DISSERTAÇÃO DE MESTRADO PROFISSIONAL

SARIK: Uma proposta de *framework* para o aprimoramento da segurança em Kubernetes por meio de políticas de rede

Jonathan Gomes Pereira dos Santos

Programa de Pós-Graduação Profissional em Engenharia Elétrica
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO PROFISSIONAL

SARIK: Uma proposta de *framework* para o aprimoramento da segurança em Kubernetes por meio de políticas de rede

Jonathan Gomes Pereira dos Santos

Dissertação de Mestrado Profissional submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica

Banca Examinadora

Prof. Vinícius Pereira Gonçalves, Ph.D, FT/UnB _____
Orientador

Edna Dias Canedo, Ph.D, CIC/UnB _____
Examinador interno

Lourenço Alves Pereira Jr, Ph.D, ITA _____
Examinador externo

Prof. Geraldo Pereira Rocha Filho, Ph.D, FT/UnB _____
Suplente

FICHA CATALOGRÁFICA

SANTOS, JONATHAN GOMES PEREISA DOS
SARIK: Uma proposta de *framework* para o aprimoramento da segurança em Kubernetes por meio de políticas de rede [Distrito Federal] 2024.
PPEE.MP.062, xvi, 107 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2024).
Dissertação de Mestrado Profissional - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Kubernetes	2. Políticas de rede
3. Framework SARIK	4. Segurança
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

SANTOS, J.G.P. (2024). *SARIK: Uma proposta de framework para o aprimoramento da segurança em Kubernetes por meio de políticas de rede*. Dissertação de Mestrado Profissional, Publicação PPEE.MP.062, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 107 p.

CESSÃO DE DIREITOS

Jonathan Gomes Pereira dos Santos
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

DEDICATÓRIA

Em primeiro lugar, àquele que é a fonte de toda a sabedoria e cuja presença foi constante mesmo nos momentos de silêncio e dúvida, expresso minha profunda gratidão. A Deus, que iluminou meu caminho com fé e esperança, dedico o início destas palavras e elevo meus pensamentos e meu mais sincero agradecimento a ti, oh, senhor Deus. Tu és a fonte inesgotável de luz e inspiração, que me guiou em cada passo neste caminho, não somente durante os momentos de clareza, mas também através das sombras da dúvida, permitindo-me enxergar o propósito maior por trás de cada desafio.

Em seguida, com o coração transbordante de amor e reconhecimento, dedico este trabalho à minha mãe, Maria Elizabete Gomes Pereira, e a toda minha família: cujo apoio incondicional foi o alicerce que sustentou minha jornada. À minha mãe, a primeira professora que me ensinou a valorizar o conhecimento e a buscar a excelência, pilar de força, refúgio seguro, e exemplo de dedicação, meu eterno obrigado. À minha família, que celebrou cada pequena vitória e me encorajou frente a cada desafio, minha eterna admiração. Com o apoio silencioso, mas sempre presente de vocês, as páginas desta dissertação não teriam a mesma essência nem a mesma vida. São vocês que dão verdadeiro significado à minha caminhada.

À minha companheira de vida, Márcia Gomes da Silva, companheira de todas as horas, que dividiu comigo a carga dos dias exaustivos e na alegria dos triunfos, dedico não só este trabalho, mas cada conquista que a vida nos reserva. Minha esposa amada: sua força e seu amor são os ventos que impulsionam meus sonhos a navegarem por mares nunca antes explorados. Cada momento de apoio e confiança compartilhados com você foi vital para a realização deste sonho.

Aos meus colegas de trabalho e companheiros de profissão, que estiveram ao meu lado nas etapas mais críticas deste projeto: cada palavra de incentivo, cada sessão de revisão textual, cada análise estatística e cada gesto de ajuda foram como tijolos fundamentais na construção deste edifício acadêmico. Vocês não somente compartilharam sua expertise, mas também a humanidade e a camaradagem que tanto valorizo.

Esta dedicatória estende-se também a todos que, incansavelmente, perseguem suas aspirações, sem jamais ceder ao desânimo. Que este trabalho possa refletir a mensagem de que o sucesso é um caminho construído pelo esforço contínuo e pela crença incansável em nossas capacidades. Dedico, este estudo, àqueles que, como eu, nunca deixaram de acreditar na força dos seus sonhos. A todos que persistem com tenacidade, que enfrentam os obstáculos com coragem e que transformam o ato de nunca desistir em um exemplo de determinação inquebrável.

E para encerrar, deixo uma reflexão inspiradora: o conhecimento é o bem mais democrático do mundo, é a única riqueza que, quanto mais compartilhada, mais cresce e se valoriza. Que as páginas a seguir possam não somente agregar ao campo do saber, mas motivar outros a deixarem sua marca no mundo. Pois cada nova ideia, cada descoberta, é um tijolo a mais na construção da imensa edificação chamada humanidade.

Com imensa gratidão,

Jonathan

AGRADECIMENTOS

Como bem lembrou meu coorientador, Dr. Geraldo Pereira Rocha Filho, a jornada de um mestrando vai muito além dos anos formalmente dedicados ao mestrado. No meu caso, essa trajetória estendeu-se por mais de três anos. Desde o término da minha graduação, passando pelas especializações, o desejo de ingressar e concluir o mestrado estava sempre presente e vivo em minha mente. Nesse percurso, acumulei não apenas conhecimento técnico, mas também preciosas lições de vida.

Essa etapa foi marcada por um enriquecimento pessoal e profundo, onde cada experiência contribuiu para aprimorar minhas habilidades em diversas áreas: técnicas, escrita, expressão oral e, especialmente, no aspecto humano. A cada etapa, contei com o apoio de pessoas excepcionais que, generosamente, compartilharam seu tempo, sabedoria e encorajamento.

A todas essas pessoas maravilhosas, dedico o meu mais sincero obrigado. Vocês foram parte essencial desta viagem e, sem vocês, a paisagem não teria sido tão rica nem a chegada tão gratificante. Para todas essas pessoas incríveis, quero registrar meus agradecimentos.

Em particular, minhas palavras de gratidão convergem para uma pessoa cujo nome ressoa como sinônimo de dedicação e amor: minha mãe, Maria Elizabete Gomes Pereira. A senhora, que investiu anos de trabalho árduo e inabalável fé em mim, proporcionou-me a mais valiosa das heranças – a educação. Sua incansável perseverança em acreditar no meu potencial foi a chama que iluminou os momentos mais obscuros desta trajetória. Mãe, a senhora me ensinou que cada desafio é uma oportunidade de crescimento e que o verdadeiro sucesso reside na força de nunca desistir. Dedico cada página deste trabalho à senhora, que é a verdadeira heroína desta conquista. A minha vitória é também a sua, e espero honrar o legado de resiliência e esperança que você tão generosamente me concedeu.

Agradeço à minha esposa, Márcia Gomes da Silva, dedico não apenas este trecho, mas a essência de cada página que compõe este trabalho. Seu sorriso e gargalhadas foram o antídoto para as adversidades, com um gesto, um olhar, você tinha o dom de reacender a chama da esperança em meu coração. Minha companheira, minha confidente, meu grande amor, as palavras são insuficientes para expressar a totalidade da minha gratidão. A cada passo, a cada linha escrita, a cada desafio superado, lá estava você. Que este reconhecimento seja um eco do meu apreço imensurável e que a vida nos permita compartilhar juntos muitas outras conquistas. Com você, todo sonho parece possível. Com você, cada momento se torna um tesouro. Obrigado por ser a companhia mais preciosa na viagem mais significativa da minha vida.

Agradeço imensamente aos meus orientadores, Dr. Vinícius Pereira Gonçalves e Dr. Geraldo Pereira Rocha Filho, cuja confiança e oportunidades oferecidas foram cruciais para o desabrochar do meu potencial. As linhas deste trabalho são marcadas pelo conhecimento e sabedoria que me transmitiram. Guardo cada momento de nossa colaboração como um tempo de enriquecimento mútuo e troca inestimável, esperando sinceramente que nossa jornada conjunta tenha sido tão valiosa para vocês quanto foi para mim.

Em especial, expresso minha profunda gratidão ao Dr. Vinícius Pereira Gonçalves, não apenas como orientador, mas como farol de esperança nos momentos turbulentos desta jornada acadêmica. Pelas suas orientações precisas, dicas esclarecedoras e *feedbacks* construtivos foram pilares fundamentais no meu desenvolvimento como pesquisador. Quando me vi às margens de uma pseudo desistência do mestrado, suas palavras de apoio e confiança foram o alento necessário para reconhecer que cada fracasso é um prelúdio

do sucesso. Lembro-me de sua sabedoria ao dizer que, mesmo nas etapas mais difíceis da pesquisa, a luz pode emergir de uma simples mudança de perspectiva. Além do suporte emocional, sou grato pelas risadas compartilhadas nas aulas de algoritmos e estruturas de dados e pelos *feedbacks* construtivos. O conhecimento que o senhor me transmitiu é um tesouro que levarei por toda a vida profissional e pessoal.

Minha gratidão ao Dr. Geraldo Pereira Rocha Filho é imensurável, pois suas orientações precisas, dicas esclarecedoras e *feedbacks* construtivos foram pilares fundamentais no meu desenvolvimento como pesquisador. Aprendi sob sua tutela a essência da pesquisa científica, a apurar o rigor técnico e a refinar o olhar analítico, especialmente na apresentação de dados. A confiança que depositou em mim e a qualidade exemplar de sua orientação, inclusive nas revisões meticulosas dos artigos, foram essenciais para a concretização deste trabalho. Reconheço que o processo educativo é permeado por altos e baixos, e por isso peço desculpas por quaisquer falhas da minha parte. Sua postura ética e profissional diante de tais momentos foi exemplar, sempre guiando-me de volta ao caminho com uma "advertência" necessária e elegante quando preciso. Esses "esporros" foram, em verdade, lições valiosas que me incentivaram a me reerguer e seguir adiante. Por tudo isso, minha eterna gratidão.

Expresso minha gratidão ao Msc. Elivaldo Ribeiro de Santana, um mentor cuja paciência e tolerância moldaram o cientista em mim, ensinando-me pacientemente a validar os dados estatísticos e a extrair significado deles. Sua energia incansável na interpretação dos dados e na orientação para uma apresentação impactante foi indispensável. Sou grato pelas revisões, que foram particularmente cruciais no desenvolvimento do meu primeiro e segundo artigos. Sua atenção aos detalhes e o rigor acadêmico não apenas aprimoraram significativamente o meu trabalho, mas também deixaram uma marca constante em minha formação. Obrigado não só pela parceria acadêmica, mas também pela disponibilidade que sempre mostrou. Com certeza, iremos à Vila Planalto saborear aquele traíra, celebrando não só o fim desta etapa, mas a continuidade de nossa colaboração e amizade.

Agradeço ao Dr. Georges Daniel Amvame Nze, no momento crítico desta pesquisa, quando me encontrava perdido e sem direção, sua paciência notável e impressionante habilidade de pesquisa foram assertivas. Em questão de minutos, você foi capaz de realinhar meu foco e me guiar para o caminho correto em relação às métricas essenciais deste projeto.

Agradeço ao Luiz Eduardo Rodrigues Lima, pela sua contribuição criativa no *design* das páginas do projeto e pela ajuda nas pesquisas sobre o tema deste trabalho. Sua assistência técnica e sua amizade foram essenciais para o sucesso deste projeto.

Estendo meus sinceros agradecimentos a Tainá N. S. Moura, cuja habilidade na revisão e correção textual enriqueceu significativamente nossos artigos. Sua dedicação não passou despercebida e é parte integrante do que esses trabalhos são hoje.

Um especial reconhecimento a Judy Pirangi, cujo talento em tradução e conhecimento dos melhores locais para revisões foram fundamentais. Sua ajuda garantiu que os artigos alcançassem um nível de qualidade e clareza significativa.

Agradeço a Juliana Cristina Sampaio Silva, cuja perícia na revisão textual do pré-projeto e os *feedbacks* construtivos ao longo dos anos foram cruciais. Sua assistência aprimorou não apenas os textos em si, mas também minha habilidade na escrita de forma geral.

Minha sincera gratidão à equipe de gestão da Direção da Faculdade de Medicina, cuja colaboração foi essencial. O apoio incondicional e a adaptabilidade dos horários forneceram o ambiente necessário para que a pesquisa fosse conduzida com a devida atenção e rigor. Além disso, estendo minha gratidão ao Diretor da Faculdade de Medicina, Dr. Gustavo Adolfo Sierra Romero. Seus ensinamentos foram inestimáveis e sua postura íntegra e profissional não apenas conduzem nossa instituição, mas também serviram como exemplo inspirador durante minha trajetória acadêmica. Sua liderança foi e continua sendo um modelo de excelência que permeia toda Faculdade.

Minha gratidão ao Dr. Hervaldo Sampaio Carvalho. Sua carta de indicação e sua orientação ao longo da última década foi o alicerce do meu crescimento acadêmico e profissional. Agradeço-lhe pelas lições valiosas e pelo amadurecimento que sua tutela me proporcionou. Suas aulas de estatística, em particular, foram cruciais para o desenvolvimento do meu segundo artigo.

A minha eterna gratidão é dedicada à memória do ex-diretor da Faculdade de Medicina, Dr. Paulo César de Jesus. Sua inteligência, ensino e vitalidade, bem como sua notável capacidade de liderança, permanecem como faróis de excelência profissional e ética. Sou grato por ele ter me incentivado a perseguir o crescimento acadêmico e por todos os conhecimentos compartilhados. Levarei sempre comigo a sua valiosa máxima: "Seja uma mente de rato pensante, em vez de uma cauda de leão". Por tudo isso e pelo legado que deixou, meu obrigado mais sincero e profundo.

Minha gratidão se estende à secretaria do PPEE, com um agradecimento especial a Tayná Gabriela Araújo Albuquerque. Sua conduta exemplar e o atendimento de qualidade foram peças-chave para a adequada condução desta pesquisa. A eficiência com que organizou a agenda, possibilitando o encontro rápido com Georges, foi um apoio inestimável. Obrigado por sua dedicação e por facilitar etapas deste processo.

A todos os membros do corpo docente do PPEE, minha gratidão é imensa. Faço um agradecimento especial aos professores Dr. Rafael Timóteo de Sousa Júnior, Dr. Rafael Rabelo Nunes, Dr. William Ferreira Giozza, Dr. Fábio Lúcio Lopes de Mendonça, Dr. Demétrio Antônio da Silva Filho e Dr.ª Edna Dias Canedo. Cada ensinamento que recebi de vocês foi um tijolo na construção da excelência deste trabalho. Agradeço profundamente pela dedicação, conhecimento compartilhado e pelo impacto significativo que tiveram em minha formação acadêmica. Muito obrigado.

Minha gratidão aos professores Dr. Rodolfo I. Meneguette, Dr. Rodrigo Bonacin e D.Sc Gustavo Pessi é profunda e sincera. O tempo e o esforço que dedicaram à revisão do segundo artigo foram fundamentais, e suas contribuições se mostraram extremamente valiosas. As perspectivas e conhecimentos que compartilharam não só enriqueceram a qualidade do trabalho, mas também ampliaram minha compreensão dos temas abordados. Cada sugestão e correção adicionou uma camada de excelência ao produto final. Por isso, agradeço imensamente por sua generosidade intelectual e apoio.

Minha gratidão se estende a todos os profissionais e professores da Universidade de Brasília (UnB), cuja postura ética e generosidade foram fundamentais em minha trajetória. Agradeço especialmente aos professores do CIC, aos parceiros da STI e do DIMEQ, e, claro, aos colegas da Faculdade de Tecnologia, Jackson e Wesley. Para todos os amigos e colegas que compartilharam dessa jornada, citados ou não, ofereço meu muito obrigado. Cada um de vocês teve um papel especial na construção do meu percurso acadêmico e profissional, e eu guardo um sentimento de gratidão sincera por cada contribuição.

RESUMO

A evolução das tecnologias de contêineres e orquestração, especialmente o Kubernetes, trouxe inúmeras vantagens para a implantação de aplicativos em ambientes distribuídos, mas também apresentou desafios significativos em termos de segurança. Esta dissertação introduz o *framework* SARIK (Segurança Automática de Regras de *Iptables* no Kubernetes), uma solução projetada para aprimorar a segurança em ambientes Kubernetes. O SARIK aborda as vulnerabilidades associadas ao tráfego de rede em *clusters* Kubernetes, implementando políticas de rede de forma eficiente e automatizada. Este estudo não apenas explora as limitações das abordagens tradicionais de segurança em Kubernetes, mas também oferece uma análise empírica da eficácia do SARIK, utilizando métricas como latência, taxa de resposta e taxa de transmissão. Os resultados confirmam que o SARIK não compromete o desempenho das aplicações enquanto fortalece a segurança. Além disso, são discutidas as implicações práticas, desafios futuros e melhores práticas para a segurança em ambientes Kubernetes. O trabalho também destaca a influência da escolha do sistema operacional e da configuração do *cluster* na eficácia das políticas de segurança. Em resumo, o SARIK representa um avanço significativo na segurança de ambientes Kubernetes, oferecendo uma abordagem prática e automatizada para mitigar riscos e aprimorar a proteção de *clusters* em um cenário tecnológico cada vez mais complexo e vulnerável.

ABSTRACT

The evolution of container and orchestration technologies, especially Kubernetes, has brought numerous advantages to deploying applications in distributed environments, but it has also presented significant challenges in terms of security. This dissertation introduces the SARIK framework (Security Automated Rules for Iptables in Kubernetes), a solution designed to enhance security in Kubernetes environments. SARIK addresses vulnerabilities associated with network traffic in Kubernetes clusters by implementing network policies efficiently and automatically. This study not only explores the limitations of traditional security approaches in Kubernetes but also empirically analyzes the effectiveness of SARIK, using metrics such as latency, response rate, and transmission rate. The results confirm that SARIK does not compromise application performance while strengthening security. In addition, we discuss practical implications, future challenges, and best practices for security in Kubernetes environments. The work also highlights the influence of the choice of the operating system and cluster configuration on the effectiveness of security policies. In summary, SARIK represents a significant advancement in Kubernetes security, offering a practical and automated approach to mitigating risks and enhancing cluster protection in an increasingly complex and vulnerable technological landscape.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

ACM	*Association for Computing Machinery* (Associação para Maquinário de Computação)
API	*Application Programming Interface* (Interface de Programação de Aplicações)
AWS	*Amazon Web Services*
CI/CD	*Continuous Integration/Continuous Delivery* (Integração Contínua/Entrega Contínua)
CNI	*Container Network Interface* (Interface de Rede dos Contêineres)
CPU	*Central Processing Unit* (Unidade Central de Processamento)
DevOps	*Development and Operations* (Desenvolvimento e Operações)
GDPR	*General Data Protection Regulation* (Regulamento Geral sobre a Proteção de Dados)
HA	*High Availability* (Alta Disponibilidade)
HIPAA	*Health Insurance Portability and Accountability Act* (Lei de Portabilidade e Responsabilidade do Seguro Saúde)
IEEE	*Institute of Electrical and Electronics Engineers* (Instituto de Engenheiros Eletricistas e Eletrônicos)
IaaS	*Infrastructure as a Service* (Infraestrutura como Serviço)
ICMP	*Internet Control Message Protocol* (Protocolo de Mensagens de Controle da Internet)
IDS	*Intrusion Detection System* (Sistema de Detecção de Intrusão)
IoT	*Internet of Things* (Internet das Coisas)
KPI	*Key Performance Indicator* (Indicador-chave de Desempenho)
PCI DSS	*Payment Card Industry Data Security Standard* (Padrão de Segurança de Dados para a Indústria de Cartões de Pagamento)
POD	Unidade básica de computação em Kubernetes
SARIK	Segurança Automática de Regras de *Iptables* no Kubernetes
TCP	*Transmission Control Protocol* (Protocolo de Controle de Transmissão)
TI	Tecnologia da Informação
UDP	*User Datagram Protocol* (Protocolo de Datagrama de Usuário)
VXLAN	*Virtual Extensible LAN* (Rede de Área Local Virtual Extensível)

1 INTRODUÇÃO

À medida que a computação em nuvem e a virtualização se tornaram a espinha dorsal das infraestruturas modernas de TI [1], a orquestração de contêineres emergiu como uma abordagem crucial para gerenciar, escalar e implantar aplicativos de forma eficiente em ambientes distribuídos [2]. Nesse cenário, o Kubernetes rapidamente conquistou destaque como a plataforma de orquestração mais popular e amplamente adotada [3].

No entanto, o sucesso notável do Kubernetes também trouxe à tona desafios complexos relacionados à segurança. À medida que as implantações de Kubernetes crescem em escopo e importância, garantir a integridade, a confidencialidade e a disponibilidade dos aplicativos em execução tornou-se uma tarefa crítica e versátil. A arquitetura de microsserviços, embora traga flexibilidade e escalabilidade [4], também introduz novos vetores de ameaça e superfícies de ataque e junto com a crise global de saúde causada pela pandemia COVID-19 fez aumentar o uso de serviços em nuvem. Como resultado, a procura aumentou significativamente por parte das três maiores empresas fornecedoras de tecnologia – Amazon AWS, Microsoft *Azure* e Google *Cloud*. Nos primeiros quatro meses de 2020, os investimentos em computação em nuvem cresceram 37% [5] à medida que os serviços se expandiam. Consequentemente, cresceu também o número de ataques cibernéticos nesses ambientes.

Nos relatórios de 2018 e 2019 publicados pelas empresas de segurança Kromtech [6] e Snyk [7], foram destacadas as vulnerabilidades predominantes encontradas em imagens hospedadas no Docker Hub. Isso é exemplificado na Figura 1.1 do relatório intitulado "*Shifting Docker Security Left*" da Snyk. A análise da Kromtech, que versa sobre o "*Security Center in Cryptojacking Invades Cloud: How Modern Containerization Trends Are Exploited by Attackers*", revelou a disseminação de 17 imagens maliciosas no repositório do Docker Hub, projetadas para a mineração de criptomoedas.

Essas descobertas ressaltam a crescente preocupação relacionada à segurança em ambientes de orquestração, como o Kubernetes. Os autores Karn et al. [8] destaca que, com o aumento do uso de contêineres em computação em nuvem, os riscos de segurança também evoluíram, incluindo um dos tipos mais sérios de *malware* recentemente identificados: o de mineração de criptomoedas. Esse *malware* sequestra recursos do servidor para mineração e pode ser iniciado por um executável oculto durante a implantação e execução de um Pod (unidade básica de computação em Kubernetes), rodando em segundo plano. Portanto, quando mal configurados ou acessíveis com facilidade, ambientes como o Kubernetes podem expor sistemas conteinerizados a potenciais ameaças, incluindo a inserção de *scripts* maliciosos em imagens. Esses *scripts* permitem que os criminosos utilizem imagens comprometidas para mineração de criptomoedas, prejudicando a integridade e a segurança dos recursos da plataforma.

À medida que mais organizações adotam arquiteturas de contêineres e migram para ambientes de orquestração, como o Kubernetes, a segurança se torna uma prioridade crítica. Vulnerabilidades em contêineres podem levar a violações de dados e interrupções de serviços. Em ambientes Kubernetes, essas preocupações se multiplicam devido à dinâmica de implantação e escala de contêineres. Este trabalho de pesquisa aborda esses desafios de segurança, propondo estratégias para aplicar políticas de rede em

ambientes Kubernetes.

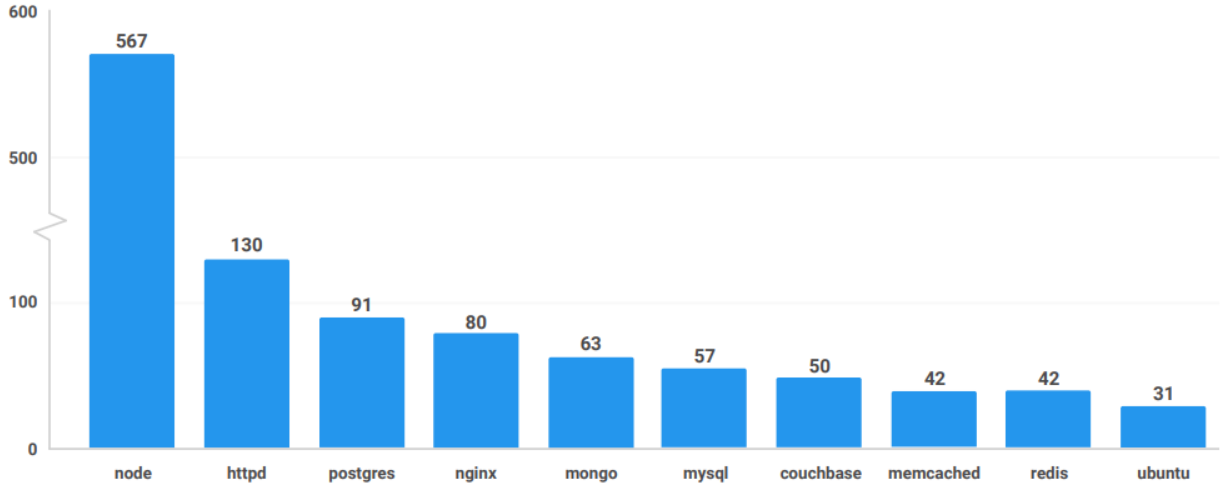

Figura 1.1: Gráfico das vulnerabilidades do relatório da Snyk de 2019
Fonte: Relatório Snyk, 2019.

Este cenário evolutivo levanta uma questão fundamental: como podemos aprimorar a segurança em Kubernetes de maneira eficaz, sem comprometer a agilidade e a flexibilidade que tornam essa plataforma tão atraente? Esta dissertação visa abordar essa pergunta, apresentando o *framework* SARIK, que é um acrônimo para "**S**egurança **A**utomática de **R**egras de ***I**ptables* no **K**ubernetes", foi originalmente concebido para reforçar a segurança dos contêineres no ambiente Kubernetes. Esse objetivo era alcançado através da aplicação de regras de bloqueio diretamente nos Pods, o que, por sua vez, exigia que esses Pods fossem criados com a *flag* privilegiada para executar essa ação. No entanto, essa abordagem se restringia à implementação de regras exclusivamente na camada da aplicação.

No decorrer desta pesquisa, aprimoramos a abordagem do SARIK, propondo que as regras de bloqueio sejam gerenciadas nas camadas de rede e transporte, por meio do kube-proxy. Essa evolução elimina a necessidade de criar Pods com a *flag privileged*, proporcionando uma abordagem mais segura e holística para a configuração de políticas de rede no ambiente do Kubernetes. Com essa melhoria, esta pesquisa oferece uma solução que não apenas amplie a segurança, mas também simplifique a implementação de políticas de rede em *clusters* Kubernetes.

Neste contexto, será explorado os desafios enfrentados pelas abordagens tradicionais de segurança em Kubernetes e demonstrado como o SARIK oferece uma alternativa inovadora. Ao alavancar a automação e a orquestração nativas do Kubernetes, o SARIK simplifica a implementação de políticas de rede granulares, controlando o fluxo de tráfego entre Pods de maneira eficiente e confiável.

Este estudo não apenas descreve a arquitetura e a implementação do SARIK, mas também investiga seu impacto prático por meio de experimentos rigorosos em vários cenários. Os resultados destacam como o SARIK aprimora a segurança em Kubernetes, ao mesmo tempo em que preserva a agilidade operacional.

Ao concluir esta dissertação, espera-se oferecer *insights* valiosos e práticos sobre como proteger eficazmente ambientes Kubernetes em um mundo cada vez mais interconectado e repleto de ameaças. O SARIK representa um passo significativo em direção a um futuro onde a segurança e a inovação podem coexistir harmoniosamente em ambientes de nuvem distribuída.

1.1 CONTEXTO, MOTIVAÇÃO E JUSTIFICATIVA

No contexto atual da revolução digital, onde a computação em nuvem e a virtualização desempenham papéis de destaque, o Kubernetes emergiu como a principal plataforma de orquestração de contêineres. Sua ascendência meteórica é resultado direto de sua eficiência e agilidade no gerenciamento de aplicativos escaláveis. O Kubernetes oferece uma base sólida para a criação e implantação de aplicativos em ambientes de contêineres, simplificando o gerenciamento de recursos de TI de forma inovadora. No entanto, essa transformação tecnológica não ocorre sem desafios, e a segurança, neste novo paradigma, assume um papel de destaque [9].

A crescente adoção do Kubernetes é impulsionada pelas vantagens evidentes que oferece: escalabilidade, eficiência operacional e portabilidade de aplicativos. Essa adoção em larga escala não é uma surpresa, dado o aumento da demanda por ambientes de desenvolvimento ágeis e escaláveis. No entanto, essa escalada de uso também trouxe consigo preocupações substanciais de segurança. A crescente complexidade dos ambientes Kubernetes, aliada à interconexão contínua de serviços e aplicativos, cria um ambiente propício para ameaças cibernéticas. À medida que a infraestrutura em nuvem se expande, novos vetores de ameaça emergem. Portanto, é crucial a necessidade de soluções que aprimorem a segurança em ambientes Kubernetes [10].

Com a crescente proliferação de dispositivos IoT (*Internet of Things*) e sua integração em uma ampla gama de setores, desde saúde até automação industrial, tem gerado um ambiente complexo e dinâmico de redes e sistemas interconectados. Nesse contexto, a segurança desses dispositivos e das redes que os sustentam se torna uma preocupação fundamental.

O Kubernetes emergiu como uma plataforma líder para a orquestração de contêineres, proporcionando escalabilidade e gerenciamento eficiente de aplicativos em ambientes de nuvem e infraestruturas locais. No entanto, sua configuração padrão nem sempre contempla as complexidades de segurança inerentes aos dispositivos IoT.

Esta dissertação aborda a necessidade iminente de aprimorar a segurança de dispositivos IoT em *clusters* Kubernetes, oferecendo um aprofundamento significativo nas políticas de rede e na aplicação do *framework* SARIK. A justificativa para este estudo se apoia em diversas considerações:

- Vulnerabilidades emergentes: À medida que a IoT se expande, novas vulnerabilidades surgem constantemente. É imperativo explorar abordagens avançadas para aprimorar a segurança, especialmente em ambientes Kubernetes, onde os dispositivos IoT frequentemente residem.
- Carência de soluções específicas: Até o momento, a pesquisa existente não ofereceu soluções abrangentes e especializadas para a segurança de dispositivos IoT em *clusters* Kubernetes. Isso deixa um vácuo que este estudo visa preencher.
- Necessidade de políticas de rede efetivas: A configuração de políticas de rede adequadas é um componente crítico da segurança em ambientes Kubernetes. A pesquisa aqui apresentada se concentra em automatizar esse processo, tornando-o mais eficaz e acessível.
- Contribuição para a comunidade científica: Este estudo oferece uma contribuição original para a

comunidade acadêmica e profissional, fornecendo uma estrutura sólida para aprimorar a segurança dos dispositivos IoT em ambientes Kubernetes.

Portanto, a justificativa para esta dissertação reside na urgente necessidade de aprimorar a segurança dos dispositivos IoT em *clusters* Kubernetes, preenchendo uma lacuna significativa na pesquisa e fornecendo soluções eficazes para mitigar as ameaças emergentes. A pesquisa conduzida neste estudo oferece uma abordagem significativa e valiosa para melhorar a segurança em um cenário de IoT em constante evolução.

1.2 QUESTÕES DE PESQUISA

A pesquisa em questão visa abordar um conjunto de questões cruciais que se alinham à complexidade inerente à segurança em ambientes Kubernetes, oferecendo hipóteses que orientarão a análise e as conclusões desta dissertação.

Questão Central:

- Como podemos garantir a segurança em ambientes Kubernetes de maneira eficaz, levando em consideração a natureza dinâmica e altamente distribuída dessa plataforma? Esta questão central identifica o principal desafio abordado nesta dissertação, que envolve a busca por soluções que assegurem a implantação segura de aplicativos em Kubernetes, mantendo sua agilidade, escalabilidade e flexibilidade inerentes.

Questões Específicas:

1. Como o *framework* SARIK pode ser desenvolvido e utilizado para automatizar a criação de políticas de rede em *clusters* Kubernetes, aprimorando a segurança nesse ambiente dinâmico? Esta questão busca investigar a eficácia e a viabilidade do SARIK como uma solução significativa para aprimorar a segurança em Kubernetes.
2. De que maneira a inserção dinâmica de regras de bloqueio pelo SARIK na interface de saída com portas e protocolos previamente mapeados pode prevenir ataques dentro do Pod e para fora do nó do *cluster*, contribuindo para um ambiente mais seguro? Esta questão avalia a importância da inserção dinâmica de regras de bloqueio na segurança dos dispositivos IoT em ambientes Kubernetes.

Hipóteses:

1. O desenvolvimento e a aplicação do *framework* SARIK resultarão na automatização eficaz da criação de políticas de rede em *clusters* Kubernetes, aprimorando a segurança em ambientes dinâmicos.
2. A inserção dinâmica de regras de bloqueio pelo *framework* SARIK, com base em portas e protocolos previamente mapeados, desempenhará um papel fundamental na prevenção de ataques dentro do Pod e para fora do nó do *cluster*, aprimorando a segurança de dispositivos IoT em Kubernetes.

1.3 OBJETIVOS E ABORDAGEM DE PESQUISA

O objetivo principal desta dissertação é propor e desenvolver um *framework* aberto e escalável para o aprimoramento da segurança em ambientes Kubernetes, fazendo uso de políticas de rede e controle de acesso. Isso será alcançado através dos seguintes objetivos específicos:

1. Realizar uma revisão aprofundada da literatura sobre as melhores práticas de segurança em ambientes Kubernetes, identificando lacunas e desafios existentes.
2. Projetar e implementar o *framework* SARIK, que visa simplificar a aplicação de políticas de rede e controle de acesso em *clusters* Kubernetes.
3. Realizar experimentos e estudos de caso para avaliar a eficácia do *framework* SARIK em ambiente controlado.
4. Analisar e discutir os resultados obtidos, identificando as implicações práticas e teóricas das políticas de rede implementadas e do controle de acesso no contexto do Kubernetes.
5. Propor recomendações e diretrizes para a implementação bem-sucedida de políticas de segurança em ambientes Kubernetes, com base nas lições aprendidas durante o desenvolvimento e avaliação do *framework* SARIK.

A abordagem de pesquisa adotada nesta dissertação será fundamentada em métodos científicos e em uma abordagem prática orientada por casos reais. A pesquisa será conduzida em várias etapas:

- Revisão Bibliográfica: Realização de uma revisão exaustiva da literatura, abrangendo as melhores práticas de segurança em ambientes Kubernetes, ferramentas de segurança existentes e desafios comuns enfrentados pelos profissionais de segurança.
- Desenvolvimento do *framework* SARIK: Projeto e implementação do *framework* SARIK, que compreende a criação de módulos de política de rede, controle de acesso e integração com a infraestrutura Kubernetes.
- Experimentação e Avaliação: Condução de experimentos em ambiente controlado, aplicando o *framework* SARIK para avaliar sua eficácia em termos de aprimoramento da segurança, desempenho e escalabilidade.
- Análise de Resultados: Análise crítica dos resultados obtidos nas experimentações, incluindo discussões sobre as métricas de segurança, desempenho e eficiência.
- Propostas e Recomendações: Com base nas conclusões da pesquisa, elaboração de propostas e recomendações para aprimorar a segurança em ambientes Kubernetes, bem como orientações práticas para sua implementação.

Essa abordagem de pesquisa integrará teoria e prática, contribuindo para o avanço do conhecimento em segurança em Kubernetes e fornecendo *insights* práticos para profissionais de segurança e administradores de sistemas que operam em ambientes Kubernetes.

1.4 PUBLICAÇÕES DESTA PESQUISA

As pesquisas decorrentes deste estudo resultaram em contribuições para a comunidade acadêmica e profissional. Dessa forma, elecamos as principais publicações que emanaram desta pesquisa que são resumidas abaixo:

1. **Título:** SARIK - *framework* para automatizar a segurança em ambientes de orquestração kubernetes
   Autores: Jonathan G. P. dos Santos, Geraldo P. Rocha Filho e Vinícius P. Gonçalves
   Publicado em: Salão de Ferramentas do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos
   Ano de Publicação: 23 a 27 de maio de 2022.
   Qualis: A4
   Esta publicação apresentou o *framework* SARIK, que automatiza a segurança em ambientes Kubernetes por meio de regras de iptables nos PODS em uma versão inicial do *framework*.
   DOI: https://doi.org/10.5753/sbrc$_estendido$.2022.223438

2. **Título:** Enhancing IoT Device Security in Kubernetes: An Approach with Network Policies and the SARIK Framework
   Autores: Jonathan G. P. dos Santos, Geraldo P. Rocha Filho, Rodolfo I. Meneguette, Rodrigo Bonacin, Gustavo Pessin e Vinícius P. Gonçalves
   Em processo de publicação em: Elsevier - *Journal: Future Generation Computer Systems*
   Qualis: A1
   Esta publicação aprimora a abordagem do *framework* SARIK, que automatiza a segurança em ambientes Kubernetes por meio da implementação de regras de iptables no kube-proxy para o controle do tráfego de rede.

1.5 ESTRUTURA DA DISSERTAÇÃO

O restante desta dissertação está dividida da seguinte forma. No Capítulo 2, é apresentada a fundamentação teórica que tem como objetivo fornecer uma base sólida para entender os conceitos e tecnologias que sustentam os sistemas modernos baseados em contêineres e orquestração. No capítulo 3, é apresentado os trabalhos relacionados fazendo uma seleção inicial e, ao mesmo tempo, a atualização mais recente dos trabalhos disponíveis até a presente data. No Capítulo 4, é apresentado o *framework* SARIK, solução desenvolvida para esta dissertação. No Capítulo 5, é apresentado os resultados, discussões e implicações. Por fim, no Capítulo 6, são apresentadas as conclusões desta pesquisa, mostrando os resultados alcançados, limitações e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

O cenário tecnológico atual é marcado por uma rápida evolução e inovação, especialmente no campo da computação em nuvem e orquestração de contêineres. Este capítulo busca explorar o conhecimento teórico em áreas fundamentais que formam a espinha dorsal de sistemas modernos baseados em contêineres, como Docker e Kubernetes, e como eles interagem com políticas de rede implementadas através de ferramentas como *iptables* e Minikube. Portanto, este capítulo tem como objetivo fornecer uma base sólida para entender os conceitos e tecnologias que sustentam os sistemas modernos baseados em contêineres e orquestração.

