

Una solución que combina una ESP32 como publicador, un Servidor Mosquito y un fluograma Node-RED

Tarea 1. Identificación del problema a resolver

Implementar un sistema IoT que consta de un sensor de temperatura conectado a un tarjeta ESP32, la cual envia a Internet los datos por MQTT, con un tópico llamado “canal”, hacia un broker Mosquitto ubicado en una VM en AWS identificada por una dirección IP. El sistema deberá incluir una interfaz gráfica para la visualización de resultados.

Tarea 2. Trabajo previos. Repaso del dia a dia

Lo siguiente son cosas que se han aprendido en prácticas pasadas. Quizá sea redundante explicar esto aquí, pero quizá también resulte útil un pequeño repaso de lo que se hace día a día cuando se trabaja en soluciones que combinan ESP32 con MQTT. Es que ya se había creado una solución de ESP-32 con MQTT cuando se usó EMQX. La novedad es que ahora usaremos Mosquitto en lugar de EMQX y además involucramos Node-RED para lograr una visualización interesante.

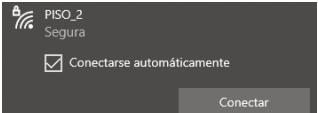
Es necesario tener muy claros los conceptos de:

- Broker
- Mosquitto
- Publicador
- Suscriptor
- Tópico
- JSON
- Arduino IDE
- ESP32

Encender la VM:

- <https://aws.amazon.com/> > Sign in > EC2 > Menu lateral > Instances > chequeas la instancia de interés > Menu horizontal > Instance state > start instance > constatamos que la VM quede corriendo
- Si se desea contar con una terminal de comandos continuar con: Menú horizontal > Connect > Connect
- Nota: En la VM recursos como Mosquito, Node-RED están programados para arrancar automáticamente con el encendido de la VM

Datos de inicio	
Descripción	El dato

Dirección IP pública de la VM	172.171.224.96
URL del fluograma, ejemplo: http://54.236.56.149:1880	http://172.171.224.96:1880
URL de la Interfaz gráfica, ejemplo: http://54.236.56.149:1880/ui	http://172.171.224.96:1880/ui/#/0?socketid=QIY-j2PZ2siZ_YtTAAAB
Login de la WiFi	
Clave de la WiFi	Planetaland2

La ESP32. Actualización del software embebido:

- Si la VM estaba apagada, con el nuevo encendido habrá cambiado la IP pública y es lo que nos interesa actualizar
- Es necesario localizar el software hecho, en prácticas anteriores, en la Arduino IDE para el ESP32, realizar allí los cambios para la IP pública.
- Conectar la ESP32 al computador que tiene la Arduino IDE
- Bajar el software actualizado del Arduino IDE a la ESP32
- Alimentar la ESP32:
 - Si la dejamos conectada al computador ya tendrá alimentación y además podrá usar la Arduino IDE para mostrar datos de printouts
 - Se puede independizar del computador si se le conecta a una batería u otra fuente de alimentación.

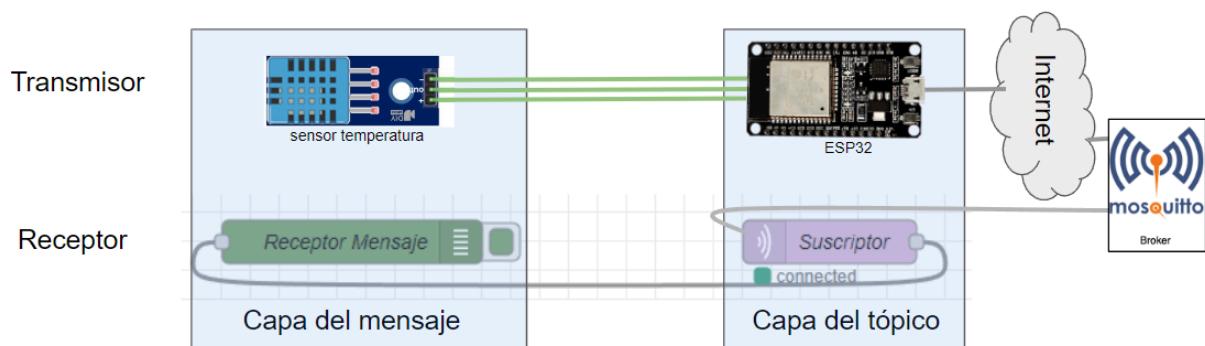
Abrir Node Red:

- Use la URL identificada en la tabla anterior.

Tarea 3. Diseño

A continuación mostramos la tarea de diseño hecha para que sirva de ejemplo para los futuros diseños que deberá hacer el mismo estudiante.

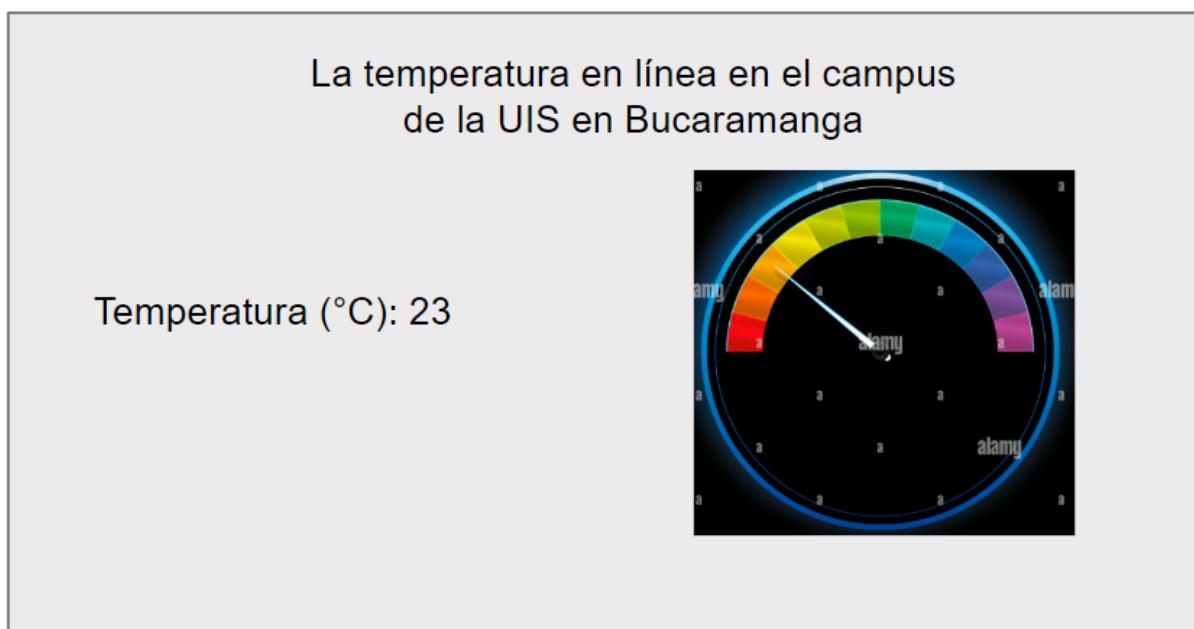
Paso 1. El diseño a nivel de red se puede representar mediante el siguiente fluograma
 En este paso, solo vamos a imaginar cómo será la solución a nivel de nodos y preferiblemente como un modelo de capas, de manera que resulte fácil comprender hacia dónde vamos, dónde hay que enfocar los esfuerzos.



