Material Apoio módulo 5

MQTT é um protocolo de mensagens leve para sensores e pequenos dispositivos móveis otimizado para redes TCP/IP. O esquema de troca de mensagens é fundamentado no modelo Publicador-Subscritor, extremamente simples e leve. O protocolo segue princípios arquitetônicos que minimizam o uso de banda de rede e de recursos dos equipamentos, enquanto provê confiabilidade e algum nível de garantia de entrega.

Broker MQTT

O broker desempenha um papel fundamental no protocolo MQTT (Message Queuing Telemetry Transport) e é um componente central em um sistema de comunicação MQTT. Ele atua como um intermediário entre os dispositivos que publicam (publishers) e os dispositivos que se inscrevem (subscribers) em tópicos específicos, gerenciando a troca de mensagens entre eles. O broker é responsável por rotear as mensagens corretamente e garantir que elas cheguem aos destinatários apropriados. Aqui estão as principais funções do broker MQTT:

1. Receber e encaminhar mensagens:
   Quando um dispositivo publica uma mensagem em um tópico específico, o broker recebe essa mensagem e a encaminha para todos os dispositivos que estão inscritos nesse mesmo tópico. Dessa forma, o broker assegura que todas as partes interessadas recebam as informações relevantes.
2. Gerenciar inscrições e publicações:
   O broker mantém um registro de todas as inscrições feitas pelos dispositivos em tópicos específicos. Quando um dispositivo publica uma mensagem em um tópico, o broker verifica seu registro e envia essa mensagem para todos os dispositivos inscritos nesse tópico.
3. Controle de Qualidade de Serviço (QoS):
   O broker lida com os diferentes níveis de QoS (Quality of Service) suportados pelo MQTT. Quando uma mensagem é publicada, o cliente especifica o nível de QoS desejado para a entrega da mensagem. O broker gerencia a entrega de acordo com esse nível de QoS, garantindo que as mensagens sejam entregues conforme os requisitos de confiabilidade.
4. Persistência de mensagens (opcional):
   Alguns brokers podem oferecer a opção de armazenar as mensagens

temporariamente, caso o destinatário não esteja disponível no momento da publicação. Isso permite que o destinatário recupere a mensagem assim que estiver online novamente.

5. Autenticação e controle de acesso:
   O broker também é responsável por garantir a segurança da comunicação MQTT. Ele pode exigir autenticação dos dispositivos antes de permitir que eles publiquem ou se inscrevam em tópicos específicos. Além disso, o broker pode controlar o acesso a determinados tópicos com base nas permissões configuradas.
6. Manter conexões com os clientes:
   O broker mantém conexões persistentes com os clientes para permitir uma comunicação contínua e assíncrona. Isso é especialmente importante no contexto da IoT, onde os dispositivos podem estar conectados e desconectados frequentemente.

Em resumo, o broker é uma peça central e crítica na arquitetura MQTT, garantindo a entrega eficiente e confiável de mensagens entre os dispositivos conectados. Sua função é essencial para o funcionamento efetivo do sistema de comunicação MQTT, tornando possível a implementação de cenários IoT escaláveis e distribuídos.

Porque o protocolo MQTT é importante?

1. **Leve e eficiente**: A implementação do MQTT no dispositivo IoT requer recursos mínimos, podendo ser usado até mesmo em pequenos microcontroladores. Por exemplo, uma mensagem de controle MQTT mínima pode ter apenas dois bytes de dados. Os cabeçalhos de mensagens MQTT também são pequenos para que você possa otimizar a largura de banda da rede.
2. **Escalável**: A implementação do MQTT requer uma quantidade mínima de código que consome pouquíssima energia nas operações. O protocolo também tem recursos integrados que oferecem suporte à comunicação com um grande número de dispositivos IoT. Portanto, você pode implementar o protocolo MQTT para se conectar a milhões desses dispositivos.
3. **Confiável**: Muitos dispositivos IoT se conectam em redes celulares não confiáveis com baixa largura de banda e alta latência. O MQTT tem recursos integrados que reduzem o tempo que o dispositivo IoT leva para se reconectar à nuvem. Além disso, define três níveis diferentes de qualidade de serviço para garantir a confiabilidade para casos de uso de IoT: no máximo uma vez (0), pelo menos uma vez (1) e exatamente uma vez (2).

