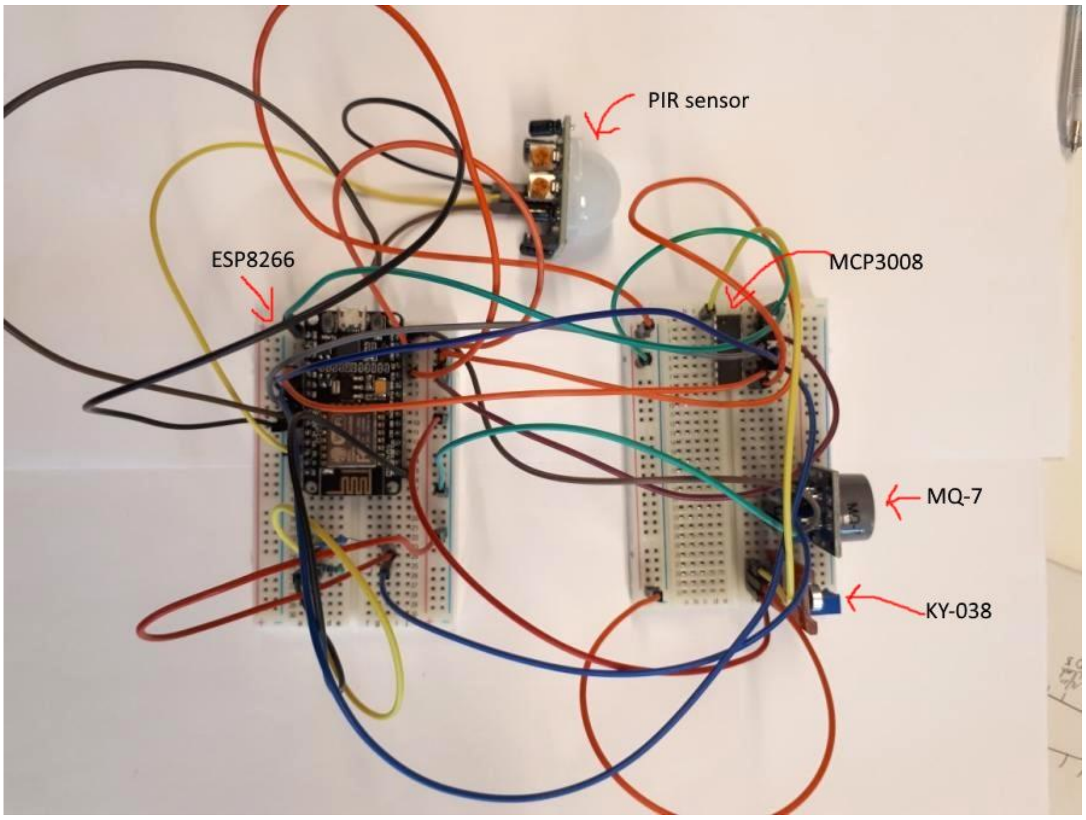
Bram Debontridder
Robin Lievens
Dean Terweduwe
Felix Roels

dB²: Project IoT

Design: Hardware

The components of this project are:

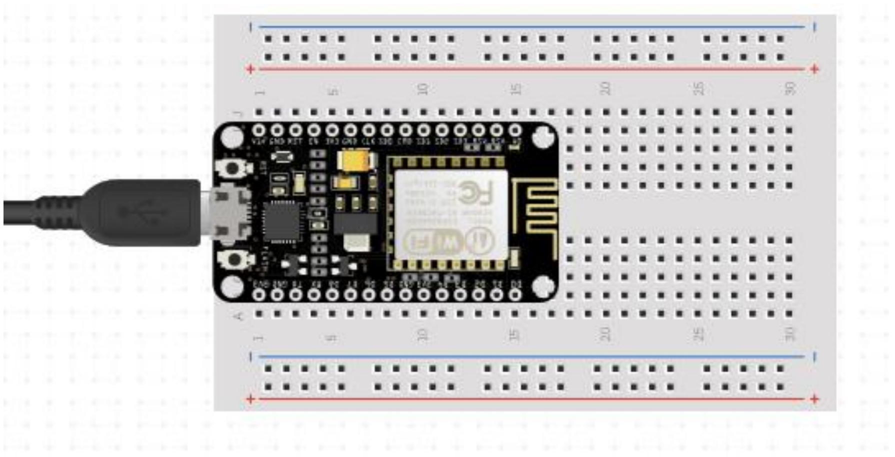
- ESP8266 NodeMCU 12-e
- 2 breadboards + a sufficient amount of Dupont wires
- MQ-7 CO sensor
- KY-038 sound level sensor
- PIR movement sensor
- MCP3008 10-bit Analog-to-Digital Converter
- 2 resistors: 470 Ohm + 1K Ohm



Overview of the 2 breadboards with all the components

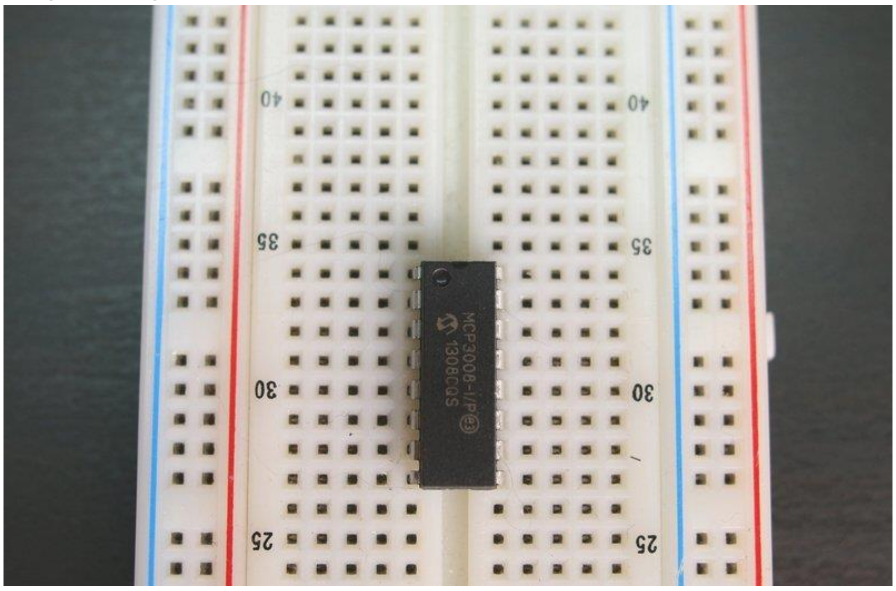
Hardware 1.1: Connecting all the components

Step 1: Plug in the ESP8266 on the breadboard.



The controller plugged into the breadboard

Step 2: Plug the MCP3008 into the other breadboard.



The MCP3008 plugged in.

Step 3: Connect the controllers to each other.

- Follow this overview to appropriately link the two together.

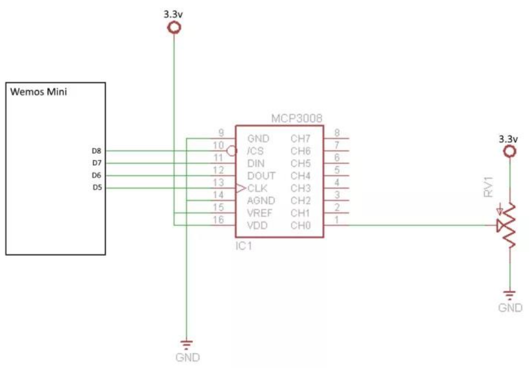
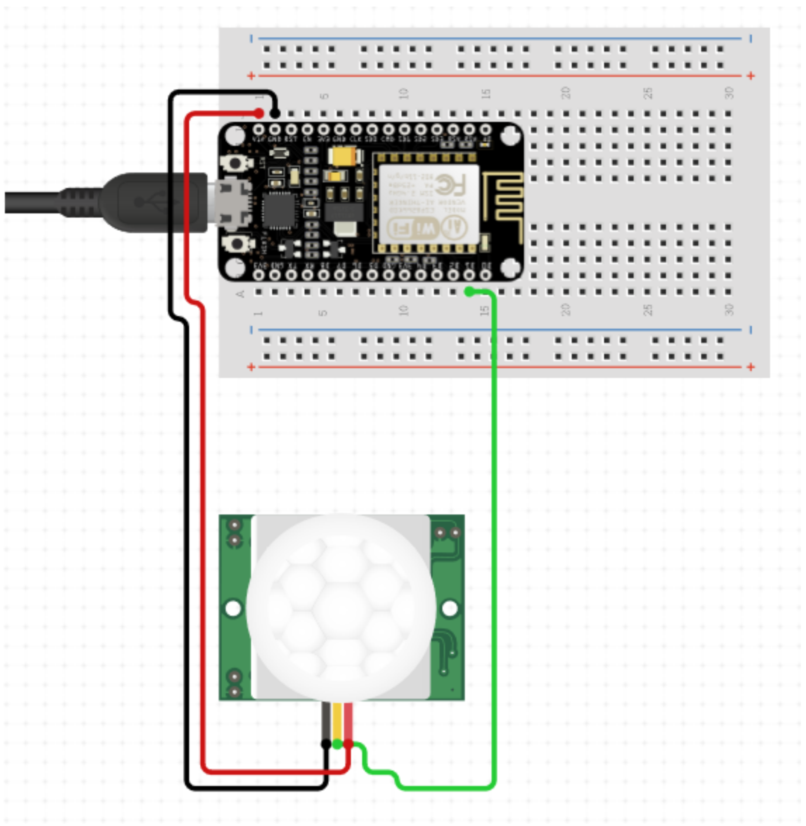


Diagram of the layout of the MCP3008 chip

Step 4: Connect the PIR sensor to the ESP8266 controller.

