UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GABRIEL HENRIQUE TESTA

SISTEMA DE AUTOMAÇÃO E CONTROLE PARA CULTIVO PROTEGIDO UTILIZANDO WEB SERVICES E INTEGRAÇÃO DE SERVIÇOS DE CLOUD COMPUTING DA AZURE

CAMPO MOURÃO

2023

GABRIEL HENRIQUE TESTA

SISTEMA DE AUTOMAÇÃO E CONTROLE PARA CULTIVO PROTEGIDO UTILIZANDO WEB SERVICES E INTEGRAÇÃO DE SERVIÇOS DE CLOUD COMPUTING DA AZURE

Automation and Control System for Protected Cultivation Using Web Services and Integration of Azure Cloud Computing Services

Dissertação apresentada como requisito para obtenção do título de Mestre em Inovações Tecnológicas do Programa de Mestrado em Inovações Tecnológicas da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Eduardo Giometti Bertogna.

CAMPO MOURÃO

2023

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Campo Mourão

GABRIEL HENRIQUE TESTA

SISTEMA DE AUTOMAÇÃO E CONTROLE PARA CULTIVO PROTEGIDO UTILIZANDO WEB SERVICES E INTEGRAÇÃO DE SERVIÇOS DE CLOUD COMPUTING DA AZURE

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Inovações Tecnológicas da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Inovações Tecnológicas.

Data de aprovação: 14 de Agosto de 2023

Dr. Eduardo Giometti Bertogna, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Glaucio Pedro De Alcantara, Doutorado - Universidade Estadual de Maringá (Uem)

Dr. Roberto Ribeiro Neli, Doutorado - Universidade Tecnológica Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 20/09/2023.

AGRADECIMENTOS

Meus sinceros agradecimentos iniciais são direcionados à minha família, a base sólida sobre a qual construí cada etapa desta jornada acadêmica. Leonel Fernando Testa, Shillia da Silva Moreira e Anne Caroline Testa, vocês se tornaram meu refúgio e minha inspiração, sempre prontos para prover palavras de motivação, oferecer apoio nos momentos de incertezas e celebrar cada conquista obtida. Este diploma, por isso, pertence tanto a vocês quanto a mim.

Gostaria de expressar minha profunda gratidão à Lais Marcela Zocatelli Biasotto, que transcendeu o papel de uma simples amiga. Lais, você esteve comigo em cada passo, compartilhando seu tempo, sabedoria e competência quando mais precisei. Seu suporte constante, seus conselhos valiosos e sua amizade genuína foram fatores decisivos para a conclusão deste trabalho. O valor da sua contribuição para esta conquista é inigualável e sempre será recordado.

Agradeço também à minha amada Ana Cláudia Martins, que foi minha companheira resiliente ao longo deste percurso. Ana, seu amor, ternura e apoio incessante foram a força motriz que me proporcionou energia para superar cada desafio dessa jornada. A paciência e compreensão que demonstrou, mesmo durante os momentos mais tensos, foram recursos inestimáveis. Sua presença constante me lembrou diariamente da razão de percorrer este caminho.

Finalmente, mas certamente não menos importante, expresso minha gratidão ao Professor Doutor Eduardo Giometti Bertogna. Sua orientação exemplar, paciência infinita e dedicação fervorosa foram o farol que iluminou meu caminho nesta pesquisa. Sua sabedoria e experiência foram fontes de inspiração e crescimento, tanto acadêmico quanto pessoal. Suas críticas construtivas, conselhos acurados e constante estímulo auxiliaram na formação deste trabalho e na minha evolução como pesquisador.

Este trabalho é a evidência tangível do suporte, do amor e da dedicação que cada um de vocês ofereceu. É o símbolo da força que nasce do apoio de pessoas extraordinárias. Este trabalho será sempre dedicado a vocês. Agradeço por terem caminhado ao meu lado e feito parte desta jornada memorável.

“Reze como se tudo dependesse de Deus. Aja como se tudo dependesse de você”

RESUMO

Neste trabalho, propõe-se a implementação de uma infraestrutura capaz de acomodar sensores de medição climática ambiental no interior de uma estufa para cultivo de plantas, oferecendo a possibilidade de monitoramento e controle remoto por meio de dispositivos conectados à internet, como *smartphones* e computadores. O objetivo é adquirir, registrar, analisar e enviar os dados de todas as variáveis medidas com baixa latência para uma infraestrutura de nuvem, onde os dados poderão ser acessados por APIs ou aplicativos móveis, além de fornecer uma API para envio de comandos e programações automáticas de parametrização climática. Para validação do modelo, foram feitos testes de performance para atestar o pleno funcionamento da arquitetura, transitando dados desde de a sua concepção nos sensores, transitando por todo o pipeline de VPNs, Gateways cloud, filas de processamento e servidores de processamento de dados até o banco de dados, e adicionalmente transitando os dados no sentido oposto para envio de comandos para os dispositivos locais tendo como origem a internet. Devido ao mercado altamente competitivo, os produtores buscam o uso de tecnologias disponíveis para aumentar a produtividade e garantir sua permanência no mercado. Atualmente, a integração com tecnologias de computação em nuvem tem revolucionado a forma como compreendemos aplicações que antes não possuíam conectividade. Este trabalho visa integrar três áreas do conhecimento: agricultura, eletrônica e tecnologia da informação, aplicando conceitos emergentes como Internet das Coisas, *Big Data* e *Business Intelligence*, além de técnicas de computação atuais. Além de fornecer a infraestrutura unitária, este estudo também se propõe a construir uma estrutura capaz de adquirir uma grande quantidade de dados de diversos clientes, aplicando conceitos de segurança da informação para garantir que os dados sejam tempestivos e seguros. Adicionalmente, o projeto visa proporcionar diversas possibilidades para futuras melhorias e entregar um produto altamente profissional.

Palavras-chave: redes mesh; cloud services; estufas de plantio; API.

ABSTRACT

In this study, we propose the implementation of an infrastructure capable of accommodating environmental climate measurement sensors within a greenhouse for plant cultivation, providing the possibility of remote monitoring and control through internet-connected devices, such as smartphones and computers. The aim is to acquire, record, analyze, and send data from all measured variables with low latency to a cloud infrastructure, where the data can be accessed by APIs or mobile applications, as well as providing an API for sending commands and automatic climate parameterization programming. To validate the model, performance tests were conducted to ensure the full functionality of the architecture. This involved transferring data from its inception in sensors, passing through the entire VPN pipeline, cloud gateways, processing queues, and data processing servers to the database. Data was also transferred in the opposite direction, enabling it to send commands to local devices from the internet. Due to the highly competitive market, producers seek the use of available technologies to increase productivity and ensure their presence in the market. Currently, the integration of cloud computing technologies has revolutionized the way we understand applications that previously had no connectivity. This work aims to integrate three areas of knowledge: agriculture, electronics, and information technology, applying emerging concepts such as the Internet of Things, Big Data, and Business Intelligence, in addition to current computing techniques. Besides providing the unitary infrastructure, this study also proposes to construct a structure capable of acquiring a large amount of data from various clients, applying information security concepts to ensure that the data is timely and secure. Additionally, the project aims to offer numerous possibilities for future improvements and deliver a highly professional product.

Keywords: mesh network; cloud services; greenhouse; API.

LISTA DE ILUSTRAÇÕES

LISTA DE FLUXOGRAMAS

LISTA DE ABREVIATURAS

SN	Sensor Node
SMCD	Server Mesh Control Device
CCSD	Client Control System Device
PMN	Private Mesh Network
VPN	Virtual Private Network

LISTA DE SIGLAS

IOT	Internet of Things
API	Application Programming Interface
REST	Representational State Transfer
VPN	Virtual Private Network
TLS	Transport Layer Security
SSL	Secure Sockets Layer
IP	Internet Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ACID	Atomicity, Consistency, Isolation, Durability
SQL	Structured Query Language
JSON	JavaScript Object Notation
WLAN	Wireless Local Network
DPS	Device Provisioning Service
VM	Virtual Machine

SUMÁRIO

1 INTRODUÇÃO

1.1 Justificativa

As estufas agrícolas têm sido utilizadas por séculos essencialmente como meio de proteção das plantas contra climas extremos, o que tornou possível criar melhores condições de crescimento das culturas devido à manutenção do ambiente interno em condições adequadas se comparado ao ambiente externo (CRITTEN; BAYLEY, 2002, p. 1-22). Além disto, o ambiente controlado no interior destas estufas, possibilita, para as plantas em regiões com longos períodos de seca, um fornecimento constante de água, com o seu uso de modo racional, bem como, a proteção contra insetos, doenças e ventos fortes (BAILEY, 2002, p. 2-5).

A operação de estufas agrícolas dotadas de monitoramento e controle de seus parâmetros ambientais (umidade relativa do ar, nível de luz, e luminosidade), tem como conseqüência um custo mais elevado de implantação, operação e manutenção se comparado ao cultivo no campo, porém, devido à maior produtividade alcançada, estes custos adicionais são facilmente contornados e a qualidade dos produtos superior (SHAMSHIRI; ISHAK, 2013, p. 176-183).

Sensores e componentes eletrônicos de baixo custo e baixo consumo de energia, serviços de comunicação cada vez mais disponíveis, processamento de dados móvel e aplicações, juntamente com os avanços tecnológicos no *design* das estruturas das estufas impulsionaram a agricultura de cultivo controlado, tornando-a bastante atrativa (SHAMSHIRI; KALANTARI, 2018, p. 1-22).

Além destes pontos mencionados acima, com o retorno da pauta ambiental em decorrência das mudanças climáticas, novas metodologias de operação das estufas surgiram, em especial com foco na reutilização da água do sistema e também na microgeração de energia elétrica, seja eólica ou através de painéis solares (CUCE, 2016, p. 34-59), tendo, portanto, perspectivas de serem as estufas com ambiente controlado, uma tecnologia altamente demandada com a possibilidade real de haver, cada vez mais, climas extremos.

1.2 Objetivos Gerais

A delimitação do tema para a presente dissertação de mestrado consiste em implementação de um sistema de envio de dados de *big data* em estufas agrícolas, considerando a infraestrutura *cloud* e o desenvolvimento de uma API (*Application Programming Interface*) para um aplicativo móvel.

1.3 Objetivos Específicos

- Identificar os principais sensores e atuadores necessários para monitorar e controlar as condições ambientais das estufas agrícolas, analisando-se suas características, funcionalidades e eficiências.
- Projetar e implementar uma infraestrutura *cloud* escalável e segura para armazenar, processar e analisar os dados coletados pelos sensores, garantindo a integridade e a disponibilidade das informações.
- Desenvolver uma API utilizável em aplicativos móveis para permitir o gerenciamento remoto das estufas agrícolas, facilitando o envio de comandos da nuvem para o controlador dos atuadores e a visualização das informações em tempo real.
- Integrar a infraestrutura cloud e a API, estabelecendo um sistema de comunicação eficiente e seguro entre as duas partes, possibilitando uma solução completa de monitoramento e controle das estufas.
- Avaliar a eficiência e a eficácia do sistema proposto em um cenário de stress de envio de dados, analisando seu impacto na consistência das informações recebidas pela infraestrutura.
- Propor melhorias e recomendações para futuras pesquisas e desenvolvimento do sistema, com base nos resultados obtidos e nas limitações identificadas durante a implementação e avaliação do projeto.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Impacto ambiental na economia brasileira

Ao longo dos últimos anos, pode-se observar uma crescente da pauta ambiental. Dentre os principais assuntos tem se popularizado o termo GEE (Gases do Efeito Estufa), que consistem de gases compostos que retêm a radiação solar na atmosfera do planeta. Segundo (ALLEN *et at.*, 2023), estudos apontam que os GEE são responsáveis pelo aumento de 1ºC na temperatura mundial nos últimos anos, em média; ao passo que no Brasil, o aumento foi de 1,5ºC. O estudo ainda revela que a concentração de gases como dióxido de carbono, metano e óxido nitroso, na atmosfera, encontra-se no nível mais elevado dos últimos 800 mil anos.

O estudo de Almeida (2016) aponta que a principal contribuição do Brasil com a mudança climática tem origem no desmatamento e na mecanização da agricultura predatória, onde o aumento da produtividade agrícola se dá pela derrubada de florestas para dar espaço à novas áreas de cultivo. Para Araújo (2018), uma das principais desvantagens referentes ao desenvolvimento da agricultura é a poluição gerada em decorrência de sua atividade que segue o modelo produtivo adotado devido à revolução industrial, que estimularam o crescimento e o desenvolvimento da indústria e dos serviços de infraestrutura no campo. Em alguns locais, por exemplo, o crescimento da atividade agrícola coincidiu com o crescimento de pequenas indústrias prestadoras de serviço da região. Em sua maioria, essas pequenas empresas fornecem peças para o maquinário agrícola e serviços de manutenção; essa modernização foi responsável por uma parcela considerável dos impactos negativos do setor.

A partir deste fato surgiram diversos programas econômicos para mitigação das emissões GEE prejudiciais ao meio ambiente mundial, entre eles a precificação do carbono. Esta consiste em atribuir custo direto aos gases do efeito estufa, e tem sido elemento central no uso de incentivos econômicos para combater mudanças climáticas (Kosnik, 2018).

Uma das ferramentas para precificar o carbono é o protocolo *cap-and-trade*, que consiste na distribuição de permissões por meio dos governos dos países signatários do protocolo. O mesmo se baseia em uma metodologia que visa limitar

as emissões dos agentes econômicos, de forma que os agentes econômicos que emitem abaixo do limite, gerem créditos de carbono, que podem ser comprados pelos agentes econômicos com emissões excessivas. O limite pode ser definido a partir da mitigação de carbono que se deseja definir como meta.

No Brasil, o Renovabio também conhecido como Política Nacional de Biocombustíveis do Brasil é o primeiro mercado doméstico de carbono regulamentado em território nacional. O programa funciona como um instrumento de mitigação de emissões com base no protocolo *cap-and-trade*. Apesar de existirem iniciativas como esta para redução dos impactos dos GEE, sua implementação enfrenta grandes desafios para consolidação e credibilidade (BOTTINI, 2022).

2.2 Contexto histórico das estufas agrícolas

As estufas agrícolas foram utilizadas durante séculos como meio de proteção das plantas contra climas extremos, como por exemplo, no cultivo de espécies tropicais em latitudes mais altas. O uso de estufas tornou possível criar melhores condições de crescimento das culturas devido à manutenção do ambiente interno em condições adequadas se comparado ao ambiente externo (CRITTEN; BAYLEY, 2002).

Os custos de cultivo em estufas agrícolas dotadas de monitoramento e controle de seus parâmetros ambientais (umidade relativa do ar, temperatura e luminosidade) são geralmente mais caros do que os cultivos no campo. Com isso, deve-se alcançar uma maior produtividade para compensar os custos adicionais de implantação, operação e manutenção (SHAMSHIRI; ISHAK, 2013).

Mundialmente a agricultura de cultivo controlado está passando por uma transição impulsionada pelos avanços na tecnologia (SHAMSHIRI; et al., 2018). Inovações em sensores de baixo custo e baixo consumo, instrumentação, dispositivos de comunicação, processamento de dados móvel e aplicações, juntamente com os avanços tecnológicos no design das estruturas (COELHO, et al, 2020), estão mudando o cultivo controlado tradicional que consistia de apenas estruturas cobertas simples, para verdadeiras indústrias de alta tecnologia que em razão disto aumentam a produtividade do cultivo (EHRET; et al., 2001).

Nos últimos anos com o retorno da pauta ambiental nos assuntos globais, surgiram estudos sugerindo metodologias de operação das estufas abrangendo desde a reutilização da água do sistema, até instalação de micro geração de energia elétrica, seja eólica ou através de painéis solares (CUCE; et al. 2016).

O motivo por trás de estufas aumentarem a produtividade é que elas fornecem um melhor controle de fatores que antes dependeriam apenas do clima e estações do ano, e por se tratar de um ambiente controlado, existe maior facilidade no controle de pragas e doenças em seu ambiente interno, possibilitando ainda, utilização de tecnologias para condicionamento climático em seu interior. Segundo NNADI (2018) os ganhos de um agricultor dependem principalmente da produtividade do cultivo, da época de comercialização e da qualidade do produto. O cultivo dentro de uma estufa permite além de um crescimento acelerado, um maior número de ciclos devido à possibilidade de se criar um ambiente artificial favorável ao crescimento das plantas durante o ano todo

De acordo com SILVA *et. al.* (2005) a região Norte (Amazônia) compreende grande parte da Amazônia brasileira, apresenta clima predominantemente tropical quente e úmido, tendo umidade média relativa do ar de 90%, com extremos de variação de intensidade de radiação de um dia para o outro, sendo assim, um grande desafio para o cultivo de diversas culturas. Para isto pode-se obter bons resultados do plantio fazendo utilização inteligente dos materiais e do tipo de estufa. Para esta região pode-se ter um bom aproveitamento da estufa do tipo Zenital, o qual possui uma abertura no teto que pode ser utilizado tanto para reter quanto para liberar o calor no interior da estufa. Porém outro fator limitante para o cultivo de hortaliças nessa região é a dificuldade de transporte do calor e da massa de vapor no interior da estufa por conta da ausência de vento, muito importante para o modelo e tipo de estufa adotado.

Outro bioma desafiador para o plantio são as regiões do Nordeste, devido às constantes secas. Segundo SILVA *et. al.* (1998) a seca é um fenômeno dito recorrente principalmente em regiões semiáridas. Os efeitos de um extenso período de seca em uma determinada região dependem, não somente da duração e da intensidade da seca, mas também das condições socioeconômicas e culturais da população ali atingida.

No Brasil, o semiárido abrange 1.262 municípios, distribuídos nos estados do Maranhão, Piauí, Ceará, Rio Grande do Norte, Paraíba, Pernambuco, Alagoas, Sergipe Bahia e Minas Gerais (ROSSATO et al., 2017).

Felizmente meio a tantos desafios para o cultivo a tecnologia é uma grande aliada, segundo Calori et. al. (2018) uma das soluções encontradas para enfrentar esse desafio é o uso de tecnologias que proporcionem maior controle das condições climáticas no interior das estufas.

Para isto segundo Parronchi (2017) se faz necessário a adoção de um novo modelo tecnológico, o que impacta em um novo modelo de custos e formação de preços que hoje é chamado de Agricultura 4.0, que une conceitos de agricultura e IoT.

Um exemplo de aplicação com conceitos de Agricultura 4.0 são as estufas inteligentes. Para SILVA (2019), estufas inteligentes permitem a coleta de informações ambientais através de sensores e a execução automática de ações de acordo com as necessidades das plantas cultivadas através de atuadores.

Através das estufas inteligentes pode-se obter bons resultados no cultivo mesmo em regiões desfavoráveis como citado por SILVA *et. al* (1998), pois através dos atuadores no interior das estufas cria-se um ambiente favorável ao cultivo escolhido. O estudo apresentado por Santos et. al. (2020) mostrou resultados bastante promissores com a utilização de um *Gateway* de IoT, onde a principal vantagem é o encapsulamento das complexidades para implantação e das complexidades existentes na rotina de operação, o que facilita a utilização da tecnologia tanto no contexto do agronegócio, quanto em outros. Após a implementação do *Gateway* o mesmo foi testado e os dados foram coletados por um período de tempo, e posteriormente analisados para constatar que não houve inconsistências nem discrepâncias com as variáveis do ambiente. Apesar da fragilidade dos equipamentos eletrônicos quando utilizados em ambientes de intensa umidade, obteve-se um resultado satisfatório.

Com o advento da Indústria 4.0 e conceitos de *Internet of Things* (IoT), métodos científicos podem ser mais facilmente implementados, já que a conectividade dos dispositivos sensores com a Internet abre caminho para um efetivo acompanhamento remoto dos dados de cultivo. A aplicação destas novas tecnologias ao cultivo em estufas possibilita a identificação de problemas e

anormalidades no cultivo ao longo do tempo, e cálculo de estimativas de desempenho e produtividade (LI; ZHANG, ZHANG, 2014).

Um dos grandes desafios enfrentados pelos sistemas de IoT é armazenar e processar o grande volume de dados que os dispositivos produzem. Para suprir esta limitação as tecnologias Cloud Computing são apontadas como uma ótima alternativa para complemento dos dispositivos IoT, devido sua alta capacidade de armazenamento, e processamento on-demand (VANELLI et.al.).

2.3 Avanços do IoT nos últimos anos

A primeira revolução industrial iniciada em meados do final do sec. XVIII foi um marco para os processos de produção mecanizados, que além de mudar o paradigma econômico também provocou mudanças nos moldes tanto na indústria quanto aos modelos de gestão de processos. Diante dessas mudanças instalou-se uma cultura de busca de maior produtividade nos processos industriais e maior qualidade dos produtos, reduzindo custos de fabricação por meio de soluções eficazes para atender os clientes com maior velocidade. Nos últimos anos essas mudanças têm ficado cada vez mais visíveis com o desenvolvimento dos sistemas de informação, que vieram alavancar o modelo de produção industrial popularizando termos como Internet das Coisas, *Cloud Computing*, *Business Intelligence* e *Machine Leaning*. Essas novas estruturas de produção, dotadas de *smart-devices* ligados à uma rede são o estado da arte nos dias de hoje para atingir as exigências dos mercados atuais (B.P. SANTOS et.al.).

Consistindo de dispositivos interconectados como celulares, sensores, veículos, televisores e outros *smart-devices*, a internet das coisas vem facilitando a aquisição e o acúmulo de uma grande massa de dados e informações, sendo também impulsionada pelo crescimento dos serviços de infraestrutura de *Cloud Computing*, que suprem a limitação de baixa capacidade de armazenamento dos dispositivos (S.A. CHAUDHRY et.al.).

Um dos grandes desafios enfrentados pelos sistemas de IoT é armazenar e processar o grande volume de dados que os dispositivos produzem. Para suprir esta limitação as tecnologias *Cloud Computing* são apontadas como uma ótima

alternativa para complemento dos dispositivos IoT, devido sua alta capacidade de armazenamento, e processamento *on-demand* (VANELLI et.al.).

2.4 Conceitos e Tecnologias que auxiliam a inovação

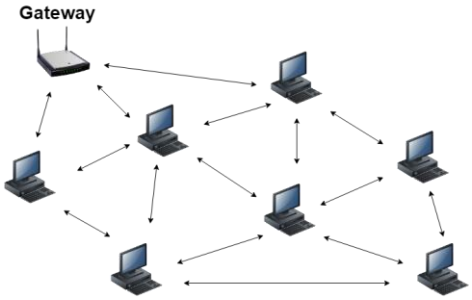
Neste tópico, serão comentados conceitos e tecnologias utilizadas neste projeto e que auxiliam a integração de sistemas distribuídos de IoT e *Big Data*.

2.4.1 Rede *Mesh*

Diferentemente da rede *WI-FI* convencional, onde temos um único roteador conectado à rede banda larga, uma rede *mesh* ideal é formada por nós dispostos em diferentes lugares formando como se fosse uma única de rede *WI-FI*. A principal característica deste modelo é que seus nós internos propagam o sinal da rede interna como se fossem repetidores, de forma a obtermos uma malha colaborativa que garante que a informação transite do ponto mais distante, até o *gateway* com uma alta resiliência a falhas. O modelo *mesh* permite que o dispositivo gerenciador conheça todas as rotas entre os clientes da rede interna com elevada precisão, possibilitando-se que, na ocorrência de mau funcionamento de um dos dispositivos, a rede de maneira colaborativa encontre uma nova rota para transitar as informações (AKYILDIZ et al, 2008).

Para estruturar a rede *mesh* do sistema aqui proposto foi utilizado este conceito para organizar uma rede privada de sensoriamento cooperativa, segura e com resiliência a falhas. Um exemplo de modelo pode ser observado na Figura 1.

Figura 1 - Modelo de uma rede mesh



Fonte: Autoria própria (2023).

A vantagem da utilização dessa metodologia é que substitui o canal de comunicação, que de maneira convencional seriam cabos, para o ar, tornando o projeto mais barato.

