

TP3: comunicación Modbus, OPC UA y MQtt

Protocolos de comunicación industriales



28 de julio de 2025

Lautaro kuhn, Esteban porporato

Profesores: Martín Pico, Milton Pozzo

# Índice:

Contenido

[Índice: 1](#_Toc204637605)

[Introducción 2](#_Toc204637606)

[Objetivos del Proyecto 3](#_Toc204637607)

[Descripción General del Sistema 3](#_Toc204637608)

[Estructura de la Comunicación y Diagrama del Sistema 4](#_Toc204637609)

[Componentes del Sistema 4](#_Toc204637610)

[Descripción del Flujo de Datos 6](#_Toc204637611)

[Descripción de Archivos y Scripts 7](#_Toc204637612)

[1. S\_OPC\_MQTT.py 7](#_Toc204637613)

[2. S\_MODBUS\_OPCUA.py 8](#_Toc204637614)

[3. interface.html 9](#_Toc204637615)

[4. Codigo\_Def.ino 9](#_Toc204637616)

[Uso de EMQX como Broker MQTT 10](#_Toc204637617)

[Precaución: 11](#_Toc204637618)

[Comunicación entre scripts: 11](#_Toc204637619)

[Comunicación entre OPC UA y MQTT: 11](#_Toc204637620)

[Comunicación entre OPC UA y Arduino (MODBUS): 12](#_Toc204637621)

[Conclusión 13](#_Toc204637622)

[Bibliografía 14](#_Toc204637623)

# Introducción

Este informe detalla el desarrollo y la implementación de un sistema de comunicación de datos basado en los protocolos **Modbus**, **OPC UA** (Unified Architecture) y **MQTT** (Message Queuing Telemetry Transport), integrados en una arquitectura distribuida basada en **Arduino, Raspberry Pi, PC y una interfaz web**, con el objetivo de lograr un sistema robusto de monitoreo y control remoto en tiempo real.

La integración de Modbus, OPC UA y MQTT en sistemas embebidos es especialmente relevante en aplicaciones industriales y de Internet de las Cosas (IoT), donde el intercambio de datos en tiempo real y la seguridad de la transmisión son primordiales.

La competencia específica que se aborda en este proyecto consiste en planificar, dirigir y ejecutar proyectos de comunicación de datos en sistemas de red complejos, desde la adquisición de datos, transporte de los mismos por la red, visualización de estos y toma de decisiones con impacto real.

La infraestructura diseñada simula una planta industrial moderna, donde se adquieren variables analógicas y digitales, se procesan y publican desde dispositivos intermedios hacia una WebApp que permite a los usuarios supervisar y operar el sistema de forma intuitiva.

La implementación se centra en la **fiabilidad de las comunicaciones**, la modularidad de los bloques funcionales y la capacidad de escalar el sistema sin comprometer su estabilidad, logrando una solución aplicable en entornos reales de IoT e Industria 4.0.

# Objetivos del Proyecto

El objetivo principal de este trabajo es integrar los protocolos Modbus, OPC UA y MQTT en el desarrollo de un sistema de comunicación de datos desde sensores físicos hasta su visualización en una interfaz web para el usuario final.

**Objetivos específicos**

* Crear, configurar y poner en funcionamiento los elementos necesarios para el desempeño correcto de la comunicación (sensores, dispositivos, scripts, conexiones y protocolos).
* Establecer una estructura de tópicos adecuada para la transmisión y almacenamiento ordenado de datos en el sistema MQTT.
* Conocer y evaluar las ventajas y desventajas de distintos niveles de seguridad, como conexiones anónimas vs. autenticadas, y protocolos TCP vs. WebSocket.
* Configurar procedimientos automáticos de verificación del estado de la comunicación en sus distintas etapas (MODBUS, OPC UA y MQTT), con indicadores en la interfaz para el monitoreo del sistema.

# Descripción General del Sistema

El presente sistema tiene como finalidad integrar de manera eficiente los protocolos Modbus RTU, OPC UA y MQTT en un entorno distribuido de monitoreo y control. La implementación busca simular una planta industrial moderna, en la que se utilizan estándares de comunicación robustos para garantizar la interoperabilidad entre dispositivos físicos y plataformas de supervisión.

