# IoT Dashboard - Sensores, MQTT, Telegraf, InfluxDB y Grafana

José Juan Sánchez Hernández - 8 de Febrero de 2021

# Tabla de contenidos

# 1. Descripción del proyecto

En cada aula del instituto vamos a tener un Wemos D1 mini, un sensor de CO2 y un sensor de temperatura/humedad DHT11 que van a ir tomando medidas de forma constante y las van a ir publicando en un **topic** de un **broker** MQTT. Podríamos seguir la siguiente estructura de nombres para los **topics** del edificio:

```
iescelia/aula<número>/temperature
iescelia/aula<número>/humidity
iescelia/aula<número>/co2
```

Por ejemplo para el `aula20` tendríamos los siguientes **topics**:

```
iescelia/aula20/temperature
iescelia/aula20/humidity
iescelia/aula20/co2
```

También existirá un agente de Telegraf que estará suscrito a los **topics** del **broker** MQTT donde se publican los valores recogidos por los sensores. El agente de Telegraf insertará los valores que recoge del **broker** MQTT en una base de datos InfluxDB, que es un sistema gestor de bases de datos diseñado para almacenar series temporales de datos.Finalmente tendremos un servicio web Grafana que nos permitirá visualizar los datos en un panel de control.

Para realizar el despliegue de los servicios de MQTT, Telegraf, InfluxDB y Grafana, vamos a utilizar Docker Compose y contenedores Docker.

# 2. Arquitectura del sistema

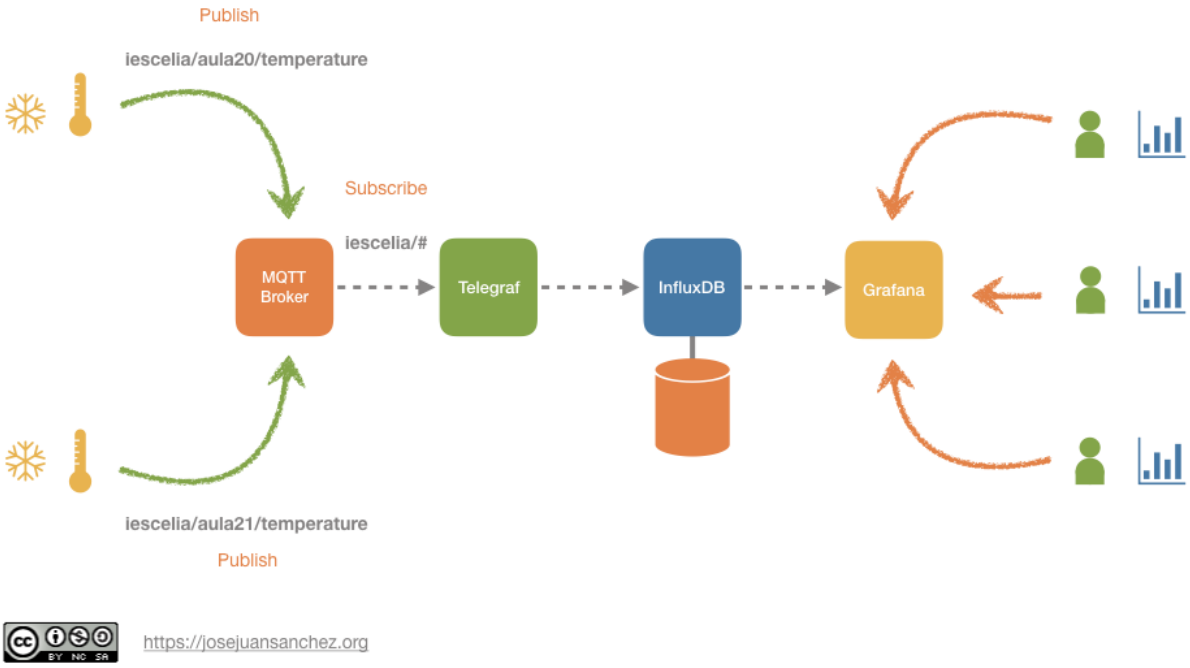


*Figure 1. Arquitectura del sistema utilizado.*

# 3. Sensores

## 3.1. Wemos D1 mini

Wemos D1 mini es una placa de prototipado que incluye el SoC (*System on Chip*) **ESP8266** que integra un microcontrolador de propósito general de 32 bits y conectividad WiFi.


*Figure 2. Wemos D1 mini*

Puedes encontrar más información en la documentación oficial.

## 3.2. Sensor de temperatura/humedad DHT11

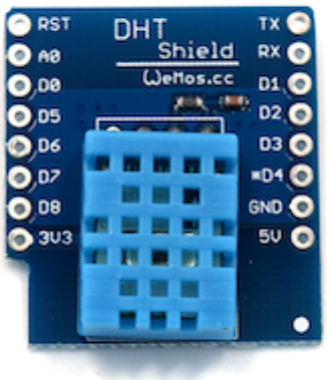


*Figure 3. Sensor de temperatura/humedad DHT11*

Puedes encontrar más información sobre el sensor DHT11 en la web de Adafruit.

