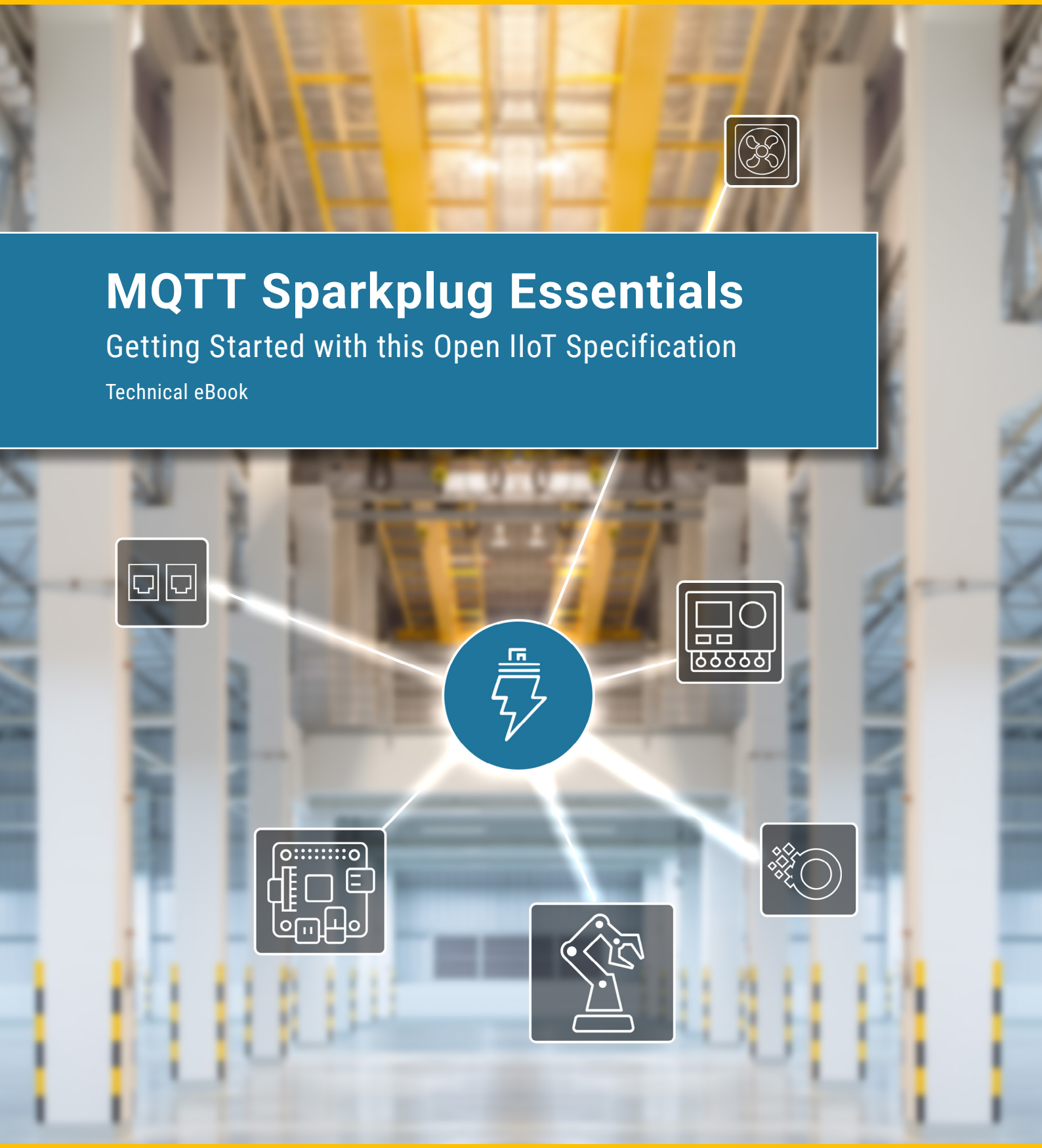
Sparkplug

HIVEMQ



MQTT Sparkplug Essentials

Getting Started with this Open IIoT Specification

Technical eBook

MQTT Sparkplug Essentials
Getting Started with this Open IIoT Specification

Table of Contents

Chapter 1 - Introduction to Sparkplug

In this e-book, we share the gist of the open Industrial Internet of Things (IIoT) specification, Sparkplug. If you want to implement Sparkplug on your own, integrate the specification into your product or just want to learn about it: This is the right starting point into your journey to one of the most important communication protocols for the IIoT.

IIoT and Industry 4.0 are key trends in the manufacturing industry. Many shopfloor operators across various industries are looking for operational efficiency gains, improved manufacturing capabilities and real-time manufacturing insights. However, this is daunting because the software and hardware stacks have traditionally been closed and proprietary, and interoperability was never a key concern for the vendors. This creates data silos.

Protocols like OPC-UA promised to break the silos and provide an industry-wide common language between devices, machines and software applications. The reality to most developers and software architects is that OPC-UA is not the silver bullet everyone hoped for. OPC-UA is extremely complex, heavyweight and is not always easy to integrate, especially in brownfield environments which you typically have in most manufacturing projects. So people felt there must be a better way.

