FIAP GRADUAÇÃO

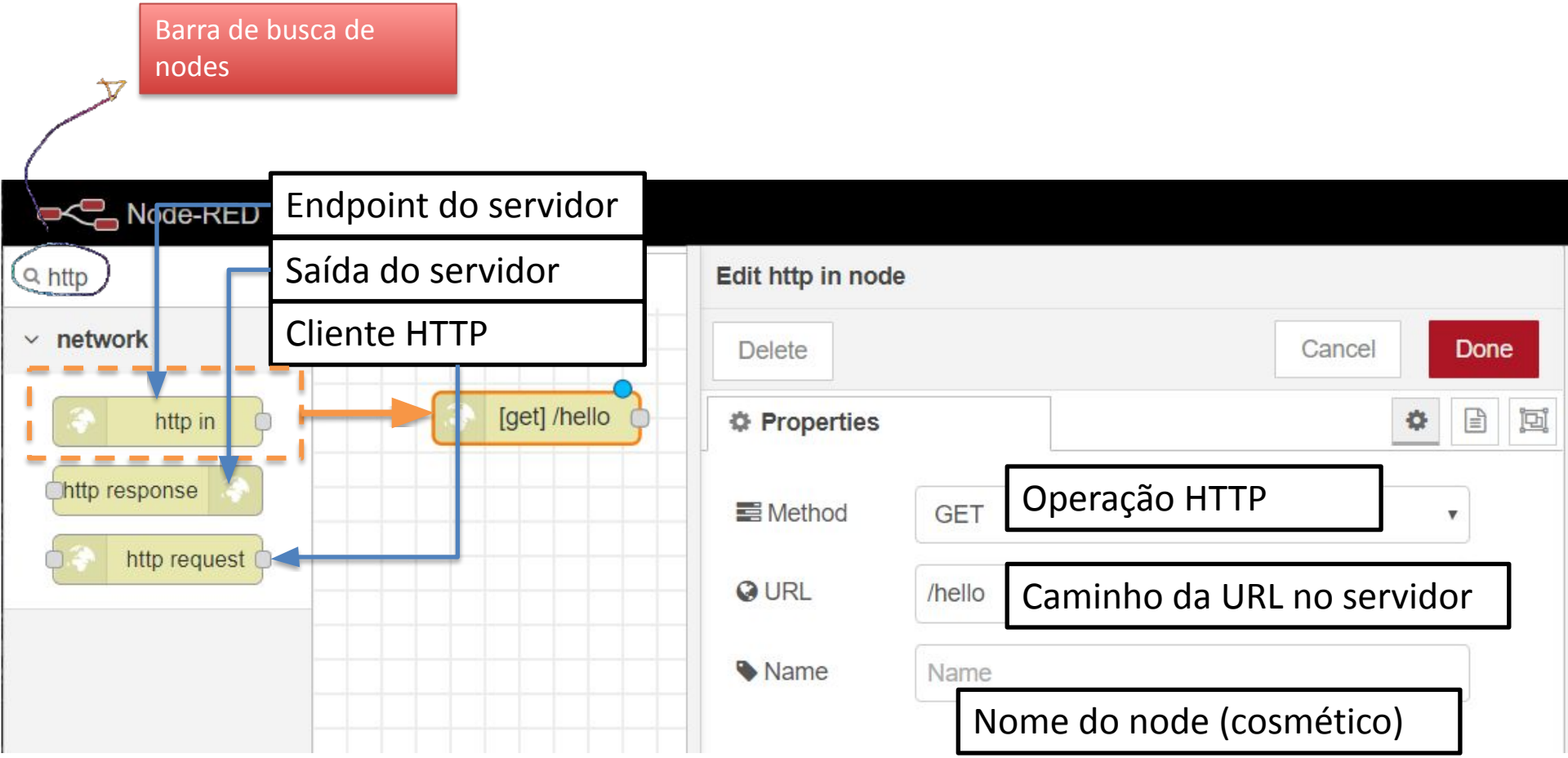
FIAP

API RESTful com Node-RED

Servidor Web Node-RED

- O próprio Node-RED é um servidor web, que escuta em geral na porta 1880, mas pode ser configurada como servidor nas portas padrão do HTTP/HTTPS
- 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