4. **Seguro**: Com o MQTT, os desenvolvedores têm mais facilidade para criptografar mensagens e autenticar dispositivos e usuários usando protocolos de autenticação modernos, como OAuth, TLS1.3, certificados gerenciados pelo cliente etc.
5. **Bom suporte**: Muitas linguagens, como Python, têm amplo suporte para implementação do protocolo MQTT. Portanto, os desenvolvedores podem implementá-lo rapidamente com codificação mínima em qualquer tipo de aplicação.

Como o MQTT funciona?

1. Um cliente MQTT estabelece uma conexão com o agente MQTT.
2. Depois de conectado, o cliente pode publicar mensagens, assinar mensagens específicas ou fazer as duas coisas.
3. Ao receber uma mensagem, o agente MQTT a encaminha aos assinantes interessados.

Tópico do MQTT

O termo “tópico” refere-se a palavras-chave que o agente 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
```

Publicação de MQTT

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*.

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*.

Praticando com o mosquitto

Faça a instalação do mosquitto pelo site: Eclipse Mosquitto

Ao fazer a instalação no ambiente windows, rode o seguinte comando no PowerShell para inicializar o MQTT:

```
./mosquitto -v
```

Para se inscrever em algum tópico deste broker, você precisará executar a aplicação mosquitto_sub. Abra outro terminal e rode este comando para assinar um tópico temperatura na instância do mosquitto que está rodando na porta 1883:

```
./mosquitto_sub -h localhost -p 1883 -t temperatura
```

Para publicar dados neste tópico, basta executar o mosquitto_pub. Abra outro terminal e rode:

```
./mosquitto_pub -h localhost -p 1883 -t temperatura -m 20
```

Para encerrar qualquer um dos serviços, digite ctrl+C em seu teclado.

Definindo regras de usuários

Para definir regras de usuário, precisamos editar o arquivo mosquitto.conf. O arquivo mosquitto.conf contém configurações de execução do broker. Para editá-lo, você deve ter permissão de administrador.

Para definir usuários, primeiramente você deve proibir o acesso de dispositivos anônimos. Portanto, na seção de Security do arquivo insira esta configuração:

```
# ========
# Security
# ========

allow_anonymous false
```

Ao definir que o cliente não pode ser anônimo, obrigatoriamente deverá existir um login e senha. Para isto, utilizaremos o mosquitto_passwd. Abra o powershell no modo administrador e rode o comando abaixo para criar um novo arquivo chamado passwordfile.pwd com o usuário admin.

```
mosquitto_passwd -c passwordfile.pwd admin
```

Ao executar, será solicitado uma senha deste usuário. Ao encerrar, o arquivo é criado. Você deverá copiar o caminho de acesso deste arquivo e inserir no mosquitto.conf.

Este ultimo passo é feito dentro da seção de ACL’s (access control) do arquivo mosquitto.conf

```
password_file C:\Program Files\mosquitto\passwordfile.pwd
```

Utilizando o mosquitto com usuários

O comando para executar o broker com o arquivo de configuração é este:

```
./mosquitto -c mosquitto.conf -v
```

Para autenticar um subscriber, coloque o usuário e senha:

```
mosquitto_sub -h localhost -p 1883 -u admin -P 123 -t temperatura
```

O mesmo vale para o publisher:

```
mosquitto_pub -h localhost -p 1883 -u admin -P 123 -t temperatura -m 25
```

Incluindo e deletando usuários do arquivo passwordfile

Para incluir novos usuários no broker, utilize o mesmo programa mosquitto_passwd com o parâmetro -b. Insira o nome do usuário e a senha. Lembre-se de utilizar o powershell no modo administrador.

