

SECRETARIA DE EDUCAÇÃO E CIÊNCIA
INSTITUTO POLITÉCNICO DE BRAGANÇA
ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

LICENCIATURA EM ENGENHARIA DE INFORMÁTICA
6º PERÍODO

FERNANDO SOUZA FURTADO CARRILHO
JOSÉ RAFAEL SOARES BORGES

RELATÓRIO PRÁTICO:
PROTOCOLO MQTT

BRAGANÇA
2023

FERNANDO SOUZA FURTADO CARRILHO
JOSÉ RAFAEL SOARES BORGES

RELATÓRIO PRÁTICO:
PROTOCOLO MQTT

Este relatório objetiva a obtenção de nota na disciplina de Internet das Coisas dos graduandos no curso de Engenharia de Informática da Escola Superior de Tecnologia e Gestão do Insitituto Politécnico de Bragança. Seu conteúdo é composto pela observação, descrição e aplicação referente ao Protocolo MQTT.

BRAGANÇA
2023

Sumário

1	Questão 02: o que é o Eclipse paho?	4
2	Questão 07: troca de Mensagens no MQTT	4
3	Questão 08: análise de código Python	5
4	Questão 10: Arquitetura de interação em residência	7
5	Questão 12: Implementação em MQTT Node-Red e Python 3.10	10
6	Questão 13 - (a) Refatoração do exercício 10 em Node-RED	14
7	Questão 13 - (b) Captação de dados do IPB pelo Node-Red	18

1 **Questão 02: o que é o Eclipse paho?**

De acordo com o Stack (The Things Stack, 2022), o Eclipse Paho é um projeto com missão destinada a fornecer implementações de ferramentas e bibliotecas de alta qualidade e performance para comunicações Machine-to-Machine, isto é, máquina para máquina.

Para a própria Eclipse Foundation, (Eclipse Foundation, 2023), criadora do Eclipse Paho, o *Message Queuing Telemetry Transport* (MQTT) é um meio de transporte leve de mensagens de publicação/assinatura para TCP/IP e protocolos sem conexão (como UDP), respectivamente.

A Eclipse Foundation ainda afirma que o projeto Eclipse Paho fornece implementações de software livre, principalmente do lado do cliente, de MQTT e MQTT-SN em uma variedade de linguagens de programação.

Neste contexto, o Eclipse Paho contém implementações de cliente em protocolo de comunicação MQTT para troca de mensagens entre dispositivos IoT, em diversas linguagens de programação atuais, a exemplo: Csharp, Go, C, Python, JavaScript, Rust, Java e Python.

2 **Questão 07: troca de Mensagens no MQTT**

Dentro do modelo MQTT para que ocorra as trocas de mensagem, é indispensável que exista, no mínimo, um publicador, assinante, broker e tópico. Neste contexto, entende-se como publicador o responsável por inserir/publicar os dados e/ou informações no broker. Consoante, compreende-se o assinante como o ator que solicita os dados e/ou as informações publicadas no broker pelo publicador.

Nesse âmbito, o MQTT é um padrão de troca de mensagens em que opera de maneira desacoplada, isto é, o publicador e assinante podem interagir sem que um conheça o outro, para isso é necessário que ambos conheçam o evento desejado. Diante disso, é possível deduzir que o broker é ator responsável por armazenar os dados e informações inseridas pelo publicador e permitir acesso pelo assinantes e publicadores.

Prontamente, o tópico, em resumo, permite o cliente se conectar com o broker. Seu modo operante é devido ao cliente ter o poder de assinar qualquer tópico de mensagem do broker: os tópicos são o meio de identificação da mensagem. Logo, o tópico é passivo ao conceito similar a URL, em que os níveis são separados por barras (“/”) no broker.

Para melhor apreciar a compreensão do método de troca de informações do MQTT, a Figura 1, abaixo, elucida seu modo operante.

Ao lado esquerdo da Figura 1, está o publicador, simbolizado no círculo de cor amarela com um termômetro na imagem, que neste caso é um sensor de temperatura.

O sensor averigua a temperatura do ambiente, que está submetido, e o publica, em tópico, no broker, nomeado em MQTT-Broker.

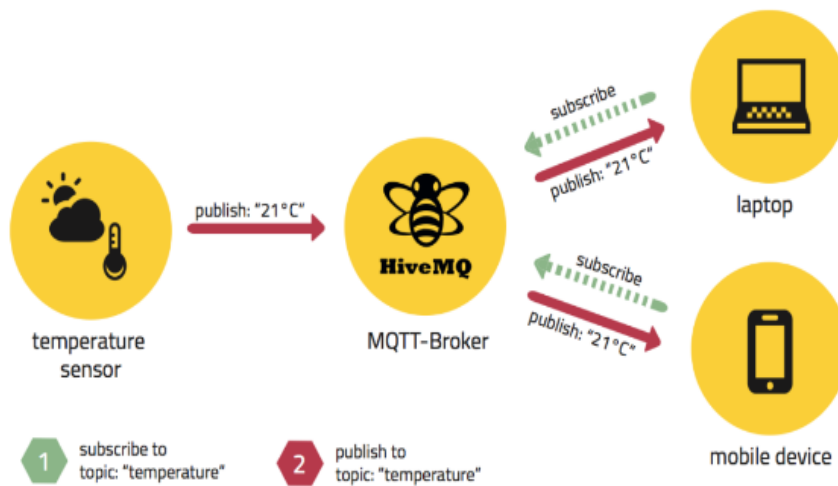


Figura 1: Modelo de Troca de Mensagens MQTT

Em seguida, o broker, após a solicitação dos assinantes, publica a informação do tópico requerido. Na Figura 1, é possível ver os dois assinantes, o laptop e mobile device, que estão verticalmente alinhados na parte direita da figura, dentro do círculo amarelo.

Ademais, para melhor compreensão, é válido apresentar a legenda da Figura 1. O hexágono com o número (1), em cor verde claro, expressa a solicitação do solicitante para obter informação do broker.

Por fim, por outro lado, o hexágono de coloração avermelhada com número (2), representa a publicação em que o publicador ou o próprio broker podem realizar.

3 Questão 08: análise de código Python

Após a análise anterior do método MQTT, agora haverá sua representação em nível de programação. No exemplo abaixo, na Figura 2, um código foi desenvolvido na linguagem de programação Python 3.10. Nele será verificado se foi escrito corretamente e quais suas funcionalidades.

Prontamente, para identificação, Figura 2, o bloco de código do lado direito da imagem representa o solicitante e do lado esquerdo o publicador.

Note que, em ambos os blocos há a conexão com o broker, pois é por intermédio deles que o solicitantes encontra, no tópico desejado, as informações que procura. Sendo assim, no código do bloco do publicador, inicialmente é importada a biblioteca do Paho que utiliza-se para realizar operação do MQTT.

Em seguida, criou-se o método, para inserir no broker, pelo publicador, por intermédio

```
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    # Subscribe in topic_test_A
    client.subscribe("topic_test_A")

# callback function
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

# broker connection
client.connect("193.136.195.56", 1883, 60)
client.loop_forever()
```

```
import paho.mqtt.client as mqtt

client = mqtt.Client()
# broker connection
client.connect("111.111.111.11", 1883, 60)
# publish in topic_test_B
client.publish("topic_test_B", "hello word" )
```

Figura 2: Código Python com uso de MQTT

do método *on-connect*, e que publica a informação. Após, com a função seguinte, *on-message*, ao usá-la é apresentado no console o tópico da informação salva.

Por conseguinte, é instanciado o cliente e as funções *on-connect* e *on-message* são aplicadas. Logo após, é estabelecida a conexão com o broker e o cliente instanciado é mantido em *looping*.

Agora, do lado direito da Figura 2, o bloco de código do assinante, é instanciado o cliente. Após, é realizada a conexão com o broker e então é solicitado acesso ao tópico desejado no broker.

