

# Baixe o material da Palestra

## acesse o github

### <https://github.com/aterroso/SemanaMQTT>

# MQTT - Da teoria à prática, comunique-se no mundo IoT - com demonstração prática

*Prof. Anderson Terroso*

*Outubro/2025*

# Quem sou??

Portoalegrense, nascido em  
15/06/74.



- Formação Acadêmica

- Formado em Engenharia Elétrica/Eletrônica (1992-1996)
- Mestre em Engenharia Elétrica – Sistemas Tolerantes a Falha (1997-1999)

- Experiência Profissional

- Prof. Da PUCRS – março/2000 até o momento
- Coordenador da Eng. De Computação – 2008 a 2010
- Coordenador da Eng. Elétrica – 2010 a 2012
- Coordenador Acadêmica da Faculdade de Engenharia – 2012 a 2017
- Líder do Núcleo de Articulação Acadêmica da Escola Politécnica – 2018 até o momento.
- Mais de 200 TCC ´s orientados.

28 anos atuando em projetos de P&D e Gerência de Projetos

# Sumário

- MQTT – Significado e Conceito
- MQTT - Origem
- MQTT - Funcionamento
- MQTT - Estrutura
- MQTT - Conexão
- MQTT – Terminologia
- MQTT – Benefícios
- MQTT – Topologia
- Aplicação com Publish e Subscriber
- Exemplos de Broker
- Configuração do Broker
- Biblioteca do MQTT para ser usado com ESP32
- Exemplo: Publish e Subscriber (demonstração prática)

# MQTT – Conceito

**MQTT** = **M**essage **Q**ueuing **T**elemetry **T**ransport  
(*Transporte de Filas de Mensagem de Telemetria*)

- O MQTT é um protocolo de mensagens baseado em padrões, ou conjunto de regras, usado para comunicação de computador para computador. Sensores inteligentes, dispositivos acessórios e outros dispositivos da Internet das Coisas (IoT) normalmente precisam transmitir e receber dados por meio de uma rede com limitação de recursos e largura de banda limitada. Esses dispositivos IoT usam o MQTT para transmissão de dados, pois é fácil de implementar e pode comunicar dados IoT com eficiência. O MQTT oferece suporte a mensagens entre dispositivos para a nuvem e da nuvem para o dispositivo.

*Fonte: <https://aws.amazon.com/pt/what-is/mqtt/>*

# MQTT - Origem

- Surgiu em 1999 para ser usado na indústria de petróleo e gás;
- Em 2010, a IBM lançou o MQTT 3.1 como um protocolo gratuito e aberto.
- Em 2019, a OASIS (Organization for the Advancement of Structured Information Standards) lançou uma versão atualizada do MQTT 5.

# MQTT - Funcionamento

- O protocolo MQTT funciona conforme os princípios do modelo de publicação/assinatura.
- Numa rede tradicional, os clientes fazem requisições ao servidor, o servidor processa estas requisições e envia a resposta.
- O MQTT usa o padrão de publicação/assinatura para desacoplar o remetente da mensagem (publicador) do destinatário da mensagem (assinante).

# MQTT - Estrutura

- **Cliente MQTT:** um cliente MQTT é qualquer dispositivo de servidor para microcontrolador que executa uma biblioteca MQTT.
- **Se o cliente estiver enviando mensagens, atuará como publicador;**
- **Se o cliente estiver recebendo mensagens, atuará como destinatário.**
- Basicamente, qualquer dispositivo que se comunique por MQTT em uma rede pode ser chamado de dispositivo cliente MQTT.



# MQTT – Estrutura - continuação

- O agente (broker) MQTT é o sistema backend que coordena mensagens entre os diferentes clientes. Entre as responsabilidades do agente estão: receber e filtrar mensagens, identificar os clientes inscritos em cada mensagem e enviar as mensagens a eles. Também é responsável por outras tarefas, como:
- Autorizar e autenticar clientes MQTT
- Transmitir mensagens a outros sistemas para análise posterior
- Solucionar mensagens e sessões de clientes perdidas

# MQTT - Conexão

- Clientes e agentes (broker) começam a se comunicar usando uma conexão MQTT. Os clientes iniciam a conexão enviando uma mensagem *CONNECT* ao agente MQTT. O agente confirma que uma conexão foi estabelecida respondendo com uma mensagem *CONNACK*. Tanto o cliente MQTT como o agente requerem uma pilha TCP/IP para se comunicarem. Os clientes nunca se conectam entre si, apenas com o agente.

# MQTT - Terminologia

- **Tópico do MQTT**

- O termo “tópico” refere-se a palavras-chave que o agente (broker) MQTT usa para filtrar mensagens para os clientes MQTT. Os tópicos são organizados de maneira hierárquica, semelhante a um diretório de arquivos ou pastas. Por exemplo, imagine um sistema de casa inteligente que está em funcionamento em uma casa de vários andares que tem diferentes dispositivos inteligentes em cada andar. Nesse caso, o agente MQTT pode organizar tópicos como:
  - *ourhome/groundfloor/livingroom/light*
  - *ourhome/firstfloor/kitchen/temperature*

# MQTT - Terminologia

- **Publicação de MQTT (PUBLISH)**

- Os clientes MQTT publicam mensagens contendo o tópico e os dados em formato de bytes. O cliente determina o formato de dados, como dados de texto, dados binários, arquivos XML ou JSON. Por exemplo, uma lâmpada no sistema de casa inteligente pode publicar uma mensagem *on* no tópico *livingroom/light*.

# MQTT - Terminologia

- **Assinatura de MQTT**

- Os clientes MQTT enviam uma mensagem *SUBSCRIBE* (ASSINAR) ao agente MQTT para receber mensagens sobre tópicos de interesse. Esta mensagem contém um identificador exclusivo e uma lista de assinaturas. Por exemplo, o aplicativo de casa inteligente de seu telefone deseja exibir quantas luzes estão acesas em sua casa. Ele assinará o tópico *light* e aumentará o contador para todas as mensagens *on*.