- A diferencia de la práctica anterior, la ESP-32 juega el papel de publicador, un sensor de temperatura juega el papel del generador del mensaje. Adicionalmente, están en un lugar remoto si se compara con la ubicación del suscriptor que está en una instancia EC2 AWS.
- Los demás elementos están en un solo lugar, que es la instancia EC2 AWS
- En la capa “tópico” vemos un suscriptor, se deduce que la ESP32 debe estar programada como publicador y que los datos que entrega la ESP32 son los que llegan al Suscriptor gracias a la intermediación de Mosquitto.
- El publicador y el suscriptor deben estar programados para usar el mismo tópico, que en nuestro caso se llama “canal”
- La ESP32 debe estar bien configurada para estar conectada a Internet, en nuestro caso será por una WIFI que tiene un usuario y una contraseña
- El “Receptor Mensaje” debe estar configurado para que muestre los datos de temperatura medidos

Paso 2. Diseño de la interfaz gráfica

La idea de este paso es imaginar cómo podría ser la interfaz de usuario más apropiada. Para el caso dado, al usuario solo le interesa visualizar la temperatura, de manera que la interfaz puede ser la siguiente:



Tarea 4. Implementación del publicador basado en ESP-32 con sensor de temperatura.

Consideraciones previas

- El Broker Mosquitto ya existe gracias a lo realizado en la práctica anterior.
- No nos interesa que el Broker tenga publicadores ya que el publicador será la ESP32 con su programa embebido.

Paso 1. El código para el publicador

Tenemos un código sugerido por el Chat GPT y usado en una práctica anterior donde se usó en una ESP-32. Debemos revisar los parámetros que corresponden a la dirección IP de la instancia EC2 AWS, el usuario y contraseña de la WiFi, que corresponden a los parámetros:

- **server:** es la IP de la instancia EC2 AWS donde se encuentra el broker (el servidor Mosquito)
- **ssid:** es el nombre de la red WiFi
- **password:** es el password para que la ESP se conecte a la red WiFi

```
// Librerías a usar
#include "WiFi.h"
#include "PubSubClient.h"

// Resolución deseada en las mediciones
#define ADC_VREF_mV 3300.0
#define ADC_RESOLUTION 4096.0

// Datos de la WiFi
WiFiClient esp32Client;
const char* ssid = "Rg";
const char* password = "e3te3te3t";

// Datos Broker y clientes
PubSubClient mqttClient(esp32Client);
char* server = "100.26.173.15"; // La IP pública de la VM en AWS
int port = 1883; // el puerto que tiene Mosquitto asignado, por defecto 1883
char datos[40]; // variable usada para los valores de las mediciones

/* Variables que no se usan
int var = 0;
int ledval = 0;
int fotoval = 0;
String resultS = "";
*/

// Datos de la ESP32
int temp = 33; // el pin al cual se conecta el sensor
int led = 2; // el led 2 (azul) será usado para mostrar que la WiFi está conectada
int botón = 0; // el botón que usaremos para resetear la tarjeta

//-----
void wifi() {
```

```

Serial.print("Conectandose a WiFi: ");
Serial.println(ssid); // el nombre del usuario de WiFi

WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) { //mostrar panticos mientras no logra conexion
    Serial.print(".");
    delay(500);
}
digitalWrite(led, HIGH);

//Serial.println("");
Serial.println("\nConectado a WIFI");
Serial.print("Dirección IP ");
Serial.println(WiFi.localIP());
}

-----
// si se pierde la conexion MQTT esta función será usada para reconectarse
void reconnect() {
    while (!mqttClient.connected()) {
        Serial.print("Conexion MQTT perdida, intentando conectarse ...");

        if (mqttClient.connect("esp32")) {
            Serial.println("Conectado");
        } else {
            Serial.print("Fallo, rc=");
            Serial.print(mqttClient.state());
            Serial.println(" intentar de nuevo en 5 segundos");
            delay(5000); // espera de 5 segundos
        }
    }
}

void setup() {
    // Inicialización de los pines
    pinMode(led, OUTPUT); // el pin del LED se configura como una salida
    pinMode(boton, INPUT); // el pin botón como una entrada
    Serial.begin(115200); // abrir el monitor serial del computador a esa rata
    delay(10);
    wifi(); // ordena la conexión a wifi
    mqttClient.setServer(server,port); // ordena la conexión al broker
}

-----
void loop() {
    // Si se pierde la conexión WiFi, recuperarla
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("Conexion WiFi perdida. Intentando reconectar...");
        digitalWrite(led, LOW);
        wifi(); // intentar restablecer la conexión
    }
    // Si se pierde la conexión con el servidor, recuperarla
    if (!mqttClient.connected()) {
        reconnect();
    }
    // Captar los datos del sensor de temperatura
}

```

```

float temperatura = (analogRead(temp) * (ADC_VREF_mV / ADC_RESOLUTION)) / 10.0 + 8;
Serial.print("Temperatura: ");
Serial.println(temperatura);
sprintf(datos, "% .2f", temperatura);
// publicar los datos de temperatura al broker
mqttClient.publish("canal", datos);
delay(3000);
}

