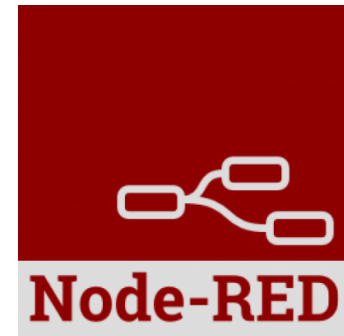
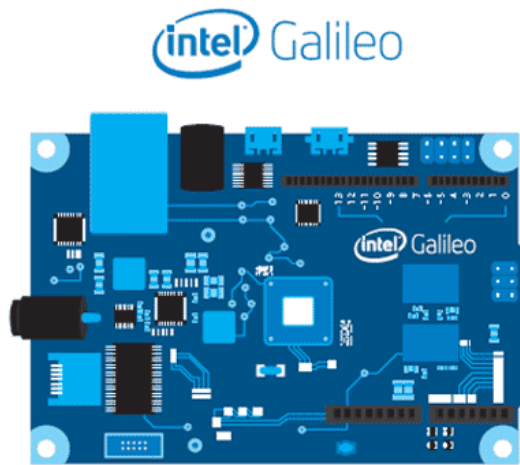


Geração de eventos para atuação do dispositivo IoT via **Node-Red**



Objetivos

- Assinar o Galileo num canal MQTT (alteração de código no eclipse).
 - paradigma *publish-subscribe*
- Criar interface para envio dos dados sobre o Node-Red.
- Processar os dados recebidos pelo Galileo, e gerar comportamentos condicionais.
- Construir estrutura para envio de dados ao tópico em que o Galileo foi subscrito.
- Construir pequena aplicação web para interação do usuário. O usuário poderá variar a intensidade de um LED utilizando um *'slider'* em uma página HTML (funciona no celular também).
- Alterar o código do eclipse para integração com o projeto de luminosidade.

Serviço Web

- Uma aplicação auto-contida, identificada por um URI (*Uniform Resource Identifier*), cujas interfaces e ligações são definidas, descritas e localizadas por artefatos que utilizam a linguagem XML (Extensible Markup Language) ou JSON (JavaScript Object Notation).
- Deve ser capaz de interagir com outras aplicações através da troca de mensagens XML/JSON utilizando os protocolos de comunicação padrão atualmente disponíveis na Internet.
- Faz com que os recursos da aplicação do software estejam disponíveis sobre a rede de uma forma normalizada.
- REST (Representational State Transfer) ou RESTfull são padrões arquiteturais que permitem definir e implementar serviços web.

