

# TP 546 Internet das coisas e redes veiculares

Prof. Samuel Baraldi Mafra  
[samuelbmafra@inatel.br](mailto:samuelbmafra@inatel.br)

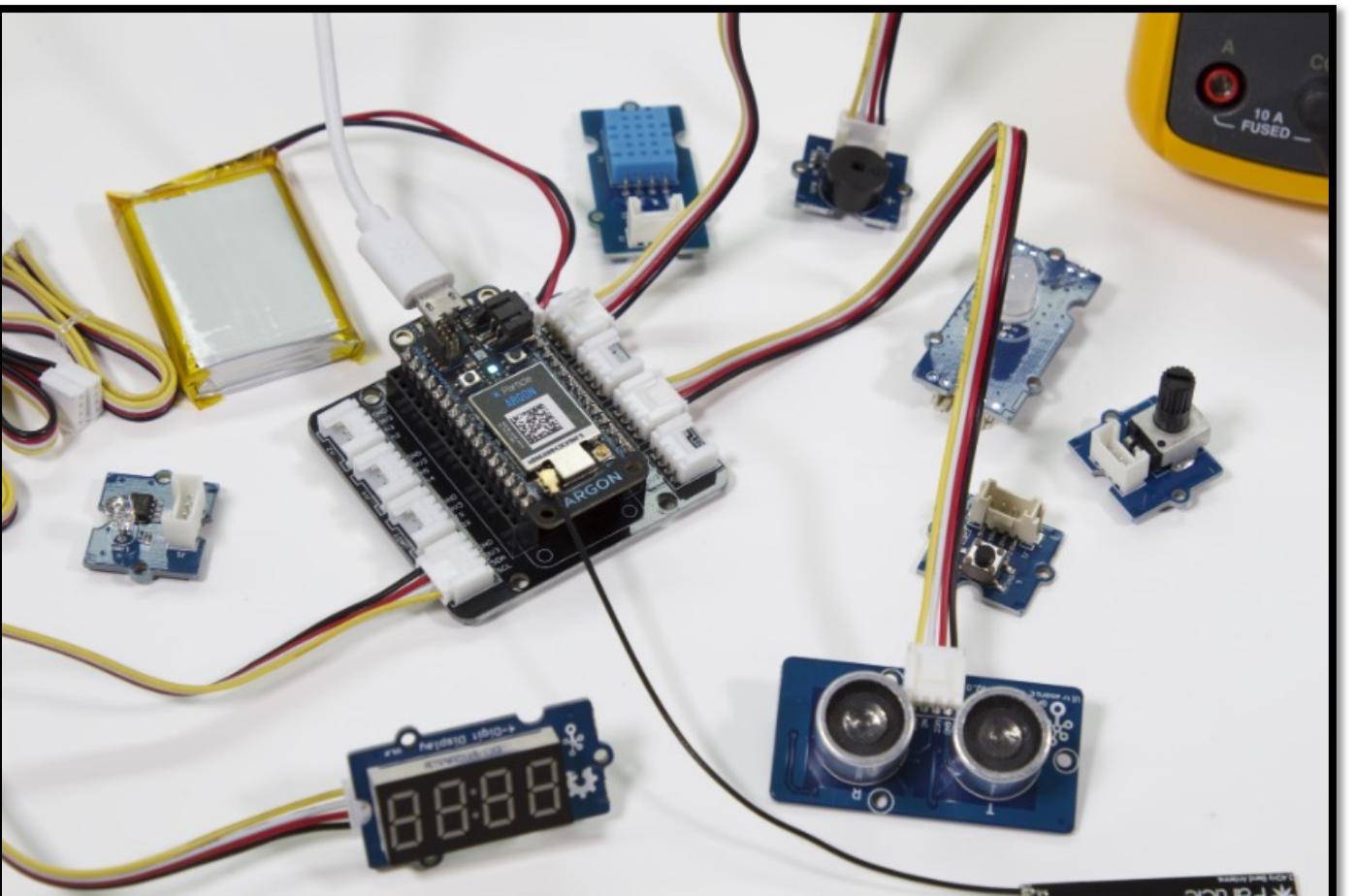


CAMINHOS  
QUE CONECTAM  
COM O FUTURO

# Introdução à prototipagem com o ESP32

# Objetivos da concepção de protótipos

- Um protótipo é um modelo experimental de uma solução proposta.
- Representação de um produto em potencial antes de seu *design* final.
- Permite simular o produto futuro através de iterações.



# Objetivos da concepção de protótipos

- Agilizar o desenvolvimento de *hardware* e *software*.
- Redução de custos de engenharia e materiais para desenvolvimento.
- Obter *feedback* de clientes e avaliação do modelo de negócios.
- Alterar prioridades e tecnologia, de acordo com resultados obtidos.
- Visualizar problemas de implementação nas etapas iniciais do projeto.

# Etapas do processo de prototipagem



# Prova de conceito, protótipo e produto piloto

## Prova de conceito (*POC - Proof of Concept*)

- Foca em avaliar a viabilidade de um produto, serviço ou caso de uso do produto.
- Geralmente, foca em um aspecto específico à ser validado (Ex.: cobertura, consumo, etc.).

## Protótipo

- Mais elaborado do que uma prova de conceito, geralmente com mais de uma funcionalidade.
- Tende a ser uma representação inicial de um produto final desejado.

## Produto Piloto

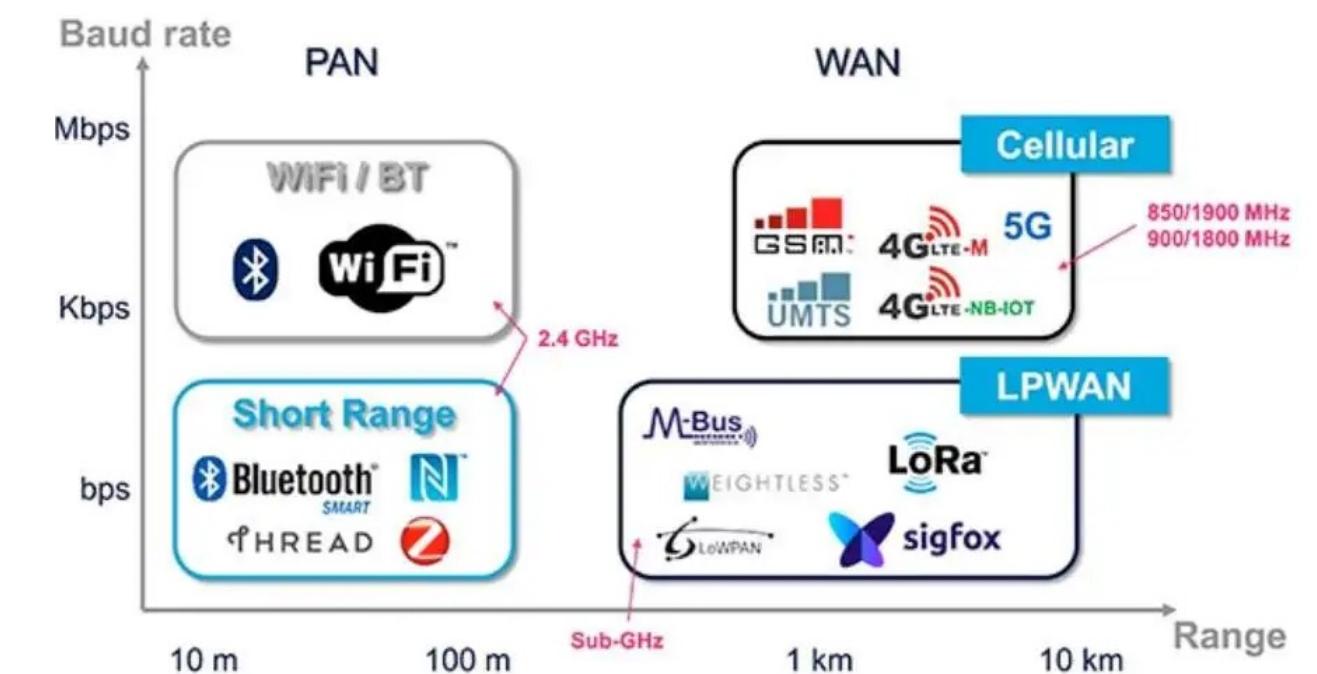
- Produto têm forma similar à final, mas ainda em fase de desenvolvimento.
- Testado pelo cliente em potencial, geralmente em campo – *feedback* e avaliação.
- Geralmente produzido em lotes de pequena escala, antes da produção final.

# Considerações: volume de dados

- É possível estimar o volume de dados requerido pela aplicação?
- Qual o período de envio de tais informações ao servidor?
- Que tipo de *hardware* / processador atende à este volume?
- Os dados do dispositivo serão adquiridos e processados em borda?  
(i. e., próximos e/ou no mesmo local onde são gerados)

# Considerações: conectividade

- De que forma o dispositivo será conectado à rede?  
(Ex.: WiFi, rede celular, Ethernet, LoRa, etc.)
- Qual a infraestrutura necessária desta interface?  
(Ex.: Gateways / torres próprias ou infraestrutura terceirizada)
- Qual a cobertura oferecida pela interface?



# Considerações: desenvolvimento do hardware

- Que tipo de sensores e/ou atuadores irão compor a solução?
- Qual o preço-teto dos componentes e PCB do projeto?
- Há estoque / produção de todos os componentes do dispositivo?
- Quais protocolos serão utilizados nas interfaces entre dispositivos?  
(Ex.: UART, SPI, I<sup>2</sup>C, CAN, GPIO)

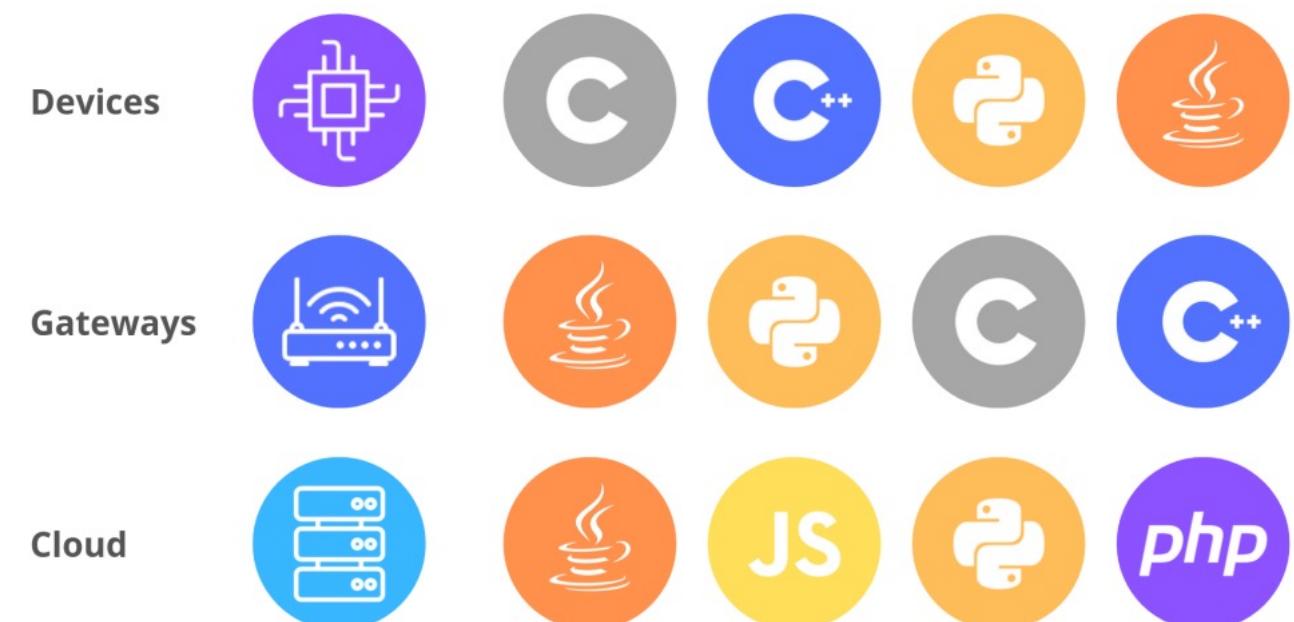
# Considerações: desenvolvimento do software

- Quais serão as linguagens de programação utilizadas para desenvolvimento?

(Ex.: C, C++, Python, Java, JavaScript, PHP, etc.)

- Quais bibliotecas & *frameworks* existem para a solução?

- Existem alternativas de licença livre para desenvolvimento?



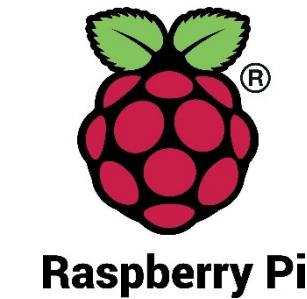
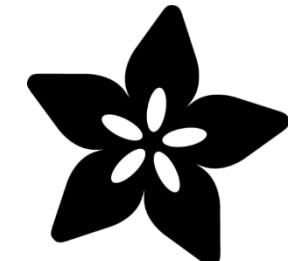
Source: Eclipse Foundation

# Considerações: segurança dos dados

- Quais será o tipo de criptografia de dados utilizada?  
(Ex.: TLS, SSL, AES)
- Haverá necessidade de atualização de certificados / chaves de acesso?
- Quem poderá acessar tais dados?

# Alguns exemplos de plataforma de hardware

- Plataformas de *hardware* padronizado permitem acesso rápido às principais funcionalidades de um processador.
- Podem conter componentes periféricos integrados (Ex.: memórias, sensores, circuitos de alimentação, etc.).
- Geralmente, são o ponto de partida para desenvolvimento de um produto, antes de desenvolver uma solução de *hardware* própria.



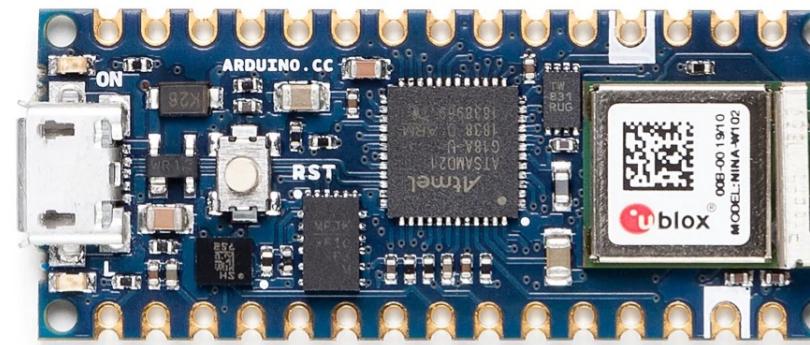


# Arduino

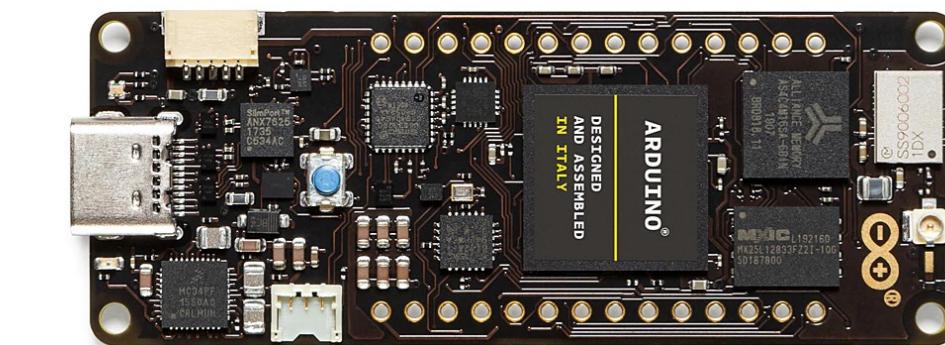
- Empresa italiana onipresente na indústria de produtos de eletrônica e prototipagem.
- Constituída por um conjunto de placas de desenvolvimento, IDE e *framework* de software.
- Originalmente utilizava placas com microcontrolador Atmel (AVR), mas expandiu para outros semicondutores.
- Diversas linhas de produtos – Classic, Nano, PRO, dentre outras.



Uno Rev. 3 (Classic)



Nano 33 IoT (Nano)

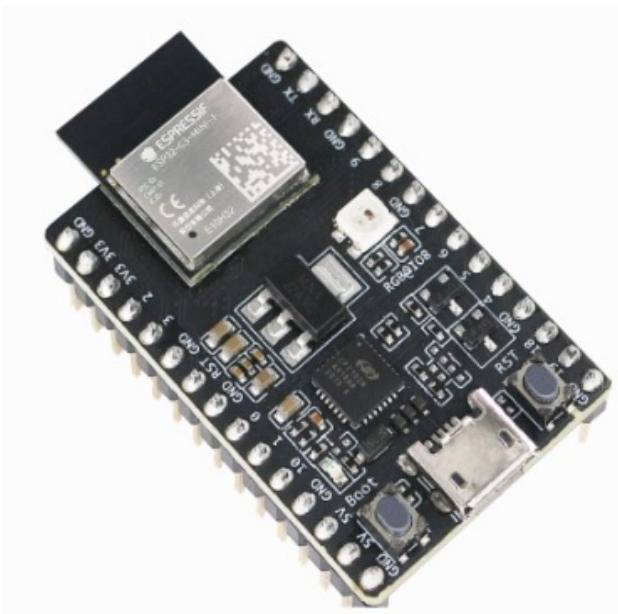


Portenta H7 (PRO)



# Espressif

- Multinacional chinesa, especializada no desenvolvimento de soluções WiFi e Bluetooth de baixa potência.
- Microcontroladores podem utilizar uma SDK mantida pela própria empresa (ESP-IDF) compatível entre suas famílias.
- Fabricante da família de microcontroladores ESP8266, ESP32 e de seus variantes.



ESP32-C3-DevKitM-1



ESP32 Devkit V1  
(ESP-WROOM-32)

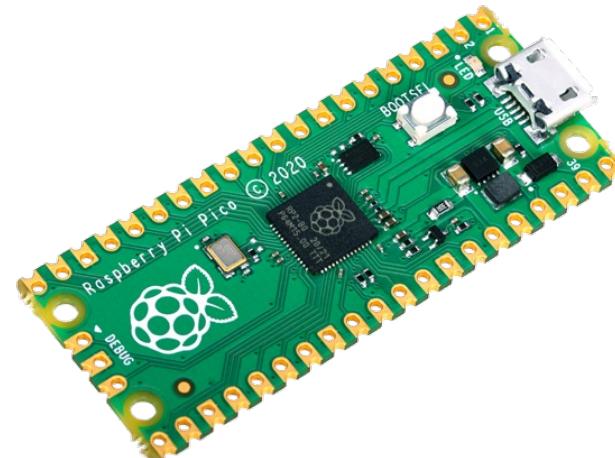


Node MCU V3  
(ESP8266)

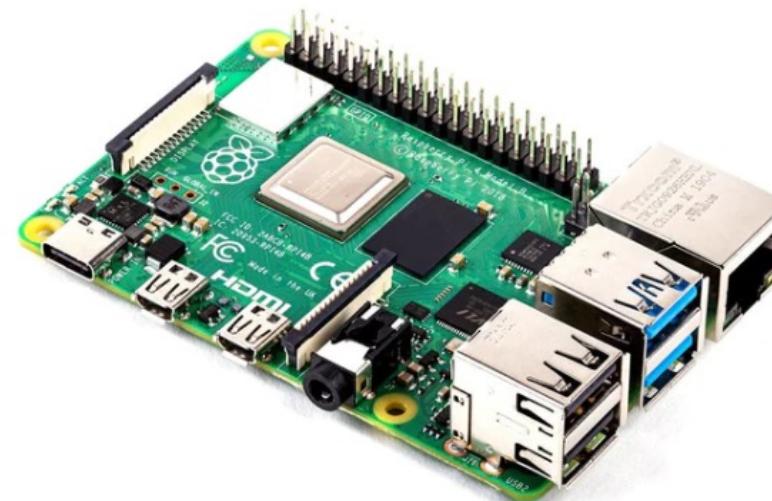


# Raspberry Pi Foundation

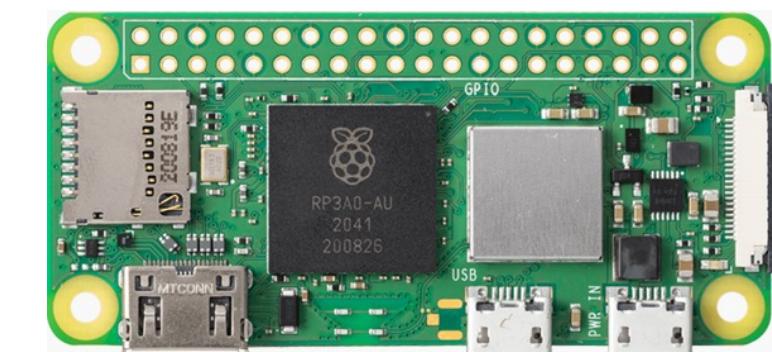
- Organização sem fins lucrativos fundada em 2009 para estimular o estudo básico de ciência da computação.
- SBC's (Single-board computers) geralmente operando com S.O. Linux customizado (Raspberry Pi OS).
- Utiliza chips da BroadCom, com arquitetura ARM.
- Altamente utilizada no mercado – comum em produtos de maior valor agregado e menor escala de produção.



Pi Pico (RP2040)



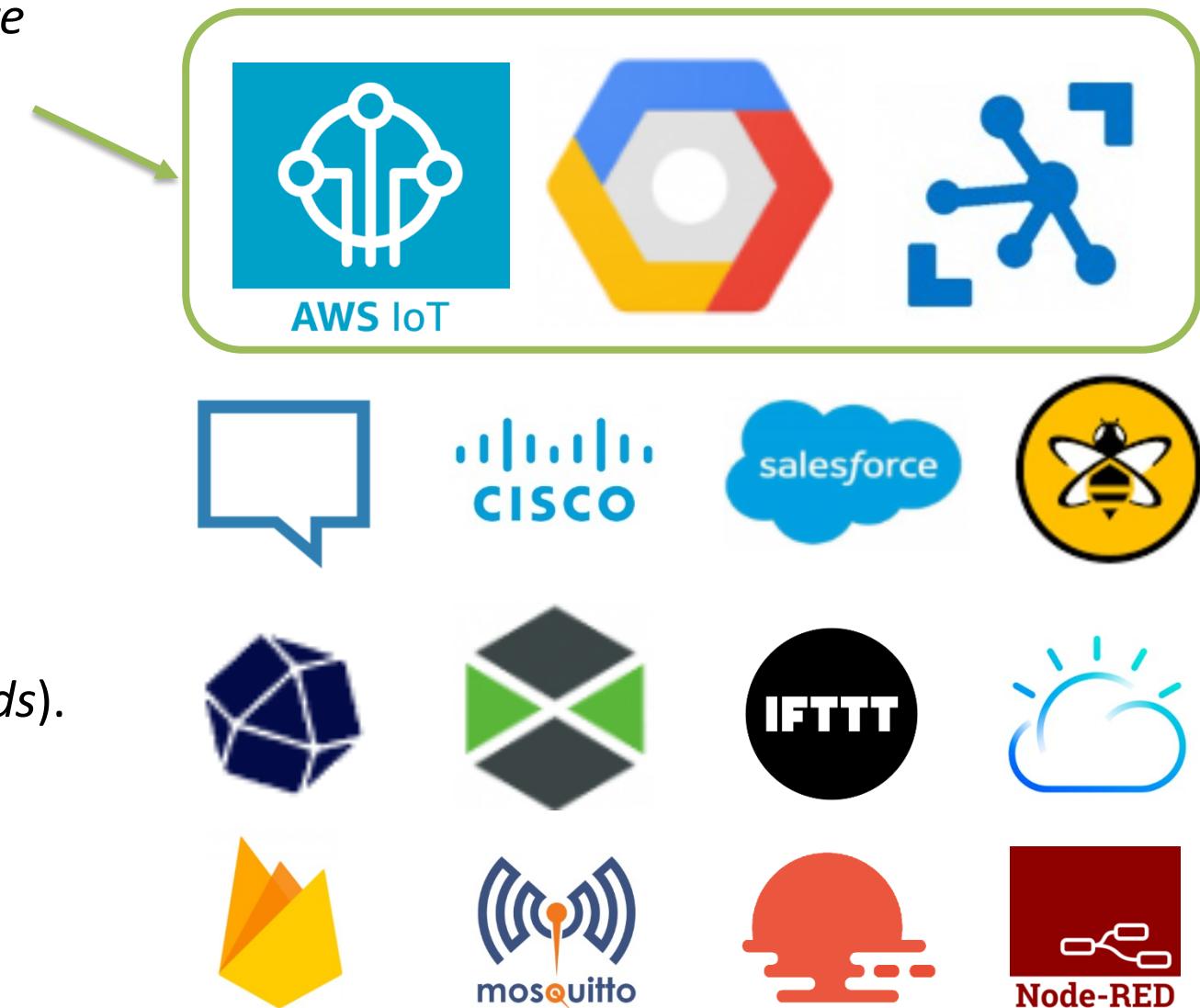
Pi 4 - Model B



Pi Zero 2W

# Alguns exemplos de plataformas de software

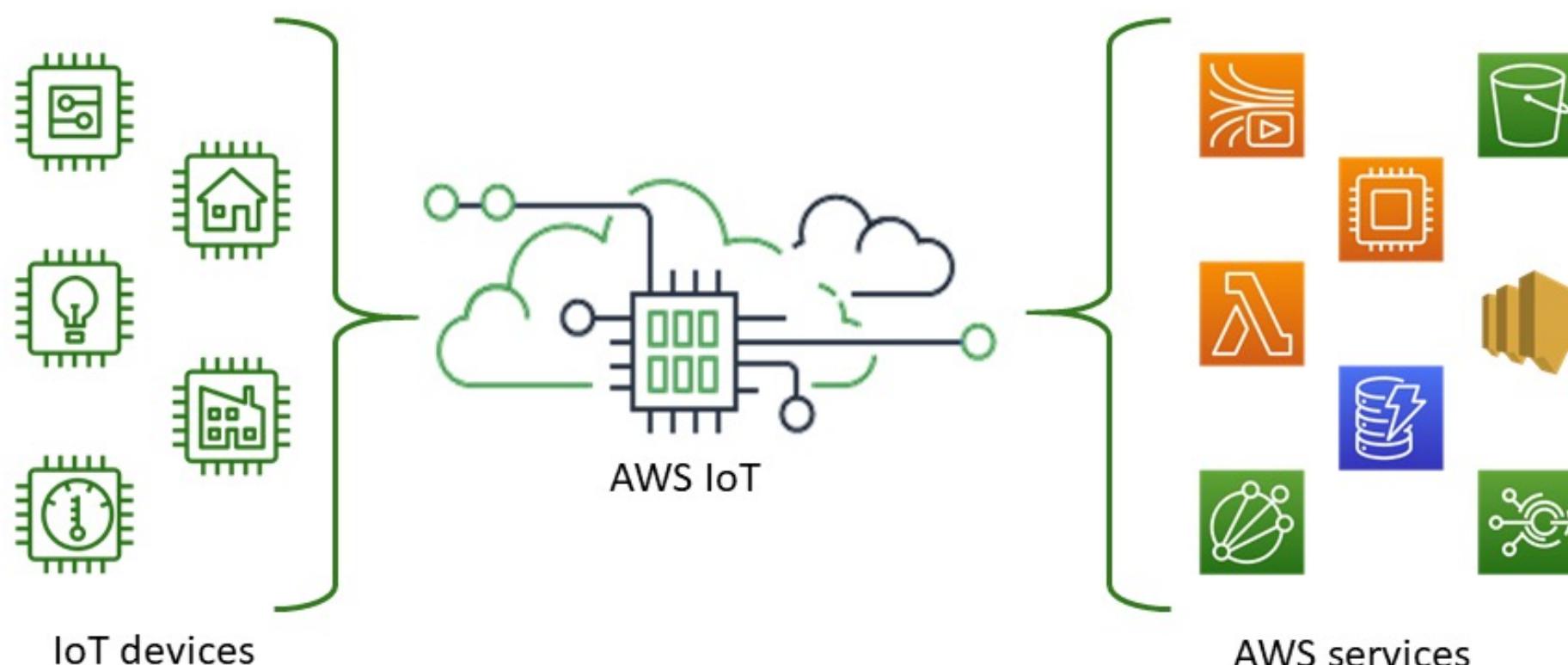
- AWS, Google Cloud e Microsoft Azure
  - As 3 maiores plataformas detêm aprox. 80% de *market share* para serviços de nuvem, para tráfego de dispositivos de IoT.
- De modo geral, as plataformas possibilitam:
  - Armazenamento de dados de dispositivos.
  - Controle automático de ações nos dispositivos.
  - Ferramentas para análise & visualização de dados (*dashboards*).





# AWS IoT Core

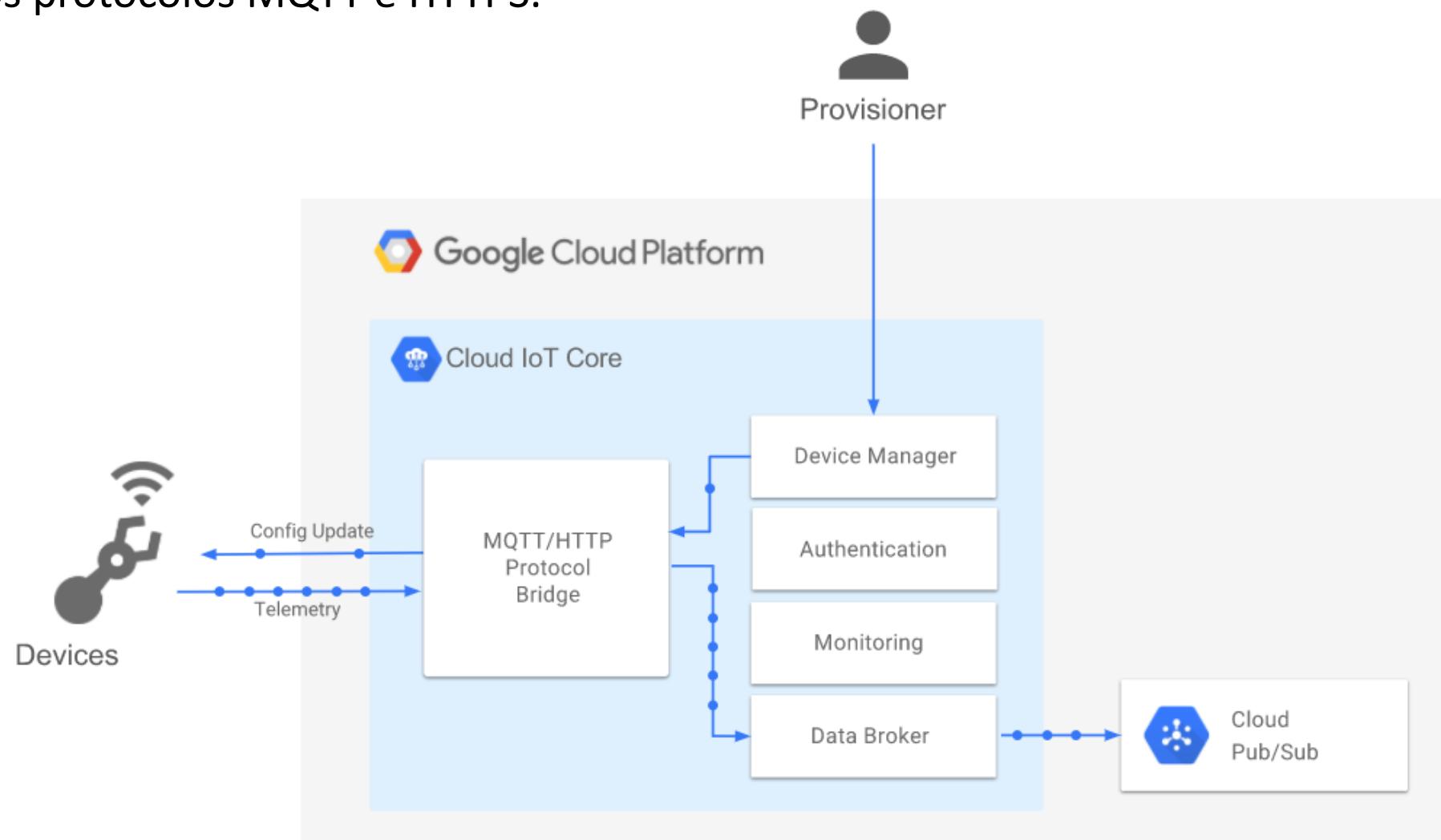
- Plataforma do ecossistema AWS da Amazon para conectar dispositivos e rotear mensagens para serviços AWS.
- Alto nível de escalabilidade, modelo de negócios flexível, infraestrutura para número enorme de dispositivos.
- Suporta os protocolos MQTT, HTTPS, e LoRaWAN.





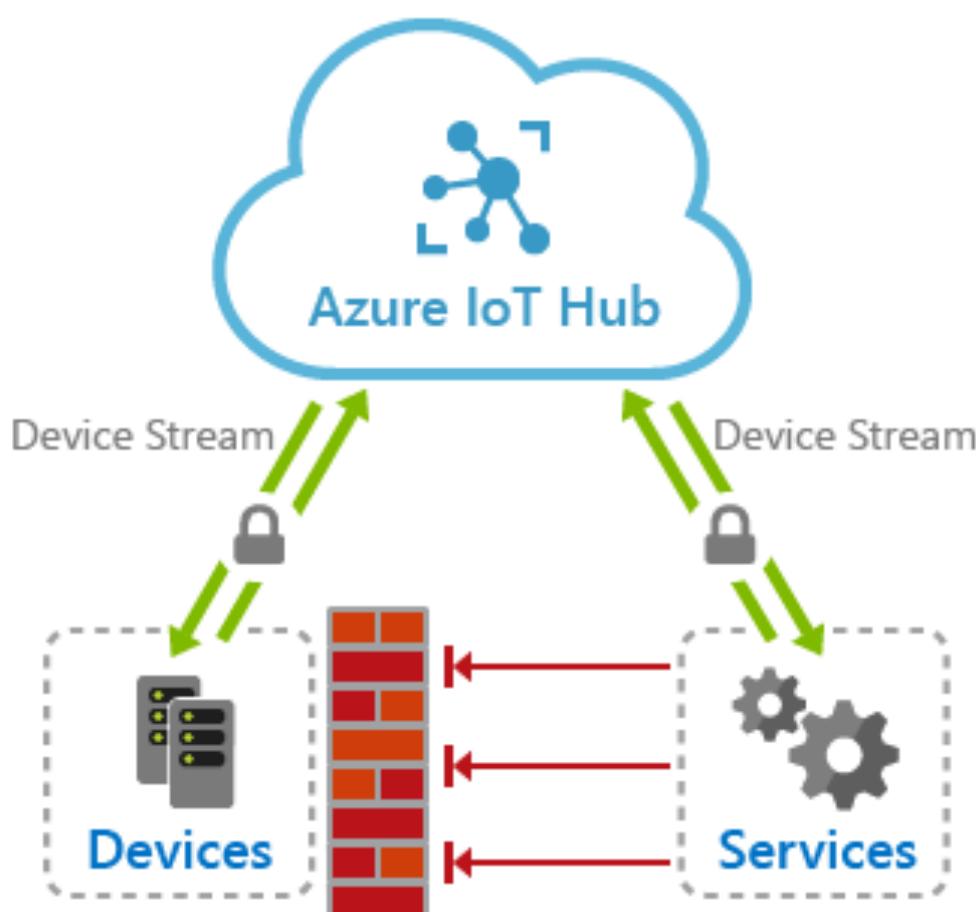
# Google Cloud IoT

- Plataforma do Google para conectar, gerenciar e ingerir dados de vários dispositivos.
- Excelente plataforma de análise de dados, alto volume de armazenamento e serviços de *machine learning*.
- Suporta os protocolos MQTT e HTTPS.



# Azure IoT Hub

- Plataforma do ecossistema Azure, da Microsoft, para vários serviços de nuvem.
- Facilidade de integração, elaborado sistema de segurança dos dados e suporte.
- Suporta os protocolos MQTT, AMQP e HTTPS.



# Diversas outras plataformas...



ThingsPeak



Cisco IoT Cloud Connect



Salesforce IoT Cloud



HiveMQ Cloud



InfluxDb



Firebase



ThingWorxs



IFTTT



IBM IoT Cloud



Mosquitto MQTT Broker



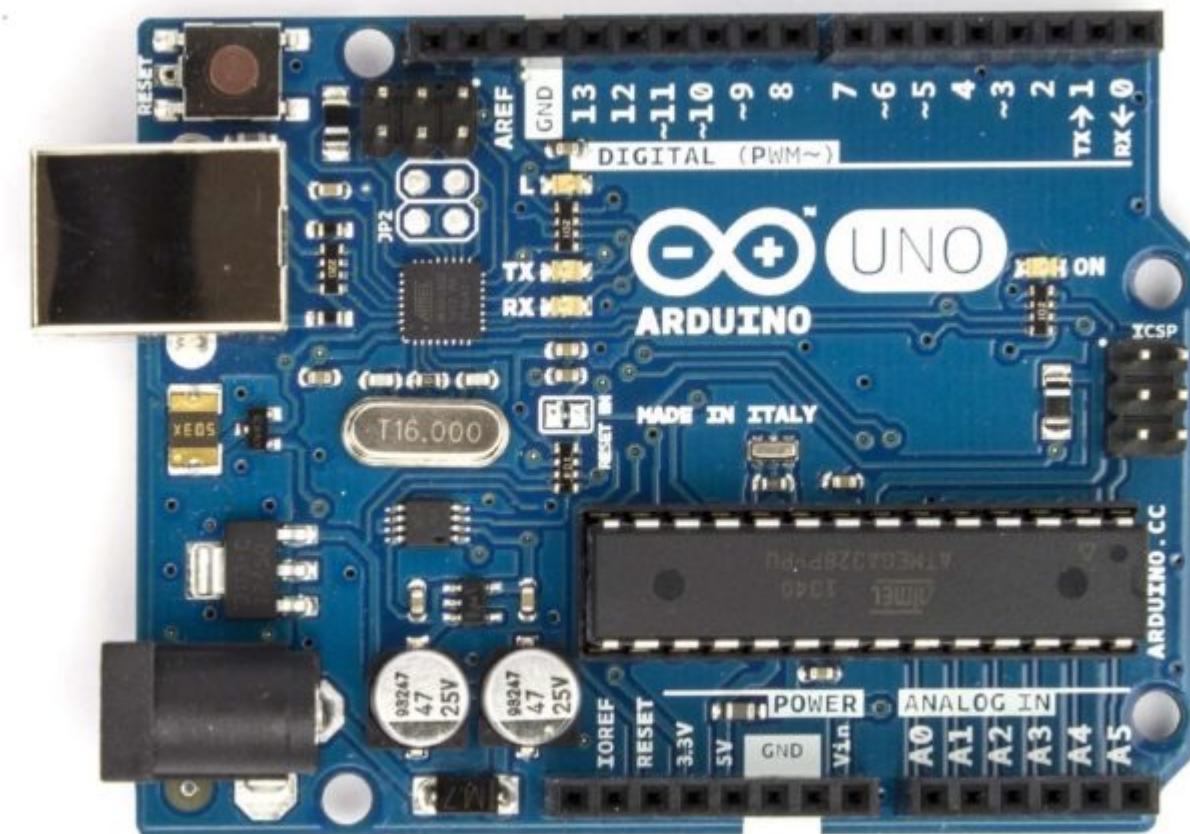
OpenWeather



Node-RED

# Arduino - Introdução

- Desenvolvido originalmente em 2005, é uma plataforma eletrônica de código aberto, baseada em *software* e *hardware* de fácil utilização.
- Objetivo: disponibilizar um dispositivo acessível, de baixo custo, funcional e fácil uso por estudantes e profissionais.



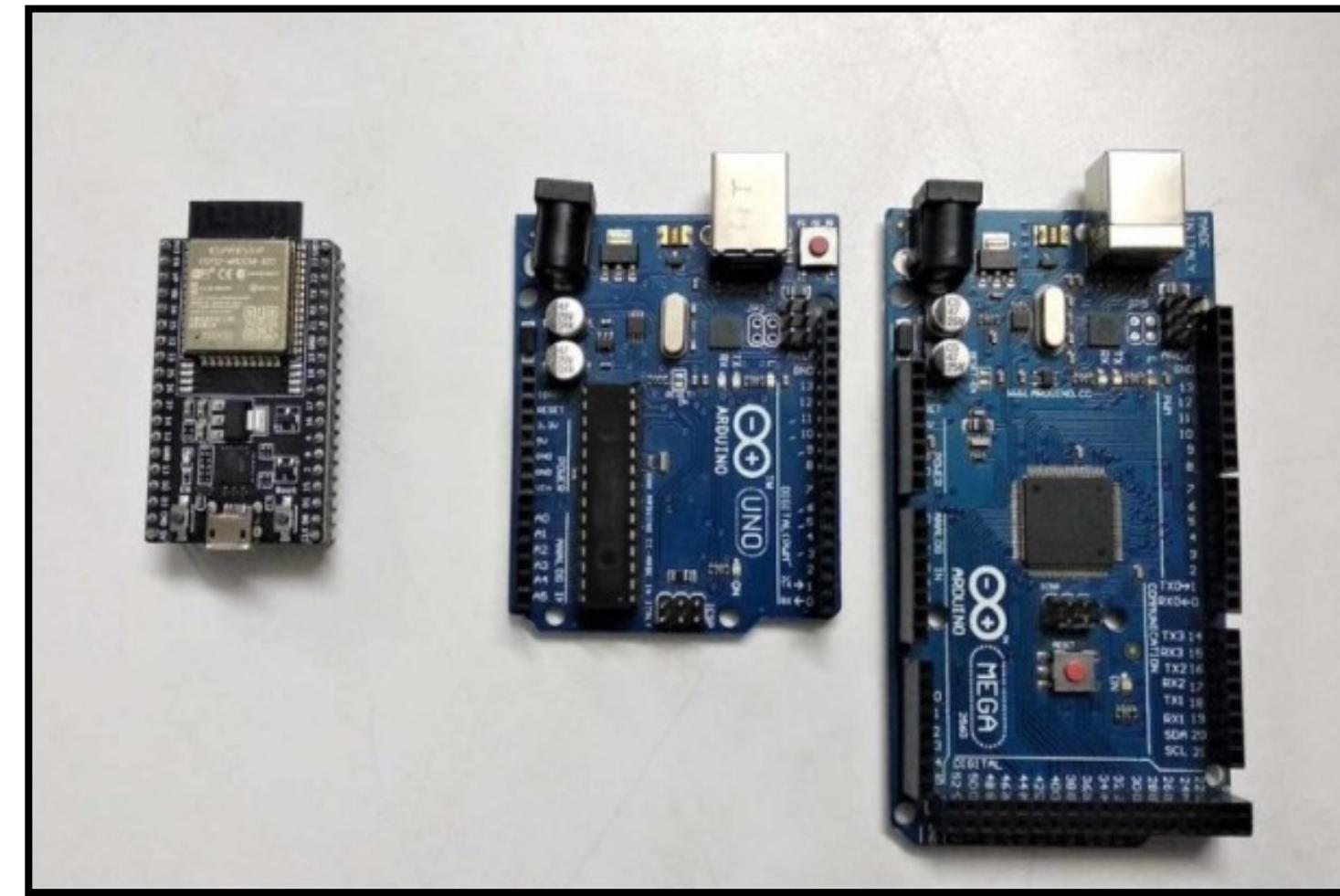
# Arduino – Algumas vantagens de uso

- Baixo custo de prototipagem.
- Softwares de simulação gratuitos disponíveis.
- Interface simplificada.
- Ampla documentação e exemplos de uso.
- Ampla comunidade de desenvolvedores.
- Vários modelos de plataformas disponíveis.



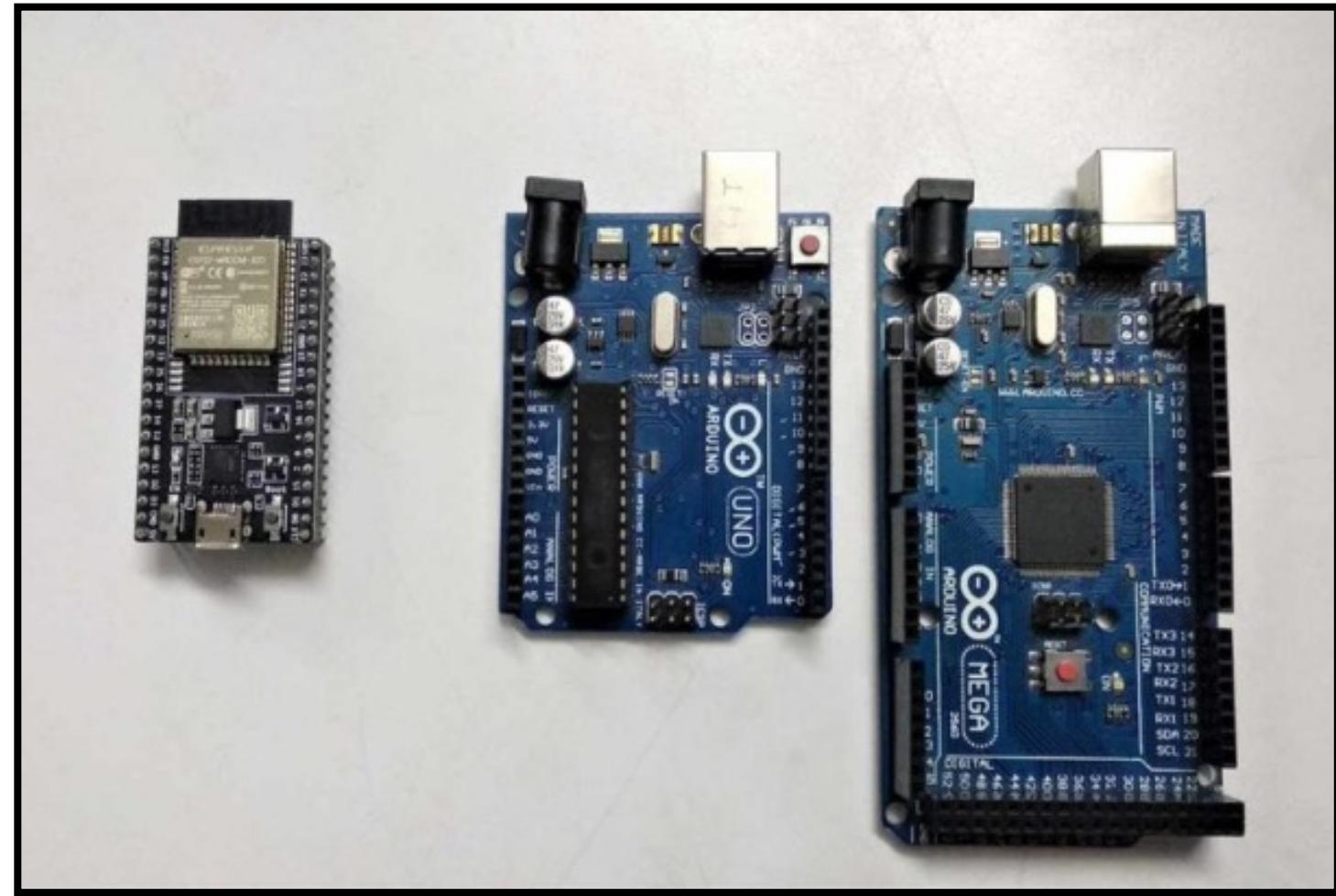
# ESP32 - Introdução

- Mesmo com os benefícios oferecidos pelo *hardware open-source* do Arduino, a placa original não é sempre adequada para todas as aplicações.
- Por exemplo, no quesito velocidade de *clock*: O AVR ATmega328P da placa original possui apenas 16 [MHz].