```

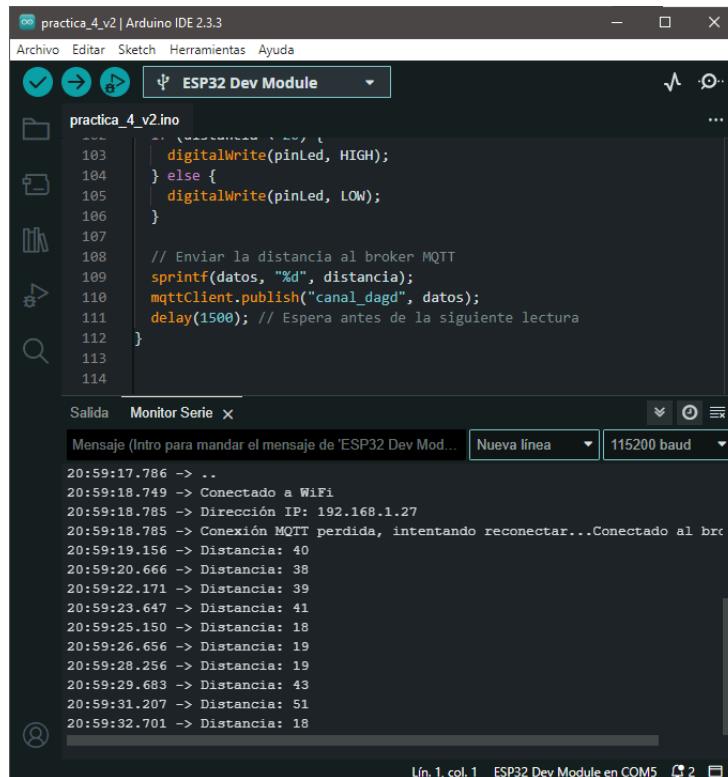
Paso 2. Se actualiza la IP y la WiFi y se compila el código en el IDE Arduino

Paso 4. Se baja el código al Arduino

Paso 5. En la terminal donde se creó el suscriptor en la VM revisamos los datos que llegan

Primero aclaro que no dispongo de un sensor de temperatura para hacer la prueba, por tal motivo y ya que dispongo de un sensor **HC-SR04 (sensor de distancia)**, por ese motivo modifique el código para que me funcione para el sensor que tengo disponible, además de agregar un led para indicar conexión a Wi-Fi y otro led para indicar conexión a MQTT, cuando el sensor detecte una distancia menor a 20 cm, se encenderá un led que indica que hay algo a menos de 20 cm del sensor.

Primero muestro la evidencia de que se cargan los datos en el “monitor serie” del IDE de Arduino:



Ahora, en la parte superior izquierda se observa que se está ejecutando **Node-RED**, en la parte inferior izquierda se observa el programa en ejecución mostrando datos en el monitor serie del IDE de Arduino, en la parte superior derecha se observa una terminal que contiene el **estado de Mosquitto**, finalmente en la parte inferior derecha se observa una terminal con el **Suscriptor** previamente conectado al tópico **canal_dagd** y es donde se muestran también las mediciones del sensor:

```

dagd_User@dagd: ~
4 Nov 01:55:54 - [info] Stopping flows
4 Nov 01:55:54 - [info] [mqtt-broker:d96d3ab6c89c1a3f] Disconnected from broker: http://172.171.224.96:1883
4 Nov 01:55:54 - [info] Stopped flows
4 Nov 01:55:54 - [info] Updated flows
4 Nov 01:55:54 - [info] Starting flows
4 Nov 01:55:54 - [error] [mqtt in:Subscriber] missing broker configuration
4 Nov 01:55:54 - [info] Started flows
4 Nov 01:56:21 - [info] Stopping flows
4 Nov 01:56:21 - [info] Stopped flows
4 Nov 01:56:21 - [info] Updated flows
4 Nov 01:56:21 - [info] Starting flows
4 Nov 01:56:21 - [info] Started flows
4 Nov 01:56:21 - [info] [mqtt-broker:d96d3ab6c89c1a3f] Connected to broker: http://172.171.224.96:1883
4 Nov 01:56:42 - [info] [mqtt-broker:d96d3ab6c89c1a3f] Disconnected from broker: http://172.171.224.96:1883
4 Nov 01:56:57 - [info] [mqtt-broker:d96d3ab6c89c1a3f] Connected to broker: http://172.171.224.96:1883
555

dagd_User@dagd: ~
mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/usr/lib/systemd/system/mosquitto.service; enabled; preset: )
   Active: active (running) since Mon 2024-11-04 01:56:55 UTC; 6min ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Process: 4655 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited)
  Process: 4656 ExecStartPre=/bin/chown mosquitto:mosquitto /var/log/mosquitto (code=exited)
  Process: 4660 ExecStartPre=/bin/chown mosquitto:mosquitto /run/mosquitto (code=exited)
 Main PID: 4663 (mosquitto)
    Tasks: 1 (limit: 9458)
      Memory: 1.2M (peak: 1.4M)
        CPU: 159ms
      CGroup: /system.slice/mosquitto.service
              └─ 4663 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Nov 04 01:56:55 dagd systemd[1]: Started mosquitto.service - Mosquitto MQTT Broker.
Nov 04 01:56:55 dagd mosquitto[4663]: 17306885415: New connection from 127.0.0.1:5222
Nov 04 01:56:55 dagd mosquitto[4663]: 17306885415: New client connected from 127.0.0.1:5222
lines 1-19

```

Salida Monitor Serie x

Mensaje (Intro para mandar el mensaje de ESP32 Dev Module Nueva linea 115200 baud)

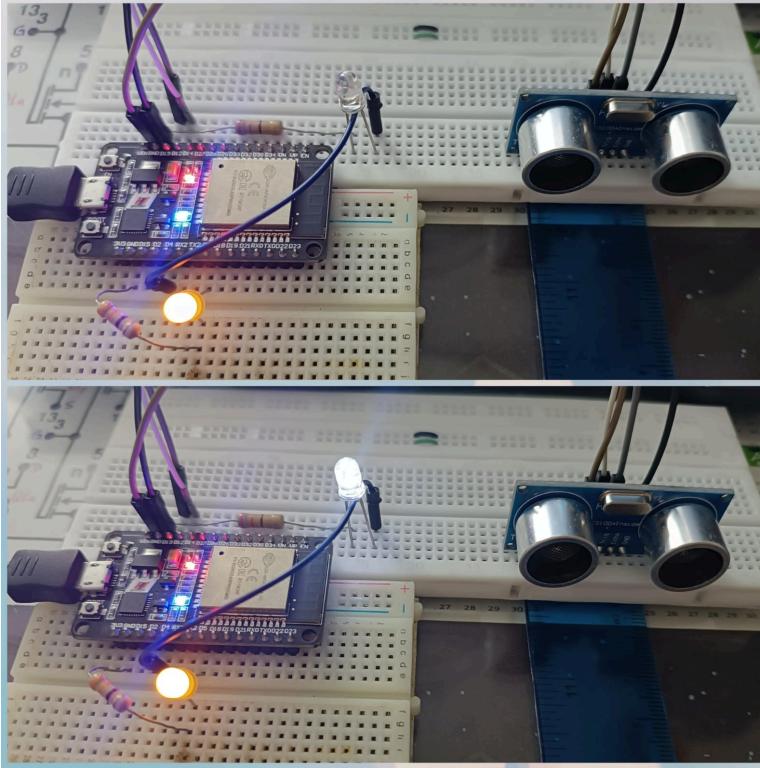
```

21:07:31.925 -> Distancia: 36
21:07:33.463 -> Distancia: 37
21:07:34.970 -> Distancia: 20
21:07:36.433 -> Distancia: 19
21:07:37.936 -> Distancia: 21
21:07:39.487 -> Distancia: 38
21:07:40.975 -> Distancia: 34
21:07:42.471 -> Distancia: 36
21:07:43.953 -> Distancia: 40
21:07:45.486 -> Distancia: 17
21:07:46.979 -> Distancia: 17
21:07:48.478 -> Distancia: 17
21:07:50.000 -> Distancia: 18
21:07:51.492 -> Distancia: 18

