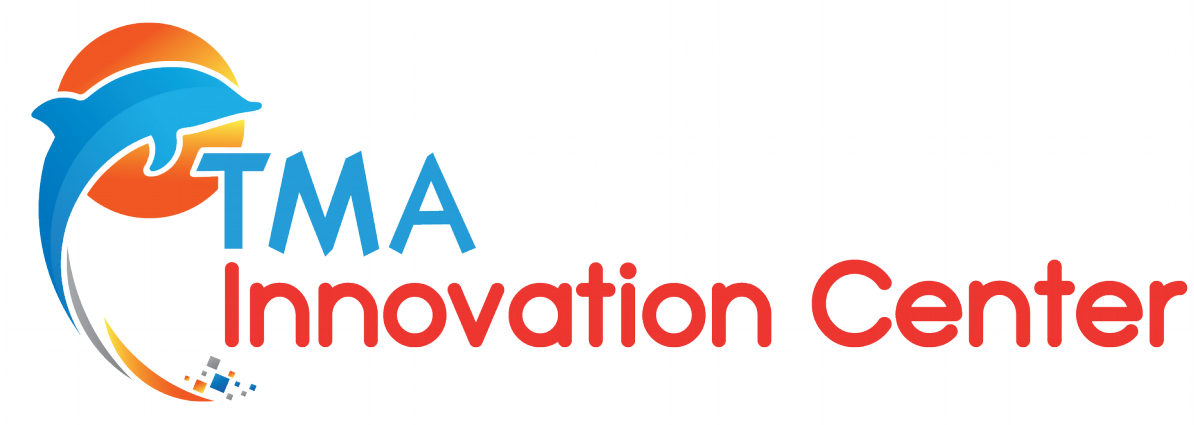
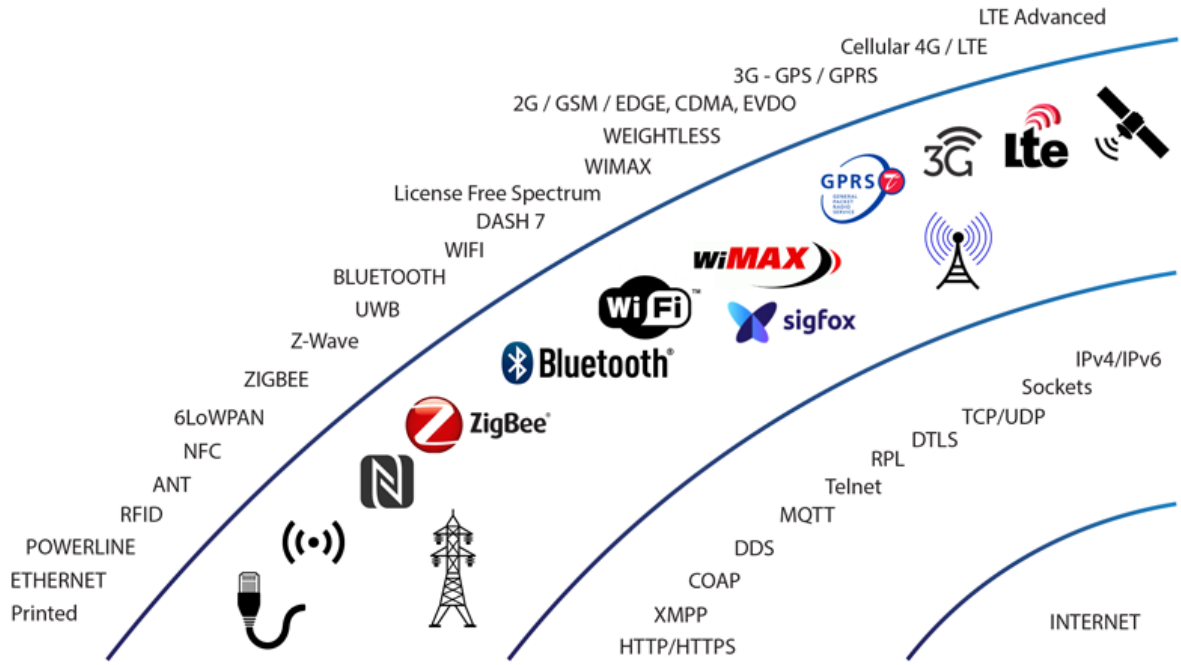



# CONNECTIVITY & COMMUNICATION PROTOCOLS INTRODUCTION

# Table of contents

## I. Overview

Connectivity, it is one of the main things to keep in mind while developing any Internet-of-Things (IoT) project.

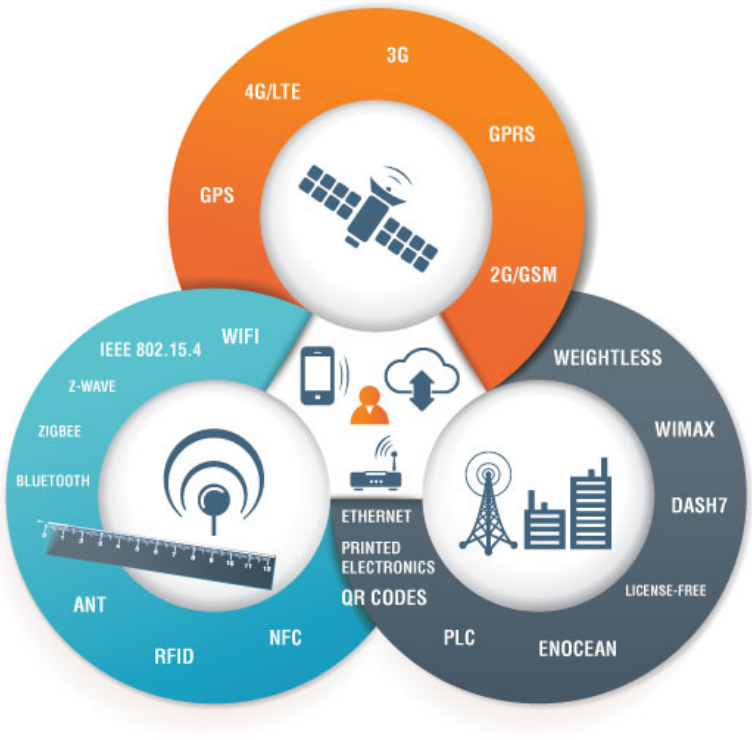



The first few questions that pop up in our head when We embark on any new IoT project are:

1. How do we want it to be connected?
2. Do we have any power or range constraints?
3. What would be our data rate?
4. What network infrastructures are currently available?

We are sure a lot of people would have the same questions when starting any IoT project. Sometimes we have an idea about what communication protocol we want to use, but it doesn't hurt to do our research and make sure that would be the most suitable for our application.

Fortunately, there are a bunch of network infrastructures and communication protocols available. Unfortunately, there are so many that they might render we confused.




In this document, we will discuss all (well, most) of the popular communication protocols and how to pick the most suitable one for our project. We will also go into detail about the pros and cons of each:

1. WiFi
2. Bluetooth
3. RFID and NFC
4. ZigBee
5. LoRa
6. GPRS/EDGE/3G/4G
7. HTTP/WebSocket
8. MQTT
9. CoAP
10. AMQP

## II. WiFi

### 1. What is WiFi?

### a. Overview

Wi-Fi, also spelled Wifi or WiFi, is a popular technology that allows an electronic device to exchange data or connect to the internet wirelessly using radio waves. The Alliance defines Wi-Fi as any "wireless local area network (WLAN) products that are based on the Institute' (IEEE) 802.11 standards". However, since most modern WLANs are based on these standards, the term "Wi-Fi" is used in general English as a synonym for "WLAN". Only Wi-Fi products that complete Wi-Fi Alliance interoperability certification testing successfully may use the "Wi-Fi CERTIFIED" trademark.

**WiFi standards**


**802.11**

In 1997, the Institute of Electrical and Electronics Engineers (IEEE) created the first WLAN standard. They called it 802.11 after the name of the group formed to oversee its development. Unfortunately, 802.11 only supported a maximum network bandwidth of 2 Mbps – too slow for most applications. For this reason, ordinary 802.11 wireless products are no longer manufactured.

**802.11b**

IEEE expanded on the original 802.11 standard in July 1999, creating the 802.11b specification. 802.11b supports bandwidth up to 11 Mbps, comparable to traditional Ethernet.

802.11b uses the same unregulated radio signaling frequency (2.4 GHz) as the original 802.11 standard. Vendors often prefer using these frequencies to lower their production costs. Being unregulated, 802.11b gear can incur interference from microwave ovens, cordless phones, and other appliances using the same 2.4 GHz range. However, by installing 802.11b gear a reasonable distance from other appliances, interference can easily be avoided.

**802.11a**

While 802.11b was in development, IEEE created a second extension to the original 802.11 standard called 802.11a.

Because 802.11b gained in popularity much faster than did 802.11a, some folks believe that 802.11a was created after 802.11b. In fact, 802.11a was created at the same time. Due to its higher cost, 802.11a is usually found on business networks whereas 802.11b better serves the home market.

802.11a supports bandwidth up to 54 Mbps and signals in a regulated frequency spectrum around 5 GHz. This higher frequency compared to 802.11b shortens the range of 802.11a networks. The higher frequency also means 802.11a signals have more difficulty penetrating walls and other obstructions.

Because 802.11a and 802.11b utilize different frequencies, the two technologies are incompatible with each other. Some vendors offer hybrid 802.11a/b network gear, but these products merely implement the two standards side by side (each connected devices must use one or the other).

**802.11g**

In 2002 and 2003, WLAN products supporting a newer standard called 802.11g emerged on the market. 802.11g attempts to combine the best of both 802.11a and 802.11b.

802.11g supports bandwidth up to 54 Mbps, and it uses the 2.4 GHz frequency for greater range. 802.11g is backward compatible with 802.11b, meaning that 802.11g access points will work with 802.11b wireless network adapters and vice versa.

**802.11n**

802.11n (also sometimes known as "Wireless N") was designed to improve on 802.11g in the amount of bandwidth supported by utilizing multiple wireless signals and antennas (called MIMO technology) instead of one.

Industry standards groups ratified 802.11n in 2009 with specifications providing for up to 300 Mbps of network bandwidth. 802.11n also offers somewhat better range over earlier Wi-Fi standards due to its increased signal intensity, and it is backward-compatible with 802.11b/g gear.

**802.11ac**

The newest generation of Wi-Fi signaling in popular use, 802.11ac utilizes dual-band wireless technology, supporting simultaneous connections on both the 2.4 GHz and 5 GHz Wi-Fi bands. 802.11ac offers backward compatibility to 802.11b/g/n and bandwidth rated up to 1300 Mbps on the 5 GHz band plus up to 450 Mbps on 2.4 GHz.

**b. How it works?**


## 2. Usage

This section will help you to use hardware (ESP8266, ESP32) connect and communicate with WiFi network.

**a. Prepare hardware and software**

**Hardware:**

+ 1 x ESP32 board

+ 1 x Micro USB cable

**Software:**

+ Arduino IDE (https://www.arduino.cc)

+ Arduino core for ESP32 (https://github.com/espressif/arduino-esp32)

**And remember to connect your ESP32 board with PC via micro USB cable**

**c. Launch Arduino IDE**

Double click on Arduino IDE shortcut on your desktop.

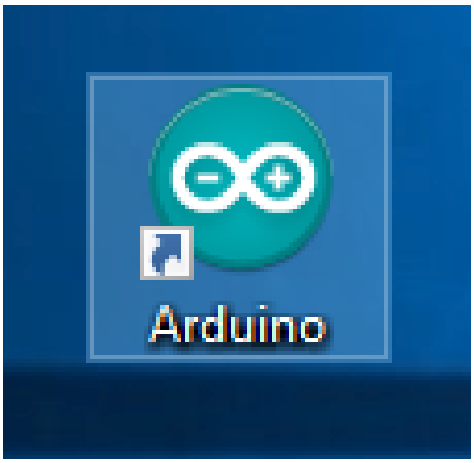



Figure 1: Arduino IDE shortcut

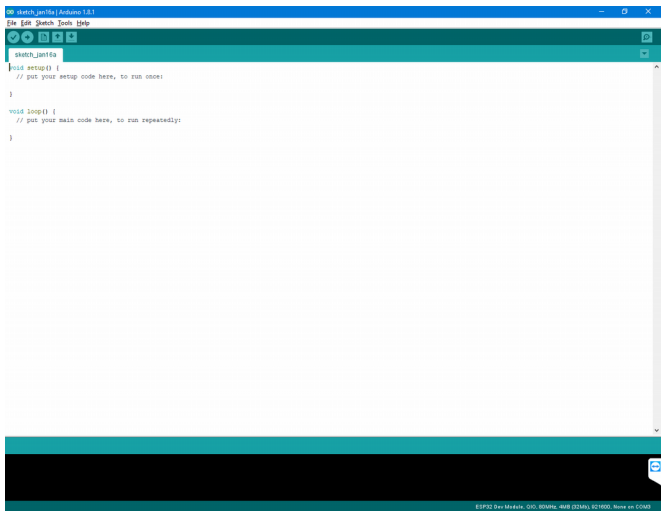
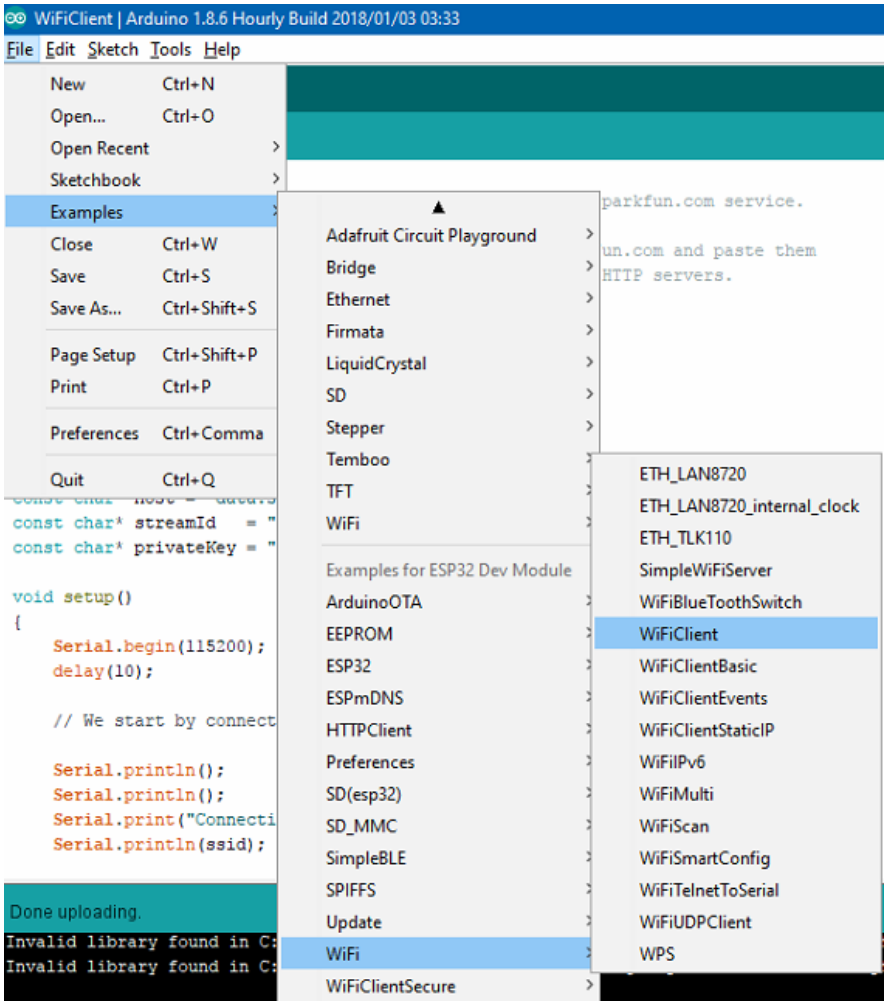


Figure 2: Arduino IDE interface

**d. Open WiFiWebClient samples code**

❖ Navigate to **File\Examples\WiFiClient for ESP Dev Module**




❖ Change **SSID** and **password** of WiFi network.

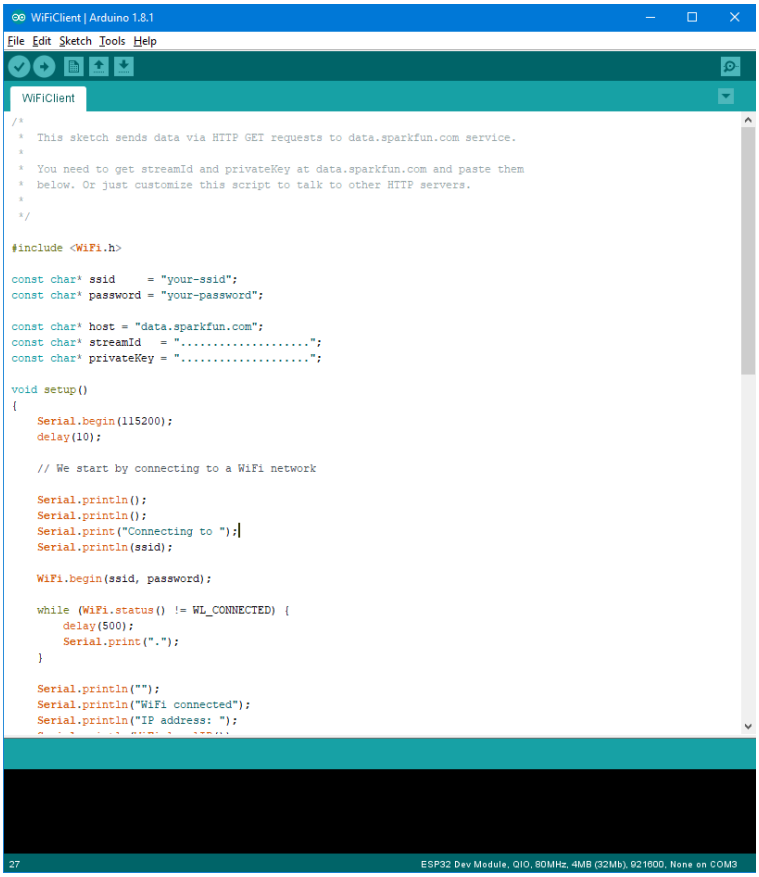



Figure 3: WiFiClient sample code

## e. Upload sketch and check result

- ❖ Configure Board and COM port for Arduino IDE
- ❖ Navigate to Tools\Boards and select your board

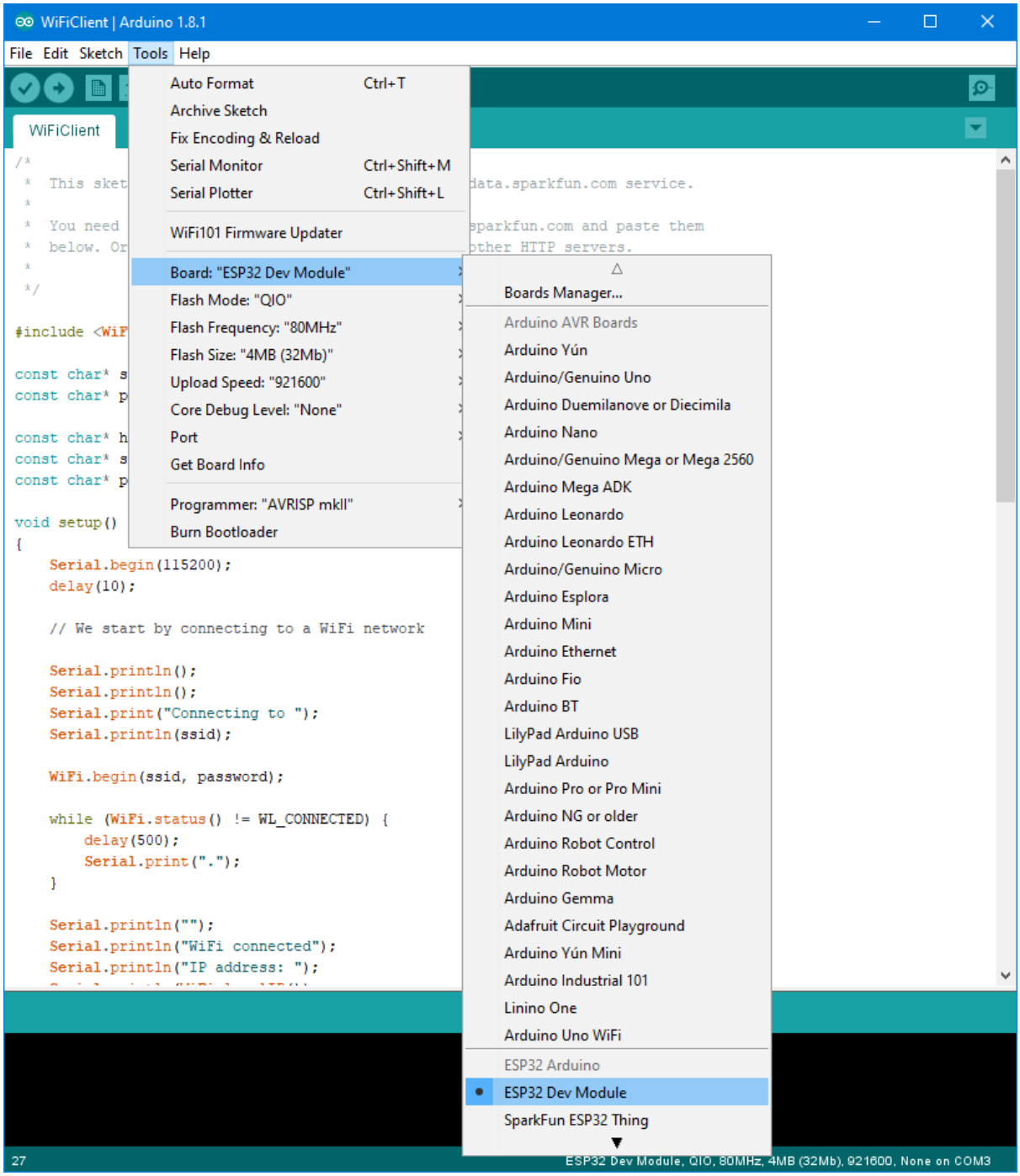



Figure 4: Configure your board

❖ Navigate to **Tools\Port** and select your board's port.

