FIAP GRADUAÇÃO

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Arquiteturas Disruptivas e Big Data

PROF. ANTONIO SELVATICI

Versão 3– Fevereiro/2017

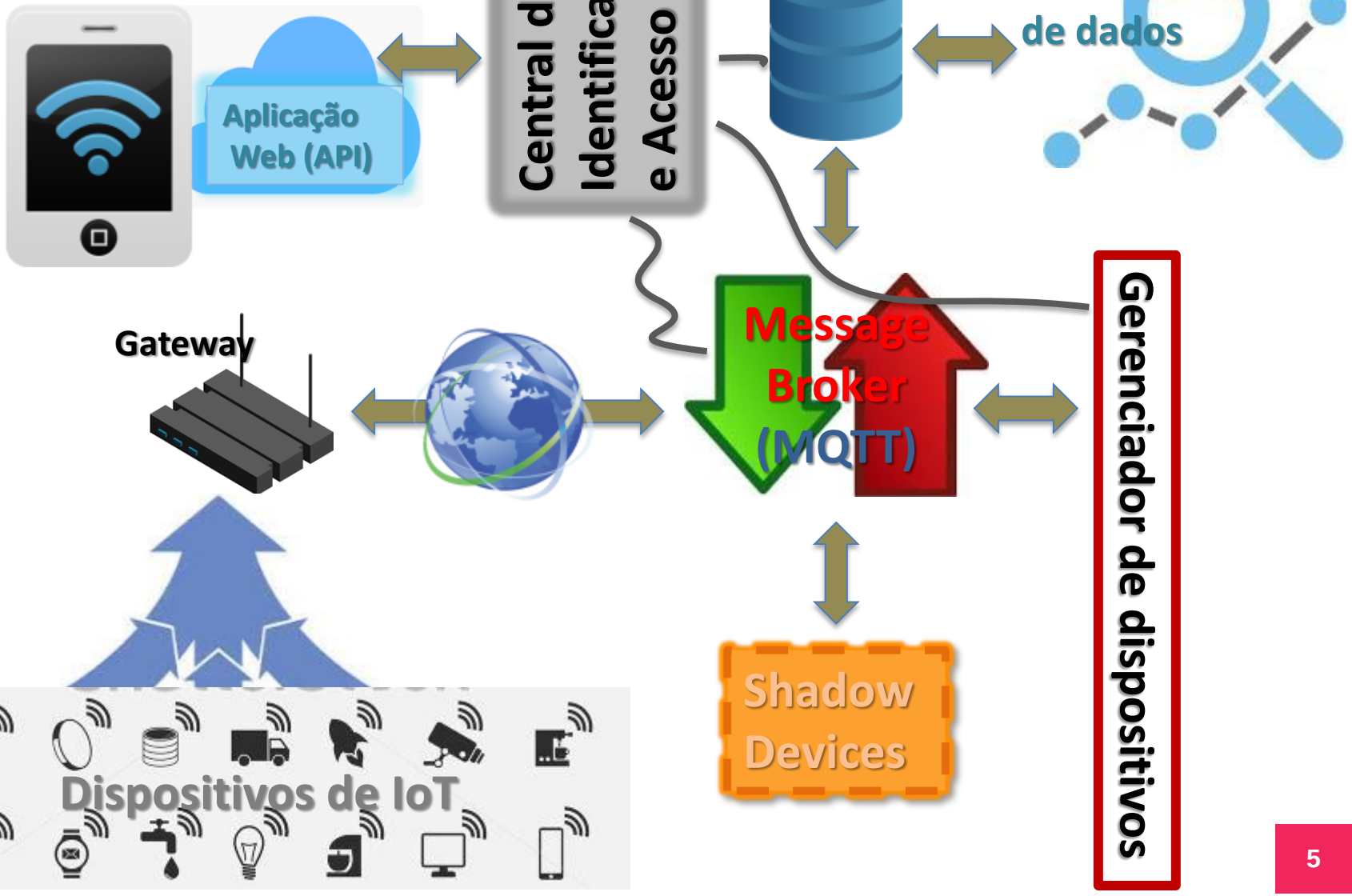
SHORT BIO

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 e Systax.

PROF. ANTONIO SELVATICI
profantonio.selvatici@fiap.com.br

INTERNET DAS COISAS

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 subscreveram 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 necessitamos 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 subscrever 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"