Visão geral das seções abordadas:

1. Virtualização e Contêineres: Esta seção aborda a evolução histórica da virtualização e como ela levou ao desenvolvimento de tecnologias de contêineres, como o Docker. Será explorado o conceito de contêineres, suas vantagens e desvantagens, e como eles têm revolucionado o ciclo de vida do desenvolvimento de *software*.

2. Orquestração e Gerenciamento com Kubernetes: Esta seção se concentra no Kubernetes como uma plataforma de orquestração de contêineres. Ela cobre suas funcionalidades, benefícios e desafios, bem como o papel do Minikube como uma ferramenta de simulação de *clusters* Kubernetes em ambientes locais.

3. Políticas de Rede e Segurança: Esta seção explora a importância das políticas de rede em ambientes Kubernetes e sistemas Linux. Ela aborda o uso de ferramentas como *iptables* para implementar políticas de rede e os desafios comuns enfrentados durante essa implementação.

4. Grafana e Prometheus: Esta seção aborda o uso de Grafana e Prometheus como ferramentas de monitoramento em ambientes Kubernetes. Grafana oferece uma interface visual para métricas e dados, enquanto Prometheus é usado para coletar e armazenar essas métricas.

Cada uma dessas seções é projetada para fornecer uma compreensão abrangente dos componentes individuais e como eles se integram para formar sistemas robustos e eficientes. Através deste capítulo, busca-se não apenas informar, mas também preparar o leitor para os tópicos que serão abordados nos capítulos subsequentes da dissertação.

2.1 VIRTUALIZAÇÃO E CONTÊINERES

A virtualização é uma tecnologia com raízes que remontam aos anos 1960, quando a IBM a introduziu pela primeira vez com o objetivo de otimizar a utilização de seus grandes sistemas *mainframe*. Naquela época, a virtualização representava uma inovação significativa, permitindo a divisão dos recursos de um único sistema físico em múltiplos ambientes virtuais [11, 12, 13]. Isso possibilitou a execução simultânea

de diferentes aplicações e sistemas operacionais em uma única máquina, proporcionando uma maneira mais eficiente de aproveitar os recursos de *hardware*, que eram caros na época.

Com o surgimento dos servidores x86 e a crescente popularidade de sistemas operacionais como o Linux e o Windows, a virtualização experimentou um ressurgimento no início dos anos 2000. Empresas renomadas, como VMware, Microsoft e Citrix, lançaram soluções de virtualização acessíveis, levando essa tecnologia ao mercado de massa. Isso permitiu que organizações consolidassem seus *data centers*, aprimorassem a continuidade dos negócios e simplificassem a gestão de suas infraestruturas de TI [14].

Hoje, a virtualização é uma tecnologia onipresente que vai além dos servidores, abrangendo áreas como armazenamento, rede e até mesmo aplicações e funções de rede. Ela desempenha um papel fundamental em tecnologias emergentes, como computação em nuvem, contêineres e infraestrutura como serviço (IaaS), desempenhando um papel crucial na evolução e na eficiência das infraestruturas de TI modernas [15].

A tecnologia de contêineres representa um marco na evolução da virtualização. Diferentemente das máquinas virtuais, que incluem um sistema operacional completo, os contêineres são uma forma de virtualização a nível de sistema operacional que permitem executar aplicações e suas dependências em um ambiente isolado, chamado de contêiner. Ao contrário da virtualização tradicional, que emula *hardware* para executar múltiplos sistemas operacionais, contêineres compartilham o mesmo sistema operacional do *host*, mas mantêm um espaço isolado para cada aplicação. Isso torna os contêineres mais leves e mais rápidos do que as máquinas virtuais tradicionais [16]. Na Figura 2.1 é apresentado as diferenças quando uma aplicação é executada na máquina física, máquina virtual e nos contêineres.

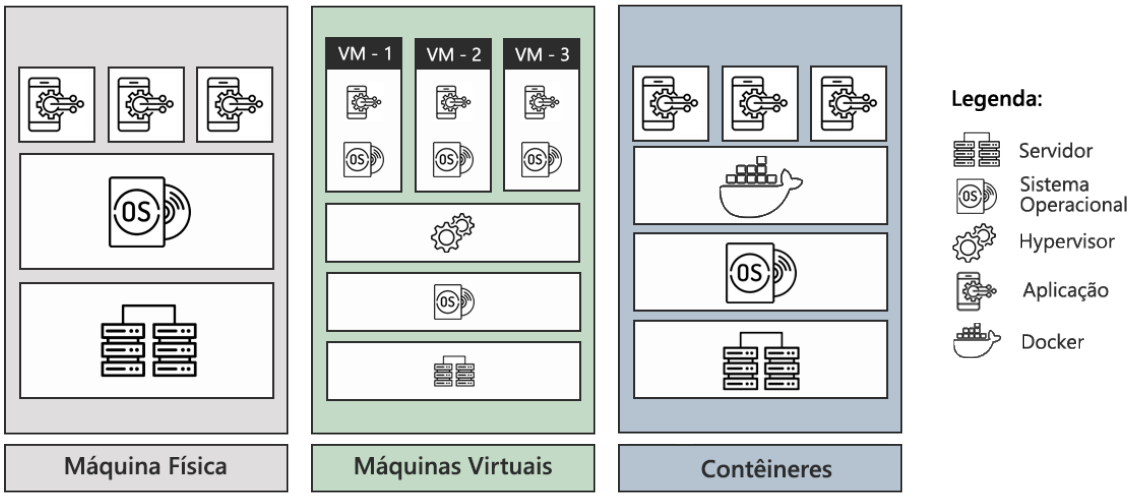

Figura 2.1: Diferenças de host/máquina virtual/contêineres.
Fonte: Adaptação do livro: Descomplicando o Docker

Os benefícios dos contêineres são numerosos e têm revolucionado o desenvolvimento de *software* e a operação de sistemas. Eles oferecem portabilidade entre diferentes sistemas e ambientes de nuvem, facilitam a orquestração e o escalonamento de aplicações, e melhoram a eficiência e a utilização de recursos [17]. Além disso, contêineres suportam DevOps e métodos ágeis de desenvolvimento, permitindo integração e entrega contínuas (CI/CD). Eles também simplificam a gestão de dependências e versões, tornando mais fácil para os desenvolvedores focar no código, em vez de se preocupar com a infraestrutura.

Nesta pesquisa/dissertação, foi adotado o Docker como o sistema de contêiner principal para a realização dos experimentos no *cluster* Kubernetes. O Docker é amplamente reconhecido como a tecnologia de containerização mais popular e é quase sinônimo desse conceito. Além disso, sua interface de comando é amplamente adotada e sua facilidade de uso o torna uma escolha sólida para a execução de contêineres em ambientes Kubernetes. Vale mencionar que existem outras tecnologias de containerização, como o Podman, que oferece benefícios em termos de segurança e integração com outras tecnologias de orquestração, o OpenVZ, focado no isolamento de servidores virtuais, e o FreeBSD Jails, específico para o sistema operacional FreeBSD. No entanto, o Docker foi escolhido devido à sua liderança no mercado e ampla adoção.

O Docker é uma plataforma de contêineres que facilita o desenvolvimento, a implantação e a execução de aplicações em um ambiente isolado. Ele usa a tecnologia de contêinerização para empacotar uma aplicação e suas dependências em um contêiner, que pode ser facilmente transferido e executado em qualquer sistema que tenha o Docker instalado [17, 18]. Isso elimina o problema comum de "funciona na minha máquina", tornando o processo de desenvolvimento e implantação muito mais eficiente e confiável.

A arquitetura do Docker é composta por vários componentes, incluindo o Docker *Engine*, que é o núcleo que executa os contêineres; o Docker *Hub*, um repositório para imagens de contêineres; e o Docker *Compose*, uma ferramenta para definir e executar aplicações multi-contêiner [19]. O Docker se diferencia de outras tecnologias de virtualização por ser mais leve, mais rápido e mais fácil de usar. Ele também integra bem com ferramentas de DevOps e serviços de nuvem, tornando-se uma escolha popular para a implantação de microserviços e outras arquiteturas de aplicação modernas.

2.2 ORQUESTRAÇÃO E GERENCIAMENTO COM KUBERNETES

Kubernetes é uma plataforma de orquestração de contêineres de código aberto que automatiza a implantação, o escalonamento e a gestão de aplicações em contêineres. Originalmente desenvolvido pelo Google, o Kubernetes agora é mantido pela *Cloud Native Computing Foundation*. A arquitetura do Kubernetes é baseada em um conjunto de componentes distribuídos que trabalham em conjunto para gerenciar o estado desejado das aplicações. Isso inclui o plano de controle, que é responsável por manter o estado global do *cluster*, e os nós de trabalho, que executam os contêineres [20].

No estudo conduzido por Borgwardt et al. [21], foi realizada uma análise comparativa entre os sistemas de orquestração de contêineres Borg, Omega e Kubernetes. O objetivo desse estudo foi destacar as características distintas de cada sistema, bem como suas respectivas vantagens e limitações. Segundo os autores, o sistema Borg, apesar de sua complexidade, é reconhecido por sua escalabilidade e maturidade. Por outro lado, o Omega é apontado como uma opção mais simples e de fácil utilização, embora menos escalável em comparação com o Borg. Por fim, o Kubernetes é considerado uma escolha intermediária, combinando a capacidade de escalabilidade com uma complexidade de uso moderada.

É importante ressaltar que a seleção do sistema de orquestração de contêineres mais apropriado para uma organização depende das necessidades específicas dessa organização. O Borg pode ser a opção preferida quando a prioridade é a escalabilidade e a maturidade do sistema, enquanto o Omega é indicado

para aqueles que valorizam a simplicidade e facilidade de uso. O Kubernetes, por sua vez, oferece uma abordagem equilibrada, combinando escalabilidade, maturidade e facilidade de utilização. Essa análise comparativa contribui significativamente para auxiliar as organizações na tomada de decisões informadas ao escolher o sistema de orquestração de contêineres que melhor atenda às suas necessidades específicas.

Já no estudo conduzido por Medel et al. [22], foi apresentada uma metodologia que se propõe a prever o desempenho de aplicações em ambientes Kubernetes. A metodologia é fundamentada na análise de dados provenientes da execução de aplicações, tais como tempo de inicialização, tempo de término, utilização de memória e utilização de CPU. Com base nesses dados, os autores desenvolvem um modelo capaz de antecipar o desempenho de aplicações futuras. Essa abordagem representa uma ferramenta de grande relevância para desenvolvedores e administradores de sistemas que buscam aprimorar o desempenho de suas aplicações em ambientes Kubernetes.

Nesse estudo conduzido por Chang et al. [23], é apresentada uma plataforma de monitoramento baseada em Kubernetes projetada para facilitar o provisionamento dinâmico de recursos em ambientes de nuvem. Essa plataforma opera por meio de um coletor de dados e um analisador de desempenho que reúnem informações relevantes das aplicações em execução no *cluster* Kubernetes. A partir desses dados, a plataforma tem a capacidade de antecipar e alocar recursos adicionais para o *cluster* conforme as necessidades emergem. A avaliação da plataforma evidenciou seu êxito na previsão e no provisionamento de recursos, contribuindo significativamente para otimizar o desempenho e a escalabilidade das aplicações hospedadas no ambiente Kubernetes. Além disso, a plataforma oferece um sistema abrangente de monitoramento, juntamente com flexibilidade na implementação, permitindo operações automáticas que aprimoram continuamente o provisionamento de recursos.

O estudo conduzido por Vayghan et al. [24] concentra-se na exploração da arquitetura de Alta Disponibilidade (HA) em aplicações de microsserviços, com ênfase na plataforma Kubernetes como uma solução viável para sua implementação. Os autores enfatizam a importância do *design* de aplicações que sejam resilientes a falhas ao empregar o Kubernetes como estrutura para a HA. O artigo apresenta um estudo de caso prático para ilustrar os desafios enfrentados e as lições aprendidas durante a implementação de um aplicativo de microsserviços no ambiente Kubernetes. A abordagem do artigo é estruturada de maneira a proporcionar uma compreensão clara e concisa dos princípios fundamentais da Alta Disponibilidade, além de oferecer recomendações valiosas para o *design* de aplicações com resiliência incorporada. No geral, esta pesquisa representa uma fonte de leitura valiosa para desenvolvedores que buscam aprofundar seus conhecimentos sobre a construção de sistemas de Alta Disponibilidade para aplicações de microsserviços, fornecendo *insights* práticos por meio do estudo de caso detalhado.

O kubernetes possuí componentes na camada de gerenciamento que desempenham funções cruciais no ecossistema do Kubernetes, tomando decisões globais sobre o *cluster*, como a alocação de Pods, e respondendo a eventos específicos do *cluster*. Estes componentes podem ser executados em qualquer nó do *cluster*, mas geralmente são centralizados em uma única máquina para simplificar a configuração. Esta organização é especialmente útil em cenários de alta disponibilidade [25]. Portanto, a Figura 2.2, apresenta arquitetura e componentes padrão do kubernetes que estão associados ao *cluster* kubernetes e são descritos de forma resumida:

- kube-apiserver: O *Frontend* da camada de gerenciamento, atua como o servidor da API e é um

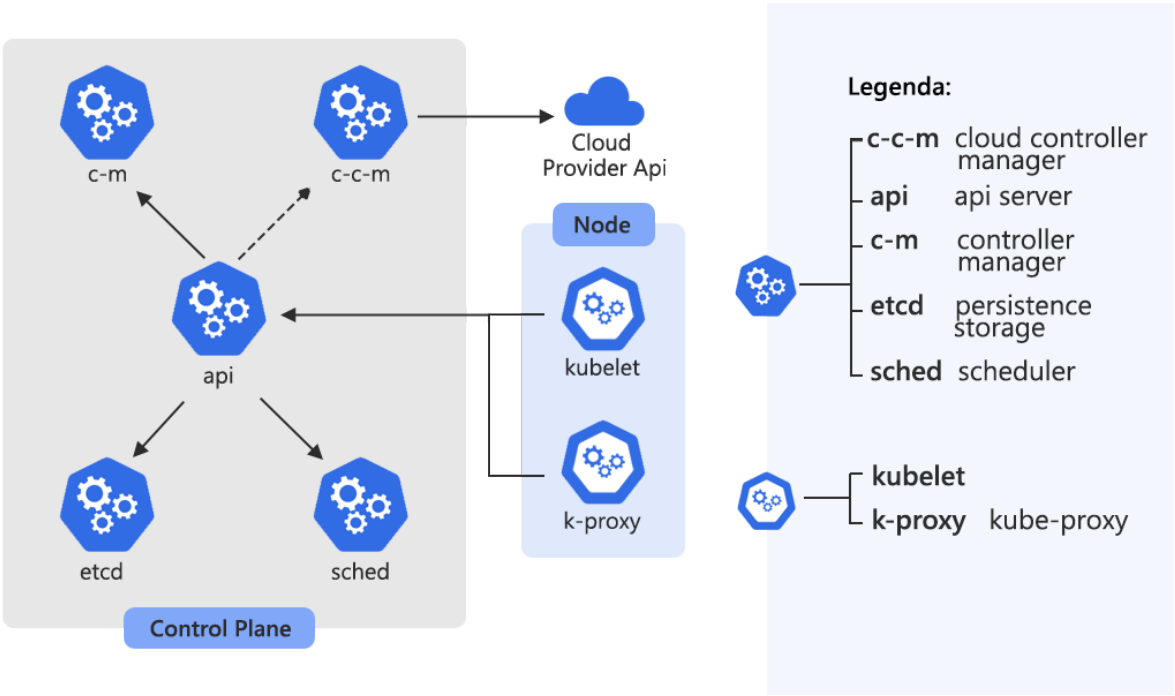

Figura 2.2: Os componentes de um *cluster* do Kubernetes.
Fonte: Adaptação de Kubernetes.io/pt-br/

componente essencial da camada de gerenciamento. Ele serve como o ponto de entrada para a API do Kubernetes e é projetado para escalar horizontalmente. Isso significa que é possível executar múltiplas instâncias do *kube-apiserver* e distribuir o tráfego entre elas para melhorar o desempenho e a tolerância a falhas.

- etcd: O armazenamento de apoio do *cluster*, é um armazenamento do tipo chave-valor de alta disponibilidade e consistência, que serve como o repositório de dados para todo o estado do *cluster* [26].
- O kube-scheduler é responsável por observar os Pods recém-criados que ainda não foram atribuídos a um nó e selecionar um nó adequado para sua execução. O processo de decisão leva em consideração vários fatores, como requisitos de recursos, restrições de *hardware* e *software*, e políticas de afinidade e antiafinidade.
- kube-controller-manager: O executor dos processos de controle, é um componente versátil que executa diversos processos de controle em um único binário. Entre os tipos de controladores estão o Controlador de Nó, responsável por monitorar a saúde dos nós, e o Controlador de *Jobs*, que gerencia tarefas únicas representadas por objetos *Job*.
- cloud-controller-manager: Integração com provedores de nuvem, é um componente especializado que permite a integração com APIs de provedores de nuvem. Ele executa controladores que são específicos para o ambiente de nuvem em que o *cluster* Kubernetes está sendo executado. Isso inclui tarefas como configurar rotas na infraestrutura de nuvem e gerenciar balanceadores de carga.
- Kube-proxy: O coração do encaminhamento de rede, componente crucial que lida com o encaminhamento de rede no *cluster* Kubernetes. Ele mantém as regras de rede nos nós, permitindo que o

tráfego de rede seja encaminhado corretamente. O *kube-proxy* é responsável por implementar uma forma de comunicação de IP virtual para os serviços. Ele pode operar em diferentes modos, incluindo o modo "*iptables*", que é o mais comum e permite o encaminhamento eficiente do tráfego. O *kube-proxy* é essencial para o balanceamento de carga entre os Pods e também para fornecer alta disponibilidade. Ele também desempenha um papel vital na implementação de políticas de rede, trabalhando em conjunto com outros componentes como o *Network Policy Controller* para garantir que as políticas de rede sejam aplicadas de forma eficaz.

Muralidharan et al. [27] aborda a implementação de um sistema de monitoramento em nuvem seguro, distribuído e confiável destinado a aplicativos de Internet das Coisas (IoT) em ambientes de cidades inteligentes. A proposta apresenta um sistema baseado em contêineres que prioriza a baixa latência e a comunicação confiável entre dispositivos IoT, com foco na interoperabilidade horizontal entre diferentes aplicativos. A contribuição essencial deste trabalho reside na oferta de uma solução eficaz para a gestão de aplicativos IoT em contextos de cidades inteligentes. O sistema em questão foi avaliado por meio de experimentações que empregaram técnicas de contêinerização utilizando Docker e a plataforma de orquestração Kubernetes.

Wei et al. [28] apresenta uma proposta de um novo algoritmo de escalonamento de recursos projetado especificamente para o Kubernetes. Este algoritmo, baseado em princípios de colônias de formigas e otimização de enxame de partículas, visa aprimorar a eficiência do processo de agendamento de tarefas. A metodologia proposta é composta por duas etapas distintas: filtragem de *hosts* e atribuição de pontuações aos *hosts*. A primeira etapa utiliza colônias de formigas para identificar os *hosts* mais adequados para alocar Pods específicos. Na segunda etapa, a atribuição de pontuações é realizada empregando técnicas de otimização que consideram fatores como a disponibilidade de recursos e a carga atual do sistema. Os experimentos conduzidos em comparação com o algoritmo padrão do Kubernetes demonstraram que o novo algoritmo foi capaz de agendar Pods de maneira mais eficiente, minimizando o desperdício de recursos e demonstrando uma maior capacidade de gerenciar cargas de trabalho dinâmicas, reduzindo, assim, o risco de congestionamentos na rede.

No âmbito deste estudo, a opção estratégica recaiu na implementação do Kubernetes em uma configuração local, recorrendo à ferramenta Minikube. Esta seleção fundamenta-se na notável capacidade do Minikube de viabilizar a criação de um *cluster* Kubernetes em ambientes de desenvolvimento e teste locais. Essa abordagem revela-se especialmente benéfica em cenários onde a rapidez na iteração e a validação de funcionalidades desempenham um papel essencial.

O Minikube [29], reconhecido como uma ferramenta de destaque, simplifica consideravelmente a execução de um *cluster* Kubernetes em um ambiente local, geralmente voltado para atividades de desenvolvimento e teste. Essencialmente, ele cria uma máquina virtual (VM) no sistema do usuário e instala um *cluster* Kubernetes de nó único. Este recurso torna-se particularmente valioso para desenvolvedores e administradores de sistemas que almejam experimentar o Kubernetes, sem os encargos complexos e dispendiosos associados à configuração de um *cluster* completo em um ambiente de nuvem ou *data center*. Importante notar que o Minikube é compatível com diversos mecanismos de virtualização, tais como VirtualBox, VMware, QEMU, KVM e Hyper-V.

O Minikube desfruta de uma posição de destaque como recurso inicial para aqueles que estão iniciando

sua jornada no aprendizado do Kubernetes e desejam adquirir experiência na orquestração de contêineres [30]. No contexto do Kubernetes, a utilização de "manifestos" configura-se como uma prática comum, sendo estes arquivos de configuração frequentemente elaborados em formato YAML e destinados a especificar a configuração dos recursos no *cluster*. Além disso, os "*Helm charts*" são pacotes que reúnem manifestos Kubernetes predefinidos, simplificando sobremaneira a implantação e gestão de aplicações de maior complexidade. O Minikube possibilita a experimentação e validação desses manifestos e *Helm charts* em um ambiente isolado, antes de sua migração para um ambiente de produção. Embora não tenha sido projetado com a finalidade de simular ambientes de produção em larga escala, o Minikube oferece uma abordagem ágil e direta para a compreensão dos princípios fundamentais do Kubernetes, bem como para a avaliação iterativa de aplicações e configurações.

Além disso, o Minikube oferece uma extensa gama de recursos que simplificam o ciclo de desenvolvimento e teste, no contexto do Kubernetes. Esses recursos compreendem o suporte para diversos *drivers* de virtualização, a disponibilidade de *plugins* especializados e uma interface de linha de comando notadamente intuitiva. Um aspecto adicional que merece destaque é a capacidade do Minikube de simular falhas de rede e latência, conferindo-lhe relevância e utilidade significativas na avaliação da resiliência de aplicações Kubernetes, especialmente em ambientes sob controle.

2.3 POLÍTICAS DE REDE E SEGURANÇA

Nesta seção, se dedica a explorar a complexa questão das políticas de rede no contexto do Kubernetes e Minikube, com especial atenção para os desafios apresentados pelo *Container Network Interface* (CNI) e a implementação do Calico [31] como uma solução para o controle de tráfego no *cluster*. Esta seção fornece uma visão abrangente dos conceitos fundamentais que embasam esse tópico crítico.

O CNI, uma especificação que define como as redes são configuradas em contêineres, tem se mostrado um fator determinante na limitação da implementação de regras de *firewall* em ambientes Kubernetes. Este obstáculo decorre, em grande parte, do fato de que o CNI não oferece suporte nativo para a configuração de regras de *firewall* em Pods e no tráfego de rede, o que cria uma lacuna na segurança que pode ser explorada por ameaças. Para contornar essa limitação, esta pesquisa de mestrado, escolheu o Calico como *plugin* de rede porque é a primeira opção sugerida na página do Minikube e, portanto, suporta políticas de rede.

Para mitigar essas limitações e garantir a segurança do tráfego no *cluster*, o Calico emergiu como uma solução eficaz. O Calico é uma plataforma de rede de código aberto que oferece um conjunto de recursos para controle avançado do tráfego no Kubernetes. Ele utiliza um modelo baseado em políticas que permite aos administradores definir como o tráfego de rede deve ser direcionado e filtrado entre os Pods e serviços. Com o Calico, é possível estabelecer políticas de rede granulares que determinam quais Pods podem se comunicar entre si, bem como quais podem acessar recursos externos.

O *framework* SARIK é capaz de gerar automaticamente as políticas de rede para *cluster* Kubernetes, independentemente do *plugin* de rede que será utilizado. Com o SARIK multi-plugin de rede, a geração de política de rede para o Flannel[32], Weave[33], Cilium[34] e Calico[31] e muitos outros *plugins* de rede populares será uma tarefa fácil e rápida.

Para escolha do *plugin* de rede mais adequado, os autores Qi et al.[35], abordam as opções de *plugins* de interface de rede em ambientes de contêineres, juntamente com as considerações de *design* essenciais para selecionar o *plugin* mais adequado. Este trabalho apresenta uma análise comparativa abrangente entre diferentes opções de *plugins* de interface de rede, levando em consideração aspectos de desempenho e recursos disponíveis. Essa análise pode ser uma ferramenta valiosa para administradores de sistemas ao determinar qual *plugin* é mais apropriado para cenários específicos. As contribuições deste estudo englobam uma visão panorâmica das alternativas de *plugins* de interface de rede disponíveis, uma análise comparativa detalhada em relação ao desempenho e recursos desses *plugins*, bem como uma lista de considerações cruciais de *design* a serem ponderadas ao optar por um determinado *plugin* de interface de rede. Este trabalho oferece informações fundamentais que podem ser úteis na tomada de decisões informadas sobre a implementação de *plugins* de interface de rede em ambientes de contêineres.

Para compreender a implementação das políticas de rede no Kubernetes, é essencial explorar os conceitos básicos do *iptables*, uma ferramenta poderosa para manipulação de regras de *firewall* em sistemas Linux. O *iptables* opera com base em três cadeias principais: entrada (*input*), saída (*output*) e encaminhamento (*forward*). A cadeia de entrada regula o tráfego direcionado ao sistema local, a cadeia de saída controla o tráfego originado no sistema local e a cadeia de encaminhamento gerencia o tráfego que é roteado pelo sistema para outros destinos. Cada uma dessas cadeias pode conter regras específicas que definem como o tráfego deve ser processado, permitindo ou bloqueando determinadas conexões [36]. No contexto do Kubernetes, as políticas de rede são traduzidas e inseridas predominantemente na tabela *filter* do *iptables*. Esta tabela é responsável por decidir se um pacote deve ser permitido ou bloqueado. Dentre as várias cadeias usadas pelo Kubernetes, destaca-se a *KUBE-FORWARD*, que é utilizada para gerenciar o tráfego de encaminhamento entre os *Pods*, e a *KUBE-SERVICES*, que gerencia o tráfego destinado a serviços Kubernetes. Adicionalmente, o Kubernetes também utiliza a cadeia auxiliar *KUBE-POD-FIREWALL* para aplicar políticas específicas aos *Pods*. Compreender esses mecanismos é fundamental para uma gestão eficaz das políticas de rede em ambientes de produção complexos que utilizam Kubernetes.

Uma consideração fundamental reside na relação entre o *iptables* e as *Network Policies* do Kubernetes. As *Network Policies* representam uma camada adicional de segurança que opera no nível do Kubernetes, permitindo que os administradores definam políticas de rede para os Pods. Embora o *iptables* e as *Network Policies* compartilhem o objetivo de controlar o tráfego de rede, eles operam em níveis diferentes. O iptables age diretamente no sistema operacional subjacente, enquanto as *Network Policies* são implementadas no plano de controle do Kubernetes. No entanto, é importante notar que o *iptables* pode ser usado em conjunto com as *Network Policies* para reforçar ainda mais o controle de segurança do tráfego no *cluster*.

As políticas de rede no Kubernetes operam por meio das cadeias de entrada (*ingress*) e saída (*egress*). A cadeia de entrada controla o tráfego direcionado a um Pod, enquanto a cadeia de saída regula o tráfego originado a partir de um Pod. Essas cadeias permitem que os administradores apliquem regras específicas para permitir ou bloquear conexões com base em critérios como IP de origem, porta de origem, porta de destino e protocolo.

2.4 GRAFANA E PROMETHEUS

O monitoramento é um aspecto crítico para qualquer sistema em produção, e isso é especialmente verdadeiro para ambientes Kubernetes. A complexidade e a natureza dinâmica desses ambientes tornam essencial ter uma visão clara do desempenho, da saúde e do estado geral do sistema. O monitoramento eficaz permite a detecção precoce de problemas, facilita a depuração e fornece dados valiosos que podem ser usados para otimizar o sistema. Além disso, em um ambiente Kubernetes, o monitoramento adequado pode ajudar a gerenciar recursos de forma mais eficiente, identificar gargalos e até mesmo prever problemas antes que eles ocorram.

Nesse contexto, o Prometheus surge como uma solução de monitoramento e alerta de código aberto altamente confiável e escalável, originalmente desenvolvido no SoundCloud. Desde a sua criação em 2012, o projeto ganhou uma adoção significativa por muitas empresas e organizações, contando com uma comunidade de desenvolvedores e usuários muito ativa. Agora, ele é um projeto independente de código aberto e mantido de forma independente por qualquer empresa. Para reforçar sua independência e esclarecer a estrutura de governança, o Prometheus juntou-se à *Cloud Native Computing Foundation* em 2016 como o segundo projeto hospedado, logo após o Kubernetes [37]. Ele foi projetado com uma arquitetura que pode se adaptar a ambientes muito dinâmicos, como é o caso dos *clusters* Kubernetes demonstrado na Figura 2.3. Uma das principais vantagens do Prometheus é sua capacidade de coletar métricas de múltiplas fontes, incluindo Kubernetes e aplicativos em contêineres. Ele usa um modelo de dados multidimensional e uma linguagem de consulta flexível, conhecida como PromQL, que permite a realização de análises complexas. Isso é particularmente útil para correlacionar métricas de diferentes serviços ou componentes e para criar alertas mais inteligentes que ajudam na manutenção proativa.

O Grafana é uma plataforma interativa de visualização de dados *open source*, desenvolvida pela Grafana Labs. Ele oferece recursos avançados para a visualização de dados, como painéis personalizáveis, gráficos e alertas. A plataforma permite aos usuários visualizar dados por meio de tabelas e gráficos unificados em um ou vários painéis, facilitando assim a interpretação e a compreensão das métricas. Além disso, o Grafana permite consultar e definir alertas sobre informações e métricas de diversas fontes, sejam elas ambientes de servidor tradicionais, *clusters* do Kubernetes ou diversos serviços em nuvem. Isso torna mais fácil analisar os dados, identificar tendências e inconsistências e, por fim, tornar os processos mais eficientes [38].

O Grafana foi construído com base nos princípios *open source* e na crença de que os dados devem ser acessíveis em toda a organização, não apenas para um pequeno grupo de pessoas. Isso promove uma cultura onde os dados podem ser facilmente encontrados e usados por qualquer pessoa que precise deles, capacitando as equipes a serem mais abertas, inovadoras e colaborativas. A integração do Grafana com o Prometheus permite que os usuários visualizem as métricas coletadas de forma mais intuitiva e significativa, tornando-se uma solução completa de monitoramento.

Combinando essas ferramentas, Grafana e Prometheus uma solução completa de monitoramento é oferecida para o monitoramento do *cluster* kubernetes. O Prometheus atua como o cérebro coletor e armazenador de métricas, enquanto o Grafana serve como os olhos que permitem a visualização dessas métricas de forma compreensível. Essa integração permite que operadores e desenvolvedores tenham uma visão

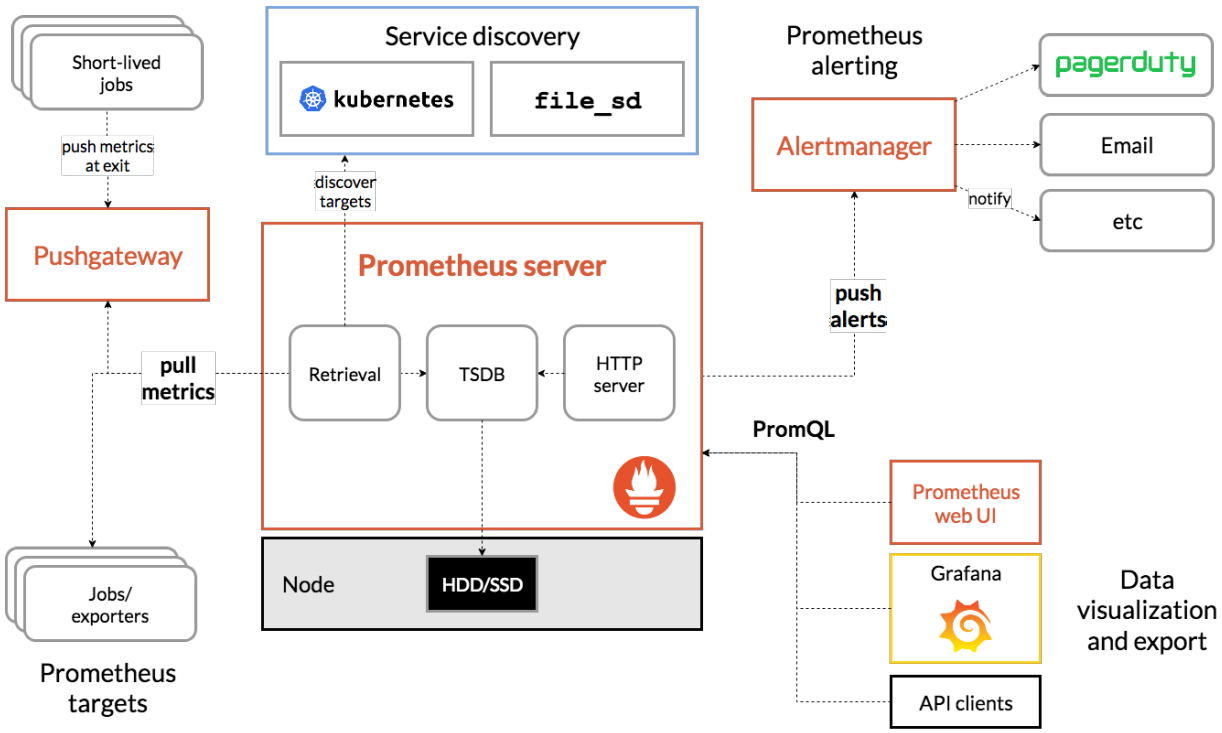

Figura 2.3: Arquitetura do Prometheus e alguns de seus componentes do ecossistema.
Fonte: Prometheus.io/docs/introduction/overview/

unificada do sistema, facilitando tanto o monitoramento em tempo real quanto a análise histórica.

Na análise das métricas mais comumente monitoradas em um ambiente Kubernetes, podemos incluir utilização de CPU, rede, memória, latência e taxa de transmissão, entre outras. Estas são métricas-chave de desempenho (KPIs) que fornecem percepções valiosas sobre a saúde e eficiência do sistema. A capacidade de monitorar essas métricas em tempo real e de visualizá-las através de painéis interativos ajuda as equipes a tomar decisões informadas, otimizar o desempenho e, em última análise, garantir que o ambiente Kubernetes seja seguro, eficiente e resiliente.

2.5 CONSIDERAÇÕES FINAIS

Este capítulo de "Fundamentação Teórica" teve como objetivo fornecer uma base sólida para a compreensão dos conceitos e componentes essenciais relacionados ao Kubernetes e à orquestração de contêineres. A inclusão de estudos acadêmicos, como o conduzido por Borgwardt et al., enriqueceu a discussão ao oferecer uma análise comparativa de diferentes sistemas de orquestração. A contribuição deste capítulo reside em sua abordagem abrangente e fundamentada, preparando o terreno para uma compreensão profunda dos desafios e soluções no contexto de orquestração de contêineres. O próximo capítulo, "Trabalhos Relacionados", expandirá essa base teórica ao explorar estudos e implementações que complementam e expandem os tópicos discutidos aqui.

3 TRABALHOS RELACIONADOS

Neste capítulo, será realizada uma revisão abrangente dos trabalhos relacionados que abordam temas essenciais no contexto desta pesquisa. A fim de contextualizar e fundamentar nossa abordagem, foi conduzido um mapeamento sistemático, abrangendo o período de 2018 até 2023, para identificar trabalhos relevantes. A busca foi realizada em bases de dados acadêmicas amplamente reconhecidas, incluindo Google *Scholar*, IEEE *Xplore*, ACM *Digital Library* e Elsevier, utilizando um conjunto de palavras-chave selecionadas, a saber: "kubernetes", "*cluster*", "segurança", "políticas de rede", "*kube-proxy*", "automação" e "*framework*", todas no idioma inglês.

Este mapeamento sistemático foi conduzido com rigor metodológico, visando identificar e analisar os estudos mais pertinentes relacionados à segurança em ambientes Kubernetes e à automação de políticas de rede. Os critérios de inclusão adotados para a seleção dos trabalhos compreendiam a relevância direta para os tópicos desta pesquisa, bem como a data de publicação dentro do intervalo especificado de cinco anos.

Os resultados deste mapeamento sistemático, apresentados na Tabela 3.1, refletem a extensiva busca realizada nas bases de dados mencionadas e evidenciam os trabalhos eleitos como relevantes até o momento desta dissertação. É importante ressaltar que, embora a busca tenha abrangido um período de cinco anos, a complexidade e dinâmica dos tópicos abordados podem influenciar a disponibilidade de estudos específicos.

Dessa forma, os trabalhos relacionados listados na Tabela 3.1 representam uma seleção inicial e, ao mesmo tempo, a atualização mais recente do conhecimento disponível até a presente data. Conforme é feita a análise e o desenvolvimento desta dissertação, novas contribuições podem emergir, complementando e enriquecendo nossa revisão da literatura e a compreensão dos tópicos abordados.

