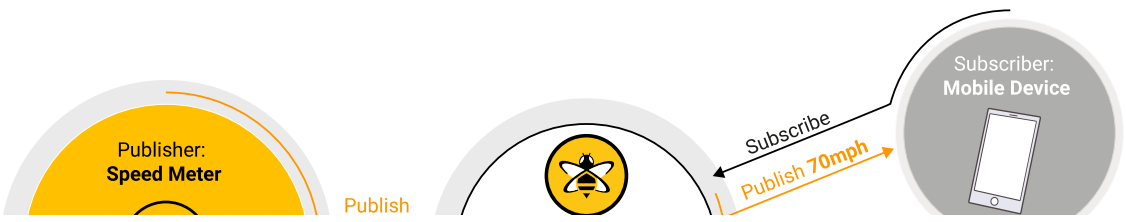
Welcome to Part 2 of **MQTT Essentials**, a blog series on the core features and concepts of the MQTT protocol. In this article, we will delve into the Pub/Sub architecture, also known as pub/sub, which is a messaging pattern in software architecture. It enables communication between different components or systems in a decoupled manner.

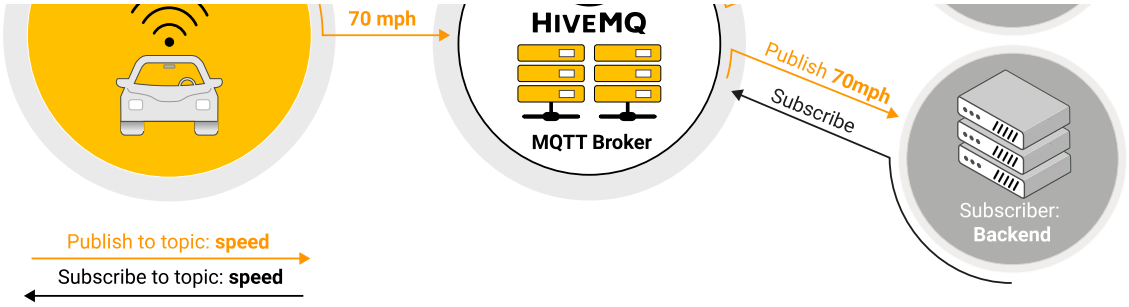
In Part 1 of this series **Introducing the MQTT Protocol**, we gave a basic overview of MQTT and its history. Now, let's explore the benefits of the Pub/Sub model for IoT applications, various message filtering techniques, and understand the distinction between MQTT, Pub/Sub, and Message Queues. To set the groundwork, we will start by clearly defining the publish/subscribe model.

MQTT: Publish/Subscribe (Pub/Sub) Architecture

In the Pub/Sub architecture, there are publishers that generate messages and subscribers that receive those messages. However, publish-subscribe is a broader concept that can be implemented using various protocols or technologies.

> **MQTT is one such specific messaging protocol that follows the publish-subscribe architecture.** MQTT uses a broker-based model where clients connect to a broker, and messages are published to topics. Subscribers can then subscribe to specific topics and receive the published messages.





Example of MQTT Publish / Subscribe Architecture

By clicking on the image, you interact with a video on YouTube.
Please read our **privacy policy page** to understand how we process data.

MQTT: Pub/Sub Decoupling Feature

The Pub/Sub architecture offers a unique alternative to traditional client-server (request-response) models. In the request-response approach, the client directly communicates with the server endpoint, creating a bottleneck that slows down performance. On the other hand, **the pub/sub model decouples the publisher of the message from the subscribers**. The publisher and subscriber are unaware that

the other exists. As a third component, a broker, handles the connection between them. This decoupling produces a faster and more efficient communication process.