# MQTT - Benefícios

- Leve e eficiente: a quantidade de bytes usados é muito pequeno.
- Escalável: a quantidade de código para usar o MQTT é mínima, podendo conectar milhões de dispositivos.
- Confiável: tempo de conexão extremamente baixo e apresentam 3 níveis diferentes de serviço de qualidade para garantir da confiabilidade.
- Seguro: as mensagens podem ser criptografadas usando protocolos de autenticação modernos.
- Suporte: hoje em dia diversas linguagens dão suporte ao MQTT.

# MQTT - topologia

## MQTT Clients

Ex: Sensors



Publish: "30°"  
Topic: "temp"



Publish: "65%"  
Topic: "moisture"



Publish: "1.8g"  
Topic: "accel-x"

MQTT  
Broker

## MQTT Clients



Ex: Valve



Publish: "30°"

Subscribed to "temp"

Publish: "closed"  
Topic: "valve"

Ex: Cloud Server



Subscribed to "accel-x"

Publish: "1.8g"

# Aplicação com “publisher” e “subscriber”

**CLIENTE**

**PUBLISHER  
(Publicador)**

```
MQTT.publish("lab318/temp", xxxxxx);  
MQTT.publish("lab318/umid", yyyyyy);
```



**CLIENTE**

**SUBSCRIBER  
(Assinante)**

```
MQTT.subscribe("lab318/temp");  
MQTT.subscribe("lab318/umid");
```



**BROKER**

**ID = TERROSO\_PUB**

**Tópicos:**

lab318/temp  
lab318/umid




**ID = TERROSO\_SUB**

**Tópicos:**

lab318/temp  
lab318/umid



# Exemplo de Brokers....

BROKER	Site	Host	Porta
	<a href="https://www.hivemq.com/public-mqtt-broker/">https://www.hivemq.com/public-mqtt-broker/</a>	broker.hivemq.com	TCP Port: 1883
	<a href="http://www.mqtt-dashboard.com/">http://www.mqtt-dashboard.com/</a>	broker.mqttdashboard.com	TCP Port: 1883
	<a href="https://test.mosquitto.org/">https://test.mosquitto.org/</a>	test.mosquitto.org	TCP Port: 1883
	<a href="https://www.emqx.io/mqtt/public-mqtt5-broker">https://www.emqx.io/mqtt/public-mqtt5-broker</a>	broker.emqx.io	TCP Port: 1883

# Cada cliente deve criar o seu nome Client\_ID. Boas práticas na criação do Client ID:

- Se você tem **poucos dispositivos fixos**, pode dar nomes fixos e descritivos.
  - Exemplo: ESP32Frigorifico
- Se vai ter **vários dispositivos iguais** (ex: muitos ESP32), é melhor gerar IDs com um sufixo único + (número aleatório, MAC do WiFi, ou timestamp).
  - Exemplo: ESP32Frigorifico\_1234

## 1) Usar sem criar conta (instalação local ou broker público)

- O EMQX é open source.
- Você pode baixar e rodar no seu próprio PC, servidor, Raspberry Pi, etc.
  - Download: <https://www.emqx.io/downloads>
- Também existem **brokers públicos gratuitos** (como broker.emqx.io), que você já pode usar direto sem conta, apenas informando:
  - Host: broker.emqx.io
  - Porta: 1883 (sem TLS) ou 8883 (com TLS)
  - Sem usuário/senha (modo aberto)

Nesse caso, **não precisa conta**, mas:

- É um broker **compartilhado** (qualquer pessoa pode publicar/assinar tópicos).
- **Não tem garantia** de disponibilidade (pode cair, pode ser limpo sem aviso).
- **Não dá para configurar regras/requisições HTTP/dashboards**, é só MQTT básico.

## 2) Criar conta no EMQX Cloud (emqx.io)

- Aí você ganha um **broker privado na nuvem**, só seu.
- Tem o plano **Free** (1 milhão de minutos de sessão, 1 GB tráfego, etc).
- Você pode configurar **usuário/senha, TLS, regras (webhooks, banco de dados, dashboards, etc.)**.
- É indicado se você quer **segurança, controle e integração com outros sistemas**.

# Conta *free* do emqx.io permite o seguinte:

1 million session minutes per month

1 GB of data traffic per month

1 million rule actions per month

## 1 million session minutes per month

- **O que é:** Cada vez que um cliente (ESP32, Node.js, app, etc.) se conecta ao broker MQTT, abre-se uma sessão. O tempo de conexão dessa sessão é contado em minutos.

- **Exemplo prático:**

- Se você tiver **1 dispositivo conectado 24h por dia durante 30 dias**, ele consumirá:  $30 \text{ dias} \times 24\text{h} \times 60 \text{ min} = 43.200$  minutos
  - Bem abaixo de 1 milhão.
- Se tiver **10 dispositivos conectados continuamente**, seriam 432.000 minutos.
- Se tiver **25 dispositivos online direto**, seriam 1.080.000 minutos, o que já ultrapassa o limite do plano gratuito.

## 1 GB of data traffic per month

- **O que é:** É o limite de tráfego de dados (upload + download) que pode passar pelo broker MQTT no mês. Isso inclui todas as mensagens **publish** e **subscribe**.

- **Exemplo prático:**

- Se cada mensagem publicada for de **100 bytes** (bem pequena, como temperatura/umidade),
  - $1 \text{ GB} = 1.000.000.000 \text{ bytes} / 100 \text{ bytes} \approx 10 \text{ milhões de mensagens.}$
- Mas se forem mensagens de **1 KB cada**,
  - $1 \text{ GB} / 1 \text{ KB} = 1 \text{ milhão de mensagens}$

## 1 million rule actions per month

- **O que é:** O EMQX Cloud tem um **Rule Engine** que permite criar regras (por exemplo: “quando receber uma mensagem no tópico X, salva em banco de dados Y” ou “manda uma requisição HTTP para API Z”).