Trabalhos	*Framework* consolidado	*Framework* sem dependência	*Egress*	*Ingress*	Condução experimentos
BALABANIAN et al., 2019 [39]	✓	✗	✗	✗	✓
NAM et al., 2020 [40]	✗	✗	✓	✓	✓
KULATHUNGA, 2021 [41]	✗	✗	✗	✗	✓
ROCHA et al., 2023 [42]	✓	✗	✗	✗	✓
SYSDIG Secure, 2023 [43]	✓	✗	✓	✓	✓
KUDO et al., 2021 [44]	✗	✗	✗	✗	✗
ZHU et al., 2022 [45]	✓	✗	✗	✗	✓
BRINGHENTI et al., 2023 [46]	✓	✗	✗	✗	✗
LI et al., 2022 [47]	✗	✗	✓	✓	✗
LEE et al., 2023 [48]	✓	✗	✓	✓	✓
SARIK	✓	✓	✓	✓	✓

Tabela 3.1: Características dos trabalhos relacionados

3.1 SEGURANÇA E ISOLAMENTO DE CONTÊINERES

No domínio da segurança de contêineres, o *framework* Tocker emerge como uma solução significativa para abordar desafios associados à segurança em ambientes que utilizam contêineres Docker. Proposto por Balabanian et al. [39], o Tocker foi desenvolvido com o objetivo específico de restringir a comunicação entre contêineres ao mínimo necessário. O *framework* alcança isso através do bloqueio de portas desnecessárias, introduzindo *firewalls* como uma camada adicional de segurança. Este enfoque não apenas minimiza as superfícies de ataque, mas também oferece uma abordagem automatizada para a gestão de segurança em ambientes de contêineres. A abordagem inovadora de Tocker serviu como um catalisador para pesquisas subsequentes na área, incluindo o desenvolvimento do *framework* SARIK para ambientes Kubernetes. O SARIK foi concebido para abordar desafios semelhantes em segurança de contêineres, mas em um contexto Kubernetes, preenchendo assim o *gap* deixado pelo trabalho de Balabanian e Henriques. Este desenvolvimento subsequente ilustra o impacto duradouro e a influência do trabalho original sobre o Tocker, demonstrando como ele abriu portas para novas inovações e avanços na segurança de sistemas de orquestração.

Na proposta do "BASTION" [40], os autores identificam uma lacuna significativa na segurança de redes de contêineres e propõem uma pilha de rede de segurança inovadora, denominada BASTION. Esta pilha não apenas oferece um isolamento eficaz entre contêineres, mas também implementa políticas de segurança de rede mais refinadas. A relevância da proposta é corroborada por sua capacidade de mitigar uma variedade de ataques adversários e melhorar o desempenho da rede em até 25,4%. A eficácia da solução é validada através de testes rigorosos, tornando BASTION uma contribuição valiosa para o campo da segurança em redes de contêineres.

3.2 MONITORAMENTO E DETECÇÃO DE INTRUSÃO

O trabalho de Kulathunga [41], apresenta uma contribuição notável para a segurança em plataformas de orquestração de contêineres. O autor propõe um modelo de segurança dinâmico que incorpora um Sistema de Detecção de Intrusão (IDS) para monitorar o tráfego de rede em ambientes Kubernetes. Este IDS é especialmente configurado para categorizar o tráfego com base nos aplicativos em *namespaces* relevantes. Foi introduzido um novo componente, denominado operador-IDS, que estende a API do Kubernetes para permitir esse nível avançado de monitoramento. Este modelo dinâmico oferece uma abordagem mais granular e adaptável para a segurança, permitindo uma resposta mais eficaz a potenciais ameaças e intrusões.

O artigo [42] aborda a necessidade de sistemas de detecção de intrusão (IDS) mais eficientes em ambientes de contêiner na nuvem. Propõe um *framework* que utiliza aprendizado de máquina para detecção de anomalias em chamadas de sistema, oferecendo uma solução robusta e adaptável para segurança em tais ambientes. A principal contribuição do trabalho é o desenvolvimento de um *framework* integrável com outras ferramentas de segurança, enriquecendo a abordagem de segurança colaborativa. O sistema foi validado em um ambiente de emulação com GNS3, demonstrando eficácia na detecção de intrusões quando comparado a métodos anteriores.

3.3 POLÍTICAS DE SEGURANÇA E CONFORMIDADE

A plataforma Sysdig *Secure* [43] representa uma abordagem abrangente para a segurança em ambientes de contêineres, fundindo monitoramento, conformidade e capacidades de resposta a incidentes. Um elemento-chave da estratégia de segurança desse sistema envolve o uso de um controlador de admissão, que gera políticas de segurança antes da implantação de imagens. O Sysdig *Secure* realiza a coleta contínua de dados em tempo real de diversas fontes, incluindo o *kernel* do Linux e as APIs do Docker e Kubernetes. Esses dados são analisados usando técnicas de aprendizado de máquina e inteligência artificial. A contribuição mais notável do Sysdig *Secure* é a entrega de uma solução completamente integrada que capacita a detecção de ameaças em tempo real, a garantia da conformidade com políticas de segurança e a resposta ágil a incidentes. Vale ressaltar que, por ser uma solução comercial, o acesso a essa plataforma pode estar sujeito a restrições de disponibilidade para alguns usuários.

Outro trabalho relevante na área [44] apresenta uma abordagem destinada a fortalecer a integridade dos recursos gerenciados pelo Kubernetes, por meio da verificação de assinaturas no controlador de admissão. A proposta desses autores visa resolver o desafio recorrente relacionado à inconsistência entre os recursos assinados na solicitação de admissão e as mensagens de assinatura geradas automaticamente pelo Kubernetes. Essa incongruência pode, por vezes, resultar em falhas na verificação de assinatura. A principal contribuição desse trabalho é a oferta de uma solução voltada para garantir a integridade dos recursos dentro do ambiente Kubernetes. No entanto, é importante observar que uma limitação identificada está relacionada ao potencial aumento na sobrecarga de verificação de assinaturas durante o processo de admissão. Essa consideração é relevante para a avaliação e implementação dessa abordagem em cenários específicos.

3.4 AUTOMAÇÃO E OTIMIZAÇÃO DE POLÍTICAS DE SEGURANÇA

No estudo conduzido por Zhu et al. [45], foi proposta uma solução automatizada denominada Kub-Sec para a criação de perfis *AppArmor* em *clusters* Kubernetes. A geração manual de perfis é um processo demorado e propenso a erros humanos, e a pesquisa atual identificou uma lacuna em relação à ausência de uma solução automatizada. A abordagem proposta utiliza um mecanismo de análise do tráfego de rede para identificar os recursos acessados pelos contêineres em execução, posteriormente gerando perfis *AppArmor* correspondentes. Essa abordagem reduz significativamente o tempo e o esforço necessários para implementar políticas de segurança, assegurando que somente recursos autorizados sejam acessados pelos contêineres. A avaliação empírica do Kub-Sec demonstrou sua capacidade de criar perfis *AppArmor* precisos e eficazes para os contêineres em execução. Essa solução, de fácil utilização, desempenha um papel fundamental na garantia da segurança do *cluster* Kubernetes.

O artigo [46] aborda a complexidade de configurar manualmente políticas de segurança em arquiteturas Kubernetes *multi-cluster*. Propõe uma solução automatizada, o *Multi-Cluster Orchestrator*, para gerar e implantar políticas de segurança de rede em cada *cluster*. Essa abordagem tem o mérito de minimizar erros humanos e facilitar a comunicação entre *clusters*, contribuindo significativamente para a eficiência e segurança em ambientes Kubernetes *multi-cluster*. Embora o artigo afirme que a solução foi validada em

casos de uso realistas, falta uma descrição detalhada da metodologia de validação.

3.5 ABORDAGENS INOVADORAS E EXTENSÕES DE API

No estudo conduzido por Kano et al.[47], uma ferramenta de verificação de políticas de rede em ambientes nativos em nuvem é proposta. Esta ferramenta adota uma abordagem de modelagem formal para verificação e otimização de políticas de rede. A metodologia subjacente envolve a criação de um modelo formal baseado em redes de autômatos que descreve com precisão o comportamento do tráfego de rede em um ambiente Kubernetes. A principal contribuição deste trabalho reside na disponibilização de uma solução eficiente de verificação de políticas de rede, capaz de detectar conflitos e inconsistências em tempo real, mitigando, assim, potenciais interrupções na rede. Contudo, é relevante notar que o estudo apresenta algumas limitações, incluindo a falta de suporte abrangente para todos os recursos de rede do Kubernetes e a necessidade de configuração manual em certos cenários específicos. Estas considerações devem ser levadas em conta ao avaliar a aplicabilidade desta abordagem em ambientes particulares.

Lee et al.[48] aborda o desafio de segurança em ambientes de contêineres, particularmente focando em erros humanos e configurações inadequadas que podem comprometer a segurança. A pesquisa introduz o KUNERVA, uma solução automatizada que utiliza *logs* de rede para gerar um conjunto mínimo, mas eficaz, de políticas de segurança de rede. A proposta se destaca por sua integração com o sistema de aplicação de políticas *Gatekeeper*, visando aprimorar a confiabilidade das políticas. O trabalho contribui significativamente para a automação da segurança em ambientes de contêineres, os autores fornecem detalhes sobre a avaliação e validação da eficácia da solução proposta.

3.6 CARACTERÍSTICAS E DIFERENCIAIS DO SARIK AOS DEMAIS TRABALHOS

O *framework* SARIK se destaca dos outros trabalhos analisados em vários aspectos. Enquanto muitos *frameworks* e soluções focam em aspectos específicos da segurança de contêineres, como isolamento de rede [39], [40], detecção de intrusão [41], [42] ou conformidade [43], o SARIK oferece uma abordagem mais holística e modular para a gestão de políticas de rede em ambientes Kubernetes. O *framework* não apenas permite a configuração manual como a automática e também a exclusão de políticas de rede tanto para tráfego de entrada (*ingress*) quanto de saída (*egress*), mas também oferece funcionalidades avançadas como monitoramento em tempo real, validação de políticas e *backup*. Essa flexibilidade e abrangência tornam o SARIK uma solução única, pois ele aborda múltiplas facetas da segurança de rede em um único *framework*. Além disso, o SARIK incorpora um módulo de ajuda integrado e verificações de ambiente, tornando-o mais acessível para usuários com diferentes níveis de expertise. Essa combinação de funcionalidades torna o SARIK uma solução robusta e flexível, preenchendo lacunas deixadas por outros trabalhos que focam em áreas mais restritas da segurança em contêineres e ambientes Kubernetes.

3.7 CONSIDERAÇÕES FINAIS

O capítulo "Trabalhos Relacionados" serviu como uma plataforma crítica para contextualizar esta pesquisa dentro do cenário mais amplo de estudos focados na orquestração de contêineres, gerenciamento de *clusters* e, mais especificamente, na segurança desses ambientes. Através da análise detalhada de diversos trabalhos, desde o Tocker, que foca na minimização da superfície de ataque em contêineres, até o Sysdig *Secure*, que oferece uma solução comercial abrangente, foi possível identificar tanto as lacunas existentes na literatura quanto as inovações mais recentes no campo.

Esta revisão foi fundamental para validar as abordagens adotadas neste estudo e para compreender o estado da arte em segurança de contêineres e orquestração. Além disso, a organização dos trabalhos em subseções temáticas permitiu uma compreensão mais nítida das várias dimensões que esta pesquisa aborda, desde políticas de rede até monitoramento e detecção de intrusões.

O capítulo também estabelece a relevância e a singularidade da contribuição deste estudo ao campo, particularmente no que diz respeito ao *framework* SARIK. Este *framework* se distingue por sua abordagem modular e flexível para a gestão de políticas de rede, algo que será explorado em profundidade no próximo capítulo, intitulado "*Framework* SARIK". Nesse capítulo subsequente, será discutido a arquitetura do SARIK, suas funcionalidades e aplicações práticas. Com essa base sólida estabelecida, o aprofundamento e discussão sobre o impacto e as implicações do *framework* SARIK no ecossistema mais amplo de segurança de contêineres e gerenciamento de *clusters* será o foco do próximo capítulo.

4 FRAMEWORK SARIK

Neste capítulo, o foco é apresentar e discutir o *framework* SARIK, uma solução para a segurança em ambientes Kubernetes através de políticas de rede e controle de acesso. O SARIK, cujo acrônimo significa "**S**egurança **A**utomática de **R**egras de **I**ptables no **K**ubernetes", é categorizado como um *framework* devido à sua estrutura modular e extensível. Ele fornece um conjunto de ferramentas e módulos funcionais para automatizar e gerenciar políticas de rede, permitindo que novas funcionalidades sejam adicionadas como módulos separados. Esta modularidade torna o SARIK altamente adaptável e extensível, surgindo como uma resposta às crescentes preocupações relacionadas à segurança em ambientes de orquestração, especialmente à medida que a adoção do Kubernetes continua a se expandir. O capítulo inicia fornecendo uma visão detalhada da arquitetura do SARIK, delineando suas funcionalidades essenciais e como se integra ao ecossistema Kubernetes. Em seguida, examinamos a relevância deste *framework* à luz do contexto atual de ameaças cibernéticas e das crescentes demandas por segurança em ambientes conteinerizados.

Para facilitar a compreensão da arquitetura e do funcionamento do *Framework* SARIK, uma Figura 4.1 ilustrativa é apresentada neste capítulo. Esta figura esclarece quatro componentes estruturais para a implementação e operação eficaz do SARIK. O primeiro componente é o *cluster* Minikube/Kubernetes, que atua como a fundação na qual o *framework* e suas respectivas políticas de rede são implementadas. O segundo componente destaca os módulos do *framework* SARIK, ilustrando as diversas funcionalidades e operações automatizadas que ele oferece para a gestão eficiente de políticas de rede. O terceiro componente se concentra nas políticas de rede, fornecendo *insights* sobre como elas são geradas, aplicadas e administradas dentro do ambiente Kubernetes. O quarto e último componente é o *Kube-proxy*, que serve como um intermediário na comunicação entre os Pods e as políticas de rede, sendo responsável por adicionar os manifestos gerados e aplicá-los no *iptables*. Esta figura tem o objetivo de elucidar a estrutura proposta do *framework* SARIK, sublinhando a interação e integração de seus componentes para oferecer uma solução de segurança em ambientes Kubernetes. As seções subsequentes detalharão cada um dos quatro componentes do *framework* SARIK.

4.1 CLUSTER MINIKUBE

Esta seção tem como objetivo elucidar o papel crítico do *plugin* de rede *Container Network Interface* (CNI) no contexto do *cluster* Minikube, especialmente no que diz respeito à implementação de políticas de rede eficazes e seguras. O foco será em como o *framework* SARIK, que é o objeto central desta dissertação, se integra com as políticas de rede do Kubernetes e como a escolha do *plugin* de rede pode impactar essa integração.

O *framework* SARIK faz uso extensivo das políticas de rede do Kubernetes para assegurar uma comunicação segura e eficiente entre os PODs. Essas políticas são vitais para estabelecer restrições de tráfego, tanto dentro de *namespaces* individuais quanto entre elas. No entanto, é crucial entender que o Minikube, uma implementação frequentemente utilizada do Kubernetes para ambientes de desenvolvimento local,

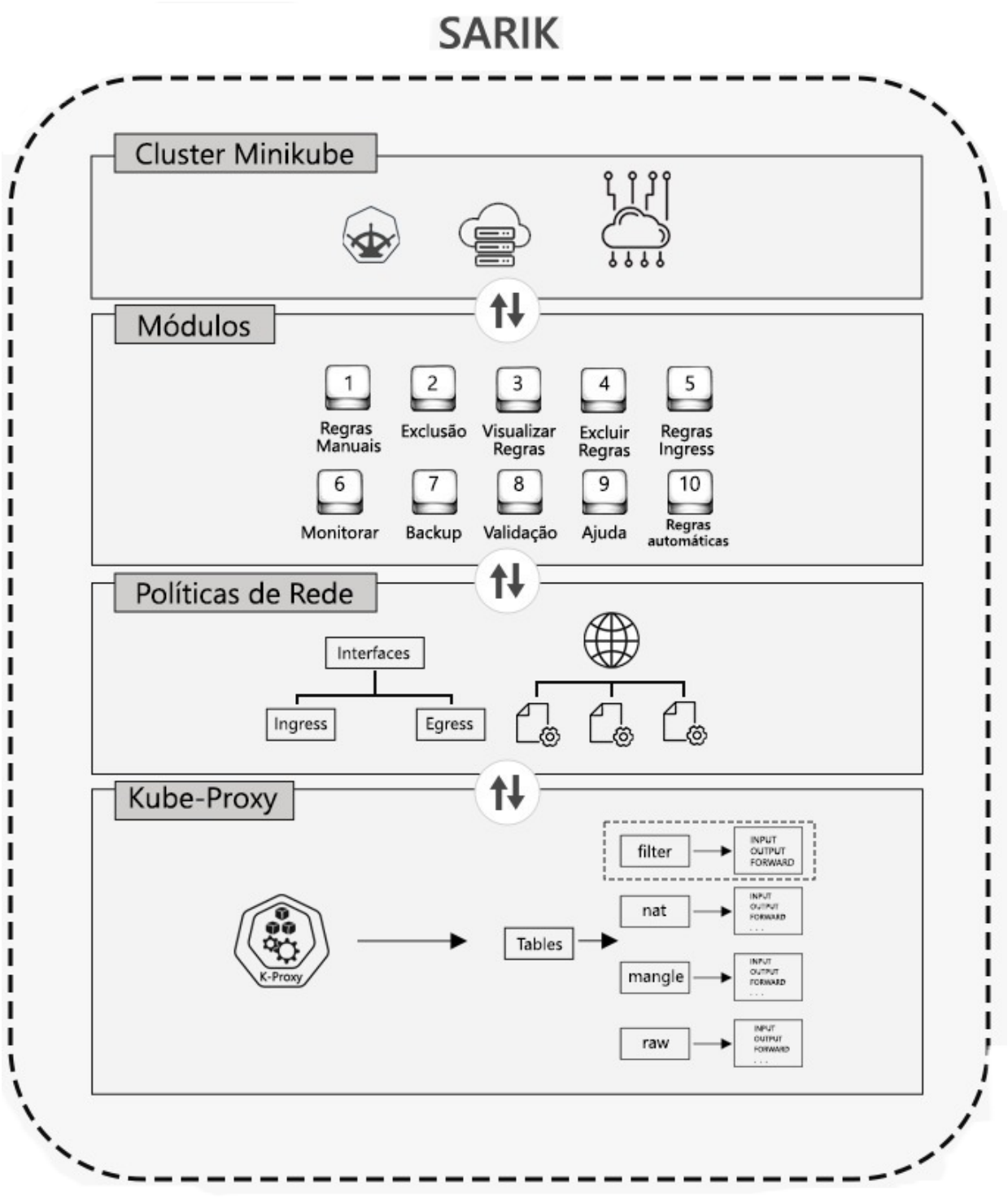

Figura 4.1: Arquitetura do *framework* SARIK
Fonte: Próprio autor

não oferece suporte nativo para políticas de rede. Isso se deve à sua dependência do plugin CNI para o gerenciamento de rede.

Embora o CNI seja eficiente em fornecer a infraestrutura de rede necessária para os PODs, incluindo configurações de endereçamento IP, ele não foi projetado para aplicar diretamente políticas de rede e traduzi-las em regras de *firewall* para *iptables*. Esta é uma lacuna significativa, especialmente para organizações e desenvolvedores que buscam um controle mais granular sobre o tráfego de rede em seus *clusters*.

Diante dessa limitação, é imperativo explorar alternativas que possam preencher essa lacuna. O estudo de Qi et al. [35] oferece *insights* valiosos neste contexto. Os autores realizaram uma análise abrangente dos diversos plugins de interface de rede disponíveis para ambientes de contêineres. Entre as várias opções consideradas, como Flannel [32], Weave [33], e Cilium [34], o plugin Calico [31] emergiu como a escolha preferencial.

A decisão de adotar o Calico foi influenciada por vários fatores. Primeiramente, ele vem com uma forte recomendação do próprio Minikube [49], o que sugere uma integração mais suave e confiável. Além disso, o Calico é conhecido por sua capacidade de suportar políticas de rede, o que o torna uma solução ideal para superar as limitações do CNI padrão. Isso não apenas permite um controle mais rigoroso sobre o tráfego de rede, mas também facilita a implementação de políticas de segurança mais complexas, algo que é de extrema importância no contexto do *framework* SARIK.

Em resumo, a escolha do *plugin* de rede é uma consideração crítica na configuração e operação de um *cluster* Kubernetes seguro e eficiente. A adoção do Calico, neste caso, não é apenas uma escolha pragmática, mas também uma decisão estratégica que potencialmente impacta a eficácia das políticas de rede implementadas, contribuindo assim para segurança do sistema como um todo.

4.2 MÓDULOS

O *framework* SARIK é uma solução modularizada para a gestão de políticas de rede em ambientes Kubernetes, projetada para atender a uma variedade de necessidades de segurança e desempenho. Composto por uma série de módulos interconectados, cada um desempenha uma função que contribui para a eficácia da ferramenta. Estes módulos foram projetados para executar uma gama de tarefas, desde automatização das regras, inclusão de regras manuais, visualiazação das regras, exclusão das regras automática e manual, *backup* das regras, monitoramento e até a validação das regras implementadas no *cluster*. Esta abordagem modular não só torna o SARIK altamente escalável e fácil de manter, mas também oferece aos usuários a flexibilidade de personalizar suas operações de acordo com requisitos desejados. Cada módulos será explicado a seguir, esclarecendo seus algoritmos e funcionalidades, para oferecer uma visão completa e detalhada da arquitetura e capacidades do SARIK.

Para estabelecer um ambiente de execução seguro e eficaz, o *framework* SARIK realiza uma série de verificações de ambiente que vão além da simples detecção de diretórios e dependências. Utilizando chamadas de sistema, ele verifica a presença de diretórios como "*policies*", "*ingress*", "*egress*" e "*backup_policies*". Se esses diretórios não estiverem presentes, o algoritmo os cria automaticamente, garan-

tindo que o ambiente esteja preparado para as etapas subsequentes. Além disso, o *script* verifica a instalação de ferramentas essenciais como "kubectl" e "docker" usando comandos "*which*". A ausência de qualquer uma dessas ferramentas resulta na interrupção da execução do *script* e na geração de uma mensagem de erro, garantindo que o usuário esteja ciente que são necessárias.

- **Módulo 1: Configuração manual de políticas na interface de saída (*Egress*)**
  Este módulo foi projetado para permitir uma configuração personalizada de políticas de rede na interface de saída (*egress*). O algoritmo começa listando todas as *namespaces* disponíveis no *cluster* Kubernetes, permitindo que o usuário faça uma seleção informada. Uma vez que a *namespace* é selecionada, o *script* lista todos os Pods dentro dessa *namespace*, fornecendo ao usuário a flexibilidade de escolher um Pod específico. O usuário pode então especificar as portas e protocolos que deseja bloquear. O *script* gera dinamicamente um arquivo YAML armazenando no diretório "*egress*" que define a política de rede e o aplica usando o comando "*kubectl apply -f egress/* ", garantindo que as regras sejam efetivamente implementadas.

- **Módulo 2: Módulo de exclusão automática das políticas**
  Este módulo é ativado quando o usuário invoca o *script* com as *flags* -D ou - *-delete*. Ele foi projetado para fornecer uma maneira rápida e eficiente de remover todas as políticas de rede das *namespaces* do *cluster*. O algoritmo lista todas as *namespaces* disponíveis e em seguida, executa o comando "*kubectl delete networkpolicy –all -n [NAMESPACE]*", garantindo que todas as políticas de rede sejam removidas, o que é especialmente útil em cenários que exigem uma redefinição rápida das políticas de rede.

- **Módulo 3: Visualização das políticas de rede ativas**
  Este módulo foi projetado para fornecer uma visão clara e compreensível das políticas de rede ativas em uma *namespace* específica. Utiliza o comando "*kubectl get networkpolicy -n [NAMESPACE]*" para extrair informações sobre as políticas de rede. Em seguida, este módulo formata essas informações e as exibe de uma maneira fácil de ler, permitindo uma rápida análise e monitoramento das políticas ativas. Este módulo é crucial para a manutenção contínua e o ajuste fino das políticas de rede.

- **Módulo 4: Exclusão manual das políticas**
  Este módulo permite a remoção seletiva de políticas de rede, fornecendo ao usuário um controle mais granular sobre as regras de segurança. O módulo lista todas as políticas de rede para o Pod selecionado na *namespace* escolhida. O usuário pode então selecionar uma política específica para exclusão. Este módulo extrai o nome da política e a porta associada e executa o comando "*kubectl delete networkpolicy [POLICY_NAME] -n [NAMESPACE]*", garantindo que apenas a política selecionada seja removida.

- **Módulo 5: Configuração manual das políticas na interface de entrada (*ingress*)**
  Este módulo é uma extensão do módulo de configuração manual de políticas na interface de saída (*egress*), mas com foco na configuração de políticas de rede para a interface de entrada (*ingress*). O algoritmo segue o mesmo fluxo operacional: seleção de *namespace*, seleção de Pod e especificação de portas e protocolos. No entanto, o arquivo YAML gerado é configurado para regras de ingresso e

armazenado no diretório "*ingress*", permitindo ao usuário estabelecer políticas de segurança para o tráfego de entrada.

- **Módulo 6: Monitoramento**
  Este módulo fornece um mecanismo de monitoramento em tempo real para as políticas de rede ativas. Utiliza uma série de comandos "*kubectl get networkpolicy -o json*" para extrair informações detalhadas sobre cada política de rede e caso exista políticas de rede na *namespace* selecionada o comando "*kubectl describe networkpolicy policy -n ns*" é executado mostrando a descrição detalhada das políticas ativas. Este módulo analisa essas informações e as apresenta de forma formatada, fornecendo *insights* valiosos sobre o estado atual das políticas de rede. Este módulo é vital para a manutenção e o ajuste contínuo das políticas de rede.

- **Módulo 7: *Backup***
  Este módulo é responsável por criar *backups* seguros das políticas de rede atuais. Ele armazena essas cópias no diretório "*backup_policies*", utilizando comandos de cópia de arquivo. Este módulo é crucial para a recuperação rápida em casos de falhas ou erros, garantindo que as políticas possam ser restauradas com facilidade. Para realização dessa etapa, o seguinte comando é utilizado "*kubectl get networkpolicy -n ns –no-headers -o custom-columns=":metadata.name"*" para testar se existe políticas de rede. Caso exista, um *loop* é utilizado para percorrer todas as regras existentes e em seguida é utilizado o comando "*kubectl get networkpolicy policy -n ns -o yaml > backup_policies/$policy.yaml*" armazenando a política no diretório específicado.

- **Módulo 8: Validação das políticas**
  Este módulo executa uma análise de segurança rigorosa nas políticas de rede ativas. Utiliza o comando "*kubectl get networkpolicy -o json*" e a ferramenta "jq" para analisar as políticas e identificar possíveis vulnerabilidades, como a permissão de tráfego de qualquer origem. Este módulo é fundamental para garantir que as políticas de rede implementadas estejam em conformidade com as melhores práticas de segurança. Contudo, até o momento em que este trabalho foi escrito, apenas a validação por porta, protocolo e especificação de interface está funcionando. O comando para testar o endereçamento por IP ainda necessita de melhorias.

- **Módulo 9: Ajuda**
  O Módulo de Ajuda serve como um manual de usuário integrado e é ativado quando o usuário invoca o *script* com as *flags* -h ou - -*help*. Este módulo não apenas imprime um guia de usuário no terminal, mas também oferece uma visão geral abrangente das funcionalidades do SARIK. Utilizando uma série de instruções "*echo*", o módulo fornece descrições detalhadas de cada módulo e suas respectivas funcionalidades, tornando a interação com o *framework* mais intuitiva. Além disso, o Módulo de Ajuda apresenta uma variedade de opções que podem ser acessadas por meio de *flags* específicas. Por exemplo, a *flag* -mn ou - -manual permite ao usuário configurar políticas de rede manualmente para a interface de saída, enquanto a *flag* -D ou - -*delete* ativa a exclusão automática de todas as políticas de rede. Outras *flags* incluem -l ou - -*view* para visualização de políticas de rede, -d ou - -del para exclusão individual de políticas, -i ou - -*ingress* para configuração manual de políticas na interface de entrada, -m ou - -monitor para monitoramento do impacto das políticas, -b ou - -*backup* para fazer *backup* das políticas atuais, e -v ou - -*validate* para validação experimental das políticas

de rede. Cada uma dessas *flags* é projetada para fornecer ao usuário um controle mais granular sobre as diversas funcionalidades do SARIK.

- **Módulo 10: Geração automática de políticas de rede na interface de saída (*egress*)**
  Este módulo constitui uma das partes mais cruciais do *framework* SARIK. Este módulo foi projetado para automatizar a configuração de políticas de rede em ambientes Kubernetes. Ele é ativado por padrão quando o *script* é executado sem parâmetros adicionais, tornando-se uma opção ideal para usuários que buscam uma configuração de políticas de rede rápida e eficiente.

  O algoritmo subjacente a este módulo inicia sua execução identificando todas as *namespaces* existentes no *cluster* Kubernetes. Posteriormente, para cada *namespace* identificada, o algoritmo lista os Pods e serviços associados. Utilizando essas informações coletadas, o módulo gera manifestos YAML que definem políticas de rede, especificamente voltadas para bloquear o tráfego de saída para portas predefinidas. Estas portas são armazenadas em um *array* denominado *BLOCKED_PORTS*, que inclui portas como 7, 80, 443 e 22, frequentemente associadas a tráfego indesejado ou potencialmente inseguro.

  Para a manipulação e extração de informações relevantes, o módulo emprega expressões regulares como "AWK". Além disso, utiliza a função *readarray* para armazenar esses dados em uma estrutura de *array*, facilitando a iteração sobre cada Pod e serviço na criação de políticas de rede individuais. Cada uma dessas políticas é armazenada em um arquivo YAML separado, que é posteriormente aplicado ao *cluster* Kubernetes através do comando "*kubectl apply -f policies/*".

  O módulo também realiza uma verificação para assegurar a existência de um diretório chamado "*policies*". Caso este diretório não exista, ele é criado automaticamente. Se já existir, qualquer conteúdo anterior é removido para evitar possíveis conflitos, garantindo que apenas as políticas de rede mais atualizadas sejam aplicadas.

  Ao concluir sua execução, o módulo aplica as políticas de rede geradas, restringindo efetivamente o tráfego de saída com base nas portas especificadas. Este processo é acompanhado por uma barra de progresso, fornecendo *feedback* visual ao usuário durante a execução.

  Em resumo, o Módulo de geração automática de políticas de rede na interface de saída (*egress*) oferece uma solução abrangente para usuários que requerem uma configuração de políticas de rede ágil, eliminando a necessidade de intervenção manual.

4.3 AUTOMAÇÃO DAS POLÍTICAS DE REDE

A implementação de políticas de rede automatizadas é um dos pilares para o aprimoramento da segurança em infraestruturas Kubernetes. Esta seção se propõe a detalhar o processo de automação no SARIK, destacando como ele se alinha com as necessidades emergentes de segurança cibernética em ambientes dinâmicos de nuvem.

No desenvolvimento do SARIK, uma ênfase particular foi colocada na automatização dessas políticas, como evidenciado pela Figura 4.2, que ilustra o fluxo de trabalho do processo de automação. Este processo é dividido em sete etapas estratégicas, cada uma com sua funcionalidade específica, garantindo uma

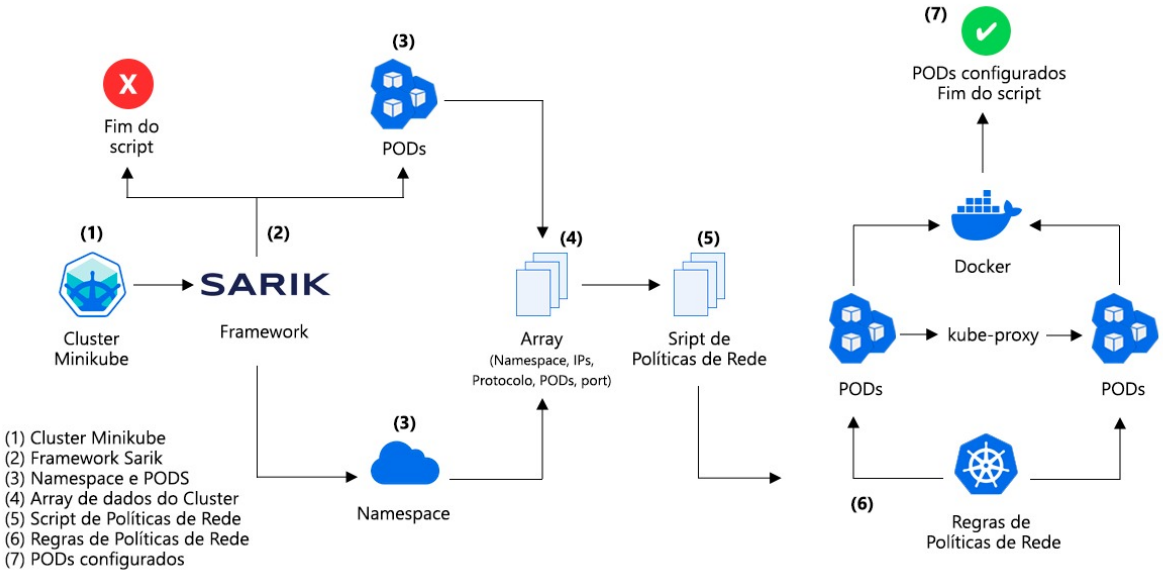

Figura 4.2: Automação das políticas de rede do SARIK

implementação precisa e eficaz.

O processo de automação inicia-se pela confirmação do framework em executar o script, o que engloba a realização de testes para verificar se todas as dependências e ajustes necessários estão em ordem. Nas etapas (1) e (2) da Figura 4.2, são realizados vários testes para determinar se o *framework* está apto a executar o *script*. Após a confirmação dessa capacidade, o SARIK avança para o levantamento minucioso das configurações do cluster. Na etapa (3), o *framework* realiza o mapeamento das configurações do *cluster* e armazena esses dados em variáveis ou *arrays*, dessa forma, utiliza-se esses dados para elaborar políticas de rede que sejam apropriadas.

Na sequência do processo, a etapa (4) do procedimento envolve a coleta e organização de dados. Durante esta etapa, o *framework* adota a estratégia de manipulação de informações que apresentam quebras de linha. Tal manipulação é realizada por meio da utilização da função *readarray*, a qual é responsável por alocar cada linha de dados em índices distintos de um *array*, garantindo assim uma organização eficaz e acessível das informações coletadas.

Na etapa (5), o SARIK emprega estas informações organizadas para a criação de diversos manifestos yaml. Esses manifestos contêm as regras de política de rede, ajustadas com base nos dados coletados, incluindo IPs, protocolos e portas específicas. Este é um exemplo claro da abordagem customizada do SARIK, que garante que as políticas de rede sejam não apenas abrangentes, mas também específicas para as necessidades de cada ambiente.

Já na etapa (6) do processo delineado, o *framework* inicia com a execução do comando destinado à criação das políticas de rede específicas para cada Pod, em seguida, essas regras são convertidas em regras de *iptables*, que são executadas no *kube-proxy*, permitindo ou negando o tráfego de pacotes entre diferentes hosts ou redes.

Por fim, ao alcançar a etapa (7), o script conclui a configuração das politicas de redes nos Pods.

4.4 DETALHAMENTO DAS ETAPAS DE IMPLEMENTAÇÃO DO FRAMEWORK SARIK

A implementação do *framework* SARIK em um ambiente Kubernetes é um processo que pode ser decomposto em etapas fundamentais. Cada etapa desempenha um papel na operacionalização das políticas de segurança de rede, garantindo que o sistema seja protegido de maneira eficiente e confiável. As figuras 4.3, 4.4 e 4.5, proporcionam uma representação visual dessas etapas, facilitando a compreensão de seu fluxo e interações.

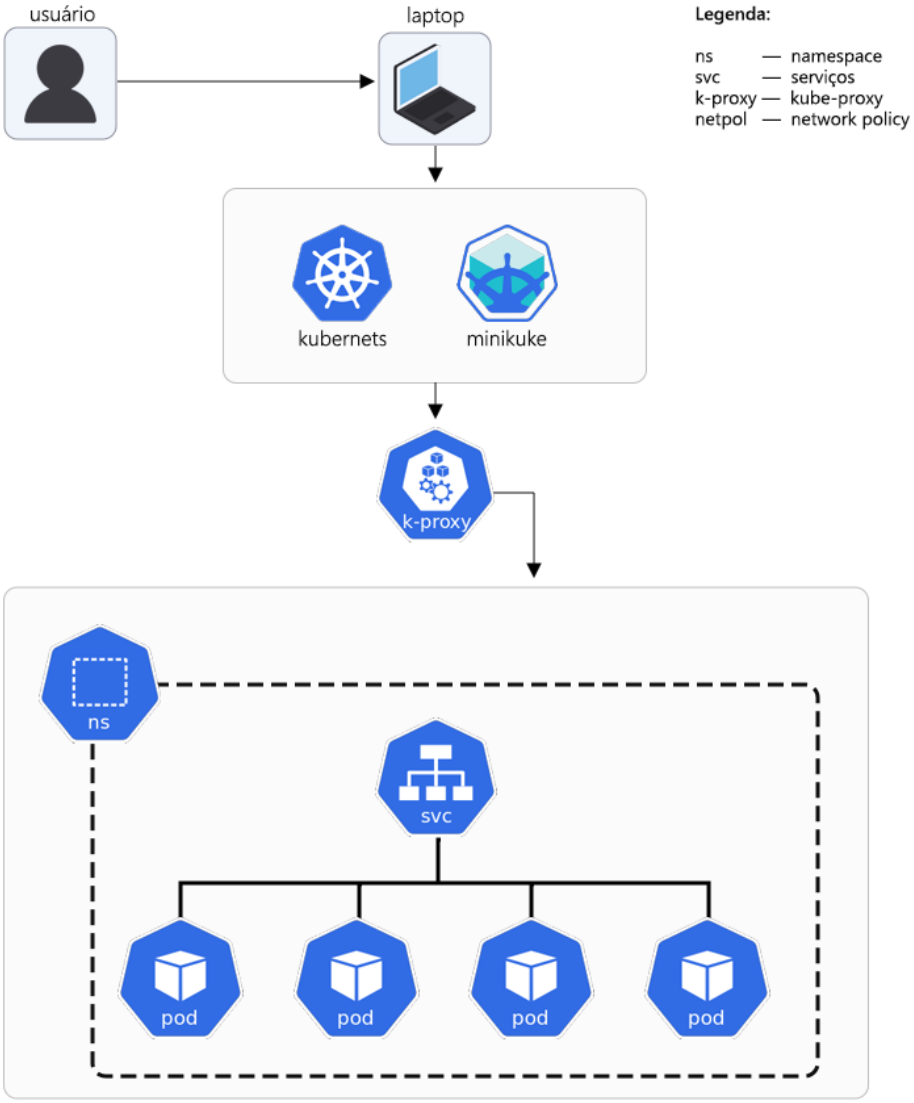

Figura 4.3: Etapa 1 da implementação

Como demonstrado na Figura 4.3, o *cluster* Kubernetes operando sob o Minikube, em sua configuração padrão, não implementa políticas de rede restritivas, permitindo que o tráfego flua livremente entre todos os pontos de comunicação. Isso é evidenciado pelo fato de que o usuário consegue acessar o *cluster* diretamente através de seu laptop. O procedimento inicia com o usuário executando o comando "mini-

kube *start*", que ativa os serviços do Docker e do Kubernetes, configurando o ambiente necessário para o *deployment* da aplicação.

