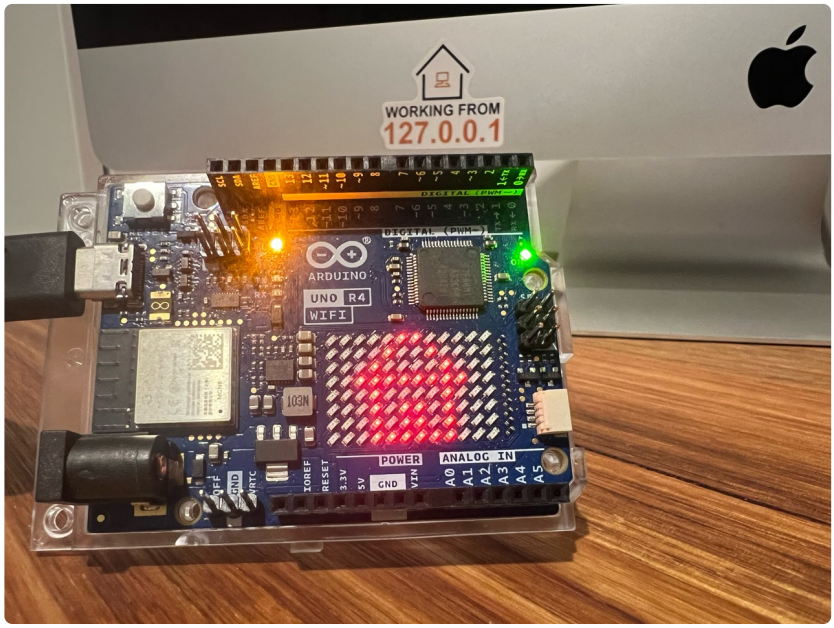
Connecting an Arduino UNO R4 WiFi to Home Assistant with MQTT

September 13, 2026 / Gavin Jackson

home-assistant arduino mqtt electronics iot automation



The UNO R4 WiFi used in this tutorial, with its onboard LED and 12 × 8 LED matrix illuminated.

After using ESPHome to bring three ESP32 devices into Home Assistant, I wanted to understand the more explicit route: have a board describe its own entities, receive commands and report state through MQTT.

This tutorial connects an Arduino UNO R4 WiFi to Home Assistant using ArduinoMqttClient and MQTT discovery. It needs no external components. By the end, Home Assistant will have two controls for the UNO's onboard hardware and two diagnostic sensors.

The earlier experiments and their ESPHome source files are in Three Small ESP32 Devices, One Home Assistant. This article concentrates on the Arduino and MQTT setup.

I run Home Assistant and ESPHome as Docker Compose services with host networking. The tutorial keeps that arrangement and adds a Mosquitto container to provide the broker required by the UNO.

Why the UNO R4 WiFi takes a different route

The Arduino UNO R4 WiFi has two processors with different jobs. The Arduino sketch runs on a **Renesas RA4M1**. An **ESP32-S3** provides wireless connectivity through the board's connectivity firmware.

For this tutorial, select the UNO R4 WiFi in the Arduino IDE and upload a normal Arduino sketch. Leave the connectivity firmware in place; flashing a generic ESP32 ESPHome configuration onto the coprocessor is a different project.

We will use **ArduinoMqttClient**, Arduino's MQTT library. Its client class is named `MqttClient`, and it works over the `WiFiClient` supplied by the UNO R4's `WiFiS3` library. This is the specific library to install when following the tutorial; other Arduino MQTT libraries have different APIs. Arduino's library repository contains its examples and source.

