



# Facens

FACULDADE DE ENGENHARIA DE SOROCABA

## **ESPECIALIZAÇÃO EM SEGURANÇA CIBERNÉTICA**

**SEGURANÇA APLICADA A IOT**

**DISPOSITIVO IOT CONECTADO À  
PLATAFORMA THINGSBOARD COM  
HTTPS E MQTT SEGURO**

Filipe Sousa RA 163610  
Gelson Filho RA 160157

Prof. Eng. MSc. Leandro Avanço

## 1. Objetivos

O presente trabalho tem por objetivo apresentar os requisitos da avaliação final da disciplina de Segurança Aplicada à IoT, ministrada no curso de Pós-Graduação em Segurança Cibernética, da instituição de ensino Centro Universitário FACENS.

Tais requisitos consistem na implementação de um dispositivo IoT simulado em linguagem Python, conectado de forma segura à plataforma Thingsboard, utilizando-se de recursos de certificados digitais através dos protocolos de HTTPS e MQTT seguro, além disso, devem ser implementados eventos de alarme gerados pelo dispositivo que enviarão notificações via Telegram.

Portanto, baseando-se no que foi apresentado em aula, deve-se:

- Instalar a plataforma Thingsboard.
- Programar o dispositivo simulado para enviar dados de temperatura e umidade para a plataforma.
- Implementar o protocolo HTTPS na interface web padrão do Thingsboard.
- Implementar comunicação segura no protocolo MQTT.
- Avaliar reconfigurações do container Docker para utilizar as portas de comunicação seguras.
- Programar o dispositivo para comunicação nas portas seguras.
- Criar regras de alarme no Thingsboard.
- Notificar via Telegram.

Tais configurações e desenvolvimentos serão apresentados no capítulo seguinte. Espera-se que ao final do desenvolvimento seja possível demonstrar os conhecimentos e habilidades adquiridas durante o curso.

## 2. Desenvolvimento

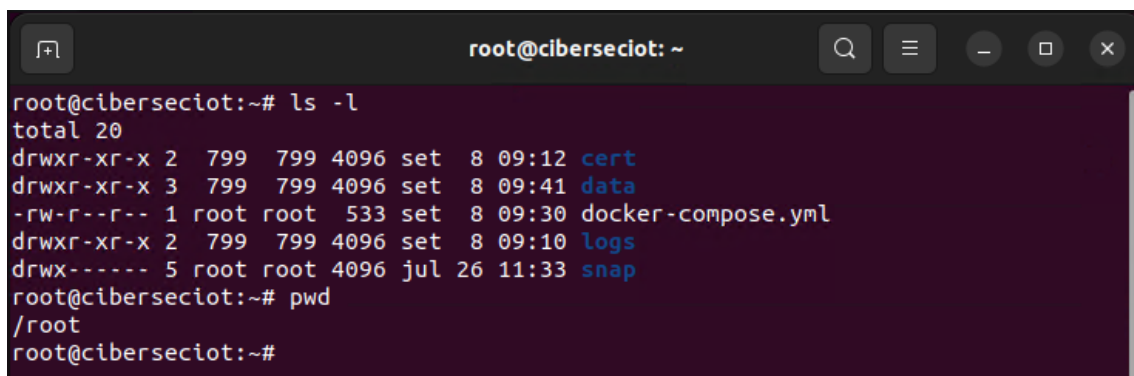
Este capítulo apresentará todos os passos e configurações que foram aplicados a fim de desenvolver um ambiente seguro através da plataforma Thingsboard, por meio da qual é possível realizar a comunicação segura com um dispositivo IoT simulado.

### 2.1 Instalação da plataforma Thingsboard

Para realizar a instalação da plataforma Thingsboard foi utilizada uma máquina virtual (VM), através do software VirtualBox, com o sistema operacional Ubuntu, VM na qual foi instalado também o software Docker, responsável por implantar aplicativos dentro de containers virtuais, muito semelhante à virtualização de nível de sistema operacional.

Na máquina host (Ubuntu), seleciona-se um diretório para organizar os arquivos do container que será utilizado para instalar a plataforma Thingsboard, no presente caso, optou-se por utilizar a pasta /root para tal função.

Seguindo-se as instruções de instalação do Thingsboard através do docker, disponíveis em: <https://thingsboard.io/docs/user-guide/install/docker/> , cria-se um arquivo denominado “docker-compose.yml”, que é um arquivo de configuração utilizado com a ferramenta “Docker Compose”, o qual descreve os serviços, redes e volumes que compõem um aplicativo Docker. Além disso, criou-se também outras três pastas que serão utilizadas posteriormente para armazenar os certificados criados (cert), montar o diretório de logs (logs) e realizar também a montagem do diretório de banco de dados (data) .



```
root@ciberseciot: ~
root@ciberseciot:~# ls -l
total 20
drwxr-xr-x 2 799 799 4096 set  8 09:12 cert
drwxr-xr-x 3 799 799 4096 set  8 09:41 data
-rw-r--r-- 1 root root 533 set  8 09:30 docker-compose.yml
drwxr-xr-x 2 799 799 4096 set  8 09:10 logs
drwx----- 5 root root 4096 jul 26 11:33 snap
root@ciberseciot:~# pwd
/root
root@ciberseciot:~#
```

Em seguida, utilizando um editor de texto, configura-se o arquivo “Docker-compose.yml” ainda seguindo as instruções de instalação do Thingsboard, e o arquivo é configurado da seguinte forma:

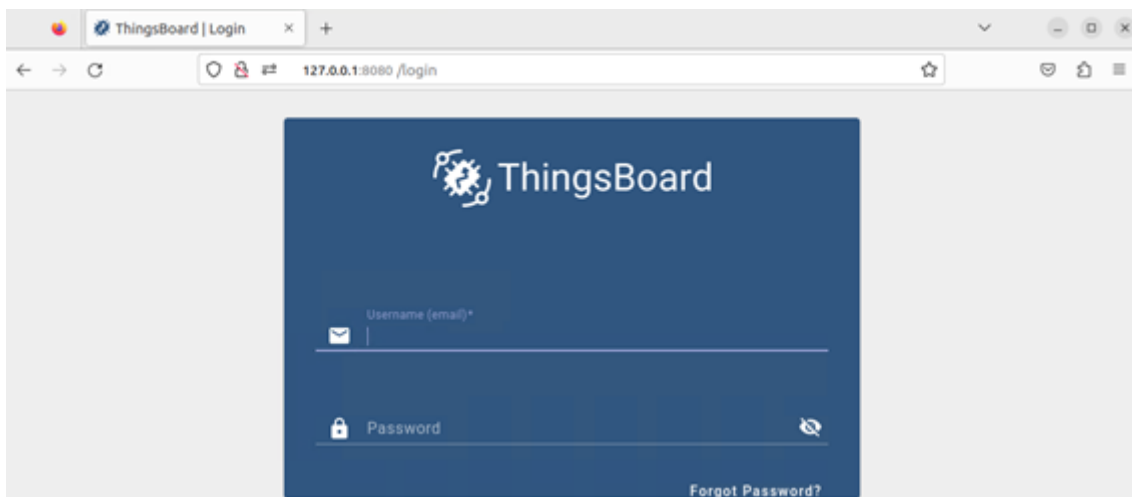
```

GNU nano 6.2
version: '3.0'
services:
  mytb:
    restart: always
    image: "thingsboard/tb-postgres"
    ports:
      - "8080:9090"
      - "1883:1883"
      - "7070:7070"
      - "5683-5688:5683-5688/udp"
    environment:
      TB_QUEUE_TYPE: in-memory
    volumes:
      - /root/data:/data
      - /root/logs:/var/log/thingsboard

```

Após a configuração do arquivo “docker-compose.yml”, é necessário alterar o ownership dos diretórios criados através do comando “chown -R 799:799 <diretório>”, e então se inicia o container através do comando “docker-compose up -d”, sendo a opção “-d” utilizada para iniciar o container em segundo plano.

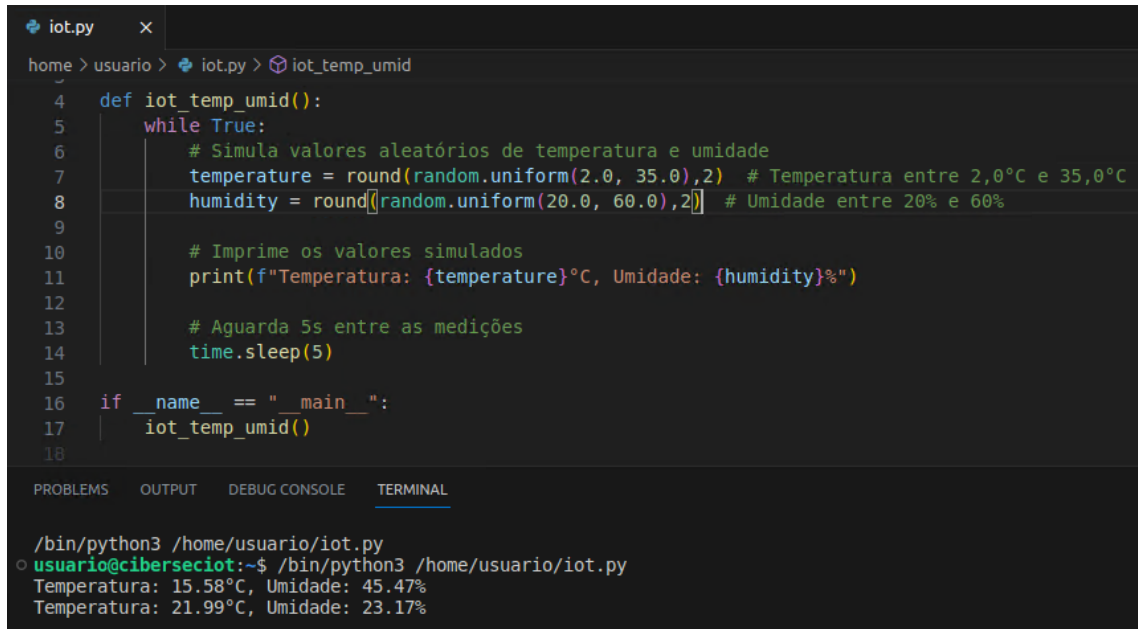
Após a inicialização do container, pode-se através do navegador web acessar o ambiente do ThingsBoard, que já se encontra em execução, através do endereço de IP referente à máquina host, e ao acessar esse endereço, observa-se a tela de login no ambiente da plataforma ThingsBoard.



Observa-se, no entanto, que a conexão está sendo realizada através do protocolo HTTP, que não configura uma conexão segura. Isso ocorre porque o HTTP não criptografa os dados transmitidos entre cliente e servidor, tornando as informações vulneráveis a interceptações maliciosas, além de não autenticar adequadamente os participantes da comunicação, possibilitando ataques de falsificação. Para aprimorar a segurança, no tópico 2.3 será implementado o uso do protocolo HTTPS (HTTP seguro) com criptografia SSL/TLS, proporcionando um ambiente de transmissão de dados mais protegido e confiável.

## 2.2 Programação do simulador de dispositivo IoT

Para realizar a demonstração do funcionamento da plataforma Thingsboard, um dispositivo IoT foi simulado através de um programa em python. Um programa simples foi desenvolvido, no qual valores de temperatura e umidade são gerados de forma aleatória a cada 5 segundos. Posteriormente este programa será adaptado para realizar a comunicação com a plataforma.



```
iot.py x
home > usuario > iot.py > iot_temp_umid
4 def iot_temp_umid():
5     while True:
6         # Simula valores aleatórios de temperatura e umidade
7         temperature = round(random.uniform(2.0, 35.0),2) # Temperatura entre 2,0°C e 35,0°C
8         humidity = round(random.uniform(20.0, 60.0),2) # Umidade entre 20% e 60%
9
10        # Imprime os valores simulados
11        print(f"Temperatura: {temperature}°C, Umidade: {humidity}%")
12
13        # Aguarda 5s entre as medições
14        time.sleep(5)
15
16 if __name__ == "__main__":
17     iot_temp_umid()
18
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
/bin/python3 /home/usuario/iot.py
usuario@ciberseciot:~$ /bin/python3 /home/usuario/iot.py
Temperatura: 15.58°C, Umidade: 45.47%
Temperatura: 21.99°C, Umidade: 23.17%
```

## 2.3 Implementação do protocolo HTTPS na plataforma Thingsboard

Para configurar corretamente o protocolo HTTPS a fim de utilizá-lo no ThingsBoard, faz-se necessário inicialmente adquirir um certificado SSL/TLS válido, que posteriormente será utilizado na configuração do ThingsBoard. Para fins de teste e demonstração, utilizar-se-á um certificado autoassinado, gerado através da ferramenta openssl.