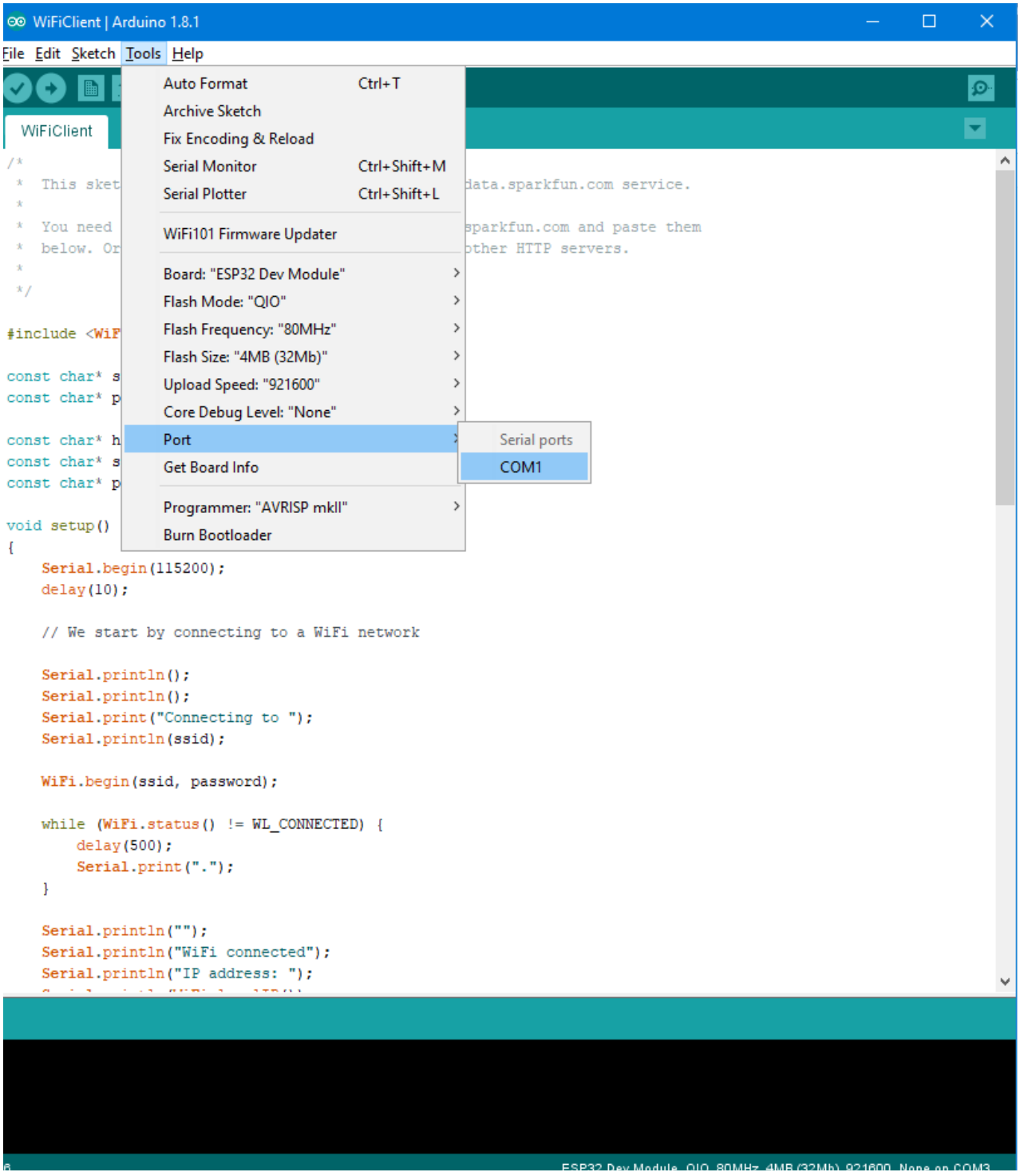



Figure 5: Choose COM port

❖ Compile and Upload sketch

+ Click  to Compile sketch (or use hotkey Ctrl + R)

+ Click  to Upload sketch (or use hotkey Ctrl + U)

**f. Open Serial Monitors and view the result**

❖ Navigate to Tools\Serial Monitors or use hotkey Ctrl + Shift + M to open Serial Monitors.
❖ You can see status and local IP address.

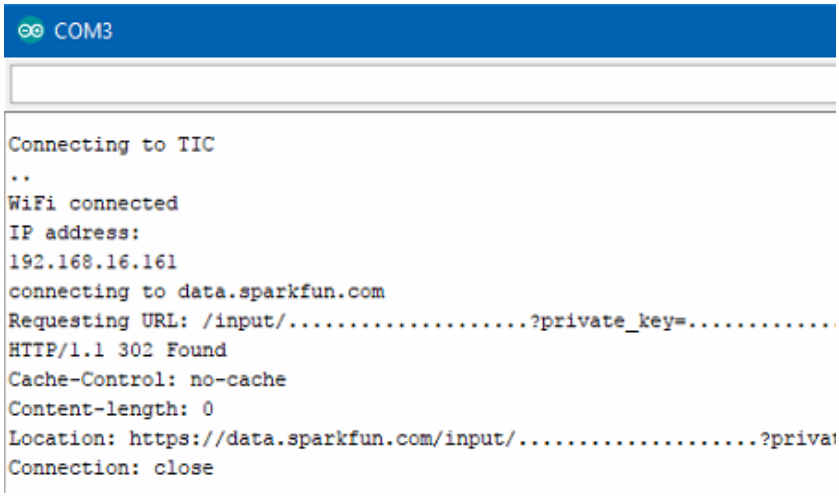



Figure 6: Serial Monitors interface

❖ **Create Web Server to control led via WiFi**

**a. Prepare hardware and software**

**Hardware:**

+ 1 x ESP32 board

+ 1 x Micro USB cable

+ 1 x LED

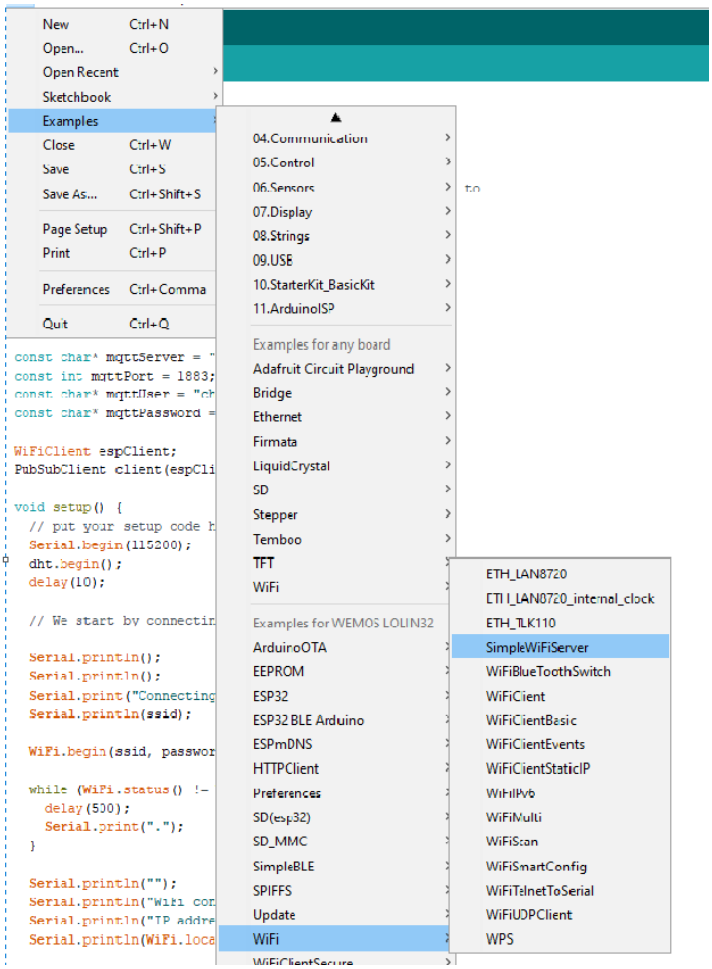
**Software:**

+ Arduino IDE (https://www.arduino.cc)

+ Arduino core for ESP32 (https://github.com/espressif/arduino-esp32)

**b. Open SimpleWiFiServer samples code**

- Connect LED with pin 27 of ESP32.
- Navigate to **File\Examples\WiFi\SimpleWiFiServer**




❖ Change **SSID** and **password** of WiFi network.

```
const char* ssid     = "yourssid";
const char* password = "yourpasswd";
```

❖ Compile and Upload sketch

**c. Check results**

❖ Open Serial Monitors and view the result

```
Connecting to Tin Phat3
.....
WiFi connected.
IP address:
192.168.1.10
```

❖ Access the IP address which you see on Serial Monitors


❖ Now you can control the leds easily from the website

**3. Exercises**

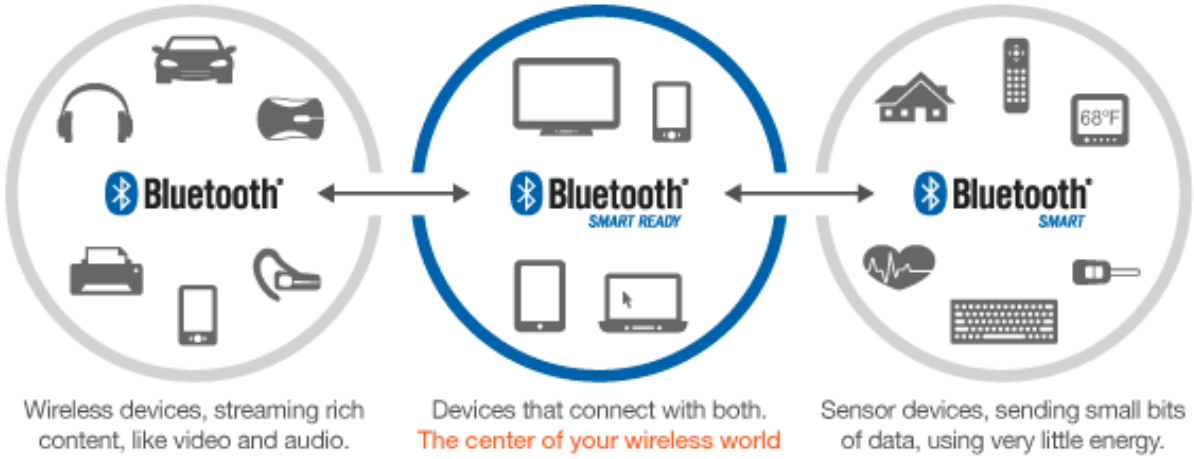
**a. Setup WiFi for Expressif ESP32 hardware.**

**g. Review the samples code and write code to GET/POST data from/to server.**

## III. Bluetooth

### 1. What is Bluetooth?

### a. Overview




Bluetooth is a standard used in links of radio of short scope, destined to replace wired connections between electronic devices like cellular telephones, Personal Digital Assistants (PDA), computers, and many other devices. Bluetooth technology can be used at home, in the office, in the car, etc. This technology allows to the users instantaneous connections of voice and information between several devices in real time. The way of transmission used assures protection against interferences and safety in the sending of information.

Between the principal characteristics, must be named the hardiness, low complexity, low consume and low cost. The Bluetooth is a small microchip that operates in a band of available frequency throughout the world. Communications can realize point to point and point multipoint.

Bluetooth technology can be classified into three types of devices:

- **Bluetooth Classic:** The traditional Bluetooth having a higher throughput, mostly used for wireless audio and file transmission. The 'classic' radio has support for Bluetooth Smart.
- **Bluetooth Smart:** Bluetooth Low Energy has been branded as Bluetooth Smart and transmits just state information. It was designed specifically for applications with low-duty cycles (i.e., the radio is effectively on for a short period of time). Bluetooth Smart devices cannot communicate with Bluetooth Classic devices.
- **Bluetooth SmartReady:** These devices are essentially the "hub" devices such as computers, smartphones, etc. They support both the "classic" and "smart" devices, just as our smartphones can connect to a Bluetooth speaker to transmit audio and also communicate to a fitness tracker.

**Bluetooth versions**

**Bluetooth 1.0 and 1.0B**

Versions 1.0 and 1.0Bhad many problems, and manufacturers had difficulty making their products interoperable. Versions 1.0 and 1.0B also included mandatory Bluetooth hardware device address (BD_ADDR) transmission in the Connecting process (rendering anonymity impossible at the protocol level), which was a major setback for certain services planned for use in Bluetooth environments.

**Bluetooth 1.1**

Many errors found in the v1.0B specifications were fixed.

Added possibility of non-encrypted channels.

Received Signal Strength Indicator (RSSI).

**Bluetooth 1.2**

Faster Connection and Discovery

Adaptive frequency-hopping spread spectrum (AFH), which improves resistance to radio frequency interference by avoiding the use of crowded frequencies in the hopping sequence.

Higher transmission speeds in practice than in v1.1, up to 721 kbit/s.

Extended Synchronous Connections (eSCO), which improve voice quality of audio links by allowing retransmissions of corrupted packets, and may optionally increase audio latency to provide better concurrent data transfer.

Host Controller Interface (HCI) operation with three-wire UART.

Introduced Flow Control and Retransmission Modes for L2CAP.

**Bluetooth 2.0 + EDR**

The main difference is the introduction of an Enhanced Data Rate (EDR) for faster data transfer.

**Bluetooth 2.1 + EDR**

The headline feature of v2.1 is secure simple pairing (SSP): this improves the pairing experience for Bluetooth devices, while increasing the use and strength of security.

**Bluetooth 3.0 + HS**

Bluetooth v3.0 + HS provides theoretical data transfer speeds of up to 24 Mbit/s, though not over the Bluetooth link itself. Instead, the Bluetooth link is used for negotiation and establishment, and the high data rate traffic is carried over a inimizet 802.11 link.

**Bluetooth 4.0 + LE**

Includes Classic Bluetooth, Bluetooth high speed and Bluetooth low energy protocols. Bluetooth high speed is based on Wi-Fi, and Classic Bluetooth consists of legacy Bluetooth protocols.

**Bluetooth 4.1**

Bluetooth 4.1 improve consumer usability. These include increased co-existence support for LTE, bulk data exchange rates—and aid developer innovation by allowing devices to support multiple roles simultaneously.[74]

**Bluetooth 4.2**

Low Energy Secure Connection with Data Packet Length Extension

Link Layer Privacy with Extended Scanner Filter Policies

Internet Protocol Support Profile (IPSP) version 6 ready for Bluetooth Smart things to support connected home

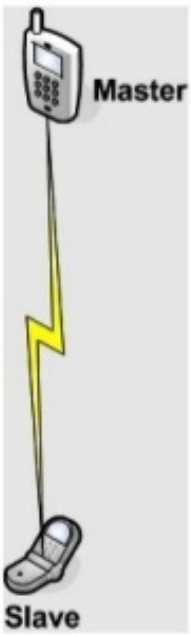
Older Bluetooth hardware may receive 4.2 features such as Data Packet Length Extension and improved privacy via firmware updates.

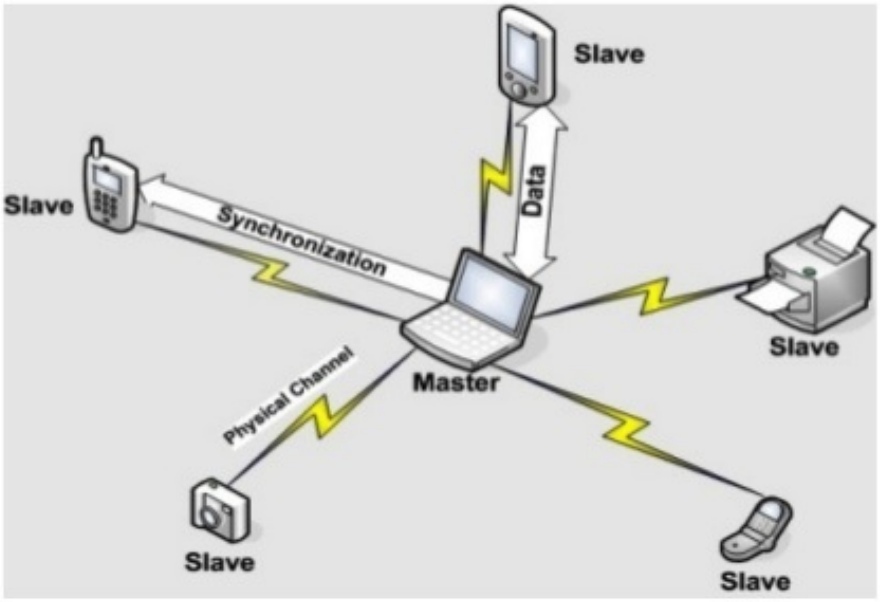
**Bluetooth 5**

Bluetooth 5 focused on emerging Internet of Things technology.

Bluetooth 5 provides, for BLE, options that can double the speed (2 Mbit/s burst) at the expense of range, or up to fourfold the range at the expense of data rate, and eightfold the data broadcasting capacity of transmissions, by increasing the packet lengths. The increase in transmissions could be important for Internet of Things devices, where many nodes connect throughout a whole house. Bluetooth 5 adds functionality for connectionless services such as location-relevant navigation of low-energy Bluetooth connections.

**h. How it works?**




**4. `Usage**

This section will help you to use hardware (ESP32) connect and communicate with other devices via Bluetooth connection.

**a. Prepare hardware and software**

**Hardware:**

+ 2 x ESP32 board

+ 2 x micro USB cable

**Software:**

+ Arduino IDE

+ Arduino cores for ESP32 board.

**And connect two board with PC**

**i. Create Bluetooth server**

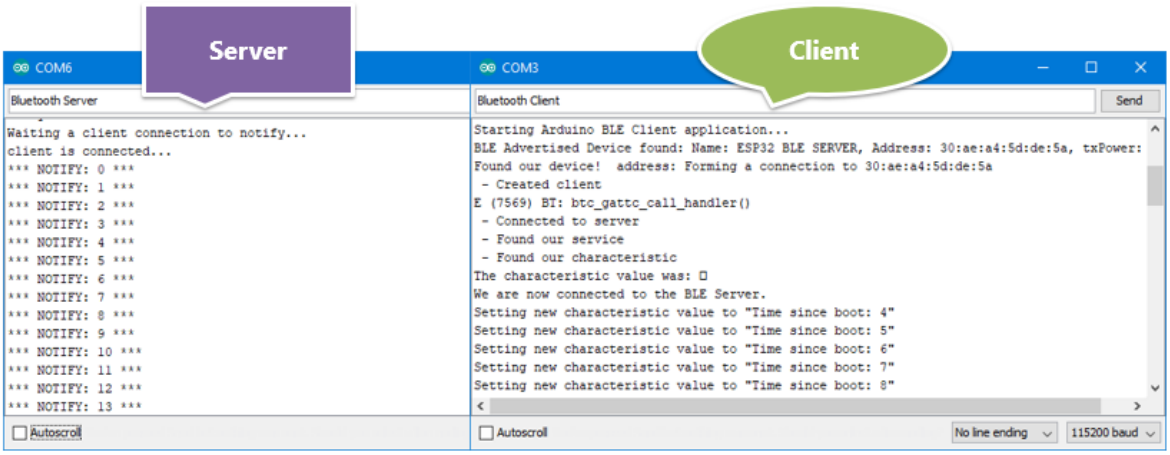
- Checkout materials and samples code from gitlab
- Navigate to Samples code folder and open **BluetoothServer** sketch
- Compile and Upload sketch to ESP32 board.

**j. Create Bluetooth client**

- Navigate to Samples code folder and open **BluetoothClient** sketch
- Compile and Upload sketch to ESP32 board.

**k. Check status and logs**

❖ Open two Serial Monitors you can see the status and logs




## ❖ Connect ESP32 with Smartphone via Bluetooth

This section will help you to use hardware (ESP32) connect and communicate with other devices via Bluetooth connection.

### a. Prepare hardware and software

**Hardware:**

+ 1 x ESP32 board

+ 1 x micro USB cable

+ 1 x Smartphone

**Software:**

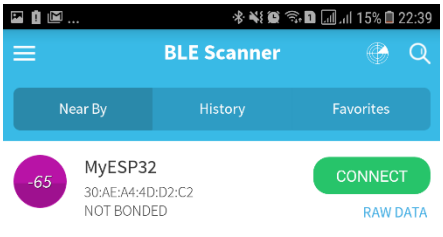
+ Arduino IDE

+ Arduino cores for ESP32 board.

