BLOG

CAS DATALOGGERS

What Is MQTT?

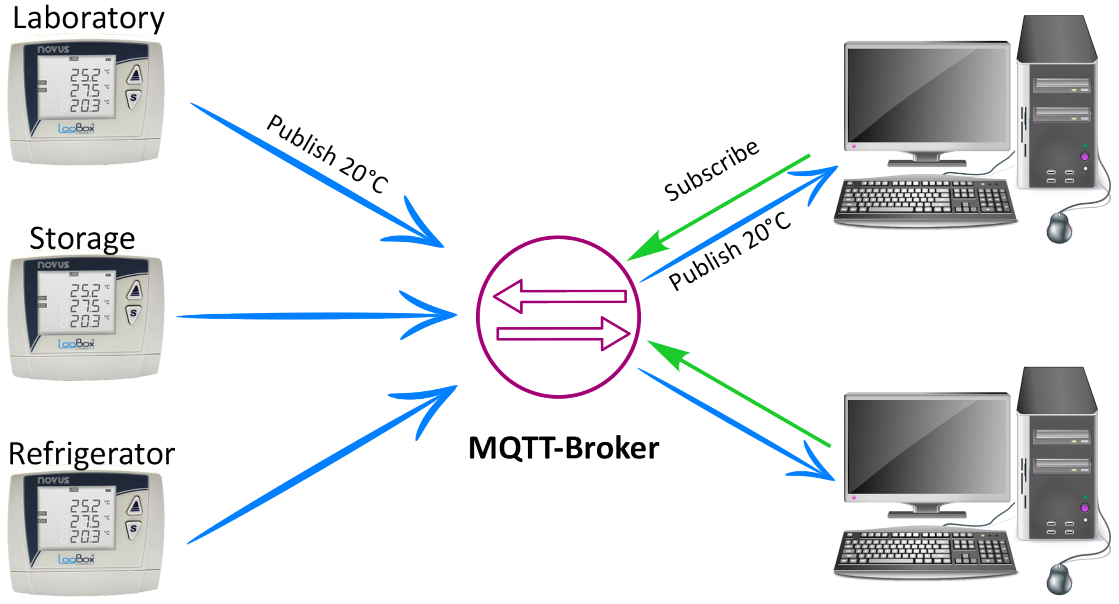
Over the last 2 years, we have seen several of our equipment manufacturers start providing support for the MQTT protocol for sending measurements from remote devices. As this is a relatively new standard, you might be wondering what it's about and how it works. In this post, I'll give you a little background on MQTT, why you might want to use it, and how it works with a small example.

What Is MQTT?

MQTT was designed as a method or protocol for sending messages (data) between pieces of equipment or machines. The name comes from Message Queue Telemetry Transport but since 2013 it has just been MQTT. It was designed for use in sending data from equipment in remote locations where there were limited resources, think battery-powered equipment with limited communications bandwidth and limited computing resources. In this sense, it is a lightweight protocol that doesn't require a lot of overhead and is efficient in using the available bandwidth. It runs on top of a standard data transport mechanism like TCP/IP that offers bidirectional communication without data loss and ordered delivery. MQTT is now a standard managed by the OASIS organization with the latest revision, Version 5 released in 2019.