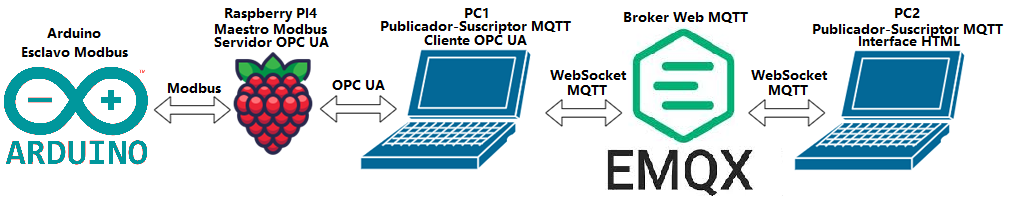
En esta arquitectura, un Arduino Mega actúa como dispositivo esclavo Modbus RTU, encargado de adquirir datos analógicos y digitales desde sensores, así como de gestionar salidas hacia actuadores. Estos datos son recibidos por una Raspberry Pi 4, que opera simultáneamente como maestro Modbus RTU y servidor OPC UA, permitiendo exponer la información a clientes de manera estructurada y segura.

Para facilitar la interacción remota, se incorporó una interfaz web que opera en tiempo real mediante el uso de WebSockets y mensajería ligera a través del protocolo MQTT. De este modo, los usuarios pueden visualizar variables, monitorear estados y enviar comandos sin necesidad de recargar la página, mejorando la experiencia y la eficiencia operativa.

El sistema fue diseñado con un enfoque modular, permitiendo escalar en cantidad de sensores, nodos o clientes sin comprometer la arquitectura existente. La separación de responsabilidades entre captura de datos, procesamiento, publicación y visualización asegura una comunicación fluida entre los distintos niveles de la red.

# Estructura de la Comunicación y Diagrama del Sistema

La comunicación en el sistema se lleva a cabo a través de tres capas de protocolos diferentes, en donde Modbus maneja la comunicación Maestro-Esclavo, OPC UA maneja la conexión y transmisión de datos entre el servidor y el cliente, mientras que MQTT con WebSocket facilita la distribución de estos datos desde el cliente publicador hacia otros dispositivos suscriptores y publicadores que cuenten con la interface. A continuación, se presenta el diagrama simplificado de la conexión del sistema:



**Figura 1**: Diagrama de conexión del sistema.

# Componentes del Sistema

**Arduino (Esclavo MODBUS RTU):**

Se utiliza como dispositivo de campo, conectado a sensores analógicos y digitales, y también a salidas digitales. Está programado para responder a peticiones MODBUS desde un maestro a través de UART, y simula el comportamiento de un equipo industrial real con registros de entrada y salida. Cuenta con 8 registros definidos funcionalidades de cálculo de CRC.

