

LAB 3: MQTT

La presente evaluación será recibida sólo si se ha cumplido con los checkpoints de todas las preguntas en el **LABORATORIO** con el docente del curso.

La entrega debe constar de:

1. **Introducción** (incluir párrafo introductorio):
 1. **Marco Teórico** (definiciones **con referencias**)
 2. **Estado del Arte** (trabajos de investigación, artículos, tesis, páginas **con referencias**).
2. **Metodología** (explicación de componentes e implementos utilizados y diagramas de flujo/bloques).
3. **Desarrollo de la experiencia** (incluye discusión)
4. **Conclusiones** (una por cada experiencia realizada **como mínimo**)
5. **Referencias** (formato IEEE)

Consideraciones:

- Subir únicamente el PDF del informe a Canvas, usar un link para el repositorio, incluir fotos de paso a paso de cada implementación y un vídeo mostrando el funcionamiento de las experiencias. Sin embargo, para simulaciones, considere la placa Arduino UNO disponible en Tinkercad.
- Considere una placa de Arduino MEGA2560 tal cual existe en el laboratorio.
- Solo un miembro del grupo sube el informe.
- No olvidar colocar los nombres de todos los integrantes.

ÍNDICE

1. Instalación del controlador para ESP32.	5
1.1. Instrucciones	5
1.2. Responder	5
2. Implementación Básica	5
2.1. Indicaciones	5
2.2. Requisitos	5
2.3. Presentar	6
4. ANEXO 1. Sensor de pulso cardiaco	7
4.1. Funcionamiento	7
4.2. Especificaciones	7
4.3. Ejemplo básico de lectura con Arduino	8
5. ANEXO 2. Joystick	9
5.1. Especificaciones	9
5.2. Conexiones	9
5.3. Enlaces	10
6. ANEXO 3. Micro Servo	11
6.1. Especificaciones	11
6.2. Uso con Arduino	12
6.3. Librería servo de Arduino	13
6.4. Ejemplo	14
6.5. Referencia	16
7. ANEXO 2. Qué es MQTT	17
8. ANEXO 3. ESP32	17
8.1. Uso del ESP32 con MQTT.	18
9. ANEXO 4. Comunicación UART (Serial) entre dos dispositivos	18
9.1. Envío de datos en Serie	19
9.2. Tasa de Baudios	20
9.3. Interfaz UART en Arduino	20
9.4. Niveles lógicos (Importante)	20
9.5. Conexiones básicas (Serial1):	20
9.6. Tutorial de Referencia	21
10. ANEXO 5. Comunicación entre Arduino (5V) y un dispositivo de 3.3V	21
10.1. Consideraciones	21
10.2. Opciones para adaptación de nivel:	21
11. ANEXO 6. Instalación de placas ESP32 en Arduino	22
11.1. Adicionar las URLs para placas ESP32	22
11.2. Instalar core y placa ESP32	23
12. ANEXO 7. Cómo programar un ESP32 con el IDE Arduino	25
13. ANEXO 8. Troubleshooting	26
13.1. Mosquitto Installing	28

{cwilliams,garias}@utec.edu.pe

CS5055: Internet of Things

16 de mayo del 2025

MQTT

Introducción

MQTT (Message Queuing Telemetry Transport) es un protocolo ligero de mensajería diseñado para dispositivos con recursos limitados y redes poco confiables. Opera sobre TCP/IP y se basa en un modelo publicador/suscriptor, donde un broker (como Mosquitto) se encarga de recibir, filtrar y distribuir los mensajes entre los dispositivos conectados.

A. ¿Cómo lo usaremos?

En nuestro esquema de comunicación:

- **Mosquitto** actuará como el **broker MQTT**, corriendo en una PC, Raspberry Pi o servidor local.
- Los módulos **ESP32 o ESP8266** funcionarán como **clientes MQTT**, conectándose vía Wi-Fi al broker para **publicar o suscribirse** a temas específicos (topics).
- El **Arduino MEGA**, que no tiene conectividad Wi-Fi nativa, se comunicará con un ESP32/ESP8266 vía **UART (serial)**, y este módulo actuará como puente entre el MEGA y el broker.

B. Flujo básico de comunicación:

1. El **ESP32/8266 se conecta al Wi-Fi** y luego al broker Mosquitto.
2. Desde el MEGA, se envían comandos por UART al ESP32.
3. El ESP32 interpreta esos comandos y **publica datos a un topic MQTT**, o bien **recibe datos de un topic** y los reenvía al MEGA.
4. Cualquier otro cliente MQTT (Laptop o ESP32) conectado al broker puede suscribirse a los mismos topics para recibir la información, o publicar datos que también llegarán al MEGA a través del ESP.

Primero iremos paso a paso en el desarrollo de la experiencia.

1. Instalación del controlador para ESP32.

1.1. Instrucciones

- Seguir la guía de laboratorio acerca de la instalación de las placas ESP32 en las librerías de Arduino IDE.
- Verificar la carga del código de prueba en el ESP32.

1.2. Responder

- a. En el informe redactar guía básica de instalación que le haya resultado.
- b. En caso de que haya tenido algún problema con la carga del código, contextualizar el problema y cómo lo solucionó.

Checkpoint 01

2. Implementación Básica

2.1. Indicaciones

Empleando un ESP32/ESP8266 y empleando Mosquitto, elija:

Un dispositivo sensor

- Sensor de humedad
- Sensor de proximidad
- Sensor PIR
- Sensor de luz LDR

- Sensor de pulso cardiaco
- Sensor de temperatura

O entradas:

- Teclado
- Joystick

Para obtener información (un dato o medición) y utilizando Mosquitto realizar un ejemplo de publicación y lectura.

2.2. Requisitos

- El sistema implementado debe hacer la medición de un dato de manera periódica.
- La medición, debe ser publicada en un topic pertinente.
- Debe tener un cliente receptor con MQTT, en el que se visualicen los datos recibidos (puede ser con otro ESP o en su computadora).

2.3. Presentar

- Incluir capturas de las señales adquiridas y comunicadas.
- Diagrama de bloques
- Diagrama de flujo

Todo en clase.

Checkpoint 02

Anexos

4. ANEXO 1. Sensor de pulso cardiaco

El sensor de pulso cardíaco fotoeléctrico es un módulo que permite detectar los latidos del corazón de forma no invasiva, utilizando el principio de fotopletismografía (PPG). Este principio se basa en medir los cambios en la absorción de luz causados por la variación del volumen sanguíneo durante el ciclo cardíaco. Este tipo de sensor incorpora generalmente un diodo LED infrarrojo (IR) y un fototransistor. Al colocar el dedo entre estos dos elementos o sobre ellos (dependiendo del diseño), la luz reflejada o transmitida varía con el flujo sanguíneo, generando una señal analógica correlacionada con la frecuencia cardíaca.



Sensor de pulso cardiaco.

Para utilizar el sensor solo es necesario alimentarlo con un voltaje entre 3V a 5V DC y conectar la salida Analógica a la entrada analógica (ADC) de un microcontrolador como Arduino o Pic.

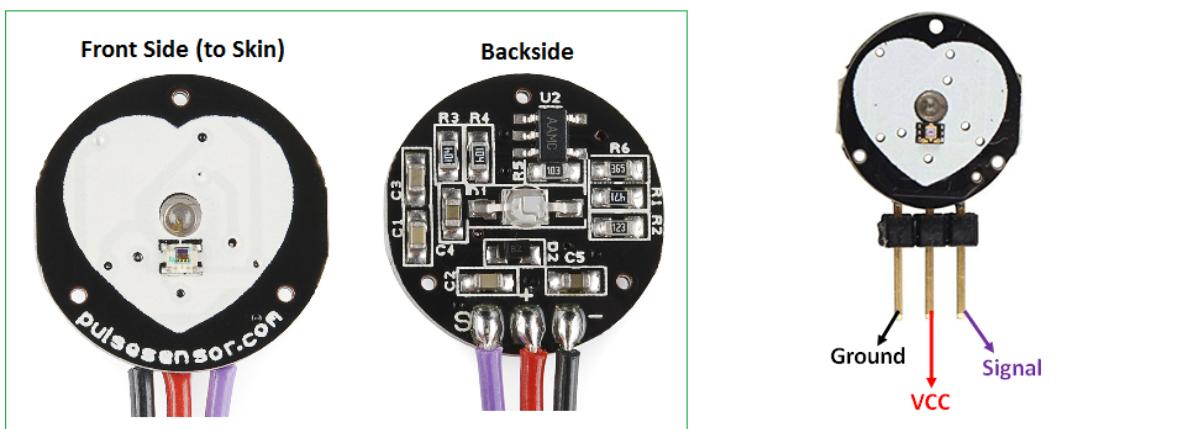
4.1. Funcionamiento

El dispositivo funciona del siguiente modo: un led de color verde emite luz que al entrar en contacto con nuestro dedo indice o el lóbulo del oído refleja cierta cantidad de luz, el flujo de sangre hace que la cantidad de luz reflejada cambie de acuerdo al pulso cardíaco. La luz reflejada es detectada por el sensor de luz APDS-9008, que convierte el flujo de luz en un voltaje analógico. Esta señal analógica es luego filtrada y amplificada en un opamp para luego entregar la señal analógica de salida del dispositivo.

