



Session 6

Agenda

01

HTTP Protocol

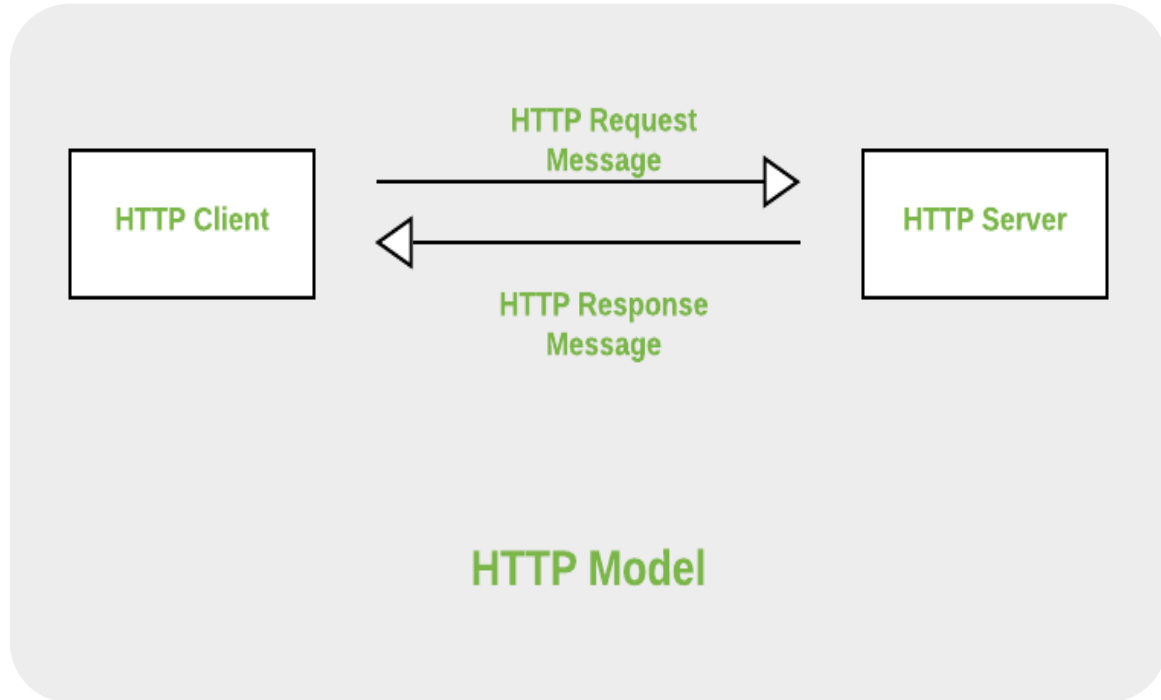
1. How it works
2. thingspeak.com

02

MQTT Protocol

How it works
ubidots.com

HTTP



- **Hyper Text Transfer Protocol**
- **This protocol has formed the foundation of data communication over the web**
- **It is the most common protocol that is used for IoT devices when there is a lot of data to be published**

HTTP

Stands for Uniform Resource Locator. A URL is nothing more than the address of a given unique resource on the Web.

- This protocol is responsible for the action that a server must take while sending information over the network

- When a **URL** is being entered into the browser, this protocol sends an HTTP request to the server and then an HTTP response is sent back to the browser

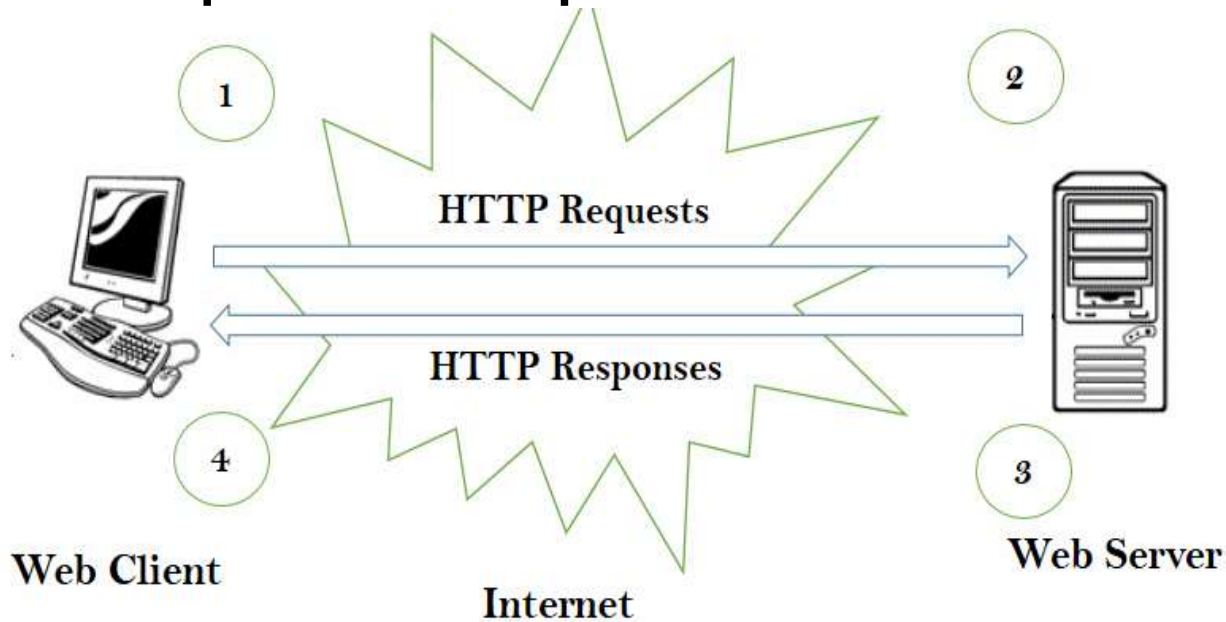
- This protocol is also responsible for the controlling of webpages on the World Wide Web

