# Reimagining Data Exchange: OPC UA to MQTT and OCPP Conversion for Mobile Charging Stations

Developing Node-RED flows for converting OPC UA data to MQTT Sparkplug B and OCPP messages, and back again, deployable on remote terminal units for mobile charging stations at emission-free construction sites.

VICTORIA VIUM LUND

SUPERVISORS
Harsha Sandaruwan Gardiyawasam Pussewalage
Indika Anuradha Mendis Balapuwaduge

**University of Agder, 2024**
Faculty of Engineering and Science
Department of Information and Communication Technology

## Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

| | | |
|---|---|---|
| 1. | Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen. | Ja |
| 2. | **Vi erklærer videre at denne besvarelsen:**<br>• Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands.<br>• Ikke refererer til andres arbeid uten at det er oppgitt.<br>• Ikke refererer til eget tidligere arbeid uten at det er oppgitt.<br>• Har alle referansene oppgitt i litteraturlisten.<br>• Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse. | Ja |
| 3. | Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31. | Ja |
| 4. | Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert. | Ja |
| 5. | Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk. | Ja |
| 6. | Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider. | Ja |
| 7. | Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet. | Nei |

## Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

| | |
|---|---|
| Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering: | Ja |
| Er oppgaven båndlagt (konfidensiell)? | Nei |
| Er oppgaven unntatt offentlighet? | Nei |

# Acknowledgements

I would like to extend my deepest gratitude to all those who have supported me throughout the course of my master's thesis. First and foremost, I would like to thank my supervisor, Harsha Sandaruwan Gardiyawasam Pussewalage, and co-supervisor, Indika Anuradha Mendis Balapuwaduge, for their support, valuable guidance, and constant encouragement throughout my research. I am also grateful to Nordic Booster for allowing me access to their offices to work, and Theodor Skaufel, of Nordic Booster, for his project suggestion, support, technical guidance, and troubleshooting help.

I am indebted to my classmates and friends, for the brainstorming sessions, and their constructive criticisms, stimulating discussions, moral support, and the occasional distractions. Additionally, I am immensely grateful for the support and resources provided by the University of Agder throughout my master's thesis journey. The conducive work environment and access to academic facilities have been instrumental in the successful completion of this work.

I would like to express my heartfelt thanks to my family and friends for their unwavering support and understanding throughout this journey. To my parents, for their continuous encouragement and belief in me, and to my partner, Ingvild, for her patience, insights, and unwavering support. Lastly, I extend my thanks to anyone who, in one way or another, contributed to the successful completion of this thesis. Your support and encouragement have been greatly appreciated, and I would also like to acknowledge ChatGPT for its invaluable assistance with grammar and sentence structure, verbal prompt assistance, and inspiration throughout the writing process.

# Abstract

As the adoption of emission-free construction sites becomes more prevalent, the need for interoperable mobile charging stations increases. This thesis, titled 'Reimagining Data Exchange: OPC UA to MQTT and OCPP Conversion for Mobile Charging Stations,' explores a novel approach to enabling efficient and reliable data communication between different protocols within said charging stations. Using Node-RED technology, the project creates data flows converting OPC UA data to MQTT Sparkplug B and OCPP messages, enhancing station interoperability and adaptability.
The results demonstrate the system's intended functionality, showcasing Node-RED's capability to convert communication protocols effectively. Despite being tested solely in controlled environments, the system's feasibility in addressing communication protocol challenges is evident. However, limitations include reliance on local testing environments and absence of testing with live systems and actual monitoring platforms, potentially impacting real-world applicability.
Nevertheless, the system's validity is asserted, offering a solution to convert communication protocol messages for mobile charging stations and their further development. Future work will focus on extensive testing with live systems, integration of other communication protocols, and automation improvements. This thesis contributes a practical and cost-effective solution, supporting the transition towards sustainable construction practices.

# Contents

# List of Figures

# List of Tables

# Acronyms

**CBID** Chargebox Identity

**CS** Charging Station

**CSMS** Charging Station Management System

**EoN** Edge of Network

**IIoT** Industrial Internet of Things

**IoT** Internet of Things

**M2M** Machine to Machine

**OCPP** Open Charge Point Protocol

**OPC UA** Open Platform Communications Unified Architecture

**QoS** Quality of Service

**RTU** Remote Terminal Unit

# Chapter 1

# Introduction

As the world moves towards more sustainable practices, the construction industry faces the challenge of reducing its environmental impact. One promising solution is the adoption of emission-free construction sites, which rely heavily on mobile charging stations for electric vehicles and equipment [1]. This thesis, titled "Reimagining Data Exchange: OPC UA to MQTT and OCPP Conversion for Mobile Charging Stations," explores a novel approach to facilitating efficient and reliable data communication between different protocols used in and with these charging stations. This project leverages Node-RED to create data flows that convert OPC UA data to MQTT Sparkplug B and OCPP messages, and vice versa. By enhancing the interoperability of these stations, this solution ensures they are not locked to one or two specific communication protocols, thereby increasing their adaptability and integration capabilities in construction site environments. The outcome aims to enhance the operational efficiency and flexibility of mobile charging stations, supporting the broader goal of sustainable and emission-free construction practices.

## 1.1 Motivation

The inspiration for this project arose from a suggestion from Nordic Booster, a company dedicated to promoting innovative and sustainable solutions in the construction industry [2]. Recognizing the lack of options to easily integrate and communicate between various protocols used in and around mobile charging stations, Nordic Booster identified a need to create a lightweight, open-source, easily adaptable, and free solution to streamline data communication between various protocols used in these stations.
Nordic Booster was originally interested in converting various communication protocols (Modbus, OPC UA, etc.) to MQTT Sparkplug B, as that's what their monitoring systems communicate on [3][4][5]. However, Klimaetaten ved Oslo Kommune[1] recently published a paper that recommends the communication between mobile charging stations and charging station platforms in Oslo municipality (where Nordic Booster is based) is done via OCPP [6], which in turn changed the scope of the project to be converting OPC UA to MQTT Sparkplug B and OCPP. This project aims to address this challenge by leveraging Node-RED to create an efficient, lightweight, and open-source system that converts OPC UA data to MQTT Sparkplug B and OCPP messages and vice versa.

## 1.2 Problem Statement

The primary focus of this project is to address the challenge of converting OPC UA data to MQTT Sparkplug B and OCPP messages, for use in mobile charging stations at emission-free construction sites, using an open-source, low-cost (free) and lightweight program. This involves designing and implementing an open source system that can seamlessly bridge the

[1]The Climate Agency of Oslo municipality

gap between OPC UA, OCPP, and MQTT protocols, enabling real-time data exchange and control in a reliable and efficient manner.

## 1.3 Objectives

1. Convert OPC UA data to MQTT Sparkplug B messages using Node-RED
2. Convert MQTT Sparkplug B messages to OPC UA messages using Node-RED
3. Convert OPC UA data to OCPP messages using Node-RED
4. Convert OCPP messages to OPC UA messages using Node-RED

## 1.4 State of the Art

There are several existing solutions that attempt to bridge the gap between different industrial protocols. For example, commercial products such as Kepware's KEPServerEX [7] and Inductive Automations's Ignition [8] offer connectivity solutions that integrate OPC UA with various other protocols, including MQTT and MQTT Sparkplug B. However, these solutions are often proprietary, costly, and may not provide the flexibility needed for specific use cases like mobile charging stations.
Then there is OPC UAs *PubSub* communication model, which utilizes the *publish subscribe* pattern to transport information either with UDP transport or with the use of messaging protocols (MQTT, etc.). Which could offer a possible solution, but only between OPC UA and MQTT and MQTT Sparkplug B, and not OCPP or other communication protocols [9].
However, comprehensive solutions that seamlessly integrate OPC UA, MQTT Sparkplug B, and OCPP in a unified framework tailored for mobile charging stations remain limited.

## 1.5 Structure of Thesis

The structure of the rest of this report is as follows:
**Chapter 2 - Background**
Chapter 2 provides background information on key technologies, including Node-RED, OPC UA, OCPP, and MQTT.
**Chapter 3 - Methodology**
Chapter 3 outlines the methodology employed in the development and implementation of the proposed solution.
**Chapter 4 - System Design**
Chapter 4 presents the system design and architecture, detailing the specific components and their interactions
**Chapter 5 - Results**
Chapter 5 discusses the results of the system, including testing and validation procedures.
**Chapter 6 - Discussion**
Chapter 6 offers a discussion of the findings, limitations of the solution.
**Chapter 7 - Conclusion**
Chapter 7 concludes the thesis and discusses future directions of the system.

# Chapter 2

# Background

This chapter explores the foundational concepts and technologies pertinent to this project. It begins with an overview of mobile charging stations, examining their purpose and functionality. Next, the chapter will delve into the communication protocols OPC UA, MQTT, MQTT Sparkplug B, and OCPP. Lastly, it will introduce Node-RED, a flow-based development tool, and some of its features.

## 2.1 Mobile Charging Stations for Emission-Free Construction Sites

As the construction industry seeks to reduce its environmental impact, the adoption of electrical construction machinery has increased and with it the energy and power demand at construction sites [1], thereby increasing the need for charging solutions that can be used anywhere and that reduce the charging time. That's where mobile and battery-powered charging stations come into play.

Mobile charging stations are portable units equipped with high-capacity batteries or generators and often feature multiple charging ports and fast-charging capabilities. The main advantage of the stations are that they are portable, meaning they can be moved around to different locations without being dependent on the power grid, making them great for remote construction sites. There are a few companies in Norway that produce mobile charging stations, Kverneland Energi [10], Aneo [11], and Hafslund Boost [12], among others. Nordic Booster, however, were the first to develop mobile fast charging station [2], and have been a part of a pilot project for emission-free construction sites in Oslo [1].

Mobile charging stations are a key component in the move towards emission-free construction sites. They offer significant environmental and operational benefits, though challenges such as initial costs and battery performance must be managed. As technology advances and the industry increasingly prioritizes sustainability, mobile charging stations are set to become an integral part of modern construction practices.

**Figure 2.1:** Example of a mobile charging station - Nordic Booster's Hummingbird [13]

## 2.2 OPC UA: Open Platform Communications Unified Architecture

OPC UA is a machine-to-machine, "platform-independent, service-oriented architecture" [14], developed by the OPC Foundation, that provides a set of specifications for the standardization of communication in industrial automation. It was built on OPC Classic and integrates all the different parts of OPC Classic into one framework. The protocol uses an information model to define the structure and semantics of the data, including types, relationships, and behaviors of objects in a hierarchical manner [15]. This model is implemented and represented in the address space, which is the framework where all the data, metadata, and their relationships are structured [16]. Each entity in the address space is identified by a unique *NodeId*, which ensures that every node can be precisely addressed and accessed, facilitating reliable communication and data exchange.
OPC UA has two different communication models, the *Client Server* model and the *PubSub* model, though the latter is still relatively new [17]. The OPC UA *Client Server* model uses a request/response pattern, where the client can access information from a server using standardized sets of *services* defined in the OPC UA specification [18]. The specification defines several transport protocols that the *Client Server* model can use to communicate, such as TCP, SOAP/HTTP, WebSockets, etc. [19]. A client can connect to multiple servers and vice versa, and an OPC UA application can also combine the server and client components, allowing it to interact with both servers and clients [20].

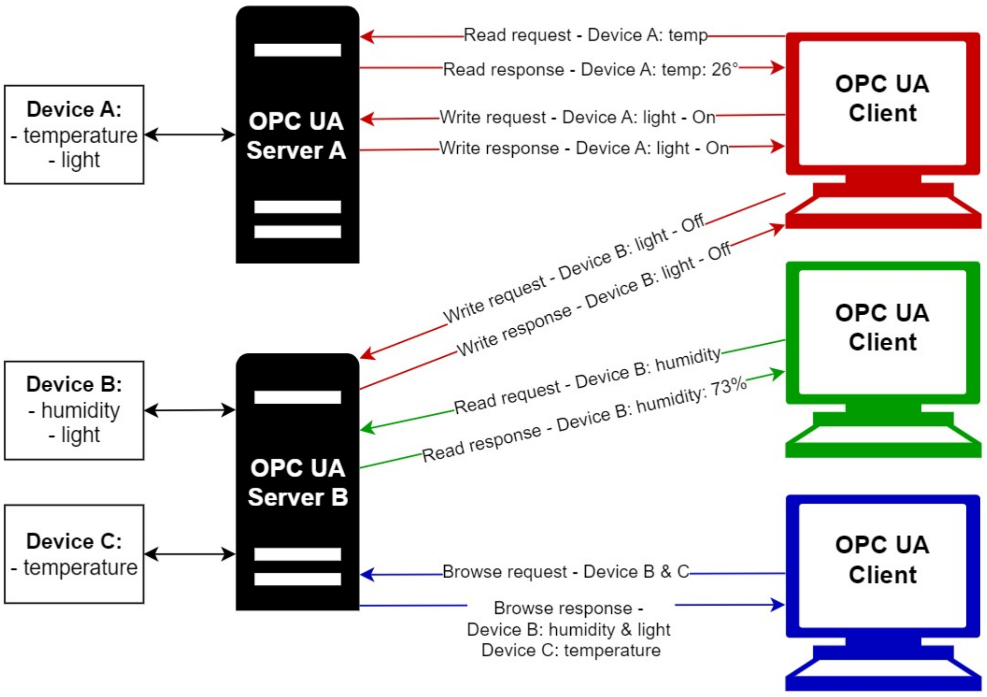

Device A:
- temperature
- light
OPC UA Server A
Read request - Device A: temp
Read response - Device A: temp: 26°
Write request - Device A: light - On
Write response - Device A: light - On
OPC UA Client
Write request - Device B: light - Off
Write response - Device B: light - Off
OPC UA Client
Read request - Device B: humidity
Read response - Device B: humidity: 73%
Device B:
- humidity
- light
OPC UA Server B
Device C:
- temperature
Browse request - Device B & C
Browse response -
Device B: humidity & light
Device C: temperature
OPC UA Client

**Figure 2.2:** OPC UA *Client Server* architecture

The client starts by opening a connection to the Server, creating a *session*, it can then request different services from the server, such as *Read, Write, Subscribe* and *Browse*. When the client is done, it closes the session and disconnects [21]. Usually in *Client Server* models, every time one wants to read data from the server, the client must request it, quickly becoming tedious, unless an automatic read request trigger is created. Which is the basic concept of the subscription service (not to be confused with subscribe in *PubSub*). The client creates a subscription which includes a set of *MonitoredItems* and a publishing interval. When the subscription has been created, the client will send *Publish* requests at the publishing interval

specified, and if there are *Notifications* to report (changes in data), the server with send a *notificationMessage* in response [18][21][9].
OPC UA ensures secure and reliable data exchange and is widely adopted due to its robustness and versatility in various industrial applications. It supports complex data structures and offers high-level communication between devices and systems. However, its implementation complexity and resource requirements pose challenges for its use in resource-constrained environments like mobile charging stations [9][14][21].

## 2.3 MQTT and Sparkplug B

"MQTT[1] is a Client Server publish/subscribe messaging transport protocol" [24, p. 1][25, p. 1]. Originally it was created as a way of connecting with oil pipelines via satellite with minimal battery loss and minimal bandwidth, until 2010 when MQTT v3.1 was released as a royalty-free version, where anyone can use the protocol [23]. Since then, MQTT became an officially approved OASIS standard (2014), and several versions have been released, MQTT-SN (2013) [26], MQTT v3.1.1 (2014) and MQTT v5.0 (2017) [27]. MQTT is lightweight and bandwidth efficient, easy to implement, and open, making it an ideal protocol to use for Machine to Machine (M2M) and Internet of Things (IoT) contexts [24][25]. Due to this it has a wide variety of use cases and is used in many different industries, including automotive, manufacturing, smart home and oil & gas [28].

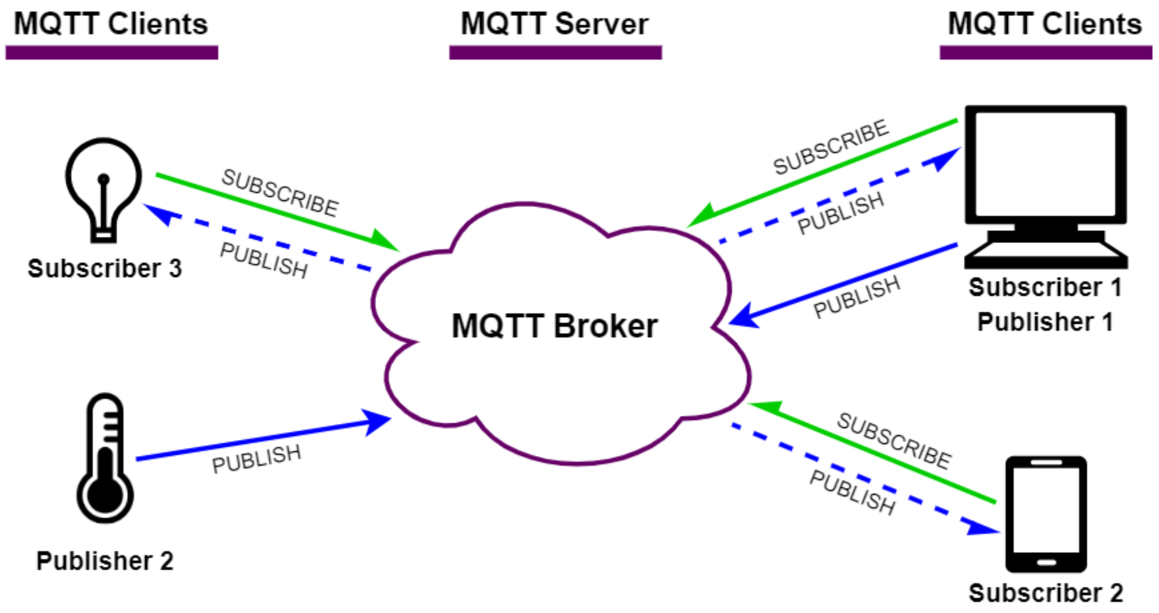


**Figure 2.3:** MQTT client server publish/subscribe model architecture

MQTT's client-server publish/subscribe model means that clients communicate through a server, known as a broker, using a publish/subscribe pattern (figure 2.3 shows the architecture of the client server publish/subscribe model). The publish/subscribe pattern decouples clients from each other, they are never in direct contact with each other and are unaware that other clients exist, this is instead handled by the broker. An MQTT client is "a program or device that uses MQTT" [24, p. 9][25, p. 11], and it can initiate and close connections to the server (broker), send messages (publish to topics), and receive messages (subscribe to topics). An MQTT client can act as both a publisher and a subscriber [24][25][23]. An MQTT

[1]MQTT used to stand for *MQ Telemetry Transport* or *Message Queuing Telemetry Transport*, however, since 2013 it is no longer considered an acronym and is now only the name of the protocol [22][23]

broker, on the other hand, is "a program or device that acts as an intermediary between clients which publish application messages and clients which have made subscriptions" [24, p. 9][25, p. 11]. The broker manages client connections, receives messages from publishers, filters messages based on the topic, and distributes messages to the correct subscribers according to the specified Quality of Service (QoS) [24][25][23]. MQTT offers three levels of QoS: QoS 0 (at most once delivery), QoS 1 (at least once delivery), and QoS 2 (exactly once delivery), each providing different levels of delivery assurance. For MQTT, QoS is between the client and the broker and not between the sending client and the receiving client(s), which means that publishers define the QoS level of the message that is to be sent to the broker and subscribers define the QoS level of the message that is to be sent to them by the broker [24][25][23].
Topics in MQTT are strings that are attached to messages as a type of label, they are hierarchical and contain one or more levels, separated by forward slashes (shown in figure 2.4). A topic does not need to be initialized before it is published or subscribed to, clients can start publishing and subscribing as soon as they are connected to the broker. Clients can subscribe to an exact topic or multiple topics simultaneously using wildcards, single-level and multi-level. A single-level wildcard, represented by a '+', replaces one topic level. It can be placed at any level, and it can be used more than once in a single topic and can be combined with the multi-level wildcard. A multi-level wildcard, represented by a '#', covers many topic levels. It can be used on its own or behind a topic level separator, but it must be the last character in the topic [24][25][23].

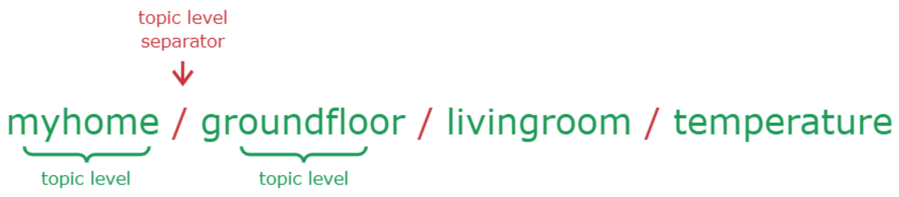


**Figure 2.4:** MQTT topic example [23]

The operation of MQTT begins with clients establishing a connection to the broker using the TCP/IP protocol. The client sends a CONNECT message to the broker, including a unique client identifier, and the broker responds with a CONNACK message to acknowledge the connection. When publishing messages, the publisher sends a PUBLISH message to the broker containing the topic and the message payload. The broker then routes this message to all subscribers of the topic. Subscribers indicate their interest by sending a SUBSCRIBE message to the broker, specifying the topics they are interested in. The broker confirms this subscription with a SUBACK message. Upon receiving messages published to a subscribed topic, the broker sends a PUBLISH message to the subscriber. Figure 2.3 shows the processes of publishing and subscribing. Clients can gracefully terminate the connection by sending a DISCONNECT message to the broker, which then cleans up any state information associated with the client.

