University of Victoria
Engineering & Computer Science Co-op
Work Term Report
Spring 2020

Efficient Well-Scheduling & Automation

Ovintiv Inc.
Technical Services
Calgary, Alberta, Canada

Nolan Caissie
[redacted]
WT 1
Electrical Engineering
[redacted]
April 26, 2020

In partial fulfillment of the academic requirements of this co-op term

Table of Contents

List of Figures

Glossary

API – An application programming interface, which in this instance, enables a user's program to interface with the Moxa computer.

C – A high-level procedural programming language.

Coil – A single bit register that can hold a single binary number (0 or 1), used for discrete input and output signals in a controller.

COMM port – COMM, is short for communications. A COMM port uses serial communications as opposed to a BUS which does not.

Communication protocol – Can be thought of as a language of digital communication between devices.

Data acquisition – The process of acquiring and storing digital information taken from field devices.

DCS – A Distributed control system is an interconnected network of controllers and devices.

Edge computing – The process of performing computations on the "edge" of a computer network. This can be thought of as computing between an interface of two networks, such as between the internet and a LAN.

Field devices – For the purpose of this document, a field device is an instrument or controller used for automation in the oil and gas industry.

GUI – A graphical user interface enables user-friendly graphical interaction with computer hardware.

Headless computer – A computer that has no desktop environment, and thus, must be controlled through a command line.

HMI – Similar to a GUI, it is a user interface. It enables a human to directly interact with a machine.

IIoT – "Industrial internet of things", is a loose term meant to describe a system connecting "things" in an industrial environment to the internet so that they can be

controlled and monitored remotely. It is often used in conjunction with the MQTT communication protocol due to its low bandwidth requirements.

LAN – A local area network, or LAN, is a network that computers or controllers use to communicate with each other but it does not reside on the internet.

Modbus – Modbus is a serial communication protocol; it can be thought of as a "language" or set of rules for serial communications.

Modbus TCP – Similar to the simple Modbus definition, Modbus TCP is a serial communication protocol that communicates over a computer network such as a LAN or internet connection, as opposed to a COMM port. It may also be called Modbus TCP/IP as it works in conjunction with the IP (internet protocol), and thus, devices are addressed by an IP.

Moxa Computer – A headless computer that runs a Linux operating system. It can act as an interface or "edge computer" to perform computations between networks, as well as, be used as a means of distributing and acquiring data between network interfaces.

MQTT – A lightweight communication protocol often used in IoT applications. Similarly to Modbus TCP, it communicates through a LAN or internet connection.

MQTT broker – A server that creates a "cloud-like" architecture where clients can publish and subscribe to topics within the server. As an example, if a client somewhere in the world published a message to a topic in a specific broker, a client on the other side of the world could subscribe to that topic to receive the message. Neither client is actually connected or networked together, and thus, the MQTT broker provides a safe and private way to share data over the internet.

Node-RED – An open-source software often used in IoT applications for visual, flow-based programming.

Peripherals – A device that allows a user to input or output data from a computer. A keyboard is an example of a peripheral device.

Petroleum – For the purposes of this document, petroleum is a broad term used to describe oil and gas products.

PLC – A Programmable logic controller is a computer used in industrial applications for control and automation. It relies on "ladder logic", which is essentially a programming language that replaced the physical relays historically used in very large numbers for automation.

Python – A high-level object-oriented computer programming language with a wide range of uses.

Register – A register is similar to a coil but can hold larger binary numbers, as opposed to just a single digit. Registers are used to hold values or instructions for quick processing.

Router – A router essentially "routes" data between computer networks that would otherwise be incapable of communicating with each other. An example of this use is a router used in a home network that routes internet traffic to and from a home's LAN.

rPAC – A remote programmable automation controller, is a controller used in industrial settings for automation and control. It is similar in some aspects to a PLC, but in this instance has less available input and output. It is, however, a newer technology and can typically run C code. As it pertains to the workplace in which this project is being conducted, the word rPAC is sometimes used interchangeably with RTU.