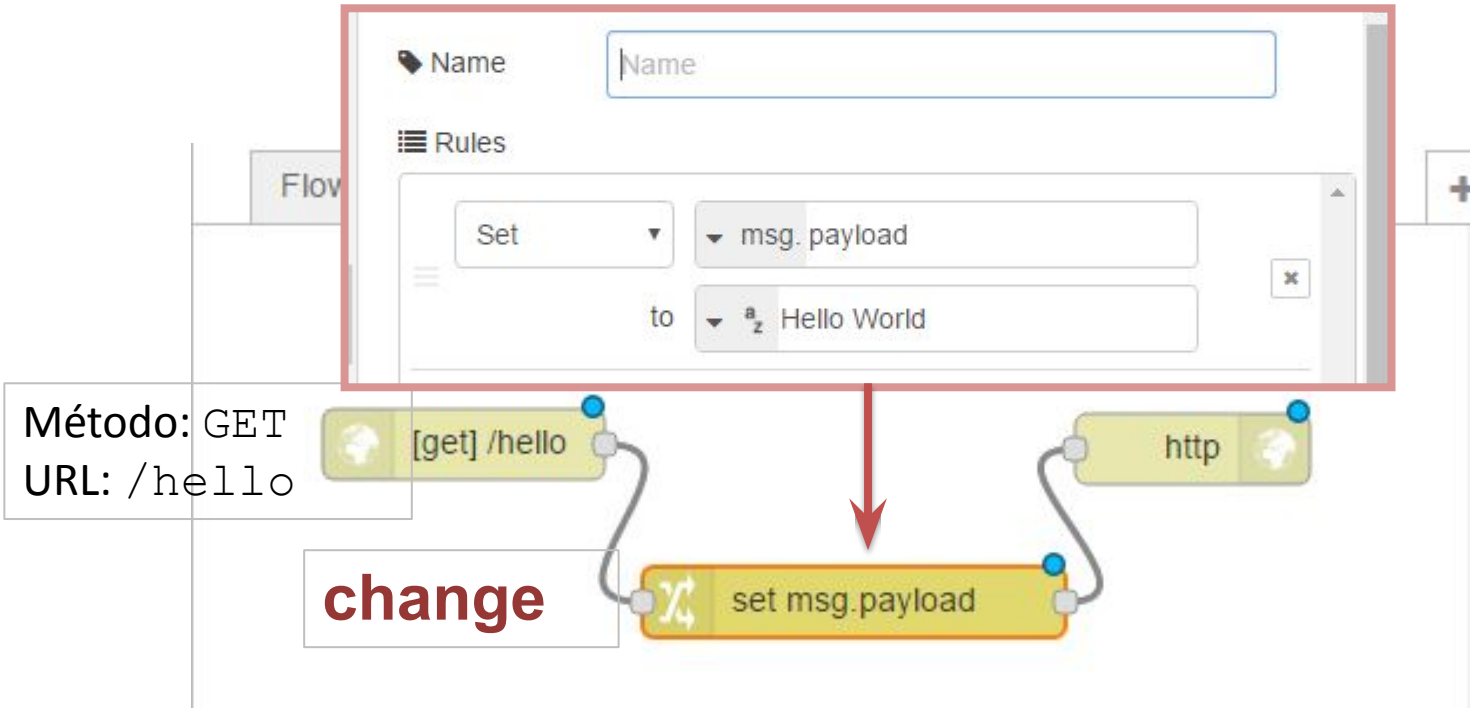
Configuração do node “HTTP In”



Aula11av

Exemplo de servidor simples

- Aqui usamos o node “**change**” para definir o valor do payload, que será enviado ao cliente HTTP



URL das APIs

- Mantemos o endereço do servidor, e alteramos apenas o caminho da URL, a partir da raiz

- Se o Node-RED estiver hospedado no próprio computador, o endereço do endpoint será `http://localhost:1880/hello`

