

FIAP GRADUAÇÃO

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Arquiteturas Disruptivas, IoT, IA e Big Data

PROF. ANTONIO SELVATICI

SHORT BIO



É engenheiro eletrônico formado pelo Instituto Tecnológico de Aeronáutica (ITA), com mestrado e doutorado pela Escola Politécnica (USP), e passagem pela Georgia Institute of Technology em Atlanta (EUA). Desde 2002, atua na indústria em projetos nas áreas de robótica, visão computacional e internet das coisas, aliando teoria e prática no desenvolvimento de soluções baseadas em Machine Learning, processamento paralelo e modelos probabilísticos. Desenvolveu projetos para Avibrás, IPT, CESP e Systax.

PROF. ANTONIO SELVATICI

profantonio.selvatici@fiap.com.br

INTERNET DAS COISAS

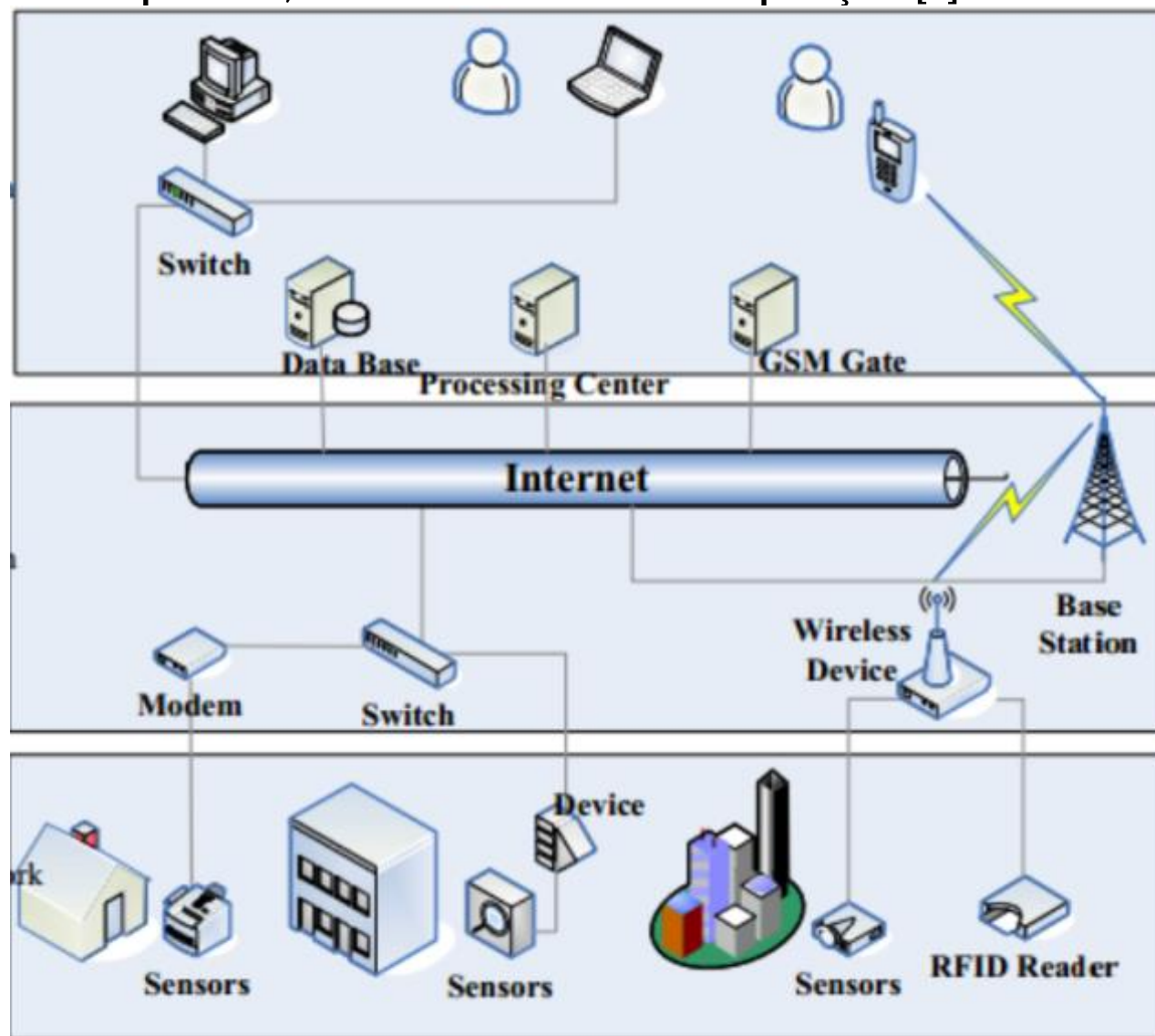
Arquitetura básica das aplicações de IoT

Como se relacionam os dispositivos, a internet e os usuários das aplicações [1]

Rede ou
Camada de
**Aplicações e
Serviços**

Rede ou
Camada de
Transmissão

Rede ou
Camada de
Sensores



Rede de Sensores

Coleta de dados, acionamento de dispositivos, comunicação local

- Rede de comunicação que interliga os diferentes objetos conectados
- É o “diferencial” da internet das coisas
 - onde atuam as tecnologias habilitadoras da IoT
- Comparada à “pele” da IoT, por onde ocorrem as trocas de informação com o mundo
 - Captura de dados por sensores
 - Execução de ações por atuadores
- Objetos sem conectividade própria são rastreados usando RFID ou outra forma de identificação
- Em geral, os objetos se comunicam em uma rede local (**WSN – Wireless Sensor Networks**), que por sua vez se comunica com a internet através de gateways ou bridges
- Redes de comunicação de objetos muitas vezes usam tecnologias alternativas ao WiFi, como Bluetooth, Zigbee, LoRaWan

Tecnologias Habilitadoras

Permitem a integração dos objetos e ambientes à internet

- **Identificação de objetos**
 - Tags de RFID/NFC
 - Código de Barras, Data Matrix Code
 - Reconhecimento de Imagens, etc.
- **Formação de redes de comunicações com/entre objetos (WSN – Wireless Sensor Network)**
 - Zigbee, 6LoPan, Bluetooth, GSM Data, RS-485, WiFi
- **Computação Ubíqua**
 - Hardware proprietário, Arduíno, Raspberry Pi, Edson, Beagle Bone
- **Interação com o ambiente**
 - Monitoramento de variáveis ambientais
 - Sensores de temperatura, luminosidade, MEMS, etc.
 - GPS e localização física em rede sem fio
 - Execução de tarefas por meio de atuadores
 - Acionadores, interruptores, motores

Rede de Transmissão

Integra a rede de sensores à internet

- Sistema nervoso central da IoT, tendo o papel de transmitir e processar dados
- Corresponde à infraestrutura de comunicação que permite a interconexão de objetos, aplicações e seus usuários
- Integra os objetos inteligentes à internet, convertendo os protocolos de transporte próprios das redes de objetos ao TCP/IP
- Os servidores da rede podem se comunicar com os dispositivos conectados através de diversos protocolos de aplicação que costumam funcionar bem para comandos simples
 - **HTTP**: Hyper-Text transfer Protocol (cabeçalho mais complexo)
 - **MQTT**: MQ Telemetry Transport
 - **CoAP**: Constrained Application Protocol
- As mensagens da IoT em geral são formatadas de acordo com regras que permitem o uso de interpretadores padrão:
 - **XML**: campos do documento definidos através de markups
 - **HTML**: subconjunto do XML usado para páginas web
 - **JSON**: documento definido como um objeto JavaScript

