Infraestrutura como Mitigação: Um Estudo de Viabilidade do Uso do Auto-Scaling Contra Ataques de DoS em Ambiente de Microsserviços

Emanuel Ávila Cruz[1], **João Batista Andrade**[1], **João Henrique Corrêa** [1]

[1]Universidade Federal do Ceará (UFC) – Campus de Itapajé

{emanuel.pires,batistajoaoguns}@alu.ufc.br

joaocorrea@ufc.br

Abstract. *Denial-of-service (DoS) attacks are becoming more frequent and sophisticated, becoming a major challenge today, impacting companies negatively, and causing financial loss and damage to reputation. Furthermore, these attacks are generated to look like legitimate clients, making it difficult for detection/mitigation tools to do their job. Therefore, this paper proposes the use of the auto-scaling feature, available in the Kubernetes platform, to offer a higher availability during a DoS attack. It was verified during experiments that clients experienced an increase in availability and a decrease in service time when the auto-scaling strategy was enabled, during attacks.*

Resumo. *Os ataques de negação de serviço - Denial-of-Service (DoS) estão mais frequentes e sofisticados, tornando-se um grande desafio da atualidade, impactando negativamente as empresas, causando prejuízo financeiro e danos à reputação. Além disso, esses ataques são gerados visando assemelhar-se a clientes legítimos, dificultando a ação das ferramentas de detecção/mitigação. Portanto, este artigo propõe o uso da funcionalidade de dimensionamento automático, disponível na plataforma Kubernetes, para oferecer uma maior disponibilidade durante um ataque DoS. Verificou-se, durante a realização de experimentos, que os clientes tiveram um aumento da disponibilidade e uma diminuição do tempo de serviço quando a estratégia de dimensionamento automático estava ativada, durante os ataques.*

1. Introdução

Ataque de negação de serviço - *Denial-of-Service* (DoS), e sua forma distribuída - *Distributed Denial-of-Service* (DDoS), visam gerar a indisponibilidade de um serviço ou até causar a lentidão do mesmo. Tais ataques representam 14% de todos os ataques em ambientes de nuvem [Beigi-Mohammadi et al. 2016]. De modo geral, os ataques de negação de serviço consistem em sobrecarregar seu alvo, uma aplicação, com solicitações ilegítimas, dificultando ou impossibilitando o acesso de um cliente autêntico [Cloudflare 2020]. Podendo vir de uma única origem (DoS) ou de múltiplas origens simultâneas (DDoS), o que os torna mais difíceis de serem atenuados.

Conforme o Relatório Global de Ameaças Cibernéticas da Radware [Radware 2022], houve um aumento significativo nos ataques DDoS em 2021,

com um crescimento de 150% em comparação com o ano anterior. Além de serem mais volumosos, os ataques se mostraram mais frequentes e complexos, atingindo principalmente os setores financeiros, tecnológicos e de saúde.

O relatório menciona que esse aumento substancial pode ser atribuído aos conflitos geopolíticos entre potências mundiais, resultando no patrocínio de ataques cibernéticos, além das vulnerabilidades deixadas pela adaptação abrupta ao trabalho remoto decorrente da pandemia de Covid-19. Esses fatores destacam a necessidade de recursos para a mitigação de tais ofensivas.

No segundo semestre de 2021 [Microsoft 2022], foi detectado na plataforma de nuvem Azure o que acreditam ser o maior ataque de DDoS já registrado na história, gerando um pico de tráfego de 3,47 Tb/s, com duração de 15 minutos. Esse e os demais ataques registrados nesse período demonstram uma predominância de ofensivas curtas, o que evidencia a clara necessidade de soluções imediatas para lidar com essa problemática.

Em 2023, os ataques de DDoS atingiram novos patamares. Esses ataques foram altamente complexos e exploraram uma vulnerabilidade do protocolo HTTP/2. Com isso, no terceiro trimestre do ano de 2023, houve um ataque de 201 milhões de requisições por segundo - *Requests per Second* (RPS). Isso é quase 8 vezes maior do que o recorde anterior de 2022, que era de 26 milhões de RPS [Cloudflare 2023].

Por outro lado, devido às características dos ataques de DDoS, que muitas vezes se passam por requisições legítimas [Zargar et al. 2013], e à dificuldade de detectar esses ataques, as estratégias de mitigação que realizam descarte/bloqueio do tráfego têm a probabilidade de atingir e negar serviço a clientes legítimos [Corrêa et al. 2021]. Dado o exposto, o presente artigo tem como hipótese a utilização da infraestrutura como meio de mitigação, analisando a viabilidade de utilizar a função de dimensionamento automático horizontal - *Horizontal Pod Autoscaler* (HPA) - para oferecer mais disponibilidade ao microsserviço que esteja sofrendo um ataque de negação de serviço.

Com isso, verifica-se a estratégia de utilizar o dimensionamento automático para lidar com ataques de negação de serviço, pois permite que a arquitetura de microsserviços possa adequar sua capacidade de atendimento conforme a demanda. Salienta-se que a funcionalidade de dimensionamento automático não foi criada para este fim específico. Dessa forma, o artigo busca verificar a viabilidade de um novo uso da funcionalidade. O uso do *auto-scaling*, conforme a proposta, é para que a infraestrutura ofereça disponibilidade a clientes, enquanto uma estratégia mais adequada de mitigação, conforme o ataque, seja acionada.

Há uma proposta semelhante, mas realizada em máquinas virtuais [Corrêa et al. 2019]. Contudo, este artigo busca verificar, a partir do orquestrador de contêineres Kubernetes, a utilização da infraestrutura como meio de ferramenta auxiliar na mitigação de ataques de negação de serviço. Em resumo, as principais contribuições deste artigo são:

- A proposta de utilizar o mecanismo de dimensionamento automático, originalmente criado para lidar com demanda variável, na mitigação de ataques de negação de serviço em um *cluster* Kubernetes;
- A análise das métricas de Qualidade de Serviço - *Quality-of-Service* (QoS) - de clientes legítimos durante um DoS e após a ativação da proposta;

- A verificação da eficácia da proposta em um ambiente real, com tráfego de ataque.

