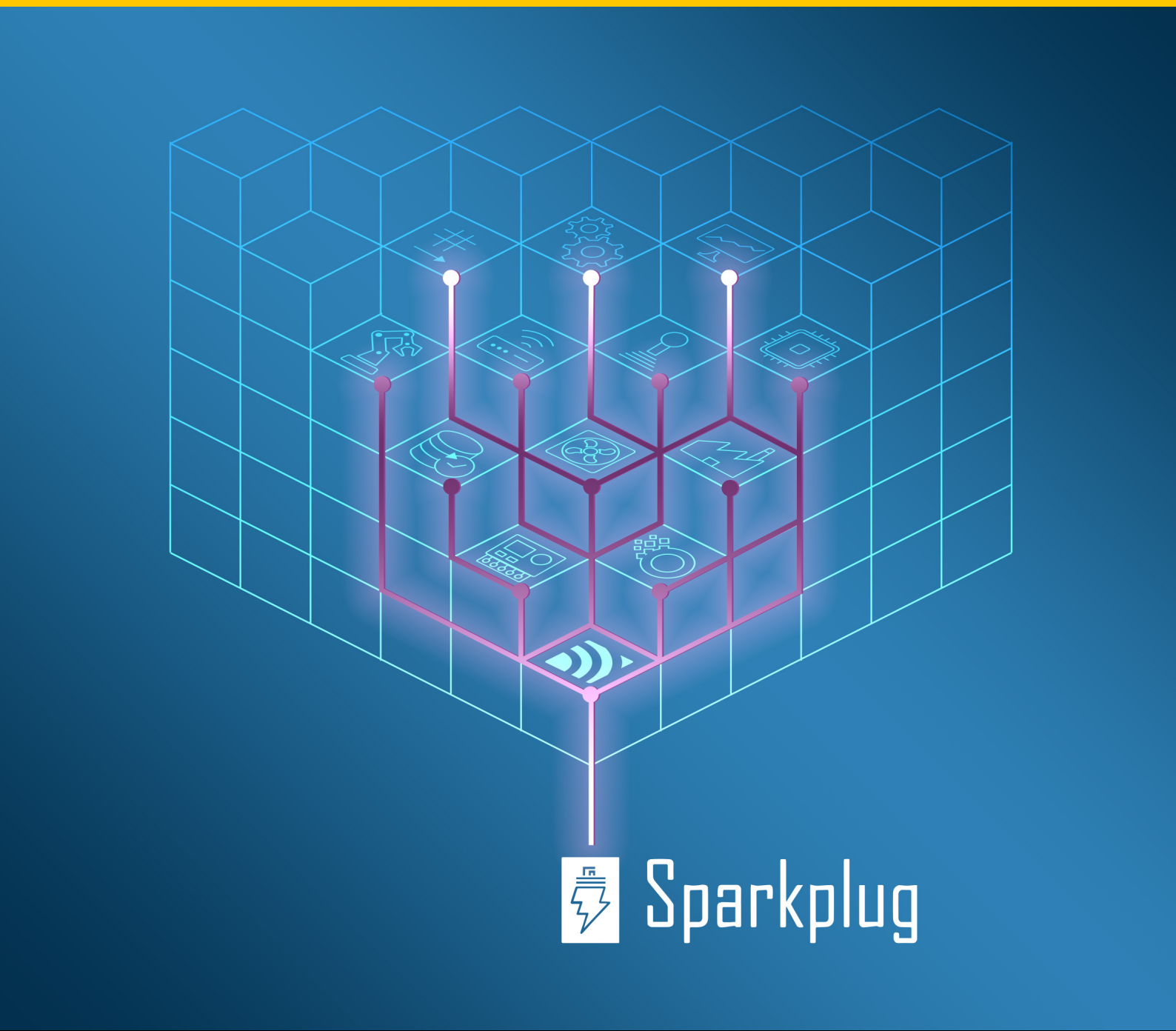
HIVEMQ

Industrial IoT Protocols: A Comparison of OPC UA and MQTT Sparkplug



Technical White Paper

Anja Helmbrecht-Schaar (HiveMQ Professional Services)

TABLE OF CONTENTS

Executive Summary

Digital transformation has been underway on the factory floor for a long time. With Industry 4.0 and the associated networking of all systems that are involved in the production process, the importance of standards to ensure cross-system data exchange and efficient machine-to-machine communication is increasing rapidly.

OPC UA is one of the most common communication protocols for production environments. However, complete implementations of the extensive OPC UA specification are extremely complex. OPC UA is based on client/server architecture. When many heterogeneous applications and devices from different manufacturers must be networked, OPC UA architecture is challenging to implement.

Based on the increasing importance of interoperability in Industrial IoT (IIoT), MQTT offers an interesting alternative to OPC UA, particularly in combination with the Sparkplug extension.

MQTT, the OASIS standard messaging protocol for IoT, is characterized by its extremely lightweight nature and ease of implementation. The use of the MQTT protocol brings a fundamental change in architecture. The publish-subscribe pattern of MQTT reduces the configuration overhead of components and requires less bandwidth to communicate.

Sparkplug is an open standard that adopts characteristics of MQTT such as simplicity, efficiency, and comprehensibility. Sparkplug uses MQTT to specify how all system components within an infrastructure bidirectionally communicate through an MQTT broker.

The combination of MQTT with Sparkplug reduces complexity and increases efficiency, especially in the operation of heterogeneous production structures. Maintenance, repair, and the addition of new components are also greatly facilitated.

For existing infrastructures, integration can be achieved with a combination of Sparkplug and MQTT. Legacy devices can connect to the Sparkplug infrastructure via MQTT sensors and provide their data centrally and for all components.

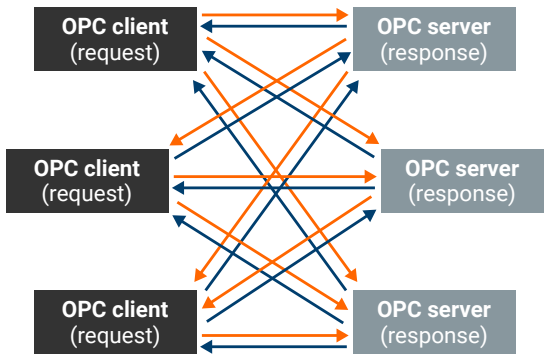
Introduction to OPC UA

OPC Unified Architecture (OPC UA) is a connectivity framework standard in the manufacturing environment that was published in 2008 as the successor to the OPC (Object Linking and Embedding for Process Control) standard. The unified architecture that OPC UA introduces is designed for platform independence.

One of the goals of OPC UA is to achieve device interoperability that is independent of the proprietary APIs of device manufacturers.

The OPC UA protocol is one of the most important communication protocols in the manufacturing industry. Typical use cases include industrial automation and process control applications as well as client-server interactions between components such as devices or applications. To facilitate configuration, browsing, monitoring, and data access, the address space of servers and devices is exposed to allow queries to all objects that are located in the space. OPC UA also supports a semantic description of data. The requests of the OPC UA client are sent to an OPC server. The OPC server processes the request and sends a response back to the respective OPC UA client. Structured data is used for this request-response communication pattern.

