

# MQTT AVEC MICROPYTHON ET ESP8266

L'objectif est de mettre en œuvre un environnement MQTT complet que se soit avec un ESP8266 ou avec un IDE Python.

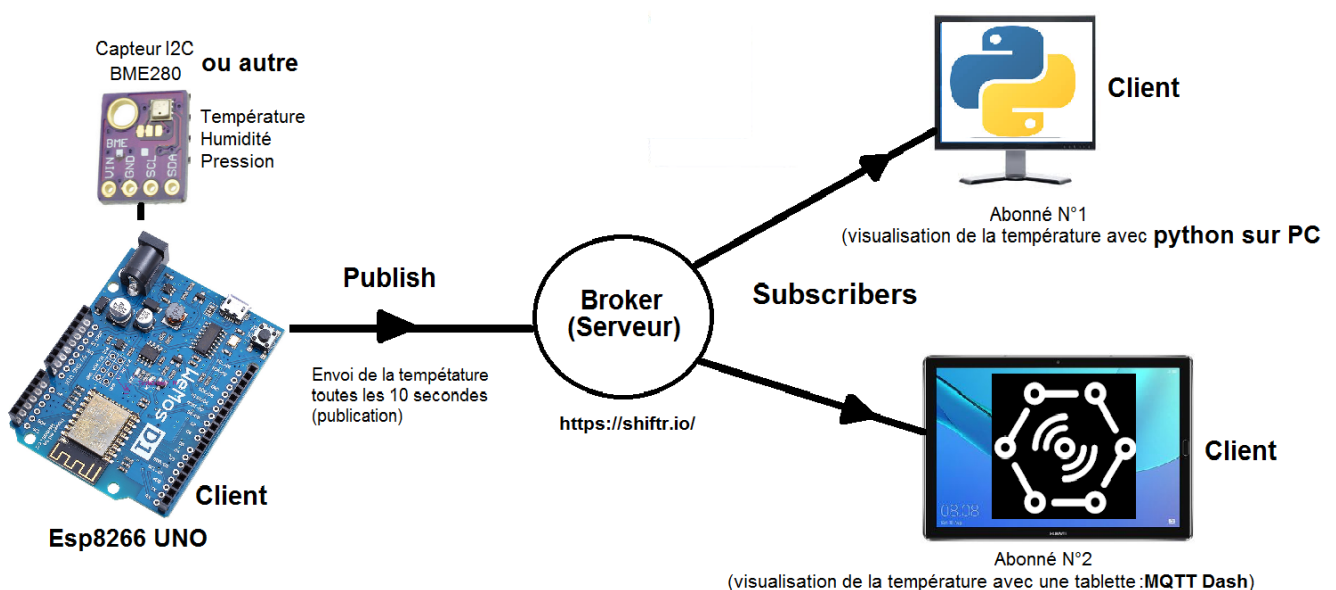
1 Qu'est-ce que MQTT ?	<ul style="list-style-type: none"> <li>Mise en situation</li> </ul>
2 Configuration d'un broker en ligne	<ul style="list-style-type: none"> <li>Inscription et configuration</li> <li>Bilan de la configuration</li> </ul>
3 Utilisation avec un ESP8266	<ul style="list-style-type: none"> <li>Partie matérielle</li> <li>Installation de la librairie</li> <li>Utilisation du programme d'exemple</li> </ul>
4 Utilisation avec un IDE python	<ul style="list-style-type: none"> <li>Installation de la librairie</li> <li>Analyse du programme d'exemple</li> </ul>
5 Utilisation avec une tablette Android	<ul style="list-style-type: none"> <li>Configuration du logiciel MQTT dash et MQTT Panel</li> </ul>
Annexes	<ul style="list-style-type: none"> <li>Annexe 1 : ESP32 et Micropython</li> <li>Annexe 2 : Installer un point d'accès Wifi RaspAP</li> <li>Annexe 3 : Exemples de programmes Micropython</li> <li>Annexe 4 : Utilisation de Node-red</li> <li>Annexe 5 : Analyse du protocole MQTT</li> </ul>

## 1 Qu'est-ce que MQTT ?

MQTT (Message Queuing Telemetry Transport) est une messagerie publish-subscribe basé sur le protocole TCP/IP. MQTT est utilisé pour les IOT (Internet of Things / objets connectés). Il est conçu comme un transport de messagerie de publication / abonnement extrêmement léger en termes de ressources.

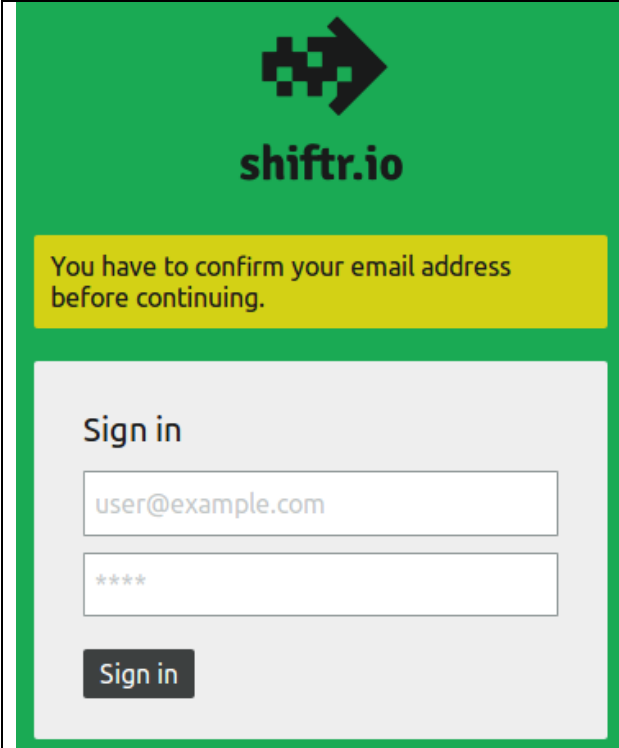
Mise en situation :


Vous avez réalisé un système à base d'ESP8266 permettant de mesurer la température d'une pièce. Vous voulez connaître cette température quand vous êtes à l'extérieur de votre maison. A première vue, une solution serait de concevoir une page WEB afin de pouvoir y accéder depuis un navigateur. MQTT va vous permettre de remplir cet objectif plus rapidement en utilisant une bande passante très réduite. Il est aussi possible de déclencher un mécanisme à distance comme une des volets roulants etc... La communication peut ainsi être bidirectionnelle.



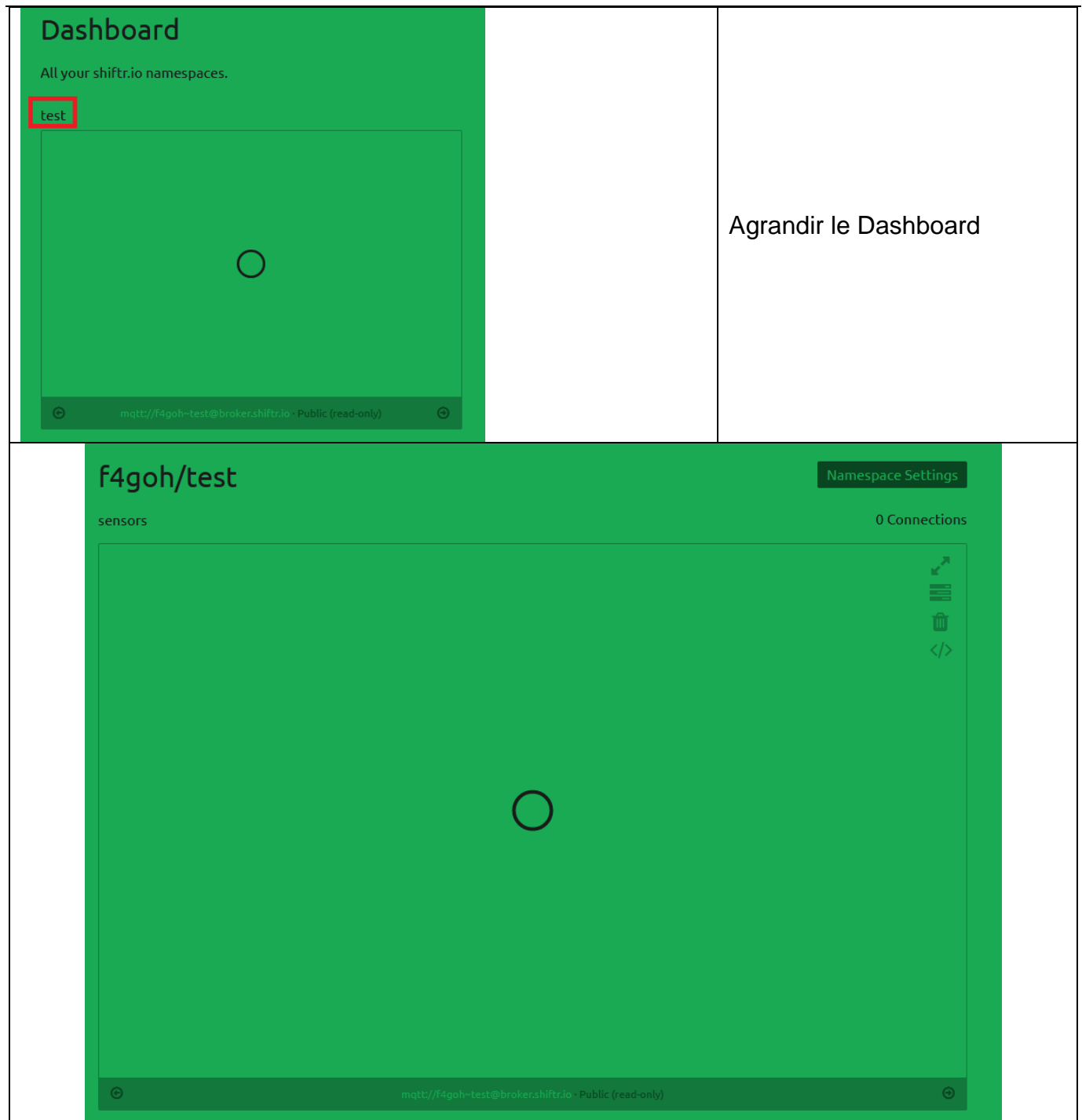
La suite du document va expliquer la configuration du serveur <https://shiftr.io/> puis l'envoi d'une température au serveur à partir d'un ESP8266 en Wifi (publish). Enfin la réception des données avec un IDE Python et une tablette (Subscribe).

## 2 Configuration d'un broker (serveur MQTT) en ligne

	<p>Créer un compte sur le site <a href="https://shiftr.io/">https://shiftr.io/</a></p>
<p>Welcome f4goh!</p> <p>You can confirm your account email through the link below:</p> <p><a href="#">Confirm my account</a></p>	<p>Confirmer le compte à partir de votre adresse mail</p>
	<p>Créer un « namespace » pour chaque projets</p>

<p><b>Name *</b></p> <input type="text" value="test"/> <p>Only characters, numbers and dashes are allowed, recommended is to use a name like 'project-x'.</p> <p><b>Description</b></p> <input type="text" value="sensors"/> <p>Optional, but you can tell other people what you plan to do.</p> <p><input type="checkbox"/> Private</p> <p>Private namespaces are excluded from your public profile and not accessible by other users.</p> <p><b>Create Namespace</b></p>		<p>Exemple :</p> <p>test</p> <p>sensors</p>
<p><b>f4goh/test</b></p> <p>sensors</p> <p>0 Connections</p> <p><b>You haven't created any "full-access" tokens yet</b></p> <p>In order to publish messages to this namespace, you have to create "full-access" tokens on the <a href="#">Namespace Settings</a> page.</p>	<p><a href="#">Namespace Settings</a></p>	<p>Ajouter un token (jeton)</p> <p>Namespace Settings, Add token</p>
<p><b>Add Token</b></p> <p>Create a new token that grants access to your namespace.</p> <p><b>Key (Username)</b></p> <input type="text" value="weatherSensors"/> <p>A key must be globally unique on shiftr.io.</p> <p><b>Secret (Password)</b></p> <input type="text" value="bme280Sensors"/> <p><b>Permission *</b></p> <p>full-access</p> <p>Only full-access tokens grant write access to namespaces.</p> <p><b>Description</b></p> <input type="text" value="bme 280 sensors test"/> <p><b>Create Token</b></p>		<p>Key et Secret 8 caractères minimum</p>
 <p>shiftr.io</p> <p>Try Explore Get Started Documentation</p> <p><b>f4goh/test</b></p> <p>Tokens Webhooks Info Embed Data Delete</p> <p>All tokens associated with your namespace.</p> <p><b>bme 280 sensors test</b></p> <p>mqtt://weatherSensors:bme280Sensors@broker.shiftr.io</p> <p>Full Access Revoke</p> <p><b>Dashboard</b></p> <p>New Namespace</p> <p>Your Profile</p> <p>Account Settings</p> <p>Logout</p>		

Aller sur le Dashboard (Tableau de bord)



Agrandir le Dashboard

La configuration du Broker (Serveur MQTT) est terminée.

Fichiers ICI

<https://github.com/f4goh/NSI>

## Bilan de la configuration

### Arduino ethernet

Adresse du serveur	broker.shiftr.io
Client id	arduino
Key	weatherSensors
Secret	bme280Sensors
Temperature Topic	/sensors/temperature
Pression Topic	/sensors/pression
Humidité Topic	/sensors/humidite

### Ide Python

Adresse du serveur	mqtt://weatherSensors:bme280Sensors@broker.shiftr.io
Client id	python
Key	weatherSensors
Secret	bme280Sensors
Temperature Topic	/sensors/temperature
Pression Topic	/sensors/pression
Humidité Topic	/sensors/humidite



### Tablette Android avec MQTT dash

<https://play.google.com/store/apps/details?id=net.routix.mqttdash>


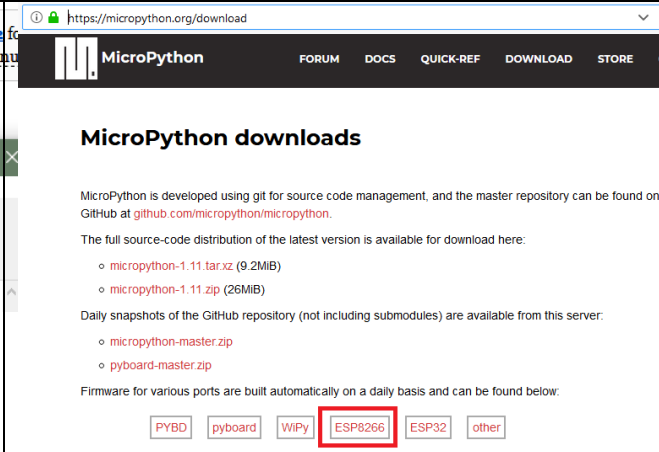
Name	sensorsTest
Adresse du serveur	broker.shiftr.io
Client id	Mqttdash-xxxxx (choisi par le logiciel MQTT dash)
User name (key)	weatherSensors
User password (password)	bme280Sensors
Topic name	sensors
Temperature Topic	/sensors/temperature
Pression Topic	/sensors/pression
Humidité Topic	/sensors/humidite

Port 1881 : communication non sécurisée (données en clair sur le réseau) par défaut dans ce document.