On the other hand, device to cloud communication for minimal latency and maximal throughput got revolutionary easy with the MQTT protocol. And many developers wished for a simple solution like MQTT for manufacturing but with the features required for the manufacturing industry like payload definitions and unified messaging behavior across machines and vendors.

The wish came true when the Sparkplug protocol, which is based on MQTT, was first released by one of the very founding fathers of MQTT: Arlen Nipper. The Sparkplug specification took the industry by storm and large companies like Chevron adopted it for operational efficiency gains and for creating next-generation manufacturing solutions.

ABOUT SPARKPLUG

Sparkplug is an open-source software specification that provides MQTT clients the framework to seamlessly integrate data from their applications, sensors, devices, and gateways within the MQTT infrastructure in a bi-directional and interoperable way.

To have a common language for the IIoT, the Sparkplug specification defines the following three goals:

- Define an MQTT topic namespace
- Define MQTT state management
- Define the MQTT payload

It's noteworthy that Sparkplug is actually designed to run 100% on MQTT as the publish/subscribe paradigm of MQTT allows for bi-directional and decoupled integration of all components of a system. When MQTT was invented in 1999, it was originally designed for SCADA systems but left out all specifications around how topics and the payload should be structured and how devices should behave. This allowed MQTT to be used in different industries like connected car, logistics but also smart manufacturing.

Sparkplug now fills the gap and provides a vendor-neutral specification for data formats, topic structures, state management, and how topologies should be structured in IIoT scenarios.

Plain MQTT Vs. Sparkplug

Sparkplug was designed for Industrial Internet of Things applications based on MQTT. Many vendors support MQTT out of the box for their PLCs (for example Siemens S7) and most Manufacturing Execution Systems (MES) and SCADA systems (like Ignition SCADA by Inductive Automation®) support MQTT. Of course most professional gateway solutions used in manufacturing contexts support MQTT.

Why now add Sparkplug to the mix? For any non-Sparkplug MQTT communication you need to make sure that all participants who are interested in the data know where to subscribe to the data and you need to make sure all participants can interpret the data. This usually involves data transformation, which requires conventions, and creates a tight coupling between all the applications. With Sparkplug, all participants settle on a common data format, how to receive specific data, how to publish their data, and how to interpret data.

Best of all, Sparkplug allows bringing in data from non-MQTT devices as well data from other protocols like OPC-UA or Modbus. Additionally, we get all these devices and applications discovery out of the box.

Chapter 2 – Requirements for Sparkplug

To use Sparkplug, you need an MQTT broker that is responsible for distributing the data. It is important that the MQTT broker implements 100% of the MQTT 3.1.1 specification because Sparkplug requires the following:

- QoS 0 and 1
- Retained Messages
- Last Will and Testament
- A flexible security system

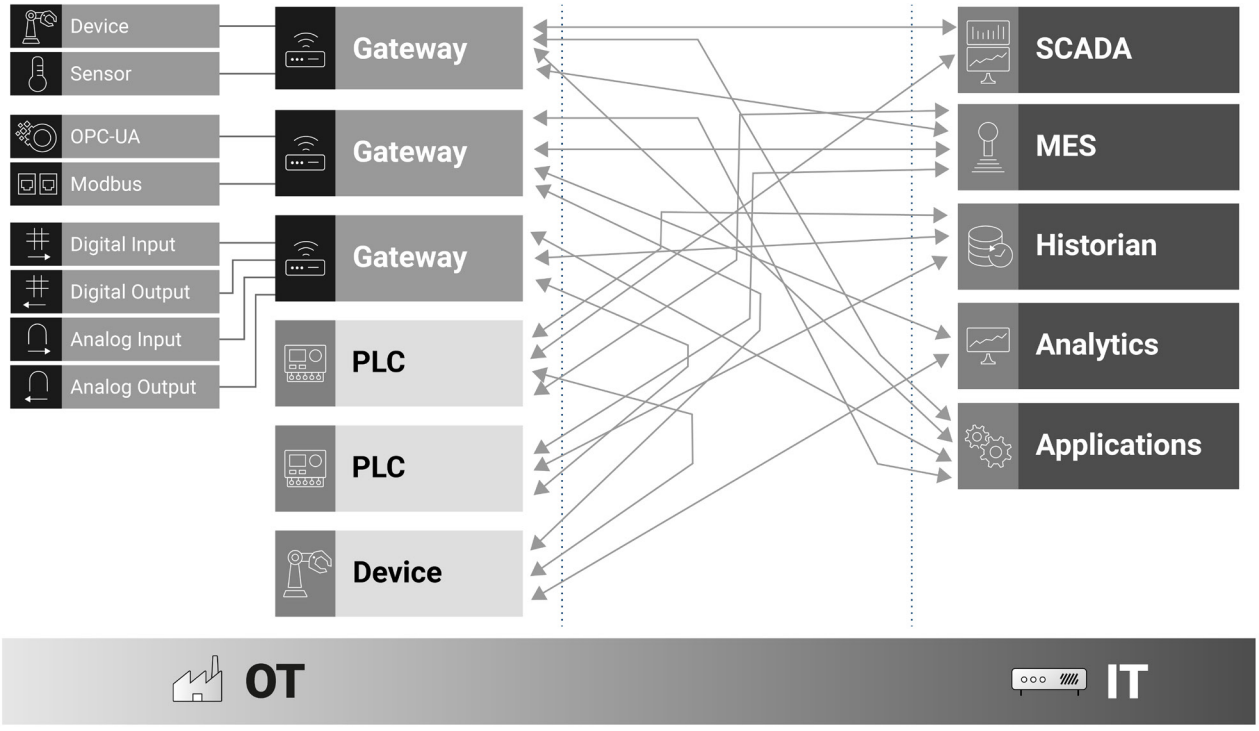
For Proof of Concepts, Eclipse Mosquitto and HiveMQ Community Edition are popular choices. For professional use cases we recommend the HiveMQ Platform. Unfortunately, AWS IoT and Azure IoTHub are not suitable for Sparkplug as they lack basic MQTT functionality.

