FIAP / CPQD

# Atividade 3

## Streaming MQTT do Device para ThingSpeak + Testes

**Disciplina: Sistemas embarcados e Internet das Coisas (IoT)**

Professor: Andre Braga
Aluno: Leonid H. Mamani

Campinas – 2026

# Sumário

# Capítulo 1

# Introdução

## 1.1 Contextualização

A Internet das Coisas (IoT) tem se consolidado como um dos principais paradigmas tecnológicos da Indústria 4.0, permitindo a interconexão de dispositivos físicos, sensores e sistemas computacionais por meio de redes de comunicação. Em ambientes modernos, dispositivos embarcados coletam dados em tempo real e os enviam para plataformas em nuvem, onde são armazenados, processados e analisados.

Entretanto, em arquiteturas IoT profissionais, não é recomendado que dispositivos dependam diretamente de uma única plataforma de nuvem. Sistemas distribuídos modernos utilizam brokers intermediários para garantir desacoplamento, escalabilidade, interoperabilidade e observabilidade do fluxo de dados.

Neste contexto, o protocolo MQTT (Message Queuing Telemetry Transport) destaca-se como uma solução leve e eficiente para comunicação em sistemas distribuídos baseados no modelo publish/subscribe. O uso de um broker MQTT intermediário permite que múltiplos consumidores recebam os mesmos dados simultaneamente, além de facilitar testes, depuração e controle de fluxo.

O presente trabalho insere-se nesse cenário ao propor a implementação de uma arquitetura distribuída de streaming MQTT, conectando um dispositivo IoT simulado à plataforma ThingSpeak por meio de um broker intermediário (HiveMQ) e de um gateway programável em Python.

## 1.2 Objetivo do Trabalho

O objetivo deste trabalho é projetar, implementar e validar uma arquitetura de transmissão de dados IoT baseada no protocolo MQTT, utilizando um broker intermediário (HiveMQ) entre o dispositivo e a plataforma em nuvem ThingSpeak, a fim de demonstrar uma abordagem desacoplada, interoperável e alinhada com arquiteturas industriais modernas.

O projeto implementa um pipeline completo de telemetria IoT, no qual um dispositivo simulado no Wokwi publica dados de sensores em um broker MQTT (HiveMQ). A integração com a plataforma ThingSpeak é realizada por meio de um script em Python utilizando a biblioteca `paho-mqtt`, responsável por consumir as mensagens do broker e encaminhá-las ao canal configurado na nuvem.

Além da implementação funcional, o foco principal do trabalho é validar as vantagens

arquiteturais do uso de um broker intermediário em sistemas IoT distribuídos.

## 1.3 Justificativa Arquitetural

A escolha por uma arquitetura distribuída com broker intermediário fundamenta-se em quatro princípios técnicos essenciais:

### 1.3.1 Desacoplamento e Interoperabilidade

Em sistemas IoT reais, dispositivos não devem depender diretamente de uma única plataforma de nuvem. Ao utilizar um broker externo como o HiveMQ:

- O dispositivo publica os dados apenas uma vez;
- Múltiplos consumidores podem se inscrever simultaneamente no mesmo tópico;
- A plataforma final pode ser substituída sem necessidade de alterar o firmware do dispositivo.

Essa abordagem garante escalabilidade, redução de custos operacionais e flexibilidade tecnológica.

### 1.3.2 Visibilidade e Debug do Fluxo de Dados

O broker intermediário atua como ponto central de inspeção do tráfego MQTT. Ferramentas como MQTT Explorer, clientes MQTT de linha de comando e logs do próprio HiveMQ permitem verificar:

- Se o dispositivo está publicando corretamente;
- Se as mensagens estão chegando ao broker;
- Se o gateway está recebendo os dados;
- Se o envio para o ThingSpeak está ocorrendo adequadamente.

Sem o broker intermediário, a identificação de falhas torna-se mais complexa, dificultando a separação entre problemas de código, rede ou autenticação.

### 1.3.3 Simulação de Arquitetura Industrial

Em ambientes industriais, sensores de campo geralmente publicam dados para um broker local ou gateway edge (por exemplo, Mosquitto em um Raspberry Pi). Esse gateway realiza:

- Filtragem e validação de dados;
- Agregação de informações;
- Controle de segurança;
- Encaminhamento estruturado para a nuvem.

A arquitetura implementada neste trabalho simula esse modelo profissional, aproximando o experimento acadêmico de cenários reais de IoT industrial.

### 1.3.4 Controle de Fluxo e Tratamento de Dados

Plataformas como o ThingSpeak possuem limitações de frequência de envio de dados. O uso de um broker intermediário e de um gateway programável permite:

- Receber dados em alta frequência;
- Processar ou agregar informações antes do envio;
- Enviar apenas dados consolidados;
- Implementar filtros e validações.

Dessa forma, o broker e o gateway atuam como elementos estratégicos de controle do ecossistema IoT.

## 1.4 Síntese do Propósito da Atividade

Este trabalho não se limita à integração entre ferramentas, mas demonstra a importância de uma arquitetura IoT desacoplada, escalável e observável, baseada em boas práticas de engenharia de sistemas distribuídos.

Ao final do projeto, pretende-se comprovar:

- O funcionamento completo do fluxo Device → Broker → Gateway → Nuvem;
- A integridade e estabilidade da comunicação MQTT;
- Os benefícios arquiteturais do uso de broker intermediário;
- A aplicabilidade do modelo em cenários industriais reais.

