



iii PRIDE  
資策會 40 週年

# Chameleon Node-RED 教育訓練

講師: Felix

felixchiu@iii.org.tw  
www.iii.org.tw



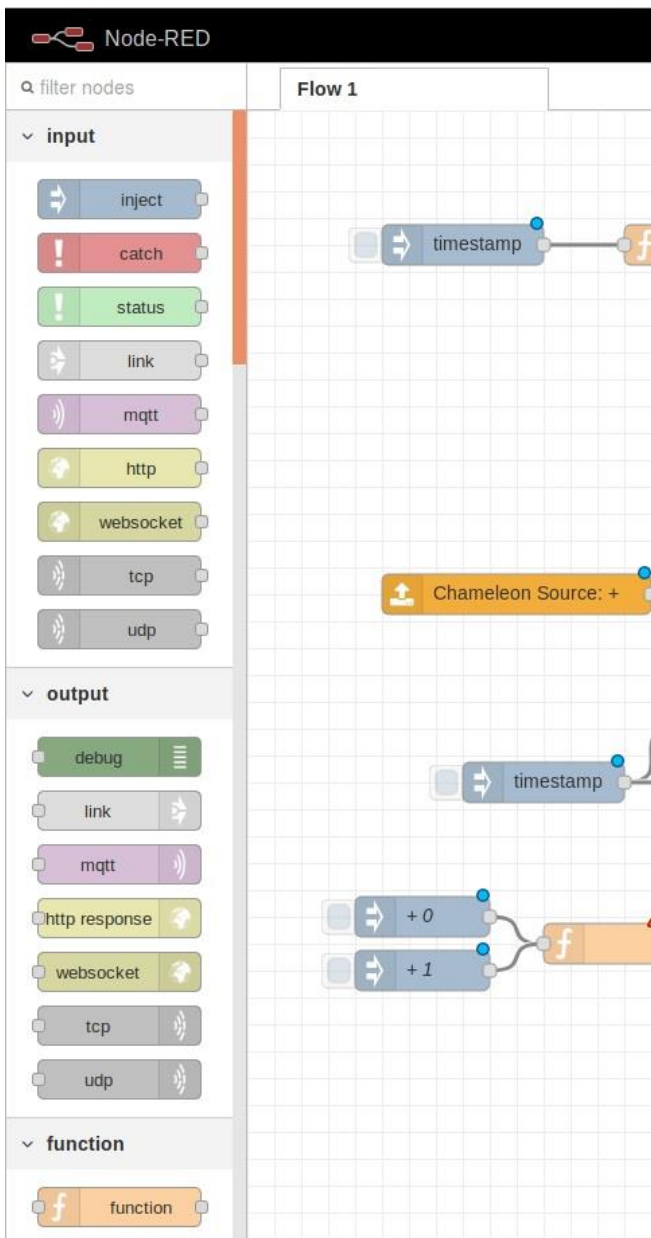
資訊工業策進會 Institute for Information Industry