### 3.2.1. Lectura del sensor de temperatura/humedad DHT11 y publicación de datos en el broker MQTT

Vamos a hacer uso de la librería Adafruit MQTT para conectar con el **broker** MQTT y publicar los datos que vamos obteniendo de los sensores DHT11. Podemos utilizar el siguiente código de ejemplo:

```
#include "DHT.h"
#include <ESP8266WiFi.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"

#define DHTPIN D4
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

#define WLAN_SSID       "PUT_YOUR_WLAN_SSID_HERE" ①
#define WLAN_PASS       "PUT_YOUR_WLAN_PASS_HERE" ②

#define MQTT_SERVER      "PUT_YOUR_MQTT_SERVER_HERE" ③
#define MQTT_SERVERPORT  1883
#define MQTT_USERNAME    ""
#define MQTT_KEY         ""
#define MQTT_FEED_TEMP   "iescelia/aula22/temperature" ④
#define MQTT_FEED_HUMI   "iescelia/aula22/humidity" ⑤

WiFiClient client;

Adafruit_MQTT_Client mqtt(&client, MQTT_SERVER, MQTT_SERVERPORT, MQTT_USERNAME, MQTT_USERNAME, MQTT_KEY);

Adafruit_MQTT_Publish temperatureFeed = Adafruit_MQTT_Publish(&mqtt, MQTT_FEED_TEMP);

Adafruit_MQTT_Publish humidityFeed = Adafruit_MQTT_Publish(&mqtt, MQTT_FEED_HUMI);

//-------------------------------------------

void connectWiFi();

//-------------------------------------------

void setup() {
  Serial.begin(115200);
  Serial.println("IoT demo");
  dht.begin();
  connectWiFi();
  connectMQTT();
}

//-------------------------------------------

void loop() {
  delay(2000);

  float h = dht.readHumidity();
  float t = dht.readTemperature();
```

```
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.println(" *C ");

  temperatureFeed.publish(t);
  humidityFeed.publish(h);
}

//-------------------------------------------

void connectWiFi() {
  WiFi.begin(WLAN_SSID, WLAN_PASS);
  while (WiFi.status() != WL_CONNECTED) {
     delay(500);
     Serial.print(".");
  }

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

//-------------------------------------------

void connectMQTT() {
  if (mqtt.connected())
    return;

  Serial.print("Connecting to MQTT... ");
  while (mqtt.connect() != 0) {
       Serial.println("Error. Retrying MQTT connection in 5 seconds...");
       mqtt.disconnect();
       delay(5000);
  }
}
```

① SSID de la red WiFi donde vamos a conectar el sensor.

② Contraseña de la red WiFi a la que vamos a conectar el sensor.

③ Dirección IP del broker MQTT al que vamos a enviar los datos que recoge el sensor.

④ Topic donde vamos a publicar los datos de temperatura que recoge el sensor.

⑤ Topic donde vamos a publicar los datos de humedad que recoge el sensor.

Ver el código fuente en GitHub.

## 3.3. Sensor de CO2

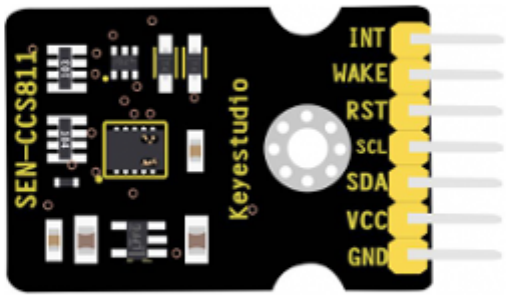


*Figure 4. Sensor de CO2*

Puedes encontrar más información sobre este sensor en la documentación oficial.