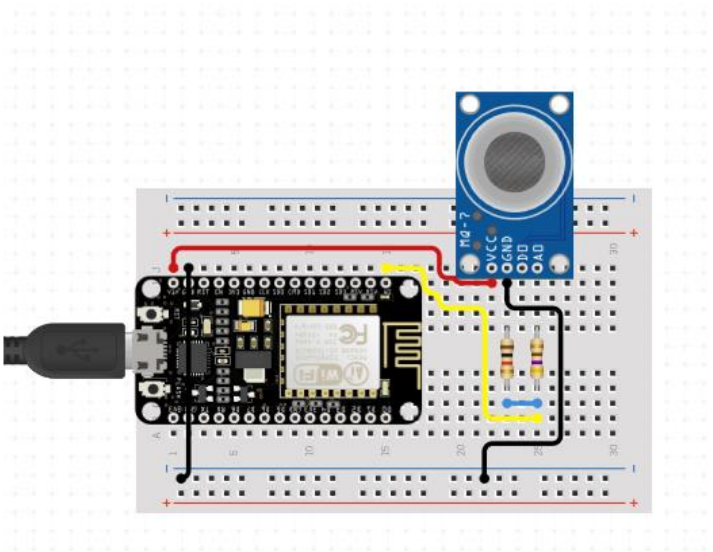
- 1 pin for ground
- 1 for signal
- 1 for power
- 3-5V → since the ESP8266 controller has 3 3v3 pins, we used one of these.



The PIR sensor connected to the ESP8266.
Note: From here on the diagrams will be one sensor at a time.

Step 5: Connect the MQ-7 CO sensor to everything.

- 4 pins in total
- We won't be using the D0 pin, just connect the A0 pin



The MQ-7 connected to the ESP8266 directly
Note: The analog out is connected to the 470 Ohm resistor.

Step 6: Connect the KY-038.

- Because we want to use an analog value for our microphone but only have one analog port, we need to use the MCP3008 (analog-to-digital).
- Video for wiring schematics.
- In the video they use a potentiometer for our microphone just use the A0, G and + port.

→ https://www.youtube.com/watch?v=NGNNDz_ylzs&t=385s

Arduino programming

2.1 Installation of software

If you haven't already downloaded the Arduino IDE, you should download it now.
https://www.arduino.cc/en/Main/Software

Install the ESP8266 Board

1. Open the preferences window from the Arduino IDE
2. Go to File > Preferences
3. Enter http://arduino.esp8266.com/stable/package_esp8266com_index.json into the "Additional Board Manager URLs" field as shown in the figure below.
4. Click OK
5. Go to Boards Manager: Tools > Board > Board Manager
6. Scroll down to ESP8266 and click install.
7. Choose the board from Tools > Board > NodeMCU V1.0 ESP 8266-12E and select CPU Frequency 80MHz and Upload Speed: 115200.

Install the PubSubClient

1. Tools > Board Manager > Type **PubSubClient** into the search.
2. Click on the one by **Nick O'Leary**

Install the EasyNTPClient

1. Tools > Board Manager > Type **EasyNTPClient** into the search.
2. Click on the one by **Harsha Alva**

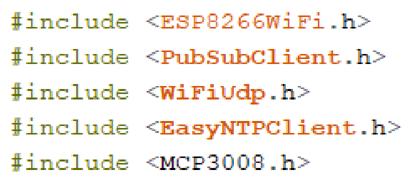
Install the MCP3008 library

From: https://github.com/nodesign/MCP3008

1. Sketch > Use libraries > .ZIP add library

2.2 Writing the program

1. Include the right libraries.



```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <WiFiUdp.h>
#include <EasyNTPClient.h>
#include <MCP3008.h>
```

2. Write the code to make the connection via MQTT to IBM Watson

```
//Watson IoT connection details
#define MQTT_HOST "<YOUR_ORG>.messaging.internetofthings.ibmcloud.com"
#define MQTT_PORT 1883
#define MQTT_DEVICEID "d:<YOUR_ORG>:<DEVICE_TYPE>:<DEVICE_ID>"
#define MQTT_USER "use-token-auth"
#define MQTT_TOKEN "<DEVICE_TOKEN>"
#define MQTT_TOPIC "iot-2/evt/<EVENT_NAME>/fmt/json"
#define MQTT_TOPIC_DISPLAY "iot-2/cmd/update/fmt/json"
```

3. Define the right pins for the sensors

```
// Sensors
#define CS_PIN D8
#define CLOCK_PIN D5
#define MOSI_PIN D7
#define MISO_PIN D6
```

4. Connect to the network

```
// Update these with values suitable for your network.
const char* ssid = "<SSID>";
const char* password = "<PASSWORD>";
const char* mqtt_server = MQTT_HOST;
```

5. we'll be using WifiClient and PubSubClient to connect to our wifi network and the EasyNTPClient to get Unix Time for our timestamp later on.

```
WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;

WiFiUDP udp;
EasyNTPClient  ntpClient(udp, "time.google.com", 7200);
```

6. Some code so that the MCP3008 works with our sensors.

```
//Sensors
MCP3008 adc(CLOCK_PIN, MOSI_PIN, MISO_PIN, CS_PIN);
int gemDb;
int gemPir;
int gemCo;
```

7. Write the setup() method

```
void setup() {
  pinMode(BUILTIN_LED, OUTPUT);     // Initialize the BUILTIN_LED pin as an output
  Serial.begin(9600);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  // open serial port
  ntpClient.getUnixTime();
 gemDb =0;
 gemPir=0;
 gemCo=0;
}
```

8. Write the setup_wifi() method so our arduino can connect to the internet.

```
void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

9. Next we write the callback() method

```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // Switch on the LED if an 1 was received as first character
  if ((char)payload[0] == '1') {
    digitalWrite(BUILTIN_LED, LOW);   // Turn the LED on (Note that LOW is the voltage level
    // but actually the LED is on; this is because
    // it is acive low on the ESP-01)
  } else {
    digitalWrite(BUILTIN_LED, HIGH);  // Turn the LED off by making the voltage HIGH
  }

}
```

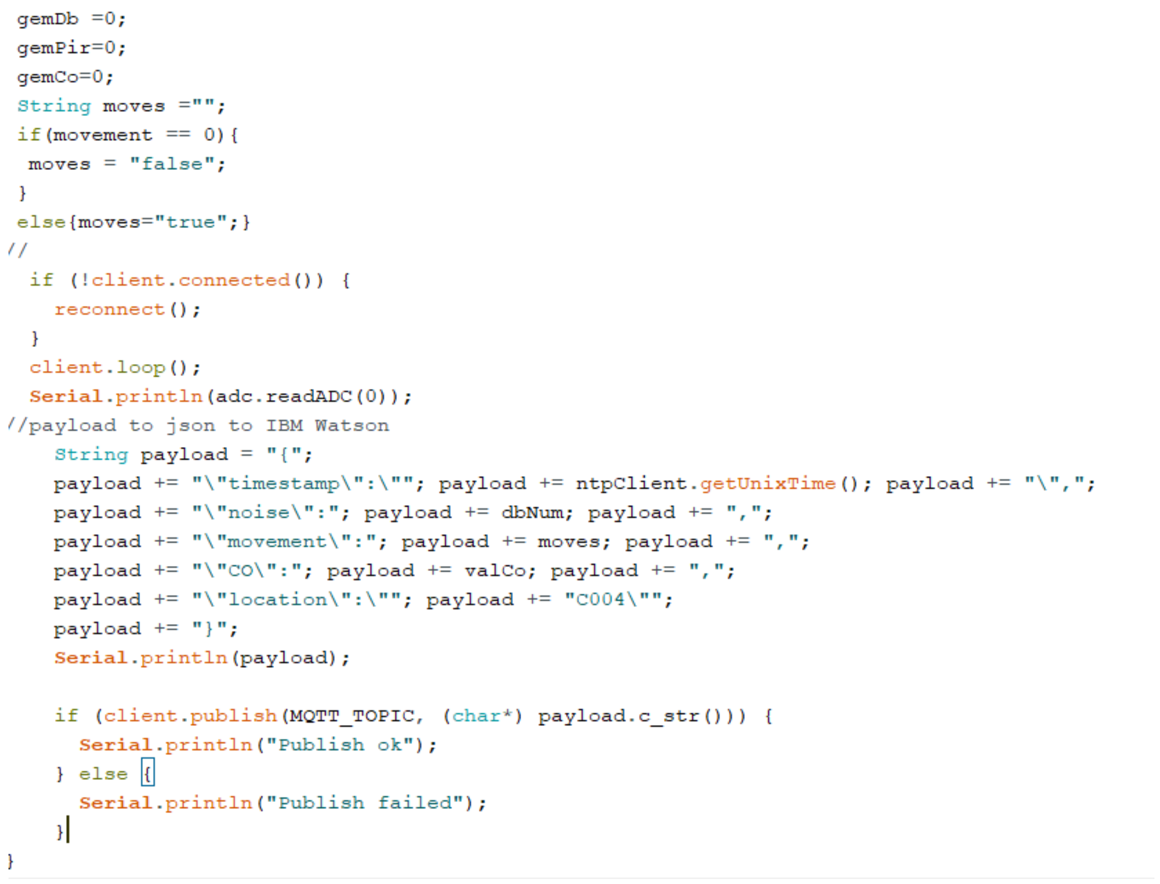
10. Reconnect() in case we lose our MQTT connection

```
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect(MQTT_DEVICEID, MQTT_USER, MQTT_TOKEN)) {
      Serial.println("connected");
      client.subscribe(MQTT_TOPIC_DISPLAY);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

11. loop() method: values coming from the sensors turned into values we can work with

```
void loop() {
  // Sensor data
   for(int i = 0; i < 100; i++){
 gemDb=gemDb+adc.readADC(0);
 gemPir = gemPir + digitalRead(D2);
 gemCo=gemCo + analogRead(A0);

  delay(100);
 }

 gemDb = gemDb/100;
 int valPir = gemPir;
 int valCo =gemCo/100;
 int dbNum = 0;
 bool movement = false;
 if(gemPir >=30) {
  movement=true;
 }
```

12. loop() method part 2: we put our values in a json format.



