UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Davi Grumiche Seemann

Arquitetura Multitier para Monitoramento em IoT: Um Estudo de Caso com Usinas Solares

Florianópolis, SC

2024

Davi Grumiche Seemann

Arquitetura Multitier para Monitoramento em IoT: Um Estudo de Caso com Usinas Solares

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia Elétrica.
Orientador: Prof. Richard Demo Souza, Dr

Florianópolis, SC

2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Seemann, Davi Grumiche
Arquitetura Multitier para Monitoramento em IoT : Um Estudo de Caso com Usinas Solares / Davi Grumiche Seemann ; orientador, Richard Demo Souza, 2024.
82 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2024.

Inclui referências.

1. Engenharia Elétrica. 2. Internet das Coisas. 3. Processamento de Dados. 4. Computação em Nuvem. 5. Monitoramento. I. Souza, Richard Demo. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

Davi Grumiche Seemann

Arquitetura Multitier para Monitoramento em IoT: Um Estudo de Caso com Usinas Solares

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia Elétrica" e aceito, em sua forma final, pelo Curso de Graduação em Engenharia Elétrica.

Florianópolis, 09 de dezembro de 2024.

Documento assinado digitalmente
Miguel Moreto
Data: 12/12/2024 14:26:21-0300
CPF: ***.850.100-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Miguel Moreto, Dr.
Coordenador do Curso de Graduação em Engenharia Elétrica

Banca Examinadora:

Documento assinado digitalmente
Richard Demo Souza
Data: 12/12/2024 14:21:04-0300
CPF: ***.267.379-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Richard Demo Souza, Dr.
Orientador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Eduardo Luiz Ortiz Batista
Data: 12/12/2024 16:31:15-0300
CPF: ***.521.889-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Luiz Ortiz Batista, Dr.
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Samir Ahmad Mussa
Data: 12/12/2024 15:49:43-0300
CPF: ***.044.430-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Samir Ahmad Mussa, Dr.
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Primeiramente, gostaria de agradecer aos meus pais, Marcello e Daniela, por todo o apoio e incentivo que me permitiram chegar até aqui. Seus ensinamentos são a base fundamental da minha formação como pessoa, e os valores que aprendi com vocês os levo comigo para onde quer que eu vá. Tudo isso só foi possível porque vocês nunca deixaram de acreditar em mim. Vocês são, sem dúvida, a minha maior inspiração.

Agradeço também à minha namorada, Julia, por ser, muitas vezes, a alegria dos meus dias. Seu apoio nos momentos difíceis e sua parceria para superar cada desafio que a vida nos impõe foram fundamentais. Você fez toda a diferença na minha trajetória até aqui, e sou profundamente grato por tê-la ao meu lado.

Na universidade, gostaria de expressar minha gratidão a todos os professores do Departamento de Engenharia Elétrica e Eletrônica e do Centro Tecnológico, que me formaram academicamente e contribuíram para que eu me tornasse o profissional que sou hoje. Vocês me proporcionaram as primeiras oportunidades e foram peças-chave no meu desenvolvimento. Em especial, agradeço aos professores André Luís Kirsten e Marcelo Ricardo Stemmer por acreditarem em mim e por me proporcionarem grandes oportunidades de crescimento técnico e pessoal, tanto durante minha trajetória como petiano quanto na realização do intercâmbio no exterior.

Quero também agradecer aos meus colegas de graduação, com quem tive o privilégio e a honra de compartilhar desafios e conquistas ao longo da jornada acadêmica, nos grupos de extensão e durante o intercâmbio. Vocês me incentivaram em muitos momentos e tiveram um papel importante na formação do profissional que me tornei. Aos colegas do Instituto Fraunhofer IPT – Gustavo, Rafael e Fernando –, o período que compartilhei com vocês foi um dos mais importantes da minha carreira. Também aos colegas Victor e Witold, que me ensinaram tanto sobre engenharia de dados e engenharia de software, vocês foram muito importantes na inspiração para o desenvolvimento deste trabalho.

Por fim, estendo minha gratidão a todos os outros amigos e colegas que estiveram ao meu lado, cada um me apoiando à sua maneira. Cada contribuição foi essencial para que eu pudesse trilhar este caminho.

RESUMO

No contexto do monitoramento de usinas solares, este trabalho propõe o desenvolvimento de uma plataforma de dados baseada na arquitetura multitier para sistemas IoT. A plataforma foi projetada para otimizar a coleta, processamento e análise de dados provenientes de sensores de inversores solares. O objetivo central é oferecer uma solução eficiente e escalável, capaz de processar grandes volumes de dados e suportar análises avançadas em tempo real. A arquitetura foi implementada em duas camadas: a camada bronze armazena os dados brutos recebidos de sensores IoT, preservando a rastreabilidade e integridade, enquanto a camada prata realiza limpeza e normalização dos dados, tornando-os prontos para análises e consumo por sistemas externos. O sistema foi implementado com microsserviços orquestrados no Kubernetes, que garantem modularidade e escalabilidade. A validação incluiu testes de carga, destacando a robustez e eficiência na ingestão e no processamento dos dados, com entregas confiáveis e capacidade de lidar com cenários de alta demanda. Os resultados demonstraram a eficácia da plataforma em atender demandas específicas do monitoramento de usinas solares, oferecendo flexibilidade para adaptação a outros setores industriais, com destaque para a preservação da rastreabilidade dos dados brutos e a eficiência no processamento de informações em tempo real.
Palavras-chave: Arquitetura Multitier. Computação em Nuvem. Internet das Coisas. Monitoramento. Processamento de Dados.

ABSTRACT

Within the realm of solar plant monitoring, this study proposes the design of a data platform predicated on a multitier architecture for IoT systems. The platform is engineered to enhance the collection, processing, and analysis of data emanating from solar inverter sensors. The principal objective is to provide an efficient and scalable solution capable of managing substantial data volumes while supporting sophisticated real-time analysis. The architecture is implemented in two stratified layers: the bronze layer is responsible for storing raw data acquired from IoT sensors, thereby ensuring the preservation of traceability and integrity, whereas the silver layer is tasked with cleansing and normalizing the data, rendering it suitable for analysis and consumption by external systems. The system's deployment utilizes microservices orchestrated via Kubernetes, thereby guaranteeing modularity and scalability. The validation process encompassed load tests, underscoring the system's robustness and efficiency in data ingestion and processing, with dependable delivery and the capacity to manage high-demand scenarios. The outcomes corroborate the platform's efficacy in addressing the particular requirements of solar plant monitoring, while also offering adaptability for application within other industrial sectors, with particular emphasis on preserving the traceability of raw data and efficiency in real-time information processing.
Keywords: Cloud Computing. Data Processing. Internet of Things. Monitoring. Multitier Architecture.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACID	*Atomicity, Consistency, Isolation, Durability*
AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interface*
AWS	*Amazon Web Services*
Azure	*Microsoft Azure*
CA	Corrente Alternada
CC	Corrente Contínua
CI/CD	*Continuous Integration/Continuous Deployment*
CoAP	*Constrained Application Protocol*
CSV	*Comma-Separated Values*
DNS	*Domain Name System*
GCP	*Google Cloud Platform*
GHCR	*GitHub Container Registry*
GKE	*Google Kubernetes Engine*
HTTP	*Hypertext Transfer Protocol*
IoT	*Internet of Things*
JSON	*JavaScript Object Notation*
MQTT	*Message Queuing Telemetry Transport*
PV	Fotovoltaicos
PVDAQ	*Photovoltaic Data Acquisition*
QoS	*Quality of Service*
REST	*Representational State Transfer*
SQL	*Structured Query Language*
TLS	*Transport Layer Security*
UDP	*User Datagram Protocol*
UTC	*Universal Time Coordinated*

CONTEÚDO

1 INTRODUÇÃO

Ano após ano, a Internet das Coisas (IoT) consolida-se como uma tecnologia transformadora, revolucionando a análise e o monitoramento de processos. Conforme demonstra Joshi, Mamaniya e Shah (2022), nos últimos anos, essa tecnologia tem sido um dos principais impulsionadores da crescente Indústria 4.0, orientada a processos inteligentes e baseados em dados. A adoção da IoT tem transformado não apenas a indústria, mas também áreas como saúde, transporte, agricultura e muitas outras, tornando os processos mais eficientes e integrados.

A IoT refere-se à interconexão de dispositivos físicos por meio da internet, permitindo a coleta e troca de dados. Incluem-se dispositivos que variam de sensores simples a sistemas eletrônicos complexos, possibilitando maior automação e inteligência em diversos sistemas (ASHTON, 2009). A capacidade de interligar dispositivos e sistemas abre uma gama de oportunidades para a inovação, desde a otimização de processos industriais até a criação de cidades inteligentes, onde sistemas de trânsito, energia e segurança trabalham de forma coordenada e eficiente.

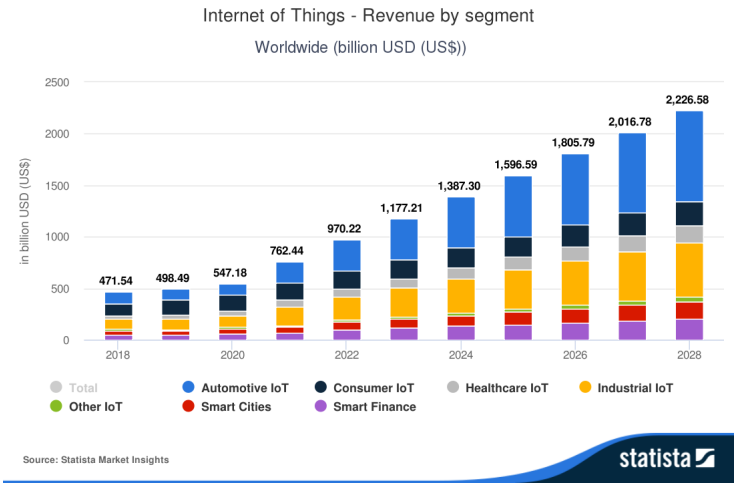
1.1 CRESCIMENTO E IMPORTÂNCIA DA IOT

Com a evolução dos dispositivos e a democratização da tecnologia, a área de IoT se expandiu significativamente, diversificando as formas de desenvolver sistemas que se comunicam utilizando redes. Nos últimos anos, o crescimento das tecnologias de computação e armazenamento em nuvem criou um novo capítulo na área. Atualmente, é possível desenvolver soluções utilizando a IoT sem a necessidade de uma grande infraestrutura. A computação em nuvem oferece recursos escaláveis e acessíveis, permitindo que empresas de todos os tamanhos implementem soluções de IoT de forma eficiente e econômica.

Um relatório da Statista Market Insights (2024) apresenta projeções otimistas para a área de IoT, indicando um forte crescimento nas receitas e no número de conexões IoT ao redor do mundo, conforme ilustrado na Figura 1 e na Figura 2. Esses dados reforçam o papel estratégico da IoT como catalisadora de mudanças na forma como as organizações operam e inovam. O ambiente de IoT parece ter um futuro bastante promissor, com grande capacidade de expansão e de se tornar um importante pilar dentro das corporações neste ambiente pós-revolução digital. As organizações, ao digitalizarem suas operações, percebem a verdadeira eficiência de seus

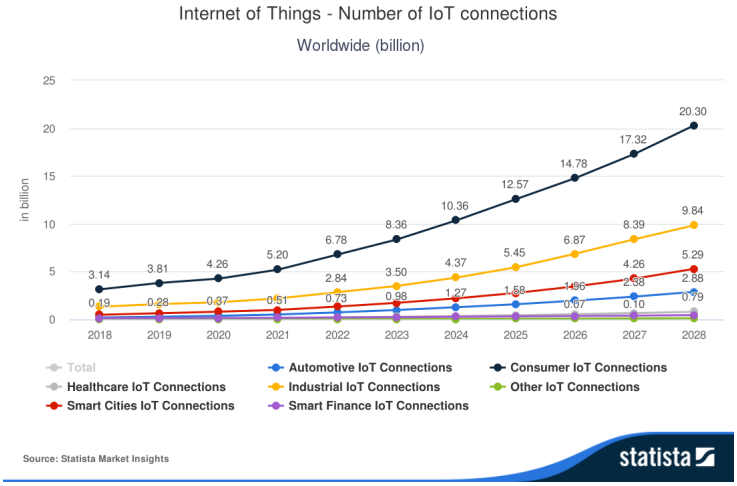
processos e identificam os potenciais e limites de sua operação de maneira dinâmica e contínua.

Figura 1 – Receita por segmento (em bilhões de dólares)



Fonte: Statista Market Insights (2024).

Figura 2 – Número de conexões IoT no mundo (em bilhões)



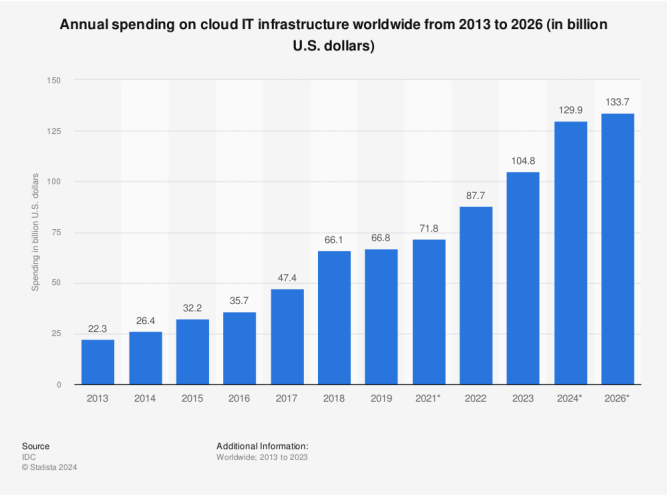
Fonte: Statista Market Insights (2024).

1.2 COMPUTAÇÃO EM NUVEM E SUPORTE À IOT

Com a crescente demanda por produtos decorrentes da transformação digital, como o aumento da geração e aquisição de dados, a infraestrutura necessária para o processamento desses dados teve que se expandir de maneira agressiva. O mercado de infraestrutura de TI na nuvem tem mostrado um crescimento contínuo nos últimos anos, com previsões de aumento até

2026. A Figura 3 ilustra os gastos mundiais no mercado de infraestrutura de TI na nuvem de 2013 a 2026, refletindo a crescente importância dessa tecnologia.

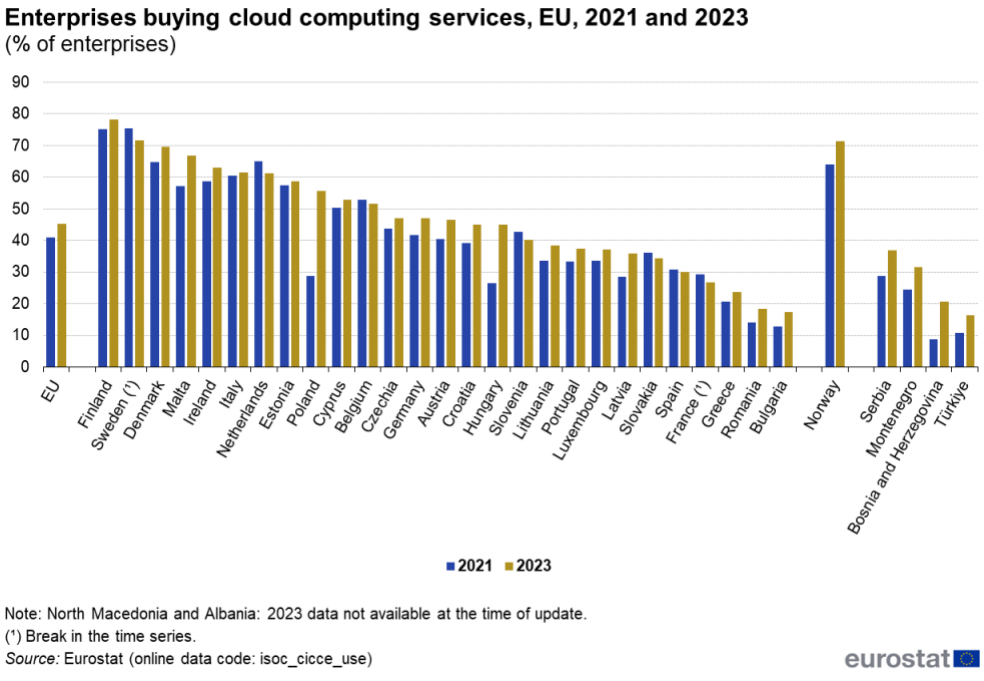
Figura 3 – Gastos mundiais no mercado de infraestrutura de TI na nuvem (2013-2026).