Após a inicialização do Minikube, dá-se início ao processo de configuração dos recursos da aplicação em teste. O primeiro passo é a criação de um *namespace* específico, denominado "vote", que serve como um escopo lógico para os recursos e componentes da aplicação. Em seguida, são definidos os serviços ("*services*") que determinam como a aplicação será exposta e acessada dentro do *cluster*, bem como fora dele, se necessário. Finalmente, procede-se com a criação dos *deployments*, que são as manifestações dos *containers* em execução, conhecidos como Pods no ecossistema Kubernetes.

É importante notar, conforme ilustrado na figura, a ausência de "netpol", ou políticas de rede, que são responsáveis por definir as regras de comunicação permitidas entre os diferentes Pods do *namespace*. A omissão das políticas de rede é devida à limitação do *plugin* CNI (*Container Network Interface*) utilizado pelo Minikube, que não suporta tais políticas na configuração padrão. Este cenário destaca a necessidade de uma solução como o SARIK, que é capaz de implementar e gerenciar políticas de rede, suprindo essa lacuna de segurança e permitindo um controle sobre o tráfego de rede dentro do *cluster* Kubernetes.

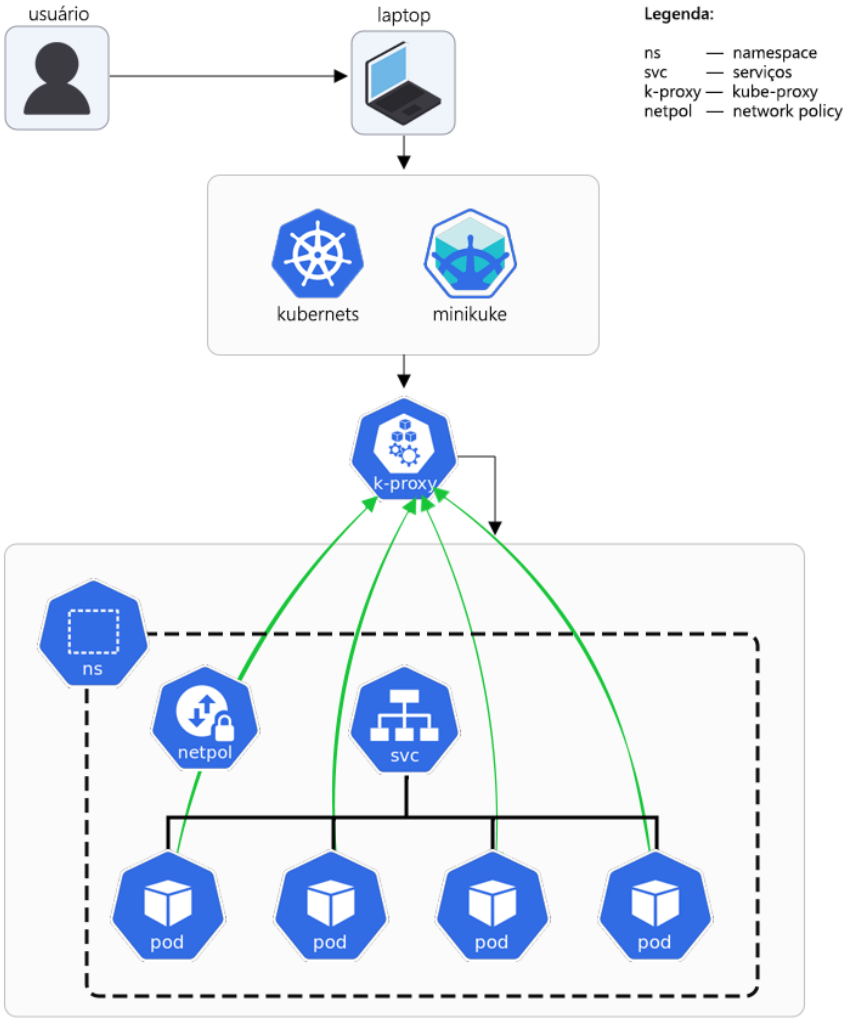

Figura 4.4: Etapa 2 da implementação

Conforme demonstrado na Figura 4.4, observamos a implementação efetiva do plugin Calico sobre o *cluster* Minikube. O processo inicia quando o usuário, através do laptop, executa o comando "minikube start –network-plugin=cni –cni=calico". Esta ação modifica a configuração padrão do CNI no Minikube para adotar o Calico, um *plugin* de rede que suporta políticas de rede e é altamente recomendado para ambientes que exigem segurança.

Ao adotar o Calico, o Minikube passa a ter a capacidade de implementar políticas de rede definidas pelo usuário, criando um ambiente propício para a execução de testes e validações, como os realizados neste estudo. Após a inicialização do servidor, que ativa os serviços do Docker e do Kubernetes da mesma forma que na configuração padrão, a aplicação é iniciada dentro do contexto alterado pelo *plugin* de rede.

Neste cenário configurado com o Calico, é perceptível a presença de "*netpol*" dentro do *cluster* Kubernetes, indicando que as políticas de rede são agora reconhecidas e aplicadas na infraestrutura. Esse requisito é essencial para a validação dos resultados de segurança pretendidos pelo *framework* SARIK.

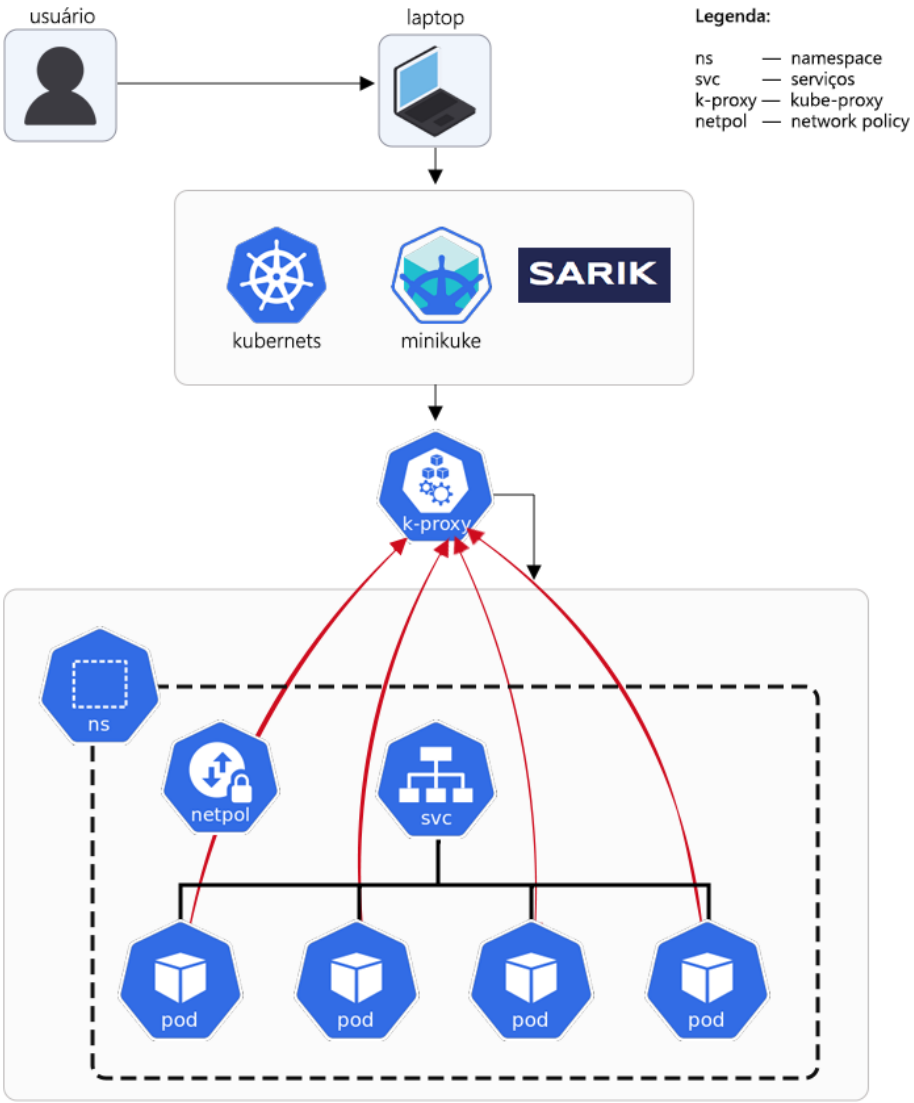

Figura 4.5: Etapa 3 da implementação

Na Figura 4.5, é ilustrada o encontro das etapas anteriores com a implementação ativa do *framework* SARIK, delineando a automação das políticas de rede que foram descritas na Seção 4.3 deste capítulo. Neste estágio, o SARIK executa a automação das políticas de rede, um processo que resulta na configuração automática dessas políticas dentro do *cluster* Minikube. O *framework* aproveita o mapeamento prévio das portas e protocolos utilizados pela aplicação para determinar as regras de acesso e tráfego entre os Pods.

A execução do SARIK transforma a abordagem de segurança do *cluster*, passando de um estado onde todas as vias de comunicação estavam abertas para um regime restritivo, onde apenas o tráfego permitido pelas políticas é autorizado. Isso é alcançado por meio de um conjunto de operações que o SARIK orquestra, a começar pelo levantamento detalhado do ambiente de rede e a identificação de padrões de tráfego normais. Em seguida, o *framework* gera as políticas de rede ('netpols') que correspondem aos requisitos de segurança especificados.

Essas políticas de rede são então aplicadas, restringindo o tráfego para as portas que não são necessárias para a operação regular dos serviços, conforme evidenciado pelas flechas vermelhas na ilustração. Estas flechas simbolizam as restrições impostas pelo SARIK, que bloqueiam o tráfego não autorizado, impedindo comunicações indesejadas ou potencialmente perigosas entre os Pods. Como resultado, qualquer tentativa de acessar ou estabelecer comunicação através de portas não aprovadas é automaticamente negada, aprimorando a postura de segurança do *cluster*.

Este processo não apenas aprimora a segurança, mas também é realizado de maneira a não comprometer a funcionalidade essencial da aplicação. O SARIK foi projetado para garantir que a configuração de segurança não interfira com a operação legítima dos serviços, uma consideração crítica em ambientes de produção onde a disponibilidade e a integridade são tão importantes quanto a segurança.

Ao concluir a automação das políticas de rede, o SARIK efetivamente muda a dinâmica do *cluster* Kubernetes, passando de uma postura aberta e possivelmente vulnerável para uma configurada com defesas proativas e deliberadas. Este nível de controle granular sobre a comunicação entre os Pods é uma adaptabilidade e potência do SARIK como um *framework* de segurança para ambientes Kubernetes, reafirmando sua posição como uma ferramenta para a gestão eficaz da segurança em infraestruturas de nuvem.

4.5 MECANISMO E POLÍTICA NO SARIK: CONSIDERAÇÕES SOBRE A DESASSOCIAÇÃO

A distinção entre "mecanismo" e "política" é um princípio fundamental na ciência da computação e na concepção de sistemas complexos, sobretudo no domínio da segurança cibernética. Este conceito, amplamente promovido por Tanenbaum [50], é importante para o design eficaz e adaptável de *frameworks* de segurança, como o SARIK, especificamente projetado para reforçar a proteção em ambientes Kubernetes.

O "mecanismo" dentro do SARIK é a fundação sobre a qual o *framework* é construído, abrangendo a infraestrutura e os processos técnicos que permitem a execução de *scripts*, automação de tarefas, processamento de dados e integração com Kubernetes. Esta camada técnica é projetada para ser agnóstica em relação às políticas específicas, garantindo uma base sólida e eficiente que suporta uma vasta gama de funções de segurança.

Em contrapartida, a "política" no SARIK envolve a aplicação de regras e diretrizes de segurança específicas, como o gerenciamento de quais portas e protocolos devem ser bloqueados ou permitidos, configurações de rede personalizadas e estratégias de resposta a ameaças. Essas políticas são estabelecidas com base em análises de risco, requisitos operacionais e práticas recomendadas, refletindo as necessidades específicas de segurança de cada *cluster* Kubernetes.

A separação entre mecanismo e política no desenvolvimento do SARIK oferece flexibilidade interessante, permitindo que administradores e usuários ajustem ou modifiquem políticas de segurança conforme necessário, sem a exigência de reconfiguração da infraestrutura subjacente. Esta abordagem não apenas eleva a eficácia do SARIK, mas também assegura sua capacidade de se adaptar rapidamente a novas ameaças ou mudanças no panorama de segurança.

Um exemplo prático dessa separação é evidenciado na automação de políticas de rede pelo SARIK, onde o mecanismo é concebido para ser genérico e abrangente, facilitando a implementação de uma ampla gama de políticas definidas pelos administradores do sistema. Tal design incrementa a resiliência do SARIK frente a mudanças, permitindo a incorporação de políticas adicionais em resposta a novos tipos de ameaças sem necessitar de uma revisão fundamental da sua arquitetura.

Em resumo, a adoção da filosofia de separação entre mecanismo e política, conforme explicado por Tanenbaum, é vital para a estrutura e eficiência do SARIK. Esta estratégia não somente facilita a gestão de segurança, mas também proporciona uma base adaptável que é capaz de enfrentar desafios de segurança em evolução nos ambientes de nuvem. A diferenciação entre mecanismo e política, portanto, emerge como um pilar no design de sistemas seguros, reiterando a relevância dessa distinção para a ciência da computação e a segurança cibernética contemporânea.

4.6 POLÍTICAS DE REDE

Esta seção aborda as políticas de rede na interface de saída (*egress*) e entrada (*ingress*) em um *cluster* Kubernetes, com foco na sua implementação e gerenciamento através do *framework* SARIK. Além disso, será discutido a estrutura dos manifestos YAML que definem essas políticas e como o SARIK automatiza esse processo.

Políticas de rede são fundamentais para a segurança e o gerenciamento eficaz de *clusters* Kubernetes. Elas determinam como os Pods dentro do *cluster* podem se comunicar entre si e com outros *endpoints* de rede. Neste contexto, as políticas de rede podem ser configuradas para controlar tanto o tráfego de saída (*egress*) quanto o tráfego de entrada (*ingress*) para os Pods.

As políticas de rede na interface de saída (*egress*) controlam o tráfego que sai dos Pods em um *cluster* Kubernetes. Essas políticas são cruciais para restringir a comunicação de Pods com recursos externos, limitando assim a superfície de ataque. No Kubernetes, as políticas de *egress* são definidas usando manifestos YAML que especificam as regras para o tráfego de saída. No exemplo do manifesto em Algoritmo 1, uma política de *egress* pode ser configurada para permitir que um Pod se comunique apenas com determinados endereços IP ou portas.

Algoritmo 1 : Manifesto sobre Políticas de rede (*egress*)

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: egress-policy
spec:
 podSelector: {}
 policyTypes:
 - Egress
 egress:
 - to:
  - ipBlock:
    cidr: 192.168.0.0/16
  ports:
  - protocol: TCP
   port: 80
```

As políticas de rede na interface de entrada (*ingress*) controlam o tráfego que entra nos Pods. Essas políticas são essenciais para proteger os Pods de acessos não autorizados e ataques. Similarmente às políticas de *egress*, as políticas de ingress são também definidas através de manifestos YAML. Uma política de *ingress* pode, por exemplo, permitir que apenas determinados endereços IP acessem um Pod específico conforme demonstrado em Algoritmo 2.

Algoritmo 2 : Manifesto sobre Políticas de rede (*ingress*)

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: ingress-policy
spec:
 podSelector: {}
 policyTypes:
 - Ingress
 ingress:
 - from:
  - ipBlock:
    cidr: 10.0.0.0/8
  ports:
  - protocol: TCP
   port: 22
```

Os manifestos "YAML" são arquivos de configuração que descrevem os recursos do Kubernetes, incluindo políticas de rede. Eles são gerados com base nas especificações fornecidas pelo usuário ou pelo sistema e são aplicados ao *cluster* usando o comando "*kubectl apply -f <manifest-file.yaml>*". Este co-

mando informa ao Kubernetes para criar ou atualizar os recursos conforme definido no manifesto.

Algoritmo 3 : Manifesto sobre Políticas de rede na interface de saída (*egress*)

```
1   apiVersion: networking.k8s.io/v1
2   kind: NetworkPolicy
3   metadata:
4    name: block-ports-egress-db-f9d96b9d6-27hzz-22
5    namespace: vote
6   spec:
7    podSelector:
8     matchLabels:
9      app: db
10   policyTypes:
11   - Egress
12   egress:
13   - ports:
14    - protocol: TCP
15     port: 22
16   - to:
17    - podSelector:
18      matchLabels:
19       app: db
```

A estrutura de um manifesto de política de rede é composta por várias seções, cada uma com seu próprio conjunto de campos. O cabeçalho padrão inclui os campos *ApiVersion* e *Kind*, que especificam a versão da API e o tipo de recurso, respectivamente. A seção *Metadata* contém informações como o nome e o *namespace* do POD, que são preenchidos automaticamente com base no ID e no nome do POD, bem como na rede em que estão localizados.

O SARIK desempenha o papel de automação e na eficácia das políticas de rede em *clusters* Kubernetes, e um dos mecanismos centrais para isso é a geração de manifestos automáticos. Os manifestos são arquivos YAML que servem como a espinha dorsal para definir como os Pods interagem entre si e com outros recursos de rede. Um manifesto típico, por exemplo, em Algoritmo 3 é dividido em várias seções principais, começando com o cabeçalho que contém os campos "*apiVersion*" e "*kind*". O campo *apiVersion* especifica a versão da API do Kubernetes que o manifesto utiliza, geralmente sendo "*networking.k8s.io/v1*" para políticas de rede. O campo *kind* define o tipo de recurso que o manifesto representa, que, no caso de políticas de rede, é sempre *NetworkPolicy*.

A seção *spec* é onde a maior parte da lógica de política de rede é definida. Ela contém o campo *podSelector*, que especifica quais PODs a política deve ser aplicada, e o campo *matchLabels*, que permite a seleção de PODs com base em suas etiquetas. A seção *policyTypes* oferece opções de restrição, como "*ingress*" e "*egress*". No contexto do SARIK, a opção "*egress*" é frequentemente utilizada para controlar o tráfego de saída dos PODs.

O SARIK automatiza o processo de preenchimento desses campos através de um mapeamento de-

talhado, que é parte integrante da etapa (3) da Figura 4.1. Este mapeamento envolve a identificação de diversos protocolos e portas que devem ser bloqueados ou permitidos. Essas informações são então inseridas nos campos *egress*, *ports*, *protocol* e *port* do manifesto.

Após a geração dos manifestos, eles são aplicados ao *cluster* usando o comando *kubectl apply -f policies/*. Este é um passo crítico, pois é aqui que as políticas de rede realmente entram em vigor. O comando instrui o Kubernetes a aplicar as configurações definidas nos manifestos, garantindo que as políticas de rede sejam rigorosamente aplicadas aos PODs selecionados.

É importante notar que o SARIK também possui mecanismos para monitorar a eficácia das políticas de rede aplicadas e fazer ajustes conforme necessário. Isso é crucial para manter a segurança da infraestrutura à medida que ela evolui.

Em resumo, o SARIK não apenas automatiza a geração de manifestos, mas também garante que eles sejam aplicados de forma eficaz, contribuindo significativamente para a segurança e a eficiência da infraestrutura de rede em *clusters* Kubernetes. Este processo detalhado de geração e aplicação de manifestos é um dos pilares da robustez do SARIK como um *framework* de segurança de rede.

4.7 KUBE-PROXY

O *kube-proxy* é um componente fundamental no ecossistema do Kubernetes, responsável por gerenciar o tráfego de rede entre os diferentes nós e serviços dentro de um *cluster*. Este elemento trabalha em sinergia com o *plug-in* de rede CNI selecionado para implementar políticas de rede específicas. Nesta seção, discutiremos em detalhes como o *kube-proxy* utiliza regras de *iptables* para efetuar o controle de tráfego e segurança de rede.

No ecossistema do Kubernetes, aliado ao *plug-in* de rede CNI selecionado, são conduzidos diversos procedimentos para direcionar as requisições de políticas de rede específicas dos manifestos gerados pelo SARIK ao *kube-proxy*. No *kube-proxy*, são criadas cadeias e regras que refletem o bloqueio da comunicação. O Algoritmo 4 ilustra um exemplo de configuração de *iptables* implementado ao longo desse processo.

Considerando o exemplo do Algoritmo 4, essas regras têm como finalidade permitir o tráfego de entrada em uma porta TCP específica (443) e aplicar uma marca ao tráfego que chega nessa porta (0x10000/0x10000). A marcação é uma técnica que facilita a identificação e o gerenciamento subsequente do tráfego de rede. A primeira regra, que está delimitada por colchetes [ ], estabelece a porta TCP (443) e a marca atribuída ao tráfego recebido nessa porta. A segunda regra verifica se o tráfego recebido possui a marca definida na primeira regra e, então, permite que o tráfego seja entregue ao processo de origem. A terceira regra especifica um conjunto de endereços IP autorizados a acessar a porta definida na primeira regra e aplica a marca previamente estabelecida. Por fim, a última regra verifica novamente a marca definida na primeira regra e, em seguida, retorna o controle do tráfego para o processo de origem.

No exemplo do Algoritmo 5, este apresenta uma série de regras do *iptables* que desempenham um papel fundamental na segurança e eficiência do gerenciamento de dados em sistemas Linux. Resumidamente,

Algoritmo 4 : Regras de *iptables* no *kube-proxy*

```
[1] -A cali-po-_dUocG07UHnoYGHwzj1q -p tcp -m comment
  --comment "cali:RUtNW8LPCr3T_zE7" -m multiport
  --dports 443 -j MARK --set-xmark 0x10000/0x10000

[2] -A cali-po-_dUocG07UHnoYGHwzj1q -m comment
  --comment "cali:AX7qYuyJPSjHM8J7" -m mark --mark
  0x10000/0x10000 -j RETURN

[3] -A cali-po-_dUocG07UHnoYGHwzj1q -m comment
  --comment "cali:KVjlLm06huMbM5nt" -m set --match-set
  cali40s:CVtTjVoVYfBCPNXq7iYGrlr dst -j MARK
  --set-xmark 0x10000/0x10000

[4] -A cali-po-_dUocG07UHnoYGHwzj1q -m comment
  --comment "cali:5EN0O5PuSQjj41WP" -m mark --mark
  0x10000/0x10000 -j RETURN
```

essas regras desempenham as seguintes funções:

A primeira regra, que está delimitada por colchetes [1], tem como objetivo permitir o tráfego que está relacionado e já estabelecido. Identificada pelo comentário "cali:ZuYBI2b8cXggjkHK", essa regra utiliza o módulo *conntrack* para verificar o estado da conexão dos pacotes, aceitando somente aqueles que possuem os estados "*RELATED*" (relacionado) ou "*ESTABLISHED*" (estabelecido). Ao empregar a ação *ACCEPT*, os pacotes que atendem a essas condições são permitidos e encaminhados para processamento subsequente.

A segunda regra, que está delimitada por colchetes [2], visa bloquear o tráfego considerado inválido. Identificada pelo comentário "cali:HbLByr_5h7-iVibx", essa regra também faz uso do módulo *conntrack* para verificar o estado da conexão dos pacotes, especificamente os pacotes marcados como "*INVALID*" (inválido). Por meio da ação *DROP*, os pacotes considerados inválidos são descartados, evitando sua disseminação na rede.

A terceira regra, que está delimitada por colchetes [3], aplica uma marca (xmark) aos pacotes. Identificada pelo comentário "cali:QuLrKssNqb4cPtof", essa regra utiliza a ação *MARK* em conjunto com o parâmetro *–set-xmark* para definir a marca dos pacotes como 0x0/0x10000. A marcação é uma maneira de atribuir um rótulo aos pacotes, permitindo a aplicação de outras regras ou políticas com base nessa marca específica. Neste cenário, a marca é configurada como 0x0/0x10000, indicando que determinada condição ou critério será aplicado aos pacotes marcados com essa identificação. Essa funcionalidade é valiosa para a implementação de políticas de segurança e roteamento personalizadas, assegurando o tratamento adequado dos pacotes conforme suas marcas.

A quarta regra, que está delimitada por colchetes [4], tem como finalidade bloquear pacotes UDP que utilizam a porta 4789 e estão encapsulados em VXLAN. Pelo comentário "cali:vXfahtOQI3y4FFWb" e "*Drop VXLAN encapped packets originating in workloads*", essa regra faz uso do parâmetro -p udp para

Algoritmo 5 : Regras de *iptables* Calino no *kube-proxy*

```
1 [1] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    ZuYBI2b8cXggjkHK" -m conntrack --ctstate RELATED,
    ESTABLISHED -j ACCEPT
2
3 [2] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    HbLByr_5h7-iVibx" -m conntrack --ctstate INVALID -j DROP
4
5 [3] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    QuLrKssNqb4cPtof" -j MARK --set-xmark 0x0/0x10000
6
7 [4] -A cali-fw-cali56dde0ed24e -p udp -m comment --comment "
    cali:vXfahtOQI3y4FFWb" -m comment --comment "Drop VXLAN
    encapped packets originating in workloads" -m multiport
8 --dports 4789 -j DROP
9
10 [5] -A cali-fw-cali56dde0ed24e -p ipencap -m comment
11 --comment "cali:6H8FrGk4Wk6Ve17w" -m comment --comment
12 "Drop IPinIP encapped packets originating in workloads"
13 -j DROP
14
15 [6] --A cali-fw-cali56dde0ed24e -m comment --comment "cali:-4
    b85r6VZynUeOd4"-j cali-pro-kns.vote
16
17 [7] --A cali-fw-cali56dde0ed24e -m comment
18 --comment "cali:h5PhrImYCuzL6qET" -m comment --comment
19 "Return if profile accepted" -m mark --mark 0x10000/0x10000
20 -j RETURN
21
22 [8] --A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    E4Gsbi4OgMASZZUh" -j cali-pro-ksa.vote.default
23
24 [9] --A cali-fw-cali56dde0ed24e -m comment --comment "cali:0
    vRfCptgGHBh-1l-" -m comment --comment "Return if profile
    accepted" -m mark --mark 0x10000/0x10000 -j RETURN
25
26 [10] --A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    tYr2s6kuv49jm0yz" -m comment --comment "Drop if no
    profiles matched" -j DROP
27
28 [11] +-A cali-fw-cali56dde0ed24e -m comment --comment "cali
    :80ryJuwKsNJyBhFx" -m comment --comment "Start of policies
    " -j MARK --set-xmark 0x0/0x20000
```

especificar o protocolo UDP e o parâmetro –dports 4789 para indicar a porta de destino. Quando identifica pacotes que atendem a esses critérios, a ação *DROP* é aplicada, resultando no descarte desses pacotes. Essa regra é empregada para evitar que pacotes VXLAN encapsulados, originados de *workloads* específicos, prossigam na rede, fornecendo uma camada adicional de segurança.

A quinta regra, que está delimitada por colchetes [5], busca bloquear pacotes encapsulados em IP no IP, que têm origem em *workloads* específicos. Identificada pelos comentários "cali:6H8FrGk4Wk6Ve17w" e "*Drop IPinIP encapped packets originating in workloads*", essa regra utiliza o parâmetro -p ipencap para identificar o protocolo de encapsulamento IP no IP. Quando encontra pacotes que correspondem a essa condição, a ação *DROP* é aplicada, resultando no descarte desses pacotes.

A sexta regra, que está delimitada por colchetes [6], direciona o tráfego para a cadeia "cali-pro-kns.vote". Esta regra não contém um parâmetro específico, como -p, para identificar um protocolo ou condição relacionada ao tráfego. Em vez disso, ela redireciona o tráfego para a cadeia "cali-pro-kns.vote", que provavelmente contém outras regras para processar o pacote.

A sétima regra, que está delimitada por colchetes [7], verifica se o pacote foi aceito pelo perfil de segurança. Isso é feito ao verificar se a marcação (*mark*) do pacote corresponde a 0x10000/0x10000. Se o pacote atender a essa condição, a regra encaminha o pacote de volta (*RETURN*) para o fluxo normal de processamento.

A oitava regra, que está delimitada por colchetes [8], direciona o pacote para a votação padrão do serviço de autorização do Calico (cali-pro-ksa.vote). A votação padrão determinará se o pacote deve ser aceito ou bloqueado com base nas políticas e permissões configuradas para o serviço de autorização.

A nona regra, que está delimitada por colchetes [9], é utilizada para realizar um retorno imediato (*RETURN*) caso o perfil associado ao pacote seja aceito. Essa regra é ativada quando o pacote possui uma marcação (*mark*) correspondente ao valor 0x10000/0x10000. O retorno é uma ação que interrompe a avaliação das regras subsequentes na cadeia, permitindo que o pacote prossiga em seu fluxo normal.

A décima regra, que está delimitada por colchetes [10], é empregada para descartar (*DROP*) pacotes que não correspondem a nenhum perfil de tráfego definido. Essa regra é ativada quando nenhum perfil de tráfego é encontrado para o pacote, ou seja, não há correspondência com as regras anteriores. O descarte do pacote é uma ação que impede que ele prossiga em seu fluxo normal, garantindo que apenas os pacotes que atendem a critérios específicos definidos pelos perfis sejam permitidos.

Por fim, a última regra, que está delimitada por colchetes [11], inicia as políticas de *firewall*, marcando (*MARK*) os pacotes no início do processo. Essa marcação é realizada por meio do conjunto de marcas ("–set-xmark") com o valor 0x0/0x20000. A marcação serve como um indicador para identificar o ponto de partida das políticas de *firewall* aplicadas posteriormente.

No exemplo do Algoritmo 6, é delineado o procedimento de bloqueio de portas e protocolos. A primeira regra, que está delimitada por colchetes [1], tem a finalidade de permitir exclusivamente pacotes com estado de conexão marcado como *RELATED* ou *ESTABLISHED*, isto é, aqueles que estão respondendo a uma conexão já estabelecida ou estão associados a uma conexão existente. A segunda regra, que está delimitada por colchetes [2], por sua vez, atua na rejeição de pacotes inválidos, assegurando que apenas pacotes legítimos sejam submetidos ao processamento subsequente.

Algoritmo 6 : Regras de bloqueio *iptables* Calico no *kube-proxy*

```
1 [1] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    ZWtp0l5QKTr_VPXU" -m comment --comment "Return if
2 policy accepted" -m mark --mark 0x10000/0x10000 -j RETURN
3
4 [2] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:v3-
    lAQEi-QRHNeZ-" -m mark --mark 0x0/0x20000
5 -j cali-po-_Ipke1sZDAZREYnL9Cs8
6
7 [3] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:56
    JZ_F-iaRRshpsl" -m comment --comment "Return if policy
8 accepted" -m mark --mark 0x10000/0x10000 -j RETURN
9
10 [4] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    dv6l1Ow96eEW40m0" -m comment --comment "Drop if
11 no policies passed packet" -m mark --mark 0x0/0x20000 -j DROP
12
13 [5] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    mB6IM8g-g1Nz-OS6" -j cali-pro-kns.vote
14
15 [6] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    RLFlTbD-Wm0owtSk" -m comment --comment "Return if
16 profile accepted" -m mark --mark 0x10000/0x10000 -j RETURN
17
18 [7] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    qKtZvJTKIWtHBwfX" -j cali-pro-ksa.vote.default
19
20 [8] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    NKUtuKuCn6gyOB8m" -m comment --comment "Return if
21 profile accepted" -m mark --mark 0x10000/0x10000 -j RETURN
22
23 [9] -A cali-fw-cali56dde0ed24e -m comment --comment "cali:
    fH_OupKrlVyQyFBo" -m comment --comment "Drop if
24 no profiles matched" -j DROP
```

A terceira regra, que está delimitada por colchetes [3], tem a função de zerar o marcador de pacotes, permitindo, assim, que o pacote seja encaminhado para as regras subsequentes sem ser influenciado pela marcação anterior. As regras quatro e cinco, que estão delimitadas por colchetes [4] e [5], desempenham a ação de descartar pacotes encapsulados VXLAN e IPinIP originados nos *workloads*, respectivamente. Isso ocorre mediante a verificação da aprovação da política e a análise da marcação do pacote. Caso a política não seja aprovada ou a marcação do pacote seja 0x0/0x20000, o pacote é descartado.

As três últimas regras, que estão delimitadas por colchetes [6], [7] e [8], estão relacionadas aos perfis de segurança. A sexta regra direciona os pacotes para um perfil específico, a sétima regra verifica se o perfil foi aceito e a oitava regra descarta pacotes caso nenhum perfil tenha sido aceito. Por fim, a nona e última regra, que está delimitada por colchetes [9], estabelece o início das políticas e define o valor do marcador de pacotes como 0x0/0x20000. Essa marcação serve como um indicador para identificar o ponto de partida das políticas de *firewall* que serão aplicadas posteriormente.

4.8 CONSIDERAÇÕES FINAIS

O capítulo "*Framework* SARIK" foi dedicado à exploração detalhada do *framework* proposto, abordando sua arquitetura, funcionalidades, e aplicações práticas, especialmente no contexto de orquestração de contêineres e gerenciamento de *clusters* com ênfase na segurança. Este capítulo serve como o núcleo técnico da dissertação, demonstrando como o *framework* aborda desafios específicos em orquestração de contêineres e gerenciamento de *clusters* com foco na segurança desse ambiente. A contribuição deste capítulo é significativa, pois apresenta uma solução que pode ser adotada e adaptada em diversos contextos. Com a fundamentação teórica e o *framework* bem estabelecidos, o próximo capítulo, "Experimentos, Resultados e Discussão", se propõem a avaliar a eficácia do *Framework* SARIK através de experimentos e análises estatísticas por meio do *software* Jamovi.

5 EXPERIMENTOS, RESULTADOS E DISCUSSÃO

Este capítulo destina-se a sintetizar os resultados provenientes dos resultados obtidos a partir da avaliação de desempenho das métricas selecionadas: latência, taxa de resposta e taxa de transmissão. O enfoque principal é oferecer uma visão abrangente dos resultados obtidos por meio da análise dos dados experimentais. Vale ressaltar que os resultados são respaldados por um intervalo de confiança de 95% (onde $p < 0,05$). O desempenho de quatro Pods foi cuidadosamente examinado com a finalidade de quantificar a taxa de resposta, latência e taxa de transmissão, em uma série de experimentos realizados em cada um destes Pods. O propósito central desses experimentos é mensurar o tempo de execução de cada métrica, expresso em segundos, e avaliar a conectividade dos Pods, bem como verificar se o tráfego está sendo bloqueado devido à ativação das políticas de rede.

Após uma análise exploratória dos dados, foram identificados valores discrepantes nos grupos de dados 1 e 2. Esses pontos apresentaram valores muito diferentes do comportamento geral da amostra e, após uma revisão adicional, foram identificados como erros de medição. Para evitar que os valores discrepantes afetassem os parâmetros estatísticos, optou-se por excluí-los da análise de acordo Hair et al. [51]. A exclusão desses pontos permitiu que os dados atendesse ao critério de normalidade, melhorando a adequação dos dados a uma distribuição normal. É importante ressaltar que a exclusão desses pontos possibilitou conduzir testes paramétricos mais robustos.

Os dados deste estudo foram analisados com os testes de normalidade e homogeneidade de variância, com a utilização do *software* Jamovi [52]. Algumas variáveis foram transformadas utilizando o logaritmo natural (LOG) para melhor adequação aos critérios de normalidade. A interface intuitiva e recursos estatísticos do Jamovi foram extremamente úteis para a análise estatística dos dados. Este capítulo está estruturado nas seguintes seções: Experimentos, Métrica Latência, Métrica Taxa de Resposta, Métrica Taxa de Transmissão e Discussões e implicações. Cada seção apresenta os resultados e achados de forma detalhada e sistemática, contribuindo para uma compreensão abrangente do desempenho e das implicações práticas do *Framework* SARIK. Todos os dados e resultados estão disponíveis na página do projeto SARIK para consulta e futuras pesquisas.

Além das análises discutidas no texto, tabelas detalhadas e conjuntos de dados estatísticos que fundamentam as discussões deste estudo estão disponíveis no link https://sarik.org/relatorio_tecnico. Cada métrica analisada nas seções subsequentes deste documento é expandida nesses relatórios, proporcionando uma visão detalhada e quantitativa das variáveis estudadas. Esses relatórios técnicos não apenas reforçam os resultados apresentados, mas também servem como uma ferramenta valiosa para pesquisas futuras, permitindo uma exploração mais profunda e um entendimento mais rico das implicações práticas do *framework* SARIK.