Exemplo de requisição HTTP com conteúdo JSON

■ Requisição

- `POST /request HTTP/1.1`
- `Accept: application/jsonrequest`
- `Content-Length: 72`
- `Content-Type: application/jsonrequest`
- `Host: json.penzance.org`
- `{"user":"doctoravatar@penzance.com","forecast":7,"t":"v1Ij","zip":94089}`

■ Resposta

- `HTTP/1.1 200 OK`
- `Content-Type: application/jsonrequest`
- `Content-Length: 15`
- `{"status":true}`

Rede de aplicação

Camada de provimento de serviços online e interação com o usuário, que se comunicam com os dispositivos de IoT através da rede de transmissão

- Formada pelos aplicativos de usuário final, bem como pelos serviços que permitem um melhor gerenciamento dos dispositivos e aplicações de IoT
- Expõe API's para acesso aos dados dos sensores e controle dos dispositivos
- Atualmente, a regra é usar *computação em nuvem* para prover os serviços de rede
 - **Gartner [2]:** Estilo de computação na qual recursos de TI escaláveis e elásticos são oferecidos como serviço usando tecnologias da internet.

Árvore de tecnologias e aplicações da IoT

Fonte: www.internet-of-things-research.eu [5]

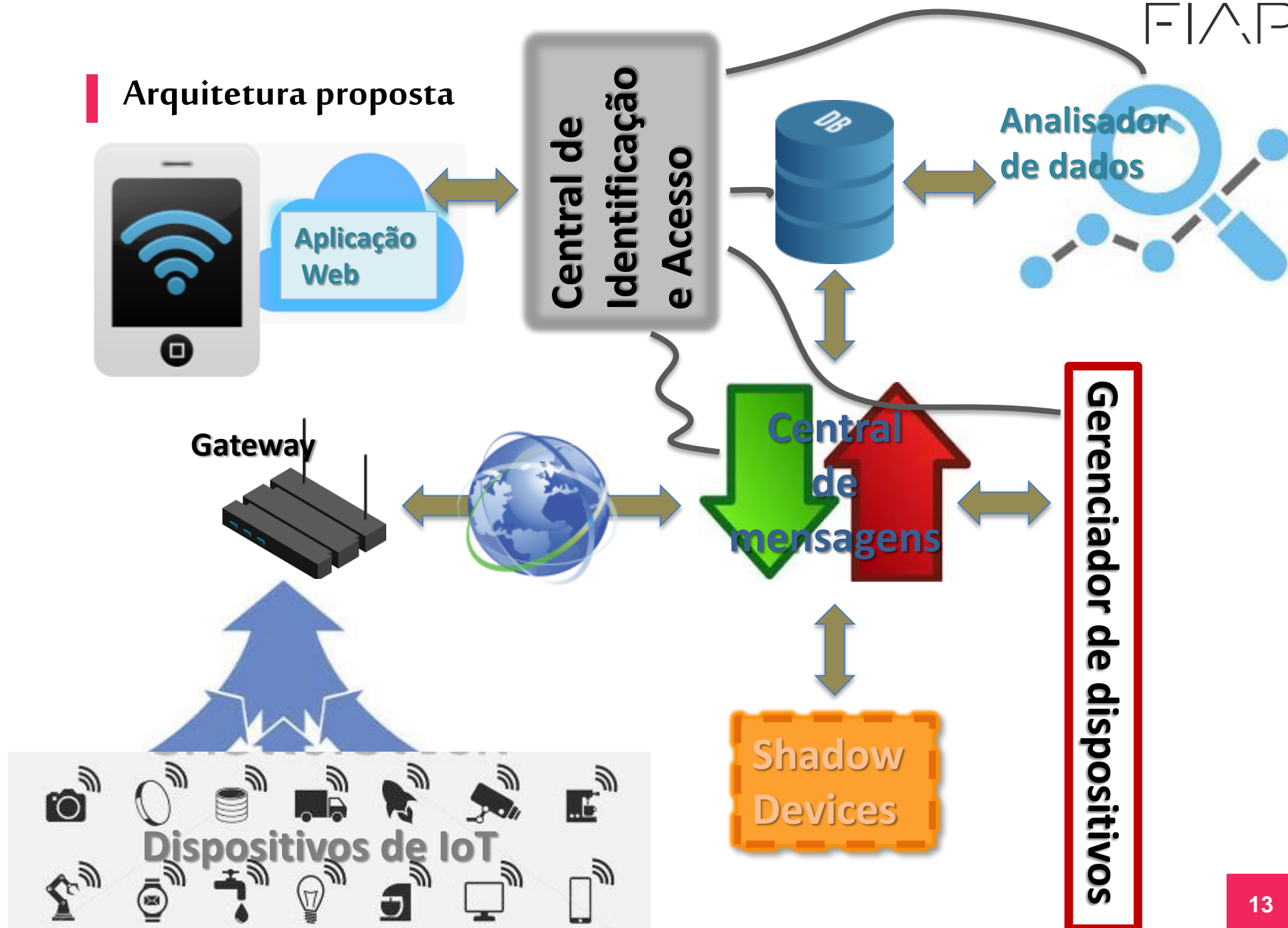
- Tecnologias na raiz do IoT existem e abundam
- Embora seja um campo a explorar, existem aplicações bem definidas para a IoT
- O que falta para a IoT? Integração...
 - ...entre as tecnologias habilitadoras
 - ...entre as tecnologias e os domínios de aplicação
 - ...principalmente, entre os diferentes elementos das áreas de aplicação



Arquitetura básica de implantação de IoT

- **Arquitetura de implantação** aqui fornecida é um desenho padrão para inspirar projetos reais a serem implementados, incluindo apenas os elementos fundamentais para a conectividade, sem detalhar soluções para problemas acessórios
- IoT envolve tantas tecnologias diferentes, permitindo tantas combinações diferentes, que projetos na área tendem a se tornar “Frankensteins”
 - Interoperabilidade: facilita a compatibilidade entre diferentes projetos de IoT
 - Modularidade: define módulos que podem ser criados separadamente ou ainda usados “off-the-shelf”
 - Compartilhamento de melhorias entre diferentes implementações

Arquitetura proposta



■ Dispositivos de IoT (devices)

- Permitem a interação com o ambiente ao seu redor, seja capturando dados de sensores como executando comandos através de seus atuadores
- Cada funcionalidade no dispositivo pode ser considerado uma **Aplicação** (Endpoint Application)
 - Sensores de temperatura e luminosidade são aplicações diferentes dentro da mesma placa Arduino, por exemplo
 - Cada aplicação deve ser univocamente endereçável
 - Contexto embutido em vários padrões de comunicação como USB
- **Shadow Devices:** dispositivos virtuais que emulam o comportamento dos dispositivos reais enquanto não é possível estabelecer comunicação com eles
 - Perda de comunicação ou polling devices
 - O dispositivo real sincroniza seu estado com o do shadow device

Central de mensagens (Message Broker)