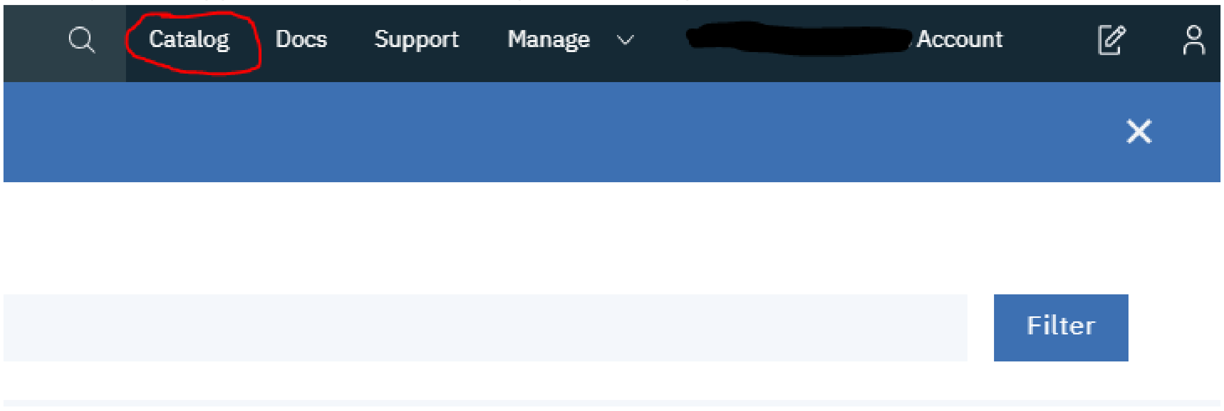
```
 gemDb =0;
 gemPir=0;
 gemCo=0;
 String moves ="";
 if(movement == 0){
  moves = "false";
 }
 else{moves="true";}
//
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  Serial.println(adc.readADC(0));
//payload to json to IBM Watson
    String payload = "{";
    payload += "\"timestamp\":\""; payload += ntpClient.getUnixTime(); payload += "\",";
    payload += "\"noise\":"; payload += dbNum; payload += ",";
    payload += "\"movement\":"; payload += moves; payload += ",";
    payload += "\"CO\":"; payload += valCo; payload += ",";
    payload += "\"location\":\""; payload += "C004\"";
    payload += "}";
    Serial.println(payload);

    if (client.publish(MQTT_TOPIC, (char*) payload.c_str())) {
      Serial.println("Publish ok");
    } else {
      Serial.println("Publish failed");
    }
}
```

MQTT

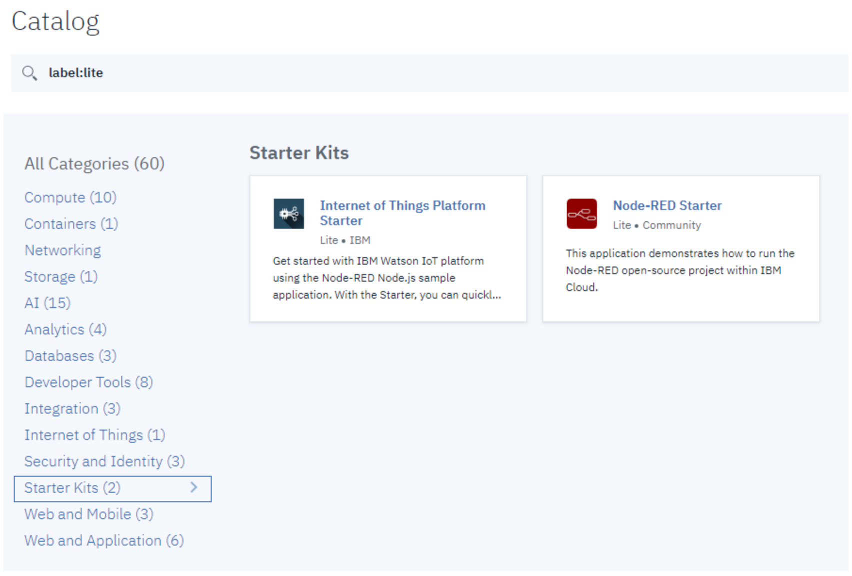
For communication between our project and computer, we use the MQTT protocol, using IBM Cloud as a broker.
To make this work we need access to the following software:
- An IBM Cloud account (https://cloud.ibm.com/login) which grants access to:
    - IBM Cloud
    - Node-RED
    - IBM Watson IoT Platform
- OPTIONAL: MQTT.fx (https://mqttfx.jensd.de/) (or any other MQTT client application) to check your MQTT connection.

1. Register a device

Start by making a free account on IBM Cloud (https://cloud.ibm.com/login). Follow the steps of the registration and login on the platform.
When you're logged in, click on "Catalog" in the navigation bar on top.



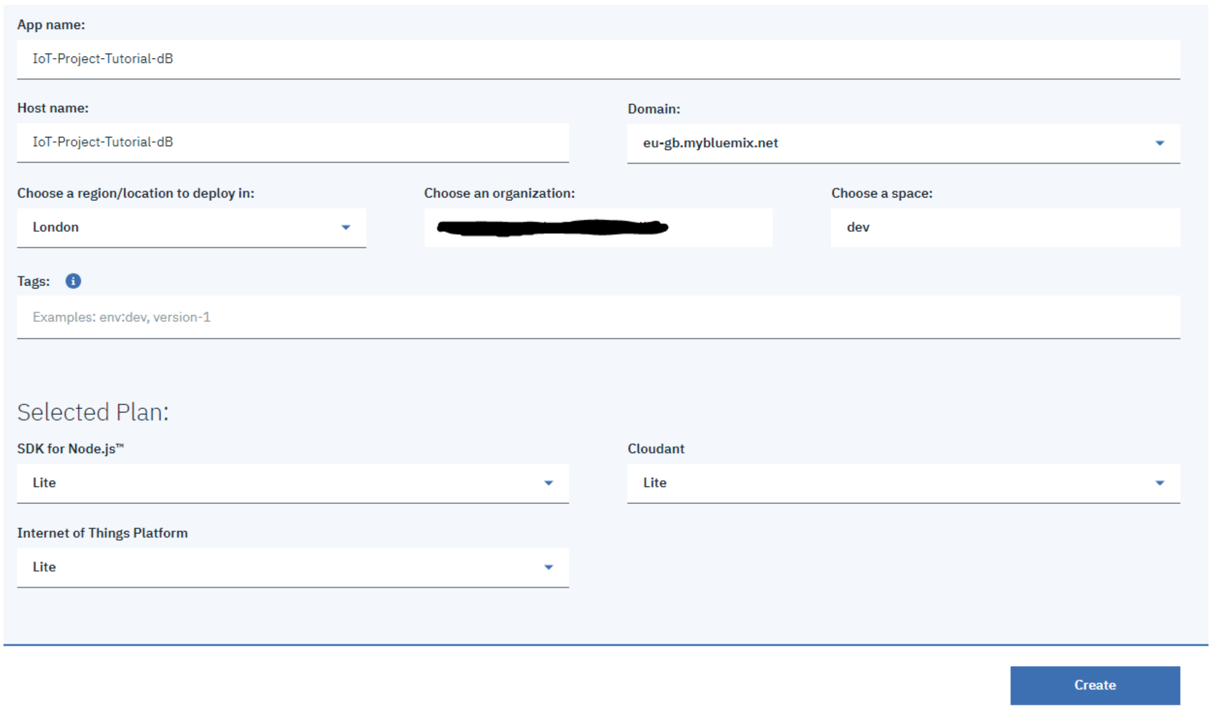
On the catalog page, click "Starter Kits" in the categories menu.
Only starter kits should show up now. Click on the "Internet of Things Platform Starter"



Get started with IBM Watson IoT platform using the Node-RED Node.js sample application. With the Starter, you can quickl...

This application demonstrates how to run the Node-RED open-source project within IBM Cloud.

Change the "App name" to anything you want and leave the rest on default.



