de forma a responder a variações das cargas efetuadas em uma aplicação de microsserviços.

2 TRABALHOS CORRELATOS

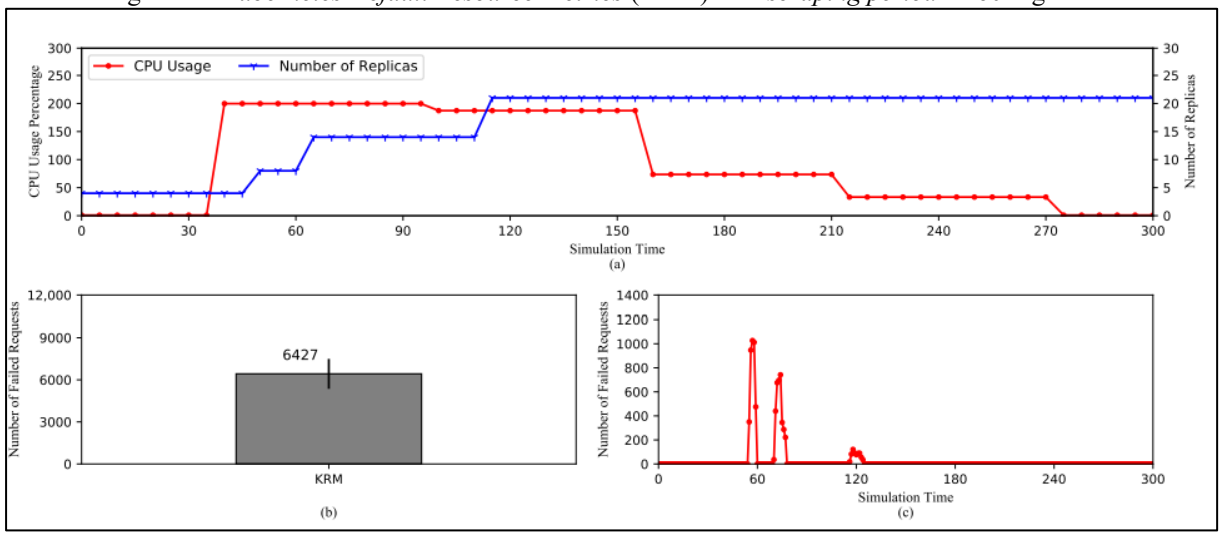
Nesta seção serão apresentados os trabalhos que se correlacionam com os objetivos deste projeto. A seção 2.1 discorre sobre o funcionamento do *Horizontal Pod Autoscaling* (HPA) do Kubernetes (NGUYEN *et al*., 2020). A seção 2.2 relata o processo de utilização do HPA em conjunto com as métricas obtidas do *Thread Pool* para ajuste de escala (ZHU; HAN; ZHAO, 2021). Por fim, a seção 2.3 aborda o processo de configuração do Kubernetes, na Amazon Web Services (AWS), em um ambiente de produção (PONISZEWSKA-MARANDA; CZECHOWSKA, 2021).

2.1 HORIZONTAL POD AUTOSCALING IN KUBERNETES FOR ELASTIC CONTAINER ORCHESTRATION

Nguyen *et al*. (2020) focaram em experimentos de exploração das tendências de escala e otimizações do *Horizontal Pod Autoscaling* (HPA). Para isso, utilizaram métricas padrões do Kubernetes, processamento de requisições e latência obtidas da *Default Kubernetes Resource Metrics* (KRM) assim como, de métricas externas obtidas da *Prometheus Custom Metrics* (PCM).

Nguyen *et al*. (2020) efetuaram os experimentos a partir de um *cluster* com 5 nós, sendo 1 nó *master* e outros 4 *workers*, em uma máquina física com um processador i7-8700 3.20GHz. O nó *master* foi alocado com 4 núcleos do processador e 8GB RAM enquanto os nós *workers* com 2 núcleos de processador com 2 GB RAM para cada. Através do Gatling, um software externo *open-source* tornou-se possível por simular a carga via requisição *Hypertext Transfer Protocol* (HTTP). Segundo os autores, a aplicação construída para atender a carga gerada no *cluster,* utiliza intensivamente o processador dos nós. Cada réplica dessa aplicação, em cada nó, utiliza 100 *milicore* (m) durante a sua inicialização e 200 m de limite de utilização. Durante os experimentos, o mínimo e máximo de réplicas foram de 4 e 24 respectivamente. Todos os experimentos foram executados durante 300 segundos, sendo que nos primeiros 100 segundos, a carga gerada foi de aproximadamente 1.800 requisições por segundo, reduzindo posteriormente para 600 requisições por segundo, totalizando 240.000 requisições. A Figura 1 apresenta o comportamento do primeiro experimento realizado.