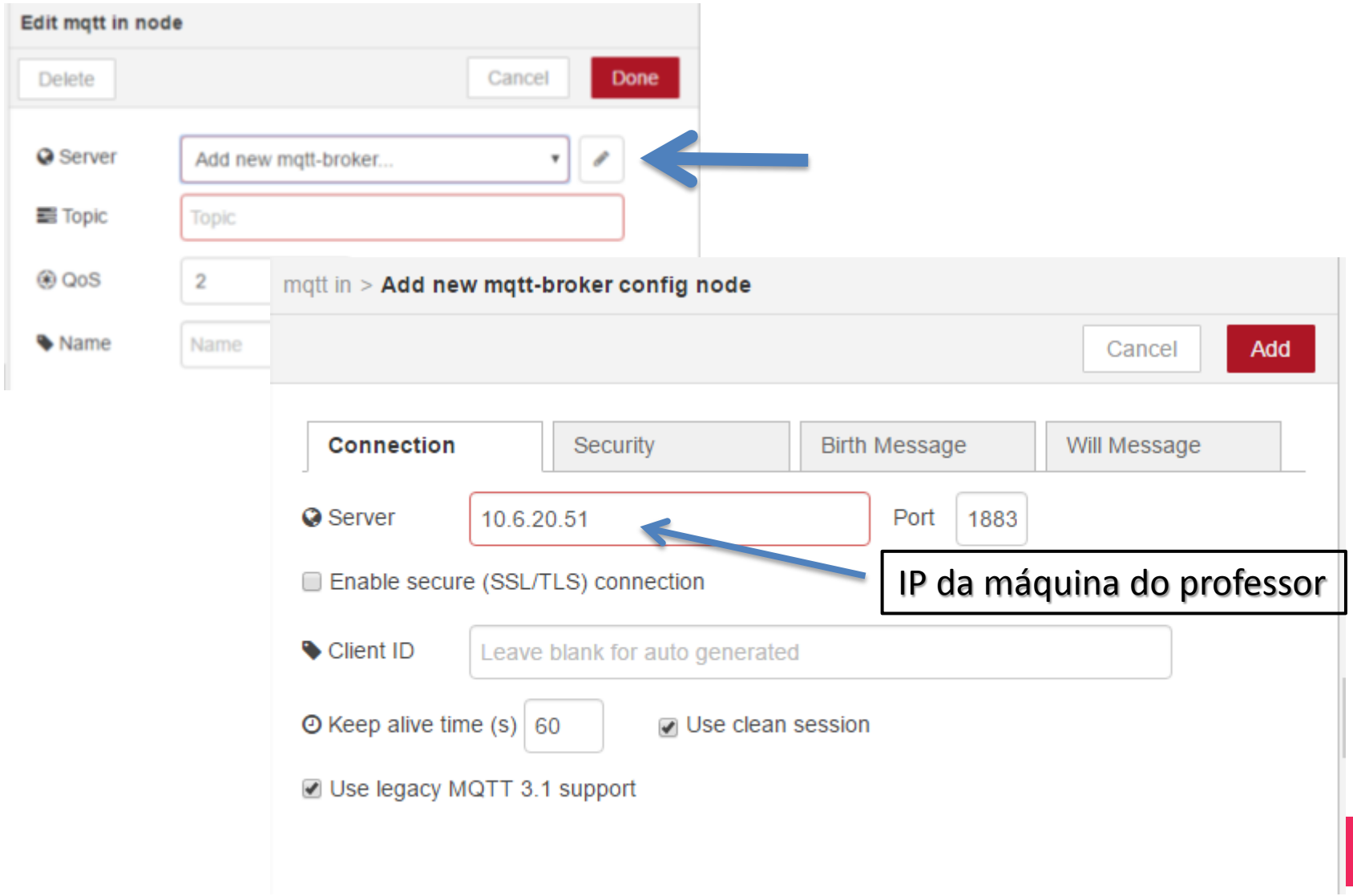
Tópicos no contexto da IoT

- 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 recebeidos
- 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