Em síntese, o uso de um broker externo permite gerenciar o ecossistema IoT como um sistema distribuído completo, e não apenas como uma integração pontual entre dispositivo e plataforma.

# Capítulo 2

# Arquitetura do Sistema

## 2.1 Arquitetura do Sistema Embarcado (ESP32)

Esta seção descreve a arquitetura do dispositivo embarcado responsável pela aquisição de dados ambientais e de segurança física, visualização local e comunicação segura via MQTT. O sistema foi desenvolvido no contexto de simulação com o ESP32 DevKit no Wokwi, integrando sensores, atuadores e interface homem-máquina (IHM) com comunicação criptografada (SSL/TLS) para um broker MQTT em nuvem.

### 2.1.1 Visão Geral Funcional

O firmware do ESP32 implementa as seguintes funcionalidades principais:

- Monitoramento térmico utilizando sensor DHT22 (temperatura e umidade);
- Detecção de presença por sensor PIR (AS312), gerando evento de alerta;
- Visualização local em display LCD 16x2 via barramento $I^2C$;
- Comunicação segura via MQTT sobre SSL/TLS;
- Controle remoto de atuador (Cooler/Alarme) simulado por LED;
- Publicação de telemetria para arquitetura distribuída (Broker + Gateway);
- Integração indireta com ThingSpeak por meio de um gateway externo.

### 2.1.2 Arquitetura em Camadas do Firmware

Para facilitar manutenção, testes e evolução do sistema, o firmware pode ser descrito em camadas funcionais:

1. **Camada de Hardware e Periféricos:** configuração de GPIO, barramento $I^2C$ e temporizações; acesso aos sinais do DHT22, PIR e LED.
2. **Camada de Drivers:** bibliotecas responsáveis pela interface com sensores e display (DHT e LCD $I^2C$).
3. **Camada de Lógica e Aplicação:** regras para leitura periódica, detecção de eventos, formatação de mensagens, atualização do display e controle do atuador.

4. **Camada de Comunicação:** conectividade Wi-Fi, sessão MQTT e criptografia SSL/TLS para publicação/assinatura de tópicos.

### 2.1.3 Componentes do Sistema Embarcado

A Tabela 2.1 apresenta os componentes do sistema embarcado (simulado) e suas funções.

Tabela 2.1: Componentes do sistema embarcado (Wokwi)

| **Componente** | **Função** | **Observação** |
|---|---|---|
| ESP32 DevKit | Microcontrolador principal | Executa firmware e comunicação MQTT |
| DHT22 | Sensor de temperatura e umidade | Aquisição periódica de telemetria |
| PIR (AS312) | Sensor de movimento/presença | Evento de alerta por detecção |
| LED Vermelho | Atuador (Cooler/Alarme) simulado | Controlado remotamente por MQTT |
| LCD 16x2 ($I^2C$) | Interface visual local | Exibe dados e status do sistema |

### 2.1.4 Mapeamento de GPIO e Interconexões

A Tabela 2.2 apresenta o mapeamento de pinos (GPIO) do ESP32 utilizado na simulação.

Tabela 2.2: Mapeamento de GPIO do ESP32

| **Módulo** | **Sinal** | **GPIO** | **Direção** |
|---|---|---|---|
| DHT22 | DATA | GPIO 15 | Entrada |
| PIR (AS312) | OUT | GPIO 13 | Entrada |
| LED (Atuador) | CTRL | GPIO 2 | Saída |
| LCD 16x2 ($I^2C$) | SDA | GPIO 21 | Bidirecional |
| LCD 16x2 ($I^2C$) | SCL | GPIO 22 | Saída (clock) |

### 2.1.5 Fluxo de Operação do Firmware

O funcionamento do firmware ocorre em um ciclo contínuo (loop), seguindo os passos:

1. **Inicialização:** configuração de GPIO, $I^2C$, LCD e sensores; conexão Wi-Fi; estabelecimento de sessão MQTT segura (SSL/TLS).
2. **Aquisição de dados:** leitura periódica do DHT22 (temperatura/umidade).
3. **Detecção de evento:** leitura do PIR para identificar presença/movimento e gerar alerta quando necessário.
4. **IHM local:** atualização do LCD com valores atuais e estado do sistema.
5. **Publicação MQTT:** envio das variáveis e eventos para tópicos MQTT correspondentes.
6. **Subscrição MQTT:** recepção de comandos remotos para acionar/desligar o atuador (LED).

### 2.1.6 Bibliotecas e Dependências do Firmware

O firmware foi desenvolvido em C++ utilizando Arduino Framework, com as bibliotecas listadas na Tabela 2.3.

Tabela 2.3: Bibliotecas utilizadas no firmware ESP32

| **Biblioteca** | **Finalidade** | **Observação** |
|---|---|---|
| `PubSubClient` | Cliente MQTT | Publish/Subscribe |
| `DHT Sensor Library` | Driver DHT22 | Leitura de temperatura/umidade |
| `LiquidCrystal I2C` | Driver LCD $I^2C$ | Interface visual local |
| `WiFiClientSecure` | SSL/TLS | MQTT seguro (porta 8883) |

### 2.1.7 Integração com Gateway e ThingSpeak

Embora o ESP32 publique os dados no broker MQTT, o envio ao ThingSpeak é realizado por um **gateway externo em Python**, responsável por subscrever os tópicos e encaminhar as medições via API HTTP do ThingSpeak. A Tabela 2.4 descreve as bibliotecas empregadas no gateway.