Chapter 3 – Fundamental Architecture of Sparkplug

The typical Industrial Internet of Things architecture works by connecting components with a poll / response approach. Applications poll data directly from PLCs, gateways or servers with protocols such as Modbus, Siemens S7 protocol or OPC-UA. This works pretty well when there are only a few systems to integrate.

The Old World – Industrial Spaghetti Architecture

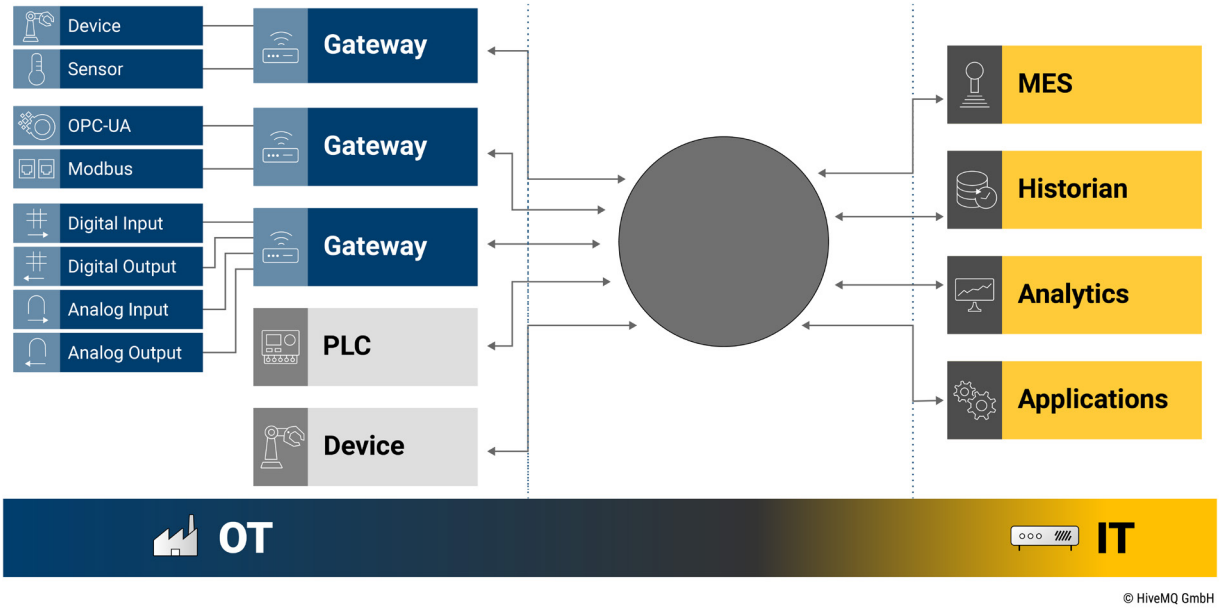
When a larger number of systems try to integrate in a typical Industrial Internet of Things architecture as described above, it will result in a huge spaghetti architecture that is very hard to maintain.



Picture 1: Industrial IIoT Architecture without Sparkplug

In picture 1, you can see that systems are connected to each other point-to-point and thus the systems and data are hardwired to each other. Modern architectures require flexibility and a clear separation of concerns in an IIoT system. Many companies are looking for the adaptiveness, flexibility and ease of implementation found in IT landscapes but with the reliability, security and predictability required for OT landscapes. A new architecture is needed for this paradigm shift.

