### **10.3 Implementation**

* Adding the ‘PubSubClient’ library to implement the MQTT to publish and subscribe to a client. (Nick O'Leary 2020)
* Adding the ‘Wi-FiSSLClient’ Library that supports encrypted communication between devices when connecting to the server. (Wolfssl 2025)

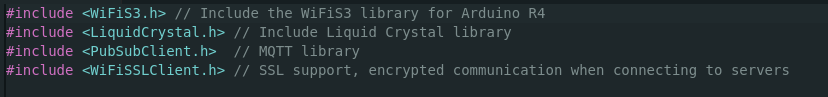


Figure 12

* Adding each credential for the connection of the HiveMQ cloud cluster

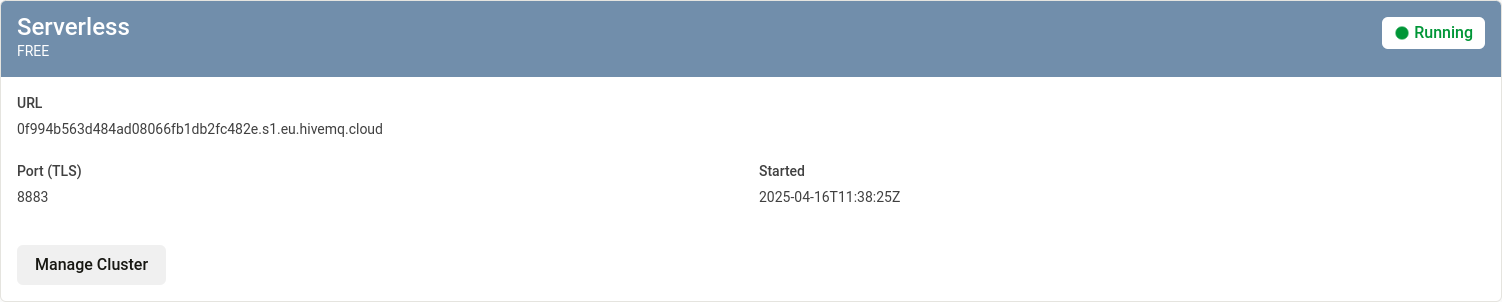


Figure 13 (HiveMQ 2025)

* Making each credential a constant character when the program is running and the MQTT port to always be on port 8883.



Figure 14

Picture

Figure 15

* Provides a secure WiFi Client connection with TLS which the HiveMQ port 8883 requires
* Pass the WiFi client into the MQTT client enabling the client to communicate over WiFi

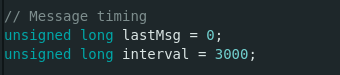


Figure 16

* An error occurred through the code where the client was sending to many requests via the same client Id
* To stop this a message timer was implemented for each message sent by MQTT and is delayed by 3 second intervals

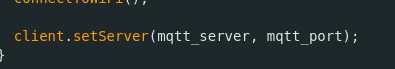


Figure 17

* Initialise the server to client connection, by passing through the MQTT server and port number or the client to connect to the server



Figure 18

* To keep the MQTT running, the variable named ‘reconnectMQTT’ if the client is connected and this is looped until the connection is disconnected or timed out

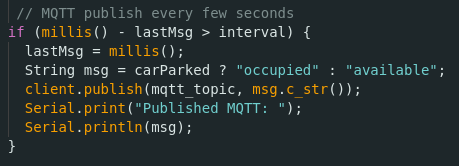


Figure 19

* Previously I created the variables ‘LastMsg’ and ‘interval’ which are used in the if statement to publish MQTT messages every few seconds to hinder frequent publishes which could cause timeouts and show the user if the space is full or available after every interval. (UKHeliBob 2017)

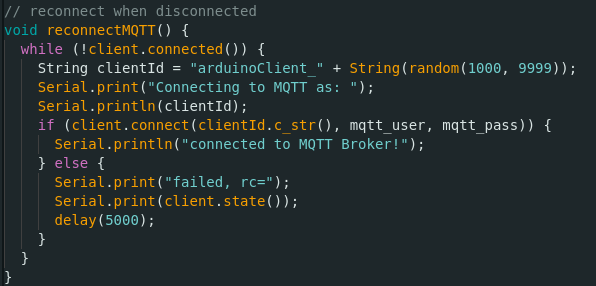


Figure 20

* Previously the ‘ReconnectMQTT’ variable was stated it is called at the end of my code where it has been configured to randomise the client's name on every publish and subscribe message, so the broker doesn't get overloaded. When relogging into the broker with the correct credentials it will present a successful connection if not it will say fail and try to reconnect every 5 seconds.

## **10.4 Testing the MQTT Broker connection**

In Section 10.3, Figure 14, I configured the MQTT broker credentials within the Arduino codebase, enabling the Arduino to establish a connection with the broker. This setup allowed real-time monitoring of the sensor data from the ultrasonic sensor when tested on the vehicle to be published to the broker. I used **MQTT-Explorer** as the MQTT client to subscribe to the sensor data and monitor the published messages. (Thomas Nordquist 2025)

