influxdata®

Working with MQTT and InfluxDB

AUGUST 2022 | REV 1

Working with MQTT and InfluxDB

Introduction

MQTT is a protocol for machine-to-machine communication based on a publish/subscribe model. This means IoT devices publish messages to a broker into a specific topic. The broker acts as an intermediary and lets other devices subscribe to topics to receive messages. Devices can publish messages, subscribe to topics, or both. This is powerful because it lets IoT devices communicate with each other and act on information from other devices quickly. Developers can define topics based on location, the type of data being collected, or any other relevant category. MQTT is lightweight and ideal for situations that require a small resource footprint.

Workers at IBM and Arcom developed MQTT in 1999 to retrieve data from industrial equipment in remote areas. IBM used MQTT internally for a decade before releasing MQTT 3.1 in 2010, which allowed public users to make their own implementations. It is very popular with IoT developers, who use it as the backbone for many open source brokers and client libraries. Since 2013, OASIS oversees the management of the MQTT protocol. It has become the default protocol in the IoT industry because of its ability to handle environments with intermittent network connectivity, hardware with low processing power, and constrained bandwidth.

MQTT exists to pass along data generated by IoT devices. Users leveraging that data still need a way to collect, store, and analyze it. This paper is an introduction to MQTT and includes the benefits and challenges of working with it, as well as some general use cases. It also includes information on InfluxData's MQTT Native Collector, which makes it easy to send data from an MQTT broker to InfluxDB Cloud.

MQTT

What is MQTT?

MQTT is a protocol that connects brokers and clients. A broker acts as a middle man for devices sending data. It lets each device establish one connection with the broker, rather than a new connection with every device it needs to communicate with. Client devices run MQTT client software in order to connect to a broker. They can both publish messages to the broker and subscribe to topics to receive messages. Every message sent to the broker needs a topic, which can be any string identifier that lets devices retrieve the information they need, such as location, data type, or device ID. Developers can add new topics to their application at any point. MQTT also supports wildcard operators for creating dynamic topics.

MQTT lets clients choose an Expiration Interval to tell the broker how long to hold a message before deleting it. It also has a Keep Alive function which lets a developer set a time interval to confirm the connection between a broker and a client. If a client doesn't send or receive messages during that interval, the broker will ping the client. If there is no response, it registers the client as disconnected.

Another MQTT feature is Retained Messages. Developers can flag messages for a broker to store so any client that subscribes to a new topic receives the newest retained message as an update, instead of waiting for devices to send new messages to the broker.

MQTT has three Quality of Service levels so developers can designate how thoroughly they want different kinds of messages delivered.

1. Messages can be sent "at most once" so that a broker sends messages to a subscribed client once it's connected, without informing the publishing client.
2. The level "at least once" means the message will definitely be delivered, but could be sent or delivered multiple times. Developers need to account for this in their applications so devices don't get confused by multiple copies of the same data.
3. Finally there's the level "exactly once" which guarantees delivery of the message and that clients receive only one copy of it. This is the most complex process and requires a four part series of responses between the broker and the subscribing client to check that it received exactly one copy of each message.

The Quality of Service feature lets developers set importance levels for different kinds of data so they can ensure delivery of important data, even in situations with intermittent connectivity. It also enables them to save bandwidth by not checking the delivery of unimportant data.

Some of the most popular MQTT broker implementations are Mosquitto, EMQX, and HiveMQ. There are also many MQTT client libraries so developers can work with it in the language that fits their application and that they're most comfortable with. The most popular client libraries are the open source Paho MQTT libraries, supported by the Eclipse Foundation.

MQTT use cases

Here are some examples of how different kinds of companies and developers use MQTT:

- **Consumer IoT** - This includes devices designed to automate processes for consumers, such as smart home devices. Consumer IoT devices use MQTT to communicate and increase efficiency in whatever they're designed to do. The broker model lets many devices communicate efficiently for near real-time actions.
- **Industrial IoT** - MQTT was initially created for IIoT use in the oil industry, and is still commonly used in manufacturing. It's more popular than other network protocols because it allows companies to send large volumes of data quickly.
- **Logistics** - MQTT can send data in close to real time, which makes it perfect for monitoring products as they're shipped around the world. It can handle intermittent internet connectivity which is also important for this use case.
- **Mobile Application Development** - Many mobile apps use MQTT, including Facebook Messenger. Mobile apps that include frequent network communication have similar workloads to IoT devices and MQTT is a natural fit. It provides apps with continuous connections in order to receive messages while expending minimal energy and bandwidth.