### 3.3.1. Lectura del sensor de CO2 y publicación de datos en el broker MQTT

Vamos a hacer uso de la librería Adafruit MQTT para conectar con el **broker** MQTT y publicar los datos que vamos obteniendo de los sensores CCS811. Podemos utilizar el siguiente código de ejemplo:

```
#include <ESP8266WiFi.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#include <CCS811.h>

// WiFi configuration
#define WLAN_SSID       "PUT_YOUR_WLAN_SSID_HERE" ①
#define WLAN_PASS       "PUT_YOUR_WLAN_PASS_HERE" ②

// MQTT configuration
#define MQTT_SERVER      "PUT_YOUR_MQTT_SERVER_HERE" ③
#define MQTT_SERVERPORT  1883
#define MQTT_USERNAME    ""
#define MQTT_KEY         ""
#define MQTT_FEED_CO2   "iescelia/aula22/co2" ④
#define MQTT_FEED_TVOC  "iescelia/aula22/tvoc" ⑤

// WiFi connection
WiFiClient client;

// MQTT connection
Adafruit_MQTT_Client mqtt(&client, MQTT_SERVER, MQTT_SERVERPORT, MQTT_USERNAME, MQTT_USERNAME, MQTT_KEY);

// Feed to publish CO2
Adafruit_MQTT_Publish co2Feed = Adafruit_MQTT_Publish(&mqtt, MQTT_FEED_CO2);
```

```
// Feed to publish TVOC
Adafruit_MQTT_Publish tvocFeed = Adafruit_MQTT_Publish(&mqtt, MQTT_FEED_TVOC);

CCS811 sensor;

//-----------------------------------------------

void connectWiFi();
void connectMQTT();
void initSensor();

//-----------------------------------------------

void setup() {
  Serial.begin(115200);
  Serial.println("CO2 IES Celia");
  connectWiFi();
  connectMQTT();
  initSensor();
}

//-----------------------------------------------

void loop() {
  // Wait a few seconds between measurements.
  delay(1000);

  if(sensor.checkDataReady() == false){
    Serial.println("Data is not ready!");
    return;
  }

  float co2 = sensor.getCO2PPM();
  float tvoc = sensor.getTVOCPPB();

  Serial.print("CO2: ");
  Serial.print(co2);
  Serial.print(" ppm\t");
  Serial.print("TVOC: ");
  Serial.print(tvoc);
  Serial.println(" ppb");

  co2Feed.publish(co2);
  tvocFeed.publish(tvoc);
}

//-----------------------------------------------

void initSensor() {
  // Wait for the chip to be initialized completely
  while(sensor.begin() != 0){
    Serial.println("Failed to init chip, please check if the chip connection is fine");
    delay(1000);
  }

  // eClosed      Idle (Measurements are disabled in this mode)
  // eCycle_1s    Constant power mode, IAQ measurement every second
  // eCycle_10s   Pulse heating mode IAQ measurement every 10 seconds
  // eCycle_60s   Low power pulse heating mode IAQ measurement every 60 seconds
```

```
  // eCycle_250ms Constant power mode, sensor measurement every 250ms
  sensor.setMeasCycle(sensor.eCycle_250ms);
}

//-----------------------------------------------

void connectWiFi() {
  WiFi.begin(WLAN_SSID, WLAN_PASS);
  while (WiFi.status() != WL_CONNECTED) {
     delay(500);
     Serial.print(".");
  }

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

//-----------------------------------------------

void connectMQTT() {
  if (mqtt.connected())
    return;

  Serial.print("Connecting to MQTT... ");
  while (mqtt.connect() != 0) {
       Serial.println("Error. Retrying MQTT connection in 5 seconds...");
       mqtt.disconnect();
       delay(5000);
  }
}
```

① SSID de la red WiFi donde vamos a conectar el sensor.

② Contraseña de la red WiFi a la que vamos a conectar el sensor.

③ Dirección IP del broker MQTT al que vamos a enviar los datos que recoge el sensor.

④ Topic donde vamos a publicar los datos de CO2 que recoge el sensor.

⑤ Topic donde vamos a publicar los datos de TVOC que recoge el sensor.

Ver el código fuente en GitHub.

# 4. MQTT Broker

**MQTT** (*MQ Telemetry Transport*) es un protocolo de mensajería estándar utilizado en las aplicaciones de **Internet de las cosas (IoT)**. Se trata de un protocolo de mensajería muy ligero basado en el patrón *publish/subscribe*, donde los mensajes son publicados en un *topic* de un **MQTT Broker** que se encargará de distribuirlos a todos los suscriptores que se hayan suscrito a dicho *topic*.

Eclipse Mosquitto será el **MQTT Broker** que vamos a utilizar en este proyecto.

## 4.1. Descripción del servicio en `docker-compose.yml`

`docker-compose.yml`

```
mosquitto: ①
  image: eclipse-mosquitto:2 ②
  ports:
    - 1883:1883 ③
  volumes:
    - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf ④
    - mosquitto_data:/mosquitto/data ⑤
    - mosquitto_log:/mosquitto/log ⑥
```

① Nombre del servicio dentro del archivo `docker-compose.yml`.

② Vamos a utilizar la versión **2** de la imagen oficial eclipse-mosquitto.

③ El broker MQTT estará aceptando peticiones en el puerto 1883.

④ Creamos un volumen de tipo *bind mount* para enlazar el archivo de configuración `mosquitto.conf` que tenemos en nuestro directorio local `mosquitto` con el directorio `/mosquitto/config/mosquitto.conf` del broker MQTT.

⑤ Creamos un volumen para almacenar los mensajes que se reciben.

⑥ Creamos un volumen para almacenar los mensajes de log.

## 4.2. Archivo de configuración `mosquitto.conf`

Este archivo estará almacenado en el directorio `./mosquitto/mosquitto.conf` de nuestra máquina local. Como vamos a utilizar la **versión 2** del broker MQTT eclipse-mosquitto vamos a necesitar crear un archivo de configuración llamado `mosquitto.conf` que incluya las siguientes opciones de configuración.

`mosquitto.conf`

```
listener 1883 ①
allow_anonymous true ②
```

① En esta línea estamos configurando el *listener* para que acepte conexiones desde cualquiera de

sus interfaces de red, es equivalente a poner `listener 1883 0.0.0.0`. Si sólo quisiéramos recibir conexiones por una interfaz de red específica se podría configurar aquí de forma manual.

② A partir de la **versión 2** es necesario indicar un método de autenticación para el *listener*. Con esta configuración vamos a permitir conexiones desde cualquier cliente. Aquí sería posible definir un método de autenticación basado en contraseña para restringir el acceso al broker.

En la documentación oficial, puede consultar los cambios que hay que tener en cuenta en el archivo de configuración para migrar de la versión 1.x a la 2.0.

También puede encontrar más información sobre todas las opciones de configuración posibles en la documentación oficial.

