Uno de sus puntos fuertes es que es extremadamente simple y ligero. Por este motivo es muy interesante para sistemas que requieren poco ancho de banda, tienen una alta latencia y requieren de poco consumo de los dispositivos.

Los objetivos del protocolo MQTT es minimizar el ancho de banda, comunicación bidireccional entre dispositivos, minimizar los requerimientos de los dispositivos tanto recursos como consumo y garantizar la fiabilidad y cierto grado de seguridad.

**La latencia es el tiempo desde que se envía un mensaje hasta que llega a su destino. Cuanto más tiempo, más latencia.**

**¿Quién creó MQTT?**

Al final, como sucede con casi todas las tecnologías, surgió por una necesidad. Los ingenieros Dr. Andy Stanford-Clark de IBM y Arlen Nipper de Eurotech fueron los creadores del protocolo MQTT.

En 1.999 estaban trabajando en la construcción de un oleoducto. De repente tenían la necesidad de enviar datos a través de un protocolo de comunicaciones que optimizara el ancho de banda y el consumo de las baterías.

Por aquel entonces para transmitir información desde un sitio remoto como un desierto, un bosque o una montaña, tenías que hacer uso de la comunicación vía satélite. Este tipo de comunicación era extremadamente caro en aquellos tiempos.

Ahora tenemos otras tecnologías como las redes [**LPWAN**](https://programarfacil.com/blog/arduino-blog/arduino-mkrfox1200-sigfox-lpwan/) como SigFox o Lora. Pero en 1999 no había tantas posibilidades. Esto les llevó a crear el protocolo MQTT para un caso concreto pero en la actualidad se utiliza en infinidad de proyectos.

En comunicaciones máquina a máquina o M2M, dispositivos que utilizan tecnología IoT, aplicaciones móviles con la necesidad de optimizar el ancho de banda y el consumo eléctrico. Puedes ver un montón de proyectos en la [**web oficial de MQTT**](https://mqtt.org/projects).

### ¿Es un estándar de comunicaciones?

Desde el año 2014 se ha convertido en un **[estándar OASIS](https://www.oasis-open.org/news/announcements/mqtt-version-3-1-1-becomes-an-oasis-standard" \t "_blank)**. Esta organización sin ánimo de lucro, se orienta al desarrollo, la convergencia y la adopción de los estándares de comercio electrónico y servicios web.

Debemos dar mucha importancia a que un protocolo o una tecnología se convierta en un estándar. Esto permite que los proyectos evolucionen de forma independiente a los protocolos o tecnologías.

Los estándares ofrecen un alto grado de compatibilidad entre desarrollos, seguridad, interoperabilidad y escalabilidad de los proyectos sin preocuparnos de los protocolos.

El ejemplo más claro es TCP/IP, un protocolo inventando en los años 70 y que a día de hoy se sigue utilizando masivamente.

Además de ser un estándar, es de código abierto. La **[especificación del protocolo](https://www.ibm.com/developerworks/webservices/library/ws-mqtt/" \t "_blank)** se ha publicado abiertamente con una licencia libre. Fue en el año 2011 cuando las empresas IBM y Eurotech donaron el código MQTT al proyecto Eclipse.

### ¿Es un protocolo seguro?

Siempre debemos partir de la suposición que mientras que una máquina esté conectada a Internet, no se puede garantizar la seguridad al 100%. Partiendo de este hecho, MQTT soporta cifrado mediante SSL.

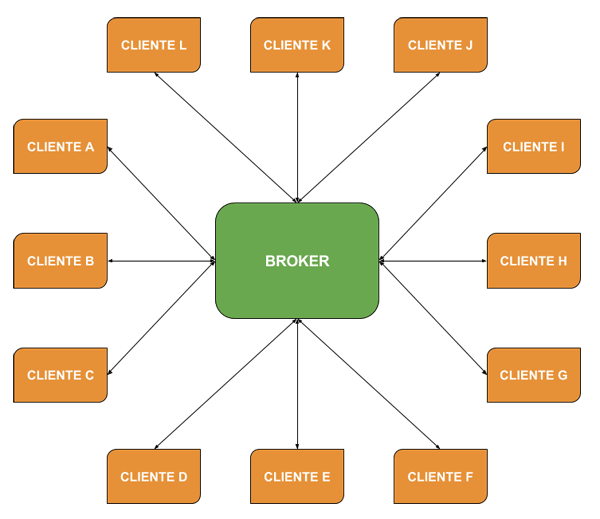
Por lo tanto, podemos enviar datos cifrados a través de la red utilizando este protocolo. Eso sí, se debe pagar un precio.

Cuando se utiliza cifrado SSL se añade una sobrecarga a la red y los microcontroladores, haciendo que MQTT ya no sea un protocolo tan ligero y simple.

## Arquitectura de un sistema MQTT

En esta sección haré un repaso rápido de cómo se estructura un sistema que utilice el protocolo MQTT. Hablaré de la arquitectura típica.