### 2.3.1 Sparkplug B

MQTT Sparkplug is an extension of the MQTT protocol specifically designed for Industrial Internet of Things (IIoT). While MQTT is highly effective for transmitting data, it lacks standardization for the data formats and topics, meaning that all devices/applications wishing to communicate via MQTT must know where to subscribe to the data, which can lead to interoperability challenges in complex industrial environments [29][30][31].
Sparkplug B[2] addresses these challenges by defining a standard payload format, topic names-

[2]Sparkplug A is deprecated, so "Sparkplug" and "Sparkplug B" will be used interchangeably throughout the report [30]

pace for MQTT messages and MQTT state management, ensuring that all devices using Sparkplug B can interpret the data correctly, fostering interoperability and simplifying the integration process. It includes mechanisms for device birth and death certificates, which inform the system when devices come online or go offline, enhancing the reliability and robustness of the network [29][30]. In operation, Sparkplug B works by specifying how devices should publish their data to an MQTT broker and how other devices should subscribe to these topics to receive the data. The payload format is based on Google's Protocol Buffers, providing a compact, efficient way to serialize structured data [30].

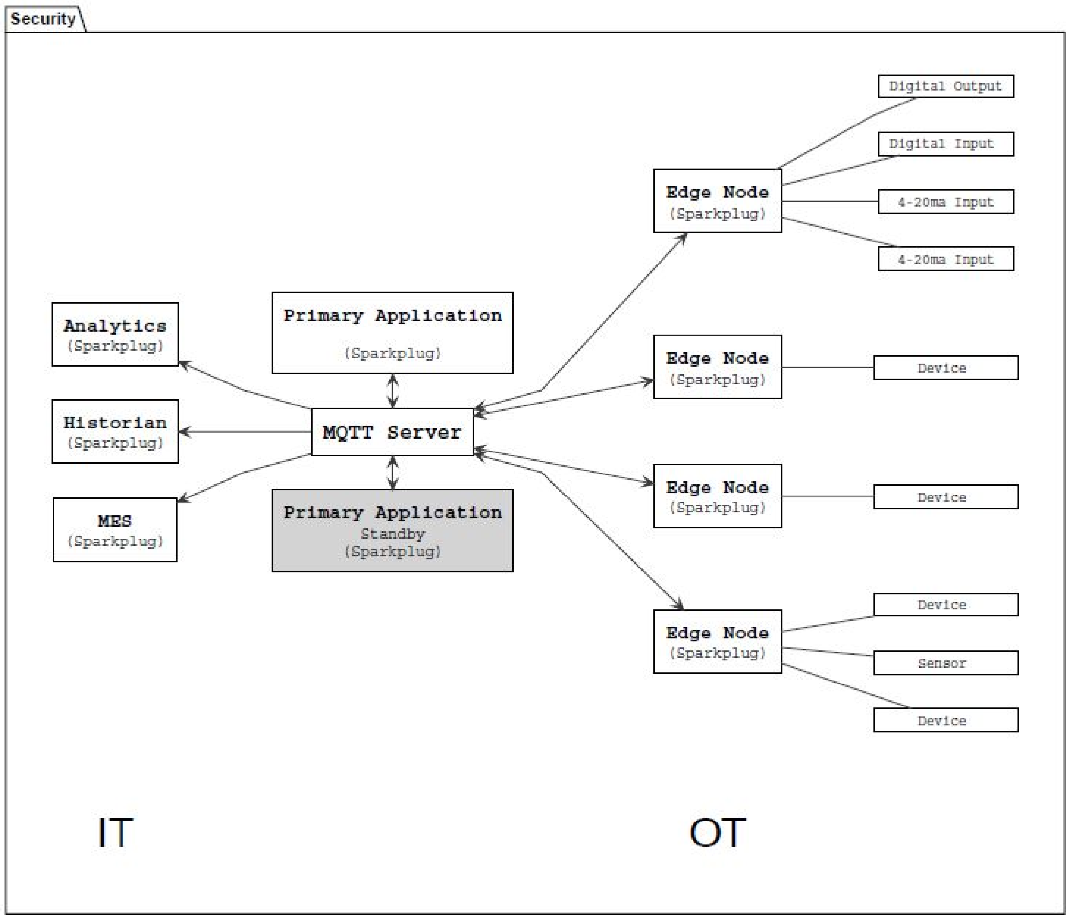


**Figure 2.5:** MQTT Sparkplug B architecture [30]

To use MQTT Sparkplug B, the broker being used must implement MQTT v3.1.1, because Sparkplug requires QoS level 0 and 1, retained messages and *Last Will and Testament* [29]. The architecture of Sparkplug B is a bit more in depth than MQTT's architecture, which can be seen in figure 2.5. The central element of the Sparkplug architecture is the MQTT Server (broker), which functions the same as in MQTT. Then there are the MQTT edge nodes, which represent physical/virtual devices at the edge of the network, such as sensors, gateways, etc. These edge nodes communicate with the MQTT server by publishing data to designated topics and subscribing to topics to receive commands or updates. The primary host application is an MQTT client application responsible for processing and analyzing the data collected from MQTT edge nodes and devices (can also be reffered to as the SCADA host or IIoT Host). The Sparkplug host application refers to the software application or system that implements the Sparkplug B specification for handling MQTT messages within the IIoT environment. This application interprets the Sparkplug B payload format, enforces the Sparkplug B topic namespace, and manages device lifecycle events such as device birth and death certificates. The Sparkplug host application is any Sparkplug MQTT client "that consumes the real-time Sparkplug messages or any other data being published with proper permission and security" [30]. It can be thought of as a software platform that integrates data from divers IoT devices [29][30].

## 2.4 OCPP 2.0.1: Open Charge Point Protocol 2.0.1

The OCPP is "the industry-supported de facto standard for communication between a Charging Station (CS) and a Charging Station Management System (CSMS)" [32], and an open standard. Developed by the Open Charge Alliance, OCPP allows for the seamless exchange of information regarding the status, usage, and maintenance of EV charging stations. It supports a variety of functions such as remote monitoring, diagnostics, and firmware updates, which helps in managing charging stations effectively and efficiently.
OCPP works by establishing a set of rules and data formats that both CS and CSMS adhere to [33]. When an EV is connected to a CS, the station communicates with the CSMS using OCPP messages. These messages can include requests for authorization, charging status updates, meter readings, and error reports. The CSMS, in turn, can send commands to the charging station for actions such as starting or stopping a charging session, locking or unlocking the connector, and updating the station's firmware [33]. By using OCPP, charging network operators can integrate various brands and models of charging stations into a unified system, ensuring compatibility and flexibility across different manufacturers and regions [32].
The protocol operates on a *client server* architecture (figure 2.6) and defines two roles: Charging Station (client) and Charging Station Management System (server), who, as of OCPP 2.0.1, communicate over WebSocket using JSON [34].

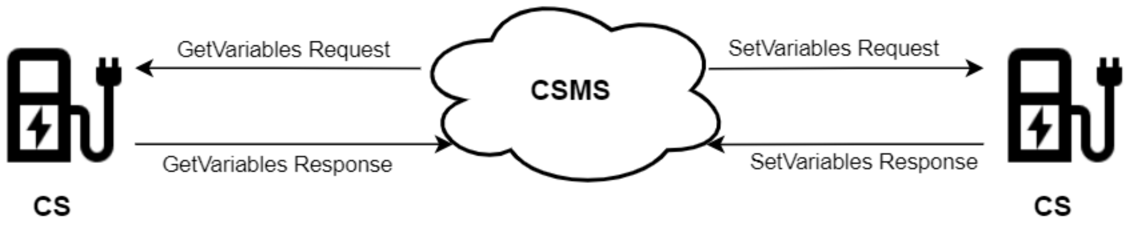


**Figure 2.6:** OCPP client server architecture

## 2.5 Node-RED: A Programming Tool for Flow-Based Development

Node-RED is a flow-based programming tool for wiring together hardware devices, APIs, and online services in a flow-based development environment. It provides a browser-based interface for creating event-driven applications by connecting pre-built nodes (black box processes), which represent different functions or services, in a flowchart-like manner. It can be further customized by installing new nodes created by the community [35]. Some of these node sets or packages are *node-red-contrib-opcua*, *MQTT-Sparkplug-Plus*, and *@anl-ioc/node-red-contrib-ocpp2*, which are used in this project. The nodes in the node sets and their functions are described below, chapter 3 provides a more in depth explanation of how they're used in the system. Node-RED is lightweight, easy to use and can run on different platforms making it particularly useful for IoT projects [36].

### OPC UA in Node-RED

Figure 2.7 shows the nodes included in the node set *node-red-contrib-opcua* and table 2.1 describes their function.

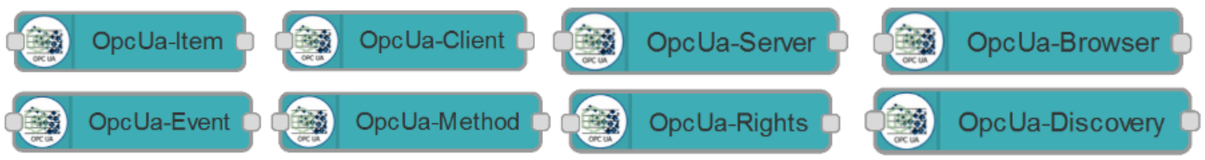


**Figure 2.7:** Nodes in the *node-red-contrib-opcua* node set

| Node | Function |
|---|---|
| OpcUa-Item | Defines an OPC UA item, type and value |
| OpcUa-Client | Creates an OPC UA client that connects to an OPC UA server |
| OpcUa-Server | Creates OPC UA server with own variables, object structures and methods |
| OpcUa-Browser | Browses an OPC UA server or specific items in an OPC UA server |
| OpcUa-Event | Defines the events that will be subscribed from the server |
| OpcUa-Method | Calls methods from the server |
| OpcUa-Rights | Sets object/variable access level, user access level and permissions |
| OpcUa-Discovery | OPC UA Discovery server for OPC UA servers to register themselves |

**Table 2.1:** Overview of the nodes included in the *node-red-contrib-opcua* node set and a simple description of their function [37][38]

### MQTT Sparkplug B in Node-RED

*MQTT-Sparkplug-Plus* has 3 nodes: *mqtt sparkplug device*, *mqtt sparkplug in*, and *mqtt sparkplug out* (figure 2.8), table 2.2 describes their function.

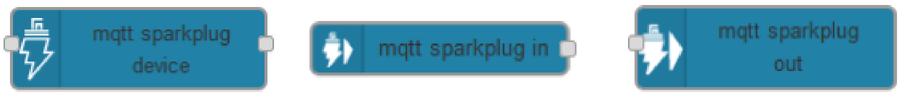


**Figure 2.8:** Nodes in the *MQTT-Sparkplug-Plus* node set

| Node | Function |
|---|---|
| mqtt sparkplug device | Acts as an MQTT Edge of Network Node |
| mqtt sparkplug in | Acts as a subscriber |
| mqtt sparkplug out | Acts as a publisher |

**Table 2.2:** Overview of the nodes available in the *MQTT-Sparkplug-Plus* node set and a simple description of their function [39]

### OCPP 2.0.1 in Node-RED

The node set *@anl-ioc/node-red-contrib-ocpp2* has two nodes: *CSMS* and *CS* (shown in figure 2.9 and described in table 2.3).

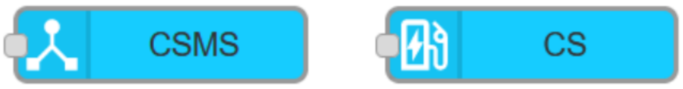


**Figure 2.9:** Nodes in the *@anl-ioc/node-red-contrib-ocpp2* node set

| Node | Function |
|---|---|
| CSMS | OCPP 2.0.1 CSMS messaging node, handles connecting, sending and receiving OCPP 2.0.1 messages to the CS |
| CS | OCPP 2.0.1 CS messaging node, handles connecting, sending and receiving OCPP messages to the CSMS |

**Table 2.3:** Overview of the nodes available in the *@anl-ioc/node-red-contrib-ocpp2* node set and a simple description of their function [40]

# Chapter 3

# Methodology

This chapter will go through the methodology employed in the development and evaluation of the proposed solution. It outlines a high-level overview of the system, the software tools used, implementation details, and how testing and validation was undergone. By providing a systematic overview of the methodology used to create the solution, this chapter aims to ensure the validity, reliability and credibility of the findings.

## 3.1 System Overview

The purpose of the system is to use Node-RED to make it possible to read and write OPC UA data from/to a mobile charging station via different monitoring platforms that communicate via OCPP and MQTT Sparkplug B (figure 3.1 shows the high-level overview of the proposed solution). To achieve this, the system must be able to connect to an OPC UA server, MQTT Sparkplug B broker, and an OCPP CSMS, and receive, transform, and send data in the correct format to/from the different entities. To be able to connect to the entities mentioned above, the system must act as an OPC UA client, MQTT Sparkplug B client, and an OCPP CS. After the system is connected to the different entities, it should then be able to get data from the OPC UA server, convert it to MQTT Sparkplug B and OCPP messages, send those messages, receive MQTT Sparkplug B and OCPP messages, convert them to OPC UA data, and send the converted data to the OPC UA server.

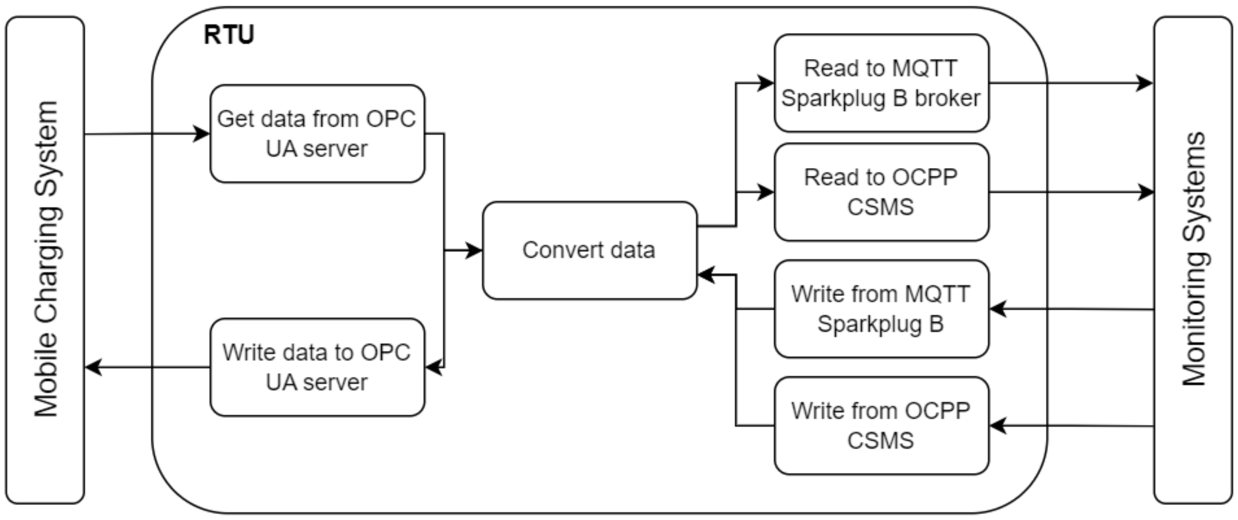

Mobile Charging System
RTU
Get data from OPC UA server
Write data to OPC UA server
Convert data
Read to MQTT Sparkplug B broker
Read to OCPP CSMS
Write from MQTT Sparkplug B
Write from OCPP CSMS
Monitoring Systems

**Figure 3.1:** High-level overview of the proposed solution

## 3.2 Software Tools

### 3.2.1 Node-RED for Flow-Based Programming

Node-RED was used to design, implement and test flows that facilitate the conversion of data between OPC UA, OCPP, and MQTT Sparkplug B. It was primarily used for the implementation of the system, but it was also used to test and validate it (described later in this chapter).
For this project, it was run locally on a Windows machine and was installed and set up using the instructions provided on Node-RED's website [41]. To further configure Node-RED to be compatible with OPC UA, OCPP, and MQTT Sparkplug B and to test the system, the following packages were installed using the palette manager:

- *node-red-contrib-opcua* [42]
- *node-red-contrib-mqtt-sparkplug-plus* [39]
- *@anl-ioc/node-red-contrib-ocpp2* [40]
- *@flowfuse/node-red-dashboard* [43]

The next subsections describe which nodes are used from the first three packages from the above list, and how they were configured. Other than those nodes, the core/standard nodes described in table 3.1 and the dashboard nodes described in table 3.2 were used to include the needed functionality to get, transform and send data between the OPC UA server, MQTT broker, and OCPP CSMS and validate the system.

| Node | Function |
|---|---|
| Inject | Used to trigger a flow, either manually or automatically |
| Function | Used to contain JavaScript code, to be run against messages received by it |
| Debug | "Used to display messages in the debug sidebar" [44] |
| Switch | Used to route messages to different branches of a flow based on the rules set |
| Change | Used to modify the properties of a message, flow context or global context |
| Link in | Used to create virtual wires between flows, connects to any `link out` node |
| Link out | Used to create virtual wires between flows, connects to any `link in` node |
| Delay | Used to delay messages passing through |

**Table 3.1:** Overview of the core/standard Node-RED nodes used [44][35]

| Widget | Function |
|---|---|
| ui-button | Clickable button |
| ui-dropdown | Dropdown menu |
| ui-gauge | Customizable gauge chart |
| ui-switch | Toggle switch |
| ui-text | Non-editable text field |
| ui-text-input | Editable text field, with configurable "type" |

**Table 3.2:** Overview of the *@flowfuse/node-red-dashboard* widget nodes used [43][45]

#### OPC UA with Node-RED

To get the system to communicate with an OPC UA server, only two OPC UA nodes were used: *OpcUa-Browser* and *OpcUa-Client*. The client node is used in two different ways, `SUBSCRIBE`, for getting the data from the OPC UA server, and `WRITE`, for writing to the OPC UA server. To subscribe to the data, the `SUBSCRIBE` client node must be given items, in the correct format, to monitor, which is done by sending an array of node IDs into the input (figure 3.3a shows the structure of the array of OPC UA items to monitor). To write to the OPC UA server, the `WRITE` client node must be given an item, in the correct format, to be written to the server (figure 3.3b shows the structure of an OPC UA item to be written to the OPC UA server). The browser node is used to browse a specific folder on the OPC UA server that has the variables to be subscribed to from the "charging station", by defining the item address (ex. `ns=0;i=85`). This is so the node IDs of the wanted variables can be saved in an array that can be used to with the `SUBSCRIBE` client node.

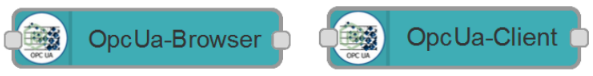
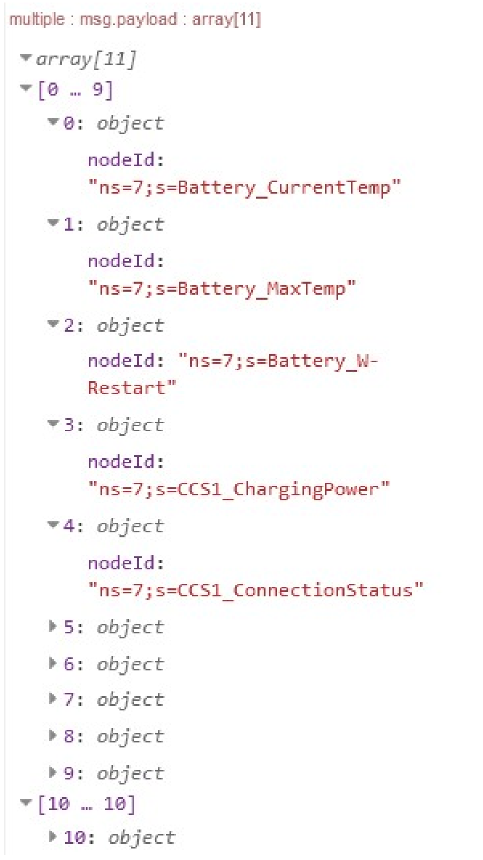
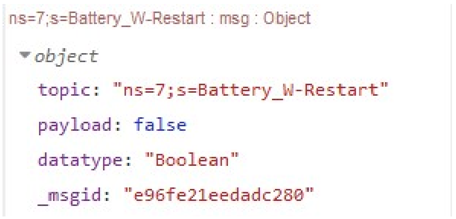


**Figure 3.2:** OPC UA nodes used in solution



```
multiple : msg.payload : array[11]
▾array[11]
▾[0 … 9]
  ▾0: object
     nodeId:
     "ns=7;s=Battery_CurrentTemp"
  ▾1: object
     nodeId:
     "ns=7;s=Battery_MaxTemp"
  ▾2: object
     nodeId: "ns=7;s=Battery_W-
     Restart"
  ▾3: object
     nodeId:
     "ns=7;s=CCS1_ChargingPower"
  ▾4: object
     nodeId:
     "ns=7;s=CCS1_ConnectionStatus"
  ▸5: object
  ▸6: object
  ▸7: object
  ▸8: object
  ▸9: object
▾[10 … 10]
  ▸10: object
```

**(a)** Structure of array of monitored OPC UA items for the OPC UA client to subscribe to

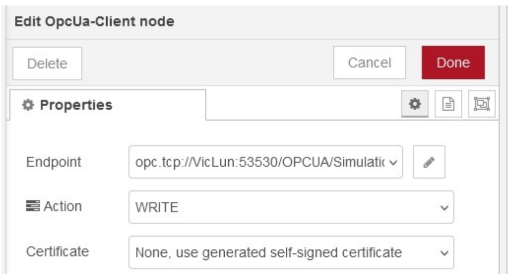


```
ns=7;s=Battery_W-Restart : msg : Object
▾object
  topic: "ns=7;s=Battery_W-Restart"
  payload: false
  datatype: "Boolean"
  _msgid: "e96fe21eedadc280"
```

**(b)** Structure of an OPC UA item to be written to the OPC UA server

**Figure 3.3:** OPC UA client structures

To be able to use the OPC UA browser and client nodes as intended, they have to be configured correctly. The browser node must define the properties "Endpoint", the connection address of the OPC UA server, and "Topic", the OPC UA item address, while the client node must define the properties "Endpoint", the connection address of the OPC UA server, and "Action", the action to be taken by the client. If the client's action is set to `SUBSCRIBE` the property "Interval" must also be defined.