Entretanto, há dois problemas no código, de ambos os blocos de código, da Figura 2. O primeiro problema é que a conexão que o bloco direito faz com o broker é diferente da conexão que o bloco esquerdo faz com o broker.

Isto é, os endereços escritos não referenciam o mesmo broker, logo, se o publicador inserir informações, o assinante não conseguirá ter acesso às informações, pois o assinante está com endereçamento divergente em relação ao broker+ referenciado pelo publicador.

O segundo problema é que, se o primeiro problema não existisse - se o endereçamento do publicador e assinante ao broker fossem o mesmo -, o tópico inserido pelo publicador é diferente do tópico solicitado pelo assinante.

Isso quer dizer que, mesmo que ambos, publicador e assinante, estivessem conectados no mesmo broker, o assinante não conseguiria acessar a informação publicada. Entretanto, estando no mesmo broker, se o assinante procurasse pelo mesmo tópico que o publicado inseriu, teria sucesso em encontrar a informação desejada.

4 Questão 10: Arquitetura de interação em residência

Numa residência, os sensores de temperatura e humidade publicam os dados que recolhem num determinado broker, através de tópicos, conforme a Figura 3, abaixo expressa. O residente através da aplicação consegue subscrever esses mesmos tópicos no mesmo broker e receber os dados. Perante os dados recolhidos, a lâmpada pode ligar ou desligar.

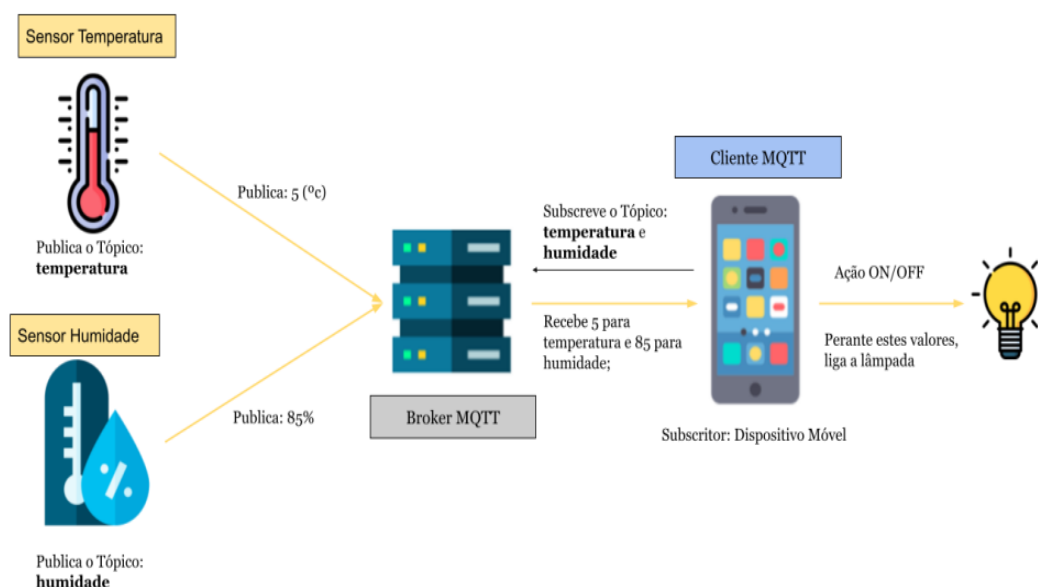


Figura 3: Arquitetura do sistema de interação

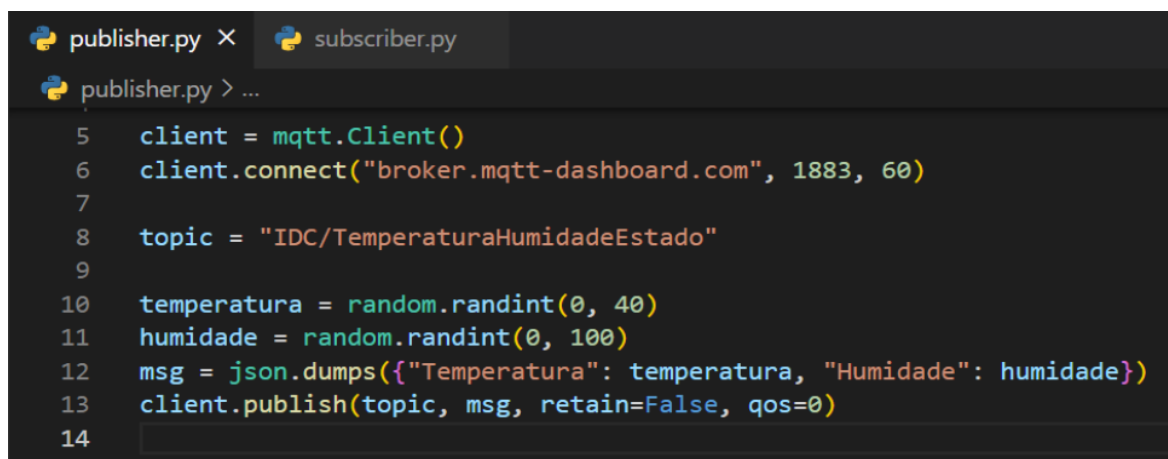
Nele no sistema de interação acima, Figura 3, os sensores de temperatura e humidade publicam, em tópico, no broker, suas informações. Após isso, o Dispositivo Móvel, o assinante, cliente MQTT, solicita no broker as informações em seus devidos tópicos publicados pelos sensores existentes.

Dado isso, ao receber do broker as informações desejadas, o dispositivo móvel toma ação de ligar ou deligar a luz, com base em comandos previamente estabelecidos em seu código fonte.

Diante disso, é possível desenvolver o código do publicador da seguinte maneira, conforme a Figura 4, abaixo elucidada. Nela é possível ver, na linguagem de programação Python 3.10, que o cliente é instanciado, conectado ao broker e adicionado nele o tópico.

Em continuidade, no código da Figura 4, após criado o tópico, o publicador encontra o valor da temperatura e da humidade - que neste caso é randômico por ser um caso-teste - e os publica no broker.

Consoante, na Figura 4 conecta-se ao broker (“broker.mqtt-dashboard.com”, porta: 1883). Os valores de temperatura e humidade são gerados automaticamente, uma forma de simular a leitura de sensores, e são publicados no tópico “IDC/TemperaturaHumidadeEstado”, com a mensagem dos valores em objeto JSON (*JavaScript Object Notation*), Temperatura: x, Humidade: y.



```
publisher.py X subscriber.py
publisher.py > ...
5 client = mqtt.Client()
6 client.connect("broker.mqtt-dashboard.com", 1883, 60)
7
8 topic = "IDC/TemperaturaHumidadeEstado"
9
10 temperatura = random.randint(0, 40)
11 humidade = random.randint(0, 100)
12 msg = json.dumps({"Temperatura": temperatura, "Humidade": humidade})
13 client.publish(topic, msg, retain=False, qos=0)
14
```

Figura 4: Código do publicador em Python 3.10

Por conseguinte, a seguir, na próxima página, há a Figura 5, nela é elucidada o código do assinante, para obter as informações publicadas pelo publicador, do código anterior, da Figura 4.

Semelhantemente ao que foi realizado no tópico 3, acima, da "Questão 08: análise de código Python", no código foi realizado as funções *on-connect* e *on-message*. Respectivamente, insere um tópico e apresenta a mensagem, conforme desenvolvido desenhado em seu bloco de código.

Observe que, a função *on-connect* faz a publicação do tópico com as informações desejadas. Mas, primeiro é preciso ter criado a instância do cliente, logo, é possível usar as funções, tanto *on-connect*, quanto *on-message*, para após realizar a conexão ao broker, conforme é visto da linha 27 à linha 30, da Figura 5.