Port 8881 : communication non sécurisé SSL (Secure Socket Layer) / TLS (Transport Layer Security)

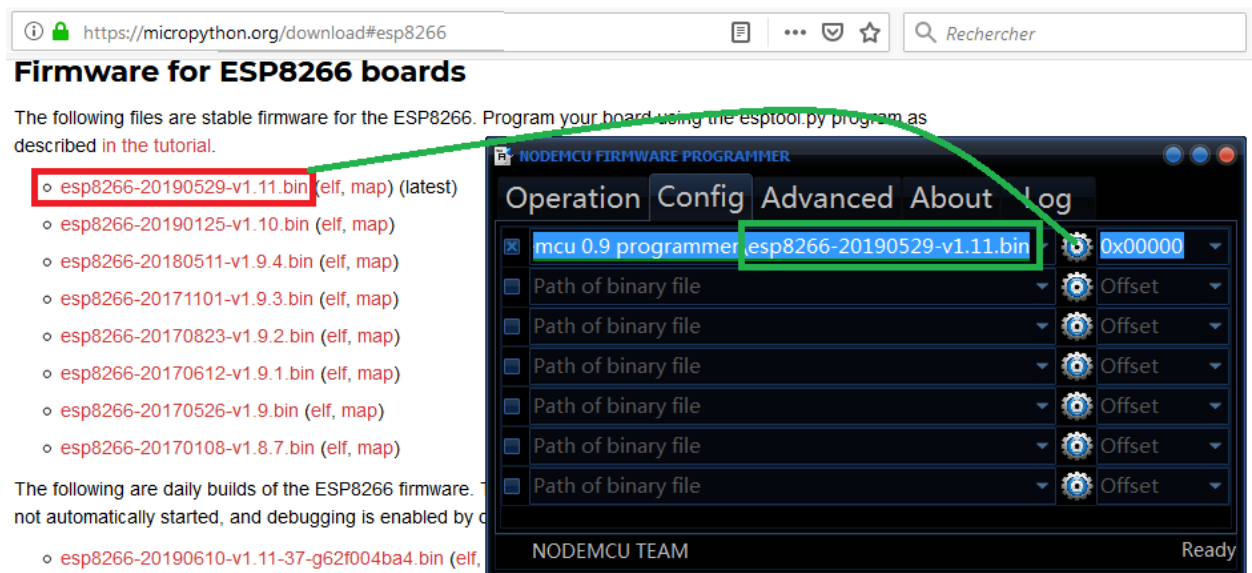
**En rouge : obligatoire en lien avec la configuration du broker**

### 3 Utilisation avec un ESP8266

Editeur	Firmware
<a href="https://thonny.org/">https://thonny.org/</a> (Windows, linux installation via pip) ou <a href="https://codewith.mu/">https://codewith.mu/</a> (Windows, esp8266 non supporté sous linux)	<a href="https://micropython.org/download">https://micropython.org/download</a>
	

Flasher le firmware dans le 8266 (<https://micropython.org/download#esp8266>)

Avec nodemcu firmware programmer disponible ici : <https://github.com/f4goh/NSI>



**Firmware for ESP8266 boards**

The following files are stable firmware for the ESP8266. Program your board using the esptool.py program as described in the tutorial.

- esp8266-20190529-v1.11.bin (elf, map) (latest)
- esp8266-20190125-v1.10.bin (elf, map)
- esp8266-20180511-v1.9.4.bin (elf, map)
- esp8266-20171101-v1.9.3.bin (elf, map)
- esp8266-20170823-v1.9.2.bin (elf, map)
- esp8266-20170612-v1.9.1.bin (elf, map)
- esp8266-20170526-v1.9.bin (elf, map)
- esp8266-20170108-v1.8.7.bin (elf, map)

The following are daily builds of the ESP8266 firmware. They are not automatically started, and debugging is enabled by default.

- esp8266-20190610-v1.11-37-g62f004ba4.bin (elf, map)

**NODEMCU FIRMWARE PROGRAMMER**

Operation Config Advanced About Log

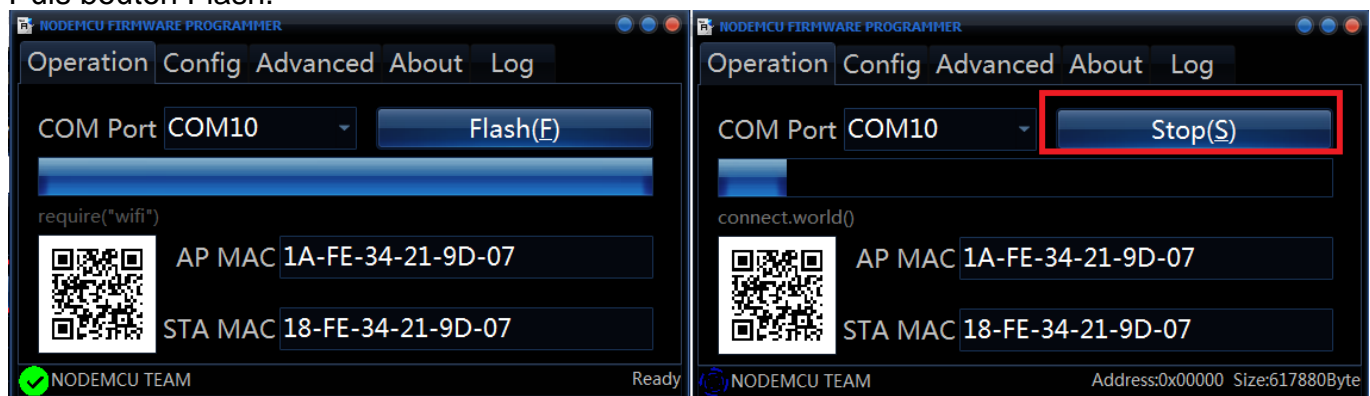
mcu 0.9 programmer esp8266-20190529-v1.11.bin 0x000000

Path of binary file

Offset

NODEMCU TEAM Ready

Puis bouton Flash.



**NODEMCU FIRMWARE PROGRAMMER**

Operation Config Advanced About Log

COM Port COM10 Flash(E)

require("wifi")

AP MAC 1A-FE-34-21-9D-07

STA MAC 18-FE-34-21-9D-07

NODEMCU TEAM Ready

**NODEMCU FIRMWARE PROGRAMMER**

Operation Config Advanced About Log

COM Port COM10 Stop(S)

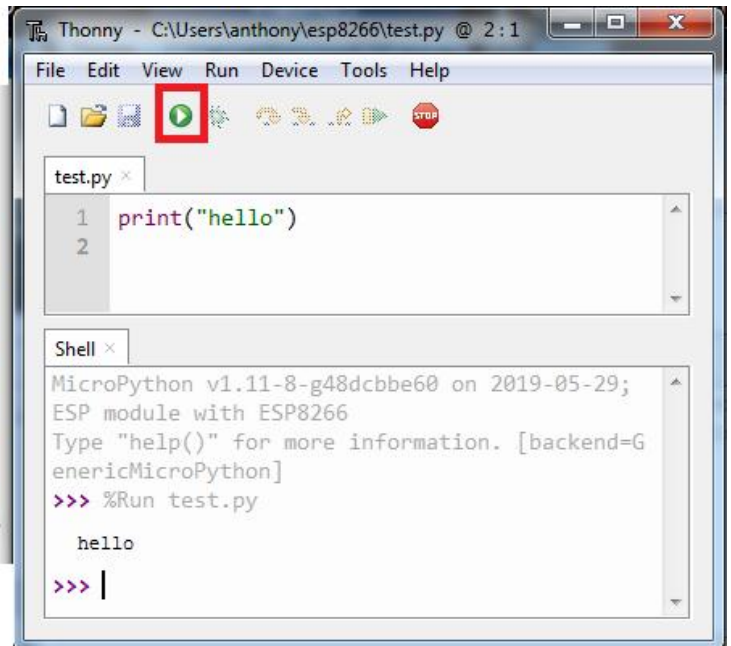
connect.world()

AP MAC 1A-FE-34-21-9D-07

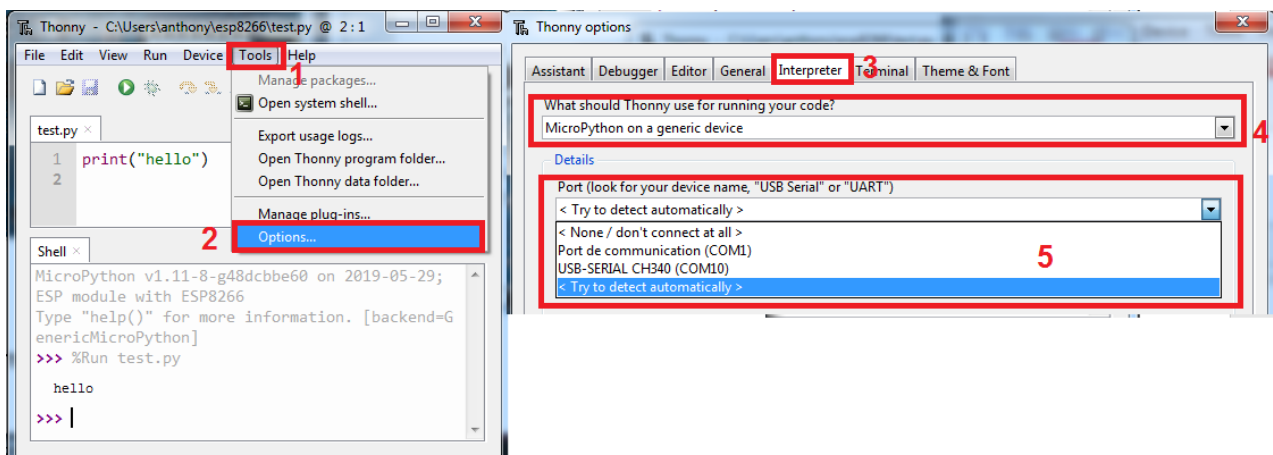
STA MAC 18-FE-34-21-9D-07

NODEMCU TEAM Address:0x000000 Size:617880Byte

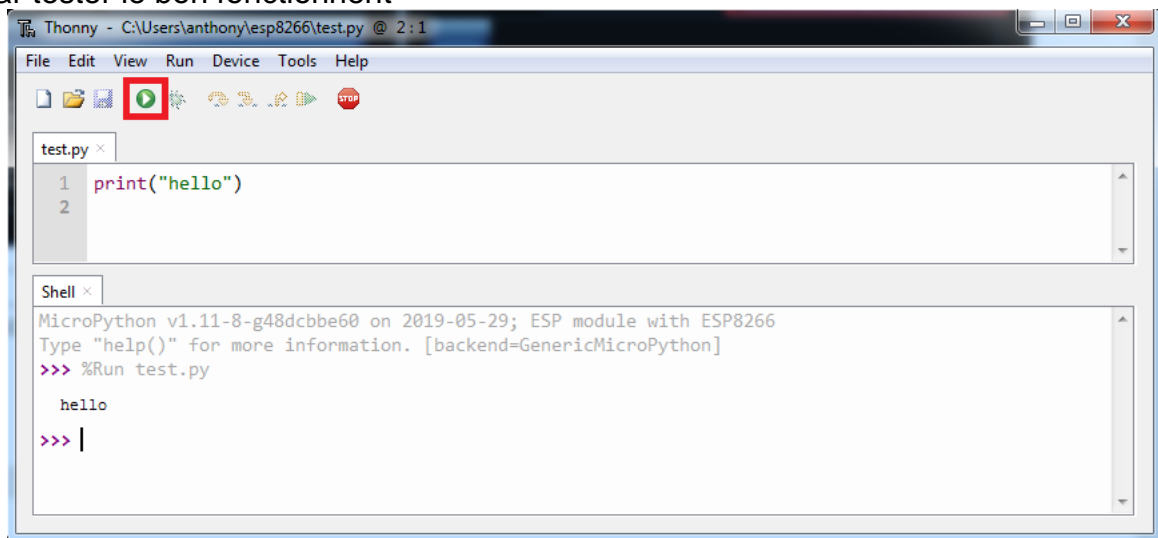
Faire un reset de la carte esp8266 **avant** de lancer Thonny



Configuration en mode esp8266



Finir par tester le bon fonctionnement





<http://docs.micropython.org/en/v1.9.3/esp8266/esp8266/tutorial/index.html>

Quels sont les modules déjà intégrés ?

```

Shell x
MicroPython v1.11-8-g48dcbb60 on 2019-05-29; ESP module with ESP8266
Type "help()" for more information. [backend=GenericMicroPython]
>>> help("modules")

_main_          http_client    socket          upip
_boot           http_client_ssl ssd1306 ■      upip_utarfile
_onewire        http_server    ssl             upysh
_webrepl        http_server_ssl struct          urandom
apa102 ■        inisetup       sys             ure
array           io             time            urequests
binascii        json           uasyncio/___init___ urllib/urrequest
btree          lwip           uasyncio/core  uselect
builtins        machine ■      ubinascii       usocket
collections     math           ucollections    ssl
dht ■          micropython    ucryptolib      ustruct
ds18x20 ■       neopixel ■     uctypes         utime
errno           network        uerrno          utimeq
esp            ntptime        uhashlib        uwebsocket
example_pub_button onewire       uheapq          uzlib
example_sub_led os             uio             webrepl
flashbdev       port_diag      ujson           webrepl_setup
framebuf        random         umqtt/robust ■  websocket_helper
gc              re             umqtt/simple    zlib
hashlib         select         uos
Plus any modules on the filesystem

>>> |

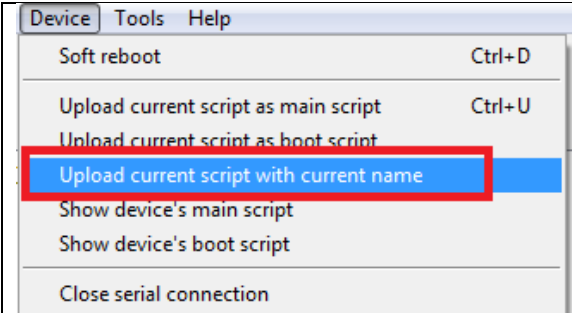
```

En **rouge** le hardware géré en natif. On remarque le protocole mqtt intéressant pour la gestion des IOT.  
En **vert** (machine). C'est le module qui gère les périphériques.