Para estruturar a rede *mesh* do sistema aqui proposto foram utilizados dispositivos ESP32. Um dos grandes desafios enfrentados pelos sistemas de IoT é armazenar e processar o grande volume de dados que os dispositivos produzem. Para suprir esta limitação as tecnologias *Cloud Computing* são apontadas como uma ótima alternativa para complemento dos dispositivos IoT, devido sua alta capacidade de armazenamento, e processamento *on-demand* (VANELLI et.al.).

2.4.2 Restful API

A definição de REST (*Representational State Transfer*) representa um modelo de arquitetura associado às aplicações *Web*, que propõe regras de implementação com o objetivo de estabelecer padrões ou normas de modo a facilitar a comunicação entre cliente e servidor de aplicações *web*. O modelo REST permite que ambas as implementações, do cliente e do servidor, sejam desenvolvidas de maneira modularizada e independente, de forma que o servidor seja desenvolvido sem que haja qualquer conhecimento pelo cliente do funcionamento interno do servidor, permitindo que as componentes associadas ao lado do servidor não interfiram no lado do cliente, e vice-versa (Farinha, 2019).

Atualmente, grande parte das comunicações entre serviços na internet utilizam o modelo arquitetural REST. Chamadas REST são uma parte fundamental na arquitetura e implementação de sistemas, pois elas permitem que diferentes operações sejam expostas à internet, de forma que outros serviços possam

consumir sua aplicação. Este tipo de aplicação pode ou não possuir algum tipo de autenticação de conexão. Para estruturar a rede *mesh* do sistema aqui proposto foi utilizado este conceito para organizar uma rede privada de sensoriamento cooperativa, segura e com resiliência a falhas. O único pré-requisito é que ambos, tanto cliente como servidor, conheçam as normas de implementação e consumo das REST APIs (Andrade et al., 2019).

Este modelo proporciona alguns benefícios em termos de gerenciabilidade de aplicação, como o controle de requisições por IP (*Internet Protocol*) para que sejam estabelecidos limites de requisições por parte do cliente, para que a integridade do servidor seja mantida e seu funcionamento preservado, evitando problemas de indisponibilidade por excesso de carga. Outra vantagem notável é que para algumas requisições pode-se utilizar uma memória cache, de forma que para requisições que retornam a mesma resposta, esta fique armazenada e evite o processamento desnecessário por parte do servidor da aplicação (Rodrigues, 2020).

Para maior segurança da comunicação entre a API e os clientes, algumas metodologias de autenticação com o servidor são possíveis, dentre elas pode-se utilizar a autenticação baseada em sessões, baseada em *tokens*, e baseada em chaves de API (Couto, 2021).

Na metodologia de autenticação baseada em sessões, o servidor cria uma sessão para o cliente que insere corretamente as credenciais de acesso. Esta sessão é armazenada em memória no servidor durante um tempo determinado pelo desenvolvedor. O identificador da sessão gerada então é armazenado como um *cookie* no *browser* do cliente. Esta autenticação é normalmente utilizada em sites para manter um *login* válido durante algum tempo.

No caso da autenticação baseada em *tokens*, o servidor gera um *token* que então é enviado para o cliente que fica incumbido de guardá-lo. Após a autenticação o cliente deve enviar o token a cada solicitação para o servidor, que ao recebê-lo valida se o mesmo é válido. Diferente da autenticação em sessões, neste caso o servidor não armazena o *token* do cliente, pois o *token* em si tem informações que permitem com que o servidor valide a conexão.

Dentre os variados métodos de autenticação possíveis para APIs do tipo REST, optou-se pelo método de autenticação por Chave de API. Neste método é gerada uma chave exclusiva para cada cliente, de forma que quando o servidor

recebe um pedido de conexão com uma chave, e este verifica se esta é uma chave válida, bem como identifica quem é o cliente que está se conectando.

Uma ressalva sobre este tipo de autenticação, é que ela somente faz sentido se o canal de comunicação entre cliente e servidor é seguro e isolado. Para isto o OpenVPN é uma solução de VPN (*Virtual Private Network*) que faz uso do mecanismo seguro TLS/SSL (*Transport Layer Security/Secure Sockets Layer*) para autenticação e trocas de chaves entre dispositivos que desejem se comunicar de forma virtual e privada.

2.4.3 OpenVPN

No atual cenário digital, a segurança na troca de informações via internet tornou-se um imperativo inegável, reforçando a relevância dos métodos seguros para a proteção dos dados em trânsito. Conforme apontado por Heyman (2017), as conexões de VPN estabelecem um link seguro entre um dispositivo remoto e uma rede doméstica. Em vez de estabelecer uma verdadeira rede privada, essas conexões funcionam por meio da criptografia dos pacotes de dados enviados. Esta é uma solução amplamente utilizada devido ao seu baixo custo, tendo um bom desempenho em criar rotas na rede pública para acessar máquinas. Seu funcionamento se baseia em desordenar a informação enviada de forma a tornar a informação incompreensível, se vista sem a chave, que funciona como um manual para reordenar os dados para possibilitar a leitura (Azevedo, 2019).

2.4.4 Máquinas Virtuais hospedadas em *Cloud*

Dentre as opções disponíveis no mercado para provisionamento de VM (Máquina Virtual), optou-se pela utilização da Microsoft Azure. Um grande ponto a favor da utilização de VMs para provisionamento de servidores é sua fácil escalabilidade sob demanda e gerenciabilidade de custos. Para fins deste trabalho optou-se por uma VM Ubuntu Server básica com as especificações que podem ser vistas no Quadro 1.

Quadro 1 - Dados do *hardware* utilizado

VM AZURE SPECIFICATION	
Local	Brazil South
Operating System	Linux
Type	Ubuntu Server
Tier	Standard
Category	General Purpose
Instance Series	Bs-Series
CPU Cores	1
Memory	1GB RAM
Storage	4 GB SSD (Premium)
Price	$12.26/month

Fonte: Autoria própria (2023).

2.4.5 IoT HUB

O Azure IoT *HUB*, um serviço oferecido pela Microsoft, atua como um mediador central de mensagens para aplicações relacionadas à Internet das Coisas (IoT). Este serviço tem a capacidade de interagir bidirecionalmente com inúmeros dispositivos IoT de forma segura, coletando e conectando-se a eles para a aquisição de dados de telemetria em larga escala (MICROSOFT, 2023).

Este serviço da Microsoft possibilita a conexão, o provisionamento e a administração segura de dispositivos IoT, independentemente da quantidade ou localização dos mesmos. Adicionalmente, o Azure IoT *HUB* oferece avançados recursos de comunicação entre dispositivos IoT e serviços em nuvem, permitindo que comandos e notificações sejam enviados para os dispositivos e que estes, por sua vez, enviem relatórios de status e dados de telemetria para a nuvem (MICROSOFT, 2023).

Uma das principais características do IoT *HUB* é sua capacidade de se adaptar às necessidades do usuário, gerenciando de forma segura um grande número de eventos simultâneos. Além disso, permite a integração com outros serviços Azure, como o Azure *Stream Analytics* para análise de dados em tempo real, e o Azure *Machine Learning* para previsões baseadas em aprendizado de máquina, formando assim uma plataforma robusta para desenvolvimento de soluções IoT (MICROSOFT, 2023).

No aspecto de segurança, o Azure IoT *HUB* proporciona autenticação por dispositivo, garantindo que cada dispositivo IoT tenha uma identidade única na solução. Isso ajuda a assegurar que somente dispositivos autenticados possam se conectar ao IoT *HUB* e que as mensagens sejam enviadas apenas para os dispositivos com as devidas permissões. Além disso, o serviço suporta uma variedade de opções de transporte e protocolos, como HTTP, AMQP e MQTT, para atender a diversos requisitos e cenários de aplicação (Mendes, 2019).

O IoT *HUB* tem a capacidade de coletar dados e gerenciar dispositivos, bem como todo o seu ciclo de vida. Para garantir uma maior segurança ao integrar dispositivos à rede que ele gerencia, o DPS (*Device Provisioning Service*) é empregado. Quando uma mensagem é recebida, o IoT *HUB* pode reter as informações para que possam ser acessadas por clientes de processamento, mesmo que estes se conectem posteriormente. Ademais, é possível encaminhar mensagens para clientes com base nos dados da mensagem ou do dispositivo (Maia, 2020).

O IoT *HUB* não só gerencia dispositivos de telemetria e dispositivos Edge, mas também lida com arquivos, podendo direcioná-los diretamente para uma conta de armazenamento. Isso facilita o despejo de dados ou a transferência de arquivos de integração que os dispositivos podem precisar enviar para executar algum processo que não seja gerenciado por mensagens ou comandos (Maia, 2020).

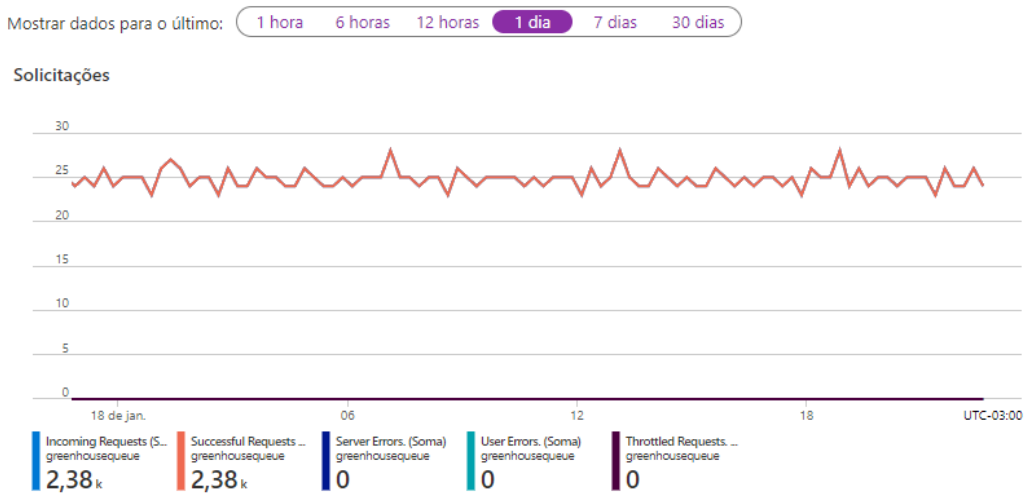
2.4.6 Serviços de Enfileiramento para soluções de *Big Data*

Um dos grandes desafios da Internet das Coisas é o processamento de um grande volume de mensagens contendo dados, e este é um ponto onde a Microsoft Azure oferece uma vasta gama de soluções, para esta implementação optou-se pelo uso do *Service Bus*, um barramento de mensagens com alto poder de escala e confiabilidade na entrega de mensagens.

O *Service Bus* é um serviço que permite a intermediação de mensagens e informações com recurso de filas e outras estruturas de dados, sendo que a fila é a estrutura de dados relevante para este projeto. Este serviço oferece um gerenciamento de filas para grandes volumes de entrada de informação, oferecendo segurança para o servidor que irá fazer consumo das mensagens no *backend* em

um volume saudável, e não sofre com picos de envios de mensagens através do hub. Um grande ponto forte desta solução também são seus relatórios de utilização do barramento, um exemplo de relatório do *Service Bus* pode ser visualizado na Figura 2, onde é mostrado o barramento em operação.

Figura 2 - Solicitações por segundo da Fila de Serviço do IoT HUB



Fonte: Autoria própria (2023).

Em sua versão mais básica, o *Service Bus* oferece um barramento que pode comportar até 1024Mb de armazenamento de mensagens no formato TXT.

2.4.7 Aplicações *Serverless* e Azure Functions

Serviços *serverless* em *cloud* são uma das formas mais recentes e populares de se criar e executar aplicativos na nuvem. Esses serviços permitem que os desenvolvedores escrevam código que é executado automaticamente em resposta a eventos específicos, sem precisar gerenciar a infraestrutura ou os servidores subjacentes. Um dos exemplos mais conhecidos desses serviços são as Azure Functions da Microsoft (GALVEIAS, 2021).

As Azure Functions são uma plataforma de computação *serverless* fornecida pela Microsoft que permite aos desenvolvedores criar aplicativos na nuvem sem precisar gerenciar a infraestrutura subjacente. Essa plataforma permite que os desenvolvedores escrevam código que é executado em resposta a eventos específicos, como uma mensagem chegando numa fila, uma solicitação HTTP

(*HyperText Transfer Protocol*) chegando a uma API ou um arquivo sendo adicionado a um armazenamento de dados.

As funções são executadas em um ambiente gerenciado e escalonado automaticamente pela plataforma, garantindo que os aplicativos sejam altamente disponíveis e escaláveis. Além disso, as Azure Functions são altamente integradas com outros serviços da Microsoft Azure, o que permite aos desenvolvedores criar aplicativos complexos que aproveitam uma variedade de recursos da nuvem (GALVEIAS, 2021).

Uma das grandes vantagens do uso de serviços *serverless* como as Azure Functions é a economia de custos. Como os desenvolvedores só pagam pelo tempo em que suas funções estão sendo executadas, os custos são reduzidos em comparação com a execução de um servidor o tempo todo. Além disso, os desenvolvedores não precisam se preocupar com a infraestrutura subjacente, o que permite que eles se concentrem em escrever código de alta qualidade.

No entanto, existem algumas desvantagens em usar serviços *serverless* como as Azure Functions. Uma das principais é a perda de controle sobre a infraestrutura subjacente, o que pode dificultar a depuração de problemas e a otimização de desempenho. Além disso, o tempo de inicialização das funções pode ser mais longo do que a execução em um servidor dedicado, o que pode levar a um pequeno atraso no tempo de resposta.

Em resumo, os serviços *serverless* como as Azure Functions fornecem uma maneira fácil e econômica de criar aplicativos na nuvem. Eles são altamente escaláveis, altamente integrados e permitem que os desenvolvedores se concentrem na criação de código de alta qualidade em vez de gerenciar infraestrutura.

Embora haja algumas desvantagens em usar serviços *serverless*, como a perda de controle sobre a infraestrutura, os benefícios dos serviços *serverless* como as Azure Functions da Microsoft superam em muito as desvantagens, tornando essa opção uma escolha atraente para desenvolvedores que buscam criar aplicativos na nuvem. Com a evolução contínua das tecnologias de nuvem, é provável que os serviços *serverless* continuem a desempenhar um papel cada vez mais importante no desenvolvimento de aplicativos na nuvem, oferecendo uma maneira fácil e econômica de criar aplicativos altamente escaláveis e disponíveis.

Com a crescente demanda por aplicativos altamente disponíveis e escaláveis, os serviços *serverless* se tornaram uma opção cada vez mais popular para desenvolvedores na nuvem. Neste contexto, as Azure Functions da Microsoft oferecem uma plataforma poderosa e integrada para criar aplicativos *serverless* com eficiência e economia de custos.

Dentre as linguagens de programação disponíveis para este serviço, optou-se pela implementação em Python, porém também é possível utilização de Node.js, Java, C#, e PowerShell. Este tipo de serviço tem como característica ser executado sob demanda, desta forma, é necessário definir acontecimentos que serão o gatilho de execução do código. Para a implementação deste projeto obteve-se proveito da fácil integração entre os serviços da Azure, definindo como evento de disparo do Azure Functions a inserção de itens a fila do Service Bus, ou seja, sempre que um item é incluído na fila de processamento, o *backend* em Azure Functions é executado para consumir os itens inseridos, até que a fila seja esvaziada. A estrutura de funcionamento do Azure Functions pode ser vista na Figura 3.

Figura 3 - Arquitetura de Função do Azure Function

Fonte: Autoria própria (2023).

2.4.8 Bancos de Dados Hospedados em *Cloud*

Por fim, após os dados transitarem por todo o pipeline e serem processados, os mesmos precisam ser armazenados em uma solução de bancos de dados, optou-se então pelo banco de dados SQL Server hospedado na Azure. Um dos motivos pela escolha do SQL Server frente a outras opções do mercado, se dá pelo menor custo inicial na plataforma da Azure, bem como fácil integração com todos os outros serviços por meio de *triggers* e eventos.

O plano contratado será o básico, com armazenamento de 2 GB inicialmente, nesta configuração o banco de dados possui custo de aproximadamente US$ 6 por mês.

O armazenamento em um SQL Server da Azure oferece uma solução segura e escalável para gerenciar dados estruturados em nuvem. Com a capacidade de armazenar e processar grandes volumes de dados em tempo real, o SQL Server da Azure é uma opção popular para muitas empresas que precisam gerenciar e acessar dados de maneira eficiente.

Com a utilização de recursos de computação e armazenamento em nuvem, o SQL Server da Azure permite a escalabilidade horizontal e vertical, o que significa que as empresas podem aumentar o poder de processamento e armazenamento de acordo com as necessidades do negócio, sem a necessidade de comprar e gerenciar *hardware* adicional.

O SQL Server da Azure também oferece recursos avançados de segurança e conformidade, como criptografia de dados em repouso e em trânsito, autenticação multifator e auditoria avançada. Esses recursos garantem que os dados sejam armazenados e gerenciados de maneira segura e em conformidade com os regulamentos e padrões de segurança do setor.

Além disso, o SQL Server da Azure suporta uma variedade de opções de implantação, incluindo implantações gerenciadas pelo cliente e serviços gerenciados, que permitem que as empresas escolham o modelo de implantação mais adequado para suas necessidades de negócios.

Em termos de funcionalidades, o SQL Server da Azure oferece uma ampla gama de recursos para gerenciar e acessar dados, incluindo o uso de consultas SQL (*Structured Query Language*) para recuperar informações, armazenamento de dados em tabelas relacionais, suporte a transações ACID (*Atomicity, Consistency, Isolation, Durability*) para garantir a integridade dos dados, além de suporte para procedimentos armazenados, *triggers* e índices.

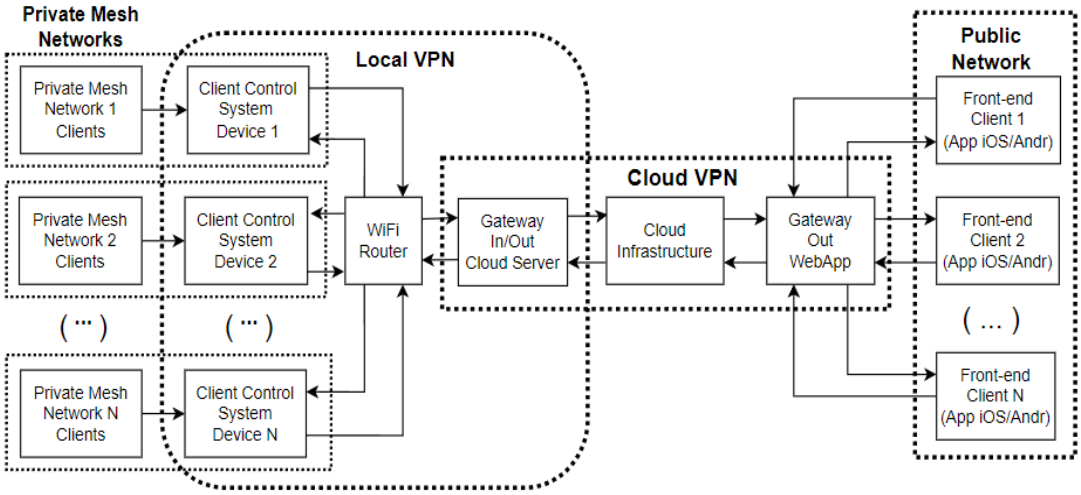
Em resumo, o armazenamento em um SQL Server da Azure oferece uma solução poderosa e segura para gerenciar grandes volumes de dados em nuvem. Com recursos de escalabilidade, segurança e conformidade avançados, o SQL Server da Azure é uma opção atraente para muitas empresas que precisam gerenciar dados de maneira eficiente e escalável em nuvem.

3 DESENVOLVIMENTO

3.1 Arquitetura do Sistema

A arquitetura do sistema proposto que utiliza a tecnologia IoT para monitoramento e controle de estufas é apresentada no Fluxograma 1. Nela, é possível identificar quatro diferentes tipos de redes em operação, sendo eles a *Private Mesh Networks*, *Local* VPN, *Cloud* VPN e *Public Network*.

Fluxograma 1 – Arquitetura de monitoramento e controle baseado em IoT aplicada ao cultivo controlado



Fonte: Autoria própria (2023).

Cada camada mostrada será detalhada nos capítulos a seguir.

3.1.1 Private Mesh Networks

Esta camada de arquitetura é a infraestrutura local onde o sensoriamento será aplicado, o diagrama detalhado da PMN (*Private Mesh Network*) pode ser visto no Fluxograma 2.

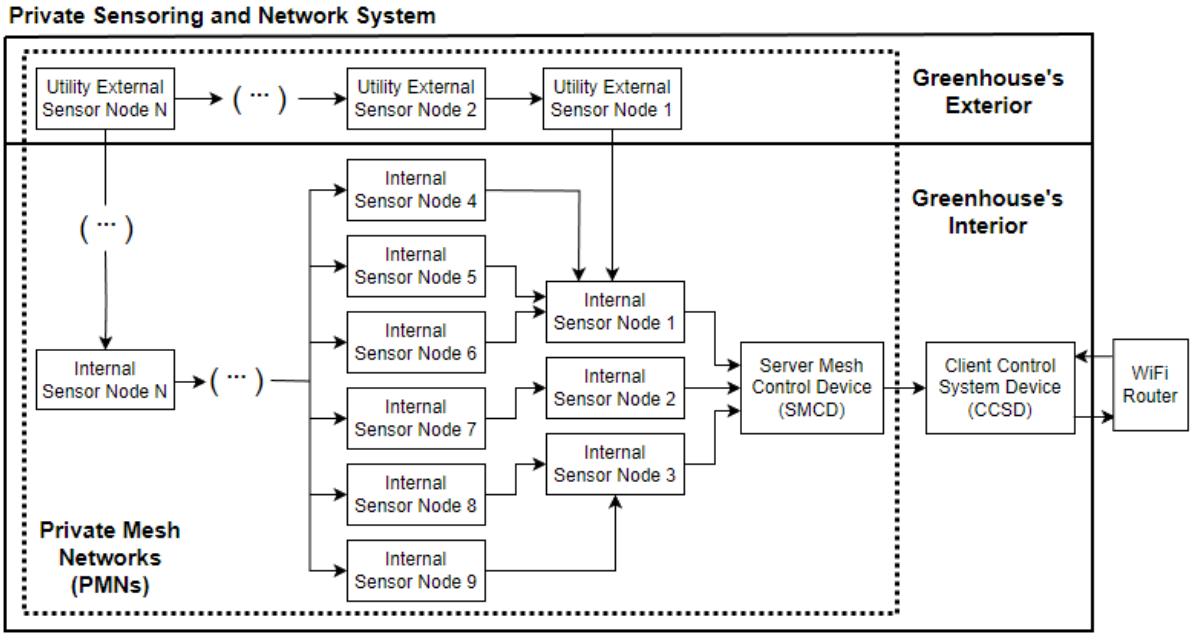
A estrutura base de sensoriamento local consiste de dois sistemas bem definidos. O sistema disposto no interior na estufa que será monitorada, e um sistema utilitário externo a estufa, os dispositivos SN (*Sensor Node*) são os

dispositivos que capturam dados dos parâmetros ambientais, conectados a uma rede *mesh* local gerenciada por um nó gerenciador denominado SMCD (*Server Mesh Control Device*), este SMCD será o responsável por endereçar os nós de sensoriamento da rede *mesh*, e enviar os dados para o CCSD (*Client Control System Device*).

O CCSD é um dispositivo responsável por se comunicar com os nós gerenciadores da rede *mesh*, para que todos os dados sejam recebidos em uma central única conectada ao roteador, além de estar conectado a um drive de acionamento de atuadores.

Para cobrir toda a área da estufa, podemos utilizar um ou mais conjuntos de dispositivos desta camada, de forma a dar flexibilidade à área de sensoriamento, podendo ser adequado para os diversos formatos e dimensões possíveis da estufas agrícolas, tornando possível dessa forma monitorar e acionar atuadores separadamente, por unidade de PMN. Cada dispositivo será detalhado a seguir.

