Análise do Nível de Isolamento entre Contêineres Durante Ataques de Negação de Serviço em Computação em Nuvem

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

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

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

Abstract. *Denial-of-Service (DoS) attacks aim to disrupt the availability of applications for legitimate users. In cloud computing environments like Kubernetes, such attacks can be harmful without proper isolation between applications. This study examines the isolation between containers in Kubernetes. Experiments revealed that metrics of one container were impacted by an attack on another within the same infrastructure. The findings suggest inadequate isolation between containers.*

Resumo. *Ataques de negação de serviço - Denial-of-Service (DoS), visam afetar a disponibilidade de aplicações para usuários legítimos. Em ambientes de computação em nuvem, como o Kubernetes, ataques podem ser prejudiciais sem isolamento entre aplicações. Este estudo analisa o isolamento entre contêineres em Kubernetes. Experimentos mostraram que métricas de um contêiner foram afetadas por um ataque em outro na mesma infraestrutura. Os resultados indicam falta de isolamento adequado entre contêineres.*

1. Introdução

Após o início da pandemia, houve um aumento exponencial nos ataques de negação de serviço - *Denial-of-Service* (DoS), conforme relatado pela Nokia em seu relatório de segurança de 2021[1]. Esses ataques cresceram mais de 40% em relação a 2019, atingindo uma taxa de transferência de 3 TB/s em 2021, o dobro da taxa de 2020. Esses números alarmantes destacam a crescente ameaça à segurança cibernética, ressaltando as perdas significativas que os ataques distribuídos (DDoS) podem causar, incluindo aspectos financeiros, produtivos e de reputação das empresas[2].

Mesmo que os sistemas sob ataque permaneçam operacionais, sua eficiência em responder às solicitações legítimas do serviço é prejudicada, afetando diretamente a experiência do usuário. A sobrecarga causada pelo tráfego malicioso consome recursos como largura de banda, capacidade de processamento e memória, resultando em tempos de resposta mais longos [Pelloso et al. 2018]. Isso pode levar à degradação do desempenho dos sistemas, prejudicando a navegação na aplicação, transações comerciais e o acesso a serviços críticos.

Diante desse contexto, o uso de ambientes em nuvem e contêineres tem gerado preocupações adicionais em relação à segurança. Embora ofereçam benefícios como disponibilidade e flexibilidade, também apresentam desafios relacionados ao isolamento de

[1]Disponível em: https://onestore.nokia.com/asset/211059. - acessado em 08 de abril de 2024.

[2]Relatório da Akamai disponível em: https://www.akamai.com/pt/glossary/what-is-ddos - acessado em 21 de junho de 2023.

recursos entre diferentes contêineres. O aumento dos ataques DoS ressalta a importância do isolamento de recursos para garantir que um contêiner sob ataque não afete negativamente outros no mesmo ambiente. Assim, este artigo propõe testar e analisar o nível de isolamento de recursos entre contêineres distintos, contribuindo para a compreensão e aprimoramento da segurança nesses ambientes. O objetivo é verificar se métricas de telemetria de um determinado contêiner são alteradas devido a outro contêiner, na mesma infraestrutura, estar recebendo um ataque de negação de serviço. Em resumo, as principais contribuições deste artigo são:

- Análise da existência do isolamento entre contêineres que estão instanciados em uma mesma infraestrutura em um *cluster* Kubernetes, durante um ataque de negação de serviço;
- A análise das métricas de Qualidade de Serviço (*Quality-of-Service* (QoS)) de clientes legítimos em um contêiner na mesma infraestrutura que outro que está recebendo um ataque de DoS;
- A verificação da análise em um ambiente real, com tráfego de ataque.

O artigo está organizado da seguinte forma: na Seção 2 são apresentados os artigos relacionados; na Seção 3 são abordadas as ferramentas e serviços em infraestruturas de contêineres; na Seção 4 são detalhados os cenários experimentais e a metodologia; na Seção 5 é feita uma análise do isolamento dos contêineres em ataques DoS; na Seção 6 são apresentadas as conclusões e propostas futuras.