<http://docs.micropython.org/en/v1.9.3/esp8266/library/machine.html>

class Pin – control I/O pins  
class Signal – control and sense external I/O devices  
class UART – duplex serial communication bus  
class SPI – a Serial Peripheral Interface bus protocol (master side)  
class I2C – a two-wire serial protocol  
class RTC – real time clock  
class Timer – control hardware timers  
class WDT – watchdog timer

Le reste à découvrir au fur et à mesure. Il est possible d'ajouter des modules en faisant :

	<p><b>A noter :</b></p> <p>L'esp8266 en mode micropython contient 2 fichiers importants.</p> <p><b>Boot.py</b> : est l'équivalent du setup sur Arduino. (Programme exécuté au démarrage) Ensuite si l'esp contient un fichier <b>main.py</b>, alors celui-ci sera exécuté juste après le boot.py</p>
---	--



## Configuration réseau avec le point d'accès wifi

[http://docs.micropython.org/en/v1.9.3/esp8266/esp8266/tutorial/network\\_basics.html?highlight=sta\\_if%20connect](http://docs.micropython.org/en/v1.9.3/esp8266/esp8266/tutorial/network_basics.html?highlight=sta_if%20connect)

Dans la console :

```
>>> import network
>>> sta_if=network.WLAN(network.STA_IF)
>>> sta_if.active()
False

>>> sta_if.active(True)

#5 ets_task(4020f4d8, 28, 3fff9e30, 10) -> normal pas de connexion définie

>>> sta_if.connect('raspi-webgui', 'touchardNSI')
>>> sta_if.isconnected()
True

>>> sta_if.ifconfig()
('10.3.141.104', '255.255.255.0', '10.3.141.1', '10.3.141.1')
```

### Après un Reset ou une remise sous tension :

```
>>> import network
>>> sta_if=network.WLAN(network.STA_IF)
>>> sta_if.isconnected()
True
```

### Afficher l'adresse MAC :

```
>>> import network
>>> import ubinascii
>>> mac = ubinascii.hexlify(network.WLAN().config('mac'),':').decode()
>>> print(mac)
a0:20:a6:21:9d:fa
```

### Remarque :

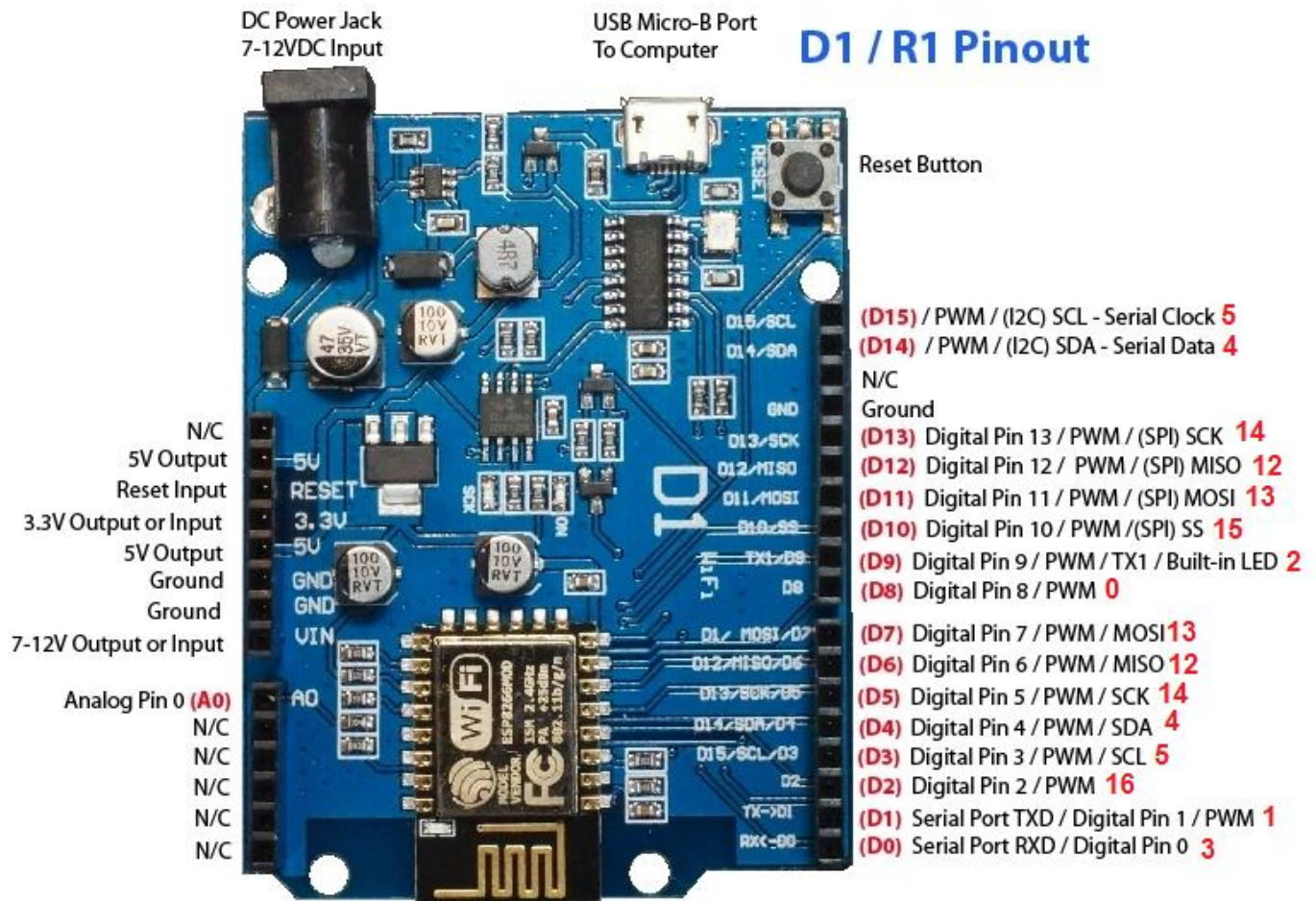
Par défaut REPL est activé, c'est-à-dire que l'on peut commander l'esp8266 avec le port Série via les drivers USB ch340/ch341.

Il existe aussi WebREPL, c'est aussi un moyen de commander l'esp8266 mais à travers le réseau wifi. Celui-ci des désactivé par défaut. Bien sûr, il faut au préalable configurer l'esp8266 sur son point d'accès via un port série. Pour activer WebREPL :

```
>>> import webrepl_setup
WebREPL daemon auto-start status: disabled

Would you like to (E)nable or (D)isable it running on boot?
(Empty line to quit)
> E
To enable WebREPL, you must set password for it
New password (4-9 chars): toto
Confirm password: toto
Changes will be activated after reboot
Would you like to reboot now? (y/n) y #reste à utiliser un client WebREPL
```

## Clignoter une Led



**only pins 0, 2, 4, 5, 12, 13, 14, 15, and 16 can be used.**

<http://docs.micropython.org/en/v1.9.3/esp8266/esp8266/tutorial/pins.html>

```
import machine
import time
led = machine.Pin(2,
machine.Pin.OUT)
while(True):
    led.on()
    time.sleep_ms(500)
    led.off()
    time.sleep_ms(500)
```

#ctrl+c pour stopper

>>> %Run test.py

Traceback (most recent call last):

File "C:\Users\anthony\esp8266\test.py", line 7, in <module>

KeyboardInterrupt:

Thonny - C:\Users\anthony\esp8266\test.py @ 9:23

File Edit View Run Device Tools Help

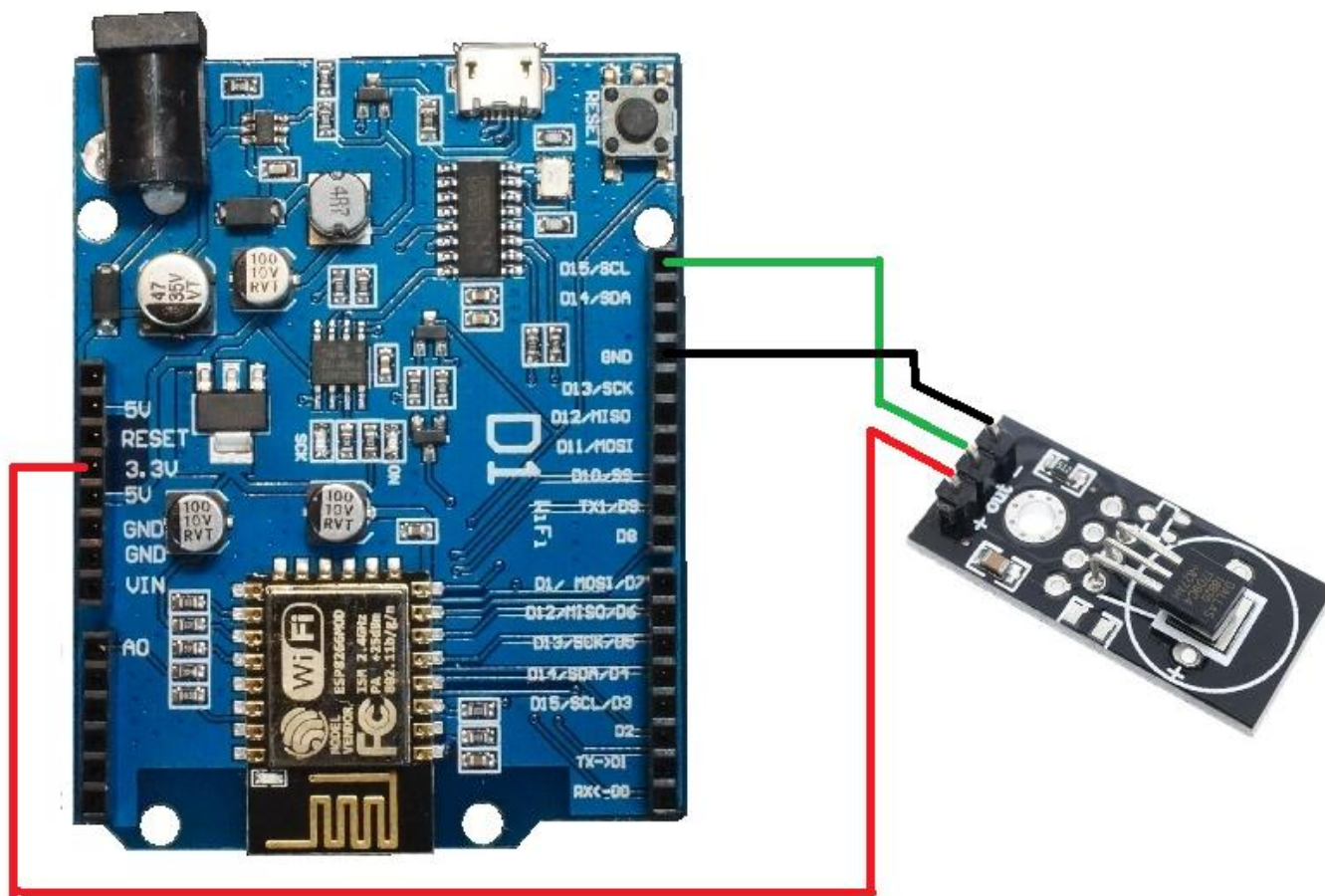
test.py x

```
1
2 import machine
3 import time
4 led = machine.Pin(2, machine.Pin.OUT)
5 while(True):
6     led.on()
7     time.sleep_ms(500)
8     led.off()
9     time.sleep_ms(500)
10
```

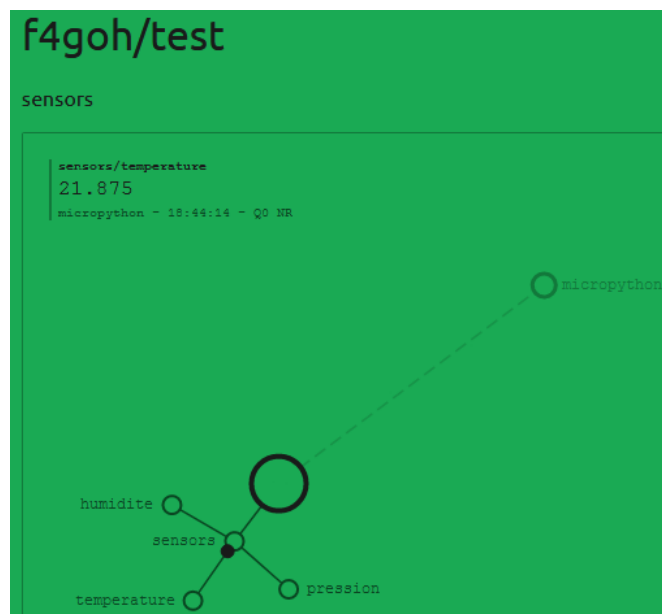
L'auto complétion fonctionne CTRL+SPACE, mais pas à chaque fois

## Envoyer une valeur de température vers un broker MQTT

Schéma de câblage avec un capteur de température ds18b20 onewire



Résultats sur le broker <https://shiftr.io/f4goh/test>



La température est bien reçue sur le broker.