Lo primero es ver cómo se conectan los dispositivos. Utilizan una topología en estrella es decir, todos los clientes se conectan directamente a un punto central que hace de servidor. **En MQTT este servidor se llama Broker**.



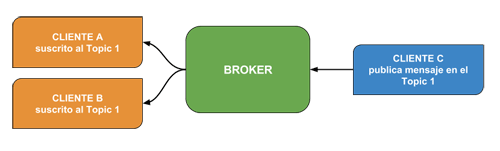
El Broker es la parte central de la arquitectura y todos los mensajes pasan por él.

### Arquitectura publicación/suscripción de MQTT

Se trata de una arquitectura basada en eventos. Cada mensaje se envía a los receptores que se hayan suscrito a una publicación concreta. El Broker se encarga de distribuir los mensajes a los receptores.

El topic es el tema donde se suscriben los receptores para recibir el mensaje. Voy a hacer una comunicación típica para entender el concepto de arquitectura publicación/suscripción.

Un cliente se suscribe a un topic que es como el tema, a quién va dirigido el mensaje. Lo podemos asemejar como el asunto de un email. Un mensaje no tiene destinatario concreto, puede haber uno, muchos o ninguno.



El emisor, el que envía el mensaje, no sabe a quién va dirigido dicho mensaje, sólo el Broker lo sabe. Cuando llega un nuevo mensaje se lo envía a aquellos que se han suscrito al topic.

El mensaje puede ser cualquier cosa, una temperatura, un 1 o un 0, un texto, … El formato es totalmente libre, lo decide el programador y el tamaño máximo depende de la implementación del protocolo MQTT y no de la especificación.

### ¿Qué es el topic?

Como ya he comentado, el topic es el tema del mensaje, a quién va dirigido ese mensaje. Es un concepto que tenemos que tener muy claro así que vamos a verlo con una analogía.

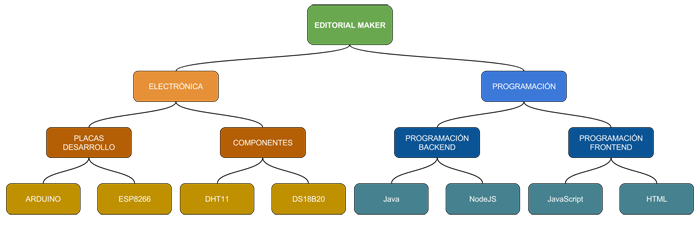
Imagínate una editorial que publica revistas sobre temas técnicos. Tiene dos categorías principales: electrónica y programación.

Dentro de estas categorías, la editorial define unas sub-categorías donde se tratan temas más específicos. Para electrónica tiene placas de desarrollo y componentes. Para programación tiene programación backend y programación frontend.

Por último, están las revistas que van destinadas a algo concreto. Dentro de electrónica/placas de desarrollo hay dos revistas una dedicada a Arduino y otra al ESP8266. En electrónica/componentes hay otras dos revistas dedicadas a los sensores [**DHT11**](https://programarfacil.com/blog/arduino-blog/sensor-dht11-temperatura-humedad-arduino/) y [**DS18b20**](https://programarfacil.com/blog/arduino-blog/ds18b20-sensor-temperatura-arduino/).

En la categoría de programación ocurre lo mismo. Dentro de programación/programación backend se publican dos revistas orientadas a Java y NodeJS. Por último en programación/programación frontend se publican otras dos revistas centradas en [**JavaScript**](https://programarfacil.com/blog/programacion/introduccion-al-javascript/) y [**HTML**](https://programarfacil.com/blog/programacion/introduccion-a-html5/).

El resumen de esta organización sería el siguiente.



Por ejemplo te puedes suscribir a la revista de Arduino y recibir sólo esa revista. Sin embargo, si te suscribes a la categoría de placas de desarrollo, recibirás todas las revistas de esa categoría: Arduino y ESP8266.

De igual forma, si te suscribes a programación recibirás todas las revistas: Java, NodeJS, JavaScript y HTML. Creo que el concepto de lo que es la publicación/suscripción con un topic o tema queda claro.

Pero los topic dentro de MQTT tienen su propia sintaxis. Por ejemplo, si quieres recibir sólo la revista de Arduino tienes que suscribirte a /editorial maker/electrónica/placas desarrollo/arduino.

### Sintaxis de los topic

El símbolo / es un separador de niveles. Siempre se pone para separar cada uno de los niveles.

Cuando trabajamos con un dispositivo del IoT utilizamos otros nombres para los topic. Por ejemplo, podríamos tener algo como esto para medir la temperatura:

* /casa/salon/temperatura
* /casa/cocina/temperatura

Existen **comodines como el símbolo + y #**.

El símbolo + se sustituye por cualquier nivel. Si por ejemplo nos queremos suscribir a todos los topic de temperatura podemos hacerlo de la siguiente manera

**/casa/+/temperatura**

El símbolo + se sustituirá por cada nivel que tenga como nivel superior casa y como nivel inferior temperatura.