Diante disso, é válido apontar que é mandatário ao assinante, para ter acesso às informações desejadas do broker, se conectar ao mesmo broker que o publicador, que neste caso seu endereço correto é: (*"broker.mqtt-dashboard.com"*, *porta: 1883*).

Feito isso, o código do assinante subscreve, solicita, o tópico *"IDC/TemperaturaHumidadeEstado"* e aguarda por resposta, as informações existentes. Em seguida, o assinante recebe a mensagem com o objeto em JSON, a qual é decomposta em valores separados de temperatura e humidade para serem combinados.

Diante disso, caso o valor de temperatura seja menor que 10 e o de humidade for maior que 45, a lâmpada acende a luz, caso contrário permanece apagada. Este é um exemplo de automação, utilizando MQTT, para tomada e decisão inteligente.

Observe que tanto no código do publicador, quanto no assinante, na última linha existe uma função que deixa em looping, repetição ininterrupta, para que os valores de publicação e de solicitação deles, repitam continuamente; seu intuito é estar, ao máximo, atualizado.

Prontamente, com os códigos da Figura 4, publicador, e da Figura 5, assinante, ao estarem simultaneamente em execução, é possível ter, como resultado, de suas interações


```
subscriber.py > ...
1  import paho.mqtt.client as mqtt
2  import json
3
4
5  def on_connect(client, userdata, flags, rc):
6      print("Connected with result code " + str(rc))
7      client.subscribe("IDC/TemperaturaHumidadeEstado")
8
9
10 def on_message(client, userdata, msg):
11     global jsonValues, temp, hum
12     # carregar json obj
13     jsonValues = json.loads(msg.payload)
14     # valores de temperatura e humidade
15     temp = int(jsonValues["Temperatura"])
16     hum = int(jsonValues["Humidade"])
17     # print
18     print("Temperatura -> " + str(int(temp))+"\nHumidade -> "+str(int(hum)))
19     # Condicao para ação ON/OFF luz
20     if (temp < 10 and hum > 45):
21         print("LIGHT ON")
22     else:
23         print("LIGHT OFF")
24     print("-----")
25
26
27 client = mqtt.Client()
28 client.on_connect = on_connect
29 client.on_message = on_message
30 client.connect("broker.mqtt-dashboard.com", 1883, 60)
31 client.loop_forever()
```

Figura 5: Código do assinante em Python 3.10

por intermédio do broker, a saída de informações conforme expressa a Figura 6, na próxima página apresentada.

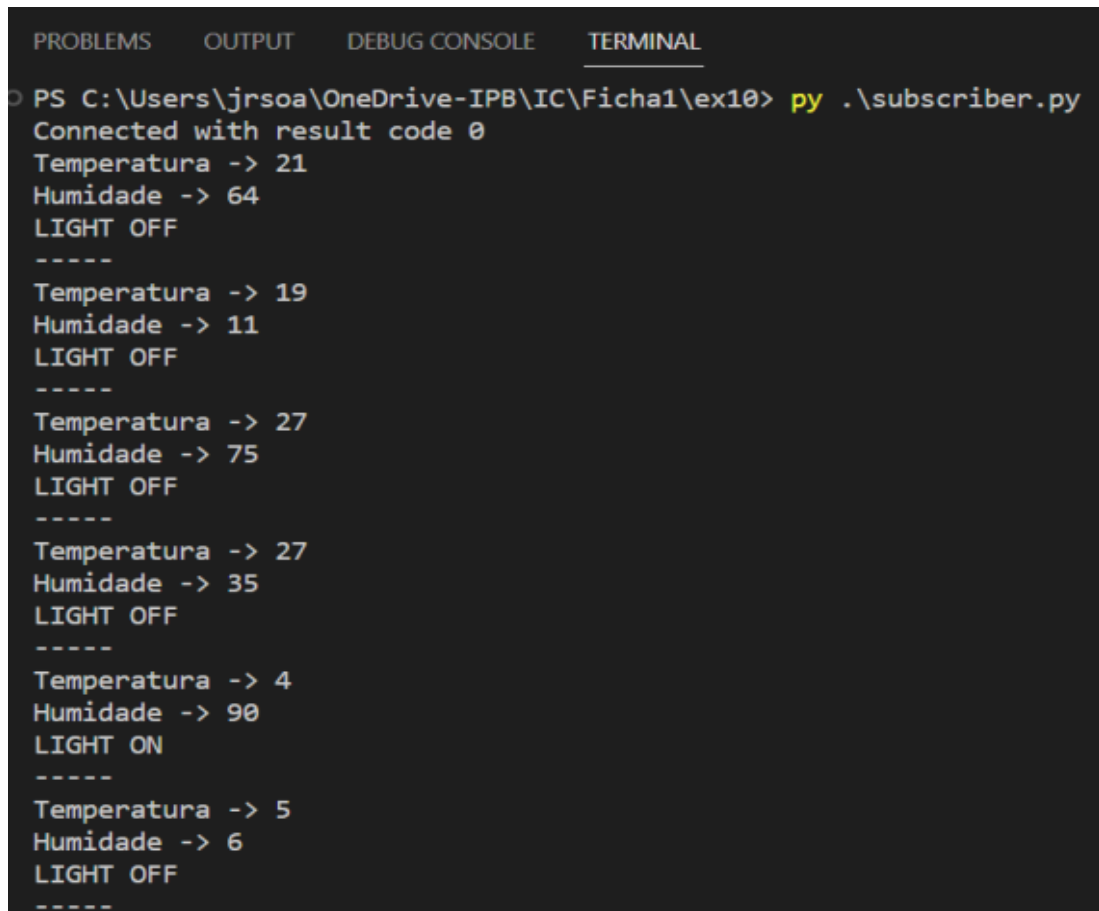
Nela, Figura 6, vê-se que o publicador conseguiu inserir as informações da temperatura e da humidade com sucesso. E a partir dessas informações, a tomada de decisão do estado da lâmpada, ligada ou apagada, é tomado pelo dispositivo móvel.

Também é visível na Figura 6 a repetição contínua dos resultados da temperatura e humidade. Isso só é possível graças à função de looping, repetição ininterrupta, que permite a todo momento inserir os dados da temperatura e humidade e os solicitar.

É devido a este motivo que na Figura 6 há diversos resultados do valor da temperatura e da humidade e qual o estado da luz, a partir de seus valores. Veja que *Light on* e *Light off*, são respectivamente, luz acesa e luz apagada.

É válido apontar que os resultados da Figura 6, são valores distintos, de momentos

singulares de um suposto dia e local de um cenário fictício, pois os dados submetidos ao broker pelo publicador são aleatórios.

A screenshot of a terminal window with a dark background. At the top, there are four tabs: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL', with 'TERMINAL' being the active tab. The terminal shows a command prompt where a Python script has been executed. The output of the script is displayed line by line, showing random data for temperature, humidity, and light status, separated by dashed lines.

```
PS C:\Users\jrsoa\OneDrive-IPB\IC\Ficha1\ex10> py .\subscriber.py
Connected with result code 0
Temperatura -> 21
Humidade -> 64
LIGHT OFF
-----
Temperatura -> 19
Humidade -> 11
LIGHT OFF
-----
Temperatura -> 27
Humidade -> 75
LIGHT OFF
-----
Temperatura -> 27
Humidade -> 35
LIGHT OFF
-----
Temperatura -> 4
Humidade -> 90
LIGHT ON
-----
Temperatura -> 5
Humidade -> 6
LIGHT OFF
-----
```

Figura 6: Resultado da interação do publicador e assinante pelo broker

5 Questão 12: Implementação em MQTT Node-Red e Python 3.10

Dentro do universo da Internet das Coisas - IOT, para sua aplicação no cotidianos há ferramentas e métodos para sua facilitação que vem adquirindo espaço, de maneira significativa, no mercado.