Tabela 2.4: Bibliotecas utilizadas no gateway Python

| **Biblioteca** | **Finalidade** | **Observação** |
|---|---|---|
| `paho-mqtt` | Cliente MQTT em Python | Subscrição e processamento de mensagens |
| `requests` | Cliente HTTP | Envio de dados ao ThingSpeak (Write API Key) |

## 2.2 Arquitetura de Comunicação Baseada em MQTT

O sistema implementa uma arquitetura de comunicação distribuída baseada no protocolo MQTT (Message Queuing Telemetry Transport), utilizando o modelo publish/subscribe para transmissão de dados entre dispositivos IoT e serviços em nuvem.

Diferentemente de arquiteturas cliente-servidor tradicionais, nas quais dispositivos comunicam-se diretamente com aplicações finais, o MQTT introduz um elemento intermediário denominado *broker*, responsável pelo gerenciamento das mensagens e distribuição dos dados entre produtores e consumidores.

A Figura 2.1 apresenta a arquitetura geral implementada neste trabalho.

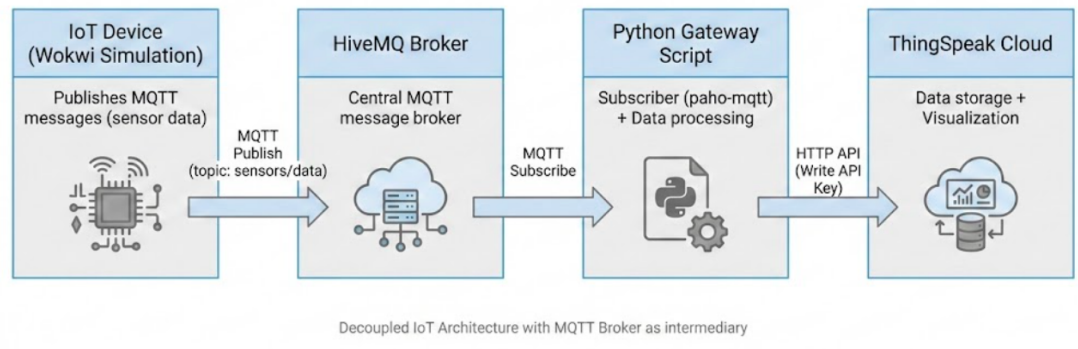


Figura 2.1: Arquitetura distribuída baseada em MQTT utilizada no sistema

### 2.2.1 Modelo Publish/Subscribe

O protocolo MQTT opera segundo o paradigma publish/subscribe, no qual:

- **Publishers** enviam mensagens para tópicos;
- **Subscribers** recebem mensagens dos tópicos de interesse;
- O **broker MQTT** intermedia toda a comunicação.

Esse modelo promove desacoplamento entre os componentes do sistema, permitindo escalabilidade e flexibilidade arquitetural.

### 2.2.2 Componentes da Arquitetura

A arquitetura implementada é composta por quatro elementos principais:

#### IoT Device (Wokwi Simulation)

O dispositivo embarcado baseado em ESP32 atua como *publisher*, sendo responsável por:

- Coletar dados dos sensores;
- Publicar mensagens MQTT contendo telemetria;
- Enviar eventos de monitoramento ao broker.

As mensagens são publicadas no tópico:

- `fiap/datacenter/rack01/temperatura`
- `fiap/datacenter/rack01/umidade`
- `fiap/datacenter/rack01/alertapresenca`

#### HiveMQ Broker

O HiveMQ atua como broker MQTT central da arquitetura, executando:

- Recepção das mensagens publicadas;
- Gerenciamento de tópicos;
- Distribuição das mensagens aos assinantes;
- Desacoplamento entre produtores e consumidores.

O broker constitui o núcleo lógico da comunicação.

#### Python Gateway Script

O gateway implementado em Python utiliza a biblioteca `paho-mqtt` para subscrever os tópicos MQTT.

Suas funções incluem:

- Receber mensagens do broker;
- Processar e validar dados;
- Converter mensagens para requisições HTTP;
- Encaminhar dados ao ThingSpeak.

Esse componente simula um gateway IoT industrial.

#### ThingSpeak Cloud

A plataforma ThingSpeak atua como camada de nuvem responsável por:

- Armazenamento histórico dos dados;
- Visualização gráfica;
- Monitoramento remoto.

A comunicação ocorre via API HTTP utilizando a Write API Key.

**Comparativo Técnico: HiveMQ vs. ThingSpeak (MQTT)**

Embora ambos utilizem o protocolo MQTT, a gestão de sessões e identidades difere fundamentalmente entre brokers genéricos e plataformas de IoT:

- **HiveMQ (Public/Open):** Opera como um broker convencional. A unicidade da conexão é baseada apenas no *Client ID*. Desde que dois dispositivos utilizem IDs distintos, eles podem compartilhar as mesmas credenciais de acesso (Username/Password) ou até operar sem autenticação em instâncias públicas.
- **ThingSpeak (Managed Platform):** Atua como um broker proprietário onde a segurança é baseada na **identidade do dispositivo**. O sistema gera um trio único e indissociável (*Client ID*, *Username* e *MQTT API Key*).
- É necessária a criação de um *MQTT Device* adicional no painel do ThingSpeak para cada cliente simultâneo, garantindo que cada um possua seu próprio *Client ID* exclusivo.