### b. Open BLE_write samples code

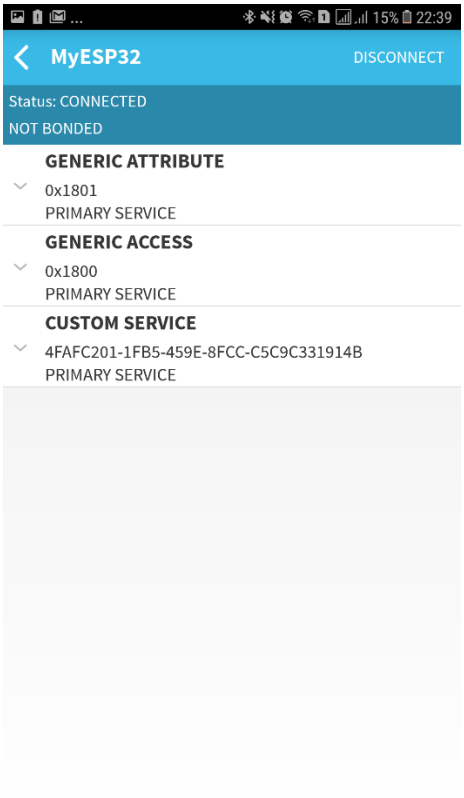
- Navigate to **File\Examples\ESP32 BLE Arduino\ BLE_write**
- Compile and Upload sketch to ESP32 board.
- Download and install an BLE scanner app in your phone


- Scan for BLE devices in the app
- Connect to MyESP32

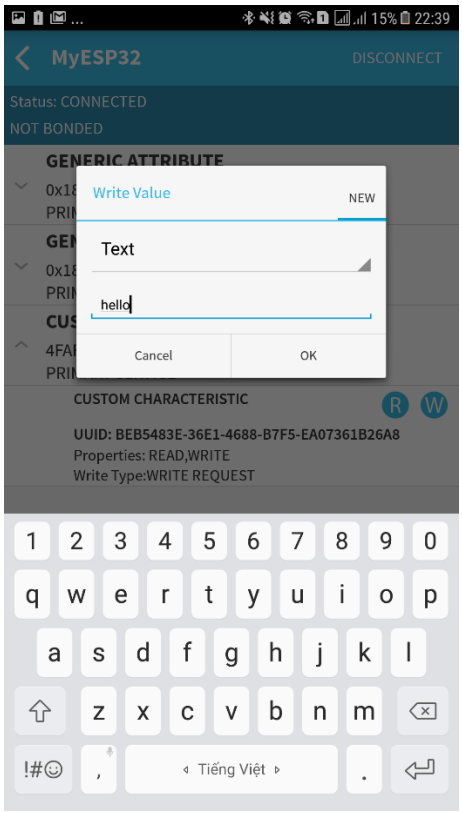




- Go to CUSTOM CHARACTERISTIC in CUSTOM SERVICE and write something




-




d. **Check result**


## 5. Exercises

### a. Setup Bluetooth for Expressif ESP32 hardware.

### l. Review the samples code and write code to send/receive data between two Bluetooth board.

## IV. RFID/NFC

### 1. What is RFID?

### a. Overview


RFID (Radio Frequency Identification) is a technology that uses electromagnetic fields to identify objects in a contactless way; it is also called proximity identification. There are 2 elements in RFID communications: the RFID module (or reader/writer device) and an RFID card (or tag).

**Tags**

The tags are the end points in an RFID system. They store identity information along with other information as required by the purpose of the tag. There are two types of tags:

- **Active Tags:** These tags have an on-board power source of some sort, usually a battery, which means they can transmit stronger signals and therefore have more range. This type of tag can periodically transmit a signal irrespective of a reader.

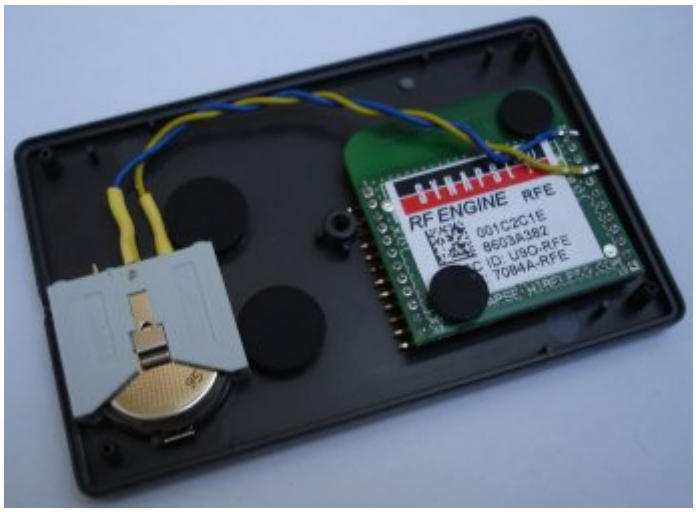



*Figure 7: Active RFID tag*

- **Passive Tags:** These tags do not have any internal power source and get activated in the vicinity of a reader. Your metro or bus pass is generally a

passive tag, which gets activated when you touch it to the reader. These tags harvest the radio energy transmitted by the reader.

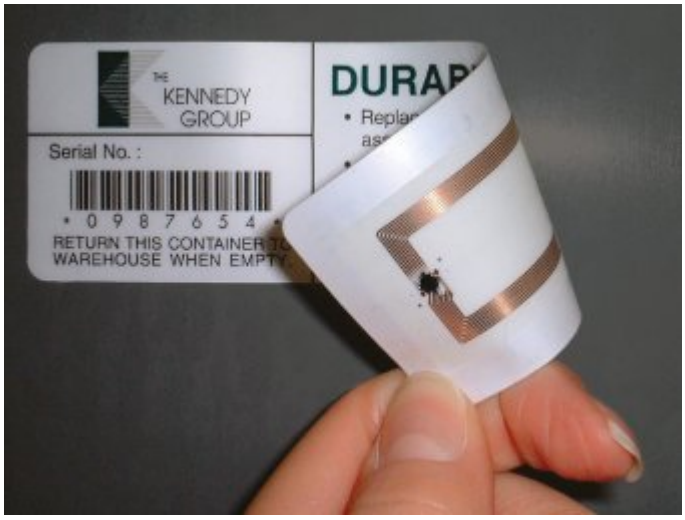



*Figure 8: Passive RFID tag*

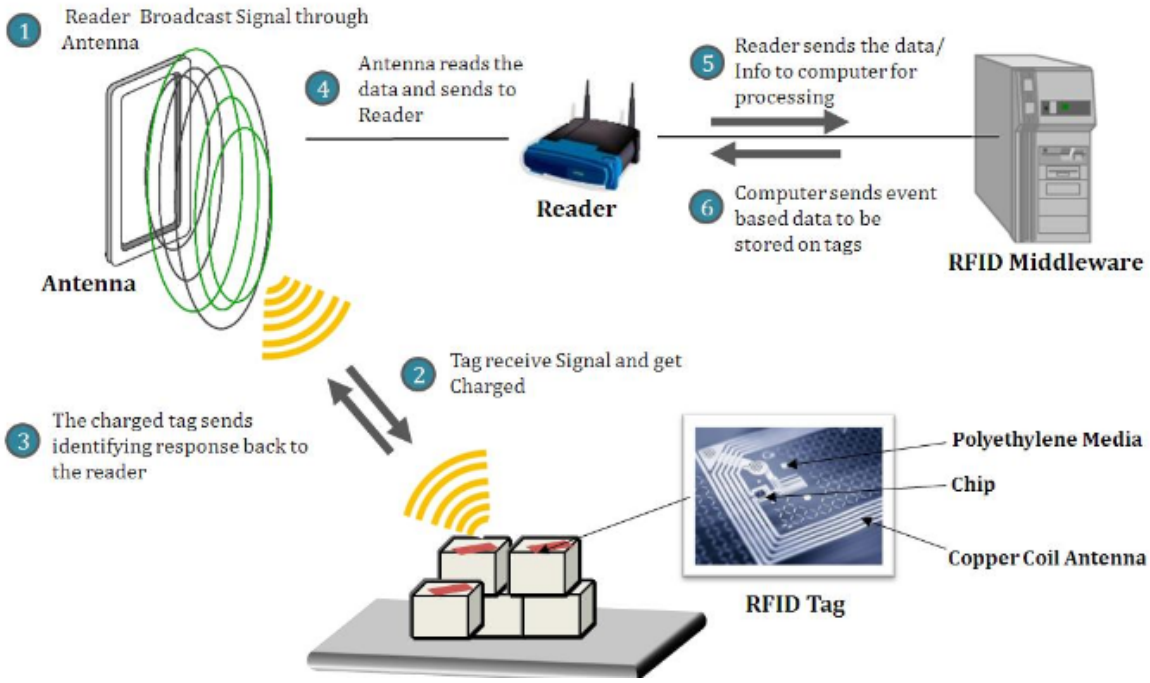
**Readers/Writers**

Readers have a similar construction to an RFID tag. They have an antenna to receive and transmit signals to/from the tags. They might be either battery powered or plugged in to a wall outlet, as a reader requires strong RF signals to activate the tag (for passive) when it comes in the vicinity of the reader. The reader is connected to a reader controller, which manages the information read by the reader. The reader may also write or update a tag depending on the application. For example, the reader in subway stations is at the entry point. When the rider places a card (tag) on the reader, it reads the available money in the card and grants the user entry. At the passenger's exit, it calculates the fare and updates the amount on the card.

RFID tags can have three main components:

- **An Integrated Circuit (IC)** for storing identity information, processing it and modulating/demodulating the RF signals
- **An antenna** to receive and send the radio signals
- **A power source** (battery) if an active tag

**m. How it works?**




## 6. What is NFC?

### a. Overview

Near-Field Communication (NFC) is an RF-based communication protocol. It is a subset of the RFID protocol, which is why it is similar to RFID but has significant differences. NFC has also become a very popular technology, and 9 out of 10 smartphones today are shipped with NFC capabilities. This has enabled contact-less payments such as Apple Pay and Google Wallet.

NFC smartphones pass along information from one smartphone to the other by tapping the two devices together, which turns sharing data such as contact info or photographs into a simple task. Applications like Android Beam facilitate this, whereas in Apple phones NFC can only be used for Apple Pay as of now.

NFC is being used to open car doors in NFC-enabled car keys with car manufacturers such as BMW. You might have also come across advertisements and marketing content that requires you to tap or wave your NFC-enabled smartphone to download an application or get more information about a product.

NFC is designed for very short-range communication (a few centimeters). It is one of the most power-efficient communication protocols. NFC also operates in the 13.56 MHz band, which is used by high-frequency RFID as well. Therefore, NFC can also read HF RFID tags.

**NFC devices**

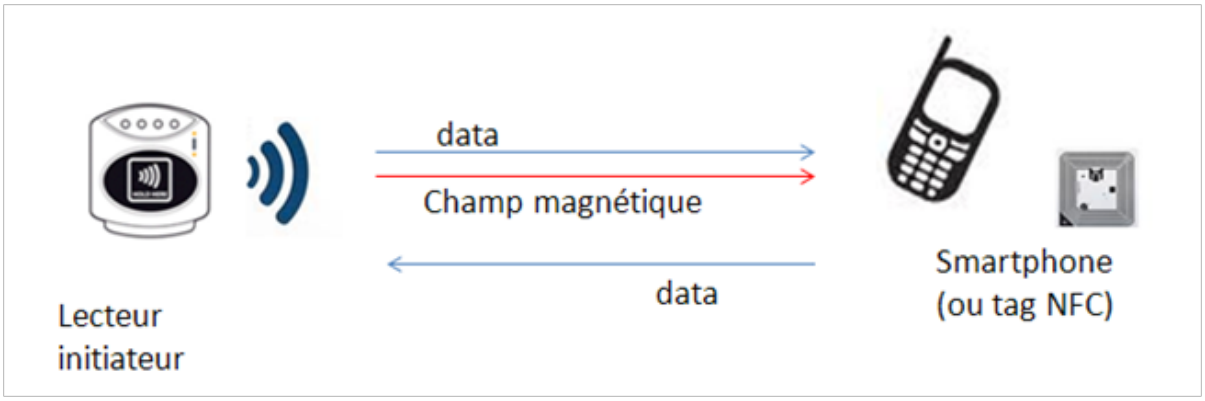
NFC has two types of devices:

- **Initiator:** The device that initiates the communication is labeled as the initiator. It actively generates an RF field that can power the passive target.
- **Target:** This is the device that receives the information from the initiator. The target can either be passive (in the case of simple NFC tags) or active for peer to peer communication, such as in smartphones.

**n. How it works?**

NFC working with three modes:

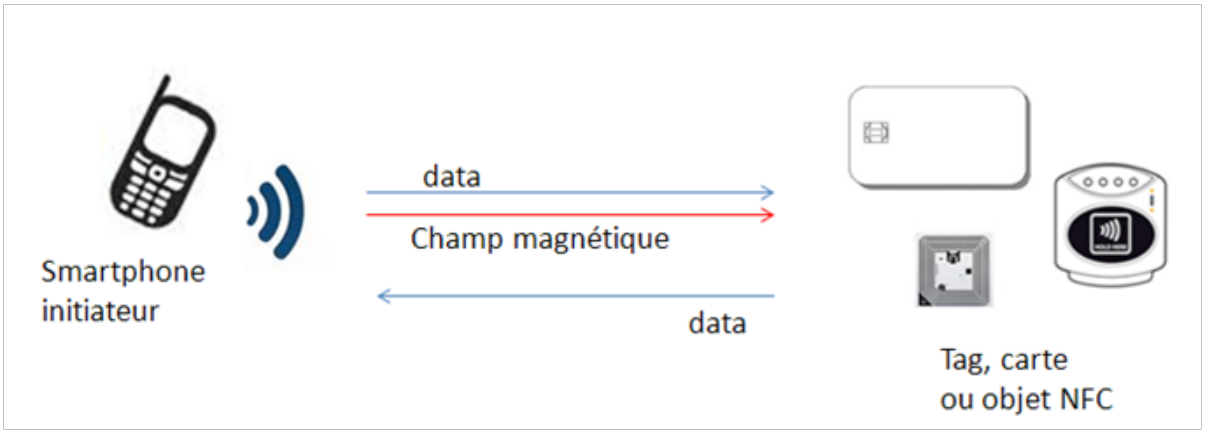
❖ **Card emulation mode**




In card emulation mode, the "NFC Device" behaves like a contactless smart card. It is functioning as a target in a passive mode.

While a contactless card is powered by the magnetic field generated by the interrogator, an "NFC Device" may require more energy to operate. Indeed, an NFC application on a mobile phone, a tablet or a consumer device may benefit from other features than just NFC (screen, applications, security, internal communications, etc.). Access to these features requires an internal power source, a battery or power supply.

❖ **Reader mode**

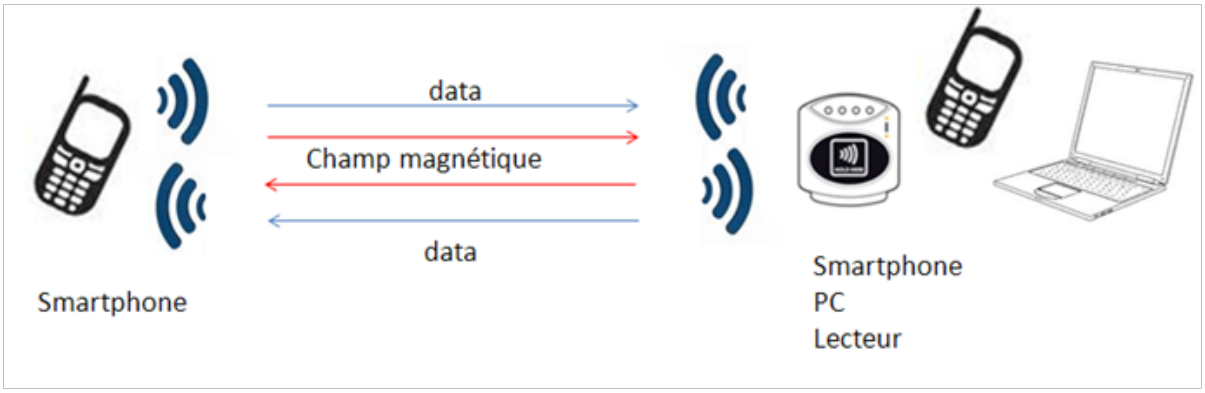



The “NFC Device” in reader mode behaves like a simple contactless card reader. It initiates communication by generating a magnetic field and then sending a command to the target. The target responds to the interrogator by retro-reflecting the incident wave.

The specificity of NFC operating modes is that the target can be not only a tag or a contactless card, but also an “NFC Device” that behaves like a contactless card (in card emulation mode).

Usages of reader mode are principally information reading, when “NFC Devices” is used to read data by waving it in front of electronic labels available on streets, bus stops, sightseeing monuments, ad banners, parcels, products or on business cards (vCard).

❖ **Peer-to-Peer mode**




This mode allows two “NFC Devices” with the same NFC performance to exchange the data with each other alternately. Each of these devices supports both interrogator and target communication modes, sending or receiving by turns the data.

Communication in peer-to-peer mode is slower than in conventional reader / card emulation mode, because of the management of a heavier protocol, which is necessary for the repartition of roles between the two “NFC Devices.”

## 7. Usage

This example will help you to read the information of RFID tags.

### a. Prepare hardware and software

**Hardware:**

+ 1 x Expressif ESP32 board

+ 1 x Micro USB cable

+ 1 x RFID – MFRC522 module

+ 1 x RFID tag or label

+ 7 x Female – female jumper wires

**Software:**

+ Arduino IDE

+ RFID RC522 library ()

## o. Connect RFID module with ESP32

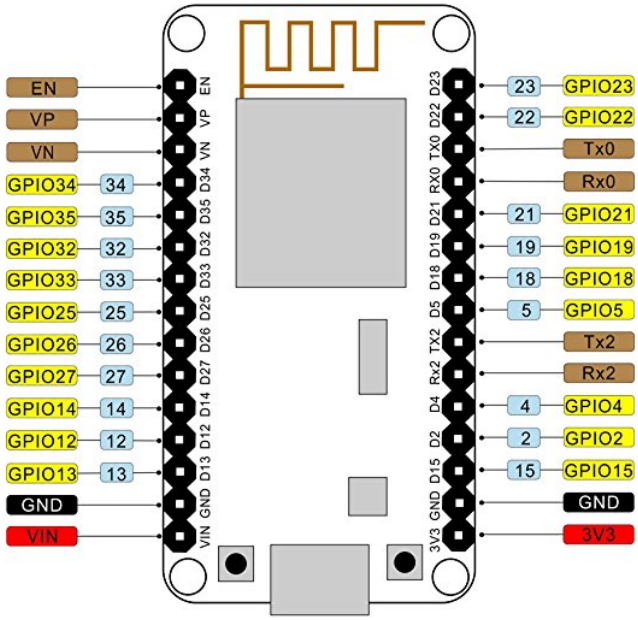



*Figure 9: ESP32 pinout*

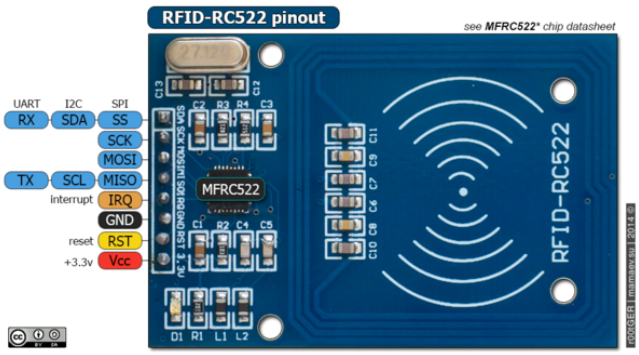



*Figure 10: RC522 pinout*

| RC522 | ESP32 |
|---|---|
| SS | 5 |
| SCK | 18 |
| MOSI | 23 |
| MISO | 19 |
| IRQ | NC |
| GND | GND |
| RST | 0(EN) |
| VCC | 3v3 |

**p. Upload RFIDReader sketch**

- Checkout materials and samples code from gitlab
- Navigate to Samples code and open **RFIDReader** sketch
- Compile and Upload sketch to ESP32 board.

**q. Check result**

- Use RFID tag to touch the RFID module you can see the informations of RFID tag in Serial Monitors.

## 8. Exercises

**a. Setup RFID/NFC for Arduino/Expressif ESP32 board.**

**r. Review the samples code and write code to read data from NFC tag.**

## V. Zigbee

### 1. What is Zigbee?

### a. Overview

The ZigBee protocol was engineered by the ZigBee Alliance, a non-profit consortium of leading semiconductor manufacturers, technology providers, OEMs and end-users worldwide. The ZigBee protocol carries all the benefits of the 802.15.4 protocol with added networking functionality. The 802.15.4 specification was developed at the Institute of Electrical and Electronics Engineers (IEEE). The specification is a packet-based radio protocol that meets the needs of low-cost, battery-operated devices. The protocol allows devices to intercommunicate and be powered by batteries that last years instead of hours.


ZigBee can be implemented in mesh networks larger than is possible with Bluetooth. ZigBee compliant wireless devices are expected to transmit 10-75 meters, depending on the RF environment and the power output consumption required for a given application, and will operate in the unlicensed RF worldwide (2.4GHz global, 915MHz Americas or 868 MHz Europe). The data rate is 250kbps at 2.4GHz, 40kbps at 915MHz and 20kbps at 868MHz. IEEE and ZigBee Alliance have been working closely to specify the entire protocol stack. IEEE 802.15.4 focuses on the specification of the lower two layers of the protocol (physical and data link layer). On the other hand, ZigBee Alliance aims to provide the upper layers of the protocol stack (from network to the application layer) for interoperable data networking, security services and a range of wireless home and building control solutions, provide interoperability compliance testing, marketing of the standard, advanced engineering for the evolution of the standard.