<https://github.com/f4goh/MQTT-Tutoriel>

Le programme est compatible avec le broker <https://thingspeak.com/>

```

from umqtt.robust import MQTTClient
import network
import sys
import time
from time import sleep_ms
from machine import Pin
from onewire import OneWire
from ds18x20 import DS18X20

sta_if = network.WLAN(network.STA_IF)
print(sta_if.active())
print(sta_if.ifconfig())

myMqttClient = b"micropython"

THINGSPEAK_URL = b"broker.shiftr.io"
THINGSPEAK_USER_ID = b'weatherSensors'
THINGSPEAK_MQTT_API_KEY = b'bme280Sensors'
client = MQTTClient(client_id=myMqttClient,
                    server=THINGSPEAK_URL,
                    user=THINGSPEAK_USER_ID,
                    password=THINGSPEAK_MQTT_API_KEY,
                    ssl=False)

try:
    client.connect()
    print("connection ok");
except Exception as e:
    print('could not connect to MQTT server {}'.format(type(e).__name__, e))
    sys.exit()

bus = OneWire( Pin(5) )
ds = DS18X20( bus )

# Scanner tous les périphériques sur le bus
# Chaque périphérique à une adresse spécifique
roms = ds.scan()
for rom in roms:
    print( rom )

PUBLISH_PERIOD_IN_SEC = 30

while True:
    try:
        ds.convert_temp()
        # attendre OBLIGATOIREMENT 750ms
        sleep_ms( 750 )
        for rom in roms:
            temp_celsius = ds.read_temp(rom)
            print( "Temp: %s" % temp_celsius )
            client.publish("/sensors/temperature", str(temp_celsius))
            print("publish ok");
            time.sleep(PUBLISH_PERIOD_IN_SEC)
        except KeyboardInterrupt:
            print('Ctrl-C pressed...exiting')
            client.disconnect()
            sys.exit()

print("exit")

```

```

True
('10.3.141.104', '255.255.255.0', '10.3.141.1', '10.3.141.1')
connection ok
bytearray(b'\x10\xd0\x13U\x03\x08\x00\xc6')
Temp: 22.25
publish ok
Ctrl-C pressed...exiting

```

## ESP8266 en Subscriber

```

from umqtt.robust import MQTTClient
import network
import sys
import time
from time import sleep_ms
from machine import Pin
from onewire import OneWire
from ds18x20 import DS18X20

def cb(topic, msg):
    print((topic, msg))
    freeHeap = float(str(msg, 'utf-8'))
    print("free heap size = {} bytes".format(freeHeap))

sta_if = network.WLAN(network.STA_IF)
print(sta_if.active())
print(sta_if.ifconfig())

myMqttClient = b"micropython"

THINGSPEAK_URL = b"broker.shiftr.io"
THINGSPEAK_USER_ID = b'weatherSensors'
THINGSPEAK_MQTT_API_KEY = b'bme280Sensors'
client = MQTTClient(client_id=myMqttClient,
                    server=THINGSPEAK_URL,
                    user=THINGSPEAK_USER_ID,
                    password=THINGSPEAK_MQTT_API_KEY,
                    ssl=False)
client.set_callback(cb)

try:
    client.connect()
    print("connection ok");
    client.subscribe("/sensors/temperature")
except Exception as e:
    print('could not connect to MQTT server {}'.format(type(e).__name__, e))
    sys.exit()

bus = OneWire(Pin(12))
ds = DS18X20(bus)

# Scanner tous les périphériques sur le bus
# Chaque périphérique à une adresse spécifique
roms = ds.scan()
for rom in roms:
    print( rom )

PUBLISH_PERIOD_IN_SEC = 30

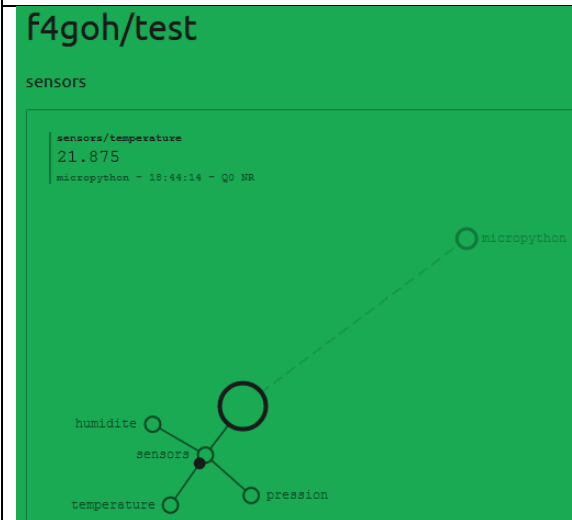
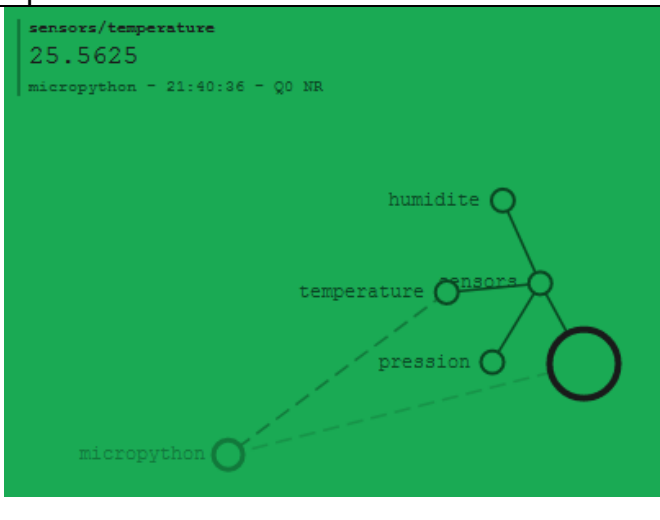
while True:
    try:
        ds.convert_temp()
        # attendre OBLIGATOIREMENT 750ms
        sleep_ms( 750 )
        for rom in roms:
            temp_celsius = ds.read_temp(rom)
            print( "Temp: %s" % temp_celsius )
        client.publish("/sensors/temperature", str(temp_celsius))
        print("publish ok");
        client.wait_msg()
        print("wait next publish");
        time.sleep(PUBLISH_PERIOD_IN_SEC)
    except KeyboardInterrupt:
        print('Ctrl-C pressed...exiting')
        client.disconnect()
        sys.exit()
print("exit")

```

True  
('192.168.1.104', '255.255.255.0', '192.168.1.1',  
'192.168.1.1')  
connection ok  
bytearray(b'\x10\xd0\x13U\x03\x08\x00\xc6')  
Temp: 25.5

publish ok  
(b'/sensors/temperature', b'25.5') #retour d'information  
free heap size = 25.5 bytes



Avant Subscribe	Après Subscribe
	

Lien en pointillés du retour de l'information température

Il est possible d'ajouter d'autres « subscribes »

Les sources sont ici dessous et compatibles avec le broker <https://thingspeak.com/>

<https://github.com/miketeachman/micropython-thingspeak-mqtt-esp8266>

## 4 Utilisation avec un IDE Python

```
import paho.mqtt.client as mqtt

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

client = mqtt.Client("python") #id must be unique
client.username_pw_set("weatherSensors", "bme280Sensors") #login, password
client.on_message = on_message

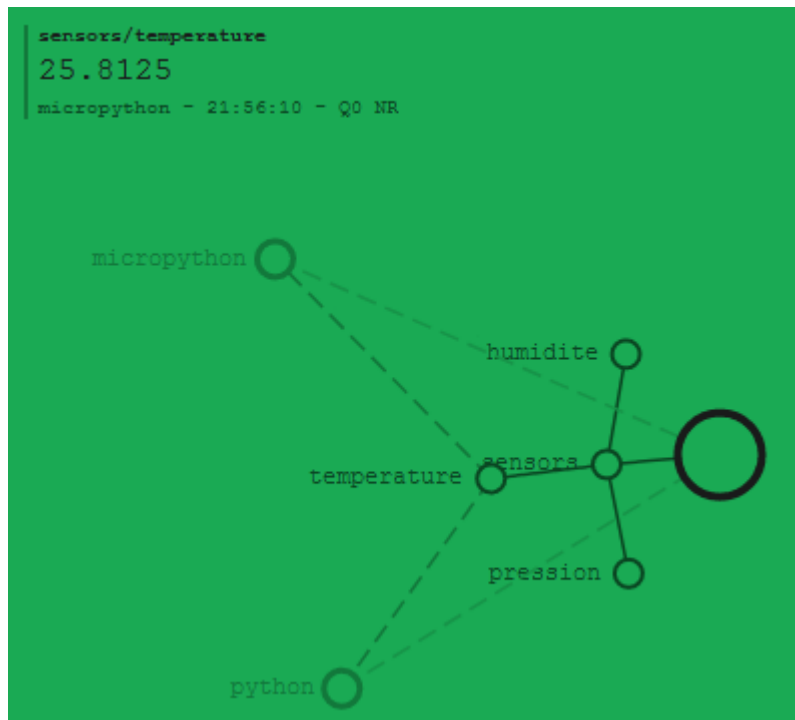
client.connect("broker.shiftr.io", 1883, 60)

client.subscribe("/sensors/temperature")

while True:
    client.loop()
```

```
/sensors/temperature b'25.6875'
/sensors/temperature b'25.6875'
/sensors/temperature b'25.6875'
```

On observe bien le client python en mode Subscriber (traits en pointillés)



Pour publier une donnée (changer la valeur de la température), ajouter utiliser la ligne suivante :

```
client.publish("/sensors/temperature", 30)
```

La température sera dorénavant de 30 degrés



## 5 Utilisation avec une tablette Android

Installer le programme MQTT dash



### MQTT Dash (IoT, Smart Home)

Routix software Communication

★★★★★ 1 911

3 PEGI 3

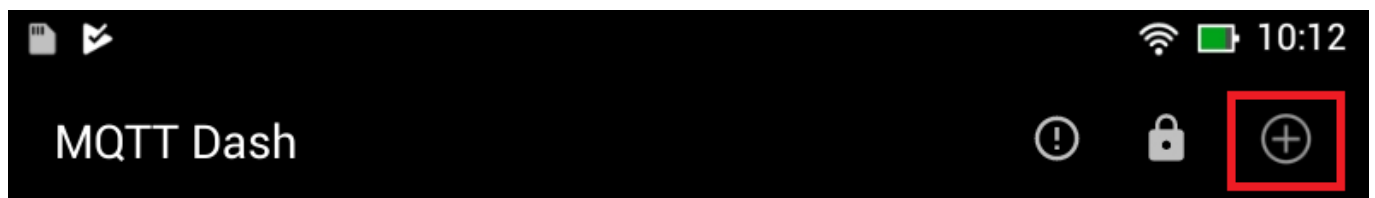
Cette application est compatible avec vos appareils.

Ajouter à la liste de souhaits

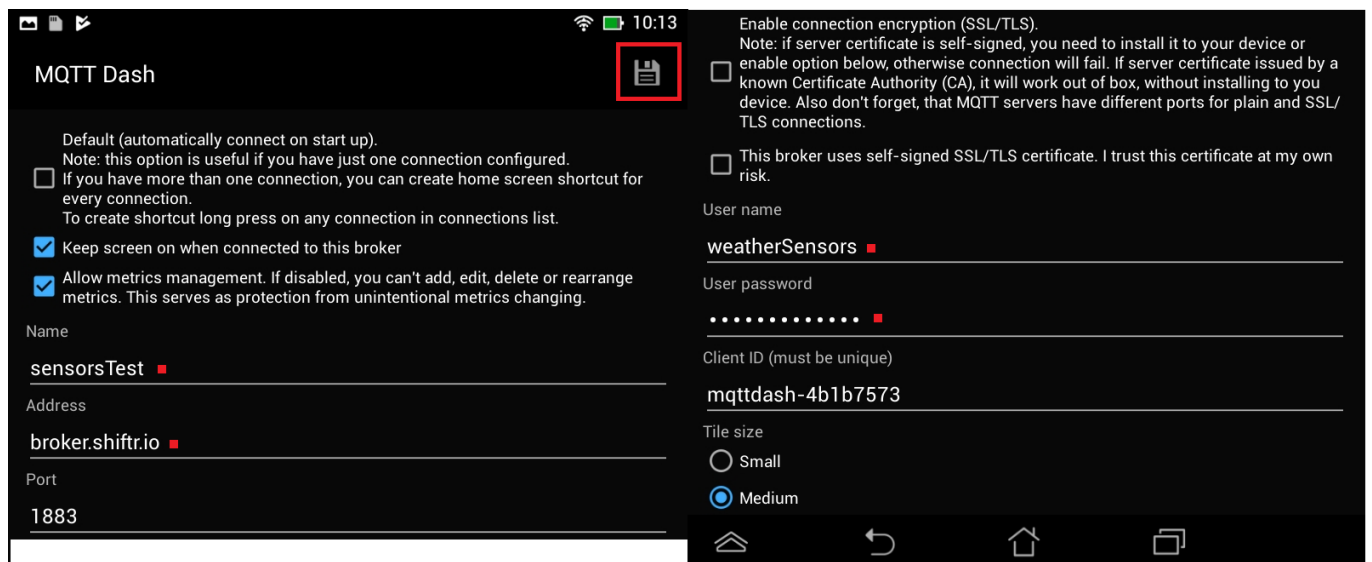
Installer

<https://play.google.com/store/apps/details?id=net.routix.mqttdash>

Configuration :



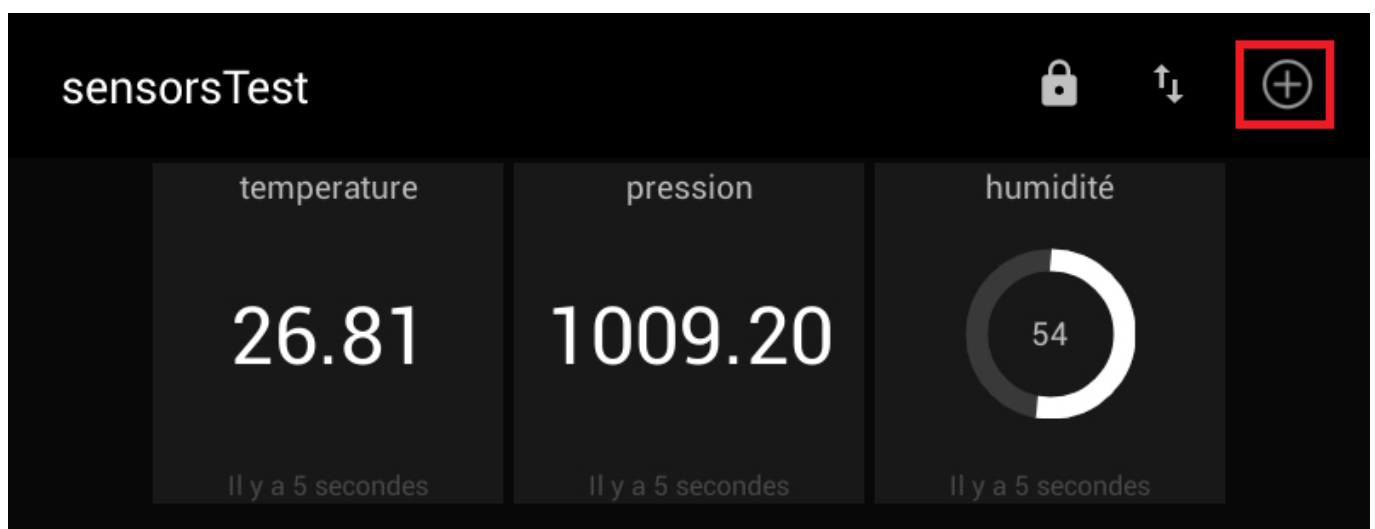
Ajouter la connexion au broker



Cliquer sur sensorsTest



Puis ajouter les 3 topics




La mise a jour des informations aura lieu toutes les 10 secondes

Exemple pour le Topic température

Ne pas oublier de décocher « Enable publishing »

# MQTT Dash



This metric is intended for displaying payload text (e.g. temperature displaying). Payload is expected to be string.