By eliminating the need for direct communication between publishers and subscribers, pub/sub architecture removes the exchange of IP addresses and ports. It also provides decoupling, allowing operations on both components to continue communication uninterrupted during publishing or receiving. The pub/sub features three dimensions of decoupling for optimal efficiency:

- **Space decoupling**: Publisher and subscriber do not need to know each other (for example, no exchange of IP address and port).
- **Time decoupling**: Publisher and subscriber do not need to run at the same time.
- **Synchronization decoupling**: Operations on both components do not need to be interrupted during publishing or receiving.

> **Pub/Sub Decoupling in MQTT Protocol**
> MQTT decouples the publisher and subscriber spatially, meaning they only need to know the broker's hostname/IP and port to publish or receive messages. Additionally, MQTT decouples by time, allowing the broker to store messages for clients that are not online. Two conditions must be met to store messages: the client must have connected with a persistent session and subscribed to a topic with a Quality of Service greater than 0.

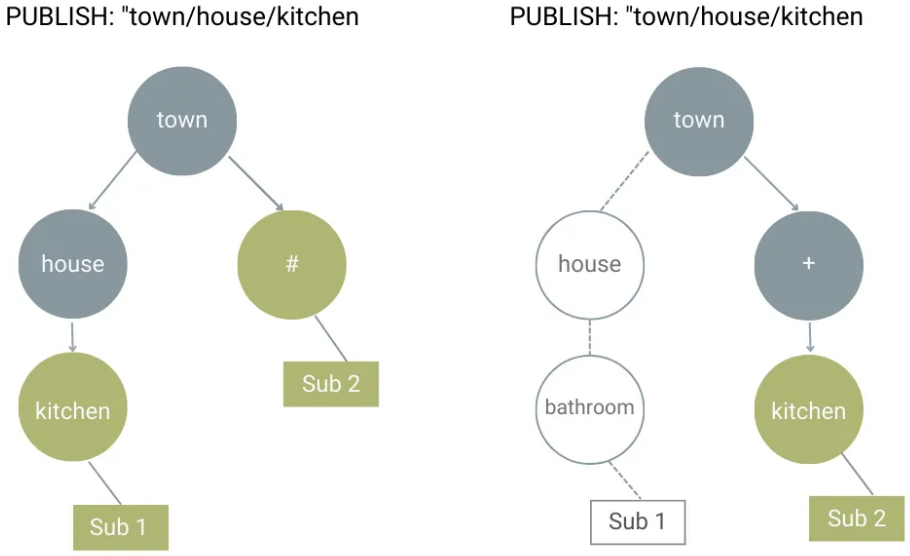
One of the most significant advantages of Pub/Sub software architecture is its ability to filter all incoming messages and distribute them to subscribers correctly, eliminating the need for the publisher and subscriber to know one another's existence.

MQTT: Pub/Sub Message Filtering Feature

Message filtering is a crucial aspect of the pub/sub architecture as it ensures subscribers only receive messages they are interested in. The pub/sub broker offers several filtering options, including subject-based filtering, content-based filtering, and type-based filtering.

Option 1: Subject-based filtering of Pub/Sub architecture

This is the most common filtering option, where the broker filters the messages based on the **MQTT topic** or subject. The subscribing clients indicate their interest by subscribing to specific topics, and the broker routes the messages to the appropriate subscribers based on the topic hierarchy. The topic structure is hierarchical, with levels separated by a forward slash (/), allowing subscribers to receive messages that match a specific topic level or a topic hierarchy. Here's an example of topic hierarchy.



Example of Topic Hierarchy

Benefits of Subject-based filtering in Pub/Sub:

- Simple and easy to use
- Flexible, allowing for a hierarchical topic structure
- Efficient, as it only forwards messages to subscribers interested in a particular topic

Drawbacks of Subject-based filtering in Pub/Sub:

- Publishers and subscribers need to agree on the topic hierarchy beforehand
- Limited to filtering messages based on topic hierarchy only

Use Case of Subject-based filtering in Pub/Sub:

Subject-based filtering is best suited for use cases where messages are organized into topics and subscribers are interested in a particular subset of those topics.

