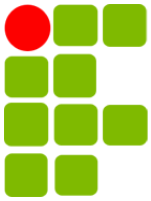


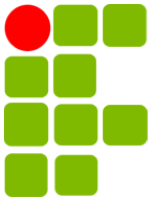
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Campus Campina Grande
Coordenação do Curso Superior de Tecnologia em Telemática

AUTOMAÇÃO DE REDES LOCAIS COM EDGE COMPUTING: UMA PROPOSTA PRÁTICA COM MICROK8S

LUCAS SILVA COSTA

Orientador: Marcelo Portela Sousa

Campina Grande, agosto de 2025



Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Campus Campina Grande
Coordenação do Cursos Superior de Tecnologia em Telemática

AUTOMAÇÃO DE REDES LOCAIS COM EDGE COMPUTING: UMA PROPOSTA PRÁTICA COM MICROK8S

LUCAS SILVA COSTA

Monografia apresentada à Coordenação do Curso de Telemática do IFPB - Campus Campina Grande, como requisito parcial para conclusão do curso de Tecnologia em Telemática.

Orientador: Marcelo Portela Sousa

Campina Grande, agosto de 2025

Catalogação na fonte:
Ficha catalográfica elaborada por Gustavo César Nogueira da Costa - CRB 15/479

C837a Costa, Lucas Silva

Automação de redes locais com edge computing: uma proposta prática com MicroK8s / Lucas Silva Costa. – 2025.
57 f.: il.

Trabalho de Conclusão de Curso (Graduação em Tecnologia em Telemática) - Instituto Federal da Paraíba, 2025.
Orientador: Prof. Dr. Marcelo Portela Sousa.

1 Redes locais – Automação. 2. Computação em borda. 3. MicroK8s. 4. Orquestração de contêineres. 5. Monitoramento de redes. I. Sousa, Marcelo Portela. II. Título.

CDU 004.75

AUTOMAÇÃO DE REDES LOCAIS COM EDGE COMPUTING: UMA PROPOSTA PRÁTICA COM MICROK8S

LUCAS SILVA COSTA

MARCELO PORTELA SOUSA
Orientador

EWERTON ROMULO SILVA CASTRO
Membro da Banca

VICTOR ANDRÉ PINHO DE OLIVEIRA
Membro da Banca

Campina Grande, Paraíba, Brasil
Agosto/2025

Dedico este trabalho Àquele que é o verdadeiro autor de todas as coisas: Deus, cuja vontade é boa, perfeita e agradável. A Ele entrego não apenas este projeto, mas toda a minha caminhada, na certeza de que viver para a Sua glória é o maior propósito da existência.

Dedico também à minha família, minha base e inspiração, com o desejo sincero de ser bênção em suas vidas, servindo com amor, respeito e gratidão.

E a todos os meus próximos, a quem sou chamado a amar e servir, que este trabalho represente não apenas um avanço acadêmico, mas um testemunho prático de que é possível unir fé, conhecimento e serviço com excelência.

Que cada passo dado nesta jornada reflita o desejo de fazer a vontade de Deus e ser instrumento de transformação e luz onde Ele me enviar.

João respondeu, e disse: O homem não pode receber coisa alguma, se não lhe for dada do céu.

João 3:27 - Bíblia Sagrada

Agradecimentos

"É necessário que Ele cresça e que eu diminua."(João 3:30)

Em primeiro lugar, agradeço a Deus, por Sua graça, sabedoria e fidelidade em cada etapa desta jornada. Sem Ele, nada faria sentido. Que este trabalho sirva para exaltá-Lo e refletir a Sua vontade em mim.

Agradeço à minha família, pelo amor constante, apoio incondicional e por serem minha base em todos os momentos — especialmente nos dias de desafios e incertezas. Cada palavra de encorajamento, cada oração e cada gesto de cuidado foram fundamentais.

Aos professores da banca examinadora, minha sincera gratidão pela dedicação, paciência e contribuições valiosas ao longo deste processo. O conhecimento e o olhar crítico de cada um enriqueceram significativamente este trabalho.

E a todos que me apoiaram direta ou indiretamente — amigos, colegas, irmãos de fé —, meu muito obrigado. Que cada gesto de ajuda, incentivo ou companhia durante esta caminhada seja recompensado por Aquele que tudo vê.

Que este trabalho não seja apenas um cumprimento acadêmico, mas uma semente de transformação para mim e para todos ao meu redor, para a glória de Deus.

Resumo

A crescente demanda por redes mais autônomas, eficientes e resilientes tem impulsionado o uso de computação em borda (*Edge Computing*) como alternativa aos modelos centralizados de computação em nuvem. Este trabalho propõe uma arquitetura prática de automação de redes locais baseada em um nó *edge* utilizando *MicroK8s* e um roteador *MikroTik RB750Gr3*. A proposta foi implementada em um ambiente experimental composto por um *Raspberry Pi 5* executando serviços em contêineres (como *Prometheus* e *Grafana*) para monitoramento e automação. Foram realizadas avaliações quantitativas e qualitativas considerando métricas como latência, consumo de recursos, escalabilidade e tolerância a falhas. Os resultados demonstram que a arquitetura com nó *edge* oferece ganhos significativos em desempenho, visibilidade e reatividade, superando as limitações do roteador isolado. A solução proposta se mostrou viável para redes locais de pequeno e médio porte, especialmente em contextos com restrições de custo e conectividade. Ao final, são sugeridas extensões envolvendo segurança avançada, inteligência artificial e escalonamento distribuído.

Palavras-chave: Automação de Redes. Computação em Borda. *MicroK8s*. Redes Locais. MikroTik.

Abstract

The growing demand for more autonomous, efficient, and resilient networks has driven the adoption of *Edge Computing* as an alternative to traditional cloud-based architectures. This work proposes a practical local network automation architecture using a Raspberry Pi 5 as an *edge* node running *MicroK8s* alongside a MikroTik RB750Gr3 router. The implementation included containerized services such as *Prometheus* and *Grafana* for monitoring and automation. Experimental evaluations were carried out based on metrics such as latency, resource consumption, scalability, and fault tolerance. The results show that the edge-enabled architecture significantly improves performance, visibility, and reactivity compared to using the router alone. The proposed solution proved viable for small to medium-sized local networks, especially in environments with cost and connectivity constraints. Future work may explore advanced security, machine learning-based automation, and multi-node distributed scaling.
Keywords: Network Automation. Edge Computing. *MicroK8s*. Local Networks. MikroTik.

Sumário

Lista de Abreviaturas

5G	*Fifth Generation Mobile Network* (Quinta Geração de Rede Móvel)
API	*Application Programming Interface* (Interface de Programação de Aplicações)
APIs RESTful	*Representational State Transfer Application Programming Interfaces* (Interfaces de Programação baseadas em Transferência Representacional de Estado)
ARM	*Advanced RISC Machine* (Máquina Avançada com Conjunto de Instruções Reduzidas)
CLI	*Command-Line Interface* (Interface de Linha de Comando)
CPU	*Central Processing Unit* (Unidade Central de Processamento)
DHCP	*Dynamic Host Configuration Protocol* (Protocolo de Configuração Dinâmica de Endereços)
DNS	*Domain Name System* (Sistema de Nomes de Domínio)
GDPR	*General Data Protection Regulation* (Regulamento Geral de Proteção de Dados)
GPIO	*General Purpose Input/Output* (Entrada/Saída de Uso Geral)
GPU	*Graphics Processing Unit* (Unidade de Processamento Gráfico)
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
IA	*Artificial Intelligence* (Inteligência Artificial)
IoT	*Internet of Things* (Internet das Coisas)
IP	*Internet Protocol* (Protocolo de Internet)
Kubectl	*Kubernetes Control* (Ferramenta de Controle do Kubernetes)
Kubernetes	*Kubernetes* (Sistema de Orquestração de Contêineres)
LGPD	Lei Geral de Proteção de Dados
mTLS	*Mutual Transport Layer Security* (Segurança Mútua na Camada de Transporte)
NAT	*Network Address Translation* (Tradução de Endereço de Rede)
NGINX	*Engine X* (Servidor Web e Proxy Reverso de Alto Desempenho)
QoE	*Quality of Experience* (Qualidade de Experiência)
QoS	*Quality of Service* (Qualidade de Serviço)
REST	*Representational State Transfer* (Transferência Representacional de Estado)
SNMP	*Simple Network Management Protocol* (Protocolo Simples de Gerenciamento de Rede)
SSH	*Secure Shell* (Shell Seguro)
USB	*Universal Serial Bus* (Barramento Serial Universal)
VLAN	*Virtual Local Area Network* (Rede Local Virtual)
VPN	*Virtual Private Network* (Rede Privada Virtual)
YAML	*YAML Ain't Markup Language* (YAML Não é Linguagem de Marcação)

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A crescente complexidade das aplicações distribuídas e a demanda por serviços com baixa latência têm evidenciado limitações nos modelos tradicionais baseados exclusivamente em computação em nuvem. Nesse contexto, a computação de borda, (*Edge Computing*), surge como uma abordagem alternativa e complementar, que descentraliza o processamento ao aproximá-lo fisicamente das fontes de dados, como sensores e dispositivos conectados. Essa mudança arquitetural visa reduzir latência, economizar largura de banda e ampliar a autonomia dos sistemas locais.

Historicamente, a arquitetura de redes seguiu uma orientação centralizada, onde servidores em *data center* (centro de dados) processavam dados provenientes de clientes remotos. Contudo, com o advento da *IoT* (*Internet of Things* - Internet das Coisas) e o crescimento exponencial do volume de dados gerados, essa estratégia tornou-se ineficiente para muitos cenários críticos, como sistemas industriais, cidades inteligentes e aplicações de saúde, nos quais decisões devem ser tomadas em milissegundos [Dastjerdi *et al.* 2016].

A borda, portanto, representa uma evolução natural da computação em nuvem, distribuindo capacidades de processamento, armazenamento e rede na camada intermediária entre a nuvem e os dispositivos finais, possibilitando uma resposta em tempo real e maior controle sobre os dados sensíveis [Luo *et al.* 2021].

Neste trabalho, propõe-se o desenvolvimento de uma proposta prática de automação de redes locais com base em *Edge computing* (Computação de borda), utilizando a ferramenta *MicroK8s* (Distribuição Leve do *Kubernetes* para Ambientes de Desenvolvimento) como núcleo da orquestração de serviços em um nó de borda [Canonical Ltd. 2025]. Para isso, será empregada uma infraestrutura composta por um Raspberry Pi (representando o nó *edge*) e um roteador *MikroTik* RB750Gr3, responsável pela roteabilidade, monitoramento e integração com serviços automatizados [MikroTik 2016].

A principal contribuição deste trabalho consiste em demonstrar como uma rede local pode ser automatizada, monitorada e gerenciada com tecnologia de borda e *software* (programa) livre, criando uma arquitetura modular, acessível e funcional. Serão discutidas estratégias de implantação de *containers*, automação com *scripts* (roteiros) e APIs *Application Program-*

ming Interface (Interface de Programação de Aplicações) de roteadores e integração com ferramentas de monitoramento como Grafana e Prometheus.

O estado da arte evidencia um aumento significativo em pesquisas voltadas à descentralização computacional, com arquiteturas tricamadas – *thing layer, edge layer e cloud layer* (Camada de Coisas, Camada de Borda, Camada de Nuvem) – que buscam responder às exigências de escalabilidade, segurança e desempenho. A utilização de MicroK8s como alternativa leve ao *Kubernetes* (Orquestrador de Contêineres) tradicional tem se mostrado viável e eficaz em contextos de *IoT Internet of Things* (Internet das Coisas) e *Edge* (Borda) [Liu *et al.* 2019].

A infraestrutura experimental contará com a implantação de serviços locais, manipulação automatizada de tráfego via roteador *MikroTik*, e orquestração de *container* (recipiente) que executam funções de controle e resposta a eventos da rede, de forma totalmente autônoma e *offline* (desconectado).

Como resultado esperado, pretende-se validar a hipótese de que é possível automatizar redes locais de forma eficiente com recursos limitados, aproveitando o paradigma da computação de borda. Além disso, busca-se demonstrar os benefícios de tal arquitetura em cenários reais e aplicáveis à educação, indústria e residências conectadas.

1.1 Justificativa e Relevância do Trabalho

A escolha do tema "Automação de Redes Locais com *Edge Computing*: Uma Proposta Prática com *MicroK8s*" está diretamente relacionada à crescente necessidade de soluções descentralizadas, seguras e eficientes para redes locais, especialmente em ambientes onde o acesso à internet é limitado ou onde a autonomia do sistema é um requisito essencial. A evolução do *IoT*, aliada à demanda por respostas em tempo real e redução da dependência de data centers, torna a computação de borda uma alternativa viável e estratégica.

Esse trabalho é relevante para a área de redes de computadores, automação e sistemas embarcados, pois apresenta uma solução prática baseada em ferramentas acessíveis e modernas — como o *MicroK8s* e o *Raspberry Pi* — integradas a roteadores *MikroTik*, amplamente usados em redes corporativas e residenciais. A proposta pode beneficiar cenários como:

- Redes locais inteligentes para escolas, residências e laboratórios;
- Ambientes industriais com baixa conectividade externa;
- Pequenos provedores de internet que desejam automatizar processos e reduzir latência.

Além disso, o tema contribui com a formação técnica de profissionais e pesquisadores da área de computação, ao apresentar uma arquitetura realista, de baixo custo e replicável, que combina conceitos de orquestração de *containers*, automação de rede e computação distribuída.

1.2 Objetivos

1.2.1 Objetivo Geral

Desenvolver uma proposta prática de automação de redes locais utilizando a abordagem de computação de borda, com ênfase na implantação e gestão de serviços em um ambiente local baseado em *MicroK8s*, *Raspberry Pi* e roteador *MikroTik*.