- Gerencia filas de mensagens que chegam de dispositivos ou são destinadas a eles
 - Em geral, são capazes de identificar e autenticar dispositivos, mas não têm controle sobre quem acessa cada mensagem específica
- IBM IoT Framework:
 - Mensagens oriundas de dispositivos são “eventos”
 - Mensagens destinadas a dispositivos são “comandos”
- Apesar de poder trabalhar com HTTP, em geral fazem uso de protocolos de aplicação mais simples. Protocolos usados:
 - MQTT
 - WebSocket

Gerenciador de dispositivos

- Cadastra novos dispositivos e aplicações
- Decide se um dispositivo anunciado pode ou não ser acrescentado à rede
- Envia comandos de gerenciamento, como:
 - Inicialização e reinicialização
 - Desligamento
 - Atualização de firmware

Banco de dados e analisador de dados

- Armazena os dados vindos da aplicação, bem como os comandos que vão para os dispositivos
- Bancos de dados NoSQL são mais indicados, uma vez que a natureza das informações que são trocadas pelos dispositivos de IoT é muito diversa, podendo variar com o tempo
 - Ex: suponha que eu tenha uma tabela com os campos “Deviceld”, “Temperatura” e “Umidade”, mas tenha acabado de plugar um sensor de luminosidade...
- Faz sentido que os dados sejam monitorados por aplicações de análise de dados para um melhor aproveitamento

Gateway

- Realiza a conversão de protocolo entre os dispositivos de IoT e a central de distribuição de mensagens
- O uso do IPv6 pelos dispositivos facilita a resolução do endereçamento do dispositivo, mas não é suficiente para resolver as mensagens específicas da aplicação
- Gerenciamento de múltiplos protocolos, especialmente com LAN's, PAN's e HAN's : Zigbee, Bluetooth, Wi-Fi, Thread/6LoWPAN, etc.
- Serviço de dados em redes WAN: uso gateways compartilhados, podendo ou não ser pagos
 - Dados móveis (GPRS, 2G, 3G, HDSP, LTE, 5G ...)
 - LoRaWAN, SigFox

Gateway IoT

Integra objetos à internet, convertendo protocolos e servindo de ponte para serviços na rede

- Como se conectar a serviços disponíveis na internet, como Webservices?
 - **Paradigma cliente/servidor:** um programa cliente faz requisições remotas a um programa servidor que “escuta” em um endereço IP e uma porta, por exemplo:



- **Paradigma Pub/Sub:** um programa publica uma mensagem em um tópico, enquanto os programas subscritos a esse tópico recebem essa mensagem

Como construir gateways para nossas aplicações de IoT?

- Os gateways devem ter duas pontas:
 - A ponta da rede de objetos, onde há a comunicação com o Arduino ou outros dispositivos através da porta serial ou módulos de comunicação sem fio
 - A ponta da rede de transmissão, onde mensagens são trocadas através da internet
 - Construiremos o gateway usando Node.js e Node-Red
- **Node.js**
 - Ambiente de execução em tempo real para a linguagem JavaScript, com foco na execução de programas no lado do servidor
 - Uso do motor Javascript V8 do Chrome, desenvolvido em C++, em conjunto com uma biblioteca de tratamento de eventos
 - Possui um conjunto de extensões elaboradas e melhoradas por contribuintes do mundo todo
- **Papel na IoT:** prover, de modo simples e eficiente, com alta disponibilidade, serviços demandando pouco esforço computacional
 - A simplicidade do servidor permite seu uso em dispositivos tipo Mini-PC mais simples, como Raspberry Pi, Beagle Bone e até Android

Diferenças de NodeJS para um servidor tradicional

- Não é focado em um único tipo de serviço, ou tipos específicos de serviço (HTTP, banco de dados, FTP, etc.)
- Servidores tradicionais usam uma thread para cada requisição
 - Servidores são projetados para aplicações pesadas
 - Threads ocupam muita memória e outros recursos do sistema
 - 2 MB por thread → 1GB = 500 requisições simultâneas, com overhead do load balancer
- Uso de processamento de eventos assíncronos acionados pelas notificações do sistema operacional (signals e file descriptors)
 - Cada requisição ao servidor gera um evento
 - A mesma thread processa vários eventos
 - Capaz de processar milhares de requisições por segundo com pouco uso de memória

Tabela comparativa com outras linguagens de programação

Linguagem	Desempenho de execução	Requisito de Memória	Abstração de código	Tamanho do bytecode
Assembly	Muito Alto	M Baixo	Baixo	M Baixo
C	Muito Alto	Baixo	Médio	Baixo
C++	Muito Alto	Baixo	Médio/Alto	Depende*
Java	Alto	Alto	Alto	Médio/Alto
Python	Baixo	Alto	Alto	Baixo
Node.js	Médio	Médio	Alto	Baixo

Baseado em <http://benchmarksgame.alioth.debian.org/>

*Usar a STL aumenta muito o tamanho do código

■ Programação assíncrona em Node.JS

- Programar Node.JS *is all about* processar eventos de forma assíncrona
- Grande parte dos métodos de objetos que vamos usar recebem como argumento a função que irá tratar algum evento relacionado
- A dificuldade está mais em mudar o paradigma de programação, de síncrono (ou seja, código executado em sequência) para assíncrono, sem bloqueio do fluxo do programa
- Por exemplo, podemos agendar uma função para ser executado daqui a 5s, mas não é usual interrompermos o programa por 5s

■ Node-Red [nodered.org]

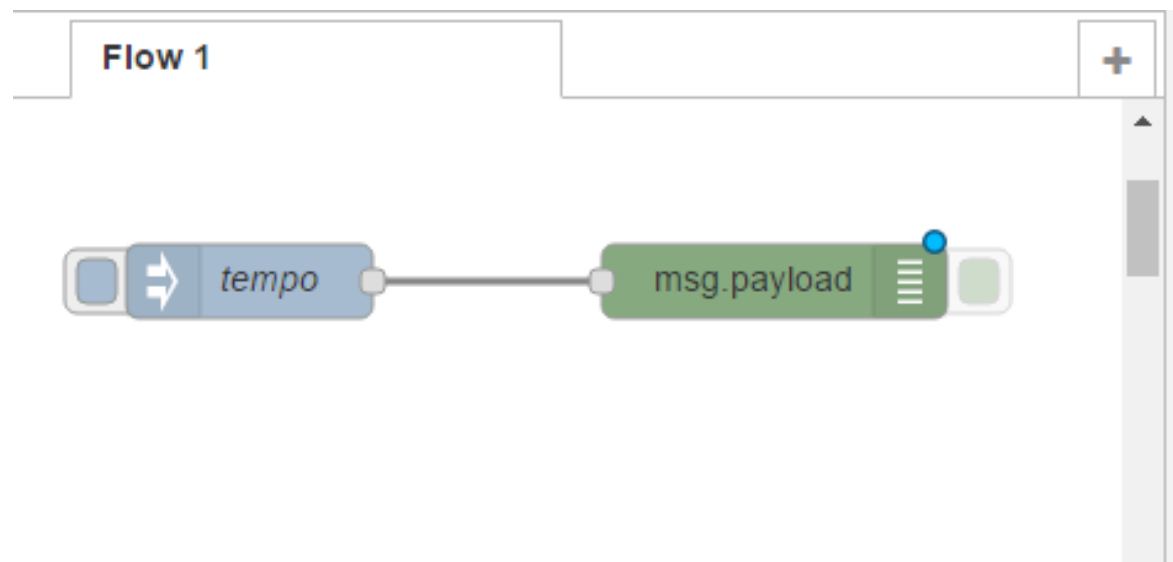
- Uma vez que a programação do Node.js é assíncrona, podemos pensar em programas em que todas as ações são acionadas por um evento gatilho.
 - Sendo assim, podemos pensar na programação do Node.js como fluxos de dados que iniciam a partir de algum evento, como o disparo de um temporizador, a requisição de um cliente de webservice ou um dado vindo do Arduino
- O Node-Red é um serviço escrito para Node.js que provê uma ferramenta visual para editar fluxos de mensagens, vindas de diferentes fontes, podendo ser processadas e mandadas para diferentes destinos, como uma conta de e-mail ou do Twitter
 - A ferramenta para edição dos fluxos roda no próprio browser
 - É possível exportar e importar fluxos no formato JSON usando o menu de opções
- O Node-Red está disponível no IBM Bluemix e em outros provedores de Cloud Computing