Name

**sensors**

Topic (sub)

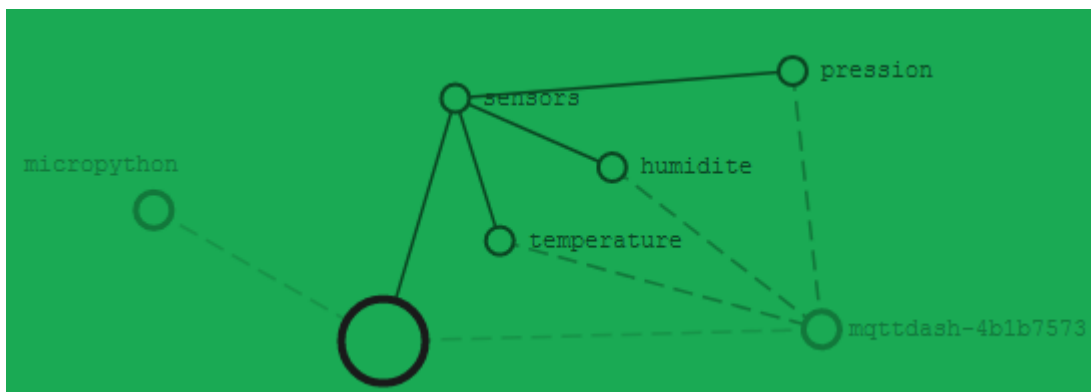
**/sensors/temperature**

Extract from JSON path (if payload is in JSON format), e.g.: \$.level.value. JSON path documentation at the URL below:  
<https://github.com/jayway/JsonPath/blob/master/README.md>

☐ Enable publishing

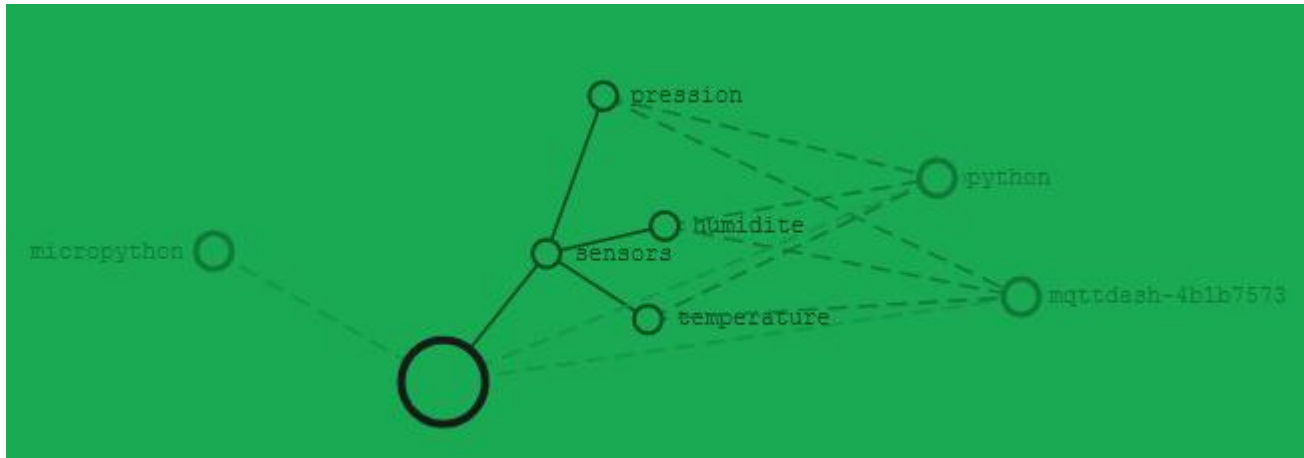
Prefix                      Postfix

Le Dashboard a changé de forme à nouveau



Mqtttdash-4b1b7573 est le nom de la tablette (client ID)

Le Dashboard complet avec un publisher (Micropython esp8266) et deux Subscriber (python sur pc et une tablette)



Informations complémentaires :

La qualité de service (QoS) ou quality of service (QoS) est la capacité à véhiculer dans de bonnes conditions un type de trafic donné.

<p>Other settings</p> <p><input checked="" type="radio"/> QoS(0)</p> <p><input type="radio"/> QoS(1)</p> <p><input type="radio"/> QoS(2)</p>	<ul style="list-style-type: none"> <li>- QoS0. Le message envoyé n'est pas stocké par le Broker. Il n'y a pas d'accusé de réception. Le message sera perdu en cas d'arrêt du serveur ou du client. C'est le mode par défaut</li> <li>- QoS1. Le message sera livré au moins une fois. Le client renvoie le message jusqu'à ce que le broker envoie en retour un accusé de réception.</li> <li>- QoS2. Le broker sauvegarde le message et le transmettra jusqu'à ce qu'il ait été réceptionné par tous les souscripteurs connectés</li> </ul>
--	--

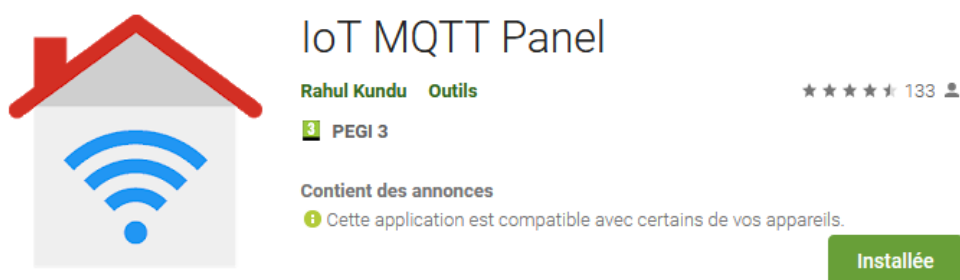
## Conclusion

Le protocole MQTT est très bien adapté aux IOT et la mise en œuvre autour de l'ESP8266 est très simple à réaliser grâce aux bibliothèques prêtes à l'emploi.

Il est possible de réaliser ce tutoriel avec d'autres capteur par exemple sur l'entrée analogique sur A0.

## 5bis Utilisation avec une tablette Android MQTT Panel

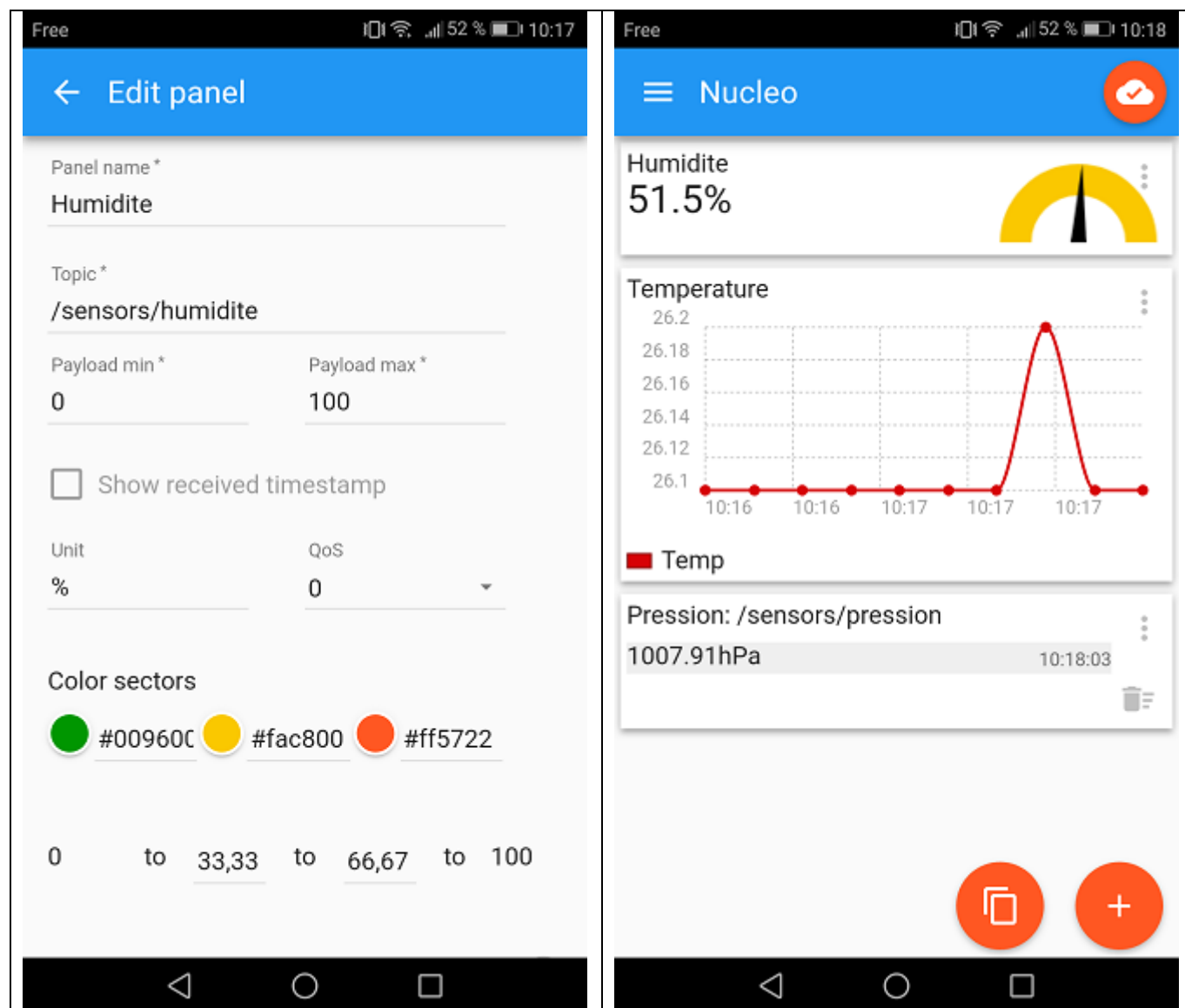
Installer le programme IoT MQTT Panel



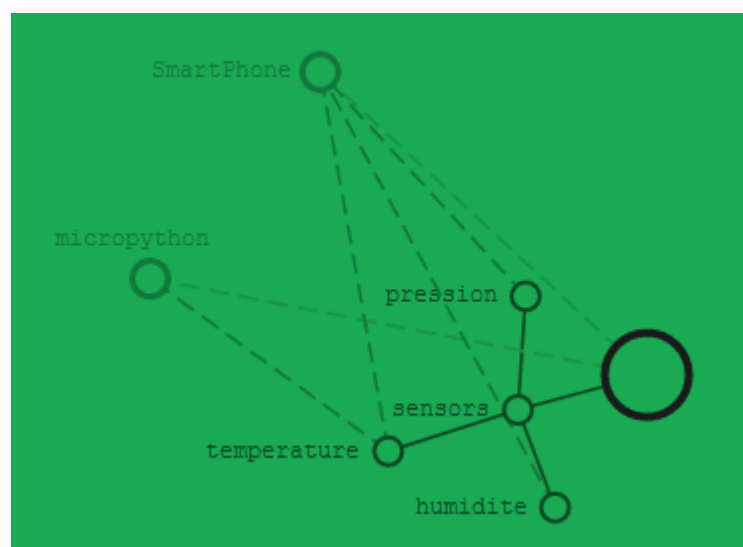
<https://play.google.com/store/apps/details?id=snr.lab.iotmqttpanel.prod>

Ce programme un plus convivial que MQTT dash et possède des widgets. (Élément de base de l'interface graphique d'un logiciel : fenêtre, barre d'outils, par exemple).