4.2. Especificaciones

- Voltaje de Operación: 3.0V – 5.5V DC
- Consumo corriente: 20mA máx.

- Sensor: APDS-9008
- Opamp: MCP6001
- Led verde
- Longitud de cable: 20cm
- Cables: GND, VCC, Señal



Esquema de conexión

4.3. Ejemplo básico de lectura con Arduino



Diagrama básico de conexión.

Enlace oficial

<https://pulsesensor.com/pages/code-and-guide>

Enlaces adicionales

<https://pulsesensor.com/pages/pulse-sensor-amped-arduino-v1dot1>

5. ANEXO 2. Joystick

El **joystick analógico de dos ejes** es un dispositivo de entrada que permite medir el desplazamiento en dos dimensiones (ejes X e Y) mediante potenciómetros internos, así como detectar pulsaciones mediante un botón integrado al presionar verticalmente la palanca. Es muy utilizado en interfaces hombre-máquina, control de robots, juegos y simuladores.



Este tipo de módulo generalmente integra:

- **Dos salidas analógicas** (X y Y), que varían de 0 a 5 V.
- **Un botón táctil digital** (Z o SW) activado por presión.

5.1. Especificaciones

- Voltaje de Operación: 3.3V - 5V DC
- 2 Potenciómetros: X, Y
- 1 Pulsador
- Dimensiones: 40*26*32mm
- Peso: 11 gramos

5.2. Conexiones

- GND: Tierra, 0V
- +5V: VCC, 3.3V-5.5V DC.
- VRX: salida analógica de potenciómetro eje X (divisor de tensión)
- VRY: salida analógica de potenciómetro eje Y (divisor de tensión)
- SW: salida digital pulsador

5.3. Enlaces

5.3.1. Datasheet:

<https://naylampmechatronics.com/img/cms/Datasheets/000036%20-%20datasheet%20KY-023-Joy-IT.pdf>

5.3.2. Tutorial con Arduino

<https://blog.uelectronics.com/tarjetas-desarrollo/arduino/control-de-servomotores-sg90-con-modulo-ky-023-sensor-joystick/>

6. ANEXO 3. Micro Servo

El micro servo SG90 es un actuador electromecánico compacto ampliamente utilizado en proyectos de electrónica y automatización. Su funcionamiento se basa en un motor de corriente continua con sistema de retroalimentación de posición (potenciómetro interno), lo que le permite ubicarse con precisión dentro de un rango angular limitado, típicamente de 0° a 180°.

Este tipo de servo **no requiere un controlador externo (driver)**, ya que puede ser manejado directamente por microcontroladores como el Arduino Mega mediante señales de modulación por ancho de pulso (PWM).

Puede rotar aproximadamente 180 grados (90° en cada dirección). Tiene la facilidad de poder trabajar con diversidad de plataformas de desarrollo como Arduino, PICs, Raspberry Pi, o en general a cualquier microcontrolador.



Micro servo SG90 de 1.5Kg

Puede ser alimentado por USB pero se recomienda alimentar por separado el microcontrolador y los servos, ya que el ruido eléctrico puede dar lugar a errores en la ejecución del programa, o en todo caso agregar un capacitor de 100uF entre 5V y GND.

Nota: Para su uso con Arduino, recomendamos conectar el cable naranja al pin 9 o 10 y usar la Librería "Servo" incluida en el IDE de Arduino. Para la posición 0° el pulso es de 0.6ms, para 90° es de 1.5ms y para 180° 2.4ms.

6.1. Especificaciones

- Voltaje de alimentación: 3.0 - 7.2V DC
- Velocidad: 0.1seg / 60 grados
- Torque reposo: 1.3Kg.cm (4.8V), 1.6Kg.cm (6.0V)
- Ancho de pulso: 4useg (Dead band)

- Engranajes: Nylon
- Longitud del conductor: 150mm
- Dimensiones: 22*11.5*27 mm
- Peso: 9g

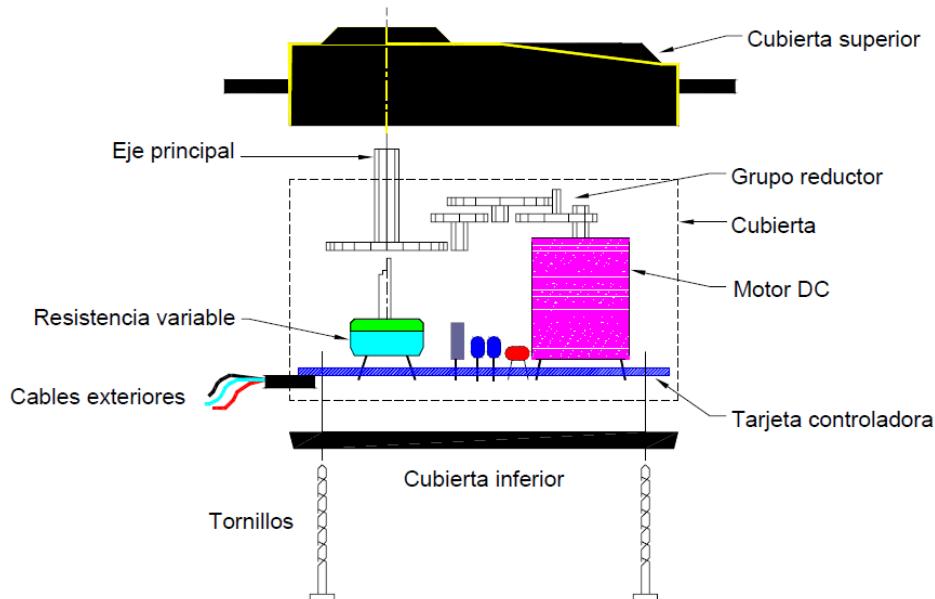


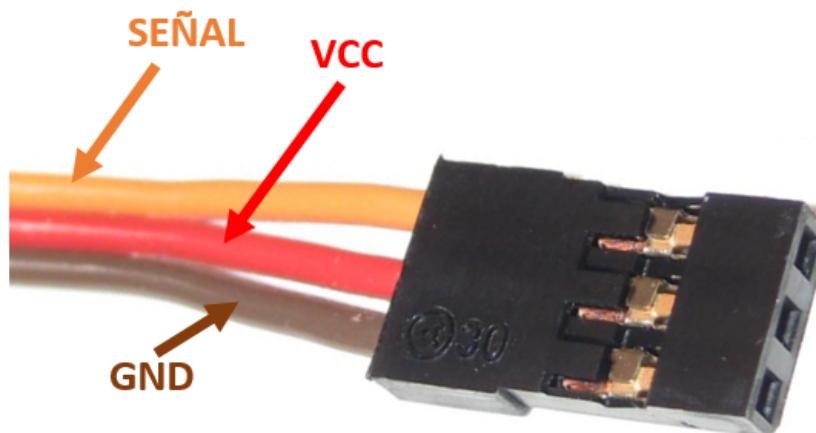
Diagrama interno de un micro servo

Servomotor	Modelo	Voltaje	Torque
	SG90	3V-7.2V	1-1.6 Kg/cm
	SG-5010	4.8V-6V	3.5K-6.5 Kg/cm
	MG995	4.8V – 7.2V	8.5-10 kg/cm

Servomotores comerciales más conocidos.

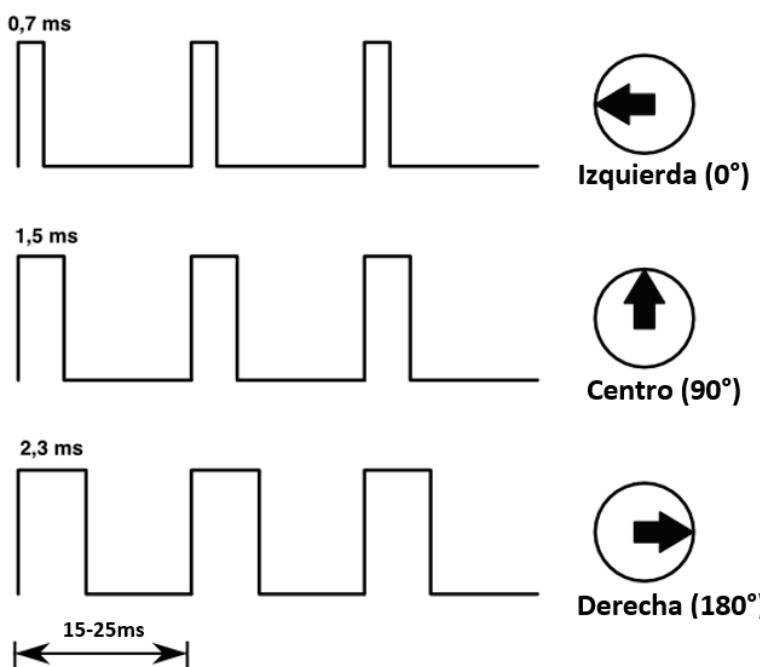
6.2. Uso con Arduino

Todos los servos usados para robótica, tiene un conector de 3 cables. VCC (rojo), GND (Marrón) y Señal (Naranja):



Conecotor servo

La señal o dato que hay que enviarle al servo es una señal de PWM donde el tiempo en alto es equivalente al ángulo o posición del servo. Estos valores pueden variar y van desde 0.5 a 1 milisegundo para la posición 0° y 2 a 2.4 milisegundos para la posición de 180°, el periodo de la señal debe ser cercano a 20 milisegundos.