v.1.2.0



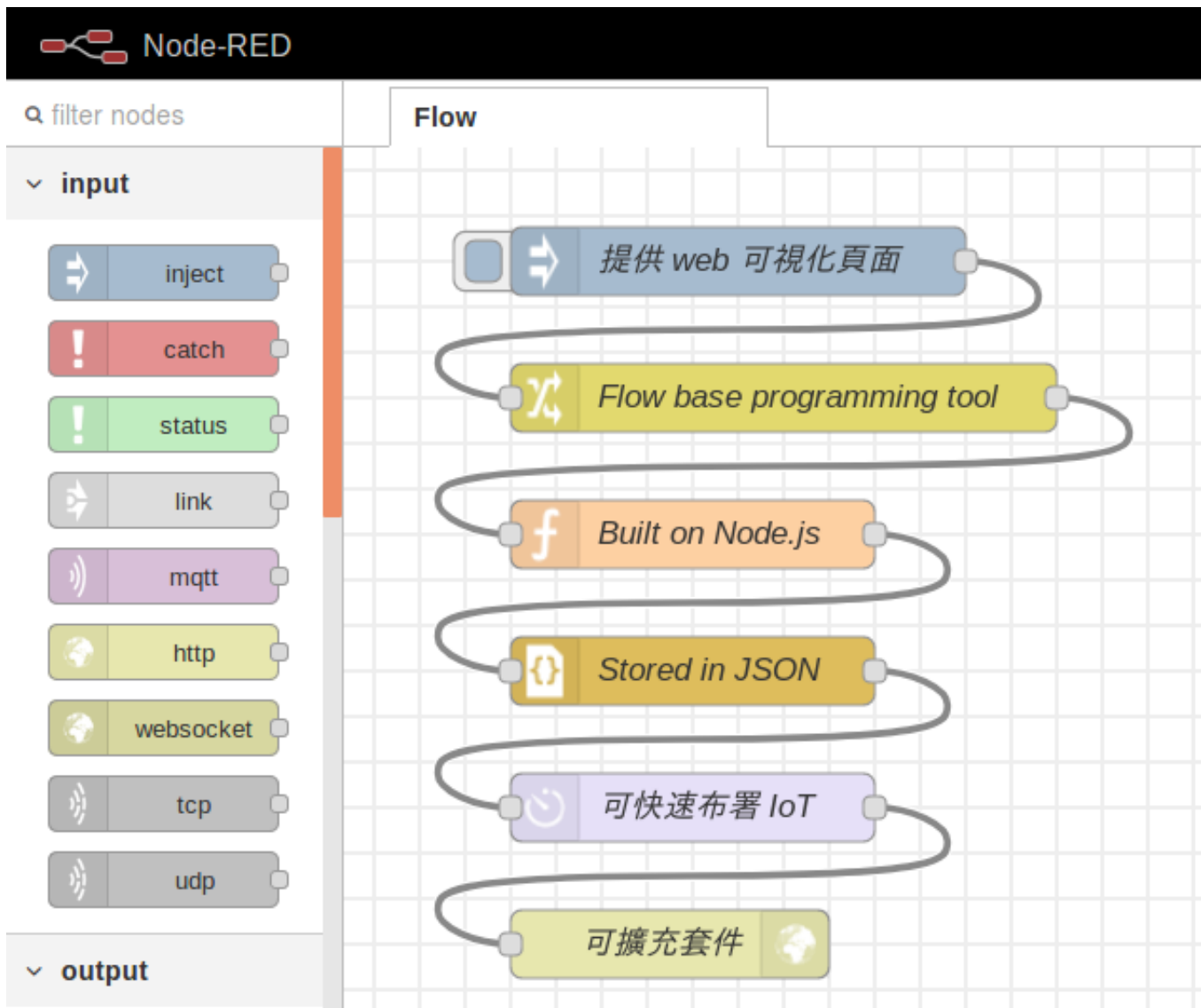
# Agenda

1. Node-RED 是什麼?
2. 基本操作
3. 常用 node 介紹
4. Chameleon x Node-RED
5. 練習 & QnA





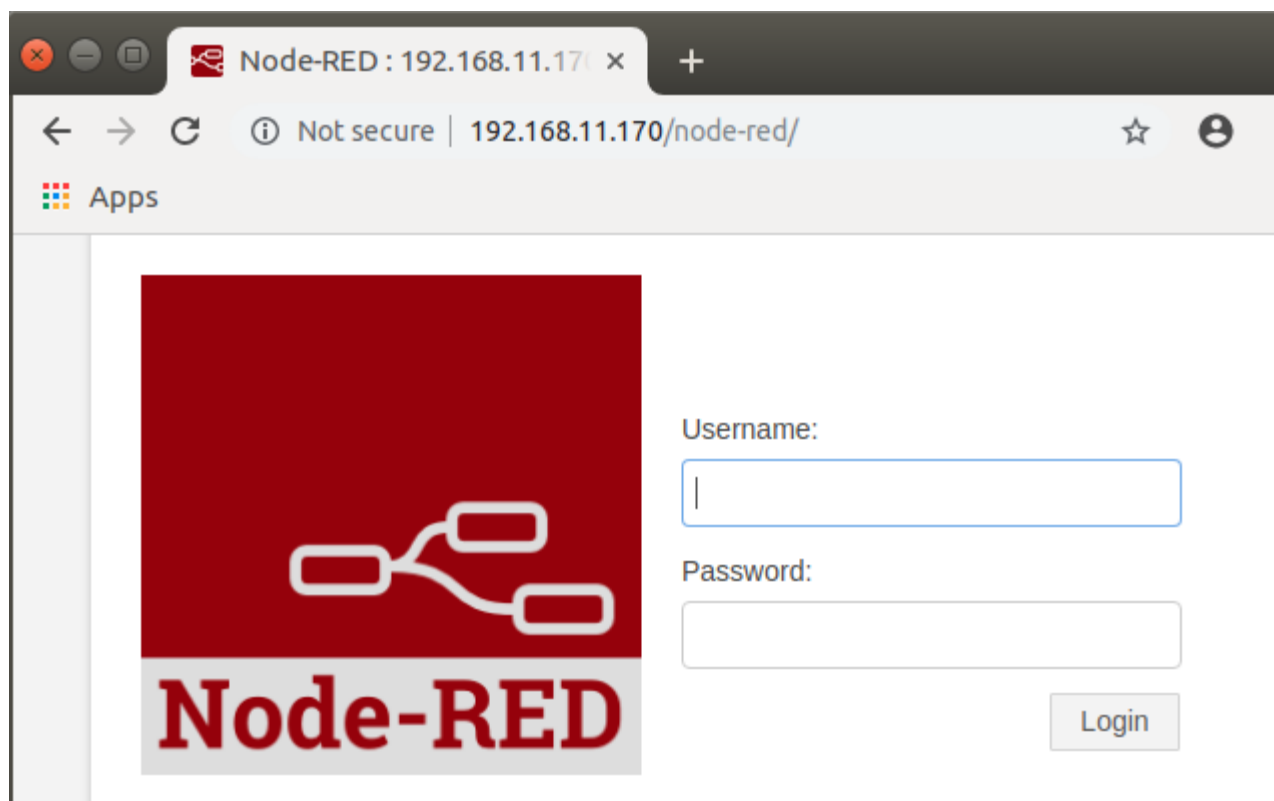
# What is Node-RED





# 如何開啟 Node-RED

- 變色龍的 URL + /node-red
- 帳號密碼和變色龍的相同





# 基本介紹

The screenshot shows the Node-RED web interface. On the left, a sidebar contains a search bar and two categories of nodes: 'output' (debug, link, mqtt, http response, websocket, tcp, udp) and 'function' (function, template, delay). A red box highlights the entire sidebar area, with a red arrow pointing to it labeled 'node'. The main workspace shows a flow named 'Flow 1' with a red arrow pointing to the title bar labeled 'flow'. Inside the workspace, a sequence of nodes (inject, function, msg.payload) is connected by wires. A red box highlights this sequence, with a red arrow pointing to it labeled '這也是 flow'. A red arrow points to the '+' button in the top right of the workspace, labeled '新增 flow'. On the right, a sidebar shows the 'info' tab with a red arrow pointing to the 'i' icon labeled 'node 訊息'. Below it, the 'debug' tab is selected, showing a table of flow information. A red arrow points to the 'debug' tab icon, labeled 'debug 訊息'.

Node-RED

Flow 1

flow

node

新增 flow

這也是 flow

node 訊息

debug 訊息

Information	
Flow	"adb52fe4.95b53"
Name	Flow 1
Status	Enabled

Description

Dragging a node onto a wire will splice it into the link



# 編輯 flow

Node-RED

Filter nodes

output

- debug
- link
- mqtt
- http response
- websocket
- tcp
- udp

Flow 1 點2下

Edit flow: Flow 1

Delete 刪除 flow

Cancel Done

Name Flow 1 改 flow 名稱

Status ☒ Enabled enable / disable

Description

h1 h2 h3 B I </> [List Icon] [List Icon] [Quote Icon] [Minus Icon] [Link Icon] [Share Icon]

1



# 編輯 node

**double click**

**Edit trigger node**

Delete Cancel Done

⚙️ Properties 📄 🖼️

Send ▼ a<sub>z</sub> 1

then wait for

250 Milliseconds

☐ extend delay if new message arrives

then send ▼ a<sub>z</sub> 0

Reset the trigger if:

- msg.reset is set
- msg.payload equals optional

Handling all messages

🔑 Name Name



# 儲存

Apps

修改完記得儲存

Node-RED

filter nodes

Flow 1

output

- debug
- link
- mqtt
- http response
- websocket
- tcp
- udp

function

- function
- template
- delay

Flow 1 diagram:

```
graph LR; inject[inject] --> function[function]; function --> msg_payload[msg.payload];
```

info

Information

Flow	"adb52fe4.95b53"
Name	Flow 1
Status	Enabled

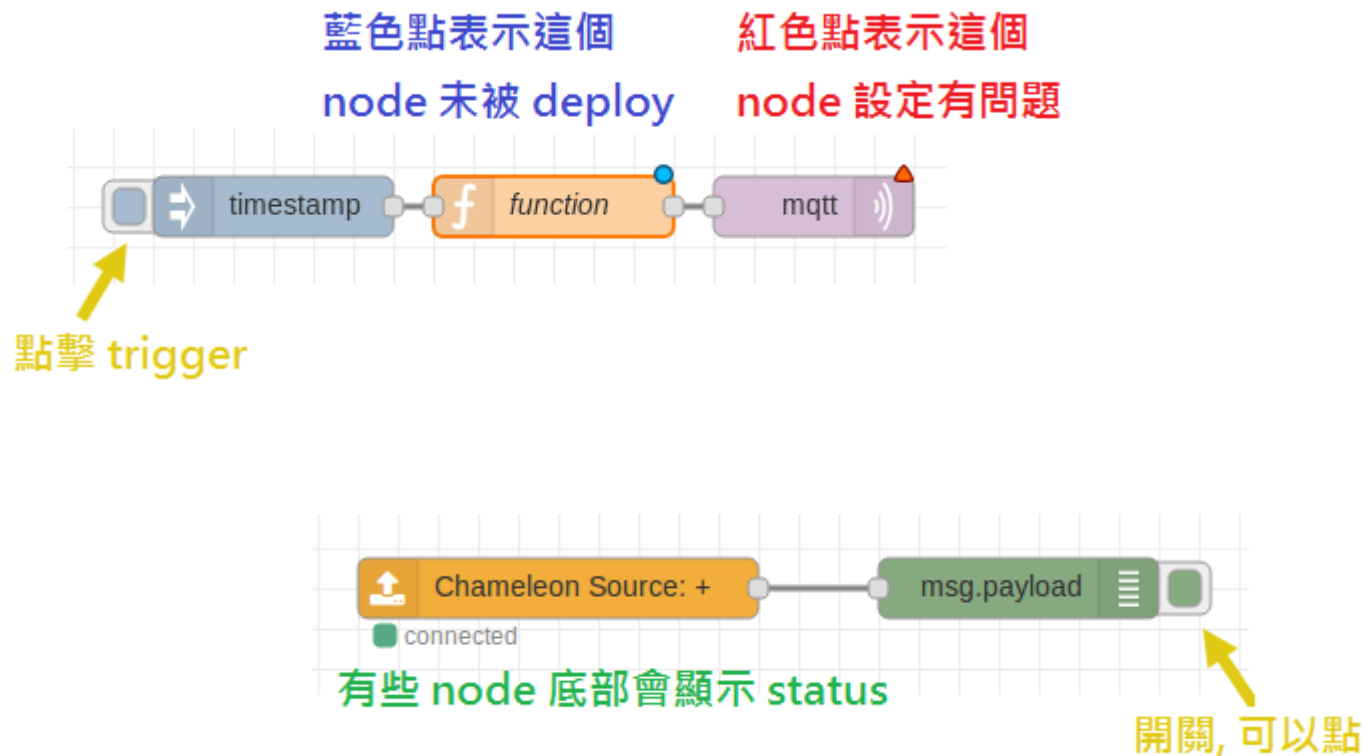
Description

Dragging a node onto a wire will splice it into the link





# node status





# 匯出 flow or node

The screenshot illustrates the process of exporting a flow or node in Node-RED. On the right, the main menu is open, with the 'Export' option highlighted. A red arrow points from the 'Export' menu item to the 'Export nodes' dialog box on the left. The dialog box has a title 'Export nodes' and three tabs: 'selected nodes', 'current flow', and 'all flows'. The 'selected nodes' tab is active, showing a JSON snippet: 

```
"x": 360,
"y": 240,
"wires": []
```

 with the text 'json 格式' (JSON format) next to it. At the bottom of the dialog, there are three buttons: 'Cancel', 'Download', and 'Export to clipboard'. The 'Download' and 'Export to clipboard' buttons are circled in red. A red arrow points from the 'Export to clipboard' button to the text '複製到剪貼簿' (Copy to clipboard). Another red arrow points from the 'Export' menu item to the 'Clipboard' option in the main menu.