Figura 1 – *Kubernetes Default Resource Metrics* (KRM) com *scraping period* de 60 segundos

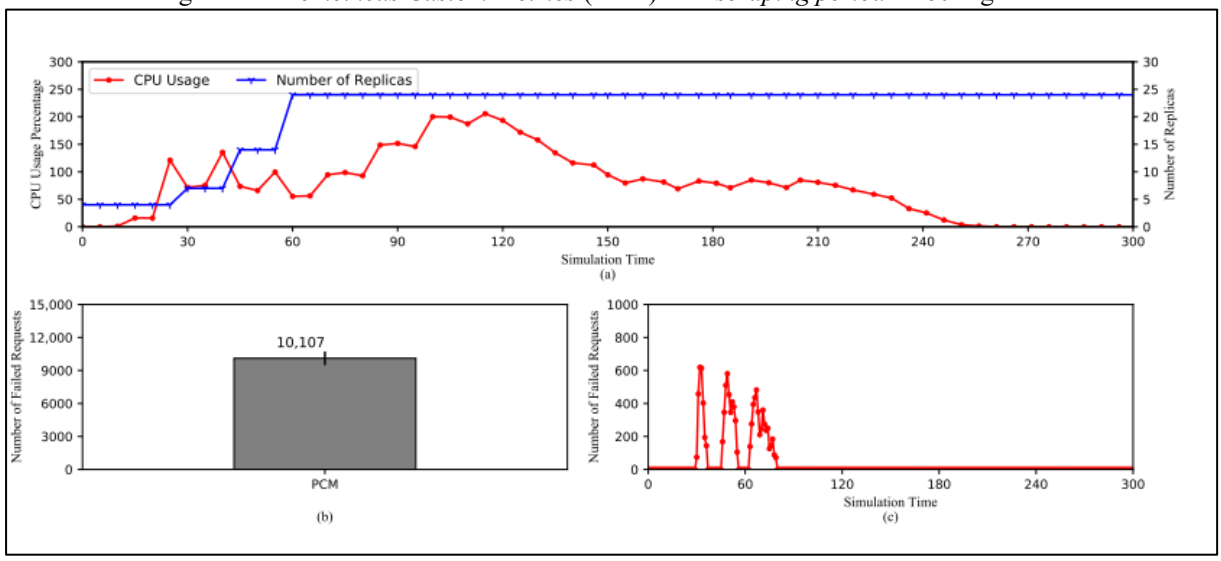


Fonte: Nguyen *et al*. (2020).

De acordo com Nguyen *et al*. (2020), dos segundos 40 até os 100 segundos, um *scraping period* de 60 segundos é completado, não apresentando alterações no uso de CPU. Dessa forma, pode-se perceber uma importante característica do HPA, que a cada 15 segundos, verifica se as métricas sofreram alterações, e caso não tenham sofrido, o número de réplicas não se altera. Após 100 segundos, as métricas de uso de CPU sofrem alteração, fazendo com que essas métricas sejam entregues para o HPA, que estabiliza a escala em 21 réplicas.

Já na Figura 2 é possível notar que existe uma alteração muito frequente nas métricas coletadas. De acordo com Nguyen *et al*. (2020), isso acontece em decorrência da forma que as métricas são obtidas pelo Prometheus. Da mesma forma que o KRM possui o *Scraping Period*, o PCM também possui, porém, as métricas do Prometheus passam por uma função chamada de rate() que estima as novas métricas com base nas antigas. Ainda segundo os autores, é dessa forma que as métricas coletadas pelo Prometheus alteram, em um período, muito mais rápido que no KRM. Além das alterações das métricas, a partir da Figura 2(a), é possível perceber que o número de réplicas atinge o máximo (24 réplicas) muito mais rápido que as 21 que a forma do KRM chegou. Segundo Nguyen *et al*. (2020), esse aumento rápido no número de réplicas permite que o HPA esteja preparado para cargas expressivas que possam chegar aos *pods* escalados.

Figura 2 – *Prometheus Custom Metrics* (PCM) com *scraping period* de 60 segundos



Fonte: Nguyen *et al*. (2020).

O tempo de coleta das métricas, chamado de *scraping period* pode ser ajustado tanto no KRM quanto no PCM, porém, de acordo com Nguyen *et al*. (2020), apenas a do PCM não afeta a performance do HPA, diferentemente do KRM. Os autores recomendam um período mais longo que o padrão para reduzir a quantidade de recursos utilizados para obter tais métricas. Deve-se observar, porém que um período muito longo pode causar imprecisão nas métricas obtidas. Segundo Nguyen *et al*. (2020), um período de coleta maior pode reduzir a quantidade de recursos alocados para novos *pods*, assim como causar a degradação da qualidade dos serviços. Dessa maneira, a recomendação dos autores é avaliar os tipos de serviços no *cluster* e, cuidadosamente definir um período de coleta adequado a situação.

Após a realização dos experimentos, Nguyen *et al*. (2020) recomendam que as KRM sejam utilizadas em aplicações com cargas mais estáveis como serviços de processamento de vídeo onde as requisições dos usuários geralmente são menores, e levam alguns minutos para processar algumas horas de vídeo. Enquanto isso, as métricas PCM do Prometheus, segundo os autores, atendem aplicações com alterações frequentes nas métricas, ou seja, que utilizam mais recursos dos *pods*.

Por fim, após a discussão dos resultados Nguyen *et al*. (2020) concluem que o Kubernetes é uma plataforma de orquestração de container poderosa para aplicações e serviços executados dessa maneira. Segundo os autores, com base nos experimentos efetuados, o HPA é uma ferramenta que possibilita a escala dos serviços sem a intervenção humana através da coleta de métricas, sejam elas por meio da KRM ou via PCM. Além disso, os autores sugerem como extensão do trabalho a realização de experimentos com o HPA a fim de desenvolver algoritmos de escalas mais eficientes para diferentes cenários.

