Setting Up MQTT

Cloud and self-hosted brokers for MicroPython IoT devices

Document type: Guide — AP Design Labs Free Library
Version: 1.0
Date: October 2026
Author: Andrew Paton, AP Design Labs

ABOUT AP DESIGN LABS

AP Design Labs is an Adelaide-based engineering studio specialising in bespoke, purpose-built technology for industries where off-the-shelf products simply don't cut it — farmers, vignerons, solar operators and EV charging. Every product is designed, prototyped and manufactured in-house, built to work reliably and independently, without subscription lock-in.

© AP Design Labs, Happy Valley, South Australia. All prices in Australian dollars (AUD).

1. Introduction

MQTT is the messaging protocol behind a huge share of the Internet of Things. It is small enough to run on a A$10 microcontroller, tolerant of patchy Wi-Fi and mobile links, and lets any number of devices, dashboards and apps talk to each other without knowing anything about one another. If you want a tank-level sensor in the back paddock to show up on your phone, or a button on a dashboard to switch a pump, MQTT is usually the simplest reliable way to do it.

This guide takes you from nothing to a working system. It explains how MQTT works, compares the free and low-cost broker options available in 2026, then walks step by step through three set-ups: a managed cloud broker (HiveMQ Cloud), a second cloud option (EMQX Serverless), and a self-hosted Mosquitto broker on a Raspberry Pi. It finishes with a complete, tested-pattern MicroPython program for the ESP32 or Raspberry Pi Pico W that you can load straight from Thonny.

1.1 What you will need

Item	Notes	Approx. cost (AUD)
Microcontroller	ESP32 dev board or Raspberry Pi Pico W / Pico 2 W with MicroPython 1.23 or newer	$10 – $15
USB data cable	Must be a data cable, not charge-only	$5
Thonny	Free IDE from thonny.org (Windows, macOS, Linux)	Free
Broker account or server	HiveMQ Cloud / EMQX free tier, or a Raspberry Pi for Mosquitto	Free – $90
MQTT Explorer (optional)	Free desktop client for watching and sending messages	Free

Table 1 — Parts and software. Hardware prices are typical Australian retail and will vary.

> **Who this is for:** makers, farmers, students and small businesses who are comfortable copying files to a board with Thonny. No networking background is assumed; every new term is explained the first time it appears.

2. How MQTT works

MQTT uses a **publish / subscribe** model. Devices never talk to each other directly. Instead, every device opens one connection to a central server called the **broker**. A device that has something to say **publishes** a message to a named **topic**. Any device that is interested in that topic **subscribes** to it, and the broker forwards a copy of every matching message. The sender does not need to know who is listening, and listeners can come and go at any time.

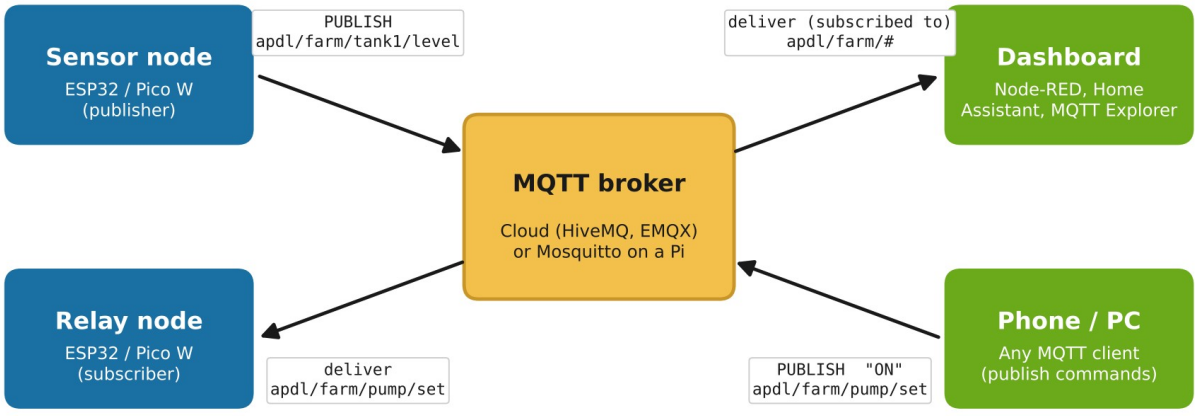


Figure 1 — Publishers and subscribers only ever talk to the broker. Topics decide who receives what.

