UNIVERSIDADE DO ESTADO DO AMAZONAS
ESCOLA SUPERIOR DE TECNOLOGIA
SISTEMAS DE INFORMAÇÃO

SISTEMA INTELIGENTE DE MONITORAMENTO E AUTOMAÇÃO PARA AMBIENTES DE ANIMAIS DOMÉSTICOS COM IOT

MANAUS – AMAZONAS

DEZEMBRO – 2025

GUILHERME LUCAS TEIXEIRA SILVA

SISTEMA INTELIGENTE DE MONITORAMENTO E AUTOMAÇÃO PARA AMBIENTES DE ANIMAIS DOMÉSTICOS COM IOT

Trabalho de Conclusão de Curso apresentado à banca avaliadora do Curso de Sistemas de Informação, da Escola Superior de Tecnologia, da Universidade do Estado do Amazonas, como pré-requisito para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Edgard Luciano Oliveira da Silva

MANAUS – AMAZONAS

DEZEMBRO – 2025

Universidade do Estado do Amazonas – UEA

Escola Superior de Tecnologia – EST

Reitor: ***André Luiz Nunes Zogahib***

Vice-Reitor: ***Kátia do Nascimento Couceiro***

Diretor da Escola Superior de Tecnologia:

Jucimar Maia Da Silva Júnior

Coordenador do Curso de Sistemas de Informação:

Marcela Sávia Picanço Pessoa

Banca Avaliadora composta por: Data da Defesa: 04/12/2025

Prof. Dr. Edgard Luciano Oliveira da Silva (Orientador)

Prof. Dr. Luis Cuevas Rodriguez

Profa. Dra. Marcela Sávia Picanço Pessoa

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).
Sistema Integrado de Bibliotecas da Universidade do Estado do Amazonas.

S586s Silva, Guilherme Lucas Teixeira
Sistema inteligente de monitoramento e automação para ambientes de animais domésticos com IoT / Guilherme Lucas Teixeira Silva. Manaus : [s.n], 2025.
44 f.: il., color.; 21.0 cm.

TCC - Graduação em Sistemas de Informação- Universidade do Estado do Amazonas, Manaus, 2025.
Orientador: Edgard Luciano Oliveira da Silva.

1. Internet das Coisas. 2. Detecção de anomalias. 3. ESP32. 4. Bem-Estar Animal. 5. AWS. I. Edgard Luciano Oliveira da Silva (Orient.) II. Universidade do Estado do Amazonas. III. Título

CDU(1997)004.414.2

Universidade do Estado do Amazonas - UEA
Escola Superior de Tecnologia - EST
Curso de Sistemas de Informação

FOLHA DE APROVAÇÃO

Sistema Inteligente de Monitoramento e Automação para Ambiente de Animais Domésticos com IoT

Guilherme Lucas Teixeira Silva

Trabalho de Conclusão de Curso defendido e aprovado pela banca avaliadora constituída pelos professores:

Prof. Dr. Edgard Luciano Oliveira da Silva
Presidente

Profa.Dra. Marcela Sávia Picanço Pessoa
Avaliador

Prof. Dr. Luis Cuevas Rodrigues
Avaliador

Aprovado em:
Manaus, 04 de dezembro de 2025

Resumo

A crescente integração de animais de estimação como membros da família, somada à rotina atribulada dos tutores, impulsiona a busca por soluções tecnológicas que garantam o bem-estar animal. Este trabalho apresenta a concepção, validação e evolução de um sistema de baixo custo baseado em Internet das Coisas (IoT) para o monitoramento e automação de ambientes para animais de estimação. O protótipo, validado em simulação de alta fidelidade, utiliza um microcontrolador ESP32 para coletar dados de múltiplos sensores e controlar atuadores, orquestrados pela ferramenta visual Node-RED. Como evolução, e em resposta a uma análise de requisitos de Engenharia de Software, o trabalho implementa duas extensões avançadas com foco na validação de resultados. Primeiramente, desenvolveu-se um módulo de saúde preditiva utilizando *Machine Learning* (*Isolation Forest*). Nos testes em ambiente controlado, este módulo alcançou um *Recall* de 96,7% e *F1-Score* de 89,2%, demonstrando eficácia superior a métodos estáticos na detecção de anomalias comportamentais. Em segundo lugar, a arquitetura foi migrada para a nuvem AWS (IoT Core e Lambda), onde testes de integração ponta-a-ponta validaram a segurança e a execução correta de comandos de voz via Amazon Alexa. O objetivo final foi alcançado ao fornecer uma solução integrada que assegura a alimentação, hidratação e conforto térmico do animal, confirmando, através dos resultados obtidos, a viabilidade técnica da arquitetura proposta para o controle remoto e o fornecimento de *insights* preditivos sobre o bem-estar do *pet*.

Palavras-chave: Internet das Coisas; Detecção de Anomalias; AWS; ESP32; Bem-Estar Animal.

Abstract

The growing integration of pets as family members, coupled with the busy routines of owners, drives the search for technological solutions that ensure animal welfare. This work presents the design, validation, and evolution of a low-cost Internet of Things (IoT) based system for monitoring and automating pet environments. The prototype, validated in high-fidelity simulation, uses an ESP32 microcontroller to collect data from multiple sensors and control actuators, orchestrated by the visual tool Node-RED. As an evolution, and in response to a Software Engineering requirements analysis, the work implements two advanced extensions focused on result validation. First, a predictive health module was developed using *Machine Learning* (*Isolation Forest*). In controlled environment tests, this module achieved a *Recall* of 96.7% and an *F1-Score* of 89.2%, demonstrating superior efficacy compared to static methods in detecting behavioral anomalies. Second, the architecture was migrated to the AWS cloud (IoT Core and Lambda), where end-to-end integration tests validated the security and correct execution of voice commands via Amazon Alexa. The final objective was achieved by providing an integrated solution that ensures the animal's feeding, hydration, and thermal comfort, confirming, through the obtained results, the technical feasibility of the proposed architecture for remote control and the provision of predictive *insights* into the pet's well-being.

Keywords: Internet of Things; Anomaly Detection; AWS; ESP32; Animal Welfare.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Códigos

Capítulo 1

Introdução

1.1 Introdução

A presença de animais de estimação nos lares tornou-se um fenômeno social de grande relevância. Essa transformação na dinâmica familiar, no entanto, colide com os desafios da vida moderna, onde rotinas de trabalho e compromissos diários frequentemente resultam em longos períodos de ausência dos tutores, comprometendo a regularidade e a qualidade dos cuidados essenciais (RODRIGUES; JúNIOR, 2019). Falhas no manejo da dieta podem levar a problemas de saúde como obesidade ou desnutrição (ALVES, 2020), enquanto a falta de água fresca e um ambiente termicamente inadequado são fatores diretos que podem causar desidratação e estresse fisiológico.

Nesse contexto, a tecnologia, em particular a Internet das Coisas (IoT), surge como uma poderosa aliada. O mercado global de cuidados tecnológicos para animais de estimação (*pets*) reflete essa tendência (Grand View Research, 2024). Contudo, apesar da proliferação de dispositivos, o mercado ainda é dominado por soluções que operam de forma isolada, tratando o cuidado animal de maneira fragmentada.

O conceito de "bem-estar animal" pode ser tecnicamente aprofundado pela literatura da Pecuária de Precisão (*Precision Livestock Farming* - PLF), que utiliza sensores para monitorar o microclima e seu impacto na saúde (TAVARES et al., 2021; NEETHIRAJAN, 2017). Isso demonstra que o monitoramento ambiental não é apenas uma funcionalidade adicional, mas

um pilar para a saúde animal baseada em dados (MASSON et al., 2024).

Este trabalho propõe-se a documentar e validar não apenas um dispositivo físico, mas uma arquitetura de *software* integrada para o monitoramento e automação de ambientes domésticos. O foco central reside na aplicação de princípios de Sistemas de Informação para a orquestração de dados, integração com nuvem e validação de algoritmos preditivos.

1.2 Justificativa

A justificativa para este projeto reside, primariamente, na necessidade de integração de dados em um cenário de IoT fragmentado. Uma análise do mercado (JACOMINI, 2024; ZIMMER et al., 2023) revela um arquipélago de dispositivos proprietários e isolados: alimentadores que não se comunicam com bebedouros, e sistemas de monitoramento ambiental que não influenciam a lógica de automação. Essa abordagem em silos impede uma visão holística, tratando sintomas em vez de gerenciar um ecossistema de cuidado conjugado.

Sob a ótica de Sistemas de Informação, o desafio técnico não se limita à construção do *hardware* (engenharia eletrônica), mas concentra-se na Engenharia de *Software* necessária para orquestrar esses dispositivos heterogêneos. O projeto justifica-se pela proposição de uma arquitetura que utiliza tecnologias abertas (ESP32, MQTT, Node-RED) para viabilizar a interoperabilidade e a flexibilidade arquitetural, permitindo que dados de diferentes sensores sejam correlacionados para gerar informação útil.

Adicionalmente, este trabalho busca preencher a lacuna de validação em projetos acadêmicos de IoT. A pesquisa aprofunda a aplicação de princípios da Engenharia de *Software* ao promover a transição de um protótipo local para uma arquitetura de nuvem escalável, bem como a implementação de um módulo de análise de dados (*Machine Learning*) para validação conceitual em ambiente controlado. Essas etapas são fundamentais para demonstrar a robustez e a escalabilidade da solução proposta.

Socialmente, o projeto sugere um caminho para impactar positivamente a relação entre tutores e seus animais, reduzindo o estresse relacionado à ausência através de uma gestão baseada em dados.

1.3 Objetivos

O desenvolvimento deste trabalho é norteado pelos seguintes objetivos:

1.3.1 Objetivo Geral

Desenvolver e validar um sistema embarcado para o monitoramento e a automação do controle de temperatura, alimentação e hidratação de animais domésticos, com notificações em tempo real, painel de controle interativo e integração com serviços de nuvem para controle por voz e análise preditiva.

1.3.2 Objetivos Específicos

- Realizar um levantamento bibliográfico sobre Internet das Coisas (IoT), Pecuária de Precisão e algoritmos de detecção de anomalias para fundamentar as decisões de projeto;
- Adotar a metodologia de Prototipagem Evolucionária para o desenvolvimento incremental do *hardware* e *software*, permitindo ciclos de validação contínua;
- Definir a arquitetura de *software* e os requisitos do sistema baseados em princípios de Sistemas de Informação para garantir escalabilidade e integração;
- Implementar sensores para a medição contínua dos níveis de insumos (água e ração) e variáveis ambientais (temperatura e umidade), integrando-os ao microcontrolador ESP32 via protocolo MQTT;
- Estruturar um banco de dados relacional e um sistema de orquestração (Node-RED) para armazenamento histórico, visualização em *dashboard* e notificações em tempo real;
- Evoluir a arquitetura local para uma solução de nuvem (*Serverless*) utilizando serviços da AWS (IoT Core e Lambda), visando segurança e controle por comandos de voz via assistentes virtuais;

- Validar a inteligência do sistema através da implementação de um módulo de saúde preditiva (*Isolation Forest*), analisando métricas de desempenho na detecção de anomalias comportamentais em um ambiente controlado.