Utilizou-se o comando “openssl ecparam -out server\_key.pem -name secp256r1 -genkey”, que gera um par de chaves EC, utilizadas na criptografia assimétrica para SSL/TLS. Após gerar a chave, utiliza-se o comando “openssl req -new -key server\_key.pem -x509 -nodes -days 365 -out server.pem”, que é responsável por realizar a criação de solicitação de certificados X.509, além disso, os parâmetros utilizados indicam que é um certificado autoassinado (-x509), não protegido por senha (-nodes), validade de 365 dias (-days 365) e salvo em um arquivo “server.pem” (-out server.pem).

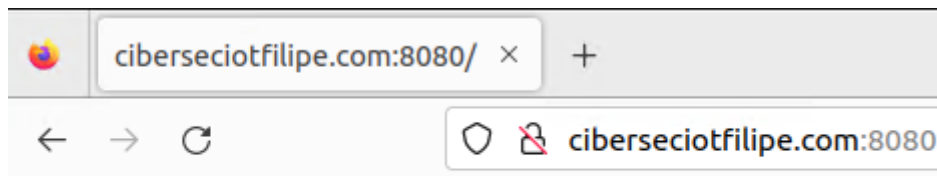
Tendo em mãos a chave privada e o certificado, pode-se então realizar a configuração do container atualizando as variáveis de ambiente do Docker Compose, editando o arquivo “docker-compose.yml”, que foi criado no tópico 2.1. Tanto a chave, quanto o certificado foram movidos para a pasta “cert”, criada anteriormente, conforme a imagem abaixo.

```
root@ciberseciot: ~
root@ciberseciot:~# ls -l
total 20
drwxr-xr-x 2 799 799 4096 set  8 09:12 cert
drwxr-xr-x 3 799 799 4096 set  8 09:41 data
-rw-r--r-- 1 root root 533 set  8 09:30 docker-compose.yml
drwxr-xr-x 2 799 799 4096 set  8 09:10 logs
drwx----- 5 root root 4096 jul 26 11:33 snap
root@ciberseciot:~# ls -l ./cert/
total 8
-rw-r--r-- 1 799 799 920 set  8 09:12 server_filipe_cert.pem
-rw----- 1 799 799 302 set  8 09:11 server_filipe_key.pem
root@ciberseciot:~#
```

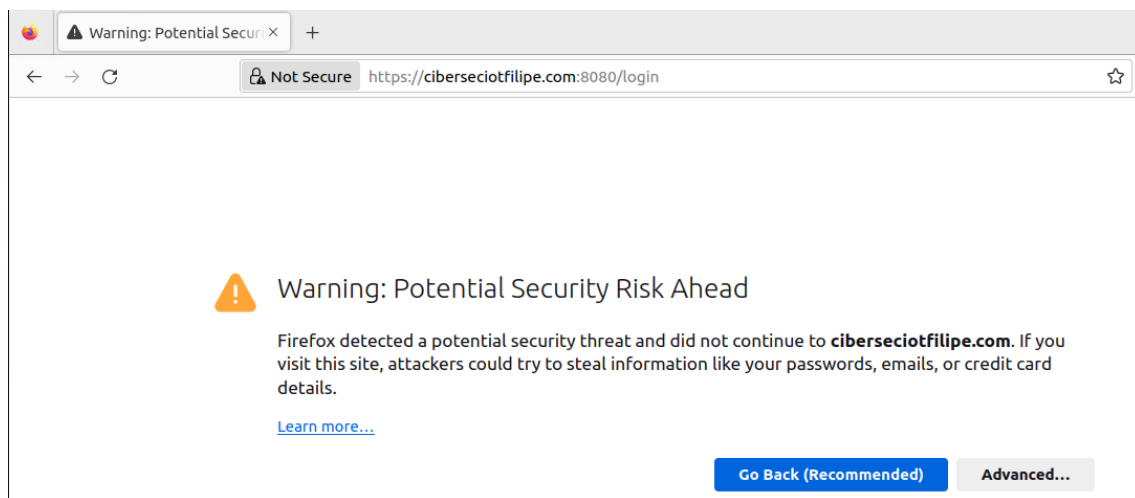
O arquivo “docker-compose.yml” foi editado seguindo a documentação oficial do thingsboard para uso do HTTP over SSL, que configura o uso do HTTPS e que pode ser encontrada em: <https://thingsboard.io/docs/user-guide/ssl/http-over-ssl/>. Seguindo a documentação, as variáveis de ambiente “SSL\_ENABLED”, “SSL\_CREDENTIALS\_TYPE”, “SSL\_PEM\_CERT”, “SSL\_PEM\_KEY” E “SSL\_PEM\_KEY\_PASSWORD” foram adicionadas ao arquivo de configuração, e nelas foram apontados os valores de “true”, o tipo de credencial “PEM”, e os locais onde o certificado e a chave foram salvos, respectivamente.

```
GNU nano 6.2 docker-compose.yml
version: '3.0'
services:
  mytb:
    image: "thingsboard/tb-postgres"
    ports:
      - "8080:9090"
      - "1883:1883"
      - "7070:7070"
      - "5683-5688:5683-5688/udp"
    environment:
      TB_QUEUE_TYPE: in-memory
      SSL_ENABLED: "true"
      SSL_CREDENTIALS_TYPE: PEM
      SSL_PEM_CERT: /config/server_filipe_cert.pem
      SSL_PEM_KEY: /config/server_filipe_key.pem
      SSL_PEM_KEY_PASSWORD:
    volumes:
      - /root/data:/data
      - /root/logs:/var/logs/thingsboard
      - /root/cert:/config
```

Após as alterações no arquivo, realizou-se o reinício do container, e então foi acessado novamente a plataforma através do navegador web, podendo se observar a seguinte tela de aviso:



Esse erro ocorreu devido ao fato de o acesso ter sido realizado através do HTTP, sendo agora necessário o uso do TLS. Ao modificar a url para acessar via HTTPS, observa-se novamente uma tela de aviso:



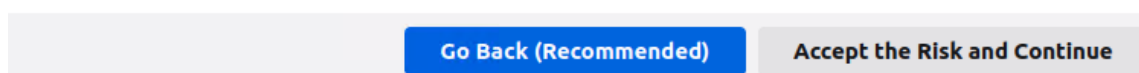
Essa nova tela de aviso se dá pelo fato de estarmos utilizando um certificado autoassinado. Em um ambiente de produção, recomenda-se o uso de um certificado assinado por uma Autoridade de Certificação (CA), de modo a garantir a integridade do certificado utilizado, porém como o foco desta atividade é a demonstração do uso do HTTPS, utilizamos somente um certificado autoassinado, portanto mesmo diante do aviso aceitamos os riscos e adicionamos esse endereço como exceção, pois sabemos sua procedência, entretanto vale ressaltar que o mesmo não deve ser feito em websites comuns, pois isso pode facilitar com que cibercriminosos obtenham informações a partir de websites fraudulentos.

ciberseciotfilipe.com:8080 uses an invalid security certificate.

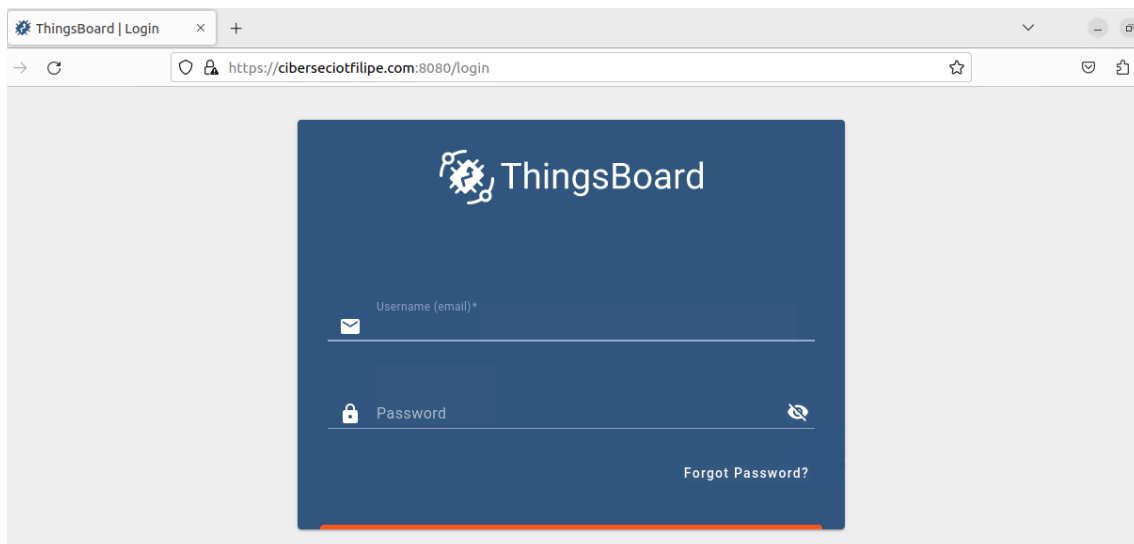
The certificate is not trusted because it is self-signed.

Error code: [MOZILLA\\_PKIX\\_ERROR\\_SELF\\_SIGNED\\_CERT](#)

[View Certificate](#)

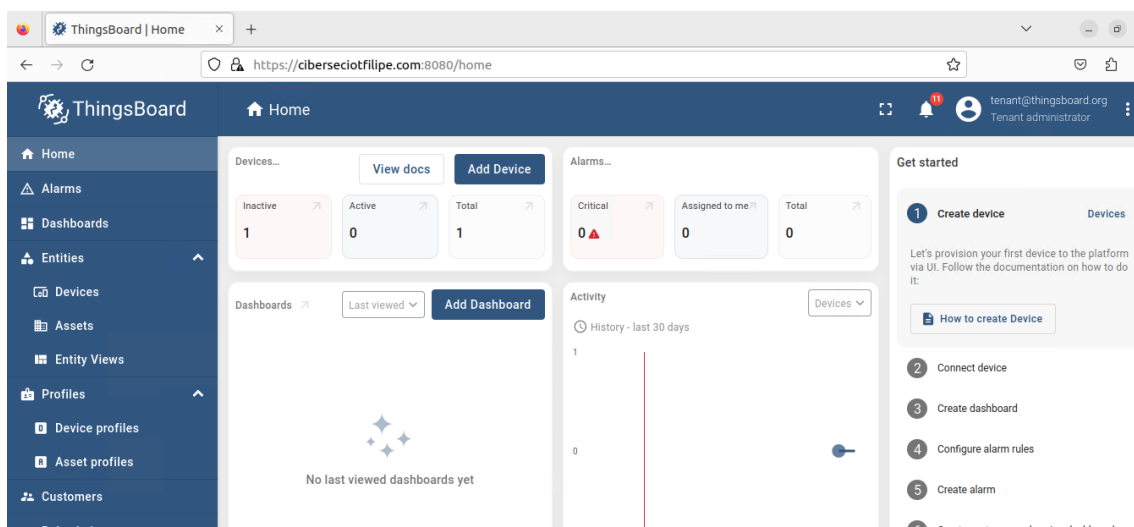


Observa-se então que ao aceitar e continuar, pode-se ter acesso ao portal da plataforma através do uso do HTTPS, conforme a imagem abaixo.