http://www.domain.com:1234/path/to/resource?a=b&x=y

protocol host resource path port query

HTTP Request / Response

- Communication between clients and servers is done by requests and responses:



1. A client (a browser) sends an HTTP request
2. A web server receives the request
3. The server runs an application to process the request
4. The server returns an HTTP response (output) to the browser
5. The client (the browser) receives the response

HTTP GET Request

method path protocol version

GET /some/path HTTP/1.1

Host: api.example.com

Origin: example.com

...more headers...

headers

blank line

The diagram illustrates the structure of an HTTP GET request. It shows the request line 'GET /some/path HTTP/1.1' with red brackets above it labeling 'method', 'path', and 'protocol version'. Below the request line are three lines of headers: 'Host: api.example.com', 'Origin: example.com', and '...more headers...'. A red bracket on the right groups these three lines under the label 'headers'. Below the headers is a red bracket labeled 'blank line', indicating the separator between the headers and the request body.

HTTP Response

HTTP/1.1 200 OK

Date: Sun, 08 Feb xxxx 01:11:12 GMT

Server: Apache/1.3.29 (Win32)

Last-Modified: Sat, 07 Feb xxxx

ETag: "0-23-4024c3a5"

Accept-Ranges: bytes

Content-Length: 35

Connection: close

Content-Type: text/html

<h1>My Home page</h1>

Status Line

Response Headers

Response
Message
Header

A blank line separates header & body

Response Message Body



Thingspeak.com

Practice 1:

1. **Create account on Thingspeak.com.**
2. **Create a new channel as a server.**
3. **Get the API keys and write the distance read by an ultrasonic sensor in the channel.**



Thingspeak.com

ThingSpeak™

Channels ▾

Apps ▾

Devices ▾

Support ▾

Commercial Use

How to Buy

YM

My Channels

New Channel

Search by tag



Help

Collect data in a ThingSpeak channel from a device, from another channel, or from the web.

Click **New Channel** to create a new ThingSpeak channel.

Click on the column headers of the table to sort by the

ThingSpeak™

Channels ▾

Apps ▾

Devices ▾

Support ▾

Commercial Use

How to Buy

YM

Access: Private

Private View

Public View

Channel Settings

Sharing

API Keys

Data Import / Export

+ Add Visualizations

+ Add Widgets

Export recent data

MATLAB Analysis

MATLAB Visualization

Thingspeak.com



Thingspeak.com

Configure widget parameters

Name

Distance

Field

Field 1

Update Interval

15

second(s)

Units

cm

Data Type

☐ Integer

☒ Decimal

2

(# of places)

Create

Cancel

Thingspeak.com

[Channels ▾](#)[Apps ▾](#)[Devices ▾](#)[Support ▾](#)[Commercial Use](#)[How to Buy](#)

Test project

Channel ID: **1460839**

Author: **mwa0000019090214**

Access: Private

[Private View](#)[Public View](#)[Channel Settings](#)[Sharing](#)[API Keys](#)[Data Import / Export](#)

Write API Key

Key

SPEGCQTKH9GQNXNJ

Generate New Write API Key

Help

API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.

API Keys Settings

- **Write API Key:** Use this key to write data to a channel. If you feel your key has been compromised, click **Generate New Write API Key**.
- **Read API Keys:** Use this key to allow other people to view your private channel feeds and charts. Click **Generate New Read API Key** to generate an additional read key for the channel.

