Primeiros passos com Internet das Coisas

Outubro 2019
www.embarcados.com.br

IOT - Primeiros passos com Internet das Coisas

Olá,

Obrigado por baixar o nosso ebook: **IOT - Primeiros passos com Internet das Coisas**

Esse ebook traz uma coleção de textos já publicados no Embarcados sobre a plataforma Iot.

Fizemos um compilado de textos que consideramos importantes para os primeiros passos com essa plataforma. Espero que você aproveite esse material e lhe ajude em sua jornada.

Continuamos com a missão de publicar textos novos diariamente no site.

Um grande abraço.

Equipe Embarcados.

IoT: Conectando uma camiseta ao Facebook

EMBARCADOS

Ao pensar nas possibilidades de aplicação da Internet das Coisas (IoT) podemos passar horas e mais horas desenhando soluções para os mais diversos problemas ao nosso redor utilizando esta poderosa ferramenta. À medida que o mundo dos negócios começa a ganhar cada vez mais complexidade, as automatizações, otimizações e melhorias de performance dos processos tornam-se medidas obrigatórias para todos os setores econômicos, colocando IoT em posição de protagonista neste processo.

Em um futuro próximo teremos literalmente qualquer coisa conectada à internet, desde as nossas próprias roupas a cidades inteiras onde todos os carros, semáforos, placas, ruas, etc, estão conectados na rede e se conversam para evitar congestionamentos ou acidentes, por exemplo.

E para provar que podemos conectar praticamente qualquer coisa na rede mundial de computadores, desenvolvi uma camiseta integrada ao Facebook. Utilizando algumas plataformas de hardware que temos à disposição, uma camiseta com uma estampa de um like do Facebook, alguns LEDs, linha condutiva e muita paciência para costurar tudo isso, é possível criar uma camiseta que se ilumina quando alguém clica em 'Curtir' em sua página da rede social.

Legal né?

Figura 1 – Camiseta com LEDs que acendem quando uma determinada página do Facebook recebe um novo like.

Além dos itens necessários descritos anteriormente, também se faz necessário o desenvolvimento de um backend para a nossa aplicação, ou seja, um software rodando em um servidor que será responsável por integrar-se com o Facebook por meio de APIs e disparar eventos para a camiseta, por meio de um protocolo como o MQTT, a medida que a sua página na rede social receba novas interações.

Vamos colocar a mão na massa, ou melhor, no like, para desenvolver esse projeto!

Hardware

Para o desenvolvimento do projeto, utilizei a placa Wemos D1 mini como responsável pela comunicação entre os LEDs da camiseta e o servidor, já que esta possui um ESP8266 embarcado e compõe um circuito compacto, sem deixar de entregar bons recursos ao desenvolvedor.

Além da placa, utilizei também três LEDs lilypad brancos, além de linha condutiva para montar todo o circuito.

A construção do circuito é bem simples, basta costurar a linha em uma das saídas digitais da placa, que será acionada de forma programada, passar por todos os leds respeitando os polos positivos e negativos, e finalizar na porta “ground” do Wemos.

Como fonte de energia, podemos utilizar um powerbank conectado por meio de um cabo USB, ou utilizar o shield do Wemos D1 Mini para uso de uma bateria de lítio, para este projeto, utilizei a primeira opção.

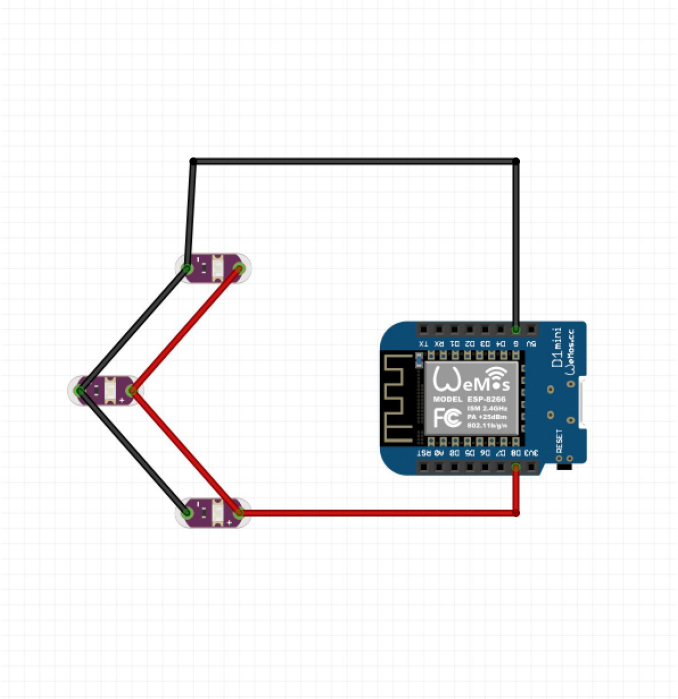


Figura 2 – Esquemático da ligação elétrica.

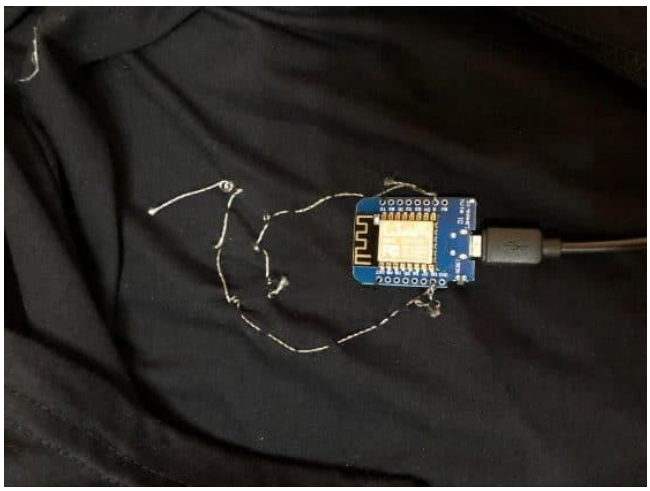


Figura 3 – Wemos D1 Mini no interior da camiseta, responsável por acionar os LEDs e comunicar-se com o servidor.

Software

Em questão de software desenvolvi um servidor, utilizando NodeJS, que é responsável por chamar a API do Facebook (Graph API), onde disponibiliza um endpoint, a qual quando informado o id de uma página, recebemos algumas informações no retorno e dentre elas a quantidade de curtidas.

Vale lembrar que após atualizações desta API o Facebook passou a permitir apenas consultas via API em páginas que o proprietário é o próprio consumidor da API, ou seja, não conseguimos mais consultar a quantidade de likes de uma página que não criamos.

Já para a placa Wemos D1 mini, a programação é feita em linguagem C++, onde criamos uma conexão com um serviço MQTT, permitindo receber dados quando alguém der um like na página do Facebook e acionar o circuito de led da camiseta.

Cloud

O servidor NodeJS chama a API do Facebook frequentemente para checar se houve alguma atualização na quantidade de curtidas, caso haja disparamos uma mensagem MQTT ao serviço Cloud MQTT.

Este serviço possui um plano gratuito com algumas restrições, mas é uma excelente opção para o desenvolvimento de protótipos.

Do outro lado, a placa Wemos foi programada como "subscriber" deste serviço que criamos no Cloud MQTT, dessa forma, assim que o servidor enviar um disparo como "publisher" informando que houve uma alteração na quantidade de curtidas, a placa recebe essa informação e aciona o circuito de LEDs e o desliga após 2 segundos, criando o efeito demonstrado no vídeo abaixo.

Juntando todas as peinhas, temos como resultado o seguinte desenho:

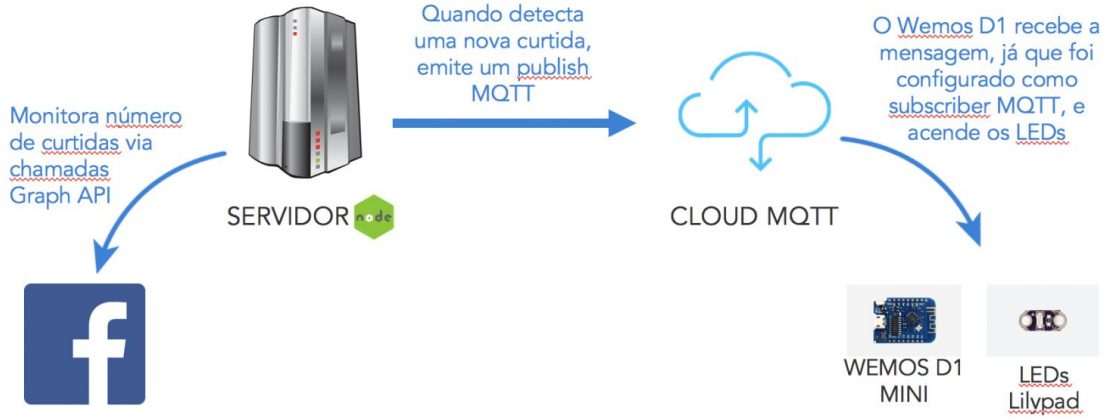


Figura 4 – Diagrama contemplando todos os componentes utilizados no desenvolvimento deste projeto e suas interações.

O projeto Iot-tshirt completo está disponível no github.

“Não é feitiçaria, é tecnologia!”

Este artigo foi escrito por **Guilherme Uezima**

Programando em C# com .NET Micro Framework para Sistemas Embarcados e IoT

Primeiros passos

EMBARCADOS

O uso das facilidades do framework .NET não é mais exclusividade de hardwares que rodem Windows. Graças ao .NET Micro Framework, é possível usá-lo em hardwares embarcados que usam microcontroladores, por exemplo os da família STM32Fxxx da empresa STMicroelectronics.

Atualmente dois projetos se destacam, usando o .NET Micro Framework:

- NetDuino da empresa Wilderness Labs;
- TinyCLR OS (FEZ boards) da empresa GHI Electronics.

Neste artigo trataremos da instalação e primeiros passos com a plataforma TinyCLR OS rodando sob as placas da GHI Electronics modelo FEZ T18. O microcontrolador utilizado é o STM32F401RET6, que tem as seguintes especificações básicas:

Tabela 1 – Especificações básicas do microcontrolador STM32F401RET6

Processador	STMicroelectronics ST32F401RET6
Clock	84 MHz
Memória RAM	96 KByte
Memória Flash interna	512 KByte

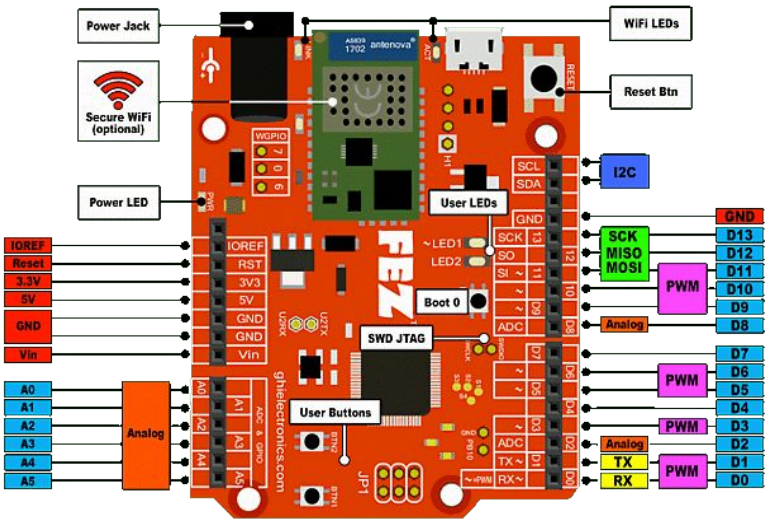


Figura 1 – Placa FEZ ® modelo FEZT18-W (com módulo Wi-Fi)

E as especificações da placa FEZT18 são listadas na tabela 2, abaixo.

Tabela 2 – Especificações das placas FEZT18

GPIO (5V tolerante quando entrada)	22
IRQ	22
UART (Serial)	1
I2C	1
SPI	1
PWM	8
12 Bit ADC	8
USB Client	1
Wi-Fi	Opcional (modelo FEZT18-W)

Instalação do framework

Para programação e debug é utilizada a IDE Visual Studio. Há uma versão gratuita que pode ser utilizada, 2017 Community, e que pode ser baixada diretamente no site da Microsoft. Um paradigma aqui também quebrado é que o Visual Studio pode ser usado não só para desenvolver programas para Windows, mas também para Android e iOS através do Xaramin, servidores Web e, claro, a nossa plataforma .NET Micro Framework.

Após instalado o VS2017, é necessário o download e instalação da extensão TinyCLR, que é super simples.

Ferramentas: Extensões e Atualizações

Vá em “Menu” (para versão em português) e na janela aberta selecione no menu à esquerda Online. Em seguida, na parte de busca digitar TinyCLR, devendo aparecer o resultado da pesquisa como na figura 2.

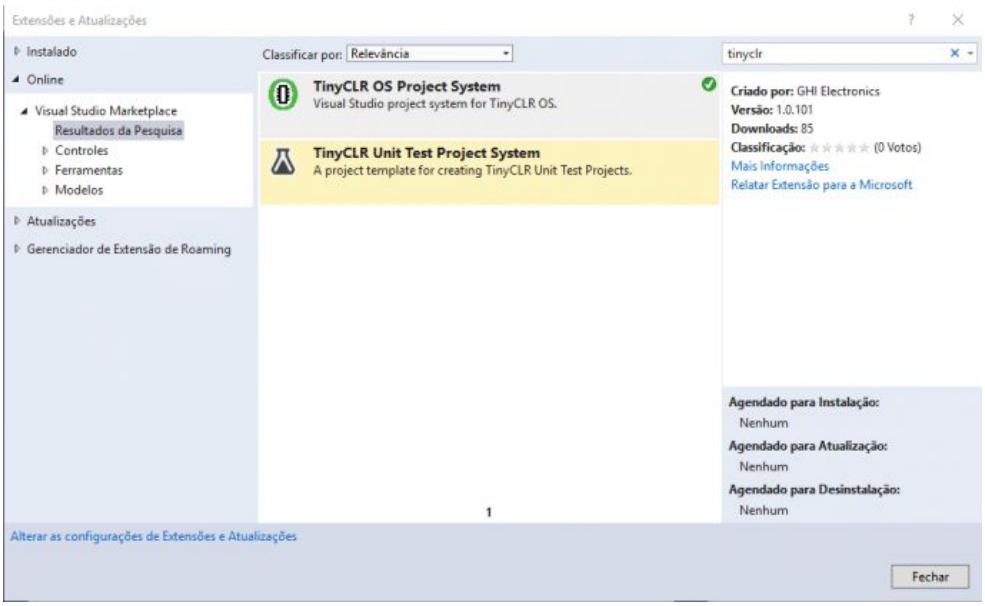


Figura 2 – Instalação da extensão TinyCLR no Visual Studio.

Clique em “Instalar” e após ter sido concluída a instalação, reinicie o Visual Studio.

Criando o Projeto

A criação de projetos é bem similar a outros em C#: vá ao menu “Arquivo -> Novo Projeto/Solução” e deverá ser exibida a janela como na figura 3.

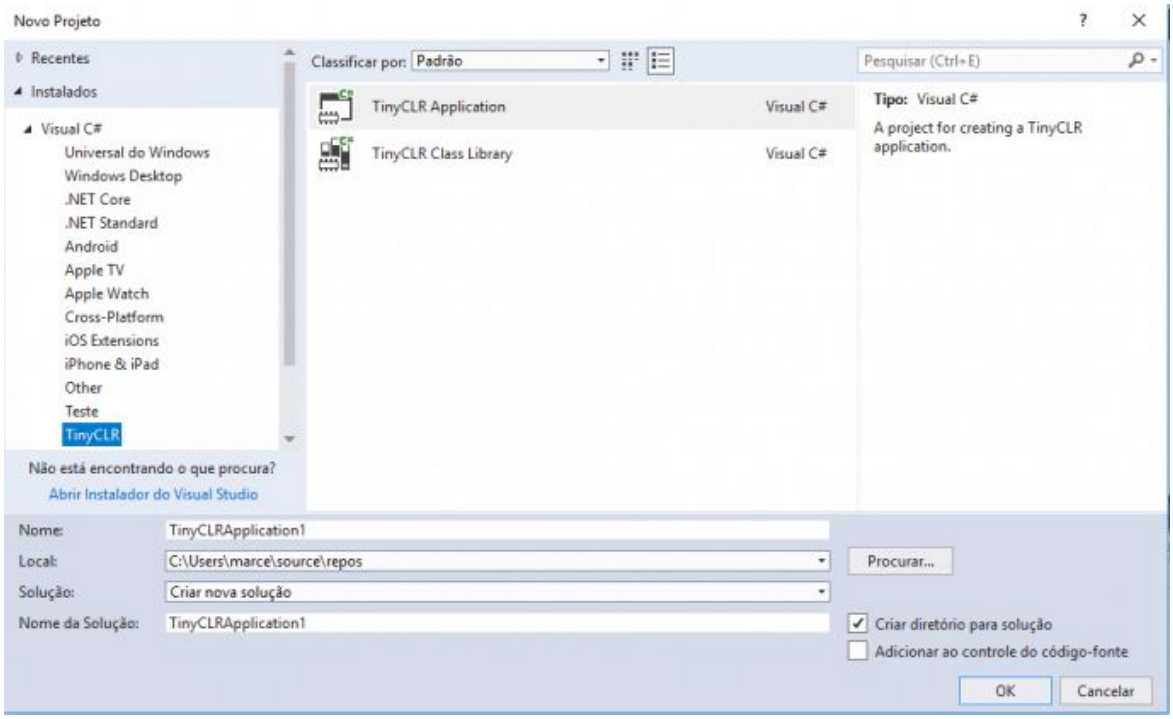


Figura 3 – Novo Projeto TinyCLR.

