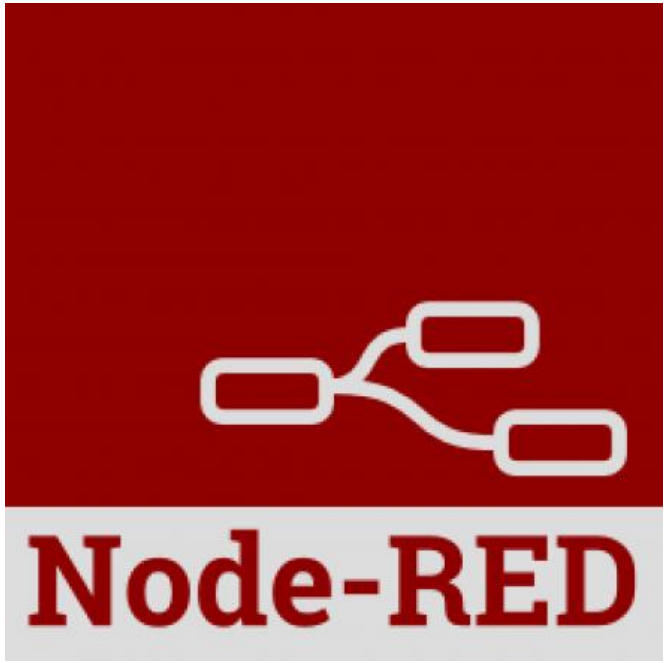
# Introduction to Node-RED

# Contents

# 1. Introduction to Node-RED




# What is Node-RED?

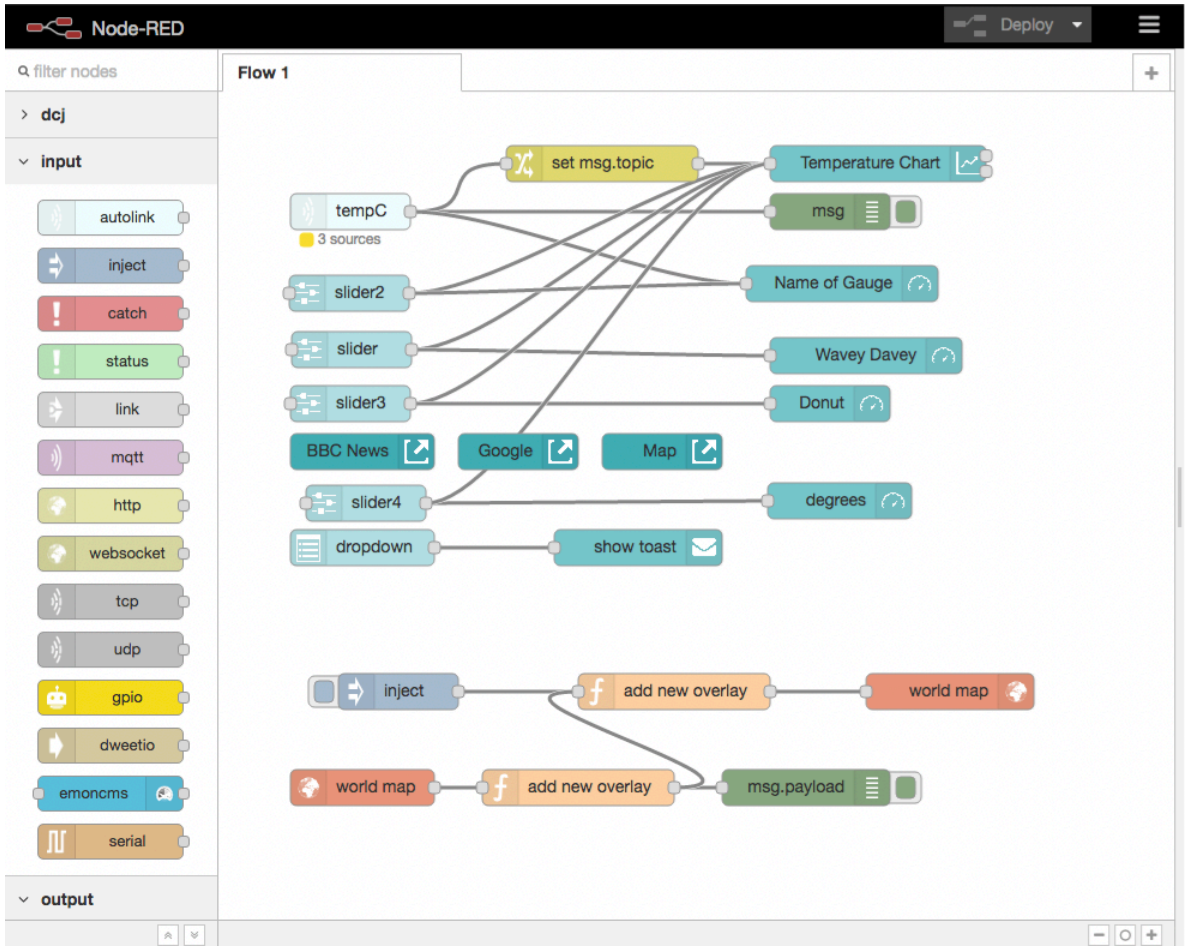
- Open-source Node.js application
- Originally developed as an open source project at IBM in 2013
- Provides a visual programming editor that makes it easy to wire together flows
- Has wide range of nodes in the palette that can be deployed to its runtime in a single-click
- Additional information: https://nodered.org/

# What’s the Problem?

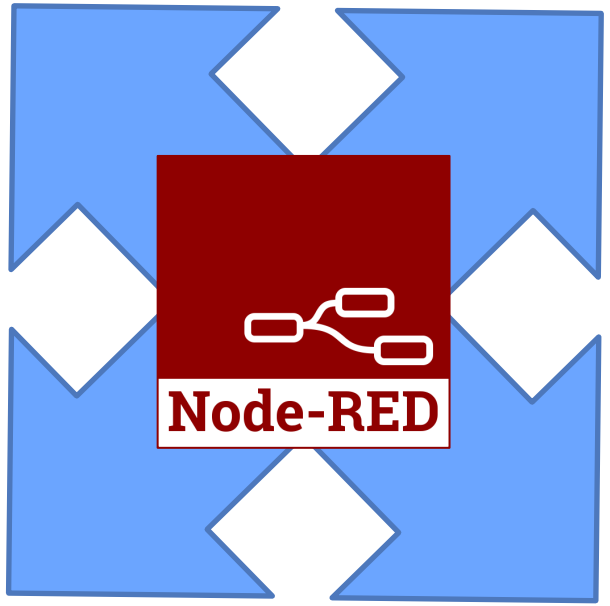
- The Internet of Things does not have a one-size-fits-all solution.
- Internet of Things solutions often require pulling together different device APIs and online services in new and interesting ways.
- Time spent figuring out how to access a Serial port, or to complete an OAuth flow is not time spent on creating the real value of a solution, and is not easy for anyone but dedicated programmers.
- We need tools that make it easier for developers at all levels to bring together the different streams of events, both physical and digital, that make up the Internet of Things.
- Standards are great – but rarely only need just one…

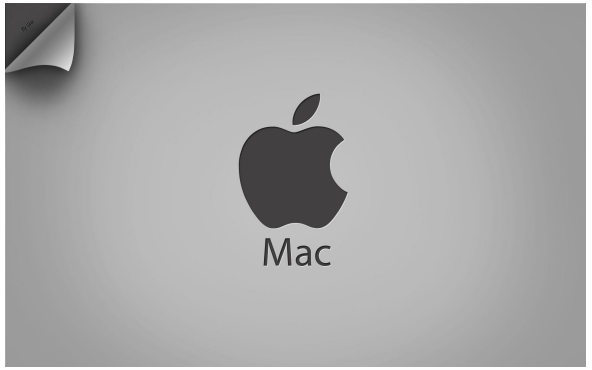
# Introducing Node-RED

- Node-RED makes it easy to wire together the Internet of Things.
- It provides a browser-based drag-drop UI for creating flows of events and deploying them to the runtime.
- The light-weight runtime, built in node.js, is ideal for edge-of-network environments or running in the cloud.
- It can be easily expanded to take add new nodes to the palette – taking full advantage of the node package manager (npm) ecosystem



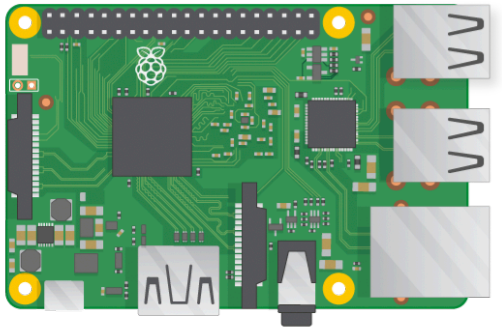
# Node-RED





# From the edge to the cloud



**Raspberry Pi**

Pre-installed on the default Raspberry Pi image, Node-RED can be used out of the box to begin creating IoT applications.