1.4 Estrutura do Trabalho

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica, revisando os principais conceitos de IoT e as novas tecnologias de nuvem e *Machine Learning* empregadas. O Capítulo 3 detalha a metodologia de Prototipagem Evolucionária e a abordagem de validação em ambiente controlado. O Capítulo 4 descreve a arquitetura do sistema, desde a concepção do protótipo simulado até a arquitetura de nuvem final, orientada a serviços. O Capítulo 5 apresenta os resultados obtidos, abrangendo a validação funcional do sistema e a análise de inteligência preditiva. O Capítulo 6 detalha a evolução da infraestrutura para a nuvem AWS e a integração com assistentes de voz. Finalmente, o Capítulo 7 apresenta as conclusões finais do trabalho e sugere os próximos passos para a evolução do projeto.

Capítulo 2

Fundamentação Teórica

Este capítulo aborda os conceitos e tecnologias essenciais que fundamentam o desenvolvimento do projeto.

2.1 Internet das Coisas (IoT) no Cuidado Animal

A Internet das Coisas (IoT) refere-se à interconexão de dispositivos físicos que coletam e compartilham dados pela internet. No setor de cuidados com animais, a IoT possibilita a criação de ecossistemas inteligentes que melhoram a qualidade de vida dos animais de estimação (*pets*) e oferecem conveniência aos tutores (RODRIGUES; JúNIOR, 2019; PARTHASARATHI et al., 2020). Soluções como coleiras com GPS, câmeras interativas e, de especial interesse para este trabalho, comedouros e bebedouros inteligentes, são exemplos práticos dessa aplicação. Esses sistemas não apenas automatizam tarefas, mas também permitem um cuidado proativo, baseado em dados, ao monitorar a saúde e o comportamento do animal, alinhando-se aos princípios da Pecuária de Precisão (*Precision Livestock Farming* - PLF), onde o monitoramento contínuo de parâmetros de saúde e ambiente é fundamental para o bem-estar (JACOMINI, 2024; BANHAZI et al., 2022; ILVO, 2023).

2.2 Sistemas Embarcados e o Microcontrolador ESP32

Sistemas embarcados são sistemas computacionais dedicados a funções específicas dentro de um sistema maior (JACOMINI, 2024). O coração desses sistemas é, frequentemente, um microcontrolador. Para este projeto, foi escolhido o ESP32, um microcontrolador de baixo custo da Espressif Systems. A escolha se justifica por suas características robustas, que incluem um processador *dual-core*, conectividade Wi-Fi e Bluetooth integradas, e um vasto conjunto de periféricos de entrada/saída (como *General Purpose Input/Output* (GPIOs), ADC, *Pulse Width Modulation* (PWM)), tornando-o ideal para aplicações de IoT que exigem processamento, conectividade e controle de múltiplos sensores e atuadores (JACOMINI, 2024).

2.3 Levantamento de Componentes: Sensores e Atuadores

A confiabilidade de um sistema de cuidado animal depende diretamente da precisão e robustez de seus componentes. A seguir, é apresentado um levantamento das escolhas de *hardware*.

2.3.1 Sensor de Peso (Célula de Carga e Módulo HX711)

Para medir a quantidade de ração, a combinação de uma célula de carga com o módulo HX711 é uma escolha interessante. O HX711 é um conversor analógico-digital (ADC) de 24 bits de alta resolução, projetado especificamente para amplificar e converter os minúsculos sinais elétricos de pontes de Wheatstone (como as encontradas em células de carga), tornando-o ideal para medições de peso precisas e de baixo custo (SparkFun Electronics, 2023; Quarktwin, 2024). A obtenção de leituras estáveis, no entanto, exige um processo cuidadoso de calibração e a implementação de algoritmos de filtragem de *software* para mitigar ruídos, um desafio prático mitigado neste projeto através da implementação de uma média móvel simples diretamente no *firmware*, que descarta leituras espúrias antes do envio via MQTT.

2.3.2 Sensor de Nível de Água: Levantamento de Alternativas

A escolha de um sensor ultrassônico de baixo custo como o HC-SR04 para medir o nível de água serviu como uma prova de conceito inicial. Contudo, um levantamento da literatura e dos requisitos de confiabilidade do sistema sugere que esta pode ser uma escolha adequada apenas para prototipagem. A precisão de sensores ultrassônicos é notoriamente afetada por fatores como espuma, turbulência na superfície, umidade e variações de temperatura. Fisicamente, a velocidade do som c no ar depende da temperatura T, aproximada por $c \approx 331{,}3 + 0{,}606\,T$ (m/s). Sem uma compensação térmica via *software*, a oscilação da temperatura ambiente introduz erros sistemáticos na medição da distância, tornando o dado não confiável para aferir o consumo milimétrico de água (BOQU Instrument, 2024, 2023; Renke, 2023).

Para endereçar proativamente essa vulnerabilidade, foi realizada uma análise comparativa abrangente, contrastando tecnologias de medição sem contato (ultrassom, laser) com uma abordagem de medição de contato e discreta (sensor de boia). Essa análise, consolidada na Tabela 2.1, avalia não apenas o custo, mas o princípio de funcionamento, a natureza da medição (contínua vs. discreta) e a resiliência de cada tecnologia frente aos desafios de um bebedouro para animais.

Tabela 2.1: Análise Comparativa de Tecnologias para Medição de Nível de Água.

Característica	**HC-SR04 (Ultrassônico)**	**VL53L1X (ToF - Laser)**	**Sensor de Boia (Mecânico)**
Princípio	*Time of Flight* (Som)	*Time of Flight* (Luz IR)	Chave magnética (*Reed Switch*) acionada por flutuador.
Tipo de Medição	Contínua (analógica)	Contínua (analógica)	Discreta (digital: cheio/vazio)
Contato com Água	Não	Não	Sim (imerso)
Prós	Custo irrisório, simplicidade de prova de conceito. Fornece medição contínua do nível.	Altíssima precisão (mm), imune a espuma/vapor, feixe estreito ideal para recipientes pequenos.	Altíssima confiabilidade para detecção de ponto único, baixo custo, imune a todas as condições da água (espuma, turbidez, vapor). Simplicidade elétrica.
Contras	Não é à prova d'água, muito sensível a espuma, vapor e turbulência (BOQU Instrument, 2024). Precisão afetada pela temperatura (Renke, 2023).	Pode ler o fundo do recipiente com água clara, requerendo calibração. Sensível à luz solar direta. Custo mais elevado.	Fornece apenas uma medição binária (um ponto de nível), não o volume exato. Risco de contaminação e desgaste mecânico por ter partes móveis imersas.
Adequação	Prova de conceito. Inadequado para um produto final confiável.	Excelente para monitoramento preciso e contínuo, mas exige ambiente controlado e possível calibração de *software*.	Ideal como sistema de segurança (ex: detectar nível mínimo para acionar alerta/enchimento) ou para lógica simples, mas insuficiente para monitoramento detalhado do consumo.

Fonte: Elaborado pelo autor (2025).

Esta análise informa que a escolha da tecnologia de sensoriamento de água é um *trade-off*

fundamental entre a granularidade da informação e a robustez da medição. Enquanto sensores ToF (*Time of Flight*) oferecem dados ricos e contínuos, sua complexidade e sensibilidade são maiores. Em contrapartida, um sensor de boia oferece uma informação binária, porém extremamente confiável. Para os fins deste projeto, que visa um monitoramento detalhado, a abordagem sem contato permanece preferencial, mas a vulnerabilidade do HC-SR04 aponta para a necessidade de sua substituição por uma tecnologia mais resiliente, como o ToF, ou a hibridização com um sensor de boia para redundância de segurança em trabalhos futuros.

2.3.3 Sensor de Temperatura e Umidade (DHT11 vs. DHT22)

Para o monitoramento ambiental, a escolha entre os sensores da família DHT deve ser bem fundamentada. O DHT22, embora ligeiramente mais caro, é tecnicamente superior ao DHT11, oferecendo maior precisão e uma faixa de operação mais ampla, fatores cruciais para garantir o conforto térmico do animal. A Tabela 2.2 detalha as diferenças.

Tabela 2.2: Comparativo técnico entre os sensores DHT11 e DHT22. Fonte: Compilado de (Seeed Studio, 2020; Random Nerd Tutorials, 2023; PCBasic, 2024).

Característica	**DHT11**	**DHT22 (AM2302)**
Faixa de Temperatura	0°C a 50°C	−40°C a 80°C
Precisão da Temperatura	±2°C	±0.5°C
Faixa de Umidade	20% a 90% RH	0% a 100% RH
Precisão da Umidade	±5% RH	±2% RH

2.3.4 Atuadores (Servo Motor e Válvula Solenoide)

A automação das tarefas de alimentação e hidratação depende de atuadores confiáveis para o controle físico dos insumos.

Para o controle de porções de ração, foi selecionado um Servo Motor. A justificativa para esta escolha, validada em trabalhos como o de (ALVES, 2020), reside na sua capacidade de controle de precisão angular. Diferente de um motor DC comum (que apenas gira continuamente), um servo motor pode ser instruído a girar para um ângulo exato (ex: 90 graus) e retornar. Essa característica é ideal para implementar um dispensador do tipo rosca ou alavanca, onde

o ângulo de rotação é diretamente proporcional ao volume de ração dispensado, garantindo a consistência das porções.

Para o controle do fluxo de água, a escolha foi uma Válvula Solenoide. Este componente atua como um portão eletromagnético (*on/off*) para líquidos. Sua principal vantagem é a simplicidade e robustez para o controle binário (liberar ou bloquear a água), sendo facilmente controlada por um microcontrolador (através de um relé ou transistor), o que a torna uma escolha direta e eficaz para a automação do reabastecimento do bebedouro.

2.4 Protocolo de Comunicação MQTT e Qualidade de Serviço (QoS)

O *Message Queuing Telemetry Transport* (MQTT) é um protocolo de mensagens leve, baseado no padrão publicar/assinar, ideal para ambientes de IoT com recursos limitados (RODRIGUES; JúNIOR, 2019; Amazon Web Services, 2024c). Sua arquitetura desacopla os dispositivos, tornando o sistema modular e escalável. Para aplicações críticas, é essencial compreender os níveis de Qualidade de Serviço (QoS) (Bevywise, 2023):

- **QoS 0 (At most once):** "Dispare e esqueça". Entrega rápida, mas sem garantia. Adequado para dados não críticos, como leituras de temperatura periódicas, onde a perda ocasional de um pacote é aceitável (Cedalo, 2023).
- **QoS 1 (At least once):** Garante que a mensagem seja entregue pelo menos uma vez, com confirmação. Pode haver duplicatas. É o padrão de fato para comandos em plataformas de nuvem gerenciadas.
- **QoS 2 (Exactly once):** Garante que a mensagem seja entregue exatamente uma vez, através de um *handshake* de quatro etapas. É o nível teórico mais confiável, porém mais lento (Bevywise, 2023; ThingsBoard, 2024).

