

# **ARDUINO Y PROGRAMACION EN EL DESARROLLO DE PROYECTOS DE CIENCIAS Y ELECTRONICA**

## **SESION 3**

# INTERNET DE LAS COSAS



IOT



¿Qué es IoT?

*Para que sirva?*

# Internet de las Cosas IoT

- Se refiere a la posibilidad de que sensores-dispositivos que arrojan datos puedan ser interconectados a Internet a través de algún medio de transmisión y subsecuente protocolo de comunicación.
- Sobre lo anterior se implementan soluciones y se toman acciones, posibilitando el intercambio de datos y flujo ininterrumpido de información que los dispositivos conectados entregan.

# Internet de las Cosas IoT - ¿Cómo funciona?

- Mediante componentes(cruciales) interconectados:

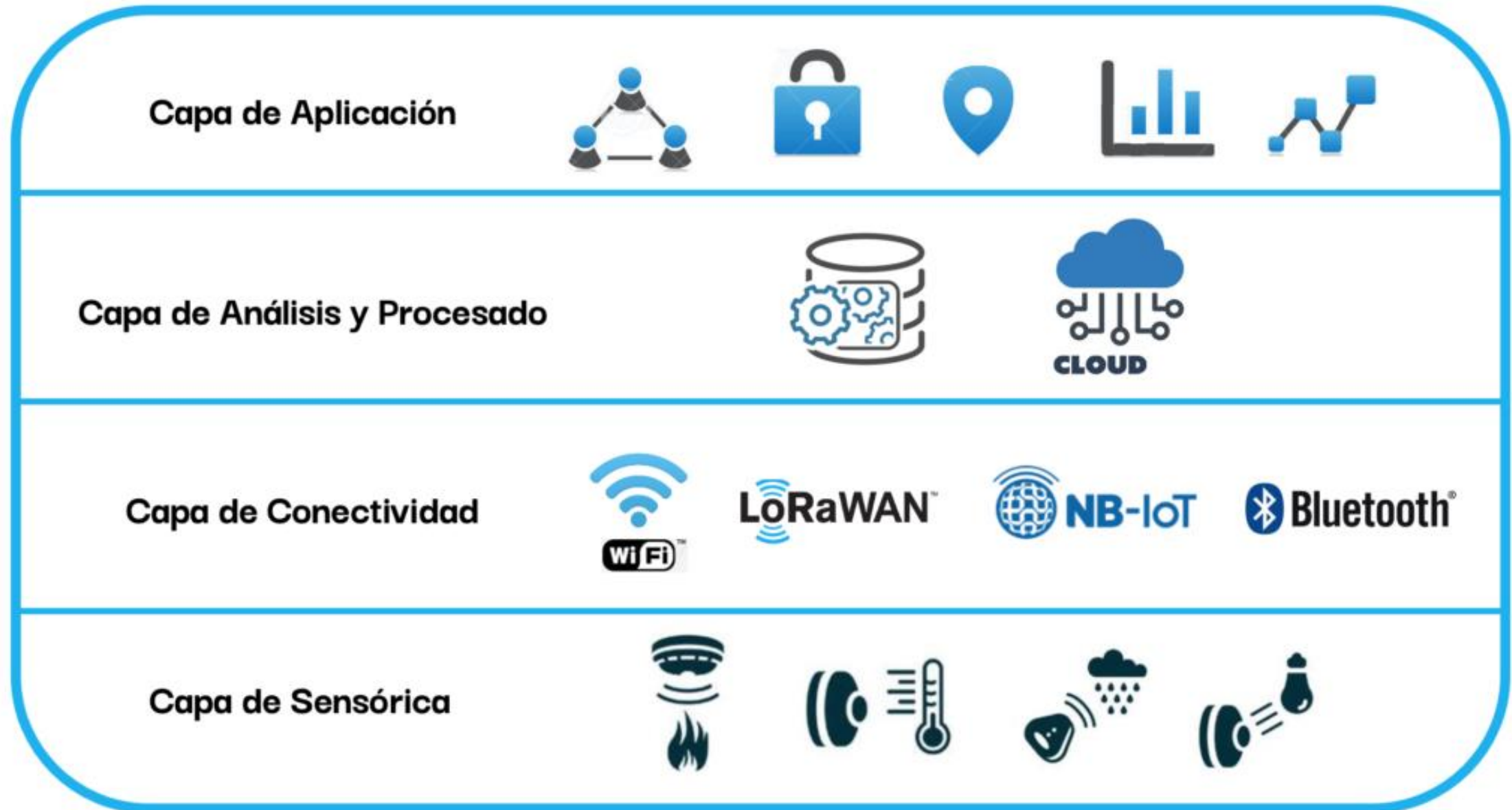
1. **Sensores y Actuadores.**- recopilan información del medio o entorno

1. **Conectividad.**- WiFi, Bluetooth, Red 4G, 5G →Envío de datos a servidor centralizado.

1. **Aplicación/Servicio IoT.**- Procesamiento de datos recopilados mediante algoritmos(machine learning, IA) → Obtención de información practica utilizable.

1. **Interfaz grafica de usuario( GUI).**- Permite administrar/monitorear lo anterior (*Cfgs, preferencias, rendimiento, estados*)

# Internet de las Cosas IoT - ¿Cómo funciona?



# IoT: Entonces?

- Este ciclo continuo:

**Recopilación → Procesamiento → Respuesta** permite la automatización de tareas y el anticipo de necesidades

# IoT: Casos comunes

- **Dispositivos Hogar Inteligentes.-** pueden ayudar en la eficiencia energética, seguridad y comodidad(enchufes inteligentes, termostatos, cámaras)
- **Tecnologías Wearables.-** pueden hacer seguimiento del estado de salud de una persona(frecuencia cardiaca, sueño, pasos dados)
- **Dispositivos Médicos.-** complementan la atención y diagnostico medico
- **Ciudades Inteligentes.-** infraestructura, salud y medio ambiente(contaminación)
- **Industria logística,, transporte y Agricultura.-** Los disp. IoT pueden seguir el rendimiento o tiempos de maquinaria, inventarios para luego realizar una gestión de activos aumentando productividad