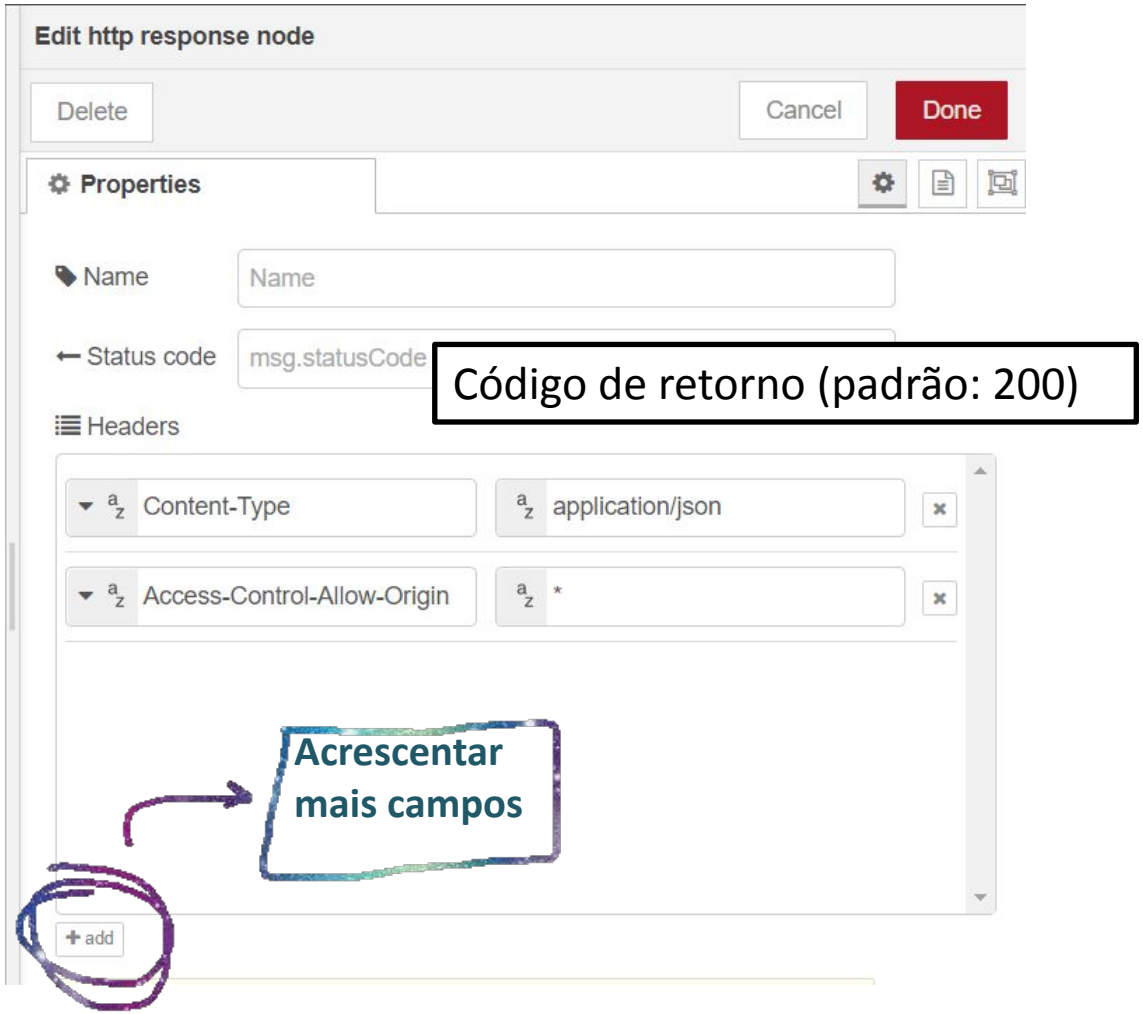
VAMOS TESTAR

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 indicar a resposta JSON e liberar as requisições *cross-site* (CORS) da nossa API, alteramos o campo `msg.headers` ou configuramos o cabeçalho da resposta como:

```
{"Content-Type":"application/json",
"Access-Control-Allow-Origin":"*"}
```