En este punto, el contenido de nuestro directorio de trabajo debe tener los siguientes archivos.

```
.
├──── docker-compose.yml
├──── mosquitto
    └──── mosquitto.conf
```

## 4.3. Cliente MQTT para publicar en un topic (*publish*)

En esta sección vamos a explicar cómo podemos hacer uso de un cliente MQTT para publicar mensajes de prueba en el broker MQTT que acabamos de crear.

Esta comprobación deberíamos realizarla antes de empezar a trabajar directamente con los sensores sobre el broker MQTT. Una vez que hayamos comprobado que podemos publicar y suscribirnos a los mensajes del broker con este cliente, ya podríamos realizar el siguiente paso para publicar mensajes con los sensores y recibirlos con el agente de Telegraf.

**Ejemplo**:

En este ejemplo vamos a publicar el valor `30` en el topic `iescelia/aula22/co2` del broker MQTT `test.mosquitto.org`. Este broker es de prueba, por lo tanto, **tendremos que cambiar este broker por la dirección IP de nuestro broker MQTT**.

```
docker run --init -it --rm efrecon/mqtt-client pub -h test.mosquitto.org -p 1883 -t "iescelia/aula22/co2" -m 30
```

Explicación de los parámetros utilizados:

```
docker run --init -it --rm efrecon/mqtt-client \ ①
pub \ ②
-h test.mosquitto.org \ ③
-p 1883 \ ④
-t "iescelia/aula22/co2" \ ⑤
-m 30 ⑥
```

① Utilizamos la imagen Docker efrecon/mqtt-client que contiene el cliente MQTT (`mosquitto_pub`)

para publicar mensajes en un topic de un broker MQTT.

② Utilizamos el comando `pub` para publicar un mensaje en el broker MQTT.

③ Con el parámetro `-h` indicamos el hosts (broker MQTT) con el que queremos conectarnos. En este ejemplo estamos utilizando el broker de prueba de `test.mosquitto.org`, pero **tendremos que cambiar este broker por la dirección IP de nuestro broker MQTT**.

④ Con el parámetro `-p` indicamos el puerto, que por defecto, será el puerto `1883`.

⑤ Con el parámetro `-t` indicamos el topic que vamos a utilizar para publicar nuestro mensaje. En este ejemplo, estamos utilizando el topic `iescelia/aula22/co2`.

⑥ Con el parámetro `-m` indicamos el contenido del mensaje que queremos publicar. En este ejemplo, estamos publicando el mensaje `30`.

No olvide que en este ejemplo debemos reemplazar el broker de prueba de `test.mosquitto.org` por la dirección IP de nuestro broker MQTT.

## 4.4. Cliente MQTT para suscribirse a un topic (*subscribe*)

Podemos hacer uso de un cliente MQTT para suscribirnos a los mensajes que se publican en un topic específico del broker MQTT.

Esta comprobación deberíamos realizarla antes de empezar a trabajar directamente con los sensores sobre el broker MQTT. Una vez que hayamos comprobado que podemos publicar y suscribirnos a los mensajes del broker con este cliente, ya podríamos realizar el siguiente paso para publicar mensajes con los sensores y recibirlos con el agente de Telegraf.

**Ejemplo:**

En este ejemplo nos vamos a suscribir al topic `iescelia/ del broker MQTT test.mosquitto.org. Con el wildcard` estamos indicando que queremos suscribirnos a todos los topics que existan dentro de `iescelia/`, es decir, todos los mensajes que se publiquen en cada una de las aulas.

Tenga en cuenta que **tendrá que cambiar el broker `test.mosquitto.org` por la dirección IP de nuestro broker MQTT**.

```
docker run --init -it --rm efrecon/mqtt-client sub -h test.mosquitto.org -t "iescelia/#"
```

Explicación de los parámetros utilizados:

```
docker run --init -it --rm efrecon/mqtt-client \ ①
sub \ ②
-h test.mosquitto.org \ ③
-t "iescelia/#" ④
```

① Utilizamos la imagen Docker efrecon/mqtt-client que contiene el cliente MQTT (`mosquitto_sub`)

para suscribirse a un topic de un broker MQTT.

② Utilizamos el comando `sub` para suscribirnos a un topic del broker MQTT.

③ Con el parámetro `-h` indicamos el hosts (broker MQTT) con el que queremos conectarnos. En este ejemplo estamos utilizando el broker de prueba de `test.mosquitto.org`, pero **tendremos que cambiar este broker por la dirección IP de nuestro broker MQTT**.

④ Con el parámetro `-t` indicamos el topic al que vamos a suscribirnos. En este ejemplo, estamos utilizando el topic `iescelia/. Con el wildcard` estamos indicando que queremos suscribirnos a todos los topics que existan dentro de `iescelia/`, es decir, todos los mensajes que se publiquen en cada una de las aulas. También sería posible suscribirnos al topic específico de un aula, por ejemplo: `iescelia/aula22/co2`.

No olvide que en este ejemplo debemos reemplazar el broker de prueba de `test.mosquitto.org` por la dirección IP de nuestro broker MQTT.

# 5. Telegraf

**Telegraf** es un agente que nos permite **recopilar y reportar métricas**. Las métricas recogidas se pueden enviar a almacenes de datos, colas de mensajes o servicios como: InfluxDB, Graphite, OpenTSDB, Datadog, Kafka, MQTT, NSQ, entre otros.