Señal pwm para un servo.

6.3. Librería servo de Arduino

El IDE Arduino trae una librería completa para el control de servomotores, la documentación completa lo encontramos en su página oficial: [Servo Library](#)

La biblioteca Servo admite hasta 12 motores en la mayoría de las placas Arduino y 48 en el Arduino Mega. En las placas que no sean los de Mega, el uso de la biblioteca desactiva la funcionalidad PWM en las patillas

9 y 10. En el Arduino Mega, hasta 12 servos pueden ser utilizados sin interferir con la funcionalidad PWM, pero si se usan de 12 a 23 motores desactivará el PWM en los pines 11 y 12.

A continuación se muestran las funciones de la librería Servo:

- **attach(Pin)**

Establece el pin indicado en la variable servo. Ej: servo.attach(3);

- **attach(Pin,min,max)**

Establece el pin indicado en la variable servo, considerando min el ancho de pulso para la posición 0° y máx. el ancho de pulso para 180°.Ej: servo.attach(3,900,2100);

- **write(angulo)**

Envía la señal correspondiente al servo para que se ubique en el ángulo indicado, ángulo es un valor entre 0 y 180°. Ej: servo.write(45);

- **writeMicroseconds(tiempo)**

Envía al servo el ancho de pulso=tiempo en microsegundos. Ej: servo.writeMicroseconds(1500);

- **read()**

Lee la posición actual del servo, devuelve un valor entre 0 y 180. Ej: angulo=servo.read();

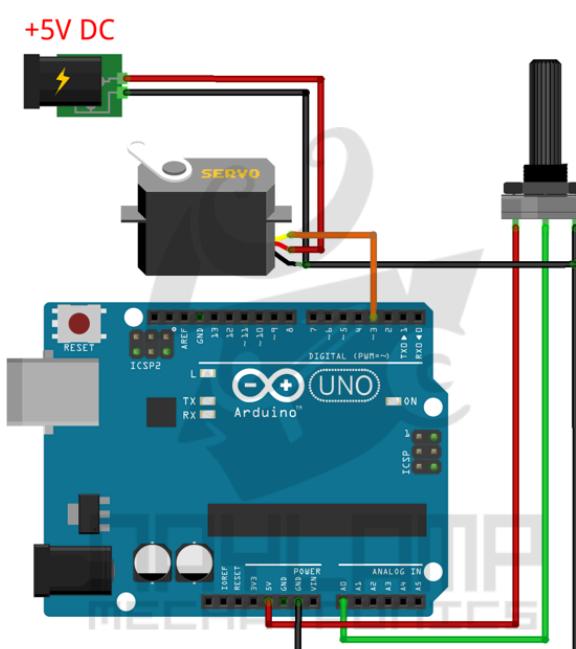
- **attached(Pin)**

Verifica si la variable servo está unido al pin indicado, devuelve true o false. Ej: if(servo.attached(3));

- **detach(pin)**

Separa la variable Servo del pin indicado. Ej: servo.detach(3);

6.4. Ejemplo



Nota: *La alimentación del motor puede ser la misma que el Arduino siempre y cuando la fuente soporte la potencia del servo y sea de 5V. Los 5V del USB solo soporta un servo SG90, más servos o de otro tipo se necesita usar una fuente externa.

6.4.1. Código ejemplo:

```
//LIBRARIES:  
  
#include <Servo.h>  
  
Servo myservo; //creamos un objeto servo  
int angulo;  
  
void setup() {  
    myservo.attach(3); // asignamos el pin 3 al servo.  
    Serial.begin(9600); // iniciamos el puerto serial  
}  
  
void loop() {  
  
    angulo= 0;  
    myservo.write(angulo);  
    Serial.print("ángulo: ");  
    Serial.println(angulo);  
    delay(2000);  
  
    angulo= 90;  
    myservo.write(angulo);  
    Serial.print("ángulo: ");  
    Serial.println(angulo);  
    delay(2000);  
  
    angulo= 180;  
    myservo.write(angulo);  
    Serial.print("ángulo: ");  
    Serial.println(angulo);  
    delay(2000);  
  
    angulo= 90;
```

```
myservo.write(angulo);
Serial.print("ángulo:  ");
Serial.println(angulo);
delay(2000);
}
```

6.4.2. Otro ejemplo (*Control de un servomotor a través de un potenciómetro*)

En este ejemplo controlaremos la posición del servo por medio de un potenciómetro, donde el servo se posicionará en función de la posición del potenciómetro.

```
#include <Servo.h>

Servo myservo; // creamos un objeto servo

void setup() {
  myservo.attach(3); // asignamos el pin 3 al servo.
  Serial.begin(9600);
}

void loop() {
  int adc = analogRead(A0); // realizamos la lectura del potenciómetro
  int angulo = map(adc, 0, 1023, 0, 180); // escalamos la lectura a un valor
entre 0 y 180
  myservo.write(angulo); // enviamos el valor escalado al servo
  Serial.print("ángulo:  ");
  Serial.println(angulo);
  delay(10);
}
```

6.5. Referencia

https://naylampmechatronics.com/blog/33_tutorial-uso-de-servomotores-con-arduino.html

7. ANEXO 2. Qué es MQTT

MQTT (Message Queuing Telemetry Transport) es un protocolo de mensajería ligero y eficiente, diseñado para dispositivos con recursos limitados y redes de baja confiabilidad. Este protocolo se basa en un modelo cliente-servidor (broker) en el que los dispositivos (clientes) se suscriben a temas específicos, enviando y recibiendo mensajes a través de un intermediario (broker). Es ideal para aplicaciones de IoT (Internet of Things), donde la comunicación entre sensores, dispositivos y plataformas debe ser rápida, confiable y con bajo consumo de recursos.

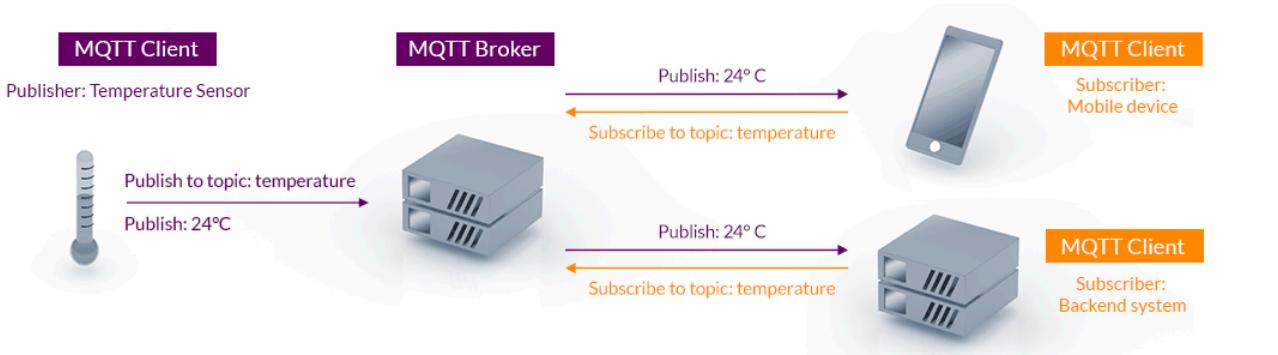


Figura 1. Arquitectura de comunicación MQTT

8. ANEXO 3. ESP32

El ESP32, un microcontrolador con capacidades Wi-Fi y Bluetooth de bajo consumo, es ampliamente utilizado para implementar aplicaciones de IoT y es compatible con MQTT. Al utilizar el ESP32 con MQTT, se pueden crear sistemas conectados de manera eficiente, donde el ESP32 puede tanto publicar datos (como la lectura de sensores) como suscribirse a temas para recibir comandos o actualizaciones en tiempo real. Esto permite que dispositivos ESP32 comuniquen información entre ellos o con servidores en la nube de manera eficiente y escalable.