**IBM Cloud**

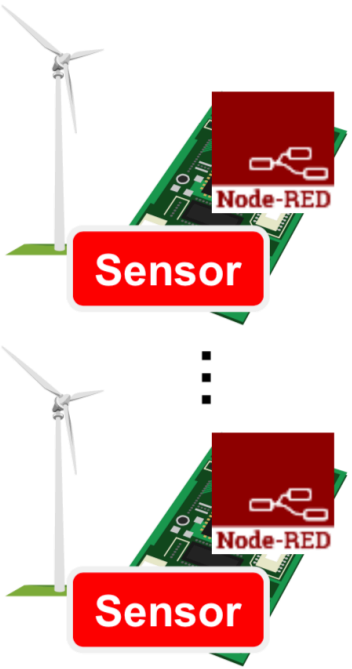
Available in the IBM Cloud catalogue as a Quick Start application, it takes moments to create cloud applications that combine services from across the platform.

# What does Node-RED realize?

Node-RED has key features to realize IoT system.

## Devices

(1) Edge analytics




Node-RED on edge devices

(2) Data Collection

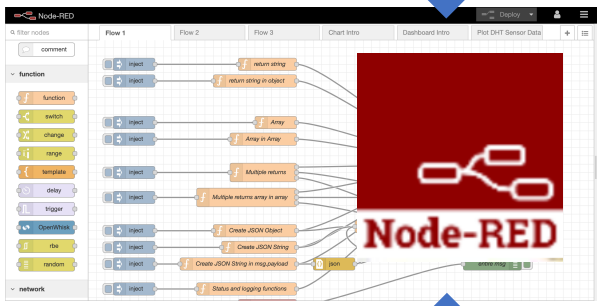
MQTT

MQTT

(3) Device Control

## Cloud


(4) Dashboard




(5) Data accumulation /retrieving

Data Lake

(6) Integration with existing systems

REST API

REST API

(7) Connection to external services

## IT Systems

Business data

External services

# Who uses Node-RED?

Major companies have used Node-RED in their productions.

**Cloud services**

- IBM, IBM Cloud
- AT&T, AT&T IoT Platform
- Fujitsu, K5 COLMINA Platform - Hitachi, Lumada

**Connectors (Node-RED nodes)**

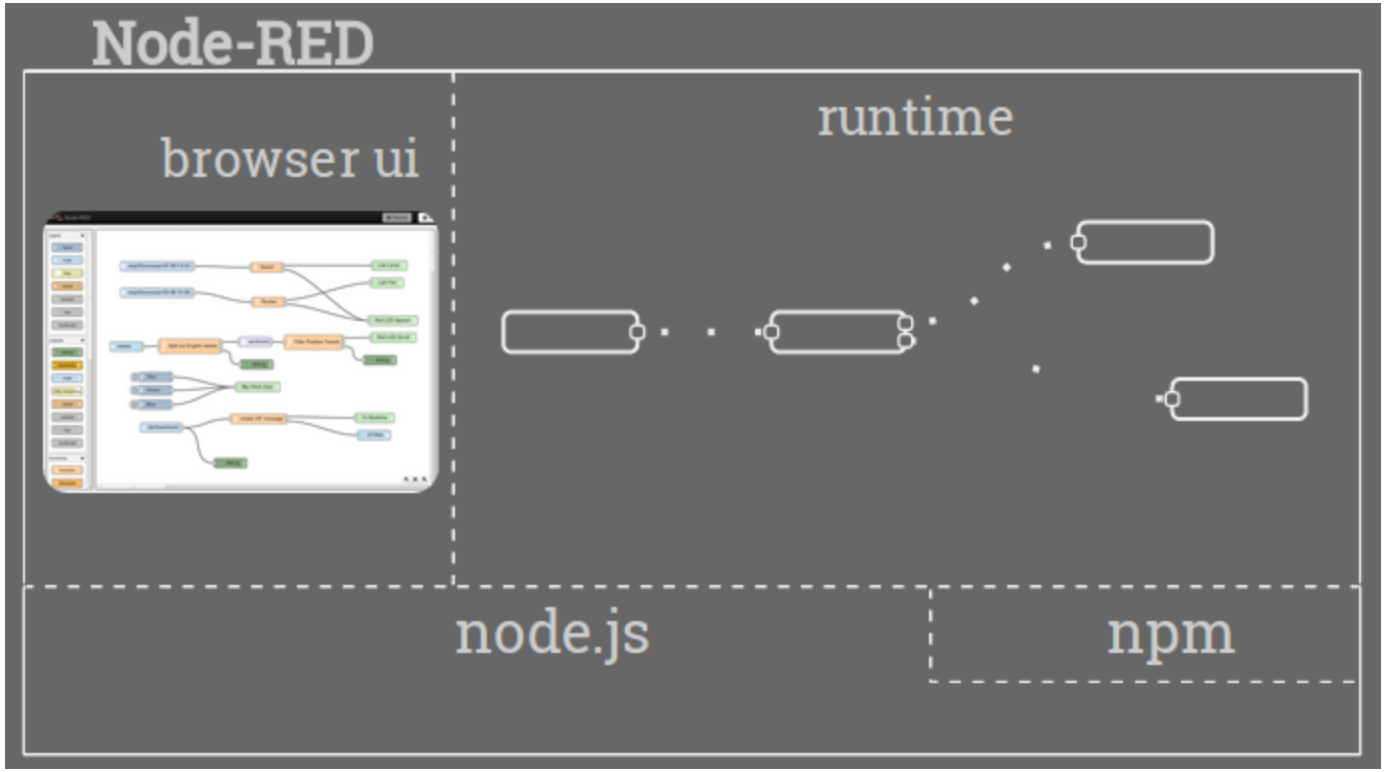
- IBM, Watson IoT Platform
- Microsoft, Azure IoT Hub
- NEC, Mobile Backend Platform

**Edge devices**

- GE, Predix Developer Kit
- Intel, Intel IoT Gateway
- NEC, CONNEXIVE IoT Connectivity Engine
- Samsung, Artik
- Siemens, SIMATIC IOT2020
- Toshiba, SPINEX

# Architecture of Node-RED

- Node.js v8-engine driven;
- Event-driven, asynchronous io;
- Single-threaded event-queue;
- Javascript front and back;
- Built using express, d3, jquery and ws



# Flow-based Programming

- Invented by J. Paul Morrison at IBM in the early 1970's
- A network of asynchronous processes communicating by means of streams of structured data chunks
- Each process is a black box – it doesn't know what has come before it, or what comes after it; it just acts on the data it receives and passes the result on

# Node-RED

- A rich library of **nodes** and **flows**

- **Nodes**: predefined code blocks
  - Input nodes
  - Processing nodes
  - Output nodes
- **Flows**: A set of connected nodes to perform a task

# Node-RED and JSON

- Node-RED flow works by passing JSON messages between nodes
- JSON messages usually have a payload property which is the default property that most nodes will work with

- Flows created are also stored using JSON which can be easily imported and exported for sharing

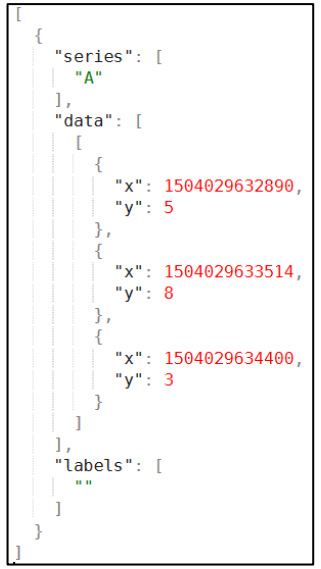
# JSON

- JavaScript Object Notation
- Standard way for representing a JavaScript object as a String and it is commonly used by web APIs to return data

**Example:**

- `{"_msgid":"c04270ce.9089b","topic":"","payload":"Hello World"}`


- `[{"series":["A"],"data":[[{"x":1504029632890,"y":5},{"x":1504029633514,"y":8},{"x":1504029634400,"y":3}]],"labels":[""]}]`



# JSON

- An *object* is an unordered set of name/value pairs
  - The pairs are enclosed within braces, { }
  - There is a colon between the name and the value
  - Pairs are separated by commas
  - Example: { "name": "html", "years": 5 }
- An *array* is an ordered collection of values
  - The values are enclosed within brackets, [ ]
  - Values are separated by commas
  - Example: [ "html", ”xml", "css" ]
- An *object array* is an array of objects
  - The values are enclosed within brackets, [ ]
  - Example: [ { "name": "html", "years": 5 }, { "name": “xml", "years": 5 } ]


```
[
  {
    "series": [
      "A"
    ],
    "data": [
      [
        {
          "x": 1504029632890,
          "y": 5
        },
        {
          "x": 1504029633514,
          "y": 8
        },
        {
          "x": 1504029634400,
          "y": 3
        }
      ]
    ],
    "labels": [
      ""
    ]
  }
]
```

