

# Projeto IoT

## Microlocalização utilizando Bluetooth Low Energy

João Pedro Brandão e Silva

1

jp.brs@poli.ufrj.br

**Resumo.** Este meta-artigo tem por objetivo descrever o funcionamento e o processo de criação de uma aplicação IoT cujo objetivo é utilizar o Bluetooth Low Energy para serviços de microlocalização dentro de um ambiente fabril.

### 1. Funcionalidade

A aplicação IoT foi criada com o foco de utilizar microlocalização para otimizar processos de uma fábrica, escritório ou empresa de maneira geral. Essa aplicação funcionaria baseando-se no fato de que sabendo a localização em tempo real de cada funcionário em uma empresa e tendo esses dados salvos em um banco de dados, pode-se através de diversos estudos detectar improdutividade, desperdício de tempo, dentre outras barreiras que impedem a otimização dos processos envolvidos dessa empresa.

### 2. Componentes do projeto

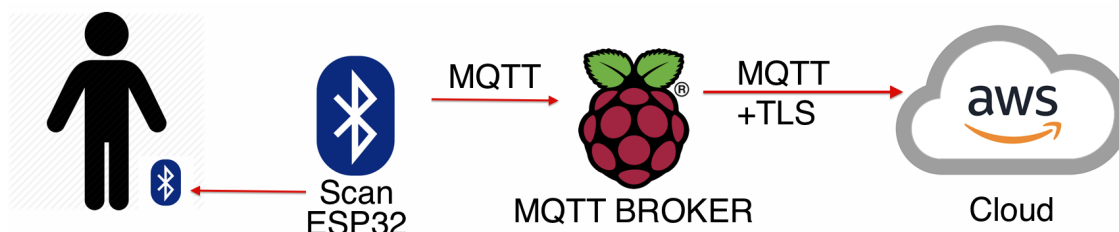


Figure 1. Diagrama de funcionamento do projeto

#### 2.1. Scanners de Localização

Na parte que diz respeito ao BLE, serão usados nesse projeto placas ESP32 que funcionarão como um scanner de BLE. Ao detectar um dispositivo BLE cadastrado, serão enviados os dados desse dispositivo via MQTT para um Broker MQTT que fará a transmissão desses dados para a Cloud.

#### 2.2. Fog Computing

O Fog Computing será um RaspberryPi3b+ responsável por fazer o armazenamento dos dados recebidos, tornar possível tomar ações baseadas nesses dados sem que seja necessário o uso da Web e por fim fazer o encaminhamento das mensagens recebidas no MQTT Broker ativo no Fog para a AWS IoT.

### **2.3. Cloud Computing**

Na nuvem serão armazenados todos os dados que trafegaram pela rede. Para cada nova localização detectada por um Scanner BLE a cada minuto existirá uma amostra armazenada no banco de dados correspondente a esse Scan.

## **3. Construção do Projeto**

### **3.1. Bluetooth Low Energy**

Esta seção será sobre o processo de programação das placas ESP32 utilizadas no projeto. Visto que as mensagens trocadas entre o ESP32 e o RaspberryPi precisam ser do menor tamanho possível por questões de gasto de energia, o nome do BLE Device, por ser parte da string que será enviada ao Raspberry, também deve possuir o menor tamanho possível. Dessa forma, cada ESP32 terá como client\_Id do MQTT uma localização LX, sendo X um número de 0 a 9, enquanto os dispositivos a serem escaneados terão como nome do BLE FUNXXX(Para funcionário) ou MANXXX(Para Manager), sendo XXX um número que pode ir de 000 a 999.

#### **3.1.1. Programação dos Scanners**

Os scanners possuíam como gatilho as strings "FUN" e "MAN" e sempre que um BLE Device tivesse como nome uma string que começasse com essas 3 letras, o gatilho seria acionado e o nome do dispositivo seria adicionado a um buffer. Após o scan, haveria uma publicação MQTT no MQTT Broker para cada nome de dispositivo no Buffer. O código dos Scanners pode ser encontrado em "<https://github.com/jpbrs/IoT-Projects/tree/master/Microlocation-BLE-MQTT-CLOUD/BLEScan>".