Practice 1

Components:

- ❖ nodeMCU
- ❖ Ultrasonic sensor

description:

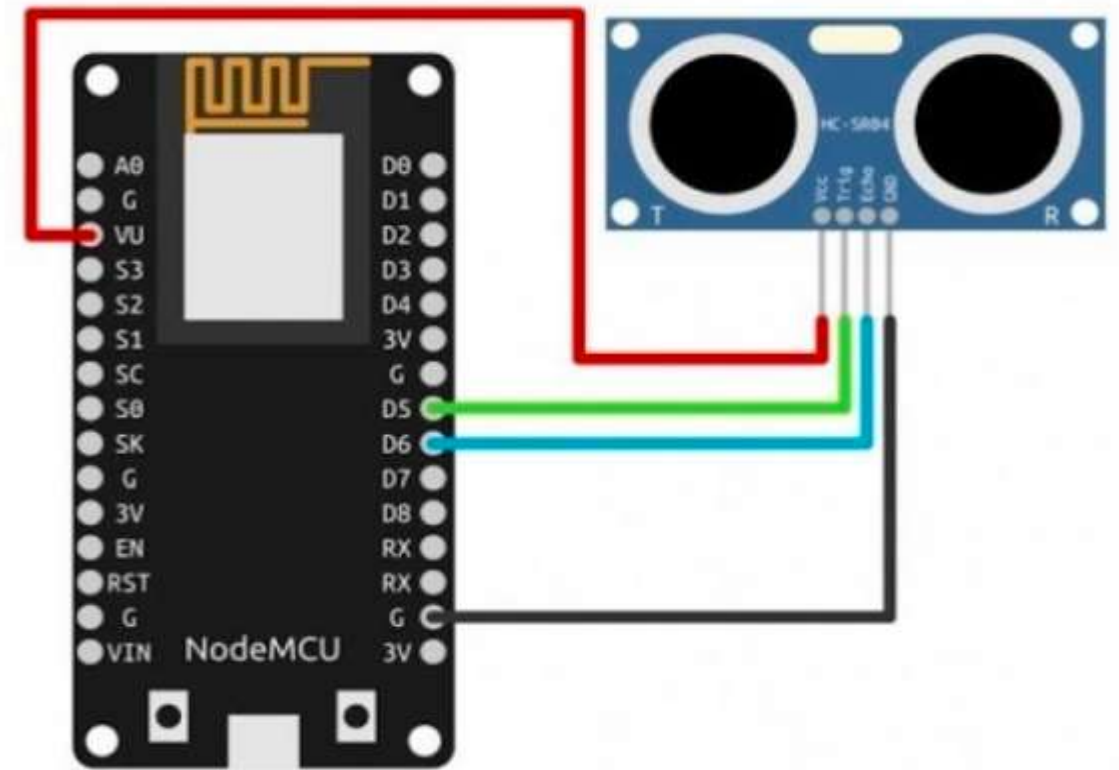
- ❖ Upload ultrasonic data to internet using thinkspeak



Practice 1 Solution

```
sketch_aug21a $
```

```
#include <ESP8266WiFi.h>
String apiWritekey = "WRITE API"; // replace with
const char* ssid = "Baraka"; // your wifi SSID name
const char* password = "000000000" ;// wifi password
const char* server = "api.thingspeak.com";
int echoPin=D6;
int trigPin=D5;
int duration;
int distance;
WiFiClient client;
```



Practice 1 solution

```
void setup() {  
  pinMode(trigPin, OUTPUT);  
  pinMode(echoPin, INPUT);  
  Serial.begin(115200);  
  WiFi.disconnect();  
  delay(10);  
  WiFi.begin(ssid, password);  
  
  Serial.println();  
  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.print("NodeMcu connected to wifi...");  
  Serial.println(ssid);  
  Serial.println();  
}
```

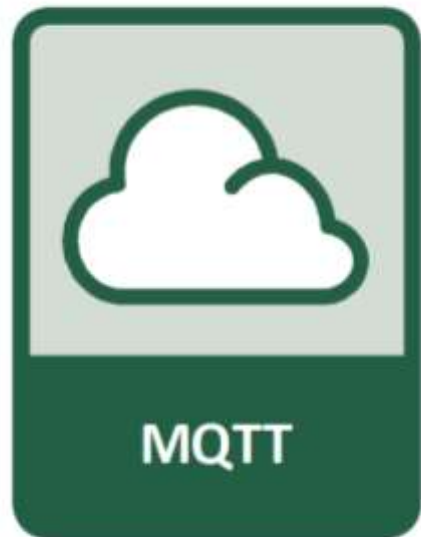
```
void loop() {  
  digitalWrite(trigPin, LOW);  
  delayMicroseconds(2);  
  // Sets the trigPin HIGH (ACTIVE) for 10 microseconds  
  digitalWrite(trigPin, HIGH);  
  delayMicroseconds(10);  
  digitalWrite(trigPin, LOW);  
  // Reads the echoPin, returns the sound wave travel time in microseconds  
  duration = pulseIn(echoPin, HIGH);  
  // Calculating the distance  
  distance = duration * 0.034 / 2;  
  //String val;  
  if (client.connect(server,80))  
  { /*if (Serial.available()) {  
    val=Serial.readString();  
  }*/  
    String tsData = apiWritekey;  
    tsData += "&field1=";  
    tsData += String(distance);  
    tsData += "\r\n\r\n";  
  }
```