É de se notar que, embora o QoS 2 seja o ideal teórico para comandos críticos (como "alimentar o *pet*"), as principais plataformas de nuvem, incluindo o AWS IoT Core, não implementam

o QoS 2, limitando o suporte ao QoS 0 e QoS 1. Conforme a própria documentação da AWS, "AWS IoT doesn't support publishing or subscribing with QoS level 2" (Amazon Web Services, 2024a). Esta é uma decisão de engenharia da plataforma: a complexidade e a latência do QoS 2 são trocadas pela simplicidade e velocidade do QoS 1. A responsabilidade de garantir a entrega "exatamente uma vez" (prevenção de duplicatas) é, portanto, movida da camada de protocolo (MQTT) para a camada de aplicação (a lógica do *software*, como a implementação de idempotência no *handler* do AWS Lambda) (Amazon Web Services, 2024b, 2016). Adicionalmente, a segurança do MQTT não é nativa; ele depende de camadas de transporte seguras como o TLS para criptografar os dados em trânsito (AL-HAWAWREH et al., 2024; Amazon Web Services, 2024c).

Tabela 2.3: Aplicação Estratégica dos Níveis de Qualidade de Serviço (QoS) do MQTT no contexto do trabalho.

Tipo de Mensagem	**Tópico Exemplo**	**Criticidade**	**QoS Escolhido**	**Justificativa Técnica**
Dados de Sensor (Temperatura/Umidade)	`canil/ambiente/temperatura`	Baixa	QoS 0	A perda ocasional de uma leitura ambiental não compromete o sistema. Prioriza-se a baixa latência e o mínimo de sobrecarga (Cedalo, 2023).
Dados de Sensor (Nível de Ração/Água)	`canil/racao/tigela/peso`	Média	QoS 1	Garante que a mensagem seja entregue **pelo menos uma vez**. É importante que o *dashboard* e a lógica de alerta recebam os dados.
Comando do Atuador (Liberar Água)	`canil/agua/valvula/comando`	Média	QoS 1	Garante a entrega do comando. Uma execução duplicada não é catastrófica, tornando o QoS 1 um bom equilíbrio entre confiabilidade e complexidade (Bevywise, 2023).
Comando do Atuador (Dispensar Ração)	`canil/racao/servo/comando`	Alta	QoS 1 *(forçado pela nuvem)*	Embora o QoS 2 seja o ideal teórico para evitar duplicatas (Bevywise, 2023), a plataforma de nuvem escolhida (AWS IoT Core) não o suporta. A garantia de "pelo menos uma vez" (QoS 1) é aceita, e a lógica de prevenção de duplicatas (idempotência) é movida para a camada de aplicação. Neste projeto, o *firmware* do ESP32 implementa uma verificação de estado (*state-check*): se o atuador já estiver em operação (`BUSY`), comandos subsequentes de `LIBERAR` são ignorados, garantindo a idempotência.

É relevante notar que a aplicação estratégica dos níveis de QoS, detalhada na Tabela 2.3, implica um *trade-off* direto de desempenho e consumo de recursos. A confiabilidade não é isenta de custos: o QoS 0 representa a linha de base com uma única mensagem `PUBLISH`. O

QoS 1 dobra o tráfego de rede para aquela mensagem, exigindo uma confirmação (`PUBACK`). Já o QoS 2, o mais robusto, quadruplica o fluxo com um *handshake* de quatro etapas (`PUBLISH`, `PUBREC`, `PUBREL`, `PUBCOMP`) (ThingsBoard, 2024). Essa sobrecarga (*overhead*) se traduz em maior latência e maior uso de memória e CPU tanto no cliente (ESP32) quanto no *broker* para gerenciar o estado das mensagens.

2.5 Node-RED como Orquestrador de Prototipagem Rápida

Node-RED é mais do que uma ferramenta de visualização; é um ambiente de desenvolvimento *low-code* que acelera a criação de protótipos e Produtos Mínimos Viáveis (MVPs) em projetos de IoT (qbee.io, 2023; PUSR, 2024). Sua interface visual baseada em fluxos permite a validação rápida de lógicas, a integração com diversas APIs e a criação de *dashboards* interativos com mínimo esforço de programação (Apifornia, 2023; IoT-Solution, 2024). Neste projeto, o Node-RED atua como um orquestrador central, um *middleware* flexível que conecta os sensores, a lógica de controle, o banco de dados e as notificações, alinhando-se à metodologia de Prototipagem Evolucionária.

2.6 Trabalhos Correlatos

A análise de projetos anteriores posiciona a presente proposta. Enquanto trabalhos como o de (RODRIGUES; JúNIOR, 2019) focaram em alimentadores de baixo custo e (ALVES, 2020) em controle via App dedicado, projetos mais recentes avançaram em áreas específicas. (JACOMINI, 2024) introduziu resiliência energética e identificação de animais de estimação (*pets*) com *Machine Learning* (ML). Pesquisas mais recentes exploram o uso de *deep learning* (com modelos como YOLOv5) para identificação individual precisa (O.E.C.-A. et al., 2024; LI et al., 2023). Outra abordagem, como a do projeto cirCAT (ZIMMER et al., 2023), propõe um ecossistema de múltiplos dispositivos para uma visão holística da saúde do animal. A Tabela 2.4 compara essas abordagens e destaca a contribuição única deste trabalho.

Tabela 2.4: Tabela comparativa de trabalhos correlatos.

Trabalho	**Foco Principal**	**Tecnologias Chave**	**Diferencial/Contribuição**	**Limitações Notáveis**
(RODRIGUES; JúNIOR, 2019)	Alimentador de baixo custo	NodeMCU, MQTT	Foco em acessibilidade e automação de ração.	Monitoramento ambiental e de água ausente.
(ALVES, 2020)	Controle via App dedicado	App Android, Bluetooth	Interface de usuário móvel dedicada.	Alcance limitado, sem conectividade nuvem.
(JACOMINI, 2024)	Identificação de pets e resiliência	ESP32-CAM, ML, No-Break	Uso de ML para múltiplos pets, backup de energia.	Foco primário na alimentação.
(O.E.C.-A. et al., 2024)	Alimentação com IA	YOLOv5, Sensores	Alta precisão na identificação e porcionamento.	Solução complexa, focada apenas na alimentação.
(ZIMMER et al., 2023)	Monitoramento holístico	Múltiplos dispositivos IoT	Visão integrada da saúde a partir de várias fontes.	Ecossistema complexo, não uma solução única.
Este Projeto	**Solução integrada e orquestrada**	**ESP32, MQTT, Node-RED**	**Arquitetura de orquestração centralizada e flexível via Node-RED, monitoramento integrado (ração, água, ambiente) em plataforma de baixo custo.**	**Dependência da precisão de sensores de baixo custo (a ser analisada).**

O presente projeto se diferencia ao propor uma solução de monitoramento mais abrangente (múltiplos sensores para ração, água e ambiente), utilizando o Node-RED como um orquestrador central e flexível, e integrando notificações via Telegram e um banco de dados MySQL, buscando um equilíbrio entre funcionalidade, custo e complexidade de implementação para um projeto de graduação.

2.7 Arquitetura de Nuvem para IoT (AWS)

A evolução de um protótipo de IoT para um produto escalável e seguro exige a migração de componentes locais, como um *broker* MQTT público (`test.mosquitto.org`), para uma plataforma de nuvem robusta. Este trabalho utiliza os serviços da Amazon Web Services (AWS) para construir uma arquitetura *serverless* (sem servidor), que gerencia a infraestrutura, a segurança e a escalabilidade automaticamente.

- **AWS IoT Core:** é um serviço gerenciado que atua como o *broker* MQTT profissional do sistema. Diferente de *brokers* públicos, ele impõe segurança rigorosa através de autenticação mútua (TLS), onde cada dispositivo (como o Node-RED) deve apresentar certificados de segurança válidos (ex: `.pem.key`, `.pem.crt`) para se conectar. Além disso, ele utiliza Políticas de IoT (semelhantes às políticas do IAM) para autorizar finamente quais ações (como `iot:Connect`, `iot:Publish`) são permitidas para quais tópicos (Amazon Web Services, 2024a).

- **AWS Lambda:** é o "cérebro" *serverless* da arquitetura. O Lambda permite a execução de código (ex: Python, Node.js) em resposta a eventos, sem a necessidade de provisionar ou gerenciar servidores. No contexto deste projeto, ele atua como o *middleware* tradutor

entre a API da Alexa e o protocolo MQTT. A Alexa gera um evento (a *Intent* de voz), que aciona a função Lambda; a função, por sua vez, executa a lógica de negócios e publica o comando MQTT apropriado no AWS IoT Core (Amazon Web Services, 2024b).

- **Alexa Skills Kit (ASK):** é a plataforma de desenvolvimento que permite "ensinar" novos comandos (habilidades) à assistente de voz Alexa. Através do Console de Desenvolvedor da Alexa, o sistema é configurado com *Intents* (intenções), que mapeiam frases faladas pelo usuário (ex: "ligar o ar") para uma ação específica, como acionar o *endpoint* da função AWS Lambda correspondente (Amazon Web Services, 2016).

2.8 Detecção de Anomalias (Machine Learning)

Para validar a inteligência do sistema e ir além do simples monitoramento, foi implementado um módulo de saúde preditiva. Este módulo utiliza técnicas de *Machine Learning* (ML) para detecção de anomalias (*outlier detection*). O objetivo não é diagnosticar, mas sim identificar mudanças sutis nos padrões de consumo de um animal que fujam do seu comportamento "normal", servindo como um alerta precoce para o tutor. Neste trabalho, foi selecionado o algoritmo *Isolation Forest* (Floresta de Isolamento). Este é um modelo de aprendizado não supervisionado ideal para detecção de anomalias por duas razões principais: (1) ele é eficaz em *datasets* com alta dimensionalidade (vários sensores); e (2) ele funciona sob a premissa de que anomalias são "poucas e diferentes", tornando-as mais fáceis de "isolar" na estrutura de dados do que os pontos normais (LIU; TING; ZHOU, 2008). Diferentemente de métodos baseados em distância (como KNN), que possuem custo computacional quadrático ($O(n^2)$), o Isolation Forest apresenta complexidade linear ($O(n)$). Essa característica de leveza computacional é estratégica para IoT, pois viabiliza, em trabalhos futuros, a migração do modelo da nuvem diretamente para o microcontrolador (*TinyML / Edge AI*), permitindo executar a inferência localmente no ESP32.

Com a apresentação do algoritmo *Isolation Forest*, encerra-se a fundamentação teórica que sustenta as decisões de projeto deste trabalho. A revisão abrangeu desde a camada física

(sensores e ESP32) e de comunicação (MQTT e Node-RED) até as camadas de inteligência e nuvem (AWS e *Machine Learning*). Estabelecida esta base conceitual, o próximo capítulo detalha a metodologia de Prototipagem Evolucionária adotada para a construção do sistema, descrevendo a modelagem da arquitetura e as estratégias de validação em ambiente simulado e controlado.