Fonte: Statista (2024).

Além disso, a Figura 4 ilustra a evolução da adoção de serviços de computação em nuvem por empresas de diferentes países europeus entre 2021 e 2023.

Figura 4 – Empresas comprando serviços de computação em nuvem



Fonte: Eurostat (2023).

De acordo com o relatório da Synergy Research Group (2024), no primeiro trimestre de

2024, os gastos mundiais com serviços de infraestrutura em nuvem alcançaram mais de US$76 bilhões, um aumento de US$13,5 bilhões (21%) em comparação com o primeiro trimestre de 2023. Amazon, Microsoft e Google lideram o mercado com participações de 31%, 25% e 11%, respectivamente.

1.3 DESAFIOS ATUAIS NAS SOLUÇÕES DE IOT

Apesar do crescimento, muitas soluções existentes enfrentam desafios como custos elevados, falta de customização e descontinuidade de serviços, exemplificado pela descontinuação do *GCP IoT Hub* (DOMINGUEZ, 2022). Essas limitações dificultam a implementação de fluxos de dados para sistemas de IoT, aumentando a necessidade de desenvolver soluções inovadoras e acessíveis.

1.4 MOTIVAÇÃO PARA O TRABALHO

O rápido crescimento da IoT e da computação em nuvem tem gerado uma demanda crescente por soluções que facilitem o processamento e a disponibilização de dados para diversas aplicações. Com a expansão desses ecossistemas, surge a necessidade de produtos mais robustos e flexíveis que possam gerenciar grandes volumes de dados de forma eficiente. Aplicações como análise de dados e aprendizado de máquina dependem diretamente da qualidade e da acessibilidade dos dados disponíveis, impactando significativamente seu desempenho e precisão. Desenvolver plataformas que ofereçam fluxos de dados otimizados e sistemas de comunicação confiáveis é fundamental para atender a essa demanda, garantindo que os dados coletados sejam processados de maneira eficaz e estejam prontamente disponíveis para análises avançadas.

Além disso, a implementação de sistemas de comunicação eficientes e fluxos de dados bem estruturados não só potencializa o desempenho das aplicações mencionadas, mas também contribui para a sustentabilidade operacional das organizações. Embora o foco principal deste trabalho seja a criação de uma plataforma de monitoramento para uma usina solar, a solução proposta tem potencial para ser adaptada a diversos outros setores, promovendo inovação e eficiência em múltiplas áreas. Dessa forma, este trabalho busca abordar a lacuna existente no mercado de soluções IoT, oferecendo uma infraestrutura que suporta o crescimento contínuo dessas tecnologias e facilitando a integração de dados para aplicações críticas como aprendizado de máquina e análise de dados.

1.4.1 Objetivo geral

O principal objetivo deste trabalho é o desenvolvimento de uma plataforma capaz de monitorar equipamentos, com foco em uma planta solar. A plataforma deve ser capaz de receber *payloads* (pacotes), processá-los e armazená-los em um banco de dados que permita o manejo e a utilização pelos usuários. Toda a infraestrutura será desenvolvida utilizando recursos de computação em nuvem, e utilizando sempre que possível ferramentas e tecnologias de código aberto.

1.4.2 Objetivos específicos

Para atingir o objetivo geral proposto, foram definidos os seguintes objetivos específicos:

- Estudar sistemas IoT e protocolos de comunicação para identificar as tecnologias mais adequadas ao caso de uso.
- Analisar as principais ferramentas e infraestruturas de nuvem disponíveis para desenvolvimento de fluxos de dados.
- Desenvolver uma arquitetura de processamento de dados eficiente.
- Modelar os dados e implementar um fluxo de dados otimizado.
- Testar o funcionamento da plataforma utilizando dados reais e emulação de mensagens.

1.5 RESUMO DO DOCUMENTO

Este trabalho está organizado da seguinte forma para facilitar a compreensão e o acompanhamento dos tópicos abordados:

- Capítulo 2: Apresenta uma revisão literária dos conceitos a serem abordados na metodologia deste trabalho.
- Capítulo 3: Detalha a metodologia de desenvolvimento da solução.
- Capítulo 4: Apresenta os a arquitetura implementada e os resultados alcançados em termos de desempenho da solução.
- Capítulo 5: Conclui o trabalho com considerações finais.

2 REVISÃO DA LITERATURA

Este capítulo apresenta os conceitos fundamentais utilizados no desenvolvimento da trabalho, com foco em IoT, Engenharia de Dados, Arquiteturas Escaláveis em Nuvem e as ferramentas relevantes para o desenvolvimento de sistemas. Essas informações servirão como base para a compreensão dos conceitos e metodologias empregados neste trabalho.

2.1 INTERNET DAS COISAS (IOT)

A IoT refere-se à interconexão de dispositivos físicos com a Internet, possibilitando a coleta e troca de dados. Essa interconexão abrange uma ampla gama de dispositivos, desde sensores simples até sistemas complexos, e tem transformado setores como manufatura, saúde, transporte e energia (AL-FUQAHA; GUIZANI *et al.*, 2015). A IoT permite a automação e otimização de processos, gerando dados massivos que precisam ser coletados, armazenados e analisados de maneira eficiente (LIN; YU *et al.*, 2017). Assim, infraestruturas escaláveis e a integração com sistemas de Engenharia de Dados tornam-se essenciais.

2.1.1 Fundamentos de Sistemas IoT

Os sistemas IoT podem ser organizados em camadas como pode ser observado na Tabela 1, incluindo dispositivos, rede, processamento, serviços e interface de usuário. Sensores e atuadores capturam dados, que são transmitidos para processamento em plataformas de nuvem, como *Amazon Web Services* (AWS) e GCP (LIN; YU *et al.*, 2017). Essas plataformas oferecem recursos de armazenamento escalável e processamento em tempo real, necessários para lidar com os desafios de dados massivos.

Tabela 1 – Uma Arquitetura de Quatro Camadas para IoT

Camadas	**Descrição**
Camada de Sensoriamento	Dispositivos para monitorar o ambiente e adquirir dados.
Camada de Rede	Transmissão de dados por redes sem fio ou cabeadas.
Camada de Serviço	Gestão e disponibilização de serviços.
Camada de Interface	Métodos de interação entre usuários e dispositivos.

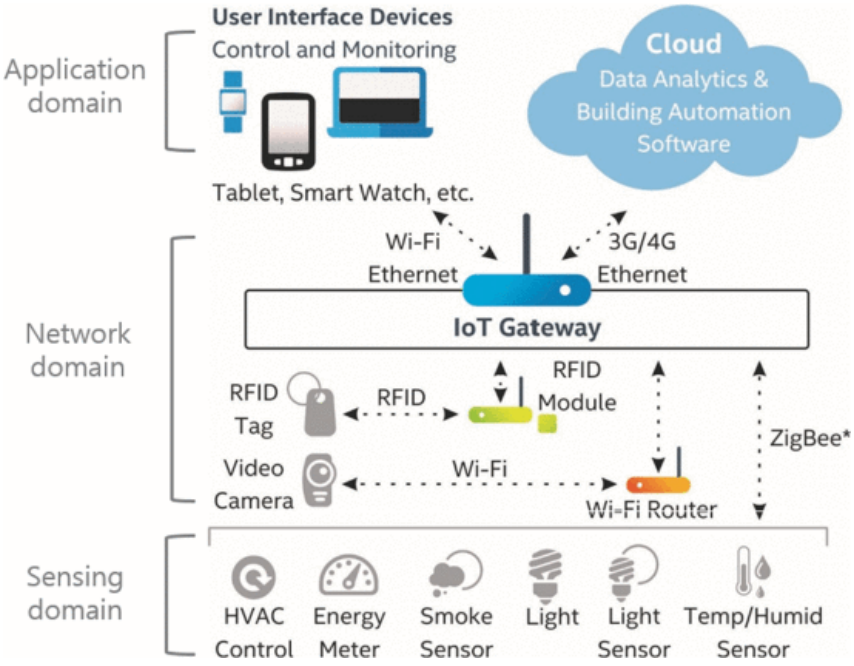
Fonte: Adaptado de Biswas e Giaffreda (2014)

Este trabalho terá como foco a camada de serviço, responsável por toda a lógica de processamento e controle de qualidade dos dados dentro do sistema. A proposta envolve o desenvolvimento de uma solução personalizada e independente para essa camada, garantindo maior controle e adaptabilidade às necessidades específicas do projeto.

A camada de rede, conforme ilustrada na Figura 5, desempenha a função de conectar sensores a redes externas, empregando um IoT *Gateway* para gerenciar a comunicação. Os sensores transmitem os dados coletados por meio de protocolos como *ZigBee*, *Bluetooth* ou *Wi-Fi* ao *Gateway*, que converte os dados para formatos compatíveis e os encaminha à internet utilizando conexões como *Ethernet* ou *4G/5G*. O *Gateway* atua como uma interface entre dispositivos heterogêneos e a nuvem, proporcionando interoperabilidade e suporte a diversas configurações de rede, conforme destacado por Kang, Kim e Choo (2017).

Ademais, neste estudo, os termos "sensor"e *"gateway"* referem-se a dispositivos com capacidade de comunicação em rede, configurados para estabelecer conexão com a Internet. É importante destacar que a camada de rede entre os dispositivos locais não será examinada em profundidade, pois o foco reside na plataforma de processamento de dados na nuvem, com ênfase no recebimento e gerenciamento de dados oriundos de múltiplos dispositivos.

Figura 5 – IoT *Gateway* para funcionalidades gerais.



Fonte: Kang, Kim e Choo (2017)

2.1.2 Protocolos e Arquitetura de Comunicação

Os protocolos de comunicação desempenham um papel fundamental nos sistemas IoT, sendo responsáveis por assegurar a troca eficiente e segura de dados entre dispositivos. Esses protocolos operam na camada de aplicação e são projetados para atender a requisitos específicos, como baixa latência, segurança e escalabilidade.

2.1.2.1 Protocolos de Comunicação em IoT

Diversos protocolos foram desenvolvidos para IoT, cada qual com características e aplicações específicas:

- **MQTT**: Protocolo leve projetado para redes com largura de banda limitada. Amplamente utilizado em monitoramento em tempo real seu modelo *publish/subscribe* suporta *Quality of Service* (QoS) configurável (ŽIVIĆ; NEMEC; BOJOVIĆ, 2023). Apresenta baixo overhead e fácil implementação, mas requer camadas adicionais, como *Transport Layer Security* (TLS), para segurança.

- ***Constrained Application Protocol* (CoAP)**: Protocolo baseado em *Representational State Transfer* (REST), otimizado para dispositivos com restrições de energia e capacidade. Utiliza *User Datagram Protocol* (UDP) para reduzir consumo de recursos (AMJAD; AZAM *et al.*, 2021). É simples e compatível com APIs *RESTful*, mas possui suporte limitado a entrega garantida, o que pode ser crítico em algumas aplicações.

- ***Advanced Message Queuing Protocol* (AMQP)**: Protocolo robusto, projetado para alta confiabilidade e interoperabilidade em sistemas industriais. Suporta enfileiramento, roteamento e transações, mas seu *overhead* elevado pode dificultar a aplicação em dispositivos com recursos limitados (AL-MASRI *et al.*, 2020).

- ***ZeroMQ***: Biblioteca de mensagens de alto desempenho, usada em sistemas que demandam alta taxa de transferência, como análise de vídeo em tempo real. É flexível e escalável, mas sua configuração inicial é complexa e possui suporte limitado para QoS (ŽIVIĆ; NEMEC; BOJOVIĆ, 2023).

2.1.2.2 Comparações de Protocolos

De acordo com Al-Masri *et al.* (2020), a escolha do protocolo depende das necessidades específicas da aplicação IoT. Por exemplo, para aplicações de monitoramento contínuo em tempo real, o MQTT é uma escolha ideal devido ao seu baixo consumo de recursos. Já em aplicações que requerem interoperabilidade com sistemas baseados na web, o CoAP oferece vantagens significativas. Um comparativo entre os aspectos de cada protocolo é apresentado na Tabela 2.

Tabela 2 – Comparação de Protocolos IoT

Protocolo	**Modelo**	**Uso Principal**	**Vantagens**	**Limitações**
MQTT	*Publish/Subscribe*	Monitoramento em tempo real	Leve, eficiente	Segurança limitada
CoAP	*Request/Response*	Integração Web	Simplicidade, RESTful	Garantia limitada de entrega
AMQP	*Message Queuing*	Sistemas industriais	Alta confiabilidade	Overhead elevado
ZeroMQ	*Messaging Library*	Análise de alto desempenho	Alta taxa de transferência	Configuração complexa

Fonte: Adaptado de Živić, Nemec e Bojović (2023) e Al-Masri *et al.* (2020).

Essa análise é crucial, pois orienta as decisões sobre a infraestrutura e os servidores necessários para a integração do sistema.

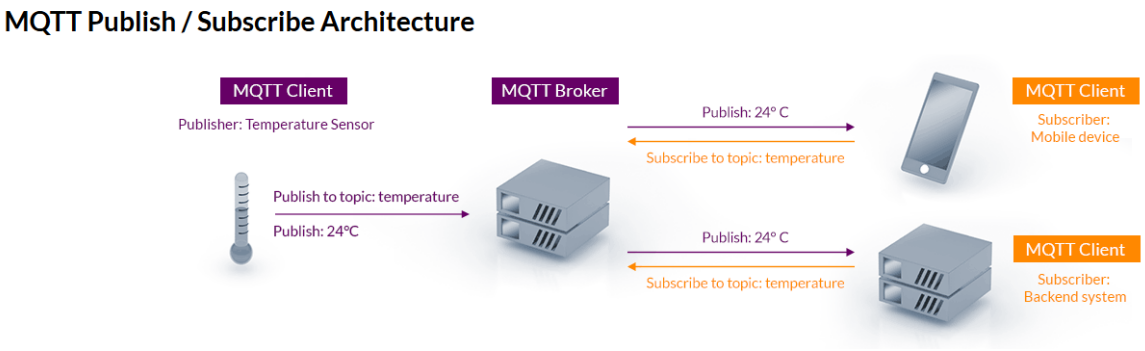
2.1.2.3 Arquiteturas de Comunicação

A arquitetura de comunicação em IoT define como os dispositivos interagem e trocam informações dentro do sistema, sendo um componente crítico para garantir eficiência, escalabilidade e confiabilidade. As arquiteturas mais comuns incluem:

- **Arquitetura Cliente-Servidor:** Utilizada em protocolos como *Hypertext Transfer Protocol* (HTTP) e CoAP, onde dispositivos clientes fazem solicitações diretas a servidores. Essa abordagem é simples e bem estabelecida, mas pode enfrentar limitações em cenários com alta densidade de dispositivos ou requisitos de tempo real.
- ***Publish/Subscribe*:** A arquitetura *Publish/Subscribe* pode ser observada na Figura 6, o *broker* atua como um intermediário central, desacoplando os dispositivos publicadores e assinantes. Esse modelo permite que os dispositivos publiquem mensagens em tópicos específicos, que são, então, entregues aos assinantes interessados, sem que haja necessidade

de uma conexão direta entre eles. Essa abordagem oferece escalabilidade, modularidade e eficiência em sistemas que lidam com um grande volume de dispositivos e fluxos de dados (ANSYAH; VIERI *et al.*, 2023).

