UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
MESTRADO ACADÊMICO EM ENGENHARIA ELÉTRICA

JOÃO LUIZ PONTES DE ARAÚJO

TESTBED DE ARQUITETURAS IOT ESCALÁVEIS PARA APLICAÇÕES DE SISTEMAS INTELIGENTES

DM: 36/2025

BELÉM-PA
2025

JOÃO LUIZ PONTES DE ARAÚJO

TESTBED DE ARQUITETURAS IOT ESCALÁVEIS PARA APLICAÇÕES DE SISTEMAS INTELIGENTES

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica como requisito parcial para obtenção de grau de Mestre em Engenharia Elétrica, pela Universidade Federal do Pará.

Orientador: Prof. Dr. Wellington da Silva Fonseca

BELÉM-PA
2025

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a) autor(a)

P813t Pontes De Araújo, João Luiz.
Testbed de Arquiteturas IoT Escaláveis para Aplicações de Sistemas Inteligentes / João Luiz Pontes De Araújo. — 2025.
178 f. : il. color.

Orientador(a): Prof. Dr. Wellington da Silva Fonseca
Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Belém, 2025.

1. Internet das Coisas. 2. Arquiteturas Distribuídas. 3. Virtualização. 4. Observabilidade. 5. Teste de Carga. I. Título.

CDD 004.6782

JOÃO LUIZ PONTES DE ARAÚJO

TESTBED DE ARQUITETURAS IOT ESCALÁVEIS PARA APLICAÇÕES DE SISTEMAS INTELIGENTES

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica como requisito parcial para obtenção de grau de Mestre em Engenharia Elétrica, pela Universidade Federal do Pará.

Orientador: Prof. Dr. Wellington da Silva Fonseca

Data de aprovação: 31/10/2025

Conceito: Aprovado

Banca Examinadora

_____

Prof. Dr. Wellington da Silva Fonseca
Presidente / PPGEE – UFPA

_____

Prof. Dr. Miércio Cardoso de Alcântara Neto
Membro Interno / PPGEE – UFPA

_____

Prof. Dr. Fabricio José Brito Barros
Membro Interno / PPGEE – UFPA

_____

Prof. Dr. Otavio Andre Chase
Membro Externo / ICIBE – UFRA

À memória, que resiste ao tempo.

À lembrança, que o transcende.

Dedico.

AGRADECIMENTOS

Agradeço, incondicionalmente, à minha família, minha maior riqueza e maior privilégio. Ao meu pai, João Batista, pela sua mansidão e firmeza, pelo amor ao trabalho e à perseverança, características as quais tenho uma inestimável admiração. À minha mãe, Maria do Socorro, por me ensinar a generosidade e a benevolência, a não agir mal ou ter pensamento semelhante, e por sempre manter nosso lar unido. Às minhas irmãs, Rita de Cássia e Roberta de Cássia, pela fraternidade, respeito e afeto, pelos quais me alegro. À minha amada companheira de vida, Maria de Lourdes, pelos sonhos compartilhados, pelo cuidado e cumplicidade, por me trazer certeza, inspiração, paz e conforto sempre que precisei. Aos demais familiares, parentes e amigos que sempre acreditaram em mim.

Agradeço ao Prof. Dr. Wellington da Silva Fonseca pela orientação e oportunidade de desenvolver a presente dissertação; muitos dos métodos e resultados elogiados foram sua ideia. À Drª. Elen Priscila de Souza Lobato pela mentoria, pelo apoio teórico e técnico e pela parceria de sucesso. Aos membros da banca — Prof. Dr. Miércio Cardoso de Alcântara Neto, Prof. Dr. Fabrício José Brito Barros e Prof. Dr. Otavio Andre Chase — pelas valorosas contribuições ao desenvolvimento da dissertação e apoio ao longo de minha jornada acadêmica. Aos colegas do Centro de Excelência em Eficiência Energética da Amazônia (CEAMAZON) e do Laboratório de Concepção de Análise de Dispositivos Elétricos (LCADE), que participaram direta ou indiretamente do presente trabalho e publicações relacionadas.

Por último, agradeço e relembro a memória dos que já não se encontram mais neste plano, mas que deixaram sua marca no caminho que me trouxe até esse momento.

Agradeço!

Assim como o olhar só ascende à luz
quando todo o corpo se orienta para ela,
também a alma só contempla o ser
quando inteira se converte para fora das
sombras.

Platão — A República, 518c–d

RESUMO

Esta dissertação apresenta o projeto, a implantação e a avaliação de arquiteturas de Internet das Coisas (IoT) voltadas para aplicações em sistemas inteligentes, por meio de um *testbed* desenvolvido no contexto do Projeto de Pesquisa e Desenvolvimento (P&D) Desenvolvimento Tecnológico e Extensão Inovadora de Materiais Avançados em Energia e Mobilidade aplicados ao contexto amazônico (EMOB-AMAZON). A proposta busca oferecer uma abordagem modular e escalável, baseada em uma arquitetura composta por quatro camadas: física (*hardware*), *middleware*, processamento e *software*. Para isso, foram empregadas tecnologias de virtualização (Proxmox VE) e conteinerização (Docker) na implantação de quatro máquinas virtuais. Três delas foram destinadas à configuração de ambientes de desenvolvimento baseados nas plataformas IoT Dojot, FIWARE e Node-RED, enquanto a quarta foi dedicada à implantação de um sistema de observabilidade e monitoramento, estruturado a partir da *stack* Prometheus-Grafana, que abrange as ferramentas Prometheus, Grafana, InfluxDB, cAdvisor e Grafana k6. A avaliação da escalabilidade foi conduzida por meio do monitoramento contínuo de métricas técnicas do sistema (CPU, RAM, disco e rede), enquanto a robustez das arquiteturas foi verificada com testes de carga que simularam cenários de verificação, típicos e extremos de uso. Os resultados obtidos revelaram aspectos operacionais e de desempenho relevantes, demonstrando que as plataformas mantiveram níveis aceitáveis de consumo, conforme os recursos de *hardware* alocados a cada máquina virtual, além de apresentarem estabilidade e latência compatíveis com os cenários simulados e seus respectivos limiares estabelecidos, evidenciando padrões, vantagens e limitações de cada ambiente de desenvolvimento. Por fim, as análises realizadas forneceram subsídios técnicos importantes para a compreensão da escalabilidade e da robustez das arquiteturas IoT implantadas, fundamentando tanto o dimensionamento e direcionamento de cada solução IoT dentro do escopo do projeto quanto a expansão do sistema de observabilidade e monitoramento.

Palavras-chave: Internet das Coisas; Arquiteturas Distribuídas; Virtualização; Observabilidade; Teste de Carga.

ABSTRACT

This dissertation presents the design, implementation, and evaluation of Internet of Things (IoT) architectures for applications in intelligent systems, using a testbed developed in the context of the Research and Development (R&D) Project Technological Development and Innovative Extension of Advanced Materials in Energy and Mobility applied to the Amazonian context (EMOB-AMAZON). The proposal seeks to offer a modular and scalable approach, based on an architecture composed of four layers: physical (hardware), middleware, processing, and software. To this end, virtualization (Proxmox VE) and containerization (Docker) technologies were employed in the implementation of four virtual machines. Three of them were used to configure development environments based on the Dojot, FIWARE, and Node-RED IoT platforms, while the fourth was dedicated to the implementation of an observability and monitoring system, structured from the Prometheus-Grafana stack, which includes the Prometheus, Grafana, InfluxDB, cAdvisor, and Grafana k6 tools. Scalability was assessed through continuous monitoring of technical system metrics (CPU, RAM, disk, and network), while the robustness of the architectures was verified with load tests that simulated typical and extreme usage scenarios. The results revealed relevant operational and performance aspects, demonstrating that the platforms maintained acceptable consumption levels, according to the hardware resources allocated to each virtual machine, in addition to presenting stability and latency compatible with the simulated scenarios and their respective established thresholds, highlighting patterns, advantages, and limitations of each development environment. Finally, the analyses provided important technical insights for understanding the scalability and robustness of the deployed IoT architectures, supporting both the sizing and targeting of each IoT solution within the scope of the project and the expansion of the observability and monitoring system.

Keywords: Internet of Things; Distributed Architectures; Virtualization; Observability; Load Testing.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

1. INTRODUÇÃO

Este capítulo introdutório apresenta o contexto da pesquisa, destacando suas justificativas e motivações. Na sequência, são definidos o objetivo geral e os objetivos específicos. Por fim, descreve-se a estrutura da dissertação, organizada em capítulos que conduzem da fundamentação teórica ao desenvolvimento experimental e análise dos resultados.

1.1. VISÃO GERAL

As infraestruturas convencionais do sistema de energia têm, em sua forma básica, apenas uma direção de movimentação de energia por meio das fases de geração, transmissão, distribuição e consumo, que são responsáveis pelo fluxo seguro e eficiente de energia de ponta a ponta (Colak; Bayindir; Sagiroglu, 2020). No entanto, a necessidade de controle de supervisão preciso e oportuno em cada estágio da rede significa que o arranjo requer uma infraestrutura digitalizada e interconectada (El-Dessouki; Saeed, 2021).

Quando incrementados com o advento das Tecnologias da Informação e da Computação (TIC), os sistemas de energia se tornam *Smart Grids* (SG), que se caracterizam pela modernização e digitalização da rede por meio de um fluxo bidirecional de energia e informações, possibilitando uma ampla gama de funcionalidades e aplicações ao melhorar a capacidade de controle de todo o sistema, integrando a geração e o consumidor a fim de garantir o fornecimento eficiente de eletricidade, com capacidade e área de cobertura suficientes, acessíveis, seguras e confiáveis (Geng; Zhang; Zhu, 2023; Qays *et al.,* 2023).

De acordo com a literatura, uma rede inteligente deve ter uma ou mais características fundamentais em sua configuração, tais como: confiabilidade, segurança, gerenciamento do lado da demanda, medição e verificação, inserção de energias renováveis e sistemas de *self-healing*[1] (Majeed Butt; Zulqarnain; Majeed Butt, 2021; Nguyen *et al.,* 2021). Nesse sentido, as redes inteligentes são vistas como um novo paradigma no setor elétrico para a próxima geração em termos de

[1] Refere-se a sistemas que podem detectar e corrigir automaticamente falhas, como no caso de sistemas de distribuição de energia elétrica.

confiabilidade, disponibilidade e eficiência energética, gerando maior bem-estar socioeconômico nessa esfera, devido à crescente demanda por eletricidade, impulsionada pela urbanização, mudança nos padrões de vida e desenvolvimento tecnológico (Mu *et al.,* 2023; Routray *et al.,* 2021).

No entanto, a transformação da rede elétrica tradicional em um SG, ou a criação de uma SG desde o estágio conceitual até a implementação prática, exige um nível avançado de colaboração, integração e interoperabilidade[2] entre diferentes áreas do conhecimento (Voumick; Deb; Khan, 2021). Essa complexidade é acentuada pela falta de padrões de rede amplamente aceitos ou difundidos, o que dificulta a expansão da rede e a incorporação efetiva de aplicativos avançados, medidores e sensores inteligentes, dispositivos e recursos de energia renovável, prejudicando sua interoperabilidade (Archana; Shankar; Singh, 2022; Song *et al.,* 2022).

Simultaneamente, enquanto as SG se consolidam e utilizam energias renováveis como elemento estrutural básico, fontes como a solar e a eólica ganham destaque no Sistema Interligado Nacional (SIN) brasileiro. Isso se deve principalmente às aplicações de Geração Distribuída (GD), que em 2023 apresentou um aumento de aproximadamente 68% em relação a 2022, devido a incentivos fiscais e ações regulatórias, por meio da Micro e Minigeração Distribuída (MMGD) (Empresa de Pesquisa Energética, 2024), e à agenda global de acordos e políticas de curto, médio e longo prazo em vigor contra as mudanças climáticas, causadas principalmente pelo consumo e queima de combustíveis fósseis (Ministério de Minas e Energia; Empresa de Pesquisa Energética, 2020).

Assim, tecnologias alternativas surgiram como opções plausíveis para a geração de eletricidade limpa, como as Plantas Fotovoltaicas Flutuantes (PFF), que se desenvolveram rapidamente nas últimas décadas devido a fatores como simplicidade e confiabilidade de operação e manutenção, potencial de escalabilidade e baixos custos de produção de energia (Campana *et al.,* 2019).

Não obstante, a integração de tecnologias promissoras e emergentes cria um ambiente complexo e heterogêneo no qual diferentes plataformas e protocolos de

---

[2] Interoperabilidade refere-se à capacidade de diferentes sistemas, tecnologias ou organizações interagirem e compartilharem informações de forma eficiente e segura, mesmo que sejam desenvolvidos por diferentes fornecedores ou utilizem diferentes tecnologias.

comunicação precisam interagir (Song *et al.,* 2022). Nesse caso, a aplicação de uma infraestrutura baseada em *middleware* de Internet das Coisas (IoT), tem a principal vantagem de resolver a diversidade de dispositivos para padrões de comunicação por meio da interoperabilidade, unificando a maneira como esses dispositivos interagem e trocam informações (Da Cruz *et al.,* 2018; Pradeep; Krishnamoorthy; Vasilakos, 2021; Zhang *et al.,* 2021). Isso facilita a implementação de sistemas complexos de IoT, em que dispositivos de diferentes fabricantes e funcionalidades podem operar juntos (Armando; Sá Silva; Boavida, 2021; Figueredo; Seed; Subotic, 2020).

Sendo assim, independentemente das diferentes visões que as infraestruturas de IoT podem assumir, seu principal objetivo é interconectar diversos dispositivos por meio de um único domínio ou *Internet Protocol* (IP). Dessa forma, permite a exploração e o compartilhamento de informações por meio dessa conexão, apoiando a tomada de decisões estratégicas e o estabelecimento de serviços e aplicativos inteligentes, o que possibilita o estabelecimento de um ecossistema dotado de diferentes funções, como comunicação, processamento, análise e detecção (Bkheet; Agbinya, 2021; Chettri; Bera, 2020; Čolaković; Hadžialić, 2018; Voumick; Deb; Khan, 2021).

Consequentemente, como essas arquiteturas se tornaram componentes essenciais de sistemas modernos, as quantidades massivas de dados gerados e a necessidade de processamento em tempo real configuram desafios recorrentes (Natu *et al.,* 2016; Shinde, 2016). Portanto, garantir a confiabilidade e funcionamento adequado entre as diversas camadas da arquitetura e falhas oportunas devido à natureza distribuída desses sistemas são fundamentais, exigindo recursos robustos de observabilidade, monitoramento e diagnóstico (Becker *et al.,* 2020; Korontanis *et al.,* 2023; Natu *et al.,* 2016; Shinde, 2016; Thantharate, 2023).

Por conseguinte, por meio dessas aplicações, as informações podem ser exploradas e compartilhadas por meio dessa conexão de forma segura e confiável, apoiando a tomada de decisões estratégicas e o estabelecimento de serviços e aplicativos inteligentes.

Dessa forma, a partir dos paradigmas e tecnologias apresentadas, foi proposta a implantação de 3 arquiteturas de *middlewares* IoT conteinerizadas baseadas nas plataformas Dojot, FIWARE e Node-RED, implantadas por meio de Máquinas Virtuais (VM), subsidiárias à interoperabilidade, centralização e normalização de dados

oriundos do monitoramento de arranjos de usinas fotovoltaicas, interligadas à rede elétrica da cidade universitária Prof. José Silveira Netto, da Universidade Federal do Pará (UFPA). Para cada arquitetura foram realizadas análises de desempenho, por meio de métricas técnicas e de estatística descritiva referentes ao consumo de *hardware* pelas VMs e os respectivos contêineres de suas aplicações, coletadas e processadas por uma *Stack*[3] de observabilidade composta do InfluxDB, Prometheus, cAdvisor e Grafana.

Portanto, o foco desta dissertação é a construção de um *testbed* que reúne e valida as arquiteturas implantadas com *middlewares* IoT, analisando de forma detalhada o desempenho de cada um deles, bem como sua aplicação na geminação e no monitoramento de sistemas fotovoltaicos integrados ao contexto da pesquisa. Não obstante, as arquiteturas e plataformas selecionadas têm como objetivo a otimização do uso de recursos, sejam eles financeiros, computacionais ou humanos, o que é proporcionado pela integração de tecnologias *Open Source*[4] e de baixo custo computacional como a virtualização de máquinas e a virtualização baseada em contêineres, auxiliadas pela documentação dos e desenvolvimento de uma documentação de suporte à metodologia estabelecida.

As principais contribuições da dissertação incluem:

- **Desenvolvimento de uma metodologia abrangente e replicável**: a dissertação descreve uma abordagem sistemática da implantação dos *middlewares* IoT, assim como as técnicas e tecnologias empregadas, a fim de estabelecer um ambiente de alta portabilidade e replicabilidade devidamente documentado;

- **Integração de tecnologias *Open Source***: a pesquisa exemplifica a integração de tecnologias, por meio da virtualização de máquinas e da virtualização baseada em contêineres, utilizando *softwares* como Proxmox VE e Docker, assim como uma pilha de observabilidade e monitoramento, baseada

[3] Refere-se a um conjunto integrado de tecnologias, ferramentas, serviços ou camadas de *software* que trabalham em conjunto para atender a um propósito específico — como o desenvolvimento, execução, monitoramento ou manutenção de aplicações.

[4] Um *software open source* consiste em um *software* que tem seu código fonte disponibilizado e licenciado com uma licença de código aberto no qual o direito autoral fornece o direito de estudar, modificar e distribuir o *software* de graça para qualquer um e para qualquer finalidade.

em bancos de dados de séries temporais, mecanismos de busca e visualização, os quais tem como objetivo monitorar o uso de recursos, potencializando a otimização de recursos computacionais, financeiros e humanos;

- **Auxílio à escalabilidade de servidores físicos**: as informações geradas a partir dos resultados alcançados por meio da metodologia aplicada são de fundamental importância para o dimensionamento e gestão dos recursos de *hardware* de servidores físicos, subsidiando a escalabilidade destes sistemas.

1.2. CONTEXTO DA PESQUISA

Seguindo o legado de projetos como o Sistema Inteligente Multimodal da Amazônia (SIMA), pioneiro em mobilidade elétrica nos modais rodoviário e aquaviário (Lobato *et al.,* 2021), em 2020, foi aprovado, por meio da Chamada Pública do Plano Nacional de Ciência e Tecnologia MCTI/FINEP/AT - Materiais Avançados e Minerais Estratégicos 2020 e financiado pela Financiadora de Estudos e Projetos (FINEP), o projeto de Pesquisa e Desenvolvimento (P&D) Desenvolvimento Tecnológico e Extensão Inovadora de Materiais Avançados em Energia e Mobilidade aplicados ao contexto amazônico (EMOB-AMAZON).

O projeto, desenvolvido na UFPA, tem como objetivo promover o desenvolvimento do setor de energia e mobilidade na região amazônica utilizando tecnologias e materiais avançados de forma inovadora. Seu principal objetivo concentra-se em desenvolver e implementar soluções sustentáveis e inovadoras para atender às demandas de energia elétrica e mobilidade na região Amazônica, utilizando tecnologias e materiais avançados, como materiais de terras raras[5].

Para que o objetivo geral do projeto seja alcançado, foram instalados 3 arranjos fotovoltaicos na orla do *campus* básico da UFPA, em Belém, dotados de sistemas de

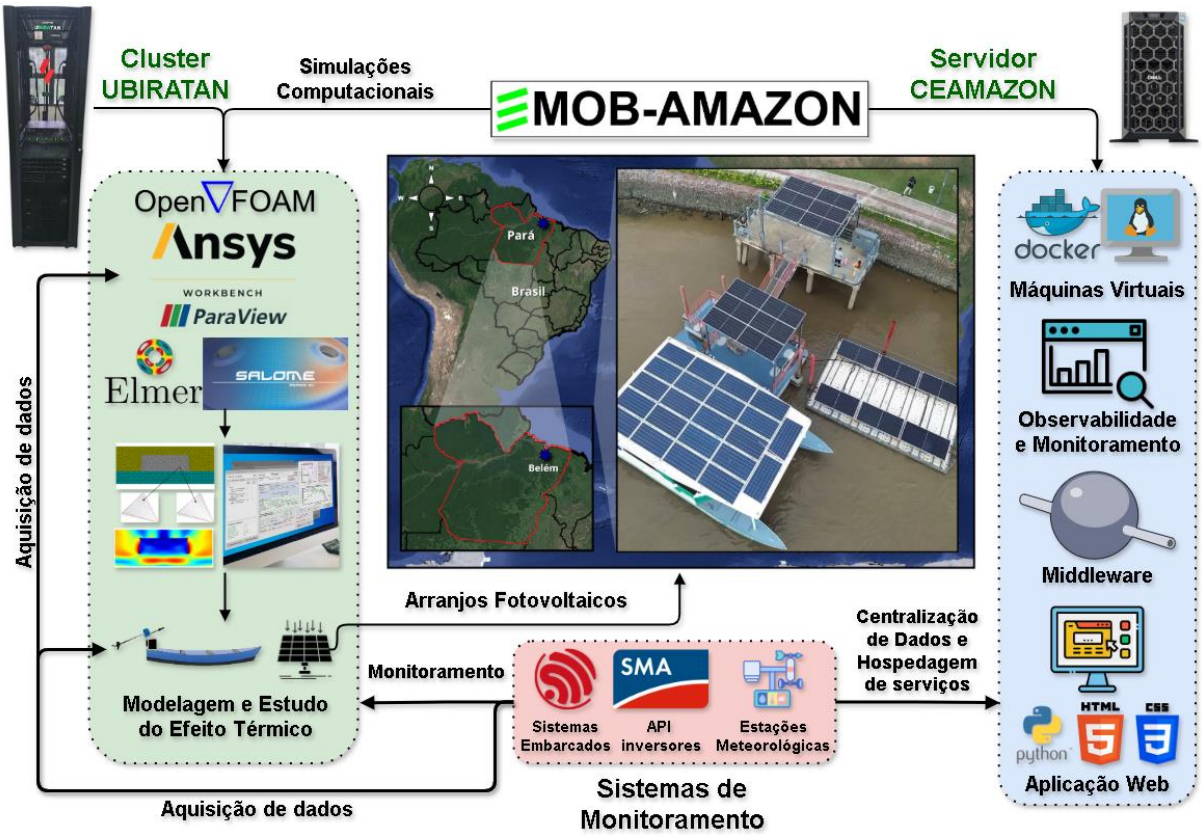
[5] As terras raras são um conjunto de 17 elementos químicos: lantânio, cério, praseodímio, neodímio, promécio, samário, európio, gadolínio, térbio, disprósio, hólmio, érbio, escândio, túlio, itérbio, lutécio e ítrio. Esses minerais estratégicos têm um papel importante na transição energética pois são utilizados na indústria de alta tecnologia, desde a produção de baterias e ímãs até a fabricação de eletrônicos e energia renovável.

telemetria[6] e interligados entre si e conectados à rede elétrica da universidade (*on-grid*).

O sistema fotovoltaico instalado também conta com um 1 PFF no Rio Guamá, 1 arranjo em uma plataforma de ferro flutuante e 1 arranjo fixo na cobertura de um píer na orla da UFPA, que conecta a parte flutuante ao solo por meio de uma passarela. De forma complementar, o projeto ainda prevê a construção de uma embarcação de pequeno porte movida por motor elétrico, sendo este motor projetado e desenvolvido especialmente para a aplicação, utilizando ímãs permanentes de terras raras para a propulsão da rabeta[7].

A Figura 1 ilustra a visão geral do projeto EMOB-AMAZON.

Figura 1 – Visão geral do projeto EMOB-AMAZON.



Fonte: Elaborado pelo autor (2025).

[6] Entende-se como "telemetria" as medições realizadas de forma remota por dispositivos físicos de aquisição de dados, como sistemas embarcados ou estações meteorológicas.
[7] Rabeta é um tipo de motor de popa alongado, usado principalmente em rios da Amazônia e regiões ribeirinhas. Ele tem um eixo comprido com hélice na ponta, permitindo navegar em águas rasas, com galhadas e vegetação

O EMOB-AMAZON conta ainda com uma infraestrutura de apoio, composta por um *cluster* (ou “supercomputador”) de *High-Performance Computing* (HPC), utilizado em simulações computacionais, um servidor físico, responsável por normalizar e centralizar dados oriundos de diferentes fontes e hospedar aplicações, e dispositivos de aquisição de dados, como sistemas embarcados, estações meteorológicas e os inversores comunicativos dos arranjos fotovoltaicos. Haja vista que cada sistema ou dispositivo englobado na execução do projeto é passível de gerar dados, todos estes serão direcionados a um *middleware* IoT, seja ele Dojot, FIWARE ou Node-RED, que assumirá a função de receber, armazenar e enviados os dados, quando solicitados, para as demais aplicações inseridas no projeto.

Por conseguinte, todas as aplicações previstas no escopo do projeto têm como objetivo passivo o atendimento à modais estratégicos da comunidade acadêmica da UFPA e das populações vizinhas, como comunidades ribeirinhas, ao passo que o sistema fotovoltaico desenvolvido visa complementar a demanda de estabelecimentos universitários próximos à área da aplicação, como o restaurante universitário.

1.3. JUSTIFICATIVAS

Dentre as justificativas para o desenvolvimento da presente dissertação, a primeira se concentra na sua relevância teórica, principalmente quanto a plataforma Dojot. Nesse caso, literatura ainda possui um número limitado de produções, mesmo embora a grande maioria destas concentrem-se em aplicações de IoT similares ao desígnio do projeto sobre o qual o tema da pesquisa se insere, como visualizado em Lisboa *et al.,* (2025), Lobato *et al.,* (2022), Moraes *et al.,* (2021), Araújo *et al.,* (2022), Andrade *et al.,* (2022), Terada *et al.,* (2022) e Santos *et al.,* (2020).

Além disso, a pesquisa se justifica, também, por meio da relevância prática para sociedade civil e comunidade acadêmica, ao integrar um sistema subsidiário a uma aplicação de bem comum, os sistemas fotovoltaicos e os demais objetivos do projeto P&D EMOB-AMAZON, como evidenciado no contexto da pesquisa. A implantação da infraestrutura de *middlewares* IoT e do sistema de observabilidade e monitoramento facilitará a interoperabilidade entre os sistemas e fontes de dados englobados pelo

projeto, a gestão e otimização de recursos de *hardware* e o planejamento da escalabilidade em sistemas físicos, seja ela vertical ou horizontal[8].

Não obstante, a literatura consultada para selecionar os métodos para análise de desempenho indica que aplicações de observabilidade e monitoramento não configuram mais questões puramente técnicas, mas englobam tópicos transversais e estratégicos, essenciais para o sucesso de uma operação (Cortez *et al.,* 2017; Niedermaier *et al.,* 2019). Dentro desse contexto, são encontrados desafios, como o aumento da dinâmica e complexidade de arquiteturas e a heterogeneidade de sistemas, e possíveis soluções, como a aplicações de abordagens holísticas por meio da colaboração e comunicação entre diferentes camadas do sistema e colaboradores (Niedermaier *et al.,* 2019). Para esse último caso, temos como exemplo a seguinte abordagem:

> Propomos um método analítico holístico para operações de *data center*. Em nosso método, a obtenção de uma visão holística requer a combinação de dados de máquina, energia e carga de trabalho; fazer isso com dados de longo prazo e de granularidade zero permite descobertas significativas (Pradeep; Krishnamoorthy; Vasilakos, 2021, p. 6).

Por último, destaca-se a viabilidade técnica da pesquisa, justificada pelo uso principal dos *middlewares* IoT implantados, a exemplo da plataforma Dojot como em Lisboa *et al.,* (2025), assim como usos acessórios do servidor no qual foram alocados, como acesso remoto de usuários e gestores participantes do projeto, assim como a viabilidade financeira, haja vista que as plataformas selecionadas para o estudo são gratuitas para uso e customização conforme as necessidades do usuário.

1.4. MOTIVAÇÃO

A principal motivação da pesquisa consta na implantação de um ambiente de desenvolvimento para cada *middleware* IoT, customizado e adequado aos objetivos

[8] A escalabilidade vertical e horizontal são estratégias para lidar com o crescimento de um sistema. A escalabilidade vertical *(scale-up*) envolve adicionar mais recursos a um único nó (como mais memória ou um processador mais rápido), enquanto a escalabilidade horizontal (*scale-out*) envolve adicionar mais nós (servidores) ao sistema para distribuir a carga.

do projeto EMOB-AMAZON, responsável por garantir a interoperabilidade, centralização e normalização de dados da arquitetura IoT proposta, e continuamente monitorado por meio do sistema de observabilidade e monitoramento estabelecido, orientando a detecção de anomalias por meio de análises estatísticas, garantindo informações fundamentais ao dimensionamento de recursos computacionais e à escalabilidade do sistema.

1.5. OBJETIVO GERAL

Mensurar, através de um sistema de observabilidade e monitoramento, o desempenho dos *middlewares* IoT Dojot, FIWARE e Node-RED, encapsulados em contêineres Docker em um ambiente virtualizado, por meio de métricas técnicas e de estatística descritiva, a fim de fundamentar ações de planejamento e dimensionamento da escalabilidade horizontal e vertical do servidor físico no qual estão hospedadas.

1.5.1. OBJETIVOS ESPECÍFICOS

Para que o definido objetivo geral fosse alcançado, foram estabelecidos os seguintes objetivos específicos:

a) Implantar os *middlewares* IoT Dojot, FIWARE e Node-RED em um ambiente de desenvolvimento, alocando-os em VMs, por meio da técnica de virtualização baseada em contêineres, utilizando a plataforma Docker, em um servidor físico gerenciado pelo hipervisor Proxmox VE;

b) Estabelecer um sistema de observabilidade e monitoramento, através do hipervisor Proxmox VE, o banco de dados de séries temporais InfluxDB, o monitor de eventos e alertas Prometheus, o monitor de contêineres cAdvisor e a aplicação *web* de visualização interativa Grafana, para a análise geral e detalhada de cada VM e serviço (contêiner) dos *middlewares* IoT por meio de métricas de sistemas; e

c) Mensurar e identificar anomalias de desempenho de cada *middleware* IoT por meio da análise de estatística das métricas de uso de CPU, uso de RAM, Leitura e Escrita (L/E) em disco e tráfego de Entrada e Saída (E/S) de rede no ambiente de desenvolvimento das VMs e os respectivos contêineres

das plataformas, calculando as métricas estatísticas de Média (μ), Mediana ($\mu\delta$), Desvio Padrão, e os Percentis 90 (Π90), 95 (Π95) e 99 (Π99).

1.6. ESTRUTURA DO TRABALHO

A presente dissertação é composta por 7 capítulos, mais as referências bibliográficas e os apêndices. Sua estrutura foi organizada da seguinte forma:

a) Capítulo 2 – Tecnologias utilizadas: Apresenta as tecnologias, ferramentas e metodologias adotadas no trabalho, fundamentando as escolhas técnicas do projeto.

b) Capítulo 3 – Revisão de literatura: Revisa estudos e trabalhos relacionados, contextualizando o tema e destacando contribuições existentes e lacunas na área.

c) Capítulo 4 – *Testbed* de arquiteturas IoT para aplicações de sistemas inteligentes: Descreve a implementação do ambiente de testes (*testbed*), incluindo configurações, arquiteturas IoT avaliadas e critérios de análise.

d) Capítulo 5 – Resultados e análises: Apresenta e discute os dados obtidos nos experimentos, comparando desempenho, eficiência e limitações das soluções testadas.

e) Capítulo 6 – Conclusão: Sintetiza os principais achados, responde aos objetivos propostos e sugere direcionamentos para pesquisas futuras.

2. REVISÃO DE LITERATURA

No presente capítulo será apresentada a revisão de literatura realizada, por meio da análise bibliométrica do estado da arte dos temas centrais da dissertação e trabalhos correlatos encontrados na literatura sobre as principais tecnologias utilizadas, enfatizando aplicações com os *middlewares* IoT utilizados no desenvolvimento da pesquisa. Por conseguinte, são descritos os critérios e métodos utilizados para a realização da revisão, desenvolvendo-a cronologicamente.

2.1. DESCRIÇÃO DA PESQUISA

A revisão de literatura desempenha um papel central na construção teórica e metodológica da dissertação, pois permite mapear o estado da arte sobre o tema investigado, identificar lacunas existentes na produção científica e fundamentar conceitualmente o percurso analítico adotado (Chigbu; Atiku; Du Plessis, 2023; Garrod, 2023; Salih, 2024). Da mesma forma, a integração da análise bibliométrica e da análise quantitativa de conteúdo oferece uma metodologia robusta para identificar tendências e lacunas, combinando tendências quantitativas com análises temáticas aprofundadas (Costa *et al.,* 2023; Hazari, 2023).

Além disso, também serve para verificar o estado da arte em um campo específico, ao sintetizar o conhecimento existente, os pesquisadores podem mapear o cenário atual da pesquisa, identificando as principais teorias, metodologias e descobertas .

Dessa forma, para garantir rigor e relevância, a presente revisão foi conduzida com base em critérios bem definidos de seleção, análise e organização das fontes consultadas. Sendo assim, o presente subtópico concentrou-se em explorar a visão geral do estado da arte da temática a qual a dissertação abrange.

Em primeiro lugar, para a constatação do estado da arte da pesquisa, considerou-se a relevância e a pertinência dos materiais em relação ao objeto de estudo. Foram selecionadas obras cuja contribuição teórica ou empírica dialoguem diretamente com os temas centrais da pesquisa. A cronologia das publicações, fator determinante para identificar o nível mais avançado de desenvolvimento do tema da pesquisa, foi delimitada entre os anos de 2020 e 2025, principalmente no que se refere

à incorporação de referências (Chigbu; Atiku; Du Plessis, 2023; Garrod, 2023; Hazari, 2023).

Este recorte temporal visa garantir que a dissertação esteja alinhada aos debates e descobertas mais recentes na área. No entanto, durante o desenvolvimento dos tópicos subsequentes, como na elaboração da metodologia e discussão dos resultados, foram admitidas publicações anteriores a esse período as quais contivessem informações relevantes ao delineamento metodológico ou apresentassem revisões históricas fundamentais para a compreensão do tema, quando necessário.

Outro critério relevante considerado foi a qualidade e a credibilidade das fontes utilizadas, a partir do qual foram priorizadas produções acadêmicas publicadas em periódicos científicos e conferências indexadas em bases de dados amplamente reconhecidas pela comunidade científica, consultados através da base Scopus. Essa seleção buscou assegurar o rigor científico das informações utilizadas como base da argumentação, além de garantir uma ampla representatividade da produção nacional e internacional (Chigbu; Atiku; Du Plessis, 2023; Hazari, 2023).

Não obstante, a integração de várias metodologias em revisões de literatura aumenta a capacidade de identificar lacunas e verificar o estado da arte, a exemplo de metodologias híbridas para revisão da literatura, como a combinação da mineração de texto e revisão sistemática manual, que oferece uma maneira rápida e eficaz de identificar os principais conceitos e lacunas de pesquisa (Abdallah; Khalil, 2024; Angioi; Hiller, 2023; Costa *et al.,* 2023).

Portanto, o levantamento bibliográfico foi realizado de forma sistemática e criteriosa, com estratégias de busca bem definidas e palavras-chave específicas, garantindo uma cobertura abrangente e relevante do tema. Essa abordagem metodológica não apenas validou a pertinência da pesquisa, como também destacou a originalidade da proposta ao diferenciá-la dos estudos já consolidados na literatura.

2.2. ANÁLISE BIBLIOMÉTRICA

Para caracterizar quantitativamente o estado da arte da temática central desta dissertação - IoT e *middlewares* - conduziu-se uma análise bibliométrica integrada a técnicas de mineração de texto. Esta abordagem permite a construção sistemática de

redes de coocorrência de termos-chave a partir do corpus literário, conforme metodologias validadas por Abdallah; Khalil, (2024), Chigbu; Atiku; Du Plessis, (2023) e Hazari, (2023). O estudo empregou baseada nos seguintes passos:

1. Coleta de Dados:

Extração de publicações científicas da base Scopus, utilizando as *strings* de busca:

I. *"(TITLE-ABS-KEY (internet AND of AND things) AND TITLE-ABS-KEY (middleware)) AND PUBYEAR > 2020 AND PUBYEAR < 2025 AND (LIMIT-TO (DOCTYPE, "cp") OR LIMIT-TO (DOCTYPE, "ar"))";*

II. *"(TITLE-ABS-KEY (dojot) AND TITLE-ABS-KEY (middleware) AND TITLE-ABS-KEY (node-red)) AND PUBYEAR > 2020 AND PUBYEAR < 2025 AND (LIMIT-TO (DOCTYPE, "cp") OR LIMIT-TO (DOCTYPE, "ar"))".*

Onde lê-se que: as palavras/termos-chave são "*internet of things*", "*middleware*", "*dojot*", "*fiware*" e "*node-red*"; o intervalo de tempo é limitado entre 2020 e 2025 e; os tipos de publicações são do tipo artigos de revistas/periódicos (ar) e artigos de conferência (cp).

2. Processamento Analítico:

Análise bibliométrica realizada no ambiente RStudio v4.2.1 com o pacote Bibliometrix v.5.0.1, contemplando:

a) Informações gerais sobre os temas gerais e específicos;

b) Mapa temático geral e específico;

c) Tópicos emergentes gerais e específicos;

Mineração de texto: processada no VOSviewer 1.6.20 para:

1. Construção de redes de coocorrência de termos e identificação de clusters temáticos via algoritmo de normalização de associação (Lin/Long *modularity*) (Newman, 2004; Noack, 2007, 2009).

Realizada a consulta com os critérios estabelecidos, as referências bibliográficas selecionadas para a execução da análise foram exportadas nos formatos *BibTex* (.bib) e *Research Information Systems* (.ris) e processadas nos *softwares* mencionados.

2.2.1. ANÁLISE BIBLIOMÉTRICA DOS TEMAS CENTRAIS

Para os temas centrais da pesquisa, as informações gerais resultantes, expostas na Tabela 1, revelam um cenário dinâmico, porém com desafios, durante o recorte temporal definido.

Tabela 1 – Informações gerais da análise bibliométrica geral.

Descrição	Resultado
Principais informações sobre os dados	
Intervalo de tempo	2020:2025
Fontes	438
Documentos	709
Taxa de crescimento anual (%)	-34.97
Idade média do documento (anos)	3.17
Média de citações por documento	10.61
Conteúdo dos documentos	
Keywords Plus (ID)	4681
Palavras-chave dos autores (DE)	1952
Autores	
Autores	2365
Autores de documentos com um único autor	43
Colaboração dos autores	
Documentos com um único autor	47
Coautores por documento	4.04
Coautorias internacionais (%)	23.7
Tipos de documentos	
Artigo	309
Artigo de conferência	400

Fonte: Elaborado pelo autor (2025).

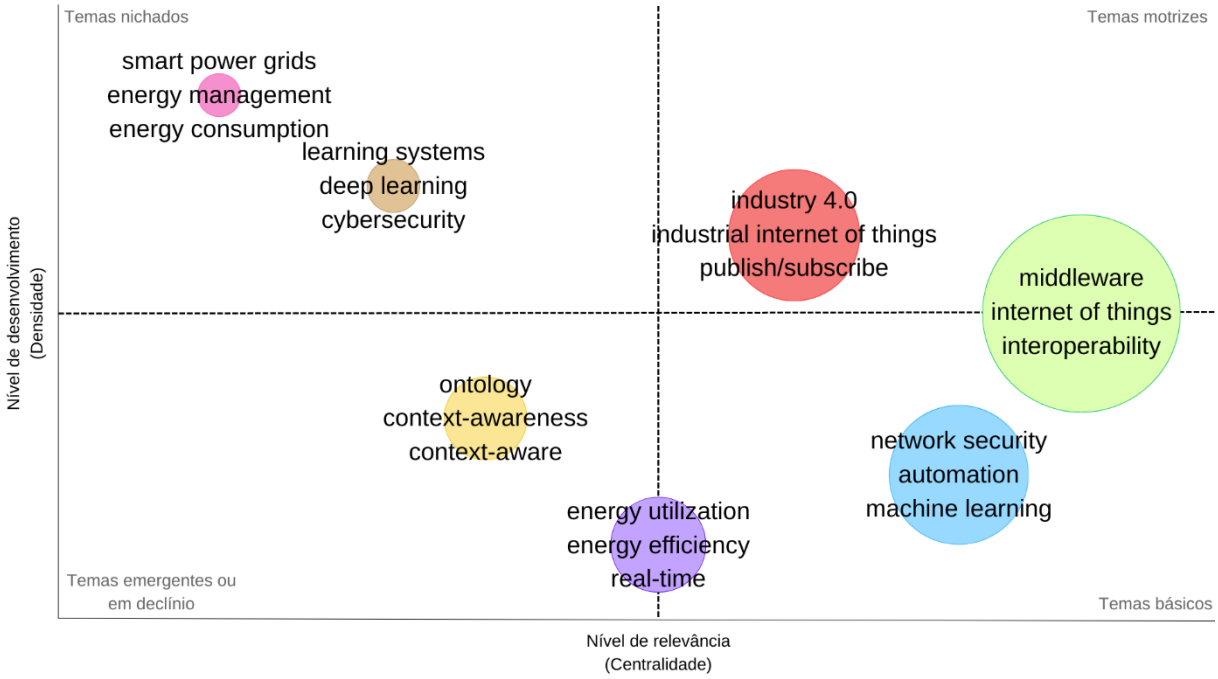
O volume de 709 documentos, publicados em 438 fontes distintas, demonstra dispersão temática, sugerindo um nicho emergente, com alta colaboração internacional (23.7% dos documentos) e uma média de 4.04 coautores por trabalho, indicativo de redes de pesquisa robustas. Apesar do impacto significativo (10.61 citações por documento, em média), a produção científica declinou acentuadamente (taxa de crescimento anual de -34.97%), sugerindo possíveis mudanças terminológicas ou saturação em subtemas.

A predominância de artigos de conferência (56.4% do total) sobre artigos tradicionais (43.6%) reflete o caráter aplicado do campo, enquanto a disparidade entre 4.681 Keywords Plus e 1.952 palavras-chave dos autores aponta para oportunidades

de padronização terminológica. A idade média dos documentos (3.17 anos) confirma a atualidade do debate, mas a queda na produção exige investigação sobre tendências emergentes. Por último, 43,6% das publicações são de revistas revisadas por pares, indicando a maturidade científica, enquanto 54,6% são artigos de conferência, indicando pesquisas aplicadas e experimentais, refletindo a consolidação teórica e as inovações recentes, respectivamente.

Quanto a distribuição dos temas e suas classificações, a Figura 2 apresenta o mapa temático para os temas gerais, que identifica e elenca as palavras-chave entre os temas nichados, motrizes, emergentes ou em declínio e temas básicos.

Figura 2 – Mapa temático dos temas IoT e *middlewares*.



Fonte: Elaborado pelo autor (2025).

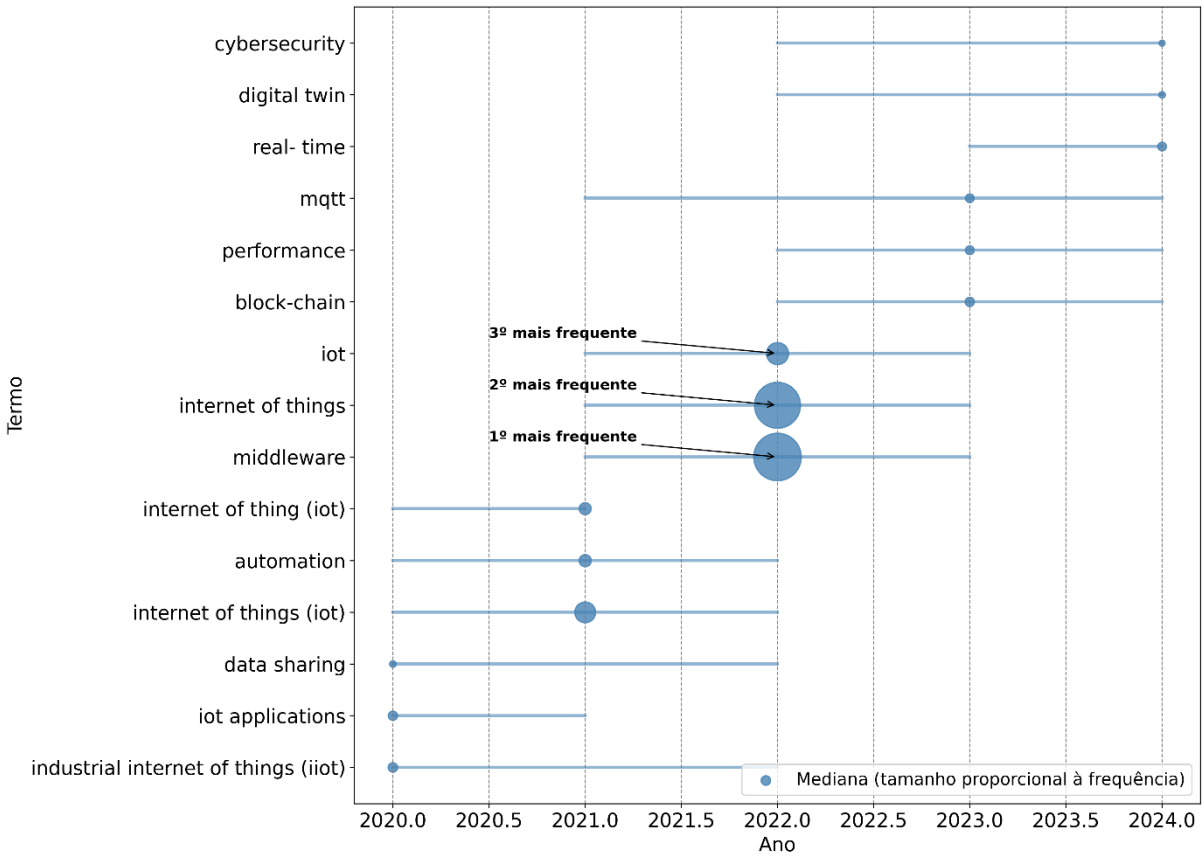
No quadrante superior direito estão os temas motrizes, que são desenvolvidos e centrais, dentre os quais destacam-se "*middleware*", "*internet of things*" e "*interoperability*", fundamentais para integrar sistemas e dispositivos em arquiteturas IoT. No quadrante inferior direito estão os temas básicos, como "*machine learning*", "*automation*" e "*network security*", os quais são relevantes, mas ainda em consolidação, com grande potencial de crescimento.

À esquerda, os temas nichados (superior esquerdo), como "*smart power grids*", "*energy management*" e "*cybersecurity*", são bem desenvolvidos, porém aplicáveis em contextos específicos. No quadrante inferior esquerdo ficam os temas emergentes ou

em declínio, como “*ontology*”, “*context-awareness*” e “*energy efficiency*”. São pouco desenvolvidos e ainda não centrais, podendo indicar áreas novas ou em desuso. Temas como “*industry* 4.0” e “*industrial internet of things*” aparecem próximos ao centro, sugerindo equilíbrio entre maturidade e relevância, com forte presença no contexto da transformação digital.

Por último, corroborando e ratificando o conteúdo do mapa temático, a Figura 3 apresenta os tópicos emergentes quanto aos temas gerais e sua distribuição em quartis durante o intervalo temporal consultado.

Figura 3 – Distribuição dos tópicos em destaque para os temas IoT e *middleware*s.



Fonte: Elaborado pelo autor (2025).

Os três termos mais frequentes são “*middleware*”, “*internet of things*” e “*iot*”, todos com pico de ocorrência em 2022. O destaque para “*middleware*” como o termo mais frequente reforça seu papel central na arquitetura de sistemas IoT, atuando como camada de integração entre dispositivos, aplicações e serviços. Já os termos “*internet of things*” e sua variação abreviada “*iot*” confirmam a consolidação do conceito como núcleo temático nas discussões tecnológicas e científicas recentes.

Outros termos com presença relevante são *automation*, “*industrial internet of things* (iiot)” e “*iot applications”*, que surgem predominantemente em torno de 2021. Isso indica uma intensificação do interesse por aplicações industriais e automatizadas da IoT nesse período. A presença desses termos sinaliza um avanço nas discussões sobre como a IoT está sendo aplicada em cenários práticos, especialmente em ambientes produtivos.

Termos como “*cybersecurity”*, “*digital twin”*, “*real-time”* e “*mqtt”* aparecem com menor frequência e com datas mais recentes, indicando áreas emergentes ou tópicos de suporte que vêm ganhando espaço gradualmente. Esses temas complementam o ecossistema IoT, oferecendo soluções voltadas à segurança, simulação digital, comunicação em tempo real e protocolos de transmissão.

De forma geral, o gráfico revela que 2022 foi o ano de maior concentração de termos centrais, refletindo um pico na produção e no interesse científico em torno da IoT. A predominância de conceitos estruturais, como *middleware*, demonstra a preocupação com a consolidação das infraestruturas que sustentam os sistemas inteligentes e conectados.

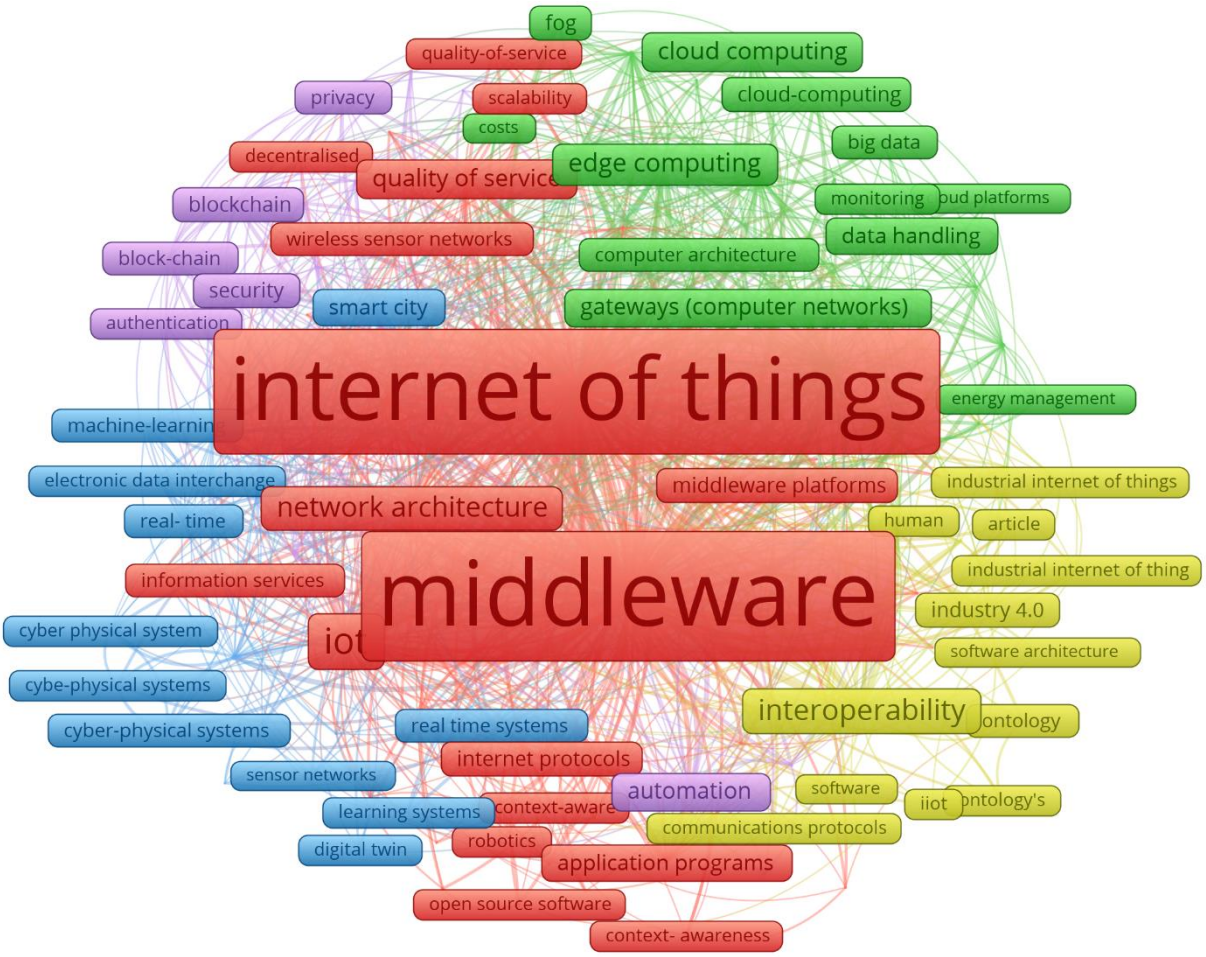
Por último, no ambiente do VOSviewer, para a construção da rede bibliométrica, foram encontradas 5.729 palavras-chave, sendo que, delimitado um limiar mínimo de 10 ocorrências de cada palavra-chave para compor a rede, 132 atingiram o estipulado. A rede gerada evidencia a estrutura temática do campo de estudo, revelando os principais tópicos, conexões entre eles e a centralidade de certos conceitos na literatura recente (Figura 4).

A partir das 132 palavras-chave que atenderam os critérios, foram identificados 5 *clusters* principais. Juntos, esses agrupamentos tematizam um panorama de pesquisa em que tecnologias de ponta, práticas de infraestrutura e desafios de segurança se entrelaçam, apontando para a IoT não apenas como uma rede de dispositivos, mas como um ecossistema integrado que exige abordagens multidisciplinares para promover inovação, sustentabilidade e resiliência.

O grafo de coocorrência gerado demonstra o quão destacados são os temas centrais "*internet of things*" e "*middleware*", representando o agrupamento vermelho, sendo fortemente conectados entre si e com diversos outros termos, evidenciando sua

importância como eixos estruturantes na área de IoT. Ao redor desses termos, destacam-se os *clusters* complementares verde, azul, amarelo e roxo.

Figura 4 – Rede de coocorrência de palavras-chave no contexto dos temas gerais.



Fonte: Elaborado pelo autor (2025).

O cluster verde reúne termos como “*cloud gateways*”, “*computer architecture*”, “*cloud computing*” e “*edge computing*”, indicando um foco em soluções para processamento e armazenamento distribuído. Esses conceitos são fundamentais para viabilizar a escalabilidade e a conectividade de dispositivos em ambientes de IoT. O *cluster* azul agrupa termos como “*smart city*”, “*machine learning*”, “*cyber-physical systems*”, “*real-time systems*” e “*digital twin*”, apontando para a aplicação de inteligência artificial e simulações digitais em sistemas urbanos e industriais. O destaque recai sobre tecnologias que viabilizam decisões autônomas e respostas em tempo real. O *cluster* amarelo concentra termos como “*interoperability*” e “*semantics*”, relacionados à integração e compreensão entre diferentes dispositivos, plataformas e padrões. Esses conceitos são essenciais para garantir a comunicação eficiente em

ecossistemas heterogêneos de IoT. Por fim, o *cluster* roxo aborda temas como "*blockchain"*, "*network security"* e "*access control"*, evidenciando preocupações com a segurança, privacidade e integridade dos dados em redes IoT, especialmente em aplicações críticas.

2.2.2. ANÁLISE DOS TEMAS ESPECÍFICOS

Já no cenário focado aos temas específicos (Dojot, FIWARE e Node-RED), as informações gerais da análise, compiladas na Tabela 2 revelam um campo de pesquisa ativo, mas com desafios maiores em comparação aos temas gerais.

Tabela 2 – Informações gerais da análise bibliométrica específica.

Descrição	Resultados
Principais informações sobre os dados	
Intervalo de tempo	2020:2025
Fontes	511
Documentos	818
Taxa de crescimento anual (%)	-14.4
Idade média do documento (anos)	2.57
Média de citações por documento	6.312
Conteúdo dos documentos	
Keywords Plus (palavras frequentes que não estão no título)	5211
Palavras-chave dos autores	2243
Autores	
Autores	3013
Autores de documentos com um único autor	25
Colaboração dos autores	
Documentos com um único autor	26
Coautores por documento	4.36
Coautorias internacionais (%)	17.85
Tipos de documentos	
Artigo	258
Artigo de conferência	560

Fonte: Elaborado pelo autor (2025).

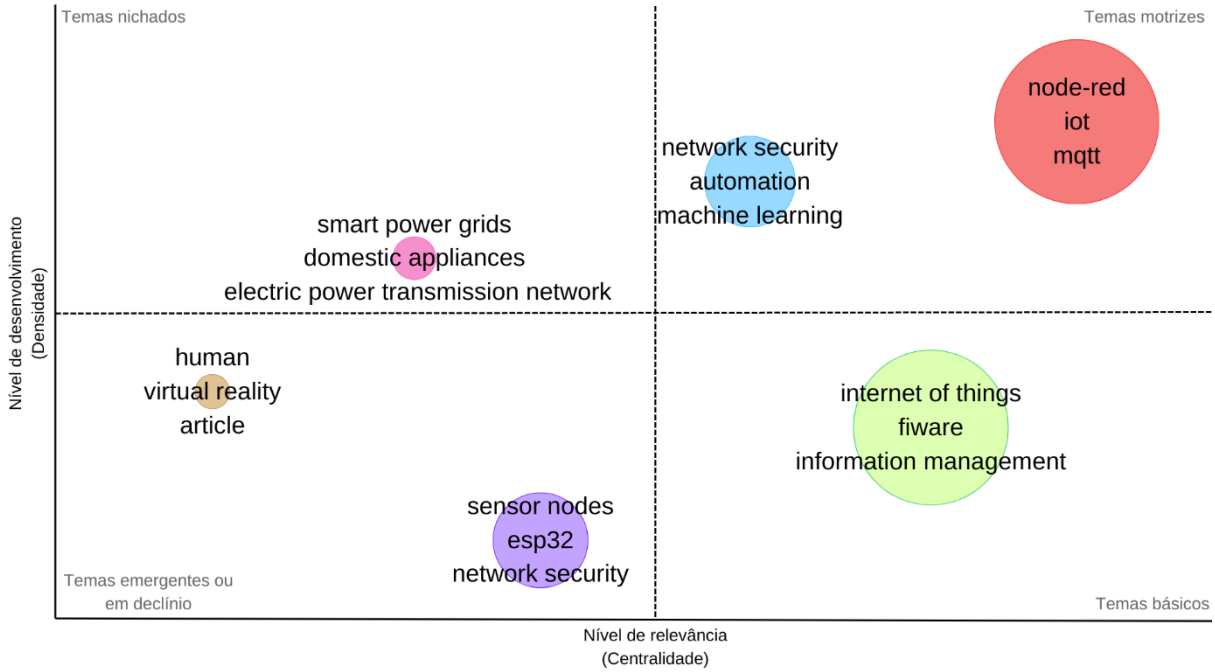
O campo demonstra expressiva atividade de pesquisa, com 818 documentos publicados em 511 fontes distintas (2020-2025), revelando significativa diversidade temática. Contudo, a taxa de crescimento anual negativa (-14.4%) aponta para uma desaceleração preocupante na produção acadêmica, sugerindo possível saturação temática ou transição para novas terminologias. Os trabalhos apresentam impacto moderado (6.31 citações/documento) e atualidade relevante (idade média de 2.57

anos), com nítido predomínio de publicações em conferências (68.5%), indicando forte orientação aplicada.