OPC UA is based on client/server architecture and uses TCP/IP and HTTP/SOAP as underlying technologies.



The OPC UA server converts the hardware communication protocol so that the device data via a standardized device model.

Security is implemented through various methods such as PKI certificates, WebSocket tokens, TLS, and username/password authentication for device clients.

Error management and exception handling are supported and communicated via different events and alarms. OPC UA services include limited support for message delivery guarantees (QoS); however, QoS functionality is not present as an underlying concept.

OPC UA defines a complete data type system. Resources in OPC UA are referred to as nodes. The different nodes that make up a system are individually addressable and can be structured as data objects from structured data types of varying complexity.

Additionally, OPC UA discovery services are available to dynamically detect new components in the infrastructure setup.

Generic device models play a central role in OPC UA architecture. Device manufacturers are responsible for providing the server that maps a generic device model to the specific device.

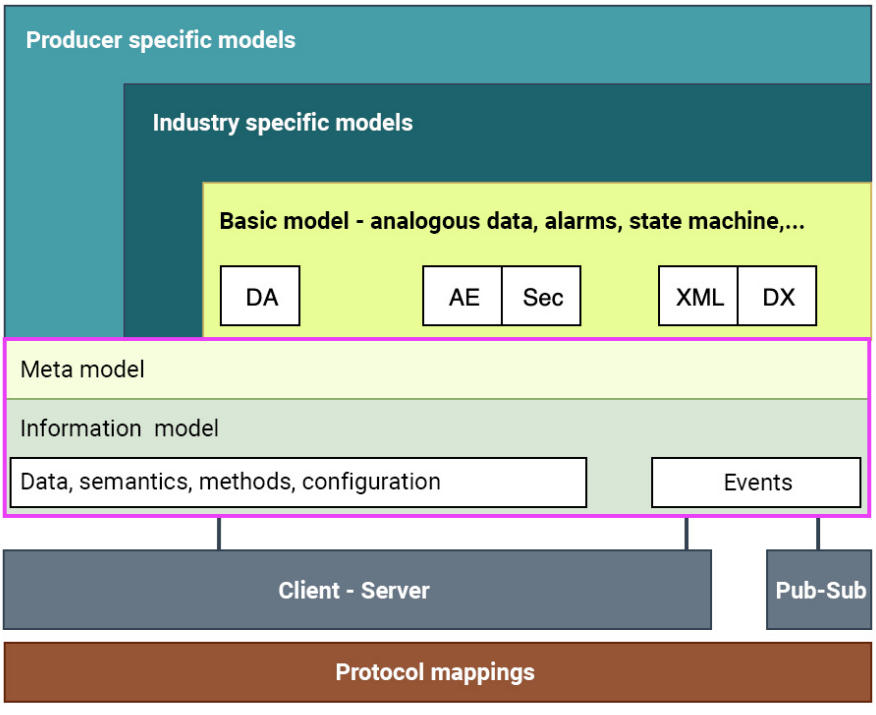
The OPC UA standard is composed of numerous individual specifications. Each specification describes a sub-function and specifies which interfaces the servers and clients must implement to support a particular function. Since it is not necessary to implement all of the individual specifications, the client-server system that is running OPC UA must identify which specifications the client and server need.

Companion specifications from various industries for the devices and systems involved can help identify which specifications are necessary. The companion specifications supply predefined structures that are created for the respective industry-specific applications and the associated objects.

Communication to other services can be established with gateway implementations for networking middleware (DDS) and direct machine to machine communication (M2M) applications, or through an HTTP gateway.

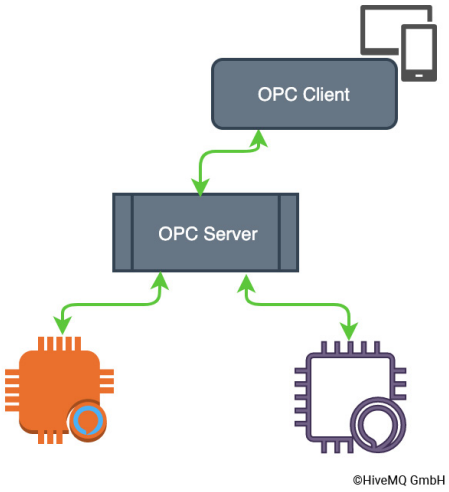
Servers provide an object-oriented remotely-callable API that implements the device model. The device can be accessed via a standard device model. There are device models for dozens of device types, from sensors to feedback controllers.

For example, the object model of a particular sensor can provide methods for setting parameters, reading data, and operating the device. The methods allow applications to directly control the sensor without knowing the exact implementation of the respective manufacturer.



One or more servers in the system wait for any number of clients to send requests. When a server receives a request, it responds and then returns to a waiting state.
If desired, the client can instruct the server to send updates as they arrive on the server.

In OPC, the client decides when and which data the server retrieves from the underlying systems. For example, even if the server subscribes to periodic status updates, it is up to the client to determine how often the server polls the devices and systems.



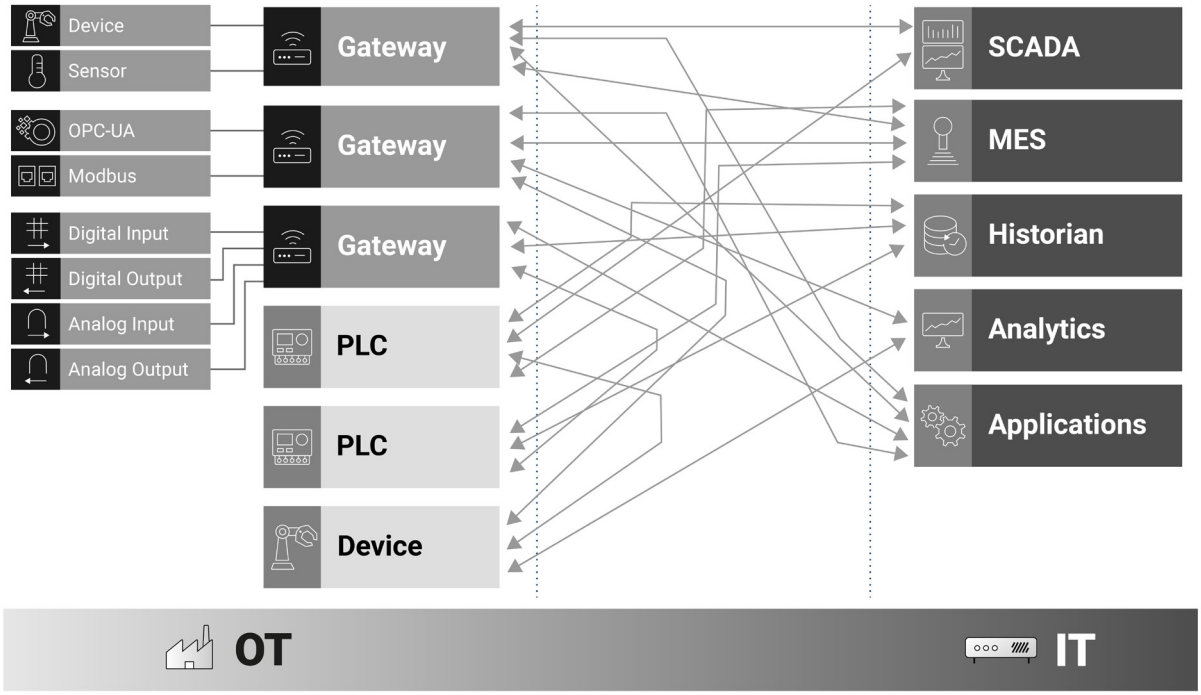
The capabilities of OPC UA, such as browsing or reading meta information and the flexible organization of variables in monitoring lists, make it easier to view and monitor machines externally.

