Universidade do Minho
Escola de Engenharia

Mestrado em Engenharia Eletrónica Industrial e Computadores

# Digitalização Industrial

*Discentes:*
José Alfredo Ribeiro Rodrigues, PG53961
José Vicente Pereira, PG53988
João Henrique Outeiro Ferreira, PG53914
José Roberto Costa Pinto, PG53987
Alexandre Pereira Pinho, PG50167

# Conteúdo


**1 Introdução e Contextualização** **3**

**2 Sensor Si7021** **4**
2.1 Estrutura Interna e Princípio de Funcionamento . . . . . 4
2.1.1 Medição de Temperatura . . . . . 4
2.2 Conversão dos dados . . . . . 4
2.2.1 Cálculo da Temperatura . . . . . 5
2.3 Comunicação via I2C . . . . . 5

**3 Arquitetura *STM-Raspberry* (Modbus)** **6**

**4 Arquitetura *Raspberry-PC* (OPC-UA)** **8**

**5 Arquitetura *PC-Cloud*** **9**
5.1 *AWS IoT Core* . . . . . 9
5.1.1 Dispositivo IoT . . . . . 9
5.2 *Node-Red* . . . . . 12
5.2.1 Instalação do *Node-Red* e *AWS-IoT Hub* . . . . . 12
5.2.2 Fluxo *Node-red* para a fase de desenvolvimento . . . . . 13
5.3 *AWS-DynamoDB* . . . . . 13
5.3.1 Configuração da base de dados . . . . . 14
5.4 *Display* dos Dados . . . . . 15
5.5 Reestruturação do fluxo *Node-Red* para *Python* . . . . . 15

**6 Testes e Resultados** **16**

# Lista de Figuras


1 Sensor Si7021 . . . . . 4
2 Fórmula que permite calcular a temperatura lida pelo sensor. . . . . 5
3 Endereço do comandos de medição de temperatura. . . . . 5

4 Sensor-STM32-Rasp System . . . . . 6
5 *Data Frame* utilizado no protocolo . . . . . 7

6 Rapsberry connected to the Computer . . . . . 8

7 Fluxo de operações AWS IoT . . . . . 9
8 Propriedades da *Thing* . . . . . 10
9 Certificados do dispositivo . . . . . 10
10 Políticas de utilização do dispositivo . . . . . 11
11 Informação do dispositivo IoT . . . . . 11
12 *Ping* ao *endpoint* . . . . . 12
13 Instalação aws-iot-hub na GUI . . . . . 12
14 *Feedback* da instalação no terminal . . . . . 13
15 Fluxo *Node-red* . . . . . 13
16 Mensagem MQTT recebida no menu de testes da AWS . . . . . 13
17 Criação da tabela e configuração de chaves . . . . . 14
18 Tabela operacional em fase de testes . . . . . 14
19 Representação visual com valores simulados no *Node-red* . . . . . 15

20 Envio de dados da Raspberry e PC . . . . . 16
21 Dados armazenados na base de dados . . . . . 17
22 Representação da temperatura no gráfico . . . . . 17

# Capítulo 1

# Introdução e Contextualização

A digitalização industrial tornou-se crucial para aumentar a eficiência e precisão dos processos em ambientes de produção e automação. Neste contexto, a integração de sensores, microcontroladores e plataformas de computação possibilita a monitorização em tempo real e a análise detalhada de dados. Este relatório descreve o desenvolvimento de um sistema de monitorização e controlo, onde um sensor de temperatura e humidade é utilizado para recolher dados ambientais, transmitidos para uma unidade central através de uma arquitetura modular.

A arquitetura é composta por uma série de componentes conectados, incluindo uma STM32, uma Raspberry Pi, um PC e um sistema de armazenamento e monitorização na *cloud* (AWS). Cada componente desempenha um papel específico na recolha e transmissão dos dados, com protocolos de comunicação que garantem integração eficiente entre eles.

O sistema opera da seguinte forma:

- **Sensores de temperatura e humidade** realizam o registo da temperatura, fornecendo informações cruciais para monitorização.
- A **STM32** atua como unidade local, responsável por ler os dados dos sensor através do protocolo I2C, operando como *slave* na comunicação.
- A **Raspberry Pi** serve como ponto intermediário, simulando uma estação de integração que recebe dados de uma ou várias STM32 usando o protocolo Modbus RTU. Depois, transmite os dados para o PC utilizando o protocolo OPC-UA.
- O **PC** funciona como unidade central, onde os dados provenientes da Raspberry Pi são consolidados e processados. O PC utiliza o protocolo MQTT para enviar os dados processados para o sistema na cloud (AWS), onde ficam armazenados e disponíveis para monitorização e análise remota.

Com esta estrutura modular e a integração de diferentes protocolos de comunicação, o sistema permite a análise detalhada e em tempo real dos dados recolhidos. A conectividade com a *cloud* permite o armazenamento seguro e o acesso remoto às informações, contribuindo para a eficiência e segurança na gestão de dados industriais. O projeto demonstra, assim, a aplicação prática de digitalização e Internet das Coisas (IoT) em ambientes de produção, reforçando o papel da tecnologia na otimização de processos industriais e na criação de uma base sólida para a automação e a tomada de decisões informadas.

# Capítulo 2

# Sensor Si7021

O Si7021, figura 1, é um sensor digital amplamente utilizado para a medição de temperatura e humidade relativa. Fabricado pela Silicon Labs, oferece alta precisão, baixo consumo de energia e comunicação simples via I2C, tornando-o ideal para aplicações em sistemas de HVAC (aquecimento, ventilação e ar condicionado), estações meteorológicas e dispositivos IoT.


Figura 1: Sensor Si7021

## 2.1 Estrutura Interna e Princípio de Funcionamento

O Si7021 contém dois tipos de sensores: um sensor para humidade e outro para temperatura, ambos utilizando princípios físicos distintos para realizar as medições. A seguir, descrevem-se os detalhes de funcionamento interno do sensor.

### 2.1.1 Medição de Temperatura

A medição de temperatura é feita utilizando um sensor resistivo, cuja resistência elétrica varia conforme a temperatura do ambiente. Este tipo de sensor pode ser um termístor ou um RTD (*Resistance Temperature Detector*). Tal como na medição de humidade, o sinal analógico gerado pela variação de resistência é convertido num valor digital através de um ADC interno.

## 2.2 Conversão dos dados

Os dados lidos do sensor Si7021 estão em formato bruto e precisam ser convertidos para valores reais de humidade e temperatura. O *datasheet* especifica as seguintes fórmulas para a conversão:

### 2.2.1 Cálculo da Temperatura

$$\text{Temperature (°C)} = \frac{175.72 * \text{Temp\_Code}}{65536} - 46.85$$

Figura 2: Fórmula que permite calcular a temperatura lida pelo sensor.