2. Trabalhos Relacionados

O isolamento em contêineres não é um objeto de estudo novo. No artigo [Zhao et al. 2021], é apresentada uma tecnologia de controle de isolamento orientada a contêineres. Esse controle é realizado através da adição de nomes de domínio a arquivos e programas, de forma que o invasor não possa roubar, adulterar dados ou causar prejuízos apenas violando a defesa do *host*. A diferença entre Zhao *et al.* e o presente trabalho é que o método utilizado por Zhao *et al.* se aplica mais à segurança em ambientes de contêineres, enquanto o trabalho atual visa verificar o isolamento de recursos computacionais entre contêineres.

De acordo com Felter *et al.* [Felter et al. 2015], contêineres possuem igual ou até melhor desempenho do que máquinas virtuais - *Virtual Machine* (VM). Felter *et al.* chegou a esta conclusão realizando experimentos utilizando - *Kernel-based Virtual Machine* (KVM) para lidar com as VMs e o Docker para gerenciar os contêineres. A conclusão de Felter *et al.* é importante para o presente trabalho, visto que o isolamento entre máquinas virtuais é bem consolidado, com os *hypervisor* separando efetivamente as instâncias. No entanto, entre contêineres, sendo uma virtualização leve, esse isolamento pode não ser efetivo. É exatamente isso que o presente trabalho visa verificar.

Em [Jiqing 2020], é verificado a relação quantitativa entre os índices de desempenho em operações de entrada e saída. Jiqing propõe um algoritmo e modelo lógico para aprimorar o isolamento do desempenho de Entrada e Saida (E/S) do sistema de contêineres. Apesar de propor uma melhoria, [Jiqing 2020] não realiza uma análise, nem faz uma verificação do nível de isolamento. Dessa forma, o presente artigo procura fazer a análise desse isolamento.

3. Ferramentas e serviços para a análise do isolamento entre contêineres

Nos seguintes parágrafos serão discutidos os *softwares* e ferramentas utilizadas para criar e monitorar ambientes em nuvem, utilizando técnicas de microsserviço e contêiner para virtualização e implementação das aplicações.

3.1. Microsserviços, Contêineres e Kubernetes

Microsserviços é uma abordagem que divide as aplicações em serviços de *software* de granularidade fina e de fraco acoplamento, baseados, principalmente, em *Service-Oriented Architecture* (SOA). Essa abordagem visa desacoplar as aplicações em várias unidades adaptadas a responsabilidades funcionais específicas. Cada microsserviço é encapsulado em seu próprio contêiner, garantindo independência, foco e escalabilidade individuais para cada serviço [Liu et al. 2020].

Contêineres e VMs possuem o mesmo objetivo: isolar uma aplicação e suas dependências em uma unidade autônoma que pode ser executada em qualquer lugar. A principal diferença entre os dois está em sua abordagem arquitetônica. Um contêiner utiliza o *kernel* do *host*, não tem um sistema operacional próprio, assim com os módulos são próprios e é composto por um conjunto de processos visíveis pela máquina *host*. Por outro lado, as VMs precisam da virtualização completa da infraestrutura e do sistema operacional [Martino et al. 2017]. Por essa característica, o isolamento de VMs é garantido pelo *hypervisor*. Mas, sendo contêiner uma virtualização leve, não se tem garantias do isolamento entre essas instâncias.

Kubernetes[3] é um sistema *open source* para automação de implantação, dimensionamento e operação de contêineres, certificado pela *Cloud Native Foundation* (CNCF)[4] e projetado para facilitar a utilização e orquestração de microsserviços. Na Figura 1 é detalhada a principal arquitetura do Kubernetes, com um nó *master* que possui um gerenciador de recursos - Escalonador; um servidor de *Application programming interface* (API) que faz a conexão entre os demais componentes no *cluster*. Em um nó do Kubernetes encontra-se os *pods*, os menores elementos dessa arquitetura [Shah and Dubaria 2019], em um *pod* podem ser implantados um ou mais contêineres que compõem os microsserviços.