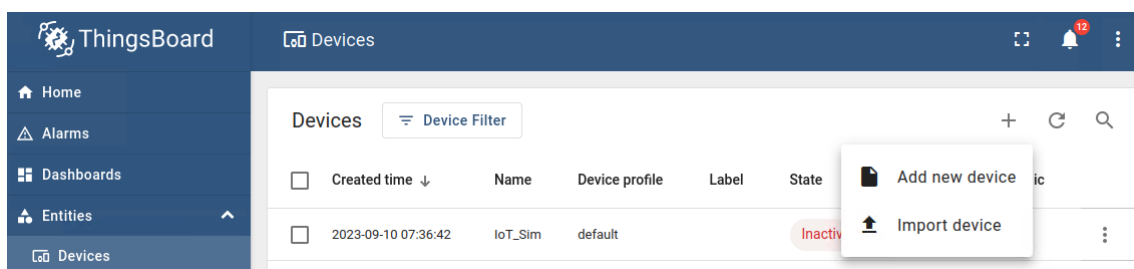


## 2.4 Criando um dispositivo no ThingsBoard

Agora que a conexão ao ThingsBoard está sendo realizada de forma mais segura, através do protocolo HTTPS ao invés do HTTP, dá-se prosseguimento ao trabalho realizando a criação do dispositivo na plataforma. Para que isso seja possível, realiza-se o login através do usuário padrão “Tenant Administrator”.



No menu esquerdo, seleciona-se a opção “Devices”, acessando então a tela na qual são gerenciados os dispositivos. Nesta tela, seleciona-se o sinal de soma e em seguida, deve-se selecionar a opção de “Add new device” e nomear este novo dispositivo, concluindo assim esta etapa.

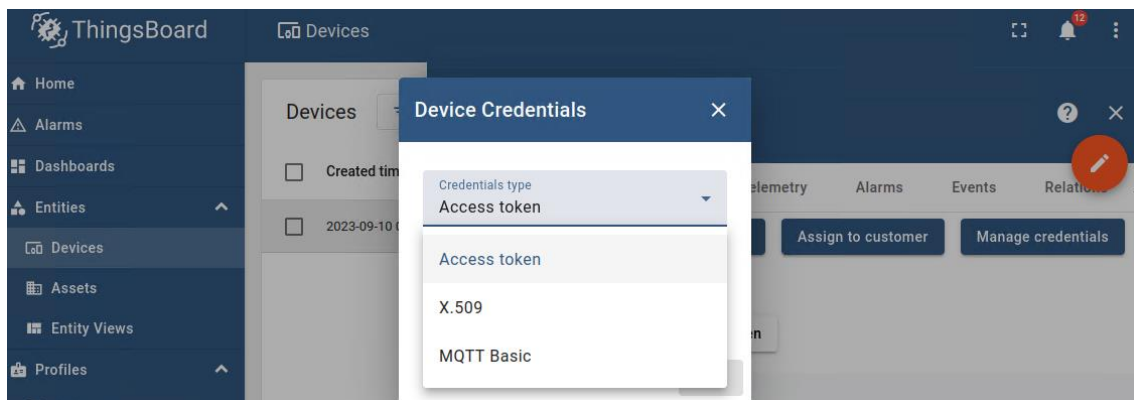




## 2.5 Configurando o dispositivo e o simulador para utilizar o MQTT

Para realizar a comunicação entre os dispositivos e a plataforma Thingsboard, o protocolo de comunicação MQTT é utilizado. De acordo com a [mqtt.org](https://mqtt.org), este protocolo de comunicação para Internet das Coisas (IoT) foi projetado como um sistema de publicação e subscrição extremamente leve, sendo ideal para realizar a conexão de dispositivos remotos, com baixa sobrecarga de codificação e consumo de largura de banda. Atualmente o MQTT é amplamente utilizado em diversas indústrias, incluindo a automotiva, de telecomunicações, óleo e gás, manufatura, entre outras.

Para permitir que o dispositivo criado anteriormente se comunique com o simulador que criamos através do MQTT, é necessário realizar a configuração das credenciais de acesso através da tela de “Devices”. Ao selecionar um dispositivo, uma nova aba de configurações será exibida. Nessa nova aba, deve-se selecionar a opção “Manage Credentials”, que permite selecionar o método de autenticação que será utilizado entre o dispositivo e a plataforma, podendo selecionar as opções “Access Token”, “X.509” ou “MQTT Basic”.

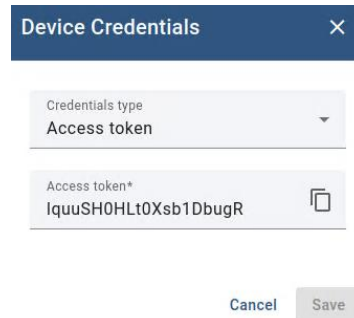


As diferenças entre essas opções é esclarecida na documentação do ThingsBoard, e pode ser acessada em <https://thingsboard.io/docs/user-guide/device-credentials/>.

Dentre essas opções, destaca-se que:

- **Basic MQTT Credentials:** Esse tipo de autenticação é a menos segura entre as três opções, requerindo um usuário e senha para autenticação, e as credenciais são enviadas em texto simples, tornando-as vulneráveis a interceptação caso a comunicação não seja protegida por TLS/SSL. Além disso, essa autenticação é mais suscetível a ataques de força bruta.
- **Access Token:** É similar ao Basic MQTT, mas é baseado na troca de Tokens entre cliente e servidor, que é uma forma muito comum de autenticação em sistemas IoT, oferecendo um nível razoável de segurança quando utilizados adequadamente, em combinação com outras medidas de segurança como TLS/SSL.
- **X.509 (Certificados Digitais):** Este é o método mais seguro de autenticação, pois utiliza certificados digitais para autenticar tanto cliente, quanto servidor. É um padrão amplamente reconhecido em infraestrutura de chave pública (PKI) e são utilizados em modo SSL de duas vias. Essa abordagem é altamente resistente a ataques de falsificação e oferece uma camada de criptografia forte.

Inicialmente utilizaremos o método de Access Token para demonstrar a comunicação entre o simulador e a plataforma, ajustando posteriormente o código a fim de utilizar os certificados digitais da mesma forma que utilizamos anteriormente durante a implementação do HTTPS. Copiamos o Access Token gerado, para então modificar o código e realizar a comunicação.



Device Credentials

Credentials type  
Access token

Access token\*  
lquuSH0HLt0Xsb1DbugR

Cancel Save

Algumas modificações são então aplicadas no código do tópico 2.2 a fim de permitir que ocorra a comunicação entre o simulador e o dispositivo criado:

- É adicionada a biblioteca Paho MQTT, para habilitar a comunicação MQTT.

```
1 import random
2 import time
3 import paho.mqtt.client as mqtt
```

- São definidas as configurações MQTT, que incluem a definição do endereço do servidor, o access token e o topic que especifica os dados a serem publicados.

```
5 # Configurando o MQTT
6 thingsboard_host = "ciberseciotfilipe.com"
7 access_token = "lquuSH0HLt0Xsb1DbugR"
8 topic = "v1/devices/me/telemetry"
```

- Cria-se um cliente MQTT, e configura-se uma função de call-back para indicar o status de conexão.

```
10 # Função para exibir sucesso ou falha de conexão
11 def on_connect(client, userdata, flags, rc):
12     if rc == 0:
13         print("Conexão estabelecida com sucesso")
14     else:
15         print(f"Erro na conexão (Código: {rc})")
16
17 # Função para exibir sucesso ou falha de conexão
18 def iot_temp_umid():
19     # Cria um cliente MQTT
20     client = mqtt.Client()
21     client.on_connect = on_connect
```

- Configura-se o cliente MQTT com as informações de Access Token, endereço do servidor e a porta “1883”, que é a porta padrão do MQTT.

```
23 # Conecta-se ao ThingsBoard
24 client.username_pw_set(access_token)
25 client.connect(thingsboard_host, 1883)
```

- Por fim, no loop principal é adicionada a criação de um payload JSON com os valores de temperatura e umidade, sendo então publicados no tópico MQTT do ThingsBoard e imprime-se no console esses valores.

```
27 while True:
28     # Simula valores aleatórios de temperatura e umidade
29     temperature = round(random.uniform(2.0, 35.0), 2) # Temperatura entre 2,0°C e 35,0°C
30     humidity = round(random.uniform(20.0, 60.0), 2) # Umidade entre 20% e 60%
31
32     # Cria um payload JSON com temperatura e umidade
33     payload = f'{{"temperature": {temperature}, "humidity": {humidity}}}'
34
35     # Publica os dados no tópico MQTT do ThingsBoard
36     client.publish(topic, payload)
37     print(f"Dados publicados: {payload}")
38
39     # Aguarda 5s entre as medições
40     time.sleep(5)
41
42 if __name__ == "__main__":
43     iot_temp_umid()
```

Tendo configurado corretamente essas informações, pode-se observar ao executar o código que o dispositivo do ThingsBoard já está recebendo as informações geradas:

The screenshot displays the ThingsBoard web interface and a terminal window. The terminal window, titled 'iot.py - Visual Studio Code', shows the execution of a Python script. The script connects to ThingsBoard and publishes data to a topic. The terminal output shows the following messages:

```
usuario@ciberseciot:~$ /bin/python3 /home/usuario/iot.py
Dados publicados: {"temperature": 22.3, "humidity": 56.97}
Dados publicados: {"temperature": 2.52, "humidity": 51.91}
```

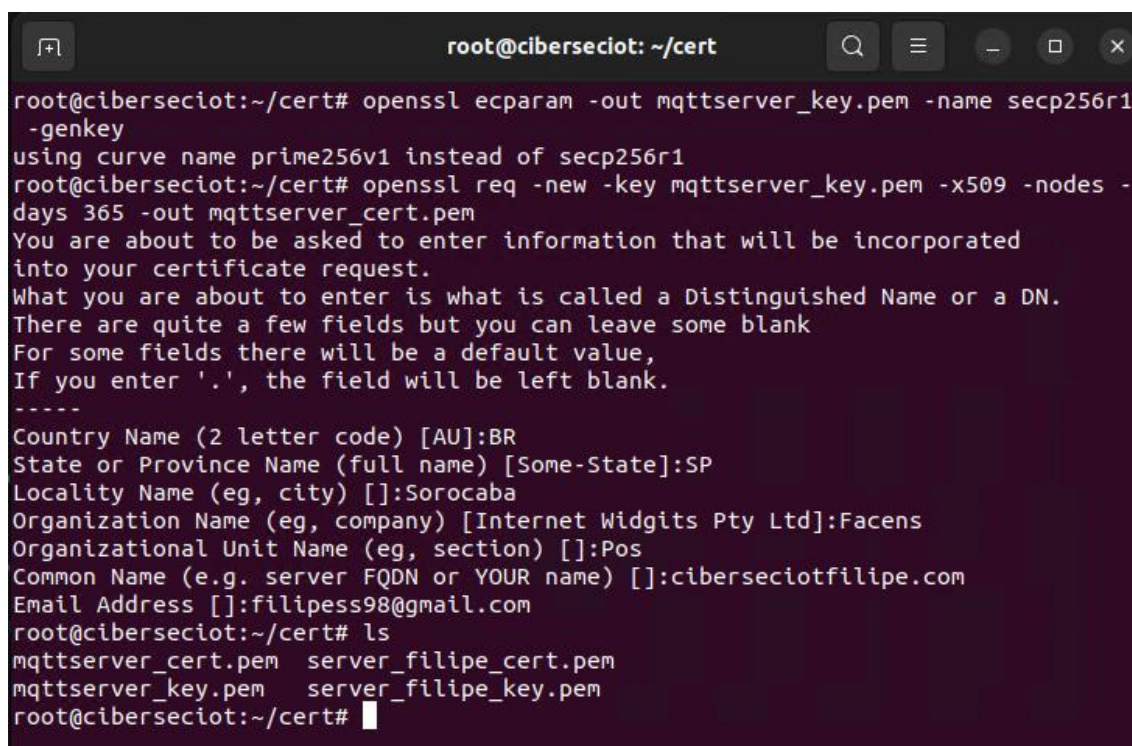
The ThingsBoard interface, titled 'ThingsBoard | Devices', shows the 'Latest telemetry' for a device named 'IoT\_Sim'. The 'Latest telemetry' table displays the following data points:

Last update time	Key	Value
2023-09-10 13:35:10	humidity	51.91
2023-09-10 13:35:10	temperature	2.52

## 2.6 Implementação do MQTT Seguro

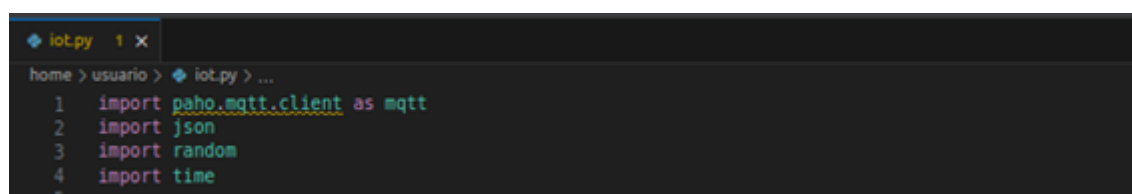
Foi realizado posteriormente uma modificação no código a fim de utilizar os certificados digitais como método de autenticação, tornando a comunicação entre cliente e servidor mais segura.

Para que essa alteração de access token para X.509 possa ser realizada, repete-se o processo de criação das chaves e certificados demonstrados no tópico 2.3, no qual os comandos “openssl ecparam -out server\_key.pem -name secp256r1 -genkey”, é utilizado para gerar um par de chaves EC, utilizadas na criptografia assimétrica para SSL/TLS, e o comando “openssl req -new -key server\_key.pem -x509 -nodes -days 365 -out server.pem”, é utilizado para realizar a criação de solicitação de certificados X.509.



```
root@ciberseciot: ~/cert
root@ciberseciot:~/cert# openssl ecparam -out mqttserver_key.pem -name secp256r1 -genkey
using curve name prime256v1 instead of secp256r1
root@ciberseciot:~/cert# openssl req -new -key mqttserver_key.pem -x509 -nodes -days 365 -out mqttserver_cert.pem
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]:SP
Locality Name (eg, city) []:Sorocaba
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Facens
Organizational Unit Name (eg, section) []:Pos
Common Name (e.g. server FQDN or YOUR name) []:ciberseciotfilipe.com
Email Address []:filipess98@gmail.com
root@ciberseciot:~/cert# ls
mqttserver_cert.pem  server_filipe_cert.pem
mqttserver_key.pem  server_filipe_key.pem
root@ciberseciot:~/cert#
```

Adicionou-se também a esse novo código a biblioteca json, para realizar a manipulação de dados no formato JSON



```
home > usuario > lot.py > ...
1 import paho.mqtt.client as mqtt
2 import json
3 import random
4 import time
5
```

Em seguida, a variável outrora denominada “access\_token” é removida e são declaradas as variáveis que serão utilizadas para realizar a configuração do MQTT, sendo estas e suas respectivas configurações:

- broker\_address = “localhost”, definindo o endereço do broker
- port = 8883, definindo a porta comumente usada para comunicações MQTT seguras através do protocolo MQTT com TLS/SSL



- cafile = “/mnt/mqttserver\_cert.pem”, que indica o caminho do certificado SSL/TLS que foi criado para realizar a autenticação e é utilizado para verificar se o servidor MQTT é confiável e emite certificados válidos
- keyfile = “/mnt/mqttserver\_key.pem”, que é a chave criptográfica criada
- certfile = “/mnt/mqttserver\_cert.pem”, que contém o certificado do cliente
- topic = “v1/devices/me/telemetry”, que indica o local/destino ao qual serão enviados os dados, ou recebidos

```

6 # Configuracoes MQTT
7 broker_address = "localhost"
8 port = 8883
9 cafile = "/mnt/mqttserver_cert.pem"
10 keyfile = "/mnt/mqttserver_key.pem"
11 certfile = "/mnt/mqttserver_cert.pem"
12 topic = "v1/devices/me/telemetry"

```

Vale ressaltar que em um ambiente de produção, o “cafile” deve ser um certificado assinado por uma Autoridade Certificadora (CA), porém como estamos utilizando um certificado autoassinado, somente para fins de testes, utilizamos o mesmo certificado que fora criado anteriormente.

São criadas então duas funções, uma para imprimir um valor aleatório da temperatura entre um valor mínimo e máximo, definidos na chamada da função, enquanto a outra é responsável por imprimir um valor aleatório de umidade, bem semelhante à função de valor de temperatura. Além disso, manteve-se a função que verifica se a conexão foi estabelecida com sucesso.

```

14 # Dados a serem enviados
15 data = {"temperature": 18}
16
17 def temperature_value(min, max):
18     random_temp = round(random.uniform(min, max), 2)
19     return {"temperature": random_temp}
20
21 def humidity_value(min, max):
22     random_humi = round(random.uniform(min, max), 2)
23     return {"humidity": random_humi}
24
25
26 # Funcao de callback para quando a conexao MQTT eh estabelecida
27 def on_connect(client, userdata, flags, rc):
28     if rc == 0:
29         print("Conexao estabelecida com sucesso")
30     else:
31         print("Erro na conexao (Codigo: " + str(rc) + ")")
32
33

```

É criada uma instância cliente do MQTT denominada “client”, e em seguida é definida a função de call-back “on\_connect”, que é chamada quando a conexão MQTT é estabelecida.

A função “cliente.username\_pw\_set(access\_token)” utilizada no tópico anterior é removida e agora é utilizada a função “client.tls\_set(ca\_cert=cafile, certfile=certfile, keyfile=keyfile)”, na qual são configuradas as opções de segurança do cliente MQTT, especificando os caminhos para os certificados e chaves que foram criados. Realiza-se então a conexão ao broker MQTT através da função “connect”, especificando o endereço do broker e a porta a ser utilizada, que agora é a porta 8883, e não mais a 1883, como no tópico anterior, pois a porta 8883 é por padrão utilizada para o MQTT Seguro.

```

33 # Cria um cliente MQTT
34 client = mqtt.Client()
35
36 # Define as funcoes de callback
37 client.on_connect = on_connect
38
39 # Configura as opcoes de TLS/SSL
40 client.tls_set(ca_certs=cafile, certfile=certfile, keyfile=keyfile)
41
42 # Conecta-se ao broker MQTT
43 client.connect(broker_address, port)
44

```

Tem-se então o loop principal, que é um loop infinito que vai gerar e publicar os dados periodicamente. A cada iteração são gerados novos valores de temperatura e umidade, os valores são combinados em um único payload JSON, que é publicado no tópico MQTT especificado, com um valor de Quality of Service de 1, que garante a entrega da mensagem ao menos uma vez. Em seguida, o programa imprime os dados publicados e aguarda 5 segundos antes de iniciar a próxima iteração.

```

45 # Loop principal
46 try:
47     while True:
48         # Gera dados de temperatura e umidade
49         temp = temperature_value(10, 30)
50         humi = humidity_value(10, 60)
51
52         # Cria um único payload JSON com temperatura e umidade
53         payload = json.dumps({"temperature": temp["temperature"], "humidity": humi["humidity"]})
54
55         # Publica os dados no tópico MQTT
56         client.publish(topic, payload, qos=1)
57         print("Dados publicados: " + payload)
58
59         time.sleep(5) # Aguarda 5 segundos antes de enviar novamente
60

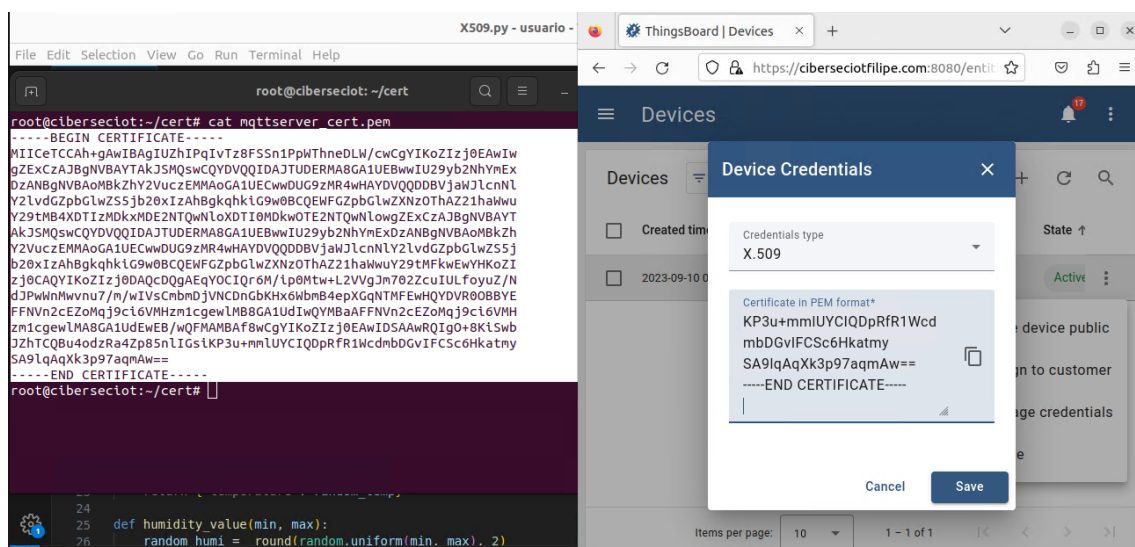
```

Após preparar o código para realizar a comunicação através do X.509, deve-se então configurar o container para utilizar as portas de maneira correta, removendo a porta 1883 a fim de impedir que sejam realizadas conexões com MQTT de forma comum, e habilitando a porta 8883 para permitir que sejam realizadas as comunicações utilizando MQTT Seguro. Deve-se também configurar as seguintes variáveis de ambiente, de forma semelhante ao que foi feito na configuração do HTTPS:

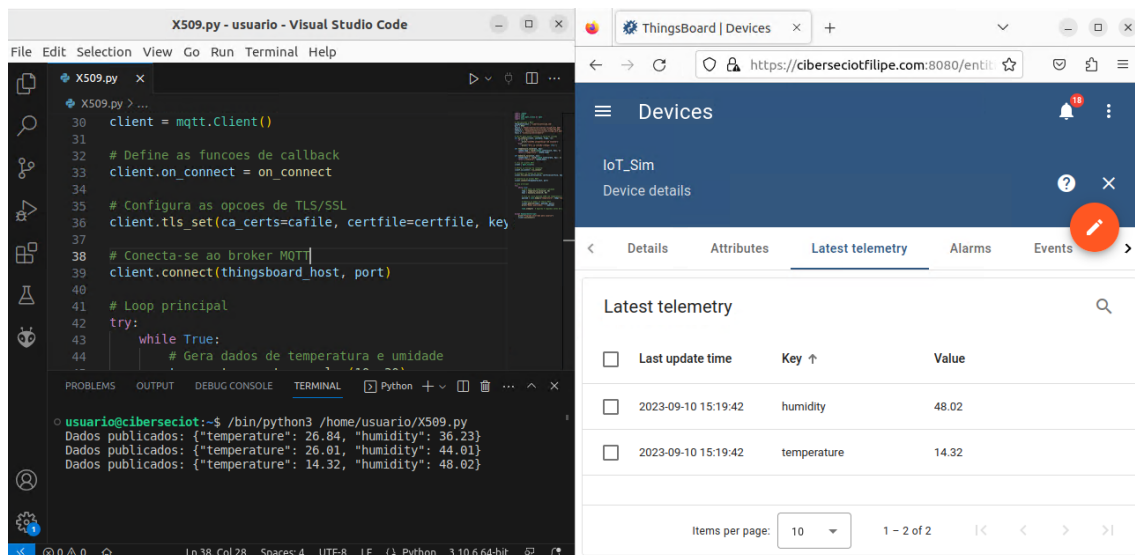
- MQTT\_SSL\_ENABLED é definida como “true”, para habilitar o MQTT over SSL
- MQTT\_SSL\_CREDENTIALS\_TYPE configura o tipo de credencial utilizada, que nesse caso é o PEM
- MQTT\_SSL\_PEM\_CERT aponta o endereço do certificado
- MQTT\_SSL\_PEM\_KEY aponta o endereço da chave
- MQTT\_SSL\_PEM\_KEY\_PASSWORD deixamos em branco por não estar utilizando uma senha nessa configuração atual

```
GNU nano 6.2
version: '3.0'
services:
  mytb:
    image: "thingsboard/tb-postgres"
    ports:
      - "8883:8883"
      - "8080:9090"
      - "7070:7070"
      - "5683-5688:5683-5688/udp"
    environment:
      TB_QUEUE_TYPE: in-memory
      SSL_ENABLED: "true"
      SSL_CREDENTIALS_TYPE: PEM
      SSL_PEM_CERT: /config/server_filipe_cert.pem
      SSL_PEM_KEY: /config/server_filipe_key.pem
      SSL_PEM_KEY_PASSWORD:
      MQTT_SSL_ENABLED: "true"
      MQTT_SSL_CREDENTIALS_TYPE: PEM
      MQTT_SSL_PEM_CERT: /config/mqttserver_cert.pem
      MQTT_SSL_PEM_KEY: /config/mqttserver_key.pem
      MQTT_SSL_PEM_KEY_PASSWORD:
    volumes:
      - /root/data:/data
      - /root/logs:/var/logs/thingsboard
      - /root/cert:/config
```