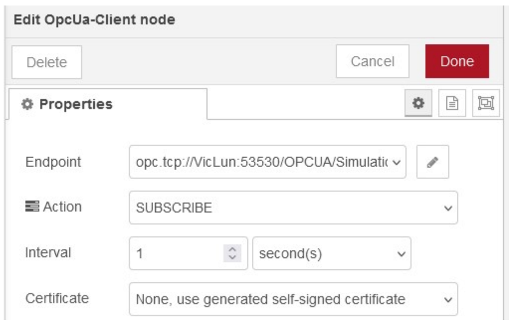

Edit OpcUa-Client node

Endpoint: opc.tcp://VicLun:53530/OPCUA/Simulatio

Action: WRITE

Certificate: None, use generated self-signed certificate

(a) The properties of the OPC UA client node with Write action


Edit OpcUa-Client node

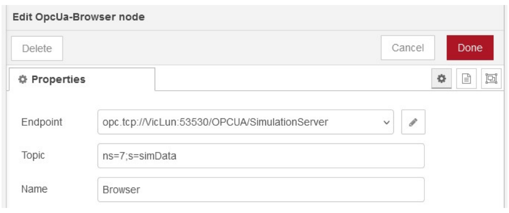
Endpoint: opc.tcp://VicLun:53530/OPCUA/Simulatio

Action: SUBSCRIBE

Interval: 1 second(s)

Certificate: None, use generated self-signed certificate

(b) The properties of the OPC UA client node with Subscribe action


Edit OpcUa-Browser node

Endpoint: opc.tcp://VicLun:53530/OPCUA/SimulationServer

Topic: ns=7;s=simData

Name: Browser

(c) The properties of the OPC UA browser node

**Figure 3.4:** Properties of the OPC UA client and server nodes

For this solution, the "Endpoint" of all the OPC UA nodes was set to the OPC UA simulation server connection address (mentioned later in this chapter), and the "Topic" of the browser node was set to the OPC UA item address of the folder to be browsed. Figure 3.4 shows the properties of the browser and client nodes.

### MQTT Sparkplug B with Node-RED

**Figure 3.5:** MQTT Sparkplug B nodes used in solution

For this system to communicate with an MQTT broker, all the MQTT Sparkplug B nodes were used: *mqtt sparkplug device* for the system and *mqtt sparkplug in* and *mqtt sparkplug out* for testing and validating the system. As mentioned in table 2.2, the Sparkplug device node acts as an MQTT Sparkplug B Edge of Network (EoN) node, which means it acts as both a publisher and subscriber, and it's used to connect the system to the MQTT Broker, to pass the converted OPC UA data between the system and the monitoring system that uses MQTT Sparkplug B to communicate. To be able to publish Sparkplug B messages to the MQTT broker, the Sparkplug device node must be given metrics definitions first, then the metrics, both in the correct format. The correct format for metrics definitions is shown in figure 3.6a, and the correct format for metrics is shown in figure 3.6b. On the other hand, the *mqtt sparkplug in* node acts only as a subscriber, while the *mqtt sparkplug out* node acts only as a publisher (table 2.2), and they are used to validate the system by subscribing to the MQTT broker and publishing any changes made, to the MQTT broker.
To be able to connect the Sparkplug nodes to the MQTT broker, they and the *mqtt-sparkplug-broker* node[1]have to be configured correctly in Node-RED. To configure the MQTT broker node, the MQTT broker's server address and port must be defined (figure 3.7a), the Sparkplug "Name" and "Group" must also be given (figure 3.7b). For the *mqtt sparkplug device* node, configuring it properly involves giving the device a name and choosing the MQTT broker (figure 3.8a). To configure the *mqtt sparkplug in* node, the MQTT

broker needs to be chosen, as well as specifying the topic to subscribe to and the QoS level of the subscription (figure 3.8c). As mentioned in chapter 2, it is not possible to subscribe to specific metrics within topics with Sparkplug B, instead the topic is set to `namespace/group_id` with the multi-level wildcard at the end to make sure all data is received. Configuring the *mqtt sparkplug out* node requires the same properties to be set as the *mqtt sparkplug in* node, as well as setting the retain flag (figure 3.8b). The topic for the publisher node is slightly different from the subscriber node, it uses the entire Sparkplug topic namespace: `namespace/group_id/message_type/edge_node_id/[device_id]`. The message type component `DCMD` is used so that the messages published to the MQTT broker, get further published to the *mqtt sparkplug device*, so that the message gets sent to the OPC UA server.

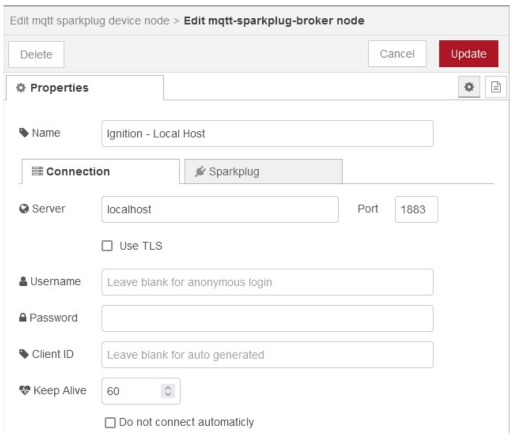
```
msg = {
    "definition" : {
        "TEST/TEST" : {
            "dataType" : "Float"
        },
        "TEST/TEST2" : {
            "dataType" : "Int32"
        },
        "TEST2/TEST" : {
            "dataType" : "Boolean"
        },
    }
};
```

**(a)** Metrics definitions format

```
msg.payload = {
    "metrics": [
        {
            "name": "TEST/TEST",
            "value": 11.34
        },
        {
            "name": "TEST/TEST2",
            "value": 12
        },
        {
            "name": "TEST2/TEST",
            "value": true
        },
    ]
}
```

**(b)** Metrics format

**Figure 3.6:** MQTT Sparkplug B metrics definitions and metrics format



Edit mqtt sparkplug device node > **Edit mqtt-sparkplug-broker node**

Delete | Cancel | Update

**Properties**

Name: Ignition - Local Host

**Connection** | Sparkplug

Server: localhost Port: 1883

☐ Use TLS

Username: Leave blank for anonymous login

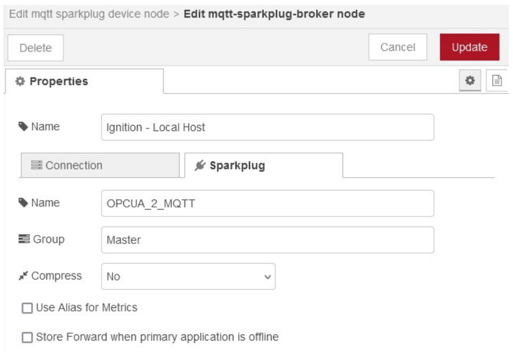
Password:

Client ID: Leave blank for auto generated

Keep Alive: 60

☐ Do not connect automaticly

**(a)** *mqtt-sparkplug-broker* connection properties



Edit mqtt sparkplug device node > **Edit mqtt-sparkplug-broker node**

Delete | Cancel | Update

**Properties**

Name: Ignition - Local Host

Connection | **Sparkplug**

Name: OPCUA_2_MQTT

Group: Master

Compress: No

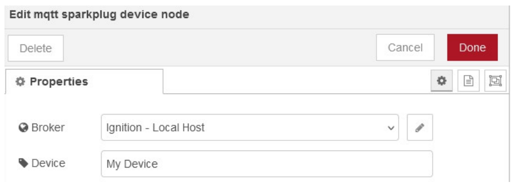
☐ Use Alias for Metrics

☐ Store Forward when primary application is offline

**(b)** *mqtt-sparkplug-broker* Sparkplug properties

**Figure 3.7:** Configuring the MQTT broker in Node-RED

[1]This is not a physical node in Node-RED, it's a property that is available to the entire node set when editing the nodes



Edit mqtt sparkplug device node

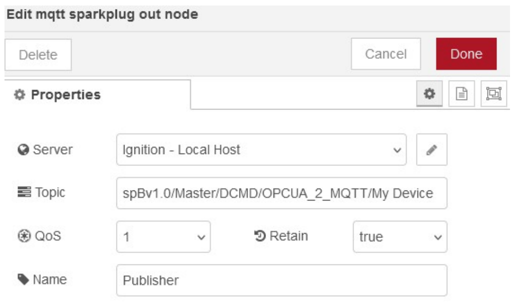
Delete | Cancel | Done

Properties

Broker: Ignition - Local Host

Device: My Device

(a) *mqtt-sparkplug-device* node properties



Edit mqtt sparkplug out node

Delete | Cancel | Done

Properties

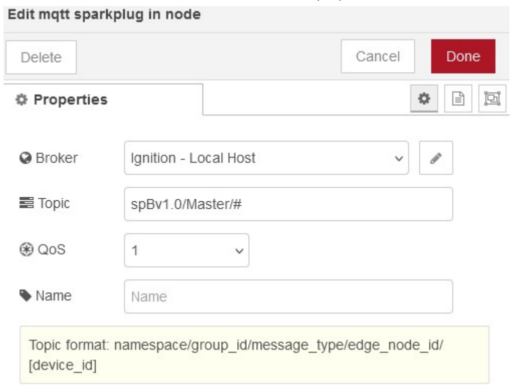
Server: Ignition - Local Host

Topic: spBv1.0/Master/DCMD/OPCUA_2_MQTT/My Device

QoS: 1 | Retain: true

Name: Publisher

(b) *mqtt-sparkplug-out* node properties



Edit mqtt sparkplug in node

Delete | Cancel | Done

Properties

Broker: Ignition - Local Host

Topic: spBv1.0/Master/#

QoS: 1

Name: Name

Topic format: namespace/group_id/message_type/edge_node_id/[device_id]

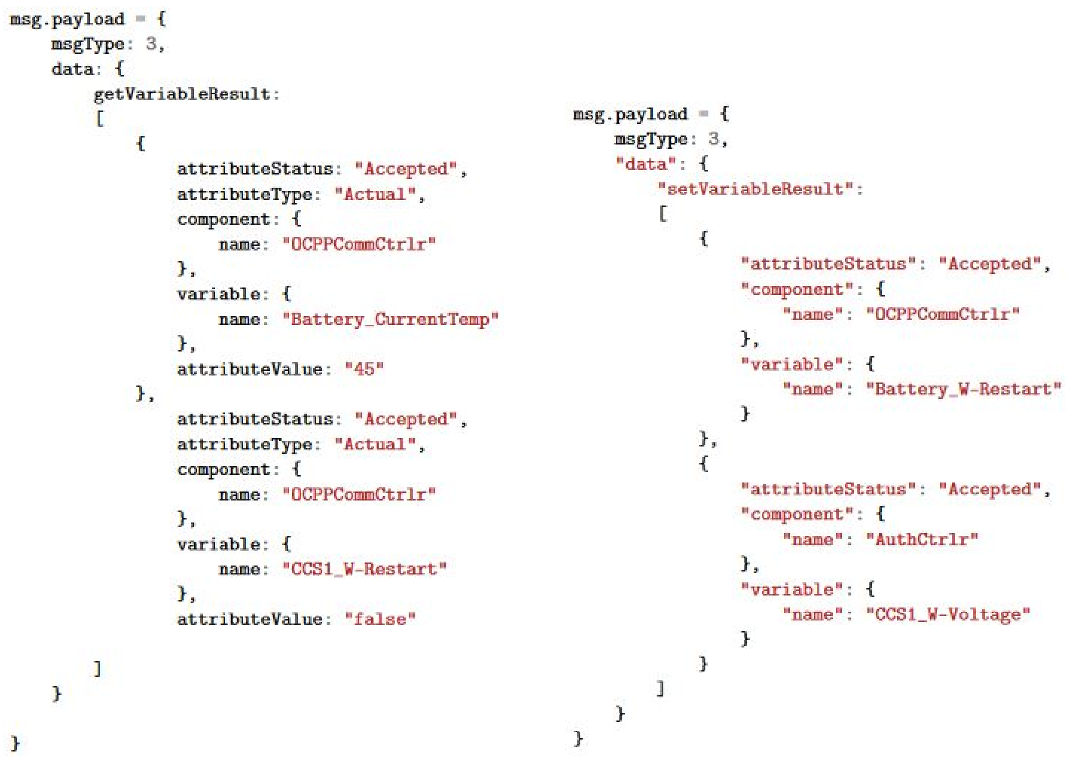
(c) *mqtt-sparkplug-in* node properties

**Figure 3.8:** Properties of the *mqtt-sparkplug-device*, *mqtt-sparkplug-in* and *mqtt-sparkplug-out* nodes

### OCPP with Node-RED

**Figure 3.9:** OCPP nodes used in solution

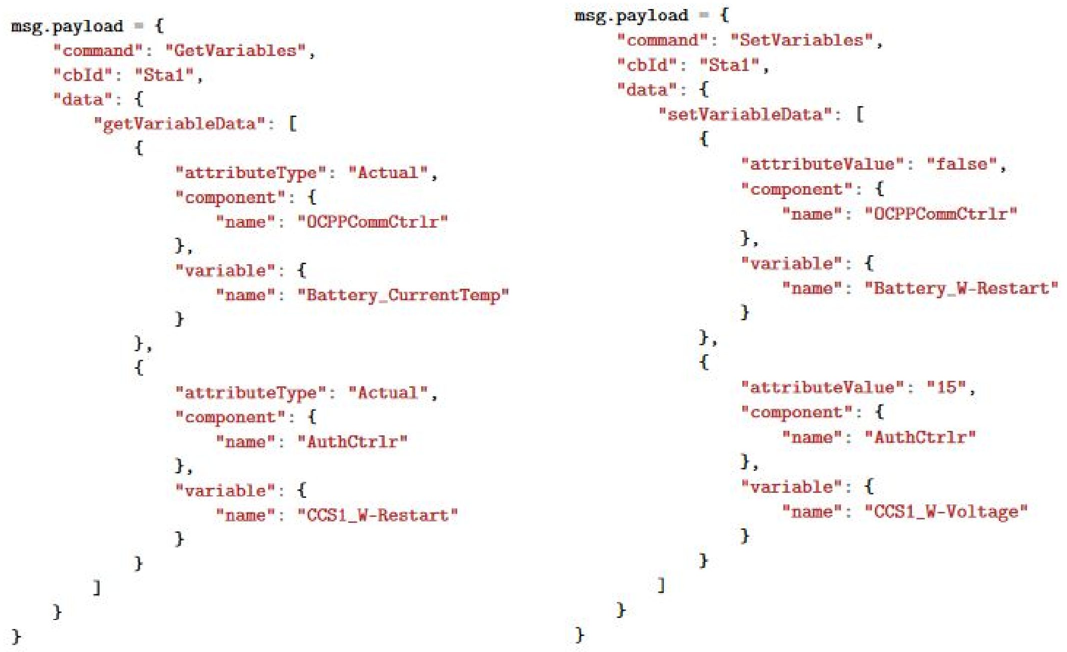
For this system to communicate over OCPP, both the nodes from the *@anl-ioc/node-red-contrib-ocpp2* package are used (figure 3.9. The CS node is used to so that the system can take on the role of a CS and receive and respond to requests made by the CSMS, with converted OCPP data. Receiving a request is handled automatically by the CS node, responding to a request, on the other hand, requires that the "respond" messages follow the correct OCPP structure based on the request. If it's the `GetVariables` request, the response message must follow the `getVariableResult` structure shown in figure 3.10a and if it's the `SetVariables` request, the response message must follow the `setVariableResult` structure shown in figure 3.10b.
The CSMS node is used to test the system by taking on the role of a CSMS, so it can send requests to the system's CS. Sending a request to the CS node, requires the "request" messages to follow the correct OCPP structure based on the type of request it's sending. If it's the `GetVariables` request, the message must follow the `getVariableData` structure shown in figure 3.11a and if it's the `SetVariables` request, the message must follow the `setVariableData` structure shown in figure 3.11b.


```
msg.payload = {
    msgType: 3,
    data: {
        getVariableResult:
        [
            {
                attributeStatus: "Accepted",
                attributeType: "Actual",
                component: {
                    name: "OCPPCommCtrlr"
                },
                variable: {
                    name: "Battery_CurrentTemp"
                },
                attributeValue: "45"
            },
                attributeStatus: "Accepted",
                attributeType: "Actual",
                component: {
                    name: "OCPPCommCtrlr"
                },
                variable: {
                    name: "CCS1_W-Restart"
                },
                attributeValue: "false"

        ]
    }

}
```

```
msg.payload = {
    msgType: 3,
    "data": {
        "setVariableResult":
        [
            {
                "attributeStatus": "Accepted",
                "component": {
                    "name": "OCPPCommCtrlr"
                },
                "variable": {
                    "name": "Battery_W-Restart"
                }
            },
            {
                "attributeStatus": "Accepted",
                "component": {
                    "name": "AuthCtrlr"
                },
                "variable": {
                    "name": "CCS1_W-Voltage"
                }
            }
        ]
    }
}
```

**(a)** OCPP*getVariableResult* structure **(b)** OCPP*setVariableResult* structure

**Figure 3.10:** CS response message structures [33]

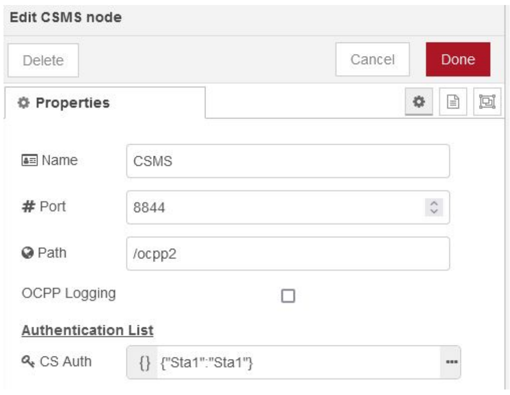

```
msg.payload = {
    "command": "GetVariables",
    "cbId": "Sta1",
    "data": {
        "getVariableData": [
            {
                "attributeType": "Actual",
                "component": {
                    "name": "OCPPCommCtrlr"
                },
                "variable": {
                    "name": "Battery_CurrentTemp"
                }
            },
            {
                "attributeType": "Actual",
                "component": {
                    "name": "AuthCtrlr"
                },
                "variable": {
                    "name": "CCS1_W-Restart"
                }
            }
        ]
    }
}
```

```
msg.payload = {
    "command": "SetVariables",
    "cbId": "Sta1",
    "data": {
        "setVariableData": [
            {
                "attributeValue": "false",
                "component": {
                    "name": "OCPPCommCtrlr"
                },
                "variable": {
                    "name": "Battery_W-Restart"
                }
            },
            {
                "attributeValue": "15",
                "component": {
                    "name": "AuthCtrlr"
                },
                "variable": {
                    "name": "CCS1_W-Voltage"
                }
            }
        ]
    }
}
```

**(a)** OCPP*getVariableData* structure **(b)** OCPP*setVariableData* structure

**Figure 3.11:** CSMS request message structures [33]

The CS node is configured by giving it a name and Chargebox Identity (CBID), choosing the target CSMS URL, and setting a password (figure 3.12b). Choosing the target CSMS URL requires a *target-csms* node to be added which requires the CSMS node to be configured first, this is done by giving it a name, assigning it a port (any port that isn't currently being used), creating a path, and setting up the CS authentication list (figure 3.12a). The target CSMS URL can then be added using the IP address of the CSMS (localhost as it's running on the same machine as the system), plus the port and path that was just created (figure 3.12c).


Edit CSMS node

Delete Cancel Done

Properties

Name: CSMS

Port: 8844

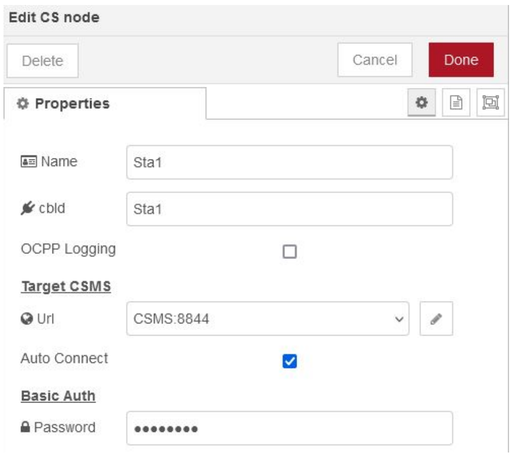
Path: /ocpp2

OCPP Logging: ☐

Authentication List

CS Auth: {"Sta1":"Sta1"}

**(a)** CSMS node properties


Edit CS node

Delete Cancel Done

Properties

Name: Sta1

cbId: Sta1

OCPP Logging: ☐

Target CSMS

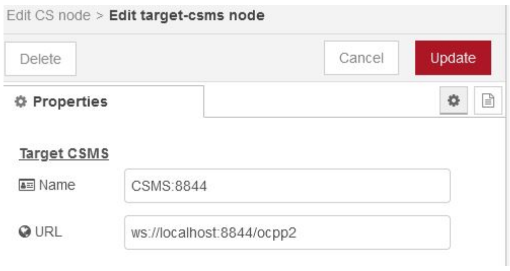
Url: CSMS:8844

Auto Connect: ✓

Basic Auth

Password: ••••••••

**(b)** CS node properties


Edit CS node > Edit target-csms node

Delete Cancel Update

Properties

Target CSMS

Name: CSMS:8844

URL: ws://localhost:8844/ocpp2

**(c)** *target-csms* node properties

**Figure 3.12:** Properties of the OCPP 2.0.1 nodes

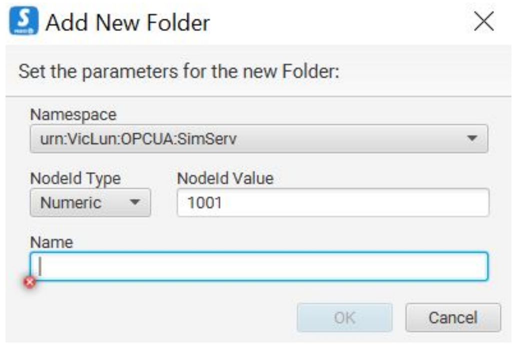
### 3.2.2 OPC UA Server - Prosys OPC

There are not a lot of free OPC UA simulation servers available, two of the choices were Integration Objects' OPC UA Server Simulator [46] and Prosys OPC's OPC UA Simulation Server [47]. In the end, Prosys OPC's OPC UA simulation server was chosen because it was simple to use, there were lots of guides online showing Node-RED interacting with Prosys OPC's OPC UA server, and Prosys OPC also offer an OPC UA Browser, to test that the server was set up correctly.
The Prosys OPC UA Simulation Server is a software tool designed for simulating OPC UA-enabled industrial devices and systems. It allows developers to test OPC UA client applications without access to physical equipment, providing a virtual environment for learning, prototyping and validation. The Simulation Server generates simulated data based on configurable models for instant use, or a developer can create their own data models for more accurate simulation [47].

