



API Days Nordic 2016 Tampere

Mika Karaila

Research manager

Valmet Automation

Contents

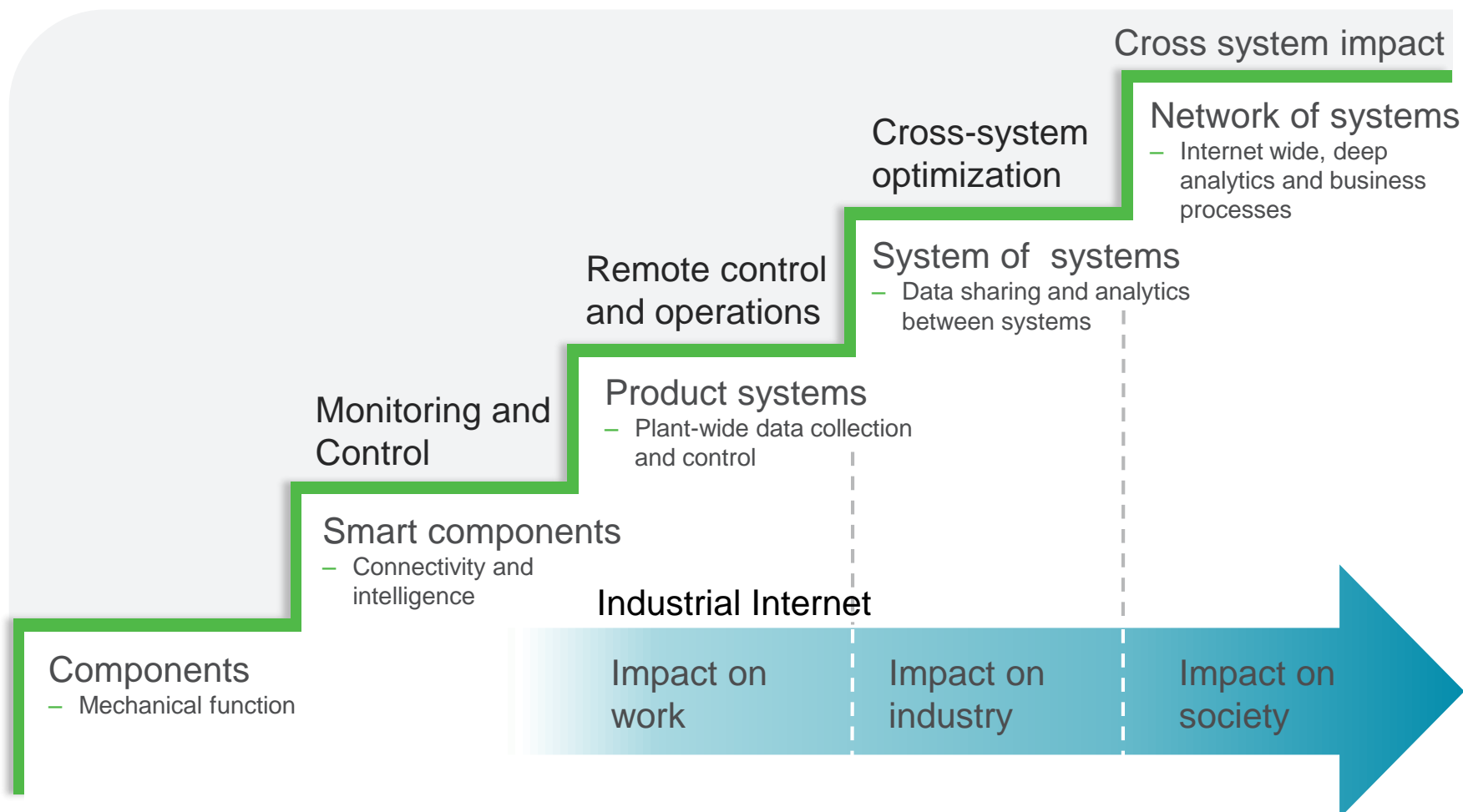
- 1 Importance of APIs for Industrial Internet
- 2 Visual flow programming: node-red
- 3 Using REST APIs and Swagger
- 4 Node-red meta programming
- 5 Summary





Importance of APIs for Industrial Internet

Evolution to Industrial Internet

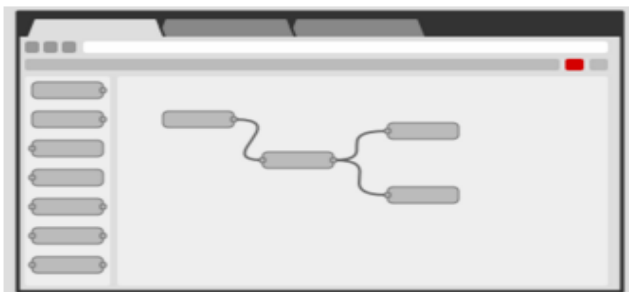




Visual flow programming

Node-red

Visual programming



Browser-based flow editing

Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range of nodes in the palette. Flows can be then deployed to the runtime in a single-click.