2.2 A BI-METRIC AUTOSCALING APPROACH FOR N-TIER WEB APPLICATIONS ON KUBERNETES

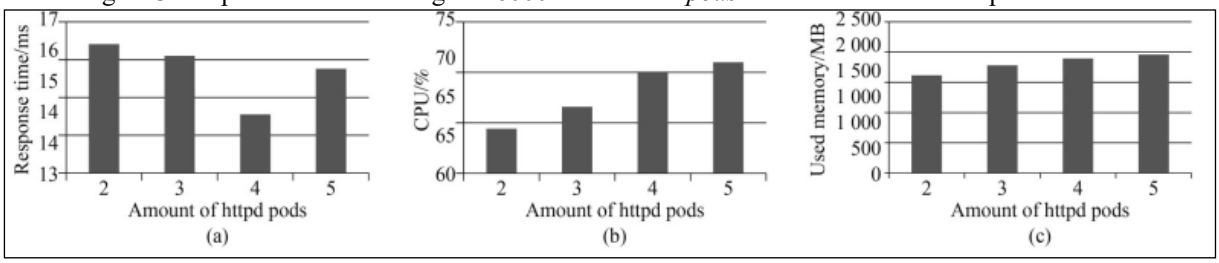
Segundo Zhu, Han e Zhao (2021), a escala do *Horizontal Pod Autoscaling* (HPA), baseado apenas na utilização da *Central Unit Processing* (CPU), cria mais *pods* do que é realmente necessário para o *cluster*. Os autores utilizaram o RUBBoS, uma ferramenta de *benchmark* utilizada por aplicações que executam *containers*, para efetuar um estudo e, ao final, concluíram que muitos *pods* consumem mais CPU, memória e pioram o tempo

de resposta das aplicações. A partir disso, Zhu, Han e Zhao (2021) propuseram uma abordagem para escala dos *pods* que leva em conta além do uso da CPU, a *thread pool* do *Hypertext Transfer Protocol daemon* (HTTPd) e Tomcat. Através dela, os autores conseguiram demonstrar que menos *pods* são criados, resultando em reduções de custos. Além disso, eles ressaltam que a melhor forma de escalar *pods* no Kubernetes é através do uso de mais que um tipo de métrica coletora.

Zhu, Han e Zhao (2021) utilizaram um *cluster* Kubernetes que foi configurado com 4 *Virtual Machines* (VM), cada uma com 6 núcleos de CPU e 8GB de memória, visando entender o comportamento do HPA assim como, mostrar existência da criação de *pods* desnecessários. Além disso, utilizou-se um host físico disponibilizado pelo emulab, com 2 CPU de 8 núcleos cada e 64GB de memória. Para o experimento em questão, os autores utilizaram 24 interações da ferramenta de *benchmark* RUBBoS, observando apenas as cargas geradas através do navegador. Ressalta-se que das configurações apresentadas, o RUBBoS foi configurado como servidor web, no qual é executado o HTTPd, o Tomcat, e o MySQL. Além disso, cada um deles é executado em um *pod* nas VMs, e o MySQL no host físico.

A partir da configuração descrita, Zhu, Han e Zhao (2021) conduziram uma série de experimentos com um nível de carga de 6000 clientes para demonstrar que os *pods* criados em excesso não resultam em mais performance. Nesse experimento, os autores deixaram os *pods* do HTTPd sem limite de CPU, estabelecendo apenas 1 CPU para cada *pod* executar, fazendo com que os recursos necessários sejam utilizados sem limites. Na Figura 3 é apresentado o resultado do experimento. Destaca-se que a quantidade de *pods* do HTTPd foi alterada de 2 até 5. Dessa maneira, de acordo com Zhu, Han e Zhao (2021), é possível perceber na Figura 3 (a) que o tempo de resposta variou entre 14 ms e 16 ms. Na Figura 3(b) Figura 3(c), os autores destacam que o uso de CPU subiu de 64% para 71% e de memória de 1615MB para 1959MB. Para Zhu, Han e Zhao (2021) isso indica que a quantidade de *pods* a mais apresentam mais uso de CPU e memória e não representa um ganho de performance expressivo.

Figura 3 - Experimentos com carga de 6000 clientes nos *pods* do HTTPd utilizando apenas o HPA



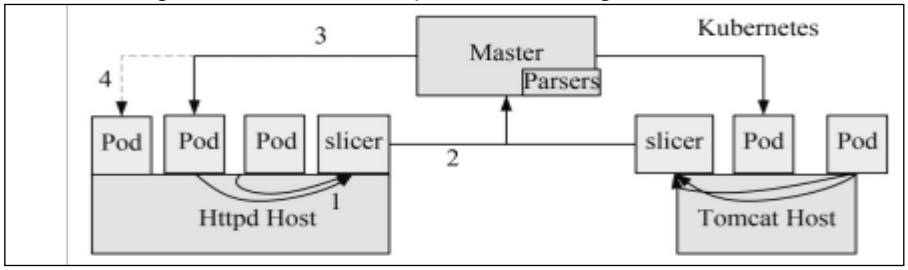
Fonte: Zhu, Han e Zhao (2021).