Basic MQTT Architecture

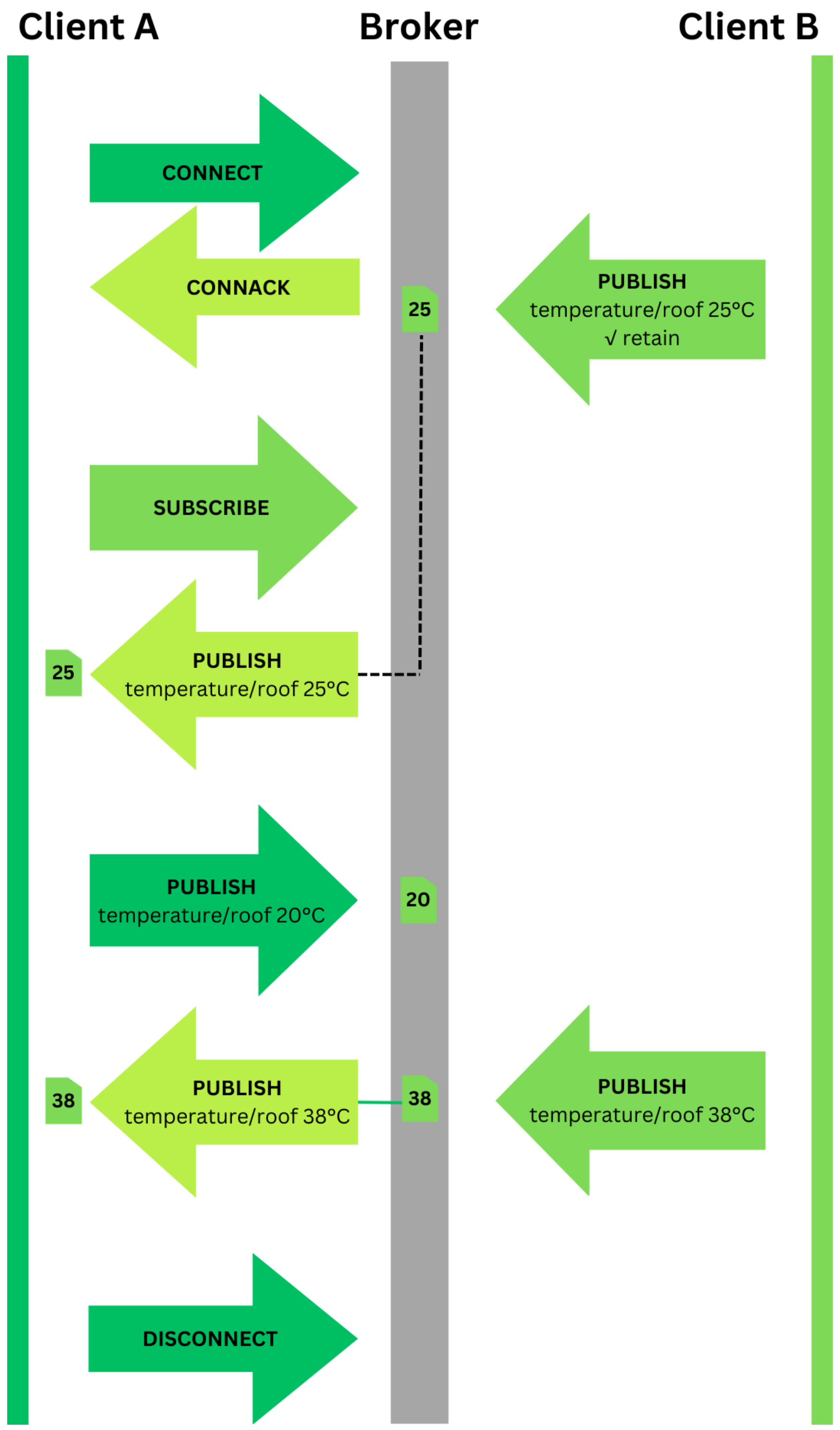
Central to MQTT is the concept of a message broker. The broker is a server that is running software that acts as an intermediary to receive messages from clients and then route or publish these messages out to other clients. A client can be a publisher which is a producer of data or a subscriber which is a consumer of data.



An important part of this architecture is the publisher has no knowledge of the number or location of any subscribers; it only talks to the broker. Likewise, the subscribers don't need to know anything about the publisher, they only need a connection to the broker. To enable the exchange of specific information, whenever a client published data, it includes a topic to identify the data. This topic is the key that clients use when subscribing to messages from certain devices. So, you can see how the role of the broker is key to managing the flow of data from publishers to one or more subscribers.

The MQTT standard defines 14 standard message types to allow clients to connect and disconnect, publish data, acknowledge the receipt of data, and manage the connections. As mentioned before, MQTT relies on the standard TCP/IP protocol to handle moving data. This protocol does not have any inherent built-in security. However, MQTT can be configured to use TLS to provide a secure connection using a certificate file, user, and password to allow data to be encrypted to prevent modification or unauthorized access. Finally, MQTT includes an optional Quality of Service (QoS) capability to ensure that important messages are properly delivered.