Configuração do node HTTP Response

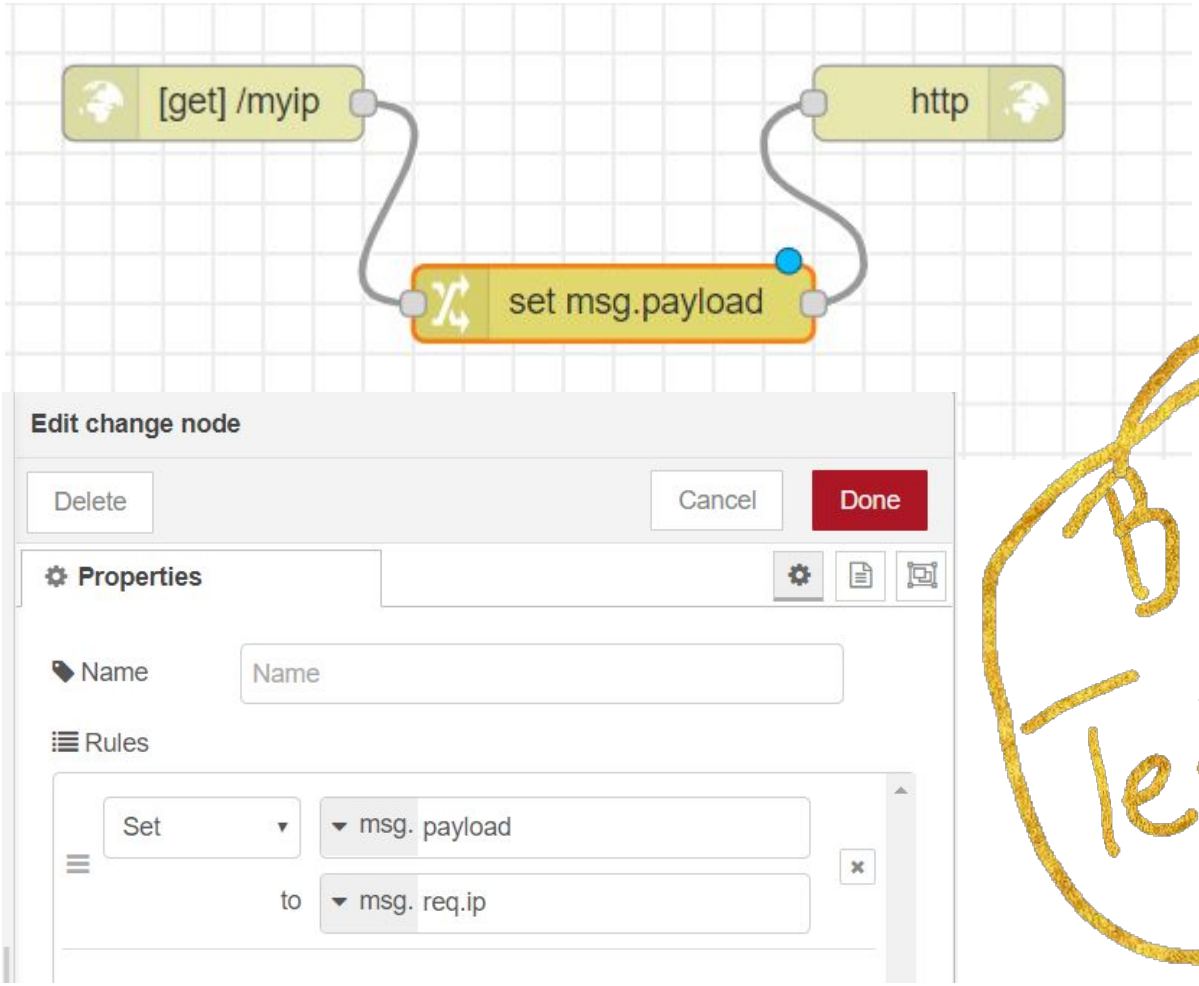


Campos do cabeçalho

Objetos de requisição e resposta

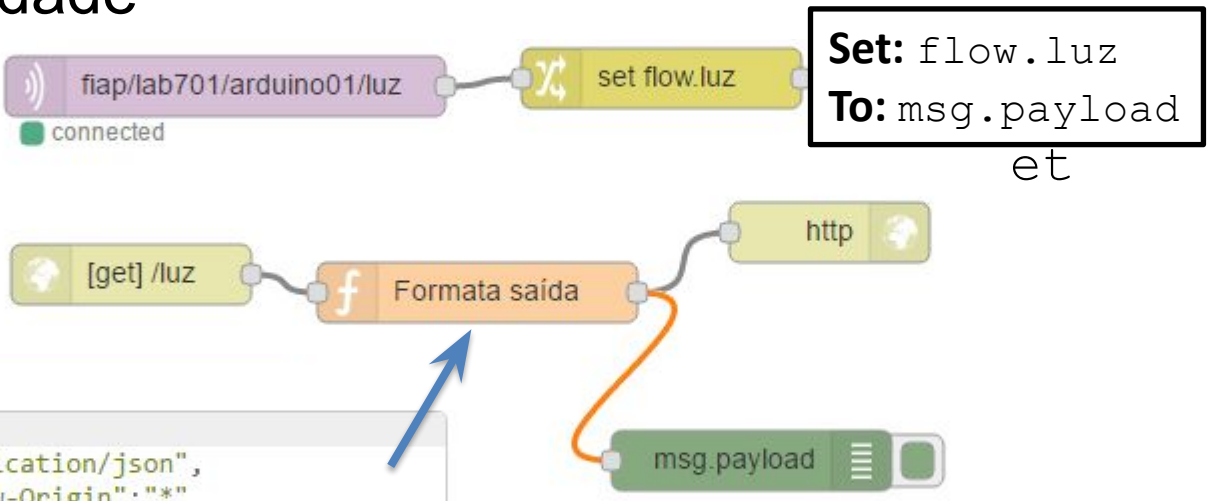
- Uma vez que Node-RED usa o pacote `express` do Node.js como servidor web, os campos `msg.req` e `msg.res` são objetos de requisição e resposta e respeitam a estrutura das classes `HTTPRequest` e `HTTPResponse` respectivamente.
- Os campos relevantes dos objetos de requisição e resposta encontram-se 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