A partir dos resultados obtidos, Zhu, Han e Zhao (2021) desenvolveram uma abordagem de escala que utiliza em conjunto com as métricas de CPU e memória a ferramenta ELBA para calcular o tamanho das filas do *thread pool* do HTTPd e do Tomcat em seus respectivos *pods* e assim tomando as decisões de escala. De acordo com os autores, a ELBA é uma ferramenta que detecta gargalos em aplicações web e que calcula e monitora o tamanho de filas de *pods* do HTTPd e do Tomcat.

Para desenvolver a abordagem de escala, Zhu, Han e Zhao (2021) efetuaram a conteinerização da ELBA dentro da RUBBoS. Com a ELBA em container, o *deploy* dela foi em forma de agente, em cada host que os *pods* executam. Tais agentes enviam os *logs* gerados pelo ELBA para um host específico que possui a inteligência de calcular o tamanho da fila do *thread pool*. A partir das entradas dos *logs*, o algoritmo dos autores determina se é necessário criar mais *pods* ou quais parar que estão em execução.

Na Figura 4 é possível visualizar como a solução de Zhu, Han e Zhao (2021) funciona. Um *pod* denominado *slicer* é responsável por extrair os *logs* após receber um comando do host *master*, enviando para si memo. No host *master*, um componente denominado *parser*, executa uma funcionalidade do ELBA que recebe os *logs* e que efetua o cálculo das filas de cada *pod*. Ressalta-se que é através desse componente *parser* que as decisões de escala são calculadas e executadas através da linha de comando do Kubernetes.

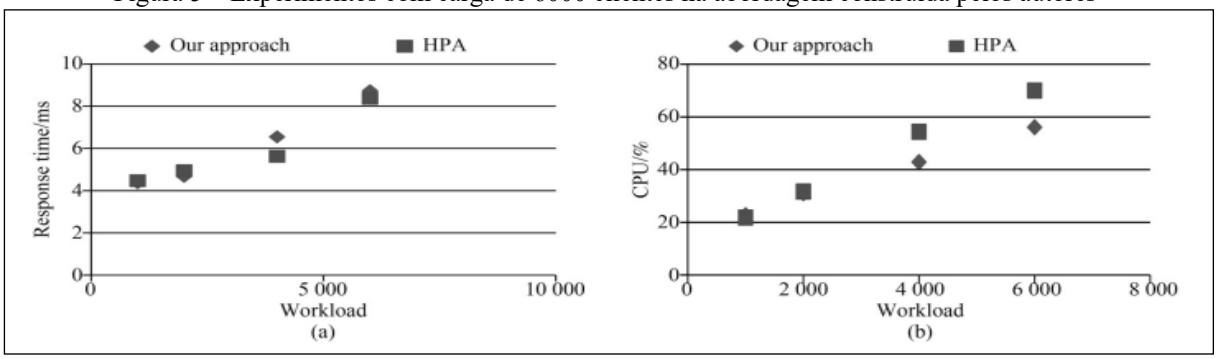
Figura 4 – *Pods* em execução com a abordagem dos autores



Fonte: Zhu, Han e Zhao (2021).

Posteriormente, Zhu, Han e Zhao (2021) efetuaram novos experimentos para verificar a eficácia da abordagem construída. A Figura 5 mostra o resultado de um dos experimentos. Nele, os autores iniciaram o *pod* com 0.5 CPU e limite de 2 CPU. No cenário de carga de 6000 clientes, é possível perceber que o custo de memória do HPA foi de 70% e na solução dos autores 56%. Já para o custo de memória foi de 1761MB do HPA enquanto na solução dos autores foi de 1341MB. Além disso, o tempo de resposta de acordo com Zhu, Han e Zhao (2021) foi quase o mesmo entre ambos os métodos de escala, como pode ser visto na Figura 5(a). Os autores argumentam que o custo elevado de CPU e memória é justamente o motivo pelo qual a solução deles cria menos *pods* que o HPA, mas, mantém o mesmo tempo de resposta.

Figura 5 – Experimentos com carga de 6000 clientes na abordagem construída pelos autores



Fonte: Zhu, Han e Zhao (2021).

Zhu, Han e Zhao (2021) concluem que o Kubernetes deixa a automação do *deploy* e da escala de aplicações, que executam em contêineres, mais fácil e com performance em ambientes de computação em nuvem. Porém, de acordo com os autores, através dos experimentos, constatou-se que a funcionalidade HPA cria mais *pods* que o necessário. Além disso, Zhu, Han e Zhao (2021) apontam que utilizar várias métricas torna o escalonamento dos *pods* mais assertiva.

2.3 KUBERNETES CLUSTER FOR AUTOMATING SOFTWARE

Poniszewska-Maranda e Czechowska (2021) apresentaram como se efetua o processo de *deploy* de um *cluster* de Kubernetes. Além disso, as autoras também compararam dois métodos de *deploys*: um através do Kubernetes *Operations* (KOps) e outro através do *Elastic Kubernetes Service Cluster* (EKSCTL), a *Command Line Interface* (CLI) da *Amazon Web Services* (AWS) para os próprios serviços gerenciados de Kubernetes. Ambos os métodos de *deploy* foram feitos no ambiente de computação em nuvem da AWS. Poniszewska-Maranda e Czechowska (2021) também demonstraram o custo de manter um *cluster* Kubernetes na AWS, sugerindo o uso de serviços gerenciados em provedores de computação em nuvem, assim como elencando atributos específicos a serem atendidos para utilizar o *deploy* de maneira *self-hosted*, sem usar serviços gerenciados.