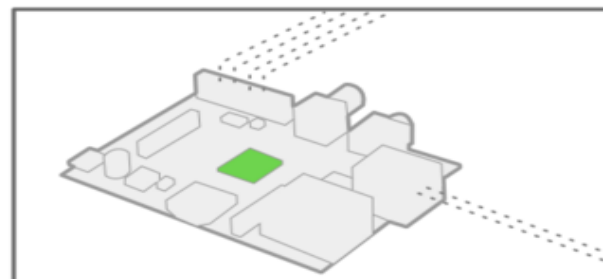
JavaScript functions can be created within the editor using a rich text editor.

A built-in library allows you to save useful functions, templates or flows for re-use.

Built on Node.js

The light-weight runtime is built on Node.js, taking full advantage of its event-driven, non-blocking model. This makes it ideal to run at the edge of the network on low-cost hardware such as the Raspberry Pi as well as in the cloud.

With over 225,000 modules in Node's package repository, it is easy to extend the range of palette nodes to add new capabilities.



Node-red principles

Add node, edit parameters

The screenshot shows the Node-RED web interface in a browser window at localhost:1880/#. The left sidebar contains a list of nodes under the 'input' category, including 'sb queue', 'sb subscription', 'inject', 'catch', 'mqtt', 'http', and 'websocket'. The main workspace shows a single 'http' node. A red arrow points to a blue dot on the top right of the 'http' node, with the text 'Blue dot: not yet deployed'. Another red arrow points to a red triangle on the bottom right of the 'http' node, with the text 'Red triangle: check parameters'. A third red arrow points to the 'Info' tab on the right sidebar, with the text 'Info: description'. The 'Info' tab displays details for the 'http in' node, including its ID 'c3fd9ece.4f654', a description of its function, and a list of properties: 'msg.req : http request' and 'msg.res : http response'. Below the properties, there is a note about parsing the body for POST/PUT requests and a 'Note' stating that the node does not send a response. The 'Edit http in node' dialog is open, showing fields for 'Method' (GET), 'URL' (/test), 'Name' (Test), and 'Docs' (add). The 'Deploy' button is visible in the top right corner of the interface.

Node-RED

localhost:1880/#

filter nodes

Sheet 1

input

sb queue

sb subscription

inject

catch

mqtt

http

websocket

Blue dot: not yet deployed

Red triangle: check parameters

Edit http in node

Method GET

URL /test

Name Test

Docs add

Info

debug

swagger

Node

Type http in

ID c3fd9ece.4f654

Properties

Provides an input node for http requests, allowing the creation of simple web services.

The resulting message has the following properties:

- msg.req : http request
- msg.res : http response

For POST/PUT requests, the body is available under `msg.req.body`. This uses the [Express bodyParser middleware](#) to parse the content to a JSON object.

By default, this expects the body of the request to be url encoded:

foo=bar&this=that

To send JSON encoded data to the node, the content-type header of the request must be set to `application/json`.

Note: This node does not send any response to the http request. This should be done with a subsequent HTTP Response node.

Deploy

Node-red example

Nodes: Inject, limit, debug

The screenshot shows the Node-RED web interface in a browser window at localhost:1880/#. The interface includes a left sidebar with a 'filter nodes' search bar and a list of nodes categorized by type (input, output, function, etc.). The main workspace, labeled 'Sheet 1', contains a flow with three nodes connected in sequence: 'Repeat timestamp/1s', 'limit 10 msg/m', and 'msg.payload'. A red arrow points from the text 'Status: msg counter value' to the 'limit 10 msg/m' node, which displays a status of '17'. Another red arrow points from the text 'Debug: timestamp value' to the 'debug' tab in the right sidebar. The 'debug' tab shows a log entry with a timestamp '4/5/2016, 9:50:40 AM' and a message 'msg.payload : number 1459839025719'.

