

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](#_Toc204452045)

[Introducción 2](#_Toc204452046)

[Objetivos del Proyecto 3](#_Toc204452047)

[Descripción General del Sistema 3](#_Toc204452048)

[**Descripción del Flujo de Datos** 4](#_Toc204452049)

[**Descripción de Archivos y Scripts** 5](#_Toc204452050)

[**1. S\_OPC\_MQTT.py** 5](#_Toc204452051)

[**2. S\_MODBUS\_OPCUA.py** 6](#_Toc204452052)

[**3. interface.html** 6](#_Toc204452053)

[**4. Codigo\_Def.ino** 6](#_Toc204452054)

[**Uso de EMQX como Broker MQTT** 6](#_Toc204452055)

[Configuración y Funcionamiento de los Componentes 7](#_Toc204452056)

[Precaución: 8](#_Toc204452057)

[Estructura de la Comunicación y Diagrama del Sistema 8](#_Toc204452058)

[Conclusión 8](#_Toc204452059)

# 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 diferentes dispositivos. La aplicación de estos protocolos responde a la necesidad de diseñar una infraestructura de transmisión de datos que permita monitorear variables en tiempo real, con un enfoque en eficiencia y seguridad. 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. Para cumplir con este propósito, se estudian en detalle los bloques funcionales del sistema, así como sus componentes y tipos de conexión, permitiendo crear un canal de comunicación seguro y eficaz entre dispositivos de monitoreo y acción en conjunto con los de almacenamiento de datos.

# Objetivos del Proyecto

El principal objetivo de este trabajo es la correcta implementación de los protocolos Modbus, OPC UA y MQTT, comprendiendo su funcionamiento, características de seguridad y estructura de datos. Estos protocolos están diseñados para facilitar el intercambio de datos en sistemas industriales y de monitoreo. Mediante el uso de sensores, el sistema desarrollado captura datos analógicos y digitales, los cuales son transmitidos de manera ordenada y segura a través de una infraestructura de red distribuida.

Dentro de este marco, se establecieron objetivos específicos que guiaron el desarrollo de este proyecto: en primer lugar, diferenciar los componentes y elementos involucrados en la comunicación en las diferentes etapas, lo que incluye identificar los roles de cada uno de los dispositivos dentro de la arquitectura del sistema. En segundo lugar, se plantea la configuración y puesta en funcionamiento de los componentes necesarios para establecer la comunicación entre dispositivos. Además, se estableció una estructura de tópicos adecuada para la transmisión y almacenamiento de datos, lo que permite que los suscriptores reciban información ordenada en tiempo real.

# 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.

**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. Cuanta 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 rescritura en caso de que el CRC no de correctamente y a su vez cuanta con función re 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.

**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 pre establecida. Paralelamente, tambien 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.

**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.

**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 este sistema se optó por utilizar **EMQX Cloud** como broker MQTT en lugar de Mosquitto. EMQX es un broker de alto rendimiento desarrollado en Erlang, que se destaca por su capacidad de manejar miles de conexiones simultáneas y su enfoque en proyectos IoT distribuidos.

El principal motivo para esta elección fue su **servicio gratuito en la nube** a través de broker.emqx.io, el cual permite conectarse sin necesidad de instalar o configurar nada en máquinas locales. Esto es especialmente útil en entornos de desarrollo, pruebas o prototipado, donde se busca la máxima accesibilidad desde distintas ubicaciones o dispositivos.

EMQX permite conexiones tanto por **TCP como WebSocket**, siendo este último fundamental para las aplicaciones web como la interface.html. Su panel de administración también facilita la supervisión de clientes conectados, tráfico y tópicos activos.

Aunque Mosquitto es una excelente alternativa local, para este proyecto en particular se privilegió la simplicidad y la apertura de EMQX, que elimina la necesidad de abrir puertos, configurar NAT o lidiar con firewalls.