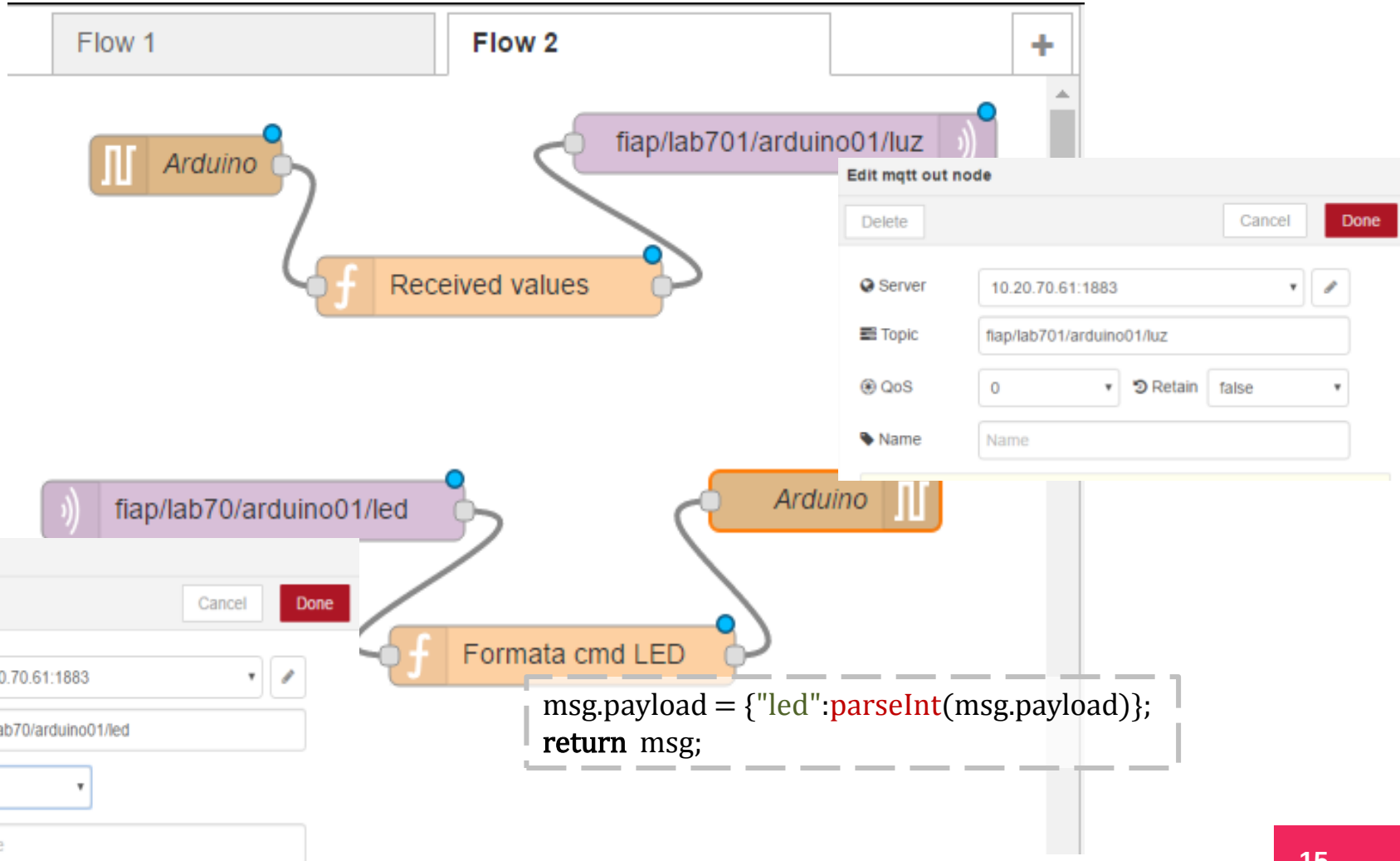
Configurando o broker MQTT



Programação de um simples gateway Arduíno no Node-RED

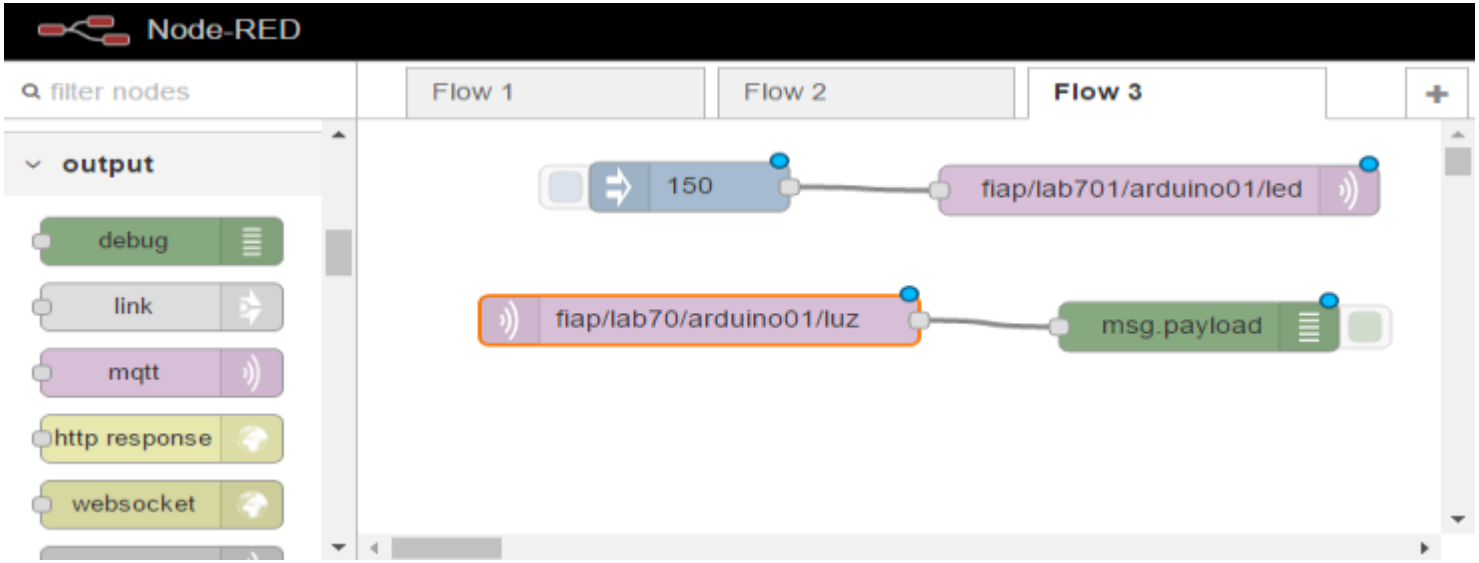
- Vamos programar um gateway simples que conecta um Arduino a um MQTT broker.
- Lembre-se de que embora apenas as máquinas do lab possam se conectar ao broker, bastaria o acesso a um broker possuindo um IP real na internet para que o Arduino pudesse ser comandado a partir de qualquer lugar
- No gateway 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
- Subscreve 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