MQTT uses a **broker** to pass messages between clients. Both the Arduino and Home Assistant connect to that broker. A **topic** is the address for a message, such as the command to turn an LED on. **Discovery messages** describe the board's entities so Home Assistant can create them automatically.

```
%%{init: {"themeVariables": {"lineColor": "#94a3b8"}}}%%
flowchart TB
 board["Arduino UNO R4 WiFi<br/>LED, matrix, Wi-Fi signal and uptime"]
 broker["Mosquitto MQTT broker<br/>Runs on your local network"]
 ha["Home Assistant<br/>Two switches and two sensors"]
 board -->|"Discovery, state and availability"| broker
 broker -->|"Discovery, state and availability"| ha
 ha -->|"ON / OFF commands"| broker
 broker -->|"ON / OFF commands"| board

 linkStyle default stroke:#94a3b8,stroke-width:2px;
```

Everything in this example runs over the local network. An Arduino Cloud account is not required.

What we will expose

You only need the UNO R4 WiFi and a USB-C data cable for the hardware side. No additional sensors or wiring are required.

Home Assistant entity	Type	What it does
Onboard LED	Switch	Turns the built-in D13 LED on or off.
LED matrix	Switch	Shows a pattern on the onboard 12 × 8 LED matrix, or clears it.
Wi-Fi signal	Sensor	Reports received signal strength in dBm.
Uptime	Sensor	Reports how many seconds the sketch has been running.

The LED controls let us test commands going to the board. The two sensors demonstrate information coming back. They are board diagnostics rather than environmental measurements.

You will also need a working Home Assistant installation, a 2.4 GHz Wi-Fi network, and an MQTT broker reachable from both the board and Home Assistant.

1. Set up the MQTT broker

If you already have a working MQTT broker connected to Home Assistant, reuse it and move to the Arduino setup. The examples below assume a fresh broker.

Add Mosquitto to docker-compose.yml

Here is the complete `docker-compose.yml` for the Linux host, with the Mosquitto MQTT broker added alongside Home Assistant and ESPHome:

```
services:
 homeassistant:
 container_name: homeassistant
 image: "ghcr.io/home-assistant/home-assistant:stable"
 volumes:
 - /opt/home-assistant/config:/config
 - /etc/localtime:/etc/localtime:ro
 - /run/dbus:/run/dbus:ro
 restart: unless-stopped
 stop_grace_period: 60s
 privileged: true
 network_mode: host
 environment:
 TZ: Australia/Sydney

 esphome:
 container_name: esphome
 image: ghcr.io/esphome/esphome:stable
 volumes:
 - /opt/home-assistant/esphome-config:/config
 - /etc/localtime:/etc/localtime:ro
 restart: unless-stopped
 network_mode: host

 mosquitto:
 container_name: mosquitto
 image: eclipse-mosquitto:2
 volumes:
 - /opt/home-assistant/mosquitto/config:/mosquitto/config
 - /opt/home-assistant/mosquitto/data:/mosquitto/data
 - /opt/home-assistant/mosquitto/log:/mosquitto/log
 restart: unless-stopped
 network_mode: host
```

All three services use host networking, so Mosquitto listens directly on the Docker host without a `ports:` mapping. ESPHome continues to provide Device Builder for the earlier projects; the UNO communicates through Mosquitto.

Prepare the broker configuration and accounts

Before starting Mosquitto, create its directories and configuration. These mount points follow the official Eclipse Mosquitto container layout:

```
sudo mkdir -p /opt/home-assistant/mosquitto/{config,data,log}

sudo tee /opt/home-assistant/mosquitto/config/mosquitto.conf >/dev/null <<'EOF'
persistence true
persistence_location /mosquitto/data/
log_dest stdout

listener 1883
allow_anonymous false
password_file /mosquitto/config/password_file
EOF

sudo chown -R 1883:1883 /opt/home-assistant/mosquitto
```

The official image runs Mosquitto as user and group ID `1883`, so that ownership lets it write retained data and read its configuration through the bind mounts. The image's Docker README documents the container user and mount points.