- Cada vez que uma regra é disparada, isso conta como **1 action**.

- **Exemplo prático:**

- Se você cria uma regra para salvar cada leitura de sensor em um banco de dados, cada mensagem recebida conta como **1 ação**.

- Se tiver 10 sensores publicando 1 vez por minuto  $\rightarrow 10 \times 60 \times 24 \times 30 = 432.000$  actions por mês.

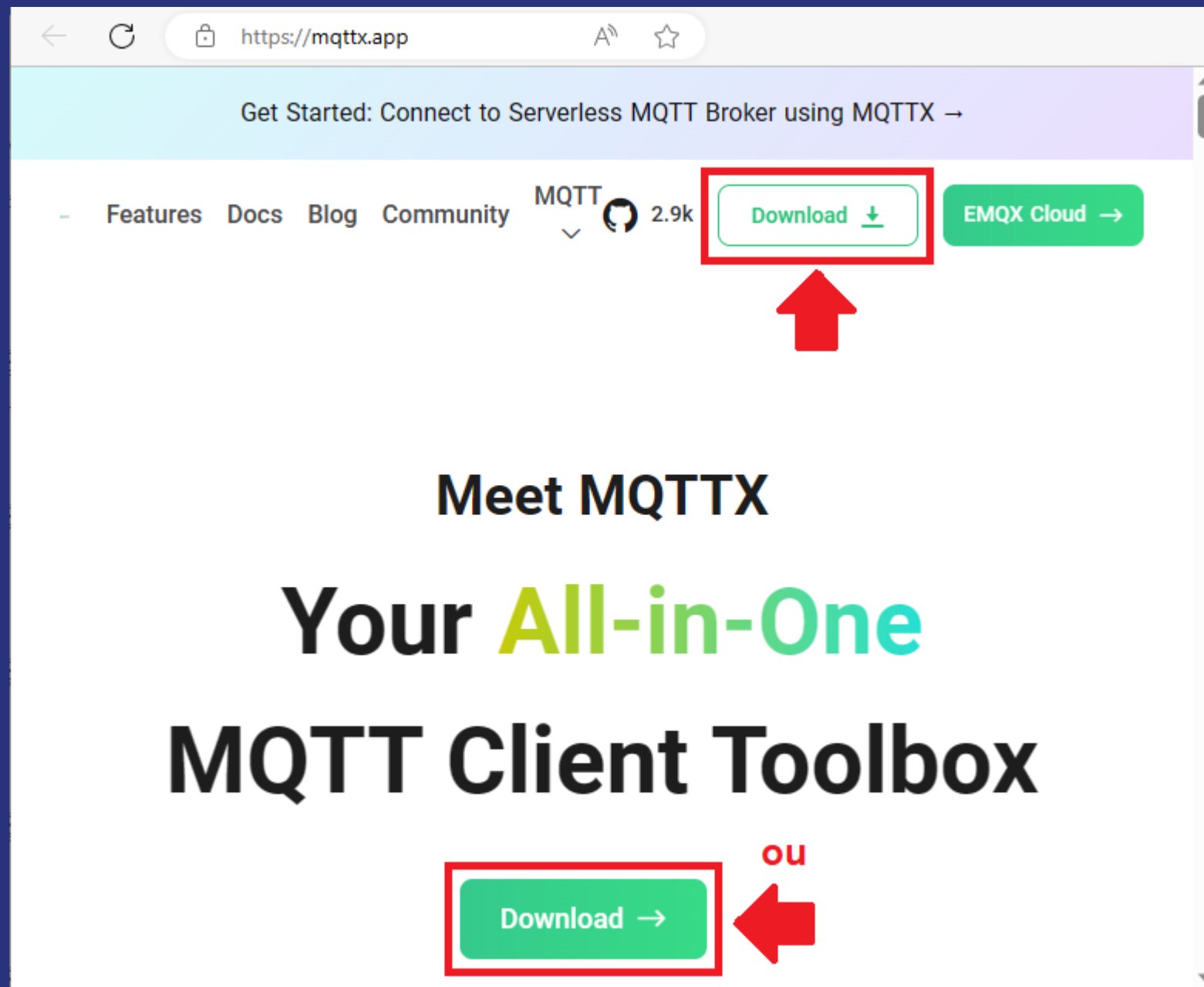
$\rightarrow$  Caberia dentro do limite de 1 milhão.

- Mas se tiver 100 sensores  $\rightarrow$  seriam **4,3 milhões actions**, ultrapassando o limite free.



Baixe o MQTTX  
que é uma  
ferramenta útil  
para monitorar as  
conexões e as  
publicações e  
assinaturas. O  
MQTTX é um  
cliente.