Da mesma forma, na figura 2, *Temp_code* refere-se ao valor bruto da temperatura lido pelo sensor. Estas fórmulas permitem que os dados digitais sejam convertidos em unidades utilizáveis para monitorização ambiental.

Esta fórmula baseia-se nas especificações do *datasheet* do sensor Si7021, que define as constantes (175.72 e 46.85) necessárias para converter os valores brutos de 16 bits em valores de temperatura (°C). A aplicação desta fórmula permite transformar os dados do sensor em medidas utilizáveis e com significado físico, facilitando a interpretação das leituras obtidas na forma de valores reais de temperatura.

## 2.3 Comunicação via I2C

A comunicação entre o sensor Si7021 e a STM32 é realizada através do protocolo I2C, no qual o microcontrolador atua como master e o sensor como slave. Neste protocolo, o master controla o início e o fim das transações de dados, além de selecionar o slave com o qual deseja comunicar, utilizando um endereço específico. No caso do Si7021, o endereço padrão é **0x40**, o que permite ao STM32 identificar e comunicar-se com o sensor.

| Command Description | Command Code |
|---|---|
| Measure Relative Humidity, Hold Master Mode | 0xE5 |
| Measure Relative Humidity, No Hold Master Mode | 0xF5 |
| Measure Temperature, Hold Master Mode | 0xE3 |
| Measure Temperature, No Hold Master Mode | 0xF3 |

Figura 3: Endereço do comandos de medição de temperatura.

Para obter dados do Si7021, a STM32 envia um comando específico ao sensor, como 0xE3 para uma leitura de temperatura, representados na figura 3. Este tipo de comando instrui o sensor (slave) a realizar a medição correspondente e a devolver os dados ao microcontrolador. A comunicação bidirecional via I2C facilita a troca de informações entre os dispositivos, mantendo o microcontrolador no controlo de todas as operações e garantindo uma interação precisa e eficiente entre o master e o slave.

# Capítulo 3

# Arquitetura *STM-Raspberry* (Modbus)

Para conectar a *STM32* à Raspberry Pi, recorreu-se ao protocolo Modbus.

Neste contexto, a STM32 é responsável pela recolha de dados do sensor, normalmente situada junto do mesmo. A Raspberry Pi atua como controlador intermédio, centralizando as leituras de vários sensores numa secção específica e simplificando o fluxo de informações.

Este protocolo industrial permite a ligação dos sensores e respetivo sistema de aquisição (STM32) a uma unidade mais distante, além de suportar a conexão de múltiplos dispositivos numa mesma rede, onde um mestre controla a comunicação com um ou mais escravos, facilitando a troca de dados de forma fiável e escalável.

O protocolo pode ser utilizado em duas versões: RTU, que usa transmissão série e permite o envio de dados em formato binário ou ASCII, e TCP/IP, que realiza a transmissão através de rede, encapsulando os dados em pacotes TCP/IP.

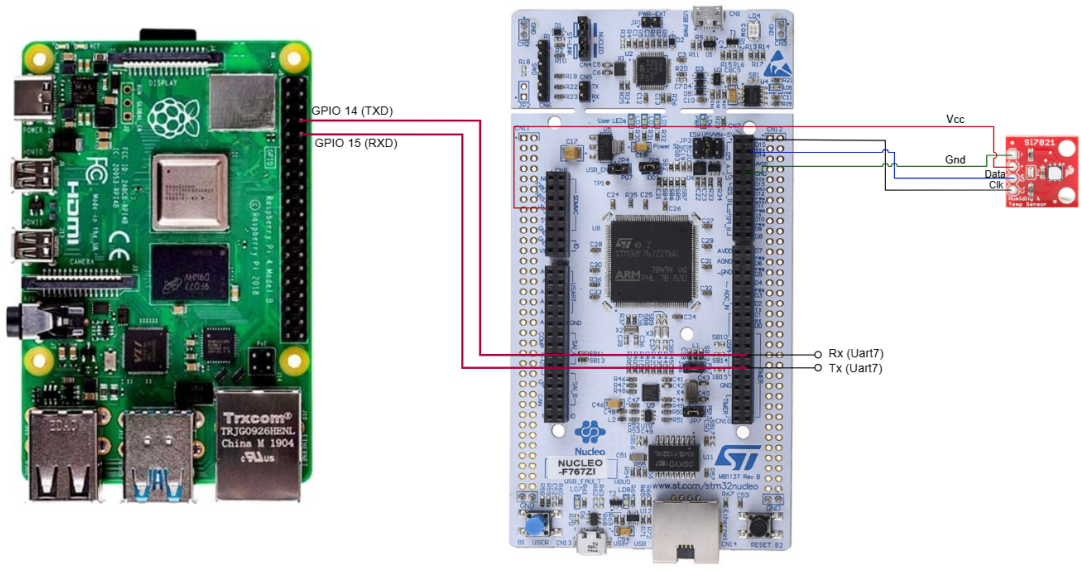


Figura 4: Sensor-STM32-Rasp System

Como se pode verificar na figura 4, a versão escolhida do protocolo foi a RTU, que utiliza transmissão série. Para implementar esta comunicação, usou-se a UART, conectando o pino RX (receção) da Raspberry Pi ao pino TX (transmissão) da STM32, e o pino TX da Raspberry Pi ao pino RX da STM32.

Como as plataformas são diferentes e o nível de abstração das bibliotecas varia, existiam algumas discrepâncias temporais e de formatação das tramas. Assim, para obter

maior controlo sobre o processo e melhorar o desempenho, uma vez que o protocolo possui muitas funcionalidades desnecessárias neste caso específico, optou-se por implementar de raiz a versão binária do protocolo Modbus RTU em ambas as plataformas, incluindo apenas as funcionalidades básicas necessárias, o que permitiu também adquirir um maior conhecimento sobre o protocolo.