Serviço Web

XML

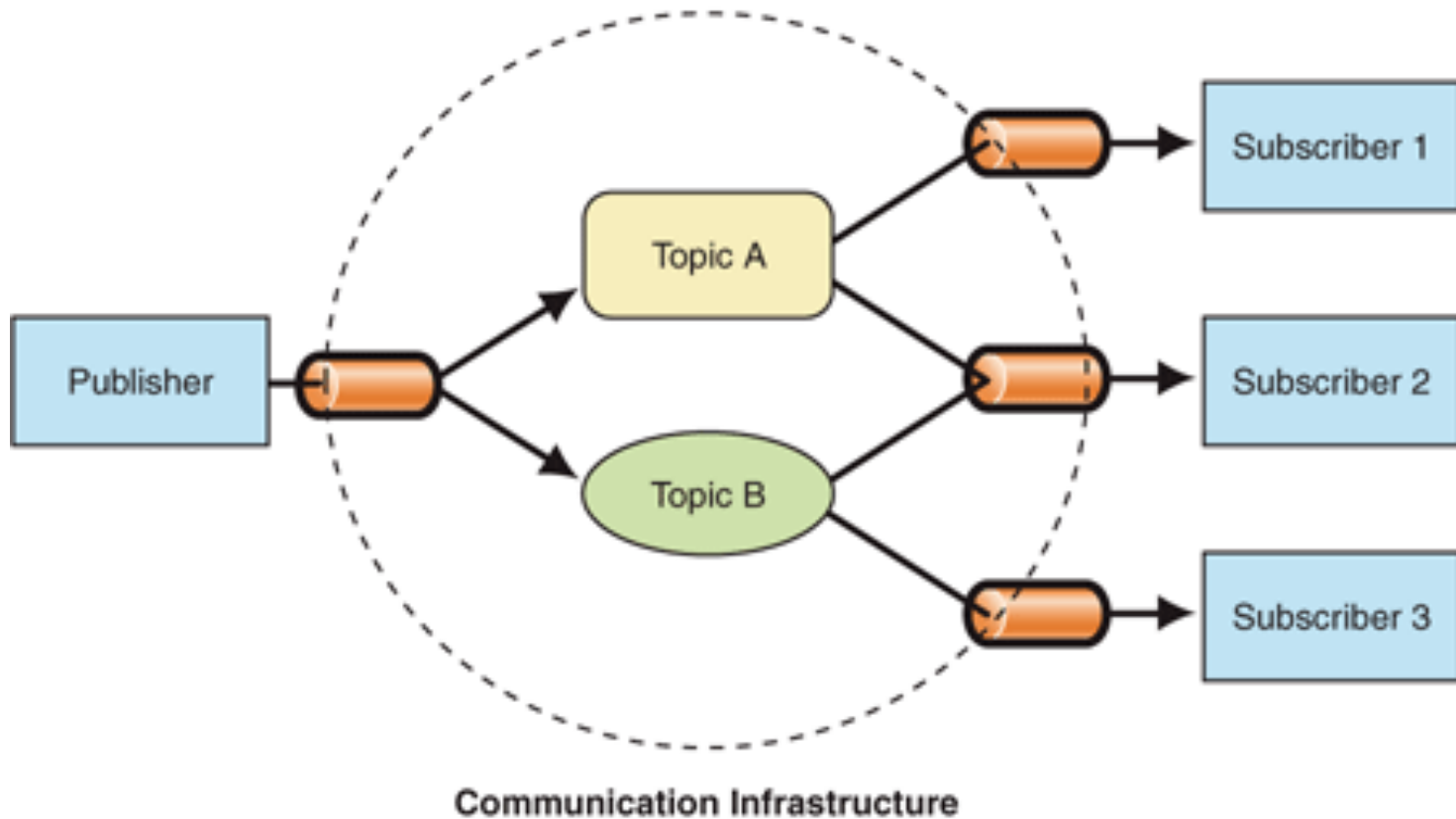
```
...>  
<name>Barry & Associates, Inc.</name>  
<phone>612-321-8156</phone>  
<street1>14597 Summit Shores Dr</street1>  
<street2></street2>  
<city>Burnsville</city>  
<state>MN</state>  
<postalcode>55306</postalcode>  
<country>United States</country>  
< ...
```

JSON

```
{  
  "name"      : "Barry & Associates, Inc.",  
  "phone"     : "612-321-8156",  
  "street1"   : "14597 Summit Shores Dr",  
  "street2"   : "",  
  "city"      : "Burnsville",  
  "state"     : "MN",  
  "postalcode": "55306",  
  "country"   : "United States"  
}
```

Paradigma *publish-subscribe*

- Padrão arquitetural usado para a comunicação assíncrona de vários processos

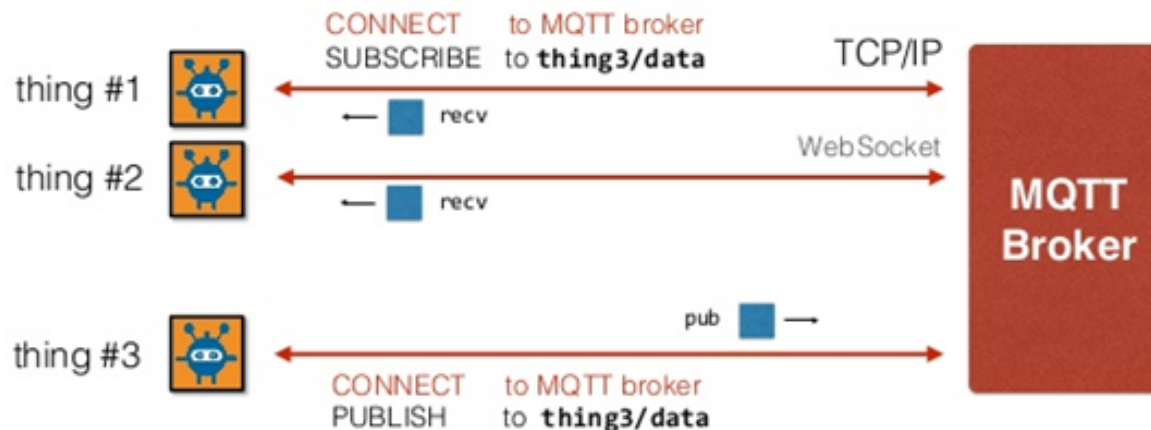


Funcionamento geral do MQTT

Message Queue Telemetry Transport (MQTT)

- O MQTT é um protocolo leve de aplicação que utiliza o protocolo de transporte TCP como via de comunicação de dados para IoT.
- O conceito é baseado na publicação de dados a partir de um *publisher*, e na leitura dos dados a partir da inscrição num canal.
- Clientes MQTT podem funcionar, simultaneamente, como *publisher* e *subscriber*.
- O sistema de monitoramento de respostas é instanciado em *multithread*, portanto os processos de inscrição e publicação são independentes.