5.1 EXPERIMENTOS

Nessa seção é detalhado os experimentos conduzidos em um *cluster* Minikube, com ênfase na distinção entre ambientes seguro e inseguro. No ambiente inseguro, a configuração padrão do Kubernetes foi empregada para implementar uma aplicação de votação, servindo como referência para uma série de experimentos. Em contraste, no ambiente seguro, os mesmos experimentos foram replicados, mas com a integração do *framework* SARIK para aprimorar a segurança dos Pods. Os resultados desses experimentos oferecem uma visão valiosa sobre o impacto e eficácia das políticas de rede geradas pelo SARIK na proteção das aplicações em *clusters* Kubernetes. A análise comparativa entre os cenários seguro e inseguro fornece perspectivas cruciais sobre a relevância das políticas de segurança em tais ambientes, bem como sobre o papel do SARIK na mitigação de riscos e vulnerabilidades associadas.

A solução proposta foi desenvolvida em um ambiente controlado, utilizando um *cluster* Minikube configurado com especificações técnicas substanciais. O *hardware* utilizado inclui um processador Intel Core i5-3360M CPU @ 2.80 GHz, 16GB de memória DDR3, uma interface de rede Intel 82579LM GigabitEthernet e um armazenamento em SSD de 256GB. O sistema operacional empregado foi o Ubuntu Mint.

No desenvolvimento do *framework* SARIK, foi tomada a decisão de implementar o código em linguagem de *shell script*, essa escolha se deve à ampla presença dessa linguagem em sistemas Unix e plataformas de computação em nuvem, essa escolha visa garantir uma compatibilidade abrangente e uma execução simplificada. Conforme ilustrado na Figura 5.1, é possível identificar a presença da *namespace kube-system*, desempenhando um papel fundamental na manutenção de componentes vitais do Kubernetes, incluindo, mais não se limitando ao controlador de rede Calico, CoreDNS, etcd, entre outros. Em contrapartida, a *namespace* denominada “vote” é reservada para a gestão dos Pods que compõem uma aplicação de votação específica. Neste contexto, os Pods db, redis, result, vote e worker estão em execução nessa *namespace*. É imperativo destacar que políticas de rede foram aplicadas a esses Pods, limitando o acesso às portas 22, 443, 7 e 80. Esses Pods interagem entre si através do Kubernetes *Service*, que atua como um mecanismo para expor esses Pods dentro do *cluster*, facilitando a comunicação com outros componentes do ambiente Kubernetes.

Neste estudo, foram realizados experimentos para investigar dois cenários distintos: um ambiente seguro e um ambiente inseguro. O principal objetivo dessas avaliações experimentais era quantificar o tempo necessário para a execução de comandos específicos em cada um desses ambientes. Os dados coletados serviram como base para uma quantificação rigorosa e uma análise detalhada da segurança inerente a cada cenário avaliado. Este enfoque permitiu não apenas uma compreensão aprofundada das vulnerabilidades e riscos associados a cada ambiente, mas também proporcionou percepções valiosas sobre a eficácia das medidas de segurança implementadas, particularmente no contexto do *framework* SARIK. Através desses experimentos, foi possível estabelecer uma base empírica sólida para avaliar o impacto das políticas de segurança em ambientes de cluster Kubernetes, contribuindo assim para o corpo de conhecimento na área de segurança em computação em nuvem.

Nos experimentos conduzidos, o foco recaiu sobre quatro Pods específicos: “db”, “redis”, “result” e “vote”. A partir desses, foram executados comandos como “ping”, “curl”, “apt” e “wget” direcionados

```
NAME                                       READY   STATUS    RESTARTS      AGE
calico-kube-controllers-58497c65d5-9cd9p   1/1     Running   0             88s
calico-node-wsx4z                          1/1     Running   0             88s
coredns-78fcd69978-kw5xj                   1/1     Running   0             88s
etcd-minikube                              1/1     Running   0             108s
kube-apiserver-minikube                    1/1     Running   0             103s
kube-controller-manager-minikube           1/1     Running   0             101s
kube-proxy-cs58s                           1/1     Running   0             88s
kube-scheduler-minikube                    1/1     Running   0             109s
storage-provisioner                        1/1     Running   1 (46s ago)   83s

Namespace "vote"
NAME                      READY   STATUS    RESTARTS   AGE
db-684b9b49fd-45h76       1/1     Running   0          2m3s
redis-67db9bd79b-9x66h    1/1     Running   0          2m3s
result-77f68799c4-wr5p6   1/1     Running   0          2m3s
vote-79787c6c8b-fz44b     1/1     Running   0          2m2s
worker-78b9ff59fc-58nk6   1/1     Running   0          2m2s

NAME                                          POD-SELECTOR   AGE
block-ports-egress-db-684b9b49fd-45h76-22     app=db         53s
block-ports-egress-db-684b9b49fd-45h76-443    app=db         53s
block-ports-egress-db-684b9b49fd-45h76-7      app=db         53s
block-ports-egress-db-684b9b49fd-45h76-80     app=db         53s
block-ports-egress-redis-67db9bd79b-9x66h-22  app=redis      53s
block-ports-egress-redis-67db9bd79b-9x66h-443 app=redis      53s
block-ports-egress-redis-67db9bd79b-9x66h-7   app=redis      53s
block-ports-egress-redis-67db9bd79b-9x66h-80  app=redis      53s
block-ports-egress-result-77f68799c4-wr5p6-22 app=result     53s
block-ports-egress-result-77f68799c4-wr5p6-443 app=result    53s
block-ports-egress-result-77f68799c4-wr5p6-7  app=result     53s
block-ports-egress-result-77f68799c4-wr5p6-80 app=result     53s
block-ports-egress-vote-79787c6c8b-fz44b-22   app=vote       53s
block-ports-egress-vote-79787c6c8b-fz44b-443  app=vote       53s
block-ports-egress-vote-79787c6c8b-fz44b-7    app=vote       53s
block-ports-egress-vote-79787c6c8b-fz44b-80   app=vote       53s
```

Figura 5.1: Diagrama de execução dos Pods
Fonte: Próprio autor

ao Pod “vote”, tanto internamente dentro do *cluster* quanto para servidores externos, incluindo o site do Google para fins de *download* e cópia de arquivos. No cenário inseguro, constatou-se que a comunicação entre os Pods e servidores externos ocorreu sem restrições, evidenciando a vulnerabilidade inerente a uma configuração padrão do Kubernetes. Este aspecto é particularmente preocupante, pois torna os Pods suscetíveis a ataques de mapeamento, que podem ser explorados por agentes maliciosos para coletar informações sobre a arquitetura do *cluster* e, potencialmente, instalar *malwares*, aumentando assim os riscos de segurança.

A Figura 5.2 oferece uma representação gráfica detalhada deste experimento, ilustrando a comunicação entre os Pods e destacando a vulnerabilidade dos mesmos a ataques de mapeamento de Pods. É importante observar que, para mitigar essas vulnerabilidades, políticas de rede podem ser implementadas para restringir tais comunicações, como foi demonstrado neste estudo. Essa medida preventiva é especialmente relevante no contexto do *framework* SARIK, que visa aprimorar a segurança em ambientes de *cluster* Kubernetes.

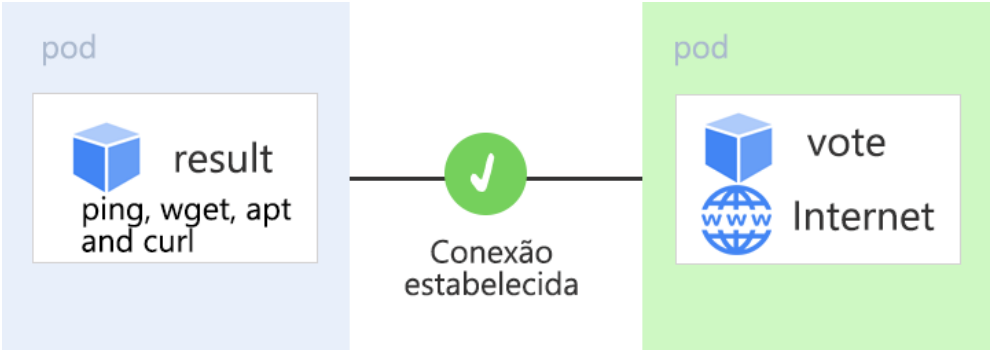

Figura 5.2: Conexão estabelecida

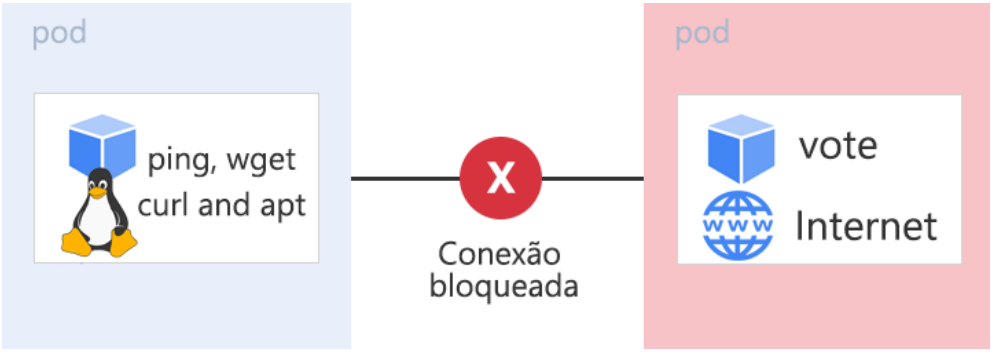

Figura 5.3: Conexão bloqueada
Fonte: Próprio autor

Nos experimentos que se seguiram, foram implementadas políticas de rede para bloquear as portas 22, 443, 7 e 80 em cada um dos Pods que compõem a aplicação de votação. O objetivo era interromper a comunicação entre os Pods e também com servidores externos. Durante os testes no ambiente seguro, observou-se que a comunicação foi efetivamente bloqueada, tanto internamente entre os Pods quanto para o exterior do *cluster*. Comandos como “ping”, “curl”, “apt” e “wget” não foram executados com sucesso, corroborando a eficácia das políticas de rede implementadas.

Este experimento teve como meta principal demonstrar que o bloqueio de portas específicas por meio

	Métrica latência		Métrica taxa de resposta		Métrica taxa de transmissão	
Dia	**Grupo 1 Sem políticas**	**Grupo 2 Com políticas**	**Grupo 1 Sem políticas**	**Grupo 2 Com políticas**	**Grupo 1 Sem políticas**	**Grupo 2 Com políticas**
1	1.5425	21.1753	3.38343	336.25	10.0823	31.831
2	1.62673	21.1511	3.28344	336.283	10.0817	31.8277
3	1.60383	21.2002	5.68344	469.55	10.0833	31.827
4	1.62517	21.1992	3.38344	336.35	10.082	31.829
5	1.5257	21.1454	2.81678	334.917	10.082	31.8533
6	1.69003	21.153	3.28344	336.283	10.0827	31.847
7	1.60157	21.1489	3.31678	334.95	10.083	31.8473
8	1.52497	21.1483	3.28344	334.917	10.085	31.8537
9	1.54467	21.139	3.38345	336.25	10.0817	31.833
10	1.59287	21.1842	3.35011	336.283	10.0813	31.8423

Tabela 5.1: A média dos grupos 1 e 2 das métricas dos experimentos

de políticas de rede é uma estratégia eficaz para reforçar a segurança dos Pods e mitigar potenciais riscos de ataques maliciosos. A Figura 5.3 oferece uma representação gráfica mais detalhada deste cenário, ilustrando a interrupção da comunicação entre os Pods e enfatizando a eficácia das políticas de rede na prevenção de ataques de mapeamento de Pods. Além disso, destaca-se que a implementação dessas políticas também atua como uma barreira eficaz contra a instalação de *malwares* por meio de comandos como "curl", "apt" e "wget".

Ao longo dos experimentos realizados neste estudo, foram elaboradas várias tabelas que apresentam a média dos tempos de execução para cada comando, conforme especificado nas métricas descritas anteriormente. Esses dados foram coletados de forma sistemática durante um período de dez dias consecutivos, utilizando-se de diversos *scripts* dedicados à execução dos testes e para a coleta dos dados resultantes. Para garantir a precisão e a robustez dos resultados, cada comando foi executado em 30 iterações, tanto no ambiente classificado como inseguro quanto no ambiente seguro. Este rigor metodológico assegura a confiabilidade dos dados, permitindo uma análise estatística robusta e conclusões bem fundamentadas.

Neste estudo, foram realizados experimentos com o objetivo de avaliar três métricas fundamentais: taxa de transmissão, latência e taxa de resposta. A métrica da taxa de transmissão foi mensurada utilizando a ferramenta *iperf*, enquanto as métricas de latência e taxa de resposta foram avaliadas por meio da execução dos comandos "apt", "curl", "ping" e "wget" em cada um dos Pods em análise. Para uma avaliação mais rigorosa, o Pod "db" foi utilizado como referência e o comando "apt" foi particularmente enfatizado, conforme ilustrado na Tabela 5.1.

Os dados coletados indicam que o tempo médio de execução dos comandos no Grupo 1 (considerado inseguro), representado na segunda coluna da métrica de latência, é menor em comparação ao Grupo 2 (considerado seguro), apresentado na terceira coluna. Esta diferença pode ser atribuída à implementação de políticas de bloqueio de portas no Grupo 2, que, embora aumentem a segurança do ambiente, também têm um impacto na eficiência da execução dos comandos. Este resultado evidencia a eficácia das políticas de bloqueio implementadas no Grupo 2, reforçando a segurança do ambiente, mas também levanta questões importantes sobre o equilíbrio entre segurança e eficiência que devem ser consideradas em implementações futuras.

Para avaliar o impacto das políticas de rede no desempenho do *cluster*, elaboramos dois cenários distintos, categorizados em dois grupos: o Grupo 1, denominado "Inseguro", onde as políticas de rede não foram aplicadas, e o Grupo 2, designado como "Seguro", no qual as políticas de rede estavam plenamente ativas.

No Grupo 1, os experimentos foram realizados sem a aplicação de políticas de rede, resultando em um ambiente em que todos os Pods tinham acesso irrestrito a todos os recursos de rede disponíveis. Os dados coletados indicaram uma média de tempo de 1,5878 segundos para a métrica de latência, com uma variação temporal que oscilou entre 1 e 21 segundos. Em relação à métrica de taxa de resposta, a média de tempo foi de 3,5167 segundos, com uma variação de 3 a 5 segundos. No que concerne à métrica de taxa de transmissão, a média de tempo registrada foi de 10,0825 segundos, com uma variação de 10 segundos. Estes resultados estão detalhadamente apresentados na Tabela 5.1.

É importante destacar que os experimentos incluíram a execução de ações específicas que simulavam comportamentos atípicos em uma aplicação. Mais especificamente, os Pods foram configurados para realizar tarefas como baixar ou atualizar o sistema operacional sem a autorização do administrador. Isso foi possível através da execução de *scripts* maliciosos que acessavam a internet para enviar e receber dados, ilustrando cenários potenciais de exploração que poderiam representar riscos significativos à segurança do ambiente. Este aspecto sublinha a importância de implementar políticas de rede robustas para mitigar tais vulnerabilidades.

No Grupo 2, onde as políticas de rede estavam plenamente ativadas, observou-se uma restrição significativa na comunicação de rede entre todos os Pods do *cluster*. Os dados coletados indicaram uma média de tempo de 21,1644 segundos para a métrica de latência, com uma variação temporal que se manteve em torno de 21 segundos. Em relação à métrica de taxa de resposta, a média de tempo foi de 338,0741 segundos, com uma variação temporal que variou de 334 a 469 segundos. No que concerne à métrica de taxa de transmissão, a média de tempo registrada foi de 31,8391 segundos, com uma variação de 31 segundos. Estes resultados estão meticulosamente detalhados na Tabela 5.1.

É importante enfatizar que os experimentos no Grupo 2 incluíram a implementação de políticas de rede rigorosas, projetadas para bloquear a comunicação entre os Pods. Essas políticas foram instauradas com o intuito de mitigar comportamentos atípicos previamente identificados como potenciais riscos à segurança do ambiente. Assim, essas medidas de segurança visaram não apenas restringir o acesso dos Pods a determinados recursos de rede, mas também aprimorar a segurança geral do ambiente, validando a eficácia das políticas de rede na mitigação de riscos e vulnerabilidades.

Neste estudo, políticas de rede foram estrategicamente implementadas para cada um dos Pods que integram a aplicação, especificamente os Pods "db", "redis", "result" e "vote". Essas políticas foram concebidas com o intuito de bloquear as portas críticas, como as portas 22, 443, 7 e 80, com o propósito central de aprimorar significativamente a segurança do *cluster* Kubernetes. Tal abordagem é fundamental para mitigar uma série de ameaças em potencial, incluindo ataques de negação de serviço, mineração de criptomoedas, exploração do *cluster*, escalonamento de privilégios e outras estratégias que possam explorar as vulnerabilidades do sistema. Essas políticas de rede servem como um mecanismo crucial para garantir a integridade e a disponibilidade do ambiente, fornecendo uma camada robusta de proteção contra ações mal-intencionadas. A implementação rigorosa dessas políticas reduz significativamente a probabilidade de

brechas de segurança, assegurando uma proteção contínua e eficaz do ambiente em questão.

5.2 MÉTRICA LATÊNCIA

Tabela 5.2: Comparação dos testes estatísticos nos Pods para métrica latência

Comparação dos testes estatísticos para métrica latência no Pod db						
Variáveis	teste Shapiro-Wilk	Transformação (LOG)	teste Levene	Student's t-test	Welch's t-test	p-value.
apt	$p \geq 0{,}05$	Não	$p < 0{,}05$*	Não	Sim	$p < 0{,}05$
curl***	$p \geq 0{,}05$**	Não	$p \geq 0{,}05$	Sim	Não	$p < 0{,}05$
ping	$p \geq 0{,}05$	Não	$p \geq 0{,}05$	Sim	Não	$p < 0{,}05$
wget***	$p \geq 0{,}05$**	Não	$p < 0{,}05$*	Não	Sim	$p < 0{,}05$
Comparação dos testes estatísticos para métrica latência no Pod redis						
Variáveis	teste Shapiro-Wilk	Transformação (LOG)	teste Levene	Student's t-test	Welch's t-test	p-value.
apt***	$p \geq 0{,}05$**	Não	$p \geq 0{,}05$	Sim	Não	$p < 0{,}05$
curl	$p \geq 0{,}05$	Não	$p < 0{,}05$*	Não	Sim	$p < 0{,}05$
ping	$p \geq 0{,}05$	Não	$p \geq 0{,}05$	Sim	Não	$p < 0{,}05$
wget***	$p \geq 0{,}05$**	Não	$p \geq 0{,}05$	Sim	Não	$p < 0{,}05$
Comparação dos testes estatísticos para métrica latência no Pod result						
Variáveis	teste Shapiro-Wilk	Transformação (LOG)	teste Levene	Student's t-test	Welch's t-test	p-value.
apt***	$p \geq 0{,}05$**	Não	$p \geq 0{,}05$	Sim	Não	$p < 0{,}05$
curl***	$p \geq 0{,}05$**	Não	$p \geq 0{,}05$	Sim	Não	$p < 0{,}05$
ping	$p \geq 0{,}05$	Não	$p \geq 0{,}05$	Sim	Não	$p < 0{,}05$
wget***	$p \geq 0{,}05$**	Não	$p < 0{,}05$*	Não	Sim	$p < 0{,}05$
Comparação dos testes estatísticos para métrica latência no Pod vote						
Variáveis	teste Shapiro-Wilk	Transformação (LOG)	teste Levene	Student's t-test	Welch's t-test	p-value.
apt	$p \geq 0{,}05$	Não	$p < 0{,}05$*	Não	Sim	$p < 0{,}05$
curl***	$p \geq 0{,}05$**	Não	$p < 0{,}05$*	Não	Sim	$p < 0{,}05$
ping	$p \geq 0{,}05$	Não	$p \geq 0{,}05$	Sim	Não	$p < 0{,}05$
wget***	$p \geq 0{,}05$**	Não	$p < 0{,}05$*	Não	Sim	$p < 0{,}05$

* Teste de Levene é significativo, sugerindo uma violação da suposição de homogeneidade das variâncias. ** Teste de normalidade significativo indica uma violação da suposição de normalidade. *** Valor discrepante foi excluído para corrigir a normalidade na distribuição dos dados.

A latência desempenha um papel crítico na análise do desempenho de sistemas distribuídos e redes de computadores, pois representa o tempo de atraso entre o envio de um pacote de dados e o recebimento da resposta pelo sistema. Para avaliar a latência, foi empregado a ferramenta *TIME*, aplicada em cada Pod do *cluster*. Foi utilizado expressões regulares para filtrar e extrair o tempo correspondente, expresso em segundos. O *script* empregado para esta medição está publicamente disponível na página de validação do projeto[1], o que confere transparência, precisão e relevância aos resultados obtidos. Esta abordagem metodológica assegura uma avaliação rigorosa do impacto das políticas de rede implementadas no *cluster* Kubernetes em estudo.

Conforme evidenciado na Figura 5.4, observa-se que o Pod "db" no Grupo 1 exibe uma média de

[1] *script* para a métrica de latência: https://github.com/jonathamgg/sarik_validation_graphics

latência inferior a 1,5 segundos para três das variáveis analisadas. No caso da variável PING, a média de latência, ao longo de um ciclo de 30 repetições do experimento, foi de 7,5 segundos. Em contraste, no Grupo 2, a média de latência registrada no mesmo ciclo de testes foi de aproximadamente 20 segundos. Vale ressaltar que as variáveis CURL e WGET, pertencentes ao Grupo 1, manifestaram valores atípicos durante a fase experimental. Este dado é crucial para a interpretação dos resultados, pois sugere que a implementação de políticas de rede no Grupo 2 impacta significativamente a latência, corroborando a eficácia dessas políticas em termos de segurança, embora com um custo em desempenho.

Foi observado que o Grupo 1, que opera sem políticas de rede, apresenta uma latência menor em comparação ao Grupo 2, onde as políticas estão ativas. Esta diferença ilustra o impacto direto das políticas de rede na latência dos pacotes de dados. O aumento da latência no Grupo 2 sugere um compromisso entre segurança e eficiência, destacando a importância de um planejamento cuidadoso para balancear esses dois fatores em ambientes de *cluster* Kubernetes.

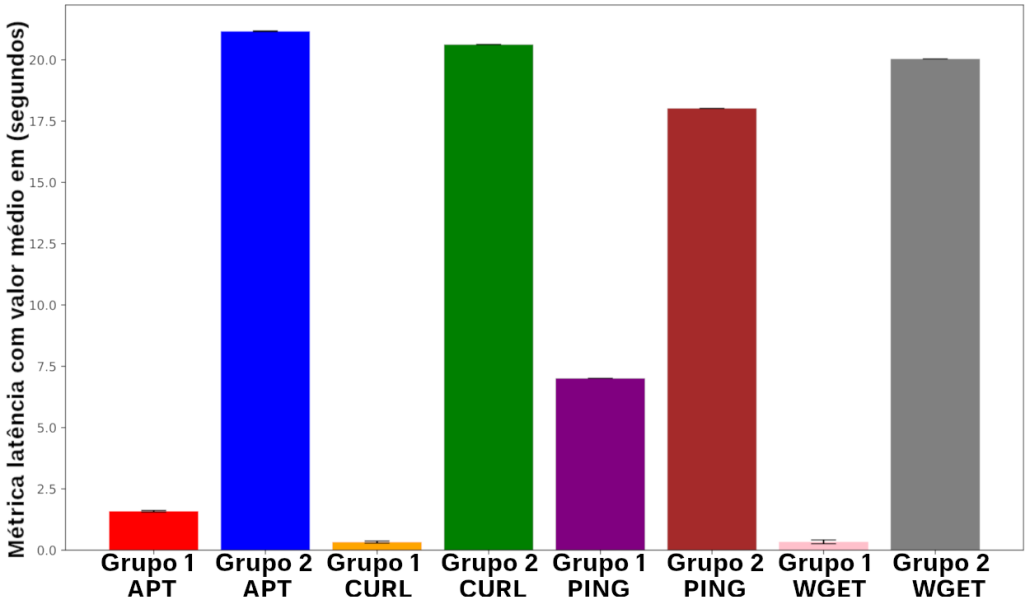

Figura 5.4: Comparação dos valores médios entre os Grupos 1 e 2 com intervalos de confiança de 95% no Pod db para métrica latência. Fonte: Próprio autor

Conforme evidenciado na Tabela 5.2, referente ao Pod “db”, observou-se que apenas as variáveis Curl e Wget apresentaram valores atípicos e uma distribuição não normal ($p < 0,05$). Após a remoção desses valores discrepantes, a distribuição das variáveis foi normalizada, tornando desnecessária qualquer transformação logarítmica. No Teste de Levene para avaliar a homogeneidade das variâncias, somente as variáveis Curl e Ping demonstraram uma distribuição homogênea, sendo, portanto, submetidas ao t-teste. Em contrapartida, as variáveis Apt e Wget não exibiram homogeneidade de variâncias, o que exigiu a aplicação do t-teste com correção de Welch. Importante ressaltar que os resultados dos experimentos apontaram um p-valor abaixo de 0,05 para todas as variáveis, indicando diferenças estatisticamente significativas entre as médias dos Grupos 1 e 2.

Conforme ilustrado na Figura 5.5, que se refere ao Pod “redis”, observa-se que o Grupo 1 exibe uma

média de latência inferior a 1 segundo para três das variáveis testadas e de 7,5 segundos para a variável PING. Estes resultados foram obtidos considerando um ciclo de 30 repetições do experimento. Em contraste, o Grupo 2 apresentou médias de latência variando entre aproximadamente 5 e 17,5 segundos no mesmo ciclo de testes. É importante reforçar que a variável WGET do Grupo 1 apresentou valores atípicos durante a fase experimental.

Segundo os dados apresentados na Tabela 5.2, referente ao Pod "redis", apenas as variáveis Apt e Wget mostraram valores atípicos e uma distribuição não normal, com um valor p inferior a 0,05. Após a eliminação desses *outliers*, a distribuição dos dados alcançou a normalidade, tornando desnecessária qualquer transformação logarítmica. No contexto do teste de Levene, as variáveis Apt, Ping e Wget exibiram uma distribuição homogênea e, por isso, foram submetidas ao t-teste. Em contraste, a variável Curl revelou uma distribuição não homogênea, o que exigiu a aplicação do t-teste com correção de Welch. Os resultados dos experimentos indicaram um valor p inferior a 0,05 para todas as variáveis, sinalizando diferenças estatisticamente significativas entre os grupos em estudo.

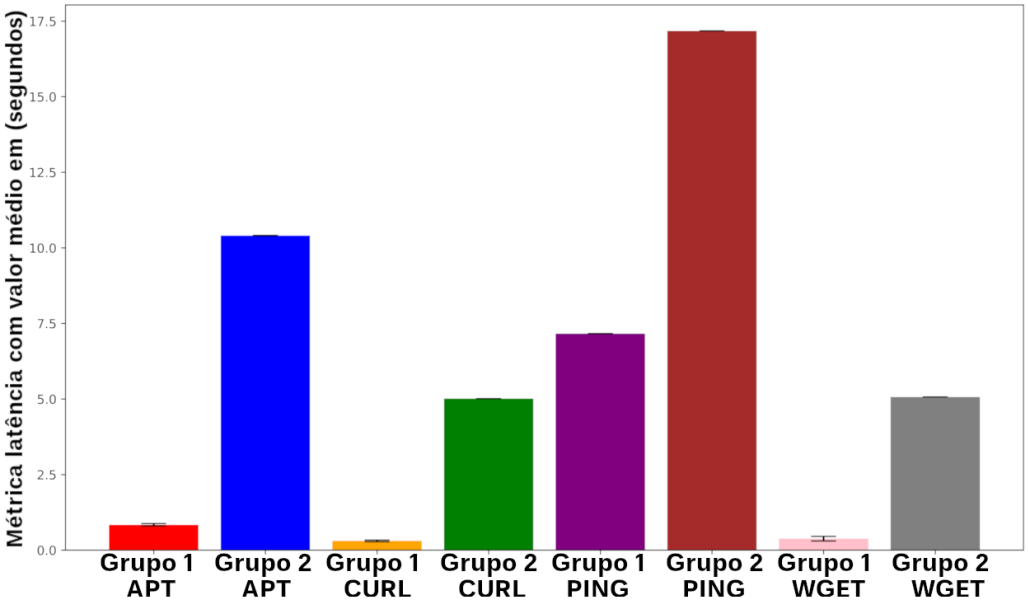

Figura 5.5: Comparação dos valores médios entre os Grupos 1 e 2 com intervalos de confiança de 95% no Pod redis para métrica latência. Fonte: Próprio autor

Na Figura 5.6, associada ao Pod "result", observa-se que o Grupo 1 exibe uma média de latência menor que 1 segundo em três das variáveis testadas e de 7 segundos para a variável PING, com base em um ciclo de 30 repetições do experimento. Em contraste, o Grupo 2 mostrou uma média de latência que variou de 7 a 21,5 segundos no mesmo conjunto de testes. Vale ressaltar que as variáveis CURL e WGET no Grupo 1, bem como a variável APT no Grupo 2, revelaram valores atípicos durante a fase experimental.

Conforme evidenciado na Tabela5.2, associada ao Pod "result", as variáveis Apt, Curl e Wget exibiram valores atípicos e uma distribuição que não é normal ($p < 0,05$). A remoção desses valores discrepantes resultou em uma normalização da distribuição, tornando desnecessária qualquer transformação logarítmica. No teste de Levene, as variáveis Apt, Curl e Ping demonstraram ter uma distribuição homogênea

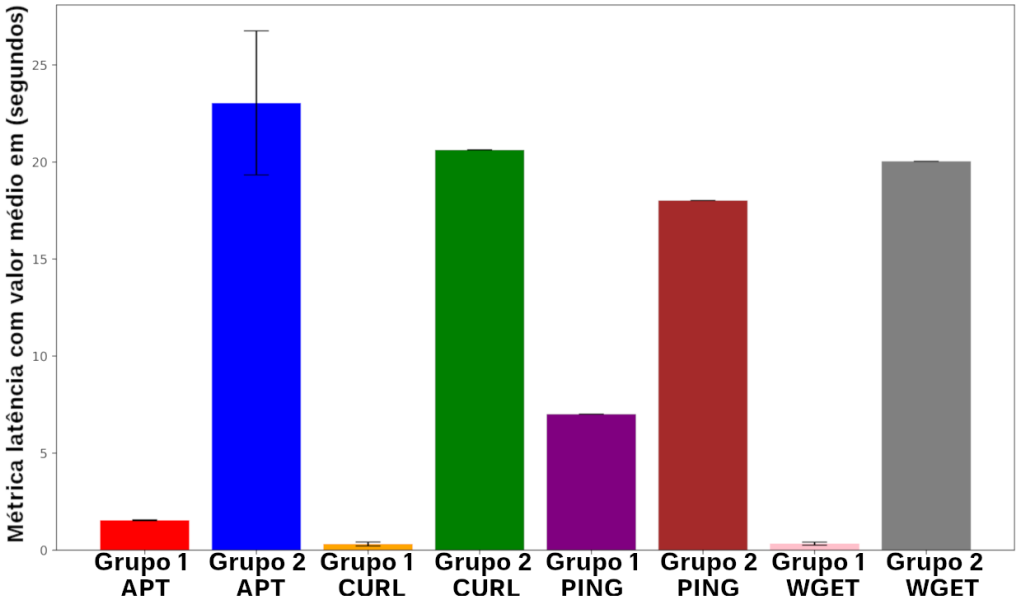

Figura 5.6: Comparação dos valores médios entre os Grupos 1 e 2 com intervalos de confiança de 95% no Pod result para métrica latência. Fonte: Próprio autor

e, portanto, foram submetidas ao t-teste. Por outro lado, a variável Wget apresentou uma distribuição não homogênea, o que levou à aplicação do t-teste com correção de Welch. Os dados obtidos nos experimentos indicaram um valor de p inferior a 0,05 para todas as variáveis, apontando para diferenças estatisticamente significativas.

Conforme ilustrado na Figura 5.7, associada ao Pod "vote", o Grupo 1 exibe uma média de latência menor que 1 segundo em três das variáveis testadas e de 7,5 segundos para a variável PING, com base em um ciclo de 30 repetições do experimento. Em contraste, o Grupo 2 mostrou uma média de latência que variou de aproximadamente 5 a 17,5 segundos no mesmo ciclo de testes. É relevante salientar que a variável WGET do Grupo 1 revelou a presença de valores atípicos durante a fase experimental.

Segundo os dados da Tabela 5.2, associada ao Pod "vote", as variáveis Curl e Wget exibiram valores atípicos e uma distribuição que não se ajusta à normalidade ($p < 0,05$). A remoção desses valores discrepantes resultou em uma normalização da distribuição. A transformação logarítmica não se fez necessária. No Teste de Levene, apenas a variável Ping apresentou uma distribuição homogênea, sendo, portanto, submetida ao t-teste. As demais variáveis - Apt, Curl e Wget - mostraram uma distribuição não homogênea, o que levou à aplicação do t-teste com correção de Welch. Os resultados experimentais indicaram um valor de p inferior a 0,05 para todas as variáveis, apontando para diferenças estatisticamente significativas entre os grupos.

Resumidamente, esta seção, sobre a análise da métrica de latência nos dois grupos estudados revelou *insights* significativos sobre o impacto das políticas de rede no desempenho dos Pods. Ficou evidente que a implementação dessas políticas, embora crucial para a segurança do *cluster*, tem um efeito perceptível no aumento da latência. Este fenômeno foi consistentemente observado em todos os Pods analisados

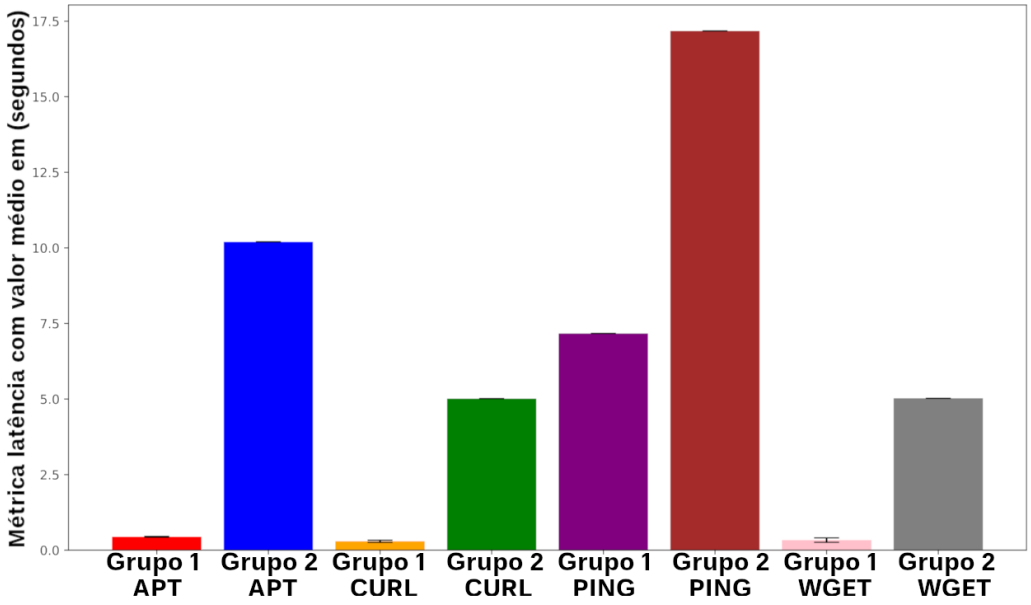

Figura 5.7: Comparação dos valores médios entre os Grupos 1 e 2 com intervalos de confiança de 95% no Pod vote para métrica latência. Fonte: Próprio autor

(“db”, “redis”, “result” e “vote”), reforçando a necessidade de um equilíbrio cuidadoso entre segurança e eficiência operacional. Os resultados estatísticos, com valores de ($p < 0{,}05$), corroboram a significância dessas observações. Com essas considerações em mente, a próxima seção se dedicará à análise da métrica de taxa de resposta, outra variável crítica que pode ser influenciada pela ativação ou desativação de políticas de rede em ambientes de *cluster* Kubernetes.

5.3 MÉTRICA TAXA DE RESPOSTA

Nesta seção, a métrica de taxa de resposta é abordada como um indicador crucial para avaliar o desempenho de sistemas distribuídos e redes de computadores. Essa métrica quantifica a eficiência com que um sistema responde a solicitações externas. Para mensurar essa variável, a ferramenta *DATE* foi empregada em cada Pod que compõe a aplicação em estudo. O procedimento envolveu o registro do horário atual utilizando o comando “*date*” para estabelecer uma marca de tempo inicial. Posteriormente, o comando *DATE* foi executado no respectivo Pod para enviar uma requisição de teste e capturar a resposta. Após essa etapa, o horário atual foi novamente registrado para determinar a marca de tempo final. A diferença entre as marcas de tempo inicial e final fornece o tempo de resposta.