選擇的 node / 這個 flow / 所有 flow

Export to clipboard selected nodes current flow all flows

json 格式

下載 json 檔

複製到剪貼簿



# 匯入 flow or node

Node-RED

Flow 1

filter nodes

output

- debug
- link
- mqtt
- http response
- websocket
- tcp
- udp

function

- function
- template
- delay

Import nodes

Paste flow json or

Clipboard

Library

Examples

select a file to import

或是這邊貼上之前匯出的 json string

Import to

current flow

new flow

Cancel

Import

View

Import

Export

Search flows

Configuration nodes

Flows

Subflows

Manage palette

Settings

Keyboard shortcuts

Node-RED website

v0.20.7

匯入

可以選擇上傳 json 檔案



# node & message

- node 是最基本的元件
- node 之間使用線連接來組成 flow
- node 與 node 間傳遞使用 message, 程式中使用 **msg** variable 來取得訊息
- msg 的 type 是 JavaScript objects, 其中 **payload** property 是最常用來放要給下個 node 的資料
  - msg.payload
  - payload 內容可以是任何的 JavaScript type



# 常用 node 介紹

- inject
- debug
- trigger
- delay
- function
- switch
- json
- http request
- mqtt



# inject & debug node

The screenshot shows the Node-RED web interface. In the center workspace, an 'inject' node is connected to a 'debug' node. The 'inject' node has a dropdown menu open for its 'Payload' property, showing options like 'flow.', 'global.', 'string', 'number', 'boolean', 'JSON', 'buffer', 'timestamp', and '\$ env variable'. The 'debug' node on the right shows the output: '9/26/2019, 2:49:52 PM node: 36af48b4.058dd8' and 'msg.payload : string[24]' followed by the message 'this is from inject node' in red text. A red arrow points from the 'inject' node to the 'debug' node's output, with the text '善用 debug node 來知道 msg 內容' (Use debug node to know msg content) next to it. Another red arrow points from the 'inject' node to its 'Edit inject node' dialog box, which is open on the left. The dialog shows the 'Payload' field with the value 'this is from inject node' and a 'Repeat' interval of '0.1 seconds, then'.

- inject:
  - 產生 string, json...等資料帶給下個 node
  - 設定每隔一段時間 inject 一次
- debug: 查看 msg 內容
- ref: [example/inject\\_and\\_debug.json](#)



# trigger node

The screenshot displays the Node-RED web interface. In the workspace, a flow consists of three nodes: a 'timestamp' node, a 'trigger' node (highlighted with an orange border), and a 'msg.payload' node. A red arrow points from the 'trigger' node to the 'Edit trigger node' dialog box. The dialog box shows the following configuration:

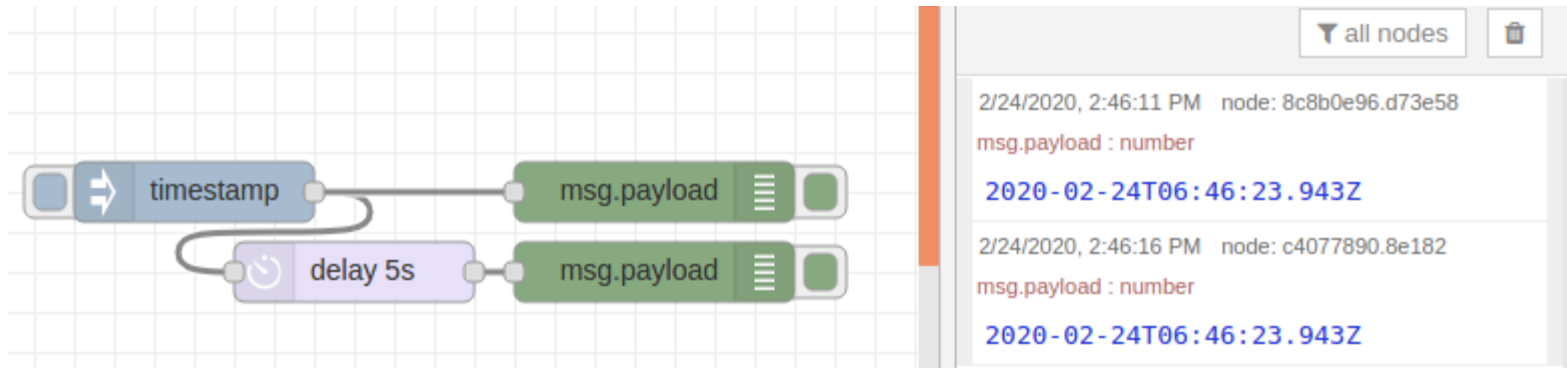
- Send:** the existing msg object
- then:** resend it every
- Interval:** 5 Seconds
- Reset the trigger if:**
  - msg.reset is set
  - msg.payload equals optional
- Handling:** all messages
- Name:** Name

On the right, the 'debug' console shows three log entries, each displaying the timestamp and the payload value '2019-09-26T07:01:34.101Z'.

- trigger
  - 可以收到 msg 後, 等待一段時間後再送給下個 node
  - 可以將 msg 每間隔一段時間重覆送給下個 node
- ref: example/trigger.json



# delay node

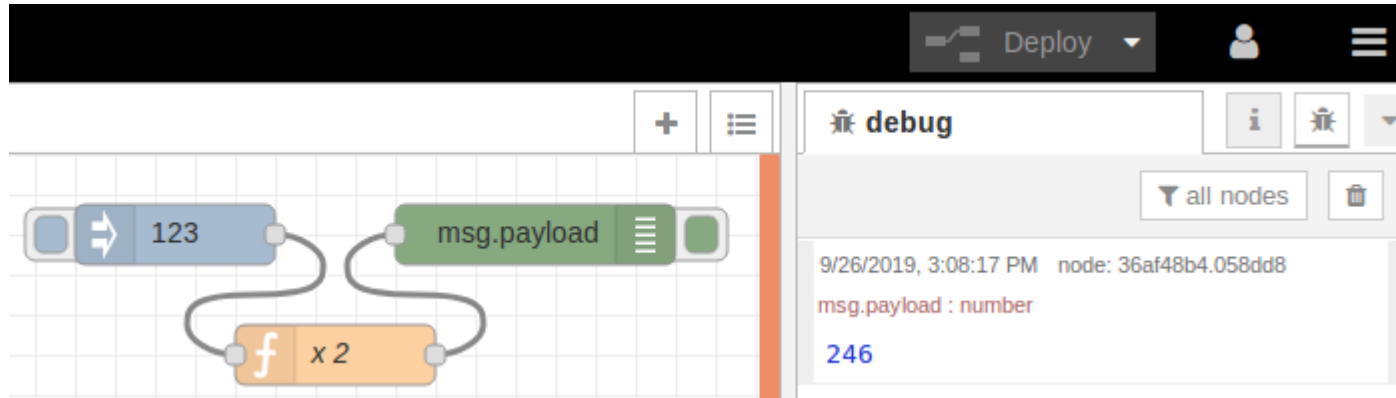


- delay: 將收到的 msg 留住, 等待一段時間後再送給下個 node
- ref: [example/delay.json](#)





# function node



**Edit function node**

Delete Cancel Done

**Properties**

Name

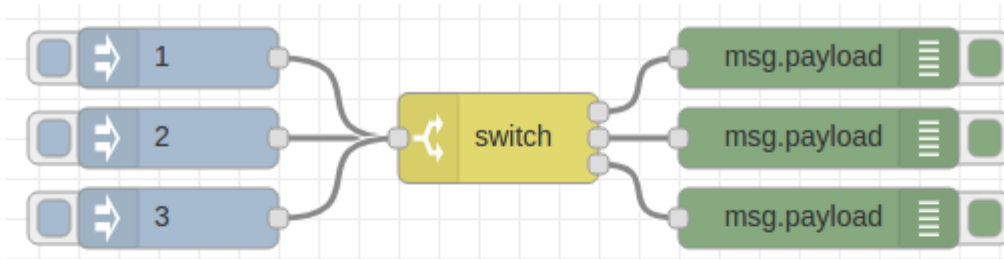
**Function**

```
1 const val = msg.payload
2
3 msg.payload = val * 2
4
5 return msg;
```

- function: 可寫自己的邏輯在裡面, 開發需要的功能
- ref: [example/function.json](#)



# switch node



- switch: 將 msg 分送到不同 flow
- ref: example/switch.json

**Edit switch node**

Delete Cancel Done

⚙️ **Properties** ⚙️ 📄 🔗

🔑 Name

⋮ Property ▼ msg. payload

≡	==	▼ 0 <sub>9</sub>	<input type="text" value="1"/>	→ 1	✕
≡	==	▼ 0 <sub>9</sub>	<input type="text" value="2"/>	→ 2	✕
≡	>	▼ a <sub>z</sub>	<input type="text" value="2"/>	→ 3	✕



# json node

The screenshot displays a Node-RED workflow and its debug output. The workflow consists of three nodes: an 'inject' node (blue), a 'json' node (yellow), and a 'msg.payload' node (green). The 'inject' node is configured with a 'Payload' of `{ "A": 123, "B": "Chameleon" }`. The 'json' node is used to convert the payload into a JSON object. The 'msg.payload' node outputs the resulting object.