Prontamente, referido aos métodos, dentre eles, o MQTT é o mais utilizado na indústria para a implementação de automações e de tecnologias como a IOT, afirma Souza (Alexandre Sousa, 2021). Consoante, ao olhar para as ferramentas, que inclusive utiliza MQTT, uma das mais difundidas hoje é o Node-Red.

É por esse motivo que, além dos exemplos anteriores, de implementação MQTT, em Python 3.10, a seguir na Figura 7 há implementação, híbrida, além do uso do Python, também encontra-se em Node-Red.

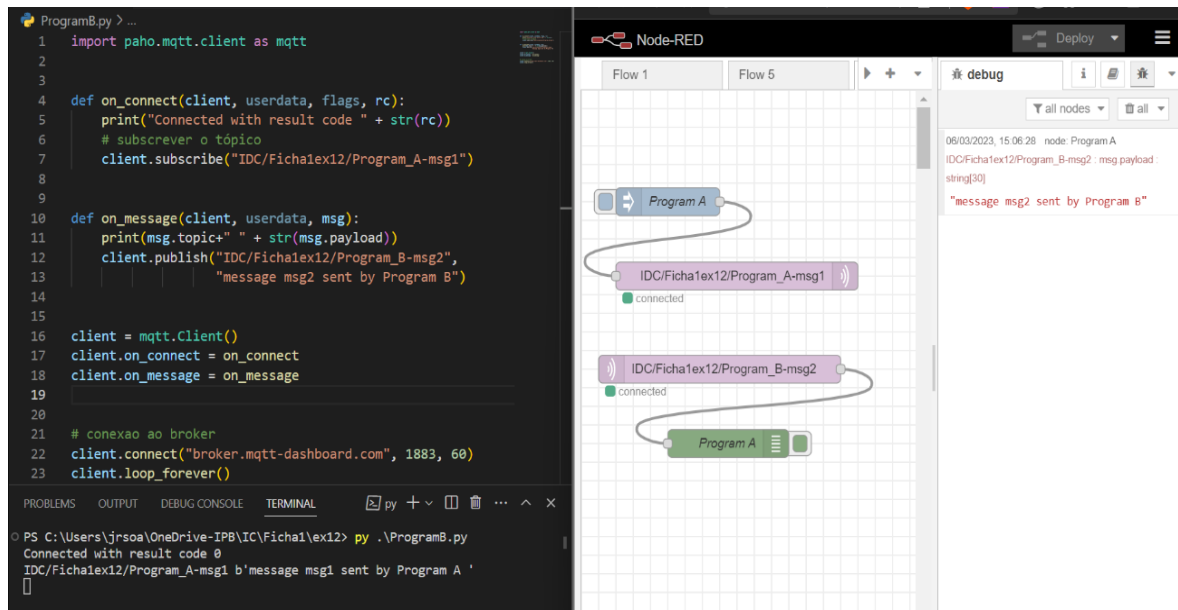


Figura 7: Implementação híbrida em Python e Node-Red com protocolo MQTT

Frente a isso, na Figura 7, percebe-se que a implementação híbrida, implica que parte da implementação foi desenvolvida em Node-Red, parte direita da imagem da Figura 7, e a outra parte foi desenvolvida em Python 3.10, parte esquerda da imagem da figura acima.

Isto posto, vislumbra-se, na Figura 7, ao lado esquerda da imagem, em código, a publicação, em tópico, no broker, de endereço *broker.mqtt-dashboard.com*, a informação pelo publicador. Diante disso, na figura acima, ao lado direito da imagem, em Node-Red, o **Program A**, submete a informação ao Broker, e o assinante obtém a informação.

Note que, tanto em Python, quanto em Node-Red, é possível publicar a informação, em tópico, no broker e ter acesso à essa informação. No lado esquerdo da Figura 7, por ser em código Python, na função *on-connect*, é publicada a informação, e na função *on-message*, o assinante capta a informação publicada.

Em vista disso, é notável enunciar que na Figura 7, do código em Python, na aba *console* está a saída que aparece ao assinante do broker. Assim, da mesma figura, no desenvolvimento em Node-Red, ao lado direito da imagem, em cor avermelhada, está a saída. Isto é, o que é emitido ao assianante do broker, a informação coletada.

Sem demora, é valioso apontar que, ao usufruir de um mesmo broker, válido, existe a possibilidade de acesso instantâneo e simultâneo da informação publicada no tópico escolhido à ele, tanto pelo Node-Red, quanto pelo Python, desde que seus endereços ao broker estejam corretos.

À priori, para a implementação da Figura 7, é válido dizer que há mais 2 prática para serem desempenhadas. Para melhor compreender este cenário, da Figura 7, na página próxima, consta a Figura 8, a qual expressa os 3 casos de interação de publicador, assinante e broker, incluso o caso da Figura acima; desenvolvidos em Python 3.10 em

conjunto, simultâneo, com Node-Red.

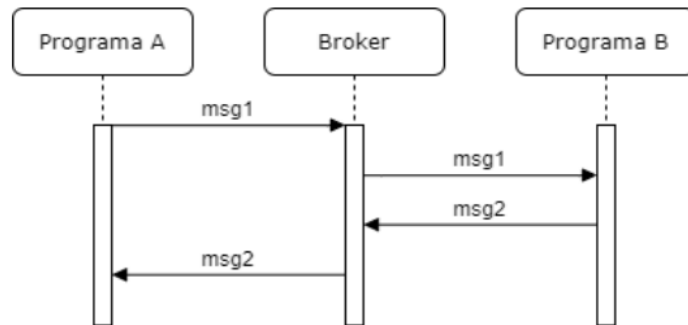


Figura 8: Protótipo 1 de comunicação Program A e Program B pelo Broker

Isto posto, é visível na figura acima, Figura 8, que o *Program A* e *Program B*, comunicam-se, com troca de mensagens, por intermédio do broker. E é com base nessa lógica que a implementação híbrida da Figura 7 foi desenvolvida.

Não distante, a seguir ver-se-á outros dois exemplos com base no protótipo da Figura 8, com uso híbrido de Python 3.10 e Node-Red; a Figura 9, abaixo refere-se ao 2º caso observação, dado o caso primeiro já abordado na Figura 7.

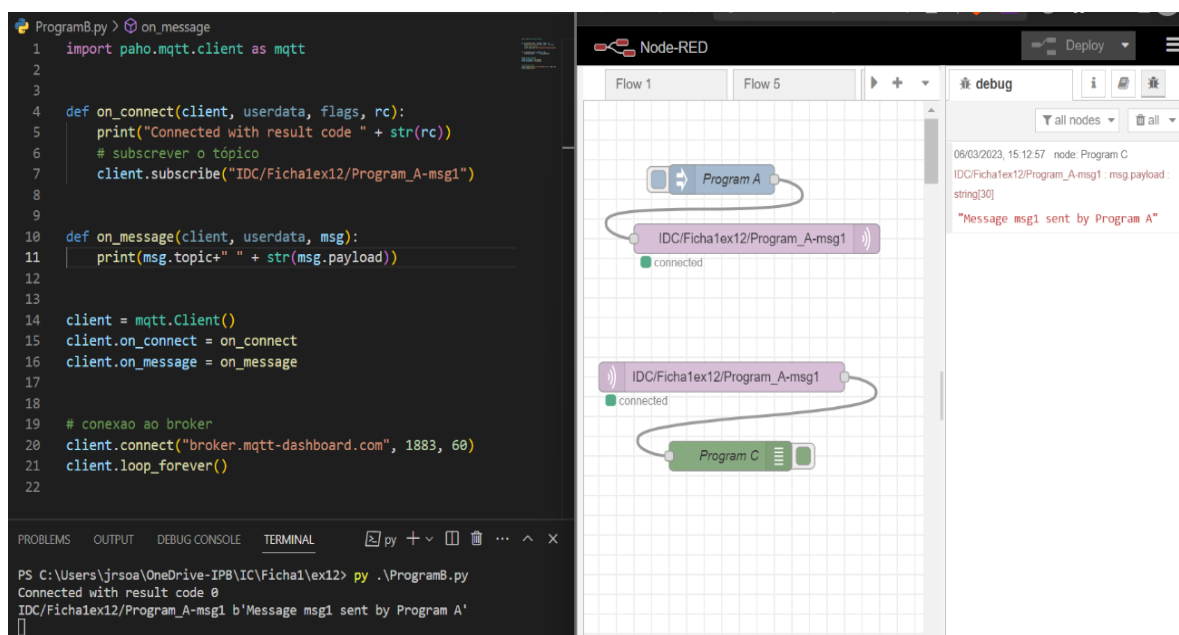


Figura 9: Protótipo 2 de comunicação Program A e Program C pelo Broker

Destarte, semelhante ao caso da Figura 7, ao lado esquerdo da imagem da Figura 9, está o código em Python 3.10 e ao lado direito da mesma imagem, está em Node-Red.