### 2.2.3 Fluxo de Comunicação

O fluxo operacional segue as etapas:

1. O dispositivo publica dados via MQTT;

2. O broker HiveMQ recebe e distribui as mensagens;
3. O gateway Python subscreve e processa os dados;
4. O gateway envia os dados ao ThingSpeak via HTTP API.

# Capítulo 3

# Implementação

## 3.1 Ferramentas Utilizadas

### 3.1.1 Wokwi

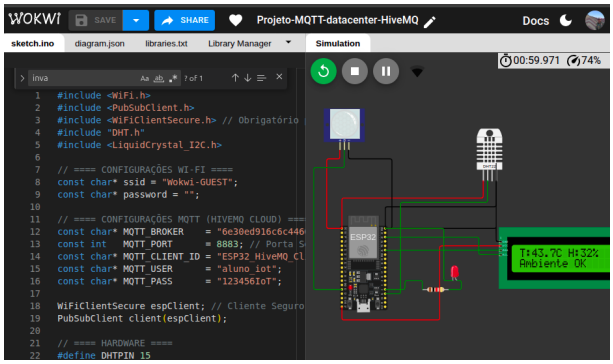


Figura 3.1: Configuração do Wokwi

### 3.1.2 HiveMQ Broker

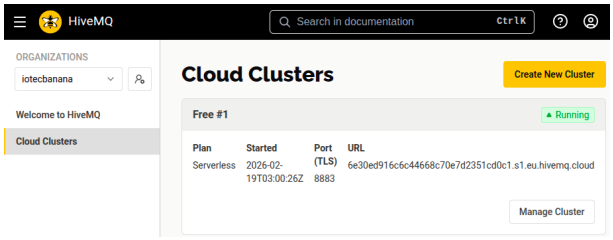


Figura 3.2: HiveMQ

### 3.1.3 Python + paho-mqtt

```
 GNU nano 7.2                        ponte_v1.py
import paho.mqtt.client as mqtt
import time

# --- CONFIGURAÇÕES HIVEMQ (Origem - Wokwi) ---
HIVEMQ_BROKER = "6e30ed916c6c44668c70e7d2351cd0c1.s1.eu.hivemq.cloud"
HIVEMQ_USER   = "aluno_iot"
HIVEMQ_PASS   = "123456IoT"

TOPICOS_HIVE = {
    "fiap/datacenter/rack01/temperatura": "field1",
    "fiap/datacenter/rack01/umidade": "field2",
    "fiap/datacenter/rack01/alerta_presenca": "field3"
}

# --- CONFIGURAÇÕES THINGSPEAK (Destino) ---
TS_BROKER     = "mqtt3.thingspeak.com"
TS_CHANNEL_ID = "3269220"
TS_CLIENT_ID  = "KDEtIxUQERwwHi4CNgYANhs"
TS_USER       = "KDEtIxUQERwwHi4CNgYANhs"
TS_PASS       = "DGF2VlaPESbD2bpH4wt9V//p"
TS_TOPIC      = f"channels/{TS_CHANNEL_ID}/publish"
```

Figura 3.3: paho-mqtt

### 3.1.4 ThingSpeak

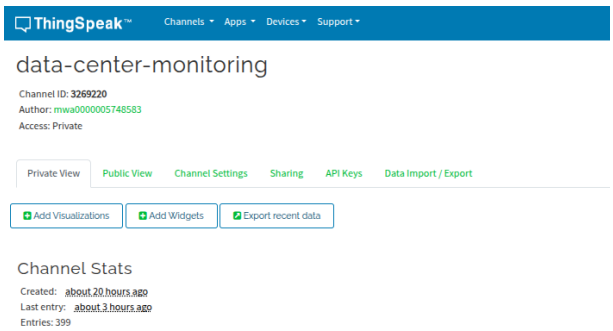


Figura 3.4: ThingSpeak

## 3.2 Configuração do HiveMQ e do Canal ThingSpeak

Tabela 3.1: Parâmetros de Configuração: Broker ThingSpeak

| [HTML]EFEFEF **Parâmetro** | **Valor Configurado** |
|---|---|
| **Broker Address** | `mqtt3.thingspeak.com` |
| **Channel ID** | `3269220` |
| **Client ID** | `KDEtIxUQERwwHi4CNgYANhs` |
| **Username** | `KDEtIxUQERwwHi4CNgYANhs` |
| **MQTT API Key** | `DGF2VlaPESbD2bpH4wt9V//p` |
| **Publish Topic** | `channels/3269220/publish` |

Descrição dos campos criados e API Keys.

Tabela 3.2: Parâmetros de Conectividade MQTT no HiveMQ

| **Parâmetro** | **Valor Configurado** |
|---|---|
| Broker Endereço | `6e30ed916c6c44668c70e7d2351cd0c1.s1.eu.hivemq.cloud` |
| Porta TCP (SSL) | 8883 |
| MQTT Client ID | `ESP32_HiveMQ_Client_01` |
| Usuário | `aluno_iot` |
| Senha | `**********` |

## 3.3 Configuração do Dispositivo no Wokwi

Descrição do streaming MQTT e sensores simulados.