A análise desses resultados permite avaliar não apenas a capacidade de processamento dos servidores, mas também a qualidade da conexão de rede e a identificação de eventuais gargalos que possam impactar negativamente a taxa de resposta. Vale destacar que expressões regulares foram utilizadas para filtrar os dados relevantes do campo de resposta fornecido pelo comando *DATE*. A metodologia empregada para

Tabela 5.3: Comparação dos testes estatísticos para métrica taxa de resposta

Comparação dos testes estatísticos para métrica taxa de resposta no Pod db						
Variáveis	teste Shapiro-Wilk	Transformação (LOG)	teste Levene	Student's t-test	Welch's t-test	p-value
apt***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
curl***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
ping***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
wget***	p < 0,05**	Sim	p < 0,05*	Não	Sim	p < 0,05
Comparação dos testes estatísticos para métrica taxa de resposta no Pod redis						
Variáveis	teste Shapiro-Wilk	Transformação (LOG)	teste Levene	Student's t-test	Welch's t-test	p-value
apt***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
curl***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
ping***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
wget***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
Comparação dos testes estatísticos para métrica taxa de resposta no Pod result						
Variáveis	teste Shapiro-Wilk	Transformação (LOG)	teste Levene	Student's t-test	Welch's t-test	p-value
apt***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
curl***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
ping***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
wget***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
Comparação dos testes estatísticos para métrica taxa de resposta no Pod vote						
Variáveis	teste Shapiro-Wilk	Transformação (LOG)	teste Levene	Student's t-test	Welch's t-test	p-value
apt***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
curl***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
ping***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05
wget***	p < 0,05	Sim	p < 0,05*	Não	Sim	p < 0,05

* Teste de Levene é significativo, sugerindo uma violação da suposição de homogeneidade das variâncias. ** Teste de normalidade significativo indica uma violação da suposição de normalidade. *** Valor discrepante foi excluído para corrigir a normalidade na distribuição dos dados.

essa métrica está disponível para consulta no repositório do projeto[2].

Conforme ilustrado na Figura 5.8, observa-se que o Pod "db" no Grupo 1 registra uma média de tempo de resposta inferior a 2 segundos para as variáveis "apt", "curl", "ping" e "wget", com base em um ciclo de 30 repetições do experimento. Em contraste, o Grupo 2 exibe uma média de tempo de resposta significativamente maior, chegando a aproximadamente 350 segundos para a variável "apt" e em torno de 50 segundos para as demais variáveis, no mesmo ciclo de teste. É importante salientar que a variável "wget" no Grupo 1 e as variáveis "apt", "ping" e "wget" no Grupo 2 apresentaram valores atípicos durante a execução dos experimentos.

Conforme evidenciado na Figura 5.9, o Pod "redis" no Grupo 1 exibe uma média de tempo de resposta inferior a 5 segundos para as variáveis "apt", "curl" e "wget", e 24 segundos para a variável "ping", com base em um ciclo de 30 repetições do experimento. Em contraste, o Grupo 2 mostra uma média de tempo de resposta de aproximadamente 29 segundos para a variável "ping", e entre 9 a 19 segundos para as demais variáveis, no mesmo ciclo de teste. Vale ressaltar que todas as variáveis, tanto no Grupo 1 quanto no Grupo

[2] *Script* da métrica de taxa de resposta: https://github.com/jonathamgg/sarik_validation_graphics

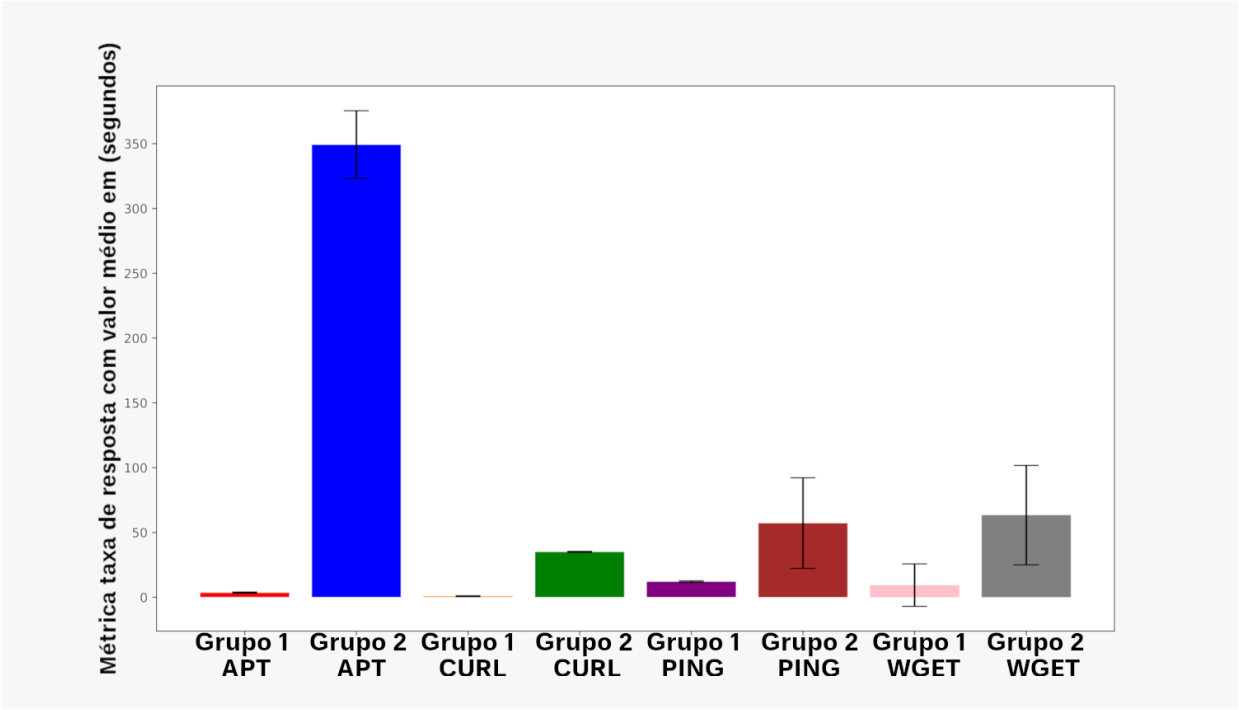

Figura 5.8: Comparação dos valores médios entre os Grupos 1 e 2 com intervalos de confiança de 95% no Pod db para métrica taxa de resposta. Fonte: Próprio autor

2, apresentaram valores atípicos durante a realização dos experimentos.

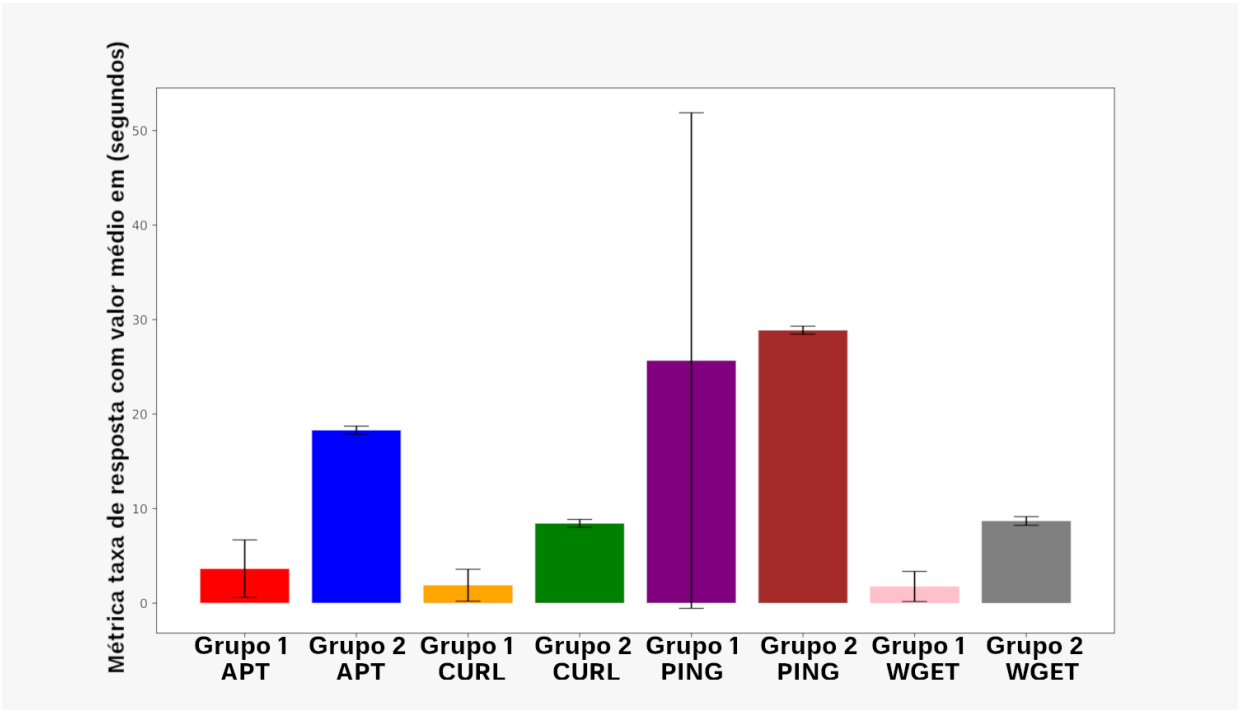

Figura 5.9: Comparação dos valores médios entre os Grupos 1 e 2 com intervalos de confiança de 95% no Pod redis para métrica taxa de resposta. Fonte: Próprio autor

Conforme ilustrado na Figura 5.10, o Pod “result” no Grupo 1 exibe uma média de tempo de resposta inferior a 1 segundo para as variáveis “apt”, “curl” e “wget”, e 25 segundos para a variável “ping”, com

base em um ciclo de 30 repetições do experimento. Em contraste, o Grupo 2 mostra uma média de tempo de resposta de aproximadamente 65 segundos para as variáveis “curl”, “ping” e “wget”, e 340 segundos para “apt”, no mesmo ciclo de teste. É importante notar que as variáveis “apt” e “curl” do Grupo 1 não apresentaram valores atípicos durante o experimento, enquanto as demais variáveis exibiram valores atípicos.

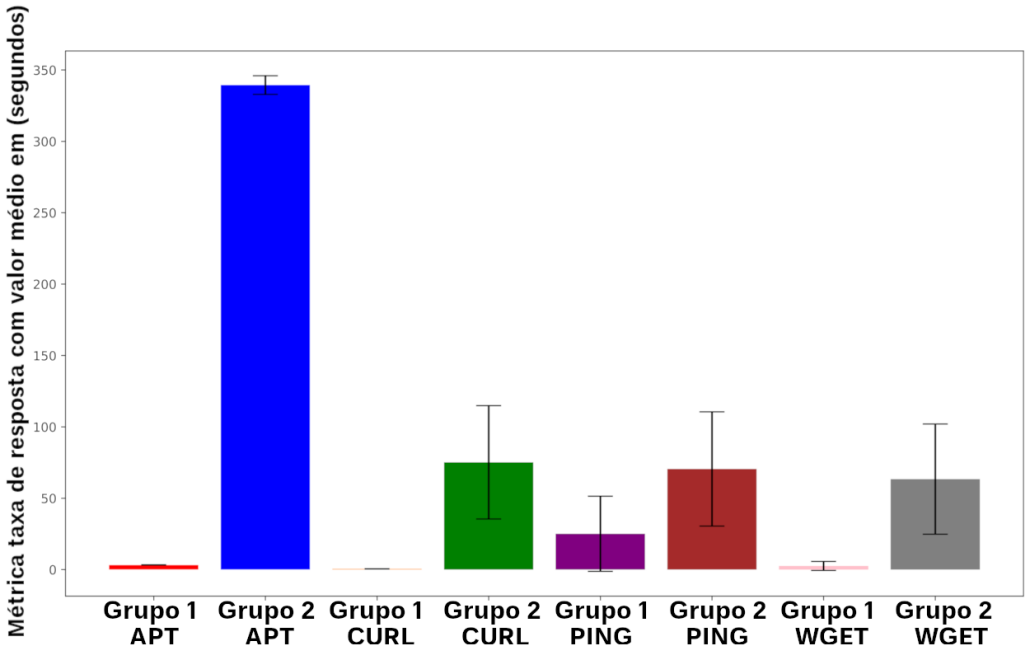

Figura 5.10: Comparação dos valores médios entre os Grupos 1 e 2 com intervalos de confiança de 95% no Pod result para métrica taxa de resposta. Fonte: Próprio autor

Conforme evidenciado na Figura 5.11, o Pod “vote” no Grupo 1 exibe uma média de tempo de resposta que varia de 2 a 11 segundos para as variáveis “apt”, “curl”, “ping” e “wget”, com base em um ciclo de 30 repetições do experimento. Em contraste, o Grupo 2 mostra uma média de tempo de resposta que varia de 9 a 41 segundos para as variáveis “apt”, “curl” e “wget”, e 58 segundos para “ping”, no mesmo ciclo de teste. Vale ressaltar que todas as variáveis apresentaram valores atípicos durante o experimento.

Conforme indicado na Tabela 5.3, relacionada aos Pods “db”, “redis”, “result” e “vote”, todas as variáveis exibiram valores atípicos e uma distribuição não normal ($p < 0,05$). Isso levou à necessidade de realizar uma transformação logarítmica para normalizar a distribuição. No teste de Levene, verificou-se que as variáveis não possuíam uma distribuição homogênea, o que exigiu a aplicação do t-teste com correção de Welch. Os resultados dos experimentos mostraram que o valor de p foi inferior a 0,05 para todas as variáveis em todos os Pods, indicando diferenças estatisticamente significativas.

Em síntese, a análise da métrica de taxa de resposta nos grupos estudados oferece uma visão abrangente do impacto das políticas de rede na eficiência dos Pods. Os dados coletados demonstram que a ativação dessas políticas, embora benéfica para a segurança do *cluster*, resulta em um aumento notável no tempo de resposta. Este fenômeno foi observado de forma consistente em todos os Pods avaliados, corroborando a importância de um planejamento meticuloso para equilibrar segurança e desempenho. Os testes estatísticos, com valores de p abaixo de 0,05, reforçam a relevância dessas descobertas. Com base nesses *insights*,

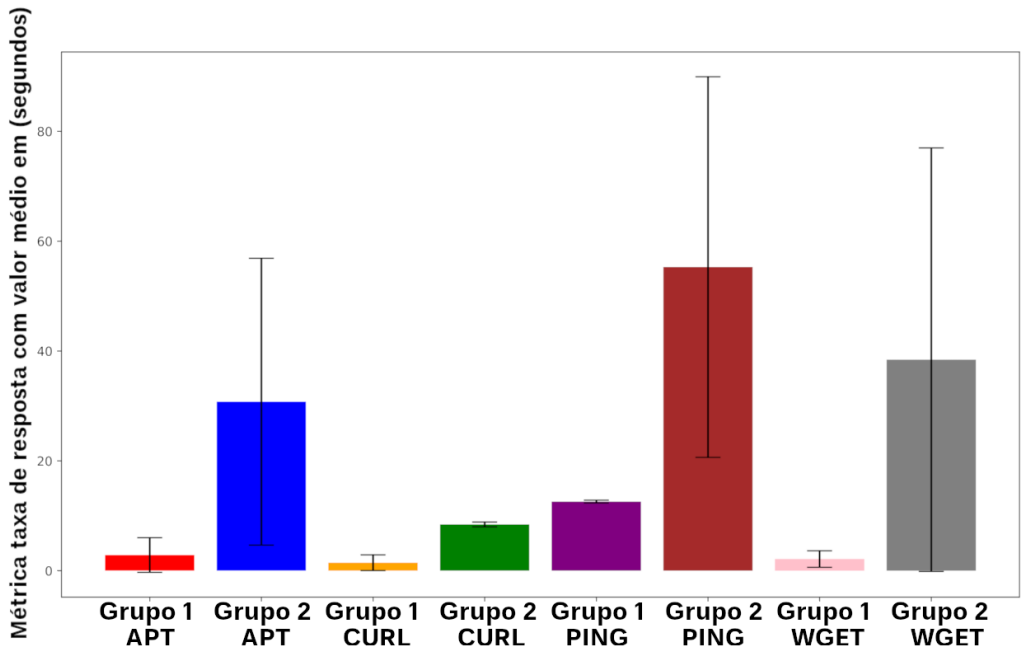

Figura 5.11: Comparação dos valores médios entre os Grupos 1 e 2 com intervalos de confiança de 95% no Pod vote para métrica taxa de resposta. Fonte: Próprio autor

a seção seguinte focará na métrica de taxa de transmissão, outra variável crítica que pode ser afetada pela implementação de políticas de rede em ambientes de *cluster* Kubernetes.

5.4 MÉTRICA TAXA DE TRANSMISSÃO

A taxa de transmissão constitui uma métrica fundamental para avaliar a eficiência na transferência de dados entre dispositivos em uma rede. Neste estudo, a ferramenta *iperf3* foi empregada tanto no *cluster* quanto nos Pods da aplicação para quantificar essa taxa. Para a execução do teste de transmissão, o comando "*time*" é utilizado para registrar o horário de início e término do teste. A diferença entre esses dois registros fornece a informação necessária para calcular a taxa de transmissão. A interpretação dos resultados, que são extraídos por meio de expressões regulares aplicadas aos dados fornecidos pelo *iperf3*, é fundamental para uma compreensão precisa da taxa de transmissão no sistema.

Os resultados exibidos na Tabela 5.4 revelaram que o pressuposto de normalidade para o Pod "redis"foi violado ($p < 0,05$). Em resposta a essa violação, optou-se pela transformação logarítmica dos dados, conforme sugerido por Hair et al.[51]. O teste de Levene indicou a ausência de homogeneidade das variâncias, o que levou à necessidade de aplicar o teste t com correção de Welch. Todos os resultados apontaram diferenças estatisticamente significativas ($p < 0,05$), reforçando a relevância dos achados.

Na Figura 5.12, observa-se que os Pods do Grupo 1 exibem uma média de tempo de transmissão aproximada de 10 segundos, mantendo uma taxa de transmissão de 10 Gbps ao longo de um ciclo de 30 repetições do experimento. Em contraste, o Grupo 2 registrou uma média de tempo de transmissão em

Tabela 5.4: Comparação dos testes estatísticos para métrica taxa de transmissão nos Pods

Variáveis	teste Shapiro-Wilk	Transformação (LOG)	teste Levene	Student's t-test	Welch's t-test	p-value
db	$p \geq 0,05$	Não	$p < 0,05$*	Não	Sim	$p < 0,05$
redis	$p < 0,05$	Sim	$p < 0,05$*	Não	Sim	$p < 0,05$
result	$p \geq 0,05$	Não	$p < 0,05$*	Não	Sim	$p < 0,05$
vote	$p \geq 0,05$	Não	$p < 0,05$*	Não	Sim	$p < 0,05$

* Teste de Levene é significativo, sugerindo uma violação da suposição de homogeneidade das variâncias. ** Teste de normalidade significativo indica uma violação da suposição de normalidade. *** Valor discrepante foi excluído para corrigir a normalidade na distribuição dos dados.

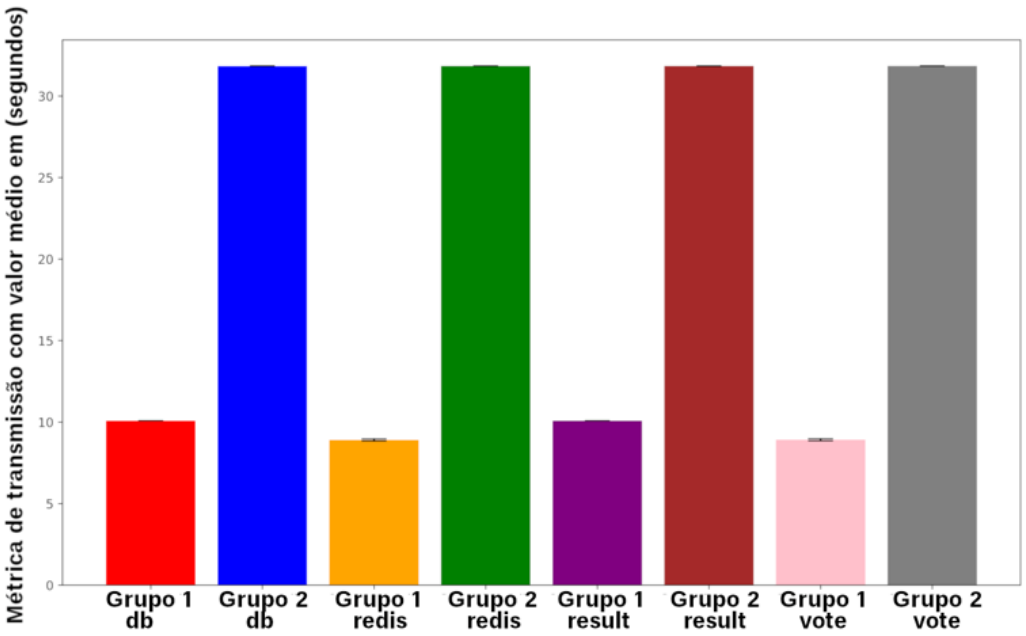

Figura 5.12: Comparação dos valores médios entre os Grupos 1 e 2 com intervalos de confiança de 95% nos Pods da métrica taxa de transmissão. Fonte: Próprio autor

torno de 30 segundos, ainda mantendo a taxa de transmissão de 10 Gbps, no mesmo ciclo de teste. Vale destacar que o Pod "redis" do Grupo 1 apresentou valores atípicos, o que inviabilizou a aplicação do teste-t para esse Pod em particular.

Em síntese, a análise da métrica de taxa de transmissão oferece *insights* valiosos sobre a eficiência da transferência de dados nos diferentes cenários de configuração de rede. Os resultados indicam que, embora a taxa de transmissão se mantenha constante em 10 Gbps para ambos os grupos, o tempo necessário para a transmissão é significativamente maior no Grupo 2, onde as políticas de rede estão ativas. Este aumento no tempo de transmissão sugere que as políticas de rede, embora eficazes na melhoria da segurança, podem ter implicações no desempenho da rede. Os resultados estatísticos, com valores de p inferiores a 0,05, reforçam a relevância dessas observações. Com base nesses achados, a seção seguinte se dedicará a discutir as implicações desses resultados, bem como suas aplicações práticas e teóricas em ambientes Kubernetes.

5.5 ANÁLISE DE DESEMPENHO DO SARIK: IMPACTO NA LATÊNCIA

Em ambientes Kubernetes, onde a agilidade e eficiência são importantes, a avaliação do desempenho de *frameworks* de segurança, como o SARIK, transcende a simples análise de sua eficácia na mitigação de ameaças. Essa avaliação deve considerar de forma meticulosa o impacto sobre os recursos do sistema, com especial atenção para a latência que tais *frameworks* introduzem. Este aspecto é fundamental, pois a latência adicional pode afetar significativamente a performance de aplicações em tempo real. Portanto, uma investigação sobre as métricas de desempenho do SARIK foi realizada, focando particularmente na latência operacional que o *framework* impõe. A análise detalhada busca compreender não apenas como o SARIK aprimora a segurança dentro dos ambientes Kubernetes, mas também avaliar o equilíbrio entre segurança e o impacto sobre o desempenho do sistema, garantindo que a implementação de medidas de segurança não comprometa a eficiência e a agilidade necessárias das aplicações.

A avaliação do SARIK foi conduzida com base em uma série de métricas operacionais selecionadas: taxa de resposta, taxa de transmissão, utilização de CPU, memória, rede e, primordialmente, a latência. Estas métricas foram selecionadas por oferecerem uma visão abrangente do comportamento do SARIK sob diversas condições operacionais, desde o estado de repouso até situações de alta carga ou estresse.

A latência é particularmente crítica em ambientes de nuvem, onde eficiência e velocidade na comunicação entre serviços são vitais. Nos testes realizados, foi observado que o SARIK introduz uma latência significativa no tráfego de rede, refletindo a complexidade das políticas de segurança implementadas. Esse acréscimo de latência, apesar de presente, deve ser ponderado frente aos benefícios de segurança fornecidos pelo *framework*, destacando um equilíbrio entre segurança e desempenho operacional. Foi revelado que, em condições normais de uso, a latência média adicionada pelo SARIK é de aproximadamente 7 a 21 segundos acima do nível de referência, dependendo do experimento conduzido. Este incremento, embora substancial, permanece dentro de limites aceitáveis para muitas aplicações, embora possa ser crítico para outras mais sensíveis ao tempo. Mesmo sob condições de alto estresse, com múltiplas políticas de segurança aplicadas simultaneamente, o aumento da latência para cerca de 17,5 segundos ainda se enquadra dentro de um espectro aceitável para uma ampla gama de cargas de trabalho.

Esta análise abrangeu também outras métricas, como tempo de resposta, essencial para a fluidez de aplicações interativas; utilização de CPU, refletindo o processamento adicional exigido; e consumo de memória, indicativo do *overhead* introduzido pelo *framework*. Contudo, a ênfase recai sobre a latência de rede, dada sua influência direta na performance de aplicações críticas. Os resultados obtidos sublinham a capacidade do SARIK em manter uma operação eficiente, introduzindo um aumento na latência que, apesar de notável, é compensado pela melhoria significativa na segurança, uma troca que algumas aplicações podem considerar vantajosa.

Analisando o desempenho do SARIK, foi notado um equilíbrio entre segurança aprimorada e desempenho consistente. O aumento na latência, embora significativo, é compensado pelos benefícios em segurança, destacando o SARIK como um *framework* efetivo que protege ambientes Kubernetes sem prejudicar excessivamente a performance. Essa análise reforça a necessidade de evolução contínua do SARIK, visando eficiência e menor latência, atendendo às necessidades de aplicações críticas em tempo real sem comprometer a agilidade ou eficácia.

5.6 APLICABILIDADE DO SARIK

A aplicabilidade do *framework* SARIK em diferentes contextos é um aspecto a ser considerado, pois influencia diretamente sua utilidade e eficácia em ambientes variados. Ao longo desta dissertação, exploramos detalhadamente as capacidades do SARIK em mitigar ameaças de segurança em ambientes Kubernetes, destacando sua eficácia em garantir a integridade e a confidencialidade dos dados.

No entanto, é importante reconhecer que os desafios de segurança não são homogêneos e variam significativamente de acordo com o contexto operacional. Portanto, a discussão sobre a aplicabilidade do SARIK deve abranger uma gama diversificada de cenários, incluindo diferentes tipos de infraestruturas de nuvem, requisitos de conformidade regulatória e modelos de ameaças específicos do setor.

Em ambientes empresariais, por exemplo, onde a conformidade com regulamentações de segurança e privacidade de dados é uma prioridade, o SARIK pode ser adaptado para implementar políticas de segurança que garantam a conformidade com padrões como GDPR, HIPAA e PCI DSS. Sua capacidade de automatizar a aplicação e o monitoramento dessas políticas oferece uma solução para empresas que buscam garantir a proteção adequada de seus dados sensíveis.

Além disso, em ambientes de pesquisa e desenvolvimento, onde a agilidade e a inovação são essenciais, o SARIK pode ser adaptado para facilitar a implantação rápida e segura de novas aplicações e serviços. Sua integração com pipelines de CI/CD e ferramentas de automação de infraestrutura permite uma abordagem DevSecOps, onde a segurança é incorporada desde o início do ciclo de desenvolvimento de *software*.

Por fim, em ambientes governamentais e de defesa, onde a segurança cibernética é uma preocupação primordial, o SARIK oferece uma solução flexível e escalável para proteger sistemas críticos contra ameaças internas e externas. Sua capacidade de adaptar-se a requisitos específicos de segurança e interoperar com sistemas legados torna-o uma escolha atraente para agências governamentais e militares que buscam abordagens eficaz para a segurança cibernética.

Em suma, a aplicabilidade do *framework* SARIK transcende os limites de um único contexto operacional e abrange uma gama de cenários e setores. Sua flexibilidade, escalabilidade e eficácia o posicionam como uma ferramenta para organizações em busca de proteção contra ameaças cibernéticas em ambientes kubernetes.

5.7 DISCUSSÕES E IMPLICAÇÕES

Nesta seção, será discutido os resultados alcançados ao longo desta pesquisa, abordando tanto as limitações identificadas quanto os pontos fortes do estudo. Também são identificados os desafios futuros que emergiram durante a investigação. Os principais achados da avaliação de desempenho das métricas — latência, taxa de resposta e taxa de transmissão — são sintetizados para oferecer uma visão abrangente do impacto das políticas de rede em ambientes de *cluster* Kubernetes.

O foco desta análise crítica recai sobre o efeito das regras de segurança tanto em cenários de baixa carga de trabalho quanto em ambientes de alta demanda. Para fundamentar as discussões e extrair *insights*

valiosos, recorreu-se ao uso das ferramentas de monitoramento: Grafana e Prometheus. Estas ferramentas forneceram dados cruciais que enriqueceram a compreensão dos resultados experimentais, permitindo uma avaliação mais precisa e detalhada do desempenho da rede sob diferentes configurações de segurança.

Ao ponderar sobre as implicações desses resultados, torna-se evidente a necessidade de um equilíbrio cuidadoso entre segurança e eficiência operacional. As políticas de rede, embora indispensáveis para a segurança do sistema, podem afetar negativamente o desempenho, como demonstrado pelas métricas analisadas. Portanto, a implementação de tais políticas deve ser feita de forma cuidadosa, considerando o contexto operacional e as necessidades de desempenho do sistema.

```
jonathan@jonathan-HP:~/monitoring$ minikube service list
|-------------|-------------------|--------------------|---------------------------|
|  NAMESPACE  |       NAME        |    TARGET PORT     |            URL            |
|-------------|-------------------|--------------------|---------------------------|
| default     |kubernetes         |No node port        |
| kube-system |kube-dns           |No node port        |
| kube-system |kube-state-metrics |No node port        |
| kube-system |metrics-server     |No node port        |
| monitoring  |grafana            |               3000 |http://192.168.39.131:32000|
| monitoring  |prometheus-service |               8080 |http://192.168.39.131:30000|
| vote        |db                 |No node port        |
| vote        |redis              |No node port        |
| vote        |result             |result-service/5001 |http://192.168.39.131:31001|
| vote        |vote               |vote-service/5000   |http://192.168.39.131:31000|
|-------------|-------------------|--------------------|---------------------------|
```

Figura 5.13: Prometheus e Grafana
Fonte: Próprio autor

Neste contexto, é importante destacar que a configuração padrão do Minikube desativa todos os complementos ("*addons*"), o que pode representar um obstáculo para a análise de dados por meio de ferramentas de monitoramento como Grafana e Prometheus. Para contornar essa limitação e assegurar uma análise e discussão mais sólida, foi necessário habilitar os complementos *metrics-server* e *pod-security-policy*. Isso foi realizado através dos comandos *minikube addons enable metrics-server* e *minikube addons enable pod-security-policy* demonstrado na Figura 5.13.

Essas ferramentas de monitoramento de tráfego são fundamentais para uma avaliação abrangente do desempenho do sistema, pois fornecem métricas detalhadas sobre o consumo de recursos como CPU, memória e rede no *cluster*. A habilitação desses complementos, não apenas enriquece a análise dos dados, mas também oferece *insights* valiosos para a otimização do ambiente de *cluster*, equilibrando eficazmente as necessidades de segurança e desempenho.

A utilização dessas ferramentas de monitoramento, aliada aos resultados experimentais, contribui para uma compreensão mais profunda das implicações práticas das políticas de rede em ambientes de *cluster* Kubernetes. Isso reforça a importância de um planejamento cuidadoso e de uma implementação criteriosa dessas políticas, especialmente em cenários que exigem alta disponibilidade e eficiência.

Neste estudo, uma *namespace* denominada "*monitoring*" foi criada especificamente para alojar dois Pods cruciais para o monitoramento do experimento: Grafana e Prometheus, como ilustrado na Figura 5.13. O Pod Prometheus assume a função de coletar dados das métricas do *cluster*, servindo como uma fonte

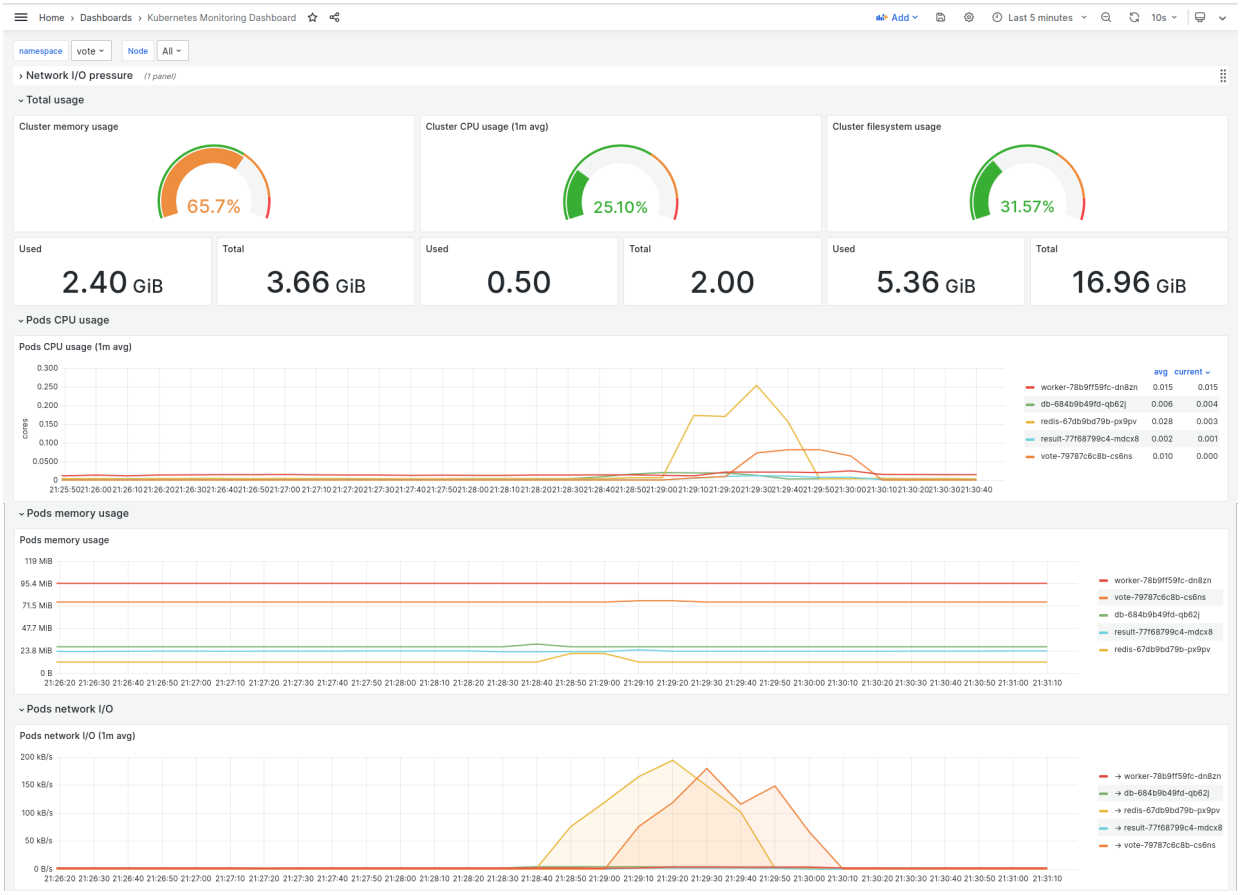

Figura 5.14: Dashboard grafana.
Fonte: Próprio autor

confiável de informações sobre o desempenho do sistema. Por outro lado, o Pod Grafana é encarregado de apresentar esses dados de forma visualmente intuitiva e acessível.

Para aumentar a eficácia dessa abordagem de monitoramento, foi elaborado um *dashboard* modelo no Grafana, permitindo uma visualização otimizada dos dados coletados. Este *dashboard*, representado na Figura 1, foi projetado para fornecer uma visão abrangente das métricas relevantes, facilitando assim a análise e interpretação dos resultados experimentais.

A integração dessas duas ferramentas de monitoramento, Grafana e Prometheus, dentro de uma *namespace* dedicada, não apenas simplifica a gestão dos recursos de monitoramento, mas também contribui para uma análise mais precisa e detalhada. Isso é particularmente útil para avaliar o impacto das políticas de rede e outras configurações de segurança no desempenho geral do *cluster* Kubernetes.

Neste estudo, os comandos curl e wget foram utilizados para coletar dados da internet, focando nas métricas de latência e taxa de resposta. Para complementar essa análise, o comando *traceroute* foi empregado com o objetivo de determinar o número de saltos de rede necessários para alcançar o domínio de destino. Essa informação é vital para entender a eficiência da rota de rede e seu impacto nas métricas estudadas.

Contudo, é importante mencionar que, devido a restrições técnicas que impediram a instalação das ferramentas "*traceroute*" ou "mtr" (*My TraceRoute*) nos Pods "db" e "result", esses Pods específicos não puderam ser avaliados nesse aspecto. Apesar dessa limitação, a discussão e as conclusões aqui apresentadas

são extensíveis e podem ser aplicadas a outros Pods não avaliados, como os Pods “redis” e “vote”. Essa generalização é possível porque os princípios e métodos de análise empregados são amplamente aplicáveis a diferentes configurações de Pod e cenários de rede dentro de um *cluster* Kubernetes.

```
traceroute to www.google.com (142.251.129.100), 30 hops max, 46 byte packets

[1]  192-168-39-131.kubernetes.default.svc.cluster.local (192.168.39.131)
0.010 ms  0.009 ms  0.008 ms
[2]  jonathan-HP.local (192.168.122.1)  0.301 ms  0.124 ms  0.115 ms
[3]  _gateway (192.168.100.1)  0.747 ms  0.599 ms  0.555 ms
[4]  *  *  *
[5]  100.120.68.108 (100.120.68.108)  5.053 ms  100.120.67.180 (100.120.67.180)
5.047 ms  5.150 ms
[6]  100.120.21.85 (100.120.21.85)  4.897 ms  100.120.21.79 (100.120.21.79)
7.757 ms  100.120.21.85 (100.120.21.85)  4.863 ms
[7]  100.120.23.69 (100.120.23.69)  27.228 ms  100.120.26.67 (100.120.26.67)
20.607 ms  100.120.31.157 (100.120.31.157)  24.641 ms
[8]  100.120.20.182 (100.120.20.182)  28.057 ms  100.120.26.190 (100.120.26.190)
24.244 ms  100.120.31.134 (100.120.31.134)  20.532 ms
[9]  72.14.198.152 (72.14.198.152)  24.869 ms  201.10.242.247 (201.10.242.247)
25.122 ms  24.872 ms
[10]  *  74.125.243.65 (74.125.243.65)  26.657 ms  28.055 ms
[11]  209.85.143.205 (209.85.143.205)  21.479 ms  24.758 ms
216.239.46.49 (216.239.46.49)  21.913 ms
[12]  gru14s30-in-f4.1e100.net (142.251.129.100)  21.440 ms  21.772 ms
216.239.46.49 (216.239.46.49)  21.682 ms
```

Figura 5.15: Traceroute no Pod redis
Fonte: Próprio autor

Os resultados da análise de rota, ilustrados nas Figuras 5.15 e 5.16, fornecem um panorama detalhado do caminho percorrido pelos pacotes de dados. Inicialmente, os pacotes foram direcionados para endereços IP dentro da rede local do Kubernetes, o que sugere que eles passaram por nós internos do *cluster*. Posteriormente, os pacotes foram encaminhados para roteadores locais e *gateways* padrão da rede local.