MQTT benefits

MQTT has a few key benefits for IoT use cases:

- **Efficiency** - MQTT is designed to send messages using the least amount of data and energy possible. It reuses connections between devices and a broker for multiple messages, which lets it use 10 times less bandwidth than HTTP. MQTT is also efficient because the messages it sends are small files and the publish/subscribe model lets devices receive messages directly from the broker rather than checking with other devices at set intervals.
- **Reliability** - MQTT's publish/subscribe architecture lets the broker store messages until devices subscribed to a topic connect and receive them. It also lets developers choose Quality of Service levels to designate certain data as critical, to prioritize the data they most need delivered.
- **Flexibility** - MQTT gives developers many options they can control, such as Quality of Service levels and Expiration Intervals. It also allows messages to contain any kind of data, from binary to ASCII text to anything else. The publish/subscribe architecture also enables faster application development because developers can simply add brokers to handle new devices or more frequent messages.

Challenges with using MQTT

MQTT is optimized for specific IoT use cases, and that comes with a few challenges. Developers need to be aware of its limitations and weigh them with the benefits to decide if MQTT is right for their application.

Latency

MQTT creates some latency by including a broker as a middle step between devices. For use cases that need extreme real-time results, MQTT might not be the best choice and developers might need a solution to pass data directly from device to destination.

Architecture complexity

Adding an MQTT broker into an application's architecture adds another service that developers need to manage and monitor. In use cases involving large data volume transfers, applications might require many broker instances which need to be balanced so the data load is equally assigned.

Resource requirements

If a developer is working with very low-powered devices, even MQTT might require too many resources. It's designed to be lightweight, but it does offer many features that are more resource intensive than alternatives optimized for extremely low-powered hardware.

Security

By default, MQTT doesn't have any security features or encryption. Developers can implement encryption using the TCP protocol MQTT is built on. Transport Layer Security (TLS) or Secure Sockets Layer (SSL) work well for encryption, but are another source of complexity for developers to implement and manage.

InfluxDB and MQTT

Many MQTT messages are time series data. Clients collect and publish metrics from sensors on IoT devices, and other clients subscribe to that data. In many IoT use cases that data needs to be analyzed and acted on very quickly. InfluxDB is built to handle these kinds of workloads and has features that help with analyzing and forecasting with time series data. Here are how some companies use InfluxDB with MQTT.

InfluxDB MQTT case studies

HiveMQ

HiveMQ is an MQTT broker messaging platform. It helps IoT and IIoT devices send data to the cloud and is built to simplify and streamline data collection. It has customers in fields such as logistics, transportation, and Industry 4.0, along with connected IoT products. Sensors generate large amounts of time series data, and HiveMQ is designed to get that data into the cloud quickly and reliably. It uses InfluxDB as part of an extension for the Sparkplug MQTT broker. HiveMQ customers collect Sparkplug metrics in InfluxDB to monitor their implementation. They can create dashboards within InfluxDB to view and analyze data from devices within their Sparkplug infrastructure.

HighByte

HighByte is an industrial software company that develops solutions to manufacturing problems focused on data architecture and integration. It created the first DataOps solution designed for the unique needs of systems at the edge. It works to fill in the gap between operational technology and informational technology, and to process and add context to industrial data so it's useful in an Industry 4.0 context. HighByte uses InfluxDB to store and analyze IIoT sensor data collected with MQTT, so their clients can get meaningful information from their data. HighByte also uses Flux in its Overall Equipment Effectiveness (OEE) calculations, which helps manufacturers optimize their processes.