<https://mqttx.app>



**O MQTTX não é um broker, o broker é o emqx.io.**

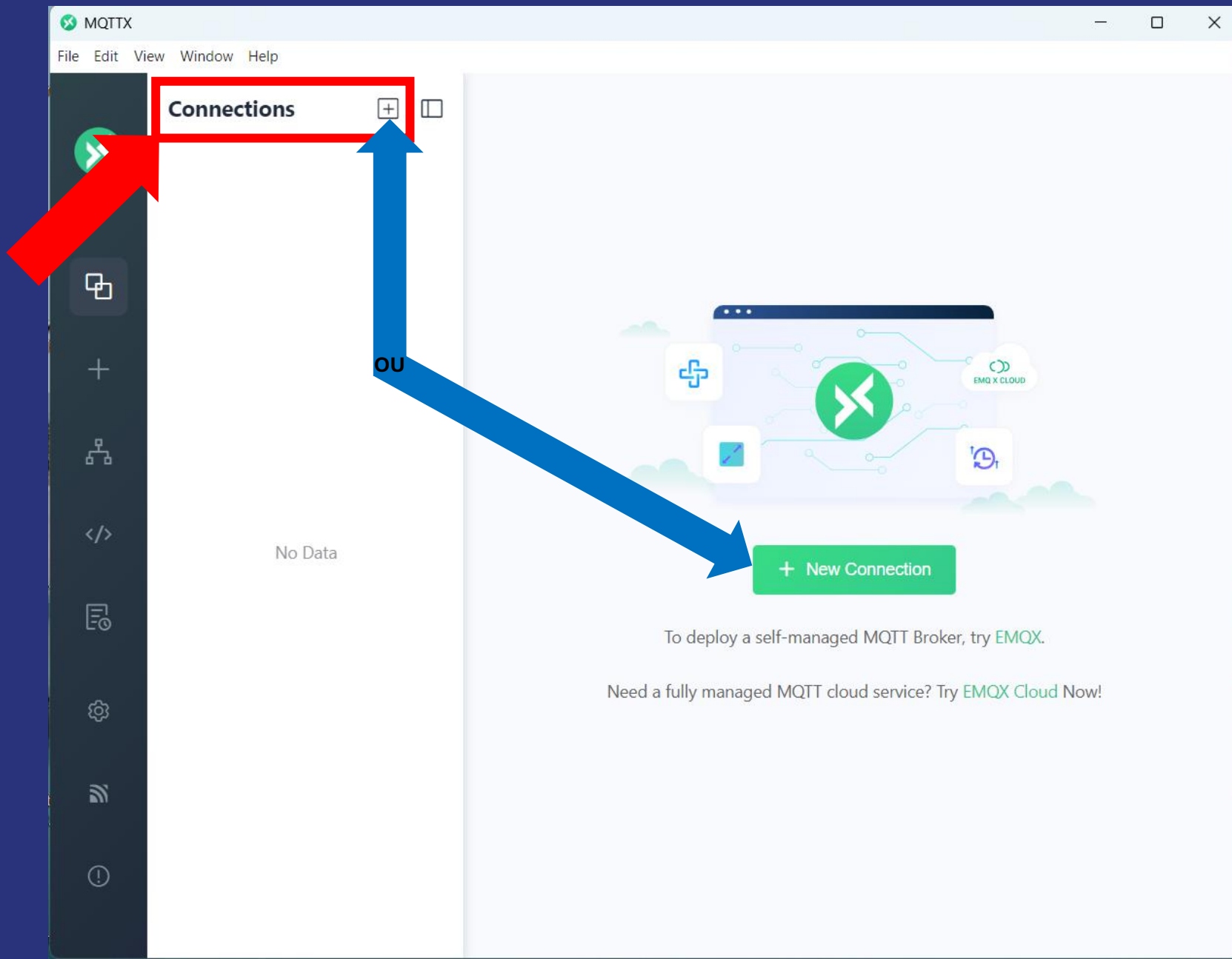
**O MQTTX é um cliente que fala com o broker emqx.io**

# Como criar uma nova conexão?

No MQTTX, o **Connections** é apenas a **lista de conexões** que você configurou para falar com brokers.

Portanto:

- Cada “Connection” representa **um broker** que você salvou para facilitar o acesso.
- Dentro dessa conexão, você pode publicar/assinar tópicos.



The screenshot shows the MQTTX application window. The title bar says 'MQTTX'. The menu bar includes 'File', 'Edit', 'View', 'Window', and 'Help'. The main interface has a 'Connections' tab on the left and a 'New' configuration panel on the right. The 'New' panel has a 'Connect' button highlighted with a red box. A dark blue callout bubble points to this button with the text: 'Por fim, clique em “Connect”'. The 'General' section of the 'New' panel contains several fields: 'Name' (with a callout: 'Nome desta conexão com o Broker. Não é usado em lugar algum.'), 'Host' (set to 'mqtt://' and 'broker.emqx.io', with a callout: 'Este é o endereço do broker e a porta usada.'), 'Port' (set to '1883'), 'Client ID' (set to 'mqttx\_fc46a0a8', with a callout: 'Como foi dito, o MQTTX não é um broker e sim um client conectado ao broker emqx.io. Portanto, ele deve ter um Client ID.'), 'Username' and 'Password' (grouped by a green bracket with a callout: 'O broker emqx.io é público, portanto aceita conexões sem usuário e senha. Se desejar segurança de dados, utilize um broker privado.'), and an 'SSL/TLS' toggle switch.