Assim sendo, dito isso, da figura acima, o que se diferencia da 7, é que ambos os tópicos referenciados ao broker são os mesmos, entretanto há, além do *Program A*, o *Program C*. E do lado esquerdo da Figura 9, a função *on-message*, passa a emitir no console o valor de suas operações.

Prontamente, já abordado o caso 2, na Figura 9, resta o último caso a ser tratado. É possível encontrar o caso 3 na Figura 10, abaixo inserido, em caráter visual, para aprimorada interpretação.

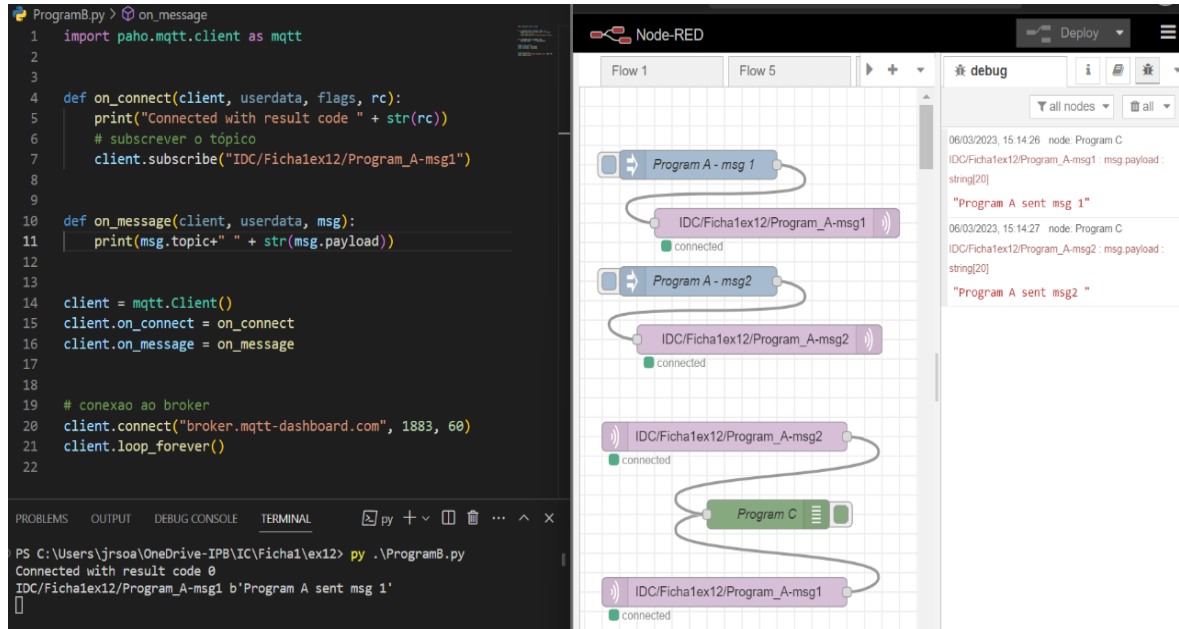


Figura 10: Protótipo 2 de comunicação Program A e Program B pelo Broker

Diante disso, na Figura 10, visualiza-se na parte direita da imagem, a implementação em Node-Red, e na parte esquerda dela, a transição da implementação em Python 3.10; ambas podem se comunicar simultaneamente com o mesmo broker, desde que estejam, publicador e assinante, seus endereços devidamente configurados.

Dessa forma, na figura acima expressada, é visível que no console, lado esquerdo da imagem, há a saída configurada. Semelhante à saída em Python, há a saída em Node-Red, com a mesma informação apresentada; sua lógica simula próximo ao caso 02.

Isso quer dizer que o *Program A* consegue se comunicar, de maneira livre e de rápido acesso, ao broker, com envio de informações e coleta, também, de informações. E isso caracteriza a facilidade de uso e eficiência do MQTT, o que valida seu alto uso no mercado global.

Ademais, vê-se que o MQTT, além de uma ferramenta de fácil acesso e publicação, às informações ali depositadas, no Broker, é provido de inúmeras plataformas, como o Node-Red, e linguagens que o torna possível de executar, com visão especial para IOT.

Por fim, conclui-se que, além do MQTT ser passível de diversos meios de uso em protocolo de comunicação, pode ser aplicado o usufruto de diálogo híbrido, ferramentas, desde que os endereçamentos do broker e tópico estejam corretos.

6 Questão 13 - (a) Refatoração do exercício 10 em Node-RED

Na Escola de Ensino Superior de Tecnologia e Gestão (ESTIG) do Instituto Politécnico de Bragança, Portugal, há um laboratório virtual da disciplina de IoT, nele contém sensores que publicam dados digitalizados através de dispositivos ESP8266/32, dentre eles, a exemplo, sensor de humidade.

Tendo isso em vista, para captar esses dados publicados por estes sensores, a ferramenta mais indicada para este papel é o Node-Red. Posto isso, é importante ressaltar que o Node-RED é uma plataforma de desenvolvimento visual para a criação de fluxos de trabalho em IoT.

Nele, é possível fazer um fluxo típico, o qual pode ser composto por nós (nodes) interligados, cada um com uma função específica. A questão de exemplo, a seguir, abaixo, na Figura 11, existe um caso particular, como é possível ver, um fluxo de publicação e outro de subscrição.

Sendo assim, na Figura 11, vislumbra-se que o bloco azul, nomeado em *inject*, aciona o ação de publicação no broker da informação desejada; a qual será armazenada em tópico especificado.

Em seguida, no bloco alaranjado, a informação é tratada por um função desejada, e por seguida, é transmitida para o bloco roxeado, o qual efetiva a publicação no broker almejado.



Figura 11: Fluxo de publicação

Por conseguinte, na Figura 11, nota-se ainda que o fluxo de publicação contém um nó Inject, que é responsável por injetar objetos JSON no fluxo. Para melhor compreensão deste fluxo, na próxima página, está a Figura 12, a qual expressa a tela de configuração do bloco *inject*, de color azul escuro na Figura 11.

Esse nó está configurado para injetar um objeto JSON contendo as propriedades "Temperatura" e "Humidade" com valores iniciais de zero a cada 30 segundos.

E sendo assim, esse objeto é publicado para o tópico `/IDC/TemperaturaHumidade`, no broker especificado no bloco roxeado, da imagem acima, a Figura 11. Lembre-se de que o tópico, no broker, deve ser corretamente referenciado, para que assim seja possível receber as informações corretas de acordo com o interesse estabelecido.