# JavaScript Basics - Syntax

```
// Two slashes start single-line comments

var x;  // declaring a variable

x = 3 + y;  // assigning a value to the variable `x`

foo(x, y);  // calling function `foo` with parameters `x` and `y`
obj.bar(3);  // calling method `bar` of object `obj`

// A conditional statement
if (x === 0) {  // Is `x` equal to zero?
    x = 123;
}

// Defining function `baz` with parameters `a` and `b`
function baz(a, b) {
    return a + b;
}
```

# JavaScript Basics – Data Types

| Variable | Explanation | Example |
|---|---|---|
| **String** | A sequence of text known as a string. To signify that the value is a string, you must enclose it in quote marks. | `let myVariable = 'Bob';` |
| **Number** | A number. Numbers don't have quotes around them. | `let myVariable = 10;` |
| **Boolean** | A True/False value. The words `true` and `false` are special keywords in JS, and don't need quotes. | `let myVariable = true;` |
| **Array** | A structure that allows you to store multiple values in one single reference. | `let myVariable = [1,'Bob','Steve',10];` Refer to each member of the array like this: `myVariable[0]`, `myVariable[1]`, etc. |
| **Object** | Basically, anything. Everything in JavaScript is an object, and can be stored in a variable. Keep this in mind as you learn. | `let myVariable = document.querySelector('h1');` All of the above examples too. |

# JavaScript Basics – Statements

## Conditionals

- If..else..

```
if (myvar === 0) {
    // then
}

if (myvar === 0) {
    // then
} else {
    // else
}

if (myvar === 0) {
    // then
} else if (myvar === 1) {
    // else-if
} else if (myvar === 2) {
    // else-if
} else {
    // else
}
```

- switch..case

```
switch (fruit) {
    case 'banana':
        // ...
        break;
    case 'apple':
        // ...
        break;
    default:  // all other cases
        // ...
}
```

## Loops

- for

```
for ([«init»]; [«condition»]; [«post_iteration»])
    «statement»

for (var i=0; i < arr.length; i++) {
    console.log(arr[i]);
}
```

- while

```
// Same as for loop above:
var i = 0;
while (i < arr.length) {
    console.log(arr[i]);
    i++;
}
```

In all loops:

- ***break*** leaves the loop.
- ***continue*** starts a new loop iteration.

# JavaScript Basics – Function

A JavaScript function is a block of code designed to perform a particular task.
A JavaScript function is executed when "something" invokes it (calls it).

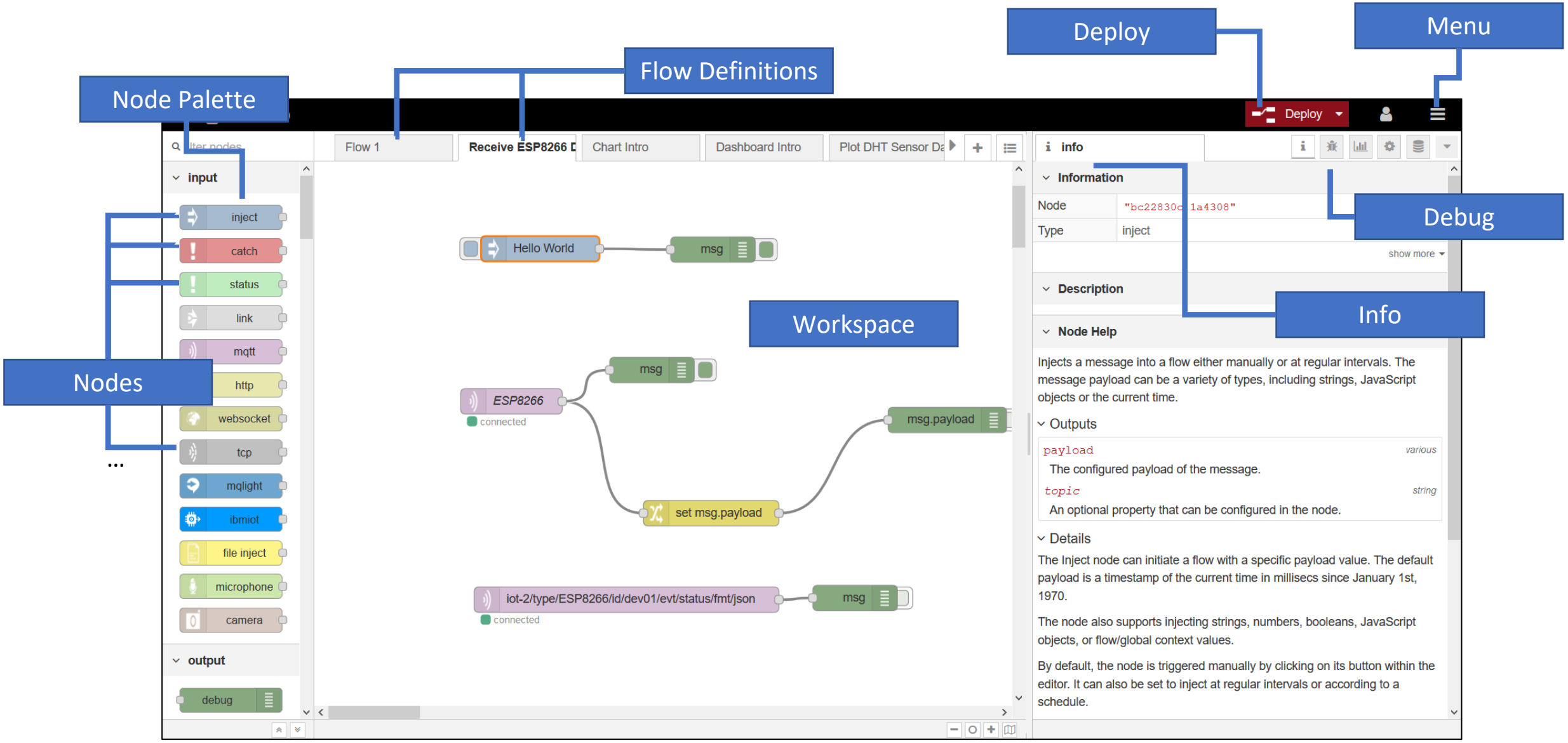
```
function add(param1, param2) {
    return param1 + param2;
}
```

```
> add(6, 1)
7
> add('a', 'b')
'ab'
```

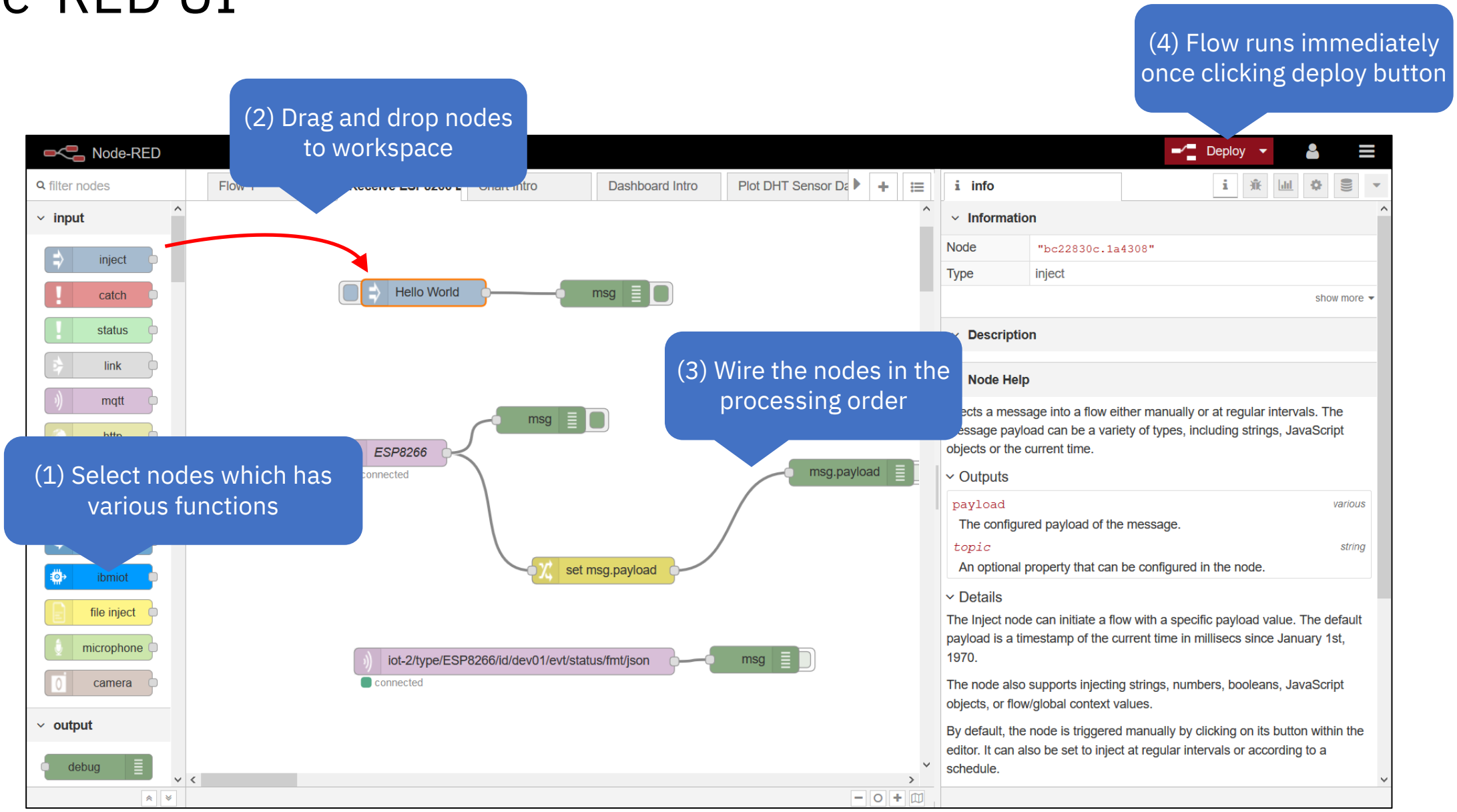
- Function Invocation
  The code inside the function will execute when "something" invokes (calls) the function:
  - When an event occurs (when a user clicks a button)
  - When it is invoked (called) from JavaScript code
  - Automatically (self invoked)