El símbolo # también es un comodín. Este símbolo cubre varios nieves aguas abajo. Por ejemplo si quieres suscribirte a todos los mensajes que se envían a casa sólo tienes que hacer lo siguiente

/casa/#

Donde estamos indicando que todos los mensajes que se envíen a cualquier nivel dentro del topic casa lo recibirás.

### Estructura de un mensaje MQTT

Lo más importante dentro del protocolo MQTT son los mensajes. Se envían de forma asíncrona es decir, no hay que esperar respuesta una vez que se envía un mensaje.

Cada mensaje consta de 3 partes:

* Encabezado fijo. Ocupa sólo 2 bytes y es obligatorio enviar esta parte en todos los mensajes.
* Encabezado variable. Ocupa 4 bits y no es obligatorio que esté en todos los mensajes.
* Mensaje o carga útil (del inglés payload). Puede tener un máximo de 256 Mb aunque en implementaciones reales el máximo es de 2 a 4 kB.

Al ser uno de los objetivos consumir el menor ancho de banda, cada bit está estudiado cuidadosamente para que cumpla con este objetivo.

### ¿Cómo funciona la arquitectura MQTT?

Una de las características más importantes es que los clientes o nodos no depende unos de otros ya que no tienen conocimiento de quién está al otro lado. Puede incluso que no haya nadie en el otro extremo.

Esto permite algo muy importantes en proyectos de este tipo: la escalabilidad.

Al contrario de lo que ocurre con el protocolo HTTP, no hay que hacer una petición para recibir información desde un cliente. Cada cliente MQTT abre una conexión permanente TCP con el Broker.

El Broker tiene la capacidad de hacer que los mensajes sean persistentes, guardando el mensaje hasta que se conecte el cliente al que va dirigido. Te recuerdo que el Broker es el único que sabe quién está suscrito a un topic.

### Servicio de Calidad o QoS

He hablado en varias ocasiones de que MQTT es un protocolo fiable. Eso se debe a que tiene implementado un Servicio de Calidad o QoS (del inglés Quality of Service). Este servicio determina cómo se entrega el mensaje a los receptores.

El QoS se especifica en cada mensaje que se envía y puede haber 3 grados de calidad:

* QoS 0: como máximo una vez. Esto implica que puede que no se entregue.
* QoS 1: al menos una vez. Se garantiza la entrega pero puede que duplicados.
* QoS 2: exactamente una vez. Se garantiza que llegará una vez el mensaje.

Utilizar un grado de calidad u otro dependerá de la fiabilidad que queramos tener en nuestro sistema. Sin embargo, debes tener en cuenta que cuanta más calidad menor será el rendimiento.

Ahora que ya tenemos los conocimientos técnicos, vamos a poner en práctica todo esto para montar nuestro sistema basado en el protocolo MQTT.

## Instalación Mosquitto en Raspberry Pi

Para instalar MQTT en una Raspberry Pi voy a utilizar un servidor ampliamente conocido como es [**Eclipse Mosquitto**](https://mosquitto.org/).

Mosquitto es un mediador de mensajes que incluye el protocolo MQTT. Además es de código abierto lo que supone una ventaja para los Makers ya que lo podemos utilizar sin ningún tipo de problema.

Para que el protocolo MQTT esté en constante disponibilidad, es recomendable instalar el broker en un servidor que esté siempre encendido. Tenemos diferentes opciones pero la más interesante es Raspberry Pi por su bajo coste y consumo.

Vamos a ver los pasos que deben seguir para tener operativo nuestro servidor MQTT.

**Debes tener instalado Raspbian y recomendable la versión Jessie. En la página web de la**[**Fundación Raspberry Pi**](https://www.raspberrypi.org/documentation/installation/installing-images/)**tienes un tutorial paso a paso.**

### Paso 1: Instalar Broker MQTT

Abre una terminal en tu Raspberry Pi.

|  |  |
| --- | --- |
|  | sudo wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key |

Añadimos la clave para a una lista para autenticar el paquete que vamos a descargar más tarde.

|  |  |
| --- | --- |
|  | sudo apt-key add mosquitto-repo.gpg.key |

Luego tienes que ir a la siguiente carpeta utilizando el comando cd.

|  |  |
| --- | --- |
|  | cd /etc/apt/sources.list.d/ |

Después descargamos la lista de repositorios de Mosquitto con wget.

Si tu versión de Raspbian es Jessie, ejecuta el siguiente comando.

|  |  |
| --- | --- |
|  | sudo wget http://repo.mosquitto.org/debian/mosquitto-jessie.list |

Si tu versión de Raspbian es Stretch, ejecuta el siguiente comando.

|  |  |
| --- | --- |
|  | sudo wget http://repo.mosquitto.org/debian/mosquitto-stretch.list |

Si utilizas la última versión, Buster, utiliza el siguiente comando.

|  |  |
| --- | --- |
| 1 | sudo wget http://repo.mosquitto.org/debian/mosquitto-buster.list |

Como ves, si utilizas una versión anterior o más reciente, lo único que tienes que hacer es cambiar el nombre la versión.