Fluxograma 2 - Fluxograma de operação do *Private Sensoring and Network System*

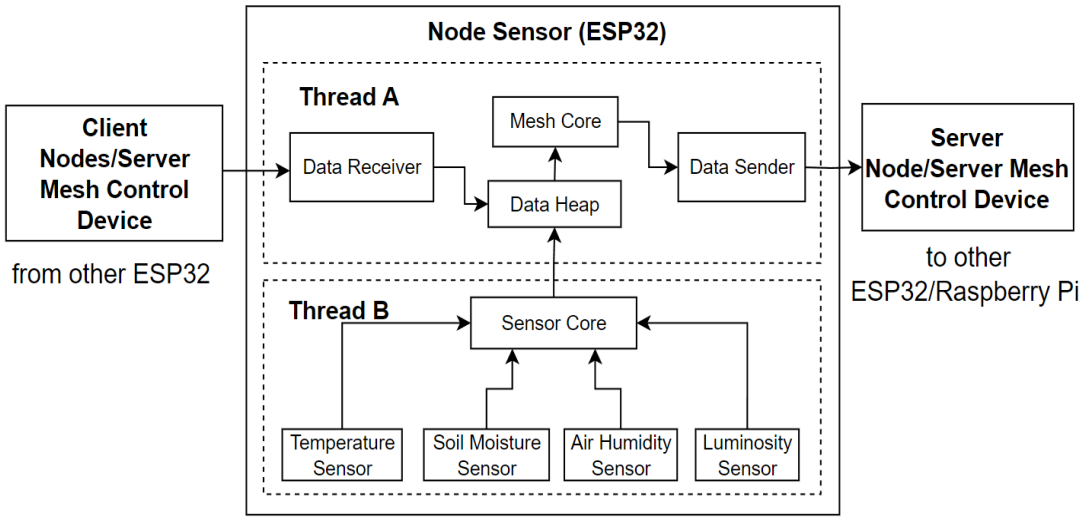


Fonte: Autoria própria (2023).

3.1.2 Sensor Nodes

A malha interna e a malha externa de sensoriamento, consistem de dispositivos ESP-32 conectados a sensores de temperatura, umidade do solo, umidade do ar e luminosidade, o diagrama detalhado pode ser visto no Fluxograma 3.

Fluxograma 3 - Fluxograma de operação dos Sensor Nodes



Fonte: Autoria própria (2023).

Para comunicação entre os sensores, foi desenvolvida uma rede local *WI-FI* com conceitos de rede *mesh***;** a mesma possui dispositivos com dois tipos distintos de operação, os SNs e o gerenciador da rede SMCD. Estas 2 operações citadas acima têm funções bem definidas.

3.1.3 Server Mesh Control Device

O Gerenciador de rede SMCD é definido via código como tal, onde para cada sistema de sensoriamento existirá apenas um gerenciador. Sua função principal é ser o host da rede privada local (sem acesso à internet) e efetuar o mapeamento de todos os dispositivos pareados via *WI-FI*, bem como transmitir as informações recebidas para uma segunda rede *WI-FI* via protocolo HTTP (com acesso à internet) onde estará conectado um terceiro dispositivo chamado CCSD que será detalhado mais à frente. O diagrama completo da rede privada de sensoriamento pode ser observado no Fluxograma 1.

Ao iniciar sua operação, o SMCD passa a fornecer uma rede *WI-FI* cujo acesso é possível através de um *login* e senha, semelhante a uma rede *WI-FI* doméstica, então passará a receber solicitações de pareamento de dispositivos que se conectarem.

Após autorizar uma conexão de um novo dispositivo em sua rede, o Gerenciador de Rede recebe um número identificador do dispositivo e atribui a ele um endereço de IP, bem como aloca este endereço no mapeamento da hierarquia de dispositivos como atrelado diretamente ao SMCD. O mesmo é feito para todos os dispositivos que forem pareados, após a conexão então o SMCD passa a receber dados dos dispositivos pareados.

Os SNs são dispositivos que possuem dois estados de operação, *online* e *offline*. Ao iniciar sua operação, todos os SN se encontram na operação *offline*. A única ação deste modo de operação é buscar uma rede previamente definida via código ao alcance do seu sinal fornecendo *login* e senha. Após estabelecer a conexão com o ponto de acesso (inicialmente o SMCD), o SN passa a operar no modo online como repetidor do sinal do SMCD, ficando disponível para conexão de novos dispositivos que tentem se parear oferecendo o *login* e senha da rede. No modo online o nó de sensoriamento também passa a fazer utilização dos seus sensores acoplados, tirando medições de tempos em tempos e enviado os dados via protocolo HTTP para o gerenciador da rede, acompanhados do seu identificador para que o gerenciador registre a origem da informação.

Quando um segundo SN se conecta ao primeiro mencionado anteriormente, o SN primário atribui um endereço de IP ao SN secundário e recebe o identificador único da nova conexão, em seguida o mesmo informa ao gerenciador da rede sobre a nova conexão. O SMCD então registra o novo dispositivo e o atribui na hierarquia como atrelado ao SN.

O SN secundário então estará apto a fazer envio também das suas informações dos sensores através da rede. Assim como o SN primário, o SN secundário passa a receber conexões de novos dispositivos, e desta forma ocorre até o último nó da estufa estar conectado a malha *mesh*. Após cada pareamento de SN, em paralelo ao roteamento dos próximos SN, o SN online já inicia o sensoriamento local e a enviar seus dados para o SMCD

O mapeamento da rede *mesh* é construído em formato JSON (*JavaScript Object Notation*), conforme pode ser visto na Figura 4.

Desta forma o SMCD começa a receber as medições dos sensores através da rede *WI-FI,* e redireciona-los via requisição HTTP para o próximo dispositivo, este definido como CCSD. Esta comunicação acontece dentro da rede *WI-FI* local de acordo com o protocolo WLAN (*Wireless Local Network*), auxiliando em segurança na transmissão da informação entre o SMCD e o CCSD. Desta forma, podemos dizer que o SMCD faz papel de ponte entre a rede *mesh* e o *WI-FI* doméstico.

Figura 4 - Modelo de mapeamento de rede mesh

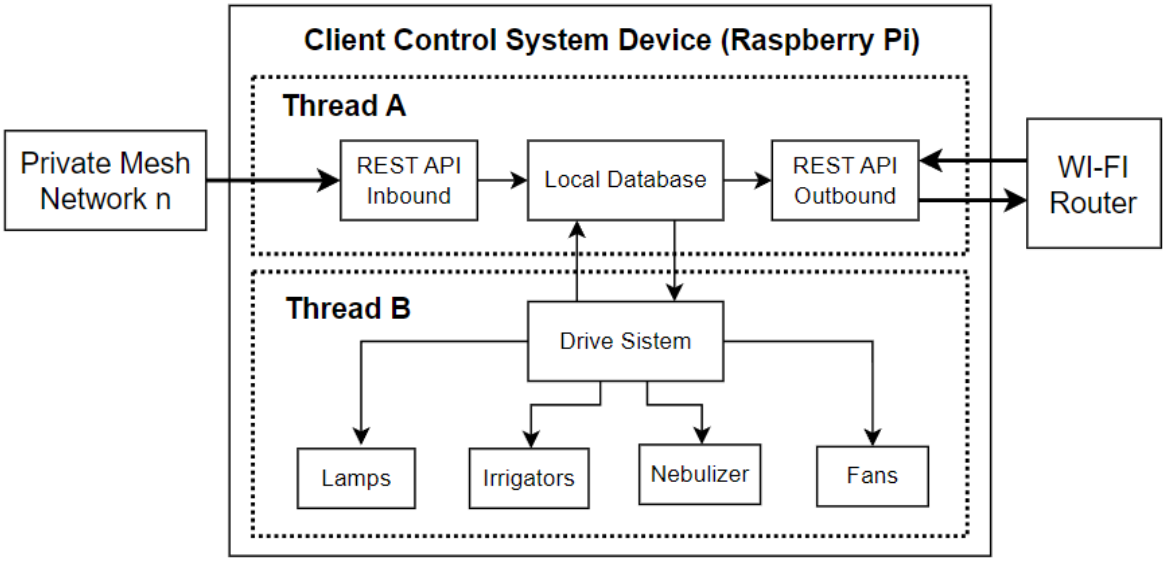

```
[
    {
        "MasterGateway": [
            {
                "id": 0,
                "node_id": 5285657415,
                "status": "connected",
                "connections": [
                    {
                        "id": 0,
                        "node_id": 8584948494,
                        "status": "connected",
                        "connections": [
                            {
                                "id": 0,
                                "node_id": 9813508447,
                                "status": "connected",
                                "connections": [
                                    {
                                        "id": 0,
                                        "node_id": 9813508447,
                                        "status": "connected",
                                        "connections": []
                                    }
                                ]
                            }
                        ]
                    }
                ]
            }
        ]
    }
]
```


Fonte: Autoria própria (2023).

3.1.4 Client Control System Device.

O CCSD por sua vez possui algumas funções mais complexas, já que consiste de um dispositivo mais robusto. Sua operação é dividida em 2 serviços diferentes detalhados a seguir. O diagrama simplificado da operação do CCSD pode ser visto abaixo no Fluxograma 4.

Fluxograma 4 - Fluxograma de operação do Client Control Sistem Device



Fonte: Autoria própria (2023).

3.1.4.1 API de Aquisição e Transmissão de Dados.

A API hospedada no CCSD tem como uma das suas principais funções receber todos os dados dos SMCD disponíveis na rede, e armazena-los em um banco de dados local, bem como sincronizar os dados do banco de dados local com o banco de dados hospedado na *cloud* através de um *gateway* de IoT. Para que os dados cheguem ao seu destino existem alguns passos adicionais que serão mencionados à frente. Ao CCSD se limita à função de verificar a cada 1 minuto quais dados ainda não foram enviados para a infraestrutura *cloud* e efetuar o envio.

Assim como o recebimento dos dados, o envio para o *Gateway* de IoT é feito através de requisições HTTP. Esta comunicação é encapsulada por uma VPN criada na Azure, para garantir que os dados transitem com segurança até o *Gateway*.

Para otimizar o envio dos dados, estes são agrupados em um objeto JSON, contendo diversas medições dos diversos nós, de forma a aproveitar o máximo cada requisição HTTP economizando recursos de processamento do *Gateway*. Um

exemplo do formato enviado no corpo da requisição HTTP para o *Gateway* pode ser visto abaixo na Figura 5:

Figura 5 - Modelo de Dados de sensores

```
[
    {
        "id": 507,
        "deviceid": 1,
        "internal_id": "5285657415",
        "topic": "sensor_stream",
        "network_name": "MESH_NETWORK_RASPBERRY01",
        "role": "sensor",
        "location": "internal",
        "temperature": 25.2923699381116,
        "soil_moisture": 98.3938603265179,
        "humidity": 41.44253262561562,
        "luminosity": 374.3324,
        "datainsert": "2023-05-21 04:22:26"
    }
]
```

Fonte: Autoria própria (2023).

3.1.4.2 Módulo de controle ambiente

O segundo módulo de operação da CCSD possui como principal função analisar os dados climáticos no interior da estufa, e interagir com um módulo de relês, desta forma é possível acionar os dispositivos disponíveis na infraestrutura local para manter o clima interno da estufa dentro dos parâmetros definidos, bem como quando isto não é possível, emitir alertas que podem ser transmitidos para o agricultor.

Este módulo pode operar de duas formas, a primeira de maneira automática onde o módulo lê os parâmetros pré-definidos e aplica o controle para manter os parâmetros dentro dos ideais, sendo a segunda acionada de maneira manual, onde os controles automáticos são desabilitados e o módulo aguarda instruções de acionamento manual via aplicativo.

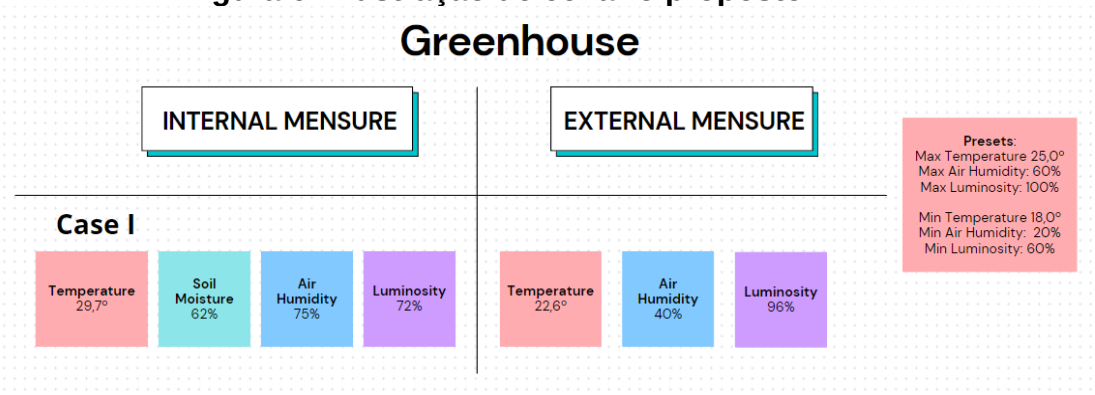
Nas duas operações, o módulo valida alguns critérios úteis para emissão de alertas, um exemplo é a identificação de nós com medições com desvio padrão de 15% para mais ou para menos, podendo indicar falhas nos sensores, ou ainda má distribuição dos irrigadores e atuadores mal posicionados. Os alertas são gravados

no banco de dados local, sendo enviados para o *Gateway* bem como os dados de medição, podendo ser utilizados para alertas em tempo real no aplicativo.

Na operação automática, teremos ainda validações com os sensores internos e externos, para verificar se existe ganho real em acionamento dos atuadores. O motivo pelo qual são necessários sensores dentro e fora da área de sensoriamento se dá pelo fato de isso possibilitar maior inteligência na atuação dos controladores internos a estufa, reduzindo o custo de operação desnecessária. Podemos observar alguns cenários de exemplo.

No caso I mostrado na Figura 6, para um determinado cultivo foi utilizado um *preset* de 25ºC de temperatura máxima. Ao observarmos a temperatura interna (29.7ºC) e a externa (22.6ºC) podemos concluir que é um cenário onde podemos obter um bom ganho de regulação de temperatura interna da estufa com a troca de gases internos e externos através de exaustores, pois devido ao fato de a temperatura externa ser menor do que a interna, existe a tendência de redução da temperatura interna através da ventilação. Apesar de a umidade do ar no meio externo estar menor do que no meio interno, caso a umidade interna fique abaixo do esperado com a troca de gases para regular a temperatura, a umidade do ar pode ser compensada com o acionamento de um nebulizador para que não ocorra ressecamento das plantas.

Figura 6 - Ilustração de cenário proposto 1

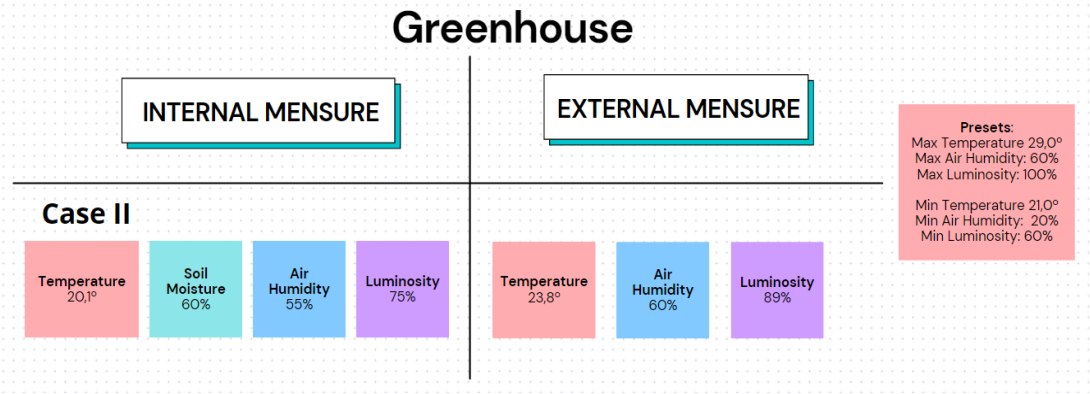


Fonte: Autoria própria (2023).

No caso II mostrado na Figura 7, podemos observar o contrário do caso I, neste cenário a temperatura interna está acima do esperado, e a externa abaixo do

esperado. Aqui também podemos obter um ganho na troca de gases internos e externos

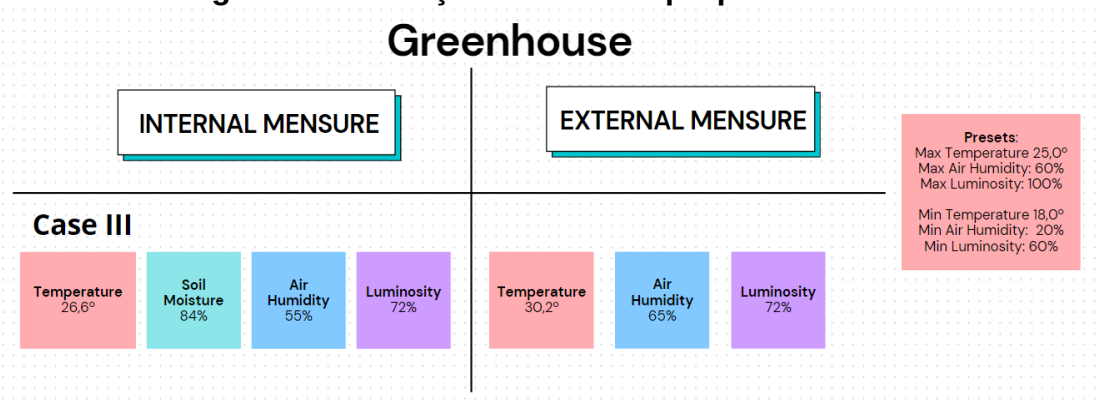
Figura 7 - Ilustração de cenário proposto 2



Fonte: Autoria própria (2023).

Analisando agora um terceiro cenário da Figura 8, podemos observar que ambas as temperaturas estão acima do esperado, nesse caso não existe ganho em utilização de exaustores e, portanto, poderão permanecer desligados. Em cenários como este é enviado uma sugestão ao cliente para que avalie a contratação ou melhoria da solução de resfriamento para seu ambiente, para otimizar a sua produção.

Figura 8 - Ilustração de cenário proposto 3



Fonte: Autoria própria (2023).

Até aqui temos toda a estrutura de aquisição de dados e envio para a infraestrutura *cloud*, partindo do diagrama mostrado no Fluxograma 2, podemos observar o diagrama incluindo o funcionamento do sistema quando temos diversos clientes conectados ao serviço, sendo nossas CCSDs representadas no diagrama

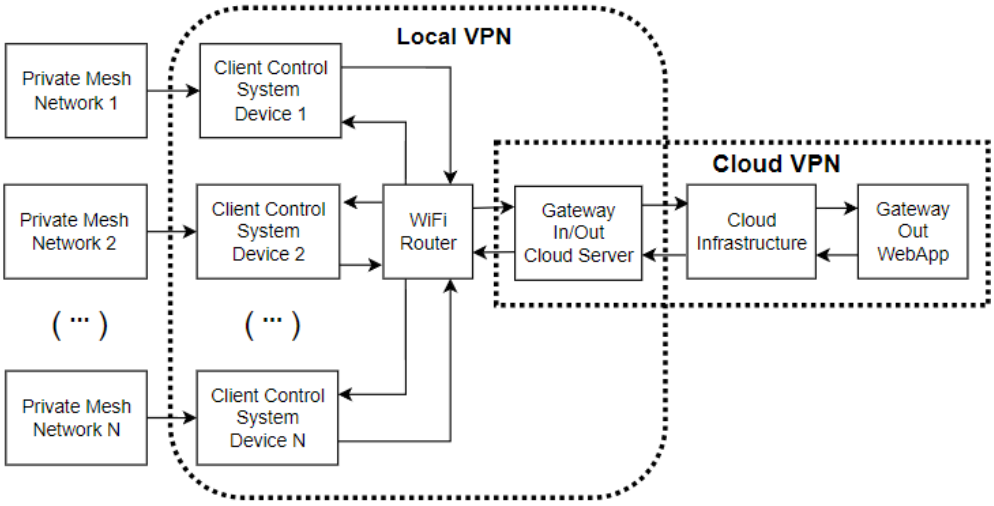
enviando os dados para um *Gateway* Central hospedado na Azure, que transporta as informações para o interior da infraestrutura *cloud*, que será apresentada com mais detalhes nos tópicos à seguir.

3.2 Local VPN

A camada Local VPN é composta por tecnologias que provem tunelamento entre origem e destino das informações de maneira segura, para isto foram utilizados duas VPNs individuais, uma dedicada a transitar dados do CCSD até a Infraestrutura interna da Cloud com destino ao banco de dados, e a outra dedicada a prover a comunicação entre a infraestrutura cloud até um CCSD específico, possibilitando o envio de comandos individuais por cliente e o controle modular.

A primeira VPN é hospedada por uma VM alocada na Azure, esta é responsável por estabelecer uma comunicação segura para aquisição de dados dos módulos de PMN diretamente com os CCSDs, já os CCSDs possuem APIs individuais para o envio de dados para a VM, o fluxograma detalhado da Local VPN pode ser visto no Fluxograma 5.

Fluxograma 5 - Fluxograma de operação do Control System Device com relação á Service VPN e a Cloud VPN



Fonte: Autoria própria (2023).

A API hospedada no CCSD, tem como uma das suas principais funções, receber todos os dados dos SMCD disponíveis na rede, e armazená-los em um

banco de dados local, bem como sincronizar os dados do banco de dados local com o banco de dados hospedado.

Para a realização dos testes apresentados, foi registrado previamente um *token* para autenticação da conexão do dispositivo de aquisição de dados, com a API do Raspberry PI.

O envio dos dados será feito no modelo de REST API, utilizando-se o método POST, para um *endpoint* local onde estará hospedado o Raspberry, e os dados serão enviados no corpo da requisição em formato JSON**.** Uma requisição de exemplo**,** pode ser vista abaixo**,** na Figura 9.

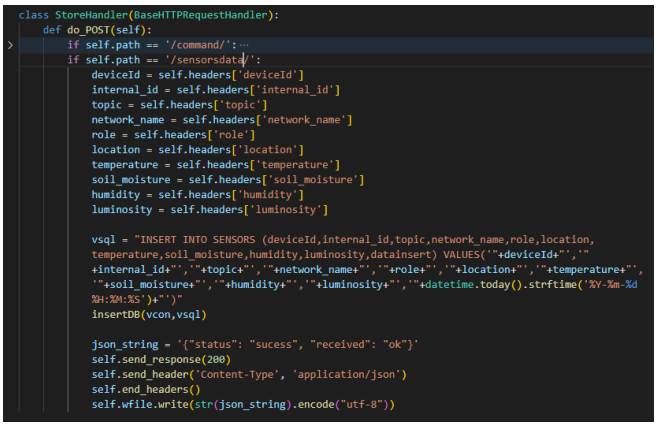
Figura 9 - Exemplo de request HTTP da API

```
curl --location --request POST 'https://local-raspberry.server.me/sensorsdata' \
--header 'Authorization: Bearer JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJleHAiOjE2NzQ1NjE3OTgsImlhdCI6MTY3NDQ3NTM5OCwic3ViIjoyMjg4NTUyMSwidXNlciI6eyJpZFVzdWFyaW8iOjIyODg1NT
IxLCJub211IjoiR0FCUklFTCBURVNUQSIsImxvZ2luIjoiZ2FicmllbC50ZXN0YUBncnVwb3ByZXZlcnN1bC5jb20uYnIiLCJlbWFp
bCI6ImdhYnJpZWwudGVzdGFAZ3J1cG9wcmV2ZXJzdWwuY29tLmJyIiwidGlwb1VzdWFyaW8iOiJfQURNIiwiYXRpdm8iOiJTIiwicH
JpdmlsZWdpb3MiOnt9LCJ0ZW1wb3Nlc3NhbyI6MTIwLCJkZGRDZWx1bGFyIjoiIiwidGVsQ2VsdWxhciI6IiIsImNvZGlnb0F0aXZh
Y2FvIjoiNzQ2NTkifX0.3kESaWhdNI1kjCFT_9CbT5Nj7QIbKgApRf7BBqc-xlo' \
--header 'Content-Type: application/json' \
--data-raw '{
"deviceId": "87589785",
"internal_id": 1,
"topic": "sensor_stream",
"network_name": "SENSORS_MESH_NETWORK",
"role": "sensor",
"location": "internal",
"temperature": 26,
"soil_moisture": 60,
"humidity": 74,
"luminosity": 92
}'
```

Fonte: Autoria própria (2023).