- Function Return
  When JavaScript reaches a return statement, the function will stop executing.
  If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
  Functions often compute a return value. The return value is "returned" back to the "caller":

# Node-RED UI

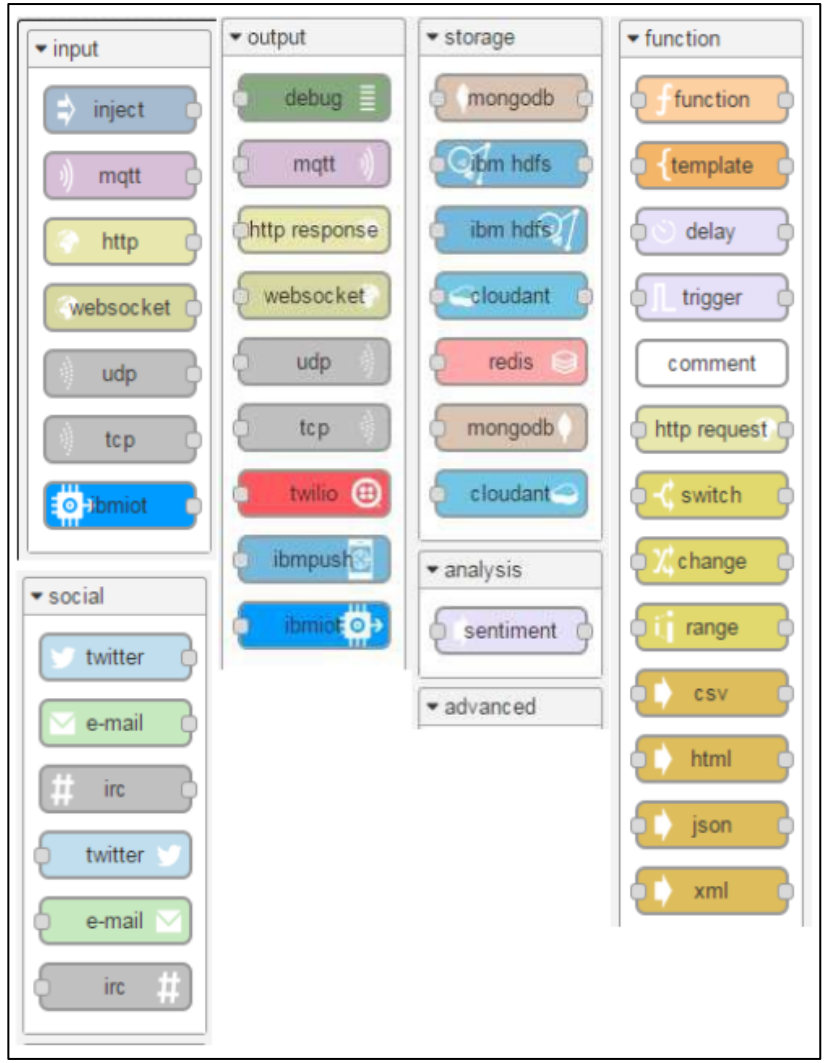


# Node-RED UI



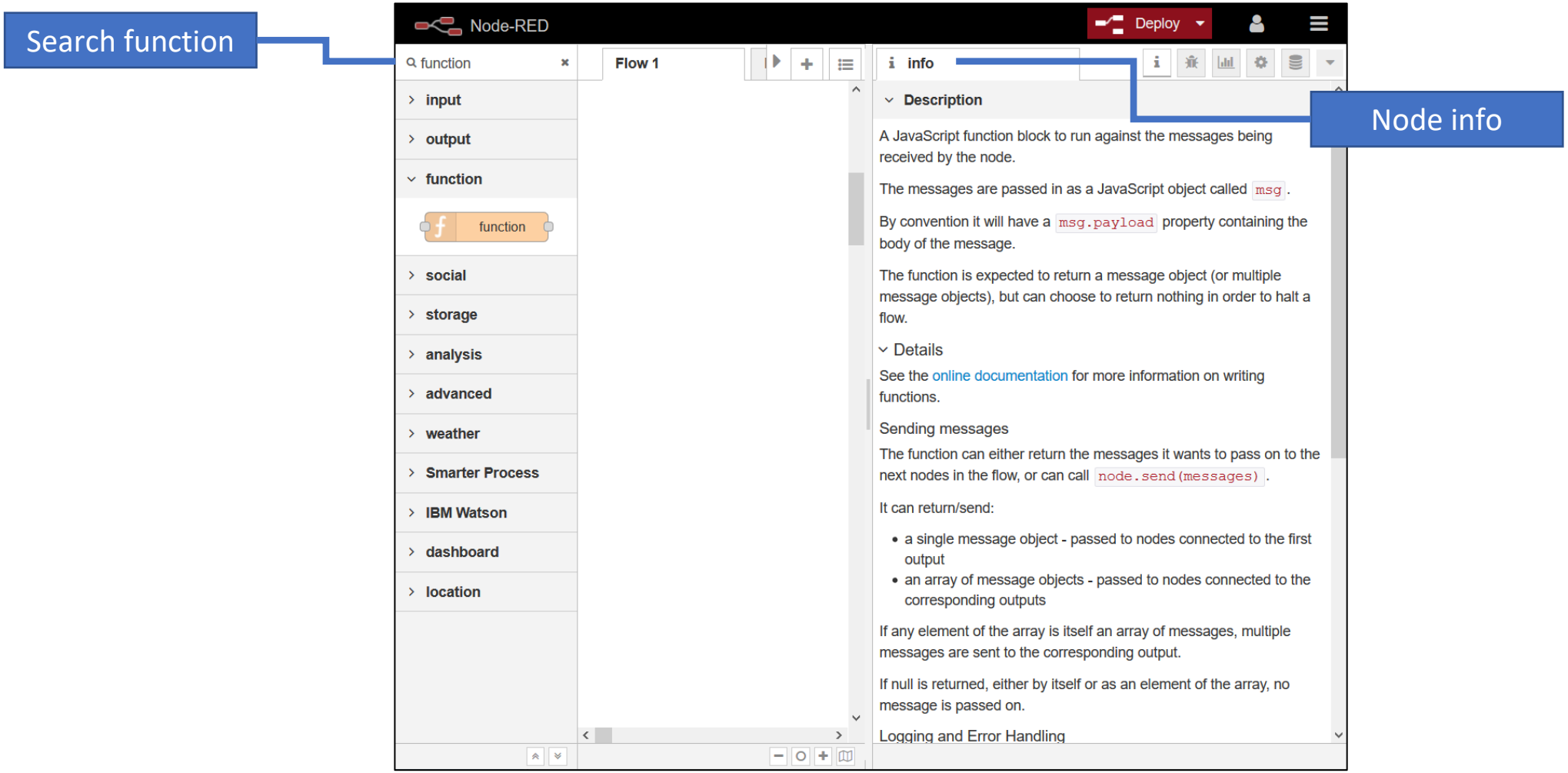
# Nodes

- Nodes seen by default are called “core nodes”
- External nodes can be installed for more functionality
- Nodes perform specific functions
  - Self-contained and reusable with most of JS program logic and API’s hidden in these colored boxes
- Nodes are used by dragging them into the node palette to workspace