2.1 Key terms

Term	Meaning
Broker	The server that receives every message and routes it to subscribers. Cloud services run this for you; Mosquitto lets you run your own.
Client	Anything that connects to the broker: a microcontroller, a dashboard, a phone app or a script. Each needs a unique client ID.
Topic	A text address made of levels separated by slashes, e.g. `apdl/farm/tank1/level`. Topics are created automatically the first time they are used.
Payload	The message body. Up to 256 MB in theory, but keep it small: a number, ON / OFF, or a short JSON object.
Wildcards	Subscribers can use `+` for exactly one level (`apdl/farm/+/level`) or `#` for everything below (`apdl/farm/#`). Publishers cannot use wildcards.
Retained message	The broker keeps the last retained message on a topic and hands it to new subscribers immediately — ideal for current state such as “pump is ON”.
Last Will (LWT)	A message the client registers when it connects. If the client drops off without saying goodbye, the broker publishes it for them, e.g. `offline`.
Keep-alive	The longest gap, in seconds, between packets. If the broker hears nothing for 1.5 × this time it treats the client as gone.

Table 2 — MQTT vocabulary.

2.2 Quality of Service (QoS)

Each publish and subscribe chooses a QoS level. The delivered level is the lower of the publisher’s and the subscriber’s choice.

QoS	Guarantee	Use it for
0	At most once — fire and forget	Frequent sensor readings where the next value will arrive soon anyway
1	At least once — acknowledged, may duplicate	Commands and alarms. Make the action safe to repeat (“set ON”, not “toggle”)
2	Exactly once — four-step handshake	Billing or counting events. Not supported by the MicroPython umqtt library

Table 3 — QoS levels.

2.3 Ports and security

Port	Transport	When to use
1883	Plain TCP, unencrypted	Only inside your own network, e.g. a Pi broker on the farm LAN
8883	MQTT over TLS (encrypted)	Anything that crosses the internet. Required by HiveMQ Cloud and EMQX Serverless
8884 / 8084 / 443	MQTT over secure WebSockets	Browser dashboards and networks that block 8883

Table 4 — Standard MQTT ports. Always check your provider’s console for the exact numbers.

3. Choosing a broker

There are three broad ways to get a broker: a managed **cloud** service, a **self-hosted** broker on your own hardware, or a **public test** broker. The table compares the options most suited to small projects. US-dollar prices have been converted at roughly A$1.44 = US$1 (October 2026) and rounded.

Option	Free allowance	Paid / beyond free	Best for
HiveMQ Cloud Serverless	Up to 100 connections, about 10 GB traffic per month, no credit card	Starter: about A$0.49 per hour plus A$1.15 per million messages	First projects, classrooms, small farms
EMQX Serverless	1 million session minutes, 1 GB traffic and 1 million rule actions per month	About A$2.90 per million session minutes, A$0.22 per GB	Pay-as-you-grow fleets; set a A$0 spend limit to stay free
Adafruit IO	Free plan, 30 data points per minute	IO+ subscription for higher limits	Quick dashboards with no extra software
Mosquitto (self-hosted)	Free, open source (v2.0.22, July 2025)	Cost of a Raspberry Pi and power	Off-grid sites, privacy, no subscription lock-in
test.mosquitto.org	Free, no sign-up	—	Five-minute experiments only. Everything is public

Table 5 — Broker options (allowances checked October 2026; providers change them, so confirm before you commit).