Instalação do Node-Red

- O nome do pacote a ser instalado é `node-red`
 - `npm install -g --unsafe-perm node-red`
- Para executar o serviço, executamos:
 - `node-red` para uma instalação global, ou para uma instalação local
 - `node node_modules\node-red\red`
- No lab executamos o comando
 - `node C:\Opensource\Node-Red\red`
- Para acessar o serviço, acessamos no browser:
 - <http://localhost:1880>
- Instalando o Node-Red no Android
 - <https://nodered.org/docs/platforms/android>

Primeiro fluxo (Flow)

- Inicialmente, ligar um nó de entrada do tipo “inject” a um nó do tipo “debug”, fazer um “Deploy” e acionar o injetor de dados
- Observar o resultado na tela de Debug



Tipos de nodes

- Há basicamente três tipos de nodes:
 - **Entrada**, que emitem mensagens a partir de eventos de entrada
 - **Processamento**, que convertem mensagens de entrada em mensagens de saída
 - **Saída**, que enviam a mensagem de entrada para ser consumida em algum lugar
- Dentre os nós de processamento, podemos também criar funções JavaScript genéricas que podem manipular os campos da mensagem como se desejar
- Exemplo: convertendo timestamp em data formatada:

```
function(msg) {  
    msg.payload = new Date(msg.payload).toString();  
    return msg;  
}
```
- O node function usa apenas o corpo da função acima

Capturando da e enviando para a porta serial

- Para usar as funcionalidades da porta serial, é necessário instalar o pacote do Node.js **node-red-node-serialport**
 - `npm install -g --unsafe-perm node-red-node-serialport`
- A captura da porta serial completa a leitura quando:
 - Recebe um caractere de terminação (default: '\n'), ou
 - Estoura o tempo limite (timeout) para novos caracteres, ou
 - Estoura o número máximo de caracteres que podem ser lidos
- O node de saída para a porta serial envia apenas o campo **msg.payload**, podendo ser configurado um caractere de terminação

Configuração do node serialport

- Quando o Arduino é a fonte de dados, ele é representado no Node-Red como um node de **entrada** do tipo serial port.
 - Não confundir: a **saída** de dados do Arduino é usada como **entrada** de dados pelo Node-Red
- Na configuração da porta serial, fundamentalmente dois itens devem ser considerados:
 - O nome da porta serial deve ser o mesmo onde o Arduino está conectado
 - a velocidade de comunicação deve ser a mesma da do Arduino, geralmente 9600 baud
 - Além disso, caso o Arduino envie quebras de linha após cada dado enviado, podemos deixar que a porta serial dispare a cada vez que encontrar o caractere `\n`

Configuração da captura da porta serial

serial in > Add new serial-port config node

Cancel

Add

Serial Port

COM3

Q

Settings

Baud Rate

9600

Data Bits

8

Parity

None

Stop Bits

1

Input

Split input

on the character

\n

and deliver

ascii strings

Output

☒ add split character to output messages

Tip: the "Split on" character is used to split the input into separate messages. It can also be added to every message sent out to the