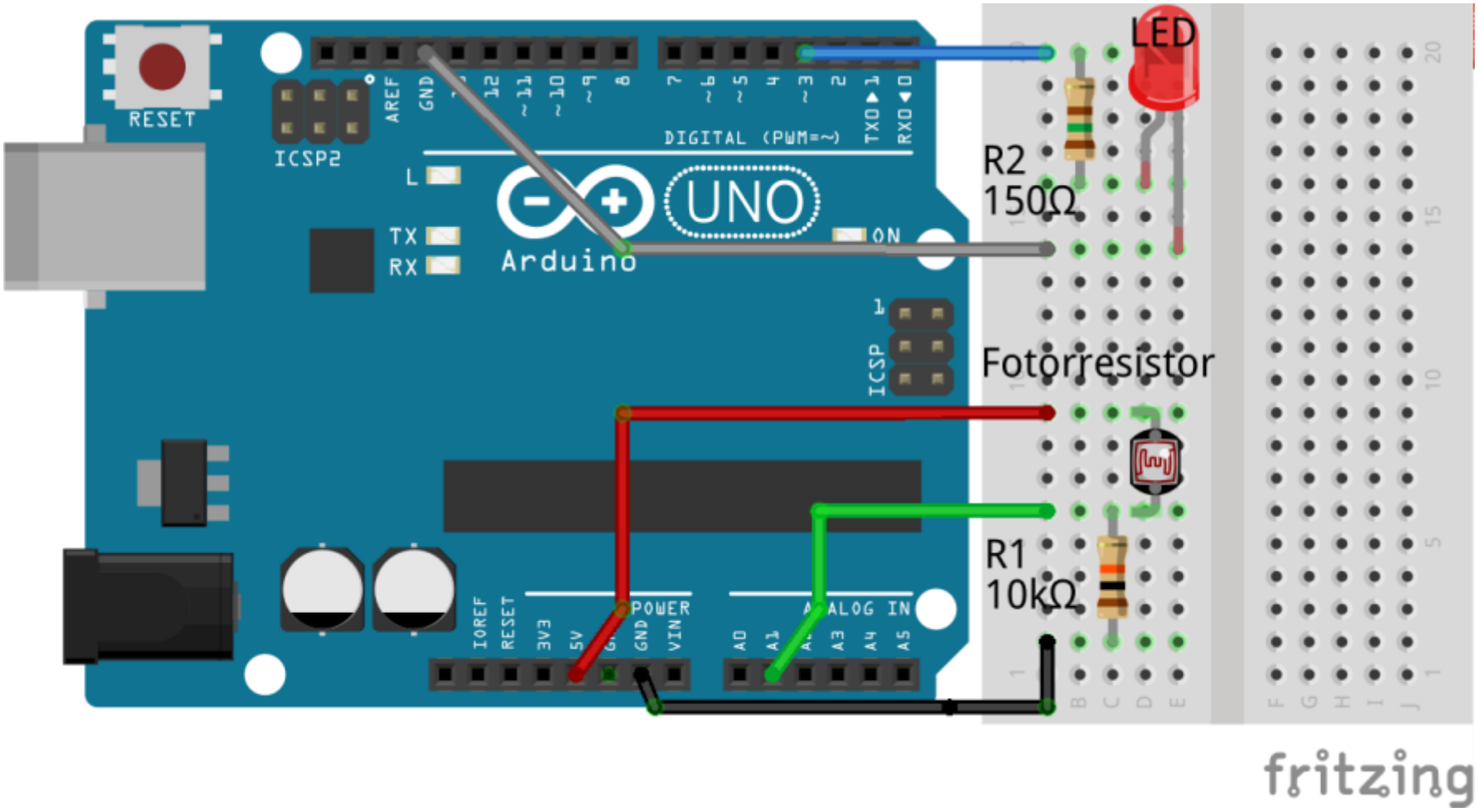


Testando o gateway

- Para testar o gateway, basta criar um outro fluxo de dados:
  - Node tipo **inject** envia o comando do LED para um node MQTT out que publica em `fiap/<laboratório-id>/arduino<IP>/led`
  - Node tipo MQTT out que subscreve o tópico `fiap/<laboratório-id>/arduino<IP>/luz` e envia o dado para um node tipo debug