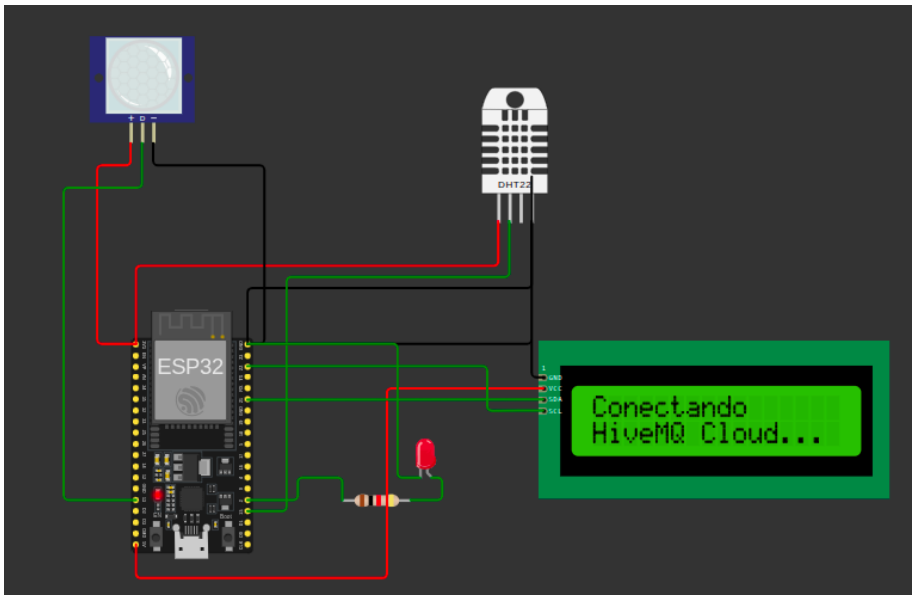


Figura 3.5: Diagrama de conexões

## 3.4 Script Gateway em Python

Explicação do funcionamento:

- Subscrição MQTT
- Processamento das mensagens
- Envio HTTP para ThingSpeak

**Resumo Técnico: Ponte MQTT (HiveMQ → ThingSpeak)**

O script opera como um *gateway* de integração, processando o fluxo de dados em três etapas fundamentais:

1. **Intermediação:** Atua como cliente simultâneo em dois *brokers*. Consome dados do **HiveMQ Cloud** (Porta 8883/TLS) e os encaminha ao **ThingSpeak** (Porta 1883).
2. **Tradução de Protocolo:** Converte tópicos hierárquicos do Datacenter em *payloads* de campos específicos (*fields*), padrão exigido pela API do ThingSpeak:
   - `.../temperatura` → `field1=valor`
   - `.../umidade` → `field2=valor`
3. **Segurança e Identidade:** Garante a persistência da conexão ao utilizar *Client IDs* exclusivos, evitando conflitos de sessão e quedas de comunicação entre o hardware (Wokwi) e o monitoramento (Dashboard).

## 3.5 Comunicação MQTT

**Fluxo de Dados: Do Hardware à Nuvem**

O percurso da informação ocorre em três etapas sequenciais integradas pelo script Python:

1. **Origem (ESP32 → HiveMQ):** O sensor no ESP32 publica dados em tópicos hierárquicos estruturados (ex: `fiap/datacenter/rack01/temperatura`). O HiveMQ Cloud recebe e retém essas mensagens.
2. **Processamento (Script Python):** O script atua como um assinante (*subscriber*) do HiveMQ. Ao receber uma mensagem, ele utiliza um dicionário de mapeamento (`TOPICOS_HIVE`) para identificar a qual campo (*field*) do ThingSpeak aquele dado pertence.
3. **Destino (Script → ThingSpeak):** O script reformata o dado bruto em uma string de atribuição (ex: `field1=25.5`) e a publica no tópico mestre do ThingSpeak (`channels/3269220/publish`), permitindo a atualização dos gráficos.

## 3.6 Tópicos de Publicação no HiveMQ e Thingspeak

| Tópico | Descrição |
|---|---|
| fiap/datacenter/rack01/temperatura | Temperatura |
| fiap/datacenter/rack01/umidade | Umidade |
| fiap/datacenter/rack01/alerta | Evento PIR |

Tabela 3.3: Tópicos MQTT publicados no HiveMQ

Estrutura de Endereçamento: MQTT ThingSpeak

Diferente de brokers convencionais, o ThingSpeak centraliza a recepção de dados em um tópico fixo. A estrutura segue o padrão abaixo:

**1. Tópico de Publicação:**

`channels/<channel_id>/publish`

No projeto utilizado: `channels/3269220/publish`

**2. Formato do Payload (Mensagem):** A mensagem enviada não contém apenas o valor, mas a atribuição ao campo (*field*) correspondente no dashboard:

- **Sintaxe:** `fieldX=valor`
- **Exemplo Real:** `field1=25.5` (para temperatura)

**3. Publicação Múltipla (Otimizada):** Caso deseje enviar vários sensores em um único pacote MQTT para economizar banda:

`field1=25.5&field2=60&status=MENSAGEM`

# Capítulo 4

# Testes e Resultados

## 4.1 Verificação de Recepção no Broker

A Figura 4.1 ilustra os dados de telemetria recebidos no *software* MQTT Explorer a partir do *broker* HiveMQ. A estrutura de tópicos está organizada sob a hierarquia `fiap/datacenter/rack01`, monitorando as variáveis de temperatura (31,00 °C), umidade (25,50%) e presença via sensor PIR (valor lógico 1). Através do histórico visual, é possível observar a variação temporal de cada grandeza, validando a integridade da comunicação entre os dispositivos e o *broker*.