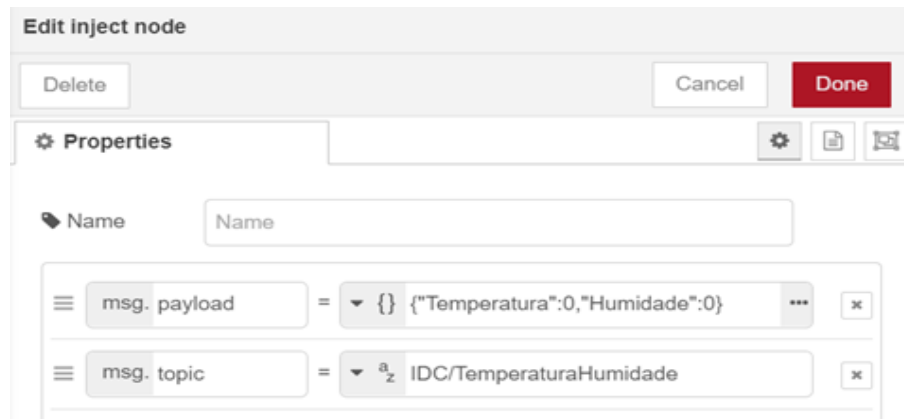


Figura 12: Edição do nó de injeção

Feito isso, o passo seguinte é editar o nó do bloco de função, de cor alaranjada na Figura 11. Nele, conforme elucida a Figura 13, abaixo, os valores de temperatura e humidade são gerados de forma aleatória para substituir os valores iniciais de zero.

Dessa forma, o uso de inteiros aleatórios no nó de função garante que as informações sejam variáveis e mais próximas da realidade.

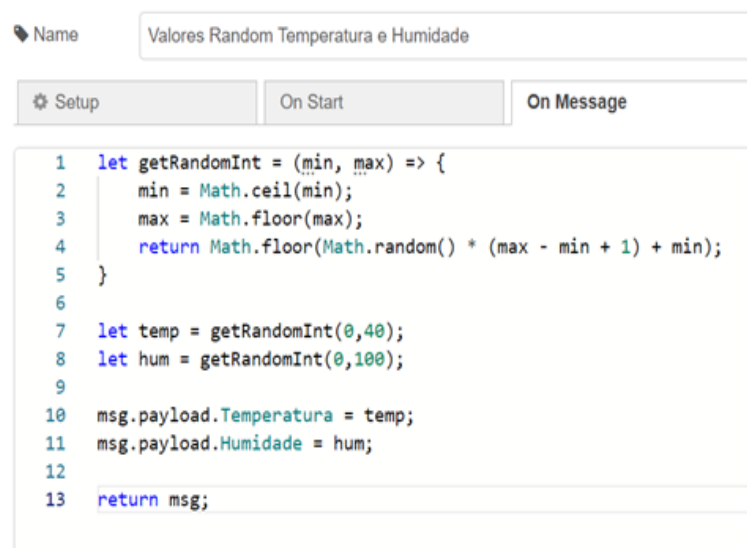


Figura 13: Edição do nó de função do publicador

Prontamente, para o término do fluxo de publicação da Figura 11, há o bloco de cor roxeado, o qual é o nó MQTT-out, responsável por publicar o objeto JSON atualizado para o tópico `/IDC/TemperaturaHumidade`. Esse tópico é então usado pelo fluxo de subscrição para receber e processar as informações.

Dito isso, torna-se possível demonstrar o outro lado da comunicação MQTT, que além do publicador, há o subscritor, também conhecido como assinante, os quais são intermediados pelo broker.

Para apreciar melhor esta lógica, a seguir, na próxima página há a Figura 14, a qual

expressa o fluxo de subscrição. Isto é, o fluxo de processo de recebimento de informação do assinante, pelo broker, o qual recebe a informação do publicador.



Figura 14: Fluxo de subscrição

Conforme vê-se na Figura 14, o fluxo de subscrição começa com a utilização do nó `mqtt-in`, de coloração roxa, nomeado em *IDC/TemperaturaHumidade*, que é responsável por receber os dados enviados pelo publicador. Posto isso, este nó é conectado ao mesmo broker que o publicador, a fim de garantir a comunicação entre eles.

À vista disso, é válido lembrar que o broker é um intermediário responsável por receber e encaminhar os dados entre os nós de subscrição e publicação. Quando os dados são recebidos, de 30 em 30 segundos pelo publicador, são enviados para o próximo nó, que é o nó de função.

Em seguida, o próximo procedimento da Figura 14, é o sinal sair do bloco do MQTT-IN para o bloco de função, de cor alaranjada, na figura acima. E para melhor explanação, consta a Figura 15, a qual expressa as configurações de edição do nó de função.



Figura 15: Edição do nó de função do subscritor

Isto posto, é mister informar que o nó de função é responsável por processar os dados recebidos e realizar algumas transformações neles.

Dessa forma, no caso em questão, da Figura 15, o objeto JSON recebido é decomposto em variáveis para que seus valores possam ser combinados. Consoante, sabe-se que o JSON, é uma forma de armazenamento e transmissão de dados que é amplamente utilizada em sistemas de IoT e outras aplicações.

Por conseguinte, ainda na Figura 15, é utilizada uma estrutura de condição *if* para determinar um novo atributo *Light* a ser criado no objeto JSON.

Nela, a condição *if* verifica se a temperatura e a humidade atendem a um determinado critério e, se isso for verdadeiro, o novo atributo *Light* é criado e atribuído ao valor *ON*, ativado. Caso contrário, o valor do atributo é definido como *OFF*, inativo.

E é por meio disso que o objeto JSON modificado é enviado para o próximo nó do fluxo, *debug*, de cor esverdeado na Figura 14, que é um nó de saída responsável por exibir os dados. Ademais, é por meio do *debug* que se encontra os resultados esperados conforme vê-se na Figura 16, abaixo exposta.

```
07/03/2023, 23:28:16 node: output
IDC/TemperaturaHumidade : msg.payload : Object
  ▶ { Temperatura: 13, Humidade: 81, Light: "OFF" }

07/03/2023, 23:28:16 node: output
IDC/TemperaturaHumidade : msg.payload : Object
  ▼ object
    Temperatura: 7
    Humidade: 28
    Light: "OFF"

07/03/2023, 23:28:17 node: output
IDC/TemperaturaHumidade : msg.payload : Object
  ▼ object
    Temperatura: 4
    Humidade: 72
    Light: "ON"

07/03/2023, 23:28:17 node: output
IDC/TemperaturaHumidade : msg.payload : Object
  ▶ { Temperatura: 35, Humidade: 85, Light: "OFF" }

07/03/2023, 23:28:19 node: output
IDC/TemperaturaHumidade : msg.payload : Object
  ▶ { Temperatura: 0, Humidade: 29, Light: "OFF" }
```

Figura 16: Resultado do fluxo de processo do subscritor

Por fim, na Figura 16 nota-se que os valores dos sensores do laboratório virtual do IPB foram adquiridos, tanto da qualidade do ar, a humidade, da luminosidade (identificado na Figura 16 como *Light*), quanto da temperatura.

7 Questão 13 - (b) Captação de dados do IPB pelo Node-Red

Inicialmente, é bom ter conhecimento de que no laboratório virtual da disciplina, existem dispositivos ESP8266/32, que são placas de desenvolvimento para Internet das Coisas, baseadas no microcontrolador ESP32/ESP8266.

E é por meio destes dispositivos que se coletam dados de diferentes parâmetros, como temperatura, humidade, pressão e outros, por meio de sensores integrados ou externos e os publicam em tópicos em um broker MQTT.

Diante disso, os dispositivos ESP8266/32 publicam dados em três tópicos MQTT no broker. E esses tópicos podem ser subscritos usando o nó MQTT-IN do Node-RED, que recebe as mensagens publicadas pelos dispositivos e as envia para outros nós do fluxo para a visualização.

Assim sendo, com visão no Node-RED, o seu esquema inclui três fluxos, cada um representando um tópico MQTT subscrito e a saída de dados pelo nó de debug. O nó debug exibe as mensagens recebidas em cada tópico, permitindo que o usuário visualize os dados coletados em tempo real.