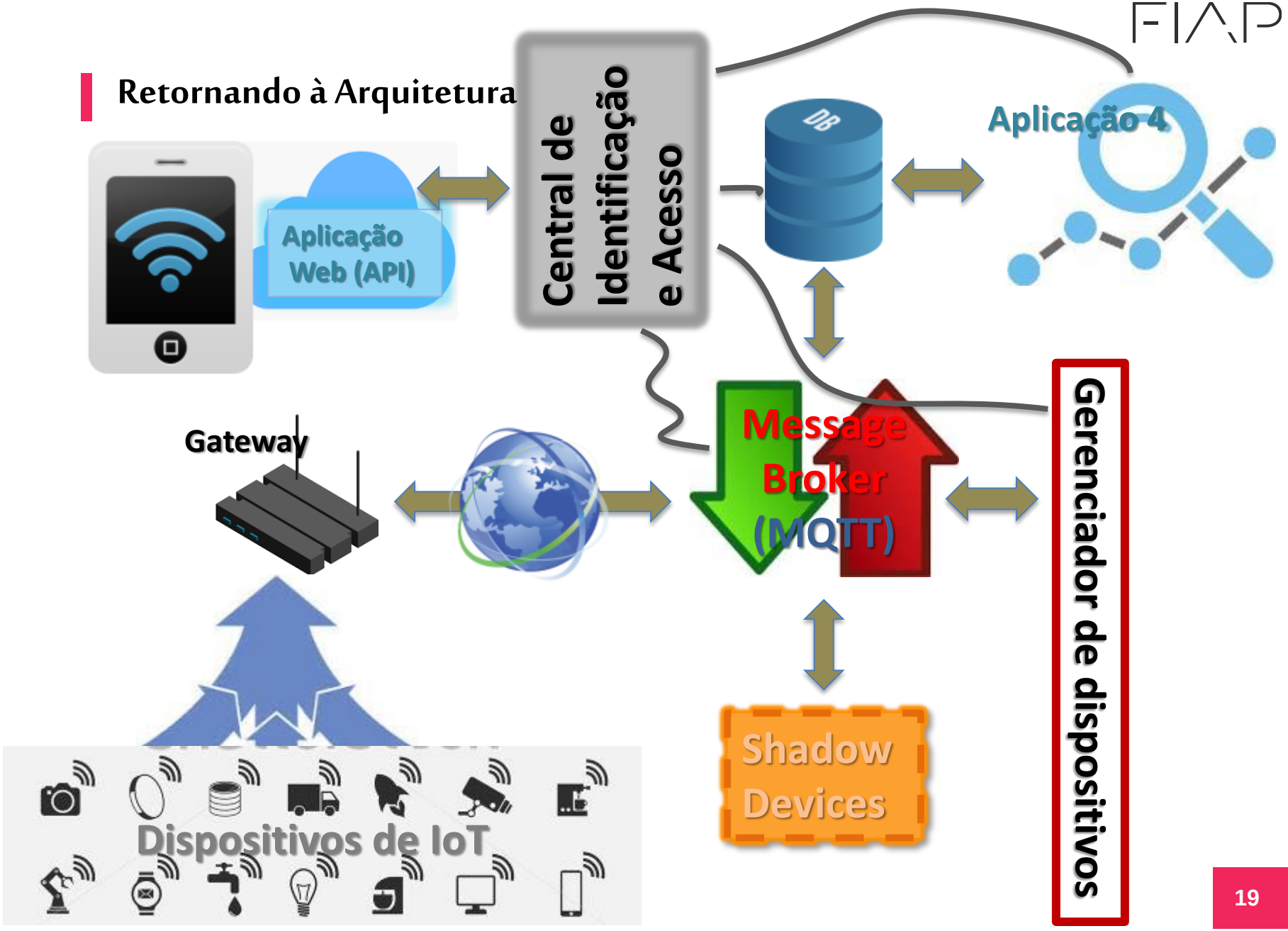
Montagem do circuito: Fotorresistor e LED