SCADA – A supervisory control and data acquisition system, is a network of computers and controllers similar to a DCS, but it provides a "supervisory" aspect where human operators can view operations remotely and make adjustments in real time.

Serial communications – A method of electronic (digital) communication that sends discrete (Boolean) bits to be interpreted by a processor.

Service – A program or software that runs in the background; a service will start running when a computer boots up, and it will run continuously unless it is told to stop running.

Topics (MQTT) – A topic operates similarly to how an IP address operates. The difference is that an IP address is different for every device in a network; whereas, a topic resides in the cloud in an MQTT broker and any client (computer, or similar device) can subscribe or publish data to that topic. This is beneficial when it is preferred to have data sent to many different clients at one time.

VPN – A virtual private network operates similarly to a private network, such as a LAN, but is called a "virtual" network because it extends privacy across networks such as the internet.

Web-based GUI – A GUI that operates in a web-browser such as Google Chrome. To access a GUI that resides on another device within a network, a user can simply type the IP address and port number of the other device (likely a headless computer) into the browser search bar and then interact with it on the local non-headless machine.

Well – A well, for the purposes of this document, is a hole that has been drilled in the ground to bring petroleum products to the surface. These include oil and natural gas.

Abstract

With the requirement of scheduling gas wells for gas lift operations in the oil and gas industry, the employment of autonomous well-scheduling is highly sought after. A Moxa UC-8100-ME-T Linux computer provides the necessary interface in which an autonomous schedule could be adapted. This device has the potential to interface with the control system being used by the oil and gas company overseeing a method of solution to the well-scheduling problem; the existing control system is comprised of a SCADA system connected to an rPAC controller that controls the choke valves on well-heads.

Several solutions were proposed to use the Moxa for well-scheduling purposes and research and testing was conducted to aid in the solution choice. Two communication protocols were tested for interfacing with the existing network: MQTT and Modbus TCP. The possibility of several programming methods to implement the schedule were also considered. The use of Moxa's ThingsPro software and Python/C APIs, as well as, Node-RED's free and open source visual programming environment, were both considered for implementing the scheduling program and communication protocol. It was concluded that the most efficient method of solution requiring no additional costs or products, aside from the Moxa, was to use Node-RED and a free Modbus TCP node package. All aspects of the interfaces were tested successfully using the method chosen. The full implementation of the well-scheduling solution should take place, first by connecting the system to the corporate control network, and then completing the programming logic for the schedule. Node-RED can handle the communications and the programming and requires nothing else. It is recommended that the program make use of spreadsheet parsing nodes in Node-RED to enable the input of well-schedule data in the form of an Excel spreadsheet.

1. Introduction

North America has been a leader in the petroleum industry since the late 1800's [1]. The information and subject matter contained in this report was procured while undergoing a project under the direction of one of North America's oldest and largest producers of petroleum products [2]. Due to the correlation between pressure in petroleum reservoirs and a having a multitude of well-sites fracking into the same reservoir, the abovementioned company's correlated wells cannot always remain operational at the same time. If a well is shut-off, this will allow pressure to rise within the reservoir for aiding gas lift operations on a separate correlated well; thus, the scheduling of wells and their on-off time is an important consideration. Not all well-heads will yield the same volume of flow and consequently it is important to choose which well flows and which does not. When a well is off, production ceases. In order to maintain efficiency in production and maximize the ensuing margin of profit, the scheduling of well-heads must be enacted with precision and intelligent use of the relevant data required in the construction of a dynamic schedule.

In the oil and gas industry, field devices are used to monitor and regulate various aspects of the process. The field device relevant to this instance of well-scheduling control is essentially an on-off valve. When the valve is in the on-position, resources flow; the converse is true when the valve is in the off-position. These valves are typically called choke valves. Field devices such as the aforementioned valve in the present-day oil and gas industry are typically controlled by a PLC, rPAC, RTU, or an analogous industrial controller. These controllers typically have the ability to perform logic, written to and residing in the controller, in the form of a computer program. They may also at other times, execute logic passed down through integration with a larger SCADA system or DCS. The logic written to a controller typically cannot be easily adjusted without shutting down the machine and re-uploading the program. Figure 1.1 and 1.2, respectively, show a SCADA HMI and DCS.