Estando o código e o container devidamente configurados, reinicia-se o container e se altera na plataforma ThingsBoard o tipo de credenciais que o dispositivo está utilizando para autenticação de “Access Token” para “X.509”, e faz-se necessário copiar o conteúdo do certificado gerado para a plataforma:



Ao executar o código python modificado para utilizar o X.509, nota-se que a plataforma ThingsBoard pôde obter os dados:

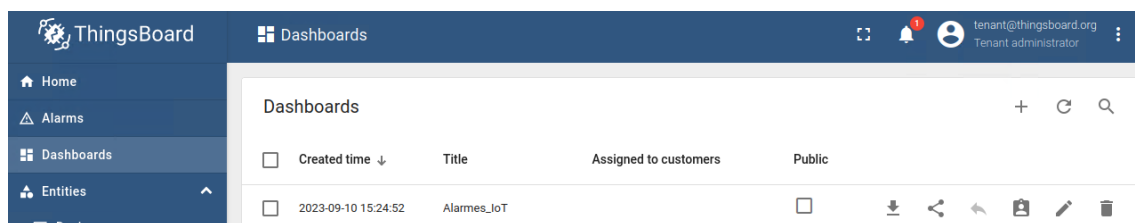


Diante destes resultados, considera-se que foi possível realizar a implementação do MQTT Seguro no simulador para que a comunicação do dispositivo simulado e plataforma ThingsBoard ocorresse de forma mais protegida, no entanto, ressalva-se que em um ambiente de produção é extremamente recomendado que se utilize um certificado assinado por uma Autoridade Certificadora (CA), e não de forma autoassinada, conforme fizemos neste experimento para demonstrar o uso destes protocolos.

## 2.7 Configuração do Dashboard do Thingsboard

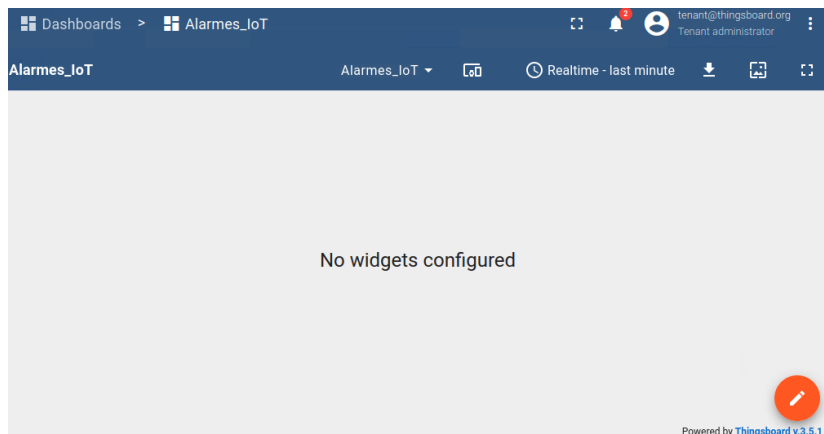
Após constatar que o dispositivo simulado está comunicando corretamente e de forma segura com a plataforma ThingsBoard, tem-se agora o desafio de melhorar a visualização destes dados e posteriormente gerar alarmes para serem disparados diante de condições específicas.

Para tornar melhor a visualização, optou-se por realizar a criação de dashboards na plataforma ThingsBoard. Tal tarefa pode ser realizada ao selecionar a opção Dashboards no menu esquerdo, e em seguida clicar no botão de adicionar, na direita da nova área, selecionando então a opção de adicionar um novo dashboard:

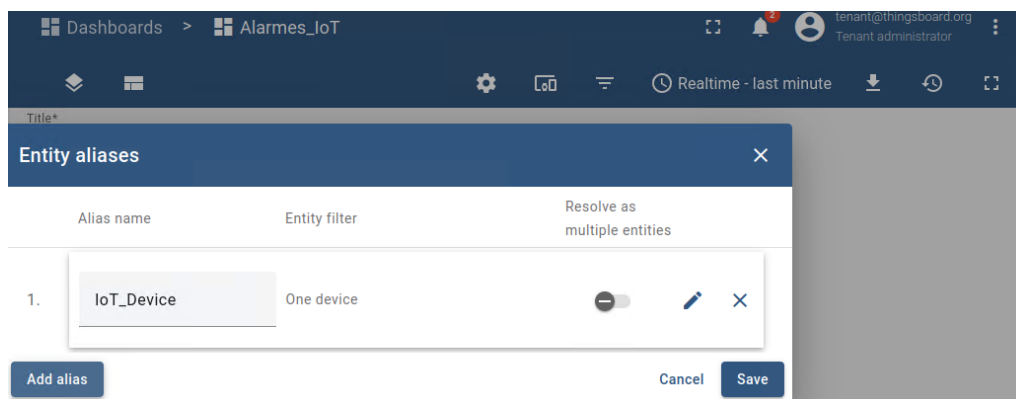


Seleciona-se o dashboard recentemente criado, "Alarmes IoT", e têm-se então a seguinte tela:

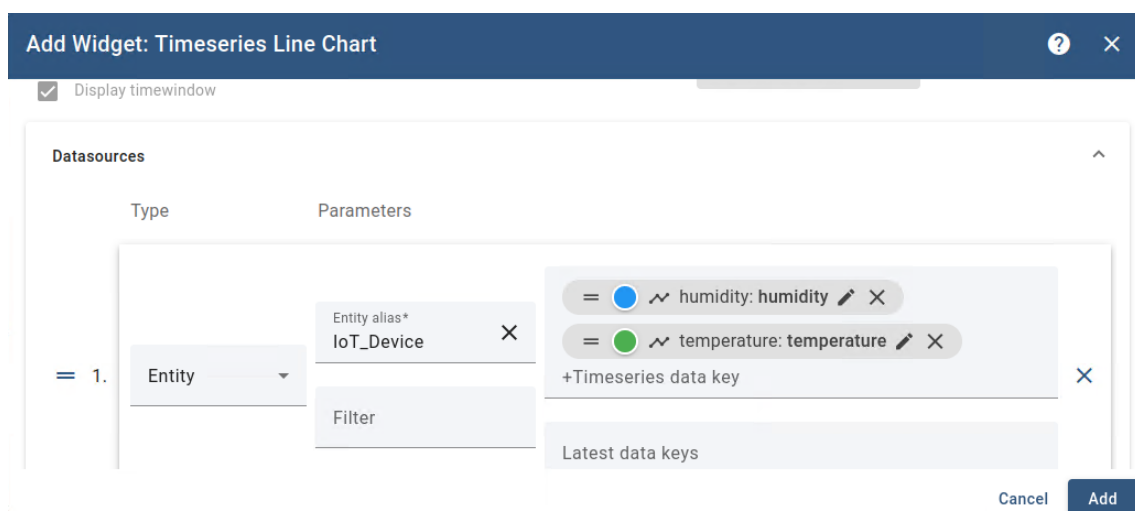




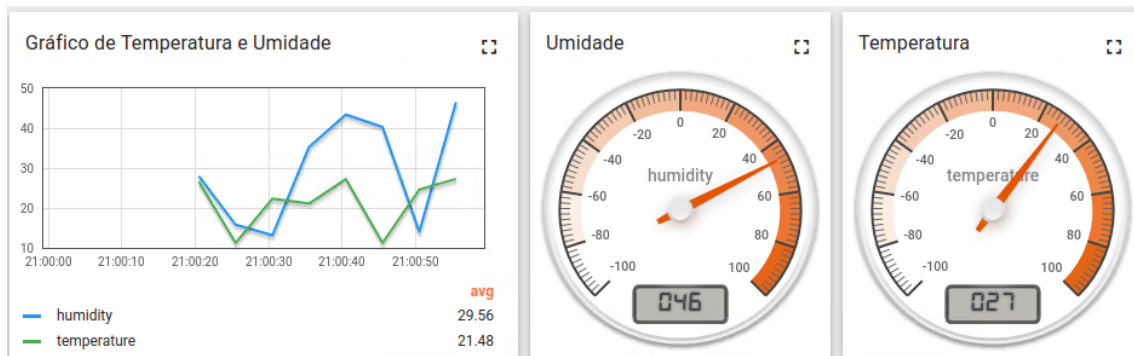
Ao clicar no ícone de lápis laranja no canto inferior direito da tela, tem-se a tela de configuração, na qual se pode selecionar a opção de “Entity Aliases”, e assim configurar o dispositivo que será conectado a este dashboard.



Tendo configurado o dispositivo, utiliza-se das opções de adicionar widgets para personalizar o dashboard da maneira que se preferir com as diversas opções disponíveis na plataforma. Quando um novo widget é selecionado, deve-se configurar o alias do dispositivo e os valores a serem exibidos.



Para melhor visualização dos dados de temperatura e umidade, utilizamos um widget de gráfico de linha do tempo, contendo as informações de leitura desses valores, bem como widgets de ponteiros analógicos que variam de acordo o valor da leitura, conforme se observa na imagem abaixo.



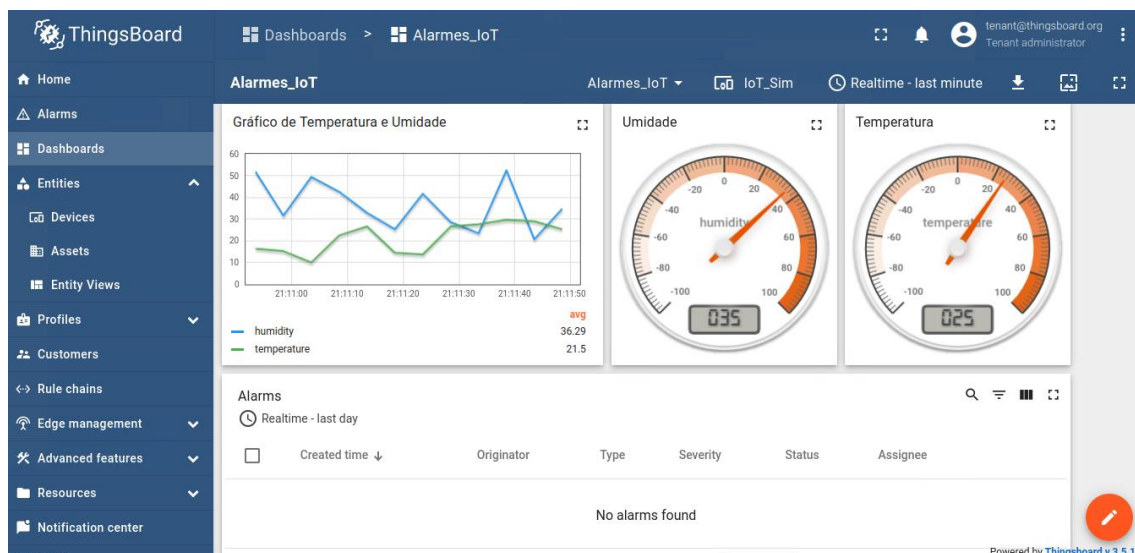
A fim de preparar o dashboard para também notificar eventos quando os valores estiverem acima, ou abaixo, da faixa esperada, adicionamos também um widget de alarmes para que estes alertas sejam exibidos no dashboard. Configurou-se a severidade e que os alertas serão de níveis “critical” e “major”.

The screenshot shows the 'Add Widget: Alarms table' configuration screen. It includes tabs for 'Data', 'Settings', 'Advanced', and 'Actions'. Under 'Settings', there are checkboxes for 'Use dashboard timewindow' (unchecked) and 'Display timewindow' (checked). The 'Timewindow' is set to 'Realtime - last day'. Below these, there are dropdown menus for 'Alarm status list' (set to 'Active') and 'Alarm severity list' (set to 'Critical, Major').