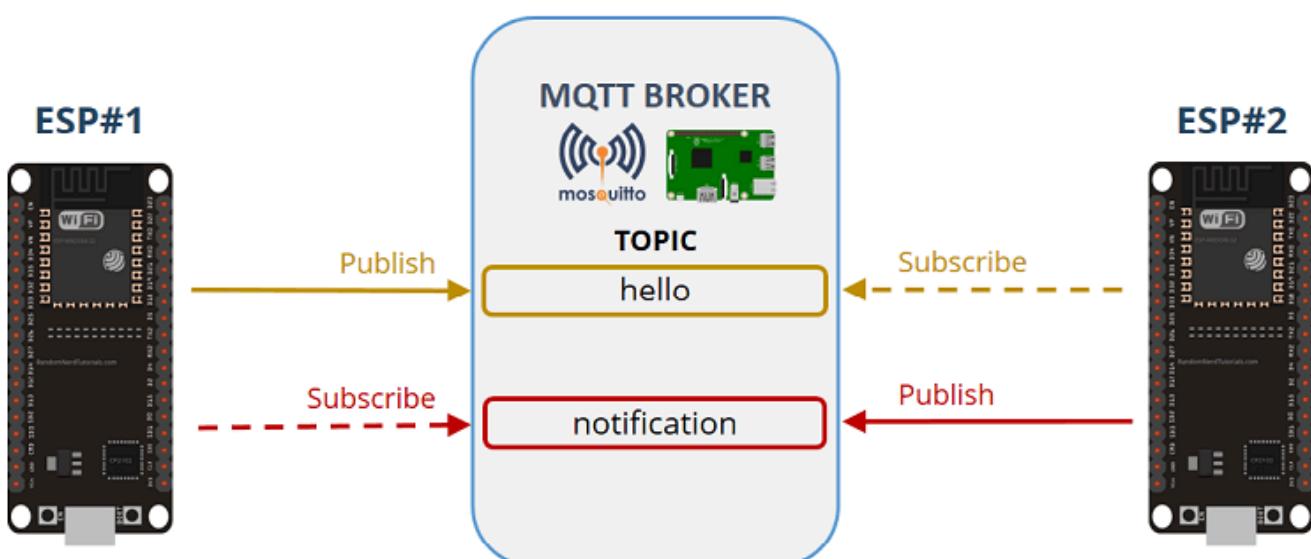


Figura 2. Arquitectura de comunicación MQTT con ESP32

8.1. Uso del ESP32 con MQTT.

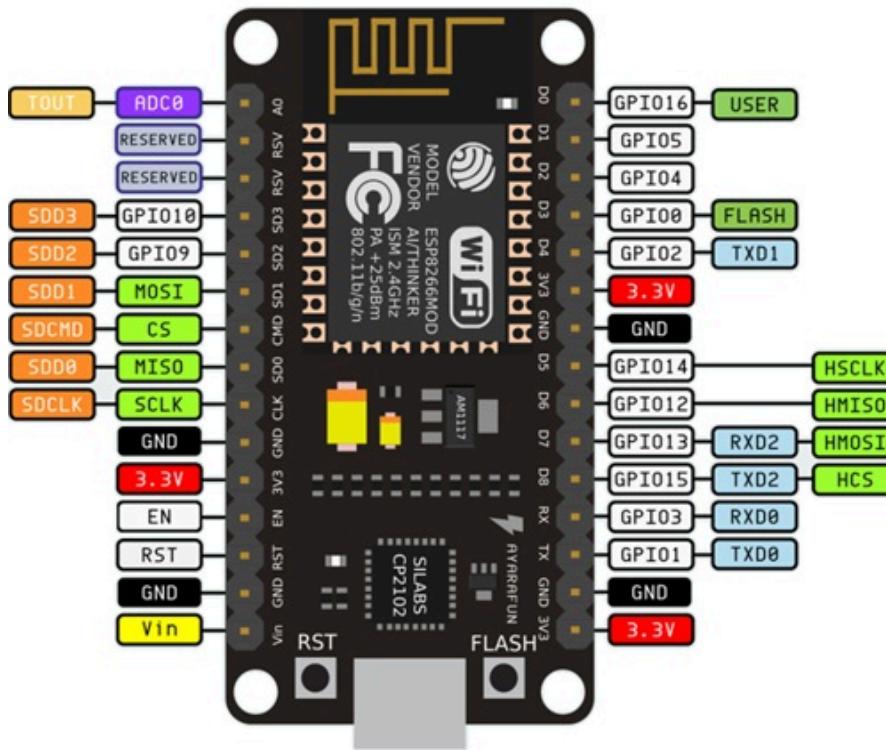


Figura 3. Pinout del ESP32.

Similar al Arduino usado, el ESP32 es un microcontrolador el cual tiene diversos pines de entrada y salida, como pines digitales, pines de lectura análoga, pines de alimentación y pines de comunicación (serial, SPI, etc.)

Este se puede controlar y programar a través del IDE de Arduino.

ATENCIÓN

Cualquier Microcontrolador ESP32 (ESP8266, CAM, NodeMCU) trabaja a 3.3 V, lo que significa que los pines de E/S trabajan también a 3.3V. Los pines no toleran voltajes de 5V, por lo que aplicar un valor de voltaje mayor a 3.6V daña permanentemente el chip del MCU. Así mismo la máxima corriente de alimentación que brinda cualquier GPIO es de 12 mA.

9. ANEXO 4. Comunicación UART (Serial) entre dos dispositivos

UART significa Receptor/Transmisor Asíncrono Universal. Es un dispositivo de hardware (o circuito) utilizado para la comunicación en serie entre dos dispositivos.

La conexión de dos dispositivos UART es sencilla y directa. La Figura a continuación muestra un diagrama de conexión UART básico.

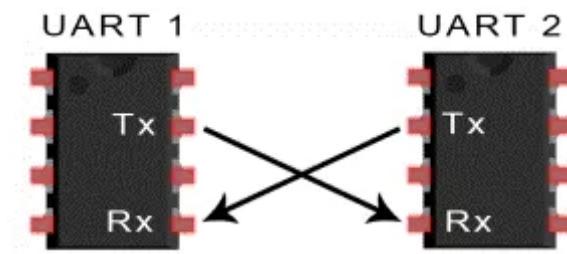


Figura 3. Esquema de conexión entre dos dispositivos con comunicación serial. Siempre debe ser cruzado, a fin de que el TX1 envíe datos al RX1, e idem.

Un cable es para transmitir datos (llamado pin TX) y el otro es para recibir datos (llamado pin RX). Sólo podemos conectar dos dispositivos UART juntos.

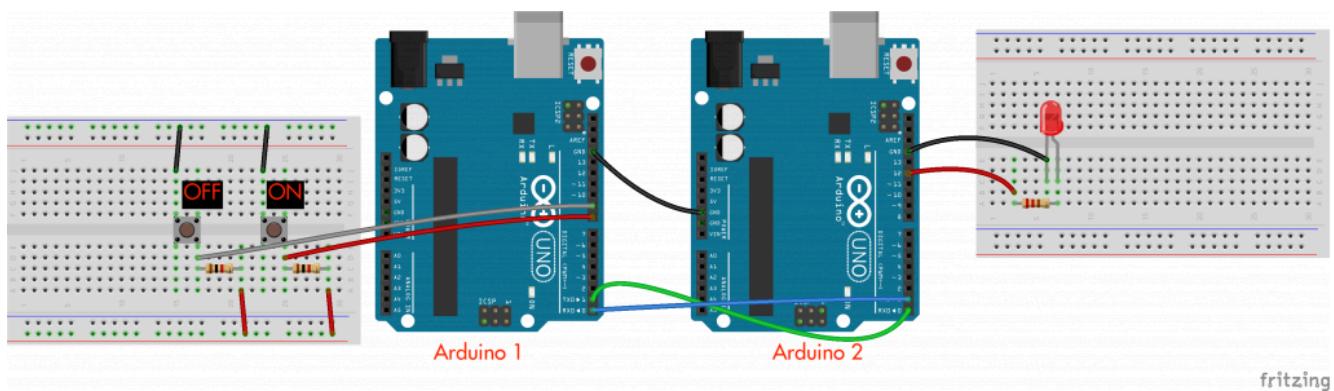


Figura 4. Ejemplo de diagrama de conexión de comunicación serial entre dos dispositivos.

9.1. Envío de datos en Serie

Antes de que el dispositivo UART pueda enviar datos, el dispositivo transmisor convierte los bytes de datos en bits. Después de convertir los datos en bits, el dispositivo UART los divide en paquetes para su transmisión. Cada paquete contiene un bit de inicio, una trama de datos, un bit de paridad y los bits de parada. La figura a continuación muestra un paquete de datos de ejemplo. Después de preparar el paquete, el circuito UART lo envía a través del pin TX.