3.2. Prometheus

Prometheus[5] É um conjunto de ferramentas abertas de monitoramento e alerta que coleta e armazena métricas como dados de séries temporais, com pares opcionais de *key-value* denominados *labels*. As métricas armazenadas podem ser visualizadas em painéis como Grafana[6] ou acessadas mediante APIs [Marques et al. 2024]. Na Figura 2, a arquitetura do Prometheus é destacada, onde ele obtém métricas, seja diretamente ou mediante um *gateway push* intermediário. No contexto do Kubernetes, o Prometheus é empregado para analisar o desempenho de um *cluster*, monitorando as aplicações em *pods*. Neste trabalho, o Prometheus foi usado para coletar métricas de telemetria de CPU, memória, disco e rede, possibilitando percepções sobre o comportamento das aplicações.

[3]Disponível em: https://kubernetes.io/ - acessado em: 08 de abril de 2024

[4]Disponível em: https://www.cncf.io/ - acessado em: 08 de abril de 2024

[5]Disponível em: https://prometheus.io/docs/introduction/overview/ - acessado em: 08 de abril de 2024

[6]Sistema de visualização de monitoramento. Disponível em: https://grafana.com/ - acessado em 08 de abril de 2024

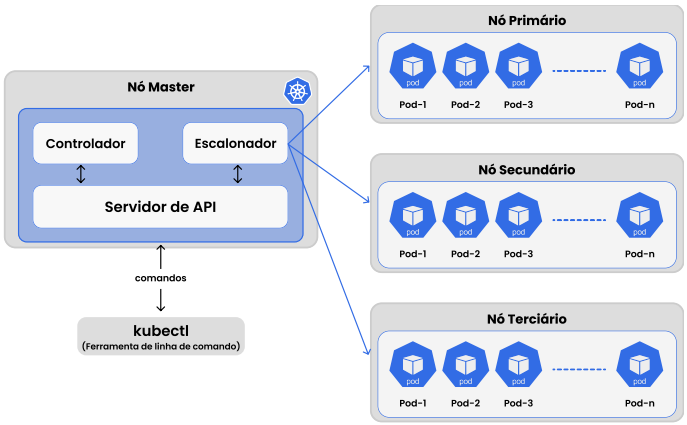


Figura 1. Arquitetura do Kubernetes Adaptado de [Perveez 2020].

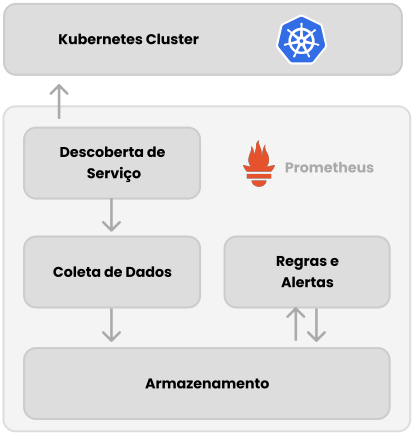
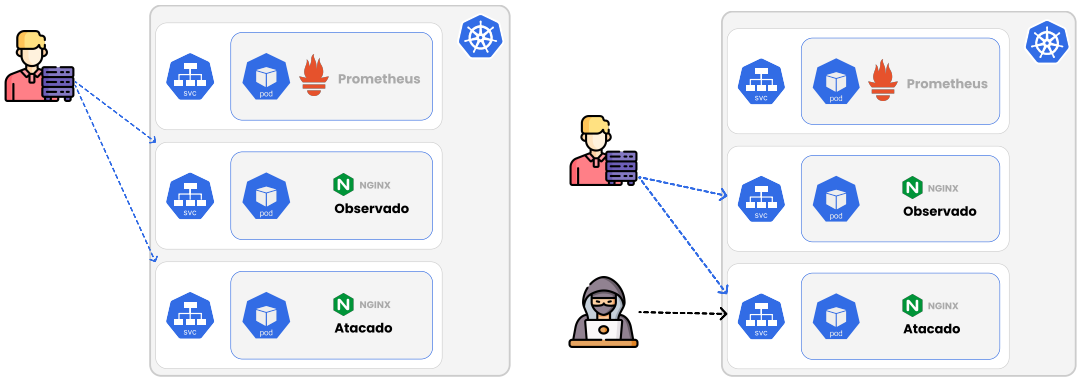