App name:
IoT-Project-Tutorial-dB

Host name:
IoT-Project-Tutorial-dB

Domain:
eu-gb.mybluemix.net

Choose a region/location to deploy in:
London

Choose an organization:
[redacted]

Choose a space:
dev

Tags:
Examples: env:dev, version-1

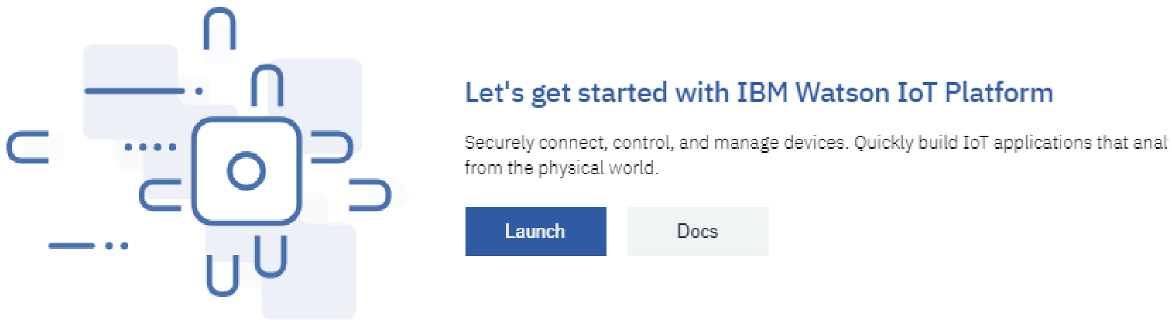
Selected Plan:

SDK for Node.js™
Lite

Cloudant
Lite

Internet of Things Platform
Lite

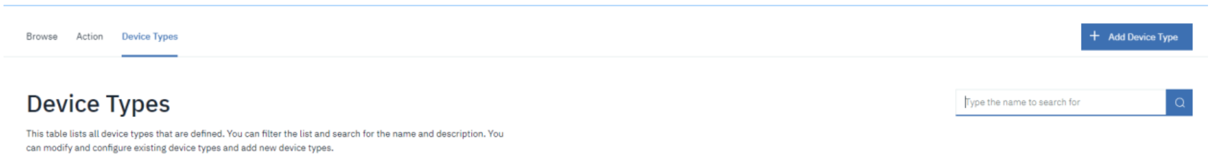
Click the blue “Create” button in the bottom right corner and wait a few minutes for it to finish.
Next, click on the hamburger menu in the top left corner and click on “Resource List”.
Under “Cloud Foundry Services”, click on the name of your service (not the NoSQLDB).
This brings you to a new page, click the blue “Launch” button.



Let's get started with IBM Watson IoT Platform

Securely connect, control, and manage devices. Quickly build IoT applications that anal
from the physical world.

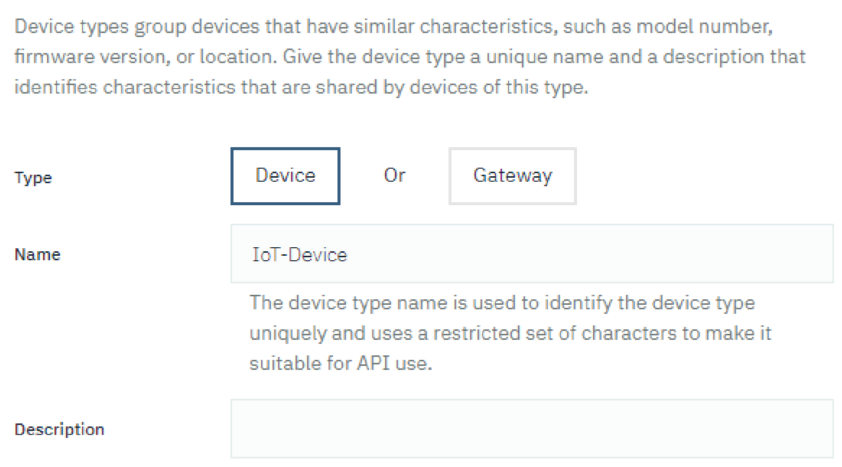
A new tab of the IBM Watson IoT Platform opens. Click “Device Types” and create a new device type by clicking the blue button “Add Device Type”.



Device Types

This table lists all device types that are defined. You can filter the list and search for the name and description. You can modify and configure existing device types and add new device types.

Choose a name for the device type and click “Next”. Click “Done”.

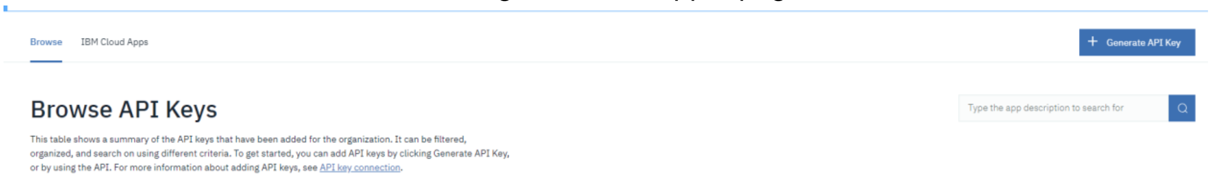

Device types group devices that have similar characteristics, such as model number, firmware version, or location. Give the device type a unique name and a description that identifies characteristics that are shared by devices of this type.

Type	[✓ Device] Or Gateway
Name	IoT-Device
	The device type name is used to identify the device type uniquely and uses a restricted set of characters to make it suitable for API use.
Description	

Now, register a new device of this type by clicking “Register Devices”.
Choose a Device ID and click “Next”, “Next”, “Next”, “Done”.
Your device is now registered. Copy the Authentication Token and save it somewhere, you will need this later.
Press the return button in your browser to go back to the Devices page. Your device is now in the list.

2. Generate an API key

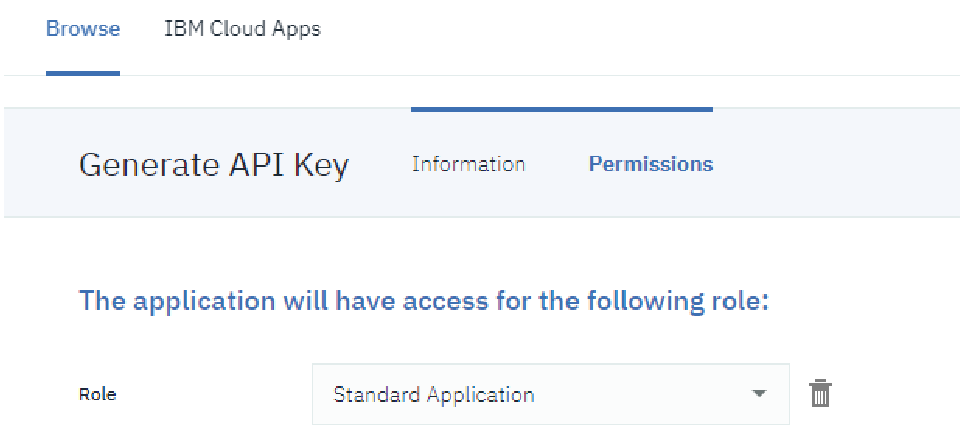
On the IBM Watson IoT Platform, navigate to the “Apps” page via the left menu.



Browse API Keys

This table shows a summary of the API keys that have been added for the organization. It can be filtered, organized, and search on using different criteria. To get started, you can add API keys by clicking Generate API Key, or by using the API. For more information about adding API keys, see API key connection.

Next, generate an API key by pressing the blue button in the top right corner.
Add a description if you want, this is not necessary. Click Next.
Set the role to “Standard Application” and click “Generate Key”.



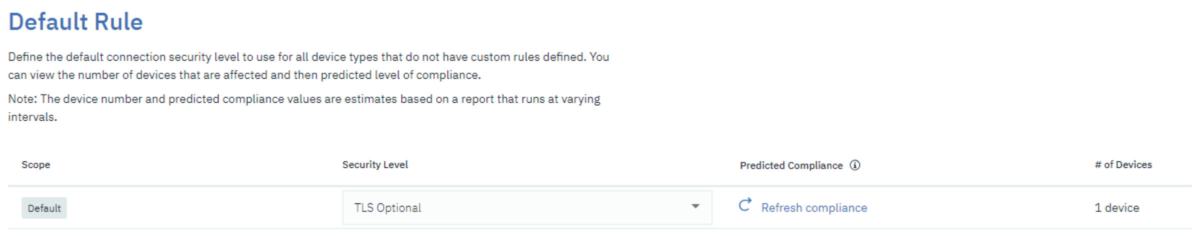
The application will have access for the following role:

Role: Standard Application

Copy the key and authentication token and save it somewhere, you will need this later.

3. Security settings

On the IBM Watson IoT Platform, navigate to the “Security” page via the left menu. Click the blue icon on the right side of “Connection Security” to edit these settings. Set security level to “TLS Optional”.



Default Rule

Define the default connection security level to use for all device types that do not have custom rules defined. You can view the number of devices that are affected and then predicted level of compliance.

Note: The device number and predicted compliance values are estimates based on a report that runs at varying intervals.

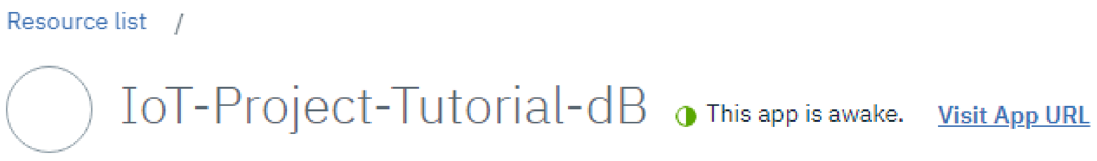
Scope	Security Level	Predicted Compliance	# of Devices
Default	TLS Optional	Refresh compliance	1 device

Press “Save” in the top right corner to save your settings.