MQTT bi-directional, async “push” communication



Assinando Galileo num canal MQTT (*subscriber*)

- Para se efetuar a inscrição é necessário ter um cliente MQTT instanciado, devidamente configurado com o *broker* MQTT (*host*) e as informações do usuário.
 - A conexão estabelecida com o *broker* utiliza o protocolo de transporte TCP, e possui a seguinte estrutura:
`tcp://organization_id.messaging.internetofthings.ibmcloud.com:1883`
Onde *organization_id* é proveniente da criação da sua organização IoT no IBM Bluemix.
- As credenciais do usuário são formadas pelo *id* da organização, tipo do dispositivo e *id* do dispositivo, com a seguinte estrutura:
 - `d:organization_id:device_type:device_id` (formato *string*)
 - `d:zis8fm:Intel_Galileo:984FEE018B08` (Exemplo)

```
#define HOST_PROTO "tcp://"
#define HOST_SUFFIX ".messaging.internetofthings.ibmcloud.com:"
#define HOST_PORT "1883"

snprintf(host, sizeof host, "%s%s%s%s", HOST_PROTO, organization_id, HOST_SUFFIX, HOST_PORT);
snprintf(clientID, sizeof clientID, "d:%s:%s:%s", organization_id, device_type, device_id);

MQTTClient_create(&client, host, clientID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
```

Assinando Galileo num canal MQTT (*subscriber*)

- Após a criação do cliente é necessário configurar a função ***callbacks()***, que tem o objetivo de gerenciar todo o sistema de mensagens de resposta, incluindo as mensagens com o conteúdo do tópico inscrito.
- Ao receber uma atualização no tópico o *broker* envia uma mensagem para os clientes inscritos, alertando a existência de novos dados no tópico.
- Desta forma a função ***callbacks()*** recebe a notificação da atualização. Para tratar os dados recebidos pelo *broker* é necessário implementar uma função com cabeçalho pré determinado e referencia-la na configuração do ***callbacks()***.

```
Antes -> MQTTClient_setCallbacks(client, NULL,&connection_lost, NULL, &delivery_complete);  
Depois -> MQTTClient_setCallbacks(client, NULL,&connection_lost, &msgarrvd, &delivery_complete);  
int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message)
```

- Portanto todos os tratamentos das mensagens serão efetuados na função ***mgsarrvd()***.
 - O nome da função pode ser escolhido pelo programador, porém o cabeçalho deve ser mantido.

Assinando Galileo num canal MQTT (*subscriber*)

- De maneira geral os *brokers* implementam o sistema de modo que o *subscriber* deva assinar o canal com o mesmo nome onde os dados estão sendo publicados. Ou seja, tanto para publicação quanto para assinatura utiliza-se o mesmo nome de canal.
- No entanto o *broker* do Bluemix funciona com um padrão diferente, onde todos os *publishers* devem publicar em tópicos com o sub caminho “*evt*” e se inscrever em tópicos com sub caminho “*cmd*”.

```
#define TOPIC_PUBLISHER "iot-2/evt/status/fmt/json"  
#define TOPIC_SUBSCRIBE "iot-2/cmd/status/fmt/json"
```

- Onde:
 - *iot-2* - é o prefixo dos tópicos utilizado pelo Bluemix
 - *evt* - indica o canal para publicação de eventos
 - *cmd* – indica o canal para utilização de comandos (*subscriber*)
 - *status* – nome dado ao evento ocorrido (pode ser escolhido pelo usuário)
 - *fmt* – indica que o próximo parâmetro determina o tipo de dados que será publicado

Assinando Galileo num canal MQTT (*subscriber*)

- Após a criação dos devidos canais, e da instância de um cliente, basta efetuar o comando para se inscrever no tópico:

```
MQTTClient_subscribe(client, const_cast<char *>(TOPIC_SUBSCRIBE), MQTT_DEFAULT_QOS);
```