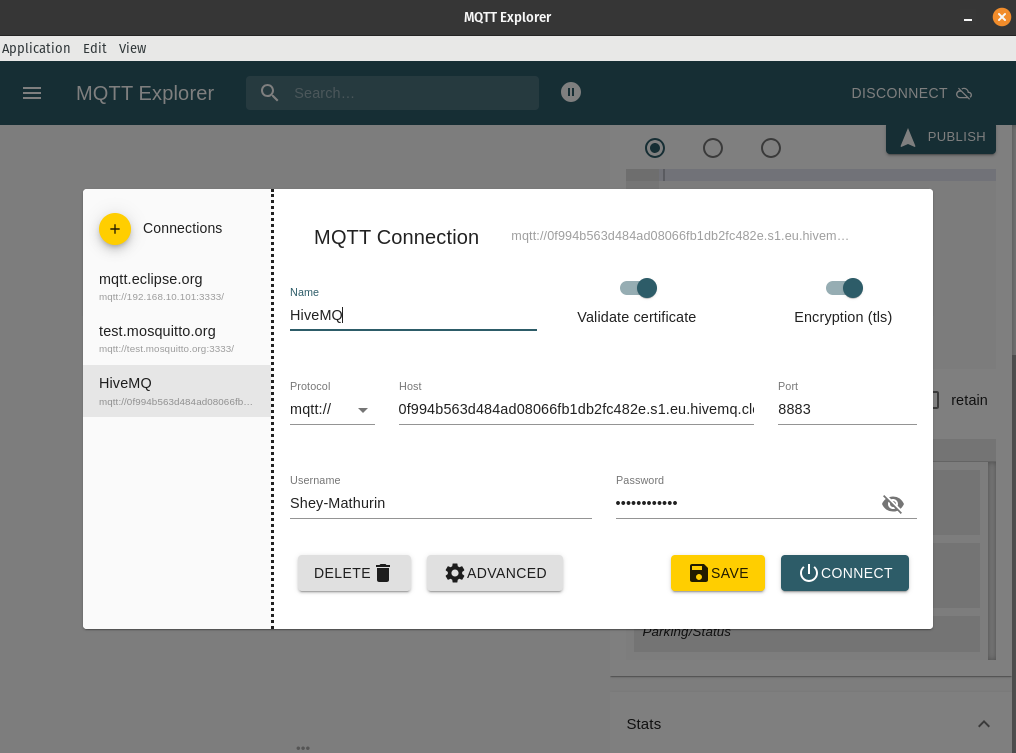


Figure 21

* This figure shows the MQTT client application MQTT-Explorer, by adding a ‘new connection’ located on the left side, the Hive MQTT Broker credentials are typed where specified, to monitor or subscribe to sensor data. In MQTT-Explorer, TLS (Transport Layer Security) must be enabled to ensure the connection between the client and the broker is secure encrypting any data transferred over the network. The Certificate validation checks if the server is trustworthy to verify and accept the digital certificate when logging in with the username and password as this is a serverless setup.



Figure 22

* When entering the MQTT-Explorer application, once logged in, the logs can be seen in the command line. As this is the Snap package version of MQTT-explorer, it acts as an application, so all logs can be seen in the command line as seen in the figure above. It shows the certificate being successfully validated and all the details inputted before specifying the MQTT protocol.

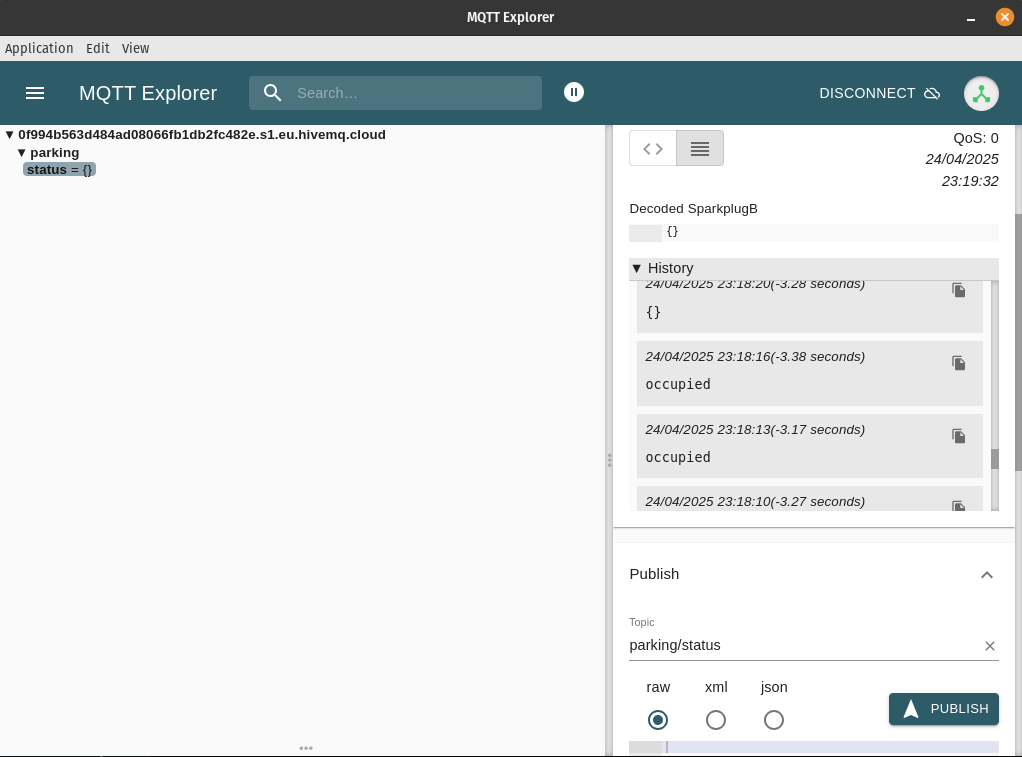


Figure 23

* For MQTT-Explorer to gain access to the sensor data it needs to be subscribed to a topic. In this case, it was ‘parking/status’ when this is inputted the status of whether a car is parked or not is not parked is stored in ‘parking’. When the vehicle is parked it will show ‘occupied’, when the vehicle is not present it will show ‘{}’ meaning the space is empty which is shown in the history logs in the figure above.