Configuration de la connexion au Broker	Ajout des panels
Exemple pour l'humidité	Rendu final



Le Dashboard complet avec un Publisher (micropython esp8266) et un Subscriber (SmartPhone)



## Annexe 1

## ESP32 et Micropython

Pour programmer le firmware de l'esp32, il faudra installer esptool sous linux en ligne de commande

<https://github.com/espressif/esptool>

```
sudo apt install python-pip
sudo pip install --upgrade pip
sudo pip install esptool
pip install pyserial
sudo pip install pyserial
```

Repérer le port série de la carte esp32

```
nsi@nsi-LIFEBLOCK-A555:~$ ls /dev/tty*
/dev/tty17 /dev/tty32 /dev/tty48 /dev/tty63 /dev/ttyS2 /dev/ttyS7
/dev/tty18 /dev/tty33 /dev/tty49 /dev/tty7 /dev/ttyS20 /dev/ttyS8
/dev/tty19 /dev/tty34 /dev/tty5 /dev/tty8 /dev/ttyS21 /dev/ttyS9
/dev/tty2 /dev/tty35 /dev/tty50 /dev/tty9 /dev/ttyS22 /dev/ttyUSB0
/dev/tty20 /dev/tty36 /dev/tty51 /dev/ttyprintk /dev/ttyS23
/dev/tty21 /dev/tty37 /dev/tty52 /dev/ttyS0 /dev/ttyS24
/dev/tty22 /dev/tty38
/dev/tty53 /dev/ttyS1 /dev/ttyS25
```

Télécharger le firmware pour l'esp32 (<https://micropython.org/download#esp32>)

Standard firmware:

- **esp32-20190610-v1.11-37-g62f004ba4.bin (latest)**
- esp32-20190529-v1.11.bin
- esp32-20190125-v1.10.bin
- esp32-20180511-v1.9.4.bin
- esp32-bluetooth.bin



Renommer le fichier par esp32.bin

Exécuter les 2 commandes suivantes :

```
python esptool.py --port /dev/ttyUSB0 erase_flash
python esptool.py --chip esp32 --port /dev/ttyUSB0 write_flash -z 0x1000 esp32.bin
```

Commandes afin de changer l'adresse MAC de l'ESP32 si nécessaire

```
python espfuse.py --port /dev/ttyUSB0 summary
python espfuse.py --port /dev/ttyUSB0 dump
python espfuse.py --port /dev/ttyUSB0 mac
python espfuse.py --port /dev/ttyUSB0 get_custom_mac
python espfuse.py --port /dev/ttyUSB0 burn_custom_mac de:ad:be:ef:fe:00
python espfuse.py --port /dev/ttyUSB0 get_custom_mac
```



## Annexe 2 : Installer un point d'accès Wifi RaspAP

<https://raspbrian-france.fr/creer-un-hotspot-wi-fi-en-moins-de-10-minutes-avec-la-raspberry-pi/>

```
sudo cp /etc/wpa_supplicant/wpa_supplicant.conf /etc/wpa_supplicant/wpa_supplicant.conf.sav
sudo cp /dev/null /etc/wpa_supplicant/wpa_supplicant.conf
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

ajouter dans le fichier

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
```

Ctrl+o, ctrl+x : sauver,quitter

```
wget -q https://git.io/voEUQ -O /tmp/raspap && bash /tmp/raspap
sudo reboot
```

IP address: 10.3.141.1

Username: **admin**


Password: **secret**

DHCP range: 10.3.141.50 to 10.3.141.255

SSID: rasp-webgui

Password: a definir dans le portail ci dessous

RaspAP Wifi Portail v1.5



- Tableau de bord
- Configurer client WiFi
- Configurer hotspot
- Configurer réseau
- Configurer server DHCP
- Configurer Auth
- Change Thème
- Data usage
- Système
- About RaspAP

Tableau de bord

Interface is up.

Informations d'interface

Nom de l'interface	wlan0
IPv4 Address	10.3.141.1
Masque de sous-réseau	255.255.255.0
IPv6 Address	
Array	
Adresse Mac	b8:27:eb:07:2a:81

Informations sans fil

Connecté à	Not connected
AP Mac Adresse	
Bitrate	
Niveau du signal	
Puissance de transmission	31.00 dBm
La fréquence	
MHz	
Qualité de lien	

Statistiques d'interface

Paquets reçus	1597
Octets reçus	222792 (217,57 KB)
Paquets transférés	2140
Octets transférés	1903992 (1,82 MB)

Appareils connectés

Nom d'hôte	Adresse IP	Adresse Mac
Honor_6X	10.3.141.237	44:c3:46:b8:d2

<https://www.raspberrypi.org/documentation/configuration/use-a-proxy.md>

### Configuration de raspbian via le terminal

```
sudo nano /etc/environment
```

```
export http_proxy="http://username:password@proxyipaddress:proxyport"
export https_proxy="http://username:password@proxyipaddress:proxyport"
export no_proxy="localhost, 127.0.0.1"
```

Pour que vos commandes via sudo gardent ces paramètres,

```
sudo visudo
```

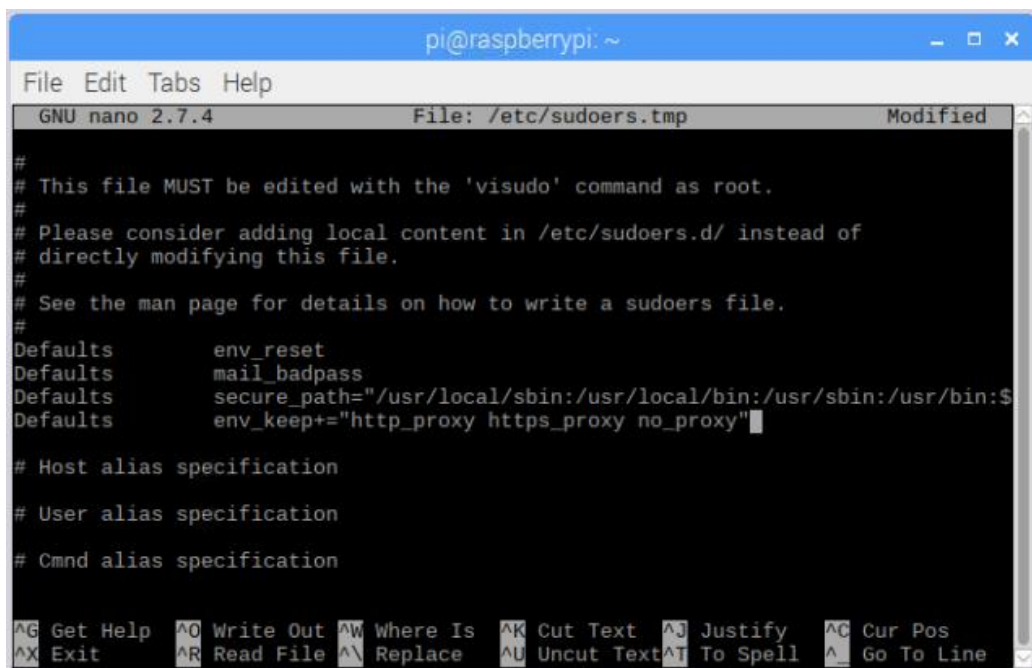
```
Defaults    env_keep+="http_proxy https_proxy no_proxy"
```

Pour que le système de package APT utilise le proxy, créer un fichier 10proxy dans /etc/apt/apt.conf.d/ :

```
cd /etc/apt/apt.conf.d
sudo nano 10proxy
```

```
Acquire::http::proxy "http://username:password@proxyipaddress:proxyport";
```

```
Acquire::https::proxy "http://username:password@proxyipaddress:proxyport";
```



## Annexe 3 : Exemples de programmes Micropython

### Timers en micropython

La précision du signal généré sur la broche 13 n'a rien de comparable avec le même programme en langage C.

```
"""
timer irq
the ESP32 has 4 hardware timers. For this example, we will use timer 0.
"""

import machine

led = machine.Pin(16, machine.Pin.OUT)
timer = machine.Timer(0)

def handleInterrupt(timer):
    global ledState
    ledState ^= 1
    led.value(ledState)

ledState = 0

timer.init(freq=1000, mode=machine.Timer.PERIODIC, callback=handleInterrupt)
# timer.init(period=1000, mode=machine.Timer.PERIODIC, callback=handleInterrupt)

while True:
    pass
```

### Scanner I2C

```
"""
scan i2c esp32
Scan i2c bus...
i2c devices found: 1
Decimal address: 60 | Hexa address: 0x3c
"""

import machine
i2c = machine.I2C(scl=machine.Pin(4), sda=machine.Pin(5))

print('Scan i2c bus...')
devices = i2c.scan()

if len(devices) == 0:
    print("No i2c device !")
else:
    print('i2c devices found:', len(devices))

for device in devices:
    print("Decimal address: ", device, " | Hexa address: ", hex(device))
```

### Afficheur OLED SSD1306

```
"""
oled test
"""

import machine, ssd1306

i2c = machine.I2C(scl=machine.Pin(4), sda=machine.Pin(5))
oled = ssd1306.SSD1306_I2C(128, 64, i2c, 0x3c)
oled.fill(0)
oled.text("Hello f4goh", 0, 0)
oled.show()
```

```

"""
esp32 pinout
gpio16 rx
gpio17 tx
"""

import uasyncio as asyncio
from machine import UART
uart = UART(2, 9600)

async def sender():
    swriter = asyncio.StreamWriter(uart, {})
    while True:
        await swriter.awrite('Hello uart. \n')
        print('Wrote')
        await asyncio.sleep(2)

async def receiver():
    sreader = asyncio.StreamReader(uart)
    while True:
        res = await sreader.readline()
        print('Recieved', res)

loop = asyncio.get_event_loop()
loop.create_task(sender())
loop.create_task(receiver())
loop.run_forever()

```

#### Réception GPS sur l'UART 2 avec asyncio

A tester : [https://github.com/alexmrqt/micropython-gps/blob/master/adafruit\\_gps.py](https://github.com/alexmrqt/micropython-gps/blob/master/adafruit_gps.py)

```

"""
esp32 pinout
gpio16 rx relié à la sortie GPS
gpio17 tx
"""

import uasyncio as asyncio
from machine import UART
uart = UART(2, 9600)

async def receiver():
    sreader = asyncio.StreamReader(uart)
    while True:
        res = await sreader.readline()
        nmea=res.split(b',')
        if nmea[0]==b'$GPGGA':
            #print('Recieved', nmea)
            print(int(float(nmea[1].decode())) ,end=',')
            print('latitude :',float(nmea[2].decode()) ,end=',')
            print(' longitude :',float(nmea[4].decode()))

loop = asyncio.get_event_loop()
loop.create_task(receiver())
loop.run_forever()

```

```

133405,latitude : 4753.415, longitude : 16.6092
133406,latitude : 4753.415, longitude : 16.6092
133407,latitude : 4753.415, longitude : 16.6092

```

## Clignoter deux Leds

En utilisant la programmation asynchrone avec librairie uasyncio

<https://github.com/peterhinch/micropython-async/blob/master/TUTORIAL.md>

Attention la dernière version d'asyncio n'est pas dans le firmware de micropython. Surveillez les mises à jour.

<pre>import machine import uasyncio as asyncio  led1 = machine.Pin(2, machine.Pin.OUT) led2 = machine.Pin(15, machine.Pin.OUT)  async def blink(led, delay): # coroutine     while True:         print(led, "on")         led.on()         await asyncio.sleep_ms(delay)         print(led, "off")         led.off()         await asyncio.sleep_ms(delay)  # boucle d'événements loop = asyncio.get_event_loop() loop.create_task(blink(led1, 500)) # Schedule ASAP loop.create_task(blink(led2, 1000)) # Schedule ASAP loop.run_forever()</pre> <p>#ctrl+c pour stopper</p>	<p>Pin(2) on</p> <p>Pin(15) on</p> <p>Pin(2) off</p> <p>Pin(15) off</p> <p>Pin(2) on</p> <p>Pin(2) off</p> <p>Pin(15) on</p> <p>Pin(2) on</p> <p>Pin(2) off</p> <p>Pin(15) off</p>
<p>Traceback (most recent call last):</p> <p>File "C:\Users\anthony\esp8266\test.py", line 21, in &lt;module&gt;</p> <p>File "uasyncio/core.py", line 173, in run_forever</p> <p>File "uasyncio/__init__.py", line 69, in wait</p> <p>KeyboardInterrupt:</p> <p>&gt;&gt;&gt;</p>	

```
>>> import uasyncio
>>> dir(uasyncio)
['__class__', '__name__', '__path__', 'DEBUG', 'log', 'select', 'sleep', 'sleep_ms', 'time', 'ucollections',
'uerrno', 'utimeq', 'type_gen', 'set_debug', 'CancelledError', 'TimeoutError', 'EventLoop', 'SysCall',
'SysCall1', 'StopLoop', 'IORead', 'IOWrite', 'IOReadDone', 'IOWriteDone', 'get_event_loop', 'SleepMs',
'cancel', 'TimeoutObj', 'wait_for_ms', 'wait_for', 'coroutine', 'ensure_future', 'Task', '_socket',
'PollEventLoop', 'StreamReader', 'StreamWriter', 'open_connection', 'start_server', 'uasyncio', 'core']
```

### Remarque :

Par défaut il n'y a pas le module uasyncio dans l'esp32, il faut l'installer manuellement après être connecté sur votre point d'accès wifi.

```
>>> import upip
>>> upip.install('micropython-uasyncio')

Installing to: /lib/
Warning: micropython.org SSL certificate is not validated
Installing micropython-uasyncio 2.0 from https://micropython.org/pi/uasyncio/uasyncio-2.0.tar.gz
Installing micropython-uasyncio.core 2.0 from https://micropython.org/pi/uasyncio.core/uasyncio.core-2.0.tar.gz
```

`upip.install('micropython-ssd1306')` # pour l'afficheur oled ssd1306

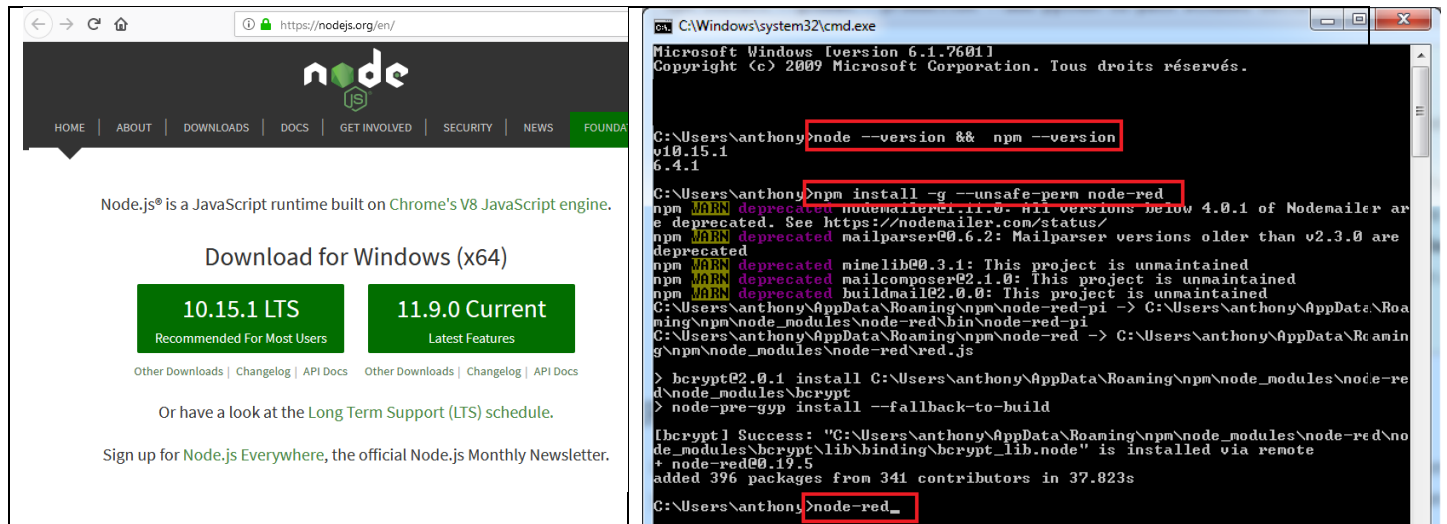
## Annexe 4 : Utilisation de Node-red

Node-RED est un outil de développement basé sur une programmation visuelle pour connecter un ensemble de périphériques matériels, des API et des services en ligne dans le cadre de l'Internet des objets.