Exemplo: responder o IP de requisição



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 escre
dentro de u



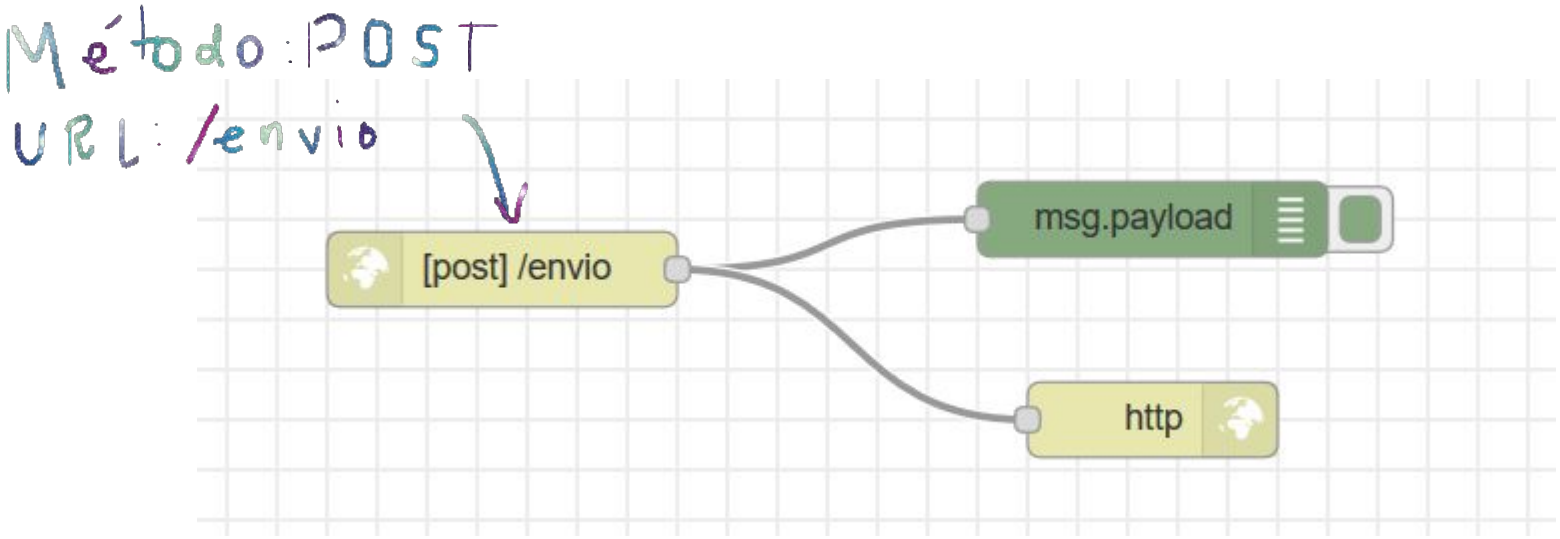
Function

```
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;
```

Recebendo informações: POST

- Para receber informações no servidor, podemos criar endpoints que tratam os métodos POST ou PUT
  - O campo `payload` carrega a informação enviada no corpo da requisição
- Não podemos esquecer de devolver uma resposta ao cliente que fez a chamada
- Para testar, usamos algum cliente de HTTP que faça requisições POST
  - **O próprio Node-RED faz isso através do node HTTP Request**