0 nodes use this config

On all flows

info

debug

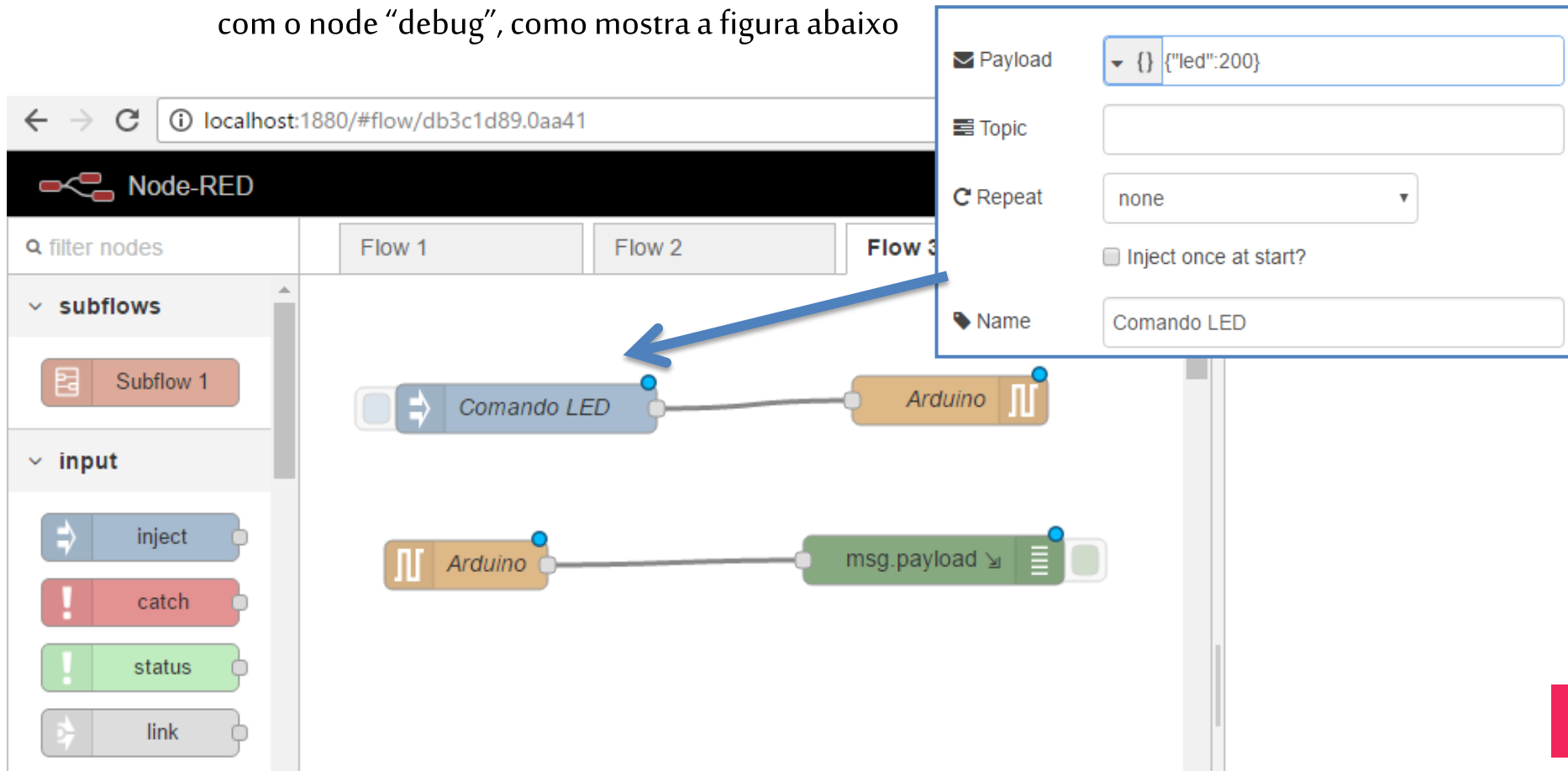
Node

Type	serial-port
ID	f7cdfb42.b939d8

Properties

Trocando dados com o Arduino

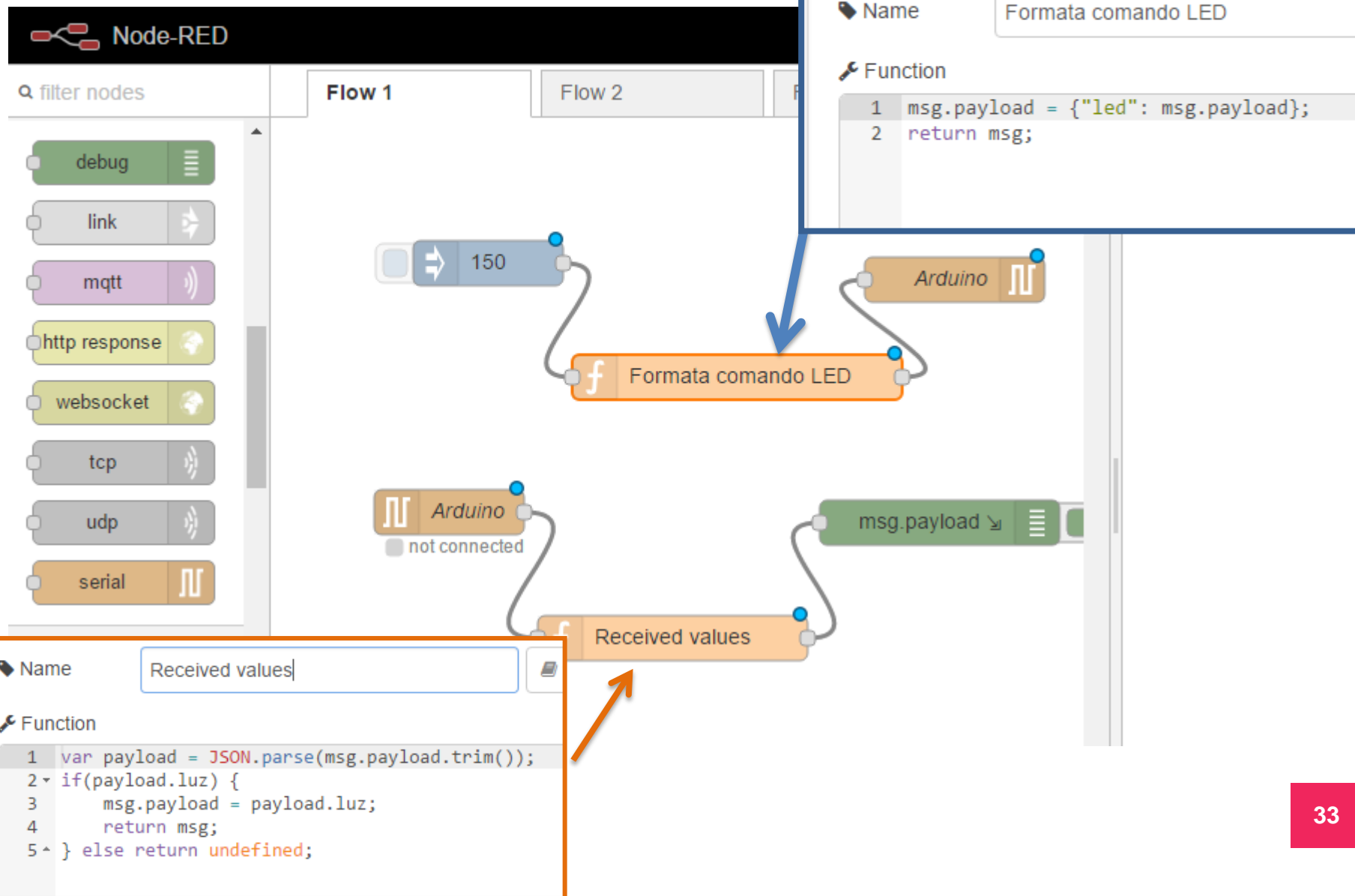
- A forma mais simples de comunicação do Node-Red com o Arduino é injetando diretamente os comandos JSON através do node “inject”, e verificar diretamente a saída com o node “debug”, como mostra a figura abaixo