Capítulo 3

Metodologia e Modelagem do Sistema

Este capítulo descreve a abordagem metodológica, a arquitetura do sistema e os detalhes de *hardware* e *software* utilizados no projeto.

3.1 Metodologia

Foi adotada a metodologia de Prototipagem Evolucionária, um modelo de processo de *software* formal que consiste no desenvolvimento de versões incrementais e funcionais do sistema. Esta abordagem é particularmente adequada para projetos de IoT, pois os requisitos nem sempre são completamente conhecidos no início e podem evoluir (GeeksforGeeks, 2024). O modelo permite a validação contínua, a identificação precoce de desafios na integração de *hardware* e *software* e a incorporação de melhorias com base no *feedback* de cada ciclo (Exforsys Inc., 2013).

Este trabalho é apresentado em dois ciclos evolutivos:

- **Ciclo 1:** focou na validação da arquitetura de protótipo, lógica de sensores e atuadores, e orquestração de *middleware* em um ambiente simulado (Wokwi), conforme descrito no Capítulo 4.
- **Ciclo 2:** foca na evolução das camadas de *software* (Engenharia de *Software*) e na validação rigorosa do sistema. Para atender ao requisito de validação em ambiente controlado, este ciclo introduz uma metodologia de *Data Science*. Conforme detalhado no Capítulo

5, esta abordagem consiste em: (1) Geração de um *dataset* de alta fidelidade para simular 90 dias de operação do sistema (cenários normais e anômalos); e (2) a aplicação de um modelo de *Machine Learning* para validar a capacidade do sistema de detectar anomalias nesse ambiente controlado.

As simulações na plataforma Wokwi foram utilizadas como a etapa de prototipagem de *software* do Ciclo 1, alinhando-se às melhores práticas que recomendam o uso de ferramentas de simulação para validar a lógica do *firmware* antes da montagem física, acelerando o desenvolvimento e reduzindo riscos (RunTime Recruitment, 2024; Motius, 2020).

3.2 Limitações Experimentais e Impacto da Simulação

É imperativo reconhecer as restrições impostas pela validação via simulação. Embora o ambiente Wokwi permita a verificação lógica do *firmware* e da arquitetura de comunicação, ele abstrai desafios físicos críticos inerentes a uma implantação em *hardware* real. A Tabela 3.1 detalha essas divergências e seus impactos teóricos.

Tabela 3.1: Divergências entre o Ambiente Simulado e o Hardware Real.

Fator	**Ambiente Simulado (Wokwi)**	**Hardware Real (Implantação)**
Ruído nos Sensores	Idealizado. Leituras são limpas e determinísticas.	Sujeito a ruído elétrico e mecânico. Exigiria filtros de *software* (Kalman, Média Móvel) e *hardware* (capacitores).
Rede e Latência	Conexão perfeita e estável com o *broker*.	Sujeito a *jitter*, perda de pacotes e desconexões. Exigiria implementação de *buffer* local no ESP32.
Consumo Energético	Não simulado. Energia infinita.	Crítico. Exigiria otimização com *Deep Sleep* para viabilidade via bateria.
Calibração (HX711)	Fator de calibração fixo e linear.	Exige calibração física com pesos padrão e sofre deriva (*drift*) com temperatura e tempo.

Fonte: Elaborado pelo autor (2025).

Portanto, os resultados apresentados neste trabalho validam a lógica de orquestração e a arquitetura de *software*, mas não constituem uma validação de engenharia de produto final.

Uma vez delimitado o escopo experimental e reconhecidas as restrições da simulação, a base metodológica para o desenvolvimento do projeto está consolidada. Com essas premissas estabelecidas, o próximo capítulo dedica-se a detalhar a **Arquitetura do Sistema**, descrevendo a organização dos componentes físicos e lógicos e a evolução técnica da solução, desde o protótipo local até a arquitetura final orientada a serviços de nuvem.

Capítulo 4

Arquitetura do Sistema

A arquitetura do sistema foi projetada seguindo princípios de Engenharia de Software para suportar a evolução de um protótipo local para uma solução de nuvem escalável. Esta abordagem evolutiva permitiu validar inicialmente os componentes físicos e lógicos em ambiente controlado, preparando a base para a implementação final orientada a serviços distribuídos.

4.1 Arquitetura Física (Protótipo Simulado)

A camada física compreende o conjunto de *hardware* responsável pela interação direta com o ambiente do animal. Nesta fase, todos os componentes foram integrados e validados no ambiente de simulação Wokwi. A composição do *hardware* é a seguinte (conforme Figura 4.1):

1. **Núcleo de Controle:** o núcleo do sistema embarcado é uma placa de desenvolvimento com o microcontrolador ESP32-WROOM-32.

2. **Sensores de Insumos (Peso):** três sistemas de pesagem independentes foram implementados utilizando células de carga (2x 50kg para os reservatórios principais e 1x 5kg para a tigela), cada uma conectada a um módulo amplificador HX711 dedicado.

3. **Sensor de Nível (Água):** um sensor ultrassônico HC-SR04 foi posicionado para medir a distância até a superfície da água, inferindo o nível.

4. **Sensor Ambiental:** um sensor DHT22 monitora as condições de temperatura e umidade.

5. **Atuadores (Simulados):**

   - **Dispensador de Ração:** a automação é realizada por um Servo Motor.
   - **Dispensador de Água:** a lógica de controle do fluxo de água foi validada utilizando um LED como indicador visual.

A interconexão de todos esses componentes de *hardware* é detalhada no diagrama da Figura 4.1.

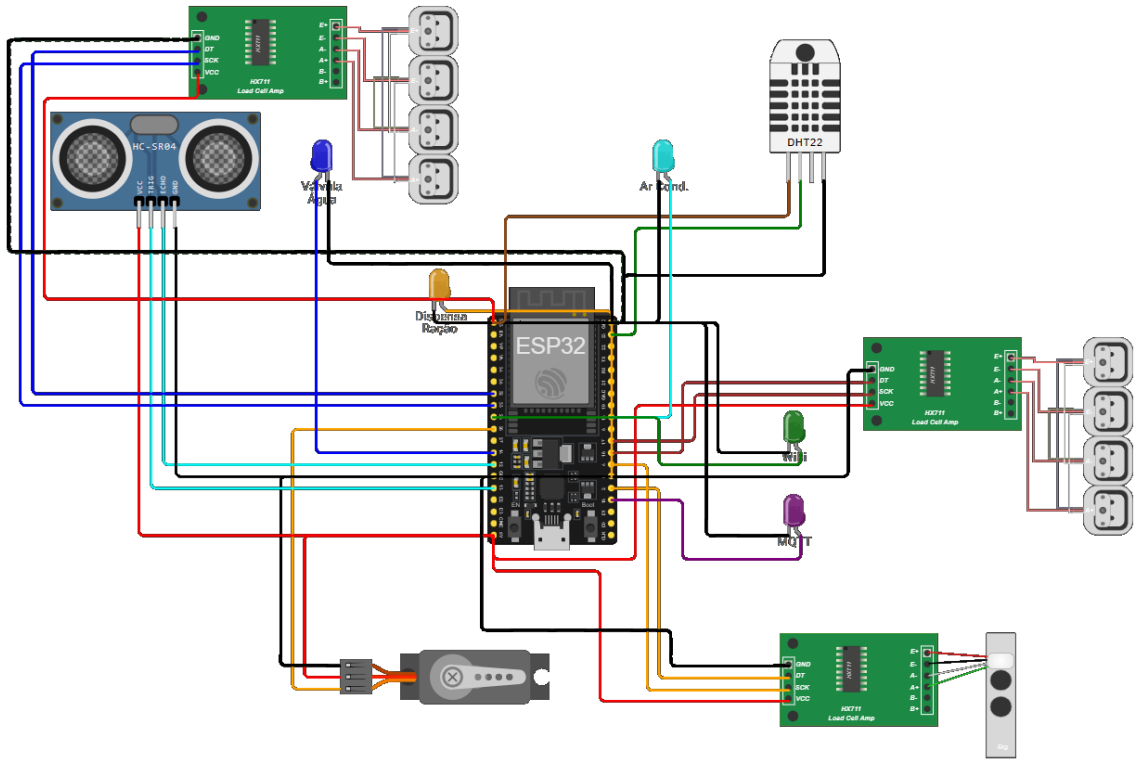

Figura 4.1: Diagrama da Arquitetura Física do sistema no ambiente de simulação Wokwi.

Fonte: Elaborado pelo autor (2025).

4.2 Arquitetura Lógica e Fluxo de Dados

Após a definição dos componentes físicos, é necessário estabelecer como os dados transitam e são processados pelo sistema. A arquitetura lógica detalha as camadas de *software*, os protocolos

de comunicação e as interfaces de integração que transformam leituras brutas de sensores em ações e notificações. Para evidenciar a evolução técnica do projeto sob a ótica da Engenharia de Software, a apresentação do fluxo de dados é dividida em dois momentos: a topologia inicial utilizada na validação do protótipo e a arquitetura definitiva orientada a serviços de nuvem.

4.2.1 Arquitetura de Prototipagem

A arquitetura lógica inicial define como os dados e comandos fluem entre os componentes locais e o *broker* MQTT público, conforme ilustrado na Figura 4.2.

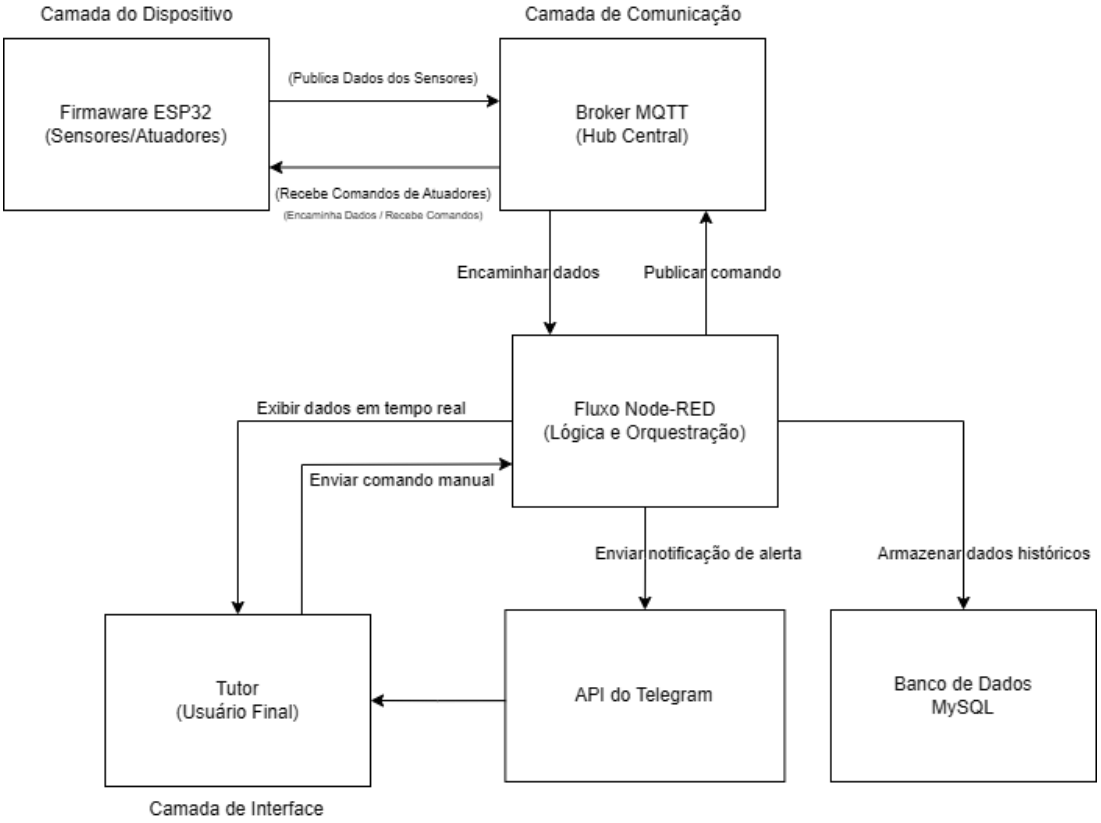

Figura 4.2: Diagrama da Arquitetura Lógica de Prototipagem.