É importante notar que, durante essa trajetória, alguns saltos não retornaram respostas. Essa ausência de resposta pode ser atribuída a configurações específicas em roteadores ou *firewalls* que impedem o retorno de pacotes ICMP, comumente usados em diagnósticos de rede.

Após ultrapassar a rede local, os pacotes entraram na rede de um provedor de serviços ou provedor de internet, passando por vários roteadores intermediários. Novamente, em um desses saltos, não houve resposta, o que pode ser devido às mesmas razões anteriormente mencionadas.

Finalmente, os pacotes alcançaram o destino final, o domínio www.google.com, após passar por diversos roteadores e nós intermediários. Esse percurso ilustra a complexidade e a quantidade de dispositivos envolvidos na transmissão de dados, mesmo para uma simples solicitação a um domínio amplamente acessível. A análise dessas rotas é fundamental para entender como as diferentes camadas da rede e suas configurações podem impactar as métricas de desempenho estudadas neste trabalho.

Os dados coletados revelam que o número total de saltos até o destino foi de 12 e 13, para os respectivos Pods avaliados. Essa informação é particularmente relevante quando consideramos que as métricas de latência e taxa de resposta mostraram uma variação de tempo médio inferior a 1 segundo. Isso ocorreu em

um ambiente de rede com uma conexão de 400 Mbps de *download* e 200 Mbps de *upload*, o que sugere uma eficiência notável na transmissão de dados dentro desse *cluster*.

Além disso, o monitoramento contínuo por meio do Grafana indicou que o uso de CPU e memória nos Pods avaliados permaneceu estável durante a execução do experimento. Essa estabilidade é um indicativo positivo da eficiência da infraestrutura de rede e dos recursos computacionais empregados.

Esses resultados não apenas fornecem *insights* valiosos sobre a infraestrutura de rede utilizada, mas também destacam a eficiência na transmissão de dados entre a origem e o destino. Essas observações são cruciais para entender como a infraestrutura de rede e as políticas de segurança podem impactar o desempenho em ambientes de *cluster* kubernetes.

```
traceroute to www.google.com (142.250.219.4), 30 hops max, 46 byte packets

[1]  192-168-39-131.kubernetes.default.svc.cluster.local (192.168.39.131)
0.006 ms  0.006 ms  0.005 ms
[2]  jonathan-HP.local (192.168.122.1)  0.156 ms  0.202 ms  0.142 ms
[3]  _gateway (192.168.100.1)  0.654 ms  0.837 ms  1.696 ms
[4]  *  *  *
[5]  100.120.68.108 (100.120.68.108)  4.830 ms  100.120.68.106 (100.120.68.106)
5.977 ms  100.120.71.154 (100.120.71.154)  3.693 ms
[6]  100.120.18.199 (100.120.18.199)  5.485 ms  177.2.210.53 (177.2.210.53)
21.800 ms  100.120.18.201 (100.120.18.201)  4.522 ms
[7]  100.120.23.67 (100.120.23.67)  19.498 ms  100.120.23.69 (100.120.23.69)
20.107 ms  100.120.22.206 (100.120.22.206)  25.430 ms
[8]  100.120.25.64 (100.120.25.64)  28.710 ms  100.120.20.240 (100.120.20.240)
42.026 ms  100.120.25.62 (100.120.25.62)  28.727 ms
[9]  72.14.198.152 (72.14.198.152)  24.475 ms  20.694 ms  201.10.242.247
(201.10.242.247) 30.837 ms
[10]  74.125.243.65 (74.125.243.65)  25.074 ms  74.125.243.1 (74.125.243.1)
27.314 ms  26.561 ms
[11]  209.85.251.5 (209.85.251.5)  29.130 ms  209.85.250.243 (209.85.250.243)
28.443 ms  216.239.56.46 (216.239.56.46)  34.628 ms
[12]  209.85.251.5 (209.85.251.5)  28.446 ms  108.170.245.141 (108.170.245.141)
28.963 ms  209.85.251.5 (209.85.251.5)  28.735 ms
[13]  gru14s27-in-f4.1e100.net (142.250.219.4)  24.919 ms  142.250.227.231
(142.250.227.231) 25.498 ms  216.239.54.143 (216.239.54.143)  25.919 ms
```

Figura 5.16: Traceroute no Pod vote
Fonte: Próprio autor

Em um cenário com bloqueio de portas e protocolos, observou-se uma variação de tempo médio inferior a 5 segundos para as métricas de latência e taxa de resposta. Essa variação representa um retardo de aproximadamente 4 segundos em comparação com o primeiro cenário, que não tinha bloqueios. Este retardo pode ser atribuído às características do protocolo TCP, que é orientado à conexão. Mesmo na presença de regras de bloqueio, o TCP tenta garantir a entrega de pacotes, seja por meio de retransmissões ou esperando um tempo determinado até que a solicitação seja concluída ou descartada.

Esse aumento no tempo de latência e na taxa de resposta, embora modesto, destaca a importância de considerar o impacto das políticas de segurança na performance da rede. Mesmo assim, é relevante notar que o uso de CPU e memória nos Pods avaliados permaneceu estável durante a execução do experimento, conforme monitorado pelo Grafana. Isso sugere que, apesar do aumento na latência e na taxa de resposta,

a estabilidade e a eficiência dos recursos do sistema não foram comprometidas.

Os resultados obtidos com o uso do Iperf3, conforme ilustrado na Figura 5.12, oferecem *insights* valiosos sobre o impacto das políticas de segurança em ambientes de *cluster* Kubernetes. No Grupo 1, onde não havia regras de bloqueio, a taxa de transmissão foi constante em 10 Gbps, com uma média de tempo de transmissão de 10 segundos. Embora isso possa indicar alta eficiência, também sugere uma vulnerabilidade a tráfego não autorizado, já que não há barreiras para a transmissão de dados.

Em contraste, o Grupo 2, com regras de bloqueio ativas, apresentou uma taxa de transmissão estável, mas com tempos de transmissão significativamente mais longos, com uma média de 30 segundos. Isso é atribuído à necessidade de retransmissão de pacotes e ao tempo de espera até que ocorra um *timeout*, evidenciando a eficácia das políticas de rede em restringir tráfego indesejado.

Esses resultados, demonstrados na Figura 5.12, sublinham a importância crítica de implementar políticas de segurança de rede cuidadosamente planejadas. Elas não apenas influenciam a taxa de transmissão, mas também contribuem para uma comunicação mais segura entre os dispositivos no *cluster*. Este equilíbrio entre segurança e eficiência é crucial para a operação otimizada de ambientes de *cluster* Kubernetes.

Certamente, as diferenças nas distribuições Linux utilizadas nos diversos Pods podem ter um impacto significativo nos resultados do experimento. A distribuição Alpine Linux é conhecida por sua simplicidade e eficiência, mas pode não ter todas as otimizações e recursos disponíveis em distribuições mais robustas como o Debian. Isso pode afetar várias métricas, incluindo latência, taxa de resposta e taxa de transmissão.

Por exemplo, configurações de *kernel* e *drivers* podem variar entre as distribuições, afetando o desempenho da rede e a eficiência na utilização dos recursos do sistema. Além disso, as otimizações de compilação e as versões de *software* podem ser diferentes, o que pode resultar em variações nos tempos de execução.

As configurações de rede e as políticas de segurança também podem variar, o que pode afetar a forma como os pacotes são roteados ou filtrados, impactando assim as métricas de desempenho. O gerenciamento de recursos e a carga do sistema também são fatores que podem introduzir variações nos resultados.

Portanto, é crucial ter em mente essas possíveis limitações ao interpretar os resultados do experimento. Cada distribuição tem suas próprias vantagens e desvantagens, e a escolha da distribuição pode ser um fator crítico que afeta o desempenho e a segurança do sistema em um ambiente de *cluster* Kubernetes.

A escolha da distribuição Linux em um ambiente Kubernetes tem implicações diretas no desempenho, segurança e eficiência dos aplicativos e processos em execução. Foi observado que o grupo 1, operando sem restrições, apresentou um desempenho distinto em comparação com o grupo 2, onde políticas de segurança estavam em vigor.

A eficiência dos *scripts*, a velocidade da rede e outros fatores podem influenciar os tempos de execução. No entanto, as políticas de segurança também desempenham um papel significativo. No caso do grupo 2, as políticas de segurança implementadas através do *kube-proxy* afetaram o tempo de execução dos comandos em ambas as distribuições, Alpine e Debian. Isso destaca a importância de um planejamento cuidadoso e de testes rigorosos ao implementar políticas de segurança em um ambiente Kubernetes.

O uso do *framework* SARIK neste contexto permite uma análise mais aprofundada de como diferentes

configurações e políticas podem afetar o desempenho e a segurança. A análise e os resultados obtidos fornecem *insights* valiosos para equipes de DevOps ou administradores de sistema que buscam equilibrar desempenho e segurança em ambientes de contêineres complexos.

O impacto da escolha da distribuição do sistema operacional em ambientes Kubernetes é notável, especialmente quando se consideram as diferenças entre distribuições minimalistas como o Alpine Linux e distribuições mais completas como o Debian. No caso do Alpine Linux, a eficiência e a leveza são características marcantes, mas a ausência de determinados recursos ou bibliotecas pode levar a variações nos tempos de execução de comandos específicos.

Por outro lado, o Debian, sendo uma distribuição mais completa e abrangente, oferece uma gama mais ampla de recursos e bibliotecas. Isso pode resultar em diferenças nos tempos de execução, já que a distribuição é potencialmente mais pesada e complexa. As bibliotecas e dependências adicionais presentes no Debian podem também contribuir para variações nos tempos de execução.

Essas observações reforçam a importância de uma análise cuidadosa na escolha da distribuição do sistema operacional, especialmente em ambientes Kubernetes, onde o desempenho e a eficiência são críticos. A seleção apropriada pode impactar não apenas o desempenho, mas também a segurança e a manutenção do sistema.

A análise dos resultados obtidos neste estudo evidencia discussões e implicações significativas, particularmente no que diz respeito à implementação de regras de *firewall* em um *cluster* Kubernetes. Essas regras, embora cruciais para a segurança do sistema, podem ter impactos colaterais que afetam o desempenho e o consumo de recursos.

Um aspecto crítico a ser considerado é o impacto no consumo de recursos do *cluster*, como a CPU e a memória. A implementação de regras de *firewall* pode aumentar a carga de processamento, uma vez que cada pacote de dados que passa pela rede deve ser inspecionado e possivelmente filtrado. Esse aumento na carga de processamento pode levar a um maior consumo de CPU, o que, por sua vez, pode afetar negativamente o desempenho geral do sistema. Adicionalmente, essas regras podem exigir mais memória para armazenar informações de estado e contextos de conexão, o que também pode impactar o desempenho do sistema.

Dada a complexidade dessas variáveis, torna-se imperativo realizar testes rigorosos e ajustes cuidadosos para assegurar que as regras de *firewall* sejam não apenas eficazes em termos de segurança, mas também eficientes em termos de consumo de recursos. A otimização desses parâmetros pode ajudar a alcançar um equilíbrio entre segurança e desempenho, permitindo que o sistema opere de forma eficaz sem comprometer a segurança.

Outra implicação relevante reside no potencial atraso que as regras de *firewall* podem introduzir no processamento das solicitações de rede. A inspeção e o filtro do tráfego de rede com base nas regras estabelecidas podem resultar em latência adicional. Embora esse atraso possa ser negligenciável em muitos cenários, em ambientes que requerem baixa latência, como sistemas de tempo real ou aplicações de comunicação sensíveis ao tempo, essa latência adicional pode ter um impacto substancial no desempenho e na experiência do usuário. Portanto, um equilíbrio cuidadoso deve ser mantido entre a segurança proporcionada pelas regras de *firewall* e a latência que é aceitável para a aplicação específica.

Adicionalmente, a escalabilidade e a manutenção das regras de *firewall* emergem como considerações críticas. À medida que o *cluster* Kubernetes se expande, tanto em termos de número de Pods quanto de nós, a gestão dessas regras pode se tornar cada vez mais complexa. A adição ou remoção de regras em um ambiente em escala pode ser desafiadora e requer uma estratégia bem pensada para evitar conflitos, inconsistências e para assegurar a conformidade com as políticas de segurança. A utilização de ferramentas automatizadas para a gestão de regras e a adoção de boas práticas em gerenciamento de *firewall* são imperativas para manter a segurança e a integridade do sistema ao longo do tempo.

Portanto, a necessidade de reconhecer e mitigar uma variedade de riscos de segurança em Pods é imperativa. Esses riscos englobam uma série de fatores, desde a seleção criteriosa do sistema operacional até o uso de imagens de contêiner comprometidas, a criação de *namespaces* individuais e o risco de escalonamento de privilégios dentro do *cluster*, entre outras ameaças potenciais.

Conforme destacado por Shamim et al.[53] em sua revisão sistemática, que foca nas melhores práticas de segurança em ambientes Kubernetes, a incorporação dessas práticas desde as fases iniciais até a conclusão de qualquer projeto em Kubernetes é crucial. Tal abordagem proativa não apenas serve para prevenir brechas de segurança, mas também desempenha um papel significativo na mitigação de riscos em sistemas distribuídos. A atenção meticulosa a essas melhores práticas emerge como um requisito fundamental para assegurar a robustez e a segurança contínua em ambientes baseados em Kubernetes.

5.8 CONSIDERAÇÕES FINAIS

O capítulo “Experimentos, Resultados e Discussão” teve como foco a avaliação empírica do *Framework* SARIK, empregando uma série de experimentos e análises estatísticas conduzidas com o auxílio do *software* Jamovi. Este capítulo é fundamental não apenas para validar a eficácia do *framework* em cenários práticos, mas também para estabelecer uma base sólida para sua implementação em ambientes diversos. A relevância deste capítulo está em sua habilidade de elucidar, por meio de dados concretos, tanto as vantagens quanto as limitações do *Framework* SARIK. Além disso, foram discutidas implicações significativas em termos de segurança e desempenho, fornecendo *insights* valiosos para profissionais e pesquisadores da área. Com a abordagem de todos os aspectos técnicos e avaliativos, o próximo e último capítulo, intitulado “Conclusões”, irá sintetizar as principais descobertas deste estudo, suas contribuições para o campo e as possíveis direções para pesquisas futuras.

6 CONCLUSÕES

6.1 RESULTADOS ALCANÇADOS

Os resultados desta pesquisa refletem a eficácia e o impacto do SARIK na segurança e desempenho de *clusters* de contêineres. Os principais resultados e descobertas estão divididos em duas partes: os benefícios gerais do SARIK e as análises estatísticas realizadas.

Benefícios do SARIK

- Aprimoramento da Segurança: A contribuição mais significativa do SARIK reside em seu potencial para reforçar a segurança em ambientes Kubernetes. Através da automação das regras de *iptables*, o SARIK possibilitou a implementação de políticas de rede mais granulares, aumentando assim a proteção dos Pods e serviços contra ameaças externas e internas. A estratégia adotada para a implementação de regras nas camadas de rede e transporte, por meio do *kube-proxy*, eliminou a necessidade de configurações de Pods com a *flag privileged*, tornando o ambiente mais seguro.
- Controle Eficiente de Tráfego: O SARIK demonstrou ser uma ferramenta eficaz para o gerenciamento do tráfego de rede em *clusters* Kubernetes. A capacidade de criar e administrar regras de *iptables* de forma dinâmica permitiu um direcionamento e filtragem mais eficazes do tráfego, com base em políticas predefinidas. Isso não apenas aprimorou a segurança, mas também simplificou a gestão do tráfego, minimizando a probabilidade de erros humanos e reduzindo riscos associados.
- Facilidade de Implementação: A simplicidade e a adaptabilidade do SARIK em ambientes Kubernetes foram notáveis. O *framework* foi projetado para ser facilmente integrado aos fluxos de trabalho já existentes, garantindo que a segurança não seja um obstáculo à adoção e implantação de contêineres. Isso faz do SARIK uma solução prática e viável para aprimorar a segurança em ambientes Kubernetes.

Análises Estatísticas

As análises estatísticas foram conduzidas para avaliar a eficácia do SARIK em cenários práticos, utilizando métricas como latência, taxa de resposta e taxa de transmissão. Os resultados dessas análises corroboram a eficácia do SARIK em manter ou até melhorar o desempenho em ambientes Kubernetes.

- Latência: Os dados coletados indicam que a latência foi minimamente afetada pela implementação do SARIK, mantendo-se dentro de limites aceitáveis. Isso sugere que o SARIK não compromete o desempenho das aplicações de forma significativa.
- Taxa de Resposta: A análise da taxa de resposta revelou que as políticas de rede implementadas pelo SARIK não impactaram negativamente a capacidade dos serviços de responder a solicitações de forma eficaz. O SARIK conseguiu manter a taxa de resposta em níveis que garantem uma experiência de usuário satisfatória.

- Taxa de Transmissão: Foi observado que o SARIK teve um impacto mínimo na taxa de transmissão de dados dentro do *cluster*. As políticas de rede implementadas permitiram um controle eficiente do tráfego, assegurando que a largura de banda fosse alocada de forma eficaz e evitando sobrecargas na rede.

Em resumo, os resultados alcançados confirmam que o SARIK é uma solução para o aprimoramento da segurança em ambientes Kubernetes, sem sacrificar o desempenho das aplicações. Tanto os benefícios gerais quanto as análises estatísticas sustentam a eficácia do SARIK como uma ferramenta para a segurança e o controle de tráfego em *clusters* kubernetes.

6.2 LIMITAÇÕES

Embora os resultados deste estudo sejam promissores, algumas limitações devem ser reconhecidas para uma compreensão completa do impacto e aplicabilidade do *Framework* SARIK em ambientes Kubernetes.

- **Escopo do Estudo:** A pesquisa foi realizada em um ambiente controlado, o que pode não refletir completamente as complexidades e variáveis de um ambiente de produção em grande escala. Portanto, os resultados podem variar quando aplicados em diferentes contextos operacionais.
- **Distribuições do Sistema Operacional:** O estudo focou em distribuições Linux específicas (Alpine e Debian), e os resultados podem não ser generalizáveis para outras distribuições. Diferenças em configurações de *kernel*, *drivers* e políticas de segurança podem afetar o desempenho e a segurança de forma diferente.
- **Métricas de Avaliação:** As métricas utilizadas para avaliar o desempenho e a segurança (latência, taxa de resposta e taxa de transmissão) são abrangentes, mas não exaustivas. Outras métricas relevantes, como consumo de recursos, poderiam fornecer *insights* adicionais.
- **Regras de *Firewall*:** Embora o SARIK permita uma gestão eficiente das regras de *iptables*, a complexidade crescente dessas regras em um ambiente em escala pode tornar o gerenciamento mais desafiador. A pesquisa não abordou completamente como o SARIK lidaria com cenários de alta complexidade.
- **Impacto na Latência:** O estudo mostrou que o SARIK tem um impacto mínimo na latência, mas não explorou cenários onde a latência é crítica, como em sistemas de tempo real ou aplicações de baixa latência.
- **Adaptação a Novas Ameaças:** O SARIK foi eficaz contra as ameaças consideradas neste estudo, mas sua eficácia contra novas e emergentes ameaças de segurança não foi avaliada.

Essas limitações apontam para a necessidade de pesquisas futuras para validar ainda mais a eficácia do SARIK em uma variedade de cenários e configurações.

6.3 TRABALHOS FUTUROS

Com base nas descobertas e limitações deste estudo, várias direções para trabalhos futuros podem ser identificadas para expandir o entendimento e a aplicabilidade do *Framework* SARIK em ambientes Kubernetes.

- **Ambientes de Produção:** Um dos próximos passos seria testar o SARIK em ambientes de produção em grande escala para avaliar sua eficácia e desempenho sob condições mais realistas e complexas.
- **Diversidade de Distribuições:** Seria útil explorar como o SARIK se comporta em diferentes distribuições do sistema operacional, além do Alpine e Debian, para entender melhor sua adaptabilidade e eficácia em diversos cenários.
- **Métricas Adicionais:** A inclusão de outras métricas de desempenho, como consumo de recursos e escalabilidade, poderia fornecer uma visão mais completa do impacto do SARIK em ambientes Kubernetes.
- **Gestão de Regras Complexas:** Estudos adicionais poderiam focar em como o SARIK gerencia regras de *iptables* em cenários de alta complexidade, especialmente em *clusters* que crescem dinamicamente.
- **Latência em Sistemas Críticos:** Uma investigação mais aprofundada sobre o impacto do SARIK em sistemas onde a latência é um fator crítico pode ser valiosa.
- **Resposta a Ameaças Emergentes:** Avaliar a capacidade do SARIK de se adaptar e responder a novas e emergentes ameaças de segurança seria um aspecto crucial para futuras pesquisas.
- **Integração com Outras Ferramentas:** Explorar a integração do SARIK com outras ferramentas de segurança e monitoramento poderia oferecer uma solução de segurança mais holística.
- **Automatização e Inteligência Artificial:** O uso de algoritmos de aprendizado de máquina para aprimorar a detecção de anomalias e a automação das regras de *iptables* poderia ser uma direção interessante para futuras investigações.

Essas direções para trabalhos futuros têm o potencial de fortalecer ainda mais o *Framework* SARIK como uma solução robusta e adaptável para aprimorar a segurança e o desempenho em ambientes Kubernetes.

Este capítulo de “Conclusões” teve como objetivo sintetizar os resultados alcançados, discutir as limitações do estudo e apontar direções para trabalhos futuros. A pesquisa validou a eficácia do *Framework* SARIK em melhorar a segurança e o controle do tráfego em ambientes Kubernetes, sem comprometer o desempenho das aplicações. As análises estatísticas reforçaram a aplicabilidade prática do *framework*, fornecendo uma base sólida para sua adoção em diferentes contextos.

As limitações e os desafios identificados ao longo deste estudo não apenas fornecem um entendimento mais profundo do *framework* SARIK, mas também abrem caminhos para futuras investigações. Estas poderão abordar as limitações do presente estudo e expandir o corpo de conhecimento em torno do SARIK.

Em resumo, este trabalho contribui para a literatura em segurança de contêineres e gestão de tráfego em ambientes Kubernetes. As descobertas aqui apresentadas têm o potencial de impactar tanto a comunidade acadêmica quanto a industrial, fornecendo *insights* valiosos para a implementação de soluções de segurança mais robustas em ambientes de contêineres.

Com isso, concluo este trabalho, na expectativa de que as contribuições e descobertas realizadas sirvam como um ponto de partida para futuras pesquisas e aplicações práticas na área.

REFERÊNCIAS BIBLIOGRÁFICAS

1 PANIZZON, G.; BATTISTI, J.; KOSLOVSKI, G.; PILLON, M.; MIERS, C. A taxonomy of container security on computational clouds: concerns and solutions. *Revista de Informática Teórica e Aplicada*, v. 26, p. 47, 2019.

2 FALCãO, E.; SILVA, M.; SOUZA, C.; BRITO, A. Autenticando aplicações nativas da nuvem com identidades spiffe. p. 100–144, 2021.

3 VASSOLER, G.; SPOHN, M. Análise de desempenho de rolling updates do kubernetes em ambientes de estresse. *Revista Brasileira de Computação Aplicada*, v. 12, p. 70–84, 2020.

4 BARBARA, D.; AZEVEDO, I.; MENEZES, P.; RODRíGUEZ, G. Um survey sobre experiências industriais na utilização de microsserviços com contêineres. *Interfaces Científicas - Exatas e Tecnológicas*, v. 4, p. 31–50, 2020.

5 ALLEY, A. *Cloud providers see "aggressive"growth amidst Covid-19 pandemic*. 2020. Disponível em: <https://www.datacenterdynamics.com/en/news/cloud-providers-see-aggressive-growth-amidst-covid-19-outbreak/>. Acesso em: 2022-01-02.

6 VERMEE, B.; HENRY, W. *Shifting Docker security left.* 2019. Disponível em: <https://snyk.io/blog/shifting-docker-security-left>. Acesso em: 2021-11-25.

7 SILVA, F. *Docker: como hackers estão explorando conteinerização.* 2018. Disponível em: <https://king.host/blog/2018/06/docker-hackers-conteinerizacao/>. Acesso em: 2023-09-10.

8 KARN, R. R.; KUDVA, P.; HUANG, H.; SUNEJA, S.; ELFADEL, I. M. Cryptomining detection in container clouds using system calls and explainable machine learning. *IEEE transactions on parallel and distributed systems*, IEEE, v. 32, n. 3, p. 674–691, 2020.

9 OZTOPRAK, K.; TUNCEL, Y. K.; BUTUN, I. Technological transformation of telco operators towards seamless iot edge-cloud continuum. *Sensors*, MDPI, v. 23, n. 2, p. 1004, 2023.

10 SENJAB, K.; ABBAS, S.; AHMED, N. et al. A survey of kubernetes scheduling algorithms. *Journal of Cloud Computing*, SpringerOpen, v. 12, n. 1, p. 1–26, 2023.

11 GOLDBERG, R. P. Architecture of virtual machines. In: *Proceedings of the workshop on virtual computer systems*. [S.l.: s.n.], 1973. p. 74–112.

12 TAURION, C. Grid computing: um novo paradigma computacional. *Rio de Janeiro: Brasport*, 2004.

13 MEYER, R. A.; SEAWRIGHT, L. H. A virtual machine time-sharing system. *IBM Systems Journal*, IBM, v. 9, n. 3, p. 199–218, 1970.

14 CAMPBELL, W. M.; MOORE, P.; SHARMA, M. Cultural transformation to support the adoption of green it. In: IEEE. *2014 28th international conference on advanced information networking and applications workshops*. [S.l.], 2014. p. 554–559.

15 BARI, M. F.; BOUTABA, R.; ESTEVES, R.; GRANVILLE, L. Z.; PODLESNY, M.; RABBANI, M. G.; ZHANG, Q.; ZHANI, M. F. Data center network virtualization: A survey. *IEEE communications surveys & tutorials*, IEEE, v. 15, n. 2, p. 909–928, 2012.

16 VITALINO, J. F. N.; CASTRO, M. A. N. *Descomplicando o Docker 2a edição*. [S.l.]: Brasport, 2018.

17 ROCHA, S. L. Proposta de um framework para detecção de intrusão em clusters de orquestração de contêineres utilizando machine learning para identificação de anomalias em system calls. 2023.

18 MERKEL, D. et al. Docker: lightweight linux containers for consistent development and deployment. *Linux j*, v. 239, n. 2, p. 2, 2014.

19 COMBE, T.; MARTIN, A.; PIETRO, R. D. To docker or not to docker: A security perspective. *IEEE Cloud Computing*, IEEE, v. 3, n. 5, p. 54–62, 2016.

20 KUBERNETES. *Kubernetes overview*. 2023. Disponível em: <https://kubernetes.io/pt-br/docs/concepts/overview/>.

21 BURNS, B.; GRANT, B.; OPPENHEIMER, D.; BREWER, E.; WILKES, J. Borg, omega, and kubernetes: Lessons learned from three container-management systems over a decade. *Queue*, ACM New York, NY, USA, v. 14, n. 1, p. 70–93, 2016.

22 MEDEL, V.; RANA, O.; BAÑARES, J. Á.; ARRONATEGUI, U. Modelling performance & resource management in kubernetes. In: *Proceedings of the 9th International Conference on Utility and Cloud Computing*. [S.l.: s.n.], 2016. p. 257–262.

23 CHANG, C.-C.; YANG, S.-R.; YEH, E.-H.; LIN, P.; JENG, J.-Y. A kubernetes-based monitoring platform for dynamic cloud resource provisioning. In: IEEE. *GLOBECOM 2017-2017 IEEE Global Communications Conference*. [S.l.], 2017. p. 1–6.

24 VAYGHAN, L. A.; SAIED, M. A.; TOEROE, M.; KHENDEK, F. Deploying microservice based applications with kubernetes: Experiments and lessons learned. In: IEEE. *2018 IEEE 11th international conference on cloud computing (CLOUD)*. [S.l.], 2018. p. 970–973.

25 HIGHLY Available Kubernetes. 2023. <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/>. Acesso em: 15 de outubro de 2023.

26 WHAT is etcd. 2023. <https://etcd.io/>. Acesso em: 15 de outubro de 2023.

27 MURALIDHARAN, S.; SONG, G.; KO, H. Monitoring and managing iot applications in smart cities using kubernetes. *Cloud Computing*, v. 11, 2019.

28 WEI-GUO, Z.; XI-LIN, M.; JIN-ZHONG, Z. Research on kubernetes' resource scheduling scheme. In: *Proceedings of the 8th International Conference on Communication and Network Security*. [S.l.: s.n.], 2018. p. 144–148.

29 SITE Minikube. 2023. <https://minikube.sigs.k8s.io/docs/>. Acesso em: 22 de setembro de 2023.

30 KHALEL, M. M.; PUGAZHENDHI, M. A.; RAJ, G. R. Enhanced load balancing in kubernetes cluster by minikube. In: IEEE. *2022 International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN)*. [S.l.], 2022. p. 1–5.

31 CALICO - Network plugin. 2023. <https://github.com/projectcalico/calico>. Acesso em: 19 março 2023.

32 FLANNEL - Network plugin. 2023. <https://github.com/flannel-io/flannel>. Acesso em: 19 março 2023.

33 WEAVE - Network plugin. 2023. <https://github.com/weaveworks/weave>. Acesso em: 19 março 2023.

34 CILIUM - Network plugin. 2023. <https://github.com/cilium/cilium>. Acesso em: 19 março 2023.

35 QI, S.; KULKARNI, S. G.; RAMAKRISHNAN, K. Understanding container network interface plugins: design considerations and performance. In: IEEE. *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*. [S.l.], 2020. p. 1–6.

36 FILHO, J. E. M. *Iptables, por João Eriberto Mota filho*. 2023. <http://eriberto.pro.br/iptables/>. Acesso em: 16 de agosto de 2023.

37 PROMETHEUS overview. 2023. <https://prometheus.io/docs/introduction/overview/>. Acesso em: 15 de outubro de 2023.

38 OVERVIEW Red Hat, O que é Grafana? 2023. <https://www.redhat.com/pt-br/topics/data-services/what-is-grafana>. Acesso em: 15 de outubro de 2023.

39 BALABANIAN, F.; HENRIQUES, M. Tocker: framework para a segurança de containers docker. In: SBC. *Anais Estendidos do XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*. [S.l.], 2019. p. 145–154.

40 NAM, J.; LEE, S.; SEO, H.; PORRAS, P.; YEGNESWARAN, V.; SHIN, S. {BASTION}: A security enforcement network stack for container networks. In: *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. [S.l.: s.n.], 2020. p. 81–95.

41 KULATHUNGA, R. *Dynamic security model for container orchestration platform*. Tese (Doutorado), 2021.

42 ROCHA, S. L.; MENDONCA, F. L. Lopes de; PUTTINI, R. S.; NUNES, R. R.; NZE, G. D. A. Dcids—distributed container ids. *Applied Sciences*, MDPI, v. 13, n. 16, p. 9301, 2023.

43 SYSDIG Secure. 2023. <https://docs.sysdig.com/en/docs/sysdig-secure>. Acesso em: 19 março 2023.

44 KUDO, R.; KITAHARA, H.; GAJANANAN, K.; WATANABE, Y. Integrity protection for kubernetes resource based on digital signature. In: IEEE. *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. [S.l.], 2021. p. 288–296.

45 ZHU, H.; GEHRMANN, C. Kub-sec, an automatic kubernetes cluster apparmor profile generation engine. In: IEEE. *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. [S.l.], 2022. p. 129–137.

46 BRINGHENTI, D.; SISTO, R.; VALENZA, F. Security automation for multi-cluster orchestration in kubernetes. In: IEEE. *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. [S.l.], 2023. p. 480–485.

47 LI, Y.; HU, X.; JIA, C.; WANG, K.; LI, J. Kano: Efficient cloud native network policy verification. *IEEE Transactions on Network and Service Management*, IEEE, 2022.

48 LEE, S.; NAM, J. Kunerva: Automated network policy discovery framework for containers. *IEEE Access*, IEEE, 2023.

49 MINIKUBE, R. *Recomendação Minikube para uso do Calico*. 2023. <https://minikube.sigs.k8s.io/docs/handbook/network_policy/>. Acesso em: 27 de outubro de 2023.

50 TANENBAUM, A. S. Redes de computadores/andrew s. *Tanenbaum: Tradução [ds 3. ed. original] Insight Serviços de Informática. Rio de Janeiro: Campus*, 1997.

51 HAIR, J. F.; BLACK, W. C.; BABIN, B. J.; ANDERSON, R. E.; TATHAM, R. L. *Análise multivariada de dados*. [S.l.]: Bookman editora, 2009.

52 JAMOVI open statistical software for the desktop and cloud. <https://www.jamovi.org/>. Access Date March, 2023. [Online].

53 SHAMIM, M. S. I.; BHUIYAN, F. A.; RAHMAN, A. Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices. *2020 IEEE Secure Development (SecDev)*, IEEE, p. 58–64, 2020.

APÊNDICES

O apêndice a seguir é dedicado principalmente ao código fonte do projeto SARIK, elemento central desta dissertação. Além disso, oferece um link para uma discussão detalhada, apresentação e análise dos dados obtidos durante a fase experimental do estudo. Este conteúdo complementar está disponível no endereço: https://sarik.org/relatorio_tecnico/.

Este apêndice destina-se a fornecer acesso direto ao código fonte desenvolvido, permitindo assim uma visão prática e aprofundada da implementação do SARIK. O código fonte incluído nestas páginas é um recurso valioso para pesquisadores e profissionais interessados em segurança em ambientes Kubernetes e representa um aspecto prático essencial do trabalho.

Além do código fonte, o link fornecido leva a uma análise técnica detalhada, incluindo discussões sobre os resultados experimentais. Esta seção do apêndice serve como um recurso adicional ao capítulo “Resultados e Discussão”, oferecendo insights técnicos e detalhados que complementam a discussão principal da dissertação.