**Raspberry Pi 4 (Maestro MODBUS + Servidor OPC UA):**

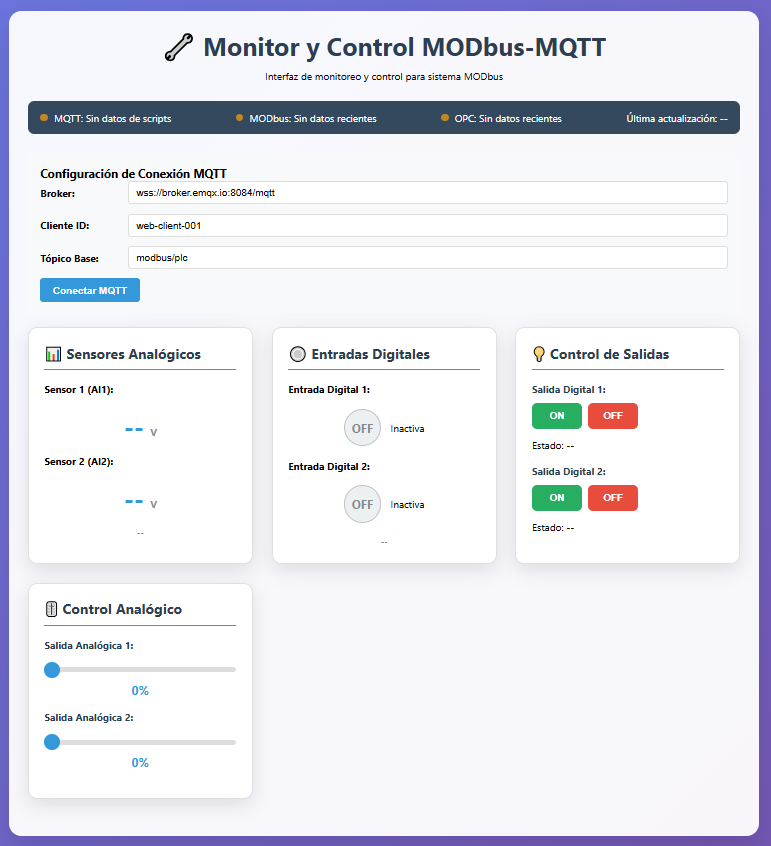
La Raspberry actúa como enlace entre el mundo físico y los niveles de control y monitoreo. Su rol es doble: su primer rol es como maestro MODBUS, encargado de interrogar periódicamente al Arduino por sus registros, cuanta con función de repregunta y reescritura en caso de que el CRC no de correctamente y a su vez cuanta con función de reconexión automática en caso de que se interrumpa la comunicación serie Modbus lo que es importante a la hora notificar el estado de conexión Modbus en la interface grafica que se creó; segundo rol, como servidor OPC UA, exponiendo esa información a clientes de más alto nivel mediante una API estructurada y recibiendo datos de clientes específicos.

**PC Cliente (Cliente OPC UA + Publicador MQTT):**

Desde una computadora personal se ejecutan scripts que cumplen funciones de intermediación: reciben los datos desde OPC UA y los traducen a tópicos MQTT con WebSocket, que luego son consumidos por una interfaz web. Además, también procesan los comandos que llegan desde la web y los aplican sobre el servidor OPC UA, cerrando el bucle de control.

**WebApp (Cliente MQTT + Interfaz gráfica):**

La interfaz web desarrollada en HTML y JavaScript permite al usuario visualizar en tiempo real el estado de sensores y actuadores, al mismo tiempo que ofrece herramientas de control. Todo esto ocurre mediante la suscripción y publicación a un broker MQTT que se encuentra en la nube, que actúa como canal de comunicación entre la web y el sistema físico.



**Figura 2:** Interface HTML para control y monitoreo del sistema.

# Descripción del Flujo de Datos

**Arduino (Esclavo MODBUS):**

El Arduino se encarga de interactuar con sensores como DHT (para temperatura y humedad), una fotorresistencia o un potenciómetro, y salidas digitales como LEDs. La lógica del programa define ocho registros MODBUS que almacenan tanto datos de sensores como el estado de las salidas y de posibles entradas digitales. No implementa librerías externas, sino que se crearon funciones que emulan funciones de dispositivos industriales, que le permite comportarse como esclavo conforme al estándar del protocolo MODBUS RTU. Cada trama enviada o recibida incluye el cálculo de CRC para garantizar la integridad de los datos transmitidos, lo que es vital en ambientes industriales donde pueden existir interferencias eléctricas.

**Raspberry Pi 4:**

Desde la Raspberry Pi, el script S\_MODBUS\_OPCUA.py inicia la conexión con el Arduino a través del puerto serie (generalmente /dev/ttyACM1 en las raspberry) y realiza consultas cada cierto tiempo. Lee registros y los escribe en base a las necesidades que se detecten, se encarga de gestionar reconexiones automáticas en caso de que se caiga la conexión preestablecida. Paralelamente, también levanta un servidor OPC UA en Python que define variables personalizadas (nodos) para cada sensor, entrada o salida del sistema. Estos nodos pueden ser leídos o escritos por clientes externos, y representan el punto de unión entre el hardware de bajo nivel y la representación lógica del sistema.