**ZigBee features:**

- ✓ Low duty cycle – provides long battery life

- ✓ Low latency
- ✓ Support for multiple network topologies: static, dynamic, star and mesh
- ✓ Direct Sequence Spread Spectrum (DSSS)
- ✓ Up to 65,000 nodes on a network
- ✓ 128-bit AES (Advanced Encryption Standard) – provides secure connections between devices
- ✓ Collision avoidance
- ✓ Link quality indication
- ✓ Clear channel assessment
- ✓ Retries and acknowledgements
- ✓ Support for guaranteed time slots and packet freshness

**Zigbee Devices**

There are three types of devices as defined by the Zigbee protocol:

- Zigbee Coordinator
- Zigbee Router
- Zigbee End Device (Node)

**Zigbee Coordinator**

The coordinator is the brains of the Zigbee network, it commissions devices to the network, stores the security keys and also bridges to other networks. There is only one Zigbee Coordinator in any network.

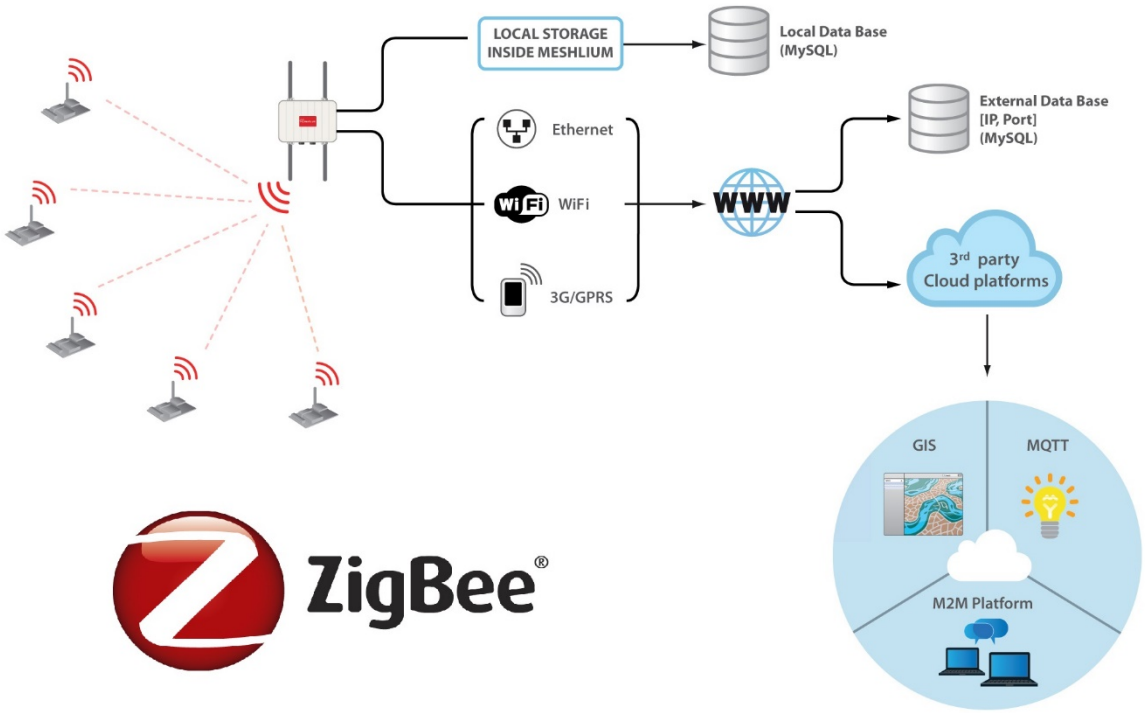
**Zigbee Router**

Zigbee Networks may have several routers to serve as intermediate routers or to transmit data within the network.

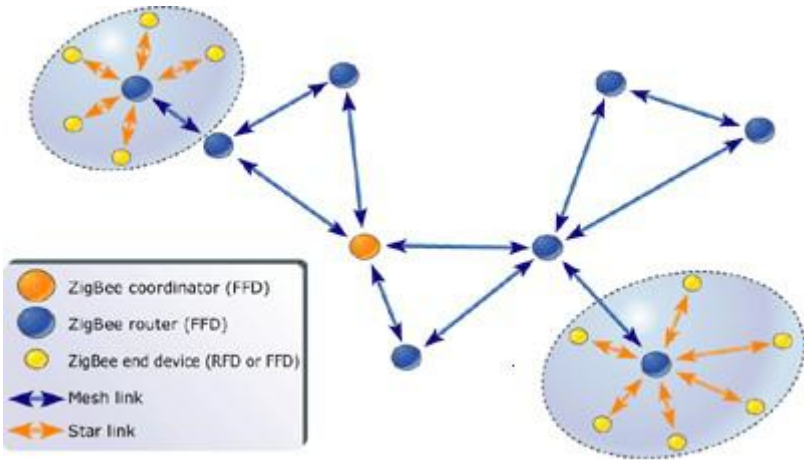
**Zigbee End Device**

The end device can only talk to the parent node ( router or coordinator). It cannot talk to other end devices directly. Similar to Thread, these devices are designed by keeping in mind that they spend most of their life times in sleep mode and only wake up to transmit data to the parent.

**s. How it works?**




ZigBee basically uses digital radios to allow devices to communicate with one another. A typical ZigBee network consists of several types of devices. A network coordinator is a device that sets up the network, is aware of all the nodes within its network, and manages both the information about each node as well as the information that is being transmitted/received within the network. Every ZigBee network must contain a network coordinator. Other Full Function Devices (FFD's) may be found in the network, and these devices support all of the 802.15.4 functions. They can serve as network coordinators, network routers, or as devices that interact with the physical world. The final device found in these networks is the Reduced Function Device (RFD), which usually only serve as devices that interact with the physical world. An example of a ZigBee network is shown in figure below.




The figure above introduces the concept of the ZigBee network topology. Several topologies are supported by ZigBee, including star, mesh, and cluster tree. Star and mesh networking are both shown in the figure above. As can be seen, star

topology is most useful when several end devices are located close together so that they can communicate with a single router node. That node can then be a part of a larger mesh network that ultimately communicates with the network coordinator. Mesh networking allows for redundancy in node links, so that if one node goes down, devices can find an alternative path to communicate with one another.

## VI. Lora

### 1. What is Lora?

### a. Overview


LoRa technology was developed by a company called Semtech and it is a new wireless protocol designed specifically for long-range, low-power communications. LoRa stands for Long Range Radio and is mainly targeted for M2M and IoT networks. This technology will enable public or multi-tenant networks to connect a number of applications running on the same network.

LoRa Alliance™

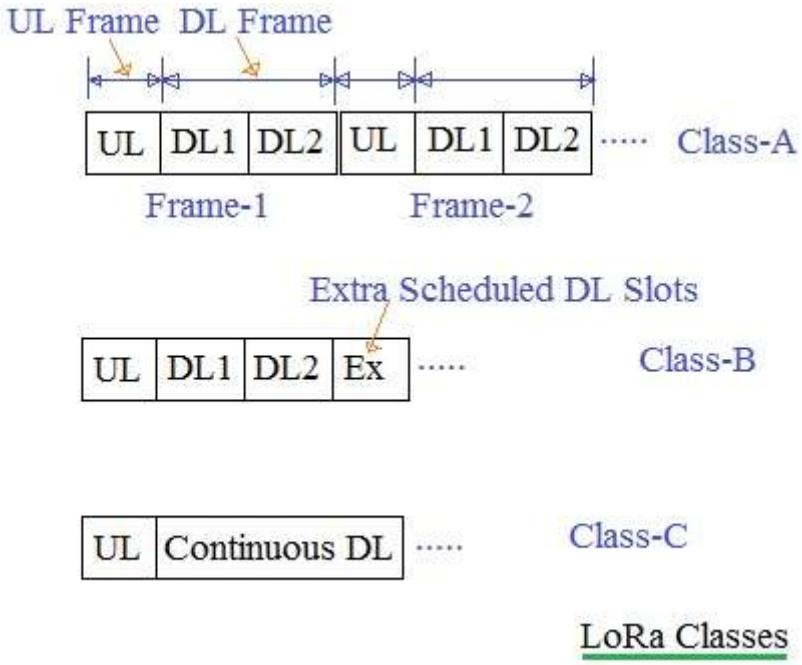
LoRa Alliance was formed to standardize LPWAN (Low Power Wide Area Networks) for IoT and is a non-profit association which features membership from a number of key market shareholders such as CISCO, actility, MicroChip, IBM, STMicro, SEMTECH, Orange mobile and many more. This alliance is key to providing interoperability among multiple nationwide networks.

Each LoRa gateway has the ability to handle up to millions of nodes. The signals can span a significant distance, which means that there is less infrastructure required, making constructing a network much cheaper and faster to implement.

LoRa also features an adaptive data rate algorithm to help maximize the nodes battery life and network capacity. The LoRa protocol includes a number of different layers including encryption at the network, application and device level for secure communications.

**LoRa devices classes**

LoRaWAN has several different classes of end-point devices to address the different needs reflected in the wide range of applications:




- ✓ **Bi-directional end-devices (Class A):** End-devices of Class A allow for bi-directional communications whereby each end-device's uplink transmission is followed by two short downlink receive windows. The transmission slot scheduled by the end-device is based on its own communication needs with a small variation based on a random time basis (ALOHA-type of protocol). This Class A operation is the lowest power end-device system for applications that only require downlink communication from the server shortly after the end-device has sent an uplink transmission. Downlink communications from the server at any other time will have to wait until the next scheduled uplink.
- ✓ **Bi-directional end-devices with scheduled receive slots (Class B):** In addition to the Class A random receive windows, Class B devices open extra receive windows at scheduled times. In order for the End-device to open its receive window at the scheduled time it receives a time synchronized Beacon from the gateway. This allows the server to know when the end-device is listening.

- ✓ **Bi-directional end-devices with maximal receive slots (Class C):** End-devices of Class C have nearly continuously open receive windows, only closed when transmitting. Class C

Following table summarizes difference between class A, class B and class C types used in LoRa.

| LoRa Class A | LoRa Class B | LoRa Class C |
|---|---|---|
| Battery Powered | Low Latency | No Latency |
| Bidirectional communications | Bidirectional with scheduled receive slots | Bidirectional communications |
| Unicast messages | Unicast and Multicast messages | Unicast and Multicast messages |
| Small payloads, long intervals | Small payloads, long intervals, Periodic beacon from gateway | Small payloads |
| End-device initiates communication (uplink) | Extra receive window (ping slot) | Server can initiate transmission at any time |
| Server communicates with end-device (downlink) during predetermined response windows | Server can initiate transmission at fixed intervals | End-device is constantly receiving |

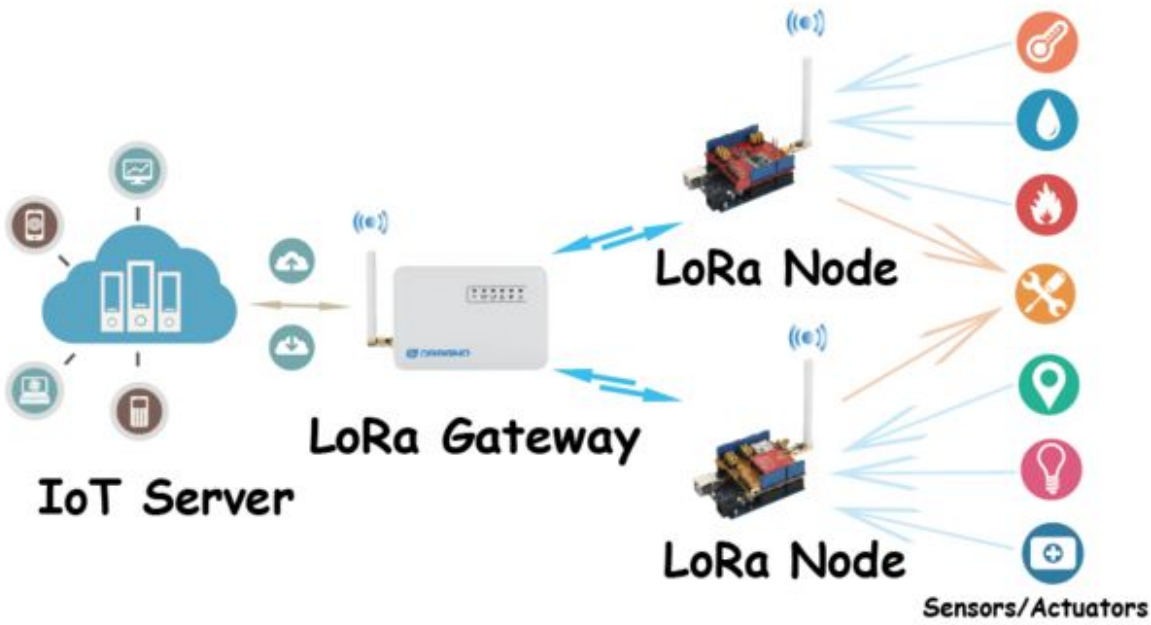
**LoRa features**

The following table showcases some of the key features of the LoRa protocol such as range, modulation and capacity.

| Specification | LoRa Feature |
|---|---|
| Range | 2-5Km Urban (1.24-3.1 mi),<br>15Km suburban (9.3 mi) |
| Frequency | ISM 868/915 MHz |
| Standard | IEEE 802.15.4g |
| Modulation | Spread spectrum modulation type based on FM pulses which vary. |
| Capacity | One LoRa gateway takes thousands of nodes |
| Battery | Long battery life |
| LoRa Physical layer | Frequency, power, modulation and |

| | |
|---|---|
| | inimize between nodes and gateways |

**t. How it works?**




**9. Usage**

This example will guide you to connect two LoRa boards.

**a. Prepare hardware and software**

**Hardware:**

+ 2 x Heltec WiFi LoRa 32 board

+ 2 x micro USB cable

**Software:**

+ Arduino IDE

+ Arduino core for LoRa board (https://github.com/Alictronix/LoRa-ESP32-OLED)

**And connect two board with PC**

**u. Create LoRa Receiver**

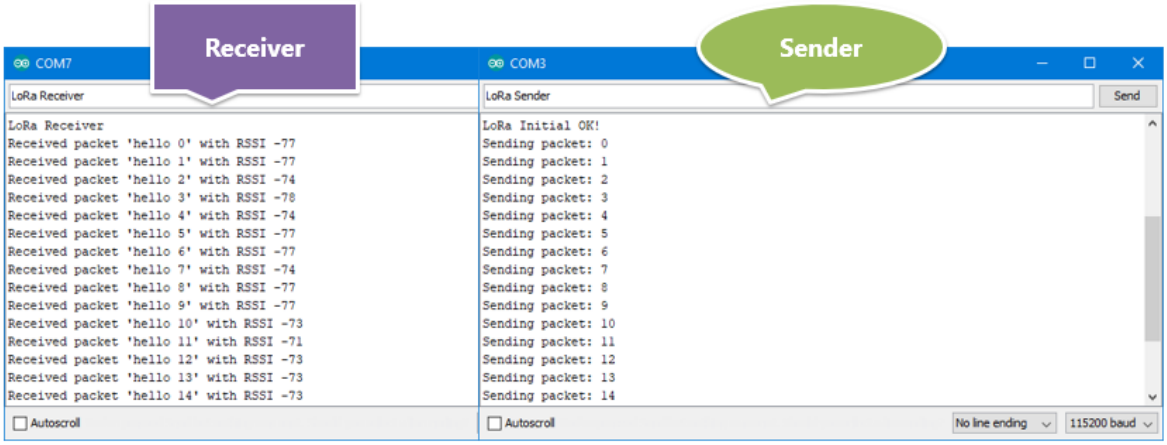
- Checkout materials and samples code from gitlab
- Navigate to **Samples code** folder and open **LoRaReceiver** sketch
- Compile and Upload sketch to LoRa board.

### v. Create LoRa Sender

- ❖ Navigate to **Samples code** folder and open **LoRaSender** sketch
- ❖ Compile and Upload sketch to LoRa board.

### w. Check status and logs

- ❖ Open two **Serial Monitors** you can see the status and logs



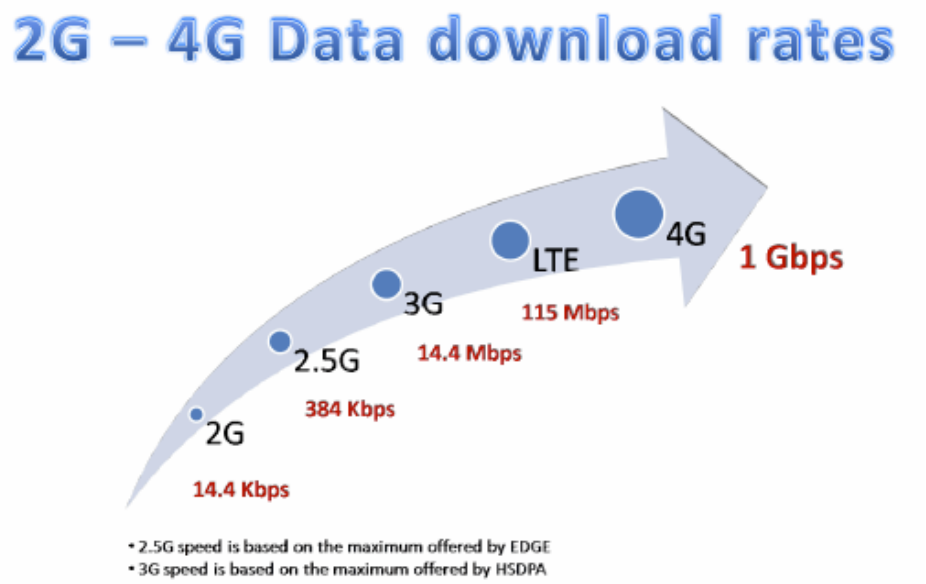

## 10. Exercises

### a. Setup LoRa for Heltec WiFi LoRa 32 board.

### x. Review the samples code and modify code to increase the communication range of two Heltec WiFi LoRa 32 board.

## VII. GPRS/3G/4G

### 1. What is GPRS/3G/4G?




#### a. Overview

❖ **GPRS**

GPRS (General Packet Radio Service) is a packet based communication service for mobile devices that allows data to be sent and received across a mobile telephone network. GPRS is a step towards 3G and is often referred to as 2.5G. Here are some key benefits of GPRS:

**Speed**

GPRS is packet switched. Higher connection speeds are attainable at around 56–118 kbps, a vast improvement on circuit switched networks of 9.6 kbps. By combining standard GSM time slots theoretical speeds of 171.2 kbps are attainable. However in the very short term, speeds of 20-50 kbps are more realistic.

**Always on connectivity**

GPRS is an always-on service. There is no need to dial up like you have to on a home PC for instance. This feature is not unique to GPRS but is an important standard that will no doubt be a key feature for migration to 3G. It makes services instantaneously available to a device.

**New and Better applications**

Due to its high-speed connection and always-on connectivity GPRS enables full Internet applications and services such as video conferencing straight to your desktop or mobile device. Users are able to explore the Internet or their own corporate networks more efficiently than they could when using GSM. There is often no need to redevelop existing applications.

**GSM operator Costs**

GSM network providers do not have to start from scratch to deploy GPRS. GPRS is an upgrade to the existing network that sits along side the GSM network. This makes it easier to deploy, there is little or no downtime of the existing GSM network whilst implementation takes place, most updates are software so they can be administered remotely and it allows GSM providers to add value to their business at relatively small costs.


❖ **3G**

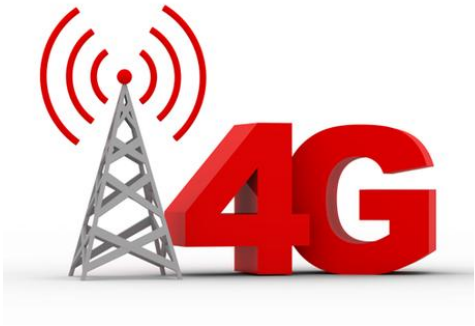
Third-generation, or 3G, technology is a wireless network technology that is commonly utilized in smart phones like iPhones and Blackberries. While its predecessor, second-generation (2G) technology, was formulated around voice applications (like talking, call-waiting and voicemail), 3G technology puts a strong emphasis on Internet and multimedia services, such as web browsing, video conferencing and downloading music. And here are some key benefits of 3G:

**High Bandwidth**

High bandwith—the measure of transmission capacity—is one of the selling points of 3G. This allows you quick and easy access to all of your favorite online multimedia and Internet tools, just like you were at home on a computer. You can pay bills, book dinner reservations, update social networking pages and check emails, all on-the-go.

**Always-Online Devices**

Another advantage of 3G technology is that it can utilize packet-based Internet protocol connectivity. This means your mobile device will always be online and ready for Internet access. However, you will not actually pay for the connection until you start sending or receiving data packets, such as sending an email or looking at a webpage. Some 3G devices are also designed to automatically pick up the closest, free-to-access Wi-Fi signals, in which case, you won't have to pay anything for Internet.




❖ **4G**

4G is the fourth generation of mobile phone technology that follows on from the existing 3G and 2G mobile technology.