A New Architecture for IIoT — Sparkplug and MQTT



Picture 2: A new architecture for the IIoT

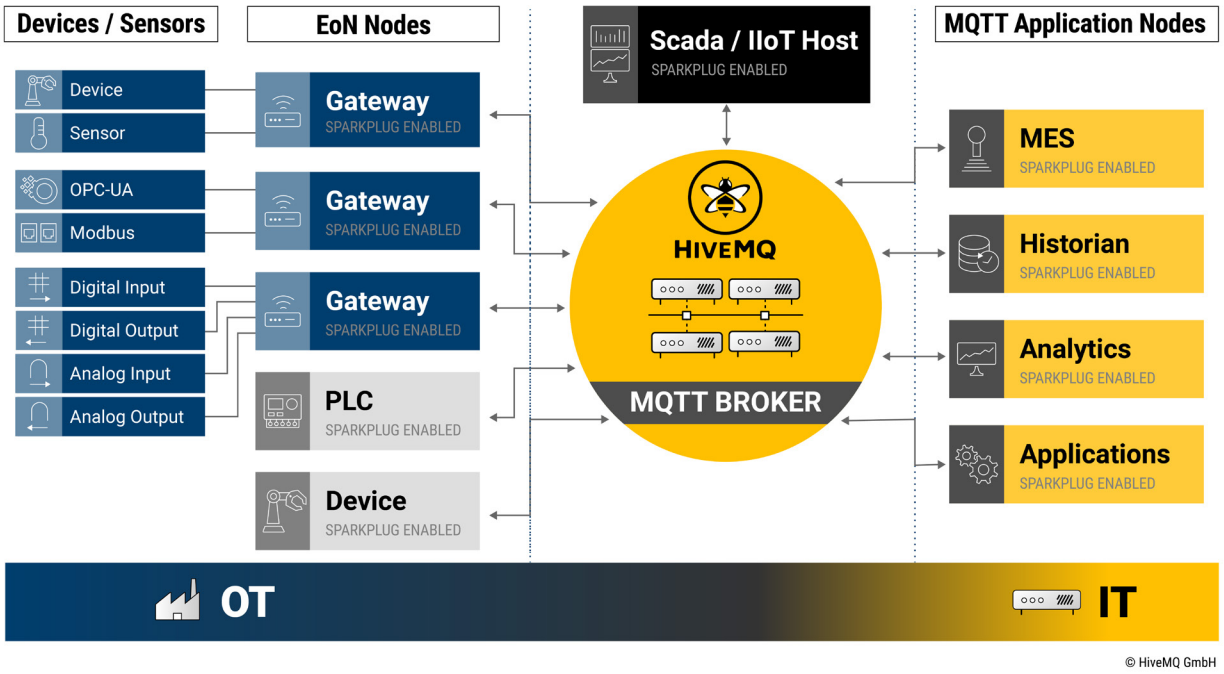
The new IIoT architecture blueprint, as depicted in picture 2, adds several benefits compared to the traditional IIoT architectures, such as:

- **Decoupling** of producers and consumers of data.
- **Report by Exception (RBE)**, which saves bandwidth, memory and computational power on the producer and the consumers of data.
- **One-to-many communication**, as data only needs to be sent once and multiple receivers can receive it.
- **Flexibility**, as devices and applications can be added and removed anytime without affecting the system as a whole.
- **Data governance** by having centralized permission and policy handling.
- **Shopfloor-to-cloud connectivity** by distributing data from cloud to the edge.

Many companies were using MQTT in the past for creating decoupled architectures for their factories. This is no surprise as MQTT was originally invented with SCADA systems in mind.

- For IIoT use cases using MQTT there are still pieces of the puzzle missing, such as:
- An MQTT topic structure definition
- MQTT state management
- Payload data definitions

Sparkplug adds these capabilities to MQTT and a Sparkplug architecture, which usually looks like in picture 3.



Picture 3: Sparkplug Architecture

Chapter 4 – Principles and Mechanisms of Sparkplug

The Sparkplug architecture is elegant compared to legacy solutions because it builds upon the following principles and mechanisms:

1. **Pub / Sub:** It uses MQTT as a publish / subscribe architecture for the underlying application transport layer and decouples producers and consumers of data. MQTT is based on push communication, which means data is distributed instantly to all interested parties.
2. **Report by Exception:** Data and device state is only updated if it changes. This dramatically saves bandwidth and computing power for all components as only new and fresh data is sent over the network.
3. **Continuous Session Awareness:** Sparkplug and MQTT have the concept of continuous session awareness. It informs all the clients that care about the real-time information of the device online/offline state if it changes. This concept also makes sure that data in transit is continued to be sent to devices if they change state from offline to online again. With Sparkplug, you get a real-time correct view of all the devices, gateways, and applications of the deployment.
4. **Death and Birth Certificates:** Sparkplug introduces Death and Birth Certificates that are used for the management and discovery of device state. Birth certificates encapsulate information about the device and the data it can and will send. Death certificates are using the MQTT Last Will and Testament mechanism to push device offline information to all interested applications.
5. **Persistent Connections:** All devices, gateways and applications are by default always on and use persistent TCP connections.
6. **Auto Discovery:** Applications and devices can auto-discover what data (and the corresponding topic) will get sent by all participants in the Sparkplug deployment as well as the online/offline devices that are connected.
7. **Standardized Payload Definition:** The Sparkplug data format for all messages is standardized and can be decoded and encoded by all communication participants.
8. **Standardized Topic Namespace:** All Sparkplug participants use a common topic namespace. The topic namespace allows for fine-grained subscriptions of specific data and allows for dynamic addition and removal of participants.

Chapter 5 – The Components of Sparkplug

Sparkplug recognizes that there are different types of devices/sensors, gateways, applications and other software (as well as hardware) involved in any non-trivial IIoT scenario. Sparkplug defines the behavior and semantics for the different kinds of participants in the architecture.

