

Implementación de Broker MQTT en la Nube con EMQX para Proyecto Hidropónico

Miguel Angel Ortiz Escobar - 20222208138

Sara Sofia Gonzalez Gomez - 20222208303

Introducción

Para habilitar la comunicación en tiempo real entre los sensores del sistema hidropónico y la plataforma de monitoreo, se integró un broker MQTT en la nube utilizando **EMQX Cloud**, una solución escalable y distribuida para la mensajería IoT. Esta integración permite transmitir datos de sensores y estados del sistema de forma eficiente y segura, sin depender de infraestructura física local.

¿Qué es EMQX?

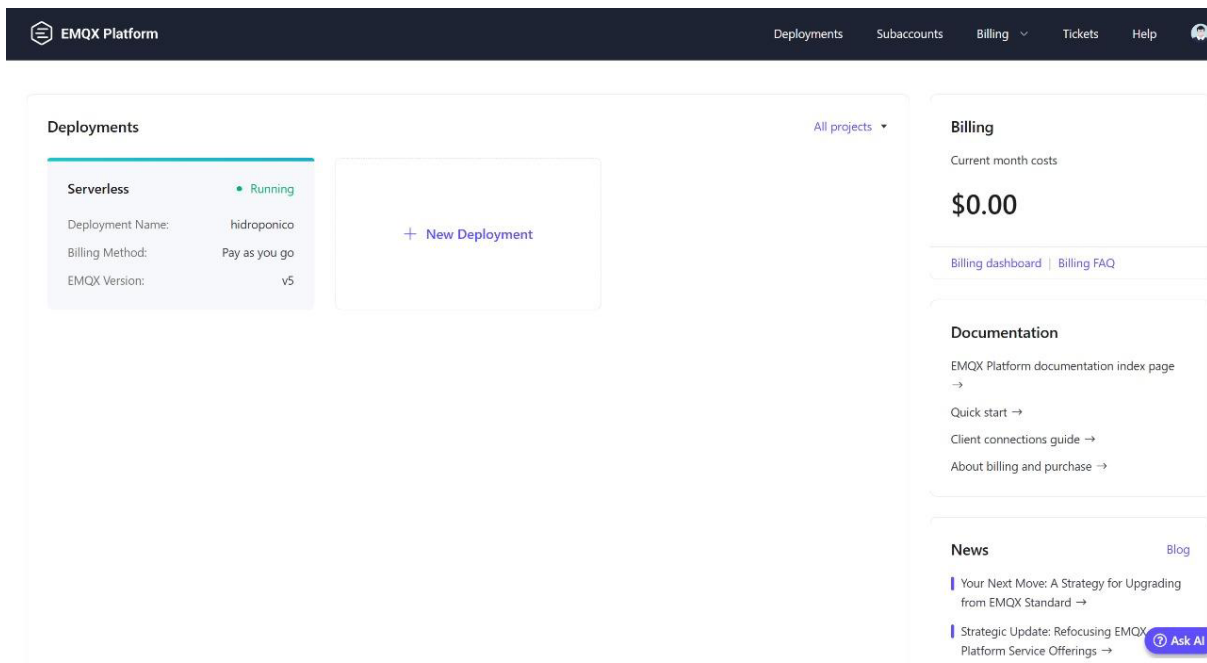
EMQX (Erlang/Enterprise MQTT Broker) es una plataforma de mensajería **basada en el protocolo MQTT**, diseñada para aplicaciones de IoT. Se caracteriza por su alta disponibilidad, escalabilidad y rendimiento, lo cual la convierte en una solución ideal para sistemas que requieren comunicación en tiempo real entre múltiples dispositivos.

Actúa como **broker MQTT**, gestionando la comunicación entre los **publicadores** (por ejemplo, sensores de humedad o temperatura del sistema hidropónico) y los **suscriptores** (por ejemplo, aplicaciones web o móviles que reciben y visualizan los datos).

Procedimiento de Implementación

1. Creación del Entorno en EMQX Cloud

Se configuró un entorno serverless en EMQX Cloud para alojar el broker MQTT remoto. Este entorno proporciona una URL pública y los puertos necesarios para realizar conexiones seguras mediante los protocolos **MQTT**, **MQTTS** y **WebSocket**.