Mas aqui uma diferença muito importante: no item Visual C# (no menu à esquerda) escolha a opção TinyCLR e, após, TinyCLR Application nas opções no box central.

Assim o Visual Studio criará um projeto com as definições para nossa placa. Após concluída a operação, estaremos na tela de projeto, que deve ser parecida com a da figura 4.

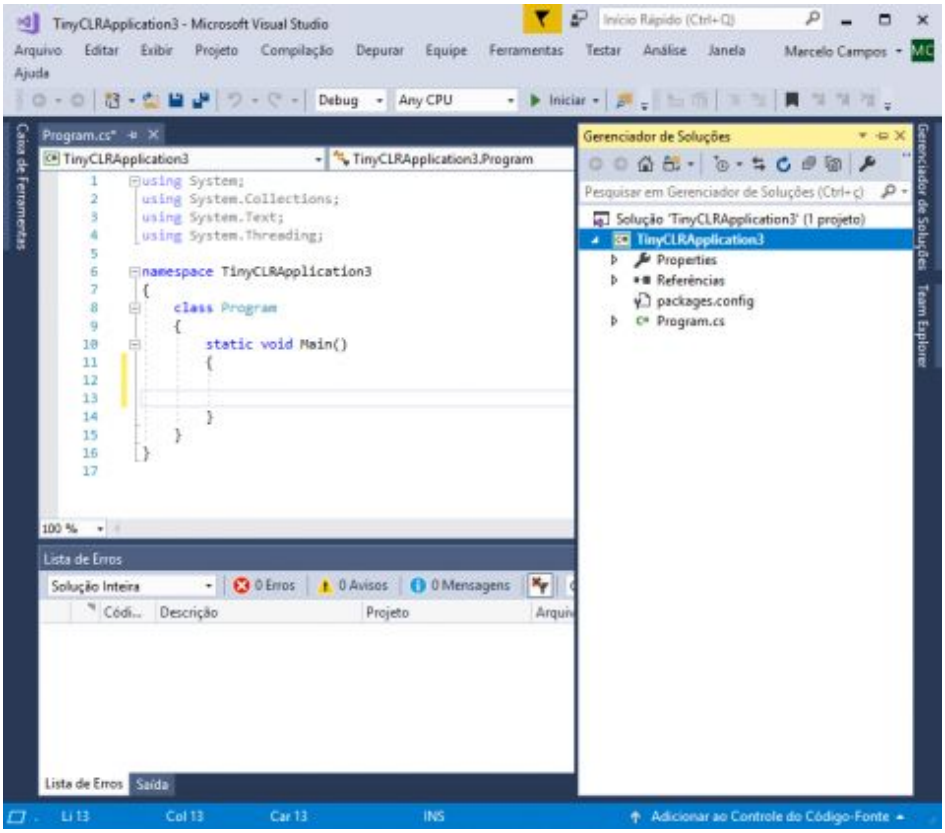


Figura 4 – Tela de Projeto.

A partir daí podemos criar nosso Hello World de sistema embarcado, ou seja, o famoso “Pisca LED”. Mas antes é preciso entender mais um detalhe sobre a plataforma TinyCLR. As Referências são adicionadas ao projeto via Pacotes NuGet e para usá-las também é simples. Basta ir no menu Projeto : Gerenciar Pacotes do NuGet como mostrado na figura 5.

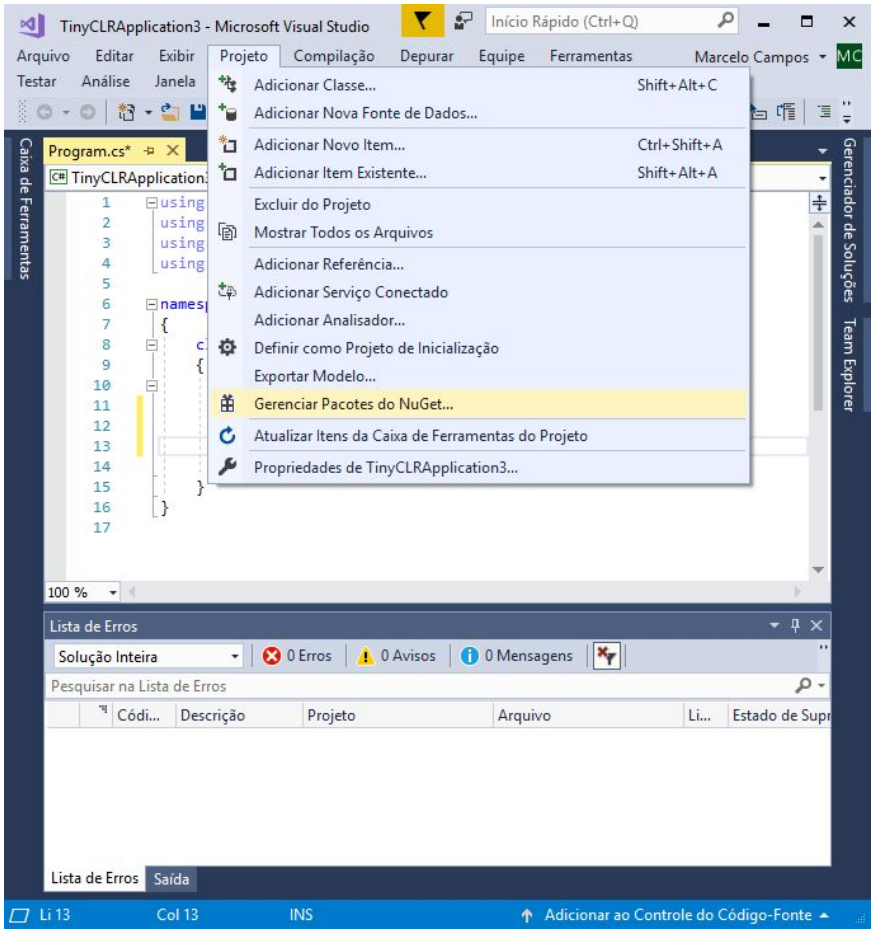


Figura 5 – Incluir e gerenciar pacotes NuGet.

A seguir será aberta a janela de gerenciamento (figura 6) propriamente dita. Nesta janela devemos selecionar a aba “Procurar” e, em seguida, deixar selecionada a opção “Incluir pré-lançamento” para então no campo de busca colocar TinyCLR ou outro texto que desejamos buscar.

Para esse nosso primeiro projeto usaremos 2 pacotes NuGet:

- GHIElectronics.TinyCLR.Devices.Gpio
- GHIElectronics.TinyCLR.Pins

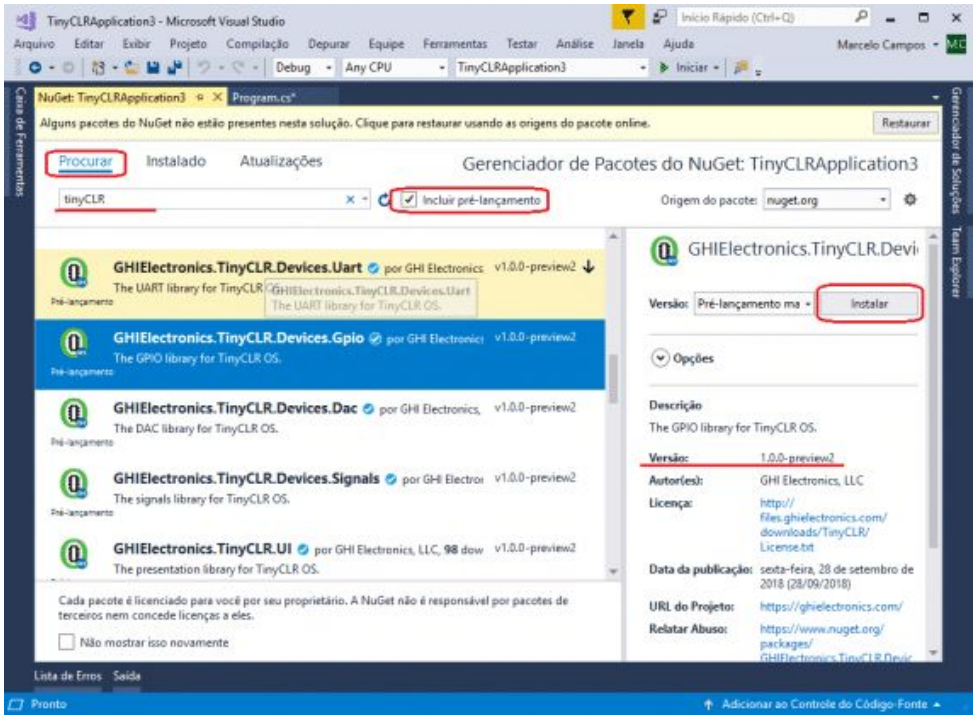


Figura 6 – Gerenciamento de Pacotes NuGet.

Para instalar basta clicar no botão “Instalar” localizado no lado direto nessa mesma janela. Importante sempre notar se a versão do pacote que vamos instalar é compatível com o firmware atual da placa. Atualmente estamos na versão 1.0.0-preview2, e para upgrade de firmware refira-se a este link.

Dica: A versão tem de ser a compatível com a do firmware da sua placa. Para saber qual versão na placa, abra o arquivo packages.config, que fica nos arquivos de projeto.

Também é necessário certificar que a configuração de comunicação com a placa está correta. Com a placa conectada, vá no menu “Projeto -> Propriedades de [nome_do_projeto]” e deverá aparecer uma tela de configuração conforme a figura 7.

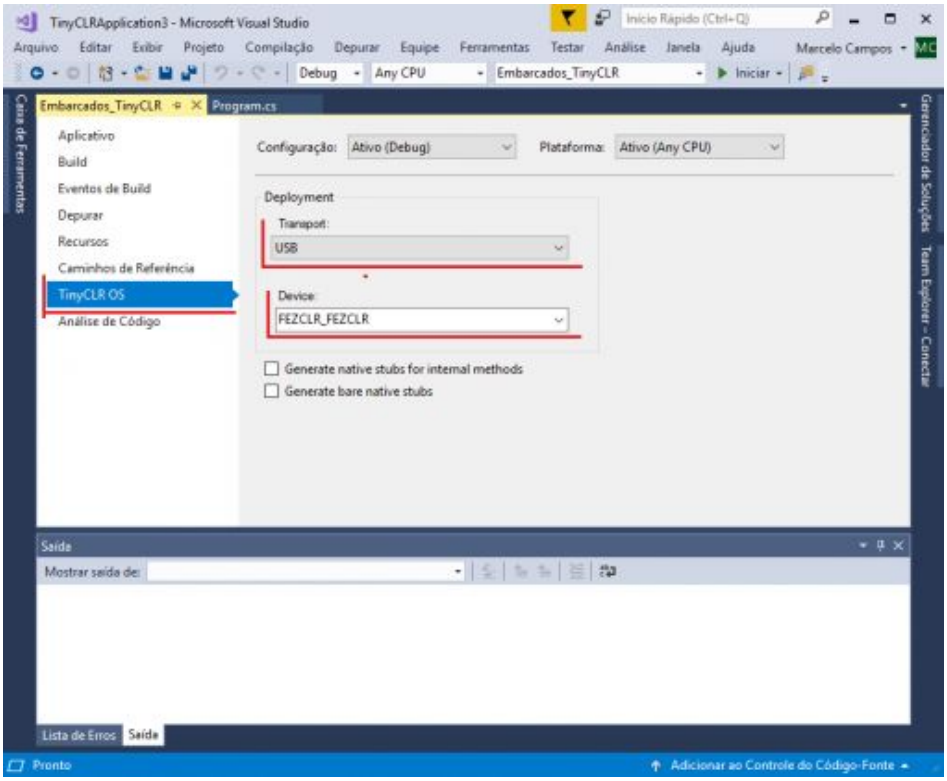


Figura 7 – Propriedades do projeto.

Nessa tela, selecione TinyCLR OS no menu esquerdo e, a seguir, configure caso necessário o item *Transport* como USB e o item Device como *FEZCLR_FEZCLR*.

Importante: A placa deve estar conectada para que as opções estejam disponíveis.

Agora sim, com tudo instalado e configurado, podemos escrever nosso código "Hello World":

Código Hello World – Para baixar o código no GitHub, use este link.

```
// adicionar o 'GHIElectronics.TinyCLR.Devices.Gpio' abaixo via Nuget: menu Projeto -> Gerenciar Pacotes do Nuget
// IMPORTANTE:
//  1. deixar tickado-selecionada a opção "Incluir pré lançamento" logo ao lado do texto de procura do pacote Nuget
//  2. A versão tem de ser a compatível com a do firmware da sua placa, para saber qual versão na placa abra o arquivo "packages.config" fica nos arquivos de projeto
//  caso necessário atualizar, o que pode acontecer mesmo com placas novas:
// https://docs.ghielectronics.com/software/tinyclr/tinyclr-config.html

using System.Threading;
using GHIElectronics.TinyCLR.Devices.Gpio;


namespace Embarcados_TinyCLR
{
    class Program
    {
        static void Main()
        {
            // Instância objeto nomeado 'OnBoardLed' como ~LED1 interno da placa como saída
            var OnBoardLed = GpioController.GetDefault().OpenPin(GHIElectronics.TinyCLR.Pins.FEZ.GpioPin.Led1);

            // coloca esse pino conectado ao led como saída
            OnBoardLed.SetDriveMode(GpioPinDriveMode.Output);

            // Escreve na janela Saída do Visual Studio (menu Exibir : Saída)
            System.Diagnostics.Debug.WriteLine("> Iniciado .NET MicroFramework OK !");

            while (true)
            {
                // Pisca Led
                OnBoardLed.Write(GpioPinValue.High);
                Thread.Sleep(500);

                OnBoardLed.Write(GpioPinValue.Low);
                Thread.Sleep(500);
            }

        }
    }
}
```

Finalmente, para executar o código é o mesmo procedimento de uma aplicação Windows no Visual Studio. Basta clicar em F5 ou no botão “Play” na barra de ferramentas e aguardar pela compilação e carregamento na placa.

Correndo tudo Ok, nossa placa deverá devolver a mensagem de Debug que programamos, informando a inicialização Ok, e entrar em Loop piscando o Led1.

Com esta breve introdução espero ter contribuído na introdução e mostrado a simplicidade do uso dessa modalidade de programação para hardwares embarcados.

Este artigo foi escrito por **Marcelo Campos**

Participe da comunidade no Facebook

AMQP - Protocolo de Comunicação para IoT

EMBARCADOS

Introdução ao AMQP

Neste primeiro artigo iremos apresentar um protocolo de comunicação para dispositivos IoT - o **AMQP** (*Advanced Message Queuing Protocol*). A própria concepção de dispositivos IoT se funde à comunicação: não faria sentido falarmos de IoT sem pensarmos em como estes dispositivos se comunicam - seja entre eles, com o usuário ou, ainda, com um servidor remoto.

Ainda assim, falar somente em "comunicação" é bastante abrangente, uma vez que a comunicação pode ocorrer por meios distintos (óptica? com fio? sem fio?), de formas diferentes (síncrona? assíncrona?) e padronizadas por diferentes protocolos. Considerando o modelo OSI, que abstrai as redes de comunicação em 7 camadas, ao falarmos de AMQP, estamos falando da camada de aplicação. Assim, o AMQP pode ser entendido como um protocolo pertencente à mesma camada do MQTT ou, ainda, do HTTP.

O AMQP é um protocolo que "roda acima" do protocolo TCP, que pertence à camada de transporte. Tal como o MQTT, o AMQP é um protocolo que permite o envio e recebimento de mensagens de forma assíncrona - ou seja, quando enviamos uma mensagem não esperamos uma resposta imediata - independente do hardware, sistema operacional e linguagem de programação.

De forma bastante genérica (e com ressalvas: vide referências), o AMQP pode ser enxergado como o equivalente assíncrono do HTTP, onde um cliente é capaz de comunicar-se com um broker "no meio do caminho".

Antes de, propriamente, falarmos sobre o AMQP, cabe mencionarmos ótimas referências já publicadas aqui no Embarcados sobre a temática da comunicação em dispositivos IoT, como o infográfico do Felipe Lavratti, bem como o artigo do Marcelo Barros, que trata sobre MQTT. Mais adiante faremos um paralelo do MQTT em relação ao AMQP.

(Muito) Breve Histórico

O AMQP é um protocolo originalmente desenvolvido pela comunidade atuante no mercado financeiro, baseado no cliente (customer-driven) que visa permitir a interoperabilidade entre dispositivos. Atualmente, a versão do protocolo padronizada pela ISO/IEC 19464 é a 1.0. No entanto, este artigo baseia-se na utilização do RabbitMQ como broker de AMQP e, nativamente, este broker utiliza o AMQP 0-9-1 (embora também ofereça suporte ao AMQP 1.0). Desta forma, iremos focar no AMQP 0-9-1, que é bastante utilizado no RabbitMQ e em outros brokers.

Conceitos Iniciais

O *broker* é uma entidade que recebe mensagens de *publishers* - ou seja, clientes que produzem mensagens - e encaminha mensagens a *consumers*, que são clientes que recebem (consomem) mensagens. Assim, o AMQP é um protocolo bi-direcional onde um cliente pode enviar e receber mensagens de outros clientes através do *broker*.

Ao publicar uma mensagem, o cliente *publisher* encaminha esta mensagem a outra entidade denominada *exchange*, que de acordo com regras específicas (e programáveis!) denominadas *bindings* as encaminham para filas (*queues*) que, por sua vez, podem estar sendo consumidas por um outro cliente, o *consumer*. *Broker*, *publishers*, *consumers*, *exchanges*, *bindings* e *queues* são as palavras chaves para entender o funcionamento do protocolo e, tal como dito por um colega (Renan Biazotto), "o AMQP é um grande canivete suíço: ele oferece uma grande variedade de regras que podem ser altamente personalizadas de acordo com a finalidade de nossa aplicação."