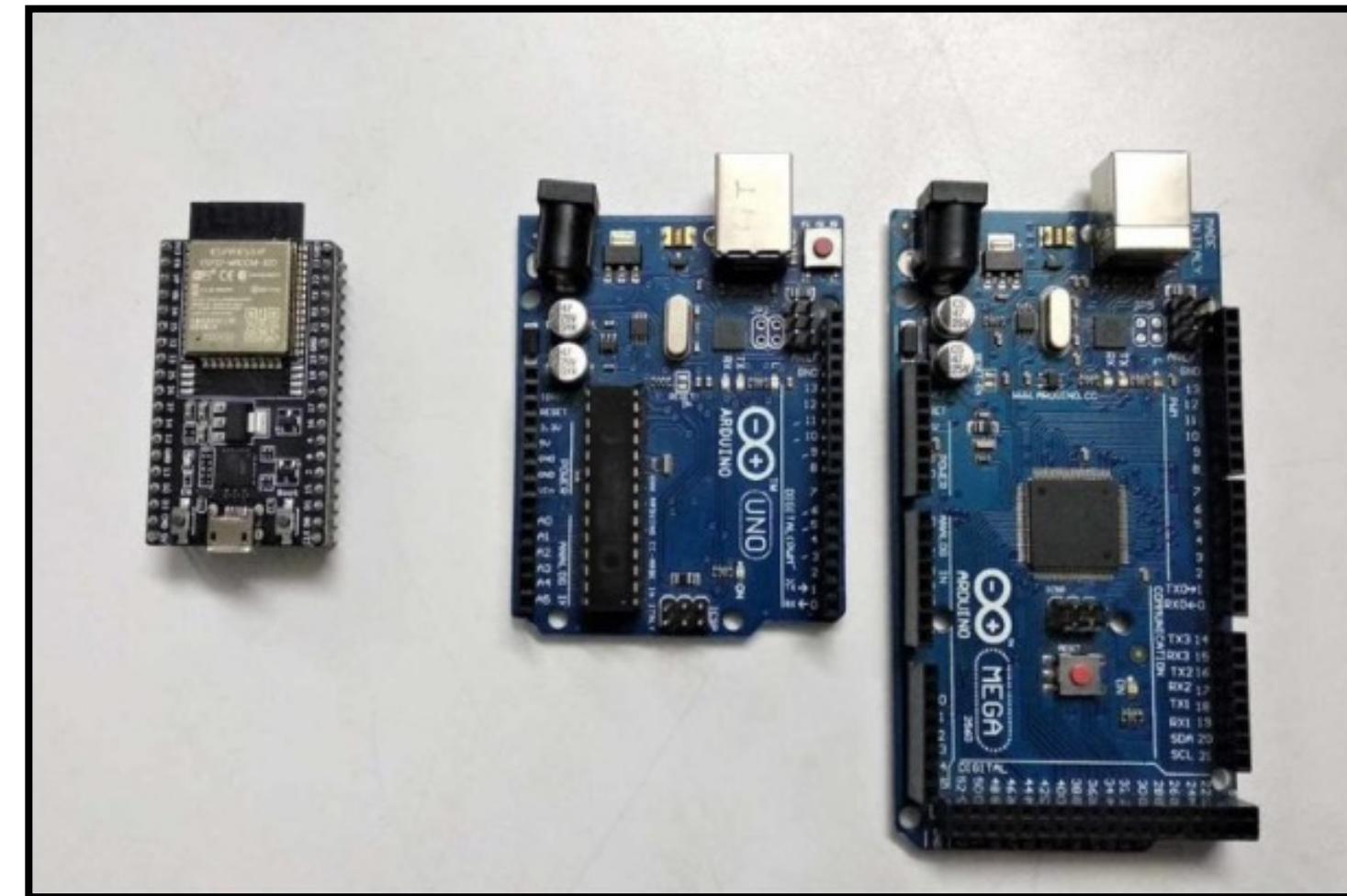
# ESP32 - Introdução

- Opções de conectividade sem fio são possíveis, mas necessitam de placas de extensão externas (*shields*).
- Assim, a linha de microcontroladores ESP da Espressif é uma opção interessante para projetos de IoT.



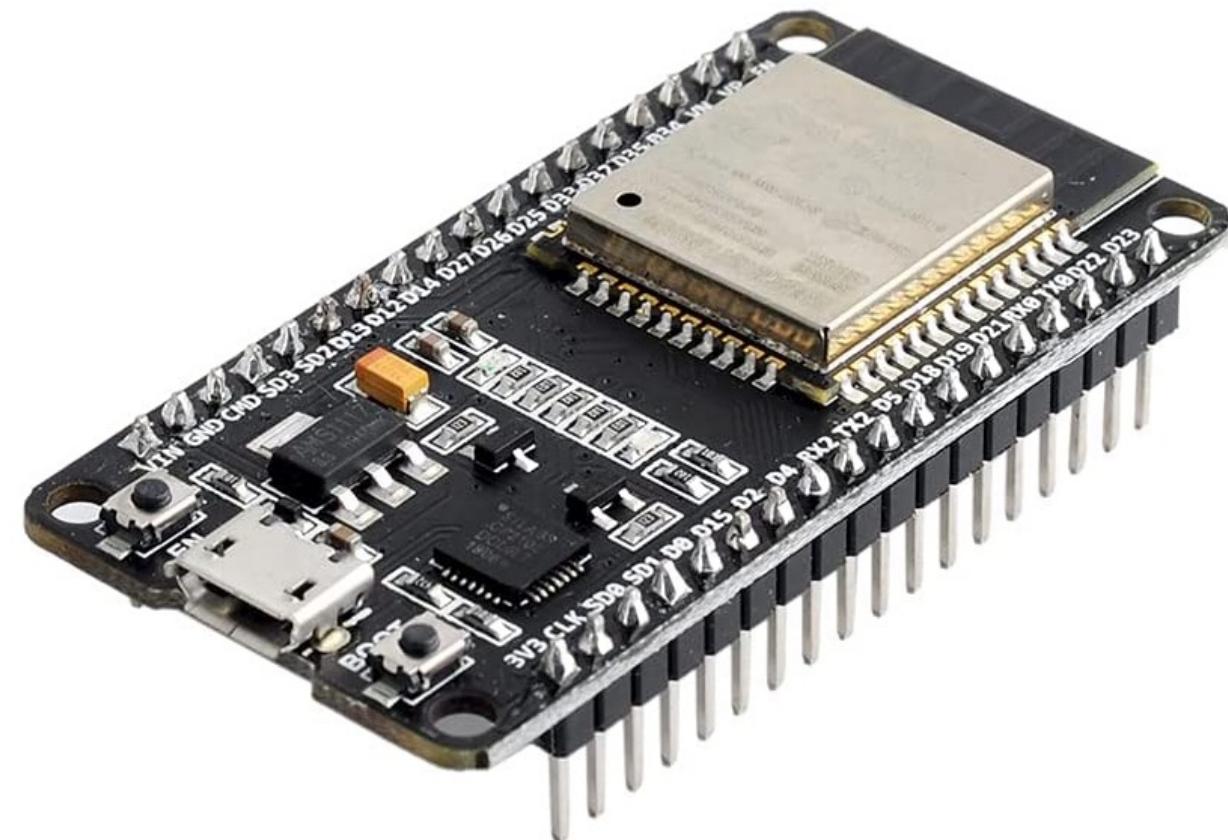
# ESP32 - Introdução

- O ESP32 apresenta opções de conectividade integradas e boa relação custo/poder de processamento.
- Há várias diferenças entre as duas famílias – em especial, no que diz respeito à *clock*, periféricos e preço.
- A família ESP32/ESP8266 pode ser utilizada com o *framework* Arduino, como será demonstrado no curso.



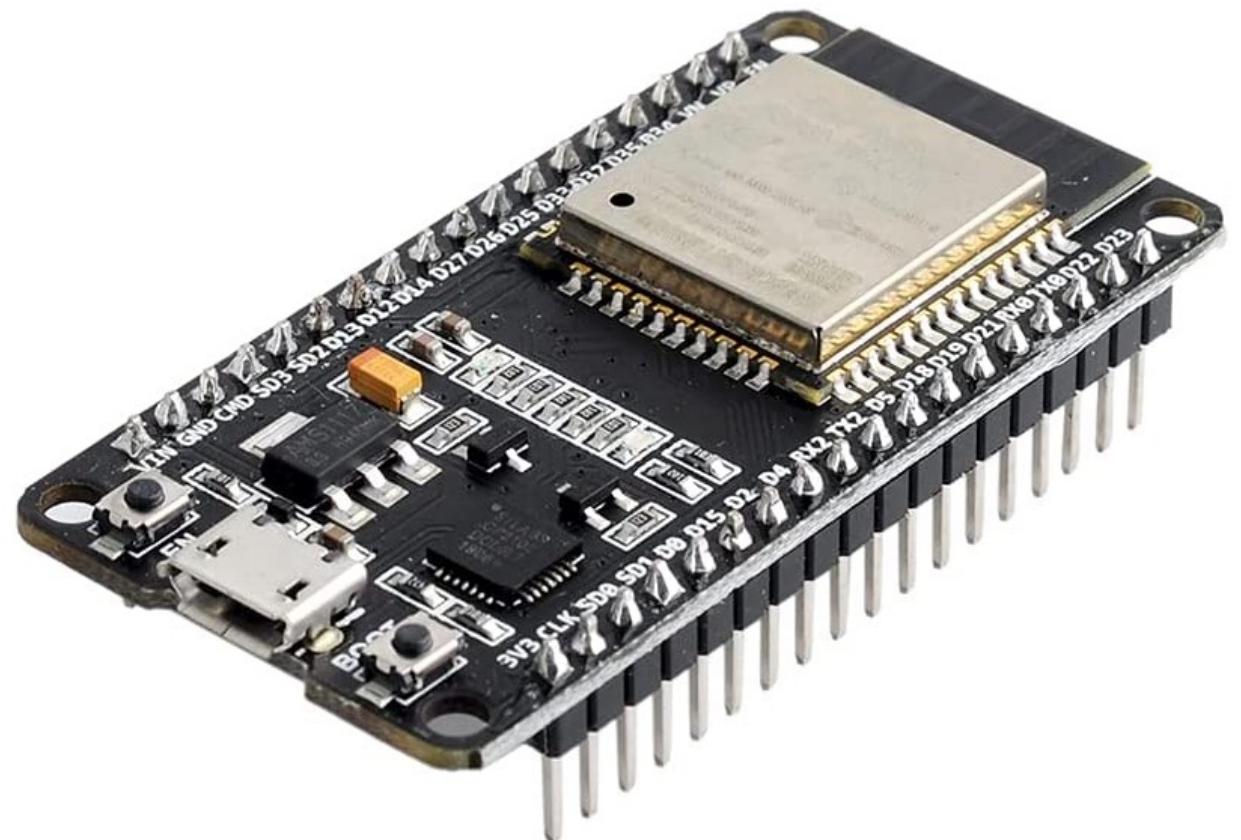
# O módulo ESP32-WROOM-32

- O ESP32 é uma versão com *design* melhorado do SoC original da Espressif, o ESP8266.
- Além de Wifi, oferece opção de conectividade BLE 4.0 (*Bluetooth Low Energy*), enquanto o ESP8266 possui apenas Wifi.



# O módulo ESP32-WROOM-32

- É mais rápido e está disponível em um *design* dual-core.
- Também é capaz de operar em um modo de baixo consumo (*ultra-low power*), ideal para aplicações alimentadas por bateria.

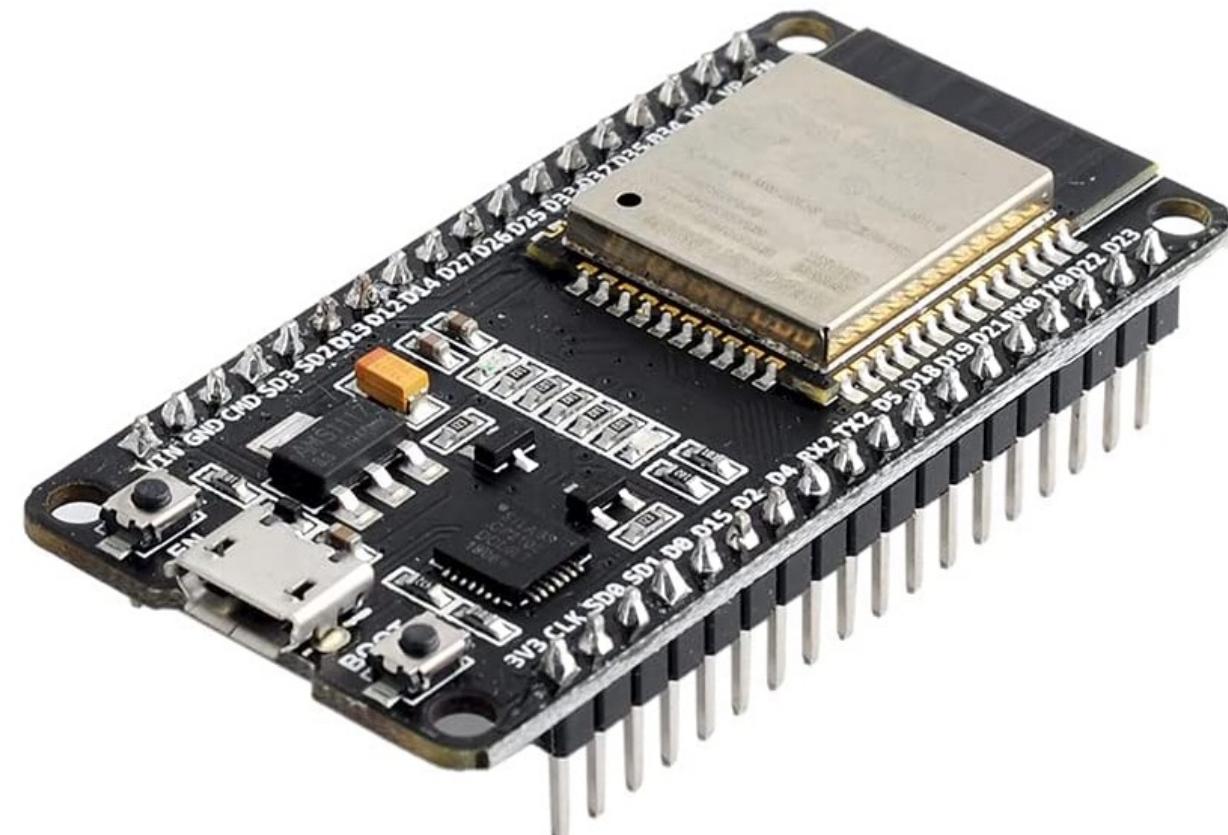


# O módulo ESP32-WROOM-32

Outros recursos do ESP32 incluem:

- Capacidade de executar programas de 32 bits.
- Frequência do *clock* de até 240MHz.
- Memória RAM de 512 kB.
- Variedade de periféricos: ADCs, DACs, UART, SPI, I2C.
- Sensor capacitivo (*touch*) integrado.
- Sensor de efeito *hall* integrado.
- Sensor de temperatura integrado

*(mas geralmente impreciso – alta variância de chip-a-chip).*

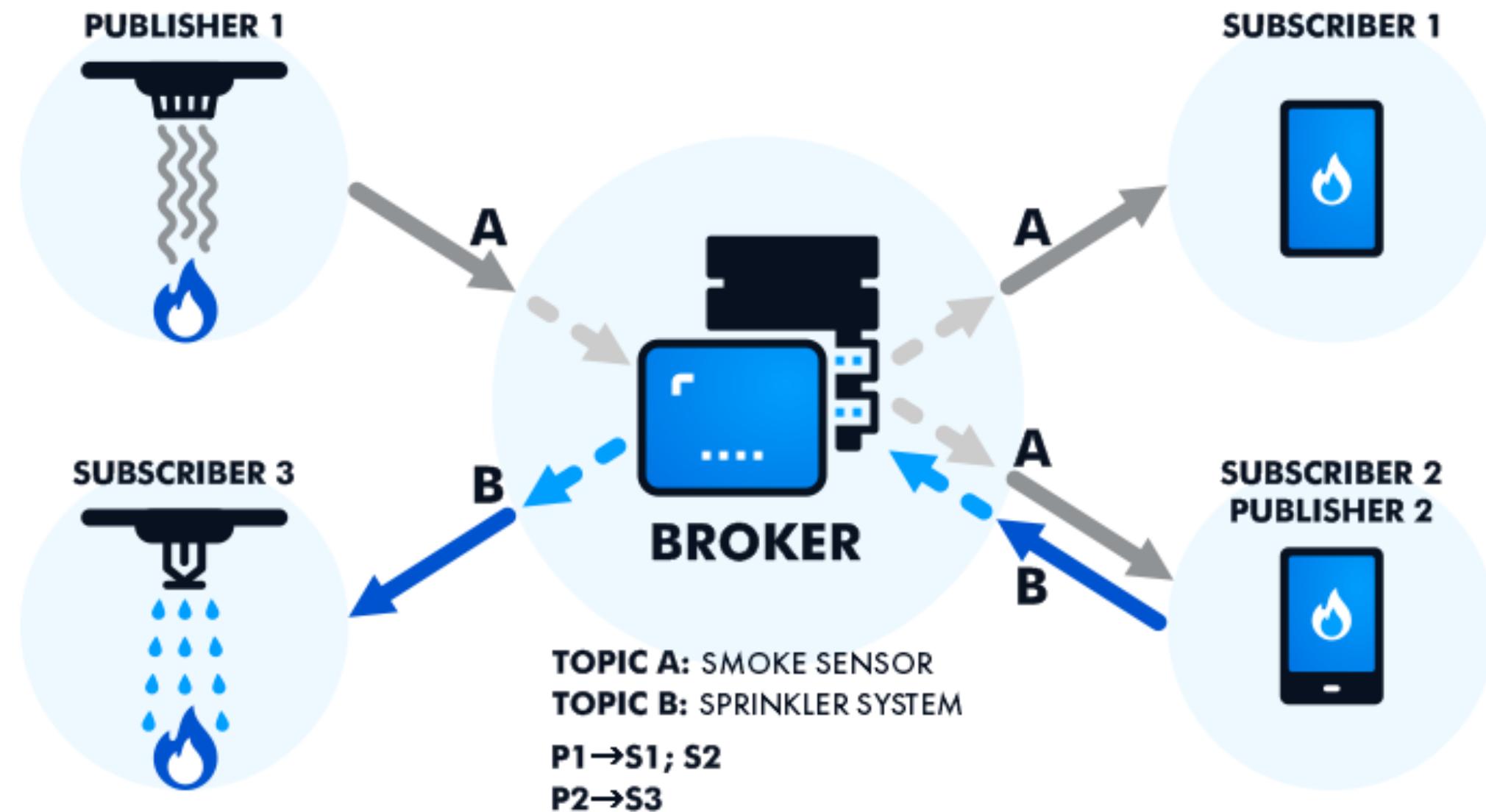


# Comparações entre ESP32, ESP8266 e Arduino Uno

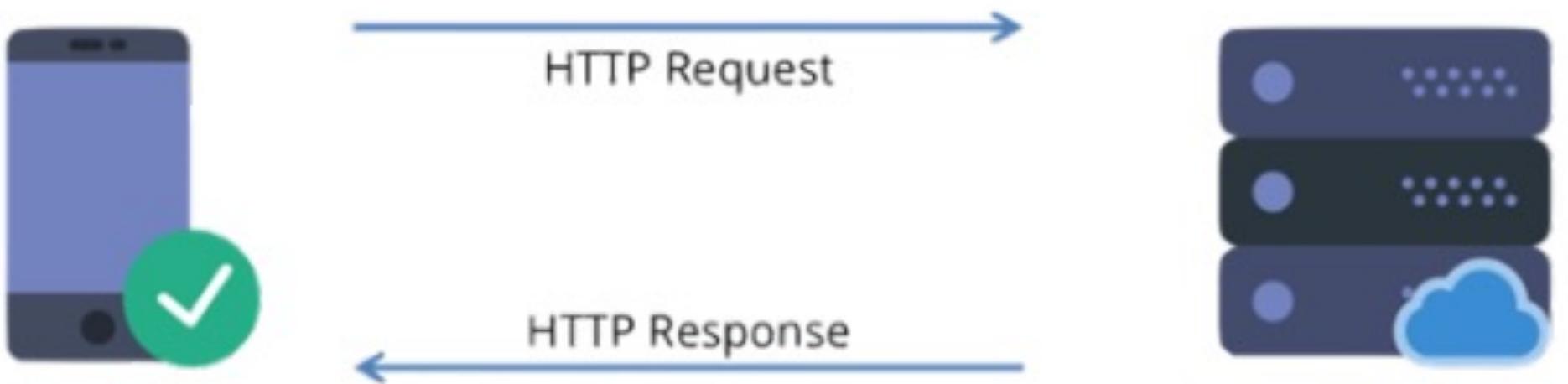
	ESP32-WROOM-32	ESP8266	Arduino Uno R3 (ATmega328P)
<b>Arquitetura</b>	32 bits	32 bits	8 bits
<b>Clock Máximo</b>	240 MHz	80 MHz	16 MHz
<b>Wifi</b>	Sim	Sim	Não
<b>Bluetooth</b>	Sim	Não	Não
<b>RAM</b>	512 KB	128 KB	2 KB
<b>FLASH</b>	4 MB	4 MB	16 KB
<b>GPIO's</b>	36	17	14
<b>ADC</b>	18	1	6
<b>DAC</b>	2	0	0

# Telemetria e controle com MQTT

# Introdução

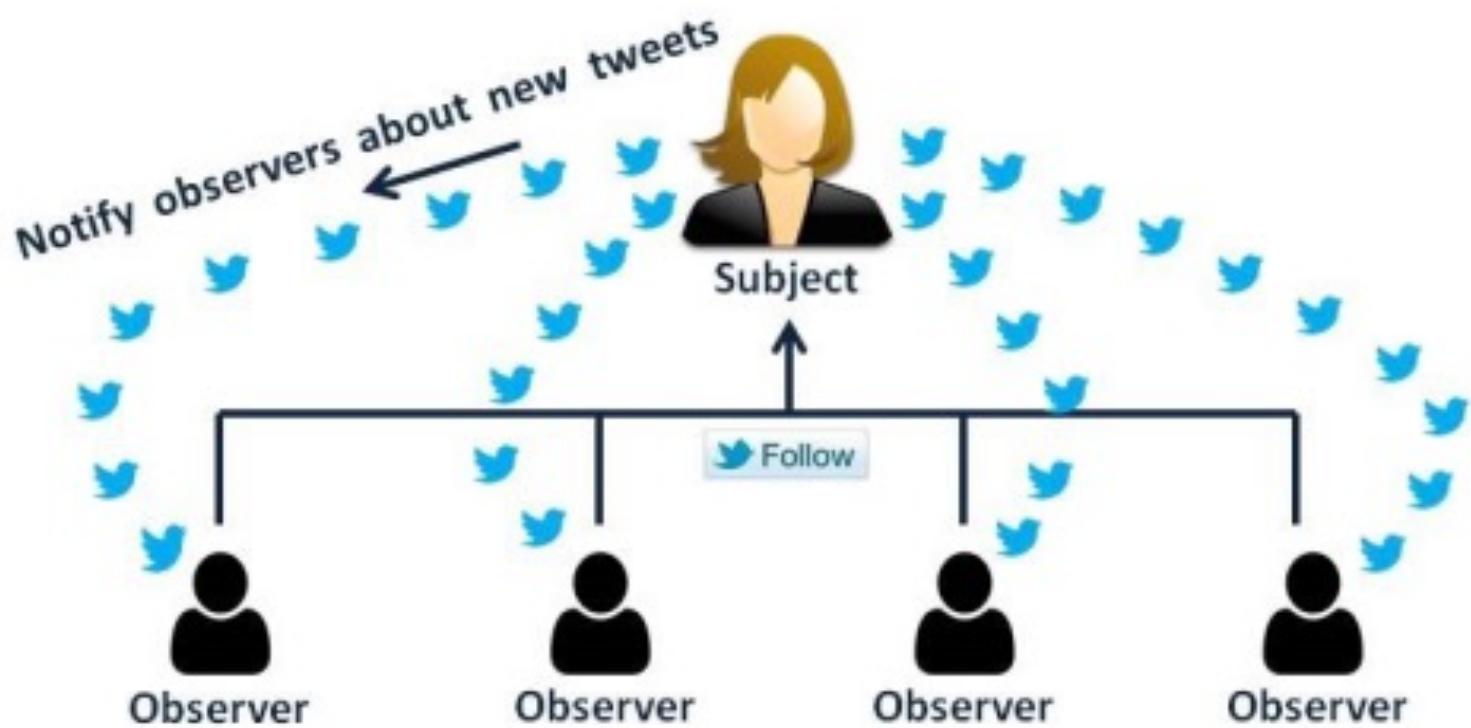


# Introdução: Cliente - servidor

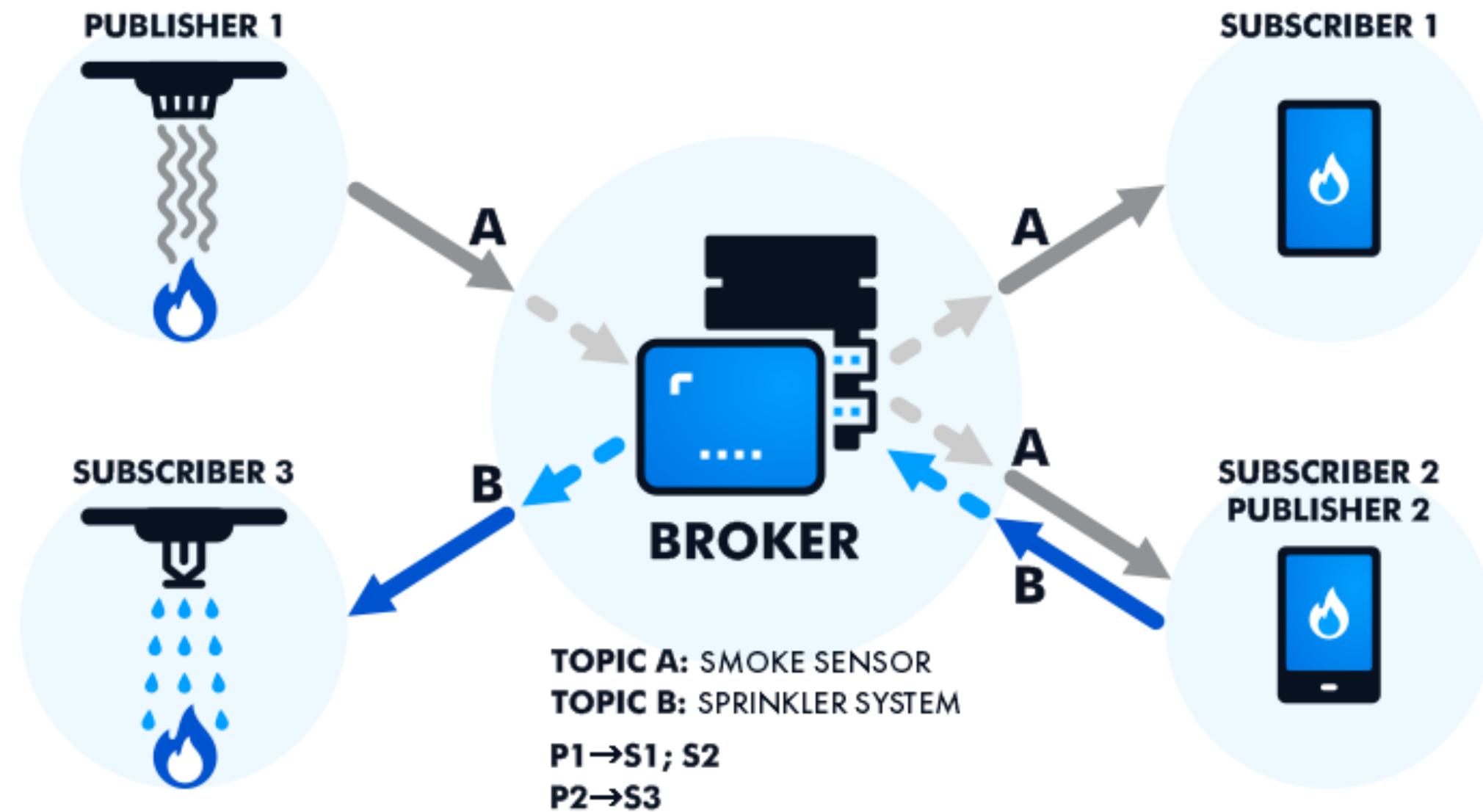


# Introdução: Observador

Observer Design Pattern

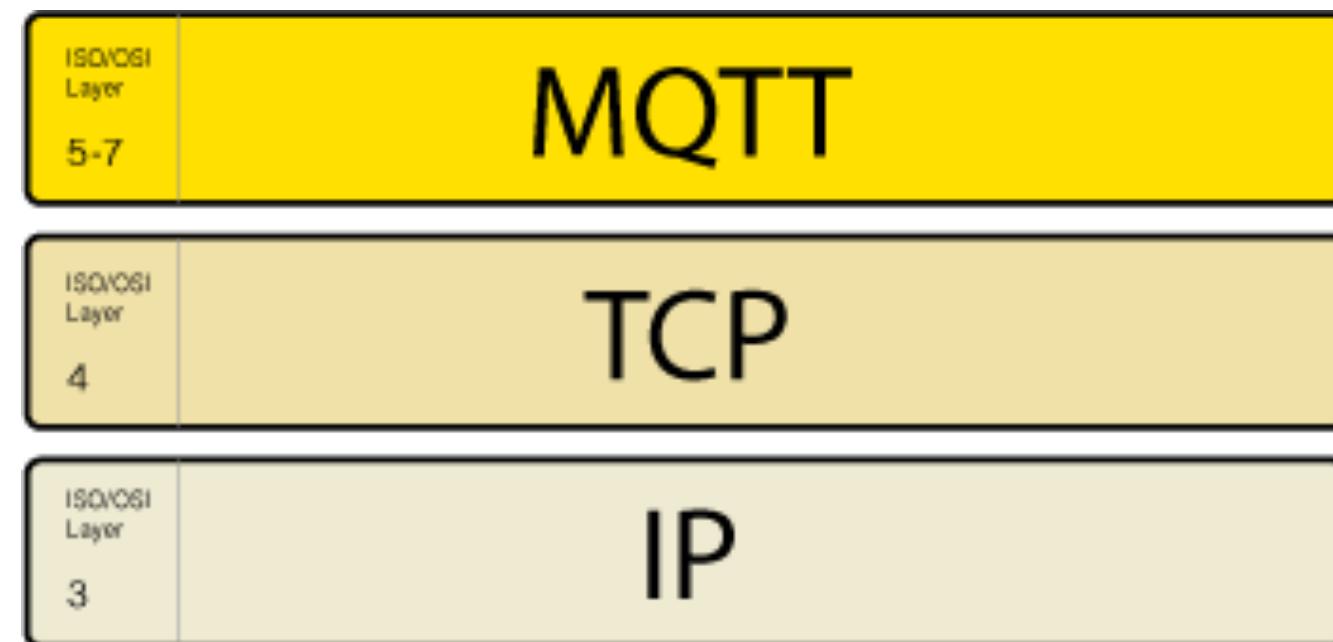


# Introdução



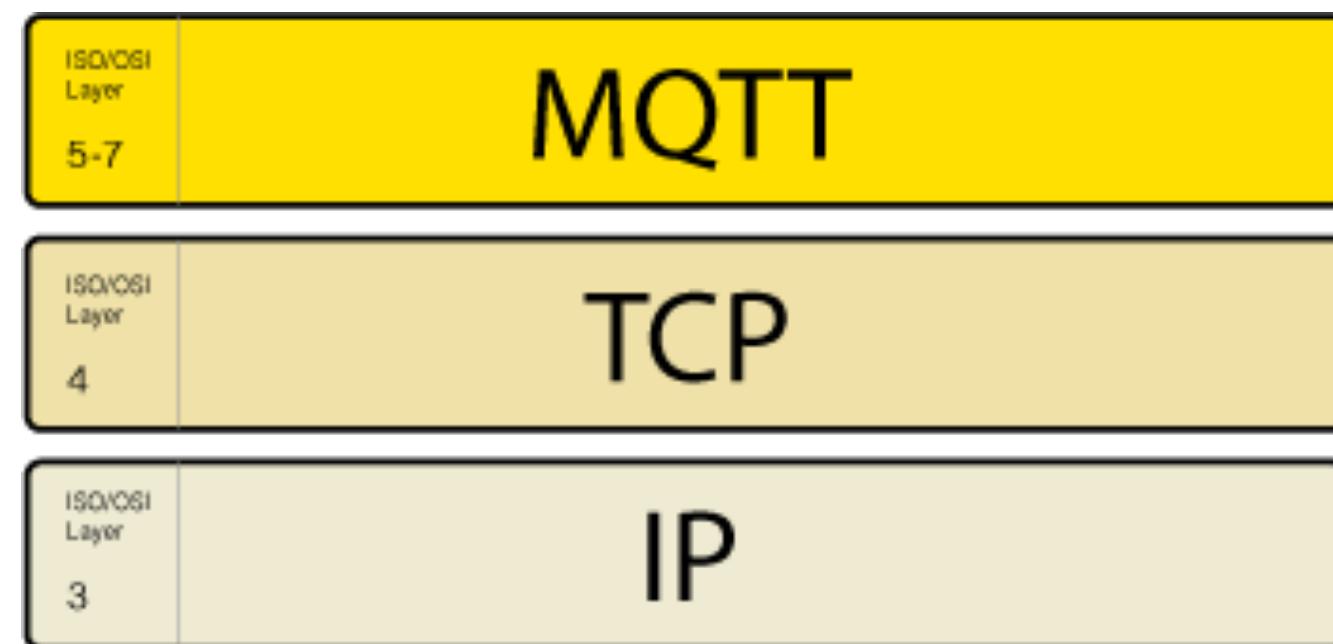
# MQTT: Introdução

- MQTT (*Message Queue Telemetry Transport*) é um protocolo de mensagens criado em 1999.
- Seu foco é a comunicação entre dispositivos, em rede local ou com acesso pela Internet.
- É implementado sobre o protocolo TCP.



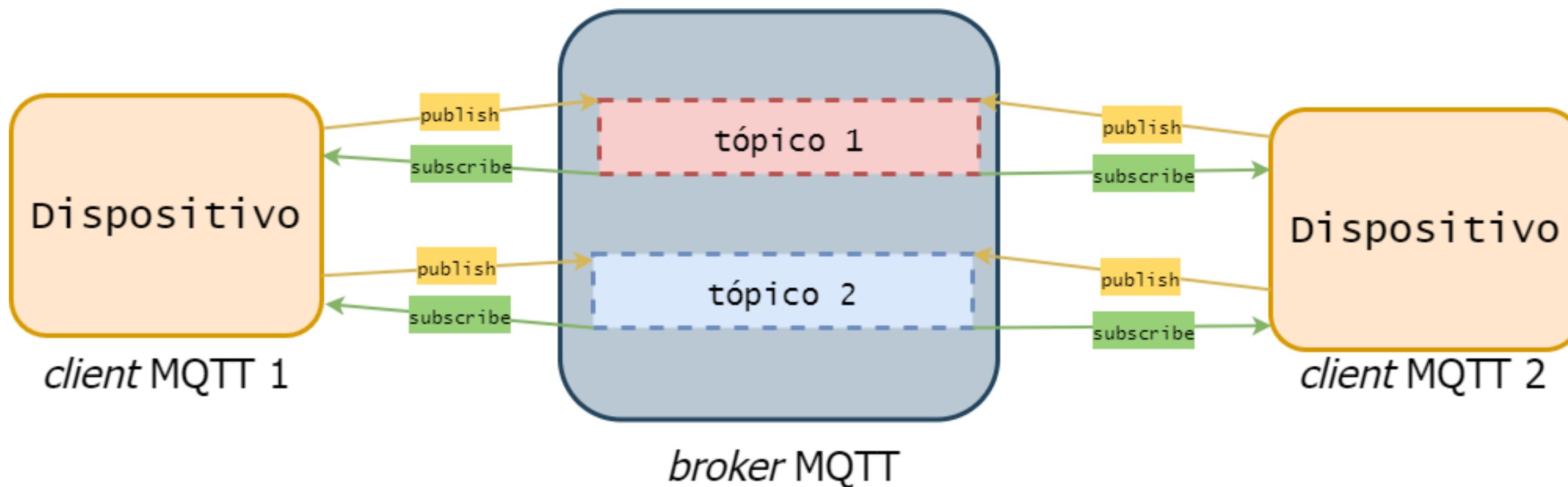
# MQTT: Introdução

- Possui *overhead* pequeno e complexidade de implementação relativamente baixa.
- Consequentemente, requer baixa carga computacional (em processamento e uso de memória RAM).
- MQTT é um dos protocolos mais adequados para telemetria em dispositivos embarcados de forma geral.



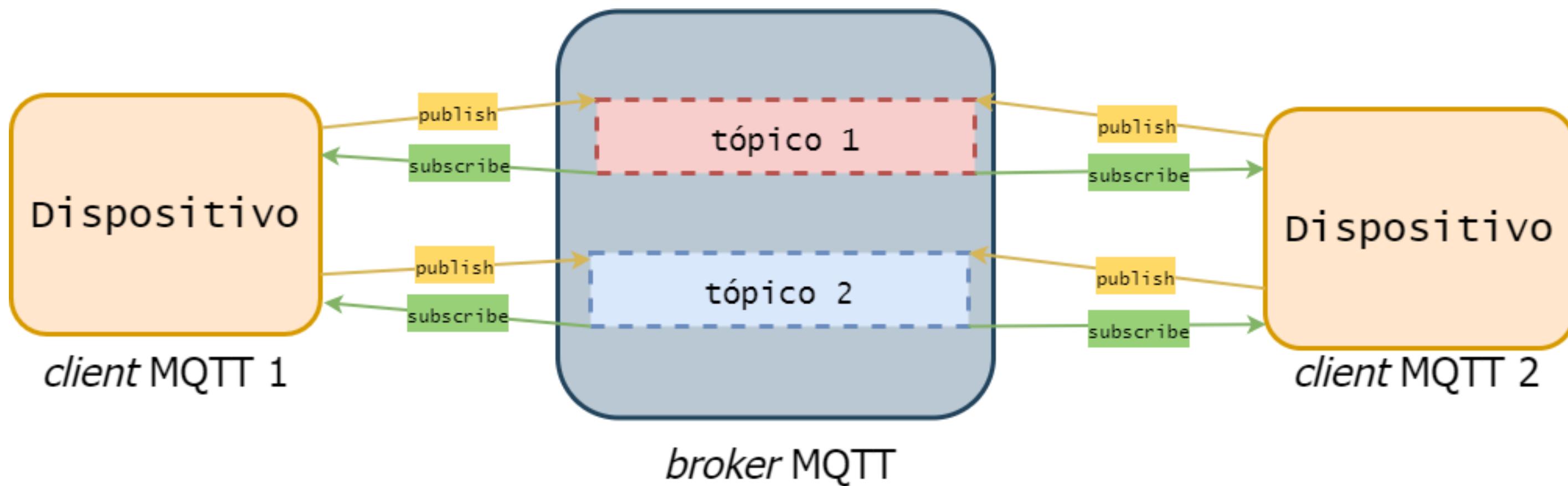
# MQTT: *Client*

- Denominamos como clientes (*clients*) os diversos dispositivos que se conectam à um servidor (*broker*) para enviar e/ou receber mensagens de dados com MQTT.
- Um *client* MQTT implementa uma *lib* MQTT para comunicação com o *broker*.



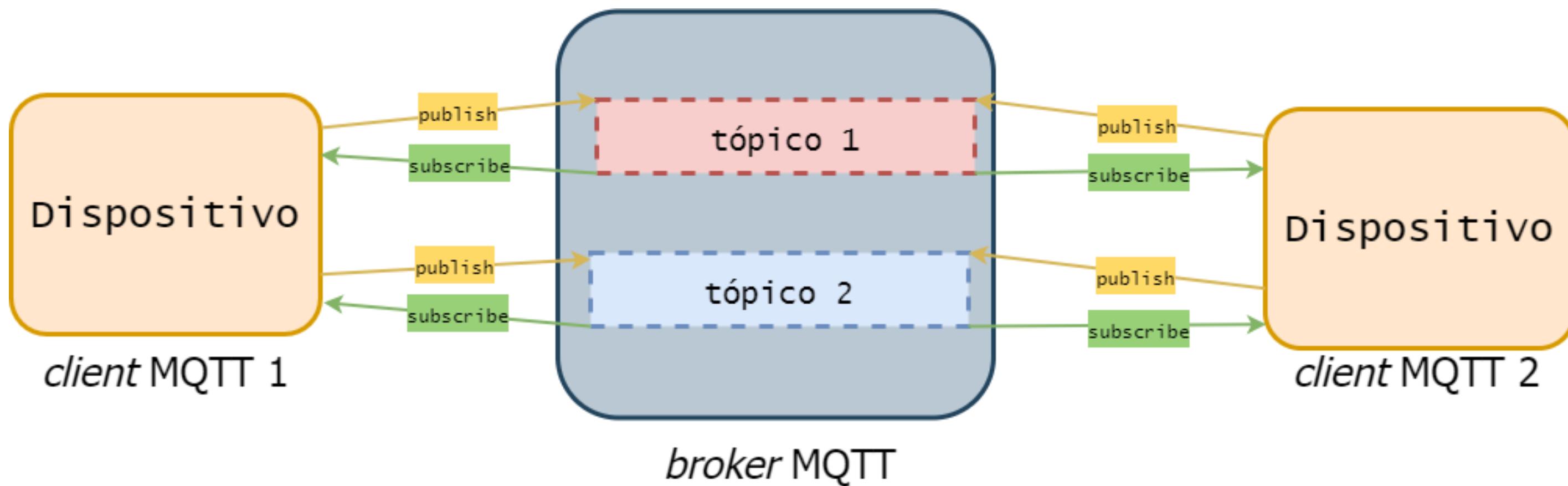
# MQTT: *Client*

- Múltiplos *clients* podem estar conectados à um mesmo *broker*.
- O ESP32 atuará como *client* nos exemplos do curso, mostrados a seguir.



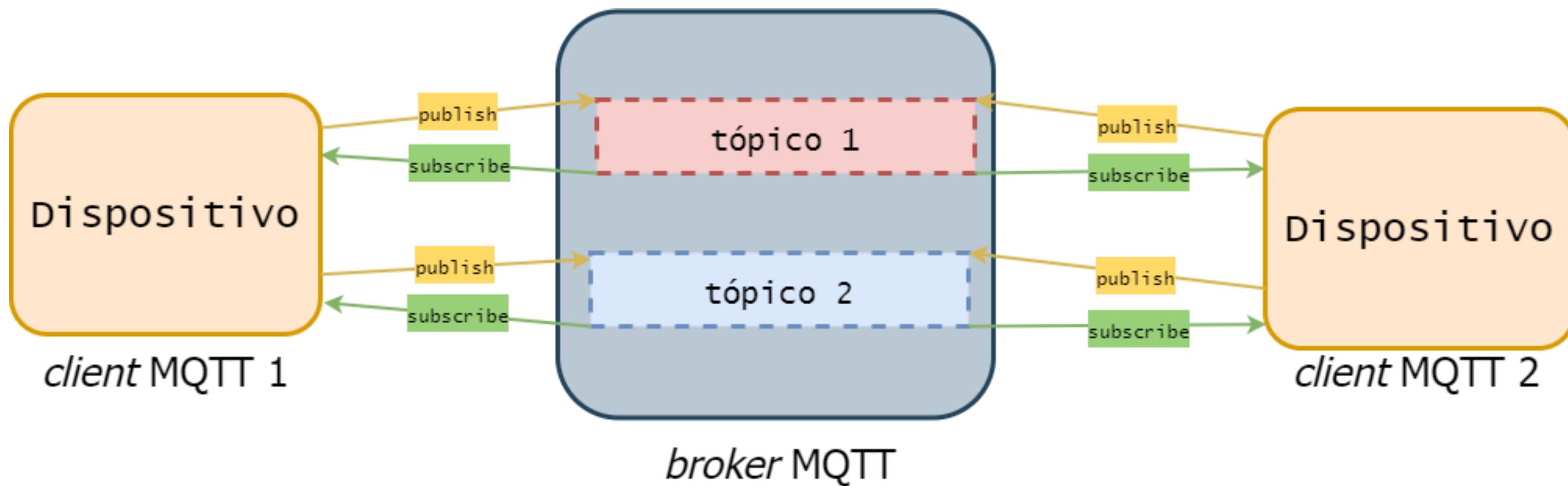
# MQTT: *Broker*

- O *broker* é o servidor que distribui mensagens entre os *clients* com os quais está conectado.
- É responsável por receber as mensagens, filtrar por tópicos, determinar a quais *clients* mensagens devem ser designadas e encaminhar as mensagens aos *clients*.



# MQTT: *Broker*

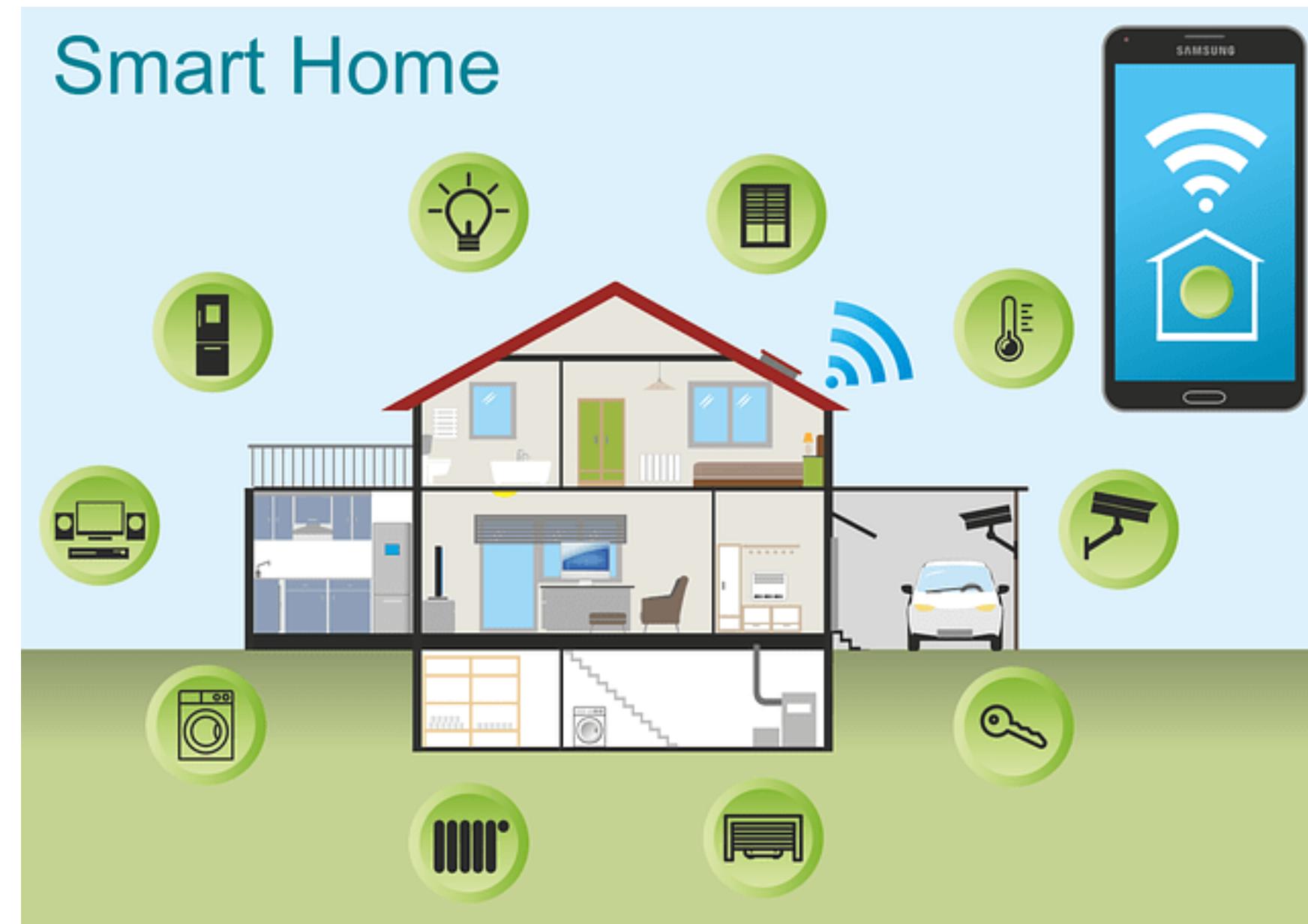
- *Brokers* podem existir em uma rede local ou hospedados em um servidor remoto.
- Podem, também, possuir controle de autenticação (usuário/senha) e criptografia ponta-a-ponta. O *broker* que acessaremos nos exemplos é público, geralmente utilizado para testes.



## MQTT: *Broker*

- Como a maior parte do “trabalho pesado” de gerenciar mensagens fica a cargo do *broker*, os *clients* podem realizar outras tarefas uma vez que as mensagens já foram enviadas, podendo, inclusive, entrar em modo de baixo consumo (*low-power*).
- Isso é um dos pontos que contribui para a eficiência de consumo de energia deste protocolo.
- Outro ponto é que, com *overhead* menor, menos bits são transmitidos: picos de consumo de energia ocorrem justamente no momento da transmissão.
- Menos bits transmitidos = menor consumo de modo geral.

## MQTT: *Broker local*

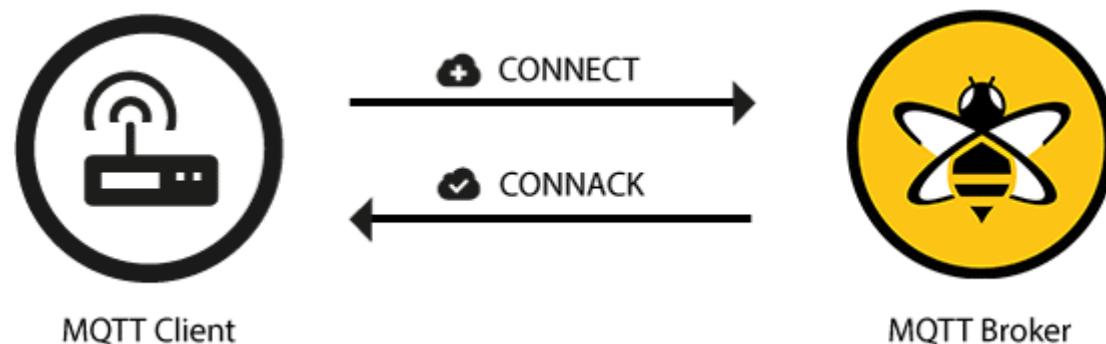


# MQTT: *Broker online*



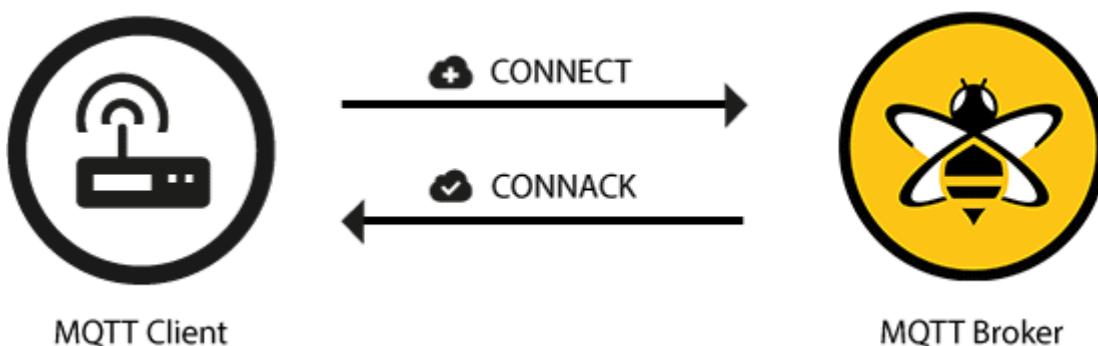
# MQTT: Conexão dos *clients* ao *broker*

- A conexão MQTT é sempre entre um *client* e um *broker*. *Clients* nunca comunicam diretamente entre si.
- Para iniciar uma conexão, um *client* envia uma mensagem com solicitação de conexão (**CONNECT**) ao *broker*.
- Se aceitar a conexão, o *broker* responde com uma mensagem (**CONNACK**) e um código de status da conexão.