Server parameters in one node

The screenshot shows the Node-RED web interface in a browser window. The address bar displays 'localhost:1880/#'. The interface includes a left sidebar with a 'filter nodes' search bar and a list of nodes categorized under 'input' and 'output'. The main workspace, labeled 'Sheet 1', contains a flow with three nodes: 'Repeat timestamp/1s', 'limit 10 msg/m' (with a '279' label), and 'msg.payload'. Below these, a 'Test' node is shown with a 'disconnected' status. A red arrow points from the 'Test' node to the 'Config' panel on the right. The 'Config' panel, titled 'mqtt-broker', shows a configuration for 'localhost:1883' with a value of '1'. Another red arrow points from the 'Config' panel to the 'Test' node. The 'Config' panel also has tabs for 'info', 'debug', 'swagger', and 'config'.

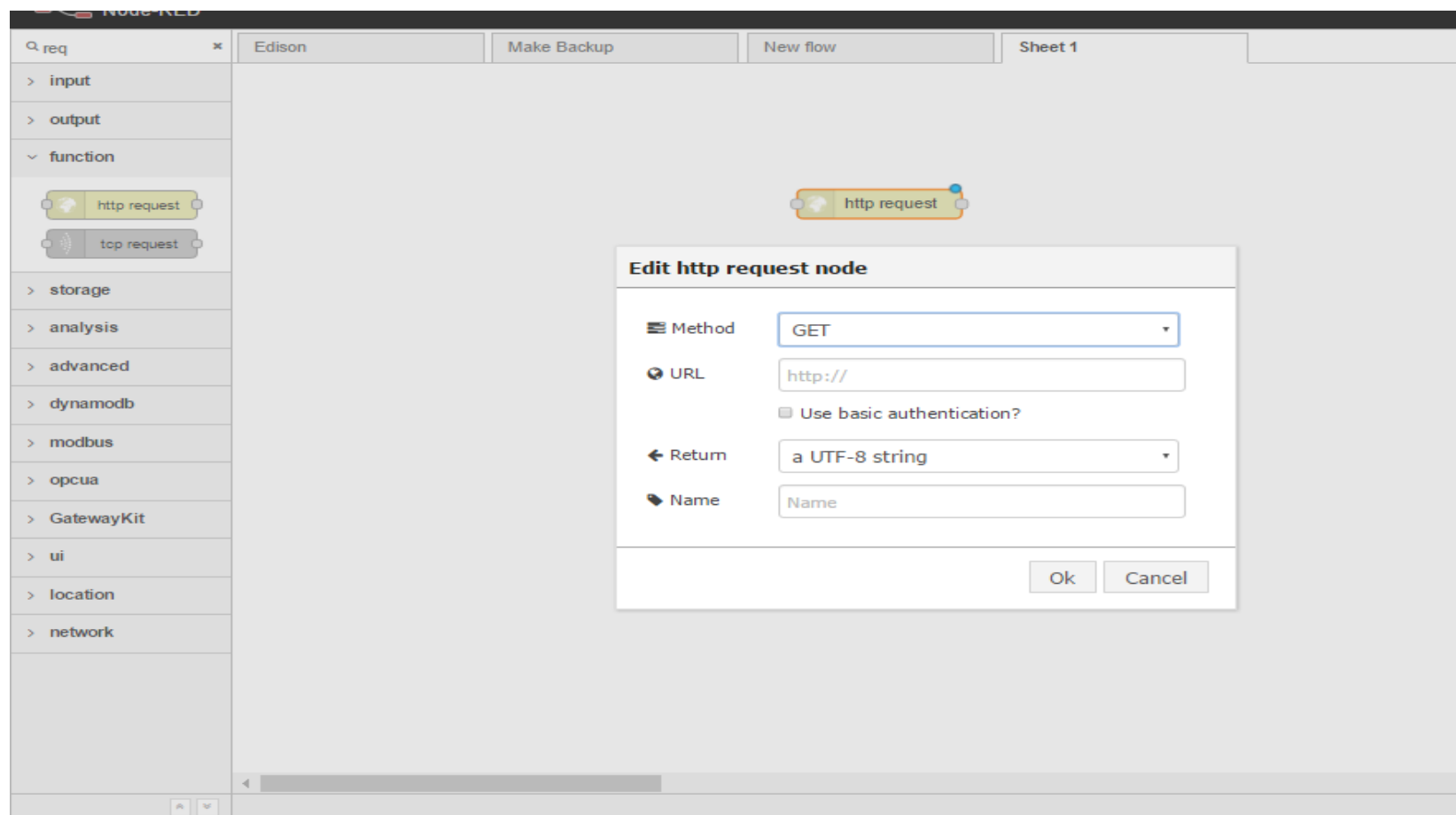
Config: keeps all parameters in one node like server IP-address & user/passwd