De forma simples e resumida, o funcionamento do AMQP pode ser comparado a um sistema de correios conforme mostra a Figura 1 a seguir: o cliente *Publisher* escreve uma mensagem e a encaminha para um *exchange*, que se assemelha a uma caixa de correio. A seguir, já no *broker* - que pode ser entendido como os Correios, por exemplo - esta mensagem é encaminhada de acordo com regras específicas (*bindings*) através de rotas a uma *queue*, que pode ser entendida como uma caixa postal, por exemplo. Finalmente, um cliente *Consumer* monitora a queue (caixa postal) para, então, ler a mensagem.

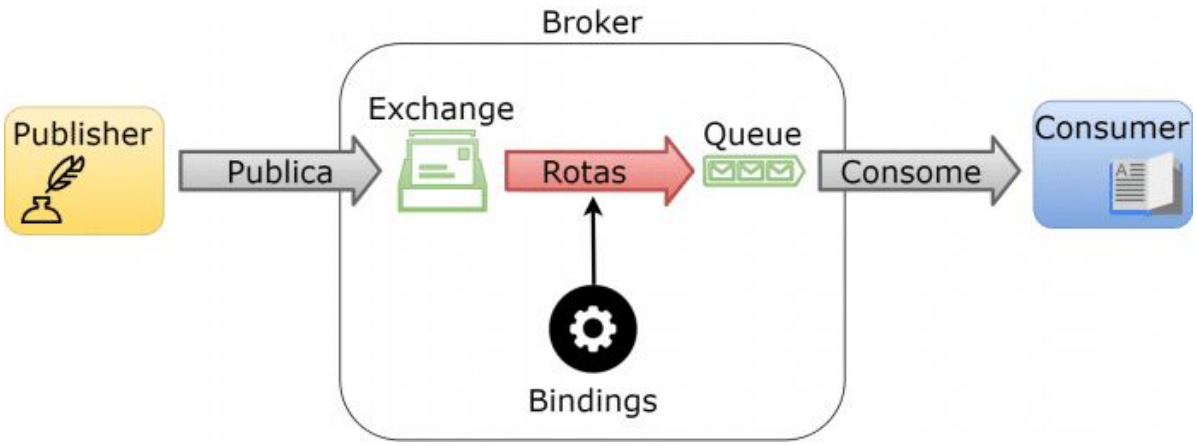


Figura 1- Diagrama do Funcionamento do Exchange tipo Direto no AMQP.

Uma das principais características do AMQP é que o *publisher* jamais publica uma mensagem diretamente em uma fila. A mensagem sempre é encaminhada ao exchange que, de acordo com seu tipo e com as configurações especificadas através de *bindings*, a encaminha para determinada fila ou a descarta.

Protocolo Programável (e Exchanges!)

Rotas, exchanges, filas e outras entidades podem ser programadas diretamente pelas aplicações (clientes). A função de administração do sistema passa a ser somente no sentido de criar regras de acesso e permissões para determinadas ações. Dessa forma, o AMQP 0-9-1 é um protocolo "programável", deixando boa parte das ações a cargo dos usuários/clientes.

Esse grande "canivete suíço" que é o AMQP 0-9-1 proporciona aos desenvolvedores grande liberdade ao desenvolver aplicações específicas e personalizáveis. No entanto, o esforço concentra-se em compreender as especificidades do protocolo, bem como a correta utilização de suas entidades.

Falaremos agora de uma dessas entidades, os *Exchanges*!

Por que é importante conhecer os Exchanges e seus tipos?

O *exchange* é uma entidade que recebe as mensagens que chegam ao broker vindas de aplicações clientes. Estas aplicações desconhecem a existência das filas. É através do *Exchange* que determinada mensagem é roteada para uma ou mais filas a partir de regras denominadas *bindings*.

Há alguns tipos de *Exchanges* que implementam diferentes características de roteamento das mensagens recebidas. A saber, são tipos de *Exchanges*: Direct, Fanout, Topic e Headers. Além do tipo, outros parâmetros definem um Exchange: nome, durabilidade, auto-delete, argumentos, etc.

As mensagens que são publicadas no *broker* e as mensagens que são consumidas possuem um parâmetro chamado *routing key*, que no caso de filas que são atreladas a um *exchange*, também pode ser chamado de *binding key*. Este parâmetro nada mais é do que um identificador do caminho da mensagem e serve de base para armazenar corretamente as mensagens em filas. Este conceito ficará mais claro a seguir, quando apresentarmos o *Exchange* do tipo Direto (Direct) e do tipo Tópico (Topic).

Direct Exchange

O Exchange do tipo Direto entrega mensagens às filas de acordo com o *routing key* da mensagem. Isto é, se uma fila foi bounded (roteada) a um exchange com determinado *routing key* (*binding key*), toda mensagem que chegar ao broker destinada a um exchange do tipo Direct com determinado *routing key*, será redirecionada à fila associada a este mesmo *routing key* (*binding key*). Este tipo de exchange é utilizado principalmente para a distribuição de tarefas entre diversos consumidores.

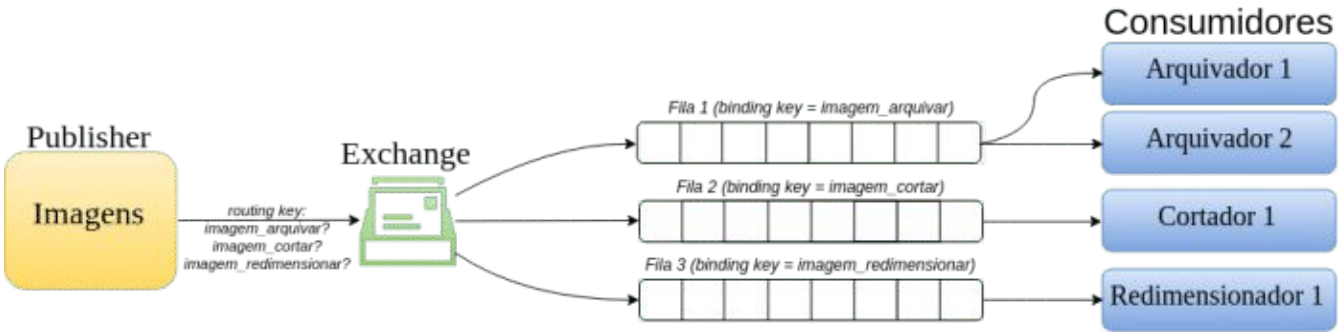


Figura 2 - Diagrama do Funcionamento do Exchange tipo Direto.

Supondo-se o caso de uma tarefa complexa que deseja-se distribuir entre diversos consumidores (workers, neste caso), a Figura 2 exibe o funcionamento do Exchange do tipo direto: a mensagem é enviada ao broker contendo uma operação a ser realizada em uma imagem e, de acordo com o routing key da mensagem, o exchange do tipo Direct encaminha a mensagem para as filas que estão atreladas (bounded) ao exchange com o mesmo routing key (também conhecido como binding key, no caso das filas).

Topic Exchange

O Exchange do tipo Tópico (Topic) é utilizado para comunicação no padrão publica/subscreve (publish/subscribe) e, assim, é utilizado para monitorar mensagens de acordo com padrões no routing key da mensagem, ou seja, o consumidor pode "escolher" que tipo de mensagem deseja receber e, simetricamente, quem publica a mensagem no Exchange indica um assunto/tópico - através do routing key - ao qual a mensagem se refere.

Como um exemplo de aplicação, suponha que gostaríamos de implementar um sistema de logs no qual os consumidores monitoram mensagens de log de diferentes origens/módulos (kernel do sistema, gerenciamento de usuários, rede) e de diferentes níveis (informação, alertas, erros). Cada mensagem gerada contém tanto a origem (kernel/users/network) como o nível do log (info/warning/error) representados como uma lista de dois elementos (poderiam ser mais!) separados por pontos (“.”) no routing key. Por exemplo, a mensagem que contém o routing key igual a “ker.info” representa uma mensagem de log gerada pelo kernel do sistema e representa um nível informacional.

Consumidores destas mensagens podem monitorar determinado routing key (ou binding key, uma vez que estão associadas a uma fila) e executar ações específicas. Por exemplo, o consumidor A, que é responsável pela saúde de todo o sistema, está interessado em monitorar todas as mensagens de log de erros, independente do módulo que gerou a mensagem. Outro consumidor (B, por exemplo) está interessado somente nas mensagens do módulo de gerenciamento de usuários (users) independente do nível (info/warning/error). Para tal roteamento das mensagens, o exchange do tipo Topic permite a utilização de caracteres especiais no binding key das filas associadas ao exchange. Por fim, o consumidor C está interessado em todas as mensagens de log. Assim:

- “*” pode substituir uma palavra no binding key;
- “#” pode substituir uma ou mais palavras no binding key.

Para o consumidor A, este poderia consumir da fila com binding key igual a “*.error”, enquanto que o consumidor B consome de “users.*” e o consumidor C de “**”. Observe que o consumidor C poderia consumir também de “#”, porém no primeiro caso (“**”) o consumidor C receberia mensagens publicadas com exatamente duas palavras no routing key, enquanto que no segundo caso (“#”) consumiria qualquer mensagem publicada no exchange, independente da quantidade de palavras no routing key. Vejamos alguns exemplos.

A Figura 3 a seguir exibe a representação do fluxo de uma mensagem de log do tipo “users.error”. Neste caso, os consumidores A, B e C - associados às filas com bindings para “*.error”, “users.*” e “**”, respectivamente - recebem a mensagem de log com routing key “users.error”, uma vez que os critérios para estes bindings são todos satisfeitos:

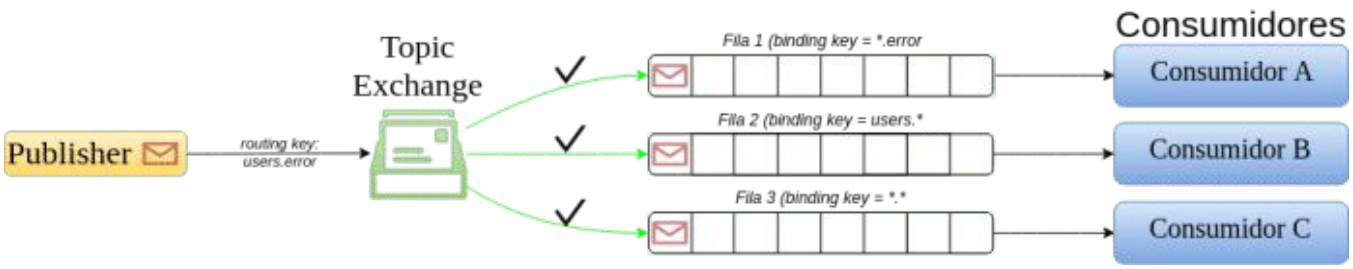


Figura 3 - Diagrama do Roteamento de uma Mensagem de Log “users.error” em um Topic Exchange.

A Figura 4 a seguir representa o roteamento para uma mensagem com routing key igual a “users.info”. Observe que, para este caso, o binding key da fila 1 não é satisfeito e, portanto, a mensagem não é entregue ao consumidor A:

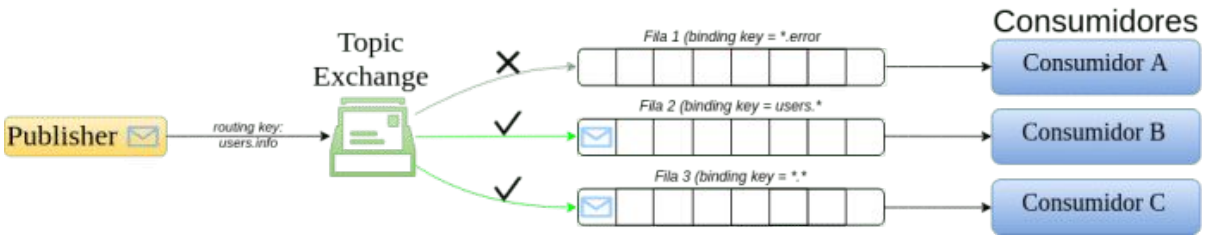


Figura 4 - Diagrama do Roteamento de uma Mensagem de Log "users.info" em um Topic Exchange.

Finalmente, a Figura 5 a seguir exibe o roteamento para uma mensagem com routing key igual a "kern.warning". Observe que somente o Consumidor C recebe a mensagem, uma vez que é o único consumidor da fila que possui um binding compatível com a mensagem.

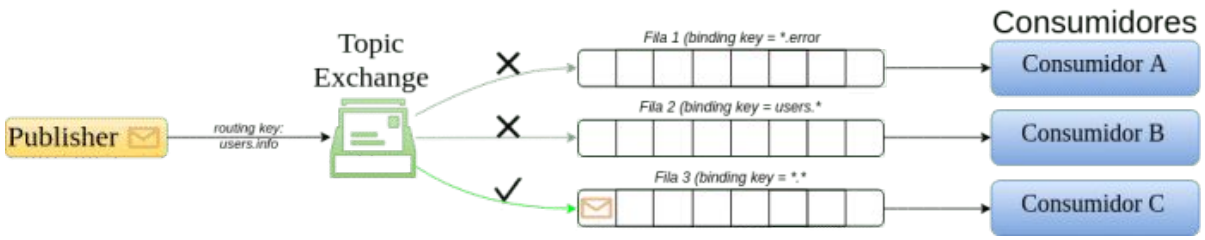


Figura 5 - Diagrama do Roteamento de uma Mensagem de Log "kern.warning" em um Topic Exchange.

Com esta breve introdução, pudemos ver (bem por cima!) um pouco do funcionamento do AMQP 0-9-1. O roteamento das mensagens, a criação de filas atreladas a determinado exchange de acordo com regras (bindings), a definição do tipo do exchange e até mesmo a criação de um exchange podem ser realizados pelos clientes. Assim, confirmando o que já dissemos, o AMQP 0-9-1 é um protocolo que possibilita bastante personalização através do lado das aplicações.

Com isto, podemos seguir para a seção *Mão na Massa!* onde iremos de fato apresentar uma aplicação utilizando o AMQP, focando no Exchange do tipo Topic. Vamos implementar nosso sistema de log! Aguarde o próximo artigo...

Este artigo foi escrito por

Tiago Medicci Serrado e Ronaldo Nunez

Idealização de um projeto IoT portátil

EMBARCADOS

Você já deve ter se questionado como baterias de relógios duram tanto tempo, e que em muitos casos duram vários anos. Então, vamos tentar chegar a este “patamar incrível” com IoT ou até sistemas embarcados? Para isso, serão aplicados desde métodos de baixo consumo em microcontroladores chamados de Sleep Modes, até estratégias de programação que podem diminuir extraordinariamente o consumo do seu projeto portátil.

Nesta série de artigos sobre projeto IoT portátil vamos abordar os conceitos básicos para se ter uma grande economia energética pela parte lógica, o microcontrolador.

Um pouco mais sobre dispositivos portáteis

Se você já se fez a pergunta acima, provavelmente acha fascinante a ideia de algo durar vários anos sem precisar trocar a pilha ou bateria. Isso é muito importante principalmente para o cliente e também para os desenvolvedores, visto que caso seu dispositivo seja um “comilão de baterias”, terá uma má reputação e, consequentemente, as vendas podem ser inferiores do que um dispositivo igual mas com maior economia.

Figura 1 - Bateria SR626 para relógio.

"Como essa bateria minúscula pode durar anos?". Certamente quem faz projetos portáteis precisa pensar em cada detalhe na questão energética, qualquer descuido pode arruinar o tempo de vida da sua bateria, sendo necessário a troca inconveniente ou recarregá-la.

Vamos abordar alguns parâmetros para cálculo de autonomia de baterias em nossos projetos:

- T: Tempo (Horas)
- C1: Capacidade da bateria
- C2: Consumo médio do circuito

$$T = \frac{C1}{C2}$$

$$C2 = \frac{C1}{T}$$

$$C1 = T * C2$$

Como citado sobre os relógios, vamos pegar alguns dados aleatórios apenas para verificar as fórmulas:

- Bateria SR626: 28mAh
- Consumo do sistema: ?
- Tempo: 3 Anos

$$C2 = \frac{28}{24 * 365 * 3}$$

$$C2 = \sim 1uA$$

Podemos observar que o consumo médio do relógio para durar aproximadamente 3 anos é 1uA, que é bem pequeno se você comparar com um LED 5mm por exemplo, que é ~15mA. A potência também deve ser levada em consideração, visto que se podemos trabalhar com uma tensão mais baixa e mantendo a corrente, o consumo será menor (P = V * I).

Baixo consumo em microcontroladores

Normalmente em um projeto de IoT ou em um sistema embarcado de forma geral, temos muitos itens além do microcontrolador (MCU) e, mais especificamente para IoT, onde são amplamente utilizadas **comunicações Wireless**, o transmissor é provavelmente o culpado com maior consumo do sistema, juntamente com motores ou similares se for o caso.

Serão abordados nesta série métodos para minimizar este maior consumo do nosso sistema, podendo, assim, elevar a duração da bateria em patamares semelhantes aos do exemplo do relógio visto acima.

Na grande maioria dos microcontroladores existem modos de economia chamados “Sleep Modes” ou similares. Esses modos desligam setores específicos do microcontrolador ocasionando no menor consumo e, assim, aumentando o tempo que a bateria aguentaria.

Com o foco é em IoT, será utilizado o ESP32 WROOM, que é um microcontrolador relativamente potente. Veja algumas de suas características:

- Processador: Dual-Core (lx6 32-bit) que trabalha de 2 até 240MHz;
- Conexões Wireless: Wi-Fi e Bluetooth (BLE);
- Conexões Wired: SPI, I2C, I2S, SDIO, UART, CAN, ETHERNET...;
- Memória RAM: 520KB;
- Memória FLASH: 4MB.