#### **3.1.2. Dispositivos a serem escaneados**

Os dispositivos a serem escaneados precisam ter o nome do BLE alterável, para que seja possível colocar como nome FUNXXX ou MANXXX para que o gatilho do Scanner seja acionado. Para o projeto foi utilizado o ESP32 e o código do serviço BLE foi o código padrão do Sketch da ArduinoIDE dos exemplos de "ESP32BLE", portanto não achei que houvesse a necessidade de colocar o código aqui.

#### **3.1.3. Funcionamento**

Por uma questão de Hardware foi visto que a placa ESP32, mesmo sendo uma das melhores do mercado, não consegue executar com 100% de eficiência o Scan BLE e a conexão de WiFi. Quando um Scan BLE era feito a partir da segunda vez, caso o WiFi estivesse conectado o nome do dispositivo BLE não era recebido, mesmo com o endereço sempre sendo entregue ao Scanner. Caso fosse uma aplicação de baixa escalabilidade que se limitasse a achar poucos dispositivos BLE específicos uma solução seria utilizar como gatilho o endereço físico ao invés do nome do dispositivo como gatilho. Porém, como o projeto foi pensado para ter a maior escalabilidade possível a maneira que encontrei para fazer o Scan eficientemente foi desligar o WiFi antes do Scan e religá-lo logo após,

uma solução que apesar de não ser boa, após algumas pesquisas aparentou ser a única que resolvesse meu problema.

### 3.2. Configurando AWS IoT

Para configurarmos a AWS IoT primeiro precisamos definir onde os dados serão armazenados e em qual tópico do MQTT Broker da nuvem acontecerão as publicações MQTT. No que diz respeito ao local de armazenamento dos dados, será usado o serviço da cloud DynamoDB, cuja tabela terá como chave-primária uma combinação dos dados publicados. Apesar do objetivo da aplicação é que os detectores BLE fiquem suficientemente distantes para nunca registrarem que um funcionário está em dois locais ao mesmo tempo, como não há garantia que isso será respeitado por quem for utilizar é melhor ter por garantia um primary key impossível de se repetir.

Após ter definido isso, pode-se começar a configurar a cloud, começando por registrar uma Thing, logo na página inicial. Para registrar essa Thing precisará criar um certificado e uma política atrelada a esse certificado.

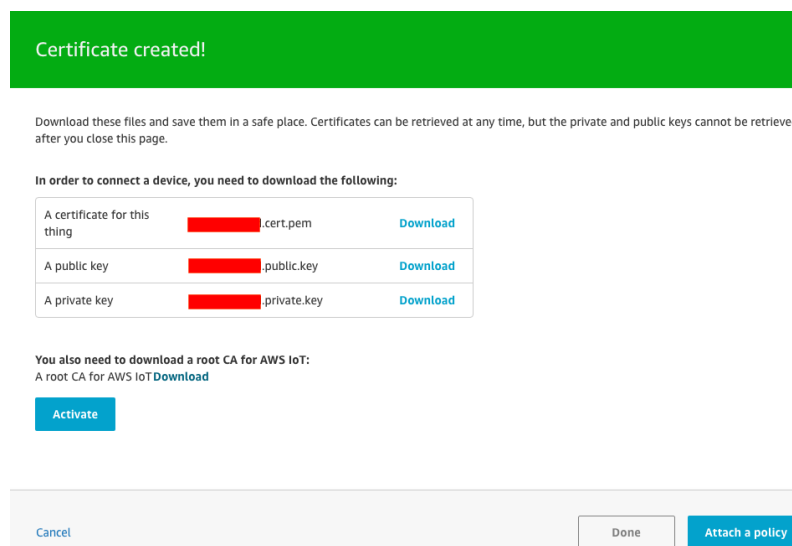
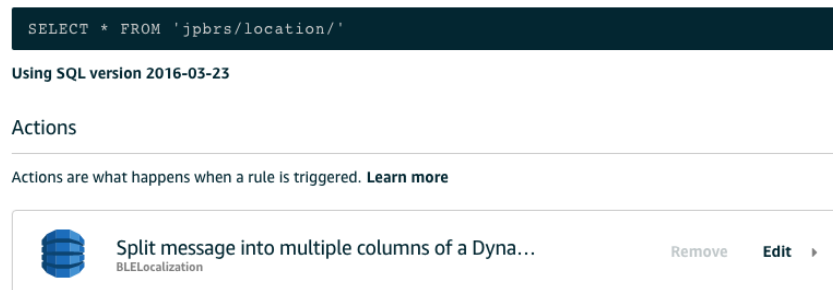