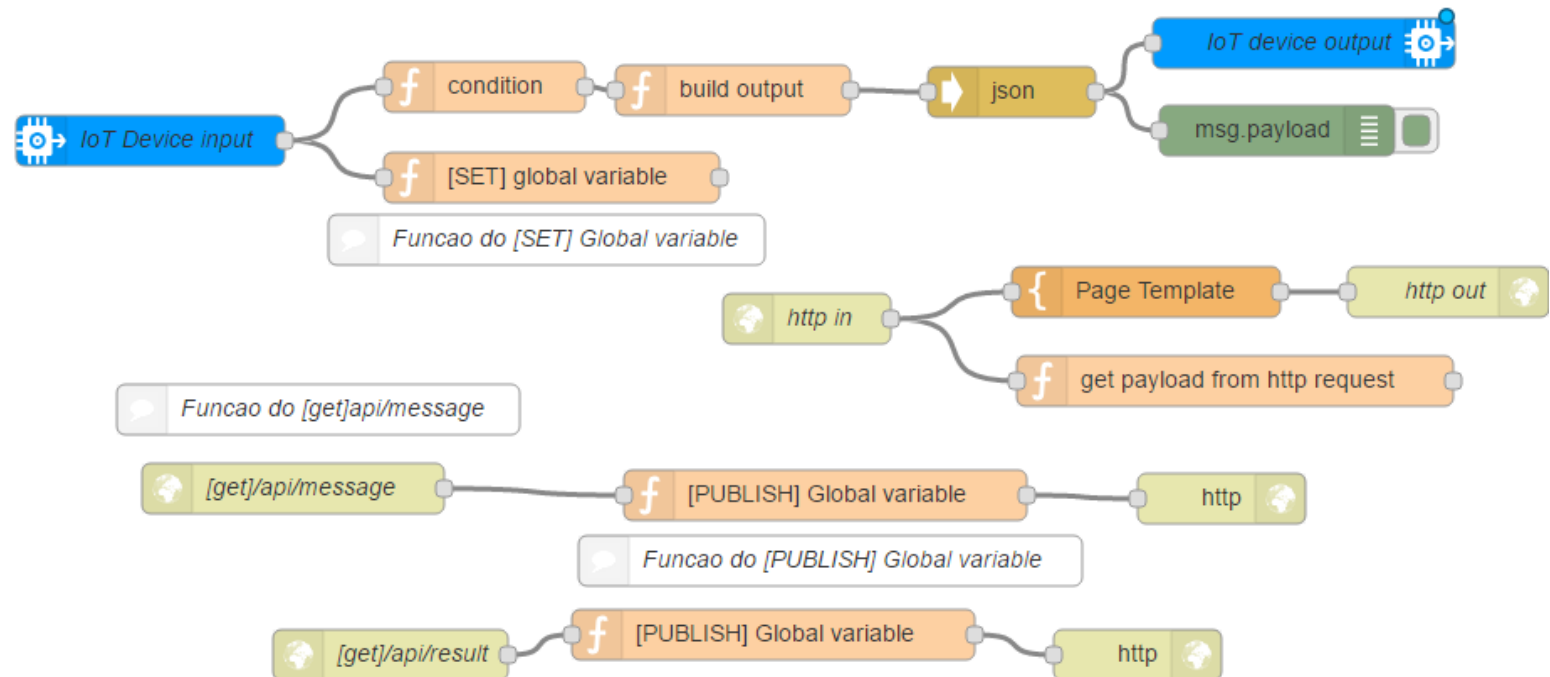
- Desta forma o cliente, a partir da função ***callbacks()***, passa a receber todas as mensagens de atualização no canal.
- Não há necessidade de manter um *loop* sobre a função de assinatura, pois o sistema *multithread* já cuida desse trabalho.

- Informações sobre a biblioteca MQTT utilizada no eclipse:

http://www.eclipse.org/paho/files/mqttdoc/Cclient/m_q_t_t_client_8h.html#aad27d07782991a4937ebf2f39a021f83

Clonando a aplicação exemplo no Node-Red

- Para copiar a aplicação basta abrir o Node-Red referente a sua aplicação IoT e colar no *clipboard* (visto na aula passada) as informações do arquivo ***webapp.json*** disponibilizado no Moodle.
- A estrutura resultante deverá estar parecida com a seguinte:
OBS: Não se esqueçam de eliminar os blocos IoT da aplicação exemplo e conectar os blocos da sua aplicação !!!