1.2.2 Objetivos Específicos

- Investigar e apresentar os conceitos de computação de borda e suas aplicações em redes locais;
- Implementar um *cluster Kubernete* (Conjunto de Nós do Kubernetes) leve com MicroK8s em um nó *Edge Raspberry Pi*;
- Configurar um roteador *MikroTik* para integração com serviços automatizados;
- Criar *scripts* (roteiros) e testes para automação de tarefas de rede em ambiente *offline* (desconectado);
- Avaliar a viabilidade e desempenho da solução em termos de latência, disponibilidade e escalabilidade.

1.3 Metodologia

A metodologia adotada neste trabalho baseia-se em uma abordagem experimental, com o objetivo de validar a eficiência da arquitetura proposta para automação de redes locais com nó de borda. Para isso, foram conduzidos testes controlados comparando dois cenários distintos:

- **Cenário 1 (base)**: rede gerenciada exclusivamente pelo roteador MikroTik RB750Gr3, executando todas as funções de controle e monitoramento de forma isolada.
- **Cenário 2 (proposto)**: rede gerenciada pela RB (Roteador de Borda estudado) integrada a um nó de borda (*Edge Node* (Nó de Processamento na Borda da Rede)) baseado em Raspberry Pi 5, executando um *cluster MicroK8s* para orquestração de contêineres, coleta de métricas com *Prometheus*, visualização em *Grafana* e execução de automações reativas por meio de *scripts* e *API REST* [Canonical Ltd. 2025] [Raspberry Pi Ltd 2025] [MikroTik 2025].

A comparação entre os cenários segui um protocolo experimental que busca avaliar a viabilidade e desempenho, controlando variáveis externas e repetindo os testes sob as mesmas condições [Pereira *et al.* 2018]. Os principais pontos do procedimento foram:

- **Definição das métricas avaliadas**: latência média de execução de comandos, consumo de CPU *Central Processing Unit* (Unidade Central de Processamento) e RAM *Random Access Memory* (Memória de Acesso Aleatório), tráfego de rede (Tx/Rx), tempo de recuperação após falhas e escalabilidade (variação de desempenho conforme o número de *pods* executados).

- **Repetição dos testes**: cada experimento foi executado no mínimo cinco vezes por cenário, descartando resultados anômalos e calculando média e desvio padrão para maior precisão.

- **Ferramentas de coleta e análise**: o *Prometheus* (Plataforma de Monitoramento e Coleta de Métricas) foi utilizado para coleta contínua de métricas, enquanto o *Grafana* (Plataforma de Visualização e Análise de Dados) forneceu visualizações gráficas. Scripts em *RouterOS* (Sistema Operacional da *MikroTik* para Roteadores) e *Python* (Linguagem de Programação de Alto Nível) registraram eventos e tempos de resposta.

- **Controle de variáveis**: os testes foram realizados em ambiente de laboratório, com topologia fixa e tráfego gerado artificialmente de forma idêntica para ambos os cenários, evitando interferências externas.

- **Critérios de avaliação**: os resultados foram analisados comparativamente, identificando ganhos de desempenho, redução de latência, aumento na visibilidade da rede e capacidade de recuperação frente a falhas.

Essa abordagem experimental permitiu comprovar de forma consistente se as melhorias observadas no cenário proposto estavam diretamente relacionadas à introdução do nó *Edge*, garantindo a validade dos resultados e fornecendo dados quantitativos para a discussão apresentada nos capítulos posteriores.

1.3.1 Procedimentos Metodológicos

A metodologia deste trabalho foi dividida em seis etapas principais:

1. Levantamento Teórico

Foi realizada uma revisão da literatura sobre os temas centrais da pesquisa — computação de borda, orquestração de *containers* (recipientes), automação de redes locais e roteadores *MikroTik* — com o objetivo de embasar conceitualmente a proposta e justificar tecnicamente as escolhas feitas.

2. Planejamento Experimental

Com base no levantamento teórico, foi definida a arquitetura lógica da solução, especificando os componentes necessários *hardware e software* (maquina e programa), os serviços a serem

implantados, os critérios de avaliação e o ambiente de teste.

3. Montagem do Ambiente de Teste

Ferramentas e Tecnologias Utilizadas

Categoria	**Ferramenta/Equipamento**
Hardware	Raspberry Pi 5, MikroTik RB750Gr3
Sistema Operacional	Ubuntu Server 24.04 LTS
Orquestração	MicroK8s (Snap Canonical)
Containers	Docker (compatível com MicroK8s)
Monitoramento	Prometheus, Grafana
Automação	*Scripts* RouterOS + API MikroTik
Linguagens	*Shell Script*, YAML, RouterOS *Script*
Documentação	VS Code, LaTeX/Overleaf ou Word

Tabela 1.1: *Ferramentas e tecnologias utilizadas na implementação do experimento*

A Tabela 1.1 apresenta as principais ferramentas e tecnologias empregadas na implementação do experimento. Na categoria ***Hardware*** (maquina), foi utilizado o *Raspberry Pi 5* (Computador de Placa Única de Quinta Geração da Série Raspberry Pi), que atuou como nó de borda, e o roteador *MikroTik RB750Gr3* (Roteador da *MikroTik* com 5 Portas *Gigabit* e Processador de Alto Desempenho), responsável pelo controle e gerenciamento do tráfego local.

Quanto ao **Sistema Operacional**, optou-se pelo *Ubuntu Server 24.04 LTS* (Sistema Operacional Linux para Servidores com Suporte de Longo Prazo), devido à sua estabilidade, leveza e compatibilidade com arquiteturas *ARM* (Máquinas Avançadas com Conjunto Reduzido de Instruções), além do suporte nativo ao *Snap* (Sistema de Pacotes e Implantação de Aplicativos da Canonical), essencial para a instalação do *MicroK8s*.

Na camada de **Orquestração**, foi empregado o *MicroK8s* (distribuição leve do *Kubernetes*, mantida pela Canonical), instalado via *Snap*. Para a execução de aplicações em contêineres, foi utilizado o *Docker* (Plataforma de Contêineres para Desenvolvimento, Distribuição e Execução de Aplicações), totalmente compatível com o *MicroK8s*.

O **Monitoramento** foi realizado por meio do *Prometheus*, responsável pela coleta de métricas, e do *Grafana*, utilizado para a visualização e análise dessas métricas em *dashboards* (Painéis de Visualização de Dados) interativos.

No que se refere à **Automação**, empregaram-se *scripts* desenvolvidos em *RouterOS Script*, integrados à *API REST Representational State Transfer Application Programming Interface* (Interface de Programação de Aplicações baseada em Transferência Representacional de Estado) do *MikroTik*, possibilitando a execução de ações reativas, como bloqueio de *Internet Protocol* (Protocolo de Internet) e alteração de regras de firewall *Firewall* (Barreira de Proteção de Rede) [MikroTik 2025].

As **Linguagens** utilizadas incluíram *Shell Script* para automações no sistema operacional, *YAML* para configuração de recursos do *Kubernetes* e *RouterOS Script* para programação no roteador [Canonical Ltd. 2025].

Por fim, para a **Documentação**, foram utilizadas ferramentas como o *Visual Studio Code (VS Code)* e o LaTeX (no *Overleaf*) para a elaboração acadêmica, bem como o *Microsoft Word* para edições complementares.

4. Implantação dos Serviços e Automação

Foram implantados serviços containerizados no *MicroK8s, Prometheus, Grafana, NGINX* (Servidor *Web* e *Proxy* Reverso de Alto Desempenho) e serviços customizados para automação da rede. *Scripts RouterOS* foram programados na RB para redirecionamento de tráfego e automação de tarefas de rede, integrados via API ao ambiente orquestrado.

Execução dos Testes

A solução foi testada com foco nos seguintes critérios:

- Latência de resposta dos serviços;
- Disponibilidade em ambiente *offline* (desconectado);
- Consumo de recursos computacionais;
- Estabilidade e escalabilidade dos containers.

5. Análise e Interpretação dos Resultados

A coleta das informações para a avaliação da solução proposta foi conduzida de forma estruturada, combinando medições automatizadas e observações qualitativas. Os dados quantitativos foram obtidos por meio do *Prometheus*, responsável pela coleta contínua de métricas de desempenho, consumo de recursos e disponibilidade dos componentes da arquitetura. As informações coletadas foram armazenadas em série temporal, possibilitando análises históricas e comparativas.

O *Grafana* foi utilizado para a visualização dessas métricas, permitindo a criação de *dashboards* personalizados que facilitaram a interpretação dos resultados e a identificação de padrões de comportamento. Essa abordagem possibilitou acompanhar, em tempo real, indicadores como latência média, desvio padrão, uso de CPU e memória RAM, tráfego de rede Tx/Rx (transmissão em relação com Recepção de dados), tempo de recuperação após falhas e escalabilidade do *cluster* (conjunto) conforme o número de *pods* (Menor Unidade de Execução no *Kubernetes*) [Canonical Ltd. 2025].

As medições foram realizadas em diferentes cenários operacionais, contemplando desde o funcionamento isolado da RB até a operação conjunta com o nó de borda (*Microk8s*). Esses testes possibilitaram comparar diretamente as capacidades de processamento, tempo de resposta e resiliência da infraestrutura nos dois contextos.

Além dos dados coletados automaticamente, foram conduzidas análises qualitativas para identificar gargalos de desempenho, falhas recorrentes e oportunidades de otimização. A

combinação de métricas objetivas e observações subjetivas forneceu uma visão abrangente da eficiência da solução, servindo de base para recomendações e melhorias futuras.

1.4 Organização do Documento

Este trabalho está estruturado em cinco capítulos, além dos elementos pré-textuais e pós-textuais. A seguir, descreve-se a organização do conteúdo:

No **Capítulo 1**, é apresentada a introdução do tema, contendo a contextualização, a justificativa, os objetivos e a relevância da proposta de automação de redes locais com uso de computação em borda.

O **Capítulo 2** aborda os fundamentos teóricos necessários para a compreensão da proposta, incluindo os conceitos de computação em borda, orquestração com *MicroK8s*, uso de contêineres, e a arquitetura em camadas *thing-edge-cloud* (dispositivos-borda-nuvem).

No **Capítulo 3**, é descrita a arquitetura da solução proposta, detalhando a infraestrutura utilizada, os serviços em contêineres, a comunicação entre os componentes e os scripts de automação empregados na integração com o roteador *MikroTik*.

O **Capítulo 4** apresenta a avaliação experimental da solução, incluindo os testes de latência, consumo de recursos, tolerância a falhas e escalabilidade. Além disso, são comparados os resultados obtidos com e sem o nó *Edge*, de forma qualitativa e quantitativa.

Por fim **Capítulo 5**, são discutidas as considerações finais sobre a eficiência da proposta, os benefícios observados, bem como as limitações encontradas. Também são apresentadas sugestões para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Desde sua concepção, a internet foi estruturada com base em uma arquitetura centralizada, na qual dados e aplicações eram processados em grandes centros de dados (*data centers*) e acessados remotamente pelos usuários. Esse modelo, conhecido como arquitetura cliente-servidor, mostrou-se eficiente em contextos de baixa demanda. No entanto, com o crescimento exponencial de dispositivos conectados e a geração massiva de dados, surgiram limitações significativas, especialmente em relação à latência, largura de banda e resiliência [Satyanarayanan 2017].

Para mitigar essas limitações, emergiu o paradigma da computação na borda, que propõe o processamento e a análise de dados mais próximos da fonte de geração. Essa abordagem reduz a latência, diminui o tráfego de dados na rede e aumenta a resiliência do sistema, sendo particularmente relevante para aplicações que exigem respostas em tempo real, como veículos autônomos, dispositivos IoT e sistemas industriais [Satyanarayanan 2017].

A evolução da internet levou ao desenvolvimento de uma arquitetura em três camadas, que integra dispositivos, borda e nuvem. Essa estrutura permite uma distribuição eficiente de tarefas, onde cada camada desempenha funções específicas, melhorando a escalabilidade, a resiliência e a eficiência energética dos sistemas [Gill *et al.* 2024].

A camada de dispositivos inclui sensores, atuadores e dispositivos IoT que coletam e transmitem dados, sendo responsáveis pela interação direta com o ambiente físico. A camada de borda realiza o processamento inicial dos dados coletados, permitindo respostas rápidas e reduzindo a necessidade de enviar grandes volumes de dados para a nuvem. Dispositivos como *gateways* (Dispositivos ou Serviços que Interligam Diferentes Redes) e servidores locais são comuns nesta camada [Gill *et al.* 2024]. Por fim, a camada de nuvem é responsável pelo armazenamento de longo prazo e pelo processamento intensivo de dados, fornecendo recursos computacionais escaláveis e suporte para análises avançadas [Gill *et al.* 2024].

A integração dessas três camadas permite a criação de sistemas mais eficientes, resilientes e adaptáveis. A distribuição de tarefas entre dispositivos, borda e nuvem resulta em melhor utilização dos recursos computacionais, maior economia de largura de banda e mais agilidade na resposta a eventos em tempo real [Gill *et al.* 2024]. Além disso, a arquitetura tricamada

contribui significativamente para o aprimoramento da segurança e privacidade. Ao processar dados localmente na borda, reduz-se o risco de exposição de informações sensíveis na nuvem, favorecendo conformidade com legislações como a *LGPD*[1] e com o *GDPR*[2] [Gill *et al.* 2024].

No contexto da automação de redes locais, a arquitetura tricamada oferece uma base sólida para o desenvolvimento de soluções distribuídas com alta capacidade de resposta. A camada de borda, em particular, desempenha papel central ao permitir a execução de serviços de rede, monitoramento de tráfego e ações automatizadas próximas ao ambiente operacional. Ferramentas como o *microk8s*, uma distribuição enxuta do *kubernetes*, viabilizam a orquestração de contêineres diretamente em dispositivos de borda, como o *Raspberry Pi*, permitindo que aplicações sejam implantadas, gerenciadas e escaladas localmente com eficiência [Canonical Ltd. 2025]. Quando integrada a roteadores programáveis, como o MikroTik RouterBOARD Hex RB750Gr3 [MikroTik 2016], essa infraestrutura se torna capaz de suportar a automação dinâmica de redes, roteamento inteligente e adaptações em tempo real às mudanças no tráfego [Gill *et al.* 2024].