Para o servidor, foi implementada uma API simples para recebimento dos envios feitos pelos sensores, que efetua o roteamento de acordo com o método da requisição e o *endpoint*. Para os dados que chegarem com método POST com *endpoint* com terminação “/sensorsdata/”, os dados serão inseridos no banco de dados SQLite, e a resposta da requisição será *statuscode* 200 (sucesso), um trecho do comportamento mencionado pode ser visto na Figura 10.

Figura 10 - Exemplo de implementação de endpoint de api



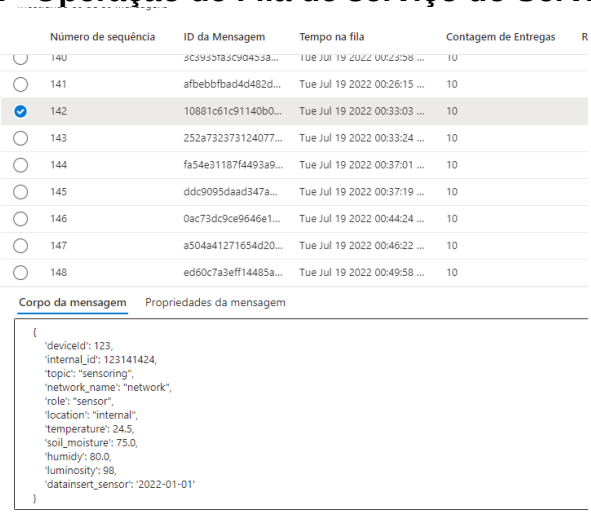
```
class StoreHandler(BaseHTTPRequestHandler):
    def do_POST(self):
        if self.path == '/command/':···
        if self.path == '/sensorsdata/':
            deviceId = self.headers['deviceId']
            internal_id = self.headers['internal_id']
            topic = self.headers['topic']
            network_name = self.headers['network_name']
            role = self.headers['role']
            location = self.headers['location']
            temperature = self.headers['temperature']
            soil_moisture = self.headers['soil_moisture']
            humidity = self.headers['humidity']
            luminosity = self.headers['luminosity']

            vsql = "INSERT INTO SENSORS (deviceId,internal_id,topic,network_name,role,location,
            temperature,soil_moisture,humidity,luminosity,datainsert) VALUES('"+deviceId+"','"
            +internal_id+"','"+topic+"','"+network_name+"','"+role+"','"+location+"','"+temperature+"',
            '"+soil_moisture+"','"+humidity+"','"+luminosity+"','"+datetime.today().strftime('%Y-%m-%d
            %H:%M:%S')+"')"
            insertDB(vcon,vsql)

            json_string = '{"status": "sucess", "received": "ok"}'
            self.send_response(200)
            self.send_header('Content-Type', 'application/json')
            self.end_headers()
            self.wfile.write(str(json_string).encode("utf-8"))
```

Fonte: Autoria própria (2023).

Passados 10 minutos, a placa Raspberry executa a rotina de carga para a *cloud*, fazendo o envio dos dados através do OpenVPN para o servidor na nuvem, que funcionará como um direcionador para o IoT *HUB,* que será detalhado no capítulo a seguir.

Os dados enviados então poderão ser visualizados no barramento de serviço do *Service Bus,* que será detalhado no capítulo a seguir, conforme pode ser visto na Figura 11, a parte superior da imagem demonstra os pacotes de dados alocados na fila do *Service Bus*, e a parte inferior demonstra a amostra de dados nº 142 da fila de processamento.

Figura 11 - Operação de Fila de serviço do Service Bus



	Número de sequência	ID da Mensagem	Tempo na fila	Contagem de Entregas	R
○	140	3c3935f3c9045 3a...	Tue Jul 19 2022 00:23:58 ...	10	
○	141	afbebbfbad4d482d...	Tue Jul 19 2022 00:26:15 ...	10	
●	142	10881c61c91140b0...	Tue Jul 19 2022 00:33:03 ...	10	
○	143	252a732373124077...	Tue Jul 19 2022 00:33:24 ...	10	
○	144	fa54e31187f4493a9...	Tue Jul 19 2022 00:37:01 ...	10	
○	145	ddc9095daad347a...	Tue Jul 19 2022 00:37:19 ...	10	
○	146	0ac73dc9ce9646e1...	Tue Jul 19 2022 00:44:24 ...	10	
○	147	a504a41271654d20...	Tue Jul 19 2022 00:46:22 ...	10	
○	148	ed60c7a3eff14485a...	Tue Jul 19 2022 00:49:58 ...	10	

Corpo da mensagem | Propriedades da mensagem


```
{
  'deviceId': 123,
  'internal_id': 123141424,
  'topic': "sensoring",
  'network_name': "network",
  'role': "sensor",
  'location': "internal",
  'temperature': 24.5,
  'soil_moisture': 75.0,
  'humidy': 80.0,
  'luminosity': 98,
  'datainsert_sensor': '2022-01-01'
}
```

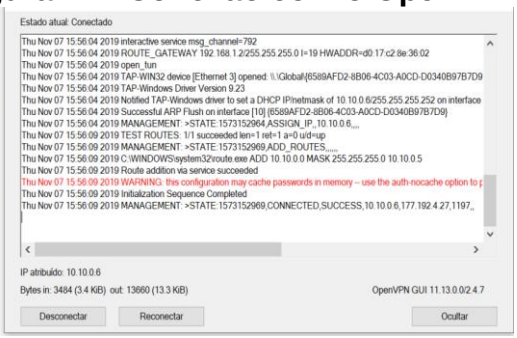

Fonte: Autoria própria (2023).

Inicialmente foi necessário configurar um servidor OpenVPN a partir da VM Linux provisionada na etapa anterior, que será a nossa autoridade certificadora. O servidor deverá ser capaz de gerar certificados, pares de chaves (pública e privada), bem como um arquivo de configuração para um cliente.

Ao servidor foi atribuído o IP “192.168.237.109” e a porta 8080. Para o pleno funcionamento também foi necessário configuração do *firewall* para permitir tráfego de dados entrando e saindo do servidor. Por fim, utilizamos o OpenVPN para gerar um arquivo de configuração para o cliente.

Como cliente utilizaremos um Raspberry PI 3B para transmitir dados para o servidor. No caso da configuração do cliente, foi necessário apenas transmitir o arquivo de configuração gerado pelo servidor de aplicação via SSH, e executar a configuração e execução automática da VPN conforme pode ser visto na Figura 12.

Figura 12 - Conexão com o Open VPN

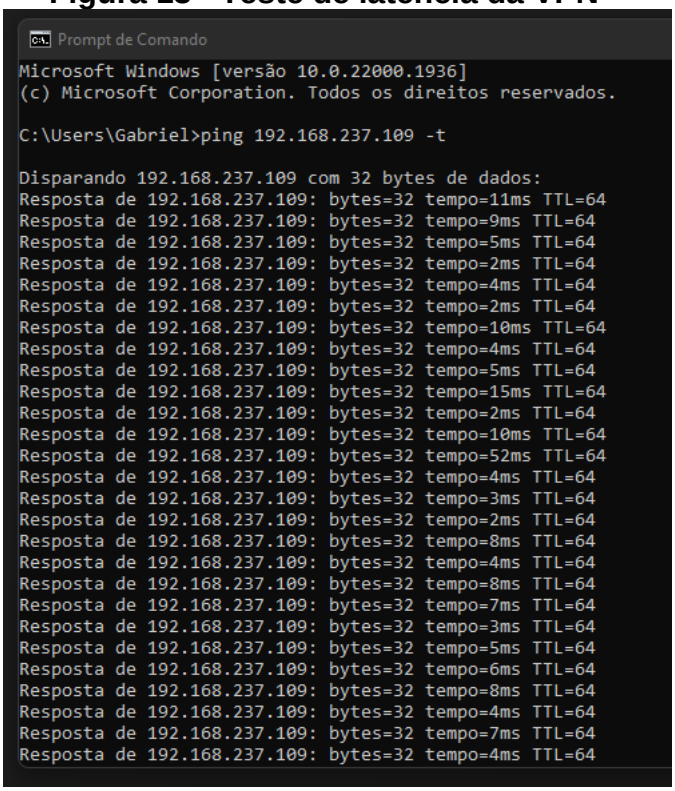


Fonte: Autoria própria (2023).

Para atestar o pleno funcionamento do tunelamento da VPN, utilizamos o comando *ping* no terminal do Raspberry, desta forma já é possível validar se o Servidor está acessível para o cliente através de um teste de *ping*. A resposta da execução do teste via terminal pode ser vista na Figura 13.

Para este trabalho, a placa Raspberry irá acumular dados durante 30 segundos, e transmitir os dados adquiridos para a infraestrutura *cloud*, de forma que em caso de indisponibilidade da rede, existem tratativas para não haver perda de informações. Para armazenar os dados de maneira a otimizar a memória, optou-se pela utilização do SQLite, que é um serviço de bancos de dados baseado em arquivos txt, bastante utilizado para armazenamento em cartões SD.

Figura 13 - Teste de latência da VPN

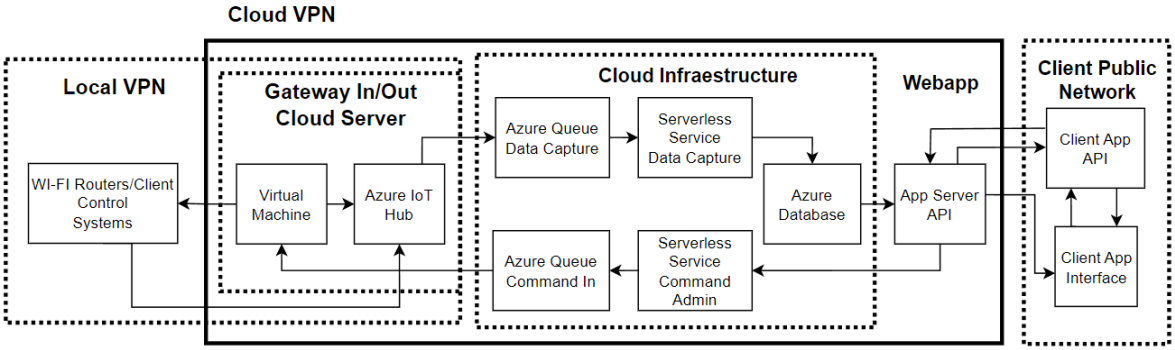


Fonte: Autoria própria (2023).

3.3 Cloud VPN.

Esta camada da arquitetura é responsável pelo recebimento e processamento do grande volume de informações. A entrada e saída de dados dos clientes é feita através de uma VM associado ao IoT *HUB,* o processamento de dados é composto filas de processamento do *Service Bus*, e códigos *serverless* que processam a fila de dados realizando a inserção no banco de dados central. Por fim, uma API que pode ser consultada por aplicativos mobile para exibição de dados. A arquitetura com os componentes mencionados pode ser vista no Fluxograma 6.

Fluxograma 6 - Fluxograma de Operação da infraestrutura proposta



Fonte: Autoria própria (2023).

Anteriormente foi mencionado que existem algumas etapas adicionais para que os dados adquiridos por sensores transitassem até o banco de dados hospedado na *Cloud* da Azure, a seguir será detalhado as tecnologias utilizadas para possibilitar o envio de grandes volumes de informação até a *web*.

3.3.1 Gateway *In/Out* Cloud Server.

O *Gateway In/Out*, mostrado na Figura 5, promove a entrada e saída de dados do cliente por meio de uma VM com OpenVPN associada ao serviço Azure IoT *HUB*.

O serviço Azure IoT *HUB* funciona como um broker de comunicação gerenciável que oferece rastreabilidade de dados, atuando como um roteador de mensagens para outros Serviços Azure, além de atuar como um proxy local.

A utilização de *Edge Computing* é muito comum em cenários onde é necessária computação na ponta, próximo a sensores, em redes internas e por vezes distribuídos na ponta como nesta implementação. A Microsoft oferece um serviço de *Edge Computing* através do IoT *Edge*, um *framework* baseado em *containeres* que permitem a gestão de módulos diretamente pelo IoT *HUB*, conferindo escalabilidade, robustez e facilidade de implementação.

Para aquisição e gerencia das informações que transitarem para dentro da infraestrutura *Cloud*, optou-se por usar o IoT *HUB*, um serviço de gerenciamento de dispositivos oferecido pela Azure, o mesmo oferece roteamento de mensagens para outros serviços da Azure de maneira confiável. O IoT *HUB* funciona como um *broker*

de comunicação gerenciável de dispositivos, oferecendo rastreabilidade dos dados que transitam pelo seu *pipeline* de entrada e saída, atuando também, como um *proxy* local para o componente IoT *HUB* que está centrado na nuvem.

Ao cadastrarmos um novo dispositivo no IoT *HUB*, o mesmo fornece uma chave única de conexão para que somente aquele dispositivo consiga se autenticar. Através desta chave, o IoT *HUB* identifica a origem da mensagem. Os metadados do dispositivo cadastrado podem ser vistos em formato JSON na Figura 14.

Figura 14 - Modelo de Dados de Cadastro de dispositivo do IoT HUB


```
[
    {
        "deviceId": "raspberry01",
        "etag": "AAAAAAAAAE=",
        "deviceEtag": "ODkyMjA2MTYz",
        "status": "enabled",
        "statusUpdateTime": "0001-01-01T00:00:00Z",
        "connectionState": "Disconnected",
        "lastActivityTime": "2022-07-08T03:47:19.0920204Z",
        "cloudToDeviceMessageCount": 0,
        "authenticationType": "sas",
        "x509Thumbprint": {
            "primaryThumbprint": null,
            "secondaryThumbprint": null
        },
        "modelId": "",
        "version": 2,
        "properties": {
            "desired": {
                "$metadata": {
                    "$lastUpdated": "2022-05-03T01:09:34.7695782Z"
                },
                "$version": 1
            },
            "reported": {
                "$metadata": {
                    "$lastUpdated": "2022-05-03T01:09:34.7695782Z"
                },
                "$version": 1
            }
        },
        "capabilities": {
            "iotEdge": false
        }
    }
]
```


Fonte: Autoria própria (2023).

Um dos grandes motivos pela escolha do IoT *HUB*, é a fácil integração com outros produtos oferecidos pela própria Microsoft, que já são consolidados em diversas empresas, alguns exemplos destes produtos são o *Active Directory Dynamics* 365 e *Power BI*. Visando escalabilidade do sistema, pode-se utilizar o IoT *HUB* para o roteamento dos dados para uma solução de enfileiramento de dados, para que nenhuma informação seja perdida em caso momentos de pico.

Já na etapa de saída de dados, para que as informações sejam enviadas para o Raspberry de destino, foi implementada uma VM na infraestrutura da Azure, operando com Ubuntu e hospedando uma VPN em OpenVPN. A necessidade de implementar uma VPN se deve à exigência de transitar de maneira segura e eficiente, sem expor os dispositivos locais na *web*. Dessa forma, foi gerado um *token*

único de autenticação da VPN para o Raspberry, garantindo que, caso um *token* seja comprometido por um código malicioso, isso não afete outros clientes que possuam chaves únicas.

Com o *token* único, o Raspberry foi conectado à VPN, permitindo que a VM esteja acessível para requisições HTTPS. Além da VPN, foi implementada uma API na VM que realiza a leitura da fila de processamento de comandos, e uma implementação adicional na API do Raspberry que monitora uma rota específica para recebimento de comandos. Após a leitura da fila, a API da VM encaminha, via requisição HTTPS POST, para o host apontado na mensagem como destino do comando, sendo, desta forma, foi enviado um comando de configuração dos atuadores.

3.4 Cloud Infraestructure.

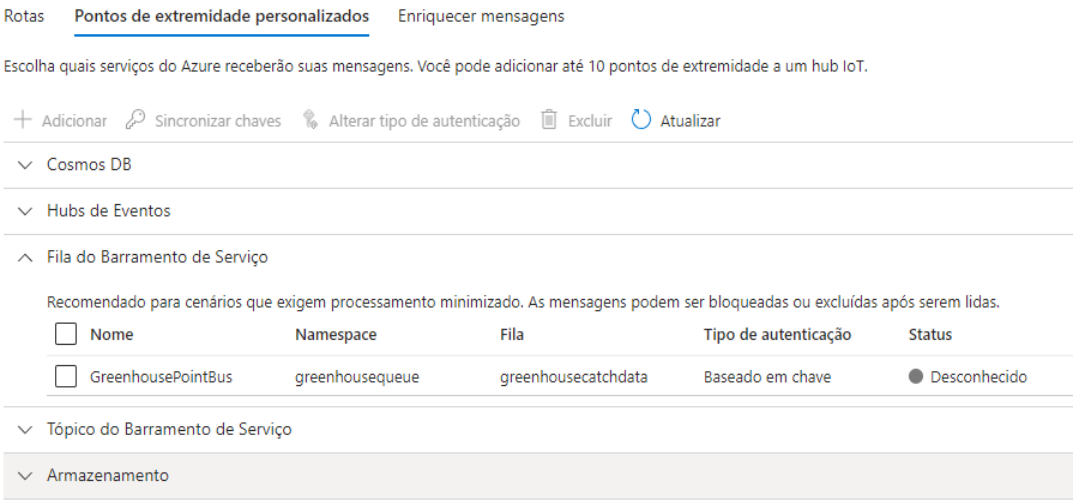
A *Cloud Infraestructure*, mostrada na Figura 5, é composta por tecnologias com capacidade de enfileirar dados de entrada e saída, mencionados como Azure *Queue Data Capture* e Azure *Queue Command In*, bem como códigos *serverless* capazes de processar informações, mencionados como *Service Data Capture* e *Service Command Admin*, além de inseri-las em um banco de dados. Mais detalhes destas tecnologias são descritos nos capítulos a seguir.

3.4.1 Azure Service Bus como Solução de Load Balance.

Para garantir que o servidor consiga processar todos os dados de sensoriamento dos diversos clientes de maneira escalável e evitando momentos de pico, optou-se pela utilização do Azure *Service Bus*. Este serviço da Azure possui integração direta com o IoT *HUB*, de forma que é possível de maneira nativa rotear todo o fluxo de entrada de dados do IoT *HUB* para filas específicas, que podem ter seu *hardware* escalado de maneira automática sob grandes volumes.

Para tal, foi configurado um ponto de extremidade personalizado, vinculado a uma fila específica para entrada de informações chamada “greenhousecatchdata” conforme pode ser visto abaixo nas Figuras 15 e 16.

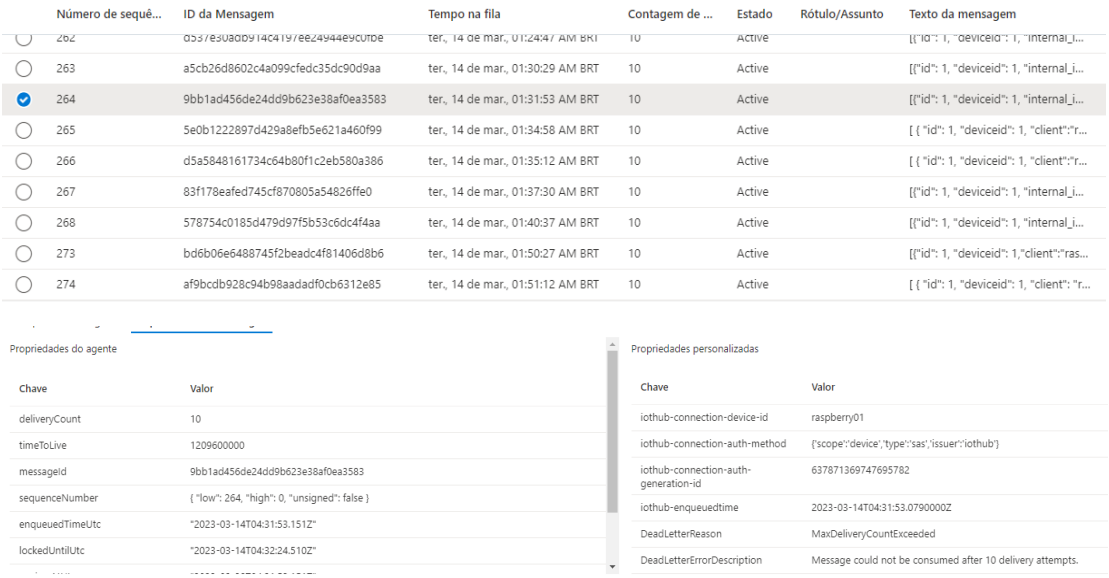
Figura 15 - Configuração de nó de extremidade do Service Bus



Fonte: Autoria própria (2023).

No contexto do uso do Azure *Service Bus*, é importante destacar a autenticação da comunicação entre o IoT *HUB* e o *Service Bus*, que é baseada em chave. Cada conexão para envio dos dados do IoT *HUB* para o *Service Bus* possui uma chave específica para aquele ponto de extremidade, o que possibilita a identificação da origem das informações, bem como do cliente responsável pelos dados. Essa abordagem permite que mais de um serviço insira informações no *Service Bus*, reduzindo o risco de perda da origem da informação e fornecendo rastreabilidade das origens, o que possibilita mensurar o consumo de cada cliente individualmente no *Service Bus*.

Figura 16 - Fila de serviço de barramento do Service Bus



Número de sequê...	ID da Mensagem	Tempo na fila	Contagem de ...	Estado	Rótulo/Assunto	Texto da mensagem
262	0537e30a0b914c4197ee24944e9c0fbe	ter., 14 de mar., 01:24:47 AM BRT	10	Active		[{"id": 1, "deviceid": 1, "internal_i...
263	a5cb26d8602c4a099cfedc35dc90d9aa	ter., 14 de mar., 01:30:29 AM BRT	10	Active		[{"id": 1, "deviceid": 1, "internal_i...
264	9bb1ad456de24dd9b623e38af0ea3583	ter., 14 de mar., 01:31:53 AM BRT	10	Active		[{"id": 1, "deviceid": 1, "internal_i...
265	5e0b1222897d429a8efb5e621a460f99	ter., 14 de mar., 01:34:58 AM BRT	10	Active		[{ "id": 1, "deviceid": 1, "client":"r...
266	d5a5848161734c64b80f1c2eb580a386	ter., 14 de mar., 01:35:12 AM BRT	10	Active		[{ "id": 1, "deviceid": 1, "client":"r...
267	83f178eafed745cf870805a54826ffe0	ter., 14 de mar., 01:37:30 AM BRT	10	Active		[{"id": 1, "deviceid": 1, "internal_i...
268	578754c0185d479d97f5b53c6dc4f4aa	ter., 14 de mar., 01:40:37 AM BRT	10	Active		[{"id": 1, "deviceid": 1, "internal_i...
273	bd6b06e6488745f2beadc4f81406d8b6	ter., 14 de mar., 01:50:27 AM BRT	10	Active		[{"id": 1, "deviceid": 1,"client":"ras...
274	af9bcdb928c94b98aadadf0cb6312e85	ter., 14 de mar., 01:51:12 AM BRT	10	Active		[{ "id": 1, "deviceid": 1, "client": "r...

Propriedades do agente

Chave	Valor
deliveryCount	10
timeToLive	1209600000
messageId	9bb1ad456de24dd9b623e38af0ea3583
sequenceNumber	{ "low": 264, "high": 0, "unsigned": false }
enqueuedTimeUtc	"2023-03-14T04:31:53.151Z"
lockedUntilUtc	"2023-03-14T04:32:24.510Z"

Propriedades personalizadas

Chave	Valor
iothub-connection-device-id	raspberry01
iothub-connection-auth-method	{'scope':'device','type':'sas','issuer':'iothub'}
iothub-connection-auth-generation-id	637871369747695782
iothub-enqueuedtime	2023-03-14T04:31:53.0790000Z
DeadLetterReason	MaxDeliveryCountExceeded
DeadLetterErrorDescription	Message could not be consumed after 10 delivery attempts.

Fonte: Autoria própria (2023).