Create separate broker accounts for the UNO and Home Assistant. Each command prompts for a password, so the passwords do not appear in shell history:

```
sudo docker run --rm -it --user 1883:1883 \
 -v /opt/home-assistant/mosquitto/config:/mosquitto/config \
 --entrypoint mosquitto_passwd \
 eclipse-mosquitto:2 \
 -c /mosquitto/config/password_file uno_r4

sudo docker run --rm -it --user 1883:1883 \
 -v /opt/home-assistant/mosquitto/config:/mosquitto/config \
 --entrypoint mosquitto_passwd \
 eclipse-mosquitto:2 \
 /mosquitto/config/password_file ha_mqtt

sudo chmod 600 /opt/home-assistant/mosquitto/config/password_file
```

Use `-c` only for the first account because it creates or overwrites the file. The second command adds another user. These are MQTT broker accounts, separate from Home Assistant users; the Mosquitto password utility documentation explains the distinction and options.

Start Mosquitto and connect Home Assistant

From the directory containing `docker-compose.yml`, start the new broker service:

```
sudo docker compose up -d mosquitto
sudo docker compose logs --tail=50 mosquitto
```

The broker log should show a listener on port `1883`. Because all three containers use host networking, Home Assistant can reach Mosquitto at `127.0.0.1:1883`. In **Settings → Devices & services**, choose **Add integration → MQTT** and enter that address with the `ha_mqtt` credentials. MQTT discovery is enabled by default. The Home Assistant MQTT documentation covers the same manual broker setup.

The Arduino is outside Docker, so it must use the **LAN IP address of the Docker host**, port `1883`, and the `uno_r4` credentials. Allow TCP 1883 from the trusted LAN if the host has a firewall. A DHCP reservation for the Docker host keeps that address stable.

> ***Keep this example on your local network***
>
> *The sketch uses authenticated MQTT on port 1883 without TLS, so MQTT credentials and messages are not encrypted in transit. Keep that listener restricted to your trusted LAN and do not forward it from the internet. TLS requires a suitable client and certificate configuration as well as a broker listener; changing the port alone does not enable it.*

2. Prepare the Arduino IDE

Install the Arduino IDE, connect the UNO R4 WiFi with a data-capable USB cable, and then:

1. Open **Boards Manager** and install **Arduino UNO R4 Boards** by Arduino (also known as the Arduino Renesas UNO board package).
2. Select **Arduino UNO R4 WiFi** and its USB port in the board selector.
3. Open **Library Manager** and install **ArduinoMqttClient** by Arduino.

`WiFiS3` and `Arduino_LED_Matrix` come with the board package. The sketch uses the matrix library directly, so it does not need a graphics library. Arduino documents the board's Wi-Fi examples and LED matrix separately if you want to explore either feature.

3. Download the sketch and add your credentials

Download the complete UNO R4 WiFi sketch and the example credentials header. If your browser displays either file as text, save it using the filename shown below.

Keep the sketch in a folder with the same name, and rename the example header to `arduino_secrets.h`:

```
uno-r4-wifi-home-assistant/
├── uno-r4-wifi-home-assistant.ino
└── arduino_secrets.h
```

Edit `arduino_secrets.h` with your own details:

```
#pragma once

#define SECRET_SSID "YOUR_WIFI_NAME"
#define SECRET_PASS "YOUR_WIFI_PASSWORD"
#define SECRET_MQTT_HOST "192.168.1.10"
#define SECRET_MQTT_PORT 1883
#define SECRET_MQTT_USER "uno_r4"
#define SECRET_MQTT_PASS "YOUR_MQTT_PASSWORD"
```

Use the broker's LAN IP address for `SECRET_MQTT_HOST`. `localhost` would mean the Arduino itself. If Mosquitto runs separately from Home Assistant, use that broker machine's address. A DHCP reservation for the broker avoids having to update and re-upload the sketch when its address changes.

Keep your filled-in credentials file out of source control and out of any public web directory. Open the sketch from your local Arduino projects folder.

The sketch has a device ID of `uno_r4_wifi_01`. Leave that alone for your first board. For a second board, change `DEVICE_ID` before uploading; each board needs its own MQTT client ID, discovery IDs and topics. The example derives those from this one value.