Um item muito interessante no ESP32 em relação a alguns MCU's desse porte, como PIC32 ou SMT32, é a existência de um terceiro processador chamado ULP (Ultra Low Power coprocessor). Este pequeno "processador" pode ser usado em Deep Sleep para não precisar acordar o MCU para colher dados de sensores ou efetuar comunicações Wired, ocasionando em um consumo médio muito menor.

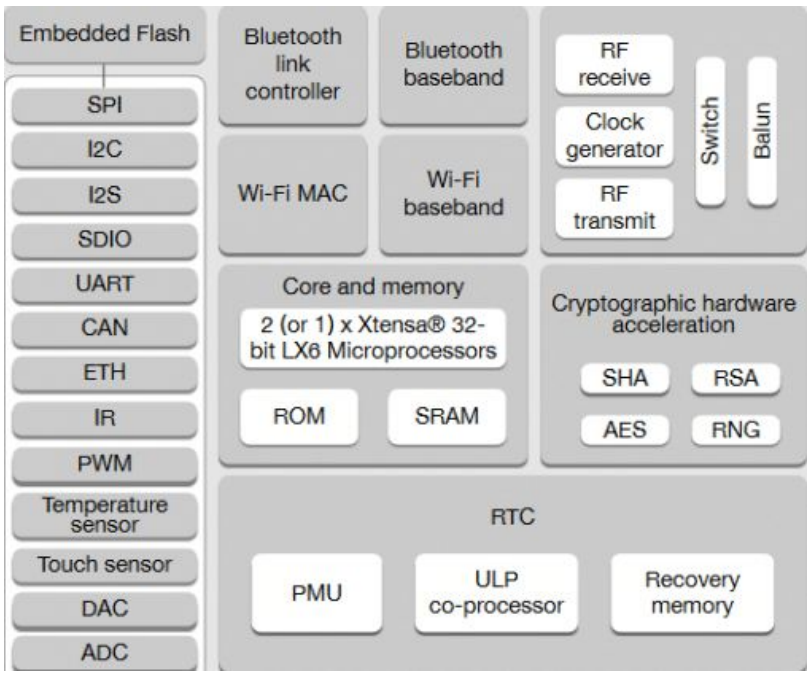


Figura 2 - Seções do ESP32.

O ESP32, em modo normal de transmissão (Tx) ou recepção (Rx), consome:

- **Wi-Fi Ativo (Tx):** 180 - 240mA;
- **Wi-Fi Ativo (Rx):** 95 - 100mA.

Modem Sleep: A parte de transmissão e recepção de dados (Antena WiFi/Bluetooth) é desligada, o consumo varia de acordo com o clock do MCU, que pode ser selecionado (2, 80, 160, 240MHz).

- **Consumo:** 2 - 50mA.

Light Sleep: Apenas o core principal, Digital Domain, está em "pausa", mantendo dados nos registradores e RAM.

- **Consumo:** 800uA.

Deep Sleep: Todo o Digital Domain, incluindo os processadores, são desligados perdendo todos os dados armazenados na RAM e registradores.

- **Consumo:** 10 - 150uA.
- **Obs:** 150uA se aplica ao usar o ULP, caso não, o consumo tende a ~10uA.

Hibernação: São desligados praticamente todos setores do RTC Domain, incluindo as memórias RTC, exceto o RTC Timer e RTC GPIO's, que fazem o MCU acordar.

- **Consumo:** 5uA.

No próximo post da série começaremos a calcular os Sleep's Modes em nosso projeto de IoT portátil, que incluirá desde as contas teóricas para autonomia, até otimizar a programação (Code-Flow) para aumentar o tempo de duração da bateria.

Este artigo foi escrito por **José Morais**

Construindo o projeto de IoT portátil

EMBARCADOS

Finalmente chegamos na parte de começar a dar vida ao projeto IoT portátil. Até aqui você já aprendeu a dimensionar a bateria pro seu projeto e também a calcular o consumo médio. Então vamos prototipar o projeto! As referências sobre instalação do ESP32 na Arduino IDE e também como usar o Google planilhas para enviar dados do microcontrolador para planilha são importantes.

Já foi citado como o projeto funcionará, mas vamos relembrar. Será um *Datalogger* que enviará a temperatura e umidade do local para um banco de dados online, usaremos o Google planilhas pois dispensa um servidor para host, gratuito, permite compartilhar os dados facilmente com qualquer pessoa, aplicar fórmulas, desenhar gráficos e muito mais ao estilo Excel. Vamos programar logo e depois as explicações. Código do projeto pode ser obtido aqui.

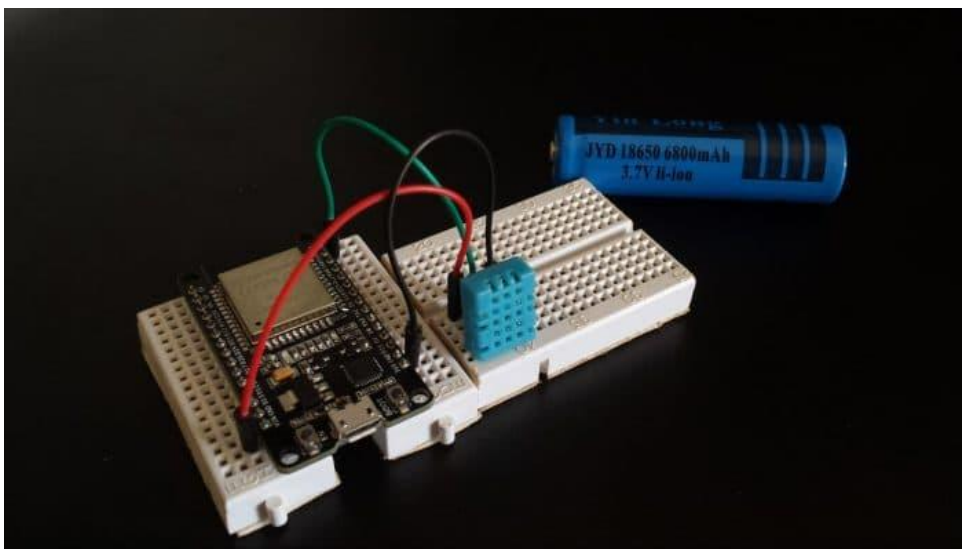


Figura 1 - Protótipo simples do projeto portátil com bateria 18650.

Carimbo de data/hora	Temperatura	Umidade
10/12/2017 10:12:55	28	42
10/12/2017 10:17:58	28	58
10/12/2017 10:23:01	28	41
10/12/2017 10:28:04	28	42
10/12/2017 10:33:07	28	42
10/12/2017 10:38:10	27	57
10/12/2017 10:43:13	28	41
10/12/2017 10:48:15	28	56
10/12/2017 10:53:19	28	41
10/12/2017 10:58:21	28	47
10/12/2017 11:03:24	28	46
10/12/2017 11:08:27	28	46
10/12/2017 11:13:30	28	46
10/12/2017 11:18:33	28	53
10/12/2017 11:23:36	28	45
10/12/2017 11:28:39	28	45
10/12/2017 11:38:44	27	50
10/12/2017 11:43:47	28	45
10/12/2017 11:48:50	28	45
10/12/2017 11:58:56	27	53
10/12/2017 12:03:59	29	44
10/12/2017 12:09:01	29	44
10/12/2017 12:19:07	27	53
10/12/2017 12:24:10	29	43
10/12/2017 12:29:13	29	43
10/12/2017 12:34:16	29	43
10/12/2017 12:39:18	28	51

Figura 2 - Dados e gráfico gerado pelo projeto com DHT11.

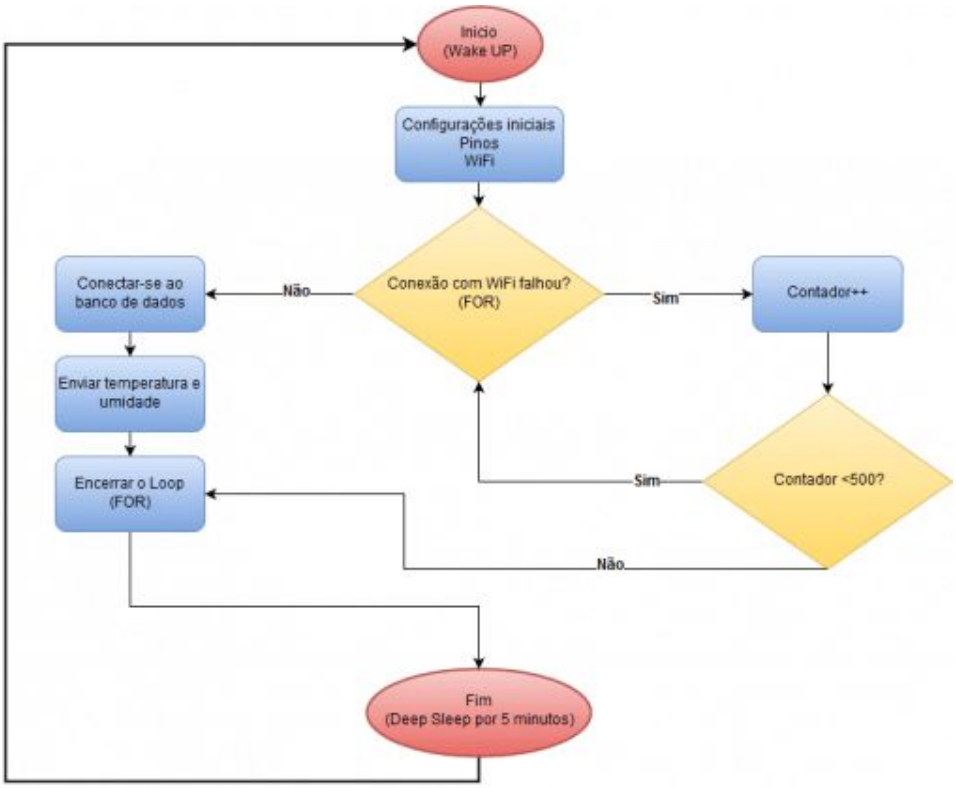


Figura 3 - Fluxograma do projeto.

Observações sobre os dados:

- **Intervalo entre leituras:** 5 minutos;
- **Total de leituras:** 92;
- **Tempo total de colheita:** 9 horas
- Apesar das leituras de umidade oscilarem bastante por motivos desde protoboard até sensor ruim, com a linha de tendência linear foi possível notar a redução da umidade conforme o aumento da temperatura.

Como em todo projeto de sistemas embarcados, as condições do ambiente a ser monitorado podem influenciar no funcionamento, dando margem a melhorias de desempenho (e, nesse caso, consumo energético). Observando o ambiente cuja temperatura foi monitorada, nota-se que a temperatura do local demorou aproximadamente 2 horas para variar 1°C e, neste meio tempo, foram enviados 24 valores iguais de temperatura. Para este projeto, estes 24 valores iguais de temperatura significam 24 envios de um mesmo valor de temperatura, levando à conclusão que estes envios causaram grande consumo energético desnecessário (uma vez que o grande "vilão" do consumo no projeto é a transmissão de dados por Wi-Fi, conforme visto na parte 1 desta série). Logo, poderíamos enviar apenas valores diferentes de temperatura, diminuindo, assim, o uso do Wi-Fi e, por consequência, aumentando a economia da bateria, nesse caso para 24x.

Vamos fazer algumas contas para dar um norte a esta nova informação, que deixará o ESP32 dormindo por 2 horas:

- **Capacidade da bateria:** 3000mAh;
- **Consumo médio com 2 horas de sleep:** 36uA;
- **Obs:** A transmissão foi feita em média a cada 2 horas, portanto o ESP32 dormiu 2 horas e depois enviou.

Consumo médio com 2 horas de sleep:

$$Cm=(0.15*1+0.000015*7200)/(1+7200)$$
$$Cm\approx 36uA$$

Tempo de duração da bateria com 2 horas de sleep:

$$T=3/0.000036$$
$$T=83333Horas$$

Veja que para este caso, com 2 horas de intervalo entre transmissões de dados, a bateria duraria (teoricamente) 9 anos e 200 dias! Incrível não?! Mas nem tudo são flores, todas contas desse projeto até agora foram **apenas para dar um norte**, excluindo todos componentes extras do circuito. O tempo real de execução do código ainda será calculado e o consumo provavelmente será bem maior por conta dos componentes extras no circuito.

Vamos começar a fazer alguns ajustes de tempo e consumo para que no próximo post seja calculado o tempo real de duração para nosso projeto.

Foi observado o tempo em que cada ação demora para ocorrer, separado por partes:

- **Conexão com Wi-Fi:** 2.5 segundos (por causa do Wi-Fi - SCAN);
- **Comunicação com a planilha:** 1.7 segundos;
- **Total:** 4.26 segundos.

Agora precisamos do real consumo do circuito, que anteriormente foi levado em consideração apenas o ESP32. Entretanto, há um componente a mais, o DHT11. De acordo com o Datasheet do DHT11, o consumo é:

- **Em medições:** 300uA;
- **Em Standby (sleep):** 60uA.

As medições do DHT11 podem demorar até ~25mS, o restante do tempo ele se encontra em Standby. Logo, vamos arrumar o último cálculo de consumo do projeto (Figura 4).

Consumo médio do projeto que se aproxima a realidade:

$$Cm=(0.15*4.26+0.000075*7200)/(4.26+7200)$$
$$Cm\approx 164uA$$

Tempo de duração da bateria que se aproxima à realidade:

T=3/0.000164

T≈18293Horas

Veja que arrumando as contas com novos tempos e componentes, o consumo ficou 4.5x maior que os anteriores, que eram apenas simulações para nos guiar. Entretanto, a bateria ainda irá durar ~2 anos, que é um bom intervalo.

Observações importantes em relação aos tempos do ESP32

1. A conexão com Wi-Fi pode chegar até ~300mS, entretanto, como o ESP32 é reiniciado totalmente, é preciso configurar toda pilha Wi-Fi novamente. E o ESP32 faz o uso da LWIP, que não é muito otimizada em velocidade, e algumas mudanças, como IP fixo, podem melhorar esses tempo;
2. Se você precisa de uma inicialização do Deep Sleep mais rápida, há inúmeras configurações que permitem isso, por exemplo, a calibragem do RTC_SLOW_CLK que leva por padrão 1024 ciclos do XTAL principal (~2.5uS) e o tempo extra de inicialização da FLASH (~2mS);

1. Você pode notar alguma perda de “temporização” ao usar longos períodos de Deep Sleep, isso acontece pois o cristal do RTC é mais instável que o cristal principal. Você pode melhorar a precisão configurando uma calibragem maior (como citado na Obs 2), que é útil em Deep Sleep’s de longo período.
2. A conexão com a planilha é relativamente lenta, pois a conexão ao servidor do Google planilhas é feita por SSL/TLS (HTTPS) e isso necessita de um processamento maior e maior número de trocas de mensagens entre cliente e servidor. Logo leva a um tempo de Wi-Fi ativo maior do que se comparado ao caso de uso de HTTP “simples” ou MQTT, por exemplo.

Nosso projeto finalmente está vivo e pronto, mas existe algo a mais que podemos fazer para melhorar o consumo e aumentar ainda mais a duração da bateria, caso seu projeto precise de extrema economia: **estratégias de programação**. Vamos melhorar a eficiência desse código apresentado, incluindo verificações de dados e um modo de economia a mais, o Modem Sleep.

Este artigo foi escrito por **José Morais**

Introdução a LoRa®, NB-IoT e Sigfox

EMBARCADOS

A redução dos custos dos componentes eletrônicos viabilizou milhares de novos produtos e aplicações: rastreamento animal, monitoramento de vias públicas, controle de iluminação, dentre outros. Neste novo contexto, o conceito IoT tem-se popularizado. A sigla IoT significa Internet of Things (Internet das Coisas), que é um conceito amplo que envolve a conexão de dispositivos à rede, análise e inteligência de dados.

Segundo a instituição de inteligência de mercado Machina Research, em 2025 o mundo terá 27 bilhões de dispositivos conectados. Este cenário abre espaço para desenvolvedores e novas empresas, assim como gera dúvidas sobre os caminhos a serem seguidos. Estes são abundantes e as escolhas situam-se tanto no campo de hardware, firmware, infraestrutura de rede e ferramentas de gerenciamento.

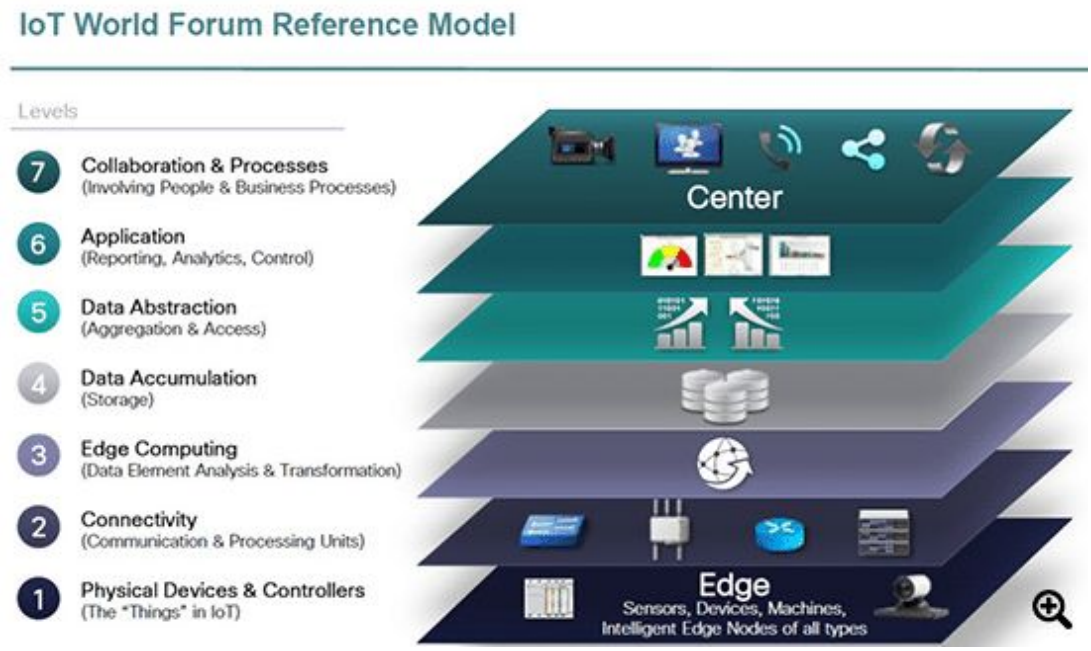


Figura 1 - Camadas de um sistema IoT. Fonte [2]

A Figura 1 mostra as diversas camadas de um sistema completo de Internet das Coisas, segundo o modelo proposto pelo IoT World Forum. Cada nível implica uma escolha e pela pluralidade de ofertas isto não é trivial. Dada a complexidade do tema o artigo focará na conectividade e em um pequeno subsistema de amplo espectro de aplicação: as tecnologias de radiofrequência de longo alcance LoRa®, Sigfox e NB-IoT.