Ademais, é importante mencionar outras possibilidades existentes, como enviar os dados do IoT *HUB* para um banco não relacional CosmosDB ou para um armazenamento genérico *Blob Storage*. Quando a CCSD envia um pacote de dados de sensores, o mesmo aparece na fila, permitindo o acesso a algumas propriedades e informações relevantes para a identificação das origens das mensagens. Entre essas informações, destacam-se o dispositivo relacionado àquele envio, o formato da mensagem e a quantidade de vezes que a mensagem foi removida da fila para tentativa de processamento. Com base nessa informação, é possível identificar mensagens com falhas de formatação ou inserções indevidas em caso de invasões da rede local.

Desta forma, sempre que a CCSD envia um pacote dados de sensores, o mesmo aparece na fila conforme pode ser visto na Figura 17.

Figura 17 - Dados recebidos pela fila de serviço do Service Bus

Fonte: Autoria própria (2023).

O processamento dos dados na fila do *Service Bus* é uma etapa crucial para o sucesso da aplicação. É importante que os dados sejam processados em tempo hábil, garantindo que as informações cheguem ao destino final de maneira adequada e eficiente.

Uma das principais vantagens de utilizar o *Service Bus* para o processamento de dados é a escalabilidade horizontal que ele oferece. Com essa funcionalidade, é possível aumentar a capacidade de processamento da aplicação de forma dinâmica e sob demanda, o que é especialmente útil em situações em que há picos de tráfego de dados.

Além disso, é possível utilizar recursos de processamento em lotes, o que permite o processamento de várias mensagens em uma única operação. Esse recurso é particularmente útil em casos em que o processamento individual de cada mensagem pode ser muito demorado ou complexo.

Outro aspecto relevante no processamento de dados na fila do *Service Bus* é a possibilidade de utilização de recursos de monitoramento e análise. Com esses recursos, é possível monitorar o desempenho da aplicação e identificar problemas de desempenho ou gargalos no processamento de dados. Dessa forma, é possível

tomar medidas corretivas para melhorar a eficiência e a qualidade do processamento de dados.

Por fim, é importante destacar que o *Service Bus* oferece recursos de segurança avançados, como autenticação, criptografia e controle de acesso. Esses recursos garantem que as informações armazenadas na fila sejam mantidas seguras e protegidas contra ameaças externas.

Até aqui, fizemos a aquisição dos dados e os mesmos foram ingeridos para dentro da nossa infraestrutura cloud, porém ainda se faz necessário processar estas informações e inseri-las em uma solução de armazenamento. Para tal, optou-se pela utilização de Azure Functions.

3.4.2 Azure Functions para Processamento de Dados da Fila.

A integração entre o *Service Bus* e as Azure Functions é feita por meio de um recurso chamado *Trigger* do *Service Bus*. Esse recurso permite que uma função seja disparada automaticamente sempre que uma nova mensagem é adicionada à fila do *Service Bus*.

Ao utilizar o *Trigger* do *Service Bus*, é possível configurar a função para processar a mensagem da fila de acordo com as necessidades da aplicação. Por exemplo, é possível realizar operações de transformação de dados, validação de conteúdo ou enriquecimento de informações.

Além disso, é possível utilizar outros recursos das Azure Functions para aumentar a eficiência do processamento de dados. Por exemplo, é possível utilizar o recurso de processamento em lote (*batching*) para processar várias mensagens de uma só vez, o que reduz o tempo total de processamento e diminui o custo operacional da aplicação.

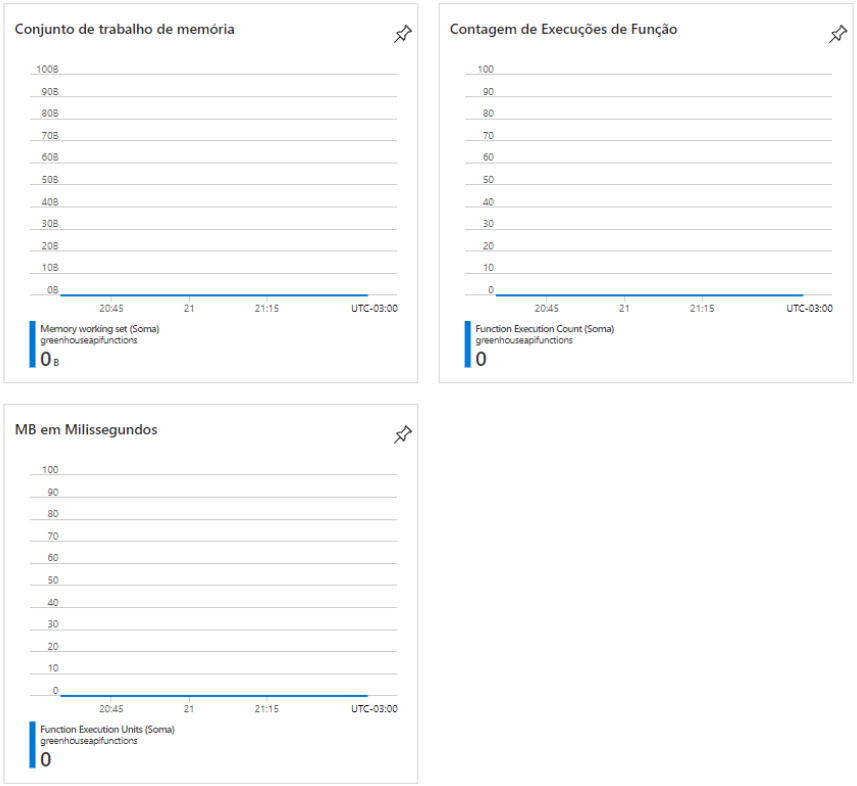
Outra vantagem da utilização de Azure Functions no processamento de dados da fila do *Service Bus* é a capacidade de integrar a função com outras ferramentas e serviços da plataforma Azure. Por exemplo, é possível utilizar o Azure *Stream Analytics* para realizar operações de análise em tempo real nos dados da fila, ou utilizar o Azure Cosmos DB ou o Azure *Blob Storage* para armazenar os dados processados.

Por fim, é importante destacar que a utilização de Azure Functions pode ajudar a reduzir significativamente os custos operacionais da aplicação, já que a plataforma é altamente eficiente e escalável, e não requer a configuração e manutenção de infraestrutura de servidores próprios. Dessa forma, a utilização de Azure Functions em conjunto com o *Service Bus* pode tornar o processo de processamento de dados mais eficiente, escalável e econômico.

A utilização de Azure Functions em conjunto com o Service Bus é uma abordagem bastante comum em aplicações que requerem o processamento de grandes volumes de dados em tempo real. Um exemplo de caso prático é o processamento de dados de sensores em uma rede IoT.

Nesse cenário, os dispositivos IoT enviam dados de sensores para um hub IoT, que envia esses dados para um tópico no *Service Bus*. Em seguida, uma Azure Function é acionada automaticamente para processar a mensagem da fila. A função pode executar tarefas como transformar os dados em um formato mais útil, validar as informações, enriquecer os dados com informações adicionais ou armazenar as informações em um banco de dados. O Azure Function disponibiliza uma ferramenta de monitoramento de suas atividades, de forma que podemos ter visibilidade de sua operação, a Figura 18 a seguir ilustra esse monitoramento.

Figura 18 - Ferramentas e métricas de relatórios do Azure Functions



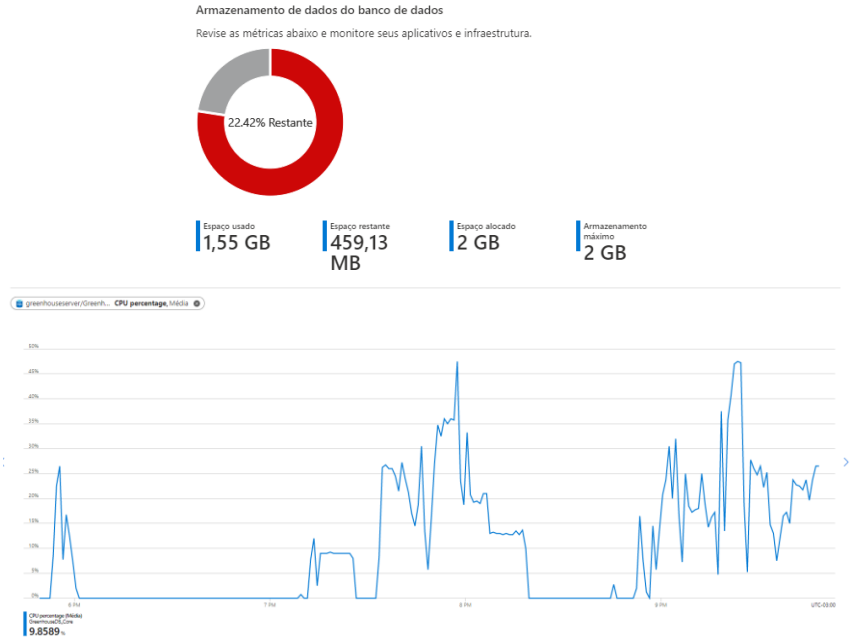
Fonte: Autoria própria (2023).

Para o processamento de dados da fila do servisse bus foi implementado uma Azure Function conforme arquitetura mostrada abaixo, onde o bloco "*Trigger*" indica a fonte de entrada que dispara a operação do código, "*Function*" é a nossa função em Python hospedada de maneira *serverless*, e sem nenhuma entrada ou saída.

3.4.3 Banco de dados SQL Server Hospedado na Azure como base central de dados.

A implementação de bancos de dados como serviço em ambientes de computação em nuvem apresenta diversas vantagens significativas, conforme discutido neste estudo. Uma das principais características favoráveis é a capacidade de escalabilidade do processamento e armazenamento com o simples acionar de um botão. Adicionalmente, essa abordagem oferece uma infraestrutura gerenciada eficiente para a realização de *backups* e a geração de relatórios de processamento, conforme ilustrado na Figura 19. Assim, é possível armazenar de maneira segura e organizada todas as informações pertinentes aos clientes e disponibilizá-las por meio de uma interface intuitiva e de fácil acesso.

Figura 19 - Relatórios de Bancos de dados cloud



Fonte: Autoria própria (2023).

3.5 WebApp

A API do *WebApp* é uma implementação de código *serverless* baseada em Azure Functions, atuando como interface de consulta e recebimento de comandos provenientes de um aplicativo móvel. A arquitetura *serverless* permite a execução de código em resposta a eventos e gerencia automaticamente os recursos computacionais, oferecendo alta escalabilidade e economia de custos. A funcionalidade central deste sistema é acionada por meio de solicitações HTTP, permitindo a comunicação entre o aplicativo móvel e a API. Essas solicitações HTTP são formas padronizadas de solicitar ou enviar dados a um servidor, com cada solicitação sendo tratada individualmente por uma instância da função Azure, assegurando uma resposta rápida e eficiente, mesmo sob alta demanda.

A exposição das funções Azure é feita por meio de um *endpoint*, que é basicamente um ponto final de uma comunicação. Esse *endpoint* é o URL onde o serviço pode ser acessado pelo aplicativo móvel. A API é projetada para ser acessível apenas mediante autenticação adequada, garantindo a segurança dos dados manipulados. Para a autenticação, utiliza-se um sistema de chave e valor. A chave é um identificador único, e o valor é uma *string* secreta associada à chave. Ao receber uma solicitação, a API verifica se a chave e o valor correspondentes foram

fornecidos e são válidos. Isso assegura que apenas usuários autorizados, que possuem a chave e o valor corretos, podem acessar a API e executar comandos.

Essa arquitetura garante um sistema altamente escalável e seguro, apto a lidar com a demanda variável e a necessidade de proteger os dados dos usuários que acessam o aplicativo móvel. Em suma, o *WebApp* API, com sua implementação baseada em Azure Functions, fornece uma solução robusta e eficiente para a gestão e a comunicação de dados em um ambiente de aplicativo móvel.

Dessa forma, o *WebApp* API é responsável por enviar os dados do banco de dados que será exibido no aplicativo móvel com os dados do cliente, bem como através dela será enviados os comandos para o drive de acionamento local.

Nos tópicos anteriores, detalhamos a infraestrutura relacionada à entrada de dados. O Fluxograma 5 apresenta um diagrama que ilustra o fluxo do serviço, desde o início do processo até o armazenamento dos dados no banco de dados SQL Server. Esse fluxo permite uma melhor compreensão das etapas envolvidas e das interações entre os diversos componentes do sistema proposto nesta pesquisa.

3.6 Client Public Network

Por meio da *Client Public Network*, os usuários podem acessar e interagir com o sistema usando um dispositivo Android ou iOS, ou seja, um tablet ou um celular, e executando um aplicativo dedicado ao monitoramento e controle da estufa. Através desse aplicativo móvel, o usuário é capaz de acessar dados de monitoramento, bem como enviar comandos usando os pontos finais da API mencionados anteriormente.

4 RESULTADOS E DISCUSSÕES

Neste capítulo detalharemos a metodologia de testes e os resultados. Três etapas foram realizadas para atestar o correto funcionamento do sistema. O primeiro teste demonstra o controle adequado dos atuadores dentro da estufa no modo automático, baseado em parâmetros ambientais predefinidos. O segundo teste avalia o processo de transmissão de dados para a nuvem. E, finalmente, o terceiro teste avaliou o envio direto de comandos para controle dos atuadores dentro da estufa no modo manual. Essas três séries de testes são explicadas nas próximas seções.

4.1 Procedimentos de simulação de dados e envio para os atuadores do sistema no modo automático

Para os testes de desempenho de envio de dados, foram gerados dados semelhantes a um cenário real de monitoramento, para tal, foram desenvolvidos códigos para geração e inserção de dados em um banco de dados de teste. Para a geração foram considerados faixas pré-definidas de temperatura máxima e mínima semelhante ao bioma brasileiro, bem como uma variação máxima entre uma amostra ou outra, reduzindo o efeito de aleatoriedade completa, e simulando as não idealidades dos sensores considerados para o trabalho.

A metodologia adotada para gerar dados sintéticos de temperatura em um ambiente de estufa envolve várias etapas, conforme detalhado nos Apêndices A a F. O Apêndice A apresenta o código-fonte em Python utilizado para estabelecer uma conexão com o banco de dados 'GreenhouseDB_Core' em um servidor remoto, fornecendo as credenciais necessárias, como o endereço do servidor, nome de usuário, senha e nome do banco de dados.

O modelo matemático empregado no código do Apêndice A utiliza parâmetros como a temperatura máxima e mínima, o número de dias para gerar dados, a variação máxima, o fator de amplitude e o fator de escala, que são ajustáveis e podem ser modificados para simular diferentes condições e cenários.

O código também inclui duas funções, 'temperature_curve' e 'variation_factor', que são usadas para calcular a temperatura em função da hora do dia. A função

'temperature_curve' retorna o valor ajustado para as temperaturas máximas e mínimas ocorrerem nos horários especificados, utilizando a função seno, enquanto a função 'variation_factor' utiliza a função ‘valor_absoluto’ do seno para ponderar a variação ao longo do dia.

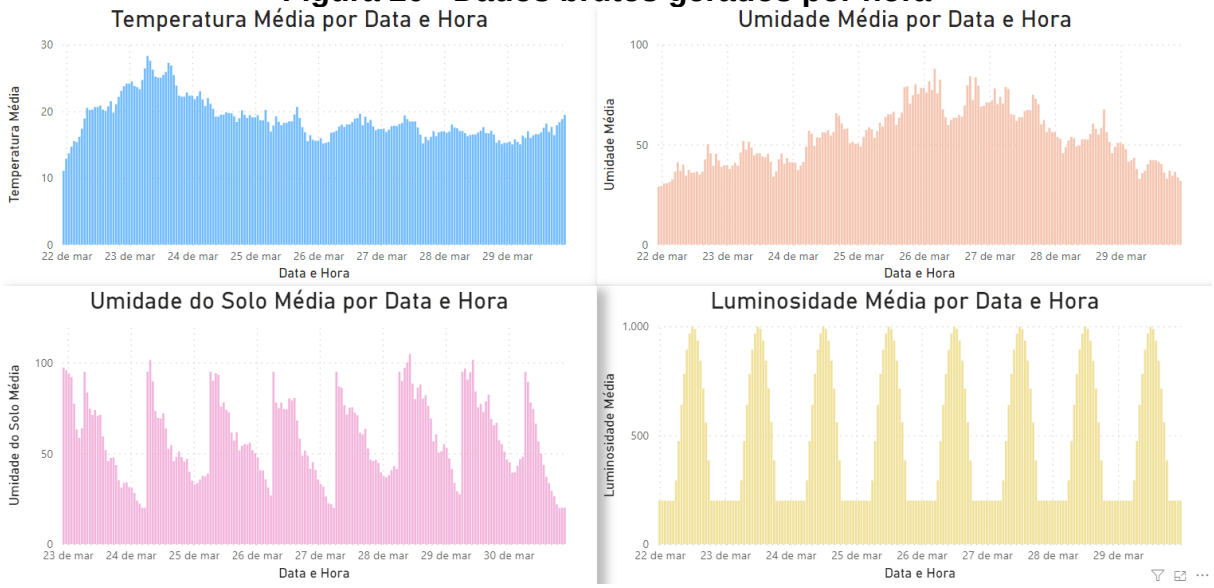
O código do Apêndice A utiliza um loop duplo para iterar sobre os dias e horas do período definido, que neste caso são 7 dias. Para cada hora, calcula-se a temperatura com base nas funções 'temperature_curve' e 'variation_factor', aplica-se a variação máxima e gera-se uma temperatura aleatória dentro do intervalo permitido. Essa temperatura aleatória simula flutuações naturais na temperatura devido a fatores como variações climáticas e práticas de manejo.

Os valores de temperatura gerados pelo código são inseridos na tabela 'temperaturas' do banco de dados 'GreenhouseDB_Core' utilizando a função 'cursor.execute' da biblioteca 'pymssql'. O processo é repetido para cada hora do período de sete dias. Ao término da inserção de todos os dados, a conexão com o banco de dados é encerrada.

A metodologia empregada, descrita neste texto e ilustrada no código do Apêndice F, permite a geração de dados sintéticos de temperatura para ambientes de estufa, possibilitando análises e estudos de diferentes cenários e condições sem a necessidade de experimentos em campo. Os parâmetros do modelo podem ser ajustados para simular condições específicas ou investigar o impacto de diferentes práticas de manejo na dinâmica da temperatura.

O mesmo foi feito para umidade do ar, luminosidade e umidade do solo, cujos os resultados são obtidos de dados gerados ao longo de 7 dias, e de hora em hora, conforme podem ser vistos a seguir, na Figura 20.

Figura 20 - Dados brutos gerados por hora



Fonte: Autoria própria (2023).

Como nosso objetivo é a obtenção de dados a cada segundo, será necessário algum tratamento neles, para que sejam obtidos de maneira semelhante a um cenário real. Para isto foi utilizada uma interpolação linear para expandir os dados de hora em hora, para minuto em minuto conforme função do Apendice F.

Para realizar essa conversão, foi desenvolvida uma função SQL, apresentada no Apêndice F, denominada dbo.ExpandHourlyDataToMinuteData().

A função dbo.ExpandHourlyDataToMinuteData() utiliza interpolação linear para estimar os valores de temperatura em intervalos de um minuto a partir dos dados de temperatura por hora. O Apêndice B contém o código SQL completo da função, que inclui a criação de uma tabela temporária, a inserção dos dados nessa tabela e a aplicação de um *loop WHILE* para iterar sobre as linhas da tabela temporária.

A interpolação linear adotada na função mantém a temperatura constante em cada intervalo de 1 minuto, gerando-se assim, uma representação mais detalhada dos dados de temperatura, ao longo do tempo. Essa abordagem permite a análise de flutuações mais sutis na temperatura, que podem ser essenciais para a compreensão do comportamento da estufa e para a identificação de possíveis melhorias no controle do ambiente.

Após a definição da função no Apêndice D, o código cria uma nova tabela chamada ‘temperaturaMinutos’, que armazena os dados de temperatura por minuto

interpolados a partir dos dados por hora. A função dbo.ExpandHourlyDataToMinuteData() é então chamada e os resultados são armazenados na tabela ‘temperaturaMinutos’. Esta tabela é posteriormente consultada para a análise e visualização dos dados interpolados.

Após expandir os dados de temperatura de hora em hora para dados de minuto a minuto, o próximo passo é realizar uma interpolação para obter os valores em intervalos de 60 segundos. O código SQL para essa etapa está disponível no Apêndice E.

O código do Apêndice E utiliza uma técnica conhecida como "Window Function" em conjunto com a função ROW_NUMBER(), gerando uma sequência de números que são utilizados para calcular os intervalos de 60 segundos. Este processo é realizado separadamente para umidade do ar, luminosidade e umidade do solo, criando-se tabelas com os dados interpolados chamadas ‘UmidadedoarSegundosFinal’, ‘luminosidadeSegundosFinal’ e ‘umidadeSegundosFinal’.

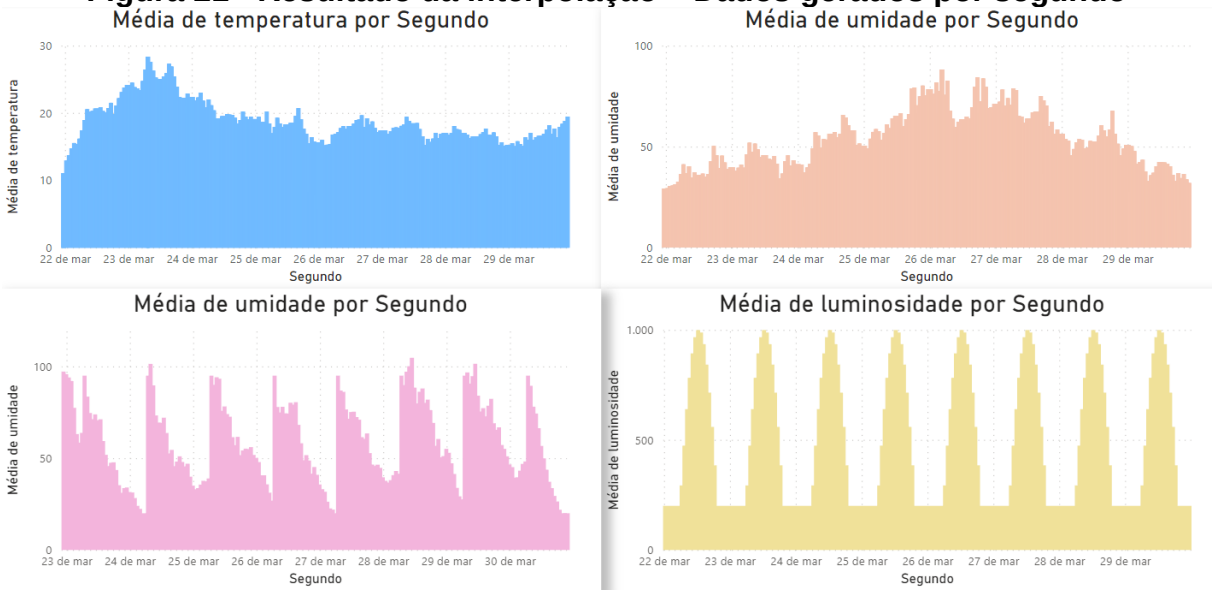
A interpolação consiste em, para cada instante de tempo, calcular um novo registro a cada 60 segundos e ajustar a umidade do ar, luminosidade e umidade do solo de acordo com o tempo decorrido. A técnica utilizada mantém os valores das variáveis constantes em cada intervalo de 60 segundos, fornecendo uma representação mais detalhada das variações das condições ambientais ao longo do tempo. O resultado das duas etapas de tratamento de dados pode ser visto na Figura 21.

Os sensores utilizados para medir as variáveis ambientais, como temperatura, umidade e luminosidade, geralmente apresentam não idealidades que podem afetar a precisão dos dados coletados. Estas não idealidades são, na maioria das vezes, devidas às imperfeições na fabricação, envelhecimento e degradação dos componentes ou mesmo interferências externas. Portanto, é importante considerar esses fatores ao analisar os dados coletados e ao tomar decisões baseadas nessas informações.

Nesta etapa, o objetivo é simular as não idealidades dos sensores ao introduzir ruído nos dados interpolados, conforme apresentado no código do Apêndice F. O ruído é adicionado por meio da função RAND(CHECKSUM(NEWID())), que gera um valor aleatório entre -1 e 1.