```
mosquitto_passwd -b passwordfile user1 123
```

Comando para deletar um usuário:

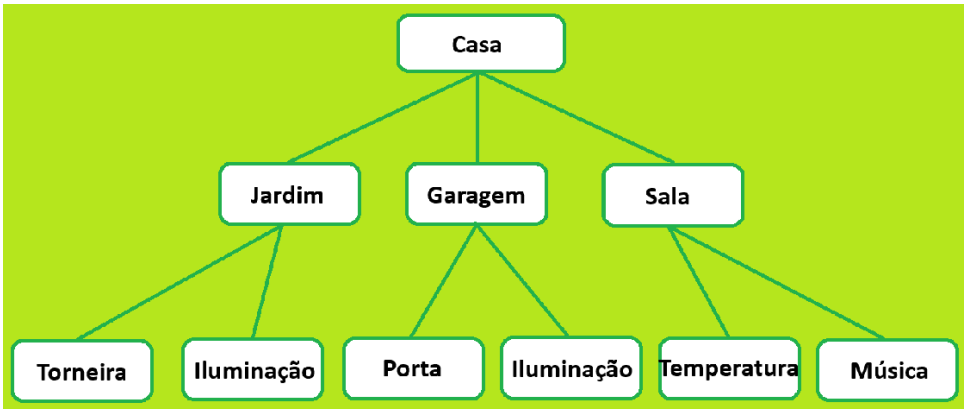
```
mosquitto_passwd -D passwordfile user1
```

Comando para atualizar os hashs dos usuários. Você pode rodar este comando como medida de segurança ao lançar seu produto.

```
mosquitto_passwd -U passwordfile
```

WildCards

+ → serve como coringa, ao colocá-lo no lugar de um tópico, você diz ao broker que deseja assinar em todos os tópicos daquele nível de hierarquia. Imagine uma casa com sensores em vários cômodos e que os tópicos estejam hierarquizados da seguinte maneira:



Imagine que um dispositivo precisasse se inscrever em todos os tópicos por qualquer motivo. Ele teria que criar rotinas de subscriber para cada um dos sensores e atuadores (6 no total), ou simplesmente utilizar o comando:

```
-t casa/+/+
```

Para simplesmente se inscrever em todos os tópicos e subtópicos disponíveis em casa, você pode utilizar o wildcard `#` .

Regras de ACL’s

ACL (Access control) são os parâmetros de controle de acesso para cada usuário. Você pode definir o que cada usuário tem permissão de fazer e não fazer dentro de seu sistema IoT.

As regras de ACL’s ficam salvas em um arquivo separado. Desta vez, não é necessário a utilização de um programa específico, basta criar um arquivo (pode ser sem extensão) e passar os parâmetros de ACL

```
user admin
topic readwrite #

user user1
topic readwrite temperatura
```

Depois, adicione o caminho deste arquivo no arquivo de configurações principal mosquitto.conf:

```
  acl_file C:\Program Files\mosquitto\aclfile
```

Dizer que **user1** tem permissões somente ao tópico temperatura, não significa que ele será impedido de assinar ou escrever em outros tópicos. Mas ele não terá acesso aos valores.

Certificado de segurança TLS/SSL

O certificado de segurança serve para criptografar os dados que transitam dentro de um protocolo TCP/IP. Para criar um certificado digital, utilizamos o software openSSL.

- Do lado do cliente MQTT deverá ser criado um certificado CA (certificate authority) que será reconhecido pelo Mosquitto.

- Do lado do broker MQTT deverá ser criado um certificado CA e um Server Private Key

Para executar o programa que gera os certificados, acesse a pasta do OpenSSL que foi instalada em seu computador e entre na pasta bin. Dentro da pasta, abra o powershell no **modo administrador.**

Primeiramente, vamos gerar a chave para o cliente. O comando a seguir gera o arquivo ca.key