MOXIE IoT

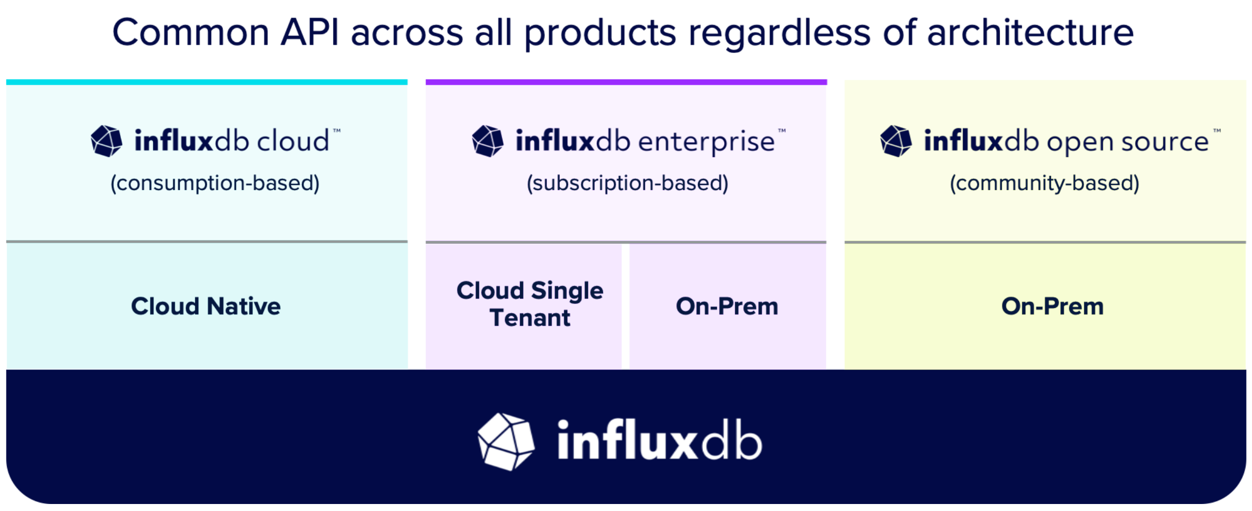
MOXIE IoT's Moxie World platform collects, stores, and analyzes IIoT data and creates real-time visualizations on its mobile app. It's designed as a tracking solution for assets such as cranes, forklifts, pallets, and more. The app stores data securely and displays charts and maps which show speed, direction, hours of usage, and other metrics. This gives industrial companies one source of truth they can use to monitor operations. Along with the app interface, Moxie World gives its customers access to live data through an MQTT stream and access to historical data with InfluxDB. It stores data from an MQTT broker in InfluxDB at regular intervals and uses Flux for queries. MOXIE IoT chose to use InfluxDB for their solution because it allowed them to store and query historical IoT data efficiently with built-in time series analysis tools.

InfluxDB’s MQTT Native Collector

InfluxDB’s MQTT Native Collector makes it simple to get data directly from MQTT brokers into InfluxDB Cloud. Developers can subscribe to supported brokers to collect and analyze data faster, without additional services or code needed to transport that data. Many MQTT brokers are cloud-based and the MQTT Native Collector allows users to keep their data completely in the cloud as it’s transported. It’s simple, fast, and reliable so developers can focus on working with the data they collect rather than building a complex system to get data into InfluxDB. It’s fully managed and eliminates the need for complex code, so developers can collect data from MQTT brokers without increasing the complexity of their application.

InfluxDB’s MQTT Native Collector highlights:

- **Simple**: It’s a straightforward process to configure by just setting the broker, topic, and parsing rules. You don’t need to manage complex code to retrieve the data you need.
- **Fully managed**: There’s no extra code required and it all takes place in the cloud. The data pipeline has the same security standards and reliability as InfluxDB Cloud.
- **Instantaneous**: You can start collecting data from brokers immediately and focus on analysis and building applications.
- **Cloud first:** You can keep all your systems running in the cloud. You don’t need to install additional software on a server to collect data and add in extra steps and potential bugs to data pipelines.



InfluxDB documentation, downloads & guide

[Get InfluxDB]

[Try InfluxDB Cloud for Free]

[Get documentation]

[Additional tech papers]

[Join the InfluxDB community]