

# **APROXIMACIÓN** **AL MUNDO** **DEL TRABAJO**

**EVIDENCIA 2:** Escalado de Plataforma IoT desde la Capa Física de Transporte de Datos con Almacenamiento en MySQL.

**Integrantes:**

**Jimena Galleguillo**

**Franco Gonzalo Arce**

**Magali Maylen Bechis**

# Índice

- 1. Introducción**
- 2. Objetivo**
- 3. Requerimientos**
- 4. Paso a Paso**
  - 4.1 Configuración del Entorno de Desarrollo**
  - 4.2 Creación de la Aplicación Flask**
  - 4.3 Configuración de la Base de Datos en Clever Cloud**
  - 4.4 Conexión de Flask con la Base de Datos**
  - 4.5 Desarrollo del Proyecto en Wokwi**
  - 4.6 Integración del Hardware Simulado con la Aplicación Flask**
  - 4.7 Despliegue de la Aplicación en Render**
  - 4.8 Pruebas y Verificación**
- 5. Conclusiones**
- 6. Anexos**
  - 1. Video**
  - 2. Imágenes**

## Introducción

Este proyecto se centra en la implementación de un Medidor de Agua IoT utilizando Wokwi para la simulación del hardware, una base de datos MySQL en Clever Cloud, y una aplicación Flask en Python. Se busca medir el nivel de agua en un tanque, almacenar esta información en una base de datos y permitir su consulta a través de una interfaz web.

## Objetivo

- Desarrollar un sistema para medir y registrar el nivel de agua de un tanque.
- Almacenar los datos en una base de datos en la nube (Clever Cloud).
- Proporcionar una interfaz web en Wokwi, usando Flask para la visualización de la información.

## Requerimientos

- Hardware y Simulación: ESP32 simulado en Wokwi, sensor de nivel de agua.
- Software: Python 3.12, Flask, MySQL Connector, Unicorn, Render.
- Herramientas: MySQL Workbench, Visual Studio Code, Git.

## Paso a Paso

1. Configuramos el Entorno de Desarrollo  
Configurar un entorno virtual y actualizar las dependencias necesarias.
2. Creamos la Aplicación Flask  
Iniciar un proyecto Flask con MySQL Connector para gestionar los datos del tanque de agua.
3. Configuramos la Base de Datos en Clever Cloud  
Crear una instancia MySQL en Clever Cloud, configurar y crear la tabla para almacenar los datos.
4. Conectamos Flask con la Base de Datos  
Verificar la conexión con la base de datos y realizar pruebas de inserción y consulta de datos.
5. Desarrollamos el Proyecto en Wokwi  
Simular un ESP32 en Wokwi, programar el envío de datos del sensor de nivel de agua hacia la API Flask.

6. **Integramos el Hardware Simulado con la Aplicación Flask**

Probar la comunicación entre la simulación y la API para asegurar que los datos se almacenan correctamente.

7. **Despliegue de la Aplicación en Render**

Desplegar la aplicación Flask en Render, configurando las variables de entorno y asegurando el acceso a la base de datos.

8. **Pruebas y Verificación**

Ejecutar pruebas para validar la comunicación entre los diferentes componentes y monitorizar el rendimiento del sistema.

## **Conclusiones:**

El desarrollo de este proyecto nos permitió integrar diversas tecnologías para crear una solución funcional en la medición y monitoreo del nivel de agua en un tanque. A través de la simulación de hardware en Wokwi, la conexión de datos con una base de datos MySQL en Clever Cloud, y la implementación de una aplicación web en Flask, logramos unir el mundo físico con el digital, creando un sistema capaz de registrar y almacenar información de manera eficiente.

En este proceso, aprendimos a configurar entornos de desarrollo, a gestionar bases de datos en la nube, y a crear aplicaciones escalables para el despliegue en plataformas como Render. Nos enfrentamos a desafíos técnicos que nos impulsaron a profundizar en tecnologías como APIs, conexiones remotas a bases de datos y despliegue de aplicaciones web, consolidando nuestras habilidades y conocimientos en desarrollo de software.

Este proyecto no solo nos permitió adquirir destrezas prácticas, sino también nos mostró la importancia de la planificación, la paciencia y la atención al detalle en cada etapa del proceso.

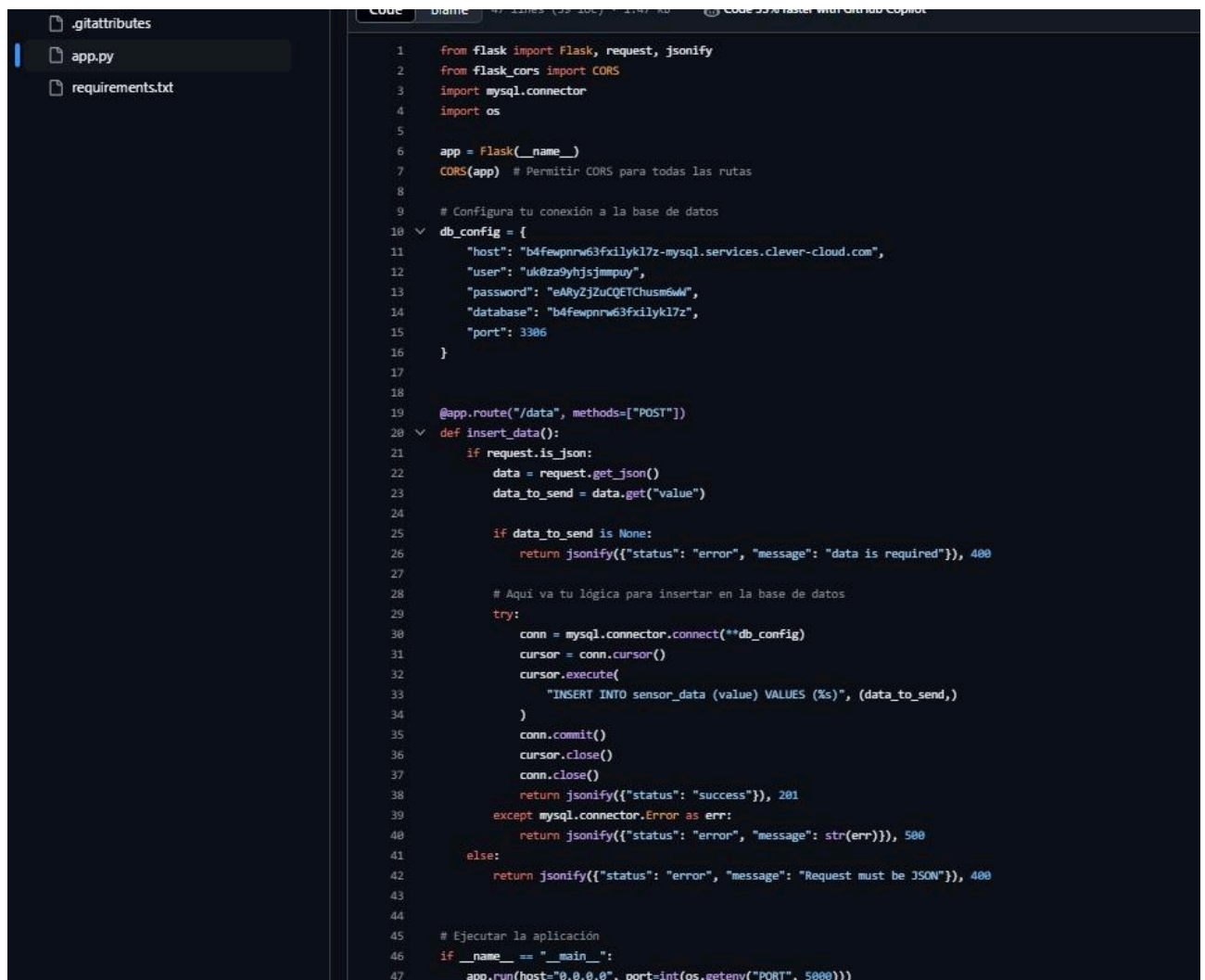
## Anexos:

1. Link al video de youtube: [Video](#)  
Link al código de Wokwi: [Wokwi](#)  
En el video mostramos el código de wokwi, y lo iniciamos.

## Imágenes de nuestros códigos

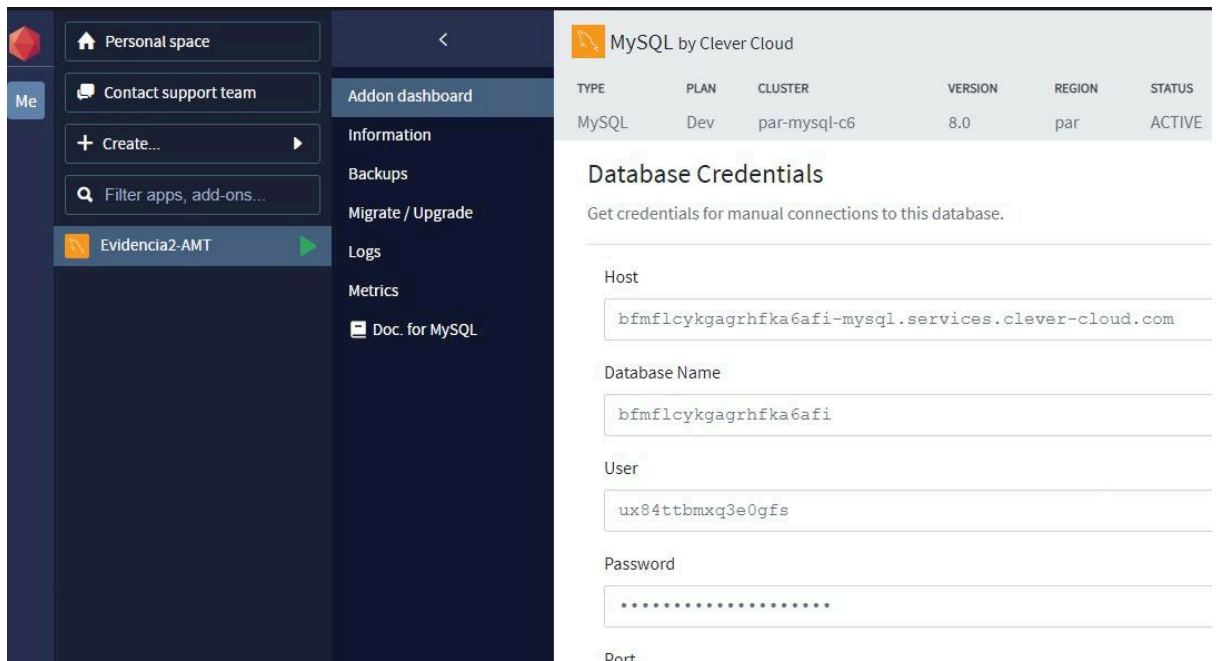
### 2. Imágenes.

- Código para la creación de Flask.



```
1  from flask import Flask, request, jsonify
2  from flask_cors import CORS
3  import mysql.connector
4  import os
5
6  app = Flask(__name__)
7  CORS(app) # Permitir CORS para todas las rutas
8
9  # Configura tu conexión a la base de datos
10 db_config = {
11     "host": "b4fewpnrw63fxilykl7z-mysql.services.clever-cloud.com",
12     "user": "uk0za9yhjsjmmuy",
13     "password": "eA8yZjZuCQETChusm6w",
14     "database": "b4fewpnrw63fxilykl7z",
15     "port": 3306
16 }
17
18
19 @app.route("/data", methods=["POST"])
20 def insert_data():
21     if request.is_json:
22         data = request.get_json()
23         data_to_send = data.get("value")
24
25         if data_to_send is None:
26             return jsonify({"status": "error", "message": "data is required"}), 400
27
28         # Aquí va tu lógica para insertar en la base de datos
29         try:
30             conn = mysql.connector.connect(**db_config)
31             cursor = conn.cursor()
32             cursor.execute(
33                 "INSERT INTO sensor_data (value) VALUES (%s)", (data_to_send,)
34             )
35             conn.commit()
36             cursor.close()
37             conn.close()
38             return jsonify({"status": "success"}), 201
39         except mysql.connector.Error as err:
40             return jsonify({"status": "error", "message": str(err)}), 500
41         else:
42             return jsonify({"status": "error", "message": "Request must be JSON"}), 400
43
44
45 # Ejecutar la aplicación
46 if __name__ == "__main__":
47     app.run(host="0.0.0.0", port=int(os.getenv("PORT", 5000)))
```

- Base de datos en clever cloud.



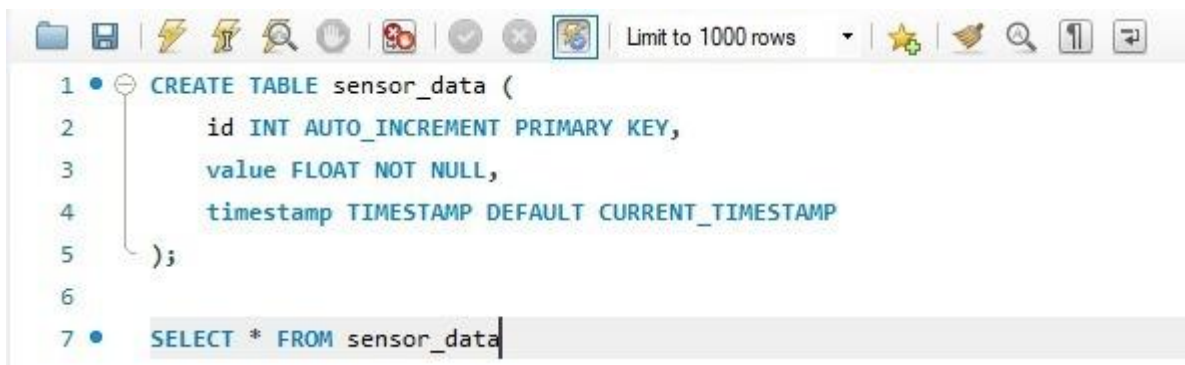
The screenshot shows the Clever Cloud MySQL dashboard. On the left is a sidebar with navigation options: Personal space, Contact support team, Create..., Filter apps, add-ons..., Evidencia2-AMT, Addon dashboard, Information, Backups, Migrate / Upgrade, Logs, Metrics, and Doc. for MySQL. The main panel displays the MySQL instance details:

TYPE	PLAN	CLUSTER	VERSION	REGION	STATUS
MySQL	Dev	par-mysql-c6	8.0	par	ACTIVE

Below the table, the 'Database Credentials' section provides the following information:

- Host: `bfmflcykgagrhfka6afi-mysql.services.clever-cloud.com`
- Database Name: `bfmflcykgagrhfka6afi`
- User: `ux84ttbmxq3e0gfs`
- Password: (masked with dots)
- Port: (blank)

- Tabla de la base de datos.



The screenshot shows a SQL editor interface with a toolbar at the top. The toolbar includes icons for file operations, execution, and a dropdown menu set to 'Limit to 1000 rows'. The SQL code is as follows:

```

1 • CREATE TABLE sensor_data (
2     id INT AUTO_INCREMENT PRIMARY KEY,
3     value FLOAT NOT NULL,
4     timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
5 );
6
7 • SELECT * FROM sensor_data
  
```

- Código de wokwi.

```
# En primer lugar mediremos el tiempo de respuesta del pulso
duracion = random.uniform(100, 30000) # simulacion de valores aleatorios

# Realizamos la conversion de tiempo a centimetros
# Formula: duracion * Vel. del sonido (0.0344) / 2
distancia = duracion * 0.0344 / 2
return distancia

# Configuración inicial
def setup():
    connect_wifi(SSID, PASSWORD)

def loop():
    # Medir la distancia
    distancia = medir_distancia()
    print("Distancia medida desde el sensor hasta la superficie del agua: " + str(distancia) + " cm")

    # Calcular el nivel de agua
    nivel_agua = ALTURA_TANQUE - distancia
    print("Nivel de agua calculado: " + str(nivel_agua) + " cm")

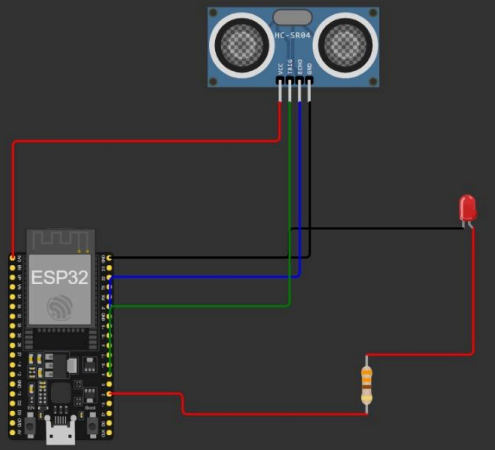
    # Controlar el LED según el nivel de agua
    if nivel_agua < UMBRAL_NIVEL_AGUA:
        print("Alerta!! El nivel de agua es demasiado bajo!")
        for i in range(5):
            led.value(1)
            time.sleep_us(2)
            led.value(0)
            time.sleep_us(10)
        else:
            led.value(0)
            print("Nivel de agua normal.")
        print(nivel_agua)
        send_data_to_api(nivel_agua)

except Exception as e:
    print("Error al medir la distancia: ", str(e))

time.sleep(5)
```

- Código con el hardware.

```
7
8 ### CONFIGURACION WIFI Y CONEXION A API
9
10 # Configuración de WIFI
11 SSID = "Wokwi-GUEST"
12 PASSWORD = ""
13
14 # Dirección de la API en el servidor
15 API_URL = "https://evidencia-2-ant-lspc.onrender.com/data" # URL de la API
16
17
18 # Función para conectarse al WiFi
19 def connect_wifi(ssid, password):
20     wlan = network.WLAN(network.STA_IF)
21     wlan.active(True)
22     wlan.connect(ssid, password)
23
24     while not wlan.isconnected():
25         print('Conectando a WiFi...')
26         time.sleep(1)
27
28     print('Conectado a WiFi. IP:', wlan.ifconfig()[0])
29
30 # Función para enviar datos a la API
31 def send_data_to_api(value):
32     data = {'value': value,} # data = {'nombreClaveJSON': value,}
33     headers = {'Content-Type': 'application/json'}
34
35     try:
36         json_data = json.dumps(data)
37         response = urequests.post(API_URL, data=json_data, headers=headers)
38         if response.status_code == 201:
39             print('Datos enviados correctamente!')
40         else:
41             print('Error al enviar los datos:', response.text)
42         response.close()
43     except Exception as e:
44         print('Error en la solicitud:', e)
45
46
```



- **Render.**

Render

Dashboard

Blueprints

Env Groups

WEB SERVICE

Evidencia2-AMT

Python 3

Free

Upgrade your instance →

Franco-Arce / Evidencia2-AMT

main

https://evidencia2-amt.onrender.com

Events

Logs

Disks

Environment

Shell

Previews

Jobs

Metrics

Scaling

Settings

ⓘ Your free instance will spin down with inactivity, which can delay requests by 50 seconds or more.

October 3, 2024 at 4:10 PM

Lve

1d9c266

Add files via upload

All logs

Q Search

Oct 3, 4:09 PM

Oct 3 04:13:10 PM

Docs on specifying a bun version: <https://render.com/docs/bun-version>

Oct 3 04:13:30 PM

Using Bun version 1.1.0 (default)

Oct 3 04:13:31 PM

Docs on specifying a bun version: <https://render.com/docs/bun-version>

Oct 3 04:13:46 PM

Running 'gunicorn -w 4 -b 0.0.0.0:5000 app:app'

Oct 3 04:13:49 PM

[2024-10-03 19:13:49 +0000] [93] [INFO] Starting gunicorn 23.0.0

Oct 3 04:13:49 PM

[2024-10-03 19:13:49 +0000] [93] [INFO] Listening at: <http://0.0.0.0:5000> (93)

Oct 3 04:13:49 PM

[2024-10-03 19:13:49 +0000] [93] [INFO] Using worker: sync

Oct 3 04:13:49 PM

[2024-10-03 19:13:49 +0000] [94] [INFO] Booting worker with pid: 94

Oct 3 04:13:49 PM

Detected a new open port HTTP:5000

Oct 3 04:13:49 PM

[2024-10-03 19:13:49 +0000] [95] [INFO] Booting worker with pid: 95

Oct 3 04:13:49 PM

[2024-10-03 19:13:49 +0000] [96] [INFO] Booting worker with pid: 96

Oct 3 04:13:49 PM

[2024-10-03 19:13:49 +0000] [97] [INFO] Booting worker with pid: 97

Oct 3 04:13:50 PM

127.0.0.1 - - [03/Oct/2024:19:13:50 +0000] "HEAD / HTTP/1.1" 404 0 "-" "Go-http-client/1.1"

Oct 3 04:13:50 PM

Your service is live 🎉

Oct 3 04:13:50 PM

127.0.0.1 - - [03/Oct/2024:19:13:50 +0000] "GET / HTTP/1.1" 404 207 "-" "Go-http-client/2.0"

Oct 3 04:14:50 PM

Detected a new open port HTTP:5000

- **Pruebas y verificación.**

	id	value	timestamp
	1	42	2024-10-03 16:31:51
	2	42	2024-10-03 16:40:36
	3	42	2024-10-03 16:56:26
	4	42	2024-10-03 17:25:27
	5	42	2024-10-03 17:29:21
	6	141.729	2024-10-03 17:52:48
	7	479.194	2024-10-03 17:53:06
	8	524.188	2024-10-03 20:47:09
	9	266.4	2024-10-03 21:11:19
	10	357.065	2024-10-03 21:11:31
	11	533.744	2024-10-03 21:23:33
	12	133.328	2024-10-03 21:23:45
	13	586.714	2024-10-03 21:23:58
	14	462.605	2024-10-03 22:24:17
	15	421.231	2024-10-03 22:24:17
	16	250.761	2024-10-03 23:06:21
	17	333.235	2024-10-03 23:06:32
	18	384.548	2024-10-03 23:06:45