Figura 6 – Arquitetura MQTT *Publish/Subscribe*



Fonte: MQTT Organization (2024)

2.1.3 Agente de Mensagens

O agente de mensagens desempenha um papel fundamental na comunicação entre dispositivos IoT e sistemas de processamento, mediando a troca de informações de maneira eficiente e confiável. De maneira geral, trata-se de um software que facilita a comunicação e o intercâmbio de informações entre aplicações, sistemas e serviços, traduzindo mensagens entre diferentes protocolos de comunicação. Essa funcionalidade permite que serviços interdependentes se comuniquem diretamente, mesmo quando desenvolvidos em linguagens distintas ou implementados em plataformas diversas (IBM, 2024).

Neste documento, os termos *broker MQTT* e *servidor MQTT* serão considerados equivalentes.

2.2 ENGENHARIA DE DADOS

A Engenharia de Dados é uma área fundamental para viabilizar a coleta, o processamento e a análise de grandes volumes de dados em sistemas modernos, especialmente em ambientes impulsionados pela IoT. Com o aumento da quantidade de dispositivos conectados, a demanda por soluções que extraiam valor desses dados cresceu exponencialmente. Nesse contexto, a Engenharia de Dados concentra-se em criar pipelines que integram, processam e disponibilizam dados de forma escalável e eficiente (AL-FUQAHA; GUIZANI *et al.*, 2015).

2.2.1 Conceitos Fundamentais

A Engenharia de Dados abrange o ciclo de vida completo dos dados, desde sua geração até seu consumo. Segundo Lin, Yu *et al.* (2017), essa disciplina requer competências em armazenamento, ingestão, transformação e disponibilização de dados, envolvendo o uso de arquiteturas modernas, como *data lakes* e *data warehouses*, além de tecnologias de *processamento em tempo real*. Essa infraestrutura serve como base para cientistas de dados, analistas e sistemas de aprendizado de máquina, permitindo operações eficientes e decisões estratégicas fundamentadas. Conforme Reis e Housley (2022), a Engenharia de Dados transcende a seleção de ferramentas, incorporando princípios sólidos e boas práticas que permanecem relevantes mesmo com a evolução tecnológica.

2.2.2 Coleta, Ingestão e Processamento de Dados

O processamento de dados é uma etapa central no ciclo de vida da informação, responsável por transformar dados brutos em informações estruturadas e úteis. A cadeia de processamento geralmente começa com a coleta e ingestão de dados provenientes de sensores, *logs* ou *Application Programming Interface* (API)s, realizada por ferramentas como *Apache Kafka* (RED HAT, 2024). Esses dados são transportados para sistemas de processamento, onde passam por uma série de etapas, incluindo limpeza, transformação e armazenamento temporário, antes de serem analisados.

A transformação dos dados envolve a padronização, o enriquecimento e a remoção de inconsistências, preparando as informações para análises ou visualizações avançadas. *Frameworks* como *Apache Spark* e *Databricks* são amplamente utilizados nessa etapa, permitindo o processamento de grandes volumes de dados de maneira eficiente (DATABRICKS, 2024a). Para atender a diferentes demandas, o processamento pode ser realizado em tempo real, com análises contínuas, ou em lote, com processamento periódico de grandes quantidades de dados. Ambas as abordagens são complementares e atendem a diferentes necessidades operacionais.

Soluções como AWS, GCP e *Microsoft Azure* (Azure) oferecem plataformas integradas que suportam tanto o *processamento em tempo real* quanto em lote. Esses serviços proporcionam escalabilidade, resiliência e integração com ferramentas analíticas avançadas, reduzindo a complexidade operacional (AMAZON WEB SERVICES, 2024).

2.2.3 Armazenamento: Bancos de Dados Relacionais

Uma base de dados é um conjunto organizado de informações projetado para ser facilmente acessado, gerenciado e atualizado. Bancos de dados estruturados, ou relacionais, armazenam informações em tabelas organizadas com linhas e colunas, permitindo a criação de relações entre elas. Esse modelo, fundamentado no trabalho de Edgar F. Codd, facilita a consulta e manipulação de dados por meio de *Structured Query Language* (SQL) (ORACLE, 2024).

Os bancos de dados relacionais garantem consistência e integridade ao impor regras de integridade referencial e suportar transações seguras baseadas em propriedades *Atomicity, Consistency, Isolation, Durability* (ACID). Além disso, ferramentas como SQL oferecem extensibilidade por meio de recursos como tipos de dados personalizados, sendo amplamente utilizadas em sistemas de IoT (POSTGRESQL, 2024).

Entre as soluções amplamente adotadas no mercado, o SQL se destaca por sua robustez e escalabilidade, sendo amplamente utilizado em sistemas corporativos críticos. SQL é uma escolha comum devido à sua facilidade de uso e integração com ferramentas *Microsoft*, enquanto SQL, uma solução de código aberto, oferece recursos avançados e flexibilidade para atender às demandas de projetos modernos (MICROSOFT, 2024; POSTGRESQL, 2024).

2.3 DESIGN DE SISTEMAS DE DADOS

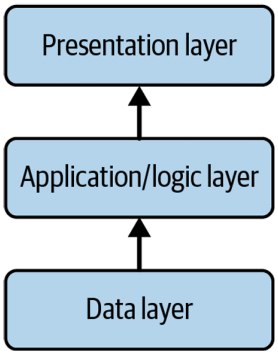
O design de sistemas é um aspecto crucial no desenvolvimento de soluções tecnológicas, abrangendo desde a concepção inicial até a implementação e manutenção. Este tema encontra-se detalhadamente elucidado no livro *Fundamentals of Data Engineering* (REIS; HOUSLEY, 2022), que discute os princípios fundamentais e as melhores práticas para projetar sistemas escaláveis e eficientes. Essa abordagem garante que os sistemas sejam projetados de maneira eficiente, escalável e alinhada aos requisitos operacionais e de negócio.

2.3.1 Abordagem Multitier e Medallion Architecture

A abordagem multitier organiza sistemas em camadas independentes, nas quais cada uma desempenha uma função específica no processamento e gerenciamento de dados. Essa estrutura modular não apenas facilita a manutenção e promove a reutilização de componentes, mas também oferece escalabilidade ao sistema de forma eficaz, alinhando-se aos objetivos operacionais e de negócio (REIS; HOUSLEY, 2022). A Figura 7 demonstra como a divisão

em tiers ou camadas modulariza o sistema, permitindo otimizações e ajustes independentes em cada estágio do fluxo de dados, seja na camada de armazenamento e gerenciamento de dados, na camada de aplicação e processamento, ou na interface com o usuário.

Figura 7 – Arquitetura Multitier



Fonte: Reis e Housley (2022)

No contexto de gerenciamento de dados, um exemplo aplicado é a *Medallion Architecture*, que organiza o fluxo de dados em camadas progressivas de refinamento e processamento (*bronze, prata e ouro*). Essa divisão estruturada permite escalabilidade, flexibilidade e rastreabilidade em pipelines de dados, tornando-a ideal para sistemas com fluxos constantes de informações, como no caso de IoT (DATABRICKS, 2024b).

2.3.1.1 Camadas da Arquitetura Medallion

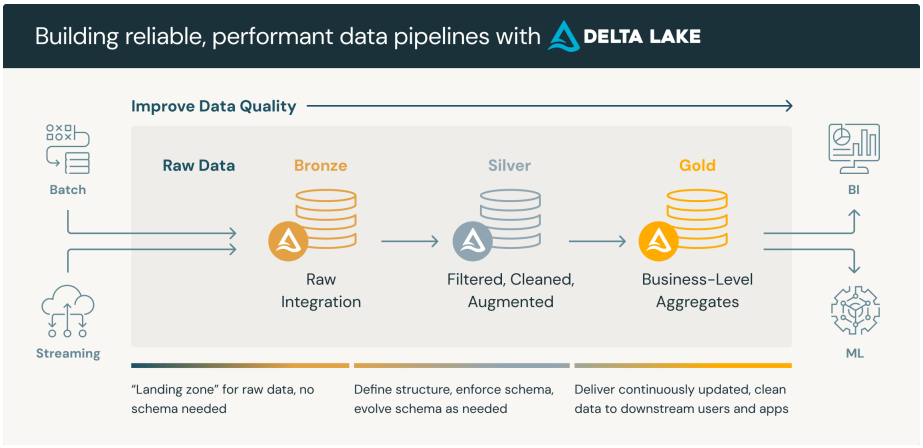
A *Medallion Architecture*, como uma extensão da abordagem multitier, divide os dados em três camadas principais:

Camada Bronze: Dados brutos são armazenados em seu estado original, sem transformações. Essa camada funciona como um repositório histórico, garantindo a rastreabilidade e preservando os registros para auditorias futuras. Formatos comuns incluem *JavaScript Object Notation* (JSON), *Comma-Separated Values* (CSV) ou *logs*, frequentemente contendo redundâncias e inconsistências, mas mantendo a integridade dos dados originais.

Camada Prata: Dados da camada bronze passam por uma etapa inicial de limpeza e padronização, abordando valores ausentes, duplicatas e inconsistências. Essa camada é otimizada para análises exploratórias e interações iniciais com ferramentas analíticas ou modelos de aprendizado de máquina.

Camada Ouro: Dados refinados são preparados para análises avançadas e consumo direto, como *dashboards*, APIs ou relatórios gerenciais. Essa camada prioriza a precisão e a utilidade, sendo crucial para aplicações críticas e decisões baseadas em dados.

Figura 8 – Esquema da Arquitetura em Camadas *(Medallion Architecture)*.



Fonte: Databricks (2024b).

A divisão modular entre essas camadas facilita a evolução progressiva da qualidade dos dados, permitindo que as camadas sejam ajustadas conforme as necessidades específicas do sistema ou das análises.

Vantagens da *Medallion Architecture*

A adoção da *Medallion Architecture* oferece vantagens significativas no gerenciamento de dados. Sua escalabilidade permite que cada camada do *pipeline* seja ajustada de forma independente, otimizando tanto o armazenamento quanto o processamento. A camada bronze garante rastreabilidade e auditoria, preservando os dados originais para investigações ou recuperações futuras. Além disso, a modularidade da arquitetura facilita ajustes e integrações sem impactar o sistema como um todo. A qualidade dos dados melhora progressivamente entre as camadas, resultando em informações confiáveis e prontas para consumo. Por fim, a otimização de recursos reduz custos computacionais ao processar dados apenas quando necessário, tornando a arquitetura eficiente e robusta.

2.3.2 Implementação com Microserviços e Kubernetes

A arquitetura multitier em sistemas de nuvem pode ser implementada de forma eficaz utilizando microsserviços em conjunto com ferramentas de orquestração como o Kubernetes

(REIS; HOUSLEY, 2022).

Os microsserviços são componentes autônomos e especializados que desempenham funções específicas dentro do sistema (KUBERNETES, 2024). Cada camada da arquitetura multitier pode ser representada por um conjunto de microsserviços que operam de maneira independente, mas se comunicam por meio de APIs. Essa abordagem aumenta a modularidade do sistema, permitindo que cada microsserviço seja desenvolvido, implantado e escalado separadamente, de acordo com as demandas específicas de cada camada.

A orquestração com Kubernetes desempenha um papel crucial no gerenciamento desses microsserviços em ambientes baseados em contêineres. Kubernetes automatiza tarefas como escalabilidade horizontal, balanceamento de carga e recuperação de falhas, assegurando alta disponibilidade e eficiência operacional. Além disso, permite a implementação de políticas avançadas de gerenciamento de recursos, otimizando o desempenho do sistema em larga escala.

Por exemplo, na camada de ingestão, microsserviços podem ser configurados para coletar dados de dispositivos IoT e transmiti-los para sistemas de processamento em tempo real. Na camada de processamento, outros microsserviços realizam transformações e análises nos dados, enquanto a camada de armazenamento utiliza serviços especializados para persistir as informações de maneira segura e escalável.

Essa integração entre microsserviços e Kubernetes possibilita a construção de sistemas altamente flexíveis, escaláveis e resilientes, capazes de se adaptar rapidamente às mudanças nas demandas de negócios e nos avanços tecnológicos.

2.3.3 Vantagens do Design Modular

- **Escalabilidade**: Sistemas modulares permitem a escalabilidade direcionada, adicionando recursos a camadas específicas para lidar com maiores volumes de dados. Em ambientes de nuvem, essa escalabilidade é dinâmica e frequentemente gerenciada automaticamente pelos provedores.

- **Facilidade de Manutenção**: A separação em camadas facilita a identificação e resolução de problemas, além de permitir atualizações localizadas sem impactar o sistema como um todo.

- **Customização**: O design modular possibilita ajustar cada componente às necessidades específicas do projeto, promovendo maior eficiência e alinhamento com os objetivos do

sistema. Serviços em nuvem frequentemente oferecem APIs e ferramentas para personalização detalhada.

- **Redução de Custos**: A modularidade evita a substituição completa de sistemas, reduzindo custos de atualização e operação a longo prazo. Modelos de cobrança por uso, oferecidos pelos provedores de nuvem, otimizam os custos operacionais.
- **Adaptação Tecnológica**: Com a rápida evolução tecnológica, sistemas modulares podem ser atualizados de forma incremental, integrando novas ferramentas e técnicas sem interrupções significativas. A utilização de orquestradores como Kubernetes facilita essa adaptação em ambientes baseados na nuvem.

Essa abordagem garante que o sistema seja capaz de atender às demandas atuais e futuras, oferecendo flexibilidade suficiente para enfrentar novos desafios e aproveitar oportunidades, enquanto explora os recursos avançados dos serviços em nuvem.

3 METODOLOGIA

O capítulo atual detalha a metodologia empregada para a execução deste projeto. A metodologia utilizada no desenvolvimento do sistema pode ser segmentada nas seguintes etapas:

1. **Definição dos Requisitos:** Identificação e análise dos requisitos de negócios e técnicos do sistema.

2. **Seleção de Ferramentas e Tecnologias:** Determinação das ferramentas e tecnologias que melhor atendem aos requisitos estabelecidos.

3. **Fase de desenvolvimento:** Configuração e implementação da infraestrutura necessária para a operação do sistema.

4. **Implementação dos Serviços:** Criação dos serviços responsáveis pela aquisição, processamento e armazenamento de dados.

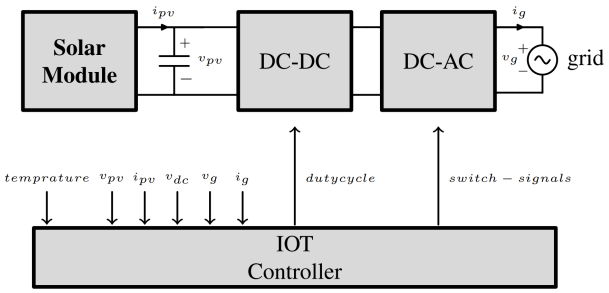
5. **Testes e Validação:** Execução de testes para assegurar a funcionalidade e o desempenho do sistema.

3.1 CONTEXTUALIZAÇÃO

Para o desenvolvimento deste trabalho, o caso escolhido é um sistema Fotovoltaicos (PV) simples. Um sistema PV simples inclui painéis solares para captar energia, um inversor ou microinversor para conversão de Corrente Contínua (CC) em Corrente Alternada (CA), e sensores de dados para monitorar parâmetros como irradiância, temperatura e produção de energia, garantindo eficiência e desempenho otimizados. A Figura 9 ilustra o esquema de um microinversor.

Neste projeto, foram utilizados os dados do módulo SunPower `SPR-X22-360-WHT-AC`, que combina células solares de alta eficiência com um microinversor integrado, criando um sistema que otimiza o arranjo fotovoltaico ao nível do módulo. Cada painel opera de forma independente, reduzindo perdas causadas por sombreamento parcial ou falhas no sistema. O microinversor captura dados detalhados, como tensão, corrente, potência gerada e eficiência de conversão, além de monitorar métricas ambientais que impactam o desempenho (SUNPOWER CORPORATION, 2024). Este documento não trata do escopo de eletrônica de potência do

Figura 9 – Esquema de um sistema fotovoltaico simples.



Fonte: SeyyedHosseini, Yazdian-Varjani e Mohamadian (2020)

sistema PV, como aspectos das topologias de conversores CC-CC e CC-CA, mas sim dos dados extraídos destes equipamentos.