Apesar da intensa colaboração entre pesquisadores (4.36 autores por trabalho), observa-se limitada internacionalização (17.85% de coautoria internacional). A marcante diferença entre termos indexados (5.211 Keywords Plus) e palavras-chave autorais (2.243) evidencia oportunidades para aprimoramento na indexação, enquanto a rara autoria singular (apenas 3.2% dos documentos) consolida o perfil colaborativo da área. Por fim, novamente, assim como nos tópicos gerais, a predominância de artigos publicados em conferência indica o caráter experimental e tendências emergentes de pesquisa.

A distribuição dos temas e suas classificações para esse cenário, como visualizado na Figura 5, indica que tecnologias como Node-RED e MQTT lideram o cenário atual como soluções consolidadas e integradoras, enquanto FIWARE e ESP32 representam vetores de inovação para espaços de amadurecimento. Por outro lado, conceitos como "*virtual reality*" e "*domestic appliances*" permanecem mais restritos a aplicações específicas.

Figura 5 – Mapa temático para os termos Dojot, FIWARE e Node-RED.



Fonte: Elaborado pelo autor (2025).

No quadrante superior direito estão os temas motrizes, que são centrais e bem desenvolvidos, sendo, neste caso, "*node-red*", "*iot*" e "*mqtt*", que aparecem com

destaque, representando tecnologias amplamente utilizadas na implementação de soluções IoT. A presença desses termos nessa posição mostra que eles são elementos-chave no avanço e integração de sistemas conectados, além de estarem bem estabelecidos tanto em pesquisa quanto em aplicações práticas.

No quadrante inferior direito encontram-se os temas básicos, como "*internet of things"*, "fiware" e "*information management*," os quais têm alta centralidade, ou seja, são fundamentais para o campo, mas ainda não atingiram o mesmo grau de desenvolvimento dos temas motrizes. Isso indica que, embora sejam pilares conceituais e técnicos, ainda há espaço para consolidação e amadurecimento, especialmente no que diz respeito à implementação e padronização de soluções.

O quadrante superior esquerdo, que corresponde aos temas nichados, apresenta tópicos bem desenvolvidos, mas que possuem menor centralidade, como "*smart power grids"*, "*domestic appliances"* e "*electric power transmission network"*. Isso sugere que são aplicados em contextos mais específicos, com maturidade técnica consolidada, mas impacto restrito ao seu nicho de aplicação, como energia elétrica e eletromobilidade.

Já o quadrante inferior esquerdo, que representa os temas emergentes ou em declínio, contém termos como "*human"*, "*virtual reality"* e "*article"*, indicando menor densidade e baixa centralidade. São tópicos que ainda não ganharam força suficiente no contexto analisado ou cuja relevância pode estar diminuindo, talvez por falta de foco específico em soluções de IoT.

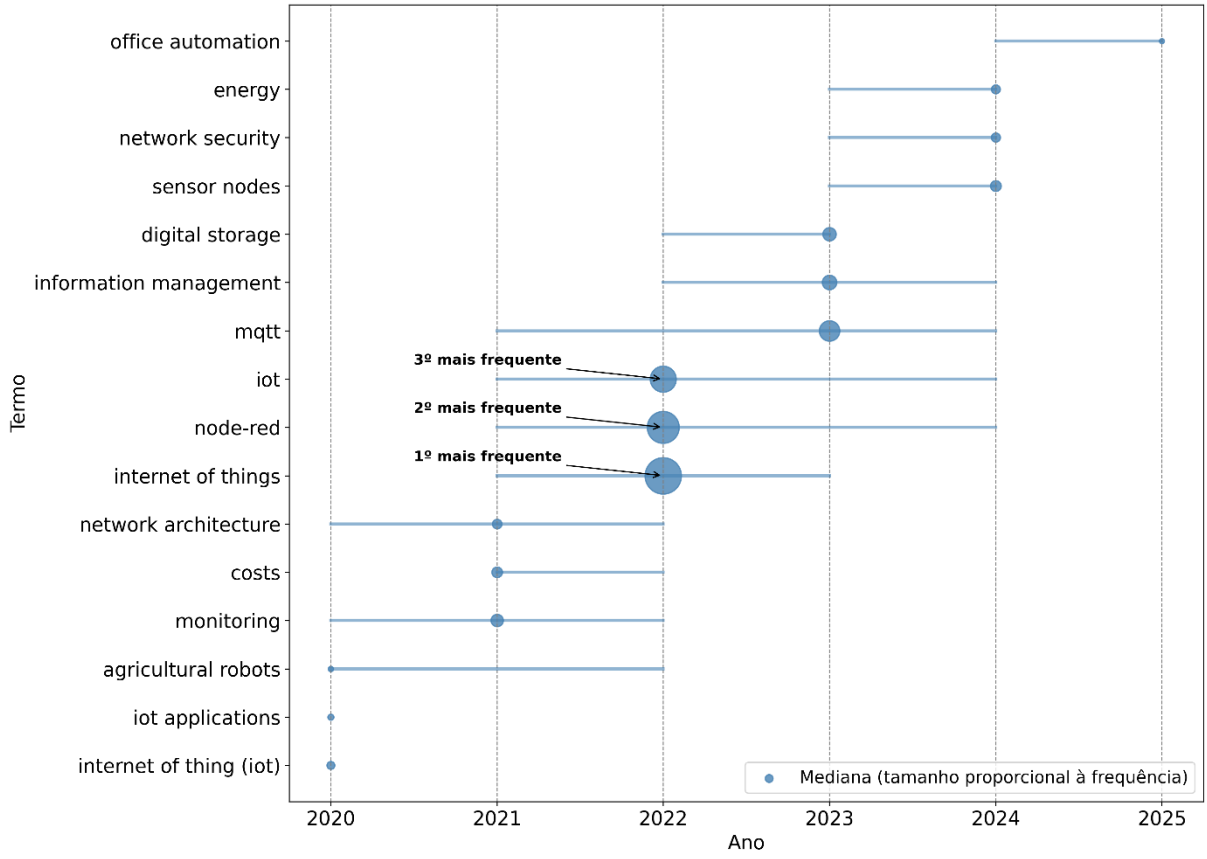
Por último, um destaque especial deve ser dado ao grupo de termos no quadrante inferior esquerdo, mas próximos da linha central, como "*sensor nodes"*, "*esp32*" e "*network security"*. Apesar de estarem em zona de baixa densidade, sua posição relativamente central indica que são temas emergentes com alto potencial de crescimento, sobretudo no que se refere ao desenvolvimento de *hardware* e segurança em dispositivos embarcados.

Quanto aos tópicos em destaque, ilustrado na Figura 6, os três termos mais frequentes são "*internet of things"*, "*node-red"* e "*iot"*, todos com mediana em 2022. O destaque para "*internet of things"* como o termo mais frequente confirma seu papel central e consolidado na literatura, tanto em linhas gerais quanto em específicas. Node-RED aparece como uma ferramenta importante na implementação de fluxos de

automação em sistemas IoT, enquanto “*iot*“ reforça o uso da sigla de forma ampla nos estudos, evidenciando uma forte padronização do termo.

Outros termos como “*mqtt*”, “*information management*” e “*digital storage*” também ganham relevância a partir de 2022, sugerindo um aumento no interesse por protocolos de comunicação e gestão de dados em sistemas conectados. Termos como “*sensor nodes*”, “*network security*” e “*energy*” surgem com mediana mais recente, indicando tópicos com potencial de crescimento ou foco atual em novas pesquisas.

Figura 6 – Distribuição dos tópicos em destaque para os termos Dojot, FIWARE e Node-RED.

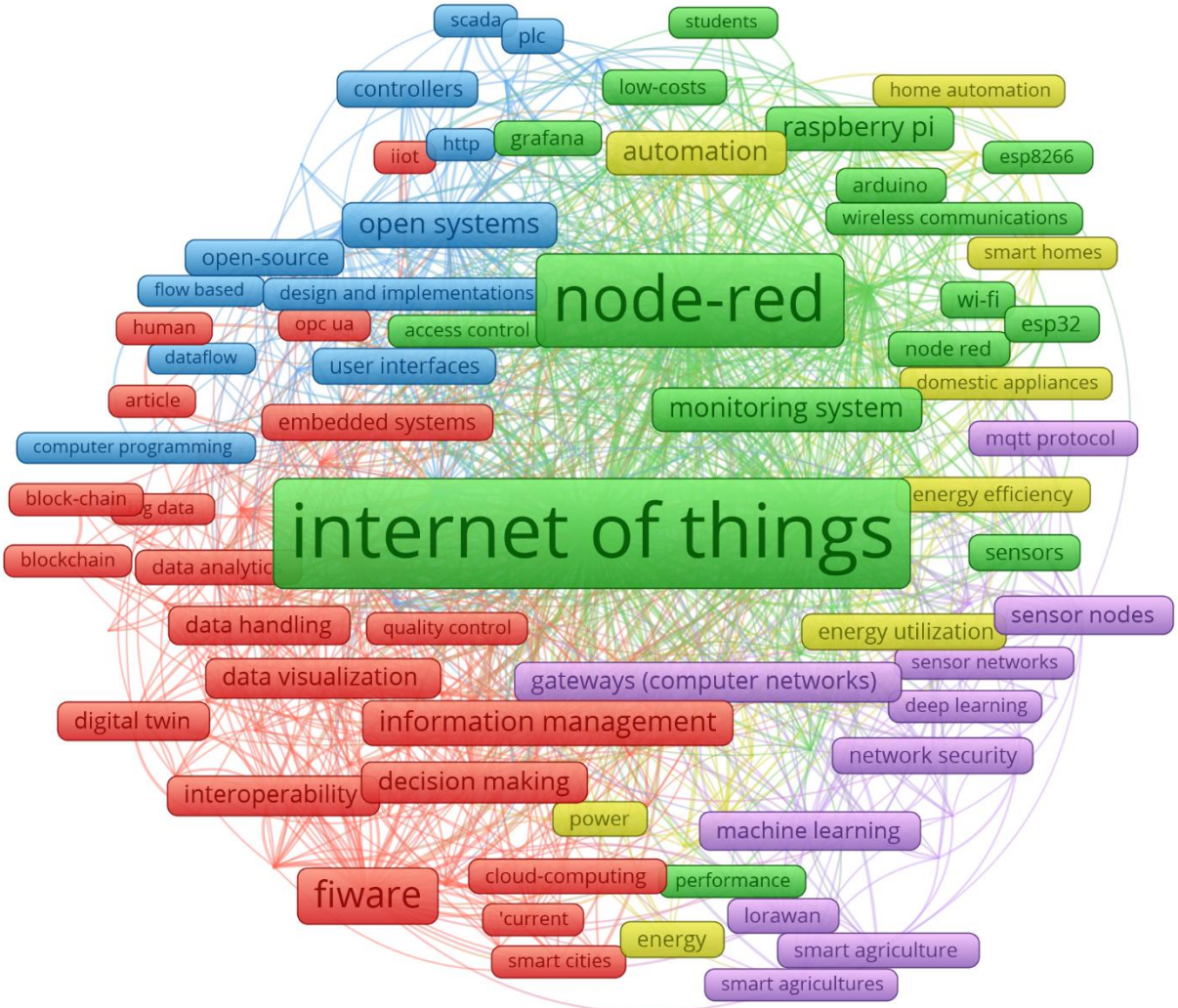


Fonte: Elaborado pelo autor (2025).

Além disso, o gráfico mostra termos de menor frequência, mas com registros desde 2020, como “*iot applications*”, “*agricultural robots*”, “*monitoring*” e “*network architecture*”, sugerindo que esses temas já vinham sendo explorados e agora podem estar em fase de estabilização ou especialização. No geral, a visualização confirma 2022 como o ponto de maior concentração de termos relevantes, marcando um pico de atividade na área de IoT. Os dados indicam tanto a consolidação de conceitos centrais quanto a emergência de novos focos, como segurança de rede e consumo energético, nos anos subsequentes.

Por último, no ambiente do VOSviewer, aplicados os mesmos critérios de filtragem de palavras-chave para construção da rede bibliométrica dos termos específicos, foram encontradas 6357 palavras-chave, dentre as quais 140 atingiram o limiar mínimo estipulado. Dessas, foram gerados 5 *clusters*, os quais, juntos, ilustram o panorama para a consulta (Figura 7). Os termos mais centrais e frequentes são "*internet of things*" e "*node-red*", indicando que esses são os principais eixos temáticos da rede.

Figura 7 – Rede de coocorrência de palavras-chave no contexto dos temas específicos.



Fonte: Elaborado pelo autor (2025).

O *cluster* verde, onde estão esses dois termos, representa a espinha dorsal da discussão, envolvendo conceitos como "*monitoring system*", "*raspberry pi*" e *automation*", com forte ligação à implementação prática de soluções IoT. O *cluster* vermelho agrupa temas como "*fiware*", "*data handling*", "*interoperability*" e "*information management*", sugerindo uma ênfase em infraestrutura, integração e tomada de

decisão baseada em dados. O *cluster* azul foca em aspectos técnicos, com termos como “*open systems”*, “*controllers”* e “*embedded systems”*, enquanto o amarelo se volta a aplicações domésticas e educacionais, com termos como “a*rduino”*, “*home automation”* e “*students”*. Por fim, o *cluster* roxo destaca tecnologias de suporte e segurança, como “*machine learning”*, “*sensor nodes*” e “*network security”*.

Esses agrupamentos refletem, assim como na rede dos temas gerais, a complexidade e a interdependência dos elementos que compõem o ecossistema IoT, com destaque para as aplicações nas quais as plataformas selecionadas para a pesquisa se encaixam.

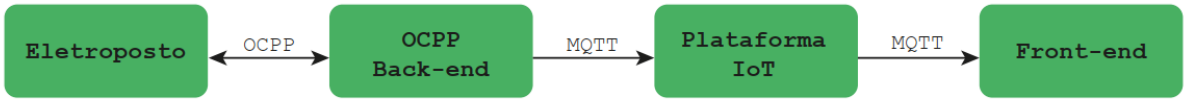
2.3. TRABALHOS RELACIONADOS

Para expandir a análise conduzida a termos práticos com as plataformas IoT estudadas, os tópicos subsequentes focarão na literatura do intervalo temporal de consulta, ilustrando como os agrupamentos refletem as mais recentes aplicações.

2.3.1. TRABALHOS RELACIONADOS À PLATAFORMA DOJOT

O estudo de Araújo *et al.,* (2022) destacou a plataforma Dojot como um *middleware* essencial para integração, coleta e análise de dados em sistemas de carregamento de veículos elétricos (eletropostos) (Figura 8). Atuando como um *hub* central, a Dojot recebeu dados em tempo real — como corrente, tensão, consumo de energia e status de carga — via protocolo MQTT, garantindo comunicação eficiente e leve, típica de soluções IoT. Além da coleta, a plataforma armazenou os dados em um banco de dados dedicado, permitindo o registro histórico de sessões de carregamento e facilitando a identificação de tendências ou anomalias no consumo energético.

Figura 8 – Arquitetura de coleta e análise de dados de eletropostos.



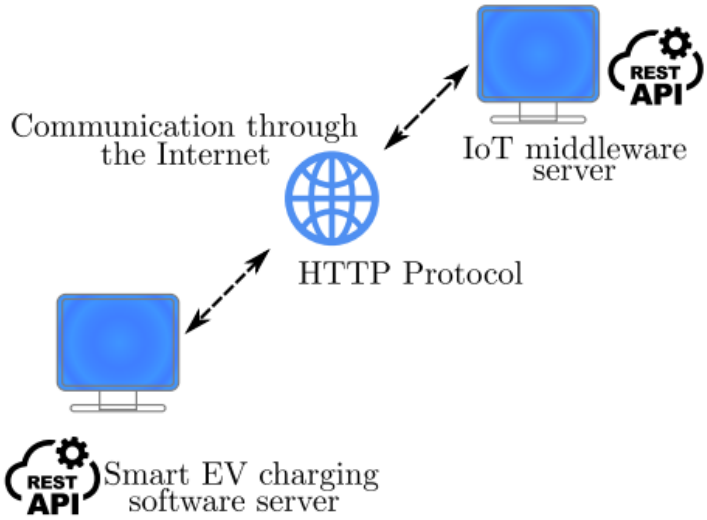
Fonte: Araújo *et al.,* (2022).

Um dos principais benefícios foi o monitoramento em tempo real, com *dashboards* que exibiam valores instantâneos (como potência e corrente), permitindo que operadores detectassem falhas ou interrupções nos eletropostos rapidamente,

sendo crucial para manutenção preventiva e operação segura. Integrada a uma arquitetura multicamadas — composta por eletropostos, *back-end* (via Open Charge Point Protocol – OCPP) e *front-end* —, a Dojot simplificou a comunicação entre dispositivos e aplicativos, eliminando a complexidade de conexões diretas e garantindo um fluxo de dados padronizado e sem atrasos.

O estudo de Terada *et al.,* (2022) reforçou a plataforma Dojot como um *middleware* IoT central para gerenciar dispositivos em sistemas de carregamento inteligente de veículos elétricos (VEs) e integração com fontes renováveis. Assim como no trabalho anterior, a Dojot atuou como *hub* de comunicação, coletando dados em tempo real de múltiplos dispositivos (eletropostos, sensores de energia) via internet, utilizando protocolos padronizados (MQTT e REST/HTTP) para garantir interoperabilidade e escalabilidade (Figura 9).

Figura 9 – Plataforma Dojot empregada como *middleware* IoT.



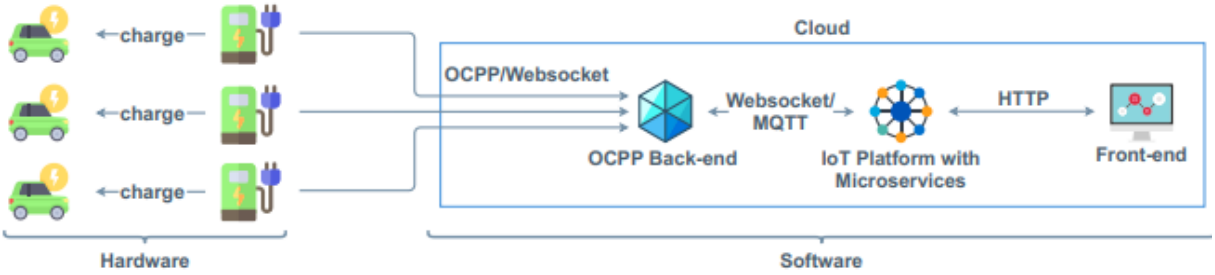
Fonte: Terada *et al.,* (2022).

Não obstante, testes de desempenho comprovaram sua eficiência: em redes estáveis, a Dojot gerenciou solicitações simultâneas em intervalos curtos, crucial para simular Ambientes Reais de Operadores (CPOs). Além disso, sua arquitetura RESTful facilitou trocas de dados entre servidores, enquanto sua infraestrutura escalável serviu como espinha dorsal experimental, permitindo testes reproduzíveis de algoritmos de carregamento inteligente em cenários com energia renovável.

Reforçando o nicho dos estudos apresentados, a pesquisa de Lobato *et al.,* (2022) reforça o padrão de aplicação da plataforma Dojot observado em pesquisas anteriores, demonstrando sua versatilidade como solução central para diversos

sistemas IoT. A plataforma manteve sua característica arquitetural principal: atuar como *hub* central entre dispositivos IoT e sistemas de análise. No caso dos eletropostos do projeto Sistema Inteligente Multimodal da Amazônia (SIMA), assim como nas aplicações anteriores, a Dojot utilizou o protocolo MQTT para coleta de dados e MongoDB para armazenamento, os quais são integrados a sua arquitetura de microsserviços (Figura 10).

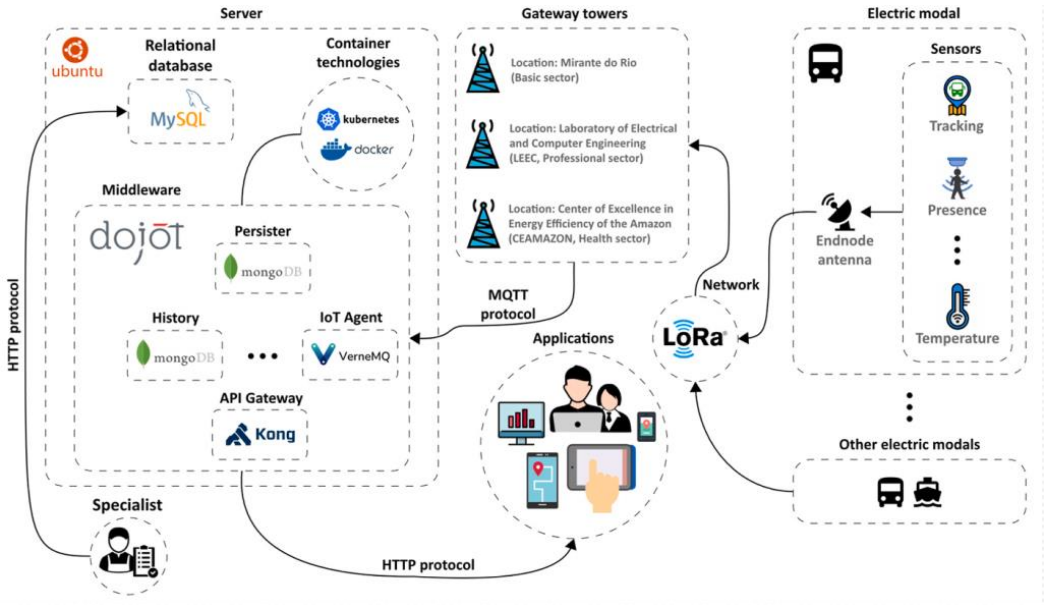
Figura 10 – Arquiteura do Sistema Interligado Multimodal da Amazônia (SIMA).



Fonte: Lobato *et al.,* (2022).

No estudo de Sá *et al.,* (2024) é proposta uma solução inteligente baseada em uma rede neural artificial (ANN) para otimizar rotas em um campus, integrando dados de IoT — armazenados e processados pela plataforma Dojot (que utiliza o MongoDB para gerir dados de geolocalização em tempo real transmitidos pela rede IoT) — e um banco de dados relacional MySQL com rotas multimodais pré-mapeadas (pedestres, ônibus, barcos) (Figura 11).

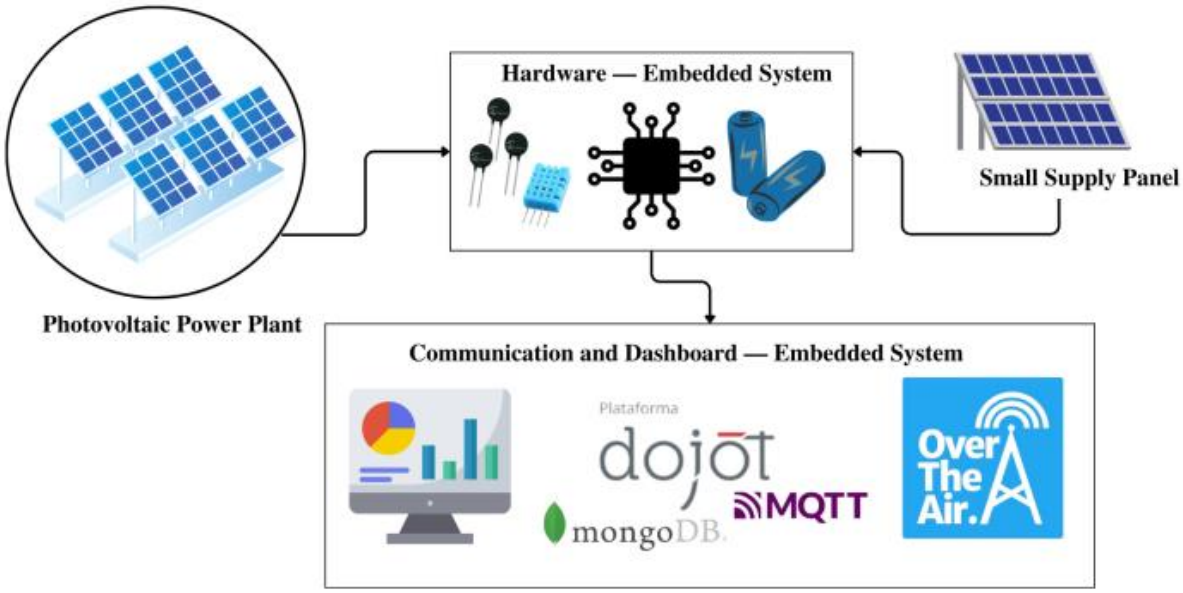
Figura 11 – Solução inteligente de otimização de rotas multimodais em um campus universitário com integração da plataforma Dojot.



Fonte: Sá *et al.,* (2024).

Por último, o estudo de Lisboa *et al.,* (2025) reforça a eficácia da plataforma Dojot como núcleo central para armazenamento, gerenciamento e monitoramento de dados IoT, complementando as aplicações já destacadas nos sistemas explorados anteriormente. Assim como nos cenários de eletromobilidade, onde a Dojot gerenciava dados de geolocalização em tempo real via MQTT para otimizar rotas multimodais, nessa aplicação ela atua como *hub* integrado, recebendo informações de sensores (temperatura, umidade) via ESP32 e assegurando armazenamento seguro em MongoDB e visualização via *dashboards* (Figura 12).

Figura 12 – Arquitetura de comunicação com integração da plataforma Dojot.



Fonte: Lisboa *et al.,* (2025).

Sua capacidade de preservar dados por longos períodos (até 100 anos) e operar em redes limitadas — graças ao protocolo MQTT — reforça sua confiabilidade tanto para monitoramento remoto de usinas solares quanto para mobilidade urbana.

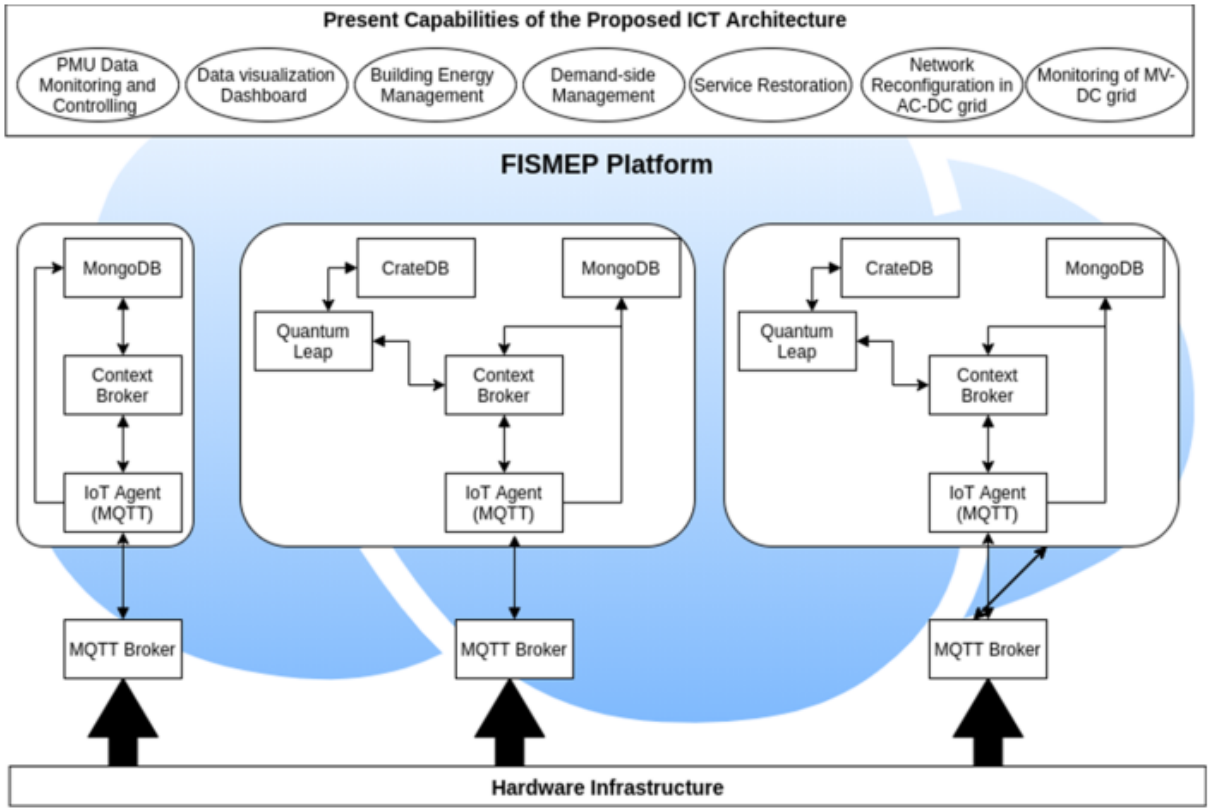
Sendo assim, as funcionalidades convergentes visualizadas nos trabalhos examinados destacam a Dojot como uma solução escalável, apta a suportar desde aplicações em *campus* inteligentes até projetos complexos de cidades sustentáveis, unindo eficiência operacional, segurança de dados e adaptabilidade a diferentes cenários de IoT.

2.3.2. TRABALHOS RELACIONADOS À PLATAFORMA FIWARE

No estudo de Haghgoo *et al.,* (2021), o FIWARE foi implementado como plataforma central para sistemas inteligentes de energia, destacando-se em três aspectos principais, onde o Orion Context Broker atuou como núcleo integrador para dados heterogêneos de redes elétricas e edifícios inteligentes, enquanto as APIs abertas permitiram a customização por diversos atores do setor energético (Figura 13).

A arquitetura em nuvem demonstrou notável escalabilidade, possibilitando a expansão geográfica e funcional do sistema conforme a inclusão de novos dispositivos e serviços. Além disso, os GEs do FIWARE garantiram a padronização necessária para interoperabilidade entre diferentes componentes do sistema energético.

Figura 13 – Arquitetura de TIC para uma solução integrada, escalável e flexível para futuros sistemas de energia inteligentes.



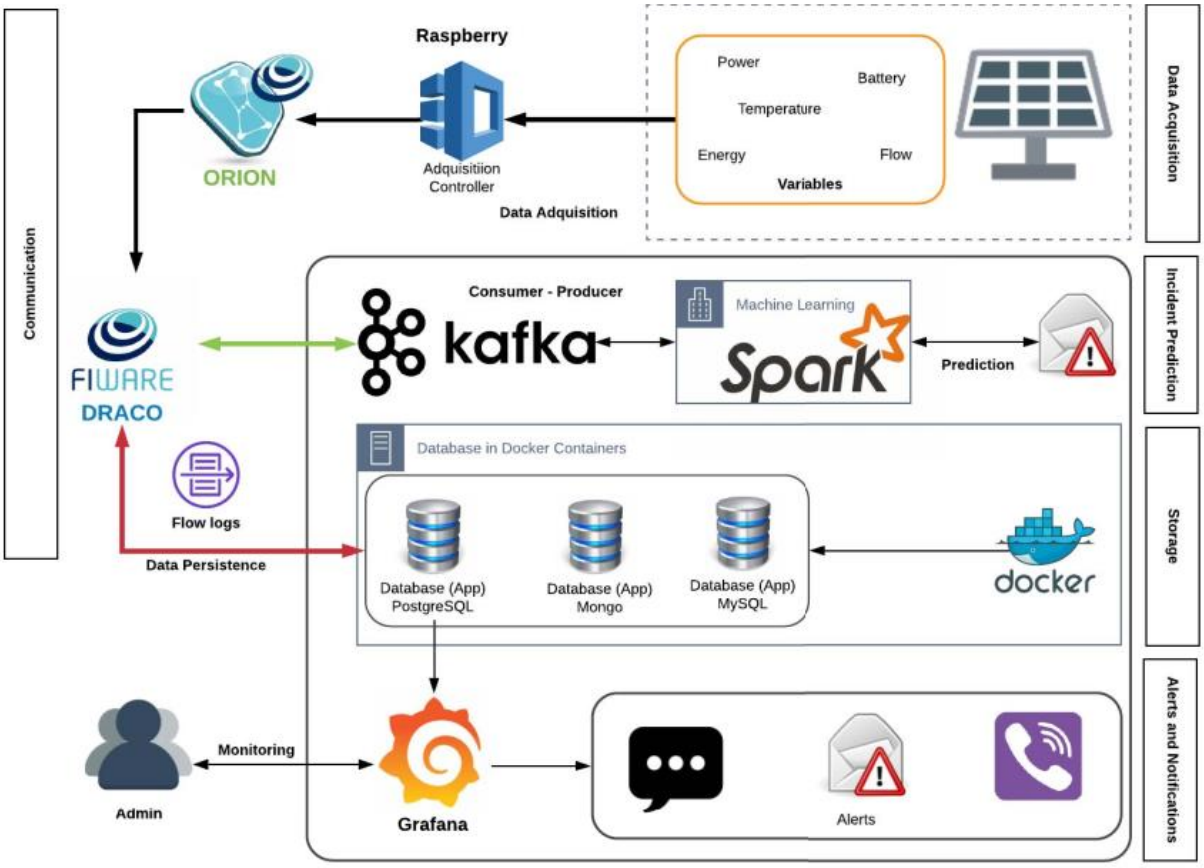
Fonte: Haghgoo *et al.,* (2021).

De forma semelhante, o trabalho de Velasquez; Tobar-Andrade; Cedeno-Campoverde, (2021) apresentou uma aplicação complementar do FIWARE, focada especificamente em sistemas de energia renovável. Aqui, a plataforma mostrou sua

versatilidade no monitoramento em tempo real de variáveis críticas como potência, corrente e status de baterias (Figura 14).

O modelo *publish-subscribe* otimizou a comunicação entre dispositivos, enquanto o FIWARE-Draco gerenciou eficientemente a distribuição de dados pelo sistema. Notavelmente, este estudo também destacou o potencial do FIWARE para integração com ferramentas de análise preditiva, como Apache Spark e Kafka, preparando o sistema para futuras funcionalidades avançadas.

Figura 14 – Proposta de arquitetura para um sistema de monitoramento e processamento de dados.



Fonte: Velasquez; Tobar-Andrade; Cedeno-Campoverde (2021).

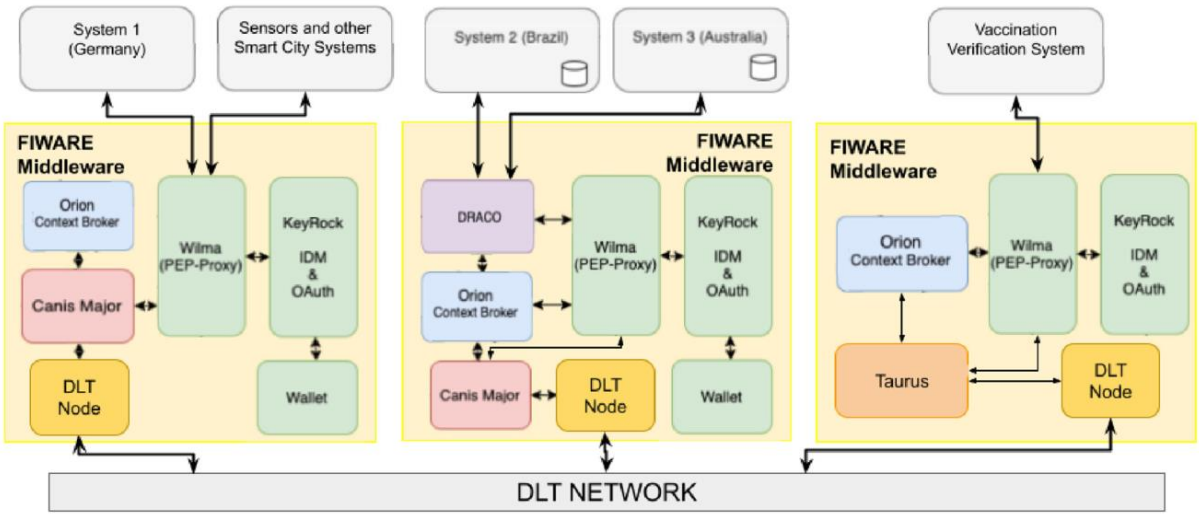
Ambos os trabalhos convergiram ao demonstrar a capacidade adaptativa do FIWARE, onde o Orion Context Broker emergiu como componente-chave, garantindo a integração eficiente de dados diversos. em diferentes cenários energéticos, desde redes complexas até sistemas isolados de energia renovável.

Na aplicação de Loss *et al.,* (2023), o FIWARE foi implementado como a espinha dorsal tecnológica para integração de sistemas em um ambiente de cidade inteligente. A plataforma demonstrou sua eficácia como *middleware* ao conectar

diversos componentes através de seus GEs, com destaque para o Orion Context Broker, que gerenciou o fluxo de informações em tempo real entre os sistemas conectados.

A inovação central desta pesquisa foi a integração do FIWARE com tecnologia Blockchain, superando uma limitação importante dos sistemas tradicionais ao garantir imutabilidade e rastreabilidade dos dados. Essa combinação foi viabilizada através do desenvolvimento de dois novos GEs especializados: o Canis Major, responsável por converter dados no formato NGSI em transações Blockchain, e o Taurus, que realiza o processo inverso ao capturar eventos da cadeia de blocos e disponibilizá-los no ecossistema FIWARE (Figura 15).

Figura 15 – Arquitetura proposta para a integração dos componentes FIWARE.



Fonte: Loss *et al.,* (2023).

Quando comparado aos trabalhos anteriores que utilizaram o FIWARE em sistemas de energia inteligente, este estudo compartilha a mesma abordagem central de utilizar a plataforma como integrador tecnológico, mas avança significativamente na dimensão de segurança dos dados. Enquanto os projetos de energia focaram no monitoramento em tempo real e análise preditiva, esta implementação resolveu um desafio crítico de integridade de dados através da imutabilidade proporcionada pelo Blockchain.

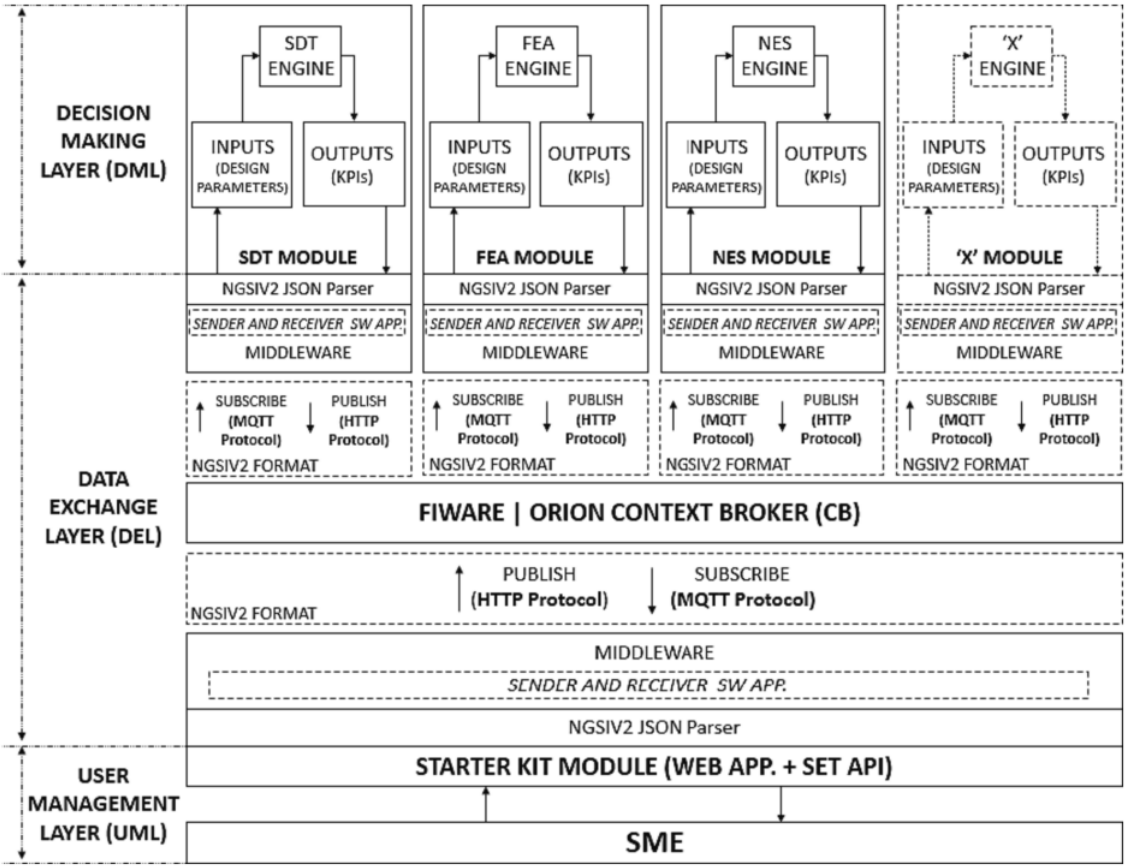
Notavelmente, o Orion Context Broker manteve seu papel fundamental como componente de integração, assim como ocorreu nos sistemas de monitoramento energético. A solução também preservou a característica modular do FIWARE,

evidenciada pela criação dos novos GEs especializados, seguindo o mesmo padrão de extensibilidade observado nos trabalhos anteriores que integraram o FIWARE com ferramentas como Apache Spark e Kafka.

Na aplicação de Cimino *et al.,* (2023), o FIWARE foi implementado como elemento central para integração de sistemas em um ambiente de produção industrial, demonstrando sua versatilidade além dos contextos de cidades inteligentes e energia. A plataforma atuou como camada de interoperabilidade crítica entre módulos de simulação, análise e otimização, com o Orion Context Broker gerenciando o fluxo de dados entre os diversos componentes do sistema (Figura 16).

O formato NGSIv2 JSON serviu como linguagem comum, permitindo que diferentes aplicativos, cada um com seus formatos nativos, compartilhassem informações sem conflitos. Esta aplicação destacou particularmente a capacidade do FIWARE em simplificar a complexidade técnica para Pequenas e Médias Empresas (PMEs), oferecendo uma ponte acessível entre usuários finais não especializados e sistemas avançados de otimização industrial.

Figura 16 – Infraestrutura de plataforma multiusuário e multifuncional.

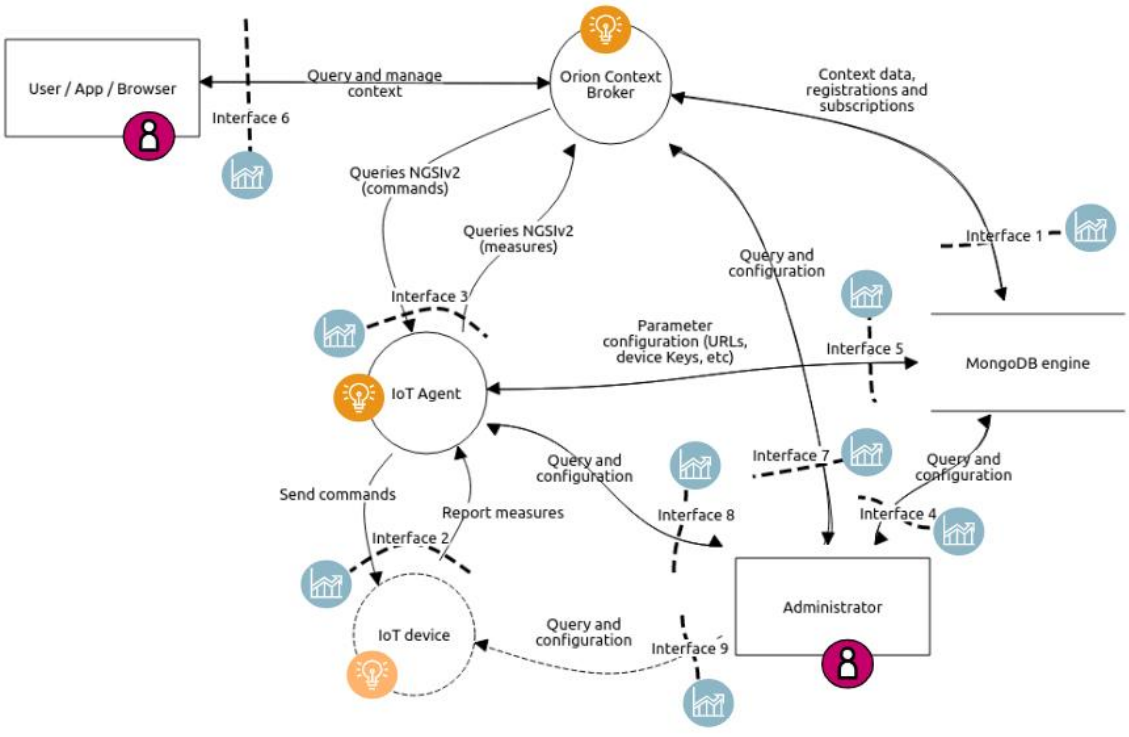


Fonte: Cimino *et al.,* (2023).

Comparando com os trabalhos anteriores, esta implementação compartilha a arquitetura fundamental observada nas aplicações de energia e cidades inteligentes, onde o FIWARE consistentemente atuou como *middleware* de integração. O Orion Context Broker manteve seu papel central como gerenciador de fluxo de dados, assim como ocorreu nos sistemas de monitoramento energético e na plataforma de vacinação com Blockchain. A inovação neste caso foi a adaptação do FIWARE para resolver desafios específicos da indústria, particularmente na harmonização de dados entre sistemas de produção heterogêneos. Enquanto os projetos anteriores focaram em escalabilidade (energia) e segurança (Blockchain), esta aplicação destacou a capacidade do FIWARE em democratizar o acesso a tecnologias avançadas, tornando sistemas complexos acessíveis a PMEs com limitada infraestrutura de TI.

No trabalho de Perata; Betarte, (2023), o FIWARE assumiu um papel único como objeto de estudo e plataforma experimental para avaliação de segurança. Diferentemente das aplicações anteriores que utilizaram o FIWARE como solução de integração, esta pesquisa empregou a plataforma como base para construir um ambiente referencial completo, incorporando seus principais componentes com o propósito específico de analisar vulnerabilidades (Figura 17).

Figura 17 – Diagrama do fluxo de dados da plataforma de referência FIWARE.



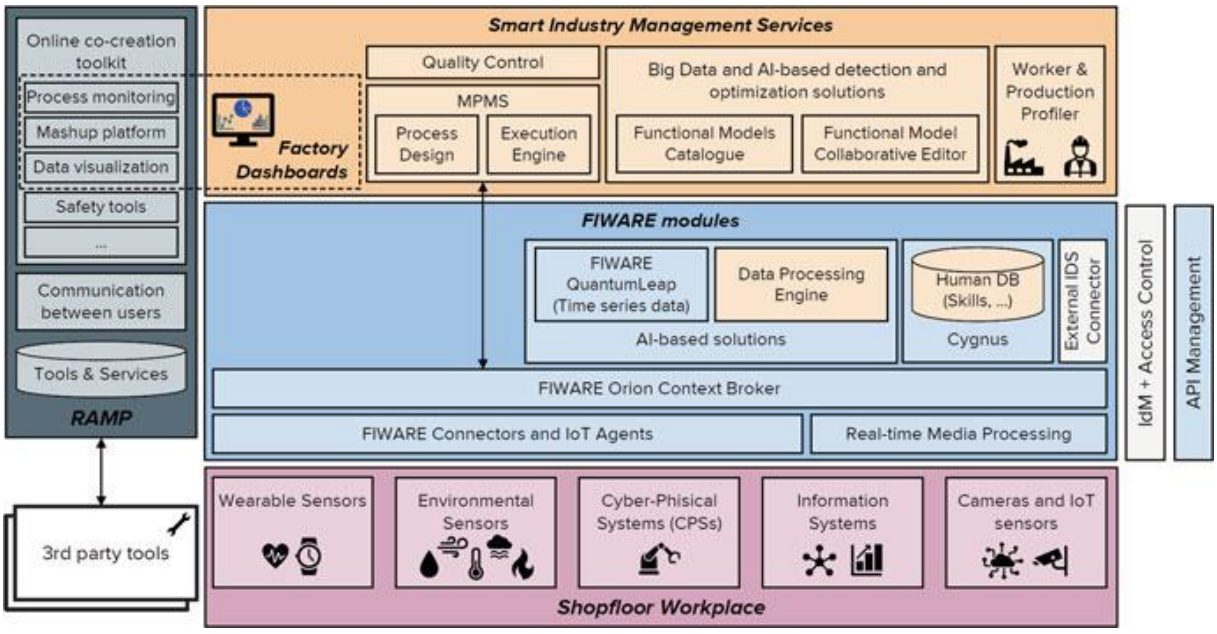
Fonte: Perata; Betarte (2023).

Os pesquisadores criaram um laboratório de segurança que replicava implementações reais do FIWARE, permitindo a execução controlada de testes ofensivos, incluindo simulações de ataques como adulteração de imagens dos componentes e exploração de canais de comunicação inseguros. A metodologia adotada combinou técnicas estruturadas de modelagem de ameaças (STRIDE e OWASP) com testes práticos em uma implantação real da plataforma por uma entidade pública, oferecendo assim uma avaliação abrangente que vai desde a teoria até a verificação em cenários do mundo real.

Já na aplicação de Cutrona *et al.,* (2024), o FIWARE foi utilizado como a espinha dorsal tecnológica da plataforma KITT4SME, desenvolvida para democratizar o acesso a ferramentas de IA para PMEs de manufatura. Atuando como *middleware* central, o FIWARE permitiu a integração perfeita entre equipamentos industriais heterogêneos e módulos de análise de IA, superando desafios de interoperabilidade comuns em ambientes fabris (Figura 18).

O Orion Context Broker, componente essencial do FIWARE, gerenciou fluxos de dados em tempo real, facilitando as etapas de "sensoriamento" e "intervenção" do workflow da plataforma. Além disso, o modelo de dados NGSI do FIWARE padronizou a comunicação entre dispositivos de diferentes fabricantes, eliminando a necessidade de adaptadores customizados e reduzindo custos para as PMEs.

Figura 18 – Arquitetura do KITT4SME alimentada pelo FIWARE.

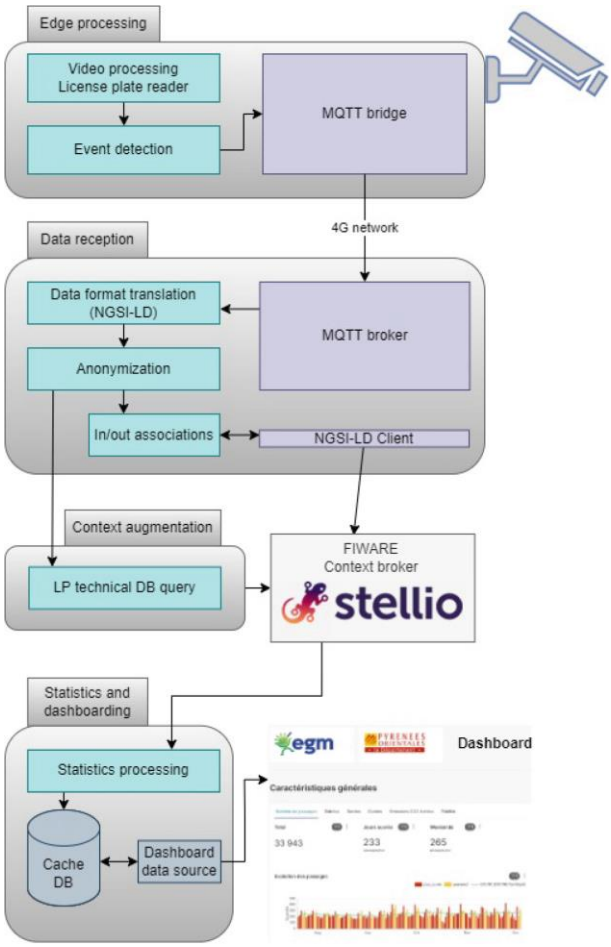


Fonte: Cutrona *et al.,* (2024).

Esta aplicação destacou-se por adaptar o FIWARE a um contexto específico de manufatura, indo além dos usos tradicionais em cidades inteligentes. A plataforma aproveitou os GEs do FIWARE para criar um *marketplace* modular de componentes de IA, onde soluções podiam ser compostas sob demanda conforme as necessidades individuais de cada PME. Diferente de implantações empresariais convencionais, o projeto focou em simplificar a complexidade técnica, tornando recursos avançados de IA acessíveis através de um modelo "*as-a-service*". A arquitetura permitiu desde o monitoramento em tempo real de equipamentos até a integração de algoritmos preditivos, sempre mantendo a escalabilidade e segurança características do FIWARE.

Por último, a aplicação de Orazi *et al.,* (2025) apresenta uma aplicação inovadora do FIWARE no domínio da mobilidade sustentável, especificamente no monitoramento de estacionamentos dedicados à carona solidária (Figura 19).

Figura 19 – Arquitetura global da nossa solução para monitoramento de estacionamentos compartilhados.



Fonte: Orazi *et al.,* (2025)

A plataforma foi utilizada como o núcleo de um sistema híbrido que combina processamento de vídeo inteligente na borda (Edge AI) com armazenamento e análise na nuvem. Enquanto algoritmos de visão computacional processavam em tempo real as entradas e saídas de veículos, o FIWARE, através de seu Context Broker e utilizando o padrão NGSI-LD da ETSI, estruturava e armazenava os dados anonimizados, permitindo consultas complexas e geração de estatísticas sobre padrões de uso, fidelidade dos usuários e cálculo do impacto ambiental.

Sendo assim, através da literatura consultada, o FIWARE se estabelece como plataforma de interoperabilidade por excelência em IoT, com três pilares de capacidade central, através do Orion + NGSI como base para integração; modularidade adaptativa, garantida pela customização de GEs para necessidades específicas e; resiliência operacional, por meio da validação em implantações de longo prazo.

Sua principal força reside na habilidade de unir domínios díspares (indústria, energia, mobilidade) sob uma mesma abordagem técnica, reduzindo complexidade e acelerando inovação – especialmente para organizações com recursos limitados.

2.3.3. TRABALHOS RELACIONADOS À PLATAFORMA NODE-RED

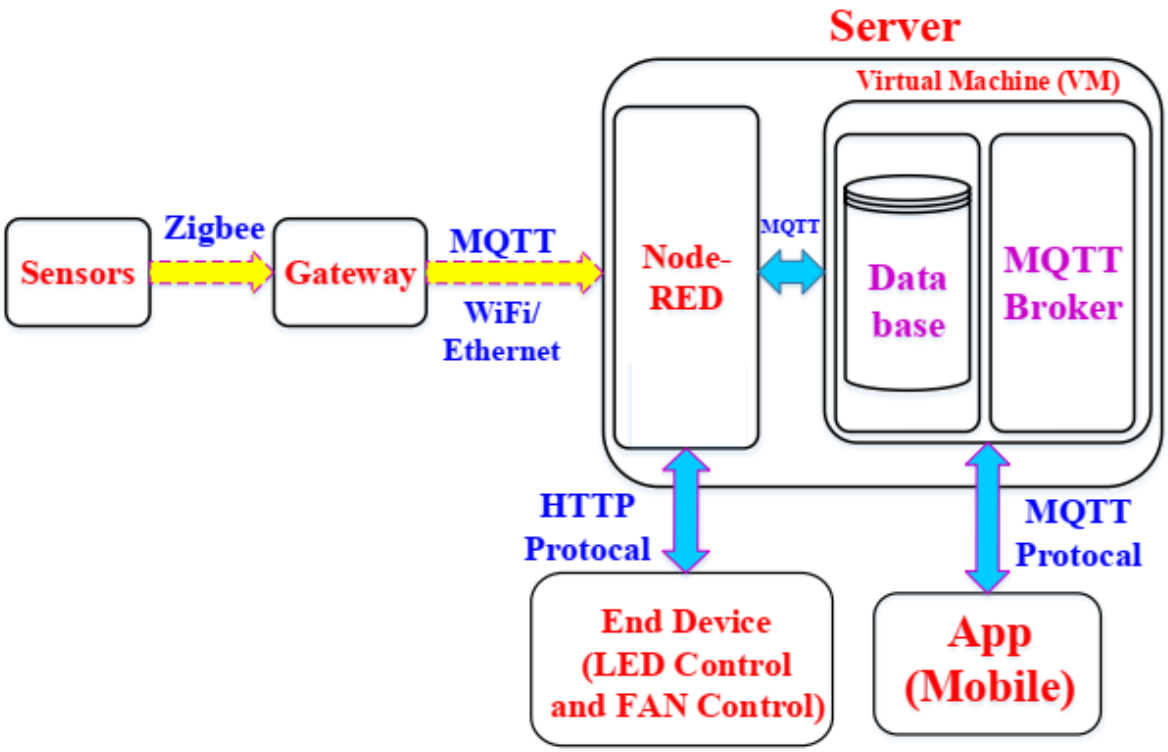
Na aplicação de Sung *et al.,* (2025), o Node-RED foi empregado como plataforma central para orquestrar fluxos de dados e ações em um sistema de monitoramento ambiental (Figura 20). Sua atuação iniciou como *subscriber* MQTT, recebendo dados de sensores (temperatura, umidade etc.) transmitidos via ZigBee através de um *gateway*.

Após a captura, o Node-RED armazenou automaticamente essas informações em um banco de dados MongoDB, garantindo a persistência segura para análises posteriores. Além do gerenciamento de dados, o Node-RED executou processamento em tempo real: validou a qualidade das medições, aplicou normalizações e comparou valores contra limites predefinidos para acionar alertas ou ações corretivas.

Como *hub* de comunicação, o Node-RED integrou componentes heterogêneos (sensores, broker MQTT, MongoDB e app móvel) em um único fluxo visual. Sua interface gráfica baseada em nós permitindo a visualização intuitiva do caminho dos dados (da coleta ao controle); depuração simplificada de falhas em tempo real e;

adaptabilidade rápida para adicionar novos sensores ou regras de negócio. Essa flexibilidade reduziu a complexidade de desenvolvimento e garantiu escalabilidade para futuras expansões do sistema, mantendo-o robusto e alinhado com demandas evolutivas.