Debugando mensagens de POST

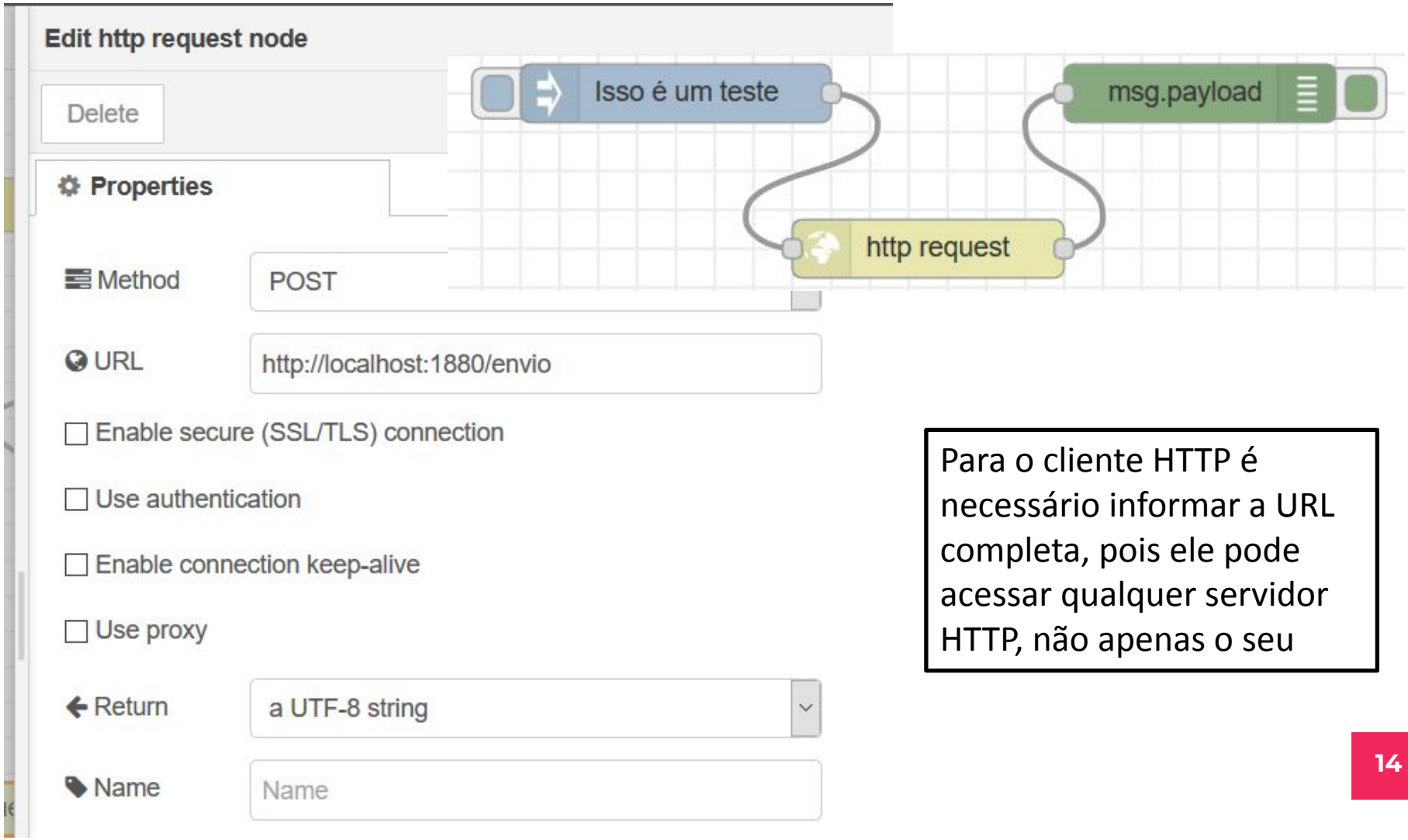


URL do endpoint

SE: Servidor ouvindo em http://localhost:1880

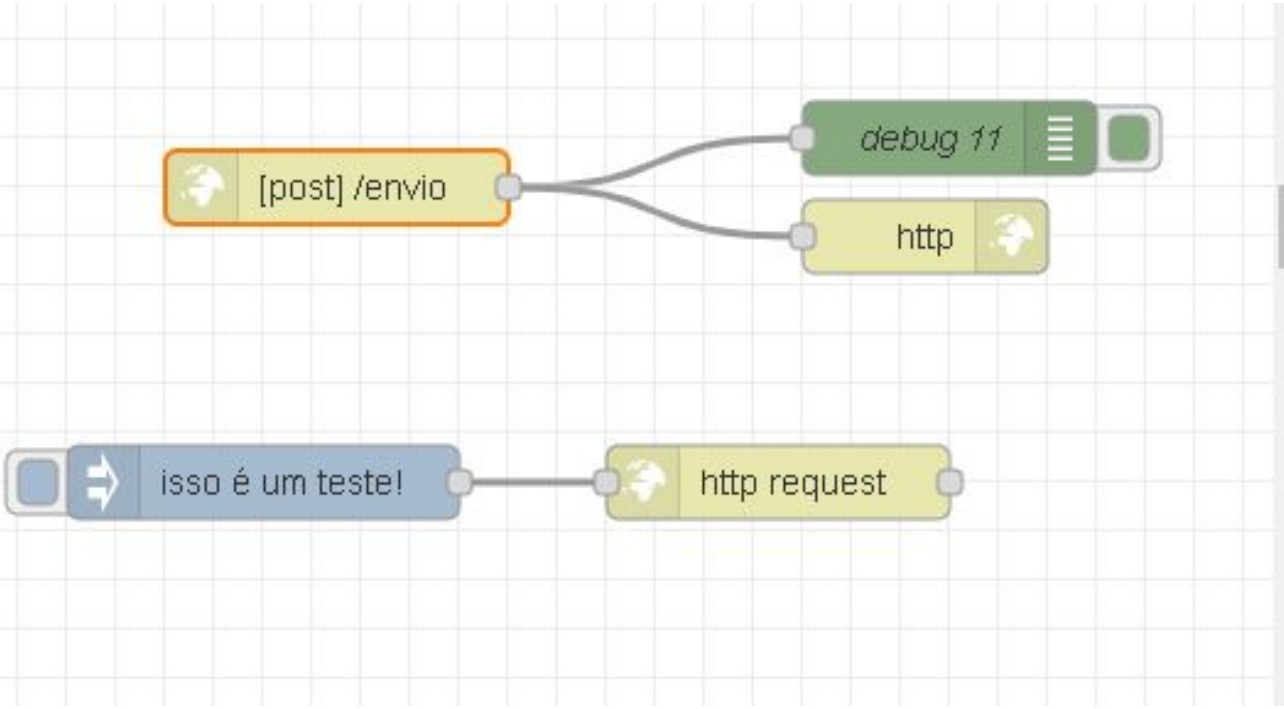
ENTÃO: Endpoint: http://localhost:1880/envio