However, when numerous heterogeneous applications and devices from different manufacturers are connected, the challenges of implementing OPC UA architecture increase. With more than 1,200 pages, the sheer length of the OPC UA specification foreshadows how extensive a complete implementation might become. Full implementations are not only expensive in terms of development and support costs, but also in the significantly higher CPU requirements OCP UA places on the devices.

The implementation of multiple consumers that one-to-many constellations and highly flexible publish-subscribe architecture necessitate are problematic. Due to the underlying architecture, real data decoupling cannot be achieved. The connection to cloud-based applications can also prove to be complicated or require a high implementation effort.

Here is a summary of the essential features of OPC UA:

- Client-server architecture
- Platform independence and interoperability through the use of TCP / HTTP as the transport protocol
- Security with TLS and certificates
- Fully defined data type setup
- Fixed structure data objects and endpoints
- Library of predefined industry-specific specifications
- Discovery service available
- Gateway compatibility



Classic OT / IT Infrastructure with OPC UA

Typical architecture in production environments includes numerous devices, sensors, and gateways that potentially communicate via different protocols. Each entity provides its data or is controlled through direct bidirectional connections. In OPC UA, there is no “single source of truth“. Any number of OPC UA servers from the operational production area can communicate directly with all the other operational technology (OT) participants. Each OT participant can, in turn, work as a client or server and with one or more OPC UA clients in the information technology (IT) area.

Introduction to MQTT

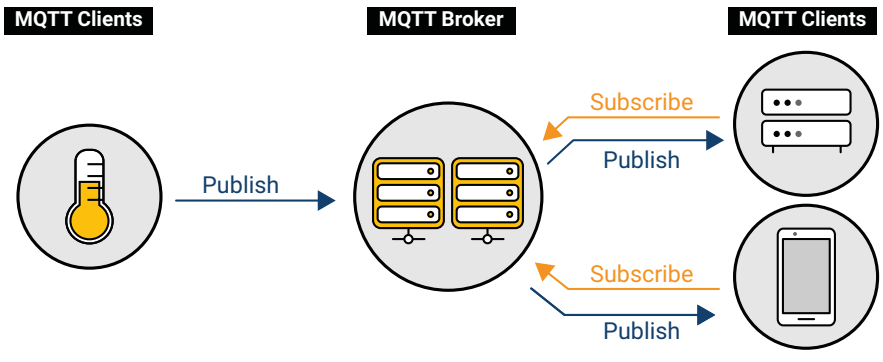
If you want to bypass complexity and move to a lean solution that guarantees a secure and reliable data exchange in industrial automation, you invariably encounter the MQTT protocol.

MQTT celebrated its 20th anniversary in 2019. As an OASIS standard, the MQTT 5 protocol offers a specification that is generally acknowledged as the de facto standard for IoT. The core principle of the exceptionally lightweight MQTT protocol is the publish-subscribe pattern. This pattern allows any number of data consumers to subscribe to individual topics or subject areas and receive the messages published about them

The slimness of MQTT is particularly suitable for very low-resource devices and communication in low-bandwidth, unreliable, or high-latency networks.

MQTT architecture enables communication with an unlimited number of clients via the publish-subscribe protocol.

Since many detailed descriptions of the MQTT specification such as the **MQTT 5 Essentials** series are available, this paper focuses only on aspects of MQTT that are interesting in comparison with OPC UA.



Important MQTT features include:

- Lightweight, TCP-based
- Publish/subscribe architecture
- Secure
- Stateful session usage
- Data-agnostic
- Highly dynamic topics
- Delivery guarantees for messages
- Last Will and retained messages concept
- MQTT 5 features
  - Introduction of semantic metadata like user properties, payload indicators, or content type descriptors
  - Request-response pattern
  - Shared subscriptions
  - Negative acknowledgments
  - Message and session expiry per client
  - And more

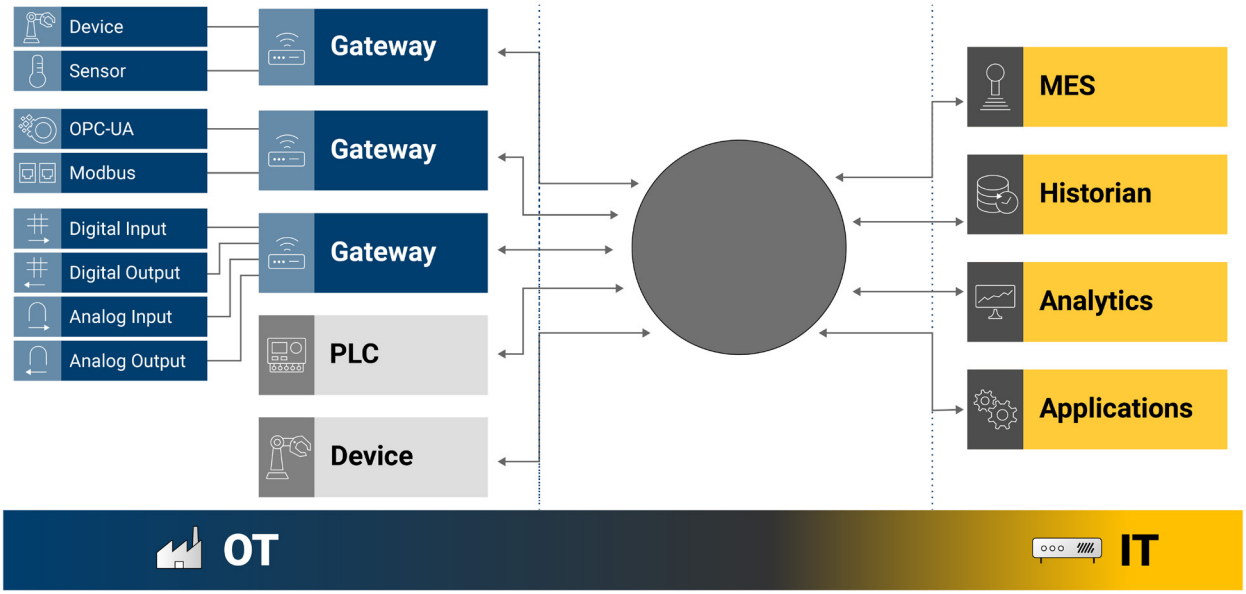
MQTT Infrastructure in the Production Environment

The use of a pub-sub protocol such as MQTT and a message broker as the central component represents a fundamental change in architecture.