2G technology launched in the 1990s and was capable of making digital phone calls and sending texts. Then 3G came along in 2003 and made it possible to browse web pages, make video calls and download music and video on the move.

4G technology builds upon what 3G currently offers, but does everything at a much faster speed.

Here are some key benefits of 4G:

**Improved download/upload speeds**

Standard 4G (or 4G LTE) is around five to seven times faster than 3G, offering theoretical speeds of up to 150Mbps. That equates to maximum potential speeds of around 80Mbps in the real world. For example, you can download a 2GB HD film in 3 minutes 20 seconds on a standard 4G mobile network, while it would take over 25 minutes on a standard 3G network.

**Reduced latency**

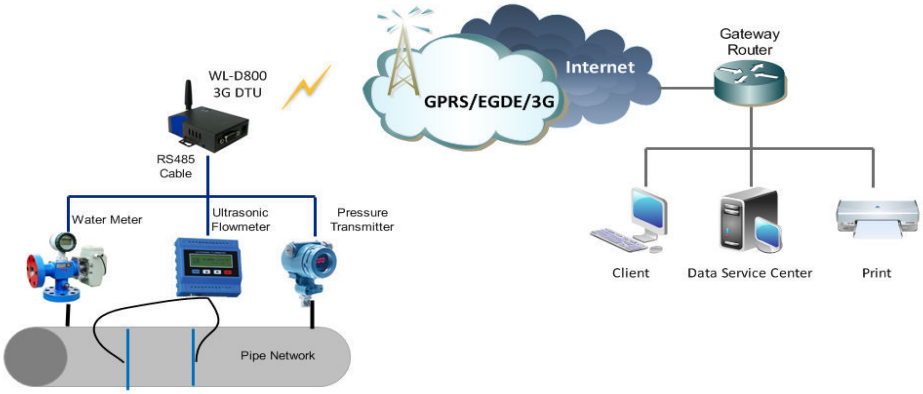
Download speeds aren't the only thing that has been improved, because 4G also has a better response time than 3G – due to lower "latency". This means that a device connected to a 4G mobile network will get a quicker response to a request than the same device connected to a 3G mobile network.

The improved latency times, reduced from 120 milliseconds (3G) to 60 milliseconds (4G), may not seem that significant on paper. However, they can make a significant difference when playing online games and streaming live video. Find out more about the benefits of 4G for gaming.

**Crystal clear voice calls**

Voice over LTE (VoLTE) is similar to Voice over Internet Protocol (VoIP), which use voice apps such as Skype to support voice calls over the internet. Effectively, VoLTE rides on the back of the 4G network and brings crystal clear voice calls and video chat to your 4G mobile phone. Voice call billing could even be eliminated as part of it.

**b. How it works?**



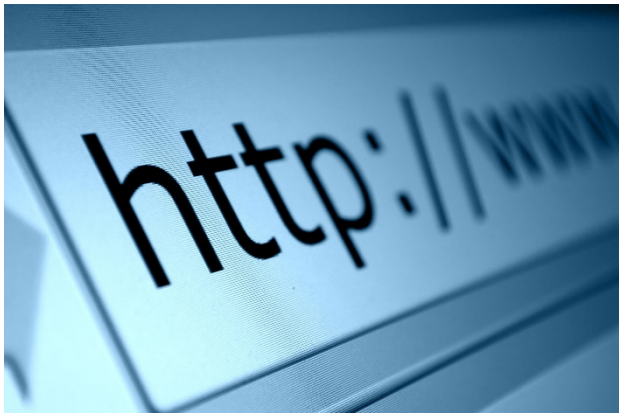

**11.**

## VIII. HTTP/WebSocket

### 1. What is HTTP?

### a. Overview


The Hypertext Transfer Protocol (HTTP) is the foundation protocol of the World Wide Web (WWW). The name is somewhat misleading in that HTTP is not a protocol for transferring hypertext; rather, it's a protocol for transmitting information with the efficiency necessary for making hypertext jumps. The data transferred by the protocol can be plain text, hypertext, audio, images, or any type of Internet-accessible information.




HTTP is a transaction-oriented client/server protocol. The most typical use of HTTP is between a web browser and a web server. To provide reliability, HTTP makes use of TCP. Nevertheless, HTTP is a "stateless" protocol; each transaction is treated independently. A typical implementation creates a new TCP connection between client and server for each transaction and then terminates the connection as soon as the transaction completes, although the specification doesn't dictate this one-to-one relationship between transaction and connection lifetimes.

**HTTP messages**

HTTP has two messages:

**Request Messages**

A request message is sent by an agent to a server to request some action. These are the possible actions, called methods:

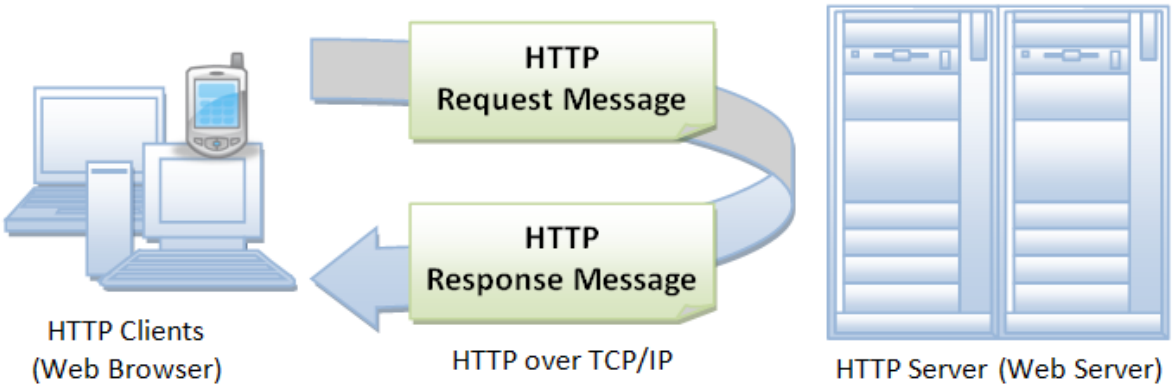
| Method | Description |
|---|---|
| OPTIONS | A request for information about the options available. |
| GET | A request to retrieve information. |
| HEAD | Like a GET except that the server's response must not include an entity body; all of the header fields in the response are the same as if the entity body were present. This enables a client to get information about a resource without transferring the entity body. |
| POST | A request to accept the attached entity as a new subordinate to the identified URL. |
| PUT | A request to accept the attached entity and store it under the supplied URL. This may be a new resource with a new URL, or a replacement of the contents of an existing resource with an existing URL. |
| DELETE | Requests that the origin server delete a resource. |
| TRACE | Requests that the server return whatever is received as the entity body of the response. This can be used for testing and diagnostic purposes. |

**Response Messages**

A response message is returned by a server to an agent in response to a request message. It may include an entity body containing hypertext-based information. In addition, the response message must specify a status code, which indicates the action taken on the corresponding request. Status codes are organized into the following categories:

| Category | Description |
|---|---|
| Informational | The request has been received and processing continues. No entity body accompanies this response. |
| Successful | The request was successfully received, understood, and accepted. |
| Redirection | Further action is required to complete the request. |
| Client Error | The request contains a syntax error or the request cannot be fulfilled. |
| Server Error | The server failed to fulfill an apparently valid request. |

**c. How it works?**




**12. What is WebSocket?**

**a. Overview**

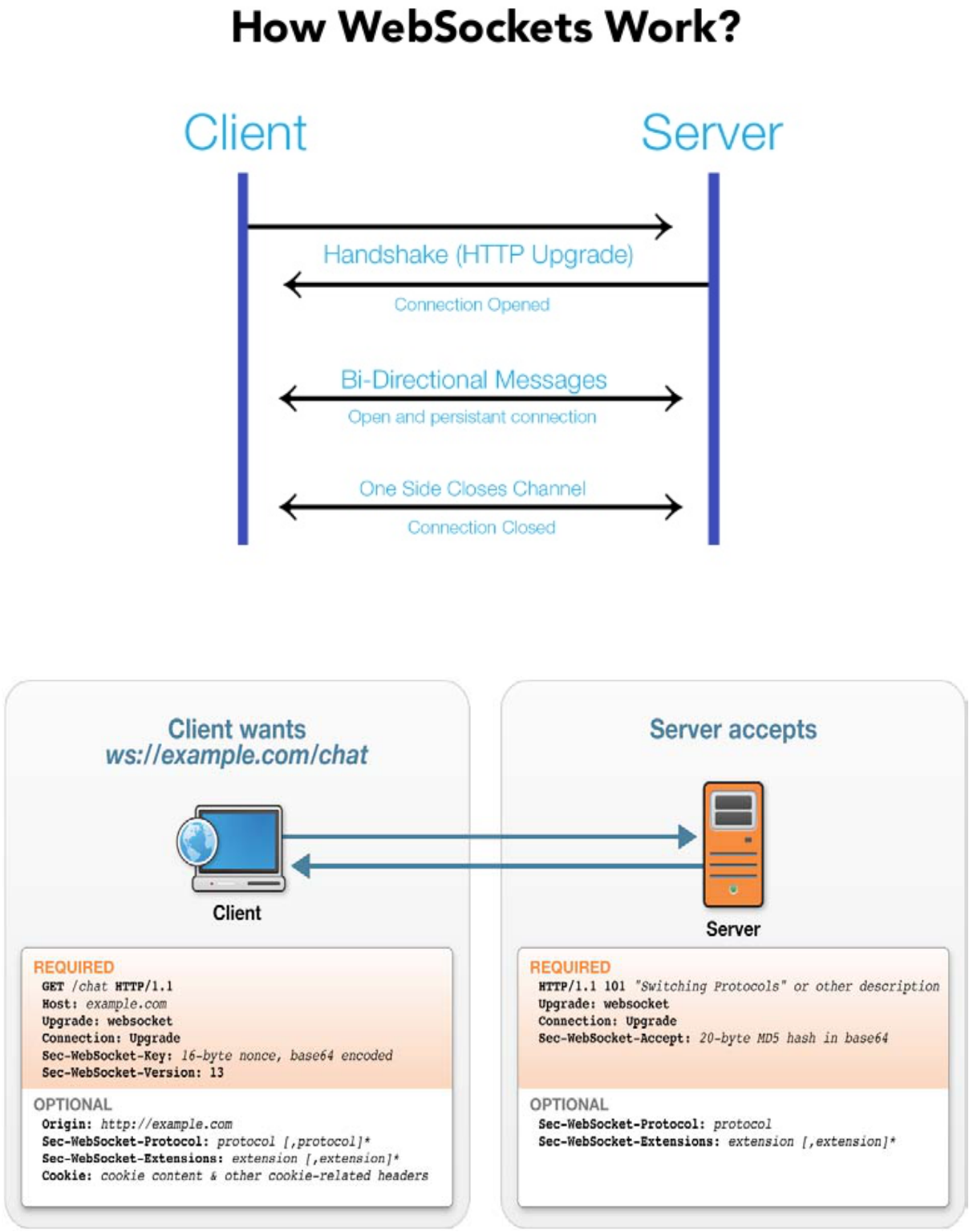
WebSocket is a protocol which allows for communication between the client and the server/endpoint using a single TCP connection. Sounds a bit like http doesn't it? The advantage WebSocket has over HTTP is that the protocol is full-duplex (allows for simultaneous two-way inimizetion) and it's header is much smaller than that of a HTTP header, allowing for more efficient inimizetion even over small packets of data.

**d. How it works?**

**Step 1:** Client sends the Server a handshake request in the form of a HTTP upgrade header with data about the WebSocket it's attempting to connect to.

**Step 2:** The Server responds to the request with another HTTP header, this is the last time a HTTP header gets used in the WebSocket connection. If the handshake was successful, they server sends a HTTP header telling the client it's switching to the WebSocket protocol.

**Step 3:** Now a constant connection is opened and the client and server can send any number of messages to each other until the connection is closed. These messages only have about 2 bytes of overhead.




**13. Usage**

This example will help you send data to **Thingspeak** cloud web services using Espressif ESP8266 hardwares.

**a. Prepare hardware and software**

**Hardware**:

+ 1 x Expressif ESP8266

+ 1 x USB cable

**Software:**

+ Arduino IDE

+ Arduino core for ESP8266 ()

**e. Create channel on Thingspeak**

- Login with your account

- Create a new channel by selecting

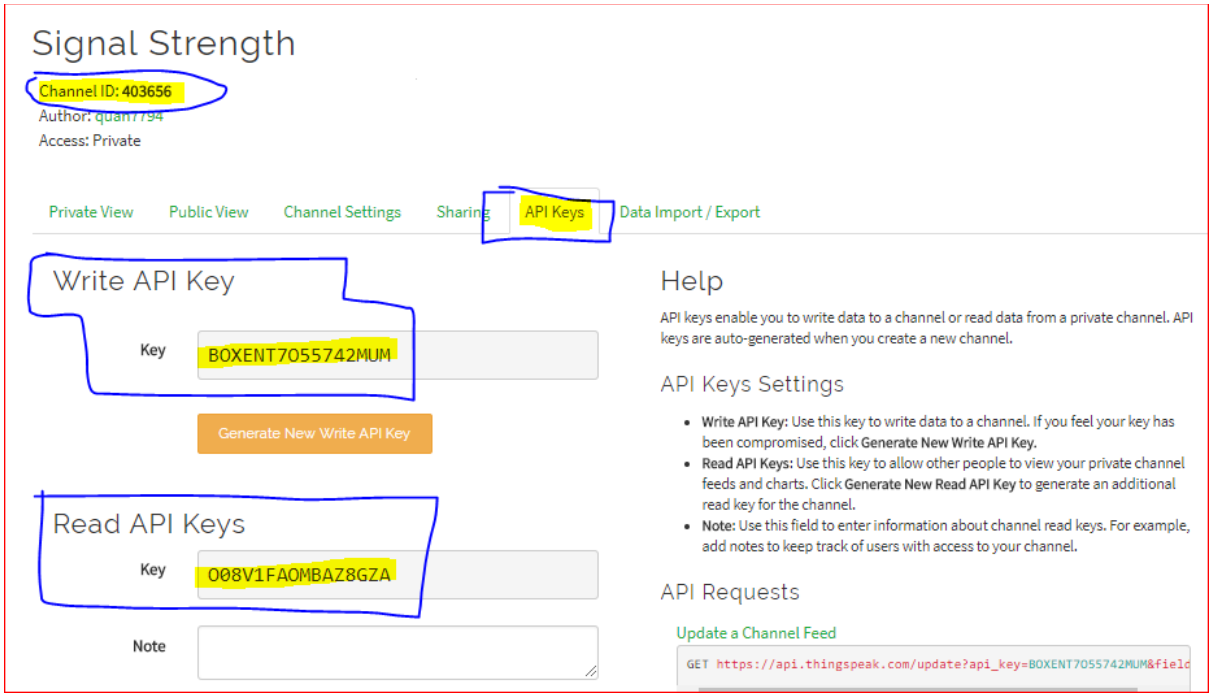
*Channels -> My Channels, -> New Channel.*

- Name the channel, "**ESP8266 Signal Strength**" and name Field 1, "RSSI".

- Click "Save Channel" at the bottom to finish the process.


❖ When your channel created, go to "API Keys" tab and remember the list of information below:

- Channel ID
- Write API key
- Read API Keys




### f. Upload sketch

❖ Checkout materials and samples code from gitlab

https://gitlab.tma.com.vn/tic/training/tree/master/Connectivity_And_Communication_Protocols

❖ Navigate to **Samples code** and open **WiFiSignal** sketch
❖ Change SSID, password of WiFi, channel ID, API key.

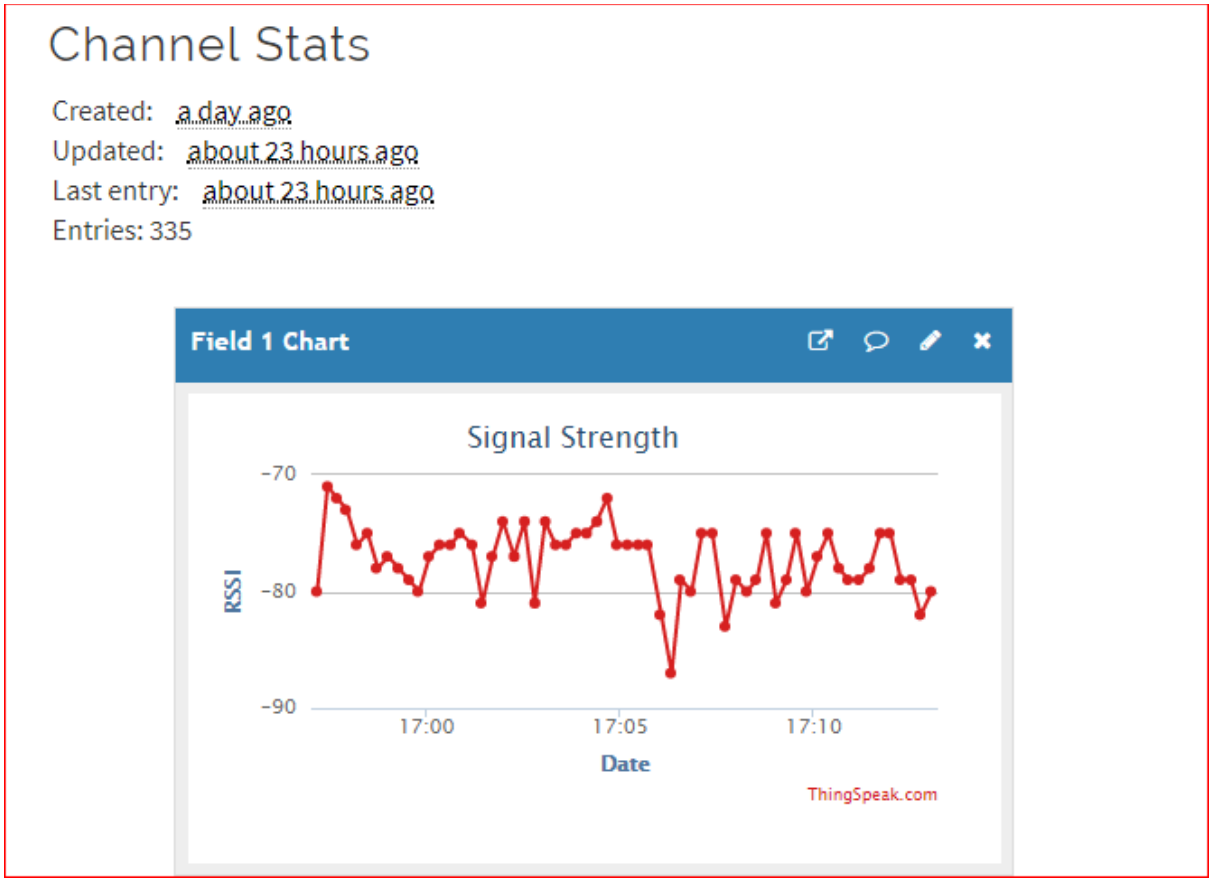
```
// Wi-Fi Settings
const char* ssid = "XXX"; // your wireless network name (SSID)
const char* password = "XXXXXXXX"; // your Wi-Fi network password

// ThingSpeak Settings
const int channelID = XXX; //Your channel ID
String writeAPIKey = "XXXXXXXXXXXXXXXX";
```

- Compile and Upload sketch to ESP8266 board.

### g. Check result

- Go to Thingspeak website you can see the status of channel




**14. Exercises**

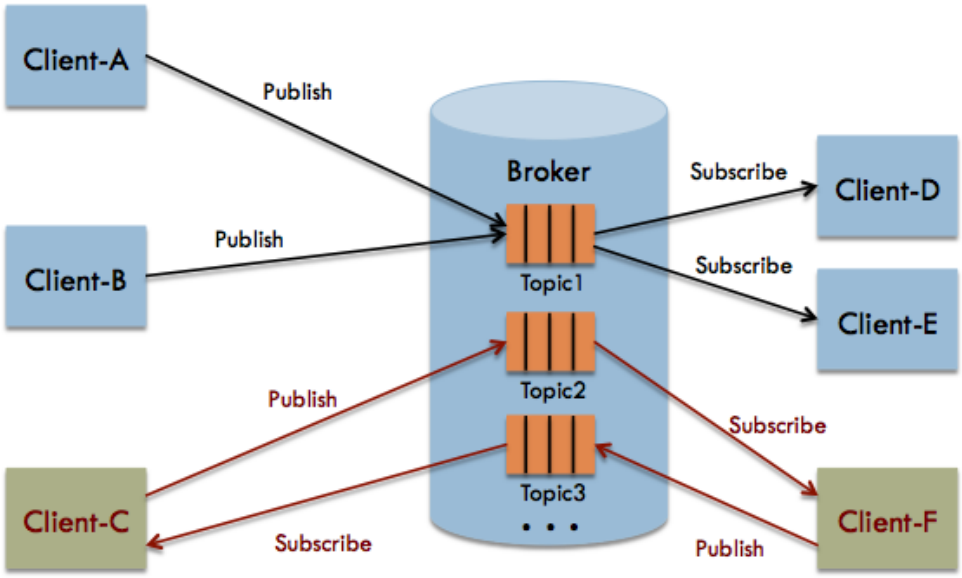
**a. Send data to IFTTT cloud web services using Wifi with Expressif ESP8266**

**IX.**

# X. MQTT

## 1. What is MQTT?

### a. Overview




MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to inimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

#### MQTT features

#### Publish/subscribe model

MQTT’s “pub/sub” model scales well and can be power efficient. Brokers and nodes publish information and others subscribe according to the message content, type, or subject. (These are MQTT standard terms.) Generally, the broker subscribes to all messages and then manages information flow to its nodes. The publish/subscribe model offers several specific benefits noted below.