Figure 2. Certificados

Após a criação do certificado será necessário fazer o download de todos os arquivos disponibilizados pelo certificado na AWS IoT, como mostra a figura 2, e ativar o CARoot. Após isso precisa-se criar uma política e atrelar a esse certificado, no caso a política criada com recursos "iot:\*" e permissões "\*" permite que a Thing registrada tenha acesso a tudo dentro da Cloud, então deve-se ter cuidado com o tipo de política a atrelar a um certificado por questões de segurança.

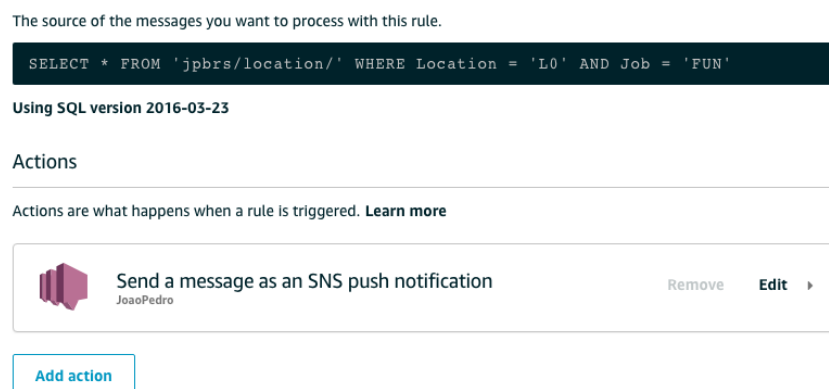
Após os certificados e as políticas terem sido criados a conexão de dispositivos IoT já é possível utilizando MQTT com os certificados de TLS, Autenticação na cloud, criptografia de payload, etc. serem feitos a partir dos arquivos disponibilizados na figura 2. Então agora pode ser feito o que realmente importa, a configuração dos serviços oferecidos pela Nuvem, no caso os serviços escolhidos foram os serviços de armazenamento de dados no Banco de Dados e Notificação via SMS a partir de um determinado gatilho.

A Primeira regra, como mostra a figura 3, faz uma seleção do payload publicado no tópico 'jpbrs/location/' dentro da AWS IoT e coloca em colunas no DynamoDB as informações dos atributos do payload que sempre será um json.



**Figure 3. Diagrama de funcionamento do projeto**

A segunda regra, como mostra a figura 4 é para justamente notificar caso haja a entrada de algum funcionário em um zona específica que não pode ser adentrada. Claro que essa query pode ser aperfeiçoada com avaliação de outros atributos, porém por se tratar de um protótipo não foi pensado em nada mais específico para demonstrar o funcionamento do serviço.



**Figure 4. Diagrama de funcionamento do projeto**

### 3.3. Fog Computing

Como Fog Computing será usado um RaspberryPi3b+ responsável por fazer um backup das mensagens e encaminhá-las com criptografia e segurança à nuvem. Para isso serão utilizados recursos descritos a seguir.