4. Understand the useful parts of the sketch

The download contains the complete program, including connection recovery. These excerpts explain its main jobs; upload the downloaded sketch rather than assembling the excerpts into a new program.

```
#include <WiFiS3.h>
#include <ArduinoMqttClient.h>
#include <Arduino_LED_Matrix.h>
#include "arduino_secrets.h"

WiFiClient wifiClient;
MqttClient mqttClient(wifiClient);
ArduinoLEDMatrix matrix;
```

`WiFiClient` provides the network connection. `MqttClient` handles publishing and subscribing. The LED matrix object controls the board's display.

Discovery creates one device with four entities

On connecting, the board publishes a configuration for each entity under Home Assistant's default `homeassistant` discovery prefix. For example, the LED configuration goes to:

```
homeassistant/switch/uno_r4_wifi_01/led/config
```

Each entity has its own `unique_id`, while all four share a device identifier. That groups the switches and sensors under a single UNO device in Home Assistant. The sketch publishes these configurations automatically; there is no manual entity YAML to add. Home Assistant describes the message format in its MQTT discovery documentation.

Commands and reported state use separate topics

For the onboard LED, the topic pair is:

```
arduino/uno_r4_wifi_01/led/set Home Assistant sends ON or OFF
arduino/uno_r4_wifi_01/led/state Arduino reports ON or OFF
```

The board handles the command, changes the output, and publishes its resulting state. The matrix follows the same pattern. This uses Home Assistant's MQTT switch state confirmation so the dashboard reflects the board's reply.

The message handler updates the LED and flags that there is a new state to publish:

```
if (topic == ledCommandTopic) {
 ledOn = turnOn;
 digitalWrite(LED_BUILTIN, ledOn ? HIGH : LOW);
 stateDirty = true;
}
```

For the matrix, the sketch uses an 8-row, 12-column bitmap of a little house:

```
if (matrixOn) {
 matrix.renderBitmap(house, 8, 12);
} else {
 matrix.clear();
}
```

The `house` array is included in the download. Change its zeroes and ones to draw your own pattern.

Recovery is part of the example

Discovery and reported states are **retained**: the broker stores their most recent values for new subscribers. Commands are **not retained**, so an old command is not replayed when a board reconnects.

The sketch also registers an MQTT **Last Will** of `offline`, publishes `online` when ready, and listens for Home Assistant's birth message on `homeassistant/status` so it can announce its entities again. Home Assistant documents these startup and availability behaviours.

The main loop polls the MQTT client frequently and spaces out reconnection attempts. A long `delay()` between sensor readings would also delay incoming commands and connection maintenance. Sensor updates use a 30-second timer instead. Connection and acknowledgement waits still use synchronous library calls; this is a small tutorial sketch, not a real-time controller.

5. Upload and find the device

Click **Verify** in the IDE, then **Upload**. Open **Serial Monitor** at **115200 baud** to follow the Wi-Fi and MQTT connection messages.

Once the board connects, open **Settings → Devices & services → MQTT** in Home Assistant and look for the UNO device. Open it to find the two switches and two diagnostic sensors. Assign it to an area such as Study, then add the entities to a dashboard if desired.

Try each control:

1. Turn **Onboard LED** on and off, and check the D13 LED on the board.
2. Turn **LED matrix** on to show the pattern, then off to clear it.
3. Watch **Uptime** increase and **Wi-Fi signal** update. A value closer to zero means a stronger received signal: `-50 dBm` is stronger than `-75 dBm`.

Reset the board and confirm that it reconnects. Both outputs start off after a reset; the sketch reports that fresh state to Home Assistant. A temporary network reconnection does not itself reset the outputs.

Unplug the board to check availability. Home Assistant should eventually mark its entities unavailable after the broker detects the lost connection; this is not necessarily immediate. Reconnect the USB cable and check that the board returns. Restarting Home Assistant's MQTT integration is another useful check that discovery and state recover.