#### Space decoupling

While the node and the broker need to have each other’s IP address, nodes can publish information and subscribe to other nodes’ published information without any knowledge of each other, since everything goes through the central broker. This

reduces overhead that can accompany TCP sessions and ports, and allows the end nodes to operate independently of one another.

### Time decoupling

A node can publish its information regardless of other nodes' states. Other nodes can then receive the published information from the broker when they are active. This allows nodes to remain in sleepy states even when other nodes are publishing messages directly relevant to them.

### Synchronization decoupling

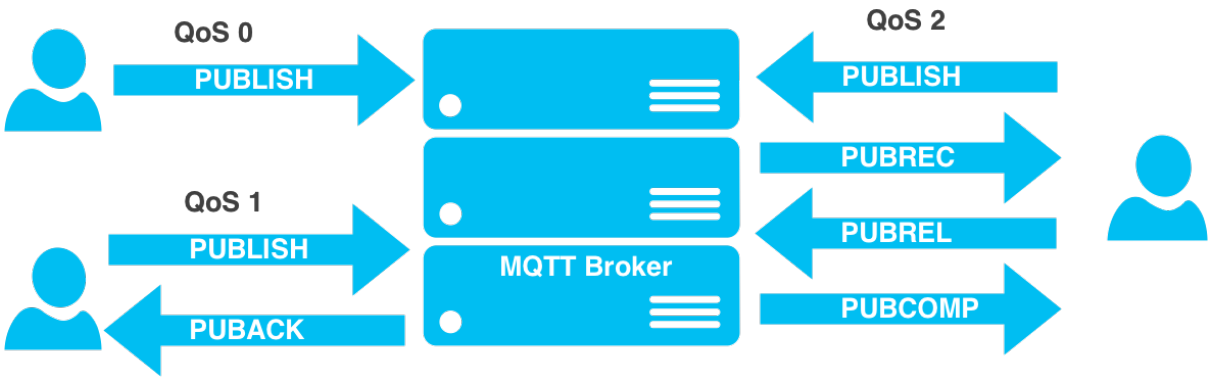
A node that's in the midst of one operation is not interrupted to receive a published message to which it's subscribed. The message is queued by the broker until the receiving node is finished with its existing operation. This saves operating current and reduces repeated operations by avoiding interruptions of ongoing operations or sleepy states.

### Security

MQTT uses unencrypted TCP and is not "out-of-the-box" secure. However, because it uses TCP, it can—and should—use TLS/SSL Internet security. TLS is a very secure method for encrypting traffic, but is also resource-intensive for lightweight clients due to its required handshake and increased packet overhead. For networks where energy is a very high priority and security much less so, encrypting just the packet payload may suffice.

### MQTT Quality of Service (QoS) levels

The term "QoS" means other things outside of MQTT. In MQTT, "QoS" levels 0, 1, and 2 describe increasing levels of guaranteed message delivery.



- ✓ **MQTT QoS Level 0 (at most once):** Commonly known as “Fire and forget,” this is a single transmit burst with no guarantee of message arrival. It might be used for highly repetitive message types or non-mission critical messages.
- ✓ **MQTT QoS Level 1 (at least once):** This level attempts to guarantee a message is received at least once by the intended recipient. Once a published messaged is received and understood by the intended recipient, it acknowledges the message with an acknowledgement message (PUBACK) addressed to the publishing node. Until the PUBACK is received by the publisher, it stores the message and retransmits it periodically. This type of message may be useful for a non-critical node shutdown.
- ✓ **MQTT QoS Level 2 (exactly once):** This level attempts to guarantee the message is received and decoded by the intended recipient. It’s the most secure and reliable MQTT level of QoS. The publisher sends a message announcing it has a QoS level 2 message. Its intended recipient gathers the announcement, decodes it, and indicates that it’s ready to receive the message. The publisher relays its message. Once the recipient understands the message, it completes the transaction with an acknowledgement. This type of message may be useful for turning on or off lights or alarms in a home.

**Last will and testament**

MQTT provides a “last will and testament (LWT)” message that can be stored in the MQTT broker in case a node unexpectedly disconnects from the network. This LWT retains the node’s state and purpose, including the types of commands it published and its subscriptions. If the node disappears, the broker notifies all subscribers of the node’s LWT. And if the node returns, the broker notifies it of its prior state. This feature nicely accommodates lossy networks and scalability.

**Flexible topic subscriptions**

An MQTT node may subscribe to all messages within a given functionality. For example a “kitchen oven node” may subscribe to all messages for

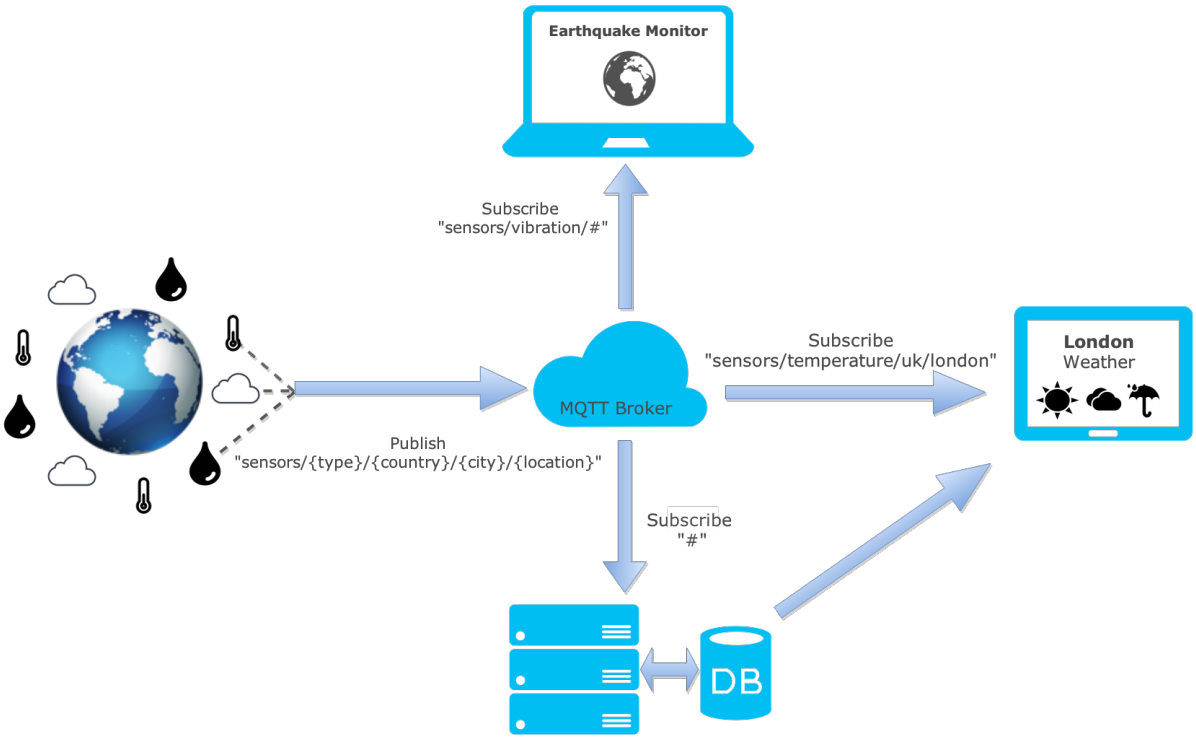
kitchen/oven/+

with the “+” as a wildcard. This allows a minimal amount of code (i.e., memory and cost). Another example is if a node in the kitchen is interested in all temperature information regardless of the end node’s functionality. In this case,

kitchen/+/temp

will collect any message in the kitchen from any node reporting “temp.” There are other equally useful MQTT wildcards for reducing code footprint and, therefore, memory size and cost.

**y. How it works?**




## 15. Usage

### a. Prepare hardware and software

**Hardware**:

+ 2 x Expressif ESP8266

+ 2 x USB cable

**Software:**

+ Arduino IDE

+ Arduino core for ESP8266 ()

+ PubSubClient library ()

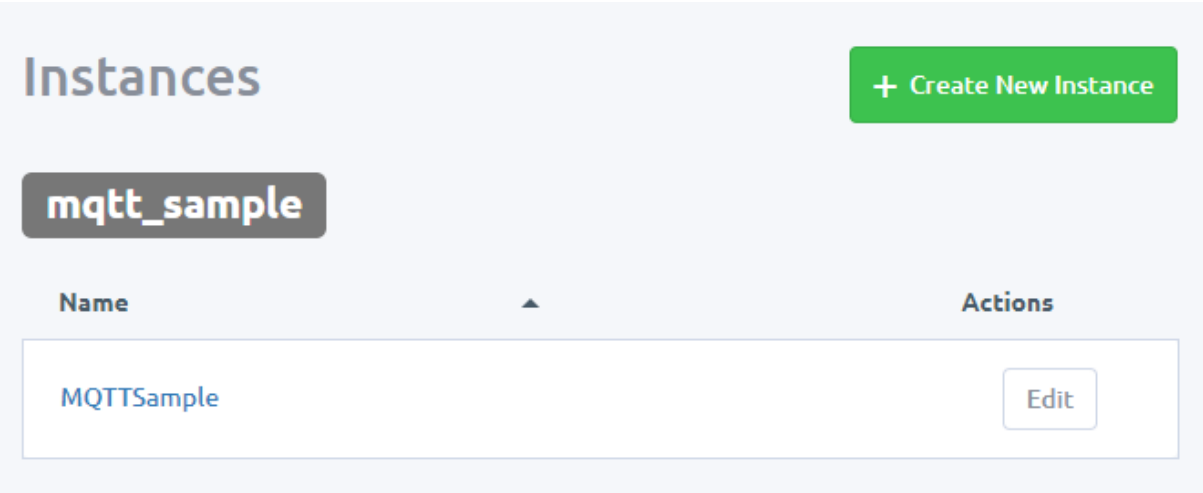
### h. Create account and instance on CloudMQTT

**Create account:**

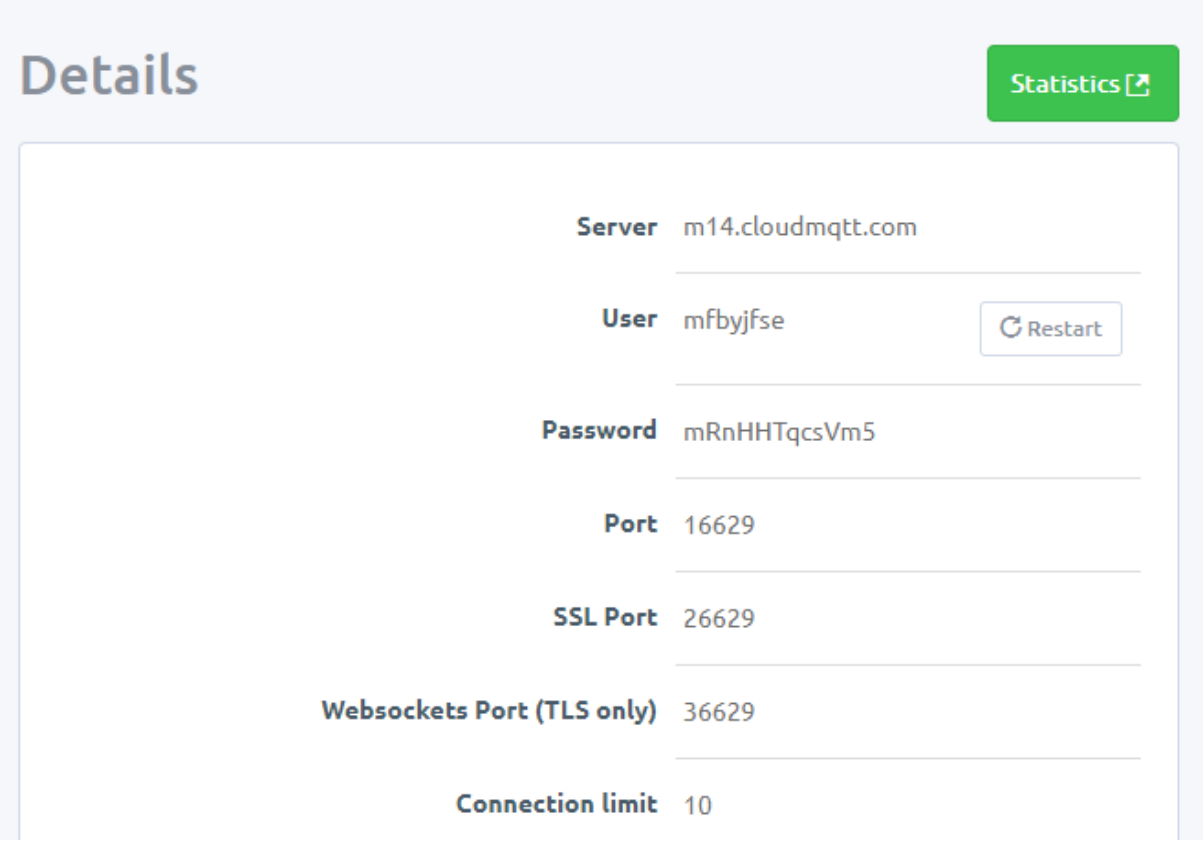
- You can easy to register account on CloudMQTT website (https://www.cloudmqtt.com/)

**Create instance:**

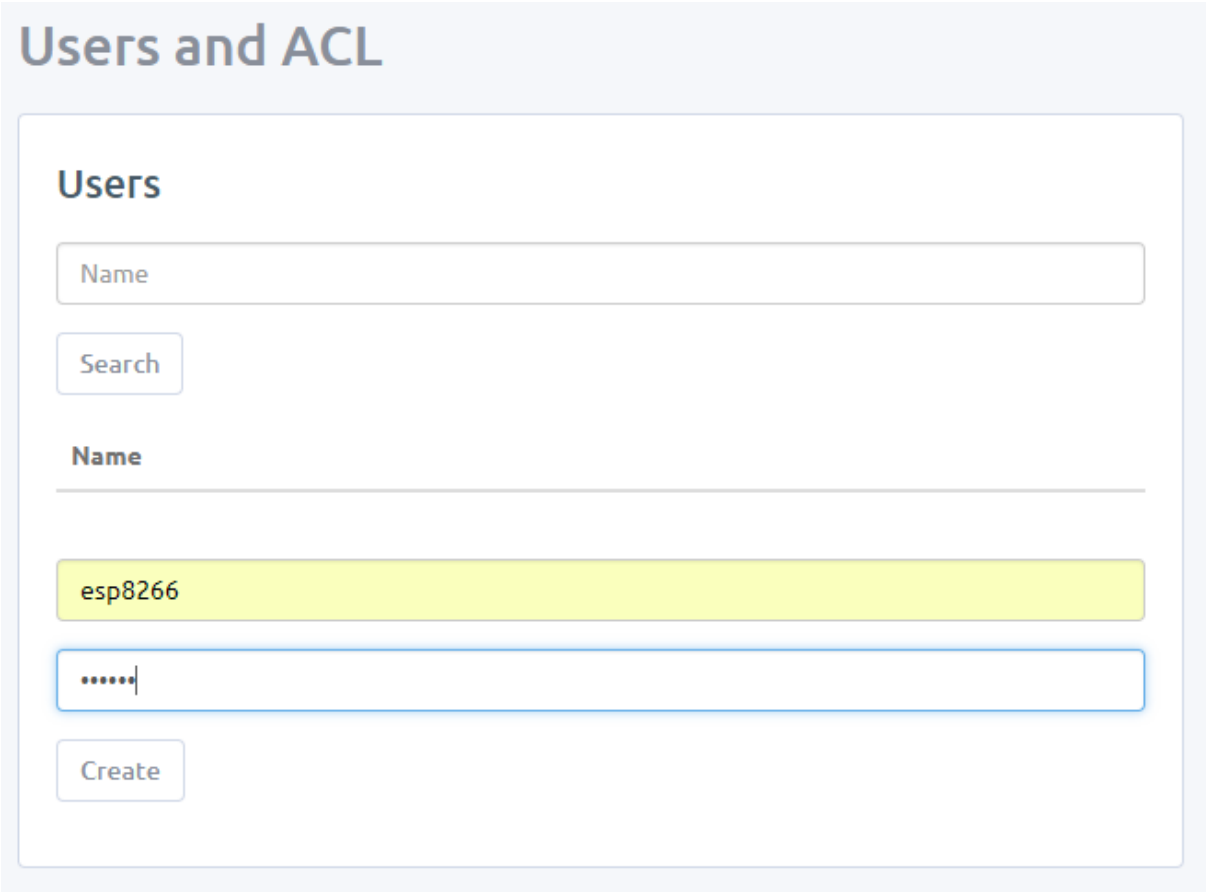
- Login with account which already register in previous step
- Click **Create New Instance**



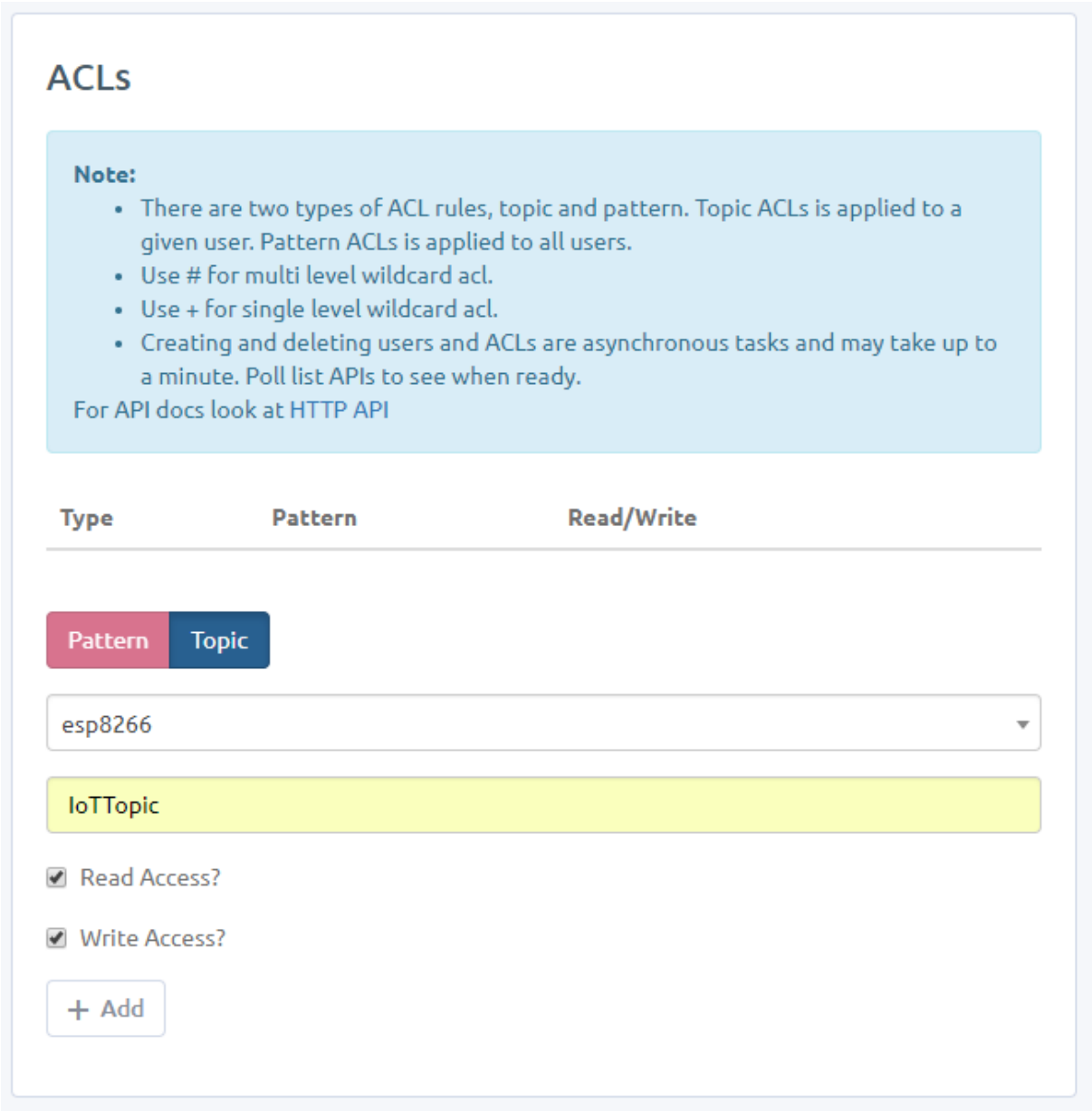

- When your instance is created, press on details for your instance to find your **server**, **port**, **username** and **password** for your cloud. Please remember this information it must be need for next step.




- After that you want to create new user for your topic.
  Click **User** tab to switch to **User and ACL** screen.