A traditional Sparkplug Architecture consists of the following components:

- SCADA / IIoT Host
- Edge of Network (EoN) Nodes
- Devices / Sensors
- MQTT Application Nodes
- MQTT Broker

We will look at these components now in detail.

5.1 SCADA / IIoT Host

The SCADA / IIoT Host, sometimes also referred to as Primary Application, is the supervising application responsible for monitoring and control of the MQTT EoN nodes and their respective devices and sensors. Continuous Session State Awareness is key in an IIoT system, which means the current state of all participants (machines, devices, PLCs, sensors, gateways and applications) needs to be known at a central place at any given time. This central application managing the state (and acting upon state changes) is the SCADA / IIoT Host application. It is the central application that is used by the operators of the system to manage and supervise the health of the overall system.

In contrast to most traditional SCADA system architectures, the SCADA / IIoT Host is NOT responsible for establishing and maintaining connections to the devices directly. In a Sparkplug architecture, devices, Edge of Network (EoN) nodes and the SCADA / IIoT Host connect to a central MQTT broker and publish and subscribe to data, which allows a report by exception (RBE) functionality to only update data when changed.

5.2 Edge of Network (EoN) Nodes

An Edge of Network (EoN) node is one of the key roles in any Sparkplug system. EoN nodes usually provide physical or logical gateway functions for sensors/devices who don't implement Sparkplug themselves and let them participate in the MQTT Topic namespace. EoN nodes manage the state and session of itself and the sensors and devices connected to this EoN node via protocols like OPC-UA, Modbus, proprietary PLC vendor protocols, HTTP, MQTT or local discrete I/O. The EoN node is responsible for managing the lifecycle and state of these connected devices and sensors as well as receiving and sending data for the devices to the Sparkplug infrastructure. EoN nodes are a critical part of any Sparkplug infrastructure and very often EoN nodes are used to bridge legacy infrastructure to Sparkplug.

5.3 Devices / Sensors

Devices and sensors are the backbone of any industrial automation. A device is usually a physical or logical thing that sends and/or receives data over one or multiple industrial communication protocols. Usually, these industrial protocols are based on poll/response protocols. In the Sparkplug context, devices are connected to the Sparkplug infrastructure via EoN nodes. The EoN nodes bridge the publish / subscribe nature of MQTT Sparkplug to these poll / response protocols.

5.4 MQTT Enabled Sensors and Devices

While most devices and sensors use protocols like Modbus, OPC-UA, Beckhoff ADS and other standardized and proprietary protocols, many vendors offer native MQTT functionality with their devices and sensors. If the MQTT enabled device is already equipped with Sparkplug by providing the appropriate data format and topic structure, then the device can participate directly with the Sparkplug infrastructure. In this case, it will identify itself as EoN node to the Sparkplug infrastructure. If the MQTT enabled device supports only standard MQTT without Sparkplug awareness, then it still needs to connect to EoN node.

5.5 MQTT Application Node

MQTT Application Nodes are nodes participating in the Sparkplug communication and can produce and consume messages but are not the SCADA / IIoT Host. These are sometimes called Secondary Applications. Usually, these are software systems which provide dedicated functionality like MES (Manufacturing Execution Systems), Historian and Analytics. Many deployments also use customized software dedicated to specific use cases that need to consume data produced by the other Sparkplug participants.

5.6 MQTT Broker

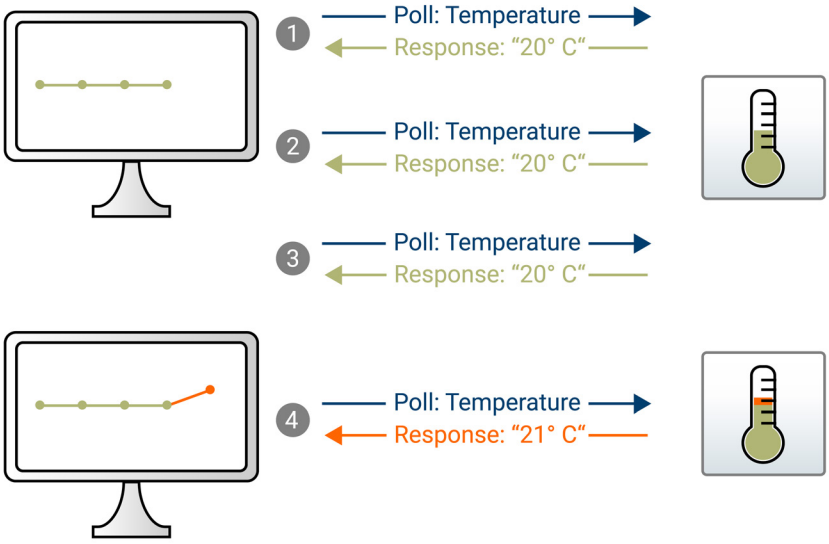
The MQTT broker is the central data distribution component. All Sparkplug enabled devices, EoN nodes, SCADA / IIoT Hosts and MQTT applications connect to the broker via MQTT. The broker is responsible for authentication, authorization, state management of the participants and data distribution between Sparkplug enabled systems. The MQTT broker needs to be 100% compliant to MQTT 3.1.1 as features like retained messages, Last Will and Testament, and QoS are needed.