Practice 1 solution

```
client.print("POST /update HTTP/1.1\n");
client.print("Host: api.thingspeak.com\n");
client.print("Connection: close\n");
client.print("X-THINGSPEAKAPIKEY: "+apiWritekey+"\n");
client.print("Content-Type: application/x-www-form-urlencoded\n");
client.print("Content-Length: ");
client.print(tsData.length());
client.print("\n\n"); // the 2 carriage returns indicate closing of Header fields & starting of data
client.print(tsData);
Serial.print("Distance ");
Serial.print(distance);
Serial.println("uploaded to Thingspeak server....");
}
client.stop();
// thingspeak needs minimum 15 sec delay between updates
// delay(15000);
}
```



MQTT



- Message Queuing Telemetry Transport
- This protocol was invented because of the need of a protocol for minimal battery loss and minimal bandwidth

Bandwidth is measured as the amount of data that can be transferred from one point to another within a network in a specific amount of time

MQTT

Characteristics

```
graph TD; A[Characteristics] --> B[1.Data agnostic]; A --> C[1.Binary]; A --> D[Efficiency]; A --> E[Bi-directional]
```

1.Data
agnostic

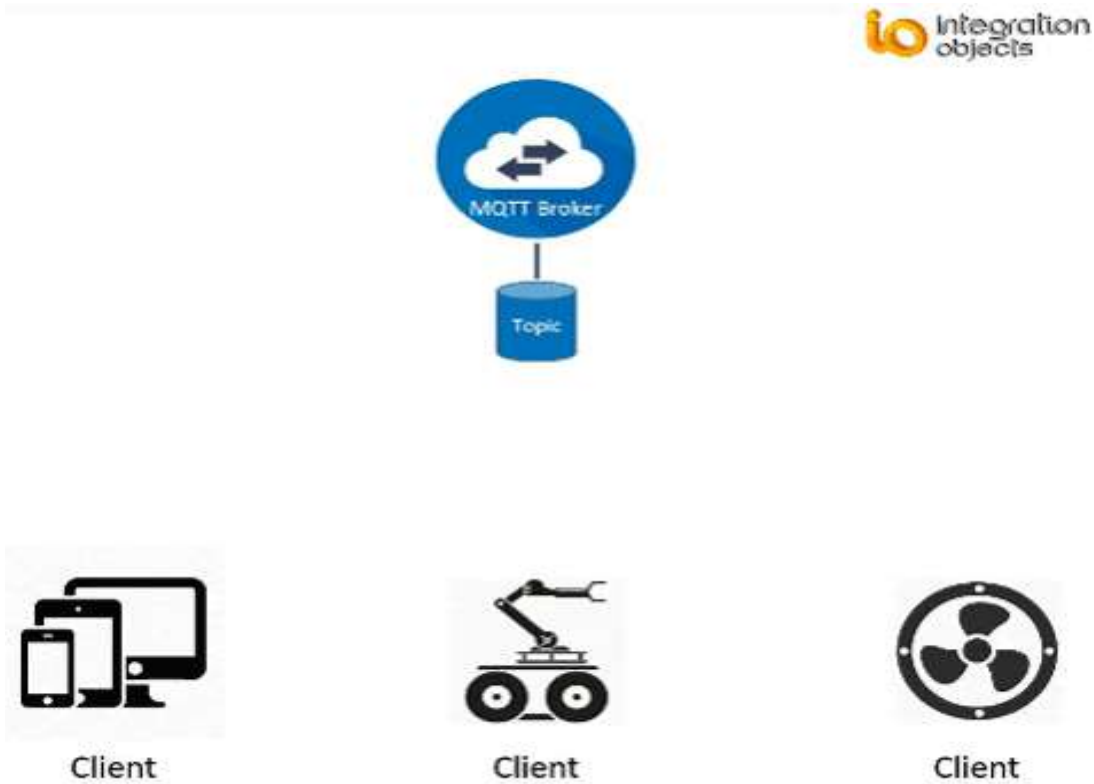
1.Binary

Efficiency

Bi-
directional

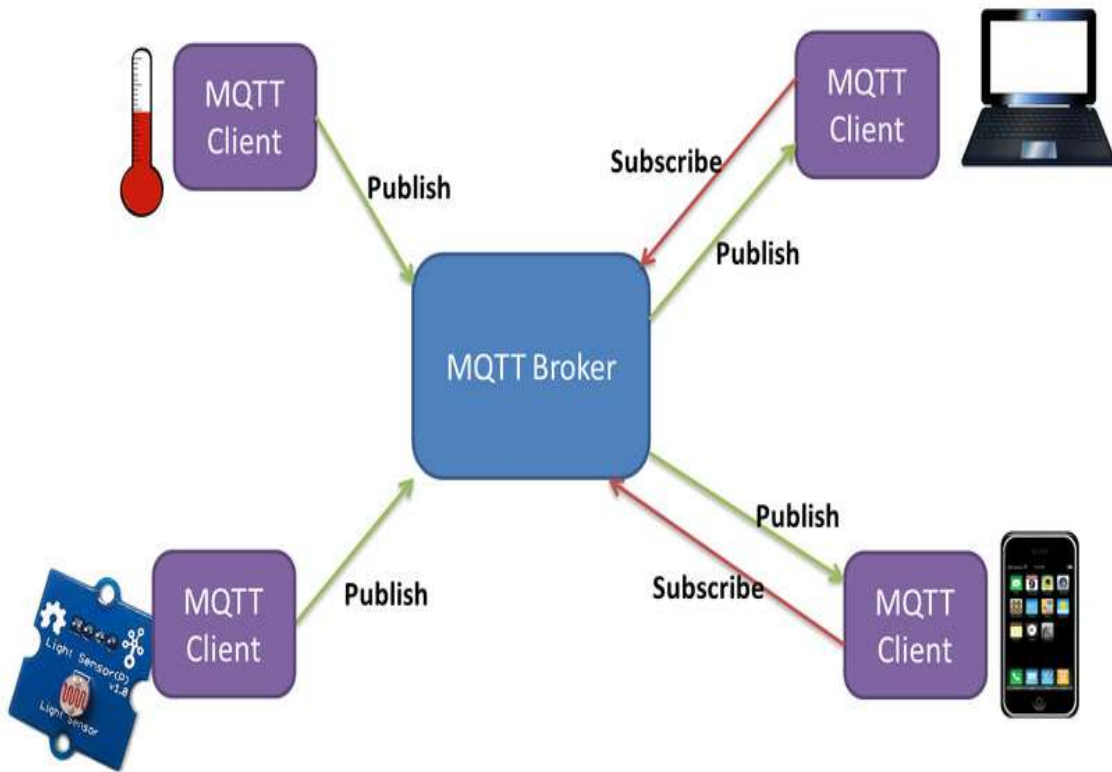


MQTT Publish / Subscribe



- The pub/sub model decouples the client that sends a message (the publisher) from the client or clients that receive the messages (the subscribers)
- The connection is handled by a third component (the broker)
- The job of the broker is to filter all incoming messages and distribute them correctly to subscribers

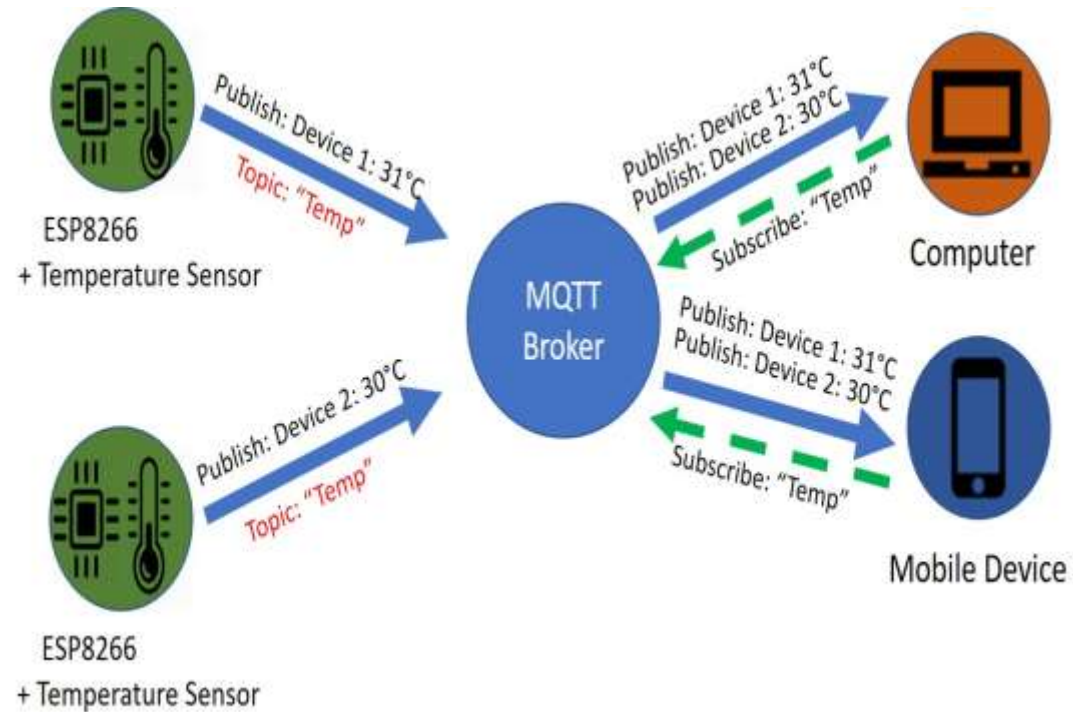
MQTT Publish / Subscribe



Clients:

- **Publish client.**
- **Subscribe client.**
- **Both publish and subscribe client.**

MQTT Publish / Subscribe



The most important aspect of pub/sub is the decoupling of the publisher of the message from the recipient (subscriber)

- Space decoupling “no need to know the port”.
- Time decoupling.
- Synchronization decoupling.