# MQTT: Conexão dos *clients* ao *broker*

- Quando uma conexão é estabelecida, o *broker* a mantém ativa até que o *client* solicite a desconexão, ou até que a conexão seja interrompida.
- Nos exemplos do curso, as mensagens de **CONNECT** / **CONNACK** já são implementadas pelas *libs* MQTT adicionadas aos projetos do ESP32, implementadas com o *framework* Arduino e ESP-IDF.



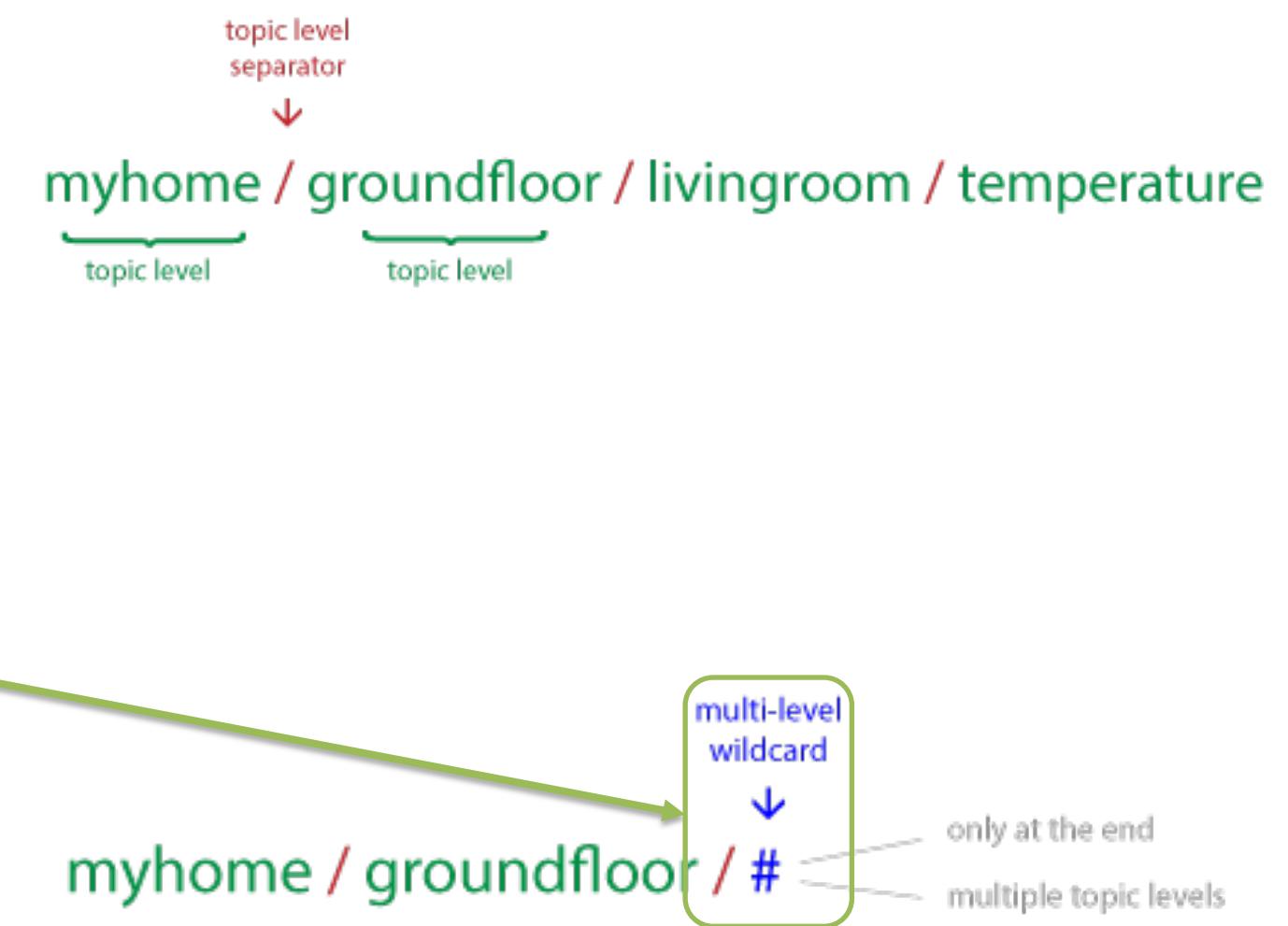
# MQTT: Tópicos

- Um tópico é, essencialmente, uma *string* de texto. O *broker* utiliza tópicos para filtrar quais *clients* irão receber as mensagens recebidas.
- Um tópico pode ter um ou mais níveis. Cada nível é separado por uma barra (/).
- É possível combinar *strings* de tópicos em vários níveis, criando uma estrutura similar a diretórios de pastas.



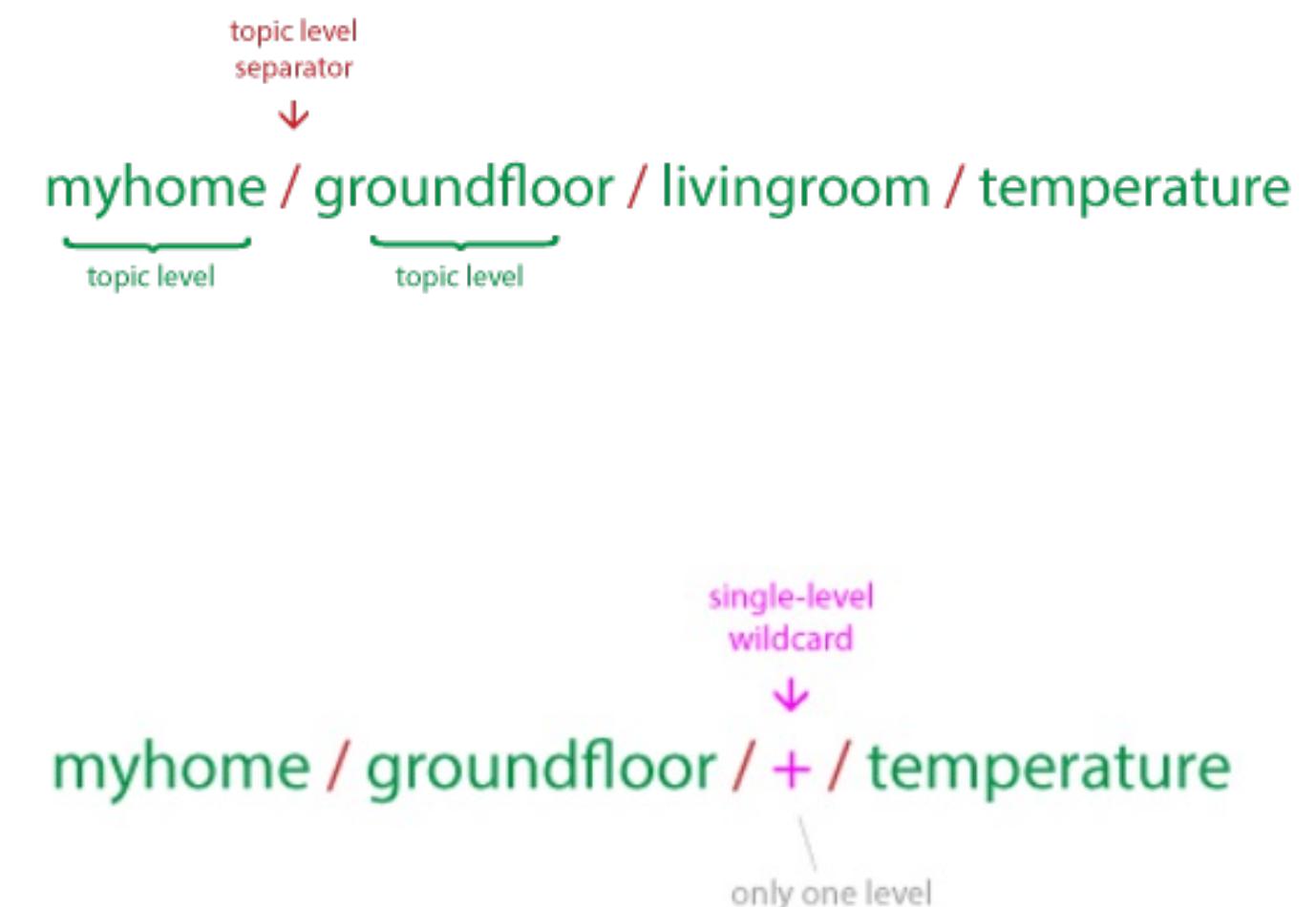
# MQTT: Tópicos

- Um *client* pode assinar um tópico de um nível superior, e receber mensagens destinadas à todos os tópicos de nível inferior.
- Isto é feito com o símbolo **#**.
- Obs.: apenas para *subscribes*! Um *client* só pode realizar um *publish* em tópicos individuais.



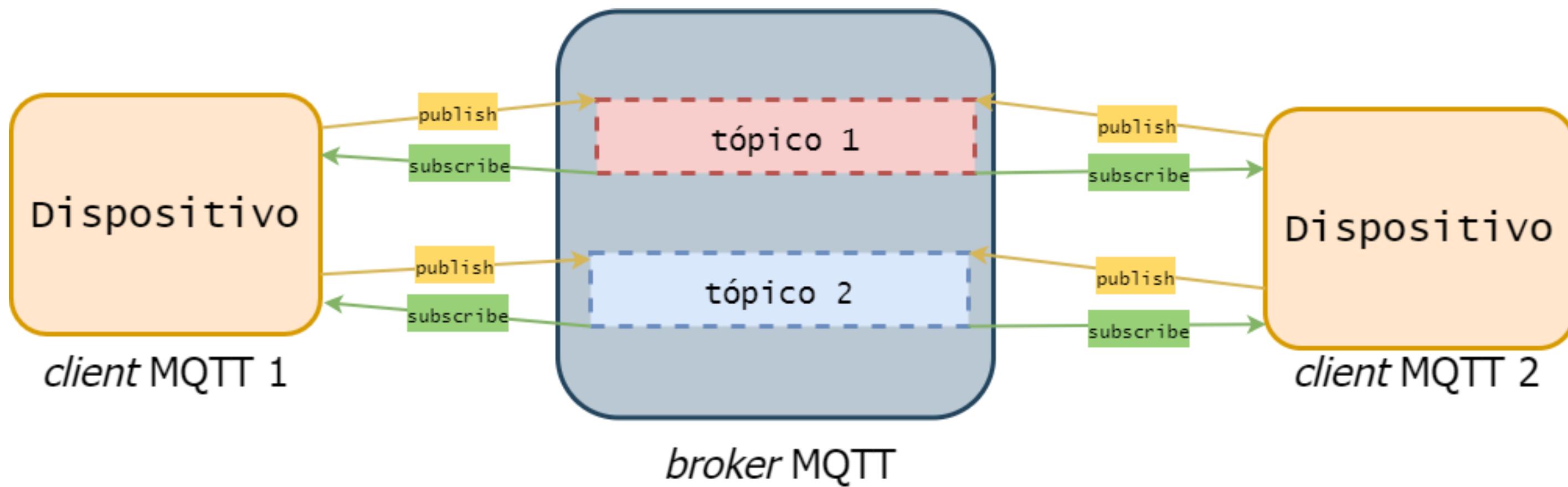
# MQTT: Tópicos

- Nível único: +
- Como o nome sugere, um curinga de nível único substitui um nível de tópico. O símbolo + representa um curinga de nível único em um tópico.
- Qualquer tópico corresponde a um tópico com curinga de nível único se contiver uma *string* arbitrária em vez do curinga.
- No exemplo ao lado, o *client* irá receber os valores dos sensores de temperatura de todos os ambientes do piso térreo.



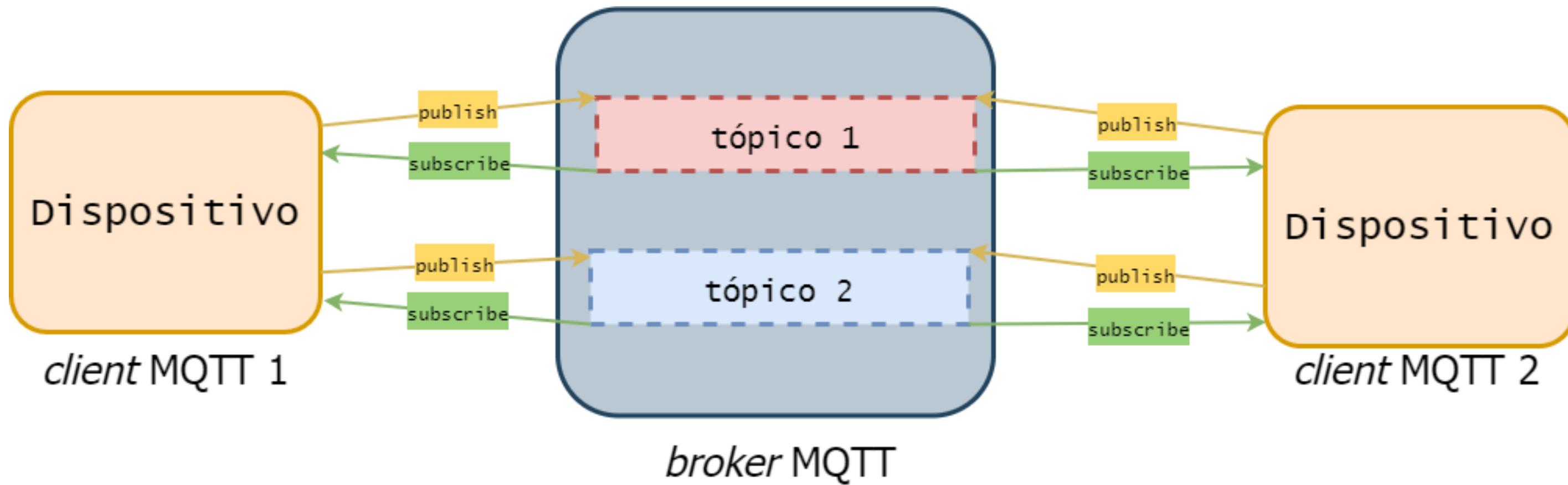
# MQTT: Tópicos

- Com o uso de tópicos, *clients* não precisam ter conhecimento prévio de outros *clients* para trocar informações.
- Isso ajuda a tornar o protocolo mais escalável - não há dependência direta dos dispositivos *clients* que produzem (*publishers*) e os que consomem (*subscribers*) as mensagens.



# MQTT: *Publish*

- Quando conectados ao *broker*, *clients* estão aptos a publicar (*publish*) mensagens em tópicos, destinadas à outros *clients*.
- Cada mensagem de *publish* deve conter uma *string* de tópico, e pode ou não conter *payload* de dados.



## MQTT: *Publish*

- No protocolo MQTT nós temos 3 qualidades de serviço(QoS) e cada conexão com o broker pode especificar qual será utilizada, sendo estas: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez".
  - QoS 0 - No máximo uma vez: Conhecido como *fire and forget* (atirar e esquecer), nesse QoS a mensagem é enviada apenas uma vez e não haverá passos seguintes, dessa forma a mensagem não será armazenada, nem haverá um *feedback* para saber se ela chegou ao destinatário.
  - Esse modo de transferência é o mais rápido, porém o menos seguro já que a mensagem será perdida caso o envio falhe ou o cliente esteja desconectado.

## MQTT: *Publish*

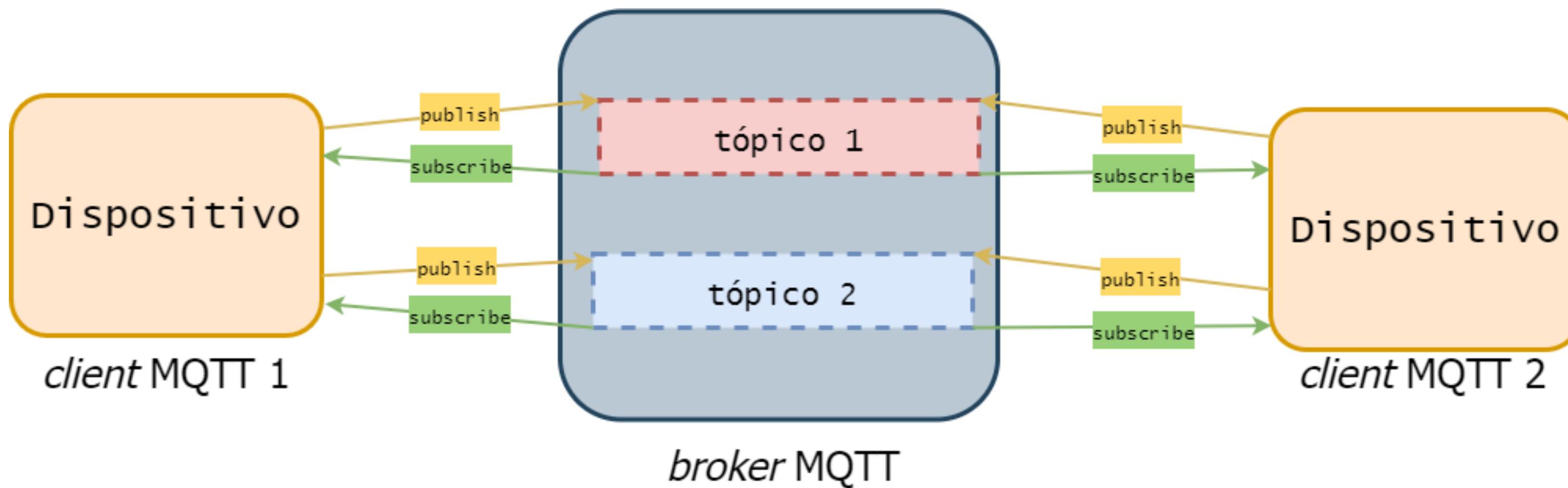
- QoS 1 - Pelo menos uma vez
  - Nesse modo de transferência, a mensagem é entregue pelo menos uma vez, havendo uma espera da recepção de feedback da entrega da mensagem, o chamado **PUBACK**.
  - Não recebendo o **PUBACK**, a mensagem continuará sendo enviado até que haja o feedback. Nesse QoS pode acontecer da mensagem ser enviada diversas vezes e ser processada diversas vezes.
  - Para que haja o envio da mensagem mais de uma vez, a mensagem precisa ser armazenada. Ela será excluída após ter recebido o *feedback* de confirmação do envio.

## MQTT: *Publish*

- QoS 2 - Exatamente uma vez
  - Nesse modo de transferência, a mensagem é entregue exatamente uma vez, necessitando que a mensagem seja armazenada localmente no emissor e no receptor até que seja processada.
  - Para garantir a segurança desse QoS é necessário o envio de 2 pares de *request-response* (chamado de *four-part handshake*), onde temos o envio da mensagem([PUBLISH](#)), a resposta de recepção([PUBREC](#)), o aviso do recebimento do [PUBREC](#)([PUBREL](#)) e a confirmação de que o processo foi concluído e pode ser feita a exclusão([PUBCOMP](#)).
  - Após o recebimento do [PUBREL](#), o *receiver* pode excluir a mensagem e ao *sender* receber o [PUBCOMP](#) ele poderá excluir a mensagem.

# MQTT: *Subscribe*

- Quando um *client* assina (*subscribe*) um tópico, mensagens publicadas no *broker* são distribuídas para os *clients* que o assinaram.
- O processo de *subscribe* de um tópico pode ser desfeito pelos *clients* (*unsubscribe*) sem perda de conexão, para parar de receber mensagens do *broker* neste tópico.



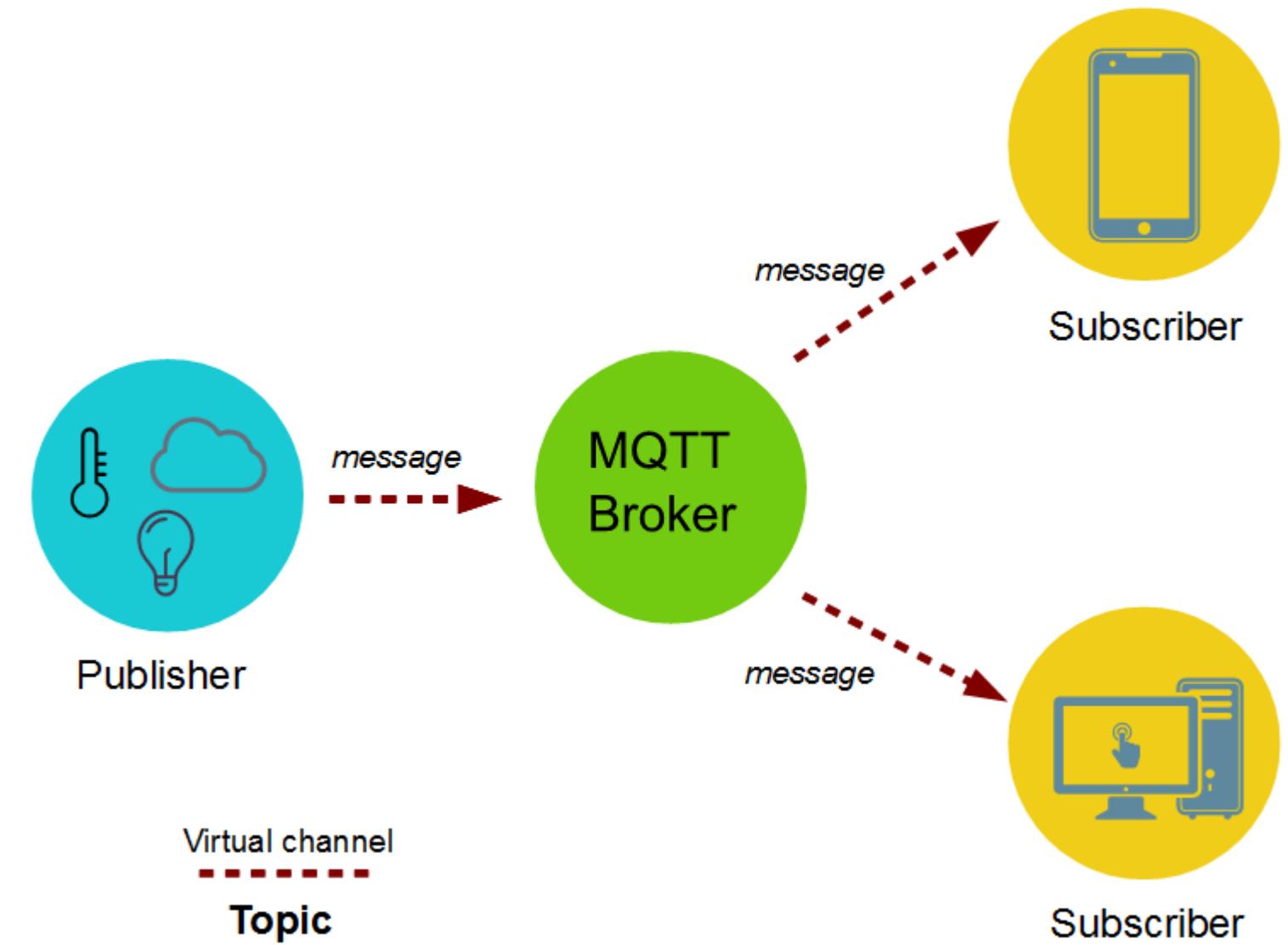
# MQTT: Resumo dos conceitos fundamentais

- No protocolo MQTT, dispositivos (*clients*) se conectam à um servidor (*broker*) responsável por pelo controle de fluxo das mensagens.
- Dispositivos enviam (*publish*) mensagens compostas por um *payload* (dados) e uma *string* de tópico, que é identificado pelo *broker* e atua como “filtro” para designar tais mensagens aos demais *clients*.
- Dispositivos que assinaram (*subscribe*) tópicos para os quais mensagens forem designadas terão estas mensagens encaminhadas pelo *broker*.

## Comunicação segura no MQTT

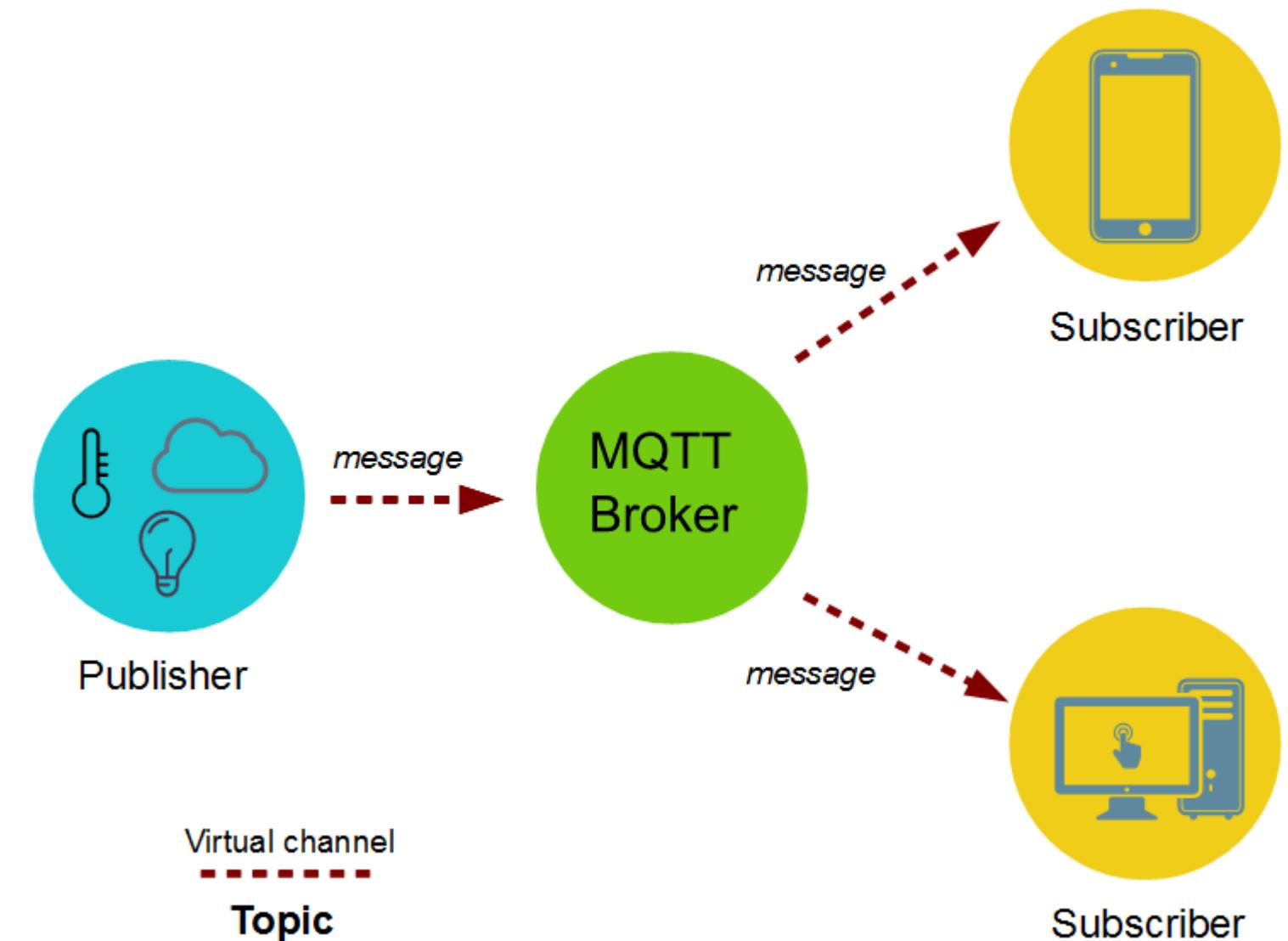
O servidor escuta nas seguintes portas:

- 1883: MQTT, não criptografado, não autenticado
- 1884 : MQTT, não criptografado, autenticado
- 8883 : MQTT, criptografado, não autenticado
- 8884: MQTT, criptografado, certificado de cliente necessário
- 8885 : MQTT, criptografado, autenticado
- 8886: MQTT, criptografado, não autenticado
- 8887: MQTT, criptografado, certificado de servidor expirado deliberadamente



## MQTT Autenticação:

- No nível do aplicativo, o protocolo MQTT fornece nome de usuário e senha para autenticação.
- No entanto, se essas informações não forem criptografadas ou em hash (por implementação ou TLS), a senha será enviada em texto simples.
- Usar as portas com criptografia



## MQTT certificado:

- O broker mosquito disponibiliza um certificado no formato crt.
- No site [www.test.mosquito.org](http://www.test.mosquito.org)
- Para conectar na porta 8883 é necessário incluir este arquivo crt para poder publicar e subscrever os tópicos.
- Além disso, pode-se incluir dados de usuário e senha.

-----BEGIN CERTIFICATE-----

```
MIIEAzCCAuugAwIBAgIUBY1hICGvdj4NhBXkZ/uLUZNILAwxDQYJKoZIhvcNAQELBQ
AwgZAxCzAJBgNVBAYTAkdCMRcwFQYDVQQIDA5Vbml0ZWQgS2luZ2RvbTEOMAw
GA1UEBwwFRGVyYnkxEjAQBgNVBAoMCU1vc3F1aXR0bzELMAkGA1UECwwCQ0Ex
FjAUBgNVBAMMDW1vc3F1aXR0by5vcmcxHzAdBgkqhkiG9w0BCQEWEHJvZ2VyQGF
0Y2hvby5vcmcwHhcNMjAwNjA5MTEwNjM5WhcNMzAwNjA3MTEwNjM5WjCBkDELMA
kGA1UEBhMCR0IxFzAVBgNVBAgMDIVuaXRIZCBLaW5nZG9tMQ4wDAYDVQQHDAV
EZXJieTESMBAGA1UECgwJTW9zcXVpdHRvMQswCQYDVQLDAJDQTEWMBQGA1
UEAwNbW9zcXVpdHRvLm9yZzEfMB0GCSqGSIb3DQEJARYQcm9nZXJAYXRjaG9v
Lm9yZzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAME0HKmlzfTOwk
KLT3THHe+ObdizamPgUZmD64Tf3zJdNeYGYn4CEXbyP6fy3tWc8S2boW6dzh8SdFf
9uo320GJA9B7U1FWTe3xda/Lm3JFfaHjkWw7jBwcauQZjpGINHapHRIpiCZsquAthOgx
W9SgDgYIGzEAs06pkEFiMw+qDfLo/sxFKB6vQIFekMeCymjLCbNwPJyqyhFmPWwio/P
DMruBTzPH3cioBnrJWKXc3OjXdLGFJOfj7pP0j/dr2LH72eSvv3PQQFI90CZPFhrCUcR
HSSxoE6yjGOdnz7f6PveLIB574kQORwt8ePn0yidrTC1ictikED3nHYhMUOUCAwEAAa
NTMFEwHQYDVR0OBYEFPVV6xBUFPiGKDyo5V3+Hbh4N9YSMB8GA1UdlwQYMB
aAFPVV6xBUFPiGKDyo5V3+Hbh4N9YSMA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZI
hvcNAQELBQADggEBAGa9ks21N70ThM6/Hj9D7mbVxKLbjVWe2TPsGfbI3rEdfZ+OK
RZ2j6AC6r7jb4TZ03dzF2p6dgbriU71Y/4K0TdzljRj3cQ3KSm41JvUQ0hZ/c04iGDg/xWf
+pp58nfPAYwuerruPNWmlStWAXf0UTqRtg4hQDWBuUFDJTuWuuBvEXudz74eh/wKs
Mwf1HFvfyj5Z0iMDU8PUDepjVoLOCue9ashlS4EB5IECdSR2TItnAlilwimx839LdUdRud
afMu5T5Xma182OC0/u/xRIEm+tvKGGMfFcN0piqVI8OrSPBglIb+1IKJEm/XriWr/Cq4h/Jf
B7NTsezVslgkBaoU=-----END CERTIFICATE-----
```

## MQTT Autenticação:

- exemplo **ESP32\_MQTT\_SSL**

```
#include <Arduino.h>
#include <PubSubClient.h>
#include <WiFiClientSecure.h>

//const char* ssid = "WIFI_NAME";
//const char* password = "WIFI_PASSWORD";
const char* ssid = "WLL-Inatel";
const char* password = "inatelsemfio";

#if defined(ESP32)
#include <WiFi.h>

// Root CA
const static char* test_root_ca PROGMEM = \
| | "-----BEGIN CERTIFICATE-----\n" \
|MIIEAzCCAuugAwIBAgIUBY1h1CGvdj4NhBXkZ/uLUZNILAwxDQYJKoZIhvcNAQEL\n" \
|BQAwgZAxCAJBgNVBAYTAkdCMRcwFQYDVQQIDA5Vbml0ZWQgS2luZ2RvbTEOMAwG\n" \
|A1UEBwwFRGVyYnkxEjAQBgNVBAoMCU1vc3F1aXR0bzELMAkGA1UECwwCQ0ExFjAU\n" \
|BgNVBAMMDW1vc3F1aXR0by5vcmcxBzAdBkgqhkig9w0BCQEWEHJvZ2VyQGF0Y2hv\n" \
|by5vcmcwbHcNMjAwNjA5MTEwNjM5WhcNMzAwNjA3MTEwNjM5WjCBkDELMAkGA1UE\n" \
|BhMCR0IXFzAVBgvNVBAgMDlVuaXR1ZCBLaW5nZG9tMQ4wDAYDVQQHDAVEZXJieTES\n" \
|MBAGA1UECgwJTW9zcXVpdHRvMQswCQYDVQQLDAJDQTEWMBQGA1UEAwNBW9zcXVp\n" \
|dHRvLm9yZzEfMB0GCSqGSIb3DQEJARYQcm9nZXJAYXRjaG9vLm9yZzCCASIwDQYJ\n" \
|KoZIhvcNAQEBBQADggEPADCCAQoCggEBAME0HKmIzfTOwkKLT3THHe+ObdizamPg\n" \
|UZmD64Tf3zJdNeYGYn4CEXbyP6fy3twc8S2bow6dzrH8sdFf9uo320GJA9B7U1FW\n" \
|Te3xda/Lm3JFfaHjkWw7jBwcauQZjpGINHapHRlpiCZsquAthOgxW9SgDgYlGzEA\n" \
|s06pkEFiMw+qDfLo/sxFKB6vQ1FekMeCymjLCbNwPJyqyhFmPWwio/PDMruBTzPH\n" \
|3cioBnrJWKXc30jXdLGFjOfj7pP0j/dr2LH72eSvv3PQQFl90CZPFhrCUcRHSSxo\n" \
|E6yjG0dnz7f6PveLIB574kQORwt8ePn0yidrTC1ictikED3nHYhMUOUCAwEAAANT\n" \
|MFEwHQYDVR00BBYEFPVV6xBUFPiGKDyo5V3+Hbh4N9YSMA8GA1UdIwQYMBaAFPVV\n" \
|6xBUFPiGKDyo5V3+Hbh4N9YSMA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQEL\n" \
|BQADggEBAGa9ks21N70ThM6/Hj9D7mbVxKLBjvWe2TPsGfb13rEDfZ+OKRZ2j6AC\n" \
|6r7jb4TZ03dzF2p6dgbrlu71Y/4K0TdzIjRj3cQ3KSm41JvUQ0hZ/c04iGDg/xwf\n" \
|+pp58nfPAYwuerruPNWmlStWAXf0UTqRtg4hQDWBuUFDJTuWuuBvEXudz74eh/wK\n" \
|sMwfuf1HFvjy5Z0iMDU8PUDepjVolOCue9ashls4EB5IECdSR2TItnAIiIwimx839\n" \\\
```

## MQTT Autenticação:

- No MQTTX é necessário incluir o arquivo .crt do certificado para conectar ao broker.

The screenshot shows the MQTTX configuration interface. The top section, 'General', contains fields for Name (msoquitossal), Client ID (mqtx\_ea448157), Host (mqtts:// test.mosquitto.org), Port (8883), Username, Password, SSL/TLS (on), and SSL Secure (on). The 'Certificate' section shows 'Self signed' selected. The bottom section, 'Certificates', shows CA File (C:\Users\mafra\Downloads\test.crt) and Client Certificate File (empty).

**General**

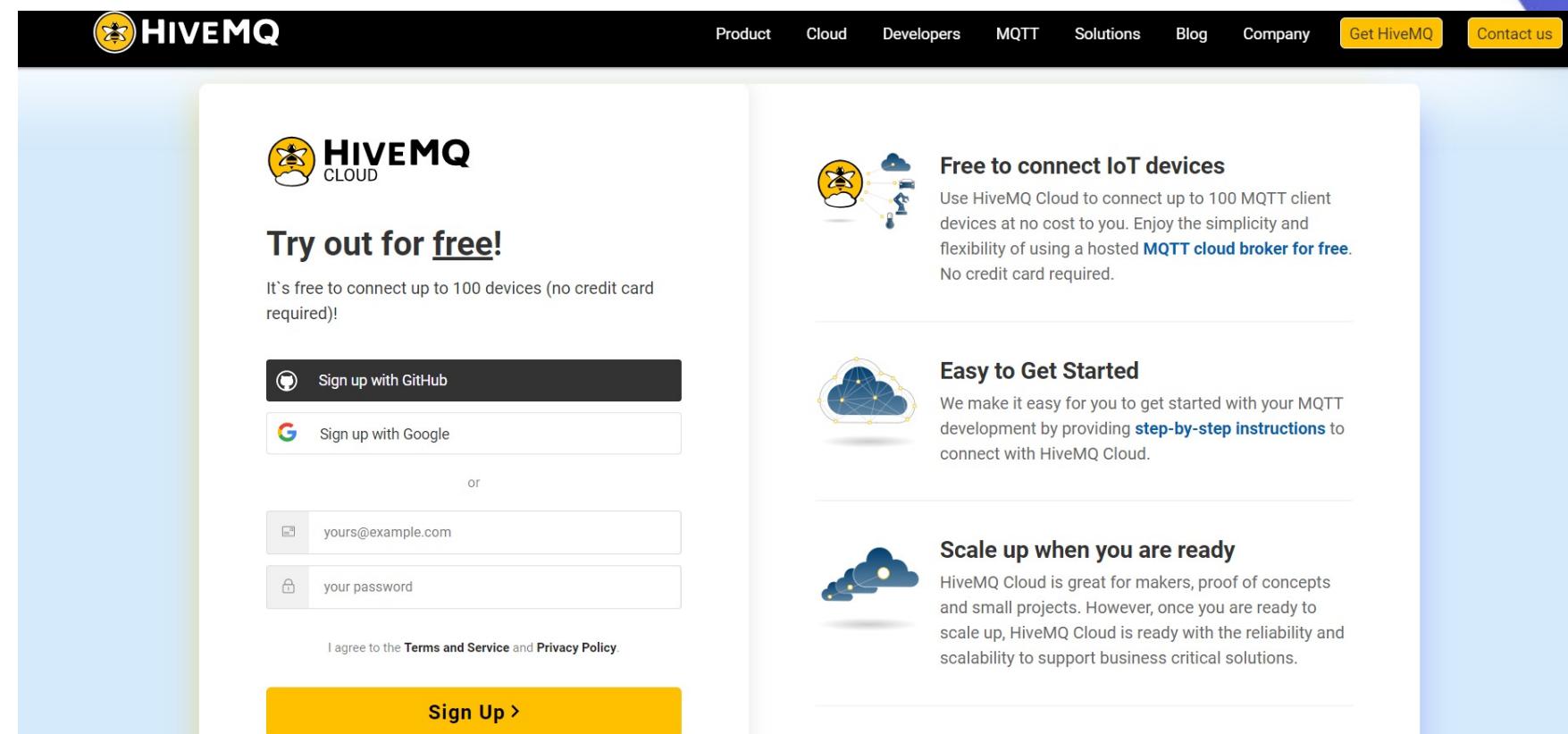
- \* Name: msoquitossal
- \* Client ID: mqtx\_ea448157
- \* Host: mqtts:// test.mosquitto.org
- \* Port: 8883
- Username:
- Password:
- SSL/TLS:
- SSL Secure:  ⓘ
- Certificate:  CA signed server  Self signed

**Certificates**

- CA File: C:\Users\mafra\Downloads\test.crt
- Client Certificate File:

## Broker MQTT com autenticação

- <https://www.hivemq.com/mqtt-cloud-broker/>
- Permite criar uma conta grátis através do site, Google e GitHub.
- A comunicação é feita de forma segura usando usuário e senha. Estes dados são criados ao fazer cadastro no aplicativo.



## Broker MQTT com autenticação

- O site cria uma URL exclusiva para o usuário;
- Porta 8883.
- É possível criar mais de uma credencial para os usuários.

The screenshot shows the HiveMQ Cloud interface with two main sections:

- Cluster URL:** f594e79fabf8464b88d8999e6cecc235.s1.eu.hivemq.cloud EDIT
- Port:** 8883 EDIT

**Set up credentials for your IoT devices**

Define the credentials that your MQTT clients can use to connect to your HiveMQ Cloud cluster.

Please visit the [HiveMQ documentation](#) for examples on how to use the credentials to connect an MQTT client to your cluster.

(All fields are mandatory)

Username	samuel1586	Actions
Password	*****	<span style="border: 1px solid red; padding: 2px;">DELETE</span>

**Active MQTT Credentials**

These credentials allow MQTT clients to publish and subscribe to your HiveMQ Cloud cluster.

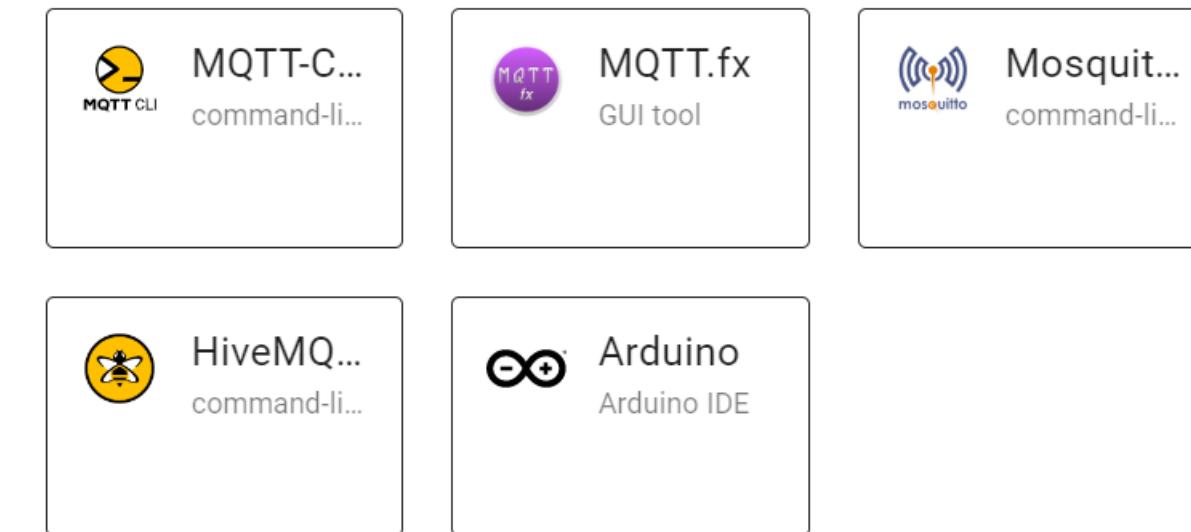
Username: samuel1586 | Password: \*\*\*\*\* | Actions: DELETE

**ADD**

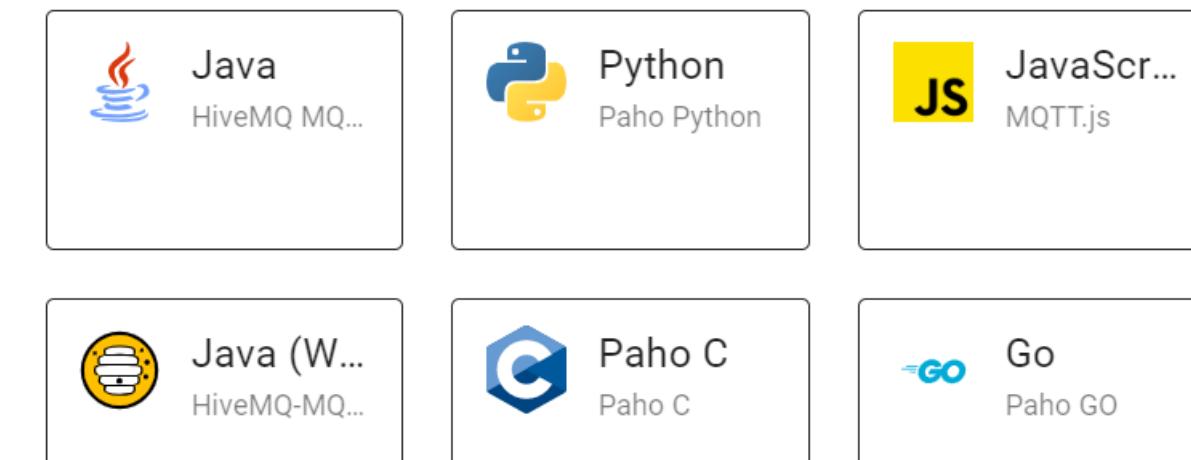
## Broker MQTT com autenticação

- No site é possível obter vários exemplos para diferentes ferramentas e linguagens de programação.
- Infelizmente o código em Arduino não funciona na ESP32 apenas na ESP8266.