# Configuración y Funcionamiento de los Componentes

Para el funcionamiento del sistema, la Raspberry Pi se configuró con un script denominado OPCUA-Server-Def.py, el cual permite simular datos de sensores en un entorno controlado. Este script establece un servidor OPC UA en la IP 192.168.0.150, en el puerto 4840, y genera datos que imitan valores reales de sensores de temperatura y presión. Estas variables están accesibles para el cliente OPC UA que se encuentra en Notebook Esteban.

Notebook de Esteban, configurada como cliente OPC UA y publicador MQTT, ejecuta el script Cliente\_Publicador.py. Este código permite al dispositivo actuar en dos roles: como cliente OPC UA para capturar datos desde el servidor y como publicador MQTT, lo que posibilita enviar los datos al broker MQTT. La información de temperatura, presión y tiempo se publica en tópicos específicos dentro del broker ubicado en Notebook de Lautaro, como "sensors/temp", "sensors/press" y "sensors/time", los cuales están disponibles para cualquier dispositivo suscriptor.

Notebook de Lautaro alberga el broker MQTT, que actúa como el nodo central para la recepción y distribución de los mensajes. Configurado mediante Mosquitto, el broker MQTT recibe los datos publicados desde Notebook de Esteban y los almacena en los tópicos previamente definidos.

Mosquitto es un bróker MQTT open source y que cuenta con diferentes opciones de seguridad, cuando se instala para permitir que cualquier dispositivo sea capaz de comunicarse con el desde cualquier IP se deben de añadir en el archivo “mosquitto.conf” ciertos parámetros.

Debemos de añadir **listener 1883 0.0.0.0** y **allow\_anonymous true**, estassentencias le dice al mosquito que escuche en todas las direcciones del puerto 1883 y que permita la conexión de dispositivos anónimos, esto si bien reduce la seguridad del sistema nos facilita el no tener que implementar mecanismos de autenticación dentro de los códigos de comunicación.

Esto habilita que cualquier cliente MQTT pueda suscribirse a los tópicos que creamos y recibir la información en tiempo real. Para que un cliente vea todos los tópicos debe de suscribirse utilizando el siguiente comando:

**mosquitto\_sub -h ip\_broker -t "sensores/#"**

Lo que hace ese comando es captar todos los tópicos que esta saciados al grupo sensores, si se cambia # por el nombre de un tópico se mostrara ese solo.

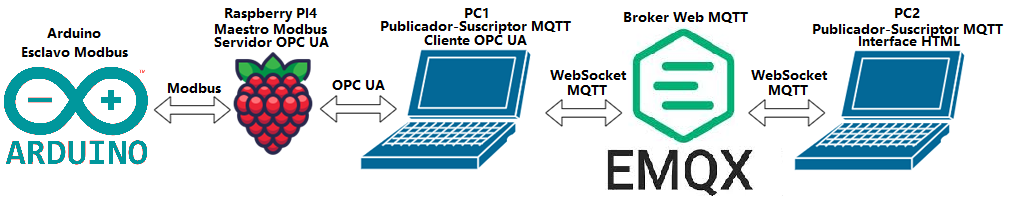
## Precaución:

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.

# 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:



# Conclusión

La implementación de un sistema de comunicación basado en los protocolos OPC UA y MQTT demuestra que ambos pueden trabajar de manera complementaria para ofrecer una solución efectiva y segura en sistemas de monitoreo de datos. OPC UA, al actuar como servidor para la adquisición de datos en un entorno industrial, proporciona una capa robusta de seguridad y estructura de datos. A su vez, MQTT facilita la distribución de estos datos de forma rápida y eficiente, mejorando la interoperabilidad y permitiendo la conectividad de múltiples dispositivos suscriptores.

La integración de estos dos protocolos permitió cumplir con los objetivos del proyecto y demostrar su potencial en aplicaciones de comunicación de datos en tiempo real, aportando seguridad, eficiencia y una estructura de datos que puede adaptarse a diferentes necesidades y configuraciones dentro de un sistema embebido.