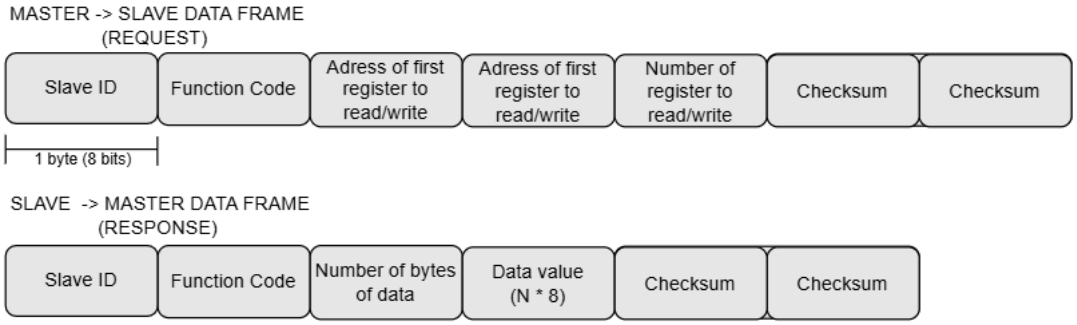


Figura 5: *Data Frame* utilizado no protocolo

Como se pode verificar na Figura 5, o slave é selecionado através do seu número correspondente. Em seguida, é enviado o function code, que permite definir a operação desejada. Depois, são especificados os registos a serem alterados e, no final, é adicionado um sistema de verificação de erros. O mesmo ocorre na resposta do slave, mas, desta vez, é enviada a informação solicitada para o master. Quando o slave não é o selecionado, a informação recebida é ignorada e descartada.

# Capítulo 4

# Arquitetura *Raspberry-PC* (OPC-UA)

O OPC UA (Unified Architecture) é um protocolo assíncrono utilizado para comunicação em sistemas industriais. Neste contexto, será utilizado para conectar as Raspberry Pis, que funcionam como pontos intermediários industriais. Cada Raspberry Pi agrupa os dados de um conjunto de sensores e envia essas informações para o PC, que atua como uma estação de supervisão central. A ligação entre a Raspberry Pi e o PC simula uma interface central que recolhe e transmite dados das diferentes localizações intermediárias para o sistema central, permitindo a monitorização e análise em tempo real das informações provenientes de todos os sensores.

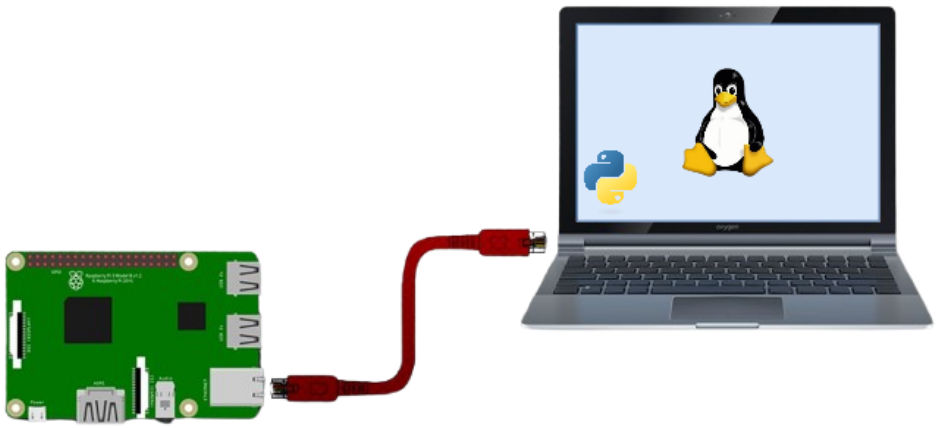

Figura 6: Rapsberry connected to the Computer

Os elementos básicos de dados no OPC UA são os nós e as referências.

Cada unidade de dados, como a leitura de um sensor de temperatura, é representada por um nó, e todos os nós se conectam entre si através de referências que descrevem o relacionamento entre eles. Neste caso, por se tratar de um exemplo simples, foi utilizado um único nó responsável por armazenar o valor da temperatura, atualizado sempre que o sistema local (Stm32) envia uma nova leitura. Este valor será publicado pelo servidor (Raspberry Pi), permitindo que o cliente (PC) o leia sempre que necessário.

Para implementar o protocolo OPC UA tanto no lado servidor quanto no lado cliente, foi utilizada a biblioteca **FreeOpcUa** em Python. Os clientes conectam-se ao servidor OPC UA através de um **Endpoint URL**, que fornece acesso à estrutura de dados do servidor. Neste caso, o Endpoint URL utilizado foi `opc.tcp://<IP_RASP>:4840/`, onde `<IP_RASP>` representa o endereço IP da Raspberry Pi e `4840` a porta utilizada.

# Capítulo 5

# Arquitetura *PC-Cloud*

## 5.1 *AWS IoT Core*

O *AWS IoT Core* é um serviço gerido pela *Amazon Web Services* que facilita a conexão segura de dispositivos IoT à *cloud*, permitindo que eles troquem dados com aplicações e outros dispositivos.

Este oferece suporte a milhões de dispositivos e permite a recolha, processamento e análise de dados em tempo real. Com recursos de segurança como autenticação, autorização e criptografia, o *AWS IoT Core* garante a segurança e integridade dos dados transmitidos. Além disso, integra-se com outros serviços da *AWS*, como *Lambda* e S3, para processamento e armazenamento eficiente dos dados IoT.

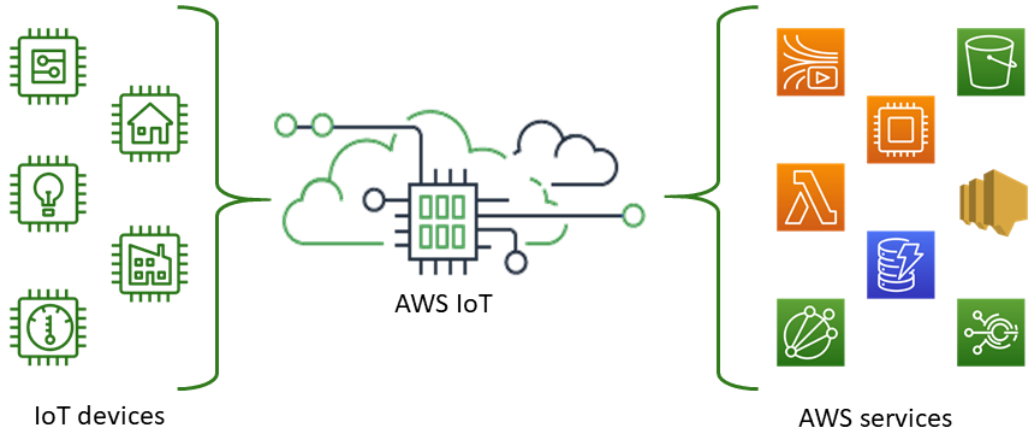


Figura 7: Fluxo de operações AWS IoT

### 5.1.1 Dispositivo IoT

Inicialmente, foi necessário criar um dispositivo IoT no *AWS IoT Core*, configurando-o para conectar e comunicar com a plataforma de forma segura e eficiente. Este processo incluiu o registro do dispositivo no sistema, a gestão de certificados para autenticação, e a definição de políticas de acesso que garantem a integridade e segurança dos dados.