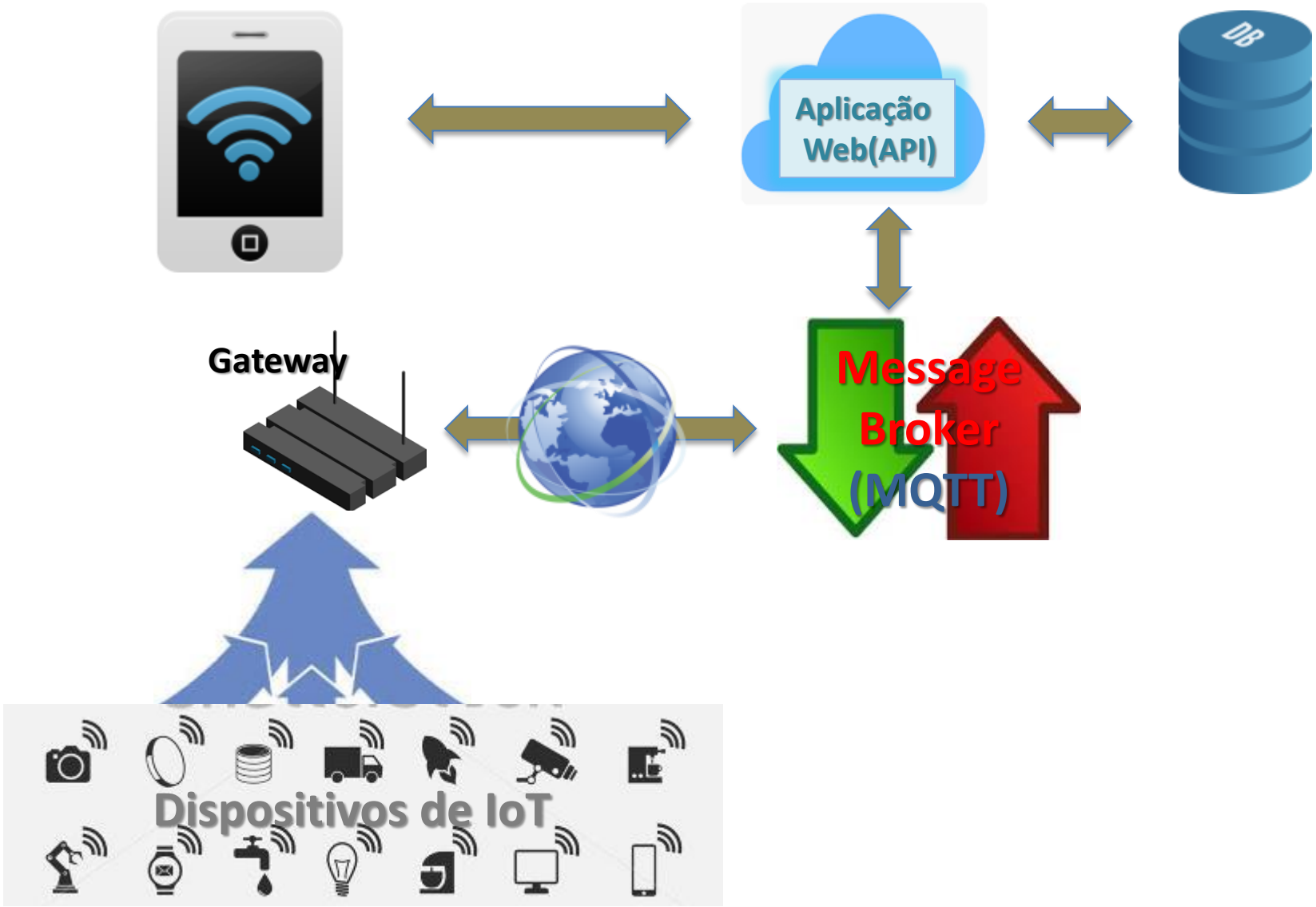
Lendo o JSON da porta Serial e mandando a luminosidade

```
#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);
}
```

Retornando à Arquitetura



Arquitetura Simplificada



Programando a Aplicação Web – RESTful API

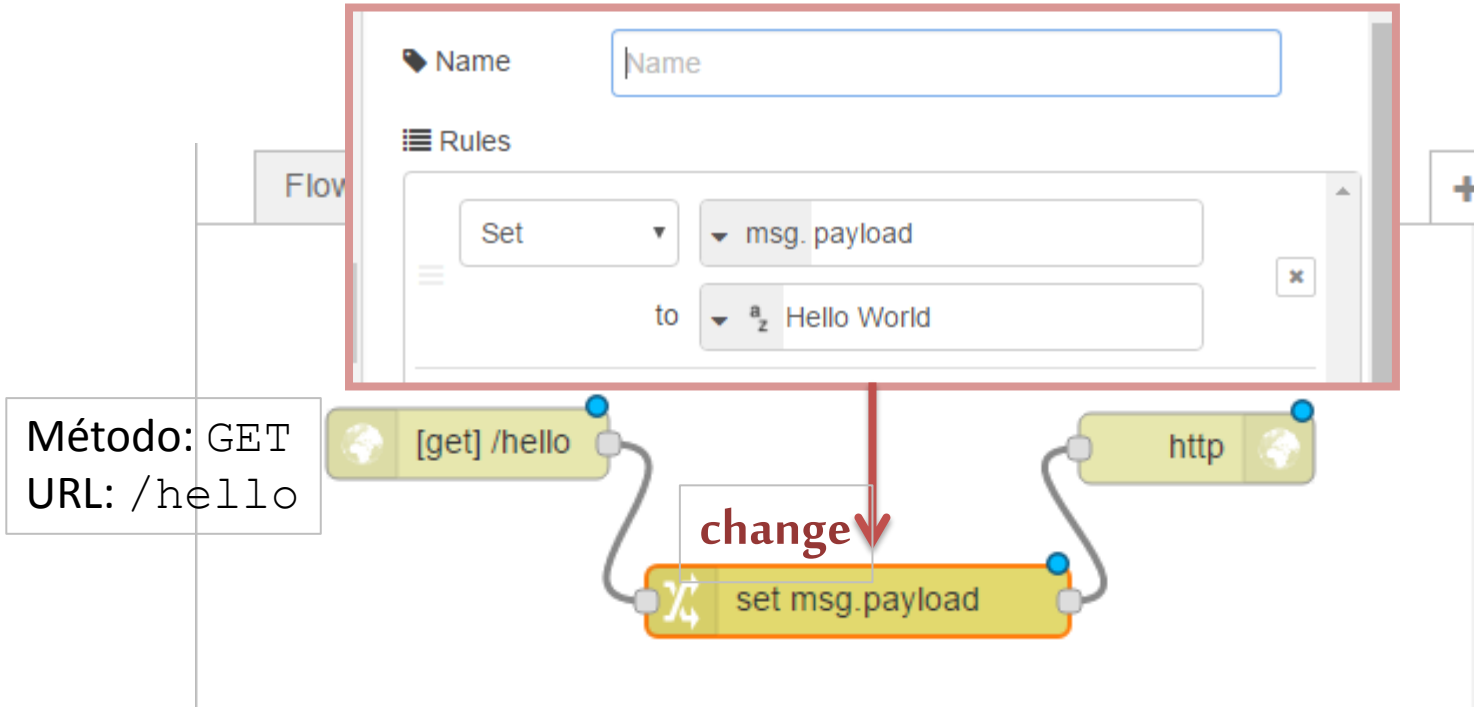
- Numa arquitetura simplificada de IoT, uma única aplicação Web se encarrega de gerenciar e controlar o acesso aos dispositivos e comandar o armazenamento em banco de dados, além de fornece uma API, possivelmente RESTful, para que programas aplicativos possam ter acesso às regras de negócio
- Essa aplicação precisa ter três pontas:
  - A ponta do cliente do banco de dados
  - A ponta do cliente MQTT
  - A ponta do servidor REST
- Via de regra, essa aplicação web é desenvolvida usando linguagens de programação flexíveis, como Java, Ruby, ou, como tem se tornado preferência no mundo de IoT, Node.js
- No entanto, para aplicações simples, podemos usar o próprio Node-RED para programá-la