For example, in a smart home system, a subscriber may be interested in receiving updates about the temperature of a specific room. The subscriber would subscribe to a topic like `"smart-home/living-room/temperature,"` and the broker would only send messages that match this topic to the subscriber.

> **Message Filtering in MQTT**
> MQTT uses subject-based filtering of messages. Every message contains a topic (subject) that the broker can use to determine whether a subscribing client gets

the message or not. See **part 5 of MQTT Essentials** to learn more about the concept of topics. To handle the challenges of a pub/sub system, MQTT has three Quality of Service (QoS) levels. You can easily specify that a message gets successfully delivered from the client to the broker or from the broker to a client. However, there is the chance that nobody subscribes to the particular topic. If this is a problem, the broker must know how to handle the situation. For example, the **HiveMQ MQTT broker** has an extension system that can resolve such cases. You can have the broker take action or simply log every message into a database for historical analyses. To keep the hierarchical topic tree flexible, it is important to design the topic tree very carefully and leave room for future use cases. If you follow these strategies, MQTT is perfect for production setups.

Option 2: Content-based filtering of Pub/Sub architecture

In this type of filtering, the broker filters messages based on their content, which is specified using a filter expression. Subscribers indicate their interest by subscribing to a specific filter expression, and the broker routes the messages to the appropriate subscribers based on the content of the messages.

How to bring Content-based Filtering in MQTT?
Even though MQTT uses subject-based filtering of messages, you can also set up content-based filtering by using the HiveMQ **MQTT broker** and our custom **extension system**.

Benefits of Content-based filtering in Pub/Sub:

- Provides more granular control over which messages are received
- Allows for filtering based on message content rather than just topic hierarchy

- Flexible, allowing for complex filter expressions

Drawbacks of Content-based filtering in Pub/Sub:

- Can be more complex to use and set up than subject-based filtering
- Requires publishers to include additional metadata in the message to enable filtering
- Performance may suffer when processing large numbers of filter expressions

Use Case of Content-based filtering in Pub/Sub:

Content-based filtering is best suited for use cases where messages are not organized into topics, and subscribers are interested in a specific subset of messages based on their content.

For example, in a logistics application, a subscriber may be interested in receiving messages only about packages with a specific tracking number. The subscriber would subscribe to a filter expression like `"tracking-number = '123456',"` and the broker would only send messages that match this expression to the subscriber.

Option 3: Type-based filtering of Pub/Sub architecture

In type-based filtering, the broker filters messages based on their type or class. This type of filtering is useful when working with object-oriented languages where messages are represented as objects. Subscribers indicate their interest by subscribing to a specific message type or class, and the broker routes messages

to the appropriate subscribers based on the message type.

Benefits of Type-based filtering in Pub/Sub:

- Allows for filtering based on message type, regardless of the topic or content
- Simple to use, especially when working with object-oriented languages
- Offers a high degree of flexibility and extensibility

Drawbacks of Type-based filtering in Pub/Sub:

- Limited to filtering based on message type only
- Publishers and subscribers need to agree on the message type hierarchy beforehand

Use Case of Type-based filtering in Pub/Sub:

Type-based filtering is best suited for use cases where messages are organized into a class hierarchy, and subscribers are interested in a specific type or subset of messages based on their class.

For example, in a financial application, a subscriber may be interested in receiving messages only about stock prices. The subscriber would subscribe to a message type like “stock-price,” and the broker would only send messages of this type to the subscriber.

These filtering options provide flexibility and granularity in deciding which messages are sent to which subscribers. Depending on the use case, you can use one or more of these filtering options to ensure that subscribers receive only the messages they are interested in

the messages they are interested in.

However, it's important to note that the pub/sub model may not be suitable for all use cases, and there are challenges to consider, such as ensuring that both the publisher and subscriber know which topics to use for subject-based filtering and dealing with instances where no subscriber reads a particular message. You need to be aware of how the published data is structured beforehand.