Ahora para no tener que estar constantemente poniendo la palabra sudo, escribimos lo siguiente en la terminal para ser usuario root.

|  |  |
| --- | --- |
|  | sudo -i |

Este paso no es obligatorio, si no lo haces tendrás que poner sudo en los comandos que van a continuación.

Como ves, la shell cambia de aspecto. A partir de ahora ya no hace falta poner la palabra sudo antes.

Actualizamos la lista de paquetes disponibles y sus versiones.

|  |  |
| --- | --- |
|  | apt-get update |
|  | apt-get install mosquitto |

Con esto ya tendríamos el Broker Mosquitto instalado en nuestra Raspberry Pi. Ahora toca instalar el cliente para hacer las pruebas.

### Paso 2: Instalar cliente MQTT en Raspberry Pi

Este paso va a ser muy sencillo. Si todavía no has cerrado la terminal anterior sólo tendrás que escribir el siguiente comando.

|  |  |
| --- | --- |
|  | apt-get install mosquitto-clients |

Recuerda que si no estás logueado como root tendrás que poner antes del comando sudo.

### Paso 3: Ejemplo con Mosquitto MQTT

Ahora mismo si tu Raspberry Pi está conectada a la red de tu casa, puedes crear un topic accesible desde cualquier cliente MQTT que esté conectado al Broker.

De momento, lo único que queremos es probar que todo está bien instalado así que lo haremos sobre la misma Raspberry Pi.

Pero antes de continuar vamos a ver los dos comandos que vamos a utilizar del cliente MQTT Mosquitto, mosquitto\_sub y mosquitto\_pub.

#### Comando mosquitto\_sub

Este comando nos permite suscribirnos a un topic escuchando y mostrando por pantalla los mensajes enviados a este topic. Puede **[tomar muchos parámetros](https://mosquitto.org/man/mosquitto_sub-1.html" \t "_blank)** pero nosotros nos vamos a centrar en 4.

|  |  |
| --- | --- |
|  | mosquitto\_sub -h BROKER -t TOPIC |

Donde:

* -h: indica que lo que viene después es el host. El host se refiere a la dirección IP o nombre de la máquina en la red del Broker Mosquitto. Si no pasamos este parámetro cogerá por defecto localhost.
* BROKER: dirección IP o nombre de la máquina en la red del Broker Mosquitto.
* -t: indica que lo que viene después es el topic al que queremos suscribirnos.
* TOPIC: nombre del topic al que nos vamos a suscribir.

#### Comando mosquitto\_pub

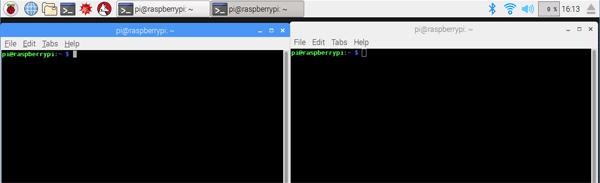
Con este comando podemos publicar mensajes muy simples. Como ocurre con mosquitto\_sub[**puede tomar muchos parámetros**](https://mosquitto.org/man/mosquitto_pub-1.html), este comando pero a nosotros nos importan sólo 6.

|  |  |
| --- | --- |
|  | mosquitto\_pub -h BROKER -t TOPIC -m MENSAJE |

Donde:

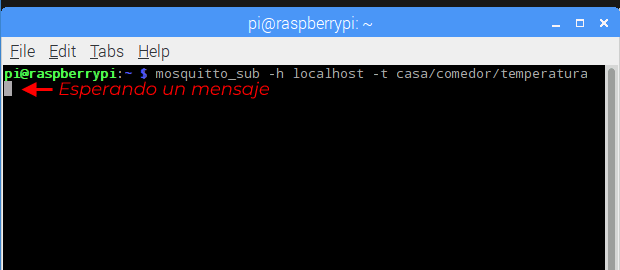
* -h: indica que lo que viene después es el host. El host se refiere a la dirección IP o nombre de la máquina en la red del Broker Mosquitto. Si no pasamos este parámetro cogerá por defecto localhost.
* BROKER: dirección IP o nombre de la máquina en la red del Broker Mosquitto.
* -t: indica que lo que viene después es el topic al que queremos enviar el mensaje.
* TOPIC: nombre del topic al que vamos a enviar el mensaje.
* -m: indica que lo que viene después es el mensaje.
* MENSAJE: el mensaje que queremos enviar. Tiene que ir entre comillas dobles («). Por ejemplo «Temperatura = 25ºC».

Ahora sólo nos queda probar así que abre dos terminales en la Raspberry Pi.



Uno va a ser el que publique y otro el que esté escuchando. Elige el que quieras y escribe el siguiente comando.