O restante deste artigo é organizado da seguinte forma: na Seção 2, são descritos alguns trabalhos relacionados ao tema abordado; na Seção 3, são apresentadas as ferramentas habilitadoras da proposta; na Seção 4, são abordados os cenários utilizados nos experimentos e a análise dos resultados obtidos durante os testes; e, por fim, na Seção 5, são apresentadas as conclusões e as perspectivas de trabalhos futuros.

2. Trabalhos Relacionados

Em uma revisão da literatura, Tran *et al.* [Tran et al. 2022] fazem referência ao uso do Kubernetes, um orquestrador de contêineres mais popular utilizado pelos maiores provedores de serviço, como *Amazon Web Service* (AWS) e *Google Cloud Platform* (GCP). Eles acrescentam também que o escalonamento é um recurso vital para atender à demanda de QoS das aplicações, mas que, no entanto, em suas configurações padrão, tem desempenho de adaptação lento contra cargas de trabalho dinâmicas. No mesmo artigo, é feito um estudo de melhorias do escalonador padrão com variação nas diversas características presentes, como diferentes arquiteturas de aplicações de destino, escalonamento baseado em horizontal ou vertical, operação reativa ou proativa, adoção ou não de técnicas de aprendizado de máquina. Apesar de Tran *et al.* não tratarem de negação de serviço, seu trabalho pode trazer grandes benefícios em propostas de trabalhos futuros com a implementação dos métodos citados.

No artigo [Dewi et al. 2019], foi apresentado um estudo de caso no qual foi implementado o uso do Kubernetes para melhorar a escalabilidade de seus servidores, comparando dois cenários - de servidor único e de múltiplos servidores - e as taxas de resposta aos usuários. Este artigo difere do presente trabalho, pois se buscou fazer um novo uso da escalabilidade do Kubernetes para aumentar a disponibilidade do serviço, mitigando o impacto dos ataques de negação de serviço no QoS.

Em [He 2020], é proposta uma estratégia de escalonamento responsivo para a plataforma Kubernetes. No artigo, He defende que o escalonamento do Kubernetes é lento e, por isso, acaba prejudicando a qualidade do serviço *web*. Para mitigar isso, ele propõe a utilização de um modelo estatístico de análise temporal, chamado ARIMA, com um algoritmo próprio para descobrir o melhor número de réplicas para os serviços. O presente artigo visa testar o *auto-scaling* e, com isso, verificar se essa funcionalidade pode ser utilizada para mitigar ataques de negação de serviço. No entanto, o presente artigo pode se beneficiar da estratégia apresentada.

Em [Beigi-Mohammadi et al. 2020], é proposta uma abordagem relacionada ao dimensionamento automático nos provedores de *Cloud*, auxiliando na prevenção da geração de custos desnecessários nas instâncias dos clientes. É proposta a utilização da Virtualização de Função de Rede - *Network Function Virtualization* (NFV) - e Redes Definidas por Software - *Software Defined Network* (SDN) - para a criação de um programa que possa ser utilizado como Interface de Programação de Aplicação - *Application Programming Interface* (API). No entanto, o presente artigo busca utilizar a função nativa da infraestrutura do Kubernetes, o *auto-scaling*, para manter a disponibilidade e a QoS aos clientes durante um ataque de DoS.

No artigo [Zhao et al. 2019], foi apresentado um dimensionador automático preditivo em um *cluster* Kubernetes para melhorar a eficiência do escalonamento automático de

contêineres. O sistema de dimensionamento automático fornecido pode prever o número de *pods* e expandir a capacidade com antecedência, reduzindo o tempo de resposta. O módulo escalonador dimensiona o *cluster* com base na predição fornecida pelo módulo de previsão. No caso do aumento do escalonamento, um algoritmo de previsão de suavização exponencial dupla é usado para previsão. Em caso de redução, não pode ser usado o algoritmo de previsão, pois reduzir o tamanho dos serviços atuais por previsão implica na possibilidade de falha do serviço. A diferença entre o artigo apresentado e o proposto é a finalidade, visto que o presente artigo visa mitigar ataques de negação de serviço, enquanto o de Zhao *et al.* é melhorar a eficiência do escalonamento automático de *pods*. Apesar da finalidade distinta, a proposta do presente artigo pode se beneficiar dos algoritmos propostos por Zhao *et al.* para pesquisas futuras.

3. Ferramentas habilitadoras para a prover a infraestrutura como mitigação

Nesta seção, será explorado a infraestrutura, juntamente com ferramentas e tecnologias que podem ser utilizadas para mitigar os ataques DoS, focando em uma abordagem baseada em microsserviços com Kubernetes, Prometheus e escalabilidade elástica na nuvem.

3.1. Microsserviços com Kubernetes

Microsserviços estão se tornando uma arquitetura bastante popular, especialmente entre as *big techs*, como as aplicações Gmail, YouTube, Google Search (2021), Netflix Hall (2018) e os serviços financeiros do PayPal (2017) [Wong et al. 2023], devido à sua capacidade de organizar uma aplicação complexa em pequenos serviços. Com isso, ganha-se autonomia para obter escalabilidade, isolamento, carga de trabalho dedicada e velocidade de comunicação, entre outros benefícios [Laigner et al. 2021].