De seguida, foram realizadas configurações de comunicação MQTT, permitindo ao dispositivo enviar e receber dados em tempo real. Esta configuração inicial foi fundamental para assegurar a integração fluida do dispositivo com o ambiente de *cloud* e outros serviços da AWS.

### Criação e configuração de um dispositivo IoT

Para realizar a criação de um dispositivo IoT no *AWS IoT Core* é necessário ir ao menu "*Manage*"e entrar na categoria de "*Things*"e criar um dispositivo no botão "*Create Thing*". Dentro do menu de criação de um dispositivo IoT são exigidos o cumprimento de 3 passos:

1. Especificação de Propriedades do dispositivo

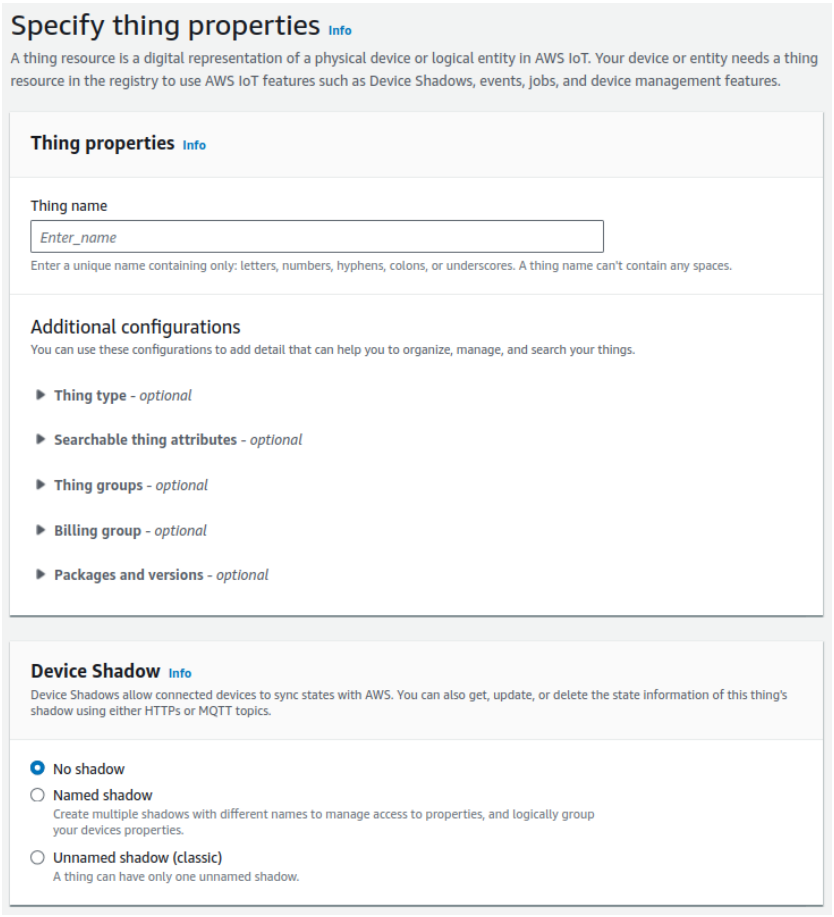


Figura 8: Propriedades da *Thing*

2. Configuração e *Download* da certificação

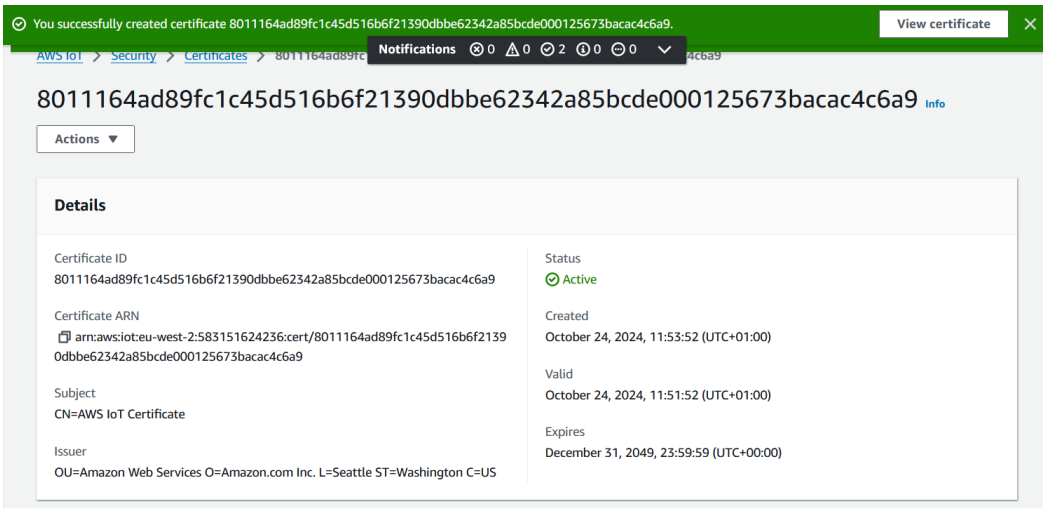


Figura 9: Certificados do dispositivo

3. Anexação das políticas de utilização ao dispositivo

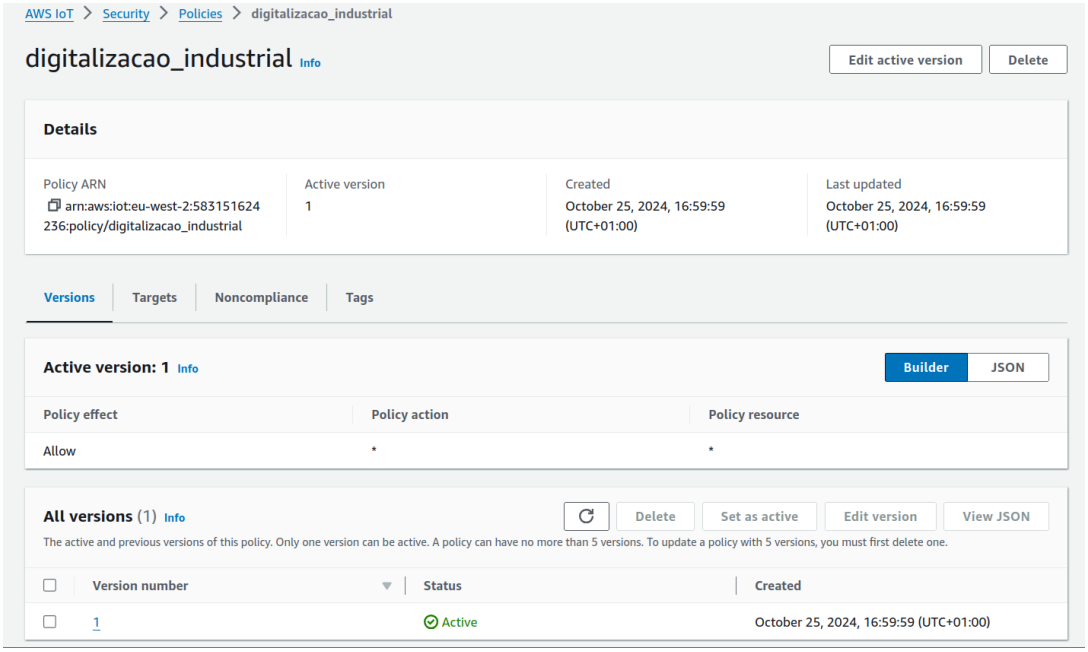


Figura 10: Políticas de utilização do dispositivo

Após a conclusão dos passos referidos anteriormente obtém-se a informação genérica à cerca do dispositivo como o nome e o seu *endpoint*, como é possível observar na figura 11

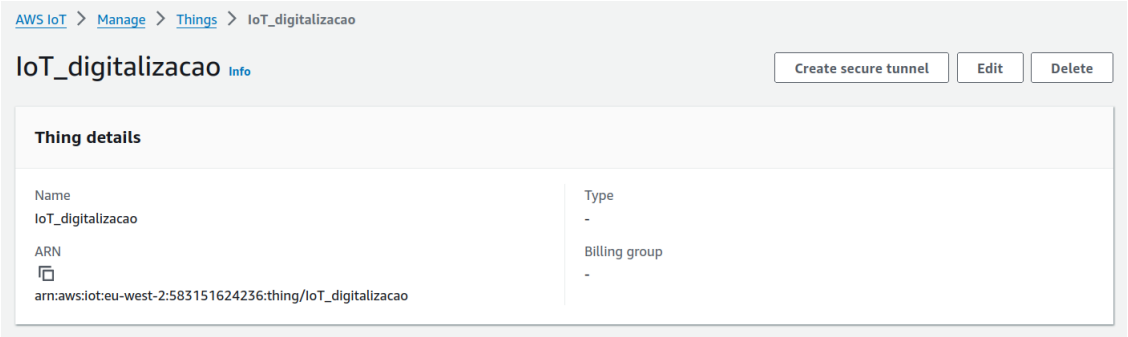


Figura 11: Informação do dispositivo IoT

### Teste de conectividade ao *endpoint*

Concluindo todos os passos de criação e configuração do dispositivo IoT prossegue-se para a realização do teste de conectividade que irá indicar se o *endpoint* previamente configurado está pronto para ser utilizado. Para realizar este teste simplesmente é feito um *ping* ao *endpoint* e verifica-se o estado de conexão entre PC e *endpoint*.