Using REST APIs and Swagger

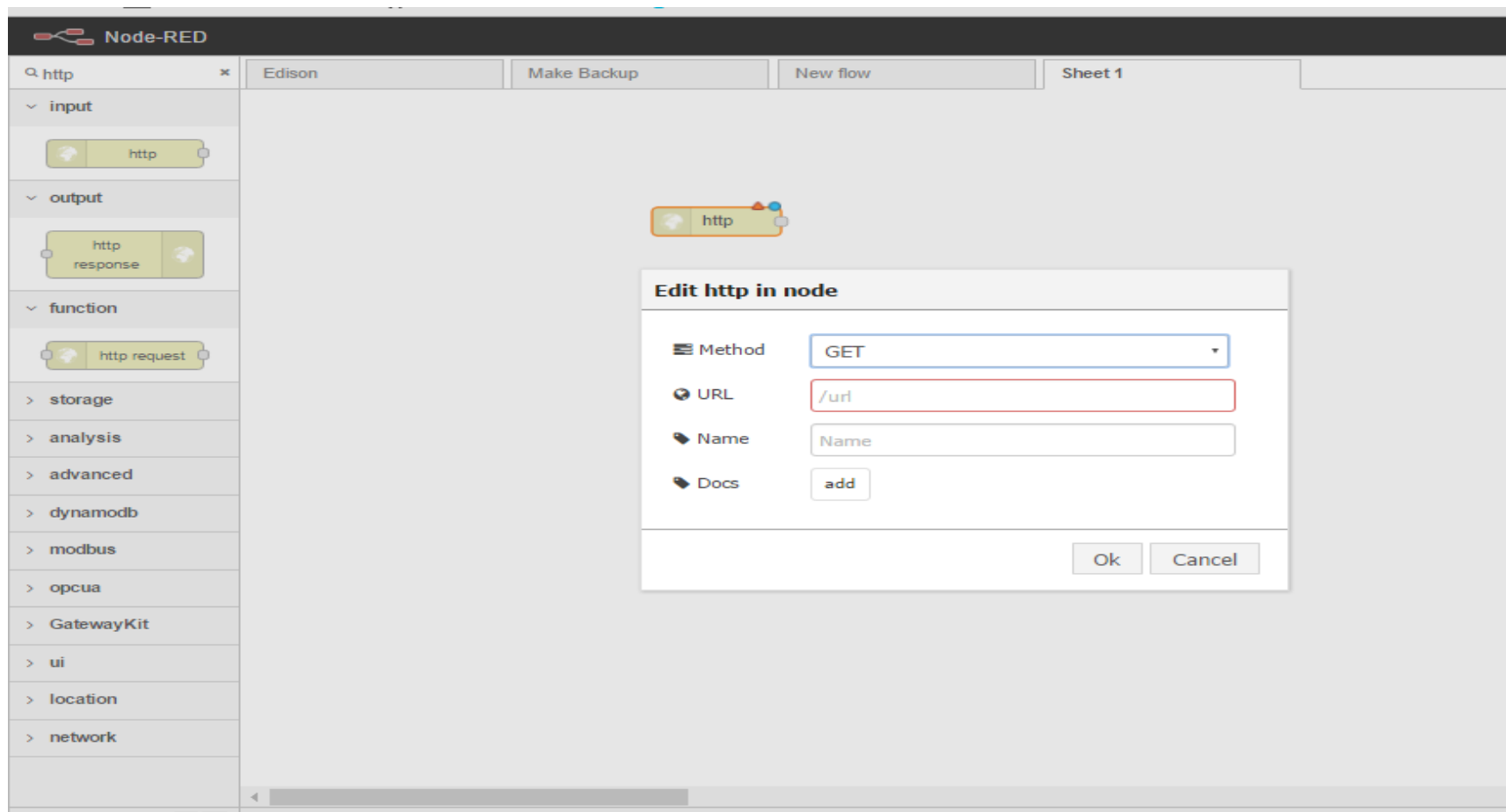
Creating REST API

http request node



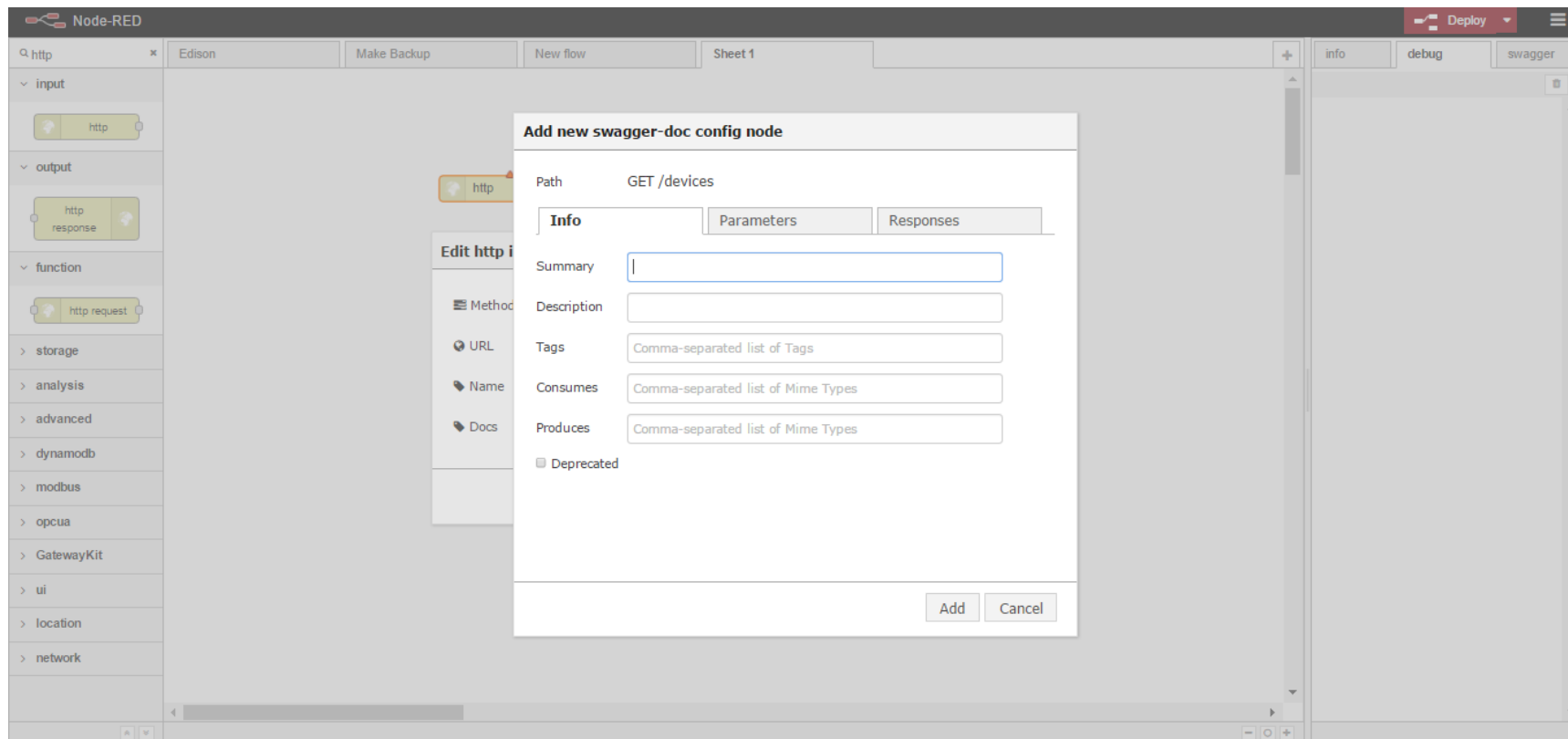
HTTP GET

Defining endpoint



Documenting endpoint

Description



Parameters for endpoint

The screenshot shows the Node-RED web interface. On the left is the palette with categories: input, output, function, storage, analysis, advanced, dynamodb, modbus, opcua, GatewayKit, ui, location, and network. The main workspace contains a flow with an 'http' node and an 'http response' node. A dialog box titled 'Add new swagger-doc config node' is open, showing the configuration for a GET /devices endpoint. The 'Parameters' tab is active, displaying a table with one parameter: 'limit' (Type: number, Description: Number of items to return). The 'in' dropdown menu is open, showing options: query, header, formData, and body. The 'query' option is selected. The dialog also includes an 'Info' tab, a 'Responses' tab, and a 'Required?' checkbox. At the bottom of the dialog are 'Add' and 'Cancel' buttons.