Kubernetes[1], também conhecido como K8s, é um sistema de código aberto certificado pela *Cloud Native Computing Foundation* (CNCF), projetado para facilitar a implantação de microsserviços. Este sistema automatiza a implantação, escalonamento e gerenciamento de aplicações em contêineres, resultando em redução significativa do tempo de implantação, da carga de trabalho e dos custos associados às aplicações baseadas em microsserviços [Liu et al. 2020].

O Kubernetes possui muitos componentes, cada um atendendo a uma finalidade diferente [Shah and Dubaria 2019]. Para garantir a separação de tarefas e a resiliência do sistema, o Kubernetes utiliza o conceito de *clusters* em máquinas físicas ou virtuais. Na Figura 1, são detalhados o funcionamento e a arquitetura do Kubernetes. No nó *master* estão localizados os principais componentes do *cluster* Kubernetes como, por exemplo, o gerenciador de recursos - Escalonador. No servidor de API é fornecida a interação, onde todos os outros componentes podem se comunicar entre si, inclusive o administrador do *cluster*. Os *pods* são os menores e mais básicos objetos implantáveis no Kubernetes [Shah and Dubaria 2019]. É neste elemento que estarão os contêineres e, especificamente, os microsserviço.

Os *pods* são conjuntos de um ou mais contêineres e são agrupados para que os recursos sejam compartilhados de modo inteligente. A organização dos *pods* é a base de uma das funcionalidades mais usadas do Kubernetes, a replicação. Além de ajudar a

[1]https://kubernetes.io/ [acessado 21 de março de 2024]

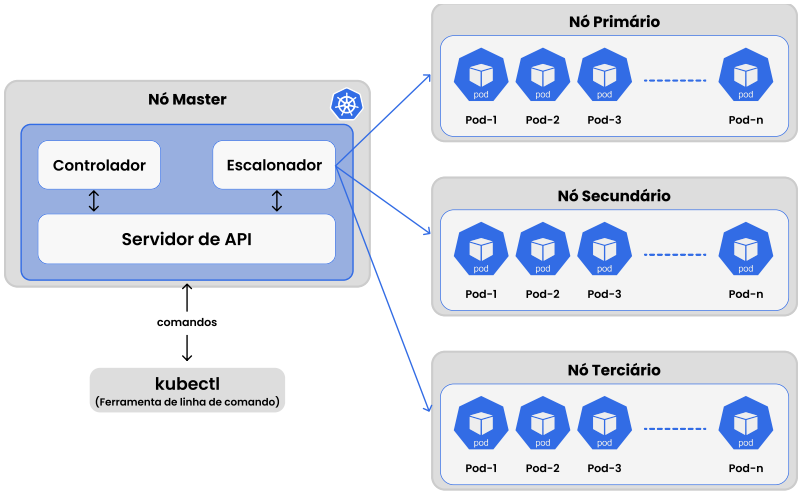

Figura 1. Arquitetura do Kubernetes. Adaptado de [Perveez 2020].

manter o funcionamento correto das operações durante os períodos de sobrecarga, os *pods* são replicados conforme a necessidade do sistema para permanecer resistente a falhas, oferecendo mais recursos conforme a demanda.

3.2. Prometheus

Prometheus[2] é um conjunto de ferramentas de monitoramento e alerta de sistemas de código aberto. Essas ferramentas coletam e armazenam métricas como dados de séries temporais, ou seja, as informações das métricas são armazenadas com o carimbo de data/hora em que foram registradas, juntamente com pares chave-valor opcionais chamados rótulos. Esses rótulos podem incluir informações sobre a fonte de dados, como o *namespace* do serviço, nome do *cluster*, etc.

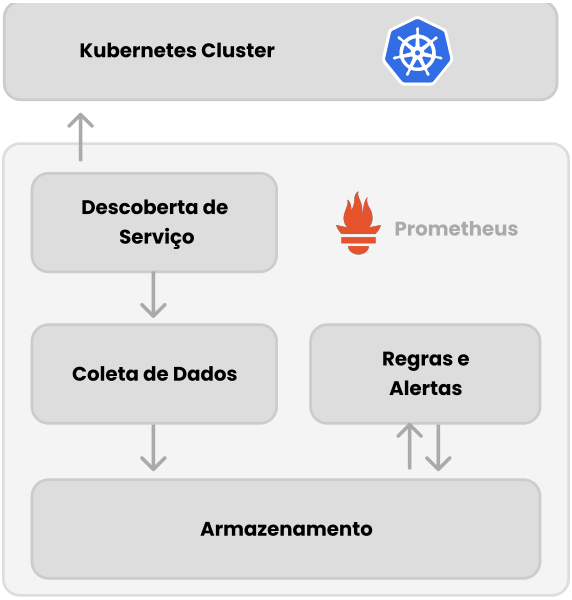

Figura 2. Arquitetura do Prometheus. Adaptada de [Prometheus 2015].

[2]Disponível em: https://prometheus.io/docs/introduction/overview/ [acessado em 21 de março de 2024]

Na Figura 2 é apresentada a arquitetura do Prometheus. Ele extrai métricas de contêineres do *cluster* Kubernetes, diretamente ou por meio de um *gateway push* intermediário. O Prometheus armazena todas as amostras coletadas localmente e executa regras sobre esses dados para agregar e registrar novas séries temporais de dados existentes. Tanto o Grafana[3] quanto outras ferramentas podem ser utilizados para visualizar os dados coletados.

No contexto do Kubernetes, o Prometheus pode ser utilizado para analisar o desempenho de um *cluster*, monitorando as aplicações contidas em *pods*. Tendo isso em vista, o presente trabalho empregou o Prometheus para coletar métricas de telemetria no consumo de CPU, memória, disco e rede. Com essas métricas em mãos, é possível obter percepções sobre o comportamento das aplicações.