### Tools



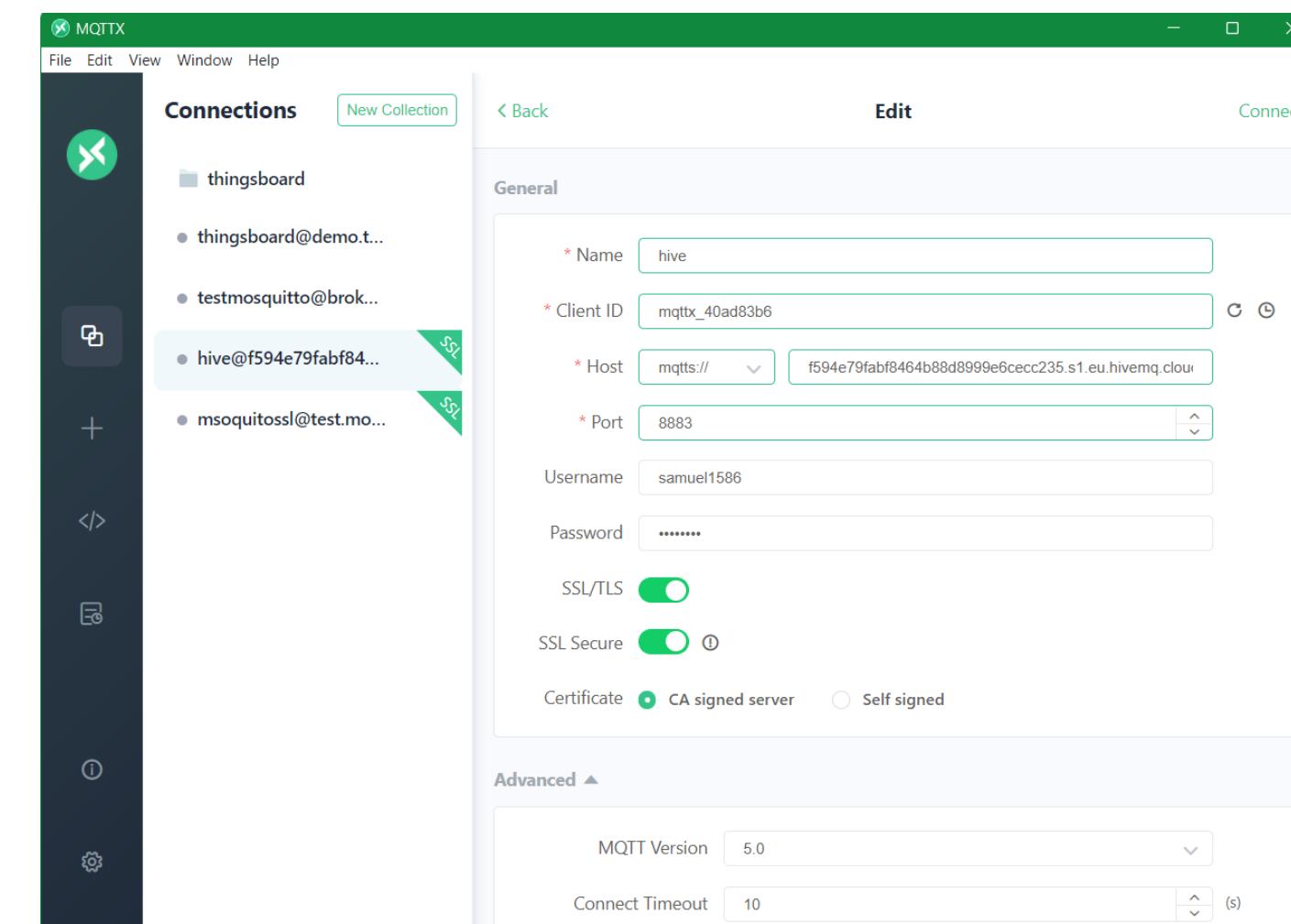
### Programming Languages

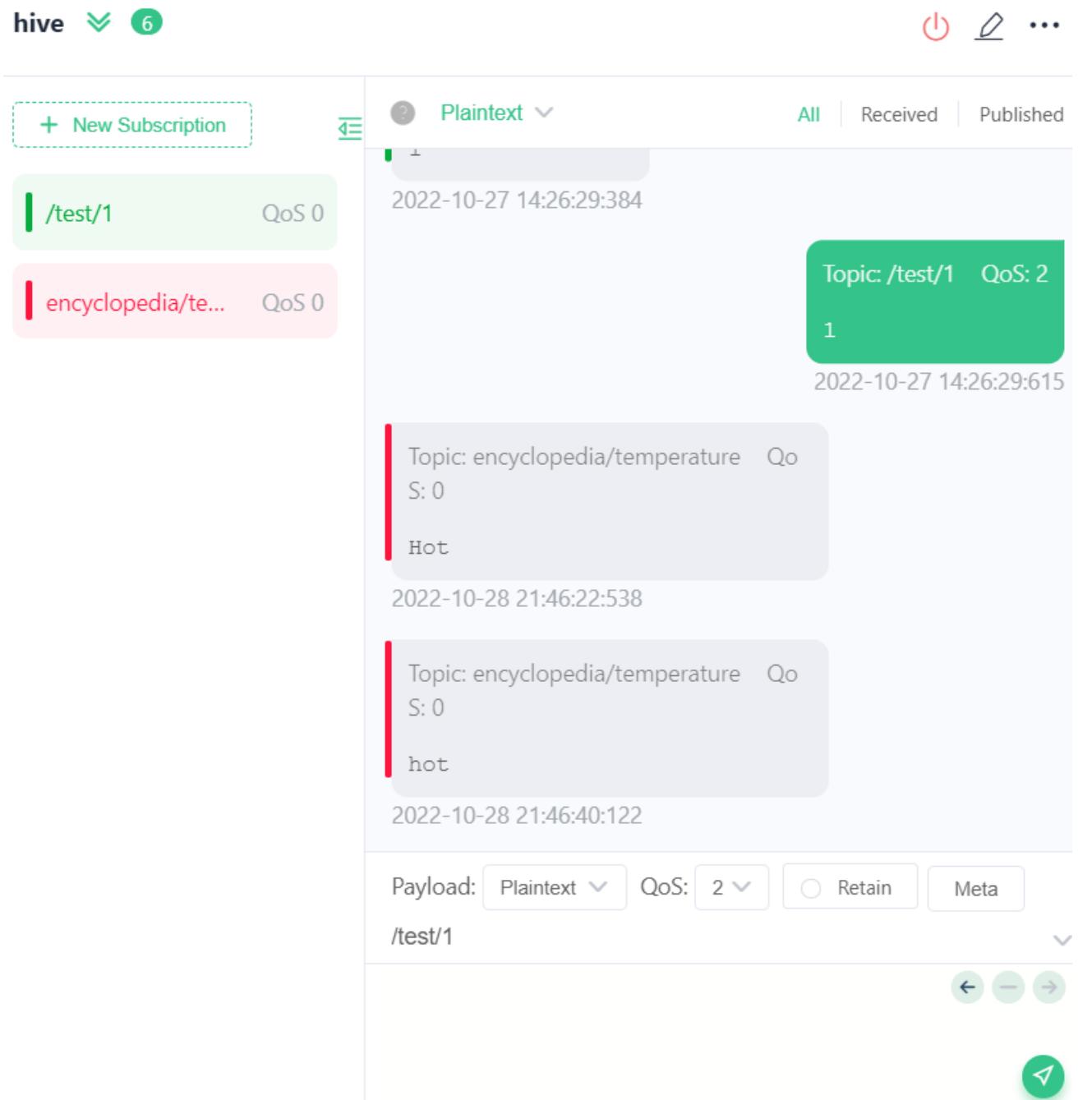


## Broker MQTT com autenticação

- Exemplo de autenticação em um arquivo de Publisher mqtt python;
- Para conectar ao broker pelo MQTTX é necessário incluir os dados iguais ao publicador em python;
- Endereço do broker
- Usuário
- Senha
- Porta 8883
- SSL/TLS marcado.

```
# enable TLS for secure connection
client.tls_set(tls_version= mqtt.client.ssl.PROTOCOL_TLS)
# set username and password
client.username_pw_set("usuário", "senha")
# connect to HiveMQ Cloud on port 8883 (default for MQTT)
client.connect("f594e79fabf8464b88d8999e6cecc235.s1.eu.hivemq.cloud", 8883)
```



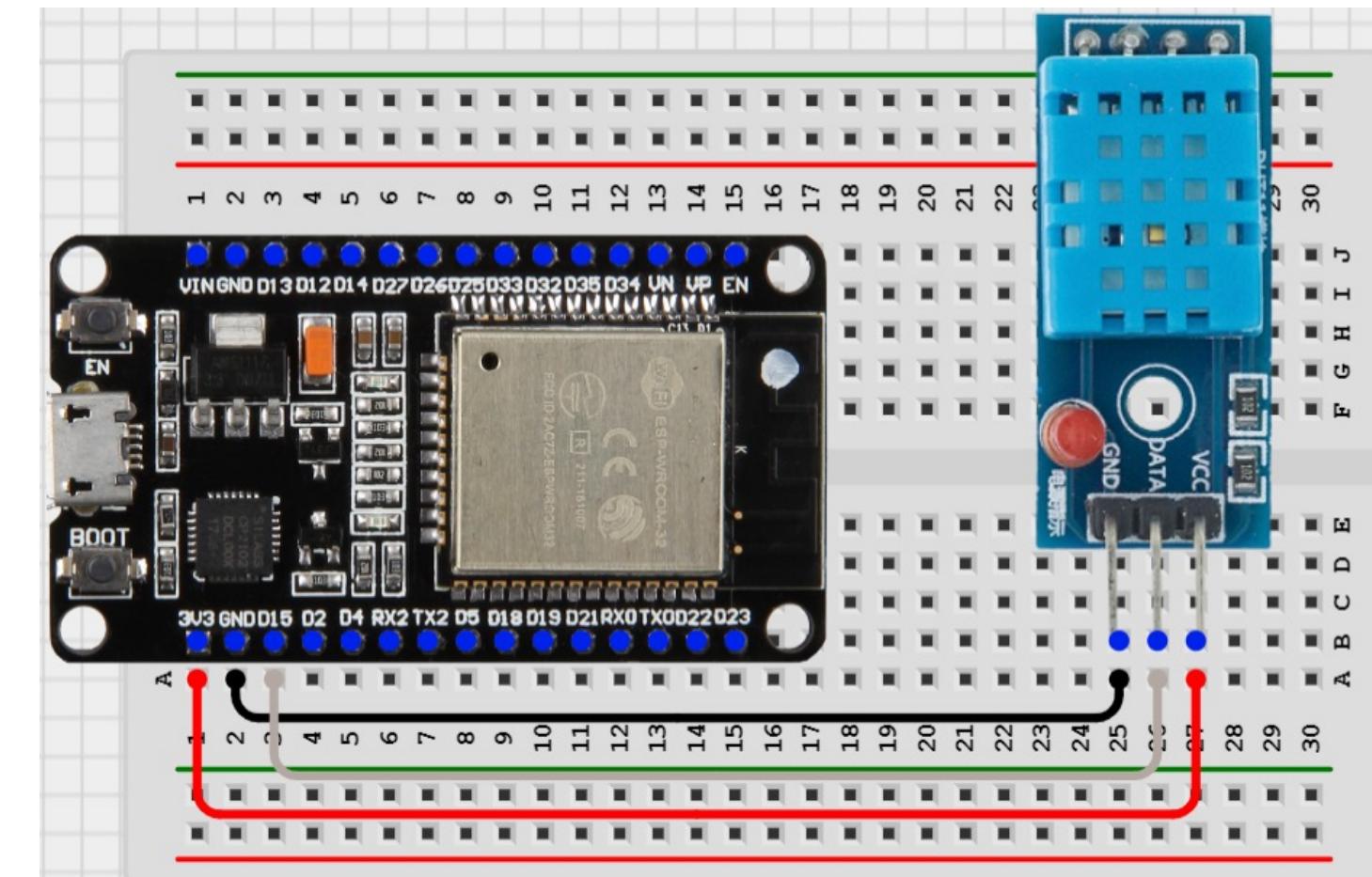


CONNACK received with code Success.

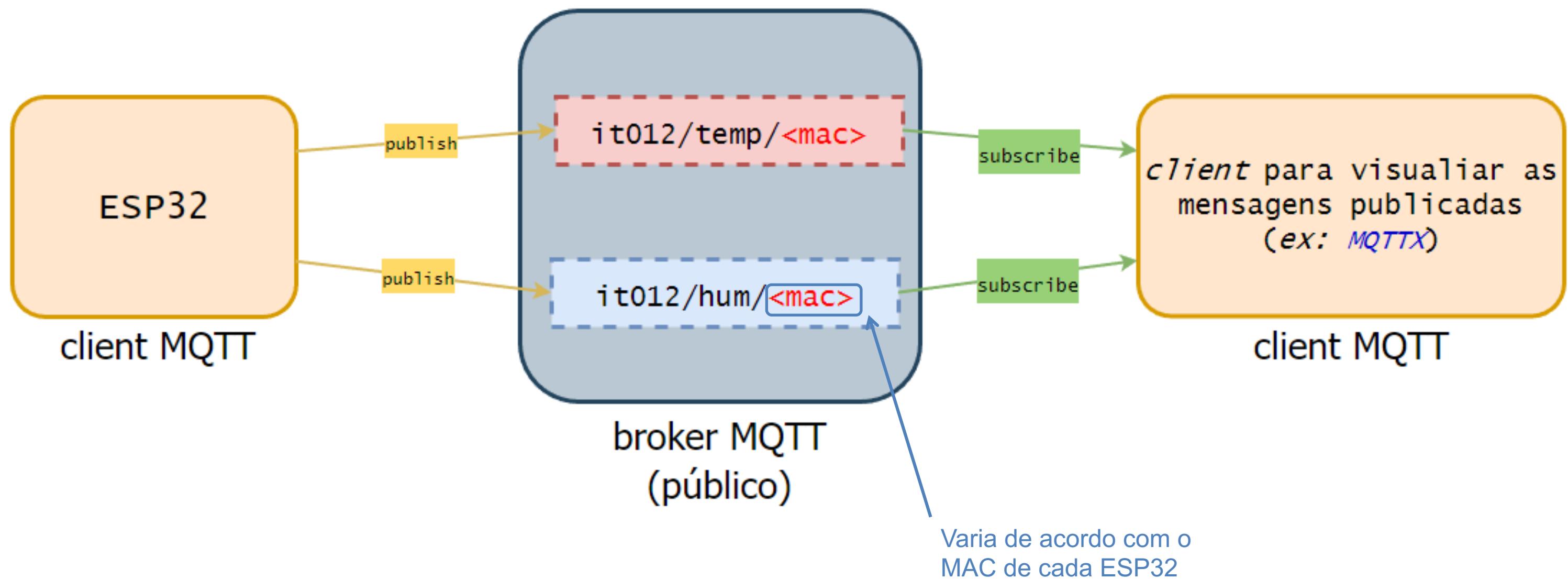
Subscribed: 1 [`<paho.mqtt.reasoncodes.ReasonCodes object at 0x03AB2D30>`]  
encyclopedia/temperature 1 b'hot'  
mid: 2

## Exemplo: MQTT (*Publish*)

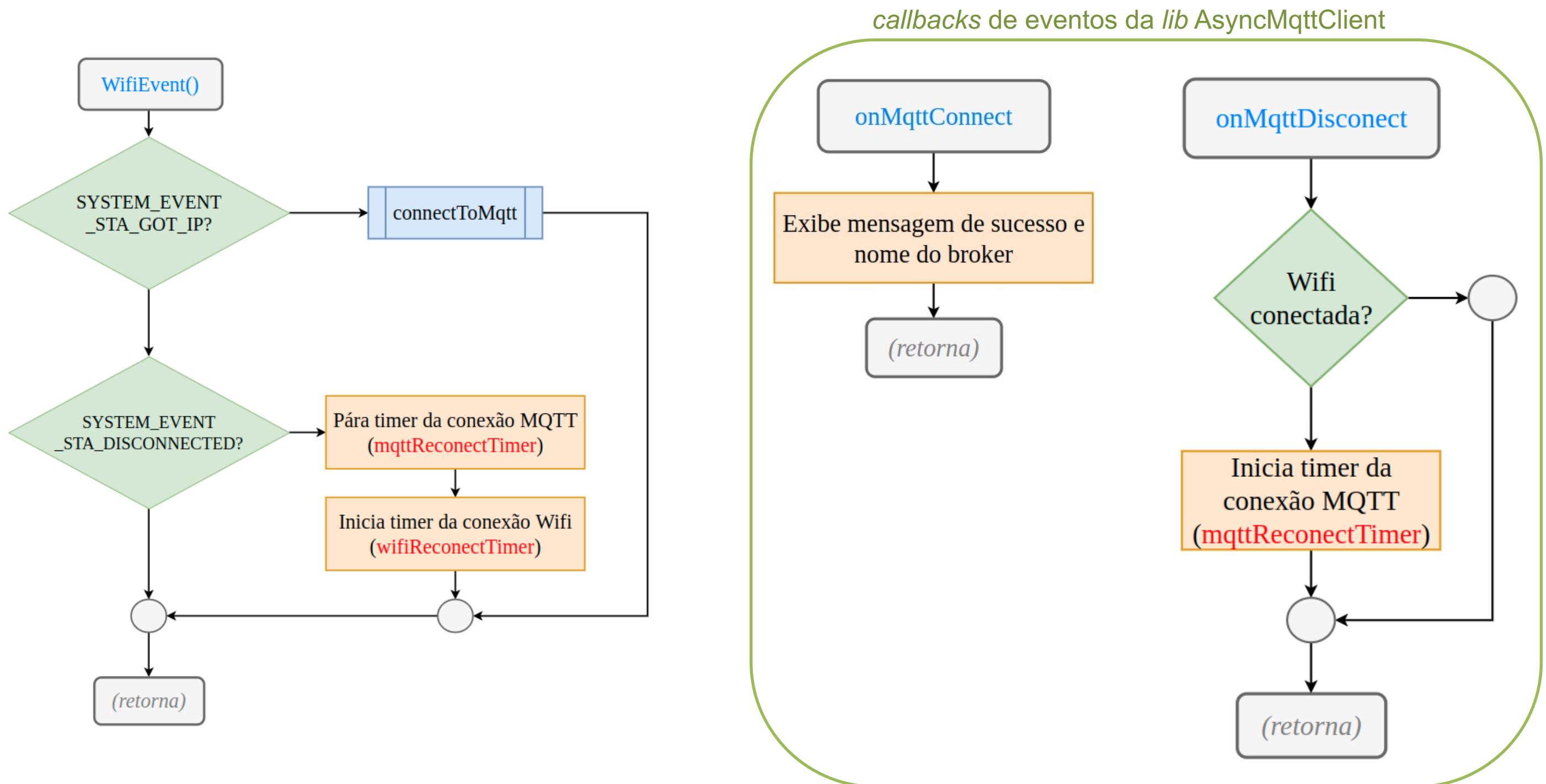
- Abra a pasta: exemplo\_mqtt\_publish
- Mesma montagem do exemplo\_dht11.
- Neste exemplo:
  - O ESP32 se conecta à uma rede Wifi e à um broker MQTT;
  - Realiza leituras de temperatura e umidade do DHT11;
  - Publica as leituras de temperatura em um tópico (it012/temp/<mac>);
  - Publica as leituras de umidade em um tópico (it012/hum/<mac>);
  - <mac> é parte do endereço MAC único de cada modulo, usado aqui para gerar um nome de tópico MQTT único para cada dispositivo.



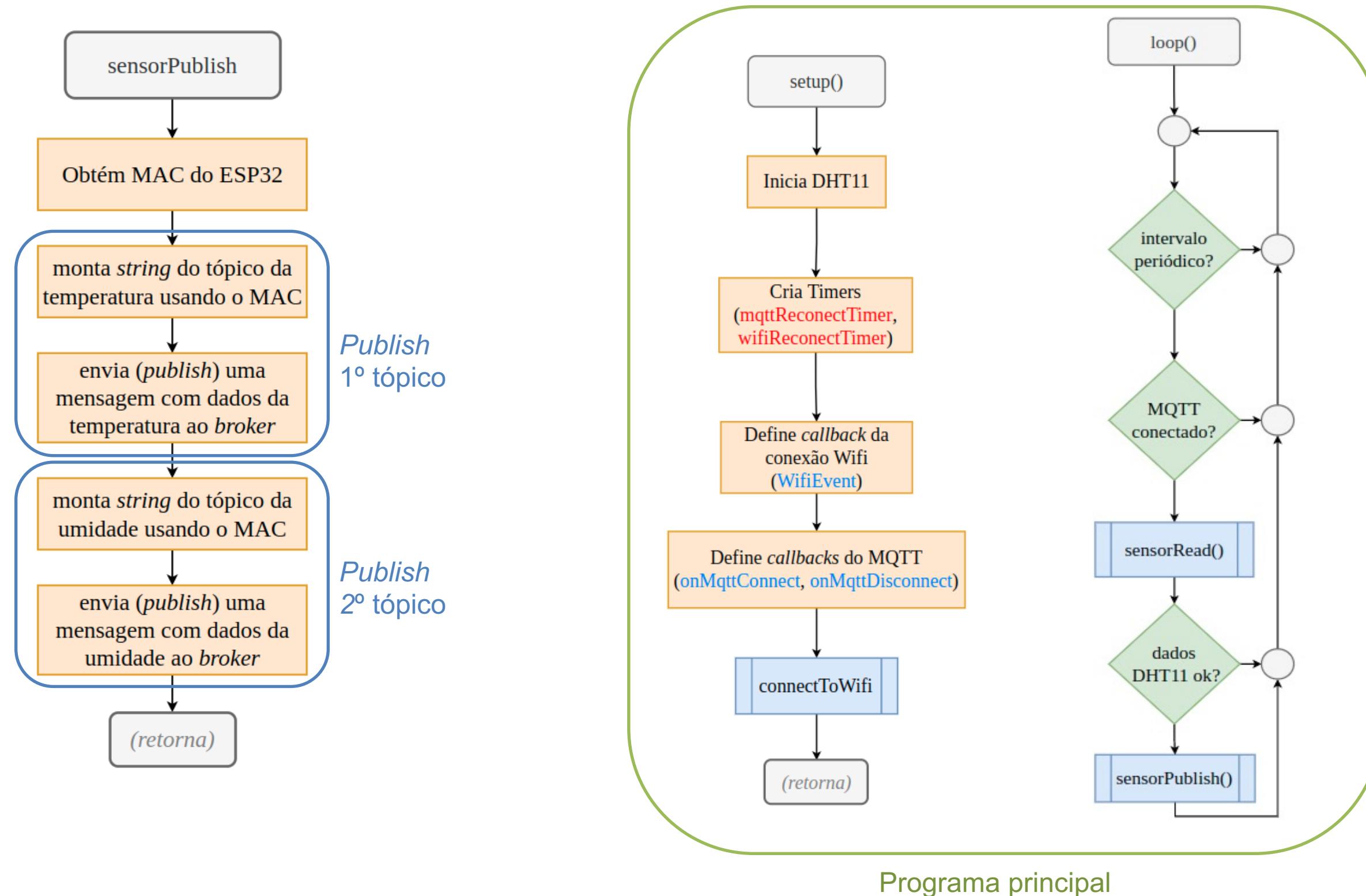
## Exemplo: MQTT (*Publish*)



# Exemplo: MQTT (Publish)



# Exemplo: MQTT (*Publish*)

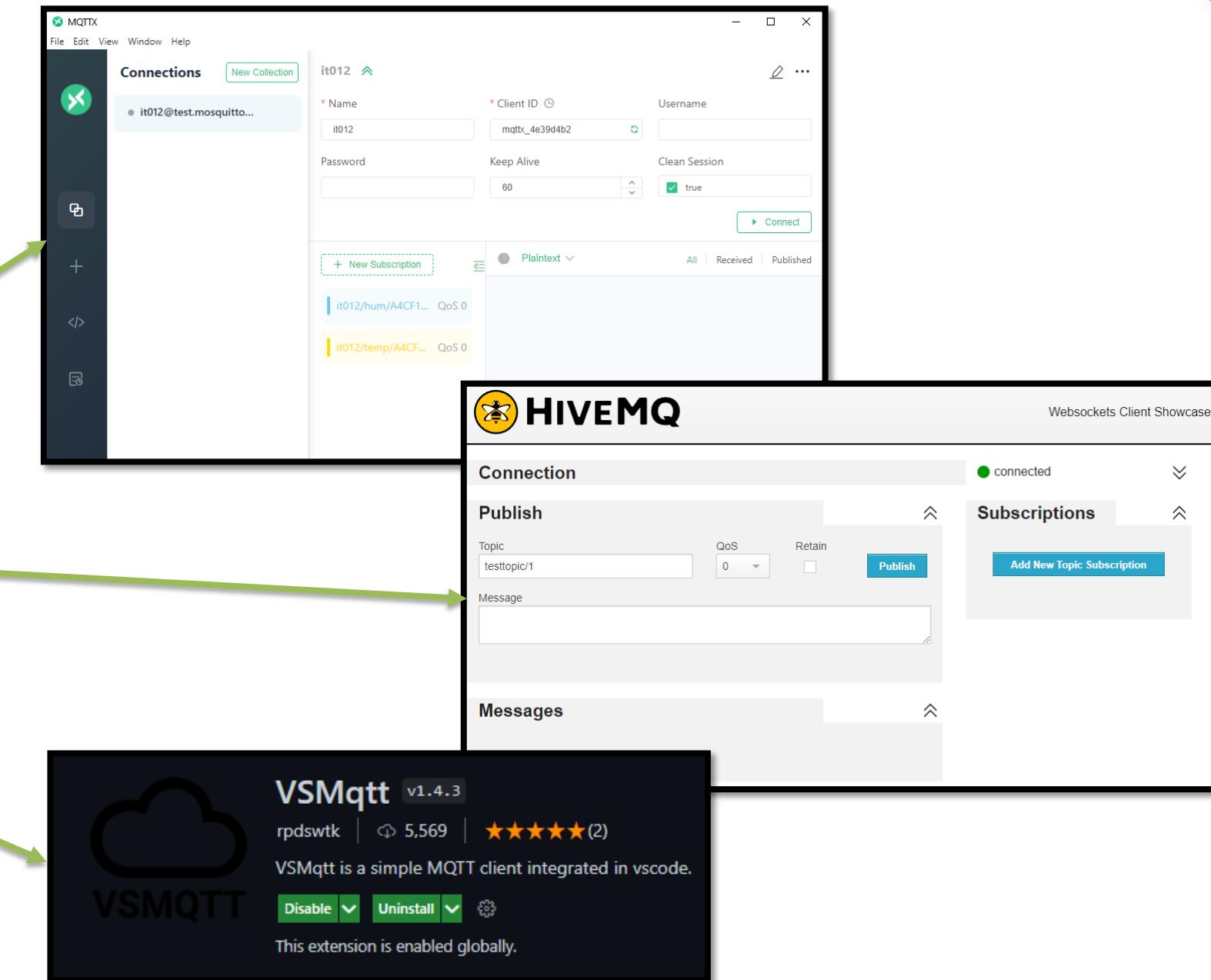


# MQTT – configuração de *clients*

- Para verificar as mensagens sendo enviadas pelo ESP32, criaremos um segundo *client*, que irá assinar os tópicos gerados.

- Há vários formas de se gerar *clients* MQTT:

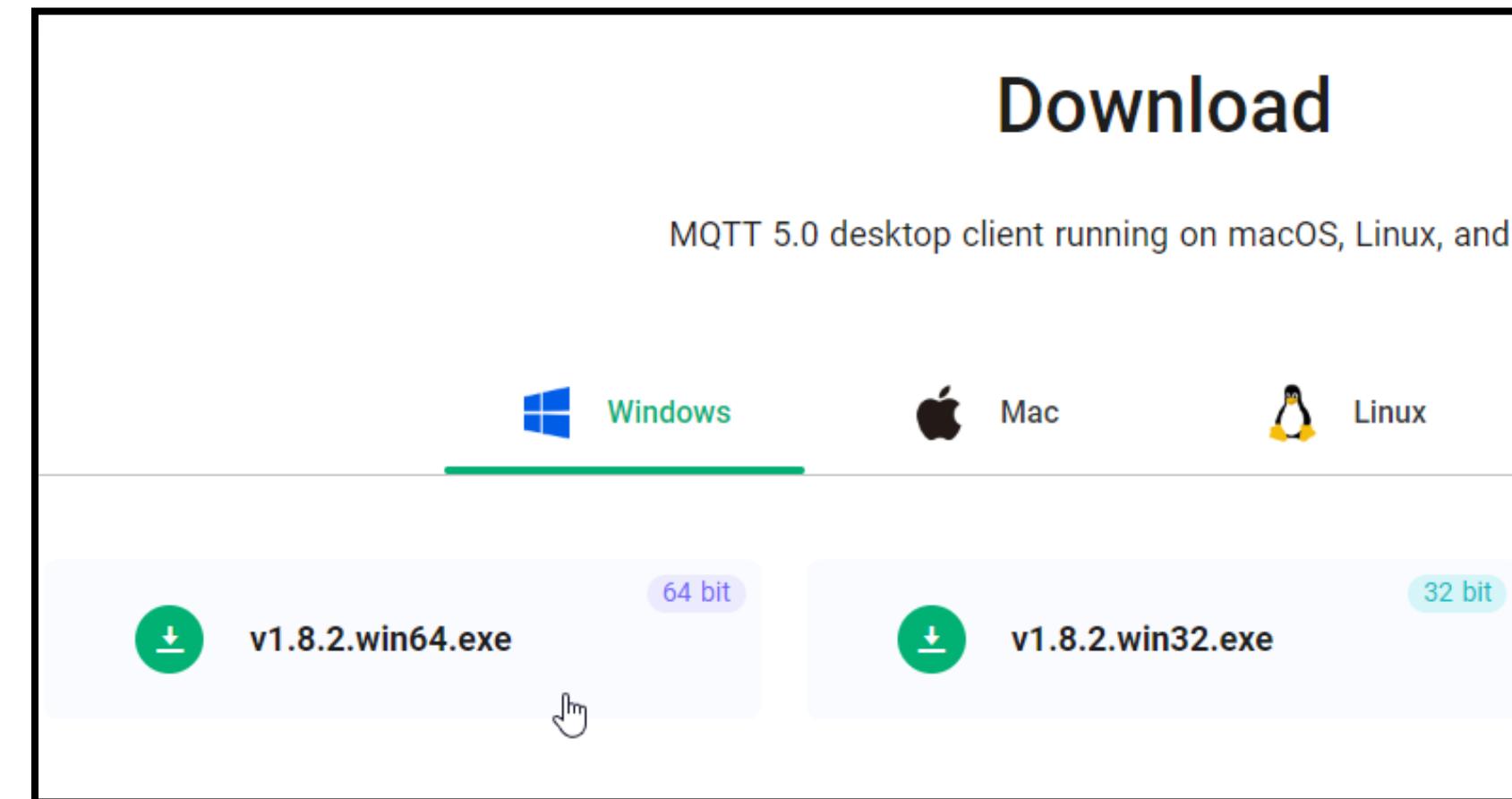
- Executáveis como [MQTT X](#) e [MQTT.fx](#);
- Websockets MQTT: [HiveMQ](#) e [EMQX](#);
- Extensão do VSCode ([VSMQTT](#)).



# MQTT – configuração de *clients*

Usaremos o aplicativo MQTTX como *client*.

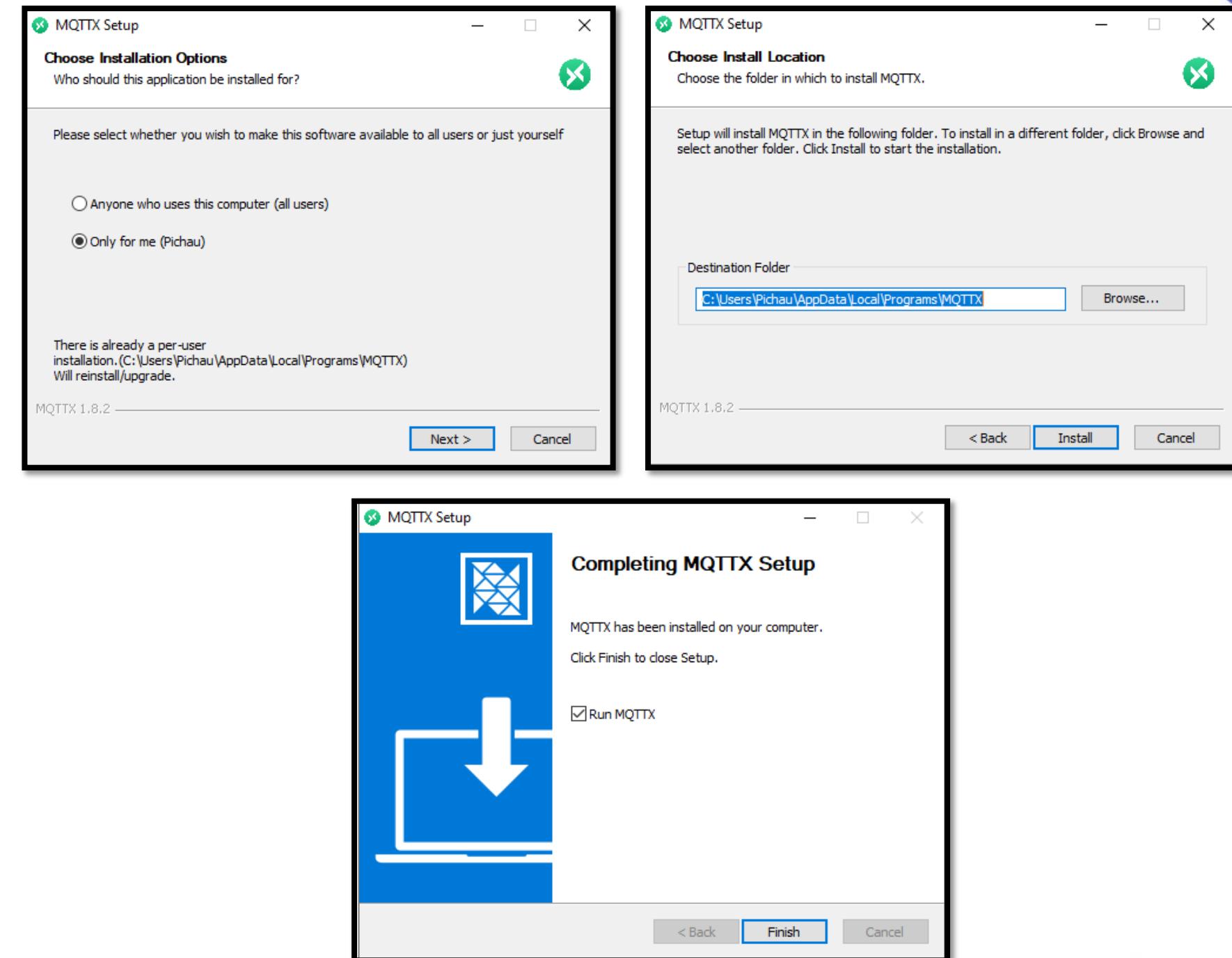
- Baixe-o em <https://mqtx.app/>.  
Há versões para Windows, Linux e Mac.
- Instale o programa com as configurações padrão.
- Na tela principal, clique em “New Connection”.



# MQTT – configuração de *clients*

Usaremos o aplicativo MQTTX como client.

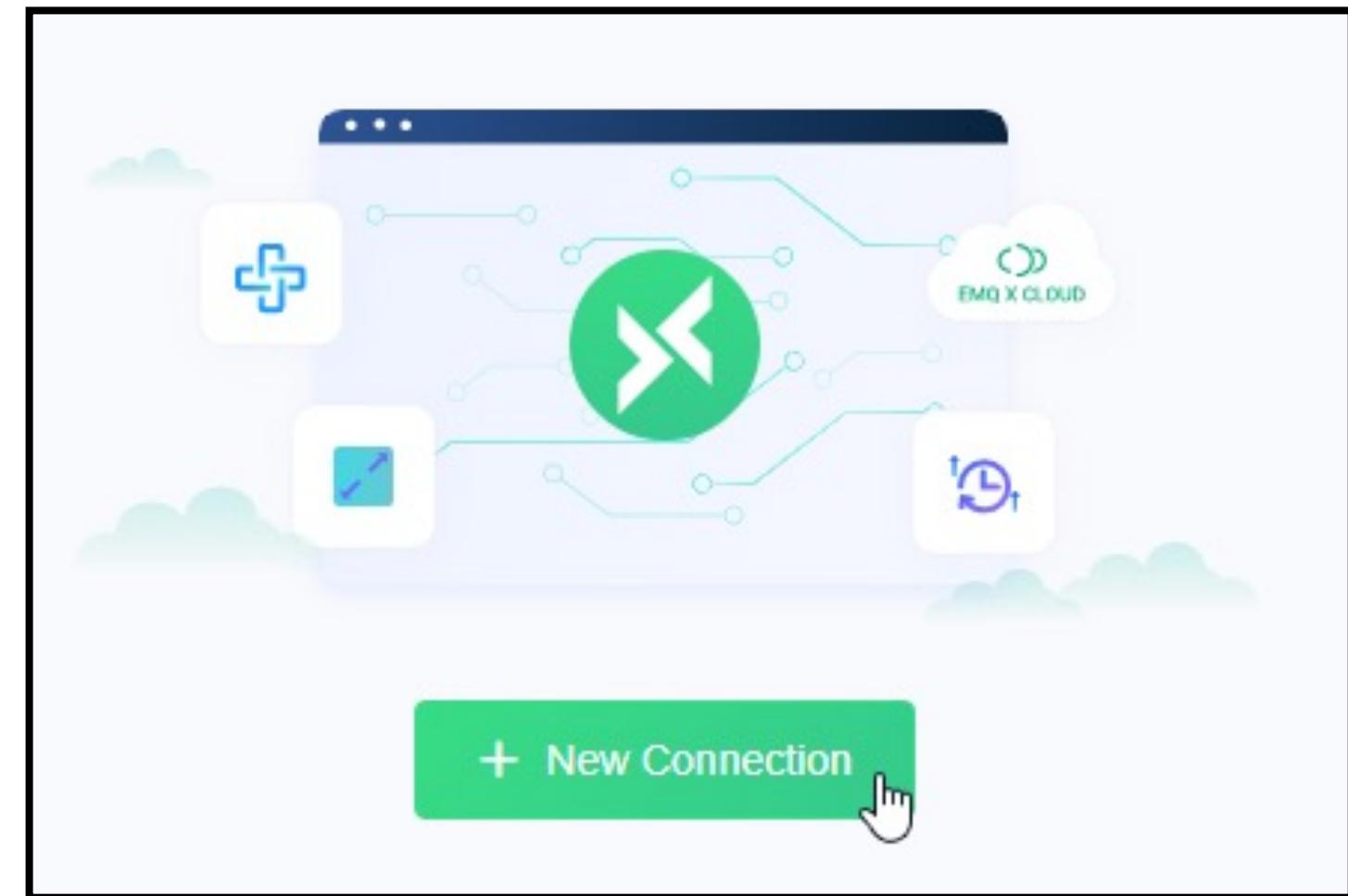
- Baixe-o em <https://mqttx.app/>. Há versões para Windows, Linux e Mac.
- Instale o programa com as configurações padrão.
- Na tela principal, clique em “New Connection”.



# MQTT – configuração de *clients*

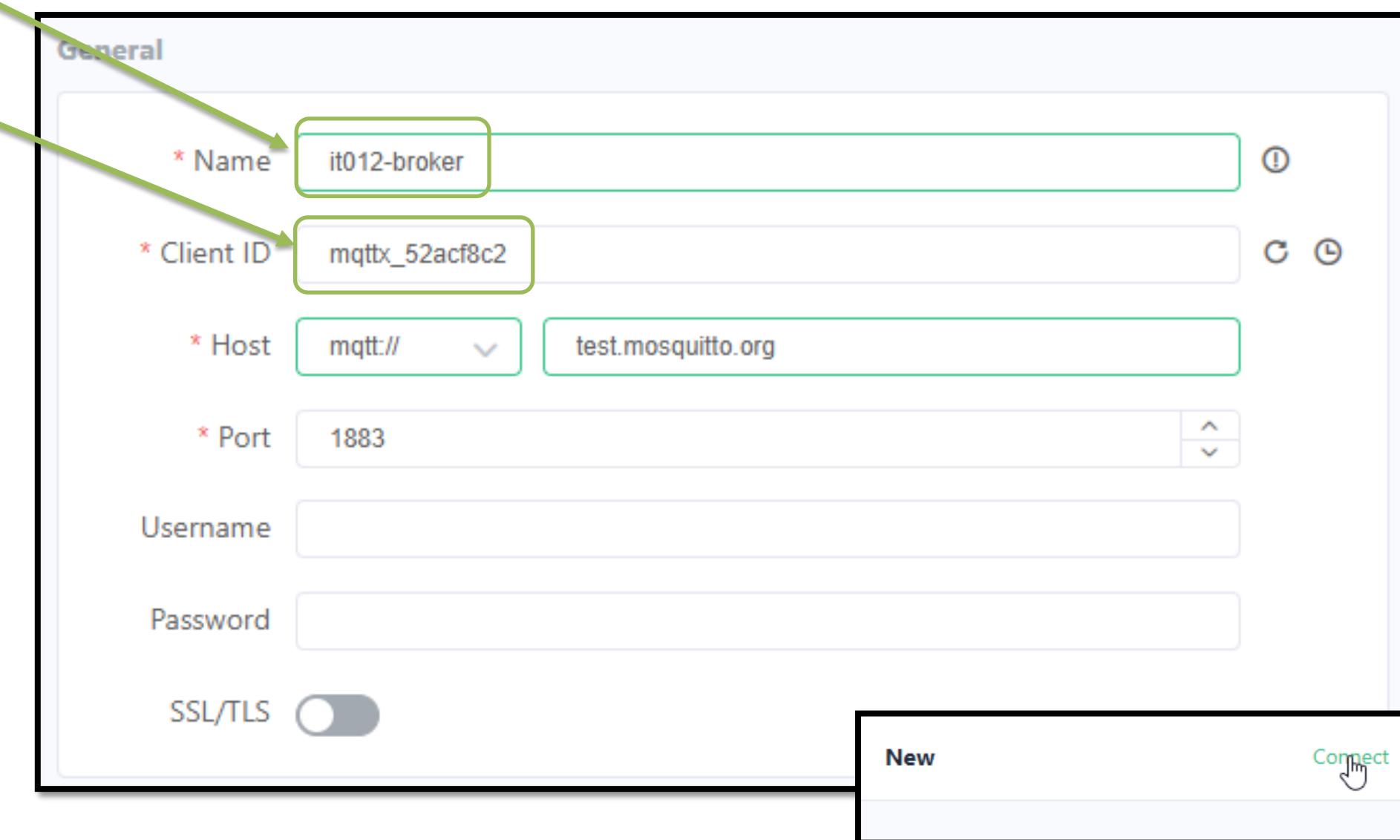
Usaremos o aplicativo MQTTX como client.

- Baixe-o em <https://mqtx.app/>.  
Há versões para Windows, Linux e Mac.
- Instale o programa com as configurações padrão.
- Na tela principal, clique em “New Connection”.



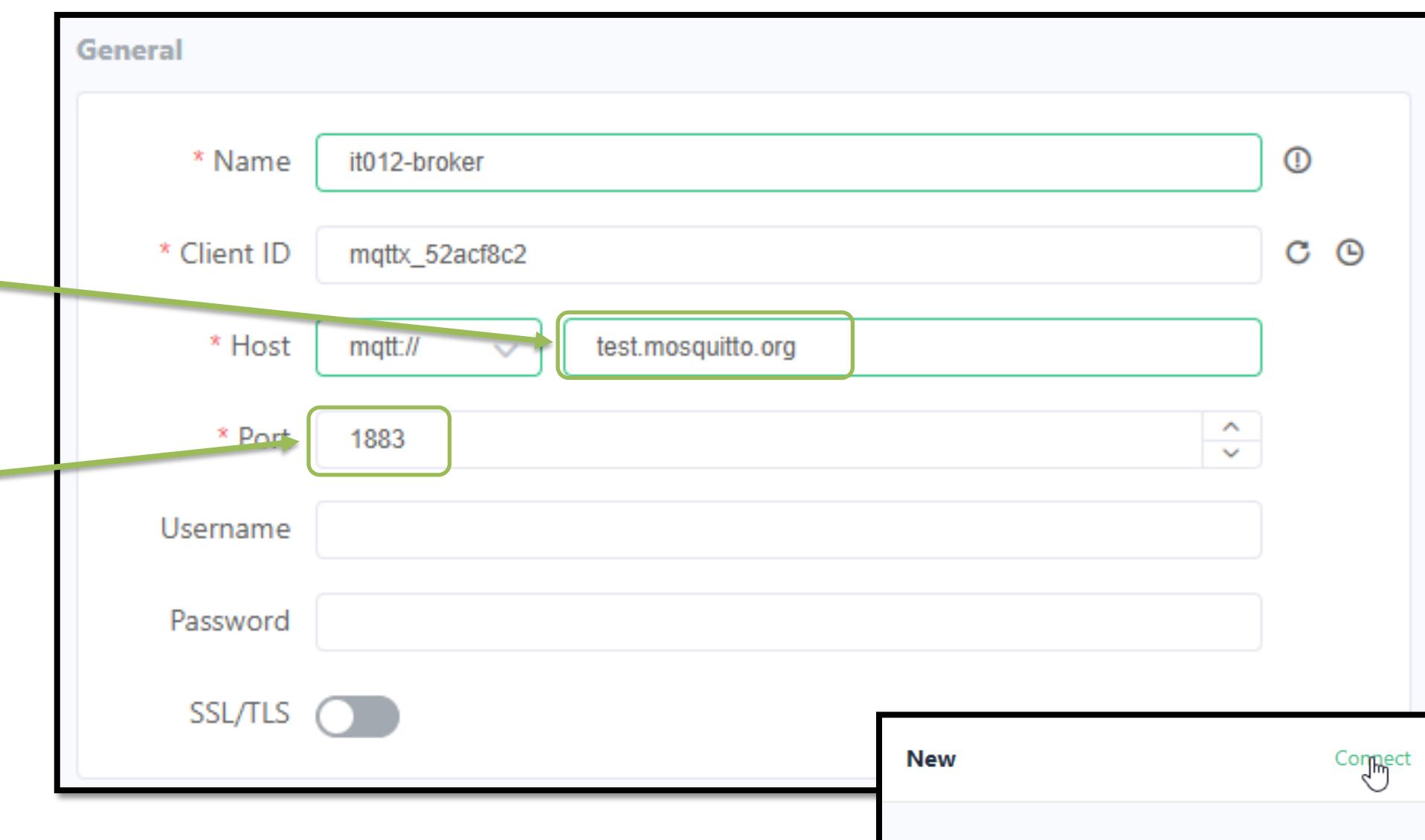
# MQTT – configuração de *clients*

- Define um nome qualquer para a conexão.
- O Client ID será auto-gerado, com uma *string* aleatória.
- Em Host, utilize o mesmo endereço do *broker* configurado no código do ESP32. No exemplo, test.mosquitto.org.
- Use a porta padrão 1883.
- Não será necessário definir *Username*, *Password* ou certificado SSL/TLS.
- Após configurar, clique em *Connect*.



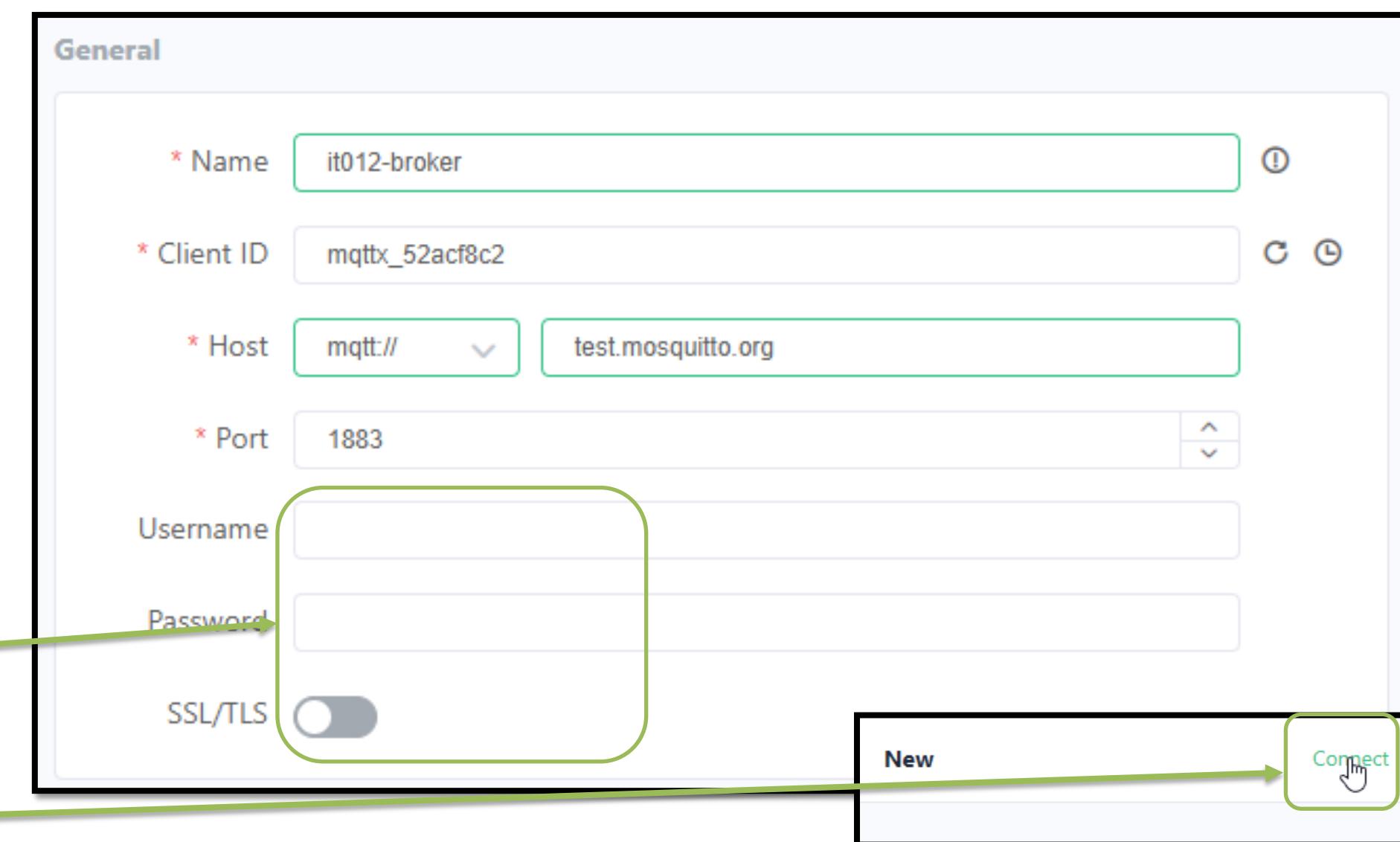
# MQTT – configuração de *clients*

- Define um nome qualquer para a conexão.
- O Client ID será auto-gerado, com uma *string* aleatória.
- Em Host, utilize o mesmo endereço do *broker* configurado no código do ESP32. No exemplo, [test.mosquitto.org](http://test.mosquitto.org).
- Use a porta padrão **1883**.
- Não será necessário definir *Username*, *Password* ou certificado SSL/TLS.
- Após configurar, clique em *Connect*.