Figura 20 – Arquitetura do sistema de monitoramento doméstico com inteligência artificial proposto.



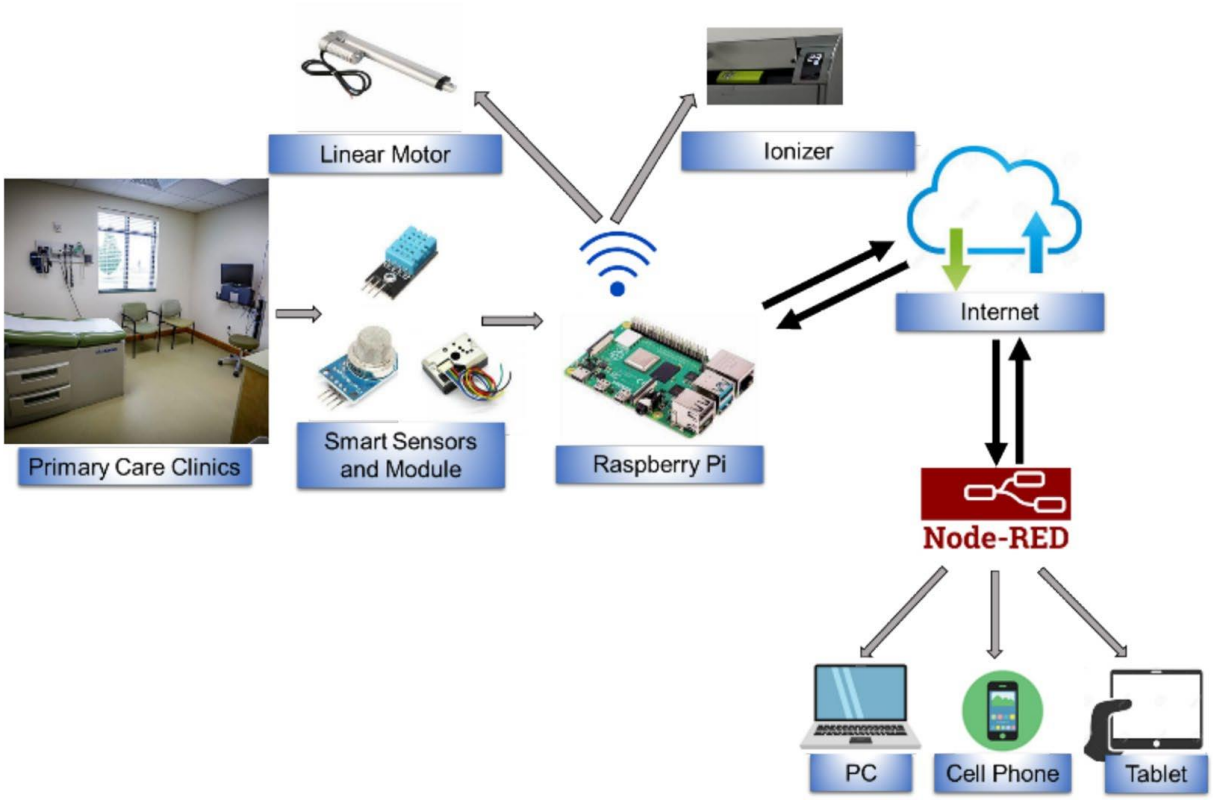
Fonte: Sung *et al.,* (2025).

O estudo de Cabuk, (2025) implementou o Node-RED como núcleo tecnológico de uma arquitetura proposta para gestão de dados de qualidade do ar em tempo real e clínicas de atendimento primário. Integrado a um Raspberry Pi (unidade de *hardware* central), a plataforma processou medições de sensores locais (material particulado, CO, NO_2) e dados externos de fontes *web*, consolidando múltiplos fluxos de informação (Figura 21).

Através de nós customizados, o Node-RED executou três funções críticas: i) processamento inteligente, separando dados em campos específicos, aplicando algoritmos para detectar violações de limites de qualidade do ar e gerando comandos para atuadores (como ionizadores e sistemas de abertura de janelas); ii) visualização acessível, criando um painel *web* intuitivo com gráficos, medidores e LEDs que permitia monitoramento em qualquer dispositivo via navegador e; iii) controle proativo,

estabelecendo comunicação bidirecional para acionar respostas automáticas quando os parâmetros ultrapassavam níveis seguros.

Figura 21 – Sistema de regulamentação da qualidade do ar em clínicas de cuidados primários projetado com IoT usando o Node-RED.



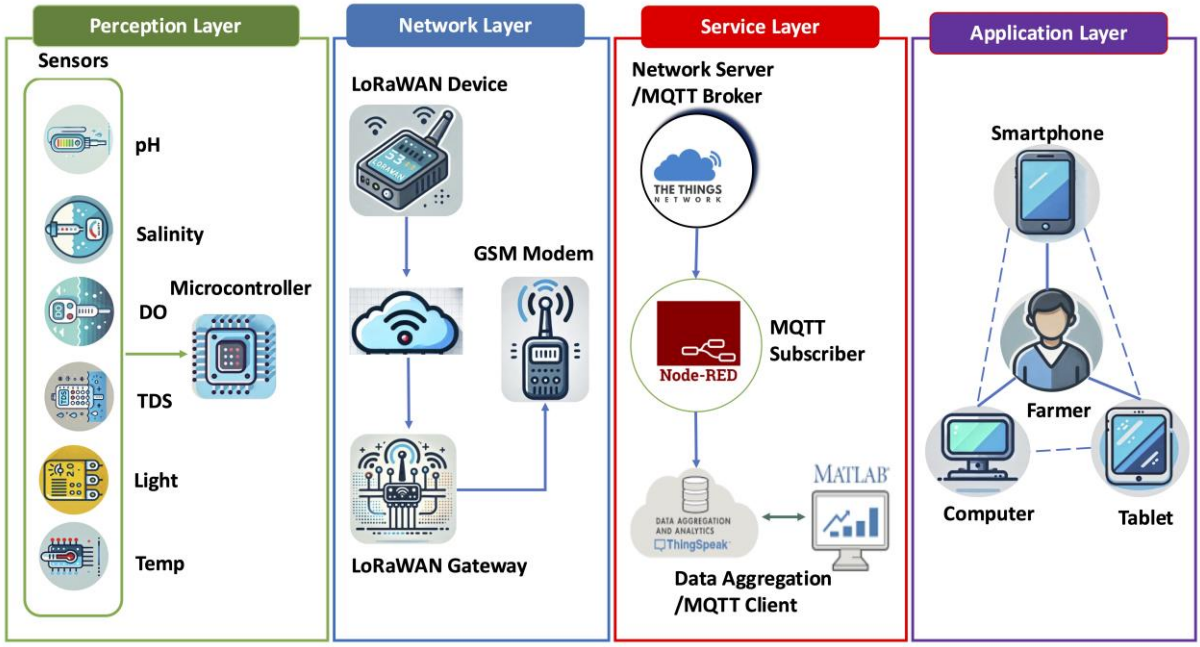
Fonte: Cabuk (2025).

De forma similar aos estudos anteriores, a implementação de Misbahuddin *et al.,* (2025) apresentou o Node-RED como núcleo operacional em uma arquitetura IoT para monitoramento ambiental em aquicultura. A plataforma atuou como assinante MQTT, recebendo dados em tempo real da The Things Network (TTN) e processando-os imediatamente para detectar anomalias – como valores estáticos prolongados ou variações abruptas em parâmetros de qualidade da água (Figura 22). Essa capacidade de análise contínua permitiu respostas rápidas a potenciais riscos no cultivo de *Sargassum sp*.

Além do processamento, o Node-RED demonstrou adaptabilidade dinâmica ao controlar os intervalos de transmissão dos sensores conforme as condições da rede. Através de nós especializados, a plataforma ajustou automaticamente a frequência de envio de dados, equilibrando eficiência energética e qualidade da comunicação – um recurso crítico para operações remotas com recursos limitados.

Após o refinamento dos dados (incluindo aplicação de filtros como Kalman para maior precisão), o Node-RED integrou-se à plataforma ThingSpeak, encaminhando informações consolidadas para visualização em painéis interativos. Essa ponte inteligente entre coleta e exibição garantiu que operadores acessassem métricas confiáveis sobre oxigênio dissolvido, temperatura e outros indicadores-chave.

Figura 22 – Arquitetura da estrutura de IoT baseada em LoRaWAN.



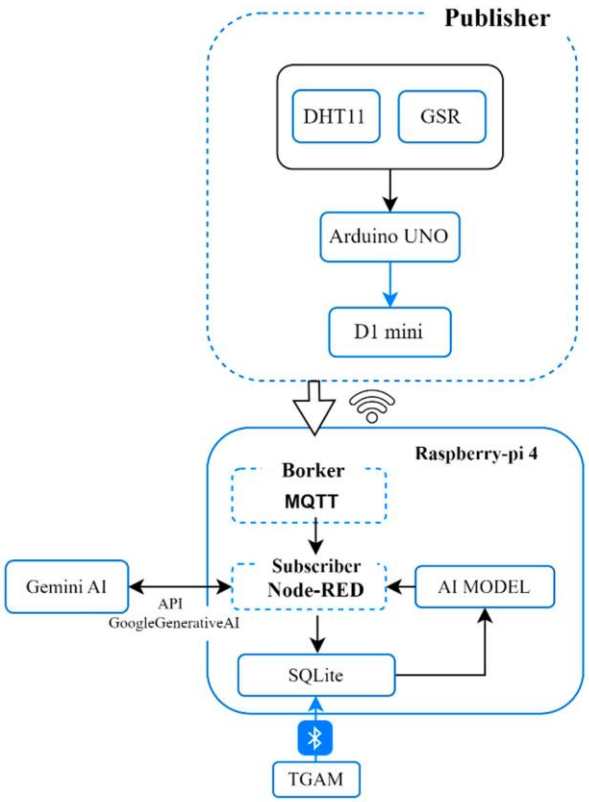
Fonte: Misbahuddin *et al.,* (2025).

A plataforma também simplificou o gerenciamento operacional: pesquisadores ajustavam limites de alerta e parâmetros de rede diretamente no Node-RED, sem reconfigurar dispositivos físicos. Essa separação entre *hardware* e lógica reduziu custos de manutenção e aumentou a resiliência do sistema em ambientes dinâmicos. Ao orquestrar coleta, processamento, otimização de rede e visualização, o Node-RED viabilizou um sistema auto adaptável que mantém a integridade dos dados mesmo sob variações ambientais. Sua abordagem *low-code* permitiu focar na sustentabilidade da aquicultura, transformando dados brutos em insights acionáveis para práticas mais eficientes.

Por último, a aplicação de Tseng *et al.,* (2025) implementou o Node-RED como núcleo operacional e de interface em uma plataforma educacional inovadora. Atuando como assinante MQTT, a ferramenta recebeu em tempo real dados de sensores fisiológicos (como *Galvanic Skin Response* – GSR – e *ThinkGear ASIC Module* –

TGAM) capturados dos alunos, permitindo monitoramento contínuo de indicadores como estresse e níveis de atenção, formando a base para uma resposta educacional dinâmica e personalizada (Figura 23).

Figura 23 – Diagrama da arquitetura da plataforma educacional proposta.



Fonte: Tseng *et al.,* (2025).

A plataforma destacou-se pela interface gráfica unificada, onde professores e alunos acessavam um painel intuitivo via navegador *web*. Para educadores, o Node-RED estruturou módulos de criação de conteúdo, monitoramento em tempo real e análise histórica. Além da visualização, o Node-RED gerenciava o armazenamento de dados em SQLite, preservando histórico completo para análises pedagógicas. Essa capacidade permitiu correlacionar dados fisiológicos com desempenho acadêmico, identificando padrões como variações de atenção durante diferentes tipos de exercícios.

Sendo assim, esta aplicação transformou dados brutos de sensores em insights pedagógicos acionáveis, demonstrando como ferramentas IoT de baixo código podem revolucionar metodologias educacionais ao fornecer *feedback* biométrico em tempo real e histórico analítico para personalização do ensino.

A análise dos casos evidencia que o Node-RED consolida-se como plataforma pivotal para soluções IoT, destacando-se por três atributos essenciais: integração simplificada, operação em tempo real e democratização tecnológica. Essa tríade – interoperabilidade técnica, resposta ágil e acessibilidade – viabiliza aplicações economicamente sustentáveis em setores diversos (educação, aquicultura, saúde, gestão urbana), transformando dados brutos em insights acionáveis enquanto reduz custos de implantação e manutenção. O Node-RED emerge, assim, não apenas como ferramenta de prototipagem, mas como solução operacional madura para IoT de impacto real.

2.4. CONSIDERAÇÕES FINAIS DO CAPÍTULO

Com base nas análises bibliométricas e na revisão de literatura realizadas sobre IoT e *middlewares*, incluindo as plataformas Dojot, FIWARE e Node-RED, observa-se um amplo potencial de aplicação em diferentes contextos do ecossistema IoT. Dessa forma, as três plataformas consolidam-se como pilares complementares, cada uma com propósitos distintos e sinérgicos:

- **Dojot:** destaca-se como solução verticalizada para ambientes que demandam escalabilidade industrial (Kubernetes), gestão centralizada de dispositivos e suporte nativo a protocolos pesados (ex: OCPP), sendo ideal para infraestruturas críticas como mobilidade urbana e redes de energia, onde a robustez operacional é prioritária;

- **FIWARE:** atua como *framework* de interoperabilidade horizontal, com seu Orion Context Broker e padrão NGSI/NGSI-LD criando uma "linguagem comum" para integração entre sistemas heterogêneos em cidades inteligentes, saúde pública e energia, reforçado por GEs que estendem suas capacidades (ex: Blockchain);

- **Node-RED:** emerge como ferramenta de democratização tecnológica, com sua abordagem *low-code* e fluxos visuais simplificando a prototipagem, o controle de borda e a automação em cenários de nicho (educação, aquicultura, monitoramento ambiental, inteligência artificial), onde agilidade e custo acessível são essenciais.

Enquanto Dojot e FIWARE são arquiteturas completas para ecossistemas complexos, o Node-RED é o "canivete suíço" para soluções ágeis. Juntas, essas plataformas cobrem todo o espectro IoT: da padronização global (FIWARE) à implantação massiva (Dojot) e à customização local (Node-RED), formando um tripé indispensável para a transformação digital em escala.

3. TECNOLOGIAS UTILIZADAS

No presente capítulo será apresentada uma visão geral sobre as técnicas de virtualização, a fim de garantir o entendimento necessário dos *softwares* Proxmox VE e Docker, utilizados para a efetiva implantação da arquitetura estabelecida. Em seguida, serão abordados os *middlewares* IoT Dojot, FIWARE, e Node-RED, plataformas sobre as quais foram determinados os principais objetivos da dissertação. Por último, serão introduzidos tópicos teóricos elementares necessários ao entendimento sobre observabilidade e monitoramento, as ferramentas utilizadas e o sistema implantado para mensurar o desempenho de cada arquitetura de *middleware* IoT.

3.1. VIRTUALIZAÇÃO

A virtualização consiste em uma tecnologia que possibilita a criação de ambientes virtuais de recursos computacionais, dentre os quais estão servidores, sistemas operacionais, armazenamento e redes, desvinculando-os do *hardware* físico subjacente (Jain, 2023; Patrão, 2024). Essa abordagem é implementada por meio de um *software* especializado chamado hipervisor (ou *virtual machine monitor*), que gerencia e isola múltiplas VM sobre uma mesma infraestrutura física (Huawei Tecnologies Co., 2023). Cada VM se comporta como um sistema independente, com seu próprio sistema operacional e aplicações, enquanto compartilha recursos do *host*, como núcleos de CPU, memória e armazenamento em disco (Portnoy, 2023).

Dentro desse contexto, existem diferentes modalidades de virtualização, cada uma adaptada a necessidades específicas. Na virtualização de servidores, por exemplo, um único servidor físico pode hospedar várias VMs com sistemas operacionais distintos, aumentando a utilização de recursos e reduzindo custos com *hardware* (Krems; Reich; Stark, 2017; Vazquez, 2024a).

Por outro lado, na virtualização de sistema operacional, utiliza-se o conceito de contêineres (como Docker), que criam ambientes isolados dentro do mesmo sistema operacional, com menor *overhead* do que as VMs tradicionais (Babu *et al.,* 2014; Vazquez, 2024b). Além disso, há virtualização de armazenamento, que agrega diversos dispositivos físicos em um *pool* lógico único, e virtualização de rede, que

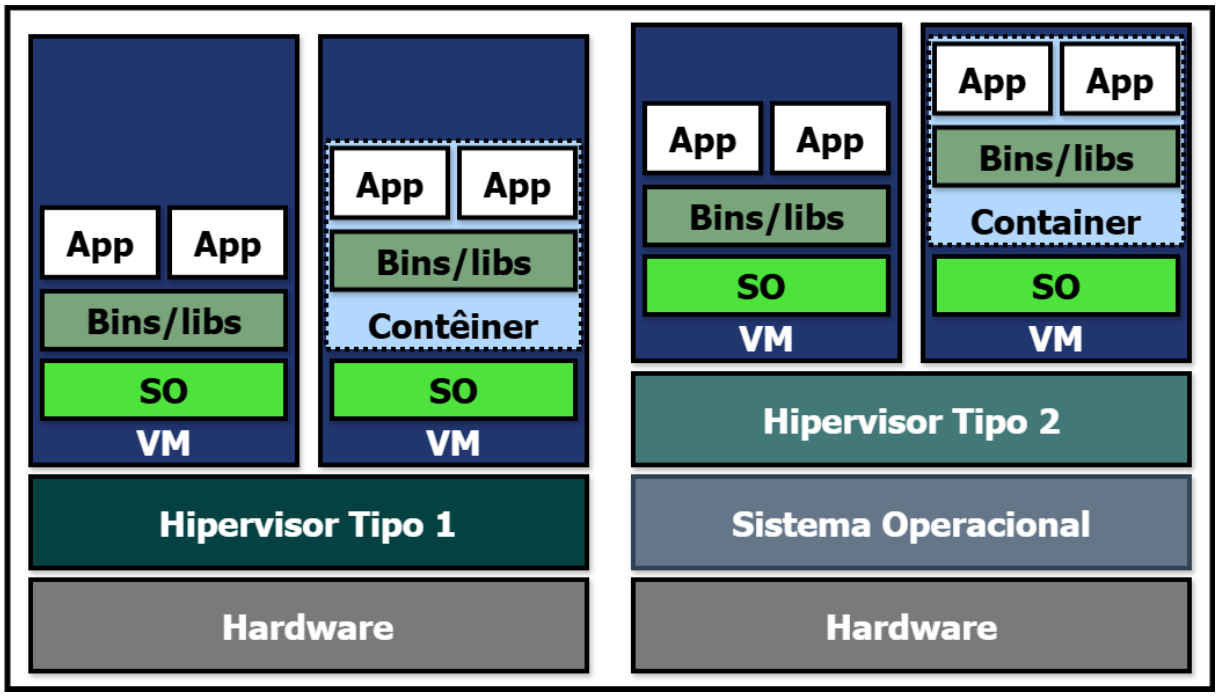
separa serviços de rede do *hardware*, possibilitando a criação de redes definidas por *software* (Süß; Süß, 2024).

O principal componente da virtualização é o hipervisor, que pode ser:

- Tipo 1 ou *native* (*bare metal*): instalado diretamente sobre o *hardware*, a exemplo de VMware ESXi e Microsoft Hyper-V;
- Tipo 2 ou *hosted*: executado sobre um sistema operacional *host*, como VirtualBox e VMware Workstation.

O hipervisor é responsável por alocar e gerenciar os recursos do *host* entre as VMs, garantindo isolamento e segurança. Cada máquina virtual contém sua própria cópia de um sistema operacional e aplicações, funcionando de forma totalmente independente, como se estivesse em *hardware* dedicado (Olzak *et al.,* 2010; Portnoy, 2023). A Figura 24 ilustra os tipos de virtualização descritos.

Figura 24 – Arquitetura dos tipos de virtualização/hipervisores.



Fonte: Elaborado pelo autor (2025).

Por conseguinte, a virtualização, ao desacoplar *software* e *hardware*, oferece soluções versáteis para desafios tecnológicos contemporâneos recorrentes. Na infraestrutura corporativa, por exemplo, a consolidação de servidores permite que empresas executem múltiplas VMs em um único *host* físico, otimizando custos com *hardware* e energia, uma prática essencial em *data centers* e ambientes que

demandam alta disponibilidade, aprimorando a eficiência energética através do conceito de computação verde, que se refere à prática de projetar, fabricar, utilizar e descartar equipamentos e sistemas computacionais de forma ambientalmente sustentável. (Hage Hassan *et al.,* 2022; Liang Chen; Longchuan Yan; Yifan Mao, 2014). Essa mesma lógica sustenta a computação em nuvem, na qual provedores como AWS e Azure utilizam hipervisores para disponibilizar infraestruturas escaláveis, permitindo que usuários provisionem recursos sob demanda, adaptando-se a picos de tráfego ou projetos temporários (Odun-Ayo; Ajayi; Okereke, 2017).

Além disso, a tecnologia viabiliza recuperação de desastres com replicação de VMs, impulsiona a computação de borda (processamento local em dispositivos IoT) e prolonga a vida útil de sistemas legados em *hardware* moderno (Odun-Ayo; Ajayi; Okereke, 2017). Com a virtualização de *desktops*, por exemplo, é possível centralizar a gestão de estações de trabalho em um *datacenter*, oferecendo acesso remoto a usuários sem comprometer desempenho ou segurança (HyungJik Lee; Jeu, 2010). Em cenários de *clusters* virtuais em *High-Performance Computing* (HPC), tarefas complexas como simulações científicas são distribuídas entre núcleos físicos de CPU, maximizando o desempenho.

Portanto, as vantagens da virtualização incluem melhor aproveitamento do *hardware*, redução de custos com infraestrutura, facilidade de *backup* e recuperação, maior escalabilidade e flexibilidade, além de isolamento de ambientes, o que aumenta a segurança e o controle (Babu *et al.,* 2014; Hage Hassan *et al.,* 2022; Liang Chen; Longchuan Yan; Yifan Mao, 2014; Mao, 2024; Odun-Ayo; Ajayi; Okereke, 2017; Tian; Gao, 2024). Essas aplicações evidenciam como a virtualização transcende a mera otimização de recursos: ela redefine paradigmas operacionais, promovendo escalabilidade, resiliência e inovação em setores que exigem agilidade e controle sobre ambientes computacionais. Dessa forma, a virtualização se torna um pilar fundamental para *data centers* modernos, *cloud computing* e estratégias de TI mais ágeis e eficientes.

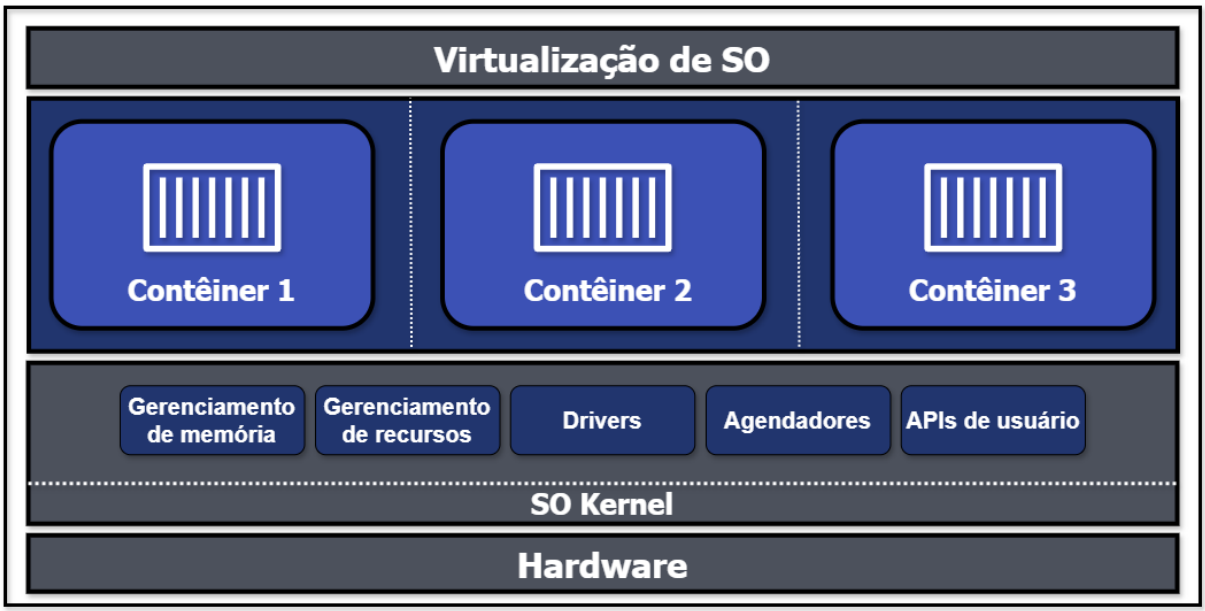
3.1.1. VIRTUALIZAÇÃO BASEADA EM CONTÊINERES

Com a evolução das técnicas de virtualização, surgiu a Virtualização Baseada em Contêineres, a qual é uma técnica que compartilha o mesmo núcleo (*kernel*) do sistema operacional hospedeiro, mas isola aplicações e suas dependências em

ambientes autônomos chamados contêineres. Diferentemente da virtualização tradicional de VMs, que emula *hardware* completo e exige uma cópia inteira do sistema operacional para cada instância, os contêineres utilizam *namespaces* e *cgroups* para garantir isolamento de processos, sistema de arquivos e recursos (CPU, memória, E/S) (Vazquez, 2024b). Essa leveza reduz drasticamente o consumo extra (*overhead*) de execução, permitindo inicialização em segundos e densidade muito maior de *workloads* por *host* (Dordevic; Stefanovic; Timcenko, 2022; Dua *et al.,* 2016).

Enquanto um hipervisor de Tipo 1 ou Tipo 2 cria VMs completas com seus próprios *kernels*, o Docker e os Linux Containers (LXC) oferecem virtualização no nível de sistema operacional, onde todos os contêineres compartilham o mesmo *kernel* do sistema operacional (Li *et al.*, 2023; Odun-Ayo; Ajayi; Okereke, 2017). Em termos de desempenho, contêineres chegam muito próximos a processos nativos, pois não há tradução de instruções nem emulação de *hardware*, apenas uma camada de abstração para controle de recursos(Babu *et al.,* 2014; Dordevic; Stefanovic; Timcenko, 2022; Dua *et al*., 2016). A Figura 25 ilustra a arquitetura da Virtualização Baseada em Contêineres.

Figura 25 – Arquitetura da virtualização baseada em contêineres.



Fonte: Adaptado de Vazquez (2024b).

Na prática, as duas abordagens frequentemente se complementam em arquiteturas modernas de TI (Li *et al.,* 2023). Por exemplo, um servidor Proxmox VE pode hospedar tanto VMs KVM, para bancos de dados críticos que demandam

isolamento rígido, quanto contêineres LXC, para microsserviços leves e escaláveis (Proxmox Server Solutions GmbH, 2025). Em ambientes de desenvolvimento e integração contínua (CI/CD), é comum utilizar contêineres para empacotar e testar aplicações rapidamente, enquanto em produção pode-se adotar VMs para serviços que necessitam de garantias adicionais de segurança e compliance (Fava *et al.,* 2024; Odun-Ayo; Ajayi; Okereke, 2017).

Em termos de armazenamento e rede, contêineres e VMs também apresentam diferenças: *snapshots* de VMs capturam o estado completo da máquina virtual, incluindo BIOS, *firmware* e dispositivos virtuais, enquanto snapshots de contêineres normalmente capturam apenas o sistema de arquivos e as configurações de *runtime* (Li *et al.,* 2023; Odun-Ayo; Ajayi; Okereke, 2017; Ye Jiaojiao; Shang Yanming, 2013). Para redes, tecnologias como Calico ou Weave Net criam redes virtuais de contêineres sobrepostas ao datacenter, comparáveis às VLANs ou SDN usadas por hipervisores para VMs, mas com configurações mais dinâmicas e integradas ao orquestrador (Kubernetes, Docker Swarm) (Li *et al.*, 2023; Ye Jiaojiao; Shang Yanming, 2013).

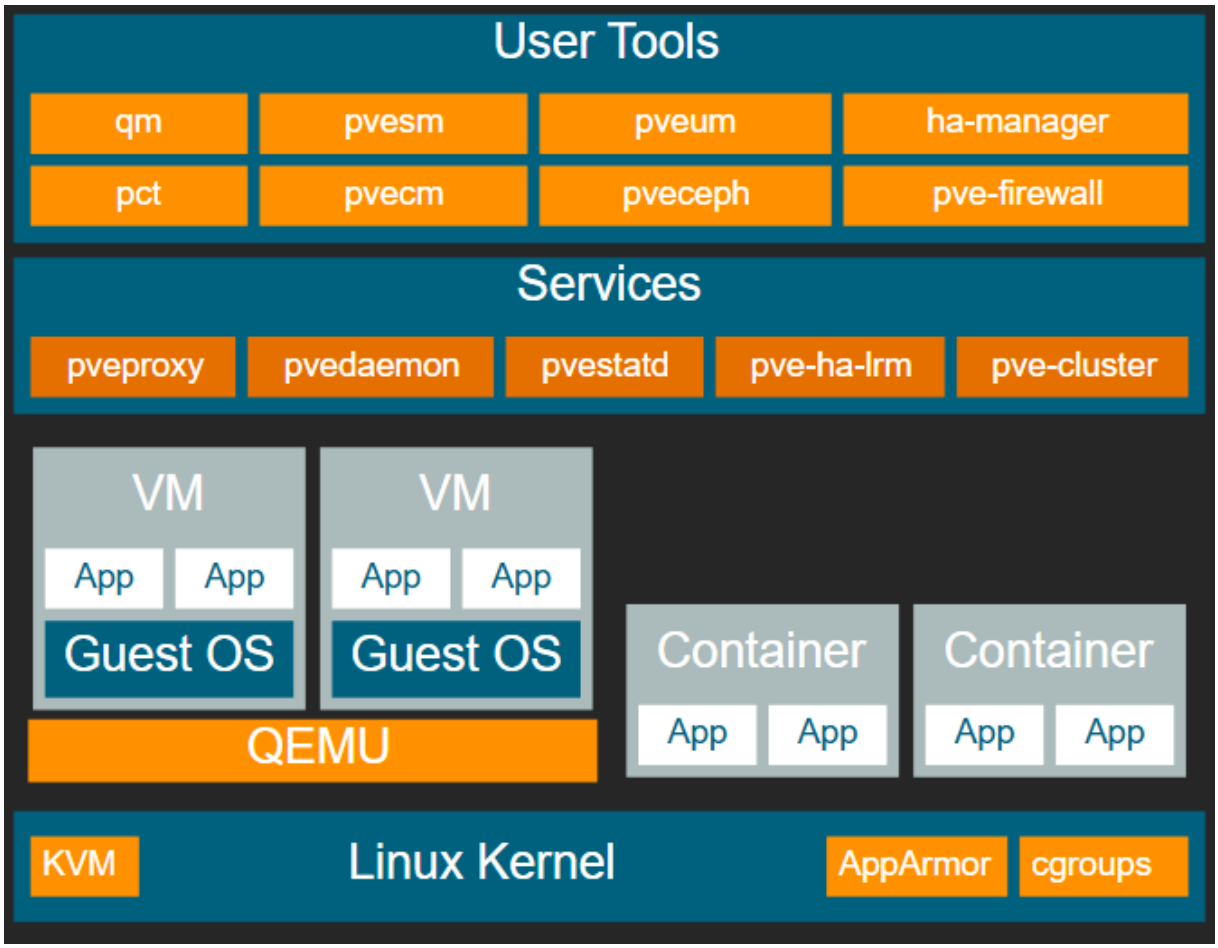
Portanto, a virtualização por contêineres é considerada uma evolução da virtualização tradicional que prioriza eficiência, portabilidade e agilidade, sacrificando parte do isolamento de nível de *hardware* em favor de performance e densidade. Em arquiteturas híbridas, contêineres e VMs coexistem, cada um atendendo cenários específicos: contêineres para microsserviços e pipelines de CI/CD, VMs para cargas de trabalho legadas, sensíveis ou que demandam kernels distintos. Dessa forma, por esses motivos, se optou pela combinação de ambas as técnicas no desenvolvimento da metodologia da dissertação, o que permitiu a otimizar os recursos disponíveis, o aumentar flexibilidade e acelerar os ciclos de implantação dos ambientes de desenvolvimento definidos.

3.1.2. PROXMOX VE

O Proxmox VE (*Virtual Environment*) é uma plataforma de virtualização de código aberto baseada em Debian que integra duas tecnologias principais: *Kernel-based Virtual Machine* (KVM) para máquinas virtuais completas e LXC para contêineres leves, sendo considerado um hipervisor do Tipo 1. Essa combinação permite ao administrador escolher, para cada *workload*, entre o isolamento total de

uma VM ou a eficiência de recursos de um contêiner. Além disso, o Proxmox VE dispõe de um núcleo de ferramentas de administração avançada por meio de Interface de Linha de Comando (CLI) e serviços nativos (Proxmox Server Solutions GmbH, 2025). A Figura 26 ilustra a arquitetura de ferramentas e serviços do Proxmox VE.

Figura 26 – Arquitetura, serviços e ferramentas do Proxmox VE.

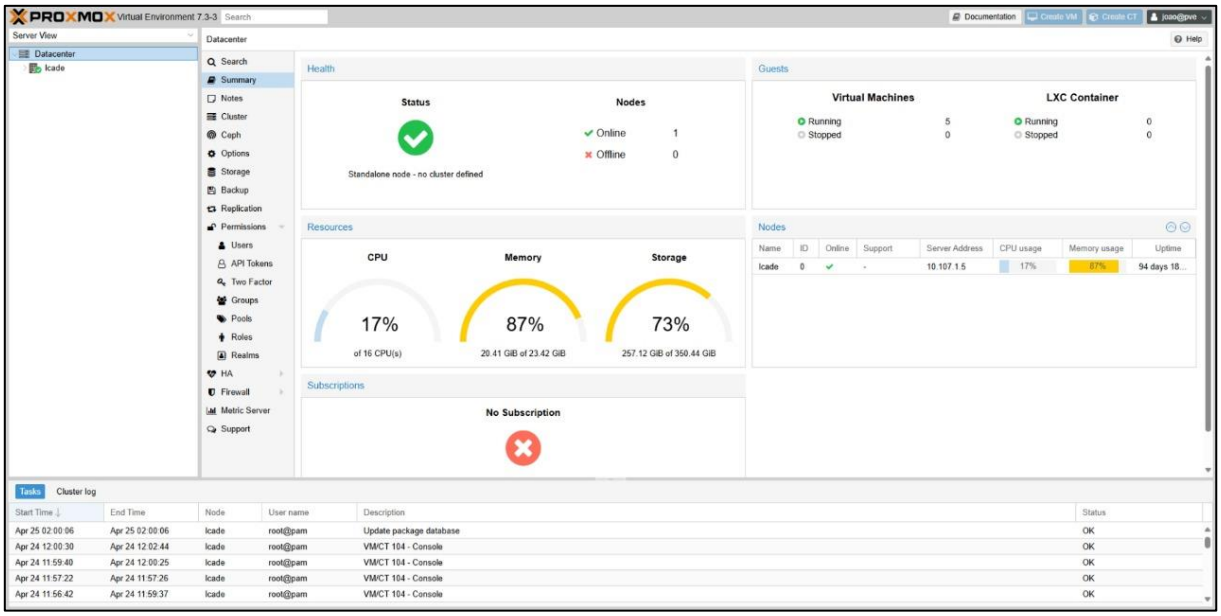


Fonte: Proxmox Server Solutions GmbH (2025).

No Proxmox VE, por meio de uma interface de gerenciamento totalmente *web-based* e responsiva que dispensa a necessidade de clientes nativos, é possível criar, iniciar, parar e migrar VMs e contêineres, configurar redes virtuais (*bridges*, *bonds*, VLANs), definir armazenamento (local, NFS, iSCSI, Ceph, ZFS) e gerenciar *snapshots* e *backups* (Figura 27). Junto à essas funcionalidades, o painel também oferece visualização em tempo real do uso de CPU, memória, I/O de disco e tráfego de rede, além de *logs* de eventos e tarefas (Proxmox Server Solutions GmbH, 2025).

Em ambientes corporativos, o Proxmox VE se destaca pelo suporte nativo a *clusters*: vários nós Proxmox podem ser agrupados para *High Availability* (HA), *failover* automático de VMs e balanceamento de carga. O mecanismo de HA monitora continuamente os serviços críticos e, em caso de falha de um nó, reinicia automaticamente as VMs afetadas em outro nó disponível. A configuração de *cluster* é simplificada pela ferramenta integrada pvecm, que gerencia *quorum*, *fencing* e comunicação segura entre nós (Proxmox Server Solutions GmbH, 2025).

Figura 27 – Interface gráfica do Proxmox VE.



Fonte: Elaborado pelo autor (2025).

No quesito armazenamento, o Proxmox VE oferece flexibilidade para diferentes necessidades: suporta armazenamento local (diretórios, LVM, ZFS), bem como soluções de rede como NFS, CIFS, iSCSI e integrações avançadas com *clusters* de armazenamento distribuído Ceph. O uso de ZFS como *backend* local traz recursos avançados como compressão transparente, deduplicação, *snapshots* eficientes e reparo automático de dados. Já com Ceph, é possível escalar armazenamento em vários nós com replicação e recuperação automática (Proxmox Server Solutions GmbH, 2025).

O Proxmox VE inclui também um sistema robusto de *backup* e restauração, a partir do qual *backups* podem ser agendados com frequência configurável, escolhendo entre *snapshots* consistentes ou *backups* "*stop-and-go*" para VMs e contêineres. O *console* permite restaurar rapidamente um sistema inteiro ou apenas

discos individuais, além de migrar *backups* entre diferentes servidores Proxmox ou ambientes (Proxmox Server Solutions GmbH, 2025).

Dentre as características do hipervisor mais desejáveis ao escopo do projeto, está a capacidade de criação de Infraestruturas Hiperconvergentes (HCI). As HCI são especialmente úteis para implementações em que uma alta demanda de infraestrutura atende a aplicações baixo orçamento e pequeno porte (Proxmox Server Solutions GmbH, 2025). Ao implantar uma HCI, temos vantagens como:

- escalabilidade contínua, rápida e independente;
- baixo custo agregado aos componentes oferecidos;
- proteção e eficiência de dados por meio dos serviços de backup e recuperação;
- simplicidade de configuração e administração centralizada; e
- código aberto, prevenindo o "*vendor lock-in*[9]".

Por último, também é possível definir, utilizando a função *External Metric Server*, servidores de métricas externos os quais receberão, periodicamente, várias estatísticas dos *hosts*, *virtual guests* e seus respectivos recursos, como CPU, memória, armazenamento em disco e tráfego de rede (Proxmox Server Solutions GmbH, 2025). A partir dessa função, foi estabelecida uma das infraestruturas da rede de observabilidade e monitoramento implantada para a dissertação, por meio da configuração do *plugin* do banco de dados de séries temporais InfluxDB e a aplicação *web* de visualização interativa Grafana.

Dessa forma, considerando todos os recursos, ferramentas e serviços provisionados pelo hipervisor, foi selecionado o Proxmox VE versão 7.3-3 para o gerenciamento das VMs nas quais foram implantados os ambientes de desenvolvimento para cada *middleware* IoT estudado, sendo instalado diretamente em um servidor Dell PowerEdge T320 de propriedade do CEAMAZON.

---

[9] *Vendor lock-in* (ou aprisionamento ao fornecedor) é uma situação em que um cliente ou organização se torna excessivamente dependente de um fornecedor específico de tecnologia — seja *hardware*, *software*, plataforma ou serviço — dificultando ou tornando onerosa a migração para alternativas concorrentes.

3.1.3. DOCKER

O Docker é uma plataforma de código aberto que permite empacotar aplicações em contêineres leves e portáteis, onde cada instância inclui tudo o que o *software* precisa para rodar — código, *runtime*, bibliotecas e configurações —, garantindo isolamento completo do sistema operacional subjacente (Docker Inc, 2025). A plataforma representa uma das mais populares soluções para aplicações desenvolvidas por meio da técnica de virtualização baseada em contêineres (Dhakate; Godbole, 2015; Li, 2021).

No coração do Docker está o Docker Engine, um *daemon*[10] (lê-se “dêimon”) que alavanca funcionalidades nativas do sistema operacional para isolar processos e controlar o uso de CPU, memória e disco. A interação se dá principalmente por uma CLI, que oferece comandos para criar, iniciar, parar e remover contêineres, bem como para construir imagens a partir de Dockerfiles — arquivos declarativos onde cada instrução gera uma camada de imagem. Além da CLI, uma API *Representational State Transfer* (REST) facilita a integração do Docker com outras ferramentas de automação e interfaces gráficas (Docker Inc, 2025).

Como as imagens Docker são compostas por múltiplas camadas somente leitura, construídas incrementalmente conforme as instruções do *Dockerfile*, novos *builds* são acelerados ao reutilizar *cache* de camadas já existentes, seja de repositórios públicos como o Docker Hub, seja de *registries* privados, importantes para empresas que exigem controle de acesso e conformidade (Kumar *et al.,* 2024). Para armazenamento persistente, o Docker disponibiliza volumes gerenciados e *bind mounts*, enquanto a rede de contêineres pode usar desde a *bridge* padrão até soluções avançadas como *overlay*, *host* e *macvlan*, adequadas a arquiteturas distribuídas (Docker Inc, 2025; Dordevic; Stefanovic; Timcenko, 2022).

Em produção, o Docker geralmente é combinado com sistemas de orquestração — Kubernetes, Docker Swarm ou Nomad — que programam contêineres em *clusters*, cuidando de balanceamento de carga, escalonamento automático e recuperação de falhas, como visualizado em Marathe; Gandhi; Shah,

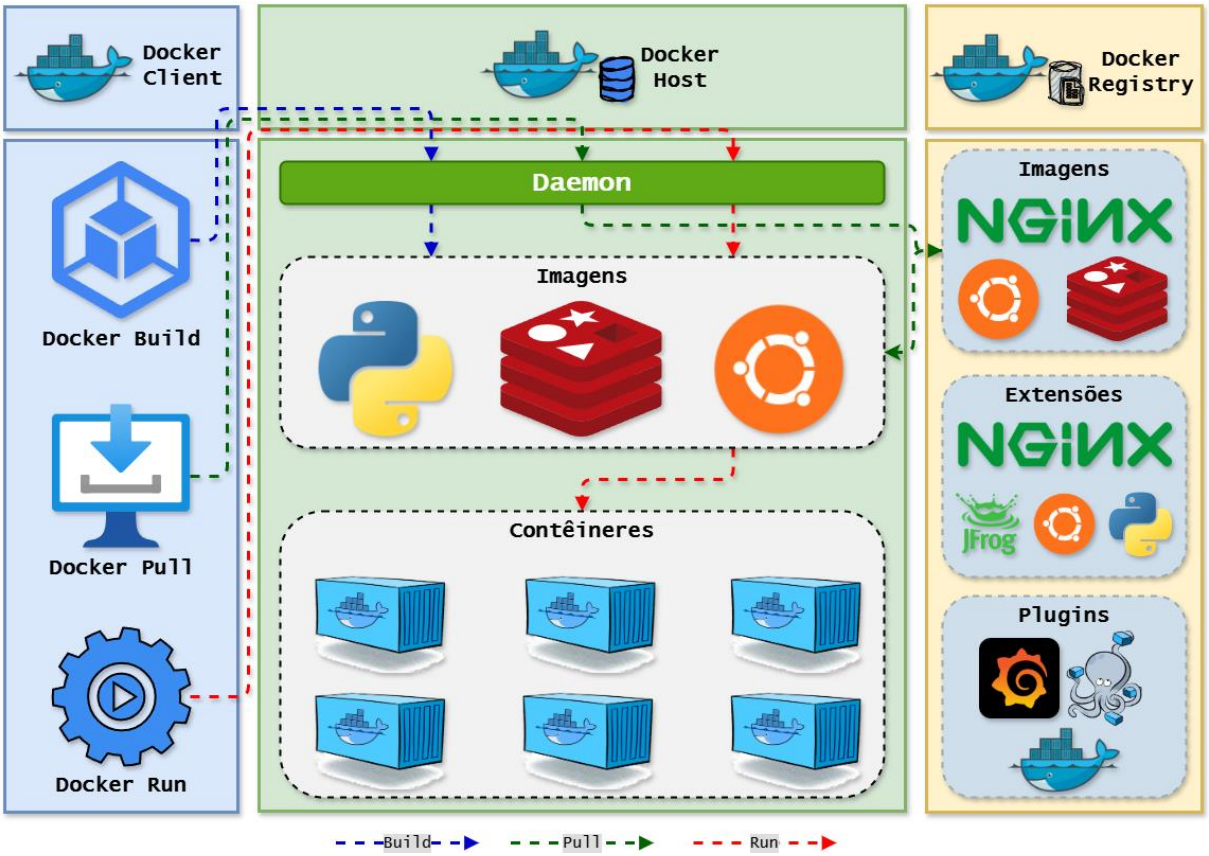
---

[10] *Daemon* é um processo em segundo plano que executa tarefas contínuas ou responde a eventos, sendo essencial para o funcionamento automatizado de serviços em sistemas operacionais.

(2019) e Adhipta; Selo; Widyawan, (2023). Essa união potencializa práticas de microsserviços, Integração Contínua (CI) e Entrega Contínua (CD), consolidando o Docker como padrão na indústria de *software* (John *et al.,* 2024; Kumar *et al.,* 2024). Por conseguinte, graças à sua portabilidade, abordagem declarativa e ecossistema rico, o Docker transformou o ciclo de desenvolvimento e impulsionou a adoção de infraestruturas escaláveis e resilientes (Eyvazov *et al.,* 2024).

A arquitetura do Docker é organizada em três camadas principais: *Client* (Cliente), *Docker Host* e *Registry* (Registro). Cada uma dessas camadas possui responsabilidades distintas e interage de maneira coordenada para permitir o empacotamento, a distribuição e a execução de aplicações de forma eficiente, isolada e portátil. A Figura 28 ilustra a arquitetura funcional dos componentes que integram o ecossistema Docker.

Figura 28 – Arquitetura do Docker.



Fonte: Elaborado pelo autor (2025).

A camada *Client* representa a interface de linha de comando utilizada pelos usuários para enviar instruções ao Docker, na qual se destacam os comandos mais comuns, como: *docker build*, utilizado para a construção de imagens a partir de um

Dockerfile; *docker pull*, que realiza a obtenção de imagens hospedadas em um repositório remoto; e *docker run*, que permite a execução de contêineres a partir de imagens previamente disponíveis. Vale ressaltar que tais comandos não executam diretamente as tarefas solicitadas, mas são repassados ao Docker *daemon*, responsável pela execução das operações (Docker Inc, 2025).

A camada *Docker Host* corresponde ao ambiente físico ou virtual onde o Docker está instalado, onde reside o *daemon*, contemplando dois elementos centrais: as imagens, que são pacotes imutáveis contendo o ambiente necessário para a execução de uma aplicação (incluindo bibliotecas, dependências e configurações); e os contêineres, que representam as instâncias em execução dessas imagens. O *daemon*, por sua vez, atua como intermediário entre o cliente e os recursos internos do sistema, garantindo o correto funcionamento das operações solicitadas (Docker Inc, 2025).

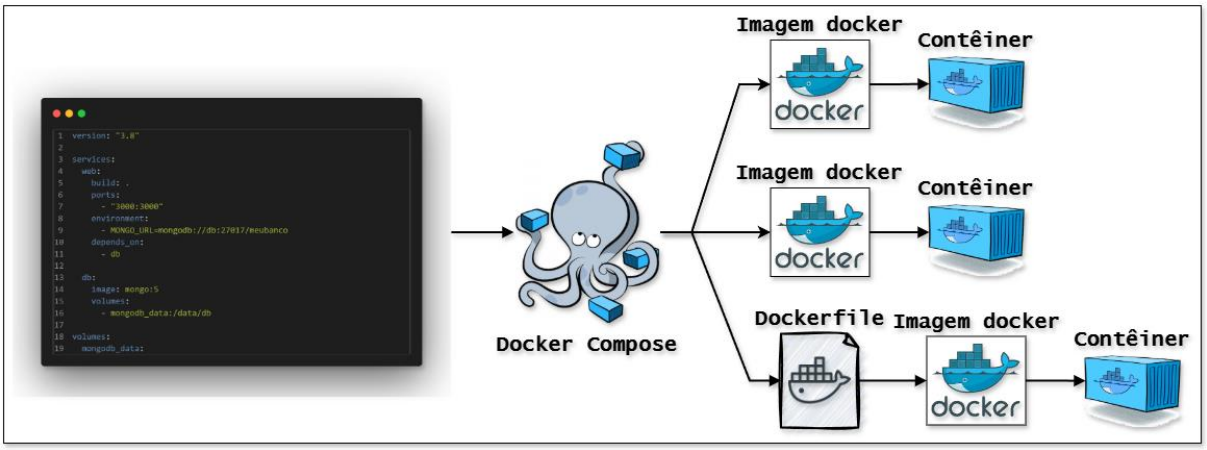
Na última camada há o *Registry*, que consiste na infraestrutura de repositório responsável pelo armazenamento, versionamento e disponibilização de imagens Docker, por meio de qual o Docker *daemon* pode buscar imagens públicas ou privadas utilizando o comando *pull*, bem como enviar novas imagens construídas localmente através do comando *push* (Docker Inc, 2025).

Por fim, dentre os principais componentes do Docker, há o Docker Compose, que é uma ferramenta oficial projetada para facilitar a definição, configuração e execução de aplicações que envolvem múltiplos contêineres. Em vez de iniciar cada contêiner separadamente com comandos individuais, o Docker Compose permite que todos os serviços necessários a uma aplicação sejam descritos em um único arquivo *YAML*, intitulado "*docker-compose.yml*" (Docker Inc, 2025; Medium, 2024). Com isso, o processo de inicialização de um ambiente completo pode ser feito com apenas um comando, como *docker compose up*.

As principais vantagens do Docker Compose incluem a automação de tarefas repetitivas, a padronização de ambientes de desenvolvimento e a facilidade de replicar e escalar aplicações. Ele é especialmente útil em contextos de desenvolvimento local, testes e CI, permitindo que equipes trabalhem com ambientes idênticos sem necessidade de configurar manualmente cada serviço (Lai; Lai, 2024; Reis *et al.,* 2022). Além disso, o Compose oferece recursos como a definição de dependências entre serviços, controle de volumes e redes, e escalabilidade horizontal por meio de

comandos simples, o que torna o gerenciamento de aplicações complexas mais eficiente, promovendo agilidade e consistência em todo o ciclo de desenvolvimento e implantação (Eyvazov *et al.,* 2024). A Figura 29 apresenta um fluxograma que resume a arquitetura e funcionamento do Docker Compose.

Figura 29 – Arquitetura e funcionamento do Docker Compose.



Fonte: Elaborado pelo autor (2025).

3.2. *MIDDLEWARES* IoT

Um *middleware* IoT é uma camada de *software* que fica entre os dispositivos físicos (sensores, atuadores, sistemas embarcados, *gateways* etc.) e as aplicações de usuário ou de negócios, oferecendo serviços padronizados de comunicação, gerenciamento e processamento de dados (Souki *et al.*, 2022; Zhang *et al.*, 2021). Seu principal objetivo é abstrair a complexidade da heterogeneidade de protocolos, formatos de mensagens e plataformas de *hardware*, permitindo que desenvolvedores se concentrem na lógica de negócio em vez de detalhes de conectividade (Deohate; Rojatkar, 2021; Pereira *et al.*, 2023).

Portanto, um *middleware* IoT atua como uma camada intermediária responsável por integrar dados provenientes de dispositivos heterogêneos, possibilitando sua comunicação eficiente e a tomada de decisões baseada nas informações coletadas. Dessa forma, essas plataformas devem atender a diversos requisitos fundamentais para garantir sua eficácia e aplicabilidade em ambientes distribuídos, dentre os quais destacam-se a escalabilidade, a alta disponibilidade, a interoperabilidade entre diferentes tecnologias, a segurança na manipulação dos

dados e a facilidade de implantação, manutenção e utilização do sistema (Jepsen *et al.,* 2021; Kang *et al.,* 2018). O Quadro 1 detalha essas funções.

Quadro 1 – Funções de um *middleware* IoT.

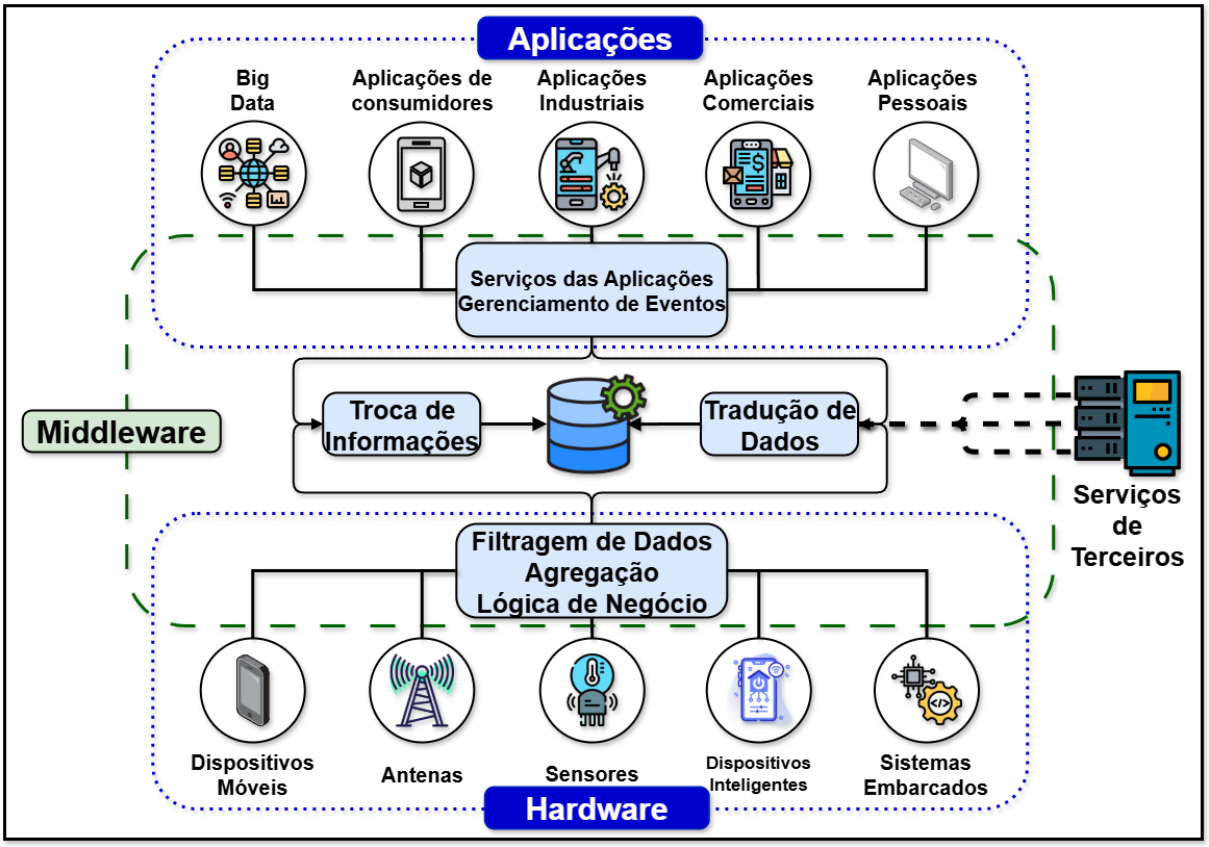
Função	Referência
Descoberta e registro de dispositivos: mantém um catálogo centralizado de dispositivos, suas capacidades e metadados, facilitando a localização e o controle de cada elemento na rede.	(Aslam; Curry, 2021; Goumopoulos, 2024; Souki *et al.*, 2022)
Coleta e roteamento de mensagens: recebe dados de telemetria de múltiplos protocolos (por exemplo, MQTT, CoAP, HTTP) e os encaminha para os componentes responsáveis pelo armazenamento, análise ou visualização.	(Andrei *et al.*, 2020; Goumopoulos, 2024; Trunzer *et al.*, 2020; Yongguo *et al.*, 2019)
Gerenciamento de estado e contexto: armazena o estado atual de cada dispositivo (por exemplo, valor de um sensor, status de um atuador) e permite consultas e assinaturas para notificações em tempo real quando esse estado muda.	(Goumopoulos, 2024; Pradeep; Krishnamoorthy; Vasilakos, 2021; Souki *et al.*, 2022; Trunzer *et al.*, 2020)
Persistência e histórico: grava séries temporais e eventos em bancos de dados especializados, garantindo que informações passadas possam ser recuperadas para relatórios, análises ou auditorias.	(Danish; Zhang; Jacobsen, 2020; Peros; Joosen; Hughes, 2021; Shen *et al.*, 2023)
Segurança e controle de acesso: implementa autenticação de dispositivos, criptografia de comunicação e políticas de autorização para garantir que apenas entidades legítimas possam enviar comandos ou ler dados sensíveis.	(Castilho; Godoy; Salmen, 2020; Desuert *et al.*, 2022; Jarwar *et al.,* 2023)
Orquestração de fluxo de dados: permite a criação de *pipelines* ou “fluxos” em que os dados são filtrados, transformados, agregados e encaminhados conforme regras definidas, suportando desde simples alertas até análises avançadas em tempo real.	(Goumopoulos, 2024; Stock; Schel; Bauernhansl, 2020)
APIs padronizadas: expõe interfaces REST, WebSocket ou gRPC que uniformizam a forma de interação das aplicações com o ecossistema IoT, promovendo interoperabilidade entre diferentes módulos e fornecedores.	(Andrei *et al.,* 2020; Jepsen *et al.,* 2021; Pereira *et al.,* 2023; Samwini; Awais; Pereira, 2023)

Fonte: Elaborado pelo autor (2025).