#### 3.3.1. MQTT Broker

Como um MQTT Broker para receber as publicações dos Scanners BLE ESP32 na rede local, será usado o MosquittoBroker. Vale lembrar que o Mosquitto Broker vêm com configurações de segurança básicas e é preciso fazer as configurações necessárias para garantir a segurança da aplicação, como criar uma lista de acesso, lista de usuários, não permitir conexões com clientID anônimo, etc.

### 3.3.2. NodeRed

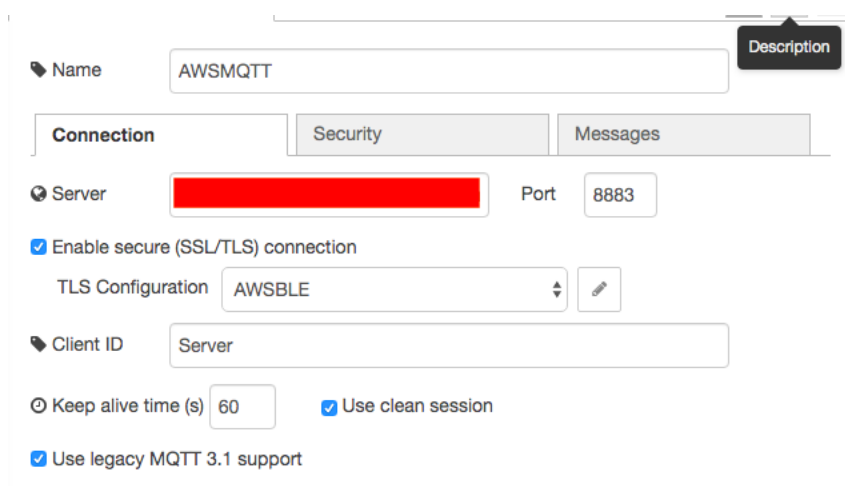
O NodeRed é uma ferramenta muito prática e útil para aplicações IoT. Vale lembrar que em relação à segurança, também deve-se tomar as devidas precauções, como por exemplo colocar um senha e um usuário de autenticação no NodeRed, senão qualquer um conectado a mesma rede que o dispositivo terá pleno acesso ao software acessando a porta 1880 do IP do dispositivo pelo próprio navegador.

O NodeRed acrescenta à aplicação um grande número de possibilidades, como controle das GPIOs do Raspberry, Envio de E-mail, SMS, armazenamento das informações num Banco de Dados, etc. Porém na aplicação o NodeRed foi utilizado para Encaminhamento, Armazenamento e Tratamento de Mensagens.

### 3.3.3. Encaminhamento das Mensagens

O encaminhamento das mensagens diz respeito à transferência das mensagens publicadas ao MQTT Broker do RaspberryPi para o Broker da AWS IoT. Isso foi feito utilizando dois nodes, um node MQTT Subscribe, para inscrever no tópico do Broker que estava recebendo as mensagens e encaminhar a msg para o segundo Node, MQTT Publish.

Para fazer a publicação no Broker da AWS é necessário fazer a devida autenticação com os arquivos da figura 2. O Broker que pode ser usado pela sua conta é encontrado na sua conta da AWS IoT no campo de "endpoint" e será esse endereço que será colocado na parte de Server na parte de configuração do Node de Publish do NodeRed. Lembrando que como se trata de MQTT com TLS e consequentemente criptografia a porta passa a ser 8883 ao invés da porta padrão MQTT 1883.



The image shows the configuration interface for an MQTT Publish node in Node-RED. The 'Name' field is set to 'AWSMQTT'. There are three tabs: 'Connection' (selected), 'Security', and 'Messages'. Under the 'Connection' tab, the 'Server' field is redacted with a black box, and the 'Port' is set to '8883'. The 'Enable secure (SSL/TLS) connection' checkbox is checked. The 'TLS Configuration' dropdown is set to 'AWSBLE'. The 'Client ID' field is set to 'Server'. The 'Keep alive time (s)' is set to '60', and the 'Use clean session' checkbox is checked. The 'Use legacy MQTT 3.1 support' checkbox is also checked. A 'Description' button is visible in the top right corner.