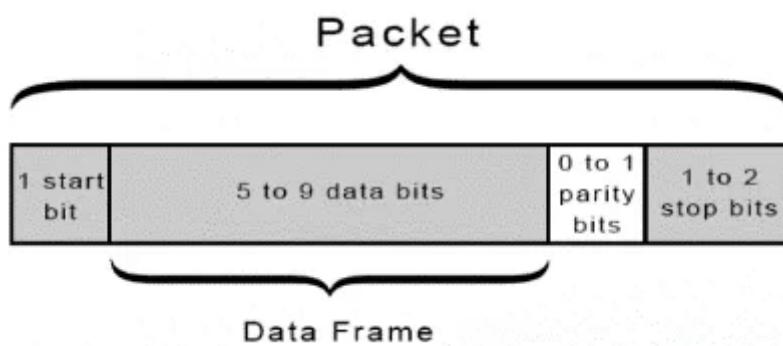


Figura 6. Ejemplo de un datagrama en comunicación Serial.

9.2. Tasa de Baudios

La tasa de baudios es el número de bits por segundo (bps) que un dispositivo UART puede transmitir/recibir. Necesitamos configurar ambos dispositivos UART con la misma tasa de baudios para tener una correcta transmisión de datos. Los valores más comunes para la tasa de baudios son 1200, 2400, 4800, 9600, 19200, 38400, 57600, y 115200 bps.

IMPORTANTE: Ambos puertos Serial empleados deben tener configurada la misma tasa de baudios (Ej. 9600 u 115200)

9.3. Interfaz UART en Arduino

Arduino tiene uno o más pines UART dependiendo de la placa. Para este caso usaremos un Arduino Uno que tiene sólo una interfaz UART, la que se encuentra en el pin 0 (RX0) y el pin 1 (TX0). Los pines 0 y 1 de Arduino también se utilizan para comunicarse con el IDE de Arduino a través del USB. Así que si vas a cargar sketches a tu UNO, asegúrate de desconectar primero los cables de los pines 0 y 1. La figura a continuación muestra la ubicación de los pines UART TX y RX.



Figura 7. Pines Tx/Rx Arduino Uno

9.4. Niveles lógicos (Importante)

Los niveles lógicos de UART pueden diferir entre fabricantes. Por ejemplo, un Arduino Uno tiene un nivel lógico de 5V pero el puerto RS232 de un ordenador tiene un nivel lógico de +/-12V. Conectar un Arduino Uno directamente a un puerto RS232 dañará el Arduino. Si ambos dispositivos UART no tienen los mismos niveles lógicos, se necesita un circuito convertidor de nivel lógico adecuado para conectar los dispositivos.

9.5. Conexiones básicas (Serial1):

- Arduino 1 TX1 → Arduino 2 RX1
- Arduino 1 RX1 ← Arduino 2 TX1

- GND ↔ GND

9.6. Tutorial de Referencia

A continuación se comparte el enlace de un ejemplo de comunicación Serial entre dos dispositivos Arduino (5V)

<https://arduino.cl/como-configurar-la-comunicacion-uart-en-arduino/>

<https://www.automatizacionparatodos.com/comunicacion-entre-dos-arduininos-con-el-puerto-serie/>

10. ANEXO 5. Comunicación entre Arduino (5V) y un dispositivo de 3.3V

10.1. Consideraciones

- El pin TX del Arduino (5V) puede dañar el RX del dispositivo de 3.3V si se conecta directamente.
- El pin RX del Arduino puede leer señales de 3.3V sin problema (generalmente se interpreta como HIGH).

10.2. Opciones para adaptación de nivel:

1. **Divisor resistivo:** Para bajar la señal de 5V a 3.3V.
 - Conecta una resistencia de $1.8\text{k}\Omega$ entre TX (Arduino) y RX (dispositivo 3.3V)
 - Otra resistencia de $3.3\text{k}\Omega$ entre RX (dispositivo 3.3V) y GND
2. **Conversor de nivel lógico:** Dispositivo dedicado para hacer la conversión segura de $5\text{V} \leftrightarrow 3.3\text{V}$.

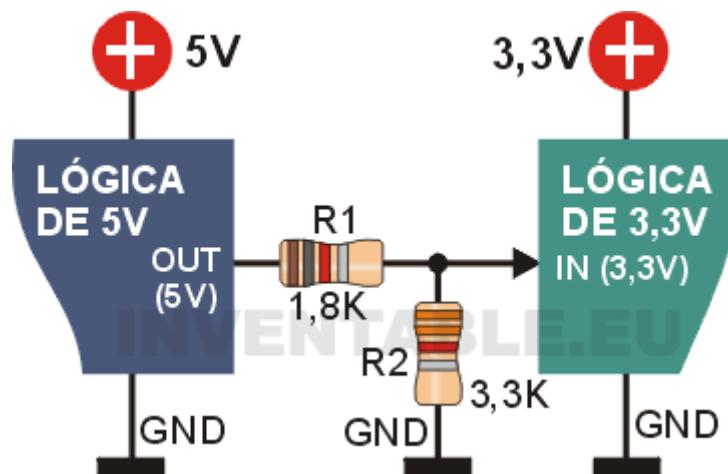


Figura 5. Esquema ejemplo del divisor de voltaje.

Específicamente, esto debe ser tomado en cuenta con mayor razón al hacer comunicación de datos entre Arduino y ESP32 (Y posteriormente con dispositivos LoRa).

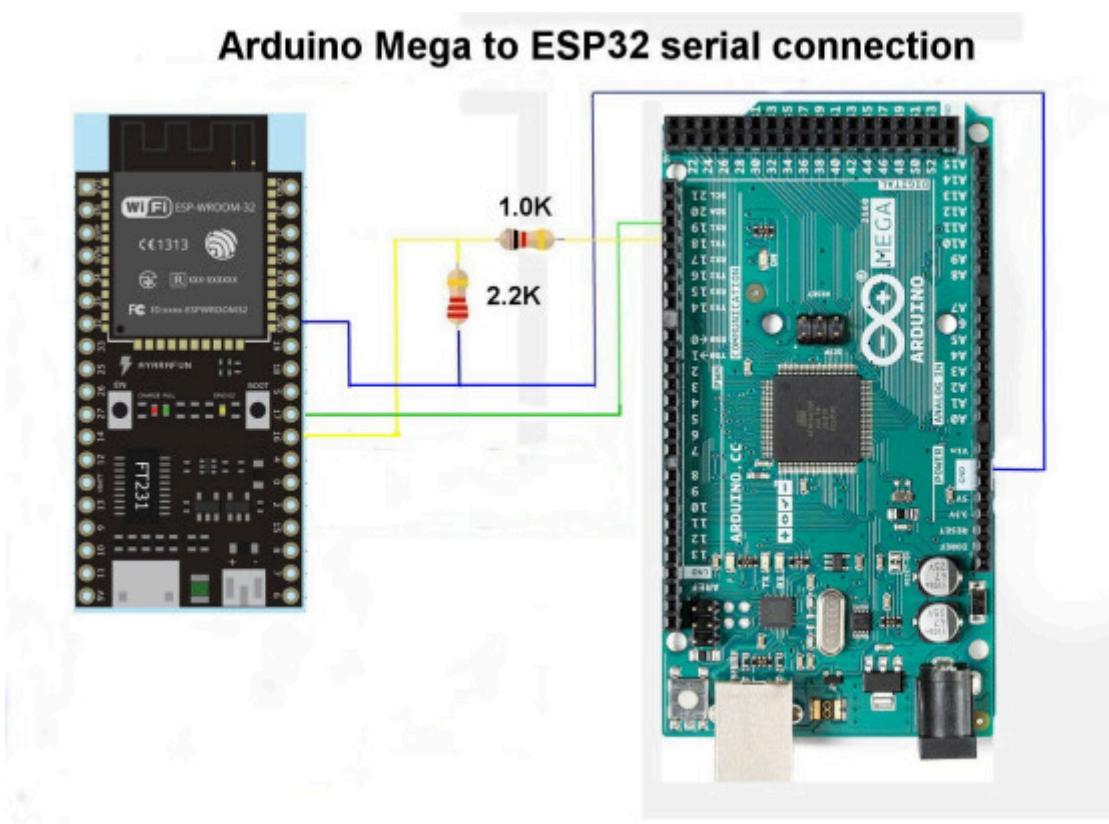


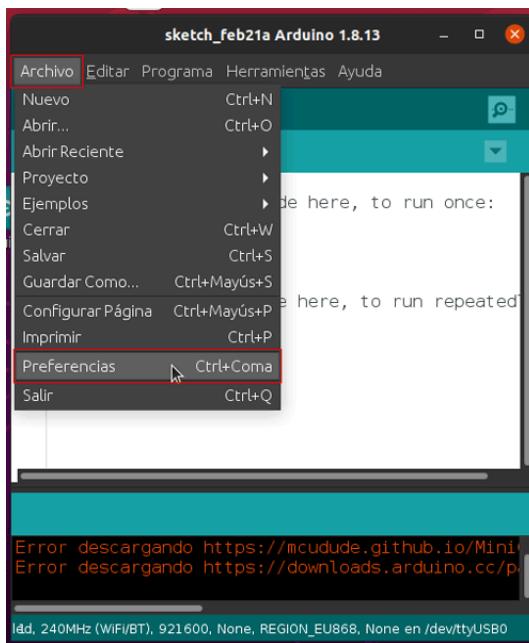
Figura 6. Ejemplo de conexión de comunicación serial entre ESP32 y Arduino Mega