All messages are sent via a central MQTT broker and all MQTT clients connect to the broker and can subscribe to specific topics.

The MQTT broker takes over the task of the server and handles each communication with an unlimited number of MQTT clients.

MQTT clients are implemented directly on gateways, devices, or within applications, and all of the clients are loosely coupled. There are no direct relationships between the clients.
In addition to functional requirements, the MQTT broker handles needs such as redundancy, failover, high availability, and scalability within a given infrastructure.



MQTT versus OPC UA

The publish-subscribe architecture of MQTT differs significantly from the client-server based architecture of OPC UA.

The central component of MQTT is always an MQTT broker. It is the responsibility of the broker to fully implement the MQTT specification. Particularly concerning MQTT 5 features, the compliance levels of brokers vary. Since many features in the MQTT 5 specification are optional, some brokers do not implement all of the features MQTT 5 offers. The different levels of support are especially relevant in the cloud area.

The use of sessions allows data for MQTT clients to be persisted beyond the duration of the client connection. For example, subscriptions, offline messages, or additional information stored for an offline client at the broker remain immediately available to the client when it reconnects.

Another essential difference in the context presented here is in the area of data agnostics. The MQTT protocol does not impose restrictions regarding data types on the specification level. As a result, the MQTT protocol can be used to transmit widely diverse types of data.

This complete freedom also applies to the topics. Beyond syntactic requirements, MQTT topics are not subject to any specifications. Topics are dynamic and do not have to be specified or created beforehand. The topic only exists in the context of an MQTT client that subscribes to the topic to consume incoming messages. If a message is published on a topic to which no MQTT client subscribes, the broker will ultimately discard the message.

To guarantee the complete transmission of messages with a stable connection, MQTT uses TCP as the transport protocol. Since MQTT was designed mainly for unstable networks, three quality of service levels (QoS), QoS 0, 1, and 2, are specified. This concept of delivery guarantees for MQTT messages is another significant difference from OPC UA.

The same relevance applies to the MQTT last will concept that specifies a message during client connection to send when an offline event occurs. A typical use case for last will messages is the transmission of status information from a client.

MQTT 5 also introduces user properties that make it possible to freely transfer meta information in MQTT packets without using the payload. User properties provide a simple and effective variant for exchanging meta information.

In contrast to OPC UA, the succinct 80 page MQTT specification is easy to implement and does not define fixed structures or data types.

IIoT Transition Challenges

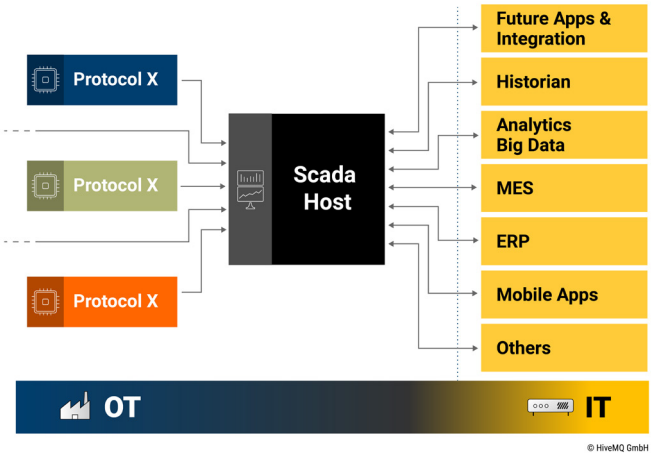
The digital transformation of the industrial sector is a megatrend worldwide that currently impacts companies that represent two-thirds of global GDP. With IIoT, traditional processes that have been established for 200 years are in the midst of a profound change.
Because digitalization has become a decisive factor in remaining competitive, change is imperative.

The Internet of Things or M2M communication must scale with the speed and simplicity of the classic human-HTTP-based Internet. The current challenge is to quickly and efficiently implement flexible and extensible IIoT systems that are easy to design and maintain.

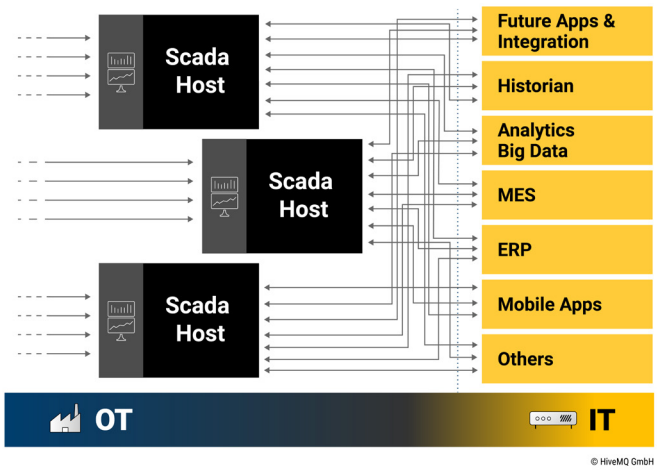
Challenges in IIoT include:

- Adding, adapting, and integrating new devices into existing systems
- Updating device configurations based on reported values
- Changing measurement, processing, or data delivery workflows

In systems that have few components, the maintenance effort required to operate a factory remains manageable, especially when a SCADA host is used as message-oriented middleware.



However, systems that require multiple host applications and connections to other enterprise IT components become very complex. This complexity creates an enormous manual configuration and maintenance load. The high manual operations activity is due to the underlying architecture pattern in which each component must use a request/ response approach to communicate with the SCADA hosts.



When multiple SCADA hosts are present, a convoluted infrastructure creates the following conditions:

- Separate integration points
- Every point must be polled via a request/ response pattern
- Increasing complex integration of new equipment into existing systems

The original goal of using IoT to make production lines more efficient is difficult to achieve in intricate systems of this type since the maintenance effort for each device update is linear to the infrastructure setup.

Conclusion on MQTT and OPC UA

- The concepts of the MQTT specification are perfectly suited for use in the digitalization of production.
- The “single source of truth“ and the decoupling of data through a central messaging component that MQTT offers provide a significant advantage.
- Security can be implemented with MQTT just as quickly and in as many ways as with OPC UA.
- The client status is an essential and inherent feature of the components involved and can be implemented flawlessly through MQTT functionality.
- Gateways can act as integrators for connecting devices that require other protocols.

The remaining question is how the lightweight MQTT protocol can best be applied to meet the interoperability requirements of industrial automation.

Introduction to Sparkplug

To fulfill the interoperability requirements of industrial automation, implementation standards that are based on MQTT must be defined. Providing this definition is the goal of Sparkplug.

Sparkplug is a project of the Eclipse Foundation's Sparkplug Working Group. Sparkplug provides an open and freely available specification that describes how edge gateways, native MQTT-enabled endpoints, and MQTT applications in infrastructure can communicate bidirectionally through a central component, the MQTT broker.

