IoT - NodeRed

Felipe Schreiber Fernandes DRE: 116206990

November 2020

1 Introdução

Dispositivos IoT são caracterizados por dispositivos eletrônicos de baixa capacidade de processamento e de memória, além de limitações energéticas. Ao mesmo tempo, vemos que cada vez mais esses dispositivos estão presentes no nosso dia a dia, gerando uma quantidade enorme de dados. Essa característica nos impõe uma barreira, como armazenar/gerenciar tantos dados? Uma das soluções é colocar esses dados na nuvem, ao invés de armazenar indefinidamente no dispositivo. Assim, o usuário não precisaria fazer requisições diretamente ao dispositivo, o que comprometeria uma parte significativa da sua, já reduzida, bateria apenas para a troca de mensagens. Vale ressaltar também que a redução do custo de memória é um dos fatores que favoreceu a migração de dados pra nuvem.

Nesse trabalho então buscou-se desenvolver uma aplicação simples com essa ideia em mente. Diversas plataformas poderiam ser utilizadas, mas optamos pelo Node-Red dada sua maior popularidade. O objetivo é construir no framework Node-Red um servidor que

Figure 1: Grafico de popularidade de dois frameworks: AWS IoT e Node-Red.

repassasse os dados coletados de sensores para um cliente conectado pelo Telegram.

2 Projeto

O objetivo era "Criar um sistema que meça a temperatura do seu quarto e te avise pelo telegram ou twitter". Devido a falta de sensores de temperatura, utilizou-se os sensores de temperatura da CPU do computador para demonstrar a viabilidade do projeto. Assim, ao invés de medir a temperatura do quarto, é a temperatura da CPU que será enviada a um usuário do Telegram. Isso é útil por exemplo em tarefas que demandam muito processamento, que levam horas, e muitas vezes a pessoa em questão nem sempre está com o computador por perto.

2.1 Decisões do Projeto

- Primeiramente optou-se por utilizar o Telegram como plataforma de mensagem por dois motivos: ser mais fácil de usar e não precisar esperar a aprovação da conta de desenvolvedor do Twitter.

- Em segundo lugar, o protocolo de comunicação adotado foi o TCP e a qualidade de serviço colocada como 2, o que significa que o dispositivo IoT irá aguardar uma mensagem do tipo ACK do servidor para confirmar a chegada dos dados. Embora o uso de UDP diminua bastante o overhead da comunicação, e por conseguinte o consumo de energia/memória, a opção pelo TCP foi principalmente pelo caráter da aplicação: Detectar aumento de temperatura no equipamento. Assim, caso se optasse pelo UDP, no cenário em que o servidor perde um pacote e até o envio do próximo ocorre uma falha, o dano pode ser irreparável.
- Nenhum dado é guardado no dispositivo de sensoreamento.
- Qualquer um que tiver acesso a chave de acesso do bot poderá se comunicar com o mesmo.
- O protocolo de comunicação será o MQTT, e para tal usou-se o pacote node-red-contrib-aedes.

2.2 Criação do Bot

Para criar um bot para o Telegram, primeiro escolheu-se um nome e nome de usuário, que são respectivamente Cpu_Monitor e Cpu_Monitor_1998_bot. Essa é uma abstração criada pelo BotFather, gerenciador dos bots do Telegram, e que possibilitará a identificação/comunicação, uma vez que esse nome é único. Para comunicar com o recém criado

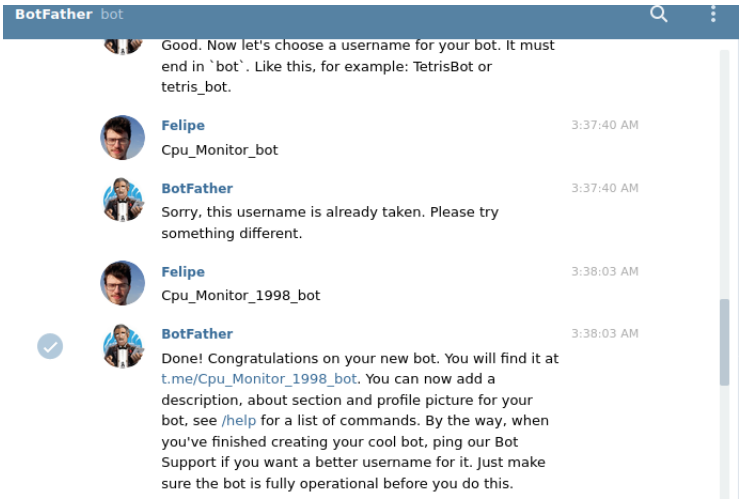

Figure 2: Escolha do nome do bot: Não completa até que ele seja válido (único)

bot do Telegram utilizou-se o pacote node-red-contrib-chatbot.