#### Setting Up the Simulation Server

The OPC UA simulation server didn't need to be configured, it was ready to use as soon as it was installed. The browser was also ready to go after it was installed, it just needed to be connected to a server to be able to use it as intended, which just requires the connection address of the OPC UA server.
With the simulation server and browser up and running, the only thing left to do was create dummy data to help test the system. This was done by first adding a folder by assigning it a unique node ID, choosing the namespace it was to be attached to, and giving it a name (figure 3.13a). After a folder was created, it was then populated with variables representing

possible data from a mobile charging station. Variables were created by choosing the namespace, reference type ("read only" - *Organizes* or "read and write" - *DataSetToWriter*), and datatype and giving them unique node IDs and names (figure 3.13b). Figure 3.14 shows the simulation server populated with dummy data for testing and validating the system.


Add New Folder

Set the parameters for the new Folder:

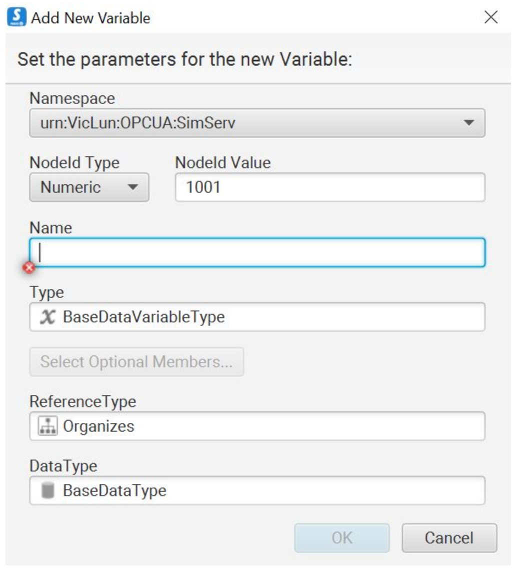
Namespace: urn:VicLun:OPCUA:SimServ

NodeId Type: Numeric

NodeId Value: 1001

Name:

**(a)** Adding a folder to the OPC UA simulation server


Add New Variable

Set the parameters for the new Variable:

Namespace: urn:VicLun:OPCUA:SimServ

NodeId Type: Numeric

NodeId Value: 1001

Name:

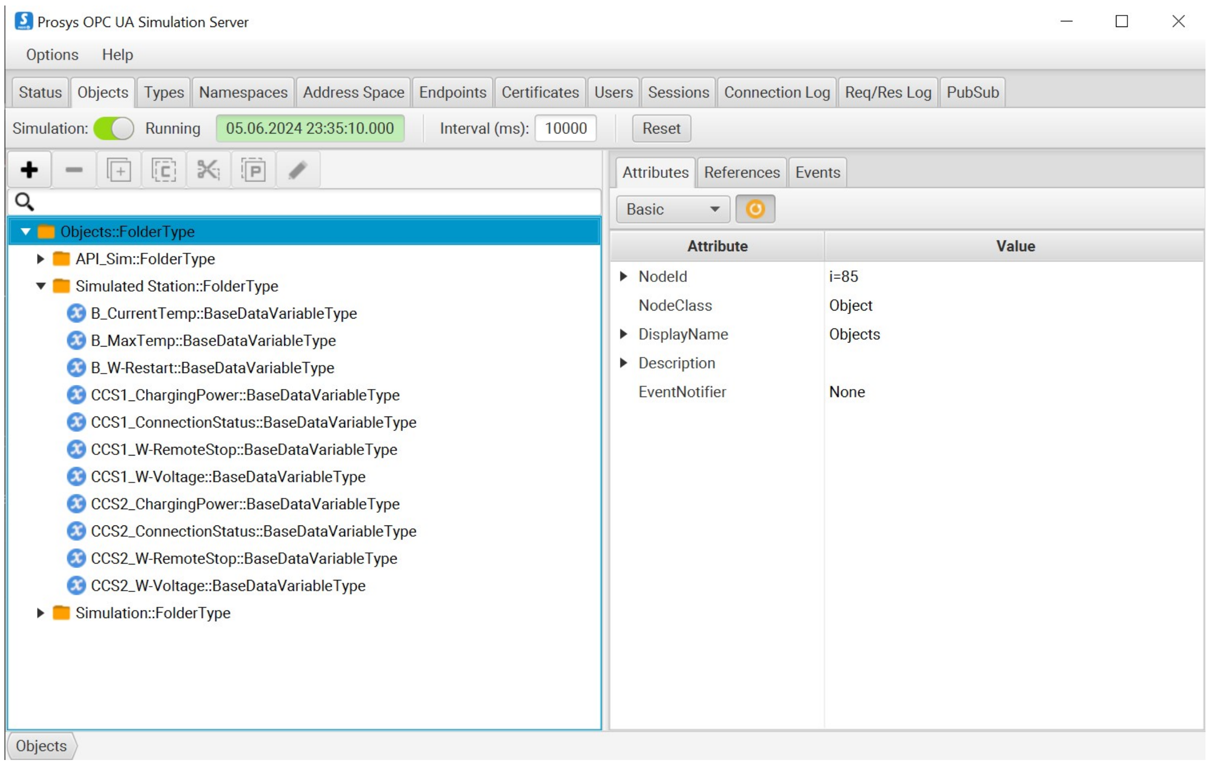
Type: BaseDataVariableType

ReferenceType: Organizes

DataType: BaseDataType

**(b)** Adding a variable to the OPC UA simulation server

**Figure 3.13:** Populating the Prosys OPC OPC UA Simulation Server


Prosys OPC UA Simulation Server

Simulation: Running 05.06.2024 23:35:10.000 Interval (ms): 10000

```
Objects::FolderType
  API_Sim::FolderType
  Simulated Station::FolderType
    B_CurrentTemp::BaseDataVariableType
    B_MaxTemp::BaseDataVariableType
    B_W-Restart::BaseDataVariableType
    CCS1_ChargingPower::BaseDataVariableType
    CCS1_ConnectionStatus::BaseDataVariableType
    CCS1_W-RemoteStop::BaseDataVariableType
    CCS1_W-Voltage::BaseDataVariableType
    CCS2_ChargingPower::BaseDataVariableType
    CCS2_ConnectionStatus::BaseDataVariableType
    CCS2_W-RemoteStop::BaseDataVariableType
    CCS2_W-Voltage::BaseDataVariableType
  Simulation::FolderType
```

| Attribute | Value |
|---|---|
| NodeId | i=85 |
| NodeClass | Object |
| DisplayName | Objects |
| Description | |
| EventNotifier | None |

**Figure 3.14:** Prosys OPC OPC UA Simulation Server populated with dummy data

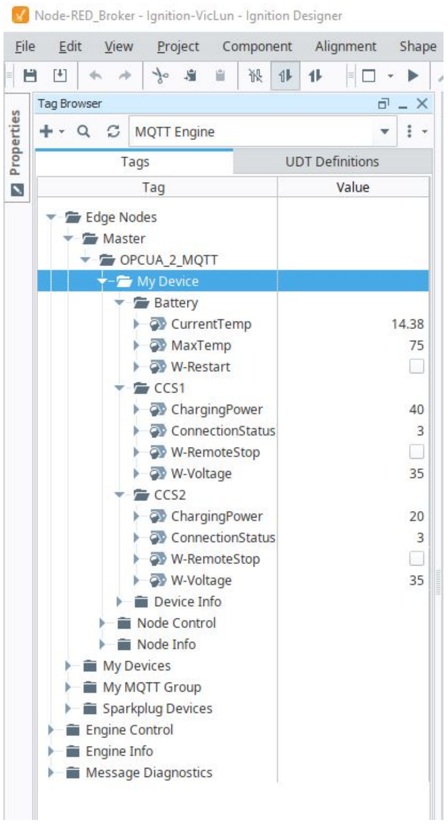
### 3.2.3 MQTT Broker - Ignition

Ignition is an industrial application platform that "acts as the hub for everything on your plant floor for total system integration" [8]. It supports various communication protocols,

database systems, and scripting languages, enabling seamless integration with existing infrastructure and easy customization to meet specific requirements.
One of the communication protocols that it supports is MQTT, through the use of Cirrus Link IIoT modules, and as an official Sparkplug Compatible software platform [48][49], Ignition makes an ideal MQTT Sparkplug broker.
There are a few different MQTT brokers available, such as EMQX [50], HiveMQ [51], and Ignition [8], however while all three are Sparkplug compatible [52][53][49] (though EMQX is not on the list of Sparkplug Compatible Software [48]), it wasn't easy to tell if the data was being received by the EMQX and HiveMQ brokers. Ignition much better for this with the Cirrus Link MQTT modules.

#### Setting Up the MQTT Broker

Installing Ignition was done by following the Ignition User Manual [54]. To make it MQTT Sparkplug B ready, two system modules were installed: MQTT Engine (acts like a Sparkplug EoN device [55]) and MQTT Distributor (an MQTT broker [56]). The installation and configuration of these modules was done by following the videos published by Inductive Automation about "Implementing MQTT in Ignition" [57][58]. The last step before the Ignition MQTT broker was ready to use for testing and validating the system was to install Ignition Designer [59]. The designer automatically organizes the data, by the tags (topics), published by the Node-RED Sparkplug device using the MQTT Engine (figure 3.15).



```
Tag                                   Value
Edge Nodes
  Master
    OPCUA_2_MQTT
      My Device
        Battery
          CurrentTemp                 14.38
          MaxTemp                     75
          W-Restart                   ☐
        CCS1
          ChargingPower               40
          ConnectionStatus            3
          W-RemoteStop                ☐
          W-Voltage                   35
        CCS2
          ChargingPower               20
          ConnectionStatus            3
          W-RemoteStop                ☐
          W-Voltage                   35
        Device Info
      Node Control
      Node Info
  My Devices
  My MQTT Group
  Sparkplug Devices
Engine Control
Engine Info
Message Diagnostics
```

**Figure 3.15:** The Ignition Designer showing the data from the OPC UA server, organized by tags

## 3.3 Testing and Validation Procedures

### 3.3.1 OPC UA to MQTT Sparkplug B and back

To test and validate that the system is able to convert OPC UA data to MQTT Sparkplug B messages and back again, a separate Node-RED flow, with a publisher node, subscriber node, and dashboard nodes, was created (figure 3.16 shows the flow and figure 3.18 shows the dashboard created in the flow). This flow starts with a subscriber that subscribes to all the data published to the topic `spBv1.0/Master/` on the MQTT broker. The data published to the subscriber is then split up based on the metric names and sent to the different dashboard nodes to visualize them. The widgets of the 5 writable variables are connected to function nodes that put the written value into a Sparkplug metric format, before sending it to the publisher node to be published to the MQTT broker, and subsequently to the Sparkplug EoN node, that converts the MQTT Sparkplug B message to OPC UA data before sending it to the OPC UA server (shown in figure 3.17).

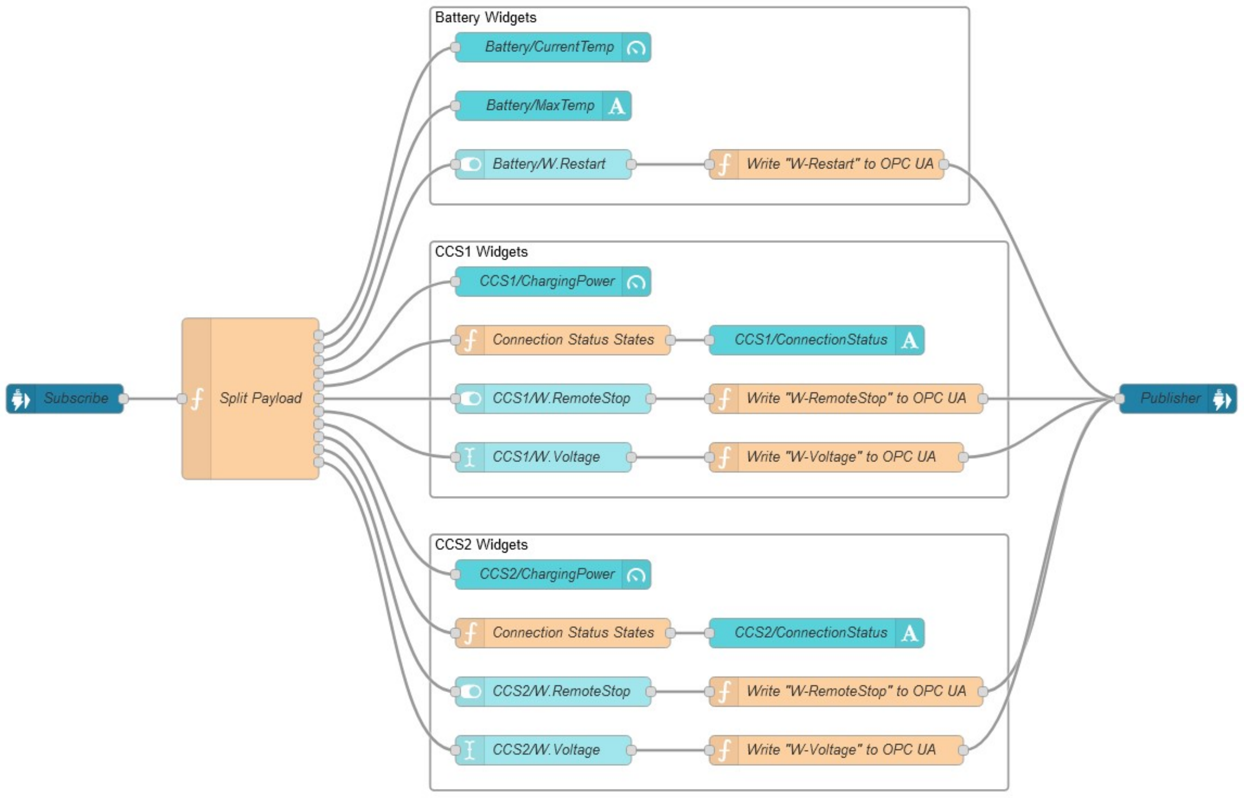


**Figure 3.16:** The flow for testing and validating the OPC UA to/from MQTT Sparkplug B part of the system

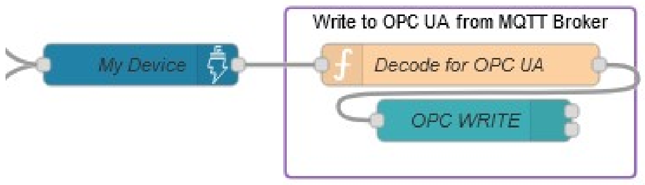


**Figure 3.17:** Converting MQTT Sparkplug B messages to OPC UA data before writing it to the OPC UA server

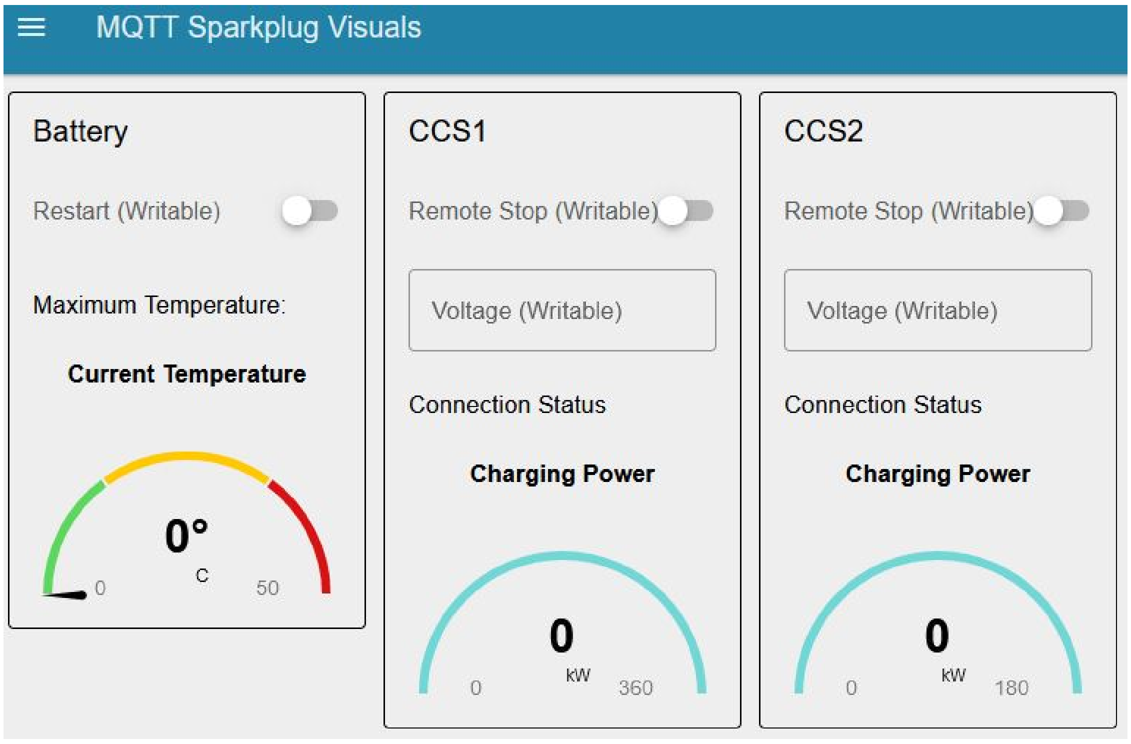


**Figure 3.18:** Dashboard of the MQTT Sparkplug B testing and validation flow

### 3.3.2 OPC UA to OCPP and back

To test and validate that the system is able to convert OPC UA data to OCPP messages and back again, a separate Node-RED flow, that acts as an OCPP CSMS, was created (shown in figure 3.19 and figure 3.20, appendix C shows the complete image). This flow has nodes for `getVariables`, `setVariables`, the CSMS, and visualizing the data (figure 3.21 shows the dashboard created in the flow). The test starts by either choosing which variables to get with `getVariables` or by choosing which variables to set and the value they're to be set to with `setVariables`. The request is sent into the CSMS node, which sends it to the connected CS node. When the CS node responds, the CSMS node sends the messages to be processed and transformed into data that can be visualized on the dashboard.