Observando a figura 14 é possível observar que o teste é bem sucedido, transferindo 100% dos pacotes com 0% de perdas na transmissão, concluindo que a criação do dispositivo IoT e a devida configuração e comunicação com os serviços *cloud* foi bem sucedida.

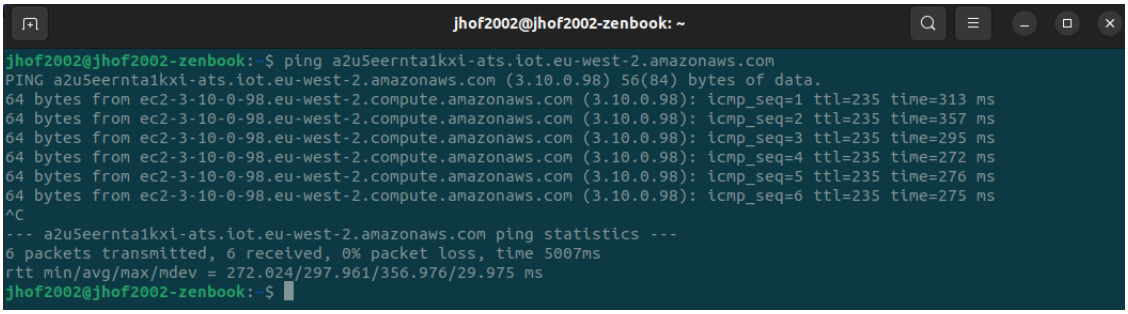


Figura 12: *Ping* ao *endpoint*

## 5.2 *Node-Red*

O *Node-RED* é uma ferramenta de desenvolvimento baseada em fluxo, projetada para facilitar a integração de dispositivos, APIs e serviços *online*, especialmente no contexto IoT. Esta ferramenta possui uma interface visual intuitiva, permitindo a criação de fluxos de dados conectando blocos predefinidos que representam certas operações, reduzindo significativamente a complexidade de programação e facilita a prototipagem.

Neste projeto, o *Node-RED* foi utilizado para simular e gerir dados provenientes dos dispositivos de baixo nível, como o sensor e o microcontrolador STM32. A ferramenta permitiu a criação de fluxos que recebiam, processavam e encaminhavam os valores simulados desses dispositivos, possibilitando uma comunicação confiável com o sistema IoT. Este uso do *Node-RED* facilitou o teste e a validação do sistema, simulando as condições reais de operação e garantindo que os dados gerados pelo sensor e microcontrolador fossem adequadamente integrados ao ambiente de monitorização.

### 5.2.1 Instalação do *Node-Red* e *AWS-IoT Hub*

Para instalar o *Node-Red* executou-se o comando **"sudo npm install -g –unsafe-perm node-red"** e dentro da interface gráfica no menu *"Manage Palettes"* foi instalado o AWS-IoT Hub, responsável por adicionar e gerir os blocos de *node-red* relativamente à conexão com os serviços AWS.

As figuras abaixo representam o processo de instalação do AWS-IoT Hub na interface do node-red, assim como o *feedback* de instalação no terminal.