The debug console shows the output of the workflow:

```
2/24/2020, 4:37:29 PM node: 1369875a.6ceac1
msg.payload : Object
▼ object
  A: 123
  B: "Chameleon"
```

- json: 做 json 轉換
- ref: example/json.json



# http request node

The screenshot displays the Node-RED interface with the 'http request' node highlighted. A red circle and arrow indicate the 'Edit http request node' dialog. The dialog shows the following configuration:

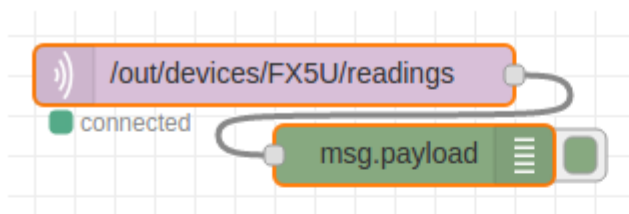
- Method:** GET
- URL:** http://localhost/api/v1/devices
- Append msg.payload as query string parameters:** ☐
- Enable secure (SSL/TLS) connection:** ☐
- Use authentication:** ☐
- Use proxy:** ☐
- Return:** a UTF-8 string
- Name:** Name

The 'debug' console shows the output of the 'http request' node, which is an empty array '[]'.

- http request: 可做GET,POST,PUT, DELETE的 http request
- ref: [example/http\\_request.json](#)



# mqtt node



**Edit mqtt in node**

Delete Cancel Done

**Properties**

Server: localhost:1883

Topic: /out/devices/FX5U/readings

QoS: 0

Output: a parsed JSON object

Name: Name

**debug**

all nodes

2/24/2020, 3:39:47 PM node: ea9b43e8.e069f8

/out/devices/FX5U/readings : msg.payload : Object

```
▼ object
▼ at: object
  sec: 2020-02-24T07:40:00.000Z
  us: 0
▼ channels: array[5]
  ▶ 0: object
  ▼ 1: object
    channelId: "D612"
    ▼ value: object
      num: 0
  ▶ 2: object
  ▼ 3: object
    channelId: "D660"
    ▼ value: object
      num: 0
  ▶ 4: object
    deviceId: "FX5U"
```

- mqtt in: 接收 mqtt 資料
- ref: [example/mqtt\\_in.json](#)



- 上機練習
- 休息 10 分鐘



# context

- 用來儲存資料的一種方法
- Context 分成
  - **Node**: 只有這個 node 可存取
  - **Flow**: 只有這個 flow (tab) 內的 node 可存取
  - **Global**: 所有 flow (tab)內的 node 都可以存取
- (預設) context 內容會在**系統重啟**情況下被清空, 若要避免被清空需要其他設定或是 plug-in
- 若 context get 不到, 值會是 **undefined**



# Node scope context

Edit function node

Delete Cancel Done

Properties

Name accumulate

Function

```
1 const val = msg.payload
2
3 count = context.get("count")
4 if (count === undefined)
5 {
6     count = 0
7 }
8
9 latest_count = count + val
10 context.set("count", latest_count)
11 msg.payload = latest_count
12
13 msg.payload = latest_count
14
15 return msg;
```

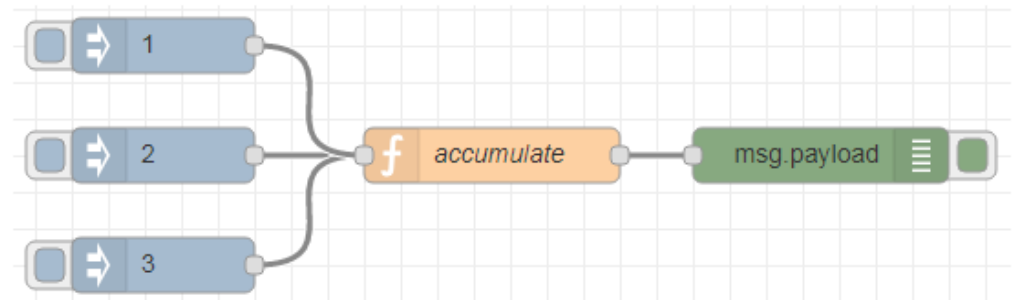
debug

all nodes

10/1/2019, 4:11:18 PM node: 6d3a0c5b.4ea4e4  
msg.payload : number  
3

10/1/2019, 4:11:19 PM node: 6d3a0c5b.4ea4e4  
msg.payload : number  
4

10/1/2019, 4:11:21 PM node: 6d3a0c5b.4ea4e4  
msg.payload : number  
6



- 使用 predefined variables **context**
- 重新 deploy 會清空 node context
- ref: example/node\_context.json





# Flow scope context

Edit function node

Delete Cancel Done

Properties

Name: set flow context

Function

```
1 const val = msg.payload
2
3 flow.set("inject_num", val)
4
5 msg.payload = "no_inject_num_info"
6
7 return msg;
```

Edit function node

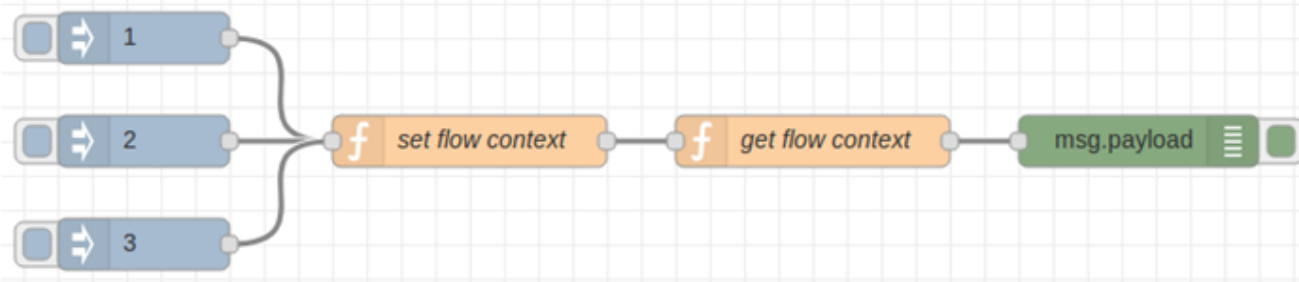
Delete Cancel Done

Properties

Name: get flow context

Function

```
1 const val = flow.get("inject_num")
2
3 msg.payload = val
4
5 return msg;
```



debug

all nodes

2020/2/24 下午8:48:54 node: 3ebf30fd.ad348  
msg.payload : number  
1

2020/2/24 下午8:48:55 node: 3ebf30fd.ad348  
msg.payload : number  
3

2020/2/24 下午8:48:56 node: 3ebf30fd.ad348  
msg.payload : number  
2

- 使用 predefined variables **flow**
- ref: [example/flow\\_context.json](#)



# Global scope context

- 使用 predefined variables **global**
- ref: example/global\_context.json

The screenshot shows the configuration for a 'set global context' node. The name field is set to 'set global context'. The function field contains the following JavaScript code:

```
1 const message = msg.payload
2
3 global.set("flow1_info", message)
4
5 return msg;
```

The screenshot shows the configuration for a 'get global context' node and its debug console output. The name field is set to 'get global context'. The function field contains the following JavaScript code:

```
1 const message = global.get("flow1_info")
2
3 if (message !== undefined)
4 {
5     msg.payload = message
6 }
7
8 return msg;
```

The debug console shows two messages:

- 2020/2/24 下午8:52:51 node: 8621e36b.1bd3c  
msg.payload : number  
2020-02-24T12:52:52.187Z
- 2020/2/24 下午8:53:00 node: 8621e36b.1bd3c  
msg.payload : string[11]  
"from flow 1"