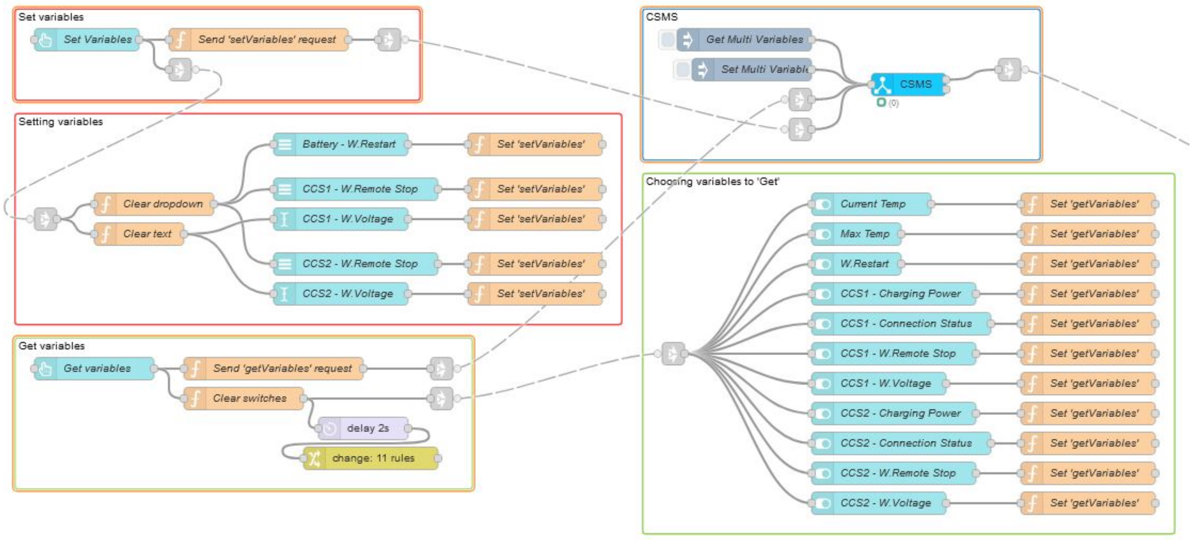


**Figure 3.19:** Part of the OCPP CSMS Node-RED flow showing *getVariables*, *setVariables* and the CSMS node

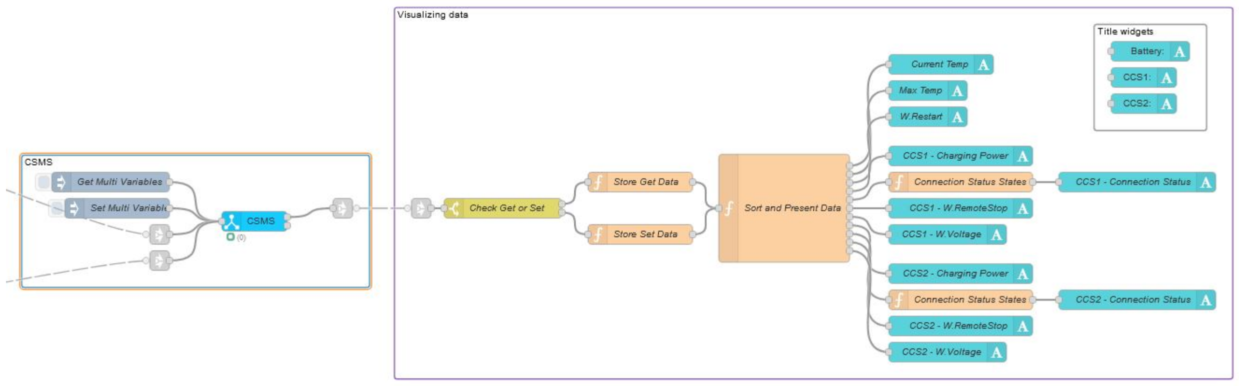
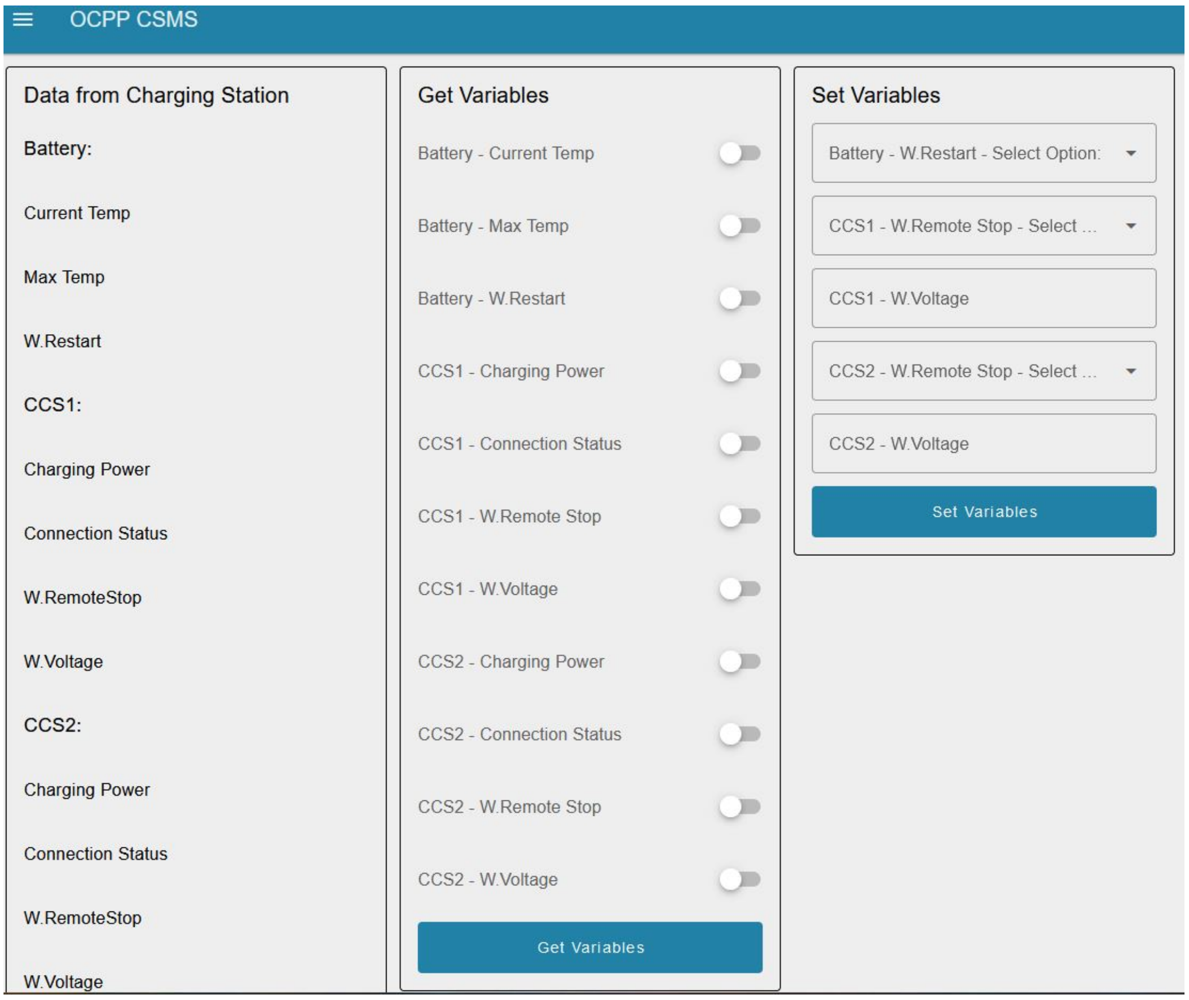


**Figure 3.20:** Part of the OCPP CSMS Node-RED flow showing the CSMS node and dashboard nodes to visualize the data


OCPP CSMS

Data from Charging Station

Battery:

Current Temp

Max Temp

W.Restart

CCS1:

Charging Power

Connection Status

W.RemoteStop

W.Voltage

CCS2:

Charging Power

Connection Status

W.RemoteStop

W.Voltage

Get Variables

Battery - Current Temp

Battery - Max Temp

Battery - W.Restart

CCS1 - Charging Power

CCS1 - Connection Status

CCS1 - W.Remote Stop

CCS1 - W.Voltage

CCS2 - Charging Power

CCS2 - Connection Status

CCS2 - W.Remote Stop

CCS2 - W.Voltage

Get Variables

Set Variables

Battery - W.Restart - Select Option:

CCS1 - W.Remote Stop - Select ...

CCS1 - W.Voltage

CCS2 - W.Remote Stop - Select ...

CCS2 - W.Voltage

Set Variables

**Figure 3.21:** Dashboard of the OCPP CSMS testing and validation flow

# Chapter 4

# System Design

In this chapter, the system design and architecture of the proposed solution is presented, detailing the specific components and their interactions. Before delving into the design specifics, it is important to note that certain design choices have been influenced by the fact that the system has only been connected to simulation programs and not actual outside systems. Also, the OPC UA to MQTT Sparkplug B flow was based on a proof of concept flow provided by Nordic Booster, that will not be made public.

## 4.1 System Architecture

The system architecture is designed to facilitate the seamless conversion of OPC UA data to OCPP and MQTT Sparkplug B messages. These messages and protocols are utilized by mobile charging stations, and communication between the mobile charging stations and monitoring platforms. The architecture consists of an OPC UA browser and client that subscribes to data from the OPC UA server (mobile charging station), function nodes that convert the OPC UA data to MQTT Sparkplug B and OCPP compliant messages, a Sparkplug EoN node that sends the converted OPC UA data, and an OCPP CS node that handles the converted OPC UA data based on the request from the CSMS node. The following diagram illustrates the architecture of the system (a larger version is shown in appendix B):

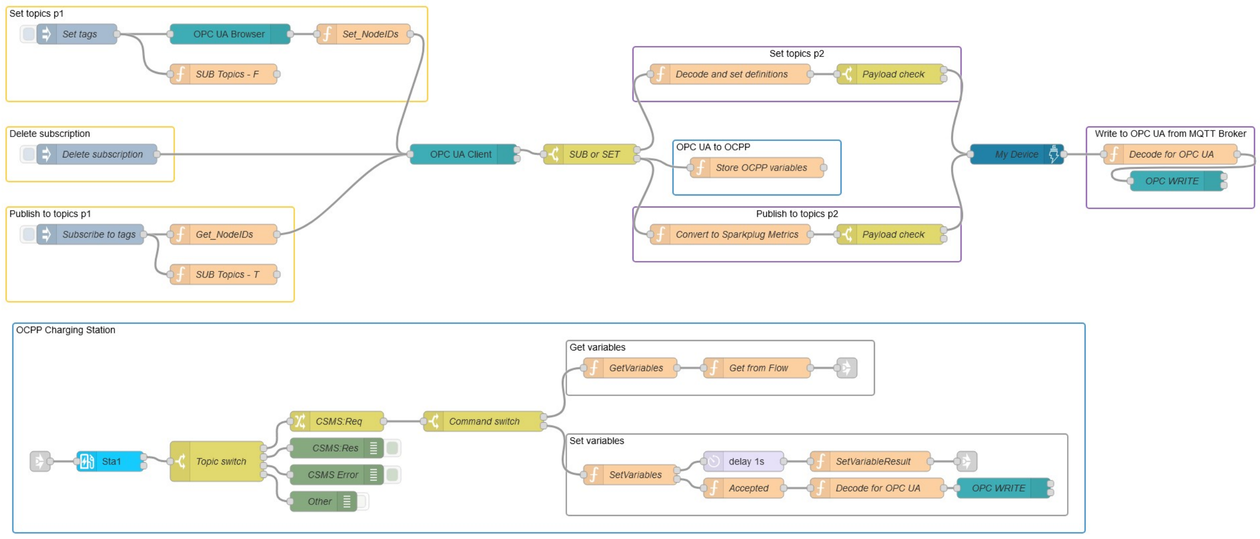


**Figure 4.1:** Architecture diagram of the proposed solution

In use, this system connects to the "outside world" in different ways (figure 4.2). The RTU running the Node-RED flows, interfaces with the MQTT Sparkplug B compliant broker, the OCPP CSMS, and the OPC UA server on the mobile charging station, which interfaces with the battery and charging equipment and collects real-time data. The Node-RED flows retrieve the OPC UA data, process it, and will send it to the OCPP CSMS, when requested and at the same time, publish the data to the MQTT broker whenever the data changes. The MQTT broker then distributes the messages to subscribers, such as monitoring and control systems at the construction sites.

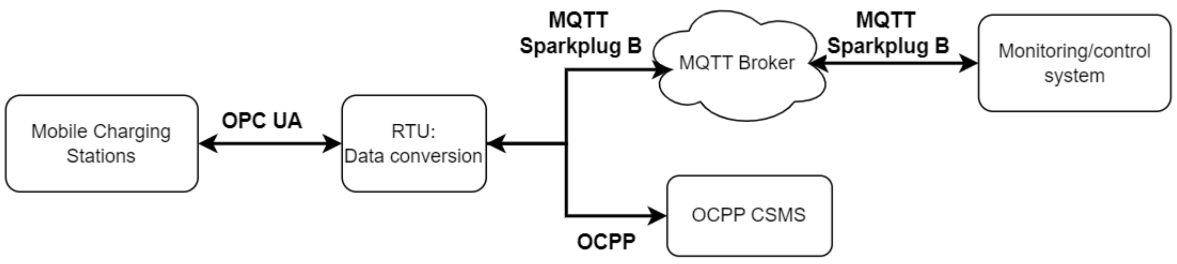


**Figure 4.2:** Diagram showing how the system (RTU) connects to the outside world

## 4.2 OPC UA to MQTT Sparkplug B flow[1]

This Node-RED flow, figure 4.3, is designed to achieve efficient and reliable conversion of OPC UA data into MQTT Sparkplug B messages and back again, in a way that doesn't require re-coding when changes to the server occur (i.e. variables are removed or added). It is composed of several nodes, each performing a specific function, some of which include data retrieval from the OPC UA server, data transformation, message publication to the MQTT broker, and writing data to the OPC UA server.

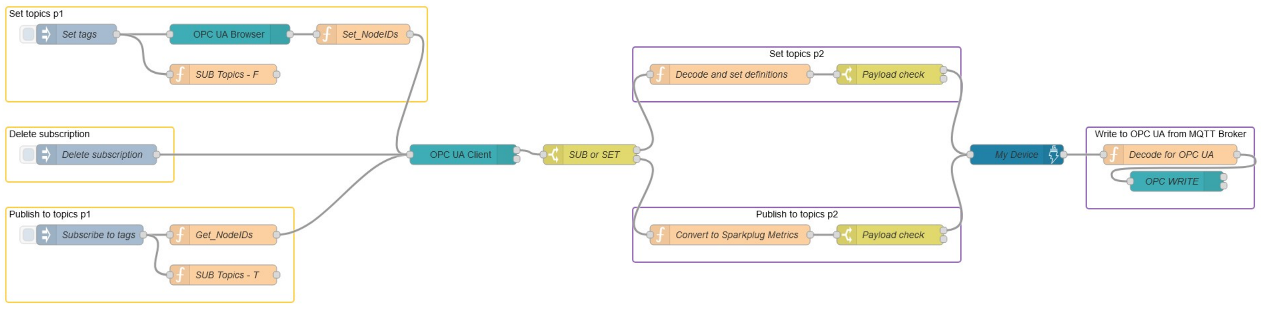


**Figure 4.3:** High-level architecture diagram of the OPC UA to MQTT Sparkplug B solution

### 4.2.1 Creating MQTT Sparkplug Definitions from OPC UA Data - Set Topics

The initial step in the Node-RED flow is to create MQTT Sparkplug definitions from the OPC UA variables so that the MQTT Sparkplug B broker knows the data type to expect for each topic (figure 4.4). This involves getting the node IDs of the relevant OPC UA variables, subscribing to them, mapping them to a Sparkplug B definition structure and sending the definitions to the MQTT Sparkplug B broker, without hard-coding the variables' node IDs. This process ensures that each piece of data from the OPC UA server is correctly identified and formatted for MQTT transmission.

[1]The code in this flow is based on Nordic Booster's proof of concept, and some code snippets have been modified from ChatGPT answers [60]

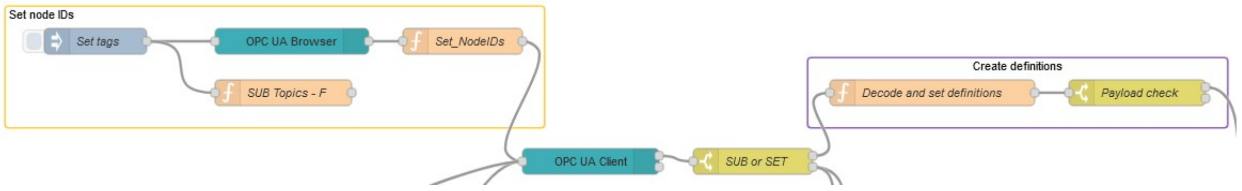


**Figure 4.4:** The part of the flow where MQTT Sparkplug definitions are created from OPC UA data

Getting the node IDs of the relevant OPC UA variables, without hard-coding them, starts with an inject node that initiates the nodes connected to it when pressed (a decision influenced by the system's current simulation programs), the *OPC UA Browser* and *SUB Topics - F* nodes. The OPC UA browser node connects to the OPC UA server and browses the OPC UA item address specified. At the same time, the function node, *SUB Topics - F*, sets the Flow context `SUB\_topics` to false and initializes the Flow context `definitions`, used later in the flow. The output of the browse node is sent as in input into a function node *Set_NodeIDs*, which takes the payload of the message, extracts only the node IDs from the variables into an array, saves the array in the Flow context `NODEIDS`, and sets the new payload of the message as the array of node IDs. Shown in figure 4.5.

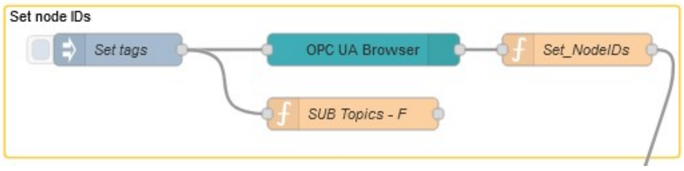


**Figure 4.5:** Setting the node IDs

The message with the array of node IDs is sent into the OPC UA Client node (shown on the left in figure 4.6) with the action *SUBSCRIBE*, to be subscribed to by the client, at an interval of one second. Each variable that has been subscribed to, is published as an individual message, each of which goes through a switch node (shown on the right in figure 4.6) that checks the value of the Flow context `SUB\_topics`, and is subsequently routed up to *Decode and set definitions* due to `SUB\_topics` being set to false earlier (figure 4.7).

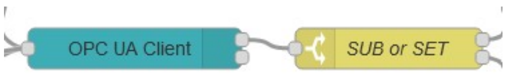


**Figure 4.6:** OPC UA client and switch node

For each message, the node ID is transformed into an MQTT topic by removing the first part of the OPC UA node ID ("ns=$x$;s=") and replacing the underscores with forward slashes. The node then checks if the `msg.payload` field exists and if it has a value property, if both conditions are true, the Flow context `set_payload_check` is set to true, otherwise it is set to false. This is done so a message containing the subscription ID isn't sent to the Sparkplug EoN, causing an error. The datatype is also remapped so that it's readable for MQTT. The remapped datatype, along with the topic, is used to create a new object with the correct MQTT message structure. This object is stored in the Flow context `definitions`, which is then set as the new payload of the message being sent out. The message is sent through *Payload check* where it checks what `set_payload_check` is set to, if it's true the message will be routed to the Sparkplug EoN and published to the MQTT Sparkplug B broker. If it's false, the message will not be routed any further.

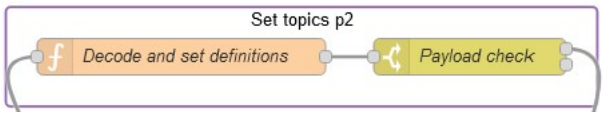


**Figure 4.7:** The part of the flow that creates Sparkplug definitions based on the data from the OPC UA server

### 4.2.2 Publishing the OPC UA data to the MQTT Sparkplug B Broker - Publish to Topics

After creating the MQTT Sparkplug B definitions, the next step in the Node-RED flow is to publish the OPC UA data to the MQTT broker (figure 4.8). This involves deleting the active subscription from earlier, subscribing to the stored node IDs, converting the variables to MQTT Sparkplug B metrics, and sending the metrics to the MQTT Sparkplug B broker. This step ensures that the data complies with the Sparkplug B specification, facilitating interoperability and efficient transmission.

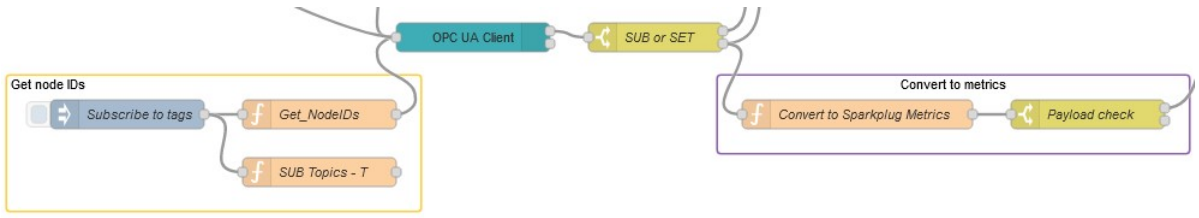


**Figure 4.8:** The part of the flow that subscribes to OPC UA items and converts the data to MQTT Sparkplug metrics

This starts with deleting the subscription created when setting the node IDs (shown in figure 4.9), so that there aren't multiple subscriptions to the same items. Then an inject node is pressed, triggering the *Get_NodeIDs* and *SUB Topics - T* nodes. The *SUB Topics - T* node sets the Flow context `SUB\_topics` to true, while the *Get_NodeIDs* node gets the Flow context `NODEIDs` and sets it as the payload of the message (figure 4.10).

Delete subscription
Delete subscription

**Figure 4.9:** Delete subscription

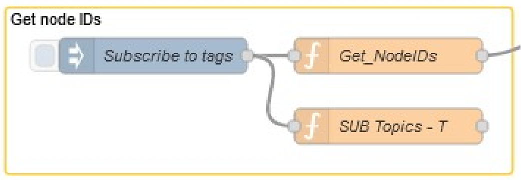


**Figure 4.10:** Get node IDs part of the flow

The message containing the node IDs of the OPC UA variables is sent to the OPC UA Client node to be subscribed to (shown on the left in figure 4.6). Each variable that has been subscribed to, is published as an individual message, each of which goes through a switch node (shown on the right in figure 4.6) that checks the value of the Flow context `SUB\_topics`, and is subsequently routed to *Convert to Sparkplug Metrics* due to `SUB\_topics` being set to true earlier.

As before in *Decode and set definitions*, for each message, the node ID is transformed into an MQTT topic by removing the first part of the OPC UA node ID ("ns=$x$;s=") and replacing the underscores with forward slashes. The node then checks if the `msg.payload` field exists and if it has a value property, if both conditions are true, the Flow context `set_payload_check` is set to true, otherwise it is set to false. This is done, so an empty message isn't sent to the Sparkplug EoN, causing an error. The newly created topic and variable value are used to create an MQTT Sparkplug B metric that is then sent to *Payload check* where it checks what `set_payload_check` is set to, if it's true the message will be routed to the Sparkplug EoN and the metrics are published to the MQTT Sparkplug B broker. If it's false, the message will not be routed any further. This is shown in figure 4.11.

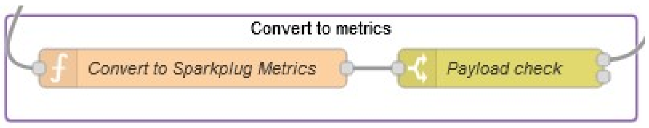


**Figure 4.11:** Convert OPC UA data to MQTT Sparkplug metrics

OPC UA data will now get converted into MQTT Sparkplug B messages and get published to the MQTT broker, whenever the OPC UA Client receives a response from the OPC UA server that variables have changed value. If changes occur to the server, i.e. variables are removed or added, the current subscription will have to be deleted, and the prior steps will have to be repeated.

### 4.2.3 Write to OPC UA from MQTT Sparkplug B

The last step in the Node-RED flow is to convert variables that have been changed via the monitoring platform from MQTT Sparkplug B message structure to OPC UA message structure (figure 4.12). When a writable variable is changed via the monitoring platform, the change is published to the broker and the Sparkplug EoN node. From the Sparkplug EoN node, the change is then sent as a message from the Sparkplug EoN node and into *Decode for OPC UA* where the datatype and value are extracted and the name (or topic) of the variable is transformed back into the OPC UA node ID by adding `"ns=7;s="` and replacing forward slashes with underscores. The node ID, value, and datatype are then used to create a new object with the desired structure (mentioned in section 3.2 under "OPC UA with Node-RED", and shown in figure 3.3b), which is then set as the new message. The new message is then sent into an OPC UA Client node set to the action *WRITE*, to then be written to the OPC UA server.