**Figure 5. Conexão com o Broker da AWS IOT**

Quanto a autenticação, os arquivos do certificado devem ser colocados na área de TLS configuration.

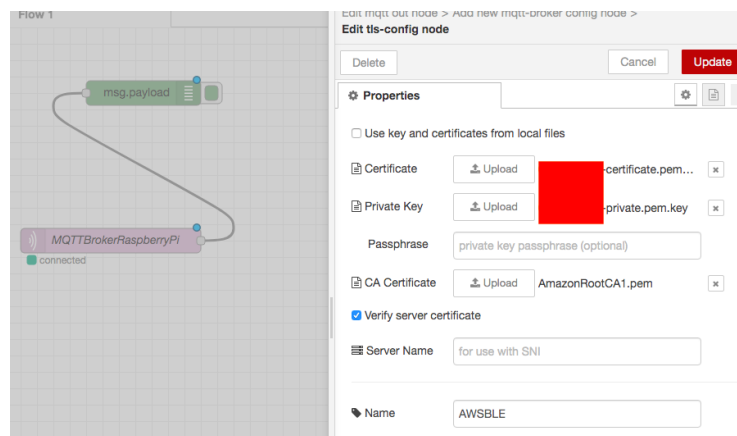


Figure 6. Parte de autenticação com AWS IOT

### 3.3.4. Armazenamento de Mensagens

Em relação ao armazenamento local das mensagens, isso pode ser feito em um banco de dados, disponibilizado como serviço em node do próprio NodeRed, ou pode ser feito por um conjunto de dois nodes para salvar o arquivo como .csv. Primeiro deve-se passar os dados para .csv com o Node csv e logo após com o Node de controle do Terminal pode-se adicionar os dados CSV a um arquivo CSV dentro do raspberry.

### 3.3.5. Tratamento de Mensagens

Como a parte de transmissão de dados é a que mais consome energia em dispositivos IoT, houve a idéia de fazer o tratamento das mensagens no próprio Raspberry ao invés de fazer em cada ESP32. Isso foi feito para transmitir um conjunto de informações mais rico e detalhado para ser armazenado na Cloud. Por exemplo, o ESP32 se limitava a transmitir o numero do Funcionário e o Local, enquanto caso fosse necessário saber a data e a Hora daquele ocorrido o próprio Raspberry adicionaria aos dados. Assim foi possível reduzir a mensagem transmitida pelo ESP32 para 8 bytes, no formato "FUNXXXLX" ou "MANXXXLX" sendo XXX o número do funcionário e X o número do Local que o funcionário se encontra. Enquanto FUN ou MAN seria o tipo do Funcionário, como Funcionário comum, gerente, etc. O que seria útil para adicionar permissões de acesso a locais, por exemplo.

```

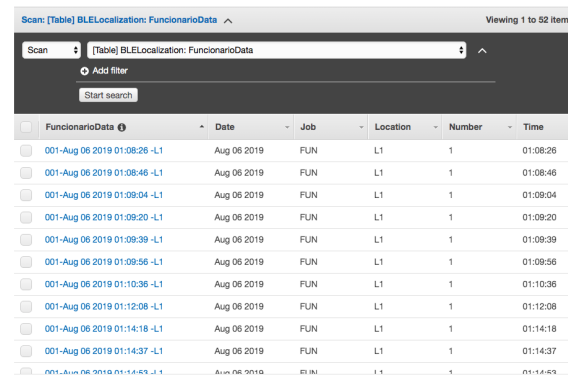
1 var raw_msg = msg.payload;
2
3 fun_type = raw_msg.slice(0,3);
4 fun_number = raw_msg.slice(3,6);
5 fun_location = raw_msg.slice(6,8);
6
7 data = Date(Date.now());
8 data_full = data.slice(0,25);
9 data_hour = data_full.slice(16,24);
10
11 primary_key = fun_number+"-"+data_hour+"-"+fun_l;
12
13
14 location_json = {"FuncionarioData":primary_key,
15 "Job":fun_type, "Number":fun_number,
16 "Location":fun_location, "Time":data_hour};
17
18 msg.payload = location_json;
19
20 return msg;