## 5.1. Descripción del servicio en `docker-compose.yml`

`docker-compose.yml`

```
  telegraf: ①
    image: telegraf ②
      volumes:
        - ./telegraf/telegraf.conf:/etc/telegraf/telegraf.conf ③
      depends_on: ④
        - influxdb
```

① Nombre del servicio dentro del archivo `docker-compose.yml`.

② Utilizamos la imagen Docker telegraf.

③ Creamos un volumen de tipo *bind mount* para enlazar el archivo de configuración `telegraf.conf` que tenemos en nuestro directorio local `telegraf` con el archivo `/etc/telegraf/telegraf.conf` del contenedor.

④ Indicamos que este servicio depende del servicio `influxdb` y que no podrá iniciarse hasta que el servicio de `influxdb` se haya iniciado.

## 5.2. Creación del archivo de configuración `telegraf.conf`

En primer lugar tenemos que crear el archivo de configuración `telegraf.conf` en nuestra máquina local. Para crear el archivo de configuración haremos uso del comando `telegram config` dentro del contenedor de Telegraf.

```
docker run --rm telegraf telegraf config > telegraf.conf
```

Una vez que hemos creado el archivo `telegraf.conf` lo movemos al directorio `telegraf` de nuestro proyecto. El contenido de nuestro directorio de trabajo debe tener los siguientes archivos.

```
.
├── docker-compose.yml
├── mosquitto
│   └── mosquitto.conf
└── telegraf
    └── telegraf.conf
```

## 5.3. Configuración del archivo `telegraf.conf` para suscribirnos a un topic MQTT (`inputs.mqtt_consumer`)

Tendremos que buscar la sección `inputs.mqtt_consumer` dentro del archivo `telegraf.conf` y configurar los siguientes valores:

- `servers`
- `topics`
- `data_format`

Existen más directivas de configuración (`qos`, `client_id`, `username`, `password`, etc.), pero en este proyecto sólo vamos a utilizar los valores que hemos indicado anteriormente.

### 5.3.1. `servers`

En esta directiva de configuración indicamos la URL del broker MQTT al que queremos conectarnos. En nuestro caso pondremos el nombre del servicio `mosquitto` que es como lo hemos definido en nuestro archivo `docker-compose.yml`.

### 5.3.2. `topics`

En esta directiva indicamos los topics a los que queremos suscribirnos. En nuestro caso pondremos el topic `iescelia/#`. El carácter `#` al final del topic indica que nos vamos a suscribir a cualquier topic que exista después de la cadena `iescelia/`.

### 5.3.3. `data_format`

En esta directiva indicamos cómo es el formato de los datos que vamos a recibir por MQTT.

Telegraf contiene muchos plugins de propósito general que permiten parsear datos de entrada para convertirlos en el **modelo de datos de métricas que utiliza InfluxDB**.

Los mensajes de entrada que recibe Telegraf pueden estar en los siguientes formatos:

- Value
- InfluxDB Line Protocol
- Collectd
- CSV
- Dropwizard
- Graphite
- Grok
- JSON
- Logfmt
- Nagios

- Wavefront

En este proyecto sólo vamos a estudiar Value y InfluxDB Line Protocol.

Referencia:

- https://github.com/influxdata/telegraf/blob/master/docs/DATA_FORMATS_INPUT.md

## 5.4. Formato: `Value`

El formato `value` permite convertir valores simples en métricas de Telegraf.

Es necesario indicar el tipo de dato al que queremos convertir el valor leído, utilizando la opción de configuración `data_type`. Los tipos de datos disponibles son:

1. `integer`
2. `float` o `long`
3. `string`
4. `boolean`

### 5.4.1. Ejemplo de configuración de la sección `inputs.mqtt_consumer`

Si los mensajes que vamos a recibir por MQTT sólo contienen valores numéricos de tipo real que representan valores de temperatura, tendríamos que indicar:

- `data_format = "value"` y
- `data_type = "float"`.

Una posible configuración para la sección `inputs.mqtt_consumer` del archivo `telegraf.conf` podría ser la siguiente.

```
[[inputs.mqtt_consumer]]
  servers = ["tcp://mosquitto:1883"] ①

  topics = [
    "iescelia/#" ②
  ]

 data_format = "value" ③
 data_type = "float" ④
```

① Indicamos la URL del broker MQTT al que queremos conectarnos. En nuestro caso pondremos el nombre del servicio `mosquitto` que es como lo hemos definido en nuestro archivo `docker-compose.yml`.

② Indicamos los topics a los que queremos suscribirnos. El carácter `#` al final del topic `iescelia/#` indica que nos vamos a suscribir a cualquier topic que haya después de `iescelia/`.

③ Indicamos cuál es el formato de los datos que vamos a recibir por MQTT. En este caso indicamos

el formato `value`, porque en el mensaje MQTT sólo vamos a recibir un valor numérico.

④ Indicamos el tipo de dato del valor numérico que vamos a recibir por MQTT.

**Esta será la opción que vamos a elegir** para configurar la sección `inputs.mqtt_consumer` en nuestro proyecto.

## 5.5. Formato: InfluxDB Line Protocol

El formato `influx` no necesita ninguna configuración adicional. Los valores que se reciben son parseados automáticamente a métricas de Telegraf.

Si utilizamos `data_format = "influx"` los mensajes de entrada tienen que cumplir la sintaxis del protocolo InfluxDB line protocol.