Para efetuarem o *deploy* de um cluster pronto para um ambiente de produção, Poniszewska-Maranda e Czechowska (2021) definiram os seguintes requisitos: (i) cluster saudável e utilizável; (ii) operações automatizadas; (iii) central de monitoramento; (iv) central de *logs*; (v) auditoria; (vi) alta disponibilidade; (vii) segurança; (viii) escala automática. A partir disso, as autoras também analisaram as formas de *deploy* com KOps e EKSCTL e se elas contemplam os requisitos estabelecidos.

De acordo com Poniszewska-Maranda e Czechowska (2021), o processo de *deploy* não é algo simples de ser feito. Por isso, optou-se por efetuar o *deploy* através de serviços gerenciados. Ainda segundo as autoras, o *deploy* através de serviços gerenciados consegue em poucos minutos entregar um *cluster* funcionando com os requisitos de segurança, alta disponibilidade por um preço baixo. A forma escolhida de *deploy* utilizada pelas autoras foram as seguintes: *deploy* na AWS, usando a AWS *Managed Service* (AWS EKS), através da EKSCTL, ferramenta oficial da AWS; *deploy* na AWS, sem usar nenhum serviço gerenciado, porém usando o KOps, que não é oficialmente de nenhum provedor de computação em nuvem, mas se propõe a efetuar o *deploy* do cluster de Kubernetes. Poniszewska-Maranda e Czechowska (2021) recomendam o uso do Kubernetes através de serviços gerenciados por conta da complexidade de gerenciamento do *cluster* ser menor. Para elas, se apenas o preço, baixa latência de rede, obrigações legais e controle total do hardware forem essenciais, o *cluster* deve ser criado manualmente a partir de *Virtual Machine* (VM) por exemplo.

Através das duas formas de *deploys* escolhidas, Poniszewska-Maranda e Czechowska (2021) atingiram os requisitos listados. Isso significa que ambas as formas podem ser utilizadas em ambientes de produção. Entretanto, segundo as autoras, há uma grande diferença nos tempos de *deploy* do *cluster*, o que pode ser um fator decisivo em relação a opção preterida. A Figura 6 ilustra os tempos de *deploy* do *cluster* de cada método de *deploy* utilizado, com todas as etapas destacadas prontas para o ambiente de produção. É possível notar um tempo alto do EKSCTL comparado ao KOps. Só para o *cluster* ser criado e ter os requisitos de produção listados pelas autoras, são 6 minutos e 35 segundos do KOps, contra 25 minutos e 40 segundos do EKSCTL. Um outro problema

apontado por Poniszewska-Maranda e Czechowska (2021) é que os tempos irão impactar de forma significativa caso alguma configuração do *cluster* necessite de alteração. Além disso, o método de *deploy* EKSCTL exige que o *cluster* seja apagado, e todo os processos destacados na Figura 6 terão que ser repetidos. Já com o KOps, é possível alterar as configurações desejadas, e apenas atualizar no ambiente já criado, o que leva muito menos tempo.

Figura 6 – Tempo de *deploy* do *cluster* nas duas formas exploradas

Operation	Using Kops Method	Using Eksctl Method
Create a minimal cluster	6 min 12 s	19 min 26 s
Create a production-grade cluster	6 min 35 s	25 min 40 s
Test a cluster	6 min 33 s	6 min 40 s
Delete a cluster	2 min 23 s	13 min 25 s

Fonte: Poniszewska-Maranda e Czechowska (2021).

Poniszewska-Maranda e Czechowska (2021) também observaram os valores estimados para um mês dos recursos criados na AWS, supondo 720h no mês, para a forma de *deploy* via EKSCTL e KOps. O valor total das formas de *deploy* foram de $162.82 e de $96.60, respectivamente. As autoras destacam que o KOps permite mais opções de configurações e pode ser executado em diferentes ambientes de computação em nuvem, além de ter apresentado um custo melhor. A partir disso, Poniszewska-Maranda e Czechowska (2021) concluem que o KOps é o melhor método de *deploy*, porém, para elas é necessário avaliar o contexto em que será usado pois, pode haver outras necessidades que façam com que tempo de *deploy* e custo não interfiram na escolha dos métodos de *deploy*.

3 PROPOSTA

Nesta seção será descrita a proposta do trabalho, justificando o experimento, definindo os requisitos funcionais e não funcionais, as metodologias abordadas e por fim o cronograma.

3.1 JUSTIFICATIVA

No Quadro 1 é apresentado um comparativo das características mais notáveis entre os trabalhos correlatos. As linhas representam as características e as colunas os trabalhos.

Quadro 1 – Comparativo dos trabalhos correlatos

Trabalhos Correlatos / Características	Nguyen *et al*. (2020)	Zhu, Han e Zhao (2021)	Poniszewska-Maranda e Czechowska (2021)
ferramenta de orquestração	Kubernetes	Kubernetes	Kubernetes
provedor de computação em nuvem utilizado	máquina local	máquina local e emulab	*Amazon Web Services* (AWS)
método de escala	HPA/Prometheus Custom Metrics	HPA e thread pool	não possui
serviço gerenciado	não possui	não possui	EKS Cluster
custos da infraestrutura	$0	$0	$96.60

Fonte: elaborado pelo autor.