A compreensão da evolução arquitetural da internet, desde seus modelos centralizados até o paradigma atual baseado em borda e nuvem, é fundamental para o desenvolvimento de soluções inovadoras em redes inteligentes. A arquitetura tricamada – formada pelas camadas de dispositivos, borda e nuvem – representa um modelo maduro e eficaz para suportar as demandas crescentes de conectividade, automação e análise de dados em tempo real [Gill *et al.* 2024]. Com base nesse arcabouço teórico, torna-se possível explorar estratégias práticas de automação de redes locais com apoio da computação na borda e ferramentas como o *microk8s*, assegurando flexibilidade, desempenho e resiliência em ambientes operacionais diversos.

2.1 Arquitetura Tricamada

A arquitetura de computação de borda é estruturada em três camadas hierárquicas interdependentes: a camada de dispositivos (*Thing Layer*), a camada de borda (*Edge Layer*) e a camada de nuvem (*Cloud Layer*). Essa organização visa otimizar o fluxo de dados e as decisões computacionais, permitindo maior agilidade na resposta do sistema, redução da latência e uso mais eficiente dos recursos disponíveis [Shi *et al.* 2016] [Hong e Wang 2018].

A camada de dispositivos compreende os dispositivos finais conectados ao ambiente físico. Nessa camada estão sensores, atuadores, câmeras, microcontroladores e dispositivos *IoT* que geram ou consomem dados. Segundo Kong *et al.* 2022, além de coletar informações, alguns dispositivos nesta camada são capazes de realizar pré-processamento leve, como compressão, agregação ou detecção de eventos anômalos. A proximidade com a fonte dos dados é essen-

[1] Lei Geral de Proteção de Dados, Lei nº 13.709/2018, que regula o tratamento de dados pessoais no Brasil

[2] *General Data Protection Regulation* (Regulamento Geral de Proteção de Dados), Regulamento (UE) 2016/679, que estabelece normas para proteção de dados pessoais na União Europeia

cial para aplicações sensíveis ao tempo, como segurança pública, saúde digital e sistemas embarcados industriais.

A camada de borda representa o ponto intermediário entre a origem dos dados e os serviços centralizados. Ela é formada por servidores locais, *gateways* inteligentes e até mesmo *microdata centers* (Centros de Processamento de Dados em Escala Reduzida) posicionados geograficamente próximos aos dispositivos. Essa camada é capaz de realizar inferência com modelos de aprendizado de máquina, aplicar políticas de segurança, tomar decisões autônomas e reduzir a carga da nuvem centralizada [Liu *et al.* 2021]. Como apontado por Andriulo *et al.* 2024, contribuindo para maior robustez do sistema, pois permite operação local mesmo em ambientes de conectividade instável ou limitada.

Já a camada de nuvem mantém o papel tradicional de fornecer alto poder de processamento, armazenamento massivo e serviços centralizados, como bancos de dados em larga escala e modelos de IA complexos *Artificial Intelligence* (Inteligência Artificial). No entanto, sua importância está cada vez mais centrada na consolidação de dados históricos, treinamento de algoritmos e gerenciamento global de aplicações distribuídas. Segundo Shi *et al.* 2016, a computação em nuvem continua sendo essencial para a escalabilidade e integração entre ambientes dispersos, funcionando em sinergia com a borda e os dispositivos.

A principal vantagem da arquitetura tricamada está na sua flexibilidade para distribuir funções computacionais de acordo com as exigências de desempenho, segurança e contexto de rede. Essa separação em camadas melhora a resiliência, a autonomia de sistemas locais e a qualidade da experiência (*QoE*) *Quality of Experience* (Qualidade de Experiência) para o usuário final [Hong e Wang 2018] [Liu *et al.* 2021]. Além disso, facilita a integração com soluções emergentes como redes 5G, cidades inteligentes, fábricas autônomas e serviços baseados em inteligência artificial.

2.1.1 Computação de Borda

É uma abordagem arquitetural que desloca o processamento e a análise de dados da nuvem centralizada para os nós localizados mais próximos da origem dos dados, como sensores, atuadores e *gateways*. Isso permite que decisões sejam tomadas mais rapidamente, o que é crucial para aplicações que demandam resposta em tempo real, como sistemas industriais, ambientes médicos, redes inteligentes e automação local. Shi *et al.* 2016 apontam que a *Edge* surge como uma evolução natural da computação em nuvem, oferecendo uma solução descentralizada que responde melhor às necessidades de latência ultrabaixa e eficiência energética.

Esta camada oferece vantagens significativas dentro dessa arquitetura. Primeiro, permite a redução da latência, uma vez que os dados não precisam viajar até a nuvem para serem processados. Segundo, promove a continuidade operacional em ambientes com conectividade instável ou inexistente, sendo ideal para redes locais offline (desconectado). Terceiro, proporciona maior segurança e privacidade, pois o tratamento dos dados sensíveis ocorre de

forma local, reduzindo a exposição a riscos cibernéticos. Quarto, possibilita a escalabilidade horizontal, já que múltiplos nós *edge* podem ser adicionados de forma modular e econômica. Como discutido por Luo *et al.* 2021, a borda é fundamental para atender a requisitos de *QoS Quality of Service* (Qualidade de Serviço) e *QoE* em cenários distribuídos, especialmente aqueles envolvendo Internet das Coisas e serviços críticos.

Orquestração com *microk8s* em Ambientes com Nó

A orquestração de *containers* é um conjunto de práticas e tecnologias voltadas à automação da implantação, gerenciamento, escalabilidade e monitoramento de aplicações que rodam em contêineres. Em contextos de computação de borda, onde o ambiente é distribuído e a conectividade pode ser limitada, a orquestração torna-se fundamental para garantir a resiliência operacional, a alta disponibilidade dos serviços e a redução da latência em aplicações sensíveis a tempo real [Shanmugam 2023].

O *kubernetes* é a principal plataforma de orquestração de *containers* da atualidade, reconhecida por sua robustez e escalabilidade. Contudo, em ambientes restritos, como dispositivos embarcados ou placas como o Raspberry Pi

O *microk8s* permite implantar, escalar, atualizar e monitorar serviços containerizados diretamente em nós. Ele foi projetado para ser executado com requisitos mínimos de memória e CPU, mantendo compatibilidade com ferramentas nativas do ecossistema *kubernetes*, como *kubectl* (Ferramenta de Linha de Comando para Controle do Kubernetes), *helm* (Gerenciador de Pacotes para Aplicações Kubernetes) e diversos *add-ons* (Extensões ou Complementos que Adicionam Funcionalidades ao Kubernetes). Isso possibilita que os nós implementem soluções de automação local com alta eficiência e sem necessidade de conectividade constante com a nuvem [Shanmugam 2023].

Um dos principais diferenciais do *microk8s* é seu suporte nativo a *add-ons* essenciais, como:

- *prometheus* e *grafana*, para monitoramento e visualização de métricas em tempo real;
- *metallb* (Balanceador de Carga para Ambientes Kubernetes em Redes Bare-Metal), que permite o uso de IPs externos e balanceamento de carga em redes locais;
- *knative* (Plataforma para Construção e Gerenciamento de Aplicações *Serverless* em *Kubernetes*) e *istio* (Plataforma de Malha de Serviços — *Service Mesh* — para Controle e Segurança de Comunicação entre Serviços em *Kubernetes*), para execução de funções *serverless* (Modelo de Computação em Nuvem sem Gerenciamento Direto de Servidores) e integração de *service mesh* (Camada de Infraestrutura para Gerenciamento de Comunicação entre Serviços);
- *openfaas* (Plataforma para Funções como Serviço — *Functions as a Service* — em Ambientes de *Contêineres*) e *kubeflow* (Plataforma para Implantação e Gerenciamento de

Fluxos de Trabalho de Aprendizado de Máquina em *Kubernetes*), voltados a aplicações de inteligência artificial e ciência de dados distribuída.

Além disso, a simplicidade da instalação e o suporte multiplataforma garantem rápida adoção mesmo em ambientes educacionais ou laboratórios de baixo orçamento. Liu *et al.* 2019 destacam o *microk8s* como uma ferramenta fundamental para projetos borda-nuvem, viabilizando desde a prototipagem até a implantação de aplicações críticas, inclusive em ambientes com infraestrutura limitada.

Complementando isso, Famá, Santos e Perkusich 2018 enfatizam que a capacidade de orquestração automatizada entre dispositivos *edge e IoT* promove a descentralização computacional e contribui diretamente para a otimização da qualidade de serviço (QoS) em redes locais.

Portanto, o uso do *microk8s* neste trabalho justifica-se pela sua leveza, simplicidade de operação em maquina, compatibilidade com o ecossistema *kubernetes* e capacidade de orquestrar serviços críticos em infraestruturas de rede locais e descentralizadas, alinhando-se perfeitamente aos objetivos da computação de borda aplicada à automação.

2.1.2 Benefícios e Desafios da Computação de Borda

O Sistema de borda tem se destacado como uma solução eficaz para atender às demandas de aplicações que requerem processamento em tempo real, alta disponibilidade e maior privacidade dos dados. Entre os principais benefícios, destaca-se a redução significativa da latência, uma vez que o processamento ocorre próximo à fonte de dados, eliminando a necessidade de transmissão para servidores remotos. Essa proximidade também contribui para a economia de largura de banda e redução de custos operacionais, especialmente em ambientes com grande volume de dados, como em aplicações industriais e de Internet das Coisas (IoT) [Shi *et al.* 2016].

Além disso, ao processar dados localmente, a computação de borda aprimora a privacidade e a segurança das informações, minimizando os riscos associados à transmissão de dados sensíveis pela rede. Essa característica é particularmente relevante em setores como saúde e finanças, onde a confidencialidade dos dados é crucial [Andriulo *et al.* 2024]. A capacidade de operar de forma autônoma, mesmo em condições de conectividade limitada ou intermitente, também confere maior robustez operacional aos sistemas baseados nesta tecnologia [Kong *et al.* 2022].

No entanto, a adoção da computação de borda também impõe desafios significativos. A heterogeneidade dos dispositivos e a falta de padronização de protocolos dificultam a interoperabilidade e a integração eficiente entre diferentes componentes da rede. Além disso, a gestão de recursos computacionais limitados em dispositivos de borda exige estratégias avançadas de alocação e balanceamento de carga para garantir desempenho adequado [Shi *et al.* 2016].

Por fim, a escalabilidade dos sistemas de computação de borda depende da capacidade de orquestrar e gerenciar eficientemente um grande número de dispositivos distribuídos. Isso implica na necessidade de ferramentas e *frameworks* (Estruturas de Desenvolvimento de Software) que facilitem a implantação, monitoramento e atualização de aplicações em ambientes heterogêneos e dinâmicos [Liu *et al.* 2021].

Dastjerdi *et al.* 2016 reforçam que o avanço da computação de borda depende da superação desses obstáculos, especialmente em sistemas críticos que requerem disponibilidade contínua e escalabilidade confiável.

2.2 Automação de Redes Locais com computação de borda

Com o crescimento em sistemas em borda e a necessidade de soluções cabíveis. O nó traz essa autonomia e é especialmente relevante em ambientes críticos, onde a latência precisa ser mínima e a resiliência do sistema é essencial [Cavadas *et al.* 2022]. Em cenários como unidades hospitalares, plantas industriais ou zonas remotas com conectividade intermitente, a borda torna possível manter serviços operando localmente mesmo diante de falhas na conexão com a nuvem [Satyanarayanan 2017].

Essa abordagem viabiliza, por exemplo, o monitoramento de tráfego de rede em tempo real, a execução de regras automatizadas de resposta a falhas e o controle dinâmico de topologias e políticas locais de roteamento. Segundo Abbas *et al.* 2018, a capacidade de processar e agir sobre dados no próprio ambiente onde eles são gerados é uma das maiores vantagens da borda, sendo estratégica em infraestruturas sensíveis à latência e à variabilidade da banda.

Outro benefício evidente está relacionado à segurança e privacidade. Ao manter os dados dentro da rede local durante as operações mais sensíveis, reduz-se o risco de exposição a ataques externos e vazamento de informações, o que é fundamental em setores como saúde, defesa e energia [Chiang e Zhang 2016]. Além disso, esse modelo ajuda a preservar a largura de banda para transmissões realmente críticas, enquanto delega tarefas operacionais rotineiras para os nós locais.

Apesar dos benefícios, há desafios consideráveis. A implementação de automação local em arquiteturas distribuídas demanda padronização entre dispositivos e interfaces, algo ainda em evolução no ecossistema do nó. Como indica Dastjerdi *et al.* 2016, a interoperabilidade entre dispositivos de diferentes fabricantes e a ausência de protocolos amplamente aceitos tornam o gerenciamento e a integração complexos em redes heterogêneas.

A escalabilidade e manutenção de ambientes distribuídos também exigem orquestração eficiente. Em resposta a isso, surgem plataformas como *microk8s*, que permitem o gerenciamento de contêineres na borda de forma leve e modular. Contudo, como ressaltam Liu *et al.* 2021, o gerenciamento de múltiplos *clusters* distribuídos, sua segurança, atualizações e monitoramento contínuo exigem novas abordagens em relação aos modelos tradicionais

centralizados de rede.

Em síntese, a automação de redes locais com a computação de borda representa uma alternativa promissora para tornar redes mais resilientes, autônomas e eficientes. Contudo, para que sua adoção seja bem-sucedida em larga escala, é necessário superar barreiras técnicas relacionadas à interoperabilidade, segurança distribuída e gestão de recursos locais — desafios que vêm sendo abordados em estudos recentes e soluções industriais emergentes.

2.3 MikroTik no ecossistema de borda

No contexto da arquitetura *edge*, o roteador *MikroTik RB750Gr3*, popularmente conhecido como hEX ou RB, tem se destacado como uma solução técnica viável e de excelente custo-benefício. Trata-se de um roteador gerenciável equipado com processador dual-core de 880 MHz, 256 MB de RAM, cinco portas *Gigabit Ethernet* e baixo consumo energético — cerca de 3 *watts* —, o que o torna ideal para aplicações de borda em ambientes com restrições energéticas ou infraestrutura limitada [MikroTik 2016].