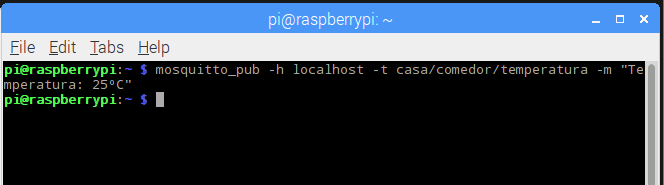
|  |  |
| --- | --- |
|  | mosquitto\_sub -h localhost -t casa/comedor/temperatura |



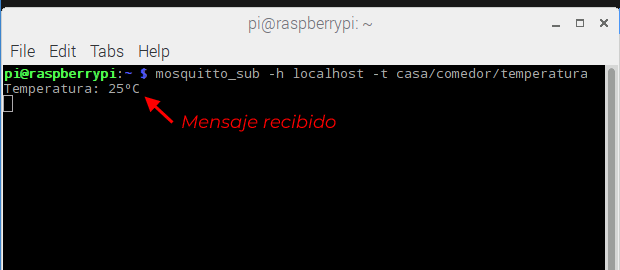
La terminal se queda esperando a recibir algún mensaje.

En el otro terminal escribe el siguiente comando que envía un mensaje al topic casa/comedor/temperatura.

|  |  |
| --- | --- |
|  | mosquitto\_pub -h localhost -t casa/comedor/temperatura -m "Temperatura: 25ºC" |



Ahora si te fijas en la terminal donde estaba esperando un mensaje aparece precisamente el que hemos enviado.



Puedes seguir jugando con los comodines que hemos visto anteriormente. Utiliza los símbolos + y #.

Por último es aconsejable que nuestra Raspberry Pi tenga una IP estática o fija. Esto quiere decir que aunque se desconecte de la red, siempre tendrá la misma IP. Esto lo puedes hacer siguiendo el tutorial de **[ModMyPi](https://www.modmypi.com/blog/tutorial-how-to-give-your-raspberry-pi-a-static-ip-address" \t "_blank)**.

Y así de fácil hemos conseguido crear una red con el protocolo MQTT. Ahora lo que haremos es poder encender un LED conectado a un **[NodeMCU](https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/)** desde la Raspberry Pi.

## Enviando y recibiendo mensajes MQTT con NodeMCU

Lo primero de todo es que tienes que tener una placa con un ESP8266. Yo suelo recomendar NodeMCU que la puedes comprar [**por menos de 8€**](https://amzn.to/2BfLpea). Pero también te puede servir cualquier placa de desarrollo como una Wemos que cuesta también **[menos de 8€](https://amzn.to/2H0wuVp" \t "_blank)**.

Antes de continuar, te recomiendo que leas el artículo [**tutorial paso a paso con NodeMCU**](https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/) y **[cómo programar NodeMCU desde el IDE de Arduino](https://programarfacil.com/esp8266/como-programar-nodemcu-ide-arduino/)**. Es necesario tener listo tanto el IDE como la placa de desarrollo.

Ahora sí, ya podemos empezar a trabajar con NodeMCU y MQTT. Lo primero será instalar la librería.

### Instalando la librería PubSubClient

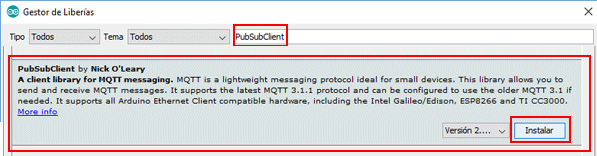
Una de las grandes ventajas que tenemos a la hora de prototipar con Arduino o ESP8266, es que tenemos un montón de librerías que nos hacen la vida mucho más fácil. Si no tienes experiencia con librerías, te aconsejo que eches un vistazo al tutorial donde explico cómo **[instalar librerías en el IDE de Arduino](https://programarfacil.com/blog/arduino-blog/instalar-una-libreria-de-arduino/)**.

[**PubSubClient**](https://pubsubclient.knolleary.net/) es una librería compatible con Arduino y ESP8266. Básicamente hace que nuestra placa se comporte como un cliente MQTT es decir, que podamos publicar mensajes y suscribirnos a un topic o varios para recibir mensajes.

Da lo mismo si utilizas un Arduino o un ESP8266, el código es prácticamente el mismo. La diferencia reside en cómo nos conectamos a la red WiFi, cada placa utiliza su propia librería.

Lo primero es instalar la librería siguiendo. Abre el  la opción del menú

Programa>Incluir Librería>Gestor de librerías y busca PubSubClient



Luego le das a instalar y listo, ya tienes el la librería instalada en tu ordenador y prepara para ser utilizada.

### Enviando un mensaje a través del protocolo MQTT con NodeMCU

Vamos a partir de uno de los ejemplos que vienen dentro de la librería. Lo encontrarás en

Archivo>Ejemplos>PubSubClient>mqtt\_esp8266.