Nguyen *et al*. (2020) exploraram o *Horizontal Pod Autoscaling* (HPA) do Kubernetes, num ambiente local. A partir disso, os autores realizaram experimentos para identificar como as métricas coletadas e entregues para o HPA influenciam na performance no orquestrador. Para isso, Nguyen et al (202) utilizaram as ferramentas de escala Kubernetes *Default Resource Metrics* (KRM) e do *Prometheus Custom Metrics* (PCM). Nguyen *et al*. (2020) relatam que a KRM apenas atualiza as métricas via *scraping period.* Isso faz com que o HPA não tenha atualizações nas métricas em dados períodos. Além disso, os autores apontam que a PCM possui maior efetividade na obtenção de métricas constantes. Já a KRM realiza a alteração de escala mais tardiamente.

Zhu, Han e Zhao (2021) propuseram uma solução baseada no *thread pool* de *pods* do Tomcat e de HTTPd. A partir das métricas coletadas, os autores efetuaram a escala levando em consideração a fila do *thread pool* e o uso de CPU, concluindo que a solução proposta crie menos *pods* que o HPA considerando apenas métricas de CPU. Além disso, Zhu, Han e Zhao (2021) executaram os experimentos num ambiente local com o Kubernetes configurado, e um host específico no emulab. A partir disso, os autores demonstraram que o HPA cria *pods* desnecessário. Contudo, Zhu, Han e Zhao (2021) afirmam que ao utilizar mais de uma métrica ao mesmo tempo tornam as decisões de escala mais assertivas.

Poniszewska-Maranda e Czechowska (2021) apresentaram e discutiram a configuração de um *cluster* de Kubernetes para ser utilizado em produção. Através do serviço de computação em nuvem da *Amazon Web Services* (AWS), elencaram requisitos que possibilitem o *deploy* de um *cluster*. Além disso, Poniszewska-Maranda e

Czechowska (2021) concluem que a criação de um *cluster* de Kubernetes é complexa, sugerindo a utilização de serviços gerenciados de computação em nuvem. Através do método de *deploy* KOps e EKSCTL, mediram o tempo de *deploy* e o custo do *cluster* ao ser executado durante um mês na AWS.

Nguyen *et al*. (2020) abordam as métricas coletadas, e como o HPA as utiliza para escala, especialmente métricas de CPU. Já Zhu, Han e Zhao (2021) também exploram o HPA, porém tentam ajustar a escala a partir do *thread pool*. Destaca-se que nenhum deles mostram as configurações da escala no *cluster*. Poniszewska-Maranda e Czechowska (2021), não exploraram a escala, mas como realizar o *deploy* em um *cluster* no ambiente da AWS.

Diante do exposto acima, é possível notar que há pesquisas sendo efetuadas para entender o funcionamento dos métodos de escala automática fornecida pelo Kubernetes, mas que nenhuma delas exploram de maneira aprofundada o conhecimento fundamental para sua compreensão. Partindo disso, neste trabalho, a configuração e a escala de recursos computacionais serão observadas a partir de métricas externas, isto é, além do consumo de memória e CPU obtidas através do HPA, também serão estabelecidas métricas considerando o tamanho de uma fila de mensageria, ou um horário específico que é conhecido que a aplicação receberá uma carga expressiva, ou uma consulta em um banco de dados. Entende-se que a escala por métricas externas favorece diferentes contextos de uso, onde apenas o uso das métricas de CPU e RAM nativas do Kubernetes podem não ser assertivas ou eficientes. Neste sentido, destaca-se que o trabalho fará uso de abstrações do que se refere a configuração do *cluster*, utilizando um serviço gerenciado de um provedor de computação em nuvem, simplificando e abstraindo, dessa maneira, a criação e alteração das configurações do *cluster*, demonstrando detalhadamente toda a parte de configuração efetuada. Além disso, também será feito o *deploy* de uma aplicação e através de testes de carga, as demandas por recursos computacionais poderão ser redimensionadas de forma automática, ou seja, dependendo do contexto, aumentando ou diminuindo os recursos. Em termos práticos, a experimentação da escala favorecerá que a demanda da aplicação seja atendida sem a intervenção humana, perfazendo que apenas em momentos de pico a aplicação modifique a quantidade de recursos disponíveis. Por fim, acredita-se que a partir dos resultados alcançados e apresentados neste trabalho, pesquisadores, desenvolvedores e administradores de sistemas poderão tomar decisões sobre quais métricas são mais adequadas para monitorar e estabelecer a escala automática de forma mais precisa e, consecutivamente possibilitando a redução dos custos em relação ao provisionamento dos recursos de forma manual.