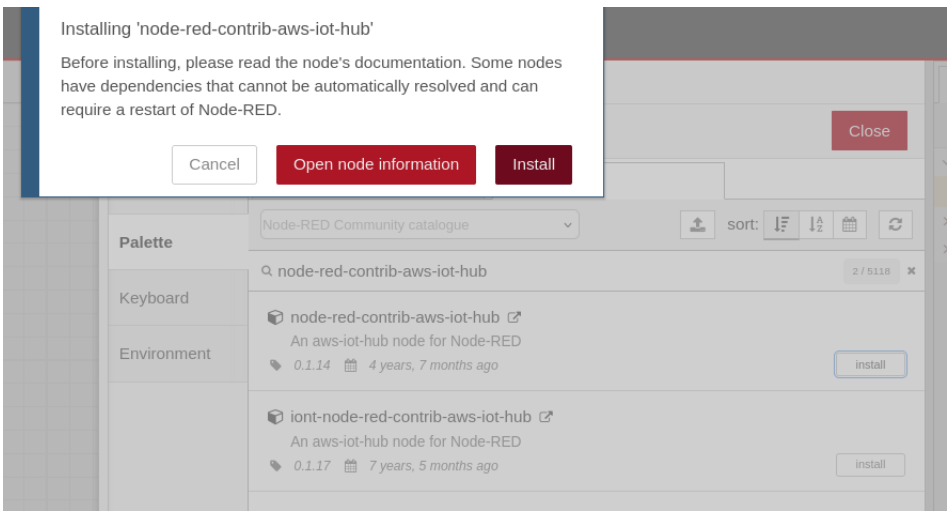


Figura 13: Instalação aws-iot-hub na GUI

```
24 Oct 22:49:08 - [warn] Encrypted credentials not found
24 Oct 22:49:08 - [info] Server now running at http://127.0.0.1:1880/
24 Oct 22:49:08 - [info] Starting flows
24 Oct 22:49:08 - [info] Started flows
24 Oct 22:53:55 - [info] Installing module: node-red-contrib-aws-iot-hub, version: 0.1.14
24 Oct 22:54:00 - [info] Installed module: node-red-contrib-aws-iot-hub
24 Oct 22:54:00 - [info] Added node types:
24 Oct 22:54:00 - [info]  - node-red-contrib-aws-iot-hub:aws-iot-device
24 Oct 22:54:00 - [info]  - node-red-contrib-aws-iot-hub:aws-mqtt out
24 Oct 22:54:00 - [info]  - node-red-contrib-aws-iot-hub:aws-mqtt in
24 Oct 22:54:00 - [info]  - node-red-contrib-aws-iot-hub:aws-thing
```

Figura 14: *Feedback* da instalação no terminal

### 5.2.2 Fluxo *Node-red* para a fase de desenvolvimento

O fluxo *Node-red* foi configurado com um bloco de injeção para realizar o envio de mensagens, um bloco de função que adicionava o conteúdo das mensagens, neste caso o *value* e o *timestamp* e mais dois blocos, um de conexão ao *endpoint* da AWS e outro bloco de *debug* local para observar o estado das operações realizadas.

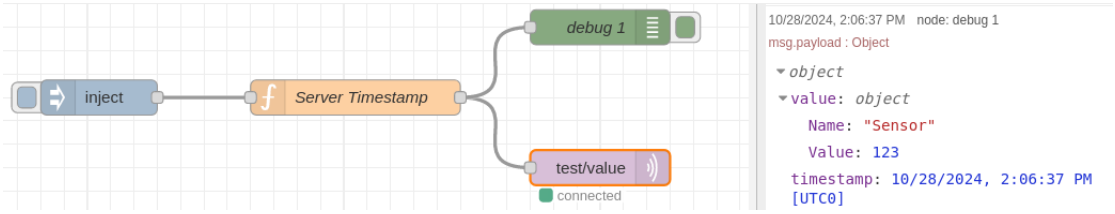


Figura 15: Fluxo *Node-red*


Figura 16: Mensagem MQTT recebida no menu de testes da AWS

## 5.3 *AWS-DynamoDB*

O Amazon DynamoDB é um serviço de banco de dados NoSQL gerido pela AWS, projetado para oferecer alto desempenho e escalabilidade. Este suporta estruturas de dados chave-valor, sendo otimizado para armazenar e acessar grandes volumes de dados com baixa latência. DynamoDB é amplamente utilizado em aplicações que exigem respostas rápidas e operações de leitura e escrita intensivas, como aplicações IoT, jogos e e-commerce.

### 5.3.1 Configuração da base de dados

Para se criar e configurar a base de dados foi necessário seguir um conjunto de passos como a criação de uma tabela, atribuição de nome, configuração de chaves - *Partition Key - "timestamp"* e *Sort Key - "value"* e configuração das regras de acesso e operações de leitura/escrita.

Na figura 17 demonstra-se a configuração da tabela, com a atribuição de nome e chaves de identificação. Já a figura 18 representa a tabela já operacional com valores estáticos injetados para realizar o teste de funcionamento, onde é possível ver o *timestamp* e *value* das mensagens recebidas via MQTT, armazenadas na base de dados.