Interface para resposta via Node-Red

- Após a configuração do dispositivo IoT, é necessário configurar a aplicação no Node-Red para que os dados sejam redirecionados para o canal no qual o Galileo foi assinado.
- Dentro da página da aplicação, no Node-Red, selecione o bloco IBMIOT na aba *output*.



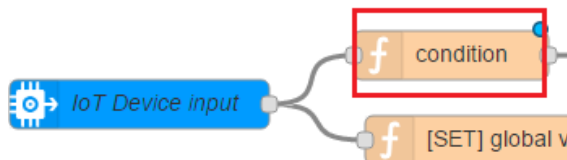
- Tal componente representa o canal de saída de dados para o dispositivo IoT conectado, desta forma, todos os dados inseridos no bloco serão enviados de volta para o Galileo. `#define TOPIC_SUBSCRIBE "iot-2/cmd/status/fmt/json"`

Command Type	<input type="text" value="status"/>	←
Format	<input type="text" value="json"/>	←
Data	<input "="" type="text" value='{"payload":{}}'/>	

- Uma das configurações mais importantes do componente de saída é o padrão dos dados de saída. Além de representar o formato (JSON) é necessário escrever a estrutura genérica de saída, representando os elementos estáticos da mesma.

Tratamento dos dados para resposta

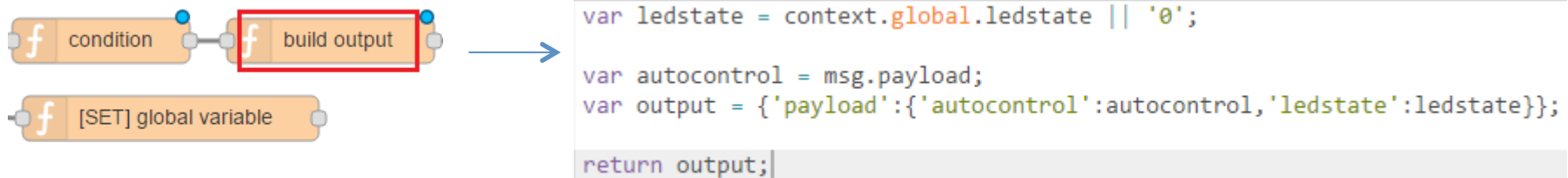
- Para efetuar a resposta para o Galileo é necessário que existam dados, estáticos ou dinâmicos.
- É interessante a possibilidade de gerar dados a partir de condicionais feitas sobre os dados de entrada, ou seja, caso um valor ultrapasse um limite a aplicação atribui o valor '1' para uma *flag*, que será enviada de volta para o Galileo, possibilitando que os dados sejam posteriormente interpretados pelo dispositivo, gerando atuações desejadas.
- Para exemplificar foi construído um bloco de função que analisa os dados de entrada, e gera uma condição booleana de acordo com o resultado. Para isso analisemos o bloco chamado '*condition*' na aplicação exemplo.



```
1 var pay = msg.payload;
i 2 msg.payload = pay['counter'];
3 var json;
4
5 ▾ if(msg.payload % 2){
6     msg.payload = '1';
7 ^ }
8 ▾ else{
9     msg.payload = '0';
10 ^ }
11 return msg;
```

Definição de estrutura para dados de resposta

- Após obtermos dados a partir dos condicionais podemos enviá-los de volta para o dispositivo IoT.
- No entanto, é necessário construir a estrutura de saída a partir dos dados obtidos. Para isso analisemos o bloco *'build output'*:



- Relembrando que o campo *'Data'* do componente de saída foi configurado com: `{"payload":{}}`.
- De maneira geral a estrutura acima representa a existência de um objeto com a propriedade *'payload'*, que tem como atributo uma outra estrutura JSON genérica (representado pelas chaves vazias).
- Dentro de tal estrutura genérica é possível incluir quantos campos forem necessários, sem que haja necessidade de alterar o cabeçalho de dados.