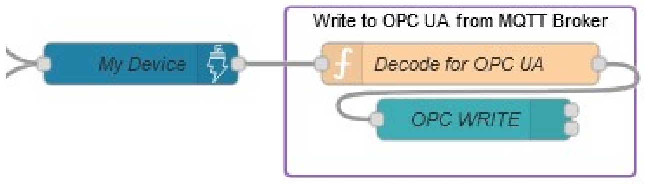


**Figure 4.12:** Write to OPC UA from MQTT Sparkplug B

## 4.3 OPC UA to OCPP Flow - Proof of Concept[2]

This Node-RED flow is designed to be show that it is possible to achieve conversion of OPC UA data into OCPP messages and back again, via Node-RED. This means that this flow

[2]Some of the code in this flow is based on Nordic Booster's proof of concept, some of the code is taken from node-red-contrib-ocpp2 example flows [61], and some code snippets have been modified from ChatGPT answers [60]

does not have all functionality for the OCPP components set up, mainly that the OCPP CS is only set up to handle `getVariables` and `setVariables` request messages from an OCPP CSMS.
The flow is composed of several nodes, each performing a specific function, some of which include handling connecting, sending, and receiving OCPP 2.0.1 messages to an OCPP 2.0.1 CSMS, data retrieval from the OPC UA server, data transformation, and writing data back to the OPC UA server.

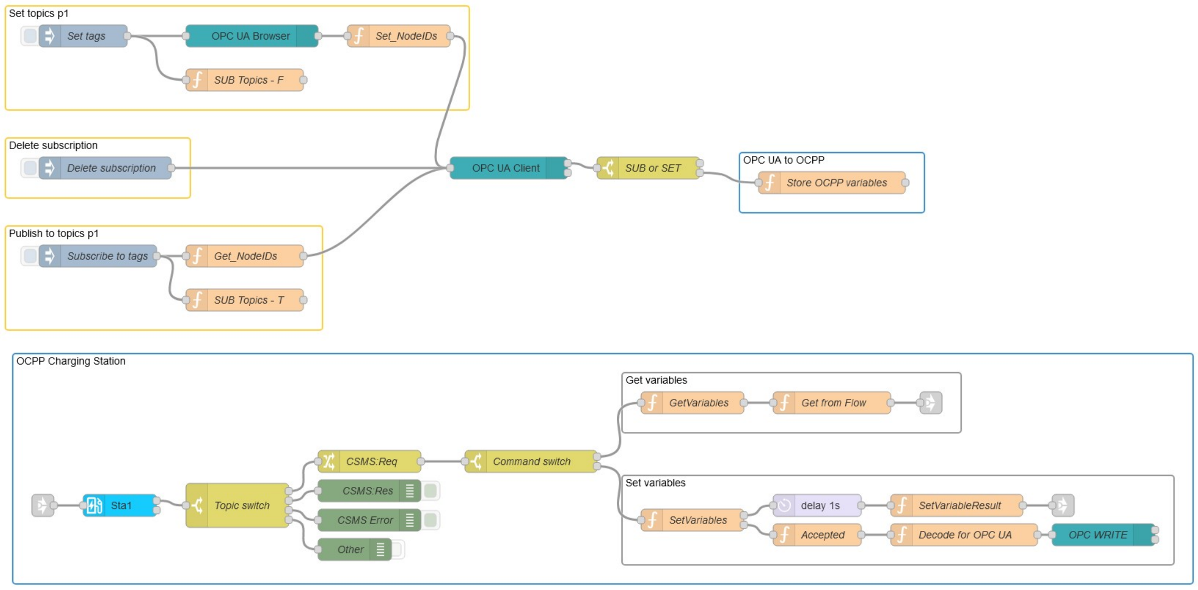


**Figure 4.13:** High-level architecture diagram of the OPC UA to OCPP solution

### 4.3.1 Subscribing to OPC UA server and Storing Data

The initial step in the Node-RED flow is subscribing to the OPC UA data and then storing it in a Flow context, so the OCPP CS node can access the data easily (figure 4.14). Subscribing to the OPC UA data is done the same way it was done for OPC UA to MQTT Sparkplug B: getting the node IDs from the OPC UA server using the OPC UA browser node, storing them, then using the stored node IDs to subscribe to the OPC UA Client (figure 4.15).

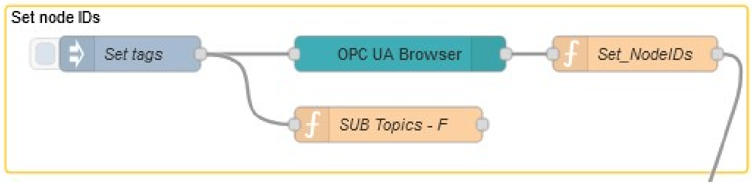


**Figure 4.14:** Setting the node IDs

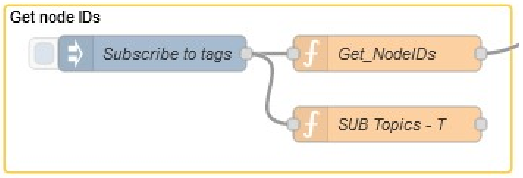


**Figure 4.15:** Get node IDs to subscribe to

Each variable that has been subscribed to, is published as an individual message that is routed to the function node *Store OCPP variables* (figure 4.16). This node extracts the variable name by removing `"ns=7;s="`, as well as the variable value and stores them in a Flow context with the variable name as the key and the variable value as the value.

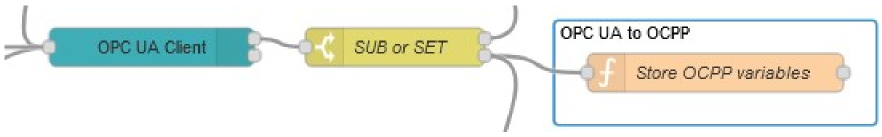


**Figure 4.16:** Store the data from the subscribed variables

### 4.3.2 OCPP Charging Station

The next step in the Node-RED flow is the OCPP CS handling the messages to and from the OCPP CSMS (figure 4.17). The standard output of the CS node is connected to the switch node, *Topic switch*, that routes the message to one of four outputs based on what the `msg.topic` is equal to. After the switch node, messages are routed to the change node *CSMS:Req* which sets the `msg.target` field to `msg.ocpp.cbid + :REQ: + msg.ocpp.command`, then returned to the flow, where the message is routed to the next node, the switch node *Command switch*. With the `msg.topic` field set to the OCPP command, the *Command switch* node can then route the message to the correct output based on what the OCPP command is, so that the correct response message is created to send back to the CSMS. Figure 4.18 shows a closer look at the nodes that route the messages after the CS receives them.

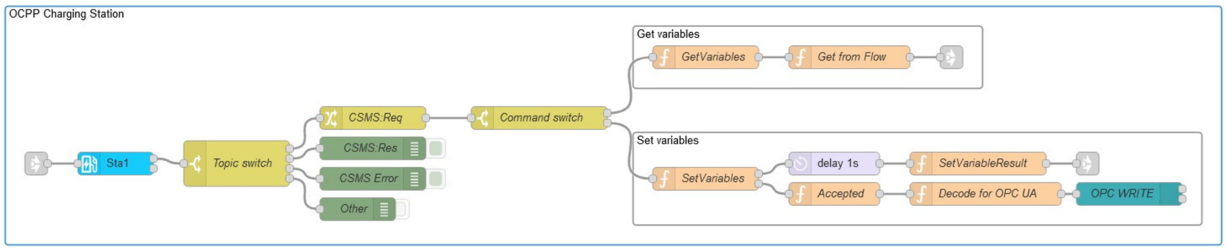


**Figure 4.17:** OCPP charging station flow

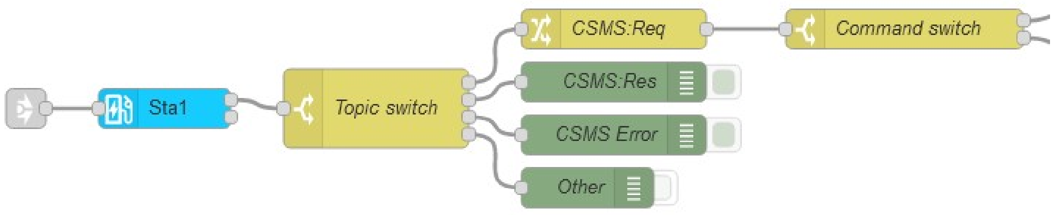


**Figure 4.18:** OCPP charging station message routing

If the OCPP request from the CSMS is `getVariables`, the response from the CS to the CSMS must include the attribute statue, component name as well as the type, name, and value of each variable requested, and if the OCPP request from the CSMS is `setVariables`, the response from the CS to the CSMS must include the attribute status, component name and variable name for each variable the CSMS requested to be set.

#### *Get Variables* from OPC UA to OCPP

If the OCPP command from the CSMS is `getVariables`, the request message is then routed to the *GetVariables* function node (the node on the far left in figure 4.19). This function node clones the `getVariableData` array, ensuring that the original message remains unaltered, this is important because the `ocpp` portion of the message is needed so that the node knows "to repackage the message with the same MessageId" [40]. The cloned array is then added into the `msg.payload` field and given `msg.Type` 3 signifying that this message is a response message.

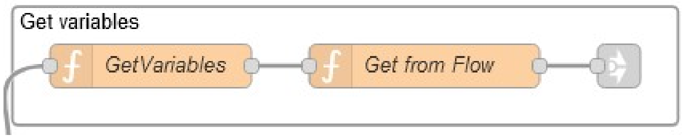


**Figure 4.19:** Function nodes that handle the *getVariables* request

The altered message is then routed to the function node *Get from Flow* (the node in the middle in figure 4.19) which loops through each item in the cloned array, and for each item it determines the attribute type (defaulting to "Actual" if it isn't specified), extracting the variable name, and getting the value of the variable from the Flow context, if there is one. If the Flow context for the variable has a value, an OCPP response is created for the variable and added to an array to store all the responses created. The array is then assigned to the `getVariableResult` field in the payload of the message, which is then returned to the flow to be routed into the input of the CS node, using *link in* and *link out* nodes (the node on the far right in figure 4.19) which sends the `getVariables` OCPP response back to the CSMS.

### *Set Variables* from OCPP to OPC UA

If the OCPP command from the CSMS is `setVariables`, the request message is routed to the *SetVariables* function node (the node to the far left in figure 4.20). This node starts in the same manner as the *GetVariables* node, by cloning the incoming message to keep the original unchanged. It then prepares the original message payload with the `msgType` 3 (response message), generates a unique set ID, and initializes an item ID. Then every variable in the `setVariableData` field of the cloned message is iterated through, where individual set commands are created, and the payload of each set command is simultaneously stored in the `itemArray` array and sent as the second output of the node, to be processed further. The `setId` and item array are added to the original message and returned to the flow as the first output to be processed further.

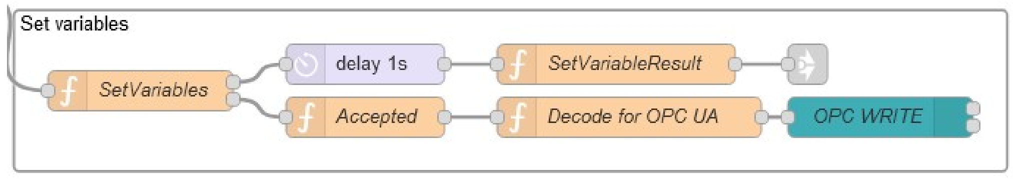


**Figure 4.20:** Function nodes that handle the *setVariables* request

The messages sent as the second output of the node are routed into the *Accepted* function node, which sets the variable status based on the datatype of the value and stores it in a unique Flow context based on the `msg.payload.setId`. If the value the variable is being requested to be set to is a string then the variable status is set to `Accepted` and the message is returned to the flow, otherwise it's set to `InvalidValue`, in which case `null` is returned. If the message is returned to the flow, it is routed to the *Decode for OPC UA* function node, which creates an OPC UA node ID from the variable name, converts the value from a string to a datatype the OPC UA client expects, and assigns the correct datatype identifier. The node ID, converted value, and datatype are then used to create a new object with the desired structure (mentioned in section 3.2 under "OCPP with Node-RED", and shown in figure 3.11b), which is then set as the new message. The new message is then sent to the

OPC UA Client node set to the action *WRITE*, which writes to the OPC UA server. This process is shown in figure 4.20, the bottom three nodes.
The messages sent as the first output of the node are routed to a delay node that waits one second before routing the message onto the *SetVariableResult* function node. This node starts by retrieving the array of variable statuses for the given `setId`, and determining the overall status of the variables; if all the status' are `Accepted` the overall status is set to `Accepted`, otherwise it's set to `Rejected`. Then each item in the `itemArray` is iterated over, creating a `setVariableResult` object, with the overall status, component name, and variable name, for each item. Each object is then added into an array, which is then set as the `setVariableResult` field in the payload of the message. The modified message is returned to the flow, being routed into the input of CS node, which sends the `setVariables` OCPP response back to the CSMS (the top two nodes in figure 4.20).

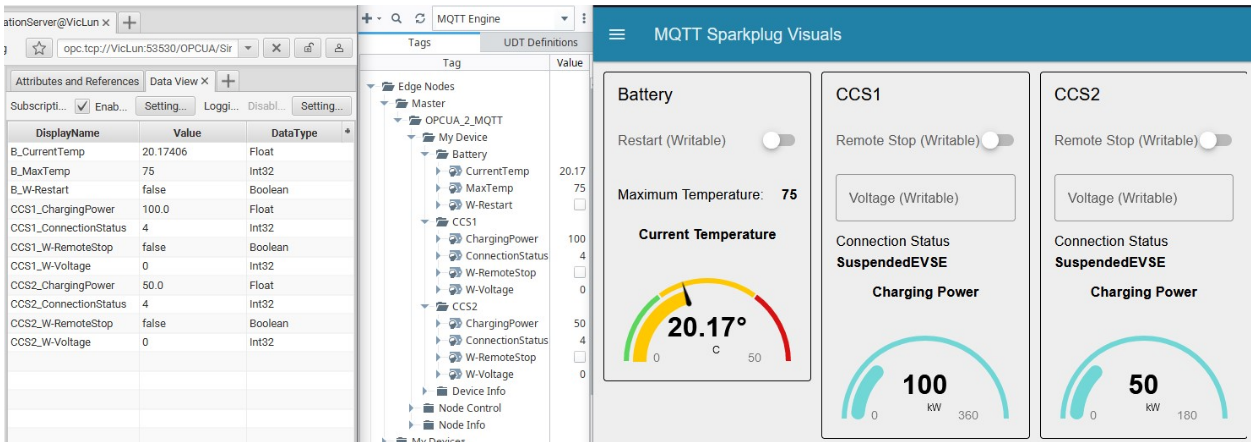
# Chapter 5

# Results

This section presents the results demonstrating the system's ability to convert data between OPC UA and MQTT Sparkplug B, as well as OPC UA and OCPP. The results are shown with a series of images that capture the reading/writing of data and validate the system's functionality. The system aims to facilitate data exchange between different industrial communication protocols: OPC UA, MQTT Sparkplug B, and OCPP. These protocols are widely used in industrial automation and electric vehicle charging infrastructure, respectively.

## OPC UA to MQTT Sparkplug B Conversion

Figure 5.1 shows data in OPC UA format being successfully converted to MQTT Sparkplug B format. The OPC UA data is displayed on the left, and the converted MQTT Sparkplug B data is displayed on the Ignition Designer in the middle, and on the Node-RED dashboard "MQTT Sparkplug Visuals" on the right.



| DisplayName | Value | DataType |
|---|---|---|
| B_CurrentTemp | 20.17406 | Float |
| B_MaxTemp | 75 | Int32 |
| B_W-Restart | false | Boolean |
| CCS1_ChargingPower | 100.0 | Float |
| CCS1_ConnectionStatus | 4 | Int32 |
| CCS1_W-RemoteStop | false | Boolean |
| CCS1_W-Voltage | 0 | Int32 |
| CCS2_ChargingPower | 50.0 | Float |
| CCS2_ConnectionStatus | 4 | Int32 |
| CCS2_W-RemoteStop | false | Boolean |
| CCS2_W-Voltage | 0 | Int32 |

```
Tag                              Value
Edge Nodes
  Master
    OPCUA_2_MQTT
      My Device
        Battery
          CurrentTemp            20.17
          MaxTemp                75
          W-Restart              ☐
        CCS1
          ChargingPower          100
          ConnectionStatus       4
          W-RemoteStop           ☐
          W-Voltage              0
        CCS2
          ChargingPower          50
          ConnectionStatus       4
          W-RemoteStop           ☐
          W-Voltage              0
        Device Info
      Node Control
      Node Info
    My Devices
```

Battery
Restart (Writable)
Maximum Temperature: 75
Current Temperature
20.17° C (0 – 50)

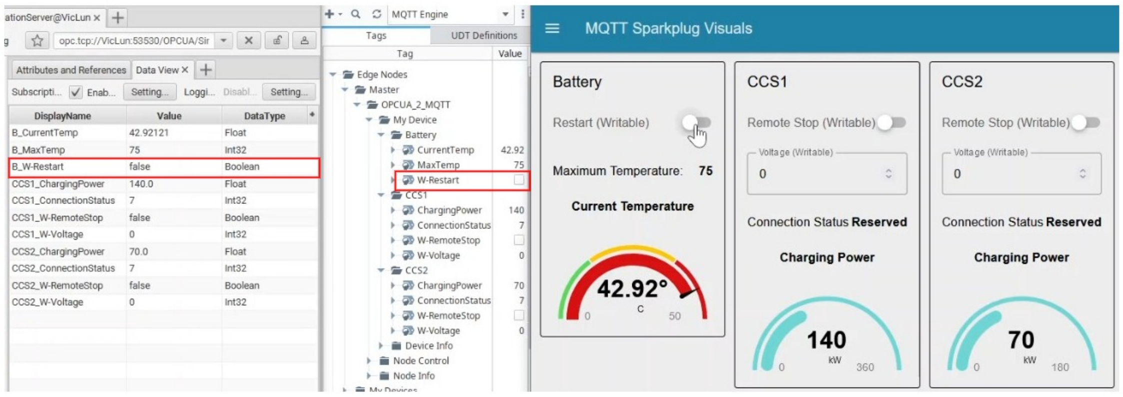
CCS1
Remote Stop (Writable)
Voltage (Writable)
Connection Status
SuspendedEVSE
Charging Power
100 kW (0 – 360)

CCS2
Remote Stop (Writable)
Voltage (Writable)
Connection Status
SuspendedEVSE
Charging Power
50 kW (0 – 180)

**Figure 5.1:** Conversion of data from OPC UA to MQTT Sparkplug B

## MQTT Sparkplug B to OPC UA Conversion

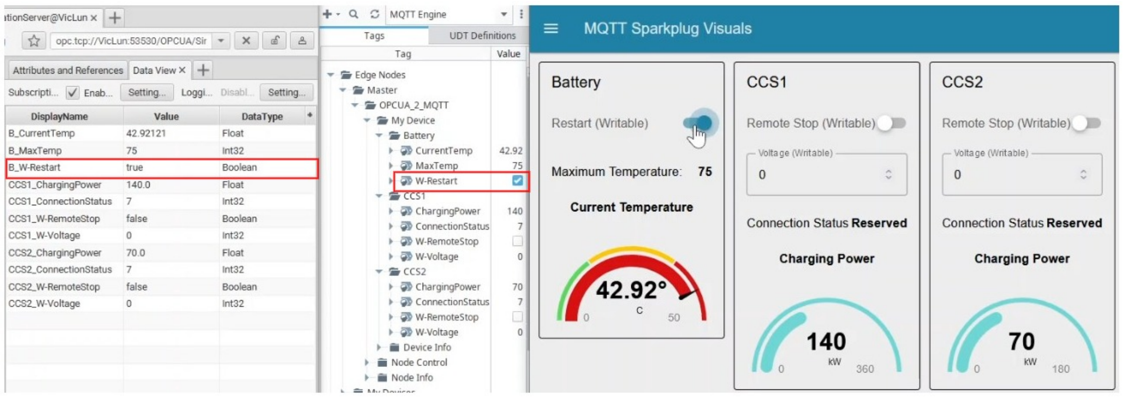
Figure 5.2 and figure 5.3 illustrate the conversion of data from MQTT Sparkplug B back to OPC UA format. Figure 5.2 shows the writing of a boolean, while figure 5.3 shows the writing of an integer.



| DisplayName | Value | DataType |
|---|---|---|
| B_CurrentTemp | 42.92121 | Float |
| B_MaxTemp | 75 | Int32 |
| B_W-Restart | false | Boolean |
| CCS1_ChargingPower | 140.0 | Float |
| CCS1_ConnectionStatus | 7 | Int32 |
| CCS1_W-RemoteStop | false | Boolean |
| CCS1_W-Voltage | 0 | Int32 |
| CCS2_ChargingPower | 70.0 | Float |
| CCS2_ConnectionStatus | 7 | Int32 |
| CCS2_W-RemoteStop | false | Boolean |
| CCS2_W-Voltage | 0 | Int32 |

| Tag | Value |
|---|---|
| Edge Nodes | |
| Master | |
| OPCUA_2_MQTT | |
| My Device | |
| Battery | |
| CurrentTemp | 42.92 |
| MaxTemp | 75 |
| W-Restart | ☐ |
| CCS1 | |
| ChargingPower | 140 |
| ConnectionStatus | 7 |
| W-RemoteStop | ☐ |
| W-Voltage | 0 |
| CCS2 | |
| ChargingPower | 70 |
| ConnectionStatus | 7 |
| W-RemoteStop | ☐ |
| W-Voltage | 0 |
| Device Info | |
| Node Control | |
| Node Info | |

**(a)** Start of writing "true" to the variable *Battery/W-Restart* OPC UA server from the MQTT Sparkplug B Node-RED dashboard