**PC Cliente:**

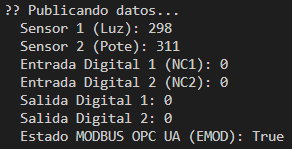
Desde la PC1, el script S\_OPC\_MQTT.py actúa como cliente del servidor OPC UA de la Raspberry. Se conecta a los nodos expuestos y extrae periódicamente sus valores, procesándolos y preparándolos para publicarlos mediante MQTT. En simultáneo, S\_OPC\_MQTT.py se encarga de mantener una conexión activa con el **broker EMQX**, el cual reemplaza al clásico Mosquitto por sus ventajas en accesibilidad y facilidad de uso. A través de EMQX se publica toda la información del sistema y se reciben comandos desde la interfaz web, los cuales son traducidos a actualizaciones en OPC UA.

**WebApp:**  
La interfaz HTML desarrollada con mqtt.js permite visualizar en vivo los valores de cada variable del sistema. Además, está equipada con botones y sliders para enviar comandos a través de tópicos MQTT, lo cual inicia un proceso inverso que termina controlando hardware físico. La interfaz también muestra en todo momento si hay conexión activa con el broker EMQX, y si el estado del servidor OPC UA y la conexión MODBUS es correcto, facilitando la detección de errores o caídas del sistema.

# Descripción de Archivos y Scripts

## 1. **S\_OPC\_MQTT.py**

Este script constituye el puente entre el servidor OPC UA y el broker EMQX. Fue diseñado para que, una vez conectado a ambos extremos, comience a publicar periódicamente los valores de sensores, entradas digitales y salidas en diferentes tópicos. Su diseño contempla la suscripción a otros tópicos mediante los cuales recibe instrucciones desde la interfaz web. Cada mensaje recibido es procesado, y si representa un cambio en el sistema (por ejemplo, modificar el PWM o cambiar el estado de una salida digital), se traduce en una actualización del nodo correspondiente en OPC UA. Además, este script reporta constantemente el estado de las conexiones (ya sea del broker como del servidor OPC), publicando en un tópico especial su estado para que la WebApp pueda indicarlo visualmente.



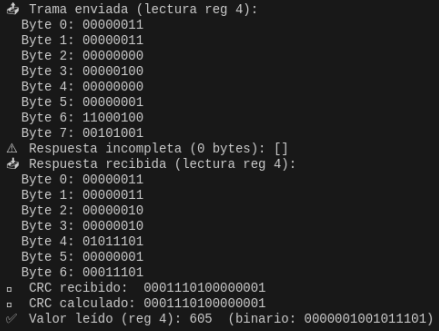
**Figura 3:** Publicación de datos MQTT.



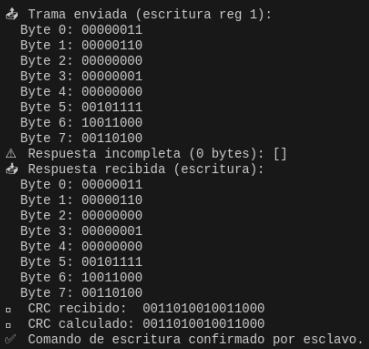
**Figura 4:** Intento de reconexión ante caída de server OPC.

## 2. S\_MODBUS\_OPCUA.py

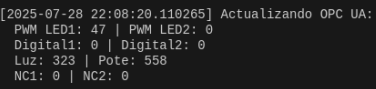
Este archivo Python cumple dos funciones esenciales. Por un lado, inicia un servidor OPC UA en la Raspberry con nodos personalizados para cada variable relevante del sistema. Por otro, implementa una lógica de maestro MODBUS que se comunica con el Arduino por el puerto serie. En cada ciclo de lectura, compara los valores actuales con los anteriores y escribe en los registros del Arduino solo si detecta un cambio. Esto minimiza el uso innecesario del canal MODBUS. A su vez, cada vez que lee datos del Arduino, actualiza los nodos del servidor OPC UA, manteniendo así la coherencia entre el mundo físico y la representación lógica del sistema. Este script también maneja reconexiones automáticas en caso de error y registra la última hora de sincronización.