Figura 2. Arquitetura do Prometheus. Adaptada de [Prometheus 2015].

4. Cenários de experimentos para a análise do isolamento entre contêineres

Para realizar os experimentos, foi instalado o orquestrador de contêineres Kubernetes, na versão 1.27, em uma máquina com processador Intel Xeon E-2224 3.4GHz, 6GB de memória RAM, com Sistema Operacional Ubuntu Server, versão 20.04.6 LTS. Além disso, no próprio Kubernetes, foi instalado um *pod* executando o Prometheus.

Para simular o ambiente de produção, foram criados dois *Pods*, ambos configurados com um servidor *web* Nginx[7]. Como o objetivo do presente artigo é verificar o nível de isolamento entre os contêineres durante um ataque de negação de serviço, há a necessidade de instanciar esses dois *pods*. Para diferenciá-los, o *pod* que irá receber o ataque de DoS será referenciado de `"Atacado"`, enquanto o *pod* que será analisado, com o intuito de verificar se as suas métricas se alteram mesmo sem receber um ataque, será chamado de `"Observado"`. Por fim, por meio do Prometheus, as métricas foram coletadas durante os experimentos.



(a) Apenas Clientes (b) Clientes e Atacantes

Figura 3. Cenários de experimentos: apenas cliente e com clientes e atacantes.

4.1. Cenários

Nestes cenários foram utilizados computadores equipados com processador Intel Core i3-10100T, 8GB de RAM e Ubuntu 22.04 LTS e nele foram simulados 2000 clientes

[7]Disponível em: http://www.nginx.com - acessado em 08 de abril de 2024

legítimos, e cada cliente realizava uma nova requisição em um intervalo de 3 segundos, durante todo o experimento. Foi escolhida a geração de 2000 clientes por ser a capacidade máxima de atendimento do Nginx sem haver negação de serviço. Dessa forma, foram simulados clientes que utilizavam todo o *buffer* de atendimento do servidor *web*.

No Cenário 1, representado na Figura 3(a), as requisições vêm exclusivamente de clientes legítimos. O teste foi conduzido com a ferramenta Siege[8], conhecida por simular tráfego de usuários e realizar testes de desempenho. Após cada teste, o Siege fornece um relatório completo contendo diversas métricas, das quais este artigo utiliza a quantidade de clientes atendidos, ou seja, a disponibilidade, e o *Time-to-Service* (TTS), que representa o tempo médio total que o cliente requisita a página, o servidor processa e a resposta retorna ao cliente.

O cenário da Figura 3(a) mostra que os clientes legítimos gerem tráfego para o *pod*, com a intenção de avaliar o sistema de forma adequada, sem qualquer intenção maliciosa. Ele serve para verificar como o sistema ou serviço se comporta quando está sob uma carga significativa de tráfego de usuários genuínos. Os clientes estão gerando solicitações legítimas, o que permite avaliar a capacidade de resposta e desempenho do sistema em condições reais e normais de uso.

No cenário 2, apresentado na Figura 3(b), além das requisições de clientes legítimos, semelhantes ao cenário anterior, foi incluído um atacante. O ataque de DoS, especificamente SYN-Flood, é realizado utilizando a ferramenta Hping3[9], que envia uma quantidade elevada de pacotes TCP com a *flag* SYN ativada para o alvo, buscando afetar sua disponibilidade. Observa-se ainda que o ataque é direcionado apenas ao *pod* `Atacado`, não afetando o `Observado`.