Por conseguinte, devidos as suas características intrínsecas, a adoção de um *middleware* IoT proporciona às organizações benefícios significativos em termos de escalabilidade e flexibilidade. Com essa abordagem, é possível adicionar ou remover dispositivos da infraestrutura sem a necessidade de reescrever códigos de integração, o que simplifica a expansão do sistema. Além disso, novos serviços — como painéis de visualização, análises preditivas ou integrações com sistemas legados — podem ser incorporados de forma mais ágil, aproveitando a mesma base de comunicação. Nesse contexto, o *middleware* assume um papel central na arquitetura IoT, atuando

como o núcleo que coordena os fluxos de dados, assegura a segurança da informação e mantém a consistência operacional em uma infraestrutura amplamente distribuída, como ilustrado na Figura 30.

Figura 30 – Camadas de uma arquitetura IoT com a aplicação de um *middleware* IoT.



Fonte: Elaboração pelo autor (2025).

3.2.1. DOJOT

A Dojot é uma plataforma brasileira de código aberto desenvolvida pelo Centro de Pesquisa e Desenvolvimento em Telecomunicações (CPqD), com o propósito de facilitar a criação de soluções de IoT, especialmente adaptadas às necessidades do contexto nacional. Inicialmente voltada para aplicações em cidades inteligentes, a Dojot foi concebida para atender áreas como segurança pública, mobilidade urbana e saúde, proporcionando um ambiente de desenvolvimento ágil, flexível e escalável (IoT Labs, 2025).

Um dos principais diferenciais da plataforma é sua arquitetura baseada em microsserviços e contêineres Docker, o que permite uma implantação modular e personalizável conforme as exigências de cada aplicação. Além disso, a Dojot

disponibiliza APIs abertas, que facilitam a integração com outras aplicações e sistemas orientados a serviços (CPqD, 2020; IoT Labs, 2025).

A Dojot oferece um conjunto de funcionalidades essenciais que a tornam uma plataforma robusta para o desenvolvimento de soluções em Internet das Coisas, dentre as quais destacam-se a capacidade de conectar e gerenciar dispositivos IoT de forma eficiente, o armazenamento de grandes volumes de dados em formatos diversos e a construção de fluxos e regras de processamento por meio de uma interface gráfica intuitiva (Figura 31). Além disso, a plataforma suporta o tratamento de eventos em tempo real, permitindo a criação de ações automatizadas baseadas nos dados coletados (CPqD, 2020; IoT Labs, 2025).

Figura 31 – Funcionalidades da Plataforma Dojot.

Fonte: CPqD (2025).

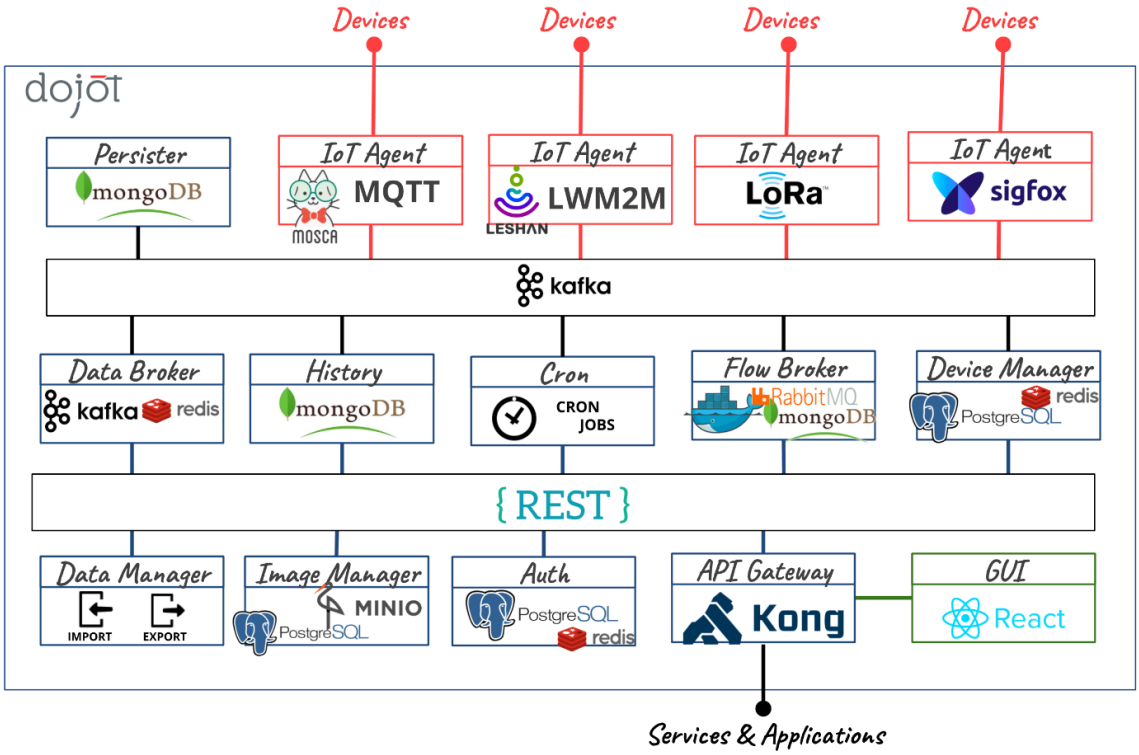
Dessa forma, a Dojot atua como um *middleware* eficaz, promovendo a integração entre dispositivos físicos e aplicações digitais e contribuindo para a viabilização de soluções inteligentes em diferentes contextos.

3.2.1.1. ARQUITETURA, COMPONENTES E COMUNICAÇÃO

Projetada para possibilitar a prototipagem rápida de soluções, fornecendo uma plataforma fácil de usar, dimensionável e robusta, sua arquitetura interna faz uso de muitos componentes de código aberto conhecidos com outros projetados e implementados pela equipe da Dojot (CPqD, 2020). A arquitetura da Dojot é fundamentada em uma abordagem de microsserviços conteinerizados organizados em diferentes camadas (Figura 32), cada uma com funções específicas no

ecossistema da plataforma, que promove escalabilidade, modularidade e flexibilidade na construção de aplicações IoT.

Figura 32 – Arquitetura interna da plataforma Dojot.



Fonte: CPqD (2020).

São as camadas constituintes da arquitetura e suas respectivas funções:

- **Camada de conectividade**: é responsável pela comunicação com dispositivos, suportando protocolos como MQTT e HTTP, viabilizando a ingestão de dados dos dispositivos conectados à plataforma;
- **Camada de dados**: inclui componentes dedicados ao armazenamento e à gestão da persistência das informações, sendo composta por serviços que armazenam tanto dados históricos (como séries temporais) quanto metadados dos dispositivos;
- **Camada de processamento**: fornece mecanismos para o tratamento e transformação dos dados recebidos, o que inclui o uso de fluxos lógicos para definição de regras, condições e ações que devem ser executadas com base em eventos de entrada; e
- **Camada de infraestrutura e suporte**: gerencia autenticação, autorização, monitoramento, serviços de descoberta e orquestração dos

microsserviços. A integração entre esses componentes assegura a interoperabilidade, a segurança e a escalabilidade da plataforma, tornando a Dojot uma solução adequada para diversos cenários de aplicações IoT.

O Quadro 2, a seguir, oferece uma visão geral dos principais componentes que compõem a arquitetura da plataforma Dojot, juntamente com suas respectivas funções, conforme descrito na documentação oficial (CPqD, 2020). Essa síntese tem como objetivo facilitar a compreensão do papel desempenhado por cada serviço no ecossistema da Dojot.

Quadro 2 – Componentes do *middleware* IoT Dojot.

Componente	**Função**
Persister	Componente responsável por conduzir todos os dados de telemetria e eventos gerados pelos dispositivos para serem armazenados no banco de dados MongoDB.
IoT Agent	Serviço que faz a adaptação entre os dispositivos físicos e os componentes principais do Dojot.
Kafka + Data Broker	Componente responsável por transmitir ou consumir dados em canais através de tópicos.
History	Microsserviço que consulta os dados de telemetria persistidos no banco de dados através da API Rest.
Cron	Microsserviço que permite agendar eventos a serem emitidos – ou requisições a serem feitas – para outros micros serviços dentro da Dojot.
Flowbroker	Serviço que fornece mecanismos para construir fluxos de processamento dos dados de telemetria para executar um conjunto de ações.
Device Manager	Entidade central responsável por armazenar as estruturas de dados de dispositivos e modelos (*templates*). Ele também é responsável por publicar quaisquer atualizações para todos os componentes interessados por meio do Kafka.
Data Manager	Serviço que gerencia a configuração de dados da Dojot, possibilitando importar e exportar os dados de modelos (*templates*), dispositivos, fluxos e tarefas agendadas da Dojot através de um arquivo JSON (.json). Vale ressaltar que os dados de telemetria não fazem parte da função de importar e exportar.
Image Manager	Componente responsável pelo armazenamento e recuperação de imagens de *firmware* de dispositivos.
Serviço de autorização de usuário (Auth)	Serviço que gerencia perfis de usuários e controla os acessos.
Kong API Gateway	É utilizado como um ponto de fronteira (*entry point*) entre as aplicações e serviços externos e os serviços internos da Dojot.
GUI	É uma aplicação web que provê interfaces responsivas para gerenciamento da Dojot e inclui as seguintes funcionalidades de gerenciamento: perfil de usuários, usuários, modelos de dispositivos, dispositivos, fluxos de processamento, notificações, alterar a senha, importar e exportar a Dojot.
Kafka2FTP	Serviço kafka2ftp permite o encaminhamento de mensagens do Apache Kafka para servidores FTP.

Fonte: Adaptado de CPqD (2020).

Quadro 2 – Componentes do *middleware* IoT Dojot.

(continua)

InfluxDB Storer e Retriever	Os serviços InfluxDB Storer e InfluxDB Retriever trabalham juntos. O InfluxDB Storer é responsável por consumir os dados Kafka de dispositivos e gravá-los no InfluxDB, já o InfluxDB Retriever tem a função de obter os dados que foram escritos pelo InfluxDB Storer no InfluxDB via API REST.
Gerenciador de identidade X.509	Componente responsável por atribuir identidades a dispositivos, tais identidades são representadas na forma de certificados x.509.
Kafka WS	Componente é responsável por obter dados em tempo real do Apache Kafka via uma conexão *websocket*.

Fonte: Adaptado de CPqD (2020).

Por último, a comunicação entre a plataforma Dojot, seus usuários e sistemas externos é viabilizada por meio de *APIs RESTful* padronizadas, além de uma interface gráfica que possibilita o gerenciamento visual de dispositivos, fluxos e dados, como visualizado na Figura 32. Além disso, de acordo com a documentação oficial da plataforma, os componentes internos da plataforma podem interagir entre si por duas vias principais: i) requisições *HTTP* ou ii) troca de mensagens utilizando o sistema de mensageria *Kafka* (Quadro 3).

Quadro 3 – Vias de comunicação da plataforma Dojot.

Via de comunicação	**Descrição**
Requisições HTTP	Quando um componente da plataforma precisa obter informações de outro, ele pode realizar essa comunicação por meio de uma requisição HTTP. Por exemplo, caso um IoT Agent necessite acessar a lista de dispositivos previamente configurados, ele deve enviar uma solicitação HTTP ao componente responsável pelo gerenciamento desses dispositivos.
Mensagens Kafka	Quando um componente deseja disseminar novas informações sobre um recurso sob seu controle, ele pode publicá-las por meio do sistema de mensageria Kafka. Dessa forma, qualquer outro componente interessado nesse tipo de informação pode simplesmente subscrever-se ao tópico correspondente e "escutar" as mensagens ali publicadas, promovendo uma comunicação assíncrona e desacoplada entre os serviços da plataforma.

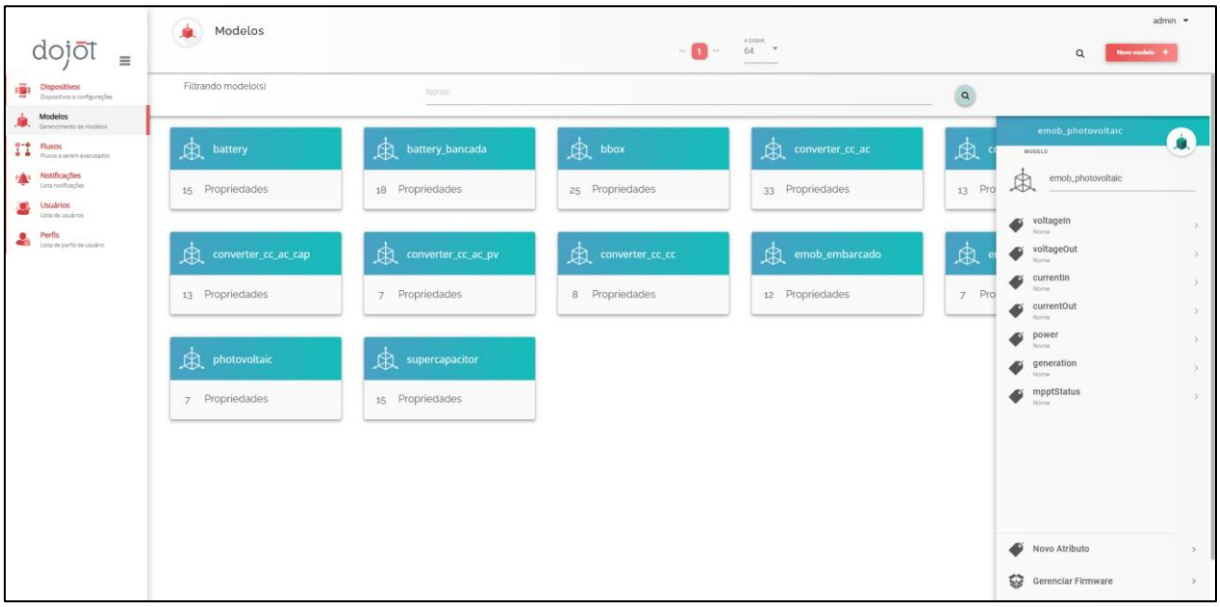
Fonte: CPqD (2020).

3.2.1.2. INTERFACE WEB

Na plataforma Dojot, um *device template* corresponde ao modelo de dispositivo: um artefato declarativo em que se definem, de forma estruturada, os atributos (sensores e atuadores), tipos de dado, metadados e comandos que caracterizam um determinado tipo de equipamento (Figura 33). Cada *template* contém ainda informações de validação (por exemplo, limites de valores, unidades de medida) e pode ser versionado para acomodar evoluções no *firmware* ou no conjunto de

funcionalidades de uma família de dispositivos (CPqD, 2020). Essa abordagem orientada a *template* garante que todos os dispositivos de um mesmo tipo compartilhem uma estrutura homogênea de dados, facilitando sua ingestão, o roteamento de mensagens e a aplicação de regras de negócio.

Figura 33 – Exemplo de *template* da plataforma Dojot.



Fonte: Elaborado pelo autor (2025).

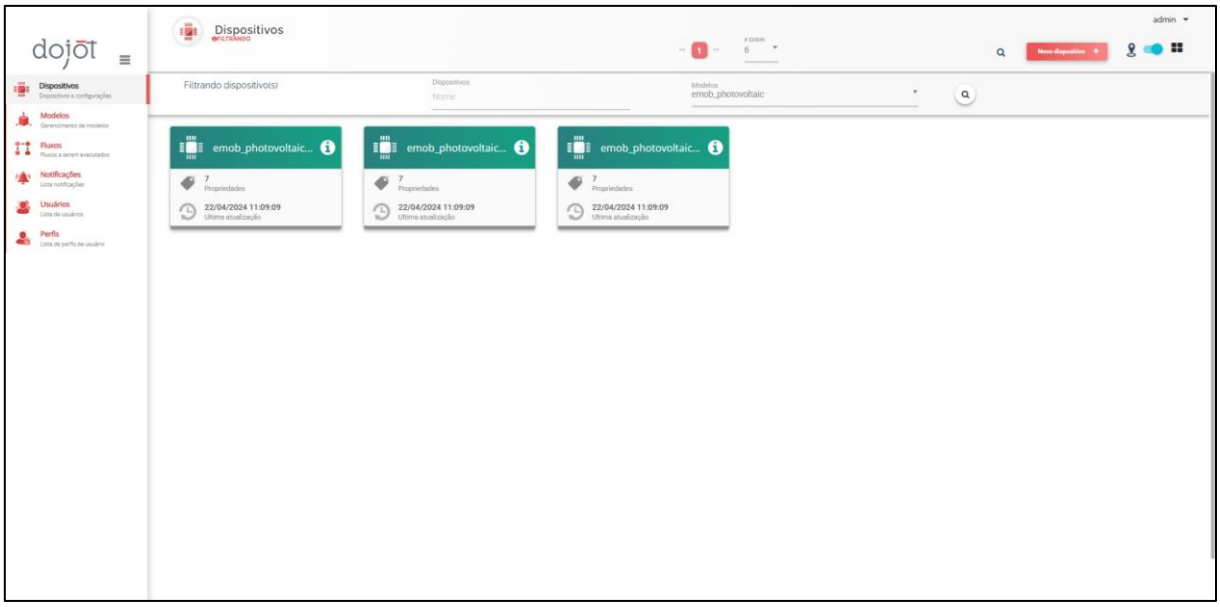
Já os *devices* (dispositivos) são as instâncias concretas criadas a partir desses *templates* (Figura 34). Quando um dispositivo físico se conecta ao Dojot — via IoT Agent ou *gateway* de borda —, ele é registrado na plataforma por meio do seu *template* correspondente, tornando-se um objeto “vivo” capaz de enviar telemetria e receber comandos.

Cada *device* carrega consigo o identificador único, seus valores de atributo atualizados em tempo real e o histórico de comunicação; além disso, pode ser individualmente configurado (por exemplo, ativação de notificações, agendamento de comandos ou ajustes de política de segurança) (CPqD, 2020). Dessa forma, *templates* e devices formam juntos o núcleo do gerenciamento de ativos no Dojot, permitindo abstrair a heterogeneidade do *hardware* e centralizar toda a lógica de integração e monitoramento em um só ambiente.

Praticamente, quando um dispositivo é criado, o usuário seleciona um ou mais modelos que serão usados para compor o novo dispositivo. Assim, os atributos de todos os modelos selecionados são combinados e associados ao dispositivo,

permitindo que ele herde características de vários modelos, juntamente com um identificador exclusivo (ID) para cada dispositivo virtual criado.

Figura 34 – Exemplo de *device* da plataforma Dojot.



Fonte: Elaborado pelo autor (2025).

Essa flexibilidade é especialmente útil para representar dispositivos complexos que precisam de funcionalidades variadas, representando um dos principais recursos do Dojot que é a forte ligação entre modelos e dispositivos, onde qualquer atualização feita em um modelo, como a adição de um novo atributo ou a modificação de um existente, é automaticamente refletida em todos os dispositivos associados a ele. Essa dinâmica garante que todas as instâncias de dispositivos baseadas nesse modelo estejam sempre atualizadas e consistentes, simplificando a manutenção e o gerenciamento de grandes frotas de dispositivos IoT.

Além do gerenciamento de dispositivos por meio de *templates* e *devices*, a plataforma oferece funcionalidades como:

- configuração de fluxos capazes de definir *pipelines* de processamento de dados através de um editor visual;
- importação e exportação da configuração completa de um *tenant* (*templates*, *devices*, fluxos, nós remotos, agendamentos etc.) por meio de um arquivo *.JSON*, simplificando *backups* e migrações, mesmo que não inclua os dados de telemetria;

- atualização de *firmware* utilizando o protocolo *LwM2M*, por meio do qual o usuário pode habilitar o gerenciador de *firmware* em um *template*, fazer *upload* de imagens (em formato .hex) e acionar a transferência e aplicação de *firmware* nos dispositivos associados;

- geração de certificados para comunicação segura via *MQTT*/*TLS*, sendo possível gerar diretamente na GUI os certificados X.509 (chave privada, certificado do dispositivo e CA), que podem então ser usados pelos dispositivos para se autenticarem junto à plataforma;

- relatório de histórico através da funcionalidade *history*, a qual permite selecionar um ou mais atributos de um dispositivo e gerar relatórios gráficos ou tabulares sobre o seu comportamento ao longo de um período definido pelo usuário; e

- acesso ao *dashboard* por meio da GUI V2, a qual pode ser acessada através do *endpoint* “http://localhost:8000/v2”. Nesse ambiente, cria-se visualizações customizadas, escolhendo dispositivos, atributos e filtros (últimos registros, intervalo, dados em tempo real), para monitoramento contínuo dos recursos implantados.

3.2.2. FIWARE

FIWARE é uma plataforma de código aberto voltada para o desenvolvimento de soluções inteligentes, gêmeos digitais e Data Spaces em uma ampla variedade de domínios e transformação digital como *Smart AgriFood*, *Smart Cities*, *Smart Energy*, *Smart Industry* e *Smart Water* (FIWARE Foundation, 2025a).

Juntamente com seus membros e parceiros, a FIWARE Foundation impulsiona a definição - e a implementação de código aberto - dos principais padrões abertos que permitem o desenvolvimento de soluções inteligentes portáteis e interoperáveis de forma mais rápida, fácil e econômica, evitando cenários de dependência de fornecedores. Ao mesmo tempo, a tecnologia FIWARE é fomentada como um ecossistema de negócios padronizado, sustentável e voltado para a inovação (FIWARE Foundation, 2025a). Esse objetivo é sintetizado através da missão da comunidade FIWARE, que se concentra em “construir um ecossistema aberto e sustentável em torno de padrões de plataforma de *software* públicos, livres de

royalties e orientados para a implementação que facilitem o desenvolvimento de novos aplicativos inteligentes em vários setores".

Para que a missão seja satisfeita, o FIWARE traz consigo uma estrutura de componentes, os *Generic Enablers* (GEs), que podem ser implantados junto a componentes de terceiros. O catálogo de GEs dispõe de gerenciadores de API, Core Context, IoT Agents, coleta, processamento e visualização de dados, segurança e autenticação, robótica, entre outros (FIWARE Foundation, 2025b). Além disso, por meio de uma colaboração conjunta com outras fundações, a FIWARE Foundation tem desenvolvido e apoiado a adoção de uma arquitetura de referência e de modelos de dados comuns compatíveis que sustentem um mercado digital de soluções inteligentes interoperáveis e replicáveis em vários setores, começando pelas cidades inteligentes, chamados de Smart Data Models (FIWARE Foundation, 2025c).

Os Smart Data Models são modelos de dados que incluem quatro elementos: o esquema, ou representação técnica do modelo que define os tipos e a estrutura dos dados técnicos, a especificação de um documento escrito para leitores humanos, um URI com um URL funcional com dados básicos sobre o atributo ou a entidade e os exemplos de cargas úteis para as versões NGSIv2 e NGSI-LD. Ainda, todos os modelos os de dados são púbicos e de natureza livre de royalties, com um modo de licença que concede ao usuário 3 direitos: uso livre, modificação livre e compartilhamento gratuito para sempre (FIWARE Foundation, 2025c).

Em síntese, o FIWARE se destaca como uma iniciativa estratégica para promover a transformação digital baseada em tecnologias abertas, interoperáveis e sustentáveis. Ao integrar componentes reutilizáveis, padrões globais e uma comunidade ativa, a plataforma viabiliza o desenvolvimento ágil e econômico de soluções inteligentes adaptáveis a diversos setores. Com seu compromisso com a padronização, a abertura e a colaboração internacional, a FIWARE Foundation consolida um ecossistema tecnológico robusto que impulsiona a inovação digital, reduz barreiras de entrada e fortalece a soberania tecnológica de cidades, empresas e governos ao redor do mundo.

3.2.2.1. ARQUITETURA E ENTIDADES

No núcleo da plataforma FIWARE está o Orion Context Broker, um componente fundamental que permite a gestão de entidades contextuais, ou seja, informações dinâmicas que representam o estado atual de elementos do mundo real (como veículos, semáforos, sensores ambientais etc.) (FIWARE Foundation, 2025d).

O Orion utiliza o padrão Next Generation Service Interface (NGSI), que define como os dados contextuais são representados e manipulados, favorecendo a interoperabilidade entre sistemas e a integração com outras plataformas (FIWARE Foundation, 2025e). Sua principal função é permitir que aplicações e sistemas interajam com informações contextuais de maneira padronizada, utilizando a API NGSI, que define operações para criar, consultar, modificar, atualizar ou excluir entidades e seus atributos (FIWARE Foundation, 2025e, 2025d). Essa abordagem torna possível que diferentes sistemas compartilhem e acessem dados de forma interoperável, independentemente da tecnologia subjacente.

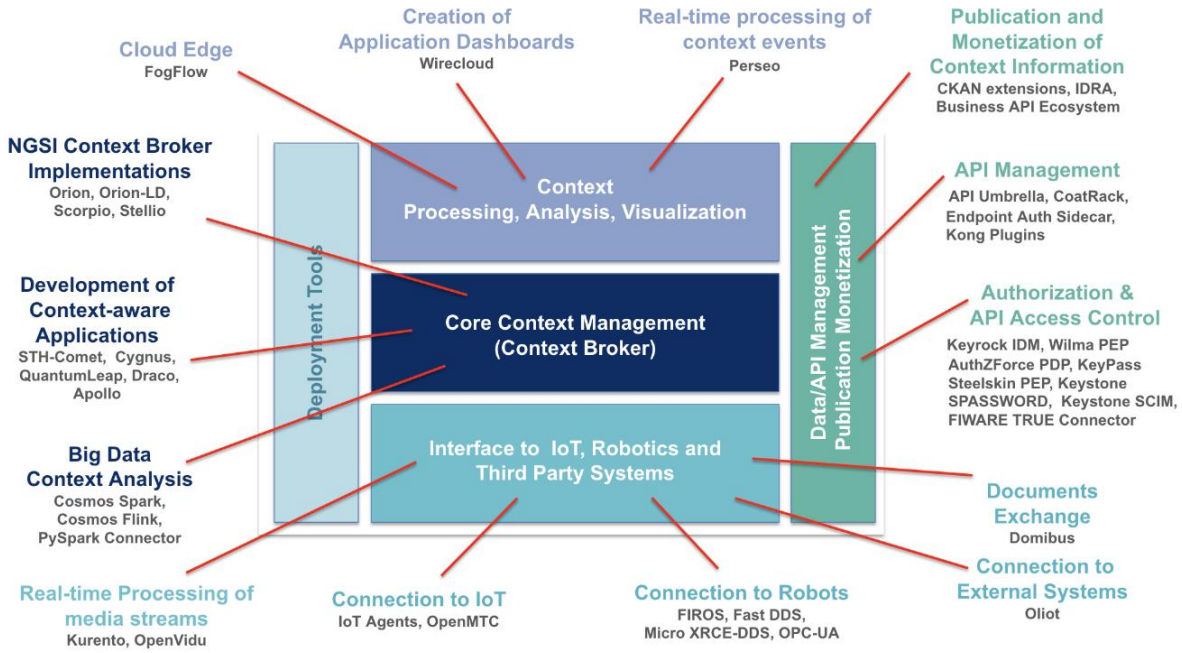
Além disso, o Orion é compatível com arquiteturas de *event-driven* (orientada a eventos), permitindo que aplicações sejam notificadas sempre que uma mudança de contexto ocorre (FIWARE Foundation, 2025d). Isso é essencial para aplicações que exigem reações em tempo real, como controle de tráfego urbano, monitoramento ambiental ou automação industrial (De La Vega *et al.,* 2020; Marquesone *et al.,* 2017; Velasquez; Tobar-Andrade; Cedeno-Campoverde, 2021b). O Orion também pode ser integrado a outros componentes do ecossistema FIWARE. Chamados de Generic Enablers (GEs), componentes reutilizáveis e modulares que oferecem funcionalidades como sistemas de persistência de dados, análise semântica, controle de acesso e interfaces de visualização, que podem ser combinados livremente conforme as necessidades da aplicação ou usuário, formando assim uma arquitetura completa e escalável para soluções inteligentes (FIWARE Foundation, 2025b).

Ademais, a arquitetura do FIWARE é orientada a microsserviços, nos mesmos moldes os quais a plataforma Dojot foi desenvolvida, o que permite flexibilidade, escalabilidade e integração com tecnologias externas, como plataformas de nuvem, bancos de dados, ferramentas de análise e sistemas legados (Bauer, 2022; Llopis *et al.,* 2021). Além disso, o FIWARE promove o uso de open APIs e padrões abertos, o

que facilita a portabilidade e a replicação de soluções entre diferentes domínios ou cidades (FIWARE Foundation, 2025a).

A Figura 35 ilustra a arquitetura funcional do FIWARE com o Orion Context Broker e GEs.

Figura 35 – Arquitetura do FIWARE.



Fonte: FIWARE Foundation (2025).

O Orion implementa o modelo de entidades, no qual cada elemento do mundo real (como um sensor de temperatura ou um semáforo inteligente) é representado por uma entidade com atributos que descrevem seu estado atual. Esses atributos podem ser atualizados dinamicamente à medida que os dados são recebidos, e qualquer sistema interessado pode consultar ou assinar notificações para receber atualizações automaticamente.

Para criar ou atualizar uma entidade no Orion utiliza-se a API NGSI por meio do método HTTP POST, que permite registrar novas entidades ou modificar atributos existentes. Cada entidade deve ter um identificador único seguindo o padrão "urn:ngsi-ld:entity-type:id", o que facilita a interoperabilidade entre aplicações e garante unicidade global. Além do ID, as entidades incluem atributos que descrevem características como o *type*, que define o modelo semântico (por exemplo, TemperatureSensor ou MotionDetector), variáveis contextuais como medições ambientais, valores elétricos ou estados operacionais, além de *timestamps* como

“dateCreated” e “dateModified” e outros metadados, como unidades de medida ou localização geográfica. Essa estrutura flexível permite representar tanto dispositivos físicos (sensores, atuadores, máquinas) quanto entidades virtuais ou lógicas, habilitando funcionalidades avançadas de monitoramento, automação e análise em aplicações de cidades inteligentes, indústria 4.0, energia, entre outras.

A Figura 36 ilustra um exemplo de entidade do Orion Context Broker.

Figura 36 – Exemplo de entidade do Orion Context Broker.

```
{
  "id": "Sensor:Temp:001",
  "type": "TemperatureSensor",
  "temperature": {
    "value": 23.5,
    "type": "Number",
    "metadata": {
      "unit": {
        "value": "Celsius",
        "type": "String"
      }
    }
  },
  "timestamp": {
    "value": "2025-05-12T10:15:30Z",
    "type": "DateTime"
  },
  "location": {
    "value": {
      "type": "Point",
      "coordinates": [-46.633309, -23.550520]
    },
    "type": "geo:json"
  }
}
```

Fonte: Elaborado pelo autor (2025).

Com base nas características exploradas, o FIWARE figura como uma plataforma robusta e flexível para aplicações inteligentes, promovendo a padronização, o reuso e a interoperabilidade de componentes e dados — elementos essenciais para o avanço da transformação digital em diversos setores — por meio do seu gestor de contexto e dos habilitadores genéricos. Nesse contexto, destaca-se o Orion Context Broker, um *middleware* orientado a contexto que facilita a integração, o compartilhamento e o gerenciamento de dados dinâmicos em sistemas distribuídos. Atuando como um componente central da arquitetura FIWARE, o Orion é fundamental para a construção de aplicações baseadas em dados em tempo real, com foco em interoperabilidade, escalabilidade e padronização.

3.2.3. NODE-RED

Desenvolvido originalmente pela International Business Machines Corporation (IBM), o Node-RED é uma ferramenta *open source* para IoT baseada em fluxos, projetada para facilitar a integração entre dispositivos, serviços e aplicações, especialmente no contexto de automação e sistemas distribuídos , por meio de um ambiente *web* (OpenJS Foundation, 2025).

A ferramenta, que surgiu como uma prova de conceito para visualizar e manipular mapeamentos entre tópicos MQTT, rapidamente se tornou uma ferramenta mais geral para a criação de aplicações leves e orientados a eventos. Sua principal característica é a interface de Programação Orientada a Fluxo, paradigma que fundamenta o funcionamento da ferramenta, em que blocos funcionais (chamados de nós) são conectados em sequência para representar o caminho dos dados desde sua origem (como sensores ou APIs) até seu destino (como bancos de dados, atuadores ou *dashboards*). Esses nós representam funções como entrada e saída de dados, processamento, transformação, comunicação via protocolos (HTTP, MQTT, WebSocket etc.) e integração com serviços externos (Hagino; O'Leary, 2021; OpenJS Foundation, 2025).

3.2.3.1. ARQUITETURA E FLUXOS

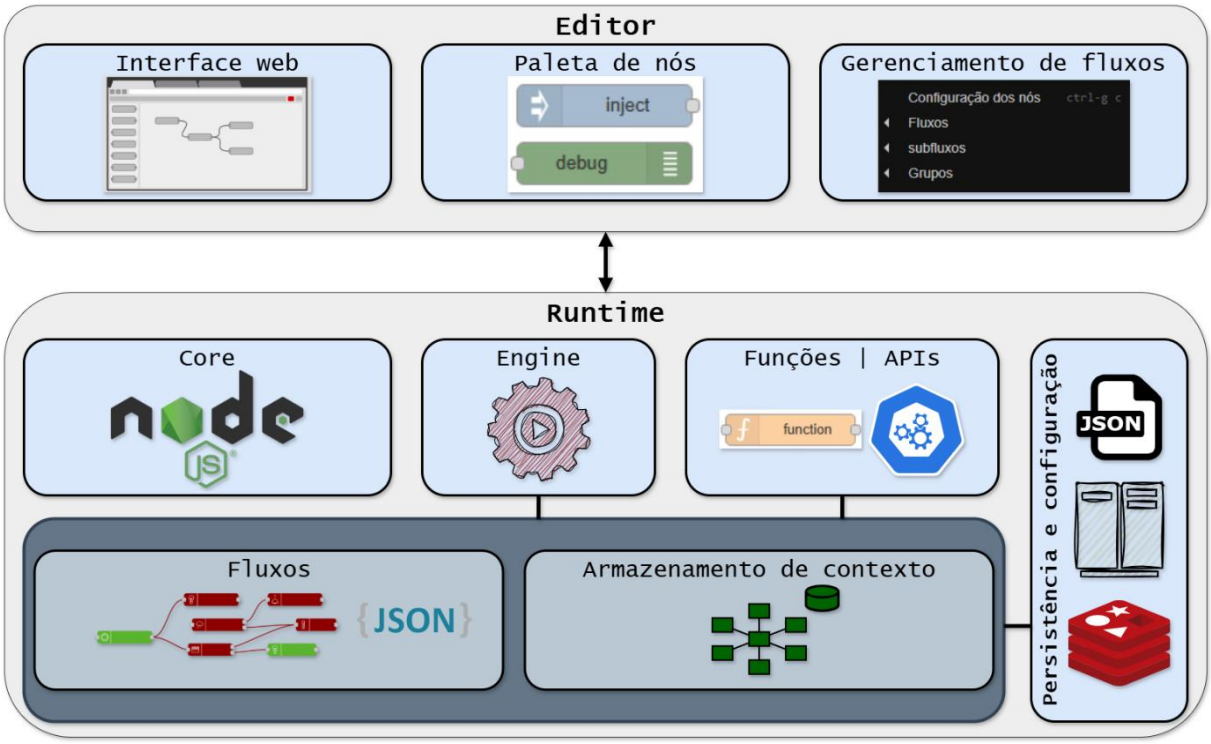
A arquitetura do Node-RED é organizada em dois grandes módulos — Editor e Runtime — apoiados por diversos componentes auxiliares que garantem sua modularidade, extensibilidade e persistência (Figura 37).

O Editor corresponde à interface de desenvolvimento baseada em navegador, que se executa sobre um servidor HTTP interno (utilizando o Express.js). Nele, o usuário conta com uma interface gráfica intuitiva para arrastar, conectar e configurar nós, construindo fluxos, os quais são responsáveis por instanciar objetos similares aos dispositivos da plataforma Dojot e as entidades do Orion[11]. A paleta de nós, por sua vez, exibe dinamicamente os módulos instalados, os quais podem ser expandidos via

[11] Embora recebam nomenclaturas distintas, todas essas instâncias cumprem o mesmo propósito dentro do escopo da pesquisa. Para padronizar a terminologia metodológica, adotou-se o termo "dispositivos virtuais" para se referir a todas as instâncias responsáveis por representar (ou "geminar") dispositivos físicos utilizados no desenvolvimento da metodologia e dos resultados.

NPM, viabilizando suporte a novos protocolos e serviços. Além disso, o editor permite a organização dos fluxos em múltiplas abas, criação de subfluxos e validação de configurações antes da implantação (Hagino; O'Leary, 2021).

Figura 37 – Arquitetura do Node-RED.

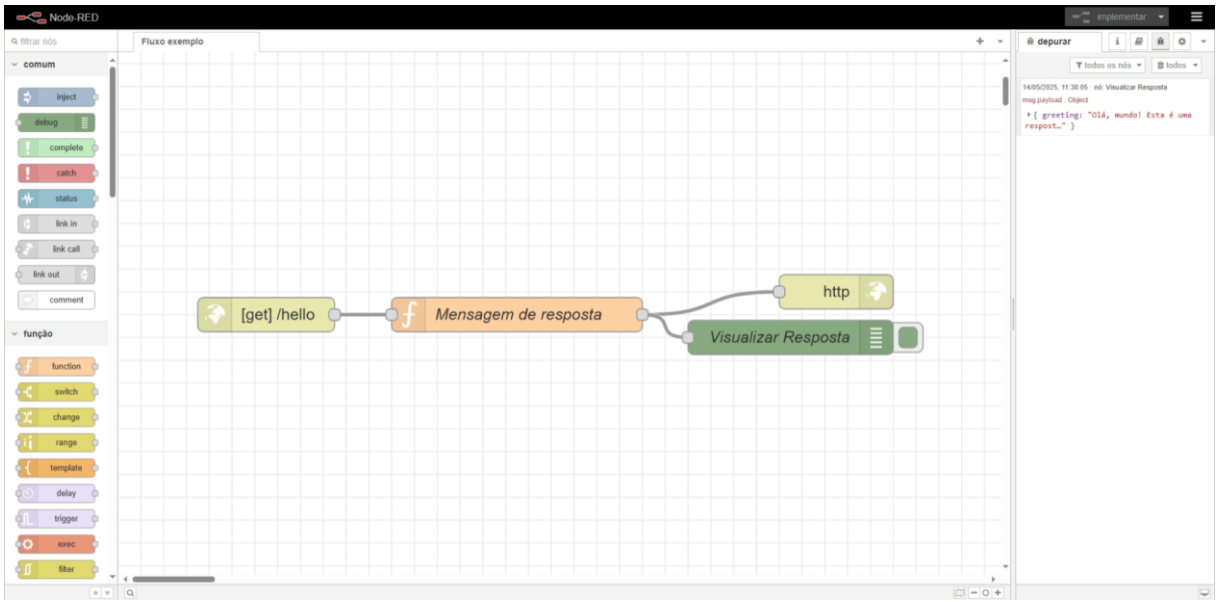


Fonte: Elaborado pelo autor (2025).

Esse ambiente de desenvolvimento visual (Figura 38) facilita não apenas a prototipagem rápida, mas também a compreensão do funcionamento do sistema, mesmo para quem tem pouca experiência em programação, sendo uma das diversas vantagens que o paradigma orientado a fluxos no Node-RED traz (Hagino; O'Leary, 2021; Rattanapoka *et al.,* 2019). Não obstante, a modularidade é uma das principais dessas vantagens: cada nó realiza uma tarefa isolada e bem definida, o que torna o sistema mais organizado, reutilizável e fácil de manter (Hagino; O'Leary, 2021). Além disso, o modelo é orientado a eventos, ou seja, os nós são ativados à medida que recebem dados, o que permite reações em tempo real — uma característica muito importante em sistemas IoT e automação (Bin Mohd Nazri *et al.,* 2020; Kopjak; Sebestyen, 2019; Roslan *et al.,* 2024). Outra vantagem significativa é a desacoplagem funcional: os nós são independentes e podem ser modificados ou substituídos sem comprometer todo o fluxo. Também é possível incluir trechos de código JavaScript personalizado quando necessário, o que amplia a flexibilidade da ferramenta (Hagino;

O'Leary, 2021). Por fim, o modelo oferece transparência no desenvolvimento, já que os fluxos visuais permitem identificar facilmente a lógica da aplicação e o caminho que os dados percorrem.

Figura 38 – Exemplo de fluxo no Node-RED.



Fonte: Elaborado pelo autor (2025).

O Runtime, por sua vez, representa o mecanismo de execução dos fluxos. É construído sobre o Node.js e tem como responsabilidade interpretar os arquivos JSON criados pelo editor, instanciando cada nó conforme sua definição. A *engine* de mensagens gerencia o tráfego interno entre os nós de forma assíncrona, ativando cada nó ao receber uma nova mensagem. O Runtime também executa funções personalizadas escritas em JavaScript nos nós do tipo "*function*", utilizando um ambiente isolado (sandbox). Além disso, expõe APIs REST que permitem automação e monitoramento do ambiente de execução, como o *deploy* de fluxos e o controle de nós (Hagino; O'Leary, 2021).

Para garantir a persistência, toda a configuração dos fluxos é armazenada em arquivos JSON, como o "flows_<hostname>.json", o que permite versionamento e *backup*. A *engine* ainda oferece um mecanismo de armazenamento de contexto (*context store*), que pode utilizar memória volátil, arquivos locais ou até bancos como Redis para persistir variáveis entre execuções (Hagino; O'Leary, 2021).

Outro ponto forte da arquitetura são os módulos de integração, integrados e gerenciados por meio do Palette Manager, que incluem conectores para protocolos

como MQTT, HTTP, WebSockets, TCP/UDP, Modbus e OPC-UA, além de nós de saída para bancos de dados, serviços em nuvem e *dashboards*. A arquitetura também contempla recursos de extensibilidade e segurança, como o gerenciamento seguro de credenciais e controladores de acesso baseados em autenticação (LDAP, OAuth2) e autorização granular. (Hagino; O'Leary, 2021)

Em conjunto, essa arquitetura permite ao Node-RED oferecer um ambiente de desenvolvimento rápido e colaborativo por meio do editor, ao mesmo tempo que mantém um Runtime leve e escalável. Essa separação clara entre criação e execução torna o Node-RED adequado tanto para aplicações em dispositivos embarcados quanto em ambientes de produção em larga escala.

Sintetizando, entre os principais recursos e vantagens do Node-RED, destacam-se:

- facilidade de uso: permite que desenvolvedores e até usuários sem formação técnica criem fluxos de dados com pouco ou nenhum código;
- alta extensibilidade: possui uma vasta biblioteca de nós prontos para uso e permite a criação de nós personalizados em JavaScript;
- suporte a múltiplos protocolos IoT: como MQTT, CoAP, Modbus, HTTP, entre outros;
- execução local ou em nuvem: pode ser instalado em computadores, servidores, ou em dispositivos embarcados como o Raspberry Pi;
- integração com plataformas de IoT: pode ser usado em conjunto com plataformas como FIWARE, Dojot, AWS IoT, Google Cloud IoT etc.; e
- *dashboard* embutido: permite a criação de interfaces gráficas simples para visualização de dados em tempo real;

3.3. ENGENHARIA DE CONFIABILIDADE DO SITE, OBSERVABILIDADE E MONITORAMENTO

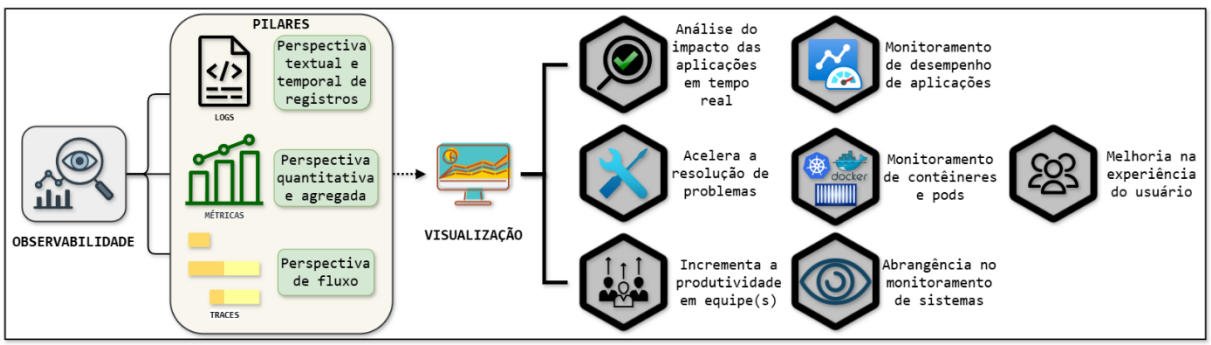
A Engenharia de Confiabilidade do Site (SRE), a observabilidade e o monitoramento são práticas interconectadas que visam garantir a confiabilidade, a escalabilidade e a eficiência de sistemas complexos (Beyer *et al.,* 2016). A SRE

consiste em uma disciplina que combina engenharia de *software* e operações de TI, concentrando-se na criação de sistemas altamente confiáveis e eficientes, aproveitando a automação, a melhoria contínua e a resposta proativa a incidentes (Beyer *et al.,* 2016, 2018).

No centro da SRE, representando um pilar fundamental, está o conceito de observabilidade, que se refere à capacidade de medir o estado interno de um sistema por meio de suas saídas externas, baseando-se em três pilares fundamentais: *logs* (ou registros), métricas e *traces* (ou rastreamentos) (Figura 39) (Beyer *et al.,* 2016; Gomes; Rego; Trinta, 2024; Lingamallu; De Oliveira, 2023; Team, 2024). Esses elementos fornecem informações sobre o comportamento do sistema, permitindo que os engenheiros identifiquem problemas, depurem problemas e otimizem o desempenho.

Os *logs* são registros detalhados de eventos que ocorrem em um sistema, os quais fornecem um relato cronológico das atividades, erros e exceções do sistema, sendo particularmente úteis para depurar e entender o contexto de eventos específicos (Sun *et al.,* 2023). Portanto, são sendo indispensáveis para a análise da causa raiz, pois fornecem informações detalhadas sobre eventos específicos, como mensagens de erro, exceções e mudanças no estado do sistema. Nas arquiteturas de microsserviços, os registros podem ser gerados em várias camadas, incluindo registros de aplicativos, registros de banco de dados e registros de rede. O *log* estruturado, que organiza os dados de registro em um formato padronizado, aprimora a capacidade de pesquisar, filtrar e analisar registros com eficiência (Boten; Majors, 2022; Lingamallu; De Oliveira, 2023; Sun *et al.,* 2023).

Figura 39 – Pilares e benefícios da observabilidade.



Fonte: Elaborado pelo autor (2025).

As métricas são medidas numéricas que quantificam o desempenho e a integridade do sistema, incluindo, dentre as mais comuns, latência de solicitação, taxas de erro, uso da CPU e consumo de memória (Beyer *et al.,* 2016; Boten; Majors, 2022; Majors; Fong-Jones; Miranda, 2022). Elas permitem a criação de sistemas de monitoramento por meio da definição de limites e alertas, garantindo que possíveis problemas sejam identificados antes que sejam agravado ao aplicar medidas proativas para evitar interrupções. Além disso, as métricas também são essenciais para a definição de Acordos de Nível de Serviço (SLA), Objetivos de Nível de Serviço (SLO), Indicadores de Nível de Serviço (SLI) e Requisitos Não Funcionais (NFR) (Beyer *et al.,* 2016, 2018).

Os *traces* fornecem uma visão detalhada da jornada de uma solicitação por meio de um sistema distribuído, capturando todo o fluxo de uma transação, incluindo os serviços envolvidos, o tempo gasto em cada etapa e os erros encontrados, sendo particularmente valiosos em arquiteturas de microsserviços, nas quais as solicitações geralmente passam por vários serviços (Sharma; Nadig, 2024; Sun *et al.,* 2023). Ferramentas de rastreamento distribuídas, como Jaeger ou Zipkin, ajudam a visualizar e analisar *traces*, auxiliando a identificação de identificar gargalos de desempenho, picos de latência e serviços propensos a erros (Kosińska *et al.,* 2023; Lingamallu; De Oliveira, 2023).

Portanto, *logs*, métricas e *traces* são componentes essenciais da observabilidade nas práticas de SRE, pois cada um fornece informações exclusivas sobre o comportamento do sistema e sua integração permite uma compreensão holística do desempenho e da confiabilidade do sistema. Ao aproveitar o *log* estruturado, a coleta de métricas e o *tracing* distribuído, as equipes de SRE podem melhorar a detecção de anomalias, a localização de falhas e a solução de problemas.

Por outro lado, o monitoramento, embora relacionado, consiste mais em observar ativamente os sistemas para garantir que eles atendam a critérios predefinidos, geralmente por meio de métricas e alertas predefinidos (Beyer *et al.,* 2016; Team, 2024). Os sistemas de monitoramento no SRE normalmente envolvem a coleta e a análise das principais métricas, como SLA, Objetivos de Nível de Serviço SLO, SLI e NFR, as quais ajudam a definir os limites de confiabilidade e desempenho de um sistema (Adkins *et al.,* 2020; Beyer *et al.,* 2016, 2018). Os sistemas avançados de monitoramento também incorporam automação e análise em tempo real para

identificar e resolver problemas de forma proativa antes que eles afetem os usuários finais (Hasan; Abdullah, 2022; Korontanis *et al.,* 2023; Thantharate, 2023).

Dessa forma, SRE, observabilidade e monitoramento estão profundamente interligados. A SRE depende da observabilidade para obter *insights* sobre o comportamento do sistema, permitindo a detecção e resolução proativas de problemas. A observabilidade, por sua vez, se baseia no monitoramento, fornecendo uma compreensão mais abrangente dos componentes internos do sistema, indo além do foco do monitoramento tradicional em métricas predefinidas. Por último, o monitoramento serve como uma ferramenta fundamental para SRE e observabilidade, fornecendo os dados necessários para análise e tomada de decisões.

Em síntese, como as soluções IoT se tornaram componentes essenciais dos sistemas de monitoramento modernos, os dispositivos IoT geram grandes quantidades de dados que podem ser analisados em tempo real para monitorar o desempenho do sistema, detectar anomalias e prever possíveis falhas. Dessa forma, garantir a confiabilidade nesses ambientes é um desafio devido à natureza distribuída desses sistemas, ao grande volume de dados gerados e à necessidade de processamento em tempo real. Além disso, como os sistemas de IoT geralmente são implantados em ambientes hostis, o que pode afetar a precisão do sensor e o desempenho do sistema. Logo, enfrentar esses desafios requer recursos robustos de monitoramento e diagnóstico.

3.3.1. STACK DE OBSERVABILIDADE E MONITORAMENTO

Dentre as tecnologias utilizadas dentro da SRE para fins de observabilidade e monitoramento, são alternativas aplicadas no desenvolvimento da pesquisa: InfluxDB, Prometheus, cAdvisor e Grafana. Conjuntamente, as ferramentas compõem a Stack elaborado para a coleta, armazenamento, processamento e visualização de dados referentes às métricas de desempenho dos ambientes de desenvolvimento implantados.

3.3.1.1. INFLUXDB

O primeiro elemento da *Stack* é o InfluxDB, que se trata de um banco de dados de séries temporais otimizado para lidar com dados de alto volume e alta velocidade.

Ele é particularmente adequado para aplicativos que exigem armazenamento e recuperação eficientes de dados com data e hora, como telemetria de IoT, análise financeira e monitoramento de sistemas (Beermann *et al.,* 2020; Correia; Oliveira; Cecílio, 2023; Giacobbe *et al.,* 2020).

No InfluxDB os dados são organizados em medidas (*measurements*), que funcionam como tabelas; *tags*, que são pares chave-valor usados para indexação e filtragem eficiente; campos (*fields*), que contêm os valores das métricas; e *timestamps*, que marcam o momento exato em que o dado foi registrado. Esse modelo permite consultas rápidas e altamente granulares sobre grandes volumes de dados com precisão temporal (InfluxData, 2025).

Para armazenamento, emprega o mecanismo *Time-Structured Merge* (TSM) que é projetado para gravações rápidas e compactação eficiente dos dados históricos e permite uma arquitetura de cluster com separação entre armazenamento e computação, favorecendo a escalabilidade e permitindo que o banco armazene milhões de pontos de dados com baixo consumo de recursos e boa performance de leitura e escrita (InfluxData, 2025). Ele também oferece políticas de retenção e *downsampling*, permitindo que dados antigos sejam resumidos ou descartados automaticamente, conforme necessário (InfluxData, 2025).

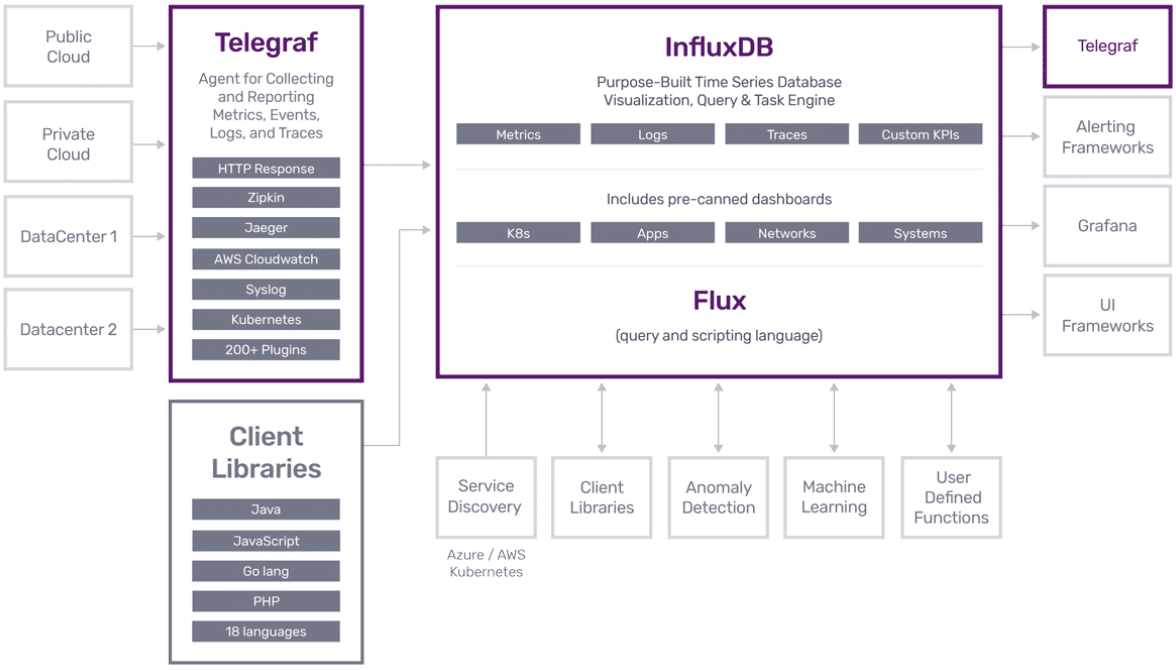
Outro destaque do InfluxDB é sua linguagem de consulta própria, chamada Flux, a qual oferece recursos avançados de análise, como agregações, filtragens, transformações de dados, união de séries temporais e até integração com outras fontes de dados externas (InfluxData, 2025).

O Flux substitui a antiga InfluxQL, oferecendo maior flexibilidade para análises complexas, como combinar dados de múltiplas fontes e aplicar funções avançadas. Essa evolução permite criar pipelines dinâmicos e visualizações diretamente nas consultas. Com isso, o InfluxDB se torna ainda mais robusto para monitoramento, séries temporais e análises preditivas.

Além disso, o InfluxDB pode ser executado de forma independente ou como parte de uma solução integrada com Telegraf, Chronograf e Kapacitor (InfluxData, 2025). Embora essas ferramentas complementares existam, hoje o InfluxDB também se integra bem com o Grafana, que é frequentemente usado para visualizar os dados armazenados no banco (Beermann *et al.,* 2020).

A Figura 40 ilustra uma visão geral da arquitetura e funcionalidades do InfluxDB.

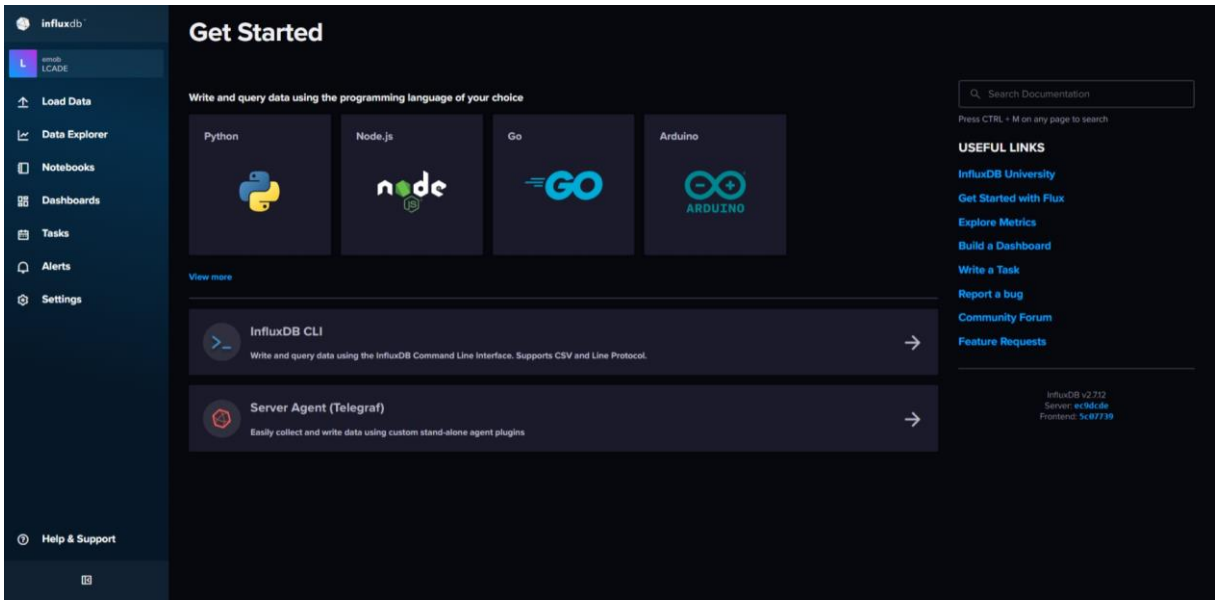
Figura 40 – Visão geral do InfluxDB.



Fonte: InfluxData (2025).

Em sua versão mais recente (InfluxDB 2.x), a plataforma passou a oferecer uma interface *web* completa, APIs unificadas e uma abordagem "tudo em um", incluindo coleta, visualização e alertas, além de suporte a armazenamento em nuvem (Figura 41). Isso torna o InfluxDB uma solução poderosa e escalável para aplicações modernas que demandam análise e monitoramento em tempo real (InfluxData, 2025).