Multiplicando esse valor por uma constante, é possível controlar a quantidade de ruído a ser adicionada aos dados.

Figura 21 - Resultado da interpolação – Dados gerados por segundo



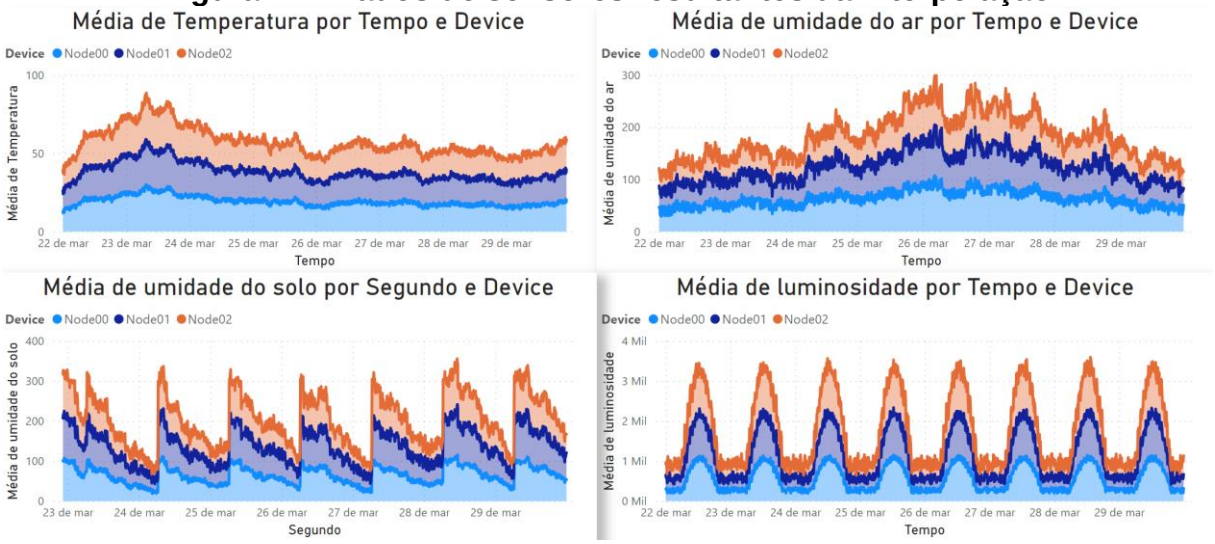
Fonte: Autoria própria (2023).

No código fornecido, o ruído é aplicado a cada variável ambiental:

- Para a temperatura, o ruído é ajustado para valores dentro do intervalo de ±2 graus Celsius conforme 1% de *Span* definido pelo fabricante do DHT11.
- Para a umidade do solo e umidade do ar, o ruído varia dentro de um intervalo de ±5% em relação aos valores originais conforme 10% de *Span* definido pelo fabricante do YL-69.
- Para a luminosidade, o ruído introduzido varia em um intervalo que depende do valor máximo de luminosidade em cada nó, podendo variar em ±10% ou 20% de *Span*.

A partir do código, obtemos uma simulação de dados para 3 nós de medição por parâmetro climático. Essa simulação de não idealidades nos sensores permite uma análise mais realista dos dados coletados, e ajuda a entender como os erros de medição podem afetar o monitoramento das condições ambientais da estufa. Ao considerar essas incertezas, é possível desenvolver estratégias de controle e manejo mais robustas e adaptativas para enfrentar os desafios apresentados pelas variações nas condições ambientais. O resultado pode ser visto na Figura 22.

Figura 22 - Dados de sensores resultantes da interpolação



Fonte: Autoria própria (2023).

No presente estudo, a metodologia empregada para testar o sistema de sensoriamento em estufas, consistiu em enviar um grande volume de dados (através do *pipeline* de dados) até o banco de dados. Como ilustrado na Figura 22, geramos dados para três nós de medição, a fim de facilitar a visualização dos mesmos. Caso houvesse mais nós com menos informações, a representação gráfica do trânsito de dados se tornaria mais complexa.

Posteriormente, os dados foram pré-carregados em cartões SD nos nós de medição, e o código foi modificado para que, a cada medida, ao invés de se adquirir as informações diretamente dos sensores, os próprios nós de medição, pudessem acessar uma amostra da pilha de dados, gerados previamente. Assim, a cada segundo, cada nó de medição é responsável por enviar uma amostra dos dados pré-carregados.

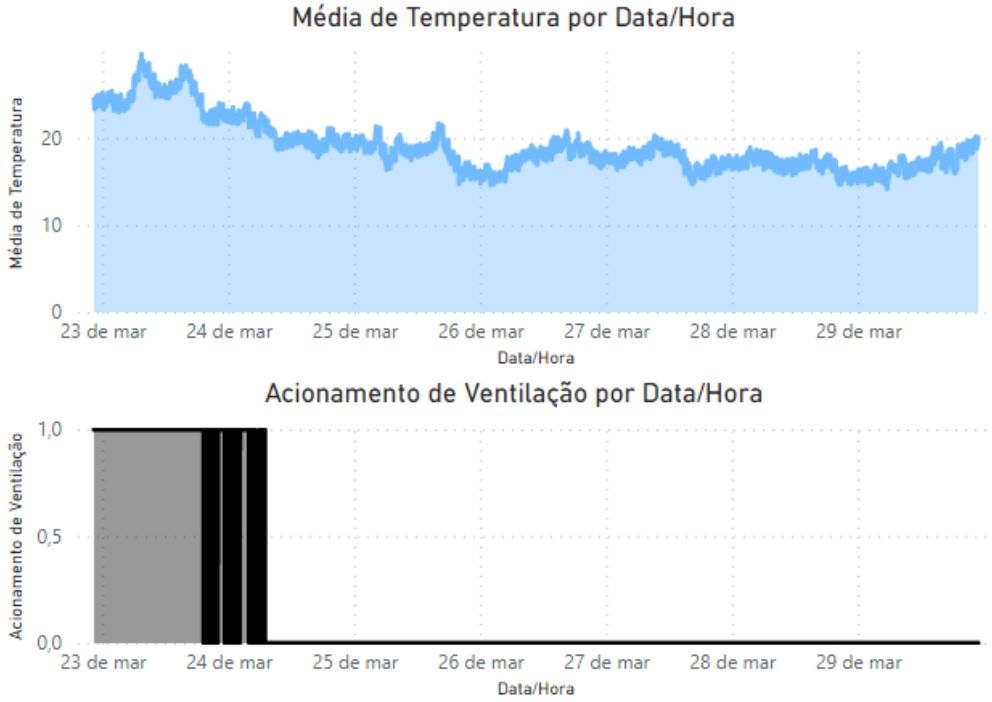
A primeira fase dos testes, envolveu a desativação da interface que envia os dados para a nuvem, permitindo avaliar o desempenho da rede local *mesh* na transmissão dos dados para o *Gateway* e analisar o comportamento dos atuadores com base nos dados recebidos. Para tanto, o sistema da rede *mesh* foi ajustado conforme mencionado anteriormente, de modo a trabalhar com os dados de amostra.

Inicialmente, o atuador calcula a média dos parâmetros entre os três sensores de cada nó, determinando se deve ou não acionar um atuador. Os parâmetros considerados no teste foram:

- Temperatura Máxima: 23ºC.
- Luminosidade Mínima: 300 lúmens.
- Umidade do ar Mínima: 40%.
- Umidade do solo Mínima: 40%.

Após o envio das informações, obtivemos os resultados para a temperatura, conforme apresentado na Figura 23. O primeiro gráfico representa a média entre as três medições, enquanto o segundo gráfico exibe o acionamento da ventilação nos momentos em que a temperatura ultrapassa 22ºC. No gráfico do acionamento, é possível observar que os períodos em que o acionamento se mantém constante são representados pela parte mais escura, enquanto a cor mais clara indica a presença de flutuações no acionamento, ou seja, o acionamento e desligamento dos ventiladores ocorrem de maneira descontrolada quando os dados apontam proximidade ao limiar do acionamento. Esse comportamento pode ser melhor visualizado ao analisar isoladamente os dias 23 e 24 de março de 2023, conforme mostrado na Figura 23.

Figura 23 - Acionamento da ventilação ao longo do dia

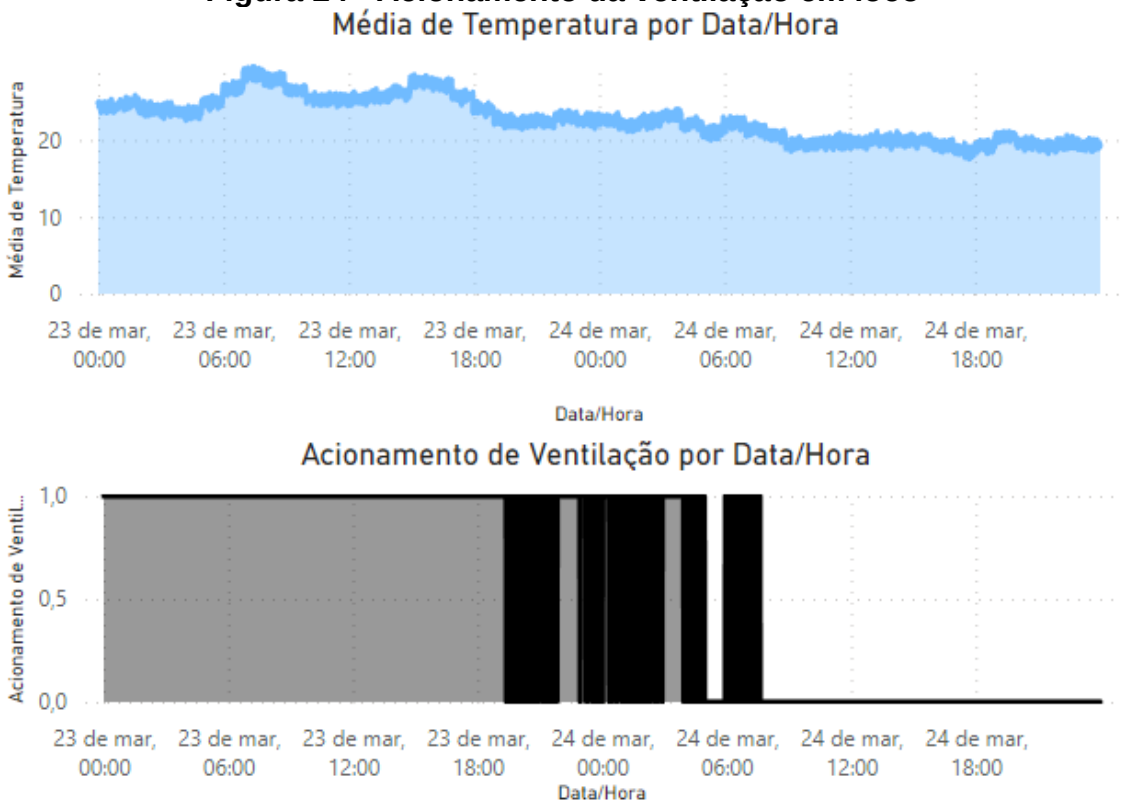


Fonte: Autoria própria (2023).

É possível observar que, entre as 19h do dia 23 e as 8h do dia 24, ocorrem momentos em que a temperatura oscila entre 21 e 22ºC. Nesses momentos, o acionamento e o desligamento da ventilação ocorrem conforme a média se move acima ou abaixo de 22ºC, resultando em uma ventilação menos eficiente com a presença de repique de acionamentos.

Em relação à umidade do ar, os resultados podem ser visualizados na Figura 24. Nota-se que o acionamento ocorre em momentos específicos ao longo dos dias e se prolonga por algumas horas quando a umidade fica abaixo de 40%. No entanto, as oscilações no acionamento são menos frequentes do que as observadas no caso da temperatura.

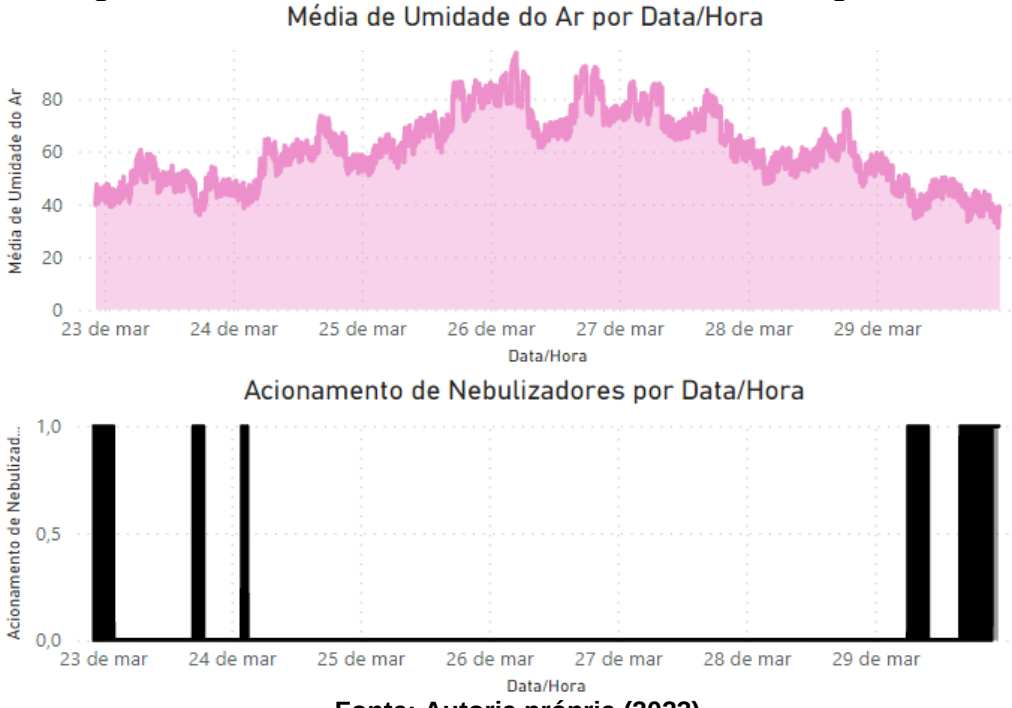
Figura 24 - Acionamento da ventilação em loco



Fonte: Autoria própria (2023).

Quanto à umidade do ar, os resultados estão representados na Figura 25.

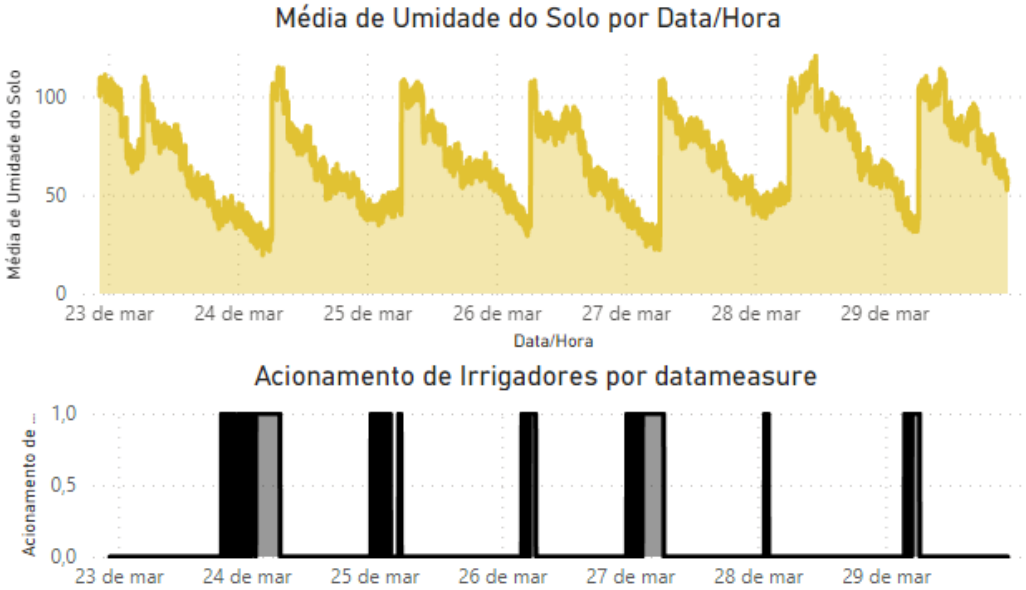
Figura 25 - Acionamento dos nebulizadores ao longo do dia



Fonte: Autoria própria (2023).

Quanto à umidade do solo, os resultados estão representados na Figura 26.

Figura 26 - Acionamento dos irrigadores ao longo do dia

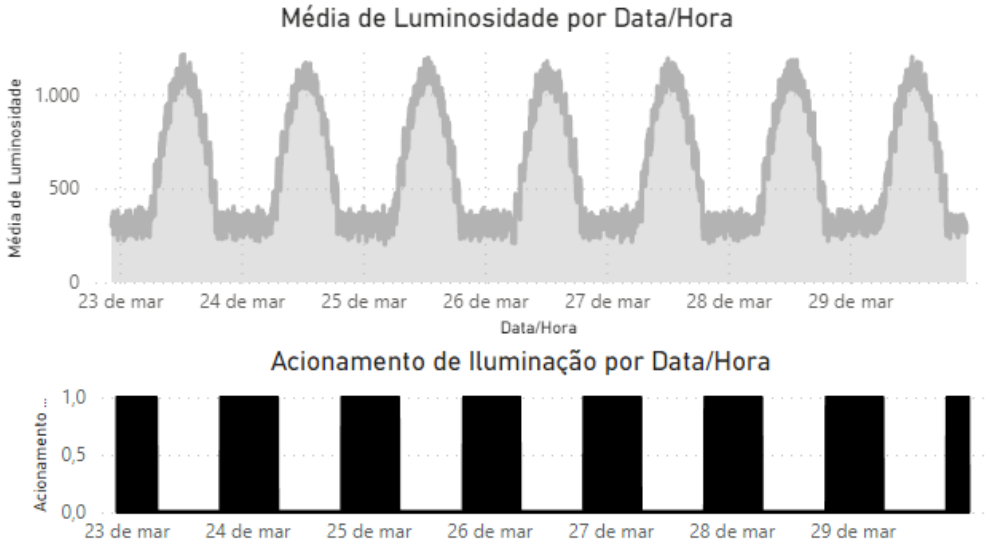


Fonte: Autoria própria (2023).

Neste cenário, é possível observar que os irrigadores seriam acionados em momentos específicos durante a madrugada para aumentar a umidade, seguidos pela irrigação regular programada para as 8h da manhã. Nesse caso, também é possível identificar a presença de oscilações no acionamento dos irrigadores.

Em relação à luminosidade, os resultados podem ser visualizados na Figura 27. Neste cenário, não foram observadas oscilações no acionamento das lâmpadas. Elas são acesas no momento em que há ausência de luz solar e permanecem ligadas até o amanhecer.

Figura 27 - Acionamento da iluminação ao longo dos dias

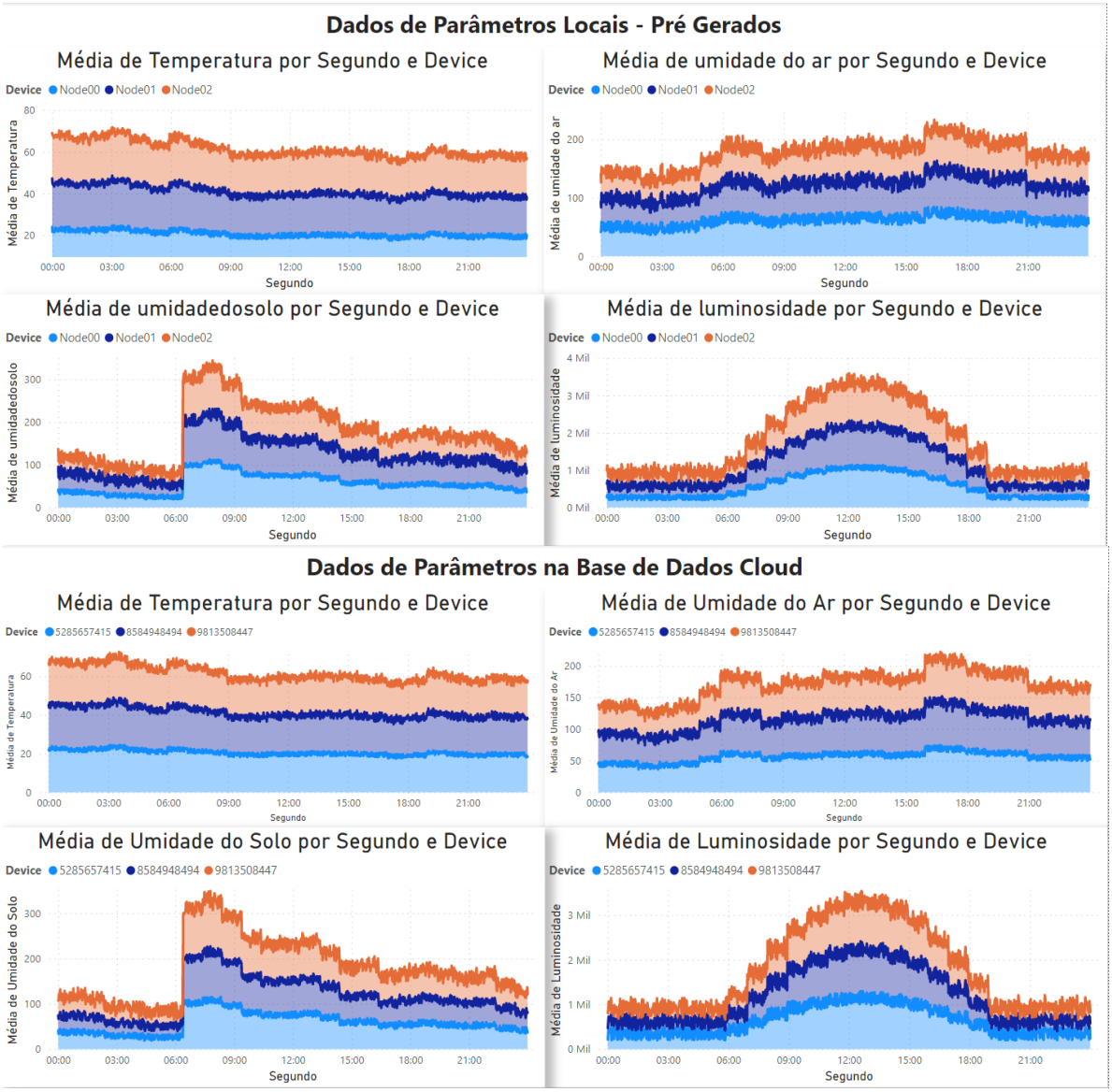


Fonte: Autoria própria (2023).

4.2 Procedimentos de avaliação de performance no envio de dados para a Cloud

Conforme as informações apresentadas até o momento, observa-se que a transmissão de dados dos nós para o *Gateway*, assim como os acionamentos dos atuadores, estão em conformidade com a proposta inicial. O próximo passo deste estudo envolve não apenas o envio de dados para o *Gateway*, mas também o encaminhamento desses dados para a nuvem, com o objetivo de comparar os resultados obtidos. Para a realização desse teste, optou-se por selecionar um único dia para o envio das informações à plataforma em nuvem. A comparação entre os dados obtidos a partir da base de dados em nuvem e a amostra original pode ser visualizada na Figura 28.

Figura 28 - Comparativo de dados originais com dados transitados através do sistema



Fonte: Autoria própria (2023).