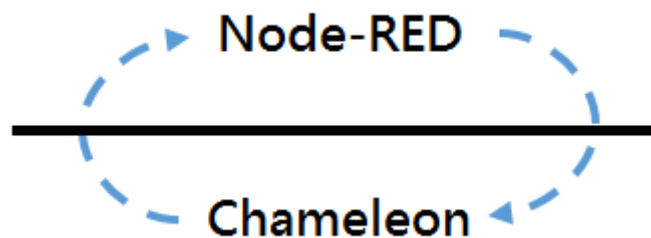


- 課堂練習
- QnA
- 中場休息



# Chameleon x Node-RED

- Node-RED 上可收到變色龍所有資料
- 在 Node-RED 處理完的資料可拋回變色龍
- 常用的功能已包成 node, 減少額外開發的時間





# 前置作業

- Chameleon 頁面新增 Mitsubishi FX5U 機台
  - ip: 貼在 plc 上
  - port: 9600
  - 3E:ASCII
- 點位表: (fx5u.xlsx)

點位	意義	長度	備註
X0	Switch alarm	1	Alarm 點
D602	溫度	1	乘 0.1
D612	濕度	1	乘 0.1
D650	Count	1	每秒加 1
D660	Double count	1	每0.1秒加 1



# 新增產線



 生產監控

 生產日誌

 檔案下載

生產監控

+ 新增產線



沒有任何產線資料

請

新增一條產線

以開始連接機台



# 新增機台



 生產監控

 生產日誌

 檔案下載

## 生產監控

+ 新增產線

產線 1   



沒有任何機台資料

請  以開始監控



# 新增機台-選擇PLC廠牌

☒ 開始新增設備

☒ 選擇廠牌

☒ 選擇類型

☒ 設定連線資料

☒ 測試設備連接

☒ 設定點位表

## 選擇設備類型

請選擇您要新增的設備類型

☒ **OMRON**

☒ **MITSUBISHI  
ELECTRIC**

☒ **MODBUS**

☒ **SIEMENS**

☒ **PANASONIC**

☒ **VIRTUAL**

上一步

下一步





# 新增機台-設定連線資料

開始新增設備

選擇廠牌

選擇類型

設定連線資料

測試設備連接

設定點位表

## 設定 MELSEC 連線資料

請輸入您要連線的設備資料



FX5U

Protocol

3E:ascii

設備 ID

FX5U

設備名稱

連線類型

TCP Host

192.168.20.78

開始新增設備

選擇廠牌

選擇類型

設定連線資料

測試設備連接

設定點位表

## 設定 MELSEC 連線資料

請輸入您要連線的設備資料



FX3U

連線類型 ☒ TCP ☐ UDP

TCP Host

192.168.20.78 填 PLC ip

TCP Port

9600 PLC tcp port

連線間隔毫秒 (pollingPeriodMS)

1000

連線時間 (requestTimeoutMS)

500

上一步

下一步



# 新增機台-連線測試

✓ 開始新增設備

✓ 選擇廠牌

✓ 選擇類型

✓ 設定連線資料

✓ 測試設備連接

✓ 設定點位表

## 測試 MELSEC 設備連接

您可以「略過」此步驟直接設定點位表，或輸入以下資料來進行「測試」



FX5U

Command ☒ Read Word ☐ Read Bit

Register Type

D

Address

602

Quantity

1

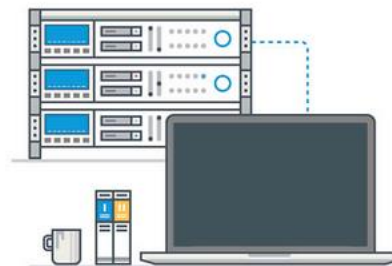
上一步

測試

略過

## 測試 MELSEC 設備連接

您可以「略過」此步驟直接設定點位表，或輸入以下資料來進行「測試」



測試成功，設備和主機之間的連線正常

上一步

重新測試

下一步，設定點位表



# 新增機台-上傳點位表

✓ 開始新增設備

✓ 選擇廠牌

✓ 選擇類型

✓ 設定連線資料

✓ 測試設備連接

✓ 設定點位表

## 設定點位表

請選擇您要建立點位表的方式



FX5U

上傳點位表

✓ 開始新增設備

✓ 選擇廠牌

✓ 選擇類型

✓ 設定連線資料

✓ 測試設備連接

✓ 設定點位表

## 設定點位表

請選擇您要建立點位表的方式



FX5U

上傳點位表

點位表 FX5U 已上傳 ✓

上一步

完成



# 新增機台-完成



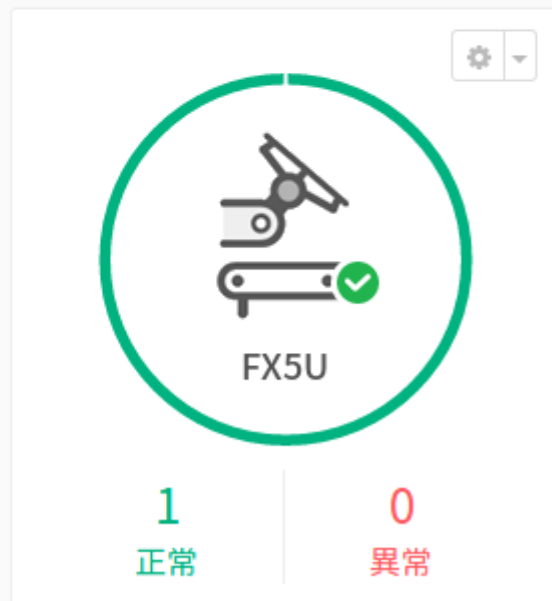
生產監控

生產日誌

檔案下載

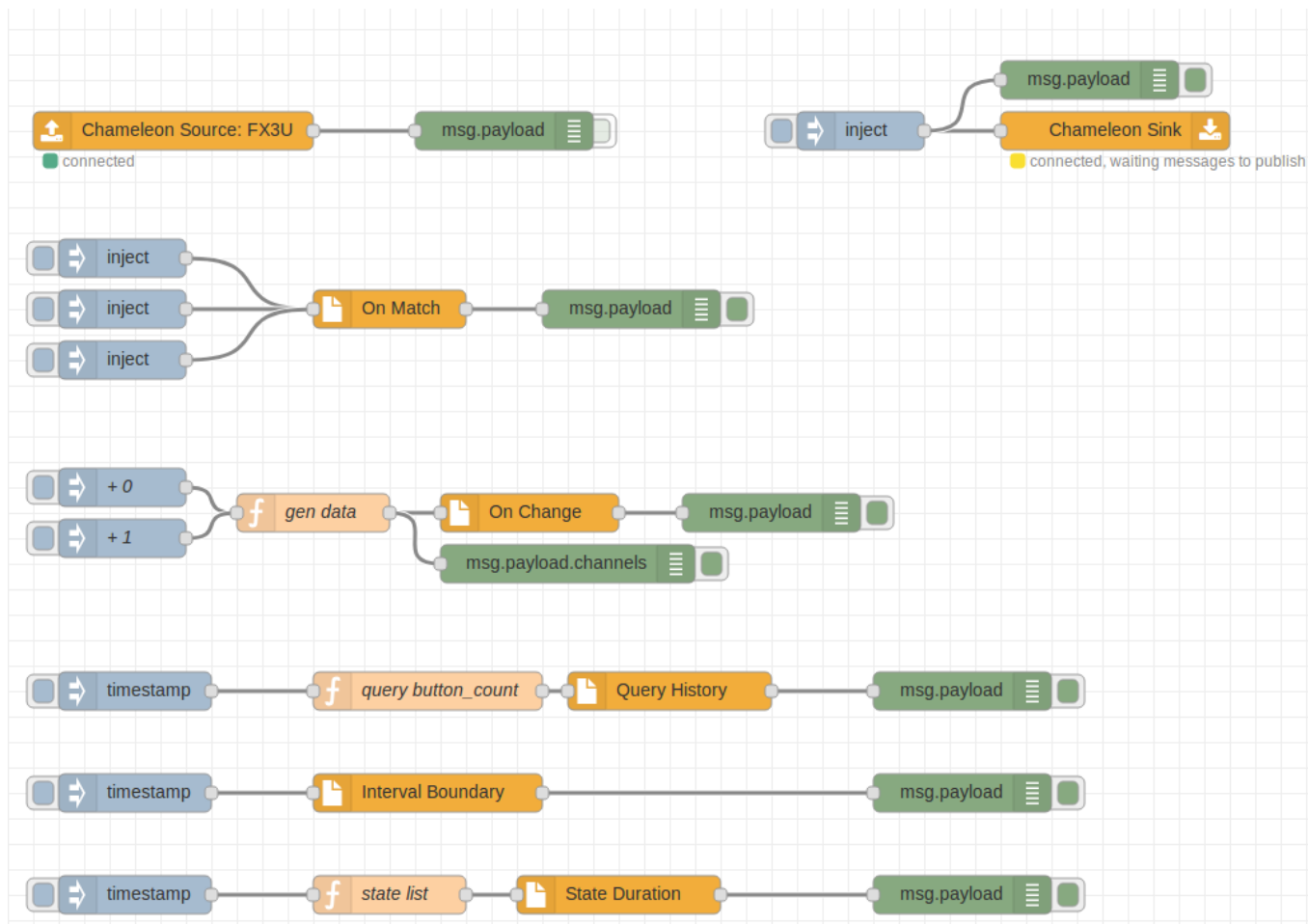
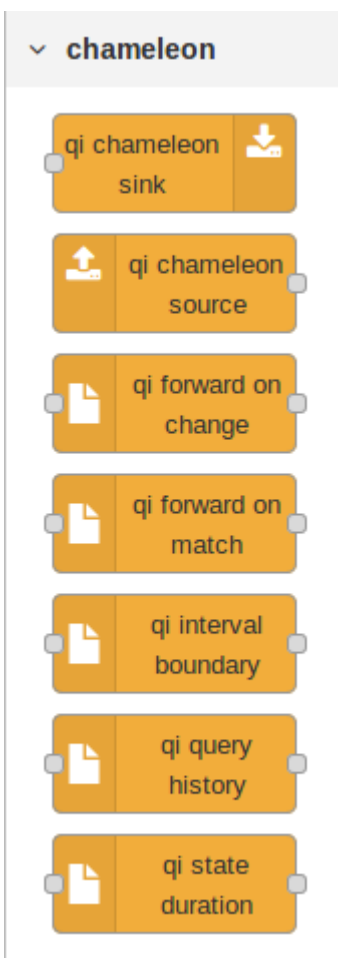
## 生產監控