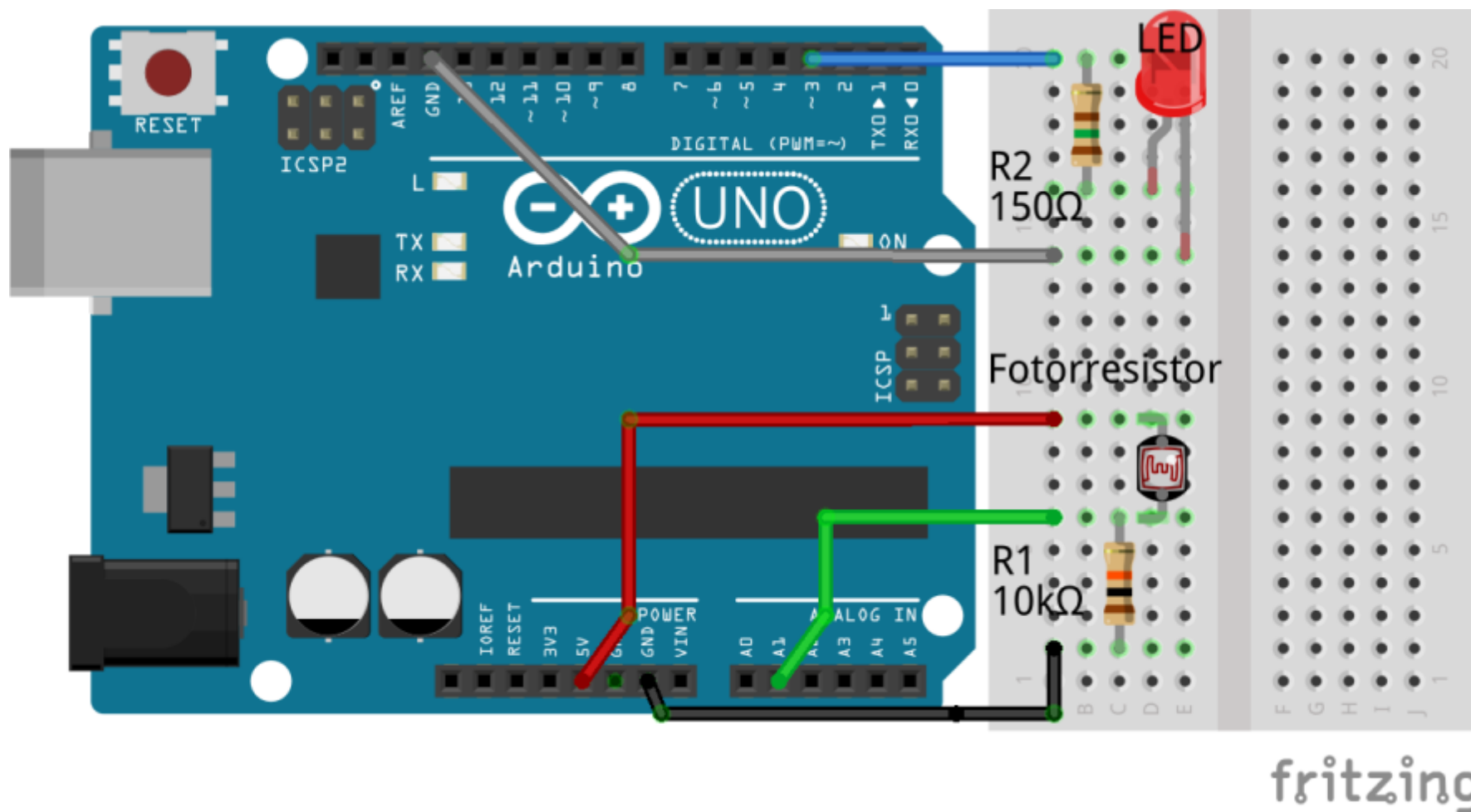
Codificando e decodificando JSON

- Por serem programados em Javascript, os nodes do tipo **function** podem facilmente criar objetos JSON ou decodificar JSON a partir de strings
- No exemplo a seguir, o node inject fornece apenas o valor a ser enviado ao led, enquanto a função “Formata comando LED” formata o comando a ser enviado ao LED do Arduino.
- Similarmente, a função “Parse luz” lê apenas o valor da luminosidade encapsulada no JSON vindo do Arduino

Trocando dados com o Arduino



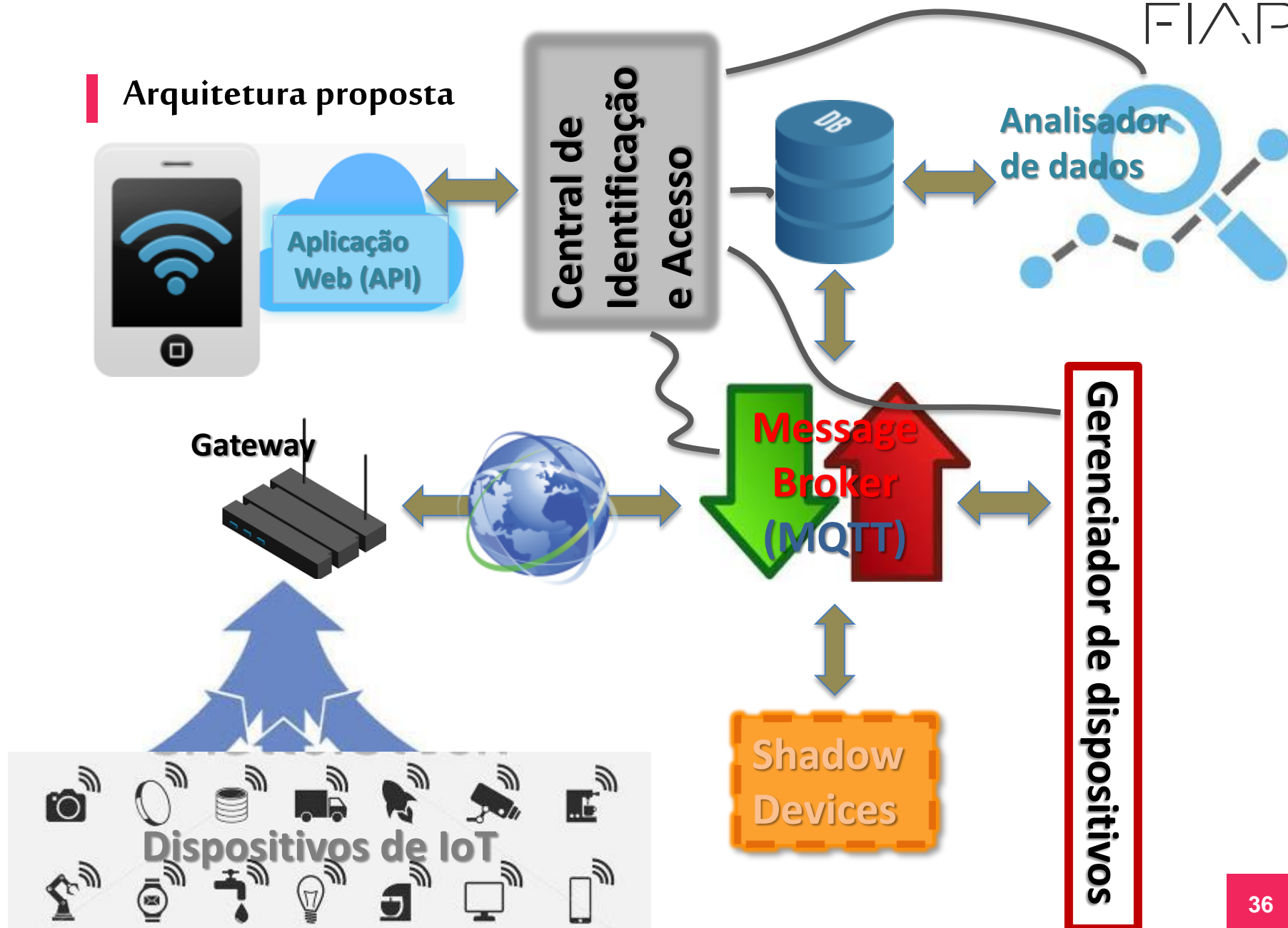
Entrada e saída analógica



Lendo o JSON da porta Serial e mandando a luminosidade

```
#include <ArduinoJson.h>
const int LED = 3;
const int LUZ = A1;
const int TAMANHO = 200;
void setup() {
    Serial.begin(9600);
    Serial.setTimeout(10); //1000ms é muito tempo
    pinMode(LED, OUTPUT);
}
void loop() {
    if (Serial.available() > 0) {
        //Lê o texto disponível na porta serial:
        char texto[TAMANHO];
        Serial.readBytesUntil('\n', texto, TAMANHO);
        //Grava o texto recebido como JSON
        StaticJsonBuffer<TAMANHO> jsonBuffer;
        JsonObject& json = jsonBuffer.parseObject(texto);
        if(json.success() && json.containsKey("led")) {
            analogWrite(LED, json["led"]);
        }
    }
    StaticJsonBuffer<TAMANHO> jsonBuffer;
    JsonObject& json = jsonBuffer.createObject();
    json["luz"] = analogRead(LUZ);
    json.printTo(Serial); Serial.println();
    delay(1000);
}
```