Esta opción te abre el siguiente código

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127 | /\*    Basic ESP8266 MQTT example      This sketch demonstrates the capabilities of the pubsub library in combination    with the ESP8266 board/library.      It connects to an MQTT server then:    - publishes "hello world" to the topic "outTopic" every two seconds    - subscribes to the topic "inTopic", printing out any messages      it receives. NB - it assumes the received payloads are strings not binary    - If the first character of the topic "inTopic" is an 1, switch ON the ESP Led,      else switch it off      It will reconnect to the server if the connection is lost using a blocking    reconnect function. See the 'mqtt\_reconnect\_nonblocking' example for how to    achieve the same result without blocking the main loop.      To install the ESP8266 board, (using Arduino 1.6.4+):    - Add the following 3rd party board manager under "File -> Preferences -> Additional Boards Manager URLs":         http://arduino.esp8266.com/stable/package\_esp8266com\_index.json    - Open the "Tools -> Board -> Board Manager" and click install for the ESP8266"    - Select your ESP8266 in "Tools -> Board"    \*/    #include <ESP8266WiFi.h>  #include <PubSubClient.h>    const char\* ssid = "........";  const char\* password = "........";  const char\* mqtt\_server = "broker.mqtt-dashboard.com";    WiFiClient espClient;  PubSubClient client(espClient);  long lastMsg = 0;  char msg[50];  int value = 0;    void setup() {    pinMode(BUILTIN\_LED, OUTPUT);     // Initialize the BUILTIN\_LED pin as an output    Serial.begin(115200);    setup\_wifi();    client.setServer(mqtt\_server, 1883);    client.setCallback(callback);  }    void setup\_wifi() {      delay(10);    // We start by connecting to a WiFi network    Serial.println();    Serial.print("Connecting to ");    Serial.println(ssid);      WiFi.begin(ssid, password);      while (WiFi.status() != WL\_CONNECTED) {      delay(500);      Serial.print(".");    }        Serial.println("WiFi connected");    Serial.println("IP address: ");    Serial.println(WiFi.localIP());  }    void callback(char\* topic, byte\* payload, unsigned int length) {    Serial.print("Message arrived [");    Serial.print(topic);    Serial.print("] ");    for (int i = 0; i < length; i++) {      Serial.print((char)payload[i]);    }    Serial.println();      // Switch on the LED if an 1 was received as first character    if ((char)payload[0] == '1') {      digitalWrite(BUILTIN\_LED, LOW);   // Turn the LED on (Note that LOW is the voltage level      // but actually the LED is on; this is because      // it is acive low on the ESP-01)    } else {      digitalWrite(BUILTIN\_LED, HIGH);  // Turn the LED off by making the voltage HIGH    }    }    void reconnect() {    // Loop until we're reconnected    while (!client.connected()) {      Serial.print("Attempting MQTT connection...");      if (client.connect("ESP8266Client")) {        Serial.println("connected");        // Once connected, publish an announcement...        client.publish("outTopic", "hello world");        // ... and resubscribe        client.subscribe("inTopic");      } else {        Serial.print("failed, rc=");        Serial.print(client.state());        Serial.println(" try again in 5 seconds");        // Wait 5 seconds before retrying        delay(5000);      }    }  }    void loop() {      if (!client.connected()) {      reconnect();    }    client.loop();      long now = millis();    if (now - lastMsg > 2000) {      lastMsg = now;      ++value;      snprintf (msg, 75, "hello world #%ld", value);      Serial.print("Publish message: ");      Serial.println(msg);      client.publish("outTopic", msg);    }  } |

#### Paso 1: poner ssid, password y mqtt\_server

Sustituye por los valores de tu red WiFi y de tu servidor o Broker MQTT. Como lo tenemos instalado en la Raspberry Pi, lo único que tienes que haces es entrar en una línea de comandos y poner lo siguiente.

|  |  |
| --- | --- |
|  | Ifconfig |

Esto te mostrará algo parecido a lo siguiente.

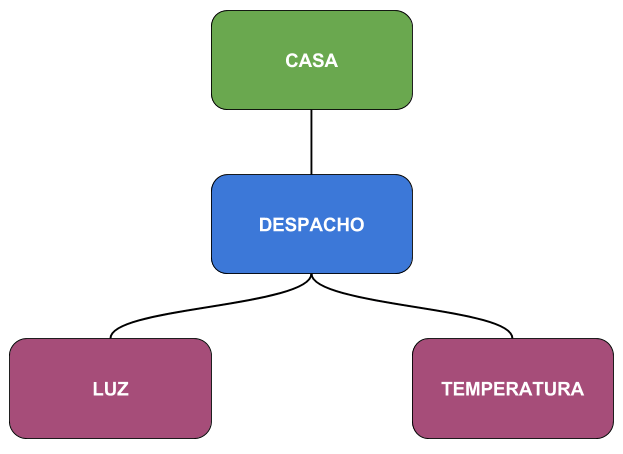
En mi caso, la IP de mi Raspberry Pi es la 192.168.0.167

Por lo tanto el código que debes cambiar quedaría algo parecido a esto.

|  |  |
| --- | --- |
| 1  2  3 | const char\* ssid = "WIFILUIS";  const char\* password = "123456";  const char\* mqtt\_server = "192.168.0.167"; |

#### Paso 2: cambiar el topic donde enviamos el mensaje

Vamos a comenzar cambiando los topic que vienen por defecto. El esquema de nuestra red MQTT será el siguiente:



Antes de continuar déjame que te explique este esquema. El nivel superior es la casa. De aquí partirán el resto de niveles. En este caso sólo tenemos uno que es el despacho pero podría haber más niveles como habitación, cocina, salón, etc…

Del despacho salen dos subniveles: luz y temperatura. Lo lógico es suscribirse a dos topic:

* casa/despacho/luz
* casa/despacho/temperatura

Como hemos visto antes, existe una gran variedad de topic si utilizamos los comodines + y #. Pero en este ejemplo no los vamos a utilizar.