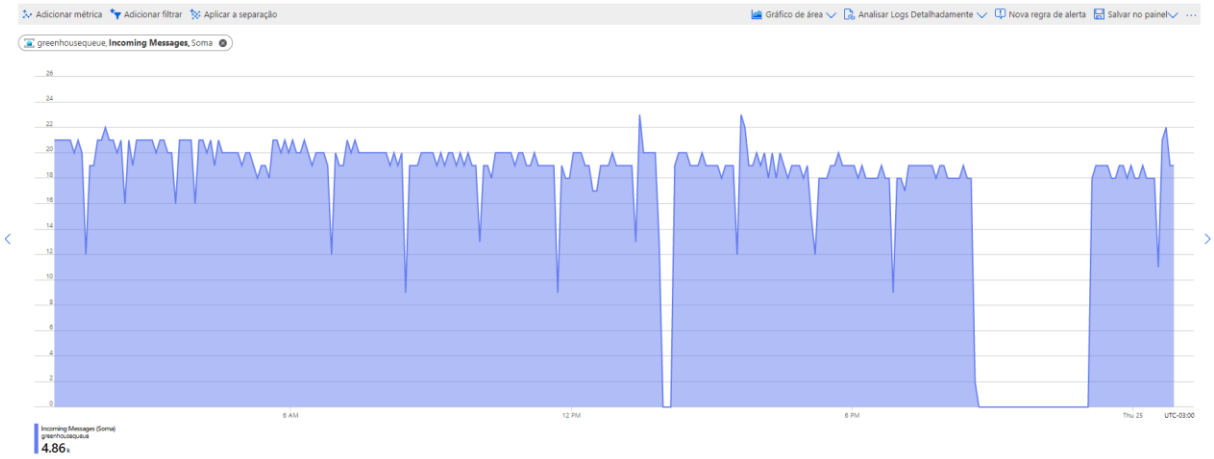
Observou-se que algumas amostras foram perdidas durante o transporte. Essas falhas foram identificadas por meio do cruzamento direto com as amostras originais. Notou-se que, durante a etapa de sincronização entre a base de dados do Raspberry e a nuvem, a API, devido à maior carga de trabalho, fica mais propensa a não conseguir processar todas as requisições enviadas, resultando em falhas durante o trânsito dos dados.

Durante as primeiras tentativas de realizar-se o experimento, houve dificuldades quanto à desconexão do dispositivo da rede, por conta de alta latência em decorrência da distância em relação ao routeador de internet. Para resolver a

situação, o código foi implementado com melhorias de tratamento de falhas e novas tentativas de conexão. Durante os resultados abaixo, poderemos observar Gaps no recebimento dos dados nos serviços *cloud*. Após a realização destas melhorias, foi possível realizer o experimento, por 24h initerruptas.

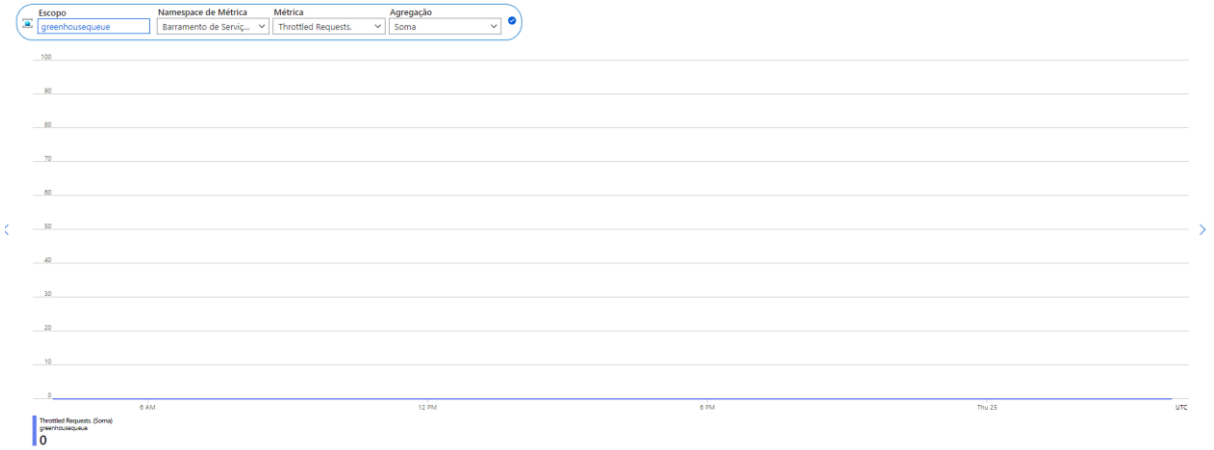
Ao analisar os registros de logs do IoT *HUB* e do *Service Bus*, constatou-se que não ocorreram gargalos ou falhas em decorrência de suas operações. A Figura 29 ilustra a carga de trabalho do *Service Bus*, e a Figura 30 apresenta as falhas identificadas durante a operação.

Figura 29 - Relatório de quantidades de mensagens do Service Bus



Fonte: Autoria própria (2023).

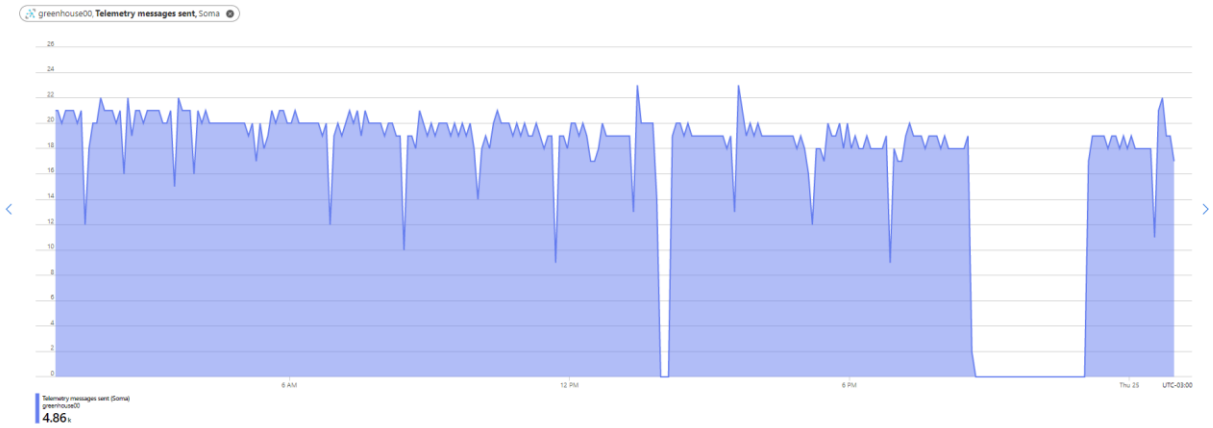
Figura 30 - Relatório de falhas de recebimento da fila de serviço do Service Bus



Fonte: Autoria própria (2023).

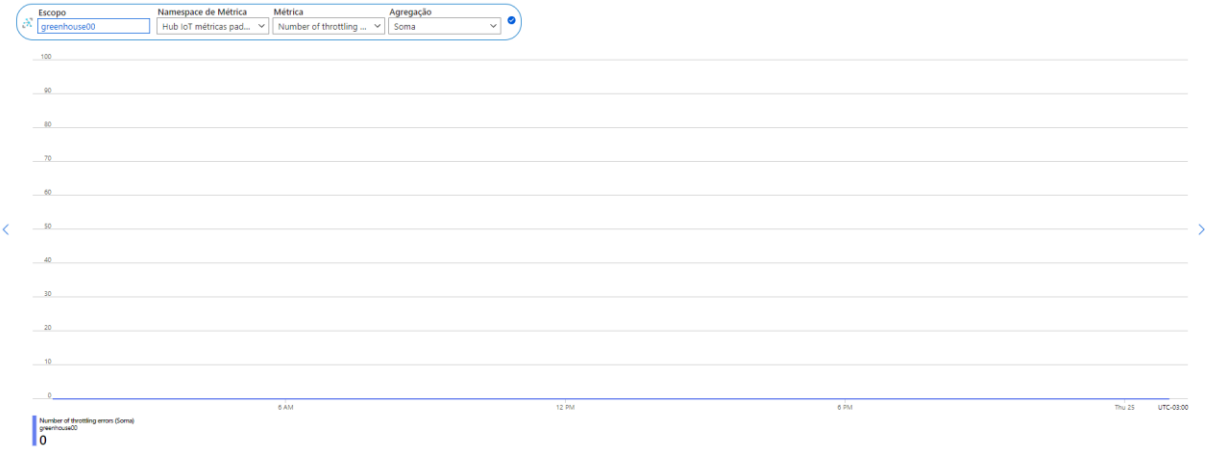
Para o IoT *HUB*, aplicaram-se as mesmas métricas com o objetivo de validar a operação bem-sucedida. A Figura 31 ilustra a quantidade de mensagens transitadas pelo IoT Edge, enquanto a Figura 32 apresenta a quantidade de falhas identificadas pelo *HUB,* a Figura 33 demonstra os momentos de conexão do cliente ao servidor do IoT *HUB*.

Figura 31 - Relatório de mensagens enviadas através do IoT HUB



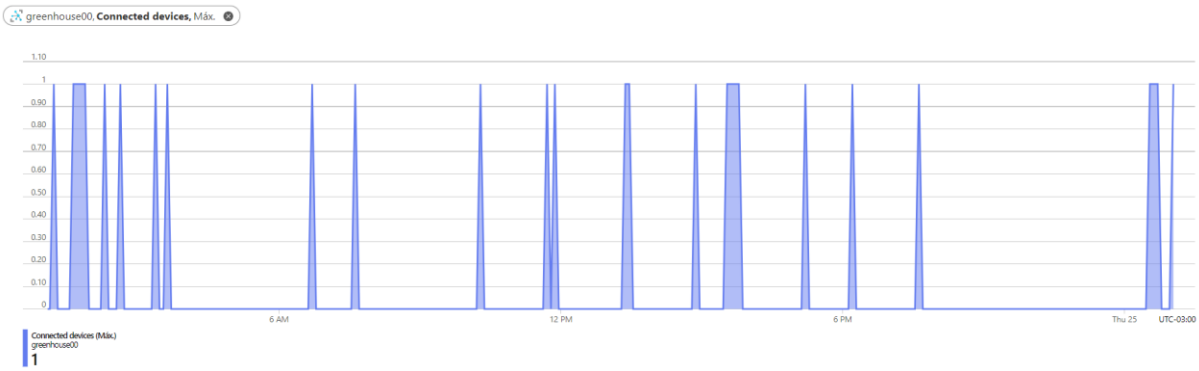
Fonte: Autoria própria (2023).

Figura 32 - Relatório de falhas do IoT HUB



Fonte: Autoria própria (2023).

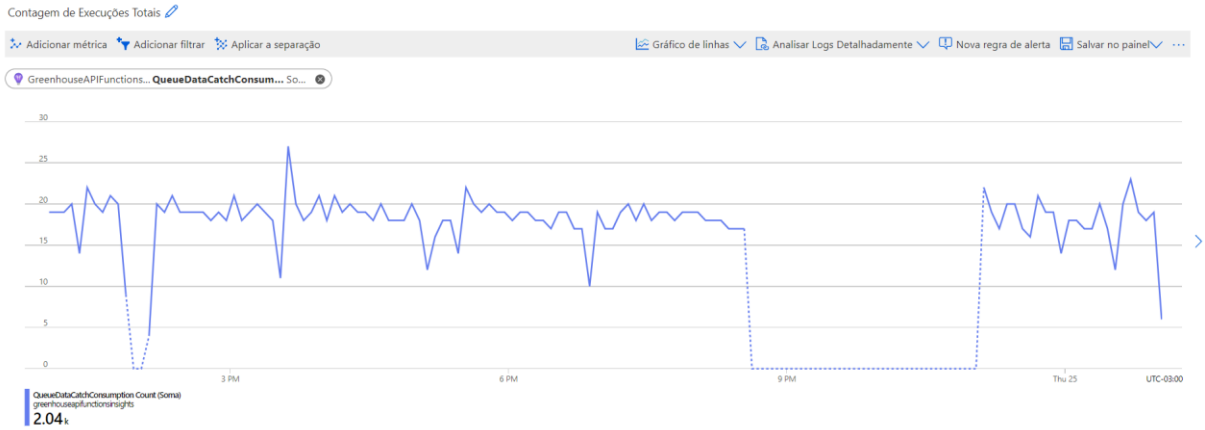
Figura 33 - Relatório de conexões de dispositivos do IoT HUB



Fonte: Autoria própria (2023).

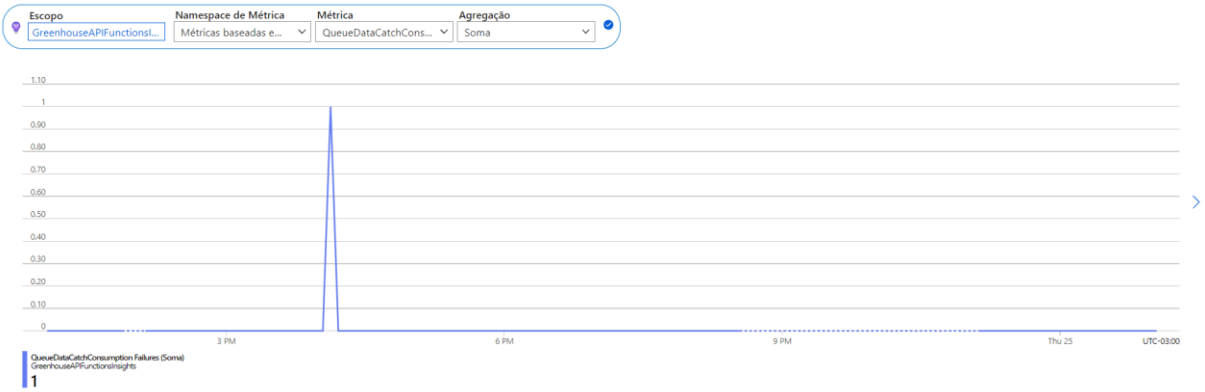
O Azure Functions também disponibiliza relatórios de sua utilização. Para o experimento aqui apresentado, pode-se observar que não ocorreram falhas em decorrência do processamento das mensagens, conforme ilustrado nas Figuras 34 e 35, a seguir.

Figura 34 - Relatório de execuções da Azure Functions



Fonte: Autoria própria (2023).

Figura 35 - Relatório de falhas de processamento do Azure Functions



Fonte: Autoria própria (2023).

4.3 Procedimentos de envio de comandos para os atuadores

Na etapa de envio de dados para o controlador dos atuadores, foi implementada uma API com Azure Functions com gatilho de acionamento do tipo request HTTP POST. Esta possui a função de receber os dados do comando e do host de destino, e escalar essa informação para uma fila de processamento de comandos, e em seguida consulta a fila para verificar se a mensagem foi processada para confirmar o envio com sucesso. A fila de processamento foi implementada de maneira semelhante à fila de processamento de dados de entrada em Azure *Service Bus*. O formato de envio dos dados para acionamento é em formato JSON, conforme pode ser visto na Figura 36.

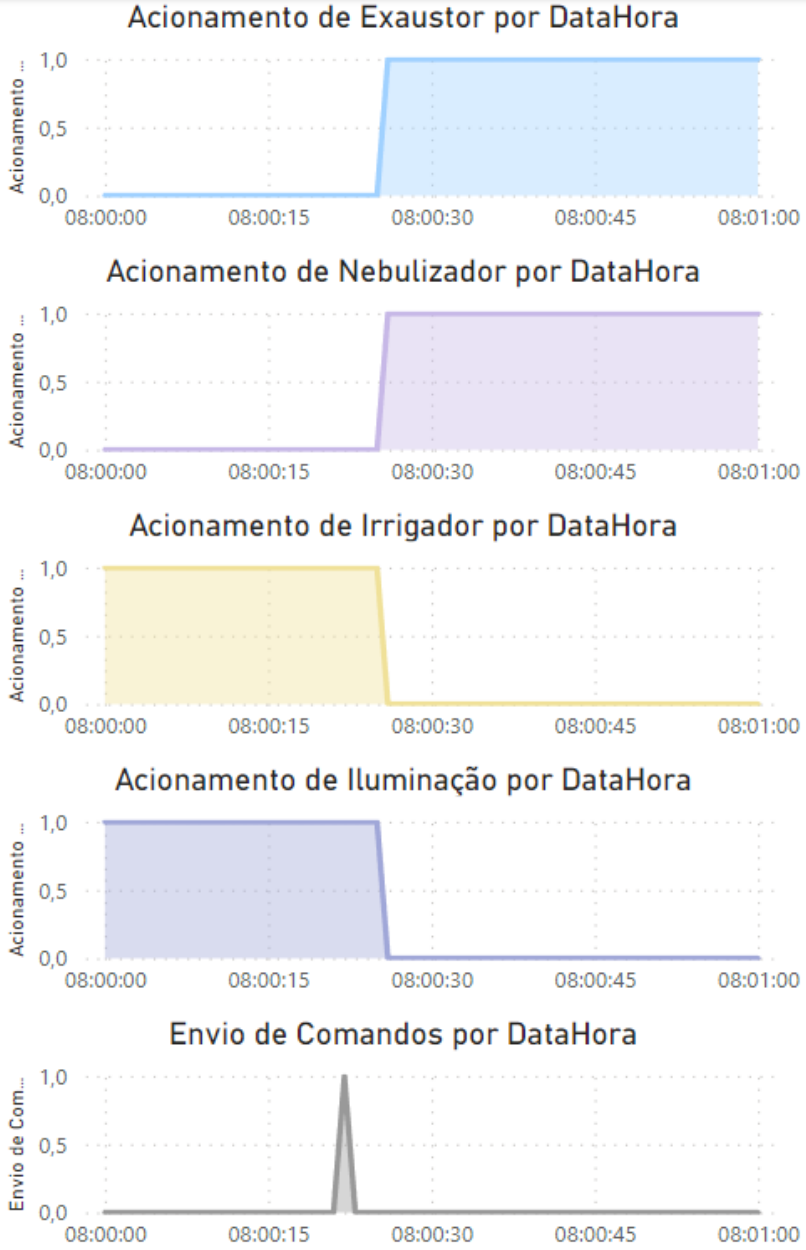
Figura 36 - Modelo de dados de comando de acionamento

```
[
    {
        "ID": 1,
        "datainsert": "2022-07-06 00:49:43",
        "device_client": "agricultor01",
        "device_name": "raspberry01",
        "exhaustion": false,
        "illumination": true,
        "irrigation": true,
        "pulverization": false,
        "topic": "command"
    }
]
```

Fonte: Autoria própria (2023).

Para o ensaio realizado, o comando foi transmitido para o endpoint da API às 8 horas e 22 segundos, sendo recebido pelo Raspberry às 8 horas e 26 segundos. Isso representa um atraso de aproximadamente 4 segundos entre a emissão do comando e o início da aplicação da nova configuração. É importante salientar que este ensaio não foi conduzido sob condições de stress, como no caso da transmissão de informações. Assim, este tempo de atraso pode variar dependendo da disponibilidade dos serviços de nuvem e da carga de trabalho no momento. Contudo, os resultados obtidos neste teste são considerados bastante satisfatórios, conforme mostra a Figura 37.

Figura 37 - Acionamento dos atuadores frente a envio de comando



Fonte: Autoria própria (2023).

4.4 Melhorias Propostas ao Modelo

Durante a implementação deste projeto, identificaram-se diversas oportunidades para futuras melhorias. As seguintes sugestões são considerações resultantes de observações realizadas durante a operação do sistema:

Em primeiro lugar, notou-se que algumas informações se perdem durante o trânsito para a nuvem. Para aumentar a eficiência do sistema, sugere-se a

otimização do código com rotinas de identificação de falha e retentativa, de modo a evitar a saturação do Raspberry durante momentos de sincronização de dados com a nuvem. Além disso, considera-se a possibilidade de substituir o Raspberry Pi 3B, utilizado neste trabalho, por uma versão com maior capacidade de hardware. Associado a isso, a falta de conexão com a internet por períodos prolongados pode levar ao esgotamento do armazenamento do Raspberry, interrompendo completamente sua operação. Uma maior disponibilidade de hardware poderia mitigar esse problema.

O sistema demonstrou robustez ao lidar com um volume significativo de informações. Uma melhoria futura que contribuiria para a completude do sistema seria a inclusão de um conjunto de câmeras que capturem imagens periodicamente ou quando solicitado via aplicativo. Esse recurso seria de grande auxílio para o controle interno das estufas.

Durante os testes, identificou-se que o formato JSON utilizado para o envio de mensagens tem um custo significativo em termos de tamanho das mensagens em kilobytes. Para uma maior economia de tráfego de dados na nuvem, sugere-se a adoção de um formato de informação mais compacto, como GZIP ou PARQUET.

Para maior sustentabilidade, o sistema pode ser incrementado com microgeração utilizando painéis solares para sua operação.

O modelo de dados atualmente usado para enviar comandos aos atuadores envolve o envio de uma configuração completa de todos os atuadores disponíveis no sistema. Nesse modelo, é necessário definir um estado de operação para todos os atuadores em um único comando. Para trabalhos futuros, uma melhoria seria estruturar o modelo de dados para permitir o envio de comandos individuais para cada atuador, tornando mais eficaz o controle de um único atuador, sem interferir potencialmente na operação dos outros.

O sistema proposto foi utilizado para estufas agrícolas, porém a infraestrutura comporta aplicações em outros setores, apenas com alteração das estruturas de dados transitadas, oferecendo um bom potencial para aplicações de automação residencial, ou industrial não críticas.

5 CONCLUSÃO

Em conclusão, este estudo de mestrado abordou a crescente competitividade no setor agrícola e a necessidade de adotar tecnologias avançadas para otimizar a utilização dos recursos naturais. Nesse contexto, as técnicas de sensoriamento remoto e a expansão das metodologias IoT, aliadas às tecnologias *Cloud* em várias áreas, têm ganhado destaque.

Este trabalho apresentou uma proposta inovadora, integrando tecnologias de sensoriamento aplicadas ao cultivo controlado em estufas, proporcionando o acesso às informações ambientais da estufa em um ambiente remoto através da internet. A pesquisa realizada contribuiu para o avanço do conhecimento na área, bem como para a aplicação prática dessas tecnologias no contexto agrícola brasileiro.

Foram identificadas diversas oportunidades para melhorias futuras e expansão do escopo do trabalho, incluindo a criação de um menu interativo para seleção dos parâmetros desejados pelo agricultor, a integração completa das bases de monitoramento via internet com Raspberry pi e a adaptação do projeto para aplicações em outros setores além da agricultura, como a indústria e a automação residencial.

A avaliação dos resultados alcançados e das possibilidades de aprimoramento permitiu concluir que o monitoramento e o controle das condições ambientais em estufas agrícolas são executados de maneira eficiente e eficaz, apesar de algumas perdas na transmissão de dados e variação na latência, proporcionando informações relevantes e acessíveis ao usuário do sistema. Este estudo de mestrado, portanto, representa um passo importante para o desenvolvimento e a implementação de soluções inovadoras no campo do sensoriamento remoto e IoT aplicados ao setor agrícola, contribuindo para a sustentabilidade e competitividade do setor, no contexto nacional.

REFERÊNCIAS

AKYILDIZ, I. F. WANG, X. AND WANG, W. Reless mesh networks: a survey. Computer Networks and ISDN Systems, Amsterdam, v. 47, n. 4, p. 445–487, 2005.

ALLEN, M.R. et al. Framing and Context. *In*: Global Warming of 1.5°C. An IPCC Special Report on the impacts of global warming of 1.5°C above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change, sustainable development, and efforts to eradicate poverty. Cambridge University Press, Cambridge, UK and New York, NY, USA, 2023. Disponível em: https://www.ipcc.ch/sr15/chapter/chapter-1/. Acesso em: 12 jul. 2023.

ANDRADE, André Luiz L.; OLIVEIRA, Pedro Burached de; CALDAS FILHO, Francisco Lopes de; DIAS, Ugo Silva; SOUSA JÚNIOR, Rafael Timóteo de; ALBUQUERQUE, Robson de Oliveira. Estudo de soluções VPN site-to-site segundo as técnicas criptográficas empregadas. In: Conferências IADIS Ibero-Americanas WWW/Internet e Computação Aplicada, 2019, Brasília. Anais […]. Brasília: Universidade de Brasília - Departamento de Engenharia Elétrica, 2019.

ARAÚJO, Clezyane Correia. Sustentabilidade da monocultura do milho em assentamentos rurais no município de Simão Dias – SE. 2018. 121 f. Dissertação (Mestrado em Desenvolvimento e Meio Ambiente) - Universidade Federal de Sergipe, São Cristóvão, SE, 2018.

AZEVEDO, Rodrigo Otávio Silva de. Sistema de gerenciamento remoto de dispositivos de Internet das Coisas protegidos por uma Rede Virtual Privada. 2019. Trabalho de Conclusão de Curso (Graduação em Engenharia de Telecomunicações) - Universidade Federal Fluminense, Niterói – RJ, 2019.