# Node Information

- Click on the node and view Information on the Info tab



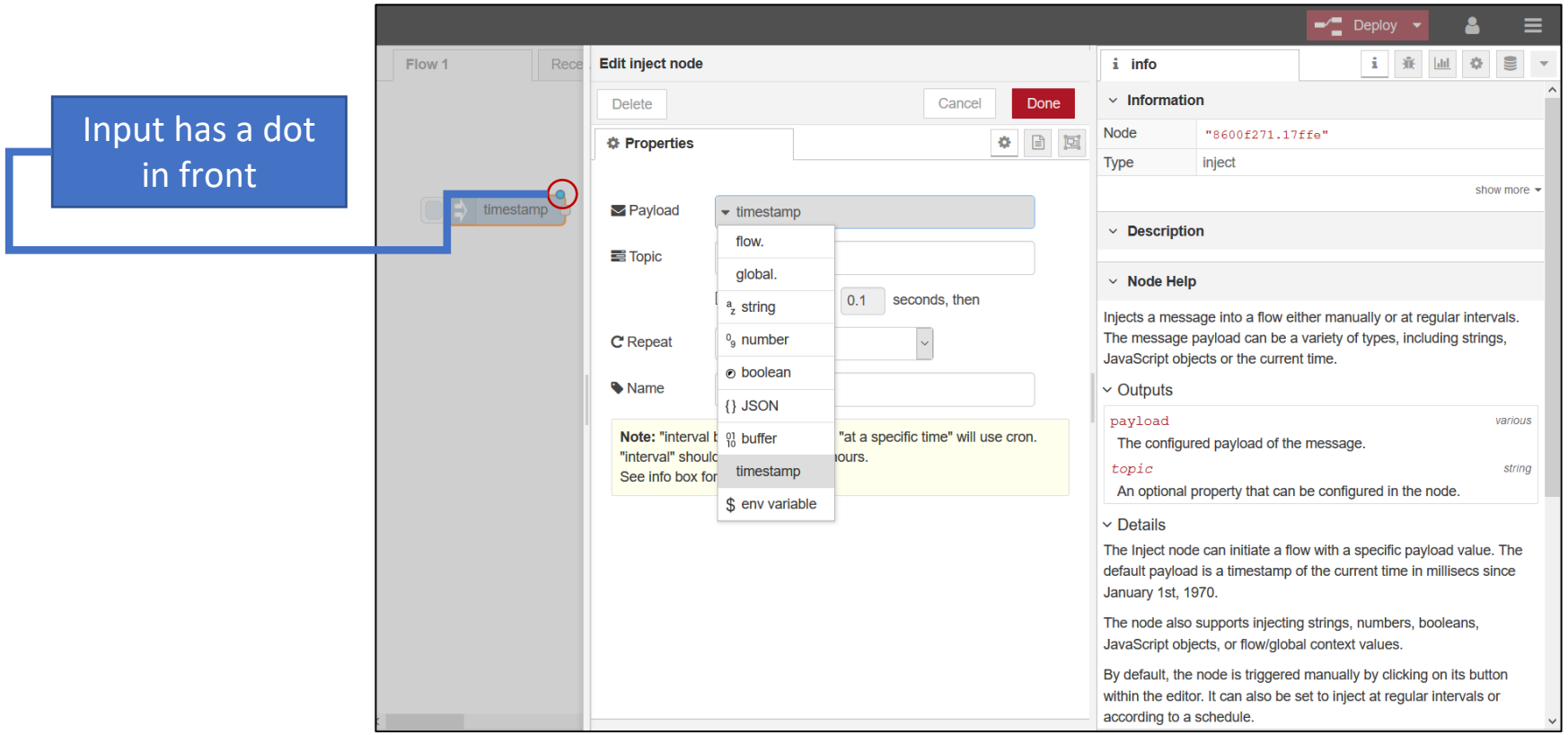
# Popular Core Nodes

- Inject Node
  - Allows manual triggering of flows
  - Can be scheduled to automatically inject at fixed intervals
- Debug Node
  - Shows message content, either just payload or entire object in the debug sidebar
- Function Node
  - Runs user-defined JS against the message flowing past
- HTTP Nodes
  - Define http endpoints for incoming REQUESTs, or trigger GETs of URLs in the middle of a flow
- MQTT Nodes
  - Define publishers or subscribers to a certain topic on a certain MQTT broker

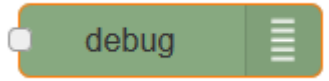
# Inject Node

- Input node 

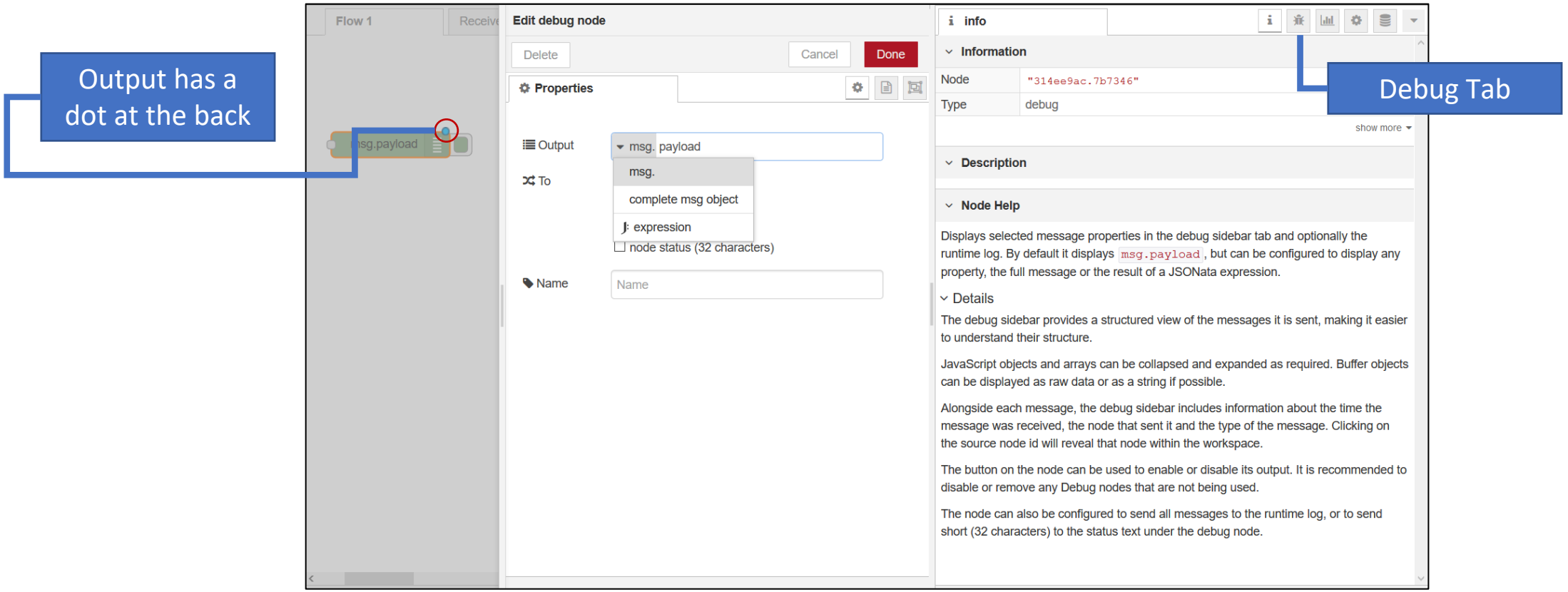
- Injects a message into a flow either manually or at regular intervals



# Debug Node

- Output node 

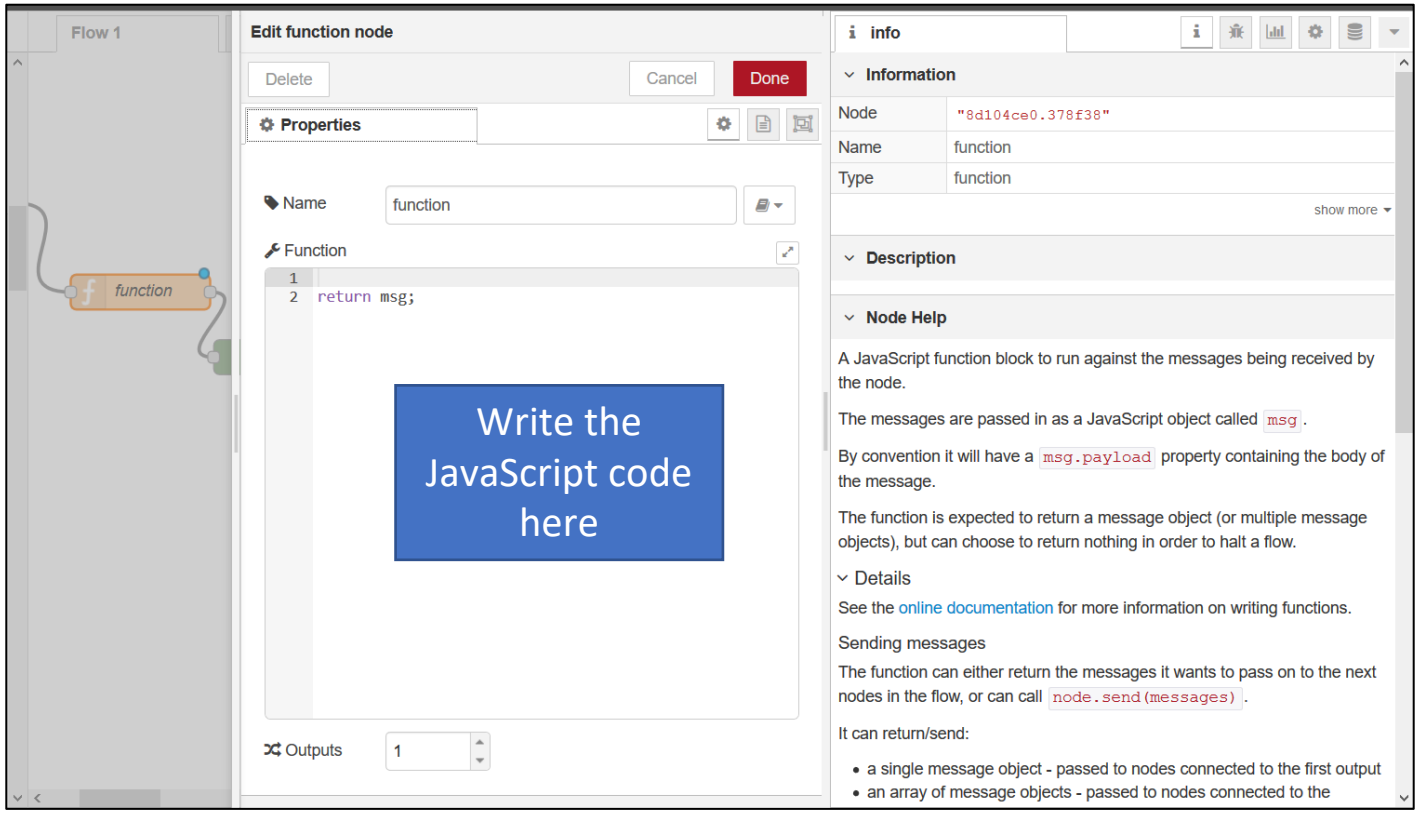
- Displays selected message properties in the **debug sidebar tab** and optionally the runtime log.