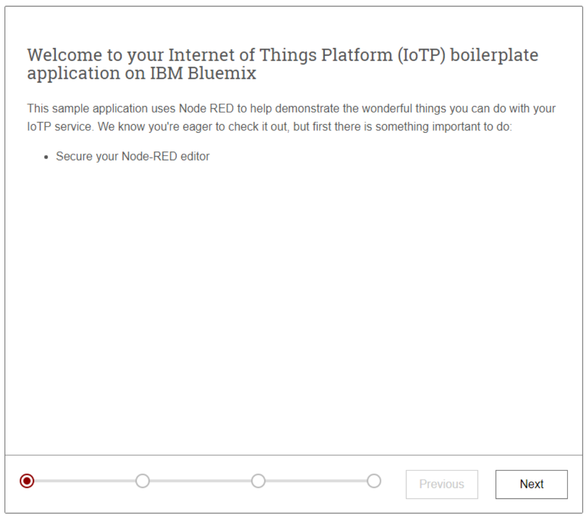
Node-RED

1. Setup

Go back to your Resource list on IBM Cloud (https://cloud.ibm.com/resources). Now, under “Cloud Foundry Apps”, click on your listed app. A new pages opens. Click “Visit App URL”.



The Node-RED page opens.

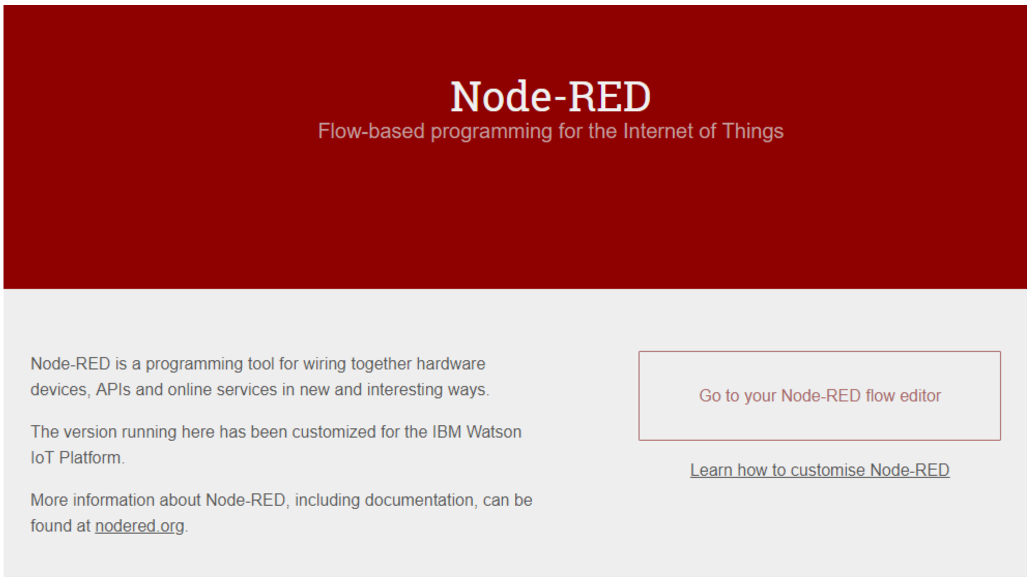


Welcome to your Internet of Things Platform (IoTP) boilerplate application on IBM Bluemix

This sample application uses Node RED to help demonstrate the wonderful things you can do with your IoTP service. We know you're eager to check it out, but first there is something important to do:

- Secure your Node-RED editor

Click “Next”.
Choose a username and password and click “Next”. Click “Finish”.


Node-RED
Flow-based programming for the Internet of Things

Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways.

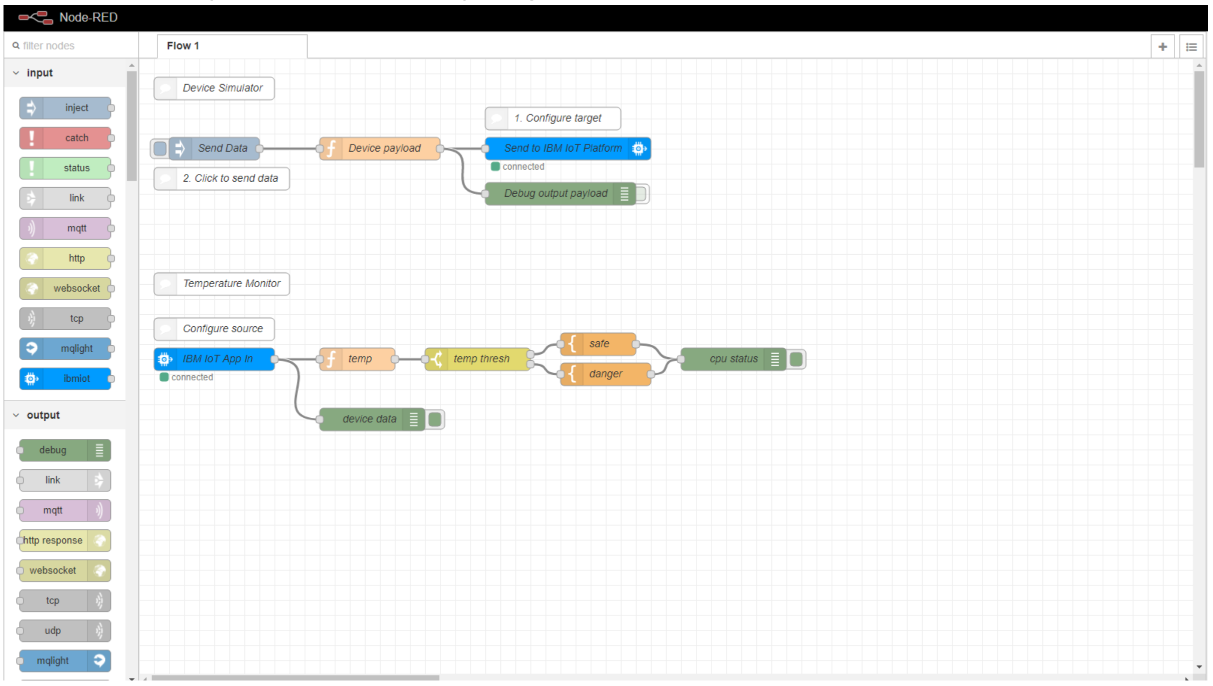
The version running here has been customized for the IBM Watson IoT Platform.

More information about Node-RED, including documentation, can be found at nodered.org.

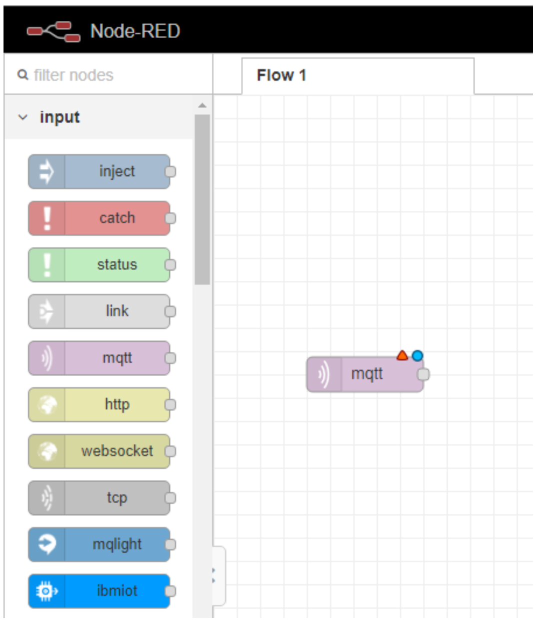
Go to your Node-RED flow editor

Learn how to customise Node-RED

Click “Go to your Node-RED flow editor”.
Login with your credentials you just created.
On this screen, you can select everything and delete it. We don’t need it.



Start by selecting and dragging an “mqtt” input node on the grid.



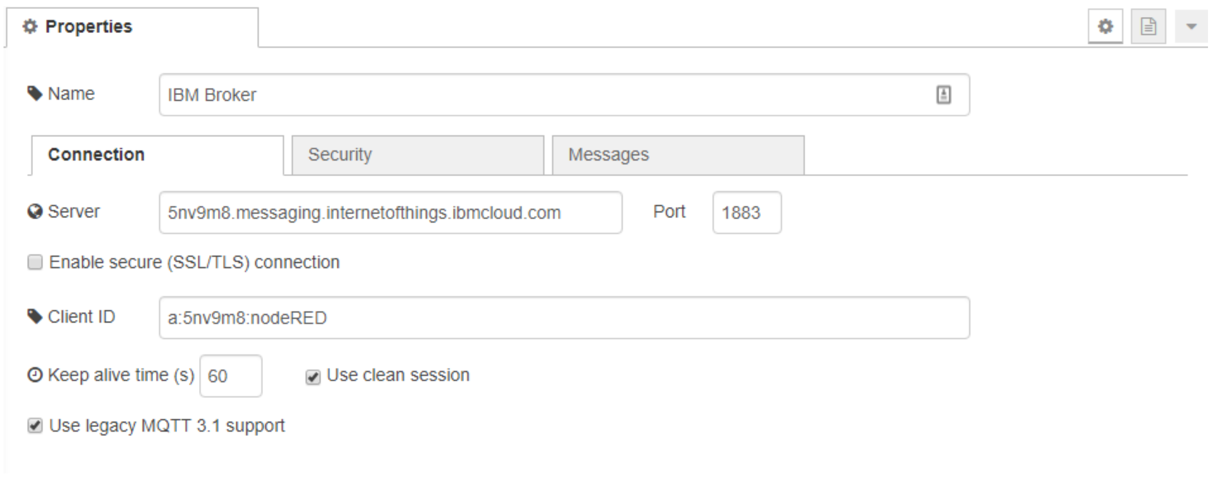
Double click the node to change the properties.
Next to “Add new mqtt-broker…” click the icon on the right to create one.
Fill in a name of your choice.
As server address, fill in the following:

YOUR_ORG_ID.messaging.internetofthings.ibmcloud.com

to find your ORG_ID, open IBM Watson. Your ORG_ID is shown under your name as “ID:”.
As Client ID, fill in the following:

a:YOUR_ORG_ID:YOUR_CHOICE

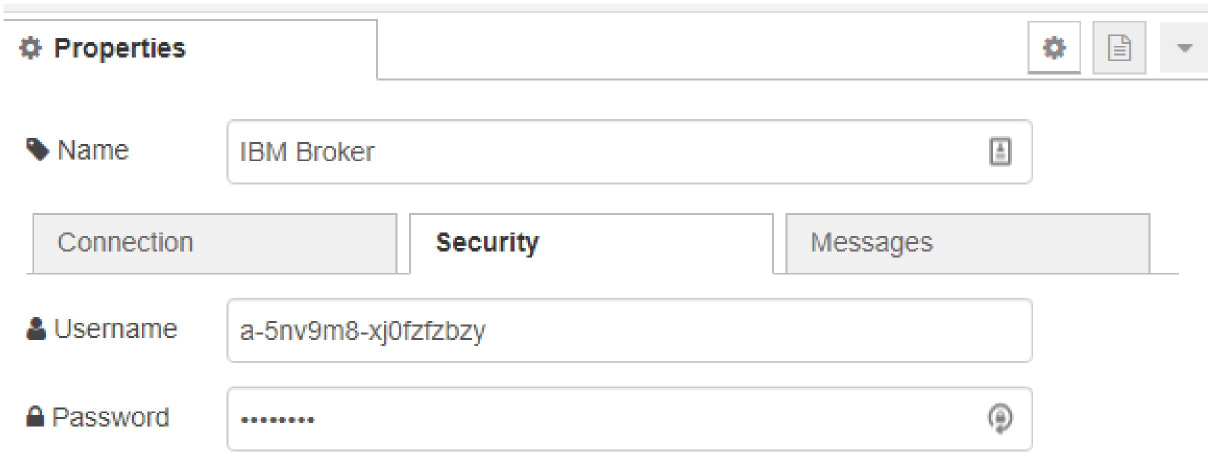
YOUR_ORG_ID is the same as above.
YOUR_CHOICE is a name of your choice, fill in any string you want.



Name: IBM Broker
Server: 5nv9m8.messaging.internetofthings.ibmcloud.com Port: 1883
☐ Enable secure (SSL/TLS) connection
Client ID: a:5nv9m8:nodeRED
Keep alive time (s): 60 ☑ Use clean session
☑ Use legacy MQTT 3.1 support

Now, click on the “Security” tab.
In the username field, fill in your API key, which you generated on IBM Watson and saved somewhere.

In the password field, fill in your API authentication token, which you saved somewhere.



Name: IBM Broker

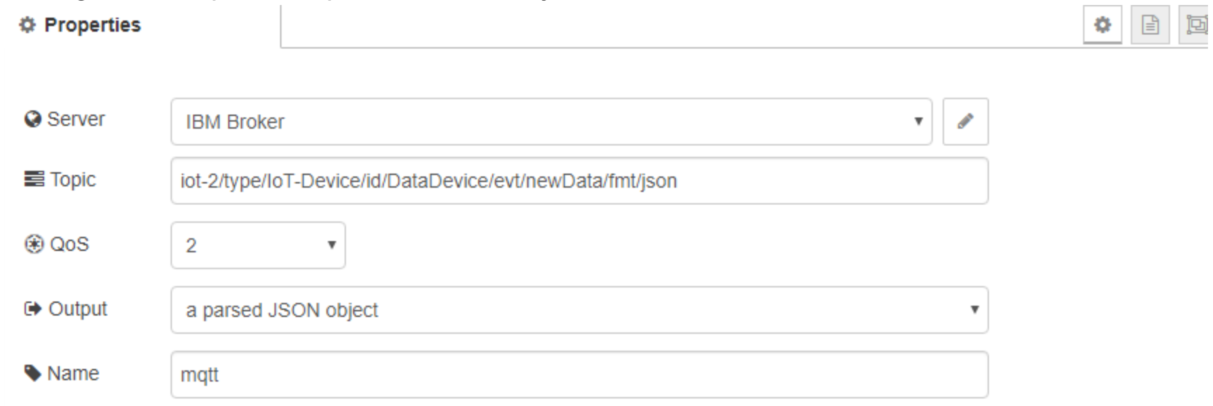
Username: a-5nv9m8-xj0fzfzbzy

Password: •••••••••

You are now configured to connect to your broker.
Go back to the client configuration.
In the topic field, fill in the following:

iot-2/type/YOUR_DEVICE_TYPE/id/YOUR_DEVICE_ID/evt/YOUR_EVENT/fmt/json

YOUR_DEVICE_TYPE is the device type you created on IBM Watson.
YOUR_DEVICE_ID is the device ID you created on IBM Watson.
YOUR_EVENT is the event you created in your ESP.
Fill in a name of your choice.
Change the Output to “a parsed JSON object” and click “Done”.



Server: IBM Broker

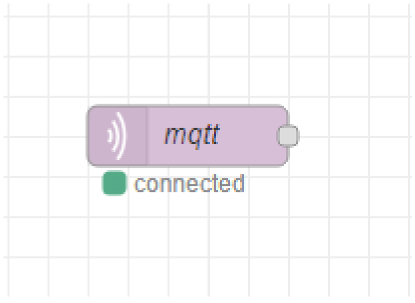
Topic: iot-2/type/IoT-Device/id/DataDevice/evt/newData/fmt/json

QoS: 2

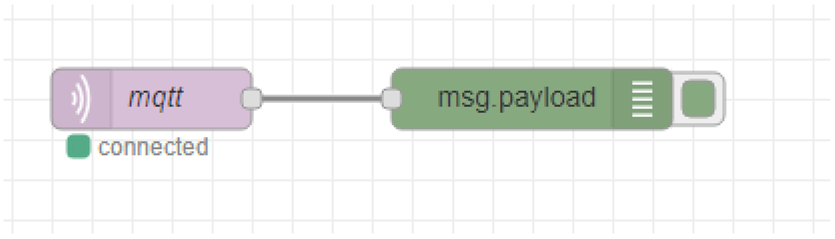
Output: a parsed JSON object

Name: mqtt

Click “Deploy” in the top right corner. Your mqtt node should now have “connected” displayed under it.

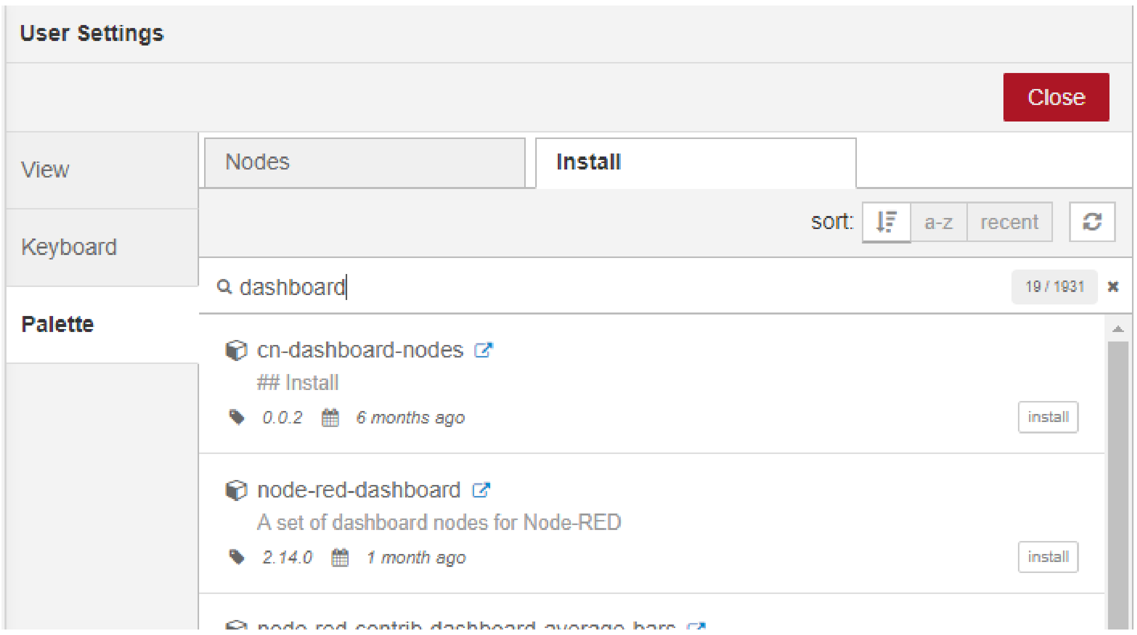


If you add a debug output node and connect it to your mqtt node, you can check if your data is coming through. Deploy and open the debug window on the right. Normally it should work and your data should be coming through.