11. ANEXO 6. Instalación de placas ESP32 en Arduino

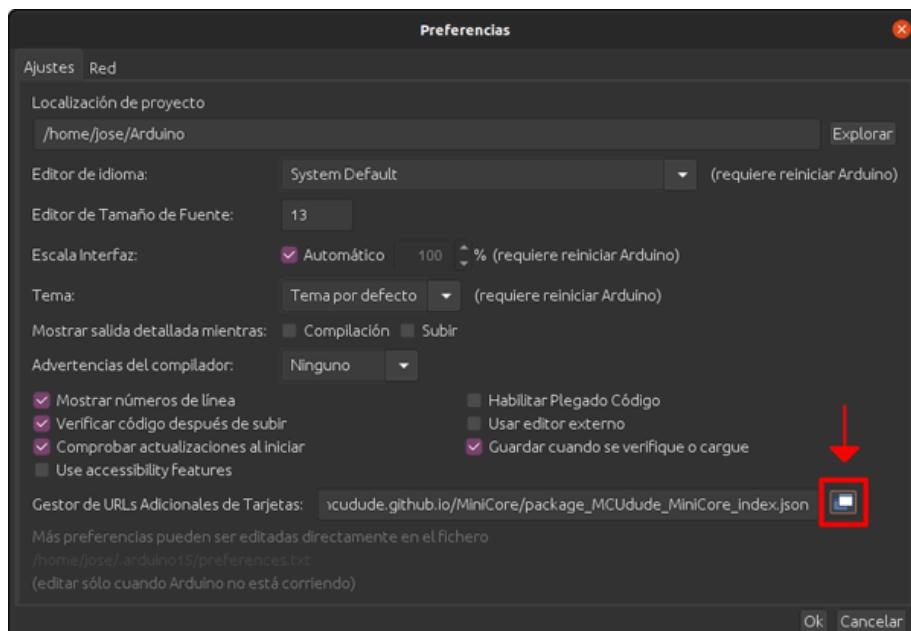
11.1. Adicionar las URLs para placas ESP32

Para programar un ESP32 desde Arduino hay que agregar las URLs de las placas ESP32 para poder descargar el núcleo (o core) de ESP32 para Arduino.

Lo primero es ejecutar Arduino IDE y hacer clic en “Archivo>Preferencias”.



En la ventana de preferencias es necesario hacer clic en el botón “Gestor de tarjetas adicionales”.



Ahora, en la nueva ventana se pega la siguiente URL.

https://dl.espressif.com/dl/package_esp32_index.json

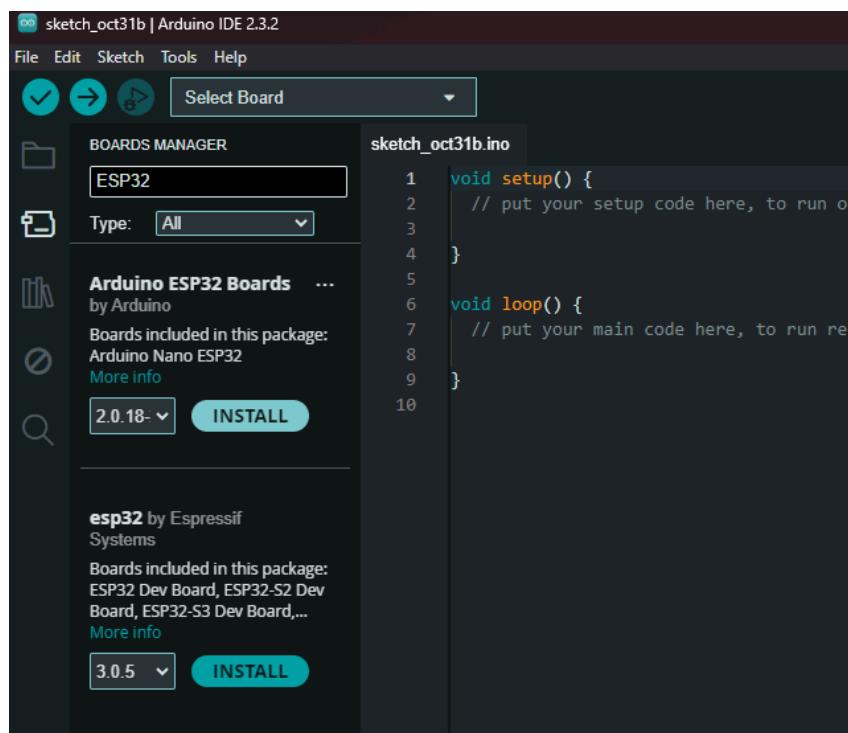
Con esta dirección el gestor de placas tendrá acceso a un conjunto elevado de placas y módulos ESP32 de varios fabricantes. Una vez agregadas las URLs es necesario hacer clic en el botón “OK” de la ventana. Esto te devuelve a la ventana de preferencias, donde también tienes que hacer clic en el botón “OK”.

11.2. Instalar core y placa ESP32

Para instalar el soporte para ESP32 y las placas de desarrollo hay que ir a “Herramientas>Placas>Gestor de Tarjetas”.

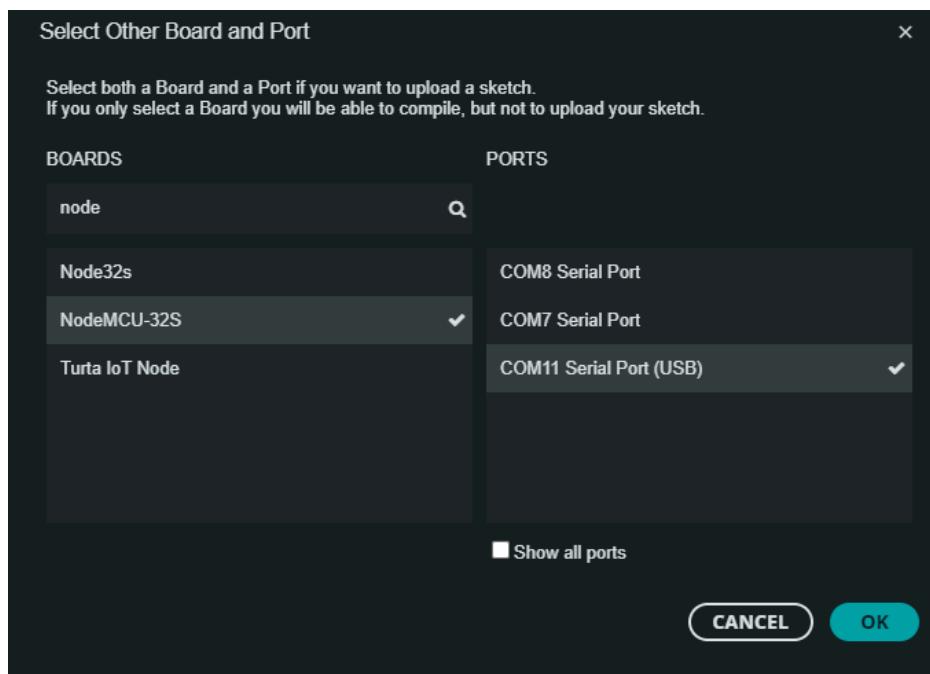
Esto abrirá el gestor de placas o tarjetas. En cuanto se inicie, comenzará a actualizar su base de datos, utilizando las URLs que se agregaron anteriormente en preferencias.

Una vez termine, hay que escribir “ esp32 ” en la barra de búsqueda para filtrar las placas disponibles.



Ahora solo resta instalar el paquete que contiene la placa de desarrollo a utilizar. En este caso sería la primera opción. Ten en cuenta que este proceso puede tardar un tiempo, ya que el software tiene que descargar todos los archivos necesarios para programar el ESP32.





12. ANEXO 7. Cómo programar un ESP32 con el IDE Arduino

Una vez instalados todos los archivos necesarios es hora de programar tu placa ESP32 .

Requisitos previos

- **Arduino IDE:** es necesario tener el software instalado en tu sistema. En caso de que no lo tengas y presentes dudas con respecto a su instalación puedes consultar un artículo anterior donde se muestra cómo instalar Arduino IDE paso a paso.
- **Placa de desarrollo ESP32:** por supuesto, es necesario tener una placa de desarrollo basada en ESP-32 . Aunque, el proceso es similar para todas las placas, en este caso se utiliza la Heltec Wireless Stick Lite.
- **Cable USB (micro-usb):** aunque puede ser diferente en algunos casos, casi todas las placas de desarrollo basadas en ESP-32 requieren un cable micro USB en un extremo (placa) y USB tipo A en el otro (ordenador). Vamos, que es el típico cable que utilizan la mayoría de los móviles.