Dessa forma, esse cenário verifica se havendo um ataque de negação de serviço ao `Atacado`, os clientes que estão realizando requisições ao `Observado` irão ter sua qualidade de serviço afetada. Ou seja, se um contêiner instanciado em uma mesma infraestrutura que outro contêiner que está recebendo um ataque de DoS sofre alguma interferência por estar na mesma infraestrutura. Além das métricas de qualidade de serviço, visa-se verificar se as métricas de telemetria do `Observado` também tem alterações. Constatando assim, que o nível de isolamento entre os contêineres é ou não efetivo.

5. Resultados

Com os cenários criados, foram realizados quatro experimentos em cada cenário, cada um com duração de 15 minutos. Em cada experimento foram coletadas as informações da telemetria dos *pods* `Atacado` e `Observado`, além do relatório dos clientes legítimos, oferecido pela ferramenta Siege. Para os resultados apresentados nesta seção, foi calculado a média e o desvio padrão.

No cenário 1 (Figura 3(a)), em que há apenas o tráfego de clientes, o sistema respondeu como esperado. Todos os clientes que realizaram requisições foram atendidos e com um tempo de resposta adequado para requisições realizadas na mesma rede.

No cenário 2 (Figura 3(b)), em que há o ataque de DoS ao *pod* `Atacado`, enquanto o `Observado` recebia apenas requisições de clientes, verificou-se, pelos dados da

[8]Disponível em: https://linux.die.net/man/1/siege - acessado em 08 de abril de 2024
[9]Disponível em: https://www.kali.org/tools/hping3/ - acessado em 08 de abril de 2024

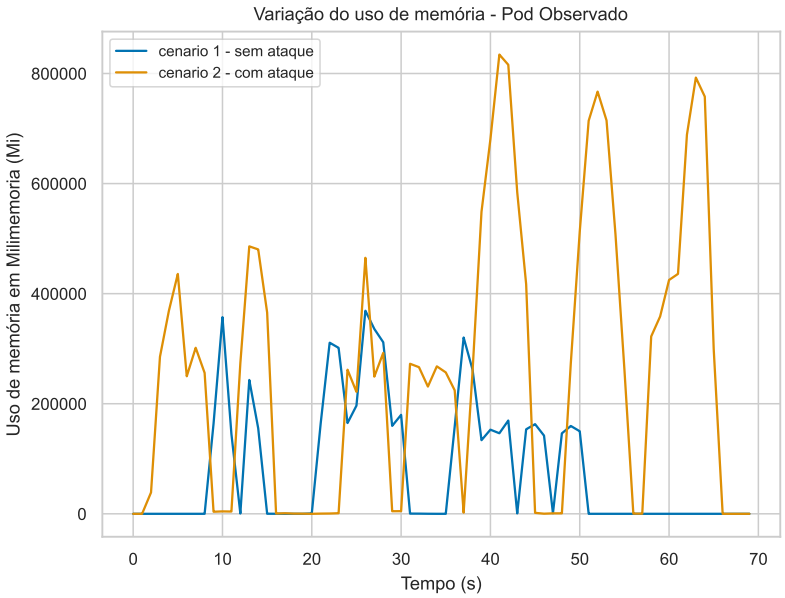


Figura 4. Uso de memória, sem e com ataque, do `Observado`.

telemetria oferecida pelo Prometheus, que o `Observado` tem algumas de suas métricas afetadas pela inserção do ataque no experimento. Na Figura 4 é apresentado os resultados da métrica de Consumo de Memória RAM do *pod* `Observado`. No eixo x, é a duração dos experimentos e no eixo y a quantidade de memória, em Milimemória (Mi)[10], utilizada pelo *pod*. Neste gráfico há duas linhas: a linha em azul são os resultados dos experimentos no cenário em que não há o ataque de negação de serviço; a segunda linha, em amarelo, são os resultados do consumo de memória durante os experimentos em que o outro *pod*, o `Atacado`, está recebendo um ataque de DoS.