Rede LoRa

A rede LoRa é uma solução sem fio sub-GHz em frequência não licenciada que endereça demandas para conexão entre dispositivos para aplicações de baixo consumo, longa distância (em alguns locais é possível conseguir 15 km) e baixo custo de infra-estrutura, considerando-se o grande número de nós. Como é possível verificar abaixo, a rede LoRa é bem popular e tem tido bastante aderência ao redor do mundo.

Figura 2 - Adoção da rede LoRa ao redor do mundo. Fonte: LoRa Alliance/agosto 2017

De maneira simplificada, o sistema Lora vai funcionar com um módulo sub-GHz no end-device e um gateway que enviará os dados para servidores locais e/ou remotos.

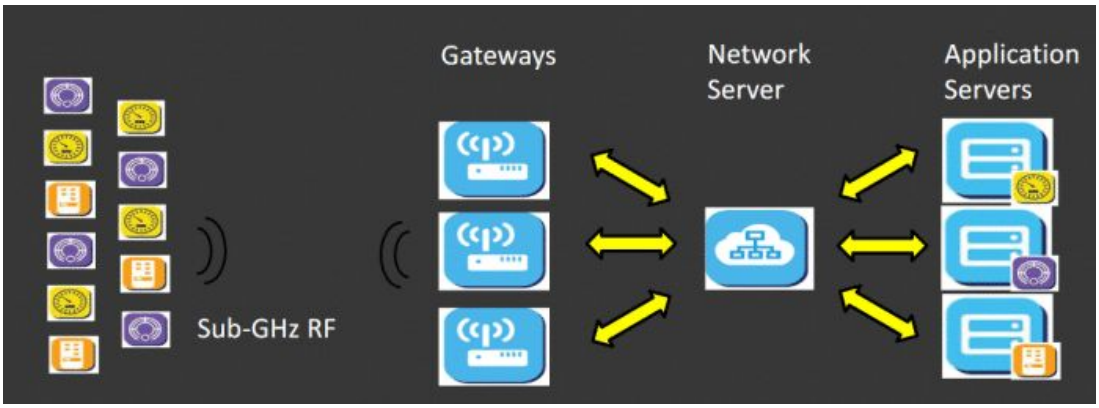


Figura 3 - Topologia da Rede Lora. Fonte: LoRa Alliance/agosto 2017

Podemos considerar a rede Lora LoRaWAN™como tendo uma arquitetura bastante aberta: as empresas podem criar redes próprias para seus end-devices ou usar redes de terceiros. Na Figura 4 abaixo exemplifica-se como uma empresa criaria sua infraestrutura básica: cada nó do sistema teria um módulo LoRa que se comunicaria com um gateway.

Figura 4 - Exemplo de uma estrutura de rede LoRa

Esta abordagem em termos físicos simplifica e dá liberdade aos usuários que podem criar uma infraestrutura própria e conseguir uma cobertura ampla.

NB-IoT

O organismo de normas internacionais 3GPP concluiu a normatização do NB-IoT (Narrow Band IoT) em meados de 2016. Este padrão usará frequências licenciadas, as mesmas usadas por algumas frequências do 4G, permitindo que algumas estações sejam atualizadas e compartilhem sua infraestrutura. A taxa de dados é relativamente alta, aproximadamente 50 kbps.

Features	LTE (Cat3+)	LTE (Cat1)	LTE-M	NB-IOT
Module Cost	$$$$	$$$	$	$
Module Availability	Now	Now	Q3 2017	TBD
Data Rates	~100 Mbps	~10 Mbps	~370 kbps	~50 kbps
Battery life	Days	Weeks	10 years (eDRX/PSM)	10 years (eDRX/PSM)
Mobility	High	High	Mid[1]	Low
Coverage	•Standard	•Standard	•Up to 15dB	•Up to 20dB
Voice support	Yes (VoLTE)	Yes (VoLTE)	Q2 2018 Target	No

Figura 5 - Características NB-IOT. Fonte AT&T

Para ilustrar a aplicação a Figura 6 mostra uma possível aplicação para medidores inteligentes. A empresa que desenvolve os medidores utiliza um módulo NB-IoT e conecta o mesmo à rede celular compatível, não sendo necessária a compra de gateways e sim a subscrição a um serviço das operadoras que oferecem o serviço, como a AT&T. Isto facilita bastante o processo de conexão dos dispositivos, contudo pode trazer alguns limitantes tal como a cobertura da telefonia na rede rural.

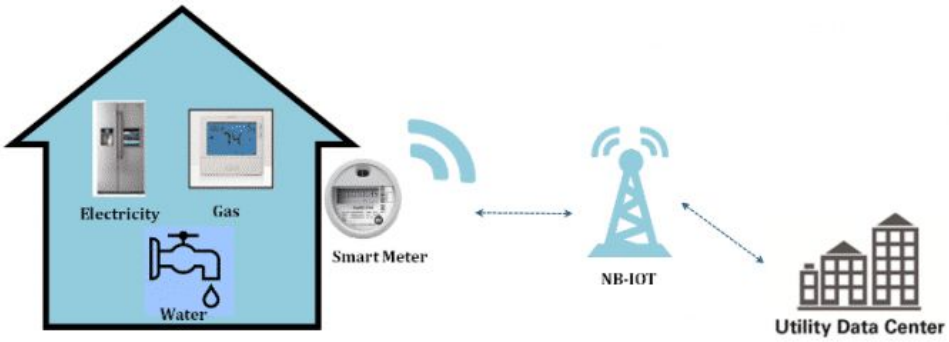


Figura 6 - Exemplo de aplicação NB-IoT. Fonte Huawei

Sigfox

A rede Sigfox é uma rede narrow-band ou ultra narrow-band. O modelo de negócios adotado é diferente daquele adotado pela LoRa Alliance. A empresa Sigfox e seus operadores detêm o controle da tecnologia, servidores, abrindo espaço para outras entidades desenvolverem os dispositivos (end-points). Empresas como a ST, Silicon Labs e Microchip fabricam rádios compatíveis com a tecnologia.

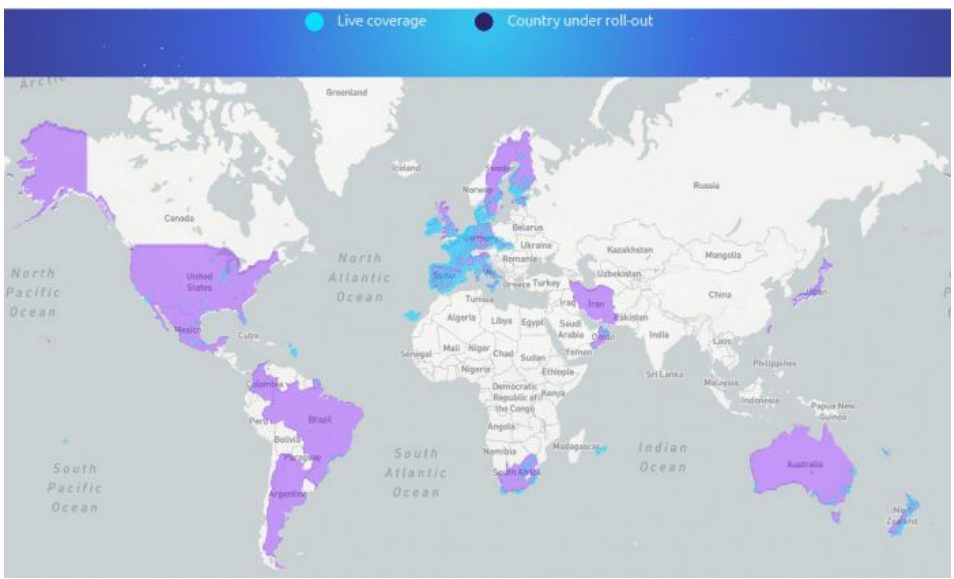


Figura 7 - Cobertura Sigfox. Fonte: Sigfox

Pela Figura 7 a cobertura da rede Sigfox é bem ampla e o custo dos módulos Sigfox tende a ser bastante atrativo. Uma empresa interessada em desenvolver um produto baseado em Sigfox deve desenvolver um hardware com um módulo sub-GHz compatível com a tecnologia e contactar o operador da região para informações sobre custo e pacotes de dados.

Conclusão

Durante os projetos que envolvem Internet das Coisas, as empresas devem fazer uma série de questionamentos sobre todas as camadas do projeto: hardware, software, servidores. O presente artigo tratou de três redes LPWAN (Low Power Wide Area Network): LoRa®, Sigfox e NB-IoT. Cada qual tem sua particularidade e possui características que se adequam a diferentes cenários. E para a realização da escolha adequada deve-se avaliar o custo de infraestrutura, serviço mensal, gerenciamento de dispositivos e cobertura de área, e não somente o custo de hardware.

Este artigo foi escrito por **Bruno Nunes**

Conectando o ESP8266 no Bluemix IoT

EMBARCADOS

Muitos já sabem que é possível usar o **ESP8266** como um microcontrolador, utilizando-o no lugar do Arduino. Com essa possibilidade fica muito mais fácil realizar diversos projetos de maneira mais simples. Um ESP8266 conectado direto ao sensor e enviando dados para a nuvem, facilita muito os projetos mais simples que tentam abocanhar um pouco dessa grande área que é a internet das coisas (IoT).

Contudo, ainda vejo muitas pessoas, nos hackathons que participei, com grandes dificuldades para conseguir realizar a conexão do ESP8266 com algumas plataformas de IoT. Neste artigo vou mostrar como realizar essa conexão com a plataforma da IBM, o Bluemix.

Pré-requisitos

- Uma conta no Bluemix;
- Um ESP8266;
- Um computador com o Arduino IDE.

Observações sobre os materiais utilizados

- Arduino IDE versão 1.8.3 (Windows e Linux);
- Configuração no Boards Manager do ESP8266 versão 2.3.0;
- Biblioteca PubSubClient versão 2.6.0;
- ESP8266 12E;
- Windows 10 Profissional.

Configurando a IDE Arduino para programar o ESP8266

Para configurar a IDE para programar o módulo ESP8266, execute os seguintes passos:

- Dentro da interface da IDE clique em *File > Preferences*;
- Adicione a URL http://arduino.esp8266.com/stable/package_esp8266com_index.json no campo *Additional Boards Manager URLs* e clique OK;
- Clique em *Tools > Board > Bards Manager*;
- Encontre a placa **esp8266 by ESP8266 Community,** selecione a versão 2.3.0 e clique em instalar.

Assim teremos disponíveis as placas da linha ESP8266, suas bibliotecas e exemplos incluídos.

Configurando o IBM Bluemix

Primeiro, é preciso criar e configurar uma aplicação na plataforma bluemix para receber os dados do nosso dispositivo. Para isso, acesse o portal do bluemix, e crie uma conta. Depois de criada, realize o login. Crie uma organização e um space para suas aplicações. A região escolhida no meu caso foi “US South”.
Você irá se deparar com uma página similar a apresentada na Figura 1.

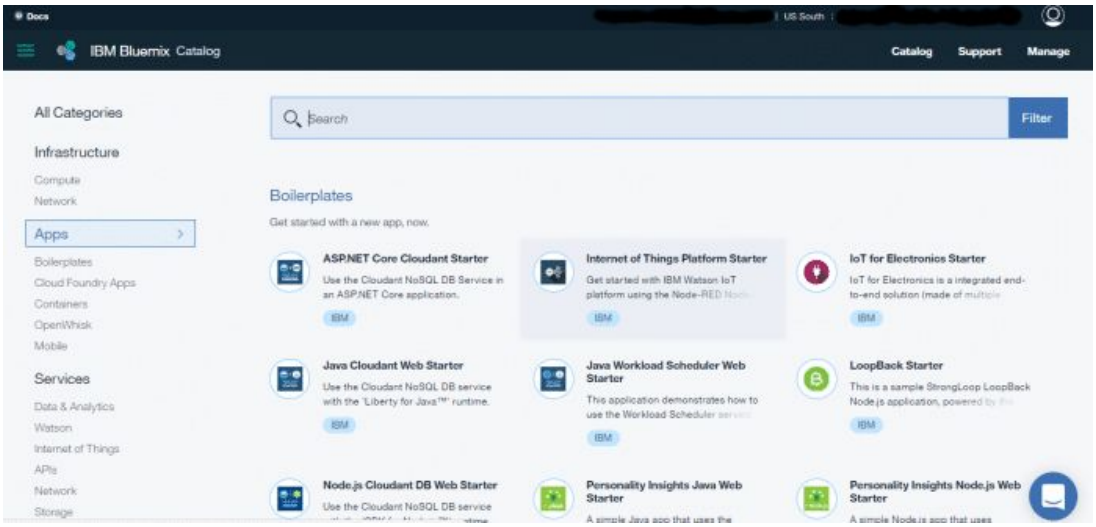


Figura 1 - Dashboard da plataforma Bluemix

Dentro da categoria “Apps”, busque por “Internet of Things Platform Starter”. Criaremos uma instância desse aplicativo na *cloud*.

Dê um nome a nosso aplicativo, e finalize a criação do App clicando em “Create”. Esta aplicação nos criará uma interface com Node-Red, a qual utilizaremos para receber e tratar os dados dos sensores recebidos. No meu caso, dei o nome de “ESP8266-IoT-Embarcados”.

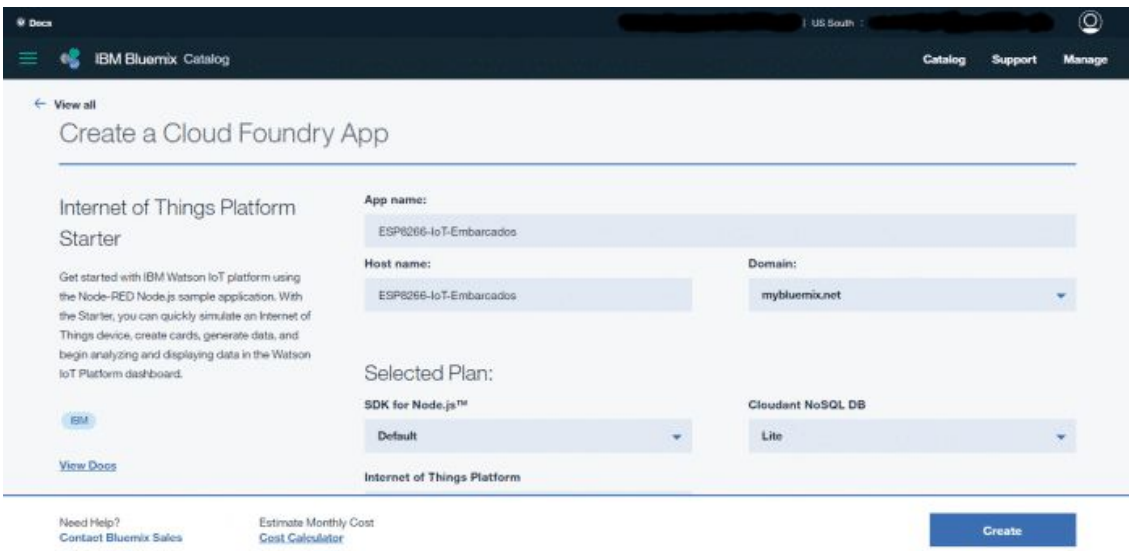


Figura 2 - Criando uma aplicação no Bluemix

Após isso, a aplicação será inicializada. O processo de inicialização da aplicação pode levar alguns minutos.

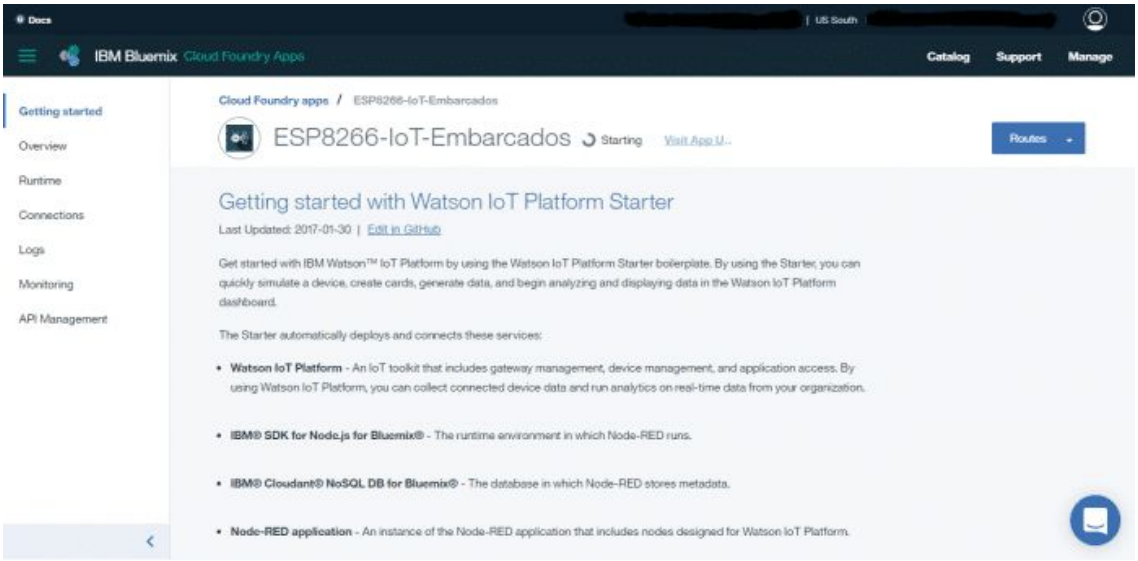


Figura 3 - Iniciando o serviço na plataforma.