A continuación se muestra un ejemplo de un mensaje con las sintaxis de InfluxDB line protocol.

```
weather,location=us-midwest temperature=82 1465839830100400200
  |    -------------------- -------------  |
  |             |             |             |
  |             |             |             |
+-----------+--------+-+---------+-+---------+
|measurement|,tag_set| |field_set| |timestamp|
+-----------+--------+-+---------+-+---------+
```

El mensaje se compone de cuatro componentes:

- `measurement`: Indica la estructura de datos donde vamos a almacenar los valores en InfluxDB.
- `tag_set`: Es **opcional**. Son *tags* opcionales asociados al dato. Van detrás de una coma después del nombre de la **measurement**, **sin espacios en blanco**.
- `field_set`: Son los datos. Aparecen **después de un espacio en blanco** después de la **meausrement**. Podemos indicar varios datos separados por comas.
- `timestamp`: Es **opcional**. Es una marca de tiempo en Unix con precisión de nanosegundos.

### 5.5.1. Ejemplos de mensajes válidos con la sintaxis del protocolo InfluxDB line protocol

```
weather,location=us-midwest temperature=82 1465839830100400200
```

```
weather temperature=82 1465839830100400200
```

```
weather temperature=82
```

```
weather temperature=82,humidity=71 1465839830100400200
```

```
weather temperature=82,humidity=71
```

### 5.5.2. Ejemplo de configuración de la sección `inputs.mqtt_consumer`

Una posible configuración para la sección `inputs.mqtt_consumer` del archivo `telegraf.conf` podría ser la siguiente.

```
[[inputs.mqtt_consumer]]
  servers = ["tcp://mosquitto:1883"] ①

  topics = [
    "iescelia/#" ②
  ]

  data_format = "influx" ③
```

① Indicamos la URL del broker MQTT al que queremos conectarnos. En nuestro caso pondremos el nombre del servicio `mosquitto` que es como lo hemos definido en nuestro archivo `docker-compose.yml`.

② Indicamos los topics a los que queremos suscribirnos. El carácter `#` al final del topic `iescelia/#` indica que nos vamos a suscribir a cualquier topic que haya después de `iescelia/`.

③ Indicamos cuál es el formato de los datos que vamos a recibir por MQTT. En este caso indicamos el formato `influx`, por lo tanto, los mensajes que vamos a recibir por MQTT tienen que cumplir la sintaxis del protocolo InfluxDB line protocol.

Esta opción **no** será la opción que vamos a elegir para configurar la sección `inputs.mqtt_consumer` en nuestro proyecto.

Referencia:

- InfluxDB line protocol tutorial

## 5.6. Configuración del archivo `telegraf.conf` para almacenar los datos en InfluxDB (`outputs.influxdb`)

Tendremos que buscar la sección `outputs.influxdb` dentro del archivo `telegraf.conf` y configurar los siguientes valores:

- `urls`
- `database`
- `skip_database_creation`

- `username`
- `password`

Existen más directivas de configuración, pero en este proyecto sólo vamos a utilizar los valores que hemos indicado anteriormente.

### 5.6.1. `urls`

En esta directiva de configuración indicamos la URL de la instancia de InfluxDB donde vamos a almacenar los datos. En nuestro caso pondremos `influxdb` que es el nombre que le hemos puesto al servicio en el archivo `docker-compose.yml`

### 5.6.2. `database`

En esta directiva indicamos el nombre de la base de datos donde vamos a almacenar las métricas.

### 5.6.3. `skip_database_creation`

Permite evitar la creación de bases de datos en InfluxDB cuando está configurada como `true`. Se recomienda configurarla a `true` cuando estemos trabajando con usuarios que no tienen permisos para crear bases de datos o cuando la base de datos ya exista.

### 5.6.4. `username` y `password`

En esta directiva configuramos los parámetros de autenticación para conectarnos a InfluxDB.

### 5.6.5. Ejemplo de configuración de la sección `outputs.mqtt_consumer`

Una posible configuración de la sección `outputs.influxdb` podría ser la siguiente:

```
[[outputs.influxdb]]
  urls = ["http://influxdb:8086"] ①

  database = "iescelia_db" ②

  skip_database_creation = true ③

  username = "root" ④
  password = "root"
```

① Indica la URL de la instancia de InfluxDB. En nuestro caso indicamos el nombre `influxdb` que es el nombre que le hemos puesto al servicio en el archivo `docker-compose.yml`

② Indica el nombre de la base de datos donde vamos a almacenar las métricas.

③ Si está a `true` no se crearán bases de datos en InfluxDB. Se recomienda configurarlo a `true` cuando estemos trabajando con usuarios que no tienen permisos para crear bases de datos o cuando la base de datos ya exista.

④ Los parámetros de autenticación para conectarnos a InfluxDB.

**Esta será la opción que vamos a elegir** para configurar la sección `outputs.mqtt_consumer` en nuestro proyecto.

# 6. InfluxDB

InfluxDB es un sistema gestor de bases de datos diseñado para almacenar bases de datos de series temporales (TSBD - *Time Series Databases*). Estas bases de datos se suelen utilizar en aplicaciones de monitorización, donde es necesario almacenar y analizar grandes cantidades de datos con marcas de tiempo, como pueden ser datos de uso de cpu, uso memoria, datos de sensores de IoT, etc.