3.2 REQUISITOS DE NEGÓCIO

O caso de uso foi baseado nos requisitos estabelecidos pelo *Photovoltaic Data Acquisition* (PVDAQ), visando servir como uma prova de conceito para uma proposta de arquitetura de processamento de dados IoT. Em 2023, o PVDAQ realizou a *Solar Data Prize*, uma iniciativa que incentivou proprietários de sistemas PVs nos Estados Unidos, ou entidades autorizadas, a submeterem pelo menos cinco anos de dados históricos de séries temporais de seus sistemas (DELINE *et al.*, 2021). Esses dados deveriam ser coletados em intervalos de 15 minutos ou menores e deveriam incluir informações como produção de energia, irradiância, temperatura ambiente e temperatura dos módulos.

Entre os objetivos do PVDAQ estão a análise de desempenho de sistemas PVs, o estudo das taxas de degradação de módulos, o *benchmarking* de tecnologias de geração de energia solar e a modelagem de previsões baseadas em dados históricos. Esses objetivos demandam dados parametrizados, tratados e coletados em intervalos regulares de tempo, permitindo análises consistentes e de alta qualidade.

O caso de uso principal foi definido para monitoramento de potência, tensão e fator de potência de cada inversor. A criação de uma base de dados parametrizada abre diversas possibilidades para estudo e análise, incluindo:

- Identificação de padrões operacionais em inversores;
- Detecção precoce de falhas ou desvios de comportamento;

- Análise histórica de desempenho para planejamento de manutenção preventiva;
- Otimização da eficiência energética com base nos fatores de potência e tensão analisados.

Para atender aos requisitos do caso de uso e com base nos dados coletados dos inversores, os dados requeridos estão apresentados na Tabela 3.

Tabela 3 – Dados Requeridos

Dado	**Descrição**
Dados de potência	Informações relacionadas à potência processada pelos inversores.
Dados de fator de potência	Valores do fator de potência calculados a partir dos inversores.
Dados de tensão CA	Dados de tensão CA na saída dos inversores.
Horário da medida	Registro do horário em que a medida foi realizada.
Tensão CC de entrada	Informações sobre a tensão CC de entrada nos inversores.
Temperatura	Medida da temperatura ambiente.

Com base na competição e nas fontes de dados disponíveis, os requisitos de negócio foram definidos para delimitar o escopo da aplicação e garantir que as funcionalidades atendam às necessidades operacionais próximas das reais. Estes requisitos estão detalhados na Tabela 4.

Tabela 4 – Requisitos de Negócio

Requisito	**Descrição**
Frequência de mensagens	O intervalo máximo entre as mensagens não deve ultrapassar 2 minutos.
Garantia de entrega	Todas as mensagens devem ser entregues e processadas pelo menos uma vez.
Armazenamento de dados	A base de dados deve ser capaz de armazenar informações dos inversores de forma estruturada e normalizada.

A frequência de mensagens foi definida em 2 minutos, garantindo uma resolução adequada para captar mudanças instantâneas no comportamento do sistema. Essa escolha facilita a aplicação de métodos estatísticos e de aprendizado de máquina baseados em séries temporais, enquanto evita sobrecarregar o sistema de armazenamento e processamento. Com esse intervalo, o volume de dados gerado seria de aproximadamente 262.980 mensagens por ano. Considerando um tamanho médio de 350 *kilobytes* por mensagem, isso corresponde a cerca de 80 *Gigabytes*

de dados por inversor ao ano. Reduzir o intervalo para menos de 2 minutos resultaria em um aumento exponencial no volume de dados armazenados, o que poderia comprometer a escalabilidade do sistema, tornando os custos de armazenamento e processamento significativamente elevados.

Além disso, os demais requisitos foram definidos para garantir a confiabilidade dos dados, assegurando sua qualidade e integridade para análises posteriores.

3.3 REQUISITOS DO SISTEMA

Com base nos requisitos de negócio, os requisitos do sistema foram definidos para garantir a implementação eficiente da aplicação. O número de sensores foi estimado pelo autor, utilizando como referência a média do volume de sensores em sistemas descritos no *Solar Data Prize 2023* para sistemas similares. Esses requisitos estão detalhados na Tabela 5.

Tabela 5 – Requisitos do Sistema

Requisito	**Descrição**
Conectar múltiplos sensores	O sistema deve ser capaz de conectar, no mínimo, 25 sensores enviando mensagens simultaneamente a cada 2 minutos.
Garantia de entrega	Todas as mensagens devem ser entregues e processadas pelo menos uma vez, garantindo consistência no fluxo de dados.
Armazenamento de metadados	A base de dados deve registrar informações como horário de recebimento, tópico de origem e identificador de cada mensagem.
Normalização de dados	Os dados de inversores devem ser normalizados para facilitar análises e integrações posteriores.

Como mencionado anteriormente, este caso de uso foi projetado como um modelo de prova de conceito. Em uma aplicação real, o sistema pode ser ampliado para atender a novos requisitos de negócio e técnicos, explorando sua escalabilidade e adaptabilidade para diferentes cenários. Esse design modular permite ajustes e melhorias conforme as necessidades operacionais evoluem.

3.4 SOBRE OS DADOS A UTILIZADOS

Para simular um sistema real, utiliza-se a infraestrutura do *Maui Ocean Center* como emulação de um sistema fotovoltaico real. Este sistema fotovoltaico de 110 kW, localizado em

Maui, Havaí, opera desde 01 de fevereiro de 2016 e participa do *DOE Solar Data Prize 2023*, garantindo a qualidade e confiabilidade dos dados coletados (DELINE *et al.*, 2021).

Os dados incluem várias tabelas com informações dos onze inversores, sensores de ambiente para vento e irradiância, e informações técnicas de placas e inversores. Como os dados são objeto de teste, valores técnicos desses equipamentos não foram considerados; o foco foi a integridade e veracidade dos dados. Uma análise mais profunda sobre o comportamento de painéis solares e inversores está fora do escopo deste trabalho.

3.4.1 Dados do escopo do Caso de Uso

No escopo deste trabalho será utilizada somente a tabela dos inversores. Cada linha da tabela será utilizada como um *payload* de mensagem que representa uma amostragem do sinal medido pelo conjunto de sensores presentes no equipamento. Na Tabela 6 se encontra um modelo de *payload* referente às mensagens advindas do inversor 1.

Tabela 6 – Dicionário de dados do sistema

Campo	**Descrição**	**Tipo do Dado**
measured_on	Data e hora da medição	Timestamp
inv_string01_ac_voltage_(v)_inv_150163	Tensão CA do inversor 01	Float
inv_string01_dc_voltage_(v)_inv_150162	Tensão CC do inversor 01	Float
inv_string01_ac_output_(kwh)_inv_150164	Saída CA em kWh do inversor 01	Float
inv_string01_temperature_(c)_inv_150166	Temperatura do inversor 01 (°C)	Float
inv_string01_ac_output_(power_factor)_inv_150165	Fator de potência da saída CA do inversor 01	Float

Fonte: Autor.

Tendo em vista o Caso de Uso do projeto, somente algumas das colunas serão necessárias na camada do usuário, que são referentes a algumas das informações dos sensores. Os dados finais necessários estão presentes na Tabela 7.

Tabela 7 – Dados esperados na tabela parametrizada

Campo	**Descrição**	**Tipo do Dado**
measured_on	Data e hora da medição	Timestamp
ac_output_kwh	Saída CA em kWh do inversor	Float
ac_output_power_factor	Fator de potência da saída CA do inversor	Float
ac_voltage_v	Tensão CA do inversor	Float
dc_voltage_v	Tensão CC do inversor	Float
temperature_c	Temperatura do inversor (°C)	Float
received_at	Horário de recebimento da mensagem	Timestamp
topic	Tópico do qual a mensagem foi recebida	String

Fonte: Autor.

3.5 SELEÇÃO DE FERRAMENTAS E TECNOLOGIAS

Neste trabalho, optou-se por não utilizar soluções nativas dos provedores de nuvem, desenvolvendo os serviços de ingestão e processamento de forma independente. Essa decisão foi motivada pela busca por maior flexibilidade e controle sobre os dados, além da redução de custos em longo prazo. O desenvolvimento independente permite adaptar os serviços a diferentes contextos, sejam eles na nuvem (escopo do projeto) ou em sistemas locais de IoT. Além disso, ao evitar soluções proprietárias, mitigam-se dependências de plataformas específicas, proporcionando maior autonomia e resiliência às mudanças tecnológicas.

Soluções proprietárias podem ser vantajosas em projetos com prazos curtos ou recursos limitados, pois oferecem implementação rápida e suporte técnico integrado. Além disso, em contextos onde segurança e conformidade são essenciais, como em setores regulados, essas opções já incluem certificações necessárias, reduzindo a complexidade. Outra vantagem ocorre quando o sistema já está amplamente integrado ao ecossistema do provedor de nuvem, permitindo uma maior eficiência operacional e simplificando a gestão ao centralizar serviços em uma única plataforma.

3.5.1 Tecnologias

A seleção das tecnologias para este projeto considerou o contexto do caso de uso e os requisitos definidos. Após análise, optou-se pelo protocolo MQTT devido à sua simplicidade, eficiência e suporte a monitoramento em tempo real. Conforme levantamento de 2023 pelo Eclipse Foundation (2023), o MQTT é a principal tecnologia utilizada em *brokers* de mensagens, reforçando sua adequação para aplicações IoT.

A arquitetura *Publish/Subscribe* foi escolhida para garantir escalabilidade, integração eficiente entre os *gateways* de dados e o sistema de processamento, além de desacoplar dispositivos, facilitando o gerenciamento de fluxos de dados em tempo real. Essa abordagem também atende aos requisitos técnicos, promovendo modularidade e flexibilidade.

O *Eclipse Mosquitto* foi selecionado como *broker MQTT* por sua robustez, leveza e compatibilidade com o padrão MQTT. Amplamente utilizado em aplicações IoT, ele oferece suporte completo ao protocolo, garantindo comunicação eficiente mesmo em dispositivos com recursos limitados. Sua flexibilidade permite integração com sistemas maiores utilizando ferramentas como *Docker* e *Kubernetes* (ECLIPSE FOUNDATION, 2024).

3.5.2 Ferramentas

A escolha das ferramentas foi guiada por critérios como eficiência, escalabilidade, custo-benefício e facilidade de uso. A seguir, apresentam-se as ferramentas selecionadas e suas justificativas.

Python

Python foi utilizado para o desenvolvimento dos *scripts* responsáveis pelo processamento das mensagens e interação com o banco de dados. Destaca-se por sua simplicidade, curva de aprendizado rápida e disponibilidade de bibliotecas como `paho-mqtt` para comunicação MQTT e `psycopg2` para integração com PostgreSQL.

Docker

O Docker assegura consistência nos ambientes de desenvolvimento, teste e produção, criando contêineres replicáveis que agilizam o processo de integração e entrega contínua (CI/CD) (DOCKER, 2024).

Kubernetes

O Kubernetes orquestra os contêineres em ambientes de nuvem, garantindo escalabilidade e alta disponibilidade. Sua integração com provedores como o *Google Cloud Platform* facilita o gerenciamento de recursos e serviços (KUBERNETES, 2024).

Google Cloud Platform (GCP)

A *Google Cloud Platform* foi escolhida pelo suporte técnico eficiente, documentação detalhada e bônus inicial de 300 dólares para testes. Além disso, sua flexibilidade permite alternativas, como o uso de outros provedores ou servidores pessoais. A desativação do *IoT Core* em agosto de 2023 impulsionou a necessidade de desenvolver soluções personalizadas. Esta ferramenta era a interface entre a GCP e todas as máquinas e dispositivos na IoT. As conexões são configuradas via o protocolo de mensagens MQTT e um corretor MQTT. Dispositivos finais e gateways podem então se conectar aos serviços em nuvem do *Google* através do *Google IoT Core*.

GitHub Container Registry

O *GitHub Container Registry* (GHCR) foi utilizado para gerenciar imagens Docker. Sua integração com repositórios GitHub facilita a automação de *builds* e *deploys*, garantindo segurança (GITHUB, INC., 2024).

PostgreSQL

O *PostgreSQL* foi escolhido para armazenar e gerenciar os dados coletados devido à sua robustez, conformidade com padrões SQL e extensibilidade, ideal para sistemas de alta performance (POSTGRESQL, 2024).

3.6 FASE DE DESENVOLVIMENTO

O desenvolvimento deste projeto foi orientado pela filosofia Ágil (BECK *et al.*, 2001), que enfatiza a flexibilidade e a entrega incremental de funcionalidades. A adoção de pipelines de *Continuous Integration/Continuous Deployment* (CI/CD) possibilitou a identificação e correção rápida de erros, além de permitir adaptações rápidas às mudanças de requisitos ao longo do ciclo de vida do projeto.

O trabalho foi realizado de forma independente, seguindo uma sequência lógica para a implementação das principais funcionalidades. Inicialmente, foi selecionado o agente de mensagens mais adequado para a comunicação via MQTT entre os dispositivos IoT e os demais serviços. Este agente foi testado em um ambiente na nuvem para validar sua compatibilidade e desempenho. Na sequência, foi desenvolvido o serviço responsável pelo armazenamento dos dados na base de dados, com foco na robustez, eficiência e na garantia de integridade e disponibilidade das informações.

Em uma etapa posterior, foi implementado o sistema de normalização dos dados, projetado para transformar os dados brutos em um formato estruturado e padronizado. Essa funcionalidade facilita análises futuras e assegura a consistência das informações processadas. Cada nova funcionalidade foi incorporada ao sistema por meio de um rigoroso processo de testes e validações, garantindo a qualidade e a estabilidade do sistema como um todo.

O desenvolvimento de cada funcionalidade segue uma sequência estruturada de etapas de teste. Inicialmente, a funcionalidade é testada em um ambiente local, permitindo a identificação e correção de erros em um estágio preliminar. Após a validação local, ela é containerizada

utilizando *Docker* e testada em um contêiner isolado, garantindo seu funcionamento adequado nesse ambiente. Em seguida, a funcionalidade é integrada aos demais serviços e testada com *Docker Compose*, simulando um ambiente multi-contêiner e validando a operação do sistema como um todo. Por fim, a funcionalidade é implantada em um *cluster* Kubernetes na nuvem, onde sua operação é verificada em um ambiente escalável e semelhante ao ambiente de produção. Este ciclo de desenvolvimento foi aplicado para todas as funcionalidades desenvolvidas.

Para apoiar o desenvolvimento e a implantação, foi utilizado um pipeline de CI/CD, que automatizou tarefas como *build* e *deploy* para validação das aplicações. Embora seja uma ferramenta poderosa para o fluxo de trabalho, a configuração detalhada da mesma não é o foco deste trabalho e será tratada de forma superficial. Sua principal função foi garantir a integração contínua e a entrega eficiente das funcionalidades implementadas possibilitando ajustes e alterações em alta frequência com baixo esforço.

Nas próximas seções, serão apresentados mais detalhes sobre cada etapa de desenvolvimento e testes.

3.7 DESIGN DA ARQUITETURA

Partindo para o design da arquitetura e considerando os requisitos estabelecidos, foi identificada como abordagem eficiente a adaptação de uma arquitetura tradicional de IoT, em uma aplicação multitier inspirada na filosofia da arquitetura *Medallion*. Essa estratégia mostrou-se ideal para proporcionar maior robustez no fluxo de dados e flexibilidade para atender a novos casos de uso. A separação dos dados em brutos e parametrizados permite atender demandas específicas de forma ágil, possibilitando que, caso o usuário necessite de uma normalização diferente, os dados brutos estejam facilmente acessíveis para processamento.

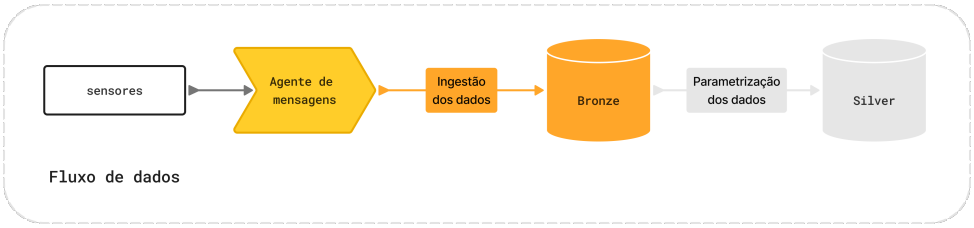
Em paralelo, para uma visão geral, este projeto foi inspirado em uma série de documentos disponibilizados pelo GCP, que apresentam propostas para conectar dispositivos à nuvem utilizando agentes MQTT independentes no contexto de IoT dentro do ambiente do GCP (GOOGLE CLOUD, 2024).