```

Lín. 1, col. 1 ESP32 Dev Module en COM5 2

Finalmente se muestra una imagen donde se evidencia el uso de la ESP32 y el sensor HC-SR04 en lugar del sensor de temperatura:



Explicación: El led encendido de color rojo, indica que esta conectada y alimentada la ESP32, el led encendido de color azul indica la conexión correcta a Wi-Fi, el led encendido de color amarillo indica la conexión correcta a MQTT, finalmente el led color blanco indica cuando el led esté encendido que la distancia medida por el sensor es menor a 20 cm, de lo contrario el led de color blanco estará apagado si esa distancia es igual o mayor a 20 cm.

Tarea 5. Implementación del primer Sprint: el sistema completo funcionando aunque sin interfaz gráfica

Consideraciones previas sobre Node-RED:

- Para los que saben que es GNU Radio, Node-RED es a Node.js lo que GRC es a GNU Radio. Es decir, Node-RED es una manera gráfica de crear soluciones con Node.js. Node.js es a su vez muy similar a Google Apps Script en el sentido de que permite crear WebApps, Web Services, Web Hooks y demás servicios web
- Para los que no conocen qué es GNU Radio, Node-RED es una herramienta de programación visual enfocada a soluciones IoT. Eso significa que está pensada en programación sin código. Aunque eso es relativo, porque si necesitas un bloque que no existe en la herramienta, pues no queda otra que implementarlo con código.
- Según las consultas hechas al momento de escribir esto, no está hecha exclusivamente para explotar las capacidades de Mosquitto, sino de una amplia variedad de Brokers. En general está hecha para crear cualquier servicio web.
- Cuando Node-RED se instala en un servidor tiene su máxima potencia para aprovechar múltiples fuentes IoT. Imagina lo siguiente: en el mundo hay muchos sensores midiendo y enviando datos por protocolo MQTT marcados con unos tópicos determinados, algunos con usuario y contraseña, otros simplemente libres. Pues bien, contando con la información y los permisos adecuados, Node-RED puede suscribirse a ellos. Ahora viene la pregunta ¿qué hacer con esos datos? Obviamente no queremos que nuestra solución se suscriba a todo, sino a aquello que tiene sentido para las necesidades que deseamos satisfacer, por ejemplo conocer parámetros útiles para controlar el cultivo de nuestra finca. Lo primero que se nos ocurre hacer con los datos es - visualizarlos. Por eso muchos consideran que Node-RED es una herramienta de visualización. En realidad podemos hacer más cosas como: procesar esos datos, almacenarlos, generar datos para activar remotamente cosas.
- Dicho en otras palabras, Node-RED permite crear suscriptores y publicadores para complementar los que existen, pero también otros bloques para dar como resultado un servicio útil para usuarios de interés.

Paso 1. Implementación de la solución

En red node creamos el flujo correspondiente. Como el transmisor es remoto, solo nos ocuparemos del receptor. En el flujo no aparece Mosquitto, porque eso es algo que se sobreentiende que está y que se accede por la configuración que tiene el suscriptor.

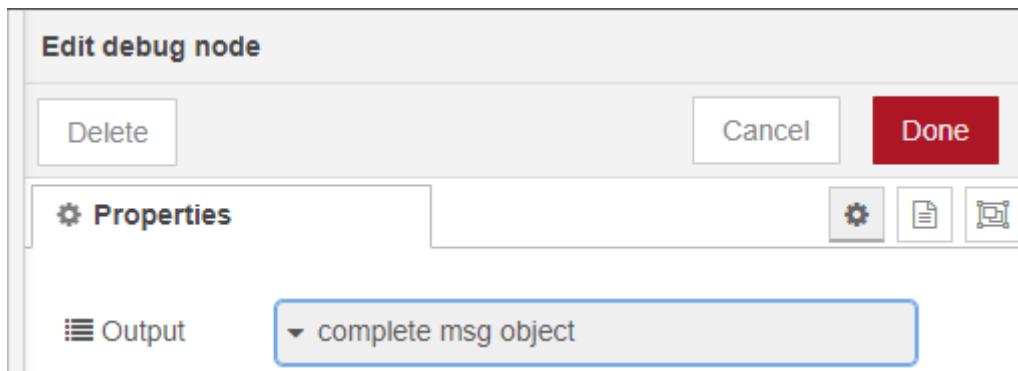


Configuraciones:

El suscriptor

<input type="radio"/> Server	localhost:1883	<input type="button" value="edit"/>
Action	Subscribe to single topic	
<input type="radio"/> Topic	canal	
<input checked="" type="radio"/> QoS	2	<input type="button" value="down"/>
<input type="radio"/> Output	auto-detect (parsed JSON object, string or buf)	
<input type="radio"/> Name	Suscriptor	

Configuramos el Receptor Mensaje (bloque debug) para que muestre todo el objeto, es decir, no solo los valores de medición, sino todo lo que viene en formato JSON. Esto con el fin de conocer con qué parámetros contaremos para la futura fase que será la interfaz gráfica