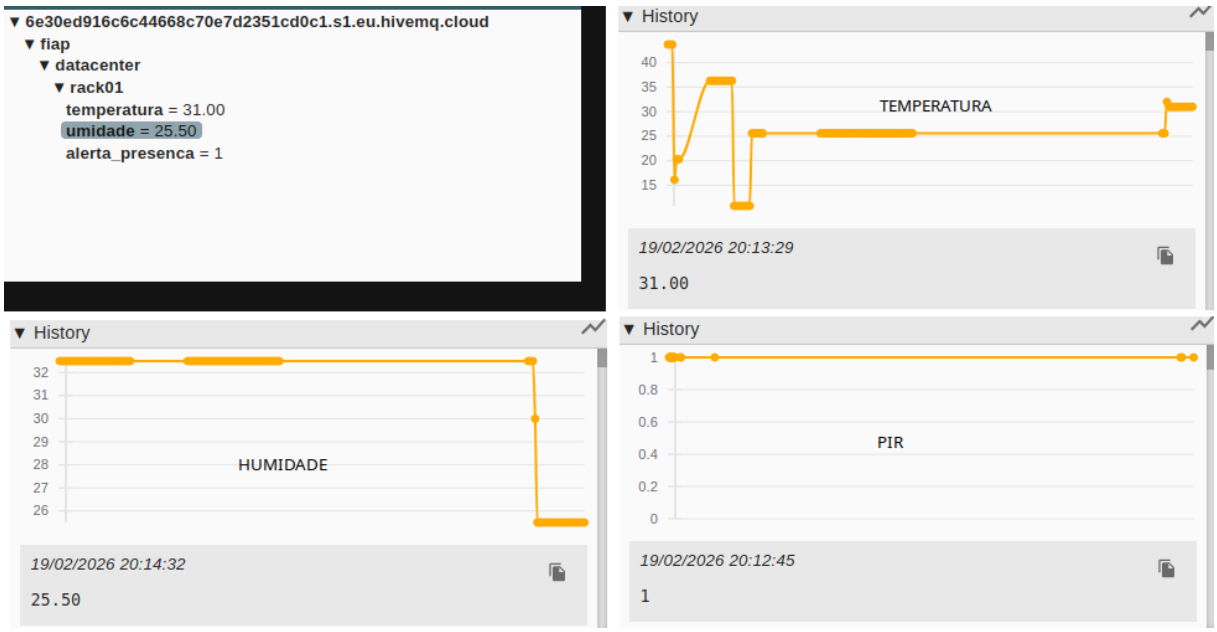


Figura 4.1: Registro mostrado pelo Mqtt Explorer

## 4.2 Validação no ThingSpeak

A Figura 4.2 apresenta os gráficos de temperatura, umidade e presença (PIR) visualizados no *dashboard* do ThingSpeak. Nota-se uma densidade de pontos significativamente menor em comparação ao MQTT Explorer, devido à restrição da plataforma que permite o

recebimento de dados apenas em intervalos de 15 segundos. Essa limitação resulta em uma amostragem mais esparsa, impactando a granularidade do monitoramento em tempo real quando comparada à conexão direta via *broker*. Use o código com cuidado.

O que foi ajustado: Termos Técnicos: Substituímos "quantidade de dados"por "densidade de pontos"ou "amostragem", que são termos mais acadêmicos. Causa e Efeito: O texto agora explica claramente que a "granularidade"(o detalhamento) é menor devido à latência de 15 segundos imposta pela plataforma. Ordem: O texto respeita a ordem dos gráficos apresentados (temperatura, umidade e presença).