In regards to the case at hand, an rPAC is used to control the valve. The rPAC is integrated with a larger SCADA system and well-scheduling is performed remotely by input from a human operator. The schedule is contained in an Excel spreadsheet and must be input to the SCADA system separately for each individual well-head valve. The human operator resides in a remote control room and inputs the data from the

spreadsheet via an HMI to the SCADA system. The SCADA system then communicates by way of Modbus TCP with the rPAC on-site. The rPAC, in turn, sends a signal to the valve which causes the valve to open or close. The rPAC used for the purposes of this project and document can be seen in Figure 1.3. Note, that there may be many rPACs all individually controlling a single choke valve.

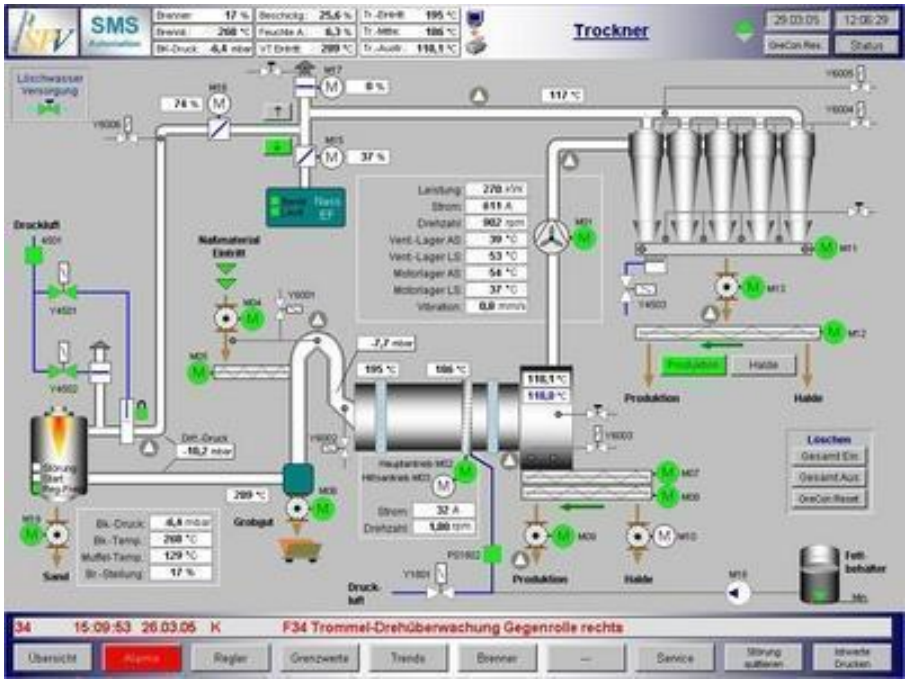

FIGURE 1.1: SCADA SYSTEM HMI [3]