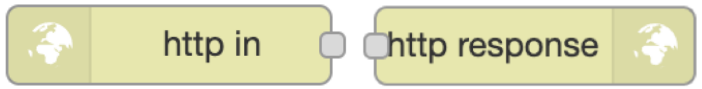
# Function Node

- Processing node 

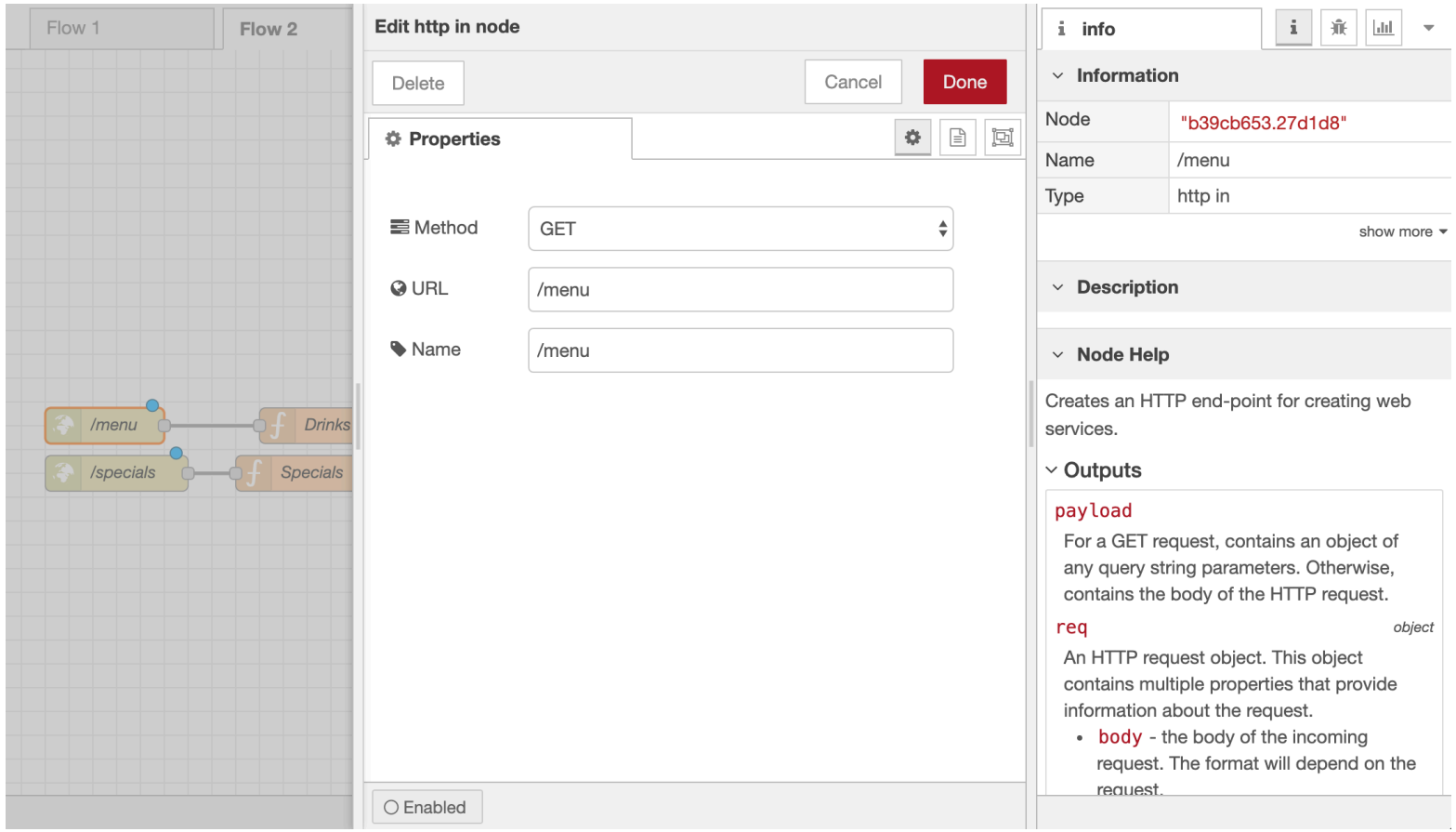
- A JavaScript function block to run against the messages being received by the node
- The messages are passed in as a JavaScript object called **msg**.



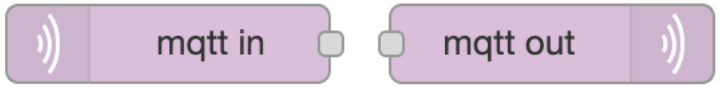
# HTTP Node

- Processing node 

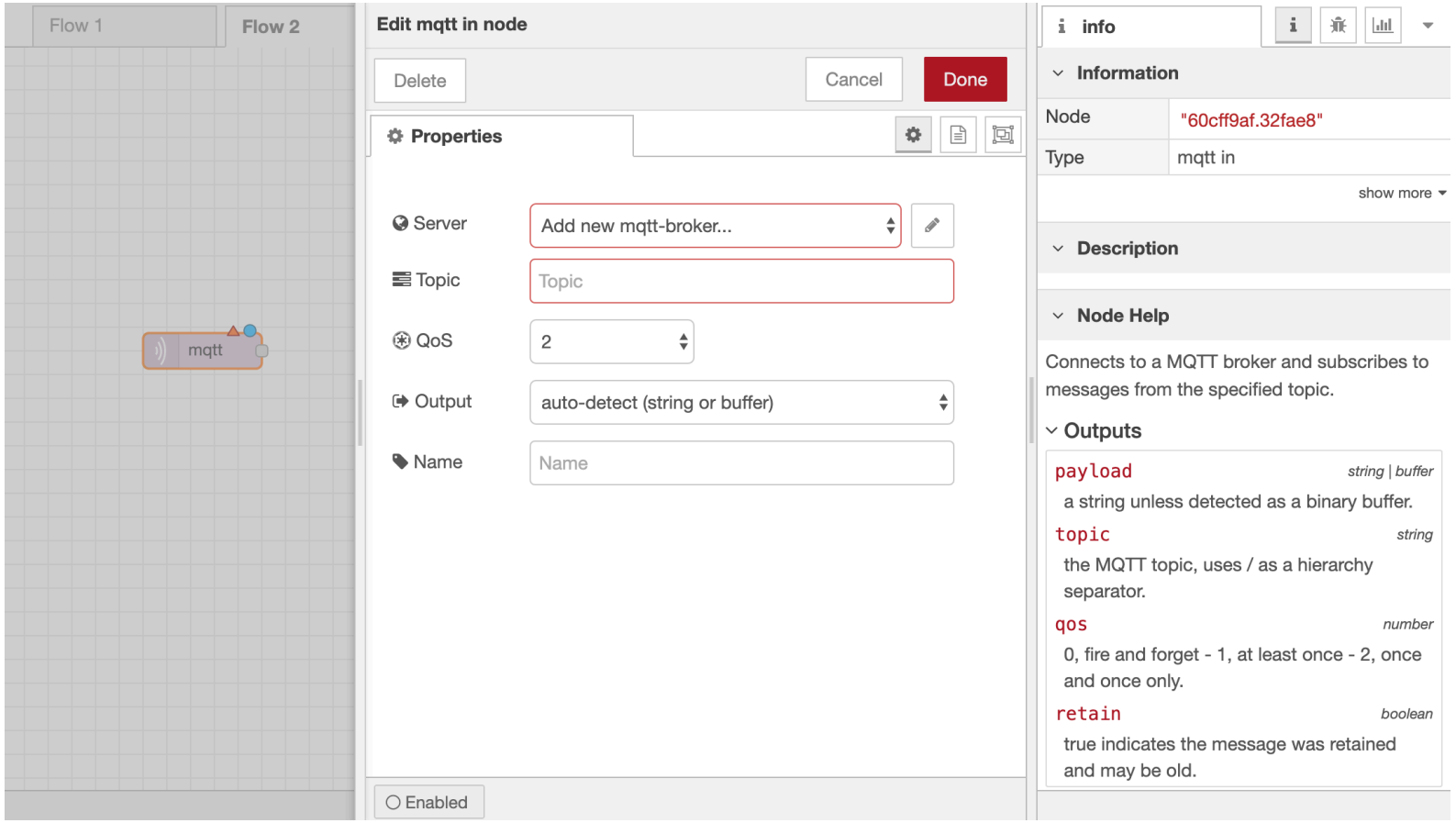
- Sends/Receives HTTP requests, allowing Node-RED to act as a basic web server. Message can contain standard URL-encoded data or JSON.