> **For subject-based filtering, both publisher and subscriber need to know which topics to use.** Also, with message delivery, the publisher can't assume somebody is listening to the messages that are sent. This is an issue because, in publish-subscribe model, the publisher sends messages to the broker without knowing who the subscribers are or whether they are currently connected to the broker. The broker is responsible for delivering messages to all connected subscribers who subscribe to the appropriate topic. However, if there are no subscribers currently connected to the broker who have subscribed to the topic of a particular message, that message will not be delivered to anyone. Therefore, it is vital for publishers to keep in mind that message delivery is not guaranteed and to design their systems accordingly.

MQTT Pub/Sub's Scalability Feature

Scalability is one of the significant benefits of using the Pub/Sub architecture. The traditional client-server model can limit scalability, particularly when dealing with large numbers of clients. However, with the pub/sub model, the broker can process messages in an event-driven way, enabling highly parallelized operations. This means that the system can handle a greater number of concurrent connections without sacrificing performance.

In addition to event-driven processing, message caching and intelligent message routing also contribute to improved scalability in Pub/Sub. By caching messages,

the broker can quickly retrieve and deliver them to subscribers without additional processing. Intelligent routing, on the other hand, ensures that messages are delivered only to the subscribers that need them, reducing unnecessary network traffic and further improving scalability.

> As MQTT follows the pub/sub architecture, scalability comes naturally to this protocol, making it ideal for several IoT use cases. Despite its advantages, scaling up to millions of connections can still pose a challenge for Pub/Sub. In such cases, clustered broker nodes can be used to distribute the load across multiple servers, while load balancers can ensure that the traffic is evenly distributed. Check out how **HiveMQ broker can scale to 200 million concurrent connections** using this method.

Now that we have covered the basic concepts of Publish/Subscribe architecture, let's look at the benefits of using it for IoT communication.

What Are the Key Benefits of MQTT Pub/Sub Architecture in IoT and IIoT?

The pub/sub model offers several benefits, making it a popular choice for various applications. Here are some of the key advantages of using pub/sub architecture:

Improved scalability: The pub/sub architecture is highly scalable, making it suitable for applications that handle many clients and messages. The broker acts as a central hub for all messages, allowing it to handle many clients without compromising performance.

Increased fault tolerance: The decoupled nature of pub/sub architecture also

provides improved fault tolerance. In a traditional client-server model, all connected clients lose their connection if the server goes down. In contrast, in pub/sub, the broker can store messages until the client reconnects, ensuring no messages are lost.

Flexibility: The pub/sub architecture is flexible and can be used in a variety of applications, ranging from low-bandwidth, high-latency networks to high-speed, low-latency networks. The MQTT protocol, which is based on pub/sub architecture, supports various quality-of-service levels, providing the flexibility to choose the appropriate level for your application.

Common Challenges in Pub/Sub Architecture and How to Overcome Them While Using MQTT

While pub/sub architecture offers several benefits, such as scalability, flexibility, and decoupling of components, it also presents some challenges that must be addressed to ensure a successful implementation. Below are some of the most common challenges of using pub/sub and solutions to overcome them:

1. **Message Delivery**: One challenge of using pub/sub is ensuring that messages are delivered to subscribers. In some instances, no subscribers may be available to receive a particular topic, resulting in the message being lost. **To overcome this, MQTT provides quality of service (QoS) levels.**

2. **Message Filtering**: Another challenge of pub/sub is filtering messages effectively so that each subscriber receives only the messages of interest. As discussed earlier, pub/sub provides three filtering options: subject-based, content-based, and type-based filtering. Each option has its benefits and drawbacks, and the choice of filtering method will depend on the use case.

MQTT uses subject-based filtering of messages and every message contains a topic that the broker uses to determine whether a subscribing client receives the message or not.