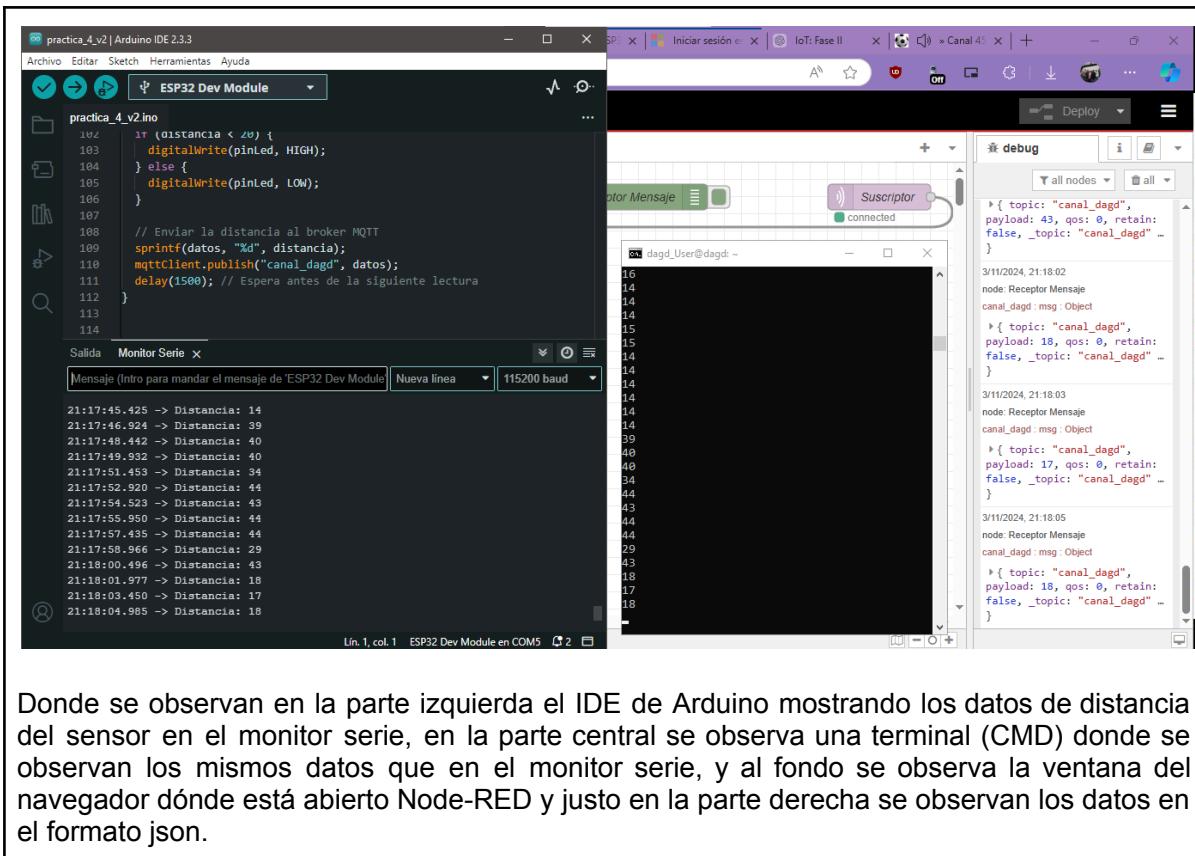


Veremos resultados como los siguientes

```
3/1/2023, 8:48:27 AM node: debug 1
canal : msg : Object
▶ { topic: "canal", payload: 23.47,
qos: 0, retain: false, _topic:
"canal" ... }

3/1/2023, 8:48:30 AM node: debug 1
canal : msg : Object
▶ { topic: "canal", payload: 23.63,
qos: 0, retain: false, _topic:
"canal" ... }
```

La evidencia para esta parte se puede observar en la siguiente imagen:

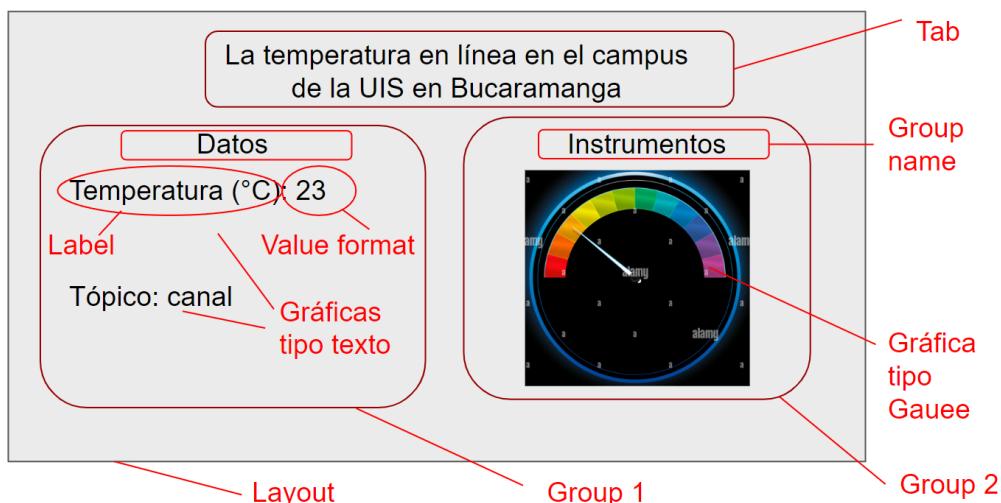


Donde se observan en la parte izquierda el IDE de Arduino mostrando los datos de distancia del sensor en el monitor serie, en la parte central se observa una terminal (CMD) donde se observan los mismos datos que en el monitor serie, y al fondo se observa la ventana del navegador dónde está abierto Node-RED y justo en la parte derecha se observan los datos en el formato json.

Tarea 6. Implementación de la interfaz gráfica

Consideraciones previas

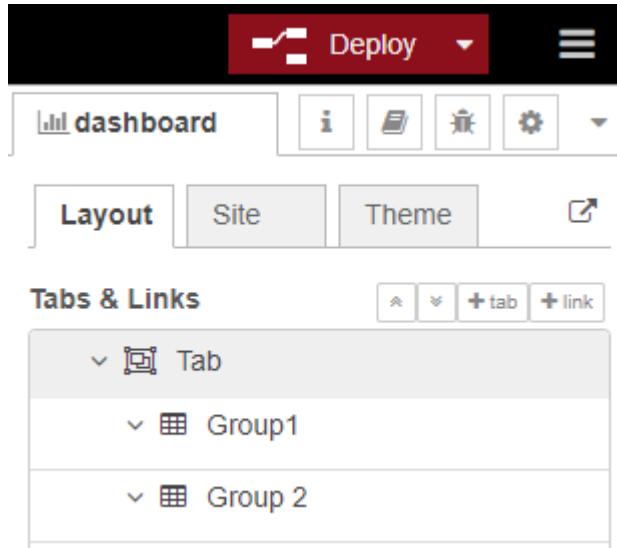
- Node-RED incluye plantillas para gráficas
- Es necesario conocer los parámetros que tiene cada uno de los campos de la plantilla con el fin de lograr configurarlos. En la siguiente figura se detallan para el tipo de interfaz gráfica que buscamos