Typical Message Exchange



Why Use MQTT?

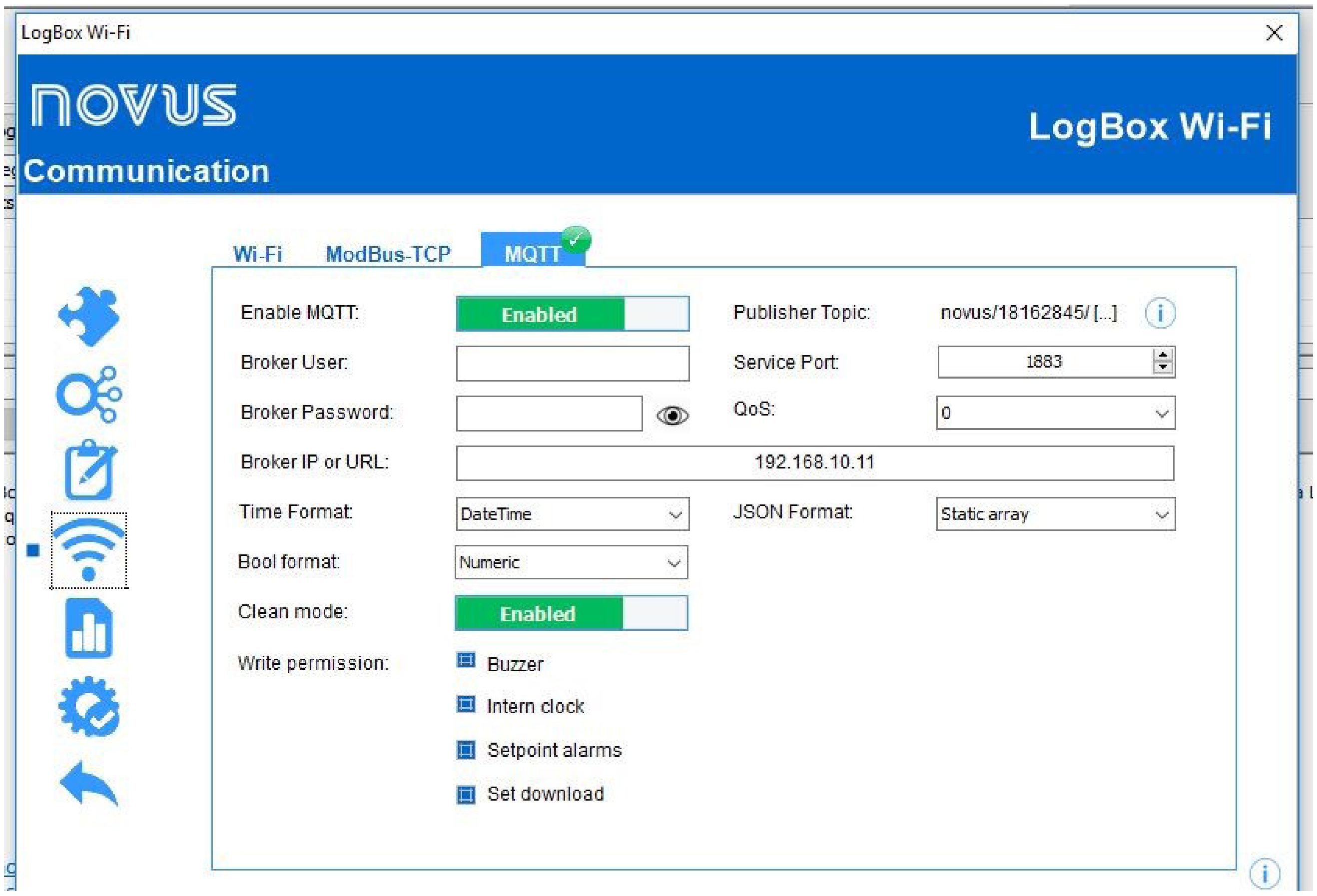
We’ve recently started using MQTT in a few simple applications where it made sense. It is useful in projects where you are looking to see data from a remote location that is updated every few minutes. This can be quite different from a more traditional data logging application where you might configure a remote device to record data every few minutes but store it in internal memory and only upload it once a day. Being a lightweight protocol, MQTT has an advantage over traditional protocols like FTP when sending small amounts of data frequently.
There is much less overhead in establishing the connection and it is much simpler to send a few data values as opposed to sending a whole file every time. For a remote, battery-powered device using a cellular modem, this translates into having to power the modem for a much shorter amount of time and in turn, much better battery life.

On the other hand, MQTT is not good for trying to continuously stream large amounts of the additional work that has to be done by the broker in processing each message. Also, the message format does not lend itself well to transferring large blocks of data, it is designed around sending just a few values for each topic.

Two Simple MQTT Applications

A typical MQTT application could be as simple as sending real-time data from a humidity transmitter in a lab or storage area within a facility to a PC in the same office to monitor and archive the data. We built a sample project using a Novus LogBox Wi-Fi data logger and an older PC running the open-source Mosquitto broker software.

The LogBox is very easy to configure to send data via MQTT using the NXperience configuration utility. The screen capture below shows the communications settings to talk to the broker software running on the PC with IP address 192.168.10.11.



LogBox Wi-Fi

novus

Communication

LogBox Wi-Fi

Wi-Fi ModBus-TCP MQTT

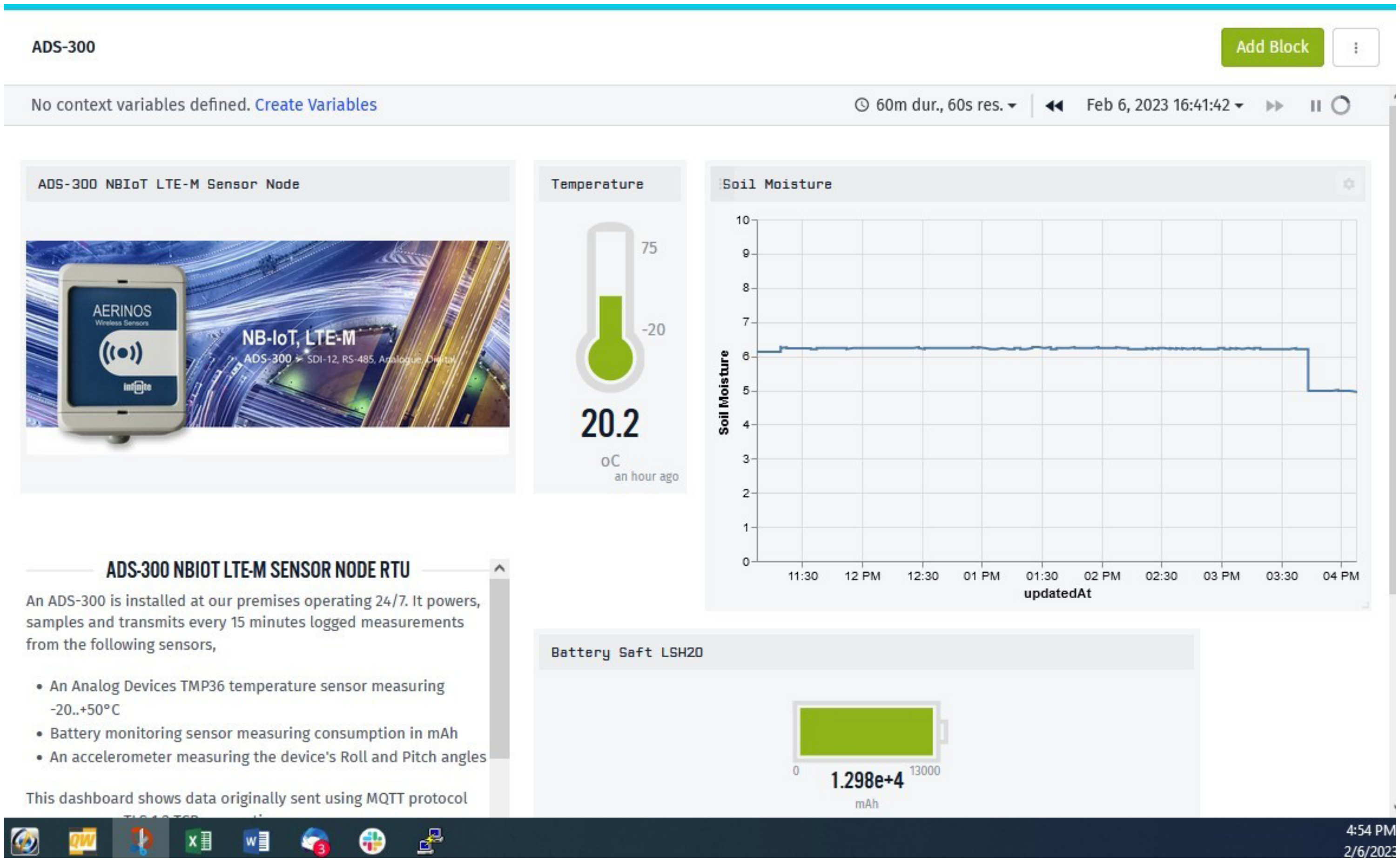
Enable MQTT:	Enabled	Publisher Topic:	novus/18162845/ [...]
Broker User:		Service Port:	1883
Broker Password:		QoS:	0
Broker IP or URL:	192.168.10.11		
Time Format:	DateTime	JSON Format:	Static array
Bool format:	Numeric		
Clean mode:	Enabled		
Write permission:	☑ Buzzer ☑ Intern clock ☑ Setpoint alarms ☑ Set download		

An important piece of information is the value of the topic shown in the upper right corner which is required when subscribing to receive the data. On the PC with the broker software, we also created a simple script in Python that would connect to the broker, subscribe to this topic, and grab, decode, and print the data.

AD300 transmits data via a 4G cellular connection to Losant with an application to display the dashboard

The second application is a remote soil monitoring project where the data is sent to a cloud-based software package to display and save the data. The data logger that was used is the Infinite ADS-300, a compact battery-powered data collector that includes a cellular NBIoT/CAT-M model to upload data. The ADS-300 was connected to a soil moisture sensor using an SDI-12 interface. Data was then uploaded to the Losant Enterprise IoT platform.

This platform includes a built-in MQTT broker plus easy-to-use tools to extract the data from the message, display it using various charts and dashboards and save it. Other typical platforms include services like Microsoft Azure and Amazon AWS both of which provide support for MQTT brokers. For this application, the dashboard includes a measurement of the current temperature, a graph of the soil moisture content, and a gauge to show how much battery capacity was left:



ADS-300 NBIOT LTE-M SENSOR NODE RTU

An ADS-300 is installed at our premises operating 24/7. It powers, samples and transmits every 15 minutes logged measurements from the following sensors,

- An Analog Devices TMP36 temperature sensor measuring -20..+50°C
- Battery monitoring sensor measuring consumption in mAh
- An accelerometer measuring the device's Roll and Pitch angles

This dashboard shows data originally sent using MQTT protocol

When building remote applications, data security is often a concern. Fortunately, the MQTT protocol allows the use of TLS to create a secure connection between the ADS-300 and the Losant server. A set of key and certificate files, one for the device and one for the Losant server were created during the configuration of the RTU and the Losant side of the connection. With this in place, any data transmitted between the two is encrypted to prevent eavesdropping or tampering.

Summary

MQTT is an established data exchange protocol with support showing up on more devices. As an alternative to the common “store data in memory and download” used by data loggers, it allows for more real-time information. MQTT uses a publish/subscribe architecture that relies on a broker software application to facilitate the exchange between the data sources or “publishers” and one or more data consumers or “subscribers” which monitor and display the values. The flexibility of this architecture allows it to be used locally within a facility or remotely between devices in the field and a cloud-based platform such as Azure, AWS, or Losant. Finally, MQTT provides support for encrypted connections using TLS security.