Which should I pick?

- **Just learning, or a handful of devices on different sites:** HiveMQ Cloud Serverless (Section 4). Free, nothing to maintain, reachable from anywhere.
- **A small fleet that may grow, or you want per-minute billing with a hard spend cap:** EMQX Serverless (Section 5).
- **Everything on one property, poor internet, or you want full control:** Mosquitto on a Raspberry Pi (Section 6). It keeps working when the internet drops.
- **Both:** run Mosquitto locally and bridge selected topics to a cloud broker. This is an advanced set-up and is not covered here.

> **Public brokers are public.** Anyone can read and write every topic on test.mosquitto.org and similar servers. Never use them for anything that switches real equipment or carries personal data.

4. Option A — HiveMQ Cloud Serverless

HiveMQ Cloud is a fully managed broker. The Serverless plan is free for small projects and only accepts encrypted (TLS) connections, which is exactly what you want for devices on the open internet.

4.1 Create the cluster

1. Go to **hivemq.com** and choose **Sign up for free** under HiveMQ Cloud. Sign up with an email address or a Google / GitHub account.
2. When asked to pick a plan, choose **Serverless** (the free option). A cluster is created in a minute or two.
3. Open the cluster's **Overview** tab and copy the **Cluster URL** (it looks like `abc123def456.s1.eu.hivemq.cloud`). Note the **TLS MQTT port, 8883**, and the WebSocket port, **8884**.

4.2 Create device credentials

1. Open the **Access Management** tab and choose **Credentials**.
2. Create a username for each device, for example `apdl-device01`, and a long random password. A password manager is the easiest way to generate one.
3. Set the permission to **Publish and Subscribe**. On the Serverless plan each credential has one permission that applies to all topics.
4. Create a second credential, such as `dashboard`, for MQTT Explorer or Node-RED, so you can revoke it separately.

4.3 Test with the web client

1. Open the **Web Client** tab, enter the `dashboard` credentials and select **Connect**.
2. Subscribe to `apdl/#`.
3. Publish `hello` to `apdl/test`. It should appear in the message list within a second. If it does, your broker is ready.

> **Write it down.** Copy the cluster URL, port and credentials straight into the `secrets.py` file from Section 8.2 now, while they are on screen.

5. Option B — EMQX Serverless

EMQX Serverless is a pay-as-you-go broker with a generous monthly free quota. A **session minute** is one device connected for one minute, so a single always-on device uses about 44,000 per month — a million minutes covers roughly 22 devices running around the clock.

1. Sign up at **emqx.com** and create a **Serverless** deployment. Choose the region closest to Australia that is offered.
2. Under **Billing › Spend limit**, set the limit to **0**. The deployment then stops at the free quota instead of charging you.
3. From the deployment **Overview**, copy the **Connection address** and note the **MQTT over TLS port, 8883**.
4. Download the **CA certificate** (`emqxsl-ca.crt`) from the Overview page. You only need it if you turn on certificate verification (Section 8.5).
5. Open **Access Control › Client Authentication** and add a username and password for each device.
6. Test with the online MQTT client linked from the console, or with MQTT Explorer as described in Section 7.

> **SNI is required.** Serverless brokers share an address between many customers and use the hostname your device sends (SNI) to find your deployment. The MicroPython code in this guide sends it automatically. If you see “connection reset” immediately after connecting, an old MicroPython version that does not send SNI is the usual cause.

6. Option C — Mosquitto on a Raspberry Pi

Eclipse Mosquitto is the most widely used open-source broker. On a Raspberry Pi it uses a few megabytes of RAM and will comfortably handle hundreds of devices. Because it lives on your own network, it keeps working when the internet is down — important on farms and remote sites.