Más información acerca de la programación se encuentra en:

[Tutorial de ESP32 con Arduino IDE](#)

[Tutorial 2 de ESP32 con Arduino IDE](#)

[Medium Example](#)

Nota. Se recomienda:

- Tomar como referencia las siguientes páginas:
 - [Ejemplo Medium](#)
 - [Ejemplo Prometec](#)
 - <https://cedalo.com/blog/enabling-esp32-mqtt/>

- <https://www.electronicwings.com/esp32/esp32-mqtt-client>

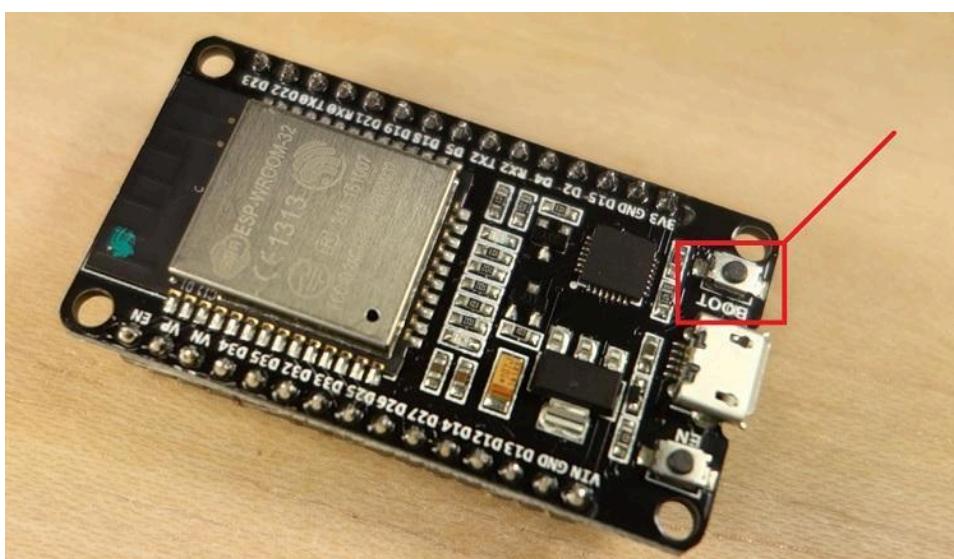
- Utilizar una red WiFi interna (utilizando el Hotspot de su celular o laptop).

13. ANEXO 8. Troubleshooting

If you try to upload a new sketch to your ESP32 and you get this error message “A fatal error occurred: Failed to connect to ESP32: Timed out... Connecting...“. It means that your ESP32 is not in flashing/uploading mode.

Having the right board name and COM port selected, follow these steps:

- Hold-down the “BOOT” button in your ESP32 board



- Press the “Upload” button in the Arduino IDE to upload your sketch:



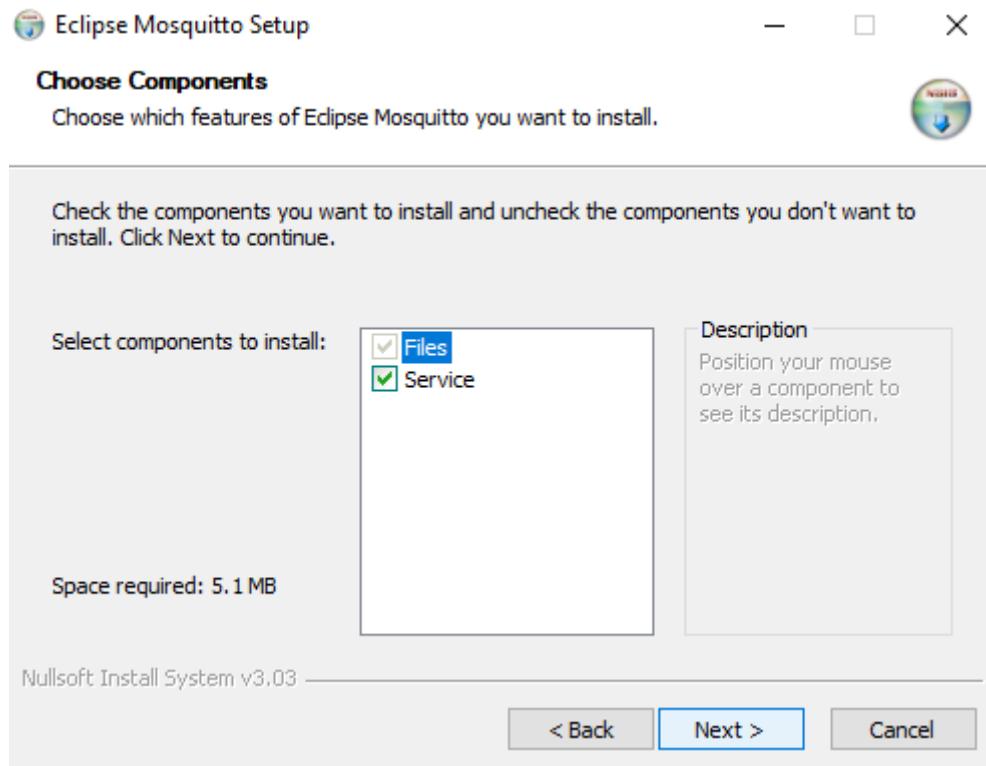
- After you see the “Connecting....” message in your Arduino IDE, release the finger from the “BOOT” button:

```
Uploading...
Archiving built core (caching) in: C:\Users\RUISAN-1\AppData\Local\Temp\arduino_cache_959083\core\core_espressif_esp32_esp32doit-devkit-v1_Flash
Sketch uses 501366 bytes (38%) of program storage space. Maximum is 1310720 bytes.
Global variables use 37320 bytes (12%) of dynamic memory, leaving 257592 bytes for local variables. Maximum is 294912 bytes.
esptool.py v2.1
Connecting.....
Chip is ESP32D0WDQ6 (revision unknown 0xa)
Uploading stub...
#running stub...
Stub running...
Changing baud rate to 921600
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 8192 bytes to 47...
Writing at 0x0000e000... (100 %)
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.0 seconds (effective 8192.1 kbit/s)...
Hash of data verified.
Compressed 12304 bytes to 8126...
Writing at 0x00001000... (100 %)
```


13.1. Mosquitto Installing

13.1.1. Step 1

[Downloading](#) and installing the mosquitto mqtt broker.



Eclipse Mosquitto version 2.0.10

13.1.2. Step 2

Running the mosquitto broker.