Configura-se também a “Entity Alias” para utilizar o dispositivo que criamos anteriormente e vinculá-lo a este widget.

The screenshot shows the 'Add Widget: Alarms table' configuration screen, specifically the 'Alarm source' section. It includes a dropdown menu for 'Entity' (set to 'IoT\_Device') and a 'Filter' input field. To the right, there is a list of alarm data keys with corresponding icons and labels: 'Created time: createdTime', 'Originator: originator', 'Type: type', 'Severity: severity', 'Status: status', and 'Assignee: assignee'. Each key has a plus icon and a delete icon. At the bottom right, there are 'Cancel' and 'Add' buttons.

A tela final do dashboard que criamos agora está preparada para receber os alertas que serão configurados no próximo tópico, e nela se observa a linha do tempo contendo os últimos valores obtidos, os ponteiros analógicos exibindo os diferentes valores de temperatura e umidade e por fim o widget de alertas, que ainda se apresenta vazio devido à falta das regras de alarme.



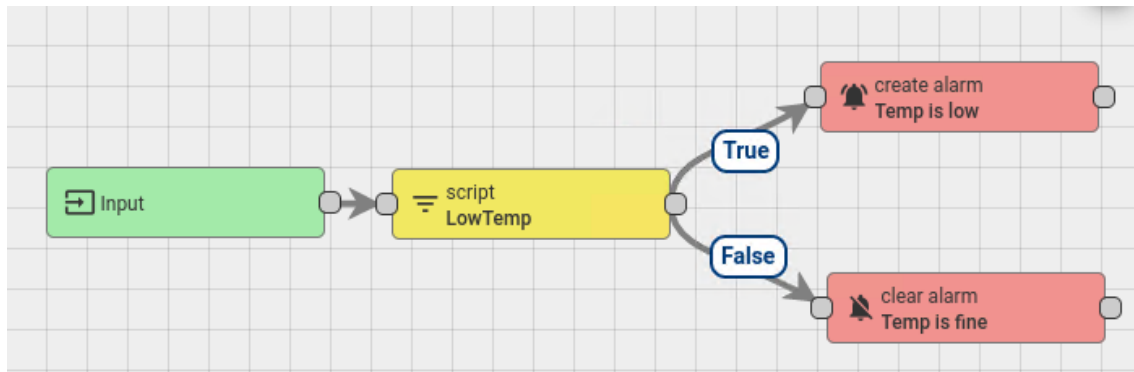
## 2.8 Criação das regras de alarme na plataforma Thingsboard

No dispositivo simulado estamos lidando com informações de temperatura e umidade, então poderíamos gerar um alarme que, por exemplo, fosse disparado quando a umidade estivesse abaixo do recomendado pela Defesa Civil, que é de 20%, alertando os usuários de uma aplicação para que estes devem ficar atentos às condições climáticas. Um outro exemplo de aplicação desse sistema pode ser dado em uma estufa, onde a umidade e a temperatura devem estar em uma faixa regulada para prevenir danos às plantas, ou cultivos.

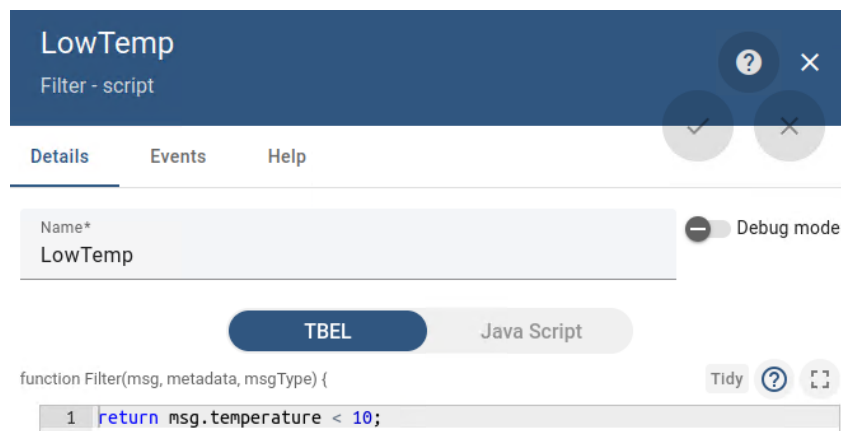
Para realizar tais configurações na plataforma ThingsBoard, seleciona-se o botão de "Rule chains" no menu esquerdo, o qual é responsável por abrir a área na qual as regras de alertas serão configuradas. Clicando no botão de adicionar, cria-se um novo conjunto de regras a serem configuradas:

Rule chains						
<input type="checkbox"/>	Created time ↓	Name	Root			
<input type="checkbox"/>	2023-09-10 21:15:00	IoT_Alerts	<input type="checkbox"/>	↓	🚩	✏️
<input type="checkbox"/>	2023-09-10 15:31:50	Root Rule Chain	<input checked="" type="checkbox"/>	↓	🚩	✏️

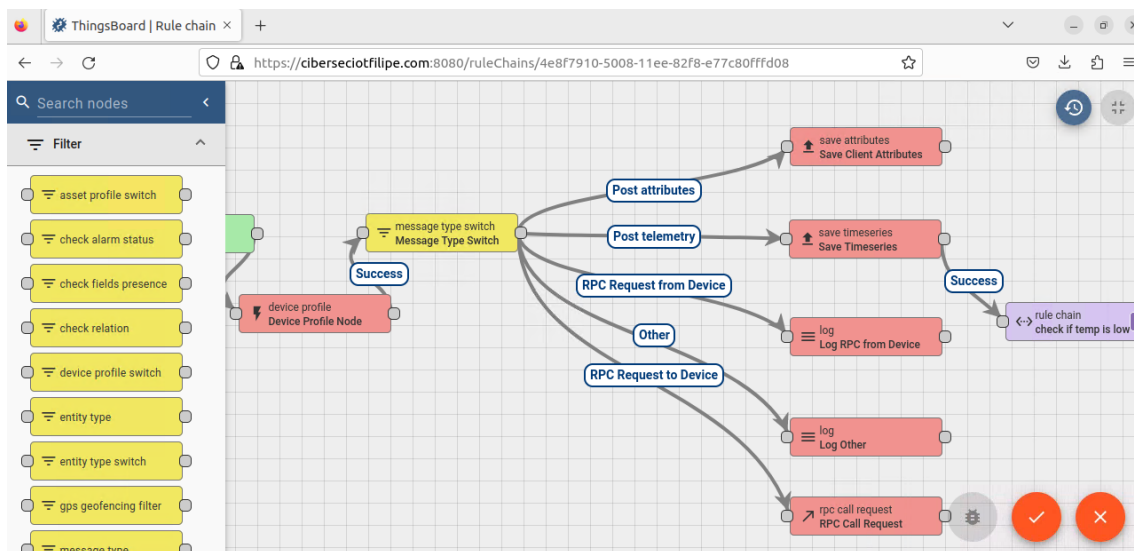
Ao selecionar a nova Rule Chain criada, "IoT\_Alerts", uma nova página é aberta, na qual diversos blocos de código são apresentados, e através dos quais as rule chains serão configuradas. Para demonstração, criou-se uma regra de alarme que cria o alerta quando a temperatura é menor que 10, e reseta o alerta quando essa condição é falsa.



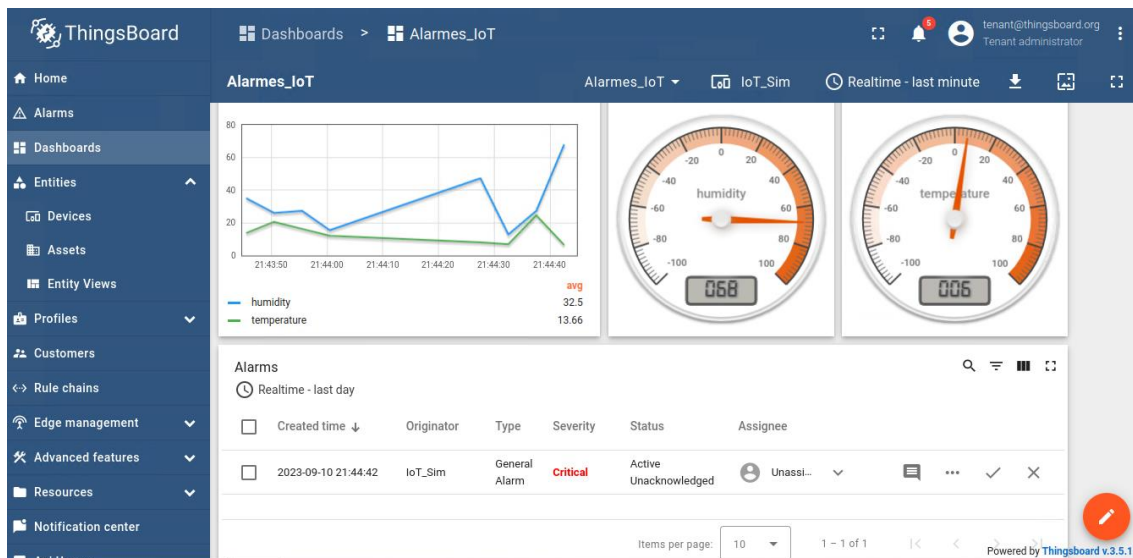
O bloco de script apresenta uma verificação se a temperatura é menor que o valor estabelecido, que neste caso é 10:



Tendo sido finalizada a criação deste alerta simples, deve-se então modificar a Root Rule Chain, incorporando este novo alerta a ela através de um bloco de “rule chain”, no qual é especificada a “IoT\_Alerts”.



Após aplicar as novas definições, pode-se observar no dashboard que ao receber um valor de temperatura menor que 10, um novo alerta foi reportado, indicando uma severidade crítica, e logo que o valor se tornou maior que 10, o alarme foi resetado, aguardando novamente um valor maior que 10 para que um novo alerta surtisse.



Pensando em um cenário de alertas da defesa civil, a mesma lógica do alarme para indicar temperaturas baixas foi replicada nos seguintes cenários:

- Um alerta é gerado quando a temperatura é abaixo de 10°C
- Um alerta é gerado quando a temperatura é acima de 35°C
- Um alerta é gerado quando a umidade é abaixo dos 50%
- Um alerta crítico é gerado quando a umidade é abaixo dos 30%

Utilizando-se os diferentes blocos de programas e aprimorando-se nos conhecimentos específicos envolvendo a plataforma ThingsBoard, pode-se criar os mais diversos tipos de notificações e alertas.

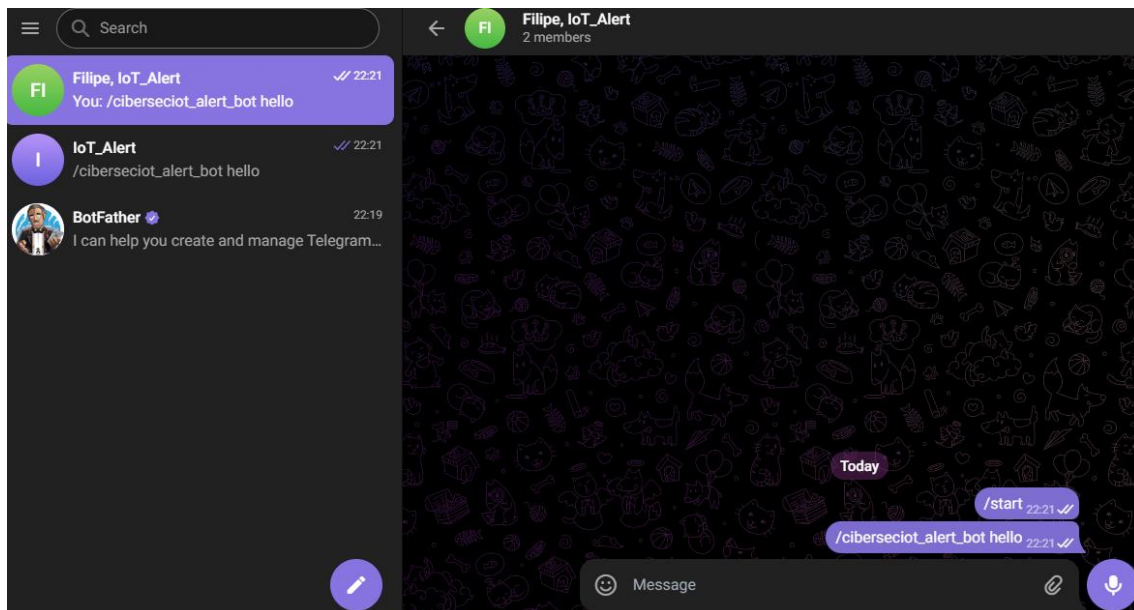
## 2.9 Notificação via Telegram

Seguindo a documentação de integrações do ThingsBoard, nota-se a possibilidade de utilizar o Telegram como método de notificação, algo muito relevante diante do atual cenário em que muitos golpistas têm fraudado e-mails e chips SIM, tornando esses métodos de autenticação cada vez menos seguros. O seguinte link apresenta toda a documentação dos passos que serão dados nesta etapa: <https://thingsboard.io/docs/user-guide/rule-engine-2-0/tutorials/integration-with-telegram-bot/>

Inicia-se este processo através da criação de um Bot do Telegram utilizando o BotFather, responsável por auxiliar na criação de novos bots. Após finalizar o processo, um token de acesso é compartilhado e deve ser armazenado para ser utilizado posteriormente.

Em seguida, deve-se enviar as seguintes mensagens para o novo bot criado, sendo uma no privado, e outra em um grupo no qual o usuário e o bot estão participando:

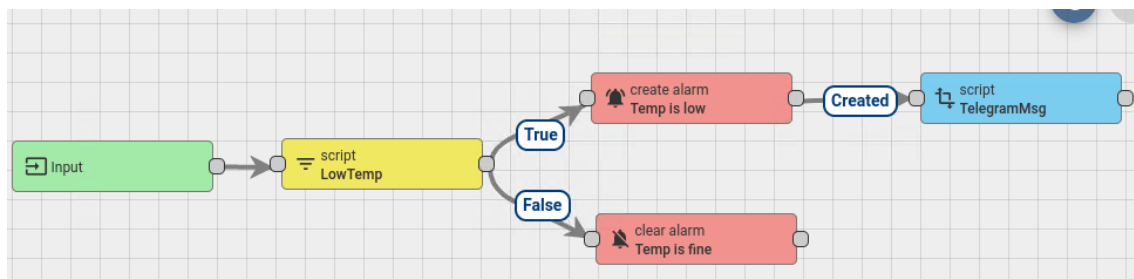
- /start
- /<nome\_do\_bot> hello



Após essas etapas, deve-se acessar uma URL específica para obter o ID das conversas: “https://api.telegram.org/bot"YOUR\_BOT\_TOKEN"/getUpdates”.

Nesta URL, o valor “YOUR\_BOT\_TOKEN” deve ser substituído pelo token obtido anteriormente, e ao acessar este endereço, deve-se procurar em meio às mensagens presentes o ID referente ao chat privado e o ID referente ao chat em grupo, para que estas informações sejam adicionadas no ThingsBoard.

Voltando ao editor de regras do ThingsBoard, optamos por modificar um dos alarmes já criados e adicionar um script de transformação ao código:



Esse script foi editado de modo a conter o ID da conversa privada com o bot, seguindo o recomendado pela documentação:





Por fim, foi adicionado ainda um Rest API Call, configurado com o token do bot recentemente criado:

Add rule node: rest api call

Name\*

Rest API CALL Telegram

Endpoint URL pattern\*

am.org/bot6288186879:AAEXKbpzqNQWYg0U5ulVEPN1ryyAWtU0IzY/sendMessage

Hint: use \${metadataKey} for value from metadata, \${messageKey} for value from message body

Request method\*

POST

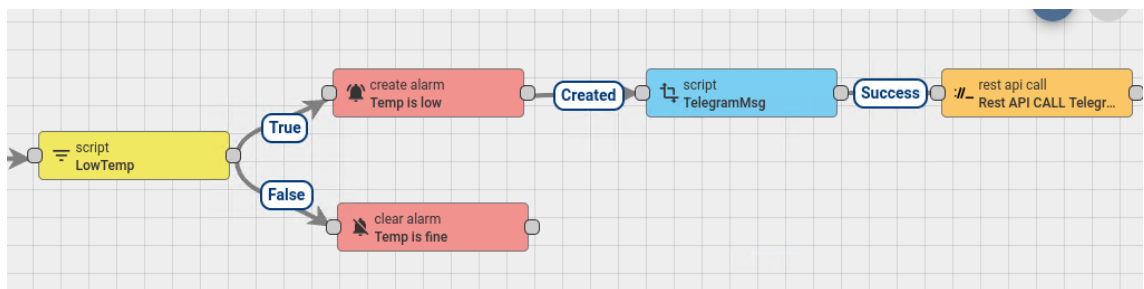
☐ Enable proxy

☐ Use simple client HTTP factory

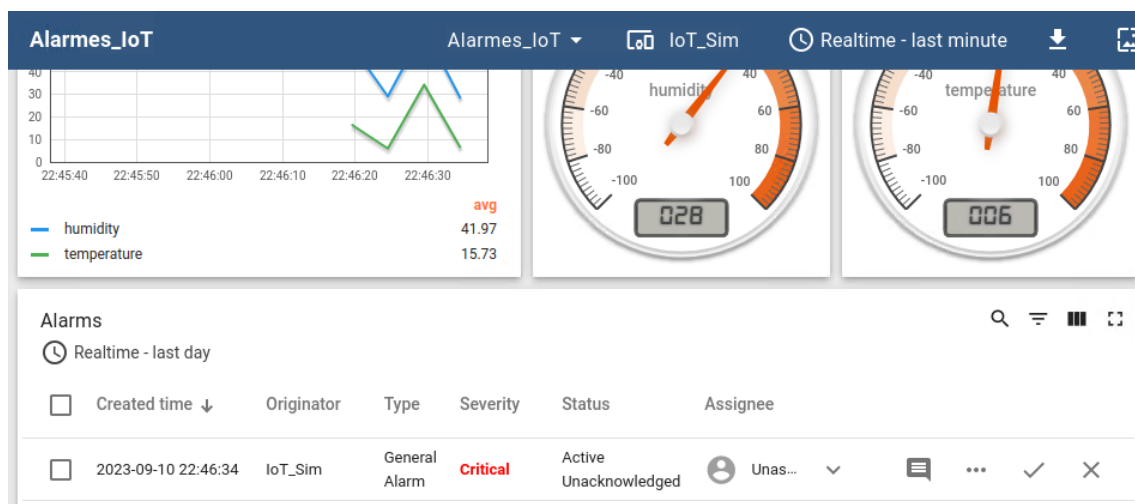
Cancel

Add

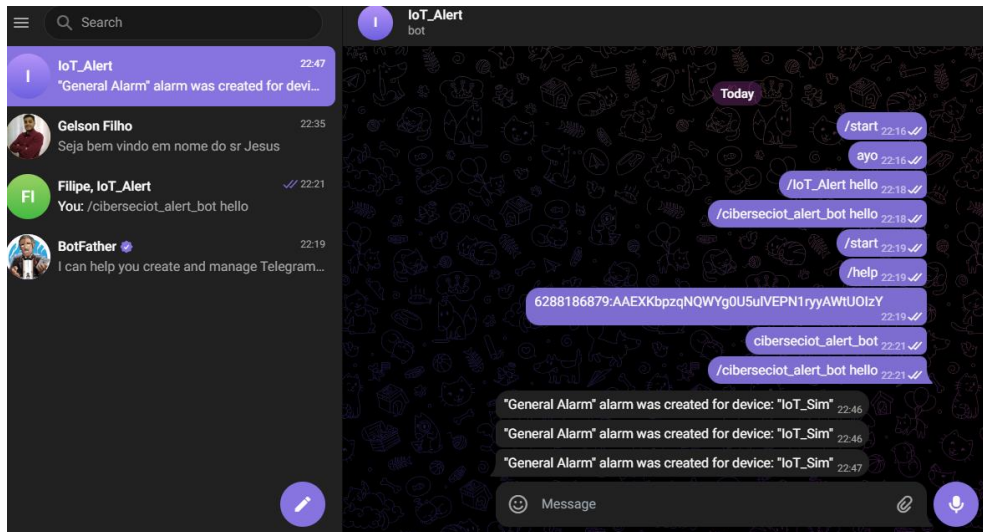
No diagrama de blocos, o Rest API Call é adicionado após o script de envio de mensagem do Telegram:



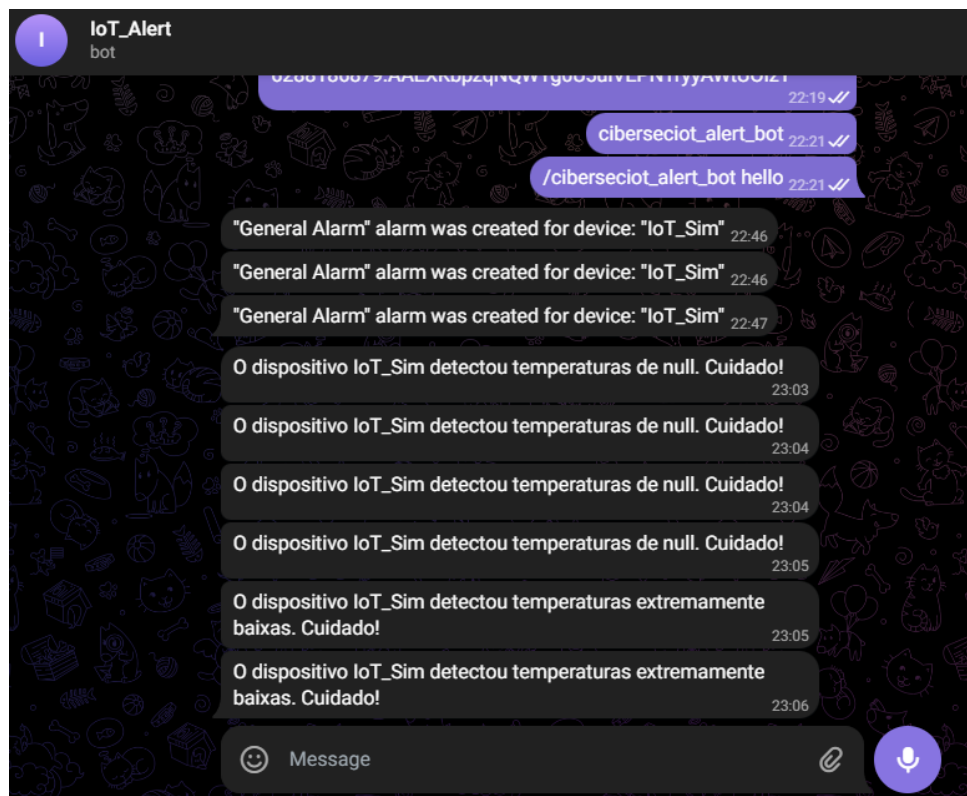
Ao aplicar essas configurações, observa-se que no momento em que o alarme foi gerado:



O Bot entrou em funcionamento e enviou mensagens indicando que os alertas haviam sido criados:



Modificamos então a mensagem a ser exibida quando o alarme fosse acionado:





## 2.10 Dispositivo ESP32 com sensor de temperatura e pressão

Tendo realizado a simulação de um dispositivo IoT através de um código em Python, optamos por efetuar então a configuração física e verificar se um dispositivo real apresentaria uma configuração muito distante do que foi simulado. Para tal procedimento, utilizamos um ESP32 e um sensor DHT11, que realiza a medição de dados de temperatura e umidade.

O maior desafio identificado durante o desenvolvimento dessa atividade se deu pelo fato de que a biblioteca que estava sendo utilizada pelo PlatformIO, Wi-Fi Client Secure, responsável por autorizar o uso de certificados para autenticação estava apresentando problemas na função “connect” em setembro de 2023, sendo necessário realizar a adição dessa biblioteca de forma manual no código. Tal desafio é mais detalhado na apresentação do vídeo disponível no tópico 2.11 deste documento.