Testando as mensagens de POST



Para o cliente HTTP é necessário informar a URL completa, pois ele pode acessar qualquer servidor HTTP, não apenas o seu

Testando as mensagens de POST



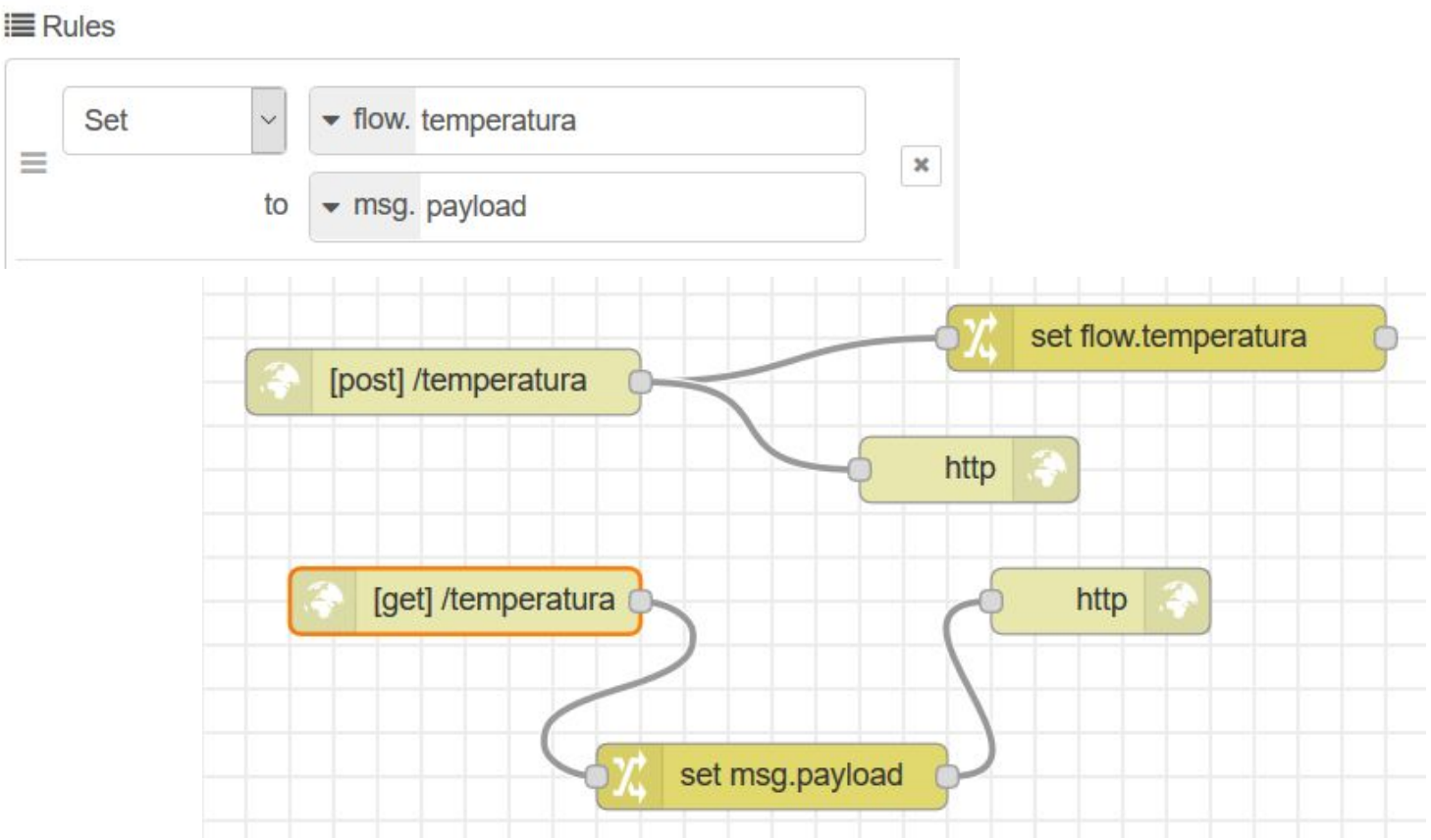
Variáveis de contexto

- Como armazenar as informações que chegam ao servidor na falta de um banco de dados?
  - As variáveis criadas em uma function são locais, ou seja, valem somente dentro da function
  - Um node pode armazenar e recuperar informações através de *contexts*, que funcionam como dicionários contendo valores de propriedades
- Há dois níveis de *contexts* que podem ser usados no Node-RED:
  - **flow:** é compartilhado por todos os nodes da mesma aba de edição
  - **global:** é compartilhado por todos os nodes do servidor

Exemplo: disponibilizando a informação recebida através da URL `/temperatura`

- Vamos criar dois endpoints:
  - Recebimento da informação de temperatura, em graus
  - Informação do último valor recebido