A arquitetura proposta é baseada na implantação de uma série de serviços independentes, sendo eles um *Broker* de mensagens MQTT, complementado por serviços de ingestão de dados que processam as mensagens e as armazenam de forma bruto. Um serviço de tratamento de dados que processa os dados brutos para formar uma tabela normalizada. Essa abordagem

mantém uma única fonte de verdade, a tabela de dados brutos, garantindo maior conformidade e integridade dos dados.

Uma representação inicial do sistema pode ser visualizada na Figura 10, que apresenta um diagrama simplificado da arquitetura. A figura ilustra as interações principais entre os componentes, incluindo o agente de mensagens MQTT, os serviços de ingestão de dados e limpeza dos dados e as camadas de armazenamento.

Figura 10 – Arquitetura Simples



Fonte: Autor

3.7.1 Agente de Mensagens

A primeira etapa é o processamento das mensagens, conduzido pelo Agente de Mensagens, que garante que todas as mensagens publicadas no tópico sejam entregues aos clientes inscritos. Uma característica importante desta etapa é a estrutura dos tópicos utilizados para a transmissão dos dados, que segue o padrão:

- `solar-data/TIPO-DO-DADO/ID`, onde:
  - `TIPO-DO-DADO`: Representa a categoria dos dados enviados, como `inverter`, `weather` ou `meter`.
  - `ID`: Identifica o sensor específico responsável pelo envio da mensagem.
- **Exemplo**: `solar-data/inverter/5` para dados provenientes do sensor do inversor 5.

Além disso, o sistema utiliza o nível de qualidade de serviço (**QoS 1**) do protocolo MQTT, garantindo que todas as mensagens sejam entregues e processadas pelo menos uma vez, assegurando a integridade e completude dos dados recebidos.

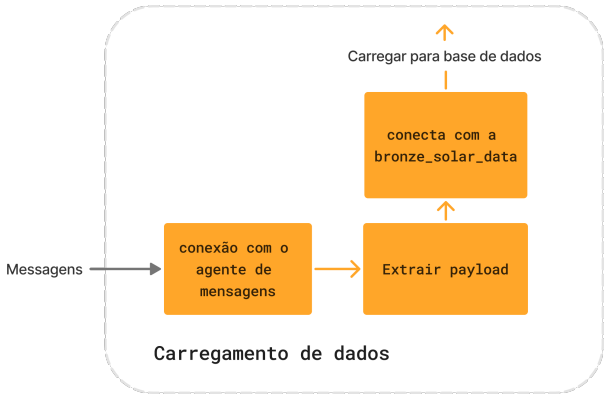
O Agente de Mensagens atua como o ponto de entrada para o sistema, garantindo que os dados recebidos sejam disponibilizados para as etapas subsequentes de ingestão e análise.

3.7.2 Ingestão de Dados

A etapa de Ingestão de Dados é responsável por executar as ações de recebimento, processamento e armazenamento das mensagens provenientes do Agente de Mensagens. Esse serviço conecta-se ao agente e inscreve-se nos tópicos configurados, capturando cada mensagem assim que ela é publicada. Durante essa etapa, os dados brutos são organizados em dois componentes principais: os metadados da mensagem, como o horário de recebimento e o nome do tópico de origem, e o *payload*, que contém as informações transmitidas pelos sensores. Após o processamento inicial, as mensagens são armazenadas na camada *Bronze*, preservando-as em seu estado original para garantir rastreabilidade e flexibilidade.

A Figura 11 apresenta um diagrama que ilustra a interação entre o Agente de Mensagens, a etapa de Ingestão de Dados e o armazenamento na camada *Bronze*.

Figura 11 – Estrutura e Fluxo de Dados até a Camada *Bronze*



Fonte: Autor.

3.7.3 Camada *Bronze*

A camada *Bronze* é o armazenamento dedicado aos dados brutos capturados e processados na etapa de Ingestão de Dados. Ela consiste em uma tabela no banco de dados onde as mensagens são registradas em seu estado original. O *payload* é armazenado em formato JSON, permitindo lidar de forma flexível e eficiente com diferentes formatos de dados enviados pelos sensores.

Por ser um armazenamento estático, a camada *Bronze* não realiza ações sobre os dados, mas garante que estejam disponíveis para transformações futuras na camada *Silver*. Manter os

dados em seu estado bruto proporciona rastreabilidade e flexibilidade, permitindo atender a novos casos de uso que possam surgir.

3.7.4 Parametrização dos Dados

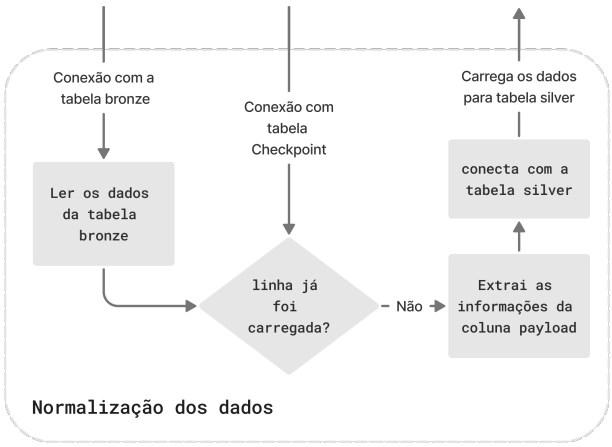
A etapa de Parametrização dos Dados é responsável pelo processamento das informações brutas armazenadas na camada *Bronze*, transformando-as em dados estruturados e prontos para futuras análises. Nesta etapa, a pipeline extrai informações essenciais, padroniza formatos e organiza os dados, preparando-os para processamento mais avançado na camada *Gold*.

Para garantir eficiência e evitar o retrabalho, é implementado um sistema de *checkpoint* que registra o *timestamp* da última mensagem processada. Isso assegura que apenas os dados novos sejam transformados em cada execução, otimizando o uso de recursos e reduzindo o tempo de processamento.

Durante o processo de parametrização, os dados originalmente armazenados em formato JSON são analisados e mapeados para colunas específicas, como potência de saída, fator de potência, temperatura e tensão. Além disso, metadados como o horário de recebimento e o tópico de origem da mensagem são preservados, fornecendo contexto adicional para análises futuras. Um exemplo dessa etapa é a padronização de diferentes formatos de horário para um único formato *Universal Time Coordinated* (UTC), facilitando análises temporais consistentes.

A Figura 12 apresenta um diagrama que ilustra o fluxo de dados e o processamento realizado na etapa de Parametrização dos Dados, destacando como os dados são transformados antes de serem armazenados na camada *Silver*.

Figura 12 – Processamento de Dados entre as Camadas *Bronze* e *Silver*



Fonte: Autor.

3.7.5 Camada Silver

A camada *Silver* é o armazenamento dedicado aos dados transformados e estruturados na etapa de Parametrização dos Dados. Após o processamento, os dados são organizados em tabelas que seguem uma estrutura definida, pronta para ser utilizada em análises avançadas na camada *Gold*.

Na camada *Silver*, cada dado mapeado durante o processamento é armazenado de forma padronizada, com colunas específicas para métricas como potência de saída, fator de potência, temperatura e tensão. Essa estruturação facilita a manipulação e a análise dos dados, eliminando inconsistências que poderiam surgir a partir de formatos brutos ou não padronizados.

Além disso, os metadados, como o horário de recebimento e o tópico de origem da mensagem, continuam a ser armazenados, garantindo o contexto necessário para futuras análises.

3.8 TESTES

A fase de Testes é essencial para garantir que o sistema funcione conforme as especificações projetadas. Esta seção aborda os métodos utilizados para verificar a funcionalidade, escalabilidade e robustez do sistema.

3.8.1 Testes Unitários

Os testes unitários foram realizados manualmente, com o objetivo de validar a correção de funções e métodos específicos. Essa abordagem permite identificar e corrigir falhas isoladas, garantindo que cada componente funcione adequadamente antes da integração total do sistema. Além disso, os testes asseguram a confiabilidade e precisão das funcionalidades implementadas, reduzindo a ocorrência de erros em estágios posteriores.

3.8.2 Testes de Carga com Locust

Para avaliar a escalabilidade e robustez do sistema, foram conduzidos testes de carga utilizando a ferramenta *Locust*, amplamente empregada para simular conexões simultâneas e medir o desempenho de sistemas (LOCUST TEAM, 2024). Essa ferramenta opera criando múltiplas instâncias de usuários simulados, permitindo configurar parâmetros como o número inicial de usuários, a taxa de crescimento de usuários ao longo do tempo e os intervalos de envio de mensagens pelos usuários.

Durante a execução, o *Locust* coleta métricas de desempenho em tempo real, incluindo tempo de resposta do sistema, taxas de transferência de mensagens e a integridade das mensagens entregues. Além disso, permite monitorar falhas de comunicação e avaliar a capacidade do sistema em lidar com cargas crescentes de forma eficiente.

Esses testes são fundamentais para identificar possíveis gargalos, validar a escalabilidade do sistema e assegurar sua robustez em cenários de alta demanda.

3.8.3 Teste dos Dados

Adicionalmente, os testes foram configurados para que todos os usuários enviem os mesmos dados. Dessa forma, ao término do teste, é possível validar a integridade dos dados utilizando uma metodologia baseada na comparação entre o número de mensagens enviadas e recebidas. Essa validação envolve a contagem de mensagens recebidas na tabela final e a comparação com o total de mensagens enviadas durante o teste, garantindo que nenhuma mensagem tenha sido perdida. Além disso, verificações de consistência nos valores dos dados armazenados são realizadas para assegurar que os conteúdos das mensagens não foram corrompidos ao longo do fluxo de processamento.

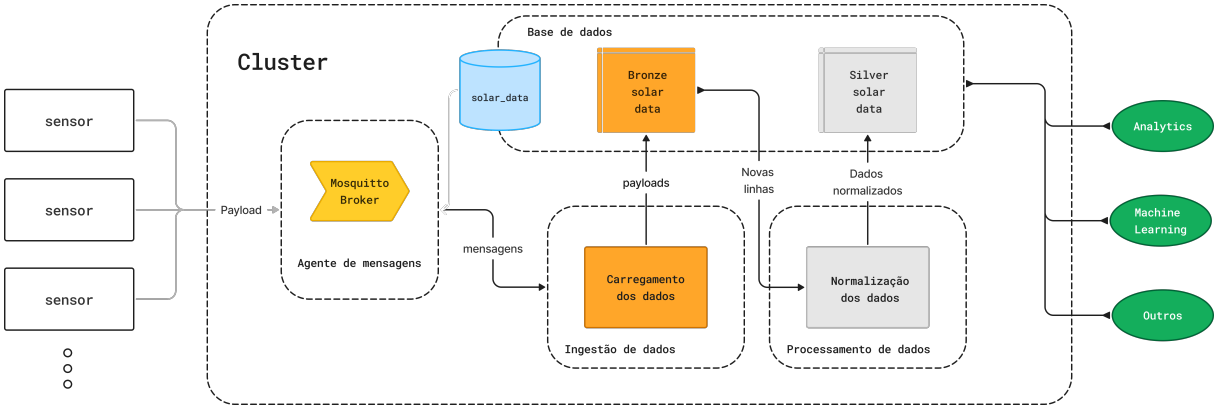
4 IMPLEMENTAÇÃO E RESULTADOS

Neste capítulo, apresentamos os resultados obtidos por meio da aplicação da metodologia descrita no Capítulo 3. São detalhadas a arquitetura final implementada, os testes realizados, e as análises de desempenho, com ênfase na confiabilidade e robustez do sistema.

4.1 A ARQUITETURA FINAL

Com base no design descrito na Seção 3.7, desenvolvemos a arquitetura final do sistema, apresentada na Figura 13. Essa arquitetura foi projetada para garantir a eficiência na ingestão, processamento e armazenamento dos dados provenientes dos sensores IoT, com foco em escalabilidade, modularidade e flexibilidade.

Figura 13 – Arquitetura Final



Fonte: Autor

Com base na arquitetura final o sistema é composto por quatro serviços, descritos na Tabela 8.

Nesta configuração, os sensores conectados aos inversores enviam dados para o *Broker MQTT*, que atua como um intermediário garantindo a entrega confiável das mensagens. Na Camada Bronze, um serviço dedicado se conecta ao *broker* e armazena as mensagens na base de dados `solar_data`, especificamente na tabela `bronze_solar_data`. Este armazenamento inicial preserva os dados brutos em seu formato original, garantindo rastreabilidade e flexibilidade para transformações futuras.

A cada minuto, a Camada *Silver* é ativada automaticamente. O serviço de normalização verifica a presença de novas entradas na tabela Bronze. Caso existam novos dados, estes são

Tabela 8 – Resumo das etapas da arquitetura final

Serviço	**Funcionalidade**
Broker MQTT	Serviço do agente de mensagens MQTT responsável por conectar quem envia e recebe as mensagens como explicado na figura 6.
Ingestão de dados	Serviço responsável por se conectar ao *Broker MQTT* e salvar as mensagens na base de dados
Processamento de dados	Serviço responsável por processar e normalizar os dados
Base de dados PostgreSQL	Ser o ambiente de armazenamento de dados onde os serviços vão se conectar para ler e escrever tabelas de dados.

Fonte: Autor.

processados, estruturados e adicionados a tabelas *Silver* específicas, organizadas por tipo de dado e identificador (por exemplo, `silver-inverter-01`). Esse design modular facilita a utilização dos dados processados por sistemas ou usuários que necessitem de análises específicas.

Essa separação entre as Camadas Bronze e *Silver* promove uma organização eficiente dos dados, permitindo que os dados brutos sejam preservados em sua forma original enquanto os dados processados ficam disponíveis para análises específicas. Esse modelo de arquitetura não apenas garante a integridade dos dados como também facilita a manutenção, escalabilidade e integração com novos serviços no futuro.

4.2 IMPLEMENTAÇÃO TÉCNICA

Nesta seção, apresentamos a implementação técnica do sistema, detalhando a criação do *cluster* Kubernetes no GCP, a configuração das camadas de dados e os serviços de processamento entre elas. Também abordamos o monitoramento e a orquestração geral, destacando como a infraestrutura foi estruturada para atender aos requisitos do projeto.