A título de explicação, abaixo, consta-se a Figura 17, a qual elucida a existência de três publicadores (Luminosidade, Temperatura e Humidade), respectivamente, de cima para baixo, na imagem; observa-se que os publicadores são sensores.

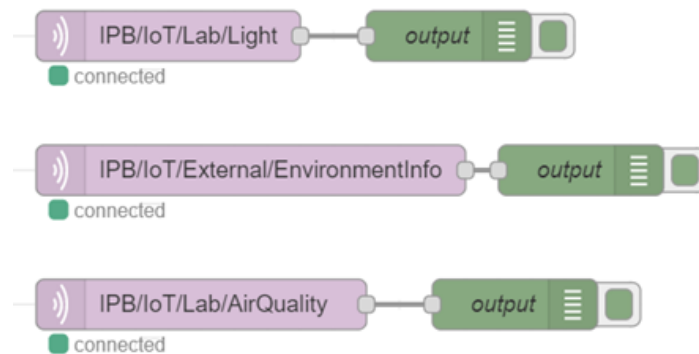


Figura 17: Sensores do laboratório virtual em publicadores no Node-RED

Frente à isso, no exemplo da Figura 17, encontra-se no tópico *IPB/IoT/Lab/AirQuality* apresenta uma humidade de 51.1 por cento. Já no tópico *IPB/IoT/External/EnvironmentInfo* encontra-se uma temperatura de 23.48 graus celsius. Por fim, no tópico *IPB/IoT/Light* captou-se uma luminosidade de, aproximadamente, 104.27.

Isso implica que o dispositivo ESP8266/32 responsável por coletar esses dados detetou que a temperatura, a humidade e a luminosidade do ambiente que está instalado são, respectivamente, 23.48 °C, 51.1 por cento e 104.27, no momento da captação dos dados.

Diante disso, com o bem executar do bloco de código, devidamente configurados, na Figura 17, é possível ter os seguintes resultados, apresentados na Figura 18, os quais são os resultados encontrados, no momento da leitura.



```
08/03/2023, 19:53:25 node: output
IPB/IoT/Lab/Light : msg.payload : Object
▼object
  light_min: 104.17
  light_avg: 104.17
  light_max: 104.17

08/03/2023, 19:53:25 node: output
IPB/IoT/Lab/AirQuality : msg.payload : Object
▼object
  r_temp: 23.54
  temp: 23.48
  r_hum: 50.9
  hum: 51.1
  press: 93202
  gas_res: 1229282
  iaq: 151.75
  iaq_accur: 3
  s_iaq: 172.36
  co2_eqv: 1723.56
  voc_eqv: 4.64

08/03/2023, 19:53:25 node: output
IPB/IoT/External/EnvironmentInfo : msg.payload : Object
▼object
  temp: 26.66
  hum: 49.06
```

Figura 18: Resposta de saída em Node-RED dos sensores do laboratório virtual

Conforme vê-se na Figura 18, no tópico *IPB/IoT/Lab/AirQuality*, a mensagem contendo o objeto é recebida pelo nó MQTT do Node-RED e encaminhada para o nó de função.

Em continuidade, é válido lembrar que é no nó de função em que se configura para extrair os valores de temperatura e humidade do objeto e criar uma nova mensagem contendo apenas esses valores. Sendo assim, essa nova mensagem é então enviada para o *node-red-dashboard*.

Frente à isso, é mister apresentar que o *node-red-dashboard* é uma interface gráfica baseada na web que permite visualizar e interagir com os fluxos de dados do node-red. Dito isso, em continuidade do exemplo acima, da Figura 18, a nova mensagem contendo os valores de temperatura e humidade é recebida pelo nó gauge e chart do Node-RED.

Prontamente, é inteirar que nó gauge é um tipo de widget do *node-red-dashboard*, um

medidor que indica a posição da temperatura, em graus celsius, e é atualizado em tempo real à medida que novas mensagens são recebidas.

Consoante, a título de conhecimento, o nó *chart* é um tipo de widget do *node-red-dashboard* que mostra um gráfico de linha com os valores de humidade ao longo do tempo. É nele que se configura, para exibição, os valores de humidade em porcentagem, sendo também atualizado em tempo real à medida que novas mensagens são recebidas.

A seguir, para a compreensão de um fluxo mais complexo, realizado no Node-RED, está a Figura 19, a qual apresenta o tratamento dos valores recebidos do laboratório virtual e elucida os valores em tempo real de captação dos dados.

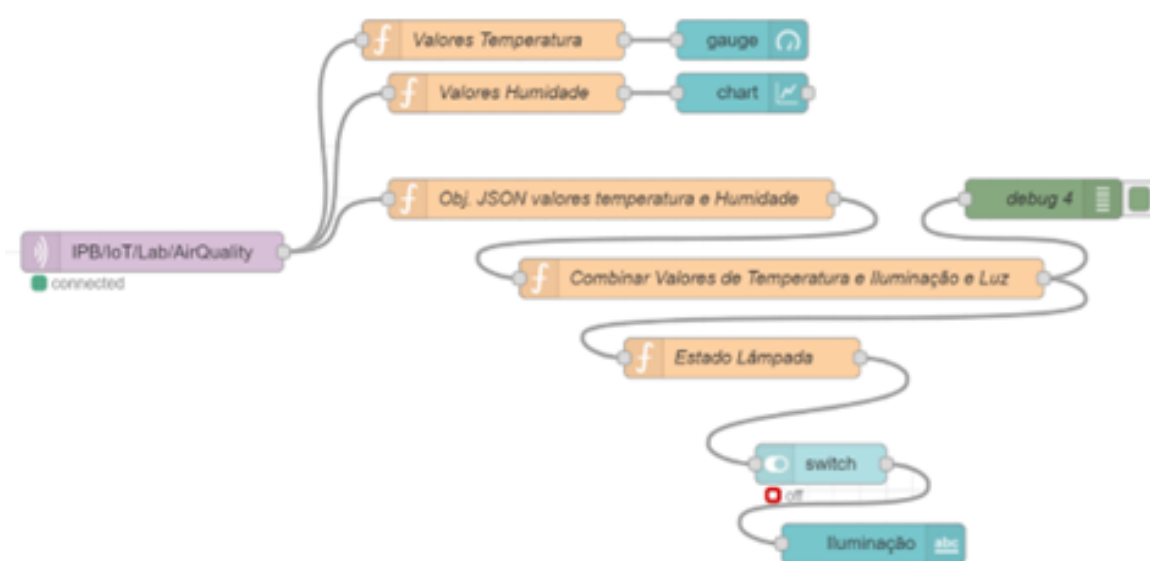


Figura 19: Fluxo avançado em Node-RED dos sensores do laboratório virtual no tópico *IPB/Lab/AirQuality*

Dessa forma, com base na Figura 19, é possível dizer que segunda parte do fluxo, a intenção é interagir com o estado da iluminação com base em uma combinação dos valores de temperatura e umidade, utilizando um conjunto de funções para processar os dados.

Em função disso, ainda na Figura 19, o conjunto de funções é responsável por processar os dados de temperatura e humidade recebidos, combinar esses valores e gerar um atributo para a mensagem com o estado da iluminação.

Enquanto isso, ainda na figura cima, tem o nó *switch*, o qual é um componente gráfico do Node-RED Dashboard que permite definir uma ação com base em um valor de entrada. Ele permite a sua configuração para enviar um comando de ligar ou desligar a iluminação com base em uma condição ou atributo da mensagem específico.

Desse modo, a exemplo, o conjunto de funções pode ser configurado para enviar um atributo com a mensagem para ligar a iluminação se a temperatura estiver abaixo de um determinado valor e a humidade acima de outro valor.