Figura 41 – Interface web do InfluxDB versão 2.x.



Fonte: Elaborado pelo autor (2025).

3.3.1.2. PROMETHEUS

O Prometheus é um poderoso kit de ferramentas de monitoramento e alerta de código aberta, destacando-se na coleta de métricas de sistemas distribuídos, projetado para operar em infraestruturas de grande escala. Uma de suas principais características é o modelo de dados baseado em séries temporais, onde cada métrica é identificada por um nome e um conjunto de pares chave-valor chamados de *labels*, o que permite consultas altamente flexíveis e segmentadas. As métricas coletadas são armazenadas localmente em um banco de dados próprio e eficiente, utilizando arquivos em formato *Time Series Database* (TSDB), o que elimina a necessidade de dependências externas para armazenamento (Prometheus Authors, 2025).

A ferramenta utiliza uma abordagem de coleta do tipo "*pull*", em que periodicamente requisita dados dos alvos de monitoramento por meio de *endpoints* HTTP, facilitando a descoberta de serviços em ambientes dinâmicos e reduzindo a complexidade de configuração dos exportadores. Todavia, também há suporte para coleta do tipo "*push*", por meio do *Pushgateway*, voltado para casos em que o modelo *pull* não é aplicável, como em *jobs* de curta duração (Prometheus Authors, 2025).

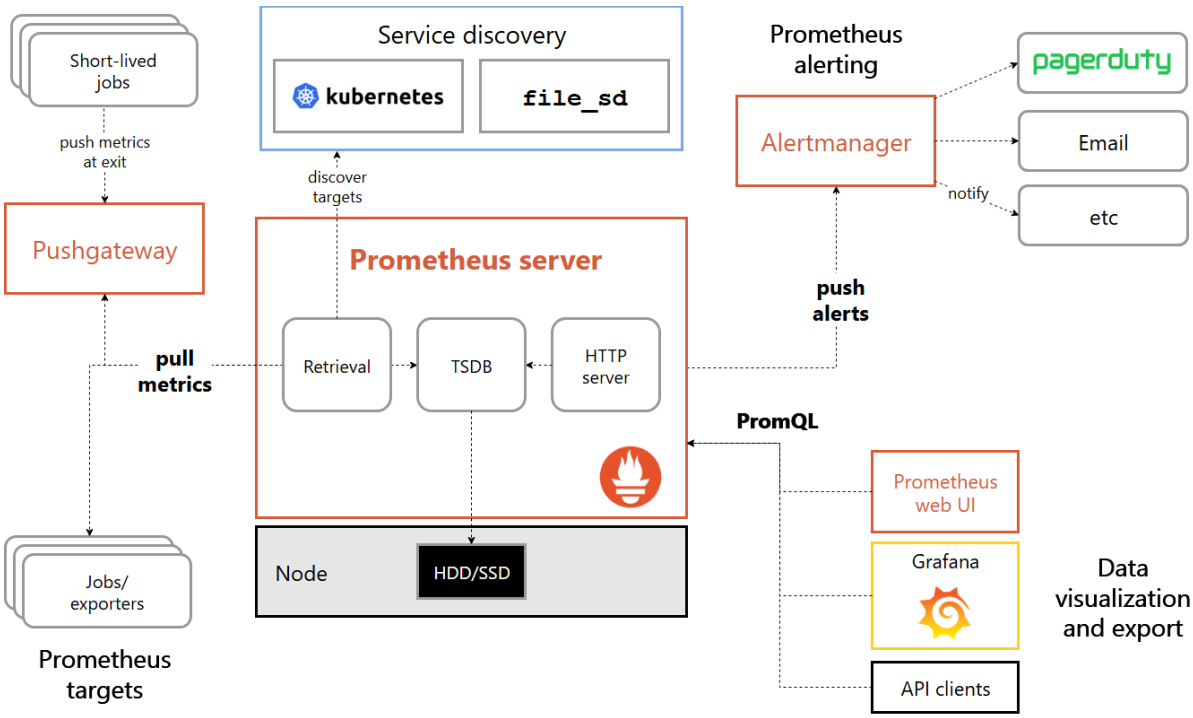
Outro diferencial técnico é a linguagem de consulta *Prometheus Query Language* (PromQL), projetada especificamente para lidar com métricas temporais. Ela permite criar consultas complexas para análise, agregação, filtragem e geração

de alertas com base em valores e tendências observadas nas métricas. As consultas PromQL são usadas tanto para visualizações em dashboards quanto para regras de alerta (Prometheus Authors, 2025).

A ferramenta também inclui um sistema interno de alertas, que permite definir condições com base em métricas monitoradas. Quando essas condições são atendidas, os alertas são encaminhados para o Alertmanager, que gerencia o roteamento, agrupamento, silenciamento e envio de notificações por diferentes canais, como e-mail, Slack e PagerDuty. Sua arquitetura é modular e extensível, com suporte a descoberta automática de serviços (via DNS, Consul, Kubernetes etc.) e uma vasta gama de exportadores para diferentes sistemas e aplicações, como Alertmanager, Grafana, Node Exporter e cAdvisor (Prometheus Authors, 2025).

A Figura 42 apresenta uma visão geral do Prometheus.

Figura 42 – Visão geral do Prometheus.



Fonte: Prometheus Authors (2025).

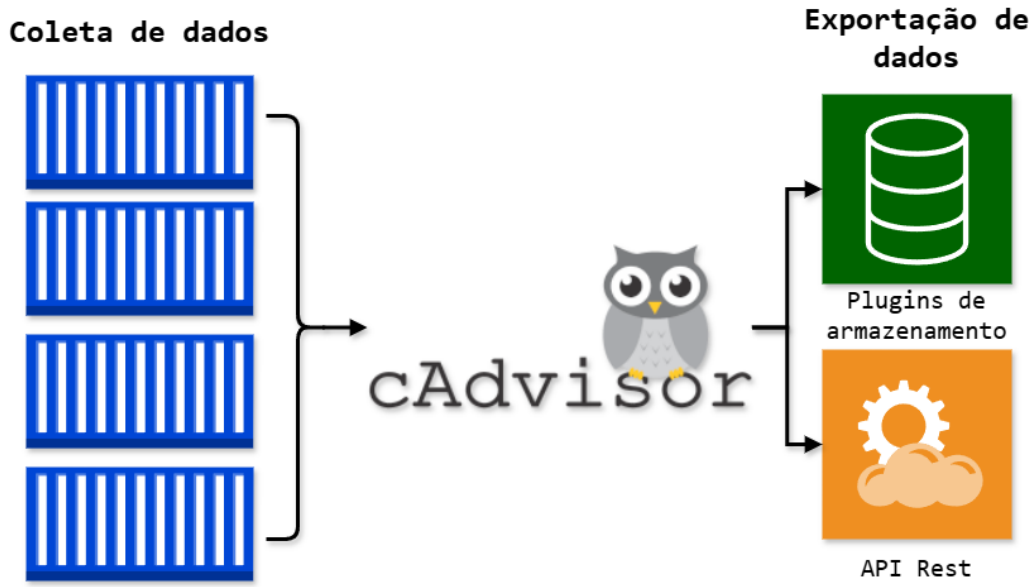
Além disso, o Prometheus pode ser escalado horizontalmente e integrar-se a soluções como Thanos ou Cortex para retenção de longo prazo e alta disponibilidade, estendendo suas capacidades para ambientes corporativos de grande porte, tornando-o substancialmente útil em práticas proativas de monitoramento (Prometheus Authors, 2025).

3.3.1.3. cAdvisor

O cAdvisor (Container Advisor) é uma ferramenta de código aberto desenvolvida pelo Google, projetada para monitorar o desempenho e o uso de recursos de contêineres, especialmente os que utilizam o Docker. Ele coleta e exporta métricas relacionadas ao uso de CPU, memória, rede e disco dos contêineres em tempo real, sendo amplamente utilizado em conjunto com o Prometheus para observabilidade de ambientes baseados em contêineres (Google, 2024).

Tecnicamente, o cAdvisor atua como um *daemon* que roda no *host* e inspeciona continuamente os contêineres em execução. Ele coleta estatísticas diretamente do *Control Groups* (cgroup) e das interfaces de sistema do Linux, o que permite uma visão detalhada e precisa dos recursos consumidos por cada contêiner (Figura 43). Cada instância do cAdvisor oferece um *endpoint* HTTP com as métricas no formato compreendido pelo Prometheus (Google, 2024).

Figura 43 – Funcionamento do cAdvisor.



Fonte:Elaborado pelo autor (2025).

Além da exportação de métricas, o cAdvisor também fornece uma interface *web* própria para visualização básica dos dados de desempenho dos contêineres, embora seja comum desativá-la em ambientes de produção por motivos de segurança ou desempenho (Google, 2024).

Entre as principais métricas coletadas pelo cAdvisor, estão:

- Uso de CPU total e por núcleo;
- Uso de memória (RSS, *cache*, limites);
- I/O de disco e rede;
- Tempo de vida dos contêineres;
- Informações sobre o contêiner *runtime*.

O cAdvisor é particularmente útil em clusters Kubernetes, já que cada nó geralmente executa uma instância do cAdvisor por padrão como parte do *kubelet*, embora em ambientes de produção o uso de soluções mais robustas (como o *kubelet* + *kube-state-metrics*) seja recomendado para uma cobertura mais ampla (Google, 2024; Tolaram, 2023).

Dessa forma, o cAdvisor representa uma solução leve e eficiente para coletar métricas de baixo nível de contêineres Docker ou *pods* Kubernetes, sendo uma peça importante na pilha de monitoramento, especialmente em fases iniciais de implantação ou em ambientes com foco em desempenho granular de contêineres.

3.3.1.4. GRAFANA

Finalizando a Stack, há o Grafana, que se trata uma ferramenta de visualização de código aberto que se destaca na criação de painéis para análise de dados em tempo real. Ele se integra perfeitamente a várias fontes de dados, incluindo InfluxDB e Prometheus, para exibir métricas e registros em um formato fácil de usar.

Sua flexibilidade e compatibilidade com diversas ferramentas o tornaram um padrão de mercado, especialmente quando combinado com sistemas como Prometheus, InfluxDB, Elasticsearch, Zabbix entre outros, permitindo o monitoramento de aplicações de naturezas variadas (Beermann *et al.,* 2020; Grafana Labs, 2025; Mehdi *et al.,* 2023).

Do ponto de vista técnico, o Grafana é escrito principalmente em Go (*backend*) e TypeScript/React (*frontend*), funcionando como um servidor *web* que permite aos usuários se conectarem a fontes de dados externas, configurar painéis, compartilhar visualizações e definir alertas (Chakraborty; Kundan, 2021; Grafana Labs, 2025). Os painéis são compostos por painéis individuais (gráficos, tabelas, mapas etc.) que

podem ser configurados com consultas dinâmicas e filtros interativos (Grafana Labs, 2025; Mehdi *et al.,* 2023).

Uma de suas maiores vantagens é o suporte a múltiplas fontes de dados simultaneamente, com conectores nativos para Prometheus, InfluxDB, Loki, Elasticsearch, MySQL, PostgreSQL, Graphite e muitos outros (Chakraborty; Kundan, 2021; Grafana Labs, 2025; Walter-Tscharf, 2021). Isso permite, por exemplo, combinar métricas de desempenho, logs e dados de aplicações em um único painel, promovendo uma visão unificada do ambiente monitorado.

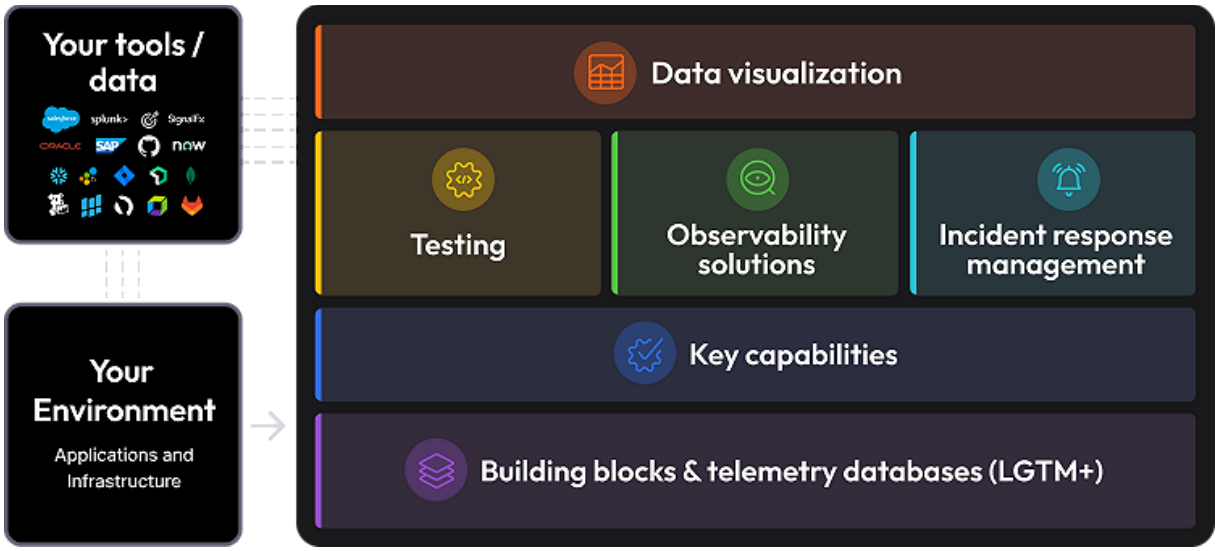
Além disso, o Grafana também oferece um sistema robusto de alertas, que permite configurar condições baseadas em métricas para acionar notificações via e-mail, Slack, PagerDuty, Microsoft Teams, entre outros (Chakraborty; Kundan, 2021; Grafana Labs, 2025). Esses alertas podem ser gerenciados diretamente na interface e são úteis para detectar anomalias e reagir proativamente a falhas ou degradações de serviço .

Outro destaque é a vasta comunidade de usuários e desenvolvedores, que mantém uma biblioteca pública de dashboards prontos, plugins e integrações com diversas ferramentas e serviços. Essa base facilita a adoção rápida e a personalização do Grafana em diferentes contextos, desde ambientes locais até infraestruturas de grande escala em nuvem (Beermann *et al.,* 2020; Siddiqui *et al.,* 2023; Walter-Tscharf, 2021).

Por fim, o Grafana está disponível em diferentes edições: Grafana OSS (*open source),* Grafana Enterprise (com recursos adicionais para grandes organizações) e Grafana Cloud, uma solução gerenciada com funcionalidades extras e escalabilidade automática (Grafana Labs, 2025).

A Figura 44 ilustra uma visão geral das funcionalidades do Grafana.

Figura 44 – Visão geral das funcionalidades do Grafana.



Fonte: Grafana Labs (2024).

3.4. GRAFANA K6

Para complementar a avaliação de desempenho e escalabilidade das arquiteturas IoT desenvolvidas neste trabalho, foi adotado o Grafana k6, uma ferramenta de código aberto voltada para testes de carga e desempenho, mantida pela Grafana Labs. Seu principal objetivo é simular múltiplos usuários acessando aplicações *web* ou APIs simultaneamente, por meio de unidades virtuais (VUs), permitindo avaliar a robustez dos sistemas sob diferentes níveis de estresse.

Os testes são definidos por meio de *scripts* em JavaScript (ou TypeScript), oferecendo flexibilidade na criação de cenários personalizados. Durante sua execução, o k6 coleta métricas fundamentais como tempo de resposta, taxa de requisições por segundo, latência, *throughput* e taxa de erros, fornecendo uma visão detalhada sobre o comportamento da aplicação sob carga.

O k6 também permite a integração de seus resultados com *dashboards* do Grafana, proporcionando uma experiência rica de visualização e análise de desempenho. Ao transmitir métricas para *backends* compatíveis—como InfluxDB, Prometheus, TimescaleDB, AWS Timestream—é possível montar painéis em tempo real que correlacionam os resultados de testes de carga com métricas do sistema, facilitando a identificação imediata de gargalos e padrões de comportamento, além de possibilitar a automação dos testes em pipelines de CI/CD.

O Quadro 4 apresenta as principais métricas coletadas pelo k6.

Quadro 4 – Principais métricas coletadas pelo k6.

Métrica	Descrição
http_reqs	Número total de requisições HTTP realizadas durante o teste.
http_req_duration	Tempo total de duração das requisições HTTP (latência total).
http_req_connecting	Tempo gasto na fase de conexão TCP com o servidor.
http_req_waiting	Tempo de espera pelo primeiro byte da resposta (TTFB – Time To First Byte).
http_req_receiving	Tempo de recebimento dos dados da resposta após o TTFB.
http_req_blocked	Tempo em que a requisição ficou bloqueada (ex.: filas, DNS, etc.).
vus	Número de usuários virtuais ativos no momento.
vus_max	Número máximo de usuários virtuais durante a execução.
checks	Total de verificações realizadas e taxa de sucesso (assertions).
data_received	Volume total de dados recebidos pelas requisições (em bytes).
data_sent	Volume total de dados enviados pelas requisições (em bytes).
iteration_duration	Duração total de uma iteração do script por VU.
iterations	Número total de iterações executadas no teste.
failures (personalizada)	Número de falhas, caso definidas via check() ou lógica personalizada.

Fonte: Grafana Labs (2024).

Graças à sua leveza, flexibilidade e facilidade de integração, o k6 é amplamente utilizado em ambientes distribuídos e aplicações IoT, apresentando-se como uma solução robusta para *benchmarking* e observabilidade em testes complexos e replicáveis.

3.5. CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo conhecemos as técnicas e tecnologias utilizadas neste trabalho, bem como importantes conceitos e definições de termos por cada uma. Além disso, apresentamos uma visão geral das funcionalidades e propósitos de cada ferramenta ou plataforma, destacando suas licenças e repositórios oficiais para facilitar futuras consultas e aprofundamentos. Essa base teórica é fundamental para compreender a integração entre os componentes do sistema e a forma como eles colaboram para atender aos objetivos propostos.

Adicionalmente, discutimos aspectos como a escalabilidade, desempenho e observabilidade das soluções adotadas, evidenciando a importância da escolha criteriosa das ferramentas para garantir a robustez e eficiência da arquitetura desenvolvida. Essa fundamentação técnica permite ainda identificar potenciais

limitações e oportunidades de melhoria, orientando futuras implementações e estudos no contexto das arquiteturas IoT.

Sintetizando o conteúdo do capítulo, o Quadro 5 apresenta um resumo das tecnologias empregadas no estudo.

Quadro 5 – Resumo das tecnologias apresentadas.

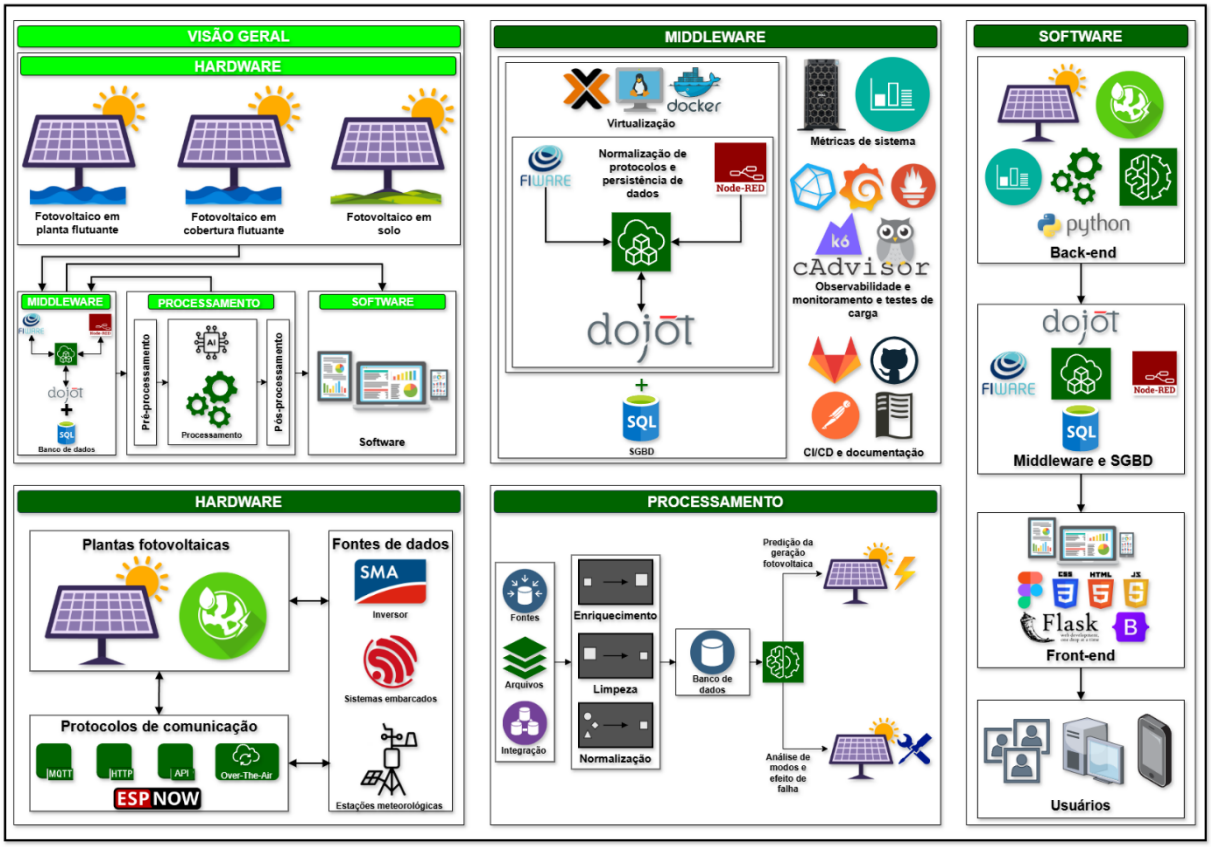
Software	**Licença**	**Função**	**Repositório Oficial**
Proxmox VE	AGPLv3	Plataforma de virtualização e gerenciamento de contêineres LXC e VMs	git.proxmox.com
Docker	Apache 2.0	Plataforma para criação, distribuição e execução de contêineres	github.com/moby/moby
Docker Compose	Apache 2.0	Ferramenta para definir e gerenciar aplicações multi-contêiner	github.com/docker/compose
InfluxDB	MIT (versão OSS)	Banco de dados séries temporais para armazenamento de métricas	github.com/influxdata/influxdb
Prometheus	Apache 2.0	Sistema de monitoramento e alerta baseado em séries temporais	github.com/prometheus/prometheus
cAdvisor	Apache 2.0	Coleta métricas de uso de recursos e performance de containers	github.com/google/cadvisor
Grafana	AGPLv3	Plataforma para visualização e análise de métricas	github.com/grafana/grafana
Grafana k6	AGPLv3	Ferramenta para teste de desempenho de servidores *web*	github.com/grafana/k6

Fonte: Elaborado pelo autor (2025).

4. *TESTBED* DE ARQUITETURAS IOT PARA APLICAÇÕES DE SISTEMAS INTELIGENTES

No contexto do projeto EMOB-AMAZON, conforme descrito na visão geral desta pesquisa, foi proposta uma arquitetura IoT escalável, composta por quatro camadas principais: camada física, responsável pela coleta de dados por meio de sensores e dispositivos embarcados; camada de *middleware*, que gerencia a comunicação, integração e interoperabilidade entre os dispositivos; camada de processamento, encarregada da análise e tratamento dos dados; e camada de *software*, onde se encontram as interfaces e aplicações finais que consomem os dados processados — conforme ilustrado na Figura 45.

Figura 45 – Visão geral da arquitetura IoT proposta para o EMOB-AMAZON.



Fonte: Elaborado pelo autor (2025).

Por conseguinte, os principais objetos de estudo desta pesquisa são a implantação dos *middlewares* IoT e a construção do *testbed*, que reúne a pilha de monitoramento e os testes necessários para acompanhar e analisar o desempenho das arquiteturas desenvolvidas. Na sequência, são detalhadas as funções específicas

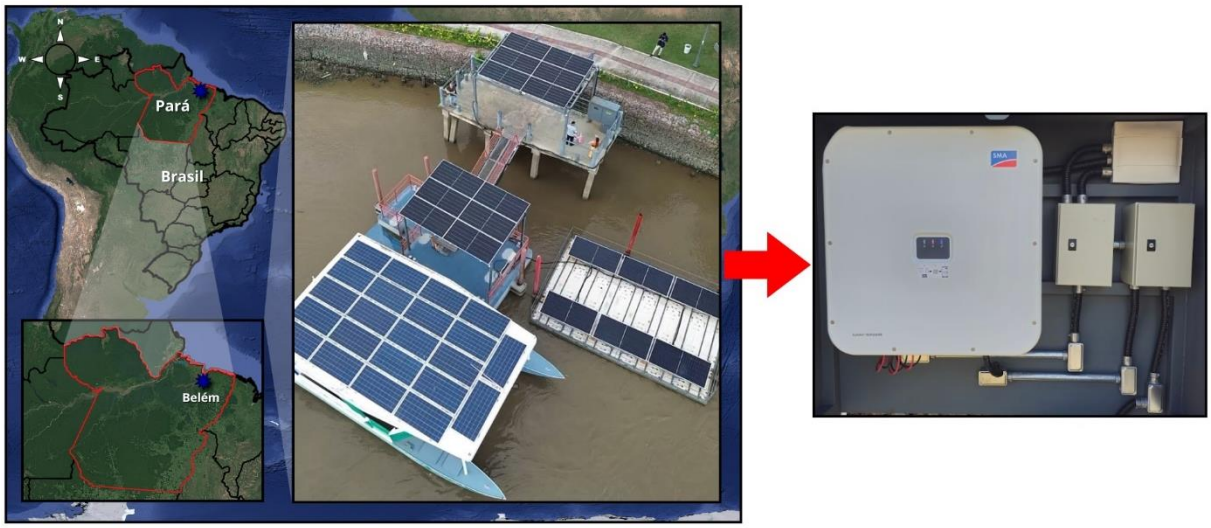
de cada uma das camadas que compõem essa arquitetura, destacando sua relevância para a implementação e operação do ambiente IoT proposto.

4.1. CAMADA FÍSICA

A camada física representa os dispositivos físicos (fontes de dados), ou seja, três arranjos fotovoltaicos e seus inversores comunicativos, sistemas embarcados e estações meteorológicas. O sistema fotovoltaico projetado, que representa a fonte de dados primária da pesquisa, consiste em um conjunto de 1 arranjo em uma plataforma de ferro flutuante e 1 arranjo fixo em um píer na orla da UFPA; sendo ambos utilizados para cobrir o píer, que conecta a parte flutuante ao solo por meio de uma passarela; e 1 PFF no Rio Guamá.

A Figura 46 ilustra a idealização das instalações dos arranjos de plantas fotovoltaicas do EMOB-AMAZON no local designado na orla do *campus* básico da UFPA.

Figura 46 – Localização das instalações dos arranjos fotovoltaicos do EMOB-AMAZON.



Fonte: Elaborado pelo autor (2025).

Para o projeto, foram escolhidos os módulos fotovoltaicos Astronergy ASTRO N5 Bifacial Series (182) 610~630 W, projetados e fabricados pela Canadian Solar. Projetado para atender às demandas de projetos comerciais e industriais, o ASTRO N5 oferece potência nominal que varia de 610 Wp a 630 Wp, o que o torna um dos módulos mais potentes e eficientes disponíveis no mercado. Ele atinge 630 Wp com 22,5% de eficiência, 46,45 V e 13,56 A nas condições de teste padrão (STC), enquanto

na temperatura nominal de operação do módulo (NMOT) atinge um máximo de 473,8 Wp, 43,73 V e 10,83 A (ASTRONERGY, 2023).

Sua tecnologia bifacial permite capturar a luz solar nos lados frontal e traseiro do painel, aumentando a geração de energia em até 30% quando instalado sobre superfícies reflexivas, como telhados claros, solo altamente reflexivo ou até mesmo água, tornando esse recurso particularmente eficaz em ambientes onde a luz adicional pode ser aproveitada (Brennan *et al.,* 2014; Ganesan *et al.,* 2023). No presente caso, esse recurso pode ajudar a melhorar o aproveitamento da luz da reflexão dos flutuadores, pois a água do Rio Guamá é mais escura e dificulta a reflexão da luz solar, o que significa que o albedo na água é muito baixo, pois a luz é refletida de forma especular e não difusa (Alam; Gul; Muneer, 2023).

Como as matrizes instaladas no píer têm uma base de 8 m de largura por 4 m de comprimento, tomando como referência as dimensões do ASTRO N5, com 2,465 m de comprimento por 1,134 m de largura, e o flutuador é cercado por pilares de ancoragem, a cobertura dos módulos fotovoltaicos não pode ultrapassar os limites do flutuador de ferro, de modo que ele só pode acomodar 3 fileiras com 3 módulos cada, totalizando 9 módulos fotovoltaicos com uma potência total de 5,670 kWp. As mesmas medidas se aplicam à cobertura do solo instalada, mas ela não tem as limitações do sistema de ancoragem das flutuantes, comportando 2 fileiras com 7 módulos cada, totalizando 14 módulos e uma potência total de 8.820 kWp. A FPP instalada no rio Guamá é composta por flutuadores de plástico e 6 módulos fotovoltaicos, totalizando 3.780 kWp. Juntos, os três sistemas instalaram um total de 18.270 kWp.

Todas essas seções foram conectadas à rede da UFPA por meio de um inversor solar SMA Sunny Tripower X 12 (STP 12-50) na saída de cada arranjo, classificando os três arranjos como *on-grid*, eliminando a necessidade de sistemas de armazenamento. O inversor SMA STP 12-50, que tem uma potência nominal de 12 kW e uma capacidade de pico de até 50 kW, é usado para conectar os arranjos fotovoltaicos à rede elétrica da UFPA, garantindo a injeção eficiente da energia gerada pelos painéis solares. Seu design compacto e modular facilita a instalação em uma variedade de ambientes, como plataformas flutuantes e estruturas de píer fixas, onde o espaço e as condições ambientais podem ser restritivos (SMA SOLAR TECHNOLOGY AG, *[S.d.]*).

Para a cobertura do píer na plataforma flutuante, o inversor suporta a geração de energia dos 9 módulos fotovoltaicos, adaptando-se às limitações físicas impostas pelos pilares de ancoragem. Para o sistema fixo no píer, com 14 módulos, o inversor garante a conversão eficiente de energia e a integração perfeita com a rede da UFPA. Da mesma forma, para a FPP no rio Guamá, com 6 módulos, o inversor garante a máxima eficiência energética, mesmo no ambiente dinâmico de um rio.

A composição das estruturas da FPP inclui os painéis; os flutuadores; a estrutura de suporte dos painéis fotovoltaicos em poliéster reforçado com fibra de vidro; e fixadores em aço inoxidável ou aço carbono revestidos com Teflon. O sistema de ancoragem utilizará cabos de poliéster de 24 mm com 12 tranças com ilhós e correntes de aço nas extremidades, manilhas e sapatas de aço, todos revestidos de poliamida 11. Os flutuadores são brancos, o que realça a característica dos painéis bifaciais, aumentando o aproveitamento da luz solar (Alam; Gul; Muneer, 2023; Brennan *et al.,* 2014; Ganesan *et al.,* 2023).

A Tabela 3 apresenta as principais informações técnicas dos sistemas solares usados nesta pesquisa.

Tabela 3 – Especificações técnicas dos sistemas fotovoltaicos.

Descrição	Sistema terrestre	Sistema flutuante
Número de painéis	23	9
Potência do painel	630 Wp	630 Wp
Potência total da planta	12.600 Wp	5.670 kWp
Dimensões	2.465m X 1.134m	2.465m X 1.134m
Estrutura de suporte	Metálico 2.5m	Metálico, 2.5m
Tipo de inversor	SMA STP 12-50	SMA STP 12-50

Fonte: Elaborado pelo autor (2025).

Como parte da metodologia, os sistemas fotovoltaicos são utilizados como referência para padronizar a geminação de dispositivos virtuais (*digital twinning*) nos *middlewares* IoT, com ênfase nas variáveis elétricas, verificadas por meio da documentação do inversor comunicativo SMA Sunny Tripower (SMA SOLAR TECHNOLOGY AG, *[S.d.]*), bem como estações meteorológicas e sistemas embarcados baseados no microcontrolador ESP-32.

Esta abordagem assegura a equivalência semântica entre as diferentes plataformas, mantendo a consistência conceitual em todos os sistemas analisados. Seu mecanismo de funcionamento será explorado nas seções seguintes. A estratégia

visa preservar a integridade dos significados durante a interoperabilidade entre sistemas heterogêneos, sendo fundamental para os resultados deste trabalho.

A modelagem dos gêmeos digitais fundamenta-se nessas plataformas físicas, de modo que suas especificações e medições orientam diretamente a definição das variáveis monitoradas e garantem que os dados simulados permaneçam compatíveis com as exigências reais de operação e monitoramento dos dispositivos instalados. Essa abordagem assegura a equivalência semântica entre os sistemas físicos e seus correspondentes virtuais, mantendo a consistência conceitual em todas as plataformas analisadas.

Desse modo, Tabela 4 apresenta a lista das variáveis coletadas, abrangendo parâmetros elétricos dos sistemas fotovoltaicos, informações registradas pelas estações meteorológicas e medições provenientes dos dispositivos embarcados. Essa consolidação permite visualizar de forma organizada os elementos que fundamentam o processo de criação de gêmeos digitais e asseguram a padronização necessária para a interoperabilidade através das plataformas IoT analisadas.

Tabela 4 – Lista de variáveis coletadas através dos dispositivos físicos.

Categoria	Variável	Descrição	Unidade
Sistema Fotovoltaico	Tensão (entrada)	Tensão contínua proveniente do arranjo fotovoltaico	V
	Tensão (saída)	Tensão alternada fornecida pelo inversor	V
	Corrente (entrada)	Corrente fornecida pelos módulos FV	A
	Corrente (saída)	Corrente entregue pelo inversor	A
	Potência	Potência elétrica instantânea gerada	W
	MPPT Status	Estado operacional do rastreador de máxima potência	—
Estação Meteorológica	Temperatura do ar	Temperatura ambiente medida pela estação	°C
	Umidade relativa	Umidade medida pela estação	%
	Velocidade do vento	Velocidade média do vento	m/s
	Precipitação	Acúmulo medido pelo pluviômetro	mm
Sistemas Embarcados	Temperatura do ar	Temperatura ambiente medida pela estação	°C
	Umidade relativa	Umidade medida pela estação	%
	Timestamp	Estado operacional e flags de funcionamento	—

Fonte: Elaborado pelo autor (2025).

4.2. CAMADA DE *MIDDLEWARE*

A camada de *middleware* exerce o papel de um intermediário padronizado entre os dispositivos físicos — como os sistemas fotovoltaicos descritos — e as camadas adjacentes, como a da camada de *software*. Essa camada é essencial para organizar

e estruturar o fluxo de dados que nasce nos dispositivos e precisa chegar limpo e padronizado às aplicações que o utilizam para análise, visualização e tomada de decisão.

No que diz respeito à normalização e centralização de dados, os *middlewares* recebem informações de telemetria por meio de protocolos amplamente utilizados, como HTTP e MQTT, provenientes de fontes heterogêneas que operam com formatos e padrões distintos. O *middleware* atua, então, na unificação desses formatos e protocolos, assegurando a interoperabilidade entre dispositivos diversos e sistemas que consomem esses dados, o que é especialmente relevante em projetos que envolvem uma grande variedade de equipamentos.

Outra função crítica é o gerenciamento de dispositivos. Os *middlewares* mantêm representações digitais do funcionamento dos dispositivos físicos, conhecidas como *Digital Twins* (DT) ou Gêmeos Digitais, que possibilitam não apenas o monitoramento em tempo real, mas também o controle remoto desses dispositivos. Além disso, armazena temporariamente os dados coletados, criando um *buffer* que garante a continuidade da operação mesmo em situações de intermitência na comunicação, algo frequente em regiões remotas como a Amazônia.

Na etapa de integração com as camadas superiores, os dados que foram padronizados pelo *middleware* tornam-se disponíveis para os demais componentes do ecossistema. Essa divisão de responsabilidades promove modularidade, permitindo que cada camada evolua de forma independente, contribuindo para a escalabilidade e facilitando a manutenção do sistema como um todo.

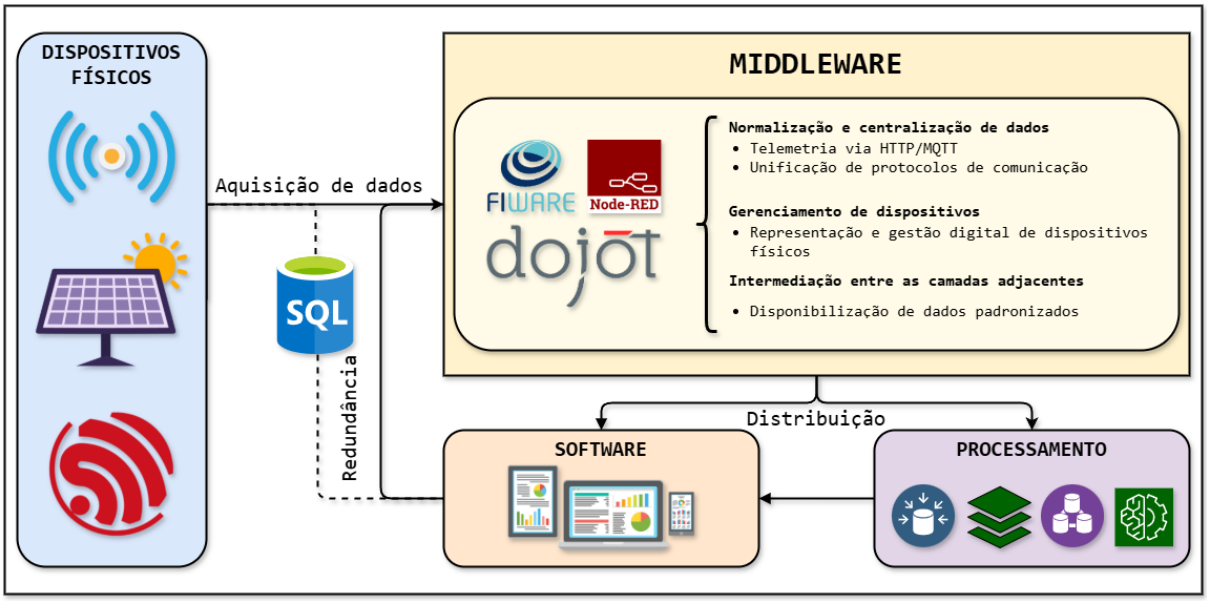
Para garantir a redundância e segurança, um Sistema de Gerenciamento de Banco de Dados (SGBD) armazena dados considerados críticos, como telemetria essencial (atuando como backup no caso de falhas dos *middlewares*) e informações do *software*, incluindo configurações do sistema e perfis de usuários. Esse armazenamento garante a continuidade operacional durante eventuais interrupções ou falhas na camada de *middleware*, minimizando riscos de perda de dados importantes para o funcionamento da plataforma, incrementando a resiliência do sistema.

Essa arquitetura distribuída foi pensada para garantir eficiência, flexibilidade e resiliência, características fundamentais para aplicações complexas e para ambientes

desafiadores como a região amazônica, onde conectividade limitada e condições extremas tornam ainda mais importante contar com soluções robustas e tolerantes a falhas.

A Figura 47 ilustra o fluxo de interação da camada de *middleware* do sistema com as demais.

Figura 47 – Funcionalidades da camada de *middleware* e interação com as demais camadas da arquitetura.



Fonte: Elaborado pelo autor (2025).

4.3. CAMADA DE PROCESSAMENTO

A camada de processamento é responsável por transformar dados brutos coletados de diferentes fontes — como sistemas embarcados, estações meteorológicas e sistemas fotovoltaicos — em informações qualificadas para apoiar a tomada de decisão. Essa transformação inclui etapas de normalização, limpeza e enriquecimento dos dados, que são realizadas de forma distribuída por meio da computação de borda. O uso de *edge computing* permite reduzir a latência, otimizar o tráfego na rede e gerar respostas mais rápidas para eventos detectados em campo. Entre as principais funções dessa camada estão:

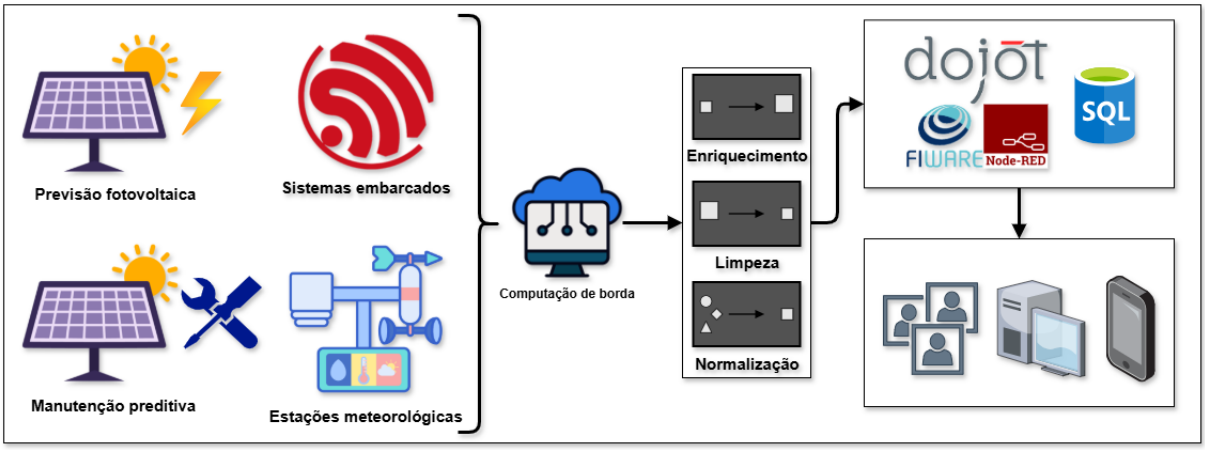
- Previsão fotovoltaica, baseada em modelos que utilizam dados meteorológicos e históricos de geração para estimar a produção futura de energia;

- Manutenção preditiva, que permite antecipar falhas em equipamentos a partir do monitoramento contínuo e do histórico de operação, baseada na Análise de Modos e Efeitos de Falha (FMEA).

Após o processamento inicial, os dados são enviados para *middlewares* e bancos de dados, que fazem a integração, armazenamento e disponibilização das informações para outros sistemas. Por fim, esses dados processados são consumidos por interfaces e aplicações que oferecem visualizações em tempo real, relatórios e alertas, apoiando a gestão operacional e a tomada de decisão de forma mais eficiente.

A Figura 48 ilustra as funcionalidades da camada de processamento e o fluxo de interação com as demais camadas da arquitetura.

Figura 48 – Funcionalidades da camada de processamento e interação com as demais camadas da arquitetura.



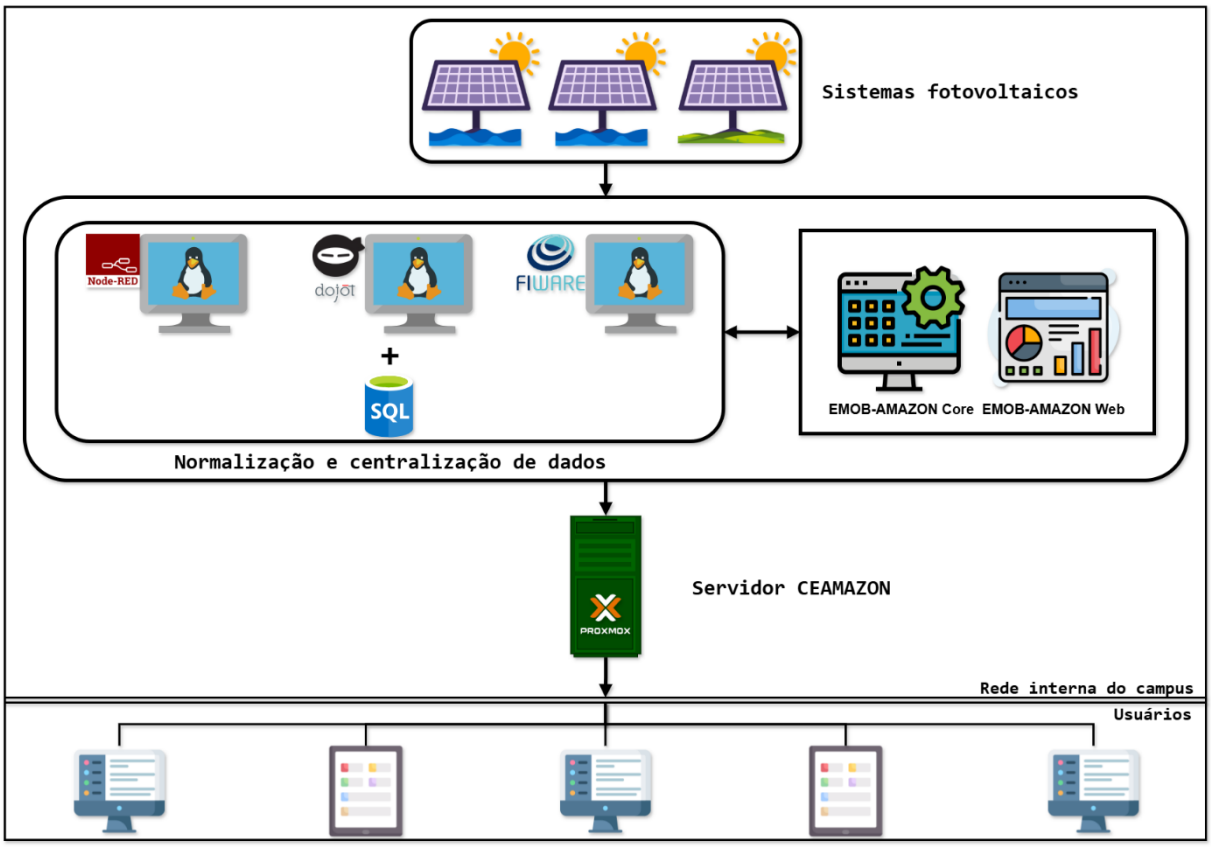
Fonte: Elaborado pelo autor (2025).

4.4. CAMADA DE *SOFTWARE*

A arquitetura modelada para o sistema de monitoramento do EMOB-AMAZON foi concebida para ser modular e escalável, permitindo que diferentes camadas do projeto sejam desenvolvidas, testadas e implantadas de forma independente (Figura 49). Essa organização facilita a evolução do sistema e a incorporação de novas funcionalidades ao longo do tempo. Estruturada de forma hierárquica, a arquitetura se estende desde a camada de dispositivos físicos até a rede interna do campus, responsável por permitir que usuários acessem os dados e funcionalidades do sistema.

No topo, encontra-se a camada de dispositivos físicos, composta por *hardware* como matrizes fotovoltaicas, inversores e sistemas embarcados. Esses dispositivos coletam dados de geração de energia e telemetria, que são então transmitidos para a camada intermediária. Essa camada inclui os *middlewares* e um SGBD responsável por centralizar, normalizar e armazenar os dados, antes de repassá-los ao núcleo central da aplicação.

Figura 49 – Estrutura da camada de *software* do EMOB-AMAZON.



Fonte: Elaborado pelo autor (2025).

O núcleo da aplicação atua como elo central entre os dados coletados na borda, o processamento e os serviços oferecidos ao usuário final. É nessa parte que estão implementadas as regras de negócio, garantindo segurança, consistência e disponibilidade das informações para o restante do sistema. Na sequência, a aplicação web transforma esses dados complexos e processados em informações úteis e de fácil visualização. Por meio de dashboards, relatórios e alertas, gestores e operadores podem monitorar o desempenho dos sistemas fotovoltaicos, analisar tendências e agir rapidamente frente a situações críticas.

4.5. IMPLANTAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

Para apoiar a construção, evolução e manutenção da arquitetura do EMOB-AMAZON, foi implantado um ambiente de desenvolvimento dedicado, estruturado de forma modular e isolada para permitir que cada componente — núcleo da aplicação, *middlewares*, base de dados e interface *web* — possa ser desenvolvido, testado e integrado de maneira independente. Esse ambiente proporciona maior segurança, rastreabilidade e flexibilidade durante o ciclo de desenvolvimento, facilitando ajustes contínuos, validação de novas funcionalidades e simulações antes da implantação no ambiente de produção, garantindo assim maior confiabilidade e qualidade no sistema final.

4.5.1. ESPECIFICAÇÕES DE *SOFTWARE* DO AMBIENTE DE DESENVOLVIMENTO

A infraestrutura projetada para o ambiente de desenvolvimento foi implementada em um servidor Dell PowerEdge T320, de propriedade do CEAMAZON, utilizando como plataforma de virtualização o Proxmox VE 7.3-3, baseado no Debian GNU/Linux com Kernel 5.15.74-1-pve. Esse arranjo permite criar, gerenciar e alocar recursos de *hardware* entre diferentes máquinas virtuais de forma centralizada e segura.

Ao todo, foram criadas quatro VMs:

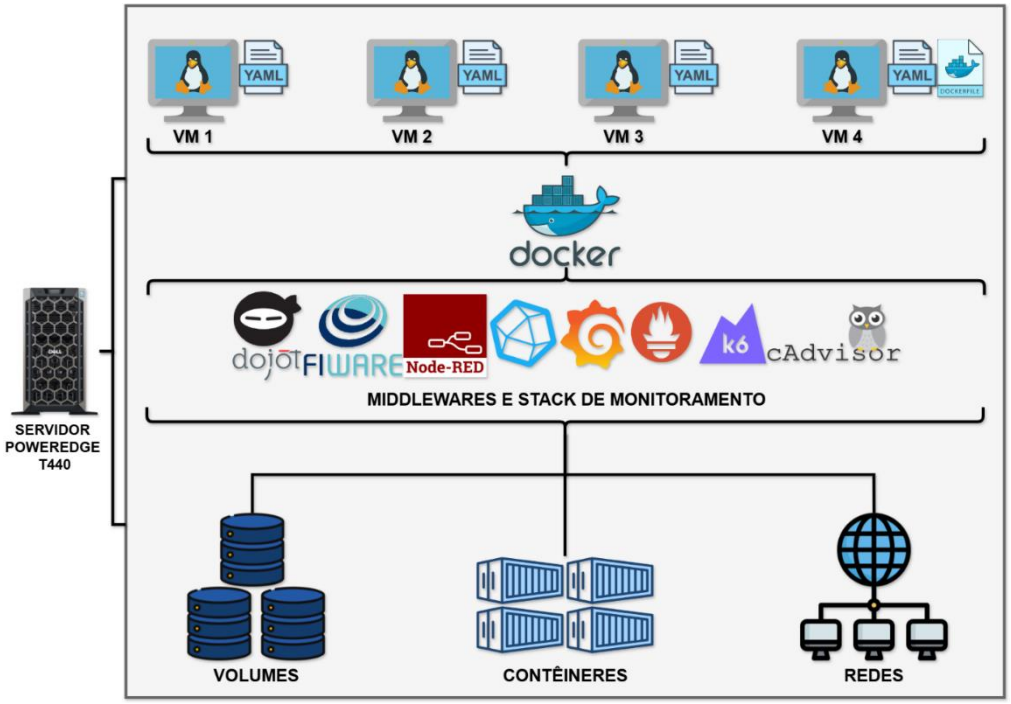
- Três delas operam com o sistema operacional Linux Ubuntu Desktop 20.04 e hospedam, respectivamente, os *middlewares* Dojot, FIWARE-Orion e Node-RED. Cada VM conta ainda com uma instância do cAdvisor (v0.47.2) para monitoramento do uso de recursos pelos contêineres Docker.

- A quarta VM utiliza o CentOS Linux 7.9 e foi destinada à *stack* de observabilidade e testes, na qual foram implantadas as instâncias do InfluxDB (v2.7.2), para persistência de métricas técnicas do servidor (VMs); Grafana (v12.0.2), para criação e visualização da *dashboards* interativas; Grafana Image Renderer (v3.10.0), utilizado para exportar os gráficos das métricas técnicas; Prometheus (v2.47.0), utilizado para persistir métricas dos contêineres Docker; e mais uma instância do cAdvisor (v0.47.2). Nesse mesmo ambiente, foi ainda criada uma versão customizada do Grafana k6, construída a partir da ferramenta xk6 e adicionando

suporte aos módulos “xk6-output-influxdb” e “xk6-dashboard”, os quais habilitam a ferramenta a exportar os relatórios para o um *bucket* do InfluxDB v2.x e gerar uma *dashboard* ao final de cada teste, respectivamente. Essa versão foi encapsulada em um contêiner Docker leve, baseado em Alpine Linux, e configurada para rodar com segurança sob um usuário dedicado não-*root*.

Para viabilizar a implantação do ambiente de desenvolvimento, foram elaborados arquivos de configuração responsáveis por definir os parâmetros de funcionamento dos serviços, suas dependências e as conexões internas entre eles. Esses arquivos foram construídos com base nas documentações oficiais dos *middlewares* e ferramentas adotados, assegurando conformidade com as boas práticas recomendadas por cada projeto. A estruturação foi realizada em *YAML* uma linguagem de sintaxe limpa, minimalista e legível, ideal para a serialização de dados e a definição de configurações de maneira clara e organizada. Uma visão geral do processo de implantação é apresentada no fluxograma de implantação das arquiteturas propostas, ilustrado na Figura 50.

Figura 50 – Fluxograma de implantação dos *middlewares* e stack de monitoramento.



Fonte: Elaborado pelo autor (2025).

A implantação do ambiente de desenvolvimento foi realizada utilizando o Docker Engine na versão v27.5.1 e o Docker Compose v2.32.3, aplicados aos *middlewares* FIWARE-Orion e Node-RED. Para o Dojot, optou-se pela versão v2.16.0

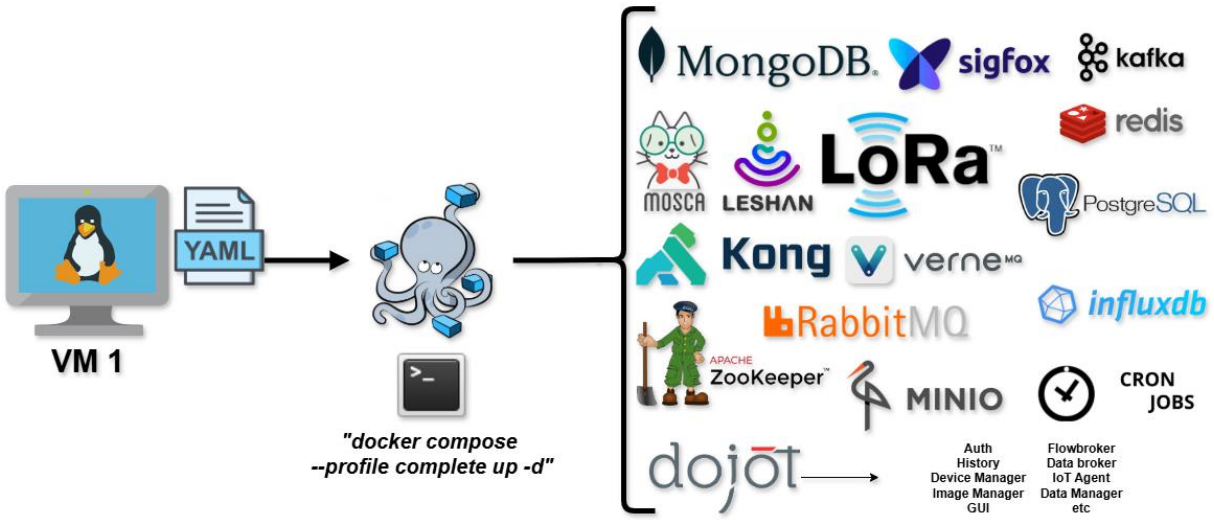
do Docker Compose por questões de retrocompatibilidade, pois, a partir da v2.17.0, o *plugin* passou a rejeitar arquivos docker-compose.yml que utilizam chaves duplicadas no recurso de mesclagem *YAML* (<<), e os *manifests* do Dojot ainda dependem dessa prática para estruturar suas configurações de contêineres.

A escolha por estruturar todo ambiente de desenvolvimento utilizando virtualização baseada em contêineres teve como principal objetivo reduzir a necessidade de escalabilidade vertical, já que essa técnica permite uma diminuição expressiva dos custos de computação em relação à virtualização tradicional baseada exclusivamente em VMs (Fava *et al.,* 2024; Hage Hassan *et al.,* 2022; Li *et al.,* 2023).

4.5.1.1. DESCRIÇÃO DAS ARQUITETURAS DOS *MIDDLEWARE*S IOT IMPLANTADOS

A plataforma Dojot disponibiliza um total de 17 perfis configuráveis via Docker Compose, que permitem iniciar grupos específicos de serviços conforme as necessidades do ambiente ou caso de uso. Entre esses perfis, destacam-se o “basic”, que contém os serviços mínimos essenciais para o funcionamento da plataforma, e “complete”, que engloba todos os serviços disponibilizados pela plataforma, o qual foi o selecionado, na versão 0.8.0 da plataforma (Figura 51).

Figura 51 – Componentes do perfil implantado da plataforma Dojot.



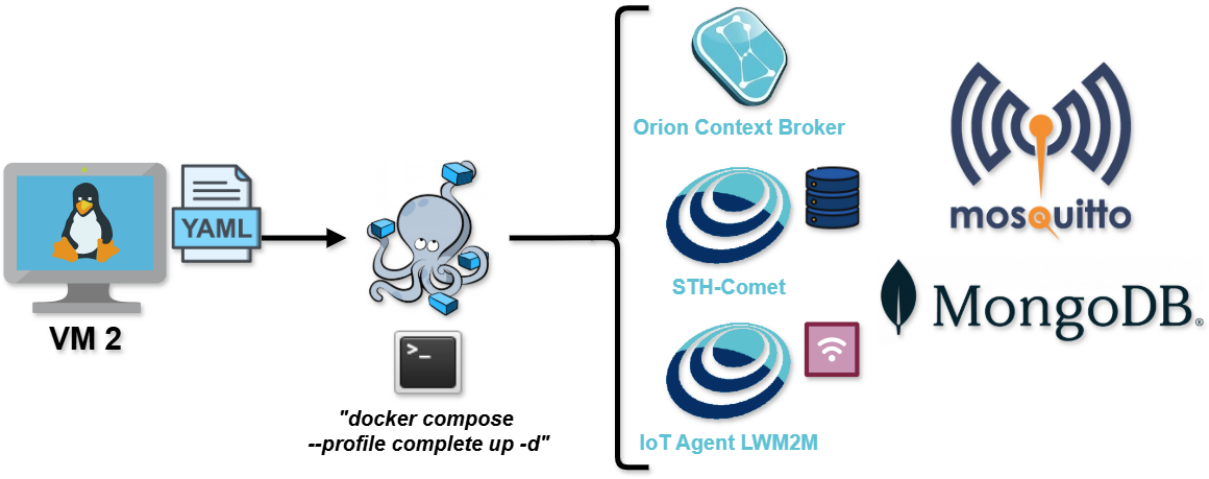
Fonte: Elaborado pelo autor (2025).