Servidor Web Node-RED

- O próprio Node-RED é um servidor web, que escuta em geral na porta 1880
- Podemos aproveitar esse mesmo servidor e criar URLs adicionais customizadas, definindo ainda o comando HTTP a ser executado (GET, POST, PUT, ou DELETE)
- Para criar um servidor simples, usamos o node "HTTP in" como fonte de dados, e finalizar o fluxo em um node "HTTP response"
- O corpo da resposta é definido pelo campo **msg.payload**, da mensagem **msg** recebida pelo node de saída HTTP, como exemplificado no próximo slide

Exemplo de servidor simples

- Servidor escutando em http://localhost:1880/hello
- O node "**change**" define o valor do campo de uma variável usando alguma regra



Servidor REST com JSON

- No caso de um servidor REST baseado em JSON, as respostas às requisições não são trechos textuais simples, mas possuem uma formatação específica
  - O formato da resposta deve ser especificado através do cabeçalho "Content-Type" da resposta HTTP
- Para ter acesso ao parâmetros de requisição e resposta, basta acessar, respectivamente, os campos `msg.req` e `msg.res` da mensagem resultante do node "HTTP in"
- Para indicar a resposta JSON e liberar as requisições *cross-site* (CORS) da nossa API, usamos o node "change" :
  - **Set:** `msg.headers`
  - **To** (JSON): `{"Content-Type":"application/json", "Access-Control-Allow-Origin":"*"}`

Objetos de requisição e resposta

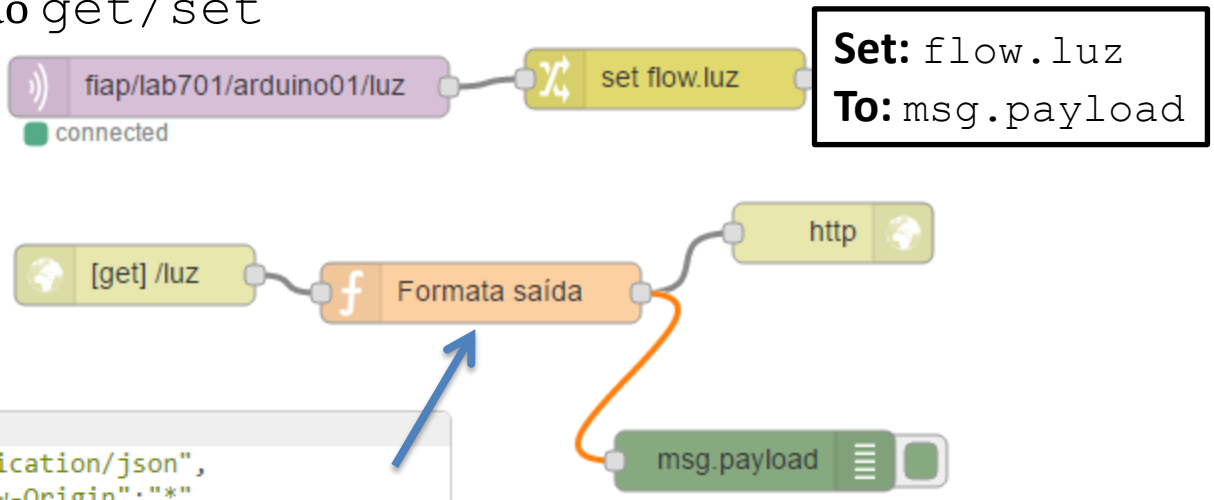
- Uma vez que Node-RED usa o pacote `express` do Node.js como servidor web, os campos `msg.req` e `msg.res` são objetos de requisição e resposta e respeitam a estrutura das classes `HTTPRequest` e `HTTPResponse` respectivamente.
- Os campos relevantes dos objetos de requisição e resposta encontram-se na documentação do próprio express . Alguns deles são:
  - **msg.req.path**: caminho do recurso requisitado no servidor
  - **msg.req.ip**: IP do cliente que realizou a requisição
  - **msg.req.body**: corpo da requisição (geralmente um JSON ou formulário URL-encoded)
  - **msg.req.headers**: cabeçalho da requisição HTTP
- Para definir os principais parâmetros da resposta, preenchemos diretamente os campos em msg:
  - **msg.payload**: corpo da resposta HTTP
  - **msg.headers**: cabeçalho da resposta HTTP
  - **msg.statusCode**: código de resposta HTTP