Após isso, acesse a página criada para sua aplicação. No meu caso este link. Esta página é para a criação de sua aplicação Node-Red. Ao acessá-la pela primeira vez, você terá que configurá-la com um usuário e senha, fato que fortemente recomendo, pois a página é pública e aberta a toda a internet. Depois disso, você poderá acessar o editor e criar sua aplicação. Um pequeno exemplo é criado quando começamos, mostrado na Figura 4.

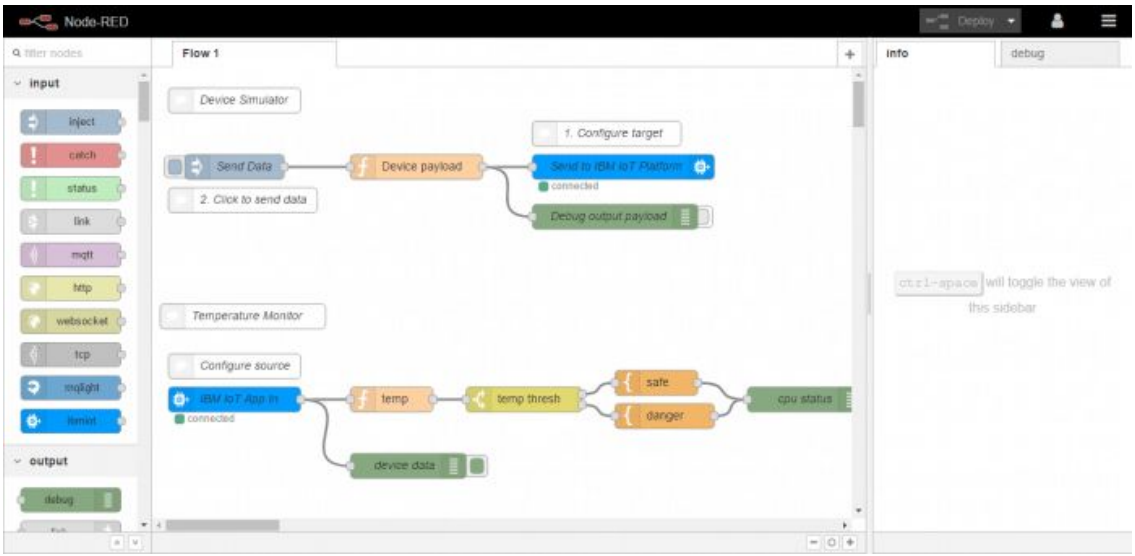


Figura 4 - Aplicação NodeRed

Aplicação embarcada

Não vou entrar no mérito da aplicação embarcada. Mas deixo aqui todo o código fonte usado.

Basicamente, a conexão com a cloud é feita com MQTT, um protocolo já vastamente discutido e cheio de exemplos aqui no Embarcados.

Para o funcionamento do código é necessário adicionar a biblioteca “PubSubClient” em Library Manager.

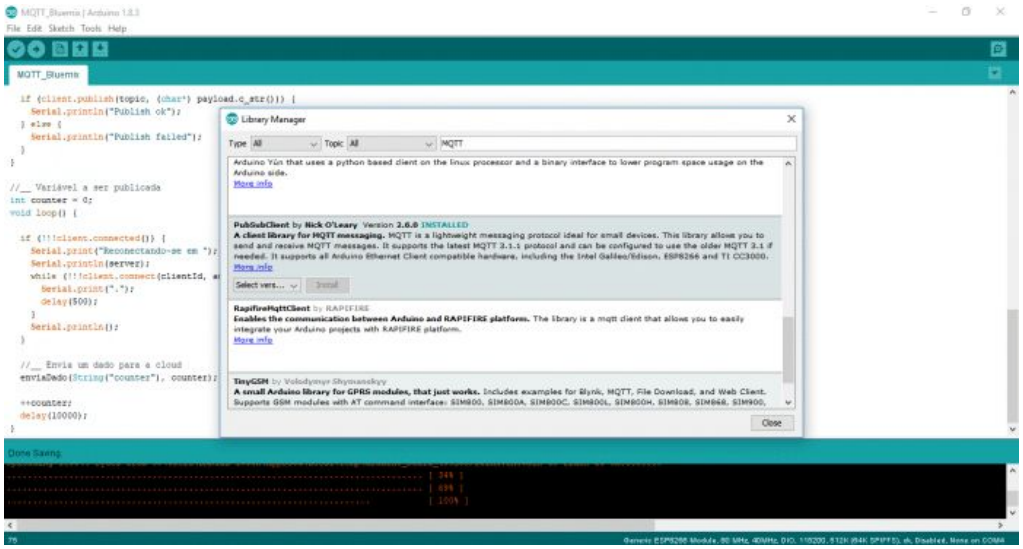


Figura 5 - Library Manager do Arduino

Segue o código. Modifique as variáveis necessárias para conectar-se na rede Wifi, e para conectar-se na *cloud*.

```
//__ Aplicação de exemplo do artigo do embarcados
// Conecta-se a uma rede wifi, conecta-se na cloud da IBM via MQTT e envia dados
//
// Daniel Junho - 09/07/2017

#include <ESP8266WiFi.h>
#include <PubSubClient.h>

//__ Informações do WIFI
const char* ssid =      "SSID";
const char* password = "Senha Wifi";

//__ Informações do dispositivo
#define DEVICE_TYPE  "Nos_da_rede"
#define DEVICE_ID    "Dispositivo1"

//__ Informações da conexão com o servidor
#define ORG       "xxxxxx"
#define TOKEN   "xxxxxxxxxxxxxxxxxx"

//__ Variáveis de conexão com o servidor (Não customizaveis)
```

```
char server[]   = ORG ".messaging.internetofthings.ibmcloud.com";
char topic[]    = "iot-2/evt/status/fmt/json";
char authMeth[] = "use-token-auth";
char token[]    = TOKEN;
char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID;

WiFiClient wifiClient;
PubSubClient client(server, 1883, NULL, wifiClient);

//__ Função de setup do arduino
void setup() {
  //__ Inicializa a serial
  Serial.begin(115200);
  Serial.println();
  Serial.print("Conectando-se na rede "); Serial.print(ssid);

  //__ Conecta-se na rede WIFI
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");

  Serial.print("Conectado, endereço de IP: ");
  Serial.println(WiFi.localIP());
}

//__ Envia os dados para a cloud
void enviaDado(String nome_campo, int dado){
  //__ Formata a string que será envia para a cloud (JSON)
  String payload = "{\"d\":{\"" + nome_campo + "\":";
  payload += dado;
  payload += "}}";
  Serial.print("Sending payload: ");
  Serial.println(payload);
  //__ Envia o dado
  if (client.publish(topic, (char*) payload.c_str())) {
    Serial.println("Publish ok");
  } else {
    Serial.println("Publish failed");
  }
}
```

```
//__ Variável a ser publicada
int counter = 0;

//__ Função principal
void loop() {

  //__ Verifica se está conectada a cloud para envio dos dados
  if (!!!client.connected()) {
      //__ Caso não esteja conectada, tenta a conexão
    Serial.print("Reconectando-se em ");
    Serial.println(server);
    while (!!!client.connect(clientId, authMeth, token)) {
      Serial.print(".");
      delay(500);
    }
    Serial.println();
  }

  //__ Envia um dado para a cloud
  enviaDado(String("counter"), counter);

  //__ Incrementa o contador, mudando o valor a ser enviado para a cloud.
  ++counter;

  //__ Faz o envio a cada 10 segundos.
  delay(10000);
}
```

Enviando os dados para a Cloud

Existem duas maneiras de receber seus dados na *Cloud*. Em uma das maneiras os dados são públicos e podem ser acessados por qualquer pessoa, desde que saiba o *device id* do dispositivo. Na outra maneira, os dados são privados, e a única forma de acesso é com credenciais.

Envio dos dados de modo público

A vantagem do envio dos dados em forma pública é que existe uma página para verificar se o seu dispositivo está enviando dados corretamente. Dessa forma, conseguimos validar uma das pontas do nosso sistema, do dispositivo para a nuvem.

Para isso, modifique o *define ORG* para **quickstart** na aplicação do ESP8266. Isso implica que os dados publicados serão enviados para uma organização já conhecida e pública da plataforma. Note que os valores do define *TOKEN* e o método de autenticação não serão validados na cloud, e portanto, seu valor não faz diferença na aplicação.

Depois de carregada sua aplicação no ESP8266, verifique se o mesmo se conectou na sua rede WIFI, observando os dados sendo impressos pela serial. Assim que conectado, o dispositivo passará a enviar os dados para a nuvem. Para validar os dados que chegam na cloud acesse esta página e insira o nome que o dispositivo se registra na cloud, nome definido no define *DEVICE_ID*. Os dados recebidos então serão apresentados na página.

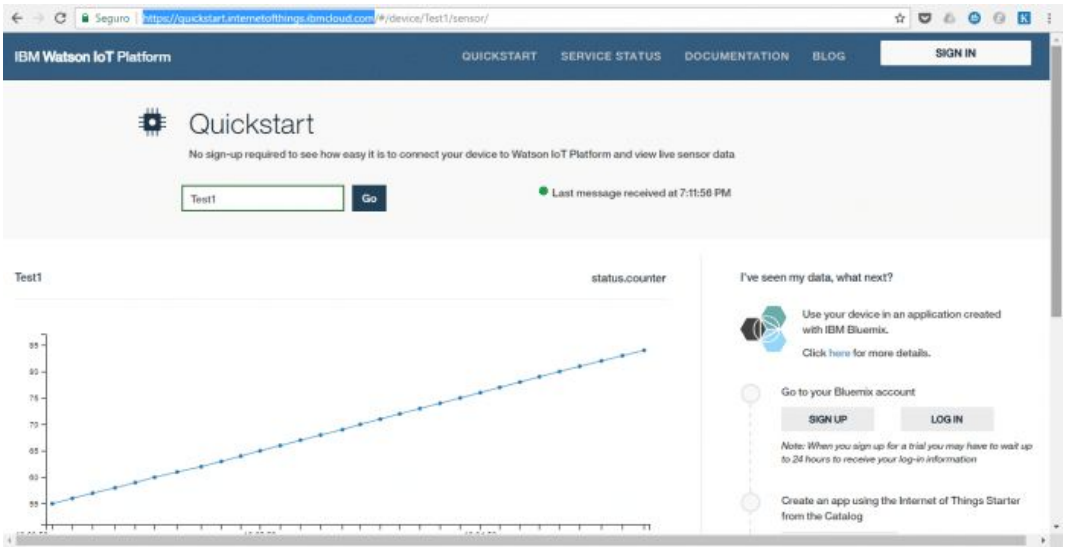


Figura 6 - Página de quickstart de dispositivos conectados

Envio dos dados de modo privado

Depois de validada a conexão entre o *device* e a *cloud*, vamos migrar os dados do nosso dispositivo para um envio de dados seguro e criptografado.

Para isso, acesse o serviço de internet das coisas, um dos serviços criados juntos com nossa aplicação.

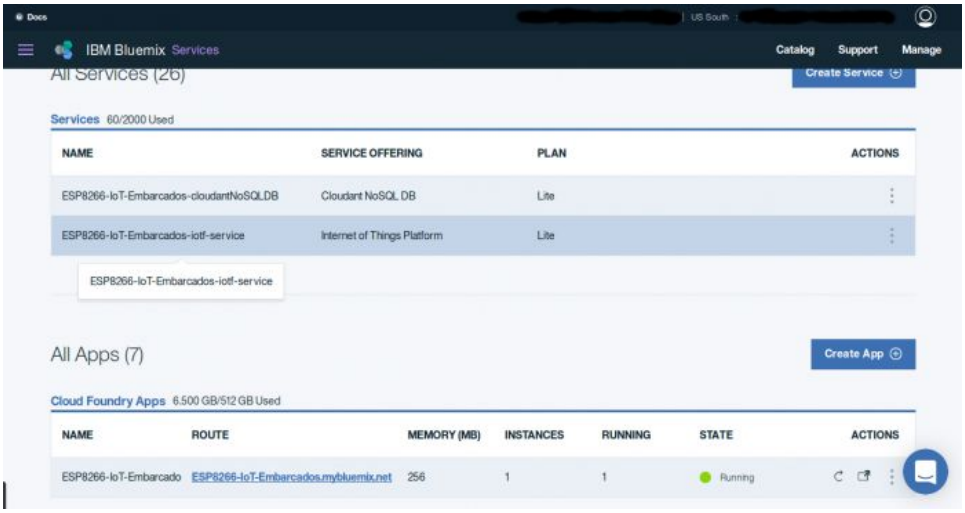


Figura 7 - Serviços rodando na plataforma.

Depois entre na dashboard para adicionar e gerenciar os dispositivos.

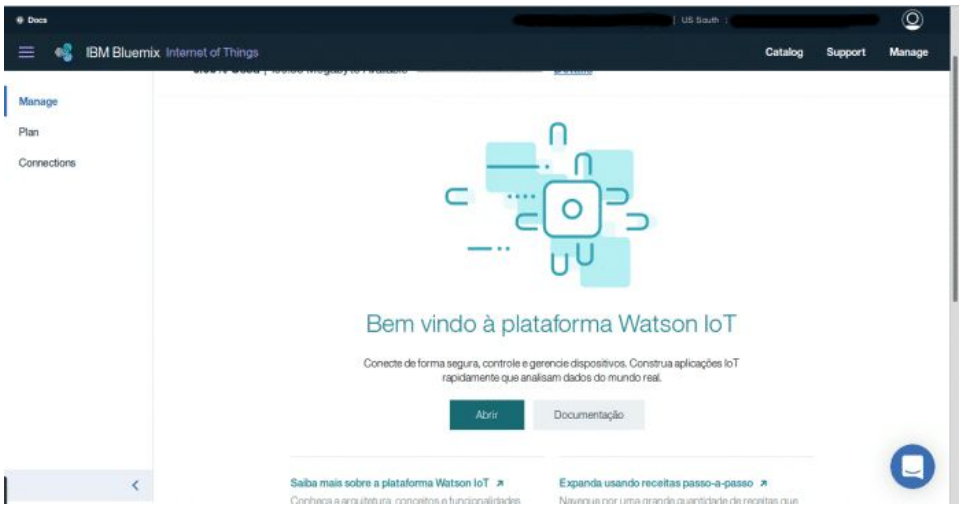


Figura 8 - Dashboard de dispositivos

Dentro dessa dashboard, selecione no menu a opção dispositivos, e depois clique no botão “Incluir dispositivo”.

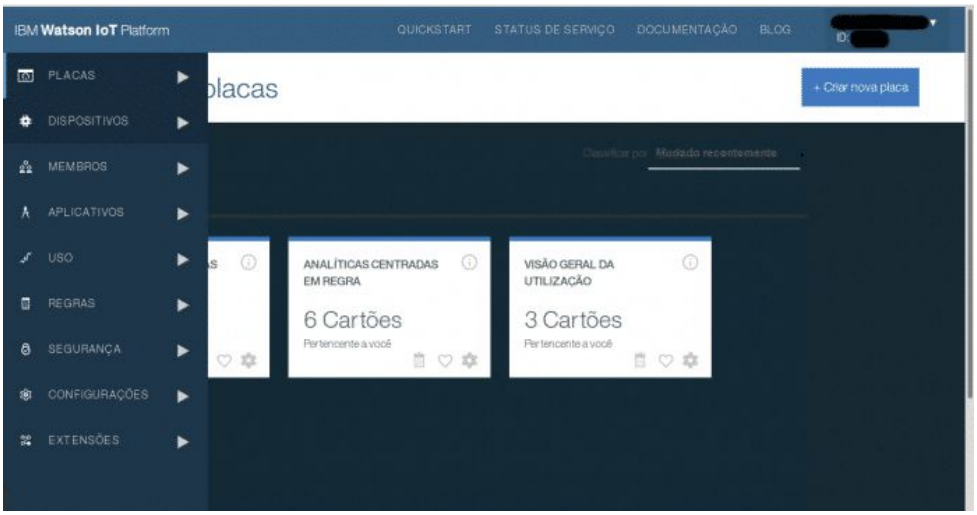


Figura 9 - Cadastro de dispositivos.

Como não existe nenhum tipo de dispositivo ainda, teremos que criar um tipo para a nossa rede. Dessa forma, clique “Criar tipo de dispositivo”, depois clique novamente em “Criar tipo de dispositivo”, e dê um nome a ele.

Figura 10 - Cadastro de tipo de dispositivo

Depois, uma página onde serão apresentados alguns atributos para vincular ao tipo de dispositivo aparecerá. Essas opções são opcionais, adicione-as conforme for o contexto da sua aplicação.

Depois de criado o tipo, iremos finalizar a criação do dispositivo. Dê um nome/ID a ele, e finalize a criação.

No final da criação, seremos questionados sobre o token de autenticação. Sugiro deixar o serviço criar um automaticamente para você. Mas, caso queira, existe a opção de fornecer o token.