Um dos diferenciais da RB está em sua compatibilidade com o sistema operacional RouterOS, desenvolvido pela própria *MikroTik* (Empresa de Equipamentos e Software para Redes). Este sistema permite implementar funcionalidades avançadas de rede, como roteamento dinâmico, gerenciamento de largura de banda, *firewalls*, VPNs *Virtual Private Network* (Rede Privada Virtual), e, sobretudo, automação de tarefas via *scripts* (Conjunto de Instruções Automatizadas para Execução de Tarefas - *RouterOS Script*) internos e chamadas por APIs RESTful *Application Programming Interface* (Interface de Programação de Aplicações). Essa característica possibilita sua integração com serviços externos baseados em orquestração de *containers*, como clusters *microk8s*, tornando-o especialmente útil para arquiteturas *edge* (Camada de Processamento de Dados na Borda da Rede) que exigem controle local de tráfego e serviços [MikroTik 2025].

Além disso, o roteador suporta interfaces de expansão como USB *Universal Serial Bus* (Barramento Serial Universal) e cartão microSD (Cartão de Memória Flash em Formato Compacto), possibilitando o uso de ferramentas adicionais como o "The Dude"(Ferramenta de Monitoramento de Redes da *MikroTik*), voltadas ao monitoramento de redes locais de forma leve e funcional [MikroTik 2016]. Essa modularidade torna o dispositivo altamente adaptável a diferentes casos de uso, desde residências inteligentes até redes industriais e experimentações educacionais.

A relevância do MikroTik RB750Gr3 em contextos de *edge* também é discutida na literatura acadêmica. Famá, Santos e Perkusich 2018 apontam que a integração entre roteadores programáveis e plataformas de containerização distribuída pode resultar em ganhos significativos de desempenho e autonomia de rede. A presença de APIs de gerenciamento torna viável o provisionamento automático de regras de rede, configuração de VLANs *Virtual Local Area Network* (Rede Local Virtual) e implementação de políticas de segurança locais —

elementos fundamentais para redes autônomas, adaptativas e resilientes.

Portanto, a RB hEX não é apenas uma opção de baixo custo, mas também uma ferramenta estratégica para arquiteturas de computação de borda. Sua leveza, robustez, automação por *script* e compatibilidade com soluções modernas de orquestração o qualificam como um nó inteligente dentro de redes *edge* modernas.

Como discutido por Famá, Santos e Perkusich 2018, essa capacidade de automação e integração com *containers* torna a RB uma ferramenta eficaz para implementar redes inteligentes e adaptativas baseadas em computação de borda.

Capítulo 3

Arquitetura

A crescente complexidade e demanda por redes locais mais dinâmicas, inteligentes e autônomas exige soluções que não apenas monitorem o tráfego e os dispositivos conectados, mas que também atuem de maneira proativa diante de falhas ou mudanças de contexto. Nesse cenário, a computação de borda se apresenta como uma alternativa promissora ao tradicional modelo centralizado de redes gerenciadas pela nuvem. Este capítulo detalha a proposta arquitetural desenvolvida, apresentando os fundamentos, componentes, tecnologias envolvidas, topologia da rede e justificativas técnicas.

A arquitetura lógica projetada neste trabalho visa distribuir a inteligência computacional para a borda da rede, reduzindo a latência de decisões e aumentando a autonomia dos sistemas locais. O conceito é inspirado na ideia de que tarefas críticas – como detecção de falhas, reconfiguração de serviços e controle de tráfego – não devem depender exclusivamente de conexão com a nuvem.

A solução é composta por quatro elementos principais:

3.1 Nó de borda:

Neste projeto, o nó de borda é materializado por um Raspberry Pi 5, um dispositivo de alto desempenho, capaz de operar de forma contínua e confiável em ambientes de rede. Ele desempenha o papel central — o "cérebro" da solução — processando informações, coordenando fluxos de dados e tomando decisões em tempo real.

Este equipamento hospeda o *cluster MicroK8s*, que funciona como uma versão otimizada e leve do *Kubernetes*, especialmente adequada para cenários de *Edge*. Dentro deste ambiente, são orquestrados e executados diversos contêineres que compõem a arquitetura da solução, cada um com funções específicas voltadas para a automação da rede, coleta de métricas, análise de desempenho e monitoramento contínuo dos recursos e serviços.

Ao operar próximo da origem dos dados, o nó de borda reduz significativamente a latência, melhora a resiliência do sistema e garante maior eficiência no tratamento das informações, mesmo em condições adversas ou em situações de conectividade limitada com a nuvem

[Simon, Wiesmeyer e Varga 2021].

Com quatro núcleos ARM Cortex-A72 *Advanced RISC Machine* (Máquina Avançada com Conjunto de Instruções Reduzidas e Otimizadas) e até 8GB de RAM, o Raspberry Pi 5 oferece potência computacional suficiente para executar serviços em contêineres, inclusive pequenos clusters de *kubernetes*. É ideal para ambientes de borda por ser compacto, eficiente energeticamente e suportar uma ampla gama de sistemas operacionais.

Além disso, sua comunidade ativa e documentação extensa tornam sua adoção prática, especialmente em ambientes educacionais e de pesquisa. A presença de portas USB 3.0, GPIOs *General Purpose Input/Output* (Entrada/Saída de Uso Geral) e *Ethernet Gigabit* (Padrão de Rede Ethernet com Taxa de Transmissão de 1 Gigabit por Segundo) ampliam suas possibilidades de uso em soluções de automação [MikroTik 2016].

3.2 Dispositivo de Roteamento:

A RB, atua como o núcleo de gerenciamento do tráfego local na arquitetura proposta. Ele é responsável por executar o controle de acesso aos recursos da rede, aplicar políticas de segurança, implementar regras de roteamento e realizar o tratamento de pacotes de forma eficiente.

Sua integração com o nó de borda é realizada por meio de *scripts* automatizados e chamadas à *API REST* nativa do *RouterOS*, permitindo que comandos sejam enviados e executados dinamicamente, sem necessidade de intervenção manual no equipamento. Essa abordagem proporciona alta flexibilidade operacional, permitindo desde ajustes finos de QoS *Quality of Service* (Qualidade de Serviço) até alterações na topologia lógica de forma remota e imediata.

Além disso, o roteador desempenha papel fundamental na coleta de métricas utilizadas pelo sistema de monitoramento, expondo dados de desempenho e status da rede que são consumidos pelos contêineres do nó de borda. Essa integração bidirecional garante controle refinado e visibilidade em tempo real, elementos essenciais para ambientes de *Edge* e automação de redes locais [MikroTik 2016].

Este modelo é amplamente utilizado em ambientes corporativos e residenciais devido à sua robustez e flexibilidade. Ele permite automação via *RouterOS Script* e exposição de serviços por *API REST*, diferenciais importantes para projetos de redes autogerenciáveis.

Sua interface amigável - *Winbox* (Aplicativo Gráfico para Configuração de Roteadores *MikroTik*) - facilita a configuração manual, mas o verdadeiro potencial para este projeto está na sua integração programática com o cluster *microk8s*.

3.3 Clientes de rede:

São representados por dispositivos variados, como computadores, sensores, ou câmeras IP, simulando um ambiente realista de rede local com múltiplos pontos de acesso.

O diferencial da proposta está na descentralização. Todas as decisões de automação, regras de firewall, ajustes de *NAT Network Address Translation* (Tradutor de Endereços de Rede) e monitoramento são executadas localmente, tornando o sistema mais resiliente a falhas externas.

Também é possível, futuramente, integrar sensores *IoT* ao sistema, explorando o potencial do *edge* para aplicações de cidade inteligente, segurança, ou automação predial.

3.4 Componentes lógicos

3.4.1 Ubuntu Server 24.04 LTS

O *Ubuntu Server 24.04 LTS (Long Term Support)* é uma distribuição Linux amplamente consolidada no mercado corporativo e acadêmico, reconhecida por sua estabilidade, segurança e suporte prolongado, com atualizações garantidas por cinco anos. Essa característica é fundamental para ambientes de produção e projetos de pesquisa que demandam alta disponibilidade e confiabilidade operacional.

A escolha dessa versão está diretamente associada à sua leveza e otimização para servidores *headless* (Modo de Operação sem Interface Gráfica) — ou seja, sistemas que operam sem interface gráfica, maximizando o aproveitamento de recursos de hardware e reduzindo consumo de memória e processamento. Isso é especialmente vantajoso no contexto do projeto, que utiliza arquiteturas *ARM*, como no caso do *Raspberry Pi*, e requer desempenho eficiente para hospedar o *cluster MicroK8s*.

Um diferencial decisivo para a adoção do *Ubuntu Server* é o suporte nativo ao *Snap*, sistema de empacotamento e distribuição de aplicações desenvolvido pela *Canonical* (Empresa Desenvolvedora do Sistema Operacional Ubuntu), que possibilita a instalação, atualização e isolamento de serviços de forma simplificada. No caso específico do *MicroK8s*, o Snap permite a implantação rápida e segura do *Kubernetes Runtime* (Ambiente de Execução de Contêineres no Kubernetes), assegurando que todos os componentes essenciais sejam entregues em um ambiente isolado e de fácil manutenção.

Além disso, o este sistema incorpora recursos atualizados de gerenciamento de rede, segurança (*AppArmor* (Módulo de Segurança do Kernel Linux para Controle de Acesso Obrigatório) e *UFW Uncomplicated Firewall* (*Firewall* Simplificado para Sistemas Linux)), suporte a protocolos modernos e compatibilidade com bibliotecas otimizadas para ARM64. Essa base sólida garante que o sistema possa operar de forma contínua "uptime prolongado", com menor risco de falhas e interrupções — fator crítico em arquiteturas de *Edge*, onde a proximidade com a origem dos dados exige respostas rápidas e consistentes [Canonical Ltd.

2024].

3.4.2 *Microk8s*

É uma distribuição leve, modular e de fácil implantação do *Kubernetes*, desenvolvida pela *Canonical* com foco em ambientes de Computação de Borda, IoT e desenvolvimento local. Apesar de sua arquitetura otimizada para dispositivos com recursos limitados, como o Rasp, o nó mantém a compatibilidade com o ecossistema *Kubernetes* completo, oferecendo suporte aos principais recursos nativos, como *DNS Domain Name System* (Sistema de Nomes de Domínio) interno, *Ingress Controller* (Componente do Kubernetes que Gerencia o Acesso Externo aos Serviços), *Storage* persistente (Armazenamento de Dados), *Prometheus* para monitoramento e *Dashboard* para gerenciamento visual do *cluster*.

O design do nó permite que todos esses serviços sejam executados em um único *cluster*, reduzindo significativamente a complexidade de implantação e eliminando a necessidade de múltiplos nós mestres em ambientes menores. Essa característica é especialmente vantajosa para o projeto, pois garante eficiência operacional, baixa latência e simplificação na manutenção, alinhando-se às restrições e demandas de um cenário de borda.

Entre os serviços integrados, destaca-se o *Grafana*, ferramenta de visualização e análise de dados que, quando combinada ao *Prometheus*, permite a criação de painéis interativos *dashboards* para acompanhamento em tempo real de métricas como uso de CPU *Central Processing Unit* (Unidade Central de Processamento), memória, tráfego de rede e disponibilidade de serviços [Canonical Ltd. 2025].

Essa integração *Prometheus–Grafana no MicroK8s* torna possível não apenas visualizar indicadores operacionais, mas também configurar alertas proativos, o que contribui para um gerenciamento preventivo e para a rápida resposta a incidentes. Em conjunto, eles formam um ecossistema robusto e de baixo custo para monitoramento e orquestração de redes locais automatizadas.

3.4.3 Programa de Aplicação e *Scrits*:

A *API REST do MikroTik* é um mecanismo de comunicação baseado em requisições HTTP *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto) que possibilita o envio e recebimento de comandos de forma remota e programática para dispositivos executando o *RouterOS*. Essa interface expõe funcionalidades avançadas de gerenciamento, permitindo a integração direta com serviços externos, como contêineres executados no *cluster MicroK8s*, e possibilitando que tarefas de rede sejam orquestradas automaticamente por meio de aplicações e agentes externos.

No contexto deste projeto, a *API REST* é utilizada como canal de controle entre o nó de borda e a RB, viabilizando que *contêineres* especializados — como os de monitoramento e automação — executem ações administrativas sem a necessidade de intervenção manual.

Isso inclui operações como: Configuração dinâmica de regras de *firewall*; Criação e remoção de filas de *QoS*; Alteração de rotas de rede; Consulta e coleta de métricas de desempenho em tempo real.

Complementando essa camada de controle remoto, os *scripts* ampliam as possibilidades de automação, permitindo a execução de instruções diretamente no sistema operacional do roteador. Esses roteiros programados podem ser acionados de forma reativa (em resposta a eventos, como um alerta de tráfego anômalo gerado pelo *Prometheus*) ou programada (em intervalos regulares ou horários específicos), garantindo que ajustes na rede sejam aplicados de forma rápida e consistente.

Essa combinação entre *API e scripts* locais cria um ecossistema de automação bidirecional: de um lado, o nó de borda envia comandos e monitora resultados; de outro, o roteador reage a eventos e executa ações previamente definidas, aumentando a resiliência, a segurança e a capacidade de resposta da rede. Essa abordagem é essencial em arquiteturas de borda, onde a latência e a autonomia operacional são fatores críticos para a continuidade dos serviços [MikroTik 2025].

3.5 Topologia da Rede

A topologia implementada é do tipo estrela, tendo a RB como ponto central, conectando os dispositivos clientes e o Rasp por meio de conexões *Ethernet*. O roteador distribui os IPs via DHCP *Dynamic Host Configuration Protocol* (Protocolo de Configuração Dinâmica de Endereços), mas o nó de borda recebe um IP fixo *Internet Protocol* (Protocolo de Internet) para facilitar sua integração com os *scripts* automatizados.