Por otro lado tenemos que tener claro cual es la función de cada uno de estos topic. Por ejemplo, el topic casa/despacho/luz debería ser un actuador y por lo tanto, el dispositivo que se encargue de controlar la luz debería estar suscrito y escuchando los mensajes que llegan a ese topic.

Puede ser un [**Arduino MKR1000**](https://programarfacil.com/blog/arduino-mkr1000/), un [**ESP8266**](https://programarfacil.com/podcast/esp8266-wifi-coste-arduino/) o cualquier otra placa con conexión a la red. En esta placa podemos conectar un relé que controle una luz. Al recibir un mensaje en el topic casa/despacho/luz, actuará en consecuencia.

Si es un 1 activará el **[relé](https://programarfacil.com/blog/rele-y-arduino-mkr1000/)** y si es un 0 lo apagará. Así de sencillo.

A la vez, el mismo dispositivo puede publicar la temperatura en el topic casa/despacho/temperatura. Ahora sólo nos falta saber quién publicará en el topic de la luz y quién recibirá los mensajes de temperatura.

Ese otro dispositivo puede ser cualquier cosa. Imagínate una Raspberry Pi que sea capaz de trabajar como cliente MQTT. Se pueden hacer aplicaciones con muchos lenguajes que tienen librerías o frameworks para conectarnos vía MQTT.

Como hemos visto al configurar Mosquitto con Raspberry Pi, hemos instalado un cliente MQTT. Por lo tanto, Raspberry Pi se comportará como Broker y como cliente.

Seremos capaces de enviar un mensaje a NodeMCU para que encienda el LED. Por otro lado, nos suscribiremos al topic de temperatura para leer la información que envíe NodeMCU.

Todo esto lo podemos hacer fácilmente con Node-RED aunque esto se sale fuera de este tutorial. Si quieres aprender cómo hacerlo, no te pierdas esta **[introducción a Node-RED](https://programarfacil.com/blog/raspberry-pi/introduccion-node-red-raspberry-pi/)**.



Panel de control Acuario con Node-RED, Wemos Mini D1 y protocolo MQTT

La idea final es que un topic puede lanzar un evento como actuador o como sensor. Lo típico es que los actuadores se suscriban y escuchen mensajes. Por otro lado los sensores publicarán en un topic los datos que van adquiriendo.

En este ejemplo, NodeMCU se suscribirá al topic casa/despacho/luz y publicará en casa/despacho/temperatura los datos de temperatura. Raspberry Pi se suscribirá al topic casa/despacho/temperatura y enviará mensajes a casa/despacho/luz.

Ahora sólo nos falta cambiar los topic. Ves a la línea 98 y sustituye esto

|  |  |
| --- | --- |
| 1 | client.publish("outTopic", "hello world"); |

Por esto

|  |  |
| --- | --- |
| 1 | client.publish("casa/despacho/temperatura", "Enviando el primer mensaje"); |

En esta línea estamos diciendo que publique un mensaje (Enviando el primer mensaje) en el topic casa/despacho/temperatura.

Luego en la línea 100 sustituye esto

|  |  |
| --- | --- |
| 1 | client.subscribe("inTopic"); |

Por esto

|  |  |
| --- | --- |
| 1 | client.subscribe("casa/despacho/luz"); |

En esta línea estamos diciendo que se suscriba al topic casa/despacho/luz.