Dado isso, é importante saber que, na Figura 19, apresenta-se o bloco de construção da lógica de tratamento dos dados, em Node-RED, enquanto a Figura 20, exposta na próxima página, estão os resultados da execução do bloco de código da Figura 19.

```
07/03/2023, 23:40:21 node: debug 4
IPB/IoT/Lab/AirQuality : msg.payload : Object
  ▶ { Temperatura: 22.6, Humidade: 41.07,
    Light: "OFF" }

07/03/2023, 23:41:06 node: debug 4
IPB/IoT/Lab/AirQuality : msg.payload : Object
  ▶ { Temperatura: 22.59, Humidade: 41.09,
    Light: "OFF" }

07/03/2023, 23:41:54 node: debug 4
IPB/IoT/Lab/AirQuality : msg.payload : Object
  ▶ { Temperatura: 22.57, Humidade: 41.14,
    Light: "OFF" }
```

Figura 20: Resultado do tratamento de dados da Figura 19

Em vista disso, torna-se relevante afirmar que, a segunda parte do fluxo realiza o uso do Node-RED Dashboard e um conjunto de funções para processar os dados e interagir com o nó switch, com o objetivo de alterar o estado da iluminação com base em valores específicos de temperatura e humidade; o que permite obter o resultado do Node-Red Dashboard, conforme a Figura 21, abaixo apresenta.

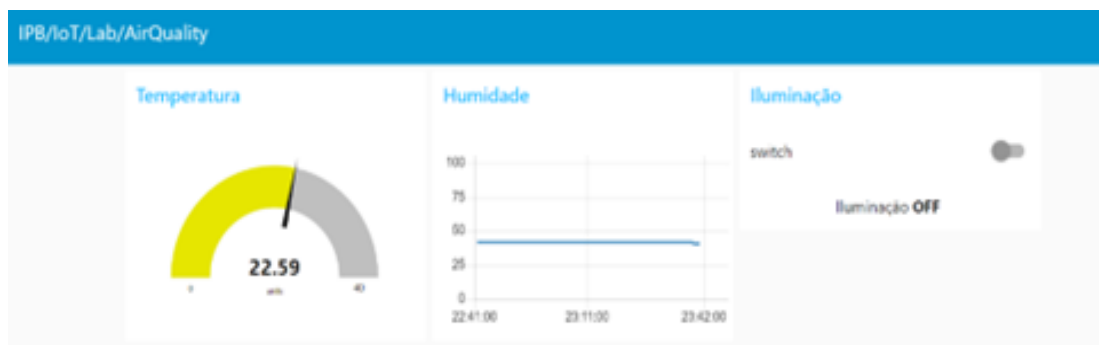


Figura 21: Resultado do node-red dashboard

Por conseguinte, no tópico *IPB/IoT/External/EnvironmentInfo*, a configuração do fluxo é semelhante ao descrito anteriormente, na Figura 19. A mensagem recebida contém um objeto com valores de temperatura e humidade que precisam ser processados e exibidos no node-red-dashboard.

À posteriori, após analisado as configurações e os resultados dos valores emitidos pelo tópico *IPB/Lab/AirQuality*, presente na Figura 19, segue-se para a análise e configuração destinado, agora, ao tópico *IPB/IoT/External/EnvironmentInfo*, elucidado, abaixo, na Figura 22.

Vale ressaltar que, na figura abaixo, o fluxo é dividido em duas partes. Na primeira,

os dados são processados usando nós de função e enviados para o node-red-dashboard usando os nós gauge e chart, que exibem, respectivamente, a temperatura e a humidade.

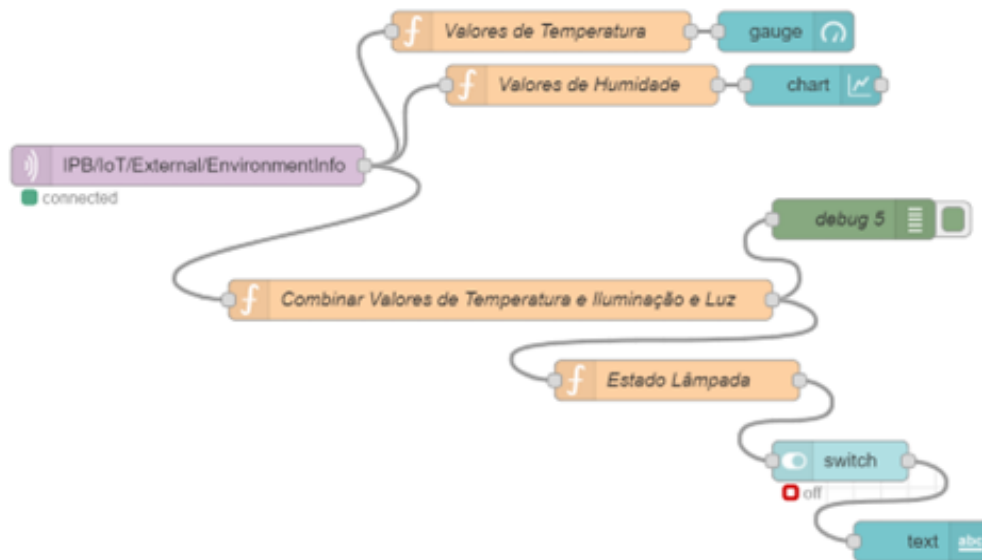


Figura 22: Fluxo em Node-RED dos sensores do laboratório virtual no tópico *IPB/IoT/External/EnvironmentInfo*

Já na segunda, no fluxo, da Figura 22, os valores de temperatura e humidade são combinados usando um nó de função e passados para o nó switch do *node-red-dashboard*. Diante disso, se a temperatura for inferior a 15 °C e a humidade superior a 45 %, a iluminação é ativada; caso contrário, a iluminação é desligada.

À vista disso, a seguir, na Figura 23, está o exemplo em que, como a temperatura está maior que 15 °C e a humidade está menor que 45 %, a luz encontra-se apagada. Veja que, a temperatura está identificada na Figura 23, como *temp*, a humidade como *hum*, a da luz como *light* e o estado da luz em *off*, o mesmo que desligada.

```
09/03/2023, 02:00:55 node: debug 5
IPB/IoT/External/EnvironmentInfo : msg.payload : Object
▼object
  temp: 25.82
  hum: 43.79
  light: "OFF"
```

Figura 23: Resultado do tratamento de dados da Figura 22 no Node-RED Dashboard

Em seguida, a única diferença em relação ao fluxo anterior, da Figura 19 é que a mensagem recebida contém apenas dois atributos de temperatura e humidade, o que

significa que não é necessário filtrar os atributos da mensagem usando nós de função adicionais.

Sendo assim, com base nos valores recebidos pelo assinante, o mesmo que subscritor, do bloco de código da Figura 22, foram tratados pelo código e com base nestes tratamentos, foram transformados em meios gráficos, pelo Node-RED, é possível ver, na Figura 24, abaixo apresentada.



Figura 24: Resultado do node-red dashboard no tópico *IPB/IoT/External/EnvironmentInfo*

Por fim, é visível na Figura 24, que a temperatura está no modelo gauge, a humidade no modelo chart e a iluminação, está em switch. Dessa forma, ademais, respectivamente, encontra-se que os valores da temperatura, humidade e iluminação, estão próximos a, respectivamente: 25.82 °C, 43.75 % e apagada.

Referências

Alexandre Sousa. Mosquitto – entenda suas funcionalidades e vantagens. Disponível em: <https://www.linkedin.com/pulse/mosquitto-entenda-suas-funcionalidades-e-vantagens-alexandre-sousa>. Acesso em: 07 Abr. 2023, 2021.

Eclipse Foundation. Eclipse paho. Disponível em: <https://www.eclipse.org/paho/>. Acesso em: 04 Abr. 2023, 2023.

The Things Stack. Eclipse paho. Disponível em: <https://www.thethingsindustries.com/docs/integrations/mqtt/mqtt-clients/eclipse-paho/>. Acesso em: 04 Abr. 2023, 2022.