Os serviços em contêineres no Rasp são expostos via *Ingress* do *microk8s*, podendo ser acessados internamente ou externamente conforme a política configurada. A comunicação entre o nó e a RB é realizada pela interface *REST* via IP local, permitindo comandos diretos e de baixa latência.

Essa topologia é facilmente escalável: novos clientes podem ser adicionados ao roteador, e o *cluster* pode ser expandido com novos nós, inclusive em nós adicionais. A arquitetura também permite futuras integrações com redes *mesh* (Topologia de Rede em Malha), *gateways, IoT* e soluções híbridas com nuvem.

3.6 Justificativa Técnica das Escolhas:

A definição dos componentes e tecnologias utilizadas nesta arquitetura considerou múltiplos critérios: custo, robustez, disponibilidade de recursos, documentação, escalabilidade e compatibilidade com automação de redes.

A escolha do Rasp 5 reflete uma busca por uma solução de baixo custo, acessível para ambientes educacionais e de pesquisa, mas ainda assim capaz de executar um *cluster kubernetes*

funcionais.

O *microk8s* foi selecionado como orquestrador pela sua leveza, simplicidade de instalação e suporte completo aos recursos do *kubernetes*. Seu design simplificado facilita a operação em dispositivos com poucos recursos, sem comprometer funcionalidades críticas.

O roteador hEX foi escolhido por seu suporte nativo a automação, robustez e flexibilidade. Alternativas como roteadores TP-Link *TP-Link Technologies Co., Ltd.* (Fabricante de Equipamentos de Rede) ou *Cisco Small and Medium Business* (Linha de Equipamentos da Cisco para Pequenas e Médias Empresas) foram descartadas devido à limitação em automações programáticas ou custo elevado [MikroTik 2016] [TP-Link Technologies Co., Ltd. 2025] [Cisco Systems, Inc. 2025].

Por fim, a integração de ferramentas como, *prometheus* e *grafana* permite não apenas uma arquitetura funcional, mas também observável e auditável, aspectos fundamentais em ambientes críticos e em projetos acadêmicos que exigem validação de desempenho.

Capítulo 4

Implementação e Configuração do Ambiente de Teste

A implementação prática da arquitetura proposta permitiu validar, por meio de experimentações reais, a viabilidade técnica e operacional da automação de redes locais baseada em computação de borda com *microk8s*. Este capítulo descreve, em detalhes, todas as etapas de construção do ambiente, instalação e configuração dos componentes físicos e lógicos, bem como a integração e os testes realizados.

4.1 Preparação do nó de borda com *Ubuntu Server*

A configuração do ambiente teve início com a instalação do sistema operacional Ubuntu Server 24.04 LTS em um Raspberry Pi 5. A escolha desta versão estável do Ubuntu está alinhada com os requisitos do *microk8s* e garante compatibilidade com a arquitetura *ARM64 Advanced RISC Machines 64-bit* (Arquitetura de Processador ARM de 64 Bits).

Etapas de instalação [Canonical Ltd. 2025]:

- Download da imagem oficial do Ubuntu Server no site da Canonical;
- Gravação da imagem em cartão microSD com uso da ferramenta Raspberry Pi Imager;
- Criação de um arquivo user-data para login automático via SSH;
- Criação de configuração de rede com IP estático via netplan;
- Primeiro boot e acesso remoto via SSH;
- Atualizações do sistema com:

```
1          sudo apt update && sudo apt upgrade;
```

- Instalação de pacotes essenciais como htop, net-tools, openssh-server, vim, curl, unattended-upgrades.

Com o sistema operacional configurado e seguro, partiu-se para a instalação do *microk8s*, núcleo da solução de automação proposta.

4.2 Instalação e Configuração do *Microk8s*

A instalação do *microk8s* foi realizada com o seguinte comando:

```
1 sudo snap install \textit{microk8s} --classic
```

Após a instalação, o usuário foi adicionado ao grupo *microk8s*, garantindo permissões de execução sem sudo [Canonical Ltd. 2025]:

```
1 sudo usermod -a -G \textit{microk8s} $USER
```

4.2.1 Módulos habilitados:

- DNS *Domain Name System* (Sistema de Nomes de Domínio): essencial para serviços internos;
- Storage: para persistência local;
- Ingress: gerencia rotas externas para serviços expostos;
- *prometheus*: coleta de métricas de sistema e serviços;
- Metrics-server: monitoração de recursos dos pods;
- Dashboard: interface visual de gerenciamento do cluster.

```
1 \textit{microk8s} enable dns \
2               hostpath-storage \
3               ingress \
4               \textit{prometheus} \
5               metrics-server \
6               dashboard \
7               helm3 \
8               registry
```

O comando apresentado executa a habilitação de diversos complementos *(add-ons)* no *MicroK8s*, cada um responsável por fornecer funcionalidades essenciais para a operação e gestão do *cluster*.

O parâmetro *dns* ativa o serviço de resolução de nomes interno do *Kubernetes*, permitindo que os *pods* e serviços se comuniquem utilizando nomes lógicos em vez de endereços *IP* estáticos.

O complemento *hostpath-storage hostPath Storage* (Tipo de Armazenamento no *Kubernetes* que Usa um Caminho do Sistema de Arquivos do Nó Hospedeiro) configura um provedor de armazenamento persistente baseado em diretórios locais do nó. Essa funcionalidade é especialmente útil para ambientes de desenvolvimento ou *edge computing*, onde não há sistemas de armazenamento distribuído.

O módulo ingress habilita o *Ingress Controller*, responsável por gerenciar o tráfego externo que entra no *cluster*, oferecendo recursos como roteamento baseado em nome de host, balanceamento de carga e terminação *mTLS Mutual Transport Layer Security* (Segurança de Camada de Transporte Mútua).

A ativação do *prometheus* integra a plataforma de monitoramento ao *cluster*, permitindo coleta, armazenamento e consulta de métricas operacionais dos serviços e recursos do ambiente.

O complemento *metrics-server* fornece métricas de uso de *CPU* e memória em tempo real para os objetos do *Kubernetes*, viabilizando funções como *Horizontal Pod Autoscaling* (Escalonamento Automático Horizontal de Pods no Kubernetes) e monitoramento de carga.

O módulo dashboard habilita a interface gráfica oficial do *Kubernetes*, facilitando a visualização e o gerenciamento de recursos, serviços e *pods* de forma centralizada.

A funcionalidade *Helm 3* (Versão 3 do Gerenciador de Pacotes para Aplicações Kubernetes) instala o gerenciador de pacotes *Helm* (versão 3), que simplifica a implantação de aplicações complexas por meio de *charts* (Pacotes de Configuração e Recursos para Implantação no Kubernetes via Helm), pacotes pré-configurados de recursos do *Kubernetes*.

Por fim, o complemento *Registry* (Repositório para Armazenamento e Distribuição de Imagens de Contêineres) ativa um repositório local de imagens *Docker*, permitindo armazenar e servir imagens diretamente no ambiente do *MicroK8s*, reduzindo a necessidade de dependência de registros externos e melhorando a performance na implantação de serviços [Canonical Ltd. 2025].

4.2.2 Teste de implantação:

Como validação do funcionamento do *cluster microk8s*, foi implantado um serviço de teste baseado na imagem *nginx*, utilizando um objeto *deployment* (Recurso do Kubernetes para Gerenciar a Implantação e Atualização de Aplicações em Contêineres) e um *Service NodePort* (Tipo de Serviço do *Kubernetes* que Expõe uma Aplicação em uma Porta Específica de Cada Nó do *Cluster*). Após a implantação, foi possível acessar a página padrão do *NGINX - Engine X* (Servidor *Web e Proxy* Reverso de Alto Desempenho) via navegador, utilizando o IP do Rasp e a porta exposta, confirmando o correto funcionamento da infraestrutura básica do *cluster*.

```
1 #########nginx-deploy.yaml############
2       apiVersion: apps/v1
3       kind: \textit{deployment}
```

```
4       metadata:
5         name: nginx-deploy
6       spec:
7         replicas: 1
8         selector:
9           matchLabels:
10            app: nginx
11        template:
12          metadata:
13            labels:
14              app: nginx
15          spec:
16            \textit{containers}:
17              - name: nginx
18                image: nginx
19                ports:
20                  - containerPort: 80
21      ---
22      apiVersion: v1
23      kind: Service
24      metadata:
25        name: nginx-service
26      spec:
27        type: NodePort
28        selector:
29          app: nginx
30        ports:
31          - port: 80
32            targetPort: 80
33            nodePort: 30080
34
35 \textit{microk8s} \textit{kubectl} apply -f nginx-deploy.yaml
```

4.3 Integração com o *MikroTik* via *API e Scripts*

Para alcançar a automação da rede, o próximo passo foi integrar o *microk8s* a RB hEX, permitindo que serviços em contêineres executassem comandos diretamente no roteador.

4.3.1 Configurações no MikroTik:

Habilitação da API [MikroTik 2025]:

```
1 /ip service enable api
2 /ip service set api port=8728 address=192.168.0.0/24
```

Houve a necessidade da criação de usuário com permissões limitadas para automação. Habilitação do modo *Safe Mode* (Modo de Segurança do Sistema Operacional) para testes iniciais. E ativação de SNMP *Simple Network Management Protocol* (Protocolo Simples de Gerenciamento de Rede) para coleta de métricas passivas. Também foi utilizado o módulo *Python RouterOS API* (Biblioteca em Python para Interação com o Sistema RouterOS da MikroTik via API) em um contêiner, que se conecta à API para executar comandos [MikroTik 2025]. Exemplo de uso:

```
1 connection = routeros_api.RouterOsApiPool('192.168.0.1', username='
    auto', password='1234')
2 api = connection.get_api()
3 interfaces = api.get_resource('/interface/ethernet')
4 print(interfaces.get())
```

Scripts programáveis foram utilizados para executar ações internas no *MikroTik*, como:

- Alterar regras de *firewall*;
- Reiniciar interfaces específicas;
- Criar regras temporárias para *NAT* ou redirecionamento;
- Registrar eventos no *log* (Registro de Sistema).

Essas automações foram disparadas via chamadas *HTTP* do *cluster microk8s*, de forma autônoma.

4.4 Implantação de Serviços em Contêineres

Para orquestrar a automação e a visualização dos dados da rede, foram implantados os seguintes serviços em contêineres:

4.4.1 Agente de Automação *(Python)*:

O Agente de Automação foi desenvolvido em *Python* com foco na orquestração de tarefas inteligentes aplicadas na RB, utilizando a *API REST*. Esse agente foi empacotado em um *container Docker* por meio de um *Dockerfile* (Arquivo de Configuração que Define as Instruções para Construção de uma Imagem Docker), gerando uma imagem compatível com ambientes *kubernetes*. Ou seja, a imagem foi construída externamente (em ambiente de desenvolvimento) com o *Docker* e exportada como um arquivo ".tar"(formato de arquivo de