6.1 Install

Start from a Raspberry Pi running Raspberry Pi OS (Bookworm or newer) with a fixed IP address, either reserved in your router or set on the Pi. Open a terminal and run:

```
sudo apt update
sudo apt install -y mosquitto mosquitto-clients
sudo systemctl enable --now mosquitto
mosquitto -h | head -n 1          # shows the installed version
```

6.2 Add users

Since version 2.0, Mosquitto refuses connections from other machines until you configure a listener and decide how clients authenticate. Create a password file:

```
# create the password file and first user (you will be asked for a password)
sudo mosquitto_passwd -c /etc/mosquitto/passwd apdl-device01
# add more users WITHOUT -c (the -c flag overwrites the file)
sudo mosquitto_passwd /etc/mosquitto/passwd dashboard
sudo chown root:mosquitto /etc/mosquitto/passwd
sudo chmod 0640 /etc/mosquitto/passwd
```

6.3 Configure the listener

Create a new file with `sudo nano /etc/mosquitto/conf.d/apdl.conf`, add the lines below, save, then run `sudo systemctl restart mosquitto`.

```
# /etc/mosquitto/conf.d/apdl.conf
listener 1883
allow_anonymous false
password_file /etc/mosquitto/passwd
```

Check the log with `sudo journalctl -u mosquitto -n 20` if the service fails to start; a typo in the file is the usual cause.

6.4 Restrict topics (recommended)

An access-control list (ACL) stops a compromised device from switching someone else's pump. With the file below, each device can only use the branch named after its own username, while the dashboard can see everything.

```
# /etc/mosquitto/acl  (add "acl_file /etc/mosquitto/acl" to apdl.conf)
user dashboard
topic readwrite apdl/#

# each device may only touch its own branch; %u = its username
pattern readwrite apdl/demo/%u/#
```

6.5 Test it

```
# terminal 1 - subscribe to everything under apdl/
mosquitto_sub -h localhost -u dashboard -P 'YourPassword' -t 'apdl/#' -v

# terminal 2 - publish a test message
mosquitto_pub -h localhost -u dashboard -P 'YourPassword' \
  -t 'apdl/test/hello' -m 'Hello from the Pi'

# cloud broker test from a PC (TLS on 8883)
mosquitto_sub -h abc123def456.s1.eu.hivemq.cloud -p 8883 \
  -u apdl-device01 -P 'YourPassword' -t 'apdl/#' -v \
  --capath /etc/ssl/certs
```

Do not port-forward 1883 to the internet. If you need to reach a Pi broker from outside, use a VPN such as Tailscale or WireGuard, or bridge to a cloud broker. Exposing an unencrypted broker invites strangers to read and control your equipment.

7. Test from your computer

Before involving a microcontroller, prove the broker works from a PC. **MQTT Explorer** (mqtt-explorer.com, free for Windows, macOS and Linux) shows every topic as a live tree and lets you publish test messages.

1. Install and open MQTT Explorer, then select **+** to add a connection.
2. Enter the host (cluster URL or the Pi's IP address), port **8883** with **Encryption (TLS)** switched on for cloud brokers, or port **1883** with it off for a local Pi.
3. Enter the `dashboard` username and password and select **Connect**.
4. Publish `hello` to `apdl/test`. It should appear in the topic tree immediately.

Keep MQTT Explorer open while you work through Section 8. Seeing your device's messages arrive (or not) is the fastest way to find a problem.

8. Connect a MicroPython device with Thonny

The program below connects an ESP32 or Pico W to any of the brokers above. It publishes a JSON reading every 10 seconds, listens for ON / OFF commands to drive the on-board LED, announces itself with a retained `online` status, and uses a Last Will so the broker marks it `offline` if it loses power. It reconnects automatically after Wi-Fi or broker drop-outs.