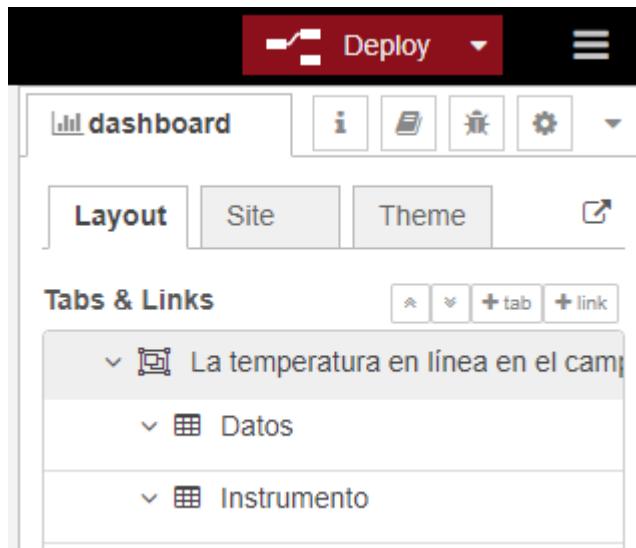
- Para entender mejor estos conceptos veámoslos como objetos así:
 - Un Dashboard es el conjunto de todas las herramientas de visualización incluidas la interfaz gráfica, la página web, etc. Una Dashboard tiene un Layout

- Un Layout es la estructura de la interfaz gráfica. Un Layout tiene un Tap
- Un Tap es algo así como el espacio en el layout reservado para el título o presentación mayor. Un Tap tiene grupos
- Un grupo es un espacio de pantalla destinado para mostrar información o datos de interés para los usuarios. Un grupo puede tener distintos tipos de instrumentos de medición como: text, gauge, etc.
- Creamos la estructura del layout con un Tab y dos grupos:

Menu lateral derecho > Dashboard > Layout

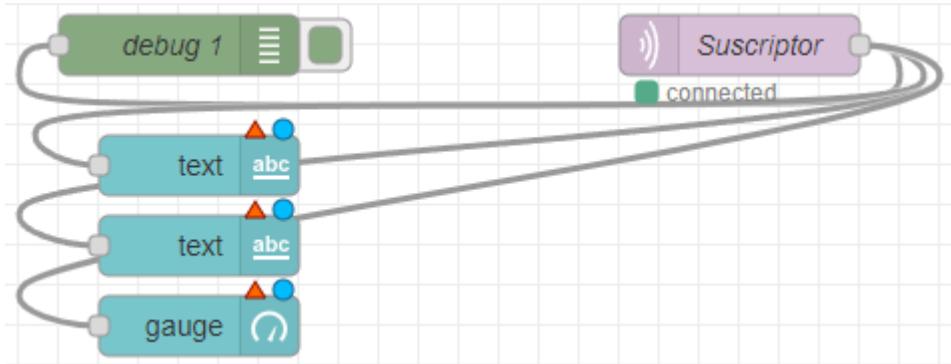


Le asignamos nombres a ese tab y grupos



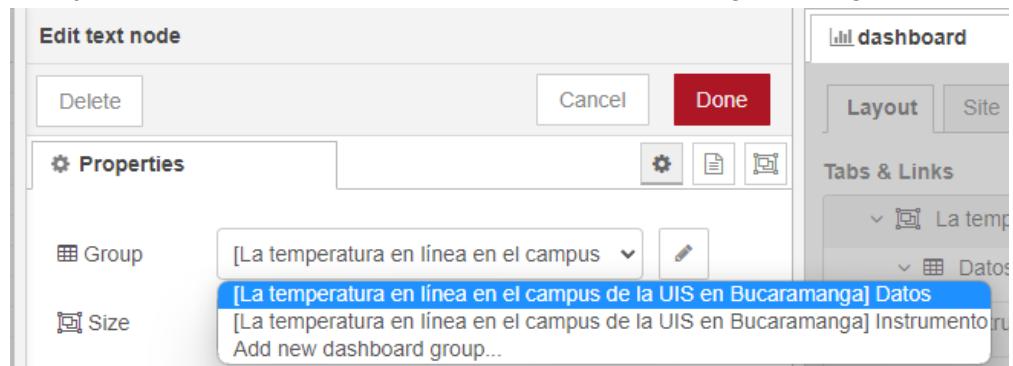
- Agregamos bloques de visualización (sumideros). De la interfaz deseada, la vista en un paso 2 de la tarea 3, vemos la necesidad de usar 3 visualizadores:
 - 2 de tipo texto
 - 1 de tipo gauge

Por lo tanto a nivel de flujogramas completamos el diseño así

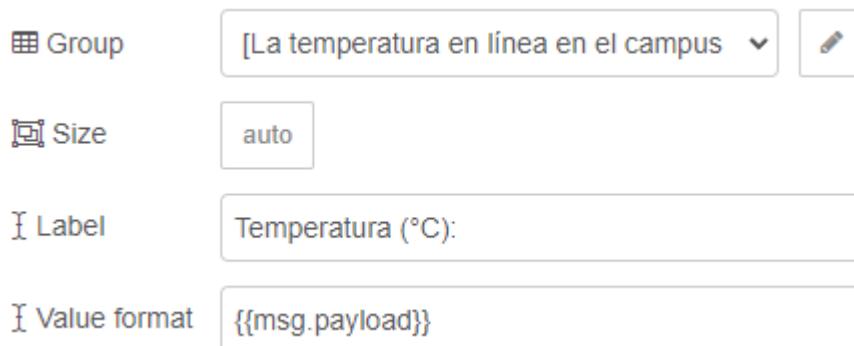


- Abrimos cada bloque y lo configuramos para que pertenezcan al grupo previsto y para que muestre los datos previstos