2. visualization of your data

To make our data visual, we need to import some extra nodes.
Click on the hamburger menu in the top right corner and click on "Manage palette".
Click on the install tab and type "dashboard".
Install the "node-red-dashboard".

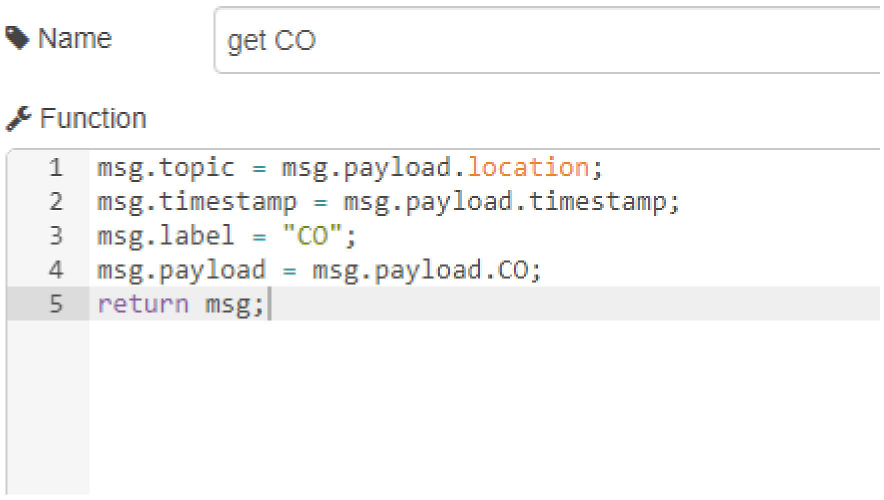


if the installation fails for any reason, just try again.

As an example we will show you how to make one value of our data visible, the other data is similar.

Example chart:

First, click and drag a function node on the grid. Double click it. This node will extract the CO from our data that is coming through.
The code for this is the following:

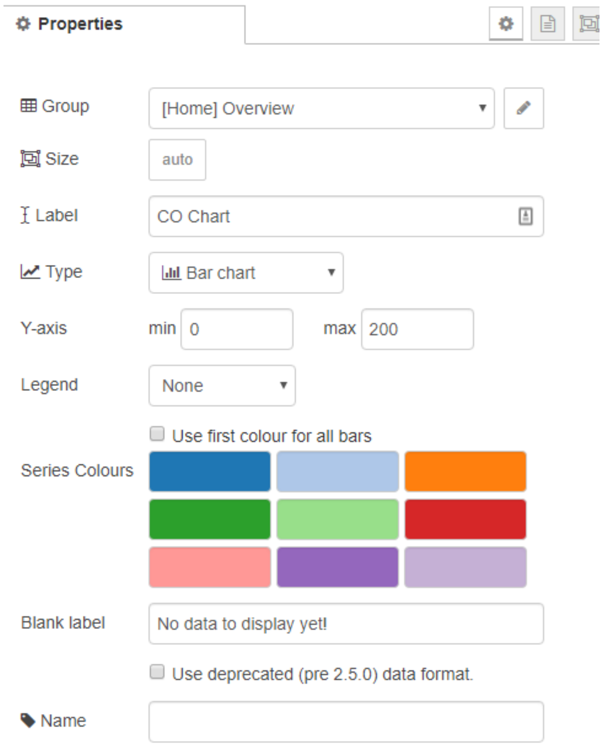


Name: get CO

Function

```
msg.topic = msg.payload.location;
msg.timestamp = msg.payload.timestamp;
msg.label = "CO";
msg.payload = msg.payload.CO;
return msg;
```

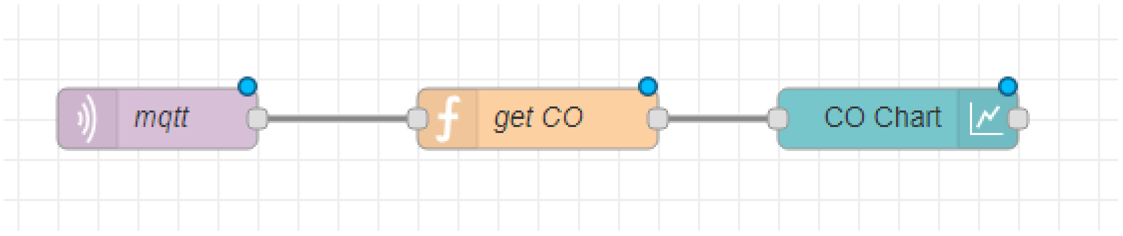
You can name if however you want.
Next, add a chart node and connect it to the function node.
Choose a group name or create a new one.
Choose a label.
Choose your minimum and maximum values to be shown.
You can customize it however you like.



Properties

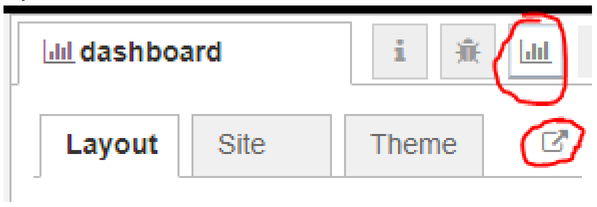
Group	[Home] Overview
Size	auto
Label	CO Chart
Type	Bar chart
Y-axis	min 0 max 200
Legend	None
	☐ Use first colour for all bars
Series Colours	🟦 🟦 🟧 🟩 🟩 🟥 🟥 🟪 🟪
Blank label	No data to display yet!
	☐ Use deprecated (pre 2.5.0) data format.
Name	

Your flow should look like this:

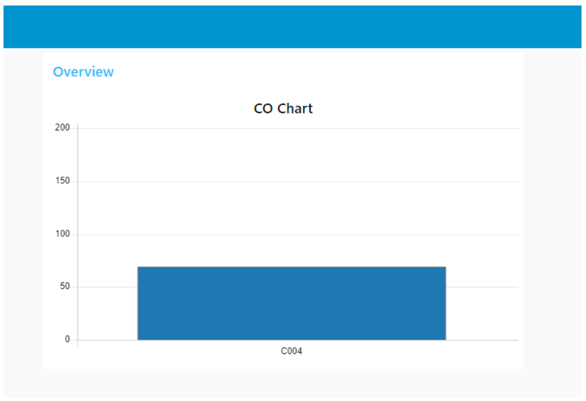


Deploy it.

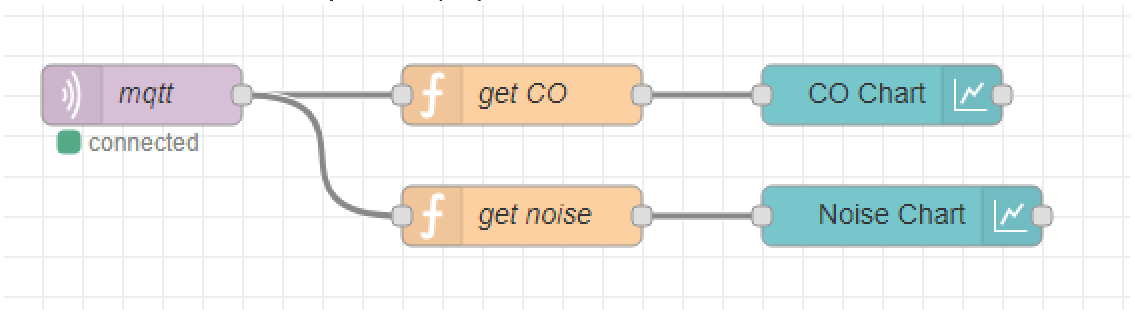
To look at the result, click on the dashboard tab. Clicking the small icon on the right will open up a browser tab.



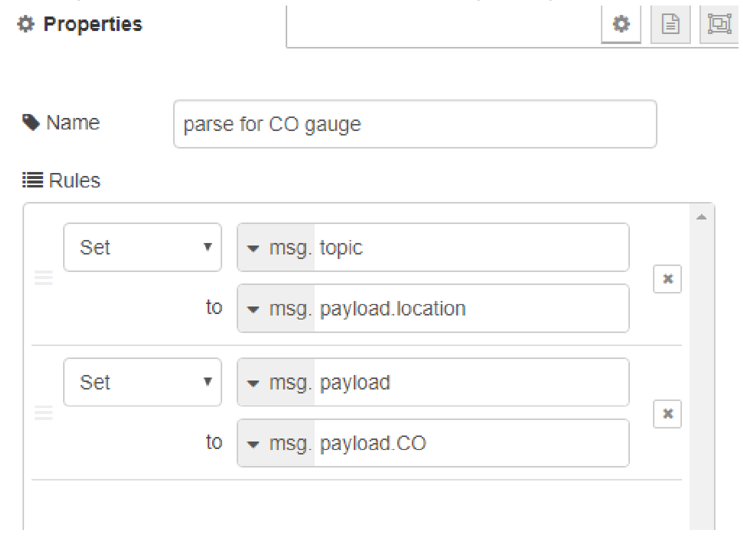
This is the result:



You can do the same steps to display the noise.