4.2.1 Criação do Cluster no GCP

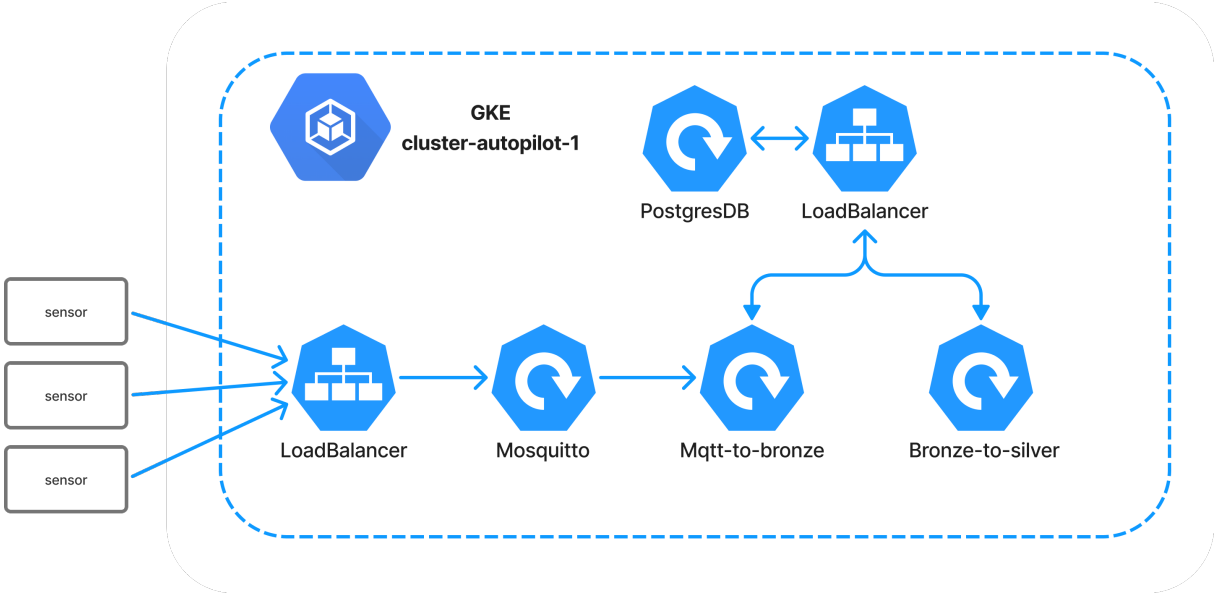
A infraestrutura foi estabelecida utilizando o Kubernetes como orquestrador de contêineres, hospedada no *Google Kubernetes Engine* (GKE) da GCP. O *cluster*, identificado como **"autopilot-cluster-1"**, foi configurado no modo *Autopilot*, o qual proporciona gerenciamento simplificado ao restringir personalizações manuais e ao automatizar elementos essenciais. Situado na região `us-central1`, o *cluster* opera no *Stable Release Channel*, assegurando atualizações estáveis e confiáveis. O gerenciamento do *cluster* é efetuado diretamente através do

painel do GCP, com métricas básicas de monitoramento e registro ativadas. O acesso à rede é configurado com suporte a *Domain Name System* (DNS) e endereços IPv4, enquanto a faixa de endereços padrão para pods segue o intervalo `/17`, permitindo escalabilidade. Tal configuração prioriza a estabilidade e a simplicidade operacional, eliminando a necessidade de gerenciar contas de serviço e configurações avançadas de segurança.

4.2.2 Visão Geral do Kubernetes

A arquitetura implementada no Kubernetes foi projetada de forma modular para facilitar o gerenciamento e a manutenção. Cada serviço principal foi containerizado utilizando *Docker* e implantado como um *Deployment* no Kubernetes, garantindo independência e escalabilidade. Na Figura 14, é possível visualizar a organização geral dos componentes.

Figura 14 – Diagrama simplificado da arquitetura implementada no Kubernetes.



Fonte: Autor.

Entre os serviços configurados no Kubernetes, destacam-se o *Mosquitto*, os serviços de *Mqtt-to-bronze*, *Bronze-to-silver* e o banco de dados PostgreSQL. O *Mosquitto* gerencia a comunicação entre os dispositivos IoT e o sistema, permitindo a ingestão de dados em tempo real. Os serviços *Mqtt-to-bronze*, *Bronze-to-silver* conectam as camadas Bronze e *Silver*, garantindo que o fluxo de dados ocorra de forma contínua e eficiente. O banco de dados foi configurado para armazenar os dados brutos e processados em diferentes tabelas, enquanto o monitoramento do *cluster* foi realizado diretamente pelo painel do GCP.

Os serviços do kubernetes são equivalentes às etapas da arquitetura final conforme apresentado na tabela 9.

Tabela 9 – Equivalência das etapas da arquitetura e serviços kubernetes

Serviço da arquitetura	**Serviço kubernetes**
Broker MQTT	Mosquitto
Ingestão de dados	Mqtt-to-bronze
Processamento de dados	Bronze-to-silver
Base de dados PostgreSQL	PostgreSQL

Fonte: Autor.

4.2.3 Deployments Implementados no Kubernetes

Abaixo serão descritos serviços implementados no kubernetes para desenvolvimento do sistema, com algumas informações mais detalhadas de seu funcionamento.

4.2.3.1 Serviço de LoadBalancer no Kubernetes

No Kubernetes, um serviço do tipo `LoadBalancer` permite expor aplicações executadas no cluster para a internet, atribuindo um endereço IP externo que direciona o tráfego para os *pods* correspondentes. Ao criar um serviço com `type: LoadBalancer`, o Kubernetes provisiona automaticamente um balanceador de carga externo (fornecido pelo provedor de nuvem subjacente), que distribui o tráfego de entrada entre os *pods* associados ao serviço, garantindo alta disponibilidade e escalabilidade da aplicação (KUBERNETES DOCUMENTATION, 2024).

4.2.4 Serviços de processamento de mensagens MQTT

O *deployment* do `"Mosquitto"` implementa o *broker MQTT* para gerenciar a comunicação de mensagens em tempo real entre dispositivos IoT e o sistema. Este serviço expõe portas específicas para conexões MQTT e *WebSocket*, utilizando uma configuração de *LoadBalancer* para facilitar a comunicação entre os dispositivos e o *broker*. A tecnologia baseia-se em uma imagem personalizada, também hospedada no *GitHub Container Registry*.

4.2.5 Serviços de processamento de dados

4.2.5.1 MQTT-to-bronze

Este serviço foi concebido com o propósito de armazenar os dados em sua forma bruta, conforme enviados pelos dispositivos, preservando-os em sua forma original para garantir rastreabilidade e flexibilidade. O serviço de ingestão, utiliza a biblioteca **paho-mqtt** em Python para estabelecer uma conexão com o *broker MQTT* e registrar os dados na tabela `bronze_solar_data` por meio da função `load_to_bronze`. Além disso, a cada inicialização deste serviço, verifica-se a existência das tabelas necessárias para sua operação, e na ausência de qualquer uma delas, estas são criadas automaticamente.

Descrição da Função `load_to_bronze`

A função `load_to_bronze` tem como objetivo inserir dados em uma tabela do banco de dados PostgreSQL. Esta função recebe um tópico e os dados como parâmetros de entrada. Um gerenciador de contexto é utilizado para estabelecer uma conexão com o banco de dados, enquanto um cursor é criado para a execução de comandos SQL. Após a inserção dos dados na tabela `bronze_solar_data`, a transação é confirmada. Em caso de sucesso na operação, o sistema registra uma mensagem de log indicando a inserção exitosa dos dados. Caso ocorra uma exceção, uma mensagem de erro é registrada. Por fim, a função assegura o fechamento adequado do cursor e da conexão, liberando assim os recursos alocados.

Código Fonte 4.1 – Função `load_to_bronze`

```
def load_to_bronze(topic, data):
    try:
        with get_connection() as connection:
            cursor = connection.cursor()

            # Insert data into the table
            insert_query = "INSERT INTO bronze_solar_data (
  payload, topic, received_at) VALUES (%s, %s, %s)"
            cursor.execute(
                insert_query, (json.dumps(data), topic, datetime.
  datetime.now())
            )

            connection.commit()
```

```
            logging.info(f"Data inserted into bronze_solar_data
for topic: {topic}")
 except Exception as e:
     logging.error(f"Error inserting data into
bronze_solar_data: {e}")
 finally:
     cursor.close()
     connection.close()
```

Na Figura 15, observamos os *logs* do funcionamento do serviço. Inicialmente, as tabelas foram verificadas, identificando a necessidade de criar a tabela `bronze_solar_data`, o que foi realizado com sucesso. Em seguida, o cliente MQTT do serviço conectou-se ao tópico `solar-data/#` e seus sub-tópicos no servidor, utilizando o endereço interno do *cluster* na porta 1883 e as credenciais adequadas, resultando em uma conexão bem-sucedida.

Figura 15 – Logs serviço Mqtt-to-bronze

```
2024-11-15 06:39:52 INFO Creating 'solar_data' table
2024-11-15 06:39:52 INFO Table 'bronze_solar_data' created.
2024-11-15 06:39:52 INFO MQTT connection started ('', 'solar-data/#')
2024-11-15 06:39:52 INFO Connected to 34.118.225.162 on port 1883
2024-11-15 06:39:52 INFO Connected successfully with code 0
2024-11-15 06:39:52 INFO Subscribed to topic 'solar-data'
```

Fonte: Autor.

4.2.5.2 Bronze-to-silver

Este serviço foi desenvolvido para transformar e estruturar os dados brutos provenientes da Camada Bronze. O *deployment* `"bronze-to-silver"` realiza essas transformações, padronizando horários no formato UTC, extraindo informações essenciais do campo `payload` e organizando os dados em tabelas distintas, categorizadas por tipo e identificador. A biblioteca *schedule* é utilizada para agendar a execução periódica da função responsável pelo processamento dos dados a cada minuto.

Para evitar redundância e reprocessamento de dados, foi implementado um sistema de *checkpoint*, que rastreia o último ponto de processamento, garantindo que apenas os novos dados sejam tratados em execuções subsequentes. Esse sistema realiza consultas SQL para recuperar o valor de `last_processed` na tabela de *checkpoint*, onde o campo `id` é igual a 1, retornando `None` caso não haja registros. Quando necessário, o campo `last_processed` é atualizado com

o *timestamp* mais recente, garantindo a persistência da atualização por meio de transações confirmadas.

Esse mecanismo assegura que o processamento seja retomado do ponto onde foi interrompido, evitando o retrabalho sobre dados já tratados. De forma semelhante ao serviço `mqtt-to-bronze`, a inicialização verifica automaticamente a existência das tabelas necessárias ao funcionamento do serviço, criando-as caso estejam ausentes.

Descrição da função `process_data`

A função `process_data` supervisiona o fluxo de processamento dos dados, começando pela extração da tabela `bronze_solar_data` por meio de uma consulta SQL. Na ausência de novos dados, uma mensagem de log é registrada, e a operação é encerrada. Caso novos dados sejam identificados, eles são transformados para o formato *Silver* utilizando a função `transform_bronze_to_silver`, que aplica a lógica de transformação linha por linha. Em seguida, os dados processados são gravados na tabela `silver_inverter_data`.

Após a gravação, o ponto de verificação (`last_received_at`) é atualizado com o *timestamp* mais recente dos dados processados, garantindo que o sistema registre o progresso. Por fim, uma mensagem de log é registrada para indicar a conclusão do processamento.

Código Fonte 4.2 – Função `process_data`

```
def process_data():
    logging.info("Starting data processing")
    bronze_data = read_bronze_data()
    if bronze_data.empty:
        logging.info("No new data to process")
        return
    silver_data = transform_bronze_to_silver(bronze_data)
    write_silver_inverter_data(silver_data)
    last_received_at = bronze_data["received_at"].max()
    update_checkpoint(last_received_at)
    logging.info("Data processing completed")
```

Descrição da função `process_payload`

A função `process_payload` processa a carga útil de uma mensagem, convertendo-a de JSON para um dicionário Python e extraindo campos específicos. Inicialmente, registra

uma mensagem de log e tenta carregar a carga útil como JSON, caso seja uma *string*, ou utiliza diretamente se já for um dicionário. Em seguida, cria um dicionário `processed_data`, mapeando os valores dos campos relevantes da carga útil para novas chaves padronizadas, como `ac_output_kwh`, `ac_output_power_factor`, `ac_voltage_v`, `dc_voltage_v` e `temperature_c`.

Se o processamento for bem-sucedido, registra uma mensagem de log e retorna o dicionário `processed_data`. Caso ocorra um erro na decodificação do JSON, registra uma mensagem de erro e retorna um dicionário vazio.

Código Fonte 4.3 – Função `process_payload`

```
def process_payload(payload):
    logging.info("Processing payload")
    try:
        payload_data = json.loads(payload) if isinstance(payload,
    str) else payload
        processed_data = {"measured_on": payload_data.get("
   measured_on")}
        for key, value in payload_data.items():
            if "ac_output_(kwh)" in key:
                processed_data["ac_output_kwh"] = value
            elif "ac_output_(power_factor)" in key:
                processed_data["ac_output_power_factor"] = value
            elif "ac_voltage_(v)" in key:
                processed_data["ac_voltage_v"] = value
            elif "dc_voltage_(v)" in key:
                processed_data["dc_voltage_v"] = value
            elif "temperature_(c)" in key:
                processed_data["temperature_c"] = value
        logging.info("Successfully processed payload")
        return processed_data
    except json.JSONDecodeError as e:
        logging.error(f"Error decoding JSON payload: {e}")
        return {}
```

Atualmente, a Camada Silver utiliza apenas uma tabela. Caso haja a necessidade de adicionar mais tabelas no futuro, basta criar novas funções específicas para cada tabela e integrá-las à lógica de processamento existente.

A Figura 16 apresenta os logs gerados pelo serviço `bronze-to-silver`, demonstrando a execução das etapas principais do pipeline. Essas mensagens evidenciam o funcionamento

Figura 16 – Logs de inicialização do serviço `bronze-to-silver`

```
2024-11-16 17:48:04,286 - INFO - Starting the scheduler
2024-11-16 17:48:04,286 - INFO - Creating checkpoint table if not exists
2024-11-16 17:48:04,419 - INFO - Database connection closed
2024-11-16 17:48:04,419 - INFO - Checkpoint table is ready
2024-11-16 17:48:04,419 - INFO - Starting data processing
2024-11-16 17:48:04,419 - INFO - Reading data from bronze_solar_data table
2024-11-16 17:48:04,419 - INFO - Getting last checkpoint
2024-11-16 17:48:04,512 - INFO - Database connection closed
2024-11-16 17:48:04,607 - INFO - Database connection closed
2024-11-16 17:48:04,619 - INFO - Successfully read data from bronze_solar_data table
2024-11-16 17:48:04,620 - INFO - No new data to process
2024-11-16 17:48:05,622 - INFO - Scheduler is running
2024-11-16 17:48:06,623 - INFO - Scheduler is running
```

do sistema, incluindo a verificação e criação da tabela de *checkpoint*, a leitura de dados da tabela `bronze_solar_data`, a utilização do ponto de *checkpoint* para evitar o reprocessamento de registros já tratados e a execução contínua do agendador. Esse registro detalhado reforça a confiabilidade e rastreabilidade do serviço durante sua operação.

4.2.6 Armazenamento e modelagem dos dados

O Banco de dados implementado foi o `"PostgreSQL"` configura a base relacional que armazena os dados em todas as etapas da arquitetura *Medallion*. A persistência dos dados é garantida pelo uso de *PersistentVolumeClaims* (PVCs), enquanto *probes* de *liveness* e *readiness* monitoram continuamente a disponibilidade do serviço. Essa configuração assegura a integridade dos dados, mesmo em casos de reinicialização de pods.

4.2.6.1 Camada Bronze

O esquema da tabela responsável pelo armazenamento dos dados na Camada Bronze é apresentado abaixo:

Código Fonte 4.4 – Esquema da tabela `bronze_solar_data`.

```
CREATE TABLE IF NOT EXISTS bronze_solar_data (
    id SERIAL PRIMARY KEY,
    payload JSONB NOT NULL,
    topic VARCHAR(255) NOT NULL,
    received_at TIMESTAMP NOT NULL
);
```

Essa configuração assegura que os dados brutos sejam armazenados de maneira consistente e confiável, servindo como ponto de entrada para os processos subsequentes de transformação e análise.

4.2.6.2 Camada *Silver*

O esquema da tabela correspondente à Camada *Silver* é apresentado a seguir:

Código Fonte 4.5 – Esquema da tabela `silver_inverter_data`.

```
CREATE TABLE IF NOT EXISTS silver_inverter_data (
    measured_on TIMESTAMP,
    ac_output_kwh FLOAT,
    ac_output_power_factor FLOAT,
    ac_voltage_v FLOAT,
    dc_voltage_v FLOAT,
    temperature_c FLOAT,
    received_at TIMESTAMP,
    topic VARCHAR
);
```

A Camada *Silver* realiza o processamento incremental dos dados provenientes da Camada Bronze. Utiliza um sistema de *checkpoint* para garantir que apenas os registros novos sejam transformados, otimizando o uso de recursos e melhorando a eficiência do sistema.

4.2.7 Pipeline de CI/CD para Implantação de Serviços

Uma das etapas cruciais no desenvolvimento deste trabalho foi a implementação de uma pipeline de CI/CD, utilizando *GitHub Actions* e o GHCR. Essa pipeline automatiza o ciclo de vida de desenvolvimento e implantação em uma arquitetura baseada em contêineres, orquestrada no GKE. A seguir, são detalhadas as etapas que compõem essa pipeline.