### 產線 1





# Node with Chameleon





# qi chameleon source

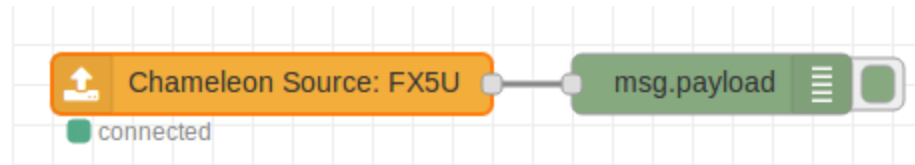
- 從 chameleon 接收資料
- 可在 node 編輯頁面設定要接收的 device id 或 “+” 接收所有 device 的資料
- 資料輸出格式

```
{  
  "deviceId": "string",  
  "at": {  
    "sec": integer,  
    "us": integer  
  },  
  "channels": {  
    "channel": value  
  }  
}
```

```
{  
  "deviceId": "FX5U",  
  "at": {  
    "sec": 1569911173,  
    "us": 0  
  },  
  "channels": {  
    "D602": 25.7,  
    "D612": 57.5,  
    "D650": 101,  
    "D660": 1006,  
    "X0": 0  
  }  
}
```



# qi chameleon source



**Edit qi-chameleon-source node**

Delete Cancel Done

**Properties**

Device ID  設定 device id 來接收 FX5U 的資料

Name

ref: example/chameleon-source.json

**debug**

all nodes

10/2/2019, 2:02:49 PM node: c2495404.c0ba5

msg.payload : Object

- object
  - deviceId: "FX5U"
- at: object
  - sec: 1569996180
  - us: 0
- channels: object
  - D602: 26.2
  - D612: 55.3
  - D650: 20620
  - D660: 9588
  - X0: 0



# qi chameleon sink

- 將資料拋回 chameleon
- **Chameleon** 需先增加虛擬設備
- 資料輸入格式

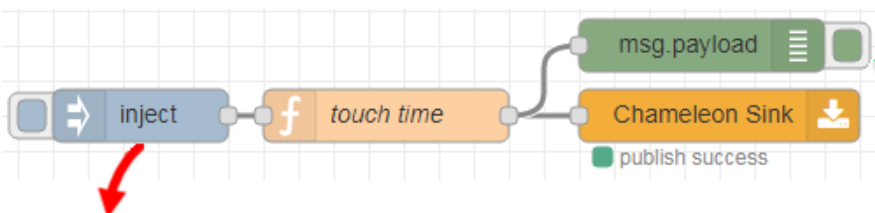
```
{  
  "deviceId": "string",  
  "at": {  
    "sec": integer,  
    "us": integer  
  },  
  "channels": {  
    "channel": value  
  }  
}
```

```
{  
  "deviceId": "test_chameleon_sink",  
  "at": {  
    "sec": 1564042800,  
    "us": 0  
  },  
  "channels": {  
    "float_channel": 1.23,  
    "integer_channel": 10,  
    "string_channel": "chameleon_no1"  
  }  
}
```





# qi chameleon sink



- ref: example/chameleon-sink.json
- test\_chameleon\_sink\_virtual.xlsx

Edit inject node > JSON editor

format JSON

```
1 {
2   "deviceId": "test_chameleon_sink",
3   "at": {
4     "sec": 1570074060,
5     "us": 1000
6   },
7   "channels": {
8     "float_channel": 94.87,
9     "integer_channel": 200,
10    "string_channel": "chameleon_no1"
11  }
12 }
```

debug

all nodes

```
2020/2/24 下午9:03:19 node: 663bf6bc.f9a9d8
msg.payload : Object
  object
    deviceId: "test_chameleon_sink"
    at: object
      sec: 1582549400
      us: 1000
    channels: object
      float_channel: 94.87
      integer_channel: 200
      string_channel: "chameleon_no1"
```

test\_chameleon\_sink

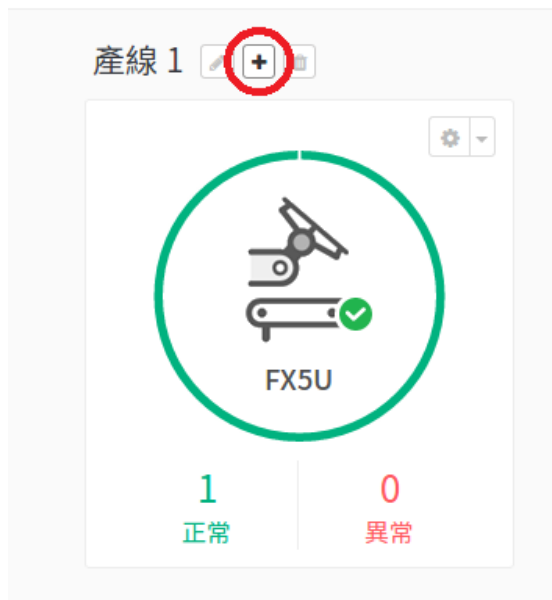
告警狀態 即時狀態 設備管理 點位表管理

#	點位 ID	點位名稱	數值類型	數值內容
👁	float_channel		number	94.87
👁	integer_channel		number	200
👁	string_channel		string	chameleon_no1



# 新增虛擬設備-選擇 VIRTUAL

生產監控



增設備

牌

型

線資料

位表

## 選擇設備類型

請選擇您要新增的設備類型



上一步

下一步



# 新增虛擬設備-設定設備 ID

✓ 開始新增設備

✓ 選擇廠牌

✓ 選擇類型

✓ 設定連線資料

✓ 設定點位表

## 設定 VIRTUAL 連線資料

請輸入您要連線的設備資料



test-chameleon-sink

設備 ID

設備名稱

上一步

下一步



# 新增虛擬設備-上傳點位表

☑ 開始新增設備

☑ 選擇廠牌

☑ 選擇類型

☑ 設定連線資料

☑ 設定點位表

## 設定點位表

請選擇您要建立點位表的方式



test-chamel  
sink

上傳點位表

☑ 開始新增設備

☑ 選擇廠牌

☑ 選擇類型

☑ 設定連線資料

☑ 設定點位表

## 設定點位表

請選擇您要建立點位表的方式



test\_chameleon\_!