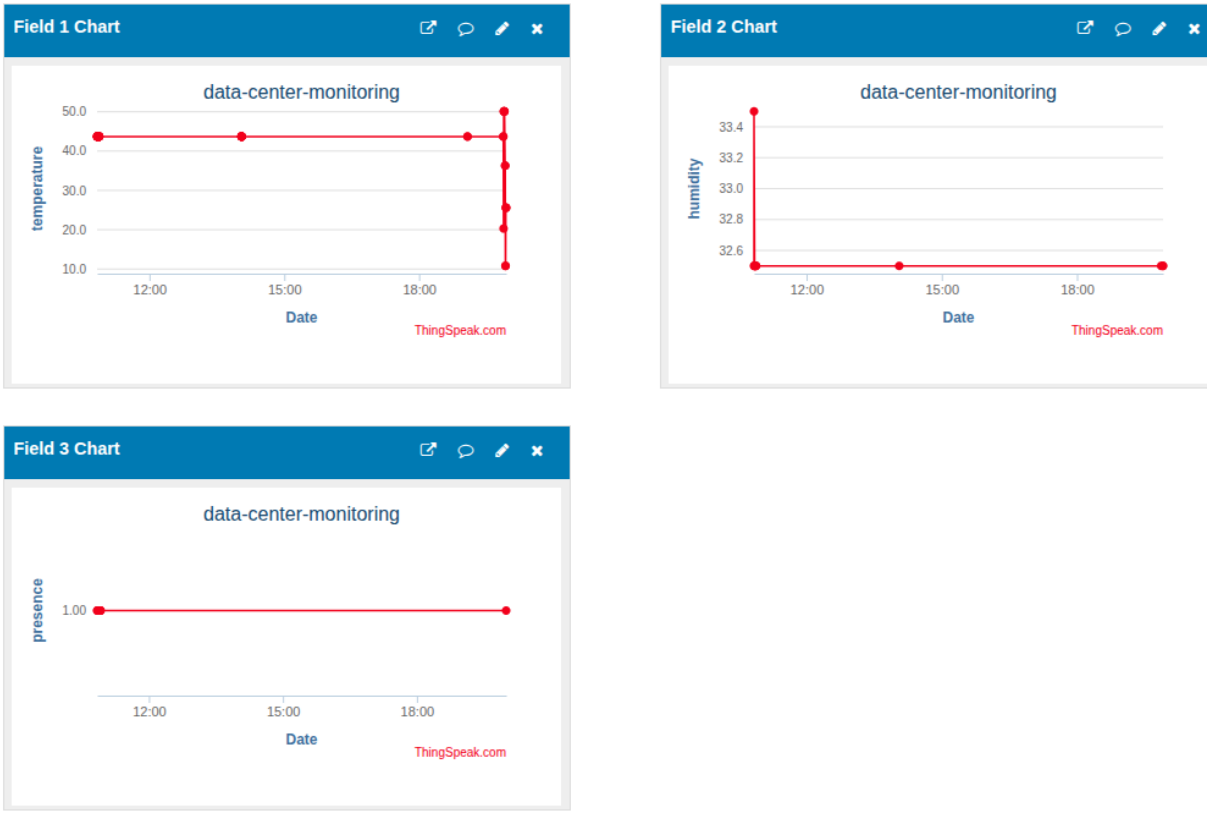


Figura 4.2: Registro mostrado pelo Thingspeak

## 4.3 Testes de Latência e Estabilidade

Análise da comunicação baseada nos logs obtidos.

### 4.3.1 Análise de Latência (Delay)

Para quantificar a latência de ponta a ponta (*end-to-end delay*) do ecossistema IoT proposto, estabeleceu-se um protocolo de observação baseado em um único pulso de dados. A análise consiste em monitorar o diferencial de tempo entre a publicação na origem e a persistência no destino final.

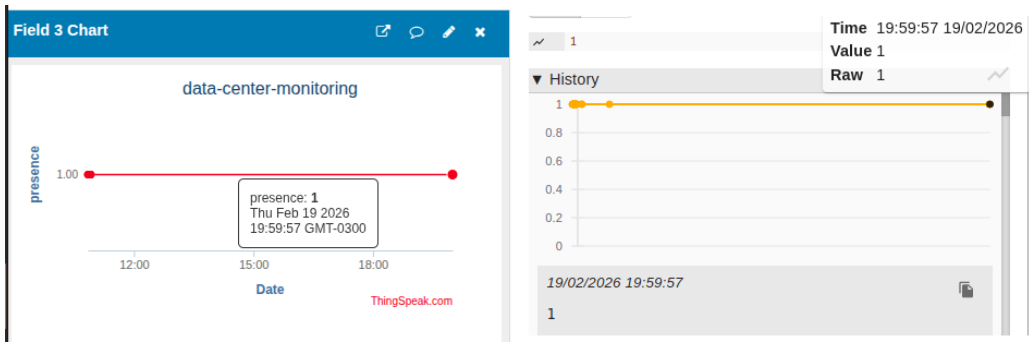


Figura 4.3: Terminal serial do simulador Wokwi

**Procedimento Experimental**

O cálculo do *delay* será realizado através da inspeção de carimbos de tempo (*timestamps*) em duas etapas críticas:

1. **Ingresso (HiveMQ):** O tempo de chegada do dado ao *broker* intermediário é monitorado via *MQTT Explorer*, servindo como t0 (referência inicial).
2. **Egresso (ThingSpeak):** O tempo de atualização nos servidores do *ThingSpeak* é registrado como tf (referência final).

A latência total é definida pela equação: $D = t_f - t_0$.

### Filtragem e Encaminhamento de Dados

Conforme ilustrado na Figura 4.3, se mostra o dados que chegaram no (a) thingspeak e o (b) no Mqtt explorer. Contudo, para fins de simplificação da análise de latência e isolamento de variáveis, o *gateway* desenvolvido em *Python* foi configurado de forma seletiva apenas para enviar dados, neste caso PIR (sensor de presença).

Embora o *hardware* (ESP32) publique o conjunto completo de telemetria no *broker HiveMQ*, o *script* de *gateway* foi configurado para atuar como um filtro lógico seletivo. Nesta arquitetura, apenas uma variável pré-definida é reconhecida e convertida para o protocolo do *ThingSpeak*. Esta abordagem de isolamento visa mitigar interferências causadas por múltiplas requisições simultâneas, permitindo uma análise mais controlada da performance do canal de comunicação.