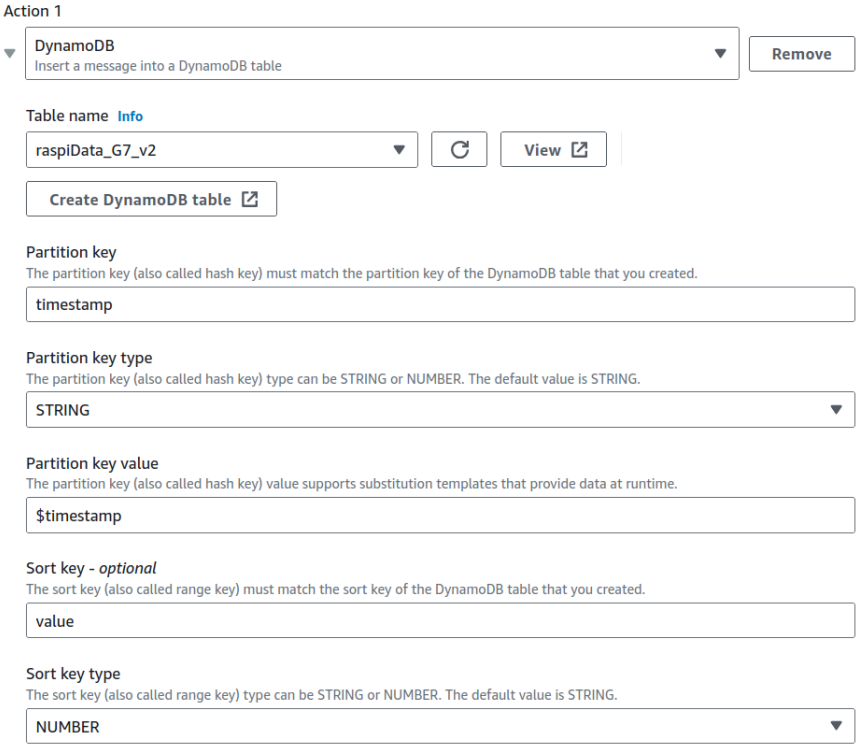


Figura 17: Criação da tabela e configuração de chaves

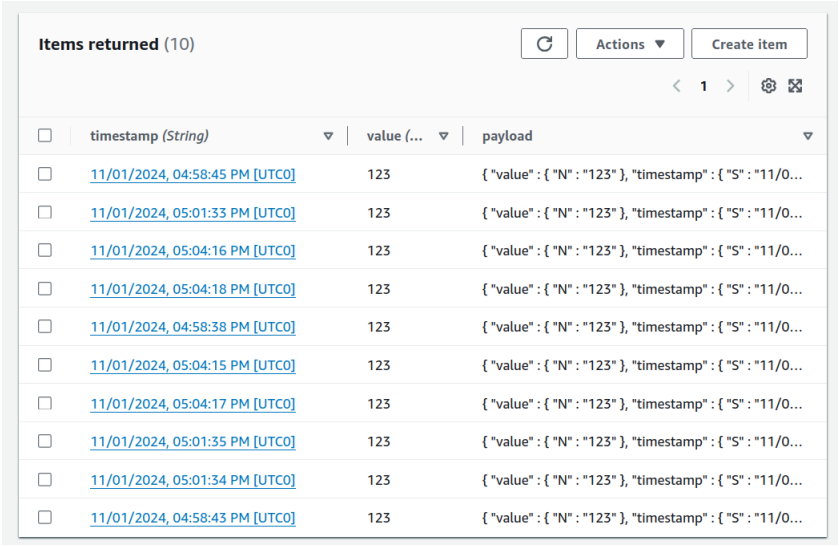


Figura 18: Tabela operacional em fase de testes

## 5.4 *Display* dos Dados

Neste projeto, utilizou-se um *script* em *Python* para exibir os dados recolhidos, utilizando a biblioteca *Streamlit* para criar uma interface intuitiva.

Para aceder à base de dados recorreu-se à biblioteca *boto3*, utilizando chaves de acesso da AWS para realizar a conexão e recolha dos dados segura. Os dados foram então recolhidos e processados no *script* para a visualização, exibindo um grafico dos valores da temperatura(*value*) em função do tempo (*timestamp*), facilitando a análise dos dados armazenados.

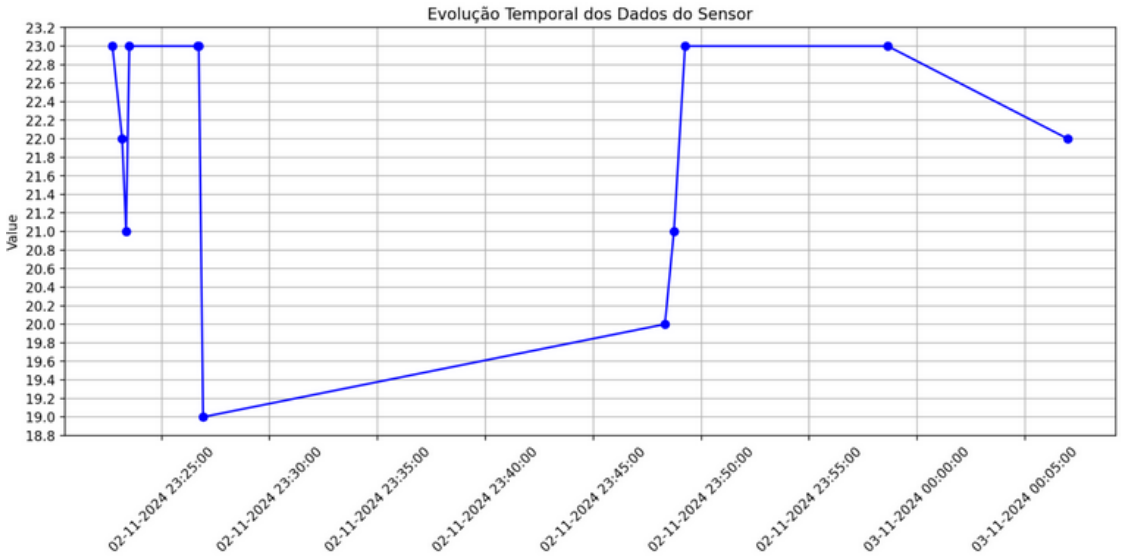


Figura 19: Representação visual com valores simulados no *Node-red*

## 5.5 Reestruturação do fluxo *Node-Red* para *Python*

Para o *deploy* final deste projeto, foi realizada uma reestruturação do fluxo de dados inicialmente desenvolvido em *Node-RED* para um *script* em *Python*, pretendendo maior controlo e flexibilidade na integração com os serviços da AWS.

O novo *script* foi configurado para atuar como cliente OPC-UA, conectando-se ao servidor OPC-UA, hospedado na *Raspberry Pi* para receber os valores de temperatura em tempo real. De seguida, utilizando a biblioteca *boto3*, o *script* estabelece uma conexão segura com os serviços da AWS, implementando o protocolo TLS com certificados digitais para autenticação e criptografia dos dados.

Os dados de temperatura e *timestamp*, são então enviados para a *cloud* (AWS IoT Core) via MQTT, garantindo comunicação segura e eficiente entre o dispositivo IoT e o ambiente *cloud* da AWS.

# Capítulo 6

# Testes e Resultados

No final do desenvolvimento, com todo o sistema implementado, foram realizados testes gerais, para verificar o correto funcionamento, onde todas as partes comunicam entre si, para criar o fluxo de dados pretendido, desde a leitura do sensor até à demonstração.

Na Figura 20, observa-se à esquerda a recepção de dados na Raspberry Pi, enviados pela STM32, e seu posterior encaminhamento ao PC por meio do protocolo OPC-UA (lado direito). É possível verificar o valor de **24,9** recebidos na Raspberry provientes da Stm32 e o seu envio para o PC através do protocolo OPC-UA.