2. Configuración del Broker

Desde el panel de administración de EMQX, se accedió a:


- **Información del entorno de despliegue**, incluyendo:
 - Dirección del host.
 - Puertos disponibles para cada protocolo.
 - Estadísticas en tiempo real sobre las sesiones activas.
- **Sesiones y conexiones activas**, que muestran los dispositivos conectados en calidad de publicadores o suscriptores.
- **Usuarios (clientes MQTT)**, los cuales fueron creados manualmente con un username y password personalizados. Estos datos se utilizan en el código del cliente para establecer la autenticación con el broker.

MQTT Connection Information

Address: i12b8e78.ala.us-east-1.emqxsl.com 

MQTT over TLS/SSL Port: 8883

WebSocket over TLS/SSL Port: 8084

CA Certificate:  CA Certificate Expiration: 2031.11.09

EMQX Platform

DeploymentsSubaccountsBillingTicketsHelp

All Deployments / Default Project / hidroponico

Overview

Access Control

Monitor

Metrics

Clients

Subscriptions

Retained messages

Alerts

Data Integration

Logs

Online Test

Connection Guide

Settings

Clients

🔍

🔔

⌵

🔄

Client ID	Username	IP Address	Keepalive	Protocol Type	Clean Start	Expiry Interval (s)	Status	Connect
No Data								

👤

🔍 Ask AI

EMQX Platform

DeploymentsSubaccountsBillingTicketsHelp

All Deployments / Default Project / hidroponico

Overview

Access Control

Monitor

Metrics

Clients

Subscriptions

Retained messages

Alerts

Data Integration

Logs

Online Test

Connection Guide

Settings

Subscriptions

🔍

🔔

⌵

Client ID	Topic	QoS	No Local	Retain as Published	Retain Handling
No Data					

👤

🔍 Ask AI

3. Conexión desde el Cliente MQTT Local

En el código del dispositivo local (ESP32 y Raspberry Pi), se estableció la conexión con el broker remoto. Para ello se reemplazaron las variables de conexión locales por los datos proporcionados por EMQX:

```
broker_url = "broker.emqx.io" # Dirección proporcionada por EMQX
port = 1883                # Puerto MQTT estándar
username = "usuario_configurado"
password = "contraseña_configurada"
```

Con esta configuración, el cliente puede:

- **Publicar datos:** Enviar la información de los sensores al broker (por ejemplo, /hidropónico/datos).
- **Suscribirse a tópicos:** Escuchar comandos o eventos que provengan desde otros servicios conectados.

```

EMQX_BROKER = os.getenv("EMQX_BROKER")
EMQX_PORT = int(os.getenv("EMQX_PORT"))
EMQX_TOPIC_PREFIX = os.getenv("EMQX_TOPIC_PREFIX")
EMQX_USERNAME = os.getenv("EMQX_USERNAME")
EMQX_PASSWORD = os.getenv("EMQX_PASSWORD")
EMQX_CLIENT_ID = os.getenv("EMQX_CLIENT_ID")
EMQX_CA_CERT = os.getenv("EMQX_CA_CERT")

def enviar_a_emqx(payload):
    try:
        client = mqtt.Client(client_id=EMQX_CLIENT_ID)
        client.username_pw_set(EMQX_USERNAME, EMQX_PASSWORD)
        client.tls_set(ca_certs=EMQX_CA_CERT, tls_version=ssl.PROTOCOL_TLSv1_2)
        client.connect(EMQX_BROKER, EMQX_PORT)
        client.loop_start()

        client.publish(EMQX_TOPIC_PREFIX, json.dumps(payload, ensure_ascii=False))
        print(f"📡 Enviado a EMQX: {EMQX_TOPIC_PREFIX}")

        time.sleep(1)
        client.loop_stop()
        client.disconnect()

    except Exception as e:
        print("⚠️ Error al enviar a EMQX:", str(e))

```

Código implementado en el MQTT local que se estaba manejando.

El siguiente código fue desarrollado en **Python** y desplegado en una **Raspberry Pi** para actuar como **punto (bridge)** entre el entorno local (sensores conectados vía ESP32 y Arduino) y la **nube (EMQX Cloud)**. Además, permite el reenvío de datos hacia un servidor Flask y el almacenamiento en una base de datos MongoDB.