Example gauge:
Select and drag a switch node to the grid.
Configure it as shown in the following image:


Properties

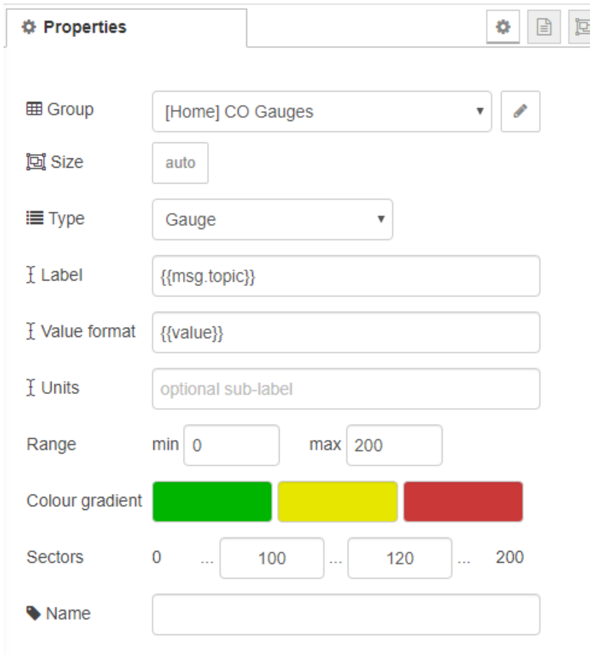
Name: parse for CO gauge

Rules

Set msg. topic
to msg. payload.location

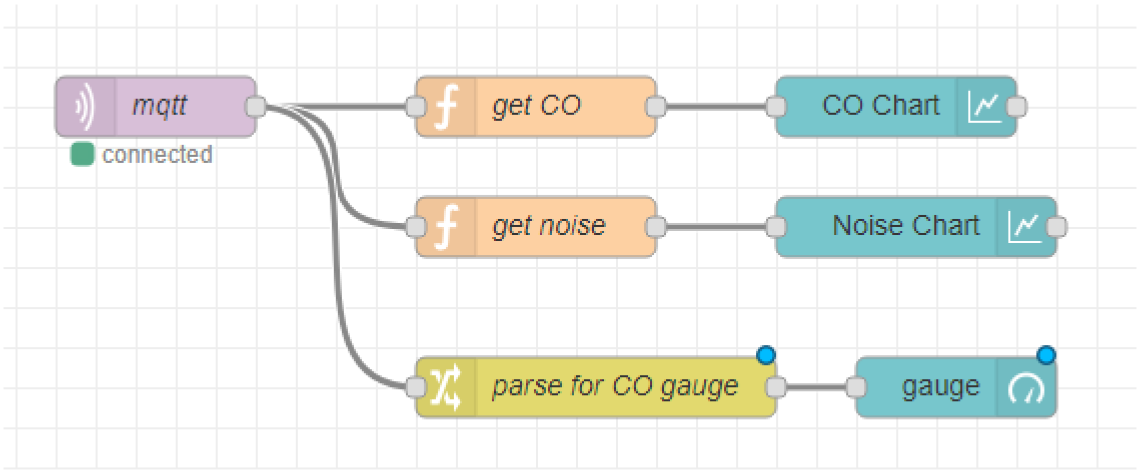
Set msg. payload
to msg. payload.CO

Connect it to your mqtt node.
Select and drag a gauge node to the grid.
Configure it as shown in the following image:

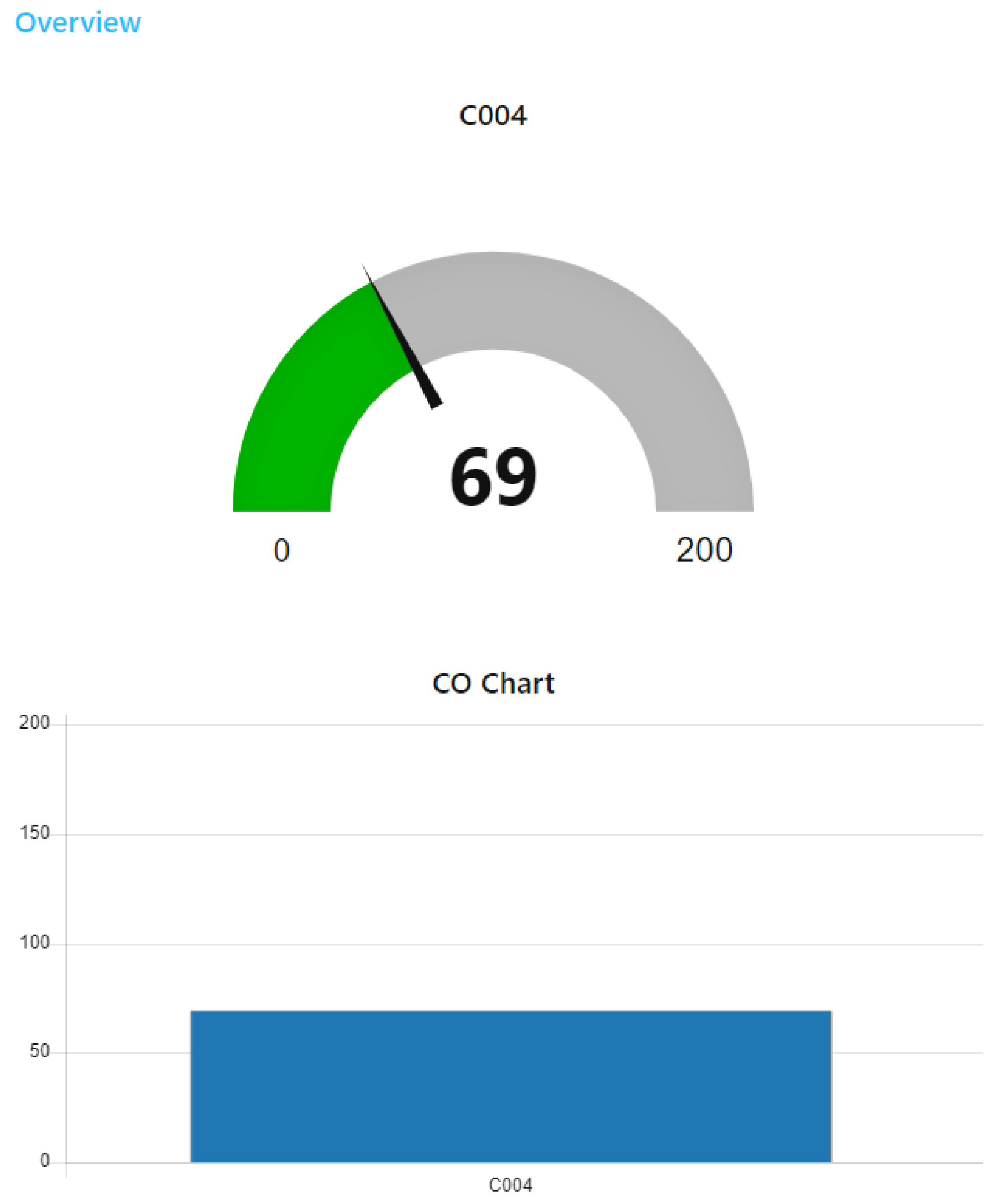

Properties

Group	[Home] CO Gauges
Size	auto
Type	Gauge
Label	{{msg.topic}}
Value format	{{value}}
Units	optional sub-label
Range	min 0 max 200
Colour gradient	🟩 🟨 🟥
Sectors	0 ... 100 ... 120 ... 200
Name	

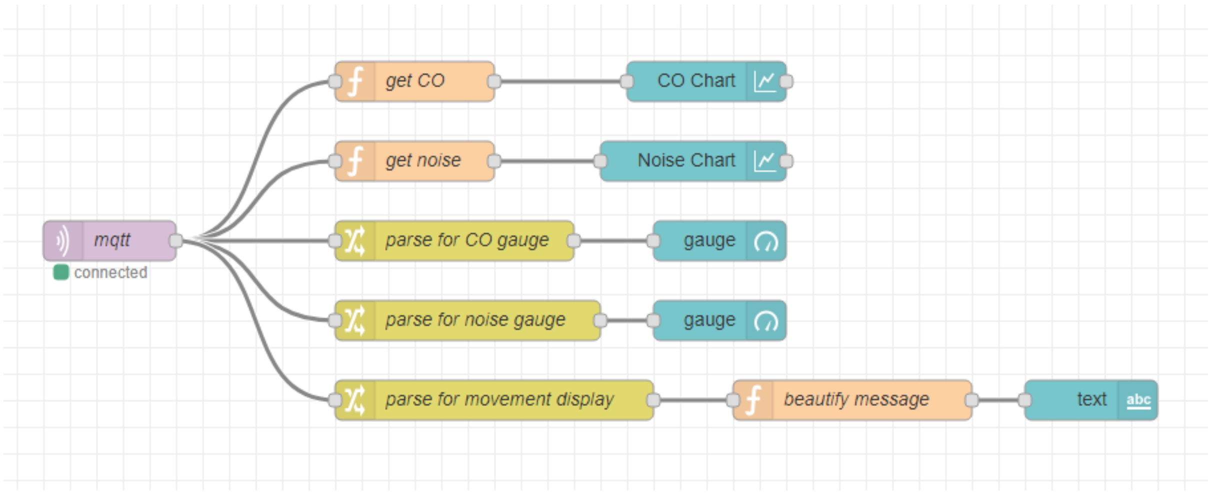
Connect it with the change node to create the following flow:



Deploy. Your dashboard should now look like this:



You can do this for all other variables to become the following result:



Congratulations, your data is now visualized.