### 3.1 Installation de Node-red sous Windows :

- Commencer par installer node js

<https://nodejs.org/en/>



<https://nodered.org/docs/platforms/windows>

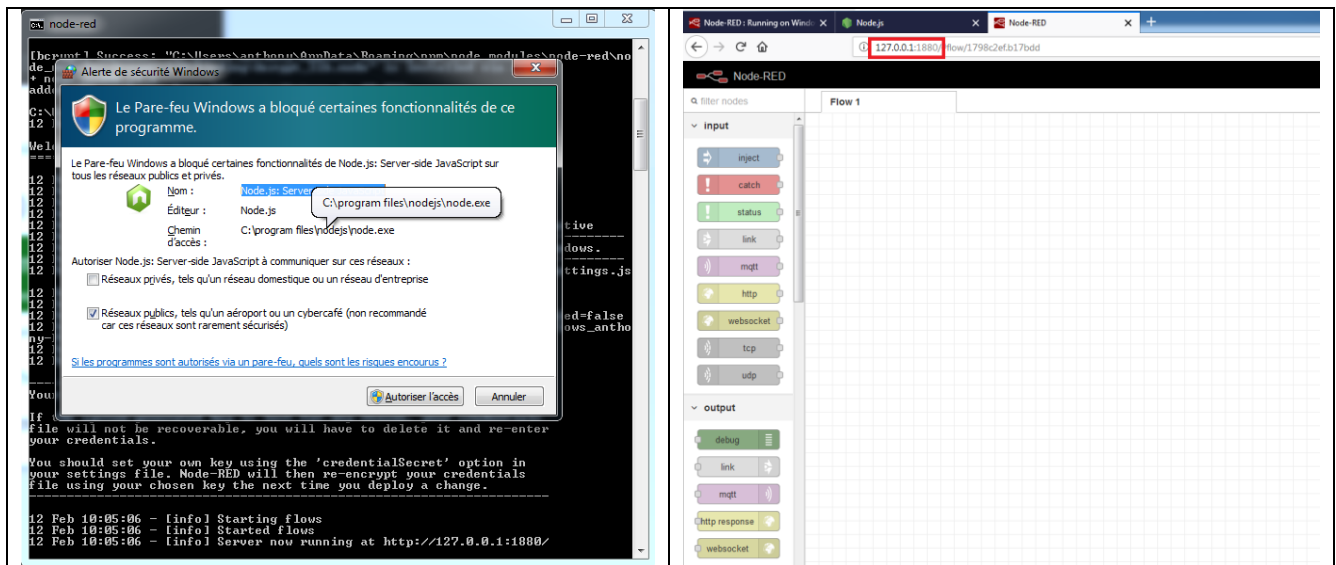
- Puis dans la console saisir les commandes suivantes :

```
node --version && npm --version
```

```
npm install -g --unsafe-perm node-red
```

```
node-red
```

## Ne pas fermer la console



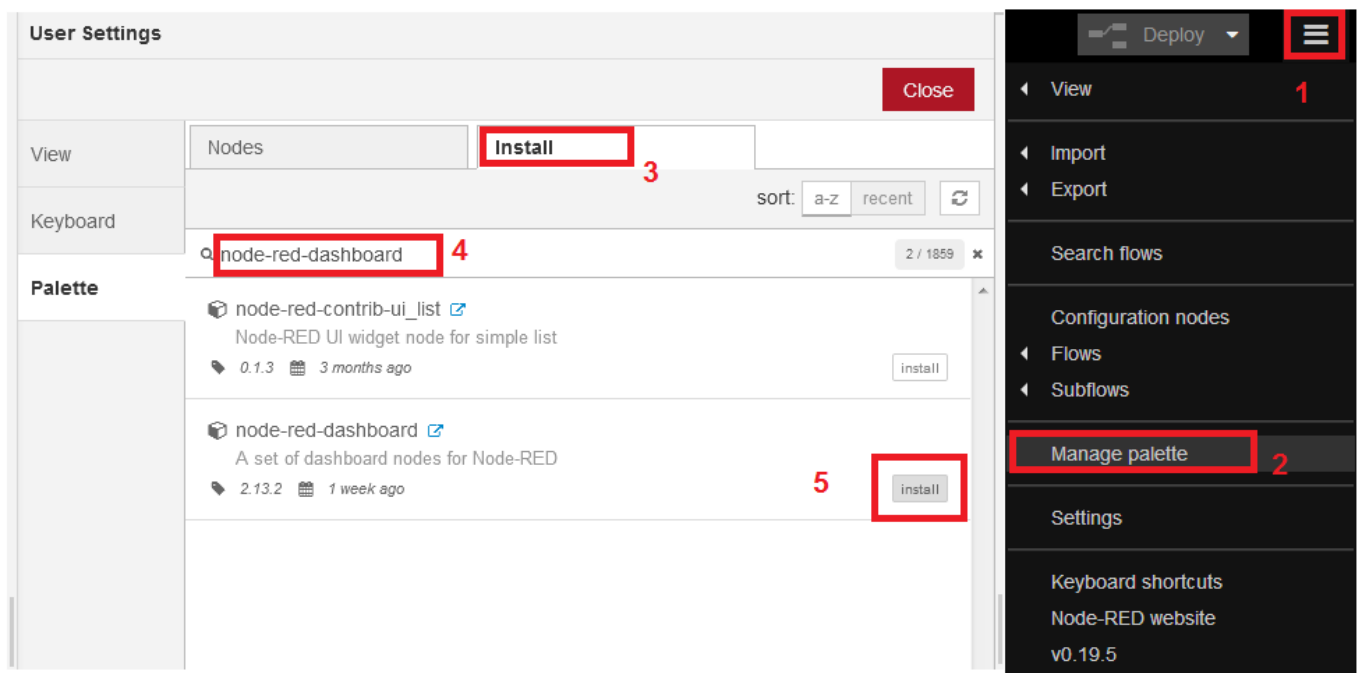
Ouvrir un navigateur puis saisir l'adresse suivante :

<http://127.0.0.1:1880>

Ne pas fermer la console.

### 3.2 Add-ons

Ajouts d'outils de gestion graphique.



Il en va de même pour d'autres outils en connaissant le nom du fichier à installer.

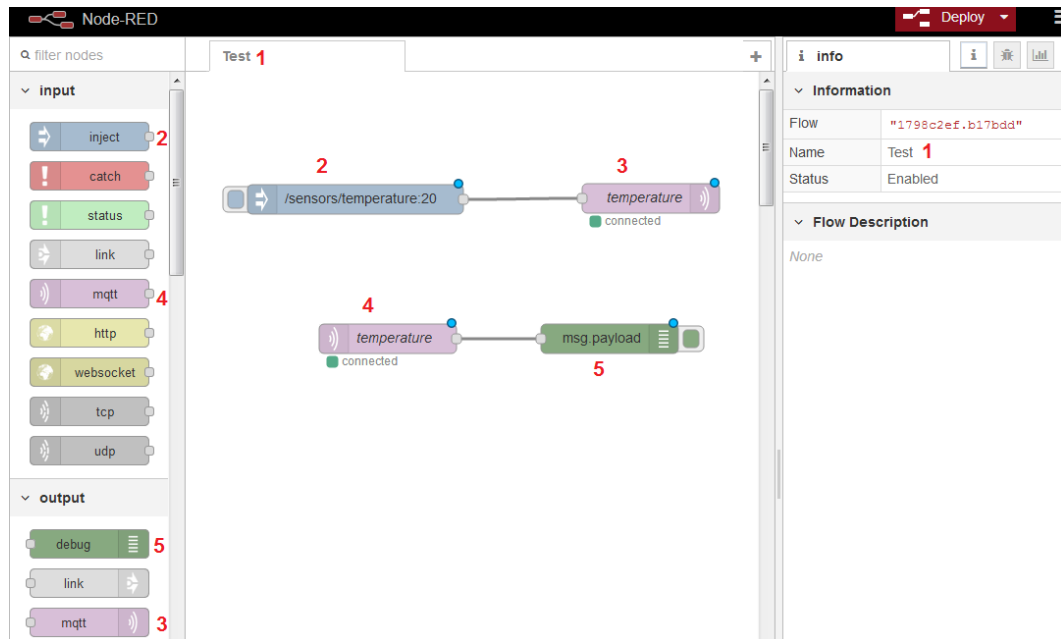


## 3.2 Installation sous linux

```
curl -sL https://deb.nodesource.com/setup_11.x | sudo -E bash -
sudo apt-get install -y nodejs
node -v
sudo npm install -g --unsafe-perm node-red
node-red
```

## 3.3 1er pas

Tracer le diagramme suivant :



Configuration du broker shiftr.io :

Edit mqtt out node
Delete
Cancel
Done

node properties

Server: shiftr.io
Topic: Topic
QoS: 0
Retain:
Name: temperature

temperature

connected

Edit mqtt out node > Edit mqtt-broker node
Delete
Cancel
Update

Name: shiftr.io

Connection

Server: broker.shiftr.io
Port: 1883

Client ID: Node-red

Keep alive time (s): 60
Use clean session:
Use legacy MQTT 3.1 support:

Security

Username: weatherSensors
Password:

Ici la valeur est fixe, 20° pour tester le fonctionnement.

Edit inject node

Delete

Cancel

Done

node properties

✉ Payload

20

📄 Topic

/sensors/temperature

☐ Inject once after

0.1

seconds, then

🔄 Repeat

none

🏷 Name

Name

Note: "interval between times" and "at a specific time" will use cron. "interval" should be less than 596 hours. See info box for details.

/sensors/temperature:20

Edit mqtt in node

Delete

Cancel

Done

node properties

🌐 Server

shiftr.io

📄 Topic

/sensors/temperature

⚙ QoS

0

🏷 Name

temperature

temperature

connected

Test du diagramme :

1

Deploy

2

debug

3 : clique

/sensors/temperature:20

temperature

connected

publish

temperature

connected

msg.payload

subscribe

sensors/temperature

20

Node-red - 14:22:47 - Q0 SR

Node-red

humidite

sensors

temperature

pression

- 1- Cliquer sur Deploy
- 2- Puis sur debug pour voir les messages de debug
- 3- Enfin sur le bouton envoi (symbole inject)

L'information de température 20 ° test envoyé vers le broker, puis revient.

Lycée Gabriel Touchard – Le Mans

Page 31/38

### 3.4 Gestion graphique

#### Utilisation d'un vu mètre (gauge)

The image shows the configuration process for a gauge widget. On the left, a palette of widgets is displayed under the 'dashboard' category, with the 'gauge' widget highlighted by a red box. In the center, two configuration panels are shown. The 'Edit gauge node' panel on the left has fields for 'Group' (set to '[Home] Temp'), 'Type' (set to 'Gauge'), 'Label' (set to 'gauge'), 'Value format' (set to '{{value}}'), 'Units' (set to 'units'), and 'Range' (min 0, max 50). The 'Edit dashboard group node' panel on the right has fields for 'Name' (set to 'Temp'), 'Tab' (set to 'Home'), 'Width' (set to 6), and checkboxes for 'Display group name' and 'Allow group to be collapsed'. A red box highlights the 'Tab' dropdown in this panel. On the right, a 'Tabs & Links' panel shows the hierarchy: 'Home' > 'Temp' > 'gauge'. At the bottom right, a small diagram shows a 'temperature' sensor connected to a 'msg payload' node, which is then connected to the 'gauge' widget.

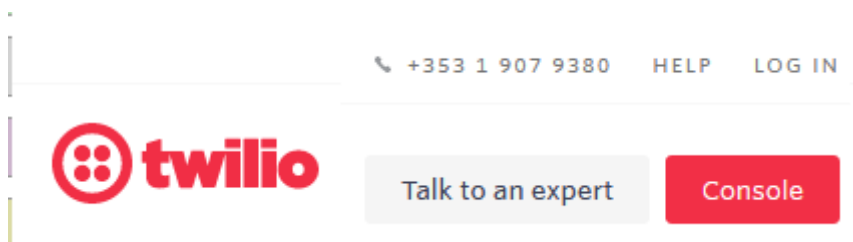
Où se trouve le vu mètre une fois que l'on clique sur **Deploy** ?