**Figura 5:** Comando Modbus de lectura.



**Figura 6:** Comando Modbus escritura.



**Figura 7:** Publicación datos en OPC.

## 3. interface.html

La interfaz gráfica web fue construida usando HTML, JavaScript puro y la librería mqtt.js, que permite conectarse al broker **EMQX** mediante WebSockets. Esta interfaz ofrece al usuario una visualización clara de las variables del sistema en tiempo real, con indicadores numéricos o de estado. Además, incluye controles interactivos como botones de encendido/apagado y sliders para PWM, que permiten enviar comandos de forma intuitiva. Cada acción del usuario genera una publicación en un tópico MQTT, que luego es interpretada por el sistema para ejecutar la acción correspondiente. También ofrece feedback visual sobre la conexión a MQTT, OPC UA y MODBUS, para facilitar la operación.

## 4. Codigo\_Def.ino

Este es el firmware cargado en el Arduino Mega. Configura sensores como DHT22 para temperatura y humedad, entradas digitales simuladas y salidas digitales. Establece los registros MODBUS a utilizar, definiendo el orden, tamaño y tipo de dato de cada uno. Implementa la lógica de esclavo utilizando la librería ModbusRTUSlave, y asegura la correcta recepción de datos mediante validación CRC. La comunicación por UART está optimizada para evitar errores y permitir una transmisión continua y estable con la Raspberry Pi.

# Uso de EMQX como Broker MQTT

Para la publicación y suscripción de datos en tiempo real, se utilizó EMQX Cloud, un broker MQTT accesible por WebSocket, ideal para integraciones con interfaces web modernas. Este reemplazó configuraciones previas con Mosquitto local, simplificando la infraestructura y eliminando la necesidad de editar archivos de configuración para permitir conexiones desde múltiples IP.

Durante etapas anteriores del proyecto, se probaron brokers MQTT locales como Mosquitto, el cual requirió configuración del archivo **mosquitto.conf** para habilitar conexiones anónimas:

**listener 1883 0.0.0.0**

**allow\_anonymous true**

Estas directivas permiten que cualquier dispositivo pueda comunicarse sin autenticación previa, facilitando el desarrollo, aunque sacrificando seguridad. Se identificó que, si bien este método es útil en fases de prueba, no resulta adecuado para entornos productivos debido a la vulnerabilidad ante accesos no autorizados.

En comparación, EMQX Cloud ofrece una opción más robusta, con soporte para autenticación de usuarios y conexiones seguras mediante TLS, lo que lo convierte en una alternativa escalable y aplicable a implementaciones reales.

Además, se validó la estructura de tópicos del sistema, basada en jerarquías como:

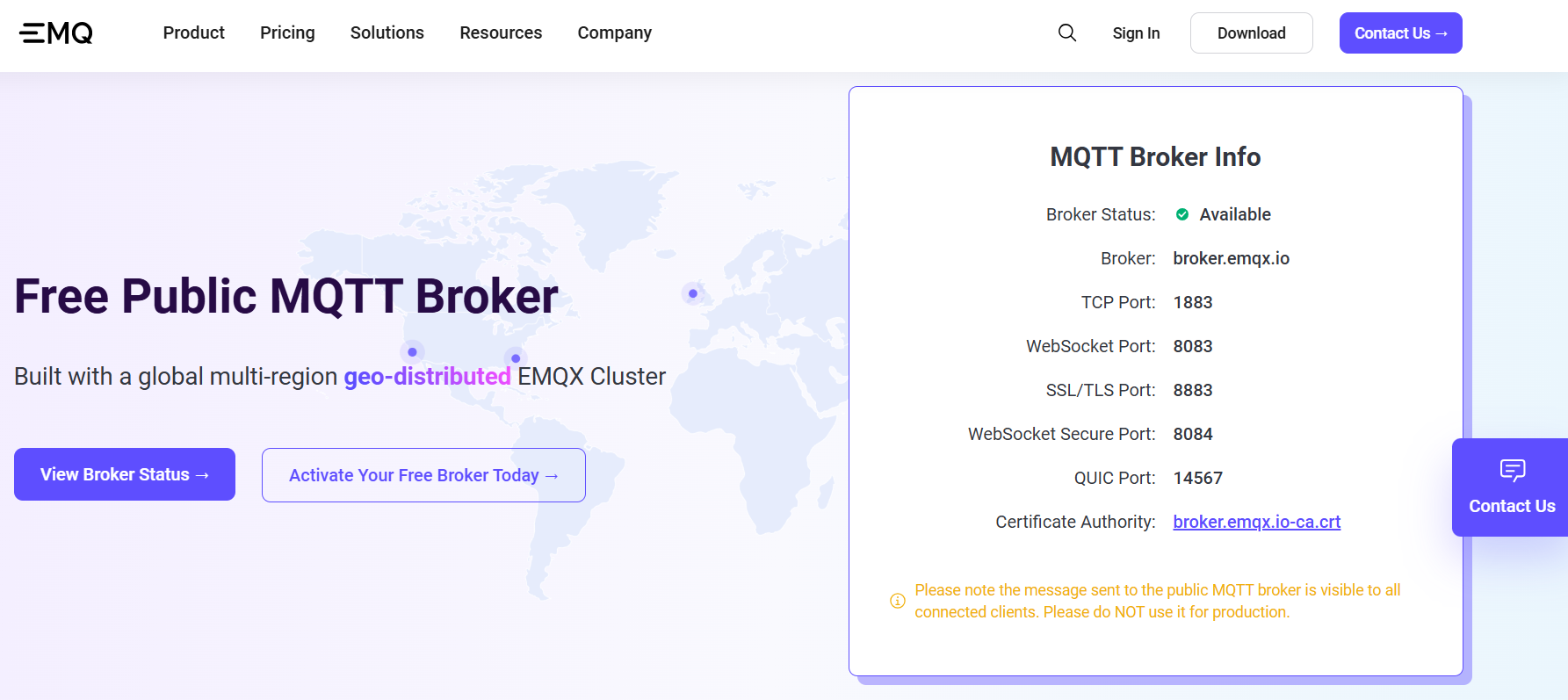
* modbus/plc/sensors/1 → Sensor analógico 1
* modbus/plc/inputs/2 → Entrada digital 2
* modbus/plc/outputs/1 → Estado salida digital 1

Esta organización permite que cualquier cliente MQTT pueda visualizar datos mediante comandos como:

**mosquitto\_sub -h <ip\_broker> -t "modbus/plc/#"**

Este comando se menciona a modo ilustrativo por haber sido utilizado en etapas previas de prueba local con Mosquitto. En la versión final del proyecto no se utilizó Mosquitto, sino EMQX Cloud, el cual ofrece una consola web y compatibilidad con WebSockets, siendo accedido desde la WebApp mediante mqtt.js sin necesidad de usar comandos por terminal.

La elección de EMQX garantizó compatibilidad inmediata con la WebApp basada en mqtt.js, permitiendo suscripciones por WebSocket sin pasos adicionales, y facilitando además la migración hacia mecanismos de seguridad más avanzados en futuras versiones del sistema.



**Figura 8:** Web informativa del broker web EMQX

## Precaución:

Las IPs de los códigos son fijas por lo que las IPs que figuran en los códigos son las utilizadas a la hora de realizar las pruebas, estas se deben de cambiar cuando se cambie de red o de dispositivos.

A su vez deben de estar todos los dispositivos en la misma red para que funcione la comunicación por wifi.

## Comunicación entre scripts:

### Comunicación entre OPC UA y MQTT:

En esta sección se muestran ejemplos de los mensajes y variables que se intercambian entre el servidor OPC UA y el broker MQTT. Esta comunicación permite que los datos del sistema físico (sensores, entradas y salidas) sean publicados en la nube y visualizados/controlados desde una interfaz web. Los tópicos MQTT y los payloads JSON reflejan el estado actual del sistema y permiten enviar comandos de control desde la web hacia el sistema físico.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dirección/Tópico MQTT** | **Variable OPC UA** | **Ejemplo de Payload (JSON)** | **Descripción** |
| modbus/plc/sensors/1 | Luz | {"value": 2.85, "unit": "V"} | Valor del sensor de luz |
| modbus/plc/sensors/2 | Pote | {"value": 1.23, "unit": "V"} | Valor del potenciómetro |
| modbus/plc/inputs/1 | NC1 | {"value": true} | Estado de entrada digital 1 |
| modbus/plc/inputs/2 | NC2 | {"value": false} | Estado de entrada digital 2 |
| modbus/plc/outputs/1 | Digital1 | {"value": 1} | Estado de salida digital 1 |
| modbus/plc/outputs/2 | Digital2 | {"value": 0} | Estado de salida digital 2 |
| modbus/plc/control/digital/1 | Digital1 | {"value": 1} | Comando para activar salida digital 1 |
| modbus/plc/control/analog/1 | PWM\_LED1 | {"value": 128} | Comando para PWM LED1 (0-255) |
| modbus/plc/status/opc | - | {"connected": true} | Estado de conexión OPC UA |
| modbus/plc/status/modbus | EMOD | {"connected": false, "error": "..."} | Estado de conexión MODBUS |