- Por ejemplo el caso el primer text, comenzamos por asignarle el grupo

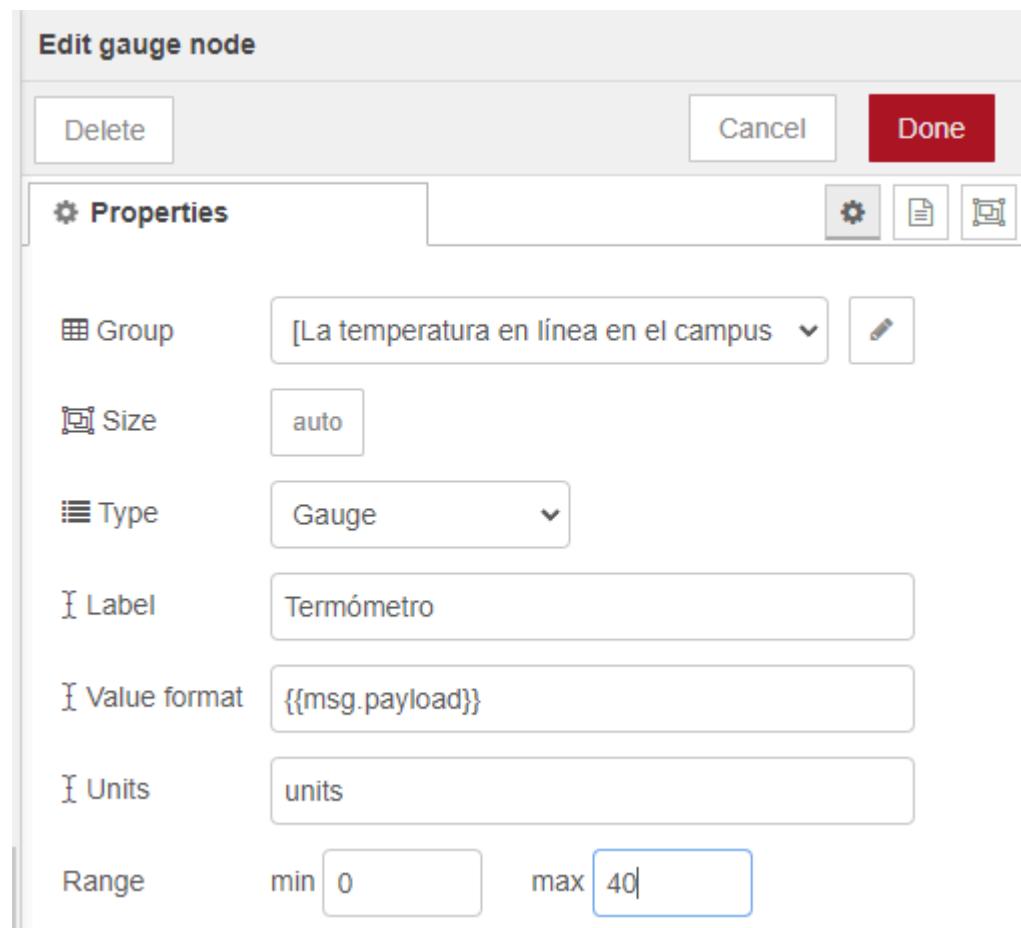


- Configuramos allí mismo el Label y el Value format, que en este caso sería la carga útil, la cual se extrae el objeto msg así: msg.payload

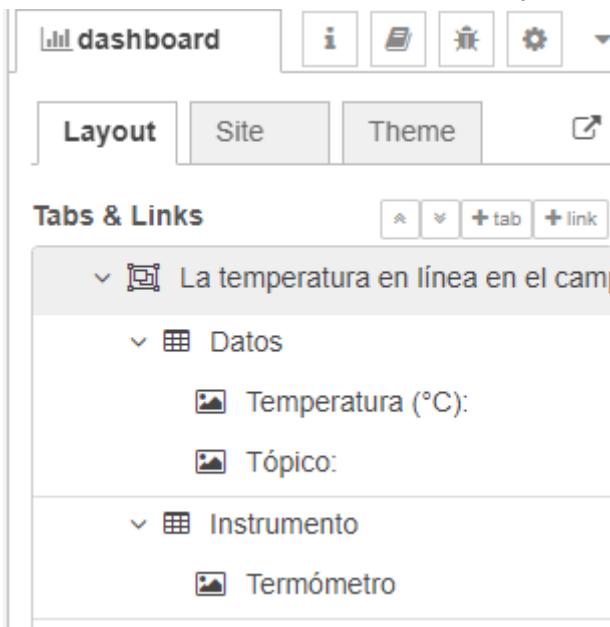


guardar con botón “Done”

- La siguiente es la configuración para el bloque gauge



- Si consultamos ahora el Dashboard nuevamente veremos ya la estructura final así



- Pruebas a la interfaz gráfica
guardamos la implementación con “Deploy”. vemos un resultado como el siguiente

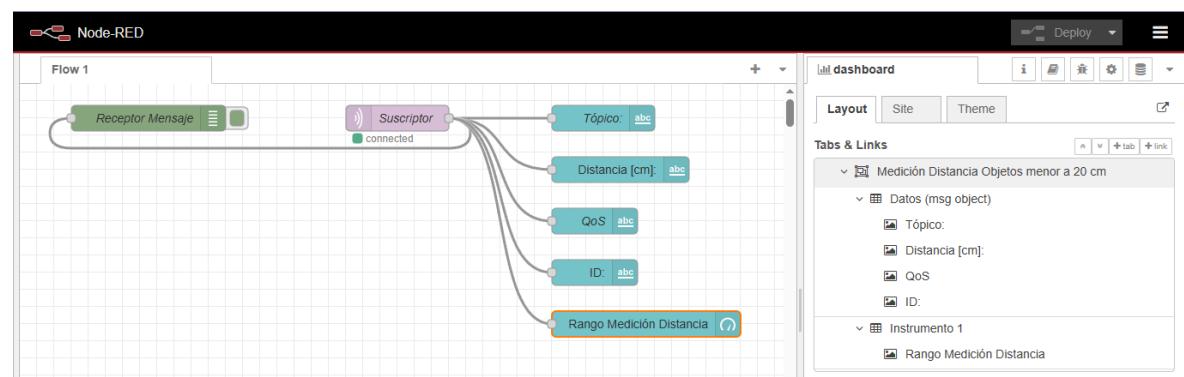


- Pruebas a la interfaz gráfica
Pruebe todas las posibles opciones que ofrece Dashboard para mejorar la visualización: Menú lateral derecho ▾ Dashboard > revisar herramientas de configuración que tiene cada cosa.

Para el informe: Evidencias

OJO: Para estas evidencias es necesario que el estudiante haga una implementación ligeramente diferente, al menos en la parte gráfica. Por ejemplo, una distribución diferentes de la cosas a visualizar

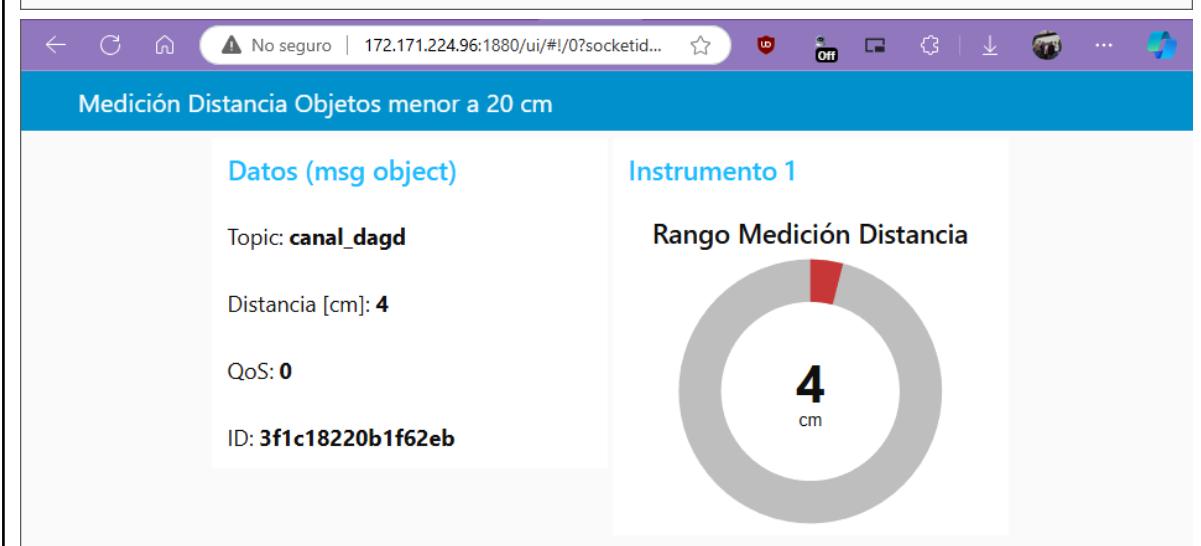
Tome captura de pantalla del flujo-node implementado