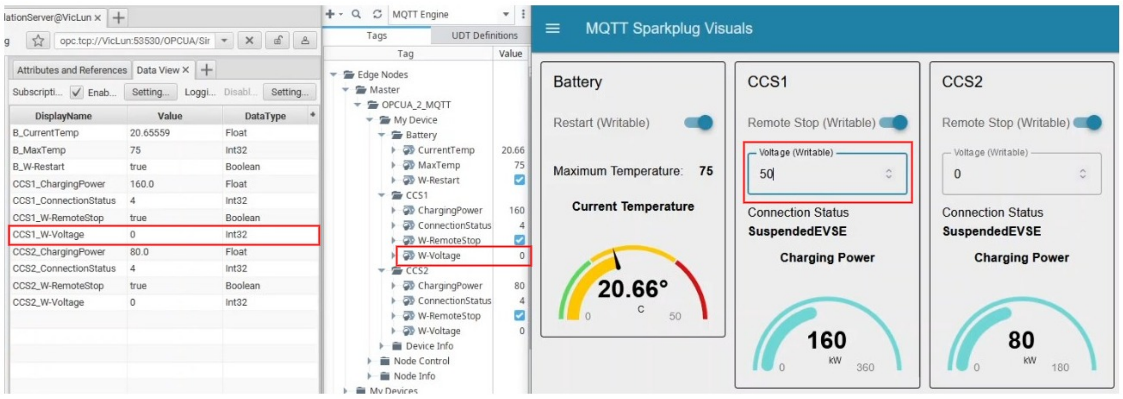


| DisplayName | Value | DataType |
|---|---|---|
| B_CurrentTemp | 42.92121 | Float |
| B_MaxTemp | 75 | Int32 |
| B_W-Restart | true | Boolean |
| CCS1_ChargingPower | 140.0 | Float |
| CCS1_ConnectionStatus | 7 | Int32 |
| CCS1_W-RemoteStop | false | Boolean |
| CCS1_W-Voltage | 0 | Int32 |
| CCS2_ChargingPower | 70.0 | Float |
| CCS2_ConnectionStatus | 7 | Int32 |
| CCS2_W-RemoteStop | false | Boolean |
| CCS2_W-Voltage | 0 | Int32 |

| Tag | Value |
|---|---|
| Edge Nodes | |
| Master | |
| OPCUA_2_MQTT | |
| My Device | |
| Battery | |
| CurrentTemp | 42.92 |
| MaxTemp | 75 |
| W-Restart | ✓ |
| CCS1 | |
| ChargingPower | 140 |
| ConnectionStatus | 7 |
| W-RemoteStop | ☐ |
| W-Voltage | 0 |
| CCS2 | |
| ChargingPower | 70 |
| ConnectionStatus | 7 |
| W-RemoteStop | ☐ |
| W-Voltage | 0 |
| Device Info | |
| Node Control | |
| Node Info | |

**(b)** After writing "true" to the variable *Battery/W-Restart* OPC UA server from the MQTT Sparkplug B Node-RED dashboard

**Figure 5.2:** Conversion of data from MQTT Sparkplug B to OPC UA

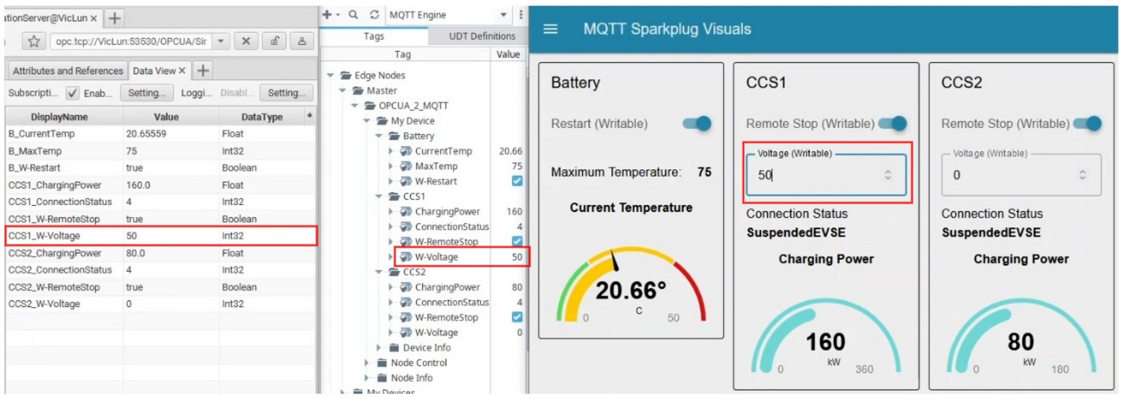


| DisplayName | Value | DataType |
|---|---|---|
| B_CurrentTemp | 20.65559 | Float |
| B_MaxTemp | 75 | Int32 |
| B_W-Restart | true | Boolean |
| CCS1_ChargingPower | 160.0 | Float |
| CCS1_ConnectionStatus | 4 | Int32 |
| CCS1_W-RemoteStop | true | Boolean |
| CCS1_W-Voltage | 0 | Int32 |
| CCS2_ChargingPower | 80.0 | Float |
| CCS2_ConnectionStatus | 4 | Int32 |
| CCS2_W-RemoteStop | true | Boolean |
| CCS2_W-Voltage | 0 | Int32 |

| Tag | Value |
|---|---|
| Edge Nodes | |
| Master | |
| OPCUA_2_MQTT | |
| My Device | |
| Battery | |
| CurrentTemp | 20.66 |
| MaxTemp | 75 |
| W-Restart | ✓ |
| CCS1 | |
| ChargingPower | 160 |
| ConnectionStatus | 4 |
| W-RemoteStop | ✓ |
| W-Voltage | 0 |
| CCS2 | |
| ChargingPower | 80 |
| ConnectionStatus | 4 |
| W-RemoteStop | ✓ |
| W-Voltage | 0 |
| Device Info | |
| Node Control | |
| Node Info | |

**(a)** Start of writing "50" to the variable *Battery/W-Voltage* OPC UA server from the MQTT Sparkplug B Node-RED dashboard

**Figure 5.3:** Conversion of data from MQTT Sparkplug B to OPC UA



| DisplayName | Value | DataType |
|---|---|---|
| B_CurrentTemp | 20.65559 | Float |
| B_MaxTemp | 75 | Int32 |
| B_W-Restart | true | Boolean |
| CCS1_ChargingPower | 160.0 | Float |
| CCS1_ConnectionStatus | 4 | Int32 |
| CCS1_W-RemoteStop | true | Boolean |
| CCS1_W-Voltage | 50 | Int32 |
| CCS2_ChargingPower | 80.0 | Float |
| CCS2_ConnectionStatus | 4 | Int32 |
| CCS2_W-RemoteStop | true | Boolean |
| CCS2_W-Voltage | 0 | Int32 |

| Tag | Value |
|---|---|
| Edge Nodes | |
| Master | |
| OPCUA_2_MQTT | |
| My Device | |
| Battery | |
| CurrentTemp | 20.66 |
| MaxTemp | 75 |
| W-Restart | ✓ |
| CCS1 | |
| ChargingPower | 160 |
| ConnectionStatus | 4 |
| W-RemoteStop | ✓ |
| W-Voltage | 50 |
| CCS2 | |
| ChargingPower | 80 |
| ConnectionStatus | 4 |
| W-RemoteStop | ✓ |
| W-Voltage | 0 |
| Device Info | |
| Node Control | |
| Node Info | |

MQTT Sparkplug Visuals

Battery
Restart (Writable)
Maximum Temperature: 75
Current Temperature
20.66° C

CCS1
Remote Stop (Writable)
Voltage (Writable) 50
Connection Status SuspendedEVSE
Charging Power
160 kW

CCS2
Remote Stop (Writable)
Voltage (Writable) 0
Connection Status SuspendedEVSE
Charging Power
80 kW

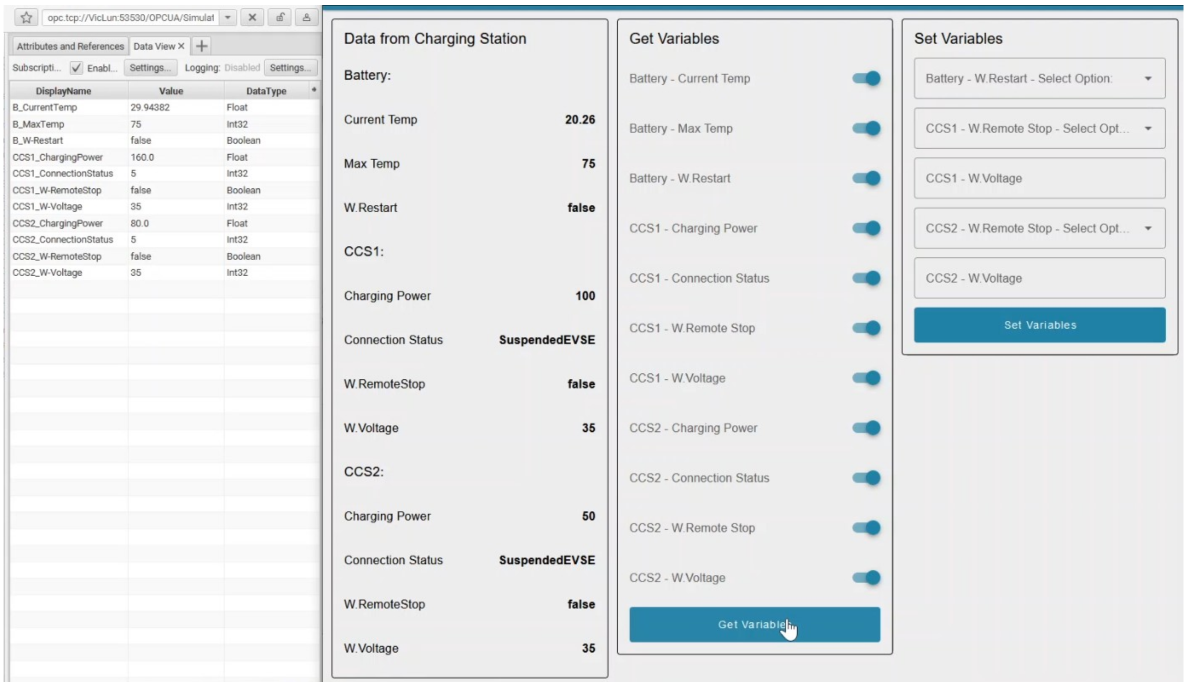
ationServer@VicLun
opc.tcp://VicLun:53530/OPCUA/Sir
Attributes and References
Data View
Subscripti... Enab... Setting... Loggi... Disabl... Setting...
MQTT Engine
Tags
UDT Definitions
0
50
360
180

**(b)** After writing "50" to the variable *Battery/W-Voltage* OPC UA server from the MQTT Sparkplug B Node-RED dashboard

**Figure 5.3:** Conversion of data from MQTT Sparkplug B to OPC UA

### OPC UA to OCPP Conversion

Figure 5.5 demonstrates the conversion process from OPC UA to OCPP format. The OPC UA data is shown on the right of both images, and the converted OCPP data is on the left, highlighting the successful `getVariables` request.



| DisplayName | Value | DataType |
|---|---|---|
| B_CurrentTemp | 29.94382 | Float |
| B_MaxTemp | 75 | Int32 |
| B_W-Restart | false | Boolean |
| CCS1_ChargingPower | 160.0 | Float |
| CCS1_ConnectionStatus | 5 | Int32 |
| CCS1_W-RemoteStop | false | Boolean |
| CCS1_W-Voltage | 35 | Int32 |
| CCS2_ChargingPower | 80.0 | Float |
| CCS2_ConnectionStatus | 5 | Int32 |
| CCS2_W-RemoteStop | false | Boolean |
| CCS2_W-Voltage | 35 | Int32 |

Data from Charging Station

| | |
|---|---|
| Battery: | |
| Current Temp | 20.26 |
| Max Temp | 75 |
| W.Restart | false |
| CCS1: | |
| Charging Power | 100 |
| Connection Status | SuspendedEVSE |
| W.RemoteStop | false |
| W.Voltage | 35 |
| CCS2: | |
| Charging Power | 50 |
| Connection Status | SuspendedEVSE |
| W.RemoteStop | false |
| W.Voltage | 35 |

Get Variables

Battery - Current Temp
Battery - Max Temp
Battery - W.Restart
CCS1 - Charging Power
CCS1 - Connection Status
CCS1 - W.Remote Stop
CCS1 - W.Voltage
CCS2 - Charging Power
CCS2 - Connection Status
CCS2 - W.Remote Stop
CCS2 - W.Voltage

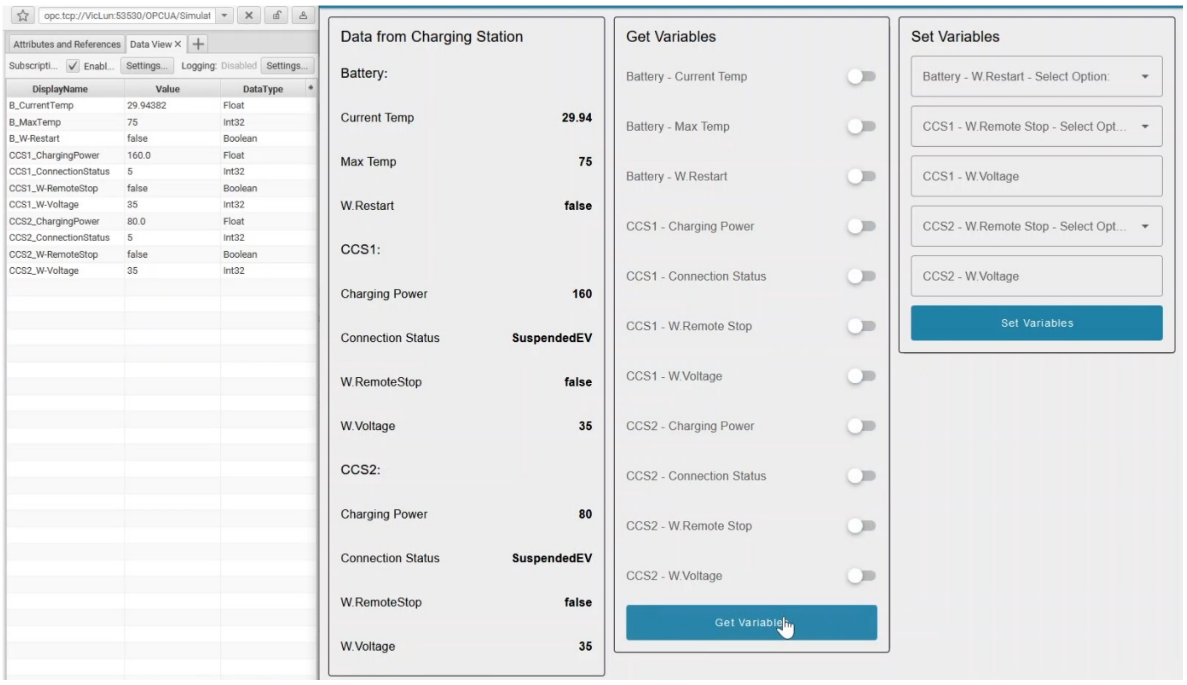
Set Variables

Battery - W.Restart - Select Option:
CCS1 - W.Remote Stop - Select Opt...
CCS1 - W.Voltage
CCS2 - W.Remote Stop - Select Opt...
CCS2 - W.Voltage

opc.tcp://VicLun:53530/OPCUA/Simulat
Attributes and References
Data View
Subscripti... Enabl... Settings... Logging: Disabled Settings...
Get Variables
Set Variables

**(a)** Triggering conversion of data from OPC UA to OCPP

**Figure 5.4:** Conversion of data from OPC UA to OCPP



opc.tcp://VicLun:53530/OPCUA/Simulat

Subscripti... ✓ Enabl... Logging: Disabled

| DisplayName | Value | DataType |
|---|---|---|
| B_CurrentTemp | 29.94382 | Float |
| B_MaxTemp | 75 | Int32 |
| B_W-Restart | false | Boolean |
| CCS1_ChargingPower | 160.0 | Float |
| CCS1_ConnectionStatus | 5 | Int32 |
| CCS1_W-RemoteStop | false | Boolean |
| CCS1_W-Voltage | 35 | Int32 |
| CCS2_ChargingPower | 80.0 | Float |
| CCS2_ConnectionStatus | 5 | Int32 |
| CCS2_W-RemoteStop | false | Boolean |
| CCS2_W-Voltage | 35 | Int32 |

Data from Charging Station

Battery:

| | |
|---|---|
| Current Temp | 29.94 |
| Max Temp | 75 |
| W.Restart | false |

CCS1:

| | |
|---|---|
| Charging Power | 160 |
| Connection Status | SuspendedEV |
| W.RemoteStop | false |
| W.Voltage | 35 |

CCS2:

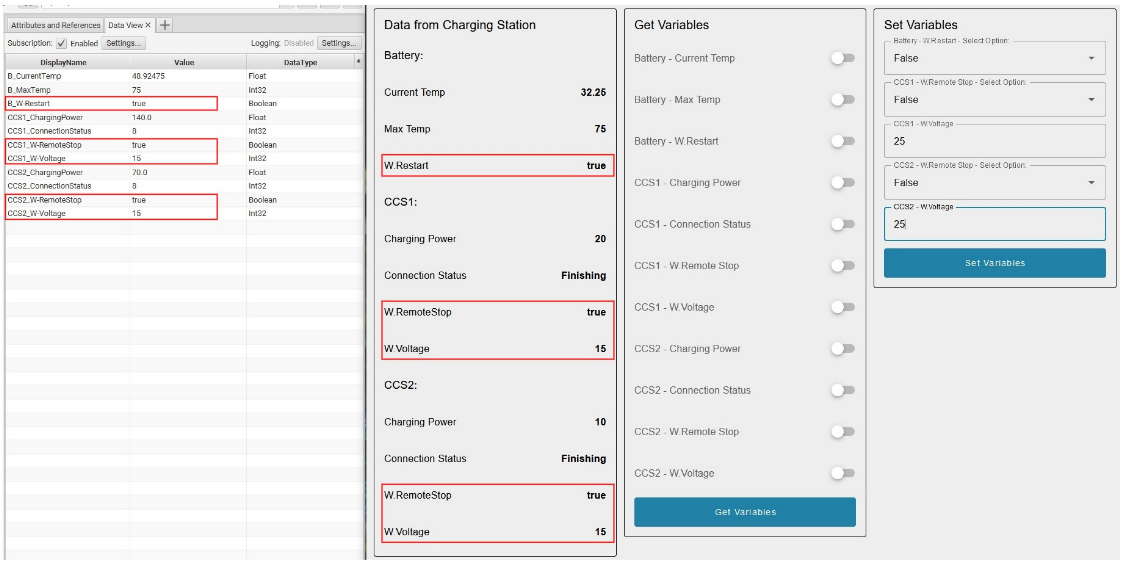
| | |
|---|---|
| Charging Power | 80 |
| Connection Status | SuspendedEV |
| W.RemoteStop | false |
| W.Voltage | 35 |

(b) Result of conversion of data from OPC UA to OCPP

**Figure 5.4:** Conversion of data from OPC UA to OCPP

### OCPP to OPC UA Conversion

Figure 5.5 demonstrates the conversion process from OCPP to OPC UA format. The OPC UA data is shown on the right of both images, and the converted OCPP data is on the left, highlighting the successful `setVariables` request.



Subscription: ✓ Enabled Logging: Disabled

| DisplayName | Value | DataType |
|---|---|---|
| B_CurrentTemp | 48.92475 | Float |
| B_MaxTemp | 75 | Int32 |
| B_W-Restart | true | Boolean |
| CCS1_ChargingPower | 140.0 | Float |
| CCS1_ConnectionStatus | 8 | Int32 |
| CCS1_W-RemoteStop | true | Boolean |
| CCS1_W-Voltage | 15 | Int32 |
| CCS2_ChargingPower | 70.0 | Float |
| CCS2_ConnectionStatus | 8 | Int32 |
| CCS2_W-RemoteStop | true | Boolean |
| CCS2_W-Voltage | 15 | Int32 |

Data from Charging Station

Battery:

| | |
|---|---|
| Current Temp | 32.25 |
| Max Temp | 75 |
| W.Restart | true |

CCS1:

| | |
|---|---|
| Charging Power | 20 |
| Connection Status | Finishing |
| W.RemoteStop | true |
| W.Voltage | 15 |

CCS2:

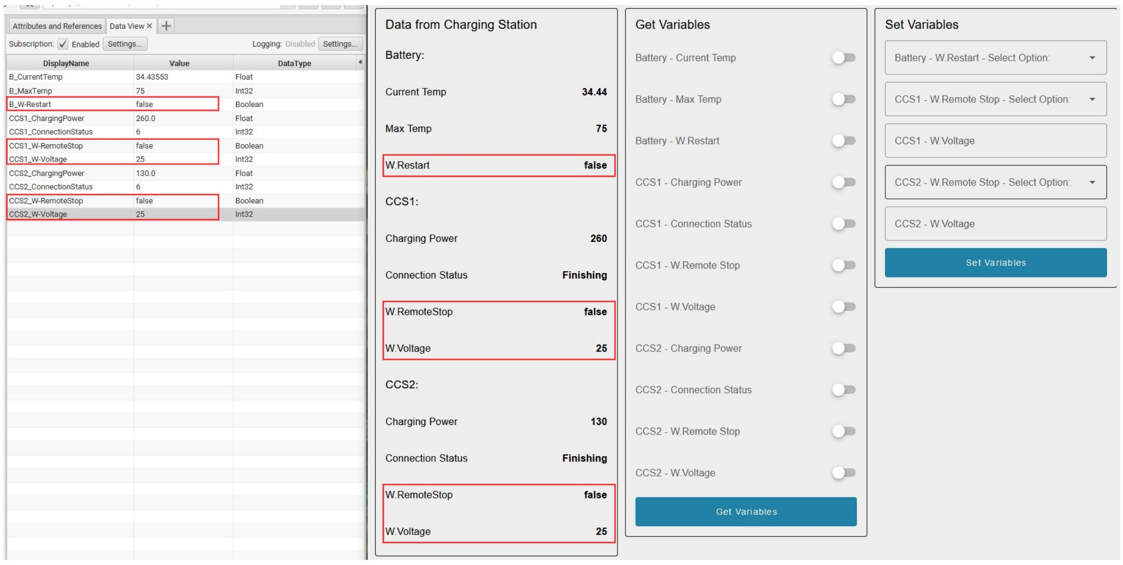
| | |
|---|---|
| Charging Power | 10 |
| Connection Status | Finishing |
| W.RemoteStop | true |
| W.Voltage | 15 |

Set Variables

| | |
|---|---|
| Battery - W.Restart - Select Option: | False |
| CCS1 - W.Remote Stop - Select Option: | False |
| CCS1 - W.Voltage | 25 |
| CCS2 - W.Remote Stop - Select Option: | False |
| CCS2 - W.Voltage | 25 |

(a) Start of writing to the OPC UA server from the OCPP CSMS Node-RED dashboard

**Figure 5.5:** Conversion of data from OPC UA to OCPP



| DisplayName | Value | DataType |
|---|---|---|
| B_CurrentTemp | 34.43553 | Float |
| B_MaxTemp | 75 | Int32 |
| B_W-Restart | false | Boolean |
| CCS1_ChargingPower | 260.0 | Float |
| CCS1_ConnectionStatus | 6 | Int32 |
| CCS1_W-RemoteStop | false | Boolean |
| CCS1_W-Voltage | 25 | Int32 |
| CCS2_ChargingPower | 130.0 | Float |
| CCS2_ConnectionStatus | 6 | Int32 |
| CCS2_W-RemoteStop | false | Boolean |
| CCS2_W-Voltage | 25 | Int32 |

Data from Charging Station

Battery:

| | |
|---|---|
| Current Temp | 34.44 |
| Max Temp | 75 |
| W.Restart | false |

CCS1:

| | |
|---|---|
| Charging Power | 260 |
| Connection Status | Finishing |
| W.RemoteStop | false |
| W.Voltage | 25 |

CCS2:

| | |
|---|---|
| Charging Power | 130 |
| Connection Status | Finishing |
| W.RemoteStop | false |
| W.Voltage | 25 |

Get Variables

Battery - Current Temp
Battery - Max Temp
Battery - W.Restart
CCS1 - Charging Power
CCS1 - Connection Status
CCS1 - W.Remote Stop
CCS1 - W.Voltage
CCS2 - Charging Power
CCS2 - Connection Status
CCS2 - W.Remote Stop
CCS2 - W.Voltage

Set Variables

Battery - W.Restart - Select Option:
CCS1 - W.Remote Stop - Select Option:
CCS1 - W.Voltage
CCS2 - W.Remote Stop - Select Option:
CCS2 - W.Voltage

**(b)** Result of writing to the OPC UA server from the OCPP CSMS Node-RED dashboard

**Figure 5.5:** Conversion of data from OCPP to OPC UA

The images above serve as a visual validation of the system's functionality. Initially, data in OPC UA format is ingested by the system, which then processes and converts it into MQTT Sparkplug B and OCPP formats. The reverse conversions are also illustrated, showcasing the system's bidirectional conversion capabilities. The results demonstrate the system's ability to convert data between OPC UA, MQTT Sparkplug B, and OCPP formats.