3. **Security**: Security is a crucial aspect of any messaging system, and pub/sub is no exception. **MQTT allows for several security options, such as user authentication, access control, and message encryption, to protect the system from unauthorized access and data breaches.**

4. **Scalability**: Pub/Sub architecture must be designed with scalability in mind, as the number of subscribers can grow exponentially in a large-scale system. **MQTT provides features such as multiple brokers, clustering, and load balancing to ensure that the system can handle a large number of subscribers and messages.**

5. **Message Ordering**: In a pub/sub system, message ordering can be challenging to maintain. As messages are sent asynchronously, it's difficult to ensure that subscribers receive messages in the correct order. However, **MQTT provides QoS levels that ensure messages are successfully delivered from the client to the broker or from the broker to a client.** Because MQTT works asynchronously, tasks are not blocked while waiting for or publishing a message. Most client libraries are based on callbacks or a similar model, making the flow of messages usually asynchronous. In certain use cases, synchronization is desirable and possible, and some libraries have synchronous APIs to wait for a specific message.

6. **Real-time Constraints**: In some use cases, real-time constraints are critical, and pub/sub architecture may not be the best choice. For example, a request/response architecture may be a better option if low latency is essential.

You can address the challenges of using pub/sub through careful design and implementation. Developers can build scalable, secure, and efficient messaging systems by understanding these challenges and utilizing MQTT's features effectively.

MQTT Vs. Message Queues

There is a lot of confusion about the name MQTT and whether the protocol is implemented as a message queue or not. We will try to shed some light on the topic and explain the differences. In **part 1 of MQTT Essentials**, we mentioned that MQTT refers to the MQseries product from IBM and has nothing to do with "message queue". Regardless of where the name comes from, it's useful to understand the differences between MQTT and a traditional message queue:

A message queue stores message until they are consumed. When you use a message queue, each incoming message is stored in the queue until it is picked up by a client (often called a consumer). If no client picks up the message, the message remains stuck in the queue and waits to be consumed. In a message queue, it is not possible for a message not to be processed by any client, as it is in MQTT if nobody subscribes to a topic.

A message is only consumed by one client. Another big difference is that in a traditional message queue a message can be processed by one consumer only. The load is distributed between all consumers for a queue. In MQTT the behavior is quite the opposite: every subscriber that subscribes to the topic gets the message.

Queues are named and must be created explicitly. A queue is far more rigid than a topic. Before a queue can be used, the queue must be created explicitly with a separate command. Only after the queue is named and created is it possible to publish or consume messages. In contrast, MQTT topics are extremely flexible and can be created on the fly. If you can think of any other differences that we overlooked, we would love to hear from you in the comments.

Conclusion

To summarize, the publish/subscribe (pub/sub) architecture provides a flexible and scalable way of building distributed systems that can handle many connected clients. MQTT's lightweight and efficient pub/sub messaging characteristics have helped it gain widespread adoption in IoT, mobile, and other distributed applications.

> Using MQTT, architecture engineers, and companies can build systems reliably and efficiently communicate data in various real-world scenarios. With its decoupling by space and time, asynchronous messaging, subject-based filtering, and Quality of Service (QoS) levels, MQTT provides a robust set of features to help developers overcome the challenges of building distributed systems. Overall, the pub/sub architecture and MQTT protocol are valuable tools for developers who want to build efficient and scalable distributed systems.
>
> If you are looking to understand MQTT control packets and their structure to design and test MQTT-based systems, read our blog **MQTT Packets: A Comprehensive Guide**.

In part 3, we are going to look closely into what makes an **MQTT client and broker and how the two connect**.

Subscribe to our **RSS feed here** to stay updated. Do check out **MQTT FAQs** and **MQTT Glossary** to know all the key MQTT terminologies. Watch the video below that complements the concepts discussed in this article.

FAQs on MQTT Publish/Subscribe

Does MQTT follow the Pub/Sub architecture?

What are the different types of message filtering options available in MQTT?