Dentre as várias opções de mecanismos de monitoramento de microsserviços no Kubernetes, o presente artigo optou por utilizar o Prometheus. Esta escolha se deve ao fato de o Prometheus ser uma ferramenta de código aberto e gratuita, além de ser um projeto graduado pelo CNCF, o que indica um alto nível de maturidade, refletindo uma ampla adoção e uma base de código segura [Almaraz-Rivera 2023].

3.3. Dimensionamento automático

O dimensionamento automático é a capacidade do sistema de expandir ou reduzir sua capacidade computacional conforme a demanda, mantendo o desempenho adequado. Em outras palavras, o sistema pode ajustar dinamicamente seus recursos para lidar com variações na carga de trabalho [Shim et al. 2023].

O orquestrador de contêineres Kubernetes oferece suporte para o dimensionamento automático, sendo denominados *Horizontal Pod Autoscaler* (HPA) e *Vertical Pod Autoscaler* (VPA). No HPA, o número de réplicas de uma aplicação é ajustado horizontalmente, ou seja, mais réplicas são adicionadas ou removidas conforme necessário. O *auto-scaling* horizontal é baseado em um limiar definido, como o uso de CPU ou memória, ambos em porcentagem. Quando a carga aumenta e ultrapassa o limiar especificado, os *pods* são replicados para distribuir a carga de trabalho e garantir uma melhor capacidade de resposta. Da mesma forma, quando a carga diminui, as réplicas são reduzidas para economizar recursos [Balla et al. 2020].

No VPA, o dimensionamento é realizado ajustando os recursos de uma réplica individual da aplicação, em vez de adicionar ou remover réplicas. Isso pode incluir aumentar ou diminuir a quantidade de recursos computacionais alocados para uma réplica. Embora o *auto-scaling* vertical seja menos comum no Kubernetes, pode ser útil em casos específicos em que a escalabilidade horizontal não é suficiente [Balla et al. 2020]. O Kubernetes faz uso de ambos os formatos de *auto-scaling*. No entanto, o modo horizontal é mais comum e amplamente utilizado na maioria dos cenários devido à sua eficácia, escalabilidade e facilidade de implementação. Ele permite uma adaptação dinâmica à demanda, garantindo a disponibilidade e o desempenho ideal das aplicações em um ambiente altamente dinâmico.

Na Figura 3 é exemplificado o funcionamento do HPA, onde uma métrica previ-

[3]Sistema de visualização de monitoramento. Disponível em: https://grafana.com/ [acessado em 28 de março de 2024]

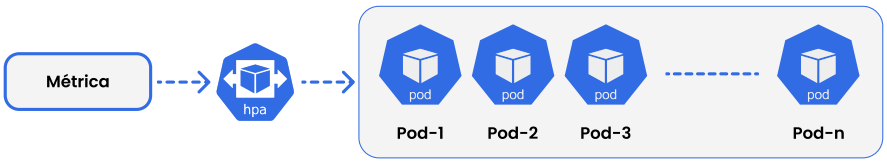

Figura 3. Funcionamento do HPA.

amente configurada, seja de CPU ou memória, é monitorada pela configuração do HPA. Se o limiar definido na configuração for ultrapassado, o *auto-scaling* do Kubernetes cria uma nova instância do *pod* configurado. A quantidade total de réplicas também pode ser definida na configuração do HPA, levando em consideração os recursos de *hardware* disponíveis no *cluster*.

Dessa forma, o HPA é um mecanismo desenvolvido para se adaptar à demanda de um microsserviços. Diante disso, o presente artigo visa explorar uma nova aplicação da funcionalidade de dimensionamento automático do Kubernetes: mitigar ataques de negação de serviço. Essa estratégia está alinhada ao fato de que os ataques de DoS se assemelham a requisições de clientes legítimos, o que dificulta a detecção e a mitigação por meio do bloqueio ou descarte desse tráfego. A estratégia se adequa como uma defesa temporária, oferecendo uma disponibilidade do serviço pelo maior tempo possível, até que alguma defesa mais direcionada seja acionada. Dessa forma, com o mecanismo proposto, a arquitetura da plataforma ganha tempo para realizar a identificação do ataque (por métodos tradicionais ou por meio de algoritmos de Aprendizado de Máquina) e assim, o acionamento das as diversas propostas e mecanismos existentes na literatura podem ser acionadas para mitigar efetivamente o ataque.

4. Experimentos e Resultados

Para verificar a viabilidade de utilizar a infraestrutura como forma de mitigação, foram criados cenários e realizados diversos experimentos, visando analisar se a funcionalidade de *auto-scaling* pode ser utilizada para mitigar ataques de negação de serviço. Esta seção consiste, na primeira parte, na descrição dos cenários e na realização dos experimentos, enquanto na segunda parte são apresentados os resultados e discutidos os mesmos.

4.1. Cenários de experimentos

Para a realização dos experimentos, foi utilizado o Kubernetes na versão 1.27, executado em uma máquina com 6 GB de memória RAM e CPU Intel Xeon E-2224 3,40 GHz, utilizando o Sistema Operacional *Ubuntu Server* 20.04.6 LTS. No Kubernetes, foram criados vários *pods*: o primeiro com o Prometheus para monitoramento das métricas; outro *pod* com o sistema *Server-Metrics*, responsável pela integração das métricas com o Kubernetes; além de um *pod* com o servidor *web* Nginx[4], que será a vítima do ataque de negação de serviço. Por fim, foi habilitada a configuração necessária para o HPA no *pod web*.