Ao final da criação do dispositivo teremos um resumo, onde as informações necessárias para conexão do dispositivo com a cloud são fornecidas. Copie e guarde bem essas informações.

Figura 11 - Resumo de informações do dispositivo.

Agora, copie a informação do “ID da organização” e substitua no valor do define ORG na aplicação embarcada. Depois faça o mesmo com a informação de “token de autenticação” no define TOKEN. Depois, com a informação “Tipo de Dispositivo” no define DEVICE_TYPE. E por último, a informação “ID do dispositivo” no define DEVICE_ID.

Depois carregue a aplicação na placa. Feito isso, e depois de conectada, veremos na dashboard uma indicação que o dispositivo está conectado e enviando dados.

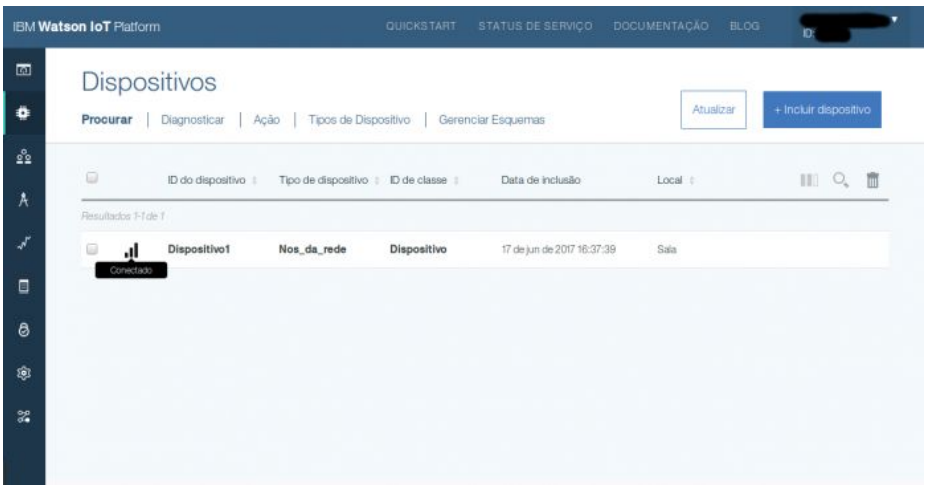


Figura 12 - Status de conexão do dispositivo.

Voltemos a nossa aplicação no NodeRed

A aplicação presente ainda é um exemplo obtido quando criada. Iremos removê-la e criar uma nova. Para isso, clique em qualquer um dos blocos e pressione as teclas “Crtl + A” e depois “delete”.

Na imagem abaixo temos a nossa aplicação NodeRed já formatada para a utilização em forma privada. Para usá-la, basta importar o código para sua aplicação. Para isso, acesse menu > Import > Clipboard e insira o código.

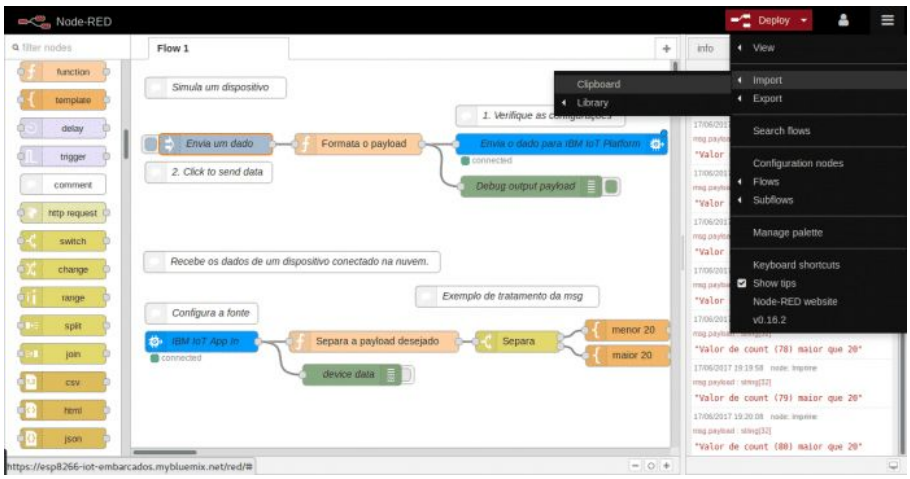


Figura 13 - Importando aplicação para o NodeRed.

Depois de importada, clique em “deploy” e então a aplicação já estará funcional. A aplicação contém alguns comentários explicando seu funcionamento.

Caso queira mudar a aplicação para receber os dados de forma pública, modifique o nó “IBM IoT App In” clicando duas vezes sobre ele para abrir suas configurações. Modifique o campo “Authentication” para “Quickstart” e insira o “Device Type” e o “Device Id” configurados em seu dispositivo.

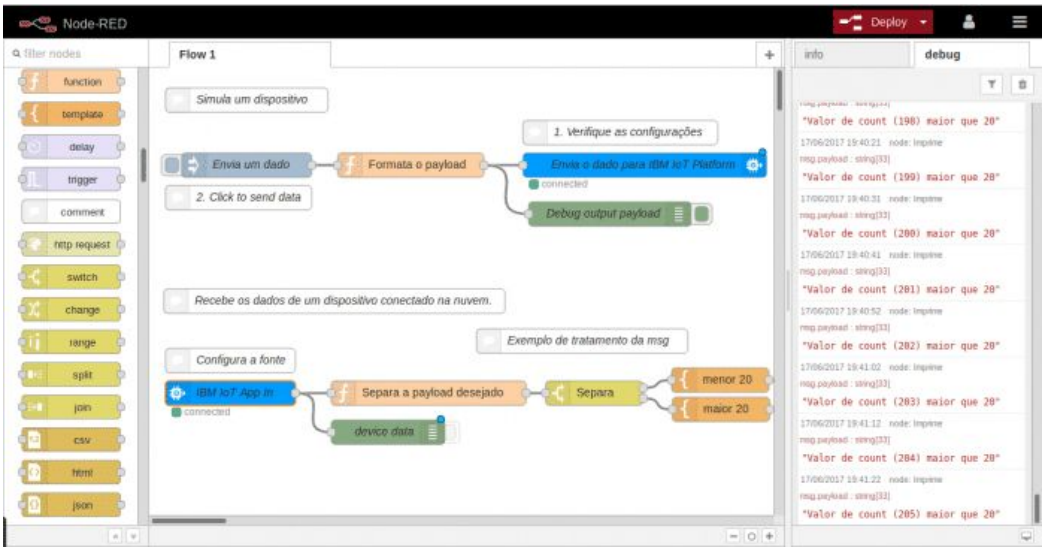


Figura 14 - Aplicação final de NodeRed.

Para aprender um pouco mais sobre nodered, acesse o site oficial. É interessante que com essa aplicação em nodeRed podemos realizar diversas análises sobre os dados recebidos e desencadear ações sobre os sistemas implementados de forma visual. Para os mais *hardcore* ainda se pode inserir códigos em node.js, unindo o útil ao agradável. Uma ferramenta simples e prática que acelera o desenvolvimento de uma aplicação.

Final

Com este artigo, construímos uma aplicação simples que simula um dado (um contador) e envia através de sua conexão com a internet o dado para a cloud da IBM. Tudo isso, de forma simples, sem grandes complicações e com segurança. Use os passos aqui tratados adaptando-os para seu projeto.

Este artigo foi escrito por **Daniel Junho**

Os motes da Internet das Coisas

EMBARCADOS

Por enquanto, para se conectar as coisas à internet, é preciso ter um hardware específico dedicado para isso. Algo que possa ser embutido em um produto ou alguma coisa para adicionar um mínimo de inteligência capaz de conectá-la à Internet.

No mundo da Internet das Coisas, podemos nos deparar com alguns termos até então inéditos que dão uma ideia sobre a arquitetura da aplicação. Enquanto isso, os nomes que estamos dando a esses equipamentos que permitem que as coisas entrem na Internet variam entre gateway, tags, beacons, motes, etc.

Sempre podemos conectar qualquer coisa à Internet, basta "pendurar" um PC na coisa e pum! Ela está na Internet. No entanto, imediatamente após fazermos isso, temos a visão de que esse hardware está sobredimensionado e que podemos fazer o mesmo com um computador muito mais simples como uma Raspberry Pi, Beagleboard, ou até mesmo com um Arduino. Fica claro que quanto menos (espaço, peso, custo, consumo de energia, etc) o hardware agregar ao produto, melhor. A bateria dura mais, as dimensões do produto, a matéria prima consumida e o custo de produção podem ser reduzidos consideravelmente. Dessa forma, os preços praticados também são menores.

E por esse motivo, hoje temos uma necessidade imensa de miniaturizar todo e qualquer hardware, computador, placa ou chip. Um exemplo extremo de miniaturização e integração de processadores e circuitos são os System-on-a-Chip, que têm praticamente um computador inteiro com vários periféricos dentro de um único chip.

Finalmente, a busca pelo menor consumo de energia viabiliza gradativamente uma série de novas aplicações com baterias, pilhas ou coleta de energia do meio. O padrão Wi-Fi não foi projetado levando em consideração o consumo de energia dos dispositivos e portanto se fez necessária a criação de um novo padrão, para atender um mercado diferente.

Muitas soluções com Wi-Fi podem ser encontradas pelo público maker, porém muitas vezes elas devem ser conectadas a grandes baterias e processadores externos para o desenvolvimento da aplicação. Veja este artigo de Pedro Minatel sobre algumas dessas placas e soluções.

Hoje quero apresentar-lhes módulos que suportam pelo menos uma das duas tecnologias alternativas emergentes de IoT, especialmente no quesito de comunicação sem fio de baixíssimo consumo:

- IEEE802.15.4 que é um padrão "parecido" com Wi-Fi, porém muito mais flexível e com menor consumo de energia, que suporta uma série de aplicações, desde sistemas de controle industriais de alta disponibilidade até redes em malha (mesh) extensas - usado em ZigBee, WirelessHART, entre outros;
- Bluetooth LE (BLE, Bluetooth 4.0, Bluetooth Smart) que "não tem nada a ver" com Bluetooth Classic (com o qual estávamos acostumados até então), que simplifica a vida do usuário e diminui drasticamente o consumo de energia da comunicação. Se você comprou seu telefone celular a partir de 2012, existe uma enorme chance do seu aparelho já ter o suporte a essa tecnologia.

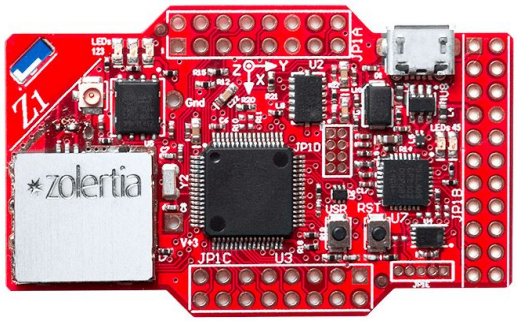
Para se ter uma ideia, os módulos que estamos interessados hoje consomem em média de 5 a 15mA quando recebendo e 4uA quando em stand-by conectado. Em contraste com os módulos Wi-Fi, que em geral consomem pelo menos 50mA quando recebendo e pelo menos 400uA em stand-by conectado.

Os Motes

O objetivo hoje é falar sobre motes. Pelo dicionário britânico Webster, a definição de **mote**, traduzido ao pé da letra, é: um "montinho de terra, de areia, etc". Baseado nessa definição eu adapto para o universo IoT como "o menor pedaço de hardware que suporta a Internet". É basicamente um hardware que você pode customizar e embutir sua aplicação, conectando assim qualquer coisa à Internet e ainda agregando o mínimo de custo possível.
Já existem vários disponíveis comerciais no mercado, no entanto, somente alguns poucos são abertos (open-source). A seguir uma breve descrição de algumas opções disponíveis para hoje.

TelosB / Tmote Sky / Zolertia Z1

TelosB (TPR2400) originalmente com um microcontrolador MSP430, contém um rádio CC2420 (IEEE802.15.4) e antena integrada de 2,4GHz. Com dimensões de 65x31mm, comunica com o computador por USB ou funciona com 2 pilhas alcalinas AA. Diversas variações desse hardware podem ser encontradas, com microcontroladores maiores ou mais sensores integrados.

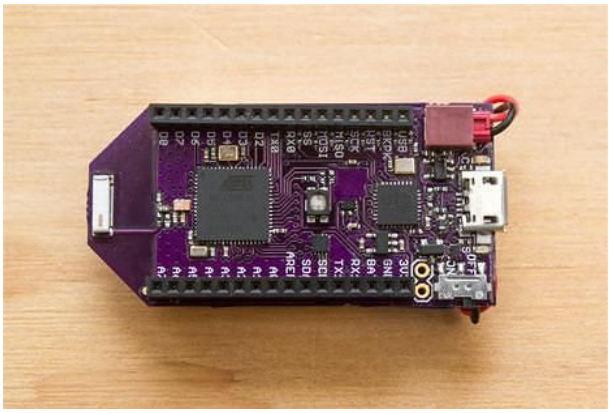


OpenMote-CC2538

O OpenMote-CC2538, usa o CC2538, um System-on-Chip que tem um processador programável com core ARM Cortex-M3 e um rádio IEEE802.15.4. Ele foi feito para ser pino a pino compatível com o *form factor* das placas XBee. Atualmente o OpenMote já suporta os RTOSs Contiki, OpenWSN e FreeRTOS.

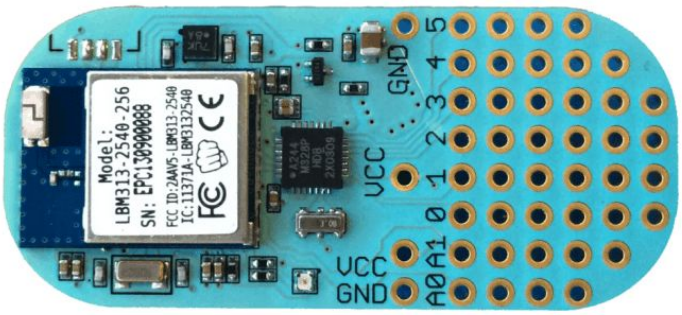
Pinoccio

Com 70x25mm, usa processador ATmega256RFR2 e, portanto, é compatível com Arduino. Tem USB e rádio IEEE802.15.4 e além de antena de 2,4GHz. Vem com uma bateria LiPo recarregável, sensor de temperatura e LED RGB. Tem suporte opcional a Wi-Fi através de uma placa de expansão.



LightBlue Bean

Tem dimensões de 45x20mm com acelerômetro, LED RGB e antena de 2,4GHz integrados. Vem com um rádio CC254x (Bluetooth LE) e um ATMega328. É compatível com Arduíno e já vem com uma bateria CR2032. Acompanha App de controle para Smartphone.



Existem vários outros motes disponíveis no mercado, além de alguns outros projetos Kickstarter.

No Brasil: momote001

No Brasil, o primeiro mote é o momote001. Ele é baseado no System-on-a-Chip CC2650 que inclui um microcontrolador ARM programável com 20KB de RAM e 128KB de flash, e um rádio que suporta os dois padrões: IEEE802.15.4 e Bluetooth Low Energy (BLE).

Pode ser usado para comunicar diretamente com o celular, para programação e atualização do software. É menor que uma foto 3x4cm, possui antena integrada de 2,4GHz e pode ser alimentado por pilhas alcalinas ou bateria CR2032.

Por ser extremamente eficiente no consumo de energia, dependendo da aplicação, é possível fazer com que a bateria dure vários anos! Seu hardware é completamente open source e todos os arquivos já podem ser consultados neste link, no site do fabricante ou no Github do projeto.

Muitos Motes ainda serão criados

Dessa forma, concluímos que os motes vieram para ficar e percebe-se de uma vez por todas que a crescente demanda por integração, miniaturização e eficiência faz com que os motes sejam, de fato, a plataforma que vai possibilitar com que tudo seja conectado em breve. Enquanto isso estamos aguardando ansiosamente para ver o que o futuro guarda para a comunidade maker e para o mundo IoT!

Este artigo foi escrito por **Marco Casaroli**

Plataformas e módulos IoT Internet das Coisas - Parte I

EMBARCADOS

O termo IoT “Internet das Coisas” vem do inglês "Internet of Things" e faz menção à atual revolução tecnológica aonde cada vez mais itens usados em nosso cotidiano se conectam à internet. Já é possível observar o surgimento de eletrodomésticos, óculos, relógios, meios de transporte e até mesmo itens como tênis, luminárias e interfones, dentre vários outros, com funcionalidades que envolvam a conexão com rede e internet, podendo ser acessados de computadores e smartphones.

Módulos e Plataformas para IoT (Internet of Things)

Seguindo a tendência dessa nova onda de Internet of Things, começamos a observar o surgimento gradativamente maior de placas e módulos voltados para essa aplicação. Sistemas com Linux embutido (ou com suporte a algum RTOS), conectividade com WiFi e alguns GPIOs disponíveis são algumas características quase constantes, salvo algumas exceções. E nessa nova tendência fatores como tamanho reduzido e baixo consumo elétrico são mandatórios para o sucesso da aplicação.