8.1 Install the MQTT library

MicroPython ships without an MQTT client, so install `umqtt.simple` from micropython-lib. Use either method:

- **Thonny:** Tools › Manage packages…, search for `umqtt.simple`, select the micropython-lib result and choose **Install**. Thonny copies it to the board's `/lib` folder.
- **On the board:** create `secrets.py` first (Section 8.2), then run the lines below in the Thonny shell. The board downloads the library itself over Wi-Fi.

```
import network, mip, secrets
w = network.WLAN(network.STA_IF); w.active(True)
w.connect(secrets.WIFI_SSID, secrets.WIFI_PASSWORD)
# wait a few seconds until w.isconnected() returns True, then:
mip.install("umqtt.simple")
```

8.2 Store your settings in secrets.py

In Thonny choose **File › New**, paste the code below, fill in your own values and save it to the device as `secrets.py`. Keeping credentials in a separate file means you can share `main.py` without giving away passwords.

```
# secrets.py - keep this file off GitHub and out of shared photos
WIFI_SSID = "YourNetwork"
WIFI_PASSWORD = "YourWiFiPassword"

# Cloud broker (HiveMQ / EMQX): TLS on port 8883
MQTT_HOST = "abc123def456.s1.eu.hivemq.cloud"
MQTT_PORT = 8883
MQTT_TLS = True
MQTT_USER = "apdl-device01"
MQTT_PASSWORD = "a-long-random-password"

# Local Mosquitto on a Raspberry Pi instead? Use these values:
# MQTT_HOST = "192.168.1.50"
# MQTT_PORT = 1883
# MQTT_TLS = False
```

8.3 The main program

Create another new file, paste the program below and save it to the device as `main.py`. MicroPython runs `main.py` automatically at power-up.

```
# main.py - AP Design Labs MQTT starter for ESP32 / Pico W
# MicroPython 1.23 or newer, written for Thonny
import sys, time, json, ssl, network, machine, ubinascii
from umqtt.simple import MQTTClient
import secrets

PUBLISH_MS = 10_000                       # send telemetry every 10 s
CLIENT_ID = b"apdl-" + ubinascii.hexlify(machine.unique_id())
BASE = b"apdl/demo/" + CLIENT_ID          # e.g. apdl/demo/apdl-e66164...
T_STATUS = BASE + b"/status"
T_TELEMETRY = BASE + b"/telemetry"
T_LED = BASE + b"/led/set"

# On-board LED: "LED" on Pico W, GPIO 2 on most ESP32 dev boards
led = machine.Pin("LED" if sys.platform == "rp2" else 2, machine.Pin.OUT)


def wifi_connect():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        wlan.connect(secrets.WIFI_SSID, secrets.WIFI_PASSWORD)
        for _ in range(30):
            if wlan.isconnected():
                break
            time.sleep(1)
```

```
    if not wlan.isconnected():
        raise OSError("Wi-Fi connection failed")
    print("Wi-Fi OK, IP", wlan.ifconfig()[0])


def tls_context():
    ctx = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
    ctx.verify_mode = ssl.CERT_NONE       # encrypted; see 8.5 to verify
    return ctx


def on_message(topic, msg):
    print("RX", topic, msg)
    if topic == T_LED:
        led.value(1 if msg.lower() in (b"1", b"on", b"true") else 0)


def mqtt_connect():
    client = MQTTClient(CLIENT_ID, secrets.MQTT_HOST,
                        port=secrets.MQTT_PORT,
                        user=secrets.MQTT_USER,
                        password=secrets.MQTT_PASSWORD,
                        keepalive=60,
                        ssl=tls_context() if secrets.MQTT_TLS else False)
    # Last Will: broker publishes "offline" if we vanish unexpectedly
    client.set_last_will(T_STATUS, b"offline", retain=True, qos=1)
    client.set_callback(on_message)
    client.connect()
    client.publish(T_STATUS, b"online", retain=True, qos=1)
    client.subscribe(T_LED, qos=1)
    print("MQTT connected to", secrets.MQTT_HOST)
    return client


def read_sensor():
    if sys.platform == "rp2":             # Pico W internal temperature
        volts = machine.ADC(4).read_u16() * 3.3 / 65535
        return round(27 - (volts - 0.706) / 0.001721, 1)
    return None                           # ESP32: read your own sensor here


def run():
    client = None
    last = time.ticks_add(time.ticks_ms(), -PUBLISH_MS)
    while True:
        try:
            if client is None:
                wifi_connect()
                client = mqtt_connect()
            client.check_msg()            # handle any incoming commands
            if time.ticks_diff(time.ticks_ms(), last) >= PUBLISH_MS:
                last = time.ticks_ms()
                payload = json.dumps({"temp_c": read_sensor(),
                                      "uptime_s": time.ticks_ms() // 1000})
                client.publish(T_TELEMETRY, payload)
                print("TX", payload)
            time.sleep_ms(100)
        except OSError as err:
            print("Connection problem:", err, "- retrying in 5 s")
            try:
```

```
                client.disconnect()
            except Exception:
                pass
            client = None
            time.sleep(5)


run()
```