## 6.1. Descripción del servicio en `docker-compose.yml`

`docker-compose.yml`

```
influxdb: ①
  image: influxdb ②
  ports:
    - 8086:8086 ③
  volumes:
    - influxdb_data:/var/lib/influxdb ④
  environment:
    - INFLUXDB_DB=iescelia_db ⑤
    - INFLUXDB_ADMIN_USER=root ⑥
    - INFLUXDB_ADMIN_PASSWORD=root ⑦
    - INFLUXDB_HTTP_AUTH_ENABLED=true ⑧
```

① Nombre del servicio dentro del archivo `docker-compose.yml`.

② Utilizamos la imagen Docker influxdb.

③ Este servicio utilizará el puerto `8086` de nuestra máquina local para enlazarlo con el puerto `8086` el contenedor.

④ Creamos un volumen con el nombre `influxdb_data` que estará enlazado con el directorio `/var/lib/influxdb` del contenedor.

⑤ Nombre de la base de datos.

⑥ Usuario de la base de datos.

⑦ Contraseña del usuario de la base de datos.

⑧ Habilitamos la autenticación básica HTTP.

## 6.2. Conectar a InfluxDB desde un terminal

Podemos conectarnos al contenedor de InfluxDB desde un terminal para comprobar si los datos que estamos recogiendo de los sensores se están insertando de forma correcta en la base de datos.

En primer lugar necesitamos obtener el `ID` del contenedor que se está ejecutando con InfluxDB. Para obtener el listado de todos los contenedores que están en ejecución podemos ejecutar el siguiente comando:

```
docker ps
```

Ahora sólo tenemos que buscar el `ID` del contenedor con InfluxDB entre la lista de contenedores. En el ejemplo que se muestra a continuación sería el valor `27b06d552719`.

```
CONTAINER ID        IMAGE               COMMAND                  ...
27b06d552719        influxdb            "/entrypoint.sh infl…"   ...
...
```

Una vez que tenemos el `ID` del contenedor de InfluxDB, creamos un nuevo proceso con el comando `/bin/bash` dentro del contenedor para obtener un terminal que nos permita interactuar con el contenedor.

```
docker exec -it 27b06d552719 /bin/bash
```

Tenga en cuenta que en su caso tendrá que reemplazar el valor `27b06d552719` por el identificador de su contenedor.

Una vez que tenemos un terminal en el contenedor de InfluxDB, utilizamos el cliente `influx` para conectarnos al sistema gestor de bases de datos con el usuario y contraseña que hemos creado en el archivo `docker-compose.yml`.

```
influx -username root -password root
```

Después de autenticarnos, tendremos acceso al shell de InfluxDB donde podemos interaccionar con la base de datos con sentencias InfluxQL (Influx Query Language), que tienen una sintaxis muy similar a SQL.

A continuación, se muestra una secuencia de sentencias InfluxQL (Influx Query Language) que podemos utilizar para comprobar si los datos de los sensores se están almacenando en la base de datos.

**Consultar el listado de bases de datos.**

```
> show databases

name
----
iescelia_db
```

**Seleccionar la base de datos `iescelia_db`.**

```
> use iescelia_db
```

**Mostrar las tablas que existen para la base de datos seleccionada.**

```
> show measurements

name
----
cpu
disk
diskio
kernel
mem
mqtt_consumer
processes
swap
system
```

**Mostrar todos los datos que existen en la tabla `mqtt_consumer`.**

```
> select * from mqtt_consumer

time                host         topic                      value
----                ----         -----                      -----
1612791727450709588 3f4be32bd18b iescelia/aula22/co2        30
1612791734718611922 3f4be32bd18b iescelia/aula22/co2        30
```

**Mostrar los campos clave de cada tabla.**

```
> show field keys
```

# 7. Grafana

Grafana es un servicio web que nos permite visualizar en un panel de control los datos almacenados en InfluxDB y otros sistemas gestores de bases de datos de series temporales.

## 7.1. Descripción del servicio en `docker-compose.yml`

`docker-compose.yml`

```
grafana: ①
    image: grafana/grafana:7.4.0 ②
    ports:
      - 3000:3000 ③
    volumes:
      - grafana_data:/var/lib/grafana ④
    depends_on: ⑤
      - influxdb
```

① Nombre del servicio dentro del archivo `docker-compose.yml`.

② Utilizamos la imagen Docker grafana.

③ Este servicio utilizará el puerto `3000` de nuestra máquina local para enlazarlo con el puerto `3000` el contenedor.

④ Creamos un volumen con el nombre `grafana_data` que estará enlazado con el directorio `/var/lib/grafana` del contenedor.

⑤ Indicamos que este servicio depende del servicio `influxdb` y que no podrá iniciarse hasta que el servicio de `influxdb` se haya iniciado.

## 7.2. Configuración de un *data source*

Grafana permite utilizar diferentes *data sources*, en nuestro proyecto utilizaremos InfluxDB pero también es posible utilizar AWS CloudWatch, Elasticsearch, MySQL o PostgreSQl entre otros.

### 7.2.1. Configuración de forma manual

Antes de crear un *dashboard* es necesario crear un *data source*. Sólo los usuarios que tengan rol de administrador podrán crear un *data source*.