- Create your topic




### i. Create Publisher

- Checkout materials and samples code from gitlab
- Navigate to **Samples code** folder and open **MQTTPublisher** sketch
- Change SSID, password of your WiFi network
- Change Server, port, topic username and password

```
#define ssid "ssid"
#define password "password"

#define mqtt_server "ml4.cloudmqtt.com"
#define mqtt_port   16629
#define mqtt_topic_pub "IoTTopic"
#define mqtt_topic_sub "IoTTopic"
#define mqtt_user "esp8266"
#define mqtt_pwd "123456"
```

- Compile and Upload to your device

### j. Create Subscriber

- Navigate to **Samples code** folder and open **MQTTSubscriber** sketch
- Change SSID, password of your WiFi network
- Change Server, port, topic username and password

```
#define ssid "ssid"
#define password "password"

#define mqtt_server "ml4.cloudmqtt.com"
#define mqtt_port   16629
#define mqtt_topic_pub "IoTTopic"
#define mqtt_topic_sub "IoTTopic"
#define mqtt_user "mfbyjfse"
#define mqtt_pwd "mRnHHTqcsVm5"
```

- Compile and Upload to your device

### k. Check results

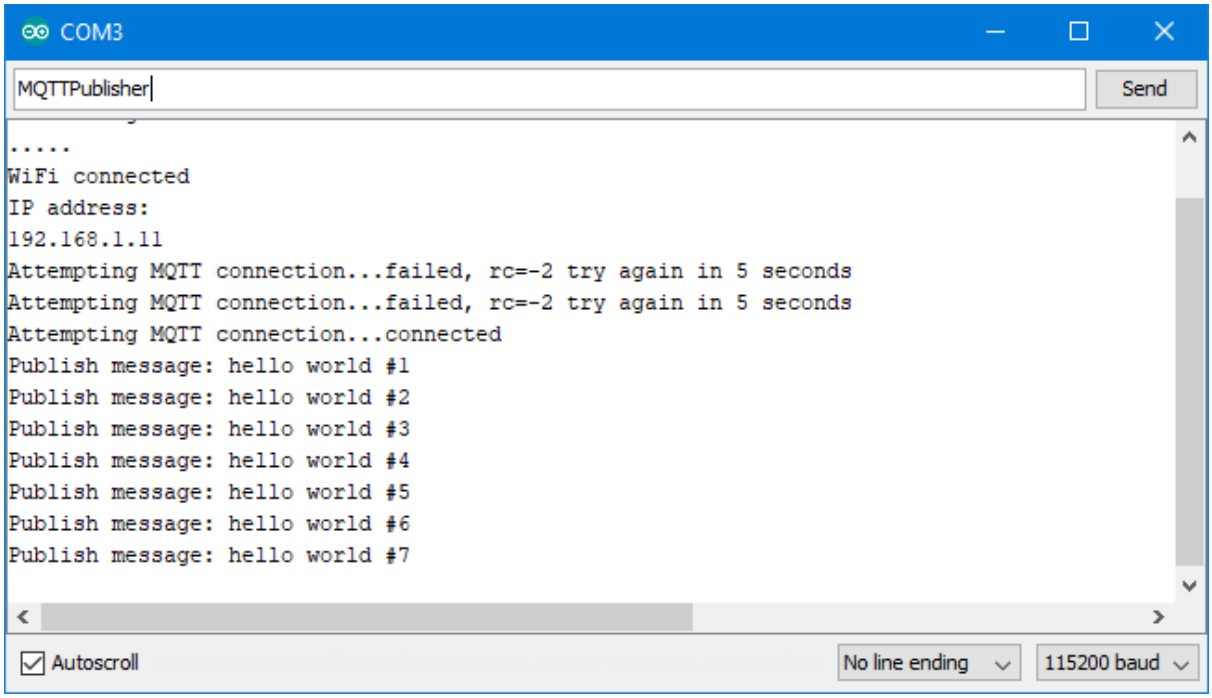



*Figure 11: MQTT Publisher*

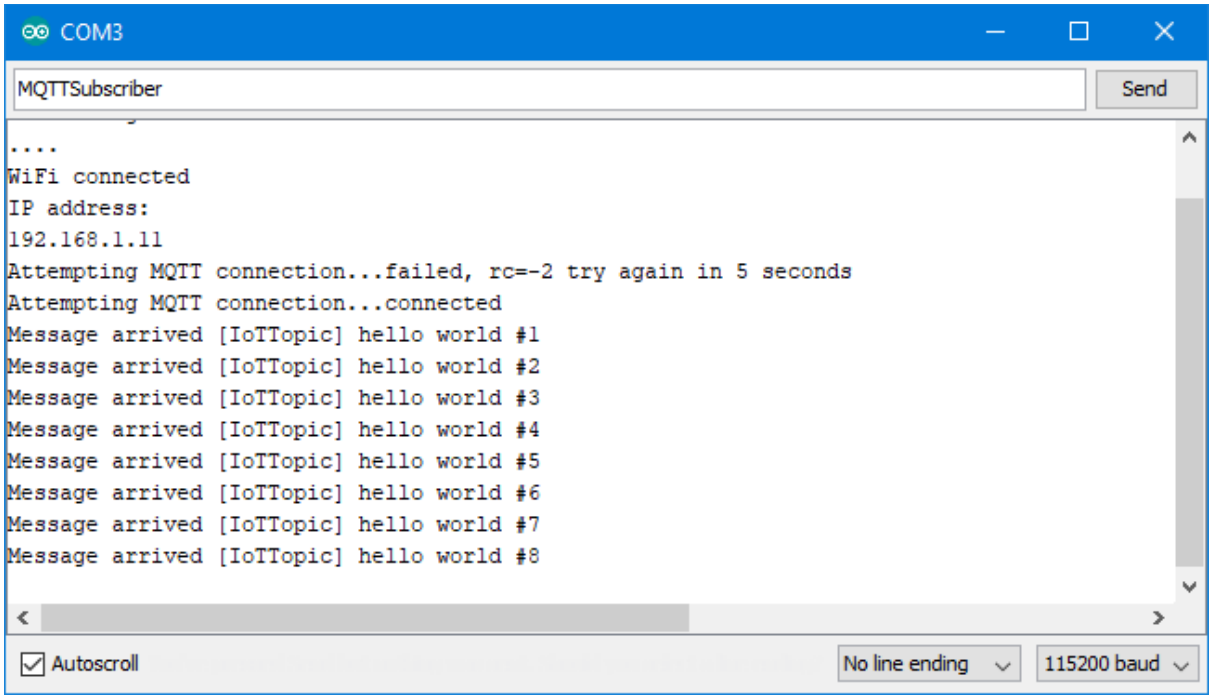



*Figure 12: MQTT Subscriber*

❖ **Publish/Subscribe data using MQTT with Expressif ESP32 hardware.**

**a. Prepare hardware and software**

**Hardware**:

+ 1 x Expressif ESP32

+ 1 x USB cable

+ 1 x DHT11

**Software:**

+ Arduino IDE

+ Arduino core for ESP32 ()

+ PubSubClient library ()

+ DHT libraby (https://github.com/adafruit/DHT-sensor-library)

**b. Create account on Blocky**

- You can easy to register account on Blocky website ()
- Get your Authentication Key into Profile.

**c. Collect data sensor and publish data**

- Checkout materials and samples code from gitlab https://gitlab.tma.com.vn/tic/training/tree/connection/Connectivity_And_Communication_Protocols/Samples Code/MQTT-Blocky

- Connect DHT11 with ESP32.

```
#define DHTPIN 27     // what digital pin we're connected to
```

- Change SSID, password of your WiFi network
- Change mqttPassword is your Authentication Key.

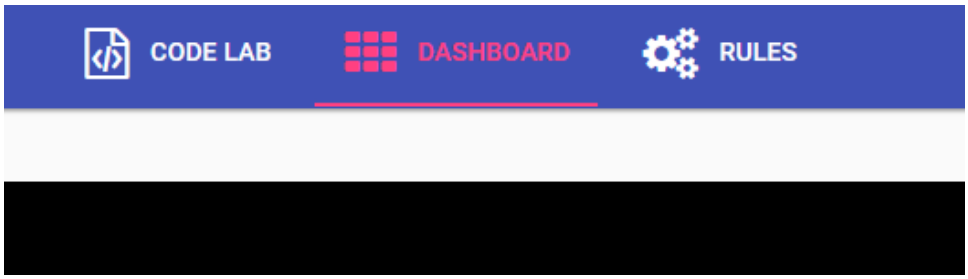
```
const char* ssid     = "TIC";
const char* password = "TlcVi3tn@m";

const char* mqttServer = "broker.getblocky.com";
const int mqttPort = 1883;
const char* mqttUser = "chipId";
const char* mqttPassword = "[redacted]";
```

- Compile and Upload to your device

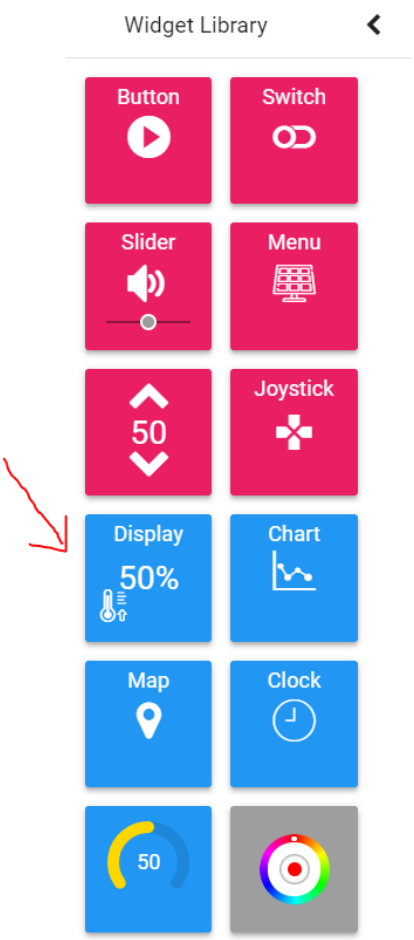
**d. Create Subscriber and Dashboard on getBlocky.com**

- Choose tab DASHBOAD on website

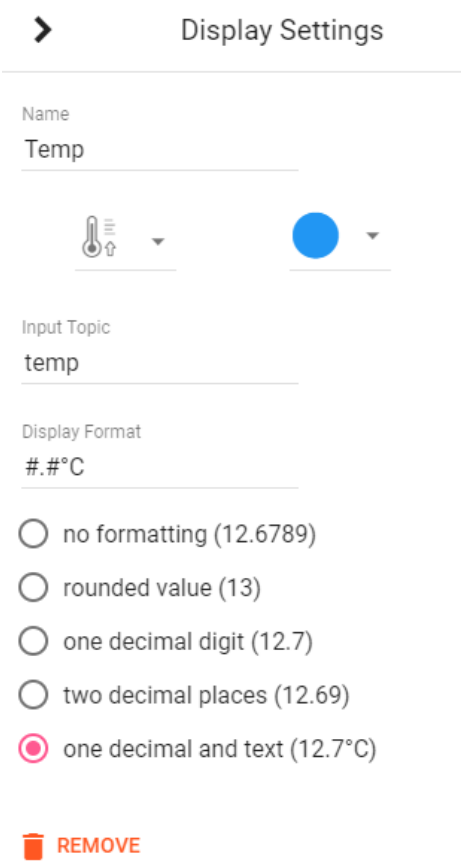


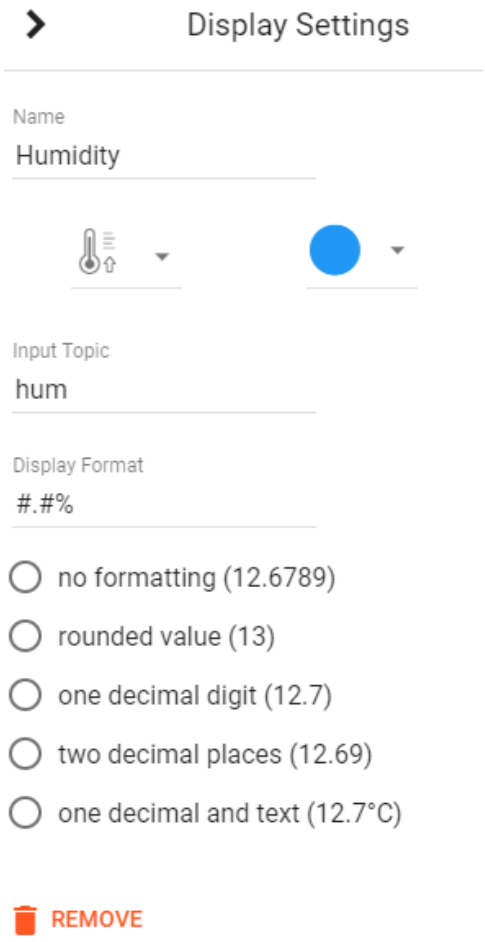

- Create new widget


- Choose widget type



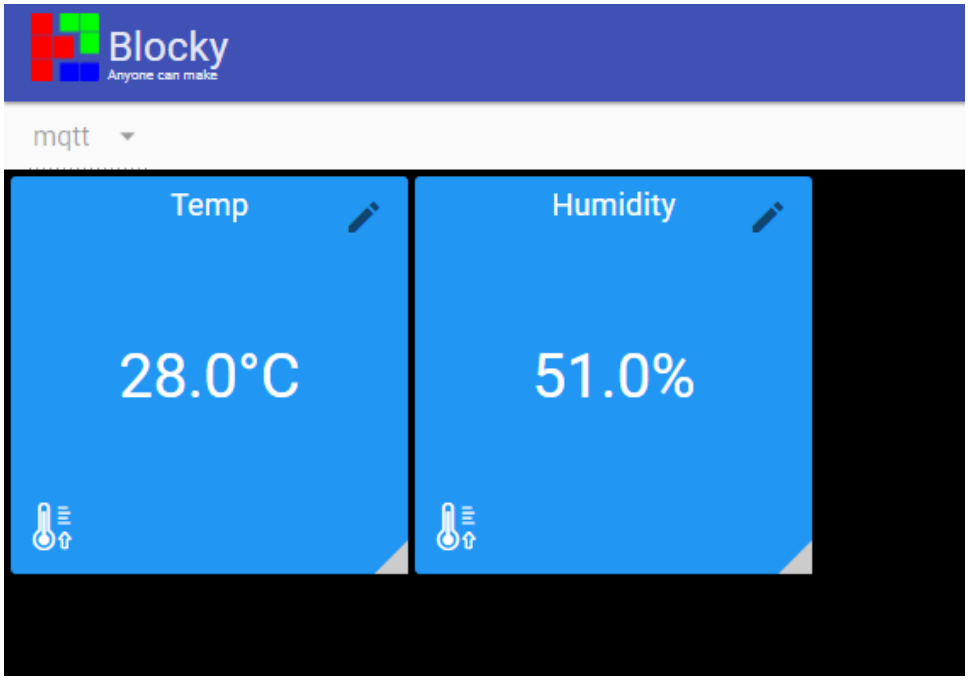

- Subcribler topic "temp" and "hum" in Input Topic.




- Run Dashboard


**e. Check results**




**16. Exercises**

**a. Publish/Subscribe data using MQTT with Expressif ESP8266 hardware.**

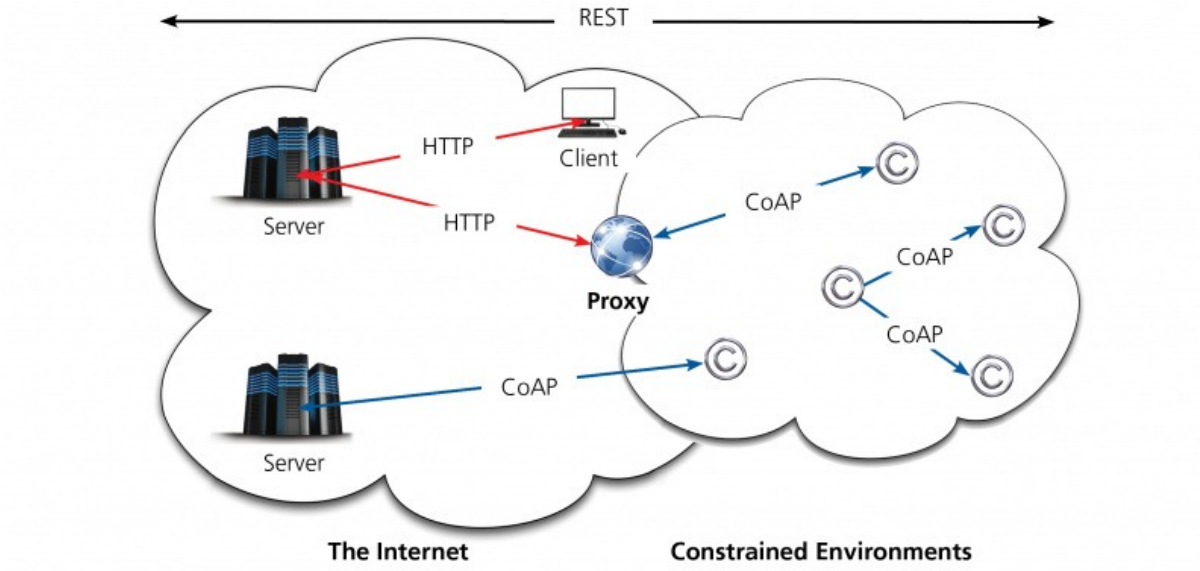
## XI. CoAP

### 1. What is CoAP?

### a. Overview

# CoAP

"The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation."




With the growing importance of the IoT, the IETF took on lightweight messaging and defined the CoAP. As defined by the IETF, CoAP is for "use with constrained nodes and constrained (e.g., low-power, lossy) networks." The Eclipse community also supports CoAP as an open standard, and like MQTT, CoAP is commercially supported and growing rapidly among IoT providers.

CoAP is a client/server protocol and provides a one-to-one "request/report" interaction model with accommodations for multi-cast, although multi-cast is still in the early stages of IETF standardization. Unlike MQTT, which has been adapted to IoT needs from a decades-old protocol, the IETF specified CoAP from the outset to support IoT with lightweight messaging for constrained devices operating in a constrained environment. CoAP is designed to interoperate with HTTP and the RESTful Web through simple proxies, making it natively compatible to the Internet.

### Strengths of CoAP

#### Native UDP

CoAP runs over UDP, which is inherently and intentionally less reliable than TCP, depending on repetitive messaging for reliability instead of consistent connections. For example, a temperature sensor may send an update every few seconds, even though nothing has changed from one transmission to the next. If a receiving node misses one update, the next will arrive in a few seconds and will likely be not much different than the first.

UDP's connectionless datagrams also enable faster wake-up and transmit cycles as well as smaller packets with less overhead. This allows devices to remain in a sleepy state for longer periods of time, conserving battery power.

#### Multi-cast support

A CoAP network is inherently one-to-one; however, it supports one-to-many or many-to-many multi-cast requirements. This is inherent in CoAP because it's built on top of Ipv6, which enables multicast addressing for devices in addition to their normal Ipv6 addresses. Note that multicast message delivery to sleeping devices is unreliable or can impact the device's battery life if it must wake regularly to receive these messages.

#### Security

CoAP uses DTLS on top of its UDP transport protocol. Like TCP, UDP is unencrypted, but can be—and should be—augmented with DTLS.

#### Resource/service discovery

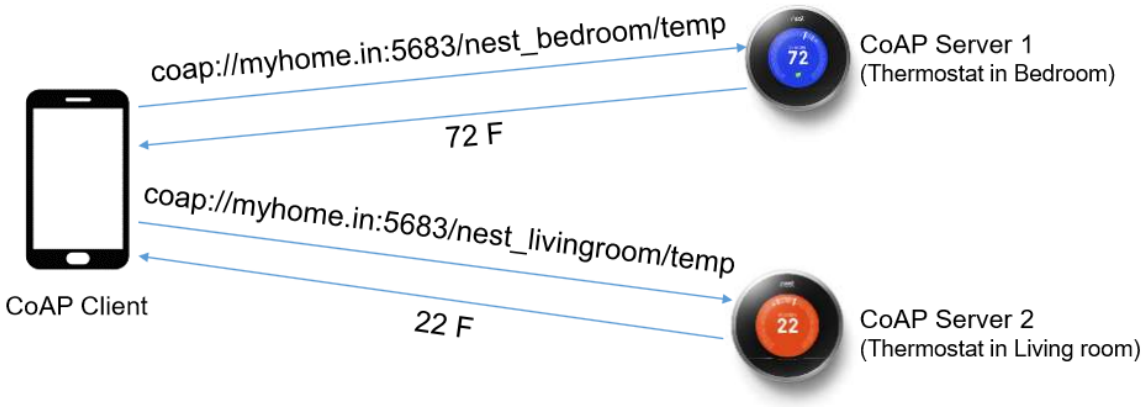
CoAP uses URI to provide a standard presentation and interaction expectations for network nodes. This allows a degree of autonomy in the message packets, since the target node's capabilities are partly understood by its URI details. In other words, a battery-powered sensor node may have one type of URI while a line-powered flow-control actuator may have another. Nodes communicating to the battery-powered sensor node might be programmed to expect longer response times, more repetitive information, and limited message types. Nodes communicating to the line-powered flow-control actuator might be programmed to expect rich, detailed messages at a very rapid pace.

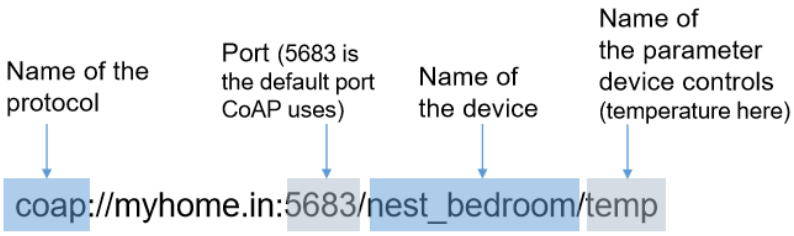
#### Asynchronous communication

Within the CoAP protocol, most messages are sent and received using the request/report model; however, other modes of operation allow nodes to be somewhat

decoupled. For example, CoAP has a simplified “observe” mechanism similar to MQTT’s pub/sub that enables nodes to observe others without actively engaging them.