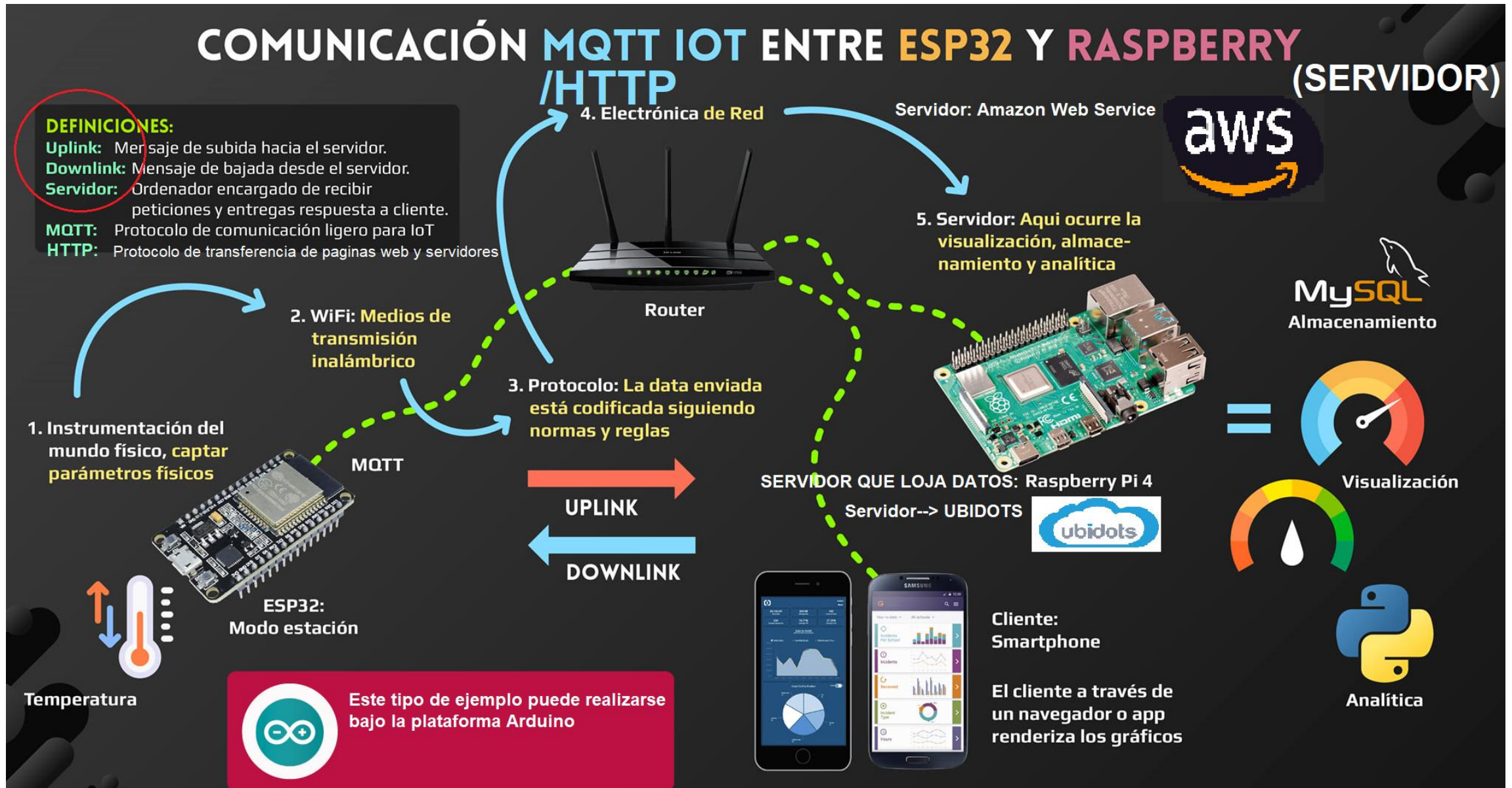
# IoT - ¿Por qué es importante?

- **Aumenta eficiencia y Productividad:** ex. Mantenimiento predictivo.
- **Genera ahorros de costos:** ex. Eficiencia energía eléctrica
- **Mejora la toma de decisiones:** basadas en datos , optimizar procesos ,el anticipo de necesidades y comprensión de comportamiento del cliente

# IoT - ¿Cómo se estructura un proyecto?



# IoT – Topología típica a nivel de HW: WiFi



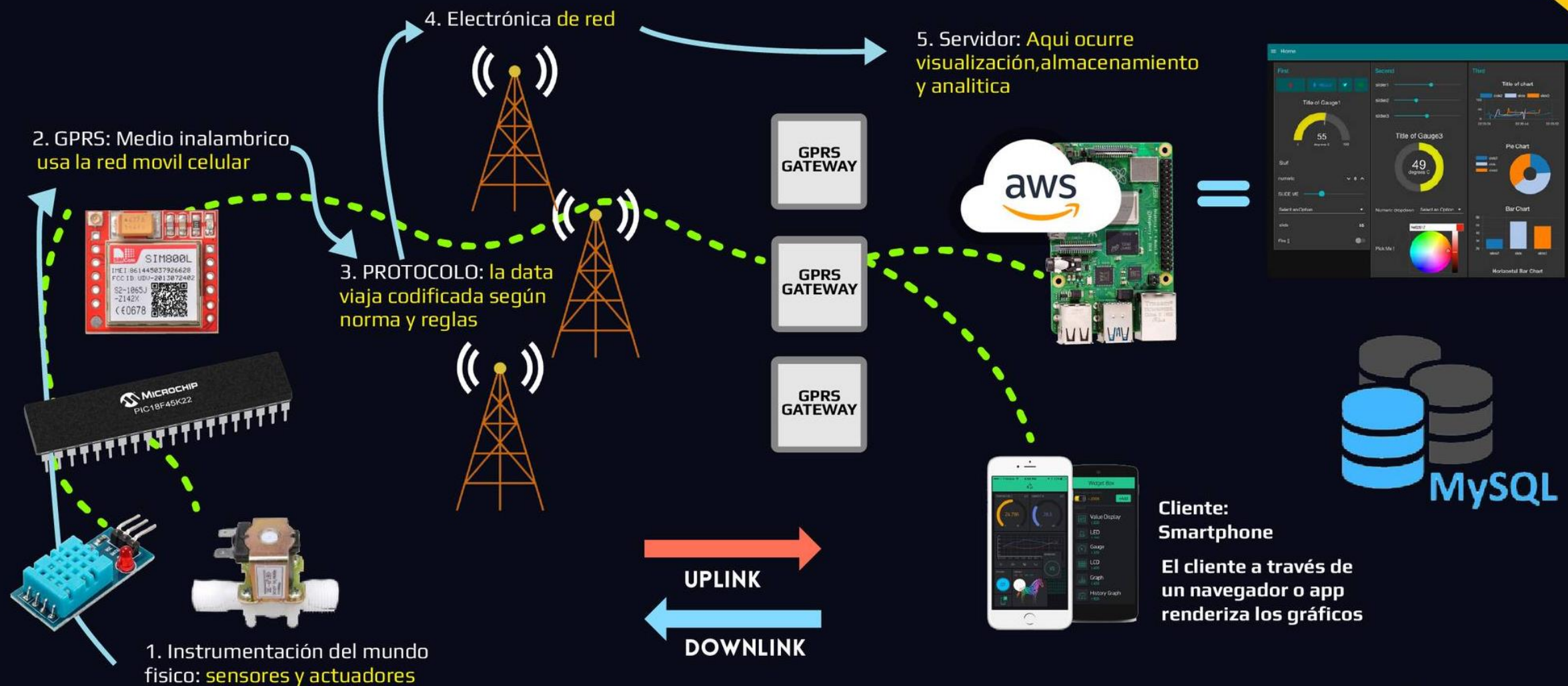
# IoT – Topología típica a nivel de HW: Sigfox





# IoT – Topología típica a nivel de HW: **GSM/GPRS**

## COMUNICACIÓN GPRS IOT ENTRE MCU Y SERVIDOR



# Medio de Transmisión

- Involucra el acceso un medio de comunicación puede ser cableado(ex. Ethernet) o inalámbrico(ex. **WiFi**)
- Suele haber un punto de acceso **(AP)** o puerta de enlace(**Gateway**) al cual conectarse.
- Un dispositivo basado en **MCU** que recolecte datos debe contar primero **con una capa de código** para acceder al medio(**WiFi**) y poder evaluarse su estado de conectividad (**Direccion IP ,Direccion MAC , conectado?** )

# Medio de Transmisión

- La placa **NodeMCU** basada en ESP8266 suele tener HW de WiFi incorporado así como demás pines para poder leer sensores o escribir en actuadores.
- **OBS:** Se debe contar con la data de instrumentación del mundo físico y el [driver USB instalado de Silicon Labs](#)
- **OBS:** *Cuidado* , Los niveles lógicos son de 3.3V
- Ver a continuación:

# ESP8266 NODE MCU

ESP8266-12E  
Wi-Fi Chip

2.4GHz  
Antenna

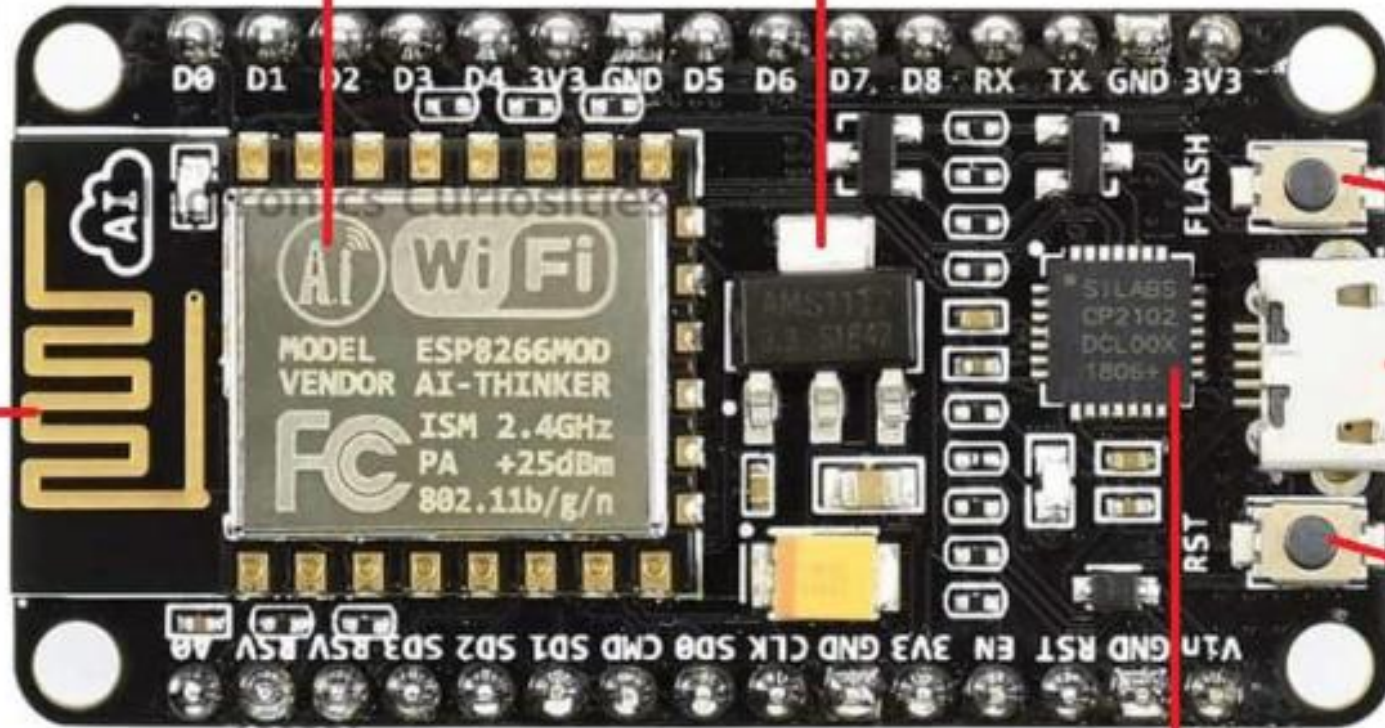
3.3V Voltage Regulator IC

Flash Button

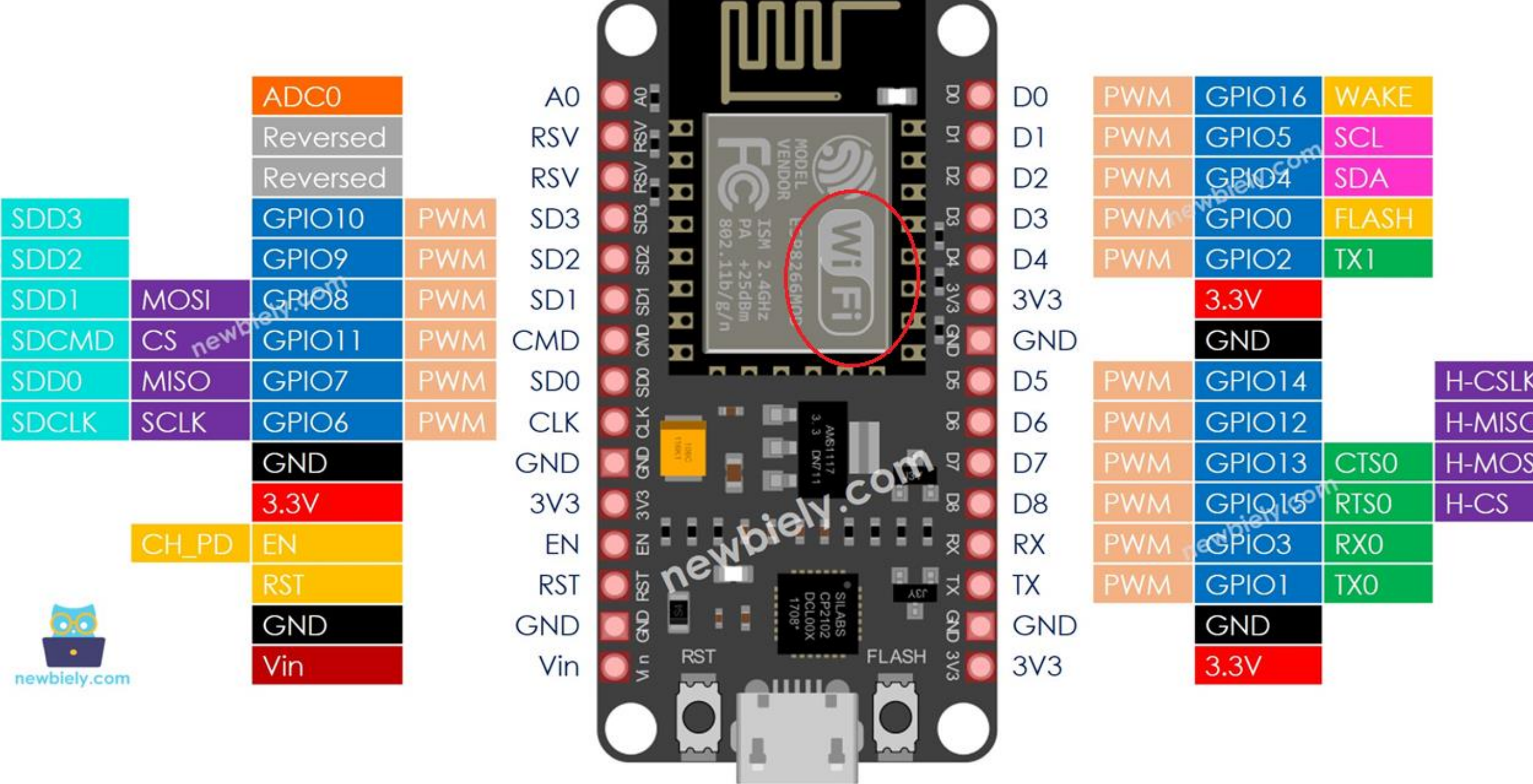
Micro USB  
Connector

Reset Button

USB to TTL  
Converter IC







ESP8266 NodeMCU Pinout

# API Librería Arduino: ESP8266

- En modo estación (**STA**) :

```
#include <ESP8266WiFi.h>
```

```
wl_status_t begin(const char* ssid, const char *passphrase = NULL, int32_t channel = 0, const uint8_t* bssid = NULL, bool connect = true);  
wl_status_t begin(char* ssid, char *passphrase = NULL, int32_t channel = 0, const uint8_t* bssid = NULL, bool connect = true);  
wl_status_t begin(const String& ssid, const String& passphrase = emptyString, int32_t channel = 0, const uint8_t* bssid = NULL, bool connect = true);  
wl_status_t begin();
```

//The argument order for ESP is not the same as for Arduino. However, there is compatibility code under the hood  
//to detect Arduino arg order, and handle it correctly. Be aware that the Arduino default value handling doesn't  
//work here (see Arduino docs for gway/subnet defaults). In other words: at least 3 args must always be given.

```
bool config(IPAddress local_ip, IPAddress gateway, IPAddress subnet, IPAddress dns1 = (uint32_t)0x00000000, IPAddress dns2 = (uint32_t)0x00000000);
```

```
// STA network info
```

```
IPAddress localIP();
```

```
uint8_t * macAddress(uint8_t* mac);
```

```
String macAddress();
```

```
IPAddress subnetMask();
```

```
IPAddress gatewayIP();
```

# Protocolo de comunicación

- La data a transmitir sigue normas y reglas: **HTTP**, MQTT, FTP, SMNTP
- **HTTP** fue de los primeros al surgir internet aun se usa



# HTTP

Es un protocolo de comunicación utilizado en la World Wide Web (WWW) para la transferencia de datos, especialmente documentos de hipertexto, como páginas web y recursos multimedia.



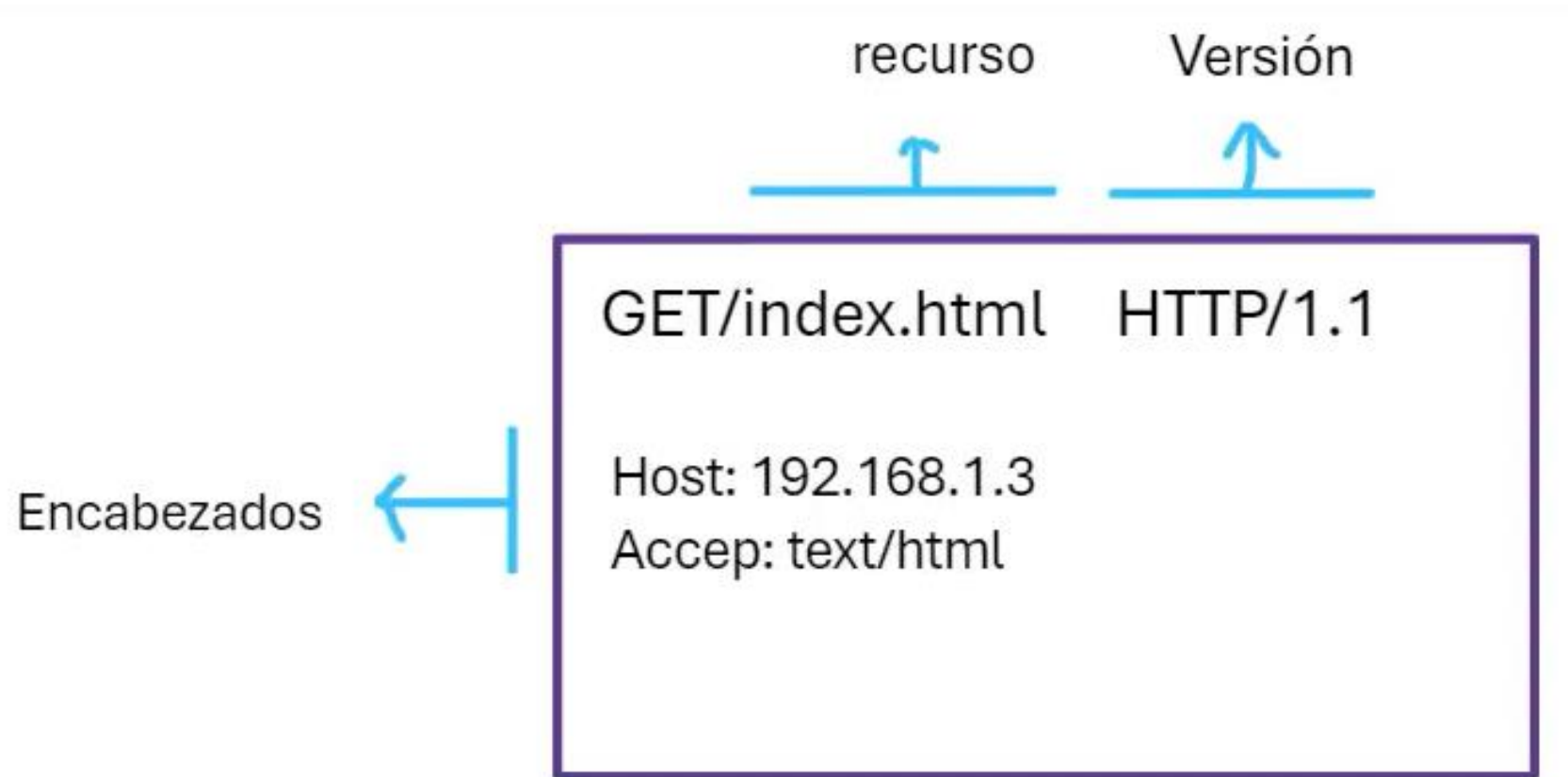
# Protocolo de comunicación: HTTP

- Trabaja en base al método **Petición(cliente)  $\leftrightarrow$  Respuesta(servidor)**
- Las peticiones se hacen hacia **dirección IP del servidor**, cuya estructura seria: **[protocolo]://[IP servidor]/[recurso]**
- Por decir: <http://192.168.1.3:80/index.html>
- O también <http://www.youtube.com/index.html>
- El **puerto (80)** suele ser tácito y el **HOME** lo completa los navegadores



# Protocolo de comunicación: HTTP

- Una vez hecha la petición , el cliente debe incluir el método(**GET**) y las **cabeceras o headers**: como formato de texto plano



# Protocolo de comunicación: HTTP

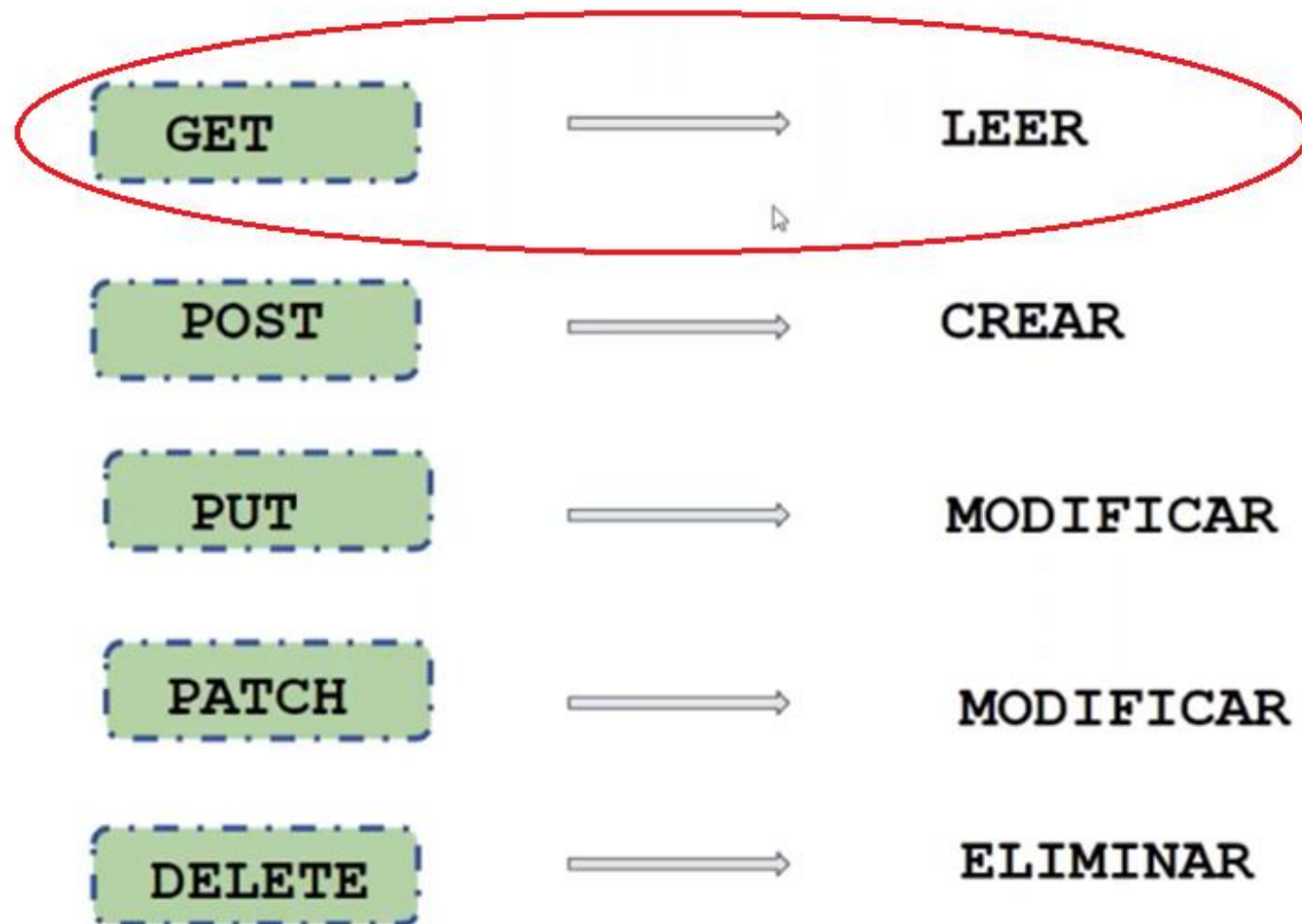
- Una vez hecha la **petición**, el servidor da una **respuesta** mediante un código **(200)**, cabeceras y **contenido(html)**





# HTTP

## MÉTODOS





# API Librería Arduino: ESP8266

- Como **cliente** `#include <ESP8266HTTPClient.h>`
- En el método **begin()** va como arg la url: **[protocolo]://[IP servidor]/[recurso]**

```
// Note that WiFiClient's underlying connection *will* be captured
```

```
bool begin(WiFiClient &client, const String& url);
```

```
bool begin(WiFiClient &client, const String& host, uint16_t port, const String& uri = "/", bool https = false);
```

```
/// request handling
```

```
int GET();
```

```
int DELETE();
```

```
int POST(const uint8_t* payload, size_t size);
```

```
int POST(const String& payload);
```

```
int PUT(const uint8_t* payload, size_t size);
```

```
int PUT(const String& payload);
```

```
int PATCH(const uint8_t* payload, size_t size);
```

# API Librería Arduino: ESP8266

- Enseguida debe enviarse el método **GET** a fin de enviar la petición al servidor.
- El contenido en lenguaje **html** que viene desde el servidor se le suele llamar **payload** y se obtiene como texto crudo: **.getString()**

```
int writeStream(Stream &stream);  
const String& getString(void);
```

# ***EJEMPLO METODO GET***

***CODIGO : 200 OK***

# APIs Web

- Son servidores que dan el servicio de entrega y/o almacenamiento de datos útiles: ***Open Weather Map, Meteoblue***...
- En el caso de *Open Weather Map*, es una empresa que da servicios de meteorología ,mediante sensores instalados en varias partes del planeta.
- Una de sus sub-API es “*Air Pollution*”, entrega información de agentes contaminantes: SO<sub>2</sub>, NO<sub>2</sub>, PM<sub>10</sub>, O<sub>3</sub>, CO

# API Web: Open Weather Map

- En su **doc** suele haber un formato de petición :

**`http://api.openweathermap.org/data/2.5/air_pollution?lat={lat}&lon={lon}&appid={API key}`**

- Por ejemplo:

API Key: abcf4542035084fa5ca8e4697b6bccc6

Lat: (ver **google maps**) -12.056860

Long: ver google maps -77.061921

**`http://api.openweathermap.org/data/2.5/air_pollution?lat=-12.060496&lon=-77.059052&appid=abcf4542035084fa5ca8e4697b6bccc6`**

# API Web: Open Weather Map(respuesta)

```
{
  "coord": {
    "lon": -77.0585,
    "lat": -12.0583
  },
  "list": [
    {
      "main": {
        "aqi": 2
      },
      "components": {
        "co": 357.15,
        "no": 0.77,
        "no2": 16.97,
        "o3": 40.05,
        "so2": 7.27,
        "pm2_5": 8.7,
        "pm10": 21.65,
        "nh3": 1.76
      },
      "dt": 1731105004
    }
  ]
}
```

# FORMATO TRAMA JSON

```
{"a":"Peru", "b":"Ecuador",...."propiedad":"valor"}
```

- Las tramas JSON son **strings o cadenas de caracteres**.
- Pero para acceder a sus propiedades , es mejor contar con un librería cuyos tipos de dato facilita el acceso.

# API Librería Arduino: ESP8266

- El contenido en formato JSON se suele manejar con la librería:

```
#include <ArduinoJson.h>
```

- Y se debe crear el documento (doc) en formato JSON para luego deserializar el payload en dicho doc.

```
//Crear el documento en formato JSON  
StaticJsonDocument<256> doc;
```

```
//Descomponer /deserializar el 'payload' en el  
documento json (pasar a JSON)  
deserializeJson(doc,payload);
```



# API Librería Arduino: ESP8266

Y finalmente se hace el acceso como si se tratara de arreglos []

```
//Guardar en variables los valores extraídos  
float co=doc["list"][0]["components"]["co"];  
float no=doc["list"][0]["components"]["no"];  
float no2=doc["list"][0]["components"]["no2"];
```

# API Librería Arduino: ESP8266

- Podemos consultar su documentación acerca de JSON en :

[https://arduinojson.org/?utm\\_source=meta&utm\\_medium=library.properties](https://arduinojson.org/?utm_source=meta&utm_medium=library.properties)



# HTTP

## MÉTODOS

GET



LEER

POST



CREAR

PUT



MODIFICAR

PATCH



MODIFICAR

DELETE



ELIMINAR

# Protocolo de comunicación: HTTP

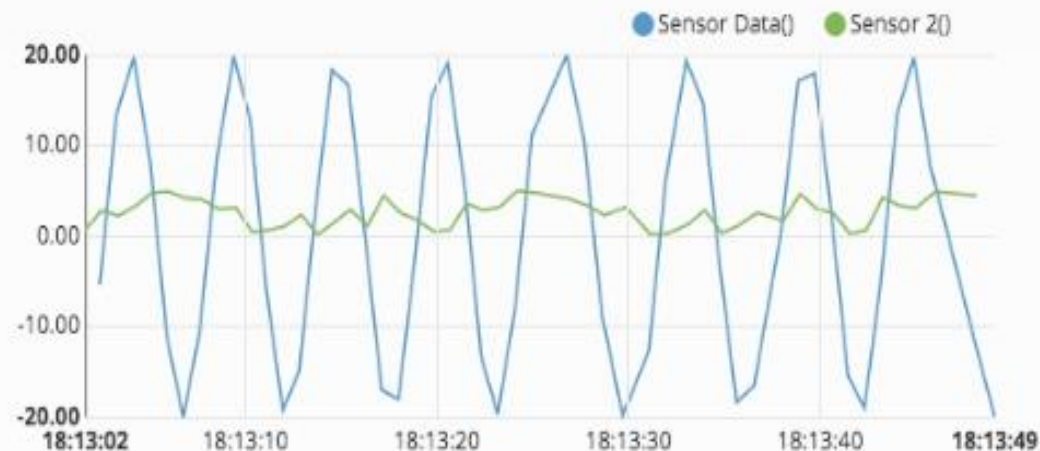
- De forma similar al método GET existe el método **POST(arg url)**, pero este nos sirve para el envío de datos hacia servidores que los van almacenar.
- Se va requerir la **URL** mas algún **recurso específico de almacenamiento receptáculo** o dispositivo ósea : [HTTP://URL/dispositivo](#)
- Lo anterior va depender de la documentación del API de la plataforma

# API WEB: **UBIDOTS**

- Ofrece servicio de alojamiento de datos directamente en una capa de presentación, ofreciendo la posibilidad de crear dashboards o tableros de visualización a tan solo unos clics.
- Funciona mediante la creación de dispositivos , cada uno con sus variables.
- Aquí su capa gratuita de [UBIDOTS STEM](#)

## My Dashboard

### Sensor Data



### Sensor A



### Sensor B

On average, Sensor B was

**1.25**

today

### Last values table

Variable name	Date	Last value
Sensor Data	March 29 2016 at 18:13:49	-19.99
Sensor 2	March 29 2016 at 18:13:49	4.129758521852203

### Sensor B



### Sensor A



### Sensor 2 ()



# ***DOCUMENTACIOM UBIDOTS***

***Protocolo HTTP y URL***

# ***DOCUMENTACIOM UBIDOTS***

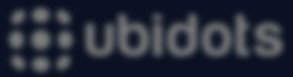
[https://docs.ubidots.com/v1.6/reference/send-data-to-a-device?\\_gl=1\\*pxemrx\\*\\_ga\\*MTQ4MTgyNjM2Ni4xNzMwMzM0OTA1\\*\\_ga\\_VE ME7QQ5EZ\\*MTczMTEwNTU2OC4yLjAuMTczMTEwNTU2OC4wLjAuMA..](https://docs.ubidots.com/v1.6/reference/send-data-to-a-device?_gl=1*pxemrx*_ga*MTQ4MTgyNjM2Ni4xNzMwMzM0OTA1*_ga_VE ME7QQ5EZ*MTczMTEwNTU2OC4yLjAuMTczMTEwNTU2OC4wLjAuMA..)



# *DOCUMENTACIOM METODO POST*

*ENVIAR DATA A LA VARIABLE DE UN DISPOSITVO*

***CREAR DISPOSITIVO -  
UBIDOTS***



Devices +

Data +

Users +

Devices

3 Devices

Search

<input type="checkbox"/>	Name	API label	Last active
<input type="checkbox"/>	 esp8266-ubidots	 esp8266-ubidots	an hour
<input type="checkbox"/>	 esp32-ubidots	 esp32-ubidots	9 days
<input type="checkbox"/>	 Demo Machine	 demo-machine	No last

DEVICES PER PAGE 10 ▾

1 - 3 of 3

Blank Device

×

## Create new device

Device name


Device label

CANCEL

BACK

NEXT

***CREAR VARIABLE  
UBIDOTS***



esp8266-ubidots

Description

Change description

Icon ⓘ

microchip

API label

esp8266-ubidots

ID


672e954f5f96f1000d2ea126

Token

.....



Tags

Add new tag



3 Variables


Search


<input type="checkbox"/>	Value	Name	Last updated ↓
<input type="checkbox"/>	 0	profundidadAguaPozo_cm	an hour ago
<input type="checkbox"/>	 7.82	distanciaHCSR04_cm	an hour ago

+

Raw variable

Synthetic variable





# Send data to a Device

Use this endpoint to send data to one or more variables of a Device

To send data to one or more variables of a Device please make a POST request to the following URL:

HTTP Method	URL
POST	<code>https://industrial.api.ubidots.com/api/v1.6/devices/&lt;device_label&gt;/</code>

Where `<device_label>` is a string with the label of the Device to which data will be sent to.

## Headers

The "X-Auth-Token" and "Content-Type" headers are required to your request:

Header	Value	Required?	Description
X-Auth-Token	Token	Yes	<a href="#">Authentication Token</a> of your account.
Content-Type	application/json	Yes	The type of data of the body.