### Comunicación entre OPC UA y Arduino (MODBUS):

Aquí se detallan las variables y registros que se sincronizan entre el servidor OPC UA (Python) y el Arduino a través del protocolo MODBUS RTU. Esta comunicación es responsable de la adquisición de datos de sensores y el control de salidas físicas. Cada variable OPC UA corresponde a un registro MODBUS en el Arduino, permitiendo la lectura y escritura de valores en tiempo real.

* 0x03: Lectura de registro (Read Holding Register)
* 0x06: Escritura de registro (Write Single Register)

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable OPC UA** | **Registro MODBUS Arduino** | **Función** | **Descripción** |
| PWM\_LED1 | 1 | 0x03/0x06 | Salida analógica PWM 1 (0-255) |
| PWM\_LED2 | 2 | 0x03/0x06 | Salida analógica PWM 2 (0-255) |
| Digital1 | 3 | 0x03/0x06 | Salida digital 1 (0/1) |
| Luz | 4 | 0x03 | Sensor de luz (lectura) |
| Pote | 5 | 0x03 | Potenciómetro (lectura) |
| Digital2 | 6 | 0x03/0x06 | Salida digital 2 (0/1) |
| NC1 | 7 | 0x03 | Entrada digital 1 (lectura) |
| NC2 | 8 | 0x03 | Entrada digital 2 (lectura) |

# Conclusión

La implementación de este sistema demuestra la capacidad de integrar de manera eficiente los protocolos industriales MODBUS RTU, OPC UA y MQTT en una arquitectura distribuida orientada al monitoreo y control en tiempo real.

OPC UA actuó como servidor estructurado, permitiendo organizar las variables adquiridas desde el entorno físico, mientras que MQTT posibilitó una distribución ágil y ligera hacia los distintos clientes, incluyendo una interfaz web que opera mediante WebSockets.

La comunicación entre sensores, nodos intermedios y usuarios finales se mantuvo estable y bidireccional durante todo el proyecto. La interfaz desarrollada funcionó como una solución tipo SCADA, ofreciendo visualización de estados, retroalimentación de errores y capacidad de control remoto sobre el sistema físico.

Gracias al enfoque modular, la solución es escalable, flexible y fácilmente adaptable a contextos industriales reales o proyectos futuros vinculados al Internet de las Cosas (IoT) y la Industria 4.0.

# Bibliografía

Eclipse Mosquitto. (s.f.). *Documentación de Mosquitto*. Recuperado el 12 de noviembre de 2024, de <https://mosquitto.org/documentation/>

EMQX (s.f.) *MQTT Broker*. Recuperado el 28 de julio de 2025, de <https://www.emqx.com/en/mqtt/public-mqtt5-broker>

IBM. (24 de agosto de 2022). *Calidad de servicio y gestión de conexiones en Integration Bus 10.0*. Recuperado el 12 de noviembre de 2024, de <https://www.ibm.com/docs/es/integration-bus/10.0?topic=bus-quality-service-connection-management>

MQTT. (2024). *MQTT.org*. Recuperado el 12 de noviembre de 2024, de <https://mqtt.org/>

OPC Foundation. (2024). *OPC UA: Tecnologías OPC*. Recuperado el 12 de noviembre de 2024, de <https://opcfoundation.org/about/opc-technologies/opc-ua/>

The Eclipse Foundation. (29 de abril de 2024). *Paho MQTT - Python Client*. Recuperado el 12 de noviembre de 2024, de <https://pypi.org/project/paho-mqtt/>