The image shows the deployed dashboard. On the left, the 'dashboard' configuration panel is visible with the 'Layout' tab selected. A red box highlights the 'Layout' tab, and a red arrow points from it to the right. On the right, a browser window displays the deployed dashboard. The browser address bar shows '127.0.0.1:1880/ui/#1/0'. The dashboard has a blue header with the text 'Home'. Below the header, there is a section titled 'Temp' containing a gauge widget. The gauge is labeled 'gauge' and shows a value of '20 units' on a scale from 0 to 50. The gauge needle is pointing to the value 20.

Une nouvelle fenêtre dans le navigateur s'ouvre avec le vu mètre.

### 3.5 Envoi de sms

Inscription sur le site <https://www.twilio.com/>



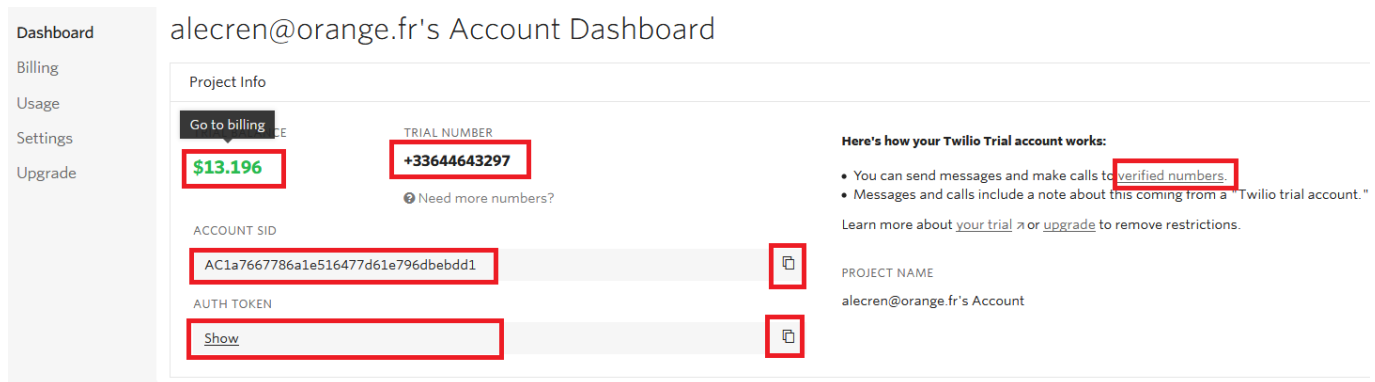
Trial version : 200 sms gratuits, puis payant : 7,6 cts /sms

Exemple de dashboard une fois l'inscription terminée

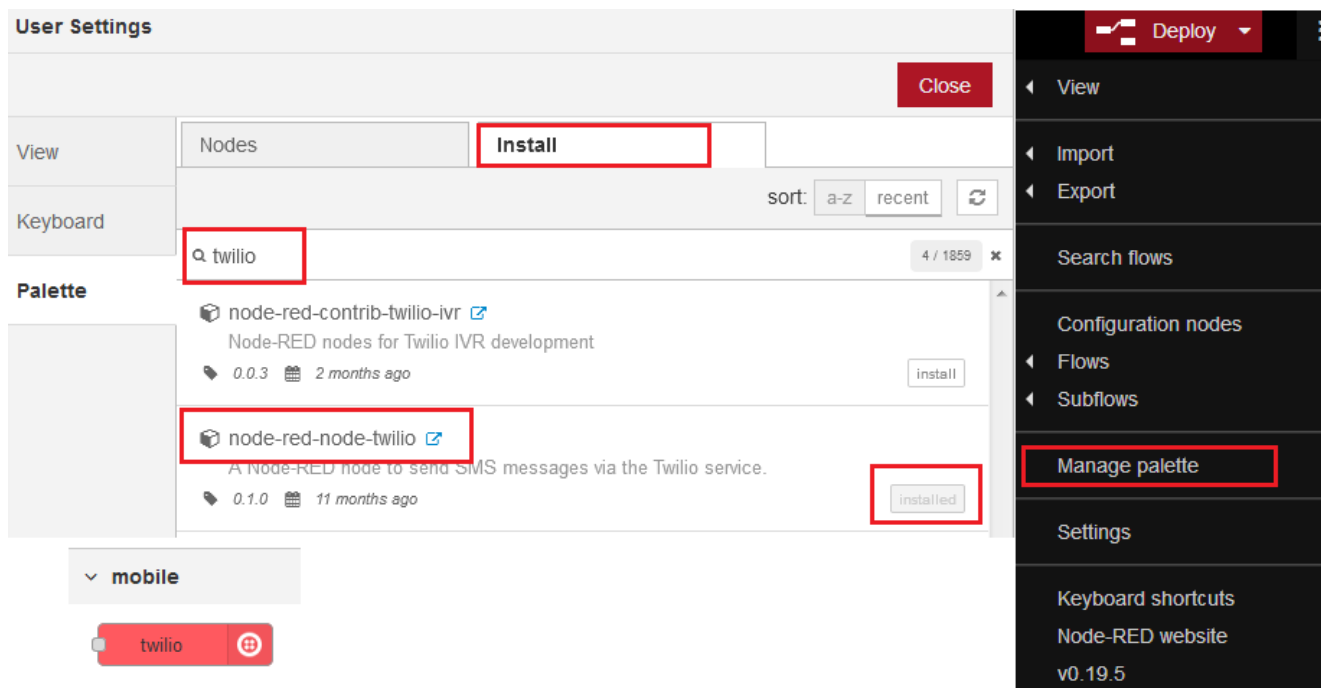
Twilio vous fourni un numéro de téléphone : ici Trial number

Il est possible d'envoyer des sms que sur des numéros de téléphones vérifiés : (le vôtre lors de l'inscription).

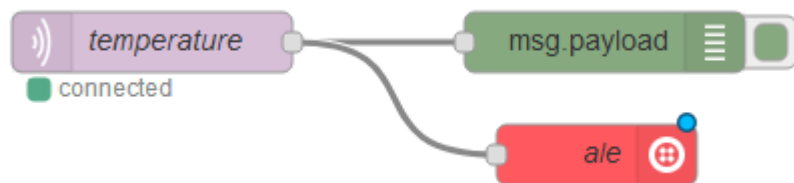
En vert le crédit sms restant



Installation dans Node-red



## Configuration de twilio



Edit twilio out node
Delete Cancel Done

node properties

Twilio: anthony
Output: SMS
To: +33 6 00 00 00 00 **Votre N°de portable**
Name: ale

Edit twilio out node > Edit twilio-api node
Delete Cancel Update

Account SID: AC1a7667786a1e516477d61e796dbebdd1
From: +33644643297 **trial number**
Token: .....
Name: anthony



Il existe de nombreuses possibilités avec Node-red

Ce document n'a pas pour but d'expliquer toutes les possibilités mais de démarrer rapidement

Liens :

<http://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/pedagogiques/8054/8054-objets-communicants.pdf>

[https://www.youtube.com/results?search\\_query=node+red](https://www.youtube.com/results?search_query=node+red)

## Annexe 5 : Analyse du protocole MQTT



<http://mqtt.org/documentation>

### Connect Command (client to server)

[http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718028](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718028)

MQ Telemetry Transport Protocol															
▼ Connect Command															
▶ 0001 0000 = Header Flags: 0x10 (Connect Command)															
Msg Len: 53															
Protocol Name: MQTT															
Version: 4															
▶ 1100 0010 = Connect Flags: 0xc2															
Keep Alive: 60															
Client ID: processing															
User Name: weatherSensors															
Password: bme280Sensors															
0000	90	4d	4a	a3	0e	00	00	25	22	8a	86	a0	08	00	45 00
0010	00	6b	93	a5	40	00	40	06	96	d9	c0	a8	01	15	36 4c
0020	18	05	b7	ca	07	5b	79	8f	ed	67	52	30	38	88	80 18
0030	00	e5	4a	0b	00	00	01	01	08	0a	d4	85	d3	f7	07 17
0040	39	fa	10	35	00	04	4d	51	54	54	04	c2	00	3c	00 0a
0050	70	72	6f	63	65	73	73	69	6e	67	00	0e	77	65	61 74
0060	68	65	72	53	65	6e	73	6f	72	73	00	0d	62	6d	65 32
0070	38	30	53	65	6e	73	6f	72	73						

### Connect Command

0001 0000 = Header Flags: 0x10 (Connect Command)

Msg Len: 53

Protocol Name: MQTT

Version: 4

1100 0010 = Connect Flags: 0xc2

Keep Alive: 60

Client ID: processing

User Name: weatherSensors

Password: bme280Sensors

0000	10	35	00	04	4d	51	54	54	04	c2	00	3c	00	0a	70 72
0010	6f	63	65	73	73	69	6e	67	00	0e	77	65	61	74	68 65
0020	72	53	65	6e	73	6f	72	73	00	0d	62	6d	65	32	38 30
0030	53	65	6e	73	6f	72	73								

**Connect Ack (server to client)**

[http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718033](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718033)

MQ Telemetry Transport Protocol															
▼ Connect Ack															
▶ 0010 0000 = Header Flags: 0x20 (Connect Ack)															
Msg Len: 2															
.... .... 0000 0000 = Connection Ack: Connection Accepted (0)															
0000	00	25	22	8a	86	a0	90	4d	4a	a3	0e	00	08	00	45 00
0010	00	38	02	8f	40	00	ea	06	7e	22	36	4c	18	05	c0 a8
0020	01	15	07	5b	b7	ca	52	30	38	88	79	8f	ed	9e	80 18
0030	00	72	ab	7f	00	00	01	01	08	0a	07	17	3a	0e	d4 85
0040	d3	f7	20	02	00	00									.. ...

**Connect Ack**

0010 0000 = Header Flags: 0x20 (Connect Ack)

Msg Len: 2

.... .... 0000 0000 = Connection Ack: Connection Accepted (0)

0000 20 02 00 00

**Subscribe Request (client to server)**

[http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718063](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718063)

MQ Telemetry Transport Protocol															
▼ Connect Ack															
▶ 0010 0000 = Header Flags: 0x20 (Connect Ack)															
Msg Len: 2															
.... .... 0000 0000 = Connection Ack: Connection Accepted (0)															
0000	00	25	22	8a	86	a0	90	4d	4a	a3	0e	00	08	00	45 00
0010	00	38	02	8f	40	00	ea	06	7e	22	36	4c	18	05	c0 a8
0020	01	15	07	5b	b7	ca	52	30	38	88	79	8f	ed	9e	80 18
0030	00	72	ab	7f	00	00	01	01	08	0a	07	17	3a	0e	d4 85
0040	d3	f7	20	02	00	00									.. ...

**Subscribe Request**

1000 0010 = Header Flags: 0x82 (Subscribe Request)

Msg Len: 25

Message Identifier: 1

Topic: /sensors/temperature

.... ..00 = Granted Qos: Fire and Forget (0)

0000 82 19 00 01 00 14 2f 73 65 6e 73 6f 72 73 2f 74 ...../sensors/t

0010 65 6d 70 65 72 61 74 75 72 65 00 emperature.

**Subscribe Ack (server to client)**

[http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718068](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718068)

**Subscribe Ack**

1001 0000 = Header Flags: 0x90 (Subscribe Ack)

Msg Len: 3

Message Identifier: 1

.... ..00 = Granted Qos: Fire and Forget (0)



A PUBLISH Control Packet is sent **from a Client to a Server** or **from Server to a Client** to transport an Application Message.

### Publish Message (server to client)

[http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718037](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718037)

MQ Telemetry Transport Protocol

Publish Message

0011 0000 = Header Flags: 0x30 (Publish Message)

Msg Len: 27

Topic: /sensors/temperature

Message: 25.80

0000	00	25	22	8a	86	a0	90	4d	4a	a3	0e	00	08	00	45	00	..%"....M J.....E.
0010	00	51	02	93	40	00	ea	06	7e	05	36	4c	18	05	c0	a8	.Q...@... ~.6L....
0020	01	15	07	5b	b7	ca	52	30	38	9b	79	8f	ed	e9	80	18	...[...R0 8.y.....
0030	00	72	2e	40	00	00	01	01	08	0a	07	17	3c	ad	d4	85	.r.@.....<....
0040	d4	3e	30	1b	00	14	2f	73	65	6e	73	6f	72	73	2f	74	.>0.../s ensors/t
0050	65	6d	70	65	72	61	74	75	72	65	32	35	2e	38	30		emperatu re25.80

### Publish Message **incoming**

0011 0000 = Header Flags: 0x30 (Publish Message)

Msg Len: 27

Topic: /sensors/temperature

Message: 25.80

```
0000 30 1b 00 14 2f 73 65 6e 73 6f 72 73 2f 74 65 6d 0.../sensors/tem
0010 70 65 72 61 74 75 72 65 32 35 2e 38 30          perature25.80
```

### Publish Message (client to server)

[http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718037](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718037)

MQ Telemetry Transport Protocol

Publish Message

0011 0000 = Header Flags: 0x30 (Publish Message)

Msg Len: 27

Topic: /sensors/temperature

Message: 25.80

0000	00	25	22	8a	86	a0	90	4d	4a	a3	0e	00	08	00	45	00	..%"....M J.....E.
0010	00	51	02	93	40	00	ea	06	7e	05	36	4c	18	05	c0	a8	.Q...@... ~.6L....
0020	01	15	07	5b	b7	ca	52	30	38	9b	79	8f	ed	e9	80	18	...[...R0 8.y.....
0030	00	72	2e	40	00	00	01	01	08	0a	07	17	3c	ad	d4	85	.r.@.....<....
0040	d4	3e	30	1b	00	14	2f	73	65	6e	73	6f	72	73	2f	74	.>0.../s ensors/t
0050	65	6d	70	65	72	61	74	75	72	65	32	35	2e	38	30		emperatu re25.80

### Publish Message **outcoming**

0011 0000 = Header Flags: 0x30 (Publish Message)

Msg Len: 24

Topic: /sensors/temperature

Message: 20

```
0000 30 18 00 14 2f 73 65 6e 73 6f 72 73 2f 74 65 6d 0.../sensors/tem
0010 70 65 72 61 74 75 72 65 32 30                    perature20
```

**Disconnect (client to server)**[http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718090](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718090)

Disconnect Req

1110 0000 = Header Flags: 0xe0 (Disconnect Req)

Msg Len: 0

0000 e0 00