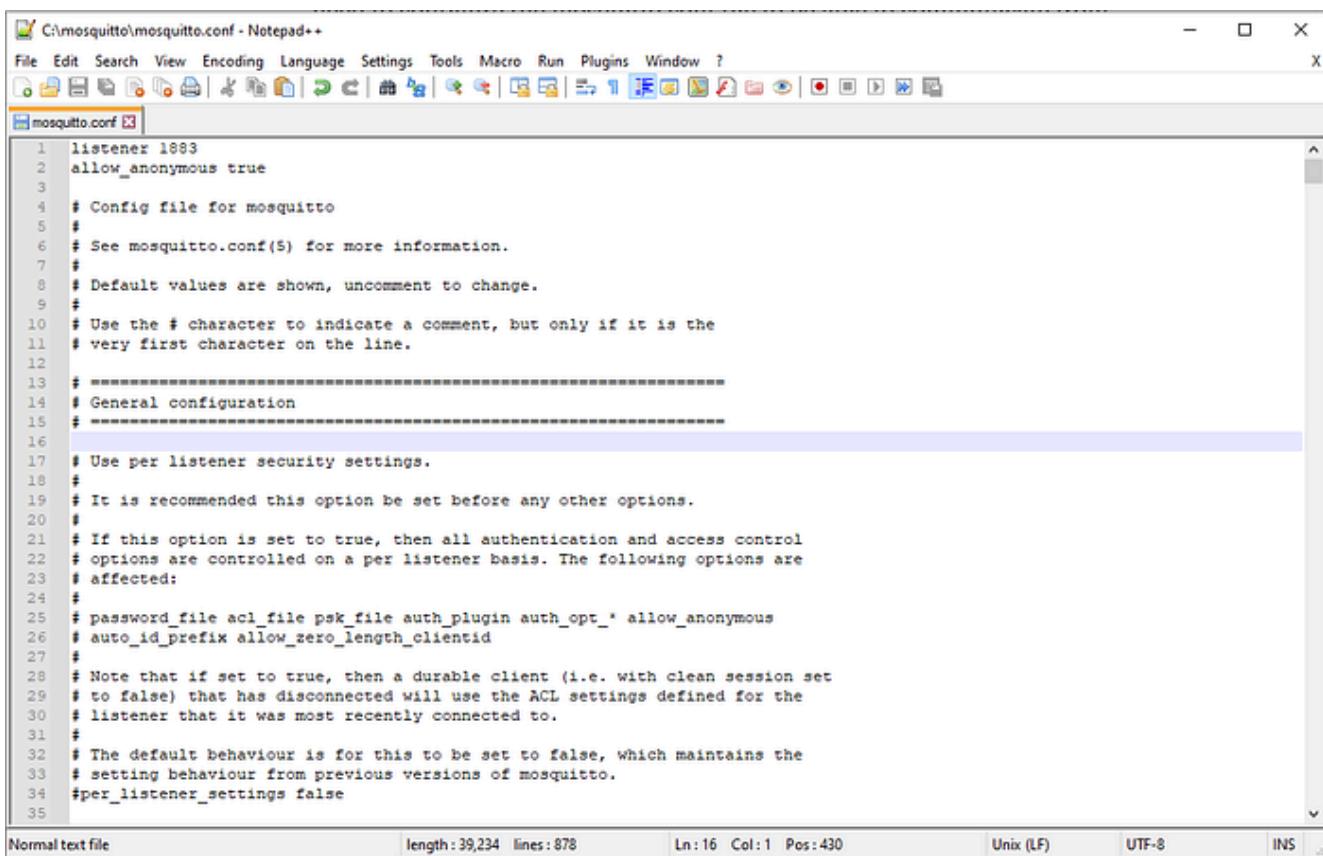
\$ mosquitto -v

A screenshot of a Windows Command Prompt window titled 'Command Prompt - mosquitto'. The window shows the output of the command 'mosquitto -v':

```
C:\>mosquitto -v
1620474608: mosquitto version 2.0.10 starting
1620474608: Using default config.
1620474608: Starting in local only mode. Connections will only be possible from clients running on this machine.
1620474608: Create a configuration file which defines a listener to allow remote access.
1620474608: For more details see https://mosquitto.org/documentation/authentication-methods/
1620474608: Opening ipv4 listen socket on port 1883.
1620474608: Opening ipv6 listen socket on port 1883.
1620474608: mosquitto version 2.0.10 running
```

The -v flag will display running information

The default config file will not allow communication from outside so we will need to configure the mosquitto.conf file to be able to communicate.

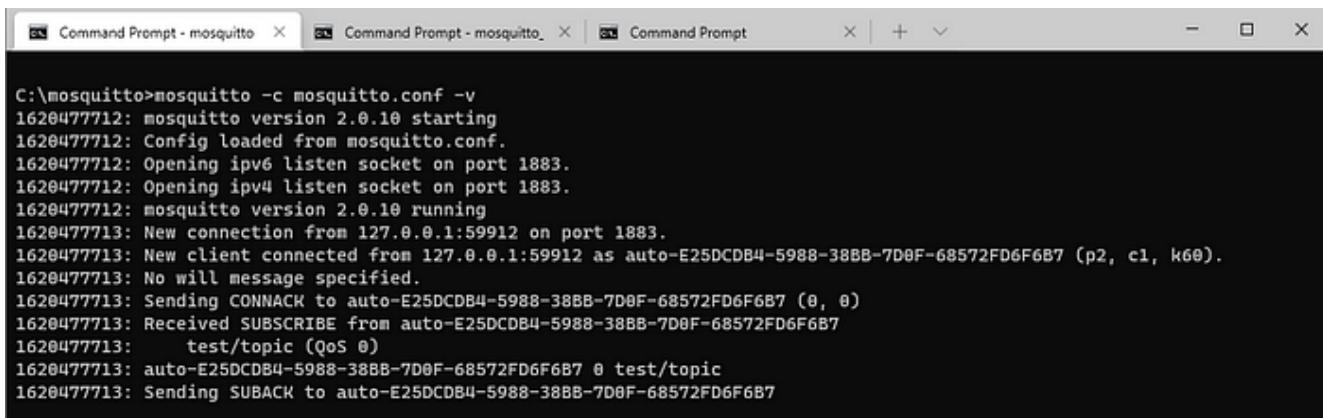


```
C:\mosquitto\mosquitto.conf - Notepad++  
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?  
mosquito.conf  
1 listener 1883  
2 allow_anonymous true  
3  
4 # Config file for mosquitto  
5 #  
6 # See mosquitto.conf(5) for more information.  
7 #  
8 # Default values are shown, uncomment to change.  
9 #  
10 # Use the # character to indicate a comment, but only if it is the  
11 # very first character on the line.  
12  
13 # =====  
14 # General configuration  
15 # =====  
16  
17 # Use per listener security settings.  
18 #  
19 # It is recommended this option be set before any other options.  
20 #  
21 # If this option is set to true, then all authentication and access control  
22 # options are controlled on a per listener basis. The following options are  
23 # affected:  
24 #  
25 # password_file acl_file psk_file auth_plugin auth_opt_* allow_anonymous  
26 # auto_id_prefix allow_zero_length_clientid  
27 #  
28 # Note that if set to true, then a durable client (i.e. with clean session set  
29 # to false) that has disconnected will use the ACL settings defined for the  
30 # listener that it was most recently connected to.  
31 #  
32 # The default behaviour is for this to be set to false, which maintains the  
33 # setting behaviour from previous versions of mosquitto.  
34 #per_listener_settings false  
35  
Normal text file length:39,234 lines:878 Ln:16 Col:1 Pos:430 Unix (LF) UTF-8 INS
```

Adding the two frst lines in the begining of the file will do the work

Running the mosquitto broker using config file.

\$ mosquitto -c mosquito.conf -v



```
C:\mosquitto>mosquitto -c mosquito.conf -v  
1620477712: mosquitto version 2.0.10 starting  
1620477712: Config loaded from mosquito.conf.  
1620477712: Opening ipv6 listen socket on port 1883.  
1620477712: Opening ipv4 listen socket on port 1883.  
1620477712: mosquitto version 2.0.10 running  
1620477713: New connection from 127.0.0.1:59912 on port 1883.  
1620477713: New client connected from 127.0.0.1:59912 as auto-E25DCDB4-5988-38BB-7D0F-68572FD6F6B7 (p2, c1, k60).  
1620477713: No will message specified.  
1620477713: Sending CONNACK to auto-E25DCDB4-5988-38BB-7D0F-68572FD6F6B7 (0, 0)  
1620477713: Received SUBSCRIBE from auto-E25DCDB4-5988-38BB-7D0F-68572FD6F6B7  
1620477713:    test/topic (Qos 0)  
1620477713: auto-E25DCDB4-5988-38BB-7D0F-68572FD6F6B7 0 test/topic  
1620477713: Sending SUBACK to auto-E25DCDB4-5988-38BB-7D0F-68572FD6F6B7
```

13.1.3. Step 3

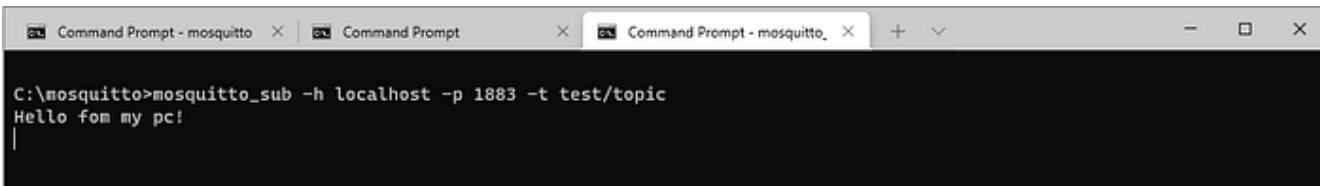
List received messages within a specific topic.

\$ mosquitto_sub -h localhost -p 1883 -t test/topic

13.1.4. Step 4

Send a message from the computer terminal to a specific topic.

\$ mosquitto_pub -h localhost -p 1883 -t test/topic -m "Hello fom my pc!"



The screenshot shows three separate Command Prompt windows. The first window (C:\mosquitto>) contains the command 'mosquitto_sub -h localhost -p 1883 -t test/topic'. The second window (C:\mosquitto>) shows the response 'Hello fom my pc!'. The third window (C:\mosquitto>) is empty.

Received message from the last command

For now everything is working locally, just by establishing a communication from the local machine to itself.

We want for sure to integrate this with other IoT devices, so let's get it working with a module!

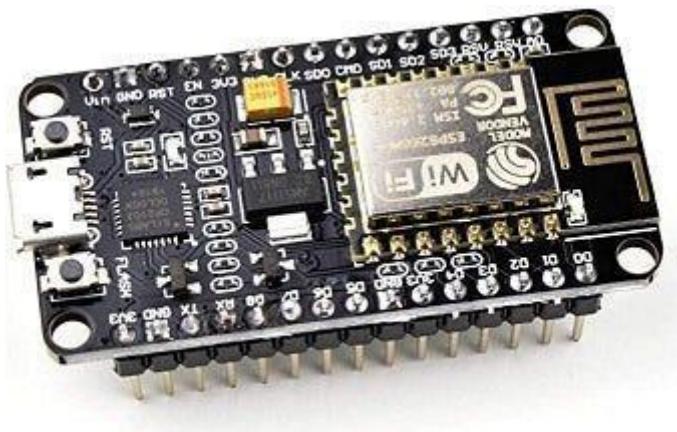


Figura 10. ESP8266 with WiFi