Construção e Implantação de Imagens Docker

A construção de imagens Docker é acionada automaticamente a partir de mudanças na *branch* `production`. Utilizando o *GitHub Actions*, o código é recuperado do repositório, seguido pela autenticação no GHCR. As imagens Docker são construídas com base em `Dockerfiles` localizados em diretórios específicos e enviadas ao GHCR com *tags* adequadas para versionamento. Foram desenvolvidos três principais *workflows*:

- **bronze-to-silver**: responsável pela transformação de dados entre camadas em uma arquitetura *Medallion*.
- **mosquitto-broker**: configuração e implantação do serviço de *broker* MQTT.

- **mqtt-to-bronze**: processamento e ingestão de mensagens MQTT na camada *Bronze*.

Deploy no Google Kubernetes Engine (GKE)

Com as imagens Docker disponíveis no GHCR, um *workflow* gerencia o processo de *deploy* no *cluster* Kubernetes. A autenticação com o *Google Cloud* é realizada automaticamente, permitindo a configuração e verificação de serviços essenciais, como *PostgreSQL* e *Mosquitto*. Arquivos *YAML* são aplicados para criar ou atualizar serviços, garantindo que estejam em execução com base na configuração mais recente. O deploy das aplicações no cluster é realizado com o comando `kubectl`.

Verificação e Monitoramento

Após o *deploy*, o estado dos serviços críticos é validado. São verificadas as condições de serviços como *PostgreSQL*, *Mosquitto* e o pipeline de dados *mqtt-to-bornze*, assegurando sua operação contínua e disponibilidade para processamento de dados em tempo real.

Ferramentas Utilizadas

O desenvolvimento dessa pipeline contou com as seguintes ferramentas principais:

- **GitHub Actions**: automação das etapas da pipeline de CI/CD.
- **GHCR**: armazenamento das imagens Docker.
- **Docker**: encapsulamento das dependências e configurações das aplicações.
- **Kubernetes**: orquestração dos serviços em um ambiente escalável e resiliente.

4.2.7.1 Considerações

A pipeline de CI/CD implementada nesta fase do trabalho contribuiu para a integração contínua, *deploy* eficiente e alta confiabilidade na operação dos serviços desenvolvidos.

4.2.8 Monitoramento e Orquestração

O monitoramento do sistema foi realizado por meio de painéis configurados no GCP e no *Prometheus*, fornecendo métricas detalhadas e em tempo real sobre a infraestrutura e a operação da pipeline de dados (AUTHORS, 2024). Abaixo estão as principais métricas monitoradas:

- **Uso de CPU** (`Compute Engine CPU utilization`): Percentual de utilização da CPU por máquina virtual, monitorado para identificar picos de processamento.

- **Uso de Memória** (`Kubernetes Container - Memory limit utilization`): Percentual de memória utilizada pelos contêineres do Kubernetes, auxiliando na identificação de possíveis gargalos de recursos.

Adicionalmente, o *Prometheus* foi configurado para coletar métricas específicas relacionadas ao desempenho da pipeline de dados:

- **Taxa de mensagens processadas** (`messages_processed_total`): Número de mensagens transformadas por segundo.

Essas informações permitem uma análise detalhada do desempenho do sistema, auxiliando na identificação de gargalos e ajustes necessários.

Durante os testes de carga, as métricas coletadas demonstraram o comportamento do sistema em condições variadas, possibilitando ajustes proativos para atender às demandas futuras. Essas práticas de monitoramento garantem a confiabilidade da infraestrutura e contribuem para a eficiência operacional do sistema.

4.3 TESTES E VALIDAÇÃO

O código desenvolvido implementa testes de carga que simulam sensores publicando dados de um conjunto de dados solar em um tópico MQTT. As principais etapas incluem:

- Configuração inicial de *host*, porta, tópico MQTT e credenciais, garantindo a correta inicialização.

- Processamento do conjunto de dados solar, substituindo valores nulos por 0, para assegurar integridade.

- Simulação de múltiplos sensores conectados ao *broker MQTT*, com suporte para autenticação e reconexão automática.

- Publicação periódica de dados nos tópicos MQTT, com mensagens rastreáveis e únicas.

4.3.1 Teste de Carga Nominal

Este teste avalia o sistema em carga nominal, simulando 25 sensores enviando mensagens a cada 2 minutos durante 10 minutos. O objetivo é verificar conectividade, entrega de mensagens e processamento dentro do intervalo esperado.

Tabela 10 – Critérios de Validação - Teste de Carga Nominal

Critério	**Descrição**
1	Todos os sensores conseguem se conectar ao *broker MQTT* sem falhas.
2	Mensagens enviadas pelos sensores são armazenadas corretamente na tabela `bronze_solar_data`.
3	O sistema processa todas as mensagens publicadas dentro do intervalo esperado.

Os resultados mostraram que todos os sensores conseguiram se conectar ao *broker MQTT* e enviar suas mensagens com sucesso. A Tabela 11 resume os dados coletados durante o teste.

Tabela 11 – Resultados dos testes de carga nominal no *Locust*.

Tipo	**Nome**	**Envios**	**Falhas**	**Média (ms)**	**Máx (ms)**	**Tamanho Médio (bytes)**
MQTT	connect	25	0	0	0	0
MQTT	publish	125	0	310	1075	306.4

Como ilustrado na Tabela 11, todas as 125 mensagens publicadas pelos sensores foram entregues sem falhas, com uma latência média de 310 ms e máxima de 1075 ms. Esses resultados estão dentro dos critérios estabelecidos, validando a confiabilidade e eficiência do sistema.

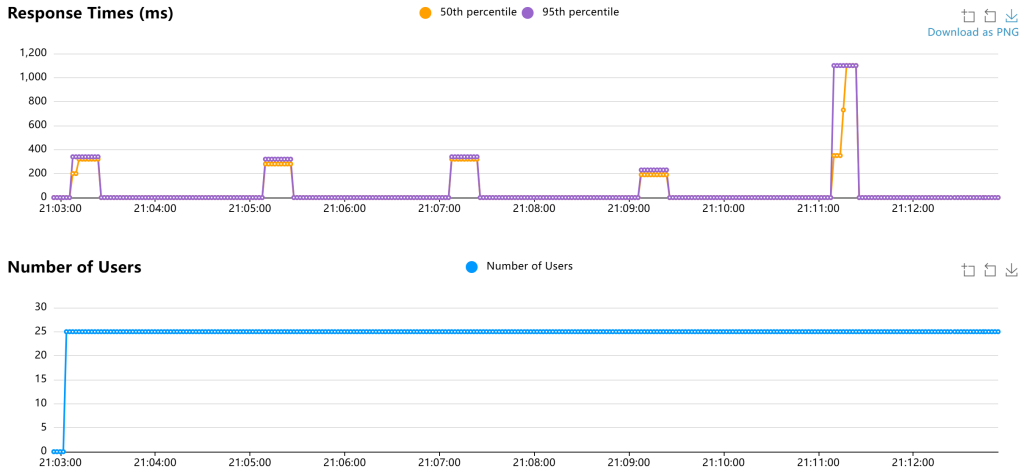
A Figura 17 ilustra o armazenamento bem-sucedido das mensagens na tabela `bronze_solar_data`, confirmando o funcionamento correto do sistema em condições de carga nominal.

4.3.2 Teste de Escalabilidade

Este teste avalia a capacidade do sistema de lidar com um número crescente de sensores conectados, iniciando com 10 sensores e aumentando para 100 que é o numero de inversores do maior sistema apresentado no *Solar Data Prize 2023*. As mensagens foram enviadas a cada 60 segundos durante 10 minutos, resultando em 1000 mensagens.

Os resultados demonstraram que o sistema foi capaz de atender aos critérios estabelecidos. A Tabela 13 apresenta os dados coletados durante o teste.

Figura 17 – Armazenamento das mensagens na tabela `bronze_solar_data`.



Fonte: Elaborado pelo autor.

Tabela 12 – Critérios de Validação - Teste de Escalabilidade

Critério	**Descrição**
1	O sistema mantém a estabilidade com 100 sensores publicando mensagens regularmente.
2	Nenhuma mensagem é perdida, mesmo com 100 sensores simultâneos.
3	A latência média de processamento por mensagem não ultrapassa dois segundos.

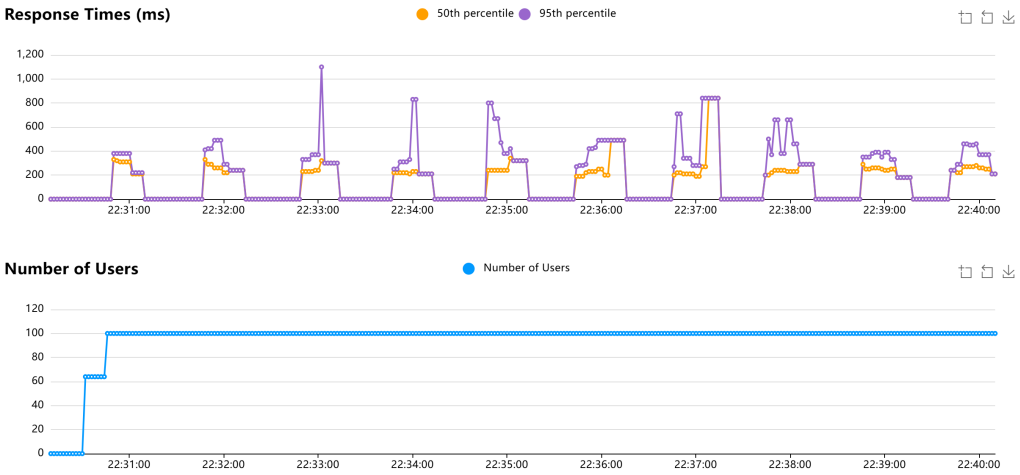
Tabela 13 – Resultados do teste de escalabilidade no *Locust*.

Tipo	**Nome**	**Envios**	**Falhas**	**Média (ms)**	**Máx (ms)**	**Tamanho Médio (bytes)**
MQTT	connect	100	0	0	0	0
MQTT	publish	1000	0	265.99	1087	304.2

Com base na análise dos resultados, como mostrado na Tabela 13 e na Figura 18, o sistema demonstrou um desempenho consistente e satisfatório. Durante o experimento, todos os 100 usuários mantiveram-se conectados de forma estável, sem interrupções ou falhas na comunicação. Além disso, 100% das mensagens enviadas foram entregues e processadas com sucesso, evidenciando a confiabilidade da plataforma.

A latência máxima registrada para entrega foi de 1087 milissegundos, bem abaixo do limite de dois segundos estabelecido como critério de aceitação. O sistema apresentou baixo consumo de recursos computacionais, com picos de CPU em 26% e uso máximo de memória em 18%, como mostrado nas Figuras 19 e 20.

Figura 18 – Resultados do teste de escalabilidade.



Fonte: Elaborado pelo autor.

Figura 19 – Uso de CPU durante o teste de escalabilidade.

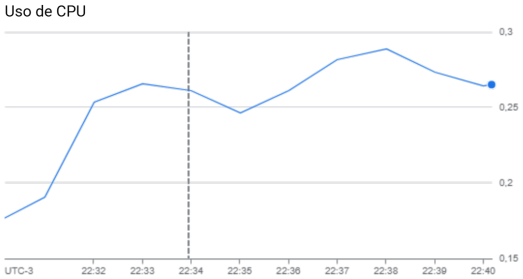
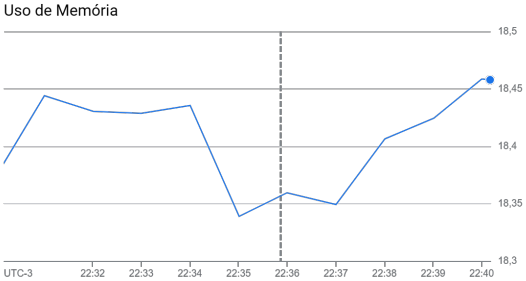


Figura 20 – Uso de memória durante o teste de escalabilidade.



Fonte: Elaborado pelo Autor.

4.3.3 Teste de Integridade e Qualidade dos Dados

Este teste avalia a integridade e a qualidade dos dados gerados durante o teste de escalabilidade, garantindo que as informações armazenadas nas tabelas `bronze_solar_data` e `silver_inverter_data` estejam consistentes, completas e corretas. Foram realizadas quatro consultas específicas para validar a integridade, qualidade e possíveis discrepâncias nos dados.

A seguir, são apresentados os testes realizados:

Teste 1: Validação de Integridade na Tabela bronze_solar_data

Esta consulta verificou se todos os registros na tabela `bronze_solar_data` continham os campos `payload`, `topic` e `received_at` devidamente preenchidos. O resultado confirmou

Tabela 14 – Critérios de Validação - Teste de Integridade e Qualidade dos Dados

Critério	**Descrição**
1	Todos os registros na tabela `bronze_solar_data` devem conter um `payload`, `topic` e `received_at` válidos.
2	Dados na tabela `silver_inverter_data` devem estar normalizados e livres de valores inconsistentes ou nulos.
3	Todas as mensagens esperadas devem estar presentes, sem duplicação ou ausência de dados.
4	Não devem ser identificadas discrepâncias significativas nos valores analisados entre as tabelas `bronze_solar_data` e `silver_inverter_data`.

que todos os registros estavam íntegros, sem valores nulos ou inconsistências.

Teste 2: Validação de Qualidade na Tabela silver_inverter_data

Nesta consulta, foi validada a presença de valores nulos ou inconsistentes nos campos principais da tabela `silver_inverter_data`, como `ac_output_kwh`, `ac_voltage_v`, e `temperature_c`. Nenhuma irregularidade foi identificada, comprovando a qualidade dos dados processados.

Teste 3: Extração Detalhada para Análise de Discrepâncias

Esta consulta verificou se os valores transformados na tabela `silver_inverter_data` estavam de acordo com os dados originais na tabela `bronze_solar_data`, analisando registros específicos por `topic`. Não foram encontradas discrepâncias significativas entre os dados brutos e os processados.

Teste 4: Análise Resumida de Qualidade na Tabela silver_inverter_data

Por fim, foi realizada uma análise resumida para validar a distribuição e a consistência dos dados na tabela `silver_inverter_data`, incluindo valores médios e máximos dos campos principais. Os resultados indicaram que os dados estavam devidamente normalizados e coerentes como podem ser observado na Tabela 15.

Tabela 15 – Resultados do Teste 4

Tópico	Total Mensagens	AC KWh	AC Tensão (V)	DC Tensão (V)
solar-data/inverter/1	10	2.91	36.54	122.57
solar-data/inverter/2	10	2.91	36.54	122.57
solar-data/inverter/3	10	2.91	36.54	122.57
solar-data/inverter/4	10	2.91	36.54	122.57
solar-data/inverter/5	10	2.91	36.54	122.57
...	...	...	...	...
solar-data/inverter/100	10	2.91	36.54	122.57

Fonte: Elaborado pelo autor.

Analise dos resultados do teste de qualidade dos dados

Os testes foram bem-sucedidos em todos os critérios estabelecidos, demonstrando que a plataforma é capaz de garantir a integridade e a qualidade dos dados mesmo em cenários de alta escalabilidade. A validação minuciosa assegura que os dados armazenados podem ser utilizados com confiança para análises futuras e para a geração de informações relevantes.

Esses resultados reforçam a robustez do sistema e sua adequação para aplicações reais que exigem alta confiabilidade e qualidade nos dados processados.

O fato de não terem sido observadas falhas nas mensagens processadas não implica que o sistema seja completamente à prova de falhas. Contudo, os resultados demonstram alta confiabilidade ao lidar com uma carga de até 100 clientes conectados simultaneamente, mantendo a frequência de mensagens definida. É importante destacar que, em cenários de escalabilidade para volumes e frequências significativamente maiores, é possível que ocorram falhas mínimas, ainda que o sistema mantenha um desempenho robusto dentro dos limites testados.