Arquitetura proposta



MQTT – MQ Telemetry Transport

- Protocolo muito simples para publicação e recebimento de mensagens, apropriado para dispositivos com alta latência e baixa largura de banda de comunicação
- A leveza do protocolo, que usa cabeçalhos de poucos bytes, o torna adequado para a comunicação de objetos no cenário da internet das coisas
- Um MQTT broker faz o papel de servidor, que gerencia as mensagens publicadas, enviando-as aos clientes que se inscreveram para recebê-las
- Os clientes MQTT são as pontas da comunicação, podendo enviar ou receber mensagens através das operações:
 - **Publish:** um cliente MQTT publica uma mensagem com determinado tópico
 - **Subscribe:** um cliente se cadastra no servidor para receber cópias de mensagens com determinado tópico

Usando o MQTT – conectando a um servidor

- O protocolo MQTT pode ser testado facilmente empregando simples aplicativos para celular
- Após a instalação do programa cliente basta configurar a conexão com o servidor MQTT, também chamado de “Message Broker”, fornecendo seu endereço IP ou URL, na porta padrão 1883
- Para uso em teste, um servidor público pode ser empregado, tais como:
 - `iot.eclipse.org`
 - `test.mosquitto.org`
 - `dev.rabbitmq.com`
 - `broker.mqttdashboard.com`
- Também é possível instalar e configurar o próprio servidor MQTT
 - No caso de uso em uma rede local, com poucas conexões, o servidor mosquitto (`mosquitto.org`) é o mais apropriado, por consumir poucos recursos, podendo ser executado em plataformas de IoT como Raspberry Pi
 - Para uso no ambiente da Cloud Computing, onde podemos esperar milhões de conexões e precisamos de escalabilidade, precisamos de um servidor do tipo RabbitMQ (`www.rabbitmq.com`)

■ Usando o MQTT – configurações de segurança

- Para experimentar o MQTT, não é necessário configurar nenhuma das opções de usuário, senha ou Client ID
 - Para se autenticar em servidores restritos, podemos fornecer usuário e senha, empregar um certificado de cliente, ou ambos, sendo a segunda opção mais indicada para o registro de dispositivos de IoT
 - O Client ID é uma identificação única do cliente dentro do servidor, usada para gerenciar as informações da sessão. Caso o usuário não configure esse campo, a API do cliente MQT deveria gerar um identificador aleatório, ignorando sessões anteriores e criando uma nova (*clean session*)
- Para que a comunicação com o servidor seja criptografada, uma das seguintes portas deve ser usada:
 - Porta 8883: socket com encriptação TLS
 - Porta 8884: socket com encriptação TLS e uso de certificado de cliente

Usando o MQTT – publicando e recebendo mensagens

- Após a conexão com o servidor, o cliente MQTT pode publicar uma mensagem com um tópico, ou se inscrever para receber as mensagens que forem publicadas naquele tópico
- QoS (Quality of Service) – indica o nível de verificação de recebimento de mensagens pelo servidor (publish) ou pelo cliente (subscribe)
 - QoS 0: nenhuma verificação é feita
 - QoS 1: garantia de recebimento da mensagem pelo menos uma vez
 - QoS 2: garantia de recebimento da mensagem exatamente uma vez
- Publicação com opção de retenção (retain)
 - No caso de ser usada a opção de retenção de mensagem, o servidor irá salvar essa mensagem e enviá-la a qualquer cliente que faça a subscrição àquele tópico.
 - Para remover esse comportamento, deve ser enviada ao tópico uma mensagem vazia usando essa opção de retenção

Tópicos do MQTT

- No MQTT, um tópico define um canal onde mensagens são enviadas e recebidas, como uma fila simples de mensagens.
- Um tópico é uma string UTF-8 sensível à caixa definindo uma estrutura hierárquica, podendo consistir em um ou mais níveis separados por barra (/), podendo ou não refletir uma estrutura predefinida, por exemplo:
 - São Paulo/Cambuci/Lins de Vasconcelos/Fiap/lab701/maquina01/luz
- Um nome de tópico deve ser bastante específico para sua finalidade
- É possível usar caracteres coringa na subscrição de tópicos:
 - Níveis múltiplos (multi-level): o caractere '#' substitui todos os níveis acima do qual ele foi definido. Por exemplo, "MinhaCasa/#" pode substituir os tópicos "MinhaCasa/Sala/Luz" e "MinhaCasa/Entrada", mas não "minha casa/cozinha"
 - Nível simples (single-level): o caractere '+' substitui um nível específico na estrutura hierárquica do tópico. Assim, o tópico "MinhaCasa/+/luz" pode substituir "MinhaCasa/Sala/Luz" e "MinhaCasa/Cozinha/Luz", mas não "MinhaCasa/Sala/Temperatura"

Tópicos no contexto da IoT

- Não existe um padrão universal para a estrutura de tópicos.
- Porém, no contexto de IoT, é interessante usar tópicos que sejam específicos o suficiente para a sua fácil interpretação pelo projetista do sistema, mas flexíveis o bastante para se adaptar às diversas possibilidades
 - A estrutura do tópico pode incluir informações como localização geográfica, localização específica, identificador de dispositivo, informações do proprietário, etc.
- De forma geral, o gateway de IoT irá publicar as informações de sensores em tópicos que definirão o tipo de sensor sendo usado, bem como subscreverá tópicos que definam o atuador para o qual comandos estão sendo recebidos
- Nos experimentos do laboratório, vamos adotar o seguinte padrão de tópicos, onde IP é o número da máquina:
 - `fiap/<laboratório-id>/arduino<IP>/<dispositivo>`
- Por exemplo, para enviar as informações do sensor de luz o gateway publicará em
 - `fiap/lab701/arduino01/luz`
- Para receber os comandos de acendimento do LED, o gateway subscreverá
 - `fiap/lab701/arduino01/led`


Cliente MQTT no Node-RED

- A instalação padrão do Node-RED inclui o cliente MQTT, com dois tipos de nodes
 - Node de entrada MQTT: realiza uma operação de `subscribe` em um tópico específico, podendo também usar caracteres coringa
 - Node de saída MQTT: realiza uma operação de `publish` em um tópico específico
- Configuração do servidor:
 - Uma vez que a porta padrão do MQTT é bloqueada pelo proxy, vamos usar um broker `mosquitto` rodando na máquina do professor.
 - Assim, nos experimentos de laboratório, a informação de servidor deve ser o IP da máquina do professor

Configurando o broker MQTT

Edit mqtt in node

Delete Cancel Done

Server Add new mqtt-broker... 