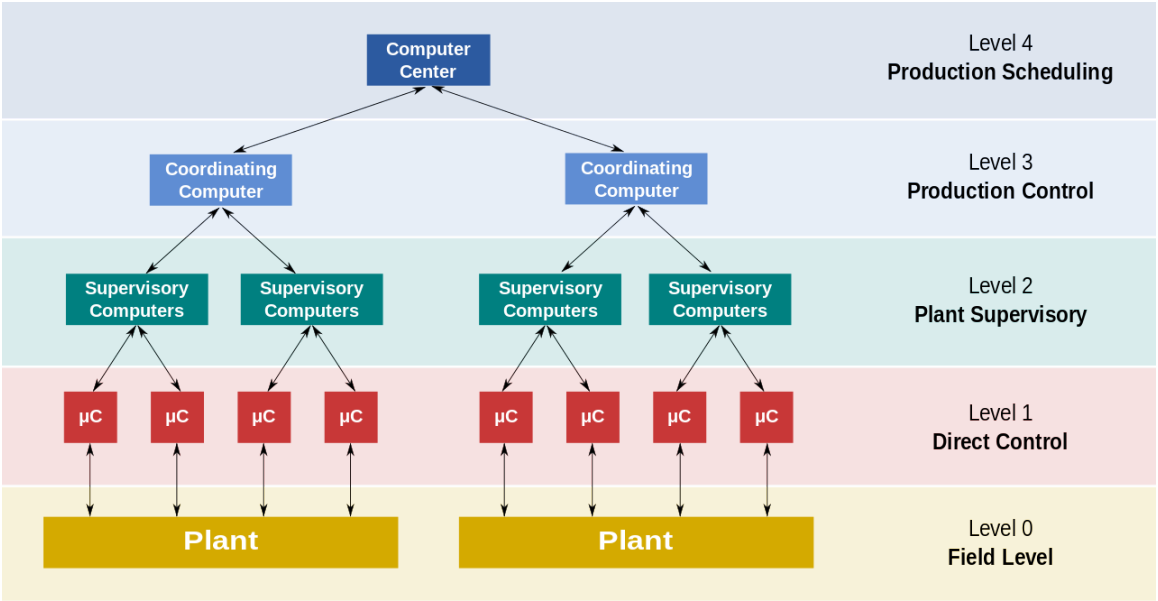

FIGURE 1.2: DISTRIBUTED CONTROL SYSTEM (DCS) [4]

FIGURE 1.3: SCADAPACK 334 RPAC [5]

1.1 The Moxa

It should be obvious that due to the time-consuming nature of inputting the multitude of values contained in the well-schedule and the level of current-day automation technologies, that an autonomous solution for well-scheduling is possible and would be greatly advantageous. The potential autonomous solution would reduce operation costs, increase efficiency, and maximize the margin of profit. For reasons beyond the scope of this document, a headless computer running a Linux operating system was chosen for integration into the pre-existing system to aid in providing a solution to the well-scheduling problem. The headless computer chosen is a Moxa UC-8100-ME-T Series device and can be seen in Figure 1.4. This device is designed to run a software developed by Moxa called ThingsPro. ThingsPro is used for IIoT applications, edge computing, and data acquisition [7]; Figure 1.5 suggests the network architecture of the Moxa device with ThingsPro installed. ThingsPro provides a web-based GUI which can be accessed from a machine on the local network. It is important to note that the Moxa device, although headless, is a Linux computer and thus can perform just about

anything that a regular Linux desktop can in regards to programming. The main limitations in this regard would be what peripherals the Moxa device can support. The UC-8100-ME-T can support a cellular module and SIM card, and has two serial COMM ports, two LAN ports, a USB port, and an SD card slot.

FIGURE 1.4: MOXA UC-8100-ME-T SERIES [6]