```
./openssl genrsa -des3 -out ca.key 2048
```

Informe uma senha de no mínimo 4 dígitos

No próximo comando será gerado o certificado ca.crt. Você irá informar vários campos como o endereço IP da sua máquina. Repare que se você estiver em um computador na nuvem, o endereço deverá ser o deste computador

```
./openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
```

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:BR
State or Province Name (full name) [Some-State]:Parana
Locality Name (eg, city) []:Jesuitas
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Instructiva
Organizational Unit Name (eg, section) []:Instructiva
Common Name (e.g. server FQDN or YOUR name) []:localhost
Email Address []:vht.luna@gmail.com
```

O certificado do cliente está gerado. Agora, é necessário criar o certificado do servidor. O próximo comando gera o server.key

```
./openssl genrsa -out server.key 2048
```

Agora, gerar o arquivo server.csr para o servidor:

```
./openssl req -new -out server.csr -key server.key
```

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:BR
State or Province Name (full name) [Some-State]:Parana
Locality Name (eg, city) []:Jesuitas
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Escola
Organizational Unit Name (eg, section) []:Escola
Common Name (e.g. server FQDN or YOUR name) []:localhost
Email Address []:vht.luna@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:1234
An optional company name []:.
```

Faça a verificação dos certificados:

```
./openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out
server.crt -days 306
```

Os arquivos criados foram este. 3 do lado do cliente e 3 do lado do servidor

Nome	Data de modificação	Tipo	Tamanho
server	05/10/2023 17:55	Certificado de Seg...	2 KB
ca.srl	05/10/2023 17:55	Arquivo SRL	1 KB
server.csr	05/10/2023 17:51	Arquivo CSR	2 KB
server.key	05/10/2023 17:36	Arquivo KEY	2 KB
ca	05/10/2023 17:31	Certificado de Seg...	2 KB
ca.key	05/10/2023 17:12	Arquivo KEY	2 KB

Crie uma pasta chamada **certs** dentro da pasta mosquitto e cole os certificados lá dentro. Depois, adicione o caminho destes certificados no mosquitto.conf.

Configurações no mosquitto.conf

Primeiramente, é necessário alterar a porta do mosquitto para a porta 8883, pois esta é a mais recomendada para comunicações TCP com certificados TLS. Faça esta configuração na sessão de Listeners

```
# ================
# Listeners
# ================

listener 8883
```

Adicione os caminhos dos certificados:

```
# "openssl rehash <path to capath>" each time you add/remove a certificate.
cafile C:\Program Files\mosquitto\certs\ca.crt

# Path to the PEM encoded server certificate.
certfile C:\Program Files\mosquitto\certs\server.crt

# Path to the PEM encoded keyfile.
keyfile C:\Program Files\mosquitto\certs\server.key
```

Utilizando o broker

Lembre-se de alterar a porta e inserir o certificado através de `--cafile` . Para o comando abaixo, é necessário estar com terminal aberto dentro do diretório com os certificados.