Tome captura de pantalla para la visualización obtenida

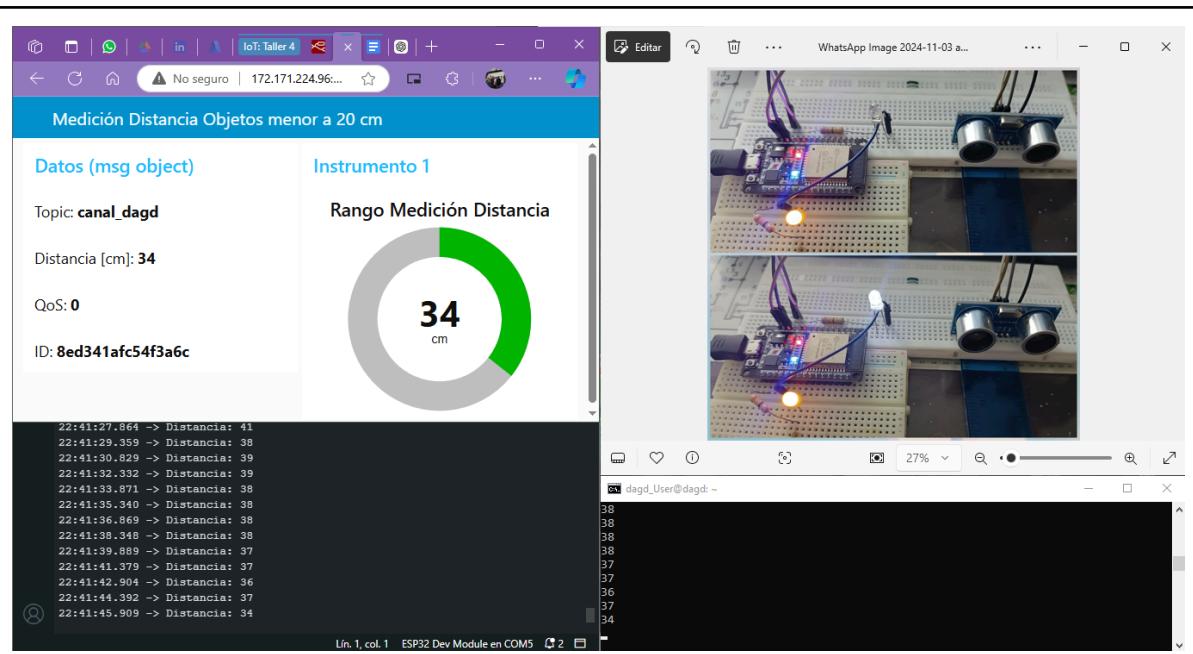




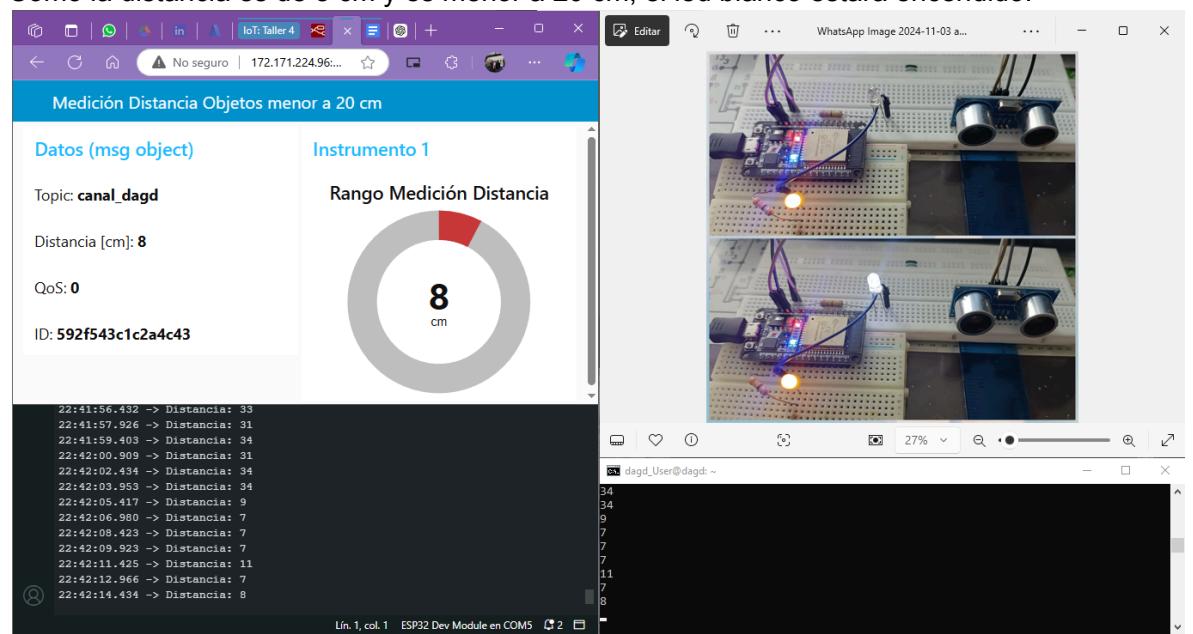


Agregue una foto del trabajo realizado con la ESP-32

Como la distancia es de 34 cm y es mayor a 20 cm, el led blanco estará apagado:



Como la distancia es de 8 cm y es menor a 20 cm, el led blanco estará encendido:



Finalmente en las dos imágenes se observa que los datos en el monitor serie del IDE de Arduino son los mismos datos que se observan en la terminal (CMD) que vienen siendo los datos del suscriptor, es decir los datos provenientes del sensor.