Para a geração do tráfego de clientes, foi utilizada a ferramenta Siege[5], que é uma aplicação de teste de carga que permite realizar requisições semelhantes às de usuários

[4]https://www.nginx.com/ [acessado 25 de março de 2024]
[5]https://linux.die.net/man/1/siege [acessado 25 de março de 2024]

para um servidor de forma simultânea. Ao final da sua execução, o Siege emite um relatório completo com diversas métricas relacionadas à QoS, sob a perspectiva dos clientes. Foram gerados 2000 clientes legítimos, que enviavam requisições HTTP GET ao servidor *web* durante todo o experimento, e cada cliente variava entre uma requisição e outra um tempo aleatório entre 0 e 3 segundos. A escolha da quantidade de clientes foi baseada na capacidade de atendimento do servidor *web* Nginx com sua configuração padrão. Essa quantidade é o limite em que o servidor consegue responder a todos os clientes sem que haja alteração em suas métricas. A ideia é manter o servidor em sua capacidade máxima, mas que a indisponibilidade seja causada pelo ataque em si.

Para o ataque de negação de serviço, foi escolhido o ataque de inundação de requisições GET - GET-Flood, gerado por meio da ferramenta Goldeneye[6]. Esse ataque envia inúmeras solicitações HTTP GET, semelhantes a clientes legítimos, ao servidor de destino, visando consumir seus recursos e torná-lo indisponível para usuários. No Goldeneye, foram utilizados os seguintes parâmetros: 10 *workers*, com 500 *sockets* cada *worker*. Essa configuração representa a utilização de 5000 atacantes distintos inundando o servidor *web*.

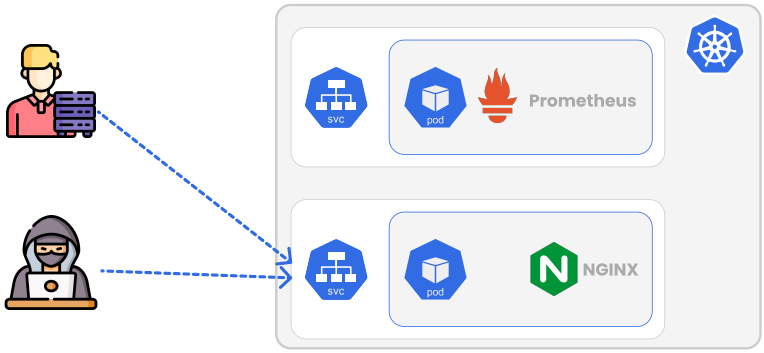

Figura 4. Cenário com clientes e atacantes sem *auto-scaling*.

Foram criados dois cenários para a realização dos experimentos:

1. No primeiro cenário, foram simuladas requisições de clientes legítimos e de atacantes, porém sem a funcionalidade do *auto-scaling* ativada. O objetivo desse cenário é verificar se o ataque está sendo efetivo e se os clientes estão tendo o serviço negado devido à sobrecarga do servidor.
2. No segundo cenário, também foram incluídas requisições de clientes legítimos e de atacantes, porém com a funcionalidade do HPA ativada. Neste cenário, o objetivo é verificar se a funcionalidade de *auto-scaling* pode ser utilizada para oferecer uma maior disponibilidade aos clientes legítimos, ajustando dinamicamente a capacidade do servidor em resposta à carga de trabalho.

4.1.1. Cenário 1 - Clientes e atacantes sem *auto-scaling*

O cenário ilustrado na Figura 4 pretende verificar as métricas dos clientes durante um ataque de negação de serviço, sem a implementação de qualquer medida defensiva. Para isso,

[6]https://github.com/jseidl/GoldenEye [acessado 25 de março de 2024]

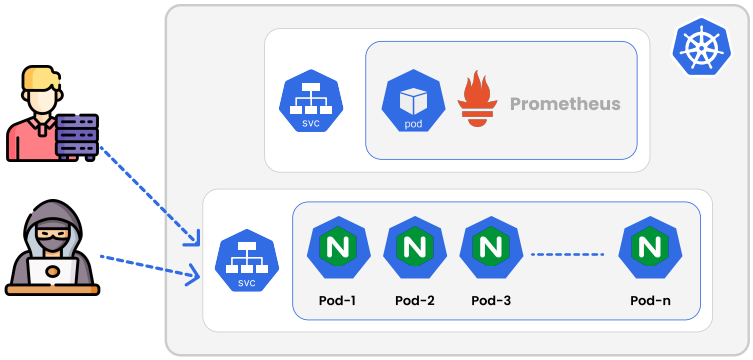

Figura 5. Cenário com clientes e atacantes com *auto-scaling*.

foi configurado um ambiente Kubernetes com um *pod* contendo um servidor *web* Nginx, responsável por receber requisições tanto de clientes legítimos quanto de atacantes. O cliente legítimo, simulado pela ferramenta Siege, conforme explicado anteriormente, foi executado em um computador com um processador Intel Core i3, 8 GB de memória RAM e 256 GB de SSD, conectado à mesma rede local que o servidor *web*. Da mesma forma, o atacante também está na mesma rede local, em outro computador com configurações similares, executando a ferramenta de ataque Goldeneye. É importante notar, ainda na Figura 4, a presença de um segundo *pod* contendo o Prometheus, responsável por coletar as métricas dos experimentos. Além disso, não há nenhuma configuração de *auto-scaling* habilitada.

4.1.2. Cenário 2 - Clientes e atacantes com *auto-scaling*

Neste cenário, apresentado na Figura 5, também há clientes e atacantes realizando requisições ao *pod* instanciado em uma infraestrutura de Kubernetes. No entanto, neste cenário, o servidor *web* é dotado com a funcionalidade de *auto-scaling*, com o HPA. O HPA funciona da seguinte forma: o Prometheus monitora os recursos de um determinado *pod*. A partir de um limiar definido, e a métrica ultrapassando esse limiar, é acionado então a funcionalidade de *auto-scaling*, adicionando a quantidade de réplicas que forem necessárias para diminuir o uso do recurso abaixo do limiar escolhido. No caso deste cenário, o limiar estabelecido foi de 70% de uso da CPU. Além disso, na configuração desse *pod*, foi estabelecido a criação inicial com 1 réplica e a quantidade máxima de 10 réplicas, com o intuito de definir a quantidade máxima de recursos utilizados da infraestrutura.