```
mosquitto_sub -h localhost -p 8883 -u admin -P 123 --cafile ca.crt -t temperatura
```

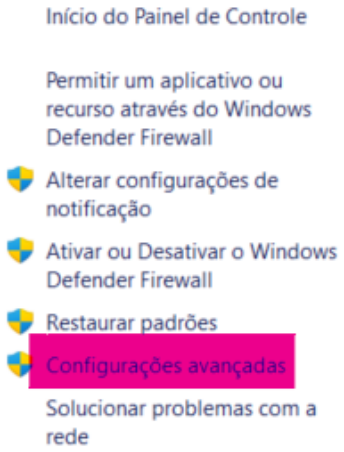
```
mosquitto_pub -h localhost -p 8883 -u admin -P 123 --cafile ca.crt -t temperatura -m 25
```

Comunicação da ESP32 com o Broker via MQTT

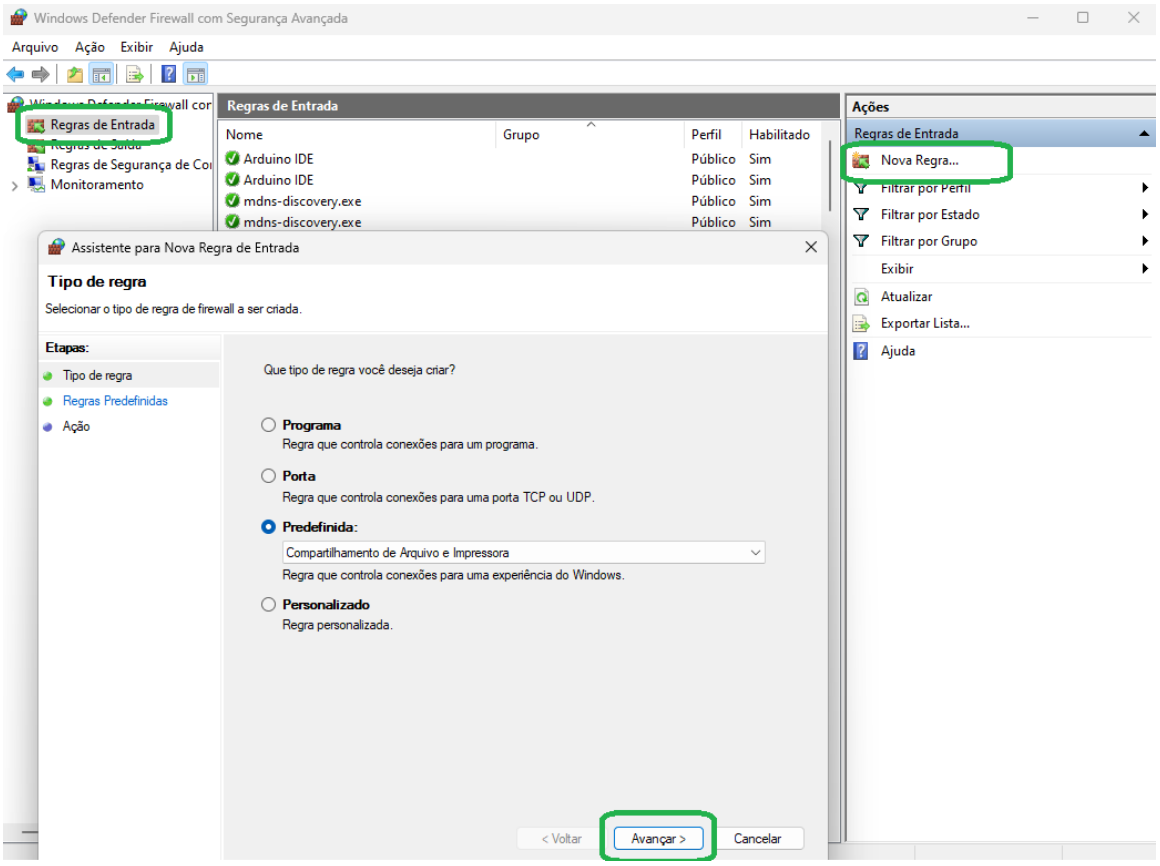
Finalmente iremos aplicar na prática os conhecimentos adquiridos neste módulo e realizar uma comunicação MQTT real entre a ESP32 e o broker que está rodando em nossa rede local.

Primeiramente, o windows não foi feito para ser utilizado como um servidor nem como um broker. Por isso, para que a conexão ocorra, é necessário permitir esta

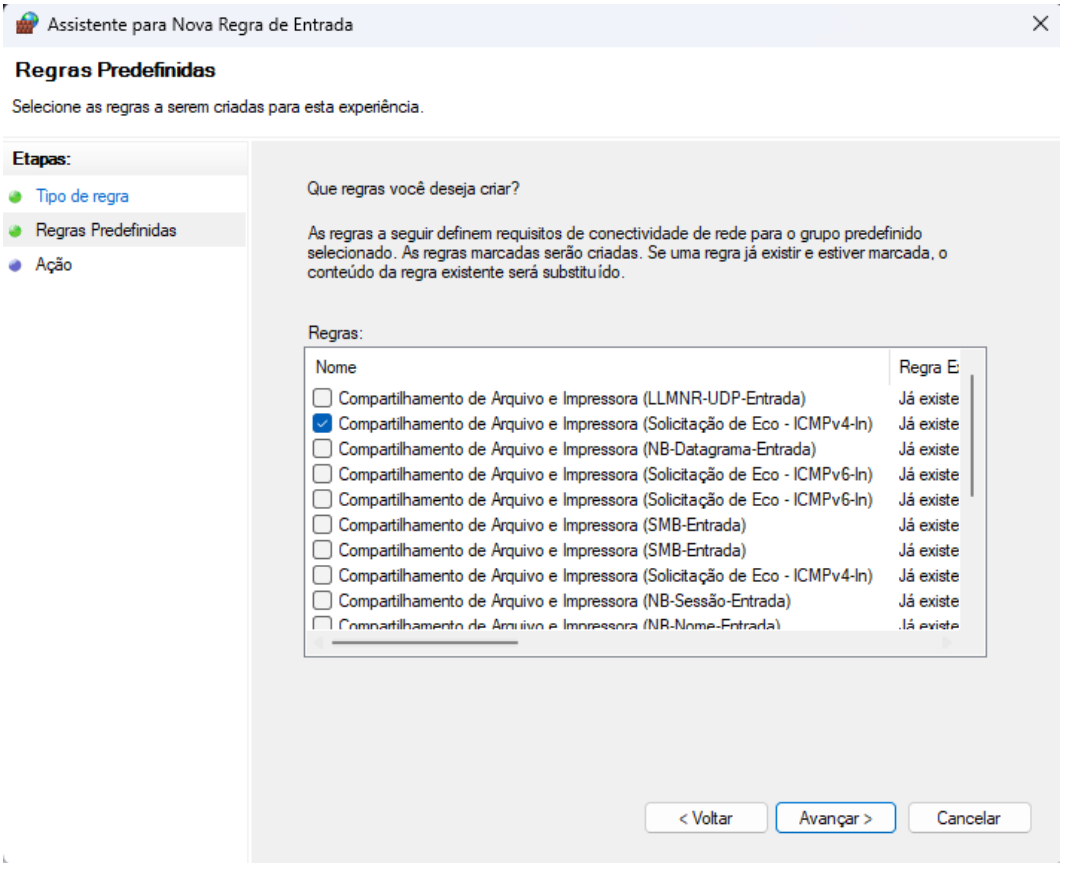
conexão externa. Acesse as configurações no windows firewall e clique em configurações avançadas:



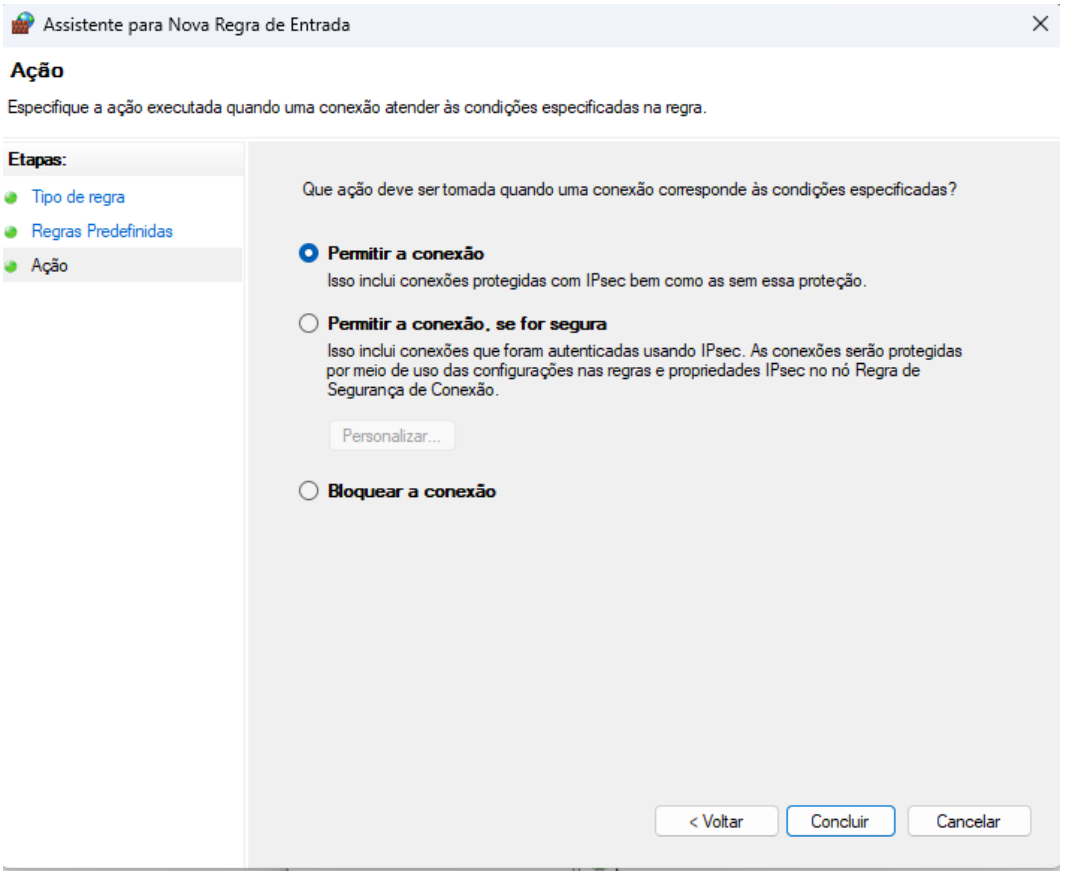
Insira um novo tipo de regra do tipo predefinida:



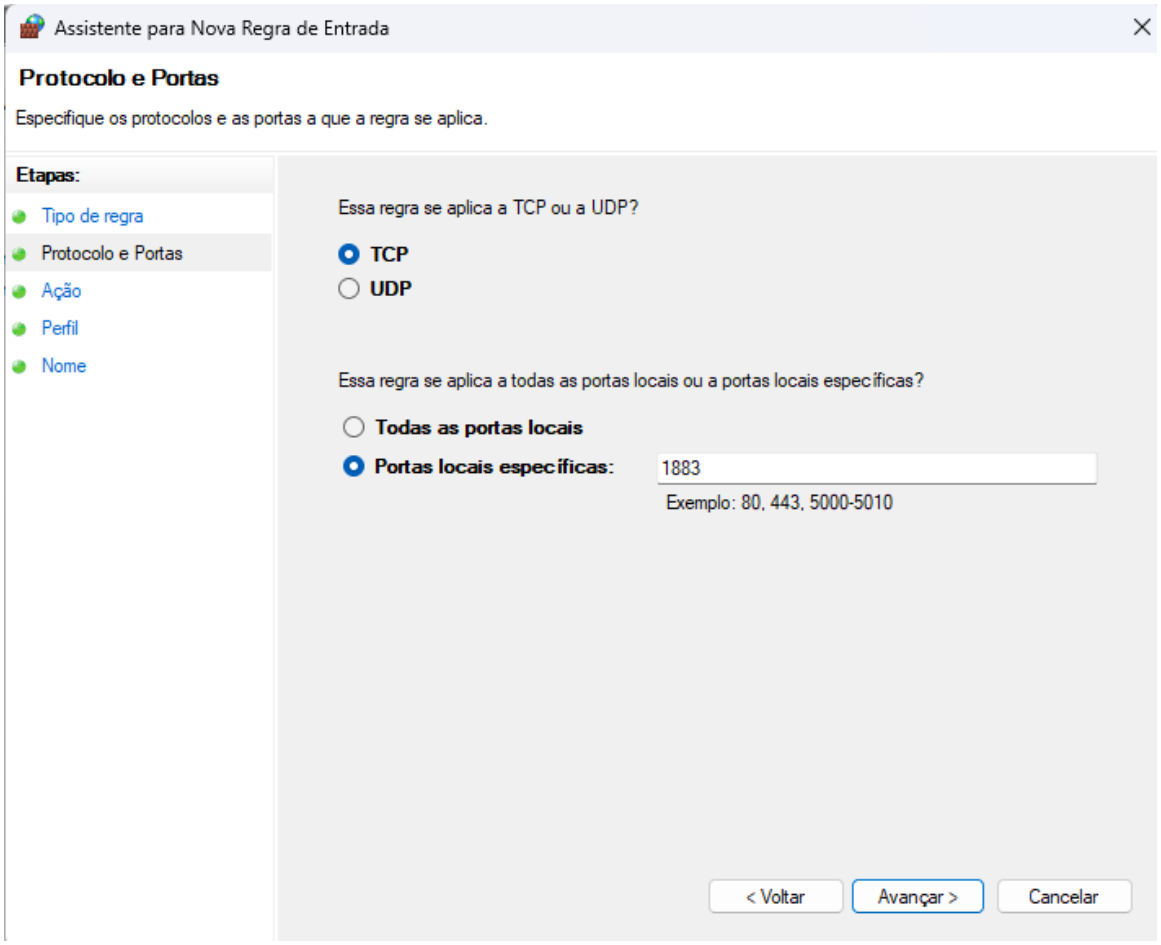
Selecione a regra:



E permita a conexão:



Agora é preciso definir a porta onde a conexão acontecerá. Refaça o processo, mas selecione a opção “Porta” na tela de tipo de regra. Você vai acessar o windows pela porta 1883 no protocolo TCP:



Vá avançando até terminar este processo. Assim você libera esta conexão.

Código da comunicação

Além das bibliotecas Arduino e WiFi, utilizaremos a biblioteca MQTT:

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

// lib_deps = 256dpi/MQTT@^2.5.1
#include <MQTT.h>
```