Sparkplug is a specification that defines how MQTT can be used in mission-critical, real-time OT environments. It establishes a standard that is optimized for use cases in industrial applications that utilize SCADA, tag, and metric representation. The specification describes how the principles can best be used in real-time SCADA implementations.

The Sparkplug specification aims to achieve three goals:

1. Definition of topic namespaces

The free and dynamic use of topic names is a key advantage of MQTT. However, it is necessary to specify an ontology when MQTT is used in production environments. The ontology shares the terms of use and maps the relationships between all devices and applications in the topic space.

2. Specification of payload data structures

Analogous to the specification of the topic namespace, when you use MQTT in production environments, a schema for the content (payload) of the messages must be defined. This schema definition can be flawlessly implemented with native MQTT 5 concepts.

3. Definition of state management

Since MQTT was originally developed to monitor real-time systems, a primary focus of the protocol from the start has been handling network failures, low bandwidth, and high latency. For example, through the full use session functionality.

The purpose of the Sparkplug specification is to pair MQTT strong points such as simplicity and ease of implementation and operation, with OT requirements. Sparkplug defines an ontology for all parties to guarantee session management for clients and keep message sizes to a minimum.

The Sparkplug specification gives developers and architects clear guidelines for designing topic namespaces and structuring payload data as well as ways to maintain and communicate client status.

MQTT Infrastructure with Sparkplug in IIoT Environments

In Sparkplug architecture, devices, EoN nodes, and the SCADA/IIoT hosts connect to a central MQTT broker to publish and subscribe to data.

As the central component, the MQTT broker must support the MQTT standard 100%. This is an important consideration that limits the suitability of cloud providers such as Amazon Web Services (AWS) or Azure IoT Hub since these vendors lack full **MQTT support** and use a proprietary version of the protocol.

Beyond full support of the MQTT protocol, the fail-safe, highly-scalable, and cluster-capable broker needs to supply metrics, monitoring, and alarm interfaces so that all system components involved can be optimally monitored.

In addition to the MQTT broker, an infrastructure that uses Sparkplug consists of the following components:

The **SCADA/IoT host** is the central application that system operators use to manage and monitor the overall state of the system. This application interacts directly with the MQTT broker as a specific MQTT client.
In contrast to a traditional SCADA system architecture, with Sparkplug the SCADA/IoT host is not responsible for directly establishing or maintaining connections to devices.

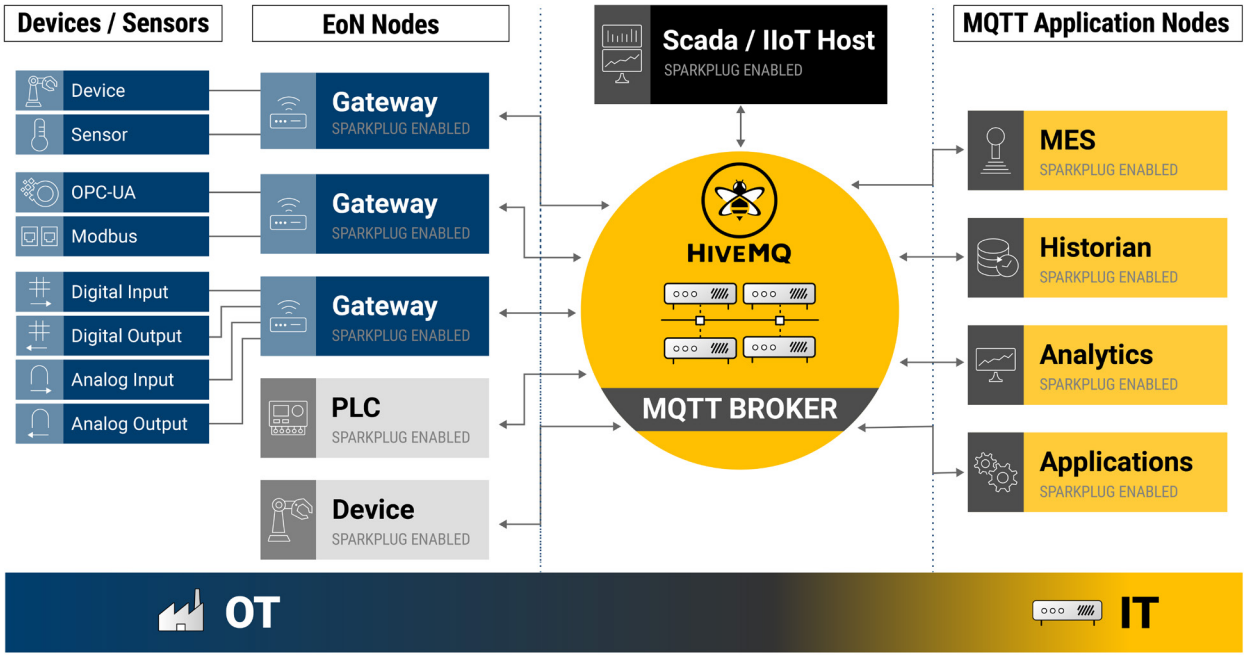
Edge of network (EoN) nodes play a key role in every Sparkplug infrastructure. Typically, EoN nodes are used to connect legacy infrastructures to Sparkplug. Legacy infrastructure elements can communicate with the EoN nodes via other protocols such as OPC UA, Modbus, or proprietary PLC manufacturer protocols. The EoN node is responsible for managing both its state and the state of the devices as well as receiving and sending data from the devices to the Sparkplug infrastructure.

Devices and sensors are the backbones of industrial automation. In the context of Sparkplug, devices connect to the Sparkplug infrastructure via EoN nodes. Although the majority of devices and sensors use protocols such as Modbus, OPC UA, and various other standardized or proprietary protocols, many vendors now offer native MQTT capabilities with their devices and sensors.
If the MQTT-enabled device is already equipped with Sparkplug, it can participate directly in the infrastructure. In this case, the device identifies itself as an EoN node for the Sparkplug infrastructure. If the device supports standard MQTT without Sparkplug detection, a connection to the EoN node must still be established.

MQTT applications, or secondary applications, are components that participate in Sparkplug communication and can generate and process MQTT messages, but are not the SCADA / IIoT host.

Conclusion

All MQTT clients, particularly the SCADA host and the IT applications, subscribe to the topics from which information is to be received. Due to the firmly defined topic structure and the data objects, each MQTT client knows where and what form of information can be retrieved. Since MQTT clients are stateful, no loss of information occurs at any time. Message types are specified for specific information.



Sparkplug Topic Structure

Every MQTT client that follows the *Sparkplug B* specification uses a predefined structure for the topic namespace that can be defined as follows:

```
namespace/group_id/message_type/edge_node_id/[device_id]
```

Name	Description	Example
namespace	Root element that sets the Sparkplug version	spBv1.0
group_id	Logical grouping for MQTT edge nodes	machine-group
message_type	The message type	NBIRTH
edge_node_id	One edge node	mqtt-edge-1

The Sparkplug specification defines several message types.