Deploy & use Swagger

Swagger integrated

The screenshot displays the Node-RED web interface. On the left, the 'function' category is selected in the palette, showing a 'http request()' node. The main workspace contains a flow with a 'List devices' node connected to a 'Response' node. On the right, the 'swagger' tab is active, showing the API documentation for 'My Node-RED API'.

My Node-RED API

default [Show/Hide](#) [List Operations](#) [Expand Operations](#)

GET /devices [List devices](#)

Parameters

Parameter	Value	Description	Parameter Type	Data Type
limit	10	Number of items to return	query	double

Response Messages

HTTP Status Code	Reason	Response Model	Headers
default			

[Try it out](#) [Hide Response](#)

Curl

```
curl -X GET --header 'Accept: application/json' "http://localhost:1880/devices?limit=10"
```

Request URL

```
http://localhost:1880/devices?limit=10
```

Response Body

```
{
  "limit": "10"
}
```

Response Code

```
200
```

Response Headers

```
{
  "date": "Fri, 13 May 2016 03:17:46 GMT",
  "x-content-type-options": "nosniff",
  "connection": "keep-alive",
  "x-powered-by": "Express",
  "content-length": "19",
  "etag": "W/\"13-PbHb8aI\"",
  "content-type": "application/json; charset=utf-8"
}
```

[BACK URL: /, API VERSION: 0.0.1]

Ready to use & test

The image shows a Swagger UI interface for an API named "My Node-RED API". The interface is in the "default" tab, which is also the "Info" tab. The "debug" and "swagger" tabs are also visible. The main content area displays the details for the "GET /devices" endpoint, which is described as "List devices".

Parameters

Parameter	Value	Description	Parameter Type	Data Type
limit	10	Number of items to return	query	double

Response Messages

HTTP Status Code	Reason	Response Model	Headers
default			

[Try it out!](#) [Hide Response](#)

Curl

```
curl -X GET --header "Accept: application/json" "http://localhost:1880/devices?limit=10"
```