Incomplete, non-MQTT compliant cloud brokers like AWS IoT and Azure IoTHub don't work with Sparkplug as they only support a subset of MQTT features and are technically not MQTT compliant. If the Sparkplug MQTT broker should reside in the cloud, AWS and Azure can still be used with a fully compliant broker implementation like HiveMQ hosted in the cloud.

It's important to understand that in MQTT architectures, the MQTT broker is a single point of failure as all communication fails when the MQTT broker is offline. This would mean the whole Sparkplug system is offline. While Sparkplug defines a very complex and limited way of high availability with multiple, separated brokers, there is a better way to achieve high availability easily without any modification on the application and EoN side: Brokers like HiveMQ allow for elastic clustering that provides high availability and resiliency with a cluster architecture. Even if one or more instances of the broker cluster fail (e.g. due to hardware problems), the system as a whole is fully operable, resulting in zero downtime. This is also true if you update the broker version, as rolling upgrades allow for zero downtime upgrades. This is especially important for mission critical 24/7 operations.

Chapter 6 – Publish / Subscribe for Sparkplug

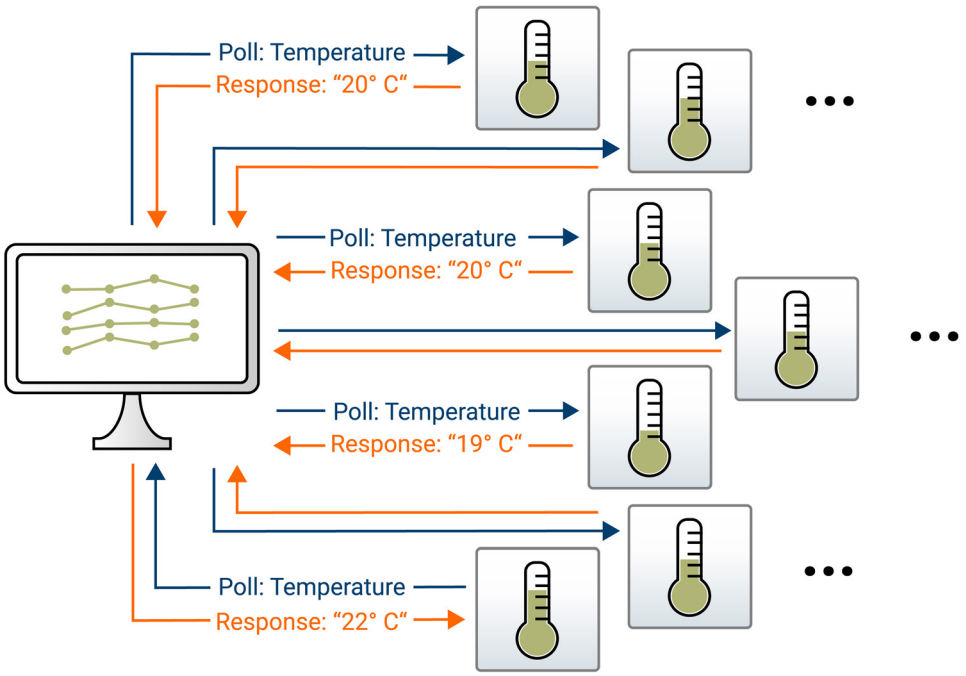
Sparkplug uses the Pub/Sub architecture pattern for scalable and efficient communication, which is dramatically different to traditional Poll/Response protocols used in the last 40+ years in industrial automation, discrete manufacturing, and in industries like Oil & Gas. This means that the communication protocols of the last decades require a very tight coupling between the producer of data (often PLCs) and the consumers of data. This makes it difficult to change workflows and processes, makes it hard to set up new systems and facilities and makes it difficult to impossible to use and analyze data across the entire system.



Picture 4: Poll / Response way

Picture 4 shows the traditional Poll / Response way to get data from PLCs, gateways and applications. A polling system is requesting data from the device that produces the data or is the single source of truth for specific data. In order to get data as soon as possible, the polling system is asking for data in a very high frequency, otherwise new data won't be available to the system that needs that data. This approach is very inefficient as it wastes bandwidth and processing power scale very well.

Jonathan Hottell compared MQTT vs OPC-UA vs Modbus in this presentation and showed that even for basic scenarios, there are orders of magnitudes in efficiency gains when using MQTT Sparkplug compared to OPC-UA. If you want to add security via encrypted communication (which you MUST do if you are connecting things over the internet), then overhead produced by legacy protocols compared to Sparkplug is even more significant. Picture 5 depicts that for each new data producer a system needs to poll. This means in the worst case each tag needs to be polled individually.



Picture 5: Poll / Response of many producers

This polling approach is used widely in the industry and protocols like Modbus and OPC-UA. The exponential growth of data produced and the hyper-connectivity locally on the site as well as worldwide showed the limitations of the poll / response approach. Companies today require data that is produced globally in up to hundreds of factories and thousands of machines in an instant and expect that important data is accessible to all stakeholders easily, independent of where they are in the world. The industry is now finally moving towards a world where data silos must be broken down.