Fonte: Elaborado pelo autor (2025).

O fluxo de dados é iniciado no *Firmware* do ESP32, que coleta as leituras dos sensores e as publica em um *Broker* MQTT público (`test.mosquitto.org`). Este *broker* atua como um intermediário, encaminhando as mensagens para o Fluxo Node-RED, que assina os tópicos de interesse. Uma vez no Node-RED, os dados são processados para múltiplos fins: são exibidos

no *Dashboard* para o tutor, persistidos no Banco de Dados MySQL e, se necessário, disparam alertas via API do Telegram.

4.2.2 Arquitetura de Nuvem e Escalabilidade

Como evolução do protótipo, e aplicando princípios de escalabilidade e segurança, propõe-se uma arquitetura de nuvem *serverless* e desacoplada (Figura 4.3). Esta arquitetura de produção elimina o *broker* público e centraliza toda a comunicação no AWS IoT Core. Nesta arquitetura ideal, todos os componentes (o dispositivo ESP32, o Node-RED e o Lambda) se conectam ao mesmo *endpoint* seguro da AWS, autenticando-se via certificados TLS e sendo autorizados por políticas de IoT.

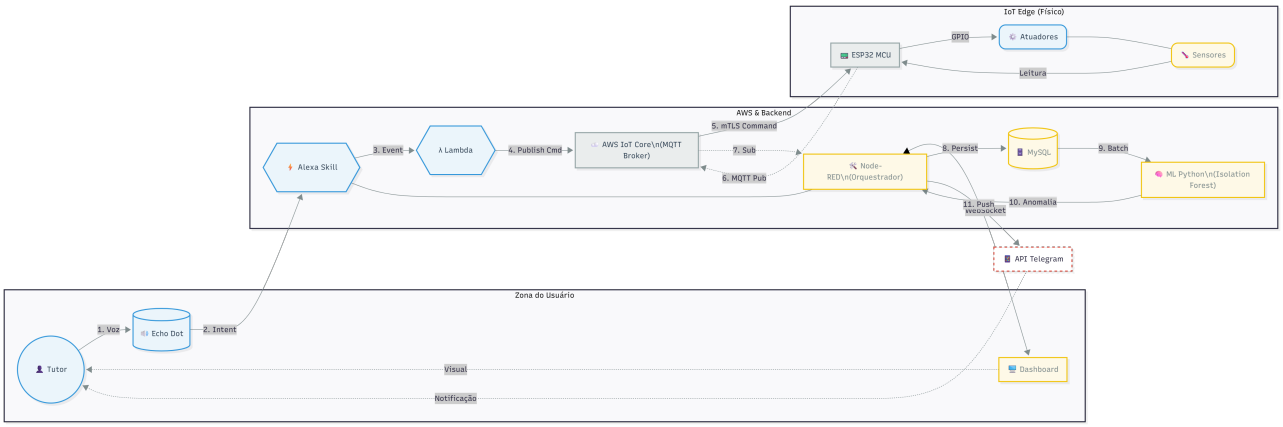

Figura 4.3: Diagrama da Arquitetura Lógica de Nuvem (Produção).

Fonte: Elaborado pelo autor (2025).

Nesta arquitetura, o fluxo de comando é iniciado pela Amazon Alexa. A *Skill* captura a *Intent* de voz e aciona a função AWS Lambda. O Lambda publica o comando MQTT (ex: "ON") diretamente no AWS IoT Core. O Node-RED (para o *dashboard*) e o ESP32 (o dispositivo físico) estariam ambos inscritos no mesmo tópico no AWS IoT Core, recebendo o comando de forma segura e simultânea.

4.2.3 Metodologia de Validação e o Artifício de Simulação

Para validar a arquitetura de nuvem (Capítulo 6) sem o protótipo físico, foi necessário um artifício de simulação que introduziu uma falha de segurança deliberada, mas controlada. A plataforma de simulação Wokwi não possui a capacidade técnica de se autenticar no AWS IoT Core usando certificados TLS. Para contornar essa limitação, o Node-RED foi configurado temporariamente como uma "ponte":

1. O Node-RED se conectou de forma segura ao AWS IoT Core para receber os comandos da Alexa.
2. Ele, então, re-publicou esses comandos no *broker* público e inseguro `test.mosquitto.org`.
3. O simulador Wokwi, por sua vez, conectou-se ao `test.mosquitto.org` para receber os comandos.

Embora imposta por limitações do simulador, esta topologia reflete o padrão de arquitetura *Industrial Edge Gateway*. Neste cenário, o Node-RED atua como o *Gateway* Seguro que intermedeia dispositivos locais restritos (o Wokwi simula um dispositivo legado sem suporte a TLS avançado) e a nuvem segura, realizando o encapsulamento criptográfico necessário para o AWS IoT Core.

4.3 Especificação dos Componentes de Software

A Tabela 4.1 resume as principais ferramentas, plataformas e bibliotecas que constituíram o ambiente de desenvolvimento e simulação do projeto.

Tabela 4.1: Ferramentas e Bibliotecas de Software Utilizadas.

Ferramenta/Biblioteca	**Versão**	**Propósito no Projeto**
Arduino IDE	2.3.2	Desenvolvimento e compilação do firmware
Wokwi	N/A (Online)	Simulação do hardware e do firmware
Node-RED	3.1.0	Orquestração da lógica e dashboard
`PubSubClient.h`	2.8.0	Cliente MQTT para o ESP32
`HX711.h`	0.7.5	Interface com o amplificador do sensor de peso
`DHT.h`	1.4.6	Leitura do sensor de temperatura e umidade
`Servo.h`	1.2.1	Controle do servo motor
Amazon Web Services	N/A	Plataforma de nuvem para IoT Core e Lambda
Alexa Skills Kit	N/A	Plataforma de desenvolvimento de assistente de voz
Python 3.10	N/A	Linguagem para scripts de geração e análise de dados
Scikit-learn	N/A	Biblioteca de Machine Learning para Isolation Forest
Pandas & Matplotlib	N/A	Bibliotecas para manipulação e visualização de dados

Fonte: Elaborado pelo autor (2025).

4.3.1 Componentes Desenvolvidos

- **Firmware do ESP32:** desenvolvido em C++ na Arduino IDE, o *firmware* é o *software* embarcado responsável pela operação autônoma do *hardware*. Sua lógica principal foi estruturada de forma não-bloqueante para garantir a responsividade na gestão de múltiplas tarefas concorrentes: leitura de sensores, controle de atuadores e comunicação via MQTT.

- **Fluxo de Orquestração (Node-RED):** atua como o *middleware* central do sistema. O fluxo desenvolvido contém nós para receber dados via MQTT, um conjunto de nós do módulo `node-red-dashboard` para a interface com o usuário, nós de lógica para o processamento e nós de integração com os serviços de persistência e notificação.

- **Estrutura do Banco de Dados (MySQL):** para a persistência dos dados, foi projetada uma estrutura relacional no MySQL. A tabela principal, denominada `historico`, centraliza todas as leituras periódicas dos sensores, conforme detalhado na Tabela 4.2.

- **Scripts de Análise (Python):** foram desenvolvidos dois *scripts* em Python: (1) para geração de *dataset* de alta fidelidade (`gerar_dados_v2.py`) e (2) para detecção de anomalias (`detectar_anomalia_v2.py`) implementando o modelo *Isolation Forest*.

- **Função *Serverless* (AWS Lambda):** foi desenvolvida uma função em Python (`PetFeeder_TriggerRacao`) que atua como o *endpoint* para a *Skill* da Alexa, responsável por traduzir *Intents* de voz em comandos MQTT.

- **Skill de Voz (Amazon Alexa):** foi configurada uma *Skill* customizada ('PetFeeder Skill') com múltiplas *Intents* (ex: 'DispensarRacaoIntent', 'LigarAguaIntent', 'LigarArIntent') para mapear frases de usuário às ações da função Lambda.

Tabela 4.2: Estrutura da tabela `historico` do banco de dados.

Nome da Coluna	Tipo de Dado	Descrição
`id`	INT, PK, AI	Identificador único da leitura
`data_hora`	TIMESTAMP	Data e hora do registro (DEFAULT CURRENT_TIMESTAMP)
`nivel_agua`	FLOAT	Nível de água na tigela em centímetros (cm)
`peso_agua_reservatorio`	FLOAT	Peso do reservatório de água em gramas (g)
`total_racao_repositorio`	FLOAT	Peso do repositório principal de ração em gramas (g)
`quantidade_racao_recipient`	FLOAT	Peso da ração na tigela do animal em gramas (g)
`temperatura_ambiente`	FLOAT	Temperatura do ambiente em graus Celsius (°C)

Fonte: Elaborado pelo autor (2025).

4.3.2 Plataformas e Serviços de Suporte

- **Broker MQTT (Protótipo):** na fase de prototipagem, foi utilizado o *broker* público `test.mosquitto.org`. Esta escolha permitiu a validação da lógica de comunicação de forma rápida.

- **Broker MQTT (Produção):** na fase de evolução da infraestrutura, o sistema migrou para o AWS IoT Core (ex: `a1avf...amazonaws.com`), um *broker* gerenciado, seguro e escalável, que autentica conexões via certificados TLS.

- **API do Telegram:** as notificações de alerta são enviadas através da API oficial do Telegram, integrada via um nó de contribuição no Node-RED (`node-red-contrib-telegrambot`), configurado com o *token* de um Bot privado.

- **Interface do Tutor (Dashboard):** a interface com o usuário é provida por um *dashboard web* dinâmico, renderizado pelo módulo `node-red-dashboard`. A interface permite a visu-

alização de dados em tempo real e o acionamento de comandos, servindo como o principal ponto de interação do tutor. A Figura 4.4 exemplifica uma das abas do *dashboard*.