Módulos utilizados

```

import paho.mqtt.client as mqtt

import ssl, json, time, os

from dotenv import load_dotenv

import requests

from mongo_utils import guardar_dato, sincronizar_periodicamente

```

- `paho.mqtt.client`: Librería cliente para conectarse y comunicarse mediante MQTT.
- `ssl`: Configura la conexión segura TLS para el broker EMQX.
- `dotenv`: Carga variables de entorno desde un archivo `.env`.
- `requests`: Permite enviar datos al servidor Flask mediante HTTP.
- `mongo_utils`: Módulo personalizado que gestiona la escritura en MongoDB y sincronizaciones periódicas.

Configuración mediante Variables de Entorno

El código utiliza `.env` para separar la configuración sensible (usuario, contraseña, puertos, certificados) del código fuente:

`EMQX_BROKER=i12b8e78.ala.us-east-1.emqxsl.com`

`EMQX_PORT=8883`

`EMQX_TOPIC_PREFIX=hidroponico/datos`

`EMQX_USERNAME=mqttRP`

`EMQX_PASSWORD=1234`

`EMQX_CLIENT_ID=raspberry-bridge`

`EMQX_CA_CERT=./emqxsl-ca.crt`

Esto permite **modularidad y seguridad**, especialmente en ambientes de producción.

Función `enviar_a_emqx(payload)`

Esta función publica datos en un tópico MQTT remoto, alojado en **EMQX Cloud**. Se establece una conexión segura mediante **TLS (puerto 8883)**.

```
client = mqtt.Client(client_id=EMQX_CLIENT_ID)
```

```
client.username_pw_set(EMQX_USERNAME, EMQX_PASSWORD)
```

```
client.tls_set(ca_certs=EMQX_CA_CERT, tls_version=ssl.PROTOCOL_TLSv1_2)
```

Una vez conectada, la función publica el payload serializado como JSON en el tópico configurado:

```
client.publish(EMQX_TOPIC_PREFIX, json.dumps(payload, ensure_ascii=False))
```

Suscripción al Broker Local

La Raspberry Pi también actúa como cliente MQTT conectado a un broker local (por ejemplo, en ESP32):

```
local_client = mqtt.Client()
```

```
local_client.on_message = on_local_message
```

```
local_client.connect(LOCAL_MQTT_HOST, LOCAL_MQTT_PORT, 60)
```

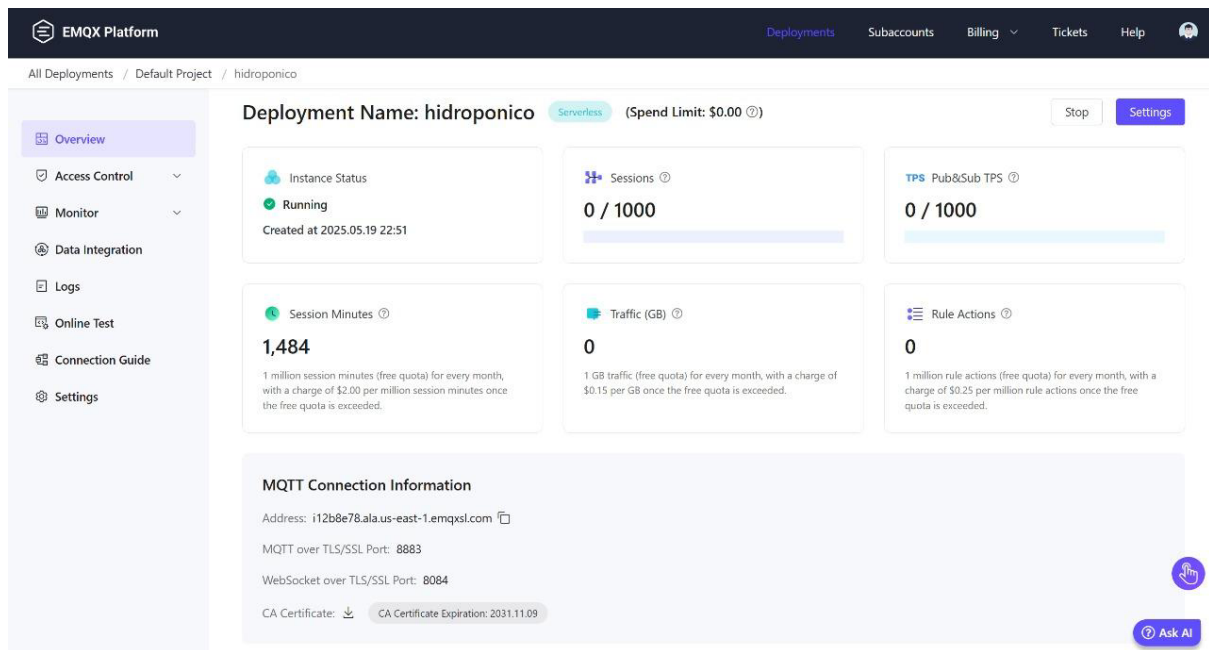
```
local_client.subscribe(LOCAL_TOPIC)
```