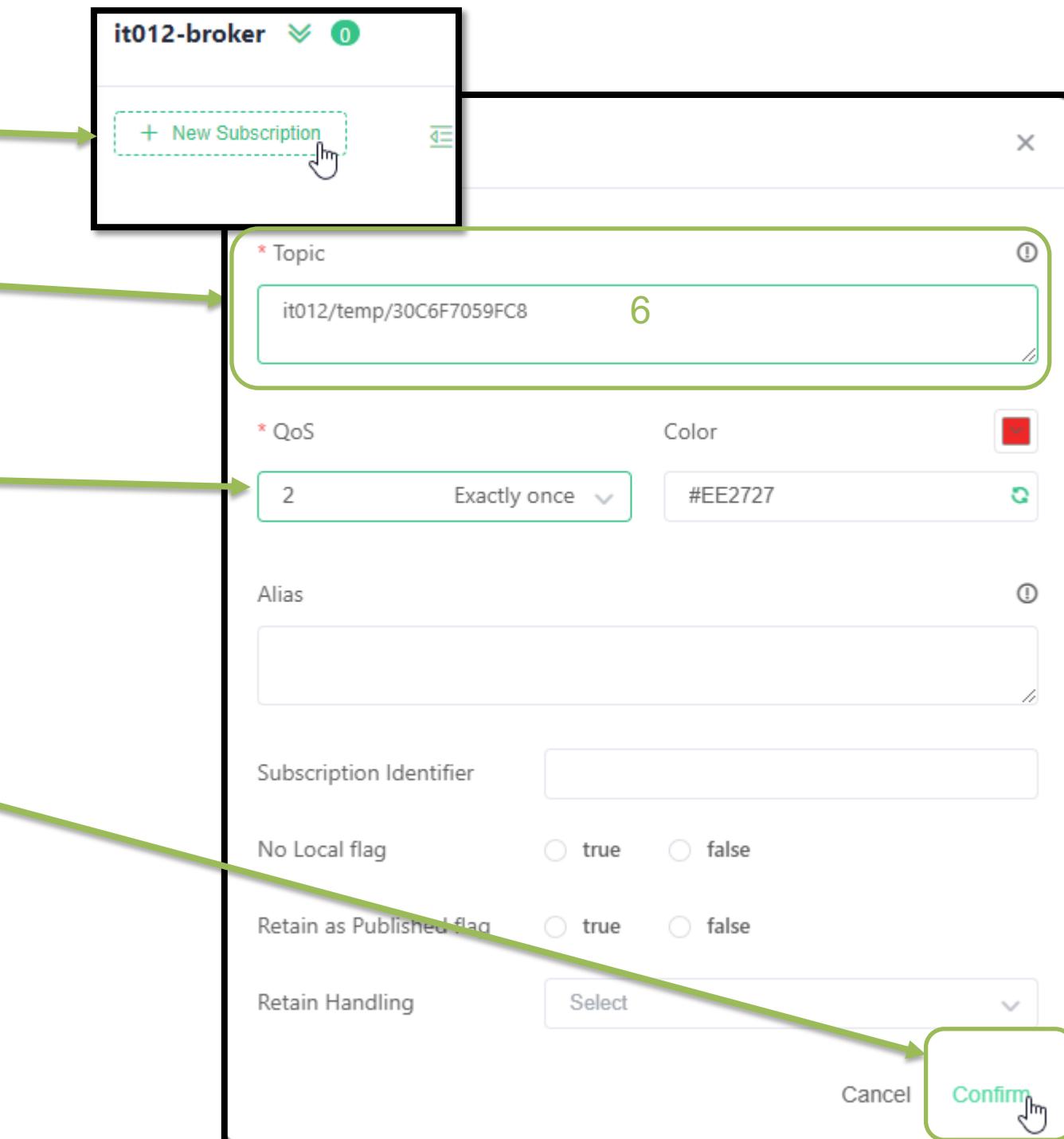
# MQTT – configuração de *clients*

- Define um nome qualquer para a conexão.
- O Client ID será auto-gerado, com uma *string* aleatória.
- Em Host, utilize o mesmo endereço do *broker* configurado no código do ESP32. No exemplo, test.mosquitto.org.
- Use a porta padrão 1883.
- Não será necessário definir *Username*, *Password* ou certificado SSL/TLS.
- Após configurar, clique em *Connect*.



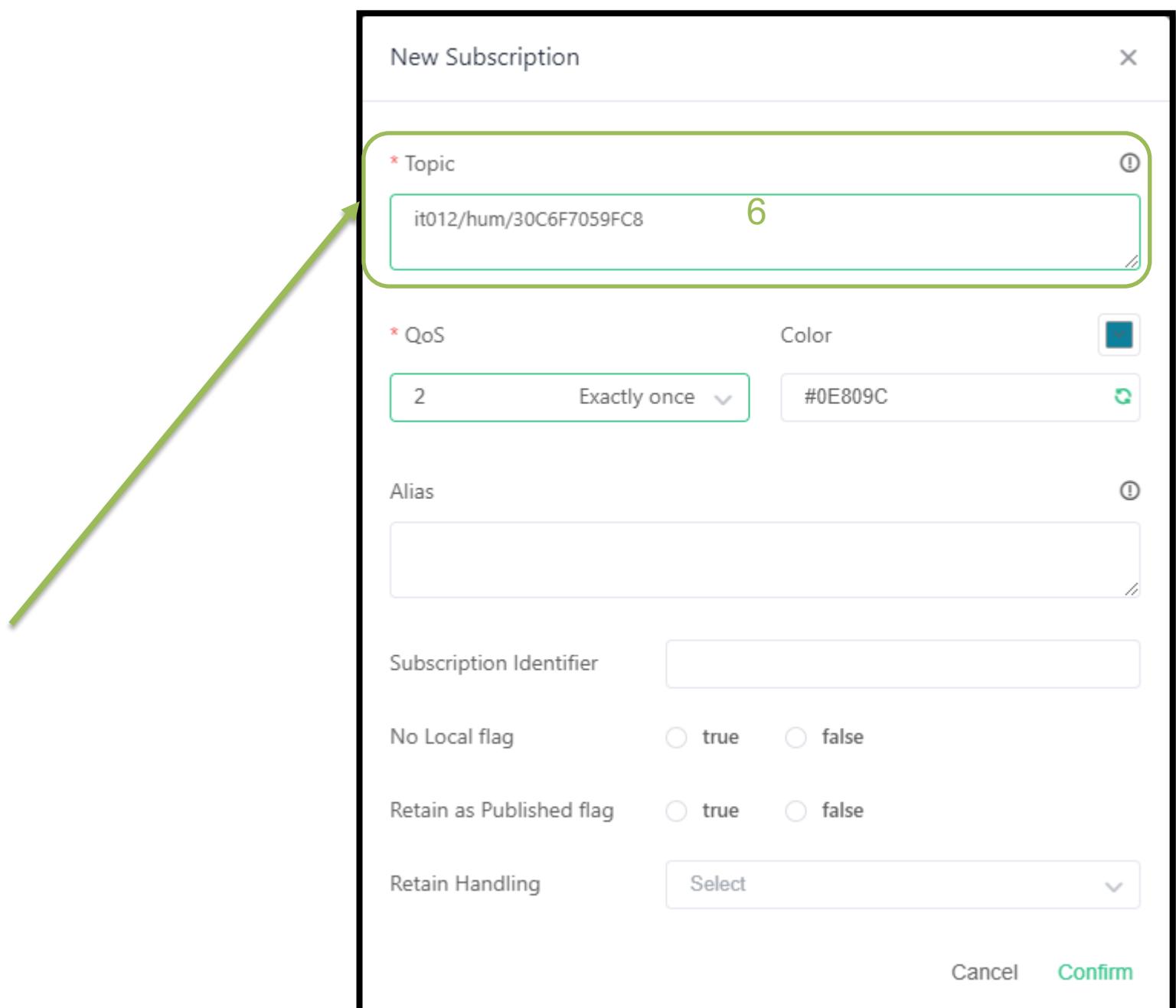
# MQTT – configuração de *clients*

- Selecione “*New Subscription*”.
- Configure a *string* do tópico como sendo a do tópico de temperatura na qual o ESP32 publica.
- Configure o QoS para 2.
- Clique em *Confirm*.
- Repita o procedimento anterior para o tópico da umidade.
- Observe que as mensagens dos tópicos que assinamos são entregues pelo *broker* ao nosso *client* conforme são enviadas pelo ESP32.



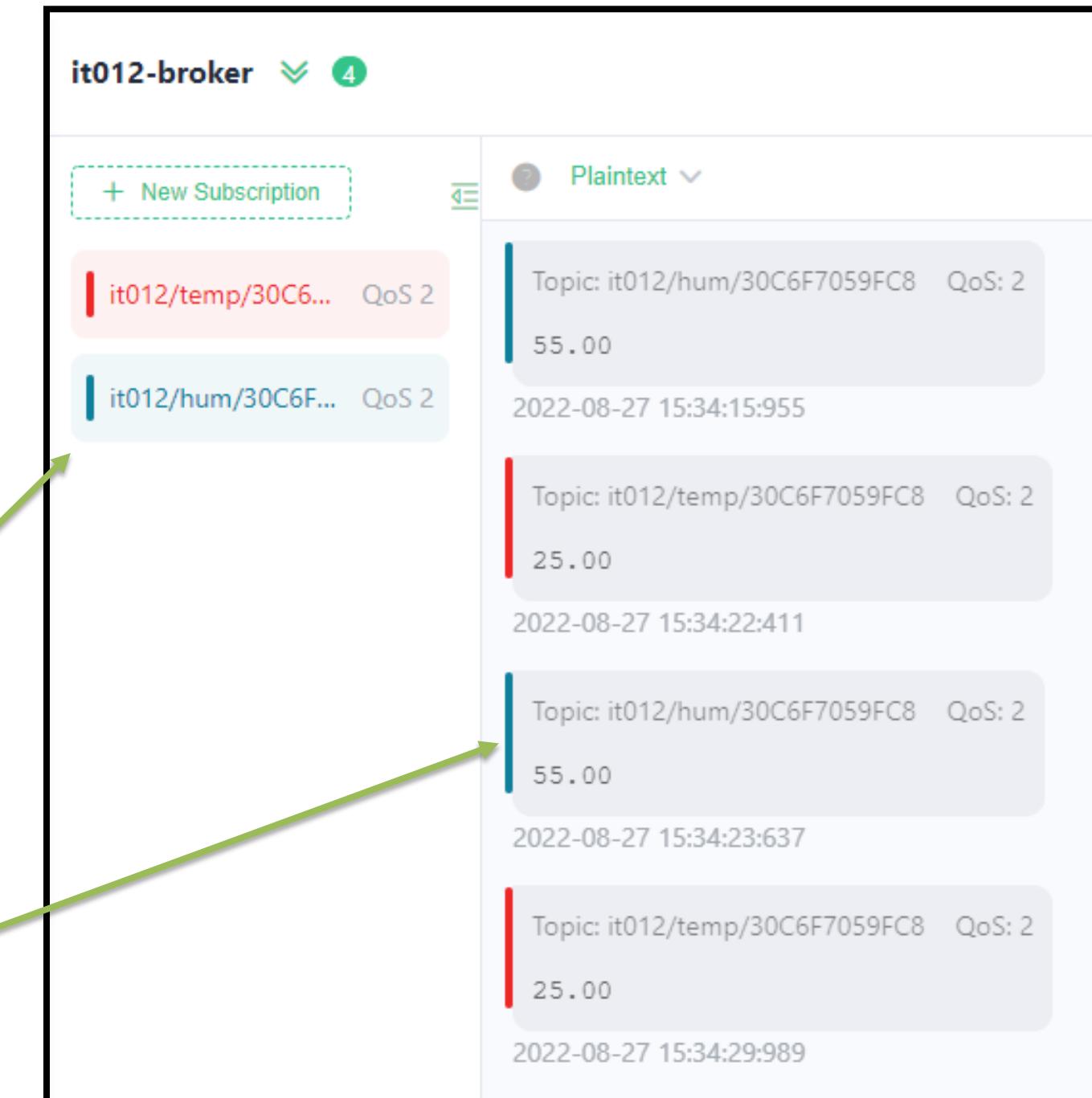
# MQTT – configuração de *clients*

- Selecione “New Subscription”.
- Configure a *string* do tópico como sendo a do tópico de temperatura na qual o ESP32 publica.
- Configure o QoS para 2.
- Clique em *Confirm*.
- Repita o procedimento anterior para o tópico da umidade.
- Observe que as mensagens dos tópicos que assinamos são entregues pelo *broker* ao nosso *client* conforme são enviadas pelo ESP32.



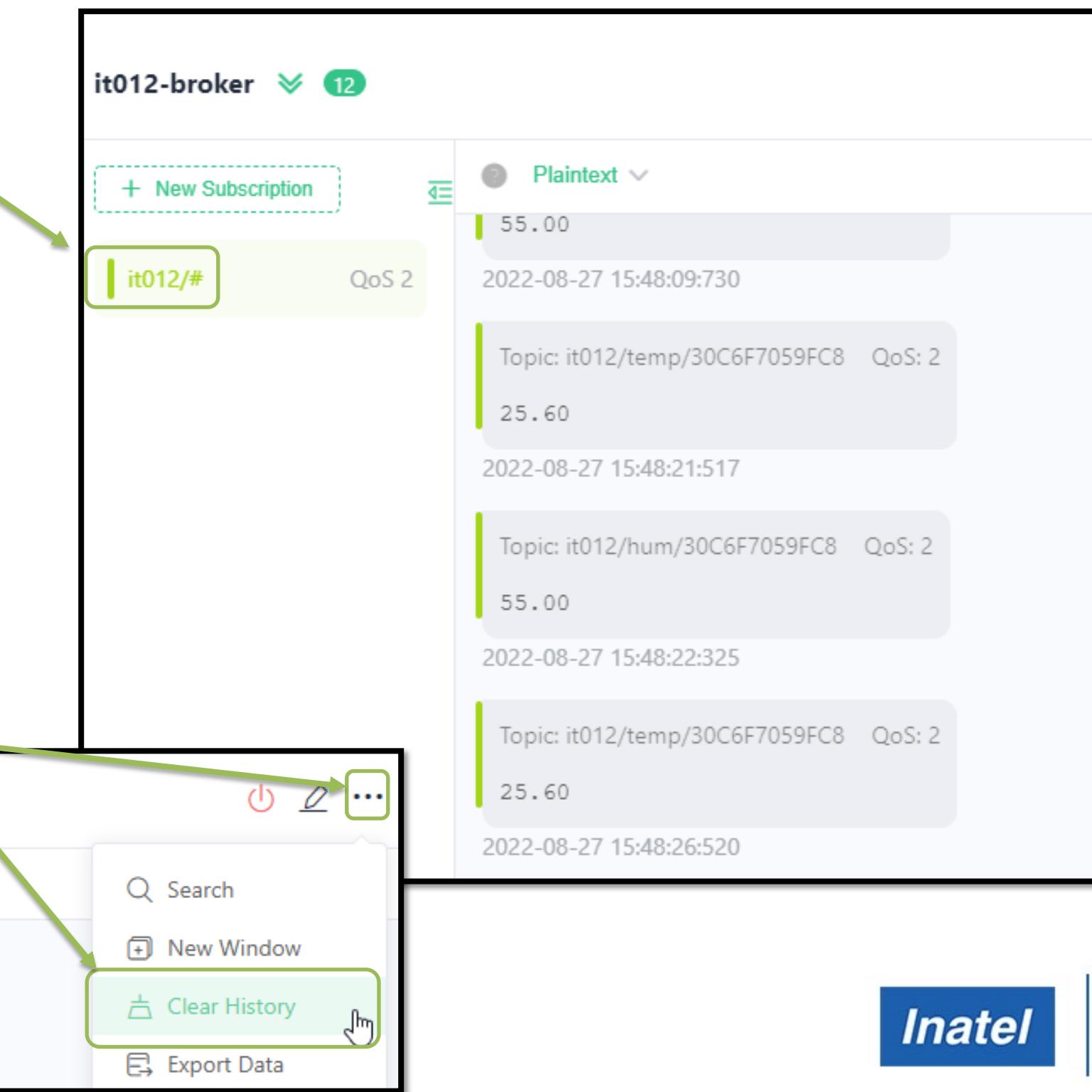
# MQTT – configuração de *clients*

- Selecione “*New Subscription*”.
- Configure a *string* do tópico como sendo a do tópico de temperatura na qual o ESP32 publica.
- Configure o QoS para 2.
- Clique em *Confirm*.
- Repita o procedimento anterior para o tópico da umidade.
- Observe que as mensagens dos tópicos que assinamos são entregues pelo *broker* ao nosso *client* conforme são enviadas pelo ESP32.



# MQTT – configuração de *clients*

- Observe que é possível realizar um *subscribe* multi-nível, utilizando o separador **#**.
- Desta forma, pode-se receber mensagens de todos os sub-níveis de uma estrutura de tópicos.
- Como exemplo, se realizarmos o *subscribe* em **it012/#**, o *client* irá receber mensagens do broker de todos os tópicos com *string* inicial “*it012*”.
- Você pode limpar as mensagens recebidas pelo *client* em “*Clear History*”.



## Exemplo: MQTT (*Subscribe*)

- Abra a pasta: exemplo\_mqtt\_subscribe
- Adicione o LED RGB à montagem. O circuito na *protoboard* ficará conforme a figura:

D5: LED Vermelho (R)

D18: LED Verde (G)

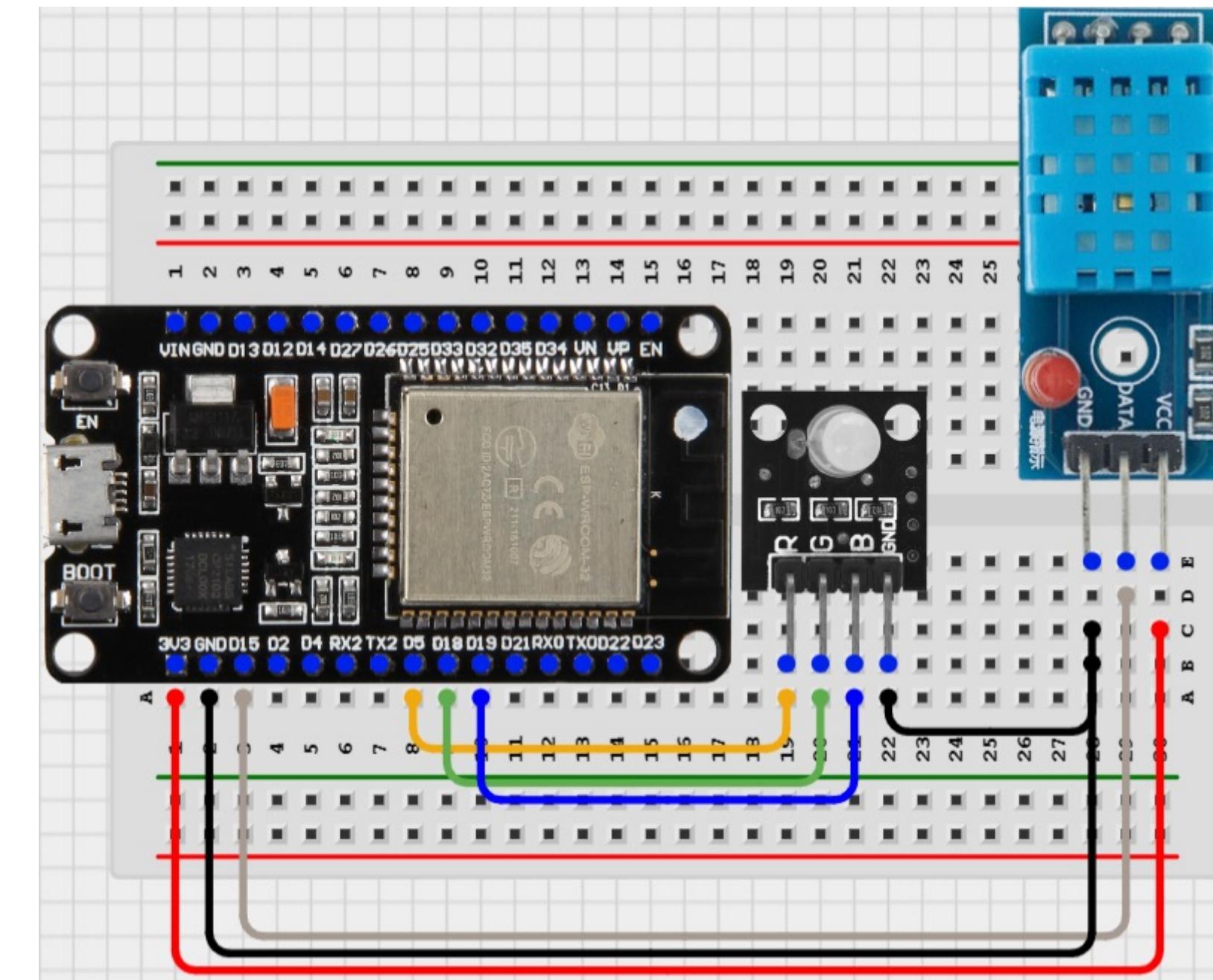
D19: LED Azul (B)

D15: Pino DATA DHT11

3V3: Pinos VCC & 3V3 em curto

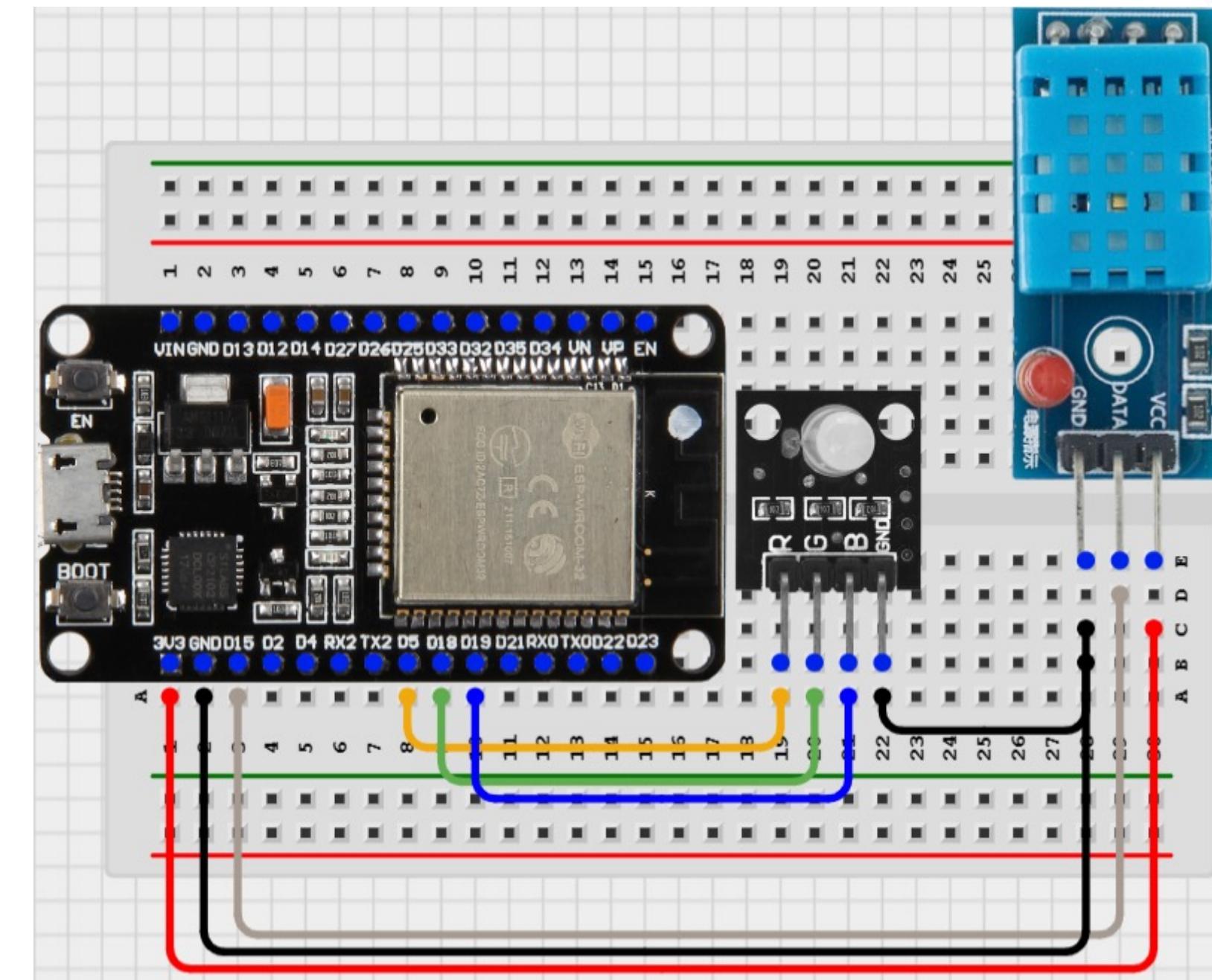
GND: Pinos GND em curto

- O DHT11 não será usado neste exemplo.

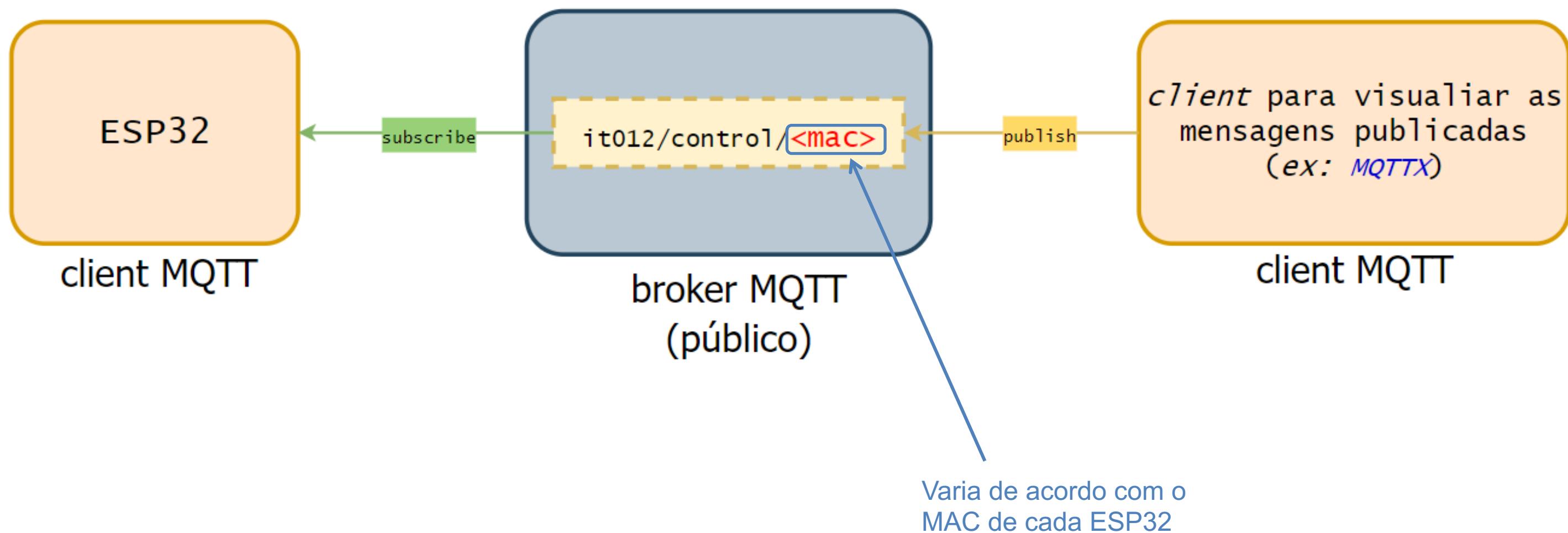


## Exemplo: MQTT (*Subscribe*)

- Neste exemplo:
  - O ESP32 se conecta à uma rede Wifi e à um broker MQTT;
  - Assina um tópico, que será usado para tratar mensagens de controle recebidas (`it012/control/<mac>`);
  - De acordo com o *payload* recebido, acende ou apaga os LED's do LED RGB.
- Similar à [exemplo\\_mqtt\\_publish](#), mas agora com o objetivo de demonstrar a assinatura de tópicos (*subscribe*).



## Exemplo: MQTT (*Subscribe*)



## Exemplo: MQTT (*Subscribe*)

- Uma nova *string* de tópico é definida, que será concatenada de forma similar ao exemplo anterior com o MAC do ESP32.
- O dispositivo irá assinar esse tópico para receber mensagens publicadas de outros *clients*.
- O método `mqttClient.onMessage()` é usado na inicialização da *lib* MQTT para definir uma função de *callback* (`onMqttMessage`) que será chamada no momento em que o dispositivo receber uma mensagem do *broker*.

```
// Configurações para acesso ao broker MQTT
const char* MQTT_BROKER = "test.mosquitto.org";
const char* MQTT_TOPIC_CONTROL = "it012/control/";
const int MQTT_PORT = 1883;
AsyncMqttClient mqttClient;
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;
```

```
// Define callback genérico para os eventos relacionados à conexão wifi
WiFi.onEvent(WiFiEvent);

// Definição de callbacks para eventos específicos do MQTT
mqttClient.onConnect(onMqttConnect);
mqttClient.onDisconnect(onMqttDisconnect);
mqttClient.onMessage(onMqttMessage);

// Seta o servidor do broker MQTT e inicia conexão Wifi
mqttClient.setServer(MQTT_BROKER, MQTT_PORT);
connectToWifi();
```

## Exemplo: MQTT (*Subscribe*)

- `controlSubscribe()` realiza o processo de assinar (*subscribe*) em um tópico, no momento em que uma conexão MQTT ocorrer com sucesso (disparando o *callback* `onMqttConnect()`, visto no exemplo anterior).
- O método `mqttClient.subscribe()` é usado para realizar o *subscribe* no *broker*.
- No exemplo, assinamos um tópico formado com a *string* `topic`, em QoS 0.

```
void onMqttConnect(bool sessionPresent)
{
    Serial.printf("\r\n[connectToMqtt] conectado ao broker MQTT %s. ", MQTT_BROKER);

    // Se há conexão ao broker, assina o tópico de controle
    controlSubscribe();
}
```

```
void controlSubscribe()
{
    // Configura um tópico específico para este dispositivo, cujo nome será
    // gerado concatenando o MAC lido do ESP32.
    uint8_t mac[6];
    char topic[60] = { 0 };
    esp_read_mac(mac, ESP_MAC_WIFI_STA);
    sprintf(topic, sizeof(topic), "%s%02X%02X%02X%02X%02X",
            MQTT_TOPIC_CONTROL, mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);

    // Assina um tópico para controle com nome definido para este dispositivo
    Serial.printf("\r\n[controlSubscribe] Assinando tópico %s...", topic);
    mqttClient.subscribe(topic, 0);
}
```

## Exemplo: MQTT (*Subscribe*)

- `onMqttMessage()` é chamada ao receber uma mensagem, caso o tópico da mesma seja um dos tópicos para os quais o ESP32 realizou o *subscribe*.
- O argumento `topic` contém o tópico ao qual a mensagem está destinada, e `payload` contém os dados da mensagem MQTT.
- Deve-se atentar ao fato de que o MQTT utiliza *payloads* com dados binários e não *strings* por padrão.
- Aqui, no entanto, os dados serão gerados por outro *client* como *arrays* com caracteres ASCII – por exemplo, a *string* (`red=1`).

```
void onMqttMessage(char* topic, char* payload,
                    AsyncMqttClientMessageProperties properties, size_t len,
                    size_t index, size_t total)
{
    // O símbolo "%.*s" abaixo é utilizado para enviar apenas os "len" primeiros
    // bytes de uma string (payload). Usado em cenários como esse, quando, por
    // exemplo, payload é um buffer de uma string de texto, mas que pode conter
    // dados além dos quais se deseja mostrar com o print.
    Serial.printf("\r\n[onMqttMessage] mensagem de %d bytes ", len);
    Serial.printf("no tópico %s [%.*s]", topic, len, payload);

    // Parser LED vermelho
    if (strstr(payload, "red=1")) {
        digitalWrite(LED_RED, HIGH);

    } else if (strstr(payload, "red=0")) {
        digitalWrite(LED_RED, LOW);
    }
    // Parser LED verde
    if (strstr(payload, "green=1")) {
        digitalWrite(LED_GREEN, HIGH);

    } else if (strstr(payload, "green=0")) {
        digitalWrite(LED_GREEN, LOW);
    }
    // Parser LED azul
    if (strstr(payload, "blue=1")) {
        digitalWrite(LED_BLUE, HIGH);

    } else if (strstr(payload, "blue=0")) {
        digitalWrite(LED_BLUE, LOW);
    }
}
```

## Exemplo: MQTT (Subscribe)

- No exemplo, é implementado um *parser* (interpretador) de *strings* simples: utiliza a função da biblioteca C `strstr()` para verificar ocorrências de *substrings* específicas no *payload* da mensagem MQTT.
- `strstr()` retorna um ponteiro para a posição do *array* buscado (*payload*) que contém o início da *substring* buscada. Retorna um ponteiro nulo (`NULL`) caso não encontre.
- Por exemplo, de acordo com a *string* recebida, o pino correspondente ao LED azul (`LED_BLUE`) do LED RGB é ativado (`blue=1`) ou não (`blue=0`).

```
void onMqttMessage(char* topic, char* payload,
                    AsyncMqttClientMessageProperties properties, size_t len,
                    size_t index, size_t total)

{
    // O símbolo "%.*s" abaixo é utilizado para enviar apenas os "len" primeiros
    // bytes de uma string (payload). Usado em cenários como esse, quando, por
    // exemplo, payload é um buffer de uma string de texto, mas que pode conter
    // dados além dos quais se deseja mostrar com o print.
    Serial.printf("\r\n[onMqttMessage] mensagem de %d bytes ", len);
    Serial.printf("no tópico %s [%.*s]", topic, len, payload);

    // Parser LED vermelho
    if (strstr(payload, "red=1")) {
        digitalWrite(LED_RED, HIGH);

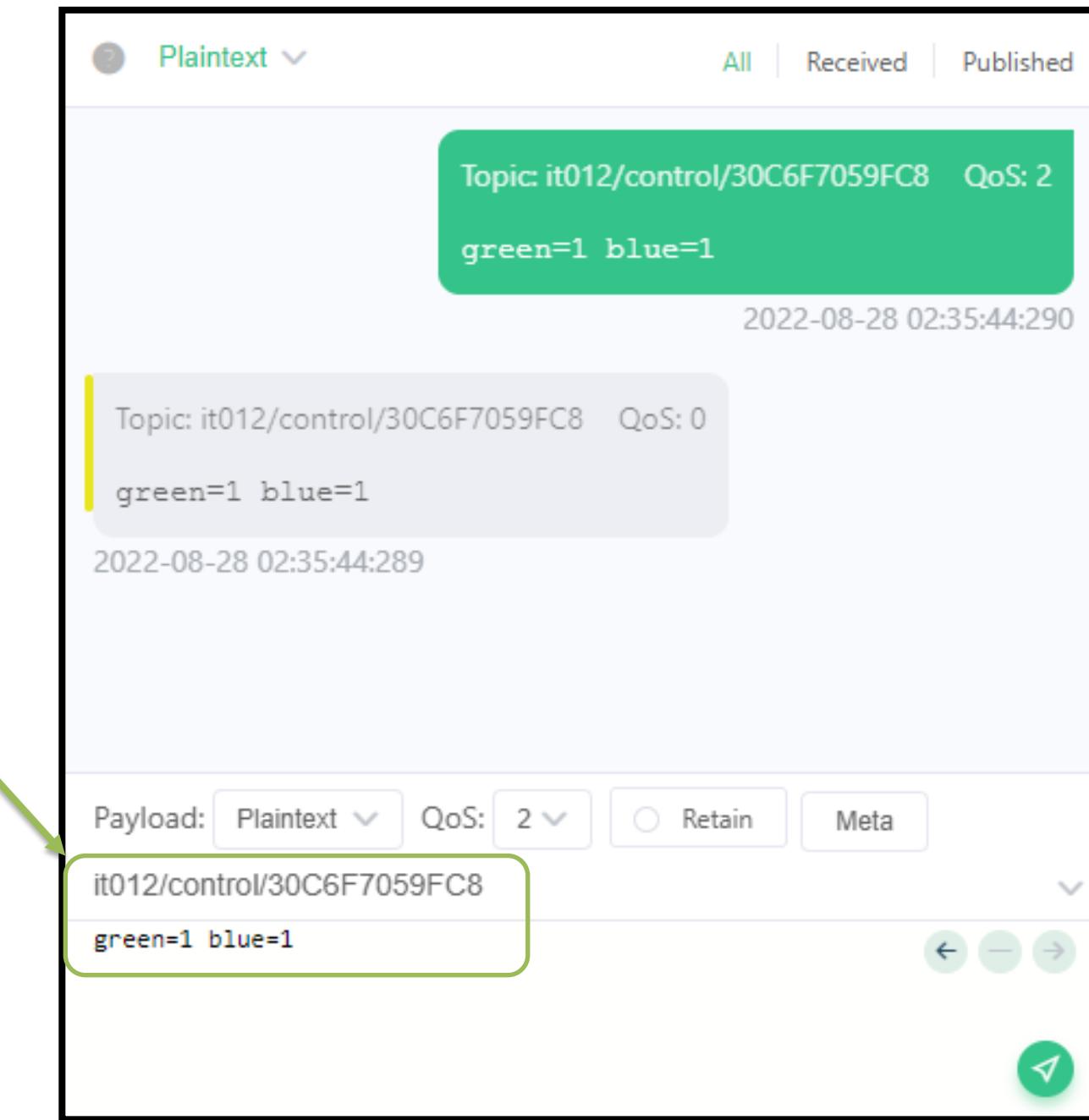
    } else if (strstr(payload, "red=0")) {
        digitalWrite(LED_RED, LOW);
    }
    // Parser LED verde
    if (strstr(payload, "green=1")) {
        digitalWrite(LED_GREEN, HIGH);

    } else if (strstr(payload, "green=0")) {
        digitalWrite(LED_GREEN, LOW);
    }
    // Parser LED azul
    if (strstr(payload, "blue=1")) {
        digitalWrite(LED_BLUE, HIGH);

    } else if (strstr(payload, "blue=0")) {
        digitalWrite(LED_BLUE, LOW);
    }
}
```

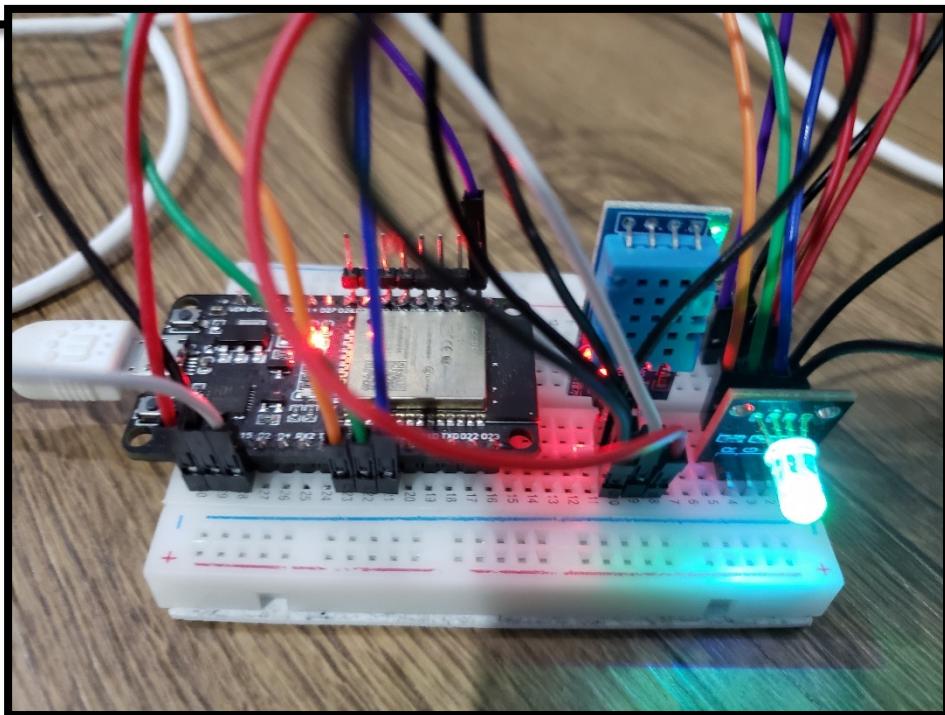
## Exemplo: MQTT (*Subscribe*)

- Para testar o recebimento de mensagens, pode-se utilizar o MQTTX para publicar mensagens no tópico de controle de cada dispositivo.
- Por exemplo, um *publish* no tópico `it012/control/<mac>` com *payload* `green=1 blue=1` deve ativar os LEDs verde e azul do LED RGB.
- Este *client* também pode assinar o mesmo tópico que o ESP32 assina, para testar o recebimento de mensagens, como feito na figura.



## Exemplo: MQTT (*Subscribe*)

```
02:34:48.745 -> --- exemplo_mqtt_sub_json ---
02:34:48.745 ->
02:34:48.745 -> [connectToWifi] conectando à rede it012...
02:34:48.883 -> [WiFiEvent] Conexão perdida...
02:34:50.884 ->
02:34:50.884 -> [connectToWifi] conectando à rede it012...
02:34:50.977 -> [WiFiEvent] IP obtido: 192.168.43.11
02:34:50.977 ->
02:34:50.977 -> [connectToMqtt] conectando ao broker MQTT test.mosquitto.org...
02:34:51.630 -> [connectToMqtt] conectado ao broker MQTT test.mosquitto.org.
02:34:51.630 -> [controlSubscribe] Assinando tópico it012/control/30C6F7059FC8...
02:34:56.540 -> [onMqttMessage] mensagem de 14 bytes no tópico it012/control/30C6F7059FC8 [green=1 blue=1]
```



# Paho

Paho Biblioteca python para MQTT



<https://www.eclipse.org/paho/>

Inatel

CAMINHOS  
QUE CONECTAM  
COM O FUTURO

# Paho

```
def publish(client):
    msg_count = 1
    while True:
        time.sleep(1)
        msg = f"messages: {msg_count}"
        result = client.publish(topic, msg)
        # result: [0, 1]
        status = result[0]
        if status == 0:
            print(f"Send `{msg}` to topic `{topic}`")
        else:
            print(f"Failed to send message to topic {topic}")
        msg_count += 1
        if msg_count > 5:
            break
```

<https://www.emqx.com/en/blog/how-to-use-mqtt-in-python>

# Paho

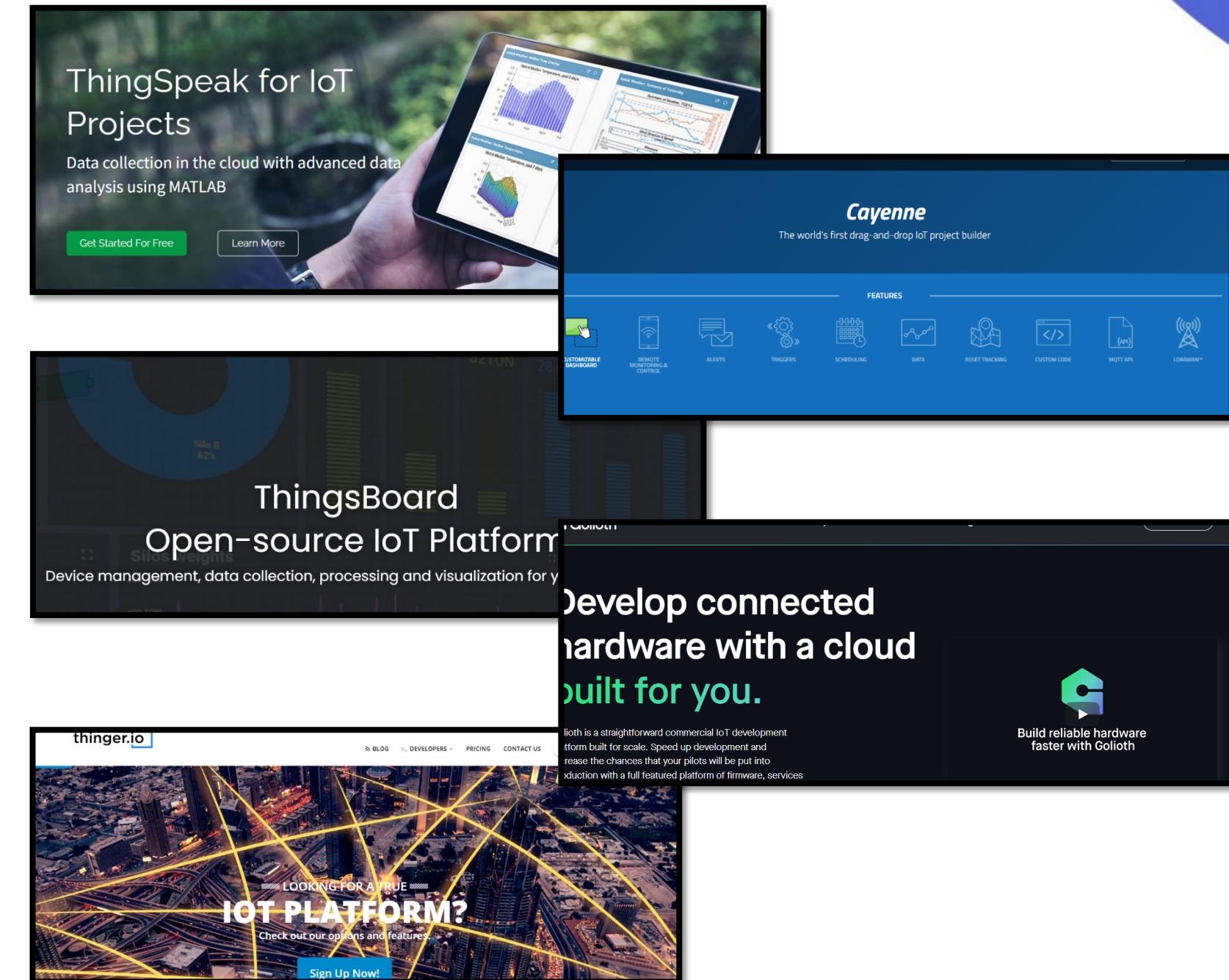
```
def subscribe(client: mqtt_client):
    def on_message(client, userdata, msg):
        print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")

    client.subscribe(topic)
    client.on_message = on_message
```

<https://www.emqx.com/en/blog/how-to-use-mqtt-in-python>

# Dashboards

- Um *dashboard* para IoT é uma ferramenta de visualização de dados que transforma, exibe e organiza uma coleção de dados transmitidos por dispositivos.
- Disponíveis em diversos serviços em nuvem: [ThingSpeak](#), [ThingsBoard](#), [Cayenne](#), [thinger.io](#), [Golioth](#), [Arduino Cloud](#), [Blynk](#), entre diversas outras.
- Também disponível na forma de aplicativos para Android / iOS, como demonstrado no exemplo a seguir.



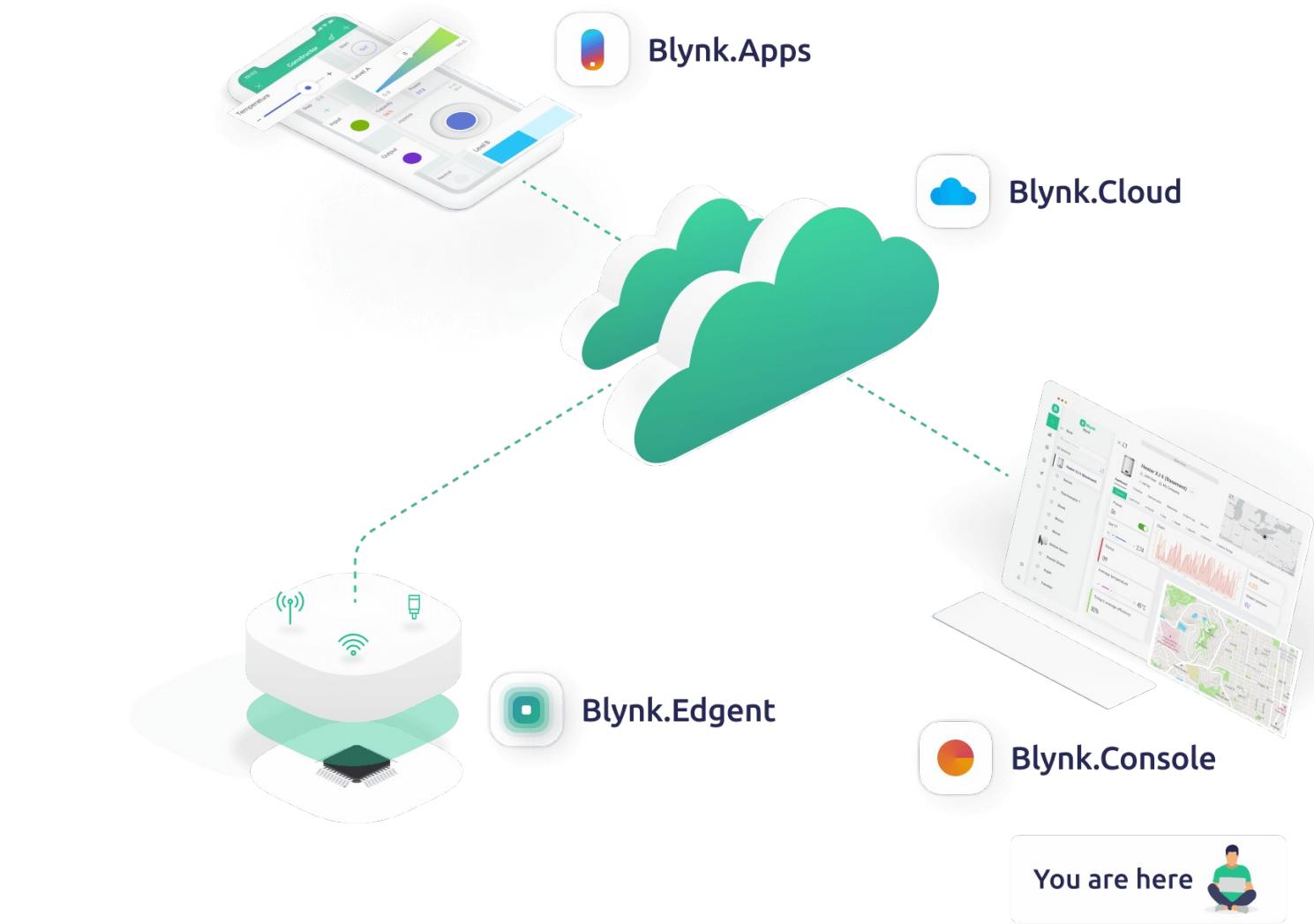
# Blynk - Introdução

- O Blynk é um conjunto completo de software necessário para prototipar, implantar e gerenciar remotamente dispositivos eletrônicos conectados em qualquer escala: de projetos pessoais de IoT a milhões de produtos comerciais conectados.
- Ele pode controlar o hardware remotamente, exibir dados do sensor, armazenar dados, visualizá-los e fazer muitas outras coisas legais.