- Cada vez que a uma mensagem é recebida como POST, ela é armazenada dentro do *context* **flow** na propriedade “temperatura”
- Cada vez que é feita uma requisição HTTP GET na URL `/temperatura`, é retornado o último valor recebido da temperatura

Exercício: servidor de temperatura



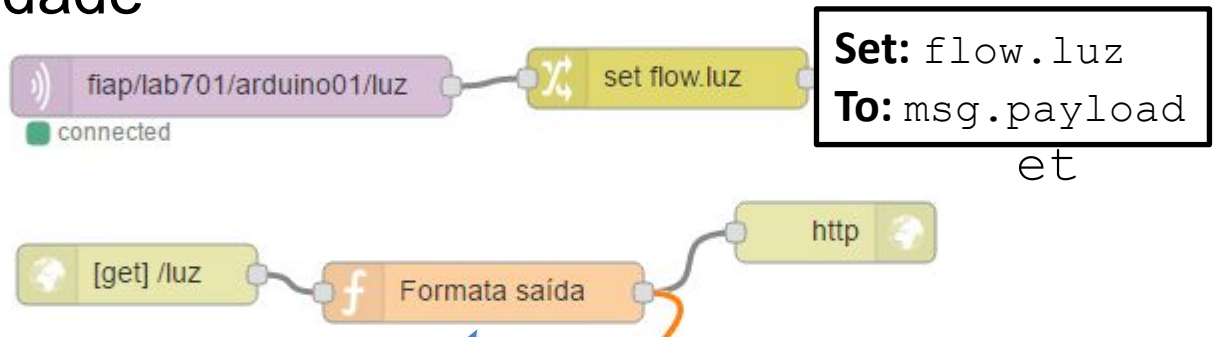
Para completar o exercício, faça a configuração do outro node “change”, e crie o programa para testar ambos endpoints

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

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 escre
dentro de u



et

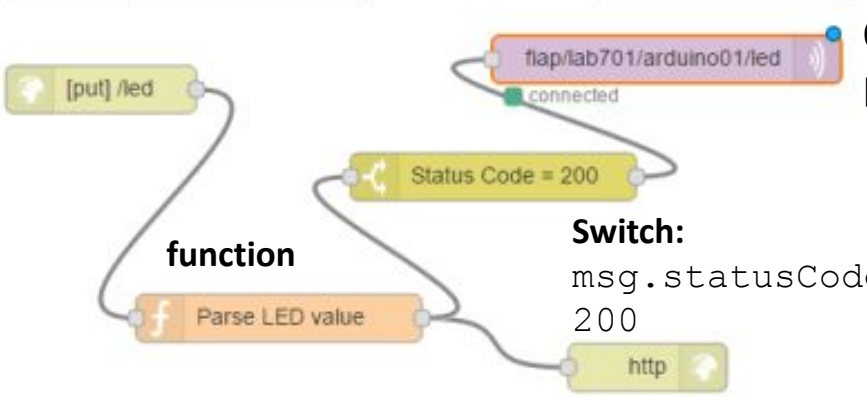
Function

```
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 `{"valor":150}` ao endereço

Processando um comando PUT com um JSON como corpo



QoS: 1
Retain: true

Switch:
`msg.statusCode == 200`

Name: Parse LED value

Function

```
if("value" in msg.payload) {
    msg.payload = msg.payload.value;
    msg.statusCode = 200;
} else {
    msg.statusCode = 400;
    msg.payload = "Bad request format";
}
return msg;
```