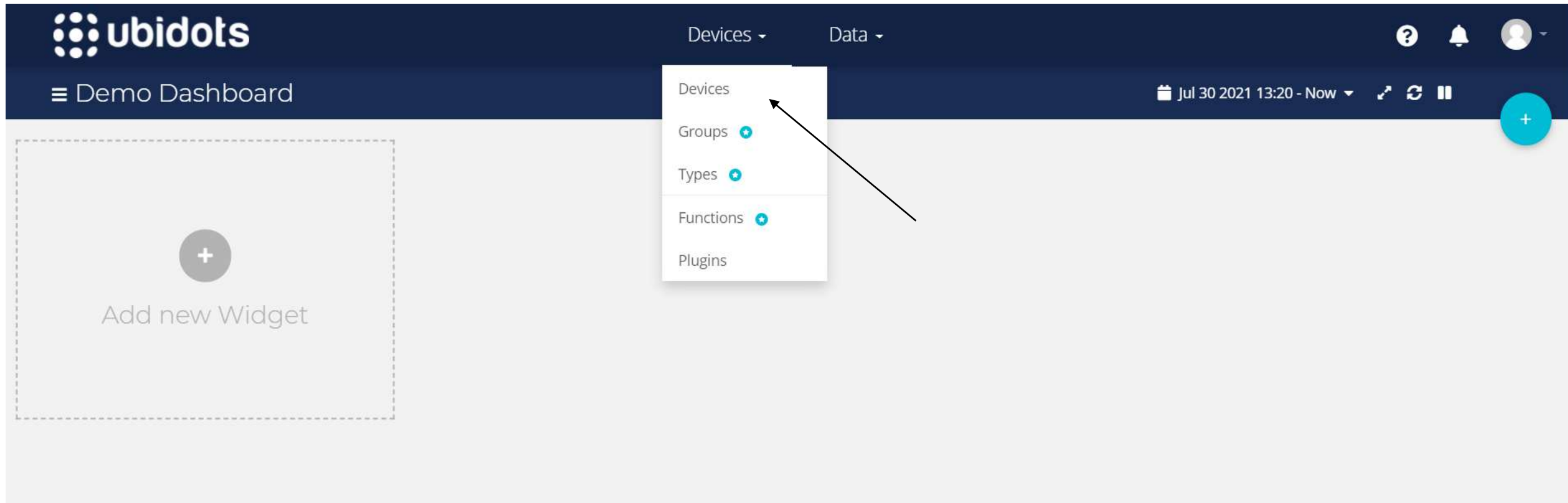
Ubidots.com

Practice 2:

1. **Create account on**
2. **Download Ubidots-MQTT-Esp library**
3. **Include the previous library in Arduino IDE**
4. **From Arduino IDE in Manage libraries download PubSubClient library**
5. **Enter a value on serial and see results on ubidots using ubidots widgets.**



Ubidots.com



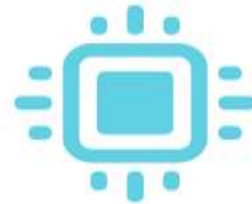
Ubidots.com



Devices

Devices ▾

Data ▾




There are no Devices created

Click the button below to start sending dots to your first Device and begin to unlock the value of your data

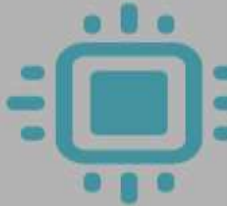
Create Device

Ubidots.com



Devices ▾

Devices



There are no Devices

Click the button below to start sending dots to your first Device

Create Device

Add New Device

Create a new blank device shell, or automatically create new devices using the below libraries and documentation. The first time a dot is sent to Ubidots, a new Device is automatically created.