Different message types can transmit metadata as well as information about the state of a device.

Sparkplug Message Types

BIRTH AND DEATH CERTIFICATES FOR NODES AND DEVICES

A BIRTH message announces that an MQTT client is online. The device or node itself provides this information to the overall system on the BIRTH topic. This type of message is sent immediately after a connection is established. EoN nodes send NBIRTH messages, DBIRTH messages are sent by devices.

To ensure that the message can also be received by devices or applications that are not online when the message is sent, these messages are sent as a RETAINED message. This method allows components that log in to a specific topic later to receive the message.

DEATH messages are used to communicate the offline status of a device or EoN.

The MQTT last will (LWT) and RETAINED messages feature in the connect package of the MQTT clients are used to implement this concept. LWT messages are managed by the broker. If the client loses connection, the broker sends the LWT message to the corresponding DEATH topic. This method allows the offline status of the client to be transmitted as well. When a client logs off in an orderly manner with a DISCONNECT package, the offline status is published before the connection ends as a RETAINED message to the DEATH Topic. NDEATH messages are sent by EoN nodes, DDEATH messages by devices.

The status management of EoN nodes and devices can be implemented uniformly with the use of BIRTH and DEATH message types that have specific payloads and are used in the topic structure.

Metadata is also transmitted along with the status of the client. The inclusion of metadata via the BIRTH message makes it possible to precisely define a schema or template for the information that a client sends during its lifecycle. This lets the consuming MQTT client (in this case the SCADA host) know what information is to be expected from the respective device or EoN.

DATA MESSAGES FOR NODES AND DEVICES

This message type is used to transmit measurement data and certain device properties. Only changes regarding the information status since the last DATA message or BIRTH message of the device or EoN are transmitted. Because only updates are sent, the communication load is kept very low. The publish/subscribe architecture eliminates the need to poll for changes because changes are sent proactively to all subscribers who subscribe to the topic.

COMMAND MESSAGES FOR NODES AND DEVICES

This message type is used to transfer updates to EoNs or devices. The associated payload of the MQTT message contains, among other things, a timestamp and the metric that must be written to the device.

STATE STATUS MESSAGES OF THE PRIMARY APPLICATION

When you use a cluster-capable, dynamically-scalable MQTT broker, the State message type is optional. This message type is intended for use with Sparkplug in an infrastructure that implements multiple parallel, non-cluster-capable brokers.

The STATE message is used to transmit the BIRTH message for the SCADA IIoT hosts via the corresponding **namespace/group_id/STATE/scada_host_id** topic.

EoNs subscribe to this topic at startup to consume information from the host.

Sparkplug Message Content

Through the definition of the namespace, the topic structure is semantically related to the information to be transmitted.

The Sparkplug B specification supports the efficient transport of real-time data. The payload can be defined with different data types:

- Complex data types
- Data sets
- Rich metricsBesides data including metadata
- Metric aliases
- Historical data
- File data

Payload data is hierarchically structured and encoded in Sparkplug B in Protobuf format.

```
{
   „timestamp“: <timestamp>,
   „metrics“: [{
        „name“: <metric_name>,
        ...
   }],
   „seq“: <sequence_number>
}
```

The payload consists of a structured record that must contain at least the send out timestamp (UTC), a metric, and a sequential ID.

```
„metric“: {
     „name“: <string>,
     „alias“: <unsignedINT>,
     „timestamp“: <UTCTimestamp>,
     „datatype“: <unsignedINT>,
     „is_historical“: <boolean>,
     „is_transient“: <boolean>,
     „is_null“: <boolean>,
     „metadata“: <MetaData>
     „properties“: [
       { <String>, <ValueType> }
     ],
     „value“: <simpleOrComplexType>
   }
```

The name and the timestamp are mandatory. Flags are optional and can be used to indicate specific data. For example, “is_null” indicates that no data is sent. Custom data can be stored in properties as a key-value pair list or in the body object.

The report by exception (RBE) concept is another important aspect of MQTT and Sparkplug RBE ensures that only changes related to the previous situation are sent. This method reduces the data volume considerably. For specific message types, payload templates that describe the complete schema of the message content and the initial values can be sent at the beginning of a session. Based on the schema, only the changes are published and interpreted by consuming MQTT clients.

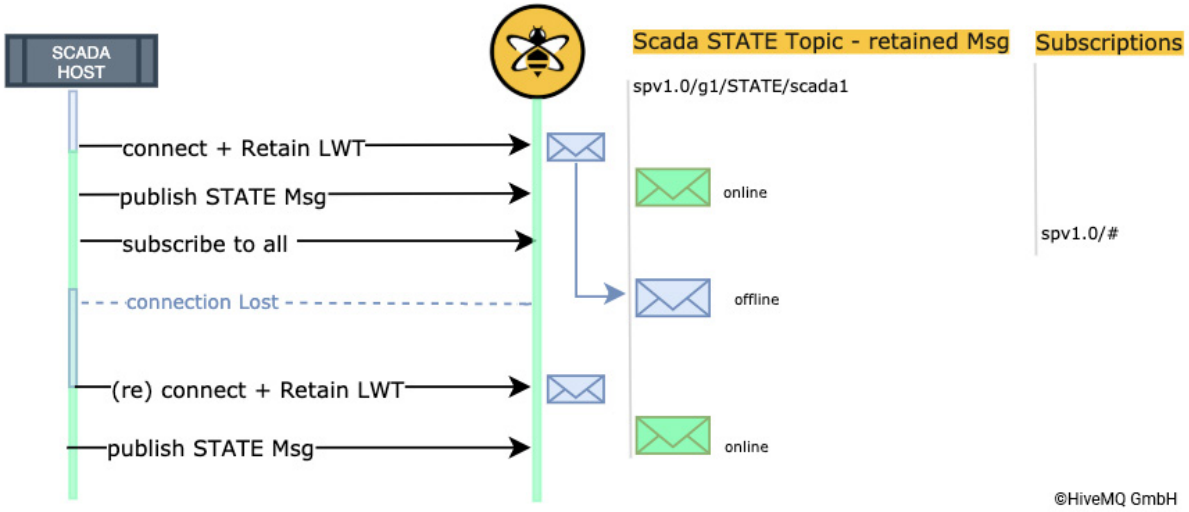
Sparkplug - Status Management

Besides the definition of the topic structure and content, it is necessary to define a concept for status management. The use of MQTT solves this requirement effectively.

The MQTT clients of the participating systems use sessions when establishing connections. With a combination of Sparkplug birth and death messages and MQTT last will and retained messages, each client in the system can map its status.