Sólo nos falta cambiar la línea 125. Sustituye esto

|  |  |
| --- | --- |
| 1 | client.publish("outTopic", msg); |

Por esto

|  |  |
| --- | --- |
| 1 | client.publish("casa/despacho/temperatura", msg); |

## Probando la aplicación

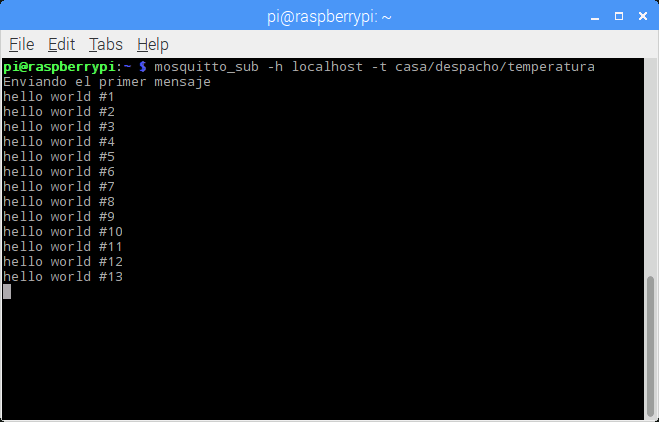
Por último nos queda probar todo el sistema. No te olvides de cargar el código en la placa con las modificaciones hechas. Te dejo aquí una copia donde sólo tienes que poner el nombre de la WiFi, la contraseña y la IP del Broker.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126 | /\*    Basic ESP8266 MQTT example      This sketch demonstrates the capabilities of the pubsub library in combination    with the ESP8266 board/library.      It connects to an MQTT server then:    - publishes "hello world" to the topic "outTopic" every two seconds    - subscribes to the topic "inTopic", printing out any messages      it receives. NB - it assumes the received payloads are strings not binary    - If the first character of the topic "inTopic" is an 1, switch ON the ESP Led,      else switch it off      It will reconnect to the server if the connection is lost using a blocking    reconnect function. See the 'mqtt\_reconnect\_nonblocking' example for how to    achieve the same result without blocking the main loop.      To install the ESP8266 board, (using Arduino 1.6.4+):    - Add the following 3rd party board manager under "File -> Preferences -> Additional Boards Manager URLs":         http://arduino.esp8266.com/stable/package\_esp8266com\_index.json    - Open the "Tools -> Board -> Board Manager" and click install for the ESP8266"    - Select your ESP8266 in "Tools -> Board"    \*/    #include <ESP8266WiFi.h>  #include <PubSubClient.h>    // Update these with values suitable for your network.  const char\* ssid = "NOMBRE-WIFI";  const char\* password = "CONTRASEÑA-WIFI";  const char\* mqtt\_server = "IP-BROKER";    WiFiClient espClient;  PubSubClient client(espClient);  long lastMsg = 0;  char msg[50];  int value = 0;    void setup() {    pinMode(BUILTIN\_LED, OUTPUT);     // Initialize the BUILTIN\_LED pin as an output    Serial.begin(115200);    setup\_wifi();    client.setServer(mqtt\_server, 1883);    client.setCallback(callback);  }    void setup\_wifi() {      delay(10);    // We start by connecting to a WiFi network    Serial.println();    Serial.print("Connecting to ");    Serial.println(ssid);      WiFi.begin(ssid, password);      while (WiFi.status() != WL\_CONNECTED) {      delay(500);      Serial.print(".");    }      Serial.println("");    Serial.println("WiFi connected");    Serial.println("IP address: ");    Serial.println(WiFi.localIP());  }    void callback(char\* topic, byte\* payload, unsigned int length) {    Serial.print("Message arrived [");    Serial.print(topic);    Serial.print("] ");    for (int i = 0; i < length; i++) {      Serial.print((char)payload[i]);    }    Serial.println();      // Switch on the LED if an 1 was received as first character    if ((char)payload[0] == '1') {      digitalWrite(BUILTIN\_LED, LOW);   // Turn the LED on (Note that LOW is the voltage level      // but actually the LED is on; this is because      // it is acive low on the ESP-01)    } else {      digitalWrite(BUILTIN\_LED, HIGH);  // Turn the LED off by making the voltage HIGH    }    }    void reconnect() {    // Loop until we're reconnected    while (!client.connected()) {      Serial.print("Attempting MQTT connection...");      // Attempt to connect      if (client.connect("ESP8266Client")) {        Serial.println("connected");        // Once connected, publish an announcement...        client.publish("casa/despacho/temperatura", "Enviando el primer mensaje");        // ... and resubscribe        client.subscribe("casa/despacho/luz");      } else {        Serial.print("failed, rc=");        Serial.print(client.state());        Serial.println(" try again in 5 seconds");        // Wait 5 seconds before retrying        delay(5000);      }    }  }    void loop() {      if (!client.connected()) {      reconnect();    }    client.loop();      long now = millis();    if (now - lastMsg > 2000) {      lastMsg = now;      ++value;      snprintf (msg, 75, "hello world #%ld", value);      Serial.print("Publish message: ");      Serial.println(msg);      client.publish("casa/despacho/temperatura", msg);    }  } |

Abre una terminal en la Raspberry Pi y escribe el siguiente comando.

|  |  |
| --- | --- |
| 1 | mosquitto\_sub -h localhost -t casa/despacho/temperatura |

Si ya has cargado el código comprobarás cómo estás recibiendo mensajes desde NodeMCU. Tendrás algo parecido a esto.



Abre otro terminal y escribe el siguiente comando.

|  |  |
| --- | --- |
| 1 | mosquitto\_pub -h localhost -t casa/despacho/luz -m 0 |

Si el LED que vienen incorporado en la placa estaba encendido, se apagará.

Si escribes este otro comando

|  |  |
| --- | --- |
| 1 | mosquitto\_pub -h localhost -t casa/despacho/luz -m 1 |