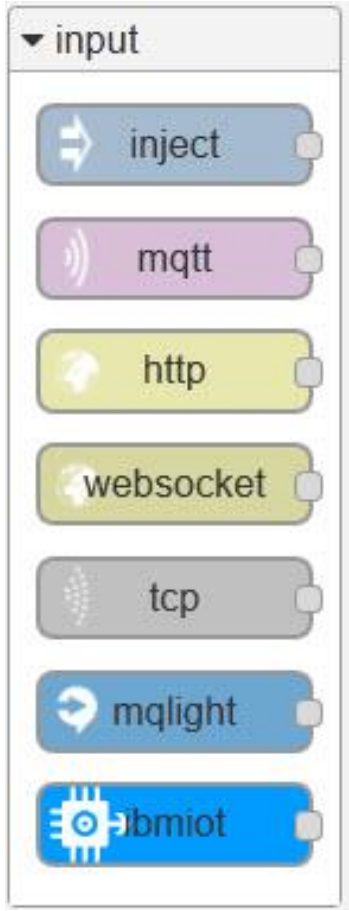
# MQTT Node

- Processing node 

- Subscribes to an MQTT broker and listens on a topic, returns any data published on the topic as a new message. Supports Quality of Service levels and last data retention.

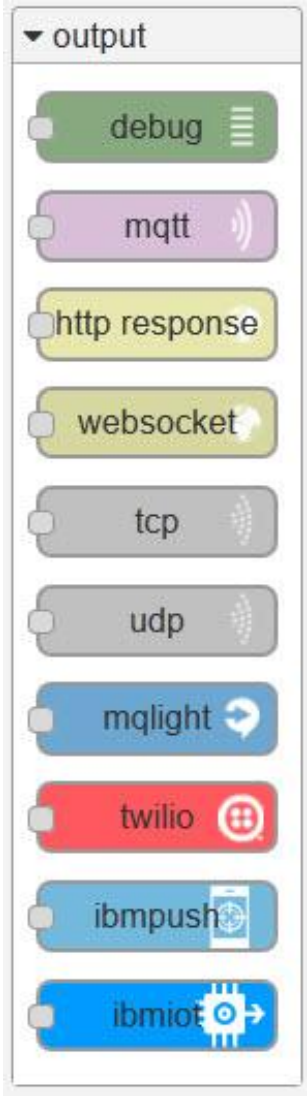


# Other Input Nodes



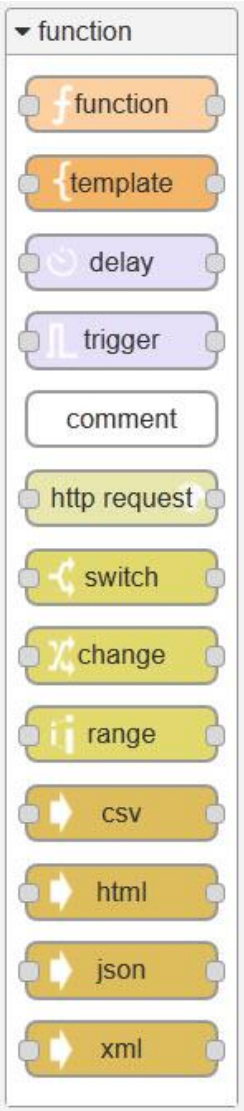

- **Websocket** - Provides an endpoint for a browser to establish a websocket connection with Node-RED. Offers a duplex connection for browser/server combinations.
- **TCP** - Used to accept incoming TCP requests on a specified port or to connect to a remote TCP port. Generates messages containing the TCP data as a single – or stream of – buffer, string or base64 encoded.
- **IBMIOT** – Receive messages from an attached IOT Foundation account
- …

# Other Output Nodes




- **IBMIOT** – send events out to the attached IOT Foundation account
- **Twilio** – send SMS messages via the Twilio service
- **IBM Push** – Send Push notifications to mobile devices
- Also can send requests through TCP, UDP, MQLight,.