MQTTX

File Edit View Window Help

Connections + □ < Back New

Connect ✓

General

\* Name

\* Host mqtt:// broker.emqx.io

\* Port 1883

Client ID mqttx\_fc46a0a8

Username

Password

SSL/TLS

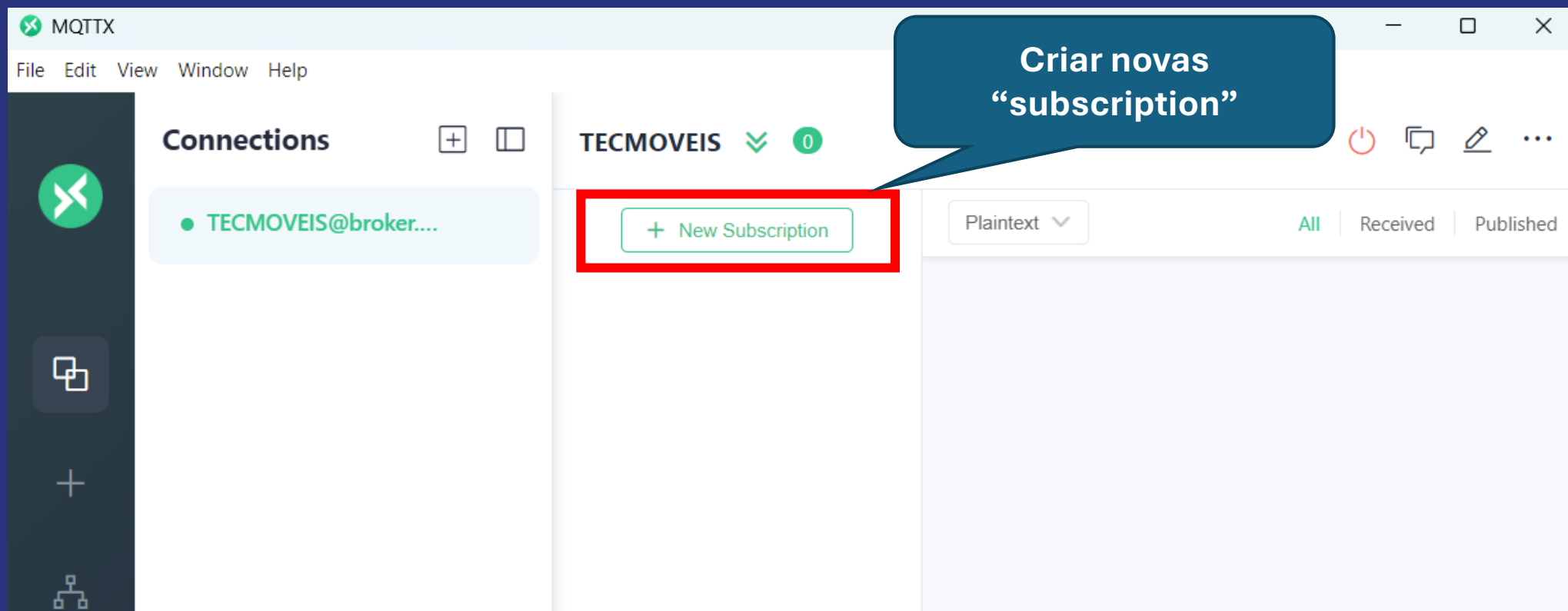
Nome desta conexão com o Broker.  
Não é usado em lugar algum.

Este é o endereço do broker e a porta  
usada.

Como foi dito, o MQTTX não é um  
broker e sim um client conectado ao  
broker emqx.io. Portanto, ele deve ter  
um Client ID.

O broker emqx.io é público, portanto  
aceita conexões sem usuário e senha.  
Se desejar segurança de dados, utilize  
um broker privado.

# Crie as “Subscription” – cuidado para criar o tópico igual ao usado no código do microcontrolador, python, appinventor, etc..



❗ New Subscription

❗

Topic

testtopic/#

❗

QoS

0 At most once

Color

#14CC0A

❗

Alias

Subscription Identifier

No Local Flag

☐ true ☒ false

Retain as Published Flag

☐ true ☒ false

Retain Handling

0

Cancel

Confirm

## Topic:

- O tópico em que você vai se inscrever.
- Pode ser específico:
  - Exemplo: lab318/temp (só mensagens desse tópico).
- Ou com **wildcards** (coringas):
  - lab318/# → pega todos os sub-tópicos de lab318/.
  - lab318/+/status → pega mensagens de qualquer cliente que publique algo como lab318/ESP32/status.

## QoS (Quality of Service):

Define o nível de garantia da entrega da mensagem:

- **0 (At most once)** → "no máximo uma vez" → rápido, mas sem confirmação (pode perder mensagens).
- **1 (At least once)** → "pelo menos uma vez" → garante entrega, mas pode duplicar.
- **2 (Exactly once)** → "exatamente uma vez" → mais confiável, mas mais lento.

New Subscription

\* Topic

testtopic/#

\* QoS

0

At most once

Color

#14CC0A

Alias

Subscription Identifier

No Local Flag

☐ true

☒ false

Retain as Published Flag

☐ true

☒ false

Retain Handling

0

Cancel

Confirm

## No Local Flag:

- **false** (padrão): se você publicar no mesmo tópico, também vai receber a mensagem.
- **true**: você não recebe mensagens que você mesmo publicou.

## Retain as Published Flag:

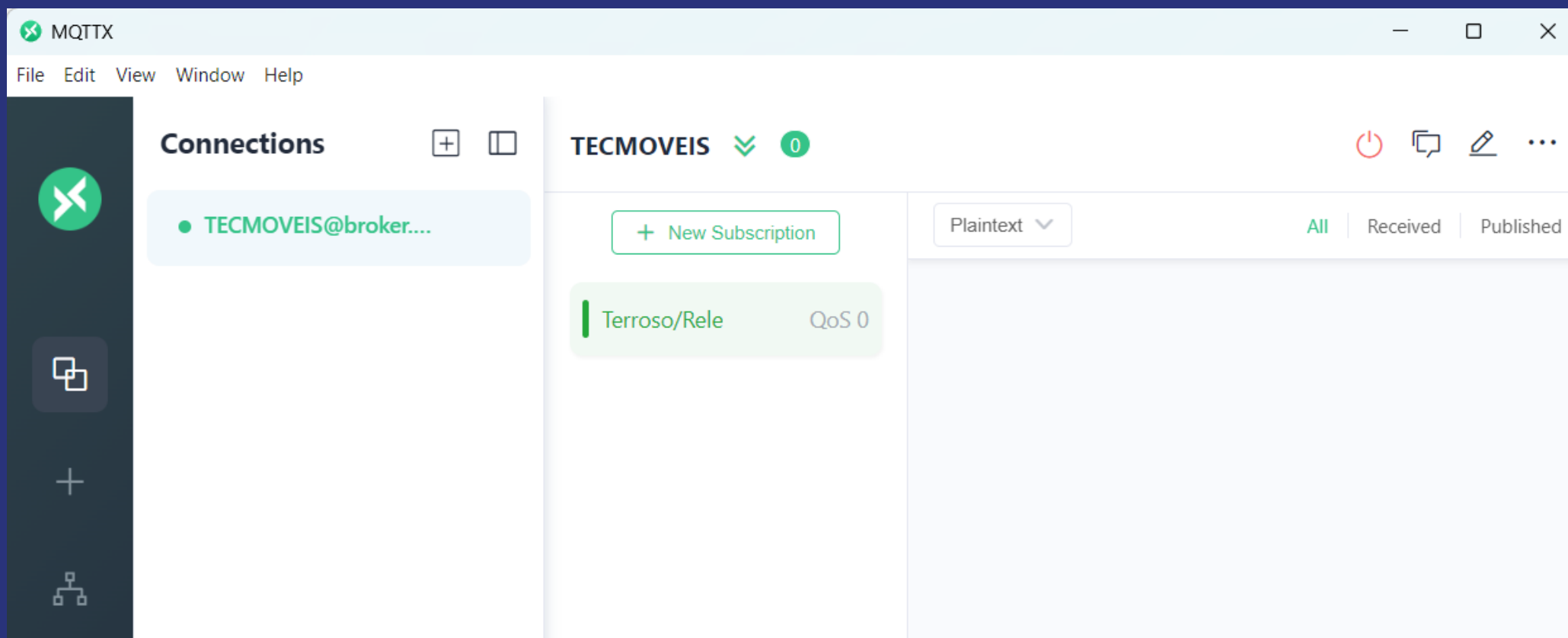
- Se o publisher marcou a mensagem como **retain**, o broker guarda a última mensagem daquele tópico.
- Essa flag define se você quer respeitar esse "retain" ou não.