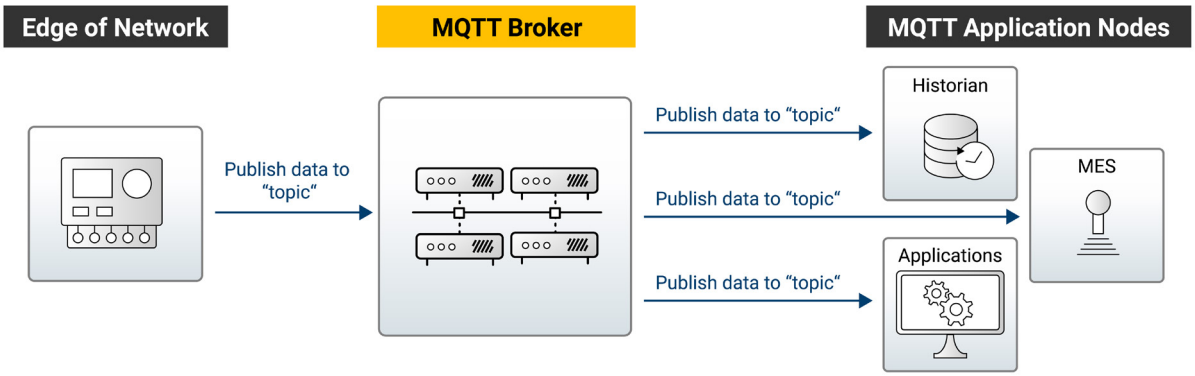
Poll / Response comes with some severe disadvantages compared to modern publish / subscribe approaches from a technical point of view:

- No state awareness: Most IIoT protocols today are not state aware, which means state needs to be polled all the time, sometimes multiple times per second for every device to make sure no important data or state change is lost
- Massive amount of unnecessary data traffic: Data producers are getting polled up to multiple times per second even if no status change or data change occurred
- Only periodic check for new data: No instantaneous push based mechanism to get data and events as they happen
- Compute intensive on the polled devices: Lots of unnecessary compute cycles are wasted as data is requested all the time even if it didn't change

- Tight coupling between polled and polling components: If a data producer needs to be changed / replaced, multiple systems need to be reconfigured, possibly with downtime
- Doesn't scale to large amounts of data producers.

There clearly must be a better way. And this is why MQTT Sparkplug started with a blank sheet of paper and asked: "What if we could use the most lightweight communication protocol (MQTT) and add in all the learnings from the last 40+ years to finally bridge the OT/IT gap and provide plug and play interoperability."

To overcome the disadvantages of legacy protocols, Sparkplug uses a modern publish-subscribe based architecture, as depicted in picture 6.



Picture 6: Publish / Subscribe Architecture

This MQTT based architecture has the following advantages:

- **Report by Exception:** Data and state are published only when something changes.
- **Minimal compute intensive:** Devices and applications decide themselves when they send data and no compute cycles are wasted unnecessarily.
- **Scales to hundreds of thousands or even millions of devices** with multiple billions of tag data and state changes per day with a single broker (cluster).
- **Push based communication**
- **Completely decoupled:** To change, add, or remove data consumers or producers, no other components need to be changed.

Chapter 7 - Use Case Examples of Implementing Sparkplug in the Real-World

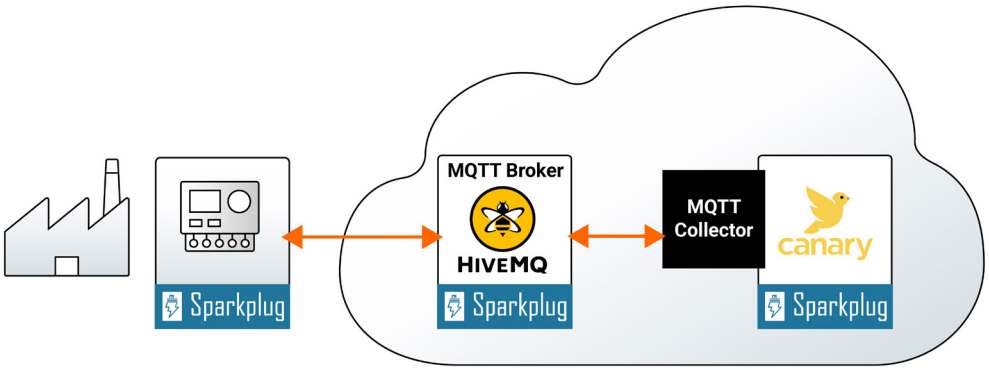
In an industry 4.0 setup, such as a smart factory and smart manufacturing ecosystem, integrating and sharing data between heterogeneous systems and services is a huge challenge. One way to overcome this challenge is adopting MQTT Sparkplug open-source IoT specification in the hardware systems and the edge devices. For the reason that, Sparkplug provides MQTT clients the framework to seamlessly integrate industrial data across the entire ecosystem of hardware vendors and application providers, like SCADA, MES, and Historian vendors, within the MQTT infrastructure in a bi-directional and interoperable way.

How to Practically Implement MQTT Sparkplug in Your IoT Applications?

We will showcase two use case examples, which will demonstrate how easy it is to connect different Industry 4.0 products using MQTT Sparkplug specification and a cloud MQTT broker.