VANELLI, B. et al., "Internet of Things Data Storage Infrastructure in the Cloud Using NoSQL Databases," in IEEE Latin America Transactions, vol. 15, no. 4, pp.

737-743, April 2017, doi: 10.1109/TLA.2017.7896402.

BAILEY, B.J. Control and monitoring of glasshouses. Proceedings of the UK Controlled Environment UsersGroup. vol. 13, pp. 2-5, 2002.

BOTTINI, F.J. 2022. Improving Greenhouse Gases Market-Based Mitigation Programs: A Case Study of Renovabio – the Brazilian Renewable Fuels Program. Master's thesis, Harvard University Division of Continuing Education.

COELHO, V.; ALMEIDA, T.; PEREIRA, L.; SANTOS, I.; DINARDI, P.; VERMEHREN, V.; VALENZUELA, W. Implementação de um Sistema de Sensoriamento Ultrassônico para Aplicações em IOT no Contexto de Smart Cities. XLI International Sodebras Congress. [online]. v. 10, n. 169, p. 163-167, jan. 2020. Disponível em: http://www.sodebras.com.br/edicoes/N169.pdf. Acesso em: 29 jul. 2020.

COUTO, Ana Flávia Matos. Sistema de monitoramento de parâmetros fisiológicos e ambientais utilizando rede mesh. 2021. Dissertação (Mestrado em Engenharia Elétrica) - Faculdade de Engenharia do Campus de Ilha Solteira, UNESP, 2021.

CRITTEN, D. L.; BAILEY, B. J. A Review of Greenhouse Engineering Developments During the 1990s. Agricultural and Forest Meteorology, v. 112, p. 1-22, 2002.

CUCE, E.; HARJUNOWIBOWO, D.; CUCE, O. M. Renewable and sustainable energy saving strategies for greenhouse systems: A comprehensive review. Renewable and Sustainable Energy Reviews, v. 64, p. 34-59, 2016.

FARINHA, Micael José de Sousa. Solução Headless REST API para e-commerce. 2019. Dissertação (Mestrado em Engenharia Informática – Computação Móvel) - IPL Escola Superior de Tecnologia e Gestão - Instituto Politécnico de Leiria, Leiria, 2019.

GALVEIAS, Pedro Martins Gomes Valsassina. Azure2SOC: Integração de Tecnologias de Segurança MS AZURE no Ecossistema do CyberSOC da Altice

Portugal. 2021. Dissertação (Mestrado em Segurança Informática) - Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, Lisboa, 2021.

HEYMAN, Karen. A new virtual private network for today's mobile world. Computer, v. 40, n. 12, p. 17-19, 2007.

Kosnik, Lea-Rachel. "Cap-and-trade versus Carbon Taxes: Which Market Mechanism Gets the Most Attention?" Climatic Change 151.3-4 (2018): 605-18. Web.

LI, G.; ZHANG, W.; ZHANG, Y. A Design of the IOT Gateway for Agricultural Greenhouse. Sensors & Transducers, v. 172, n. 6, p. 75-80.

MAIA, Jorge Andrade Seixas. Arquitetura base para soluções de Internet das Coisas: Aplicações de telemetria e computação na ponta com uso de Microsoft Azure nos modelos de IaaS, PaaS e SaaS. 2020. Dissertação (Mestrado) - Universidade de Brasília, Brasília, 2020.

MENDES, Diogo dos Santos. Desenvolvimento de infraestrutura de software do projeto VITASENIOR. 2019. Relatório (Mestrado em Engenharia Informática – Internet das Coisas) - Instituto Politécnico de Tomar, Tomar, 2019.

MICROSOFT. Azure IoT Hub: Conecte, monitore e gerencie bilhões de ativos de IoT. Disponível em: https://azure.microsoft.com/pt-br/products/iot-hub. Acesso em: 11 de Maio de 2023.

PARRONCHI, Pietro. The Development Pioneers and the New Agriculture 4.0: economic development from the countryside? 2017. Disponível em: https://www.researchgate.net/profile/Pietro-Parronchi-2/publication/321918194_Os_Pioneiros_do_desenvolvimento_e_a_Nova_Agricultura_40_desenvolvimento_economico_a_partir_do_campo_The_Development_Pioneers_and_the_New_Agriculture_40_economic_development_from_the_countryside/links/5b23b28aa6fdcc6974658603/Os-Pioneiros-do-desenvolvimento-e-a-Nova-Agricultura-40-desenvolvimento-economico-a-partir-do-campo-The-Development-Pioneers-and-the-New-Agriculture-40-economic-development-from-the-

countryside.pdf. Acesso em: [12 de Julho de 2023].

RODRIGUES, Leandro Emanuel Almeida. Implementação de uma RESTful API para votação eletrônica. 2020. Dissertação (Mestrado em Segurança Informática) - Faculdade de Ciências e Tecnologia da Universidade de Coimbra, Departamento de Engenharia Informática, Coimbra, 2020.

SHAMSHIRI, R. R.; KALANTARI, F.; TING, K. C.; THORP, K. R.; HAMEED, I. A.; WELTIZIEN, C.; AHMAD, D.; SHAD, Z. M. Advances in greenhouse automation and controlled environment agriculture: A transition to plant factories and urban agriculture. Int J Agric Biol Eng, v. 11, n. 1, pp. 1-22, 2018.

SHAMSHIRI, R.; ISMAIL, W. I. W. A Review of Greenhouse Climate Control and Automation Systems in Tropical Regions. Journal of Agricultural Science and Applications, v. 2, pp. 176-183, 2013.

SILVA, J. L. C., VIDAL, C. A. S., BARROS, L. M., & FREITA, F. R. V. (2018). ASPECTOS DA DEGRADAÇÃO AMBIENTAL NO NORDESTE DO BRASIL. Revista Gestão & Sustentabilidade Ambiental, 7(2), 180–191. https://doi.org/10.19177/rgsa.v7e22018180-191

TESTA, G. H.; BERTOGNA, E. G. Automatização de Estufas para Cultivo Controlado Utilizando Redes Mesh. TCC, UTFPR, Campo Mourão, BR, 2019.

TUCOWS. Howtomechatronics, Arduino Wireless Network with Multiple NRF24L01 Modules. [Online]. Disponível em: https://howtomechatronics.com/tutorials/arduino/how-to-build-an-arduino-wireless-network-with-multiple-nrf24l01-modules/. Acesso em: 27 abr. 2020.

APÊNDICE A - Código Fonte para geração de dados de temperatura.

```
import random
import pymssql
from datetime import datetime, timedelta
import math

# Define as configurações de conexão com o banco de dados
server = ''
user = ''
password = ''
database = ''

# Conecta ao banco de dados
conn = pymssql.connect(server, user, password, database)

max_temperature = 34
min_temperature = 15
days_to_generate = 7
max_variation = 0.1
amplitude_factor = 2
scale_factor = 1

def temperature_curve(hour):
    # Ajusta os valores para que as temperaturas máximas e mínimas ocorram nos horários especificados
    return amplitude_factor * math.sin((hour - 3) * math.pi / 12)

def variation_factor(hour):
    # Utiliza a função valor absoluto do seno para ponderar a variação ao longo do dia
    return scale_factor * abs(math.sin((hour - 1) * math.pi / 12))


start_date = datetime.now()
end_date = start_date + timedelta(days=days_to_generate)

current_date = start_date
previous_temperature = None
while current_date <= end_date:
    for hour in range(0, 24):
        temp_range = max_temperature - min_temperature
        temp_variation = (temperature_curve(hour) + 1) / 2 * temp_range
        base_temperature = min_temperature + temp_variation

        if previous_temperature is None:
            temperature = base_temperature
        else:
```

```
                current_max_variation = max_variation *
variation_factor(hour)
                min_allowed_temp = max(min_temperature,
previous_temperature - current_max_variation *
previous_temperature)
                max_allowed_temp = min(max_temperature,
previous_temperature + current_max_variation *
previous_temperature)
                temperature =
random.uniform(min_allowed_temp, max_allowed_temp)

            # Atualiza a temperatura anterior
            previous_temperature = temperature

            # Insere os dados no banco de dados
            cursor = conn.cursor()
            cursor.execute('INSERT INTO temperaturas (hora,
temperatura) VALUES (%s, %s)', (current_date, temperature))
            conn.commit()

            # Incrementa uma hora
            current_date += timedelta(hours=1)

    # Fecha a conexão com o banco de dados
    conn.close()
```

APÊNDICE B - Código fonte para geração de dados de luminosidade.

```
import random
import pymssql
from datetime import datetime, timedelta
import math

# Define as configurações de conexão com o banco de dados
server = ''
user = ''
password = ''
database = ''

# Conecta ao banco de dados
conn = pymssql.connect(server, user, password, database)

max_luminosity = 1000
min_luminosity = 200
days_to_generate = 7

def luminosity_curve(hour):
    # Ajusta os valores para que a luminosidade máxima
ocorra ao meio-dia e a mínima à meia-noite
    if hour < 5.5 or hour > 19:
        return min_luminosity
    else:
            return min_luminosity + (max_luminosity -
min_luminosity) * math.sin((hour - 5.5) * math.pi / (19 -
5.5))
# Implementa a lógica aqui
start_date = datetime.now()
end_date = start_date + timedelta(days=days_to_generate)

current_date = start_date
while current_date <= end_date:
    for hour in range(0, 24):
        base_luminosity = luminosity_curve(hour)
        luminosity = max(min_luminosity, base_luminosity)
        # Insere os dados no banco de dados
        cursor = conn.cursor()
         cursor.execute('INSERT INTO luminosidade (hora,
luminosidade) VALUES (%s, %s)', (current_date, luminosity))
        conn.commit()

        # Incrementa uma hora
        current_date += timedelta(hours=1)
# Fecha a conexão com o banco de dados
conn.close()
```

APÊNDICE C - Código fonte para geração de dados de Umidade do ar.

```
      import random
      import pymssql
      from datetime import datetime, timedelta
      import math

      # Define as configurações de conexão com o banco de dados
      server = 'greenhouseserver.database.windows.net'
      user = 'bielht121'
      password = 'qwer1234@'
      database = 'GreenhouseDB_Core'

      # Conecta ao banco de dados
      conn = pymssql.connect(server, user, password, database)

      max_soil_moisture = 95
      min_soil_moisture = 30
      days_to_generate = 7
      decay_rate = 0.9999  # Ajustado para obter um decaimento
mais suave
      variance_factor = 0.008

      def soil_moisture_curve(hour):
          # Ajusta os valores para que a umidade máxima ocorra
às 8h da manhã e decresça ao longo do dia
          if hour == 8:
              return max_soil_moisture
          else:
              return min_soil_moisture

      def adjusted_variance(soil_moisture):
          if soil_moisture < 40:
              return variance_factor * 0.25
          elif soil_moisture < 50:
              return variance_factor * 0.4
          elif soil_moisture < 60:
              return variance_factor * 0.55
          else:
              return variance_factor

      def adjusted_variance_factor(hour, soil_moisture):
          if hour >= 8 and hour <= 17:
              return adjusted_variance(soil_moisture)
          else:
              return adjusted_variance(soil_moisture) * 0.1

      # Implementa a lógica aqui
      now = datetime.now()
      start_date = datetime(now.year, now.month, now.day)  #
Inicia a data com o horário 00:00:00
      end_date = start_date + timedelta(days=days_to_generate)
```

```
      current_date = start_date
      previous_soil_moisture = max_soil_moisture
      while current_date <= end_date:
          for hour in range(0, 24):
                for _ in range(720):  # 720 vezes a cada 5
segundos corresponde a 1 hora
                                          base_soil_moisture    =
soil_moisture_curve(hour)

                  if hour == 8 and current_date.second == 0:
                      soil_moisture = max_soil_moisture
                  else:
                                           adjusted_var_factor   =
adjusted_variance_factor(hour, previous_soil_moisture)
                        random_variance = 1 + random.uniform(-
adjusted_var_factor, adjusted_var_factor)
                        soil_moisture = max(min_soil_moisture,
previous_soil_moisture * decay_rate * random_variance)

                     # Atualiza a umidade do solo anterior e
arredonda para desprezar os milésimos
                  previous_soil_moisture = round(soil_moisture,
2)

                  # Insere os dados no banco de dados
                  cursor = conn.cursor()
                    cursor.execute('INSERT INTO umidade (hora,
umidade) VALUES (%s, %s)', (current_date, soil_moisture))
                  conn.commit()

                  # Incrementa 5 segundos
                  current_date += timedelta(seconds=5)

      # Fecha a conexão com o banco de dados
      conn.close()
```

APÊNDICE D - Código Fonte para geração de dados de umidade do solo.

```
import random
import pymssql
from datetime import datetime, timedelta
import math

# Define as configurações de conexão com o banco de dados
server = 'greenhouseserver.database.windows.net'
user = 'bielht121'
password = 'qwer1234@'
database = 'GreenhouseDB_Core'

# Conecta ao banco de dados
conn = pymssql.connect(server, user, password, database)

max_soil_moisture = 95
min_soil_moisture = 20
days_to_generate = 7
decay_rate = 0.95
variance_factor = 0.15

def soil_moisture_curve(hour):
    # Ajusta os valores para que a umidade máxima ocorra
às 8h da manhã e decresça ao longo do dia
    if hour == 8:
        return max_soil_moisture
    else:
        return min_soil_moisture

# Implementa a lógica aqui
start_date = datetime.now()
end_date = start_date + timedelta(days=days_to_generate)

current_date = start_date
previous_soil_moisture = max_soil_moisture
while current_date <= end_date:
    for hour in range(0, 24):
        base_soil_moisture = soil_moisture_curve(hour)

        if hour == 8:
            soil_moisture = max_soil_moisture
        else:
                random_variance = 1 + random.uniform(-
variance_factor, variance_factor)
                 soil_moisture = max(min_soil_moisture,
previous_soil_moisture * decay_rate * random_variance)

        # Atualiza a umidade do solo anterior
        previous_soil_moisture = soil_moisture
```

```
                # Insere os dados no banco de dados
                cursor = conn.cursor()
                    cursor.execute('INSERT INTO umidade (hora,
umidade) VALUES (%s, %s)', (current_date, soil_moisture))
                conn.commit()

                # Incrementa uma hora
                current_date += timedelta(hours=1)

        # Fecha a conexão com o banco de dados
        conn.close()
```

APÊNDICE E - Código fonte de interpolação para obter dados em minutos.

```
alter FUNCTION dbo.ExpandHourlyDataToMinuteData()
RETURNS @MinuteData TABLE
(
    hora DATETIME,
    temperatura FLOAT
)
AS
BEGIN
     -- Cria uma tabela temporária para armazenar os
resultados intermediários
    DECLARE @TempData TABLE
    (
        RowNum INT,
        hora DATETIME,
        temperatura FLOAT,
        NextHora DATETIME,
        NextTemperatura FLOAT
    );

     -- Insere os dados na tabela temporária com as
informações das linhas subsequentes
     INSERT INTO @TempData (RowNum, hora, temperatura,
NextHora, NextTemperatura)
    SELECT
        ROW_NUMBER() OVER (ORDER BY hora) AS RowNum,
        hora,
        temperatura,
          LEAD(hora, 1, NULL) OVER (ORDER BY hora) AS
NextHora,
         LEAD(temperatura, 1, NULL) OVER (ORDER BY hora)
AS NextTemperatura
    FROM Temperaturas;

    -- Insere os dados interpolados na tabela de saída
    DECLARE @CurrentRow INT = 1, @MaxRow INT;
    SELECT @MaxRow = MAX(RowNum) FROM @TempData;

    WHILE @CurrentRow <= @MaxRow
    BEGIN
           DECLARE @hora DATETIME, @temperatura FLOAT,
@NextHora DATETIME, @NextTemperatura FLOAT;

        SELECT
            @hora = hora,
            @temperatura = temperatura,
            @NextHora = NextHora,
            @NextTemperatura = NextTemperatura
        FROM @TempData
        WHERE RowNum = @CurrentRow;

        IF @NextHora IS NULL
```

```
                    SET @NextHora = DATEADD(hour, 1, @hora);

                WHILE @hora < @NextHora
                BEGIN
                    INSERT INTO @MinuteData (hora, temperatura)
                    VALUES (@hora, @temperatura);

                    SET @hora = DATEADD(minute, 1, @hora);
                END;

                SET @CurrentRow = @CurrentRow + 1;
            END;

            RETURN;
        END;

        SELECT *
        INTO temperaturaSegundos
        FROM
[GreenhouseDB_Core].dbo.ExpandHourlyDataToMinuteData();

        select * from temperaturaSegundos
```

APÊNDICE F - Código fonte para obter dados de sensores em segundos.

```
          --insert into measure_temperature_raspberry01_final
          SELECT
               CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
              'Raspberry01' Client,
              'Node00' Device,
                           temperatura    +    (CAST((2.8    *
RAND(CHECKSUM(NEWID())) - 1) * 100 AS DECIMAL(10, 2)) * 0.01)
AS Temperatura  --+- 2 graus
          into temperaturaRaspberry01Node00
          FROM temperaturaSegundosFinal
          --insert into measure_temperature_raspberry01_final
          SELECT
               CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
              'Raspberry01' Client,
              'Node01' Device,
                           temperatura    +    (CAST((2.1    *
RAND(CHECKSUM(NEWID())) - 1) * 100 AS DECIMAL(10, 2)) * 0.01)
AS Temperatura  --+- 2 graus
          into temperaturaRaspberry01Node01
          FROM temperaturaSegundosFinal
          --insert into measure_temperature_raspberry01_final
          SELECT
               CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
              'Raspberry01' Client,
              'Node02' Device,
                           temperatura    +    (CAST((2.5    *
RAND(CHECKSUM(NEWID())) - 1) * 100 AS DECIMAL(10, 2)) * 0.01)
AS Temperatura  --+- 2 graus
          into temperaturaRaspberry01Node02
          FROM temperaturaSegundosFinal
          -----

          --insert Into measure_soil_moisture_raspberry01_final
          SELECT
               CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
              'Raspberry01' Client,
              'Node00' Device,
               umidade + (CAST((12 * RAND(CHECKSUM(NEWID())) -
1) * 100 AS DECIMAL(10, 2)) * 0.01) AS umidadedosolo  --+- 5%
          INTO umidadeSoloRaspberry01Node00
          FROM UmidadesegundosFinal
          --insert Into measure_soil_moisture_raspberry01_final
          SELECT
               CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
              'Raspberry01' Client,
              'Node01' Device,
```

```
              umidade + (CAST((25 * RAND(CHECKSUM(NEWID())) -
1) * 100 AS DECIMAL(10, 2)) * 0.01) AS umidadedosolo  --+- 5%
          INTO umidadeSoloRaspberry01Node01
          FROM UmidadesegundosFinal
          --insert Into measure_soil_moisture_raspberry01_final
          SELECT
               CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
              'Raspberry01' Client,
              'Node02' Device,
               umidade + (CAST((15 * RAND(CHECKSUM(NEWID())) -
1) * 100 AS DECIMAL(10, 2)) * 0.01) AS umidadedosolo  --+- 5%
          INTO umidadeSoloRaspberry01Node02
          FROM UmidadesegundosFinal
          ------


          --insert into measure_air_humidity_raspberry01_final
          SELECT
               CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
              'Raspberry01' Client,
              'Node00' Device,
               umidade + (CAST((15 * RAND(CHECKSUM(NEWID())) -
1) * 100 AS DECIMAL(10, 2)) * 0.01) AS umidadedoar  --+- 5%
           INTO umidadeArRaspberry01Node00
          FROM UmidadedoarSegundosFinal
          --insert into measure_air_humidity_raspberry01_final
          SELECT
               CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
              'Raspberry01' Client,
              'Node01' Device,
              umidade + (CAST((8 * RAND(CHECKSUM(NEWID())) - 1)
* 100 AS DECIMAL(10, 2)) * 0.01) AS umidadedoar  --+- 5%
           INTO umidadeArRaspberry01Node01
          FROM UmidadedoarSegundosFinal
          --insert into measure_air_humidity_raspberry01_final
          SELECT
               CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
              'Raspberry01' Client,
              'Node02' Device,
               umidade + (CAST((10 * RAND(CHECKSUM(NEWID())) -
1) * 100 AS DECIMAL(10, 2)) * 0.01) AS umidadedoar  --+- 5%
           INTO umidadeArRaspberry01Node02
          FROM UmidadedoarSegundosFinal
          --------

          --insert into measure_luminosity_raspberry01_final
          SELECT
```

```
              CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
             'Raspberry01' Client,
             'Node02' Device,
                          luminosidade    +    (CAST((300    *
RAND(CHECKSUM(NEWID())) - 1) * 100 AS DECIMAL(10, 2)) * 0.01)
AS luminosidade  --+- 5%
          INTO luminosidadeRaspberry01Node02
          FROM luminosidadeSegundosFinal
          --insert into measure_luminosity_raspberry01_final
          SELECT
               CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
             'Raspberry01' Client,
             'Node01' Device,
                          luminosidade    +    (CAST((210    *
RAND(CHECKSUM(NEWID())) - 1) * 100 AS DECIMAL(10, 2)) * 0.01)
AS luminosidade  --+- 5%
          INTO luminosidadeRaspberry01Node01
          FROM luminosidadeSegundosFinal
          --insert into measure_luminosity_raspberry01_final
          SELECT
               CAST(CAST(horasegundo AS datetime) AS datetime)
AS hora,
             'Raspberry01' Client,
             'Node00' Device,
                          luminosidade    +    (CAST((150    *
RAND(CHECKSUM(NEWID())) - 1) * 100 AS DECIMAL(10, 2)) * 0.01)
AS luminosidade  --+- 5%
          INTO luminosidadeRaspberry01Node00
          FROM luminosidadeSegundosFinal


      select
      *
      into SensorData_Samples_Final
      from (
          select
          a.hora datameasure,
          'Raspberry01' client,
          1 deviceid,
          8584948494 internal_id,
          'sensor_stream' topic,
          'MESH_NETWORK_RASPBERRY01' network_name,
          'sensor' role,
          'internal' location,
          a.temperatura,
          b.luminosidade,
          c.umidadedoar,
          d.umidadedosolo,
          getdate() dateinsert
```

```
          from temperaturaRaspberry01Node00 a
           join luminosidadeRaspberry01Node00 b on a.Device =
b.Device and a.hora = b.hora
            join umidadeArRaspberry01Node00 c on c.Device =
b.Device and c.hora = b.hora
           join umidadeSoloRaspberry01Node00 d on d.Device =
c.Device and d.hora = c.hora
          union all
              select
          a.hora datameasure,
          'Raspberry01' client,
          1 deviceid,
          9813508447 internal_id,
          'sensor_stream' topic,
          'MESH_NETWORK_RASPBERRY01' network_name,
          'sensor' role,
          'internal' location,
          a.temperatura,
          b.luminosidade,
          c.umidadedoar,
          d.umidadedosolo,
          getdate() dateinsert
          from temperaturaRaspberry01Node01 a
           join luminosidadeRaspberry01Node01 b on a.Device =
b.Device and a.hora = b.hora
            join umidadeArRaspberry01Node01 c on c.Device =
b.Device and c.hora = b.hora
           join umidadeSoloRaspberry01Node01 d on d.Device =
c.Device and d.hora = c.hora
          union all
              select
          a.hora datameasure,
          'Raspberry01' client,
          1 deviceid,
          5285657415 internal_id,
          'sensor_stream' topic,
          'MESH_NETWORK_RASPBERRY01' network_name,
          'sensor' role,
          'internal' location,
          a.temperatura,
          b.luminosidade,
          c.umidadedoar,
          d.umidadedosolo,
          getdate() dateinsert
          from temperaturaRaspberry01Node02 a
           join luminosidadeRaspberry01Node02 b on a.Device =
b.Device and a.hora = b.hora
            join umidadeArRaspberry01Node02 c on c.Device =
b.Device and c.hora = b.hora
           join umidadeSoloRaspberry01Node02 d on d.Device =
c.Device and d.hora = c.hora
```

```
        ) auxiliar


      alter      table      SensorData_Samples_Final      add
status_integracao int
      alter      table      SensorData_Samples_Final      add
datetime_integration datetime2
      alter table SensorData_Samples_Final add id int

      update SensorData_Samples_Final set status_integration =
0
```