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A ferramenta de configuração de escala a ser desenvolvida deverá ter os seguintes Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF):

a) permitir que a infraestrutura seja criada e destruída de forma automatizada (RF);
b) utilizar uma arquitetura de microsserviços e um orquestrador de contêineres (RF);
c) permitir a configuração do *cluster* Kubernetes (RF);
a) gerenciar o ciclo de vida e orquestrar os contêineres dos serviços (RF);
b) obter métricas de escala disponibilizadas pelos serviços do *cluster* Kubernetes (CPU, memória e tamanho da fila de mensageria) (RF);
c) mostrar detalhadamente as configurações de escala realizadas na ferramenta de orquestração (RF);
d) mostrar os resultados das métricas após a realização dos testes de carga (RF);
e) utilizar Kubernetes como ferramenta de orquestração de container (RNF);
f) utilizar algum provedor de computação em nuvem (RNF);
g) utilizar alguma ferramenta que permita o gerenciamento e provisionamento da infraestrutura como código (*Infrastructure as Code*) (RNF);
h) utilizar .Net para a construção da aplicação microsserviços que será feito *deploy* para testes (RNF);

3.3 METODOLOGIA

O trabalho será desenvolvido observando as seguintes etapas:

a) levantamento bibliográfico: realizar levantamento bibliográfico sobre Kubernetes executando a partir de um serviço gerenciado, microsserviços, métricas externas ao Kubernetes, provedores de computação em nuvem e trabalhos correlatos;
b) elicitação de requisitos: com base nas informações da etapa anterior, realizar reavaliação dos requisitos, e caso necessário, especificar novos requisitos a partir das necessidades identificadas a partir da revisão bibliográfica;
c) especificação da ferramenta: formalizar a ferramenta e a infraestrutura em nuvem através de ferramentas de diagramação Lucidchart e Cloudcraft para elaborar os diagramas de classes, sequência e atividades de acordo com a Unified Modeling Language (UML);
d) definição do provedor de computação em nuvem: pesquisar e estudar qual provedor de computação em nuvem oferece os recursos adequados em relação ao desenvolvimento da ferramenta;
e) criação do *cluster* de Kubernetes: a partir do item (d), será realizado o desenvolvimento dos *scripts* de criação do *cluster* Kubernetes;

f) criação da infraestrutura *cloud*: a partir do item (d) desenvolver e hospedar a arquitetura de microsserviços utilizando o *cluster* de Kubernetes do item (e);
g) criação da aplicação distribuída: implementar uma aplicação de teste ao qual irá executar no *cluster* de Kubernetes;
h) definição das métricas: pesquisar e definir quais métricas internas e externas serão coletadas a partir dos itens (f) e (g);
i) efetuar as configurações de escala: a partir do *cluster* criado (item f), com a aplicação executando (item g), as métricas definidas (item h), implementar os scripts para realizar as configurações automáticas de escalas, sendo elas externas ao Kubernetes;
j) testes: validar a eficiência da ferramenta em relação ao provisionamento e funcionamento da infraestrutura a partir de uma aplicação com arquitetura de microsserviços. Além disso, também será efetuado testes de carga para averiguar a influência e eficiência das métricas em relação as configurações de escala efetuadas para atender a demanda de processamento computacional dos serviços da aplicação distribuída.

As etapas serão realizadas nos períodos relacionados no Quadro 2.

Quadro 2 – Cronograma de atividades a serem realizadas

	2023									
	fev.		mar.		abr.		maio		jun.	
etapas / quinzenas	1	2	1	2	1	2	1	2	1	2
levantamento bibliográfico										
elicitação de requisitos										
especificação da ferramenta										
definição do provedor de computação em nuvem										
criação do *cluster* de Kubernetes										
criação da infraestrutura *cloud*										
criação da aplicação distribuída										
definição das métricas										
efetuar as configurações de escala										
testes										

Fonte: elaborado pelo autor.

4 REVISÃO BIBLIOGRÁFICA

Esta seção descreve brevemente sobre os assuntos que fundamentarão o estudo a ser realizado: microsserviços, configuração de um *cluster* de Kubernetes, métricas externas para efetuar a escala automática de uma aplicação em arquitetura de microsserviços em um provedor de computação em nuvem.

Para Fowler (2014), o termo microsserviços não tem uma definição, e sim características comuns que definem o termo, sendo elas: *deploy* automático, organizado em torno de áreas específicas do negócio do contexto inserido e controle decentralizado de linguagem e dados. Além disso, os microsserviços não devem ter dependência entre eles no momento do *deploy*, devendo ter a capacidade de terem atualizações em momentos distintos uns dos outros. Neste sentido, torna-se indispensável a utilização de uma ferramenta de orquestração de contêineres, como por exemplo, o Kubernetes (NEWMAN, 2021).

Segundo Jawarneh *et al.* (2019), o Kubernetes foi desenvolvido pela Google em 2014. Ele funciona a partir de uma arquitetura *master/slave*, no qual uma lista de aplicações é enviada ao nó master e estas são entregues aos vários nós *slaves* para execução. O nó *master* representa o painel de controle, e esse pode ser replicado em outras máquinas para garantir alta disponibilidade. Além disso, é através dos nós *slaves* que o Kubernetes executa os *pods*, aos quais podem ser gerenciados de forma automática (JAWARNEH *et al.*, 2019).

De acordo com Lehtinen (2022), para efetuar a escala automática, isto é, aumentar ou diminuir a quantidade de máquinas que estão executando, o Kubernetes possui o *Horizontal Pod Autoscaling* (HPA) que é capaz de efetuar as escalas de maneira automática, analisando valores definidos por configuração, podendo ser a quantidade mínima e máxima de CPU e RAM ao qual o *cluster* não deve ultrapassar ao executar todos os *pods* da aplicação desejada.