## Retain Handling:

Controla **como o broker envia mensagens retidas**:

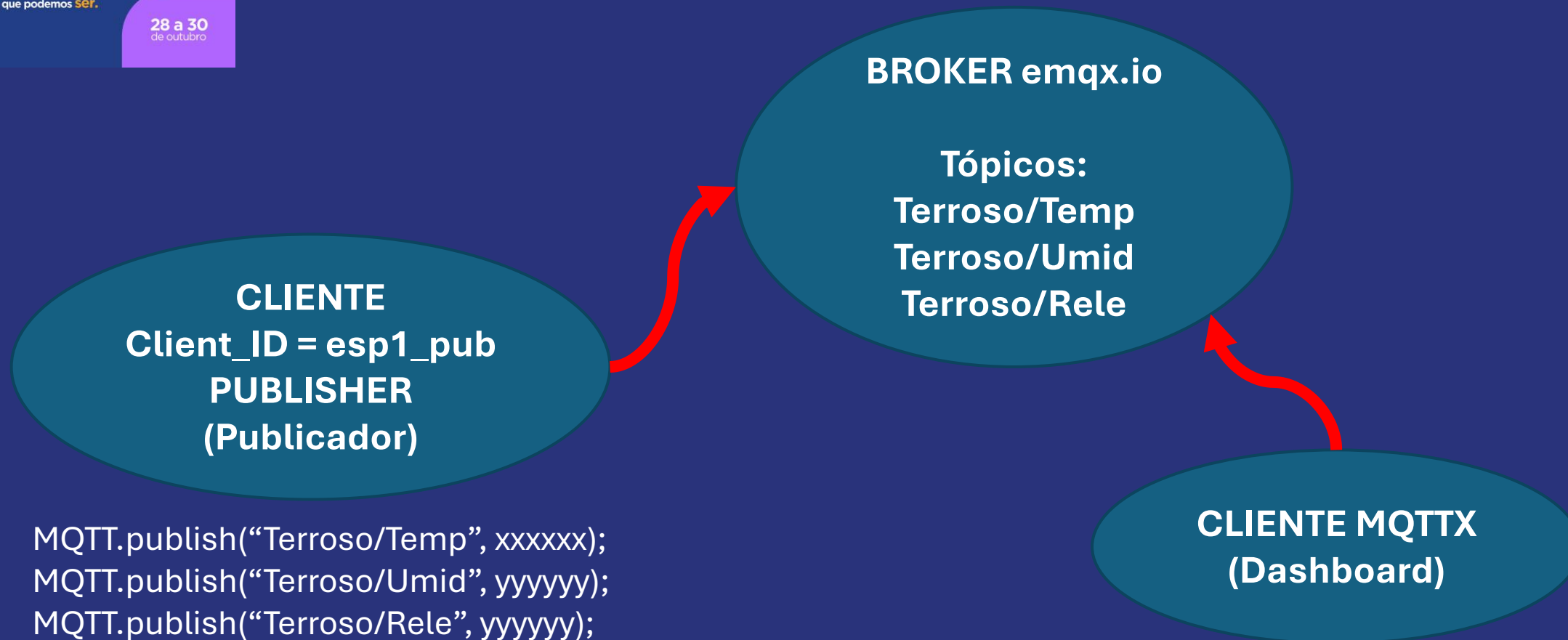
- **0** → envia a mensagem retida sempre que você se inscreve.
- **1** → envia só se você se inscrever pela primeira vez.
- **2** → nunca envia a mensagem retida.

Esse cliente MQTTX agora está inscrito no tópico Terroso/Rele. Caso seja escrito alguma coisa no tópico Terroso/Rele, essa mensagem será roteada pelo broker emqx.io também para o Client MQTTX.





# ESP32 somente como Publish



```
#include <WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"
```

```
#define DHTPIN 4
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
```

```
static long long tempo=0;
```

```
const char* ssid = "mototerroso";
const char* password = "0123456789";
```

```
const char* mqtt_broker = "broker.emqx.io";
const int mqtt_port = 1883;
```

```
#define MQTT_ID "esp1_pub"
#define topico_pub_temp "Terroso/Temp"
#define topico_pub_umid "Terroso/Umid"
#define tópico_pub_rele "Terroso/Rele"
```

```
WiFiClient espClient;
PubSubClient MQTT(espClient);
```

# ESP32: Cliente Publisher (Publicador)

Libs do wifi, MQTT e DHT  
(sensor temp/umid)

Pino físico do ESP (4) ⇔ DHT  
Tipo de DHT => DHT22  
Criando o objeto dht da classe DHT

Parâmetros da rede wifi  
(configure conforme a sua rede)

Constantes para armazenar  
o endereço do broker e a porta.