CONNECTIVITY

Ethernet

Cellular

LoRaWAN

LTE-M

NB-IoT

Sigfox

WiFi

HARDWARE TYPE

Chips & Modules

Dev Kits

Gateways


Production Ready

INTEGRATION TYPE


Library

Plug-n-Play


Tutorial




Blank Device




Adafruit




Alorium Technology




Ambient Weather




AMPLIFIED ENGINEERING




Arduino




Controllino



Decentlab



Dragino Gateway



Oyster

Ubidots.com

The image shows a screenshot of the Ubidots.com web interface. At the top, there is a dark blue header with the Ubidots logo on the left and navigation links for 'Devices' and 'Data' on the right. Below the header, the main content area has a light gray background with a world map. On the left side, there is a sidebar with a 'Devices' link. The main area displays a device detail card for a device named 'test'. The card includes fields for 'API Label', 'ID', 'Token', and 'Tags'. The 'Token' field is highlighted with a blue cloud annotation that says 'Copy Token'. An arrow points from this cloud to the 'Token' field. To the right of the device card, there is a dashed box containing a plus sign icon and the text 'Add Variable'. A blue cloud annotation above this box says 'Add your code variables', with an arrow pointing from the cloud to the plus sign icon. The bottom of the image features a dark blue silhouette of a city skyline.

Copy Token

Add your code variables

Ubidots

Devices Data

← Devices

SET LOCATION

test

61053a0803372f000d1fa001

Token

Tags

Add Variable

Ubidots.com

The screenshot displays the Ubidots.com web interface. At the top, there is a dark blue header with the Ubidots logo on the left, navigation links for 'Devices' and 'Data' in the center, and user icons (help, notifications, profile) on the right. Below the header, a dark blue sidebar on the left contains a '← Devices' link. The main content area features a light gray world map background. On the left, a device card for 'test' is shown with a teal header. It includes fields for 'Description' (with a 'Change description' link), 'API Label' (set to 'test'), 'ID' (61053a0803372f000d1fa001), 'Token' (masked with dots), and 'Tags'. To the right of the device card is a variable card for 'var' with an orange header. It shows 'Last activity: No last activity'. Further right is a dashed box with a plus icon and the text 'Add Variable'. At the bottom of the page, there is a dark blue silhouette of a city skyline.

ubidots

Devices ▾ Data ▾

← Devices

test

Description
Change description

API Label ⓘ
test

ID ⓘ
61053a0803372f000d1fa001

Token
.....