Através da plataforma PlatformIO, foi desenvolvido o código para ser utilizado no ESP32. A seguir o código desenvolvido será analisado em partes:

- Inicialmente se tem a inclusão das bibliotecas utilizadas no programa:

```
#include <Arduino.h>
#include <PubSubClient.h>
#include "dependencies/WiFiClientSecure/WiFiClientSecure.h"
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

- São então declarados os certificados para autenticação do MQTT:

```
const char* CA_cert = \
"-----BEGIN CERTIFICATE-----\n" \
"MIIBFTCCASQgAwIBAgIUBFKsPeVgNGYpv+1T3mh6ZC6c1KowCgYIKoZIzj0EAwIw\n" \
"FDESMBAGA1UEAwJbG9jYXRob3N0MB4XDTEzMDkwODIyMjc5MjFoXDTI0MDkwNzIy\n" \
"Mjc5MjFoXDESMBAGA1UEAwJbG9jYXRob3N0MFkwEwYHKoZIzj0CAQYIKoZIzj0D\n" \
"AQcDQgAEK2bFZc/6k8xazr+8gkkC83/Alwy/8sIdAbJGnv6qAfd9A/G27nLuEBFu/\n" \
"w3wLwYhrz1ZTnkWvPCXVjz2R3xryaKNTMFEwHQYDVR0OBBYEFAl+kF3rS3KjE4VU\n" \
"Xo2Eug05/mZgMB8GA1UdIwQYMBAAFAI+kF3rS3KjE4VUXo2Eug05/mZgMA8GA1Ud\n" \
"EwEB/wQFMAMBAf8wCgYIKoZIzj0EAwIDSAAwRQIgV9cUAib2a+fgbkf15VxUqCLE\n" \
"uRHSkxd4RKho5TVUjQICIQ45BMnDVja15IvO6/IUzq6qj+a6Vh+xpweHRZITMaa\n" \
"Hg==\n" \
"-----END CERTIFICATE-----\n";

const char* ESP_CA_cert = \
"-----BEGIN CERTIFICATE-----\n" \
"MIIBFTCCASQgAwIBAgIUBFKsPeVgNGYpv+1T3mh6ZC6c1KowCgYIKoZIzj0EAwIw\n" \
"FDESMBAGA1UEAwJbG9jYXRob3N0MB4XDTEzMDkwODIyMjc5MjFoXDTI0MDkwNzIy\n" \
"Mjc5MjFoXDESMBAGA1UEAwJbG9jYXRob3N0MFkwEwYHKoZIzj0CAQYIKoZIzj0D\n" \
"AQcDQgAEK2bFZc/6k8xazr+8gkkC83/Alwy/8sIdAbJGnv6qAfd9A/G27nLuEBFu/\n" \
"w3wLwYhrz1ZTnkWvPCXVjz2R3xryaKNTMFEwHQYDVR0OBBYEFAl+kF3rS3KjE4VU\n" \
"Xo2Eug05/mZgMB8GA1UdIwQYMBAAFAI+kF3rS3KjE4VUXo2Eug05/mZgMA8GA1Ud\n" \
"EwEB/wQFMAMBAf8wCgYIKoZIzj0EAwIDSAAwRQIgV9cUAib2a+fgbkf15VxUqCLE\n" \
"uRHSkxd4RKho5TVUjQICIQ45BMnDVja15IvO6/IUzq6qj+a6Vh+xpweHRZITMaa\n" \
"Hg==\n" \
"-----END CERTIFICATE-----\n";

const char* ESP_RSA_key= \
"-----BEGIN EC PARAMETERS-----\n" \
"BggqhkJOPQMBBw==\n" \
"-----END EC PARAMETERS-----\n" \
"-----BEGIN EC PRIVATE KEY-----\n" \
"MHcCAQEEIEX/OM+QNsntZGM1N2Y1V137BYQ23dQxLEVmp7svm5proAoGCCqGSI4\n" \
"AwEHoUQDQgAEK2bFZc/6k8xazr+8gkkC83/Alwy/8sIdAbJGnv6qAfd9A/G27nLuE\n" \
"Bfu/w3wLwYhrz1ZTnkWvPCXVjz2R3xryaA==\n" \
"-----END EC PRIVATE KEY-----\n";
```

- São configuradas as redes Wi-Fi e o MQTT:

```
const char* ssid      = "JAO FILHO";
const char* password  = "Mud@r$321";
const char* mqtt_server = "192.168.0.9";
int port             = 8883;
const char* mqtt_user  = "";
const char* mqtt_pass  = "";
```

- Cria-se um objeto `WiFiClientSecure` denominado “cliente” que será utilizado para estabelecer a conexão Wi-Fi segura, bem como será criado um objeto “`PubSubClient`”, que utilizará a conexão para comunicação segura do MQTT:

```
WiFiClientSecure client;
PubSubClient mqtt_client(client);
```

- Configura-se o sensor DHT11:

```
#define DHT_SENSOR_PIN 10
#define DHT_SENSOR_TYPE DHT11
DHT dht(DHT_SENSOR_PIN, DHT_SENSOR_TYPE);
```

São por fim criadas as três principais funções do código, que estarão disponíveis no código completo, que estará em anexo ao final deste documento. Estas funções são:

- “`get_send_temp_humid_data()`”, responsável por realizar a leitura dos valores de temperatura e umidade do sensor DHT11 e publicá-los no tópico do MQTT no formato Json;
- “`setup()`”, na qual a conexão Wi-Fi é estabelecida, e os certificados e chaves são configurados, junto às definições do servidor MQTT e aqui também é inicializado o sensor DHT11;
- “`loop()`”, executado continuamente, é responsável por conectar-se ao servidor MQTT e então realizar a chamada da função “`get_send_temp_humid_data()`”.

## 2.11 Vídeo do funcionamento da simulação e do dispositivo ESP32

O vídeo que contempla o correto funcionamento dos protocolos HTTPS, MQTT seguro, Dispositivo Simulado, Dispositivo IoT Real, Alarmes e Notificações via Telegram pode ser acessado através do seguinte link:

<https://youtu.be/cTR7JH-ZSoo?si=W9EeECGy4qxmCb-1>

## ANEXO A – Código utilizado na comunicação do ESP32

```
#include <Arduino.h>
#include <PubSubClient.h>
#include "dependencies/WiFiClientSecure/WiFiClientSecure.h"
#include <Adafruit_Sensor.h>
#include <DHT.h>

const char* CA_cert = \
    "-----BEGIN CERTIFICATE-----\n" \
    "MIIBftCCAS0gAwIBAgIUBFKsPeVgNGYpv+1T3mh6ZC6clKowCgYIKoZIZj0EAWIw\n" \
    \
    "FDESMBAGA1UEAwJbG9jYWxob3N0MB4XDTIzMDkwODIyMjcYMFoXDTI0MDkwNzIy\n" \
    \
    "MjcYMFowFDESMBAGA1UEAwJbG9jYWxob3N0MFkwEwYHKoZIzj0CAQYIKoZIzj0D\n" \
    \
    "AQcDQgAEK2bFZc/6kBxazr+8gkkC83/AWy/8sIdAbJGnv6qAfd9A/G27nLuEBfu/\n" \
    \
    "w3wLwYhrz1ZTnkWvPCXVjz2R3xryaKNTMFewHQYDVR00BBYEFAI+kF3rS3KjE4VU\n" \
    \
    "Xo2Eug05/mZgMB8GA1UdIwQYMBaAFAI+kF3rS3KjE4VUXo2Eug05/mZgMA8GA1Ud\n" \
    \
    "EwEB/wQFMAMBAf8wCgYIKoZIzj0EAWIDSAAwRQIgV9cUAib2a+fgbkf15VxUqCLE\n" \
    \
    "uRHskxd4RKho5TVUjqICIQc45BMnDVja1SIv06/IUzq6qj+a6Vh+xPweHRZITMaa\n" \
    \
    "Hg==\n" \
    "-----END CERTIFICATE-----\n";

const char* ESP_CA_cert = \
    "-----BEGIN CERTIFICATE-----\n" \
    "MIIBftCCAS0gAwIBAgIUBFKsPeVgNGYpv+1T3mh6ZC6clKowCgYIKoZIZj0EAWIw\n" \
    \
    "FDESMBAGA1UEAwJbG9jYWxob3N0MB4XDTIzMDkwODIyMjcYMFoXDTI0MDkwNzIy\n" \
    \
    "MjcYMFowFDESMBAGA1UEAwJbG9jYWxob3N0MFkwEwYHKoZIzj0CAQYIKoZIzj0D\n" \
    \
    "AQcDQgAEK2bFZc/6kBxazr+8gkkC83/AWy/8sIdAbJGnv6qAfd9A/G27nLuEBfu/\n" \
    \
    "w3wLwYhrz1ZTnkWvPCXVjz2R3xryaKNTMFewHQYDVR00BBYEFAI+kF3rS3KjE4VU\n" \
    \
    "Xo2Eug05/mZgMB8GA1UdIwQYMBaAFAI+kF3rS3KjE4VUXo2Eug05/mZgMA8GA1Ud\n" \
    \
    "EwEB/wQFMAMBAf8wCgYIKoZIzj0EAWIDSAAwRQIgV9cUAib2a+fgbkf15VxUqCLE\n" \
    \
    "uRHskxd4RKho5TVUjqICIQc45BMnDVja1SIv06/IUzq6qj+a6Vh+xPweHRZITMaa\n" \
    \
    "Hg==\n" \
    "-----END CERTIFICATE-----\n";
```

```

const char* ESP_RSA_key= \
    "-----BEGIN EC PARAMETERS-----\n" \
    "BgqhkjOPQMBBw==\n" \
    "-----END EC PARAMETERS-----\n" \
    "-----BEGIN EC PRIVATE KEY-----\n" \
    "MHcCAQEEIEX/OW+QNsTGM1N2Y1V137BYQ23dQxLEVMP7svm5proAoGCCqGSM49\n"
\
    "AwEHoUQDQgAEK2bFZc/6kBxazr+8gkkC83/AWy/8sIdAbJGnv6qAfd9A/G27nLuE\n"
\
    "Bfu/w3wLwYhrz1ZTnkWvPCXVjz2R3xryaA==\n" \
    "-----END EC PRIVATE KEY-----\n";

const char* ssid      = "JAO FILHO";
const char* password  = "Mud@r$321";
const char* mqtt_server = "192.168.0.9";
int port              = 8883;
const char* mqtt_user  = "";
const char* mqtt_pass  = "";

WiFiClientSecure client;
PubSubClient mqtt_client(client);

#define DHT_SENSOR_PIN 10
#define DHT_SENSOR_TYPE DHT11
DHT dht(DHT_SENSOR_PIN, DHT_SENSOR_TYPE);

void get_send_temp_humid_data()
{
    Serial.println("Adquirindo valores.");

    float humi = dht.readHumidity();
    float temp = dht.readTemperature();

    if(isnan(humi) || isnan(temp))
    {
        Serial.println("Falha na leitura do sensor!");
        return;
    }

    String temperature = String(temp);
    String humidity = String(humi);

    String payload = "{";
    payload += "\"temperature\":";
    payload += temperature;
    payload += ",";
    payload += "\"humidity\":";

```

```

    payload += humidity;
    payload += "}";

    char attributes[100];
    payload.toCharArray(attributes, 100);
    mqtt_client.publish("v1/devices/me/telemetry", attributes);
    Serial.println(attributes);
}

void setup() {
    Serial.begin(115200);
    delay(100);

    dht.begin();
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);

    // attempt to connect to Wifi network:
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        // wait 1 second for re-trying
        delay(1000);
    }

    Serial.print("Connected to ");
    Serial.println(ssid);

    //Set up the certificates and keys
    client.setCACert(CA_cert);           //Root CA certificate
    client.setCertificate(ESP_CA_cert);
    client.setPrivateKey(ESP_RSA_key);

    mqtt_client.setServer(mqtt_server, port);

    Serial.println("Saiu do void setup");
}

void loop() {
    Serial.println("\nStarting connection to server...");
    if (mqtt_client.connect("ESP32")) {
        Serial.print("Connected, mqtt_client state: ");
        Serial.println(mqtt_client.state());
        get_send_temp_humid_data();
    }
    else {
        Serial.println("Connected failed! mqtt_client state:");
        Serial.print(mqtt_client.state());
    }
}

```

```
Serial.println("WiFiClientSecure client state:");  
char lastError[100];  
Serial.print(lastError);  
}  
delay(10000);  
}
```