Definições do MQTT\_ID e os tópicos

Cria uma instância do WiFiClient => espClient  
Cria um cliente MQTT conectado do espClient

```
void conectaWifi();  
void conectaBroker();  
void publicaDados();
```

Declara os protótipos das funções que serão usadas.

```
void setup()  
{  
  Serial.begin(9600);  
  dht.begin();  
  conectaWifi();  
  MQTT.setServer(mqtt_broker, mqtt_port);  
  conectaBroker();  
}
```

#### Função SETUP:

- Configura a comunicação serial a um baudrate de 9600bps.
- Inicializa o dht.
- Chama a função conectaWifi
- Conecta-se ao broker mqtt.

```
void conectaWifi()  
{  
  Serial.println("Conectando a rede wifi!");  
  WiFi.begin(ssid, password);  
  while (WiFi.status() != WL_CONNECTED)  
  {  
    delay(500);  
    Serial.println("Conectando a rede wifi....");  
  }  
  Serial.println("Conectado a rede wifi com sucesso!!!");  
}
```

A função conectaWifi é responsável por fazer a conexão com a rede wifi. Neste caso não é feito um “timeout”...ele fica tentando.

```
void conectaBroker()
{
    while(!MQTT.connected())
    {
        if (MQTT.connect(MQTT_ID))
        {
            Serial.println("Conectado ao Broker!");
        }
        else
        {
            Serial.print("Falha na conexão. O status é: ");
            Serial.print(MQTT.state());
            delay(2000);
        }
    }
}
```

A função `conectaBroker` é responsável por fazer a conexão com o broker MQTT. Neste caso não é feito um “timeout”...ele fica tentando e reportando os erros de conexão, caso não consiga se conectar

```
void publicaDados()
{
    float umid = dht.readHumidity();
    float temp = dht.readTemperature();
    MQTT.publish(topico_pub_temp, String(temp).c_str());
    MQTT.publish(topico_pub_umid, String(umid).c_str());
    if(temp<15) MQTT.publish(topico_pub_rele,"rele_on");
    else MQTT.publish(topico_pub_rele,"rele_off");
}
```

A função `publicaDados` é responsável por ler os valores de umidade e temperatura do DHT e publicar os dados no broker MQTT. O valor float precisa ser transformado em string e depois `.c_str()` → converte para um array de caracteres.

```
void loop()
{
    static long long pooling = 0;
    if(WiFi.status() != WL_CONNECTED) conectaWifi();

    if(!MQTT.connected()) conectaBroker();

    if(millis() > pooling + 10000)
    {
        pooling = millis();
        publicaDados();
    }
    MQTT.loop();
}
```

A função loop é responsável por:

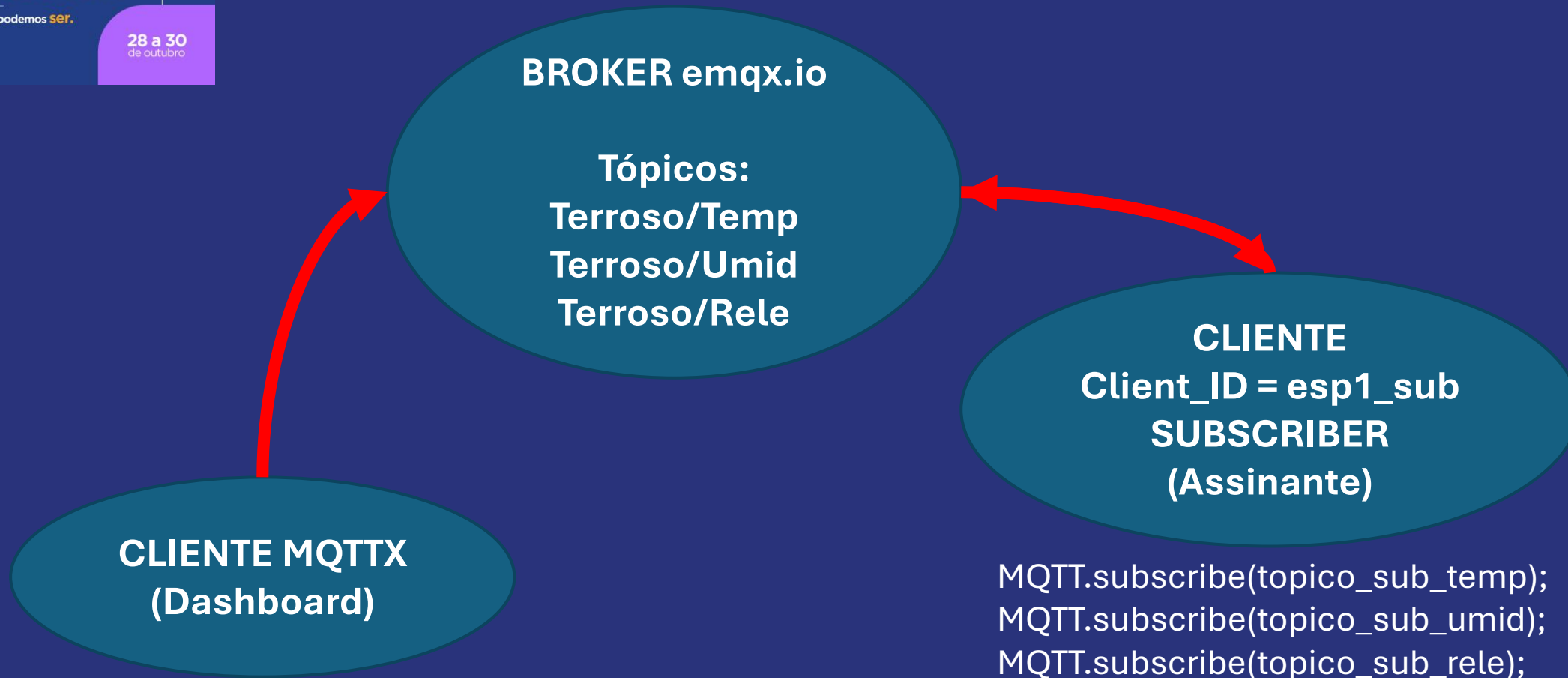
- Verificar se o wifi está conectado, senão conecte-se.
- Verificar se está conectado ao broker, senão conecte-se.
- Faz uma publicação de dados a cada 10 segundos.