Verifica-se, ainda na Figura 4, que há uma diferença no consumo de memória entre os dois cenários. No entanto, ressalta-se, que o `Observado` não está recebendo ataque, apenas recebendo o tráfego de cliente legítimo. Observa-se ainda, que em determinados momentos, o consumo de memória mais que dobra quando está acontecendo um ataque de negação de serviço a outro *pod* instanciado na mesma infraestrutura. Dessa forma, verifica-se que, relacionado a memória, não há isolamento entre os contêineres, pois o ataque de negação de serviço realizado contra o *pod* `Atacado` elevou o consumo de memória do `Observado`.

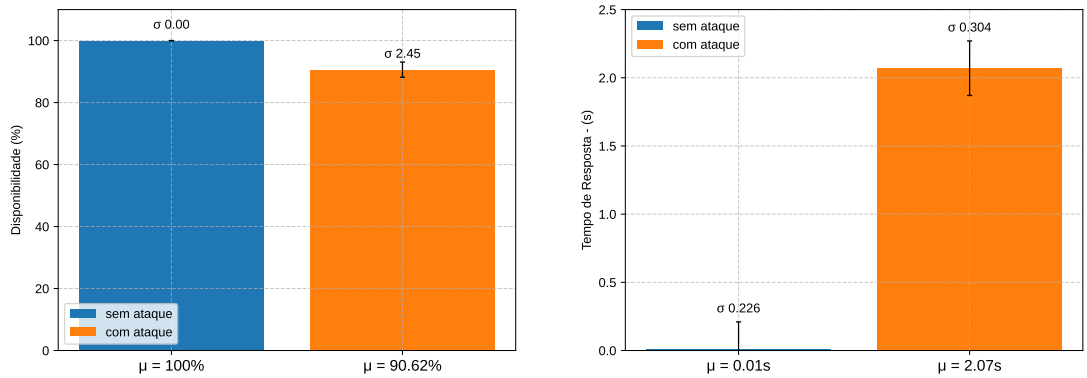
Vale salientar que nas outras métricas da telemetria, por exemplo, CPU e disco, não foi constatado mudança significativa nos resultados comparativos entre os dois cenários. Esse fato pode ser explicado pelo motivo do ataque não gerar tanto consumo nessas outras métricas, fazendo com que a administração dos recursos de *hardware*, realizada pelo Kubernetes, fosse efetiva, garantindo certo nível de isolamento.

Tratando dos resultados gerado pelos clientes, a Figura 5(a) apresenta os resultados da disponibilidade. A métrica de disponibilidade é a quantidade total de clientes que solicitaram a página *web* e obtiveram sucesso na solicitação. No gráfico é apresentado o resultado da disponibilidade para o cenário sem e com ataque, em relação aos clientes que realizaram requisições ao *pod* `Observado`. A barra em azul, a esquerda, é do cenário

[10]Disponível em: https://kubernetes.io/pt-br/docs/concepts/configuration/manage-resources-containers/#unidades-de-recursos-no-kubernetes - acessado em 27 de abril de 2024

sem ataque, enquanto a barra em vermelho, a direita, é dos experimentos com ataque de negação de serviço.

Como visto na Figura 5(a), o *pod* `Observado` no cenário 1, em que não houve ataque, apresentou 100% de disponibilidade. Isso significa que o pod estava operando sem interrupções ou problemas, garantindo o pleno funcionamento do sistema. No entanto, no cenário 2, em que há o ataque no *pod* `Atacado`, houve uma redução da disponibilidade para 90,62%, com desvio padrão de 2,446, indicando que o `Observado` foi afetado pelo ataque ao *pod* `Atacado` e, consequentemente, não pode fornecer seu serviço de forma ininterrupta. Apesar da redução de 10% na disponibilidade, se houvesse um isolamento efetivo entre os contêineres, o *pod* `Observado` deveria manter a disponibilidade.



(a) Disponibilidade. (b) Tempo de resposta.