compactação), sendo posteriormente importada no *microk8s* via *containerd* (*Runtime* de Contêiner de Alto Desempenho Utilizado pelo *ubernetes e Docker*).

```
import requests
import time
import logging

# Configurao do MikroTik
MIKROTIK_IP = "<192.168.0.1>"
USERNAME = "admin"
PASSWORD = "sua_senha"
API_BASE = f"http://{MIKROTIK_IP}/rest"
AUTH = (USERNAME, PASSWORD)

# Inicializa o log local
logging.basicConfig(filename='automacao.log', level=logging.INFO,
    format='%(asctime)s - %(message)s')

# Funo 1: Bloqueia um IP
def bloquear_ip(ip):
   endpoint = f"{API_BASE}/ip/firewall/filter"
   payload = {
      "chain": "forward",
      "src-address": ip,
      "action": "drop",
      "comment": "Bloqueio automtico via \textit{microk8s}"
   }
   try:
      response = requests.post(endpoint, json=payload, auth=AUTH)
      if response.status_code in [200, 201]:
         logging.info(f"IP {ip} bloqueado com sucesso.")
      else:
         logging.error(f"Erro ao bloquear IP {ip}: {response.
            status_code} - {response.text}")
   except Exception as e:
      logging.error(f"Exceo ao bloquear IP {ip}: {e}")

# Funo 2: Remove regra por comentrio
def remover_regra_por_comentario(comentario):
   endpoint = f"{API_BASE}/ip/firewall/filter"
   try:
      regras = requests.get(endpoint, auth=AUTH).json()
      for regra in regras:
```

```
39          if regra.get("comment") == comentario:
40              id_regra = regra[".id"]
41              delete_endpoint = f"{endpoint}/{id_regra}"
42              r = requests.delete(delete_endpoint, auth=AUTH)
43              if r.status_code == 200:
44                  logging.info(f"Regra '{comentario}' removida com
                        sucesso.")
45    except Exception as e:
46        logging.error(f"Erro ao remover regra: {e}")
47
48 # Funo 3: Monitorar se MikroTik est acessvel
49 def monitorar_conexao():
50    try:
51        response = requests.get(f"{API_BASE}/system/identity", auth=
              AUTH, timeout=5)
52        if response.status_code == 200:
53            logging.info("MikroTik acessvel.")
54            return True
55        else:
56            logging.warning("MikroTik respondeu, mas com erro.")
57            return False
58    except requests.exceptions.RequestException as e:
59        logging.error(f"MikroTik offline: {e}")
60        return False
61
62 # Funo 4: Reiniciar interface de rede
63 def reiniciar_interface(nome_interface):
64    endpoint = f"{API_BASE}/interface"
65    try:
66        interfaces = requests.get(endpoint, auth=AUTH).json()
67        for iface in interfaces:
68            if iface.get("name") == nome_interface:
69                id_iface = iface[".id"]
70                requests.put(f"{endpoint}/{id_iface}", json={"disabled":
                      True}, auth=AUTH)
71                time.sleep(2)
72                requests.put(f"{endpoint}/{id_iface}", json={"disabled":
                      False}, auth=AUTH)
73                logging.info(f"Interface {nome_interface} reiniciada.")
74                return
75        logging.warning(f"Interface {nome_interface} no encontrada.")
76    except Exception as e:
```

```
77       logging.error(f"Erro ao reiniciar interface: {e}")
78
79 # Funo 5: Executar qualquer comando REST
80 def executar_comando(path, metodo="get", dados=None):
81    url = f"{API_BASE}/{path}"
82    try:
83       if metodo == "get":
84          r = requests.get(url, auth=AUTH)
85       elif metodo == "post":
86          r = requests.post(url, json=dados, auth=AUTH)
87       elif metodo == "put":
88          r = requests.put(url, json=dados, auth=AUTH)
89       elif metodo == "delete":
90          r = requests.delete(url, auth=AUTH)
91       else:
92          raise ValueError("Mtodo HTTP invlido.")
93       logging.info(f"{metodo.upper()} {path} - Status: {r.
            status_code}")
94       return r
95    except Exception as e:
96       logging.error(f"Erro REST {metodo.upper()} {path}: {e}")
97
98 # Funo 6: Failover automtico entre duas interfaces
99 def executar_failover(ip_monitorado="8.8.8.8", rota_principal="
     ether1", rota_backup="ether2"):
100    logging.info("Verificando necessidade de failover...")
101    try:
102       response = requests.get(f"{API_BASE}/tool/ping", params={
103          "address": ip_monitorado,
104          "interface": rota_principal,
105          "count": 3
106       }, auth=AUTH)
107
108       if response.status_code == 200:
109          resultado = response.json()
110          if int(resultado[0].get("received", 0)) > 0:
111             logging.info(f"Conectividade OK via {rota_principal}")
112             return
113          else:
114             logging.warning(f"Sem resposta via {rota_principal},
                  ativando failover.")
115       else:
```

```
116         logging.warning("Ping falhou, ativando failover.")
117
118       _desabilitar_rota_por_interface(rota_principal)
119       _habilitar_rota_por_interface(rota_backup)
120
121    except Exception as e:
122       logging.error(f"Erro no failover: {e}")
123
124 def _desabilitar_rota_por_interface(interface):
125    try:
126       rotas = requests.get(f"{API_BASE}/ip/route", auth=AUTH).json()
127       for rota in rotas:
128          if rota.get("\textit{gateway}") == interface:
129             id_rota = rota[".id"]
130             requests.put(f"{API_BASE}/ip/route/{id_rota}", json={"
                   disabled": True}, auth=AUTH)
131             logging.info(f"Rota via {interface} desabilitada.")
132    except Exception as e:
133       logging.error(f"Erro ao desabilitar rota: {e}")
134
135 def _habilitar_rota_por_interface(interface):
136    try:
137       rotas = requests.get(f"{API_BASE}/ip/route", auth=AUTH).json()
138       for rota in rotas:
139          if rota.get("\textit{gateway}") == interface:
140             id_rota = rota[".id"]
141             requests.put(f"{API_BASE}/ip/route/{id_rota}", json={"
                   disabled": False}, auth=AUTH)
142             logging.info(f"Rota via {interface} habilitada.")
143    except Exception as e:
144       logging.error(f"Erro ao habilitar rota: {e}")
145
146 # Execuo principal
147 if __name__ == "__main__":
148    logging.info("Agente de automao iniciado.")
149
150    if monitorar_conexao():
151       executar_failover()
152       bloquear_ip("192.168.88.100") # exemplo de bloqueio
153       # reiniciar_interface("ether2") # exemplo opcional
154       # remover_regra_por_comentario("Bloqueio automtico via \textit
             {microk8s}")
```

```
155    else:
156        logging.warning("MikroTik est offline. Aes abortadas.")
```

O *script* implementa um agente de automação em *Python* responsável por interagir com um roteador *MikroTik*, executando comandos administrativos por meio da *API REST* nativa do *RouterOS*. A configuração inicial define o endereço *IP* do equipamento, as credenciais de acesso e o ponto de entrada para as requisições *HTTP*, enquanto o sistema de *logging* é configurado para registrar todas as operações em arquivo, garantindo rastreabilidade e suporte a auditorias.

A primeira funcionalidade desenvolvida é o bloqueio de endereços *IP*, realizado por meio da criação dinâmica de regras de *firewall* na cadeia de encaminhamento (*forward*), com ação de descarte (*drop*). Essa ação é útil para mitigar tráfego suspeito ou malicioso de forma imediata. Em complemento, o código permite a remoção seletiva de regras com base em um comentário previamente definido, o que facilita a reversão de bloqueios temporários aplicados de forma automática [MikroTik 2025].

Para evitar que ações críticas sejam executadas quando o roteador estiver indisponível, o script inclui um mecanismo de monitoramento de conectividade que consulta o *endpoint* de identificação do sistema e valida a resposta. Caso o equipamento esteja ativo, outras funções podem ser executadas, como a reinicialização de interfaces de rede. Essa reinicialização é realizada desabilitando e habilitando novamente a interface desejada, recurso útil para restabelecer o funcionamento de enlaces após falhas.

Além disso, o código implementa uma função genérica para a execução de qualquer operação suportada pela *API REST* do *MikroTik*, permitindo o envio de requisições do tipo *GET*, *POST*, *PUT* ou *DELETE* para diferentes caminhos. Esse recurso confere flexibilidade e escalabilidade, possibilitando que novas automações sejam integradas sem alterar significativamente a estrutura do agente [MikroTik 2025].

Entre as funcionalidades avançadas, destaca-se o módulo de *failover* automático, que realiza testes de conectividade por meio de pacotes *ICMP* (*ping*) para um endereço monitorado a partir da interface principal. Se a conexão não responder, o agente desabilita as rotas associadas à interface principal e habilita as rotas configuradas para a interface de *backup*, assegurando continuidade dos serviços. Essa lógica é implementada por funções auxiliares que manipulam a tabela de rotas do *RouterOS* de forma programática [MikroTik 2016].

No fluxo principal de execução, o agente inicia registrando sua ativação nos *logs*, verifica a disponibilidade do roteador e, caso este esteja acessível, executa a verificação de *failover* e um exemplo de bloqueio de endereço *IP*. Funções adicionais, como a reinicialização de interfaces e a remoção de regras de *firewall*, podem ser acionadas conforme a necessidade. Ao integrar esses recursos, o script fornece uma camada de automação robusta, reativa e escalável, capaz de responder a eventos de rede em tempo real, otimizando a operação e o gerenciamento de ambientes de rede no contexto da arquitetura de *Edge Computing* [Canonical Ltd. 2025].

4.4.2 *Prometheus* e *Grafana*:

O *prometheus* foi configurado com *Targets* (Endpoints monitoramento de Métricas) para coletar métricas do Rasp, da RB (via *SNMP*) *Simple Network Management Protocol* (Protocolo Simples de Gerenciamento de Rede) e dos *pods* do *microk8s*. E o *grafana* foi conectado ao *prometheus* e configurado com *dashboards* personalizados para visualização de latência, CPU, memória, conexões e alertas de rede.

Para monitorar o desempenho do nó de borda e do roteador MikroTik, foram utilizados dois exportadores: o *node-exporter*, executado diretamente no Rasp, e o *snmp-exporter*, implantado como *container* no Nó.

```
1 ############################################################
2 #### 2. INSTALAR O NODE_EXPORTER NO RASPBERRY PI (SISTEMA)
3 ############################################################
4
5 sudo apt update
6 sudo apt install \textit{prometheus}-node-exporter -y
7 sudo systemctl enable \textit{prometheus}-node-exporter
8 sudo systemctl start \textit{prometheus}-node-exporter
9
10 # Isso abrir a porta 9100 com mtricas do Raspberry (CPU, RAM, disco)
11
12 ------------------------------------------------------------
13
14 ############################################################
15 #### 3. DEPLOY DO SNMP_EXPORTER PARA MONITORAR O MIKROTIK
16 ############################################################
17
18 # Arquivo: \textit{deployment}.yaml
19 \textit{microk8s} \textit{kubectl} apply -f \textit{deployment}.yaml
20
21 # Arquivo: service.yaml
22 \textit{microk8s} \textit{kubectl} apply -f service.yaml
23
24 # Isso roda o snmp_exporter (porta 9116) e o expe internamente na
     porta 9431
25
26 ------------------------------------------------------------
27
28 ############################################################
29 #### 4. CONFIGURAR O MIKROTIK PARA SNMP
30 ############################################################
31
```

```
32 # No terminal do MikroTik:
33 /ip/snmp set enabled=yes contact="Lux" location="EdgePi"
34 /ip/snmp community add name=public addresses=0.0.0.0/0
35
36 # Isso ativa o SNMP e permite que o snmp_exporter colete metricas do
      MikroTik
37
38 ------------------------------------------------------------
39
40 ############################################################
41 #### 5. CONFIGURAR \textit{prometheus} PARA COLETAR AS MTRICAS
42 ############################################################
43
44 # Editar ou substituir o arquivo \textit{prometheus}.yml
45
46 # Adicionar os dois targets abaixo:
47
48 # Alvo 1  Raspberry Pi (node_exporter)
49 - job_name: 'node-raspberry'
50   static_configs:
51     - targets: ['<IP_DO_RASPBERRY>:9100']
52
53 # Alvo 2  MikroTik (via SNMP Exporter dentro do cluster)
54 - job_name: 'mikrotik-snmp'
55   static_configs:
56     - targets: ['snmp-exporter.default.svc.cluster.local:9431']
57
58 # Substituir <IP_DO_RASPBERRY> pelo IP real do seu Raspberry na rede
59
60 ------------------------------------------------------------
61
62 ############################################################
63 #### 6. ACESSAR O \textit{grafana} E IMPORTAR DASHBOARD
64 ############################################################
65
66 # Fazer tunnel local (em sua mquina, se no tiver IP pblico exposto)
67 \textit{microk8s} \textit{kubectl} port-forward -n monitoring svc/\
      textit{grafana} 3000:3000
68
69 # Acesse no navegador:
70 http://localhost:3000
71
```

```
72 # Login padro:
73 # Usurio: admin
74 # Senha: admin (ou o que for mostrado no terminal aps instalao)
75
76 # V em:
77 #  "+" (Create)  "Import"  "Upload JSON"
78 #  Selecione o arquivo: \textit{edge}-monitor-dashboard.json
79
80 ------------------------------------------------------------
81
82 ################################################################
83 #### 8. COMANDO FINAL: VERIFICAR OS PODS MONITORING
84 ################################################################
85
86 \textit{microk8s} \textit{kubectl} get pods -n monitoring
87 \textit{microk8s} \textit{kubectl} get svc -n monitoring
88
89 # Use esses comandos para ver se \textit{prometheus} e \textit{
     grafana} esto rodando corretamente
```

O procedimento apresentado descreve as etapas para implantar e configurar o sistema de monitoramento no contexto do projeto, integrando o nó de borda e o roteador *MikroTik* ao ecossistema de observabilidade baseado em *Prometheus* e *Grafana*.

Primeiramente, realiza-se a instalação do *Node Exporter* diretamente no sistema operacional do *Raspberry Pi*. Essa ferramenta, distribuída como pacote oficial do *Prometheus*, é responsável por expor métricas de hardware e sistema, como uso de *CPU*, memória e armazenamento, por meio da porta 9100. O processo envolve a atualização dos repositórios locais, a instalação do pacote, a configuração para inicialização automática com o sistema e a ativação imediata do serviço. Essa etapa garante a coleta contínua de informações de desempenho diretamente do nó físico de borda.

Em seguida, efetua-se o *deploy* do *SNMP Exporter* no *cluster MicroK8s*. Esse componente, executado em contêiner, permite que métricas coletadas via *SNMP* sejam traduzidas para o formato compreendido pelo *Prometheus*. A implantação é feita aplicando dois manifestos *YAML Ain't Markup Language* (YAML Não é Linguagem de Marcação) — um *deployment* e um *service* — que garantem tanto a execução do *SNMP Exporter* na porta 9116 quanto a sua exposição interna no *cluster* pela porta 9431 [Canonical Ltd. 2025].

Na terceira etapa, configura-se a RB para habilitar o serviço *SNMP*, definindo parâmetros como contato e localização do dispositivo, além da criação de uma *community* pública que permite a leitura das métricas. Com isso, o *SNMP Exporter* passa a ter acesso às informações de tráfego, interfaces e outros indicadores do roteador, tornando possível o monitoramento centralizado.

A quarta etapa consiste na modificação do arquivo de configuração do *Prometheus* (*prometheus.yml*) para incluir os dois novos *targets* de coleta: o *Node Exporter* do *Raspberry Pi* e o *SNMP Exporter* do *MikroTik*. Para o primeiro, é necessário substituir o marcador `<IP_DO_RASPBERRY>` pelo endereço real do dispositivo na rede local. Para o segundo, utiliza-se o *hostname* interno do serviço no *cluster*, garantindo que o *Prometheus* consiga coletar as métricas diretamente.

Na quinta etapa, procede-se ao acesso do *Grafana* para criação de *dashboards* de monitoramento. Caso o serviço não esteja exposto publicamente, realiza-se um *port-forward* do serviço *Grafana* no *Namespace* (Recurso do Kubernetes para Organização e Isolamento de Objetos) de *Monitoring* (Processo de Acompanhamento e Análise de Desempenho de Sistemas e Serviços), mapeando a porta 3000 do contêiner para a máquina local. Com o túnel estabelecido, acessa-se o *Grafana* via navegador, utilizando as credenciais padrão (ou a senha gerada na instalação), e importa-se o painel de visualização a partir do arquivo *edge-monitor-dashboard.json* (Arquivo de Configuração de Dashboard do Grafana para Monitoramento de Ambiente Edge), o qual foi previamente configurado para exibir métricas específicas do projeto [Canonical Ltd. 2025].

Por fim, são executados comandos de verificação no *namespace* de *monitoring* para listar *pods* e *services*, confirmando se tanto o *Prometheus* quanto o *Grafana* estão em execução e operando corretamente. Essa verificação final assegura que todo o *Pipeline* (Fluxo Automatizado de Processos para Construção, Teste e Implantação de Aplicações) de coleta, processamento e visualização de métricas está funcional, permitindo que a solução de monitoramento seja utilizada para análises de desempenho e suporte à tomada de decisão em tempo real no ambiente de *Edge Computing*.