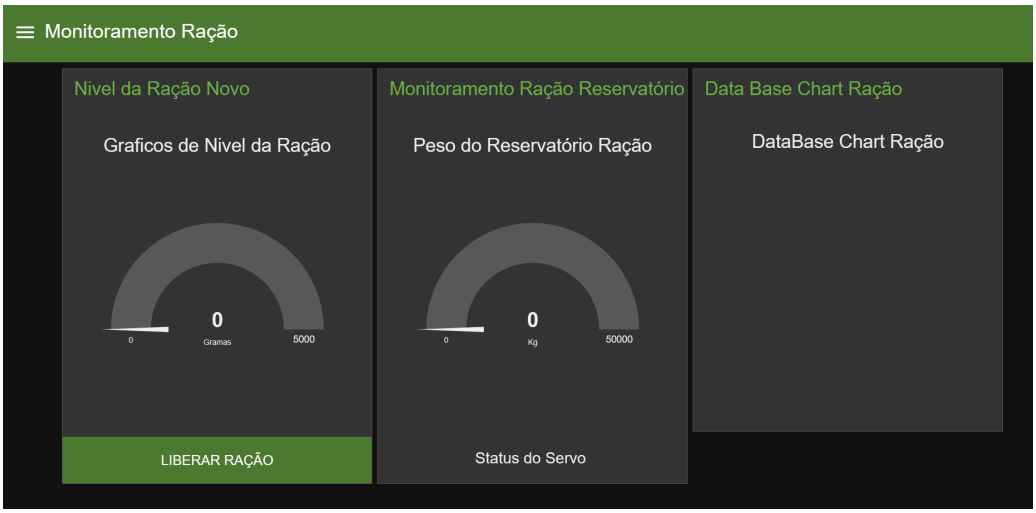

Figura 4.4: Aba de monitoramento de ração no *dashboard web*.

Fonte: Elaborado pelo autor (2025).

Capítulo 5

Resultados e Discussão

Este capítulo apresenta os resultados obtidos com o desenvolvimento e a validação do sistema proposto. A análise dos resultados é dividida em duas etapas complementares que refletem a evolução do projeto: primeiramente, apresenta-se a validação funcional da arquitetura de orquestração e monitoramento em tempo real (*Hardware* e *Middleware*); em seguida, detalha-se a validação da inteligência do sistema através da *pipeline* de análise de dados preditiva.

5.1 Validação Funcional: Monitoramento e Automação

Antes da aplicação de algoritmos de inteligência, a validação primária consistiu em verificar a estabilidade da arquitetura de orquestração proposta no Capítulo 4. Os testes realizados no ambiente de simulação confirmaram a eficácia do protocolo MQTT e do fluxo Node-RED na integração dos componentes.

Os principais resultados funcionais alcançados incluem:

- **Coleta e Transmissão:** O *firmware* do ESP32 realizou a leitura cíclica dos sensores (Temperatura, Umidade e Células de Carga) e transmitiu os pacotes de telemetria sem perda significativa de dados para o *broker*, validando a escolha do protocolo MQTT para comunicação assíncrona.

- **Orquestração Visual:** O *Dashboard* desenvolvido no Node-RED permitiu a visualização

instantânea das variáveis, bem como o acionamento manual remoto dos atuadores (Servo Motor e Válvula), cumprindo o requisito de controle centralizado.

- **Integridade dos Dados:** A persistência no banco de dados MySQL ocorreu de forma estruturada, criando a base histórica necessária para a etapa subsequente de análise preditiva.

Esta etapa validou a infraestrutura básica do sistema, demonstrando que a solução é capaz de gerenciar o ecossistema de cuidado animal de forma reativa e manual, servindo de alicerce para as funcionalidades autônomas.

5.2 Validação da Inteligência Preditiva: Pipeline de Análise de Dados

Com a arquitetura funcional validada, implementou-se o módulo de saúde preditiva para agregar inteligência ao sistema. Do ponto de vista da engenharia, é relevante diferenciar a validação de uma *pipeline* de *software* da validação de um modelo de *Machine Learning* clínico. O objetivo desta etapa, portanto, foi implementar um teste de unidade (*unit test*) para a *pipeline* de dados, provando que a arquitetura de *software* era capaz de: (1) agregar dados históricos do MySQL, (2) treinar um modelo de ML e (3) executar uma detecção de anomalias.

5.2.1 Metodologia de Validação: O Ambiente Controlado

Para validar a arquitetura de análise de dados, foi utilizado um ambiente controlado de alta fidelidade baseado em dados sintéticos. É fundamental esclarecer a natureza e o propósito desta validação para alinhar as expectativas quanto aos resultados.

Validade Científica do Dataset Sintético

Diferente de um estudo clínico veterinário, onde o objetivo seria provar a eficácia médica de um diagnóstico, este trabalho de Sistemas de Informação visa validar a *pipeline* de Engenharia

de Software. O uso de um *dataset* sintético, gerado pelos *scripts* desenvolvidos no projeto, introduz riscos conhecidos, como o viés de confirmação.

Contudo, esta abordagem é cientificamente válida para o escopo de:

1. **Teste de Integração de Dados:** provar que a arquitetura consegue ingerir, armazenar e processar séries temporais longas (90 dias simulados).

2. **Prova de Conceito do Algoritmo:** demonstrar que o algoritmo *Isolation Forest* é tecnicamente capaz de segmentar dados normais de anômalos dentro da topologia de dados proposta, superando abordagens estatísticas simples.

Assim, os resultados a seguir validam a capacidade da arquitetura em detectar anomalias, assumindo que os sensores reais forneceriam dados com padrões semelhantes aos simulados.

Geração dos Cenários (Normal vs. Patológico)

O *dataset* simula 90 dias de operação, divididos em três fases distintas para modelar o surgimento de uma patologia crônica:

- **Período Normal (Dias 1-60):** o animal exibe comportamento saudável, com consumo parametrizado por regras veterinárias (ex: ≈55ml/kg de água). O modelo é treinado exclusivamente com este segmento para estabelecer a "linha de base".

- **Período de Transição (Dias 61-67):** Uma transição gradual de 7 dias onde os padrões de consumo sofrem desvios progressivos.

- **Período Anômalo (Dias 68-90):** o sistema simula sintomas agudos: Polidipsia (aumento de 2,5x na ingestão de água) e Inapetência (redução para 25% da ingestão de ração).

5.2.2 Engenharia de Features e Agregação Temporal

Uma etapa fundamental da *pipeline*, implementada na função `processar_features_diarias`, é a transformação dos dados brutos. Embora o dispositivo IoT gere leituras horárias, as anomalias de saúde crônicas manifestam-se em desvios no padrão de consumo diário total.

Portanto, foi aplicada uma agregação temporal, agrupando os dados horários por dia e calculando o consumo acumulado. Esta agregação transforma o *ruído* horário em um *sinal* diário claro, permitindo que o modelo *Isolation Forest* identifique com precisão mudanças sutis na linha de base comportamental.

5.2.3 Avaliação Quantitativa e Definição de Métricas

Para a avaliação objetiva, estabeleceu-se a convenção padrão onde a classe Positiva (1) representa o estado anômalo. A validação cruzada foi realizada temporalmente, testando o modelo treinado nos primeiros 60 dias contra os 30 dias subsequentes.

A interpretação da Matriz de Confusão para o contexto de monitoramento de saúde animal é definida da seguinte forma:

- **Verdadeiro Positivo (TP):** o animal estava doente e o sistema gerou um alerta corretamente.
- **Verdadeiro Negativo (TN):** o animal estava saudável e o sistema permaneceu em silêncio.
- **Falso Positivo (FP - Alarme Falso):** alerta indevido, causando apenas incômodo.
- **Falso Negativo (FN - Risco Crítico):** falha em detectar a doença, impedindo tratamento precoce.

Comparativo de Desempenho: ML vs. Baseline

Para justificar o custo computacional do *Isolation Forest*, seu desempenho foi comparado a uma *Baseline* determinística (Regra de Limiar Fixo). A Tabela 5.1 apresenta os resultados comparativos.

Tabela 5.1: Comparativo de Métricas: Isolation Forest vs. Baseline (Regra Fixa).

Métrica	Isolation Forest	Baseline	Interpretação Clínica
Recall (Sensibilidade)	**96.67%**	100.00%	Capacidade de detectar a doença. O Baseline detectou tudo, mas com custo excessivo de alarmes falsos.
Precisão	**82.86%**	33.71%	Confiabilidade do alerta. O Baseline erra 2 em cada 3 alertas.
F1-Score	**89.23%**	50.42%	Média harmônica. O ML apresenta o melhor equilíbrio geral.
Falsos Negativos	**1 dia**	0 dias	O ML falhou em apenas 1 dia do período anômalo.

Fonte: Elaborado pelo autor (2025).

A análise demonstra a superioridade do modelo de ML. Embora a *Baseline* tenha obtido 100% de *Recall*, sua baixa precisão (33%) a torna inviável na prática. O *Isolation Forest*, com *Recall* de 96.67% e *F1-Score* de 89.23%, oferece o equilíbrio robusto necessário para um produto de monitoramento real.

5.2.4 Análise Visual da Detecção

A eficácia da detecção pode ser visualizada na Figura 5.1, que plota o consumo diário de ração e água, destacando os pontos classificados como anômalos.

A análise visual confirma que o modelo aprendeu corretamente a linha de base (azul) e identificou quase todos os dias do período anômalo, validando a *pipeline* de Engenharia de Software proposta. O sistema provou ser capaz de coletar, processar e analisar dados para gerar *insights* de saúde.

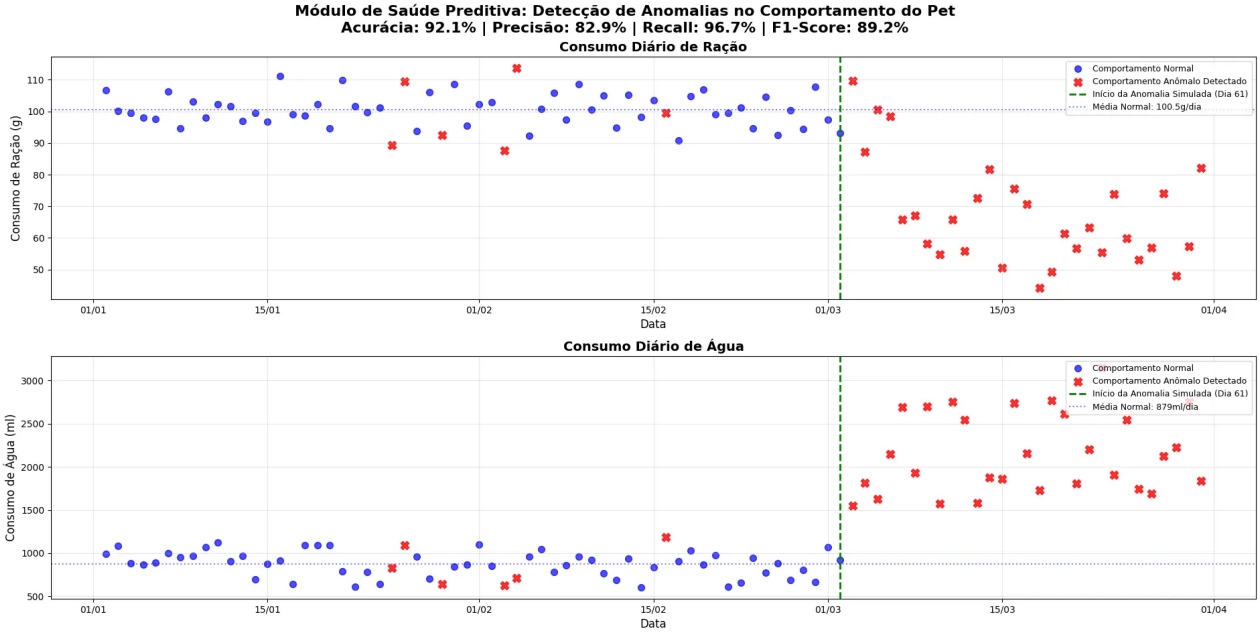

Figura 5.1: Resultado da validação da *pipeline* de ML. Os pontos em vermelho indicam anomalias detectadas corretamente após a linha verde (início da doença).