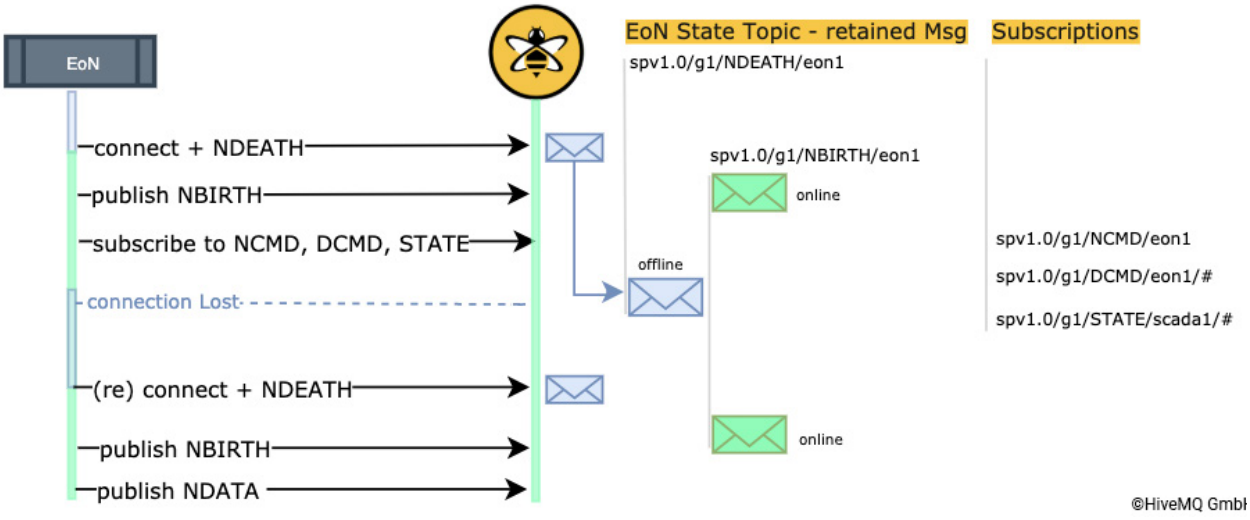
The provision of the status is subject to the individual components and available via the MQTT broker.

SCADA HOST STATUS MANAGEMENT



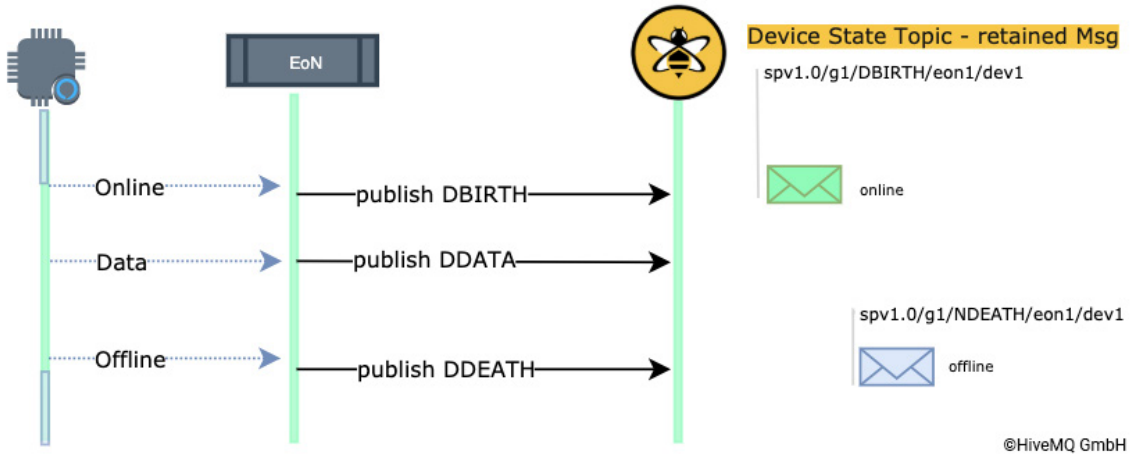
In the example, all components of the system can obtain the current state of the SCADA host by subscribing to the spv1.0/g1/STATE/scada1 topic.
Only changes in status are published. The last information is kept due to the use of RETAINED messages on the topic so that newly added clients get the information. In case of message loss, the broker publishes the LWT message to the topic as the RETAINED message.
The SCADA host also subscribes to all devices involved in the context so that all messages from the EoNs and their devices and other MQTT applications can be received. This can be further restricted to one group.

STATUS MANAGEMENT OF EON NODES



The online status for EoN nodes is also implemented using LWT and RETAINED messages. The message types NDEATH and NBIRTH are used.
By subscribing to the STATE topic of the SCADA host, the EoN node stays informed about the state of one or more hosts. By subscribing to the NCMD topic, the EoN node receives all relevant control information.
The subscription to the DCMD topic is used to receive control information for the devices that are linked to the EoN and for transmission of the information to the device.
The data of the EoN node is sent with the message type NDATA

STATUS MANAGEMENT OF DEVICES VIA AN EON NODE



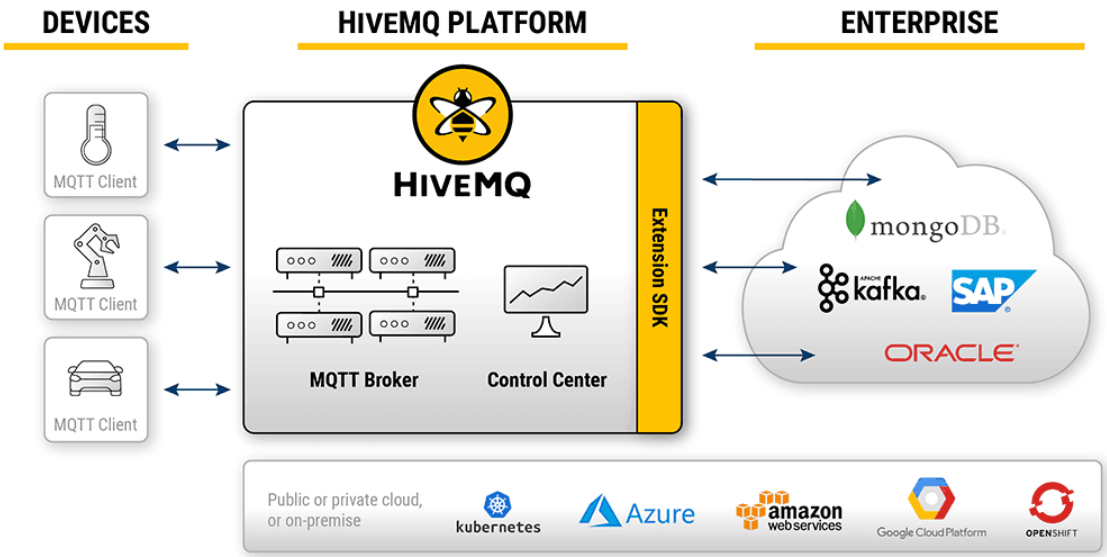
The communication of the devices and sensors connected to the EoN is handled by the EoN node. The online and offline information for devices is transmitted via the retained messages of the device-specific topics. The data of the devices is packed into data messages and published on the corresponding topic. All MQTT clients that are registered on the topic receive the measurement data.

Message Broker Selection

In MQTT, the message broker is the central component through which all messages are sent and to which all MQTT clients subscribe to specific topics.