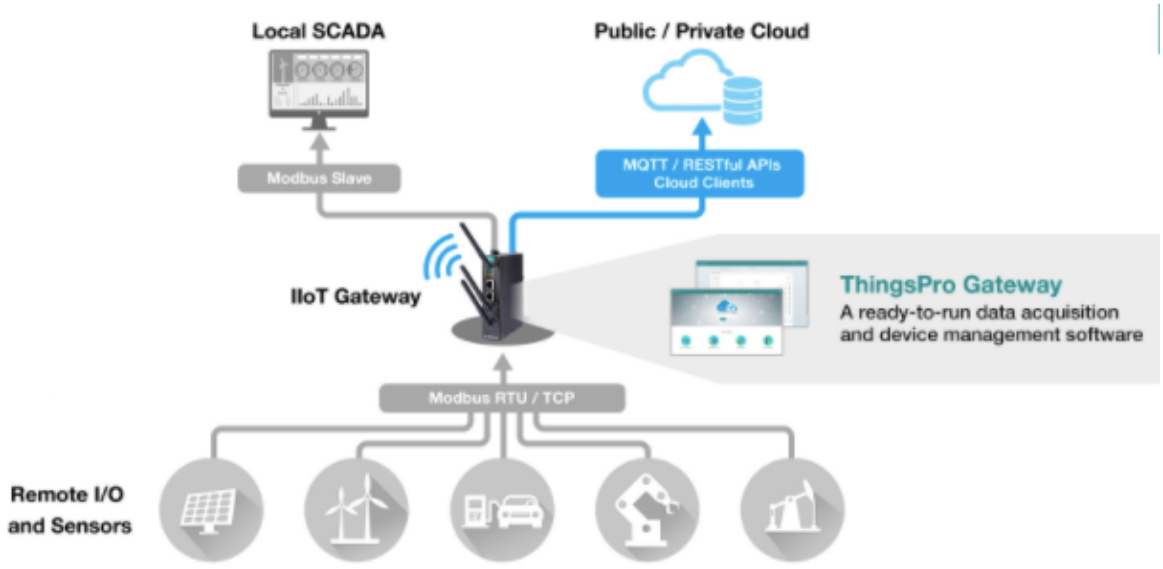

FIGURE 1.5: THINGSPRO NETWORK [7]

1.2 Scope

With the Moxa computer chosen as a precursor for implementing a solution, the problem has thus now evolved into the method of interfacing the Moxa edge computer with the existing corporate control network and implementing the well-schedule as a program running on the Moxa. The method of interfacing and implementation to manage the well-scheduling must provide the most efficient and maintainable system. It is the belief of those involved in the project that this particular issue of well-scheduling has not been previously solved using the technology at hand and thus lacks historical solutions to aid in the integration and solution to any other associated issues. This problem is challenging due to the possible existence of many solutions. There are many factors that must be considered to aid in finding solutions to this problem; the factors are as follows: the communication protocol(s) to be used, the method of programming to implement the schedule, the ability of the Moxa to run certain software, whether or not it is integrable with the current system without the need to acquire other materials or equipment, and its ability to actually write data through the chosen communication protocol as opposed to just acquiring data for storage.

Through research, a few viable solutions were brought to light. These solutions include: using ThingsPro's Modbus framework and Python/C API for programming, using ThingsPro's MQTT client and Python/C API for programming, using MQTT and Node-RED for programming without the use of ThingsPro, and lastly, using Modbus TCP with Node-RED for programming without the use of ThingsPro. The following sections will analyze the aforementioned solutions.

2. ThingsPro

The ThingsPro software, once installed, runs as a service in the background to enable consistent access to the GUI to configure the Moxa computer. Out of the box, the Moxa must be configured through an SSH or serial COMM connection with the Linux terminal.

2.1 ThingsPro and Modbus

The Moxa UC-8100-ME-T, combined with the ThingsPro software, has the ability to poll data via Modbus TCP [8]. There are multiple ways to utilize this capability. Arguably the simplest method would be to use the ThingsPro GUI to create a Modbus template, device tag, and then add the device as described in [9]. Testing determined successful

readability of register values in the rPAC device being used when the "Test" function as seen in Figure 2.1 was pressed. Through the enabling of the Modbus slave function, as seen in Figure 2.2 in the ThingsPro GUI, a local SCADA system can poll this data from field devices connected to the Moxa computer [8]. This can be done using an IoT service such as Azure IoT or AWS IoT, or by a user developed program using the C or Python API; the guide [10], contains detailed tutorials on these processes.