Figura 5. Resultados de Disponibilidade e TTS, nos dois cenários, do *pod* `Observado`.

Na Figura 5(b) é apresentado os resultados da métrica Tempo de Resposta (TTS - *Time-to-Service*), em segundos. Essa métrica é o tempo total em que o cliente solicitou a página *web*, o servidor recebeu, processou o pedido e a resposta chegou até o cliente. Novamente, são apresentados dois gráficos, o da esquerda sendo o resultado do cenário sem ataque, e a barra da direita o resultado do cenário com ataque de DoS.

Nos resultados, conforme mostrado na Figura 5(b), o tempo de resposta no cenário 1, sem ataques, foi de 0,01 segundo, com um desvio padrão de 0,226, indicando eficiência e rapidez do sistema. Esse resultado era esperado, considerando que os experimentos foram conduzidos na mesma rede local. Porém, no cenário 2, com o ataque, o tempo aumentou consideravelmente para 2,07 segundos, com desvio padrão de 0,304. Esse aumento de tempo sugere que o *pod* foi negativamente impactado pelo ataque em outro contêiner, resultando em prejuízos na experiência do usuário. Assim, um usuário pode esperar, em média, 2 segundos para acessar a página solicitada, o que é considerado longo para uma rede local. Apesar de mais de 90% das requisições terem sido bem-sucedidas, a resposta demora a chegar até o cliente.

6. Conclusão e Trabalhos Futuros

Este artigo realizou experimentos para verificar o nível de isolamento entre contêineres instanciados em uma mesma infraestrutura oferecido pelo orquestrador Kubernetes. Foram criados cenários com dois *pods*, e realizados experimentos, gerando um ataque de negação de serviço a um desses *pods*. Durante o ataque, as métricas do outro contêiner foram coletadas para verificar se haveria ou não algum tipo de interferência nessas métricas, a fim de constatar se há ou não o isolamento entre os *pods*.

Ao analisar os experimentos de isolamento entre contêineres na mesma infraestrutura, constatou-se um isolamento parcial devido ao gerenciamento ineficiente de recursos, resultando em interferência e competição entre os *pods*. Isso causa consequências significativas na experiência do usuário, incluindo degradação de desempenho, tempos de resposta prolongados, aumento da latência e indisponibilidade para os clientes.

Por fim, como indicação de trabalhos futuros, propõe-se a realização de novos experimentos, visando exaurir, por meio de outros tipos de ataques, outros recursos computacionais a fim de verificar o isolamento desses outros recursos. Além disso, pretende-se verificar o aprofundamento do isolamento entre aplicações, por meio de novos experimentos buscando pontos vulneráveis e brechas que permitam acesso a contêineres distintos instanciados na mesma infraestrutura.

7. Agradecimentos

Este trabalho recebeu financiamento de bolsas de PIBIC e PET da UFC. Os autores gostariam de agradecer o financiamento da FAPESP - Projeto de Pesquisa 2018/23097-3.

Referências

Felter, W. et al. (2015). An updated performance comparison of virtual machines and linux containers. In *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 171–172. IEEE.

Jiqing, C. (2020). I/o performance optimization analysis of container on cloud platform. In *2020 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*, pages 84–86.

Liu, G. et al. (2020). Microservices: architecture, container, and challenges. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 629–635.

Marques, G. et al. (2024). Proactive resource management for cloud of services environments. *Future Generation Computer Systems*, 150:90–102.

Martino, B. et al. (2017). Lxc and dockers: Migrating osa software on linux containers. page 073.

Pelloso, M. et al. (2018). Um sistema autoadaptável para predição de ataques ddos fundado na teoria da metaestabilidade. In *Anais do XXXVI SBRC*, pages 726–739, Porto Alegre, RS, Brasil. SBC.

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: 27-03-2024.

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.

Zhao, B. et al. (2021). Research on container-oriented isolation control technology. In *Journal of Physics: Conference Series*, volume 1871, page 012016. IOP Publishing.