Ao confrontar os registros de tempo (*timestamps*) no *MQTT Explorer* e no painel do *ThingSpeak*, observa-se que os horários de chegada convergem para o mesmo valor na escala de segundos. Contudo, devido à limitação das interfaces de monitoramento, que não disponibilizam a granularidade em milissegundos, conclui-se que:

- A latência total do sistema (*end-to-end*) é inferior a 1 segundo;
- O sistema apresenta um comportamento de tempo real adequado para aplicações de monitoramento de *datacenters*;
- A latência real encontra-se abaixo do limiar de detecção das ferramentas utilizadas, caracterizando uma transmissão de alta eficiência.

### 4.3.2 Análise de Estabilidade do Sistema

A estabilidade da infraestrutura de comunicação foi avaliada com base em três pilares fundamentais, observados durante um período de teste contínuo:

1. **Taxa de Retenção de Conexão (*Uptime*):** Verificação de quedas de conexão nos *brokers*. No *HiveMQ*, a estabilidade é confirmada pela permanência do *Client ID* ativo no console. No *ThingSpeak*, a estabilidade é observada pela ausência de "gaps"(buracos) nos gráficos temporais.

2. **Consistência do Intervalo de Publicação:** Um sistema estável deve manter o intervalo entre envios (ex: 15 segundos) constante. Oscilações frequentes nesse intervalo indicam instabilidade no *script* de *gateway* ou congestionamento na rede.

3. **Integridade dos Dados (Perda de Pacotes):** Comparação entre o número de mensagens enviadas pelo terminal serial do *Wokwi* e o número de pontos registrados no *ThingSpeak*.

**Métrica de Confiabilidade**

A estabilidade ($S$) pode ser expressa pela razão entre pacotes recebidos ($P_{rx}$) e pacotes transmitidos ($P_{tx}$):

$$S = \left( \frac{P_{rx}}{P_{tx}} \right) \times 100\%$$

Valores próximos a 100% indicam um *gateway* robusto e uma configuração de *Keep Alive* adequada no protocolo MQTT.

Conforme ilustrado na Figura 4.1, os dados transmitidos pelo broker HiveMQ foram integralmente captados pelo MQTT Explorer. Essa integridade na recepção demonstra a estabilidade do broker.

A Figura 4.2 apresenta os dados recebidos a partir do gateway Python. Observa-se uma densidade de pontos menor em comparação aos capturados pelo MQTT Explorer. Isso ocorre porque o ThingSpeak limita o envio de dados a intervalos de 15 segundos; portanto, o sistema, ao incluir este dashboard, apresenta instabilidade.

# Capítulo 5

# Documentação

## 5.1 GitHub

https://github.com/khipucode/MQTT-to-ThingSpeak-IoT-Streaming-Integration

## 5.2 Vídeo Demonstrativo

Link do vídeo demonstrando o funcionamento do sistema:

https://youtu.be/wQ84ej0GdCg

# Capítulo 6

# Conclusão

## 6.1 Resultados Obtidos

Os principais resultados alcançados neste trabalho podem ser sumarizados da seguinte forma:

- A arquitetura distribuída de streaming MQTT foi implementada e validada com sucesso, comprovando o funcionamento completo do fluxo de comunicação entre o dispositivo IoT simulado, o broker intermediário (HiveMQ), o gateway em Python e a nuvem (ThingSpeak).
- A introdução de um broker intermediário confirmou os benefícios de uma arquitetura industrial moderna, garantindo o desacoplamento do sistema e proporcionando visibilidade e controle do fluxo de dados por meio de ferramentas de inspeção como o MQTT Explorer.
- Na análise de latência, o ecossistema demonstrou alta eficiência, apresentando um atraso de ponta a ponta (*end-to-end delay*) inferior a 1 segundo, caracterizando um comportamento de tempo real adequado para o monitoramento de datacenters.
- A camada intermediária baseada no HiveMQ provou ser robusta, captando integralmente os dados transmitidos e garantindo a integridade e a estabilidade da comunicação MQTT na recepção.
- A integração com o ThingSpeak evidenciou limitações práticas de plataformas de nuvem restritivas, visto que a imposição de um intervalo de 15 segundos para a recepção de dados resultou em uma amostragem esparsa e gerou instabilidade na visualização final do *dashboard*.

## 6.2 Trabalhos Futuros

Com base nas análises realizadas, propõem-se as seguintes melhorias e continuidades para este projeto:

- **Agregação de Dados no Gateway:** Desenvolver lógicas mais complexas no script Python para processar, filtrar ou agregar as informações em médias temporais antes do envio, mitigando as limitações de frequência do ThingSpeak e enviando apenas dados estruturados e consolidados.

- **Testes de Estabilidade com Carga Completa:** Realizar novas medições de latência e de perda de pacotes enviando simultaneamente todas as variáveis de telemetria (temperatura, umidade e alertas), expandindo a análise atual que foi simplificada de forma seletiva apenas para os eventos do sensor PIR.
- **Alternativas de Visualização na Nuvem:** Explorar a substituição do ThingSpeak por plataformas ou bancos de dados que suportem a alta frequência de dados gerada pelo ESP32, eliminando a instabilidade e garantindo a granularidade necessária para o monitoramento contínuo.
- **Validação em Hardware Físico:** Transpor a arquitetura atualmente validada via simulação no Wokwi para componentes físicos reais, avaliando o comportamento dos sensores (DHT22 e AS312) e a latência de rede em um ambiente físico de datacenter.