Vale salientar que neste cenário há um módulo de serviço que trabalha como *Load Balance* (LB), que, devido à quantidade de réplicas a serem criadas, se faz necessário para haver uma divisão das requisições entre todas as réplicas. O LB é criado automaticamente pelo Kubernetes, e utiliza o algoritmo de balanceamento *Round Robin*, sem dar preferência para nenhum *pod*.

Em ambos os cenários, duas métricas foram utilizadas para verificar a eficácia de utilizar o mecanismo de *auto-scaling* como forma de mitigação de ataques de DoS: a disponibilidade e o tempo de serviço - *Time-to-Service* (TTS). A primeira, a disponibilidade, é a quantidade de clientes legítimos, gerados pela ferramenta Siege, que obtiveram su-

cesso ao requisitar uma página ao servidor *web*. Essa métrica visa verificar o quanto de clientes legítimos foram impactados com o ataque. A segunda métrica, o TTS, é o tempo total em que o cliente solicitou uma página, o servidor processou a solicitação e a resposta chegou ao cliente. Tanto a Disponibilidade quando o TTS são métricas oferecidas pela ferramenta Siege, que cria as requisições de clientes legítimos e são métricas relacionadas a QoS desses clientes.

4.2. Resultados

Com a infraestrutura configurada e cenários criados, foram realizados dez experimentos para cada cenário, cada um com duração de 15 minutos. Os resultados apresentados a seguir serão a média das métricas coletadas, bem como o desvio padrão.

Na Figura 6 é apresentado o resultado da métrica de disponibilidade, mostrado em porcentagem. Essa métrica é a quantidade de clientes legítimos que realizaram uma requisição ao servidor *web* e foram atendidos. Na barra em azul, à esquerda da Figura 6, é apresentada a média da disponibilidade nos experimentos em que a funcionalidade de *auto-scaling* não está ativa. A barra em amarelo, à direita, é o resultado do experimento com a funcionalidade de *auto-scaling* ativa.

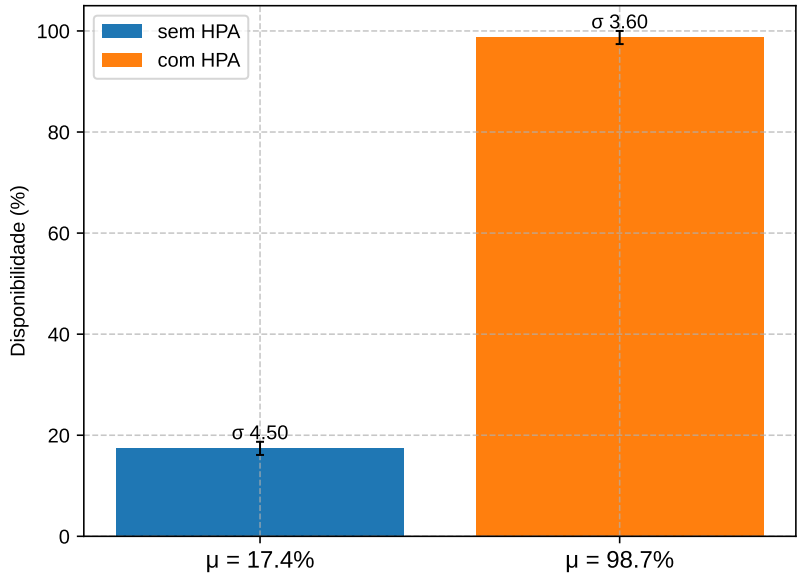

Figura 6. Resultado comparativo da disponibilidade sem e com HPA.

Verifica-se na Figura 6 que, aproximadamente, 17% dos clientes legítimos conseguiram ser atendidos pelo servidor *web* durante um ataque de negação de serviço, com um desvio padrão de 4,5. Isso demonstra que o ataque realizado neste cenário está sendo efetivo, levando a maioria dos clientes a ficarem sem atendimento pelo servidor *web*. Por outro lado, com o mecanismo de *auto-scaling* ativado, 98,7% dos clientes legítimos foram atendidos, tendo um desvio padrão de 3,6. Um aumento considerável na disponibilidade dos clientes.

Na Figura 7 estão os resultados do TTS. Essa métrica é o tempo total em que o cliente solicita a página *web* ao servidor, e o servidor retorna uma resposta ao cliente. A barra em azul, à esquerda, mostra a média do TTS no experimento em que não há o

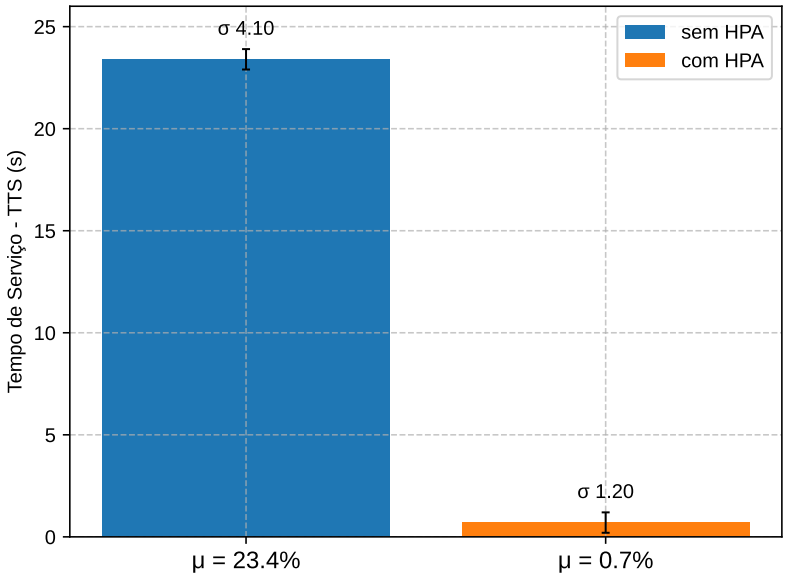

Figura 7. Resultado comparativo do Tempo de Serviço (TTS) com e sem o HPA.

auto-scaling. Enquanto a barra em amarelo, à direita, é o TTS utilizando a proposta de defesa.