Declare os objetos de WiFi e MQTT

```
WiFiClient net;
MQTTClient client;
```

Precisamos de uma função que faz a conexão com o broker, passando usuário e senha, e se inscreve em algum tópico (se necessário):

```
void connect() {
  // Conecta no WiFi
  Serial.print("checking wifi...");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(1000);
  }

  // Conecta com o broker
  Serial.print("\nconnecting...");
  // Informa o ID (que precsisa ser unico), usuário e senha do broker
  while (!client.connect("ESP32", "admin", "123")) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("\nconnected!");

  // Se inscreve em um tópico
  client.subscribe("hello");
  // client.unsubscribe("/hello");
}
```

Precisamos de uma função que recebe a mensagem vinda através de um tópico. Aqui, esta função somente retorna o dado pela serial, mas poderia fazer qualquer lógica que você quiser, como ligar acionar GPIO’s e controlar PWM:

```
void messageReceived(String &topic, String &payload) {
  Serial.println("incoming: " + topic + " - " + payload);
}
```

Faça todas as conexões com o endereço IP do broker:

```
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, pass);

  client.begin("192.168.248.13", net);
  client.onMessage(messageReceived);

  connect();
}
```

Por fim, o loop vai sempre tentar a conexão tanto WIFI quando MQTT. A ESP32 estará inscrita nos tópicos antes definido e também enviará um valor no tópico temperatura:

```
void loop() {
  client.loop();

  if (!client.connected()) {
    connect();
  }

  // publish a message roughly every second.
  if (millis() - lastMillis > 1000) {
    lastMillis = millis();
    client.publish("temperatura", "20");
  }
}
```