2.3 Fluxo de Mensagens

Conforme pode-se ver na figura abaixo, tem-se basicamente dois fluxos principais: Um que atende a requisição do usuário usando o Telegram, mais acima, e outro que atende o dispositivo IoT. O fluxo do meio é responsável por definir a porta de comunicação do MQTT, que nesse caso será 1883, e receber mensagem de debug quando começa a comunicação. O protocolo do MQTT possui três mensagens principais:

- Connect- Define qual o endereço Ip e porta para estabelecer conexão.
- Publish- Define o tópico e o payload (conteúdo) que será adcionado.

- Subscribe- Uma vez definido o tópico de interesse, o cliente recebe uma mensagem sempre que um novo conteúdo chega.

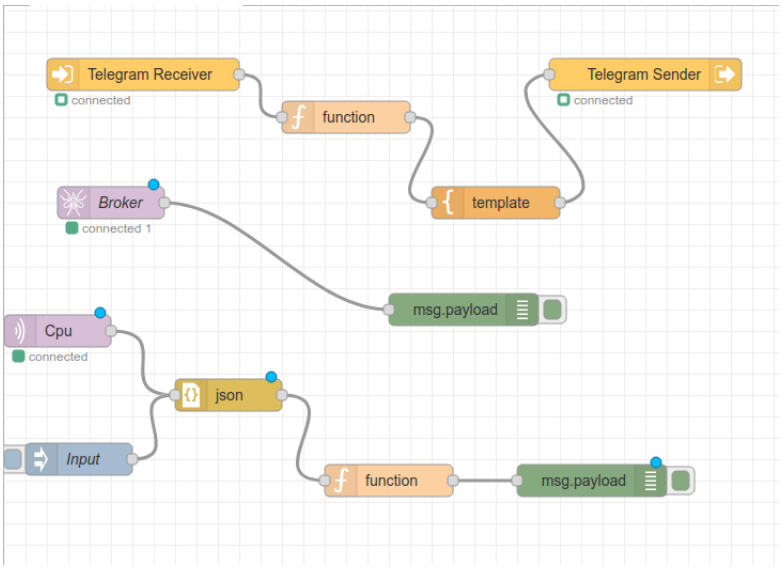

Figure 3: Abstração do fluxo de mensagens em blocos

2.3.1 Fluxo do Publish

Conforme pode ver da figura acima, teremos apenas a parte do publish implementada (bloco roxo encrito Cpu). Essa é a comunicação do cliente com o servidor, que recebe os dados. Esse bloco é uma abstração da interface de comunicação com o meio externo. Portanto precisa-se definir duas coisas:

1. O tipo de tópico que ficará aguardando. Como pode haver mais de um tópico de interesse, deverá ser colocada uma interface para cada uma.

2. O número do Ip e porta do servidor para o qual a mensagem será enviada. Nesse caso será (localhost)127.0.0.1:1883.

Para que o cliente se conecte com o servidor é necessário que ele envie as mensagens respeitando o protocolo do MQTT. Por isso utilizou-se a biblioteca paho, mais precisamente paho.mqtt.client para efetuar essa comunicação.

Uma vez que o dispositivo IoT está conectado, espera-se que ele envie os dados no formato JSON, conforme ilustrado abaixo: O cliente envia a temperatura de dois cores e além

```
"{"cpu0": {"Temperature": 68.0, "Time": 1605473963885},
"cpu1": {"Temperature": 68.0, "Time": 1605473905743}}"
```

Figure 4: Formato da Mensagem

disso o timestamp em milissegundos para o servidor. Portanto o bloco a seguir, "json", fica responsável por garantir que as mensagens estejam no formato especificado. No bloco seguinte implementou-se uma função que pega os dados no formato JSON, efetua um parse, e salva, com a chamada "flow.set()", o conteúdo na variável 'cpuX_str' que pode ser vista por qualquer outro nó dentro do mesmo flow. Por fim, ele repassa a mensagem já parseada para o bloco de debug.

```
var cpu0_str=flow.get('cpu0_str') || '';
var cpu1_str=flow.get('cpu1_str') || '';
parsed = JSON.parse(msg.payload);
cpu0_str = "Cpu0 temperature is "+parsed.cpu0.Temperature+" °C\n";
cpu1_str = "Cpu1 temperature is "+parsed.cpu1.Temperature+" °C\n";
flow.set('cpu0_str',cpu0_str);
flow.set('cpu1_str',cpu1_str);
msg.payload = parsed;
return msg;
```

Figure 5: Implementação do bloco Function no fluxo de publish

2.3.2 Fluxo do Telegram

O primeiro bloco desse fluxo busca atender sempre que o cliente envia uma mensagem ao bot do Telegram, enquanto o último bloco envia para o cliente uma resposta de acordo com o input recebido. Para isso é necessário que ambos os blocos estejam conectados ao mesmo bot e isso é feito passando o nome de usuário do bot bem como o token gerado pelo BotFather durante sua criação.