Topic Topic

QoS 2

Name Name

mqtt in > **Add new mqtt-broker config node** Cancel Add

Connection Security Birth Message Will Message

Server 10.6.20.51 Port 1883

☐ Enable secure (SSL/TLS) connection

Client ID Leave blank for auto generated

Keep alive time (s) 60 ☒ Use clean session

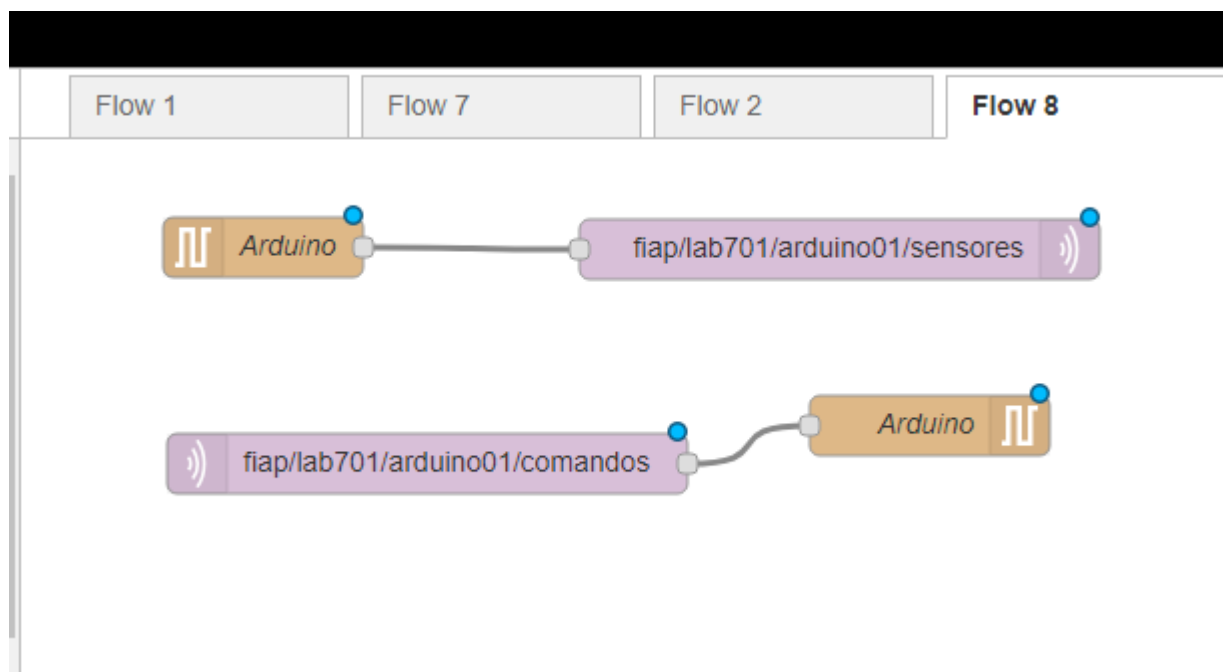
☒ Use legacy MQTT 3.1 support

IP da máquina do professor

■ Programação de um simples gateway Arduino no Node-RED

- Vamos programar um gateway simples que conecta um Arduino a um MQTT broker.
- Lembre-se de que embora apenas as máquinas do lab possam se conectar ao broker, bastaria o acesso a um broker possuindo um IP real na internet para que o Arduino pudesse ser comandado a partir de qualquer lugar
- No gateway usaremos apenas dois tópicos MQTT:
- Publica as leituras do sensor de luminosidade na forma numérica (0 a 1023)
 - `fiap/<laboratório-id>/arduino<IP>/luz`
 - Vamos usar um QoS 0, já que não há problemas em se perder alguma leitura
- Subscrive comandos numéricos para definir o brilho do LED (0 a 255)
 - `fiap/<laboratório-id>/arduino<IP>/led`
 - Vamos usar um QoS 1, já que há problemas em perder um comando, mas não há problemas em receber o mesmo comando múltiplas vezes

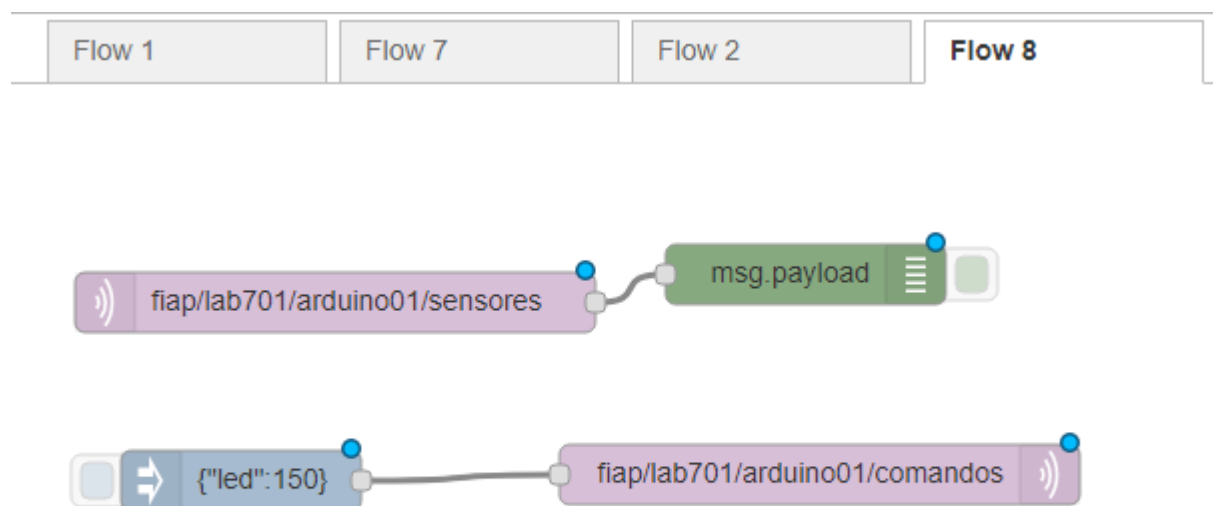
Programação do gateway Arduino com MQTT



■ Testando o gateway

- Para testar o gateway, basta criar um outro fluxo de dados que funciona como um cliente remoto do Arduino:
 - Node tipo **inject** envia o comando do LED para um node MQTT out que publica em `fiap/<laboratório-id>/arduino<IP>/comandos`
 - Node tipo MQTT out que subscreve o tópico `fiap/<laboratório-id>/arduino<IP>/sensores` e envia o dado para um node tipo debug
 - Vamos tentar?

Cliente remoto



REFERÊNCIAS



1. Min-Woo Ryu et al. **Survey on Internet of Things: Towards Case Study**. The Smart Computing Review, v. 2(3), 2012.
2. Gartner. **Gartner IT Glossary**. url: <http://www.gartner.com/it-glossary/cloud-computing>
Acesso em 17/01/2016
3. P. Mell e T. Grance. **The NIST Definition of Cloud Computing**. NIST, 2011. url: <http://dx.doi.org/10.6028/NIST.SP.800-145>
Acesso em 17/01/2016
4. European Technology Platform for Electricity Networks of the Future. **Smart Grids**. url: http://www.smartgrids.eu//ETP%20SG%20leaflet%20_2015.pdf
Acesso em 17/01/2016
5. O. Vermesan e P. Fries. **Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems**. Rivers, 2013. url: http://www.internet-of-things-research.eu/pdf/Converging_Technologies_for_Smart_Environments_and_Integrated_Ecosystems_IERC_Book_Open_Access_2013.pdf
Acesso em 15/02/2015

REFERÊNCIAS

6. Joyent. **Node.js**. url: <http://www.nodejs.org>
Acesso em 20/01/2016
7. Joyent. **Node.js API**. url:
<http://nodejs.org/api/>
8. IBM Emerging Technologies. **Node-Red**. url:
<http://nodered.org>





Copyright © 2019 Prof. Antonio Selvatici

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).