> ***Example validation***
>
> *The downloadable sketch has been compiled for `arduino:renesas_uno:unor4wifi` using board package 1.6.0 and ArduinoMqttClient 0.1.8. The screenshots below show the UNO's two controls, diagnostic readings and state changes in Home Assistant. Use the checks above to verify reconnection and availability on your own network.*

If something does not appear

Symptom	What to check
`WiFiS3.h` or `Arduino_LED_Matrix.h` is missing	Install the UNO R4 board package and select UNO R4 WiFi, not UNO R3 or a generic ESP32.
`ArduinoMqttClient.h` is missing	Install ArduinoMqttClient by Arduino in Library Manager.
Wi-Fi never connects	Check the SSID, password, 2.4 GHz coverage and the connection messages in Serial Monitor.
Wi-Fi connects but MQTT fails	Check the broker's LAN IP, port, username and password. Allow TCP 1883 between the board and broker; guest Wi-Fi isolation can block it. Check the broker logs for authentication failures.
MQTT connects but no device appears	Confirm Home Assistant uses the same broker and that MQTT discovery is enabled with the default `homeassistant` prefix.
Switches appear but do not respond	Listen to the command and state topics, check the board is online, and confirm it is receiving `ON` or `OFF`.
Two boards keep disconnecting	Give each board a different `DEVICE_ID`; MQTT client IDs must be unique.
Old entities return after renaming a device	The broker still holds the old retained discovery messages. Stop the old publisher and publish an empty retained payload to each old entity's exact `.../config` topic.

The MQTT integration's **Configure** screen includes **Listen to a topic**. Listening to `arduino/uno_r4_wifi_01/#` lets you see the example's traffic. To inspect discovery, listen to `homeassistant/+/uno_r4_wifi_01/+/config`. The `#` wildcard includes all topics below a prefix; `+` matches one topic level.

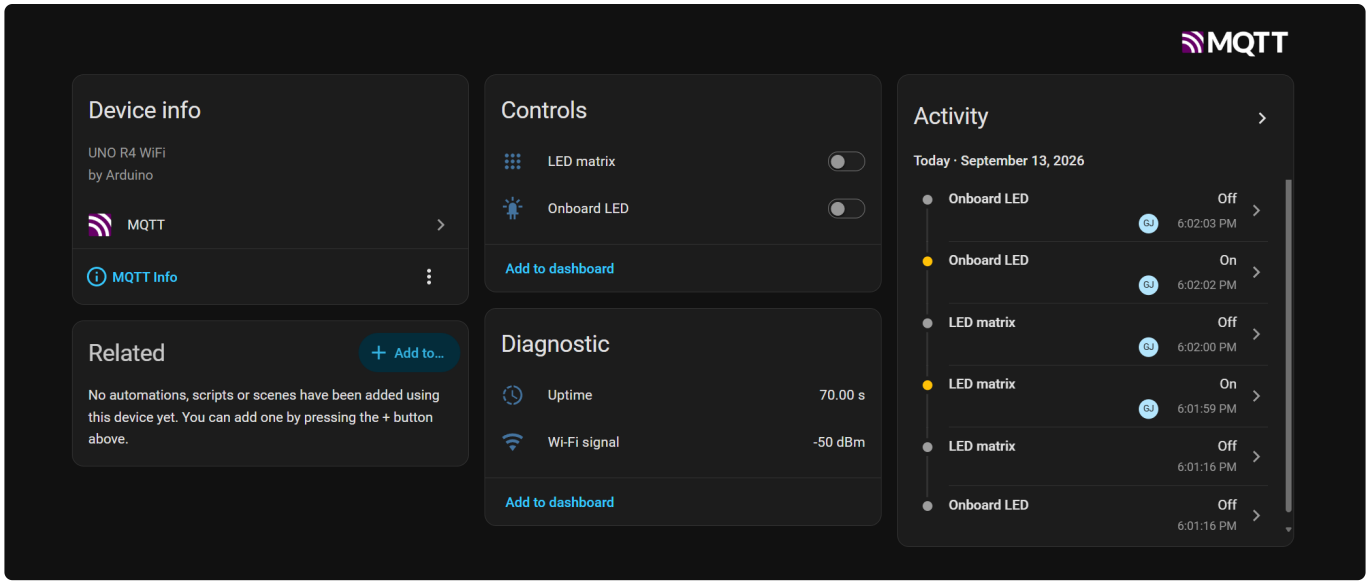
You can also use that screen to publish `ON` to `arduino/uno_r4_wifi_01/led/set`. Leave **Retain** off for this command. The LED should change and the board should publish `ON` on its state topic.