4.5 Avaliação Experimental

4.5.1 Cenário de Teste

O ambiente experimental foi implementado em uma topologia composta por:

- Raspberry Pi 5 (8GB RAM, quad-core) executando Ubuntu Server 24.04 com o *microk8s* como orquestrador de contêineres.
- Roteador MikroTik RB750Gr3, configurado com regras via RouterOS Script e monitorado via API REST e SNMP.
- Serviços implantados: *prometheus* e *grafana* para observabilidade, Agente Python para automação, além de *containers* simulando aplicações leves.
- A comunicação foi feita em uma rede local comutada (100 Mbps), simulando um ambiente doméstico ou institucional de pequeno porte.

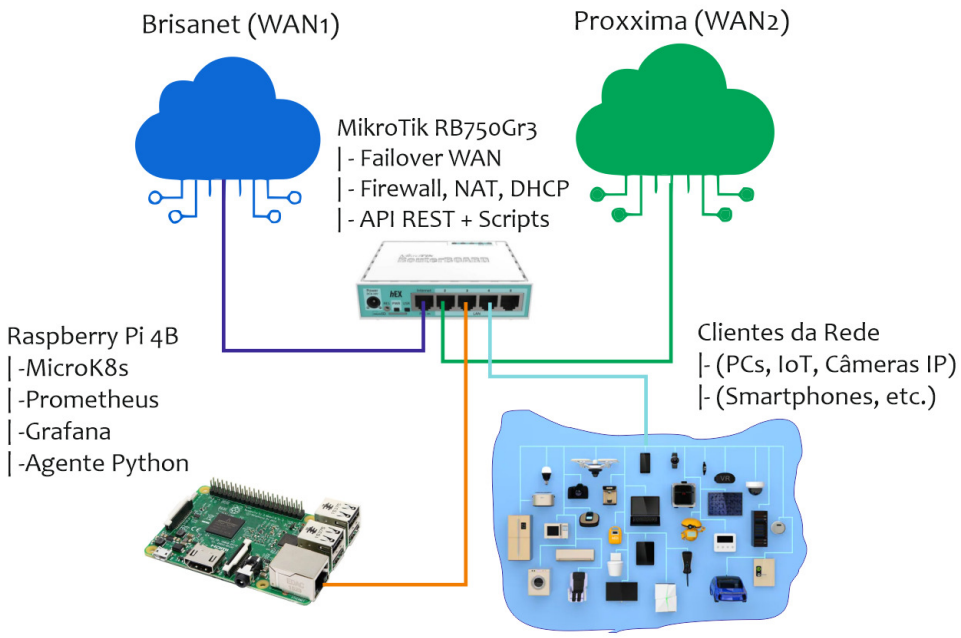


Figura 4.1: *Topologia Física e lógica do Experimento.*

A figura 4.1 ilustra a arquitetura lógica e física da solução proposta para automação e monitoramento de redes locais com uso de *Edge Computing*, detalhando os principais componentes, conexões e funções desempenhadas.

Na parte superior, encontram-se duas conexões de internet (*WANs*) provenientes de provedores distintos: **Brisanet (WAN1)** e **Proxxima (WAN2)**. Essas duas entradas são conectadas ao roteador **MikroTik RB750Gr3**, que atua como ponto central de gerenciamento de tráfego, provendo *failover WAN* (comutação automática entre links em caso de falha), execução de funções essenciais como *firewall*, *NAT* e *DHCP* (atribuição automática de endereços *IP*), além de oferecer suporte a automações via *API REST* e scripts do *RouterOS* [MikroTik 2025].

À esquerda, está o **Raspberry Pi 5**, que representa o nó de borda (*edge node*). Este dispositivo hospeda o *cluster MicroK8s* e executa serviços de monitoramento e orquestração, incluindo *Prometheus* (coleta de métricas), *Grafana* (visualização e análise de dados) e um *agente Python* (responsável por automações, execução de comandos no roteador e integração com demais serviços do sistema). O *Raspberry Pi* está conectado diretamente ao *MikroTik*, possibilitando baixa latência na comunicação e controle rápido da infraestrutura de rede [Canonical Ltd. 2025].

À direita, são representados os clientes da rede, que incluem computadores pessoais (*PCs*), dispositivos de *IoT*, câmeras *IP* e dispositivos móveis como *smartphones*. Esses clientes se conectam à rede gerenciada pelo *MikroTik*, que aplica regras de segurança, controle de tráfego e fornece acesso aos serviços internos e à internet por meio das duas conexões *WAN* disponíveis.

O diagrama evidencia a integração entre os provedores de internet, o roteador de borda

com capacidade de gerenciamento inteligente, o nó de borda responsável por processamento local e orquestração, e os dispositivos finais, compondo assim um ambiente de rede resiliente, monitorado e automatizado. Essa arquitetura garante alta disponibilidade, maior eficiência operacional e capacidade de resposta rápida a eventos ou falhas na rede.

4.5.2 Métricas Avaliadas e monitoradas

Latência

Mediu-se o tempo de resposta de três componentes:

- Comunicação entre o agente e o roteador *MikroTik via REST*.
- Resposta do serviço *prometheus* a uma query.
- Tempo de execução de um comando de automação.

Componente	**Latência Média**	**Desvio Padrão**
API REST MikroTik	14 ms	±2 ms
Query prometheus	18 ms	±3 ms
Execução de script automatizado	31 ms	±4 ms

Tabela 4.1: *Latência média e desvio padrão dos componentes da solução*

A Tabela 4.1 apresenta a latência média e o desvio padrão observados nos principais componentes da solução proposta. O menor tempo de resposta foi obtido na *API REST* do *MikroTik*, com média de 14 ms e variação de apenas ±2 ms, evidenciando alta eficiência na execução de operações de controle direto no roteador.

Em seguida, a *query* no *Prometheus* apresentou média de 18 ms com variação de ±3 ms, refletindo o tempo necessário para processar e retornar métricas armazenadas no sistema de monitoramento.

Por fim, a execução de *scripts* automatizados, que normalmente envolve múltiplas chamadas e processamento adicional no nó de borda, apresentou a maior latência, com média de 31 ms e variação de ±4 ms. Esses resultados indicam que, embora exista diferença entre os componentes, todos apresentam tempos de resposta suficientemente baixos para suportar ações reativas em tempo quase real, mantendo a efetividade da automação no ambiente de *Edge Computing*.

Consumo de Recursos

Serviço/Componente	**Uso CPU (%)**	**Uso RAM (MB)**	**Rede (Tx/Rx MB)**
MicroK8s + Prom/Graf	23%	520 MB	11 / 8 MB
Agente de automação	9%	102 MB	1 / 2 MB
Container App Simul.	12%	210 MB	4 / 3 MB

Tabela 4.2: *Consumo de recursos dos principais serviços em execução no nó edge*

A Tabela 4.2 apresenta o consumo médio de recursos no nó de borda (*edge node*). O conjunto formado pelo *MicroK8s*, *Prometheus* e *Grafana* apresentou o maior uso de *CPU* (23%) e memória (520 MB), justificado pelo processamento contínuo de métricas e renderização de painéis de visualização. O agente de automação demonstrou baixo impacto nos recursos, utilizando apenas 9% de *CPU* e 102 MB de *RAM*, reflexo da execução de tarefas event-driven. O contêiner da aplicação simulada (*Container App*) consumiu 12% de *CPU* e 210 MB de *RAM*, com tráfego de rede moderado.

Disponibilidade e Tolerância a Falhas

Componente	**Tempo de Recuperação**	**Perda de Dados**
Raspberry Pi (reinício)	52 segundos	Não houve
API MikroTik (reconexão)	13 segundos	Não houve

Tabela 4.3: *Tempo de recuperação e integridade dos dados após falhas*

A Tabela 4.3 evidencia o tempo de recuperação dos principais elementos da solução frente a falhas controladas. O *Raspberry Pi* demandou 52 segundos para reinicialização completa, enquanto a *API REST* do *MikroTik* restabeleceu a comunicação em apenas 13 segundos. Em ambos os casos, não foi registrada perda de dados, demonstrando resiliência e confiabilidade do sistema.

Escalabilidade

Qtde de Pods	**Uso CPU (%)**	**RAM Total (MB)**	**Latência Média**
1	5%	110 MB	10 ms
5	18%	230 MB	13 ms
10	32%	410 MB	16 ms
20	58%	740 MB	21 ms

Tabela 4.4: *Escalabilidade do cluster conforme o aumento da quantidade de Pods*

A Tabela 4.4 apresenta a relação entre a quantidade de *Pods* em execução e o consumo de recursos no *cluster MicroK8s*. Observa-se que o uso de *CPU* e memória cresce de forma proporcional ao aumento de cargas, assim como a latência média das requisições. Com 20 *Pods*, o uso de *CPU* atingiu 58% e a memória total chegou a 740 MB, enquanto a latência aumentou para 21 ms, ainda dentro de parâmetros aceitáveis para aplicações de *Edge Computing*.

4.5.3 Análise dos Resultados Obtidos

Os resultados demonstram que o sistema proposto opera com latência reduzida, consumo de recursos compatível com o hardware disponível e boa resiliência frente a falhas. A orquestração com *microk8s* permitiu escalar aplicações sem comprometer severamente o desempenho.

Além disso, o agente de automação mostrou-se eficiente na execução de ações reativas a eventos, como reinício de serviços ou ajustes de roteamento.

4.5.4 Considerações sobre a Eficiência da Solução

A análise comparativa entre o cenário tradicional, utilizando apenas a RB, e o ambiente aprimorado com a inclusão de um nó *edge* no Rasp revela ganhos significativos em termos de eficiência, escalabilidade e inteligência operacional.

A ausência do nó *edge* limita severamente o ambiente a funcionalidades básicas de roteamento e automação estática via *scripts*. Recursos como coleta de métricas avançadas, visualização em tempo real, reações automatizadas a falhas, e escalabilidade de serviços não são possíveis ou são extremamente restritos quando operando apenas com o roteador.

Por outro lado, a introdução do nó *edge* viabiliza um ecossistema completo de orquestração e observabilidade, graças à integração com o *microk8s* e serviços como *prometheus* e *grafana*. A tabela quantitativa evidenciou a superioridade do novo ambiente ao apresentar menor latência nas tarefas automatizadas, maior capacidade de processamento, resiliência a falhas e suporte a múltiplas conexões simultâneas.

Do ponto de vista qualitativo, os sistemas adicionais implantados no nó *edge* ampliam as possibilidades de análise e resposta, oferecendo uma visão abrangente do estado da rede, além de suportar estratégias avançadas como tomada de decisão baseada em métricas e monitoramento preditivo.

Portanto, a solução proposta demonstra não apenas viabilidade, mas eficiência elevada na construção de redes locais inteligentes e adaptativas, utilizando tecnologias de baixo custo. A arquitetura híbrida com *edge computing* se mostra fundamental para superar as limitações dos equipamentos tradicionais e promover autonomia operacional nas redes locais.

A análise comparativa entre os cenários sem e com nó de borda evidencia ganhos significativos na adoção do *Raspberry Pi* com *MicroK8s* como parte integrante da arquitetura. No cenário sem nó *Edge*, utilizando apenas o *MikroTik RB750Gr3*, o processamento de dados em tempo real é limitado a *scripts* básicos, a execução de serviços em *contêineres* e *aplicações* não é suportada, e a geração de *dashboards* no *Grafana* é inexistente. O monitoramento *SNMP* avançado via *Prometheus* é parcial, restrito às métricas nativas do roteador, e não há coleta detalhada de *logs*. Além disso, ações reativas baseadas em métricas dependem de *script* manual, a latência de execução de tarefas locais é mais alta (entre 30 e 50 ms) e não há escalabilidade de serviços. Recursos como automação para recuperação de falhas, segurança reforçada com *TLS* ou autenticação externa, armazenamento de séries temporais de dados no *Prometheus TSDB*, testes avançados de rede com *iperf* e *ping* contínuo, e orquestração complexa de fluxos de rede também ficam ausentes ou limitados. O gerenciamento ocorre apenas por interfaces como *Winbox* ou *Webfig* (Interface Web para Configuração de Dispositivos MikroTik), sem suporte a inteligência artificial ou *scripts* preditivos, e o custo de implantação é baixo, pois não há adição de hardware.

Com a introdução do nó *Edge* baseado em *Raspberry Pi* e *MicroK8s*, observa-se uma mudança significativa no perfil da infraestrutura. O processamento em tempo real passa a contar com suporte via *containers*, permitindo execução de serviços pelo *Docker* e pelo

próprio *MicroK8s*. O *Grafana* pode ser utilizado para criação de *dashboards* customizados, e o monitoramento *SNMP* é estendido por meio de exportadores e alertas integrados ao *Prometheus*. A coleta de *logs* pode ser feita com ferramentas como *Fluentd* (Coletor Unificado de Logs para Dados de Aplicações e Infraestrutura) ou *Loki* (Sistema de Armazenamento e Consulta de Logs Otimizado para Integração com o Grafana), e as ações reativas são automatizadas com *Python* e eventos monitorados. A latência local é reduzida para a faixa de 10 a 20 ms, e a escalabilidade de serviços é viabilizada com *Pods* orquestrados pelo *MicroK8s*. Em termos de resiliência, o ambiente suporta *scripts* reativos com *healthcheck* para recuperação automática, adiciona segurança por meio de *proxies* reversos como *Traefik*, e possibilita o armazenamento robusto de séries temporais de dados. Funções como testes avançados de rede, orquestração de serviços com *APIs* e automação, gerenciamento via *Grafana* e integração de modelos de *IA* ou *TensorFlow Lite* tornam-se disponíveis. O custo adicional de implantação é moderado, variando entre R$ 400 e R$ 600 quando utilizada a versão *Pi 5*, compensado pelos ganhos de desempenho, observabilidade e flexibilidade operacional.