Use Case Example 1 – Connecting a Smart Factory to the Cloud Using Sparkplug

In a smart factory and smart manufacturing ecosystem, integrating and sharing data between heterogeneous systems and services is a huge challenge. One way to overcome this challenge is adopting MQTT Sparkplug open-source IoT specification in hardware systems and edge devices. Sparkplug provides MQTT clients the framework to seamlessly integrate industrial data across the entire ecosystem of hardware vendors and application providers, like SCADA, MES, and Historian vendors, within the MQTT infrastructure in a bi-directional and interoperable way.
So, with the help of Sparkplug, you can connect an edge device on the plant floor, such as Opto 22 groov Rio using HiveMQ Cloud MQTT broker that supports MQTT Sparkplug specification and then forward the data to Canary Cloud for analysis as shown in picture 7.

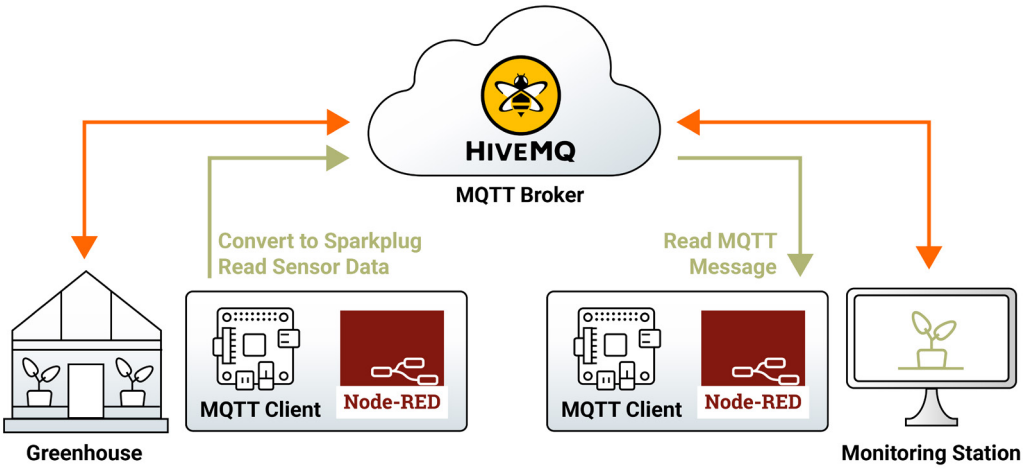


Picture 7: Using Sparkplug to Connect a Smart Factory to the Cloud

This use case example not only shows how Sparkplug helps connect a Smart Factory to the cloud but also shows how it can bring plug-and-play interoperability while integrating old existing OT infrastructure and the current software applications.

Use Case Example 2 – Monitoring Greenhouse Data Remotely Using Sparkplug

This use case example showcases how to use the MQTT Sparkplug specification to make greenhouse information discoverable by industry 4.0 applications that may join an MQTT network.



Picture 8: Using Sparkplug to Remotely Monitor a Greenhouse Data

Picture 8 depicts a simulation of a simple greenhouse remote monitoring system using Raspberry Pis, Node-Red, and HiveMQ Cloud as the MQTT broker. In the setup, there are two Raspberry Pis. One of the Raspberry Pis will act as a greenhouse control unit measuring humidity and temperature. We will use a DHT11 sensor for that. The same Raspberry Pi will also monitor unauthorized access using a proximity sensor. The other Raspberry Pi will act as a remote monitoring station, which receives greenhouse telemetry data and displays the temperature and humidity data on an HMI. The second Raspberry Pi will also monitor the intrusion and switch ON an AC lamp when an intrusion is detected. To monitor the telemetry data, we will send and receive MQTT Sparkplug messages using Raspberry Pi, Node-RED, and HiveMQ Cloud.

With the help of the above two use case examples, you now have a gist of how real-world implementation of Sparkplug looks like.

Chapter 8 – Conclusion

When you adopt Sparkplug specification, you bring in a common language to establish communication between MQTT supporting devices, non-MQTT devices that support protocols like OPC-UA or Modbus, and the software. This brings plug-and-play interoperability to IIoT.

The increasing popularity of Sparkplug in most industries can be explained with the clear advantages over legacy protocols and the ability to even integrate these legacy protocols in a Sparkplug architecture.

HiveMQ and Sparkplug

HiveMQ provides the MQTT broker platform required for any Sparkplug architecture. HiveMQ offers the following capabilities to Sparkplug deployments:

- 100% compatible with MQTT 3.1.1 and MQTT 5
- An MQTT broker for business critical systems that is reliable and scalable.
- Easy integration with OT and IT systems with an extension SDK.
- The ability to deploy on-premise, on Microsoft Azure or AWS, or with HiveMQ Cloud.

HiveMQ is also a member of the Sparkplug working group. Get in touch with us if you need assistance in implementing Sparkplug in your IoT use case.

HiveMQ Partners in Sparkplug

Contact us:

Contact us info@hivemq.com
Find out more hivemq.com
Subscribe to our Newsletter

HiveMQ
Ergoldingerstr. 2a
84030 Landshut
Germany
hivemq.com

HIVEMQ