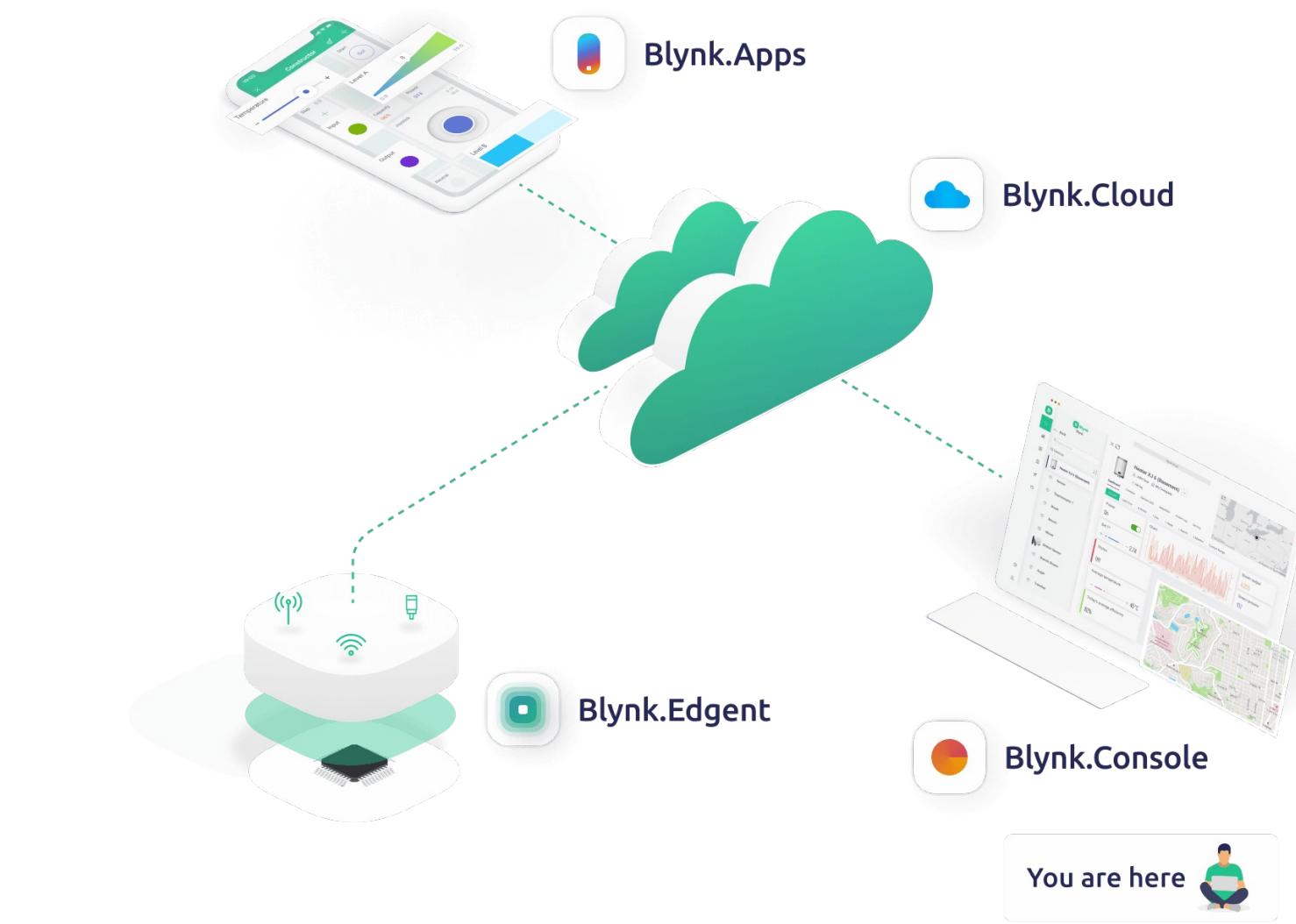
# Blynk - Introdução

- A plataforma Blynk consiste em quatro componentes principais que funcionam perfeitamente juntos:
- Blynk.Edgent: software que roda no seu dispositivo e se comunica com o Blynk.Cloud.



# Blynk - Introdução

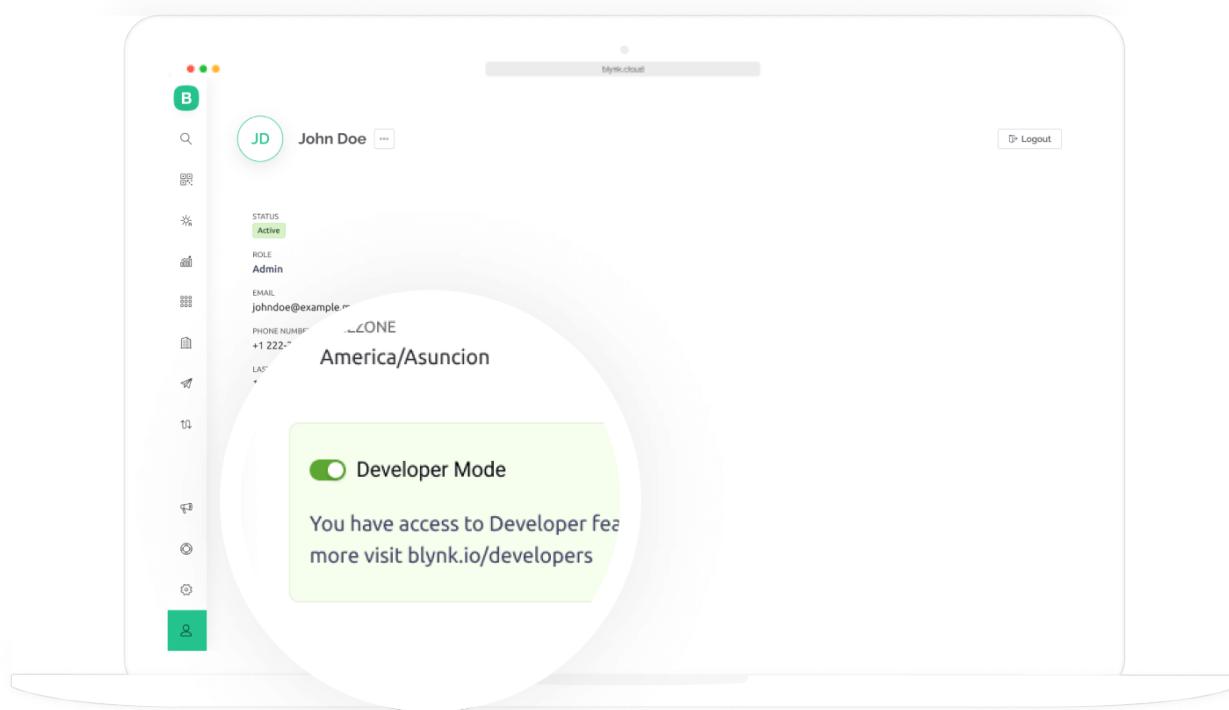
- Blynk.Console: aplicativo web onde você pode configurar, conectar, supervisionar seus dispositivos, analisar dados de sensores, atualizar firmware OTA e gerenciar como outros usuários e organizações acessam seus dispositivos.
- Blynk.Apps: aplicativos móveis para iOS e Android onde você pode criar a interface do usuário para seus dispositivos sem codificação e compartilhá-la com outros usuários.



# Blynk - Introdução

Blynk opera em 2 modos:

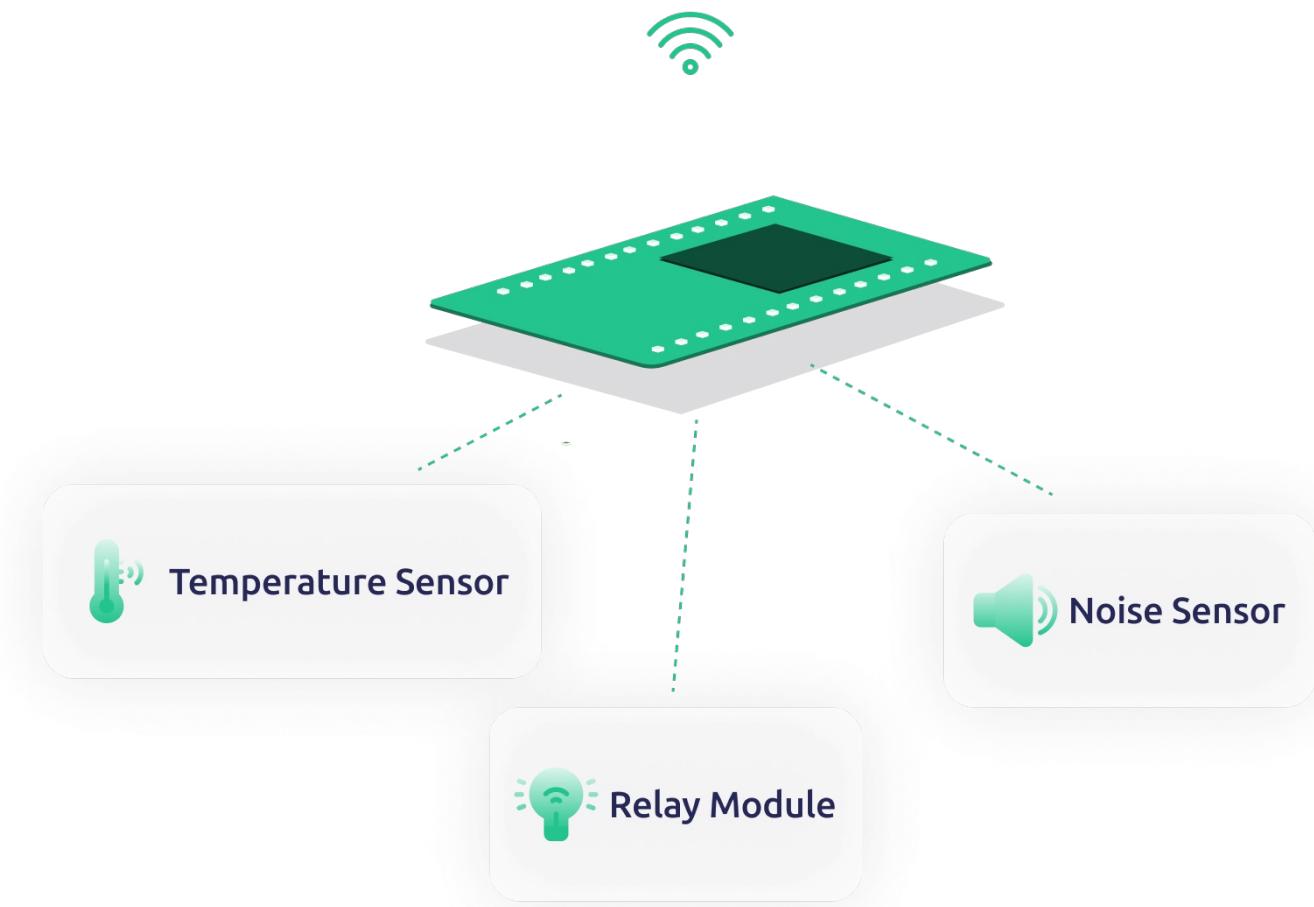
- O modo de desenvolvedor (que você está usando atualmente) permite configurar como os dispositivos devem funcionar.
- O Modo de Usuário permite monitorar e controlar os dispositivos, mas não permite modificar nenhuma configuração.
- Você pode ativar/desativar o Modo de Desenvolvedor na seção Perfil do Usuário do menu principal.



# Blynk - Introdução

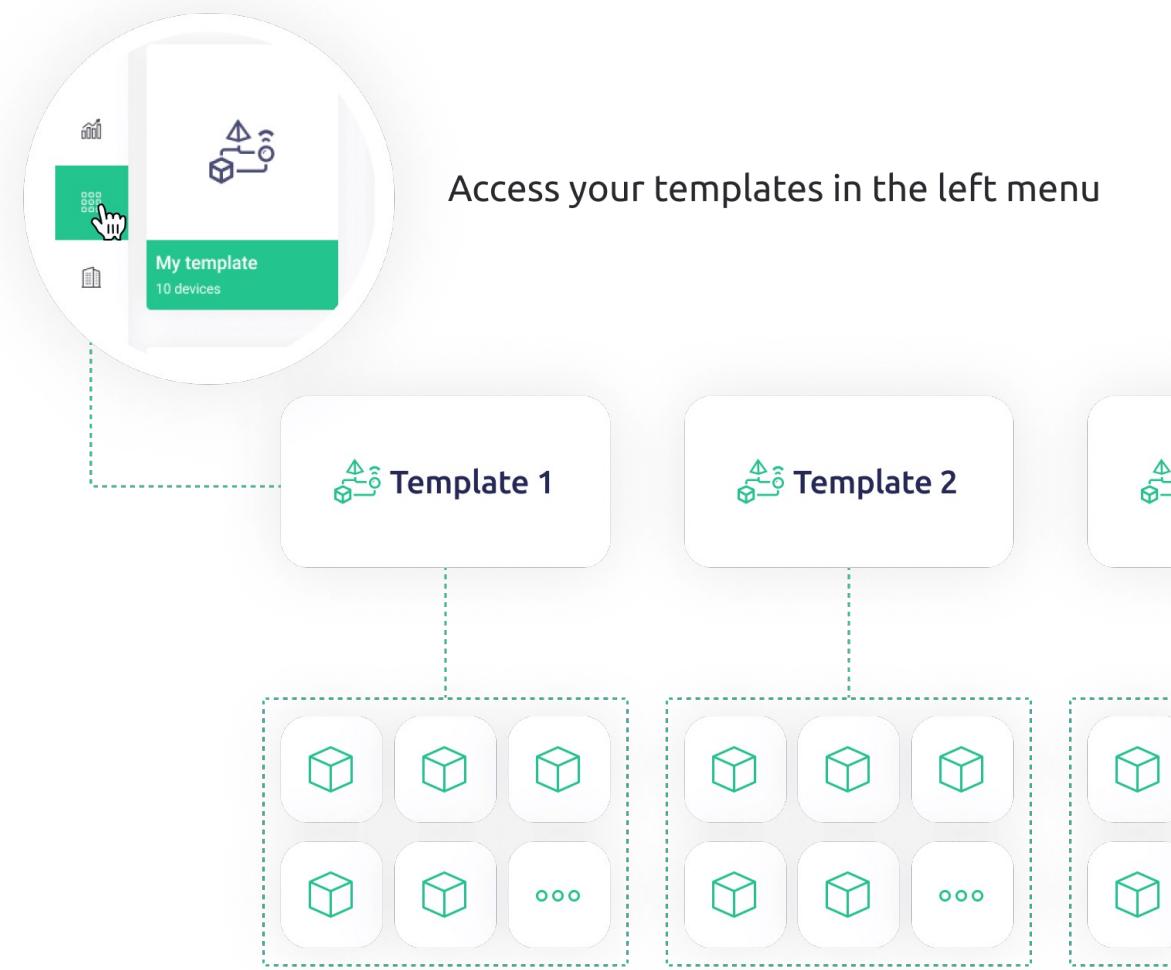
## Dispositivo

- Um dispositivo geralmente é um microcontrolador (MCU) como ESP32, Arduino, etc.
- Você pode conectar sensores e atuadores a um MCU e monitorá-los ou controlá-los com o Blynk. Blynk pode conectar seu dispositivo à Internet usando conectividade Wi-Fi, celular ou Ethernet.



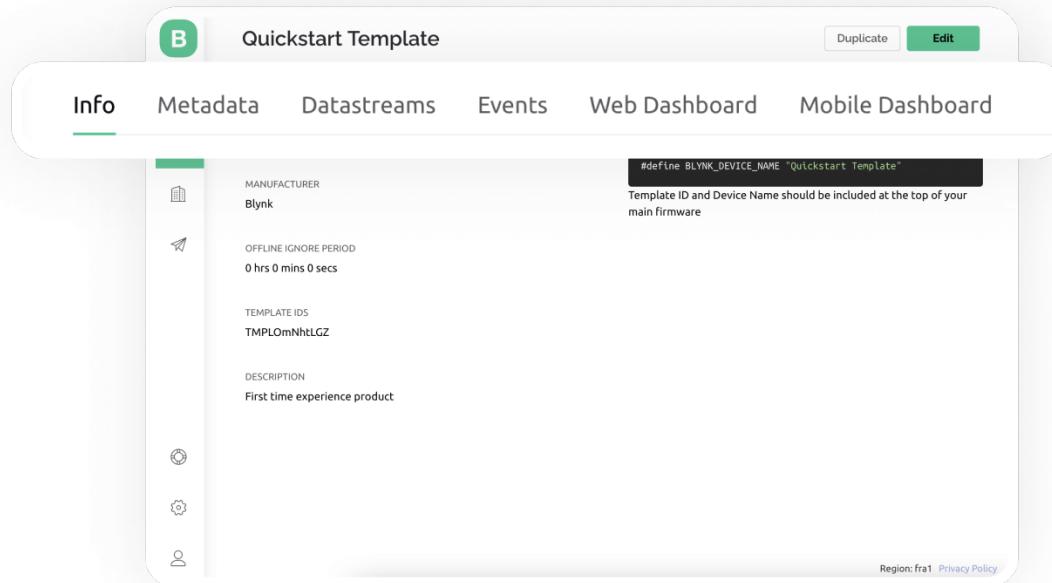
# Blynk - Introdução

- Modelo de dispositivo
- As configurações do dispositivo são armazenadas em algo que chamamos de Device Template.
- Cada dispositivo começa a partir de um modelo, o que facilita o trabalho com vários dispositivos que executam funções semelhantes.
- Por exemplo, você pode criar um modelo de sensor de temperatura e reutilizá-lo para todos os sensores semelhantes em sua casa.



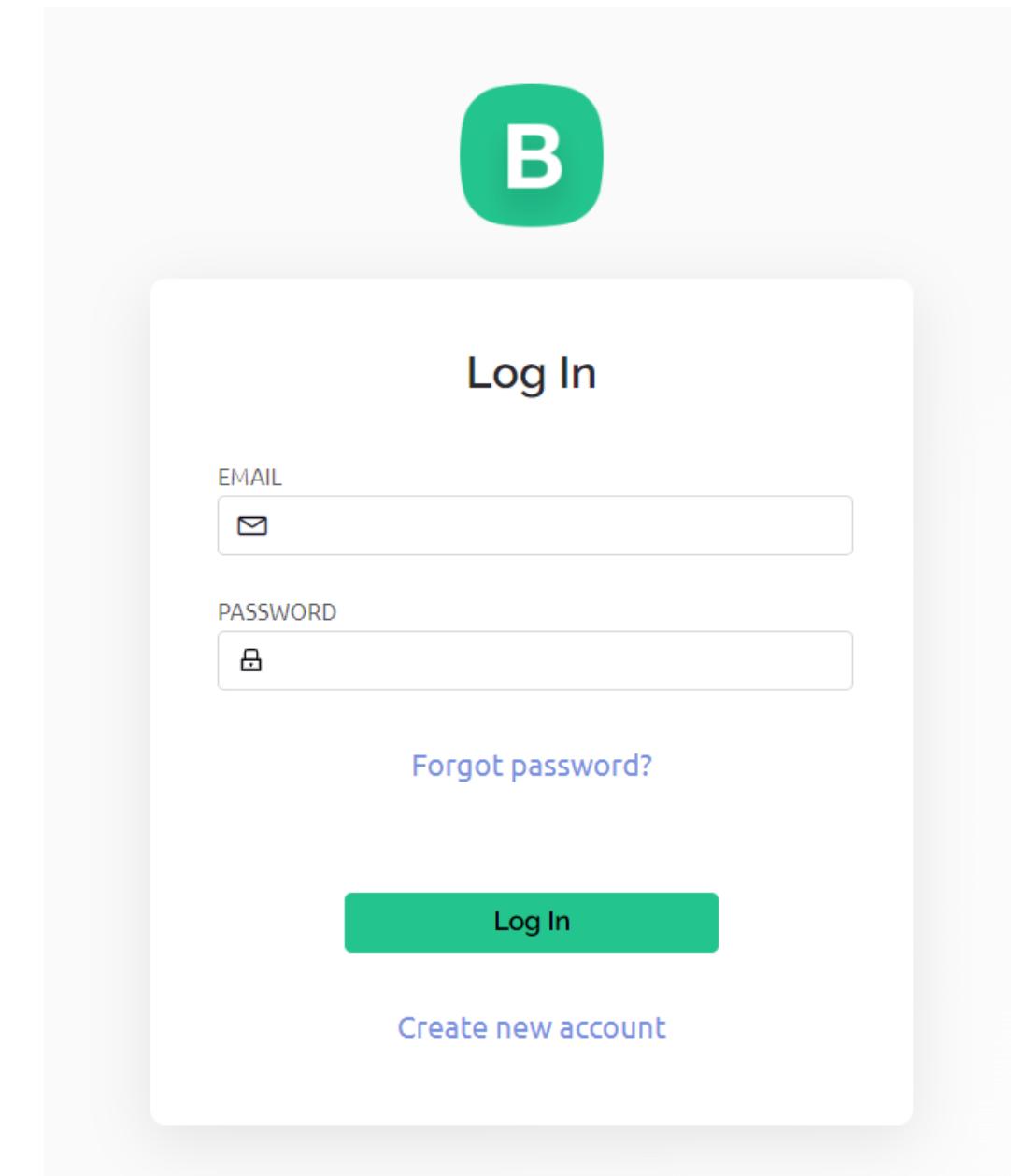
# Blynk - Introdução

- Componentes do modelo
- Cada modelo consiste em:
- Datastreams - canais para transferir dados de/para o dispositivo
- Aplicativo para dispositivos móveis
- Interface de usuário do painel da Web
- Notificações Ao atualizar um modelo, as alterações serão aplicadas a todos os dispositivos criados a partir desse modelo.



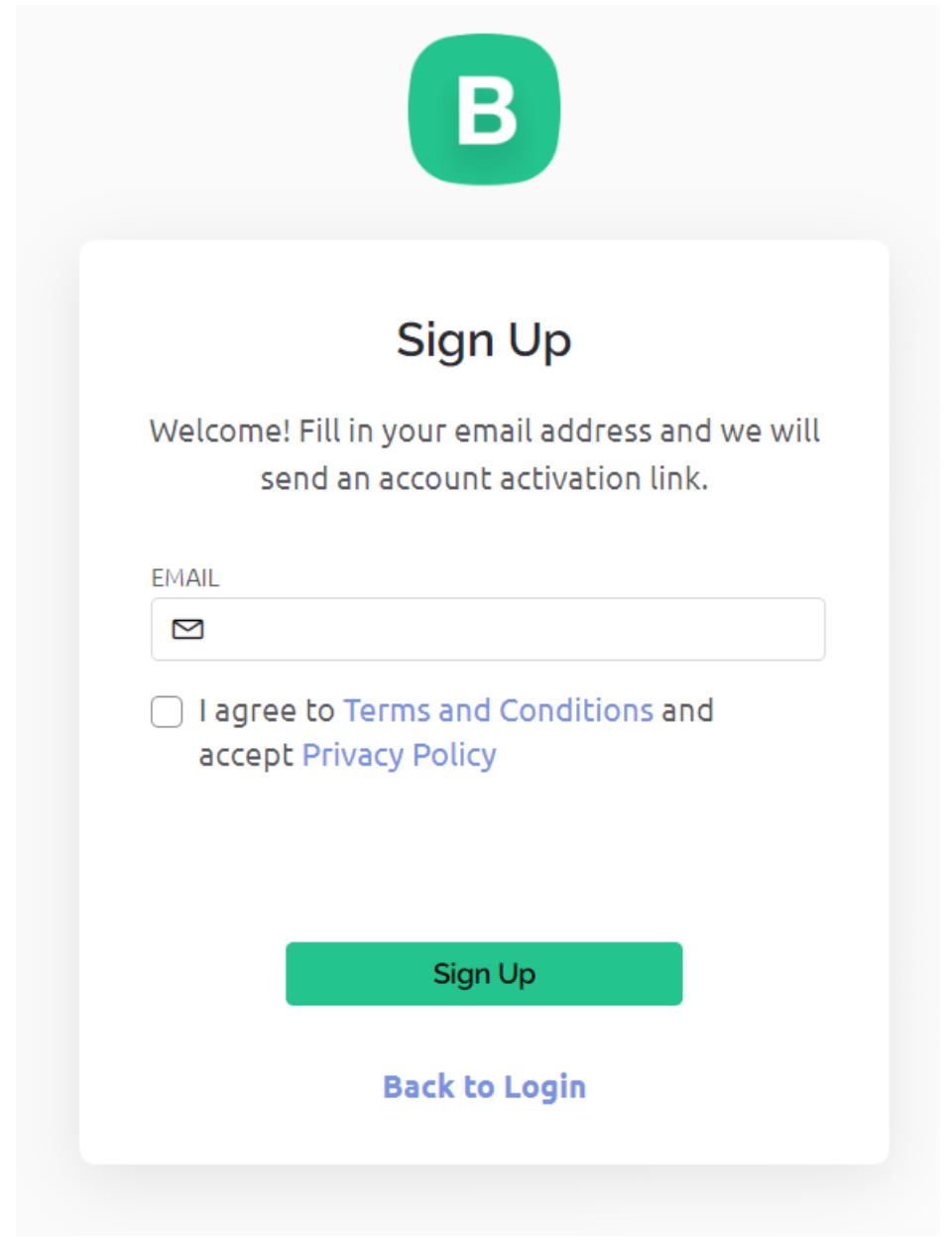
# Blynk – Plataforma Cloud

- Acesse o link  
<https://blynk.cloud/dashboard/login>
- Plataforma gratuita
- Algumas funções podem ser desbloqueadas com pagamento de versão PRO ou business
- Clique em Create new account



# Blynk – Plataforma Cloud

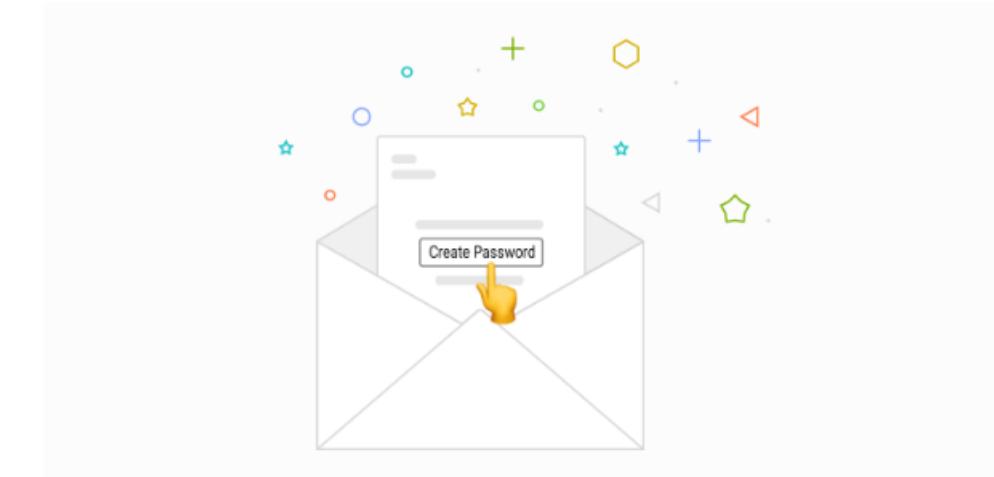
- Insira o seu email
- Aceite os termos e condições
- Clique em Sign Up



The image shows a screenshot of the Blynk sign-up page. At the top center is a green circular icon containing a white letter 'B'. Below it, the word 'Sign Up' is centered in a large, bold, dark font. Underneath, a welcome message reads: 'Welcome! Fill in your email address and we will send an account activation link.' A text input field labeled 'EMAIL' is provided for entering an email address. Below the input field is a checkbox followed by the text: 'I agree to [Terms and Conditions](#) and accept [Privacy Policy](#)'. At the bottom of the form is a green rectangular button with the text 'Sign Up' in white. Below the button is a blue link that says 'Back to Login'.

# Blynk – Plataforma Cloud

- Um email vai ser enviado para criação de senha.



## Confirm Your Email Now

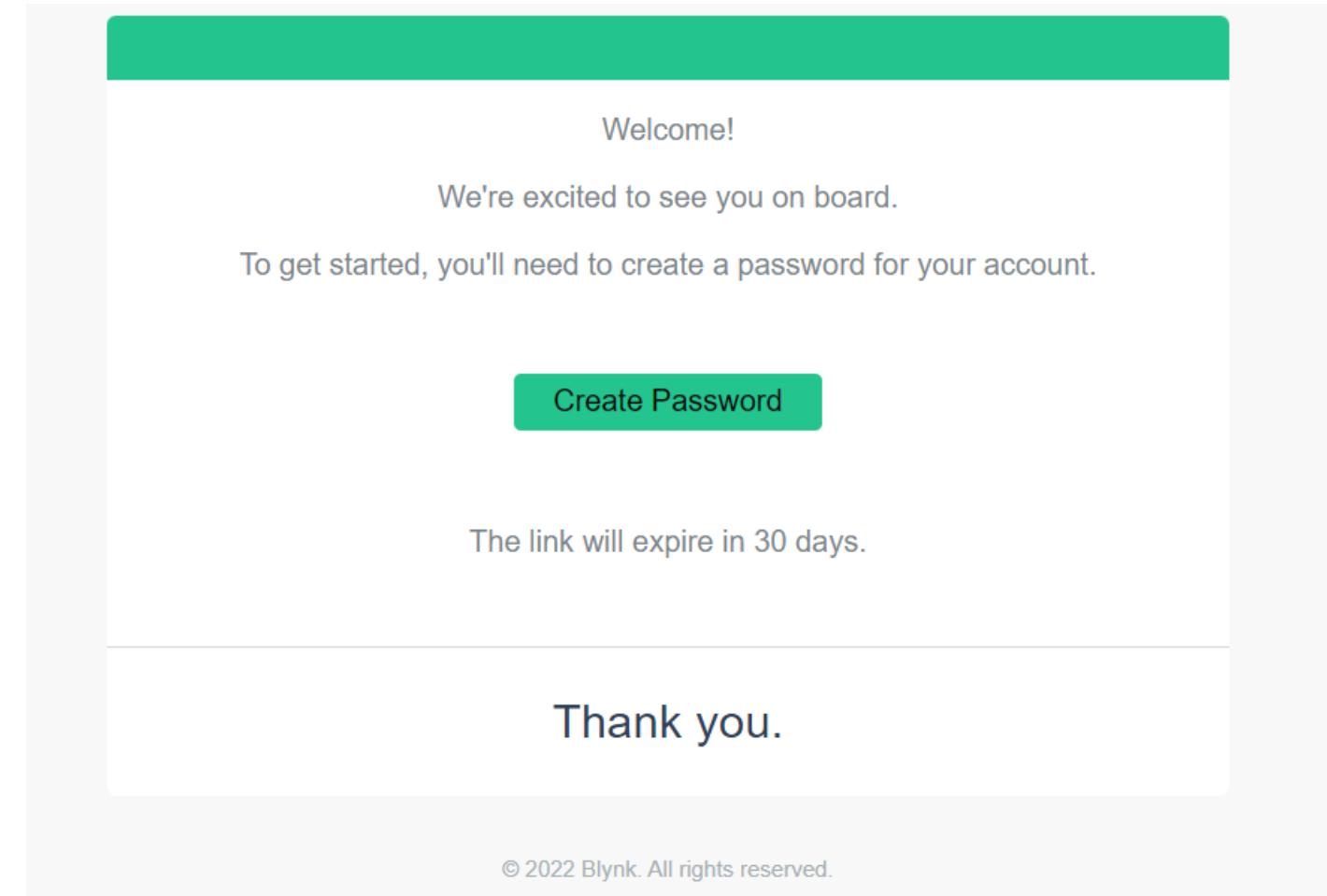
Check your inbox for an email from **Blynk**  
Click on the link there to confirm your email.

Don't see the email?

Search SPAM folder for an email from **Blynk**  
Also add it to your address book.

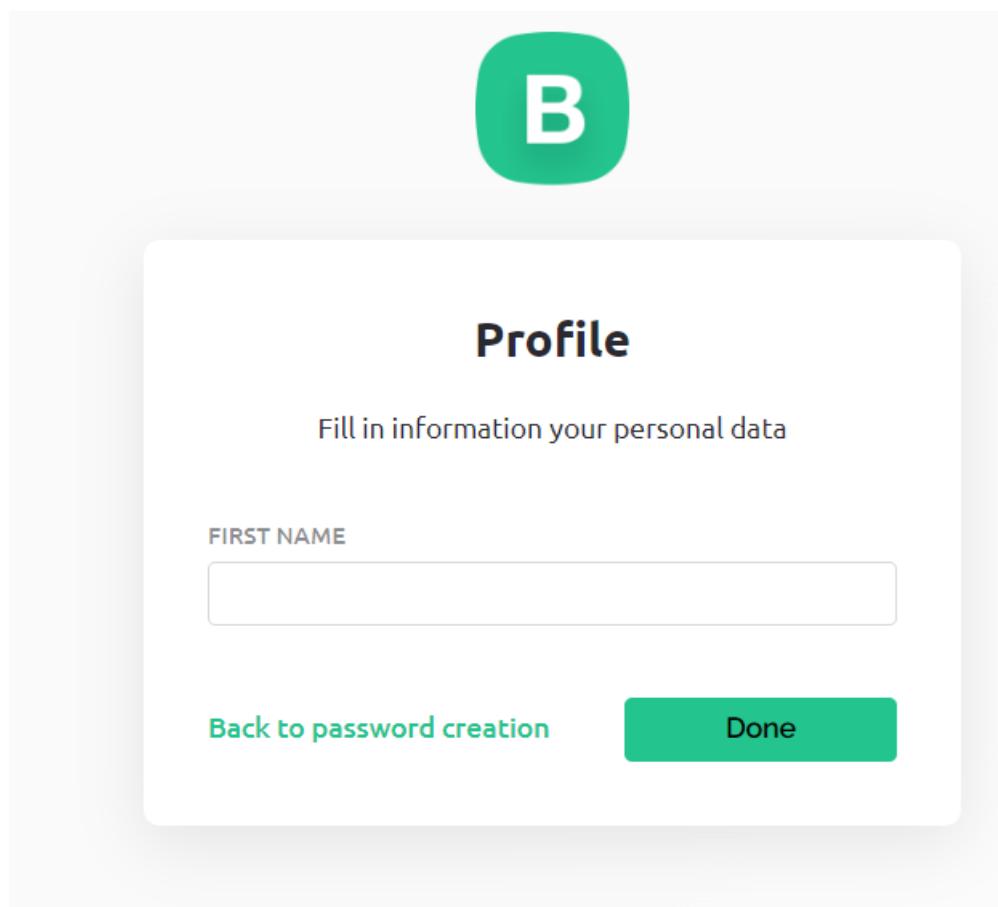
# Blynk – Plataforma Cloud

- Acesse o seu email e clique em Create Password

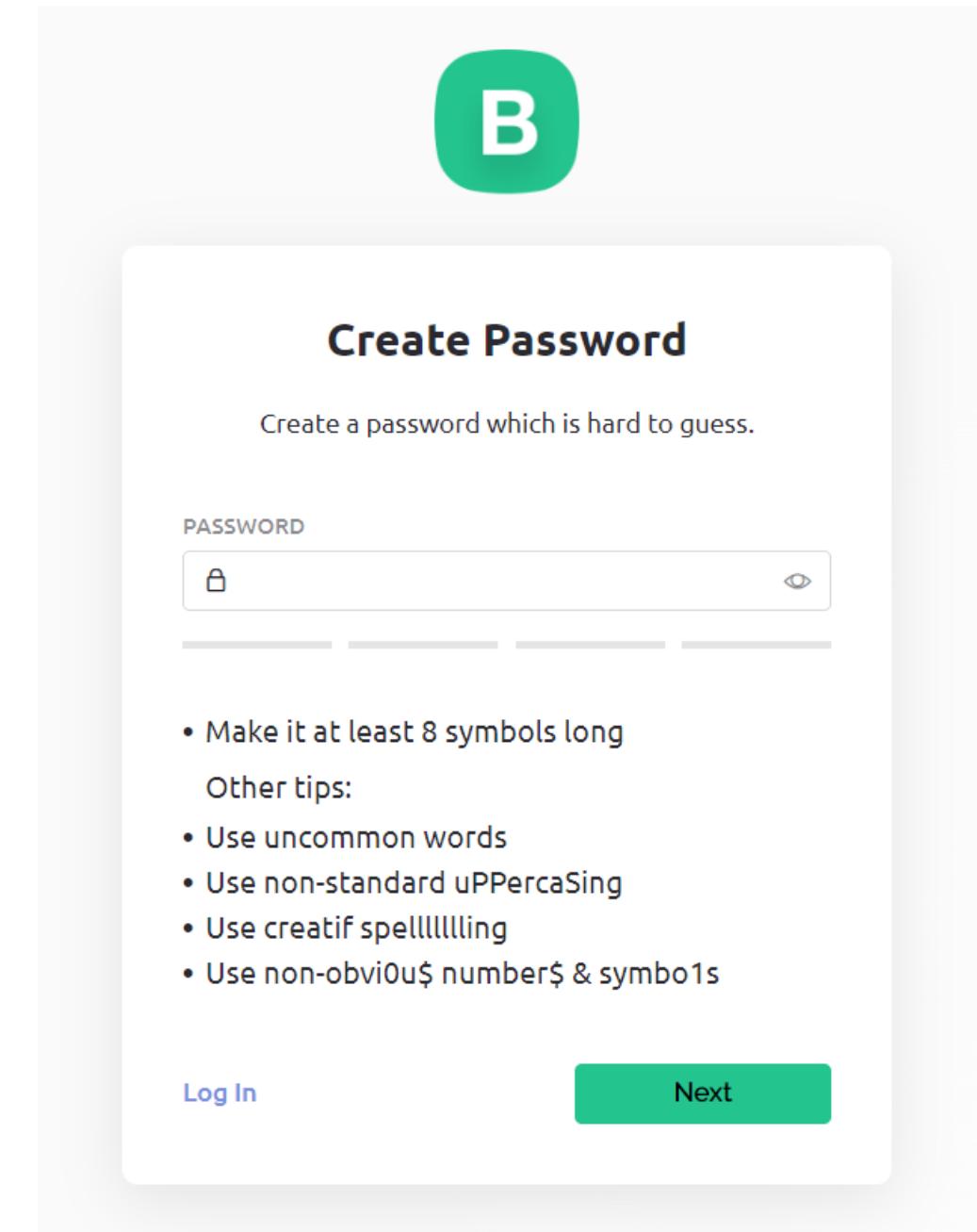


# Blynk – Plataforma Cloud

- Crie uma senha seguindo as regras e clique em next
- Após insira seu primeiro nome e clique em Done.



The screenshot shows the 'Profile' creation step of the Blynk setup process. At the top is a green circular icon with a white letter 'B'. Below it is a white card with the title 'Profile' in bold. The card contains the instruction 'Fill in information your personal data'. There is a 'FIRST NAME' field with a placeholder 'John Doe' and a 'Done' button at the bottom right. At the very bottom left is a 'Back to password creation' link.



The screenshot shows the 'Create Password' step of the Blynk setup process. At the top is a green circular icon with a white letter 'B'. Below it is a white card with the title 'Create Password' in bold. The card contains the instruction 'Create a password which is hard to guess.' and a 'PASSWORD' input field with a placeholder 'password' and an eye icon. To the right of the input field is a list of tips:

- Make it at least 8 symbols long
- Other tips:
  - Use uncommon words
  - Use non-standard uPPercASing
  - Use creatif spellllllling
  - Use non-obviou\$ number\$ & symb01s

At the bottom are 'Log In' and 'Next' buttons.

# Blynk – Plataforma Cloud

**B**

My organization

ORGANIZATION

General

Users

Locations

**Billing**

Tags

ACCESS

Roles and permissions

DEVELOPERS

Webhooks

Skip Let's go!

Region: ny3 Privacy Policy

Business \$499 /month Billed yearly Only One App Two Apps Contact us

**Blynk Tour**

1 Welcome — 2 Platform — 3 Modes — 4 Devices — 5 Template — 6 Template components — 7 Features — 8 Business

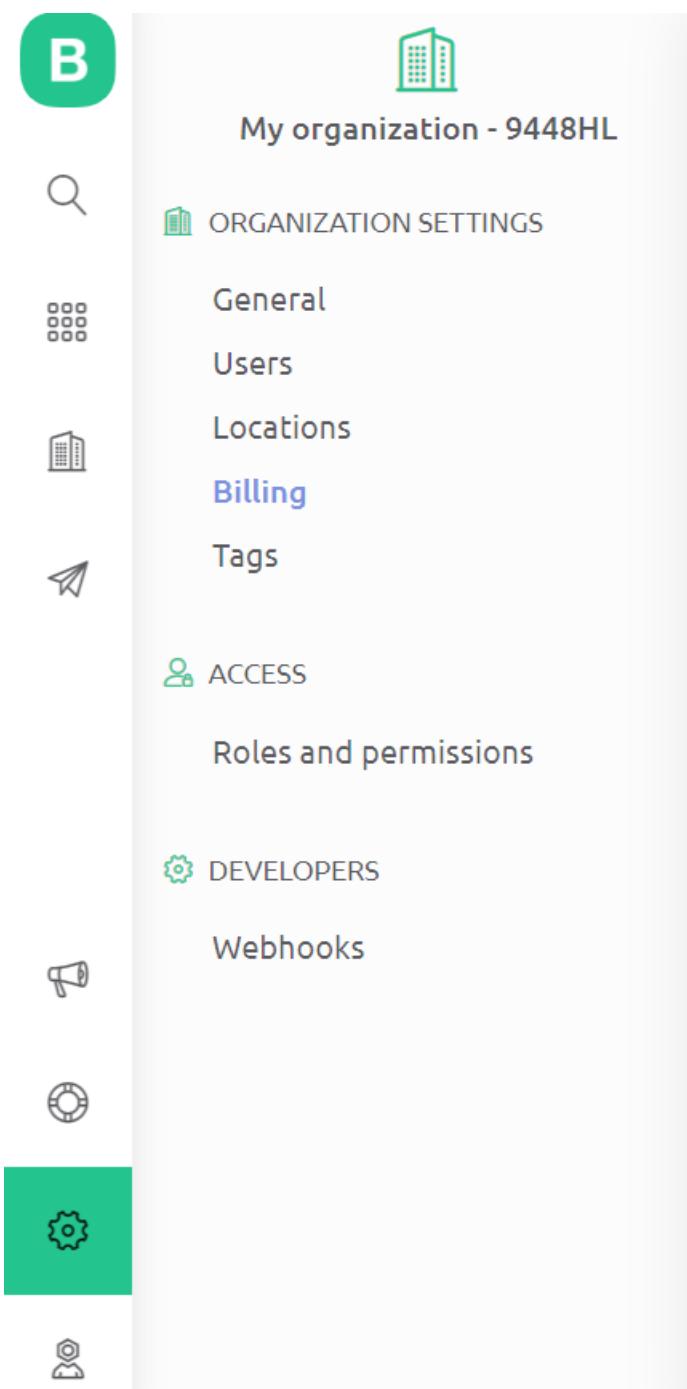
**Hi Blynker!**

You've just joined the community of more than 500,000+ developers building amazing IoT products and projects.

With Blynk you can connect your devices to the Internet and create mobile and web dashboards to control your devices from anywhere in the world.

Let's save your learning time with a few quick steps.

# Blynk – Plataforma Cloud



## Billing

You are using Free plan



## Plan Comparison

Monthly  Yearly (Save ~ 20%) [View Details](#)

Free	Plus	PRO	Business
\$0 /month Billed yearly	\$4.99 /month Billed yearly	\$42 /month Billed yearly	\$499 /month Billed yearly
Devices <a href="#">10</a> <a href="#">20</a>	Devices <a href="#">40</a> <a href="#">100</a> <a href="#">500</a>	Devices <a href="#">40</a> <a href="#">100</a> <a href="#">500</a>	Web Only One App Two Apps <a href="#">Contact us</a>

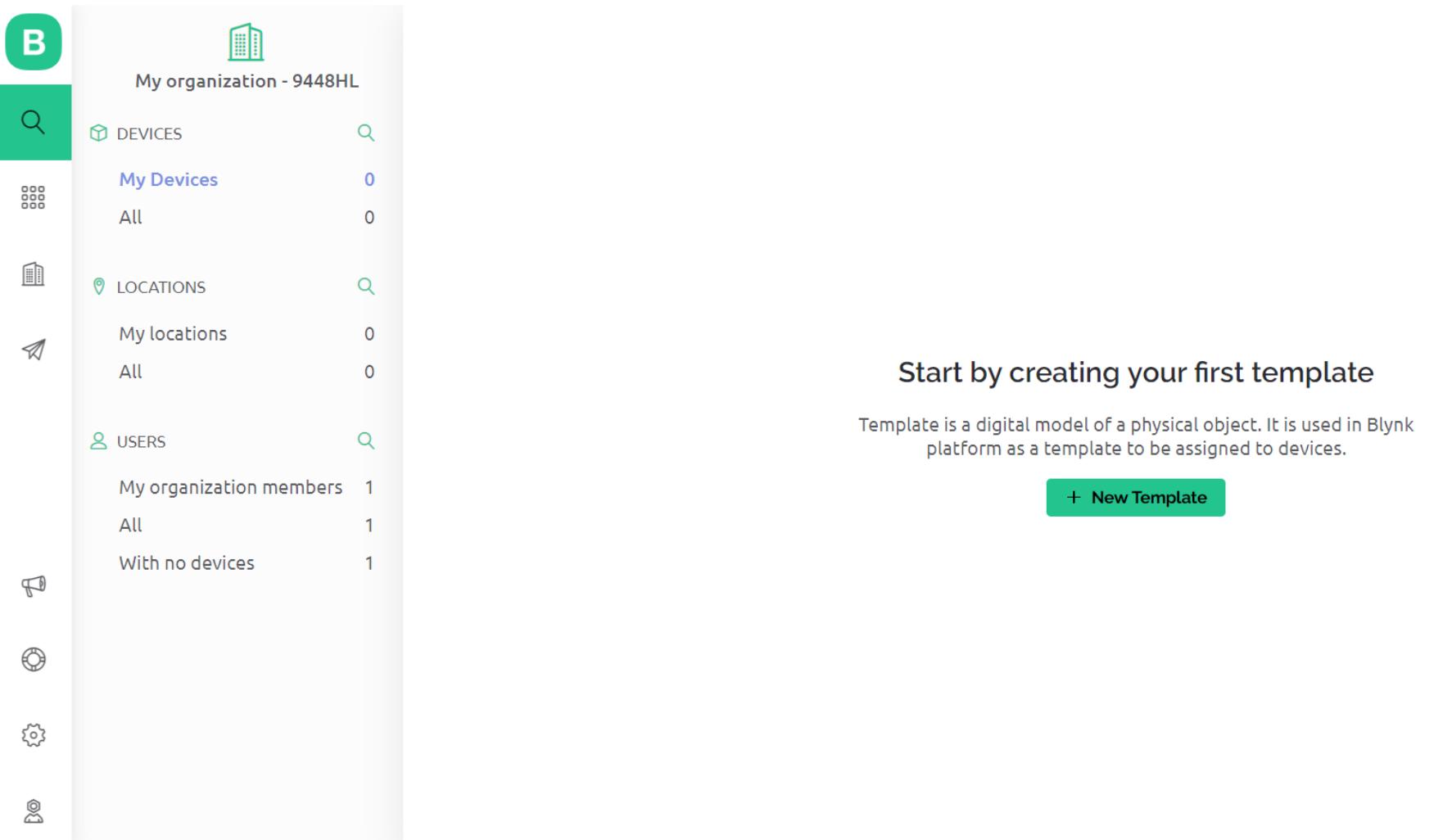
Region: ny3 [Privacy Policy](#)

Inatel

CAMINHOS  
QUE CONECTAM  
COM O FUTURO

# Blynk – Plataforma Cloud

- Vamos começar criando um modelo/template novo.
- Clique na lupa no lado esquerdo
- Clique +New Template



# Blynk – Plataforma Cloud

- Insira um nome para o modelo
- Selecione ESP32 em hardware
- Selecione WiFi em conexão

Create New Template

NAME

HARDWARE

ESP32

CONNECTION TYPE

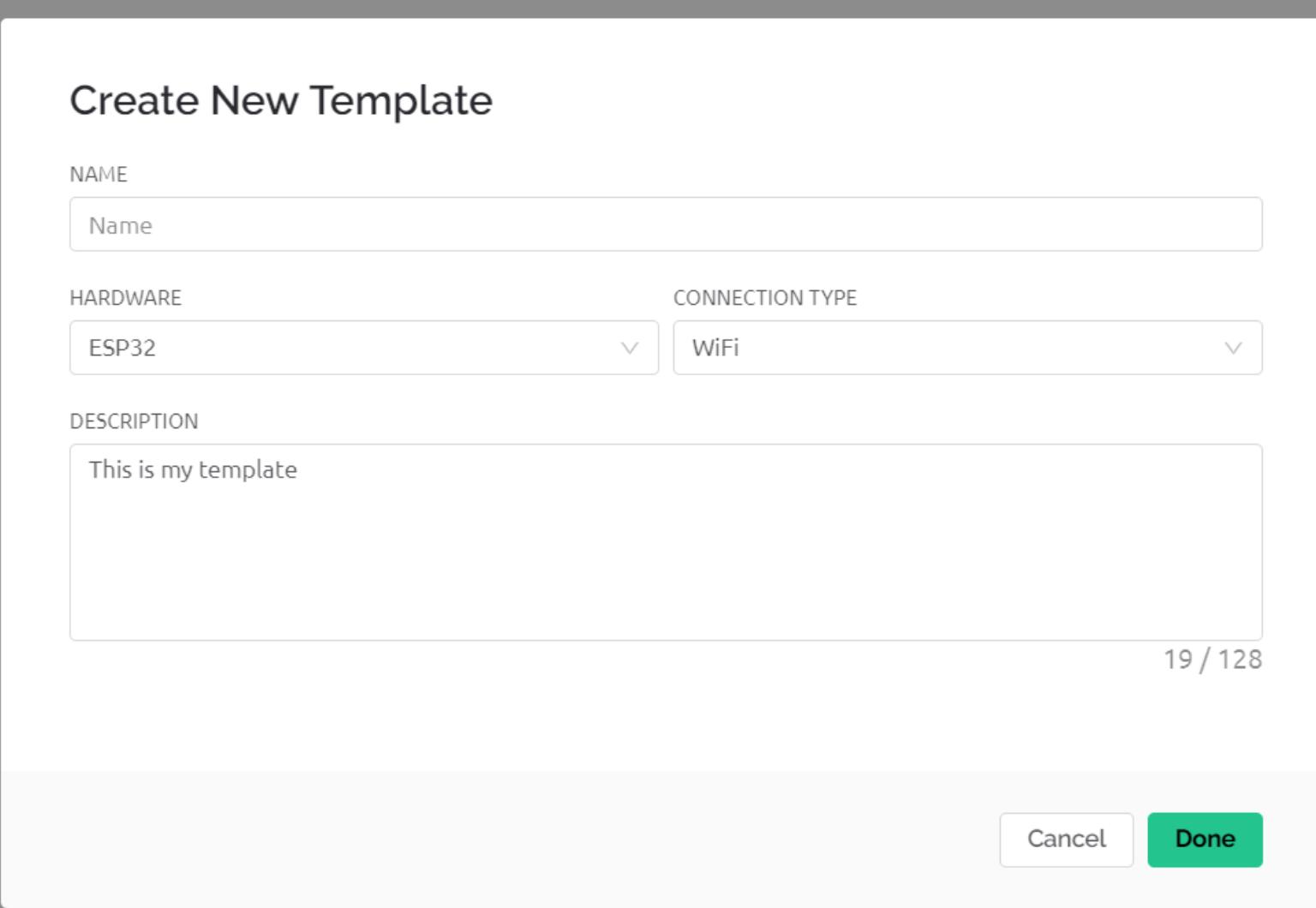
WiFi

DESCRIPTION

This is my template

19 / 128

Cancel Done



# Blynk – Plataforma Cloud

- Nesta tela aparecem as principais informações do modelo.
- Nela consta o Template ID e Device Name
- Estas informações devem ser colocadas no código da aplicação da ESP32

The screenshot shows the 'Info' tab of a device template in the Blynk Cloud Platform. The template is named 'model 1'. It is configured for 'ESP32' hardware and 'WiFi' connection type. The description is 'This is my template'. The template ID is 'TMPLmT2xktrK' and the manufacturer is 'My organization 9448HL'. The offline ignore period is set to '00 hrs 00 mins 00 secs'. The hotspot prefix is 'Hotspot Prefix'. The 'Template Image (Optional)' section has a placeholder for an image upload. The 'Firmware Configuration' section contains the following code:

```
#define BLYNK_TEMPLATE_ID "TMPLmT2xktrK"
#define BLYNK_DEVICE_NAME "model 1"
```

A note at the bottom states: 'Template ID and Device Name should be included at the top of your main firmware'.