8.4 Run and check

1. With `main.py` open in Thonny, press **F5** (Run). The shell should print `Wi-Fi OK`, then `MQTT connected`, then a `TX` line every 10 seconds.
2. In MQTT Explorer, find `apdl/demo/apdl-…/telemetry`. A new reading should arrive every 10 seconds, and `status` should read `online`.
3. Publish `ON` to `apdl/demo/apdl-…/led/set` (copy the exact topic from MQTT Explorer). The on-board LED lights; publish `OFF` to turn it off.
4. Unplug the board. Within about 90 seconds (1.5 × the keep-alive) the `status` topic changes to `offline` — that is the Last Will at work.
5. Press **Stop** in Thonny before editing files, then unplug and re-power the board to run it without a computer attached.

Using the Mosquitto ACL from Section 6.4? The ACL only lets a device use the branch named after its username, so change `BASE` to `b"apdl/demo/" + secrets.MQTT_USER.encode()` to match.

8.5 Verifying the broker's certificate

With `CERT_NONE` the connection is encrypted, but the device does not prove it is talking to the genuine broker. That is fine for learning. For equipment that matters, turn on verification: copy the broker's root CA certificate to the board as `ca.der` and replace `tls_context()` with the version below.

```
import ntptime

def tls_context():
    ntptime.settime()                     # certificates need a real clock
    ctx = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
    ctx.verify_mode = ssl.CERT_REQUIRED
    ctx.load_verify_locations(cafile="ca.der")   # broker's root CA
    return ctx
```

- **HiveMQ Cloud** uses Let's Encrypt certificates. Download the **ISRG Root X1** certificate from letsencrypt.org/certificates in DER format.
- **EMQX Serverless:** convert the downloaded `emqxsl-ca.crt` with the OpenSSL command below.
- The device clock must be correct for certificate dates to check out, which is why the code calls `ntptime.settime()` first.

```
openssl x509 -in emqxsl-ca.crt -outform DER -out ca.der
```

8.6 Good habits for battery and low-power nodes

- umqtt.simple does not send keep-alive pings on its own. If you publish less often than the keep-alive (60 s here), call `client.ping()` periodically or raise the keep-alive.
- For battery sensors that deep-sleep, connect, publish with `retain=True`, disconnect and sleep. Retained values mean dashboards always show the latest reading.
- Every connection on EMQX Serverless counts session minutes, so short wake-publish-sleep cycles are also the cheapest pattern there.

9. Designing topics and payloads

A little planning now saves a lot of renaming later. Use the pattern **owner / site / device / measurement**, all lower case, no spaces, and no leading slash.

Topic	Purpose
`apdl/farm/tank1/level`	Sensor value published by the tank node
`apdl/farm/tank1/status`	Retained `online` / `offline` with Last Will
`apdl/farm/pump1/set`	Command topic the pump node subscribes to
`apdl/farm/pump1/state`	Retained state the pump reports back after acting
`apdl/farm/#`	Dashboard subscription for the whole site

Table 6 — An example topic tree for a farm installation.