Nos experimentos sem o *auto-scaling*, verificou-se uma média de 23 segundos para que o cliente solicitasse e recebesse a página do servidor *web*. Esse tempo é exageradamente demorado, o que leva a uma degradação da QoS e de Experiência do usuário. Por outro lado, a funcionalidade de *auto-scaling*, utilizada como estrategia de mitigação, oferece uma redução do TTS para os clientes, constatando que este cliente espera, em média, apenas 0,7 segundo para receber a página. Valor aceitável para um serviço que está passando por um ataque de negação de serviço. Dessa forma, os resultados apresentados nas Figuras 6 e 7 mostram que a funcionalidade de *auto-scaling* em sua configuração padrão, auxilia na mitigação de um ataque de DoS e na disponibilidade do serviço.

Por fim, na Figura 8 é apresentado o gráfico de evolução da métrica de CPU de todas as instâncias geradas em um experimento. Vale salientar que a métrica de consumo de *Computer Processing Unit* (CPU) é a métrica utilizada para ativar o *auto-scaling*. No eixo x, tem-se o tempo, em segundos, do experimento. Já no eixo y, é apresentado a média da porcentagem de utilização da CPU de todas as instâncias. A linha em vermelha é o consumo do recurso e a linha em cinza, tracejada, representa o limiar, de 70%, do uso da CPU.

Verifica-se, na Figura 8, que quando o consumo de CPU ultrapassa o limiar, depois de um certo momento a utilização de CPU começa a diminuir. Esse é o momento em que o *auto-scaling* entra em funcionamento e instancia uma réplica do serviço. Logo em seguida, há uma queda no consumo, pois é o balanceamento de carga funcionando e evitando que a indisponibilidade aconteça a clientes legítimos. Finalizando, ainda na Figura 8, verifica-se que durante todo o experimento, com duração de 15 minutos, houve a criação de 4 novos *pods*, totalizando 5 instâncias, oferecendo mais disponibilidade aos clientes legítimos.

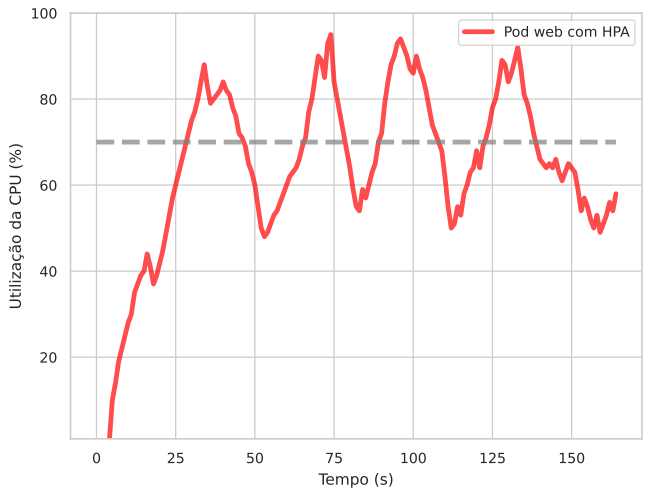

Figura 8. Evolução da utilização de CPU durante o experimento com HPA ativo.

5. Conclusão e Trabalhos Futuros

Os DoS, e sua forma distribuída (DDoS), visam gerar a indisponibilidade do serviço a clientes legítimos e por consequência degradam o QoS que o cliente deveria ter. São agravantes ainda, pois alguns ataques têm características similares a clientes legítimos, dificultando a detecção desses ataques e por consequência sua mitigação, pois, por se assemelhar a clientes, a mitigação dos ataques podem acabar afetando clientes legítimos.

Diante desse cenário, o presente artigo propôs a verificação da viabilidade de utilizar da infraestrutura como mitigação de ataques de negação de serviço. A ideia consiste em utilizar a funcionalidade de *auto-scaling*, disponível, de forma nativa, na infraestrutura do Kubernetes, para mitigar os ataques de DoS. Apesar de não ser criado para este fim, a proposta é dar um novo uso a essa ferramenta, adaptando os *pods* à demanda exigida pelo ataque.

Para verificar essa proposta, foram criados cenários em que clientes e atacantes realizavam requisições a um servidor *web*. Em um cenário, a funcionalidade de *auto-scaling* estava desativada, para verificar a eficácia do ataque. Em outro cenário, o *auto-scaling* foi acionado, verificando se mitigava o ataque. Os resultados mostraram que houve um aumento na disponibilidade de clientes, saindo de 17,4% no cenário sem a funcionalidade para 98,7% de disponibilidade com o *auto-scaling* em funcionamento. Além disso, verificou-se uma diminuição do tempo de serviço (*Time-to-Service* - TTS) ao cliente, saindo de 23 segundos no cenário sem o *auto-scaling* para 0,7 segundo, com a proposta.