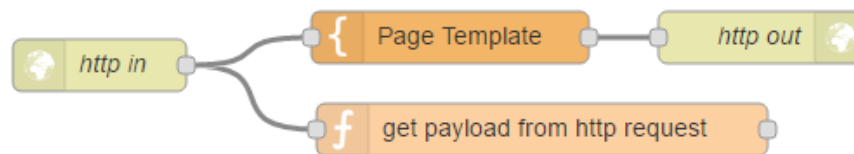
Executando a aplicação exemplo

- Como resultado da sequencia desenvolvida será possível visualizar os dados enviados pela aplicação a partir do terminal do Galileo.
- De maneira geral os dados recebidos serão 0 e 1 repetidos alternadamente.
- OBS: Os dados configurados anteriormente remetem à propriedade *'autocontrol'*. A propriedade *'ledstate'* será avaliada posteriormente.

```
Message arrived
  topic: iot-2/cmd/status/fmt/json
  message: {"autocontrol":"0","ledstate":"0"}
```

Construindo uma aplicação web para controle do dispositivo

- A aplicação web que será construída tem como intuito obter dados a partir da variação de um *slider*.
- Os dados obtidos serão enviados para o Galileo, para controlar a luminosidade de um LED.
- Para construir a aplicação Web utilizaremos conceitos de HTML, JavaScript e jQuery.
- De maneira geral a construção da aplicação é feita pelos seguintes blocos do Node-Red:



- A estrutura é dada por:
 - Requisição GET HTTP que obtém as referências para a página Web.
 - Construção de um *template* com interface gráfica(HTML) e manipulação de dados (JavaScript e jQuery)
 - Transferência dos dados introduzidos na interface para o Node-Red

Construindo aplicação web para controle do dispositivo


- Observando-se o bloco *'http in'* pode-se determinar qual o complemento da URL que disponibilizara a aplicação.
- Portanto a URL de acesso para a aplicação web será dada pelo complemento */webapp* (configurado pelo desenvolvedor).
 - *Nome_app.mybluemix.net/webapp*
- Para alterar o código referente a aquisição dos dados via HTML, ou alterar o layout da página web, basta editar o código do bloco *'Page Template'*

☰ Method	GET
🌐 URL	<u>/webapp</u>
👤 Name	http in

Slider Control

Points:

135



Enviar

Interface web da aplicação exemplo

Construindo aplicação web para controle remoto

- De maneira geral o conteúdo do bloco *'Page Template'* define:

```
function sliderMethod(){
  var sliderValue = document.getElementById("slider").value;
  var getStatement = 'http://pad-iot-cloud.mybluemix.net/webapp?ledstate=' + sliderValue.toString();
  httpGet(getStatement);
}
```

- O método para obter o valor atual do *slider* e efetuar uma requisição para atualizar o valor da variável no Node-Red

```
<form action="Javascript:sliderMethod()">
  <label for="points">Points:</label>
  <input type="range" name="slider" id="slider" value="127" min="0" max="255" data-highlight="true">
  <input type="submit" data-inline="true" value="Enviar">
  <p id="demo"></p>
</form>
```

- Construção de um formulário que define a existência de um botão do tipo *"Submit"* e um *"slider"*. E define qual o método será chamado ao clicar no botão, a partir do campo *'action'* no formulário.
- IMPORTANTE!!
 - Para que a aplicação exemplo funcione em sua organização, é necessário trocar o nome da aplicação nas URLs inseridas no bloco *"Page Template"*
 - No caso exemplo o nome da aplicação é *pad-iot-cloud*, mude-a para o nome de sua aplicação

Resultados

- Ao executar a aplicação final será possível variar o valor do slider e observar o novo valor sendo impresso no terminal do Galileo.
- Além disso, haverá o acionamento PWM da porta digital 11 no galileo, de acordo com o valor introduzido na aplicação web.
- Será necessário conectar um LED na porta digital 11 para ver o resultado prático. (O código para atuação do LED já está implementado no código exemplo para o eclipse).

```
Message arrived
  topic: iot-2/cmd/status/fmt/json
  message:
{"autocontrol":"1","ledstate":"202"}
```

```
Message arrived
  topic: iot-2/cmd/status/fmt/json
  message:
{"autocontrol":"0","ledstate":"202"}
```

- Portanto propõe-se que se utilize o código exemplo para implementar melhorias em sua aplicação do sensor de luminosidade.

Referências

- <http://mitsuruog.github.io/what-mqtt/> (websocket MQTT)
- <http://tech.scargill.net/a-node-red-websockets-web-page/>
- <https://www.ibm.com/developerworks/cloud/library/cl-rtchat-app/>
- <http://l0l.org.uk/2014/01/simple-node-red-web-page/>

- HTML
 - <http://www.w3schools.com/>
- Node-Red
 - <http://nodered.org/docs/>

Helder Rodrigues
helder.rodrigues@pad.lsi.usp.br