- Separate **commands** (`…/set`) from **state** (`…/state`). The device confirms what it actually did, so the dashboard never shows a pump as ON when the command was lost.
- Send JSON when a reading has several values, e.g. `{"temp_c": 21.4, "rh": 55}`, and a plain value when it has one.
- Include units in the key name (`temp_c`, `level_mm`) so nobody has to guess later.

10. Security checklist

- Use TLS (port 8883) for anything that crosses the internet.
- Give every device its own username and password, so one can be revoked without touching the rest.
- Keep `secrets.py` out of source control and screenshots.
- On Mosquitto, set `allow_anonymous false` and use an ACL so devices can only reach their own topics.
- Design commands so that repeating them is harmless (QoS 1 can deliver twice).
- Never port-forward an unencrypted broker; use a VPN or a cloud bridge instead.
- Turn on certificate verification (Section 8.5) for equipment that switches pumps, gates or power.
- Set a spend limit or billing alert on any pay-as-you-go cloud account.

11. Troubleshooting

Symptom	Likely cause and fix
`ImportError: no module named 'umqtt'`	Library not installed. Repeat Section 8.1 and check `/lib/umqtt` exists on the board.

Symptom	Likely cause and fix
`Wi-Fi connection failed`	Wrong SSID or password, or a 5 GHz-only network. ESP32 and Pico W need 2.4 GHz.
`OSError: -2` or `-202`	The broker hostname could not be found. Check `MQTT_HOST` for typos and remove any `mqtts://` prefix.
`OSError: 104` (connection reset) straight after connecting	TLS problem: wrong port, TLS off for a cloud broker, or an old MicroPython that does not send SNI. Update to 1.23 or newer.
`MQTTException: 4` or `5`	Broker rejected the login (4 = bad username or password, 5 = not authorised). Re-check credentials and permissions.
Connects, but no messages arrive	Topic mismatch. Topics are case-sensitive; watch `#` in MQTT Explorer to see what is really being sent.
Device keeps disconnecting every minute	Nothing sent within the keep-alive. Publish more often, call `client.ping()`, or increase `keepalive`.
Two devices keep kicking each other off	Duplicate client IDs. The code builds a unique ID from `machine.unique_id()`; do not hard-code it.
Mosquitto: "Connection refused" from another PC	No listener or no password file configured. Revisit Sections 6.2 and 6.3 and restart the service.

Table 7 — Common problems.

12. Next steps

- **Dashboards:** Node-RED and Home Assistant both have built-in MQTT support and can subscribe to `apdl/#` with the dashboard credentials.
- **Long-range nodes:** pair this guide with a LoRa gateway that republishes readings to MQTT for paddocks beyond Wi-Fi range.
- **Data logging:** feed MQTT into InfluxDB and Grafana, or use the rule engine in EMQX, to keep a history of every reading.
- **Robust reconnects:** micropython-lib also provides `umqtt.robust`, a drop-in wrapper that retries automatically.

Sources (checked October 2026): HiveMQ pricing (hivemq.com/pricing/fully-managed); HiveMQ blog "Serverless vs Starter" (2026); FreeTier.co HiveMQ Cloud listing (June 2026); EMQX Cloud pricing and "Connect with ESP32 and MicroPython" guides (docs.emqx.com); Adafruit IO MQTT API (io.adafruit.com/api/docs/mqtt); Mosquitto authentication methods and release notes (mosquitto.org); micropython-lib umqtt.simple source (github.com/micropython/micropython-lib); HiveMQ community forum, MicroPython TLS thread (January 2025). Exchange rate A$1 = US$0.693 (exchange-rates.org, 1 October 2026). Pricing and free allowances change without notice; confirm current terms with each provider. Provided free in the AP Design Labs Free Library for personal and educational use, without warranty. Product names are trademarks of their respective owners.