Request URL

```
http://localhost:1880/devices?limit=10
```

Response Body

```
{
  "limit": "10"
}
```

Response Code

```
200
```

Response Headers

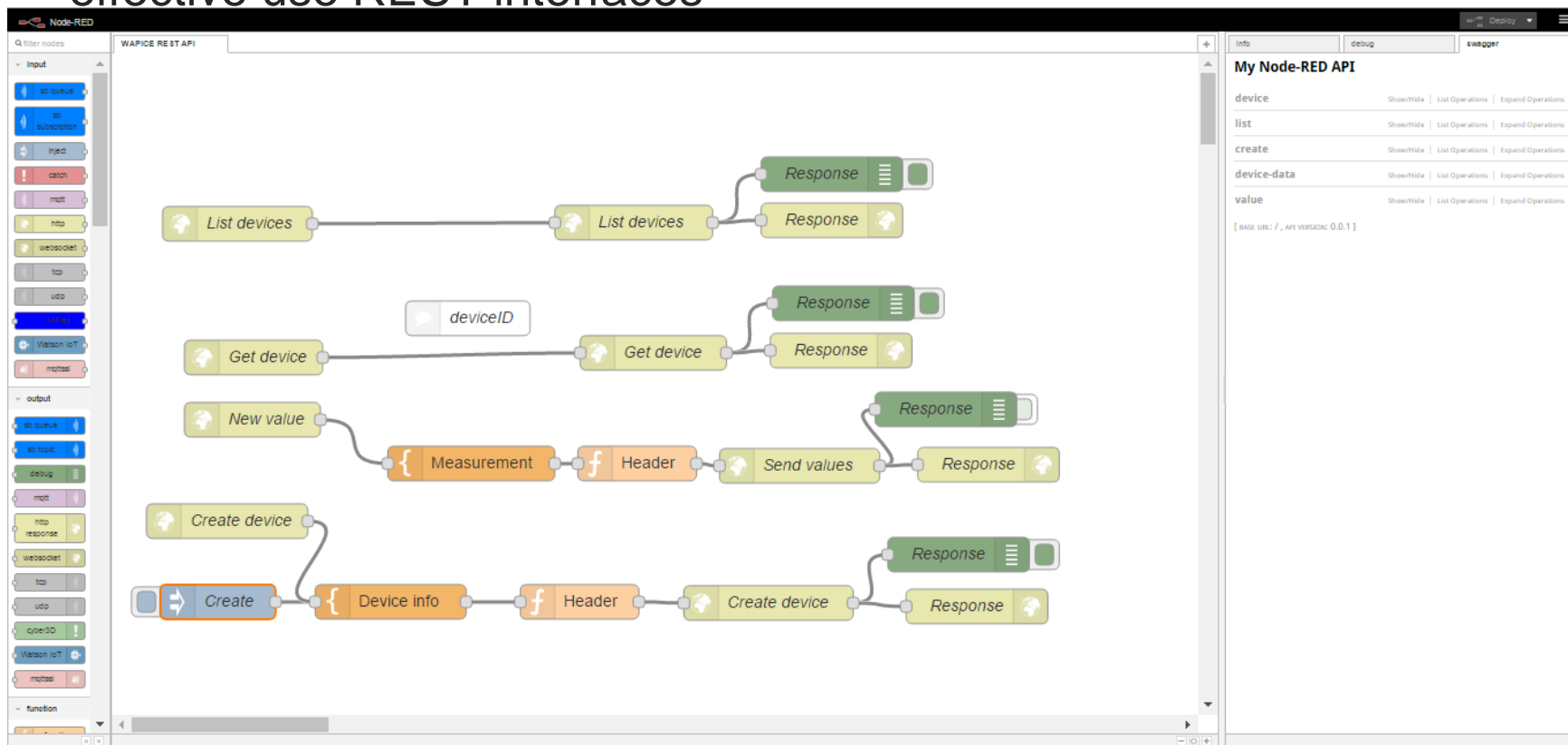
```
{
  "date": "Fri, 13 May 2016 03:17:44 GMT",
  "x-content-type-options": "nosniff",
  "connection": "keep-alive",
  "x-powered-by": "Express",
  "content-length": "19",
  "etag": "W/11-P0c0BdSa",
  "content-type": "application/json; charset=utf-8"
}
```

[BASE URL: / , API VERSION: 0.0.1]

DEMO

Live demo with Wapice IoT-Ticket: swagger.json

- Demonstrates how to use node-red with swagger to simplify and effective use REST interfaces





Node-red meta programming

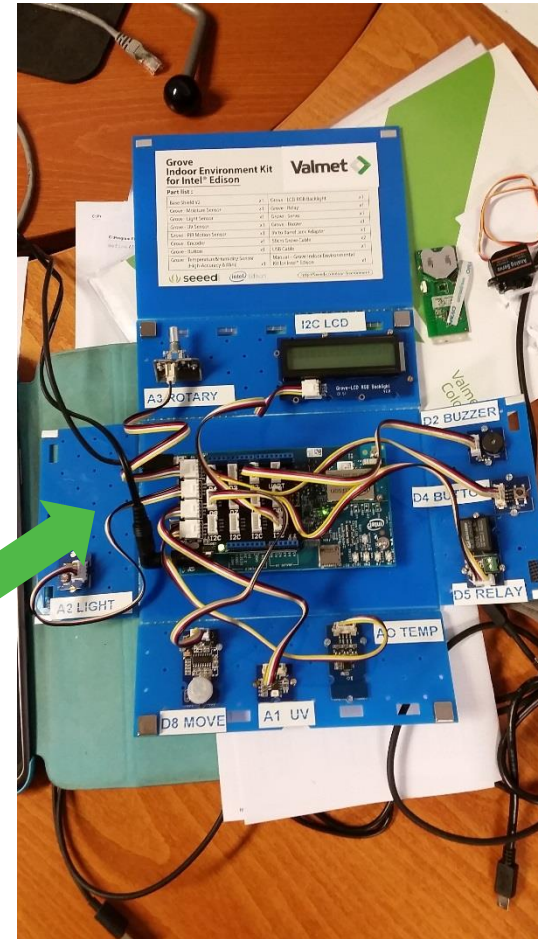
Meta-programming

Program that creates a program

- Node-red supports admin level REST interface
 - Endpoint to GET / POST to access flows
- Ideal for managing & testing larger amount of nodes