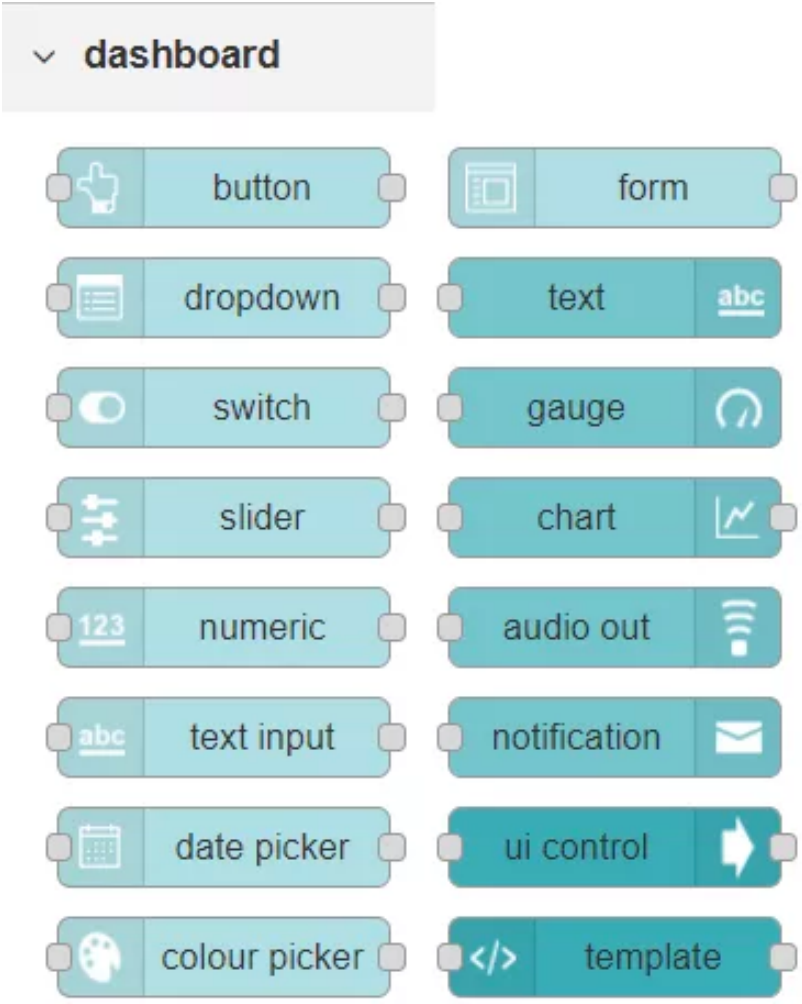
# Function Node Types

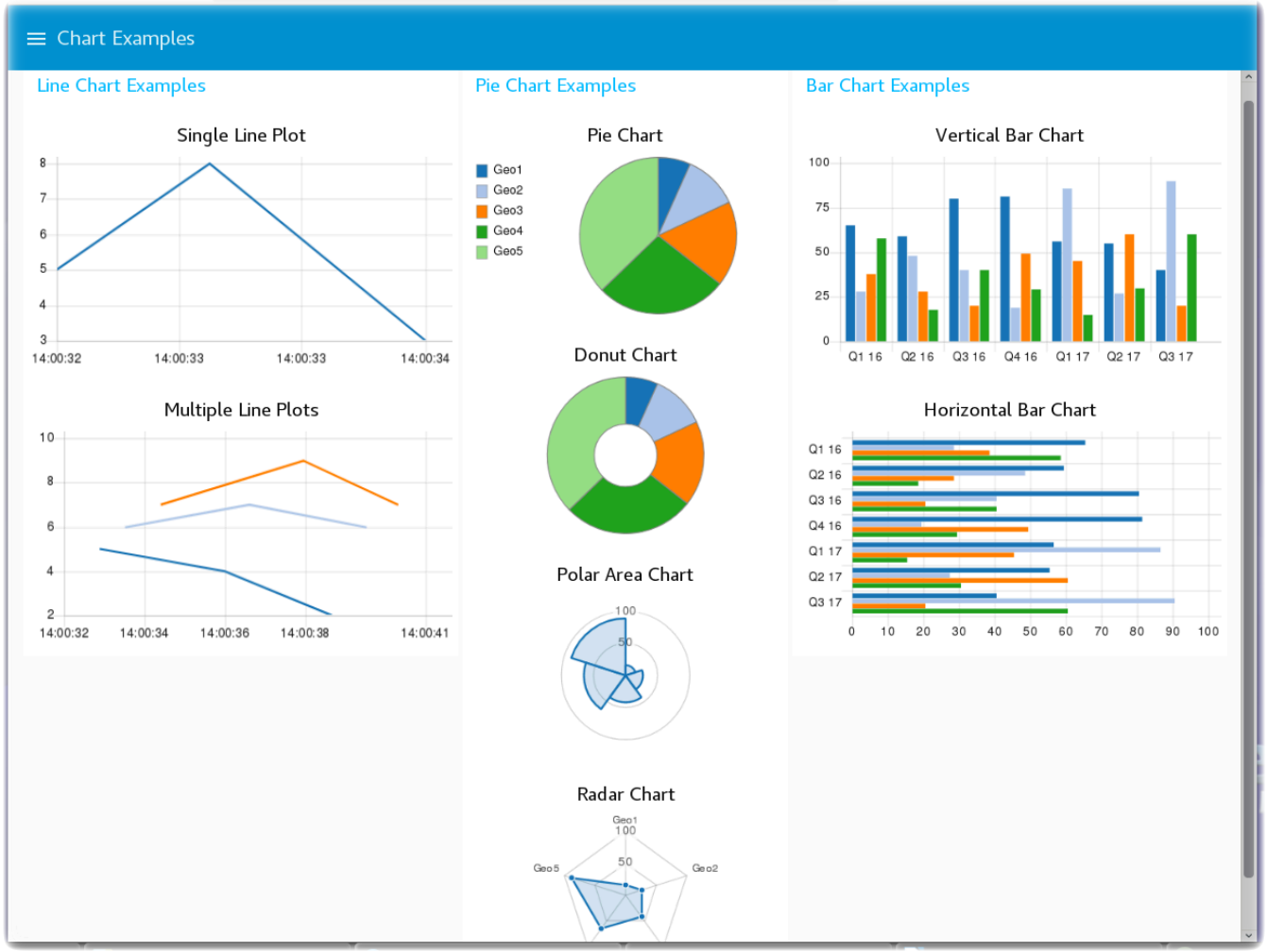



- **Template –** Configured with a template (using moustache format) of arbitrary complexity, takes an input message containing name:value pairs and inserts into the template.
- **Delay –** delays messages by a specific or random time. Can also be configured to throttle a message flow (e.g. 10 msg per sec).
- **Trigger –** Creates two output messages separated by a configurable time interval whenever an input message is received.
- **Switch** - routes messages based on their properties. Properties are configured using the UI and can be a variety of logic (>, <, >= etc) applied to a message property.
- **Change** - can be used to set, change or delete properties of incoming messages.
- **Range** - A simple scaling node that will map numerical input to a new output.
- **CSV/JSON/XML** – converts the JavaScript object to/from CSV/JSON/XML format.

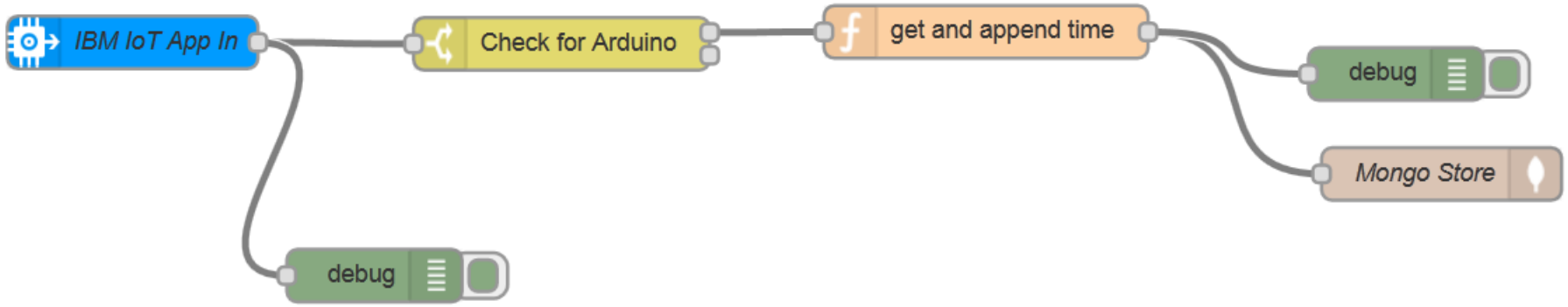
# Node-RED Dashboard

Node-RED Dashboard is a module that provides a set of nodes in Node-RED to quickly create a live data dashboard.



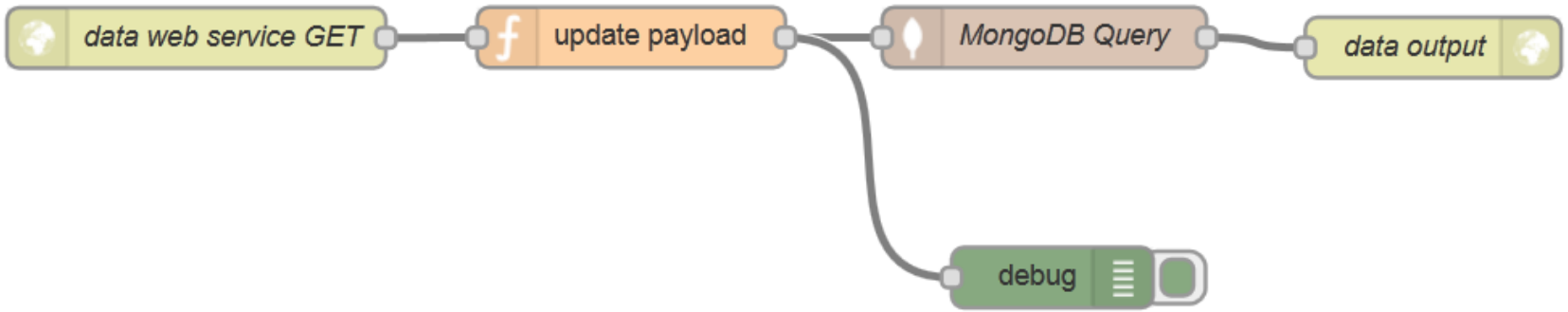



# Storing IoT Data



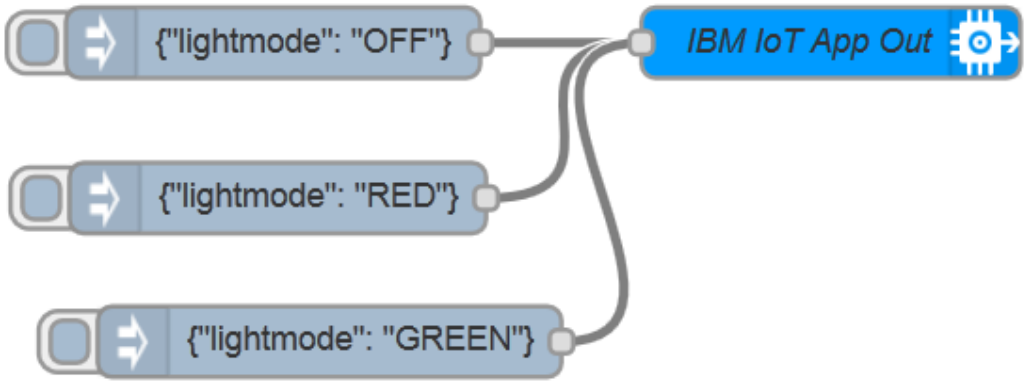

- IOT App In connects to the IOT foundation
- Check for Ardunio is a simple switch based on a property of the payload
- Get and append time is a function node that gets the current time and adds it to the payload
- MongoStore takes the final payload and places it in the MongoDB

# Building a RESTful service




- The Data Web Service GET is an HTTP input endpoint responding to a GET on /data
- Update Payload is a function node that changes the format of the payload to the MongoDB query format (avoiding Mongo injection attacks)
- MongoDB Query simply returns the result of evaluating what was in the payload (in Mongo DB query form, e.g. "payload.d.time": {$gt: x, $lt: y})
- DataOutput is an HTTP Response node returning the payload as its result to the caller

# Making actions happen – sending events




- The three inject nodes simply add a predetermined payload into the IBM IOT Output node
- Two approaches; different commands in the topic name, or different commands in the payload (using payload more common)