### z. How it works?




## 17. Usage

### a. Prepare hardware and software

**Hardware:**

+ 2 x Heltec WiFi LoRa 32 board

+ 2 x micro USB cable

**Software:**

+ Arduino IDE

+ Arduino cores for ESP32 board.

**And connect two board with PC**

### aa. Create CoAP Server

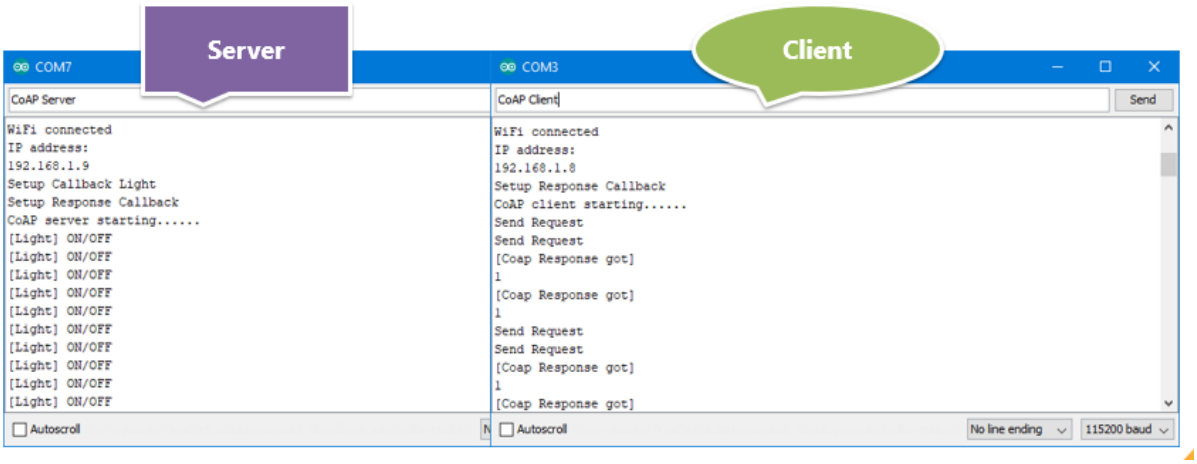
- Checkout materials and samples code from gitlab
- Navigate to **Samples code** folder and open **CoAPServer** sketch
- Change **SSID** and **password** of WiFi network
- Compile and Upload sketch to ESP32 board.

### bb.Create CoAP Client

- Navigate to **Samples code** folder and open **CoAPClient** sketch
- Change **SSID** and **password** of WiFi network
- Change IP address of CoAP server
- Compile and Upload sketch to ESP32 board.

### cc. Check status and logs

- Open two Serial Monitors you can see the status and logs




## 18. Exercises

**a. Setup CoAP for ESP32 board.**

## XII. AMQP

### 1. What is AMQP?

### a. Overview


"The Advanced Message Queuing Protocol (AMQP) is an open standard for passing business messages between applications or organizations. It connects systems, feeds business processes with the information they need and reliably transmits onward the instructions that achieve their goals." –

The Advanced Message Queuing Protocol (AMQP) creates interoperability between clients and brokers (i.e. messaging middleware). Its goal of creation was to enable a wide range of different applications and systems to be able to work together, regardless of their internal designs, standardizing enterprise messaging on industrial scale.

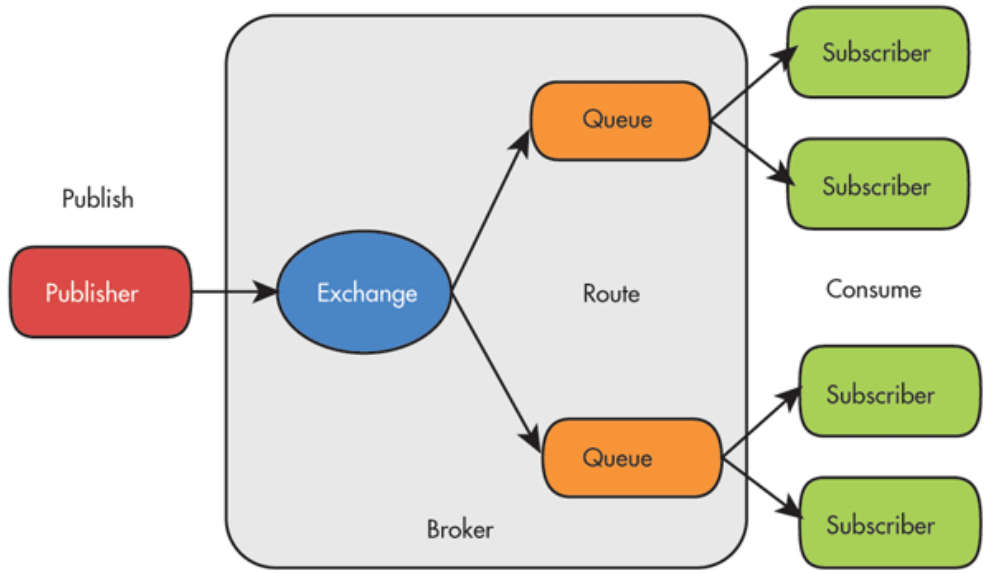
AMQP includes the definitions for both the way networking takes place and the way message broker applications work. This means the specifications for:

+ Operations of routing and storage of messages with the message brokers and set of rules to define how components involved work

+ And a wire protocol to implement how communications between the clients and the brokers performing the above operations work

**AMQP Assembly and Terminology**

Understanding and working with AMQP involves being familiar with quite a few different terms and terminology. In this section, we will go over these key parts:



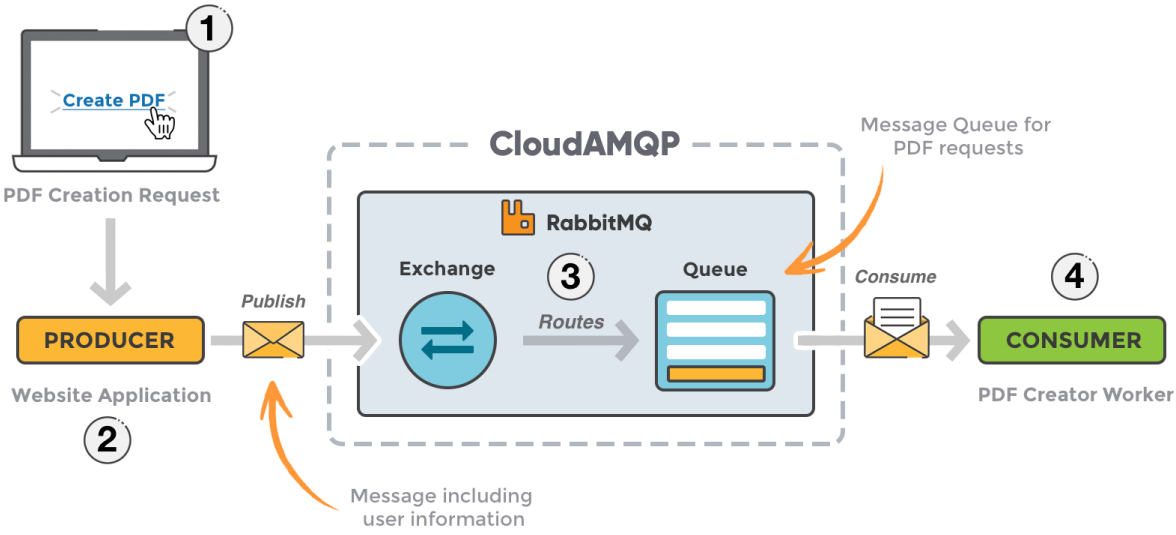

- ✓ Broker (Server): An application – implementing the AMQP model – that accepts connections from clients for message routing, queuing etc.
- ✓ Message: Content of data transferred / routed including information such as payload and message attributes.
- ✓ Consumer: An application which receives message(s) – put by a producer – from queues.
- ✓ Producer: An application which put messages to a queue via an exchange.

**AMQP Components**

The AMQP Model defining how messages are received, routed, stored, queued and how application parts handling these tasks work rely on the clear set definitions of the below components:

- ✓ Exchange: A part of the broker (i.e. server) which receives messages and routes them to queues
- ✓ Queue (message queue): A named entity which messages are associated with and from where consumers receive them
- ✓ Bindings: Rules for distributing messages from exchanges to queues

**dd.How it works?**




**19. Usage**

This example will help you to understand working flow of AMQP protocol

**a. Prepare hardware and software**

**Hardware:**

+ Raspberry Pi 3

**Software:**

+ PuTTy (http://www.putty.org/)

+ SSH to Raspberry Pi 3 and execute the list of commands below (line by line)

```
sudo apt-get update
sudo apt-get install python python-pip
sudo pip install pika
```

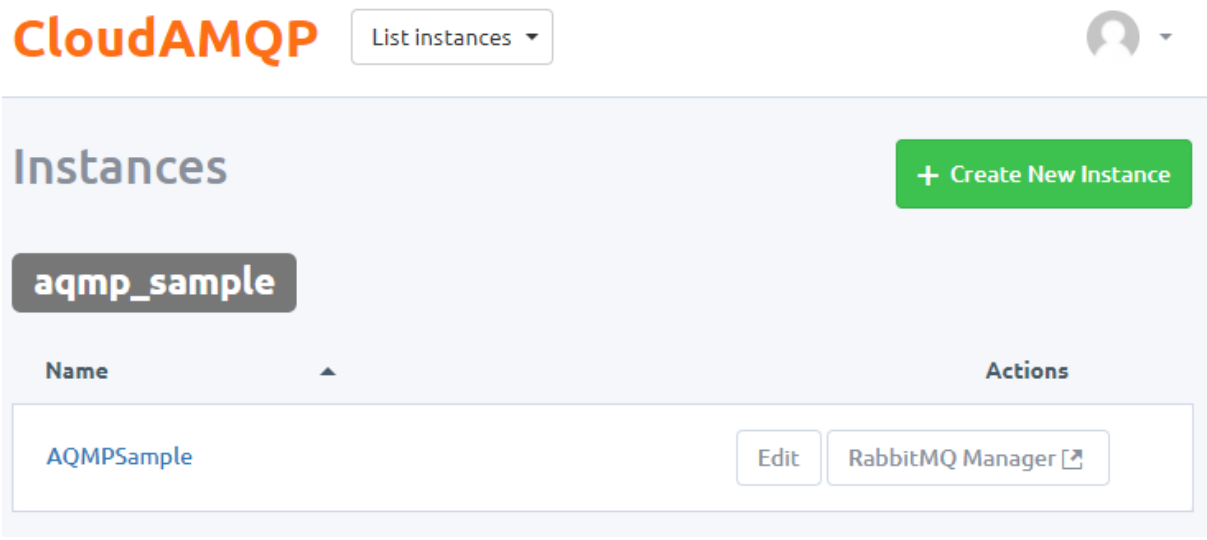
**l. Create account and instance on CloudAMQP**

**Create account:**

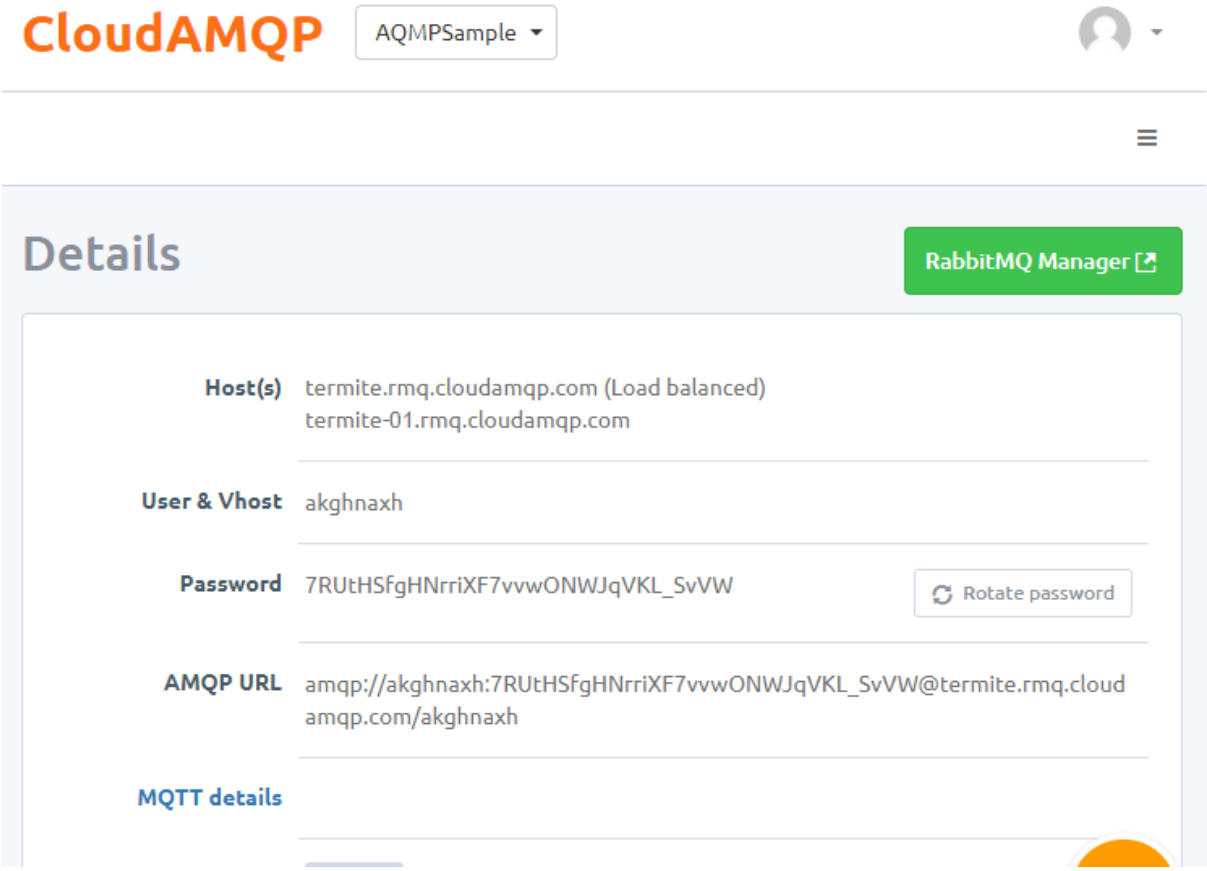
❖ You can easy to register account on CloudAMQP website (https://www.cloudamqp.com/)

**Create instance:**

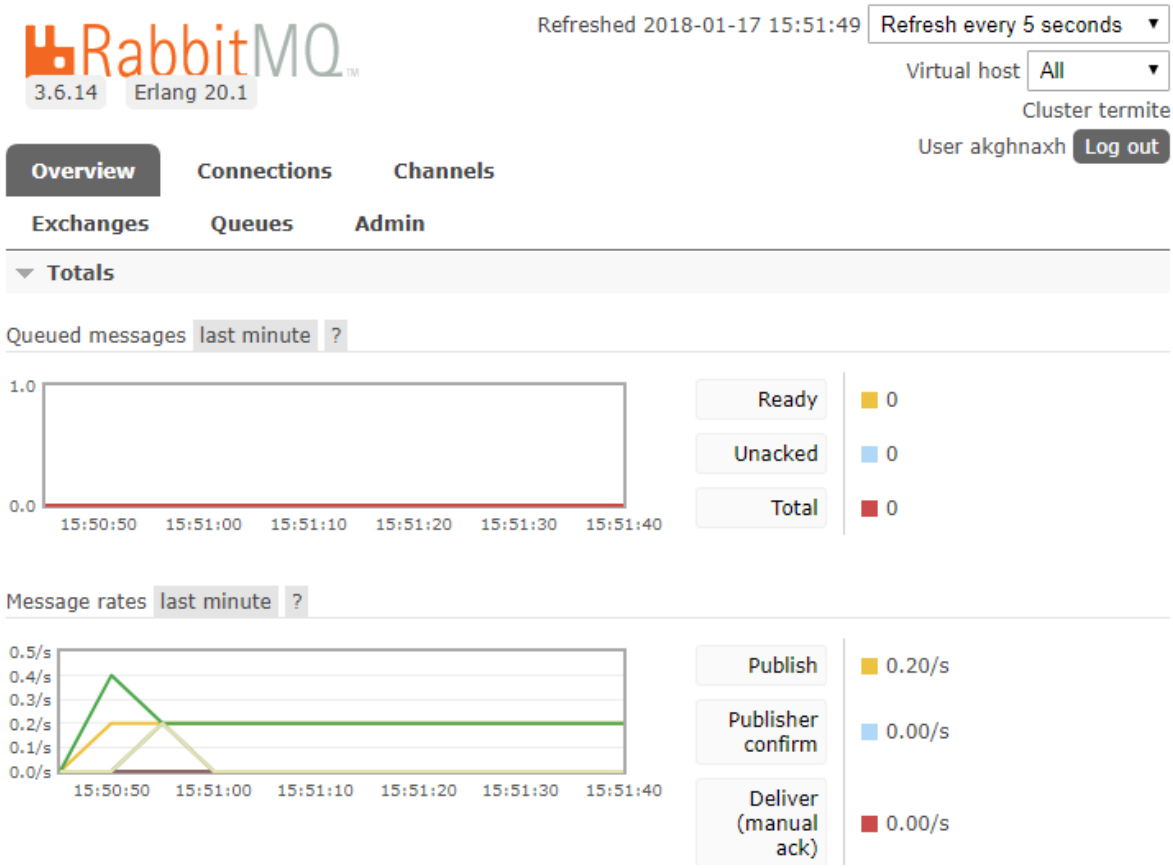
❖ Login with account which already register in previous step
❖ Click **Create New Instance**




❖ When your instance is created, press on details for your instance to find your **username**, **password** and **connection URL** for your cloud. Please remember this information it must be need for next step.




❖ Otherwise you can manage your instance by click to **RabbitMQ Manager** button.




### m. Create Publisher

- Checkout materials and samples code from gitlab
- Navigate to **Samples code\AMQP**
- Change CloudAMQP url

```
# Access the CLODUAMQP_URL environment variable and parse it
url = 'amqp://akghnaxh:7RUtHSfqHNrriXF7vvwONWJqVKL_SvVW@termite.rmq.cloudamqp.com/akghnaxh'
params = pika.URLParameters(url)
params.socket_timeout = 5
```

- Run Publisher
  *python AMQPPublisher.py*

### n. Create Consumer

- Navigate to **Samples code\AMQP**
- Change CloudAMQP url

```
# Access the CLODUAMQP_URL environment variable and parse it
url = 'amqp://akghnaxh:7RUtHSfqHNrriXF7vvwONWJqVKL_SvVW@termite.rmq.cloudamqp.com/akghnaxh'
params = pika.URLParameters(url)
params.socket_timeout = 5
```

- Run Consumer
  *python AMQPConsumer.py*

### o. Check results

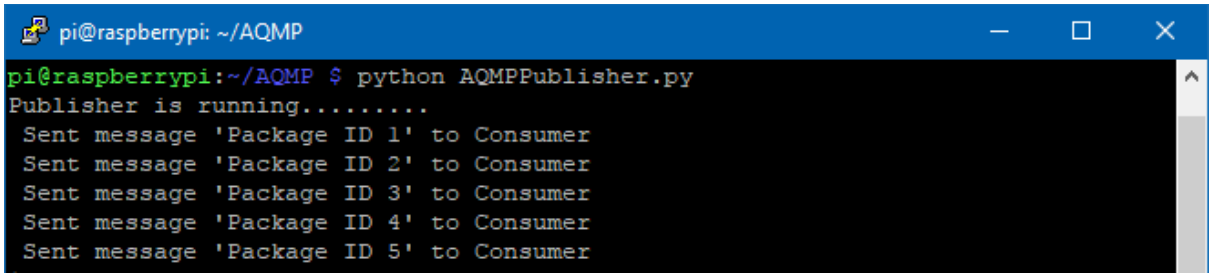



*Figure 13: Publisher AMQP*

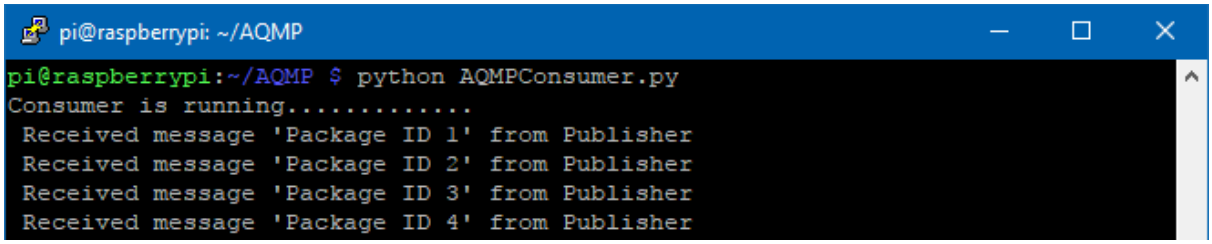



*Figure 14: Consumer AMQP*

**20. Exercises**

**a. Setup AMQP for Raspberry Pi 3**

**b. How to communicate with CloudAMQP using Nodejs?**

Ref: