# 💧Secure IoT Environmental Monitoring System

This project implements a complete, secure Internet of Things (IoT) pipeline for real-time environmental data collection and analysis. It uses an **ESP32** microcontroller for sensing and a **FastAPI** service for cloud ingestion into a **Supabase** database.

## 🚀 Architecture and Data Flow

The system is designed for reliability and security, utilizing industry-standard protocols for end-to-end data transfer.

### Data Path:

1. **ESP32:** Reads sensor data, formats it into a JSON payload.
2. **MQTT (TLS/SSL):** Publishes the JSON payload securely to the cloud broker over Port 8883.
3. **HiveMQ Cloud:** Acts as the private, managed MQTT Broker.
4. **FastAPI Ingestion Service:** Subscribes to the topic via **Secure WebSocket (WSS/8884)**, handles authentication, and processes the incoming data.
5. **Supabase:** The final destination where data is stored in a PostgreSQL table for analysis and visualization.

## 🛠️ Components

### Hardware & Sensing

|  |  |
| --- | --- |
| **Component** | **Function** |
| **ESP32** | Main processing unit; runs embedded C++ for connectivity and control. |
| **DHT11** | Measures real-time **Temperature** and **Humidity**. |
| **Rain Sensor** | Detects and reports **Rain Status** (e.g., wet/dry). |
| **Water Level Sensor** | Measures and reports **Water Level**. |
| **Servo Motor** | Controlled actuator for physical interaction (e.g., opening a vent). |
| **LCD/Keypad** | Local interface for display and input. |

### Software & Cloud Services

|  |  |  |
| --- | --- | --- |
| **Component** | **Role** | **Protocol/Language** |
| **HiveMQ Cloud** | Managed MQTT Broker for messaging between device and server. | MQTT (TLS/WSS) |
| **FastAPI** | Python Web API framework used for the Subscriber and Ingestion Service. | Python |
| **Supabase** | Backend-as-a-Service (PostgreSQL database) for persistent storage. | SQL, REST |
| **paho-mqtt** | Python library used by FastAPI to handle MQTT connection. | Python |

## 💾 Supabase Database Schema

The ingested data is stored in the sensor\_readings table with the following structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Source** | **Description** |
| id | uuid | Auto-generated | Primary key. |
| created\_at | timestampz | Auto-generated | Time of record insertion. |
| temperature | float | DHT11 | Ambient temperature reading. |
| humidity | float | DHT11 | Ambient humidity reading. |
| soil\_moisture | float | (Assumed Sensor) | Reading for soil moisture levels. |
| rain\_status | boolean/int | Rain Sensor | Current rain detection status. |
| water\_level | float | Water Level Sensor | Raw water level reading. |
| servo\_angle | int | ESP32 | Angle of the controlled servo motor. |

## ⚙️ Setup and Deployment

### Prerequisites

1. **Supabase Account:** Database URL and Service Role Key.
2. **HiveMQ Cloud Account:** Broker URL, Port 8883, WSS Port 8884, MQTT Username, and Password.
3. **Python Environment:** Python 3.9+ with fastapi, uvicorn, paho-mqtt, supabase-py, and python-dotenv.
4. **Arduino/PlatformIO:** Environment for compiling and flashing the ESP32 code.

### 1. Configure Environment Variables

Create a .env file in the root directory of the FastAPI project and populate it with your private cloud credentials:

# Supabase  
SUPABASE\_URL="[https://your-project-ref.supabase.co](https://your-project-ref.supabase.co)"  
SUPABASE\_SERVICE\_KEY="your-supabase-key"  
  
# HiveMQ Cloud (WSS Path added directly to Broker URL for compatibility)  
MQTT\_BROKER="<YOUR\_HIVE\_MQ\_CLOUD\_URL>/mqtt"   
MQTT\_PORT="8884"   
MQTT\_USER="your\_mqtt\_username"   
MQTT\_PASSWORD="your\_mqtt\_password"  
MQTT\_TOPIC="user/7/rain\_data"

### 2. Run the Ingestion Service

Start the FastAPI application (the MQTT subscriber):

uvicorn data\_ingestion:app --reload

The service will connect securely to HiveMQ and immediately begin listening for data, pushing any received JSON payloads into Supabase.

### 3. Deploy ESP32 Code

Ensure the ESP32 code is configured with the same MQTT\_BROKER (on Port 8883, using WiFiClientSecure) and the same MQTT\_TOPIC before flashing the device.