A escolha do perfil é feita diretamente pelo comando de inicialização “docker compose up”, utilizando a opção “--profile” e o nome do perfil desejado. Essa escolha

garante a disponibilidade de todas as funcionalidades e componentes integrados, facilitando testes abrangentes, a validação completa dos fluxos e maior flexibilidade para adaptações futuras, sem necessidade de reconfiguração adicional. Dessa forma, o perfil completo assegura um ambiente robusto e alinhado ao escopo do projeto. Além dos perfis “basic” e “complete”, também estão disponíveis os perfis de persistência, como “mongodb-persistence” e “influxdb-persistence”, responsáveis pelo armazenamento de dados em bancos especializados outros que incluem agentes para comunicação com dispositivos utilizando protocolos variados, como “iot-agent-mqtt”, “iot-agent-http” e “lwm2m-iot-agent”, bem como serviços para interface *web*, gestão de usuários, dispositivos e tarefas periódicas.

Para o ambiente da VM FIWARE, foi implantado um conjunto integrado de serviços essenciais da plataforma, assegurando a ingestão, gerenciamento, persistência e análise de dados IoT (Figura 52). O principal componente dessa infraestrutura é o Orion Context Broker (v3.11.0), responsável por gerenciar o contexto da plataforma, armazenando entidades e seus atributos, além de possibilitar a atualização e consulta em tempo real dessas informações.

Figura 52 – Componentes da arquitetura implantada do FIWARE.



Fonte: Elaborado pelo autor (2025).

O armazenamento dos dados foi realizado por meio de duas instâncias do MongoDB (v4.4): uma destinada ao armazenamento interno do Orion e do IoT Agent, configurada sem autenticação; e outra dedicada à persistência dos dados históricos gerados pelo componente STH Comet (2.11.0), que conta com autenticação ativada para garantir maior segurança. O STH Comet é o serviço responsável por armazenar

e fornecer acesso às séries temporais dos dados coletados, facilitando a análise histórica.

A comunicação entre os dispositivos IoT e a plataforma foi realizada por meio do *broker* MQTT Mosquitto (v2.0.21), que recebe as mensagens enviadas pelos dispositivos. O IoT Agent for Ultralight (3.7.0), configurado para o protocolo Ultralight MQTT, atua como tradutor desses dados, convertendo as informações para o formato NGSI utilizado pelo Orion, integrando assim o mundo físico com a camada de gerenciamento de contexto.

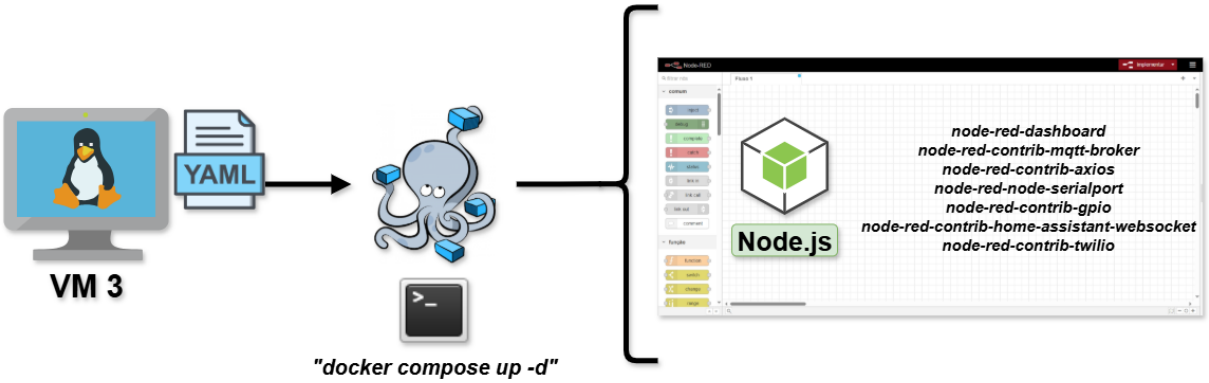
Essa infraestrutura constitui uma base funcional para o desenvolvimento e testes similar à plataforma Dojot, fornecendo serviços essenciais para o controle, monitoramento e análise de dispositivos IoT, com a flexibilidade necessária para futuras expansões e adaptações.

Por último, para o ambiente da VM Node-RED, foi implantada uma instância dedicada do Node-RED (v4.0.9) configurada com um conjunto de *nodes* específicos voltados para o desenvolvimento, integração e teste de fluxos IoT (Figura 53).

Entre os principais *nodes* instalados estão os do tipo “node-red-dashboard”, que permitem criar interfaces gráficas para visualização de dados e controle de dispositivos; os *nodes* MQTT, essenciais para comunicação em tempo real com o *broker* Mosquitto; além dos *nodes* “http request” e “http in/out”, que possibilitam o envio e o recebimento de dados via requisições HTTP. Também foram incluídos *nodes* destinados à integração com bancos de dados e APIs externas, além de utilitários para transformação de dados, tratamento de *strings*, manipulação de JSON, lógica condicional e agendamento de tarefas, que tornam o ambiente mais flexível e adaptável às diferentes necessidades do projeto.

Além desses, o ambiente também contemplou *nodes* para protocolos industriais e dispositivos físicos, como “node-red-node-serialport” para leitura e escrita em portas seriais e “node-red-node-pi-gpio” para o controle direto de General Purpose Input/Output (GPIO). Também foram instalados *nodes* voltados para integrações avançadas com plataformas e serviços externos, como “node-red-contrib-home-assistant-websocket” para automação residencial e “node-red-contrib-twilio” para envio de mensagens SMS e WhatsApp.

Figura 53 – Componentes implantados do Node-RED.



Fonte: Elaborado pelo autor (2025).

Essa configuração garante ao Node-RED a capacidade de atuar como camada de orquestração, processamento e integração, permitindo construir, monitorar e ajustar fluxos de dados de forma visual e interativa. Com essa estrutura, a VM Node-RED complementa a arquitetura geral do ambiente, oferecendo uma interface amigável para o desenvolvimento rápido de protótipos, integração de serviços e visualização das informações geradas pelos dispositivos IoT, de forma totalmente alinhada ao restante da infraestrutura do projeto.

4.5.2. ESPECIFICAÇÕES TÉCNICAS DE *HARDWARE*

O servidor Dell PowerEdge T320, utilizado para hospedar o ambiente de desenvolvimento, foi equipado com um processador Intel Xeon E5-2440 v2, que conta com 8 núcleos e 16 *threads*, frequência base de 1,9 GHz e turbo de até aproximadamente 2,4 GHz. Esse processador pertence à família Xeon E5-2400 v2, utiliza o soquete LGA1356 e oferece suporte a tecnologias como virtualização (VT-x) e instruções avançadas, sendo adequado para cargas de trabalho de virtualização em plataformas como o Proxmox VE.

Em relação à memória RAM, o servidor dispõe de 24 GB de DDR3 ECC Registered, operando a 1333 MT/s. Essa configuração é composta por um módulo de 16 GB e dois módulos de 4 GB, ocupando assim 3 dos 6 slots disponíveis na placa-mãe. Essa arquitetura possibilita expansão significativa, pois cada *slot* pode receber módulos de maior capacidade, desde que sejam respeitados os limites impostos pelo *chipset* e pelo processador.

No que se refere ao armazenamento, o servidor conta com dois discos principais: um HDD de aproximadamente 300 GB, que abriga o sistema Proxmox e os discos virtuais das máquinas, organizado em volumes lógicos (LVM); e um SSD de 450 GB, formatado em ext4, que pode ser integrado ao ambiente de virtualização para ampliar o espaço disponível e melhorar o desempenho.

A Tabela 5 detalha os principais componentes de *hardware* do servidor Dell PowerEdge T320 utilizado para hospedar o ambiente de desenvolvimento, incluindo processador, memória, armazenamento e a estratégia de virtualização adotada.

Tabela 5 – Especificações de *hardware* do servidor Dell PowerEdge T320.

Componente	Detalhes
Servidor	Dell PowerEdge T320
Processador	Intel Xeon E5-2440 v2 – 8 núcleos / 16 threads, 1,9 GHz (turbo até ~2,4 GHz), soquete LGA1356, suporte a VT-x
Memória RAM	24 GB DDR3 ECC Registered (1×16 GB + 2×4 GB), frequência de 1333 MT/s; 3 de 6 slots ocupados, suportando expansão
Armazenamento	- HDD ~300 GB (sistema Proxmox + discos virtuais das VMs, gerenciado via LVM) - SSD 450 GB (ext4, não integrado ao LVM inicialmente)
Virtualização	Hospedagem em contêineres distribuídos em máquinas virtuais isoladas, com redes internas configuradas para controle de recursos e coleta individual de métricas

Fonte: Elaborado pelo autor (2025).

Simultaneamente ao encapsulamento do ambiente de desenvolvimento em contêineres, optou-se por hospedar toda a arquitetura em um servidor físico segmentado em máquinas virtuais isoladas. Nessas VMs são executados os contêineres e configuradas as redes internas, atendendo à necessidade de separar e controlar de maneira precisa o consumo de recursos de *hardware* por VM, o que permite um gerenciamento mais eficiente da infraestrutura.

Essa estratégia possibilitou a coleta individualizada de métricas de desempenho de cada VM, conforme planejado, garantindo monitoramento contínuo e análise de performance das diferentes arquiteturas implantadas para cada *middleware*. O objetivo central dessa abordagem é fornecer subsídios para o planejamento e o gerenciamento eficiente dos recursos físicos do servidor, assegurando a escalabilidade necessária para futuras demandas e aplicações.

Com isso, a alocação planejada garante não apenas o funcionamento estável e otimizado de cada componente do ambiente, mas também permite capturar métricas detalhadas de uso de CPU, memória, rede e disco de forma isolada, proporcionando maior controle, transparência e visibilidade sobre o comportamento de toda a infraestrutura.

A Tabela 6 apresenta detalhadamente a quantidade de recursos de *hardware* destinados a cada máquina virtual configurada no servidor físico.

Tabela 6 – Recursos de *hardware* alocados para cada VM.

VM	Núcleos	RAM (GB)	HD (GB)	OS
CentOS	2	2	100	Linux CentOS 7.9
Dojot	4	6	32	Ubuntu Desktop 20.04
FIWARE	4	6	32	Ubuntu Desktop 20.04
Node-RED	4	6	32	Ubuntu Desktop 20.04

Fonte: Elaborado pelo autor (2025).

Observa-se que a VM CentOS recebeu menos recursos, uma vez que sua função está restrita à camada de monitoramento e observabilidade do ambiente, cuja carga de processamento é naturalmente mais baixa. Por outro lado, as VMs responsáveis pelos *middlewares* IoT foram configuradas com recursos mais robustos e de mesma proporção, assegurando maior capacidade de processamento e armazenamento para lidar com o tráfego de dados e as demandas de processamento específicas de cada plataforma.

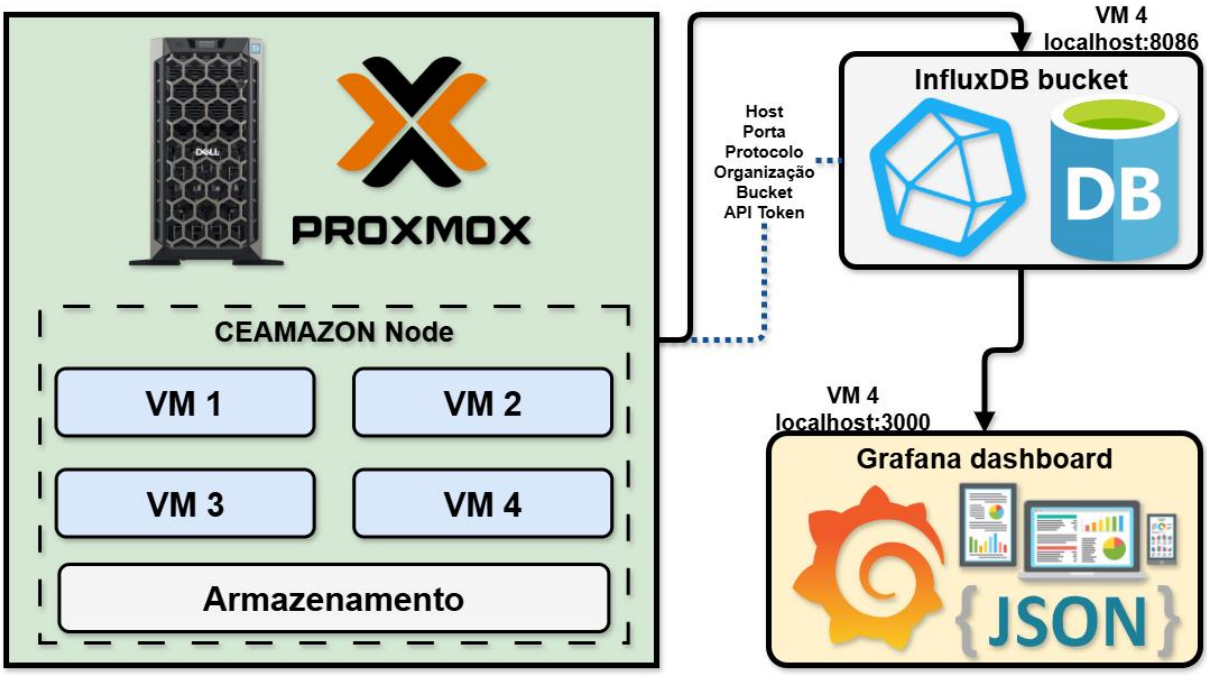
Essa estratégia de configuração busca equilibrar o uso eficiente dos recursos físicos do servidor com a necessidade de isolamento entre as aplicações, facilitando o monitoramento individual de desempenho e preservando a possibilidade de escalabilidade para futuras aplicações que venham a ser incorporadas ao ambiente.

4.6. IMPLANTAÇÃO DO SISTEMA DE OBSERVABILIDADE E MONITORAMENTO

O primeiro elemento do sistema de observabilidade implantado é composto por um servidor de métricas externo configurado no Proxmox VE, que integra as instâncias do InfluxDB e Grafana (Figura 54). Essa camada é responsável por coletar e armazenar diversas estatísticas detalhadas sobre os *hosts* físicos, máquinas virtuais e volumes de armazenamento do ambiente. Entre os dados monitorados estão o uso

de CPU e memória (RAM), operações de leitura e escrita em disco, tráfego de rede (entrada e saída) e a ocupação de espaço em armazenamento.

Figura 54 – Servidor de métricas externo do Proxmox VE com InfluxDB e Grafana.



Fonte: Elaborado pelo autor (2025).

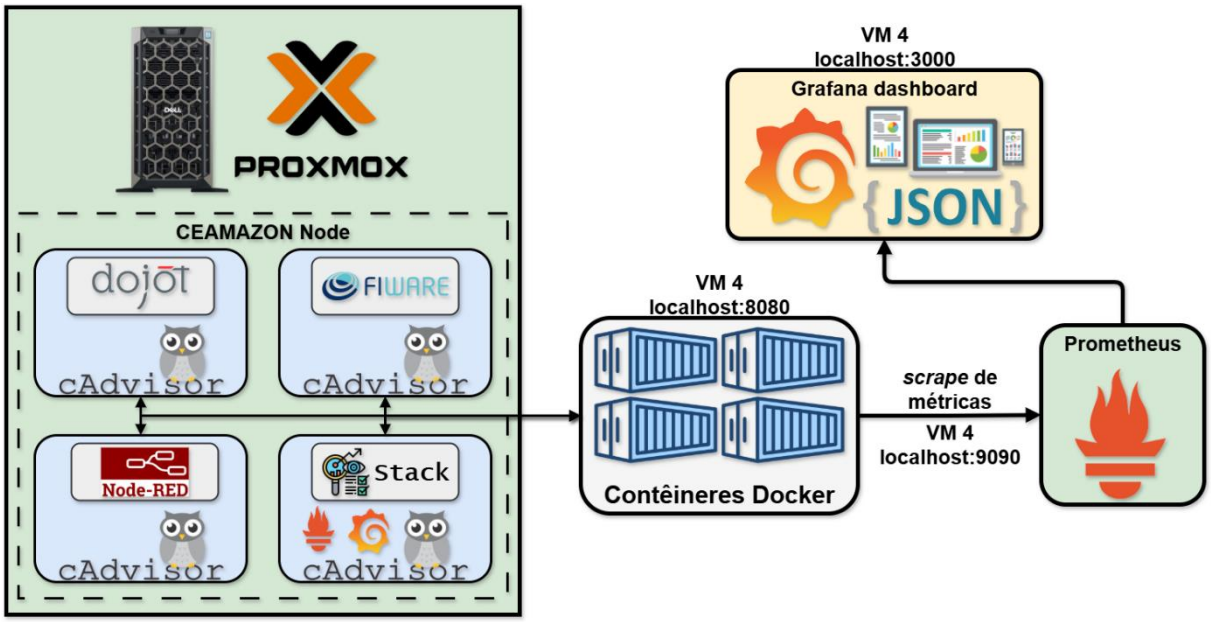
Os dados coletados são armazenados em um *bucket*, que é um repositório lógico do InfluxDB destinado ao armazenamento de séries temporais, instanciado na VM 4 e acessível localmente pelo endereço <localhost:8086>. Essas informações são consultadas por meio de um código Flux, que integra consultas complexas ao banco e retorna os resultados estruturados. Esse código está associado a um *template* JSON do Grafana, configurado para que essa instância reconheça o InfluxDB, implantado como servidor de métricas externas do Proxmox VE, como a fonte de dados principal para a construção das *dashboards*. Dessa forma, é possível visualizar de forma dinâmica e interativa as métricas coletadas, mantendo a observabilidade centralizada e alinhada ao restante da infraestrutura.

Dessa forma, essa infraestrutura garante uma visão abrangente e em tempo real do comportamento e desempenho geral do ambiente virtualizado, servindo como base para análises históricas, identificação de gargalos e planejamento de capacidade futura.

O segundo conjunto da pilha de monitoramento foi estruturado utilizando cAdvisor e Prometheus, compondo uma camada dedicada ao acompanhamento em

tempo real do consumo de recursos e do desempenho dos contêineres Docker das aplicações (Figura 55).

Figura 55 – Monitoramento de métricas de contêineres Docker com cAdvisor e Prometheus.



Fonte: Elaborado pelo autor (2025)

O cAdvisor é responsável por coletar métricas detalhadas de cada contêiner em execução, incluindo uso de CPU, consumo de memória, estatísticas de rede, leituras e escritas em disco, além da quantidade de contêineres ativos. Essas informações são expostas continuamente por meio de um *endpoint* HTTP configurado na VM 4 (localhost:8080), em um formato que o Prometheus consegue interpretar diretamente.

O Prometheus, instanciado na VM 4 e acessível pelo endereço <localhost:9090>, funciona como o componente central responsável pela coleta de dados. Ele realiza periodicamente *scrapes* no *endpoint* exposto pelo cAdvisor, armazenando essas métricas em seu banco de dados de séries temporais. Essa estrutura possibilita não apenas manter um histórico detalhado do uso de recursos, mas também executar consultas flexíveis e configurar alertas automáticos baseados em critérios definidos, contribuindo para a identificação proativa de anomalias e problemas de desempenho.

Por fim, os dados coletados pelo Prometheus são visualizados no Grafana, que gera *dashboards* interativos para análise, diagnóstico de falhas, detecção de tendências e tomada de decisão baseada em dados reais do comportamento dos

contêineres Docker. Essa integração, somada ao monitoramento geral do servidor físico, resulta em uma solução robusta, detalhada e escalável para o monitoramento completo do ambiente de desenvolvimento.

4.6.1. MÉTRICAS TÉCNICAS E ÍNDICES ESTATÍSTICOS

As métricas técnicas monitoradas foram cuidadosamente definidas para oferecer uma visão abrangente tanto do comportamento geral do servidor físico e das VMs, quanto do desempenho específico dos contêineres em execução nos ambientes virtualizados.

Para a visão geral do consumo por máquinas virtuais e o monitoramento detalhado por contêineres, as principais métricas acompanhadas incluem:

- **Uso de CPU**: que indica o percentual de utilização do processador, essencial para identificar gargalos de processamento ou sobrecargas;
- **Uso de memória**: monitora o consumo da memória RAM disponível, permitindo detectar situações de uso excessivo ou vazamentos de memória;
- **Leitura e escrita em disco**: avalia a taxa de operações de leitura e gravação, auxiliando no diagnóstico de problemas relacionados ao desempenho do armazenamento;
- **Tráfego de rede (entrada/saída)**: registra o volume de dados que entra e sai pelas interfaces de rede, ajudando a compreender o perfil de uso da infraestrutura e detectar picos de comunicação.

As métricas técnicas referentes ao desempenho das máquinas virtuais e dos contêineres Docker foram coletadas ao longo do intervalo de testes com o Grafana k6, abrangendo diferentes variações de carga e operação do ambiente. Esse período foi escolhido para capturar padrões representativos de uso e possibilitar uma análise mais precisa dos recursos computacionais consumidos por cada componente da arquitetura.

Os índices estatísticos foram calculados a partir das métricas coletadas e desempenharam um papel essencial na interpretação e análise detalhada do comportamento do ambiente monitorado. Essa etapa permitiu a transformação dos

dados brutos em informações relevantes, servindo de base para embasar decisões técnicas e operacionais com maior precisão e confiabilidade.

Entre os principais índices utilizados estão:

- **Média**: que fornece uma visão geral do valor médio das métricas monitoradas ao longo de um determinado período, ajudando a identificar tendências gerais de consumo e desempenho (Equação 1).

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{1}$$

onde:

μ é a média,

N é o número de dados coletados,

x_i são os valores individuais das métricas.

- **Mediana**: que representa o valor central da distribuição dos dados, oferecendo uma medida robusta menos sensível a valores extremos ou picos eventuais que poderiam distorcer a análise (Equação 2 e 3).

$$\mu\delta = x\left(\frac{N+1}{2}\right) \tag{2}$$

Se N for ímpar;

$$\mu\delta = \frac{x_{\left(\frac{N}{2}\right)} + x_{\left(\frac{N}{2}+1\right)}}{2} \tag{3}$$

Se N for par.

onde $x_{(k)}$ representa o k-ésimo menor valor no conjunto ordenado.

- **Desvio padrão**: que quantifica o grau de variação ou dispersão em relação à média, permitindo compreender a estabilidade ou variabilidade do uso de recursos como CPU, memória, rede e disco (Equação 3).

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \quad (4)$$

onde:

σ é o desvio padrão,

$\bar{x}$ é a média,

x_i são os valores individuais.

- **Percentis (90, 95 e 99)**: que indicam os valores abaixo dos quais se encontram, respectivamente, 90%, 95% e 99% dos dados registrados, sendo especialmente úteis para identificar picos, dimensionar limites de capacidade e configurar alertas para situações de risco (Equação 5).

$$\Pi_\pi = x_{(k)}\ com\ k = \left\lceil \frac{p}{100} \times N \right\rceil \quad (5)$$

onde:

p é o percentil desejado,

N é o número total de dados,

$x_{(k)}$ é o k-ésimo valor após ordenar os dados em ordem crescente,

$\lceil \cdot \rceil$ é a função teto, que arredonda para cima.

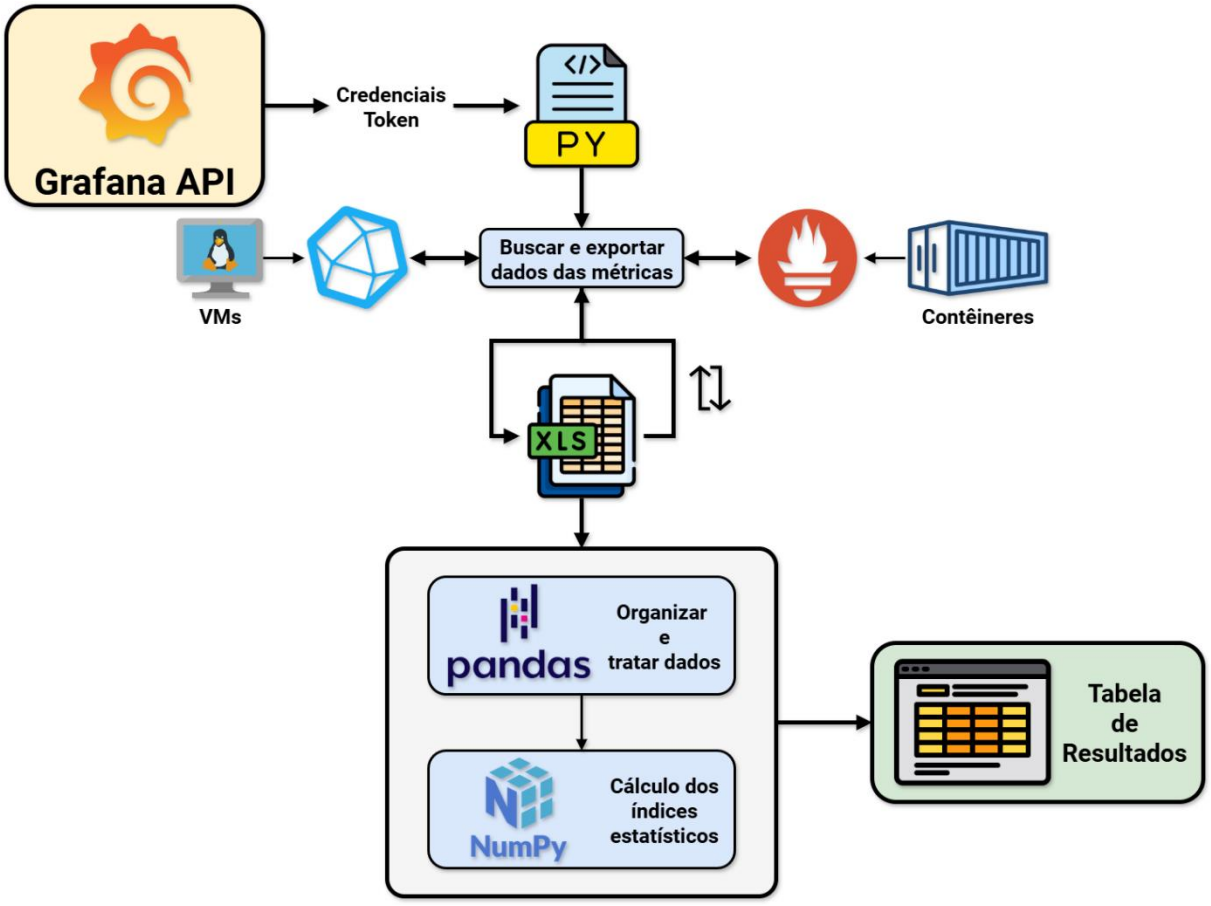
Os índices estatísticos foram calculados de forma automatizada por meio de um *script* em Python, desenvolvido especificamente para integrar e processar os dados extraídos dos painéis do Grafana. A Figura 56 apresenta o fluxograma que ilustra o funcionamento desse *script*, desde a obtenção dos dados até o cálculo dos índices estatísticos.

Nesse processo, o *script* realiza consultas diretas às APIs do Grafana, que fornecem os dados brutos das métricas já coletadas e armazenadas nos bancos de séries temporais, como InfluxDB e Prometheus. Ao receber essas informações em formato XLS, o *script* as organiza em estruturas apropriadas — como *dataframes* do Pandas — e executa a normalização das unidades das métricas monitoradas (por exemplo, convertendo kB para MB). Em seguida, são aplicados procedimentos de

limpeza para eliminar valores nulos ou inconsistentes. Somente após essa preparação, o *script* calcula automaticamente os índices estatísticos selecionados gerando resultados prontos para análise e interpretação.

Essa abordagem permite atualizar os índices sempre que novas métricas são registradas, eliminando a necessidade de cálculos manuais, além de garantir maior consistência e rastreabilidade nos resultados. Como resultado, obtêm-se relatórios estatísticos completos que servem de base para análises históricas, ajustes de capacidade e decisões sobre a infraestrutura monitorada, mantendo todo o processo integrado ao fluxo de monitoramento já existente.

Figura 56 – Fluxograma do *script* de cálculo dos índices estatísticos.



Fonte: Elaborado pelo autor (2025).

Esses índices estatísticos foram aplicados tanto às métricas das máquinas virtuais quanto aos contêineres Docker, permitindo uma análise mais completa do comportamento da infraestrutura. Ademais, para a análise dos resultados, foram selecionados os desempenhos por VM e por contêiner de maior média de consumo/uso por VM. Com isso, é possível antecipar gargalos, ajustar alocação de

recursos de forma mais precisa e garantir maior confiabilidade e desempenho no ambiente de desenvolvimento e produção.

4.7. TESTES DE DESEMPENHO E COMPORTAMENTO OPERACIONAL SOB CARGAS SINTÉTICAS

Os testes selecionados, desenvolvidos e executados neste trabalho abrangem três tipos principais: *smoke test*, *load test* e *stress test*, cada um com objetivos complementares para avaliar a robustez e a confiabilidade dos serviços envolvidos. Os testes conduzidos neste trabalho foram definidos com base na classificação oficial (Tabela 7) apresentada na documentação do Grafana k6, sintetizada na Tabela correspondente. Essa referência orientou a seleção e a configuração dos diferentes cenários de avaliação, garantindo que cada teste fosse executado de acordo com as melhores práticas de análise de desempenho recomendadas pela ferramenta.

Tabela 7 – Tipos de testes de desempenho e critérios conforme a documentação do Grafana k6.

Tipo	VUs / Taxa de Throughput	Duração	Quando?
Smoke	Baixa	Curta (segundos ou minutos)	Quando o sistema ou o código da aplicação sofre alterações relevantes. Verifica lógica funcional, métricas básicas e desvios.
Load	Produção média	Média (5–60 minutos)	Usado frequentemente para verificar se o sistema mantém o desempenho com carga média.
Stress	Alta (acima da média)	Média (5–60 minutos)	Quando o sistema pode receber cargas acima da média, para verificar como ele se comporta.
Soak	Média	Longa (horas)	Após mudanças, para testar o sistema sob uso contínuo prolongado.
Spike	Muito alta	Curta (poucos minutos)	Quando o sistema precisa se preparar para eventos sazonais ou picos frequentes de tráfego.
Breakpoint	Aumenta até o limite de falha	O tempo necessário	Usado algumas vezes para identificar os limites superiores do sistema.

Fonte: Adaptado de Grafana Labs (2025b).

Os testes foram planejados para avaliar progressivamente a disponibilidade, o desempenho e os limites da infraestrutura. O *smoke test*, por ser preliminar e de baixa carga, teve duração de 30 segundos, servindo apenas para confirmar a disponibilidade básica dos *endpoints* antes da execução dos cenários mais intensivos. Em seguida, o *load test* avaliou o comportamento dos serviços sob uma carga representativa do uso normal, enquanto o *stress test* simulou um aumento progressivo de requisições para identificar os limites de estabilidade e a capacidade de recuperação do sistema.

As durações adotadas seguiram as recomendações da documentação oficial do Grafana k6, sintetizadas na tabela de tipos de teste. O *smoke test* enquadra-se na categoria de curta duração. O *load test*, classificado como teste de duração média, foi executado por 5 minutos para permitir a coleta de métricas estáveis sob carga típica. O *stress test*, também de duração média, teve 21 minutos, suficientes para observar o comportamento do sistema sob escalonamento de usuários e identificar o ponto de degradação.

A execução dos três cenários em três máquinas virtuais resultou em aproximadamente 80 minutos totais, considerando tanto o tempo de cada teste quanto os intervalos necessários entre eles para estabilização dos serviços, descarte de caches e reinicialização dos coletores de métricas. Esse período corresponde à janela completa de observação utilizada para monitorar o comportamento das VMs e dos contêineres durante variações controladas de carga, de forma individual e sequencial, iniciando pela VM da plataforma Dojot, em seguida FIWARE e, por fim, Node-RED.

A combinação dos testes *smoke*, *load* e *stress* tem como objetivo fornecer uma visão abrangente da saúde, estabilidade e resiliência do ambiente, além de representar a experiência do usuário por meio do tempo de resposta das plataformas. Essa abordagem se torna especialmente importante diante da impossibilidade de padronizar um teste de usabilidade ou interface, uma vez que apenas o Dojot e o Node-RED dispõem de interface gráfica. O *smoke test* confirmou a operação mínima dos serviços; o *load test* avaliou o desempenho em condições de uso realistas; e o *stress test* identificou os limites máximos de operação e a capacidade de resposta da infraestrutura sob sobrecarga. Assim, foi possível realizar uma análise consistente do impacto das cargas simuladas sobre os recursos computacionais do ambiente virtualizado.

Para garantir o rigor metodológico na aplicação dos testes, cada experimento foi executado cinco vezes, com o objetivo de verificar a consistência dos resultados obtidos e minimizar eventuais variações provocadas por fatores transitórios do ambiente. Após a validação do comportamento das métricas, uma sexta execução foi realizada como amostra definitiva, servindo de base para a elaboração e interpretação dos resultados. Essa etapa adicional reforçou a validade estatística dos dados gerados, assegurando maior confiabilidade nas análises de desempenho conduzidas.

4.7.1. SMOKE TEST

O *smoke test* foi desenvolvido como uma etapa inicial de verificação da disponibilidade básica do serviço exposto no *endpoint* HTTP (Figura 57). O objetivo desse teste é simples: confirmar que o servidor responde corretamente a uma requisição e que o corpo da resposta não está vazio, garantindo que o serviço está minimamente funcional antes de seguir com testes mais complexos, como os de carga e estresse.

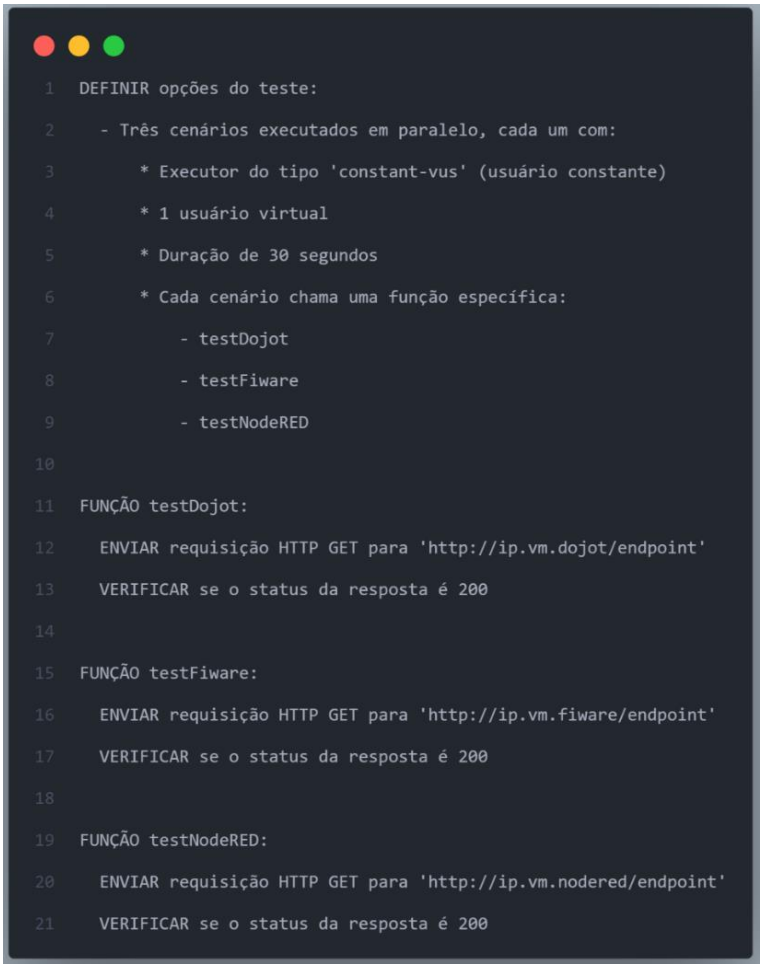
Na prática, o teste foi configurado para executar por apenas 30 segundos com um único usuário virtual (VU). Durante esse curto período, o *script* realiza requisições HTTP do tipo GET para o *endpoint* definido, que representa a URL do serviço que se deseja validar. Em cada resposta recebida, são aplicadas verificações simples (*checks*): primeiro, é testado se o código de status HTTP retornado é 200, que indica sucesso. Em seguida, é verificado se o corpo da resposta não está vazio, o que ajuda a detectar falhas onde o servidor responde, mas sem fornecer dados úteis.

No script em K6, desenvolvemos três funções independentes, cada uma direcionada a um *endpoint* específico:

- /#/devices da plataforma Dojot;
- /v2/entities do FIWARE Orion Context Broker;
- /flows do Node-RED.

Esse teste rodou de forma paralela para os três serviços, confirmando se todos estavam prontos para receber mais requisições. Com ele, identificamos rapidamente se existe algum serviço *offline*, erro de configuração ou se o *endpoint* assinalado está inatingível — economizando tempo antes de testes mais complexos.

Figura 57 – Pseudocódigo do *smoke test* executado.



```
DEFINIR opções do teste:
  - Três cenários executados em paralelo, cada um com:
      * Executor do tipo 'constant-vus' (usuário constante)
      * 1 usuário virtual
      * Duração de 30 segundos
      * Cada cenário chama uma função específica:
          - testDojot
          - testFiware
          - testNodeRED

FUNÇÃO testDojot:
  ENVIAR requisição HTTP GET para 'http://ip.vm.dojot/endpoint'
  VERIFICAR se o status da resposta é 200

FUNÇÃO testFiware:
  ENVIAR requisição HTTP GET para 'http://ip.vm.fiware/endpoint'
  VERIFICAR se o status da resposta é 200

FUNÇÃO testNodeRED:
  ENVIAR requisição HTTP GET para 'http://ip.vm.nodered/endpoint'
  VERIFICAR se o status da resposta é 200
```

Fonte: Elaborado pelo autor (2025).

Esse tipo de teste é essencial como porta de entrada para qualquer ciclo de testes mais detalhado, pois identifica rapidamente problemas básicos de conectividade, indisponibilidade ou falhas de configuração no servidor. Por ser rápido e leve, ele pode ser executado sempre que for necessário, como parte de um pipeline de CI/CD ou antes de testes de carga mais robustos.

4.7.2. LOAD TEST

O *load test* desenvolvido e executado (Figura 58) tem como objetivo avaliar a capacidade de atendimento e a estabilidade do serviço exposto em *endpoints* centrais das aplicações, sob condições próximas de uso real, porém controladas, permitindo identificar gargalos, lentidões ou falhas que possam comprometer a experiência do usuário final.

No início do *script*, são definidas as chamadas configurações de teste (exportadas como *options*), que especificam o cenário de execução por meio de três etapas (*stages*):

1. primeiro, há um *ramp-up*, em que o número de usuários virtuais (VUs) é aumentado gradualmente de 1 até 10 ao longo de 1 minuto;
2. em seguida, ocorre a fase de carga estável, onde 50 VUs permanecem simultaneamente acessando o serviço durante 3 minutos;
3. por fim, há o *ramp-down*, que reduz a quantidade de usuários para 0 em mais 1 minuto.

Figura 58 – Pseudocódigo do *load test*.

```
DEFINIR opções do teste:
    - Etapas (stages):
        * Aumentar usuários de 0 para 10 em 1 minuto
        * Manter 50 usuários por 3 minutos
        * Reduzir usuários para 0 em 1 minuto
    - Limites (thresholds):
        * 95% das requisições devem ter duração menor que 2000 ms
        * Taxa de falhas deve ser menor que 1%

INICIAR função principal do teste:
    PARA cada iteração:
        ENVIAR requisição HTTP GET para 'http://ip.da.vm/endpoint'
        VERIFICAR:
            - Se o status da resposta é igual a 200
        AGUARDAR 1 segundo antes de repetir
```

Fonte: Elaborado pelo autor (2025).

Essa estratégia foi planejada para reproduzir o comportamento de crescimento progressivo no volume de acessos, seguido de uma estabilização e, por fim, a queda da demanda — cenário típico em períodos de pico ou durante janelas de uso intenso do sistema.

Além da configuração das etapas, o *script* define limites de qualidade, chamados de *thresholds*, que funcionam como critérios de aprovação do teste. Foram estabelecidos dois limites principais: o primeiro exige que 95% das requisições HTTP

sejam concluídas em até 500 milissegundos (0,5 segundos), garantindo que o serviço mantenha respostas rápidas mesmo sob carga; o segundo impõe que a taxa de requisições que resultam em erro (código HTTP maior ou igual a 400) seja inferior a 1% do total, assegurando alta disponibilidade e confiabilidade. Caso esses limites sejam ultrapassados, o teste sinaliza que a performance está abaixo do aceitável, orientando ajustes na aplicação ou na infraestrutura.

Durante a execução do teste, cada VU realiza continuamente uma requisição GET ao *endpoint* definido, utilizando a função '*http.get*". Logo após receber a resposta, o *script* executa a função *check*, responsável por validar se o status HTTP retornado é 200 (OK), garantindo que o serviço respondeu corretamente. Por fim, é inserida uma pausa de 1 segundo (sleep(1)), que simula o comportamento natural de um usuário real, evitando que as requisições sejam feitas em sequência sem intervalos e proporcionando um padrão mais próximo do uso cotidiano.

4.7.3. STRESS TEST

Já o *stress test* (Figura 59) foi projetado para ir além da carga prevista, aumentando progressivamente o número de VUs até níveis muito superiores ao esperado em produção. O objetivo foi identificar:

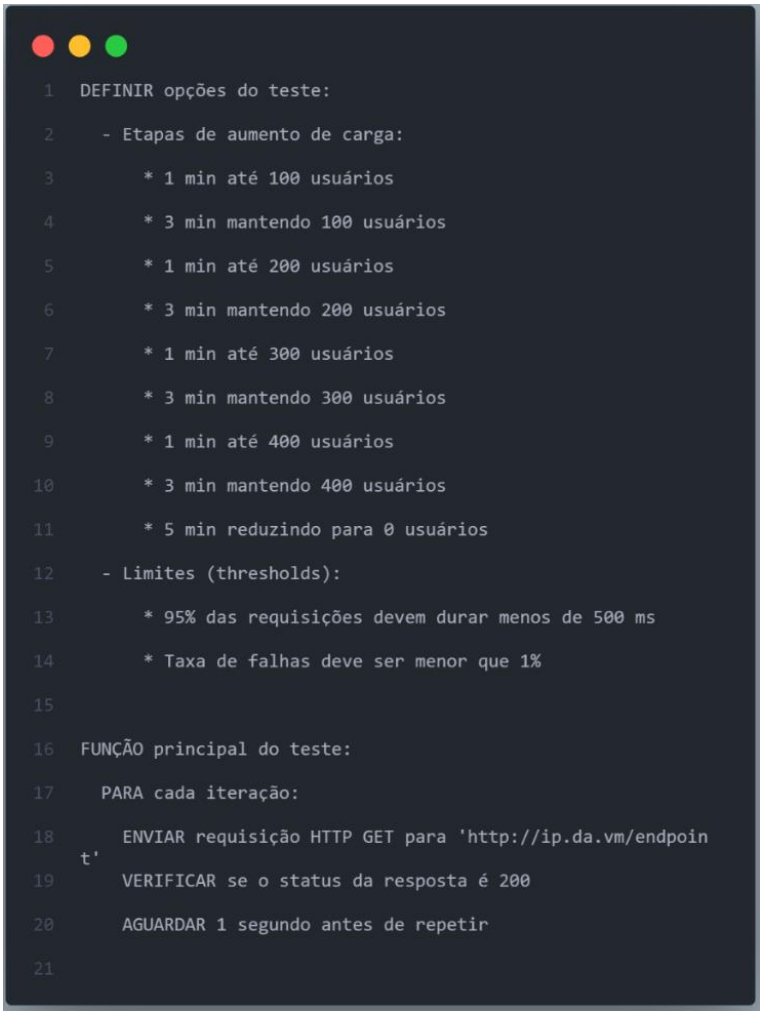
- até onde a aplicação suporta requisições antes de começar a falhar;
- qual é o limite máximo de usuários simultâneos;
- como os serviços se recuperam após sobrecarga.

No *script*, foi configurada uma sequência de etapas (*stages*) que aumenta gradualmente o número de usuários virtuais, mantendo cada patamar por alguns minutos: começa com 100 usuários, depois sobe para 200, 300 e chega até 400 usuários simultâneos. Após o pico, há uma etapa final para reduzir a carga até zero.

Além disso, foram mantidos os *thresholds* já definidos no *load test*, estabelecendo que 95% das requisições devem ser concluídas em menos de 500 ms e que a taxa de falhas não deve ultrapassar 1%. Essa escolha garante um parâmetro consistente para identificar até que ponto o serviço consegue manter níveis aceitáveis de desempenho e estabilidade mesmo sob cargas progressivamente maiores.

Por fim, em cada resposta recebida, é feita uma verificação (*check*) para confirmar que o status HTTP retornado é 200, validando que o serviço continua funcional ao longo de todo o teste. Esse tipo de teste é essencial para planejamento de capacidade, pois revela os limites reais de operação e ajuda a identificar possíveis gargalos.

Figura 59 – Pseudocódigo do *stress test*.



```
DEFINIR opções do teste:
  - Etapas de aumento de carga:
      * 1 min até 100 usuários
      * 3 min mantendo 100 usuários
      * 1 min até 200 usuários
      * 3 min mantendo 200 usuários
      * 1 min até 300 usuários
      * 3 min mantendo 300 usuários
      * 1 min até 400 usuários
      * 3 min mantendo 400 usuários
      * 5 min reduzindo para 0 usuários
  - Limites (thresholds):
      * 95% das requisições devem durar menos de 500 ms
      * Taxa de falhas deve ser menor que 1%

FUNÇÃO principal do teste:
  PARA cada iteração:
    ENVIAR requisição HTTP GET para 'http://ip.da.vm/endpoint'
    VERIFICAR se o status da resposta é 200
    AGUARDAR 1 segundo antes de repetir
```

Fonte: Elaborado pelo autor (2025).

4.8. CONSIDERAÇÕES FINAIS DO CAPÍTULO

O desenvolvimento deste *testbed* permitiu estruturar, validar e analisar uma arquitetura IoT completa, adequada ao suporte de aplicações voltadas para sistemas inteligentes. A construção iniciou-se com a definição da camada física, encarregada da interface com os dispositivos e sensores responsáveis pela coleta de dados do ambiente. Na sequência, a camada de *middleware* foi concebida para promover a

integração e interoperabilidade entre dispositivos heterogêneos, assegurando a fluidez da comunicação entre o *hardware* e as aplicações. A camada de processamento foi implementada para oferecer os recursos necessários ao tratamento, análise e enriquecimento dos dados, enquanto a camada de *software* concentrou as funcionalidades de aplicação, visualização e controle dos sistemas embarcados.

A implantação do ambiente de desenvolvimento incluiu a configuração detalhada das soluções de *software* e a definição da infraestrutura de *hardware*, com ênfase nas arquiteturas dos *middlewares* IoT selecionados. Essa etapa foi decisiva para garantir a consistência e integridade da infraestrutura proposta. A inclusão de um sistema de observabilidade e monitoramento trouxe suporte analítico fundamental para o acompanhamento em tempo real do desempenho computacional dos serviços, viabilizando ajustes com base em métricas técnicas confiáveis. Além disso, os testes de carga permitiram avaliar a estabilidade e a robustez do *testbed* em diferentes cenários de uso, assegurando que a solução projetada atende aos requisitos de desempenho e confiabilidade esperados.

Portanto a metodologia adotada para o desenvolvimento do *testbed* proporcionou uma base sólida e estruturada para a experimentação controlada e a validação de arquiteturas IoT abrangentes. A integração entre as camadas física, de *middleware*, de processamento e de *software* foi fundamental para assegurar a interoperabilidade e a funcionalidade dos sistemas, enquanto a infraestrutura cuidadosamente configurada garantiu a coerência técnica do ambiente.

A integração entre os testes de desempenho e o sistema de observabilidade e monitoramento consolida o *testbed* desenvolvido, permitindo não apenas aplicar cargas sintéticas ao ambiente, mas também compreender, em profundidade, como cada serviço reage a diferentes condições operacionais. Enquanto os testes *smoke*, *load* e *stress* expõem a infraestrutura a cenários que variam do funcionamento mínimo ao limite de saturação, o ecossistema de observabilidade fornece a visibilidade necessária para interpretar esses comportamentos, identificar gargalos e validar a resiliência dos contêineres e máquinas virtuais. Dessa forma, o conjunto dos experimentos e da instrumentação adotada entrega uma avaliação completa, coerente e replicável do desempenho do sistema, concluindo de forma robusta o *testbed* proposto.

Dessa forma, a metodologia descrita não apenas permitiu a construção de um ambiente realista e replicável, mas também forneceu os instrumentos necessários para avaliar, de forma confiável, a capacidade das arquiteturas implantadas em sustentar aplicações IoT em contextos complexos e exigentes.

5. RESULTADOS E ANÁLISES

Esta seção apresenta os resultados obtidos a partir dos cenários experimentais propostos. Inicialmente, são analisados o desempenho computacional de cada máquina virtual e o comportamento dos contêineres que compõem os serviços avaliados. Em seguida, descrevem-se as fases dos testes de performance — verificação (*smoke*), carga média (*load*) e estresse (*stress*), que fundamentaram a geração das cargas sintéticas. Os resultados estatísticos, tanto referentes ao consumo de recursos computacionais (CPU, RAM, disco e rede) quanto aos testes de desempenho, correspondem à média das amostras coletadas ao longo das execuções. Já o intervalo temporal representado nos gráficos abrange aproximadamente 80 minutos, período total necessário para a condução dos cenários de teste e estabilização entre etapas. Essa organização permite uma interpretação clara e estruturada dos efeitos das cargas simuladas sobre o ambiente virtualizado.

5.1. USO DE CPU E RAM POR MÁQUINA VIRTUAL

Para a métrica de uso de CPU por máquina virtual (Figura 60), a VM CentOS apresentou um uso médio de 36,4%, com uma mediana de 34,7%, indicando uma distribuição relativamente simétrica no consumo de CPU. O desvio padrão de 16,5% revela certa variabilidade, mas sem grandes extremos. Os percentis superiores (Π90 = 55,9%, Π95 = 66,2% e Π99 = 75,6%) indicam que, em momentos de maior carga, o consumo se eleva significativamente, o que pode refletir picos de atividade em processos de suporte ao ambiente.

A VM que hospeda o *middleware* Dojot registrou o maior uso médio de CPU (58%), com valores de percentis também elevados, atingindo 87,2% no Π99. A mediana coincide com a média (58%), sugerindo uma distribuição estável e consistente ao redor desse valor. O desvio padrão de 10,2% é moderado, indicando uma carga de CPU mais contínua e menos sujeita a variações extremas. Esses dados são coerentes com o papel intensivo do Dojot em operações de gerenciamento e processamento de mensagens IoT, conforme os cenários de carga propostos.

A VM FIWARE teve o menor uso médio de CPU entre as VMs analisadas (16,2%), mas com um desvio padrão alto (16,4%), o que revela alta variabilidade. A

mediana foi de apenas 8,43%, indicando que, na maior parte do tempo, o uso da CPU foi baixo. No entanto, os percentis mais altos (Π90 = 48,7%, Π95 = 51,5%, Π99 = 55,9%) apontam para eventos ocasionais de uso intensivo. Remetendo a esporadicidade causada pelos testes conduzidos, influenciada por picos de solicitação ou por serviços específicos da arquitetura implantada sendo ativados em momentos pontuais.

A VM Node-RED apresentou um uso médio de CPU elevado (54%), com uma mediana um pouco inferior (51,8%) e um desvio padrão de 14,2%, indicando uma carga relativamente constante, mas com flutuações. Os percentis superiores são expressivos (Π99 = 93,4%), sugerindo que, sob certas condições, o Node-RED pode atingir níveis muito altos de utilização de CPU. Esse comportamento é consistente com cenários em que o Node-RED processa múltiplos fluxos de dados ou interações simultâneas, exigindo mais do sistema.

Figura 60 – Uso de CPU por VM.

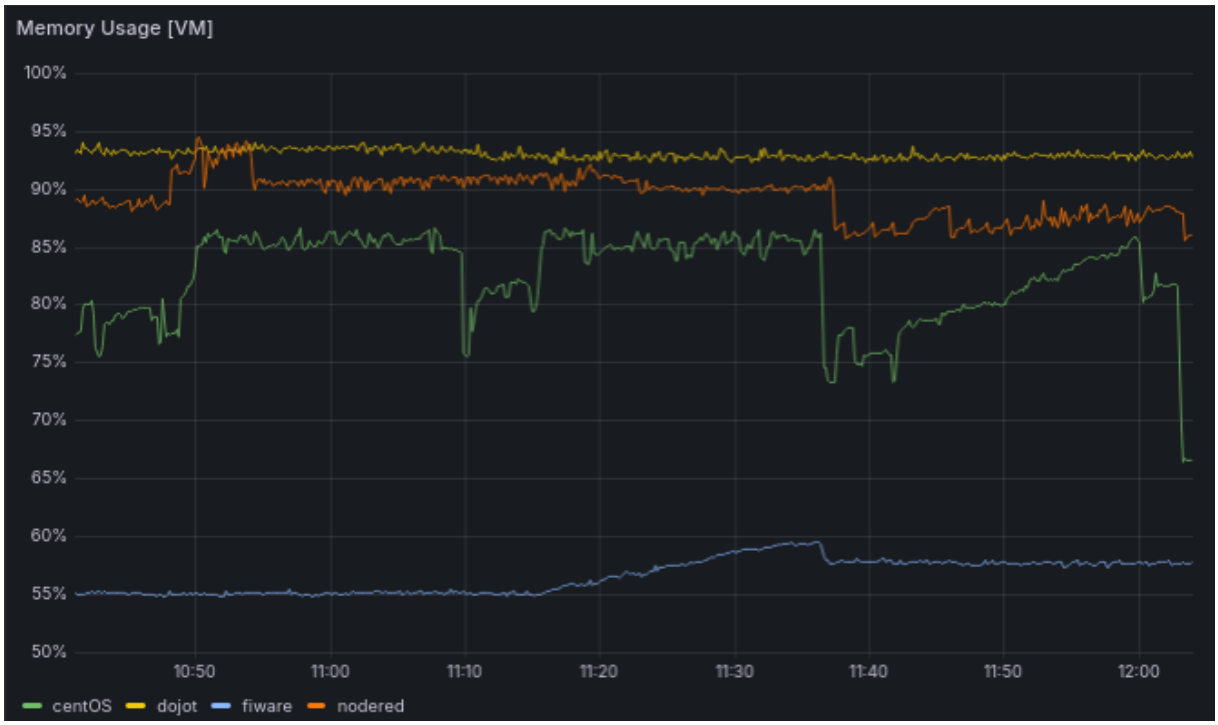
Fonte: Elaborado pelo autor (2025).

Quanto à métrica de uso de RAM (Figura 61), a VM CentOS apresentou um custo elevado, com média de 82,5% e mediana de 84,3%, sugerindo que o sistema opera com memória constantemente ocupada. O desvio padrão é baixo (3,78%), e os percentis (Π90 a Π99) oscilam pouco, entre 86,0% e 86,5%. Isso indica um padrão de

uso estável e possivelmente próximo do limite ideal de alocação. A constância e consistência dos valores reflete o funcionamento intensivo da pilha monitoramento e às operações de infraestrutura do ambiente da VM.

A VM Dojot apresentou o maior uso de RAM de todo o conjunto: média e mediana de 93%, com mínima variabilidade (desvio padrão de apenas 0,38%). Os percentis superiores atingem até 93,9%, evidenciando um uso extremamente constante e alto da memória. Esse padrão reforça o caráter intensivo do *middleware* Dojot, que mantém diversas estruturas e *buffers* em memória para garantir desempenho em tempo real. Contudo, esse nível de uso pode indicar necessidade de monitoramento detalhado e extensivo para evitar gargalos futuros.

Figura 61 – Uso de RAM por VM.



Fonte: Elaborado pelo autor (2025).

A VM FIWARE teve um uso de RAM moderado, com média de 56,6% e mediana de 56,7%. O desvio padrão de 1,45% aponta para estabilidade, e os percentis superiores permanecem abaixo de 60%. Isso sugere que o *middleware* FIWARE mantém um perfil de consumo de memória eficiente, com baixa variabilidade, mesmo considerando as flutuações de carga. Isso pode sugerir que a arquitetura do FIWARE, com um número reduzido de microsserviços em relação à plataforma Dojot, pode estar bem ajustada ao ambiente, consumindo apenas os recursos necessários.

Já a VM Node-RED utilizou em média 89,5% da RAM, com mediana de 90%, e um desvio padrão de 1,86%, indicando também estabilidade, mas em um patamar alto de uso. Os percentis Π90 a Π99 vão de 91,2% a 93,8%, mostrando que o uso de memória é consistentemente elevado, semelhante ao Dojot. Isso pode refletir a carga de execução de fluxos simultâneos e a manutenção de estados internos de lógica.

Por conseguinte, o uso de CPU apresentou maior variabilidade entre as VMs, com destaque para o Dojot e o Node-RED, que mantêm cargas médias elevadas e percentis superiores expressivos, indicando atividades constantes e picos de processamento. Em contraste, o FIWARE demonstrou um perfil mais esporádico, com média baixa, mas com momentos de alto consumo. Já o uso de RAM mostrou-se mais estável entre as VMs, embora com níveis elevados nas máquinas que executam o Dojot e o Node-RED, sugerindo que esses serviços mantêm muitos dados em memória para garantir desempenho.

A Tabela 8 apresenta um sumário dos índices estatísticos para as métricas de uso de CPU e RAM por VM.

Tabela 8 – Sumário dos índices estatítiscos para as métricas de uso de CPU e uso de RAM por VM.