Cada vez que se recibe un mensaje, se dispara la función `on_local_message`, que llama a `procesar_payload`.

4. Pruebas de Comunicación y Monitorización

Desde el panel de EMQX se verificó:

- El establecimiento correcto de las sesiones cliente-broker.
- La correcta publicación y suscripción a los tópicos definidos.
- El tráfico de mensajes en tiempo real para validar que la integración entre el sistema hidropónico local y la nube funciona correctamente.



La integración de **EMQX Cloud** como broker MQTT remoto trajo múltiples beneficios al desarrollo del sistema hidropónico, tanto en términos de rendimiento como de escalabilidad:

1. Alta disponibilidad y escalabilidad

EMQX permite manejar múltiples conexiones simultáneas de dispositivos IoT, sin necesidad de infraestructura física local. Esto facilita la expansión del sistema (más sensores o zonas) sin modificar la arquitectura base.

2. Comunicación en tiempo real y baja latencia

El uso del protocolo MQTT permite una comunicación eficiente y ligera entre dispositivos, ideal para el envío periódico de datos como temperatura, humedad y otros parámetros agrícolas.

3. Conexión segura mediante TLS/SSL

Se configuró la conexión con cifrado TLS (puerto 8883), lo cual garantiza que los datos transmitidos entre los sensores y la nube estén protegidos frente a ataques o interceptaciones.

4. Monitorización centralizada desde la nube

EMQX Cloud proporciona una interfaz gráfica desde donde se pueden observar las conexiones activas, los tópicos utilizados y la cantidad de mensajes transmitidos, lo cual permite verificar en tiempo real el correcto funcionamiento del sistema.

5. Integración sencilla con otros servicios

Al estar en la nube, el broker puede integrarse fácilmente con servidores

web, APIs o servicios externos como bases de datos, visualizadores o sistemas de inteligencia artificial.

Conclusión

La implementación de un broker MQTT remoto utilizando **EMQX Cloud** representó un componente fundamental para lograr una arquitectura distribuida, segura y eficiente en el proyecto hidropónico. Este enfoque permitió separar la capa de adquisición de datos (ESP32, Raspberry Pi) de la capa de procesamiento y visualización (servidor Flask y MongoDB), lo que mejora la modularidad y mantenibilidad del sistema.

Además, al evitar depender exclusivamente de conexiones locales, se facilita el monitoreo remoto del estado de las plantas, lo que representa un avance significativo hacia la automatización de cultivos inteligentes. Esta solución no solo es viable para prototipos, sino que también es escalable para implementaciones reales en campo.

Aprendizajes

Durante el desarrollo de esta parte del proyecto, se adquirieron conocimientos técnicos valiosos, entre los cuales destacan:

- Comprensión del protocolo **MQTT** y su utilidad en sistemas IoT distribuidos.
- Configuración de un **entorno serverless** en la nube usando EMQX Cloud.
- Establecimiento de conexiones seguras mediante **TLS/SSL** y autenticación por usuario y contraseña.
- Uso de **variables de entorno (.env)** para mejorar la seguridad y la portabilidad del código.
- Integración de múltiples tecnologías en un flujo funcional: **ESP32 (datos), Raspberry Pi (puente MQTT), Python, Flask, MongoDB y EMQX.**

Asimismo, se enfrentaron y resolvieron desafíos relacionados con la sincronización entre clientes, la gestión de errores en la red y la depuración de conexiones MQTT, lo que permitió fortalecer las habilidades de desarrollo de soluciones IoT en la nube.