Simples servidor para informar o último valor lido do sensor

- É necessário tratar dois eventos que estão fora de sincronia: a chegada de dados do Arduino via tópico MQTT e chegada de requisição HTTP do cliente.
- Como sincronizar esses eventos?
  - Armazenar o dado recebido do MQTT em uma variável, e enviar o valor dessa variável quando da requisição HTTP
- Como trabalhar com variáveis no Node-RED?
  - Um node pode armazenar e recuperar informações através de *contexts*, que funcionam como dicionários contendo valores de propriedades
- Há três níveis de *contexts* que podem ser usados no Node-RED:
  - **Local**: pode ser acessado dentro do próprio node
  - **Flow:** é compartilhado por todos os nodes da mesma aba de edição
  - **Global:** é compartilhado por todos os nodes do servidor

Disponibilizando a luminosidade através da URL /luz

- Cada vez que a uma mensagem é recebida do MQTT, ela é armazenada dentro do *context* **flow** na propriedade "luz"
- Cada vez que é feita uma requisição HTTP GET na URL `/luz`, é retornado um JSON com o valor da luminosidade
- Para escrever ou ler o valor de uma propriedade dentro de um context, usamos a notação `get/set`



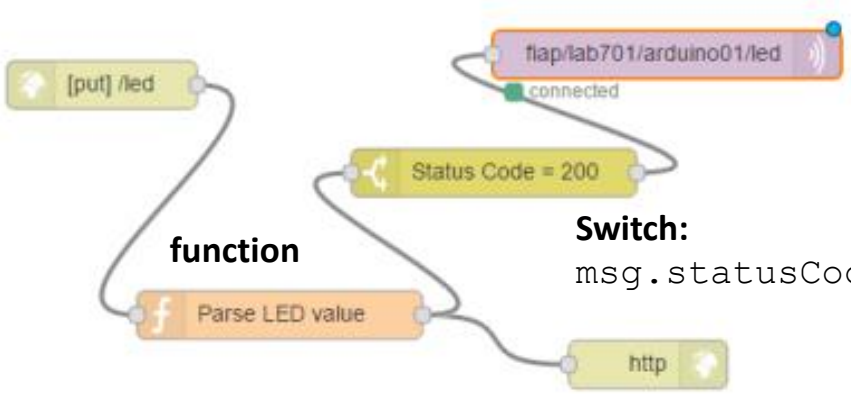
Function

```
1  msg.headers = {
2      "Content-Type":"application/json",
3      "Access-Control-Allow-Origin":"*"
4  }
5  msg.payload = {"value": parseInt(flow.get("luz"))}
6  return msg;
```

Enviando comandos para o Arduino

- Para um programa aplicativo enviar comandos para o Arduino, ele deve comunicar-se com a aplicação Web enviando um comando através de sua API
- A forma como esses comandos são enviados para a API dependem do próprio projeto da API
  - Uma forma de atualizar o brilho do LED do Arduino para 150 é passar um comando POST para a URL: http://hostname/meuarduino/led/150
  - Ou ainda podemos enviar um comando PUT contendo o objeto `{"value":150}` ao endereço http://hostname/meuarduino/led

Processando um comando PUT com um JSON como corpo



QoS: 1
Retain: true

Switch:
`msg.statusCode == 200`

Name: Parse LED value

Function

```
1  if("value" in msg.payload) {
2      msg.payload = msg.payload.value;
3      msg.statusCode = 200;
4  } else {
5      msg.statusCode = 400;
6      msg.payload = "Bad request format";
7  }
8  return msg;
```

REFERÊNCIAS

1. Dc square Gmbh. **MQTT Essentials**. url: http://www.hivemq.com/blog/mqtt-essentials/
2. IBM Emerging Technologies. **Node-Red.** url: http://nodered.org
3. Rodger Lea. **Node RED programming guide.** http://noderedguide.com/tag/mqtt/

FIAP