Para crear un *data source* debe seguir los siguientes pasos.

1. Accede al menu lateral haciendo clien en el icono de Grafana del encabezado superior.
2. Accede a la sección "Data Sources".
3. Añada un nuevo *data source*.
4. Seleccione InfluxDB en el desplegable donde se indica el tipo de *data source*.
5. Configure los parámetros `url`, `database`, `user` y `password` de su servicio InfluxDB.

En la documentación oficial puede encontrar más información sobre cómo utilizar InfluxDB con Grafana.

### 7.2.2. Configuración con aprovisionamiento automático

Es posible configurar Grafana para que utilize un archivo de configuración de forma automática para evitar tener que realizar la configuración de forma manual.

En nuestro proyecto, vamos a crear un archivo con el nombre `datasource.yml` dentro de la siguiente estructura de directorios.

```
├──── grafana-provisioning
│     └──── datasources
│           └──── datasource.yml
```

El contenido del archivo `datasource.yml` será el siguiente.

```
apiVersion: 1
datasources:
  - name: InfluxDB
    type: influxdb
    access: proxy
    database: iescelia_db ①
    user: root ②
    password: root ③
    url: http://influxdb:8086 ④
    isDefault: true
    editable: true
```

① Nombre de la base de datos InfluxDB

② Nombre de usuario para acceder a la base de datos

③ Contraseña del usuario para acceder a la base de datos

④ URL del servicio InfluxDB

Para que el aprovisionamiento se realice forma correcta, tenemos que definir un nuevo volumen en el servicio de Grafana en el archivo `docker-compose.yml`.

**`docker-compose.yml`**

```
grafana: ①
    image: grafana/grafana:7.4.0 ②
    ports:
      - 3000:3000 ③
    volumes:
      - grafana_data:/var/lib/grafana ④
      - ./grafana-provisioning/:/etc/grafana/provisioning ⑤
    depends_on: ⑥
      - influxdb
```

① Nombre del servicio dentro del archivo `docker-compose.yml`.

② Utilizamos la imagen Docker grafana.

③ Este servicio utilizará el puerto `3000` de nuestra máquina local para enlazarlo con el puerto `3000` el contenedor.

④ Creamos un volumen con el nombre `grafana_data` que estará enlazado con el directorio `/var/lib/grafana` del contenedor.

⑤ Creamos un volumen de tipo *bind mount* entre el directorio local de nuestra máquina `./grafana-provisioning/` y el directorio `/etc/grafana/provisioning` del contenedor.

⑥ Indicamos que este servicio depende del servicio `influxdb` y que no podrá iniciarse hasta que el servicio de `influxdb` se haya iniciado.

En la documentación oficial puede encontrar más información sobre https://grafana.com/docs/grafana/latest/datasources/influxdb/#configure-the-data-source-with-provisioning [cómo configurar un *data source* con aprovisionamiento].

# 7.3. Configuración de un *dashboard*

## 7.3.1. Configuración con aprovisionamiento automático

Es posible configurar Grafana para que utilize un archivo de configuración de forma automática para evitar tener que realizar la configuración del *dashboard* de forma manual.

En nuestro proyecto, vamos a crear los archivos `dashboard.yml` y `CO2Dashboard.json` dentro de la siguiente estructura de directorios.

```
├──── grafana-provisioning
│    ├──── dashboards
│    │    ├──── CO2Dashboard.json
│    │    └──── dashboard.yml
│    └──── datasources
│         └──── datasource.yml
```

El contenido del archivo `dashboard.yml` será el siguiente.

```
apiVersion: 1
providers:
- name: InfluxDB
  folder: ''
  type: file
  disableDeletion: false
  editable: true
  options:
    path: /etc/grafana/provisioning/dashboards
```

Un *dashboard* de Grafana se representa por un objeto JSON, que almacena todos los metadatos del *dashboard*. Estos metadatos incluyen propiedades de los paneles, variables, etc. El archivo `CO2Dashboard.json` contiene los metadatos del los paneles que hemos creado para el *dashboard* de este proyecto.

A continuación se muestra una imagen del *dashboard* almacenado en el archivo `CO2Dashboard.json`.


*Figure 5. Ejemplo de un dashboard de Grafana mostrando valores de CO2 y TVOC.*

En la documentación oficial puede encontrar más información sobre cómo configurar un *dashboard* en Grafana.

# 8. Repositorio en GitHub

El código fuente del proyecto está disponible en GitHub en el siguiente repositorio.

https://github.com/josejuansanchez/co2-celia.

# 9. Autor

Este material ha sido desarrollado por José Juan Sánchez.

# 10. Licencia

# 11. Referencias

- Proyecto de IoT sencillo con Wemos D1 mini, sensores DHT11, MQTT y un panel de control web con node.js.

```
<script src="https://josejuansanchez.org/javascripts/scale.fix.js"></script>
<script type="text/javascript">
    var gaJsHost = (("https:" == document.location.protocol) ? "https://ssl." : "http://www.");
    document.write(unescape("%3Cscript src='" + gaJsHost + "google-analytics.com/ga.js'
type='text/javascript'%3E%3C/script%3E"));
</script>
<script type="text/javascript">
    try {
        var pageTracker = _gat._getTracker("UA-38421373-1");
        pageTracker._trackPageview();
    } catch(err) {}
</script>
```