Métrica	VM	μ	μδ	σ	Π90	Π95	Π99
Uso de CPU (%)	CentOS	36,40	34,70	16,50	55,90	66,20	75,60
	Dojot	58,00	58,00	10,20	70,50	76,60	87,20
	FIWARE	16,20	8,43	16,40	48,70	51,50	55,90
	Node-RED	54,00	51,80	14,20	76,60	81,80	93,40
Uso de RAM (%)	CentOS	82,50	84,30	3,78	86,00	86,20	86,50
	Dojot	93,00	93,00	0,38	93,60	93,70	93,90
	FIWARE	56,60	56,70	1,45	58,20	59,00	59,40
	Node-RED	89,50	90,00	1,86	91,20	92,20	93,80

Fonte: Elaborado pelo autor (2025).

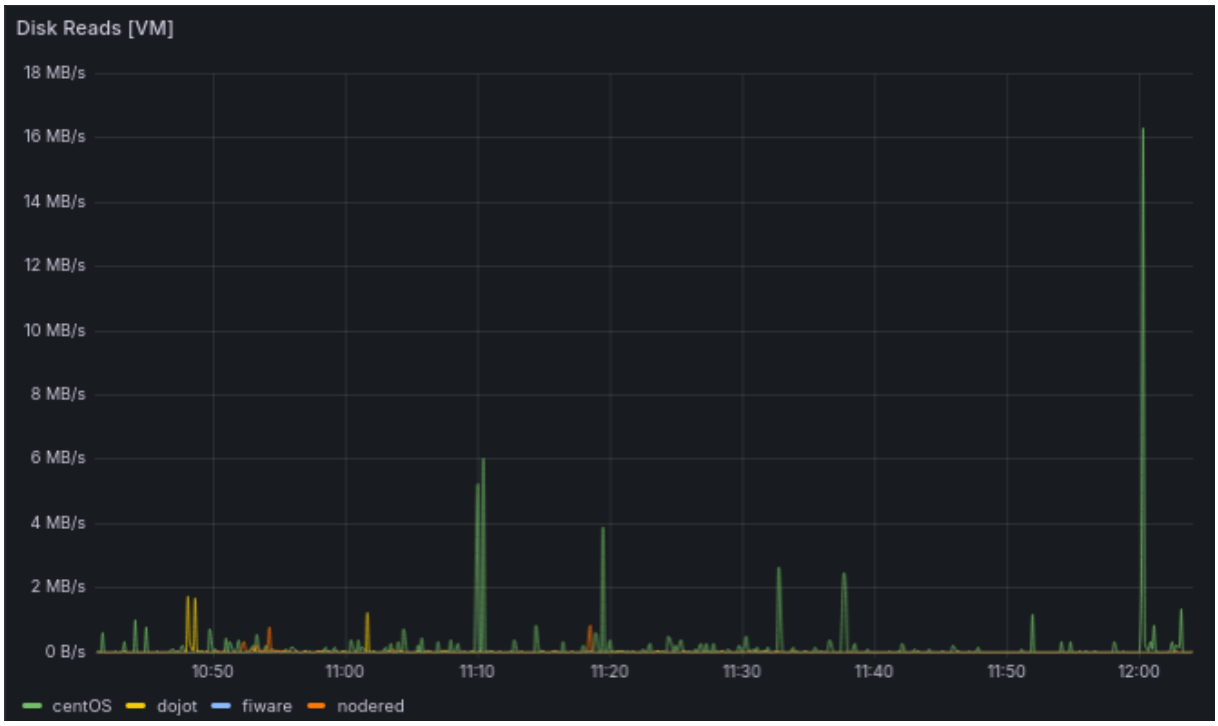
5.2. LEITURA E ESCRITA EM DISCO POR MÁQUINA VIRTUAL

Para métrica de leitura em disco (Figura 62), a VM CentOS apresentou a maior média de (132 kB/s) entre todas as máquinas, apesar de uma mediana igual a zero, o que indica que em grande parte do tempo não houve leitura significativa. O desvio padrão bastante elevado (860 kB/s) e os percentis superiores — especialmente o Π99 com 2460 kB/s — revelam a ocorrência de picos intensos e pontuais de leitura em disco. Esse comportamento, novamente, reflete a natureza da pilha de

monitoramento, que realiza operações contínuas de leitura, como *scrape* de métricas e ingestão de dados.

A VM Dojot apresentou uma média de leitura muito baixa (10,7 kB/s) e mediana zero, o que indica que o uso de leitura em disco é raro. No entanto, os percentis mostram um comportamento inconsistente: o Π90 atinge 410 kB/s, enquanto o Π95 cai para 3,69 kB/s. Essa inconsistência pode ser resultado de variações nos dados amostrados ou eventos isolados de leitura mais intensa. Ainda assim, o uso geral de leitura em disco pelo Dojot é reduzido, provavelmente devido ao uso intensivo de memória e *cache* para tratamento de mensagens em tempo real.

Figura 62 – Leitura em disco por VM.



Fonte: Elaborado pelo autor (2025).

A VM que executa o FIWARE apresentou valores nulos em todas as estatísticas de leitura em disco, o que pode parecer, à primeira vista, um comportamento anômalo. No entanto, esse padrão é coerente com a arquitetura dos componentes FIWARE utilizados, como o Orion Context Broker e o STH-Comet. O Orion mantém dados de contexto em memória e emprega mecanismos de *cache* e índices no MongoDB para reduzir leituras frequentes em disco. Já o STH-Comet realiza agregações pré-processadas das séries temporais e mantém essas estruturas prontas para consulta, minimizando ainda mais a necessidade de leitura. Assim, a ausência de leitura

detectada indica um uso eficiente da memória e da persistência otimizada, concentrando os acessos em escrita contínua, como evidenciado nos dados correspondentes.

No caso do Node-RED, a média de leitura em disco foi baixa (5,48 kB/s), com mediana também nula, reforçando um comportamento ocioso na maior parte do tempo. Porém, os percentis superiores revelam picos pontuais de leitura: o Π95 chega a 819 kB/s e o Π99 a 99,5 kB/s. Isso sugere que, em certos momentos, o Node-RED acessa arquivos, fluxos ou persistências locais com maior intensidade. Apesar desses picos, a leitura em disco não é um componente dominante de sua carga de trabalho, provavelmente devido ao seu modelo de execução orientado a eventos e memória.

Para a métrica de escrita em disco (Figura 63), A VM CentOS apresentou uma média de 36,8 kB/s, com mediana de 20 kB/s e um desvio padrão de 124 kB/s. Esses valores indicam uma carga de escrita moderada, com variações ocasionais e picos de até 112 kB/s (Π99). Esse comportamento é compatível com o papel da VM, já que ferramentas como o Prometheus e o InfluxDB realizam gravações periódicas de métricas, séries temporais e *snapshots*, enquanto o Grafana pode registrar configurações e *dashboards*. O cAdvisor, por sua vez, coleta métricas em tempo real sobre os contêineres e contribui para o volume de dados escritos. Apesar de não serem volumes massivos, os dados revelam uma atividade constante de escrita típica de sistemas de monitoramento operando em tempo real.

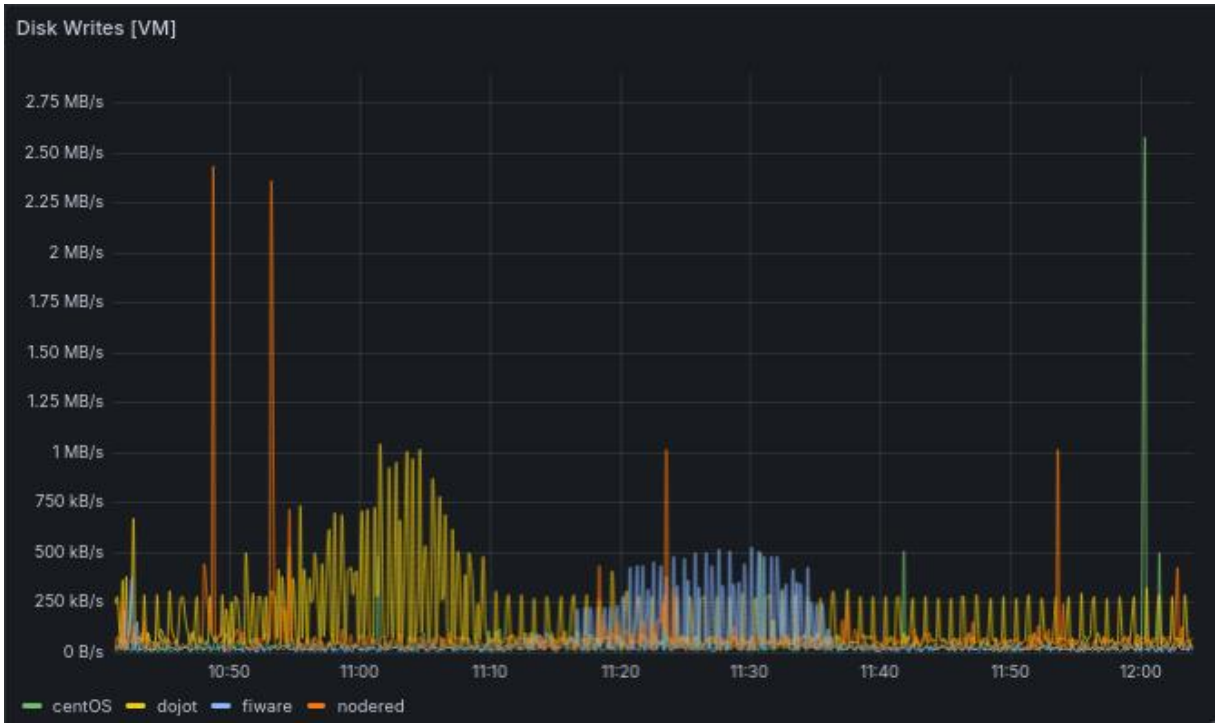
A escrita em disco do Dojot foi a mais intensa entre todas as VMs, com média de 151 kB/s e uma mediana de 66,4 kB/s. O desvio padrão de 175 kB/s, aliado aos altos valores nos percentis superiores (Π99 = 926 kB/s), revela que o Dojot realiza operações frequentes e volumosas de escrita em disco. Isso é coerente com o seu papel de *middleware* central no gerenciamento de dispositivos IoT, já que frequentemente armazena *logs*, mensagens, registros de dispositivos e dados persistentes em disco.

A VM FIWARE também apresentou atividade considerável de escrita em disco, com média de 51,5 kB/s e mediana de 18 kB/s. O desvio padrão (105 kB/s) e os percentis mais altos (P99 = 494 kB/s) demonstram que, embora a maior parte do tempo a carga seja baixa, há momentos de escrita intensa. Isso é compatível com o comportamento esperado de componentes do FIWARE que fazem persistência de

dados, como bancos de dados (e.g., Orion Context Broker, QuantumLeap) e logs de serviços.

A escrita em disco do Node-RED também foi significativa, com média de 67,7 kB/s e mediana de 41,4 kB/s. O desvio padrão elevado (170 kB/s) e os valores altos nos percentis (P99 = 439 kB/s) indicam que o sistema realiza gravações frequentes e com variação de intensidade. Esse padrão é condizente com o fluxo contínuo de mensagens, registros de execução de fluxos e persistência local de dados processados, especialmente em ambientes com alto tráfego ou múltiplas integrações ativas.

Figura 63 – Escrita em disco por VM.



Fonte: Elaborado pelo autor (2025).

A análise das métricas de leitura e escrita em disco revela perfis distintos de uso entre as VMs, diretamente relacionados às funções de cada uma no ambiente. A VM CentOS apresentou picos significativos de leitura em disco e uma escrita moderada e constante, compatível com a coleta e persistência contínua de métricas e séries temporais. A VM Dojot exibiu a maior intensidade de escrita, reflexo do alto volume de dados processados e armazenados pelo *middleware* IoT, enquanto sua leitura foi pontual e ocasional. A VM FIWARE praticamente não realizou leituras em disco, comportamento esperado devido ao uso de *cache* e agregações em memória

pelos componentes Orion e STH-Comet, mas apresentou escrita contínua e com picos relevantes, caracterizando uma persistência eficiente de dados contextuais e séries históricas. Já a VM Node-RED teve leituras esporádicas, com picos em momentos específicos, e uma escrita consistente, revelando sua atuação como intermediador de fluxos e processador de mensagens com persistência dinâmica. Esses comportamentos reforçam a necessidade de um provisionamento de I/O alinhado às funções específicas de cada serviço, especialmente em arquiteturas IoT distribuídas e orientadas a contêineres.

A Tabela 9 apresenta um sumário dos índices estatísticos para as métricas de de leitura e escrita em disco por VM.

Tabela 9 – Sumário dos índices estatísticos para as métricas de leitura e escrita em disco por VM.

Métrica	VM	μ	$\mu\delta$	σ	Π90	Π95	Π99
Leitura em disco (kB/s)	CentOS	132	0	860	224	367	2460
	Dojot	10,7	0	122	410	3,69	44,2
	FIWARE	0	0	0	0	0	0
	Node-RED	5,48	0	53	0	819	99,5
Escrita em disco (kB/s)	CentOS	36,8	20	124	64,8	75	112
	Dojot	151	66,4	175	305	492	926
	FIWARE	51,5	18	105	73,3	340	494
	Node-RED	67,7	41,4	170	93,8	137	439

Fonte: Elaborado pelo autor (2025).

5.3. TRÁFEGO DE ENTRADA E SAÍDA DE REDE POR MÁQUINA VIRTUAL

Para a métrica de tráfego de entrada de rede (Figura 64), a VM CentOS registrou o maior entre todas as VMs, com uma média de 587,0 kB/s e um desvio padrão elevado de 702,0 kB/s. A mediana foi significativamente mais baixa (228,0 kB/s), o que indica que boa parte do tempo o tráfego se manteve em níveis moderados, mas com picos altos nos percentis superiores — chegando a 2755,0 kB/s no Π99. Esse comportamento é coerente com o papel da CentOS como nó de monitoramento, recebendo continuamente grandes volumes de métricas dos demais contêineres por meio do Prometheus e do cAdvisor, além de requisições via Grafana e fluxos de dados do Proxmox VE armazenados no InfluxDB, refletindo a natureza centralizada da coleta de dados no ambiente.

A VM Dojot apresentou tráfego de entrada relativamente baixo, com média de 13,6 kB/s e mediana de apenas 1,07 kB/s. O desvio padrão (24,0 kB/s) e os percentis superiores (P99 = 85,8 kB/s) indicam que o tráfego de entrada é geralmente baixo, mas com picos ocasionais, possivelmente associados a bursts de dados de sensores ou dispositivos IoT sendo simulados. O padrão é compatível com o modelo publish-subscribe do Dojot, no qual os dados recebidos são rapidamente processados e repassados para outras aplicações ou armazenamentos.

Figura 64 – Tráfego de entrada de rede por VM.

Fonte: Elaborado pelo autor (2025).

Assim como o Dojot, a VM FIWARE apresentou baixa taxa de entrada, com média de 12,8 kB/s e mediana de apenas 471 *bytes* por segundo, sugerindo períodos longos com tráfego mínimo. Apesar disso, os percentis $\Pi 95$ e $\Pi 99$ atingem 53,2 e 75,4 kB/s, respectivamente, mostrando que há momentos específicos de maior carga, provavelmente quando há atualizações de contexto ou registros sendo processados pelo Orion Context Broker. Isso reforça a ideia de que o FIWARE opera de forma orientada a eventos, com tráfego de entrada intermitente e dependente do número de interações contextuais.

A VM Node-RED exibiu uma média de tráfego de entrada superior ao Dojot e ao FIWARE (35,8 kB/s), com uma mediana de 12,4 kB/s. O desvio padrão elevado

(132,0 kB/s) e os percentis superiores (Π99 = 92,5 kB/s) indicam que o Node-RED recebe cargas variáveis ao longo do tempo, o que é coerente com seu papel de processador de fluxos e integrador de dados entre serviços.

Dessa forma, a análise do tráfego de rede evidencia o papel funcional e a carga de comunicação de cada VM no ambiente. A VM CentOS, encarregada da coleta e visualização de métricas, concentra o maior volume de tráfego de entrada, refletindo sua função como ponto central de monitoramento. Seu tráfego de saída, no entanto, é modesto e estável, indicando um padrão de consumo passivo de dados.

Em contraste, a VM Dojot apresenta um tráfego de saída significativamente superior ao de entrada, comportamento típico de um *middleware* que publica dados e notificações para múltiplos consumidores, mas que recebe dados de forma mais esporádica.

A VM FIWARE demonstra um perfil de rede mais discreto, com baixos volumes tanto de entrada quanto de saída, o que está alinhado à sua operação baseada em eventos contextuais, acionada sob demanda.

Já a VM Node-RED se destaca pelo tráfego de saída mais elevado entre todas, confirmando seu papel como integrador ativo de fluxos de dados, com distribuição para diferentes destinos. Esses padrões de comunicação reforçam a importância de alinhar o provisionamento de rede à função operacional de cada componente, especialmente em arquiteturas IoT onde a troca de da dos em tempo real é essencial para o desempenho e a confiabilidade do sistema.

A Tabela 10 apresenta um sumário dos índices estatísticos para as métricas de tráfego de entrada e saída de rede por VM.

Tabela 10 – Sumário das métricas de tráfego de entrada e saída de rede por VM.

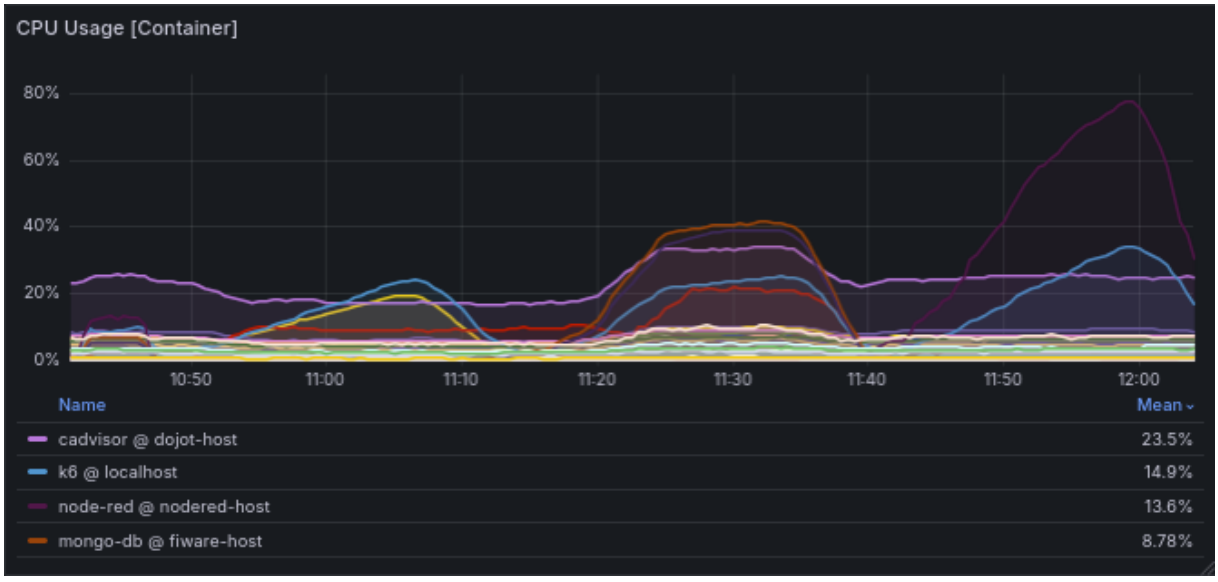
Métrica	**VM**	**μ**	**μδ**	**σ**	**Π90**	**Π95**	**Π99**
Tráfego de entrada (kB/s)	CentOS	587	228	702	1894	2243	2755
	Dojot	13,6	1,07	24	58,6	70,2	85,8
	FIWARE	12,8	0,471	20,8	47,5	53,2	75,4
	Node-RED	35,8	12,4	132	64,2	82,8	92,5
Tráfego de saída (kB/s)	CentOS	46	46,5	32	87,3	98,7	125
	Dojot	175	65,1	266	616	750	888
	FIWARE	22,9	15,1	24,3	64,1	79,9	87,5
	Node-RED	415	26,4	724	1790	2120	2630

Fonte: Elaborado pelo autor (2025).

5.4. USO DE CPU E RAM POR CONTÊINER

A análise do uso de CPU (Figura 65) revela padrões distintos entre os contêineres das diferentes VMs. Na VM CentOS, o contêiner “k6” apresentou uma média de uso de CPU de 0,149%, com valores bastante estáveis e picos moderados (Π99 de 0,335%), o que está alinhado ao seu papel como gerador de carga controlada nos testes. Já o contêiner “cadvisor”, executando na VM Dojot, teve uma média levemente superior (0,235%), com picos similares, indicando seu esforço contínuo de coleta de métricas, mas sem sobrecarga. O contêiner “node-red”, na VM Node-RED, mostrou comportamento mais variável: a mediana de 0,003% contrasta fortemente com o desvio padrão elevado (0,237%) e com os picos (Π99 de 0,767%), sinalizando que, embora o uso médio seja baixo, há momentos de alta demanda computacional, provavelmente associados ao processamento de fluxos de dados intensivos. Por fim, o contêiner “mongo-db”, na VM FIWARE, também demonstrou esse padrão intermitente: média e mediana baixas (0,088% e 0,006%, respectivamente), mas com picos notáveis (P99 de 0,414%), sugerindo acessos eventuais ao banco de dados durante a execução dos testes.

Figura 65 – Uso de CPU por contêiner.

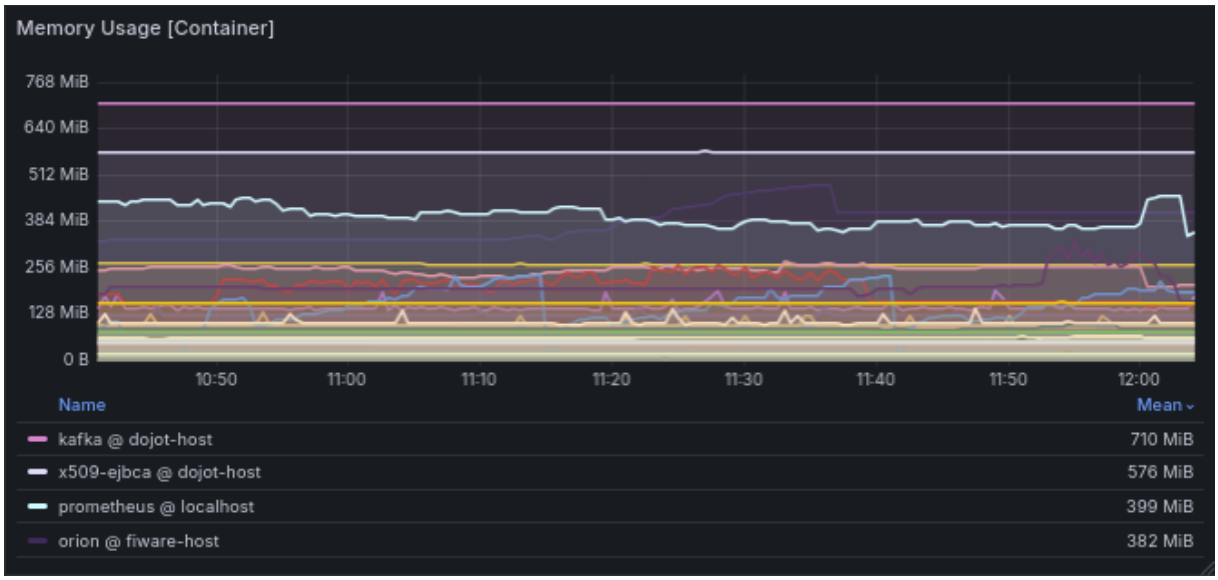


Fonte: Elaborado pelo autor (2025).

Quanto ao uso de RAM (Figura 66), observa-se uma divisão mais clara entre os contêineres com consumo constante e os que apresentam variação significativa. O contêiner “prometheus”, na VM CentOS, utilizou em média 6,49% da memória da VM,

com picos de até 7,39%, o que demonstra um uso relativamente estável para uma ferramenta de coleta contínua de métricas. O contêiner "kafka", na VM Dojot, apresentou o maior consumo de RAM entre os contêineres analisados, com média de 11,56% e praticamente nenhum desvio, evidenciando a alta e constante demanda de memória dessa ferramenta de mensageria. Na VM FIWARE, o contêiner "orion" teve média de uso de 6,22%, mas com desvio padrão considerável (0,77%) e picos de até 7,88%, indicando oscilações no uso de memória conforme os eventos contextuais são processados. Por fim, o contêiner "node-red", na VM Node-RED, teve o menor uso relativo de RAM (3,38% em média), mas com aumentos eventuais até 5,05%, o que reflete sua função como orquestrador leve, com picos pontuais de uso durante execuções de fluxos mais complexos. Esses padrões de uso reforçam a importância de se considerar não apenas o consumo médio, mas também os picos e variações no planejamento de recursos para ambientes IoT.

Figura 66 – Uso de RAM por contêiner.



Fonte: Elaborado pelo autor (2025).

Consequentemente, a análise do uso de CPU e RAM por contêiner revela comportamentos distintos conforme a função de cada serviço nas respectivas VMs. No consumo de CPU, os contêineres que operam como monitoramento ou *middleware* — como o "cadvisor" (Dojot) e o "node-red" (Node-RED) — demonstraram picos de uso mais acentuados, refletindo sua carga contínua de processamento ou reatividade a eventos, com destaque para o Node-RED, o qual foi implantado através de um contêiner único.

Em contraste, contêineres de banco de dados como “mongo-db” (FIWARE) apresentaram uso médio discreto, mas com variações que indicam momentos de maior atividade. Quanto ao uso de RAM, destaca-se o “kafka" no Dojot, com alocação praticamente constante, sinalizando exigência previsível e elevada de memória. O contêiner “orion” (FIWARE) também apresentou consumo expressivo e variável, compatível com seu papel de gerenciador de contexto. Já o “prometheus” (CentOS) demonstrou uso elevado e consistente de memória, esperado para sua função de coleta contínua de métricas.

Esses padrões evidenciam que, enquanto o uso de CPU varia conforme a carga momentânea, o uso de RAM tende a ser mais estável e previsível, refletindo a natureza da aplicação hospedada.

A Tabela 11 apresenta um sumário dos indicadores estatísticos para as métricas de uso de CPU e RAM por contêiner.

Tabela 11 – Sumário dos índices estatístiscos para as métricas de uso de CPU e RAM por contêiner.

Métrica	Contêiner	μ	$\mu\delta$	σ	Π90	Π95	Π99
Uso de CPU (%)	k6 @ CentOS	0,149	0,143	0,091	0,252	0,309	0,335
	cadvisor @ Dojot	0,235	0,242	0,053	0,333	0,335	0,342
	node-red @ Node-RED	0,136	0,003	0,237	0,589	0,714	0,767
	mongo-db @ FIWARE	0,088	0,006	0,141	0,390	0,409	0,414
Uso de RAM (%)	prometheus @ CentOS	6,49 %	6,38 %	0,45 %	7,19 %	7,26 %	7,39 %
	kafka @ Dojot	11,56 %	11,54 %	0,03 %	11,60 %	11,60 %	11,60 %
	orion @ FIWARE	6,22 %	6,38 %	0,77 %	7,21 %	7,71 %	7,88 %
	node-red @ Node-RED	3,38 %	3,22 %	0,45 %	3,81 %	4,54 %	5,05 %

Fonte: Elaborado pelo autor (2025).

5.5. LEITURA E ESCRITA EM DISCO POR CONTÊINER

A análise da leitura em disco por contêiner (Figura 67) revela que a maior carga recai sobre o contêiner “prometheus”, na VM CentOS, com uma média de 161 kB/s e um desvio padrão elevado (1960 kB/s), o que indica picos esporádicos de atividade. Ainda que a mediana e os percentis superiores estejam zerados, os valores extremos sugerem momentos pontuais de varredura intensiva — comportamento esperado de um sistema de monitoramento que realiza coletas periódicas. No ambiente Dojot, o contêiner “backstage” apresentou atividade mínima de leitura, com média de apenas 6,19 kB/s e mediana e percentis nulos, indicando acessos raros ou esporádicos ao

disco. Já nas VMs FIWARE e Node-RED, não houve registros de leitura em disco nos contêineres monitorados, o que pode ser consequência tanto de um uso extremamente leve quanto da ausência de componentes com operações de leitura relevantes nesses ambientes.

Figura 67 – Leitura em disco por contêiner.

Fonte: Elaborado pelo autor (2025).

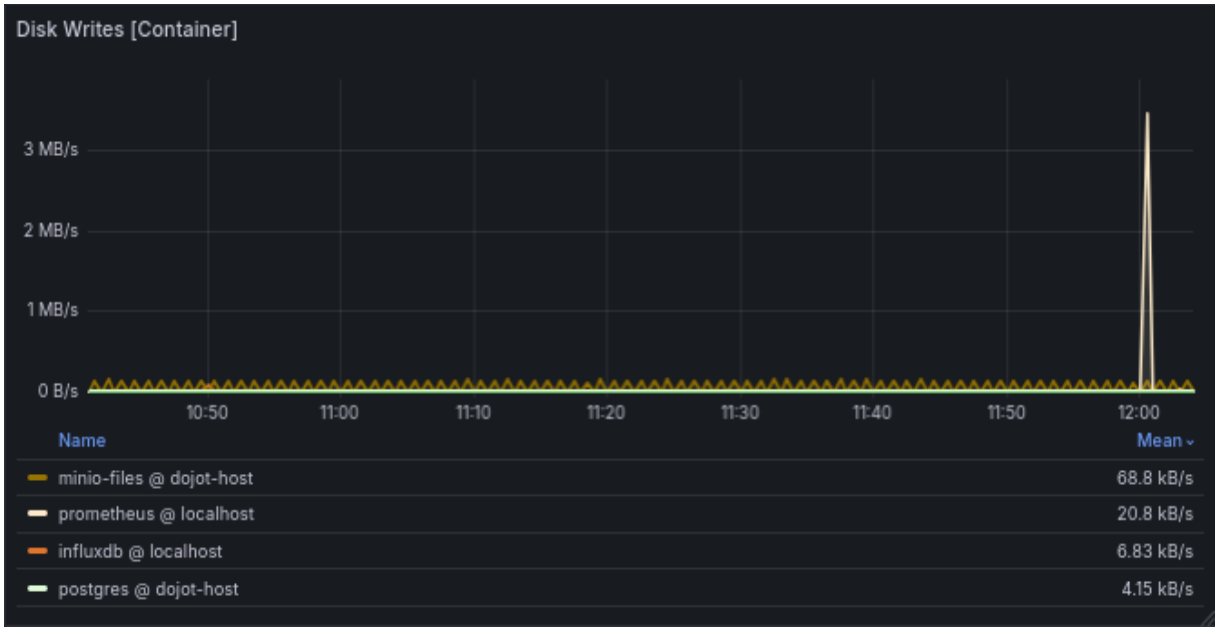
Quanto à escrita em disco (Figura 68), novamente o destaque vai para a VM CentOS, com o contêiner “prometheus” registrando uma média de 20,8 kB/s. Embora os percentis permaneçam zerados, o desvio padrão relativamente alto (268 kB/s) indica a ocorrência de momentos isolados de escrita mais intensa, provavelmente relacionados à persistência de séries temporais. Na VM Dojot, o contêiner “minio-files” registrou uma escrita média considerável (68,8 kB/s), com percentis superiores elevados (Π90 a Π99 entre 144 e 153 kB/s), refletindo operações constantes de armazenamento de arquivos — consistentes com o papel do Minio como servidor de objetos. O contêiner “mongo-db” na VM FIWARE mostrou um padrão de escrita moderado e estável, com média de 3,66 kB/s e crescimento suave nos percentis, condizente com a natureza transacional do banco de dados. Já a ausência de dados sobre escrita na VM Node-RED pode indicar que os contêineres desta instância dependem majoritariamente de operações em memória ou comunicação em rede, sem persistência local relevante.

Em vista disso, a análise das métricas de leitura e escrita em disco por contêiner evidencia que o comportamento de I/O está fortemente relacionado às funções específicas dos serviços. A ausência de leitura em muitos contêineres e a predominância de escrita em pontos específicos reforçam a ideia de que o tráfego de disco está concentrado em serviços voltados ao armazenamento e monitoramento, enquanto contêineres com papel mais reativo ou transitório apresentam baixa demanda por I/O persistente.

No entanto, como esta seção de análise abrange tanto operações de leitura quanto de escrita, os contêineres com maior volume de leitura não necessariamente serão os mesmos que apresentam maior volume de escrita. Isso ocorre porque, dependendo da arquitetura da plataforma, diferentes serviços assumem responsabilidades distintas dentro do fluxo de dados.

O mesmo padrão é observado na análise de tráfego de entrada e saída, onde a assimetria entre funções internas também resulta em comportamentos diferenciados entre os contêineres.

Figura 68 – Escrita em disco por VM.



Fonte: Elaborado pelo autor (2025).

A Tabela 12 apresenta um sumário dos indicadores estatísticos para as métricas de leitura e escrita em disco por contêiner.

Tabela 12 – Sumário dos índices estatísticos para as métricas de leitura e escrita em disco por contêiner.

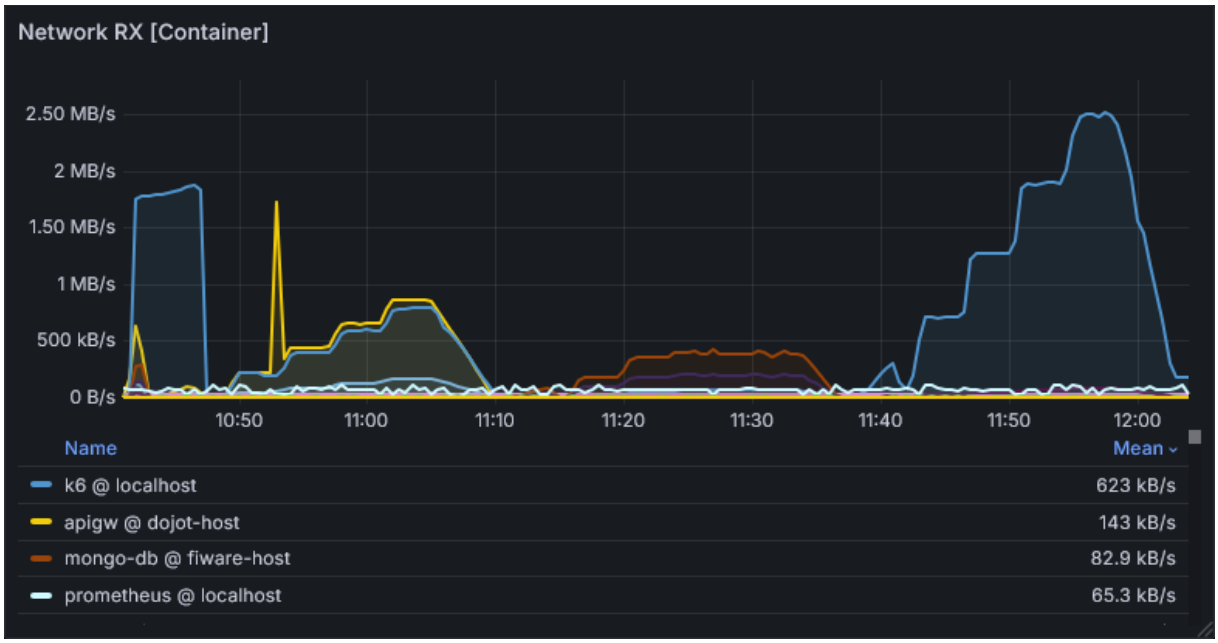
Métrica	Contêiner	μ	μδ	σ	Π90	Π95	Π99
Leitura em disco (kB/s)	prometheus @ CentOS	161	0	1960	0	0	489
	backstage @ Dojot	6,19	0	79,7	0	0	0
	- @ FIWARE	-	-	-	-	-	-
	- @ Node-RED	-	-	-	-	-	-
Escrita em disco (kB/S)	prometheus @ CentOS	20,8	0	268	0	0	0
	minio-files @ Dojot	68,8	0	69,4	144	150	153
	mongo-db @ FIWARE	3,66	3,14	1,65	5,68	8,03	8,56
	- @ Node-RED	-	-	-	-	-	-

Fonte: Elaborado pelo autor (2025).

5.6. TRÁFEGO DE ENTRADA E SAÍDA DE REDE POR CONTÊINER

A análise do tráfego de entrada (Figura 69) evidencia diferenças significativas entre os contêineres avaliados. O contêiner “k6”, da VM CentOS, responsável pela geração de carga, apresentou o maior volume médio de tráfego de entrada (623 kB/s), com picos expressivos que chegaram a 2.5 MB/s (2560 kB/s) nos momentos de maior atividade (P99), o que é esperado, considerando sua função no stress test.

Figura 69 – Tráfego de entrada de rede por contêiner.



Fonte: Elaborado pelo autor (2025).

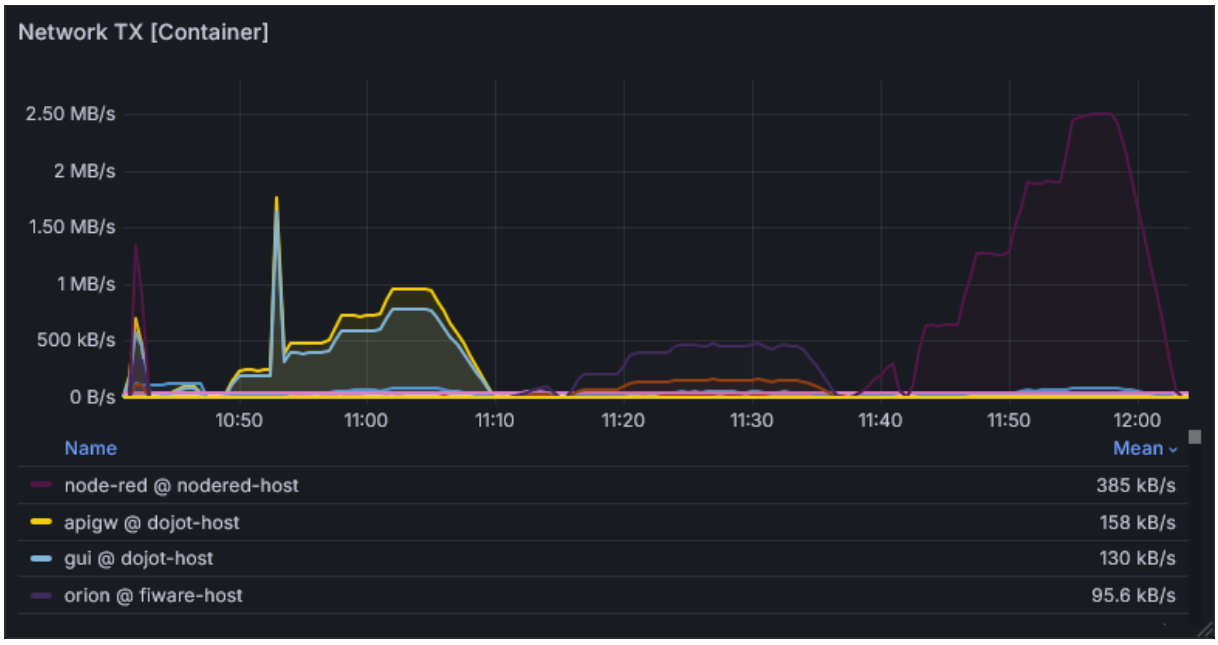
Já o contêiner “apigw”, principal interface de entrada na plataforma Dojot, apresentou uma média de 143 kB/s, com valor mediano extremamente baixo (0.0573

kB/s), sugerindo que a maioria das requisições foi breve, mas com variações de carga que alcançaram até 865 kB/s. Isso reflete um comportamento esporádico e variável de acesso ao *gateway*, típico em cenários com múltiplas requisições paralelas.

O contêiner “node-red”, da VM Node-RED, registrou uma média de 82.9 kB/s, também com mediana muito próxima de zero (0.136 kB/s), o que indica que, apesar de momentos de maior carga (Π99 de 405 kB/s), o fluxo de entrada foi esparso e concentrado em picos. Por sua vez, o “mongo-db”, da VM FIWARE, teve o menor tráfego de entrada entre os analisados, com média de apenas 12.6 kB/s, o que é compatível com sua função de banco de dados que recebe dados processados ou filtrados. No entanto, os picos de até 81.2 kB/s mostram que, em determinados momentos, houve maior atividade de escrita na base de dados, provavelmente relacionada ao armazenamento de eventos ou entidades.

Em relação ao tráfego de saída, o k6 @ CentOS novamente lidera em volume, com média de 385 kB/s e picos chegando a 2.5 MB/s (2560 kB/s), o que reforça seu papel ativo como gerador de tráfego. É notável, no entanto, que sua mediana foi 0 kB/s, indicando períodos em que não houve transmissão de dados, intercalados por picos intensos — um padrão típico de testes de carga.

Figura 70 – Tráfego de saída de rede por contêiner.



Fonte: Elaborado pelo autor (2025).

O contêiner “apigw” aparece logo em seguida, com média de 158 kB/s e Π99 de 959 kB/s, refletindo a natureza dinâmica da comunicação com os demais serviços e consumidores. Já o “orion”, atuando como *broker* de contexto, exibiu média de 95.6 kB/s e variações até 471 kB/s, indicando uma comunicação constante e escalável, coerente com seu papel de distribuir atualizações de contexto para os assinantes.

Por fim, o “node-red” apresentou tráfego de saída mais modesto, com média de 42.5 kB/s, mas com uma mediana elevada (41.5 kB/s), sugerindo uma constância maior nas respostas aos fluxos recebidos. Seus picos de saída chegaram a 120 kB/s, o que demonstra uma operação eficiente e estável mesmo em situações de maior demanda. Esses dados corroboram a adequação do Node-RED para cenários IoT com cadência moderada de mensagens e respostas previsíveis. A Tabela 13 apresenta um sumário dos indicadores estatísticos para as métricas de tráfego de entrada e saída de rede por contêiner.

Tabela 13 – Sumário dos índices estatísticos para as métricas de tráfego de entrada e saída por contêiner.

Métrica	Contêiner	μ	μδ	σ	Π90	Π95	Π99
Tráfego de entrada (kB/s)	k6 @ CentOS	623,0	261,0	743,0	1915,7	2232,3	2560,0
	apigw @ Dojot	143,0	0,1	279,0	645,0	774,0	865,0
	node-red @ Node-RED	82,9	0,1	145,0	380,0	389,0	405,0
	mongo-db @ FIWARE	12,6	0,0	23,8	55,1	71,5	81,2
Tráfego de saída (kB/s)	k6 @ CentOS	385,0	0,0	730,0	1720,3	2240,6	2560,0
	apigw @ Dojot	158,0	0,2	305,0	715,0	858,0	959,0
	orion @ FIWARE	95,6	0,0	168,0	445,0	454,0	471,0
	node-red @ Node-RED	42,5	41,5	30,5	81,4	115,0	120,0

Fonte: Elaborado pelo autor (2025).

5.7. MÉTRICAS DO SMOKE TEST

O resultado do *smoke test* realizado com o k6, de forma simultânea para as 3 plataformas, demonstra um comportamento médio bastante estável e eficiente da aplicação sob uma carga de verificação (Tabela 14). A métrica de duração total das requisições HTTP (http_req_duration) apresentou média de 4 milissegundos (ms) e mediana de 3 ms, com valores no Π99 abaixo de 13 ms — o que indica boa responsividade mesmo em situações mais extremas. A maior parte desse tempo foi consumida em espera pela resposta do servidor (http_req_waiting), com valores muito

semelhantes à duração total, evidenciando que o servidor é o principal responsável pelo tempo de latência.

Tabela 14 – Sumário do *smoke test* das plataformas Dojot, FIWARE e Node-RED.

Tendências							
Métricas	**μ**	**max**	**μδ**	**min**	**Π90**	**Π95**	**Π99**
http_req_blocked	21μs	11ms	12μs	4μs	16μs	20μs	106μs
http_req_connecting	6μs	10ms	0ms	0ms	0ms	0ms	0ms
http_req_duration	4ms	104ms	3ms	927μs	6ms	8ms	13ms
http_req_receiving	205μs	13ms	182μs	42μs	307μs	359μs	529μs
http_req_sending	56μs	14ms	35μs	11μs	99μs	121μs	272μs
http_req_tls_handshaking	0ms	0ms	0ms	0ms	0ms	0ms	0ms
http_req_waiting	4ms	104ms	3ms	846μs	6ms	7ms	12ms
iteration_duration	4ms	104ms	4ms	1ms	7ms	8ms	13ms
Contadores							
Métricas	**Contagem**	**Taxa**					
data_received	108 MB	1.8 MB/s					
data_sent	2.84 MB	47.2 kB/s					
http_reqs	37.1k	617.61/s					
iterations	37.1k	617.61/s					
Taxas							
Métricas	**Taxa**						
checks	100.0%						
http_req_failed	0.0%						
Medidores							
Métricas	**Valor**						
vus	3						
vus_max	3						

Fonte: Elaborado pelo autor (2025).

As demais métricas relacionadas à comunicação se mantiveram em níveis bastante baixos. O tempo em que a requisição ficou bloqueada (http_req_blocked) teve média de 21 microssegundos (μs), e o tempo de conexão (http_req_connecting) foi praticamente nulo na maior parte das execuções. Já os tempos de envio (http_req_sending) e recebimento (http_req_receiving) de dados se mantiveram abaixo de 1 milissegundo em praticamente todos os percentis, chegando a no máximo 529 μs no Π99 de recebimento.

Do ponto de vista de confiabilidade, a taxa de falhas (http_req_failed) foi de 0%, e todas as verificações definidas (checks) foram concluídas com sucesso, com taxa de 100%. Durante o teste, foram executadas 37.100 iterações (iterations), com média

de 617 iterações por segundo, gerando um total de 108 MB de dados recebidos (data_received) e 2,84 MB de dados enviados (data_sent). A execução foi realizada por 3 usuários virtuais simultâneos (vus), mantendo esse valor constante durante toda a simulação.

Dessa forma, os resultados indicam que os *endpoints* das aplicações se comportam de maneira consistente, com baixa latência, sem falhas e com uso eficiente de rede, validando sua prontidão funcional para etapas posteriores de testes mais intensivos.

5.8. MÉTRICAS DO LOAD TEST

O *load test* aplicado à plataforma Dojot (Tabela 15) revelou desempenho estável sob carga moderada, com tempo médio de duração das requisições HTTP (http_req_duration) em torno de 3 milissegundos e mediana de 2 ms, o que reflete boa capacidade de resposta.

Tabela 15 – Sumário do *load test* da plataforma Dojot.

Tendências							
Métricas	**μ**	**max**	**μδ**	**min**	**Π90**	**Π95**	**Π99**
http_req_blocked	40μs	3ms	15μs	5μs	22μs	28μs	1ms
http_req_connecting	19μs	2ms	0ms	0ms	0ms	0ms	1ms
http_req_duration	3ms	30ms	2ms	928μs	5ms	6ms	10ms
http_req_receiving	218μs	1ms	201μs	49μs	318μs	367μs	540μs
http_req_sending	75μs	1ms	60μs	13μs	123μs	157μs	342μs
http_req_waiting	3ms	30ms	2ms	835μs	4ms	6ms	9ms
iteration_duration	1s	1s	1s	1s	1s	1s	1s
Métricas	**Contagem**	**Taxa**					
data_received	10.3 MB	42.8 kB/s					
data_sent	387 kB	1.61 kB/s					
http_reqs	5.4k	22.37/s					
iterations	5.4k	22.37/s					
Taxas							
Métricas	**Taxa**						
checks	100.0%						
http_req_failed	0.0%						
Medidores							
Métricas	**Valor**						
vus	1						
vus_max	50						

Fonte: Elaborado pelo autor (2025).

O tempo de espera pela resposta (http_req_waiting) acompanha esses valores, sugerindo que o principal fator de latência está no tempo de processamento da requisição no servidor, enquanto os tempos de envio (http_req_sending) e recebimento (http_req_receiving) permaneceram baixos, todos abaixo de 1 ms nos percentis mais elevados.

A quantidade de dados processados durante o teste foi de 10,3 MB recebidos e 387 kB enviados, com uma taxa de 22 requisições por segundo e 5.400 iterações no total. Destaca-se que todas as verificações definidas foram bem-sucedidas (checks = 100%) e não foram registradas falhas de requisição (http_req_failed = 0%), demonstrando confiabilidade.

Dessa forma, o teste evidencia um sistema capaz de manter baixa latência e alta confiabilidade sob cargas moderadas. Os tempos médios e percentis das requisições HTTP indicam respostas rápidas e consistentes, com o maior tempo sendo o de espera pela resposta do servidor, o que é esperado em arquiteturas distribuídas. A ausência de falhas e a taxa de sucesso total nas verificações confirmam a estabilidade funcional do sistema durante o teste. O volume de requisições e dados processados demonstra que a plataforma suporta um fluxo constante de interações sem degradação perceptível.

Esses resultados apontam para uma boa base para escalabilidade e sustentação de cargas maiores em testes futuros, indicando que a Dojot está preparada para operar eficientemente em ambientes IoT com demandas reais.

Quanto ao *load test* da plataforma FIWARE (Tabela 16), este indicou um desempenho consistente e responsivo sob carga leve a moderada. A duração média das requisições HTTP (http_req_duration) foi de 4 milissegundos, com a mediana igual, e percentis máximos (Π99) abaixo de 12 ms, demonstrando tempos de resposta rápidos e pouca variação. O tempo de espera pela resposta do servidor (http_req_waiting) seguiu esse padrão, indicando que o processamento no *backend* é eficiente e estável.

Os tempos de bloqueio (http_req_blocked) e conexão (http_req_connecting) foram mínimos, evidenciando que não houve contenção significativa de recursos ou atrasos na negociação da conexão. O envio (http_req_sending) e o recebimento

(http_req_receiving) de dados também se mantiveram em níveis baixos, com médias inferiores a 200 microssegundos, o que contribui para a fluidez do tráfego.

Tabela 16 – Sumário do *load test* da plataforma FIWARE.

Tendências							
Métricas	**μ**	**max**	**μδ**	**min**	**Π90**	**Π95**	**Π99**
http_req_blocked	29μs	10ms	11μs	4μs	20μs	24μs	139μs
http_req_connecting	13μs	10ms	0ms	0ms	0ms	0ms	0ms
http_req_duration	4ms	24ms	4ms	2ms	7ms	9ms	12ms
http_req_receiving	167μs	1ms	148μs	42μs	264μs	314μs	441μs
http_req_sending	57μs	3ms	51μs	12μs	97μs	115μs	193μs
http_req_waiting	4ms	23ms	3ms	2ms	7ms	9ms	12ms
iteration_duration	1s	1s	1s	1s	1s	1s	1s
Contadores							
Métricas	**Contagem**	**Taxa**					
data_received	890 kB	3.7 kB/s					
data_sent	445 kB	1.85 kB/s					
http_reqs	5.4k	22.29/s					
iterations	5.4k	22.29/s					
Taxas							
Métricas	**Taxa**						
checks	100.0%						
http_req_failed	0.0%						
Medidores							
Métricas	**Valor**						
vus	1						
vus_max	50						

Fonte: Elaborado pelo autor (2025).

Durante o teste, a plataforma processou 5.400 iterações, com uma taxa constante de aproximadamente 22 requisições por segundo. Os dados transferidos foram modestos, com 890 kB recebidos e 445 kB enviados, compatíveis com um cenário típico de comunicação de sensores ou mensagens IoT. A ausência total de falhas (http_req_failed = 0%) e a taxa perfeita de verificações (checks = 100%) reforçam a confiabilidade do sistema.

De forma similar à plataforma Dojot, os resultados obtidos confirmam que a plataforma FIWARE oferece respostas rápidas, estáveis e confiáveis, com baixo volume de tráfego de dados. Essa combinação a torna especialmente adequada para aplicações IoT que exigem comunicação contínua, eficiente e com baixa latência,

assegurando um desempenho consistente mesmo em ambientes com múltiplos dispositivos conectados.

Por último, O *load test* da plataforma Node-RED (Tabela 17) demonstrou desempenho estável e responsivo sob carga leve, com duração média das requisições HTTP (http_req_duration) em torno de 4 milissegundos e mediana de 3 ms. Os percentis superiores indicam que 99% das requisições foram concluídas em até 10 ms, o que ressalta boa capacidade de resposta mesmo sob picos de carga.

Tabela 17 – Sumário do *load test* da plataforma Node-RED.

Tendências							
Métricas	**μ**	**max**	**μδ**	**min**	**Π90**	**Π95**	**Π99**
http_req_blocked	34μs	5ms	15μs	5μs	22μs	26μs	203μs
http_req_connecting	14μs	5ms	0ms	0ms	0ms	0ms	0ms
http_req_duration	4ms	28ms	3ms	1ms	5ms	6ms	10ms
http_req_receiving	258μs	3ms	243μs	67μs	365μs	405μs	527μs
http_req_sending	73μs	2ms	67μs	14μs	118μs	140μs	221μs
http_req_waiting	3ms	27ms	3ms	1ms	5ms	6ms	10ms
iteration_duration	1s	1s	1s	1s	1s	1s	1s
Contadores							
Métricas	**Contagem**	**Taxa**					
data_received	34 MB	141 kB/s					
data_sent	413 kB	1.72 kB/s					
http_reqs	5.4k	22.31/s					
iterations	5.4k	22.31/s					
Taxas							
Métricas	**Taxa**						
checks	100.0%						
http_req_failed	0.0%						
Medidores							
Métricas	**Valor**						
vus	1						
vus_max	50						

Fonte: Elaborado pelo autor (2025).

Os tempos de bloqueio (http_req_blocked) e conexão (http_req_connecting) foram baixos, mostrando que não houve contenção ou atrasos significativos no estabelecimento da conexão. O envio (http_req_sending) e recebimento (http_req_receiving) de dados apresentaram valores baixos, com médias inferiores a 300 microssegundos, contribuindo para a agilidade no processamento das requisições.

Durante o teste, a plataforma processou o maior volume de dados recebidos, 34 MB, com taxa de 141 kB/s, indicando uma comunicação mais intensa em relação às outras plataformas testadas. A taxa de requisições foi de aproximadamente 22 por segundo, com 5.400 iterações totais, todas sem falhas e com 100% de sucesso nas verificações, reforçando a confiabilidade da aplicação.

Assim, o teste validou a capacidade da Node-RED de responder com rapidez e manter a estabilidade em um cenário típico de operação, demonstrando estar preparada para suportar cargas IoT com alta demanda de dados, de forma similar às plataformas Dojot e FIWARE.

Consequentemente, os resultados dos testes de carga realizados nas plataformas Dojot, FIWARE e Node-RED demonstraram que todas apresentam desempenho consistente, estável e adequado para aplicações IoT com requisitos de comunicação contínua e simultaneidade. As três plataformas mantiveram baixas latências nas requisições HTTP, com tempos de resposta médios entre com $\Pi 95$ que variou de 6 a 9 ms, satisfazendo o limiar de 500 ms configurado, ausência de falhas nas requisições e estabilidade no comportamento dos usuários virtuais. Além disso, o volume de dados trafegados permaneceu compatível com o perfil leve de aplicações IoT, evidenciando que os *middlewares* são capazes de lidar eficientemente com múltiplas conexões e alto número de iterações. Esses resultados confirmam a prontidão das plataformas para integrações robustas em ambientes distribuídos com múltiplos dispositivos inteligentes.

5.9. MÉTRICAS DO STRESS TEST

O *stress test* aplicado à plataforma Dojot () evidenciou sua robustez mesmo diante de uma carga substancialmente elevada. Com uma média de 227 requisições por segundo e um total de 286.500 iterações, a plataforma manteve um desempenho consistente. A duração média das requisições HTTP (http_req_duration) foi de apenas 4 milissegundos, ainda que tenham ocorrido picos de até 348 milissegundos, indicando variações pontuais em cenários de maior concorrência.

Os tempos de bloqueio (http_req_blocked) e conexão (http_req_connecting), apesar de médios na faixa dos microssegundos, apresentaram máximos de 124 milissegundos, provavelmente reflexo de contenção de recursos em momentos de

saturação. Já o tempo de espera pela resposta (http_req_waiting) seguiu alinhado com a duração total da requisição, o que aponta que a maior parte da latência está relacionada ao processamento do lado servidor.

Tabela 18 – Sumário do *stress test* da plataforma Dojot.

Tendências							
Métricas	**μ**	**max**	**μδ**	**min**	**Π90**	**Π95**	**Π99**
http_req_blocked	26μs	124ms	9μs	3μs	16μs	20μs	659μs
http_req_connecting	12μs	124ms	0ms	0ms	0ms	0ms	542μs
http_req_duration	4ms	348ms	2ms	809μs	8ms	13ms	42ms
http_req_receiving	163μs	87ms	117μs	41μs	263μs	357μs	738μs
http_req_sending	74μs	196ms	27μs	12μs	91μs	208μs	722μs
http_req_waiting	4ms	348ms	2ms	715μs	8ms	13ms	41ms
iteration_duration	1s	1s	1s	1s	1s	1s	1s
Contadores							
Métricas	**Contagem**	**Taxa**					
data_received	549 MB	435 kB/s					
data_sent	20.6 MB	16.4 kB/s					
http_reqs	286.5k	227.23/s					
iterations	286.5k	227.23/s					
Taxas							
Métricas	**Taxa**						
checks	100.0%						
http_req_failed	0.0%						
Medidores							
Métricas	**Valor**						
vus	1						
vus_max	400						

Fonte: Elaborado pelo autor (2025).

O tráfego também foi expressivo: 549 MB recebidos e 20,6 MB enviados, representando uma carga realista para aplicações IoT em escala. Apesar desse volume, o teste registrou 0% de falhas (http_req_failed), com 100% de sucesso nas verificações (checks), reforçando a confiabilidade da plataforma sob pressão.

A Tabela 18 apresenta um sumário das categorias e métricas resultantes do *stress test d*a plataforma Dojot.

O teste, que iniciou com 1 usuário virtual e escalou até 400 usuários simultâneos (vus_max), confirma, junto ao *load test*, que a Dojot está apta a lidar com altos níveis de concorrência sem comprometer a disponibilidade ou a responsividade.

Esses resultados atestam a capacidade da Dojot para operar de forma estável e eficiente em cenários críticos e de alta demanda, como exigido em ambientes IoT robustos.