Este formato de apresentação tem o objetivo de garantir que tanto o aspecto prático (código fonte) quanto o teórico-analítico (análise de dados) da pesquisa sejam facilmente acessíveis e compreensíveis, assegurando transparência e rigor no processo de pesquisa e facilitando a replicação ou expansão dos estudos apresentados

I.1 RELATÓRIOS TÉCNICOS DAS MÉTRICAS

Figura 1: Página do relatório técnico.
Fonte: Próprio autor

I.2 CÓDIGO-FONTE SARIK

Algoritmo 7 Módulo principal - SARIK

```
#!/usr/bin/env bash
#
# SARIK – SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
#
# SITE: https://sarik.org
# Autor: Jonathan G.P. dos Santos
# Manutenção: Jonathan G.P. dos Santos
#
#
#––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––#
#Exemplo:
#
# Access the director AppTest/minikube
# Execute command:
# 1 – kubectl create –f namespaces/
# 2 – kubectl create –f services/
# 3 – kubectl create –f deployements/
# After the command, return for / with command ../..
# Execute the script sarik.sh
#
#––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––#
#Histórico
#
# v1.0 03/12/2021, Jonathan G.P. dos Santos
# – Beta.
# V1.01 02/01/2022, Jonathan G.P. dos Santos
# – until, loop for configuration iptables
# – variables
# V1.02 04/01/2022, Jonathan G.P. dos Santos
# – Add While and code tweaks
# – Variables IP, PORT etc
# V1.03 04/08/2023, Jonathan G.P. dos Santos
# – Add networkpolicy
# – Add Modules for CRUD
# V1.04 01/10/2023, Jonathan G.P. dos Santos
# – Add networkpolicy ingress
# – Add Modules 6 nd 7
# V1.05 05/10/2023, Jonathan G.P. dos Santos
# – Add Modules and function
#––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––#
#Testado em:
# bash 5.0.17
```

```
#
#
#-------------------------------------------------------------#
#Agradecimentos:
#
# Dr. Vinícius Pereira Gonçalves – Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto
# MSc. Elivaldo Ribeiro De Santana – Estatística
# Luiz Eduardo Rodrigues Lima – Layout página SARIK e auxílio na pesquisa
# Tainá Naró S. Moura – Revisão da escrita dos artigos
# Juliana Cristina Sampaio Silva – Revisão da escrita do pré–projeto
# Dr. Hervaldo Sampaio Carvalho – Pelo apoio e supervisão
# Dr. Gustavo Adolfo Sierra Romero – Pelo apoio e supervisão
#
#-------------------------------------------------------------#

# Carregando todos os módulos, function and variables
source module1.sh
source module2.sh
source module3.sh
source module4.sh
source module5.sh
source module6.sh
source module7.sh
source module8.sh
source module9.sh
source module10.sh
source variables.sh
source function.sh

# Criando os diretórios, caso seja necessário

# Checa se o diretório policies existe
if [ ! –d "policies" ]; then
   # Cria o diretório policies caso não exista
   mkdir policies
fi

# Checa se o diretório backup_policies existe
if [ ! –d "backup_policies" ]; then
   # Cria o diretório backup_policies caso não exista
   mkdir backup_policies
fi

# Checa se o diretório egress existe
if [ ! –d "egress" ]; then
   # Cria o diretório egress caso não exista
```

```
    mkdir egress
fi

# Checa se o diretório ingress existe
if [ ! –d "ingress" ]; then
    # Cria o diretório ingress caso não exista
    mkdir ingress
fi

# Ative extglob logo no início
shopt –s extglob

# Verificar o diretório atual

CURRENT_DIR=$(pwd)
TARGET_DIR="SARIK"

    if [[ $CURRENT_DIR == *$TARGET_DIR ]]; then

        sleep 1
        # Remove todos os arquivos dos diretórios
        rm –R policies/*
        rm –R backup_policies/*
        rm –R ingress/*
        rm –R egress/*

    else
        echo "Por favor, crie a pasta 'SARIK' e coloque este(s) script(s) dentro deste diretório."
        sleep 2
        exit 1
    fi

# Desative extglob se não for mais necessário
shopt –u extglob


echo " "
cat << "EOF"
      ______ ______ ______ _ __
    / _____| /\ | __ \ |_ _| | |/ /
   | (___ / \ | |__) | | | | ' /
    \___ \ / /\ \ | _ / | | | <
    ____) | / ____ \ | | \ \ _| |_ | . \
   |_____/ /_/ \_\ |_| \_\ |_____| |_|\_\

EOF
echo
```

```
    "=============================================================================================="
echo "============ AUTOMATIC SECURITY OF RULES ON IPTABLES IN KUBERNETES ================="
echo "============ By @jonathamgg and @jonathan | DC ================="
echo
    "=============================================================================================="
echo " "
echo " "

echo "Agradecimentos aos professores:"
sleep 1
echo "Dr. Vinícius Pereira Gonçalves – Orientador do projeto"
sleep 1
echo "Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto"
sleep 1
echo " "
echo " "
#––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––#

#––––––––––––––––––––––––––––––––––TESTES––––––––––––––––––––––––––#
#Docker install?
#[ ! –x "$(which docker)" ] && printf "Precisa instalar o docker, por favor, instale.\n" && exit 1
#minikube install?
#[ ! –x "$(which minikube)" ] || [ ! –x "$(which kubectl)" ] && printf "Precisa instalar o minikube ou kubectl, por favor, instale.\n" && exit 1
#––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––#

#––––––––––––––––––––––––––––––––––EXECUÇÃO––––––––––––––––––––––––––#
#
main() {
  #Fazendo uma checagem caso tenha passado parametro no script
  if [ "$#" –eq 0 ]; then
    configure_firewall_rules_auto
    exit 0
  fi

  while [ "$1" != "" ]; do
    case $1 in
      –mn | ––manual )
        configure_firewall_rules_manual
        ;;
      –D | ––delete )
        delete_rules_auto
        ;;
      –l | ––view )
        view_networkpolicy
        ;;
```

```
180         -d | --del )
181           delete_networkpolicy
182           ;;
183         -i | --ingress )
184           configure_firewall_rules_ingress
185           ;;
186         -m | --monitor )
187           policies-monitor
188           ;;
189         -b | --backup )
190           backup
191           ;;
192         -v | --validate )
193           validate
194           ;;
195         -h | --help )
196           show_help
197           ;;
198         * )
199           echo "Opção inválida: $1"
200           exit 1
201       esac
202       shift
203     done
204 }
205
206 # Chama a função principal com todos os argumentos passados para o script
207 main "$@"
208 #------------------------------------------------------------------#
```

Algoritmo 8 Módulo 1 - Políticas manual (Egress)

```
1  #!/usr/bin/env bash
2  #
3  # SARIK - SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
4  #
5  # SITE: https://sarik.org
6  # Autor: Jonathan G.P. dos Santos
7  # Manutenção: Jonathan G.P. dos Santos
8  #
9  #
10 #------------------------------------------------------------------------------------#
11 #Histórico
12 #
13 # V1.05 05/10/2023, Jonathan G.P. dos Santos
14 # - Add Module 1
```

```
#-------------------------------------------------------------------------------#
#Testado em:
# bash 5.0.17
#
#
#-------------------------------------------------------------------------------#
#Agradecimentos:
#
#
# Dr. Vinícius Pereira Gonçalves – Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto
#
#
#-------------------------------------------------------------------------------#

#----------------------------------MODULOS--------------------------------------

# Carregando todas as funções e variáveis
source variables.sh
source function.sh


# Módulo 1: Cadastrar regras manuais na interface de saída (egress)


# Criando o diretorio para criação das regras individuais, caso contrário, exclui a regras nesse
    diretorio
if [ -d "egress" ]; then
    #Deletetando regras existentes no diretório
    rm -R egress/* 2>/dev/null
  else
    #criando o diretório
    mkdir egress
fi

configure_firewall_rules_manual() {
  echo "Configurando políticas de rede manualmente..."
  sleep 3
  progress-bar2 $CONT2
# Listar todas as namespaces e permitir que o usuário selecione uma
namespaces=$(kubectl get namespaces --no-headers -o custom-columns=":metadata.name")
echo "Selecione uma namespace:"
select ns in $namespaces; do
  if [[ -n "$ns" ]]; then
    break
  else
    echo "Seleção inválida. Tente novamente."
```

```
    fi
done

# Listar todos os pods na namespace selecionada e permitir que o usuário selecione um
pods=$(kubectl get pods –n $ns ––no–headers –o custom–columns=":metadata.name")
echo "Selecione um pod:"
select pod in $pods; do
    if [[ –n "$pod" ]]; then
        break
    else
        echo "Seleção inválida. Tente novamente."
    fi
done

while true; do
    # Perguntar ao usuário qual protocolo e porta ele deseja bloquear
    read –p "Informe o protocolo que você deseja bloquear (ex: TCP, UDP): " protocol
    read –p "Informe a porta que você deseja bloquear: " port

    # Criar a regra de bloqueio
    egress/create_block_rule $pod $ns $protocol $port

    # Perguntar ao usuário se ele deseja continuar
    read –p "Deseja continuar adicionando regras? (y/N): " choice
    if [[ "$choice" != "y" && "$choice" != "Y" ]]; then
        break
    fi
done

# Aplicar todas as regras
kubectl apply –f egress/

echo "Regras aplicadas com sucesso."
}
```

Algoritmo 9 Módulo 2 - Exclusão automática das políticas

```
#!/usr/bin/env bash
#
# SARIK – SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
#
# SITE: https://sarik.org
# Autor: Jonathan G.P. dos Santos
# Manutenção: Jonathan G.P. dos Santos
#
#
```

```
#-------------------------------------------------------------------------------------------#
#Histórico
#
# V1.05 05/10/2023, Jonathan G.P. dos Santos
# – Add Module 2
#-------------------------------------------------------------------------------------------#
#Testado em:
# bash 5.0.17
#
#
#-------------------------------------------------------------------------------------------#
#Agradecimentos:
#
#
# Dr. Vinícius Pereira Gonçalves – Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto
#
#
#-------------------------------------------------------------------------------------------#

#----------------------------------MODULOS-------------------------------------------------

# Carregando todas as funções e variáveis
source variables.sh
source function.sh

# Módulo 2: Exclusão de todas as políticas de rede automática

delete_rules_auto() {
  echo "Excluindo todas as políticas de rede atuais..."
  sleep 3
  #Chamando a função para imprimir logo
  sarik–msg–exclusao
VALUE_NAMESPACE=$(kubectl get namespace | awk '{print $1}' | grep –v ^'default' | grep –v
    ^NAME | grep –v ^kube)

  if [[ –z "$VALUE_NAMESPACE" ]]; then
    echo "Não há namespaces personalizados no cluster."
    exit 1
  else
    echo "Namespace(s) encontrado(s): $VALUE_NAMESPACE"
    for ns in $VALUE_NAMESPACE; do
      # Verificar se existem políticas de rede para o namespace
      NETWORK_POLICIES=$(kubectl get networkpolicy –n $ns ––no–headers 2>/dev/null)
      if [[ –z "$NETWORK_POLICIES" ]]; then
        echo "Não há políticas de rede para o namespace $ns."
      else
```

```
            echo "Políticas de rede para o namespace $ns:"
            echo "$NETWORK_POLICIES"

            # Excluir todas as políticas de rede para o namespace
            echo "Excluindo todas as políticas de rede para o namespace $ns..."
            kubectl delete networkpolicy --all -n $ns
            echo ""
            echo "Todas as políticas de rede para o namespace $ns foram excluídas."
            echo ""
        fi
    done
  fi
}
```

Algoritmo 10 Módulo 3 - Visualização das políticas

```
#!/usr/bin/env bash
#
# SARIK - SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
#
# SITE: https://sarik.org
# Autor: Jonathan G.P. dos Santos
# Manutenção: Jonathan G.P. dos Santos
#
#
#------------------------------------------------------------------------------#
#Histórico
#
# V1.05 05/10/2023, Jonathan G.P. dos Santos
# - Add Module 3
#------------------------------------------------------------------------------#
#Testado em:
# bash 5.0.17
#
#
#------------------------------------------------------------------------------#
#Agradecimentos:
#
#
# Dr. Vinícius Pereira Gonçalves - Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho - Coorientador do projeto
#
#
#------------------------------------------------------------------------------#

#----------------------------------MODULOS-------------------------------------
```

```
31
32 # Carregando todas as funções e variáveis
33 source variables.sh
34 source function.sh
35
36 # Módulo 3: Visualização das políticas de rede
37
38 view_networkpolicy() {
39   echo "Aguarde, processando a solicitação..."
40   sleep 3
41   #Chamando a função para visualizar logo
42   sarik–msg–view
43 VALUE_NAMESPACE=$(kubectl get namespace | awk '{print $1}' | grep –v ^'default' | grep –v
       ^NAME | grep –v ^kube)
44
45   if [[ –z "$VALUE_NAMESPACE" ]]; then
46     echo "Não há nespaces no cluster."
47     exit 1
48   else
49     echo "Namespace(s) encontrado(s): $VALUE_NAMESPACE"
50     echo " "
51     for ns in $VALUE_NAMESPACE; do
52       #Verificar se existem políticas de rede para o namespace
53       NETWORK_POLICIES=$(kubectl get networkpolicy –n $ns ––no–headers 2>/dev/null )
54
55       if [[ –z "$NETWORK_POLICIES" ]]; then
56         echo "Não há políticas de rede para o namespace $ns."
57       else
58         echo "Políticas de rede para o namespace $ns:"
59         echo "$NETWORK_POLICIES"
60       fi
61     done
62   fi
63 }
```

Algoritmo 11 Módulo 4 - Exclusão das políticas manual

```
1 #!/usr/bin/env bash
2 #
3 # SARIK – SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
4 #
5 # SITE: https://sarik.org
6 # Autor: Jonathan G.P. dos Santos
7 # Manutenção: Jonathan G.P. dos Santos
8 #
9 #
```

```
#-------------------------------------------------------------------------------------#
#Histórico
#
# V1.05 05/10/2023, Jonathan G.P. dos Santos
# – Add Module 4
#-------------------------------------------------------------------------------------#
#Testado em:
# bash 5.0.17
#
#
#-------------------------------------------------------------------------------------#
#Agradecimentos:
#
#
# Dr. Vinícius Pereira Gonçalves – Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto
#
#
#-------------------------------------------------------------------------------------#

#----------------------------------MODULOS--------------------------------------------

# Carregando todas as funções e variáveis
source variables.sh
source function.sh

# Módulo 4: Exclusão de políticas de rede individuais

delete_networkpolicy(){
  echo "Aguarde, processando a solicitação..."
  sleep 3
# Listar todas as namespaces e permitir que o usuário selecione uma
namespaces=$(kubectl get namespaces ––no–headers –o custom–columns=":metadata.name")
echo "Selecione uma namespace:"
select ns in $namespaces; do
  if [[ –n "$ns" ]]; then
    break
  else
    echo "Seleção inválida. Tente novamente."
  fi
done

# Listar todos os pods na namespace selecionada e permitir que o usuário selecione uma
pods=$(kubectl get pods –n $ns ––no–headers –o custom–columns=":metadata.name")
echo "Selecione um pod:"
select pod in $pods; do
  if [[ –n "$pod" ]]; then
```

```
57        break
58      else
59        echo "Seleção inválida. Tente novamente."
60      fi
61    done
62
63    # Listar todas as políticas de rede para o pod selecionado
64    policies=$(kubectl get networkpolicy –n $ns ––no–headers –o custom–columns=":metadata.name" |
          grep "$pod")
65    echo "Selecione uma política de rede para excluir:"
66    select policy in $policies; do
67      if [[ –n "$policy" ]]; then
68        break
69      else
70        echo "Seleção inválida. Tente novamente."
71      fi
72    done
73
74    # Excluir a regra de bloqueio
75    port=$(echo $policy | awk –F '–' '{print $3}')
76    delete_block_rule $policy $ns
77
78    echo "Regra excluída com sucesso."
79
80    }
```

Algoritmo 12 Módulo 5 - Configuração das políticas manual (ingress)

```
1   #!/usr/bin/env bash
2   #
3   # SARIK – SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
4   #
5   # SITE: https://sarik.org
6   # Autor: Jonathan G.P. dos Santos
7   # Manutenção: Jonathan G.P. dos Santos
8   #
9   #
10  #––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––#
11  #Histórico
12  #
13  # V1.05 05/10/2023, Jonathan G.P. dos Santos
14  # – Add Module 5
15  #––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––#
16  #Testado em:
17  # bash 5.0.17
18  #
```

```
#
#-------------------------------------------------------------------------------#
#Agradecimentos:
#
#
# Dr. Vinícius Pereira Gonçalves – Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto
#
#
#-------------------------------------------------------------------------------#

#----------------------------------MODULOS--------------------------------------

# Carregando todas as funções e variáveis
source variables.sh
source function.sh

# Módulo 5: Cadastrar regras manuais na interface de entrada (ingress)


# Criando o diretorio para criação das regras individuais, caso contrário, exclui a regras nesse
    diretorio
if [ -d "ingress" ]; then
    #Deletetando regras existentes no diretório
    rm -R ingress/* 2>/dev/null
  else
    #criando o diretório
    mkdir ingress
fi

configure_firewall_rules_ingress(){
# Listar todas as namespaces e permitir que o usuário selecione uma
namespaces=$(kubectl get namespaces --no-headers -o custom-columns=":metadata.name")
echo "Selecione uma namespace:"
select ns in $namespaces; do
  if [[ -n "$ns" ]]; then
    break
  else
    echo "Seleção inválida. Tente novamente."
  fi
done

# Listar todos os pods na namespace selecionada e permitir que o usuário selecione uma
pods=$(kubectl get pods -n $ns --no-headers -o custom-columns=":metadata.name")
echo "Selecione um pod:"
select pod in $pods; do
  if [[ -n "$pod" ]]; then
```

```
65        break
66      else
67        echo "Seleção inválida. Tente novamente."
68      fi
69  done
70
71  while true; do
72    # Perguntar ao usuário qual protocolo e porta ele deseja bloquear
73    read –p "Informe o protocolo que você deseja bloquear (ex: TCP, UDP): " protocol
74    read –p "Informe a porta que você deseja bloquear: " port
75
76    # Criar a regra de bloqueio
77    ingress/create_ingress_block_rule $pod $ns $port $protocol
78
79    # Perguntar ao usuário se ele deseja continuar
80    read –p "Deseja continuar adicionando regras? (y/N): " choice
81    if [[ "$choice" != "y" && "$choice" != "Y" ]]; then
82      break
83    fi
84  done
85
86  # Aplicar a política de rede
87   kubectl apply –f ingress/
88
89  echo "Regras aplicadas com sucesso."
90
91  }
```

Algoritmo 13 Módulo 6 - Monitoramento

```
1   #!/usr/bin/env bash
2   #
3   # SARIK – SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
4   #
5   # SITE: https://sarik.org
6   # Autor: Jonathan G.P. dos Santos
7   # Manutenção: Jonathan G.P. dos Santos
8   #
9   #
10  #------------------------------------------------------------------------------#
11  #Histórico
12  #
13  # V1.05 05/10/2023, Jonathan G.P. dos Santos
14  # – Add Module 6
15  #------------------------------------------------------------------------------#
16  #Testado em:
```

```
# bash 5.0.17
#
#
#--------------------------------------------------------------------------------------#
#Agradecimentos:
#
#
# Dr. Vinícius Pereira Gonçalves – Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto
#
#
#--------------------------------------------------------------------------------------#

#----------------------------------MODULOS---------------------------------------------

# Carregando todas as funções e variáveis
source variables.sh
source function.sh

# Módulo 6: Monitoramento das políticas de rede

policies-monitor(){
#Verificar se o usuário quer monitorar políticas
read -p "Você deseja monitorar políticas de rede? (y/N): " choice

if [[ "$choice" == "y" || "$choice" == "Y" ]]; then
   monitor_policies
else
   echo "Monitoramento de políticas de rede cancelado."
fi
}
```

Algoritmo 14 Módulo 7 - Backup

```
#!/usr/bin/env bash
#
# SARIK – SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
#
# SITE: https://sarik.org
# Autor: Jonathan G.P. dos Santos
# Manutenção: Jonathan G.P. dos Santos
#
#
#--------------------------------------------------------------------------------------#
#Histórico
#
```

```
# V1.05 05/10/2023, Jonathan G.P. dos Santos
# – Add Module 7
#-------------------------------------------------------------------------------#
#Testado em:
# bash 5.0.17
#
#
#-------------------------------------------------------------------------------#
#Agradecimentos:
#
#
# Dr. Vinícius Pereira Gonçalves – Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto
#
#
#-------------------------------------------------------------------------------#

#-----------------------------------MODULOS-------------------------------------

# Carregando todas as funções e variáveis
source variables.sh
source function.sh

# Módulto 7: Backup

backup(){
# Verificar se o usuário quer fazer backup das políticas
read –p "Você deseja fazer backup das políticas de rede? (y/N): " choice

if [[ "$choice" == "y" || "$choice" == "Y" ]]; then
   backup_policies
else
   echo "Backup de políticas de rede cancelado."
fi
}
```

Algoritmo 15 Módulo 8 - Validação das políticas

```
#!/usr/bin/env bash
#
# SARIK – SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
#
# SITE: https://sarik.org
# Autor: Jonathan G.P. dos Santos
# Manutenção: Jonathan G.P. dos Santos
#
```

```
#
#------------------------------------------------------------------------------#
#Histórico
#
# V1.05 05/10/2023, Jonathan G.P. dos Santos
# – Add Module 8
#------------------------------------------------------------------------------#
#Testado em:
# bash 5.0.17
#
#
#------------------------------------------------------------------------------#
#Agradecimentos:
#
#
# Dr. Vinícius Pereira Gonçalves – Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto
#
#
#------------------------------------------------------------------------------#

#----------------------------------MODULOS-------------------------------------

# Carregando todas as funções e variáveis
source variables.sh
source function.sh

# Módulo 8: Validate

validate(){
# Verificar se o usuário quer validar as políticas
read –p "Você deseja validar as políticas de rede? (y/N): " choice

if [[ "$choice" == "y" || "$choice" == "Y" ]]; then
   validate_policies
else
   echo "Validação de políticas de rede cancelada."
fi
}
```

Algoritmo 16 Módulo 9 - Ajuda

```
#!/usr/bin/env bash
#
# SARIK – SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
#
```

```
# SITE: https://sarik.org
# Autor: Jonathan G.P. dos Santos
# Manutenção: Jonathan G.P. dos Santos
#
#
#------------------------------------------------------------------------------#
#Histórico
#
# V1.05 05/10/2023, Jonathan G.P. dos Santos
# – Add Module 9
#------------------------------------------------------------------------------#
#Testado em:
# bash 5.0.17
#
#
#------------------------------------------------------------------------------#
#Agradecimentos:
#
#
# Dr. Vinícius Pereira Gonçalves – Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto
#
#
#------------------------------------------------------------------------------#

#----------------------------------MODULOS-------------------------------------

# Carregando todas as funções e variáveis
source variables.sh
source function.sh

# Módulo *: Ajuda com 'h'
show_help() {
    echo "Uso: ./sarik.sh [opções]"
    echo "Opções:"
    echo " –h, ––help Mostrar esta ajuda"
    echo " –mn, ––manual Configurar políticas de rede manualmente (interface de saída)"
    echo " –D, ––delete Exclusão de todas políticas de rede automaticamente"
    echo " –l, ––view Visualização políticas de rede"
    echo " –d, ––del Exclusão de políticas de rede individualmente"
    echo " –i, ––ingress Configurar políticas de rede manualmente (interface de entrada)"
    echo " –m, ––monitor Monitorar o impacto das políticas de rede"
    echo " –b, ––backup Fazer backup das políticas de rede atuais"
    echo " –v, ––validate Validar as políticas de rede (EXPERIMENTAL)"
}
```

Algoritmo 17 Módulo 10 - Geração automática das políticas de rede

```
#!/usr/bin/env bash
#
# SARIK – SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
#
# SITE: https://sarik.org
# Autor: Jonathan G.P. dos Santos
# Manutenção: Jonathan G.P. dos Santos
#
#
#------------------------------------------------------------------------------------#
#Histórico
#
# V1.05 05/10/2023, Jonathan G.P. dos Santos
# – Add Module 10
#------------------------------------------------------------------------------------#
#Testado em:
# bash 5.0.17
#
#
#------------------------------------------------------------------------------------#
#Agradecimentos:
#
#
# Dr. Vinícius Pereira Gonçalves – Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto
#
#
#------------------------------------------------------------------------------------#

#----------------------------------MODULOS-------------------------------------------

# Carregando todas as funções e variáveis
source variables.sh
source function.sh

# Módulo Padrão: Configuração Automática de Regras


configure_firewall_rules_auto() {
   echo "Configurando regras de firewall automaticamente..."
sleep 5
# Especificando a namespace
NAMESPACE=$VALUE_NAMESPACE

# Coletando informações dos pods
```

```
kubectl get pods –n $NAMESPACE | awk '{if(NR>1)print $1}' > ngetPods.txt
readarray pods < ngetPods.txt

# Armazena cada linha do resultado do comando em um array
resultado1=($(kubectl get services –n $NAMESPACE –o jsonpath='{range
    .items[*]}{.metadata.name}{"\n"}{end}'))

# Inicializa o array de colunas para serem usados no JSON do networkPolicy
colunas=()

# Itera sobre cada linha do resultado
for linha in "${resultado1[@]}"; do
    # Armazena cada coluna da linha em um array
    colunas+=($(echo $linha))
done

CONT3=0

#Criação do diretorio para as regras

if [ –d "policies" ]; then
    #Deletetando regras existentes no diretório
    rm –R policies/* 2>/dev/null
  else
    #criando o diretório
    mkdir policies
fi


# Criando array com as portas a serem bloqueadas
BLOCKED_PORTS=(7 80 443 22)

EXISTING_POLICIES=$(kubectl get networkpolicy –n $NAMESPACE ––no–headers 2>/dev/null)

if [[ ! –z "$EXISTING_POLICIES" ]]; then
   echo "Já existem políticas de rede para o namespace $NAMESPACE."
   read –p "Deseja continuar e aplicar novas regras? (y/N): " choice
   case "$choice" in
     y|Y ) echo "Continuando..."
           sleep 2
progress–bar2 $CONT2
# Loop para percorrer cada pod
for pod in ${pods[@]}; do
    # Loop para percorrer cada porta a ser bloqueada
    for i in "${BLOCKED_PORTS[@]}"; do
        # Cria a regra de egress para a porta atual
        cat > policies/$(echo block–egress–$pod–$i.yaml) <<EOF
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: block-ports-egress-$pod-$i
  namespace: $NAMESPACE
spec:
  podSelector:
    matchLabels:
      app: $(echo ${colunas[$CONT3]})
  policyTypes:
  - Egress
  egress:
  - ports:
    - protocol: TCP
      port: $i
  - to:
    - podSelector:
        matchLabels:
          app: $(echo ${colunas[$CONT3]})
EOF
    done
    ((CONT3=CONT3+1))
done

#Aplicando as políticas de rede no cluster
kubectl apply -f policies/

#Chamando a função para imprimir mensagem de conclusão
sarik-msg-default;;

    * ) echo "Operação cancelada pelo usuário."; exit 1;;
  esac
 else
  echo "Não há políticas de rede existentes para o namespace $NAMESPACE."
  echo "Prosseguindo com a aplicação das novas regras..."
progress-bar2 $CONT2
# Loop para percorrer cada pod
for pod in ${pods[@]}; do
    # Loop para percorrer cada porta a ser bloqueada
    for i in "${BLOCKED_PORTS[@]}"; do
        # Cria a regra de egress para a porta atual
        cat > policies/$(echo block-egress-$pod-$i.yaml) <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: block-ports-egress-$pod-$i
  namespace: $NAMESPACE
```

```
spec:
  podSelector:
    matchLabels:
      app: $(echo ${colunas[$CONT3]})
  policyTypes:
  – Egress
  egress:
  – ports:
    – protocol: TCP
      port: $i
  – to:
    – podSelector:
        matchLabels:
          app: $(echo ${colunas[$CONT3]})
EOF
    done
    ((CONT3=CONT3+1))
done

#Aplicando as políticas de rede no cluster
kubectl apply –f policies/
sleep 5
clear
echo " "
echo $(echo –e ${VERDE}) "System configuration done!!" $(echo –e ${BRANCO})
echo " "
cat << "EOF"
    ______ ______ ______ _ __
   / ____| /\ | __ \ |_ _| | |/ /
  | (___ /  \ | |__) | | | | ' /
   \___ \ / /\ \ | _ / | | | <
   ____) | / ____ \ | | \ \ _| |_ | . \
  |_____/ /_/ \_\ |_| \_\ |_____| |_|\_\

EOF
echo " "
echo "================================================================================"
echo "============ AUTOMATIC SECURITY OF RULES ON IPTABLES IN KUBERNETES ============="
echo "============ By @jonathamgg and @jonathan | DC ================="
echo "================================================================================"
fi
}

#––––––––––––––––––––––––––––––––––MODULOS––––––––––––––––––––––––––––––––––––––––––
```

Algoritmo 18 Funções

```
#!/usr/bin/env bash
#
# SARIK – SEGURANÇA AUTOMÁTICA DE REGRAS EM IPTABLES EM KUBERNETES
#
# SITE: https://sarik.org
# Autor: Jonathan G.P. dos Santos
# Manutenção: Jonathan G.P. dos Santos
#
#
#------------------------------------------------------------------------------------#
#Histórico
#
# V1.05 05/10/2023, Jonathan G.P. dos Santos
# – Add function
#------------------------------------------------------------------------------------#
#Testado em:
# bash 5.0.17
#
#
#------------------------------------------------------------------------------------#
#Agradecimentos:
#
#
# Dr. Vinícius Pereira Gonçalves – Orientador do projeto
# Dr. Geraldo Pereira Rocha Filho – Coorientador do projeto
#
#
#------------------------------------------------------------------------------------#

#----------------------------------FUNÇÕES-------------------------------------------
#

#Progress-bar with problem
progress-bar() {
  local duration=${1}

    already_done() { for ((done=0; done<$elapsed; done++)); do printf ""; done }
    remaining() { for ((remain=$elapsed; remain<$duration; remain++)); do printf " "; done }
    percentage() { printf "| %s%%" $(( (($elapsed)*100)/($duration)*100/100 )); }
    clean_line() { printf "\r"; }

  for (( elapsed=1; elapsed<=$duration; elapsed++ )); do
      already_done; remaining; percentage
      sleep 1
```

```
45          clean_line
46      done
47      clean_line
48  }
49  #Progress-bar ok
50  progress-bar2() {
51  [[ $# -ne 1 ]] && error 1
52  [[ $1 =~ ^[0-9]+$ ]] || error 2
53
54  duration=${1}
55  barsize=$((`tput cols` - 7))
56  unity=$(($barsize / $duration))
57  increment=$(($barsize%$duration))
58  skip=$(($duration/($duration-$increment)))
59  curr_bar=0
60  prev_bar=
61  for (( elapsed=1; elapsed<=$duration; elapsed++ ))
62  do
63      # Elapsed
64  prev_bar=$curr_bar
65      let curr_bar+=$unity
66      [[ $increment -eq 0 ]] || {
67          [[ $skip -eq 1 ]] &&
68              { [[ $(($elapsed%($duration/$increment))) -eq 0 ]] && let curr_bar++; } ||
69          { [[ $(($elapsed%$skip)) -ne 0 ]] && let curr_bar++; }
70      }
71      [[ $elapsed -eq 1 && $increment -eq 1 && $skip -ne 1 ]] && let curr_bar++
72      [[ $(($barsize-$curr_bar)) -eq 1 ]] && let curr_bar++
73      [[ $curr_bar -lt $barsize ]] || curr_bar=$barsize
74      for (( filled=0; filled<=$curr_bar; filled++ )); do
75          printf ""
76      done
77
78      # Remaining
79      for (( remain=$curr_bar; remain<$barsize; remain++ )); do
80          printf " "
81      done
82
83      # Percentage
84      printf "| %s%%" $(( ($elapsed*100)/$duration))
85
86      # Return
87      sleep 1
88      printf "\r"
89  done
90  printf "\n"
91  }
```

```
#

# Função para criar regra de bloqueio na interface de saída
create_block_rule() {
  local pod=$1
  local namespace=$2
  local protocol=$3
  local port=$4

  cat > "block-$pod-$port.yaml" <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: block-$port-$pod
  namespace: $namespace
spec:
  podSelector:
    matchLabels:
      app: $(echo $pod | cut -d'-' -f1)
  policyTypes:
  - Egress
  egress:
  - ports:
    - protocol: $protocol
      port: $port
  - to:
    - podSelector:
        matchLabels:
          app: $(echo $pod | cut -d'-' -f1)
EOF
}

# Função para criar regra de bloqueio na interface de entrada
create_ingress_block_rule() {
  local pod=$1
  local namespace=$2
  local port=$3
  local protocol=$4

  # Criar o arquivo YAML para a política de rede
  cat > ingress-block-$pod-$port.yaml <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: block-$port-ingress-$pod
  namespace: $namespace
spec:
```

```
    podSelector:
      matchLabels:
        app: $(echo $pod | cut –d'–' –f1)
    policyTypes:
    – Ingress
    ingress:
    – ports:
      – protocol: $protocol
        port: $port
    – to:
      – podSelector:
          matchLabels:
            app: $(echo $pod | cut –d'–' –f1)
EOF
}

# Função para excluir regra de bloqueio
delete_block_rule() {
    local policy_name=$1
    local namespace=$2

    # Excluir a política de rede
    kubectl delete networkpolicy $policy_name –n $namespace
}

# Função para monitorar políticas de rede
monitor_policies() {
    echo "Monitorando políticas de rede..."
    sleep 2
    # Listar todas as namespaces
    namespaces=$(kubectl get namespaces ––no–headers –o custom–columns=":metadata.name")

    echo "Escolha uma namespace para monitorar as políticas de rede:"

    select ns in $namespaces; do
      if [ –n "$ns" ]; then
        echo "Você selecionou a namespace $ns."

        # Verificar se existem políticas de rede na namespace selecionada
        policies=$(kubectl get networkpolicy –n $ns ––no–headers –o
            custom–columns=":metadata.name")

        if [ –z "$policies" ]; then
          echo "Não há políticas de rede para a namespace $ns."
        else
          echo "Políticas de rede na namespace $ns:"
          echo "$policies"
```

```
185
186             # Detalhes adicionais sobre cada política
187             for policy in $policies; do
188               echo "Detalhes da política $policy:"
189               kubectl describe networkpolicy $policy –n $ns
190             done
191           fi
192           break
193         else
194           echo "Seleção inválida. Tente novamente."
195         fi
196       done
197     }
198
199     # Função para fazer backup das políticas de rede
200     backup_policies() {
201       echo "Realizando backup das políticas de rede..."
202       sleep 3
203       # Checa se o diretório backup_policies existe
204       if [ –d "backup_policies" ]; then
205         #Deletetando arquivos existentes no diretório
206         rm –R backup_policies/* 2>/dev/null
207       else
208         #criando o diretório
209         mkdir backup_policies
210       fi
211
212       # Listar todas as namespaces
213       namespaces=$(kubectl get namespaces ––no–headers –o custom–columns=":metadata.name")
214
215       echo "Escolha uma namespace para fazer backup das políticas de rede:"
216
217       select ns in $namespaces; do
218         if [ –n "$ns" ]; then
219           echo "Você selecionou a namespace $ns."
220
221           # Verificar se existem políticas de rede na namespace selecionada
222           policies=$(kubectl get networkpolicy –n $ns ––no–headers –o
                custom–columns=":metadata.name")
223
224           if [ –z "$policies" ]; then
225             echo "Não há políticas de rede para a namespace $ns."
226           else
227             echo "Fazendo backup das políticas de rede na namespace $ns..."
228
229             # Fazer backup de cada política
230             for policy in $policies; do
```

```
231             kubectl get networkpolicy $policy -n $ns -o yaml > backup_policies/$policy.yaml
232           done
233
234           echo "Backup concluído. Os arquivos foram salvos no diretório 'backup_policies'."
235         fi
236
237         break
238       else
239         echo "Seleção inválida. Tente novamente."
240       fi
241     done
242 }
243
244 # Função para validar políticas de rede
245 validate_policies() {
246     echo "Validando políticas de rede..."
247     sleep 3
248     # Listar todas as namespaces
249     namespaces=$(kubectl get namespaces --no-headers -o custom-columns=":metadata.name")
250
251     echo "Escolha uma namespace para validar as políticas de rede:"
252
253     select ns in $namespaces; do
254       if [ -n "$ns" ]; then
255         echo "Você selecionou a namespace $ns."
256
257         # Verificar se existem políticas de rede na namespace selecionada
258         policies=$(kubectl get networkpolicy -n $ns --no-headers -o
                custom-columns=":metadata.name")
259
260         if [ -z "$policies" ]; then
261           echo "Não há políticas de rede para a namespace $ns."
262         else
263           echo "Validando políticas de rede na namespace $ns..."
264
265           # Validar cada política
266           for policy in $policies; do
267             # Aqui, são inseridas as regras de validação.
268
269             # Verificar se a política permite o tráfego de qualquer origem (inseguro)
270             is_insecure=$(kubectl get networkpolicy $policy -n $ns -o json | jq
                  '.spec.ingress[0].from[0].ipBlock.cidr == "0.0.0.0/0"')
271
272             # Verificar se a política permite tráfego para qualquer porta (inseguro)
273             is_any_port=$(kubectl get networkpolicy $policy -n $ns -o json | jq
                  '.spec.ingress[0].ports[0].port == null')
274
```

```
275             # Verificar se a política não especifica um tipo de política (Egress ou Ingress)
276             is_policy_type_missing=$(kubectl get networkpolicy $policy –n $ns –o json | jq
                    '.spec.policyTypes == null')
277
278             if [ "$is_insecure" == "true" ]; then
279               echo "A política $policy permite tráfego de qualquer origem. Isso é inseguro."
280             elif [ "$is_any_port" == "true" ]; then
281               echo "A política $policy permite tráfego para qualquer porta. Isso é inseguro."
282             elif [ "$is_policy_type_missing" == "true" ]; then
283               echo "A política $policy não especifica um tipo de política (Egress ou Ingress). Isso
                      pode ser confuso."
284             else
285               echo "A política $policy parece segura."
286             fi
287           done
288
289           echo "Validação concluída."
290         fi
291
292         break
293       else
294         echo "Seleção inválida. Tente novamente."
295       fi
296     done
297 }
298
299
300 #================================================================================================
301 sarik–msg–exclusao(){
302 progress–bar2 $CONT2
303 clear
304 echo " "
305 echo $(echo –e ${VERDE}) "System network policies delete done!!" $(echo –e ${BRANCO})
306 echo " "
307 cat << "EOF"
308     _____ _____ _____ _ __
309    / ____| /\ | __ \ |_ _| | |/ /
310   | (___ /  \ | |__) | | | | ' /
311    \___ \ / /\ \ | _ / | | | <
312    ____) | / ____ \ | | \ \ _| |_ | . \
313   |_____/ /_/ \_\ |_| \_\ |_____| |_|\_\
314
315 EOF
316 echo " "
317 echo
        "================================================================================"
318 echo "============ AUTOMATIC SECURITY OF RULES ON IPTABLES IN KUBERNETES
```

```
319 ==================="
320 echo "============ By @jonathamgg and @jonathan | DC ================="
    echo
        "================================================================================================"
321 }
322
323 sarik-msg-view(){
324 progress-bar2 $CONT2
325 clear
326 echo " "
327 echo $(echo -e ${VERDE}) "System network policies view done!!" $(echo -e ${BRANCO})
328 echo " "
329 cat << "EOF"
330     _____ _____ _____ _ __
331    / ____| /\ | __ \ |_ _| | |/ /
332   | (___ /  \ | |__) | | | | ' /
333    \___ \ / /\ \ | _ / | | | <
334    ____) | / ____ \ | | \ \ _| |_ | . \
335   |_____/ /_/ \_\ |_| \_\ |_____| |_|\_\
336
337 EOF
338 echo " "
339 echo
        "================================================================================================"
340 echo "============ AUTOMATIC SECURITY OF RULES ON IPTABLES IN KUBERNETES
        ==================="
341 echo "============ By @jonathamgg and @jonathan | DC ================="
342 echo
        "================================================================================================"
343 }
344
345 sarik-msg-default(){
346 clear
347 echo " "
348 echo $(echo -e ${VERDE}) "System configuration done!!" $(echo -e ${BRANCO})
349 echo " "
350 cat << "EOF"
351     _____ _____ _____ _ __
352    / ____| /\ | __ \ |_ _| | |/ /
353   | (___ /  \ | |__) | | | | ' /
354    \___ \ / /\ \ | _ / | | | <
355    ____) | / ____ \ | | \ \ _| |_ | . \
356   |_____/ /_/ \_\ |_| \_\ |_____| |_|\_\
357
358 EOF
359 echo " "
360 echo
```

```
    "================================================================================"
361 echo "============ AUTOMATIC SECURITY OF RULES ON IPTABLES IN KUBERNETES
    ================="
362 echo "============ By @jonathamgg and @jonathan | DC ================="
363 echo
    "================================================================================"
364 }
365
366 #-------------------------------------------------------------------------------#
```