Tags

var

Last activity:
No last activity

+
Add Variable

SET LOCATION

Ubidots.com



Devices ▾

Data ▾



≡ Demo Dashboard

Jul 30 2021 13:56 - Now ▾

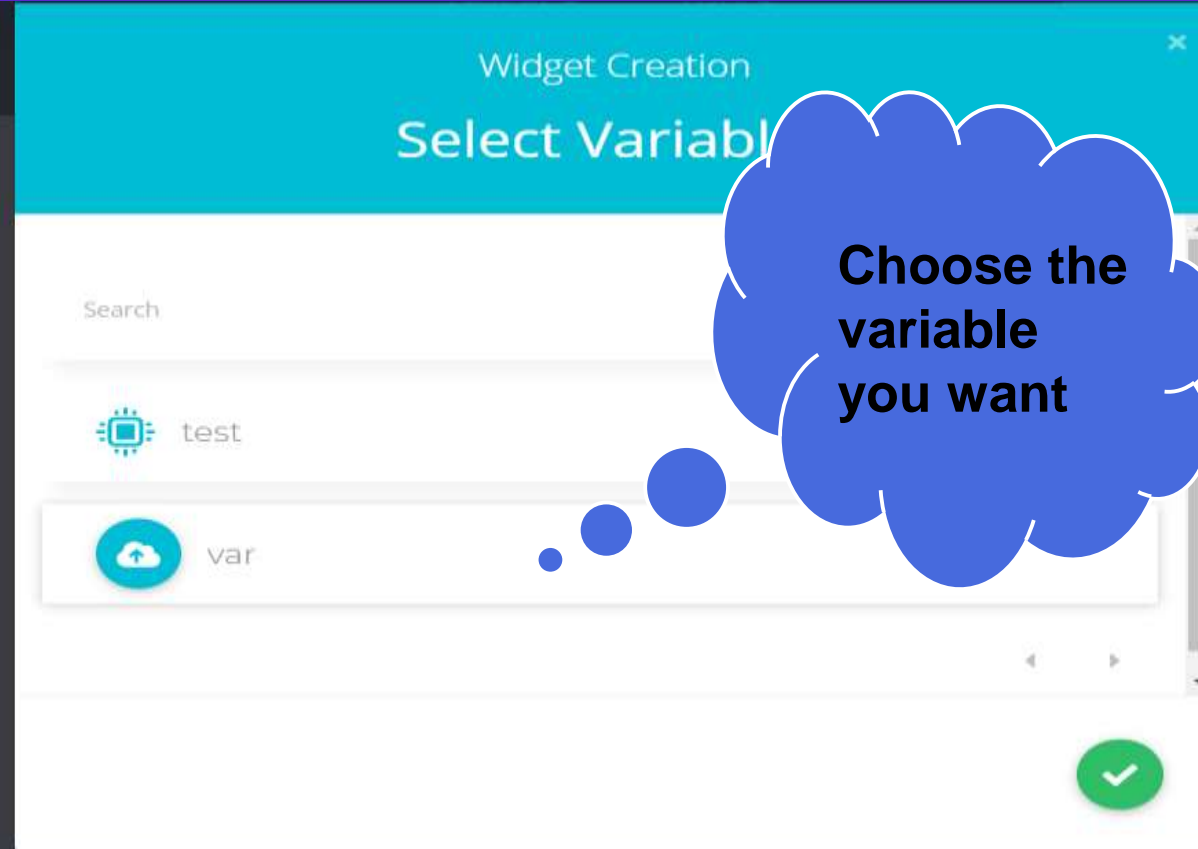


Add new Widget



**Go back to
the
dashboard**

Ubidots.com



Practice 2

Components:

- ❖ nodeMCU
- ❖ LED

description:

- ❖ Control the led from anywhere using ubidots

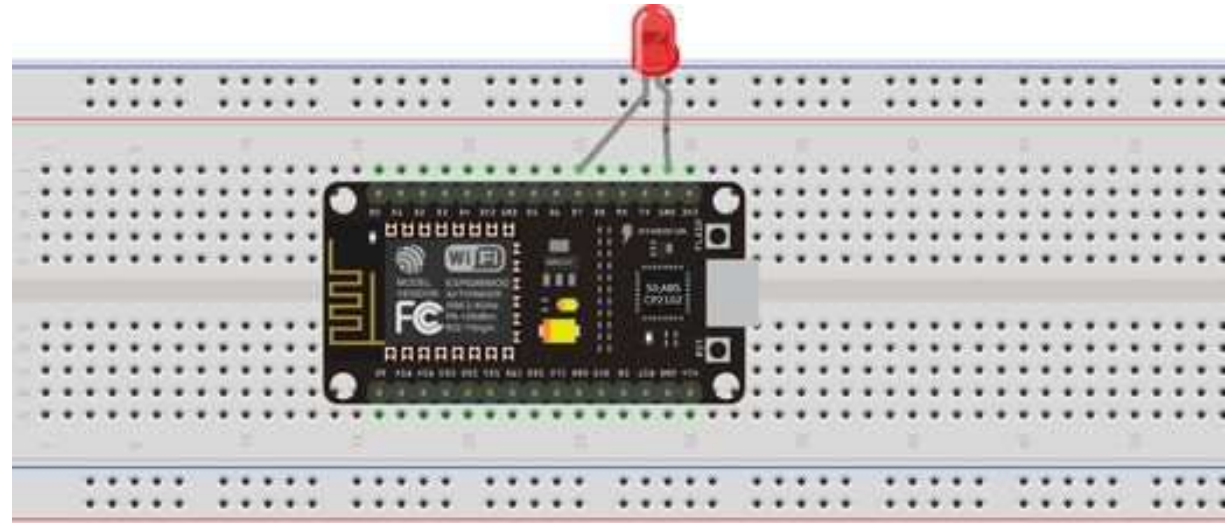


Practice 2 solution

```
#include "UbidotsESPMQTT.h"
#define TOKEN "your token" // Your Ubidots TOKEN
#define WIFINAME "Baraka" //Your SSID
#define WIFIPASS "00000000" // Your Wifi Pass
#define DEVICE_LABEL "MCU" // Put here your Ubidots device label
#define VARIABLE1 "test" // Put here your Ubidots variable label
int ledPin = D0;
String value="";
Ubidots client(TOKEN);

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();
  if (payload[0]==1){
    digitalWrite(ledPin, HIGH);
    Serial.println("1");
  }
  else if (payload[0]==0){
    digitalWrite(ledPin, LOW);
    Serial.println("0");
  }
}
```



Practice 2 solution

```
void setup() {  
  // put your setup code here, to run once:  
  
  client.setDebug(true); // Pass a true or false bool value to activate debug messages  
  Serial.begin(115200);  
  client.wifiConnection(WIFINAME, WIFIPASS);  
  client.begin(callback);  
  client.ubidotsSubscribe(DEVICE_LABEL, VARIABLE1); //Insert the dataSource and Variable's Labels  
  pinMode(ledPin, OUTPUT);  
  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
  if (!client.connected()) {  
    client.reconnect();  
    client.ubidotsSubscribe(DEVICE_LABEL, VARIABLE1); //Insert the dataSource and Variable's Labels  
  
  }  
  client.loop();  
}
```

HTTP or MQTT ?



HTTP VS MQTT



VS



Full form :

**Message Queue
Telemetry Transport**

**Hyper Text Transfer
Protocol**

Architecture :

**publish/subscribe
architecture**

**request/response
(Client/Server) architecture**

message size :

small

large

HTTP VS MQTT



VS



Message format :

binary

ASCII format

distribution

1 to 0/1/N

one to one only

Complexity :

Simple

**Client more
complex**

The End

THANK YOU