REFERÊNCIAS

FOWLER, Martin. **Microservices**. 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 25 set. 2022.

FRITZSCH, Jonas *et al*. From Monolith to Microservices: a classification of refactoring approaches. **Arxiv**, [S.l.], p. 1-13, 2018. ArXiv. http://dx.doi.org/10.48550/ARXIV.1807.10059. Disponível em: https://arxiv.org/abs/1807.10059. Acesso em: 22 set. 2022.

JAWARNEH, Isam Mashhour *et al*. Container Orchestration Engines: a thorough functional and performance comparison. **Icc 2019 - 2019 Ieee International Conference On Communications (Icc)**, [S.l.], p. 1-6, maio 2019. IEEE. http://dx.doi.org/10.1109/icc.2019.8762053. Disponível em: https://ieeexplore.ieee.org/document/8762053. Acesso em: 25 set. 2022.

LEHTINEN, Kim. **Scaling a Kubernetes Cluster**. 2022. 73 f. Dissertação (Mestrado) - Curso de Automation And Computer Science, University Of Vaasa, Vaasa, 2022. Disponível em: https://osuva.uwasa.fi/handle/10024/13971. Acesso em: 22 set. 2022.

NEWMAN, Sam. **Building Microservices**: designing fine-grained systems. 2. ed. [S.l.]: O'Reilly Media, 2021. 612 p.

NGUYEN, Thanh-Tung *et al*. Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration. **Sensors**, [S.l.], v. 20, n. 16, p. 4621, 17 ago. 2020. MDPI AG. http://dx.doi.org/10.3390/s20164621. Disponível em: https://www.mdpi.com/1424-8220/20/16/4621. Acesso em: 07 set. 2022.

PONISZEWSKA-MARANDA, Aneta; CZECHOWSKA, Ewa. Kubernetes Cluster for Automating Software Production Environment. **Sensors**, [S.l.], v. 21, n. 5, p. 1910, 9 mar. 2021. MDPI AG. http://dx.doi.org/10.3390/s21051910. Disponível em: https://www.mdpi.com/1424-8220/21/5/1910. Acesso em: 13 set. 2022.

VAYGHAN, Leila Abdollahi *et al*. Kubernetes as an Availability Manager for Microservice Applications. **Arxiv**, [S.l.], p. 1-10, 2019. ArXiv. http://dx.doi.org/10.48550/ARXIV.1901.04946. Disponível em: https://arxiv.org/abs/1901.04946. Acesso em: 22 set. 2022.

ZHU, Changpeng; HAN, Bo; ZHAO, Yinliang. A bi-metric autoscaling approach for n-tier web applications on kubernetes. **Frontiers Of Computer Science**, [S.l.], v. 16, n. 3, p. 1-12, 27 set. 2021. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s11704-021-0118-1. Disponível em: https://link.springer.com/article/10.1007/s11704-021-0118-1. Acesso em: 07 set. 2022.

FORMULÁRIO DE AVALIAÇÃO BCC – PROFESSOR AVALIADOR – PRÉ-PROJETO

Avaliador(a): Francisco Adell Péricas

Atenção: quando o avaliador marcar algum item como atende parcialmente ou não atende, deve obrigatoriamente indicar os motivos no texto, para que o aluno saiba o porquê da avaliação.

	ASPECTOS AVALIADOS	Atende	atende parcialmente	não atende
ASPECTOS TÉCNICOS	1. INTRODUÇÃO O tema de pesquisa está devidamente contextualizado/delimitado?			
	O problema está claramente formulado?			
	2. OBJETIVOS O objetivo principal está claramente definido e é passível de ser alcançado?			
	Os objetivos específicos são coerentes com o objetivo principal?			
	3. TRABALHOS CORRELATOS São apresentados trabalhos correlatos, bem como descritas as principais funcionalidades e os pontos fortes e fracos?			
	4. JUSTIFICATIVA Foi apresentado e discutido um quadro relacionando os trabalhos correlatos e suas principais funcionalidades com a proposta apresentada?			
	São apresentados argumentos científicos, técnicos ou metodológicos que justificam a proposta?			
	São apresentadas as contribuições teóricas, práticas ou sociais que justificam a proposta?			
	5. REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO Os requisitos funcionais e não funcionais foram claramente descritos?			
	6. METODOLOGIA Foram relacionadas todas as etapas necessárias para o desenvolvimento do TCC?			
	Os métodos, recursos e o cronograma estão devidamente apresentados e são compatíveis com a metodologia proposta?			
	7. REVISÃO BIBLIOGRÁFICA Os assuntos apresentados são suficientes e têm relação com o tema do TCC?			
	As referências contemplam adequadamente os assuntos abordados (são indicadas obras atualizadas e as mais importantes da área)?			
ASPECTOS METODOLÓGICOS	8. LINGUAGEM USADA (redação) O texto completo é coerente e redigido corretamente em língua portuguesa, usando linguagem formal/científica?			
	A exposição do assunto é ordenada (as ideias estão bem encadeadas e a linguagem utilizada é clara)?			