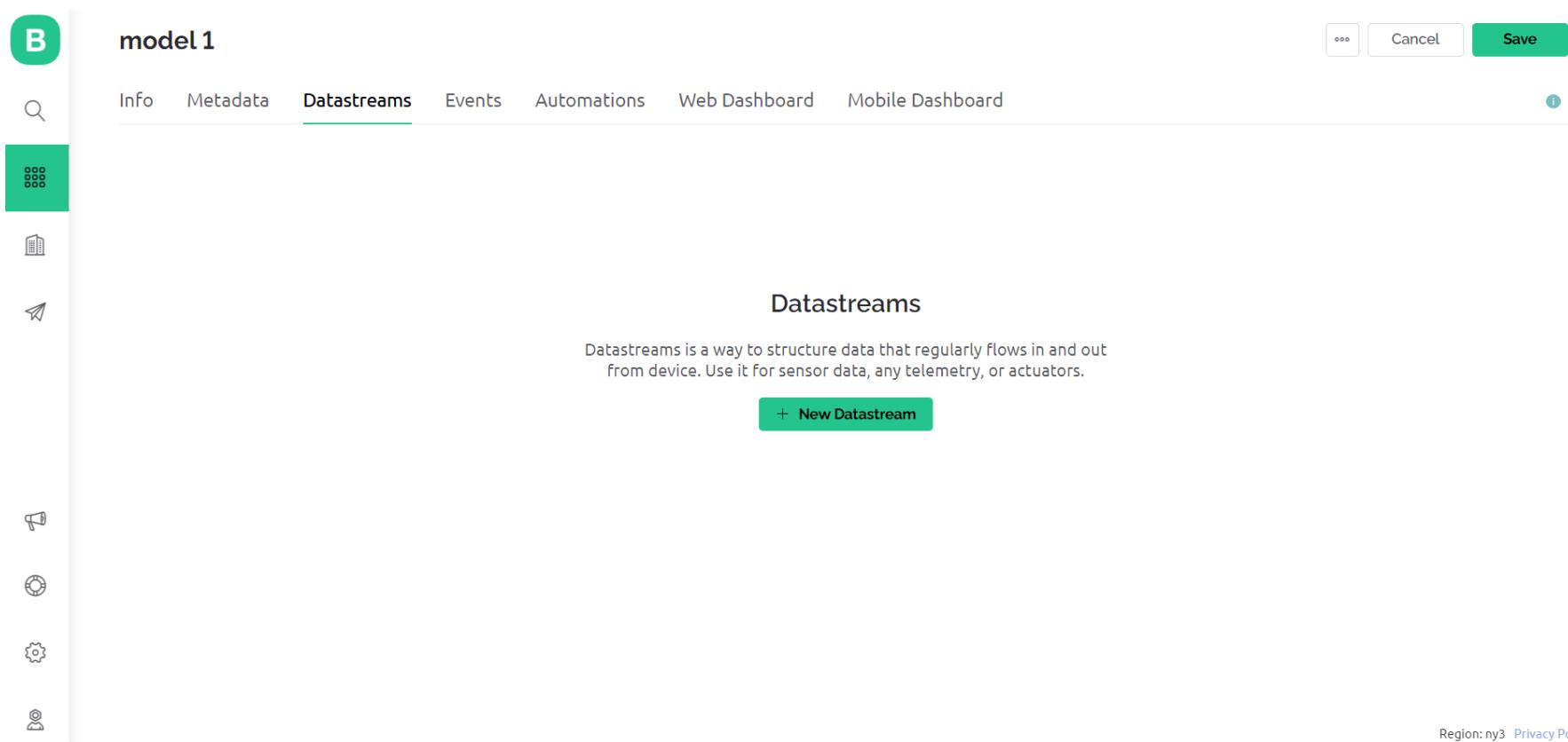
# Blynk – Plataforma Cloud

- Clique em DataStreams
- Vamos criar os fluxos de comunicação entre o dispositivo e a nuvem
- Clique em + New DataStreams

The screenshot shows the Blynk platform interface for managing DataStreams. The top navigation bar includes tabs for Info, Metadata, Datastreams (which is currently selected), Events, Automations, Web Dashboard, and Mobile Dashboard. On the left, there's a sidebar with icons for search, dashboard, and other settings. The main content area is titled "model 1" and contains a section for "Datastreams". A descriptive text explains that Datastreams are used to structure data that regularly flows in and out from a device, suitable for sensor data, telemetry, or actuators. A green button labeled "+ New Datastream" is visible. At the bottom right, it says "Region: ny3" and "Privacy Policy".

# Blynk – Plataforma Cloud

- Clique em DataStreams
- Vamos criar os canais de comunicação entre o dispositivo e a nuvem
- Clique em + New DataStreams

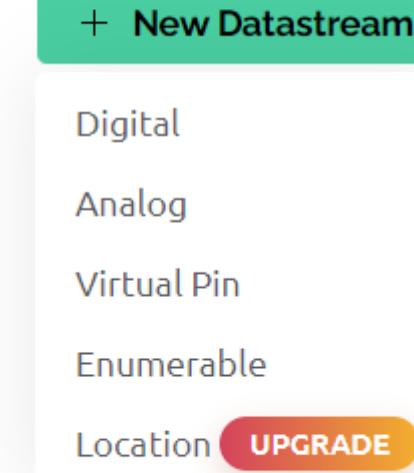


# Blynk – Plataforma Cloud

- Existem canais relacionados com
- Pinos digitais
- Pinos analógicos
- Pinos Virtuais
- Enumerable
- Localização

## Datastreams

Datastreams is a way to structure data that regularly flows in and out from device. Use it for sensor data, any telemetry, or actuators.



# Blynk – Plataforma Cloud

- Vamos criar pinos virtuais

## Virtual Pin Datastream

NAME  ALIAS  

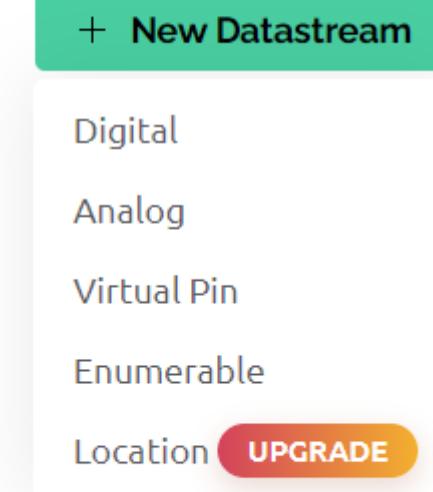
PIN  DATA TYPE

UNITS

MIN  MAX  DEFAULT VALUE

## Datastreams

Datastreams is a way to structure data that regularly flows in and out from device. Use it for sensor data, any telemetry, or actuators.



# Blynk – Plataforma Cloud

- Vamos criar pinos virtuais
- É possível criar 256 pinos de virtuais
- Podem ser do tipo inteiro, double e string
- Colocar os limites de valores

Virtual Pin Datastream

NAME	ALIAS
<input type="text" value="Integer V0"/>	<input type="text" value="Integer V0"/>

PIN	DATA TYPE
V0	Integer

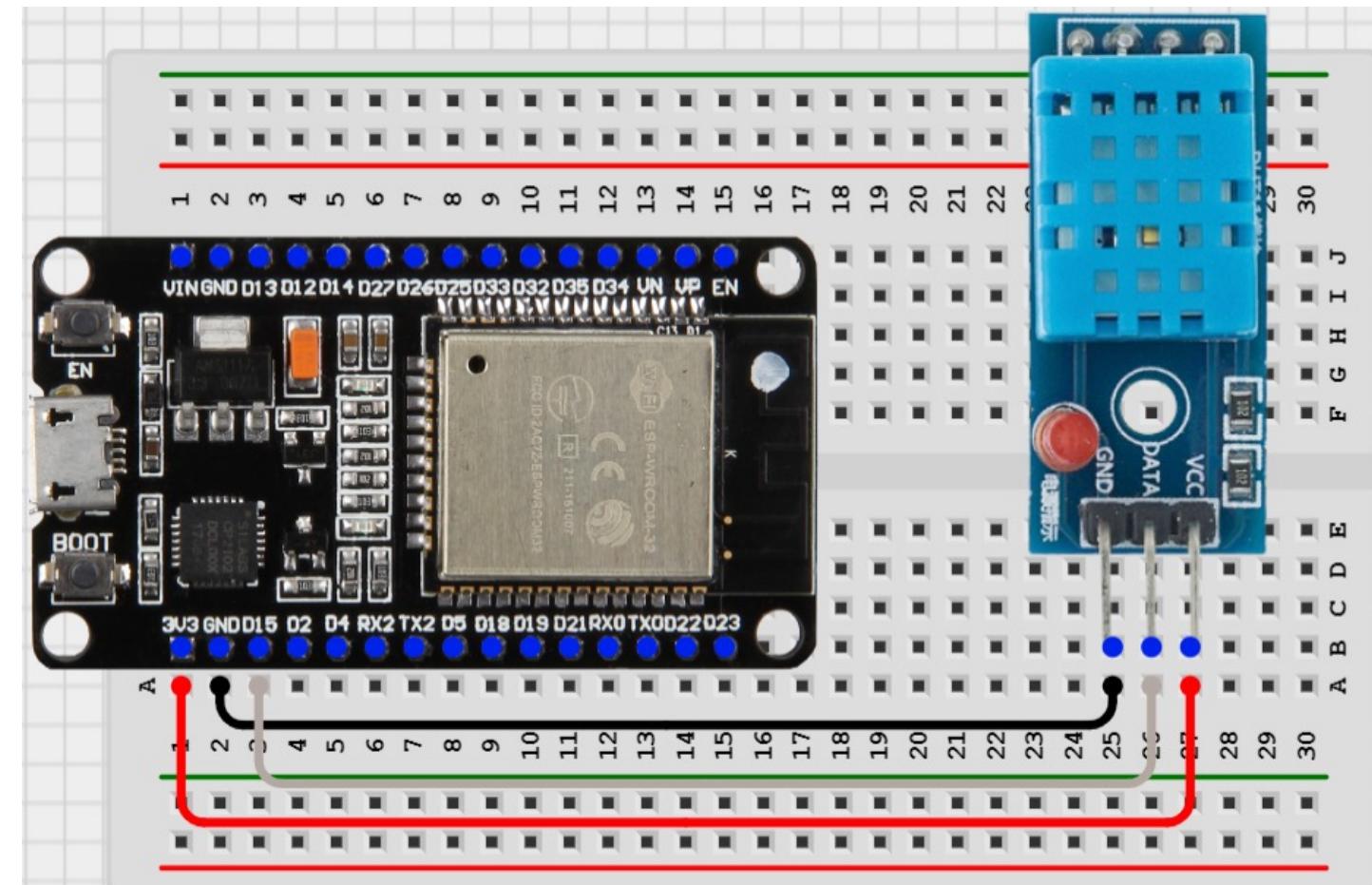
UNITS
None

MIN	MAX	DEFAULT VALUE
0	1	0

# Blynk – Plataforma Cloud

- Vamos criar pinos virtuais para comunicação dos valores de temperatura e umidade
- Abra a pasta: **blynkdht11**



# Blynk – Plataforma Cloud

## Virtual Pin Datastream

NAME	ALIAS
 Umidade	Umidade 

PIN	DATA TYPE
V5	Double 

UNITS
None 

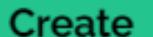
  

MIN	MAX	DECIMALS	DEFAULT VALUE
0	100	#.## 	Default Value 

 ADVANCED SETTINGS

# Blynk – Plataforma Cloud

## Virtual Pin Datastream

NAME  ALIAS  +

PIN  DATA TYPE

UNITS

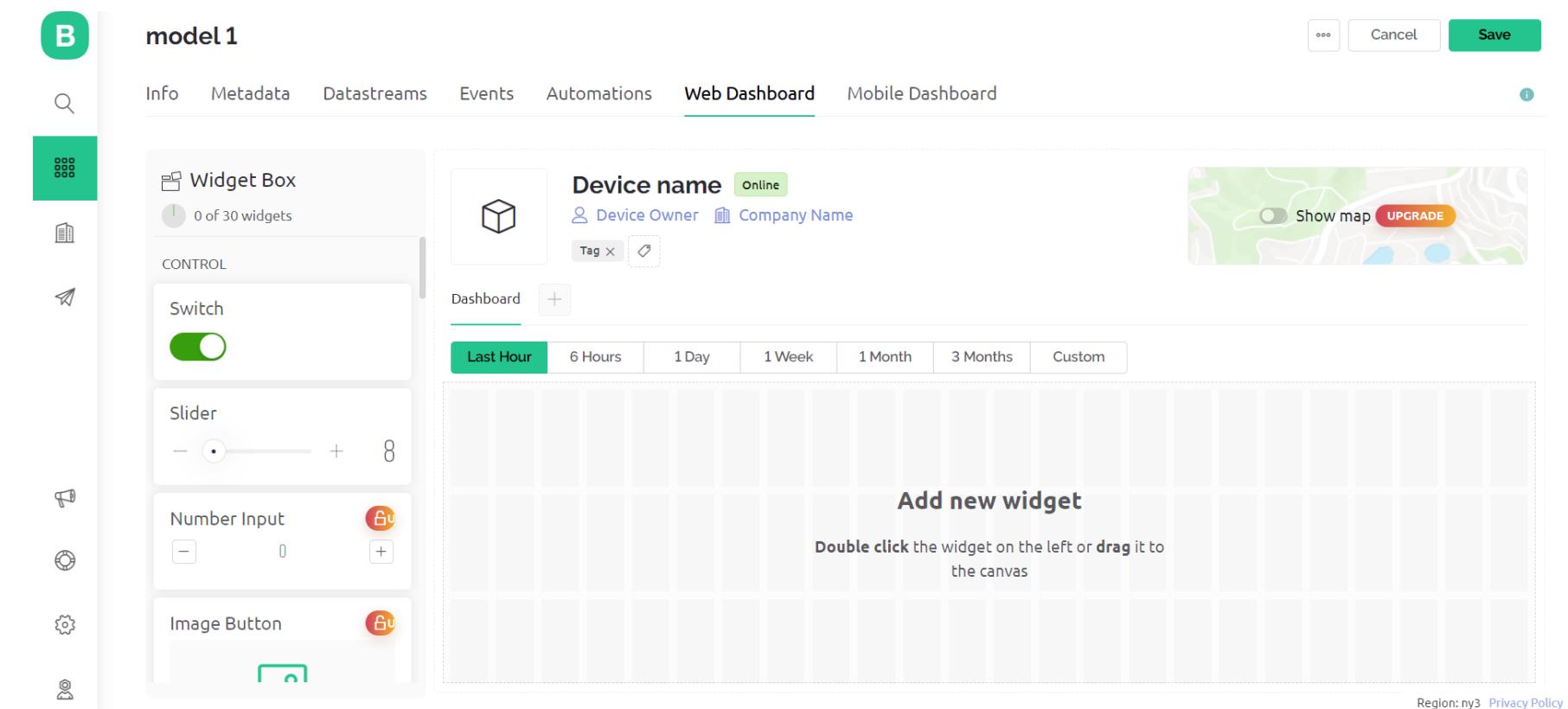
MIN  MAX  DECIMALS  DEFAULT VALUE

+ ADVANCED SETTINGS

Cancel Create

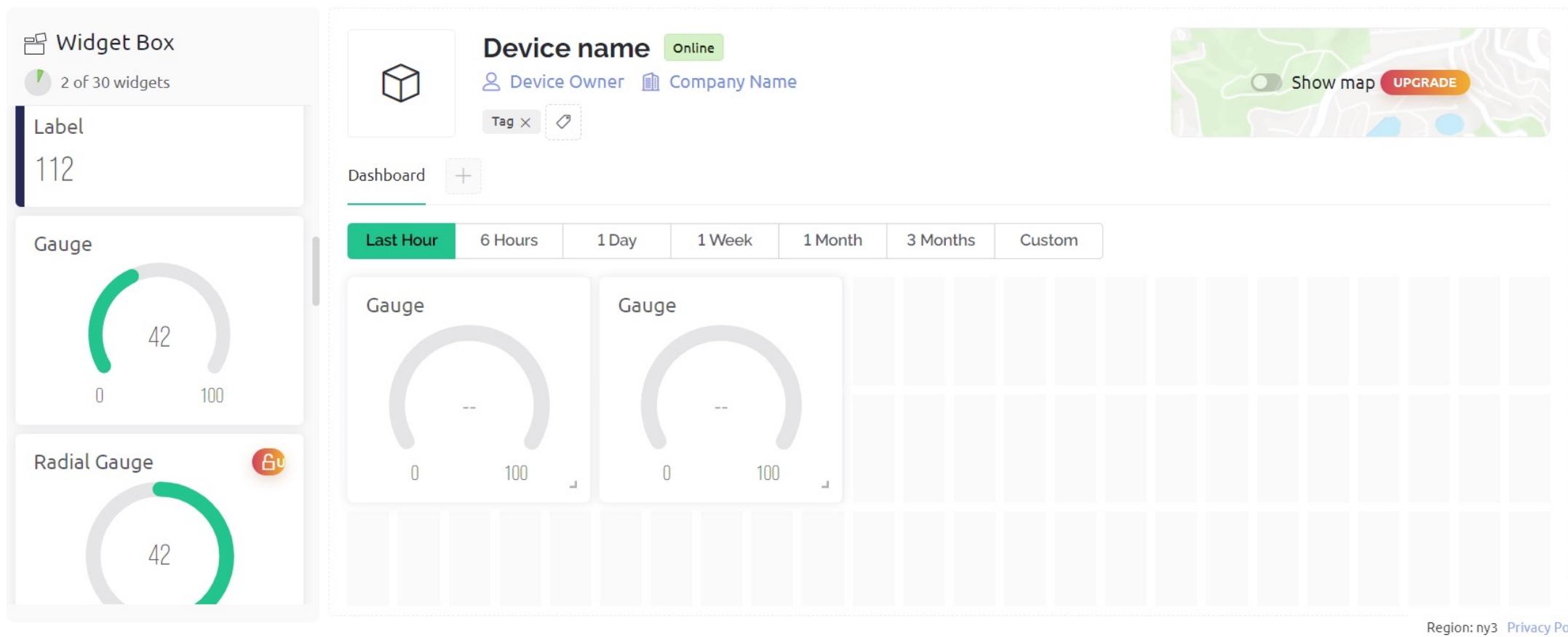
# Blynk – Plataforma Cloud

- Vamos criar um dashboard Web para ver os valores de temperatura e umidade
- No lado esquerdo há vários widgets, alguns grátis e outros disponíveis somente no plano pago.



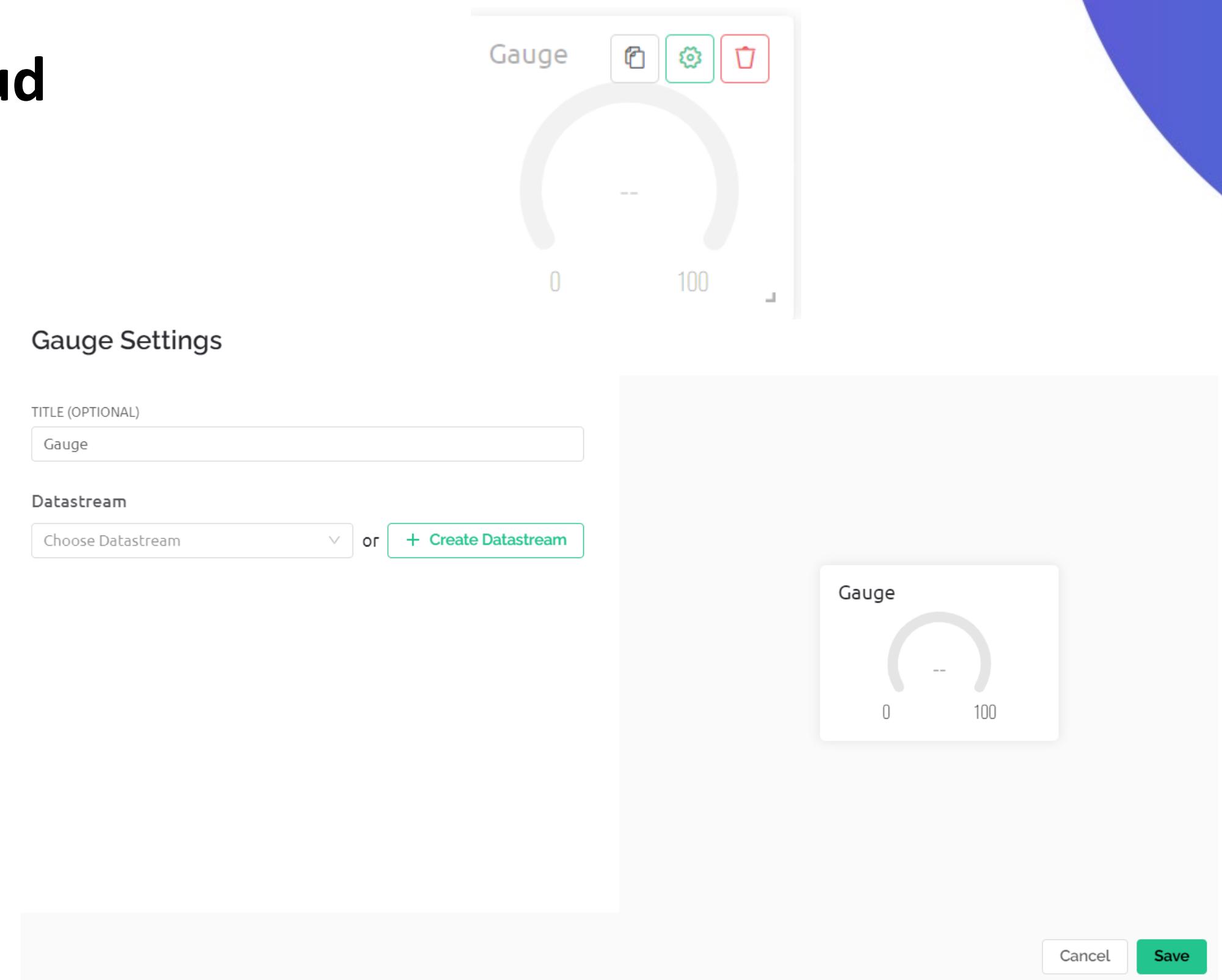
# Blynk – Plataforma Cloud

- Vamos usar medidores/Gauges para visualizar temperatura e umidade



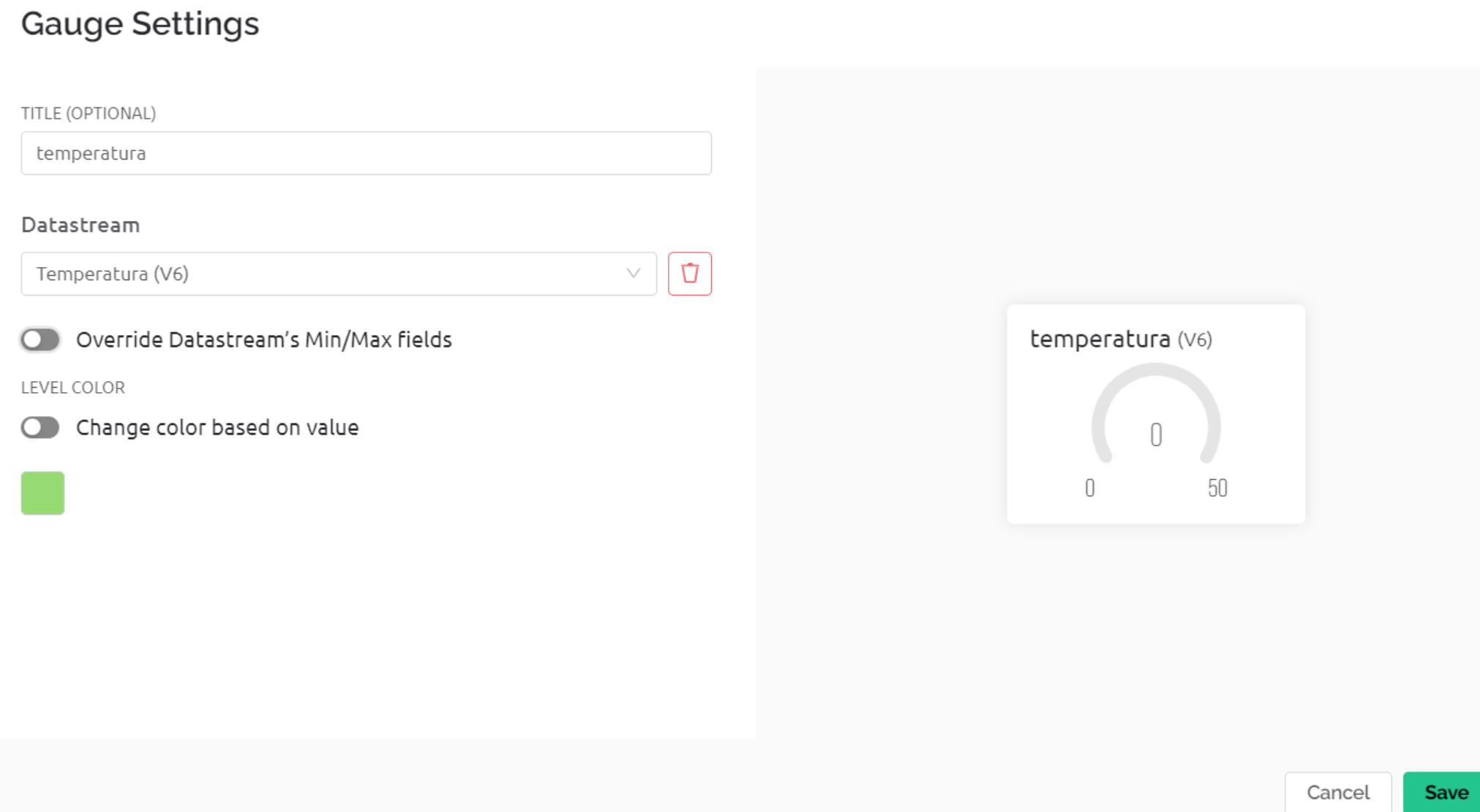
# Blynk – Plataforma Cloud

- Vamos usar medidores/Gauges para visualizar temperatura e umidade
- Aproxime o mouse e clique na engrenagem do medidor para abrir o menu ao lado
- Coloque um título para o medidor por exemplo temperatura e indique o datastream temperatura



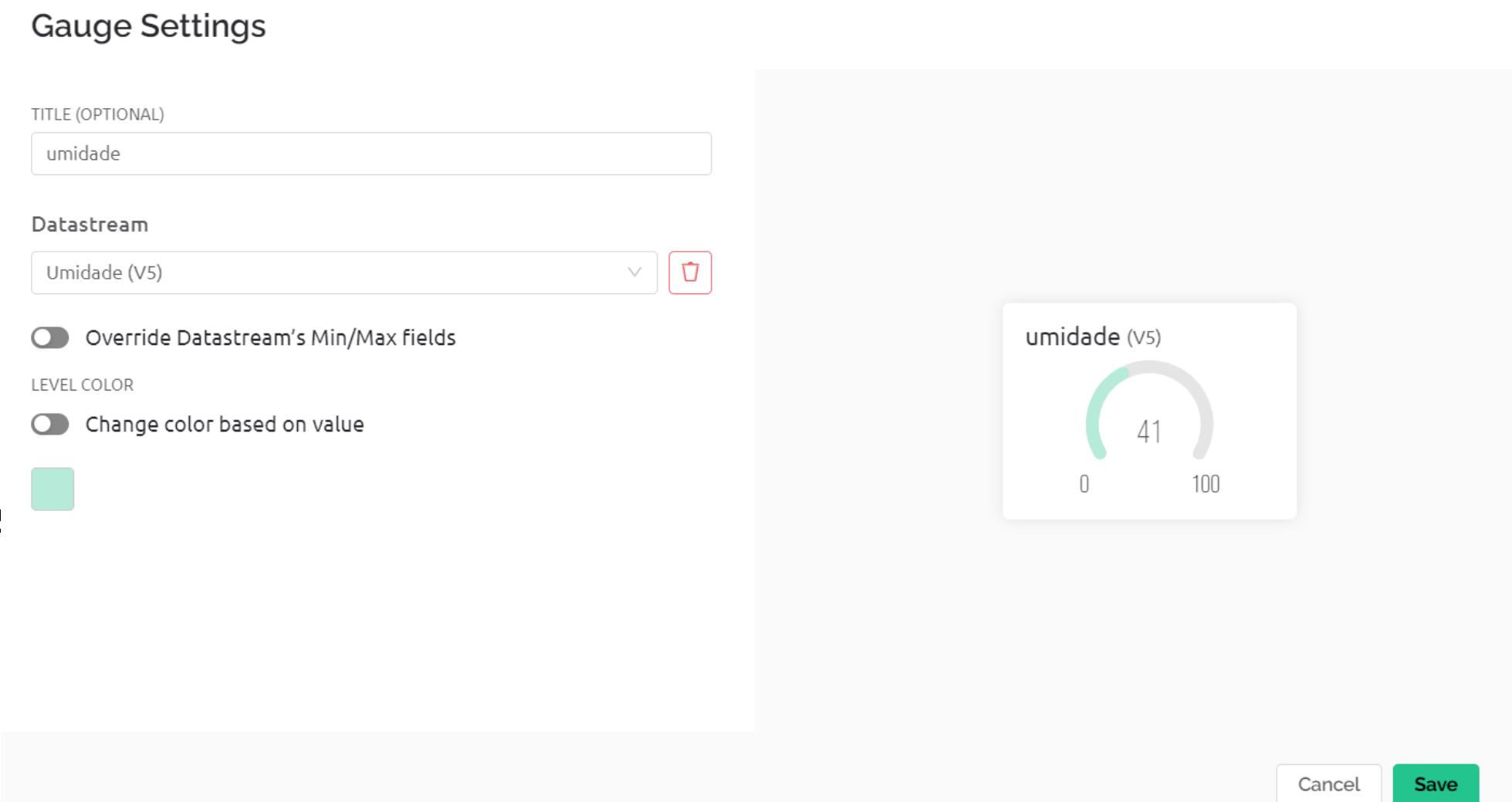
# Blynk – Plataforma Cloud

- Vamos usar medidores/Gauges para visualizar temperatura e umidade
- Aproxime o mouse e clique na engrenagem do medidor para abrir o menu ao lado
- Coloque um título para o medidor por exemplo temperatura e indique o datastream temperatura
- Clique em Save



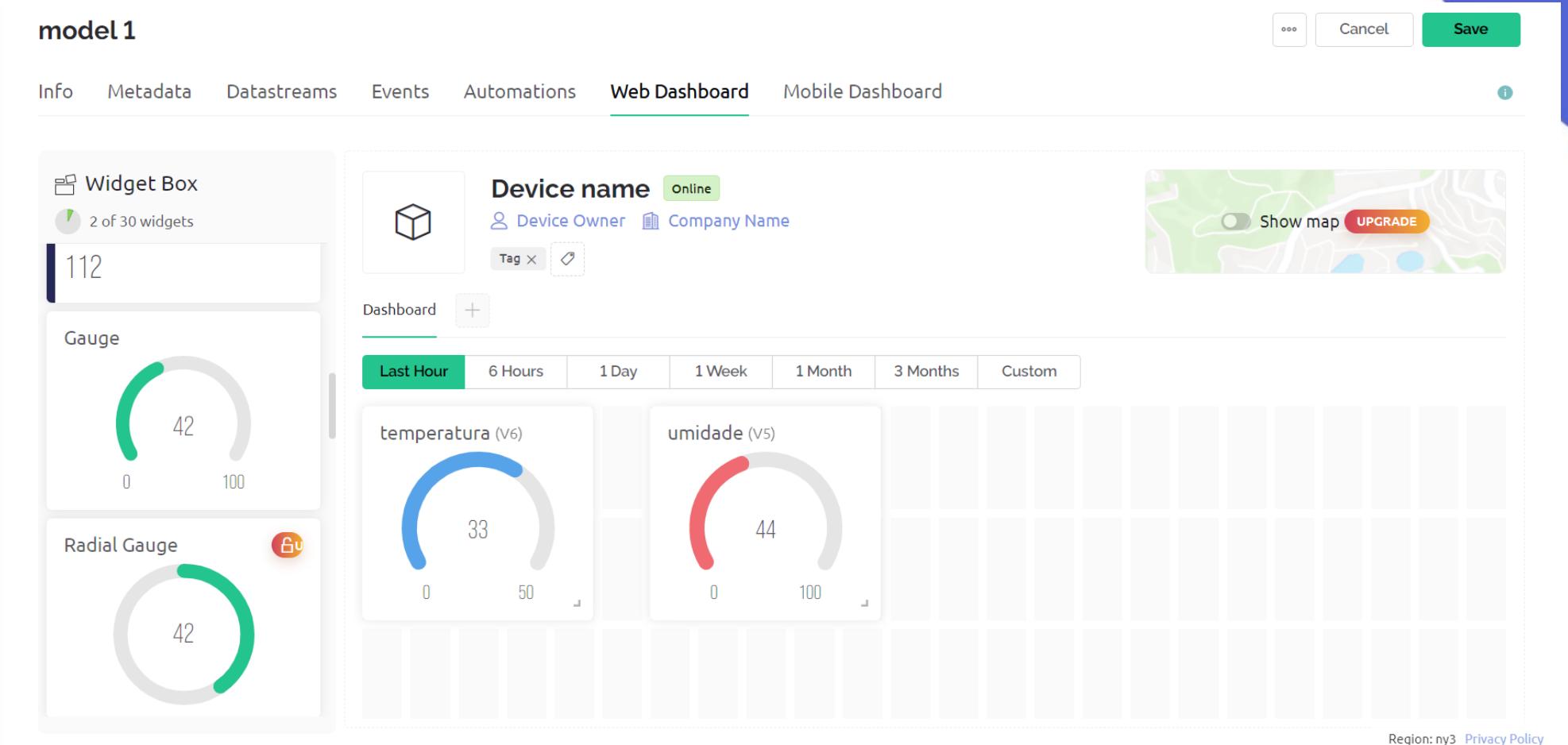
# Blynk – Plataforma Cloud

- Vamos usar medidores/Gauges para visualizar temperatura e umidade
- Aproxime o mouse e clique na engrenagem do medidor para abrir o menu ao lado
- Coloque um título para o medidor por exemplo umidade e indique o datastream umidade
- Clique em Save



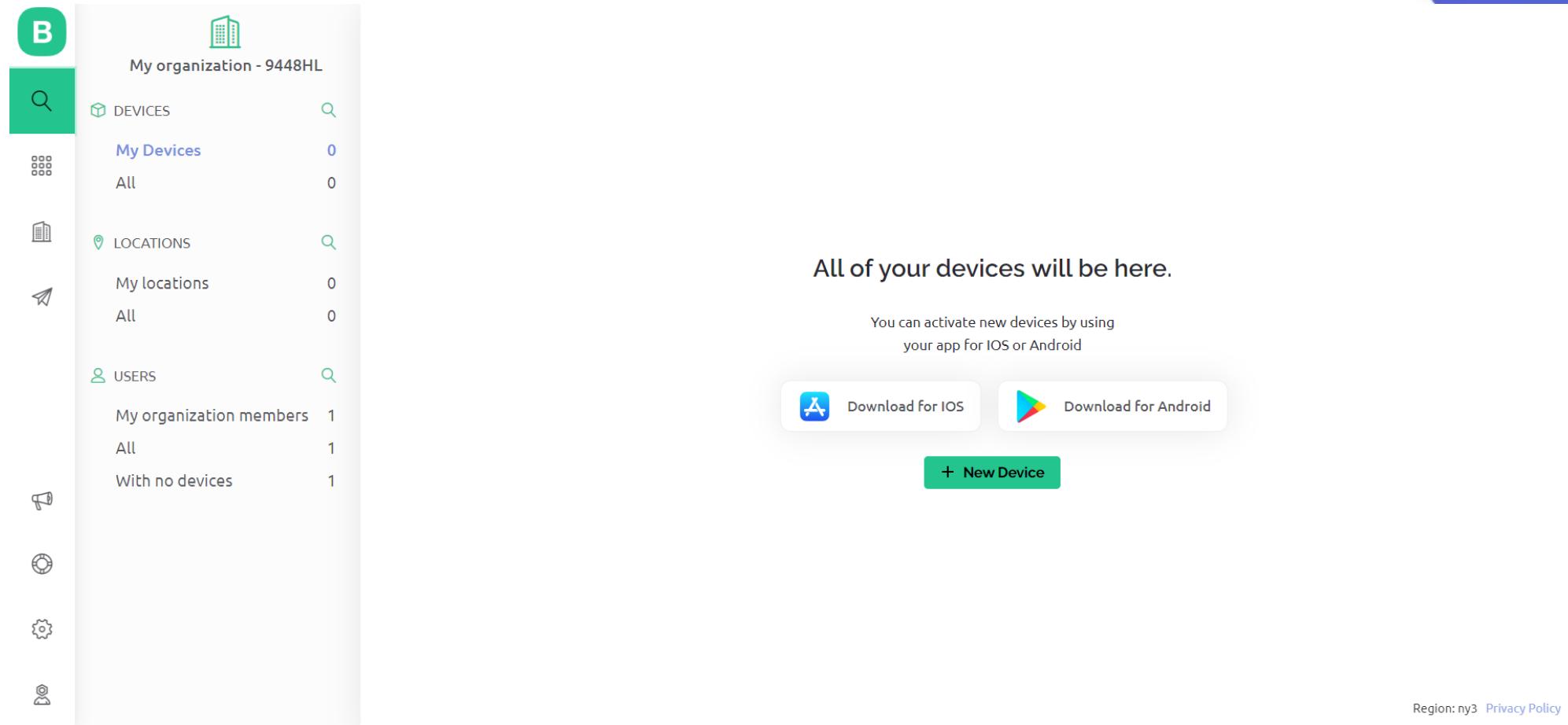
# Blynk – Plataforma Cloud

- Clique em save no canto direito para salvar o template



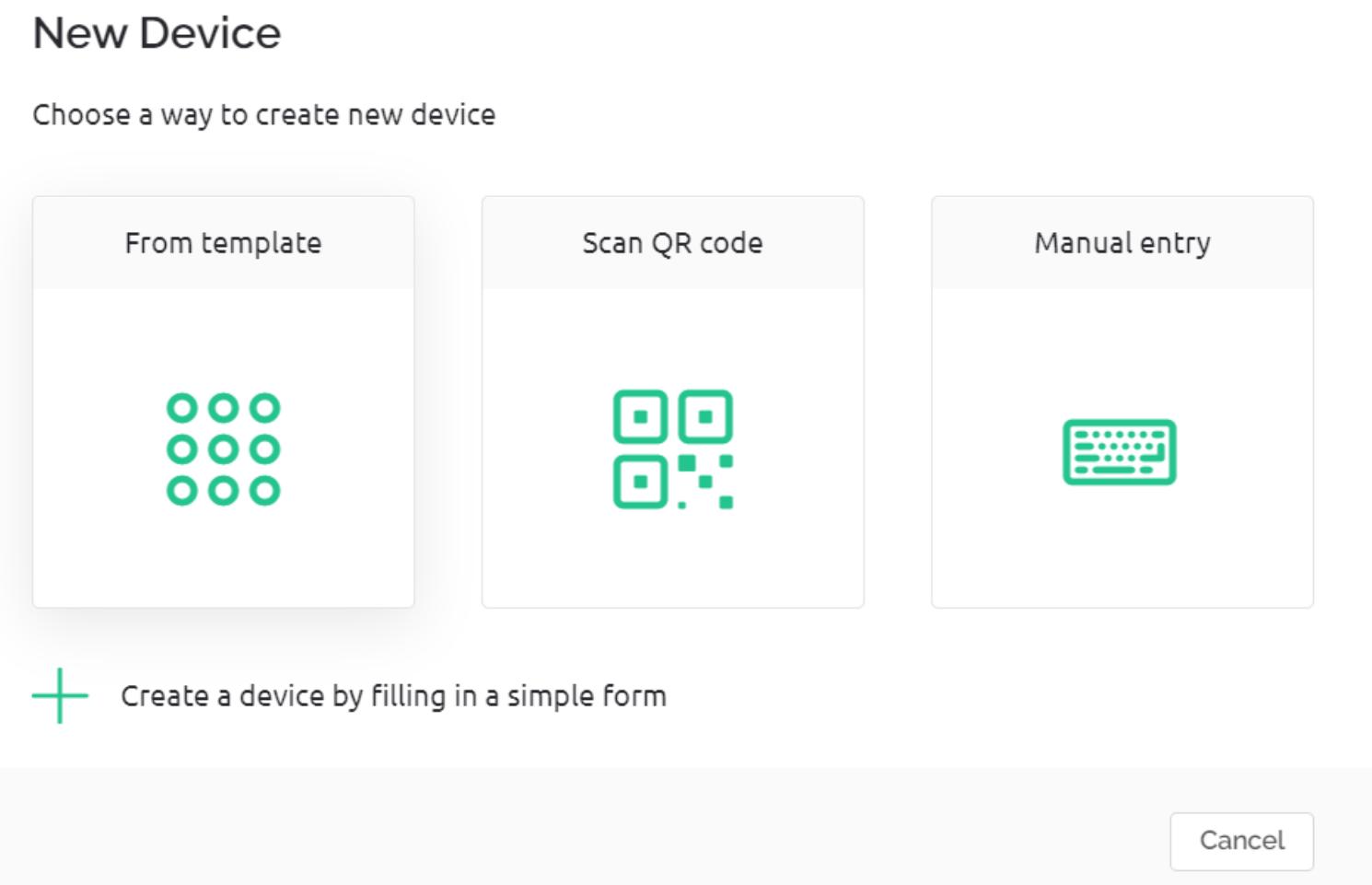
# Blynk – Plataforma Cloud

- Vamos associar um dispositivo ao template
- Clique na lupa para criar um novo dispositivo
- Clicar em +New Device



# Blynk – Plataforma Cloud

- Vamos criar dispositivo a partir de um template
- Clique em From Template



# Blynk – Plataforma Cloud

- Selecione o template criado
- O Blynk permite criar dois dispositivos
- Clique em Create

## New Device

Create new device by filling in the form below

TEMPLATE

Choose template

DEVICE NAME

New Device

Cancel

Create

# Blynk – Plataforma Cloud

The screenshot displays the Blynk Platform Cloud interface. On the left, a sidebar shows organization details: "My organization - 9448HL" with a green building icon, a search bar, and a "Back" button. Below this, there are icons for a dashboard, a device named "model 1", a megaphone, a gear, and a user profile. The main area shows a device named "model 1" (offline) owned by "Samuel". It includes tabs for "Dashboard", "Timeline", "Device Info", "Metadata", and "Actions Log". The "Last Hour" tab is selected. The dashboard features two circular gauges: one for "temperatura" (temperature) ranging from 0 to 50, and another for "umidade" (humidity) ranging from 0 to 100. A modal window titled "New Device Created!" contains the following code:

```
#define BLYNK_TEMPLATE_ID "TMPLnT2xktrK"  
#define BLYNK_DEVICE_NAME "model 1"  
#define BLYNK_AUTH_TOKEN  
"2FpCYGa1puXV0tf7U4afsW2VDaU7oDmF"
```

Below the code, a note states: "Template ID, Device Name, and AuthToken should be declared at the very top of the firmware code." There are "Documentation" and "Copy to clipboard" buttons.

Region: ny3 Privacy Policy

# Blynk – Plataforma Cloud

- É criado um token para comunicar com a cloud.

New Device Created!

X

```
#define BLYNK_TEMPLATE_ID "TMPLmT2xktrK"  
#define BLYNK_DEVICE_NAME "model 1"  
#define BLYNK_AUTH_TOKEN  
"2FpCYGa1puXv0tf7U4afsW2VDaU7oDmF"
```

Template ID, Device Name, and AuthToken should be declared at the very top of the firmware code.

 Documentation

 Copy to clipboard

# Blynk – Plataforma Cloud

Dashboard	Timeline	Device Info	Metadata	Actions Log
STATUS		LAST UPDATED		FIRMWARE CONFIGURATION
		Not updated yet		
DEVICE ACTIVATED		ORGANIZATION		
5:27 PM Today by samuelbmafra@inatel.br		My organization - 9448HL		
AUTHTOKEN		TEMPLATE NAME		
2FpC - -----		model 1		
MANUFACTURER				
My organization 9448HL				
SSL				
No SSL				
BOARD TYPE				
ESP32				

# Blynk – Código

- O template\_ID e device\_name são colocados no topo do código
- Substituir pelos valores obtidos por você
- Assim como o token de autenticação

```
#define BLYNK_TEMPLATE_ID "TMPLmt2xktrK"
#define BLYNK_DEVICE_NAME "model 1"
#define BLYNK_PRINT Serial
#include "DHTesp.h"
#include <BlynkSimpleEsp32.h>

//Coloque abaixo o token de autorizacao recebido por email
char auth[] = "2FpCYGa1puXV0tf7U4afsW2VDaU7oDmF";

//Preencha com os dados da sua rede wifi
char ssid[] = "WLL-Inatel";
char pass[] = "inatelsemfio";

//Definicoes de pino DHT
#define pino_DHT 15

BlynkTimer timer;

DHTesp dht;

int temperatura;
```

# Blynk – Código

- Criação das funções no blynk
- São associados os valores obtidos de temperatura e umidade com os pinos virtuais V5 e v6.

```
BlynkTimer timer;  
  
DHTesp dht;  
  
int temperatura;  
int umidade;  
  
void enviardados()  
{  
    //Le os dados do sensor  
    temperatura = dht.getTemperature();  
    umidade = dht.getHumidity();  
  
    //Envia os dados para o Blynk  
    Blynk.virtualWrite(V6, temperatura);  
    Blynk.virtualWrite(V5, umidade);  
}  
  
void setup()  
{  
    //Inicializa a comunicacao com o Blynk
```

# Blynk – Código

- A função Blynk. begin faz a autenticação do device
- A função Blynk.run estabelece a comunicação.
- Inicia o temporizador

```
void setup()
{
    //Inicializa a comunicacao com o Blynk
    Blynk.begin(auth, ssid, pass);

    //Inicializa o DHT11
    dht.setup(pino_DHT, DHTesp::DHT11);

    //Envia as informacoes para o smartphone
    timer.setInterval(5000L, enviardados);
}

void loop()
{
    Blynk.run();
    timer.run();
}
```

# Blynk – Resultado

The screenshot shows the Blynk app interface for monitoring a device named "model 1".

**Left Sidebar:** Contains icons for Back, Search, Grid View, Device List (1 Device), Share, Megaphone, Circular Progress, Gear, and User.

**Top Header:** Shows "My organization - 9448HL" and a "Back" button.

**Device Overview:** Displays "model 1" status as "Online", owned by "Samuel" from "My organization - 9448HL", with an "Add Tag" option.