## Body

An object containing keys as variable labels and Dots as values. The following are all valid payloads:

JUMP TO

CTRL-/

## INTRODUCTION

Getting started

Navigating our Data API Docs

## GENERAL FEATURES

Authentication

## HTTP

Overview

API URLs

▼ Devices

Send data to a Device

Send data to a Variable

Get last value of a variable

Get variable data

> Variables

> Data

> Examples

Response Codes

## MQTT

Overview

Broker URLs

### Send a value to multiple variables:

Update two variables and let Ubidots apply a timestamp equal to the reception time:

cURL 200 OK 400 Bad Request 401 Unauthorized

```
$ curl -X POST 'https://industrial.api.ubidots.com/api/v1.6/devices/<device_label>/' \
-H 'Content-Type: application/json' \
-H 'X-Auth-Token: oaXBo60DhIjPsusNRPUGIK4d72bc73' \
-d '{"temperature": 10, "humidty": 90}'
```

### Send a Dot to multiple variables:

Update multiple variables, including context and timestamp:

cURL 200 OK 400 Bad Request 401 Unauthorized

```
{
  "temperature": [{"status_code": 201}],
  "humidity": [{"status_code": 201}],
  "pressure": [{"status_code": 201}]
}
```

**EL SERVER RECIBIO LA DATA**

### Send location data:

Update a `position` variable so that the Device displays its location:

cURL 200 OK 400 Bad Request 401 Unauthorized

```
curl -X POST 'https://industrial.api.ubidots.com/api/v1.6/devices/<device_label>/' \
-H 'Content-Type: application/json' \
-H 'X-Auth-Token: oaXBo60DhIjPsusNRPUGIK4d72bc73' \
-d '{"position": {"lat": 6.2442, "lng": -75.5812}}'
```

# *EJEMPLO METODO POST*

*Añadir Headers(http.addHeader())*

*Añadir TOKEN ID*

*POST(payloadUPLINK)*



conexion\_WiFi\_NodeMCU\_POST\_ubidots.ino

```
52
53 //Hasta aqui se imprime
54 /*
55 Random Data: 13
56 JSON payload UPLINK [TX]:
57 {"randomData":13}
58 */
59
60 //INICIAR PROTOCLO HTTP
61 http.begin(client,"http://industrial.api.ubidots.com/api/v1.6/devices/esp8266-ubidots/");
62 http.addHeader("Content-Type", "application/json");
63 http.addHeader("X-Auth-Token", "BBUS-FCNCDjvXGpmc8nQ8vlgTqNgFvdcRqj");
64
65 //Enviar la payload de UPLINK
66 int httpCode=http.POST(payloadUPLINK);
67
68 if(httpCode>=0)
69 {
70   String payloadDOWNLINK=http.getString();
71   Serial.println("JSON payload DOWNLINK [RX]: ");
72   Serial.println(payloadDOWNLINK);
73 }
74
75 delay(15000);
```

Salida Monitor Serie x

Mensaje (Intro para mandar el mensaje de 'NodeMCU 1.0 (ESP-12E Module)' a 'COM8')

Sin ajuste de línea

9600 baud

```
18:30:45.955 -> Random Data: 67
18:30:45.985 -> JSON payload UPLINK [TX]:
18:30:46.031 -> {"randomData":67}
18:30:49.122 -> JSON payload DOWNLINK [RX]:
18:30:49.154 -> {"randomdata":[{"status_code":201}]}
18:31:04.142 ->
18:31:04.142 -> Random Data: 67
18:31:04.142 -> JSON payload UPLINK [TX]:
18:31:04.173 -> {"randomData":67}
18:31:07.365 -> JSON payload DOWNLINK [RX]:
18:31:07.397 -> {"randomdata":[{"status_code":201}]}
```

# *ESTADISITICA + REPORTE UBIDOTS*



92  
randomData

### Description

Change description

### Icon ?

cloud-upload-alt

### API label

randomdata



### ID

672e956bb097e1000da010a1



### Allowed

From: Min To: Max

### Tags

Add new tag

### Pre-processing ?

Slope 1

Offset 0

### Unit



Nov 08 2024 00:00 - Now

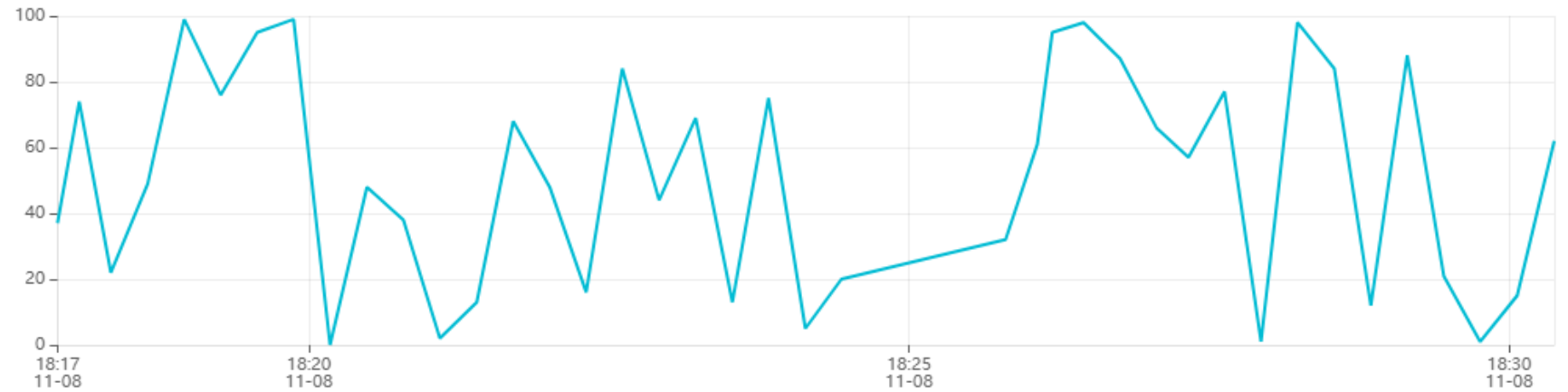


Aggregation

Raw



Sample period 5 minutes



Date

Value

Context



2024-11-08 18:30:22 -05:00

62

{}



2024-11-08 18:30:03 -05:00

15

{}



2024-11-08 18:29:45 -05:00

1

{}