Em seguida criamos outra função que recebe a mensagem do usuário no Telegram. Ela compara se a ação desejada, ou seja aquilo que o usuário digitou, corresponde a "monitor". Se sim então lê o conteúdo armazenado na variável global a partir da chamada "flow.get()" e passa esse valor para o bot do Telegram dentro do campo "payload.content" da mensagem.

```
recv = msg.payload.content;
check = "monitor"
if (recv == check)
{
    var cpu0_str = flow.get('cpu0_str')
    var cpu1_str = flow.get('cpu1_str')
    msg.payload.content = cpu0_str+cpu1_str;
    return msg;
}
```

Figure 6: Função que checa o input do usuário e responde com a temperatura das cpu's

2.4 Dispositivo IoT

No lado do cliente vamos usar biblioteca psutils para pegar a temperatura de cada core e salvar num dicionário que irá conter, também, a hora atual em milisegundos. Importante que poderíamos monitorar outras características do sistema. Em seguida criamos um objeto do tipo client que vai fazer a comunicação com o servidor. Para isso definimos um identificador para o cliente tal como o endereço de ip/porta de comunicação. Logo após isso efetua-se a inscrição no tópico "Cpu" e dentro do loop chamamos a função "psutils.sensor_temperatures", que retorna a temperatura de alguns componentes, dentre eles os da cpu. Também pegamos a hora atual, salva no dicionário e com o método "json.dumps()" converte-se o dicionário para o formato JSON. Com isso a mensagem está pronta para ser enviada e isso é efeito com a função ".publish()", passando o tópico e a mensagem em si. Após isso o cliente espera 3 segundos até enviar a próxima mensagem.

```python
import paho.mqtt.client as paho
import psutil as ps
import time
import json

broker="127.0.0.1"
port=1883
data = {'cpu0':{},'cpu1':{}}

def on_publish(client,userdata,result):             #create function for callback
    print("data published \n")
    pass
def on_disconnect(client, userdata, rc):
   print("client disconnected ok")

print(json.dumps(data))
client1= paho.Client("control1")                          #create client object
client1.on_disconnect = on_disconnect
client1.on_publish = on_publish                          #assign function to callback
client1.connect(broker,port)                                 #establish connection
client1.subscribe("Cpu")
while True:
        result = ps.sensors_temperatures()
        millis = int(round(time.time() * 1000))
        data['cpu0']['Temperature'] = result['coretemp'][1][1]
        data['cpu1']['Temperature'] = result['coretemp'][2][1]
        data['cpu0']['Time'] = millis
        data['cpu1']['Time'] = millis
        ret= client1.publish("Cpu",json.dumps(data))
        time.sleep(3)
client1.disconnect()
```

Figure 7: Cliente/Dispositivo Iot

3 Conclusão

Conseguimos atender aos requesitos do trabalho informando a temperatura da Cpu para um usuário do Telegram que se conecta com o bot, mediante o envio da mensagem "monitor". Abaixo tem-se um exemplo da comunicação estabelecida.

4 Referências

ShirleyBasílio(2019)"ChatBotcomNode-REDeTelegram",https://blogmasterwalkershop.com.br/outros/chatbot-com-node-red-e-telegram/

http://www.steves-internet-guide.com/into-mqtt-python-client/

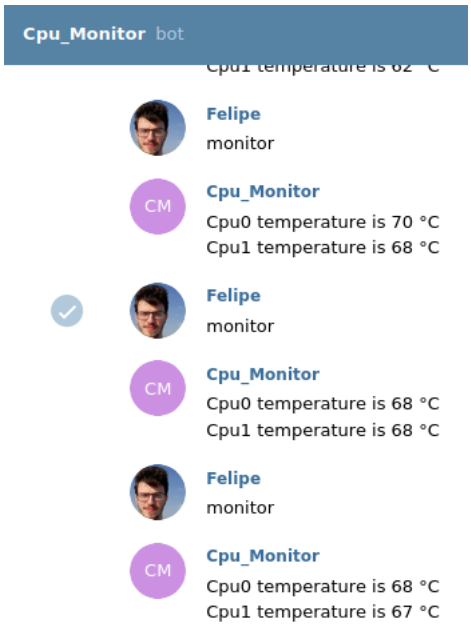

Figure 8: Típica troca de mensagem entre o usuário e o bot