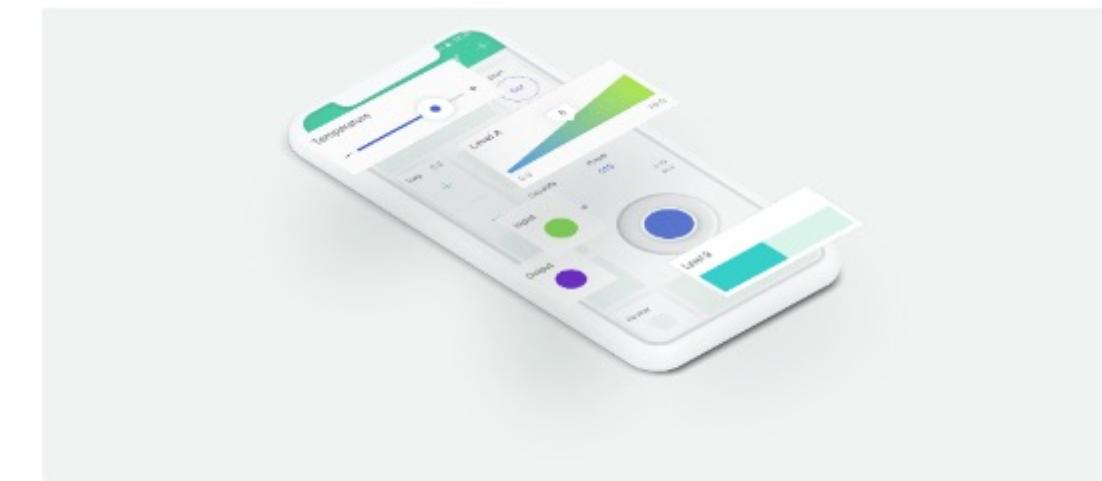
**Dashboard Navigation:** Includes tabs for Dashboard, Timeline, Device Info, Metadata, and Actions Log. The "Last Hour" tab is selected.

**Metrics:** Two circular gauges show "temperatura" at 24 and "umidade" at 67.

**Bottom Right:** Shows "Region: ny3" and "Privacy Policy".

# Blynk – Versão Mobile

- Blynk IoT app



How to create mobile dashboard?

1. Open Blynk.App
2. Log In to your account
3. Switch to Developer Mode (🔍 at the top)
4. Find a template you just created on the web and tap on it.

[Documentation](#)

Don't have the app?



[Download for iOS](#)



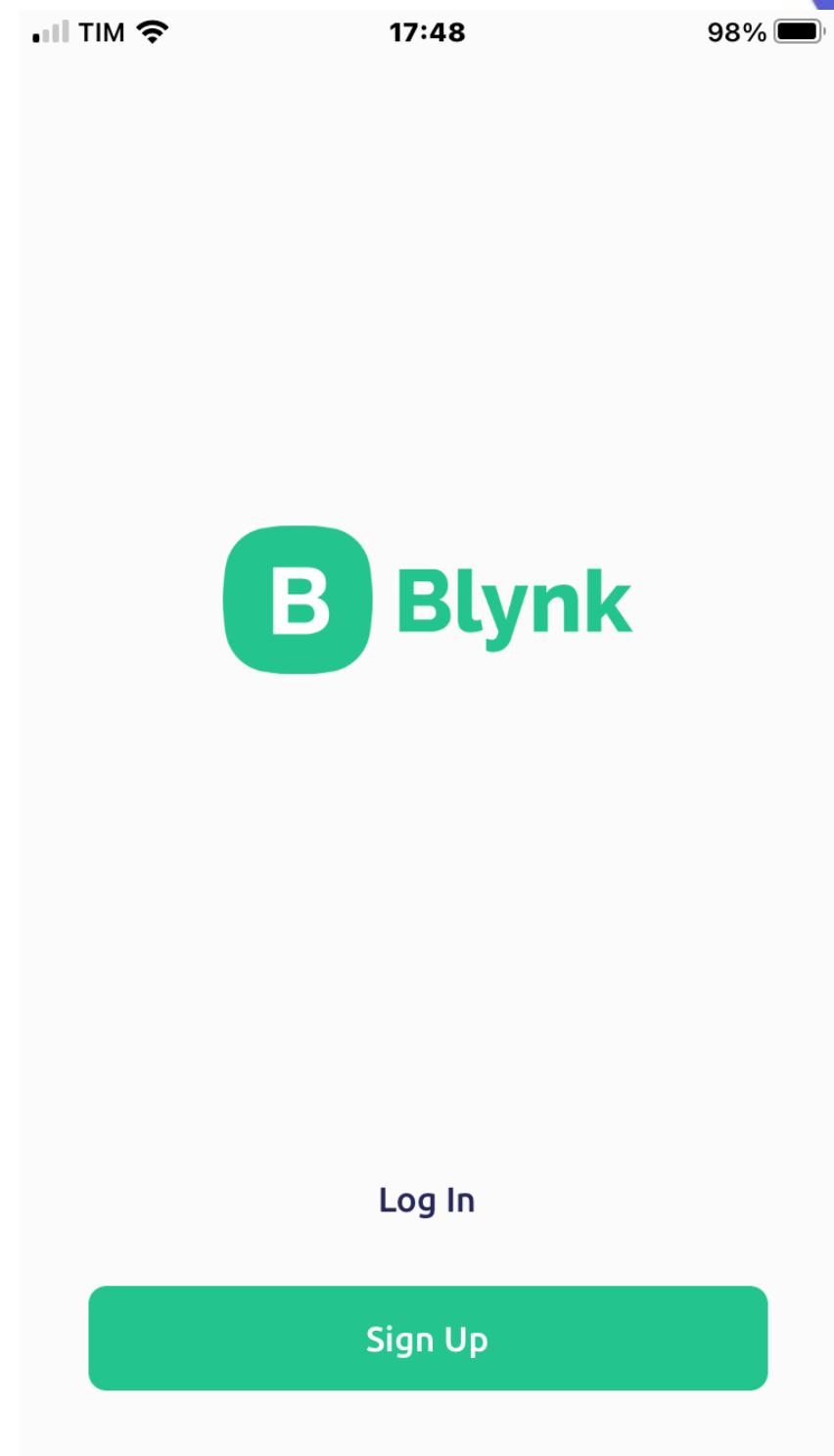
[Download for Android](#)

**Inatel**

CAMINHOS  
QUE CONECTAM  
COM O FUTURO

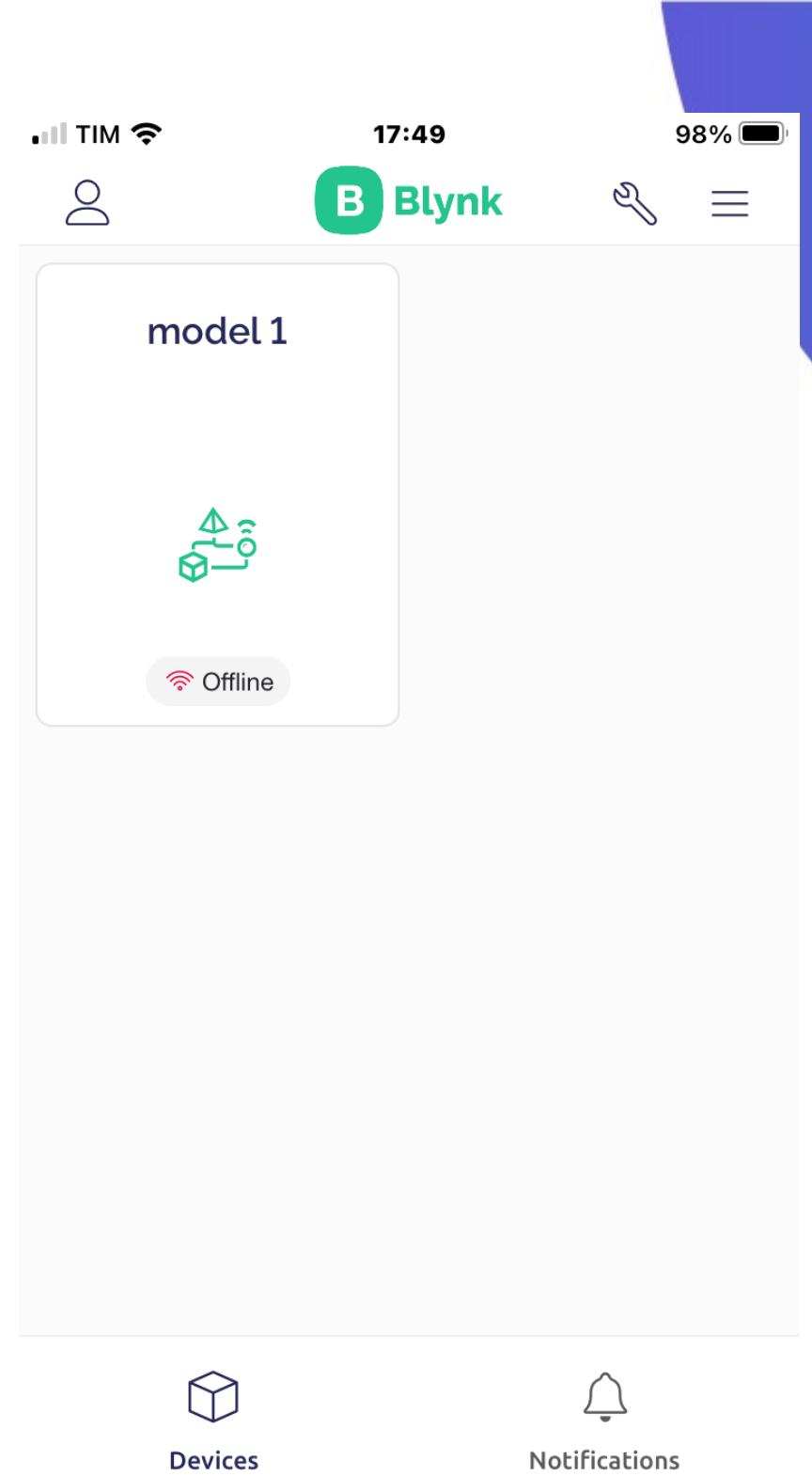
# Blynk – Versão Mobile

- Fazer um login no aplicativo



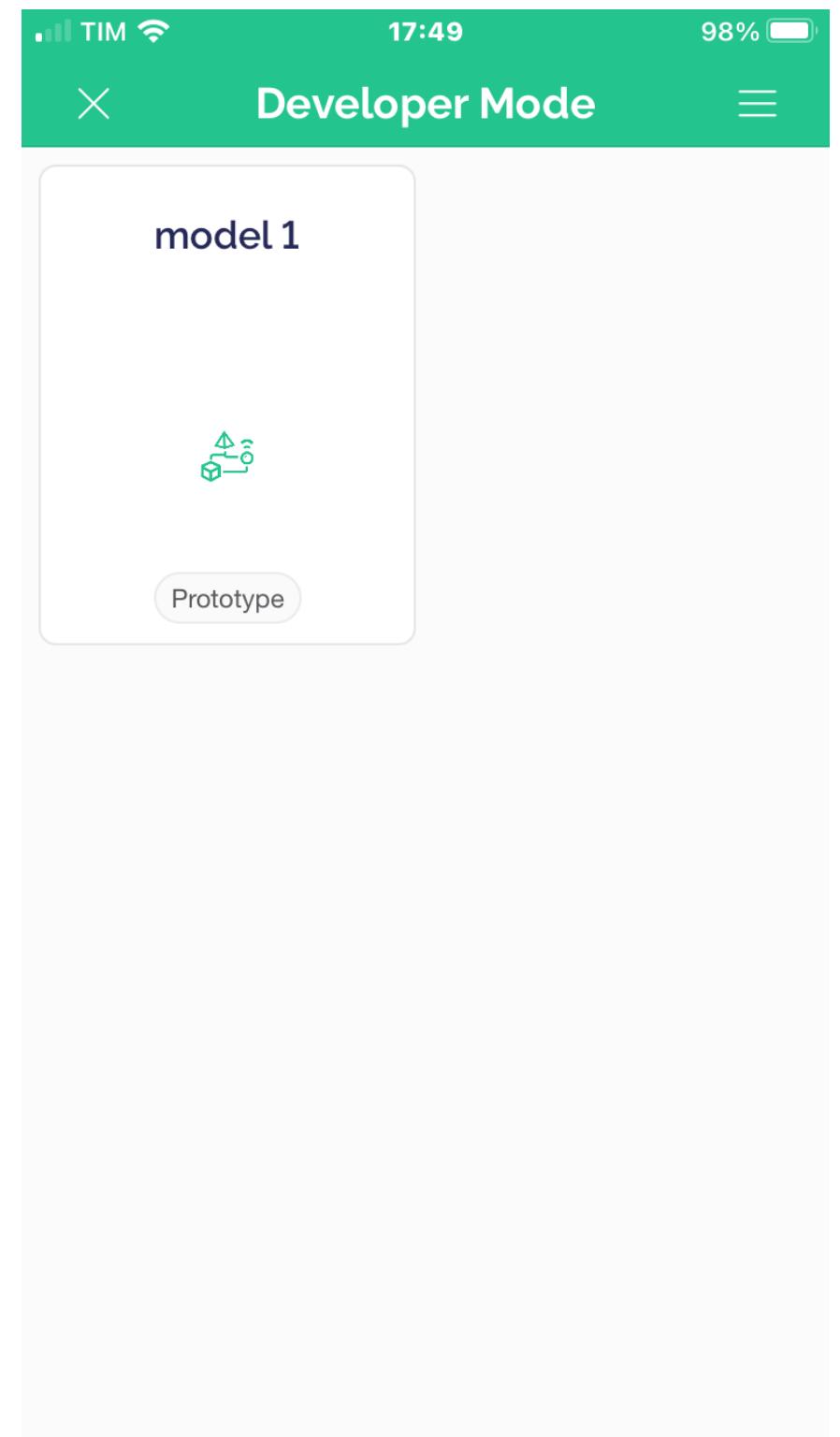
# Blynk – Versão Mobile

- Clique na ferramenta para abrir o modo de desenvolvimento



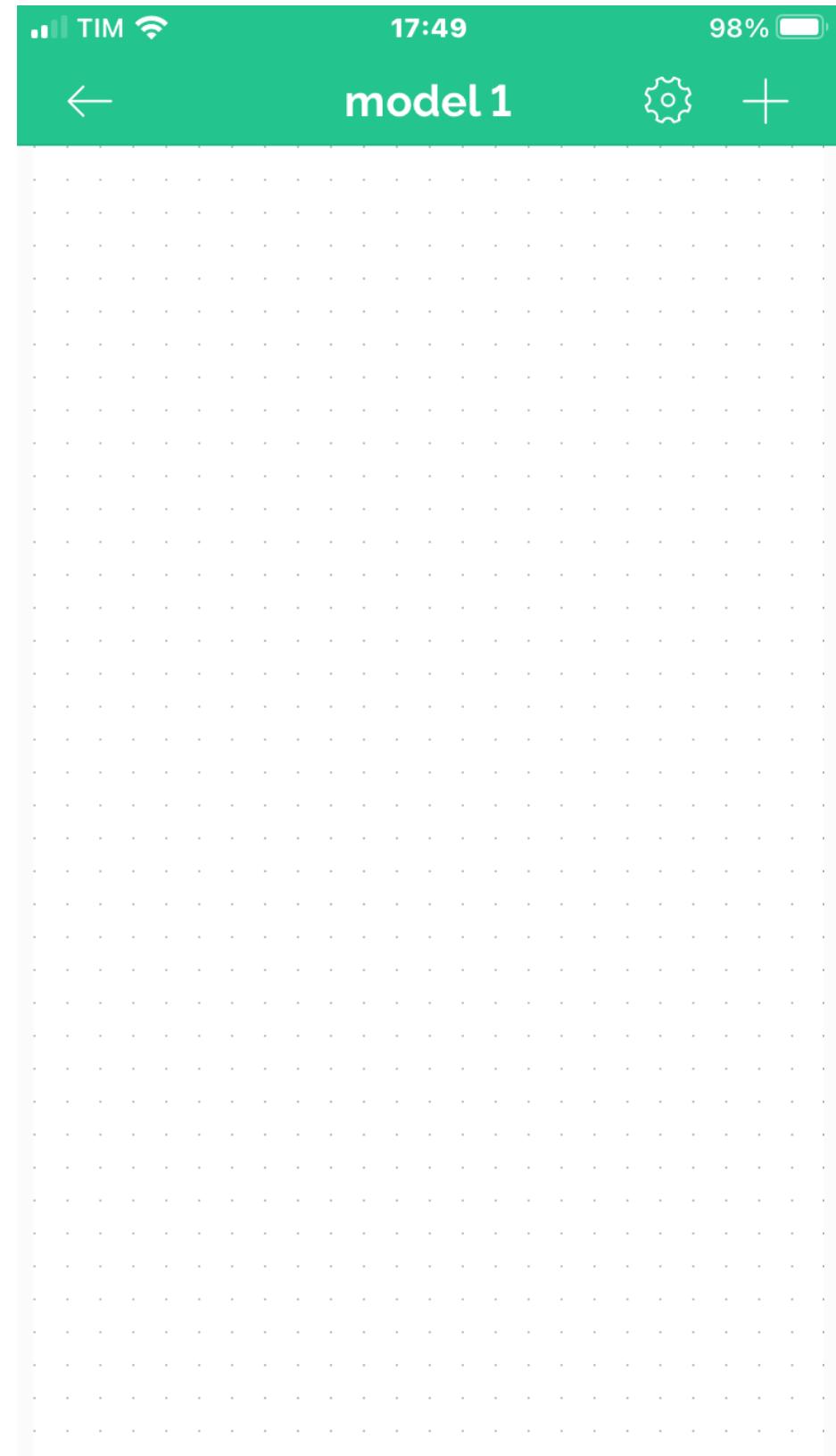
# Blynk – Versão Mobile

- Clique na ferramenta para abrir o modo de desenvolvimento
- Clique no seu modelo criado.



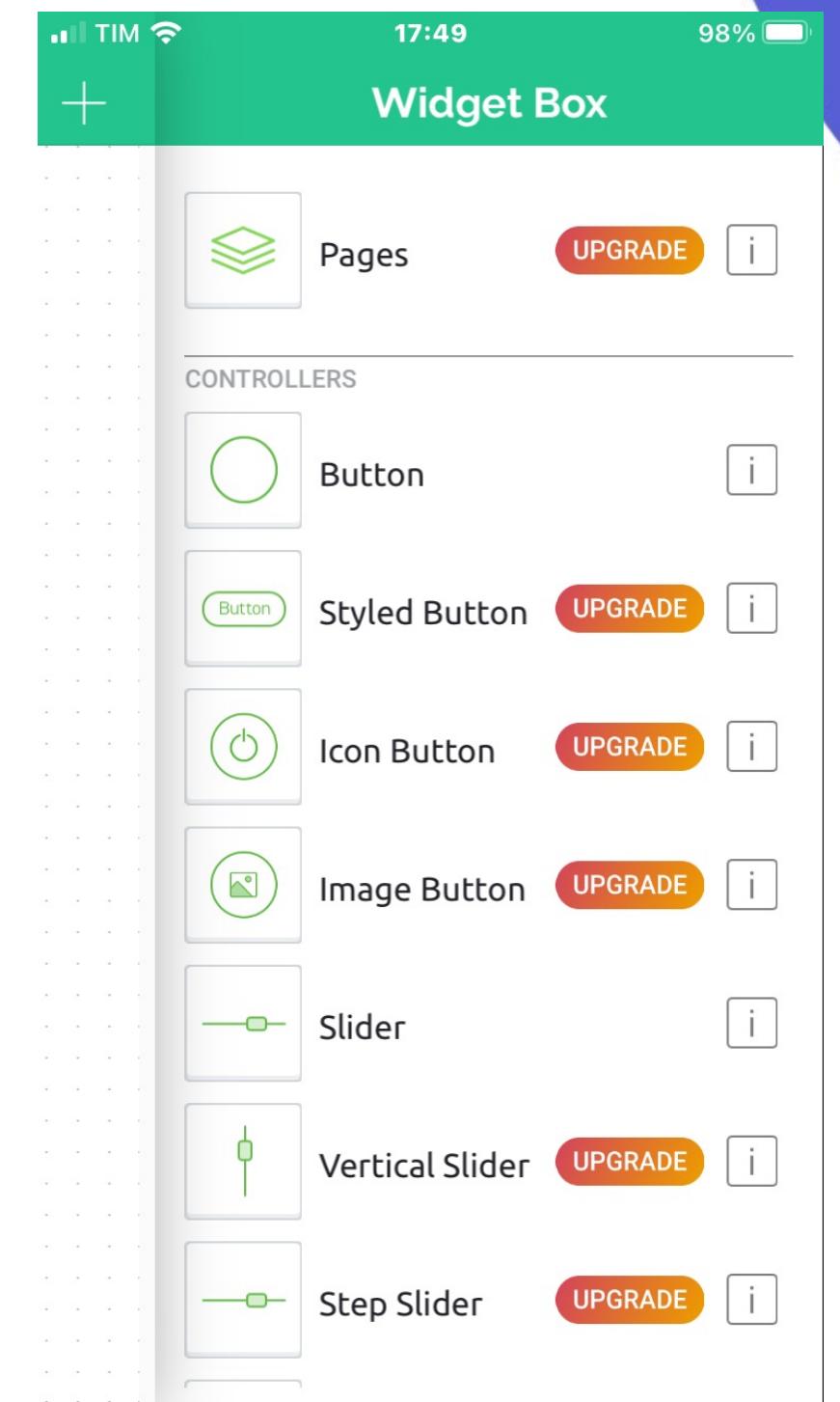
# Blynk – Versão Mobile

- Clique na ferramenta para abrir o modo de desenvolvimento
- Clique no seu modelo criado.
- Abre-se o dashboard móvel
- Clique no sinal de + para incluir widgets



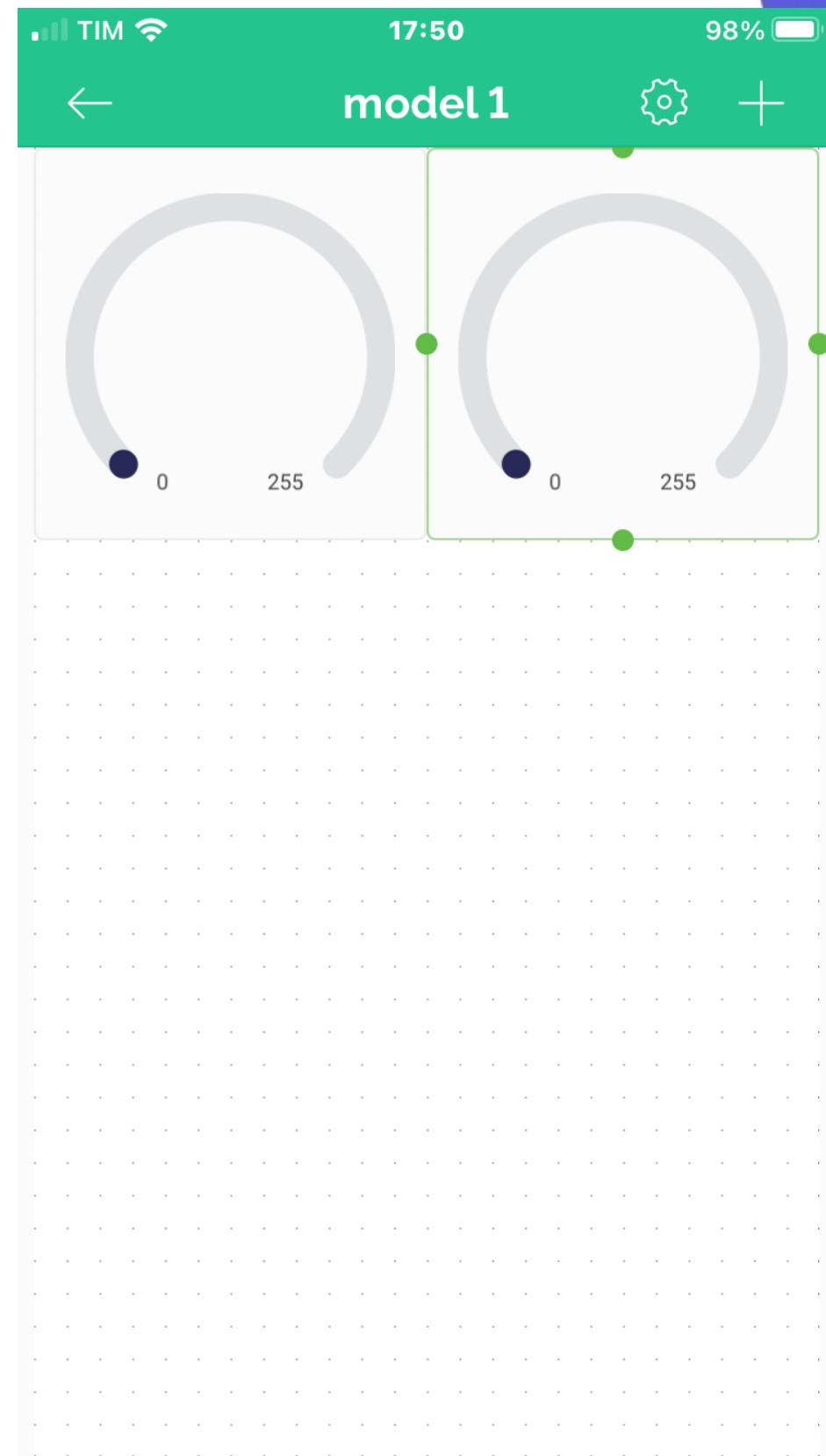
# Blynk – Versão Mobile

- Clique na ferramenta para abrir o modo de desenvolvimento
- Clique no seu modelo criado.
- Abre-se o dashboard móvel
- Clique no sinal de + para incluir widgets
- Selecione dois gauges



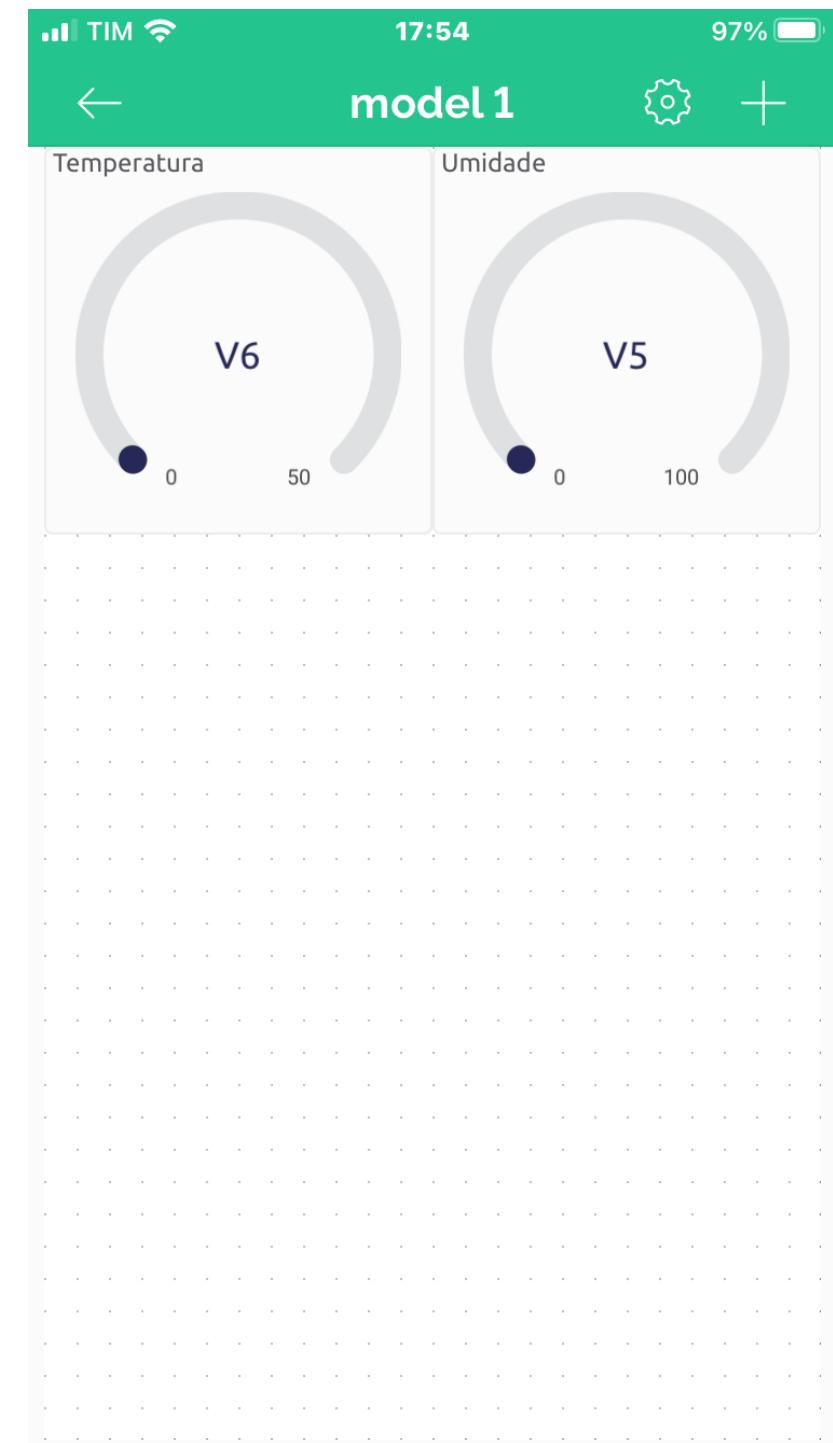
# Blynk – Versão Mobile

- Clique na ferramenta para abrir o modo de desenvolvimento
- Clique no seu modelo criado.
- Abre-se o dashboard móvel
- Clique no sinal de + para incluir widgets
- Selecione dois gauges

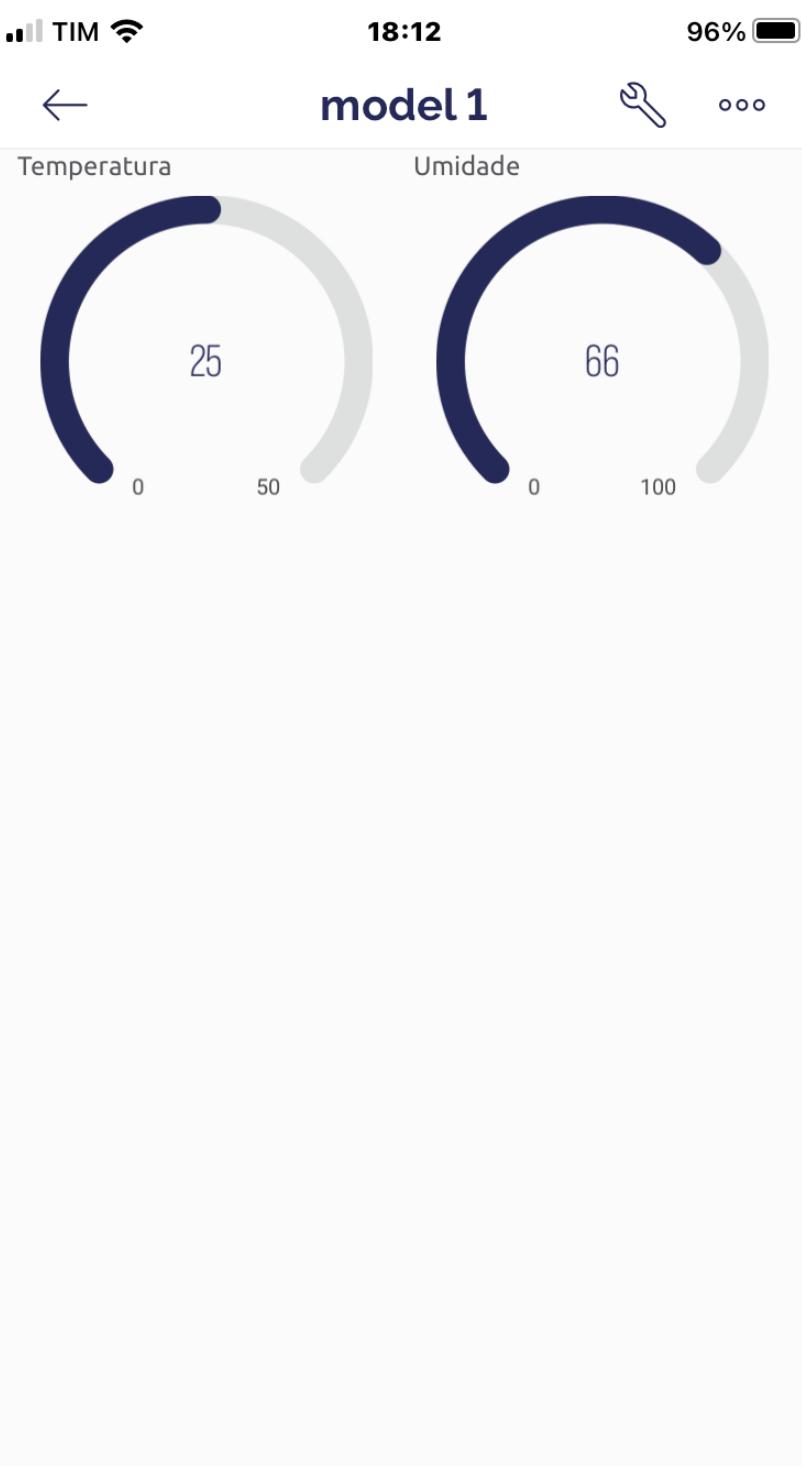


# Blynk – Versão Mobile

- Clique na ferramenta para abrir o modo de desenvolvimento
- Clique no seu modelo criado.
- Abre-se o dashboard móvel
- Clique no sinal de + para incluir widgets
- Selecione dois gauges
- Inclua as informações como no dashboard web



# Blynk – Versão Mobile



Inatel

CAMINHOS  
QUE CONECTAM  
COM O FUTURO

# Blynk – Led multicolor

- Abra a pasta: **led blynk**
  - Adicione o LED RGB à montagem.  
O circuito na *protoboard* ficará  
conforme a figura:

## D5: LED Vermelho (R)

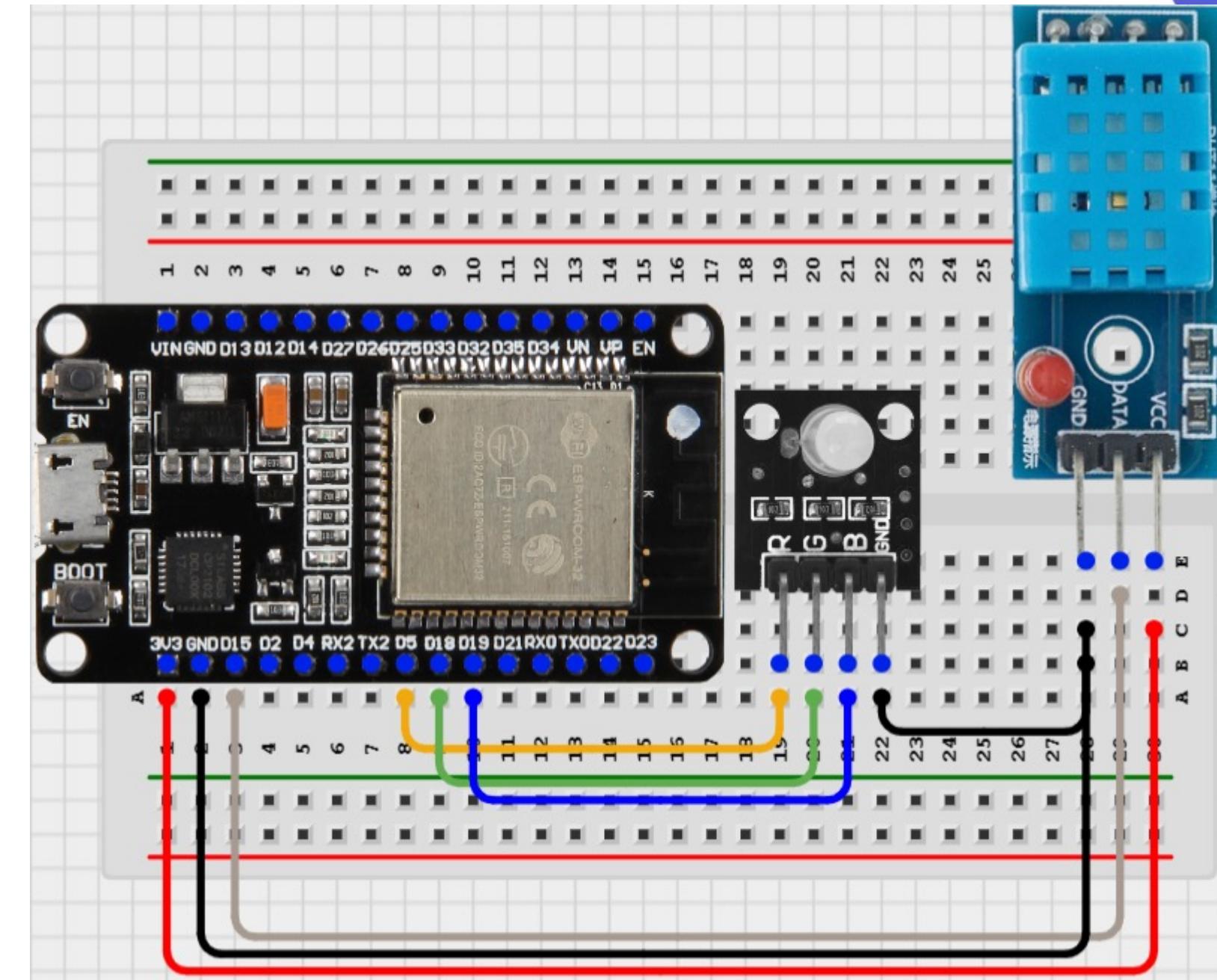
## D18: LED Verde (G)

## D19: LED Azul (B)

# D15: Pino DATA DHT11

# 3V3: Pinos VCC & 3V3 em curto

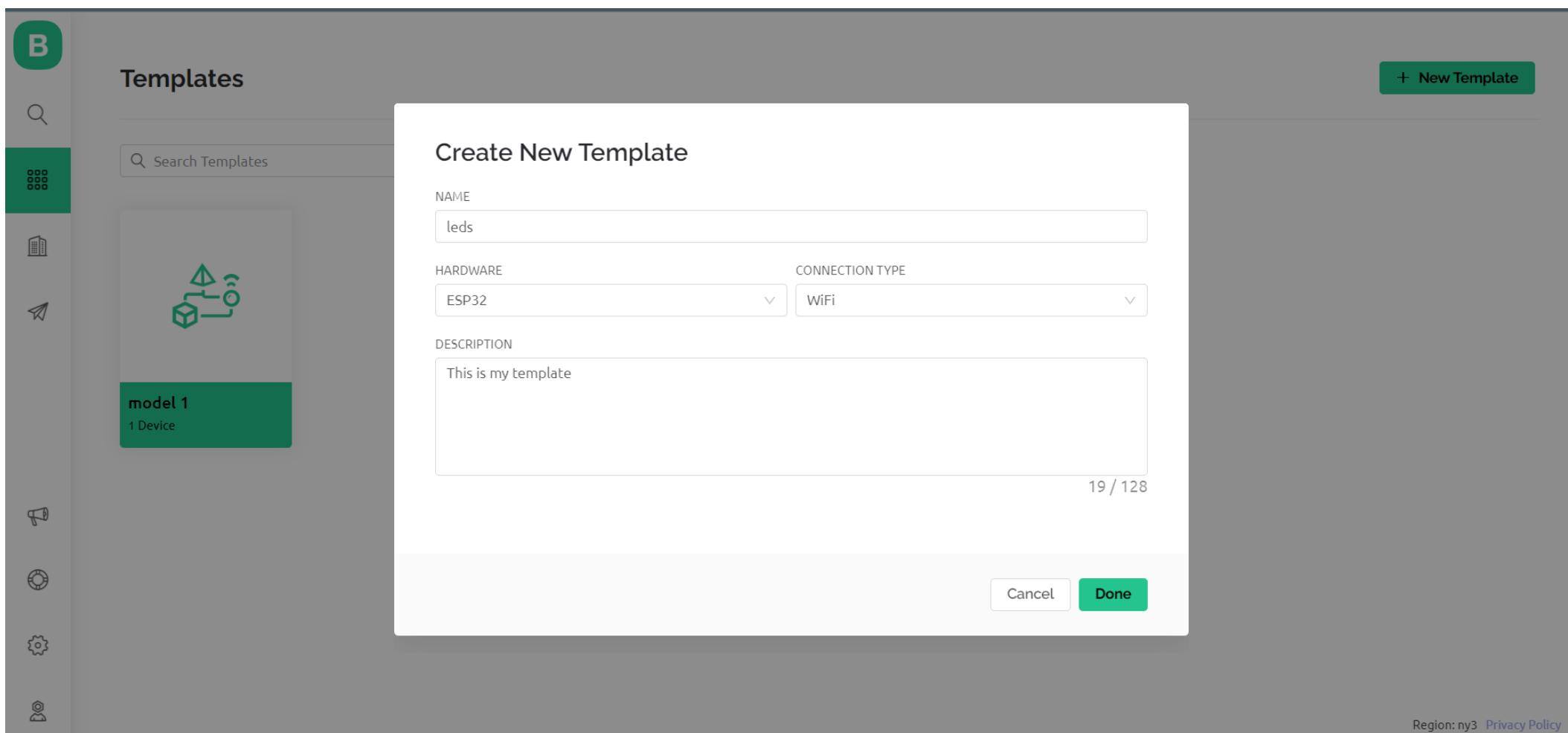
# GND: Pinos GND em curto



- O DHT11 não será usado neste exemplo

# Blynk – Led multicolor

- Criar um novo template



# Blynk – Led multicolor

- Criação de três canais, um para cada cor do led rgb

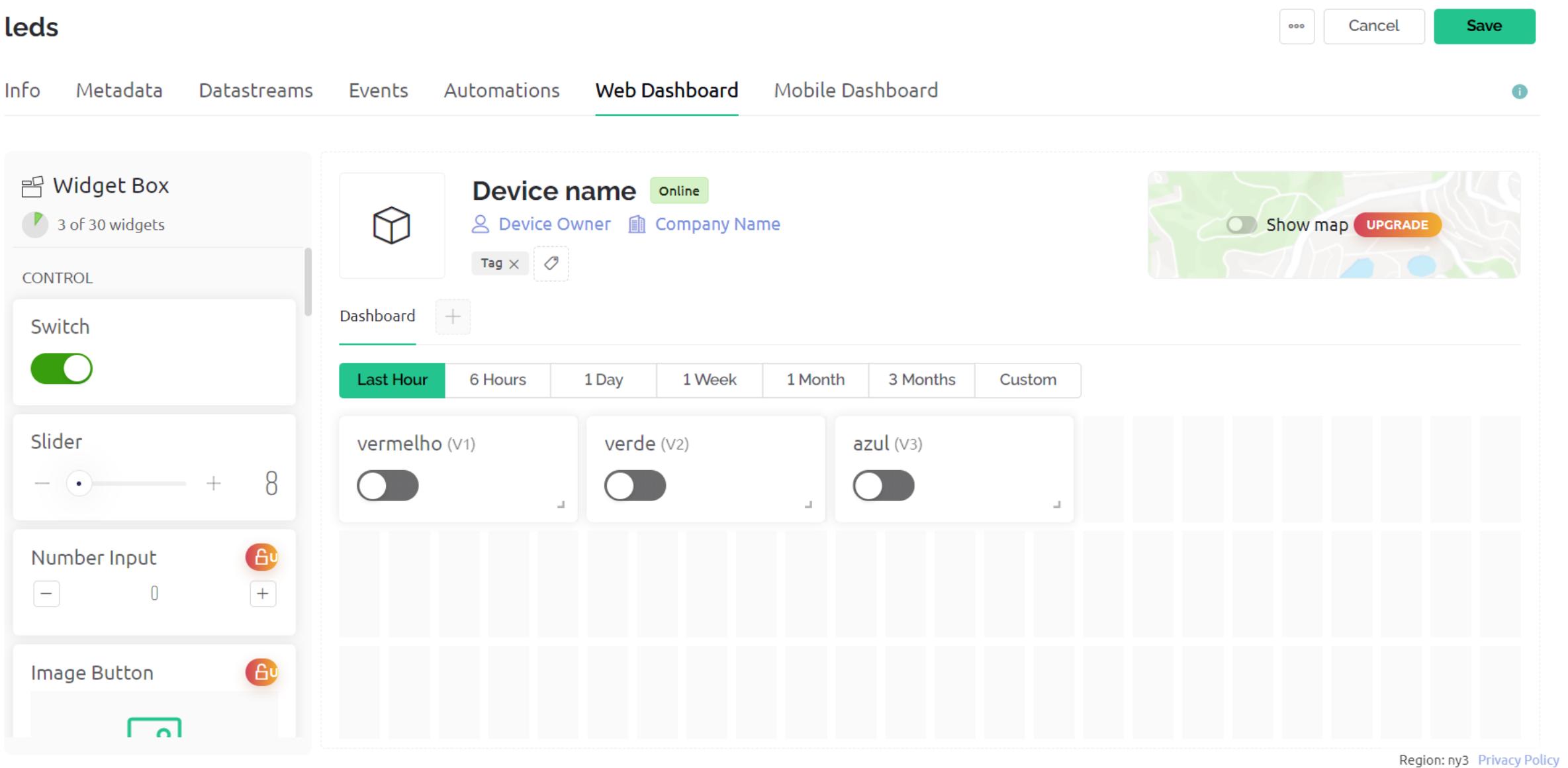
The screenshot shows the Blynk Datastreams interface for a project named "leds". The top navigation bar includes tabs for Info, Metadata, Datastreams (which is selected), Events, Automations, Web Dashboard, and Mobile Dashboard. There is also a "Save" button and a "Cancel" button. A search bar labeled "Search datastream" and a "New Datastream" button are visible. The main area displays a table titled "3 Datastreams" with the following data:

	ID	Name	Alias	Color	Pin	Data Type	Units	Is Raw	Min	Max	Actions
1	vermelho	vermelho		Red	V1	Integer		false	0	1	
2	verde	verde		Green	V2	Integer		false	0	1	
3	azul	azul		Blue	V3	Integer		false	0	1	

At the bottom right of the interface, there is a note: "Region: ny3 Privacy Policy".

# Blynk – Led multicolor

- Criação de três botões, um para cada cor do led rgb



# Blynk – Led multicolor

The screenshot shows the Blynk app interface. On the left, there's a sidebar with icons for Back, Search, Devices (2), Model 1, and Leds. The main area shows "My organization - 9448HL" with a search bar and a list of devices: "model 1" and "leds". The "leds" device is selected, showing its details. The "Device Info" tab is active, displaying the following information:

STATUS	LAST UPDATED
Offline	6:59 PM Today
DEVICE ACTIVATED	ORGANIZATION
6:59 PM Today by samuelbmafra@inatel.br	My organization - 9448HL
AUTHTOKEN	TEMPLATE NAME
LOhj - **** - **** - ****	leds
MANUFACTURER	
My organization 9448HL	
SSL	
No SSL	
BOARD TYPE	
ESP32	

Below the device info, there's a code block for FIRMWARE CONFIGURATION:

```
#define BLYNK_TEMPLATE_ID "TMPL21KbD8w7"  
#define BLYNK_DEVICE_NAME "leds"  
#define BLYNK_AUTH_TOKEN "L0hf7vgPsLQfaQW6UkZ2Trv0SaL_2Zb"
```

A note below the code says: "Template ID, Device Name, and AuthToken should be declared at the very top of the firmware code."

At the bottom right, it says "Region: ny3" and "Privacy Policy".

# Blynk – Led multicolor

- Abra a pasta: led blynk
- A função BLYNK\_WRITE é ativada sempre que o botão é acionado no dashboard.
- O valor (0 ou 1) é gravado em pinValue e o estado é acionado na respectiva cor.

```
src > main.cpp > ssid
1 #define BLYNK_TEMPLATE_ID "TMPL21KbD8W7"
2 #define BLYNK_DEVICE_NAME "leds"
3 #define BLYNK_PRINT Serial
4 #include <BlynkSimpleEsp32.h>
5 char auth[] = "Lohjf7vgPsLQfaQW6UkZ2Trv0SaL_2zb";
6 char ssid[] = "WLL-Inatel";
7 char pass[] = "inatelsemfio";
8 // LED RGB
9 #define LED_RED (5)
10 #define LED_GREEN (18)
11 #define LED_BLUE (19)
12 BLYNK_WRITE(V1)
13 {
14     int pinValue = param.asInt();           // assigning incoming value from pin V1 to a variable
15     if (pinValue == 1) {                   // If value is 1 run this command
16         digitalWrite(LED_RED, HIGH);      //D4 output from Wemos D1 mini
17     }
18     else {                                // If value is 0 run this command
19         digitalWrite(LED_RED, LOW);
20     }
21 }
```

# Blynk – Led multicolor

The screenshot shows the Blynk app interface for a device named "leds". The device is online, connected to "Samuel" in "My organization - 9448HL". There is an "Add Tag" button. The dashboard has tabs for Dashboard, Timeline, Device Info, Metadata, and Actions Log, with "Dashboard" selected. A timeline selector shows "Last Hour" is active. Below are three control cards: "vermelho" (red) with a red toggle switch, "verde" (green) with a green toggle switch, and "azul" (blue) with a blue toggle switch.

leds Online ...

Samuel My organization - 9448HL

Add Tag

Dashboard Timeline Device Info Metadata Actions Log

Latest Last Hour 6 Hours 1 Day 1 Week 1 Month 3 Months Custom

vermelho verde azul