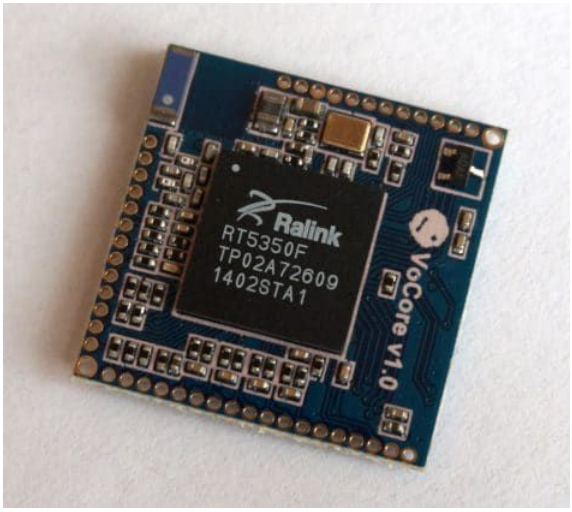
VoCore

Um dos destaques de módulos para Internet of Things fica para o VoCore, também chamado de “O computador com Linux do tamanho de uma moeda com WiFi”. O projeto VoCore começou através de crowdfunding por meio do portal IndieGogo (disponível em https://www.indiegogo.com/projects/vocore-a-coin-sized-linux-computer-with-wifi), e partindo de uma premissa inicial de $ 6.000 para viabilizar o projeto, alcançou a marca de $ 116.237. Ainda bem, é um projeto sério que mantém feedback constante aos “patrocinadores”, e agora possui seu site próprio Vonger.

Basicamente, a plataforma VoCore consiste em:

- SoC Ralink RT5350, de arquitetura MIPS e frequência de 360MHz, com WiFi embutido;
- Sistema Operacional Linux OpenWRT;
- 8MB de memória Flash SPI;
- 32MB de memória SDRAM;
- Interfaces como 10/100M Ethernet, USB, UART, I2C, I2S, PCM, JTAG e 20 GPIOs;
- Mede 25mm x 25mm;
- Preço de $ 20,00 pelo crowdfunding no IndeGogo;
- Consumo de energia: Cerca de 200mA a 210mA (durante eventos como Ping).

O VoCore está disponível em uma placa simples, contendo os elementos essenciais, como mostrado abaixo:

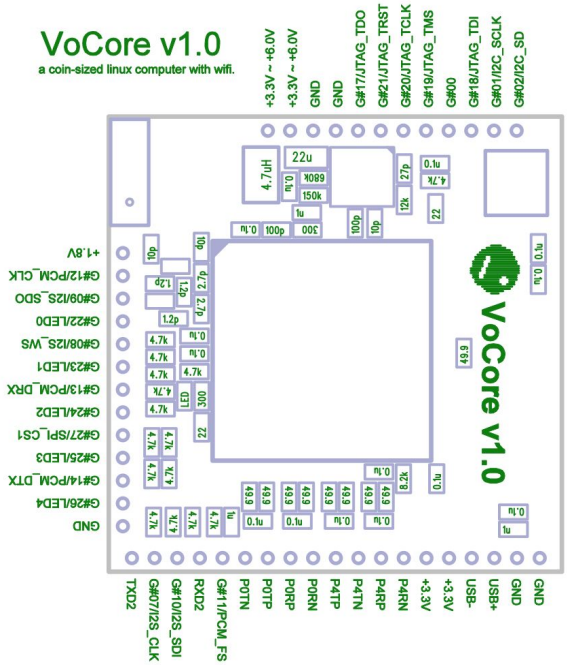


VoCore

No próprio site do crowdfunding, o projetista já apresenta algumas possibilidades de uso com o VoCore, tais como:

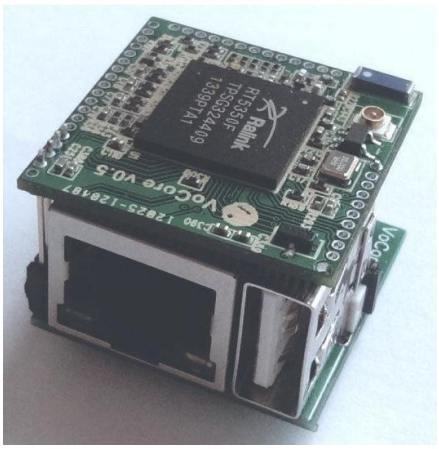
- Robô móvel controlado com câmera;
- Roteador VPN portátil;
- Alto-falante portátil;
- Conversor WiFi-TTL para interface com microcontroladores como o Arduino.

Para se ter uma noção da placa em geral e de seus pinos GPIO, de comunicação e alimentação, segue o esquemático adiante.



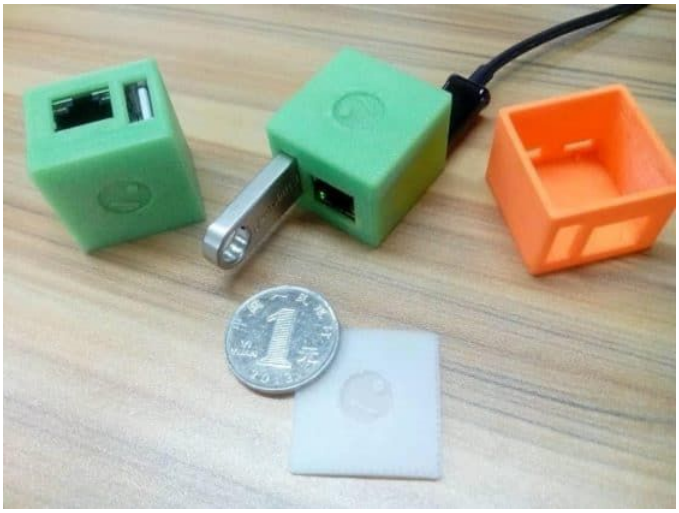
VoCore

E também há a opção de pedir o VoCore com o Dock, que possui USB Host e Ethernet, tornando possível o acesso físico aos componentes já presentes para conectividade com rede cabeada e dispositivos USB tais como câmeras, pendrives, modems 3G, impressoras, scanners e outros.



VoCore com Dock

O projetista já está fazendo até mesmo cases para o VoCore + Dock, como mostrado abaixo. Reparem no tamanho comparado à uma moeda.



Cases para VoCore + Dock

AsiaRF AWM002

Outra alternativa contendo um SoC com suporte à WiFi e capaz de executar o sistema operacional Linux é o AsiaRF, disponível nas versões AWM002 e AWM003, desenvolvido diretamente pela empresa AsiaRF.

AsiaRF

Apesar de ser uma empresa já fundamentada com 10 anos de experiência em WiFi, o projeto AsiaRF AWM002 foi submetido à uma campanha de crowdfunding no site IndieGogo, com um preço base para acesso ao dispositivo de $15.

Basicamente, a intenção da AsiaRF era obter fundos para conseguir produzir um lote de placas para validação frente às certificações FCC e CE, o que eles conseguiram, dado o fato de que a campanha arrecadou $ 7.386 dólares, de um objetivo inicial de $ 6.000 dólares.

Assim como o VoCore, também há a opção de um Dock que permite o acesso às conexões Ethernet e USB Host do dispositivo.

Suas características principais são:

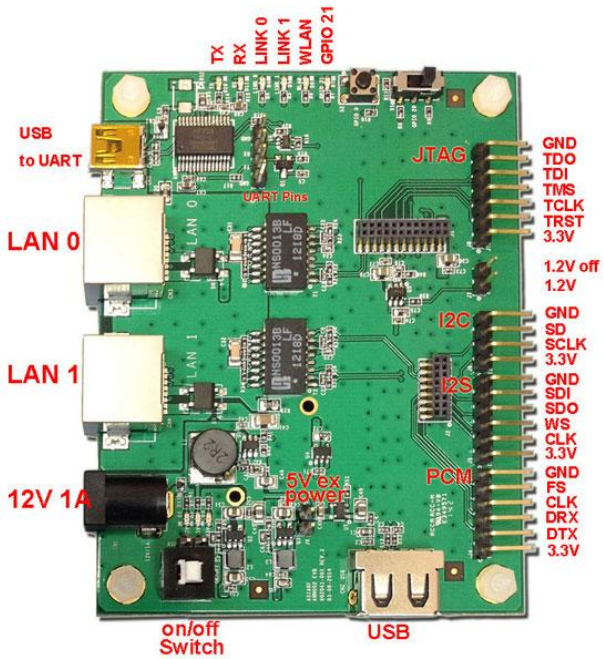
Dock

- SoC Ralink RT5350 (basicamente o mesmo do VoCore), rodando a 360 MHz e com suporte a WiFi 802.11n
    - Versão AWM002 – 32 MB de RAM;
    - Versão AWM003 – 64 MB de RAM;
- Linux OpenWRT;
- Armazenamento: 8 MB de memória NAND Flash;
- Suporta criptografias WEP, WPA, WPA2 com TKIP, AES e WPS;
- Funciona como Cliente WiFi e faz conexões Ponto-a-Ponto;
- Alimentação 3.3V;
- Dimensões: 25mm x 35mm;
- Preço $ 15,00 (IndieGogo - modelo AWM002).

E assim como na campanha do VoCore, a página da campanha do AsiaRF no IndieGogo cita algumas possibilidades com o módulo:

- Câmera IP com WiFi;
- WiFi Walkie-Talkie;
- Alto-Falante WiFi;
- Estação 3G;
- Servidor de Impressão;
- Servidor de armazenamento de dados na nuvem.

Existe a opção de adquirir o módulo com uma placa base, o chamado Evaluation Kit, em que a placa permite acessar diretamente os GPIOs, conexões USB e Ethernet presentes no AsiaRF AWM002 ou AWM003.



Evaluation Kit

Este artigo foi escrito por **André Curvello**

Publicado originalmente no Embarcados, no dia 19/09/2014: link para o artigo original, sob a Licença Creative Commons Atribuição-CompartilhaIgual 4.0 Internacional.

Plataformas e módulos IoT
Internet das Coisas - Parte I

EMBARCADOS

WRTnode

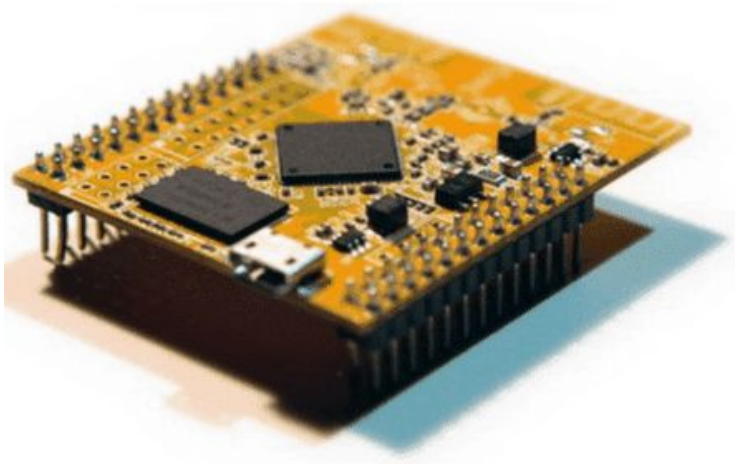


Figura 1 - WRTnode

Considerado uma adaptação melhorada do VoCore e do AsiaRF, pelo fato de possuir um SoC com frequência de operação maior e pinos com espaçamento 2.54mm, o WRTnode é outra obra de arte chinesa. Diferentemente dos outros dois anteriores, não foi fruto de crowdfunding.

Suas características principais são:

- SoC MediaTek MY7620N MIPS de 600 MHz;
- Memória RAM de 64 DDR2;
- 16 de memória SPI Flash;
- 23 GPIOs, conexões JTAG, UART, USB Host 2.0 e USB micro para device;
- Dimensões: 45mm x 50mm;
- Linux OpenWRT;
- Site do projeto: WRTnode;
- $ 25,00 no SeeedStudio com placa-base de desenvolvimento.

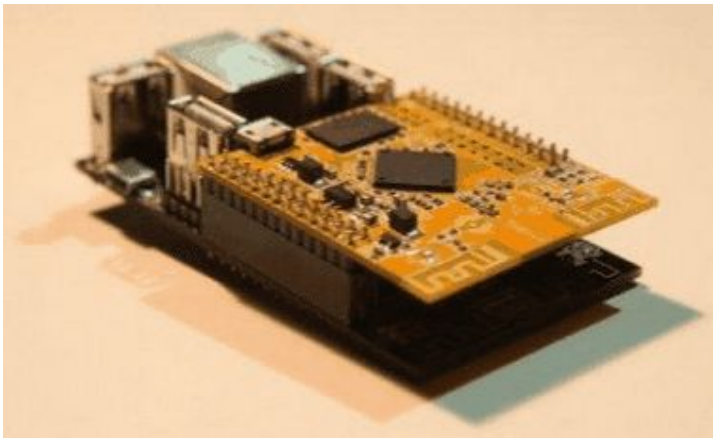


Figura 2 - WRTNode

A empresa responsável pretende fornecer também toda uma gama de soluções complementares ao módulo WRTnode, incluindo uma SDK voltada para IoT usando REST, além de um shield padrão que destaca as conexões USB e rede ethernet, como mostrado na Figura 2, um kit de robótica como mostrado na Figura 3, além de módulos chamados UIXO, que permitem incorporar periféricos como sensores de temperatura e umidade como o DHT11, e sensores de distância como o HC-SR04. A empresa também destaca que futuramente fornecerão um kit para quadcópteros. Além disso, todo o projeto é opensource e está disponível no GitHub.

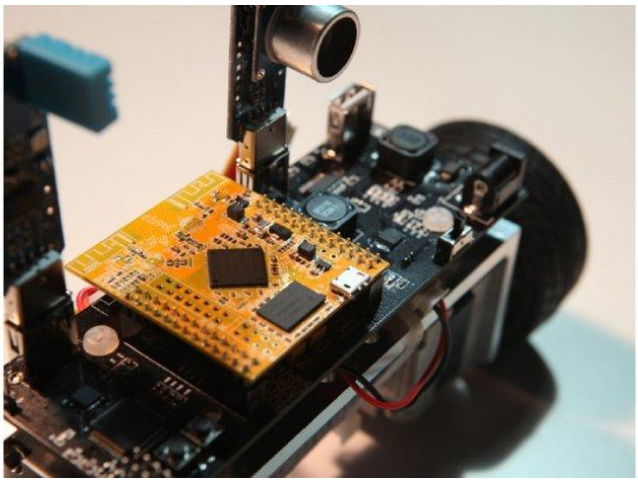


Figura 3 - Kit de robótica para WRTnode

No site da empresa e em outros links, percebe-se o destaque que os criadores dão com relação à visão computacional, pelo fato de o módulo já vir com suporte de fábrica à OpenCV 2.4.8 e aplicações de demonstração usando OpenCV. E também destacam a presença do gcc e bin-utils para MIPS no firmware padrão.

Demonstrando as funcionalidades com a biblioteca OpenCV para o módulo, segue um vídeo de demonstração de um braço robótico capturando moedas, controlado pela WRTnode.

ESP8266

É um módulo que apareceu com grande surpresa no mercado recentemente, tanto pelo seu tamanho, e muito mais pelo seu preço (cerca de $ 5,00 na AliExpress).

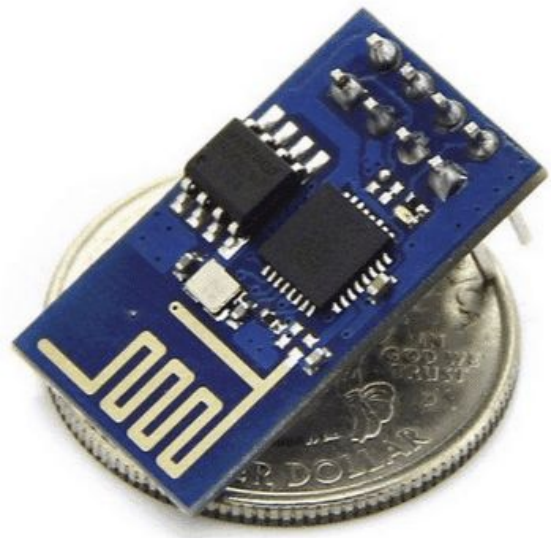


Figura 4 - ESP8266

Ao comprar o produto, o vendedor envia um datasheet (em chinês), contento guia de instrução, uso e configuração do módulo. Basicamente, ele faz uso de comandos seriais AT para configurar os modos de operação WiFi, ver status da conexão, realizar as conexões, dentre outras funcionalidades, o que o torna uma mão na roda em projetos microcontrolados dado a facilidade de então incorporar conectividade WiFi ao projeto, com um preço bem baixo, bastando o uso dos tradicionais RX/TX!

Basicamente o ESP8266 é composto por:

- SoC ESP8266 32-bits com a suporte a WiFI 802.11 b/g/n e interfaces SDIO, SPI, UART e I2S;
- WiFi com suporte a criptografias WEP, TKIP, AES e conexão direta ponto-a-ponto;
- Pinos de conexões: GND, VCC, RX e TX, o restante é NC;
- Dimensões: 21mm x 11mm;
- Consumo em standby: < 1,0mW.

Link para compra no AliExpress.

Outros Módulos IoT

Encerra-se aqui a Parte 2 de Módulos IoT, voltados para Internet das Coisas, ou “Internet of Things”, em inglês.

Como percebemos, a tendência é que a maioria das placas sejam módulos pequenos, com destaque para conexão sem fio WiFi 802.11, baixo consumo de energia, e baixo preço.

Há mais alguns colegas nessa onda, como o Arduino Yún e o Intel Edison, que dados os seus “fatores de impacto”, são abordados em artigos exclusivos. Confira meu artigo sobre o Intel Edison no Embarcados.

Este artigo foi escrito por **André Curvello**

Considerações Finais

Chegamos ao final do nosso ebook.

Caso você tenha alguma dúvida, não deixe ela sem resposta. Você pode entrar em contato com o autor através da seção de comentários do artigo, ou então participar da nossa comunidade, interagindo com outros profissionais da área:
Comunidade Embarcados no Facebook
Comunidade Embarcados no Telegram

Caso você tenha encontrado algum problema no material ou tenha alguma sugestão, por favor, entre em contato conosco. Sua opinião é muito importante para nós:

contato@embarcados.com.br

Siga o Embarcados na Redes Sociais

facebook.com/osembarcados/

instagram.com/portalembarcados/

youtube.com/embarcadostv/

linkedin.com/company/embarcados/

twitter.com/embarcados