# Chapter 6

# Discussion

The results of the project demonstrate that the system functions as intended. It shows that Node-RED technology can be used to construct data flows capable of converting OPC UA data to MQTT Sparkplug B and OCPP messages, and vice versa. This integration enhances the interoperability of mobile charging stations, expanding their adaptability within construction site environments. However, it's important to note that the system was tested solely in a controlled environment, either locally on the same machine or across different machines within the same network.
The system's operability should not be affected by testing/running it on different machines on different networks, but the transmission times and reliability might change. A few tests were run, testing the difference in transmission times, in a purely OPC UA system run on different machines on different networks, with multiple OPC UA Client nodes subscribed to the OPC UA server, all with a large set of *MonitoredItems*. However, the results of these tests were not reliable, as the system time of the machines were not equal and the difference between the times varied constantly. Therefore, the results were not included in the result section.
The system also wasn't tested with actual monitoring platforms (mobile charging station monitoring systems or OCPP CSMSs) or a live OPC UA server, so the results from testing with an OPC UA simulation server and monitoring systems running locally on Node-RED might not be an accurate representation of the system's ability to connect and communicate with the three protocols nearly simultaneously.
Despite these limitations, the overall validity of the system was proven, because the outcome of the tests show that it is possible to convert the different communication protocol messages using Node-RED nodes. Which addresses and provides and solution to the challenge of converting OPC UA data to MQTT Sparkplug B and OCPP messages, for use in mobile charging stations at emission-free construction sites, using an open-source, low-cost (free) and lightweight program.

# Chapter 7

# Conclusion

In conclusion, this thesis presents a novel approach to enhancing the interoperability of mobile charging stations within emission-free construction sites. The results demonstrate the effectiveness of using Node-RED technology to construct data flows for converting OPC UA data to MQTT Sparkplug B and OCPP messages, thereby increasing the adaptability and integration capabilities of these stations. While the testing environment was limited to local setups, the outcomes of the tests underscore the feasibility of the proposed system in addressing the challenge of communication protocol conversion.
Moving forward, future work on this system will include extensive testing using live systems, integrating other communication protocols used at emission-free construction sites, like Modbus, and making the system more automatic (remove the inject nodes). By addressing these areas, the proposed solution aims to contribute to the broader goal of promoting sustainable and emission-free construction practices.
In summary, this thesis provides a valuable contribution to the field by offering a practical and cost-effective solution for facilitating data exchange using multiple communication protocols, in mobile charging stations. Through continued refinement and validation, the proposed system holds promise in supporting the transition towards more sustainable construction practices.

# Bibliography

[1] *Pilot testing of technology for emission-free construction sites is complete - and here are the results!* (in Norwegian), Aneo, Nov. 14, 2023. [Online]. Available: https://www.aneo.com/tjenester/build/nyheter/enova-pilot-ferdig/.

[2] *About Us - Nordic Booster*, Nordic Booster. [Online]. Available: https://www.nordicbooster.com/en/om-oss.

[3] *Hummingbird |*, Nordic Booster. [Online]. Available: https://www.nordicbooster.com/hummingbird-1.

[4] *Boost Point |*, Nordic Booster. [Online]. Available: https://www.nordicbooster.com/boostpoint-1.

[5] *Boost Charger |*, Nordic Booster. [Online]. Available: https://www.nordicbooster.com/boostcharger-1.

[6] Hordnes, Eirik and Nguyen, Linna V., "Today's power requirements at construction sites," (in Norwegian), Klimaetaten ved Oslo kommune and Sweco, Tech. Rep., 2023. [Online]. Available: https://www.klimaoslo.no/wp-content/uploads/sites/2/2024/01/Sweco_Kartlegging-av-effekt-og-energibehov-pa-utslippsfri-bygge-og-anleggsplass.pdf.

[7] *KEPServerEX: One Data Source for Your Industrial Automations*, PTC. [Online]. Available: https://www.ptc.com/en/products/kepware/kepserverex.

[8] *Ignition - The Platform for Unlimited Digital Transformation*, Inductive Automation. [Online]. Available: https://inductiveautomation.com/ignition/.

[9] *OPC 10000-14: UA Part 14: PubSub*, version 1.05.03, OPC Foundation, Dec. 13, 2023.

[10] *Mobil lynlading*, Kverneland Energi. [Online]. Available: https://kvernelandenergi.no/losninger/mobil-lynlading.

[11] *Våre ladeløsninger*, Aneo. [Online]. Available: https://www.aneo.com/tjenester/build/produkter.

[12] *Lading til byggeplass - Hafslund Mobil Energi*, Hafslund Boost. [Online]. Available: https://www.hafslundboost.no/produkter.

[13] *Nordic Booster*, Nordic Booster. [Online]. Available: https://www.nordicbooster.com.

[14] *Unified Architecture*, OPC Foundation. [Online]. Available: https://opcfoundation.org/about/opc-technologies/opc-ua/.

[15] *OPC 10000-5: UA Part 5: Information Model*, version 1.05.03, OPC Foundation, Dec. 13, 2023.

[16] *OPC 10000-3: UA Part 3: Address Space Model*, version 1.05.03, OPC Foundation, Dec. 13, 2023.

[17] *OPC Foundation Announces OPC UA PubSub Release*, OPC Connect, 2018. [Online]. Available: https://opcconnect.opcfoundation.org/2018/04/opc-foundation-announces-opc-ua-pubsub-release.

[18] *OPC 10000-4: UA Part 4: Services*, version 1.05.03, OPC Foundation, Dec. 13, 2023.

[19] *OPC 10000-6: UA Part 6: Mappings*, version 1.05.03, OPC Foundation, Dec. 13, 2023.

[20] *OPC 10000-1: UA Part 1: Overview and Concepts*, version 1.05.02, OPC Foundation, Nov. 1, 2022.

[21] *OPC UA PubSub Explained - Prosys OPC*, Prosys OPC, Dec. 14, 2021. [Online]. Available: https://prosysopc.com/blog/opc-ua-pubsub-explained.

[22] "OASIS MQTT Technical Committee Minutes of for the meeting of Thursday, 25th April 2013 Teleconference," OASIS MQTT TC, Apr. 25, 2013. [Online]. Available: https://groups.oasis-open.org/higherlogic/ws/public/document?document_id=49028 (Accessed May 23, 2024).

[23] HiveMQ, *MQTT & MQTT 5 Essentials.* HiveMQ GmbH, 2020. [Online]. Available: https://www.hivemq.com/resources/download-mqtt-ebook/ (Accessed May 23, 2024).

[24] A. Banks and R. Gupta, Eds., *MQTT Version 3.1.1*, OASIS Standard, Oct. 29, 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html. (Accessed May 23, 2024).

[25] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, Eds., *MQTT Version 5.0*, OASIS Standard, Mar. 7, 2017. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (Accessed May 23, 2024).

[26] A. Stanford-Clark and T. H. Linh, Eds., *MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2*, IBM, Nov. 14, 2013. [Online]. Available: https://groups.oasis-open.org/higherlogic/ws/public/document?document_id=66091 (Accessed May 23, 2024).

[27] *Standards*, OASIS Open. [Online]. Available: https://www.oasis-open.org/standards (Accessed May 23, 2024).

[28] *Use Cases*, MQTT.org. [Online]. Available: https://mqtt.org/use-cases (Accessed May 23, 2024).

[29] HiveMQ, *MQTT Sparkplug Essentials - Getting Started with this Open IIoT Specification.* HiveMQ GmbH. [Online]. Available: https://www.hivemq.com/resources/download-sparkplug-ebook/ (Accessed May 23, 2024).

[30] *Sparkplug 3.0.0 Sparkplug Specification*, Eclipse Foundation, Nov. 16, 2022. [Online]. Available: https://www.eclipse.org/tahu/spec/sparkplug_spec.pdf (Accessed May 31, 2024).

[31] *MQTT and Sparkplug B Simplified*, Corso Systems. [Online]. Available: https://corsosystems.com/posts/mqtt-and-sparkplug-b-simplified.

[32] *OCPP 2.0.1 Part 0 - Introduction*, 3rd ed., Open Charge Alliance, May 6, 2024. [Online]. Available: https://openchargealliance.org/my-oca/ocpp/.

[33] *OCPP 2.0.1 Part 2 - Specification*, 3rd ed., Open Charge Alliance, May 6, 2024. [Online]. Available: https://openchargealliance.org/my-oca/ocpp/.

[34] *OCPP 2.0.1 Part 4 - JSON over WebSockets implementation guide*, 3rd ed., Open Charge Alliance, May 6, 2024. [Online]. Available: https://openchargealliance.org/my-oca/ocpp/.

[35] *Node-RED*, OpenJS Foundation & Contributors. [Online]. Available: https://nodered.org/.

[36] *Getting Started : Node-RED*, OpenJS Foundation & Contributors. [Online]. Available: https://nodered.org/docs/getting-started.

[37] FlowFuse and M. Karaila, *Getting started with opc-ua and node-red*, Aug. 31, 2023. [Online]. Available: https://www.youtube.com/watch?v=9Kfo79Rkk2w (Accessed May 20, 2024).

[38] M. Karaila, *Node-red-contrib-opcua/opcua.* [Online]. Available: https://github.com/mikakaraila/node-red-contrib-opcua/tree/master/opcua (Accessed May 20, 2024).

[39] T. Sorensen, *Node-red-contrib-mqtt-sparkplug-plus.* [Online]. Available: http://flows.nodered.org/node/node-red-contrib-mqtt-sparkplug-plus.

[40] Nystrom, Bryan and Argonne National Library, *@anl-ioc/node-red-contrib-ocpp2*. [Online]. Available: https://flows.nodered.org/node/@anl-ioc/node-red-contrib-ocpp2 (Accessed Jun. 5, 2024).

[41] *Running on Windows*, OpenJS Foundation & Contributors. [Online]. Available: https://nodered.org/docs/getting-started/windows.

[42] M. Karaila and K. Landsdorf, *Node-red-contrib-opcua*. [Online]. Available: http://flows.nodered.org/node/node-red-contrib-opcua.

[43] *@flowfuse/node-red-dashboard*, FlowFuse. [Online]. Available: https://flows.nodered.org/node/@flowfuse/node-red-dashboard (Accessed Jun. 6, 2024).

[44] *The Core Nodes : Node-RED*, OpenJS Foundation & Contributors. [Online]. Available: https://nodered.org/docs/user-guide/nodes (Accessed Jun. 6, 2024).

[45] *Widgets | Node-RED Dashboard 2.0*, FlowFuse. [Online]. Available: https://dashboard.flowfuse.com/nodes/widgets.html (Accessed Jun. 6, 2024).

[46] *OPC UA Server Simulator*, Integration Objects. [Online]. Available: https://integrationobjects.com/sioth-opc/sioth-opc-unified-architecture/opc-ua-server-simulator (Accessed May 7, 2024).

[47] *OPC UA Simulation Server*, Prosys OPC. [Online]. Available: https://prosysopc.com/products/opc-ua-simulation-server/.

[48] *Sparkplug Compatible Program*, Eclipse Foundation. [Online]. Available: https://sparkplug.eclipse.org/compatibility/compatible-software/.

[49] *Ignition IIoT Software*, Inductive Automation. [Online]. Available: https://inductiveautomation.com/solutions/iiot.

[50] *EMQX: The #1 MQTT Platform for IoT, IIoT and Connected Cars*, Emq Technologies Inc. [Online]. Available: https://www.emqx.com/en.

[51] *Get HiveMQ On-Premise or Cloud-Based MQTT Broker*, HiveMQ. [Online]. Available: https://www.hivemq.com/company/get-hivemq.

[52] Joey, *MQTT Sparkplug in Action: A Step-by-Step Tutorial*, Feb. 11, 2024. [Online]. Available: https://www.emqx.com/en/blog/mqtt-sparkplug-in-action-a-step-by-step-tutorial.

[53] *MQTT Sparkplug: Building Powerful Industrial IoT Systems*, HiveMQ. [Online]. Available: https://www.hivemq.com/solutions/technology/mqtt-sparkplug.

[54] *1. Download and Install Ignition | Ignition User Manual*, Inductive Automation. [Online]. Available: https://docs.inductiveautomation.com/docs/8.1/getting-started/quick-start-guide/download-and-install (Accessed Feb. 13, 2024).

[55] *MQTT Engine - MQTT Modules for Ignition*, Cirrus Link Solutions. [Online]. Available: https://docs.chariot.io/display/CLD80/MQTT+Engine (Accessed Feb. 13, 2024).

[56] *MQTT Distributor - MQTT Modules for Ignition*, Cirrus Link Solutions. [Online]. Available: https://docs.chariot.io/display/CLD80/MQTT+Distributor (Accessed Feb. 13, 2024).

[57] *MQTT Engine Module*, Inductive Automation, 2019-12-06. [Online]. Available: https://inductiveautomation.com/resources/video/mqtt-engine-module (Accessed Feb. 13, 2024).

[58] *MQTT Distributor Module*, Inductive Automation, Dec. 9, 2019. [Online]. Available: https://inductiveautomation.com/resources/video/mqtt-distributor-module (Accessed Feb. 13, 2024).

[59] *5. Open the Designer*, Inductive Automation. [Online]. Available: https://docs.inductiveautomation.com/docs/8.1/getting-started/quick-start-guide/open-the-designer (Accessed Feb. 13, 2024).

[60] OpenAI, *ChatGPT*, version GPT-4, 2024. [Online]. Available: %7Bhttps://www.openai.com%7D.

[61] Argonne National Laboratory, *node-red-contrib-ocpp2*, 2024. [Online]. Available: https://github.com/Argonne-National-Laboratory/node-red-contrib-ocpp2 (Accessed Jun. 5, 2024).

## Appendix A

# GitHub Repository Link

Master's Thesis 2024 - GitHub

# Appendix B

# System Architecture Diagram

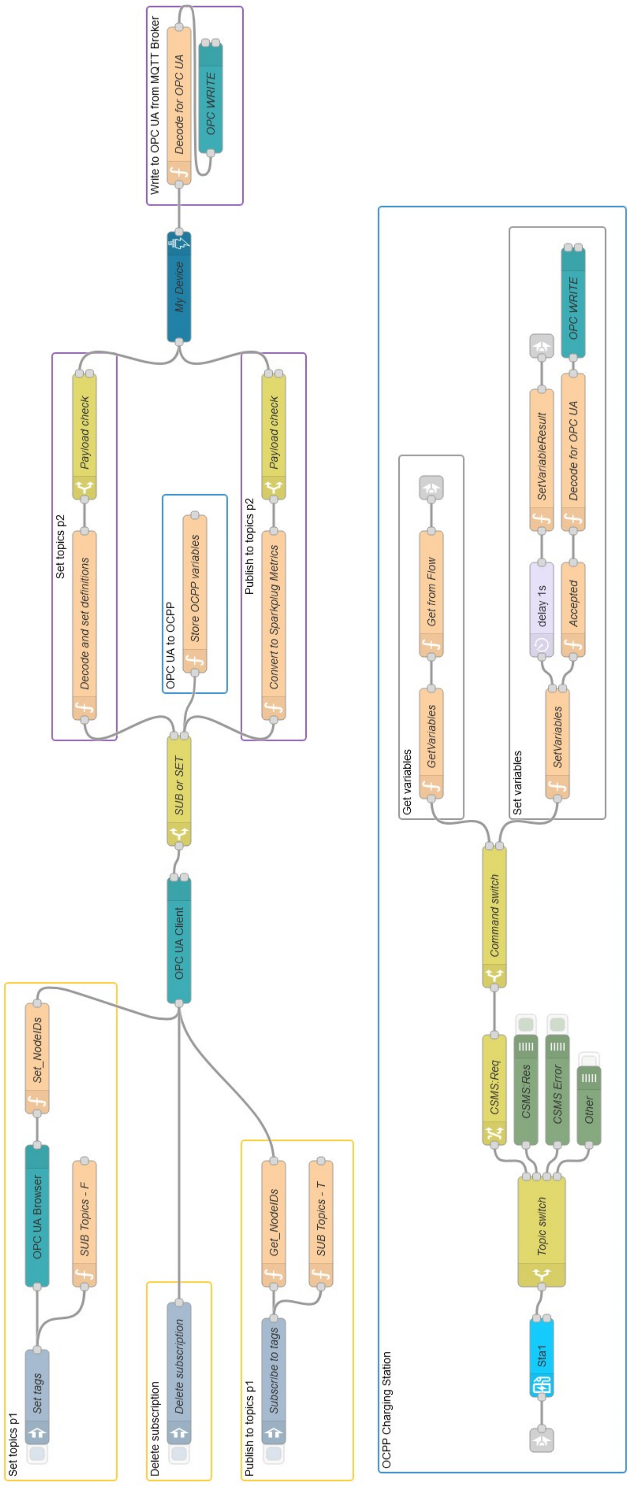


**Figure B.1:** System architecture diagram

# Appendix C

# Complete OCPP Test Visual

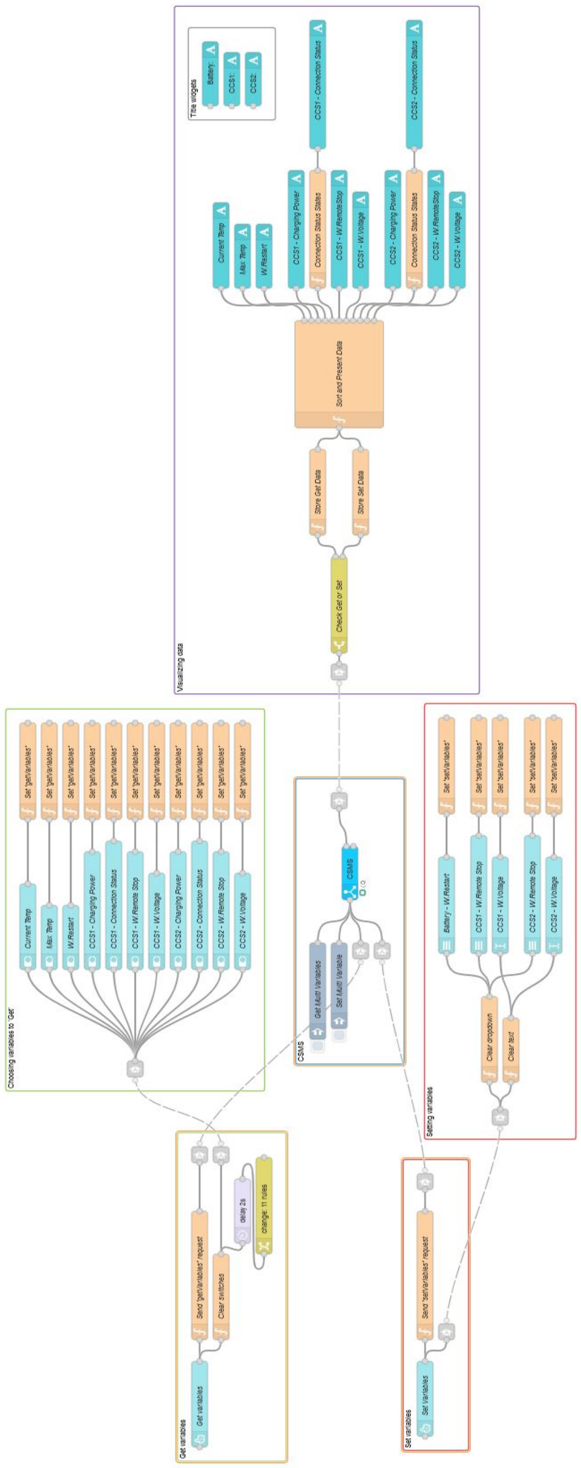


**Figure C.1:** The complete image of the OPC UA to/from OCPP test flow