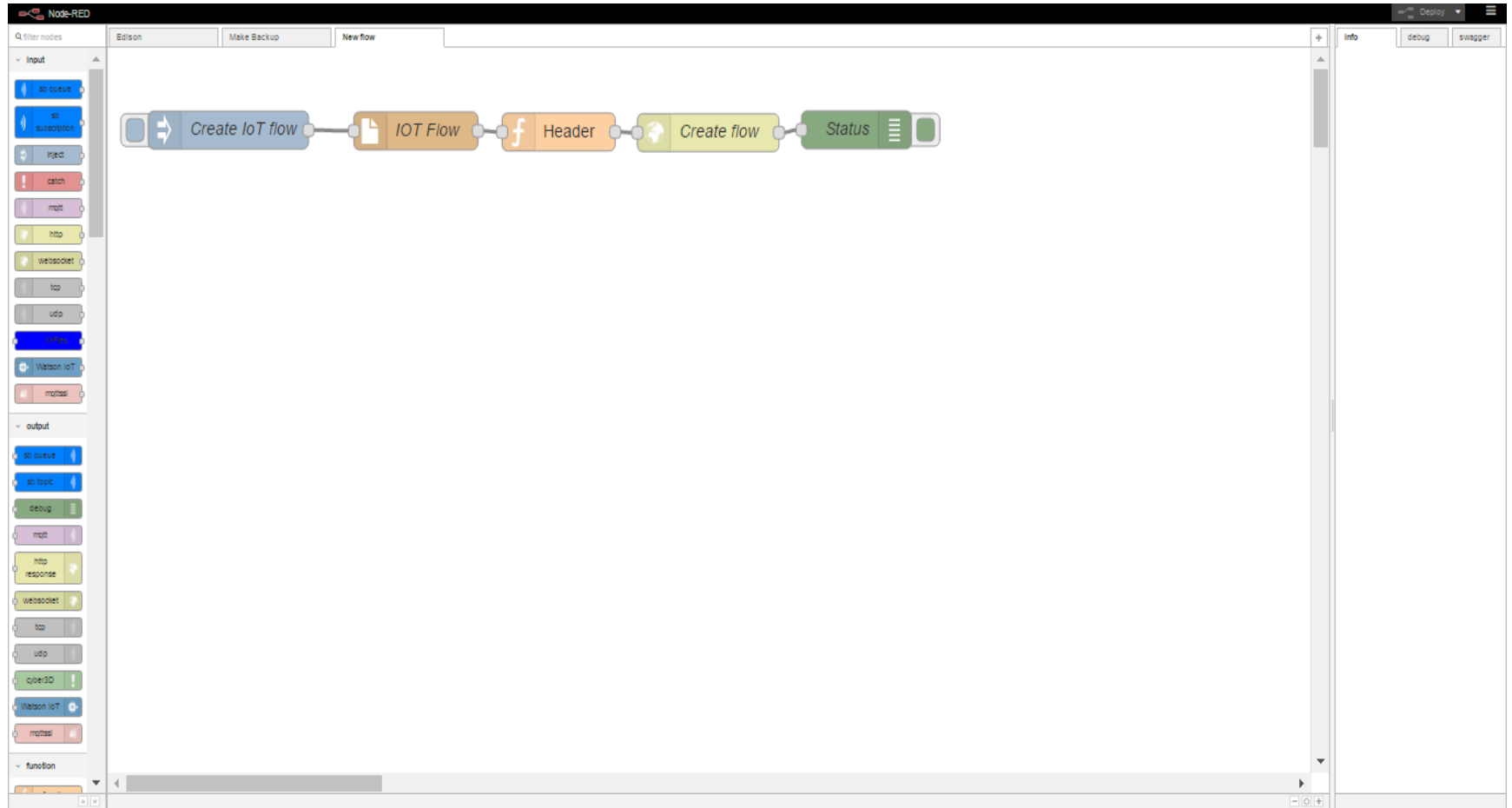
Demo hardware: Intel Edison + Groove kit

1. Intel Edison into base board
2. Base board into enclosure:
4 x metal lifters + 8 x screws
3. Arduino sensor adapter
4. Sensors:
2 x screws + 2 x nuts
4 x washers



Leave space for connectors:
Power line + USB-debug

Simple flow that will send “program” to Edison



Demo summary

- Same programming environment
- Simple REST APIs
- Effective and very easy to reuse => high productivity



Summary

Highlights

- Document as you program
- Test immediately
- Easiness & productivity
- Have fun !

Questions & discussion