4.4 CONSIDERAÇÕES FINAIS

Os resultados apresentados neste capítulo demonstraram a eficiência, robustez e confiabilidade da plataforma desenvolvida para ingestão, processamento e armazenamento de dados provenientes de sensores IoT. A seguir, destacam-se os principais aspectos observados:

- **Desempenho Satisfatório em Cenários de Carga e Escalabilidade**:
  - No teste de carga nominal, o sistema processou um total de 125 mensagens sem falhas, com uma latência máxima registrada de 1075 ms, validando sua eficiência

em condições padrão de operação.

  - No teste de escalabilidade, o sistema gerenciou 1000 mensagens provenientes de 100 sensores conectados simultaneamente, sem interrupções ou falhas na comunicação, apresentando uma latência máxima de 1087 ms. Esses resultados atestam a capacidade da arquitetura em suportar cenários de alta demanda.

- **Garantia de Integridade e Qualidade dos Dados**:

  - Os testes de integridade e qualidade confirmaram a consistência e confiabilidade dos dados armazenados nas tabelas `bronze_solar_data` e `silver_inverter_data`. Todos os critérios de validação foram atendidos, assegurando que os dados estão devidamente normalizados e livres de inconsistências, prontos para análises futuras.

- **Eficiência no Uso de Recursos Computacionais**:

  - Mesmo sob condições de alta carga, o sistema demonstrou um consumo controlado de recursos, com picos de utilização de CPU em 26% e de memória em 18%. Esses resultados indicam que a arquitetura possui margem significativa para expansão sem comprometer a performance operacional.

Com base nos resultados obtidos, conclui-se que a plataforma atende plenamente aos requisitos de confiabilidade, escalabilidade e qualidade, sendo capaz de operar de forma robusta em ambientes de alta demanda. A arquitetura desenvolvida mostrou-se flexível e preparada para futuras adaptações e expansões, consolidando-se como uma solução adequada para aplicações reais no contexto de sistemas IoT e análise de dados.

5 CONCLUSÃO

5.1 ANÁLISE DOS OBJETIVOS

O objetivo principal deste trabalho foi o desenvolvimento de uma plataforma para monitoramento de equipamentos em uma planta solar, com capacidade de receber, processar e armazenar *payloads* em uma infraestrutura baseada em computação em nuvem. Os resultados obtidos demonstram o sucesso no alcance desse objetivo, assim como no cumprimento dos objetivos específicos propostos. A seguir, cada objetivo específico definido no Capitulo 1 é discutido, juntamente com os métodos empregados para sua realização e os resultados alcançados.

Estudo de Sistemas IoT e Protocolos de Comunicação

Para assegurar uma escolha fundamentada das tecnologias mais apropriadas, foi conduzida uma investigação sobre os principais sistemas IoT e protocolos de comunicação empregados em cenários de monitoramento em tempo real. Entre as tecnologias avaliadas, o protocolo MQTT foi escolhido devido à sua leveza, eficiência e confiabilidade na transmissão de dados. Este protocolo foi implementado utilizando o *broker Mosquitto*, que desempenhou um papel central na arquitetura do sistema ao conectar os dispositivos IoT à plataforma de processamento. Durante os testes, o MQTT evidenciou um desempenho consistente e escalável, corroborando sua seleção para cenários de alta demanda.

Análise de Ferramentas e Infraestrutura em Nuvem

A análise das principais plataformas de computação em nuvem levou à escolha do *Google Cloud Platform* (GCP) como provedor mais adequado para atender aos requisitos do sistema. O GCP ofereceu suporte nativo ao Kubernetes, garantindo escalabilidade, flexibilidade e confiabilidade. A modularidade proporcionada pelo Kubernetes simplificou a manutenção e permitiu a adaptação da plataforma para novas demandas. As ferramentas de monitoramento nativas do GCP também desempenharam um papel essencial na análise do desempenho durante os testes de carga e escalabilidade.

Desenvolvimento de uma Arquitetura de Processamento de Dados Eficiente

A arquitetura final foi projetada seguindo o modelo *Medallion*, com uma separação clara entre as camadas *Bronze* e *Silver*. A camada *Bronze* foi responsável pelo armazenamento dos dados brutos recebidos diretamente dos dispositivos IoT, preservando sua integridade e rastreabilidade. Já a camada *Silver* realizou o processamento e a transformação dos dados, estruturando-os em tabelas organizadas para análises específicas. Essa arquitetura modular e escalável assegurou eficiência no processamento e flexibilidade para integração com novos serviços e sistemas no futuro.

Modelagem de Dados e Implementação de um Fluxo Otimizado

Os dados foram modelados no PostgreSQL, com esquemas dedicados para as camadas *Bronze* e *Silver*. A tabela `bronze_solar_data` armazenou os dados brutos, enquanto a tabela `silver_inverter_data` estruturou os dados processados de maneira normalizada. O fluxo de dados foi otimizado utilizando processamento incremental com *checkpoints*, garantindo que apenas novos dados fossem processados, reduzindo o uso de recursos e aumentando a eficiência do sistema. A separação de responsabilidades por meio de serviços independentes contribuiu para a escalabilidade e a modularidade da plataforma.

Testes de Funcionamento

A validação da plataforma foi realizada por meio de uma série de testes de carga, escalabilidade e integridade dos dados, utilizando tanto dados reais quanto simulados. No teste de carga nominal, a plataforma demonstrou capacidade de lidar com 25 sensores enviando mensagens simultaneamente, com uma latência média de 310 ms e máxima de 1.075 ms. Já no teste de escalabilidade, o sistema manteve desempenho estável com 100 sensores, processando 1.000 mensagens com uma latência média de 265 ms. Os testes de integridade e qualidade dos dados confirmaram a consistência e a confiabilidade das informações armazenadas, validando a robustez do sistema mesmo em cenários de alta demanda.

5.2 CONSIDERAÇÕES FINAIS

Este trabalho representou um avanço significativo no monitoramento de plantas solares, integrando tecnologias de IoT, computação em nuvem e uma arquitetura de dados escalável.

Os resultados demonstram que a plataforma atende plenamente aos requisitos funcionais e não funcionais definidos, oferecendo uma solução confiável, eficiente e escalável para aplicações no setor de energias renováveis.

A modularidade e a flexibilidade da arquitetura implementada tornam a plataforma facilmente adaptável para outros cenários de monitoramento de equipamentos, ampliando seu potencial de aplicação. Os testes realizados validaram o desempenho e a confiabilidade do sistema, reforçando sua viabilidade para implantação em ambientes reais.

Além disso, este trabalho destaca a importância de práticas robustas de modelagem de dados, processamento incremental e uso de tecnologias modernas para desenvolver plataformas escaláveis. Essas práticas oferecem um modelo replicável e ampliável para futuros projetos no domínio de IoT e processamento de dados, promovendo inovações que podem impactar positivamente diversos setores da indústria.

REFERÊNCIAS

AMAZON WEB SERVICES. **What is Amazon S3?** [S.l.: s.n.], 2024. Acessado em 14 de novembro de 2024. Disponível em: <https://aws.amazon.com/s3/>.

AMJAD, Anam; AZAM, Farooque *et al.* A Systematic Review on the Data Interoperability of Application Layer Protocols in Industrial IoT. **IEEE Access**, IEEE, 2021.

ANSYAH, Adi Surya Suwardi; VIERI, Matthew *et al.* MQTT Broker Performance Comparison between AWS, Microsoft Azure, and Google Cloud Platform. In: **IEEE ICRTEC 2023**. [S.l.]: IEEE, 2023.

ASHTON, Kevin. That 'Internet of Things' Thing. **RFID Journal**, 2009. Disponível em: <https://www.rfidjournal.com/articles/view?4986>.

AUTHORS, Prometheus. **Prometheus Documentation: Overview**. [S.l.: s.n.], 2024. Acessado em 28 de novembro de 2024. Disponível em: <https://prometheus.io/docs/introduction/overview/>.

BECK, Kent *et al.* **Manifesto para o Desenvolvimento Ágil de Software**. [S.l.: s.n.], 2001. Acessado em 27 de novembro de 2024. Disponível em: <http://agilemanifesto.org/>.

BISWAS, Abdur Rahim; GIAFFREDA, Raffaele. IoT and Cloud Convergence: Opportunities and Challenges. **IEEE World Forum on Internet of Things**, IEEE, 2014.

DATABRICKS. **What is Data Processing?** [S.l.: s.n.], 2024. Acessado em 14 de novembro de 2024. Disponível em: <https://www.databricks.com/discover/glossary/data-processing>.

_____. **What is the Medallion Architecture?** [S.l.: s.n.], 2024. Acessado em 14 de novembro de 2024. Disponível em: <https://www.databricks.com/glossary/medallion-architecture>.

DELINE, Chris *et al.* **Photovoltaic Data Acquisition (PVDAQ) Public Datasets**. [S.l.: s.n.], dez. 2021. United States. DOI: 10.25984/1846021. Disponível em: <https://data.openei.org/submissions/4568>.

DOCKER. **Docker Documentation**. [S.l.: s.n.], 2024. Acessado em 14 de novembro de 2024. Disponível em: <https://docs.docker.com/>.

DOMINGUEZ, Daniel. **Google Cloud IoT Core Will Be Shut Down in 2023**. Acessado em 24 de novembro de 2024. agosto 2022. Disponível em: <https://www.infoq.com/news/2022/08/google-iot-core-discontinued/>.

ECLIPSE FOUNDATION. **2023 IoT & Edge Developer Survey Report**. [S.l.: s.n.], outubro 2023. Available under CC BY 4.0 license. Disponível em: <https://iot.eclipse.org/>. Acesso em: 14 nov. 2024.

_____. **Eclipse Mosquitto Documentation**. [S.l.: s.n.], 2024. Acessado em 14 de novembro de 2024. Disponível em: <https://mosquitto.org/documentation/>.

EUROSTAT. **Cloud computing - statistics on the use by enterprises**. [S.l.: s.n.], 2023. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Cloud_computing_-_statistics_on_the_use_by_enterprises. Acessado em 24 de maio de 2024.

AL-FUQAHA, Ala; GUIZANI, Mohsen *et al.* Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. **IEEE Communications Surveys & Tutorials**, IEEE, v. 17, n. 4, p. 2347–2376, 2015.

GITHUB, INC. **Trabalhar com o Registro de Contêineres**. [S.l.: s.n.], 2024. https://docs.github.com/pt/packages/working-with-a-github-packages-registry/working-with-the-container-registry. Acessado em 31 de outubro 31 de 2024.

GOOGLE CLOUD. **MQTT Broker Architecture: Connected Devices**. [S.l.: s.n.], 2024. https://cloud.google.com/architecture/connected-devices/mqtt-broker-architecture?hl=pt-br. Acessado em 14 de novembro de 2024.

IBM. **What are Message Brokers?** [S.l.: s.n.], 2024. Acessado em 27 de novembro de 2024. Disponível em: <https://www.ibm.com/br-pt/topics/message-brokers>.

JOSHI, Sagar Jatin; MAMANIYA, Smit; SHAH, Ruchit. Integration of Intelligent Manufacturing in Smart Factories as part of Industry 4.0 - A Review. In: **2022 SARDAR PATEL INTERNATIONAL CONFERENCE ON INDUSTRY 4.0 - NASCENT TECHNOLOGIES AND SUSTAINABILITY FOR 'MAKE IN INDIA' INITIATIVE**. [S.l.: s.n.], 2022. P. 1–5. DOI: 10.1109/SPICON56577.2022.10180471.

KANG, Byungseok; KIM, Daecheon; CHOO, Hyunseung. Internet of Everything: A Large-Scale Autonomic IoT Gateway. **IEEE Transactions on Multi-Scale Computing Systems**, v. 3, n. 3, p. 206–214, 2017. DOI: 10.1109/TMSCS.2017.2705683.

KUBERNETES. **Kubernetes Documentation**. [S.l.: s.n.], 2024. Acessado em 14 de novembro de 2024. Disponível em: <https://kubernetes.io/docs/>.

KUBERNETES DOCUMENTATION. **Create an External Load Balancer**. [S.l.: s.n.], 2024. Accessed: December 11, 2024. Disponível em: <https://kubernetes.io/pt-br/docs/tasks/access-application-cluster/create-external-load-balancer/>.

LIN, Jie; YU, Wei *et al.* A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. **IEEE Internet of Things Journal**, IEEE, v. 4, n. 5, p. 1125–1142, 2017.

LOCUST TEAM. **What is Locust?** [S.l.: s.n.], 2024. Acessado em 28 de novembro de 2024. Disponível em: <https://docs.locust.io/en/stable/what-is-locust.html>.

AL-MASRI, Eyhab *et al.* Investigating Messaging Protocols for the Internet of Things (IoT). **IEEE Access**, IEEE, v. 8, p. 94879–94899, 2020. DOI: 10.1109/ACCESS.2020.2993363. Disponível em: <https://ieeexplore.ieee.org/document/9097015>.

MICROSOFT. **What is Microsoft SQL Server?** [S.l.: s.n.], 2024. Acessado em 14 de novembro de 2024. Disponível em: <https://www.microsoft.com/en-us/sql-server>.

MQTT ORGANIZATION. **MQTT: Lightweight messaging protocol for IoT**. [S.l.: s.n.], 2024. https://mqtt.org/. Acessado em novembro 15 de 2024.

ORACLE. **What is a Database? Database Management Defined**. [S.l.: s.n.], 2024. Acessado em 14 de novembro de 2024. Disponível em: <https://www.oracle.com/database/what-is-database/#:~:text=and%20Autonomous%20Databases-,Database%20Defined,database%20management%20system%20(DBMS)>.

POSTGRESQL. **About PostgreSQL**. [S.l.: s.n.], 2024. Acessado em 14 de novembro de 2024. Disponível em: <https://www.postgresql.org/about/>.

RED HAT. **What is Apache Kafka?** [S.l.: s.n.], 2024. Acessado em 27 de novembro de 2024. Disponível em: <https://www.redhat.com/pt-br/topics/integration/what-is-apache-kafka>.

REIS, Joe; HOUSLEY, Matt. **Fundamentals of Data Engineering: Plan and Build Robust Data Systems**. Sebastopol, CA: O'Reilly Media, 2022. ISBN 978-1-098-13980-3.

SEYYEDHOSSEINI, M.; YAZDIAN-VARJANI, A.; MOHAMADIAN, M. IOT Based Multi agent Micro Inverter for Condition Monitoring and Controlling of PV Systems. In: **2020 11TH POWER ELECTRONICS, DRIVE SYSTEMS, AND TECHNOLOGIES CONFERENCE (PEDSTC)**. [S.l.: s.n.], 2020. DOI: 10.1109/PEDSTC49159.2020.9088449.

STATISTA. **Worldwide Cloud IT infrastructure market spending 2013-2026**. [S.l.: s.n.], 2024. https://www.statista.com/statistics/503686/worldwide-cloud-it-infrastructure-market-spending/. Acessado em 22 de maio de 2024.

STATISTA MARKET INSIGHTS. **Internet of Things - Worldwide**. [S.l.: s.n.], 2024. https://www.statista.com/outlook/tmo/internet-of-things/worldwide. Acessado em 14 de maio de 2024.

SUNPOWER CORPORATION. **SunPower AC Module SPR-X22-360 Datasheet**. [S.l.: s.n.], 2024. Acessado em 27 de novembro de 2024. Disponível em: <https://solarforward.com/pdf/products/AC-Modules/SPR-X22-360-WHT-AC.pdf>.

SYNERGY RESEARCH GROUP. **Huge Cloud Market Sees a Strong Bounce in Growth Rate for the Second Consecutive Quarter**. [S.l.: s.n.], 2024. https://www.srgresearch.com/articles/huge-cloud-market-sees-a-strong-bounce-in-growth-rate-for-the-second-consecutive-quarter. Acessado em 22 de maio de 2024.

ŽIVIĆ, Miloš; NEMEC, Dejan; BOJOVIĆ, Živko. MQTT Protocol in IoT Environment: Comparison with CoAP and ZeroMQ Protocols. In: **PROCEEDINGS OF TELFOR 2023**. [S.l.]: IEEE, 2023.