If the IDE reports that the UNO's connectivity firmware needs updating, use Arduino's Firmware Updater instructions. The update overwrites the current sketch. **Disconnect and reconnect the board after the firmware update**, then upload the tutorial sketch again.

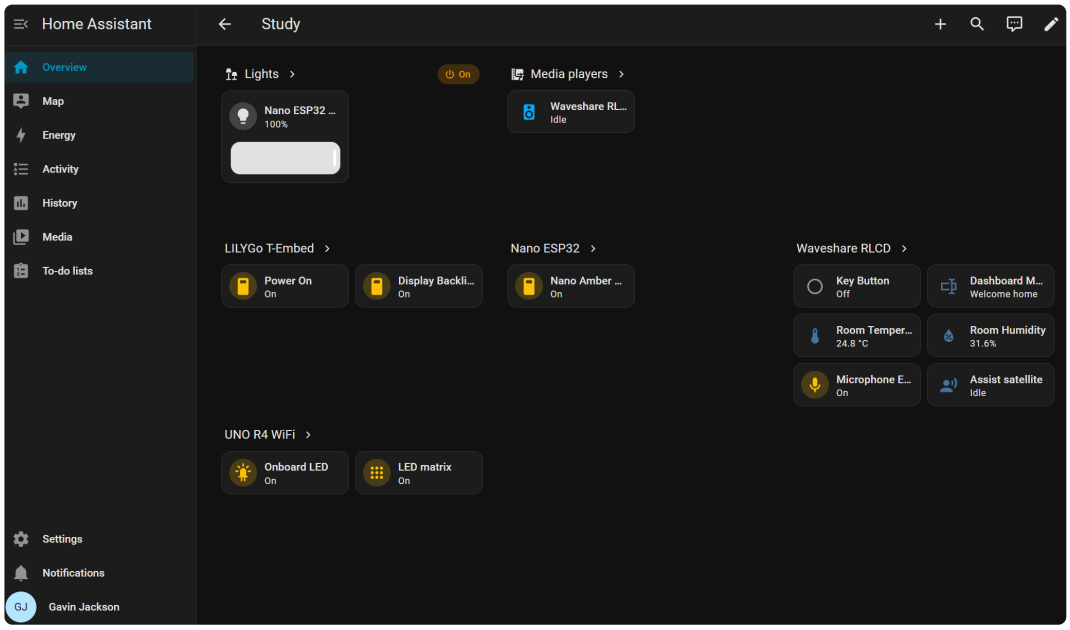
From an LED to something useful

An LED is a good first test because the result is visible. Once the two-way connection works, the same approach can publish readings from an attached sensor or show an automation's status on the matrix. Give each new entity a unique ID, define its command or state topic, and let discovery describe it to Home Assistant.

Writing this small MQTT client shows what an integration needs to do: describe the device's features, receive commands, report the resulting state and recover when either end restarts. There is plenty to learn from making a board on the bench respond to a switch on the phone.



It's a beautiful thing when a plan comes together!



The UNO R4 WiFi joins the LILYGO T-Embed, Nano ESP32 and Waveshare RLCD on the Study dashboard.