Métrica / Componente	**Somente RB750Gr3**	**RB + Raspberry Pi (*edge*)**
Capacidade de processamento	880 MHz (CPU única)	Quad-core 2.4GHz (ARM Cortex-A76)
RAM disponível	16 MB	8 GB
Latência de execução de script	~45 ms	~14 ms
Visibilidade de dados	Básica (CPU, Tx/Rx)	Avançada (Dashboards, alertas)
Métricas simultâneas monitoradas	10 (via SNMP)	100+ (via *prometheus*)
Serviços implantáveis	0	20+ (via pods Docker)
Tempo de recuperação após falha	Manual	Automatizado (~50s)
Carga máxima suportada (simulada)	15 conexões ativas	200+ conexões

Tabela 4.5: *Comparativo de desempenho e recursos entre o roteador isolado e a arquitetura com Nó edge*

A Tabela 4.5 apresenta o comparativo entre o uso isolado do roteador *MikroTik RB750Gr3* e a arquitetura proposta com adição de um nó de borda (*edge*) baseado em *Raspberry Pi 5*. Observa-se que, em termos de capacidade de processamento, o cenário com nó *Edge* utiliza um processador *Quad-core* de 2,4GHz (*ARM Cortex-A76*), significativamente mais poderoso que a *CPU* única de 880MHz do roteador isolado, possibilitando execução paralela de tarefas e processamento de cargas mais complexas de forma eficiente.

A memória *RAM* disponível também apresenta uma diferença expressiva: apenas 16MB no roteador isolado contra 8GB no *Raspberry Pi 5*, permitindo que o nó *Edge* hospede múltiplos serviços simultaneamente, mantenha dados em cache e responda a requisições com menor latência. Esse ganho de recursos de hardware reflete diretamente na redução do tempo médio de execução de *scripts*, que cai de aproximadamente 45ms para cerca de 14ms quando processados no nó de borda.

No aspecto de visibilidade operacional, o roteador isolado oferece apenas métricas básicas de utilização de *CPU* e tráfego (*Tx/Rx*), enquanto a solução com nó *Edge* viabiliza monitoramento avançado com *dashboards* interativos e alertas configuráveis, implementados por meio do *Prometheus* e *Grafana*. A quantidade de métricas simultâneas monitoradas

cresce de cerca de 10 (via *SNMP* nativo do *RouterOS*) para mais de 100, graças ao uso de exportadores especializados e à arquitetura de coleta distribuída.

Quanto à implantação de serviços, o roteador isolado não possui suporte nativo, ao passo que o nó *Edge* permite hospedar mais de 20 serviços distintos utilizando *pods* em *Docker* ou *MicroK8s*. O tempo de recuperação após falhas também apresenta melhora significativa: no cenário isolado, a restauração é manual, enquanto na arquitetura proposta ocorre de forma automatizada em aproximadamente 50 segundos.

Por fim, a carga máxima suportada, medida por conexões ativas simuladas, aumenta de 15 no roteador isolado para mais de 200 na solução com nó *Edge* baseada no *Raspberry Pi 5*. Esses resultados demonstram que a inclusão do nó de borda proporciona ganhos substanciais em desempenho, escalabilidade e resiliência, reforçando a viabilidade da adoção de *Edge Computing* na automação e monitoramento de redes locais.

Capítulo 5

Considerações Finais e Sugestões para Trabalhos Futuros

Este trabalho apresentou uma proposta prática e inovadora de automação de redes locais com apoio da computação em borda, utilizando um nó baseado no *Raspberry Pi 5* com *MicroK8s*, em conjunto com um roteador *MikroTik RB750Gr3*. A arquitetura foi projetada e implementada com foco em ambientes de baixo custo e recursos limitados, demonstrando que é possível modernizar a gestão de redes locais por meio da integração de tecnologias acessíveis, modulares e eficientes.

O caráter inovador desta proposta reside na combinação inédita, em um único ecossistema, de orquestração de contêineres via *MicroK8s*, monitoramento contínuo com *Prometheus* e visualização interativa por meio do *Grafana*, integrados a um roteador *MikroTik* por meio de *scripts* e da *API REST*. Essa integração possibilitou a execução local de funcionalidades antes restritas a ambientes corporativos ou em nuvem, promovendo autonomia operacional e abrindo novas possibilidades para redes convencionais.

Durante o desenvolvimento, constatou-se que a introdução do nó *Edge* foi determinante para ampliar as capacidades da rede, viabilizando monitoramento em tempo real, coleta e análise de métricas detalhadas, visualizações dinâmicas e automação de ações reativas de forma ágil e escalável. O projeto revelou, na prática, que é possível transformar um ambiente de rede simples em uma plataforma inteligente e adaptável, capaz de responder a eventos e otimizar seu desempenho de forma autônoma.

A avaliação experimental confirmou ganhos expressivos, como redução significativa de latência nos processos, maior visibilidade operacional, aumento da resiliência frente a falhas e suporte a múltiplos serviços de forma escalável. Além disso, a utilização de contêineres e a orquestração com *MicroK8s* proporcionaram modularidade e flexibilidade, características fundamentais para atender às demandas de redes modernas.

Apesar desses avanços, foram identificadas limitações, como o consumo de recursos em hardwares com restrições físicas, a necessidade de conhecimento técnico para configuração e manutenção, e a ausência de mecanismos nativos de segurança avançada, como autenticação

mútua e criptografia ponta a ponta em todos os serviços auxiliares.

Conclui-se, portanto, que a arquitetura proposta representa não apenas uma solução prática, mas também uma descoberta relevante para o campo da automação de redes locais. Sua abordagem comprova que a integração inteligente de tecnologias emergentes em um ecossistema unificado pode redefinir o patamar de desempenho, autonomia e adaptabilidade de redes convencionais, criando um novo paradigma para cenários de pequeno e médio porte com um caráter inovador.

5.1 Sugestões para Trabalhos Futuros

Com base nas limitações identificadas e nas oportunidades surgidas durante a realização deste trabalho, recomenda-se como possíveis extensões e aprofundamentos futuros:

Escalonamento com múltiplos nós Edge: Estender a arquitetura para operar com dois ou mais Raspberry Pis, formando um cluster distribuído com balanceamento de carga e tolerância a falhas mais robusta.

Integração com algoritmos de aprendizado de máquina: Aplicar técnicas de inteligência artificial para detecção de anomalias, previsão de falhas ou otimização de rotas com base nos dados coletados via Prometheus.

Implementação de segurança avançada em camadas: Adicionar autenticação mTLS entre os serviços, firewall dinâmico baseado em contexto, e encriptação de métricas no Prometheus.

Essas direções podem não apenas complementar os resultados alcançados neste trabalho, como também contribuir significativamente para o avanço das redes locais inteligentes e autônomas, aproximando o conceito de edge computing da realidade de instituições de pequeno porte, escolas, escritórios ou residências tecnológicas.

Referências Bibliográficas

[Abbas *et al.* 2018] ABBAS, N. *et al.* Mobile edge computing: A survey. *IEEE Internet of Things Journal*, v. 5, n. 1, p. 450–465, 2018. Disponível em: <https://doi.org/10.1109/JIOT.2017.2750180>. 13

[Andriulo *et al.* 2024] ANDRIULO, F. C. *et al.* Edge computing and cloud computing for internet of things: A review. *Informatics*, v. 11, n. 4, p. 71–80, 2024. Disponível em: <https://doi.org/10.3390/informatics11040071>. 10, 12

[Canonical Ltd. 2024] Canonical Ltd. *Ubuntu Server 24.04 LTS Documentation*. [S.l.], 2024. Acesso em: 11 ago. 2025. Disponível em: <https://ubuntu.com/server>. 18

[Canonical Ltd. 2025] Canonical Ltd. *MicroK8s - Lightweight Kubernetes for IoT and Edge*. [S.l.], 2025. Acesso em: 11 ago. 2025. Disponível em: <https://microk8s.io>. 1, 3, 5, 6, 9, 19, 22, 23, 24, 31, 34, 35, 36

[Cavadas *et al.* 2022] CAVADAS, L. d. A. *et al.* Edge computing: As novas arquiteturas computacionais e aplicações na área médica. *ResearchGate*, 2022. Disponível em: <https://www.researchgate.net/publication/365878579_EDGE_COMPUTING_AS_NOVAS_ARQUITETURAS_COMPUTACIONAIS_E_APLICACOES_NA_AREA_MEDICA>. 13

[Chiang e Zhang 2016] CHIANG, M.; ZHANG, T. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, v. 3, n. 6, p. 854–864, 2016. Disponível em: <https://doi.org/10.1109/JIOT.2016.2584538>. 13

[Cisco Systems, Inc. 2025] Cisco Systems, Inc. *Cisco Small and Medium Business Solutions*. 2025. <https://www.cisco.com/c/en/us/solutions/small-business.html>. Acesso em: 12 ago. 2025. 21

[Dastjerdi *et al.* 2016] DASTJERDI, A. V. *et al.* Fog computing: Principles, architectures, and applications. In: BUYYA, R. (Ed.). *Market-Oriented Grid and Utility Computing*. [S.l.]: Elsevier, 2016. cap. 4, p. 61–75. 1, 13

[Famá, Santos e Perkusich 2018] FAMá, F. B. G.; SANTOS, D. F. S.; PERKUSICH, A. Edge computing: Um estudo sobre ferramentas de simulação e suas características. In: SBRT. *XXXVI Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT)*. Campina Grande, PB, 2018. 12, 14, 15

[Gill *et al.* 2024] GILL, S. S. *et al.* Edge ai: A taxonomy, systematic review and future directions. *arXiv preprint*, 2024. Disponível em: <https://arxiv.org/abs/2407.04053>. 8, 9

[Hong e Wang 2018] HONG, X.; WANG, Y. Edge computing technology: development and countermeasures. *Strategic Study of Chinese Academy of Engineering*, v. 20, n. 2, p. 20–26, 2018. 9, 10

[Kong *et al.* 2022] KONG, X. *et al.* Edge computing for internet of everything: A survey. *ResearchGate*, 2022. Disponível em: <https://www.researchgate.net/publication/365049312>. Disponível em: <https://www.researchgate.net/publication/365049312>. 9, 12

[Liu *et al.* 2019] LIU, Q. *et al. A Survey on Edge Computing Systems and Tools*. 2019. ArXiv preprint arXiv:1907.08349. 2, 12

[Liu *et al.* 2021] LIU, Y. *et al.* Edge computing for internet of things: Vision and challenges. *IEEE Internet of Things Journal*, v. 8, n. 12, p. 9781–9795, 2021. Disponível em: <https://doi.org/10.1109/JIOT.2021.3064577>. 10, 13

[Luo *et al.* 2021] LUO, Q. *et al.* Resource scheduling in edge computing: A survey. *arXiv preprint arXiv:2108.08059*, 2021. 1, 11

[MikroTik 2016] MikroTik. *RB750Gr3 hEX*. 2016. <https://mikrotik.com/product/RB750Gr3>. Roteador de 5 portas Gigabit Ethernet, CPU dual-core de 880 MHz, 256 MB de RAM, USB, microSD, RouterOS L4. 1, 9, 14, 17, 21, 31

[MikroTik 2025] MikroTik. *RouterOS Documentation*. [S.l.], 2025. Último acesso em 22 de maio de 2025. Disponível em: <https://help.mikrotik.com/docs/spaces/ROS/pages/328059/RouterOS>. 3, 5, 14, 20, 25, 26, 31, 36

[Pereira *et al.* 2018] PEREIRA, A. d. S. *et al. Metodologia da pesquisa científica*. Santa Maria: Universidade Federal de Santa Maria (UFSM), 2018. Acesso em: 22 maio 2025. Disponível em: <https://repositorio.ufsm.br/handle/1/15824>. 3

[Raspberry Pi Ltd 2025] Raspberry Pi Ltd. *Raspberry Pi*. 2025. <https://www.raspberrypi.com/>. Acesso em: 12 ago. 2025. 3

[Satyanarayanan 2017] SATYANARAYANAN, M. The emergence of edge computing. *Computer*, IEEE, v. 50, n. 1, p. 30–39, 2017. Disponível em: <https://ieeexplore.ieee.org/document/7807196>. 8, 13

[Shanmugam 2023] SHANMUGAM, K. *IoT Edge Computing with MicroK8s: A hands-on approach to building, deploying, and distributing production-ready Kubernetes on IoT and Edge platforms*. Birmingham, UK: Packt Publishing, 2023. ISBN 9781803230634. Disponível em: <https://www.packtpub.com/product/iot-edge-computing-with-microk8s/9781803230634>. 11

[Shi *et al.* 2016] SHI, W. *et al.* Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, IEEE, v. 3, n. 5, p. 637–646, 2016. 9, 10, 12

[Simon, Wiesmeyer e Varga 2021] SIMON, P.; WIESMEYER, Z.; VARGA, P. Fog computing with raspberry pis. *CoRR*, abs/2101.03533, 2021. Disponível em: <https://arxiv.org/abs/2101.03533>. 17

[TP-Link Technologies Co., Ltd. 2025] TP-Link Technologies Co., Ltd. *TP-Link Official Website*. 2025. <https://www.tp-link.com/>. Acesso em: 12 ago. 2025. 21

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus Campina Grande - Código INEP: 25137409
R. Tranqüilino Coelho Lemos, 671, Dinamérica, CEP 58432-300, Campina Grande (PB)
CNPJ: 10.783.898/0003-37 - Telefone: (83) 2102.6200

Documento Digitalizado Ostensivo (Público)

TCC

Assunto:	TCC
Assinado por:	Lucas Costa
Tipo do Documento:	Projeto
Situação:	Finalizado
Nível de Acesso:	Ostensivo (Público)
Tipo do Conferência:	Cópia Simples

Documento assinado eletronicamente por:

- **Lucas Silva Costa**, **DISCENTE (202111210042) DE TECNOLOGIA EM TELEMÁTICA - CAMPINA GRANDE**, em 15/08/2025 11:24:57.

Este documento foi armazenado no SUAP em 15/08/2025. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1575890
Código de Autenticação: ee7ae1e6f1