O *stress test* da plataforma FIWARE (Tabela 19) apresentou desempenho funcional sob alta carga, com 220 mil requisições executadas a uma taxa constante de aproximadamente 175 requisições por segundo. No entanto, ao contrário dos demais testes realizados, este foi o único cenário em que os tempos de resposta ultrapassaram o limiar considerado ideal de 500 ms no percentil 95 ($\Pi 95$) para a métrica de duração da requisição HTTP (http_req_duration), alcançando 879 ms no P95 e até 3 segundos no pico máximo.

Tabela 19 – Sumário do *stress test* para a plataforma FIWARE.

Tendências							
Métricas	**μ**	**max**	**$\mu\delta$**	**min**	**$\Pi 90$**	**$\Pi 95$**	**$\Pi 99$**
http_req_blocked	23µs	175ms	15µs	3µs	22µs	27µs	96µs
http_req_connecting	3µs	24ms	0ms	0ms	0ms	0ms	0ms
http_req_duration	304ms	3s	244ms	2ms	800ms	879ms	1s
http_req_receiving	382µs	379ms	205µs	43µs	381µs	649µs	1ms
http_req_sending	120µs	404ms	45µs	13µs	112µs	140µs	456µs
http_req_waiting	304ms	3s	243ms	2ms	799ms	879ms	1s
iteration_duration	1s	4s	1s	1s	1s	1s	2s
Contadores							
Métricas		**Contagem**		**Taxa**			
data_received		36.6 MB		29 kB/s			
data_sent		18.3 MB		14.5 kB/s			
http_reqs		220.6k		174.9/s			
iterations		220.6k		174.9/s			
Taxas							
Métricas		**Taxa**					
checks		100.0%					
http_req_failed		0.0%					
Medidores							
Métricas		**Valor**					
vus		1					
vus_max		400					

Fonte: Elaborado pelo autor (2025).

Esse comportamento sugere que, embora a FIWARE mantenha sua estabilidade e não registre falhas nas requisições, a plataforma demonstrou limitação

de desempenho sob carga extrema, principalmente no tempo de resposta do servidor. Os tempos médios de espera (http_req_waiting) e duração da requisição (http_req_duration) giraram em torno de 304 ms, o que, apesar de ainda estar dentro de um intervalo aceitável para muitos sistemas IoT, pode se tornar um gargalo em aplicações sensíveis à latência.

Outros aspectos, como baixo tempo de bloqueio e conexão, e os tempos de envio e recebimento em microssegundos, apontam que a sobrecarga está mais relacionada ao processamento interno das requisições do que à infraestrutura de rede. Apesar do aumento do tempo de resposta, a estabilidade da plataforma foi mantida com 100% de sucesso nas verificações, evidenciando confiabilidade operacional mesmo sob estresse. Esse resultado é um indicativo valioso para arquiteturas IoT críticas, ressaltando a necessidade de dimensionamento adequado da FIWARE em cenários de alta concorrência, ou a adoção de estratégias de escalabilidade para mitigar impactos de latência sob picos de carga.

Por fim, o *stress test* realizado na plataforma Node-RED (Tabela 20) demonstrou excelente desempenho sob alta carga, com mais de 283 mil requisições processadas, a uma média de 225 requisições por segundo, sem registrar qualquer falha ou interrupção. A métrica de duração das requisições HTTP (http_req_duration) manteve-se significativamente baixa, com valor médio de apenas 16 milissegundos e pico máximo de 291 milissegundos, mantendo o percentil 95 (Π95) dentro de 68 milissegundos.

Além disso, os tempos de bloqueio e conexão foram praticamente desprezíveis, e os tempos de envio e recebimento de dados também permaneceram baixos, indicando ótima eficiência na comunicação entre cliente e servidor. O tempo de espera pelas respostas (http_req_waiting) foi praticamente idêntico à duração total da requisição, sinalizando que a latência observada está majoritariamente associada ao tempo de resposta do servidor, e não a atrasos na rede.

Um destaque importante foi o grande volume de dados trafegado, com 1,79 GB recebidos, superando as demais plataformas testadas. Isso sugere que o Node-RED lida bem com cargas de dados intensas, o que é altamente relevante em cenários de IoT com alto volume de eventos e mensagens. Essa capacidade de recepção reforça

sua adequação como ferramenta de orquestração e integração de fluxos complexos, característica essencial em ambientes heterogêneos e distribuídos.

Tabela 20 – Sumário do stress test da plataforma Node-RED.

Tendências							
Métricas	**μ**	**max**	**μδ**	**min**	**Π90**	**Π95**	**Π99**
http_req_blocked	17μs	10ms	12μs	4μs	18μs	22μs	80μs
http_req_connecting	1μs	9ms	0ms	0ms	0ms	0ms	0ms
http_req_duration	16ms	291ms	5ms	1ms	44ms	68ms	124ms
http_req_receiving	263μs	41ms	222μs	55μs	375μs	492μs	817μs
http_req_sending	61μs	24ms	36μs	13μs	101μs	123μs	390μs
http_req_waiting	16ms	290ms	5ms	1ms	43ms	68ms	124ms
iteration_duration	1s	1s	1s	1s	1s	1s	1s
Contadores							
Métricas	**Contagem**	**Taxa**					
data_received	1.79 GB	1.42 MB/s					
data_sent	21.8 MB	17.3 kB/s					
http_reqs	283.3k	224.59/s					
iterations	283.3k	224.59/s					
Taxas							
Métricas	**Taxa**						
checks	100.0%						
http_req_failed	0.0%						
Medidores							
Métricas	**Valor**						
vus	1						
vus_max	400						

Fonte: Elaborado pelo autor (2025).

Além disso, a estabilidade observada nos testes — com 0% de falhas nas requisições HTTP e 100% de *checks* bem-sucedidos — confirma sua confiabilidade mesmo sob condições extremas. O comportamento consistente no envio e recebimento de dados, aliado à manutenção do desempenho durante picos de tráfego, indica que o Node-RED está apto a operar como um componente robusto dentro de arquiteturas IoT críticas, especialmente na interligação entre sensores, atuadores e serviços de processamento.

No geral, os resultados indicam que a plataforma Node-RED está bem-preparada para operar em ambientes com alta simultaneidade e grande tráfego de dados, apresentando baixa latência, alta estabilidade e eficiência na comunicação, características que a tornam altamente indicada para aplicações IoT em larga escala.

Finalmente, os *stress tests* realizados nas plataformas Dojot, FIWARE e Node-RED confirmaram que todas são capazes de lidar com cargas elevadas, mantendo desempenho estável e confiável. A Dojot demonstrou robustez com uma média de 227 requisições por segundo e baixas latências, mesmo em cenários de tráfego intenso. A Node-RED obteve o maior volume de dados processados, com 1.79 GB recebidos, mantendo latências extremamente baixas — média de apenas 16 ms — e sem ultrapassar os limiares de desempenho. A FIWARE, embora também tenha sustentado alta taxa de requisições com estabilidade, foi a única a ultrapassar o limite de 500 ms no percentil 95 de duração das requisições, atingindo até 879 ms, o que sugere variações mais acentuadas em condições extremas. Ainda assim, nenhuma das plataformas apresentou falhas ou perda de disponibilidade, validando sua adequação para aplicações IoT exigentes e com alta simultaneidade.

5.10. CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou uma análise abrangente do comportamento das máquinas virtuais e dos contêineres que compõem a arquitetura de *middlewares* IoT estudada, por meio de experimentos projetados para avaliar consumo de recursos, desempenho em condições reais de operação e limites das plataformas Dojot, FIWARE e Node-RED. Os resultados revelaram aspectos operacionais e de performance fundamentais para compreender a escalabilidade e a robustez do ambiente virtualizado.

Na primeira etapa, foram analisados os consumos de CPU e memória RAM, tanto no nível das máquinas virtuais quanto dos contêineres, permitindo identificar o perfil de alocação de recursos de cada *middleware*. Em seguida, avaliaram-se as atividades de leitura e escrita em disco, essenciais para identificar possíveis gargalos relacionados à persistência de dados. O tráfego de rede também foi investigado, evidenciando o impacto das comunicações internas entre serviços em diferentes cenários de operação.

Na segunda etapa, três tipos de testes — *smoke*, *load* e *stress* — foram aplicados para simular níveis crescentes de carga. O *smoke test* verificou a estabilidade básica dos serviços após a inicialização; o *load test* avaliou o comportamento sob carga moderada e contínua, confirmando o funcionamento adequado das plataformas; e o *stress test* expôs os limites operacionais dos sistemas,

revelando que Dojot e Node-RED mantiveram latências abaixo dos limites críticos, enquanto a FIWARE demonstrou maior sensibilidade sob carga extrema.

Com base nesses resultados, verificou-se que cada plataforma apresenta diferentes potenciais de aplicação. A Dojot mostrou-se mais adequada para cenários de médio porte, por ser uma solução pronta e por apresentar saturação de recursos em cargas mais elevadas. A FIWARE destacou-se como a plataforma com maior potencial de escalabilidade, recomendada para aplicações de pequeno, médio e grande porte devido à flexibilidade de customização e ao uso mais eficiente de recursos computacionais, mesmo apresentando maior sensibilidade quando exposta a cargas extremas. O Node-RED, por sua vez, demonstrou melhor desempenho em aplicações de pequeno porte, dado seu modelo baseado em um serviço central e dependente de nós adicionais para funcionalidades avançadas.

Embora o sistema de observabilidade implementado tenha se mostrado eficaz na coleta e visualização das métricas, melhorias adicionais podem ser obtidas com a incorporação de ferramentas complementares do ecossistema Grafana, como Loki para centralização de *logs* e Tempo para *tracing* distribuído, ampliando a capacidade de correlação entre eventos e fortalecendo a confiabilidade do ambiente IoT.

Em síntese, os experimentos realizados demonstram que a arquitetura proposta é funcional, escalável e resiliente, apresentando potencial para suportar aplicações IoT críticas. Os achados fornecem diretrizes relevantes para aprimoramentos futuros, tanto na infraestrutura de monitoramento quanto na gestão integrada do sistema.

6. CONCLUSÃO

Neste capítulo final, apresentam-se as considerações conclusivas, revisitando desde a contextualização da pesquisa e das tecnologias empregadas até a análise dos resultados alcançados por meio da metodologia adotada. Por fim, são discutidas as perspectivas para trabalhos futuros e as produções acadêmicas derivadas do escopo desta dissertação.

6.1. CONSIDERAÇÕES FINAIS

Diante de todo o conteúdo apresentado ao longo deste trabalho, conclui-se que a dissertação alcançou plenamente seu objetivo principal: a implantação de uma arquitetura computacional virtualizada e distribuída, fundamentada em *middlewares* para IoT, concebida para atuar como componente central na centralização de dados e informações do projeto EMOB-AMAZON.

Com base nas análises bibliométricas e na revisão de literatura, observa-se que as plataformas Dojot, FIWARE e Node-RED se configuram como soluções complementares dentro do ecossistema IoT, cada uma atendendo a demandas específicas. A Dojot destaca-se pela robustez e escalabilidade voltadas a aplicações críticas; o FIWARE consolida-se como referência em interoperabilidade entre sistemas heterogêneos; e o Node-RED se sobressai como ferramenta acessível e ágil para prototipagem e automação. Juntas, essas plataformas compõem um tripé estratégico que abrange desde a padronização global até soluções locais e customizadas, viabilizando a transformação digital em múltiplos contextos.

No escopo do *testbed* desenvolvido, foi possível estruturar, validar e analisar uma arquitetura IoT completa, composta por múltiplas camadas — física, *middleware*, processamento e *software* — integradas de forma coesa para suportar aplicações em sistemas inteligentes. A definição cuidadosa da infraestrutura envolveu tanto a configuração detalhada dos *middlewares* IoT quanto a seleção criteriosa de ferramentas de virtualização, conteinerização, monitoramento e testes de desempenho, garantindo a consistência técnica e a interoperabilidade dos componentes.

Nesse contexto, foram utilizadas tecnologias como o Proxmox VE, para virtualização e gerenciamento de VMs; o Docker e o Docker Compose, para implantação modular dos serviços; e o InfluxDB, Prometheus, cAdvisor e Grafana, para coleta, armazenamento e visualização de métricas de desempenho. Complementarmente, o Grafana k6 foi empregado para a realização de testes de carga, permitindo a avaliação da robustez e estabilidade da arquitetura em diferentes cenários operacionais.

Os experimentos conduzidos revelaram achados significativos. No consumo de recursos, observou-se que a FIWARE apresentou melhor eficiência e distribuição de carga entre contêineres, reforçando seu maior potencial de escalabilidade. A Dojot, embora funcional e estável, demonstrou tendência à saturação sob cargas crescentes, situando-se como alternativa ideal para aplicações de médio porte. O Node-RED exibiu desempenho consistente em cenários leves, mas suas limitações arquiteturais o direcionam preferencialmente a aplicações de pequeno porte. Nos testes de desempenho, verificou-se que todas as plataformas atenderam adequadamente ao *smoke* e ao *load test*, mantendo estabilidade e latências aceitáveis; já no *stress test*, Dojot e Node-RED resistiram até o limite crítico estabelecido, enquanto a FIWARE apresentou maior sensibilidade, com aumento expressivo na latência sob carga extrema. Esses achados destacam como características arquiteturais distintas influenciam diretamente a robustez e a resiliência de cada plataforma.

A integração dessas ferramentas foi fundamental para a construção de um ambiente realista, observável e replicável. O sistema de monitoramento implementado forneceu suporte analítico para a análise contínua do desempenho computacional, enquanto os testes de carga asseguraram a conformidade da solução com os requisitos de desempenho e confiabilidade. Dessa forma, a infraestrutura tecnológica adotada não apenas viabilizou a experimentação controlada de arquiteturas IoT, mas também ofereceu uma base sólida para futuras expansões e estudos em contextos complexos e exigentes.

A escolha dessas tecnologias de código aberto, cujos repositórios oficiais e licenças foram devidamente documentados, garantiu não apenas a viabilidade técnica e econômica do projeto, mas também sua extensibilidade e transparência. Essa base tecnológica sólida foi essencial para assegurar a escalabilidade, observabilidade e

confiabilidade da arquitetura IoT desenvolvida, além de oferecer um ponto de partida estruturado para futuras pesquisas e evoluções da plataforma.

Como resultado, os experimentos realizados comprovaram que a arquitetura proposta é funcional, escalável e resiliente, evidenciando seu potencial para sustentar aplicações IoT críticas. Além disso, os resultados forneceram diretrizes importantes para aprimoramentos futuros, tanto na infraestrutura de monitoramento quanto na gestão integrada do sistema.

Portanto, conclui-se que a arquitetura proposta nesta dissertação contribui de forma estratégica para o escopo do projeto EMOB-AMAZON, promovendo avanços tecnológicos, acadêmicos e sociais. Por meio do monitoramento contínuo da robustez operacional da solução, a iniciativa assegura um atendimento sustentável e eficaz às lacunas que o projeto se propõe a preencher, fortalecendo sua relevância nos contextos em que está inserido.

6.2. TRABALHOS FUTUROS

Como direcionamento para pesquisas futuras, recomenda-se uma abordagem abrangente para o desenvolvimento do sistema, contemplando três eixos principais: escalabilidade, testes de robustez e análise de componentes. Primeiramente, sugere-se um planejamento integrado de escalabilidade, combinando a vertical (por meio de aprimoramento de *hardware*) com a horizontal (através de distribuição eficiente de carga). Esta estratégia permitirá à infraestrutura adaptar-se progressivamente a demandas crescentes, garantindo tanto a expansão individual dos nós quanto o balanceamento otimizado entre múltiplos dispositivos.

Complementarmente, torna-se essencial a realização de testes adicionais, incluindo *soak*, *spike* e *breakpoint* para avaliação do sistema em condições operacionais críticas. Estes testes possibilitarão a identificação de vulnerabilidades sob cargas prolongadas, picos abruptos e situações extremas, fornecendo subsídios para otimizações de desempenho e aumento da robustez sistêmica. Paralelamente, propõe-se uma análise qualitativa criteriosa dos *middlewares* adotados, considerando cinco dimensões fundamentais: formalidade semântica, usabilidade, suporte a tipagem e metadados, eficiência computacional e capacidade de escalamento. Tal

avaliação multidimensional permitirá a seleção mais adequada de tecnologias para diversos cenários IoT.

Como terceiro pilar de desenvolvimento, recomenda-se aprimoramentos específicos em cada camada tecnológica. Na camada física da arquitetura do projeto, destacam-se: implantação estratégica de dispositivos, aquisição otimizada de dados sensoriais, garantia de estabilidade nas conexões de rede e execução de testes de desempenho exaustivos. Na camada de processamento, a implementação de algoritmos avançados para previsão de geração fotovoltaica e manutenção preditiva trará ganhos significativos em eficiência energética e redução de custos operacionais. Finalmente, no âmbito do *software*, propõe-se o desenvolvimento de soluções integradas para controle supervisório, aquisição contínua de dados e monitoramento em tempo real de parâmetros críticos.

Estas propostas, quando implementadas de forma coordenada, proporcionarão significativos avanços na eficiência, confiabilidade e adaptabilidade do sistema como um todo, preparando-o adequadamente para os desafios operacionais futuros e diversas condições de implantação.

6.3. PRODUÇÕES

Com base no desenvolvimento apresentado ao longo deste trabalho, a arquitetura IoT proposta passou por um processo contínuo e iterativo de concepção, implantação, monitoramento e validação, culminando em um *testbed* robusto, escalável e apto a sustentar aplicações voltadas a sistemas inteligentes. Essa evolução foi impulsionada pelo aprimoramento progressivo das camadas física, de *middleware* e de observabilidade, bem como pelas análises quantitativa e qualitativa realizadas sobre os *middlewares* utilizados. Como resultado, foram elaborados e submetidos trabalhos científicos que abordam diferentes aspectos da arquitetura proposta, com o propósito de divulgar as metodologias aplicadas, os resultados alcançados e as contribuições técnicas geradas no âmbito desta pesquisa.

A seguir, são apresentados, em ordem cronológica, os cabeçalhos das produções científicas associadas a esta pesquisa, contendo seus respectivos títulos, autores e eventos de publicação. Cada produção é acompanhada de seu resumo,

com o intuito de contextualizar brevemente os objetivos, metodologias e contribuições de cada trabalho.

Produção 1: *Middleware*s IoT para aplicações de Digital Twins de Sistemas Fotovoltaicos

I Congreso Internacional de Transición Enegética y Medio Ambiente (2023)

MIDDLEWARES IOT PARA APLICAÇÕES DE DIGITAL TWINS DE SISTEMAS FOTOVOLTAICOS
IOT MIDDLEWARES FOR DIGITAL TWINS APPLICATIONS OF PHOTOVOLTAIC SYSTEMS

João Luiz Pontes de Araújo[1]; Elen Priscila de Souza Lobato[1]; Damasio Alves de Lima Júnior[1]; Lucas Henrique Brito Santos[1]; Wellington da Silva Fonseca[1]

Resumo:

Este trabalho tem como objetivo apresentar a integração de diferentes tecnologias da Indústria 4.0 presentes em uma arquitetura computacional criada para aplicações de Digital Twins de Sistemas Solares Fotovoltaicos. Para isso, foi criado um ambiente de desenvolvimento, com três (3) Máquinas Virtuais (VM, sigla em inglês), de sistema operacional Linux Ubuntu 20.04, através do hipervisor Proxmox VE 7.3-3. Onde, cada máquina contém uma imagem Docker com um *middleware* IoT. Os *middlewares* IoT escolhidos para este trabalho foram: Dojot, FIWARE e Node-RED. Em cada *middleware* foram criados modelos e dispositivos virtuais que recebem os dados de um Sistema Solar Fotovoltaico físico, obtidos através de um sistema embarcado (baseado no microcontrolador ESP-32). Os dados aferidos são utilizados como variáveis de entrada de uma simulação que utiliza o Método de Elementos Finitos (MEF) e os resultados desta simulação, somando aos dados aferidos, são utilizados no Digital Twin de Sistemas Solares Fotovoltaicos. Dessa forma, através do uso e integração das tecnologias supracitadas, com o desenvolvimento do presente trabalho, foi possível demonstrar a viabilidade da construção da arquitetura proposta. Além disso, constatou-se que a técnica de virtualização baseada em contêineres simplifica e reduz custos computacionais, em relação a implementação da camada de gestão de dados de uma aplicação. Concluindo-se, assim, que a utilização de tecnologias da Indústria 4.0, como IoT, virtualização, simulação, Digital Twin, entre outras, que, quando bem integradas, podem contribuir no desenvolvimento e

implementação de arquiteturas voltadas para aplicações de Energias Renováveis, como a proposta neste trabalho.

Produção 2: Digital Twin de Plantas Fotovoltaicas para monitoramento e análise térmica

Congresso Nacional de Engenharia Mecânica (2023)

CONEM 2024 - 1213
DIGITAL TWIN DE PLANTAS FOTOVOLTAICAS PARA ANÁLISE E MONITORAMENTO TÉRMICO

João Luiz Pontes de Araújo, joao.pontes.araujo@itec.ufpa.br[1]
Damasio Alves de Lima Júnior, damasio.junior@itec.ufpa.br[1]
Lucas Henrique Brito Santos, lucas.brito.santos@itec.ufpa.br[1]
Paulo Antônio Jordão Couto, paulo.couto@itec.ufpa.br[1]
Elen Priscila de Souza Lobato, elen.lobato@itec.ufpa.br[1]
Wellington da Silva Fonseca, fonseca@ufpa.br[1]

[1]Universidade Federal do Pará (UFPA) - Centro de Excelência em Eficiência Energética da Amazônia - CEAMAZON

Resumo:

O presente trabalho tem como objetivo apresentar a camada de tecnologias relacionadas a dados de um Digital Twin de uma planta fotovoltaica. Para isso, realizou-se o monitoramento térmico, de uma planta fotovoltaica, situada nas instalações do Centro de Excelência em Eficiência Energética da Amazônia (CEAMAZON), entidade da Universidade Federal do Pará (UFPA). A arquitetura proposta consiste em um sistema embarcado, baseado no microcontrolador ESP-32, o *middleware* Dojot e simulações computacionais, com o *software* Elmer FEM, utilizando o Método de Elementos Finitos (MEF). O ambiente de desenvolvimento criado, para a realização do trabalho, foi hospedado em um servidor do CEAMAZON. Para a persistência e visualização dos dados, foram enviados à plataforma *middleware* Dojot. A arquitetura criada demonstra uma complexidade reduzida no acesso à informação gerada, ocasionado pela implementação dos *middlewares* como uma subcamada responsável por centralizar e ocultar as dificuldades inerentes das tecnologias utilizadas. Sublinhando, assim, uma alternativa plausível à camada de dados do Digital Twin. Destaca-se que este trabalho foi desenvolvido dentro do escopo do projeto nomeado “Desenvolvimento Tecnológico e Extensão Inovadora de Materiais Avançados em Energia e Mobilidade aplicados ao Contexto Amazônico” (EMOB-AMAZON), financiado pela Financiadora de Estudos e Projetos (FINEP) e implantado na Universidade Federal do Pará (UFPA).

Produção 3: Observabilidade em Ambientes Virtualizados para Sistemas de Energias Renováveis Baseados em *Middleware*

16th INDUSCON (2025)

Observabilidade em Ambientes Virtualizados para Sistemas de Energias Renováveis Baseados em Middleware

João Luiz Pontes de Araújo
Centro de Excelência em Eficiência Energética
Instituto de Tecnologia, Universidade Federal do Pará
Belém, Brasil
joao.pontes.araujo@itec.ufpa.br

Elen Priscila de Souza Lobato
Centro de Excelência em Eficiência Energética
Instituto de Tecnologia, Universidade Federal do Pará
Belém, Brasil
elen.lobato@itec.ufpa.br

Wellington da Silva Fonseca
Centro de Excelência em Eficiência Energética
Instituto de Tecnologia, Universidade Federal do Pará
Belém, Brasil
fonseca@ufpa.br

Resumo:

Este artigo apresenta a implantação e a análise de desempenho de um sistema de energia renovável baseado em *middleware* em um ambiente virtual no campus da Universidade Federal do Pará. Sua análise de desempenho envolveu a operação ociosa de três máquinas virtuais (VMs) executando os *middlewares* de IoT Dojot, FIWARE e Node-RED, com um sistema de observabilidade sendo implantado em uma quarta VM integrando Proxmox VE, InfluxDB e Grafana. As principais métricas incluem uso de CPU, I/O wait, Loadavg, uso de RAM, L/E de disco e E/S de tráfego de rede. Os resultados mostram um sistema bem dimensionado e variações de desempenho estáveis para cada VM analisada e um sistema descomplicado e eficiente para monitorar os recursos do servidor por meio da integração do Proxmox VE, InfluxDB e Grafana. Trabalhos futuros se concentrarão em expandir a análise de recursos para um estado de produção, com cenários simulados, bem como o sistema de observabilidade para coletar e analisar logs e traces, dando suporte à implantação de alertas e gatilhos baseados no monitoramento implantado.

Produção 4: Arquitetura de Observabilidade para Contêineres Docker de *Middlewares* IoT em Ambientes Virtualizados

16th INDUSCON (2025)

Arquitetura de Observabilidade para Contêineres Docker de Middlewares IoT em Ambientes Virtualizados

João Luiz Pontes de Araújo
Universidade Federal do Pará
Centro de Excelência em Eficiência Energética da Amazônia
Belém, Brasil
joao.pontes.araujo@itec.ufpa.br

Elen Priscila de Souza Lobato
Universidade Federal do Pará
Centro de Excelência em Eficiência Energética da Amazônia
Belém, Brasil
elen.lobato@ufpa.br

Aleqssandro Alexandre de Oliveira Farias
Universidade Federal do Pará
Centro de Excelência em Eficiência Energética da Amazônia
Belém, Brasil
aleqssandro.farias@ufpa.br

Wellington da Silva Fonseca
Universidade Federal do Pará
Centro de Excelência em Eficiência Energética da Amazônia
Belém, Brasil
fonseca@ufpa.br

Resumo:

Este artigo apresenta a implantação e observabilidade de arquiteturas IoT baseada em *middlewares* conteinerizados, com base nas plataformas Dojot, FIWARE e Node-RED, implantadas em contêineres Docker executados em máquinas virtuais (VM) gerenciadas pelo hipervisor Proxmox VE, compondo uma infraestrutura flexível para criação e gerenciamento de dispositivos IoT. Para assegurar a visibilidade operacional, uma *stack* baseada em Grafana, Prometheus e cAdvisor coleta e monitora métricas de desempenho das VMs, incluindo uso de CPU, memória e outros recursos essenciais. A proposta demonstra como a combinação de *middlewares* conteinerizados e monitoramento avançado pode facilitar a implantação, o gerenciamento de recursos e o planejamento da escalabilidade em sistemas baseados ou integrados a tecnologias IoT.

7. DECLARAÇÃO DE USO DE INTELIGÊNCIA ARTIFICIAL

Durante o desenvolvimento deste trabalho, foi utilizada a Inteligência Artificial (IA) para auxiliar em algumas etapas da pesquisa e elaboração, conforme segue:

1. Ferramentas de IA utilizadas: ChatGPT, DeepSeek, SciSpace.

2. Objetivo do uso: A IA foi utilizada para geração de resumos de textos, revisão gramatical, apoio na redação de trechos técnicos e auxílio na busca de literatura indexada relacionada a tópicos específicos da pesquisa.

3. Processo de integração da IA: As ferramentas foram utilizadas de forma complementar ao processo de escrita e análise, oferecendo sugestões de revisão linguística, apoio na organização de ideias e identificação de referências relevantes. Todo o conteúdo sugerido pela IA foi revisado criticamente pelo autor antes de sua incorporação ao trabalho.

A autoria do conteúdo original e as análises realizadas neste trabalho são de responsabilidade exclusiva do autor. A utilização da IA não substitui a análise crítica, interpretação ou conclusões da pesquisa. A IA foi empregada como ferramenta auxiliar para facilitar algumas etapas do processo de escrita e análise.

Declaro, portanto, que o uso da IA foi realizado dentro das normas éticas e acadêmicas, com o intuito de melhorar a qualidade e precisão do trabalho, sem comprometer a integridade e a originalidade do conteúdo.

gov.br
Documento assinado digitalmente
JOAO LUIZ PONTES DE ARAUJO
Data: 02/12/2025 11:29:06-0300
Verifique em https://validar.iti.gov.br

_____

Assinatura

REFERÊNCIAS

ABDALLAH, Salam; KHALIL, Ashraf. Using a hybrid methodology for literature review: a case study in depression research. Information Discovery and Delivery, v. 52, n. 3, p. 305–323, 2 jul. 2024.

ADHIPTA, Dani; SELO; WIDYAWAN. Docker Swarm Orchestration for High Performance Computing Cluster as Container. *In*: IEEE, 11 dez. 2023.

ADKINS, H. *et al.,* Building Secure and Reliable Systems: Best Practices for Designing, Implementing, and Maintaining Systems. *[S.l.]*: O'Reilly Media, 2020.

ALAM, Marzia; GUL, Mehreen Saleem; MUNEER, Tariq. Performance analysis and comparison between bifacial and monofacial solar photovoltaic at various ground albedo conditions. Renewable Energy Focus, v. 44, p. 295–316, mar. 2023.

ANDREI, Gavrilov *et al.,* Industrial Messaging *Middleware*: Standards and Performance Evaluation. *In*: IEEE, 7 out. 2020.

ANGIOI, Manuela; HILLER, Claire E. Systematic Literature Reviews. *In*: Research Methods in the Dance Sciences. *[S.l.]*: University Press of Florida, 2023. p. 265–280.

ARAÚJO, Felipe *et al.,* Plataforma para Coleta de Dados de Eletropostos para Veículos Elétricos: Prototipação em Bancada de Testes Real. *In*: Sociedade Brasileira de Computação - SBC, 31 jul. 2022.

ARCHANA; SHANKAR, Ravi; SINGH, Shveta. Development of smart grid for the power sector in India. Cleaner Energy Systems, v. 2, p. 100011, jul. 2022.

ARMANDO, Ngombo; SÁ SILVA, Jorge; BOAVIDA, Fernando. An Approach to the Unified Management of Heterogeneous IoT Environments. IEEE Internet of Things Journal, v. 8, n. 8, p. 6916–6927, 15 abr. 2021.

ASLAM, Asra; CURRY, Edward. A Survey on Object Detection for the Internet of Multimedia Things (IoMT) using Deep Learning and Event-based *Middleware*: Approaches, Challenges, and Future Directions. Image and Vision Computing, v. 106, p. 104095, fev. 2021.

ASTRONERGY. ASTRO-N5_CHSM78NDGF-BH, Bifacial Series (182). Disponível em: <https://www.astronergy.com/wp-content/uploads/2023/08/182610630ASTRO-N5_CHSM78NDGF-BH_2465x1134x30_EN_20230829.pdf.>. Acesso em: 22 jun. 2025.

BABU, S. Anish *et al.,* System Performance Evaluation of Para Virtualization, Container Virtualization, and Full Virtualization Using Xen, OpenVZ, and XenServer. *In*: IEEE, ago. 2014.

BAUER, Martin. FIWARE: Standard-based Open Source Components for Cross-Domain IoT Platforms. *In*: IEEE, 26 out. 2022.

BECKER, Soeren *et al.,* Towards AIOps in Edge Computing Environments. *In*: IEEE, 10 dez. 2020.

BEERMANN, Thomas *et al.,* Implementation of ATLAS Distributed Computing monitoring dashboards using InfluxDB and Grafana. EPJ Web of Conferences, v. 245, p. 03031, 16 nov. 2020.

BEYER, B. *et al.,* Site Reliability Engineering: How Google Runs Production Systems. *[S.l.]*: O'Reilly, 2016.

BEYER, Betsy *et al.,* The Site Reliability Workbook: Practical Ways to Implement SRE. 1st. ed. *[S.l.]*: O'Reilly Media, Inc., 2018.

BIN MOHD NAZRI, Mohd Syukri *et al.,* IoT Parking Apps with Car Plate Recognition for Smart City using Node Red. *In*: IEEE, abr. 2020.

BKHEET, Sana Abdelaziz; AGBINYA, Johnson I. A Review of Identity Methods of Internet of Things (IOT). Advances in Internet of Things, v. 11, n. 04, p. 153–174, 2021.

BOTEN, Alex; MAJORS, Charity. Cloud-Native Observability with OpenTelemetry: Learn to gain visibility into systems by combining tracing, metrics, and logging with OpenTelemetry. *[S.l.]*: Packt Publishing, 2022.

BRENNAN, M. P. *et al.,* Effects of spectral albedo on solar photovoltaic devices. Solar Energy Materials and Solar Cells, v. 124, p. 111–116, maio 2014.

CABUK, Ali Sinan. Air Quality Regulatory of Primary Care Clinics System Designed with IoT Using the Node-RED. Water, Air, & Soil Pollution, v. 236, n. 6, p. 370, 23 jun. 2025.

CAMPANA, Pietro Elia *et al.,* Optimization and assessment of floating and floating-tracking PV systems integrated in on- and off-grid hybrid energy systems. Solar Energy, v. 177, p. 782–795, jan. 2019.

CASTILHO, Sergio D.; GODOY, Eduardo P.; SALMEN, Fadir. Implementing Security and Trust in IoT/M2M using *Middleware*. *In*: IEEE, jan. 2020.

CHAKRABORTY, Mainak; KUNDAN, Ajit Pratap. Grafana. *In*: Monitoring Cloud-Native Applications. Berkeley, CA: Apress, 2021. p. 187–240.

CHETTRI, Lalit; BERA, Rabindranath. A Comprehensive Survey on Internet of Things (IoT) Toward 5G Wireless Systems. IEEE Internet of Things Journal, v. 7, n. 1, p. 16–32, jan. 2020.

CHIGBU, Uchendu Eugene; ATIKU, Sulaiman Olusegun; DU PLESSIS, Cherley C. The Science of Literature Reviews: Searching, Identifying, Selecting, and Synthesising. Publications, v. 11, n. 1, p. 2, 6 jan. 2023.

CIMINO, Antonio *et al.,* Integrating multiple industry 4.0 approaches and tools in an interoperable platform for manufacturing SMEs. Computers & Industrial Engineering, v. 186, p. 109732, dez. 2023.

COLAK, Ilhami; BAYINDIR, Ramazan; SAGIROGLU, Seref. The Effects of the Smart Grid System on the National Grids. *In*: IEEE, jun. 2020.

ČOLAKOVIĆ, Alem; HADŽIALIĆ, Mesud. Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues. Computer Networks, v. 144, p. 17–39, out. 2018.

CORREIA, Miguel; OLIVEIRA, Wellington; CECÍLIO, José. Monintainer: An orchestration-independent extensible container-based monitoring solution for large clusters. Journal of Systems Architecture, v. 145, p. 103035, dez. 2023.

CORTEZ, Eli *et al.,* Resource Central. *In*: New York, NY, USA: ACM, 14 out. 2017.

COSTA, António Pedro *et al.,* Integrating Bibliometrics and Qualitative Content Analysis for Conducting a Literature Review. *In*: IEEE, 6 dez. 2023.

CPQD. dojot documentation. Disponível em: <https://dojotdocs.readthedocs.io/en/latest/>. Acesso em: 24 abr. 2025.

CPQD. Crie valor a partir da integração entre mundo físico e digital com Internet das Coisas (IoT).

CUTRONA, Vincenzo *et al.,* Architecture of a *Software* Platform for Affordable Artificial Intelligence in Manufacturing. *In*: Artificial Intelligence in Manufacturing. Cham: Springer Nature Switzerland, 2024. p. 87–103.

DA CRUZ, Mauro A. A. *et al.,* A Reference Model for Internet of Things *Middleware*. IEEE Internet of Things Journal, v. 5, n. 2, p. 871–883, abr. 2018.

DANISH, Syed Muhammad; ZHANG, Kaiwen; JACOBSEN, Hans-Arno. BlockAM: An Adaptive *Middleware* for Intelligent Data Storage Selection for Internet of Things. *In*: IEEE, ago. 2020.

DE LA VEGA, Francisco *et al.,* Implementation of a Fiware-based Integration Platform and a Web Portal as Aids to Improve the Control of Ships Navigation in a River. *In*: IEEE, 12 out. 2020.

DEOHATE, Ankita; ROJATKAR, Dinesh. *Middleware* Challenges and Platform for IoT-A Survey. *In*: IEEE, 3 jun. 2021.

DESUERT, Arthur *et al.,* A *Middleware* for Secure Integration of Heterogeneous Edge Devices. *In*: IEEE, jul. 2022.

DHAKATE, Suchit; GODBOLE, Anand. Distributed cloud monitoring using Docker as next generation container virtualization technology. *In*: IEEE, dez. 2015.

DOCKER INC. What is Docker? Disponível em: <https://docs.docker.com/get-started/docker-overview/>. Acesso em: 4 maio. 2025.

DORDEVIC, Borislav; STEFANOVIC, Aleksandar; TIMCENKO, Valentina. File system performance comparison in the case of Docker container-based virtualization with Volumes and Bind mounts. *In*: IEEE, 15 nov. 2022.

DOS SANTOS SÁ, Joiner *et al.,* Multimodal urban mobility solutions for a smart campus using artificial neural networks for route determination and an algorithm for arrival time prediction. Engineering Applications of Artificial Intelligence, v. 137, p. 109074, nov. 2024.

DUA, Rajdeep *et al.,* Performance analysis of Union and CoW File Systems with Docker. *In*: IEEE, dez. 2016.

EL-DESSOUKI, Iman; SAEED, Nagham. Smart Grid Integration into Smart Cities. *In*: IEEE, 7 set. 2021.

EMPRESA DE PESQUISA ENERGÉTICA. Balanço Energético Nacional 2024: Ano base 2023. Rio de Janeiro: Empresa de Pesquisa Energética, 2024.

EYVAZOV, Farid *et al.,* Beyond Containers: Orchestrating Microservices with Minikube, Kubernetes, Docker, and Compose for Seamless Deployment and Scalability. *In*: IEEE, 14 mar. 2024.

FAVA, Felipe Bedinotto *et al.,* Assessing the Performance of Docker in Docker Containers for Microservice-Based Architectures. *In*: IEEE, 20 mar. 2024.

FIGUEREDO, Ken; SEED, Dale; SUBOTIC, Vanja. Preparing for Highly Scalable and Replicable IoT Systems. IEEE Internet of Things Magazine, v. 3, n. 3, p. 94–98, set. 2020.

FIWARE FOUNDATION. About FIWARE. Disponível em: <https://www.fiware.org/about-us/>. Acesso em: 12 maio. 2025a.

FIWARE FOUNDATION. FIWARE Catalogue. Disponível em: <https://www.fiware.org/catalogue/>. Acesso em: 12 maio. 2025b.

FIWARE FOUNDATION. Smart Data Models. Disponível em: <https://www.fiware.org/smart-data-models/>. Acesso em: 12 maio. 2025c.

FIWARE FOUNDATION. FIWARE-ORION. Disponível em: <https://fiware-orion.readthedocs.io/en/master/>. Acesso em: 12 maio. 2025d.

FIWARE FOUNDATION. NGSI-LD HowTo. Disponível em: <https://fiware-datamodels.readthedocs.io/en/stable/ngsi-ld_howto/>. Acesso em: 12 maio. 2025e.

GANESAN, K. *et al.,* Performance analysis of n-type PERT bifacial solar PV module under diverse albedo conditions. Solar Energy, v. 252, p. 81–90, mar. 2023.

GARROD, Brian. What Makes a Good Critical Literature Review Paper? Tourism and Hospitality, v. 4, n. 1, p. 141–147, 1 mar. 2023.

GENG, Yuqing; ZHANG, Naiguang; ZHU, Renjun. Research progress analysis of sustainable smart grid based on CiteSpace. Energy Strategy Reviews, v. 48, p. 101111, jul. 2023.

GIACOBBE, Maurizio *et al.,* An Implementation of InfluxDB for Monitoring and Analytics in Distributed IoT Environments. *In*: *[S.l.: S.n.]*. p. 155–162.

GOMES, Francisco A. A.; REGO, Paulo A. L.; TRINTA, Fernando A. M. Rumo a uma Taxonomia de Observabilidade para Aplicações Baseadas em Microsserviços. *In*: Sociedade Brasileira de Computação, 30 set. 2024.

GOOGLE. cAdvisor - Container Advisor. Disponível em: <https://github.com/google/cadvisor/blob/master/README.md>. Acesso em: 10 jun. 2025.

GOUMOPOULOS, Christos. Smart City *Middleware*: A Survey and a Conceptual Framework. IEEE Access, v. 12, p. 4015–4047, 2024.

GRAFANA LABS. Grafana k6. Disponível em: <https://k6.io/>. Acesso em: 28 jun. 2025.

GRAFANA LABS. Grafana documentation. Disponível em: <https://grafana.com/docs/grafana/latest/>. Acesso em: 11 jun. 2025.

HAGE HASSAN, Batoul *et al.,* Virtualization for performance guarantees of state estimation in cyber-physical energy systems. Energy Informatics, v. 5, n. S1, p. 30, 7 set. 2022.

HAGHGOO, Maliheh *et al.,* A cloud-based service-oriented architecture to unlock smart energy services. Energy Informatics, v. 4, n. 1, p. 8, 16 dez. 2021.

HAGINO, Taiji; O'LEARY, Nick. Practical Node-RED Programming: Learn powerful visual programming techniques and best practices for the web and IoT. Birmingham: Packt Publishing, 2021.

HASAN, Balqees Talal; ABDULLAH, Dhuha Basheer. Real-Time Resource Monitoring Framework in a Heterogeneous Kubernetes Cluster. *In*: IEEE, 16 mar. 2022.

HAZARI, Animesh. Review of Literature: Search Engines and Strategies. *In*: Research Methodology for Allied Health Professionals. Singapore: Springer Nature Singapore, 2023. p. 41–57.

HUAWEI TECNOLOGIES CO., Ltd. Virtualization Technology. *In*: Cloud Computing Technology. Singapore: Springer Nature Singapore, 2023. p. 97–144.

HYUNGJIK LEE; JEU. Design for management *software* of desktop virtualization solutions. *In*: IEEE, nov. 2010.

INFLUXDATA. InfluxDB OSS v2 Documentation. Disponível em: <https://docs.influxdata.com/influxdb/v2/>. Acesso em: 10 jun. 2025.

IOT LABS. Plataforma Dojot | CPQD.

JAIN, Shashank Mohan. Virtualization Basics. *In*: Linux Containers and Virtualization. Berkeley, CA: Apress, 2023. p. 1–14.

JARWAR, Muhammad Aslam *et al.,* Taking IoT Security to the Next Level: Hyperledger Fabric Private Blockchain Enabled IoT *Middleware*. *In*: IEEE, 4 dez. 2023.

JEPSEN, Sune Chung *et al.,* Industry 4.0 *Middleware Software* Architecture Interoperability Analysis. *In*: IEEE, jun. 2021.

JOHN, Petr *et al.,* A Self-Hosted Approach to Automatic CI/CD Using Open-Source Tools on Low-Power Devices. *In*: IEEE, 13 nov. 2024.

JOSÉ C. G. ANDRADE *et al.,* Ferramenta Computacional de Análise Integrada para Alocação de Estações de Recarga Públicas para Veículos Elétricos. *In*: 19 out. 2022.

KANG, Rui *et al.,* Distributed Monitoring System for Microservices-Based IoT *Middleware* System. *In*: *[S.l.: S.n.]*. p. 467–477.

KOPJAK, Jozsef; SEBESTYEN, Gergely. Event-driven Fuzzy Inference System Implementation in Node-RED. *In*: IEEE, set. 2019.

KORONTANIS, Ioannis *et al.,* Real-time Monitoring and Analysis of Edge and Cloud Resources. *In*: New York, NY, USA: ACM, 14 ago. 2023.

KOSIŃSKA, Joanna *et al.,* Toward the Observability of Cloud-Native Applications: The Overview of the State-of-the-Art. IEEE Access, v. 11, p. 73036–73052, 2023.

KREMS, Sebastian; REICH, Diana; STARK, Rainer. Virtualization technologies in product development: A cross-industrial user-study. *In*: IEEE, dez. 2017.

KUMAR, Shivansh *et al.,* Cloud-Native Countinous Integration/Continous Deployment (CI/CD) Pipeline. *In*: IEEE, 16 nov. 2024.

LAI, Wen-Ping; LAI, Hong-Lun. A Fast and Scalable Deployment Scheme of 5G Containerized Network Functions Using Docker Compose. *In*: IEEE, 9 jul. 2024.

LI, Guoqing *et al.,* The Convergence of Container and Traditional Virtualization: Strengths and Limitations. SN Computer Science, v. 4, n. 4, p. 387, 11 maio 2023.

LI, Ziyu. Comparison between common virtualization solutions: VMware Workstation, Hyper-V and Docker. *In*: IEEE, 12 nov. 2021.

LIANG CHEN; LONGCHUAN YAN; YIFAN MAO. Research and application of virtualization technology power in data center. *In*: IEEE, set. 2014.

LINGAMALLU, Phani Kumar; DE OLIVEIRA, Fabio Braga. AWS Observability Handbook: Monitor, trace, and alert your cloud applications with AWS' myriad observability tools. *[S.l.]*: Packt Publishing, 2023.

LISBOA, Yasmim *et al.,* Design and Implementation of a Sustainable IoT Embedded System for Monitoring Temperature and Humidity in Photovoltaic Power Plants in the Amazon. Sustainability, v. 17, n. 6, p. 2347, 7 mar. 2025a.

LLOPIS, Juan Alberto *et al.,* MI-FIWARE: A web component development method for FIWARE using microservices. *In*: IEEE, jul. 2021.

LOBATO, Elen *et al.,* A Monitoring System for Electric Vehicle Charging Stations: A Prototype in the Amazon. Energies, v. 16, n. 1, p. 152, 23 dez. 2022.

LOBATO, Elen Priscila de Souza *et al.,* Smart City: application of the ABNT NBR ISO 37122:2020 Standard in the University City of UFPA. *In*: IEEE, 15 ago. 2021.

LOSS, Stefano *et al.,* Using FIWARE and blockchain in smart cities solutions. Cluster Computing, v. 26, n. 4, p. 2115–2128, 7 ago. 2023.

MAJEED BUTT, Osama; ZULQARNAIN, Muhammad; MAJEED BUTT, Tallal. Recent advancement in smart grid technology: Future prospects in the electrical power network. Ain Shams Engineering Journal, v. 12, n. 1, p. 687–695, mar. 2021.

MAJORS, Charity; FONG-JONES, Liz; MIRANDA, George. Observability Engineering: Achieving Production Excellence. *[S.l.]*: O'Reilly Media, Inc, 2022.

MAO, Chen. Research and Design of Cloud Storage Server Based on Virtualization Technology. *In*: *[S.l.: S.n.]*. p. 423–428.

MARATHE, Nikhil; GANDHI, Ankita; SHAH, Jaimeel M. Docker Swarm and Kubernetes in Cloud Computing Environment. *In*: IEEE, abr. 2019.

MARQUESONE, Rosangela de Fatima Pereira *et al.,* A FIWARE-Based Component for Data Analysis in Smart Mobility Context. *In*: IEEE, ago. 2017.

MEDIUM. What and Why of Docker Compose?

MEHDI, Abbas *et al.,* "Unleashing the Potential of Grafana: A Comprehensive Study on Real-Time Monitoring and Visualization". *In*: IEEE, 6 jul. 2023.

MINISTÉRIO DE MINAS E ENERGIA; EMPRESA DE PESQUISA ENERGÉTICA. Plano Nacional de Energia 2050. Brasília, 2020. Disponível em: <https://www.epe.gov.br/sites-pt/publicacoes-dados-abertos/publicacoes/PublicacoesArquivos/publicacao-227/topico-563/Relatorio%20Final%20do%20PNE%202050.pdf>. Acesso em: 11 set. 2024

MISBAHUDDIN, Misbahuddin *et al.,* Kalman Filter-Enhanced Data Aggregation in LoRaWAN-Based IoT Framework for Aquaculture Monitoring in Sargassum sp. Cultivation. Computers, v. 14, n. 4, p. 151, 18 abr. 2025.

MORAES, Jean *et al.,* Implementação de um cluster Kubernetes com a plataforma Dojot para Aplicações de Internet das Coisas. *In*: Sociedade Brasileira de Computação - SBC, 18 jul. 2021.

MU, Tianshi *et al.,* A user-friendly attribute-based data access control scheme for smart grids. Alexandria Engineering Journal, v. 67, p. 209–217, mar. 2023.

NATU, Maitreya *et al.,* Holistic Performance Monitoring of Hybrid Clouds: Complexities and Future Directions. IEEE Cloud Computing, v. 3, n. 1, p. 72–81, jan. 2016.

NEWMAN, M. E. J. Fast algorithm for detecting community structure in networks. Physical Review E, v. 69, n. 6, p. 066133, 18 jun. 2004.

NGUYEN, Tu N. *et al.,* Smart Grid Vulnerability and Defense Analysis Under Cascading Failure Attacks. IEEE Transactions on Power Delivery, v. 36, n. 4, p. 2264–2273, ago. 2021. NIEDERMAIER, Sina *et al.,* On Observability and Monitoring of Distributed Systems – An Industry Interview Study. *In*: *[S.l.: S.n.]*. p. 36–52.

NOACK, Andreas. Energy Models for Graph Clustering. Journal of Graph Algorithms and Applications, v. 11, n. 2, p. 453–480, 2007.

NOACK, Andreas. Modularity clustering is force-directed layout. Physical Review E, v. 79, n. 2, p. 026102, 2 fev. 2009.

ODUN-AYO, Isaac; AJAYI, Olasupo; OKEREKE, Chinonso. Virtualization in Cloud Computing: Developments and Trends. *In*: IEEE, dez. 2017.

OLZAK, Thomas *et al.,* Microsoft Virtualization. *[S.l.]*: Elsevier, 2010.

OPENJS FOUNDATION. About Node-RED. Disponível em: <https://nodered.org/about/>. Acesso em: 12 maio. 2025.

ORAZI, Gilles *et al.,* Edge-Cloud Solution Based on FIWARE and Context Augmentation to Monitor Usages of Carpooling Car Parks. *In*: *[S.l.: S.n.].* p. 134–150.

PATRÃO, Luciano. Understanding Virtualization. *In*: VMware vSphere Essentials. Berkeley, CA: Apress, 2024. p. 1–7.

PERATA, Juan Pablo; BETARTE, Gustavo. A Security Analysis of a Referential Architecture of the FIWARE Platform. *In*: IEEE, 16 out. 2023.

PEREIRA, Pedro Henrique Morgan *et al.,* Interoperability *Middleware* for IIoT Gateways based on Standard Ontologies and AAS. IFAC-PapersOnLine, v. 56, n. 2, p. 9831–9836, 2023.

PEROS, Stefanos; JOOSEN, Wouter; HUGHES, Danny. Ermis: a *middleware* for bridging data collection and data processing in IoT streaming applications. *In*: IEEE, jul. 2021.

PORTNOY, Matthew. Virtualization Essentials. 3. ed. Hoboken, New Jersey: John Wiley & Sons, Inc., 2023.

PRADEEP, Preeja; KRISHNAMOORTHY, Shivsubramani; VASILAKOS, Athanasios V. A holistic approach to a context-aware IoT ecosystem with Adaptive Ubiquitous *Middleware*. Pervasive and Mobile Computing, v. 72, p. 101342, 1 abr. 2021.

PROMETHEUS AUTHORS. Overview - Prometheus Documentation. Disponível em: <https://prometheus.io/docs/introduction/overview/>. Acesso em: 10 jun. 2025.

PROXMOX SERVER SOLUTIONS GMBH. Proxmox VE Administration Guide. Disponível em: <https://pve.proxmox.com/pve-docs/pve-admin-guide.html>. Acesso em: 11 set. 2024.

QAYS, Md. Ohirul *et al.,* Key communication technologies, applications, protocols and future guides for IoT-assisted smart grid systems: A review. Energy Reports, v. 9, p. 2440–2452, dez. 2023.

RATTANAPOKA, Choopan *et al.,* An MQTT-based IoT Cloud Platform with Flow Design by Node-RED. *In*: IEEE, dez. 2019.

REIS, David *et al.,* Developing Docker and Docker-Compose Specifications: A Developers' Survey. IEEE Access, v. 10, p. 2318–2329, 2022.

ROSLAN, Lidyana *et al.,* Real-Time Monitoring of Modular Carbon Capture Machines with Sensor Networks and Node-RED. *In*: IEEE, 30 jul. 2024.

ROUTRAY, Sudhir K. *et al.,* Narrowband IoT (NBIoT) Assisted Smart Grids. *In*: IEEE, 25 mar. 2021.

SALIH, E. S. Ehsan's Three Tables Model: A comprehensive guide for identifying research gaps and conducting systematic literature reviews in business and economics studies. Review of Business and Economics Studies, v. 12, n. 3, p. 102–114, 31 out. 2024.

SAMWINI, Hezekiah; AWAIS, Muhammad; PEREIRA, Ella. *Middleware* for Resource Sharing in Fog Computing with IoT Applications. *In*: IEEE, 6 jul. 2023.

SANTOS, Sofia C. *et al.,* An IoT Rainfall Monitoring Application based on Wireless Communication Technologies. *In*: IEEE, 7 out. 2020.

SHARMA, Bhavye; NADIG, Deepak. eBPF-Enhanced Complete Observability Solution for Cloud-native Microservices. *In*: IEEE, 9 jun. 2024.

SHEN, Yi *et al.,* Building Data Management Systems for Power Grid Companies Using Data *Middleware* Architecture. *In*: IEEE, 26 out. 2023.

SHINDE, Ashutosh. Challenges in performance monitoring of hyper connected IoT systems. *In*: IEEE, jan. 2016.

SIDDIQUI, Ishan *et al.,* Comprehensive Monitoring and Observability with Jenkins and Grafana: A Review of Integration Strategies, Best Practices, and Emerging Trends. *In*: IEEE, 26 out. 2023.

SMA SOLAR TECHNOLOGY AG. Sunny Tripower X_12/15/20/25. Disponível em: <https://files.sma.de/downloads/STPxx-50-DS-en-21.pdf.>. Acesso em: 22 jun. 2025a.

SMA SOLAR TECHNOLOGY AG. SMA Monitoring Platform - Sandbox.

SONG, Eugene Y. *et al.,* A Methodology for Modeling Interoperability of Smart Sensors in Smart Grids. IEEE Transactions on Smart Grid, v. 13, n. 1, p. 555–563, jan. 2022.

SOUKI, Olfa *et al.,* A Survey of *Middleware*s for self-adaptation and context-aware in Cloud of Things environment. Procedia Computer Science, v. 207, p. 2804–2813, 1 jan. 2022.

STOCK, Daniel; SCHEL, Daniel; BAUERNHANSL, Thomas. *Middleware*-based Cyber-Physical Production System Modeling for Operators. Procedia Manufacturing, v. 42, p. 111–118, 1 jan. 2020.

SUN, Chang-Ai *et al.,* A Trace-Log-Clusterings-Based Fault Localization Approach to Microservice Systems. *In*: IEEE, jul. 2023.

SUNG, Guo-Ming *et al.,* An Artificial Intelligence Home Monitoring System That Uses CNN and LSTM and Is Based on the Android Studio Development Platform. Applied Sciences, v. 15, n. 3, p. 1207, 24 jan. 2025.

SÜSS, Ralf; SÜSS, Yannik. Storage and Virtualization. *In*: IT Infrastructure. Berkeley, CA: Apress, 2024. p. 107–160.

TEAM, FreeWheel Biz-UI. Service Observability. *In*: Cloud-Native Application Architecture. Singapore: Springer Nature Singapore, 2024. p. 213–281.

TERADA, Lucas Zenichi *et al.,* Evaluation of an IoT-based Smart Charging Algorithm for Electric Vehicles Considering Multiprocessing. *In*: IEEE, 24 out. 2022.

THANTHARATE, Pratik. IntelligentMonitor: Empowering DevOps Environments with Advanced Monitoring and Observability. *In*: IEEE, 9 ago. 2023.

TIAN, Yu-Chu; GAO, Jing. Virtualization and Cloud. *In*: *[S.l.: S.n.]*. p. 447–499.

TOLARAM, Nanik. cadvisor. *In*: *Software* Development with Go. Berkeley, CA: Apress, 2023. p. 347–376.

TRUNZER, Emanuel *et al.,* Comparison of Communication Technologies for Industrial *Middleware*s and DDS-based Realization. IFAC-PapersOnLine, v. 53, n. 2, p. 10935–10942, 1 jan. 2020.

TSENG, Chun-Kai *et al.,* Developing Frugal Internet of Things with Backpropagation Neural Network for Predicting Impact of Gemini Artificial Intelligence on Student Meditation and Relaxation. *In*: Basel Switzerland: MDPI, 17 abr. 2025.

VAZQUEZ, Antonio. Virtualization Concepts and Theory. *In*: *[S.l.: S.n.]*. p. 1–14, 2024a.

VAZQUEZ, Antonio. Container Virtualization Concepts. *In*: *[S.l.: S.n.]*. p. 319–362, 2024b.

VELASQUEZ, Washington; TOBAR-ANDRADE, Luis; CEDENO-CAMPOVERDE, Ivan. Monitoring and Data Processing Architecture using the FIWARE Platform for a Renewable Energy Systems. *In*: IEEE, 27 jan. 2021a.

VELASQUEZ, Washington; TOBAR-ANDRADE, Luis; CEDENO-CAMPOVERDE, Ivan. Monitoring and Data Processing Architecture using the FIWARE Platform for a Renewable Energy Systems. *In*: IEEE, 27 jan. 2021b.

VOUMICK, Dipta; DEB, Prince; KHAN, Mohammad Monirujjaman. Operation and Control of Microgrids Using IoT (Internet of Things). Journal of *Software* Engineering and Applications, v. 14, n. 08, p. 418–441, 2021.

WALTER-TSCHARF, Viktor. Practical Approach to Monitor Runtime Engine Statistics for Apache Spark and Docker Swarm using Graphite, Grafana, and CollectD. *In*: IEEE, 27 out. 2021.

YE JIAOJIAO; SHANG YANMING. An overview of open-source virtualization technology. *In*: Institution of Engineering and Technology, 2013.

YONGGUO, Jiang *et al.,* Message-oriented *Middleware*: A Review. *In*: IEEE, ago. 2019.

ZHANG, Jingbin *et al., Middleware* for the Internet of Things: A survey on requirements, enabling technologies, and solutions. Journal of Systems Architecture, v. 117, p. 102098, ago. 2021