Fonte: Elaborado pelo autor (2025).

Capítulo 6

Evolução da Infraestrutura: Nuvem e Interfaces de Voz

Este capítulo detalha a etapa final de desenvolvimento do sistema: a transição da arquitetura de protótipo local para uma solução de nuvem profissional, segura e escalável, integrando o controle por assistentes de voz. Esta implementação visa atender aos requisitos não funcionais de interoperabilidade e segurança, aplicando princípios robustos de Engenharia de Software.

6.1 Transição Arquitetural: Do Protótipo à Nuvem

A arquitetura inicial (apresentada no Capítulo 4 como "Protótipo"), embora funcional para validação de bancada, dependia de um *broker* MQTT público (`test.mosquitto.org`) (Amazon Web Services, 2016). Esta abordagem apresentava riscos significativos de segurança (falta de criptografia) e nenhuma garantia de SLA (*Service Level Agreement*) para escalabilidade.

A evolução para a arquitetura de produção migrou a camada de comando e controle para a Amazon Web Services (AWS), adotando um modelo *serverless* (sem servidor) e orientado a eventos. Os componentes desta nova infraestrutura são detalhados a seguir.

6.2 Implementação dos Componentes de Nuvem (AWS)

A concretização da arquitetura de nuvem demandou a seleção e configuração de serviços gerenciados que garantissem segurança, escalabilidade e alta disponibilidade, eliminando a sobrecarga operacional de servidores tradicionais. A estratégia de implementação baseou-se no paradigma *Serverless*, utilizando serviços nativos da Amazon Web Services para compor as camadas de comunicação e processamento. A seguir, são detalhados os componentes fundamentais configurados para suportar a nova topologia do sistema.

6.2.1 AWS IoT Core: O Broker MQTT Seguro

O primeiro passo para a profissionalização da comunicação foi a substituição do *broker* público por um *endpoint* gerenciado no AWS IoT Core (ex: `a1avf...-ats...`). Diferente da solução anterior, o AWS IoT Core impõe autenticação mútua TLS (*Transport Layer Security*).

A conexão foi validada no Node-RED através da configuração de certificados de segurança X.509 (Certificado do Dispositivo, Chave Privada e Certificado CA). Para autorizar a comunicação, uma política de IoT estrita foi implementada (Política `PetFeeder_Policy_FINAL`), garantindo o princípio do "menor privilégio":

- **Conexão:** Permitida apenas para IDs de clientes autorizados (‘iot:Connect‘).
- **Tópicos:** Publicação e assinatura restritas à hierarquia do projeto (‘topic/canilGuilherme/*‘).

6.2.2 AWS Lambda: O "Cérebro" Serverless

Para viabilizar a integração com comandos de voz, foi necessário um *middleware* capaz de traduzir intenções da interface de voz em protocolos de IoT. Para isso, foi implementada uma função AWS Lambda (`PetFeeder_TriggerRacao`) em Python.

O Código 6.2.1 apresenta a lógica final da função, que demonstra a escalabilidade do sistema: ela recebe um evento JSON da Alexa, identifica a *Intent* (intenção) do usuário e publica o *payload* MQTT correto no tópico apropriado do AWS IoT Core.

```
import json
import boto3

iot_client = boto3.client('iot-data', region_name='us-east-1')

def lambda_handler(event, context):
    mensagem_de_voz = "Desculpe, não entendi o comando."
    should_publish = False
    MQTT_TOPIC = ""
    MQTT_PAYLOAD = ""

    try:
        intent_name = event['request']['intent']['name']
        print(f"Recebida Intent: {intent_name}")

        if intent_name == "DispensarRacaoIntent":
            MQTT_TOPIC = "canilGuilherme/racao/dispensador/comando"
            MQTT_PAYLOAD = "LIBERAR"
            mensagem_de_voz = "Ok, enviando comando para alimentar o
               pet."
            should_publish = True

        elif intent_name == "LigarAguaIntent":
            MQTT_TOPIC = "canilGuilherme/agua/valvula/comando"
            MQTT_PAYLOAD = "manual_on"
            mensagem_de_voz = "Ok, ligando a válvula de água."
            should_publish = True

        elif intent_name == "LigarArIntent":
```

```
            MQTT_TOPIC = "canilGuilherme/ambiente/ar/comando"
            MQTT_PAYLOAD = "ON"
            mensagem_de_voz = "Ok, ligando o ar condicionado."
            should_publish = True

        elif intent_name == "DesligarArIntent":
            MQTT_TOPIC = "canilGuilherme/ambiente/ar/comando"
            MQTT_PAYLOAD = "OFF"
            mensagem_de_voz = "Ok, desligando o ar condicionado."
            should_publish = True

        elif intent_name == "ArAutomaticoIntent":
            MQTT_TOPIC = "canilGuilherme/ambiente/ar/comando"
            MQTT_PAYLOAD = "AUTO"
            mensagem_de_voz = "Ok, colocando o ar em modo automático."
            should_publish = True

        if should_publish:
            print(f"Publicando '{MQTT_PAYLOAD}' no tópico: {MQTT_TOPIC
                }...")
            iot_client.publish(
                topic=MQTT_TOPIC, qos=1, payload=MQTT_PAYLOAD
            )
            print("Mensagem publicada com sucesso!")

    return {
            "version": "1.0",
            "response": {
                "outputSpeech": {"type": "PlainText", "text":
                    mensagem_de_voz},
```

```
                "shouldEndSession": True
            }
        }
    except Exception as e:
        # Tratamento de erros omitido para brevidade
```

Código 6.2.1: Código-fonte da função AWS Lambda (`lambda_function.py`), demonstrando o roteamento de *Intents* para tópicos MQTT. Fonte: Elaborado pelo autor (2025).

6.2.3 Alexa Skills Kit: A Interface de Voz

A interface de usuário natural (NUI) foi implementada no *Alexa Developer Console*. Foi criada uma *Skill* customizada (`PetFeeder Skill`) com o nome de invocação "*pet feeder*".

Visando validar a flexibilidade do sistema, a *Skill* foi configurada com múltiplas *Intents* (ex: `DispensarRacaoIntent`, `LigarArIntent`). Cada *Intent* foi treinada com frases de exemplo variadas (*utterances*), garantindo que o acionamento do *endpoint* AWS Lambda ocorra de forma fluida, validando o padrão de projeto de entrada única e processamento centralizado.

6.3 Validação da Integração End-to-End

O teste de integração final consistiu em validar o fluxo completo de dados: desde o comando de voz em um dispositivo físico (Alexa Echo Dot) até o acionamento efetivo do atuador no ambiente simulado.

Para viabilizar este teste sem o *hardware* físico final, o Node-RED foi configurado como um *Secure Gateway*, ouvindo os comandos criptografados do AWS IoT Core e repassando-os para o ambiente de simulação Wokwi. Os resultados, capturados no painel de depuração (Figura 6.1), confirmam o sucesso da integração.

A chegada dos *payloads* corretos e a resposta audível da assistente virtual (ex: "Ok, ligando o ar condicionado") validam a implementação de uma arquitetura de nuvem robusta, atendendo aos requisitos de latência e confiabilidade definidos no projeto.

```
5/11/2025, 12:01:44 PM - ... (debug 13)
canilGuilherme/racao/dispensador/comando : msg.payload : string[7]
"LIBERAR"

5/11/2025, 12:05:10 PM - ... (debug 14)
canilGuilherme/agua/valvula/comando : msg.payload : string[9]
"manual_on"

5/11/2025, 12:07:30 PM - ... (debug 15)
canilGuilherme/ambiente/ar/comando : msg.payload : string[2]
"ON"

5/11/2025, 12:07:45 PM - ... (debug 15)
canilGuilherme/ambiente/ar/comando : msg.payload : string[3]
"OFF"

5/11/2025, 12:07:55 PM - ... (debug 15)
canilGuilherme/ambiente/ar/comando : msg.payload : string[4]
"AUTO"
```

Figura 6.1: Logs de depuração do Node-RED confirmando o recebimento dos comandos MQTT originados pela Alexa.

Fonte: Elaborado pelo autor (2025).

6.4 Análise Crítica: Segurança e Viabilidade Econômica

A transição de um experimento acadêmico para um produto viável exige a mitigação de riscos de segurança e a análise de custos, conforme discutido a seguir.

6.4.1 Riscos de Segurança da Topologia de Simulação

É mandatório ressaltar que a utilização de uma "ponte" no Node-RED (usada apenas para conectar o AWS IoT Core ao simulador Wokwi) introduz vulnerabilidades que não devem existir na versão física do produto:

- **Falta de Criptografia na Ponta:** A comunicação final com o simulador ocorre em texto plano.
- **Vulnerabilidade a Injeção:** O *broker* público do simulador não possui autenticação forte.

Para a implementação final em *hardware*, a mitigação definitiva é a eliminação desta ponte e a

implementação de mTLS (*Mutual TLS*) diretamente no microcontrolador ESP32, conectando-o exclusivamente ao *endpoint* da AWS.

6.4.2 Estimativa de Custos Operacionais (AWS)

A sustentabilidade econômica do projeto em escala depende do modelo de precificação da nuvem (*Pay-as-you-go*). Considerando um cenário de uso residencial típico:

- **AWS IoT Core:** Com envio de telemetria a cada 10 minutos, o volume mensal permanece abaixo de 50.000 mensagens, amplamente coberto pelo nível gratuito (*Free Tier*), que oferece até 2,25 milhões de mensagens.
- **AWS Lambda:** O consumo de tempo de computação para comandos de voz esporádicos é desprezível frente ao limite gratuito de 400.000 GB-segundos.

Conclui-se que, para uso doméstico ou fase de prototipagem (até $\approx$50 dispositivos), o custo de infraestrutura é virtualmente nulo, validando a viabilidade econômica da solução arquitetural proposta.