```

Figure 7. Função JS para tratamento das mensagens

## 4. Resultados

Como resultado as duas placas ESP32 escaneada pelo outro Scanner ESP32 tiveram seus BLE Device Name e Localização enviados para a nuvem. Após o armazenamento na AWS houve a exportação dos dados para que pudesse ser feito a visualização dos dados, que para o dia 06/08/2019 teve o comportamento dos Funcionários como mostra a figura abaixo. Infelizmente por questões de recursos o projeto foi executado com apenas 3 placas ESP32 e portanto só foi possível ter dois como número máximo de funcionários escaneados ao mesmo tempo.



Scan: [Table] BLELocalization: FuncionarioData Viewing 1 to 52 items

Scan [Table] BLELocalization: FuncionarioData

Add filter

Start search

FuncionarioData	Date	Job	Location	Number	Time
001-Aug 06 2019 01:08:26 -L1	Aug 06 2019	FUN	L1	1	01:08:26
001-Aug 06 2019 01:08:46 -L1	Aug 06 2019	FUN	L1	1	01:08:46
001-Aug 06 2019 01:09:04 -L1	Aug 06 2019	FUN	L1	1	01:09:04
001-Aug 06 2019 01:09:20 -L1	Aug 06 2019	FUN	L1	1	01:09:20
001-Aug 06 2019 01:09:39 -L1	Aug 06 2019	FUN	L1	1	01:09:39
001-Aug 06 2019 01:09:56 -L1	Aug 06 2019	FUN	L1	1	01:09:56
001-Aug 06 2019 01:10:36 -L1	Aug 06 2019	FUN	L1	1	01:10:36
001-Aug 06 2019 01:12:08 -L1	Aug 06 2019	FUN	L1	1	01:12:08
001-Aug 06 2019 01:14:18 -L1	Aug 06 2019	FUN	L1	1	01:14:18
001-Aug 06 2019 01:14:37 -L1	Aug 06 2019	FUN	L1	1	01:14:37
001-Aug 06 2019 01:14:53 -L1	Aug 06 2019	DI INJ	L1	1	01:14:53

Figure 8. Dados do Database configurado na AWS IOT

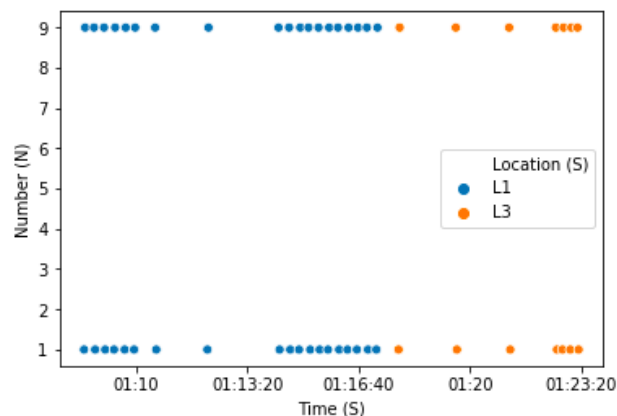


Figure 9. Sendo Number o Numero do ESP escaneado e L1 e L3 os locais onde o suposto Scan foi feito

Lembrando que este trabalho é um protótipo, feito a partir de um compilado de idéias e sem nenhum fim lucrativo ou universitário, o objeto do projeto desde o princípio foi avaliar o funcionamento da aplicação e aprender a construí-la.