Conclui-se que, em suas configurações padrões, o *auto-scaling*, em específico o HPA, é capaz de satisfazer as expectativas, fornecendo uma maior disponibilidade para os clientes legítimos e evitando uma degradação da Qualidade de Serviço. Dessa forma, o presente artigo verificou ser viável a utilização da ferramenta de *auto-scaling* para mitigar ataques de negação de serviço. Obviamente há uma limitação dessa proposta, pois a mitigação é efetiva até o limite dos recursos computacionais disponíveis na infraestrutura. No entanto, a estratégia pode ser utilizada para suportar o tráfego, até que um mecanismo

de defesa mais robusto seja acionado, realizando a identificação e mitigação de forma mais eficiente.

Como trabalhos futuros, pretende-se testar a efetividade da proposta com outras classes e tipos de ataques, como ataques de inundação e de amplificação. Além disso, há a proposta de gerar um sistema de defesa que utilize diversos métodos de mitigação, utilizando as ferramentas e funcionalidades que o ambiente de nuvem oferece, instanciando esses mecanismos de defesa sob demanda, utilizando o paradigma de NFV.

6. Agradecimentos

Este trabalho recebeu financiamento de bolsas de Iniciação Científica (PIBIC) da UFC e do Programa de Educação Tutorial da UFC (PET-UFC). Além disso, os autores gostariam de agradecer o financiamento proveniente da Fundação de Apoio à Pesquisa do Estado de São Paulo (FAPESP) - Projeto de Pesquisa 2018/23097-3.

Referências

Almaraz-Rivera, J. G. (2023). An anomaly-based detection system for monitoring kubernetes infrastructures. *IEEE Latin America Transactions*, 21(3):457–465. Acesso em: 4 jan. 2024.

Balla, D., Simon, C., and Maliosz, M. (2020). Adaptive scaling of kubernetes pods. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–5.

Beigi-Mohammadi, N., Barna, C., Shtern, M., Khazaei, H., and Litoiu, M. (2016). Caamp: Completely automated ddos attack mitigation platform in hybrid clouds. In *2016 12th International Conference on Network and Service Management (CNSM)*, pages 136–143.

Beigi-Mohammadi, N., Shtern, M., and Litoiu, M. (2020). Adaptive load management of web applications on software defined infrastructure. *IEEE Transactions on Network and Service Management*, 17(1):488–502.

Cloudflare (2020). O que é um ataque de negação de serviço (dos)? `https://www.cloudflare.com/pt-br/learning/ddos/glossary/denial-of-service/`. Acessado em 21 de junho de 2023.

Cloudflare (2023). Ddos threat report for 2023 q4. `https://blog.cloudflare.com/ddos-threat-report-2023-q4`. Acessado em 10 de janeiro de 2024.

Corrêa, J. H., Ciarelli, P. M., Ribeiro, M. R., and Villaça, R. S. (2021). Ml-based ddos detection and identification using native cloud telemetry macroscopic monitoring. *Journal of Network and Systems Management*, 29:1–28.

Corrêa, J. H. G. M., Sousa Junior, E. A., Fonseca, I. E., Nigam, V., Ribeiro, M. R. N., and Villaça, R. S. (2019). Selectivity and autoscaling as complementary defenses for ddos protection to cloud services. In *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, pages 1–3.

Dewi, L. P., Noertjahyana, A., Palit, H. N., and Yedutun, K. (2019). Server scalability using kubernetes. In *2019 4th technology innovation management and engineering science international conference (TIMES-iCON)*, pages 1–4. IEEE.

He, Z. (2020). Novel container cloud elastic scaling strategy based on kubernetes. In *2020 IEEE 5th information technology and mechatronics engineering conference (ITOEC)*, pages 1400–1404. IEEE.

Laigner, R., Zhou, Y., Salles, M. A. V., Liu, Y., and Kalinowski, M. (2021). Data management in microservices: State of the practice, challenges, and research directions. *arXiv preprint arXiv:2103.00170*.

Liu, G., Huang, B., Liang, Z., Qin, M., Zhou, H., and Li, Z. (2020). Microservices: architecture, container, and challenges. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 629–635.

Microsoft (2022). Azure ddos protection—2021 q3 and q4 ddos attack trends. `https://azure.microsoft.com/en-us/blog/azure-ddos-protection-2021-q3-and-q4-ddos-attack-trends`. Acessado em 10 de janeiro de 2024.

Perveez, S. H. (2020). Understanding kubernetes architecture and its use cases. `https://www.simplilearn.com/tutorials/kubernetes-tutorial/kubernetes-architecture`. Acessado em 2023-05-29.

Prometheus (2015). Overview. `https://prometheus.io/docs/introduction/overview/`. Acessado: 21-06-2023.

Radware (2022). Radware-2022-global-threat-analysis-report. `https://www.radware.com/getattachment/b775fe9c-326f-4f97-b6c7-8545b5b68e42/Radware-2022-Global-Threat-Analysis-Report.pdf.aspx`. Acessado em 10 de janeiro de 2024.

Shah, J. and Dubaria, D. (2019). Building modern clouds: Using docker, kubernetes google cloud platform. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0184–0189.

Shim, S., Dhokariya, A., Doshi, D., Upadhye, S., Patwari, V., and Park, J.-Y. (2023). Predictive auto-scaler for kubernetes cloud. In *2023 IEEE International Systems Conference (SysCon)*, pages 1–8.

Tran, M.-N., Vu, D.-D., and Kim, Y. (2022). A survey of autoscaling in kubernetes. In *2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 263–265. IEEE.

Wong, A. Y., Chekole, E. G., Ochoa, M., and Zhou, J. (2023). On the security of containers: Threat modeling, attack analysis, and mitigation strategies. *Computers Security*, 128:103140.

Zargar, S. T., Joshi, J., and Tipper, D. (2013). A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys and Tutorials*, 15(4):2046–2069.

Zhao, H., Lim, H., Hanif, M., and Lee, C. (2019). Predictive container auto-scaling for cloud-native applications. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1280–1282. IEEE.