The Sparkplug specification builds on the properties and features of MQTT. As discussed in the MQTT Infrastructure with Sparkplug in IIoT Environments section of this paper, the use of a broker that fully supports the MQTT protocol is crucial.

The possibility of using several separate brokers to guarantee failure safety, as described in the Sparkplug specification, is unnecessary when a cluster-capable, highly-available, and fail-safe MQTT broker such as HiveMQ is used.
The use of a cluster-capable broker also makes status management far simpler than what is described in the specification. From the outside, a cluster-capable broker is viewed as a logical unit and eliminates the need to specify or recognize at which broker the components connect.



HiveMQ has a web interface that is used to monitor the MQTT clients and the MQTT and system metrics and provides an administration interface.

HiveMQ also provides a flexible and well-developed extension system that can be used to add expanded business-specific logic.

Existing extensions can be easily integrated into an existing system and a wide range of open-source solutions are freely available,
Enterprise extensions for security, streaming data to external systems, or transferring data to another MQTT broker (bridging) are also available

The HiveMQ Extension SDK can be used to implement additional functionality that a particular context requires. For example, schema validation and payload transformation for messages of certain topics could be implemented with a custom extension.

Conclusion on Sparkplug and OPC UA

The concepts of the MQTT specification are perfectly suited for the digitization of production. The single source of truth and decoupling of data through the use of a central messaging component that MQTT offers are distinct advantages. The MQTT protocol supports diverse security options that can be quickly implemented and on-board tools are available for state management. Client status is an essential and inherent feature of the components involved and can be implemented flawlessly through MQTT functionality. Additionally, the inherent speed and low overhead of MQTT are highly desirable for the operation of real-time IIoT

The Sparkplug specification builds on MQTT to create a framework that is tailor-made for the requirements for IIoT systems. For example, providing the context data needed to define tag values for use with OT. By only sending data changes and using a publish-subscribe pattern, network traffic is significantly reduced. The use of an encoded and compressed data format for the message payload conserves network resources further.

This comparative study highlights the extreme bandwidth savings of MQTT and Sparkplug deliver over OPC UA or HTTP.

The defined ontology for data and topics ensures that sent information is interpretable by the components of the IT structures without the need for further meta-information. This uniform approach to how data and topics are designed enables manufacturers to deliver devices that fit the specification. Greater consistency makes the configuration and use of devices in IIoT easier, faster, and ultimately more cost-efficient.

Device integration in architecture that uses message-oriented middleware such as the MQTT broker is far easier than in a complex client-server structure. Each component in the system communicates only with the broker. No additional configuration is required.

OPC UA and MQTT can work together even when there are significant differences in the way data is handled. Some older devices need an OPC server and should remain connected to the Sparkplug infrastructure. By using an MQTT-capable sensor connected to an older device, it is possible to use MQTT and Sparkplug to make this data centrally available for all components in the IT.

Glossary

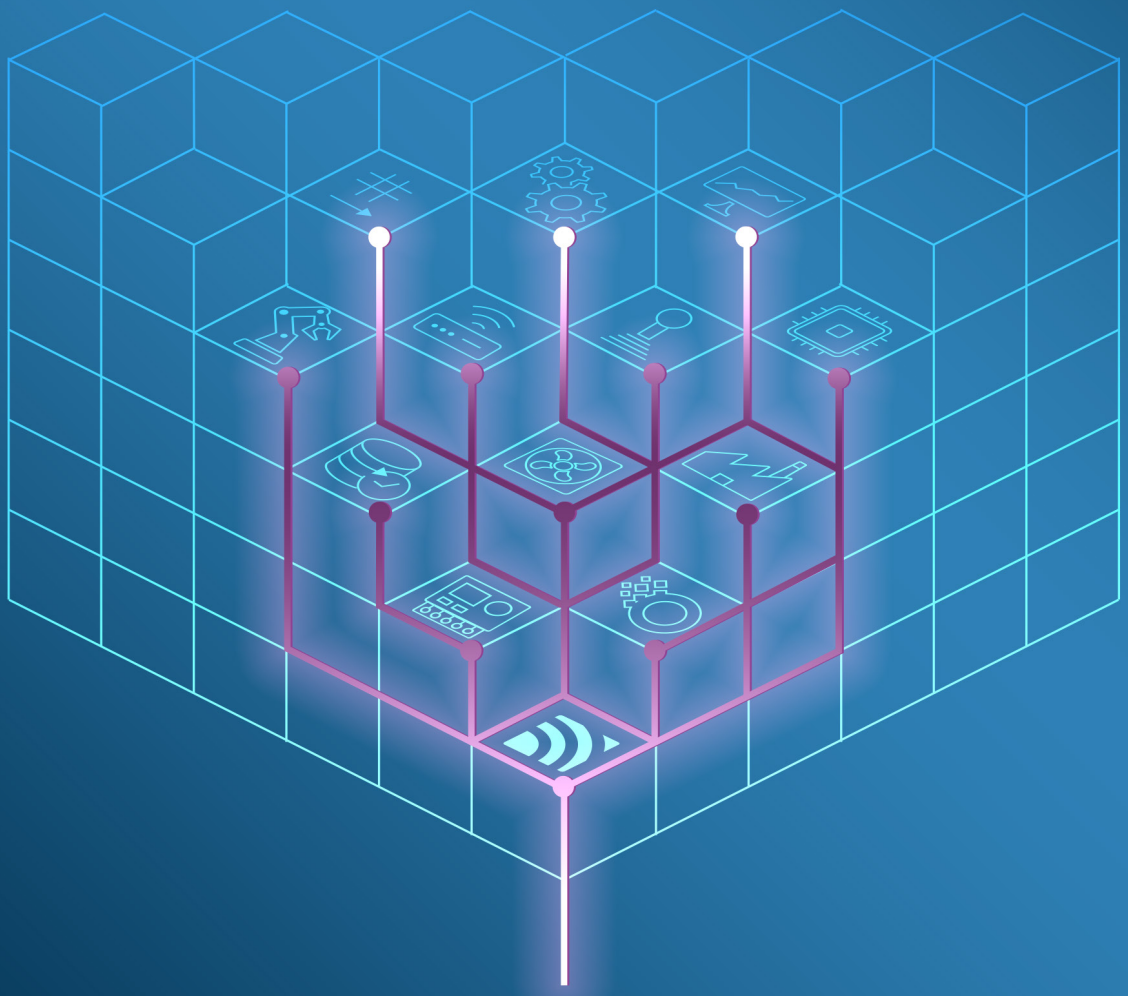
OPC UA	**opcfoundation.org/about/opc-technologies/opc-ua/**
Sparkplug	**sparkplug.eclipse.org/about/faq/**
HiveMQ	**hivemq.com**
MQTT	**hivemq.com/mqtt-5/**

For more information about Sparkplug and MQTT

> HiveMQ MQTT Sparkplug Essentials

> visit hivemq.com

> contact HiveMQ



Sparkplug

Ergoldingerstr. 2a
84030 Landshut
Germany

www.hivemq.com