上傳點位表

點位表 test\_chameleon\_sink 已上傳 ✓

上一步

完成



# 新增虛擬設備-完成



生產監控

生產日誌

檔案下載

## 生產監控

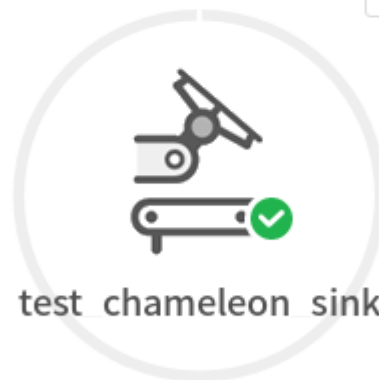
### 產線 1



FX5U

1  
正常

0  
異常



test chameleon sink

-  
正常

-  
異常



# qi on match

- 資料 filter
- 需在 node 編輯頁面設定 Schema
- Schema 格式 (example: qi-on-match.schema)
  - \$schema
  - type
  - required
  - properties



# qi on match

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "required": [
    "deviceId",
    "at",
    "channels"
  ],
  "properties": { ...
  }
}
```



# qi on match

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "required": [ ],
  "properties": {
    "deviceId": {
      "type": "string"
    },
    "at": {
      "type": "object",
      "required": [
        "sec",
        "us"
      ],
      "properties": {
        "sec": {
          "type": "integer",
          "minimum": 0
        },
        "us": {
          "type": "integer",
          "minimum": 0,
          "maximum": 999999
        }
      }
    },
    "channels": [ ]
  }
}
```



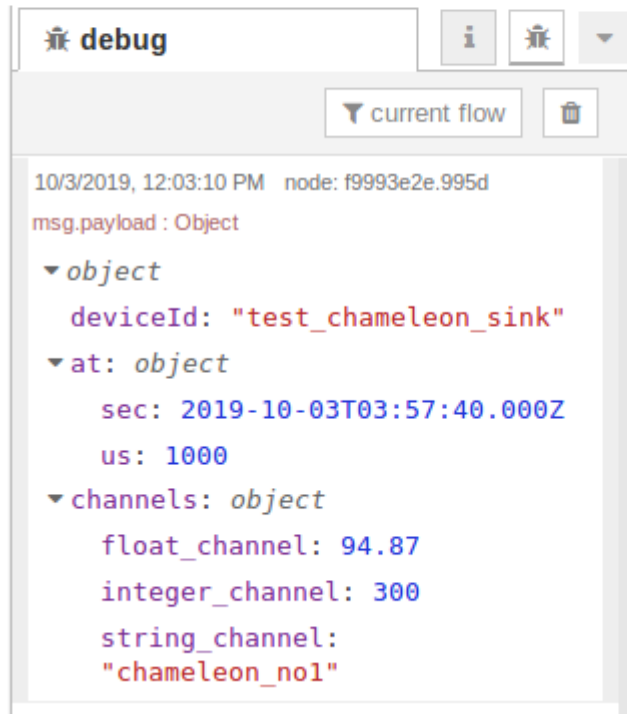
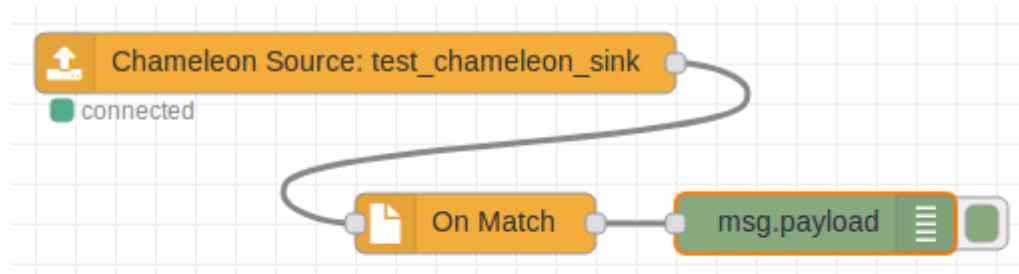


# qi on match

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "required": [
  ],
  "properties": {
    "deviceId": {
      "type": "string"
    },
    "at": {
    },
    "channels": {
      "type": "object",
      "required": [
        "float_channel",
        "integer_channel",
        "string_channel"
      ],
      "properties": {
        "float_channel": {
          "type": "number"
        },
        "integer_channel": {
          "type": "number"
        },
        "string_channel": {
          "type": "string"
        }
      }
    }
  }
}
```



# qi on match



- ref: example/on-match.json
- qi-on-match.schema



# qi on match

Edit qi-forward-on-match node

Delete Cancel Done

Properties

Schema  {"\$schema": "http://json-schema.org/draft-07/schema#"} ...

☒ Remove additional

Name

經過 on-match node 後  
不在 properties::channels::required 的  
channel 會被過濾掉

Edit qi-forward-on-match node > JSON editor

Cancel Done

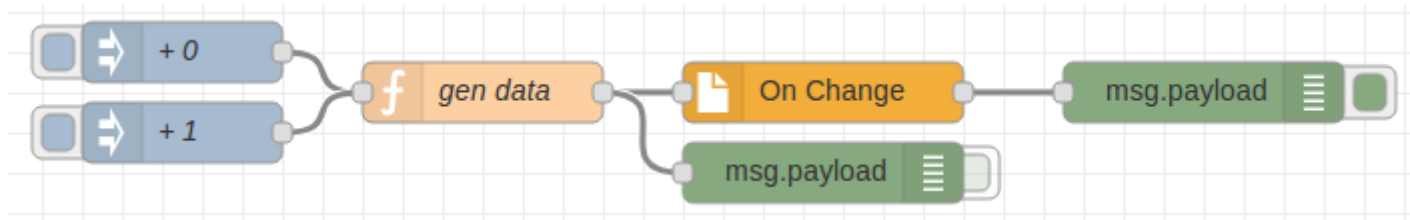
format JSON

```
1 {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "type": "object",
4   "required": [
5     "deviceId",
6     "at",
7     "channels"
8   ],
9   "properties": {
10    "deviceId": {
11      "type": "string"
12    },
13    "at": {
14      "type": "object",
15      "required": [
16        "sec",
17        "us"
18      ],
19      "properties": {
20        "sec": {
21          "type": "integer",
22          "minimum": 0
23        },
24        "us": {
25          "type": "integer",
26          "minimum": 0,
27          "maximum": 999999
28        }
29      }
30    },
31    "channels": {
32      "type": "object",
33      "required": [
34        "float channel",
35        "integer channel",
36        "string channel"
37      ],
38      "properties": {
39        "float channel": {
40          "type": "number"
41        },
```



# qi on change

- Channel 的 value 值改變才會往下送



ref: example/on-change.json

debug

all nodes

10/3/2019, 2:04:53 PM node: 1726e160.7a587f  
msg.payload : Object

- object
  - deviceId: "dev"
  - at: object
  - channels: object
    - accumulate\_value: 0

10/3/2019, 2:04:55 PM node: 1726e160.7a587f  
msg.payload : Object

- object
  - deviceId: "dev"
  - at: object
  - channels: object
    - accumulate\_value: 1



# qi query history

- 從變色龍的 database query 資料
- Query string (以 msg.query 為主)
  1. 編輯 node 內的 **Query** 欄位
  2. 或是在之前的 node 準備好 **msg.query**
- Query string 格式: **InfluxQL**
  - the InfluxDB SQL-like query language
  - [https://docs.influxdata.com/influxdb/v1.7/query\\_language/](https://docs.influxdata.com/influxdb/v1.7/query_language/)



# qi query history

The screenshot displays the qi query history interface. At the top, a workflow bar shows four nodes: 'timestamp', 'build query string', 'Query History', and 'msg.payload'. The 'build query string' node is selected, opening the 'Edit function node' dialog.

**Edit function node**

Buttons: Delete, Cancel, Done

**Properties**

Name: build query string

**Function**