**O que faz o MQTT.loop() ?????**

**Verifica se o websocket (conexão) ainda está aberto**  
**Lê dados vindos do broker**  
**Envia keep-alive (PINGREQ) – ping de 15 em 15 seg.**

Obs.: se o cliente fosse apenas PUBLISH, não haveria a necessidade de ter a função MQTT.loop(), entretanto a cada publicação, teria que verificar novamente a conexão e provavelmente teria que fazer uma nova conexão devido ao tempo de inatividade.

# ESP32 somente como Subscriber



**OBS.:** CADA CLIENTE TEM QUE TER UM CLIENT\_ID DIFERENTE, SENÃO O BROKER DERRUBA A CONEXÃO. NO CLIENTE PUBLISH, USANDO esp1\_pub.



# Código Fonte ESP32: Cliente Subscriber (Assinante)

```
#include <WiFi.h>  
#include <PubSubClient.h>
```

Libs do wifi e MQTT

```
const char* ssid = "mototerroso";  
const char* password = "0123456789";
```

Parâmetros da rede wifi  
(configure conforme a sua  
rede)

```
const char* mqtt_broker = "broker.emqx.io";  
const int mqtt_port = 1883;
```

Constantes para armazenar  
o endereço do broker e a  
porta.

```
#define MQTT_ID "esp1_sub"  
#define topico_sub_temp "Terroso/Temp"  
#define topico_sub_umid "Terroso/Umid"  
#define topico_sub_rele "Terroso/Rele"
```

Definições do MQTT\_ID e os  
tópicos

```
WiFiClient espClient;  
PubSubClient MQTT(espClient);
```

Cria uma instância do  
WiFiClient => espClient  
Cria um cliente MQTT  
conectado do espClient

```
void conectaWifi();  
void conectaBroker();  
void callback(char *topico, byte *msg, unsigned int tamanho);
```

Declara os  
protótipos das  
funções que serão  
usadas.

```
void setup()  
{  
  Serial.begin(9600);  
  conectaWifi();  
  MQTT.setServer(mqtt_broker, mqtt_port);  
  MQTT.setCallback(callback);  
}
```

**Função SETUP:**

- Configura a comunicação serial a um baudrate de 9600bps.
- Chama a função conectaWifi
- Conecta-se ao broker mqtt.
- MQTT.setCallback → define uma função de retorno de chamada (neste caso o nome da função é callback, mas poderia ser qualquer outro).

```
void conectaWifi()  
{  
  Serial.println("Conectado a rede wifi!");  
  WiFi.begin(ssid, password);  
  while (WiFi.status() != WL_CONNECTED)  
  {  
    delay(500);  
    Serial.println("Conectando a rede wifi....");  
  }  
  Serial.println("Conectado a rede wifi!");  
}
```

A função  
conectaWifi é  
responsável por  
fazer a conexão com  
a rede wifi. Neste  
caso não é feito um  
“timeout”...ele fica  
tentando.



```
void conectaBroker()
{
  while(!MQTT.connected())
  {
    if(MQTT.connect(MQTT_ID))
    {
      Serial.println("Conectado ao Broker!");
      MQTT.subscribe(topico_sub_temp);
      MQTT.subscribe(topico_sub_umid);
      MQTT.subscribe(topico_sub_rele);
    }
    else
    {
      Serial.print("Falha na conexão. O status é: ");
      Serial.print(MQTT.state());
      delay(2000);
    }
  }
}
```

A função `conectaBroker` é responsável por fazer a conexão com o broker MQTT. Neste caso não é feito um “timeout”...ele fica tentando e reportando os erros de conexão, caso não consiga se conectar. Se ele conectar ao Broker, então o cliente informa que é assinante dos tópicos x, y, z.....

```
void callback(char *topico, byte *msg, unsigned int tamanho)
{
    Serial.print("Mensagem recebida do topico ");
    Serial.println(topico);
    Serial.print("Mensagem: ");
    for (int i = 0; i < tamanho; i++)
    {
        Serial.print((char) msg[i]);
    }
    Serial.println();
    Serial.println("-----");
}
```

```
void loop()
{
    if(WiFi.status() != WL_CONNECTED) conectaWifi();
    if(!MQTT.connected()) conectaMQTT();
    MQTT.loop();
}
```

Na função loop, verifica conexão wifi, verifica a conexão com o Broker e fica mantendo a conexão ativa.

A função callback (ou outro nome qualquer) recebe três parâmetros → topico, mensagem e tamanho do dado. Costuma-se usar os termos topic, payload e lenght, neste exemplo, usei: topico, msg e tamanho.

Sempre que chega no Broker uma nova mensagem no tópico inscrito, a função callback é chamada, isto porque a conexão entre o broker e o ESP estão ativas. Neste caso, a mensagem é escrita na serial.



Como filtrar apenas o dado de um tópico (ex.: Terroso/Temp)?

```
if(String(topico) == "Terroso/Temp")
{
    for (int i = 0; i < tamanho; i++)
    {
        Serial.print((char) msg[i]);
        msg_temp += (char) msg[i];
    }
}
```

# OBRIGADO!!!!!!

## Baixe o material da Palestra

## acesse o github

## <https://github.com/aterroso/SemanaMQTT>