Capítulo 7

Conclusão

7.1 Considerações Finais

Ao término deste trabalho, conclui-se que os objetivos de pesquisa foram alcançados através da validação de uma arquitetura de referência para sistemas de IoT. Diferente de uma abordagem puramente de engenharia de produto, o foco deste estudo recaiu sobre a integração de sistemas e a engenharia de *software* necessárias para orquestrar um ambiente fragmentado.

O ciclo de desenvolvimento demonstrou que a utilização do Node-RED como *middleware* centralizador, associada a serviços de nuvem gerenciados (AWS), oferece uma solução viável para superar a interoperabilidade limitada de dispositivos comerciais. A validação sintética do módulo de *Machine Learning* (Capítulo 5) comprovou que, havendo dados confiáveis, a arquitetura proposta possui a capacidade computacional e lógica para processar inferências de saúde, embora a validação clínica real permaneça como um passo futuro necessário.

Os testes de integração com a Amazon Alexa validaram a flexibilidade da arquitetura *serverless*, demonstrando que o desacoplamento via MQTT permite a expansão de interfaces (voz, *dashboard*, *mobile*) sem refatoração do núcleo do sistema.

7.2 Contribuições do Trabalho

Como principais contribuições acadêmicas e técnicas deste estudo, destacam-se:

- **Proposta de Arquitetura de Orquestração:** A definição de uma topologia que utiliza o Node-RED não apenas como visualizador, mas como *Gateway* lógico para integrar protocolos distintos e serviços de nuvem.
- **Análise Comparativa de Sensoriamento:** O levantamento crítico das limitações de sensores ultrassônicos para medição de líquidos e a proposição de alternativas baseadas em Tempo de Voo (ToF) para maior precisão.
- **Pipeline de Validação Sintética:** O desenvolvimento de uma metodologia de geração de dados controlados para validar algoritmos de detecção de anomalias em IoT, permitindo testes de *software* sem a dependência imediata de longos experimentos biológicos.
- **Integração Segura de Voz:** A documentação prática da integração entre *Alexa Skills Kit*, *AWS Lambda* e *AWS IoT Core*, servindo como referência para futuros projetos de automação residencial no curso.

7.3 Trabalhos Futuros

Reconhecendo as limitações experimentais detalhadas no Capítulo 3, sugere-se a seguinte ordem de prioridade para a evolução da pesquisa:

- **Validação Física e Calibração:** A montagem do protótipo em ambiente real para coletar métricas de ruído dos sensores e validar a durabilidade mecânica dos atuadores.
- **Implementação de IA na Borda (*Edge AI*):** Migração do modelo de detecção de anomalias da nuvem para o microcontrolador (*TinyML*), reduzindo a latência e a dependência de conectividade, possivelmente utilizando um ESP32-S3.
- **Estudo de Eficiência Energética:** Análise do consumo de energia do sistema e implementação de rotinas de *Deep Sleep* para viabilizar a operação por baterias.
- **Coleta de Dados Reais:** Substituição do *dataset* sintético por dados coletados de animais reais, permitindo o refinamento dos limiares do algoritmo *Isolation Forest*.

Referências Bibliográficas

AL-HAWAWREH, M. et al. Mqtt-based intrusion detection system for iot networks. *Sensors*, v. 24, n. 6, p. 1954, 2024.

ALVES, G. L. *AUTOFEEDER: Alimentador automático para animais domésticos de pequeno porte*. Lages, SC, Brasil, 2020.

Amazon Web Services. *Using Alexa Skills Kit and AWS IoT to Voice-Control Connected Devices*. 2016. <https://developer.amazon.com/blogs/alexa/post/Tx3828JHC7O9GZ9/using-alexa-skills-kit-and-aws-iot-to-voice-control-connected-devices>. Acessado em 14 de junho de 2025.

Amazon Web Services. *O que é o AWS IoT Core? - Guia do Desenvolvedor do AWS IoT Core*. 2024. <https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html>. Acessado em 05 de novembro de 2025.

Amazon Web Services. *O que é o AWS Lambda? - Guia do Desenvolvedor do AWS Lambda*. 2024. <https://docs.aws.amazon.com/lambda/latest/developerguide/what-is-lambda.html>. Acessado em 05 de novembro de 2025.

Amazon Web Services. *What is MQTT?* 2024. <https://aws.amazon.com/what-is/mqtt/>. Acessado em 14 de junho de 2025.

Apifornia. *Node-RED: a powerful tool for integration and automation*. 2023. <https://blog.apifornia.com/node-red/>. Acessado em 14 de junho de 2025.

BANHAZI, T. M. et al. Precision livestock farming: An overview of recent developments and future prospects. *Animal Production Science*, v. 62, n. 14, p. 1335–1347, 2022.

Bevywise. *MQTT QoS Levels Explained with Examples for Usage*. 2023. <https://www.bevywise.com/blog/mqtt-qos-level-use/>. Acessado em 14 de junho de 2025.

BOQU Instrument. *Exploring the Limitations of Ultrasonic Water Level Sensors*. 2023. <https://www.boquinstrument.com/exploring-the-limitations-of-ultrasonic-water-level-sensors.html>. Acessado em 14 de junho de 2025.

BOQU Instrument. *The Limitations of Ultrasonic Water Level Sensors in Measuring Foam and Bubbles*. 2024. <https://www.boquinstrument.com/the-limitations-of-ultrasonic-water-level-sensors-in-measuring-foam-and-bubbles.html>. Acessado em 14 de junho de 2025.

Cedalo. *Understanding MQTT QoS (Quality of Service)*. 2023. <https://cedalo.com/blog/understanding-mqtt-qos/>. Acessado em 14 de junho de 2025.

Exforsys Inc. *The Evolutionary Prototyping Model*. 2013. <https://www.exforsys.com/career-center/project-management-life-cycle/the-evolutionary-prototyping-model.html>. Acessado em 14 de junho de 2025.

GeeksforGeeks. *Software Engineering | Prototyping Model*. 2024. <https://www.geeksforgeeks.org/software-engineering-prototyping-model/>. Acessado em 14 de junho de 2025.

Grand View Research. *Smart Pet Feeder Market Size, Share & Trends Analysis Report*. 2024. <https://www.grandviewresearch.com/industry-analysis/smart-pet-feeder-market-report>. Acessado em 14 de junho de 2025.

ILVO. *Dossier Precision livestock farming*. 2023. <https://ilvo.vlaanderen.be/en/dossiers/precision-livestock-farming>. Acessado em 14 de junho de 2025.

IoT-Solution. *Why is Node-RED so popular in the IoT field?* 2024. <https://www.iot-solution.com/article/node-red-embedded-computer-i00437i1.html>. Acessado em 14 de junho de 2025.

JACOMINI, R. d. S. *Sistema Embarcado Aplicado a Automação na Alimentação de Pets: ML e IoT*. Dissertação (Dissertação de Mestrado) — Universidade Estadual Paulista "Júlio de Mesquita Filho", Sorocaba, SP, Brasil, 2024.

LI, J. et al. Wild animals detection based on yolov5. *Journal of Physics: Conference Series*, v. 2584, p. 012023, 2023.

LIU, F. T.; TING, K. M.; ZHOU, Z.-H. Isolation forest. In: *2008 Eighth IEEE International Conference on Data Mining*. [S.l.: s.n.], 2008. p. 413–422.

MASSON, S. et al. An iot system for monitoring environmental conditions on livestock farms. *Animals*, v. 14, n. 5, p. 644, 2024.

Motius. *Embedded Systems: How to Prototype a Computer That Nobody Cares About*. 2020. <https://www.motius.com/post/embedded-systems-how-to-prototype-a-computer-that-nobody-cares-about>. Acessado em 14 de junho de 2025.

NEETHIRAJAN, S. Recent advances in wearable sensors for animal health management. *Sensing and Bio-Sensing Research*, v. 12, p. 15–29, 2017.

O.E.C.-A. et al. Design and development of a smart pet feeder with iot and deep learning. *Engineering Proceedings*, v. 82, n. 1, p. 63, 2024.

PARTHASARATHI, V. et al. Internet of things in animal healthcare (iotah): Review of recent advancements in architecture, sensing technologies, and real-time monitoring. *Journal of Critical Reviews*, v. 7, n. 13, p. 474–483, 2020.

PCBasic. *DHT11 vs DHT22: Which Humidity Sensor is Right for You?* 2024. <https://www.pcbasic.com/blog/dht11_vs_dht22.html>. Acessado em 14 de junho de 2025.

PUSR. *What can the node-red function of edge computing gateway help us do?* 2024. <https://www.pusr.com/blog/What-can-the-node-red-function-of-edge-computing-gateway-help-us-do>. Acessado em 14 de junho de 2025.

qbee.io. *Node-RED is perfect for industrial low-code development*. 2023. <https://qbee.io/node-red-is-perfect-for-industrial-low-code-development/>. Acessado em 14 de junho de 2025.

Quarktwin. *Unraveling the Mysteries of the HX711: A Comprehensive Guide*. 2024. <https://www.quarktwin.com/blogs/undefined/unraveling-the-mysteries-of-the-hx711-a-comprehensive-guide/277>. Acessado em 14 de junho de 2025.

Random Nerd Tutorials. *DHT11 vs DHT22 vs LM35 vs DS18B20 vs BME280 vs BMP180*. 2023. <https://randomnerdtutorials.com/dht11-vs-dht22-vs-lm35-vs-ds18b20-vs-bme280-vs-bmp180/>. Acessado em 14 de junho de 2025.

Renke. *What is Ultrasonic Sensor? How does it work?* 2023. <https://www.renkeer.com/what-is-ultrasonic-sensor/>. Acessado em 14 de junho de 2025.

RODRIGUES, J. M. d. L.; JúNIOR, I. d. S. Q. *PROJETO IOT APLICADO À CONSTRUÇÃO DE UM ALIMENTADOR AUTOMÁTICO PARA ANIMAIS DOMÉSTICOS*. Mossoró, RN, Brasil, 2019.

RunTime Recruitment. *Best Practices in Rapid Prototyping for Embedded Systems*. 2024. <https://runtimerec.com/best-practices-in-rapid-prototyping/>. Acessado em 14 de junho de 2025.

Seeed Studio. *DHT11 vs DHT22 (AM2302) – Which Temperature & Humidity Sensor Should You Use?* 2020. <https://www.seeedstudio.com/blog/2020/04/20/dht11-vs-dht22-am2302-which-temperature-humidity-sensor-should-you-use/>. Acessado em 14 de junho de 2025.

SparkFun Electronics. *Load Cell Amplifier HX711 Breakout Hookup Guide*. 2023. <https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide/all>. Acessado em 14 de junho de 2025.

TAVARES, G. et al. Precision livestock farming: A systematic review of the literature. *Computers and Electronics in Agriculture*, v. 191, p. 106511, 2021.

ThingsBoard. *MQTT Quality of Service (QoS) levels explained*. 2024. <https://thingsboard.io/docs/mqtt-broker/user-guide/qos/>. Acessado em 14 de junho de 2025.

ZIMMER, A. et al. circat: Cat-centered smart home system and veterinary complementary devices. In: *Proceedings of the 2023 ACM International Conference on Animal-Computer Interaction*. [S.l.: s.n.], 2023.