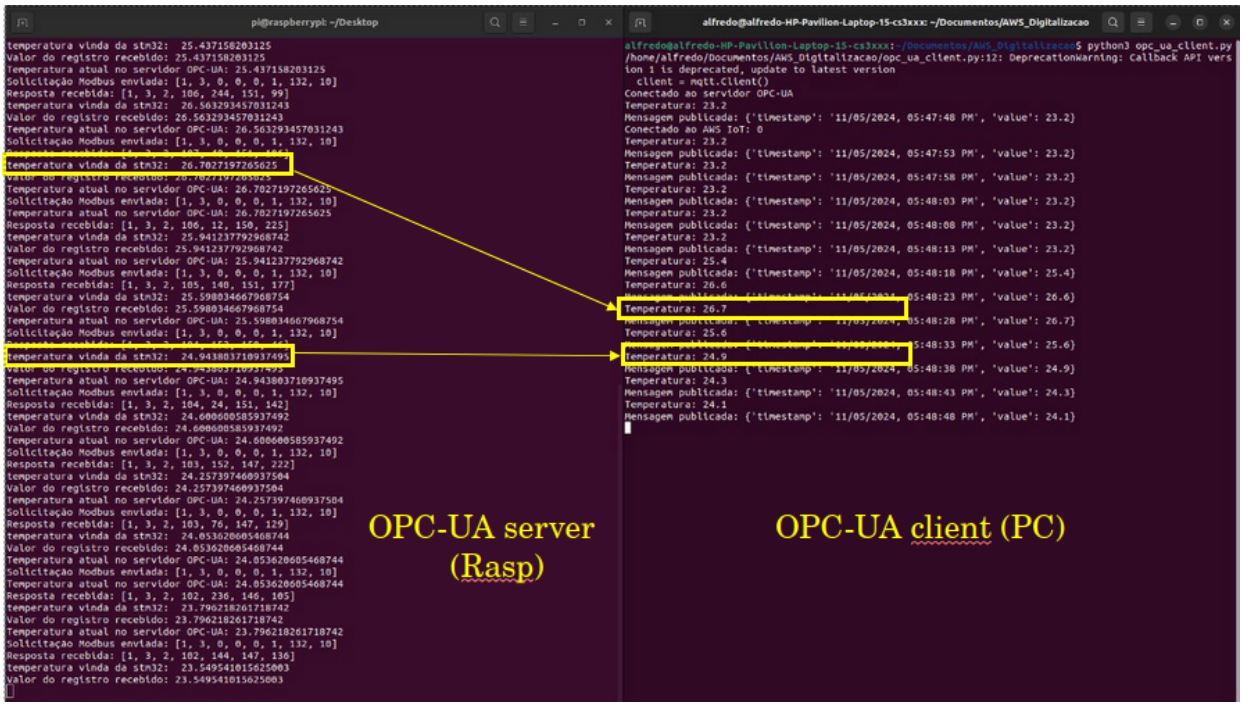


Figura 20: Envio de dados da Raspberry e PC

Quando recebidos no computador, os dados são armazenados na database, com o respetivo dia e hora da leitura, é possível verificar o valor **24,9** lido anteriormente, agora presente na base de dados.

**Items returned** (15)

Actions ▼ | Create item

1

| timestamp *(String)* | value *(Number)* | payload |
|---|---|---|
| 11/05/2024, 05:49:03 PM | 23.4 | { "value" : { "S" : "23.4" }, "timestamp" : { "S" : "11/05/2... |
| 11/05/2024, 05:48:58 PM | 23.5 | { "value" : { "S" : "23.5" }, "timestamp" : { "S" : "11/05/2... |
| 11/05/2024, 05:48:18 PM | 25.4 | { "value" : { "S" : "25.4" }, "timestamp" : { "S" : "11/05/2... |
| 11/05/2024, 05:48:48 PM | 24.1 | { "value" : { "S" : "24.1" }, "timestamp" : { "S" : "11/05/2... |
| 11/05/2024, 05:48:38 PM | 24.9 | { "value" : { "S" : "24.9" }, "timestamp" : { "S" : "11/05/2... |
| 11/05/2024, 05:48:43 PM | 24.3 | { "value" : { "S" : "24.3" }, "timestamp" : { "S" : "11/05/2... |
| 11/05/2024, 05:48:33 PM | 25.6 | { "value" : { "S" : "25.6" }, "timestamp" : { "S" : "11/05/2... |
| 11/05/2024, 05:48:23 PM | 26.6 | { "value" : { "S" : "26.6" }, "timestamp" : { "S" : "11/05/2... |
| 11/05/2024, 05:48:03 PM | 23.2 | { "value" : { "S" : "23.2" }, "timestamp" : { "S" : "11/05/2... |
| 11/05/2024, 05:48:13 PM | 23.2 | { "value" : { "S" : "23.2" }, "timestamp" : { "S" : "11/05/2... |
| 11/05/2024, 05:47:53 PM | 23.2 | { "value" : { "S" : "23.2" }, "timestamp" : { "S" : "11/05/2... |

Figura 21: Dados armazenados na base de dados

Para possível monitorização os dados mostrados num gráfico final, permitindo a análise dos dados e a verificação do seu correto funcionamento.

Para possibilitar a monitorização, os dados são apresentados num gráfico, permitindo a análise e verificação, aqui o valor **24,9** encontra-se presente, juntamente com a data e hora da leitura.

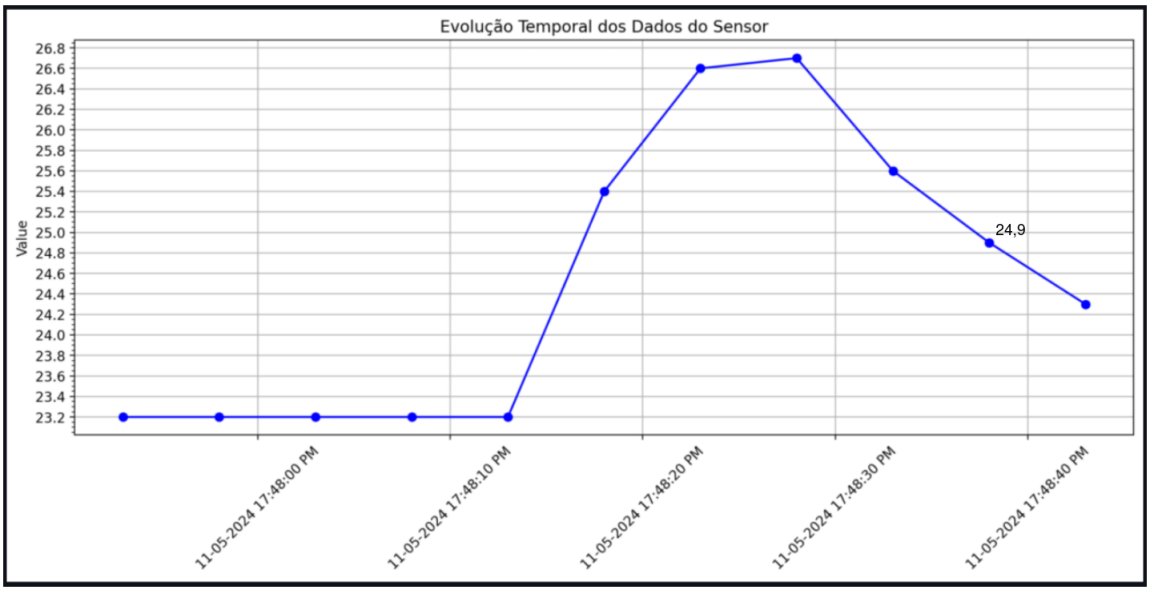


Figura 22: Representação da temperatura no gráfico