```
1 msg.query = `  
2   SELECT *  
3   FROM FX5U  
4   LIMIT 2  
5 `  
6  
7 return msg;
```

**debug**

10/3/2019, 2:18:15 PM node: 80b6e92c.449b68  
msg.payload : array[2]  
▼ array[2]  
▼ 0: object  
time:  
"2019-10-03T02:31:00.003Z"  
D602: 24.4  
D612: 59.9  
D650: 3653  
D660: 36526  
X0: 0  
► 1: object

- ref: [example/query-history.json](#)



# qi interval boundary

- 取得某段的開始及結束時間
- node 編輯頁面設定
  - Timezone: UTC/Local Time
  - Interval Duration(秒): default is 86400 (1 天)
  - Shift interval: 輸入整數, 以當下時間為基礎做 interval 的 shift, 若 interval duration 是 86400, 輸入 “-2” 會取得前天的開始/結束時間



# qi interval boundary

The image shows a workflow diagram at the top with three nodes: 'timestamp', 'Interval Boundary', and 'msg.payload'. The 'timestamp' node is connected to the 'Interval Boundary' node, which is then connected to the 'msg.payload' node. Below the diagram is the 'Edit qi-interval-boundary node' dialog box. It has a 'Delete' button, 'Cancel' button, and a red 'Done' button. The 'Properties' section includes: 'Timezone' set to 'Local Time', 'Interval Duration' set to '86400', 'Shift Interval' set to '0', and 'Name' set to 'Name'. To the right is the 'debug' console showing two log entries. The first entry is for node '7fa84e0c.d33e2' with 'msg.payload : number' and the value '2020-02-24T10:02:41.268Z'. The second entry is for node 'd67218ea.5f1d08' with 'msg.payload : Object' and the value '{ begin: "2020-02-23T16:00:00.000Z", end: "2020-02-24T16:00:00.000Z" }'.

**Edit qi-interval-boundary node**

Delete Cancel Done

**Properties**

Timezone Local Time

Interval Duration 86400

Shift Interval 0

Name Name

**debug**

2/24/2020, 6:02:40 PM node: 7fa84e0c.d33e2  
msg.payload : number  
2020-02-24T10:02:41.268Z

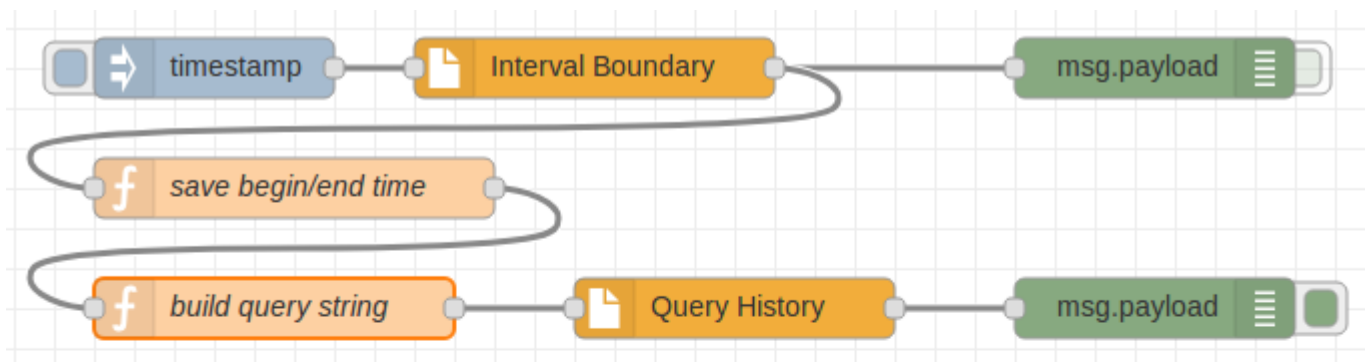
2/24/2020, 6:02:40 PM node: d67218ea.5f1d08  
msg.payload : Object  
{ begin: "2020-02-23T16:00:00.000Z", end: "2020-02-24T16:00:00.000Z" }

- ref: [example/interval-boundary.json](#)





# interval boundary + query history example



## result of Interval Boundary

10/4/2019, 4:29:07 PM node: 1798a542.6dab9b

msg.payload : Object

```
▶ { begin:
  "2019-10-03T16:00:00.000Z", end:
  "2019-10-04T16:00:00.000Z" }
```

## debug

all nodes

10/4/2019, 4:26:35 PM node: 9fc5215a.da1d6

msg.payload : array[5]

▼ array[5]

▶ 0: object

▶ 1: object

▼ 2: object

time:

"2019-10-04T07:45:00.000Z"

D612: 62

▶ 3: object

▶ 4: object

- ref: [example/example\\_query-history.json](#)



# interval boundary + query history example

**Edit function node**

Delete

Cancel

Done

⚙ Properties

⚙

📄

🔗

📁 Name

save begin/end time

📄

🔧 Function

🔗

i 1

const begin = msg.payload.begin

i 2

const end = msg.payload.end

3

i 4

msg.begin\_time = begin

i 5

msg.end\_time = end

6

7

return msg;

**Edit function node**

Delete

Cancel

Done

⚙ Properties

⚙

📄

🔗

📁 Name

build query string

📄

🔧 Function

🔗

1

const begin\_time = msg.begin\_time

2

const end\_time = msg.end\_time

3

4

msg.query = `

5

SELECT D612

6

FROM FX5U

7

WHERE time >= '\${begin\_time.toISOString()}'

8

AND time < '\${end\_time.toISOString()}'

9

LIMIT 5

10

`

11

12

return msg;



# qi state duration

- 取得每個 state 的累積時間(單位: 毫秒 **millisecond**)
  - 設定要算的 state name
  - 資料輸入格式:
    - msg.payload: influxDB query result
    - msg.since: (optional) Date
    - msg.until: (optional) Date
- ```
[  
  {  
    "time": <date>,  
    "<channel>": <value>  
  },  
]
```

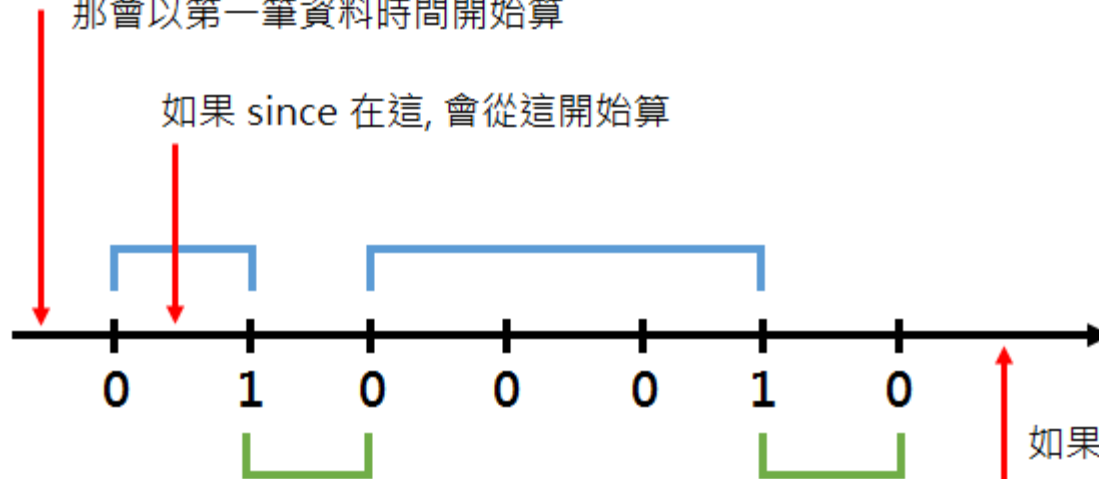


# qi state duration

- msg.payload: 允許重覆連續的 state
- msg.since: 若沒給, 會以 payload 第一筆資料的時間開始算
- msg.until: 若沒給, 會以 payload 最後一筆資料的時間為止

如果 since 小於第一筆資料時間, 或是沒給 since  
那會以第一筆資料時間開始算

如果 since 在這, 會從這開始算



until 時間不能小於最後一筆資料的時間



# qi state duration

Diagram showing a sequence of nodes: timestamp, state list, State Duration, and msg.payload.

**Edit qi-state-duration node**

Delete Cancel Done

**Properties**

State Name: operation

Name: Name

**debug**

10/3/2019, 3:34:35 PM node: 39834e25.7bfdba  
msg.payload : array[2]

```
▼ array[2]
  ▼ 0: object
    state: 0
    duration: 255000
  ▼ 1: object
    state: 1
    duration: 44000
```

**Function**

```
1 msg.payload = [
2   {
3     "time":new Date("2019-10-02T02:50:35.000Z"),
4     "operation":0
5   }, // 0:46 -> since: 2019-10-02T02:50:54.000Z -> 0:27
6   {
7     "time":new Date("2019-10-02T02:51:21.000Z"),
8     "operation":1
9   }, // 0:26
10  {
11    "time":new Date("2019-10-02T02:51:47.000Z"),
12    "operation":0
13  }, // 0:42
14  {
15    "time":new Date("2019-10-02T02:52:29.000Z"),
16    "operation":1
17  }, // 0:9
18  {
19    "time":new Date("2019-10-02T02:52:38.000Z"),
20    "operation":0
21  }, // 0:6
22  {
23    "time":new Date("2019-10-02T02:52:44.000Z"),
24    "operation":1
25  }, // 0:9
26  {
27    "time":new Date("2019-10-02T02:52:53.000Z"),
28    "operation":0
29  } // 3:0
30 ]
31
32 msg.since = new Date("2019-10-02T02:50:54.000Z")
33 msg.until = new Date("2019-10-02T02:55:53.000Z")
34
35 return msg;
```

- ref: [example/state-duration.json](#)