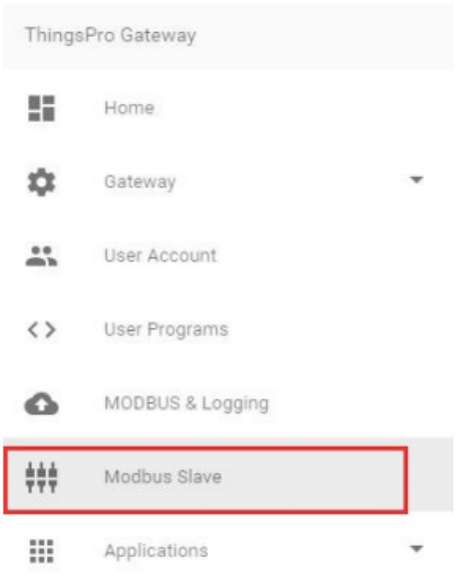

FIGURE 2.1: DEVICE READ TEST [9]

FIGURE 2.2: MODBUS SLAVE FUNCTION [9]

The RESTful API described in the Rapid Development section of [9] enables remote configuration of the Moxa computer through user developed programs. As seen in Figure 2.3, configuration details can be read or written by creating a token that is utilized in a user's program. Through use of a VPN tunnel and ThingsPro's OpenVPN client function, a user may connect to an OpenVPN server to establish secure communications from a remote SCADA system [11]. This essentially turns the Moxa computer into a network router. Thus, the combination of a VPN tunnel and the RESTful API allows for remote configuration, Modbus polling, and direct access to field devices through the routed connection. An example of this routing effect is shown in Figure 2.4.

The aforementioned testing and research appears to show that IoT service applications and user developed programs can directly access field devices through the VPN tunnel, which in turn implies that a remote computer or SCADA system can read or write via direct access while the Moxa is acting as a router and firewall. It is also shown that

device tags and field equipment configuration, as well as the Moxa computers configuration, can be modified using the Python/C API, and RESTful API, respectively. With these abilities, and the development of a program by a user, the Moxa device will increase the efficiency of a SCADA system's polling operations, enable the creation of more sophisticated data logging operations, and read Modbus data from a field device such as the rPAC being used in this project. As seen in [9], ThingsPro provides an easy method to upload user developed programs to the Moxa computer via the GUI. The RESTful API can facilitate a gateway for a remote computer to program the rPAC; it is not inherently obvious, however, that the RESTful API can allow changes or input to the user developed program that may have been uploaded to the Moxa.

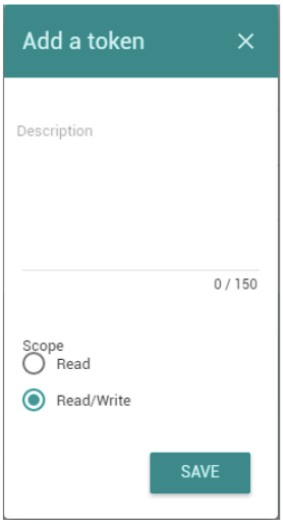

FIGURE 2.3: CREATING A TOKEN [10]

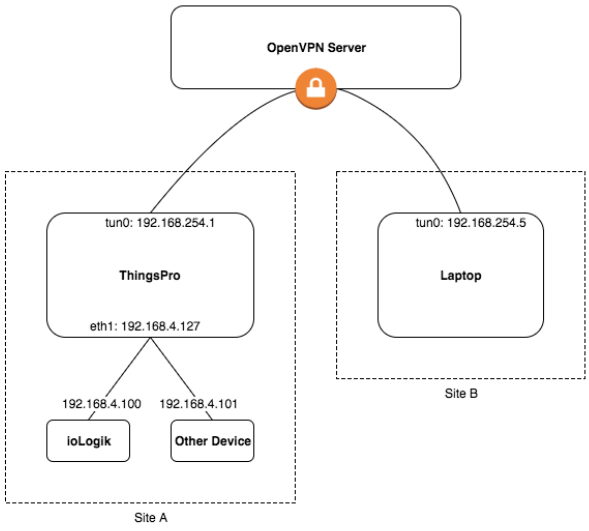

FIGURE 2.4: VPN TUNNEL [11]

2.2 ThingsPro and MQTT

With the ability of ThingsPro's Modbus polling, as spoken about in the previous subsection, we can set a polling interval as seen in Figure 2.5. ThingsPro will then continuously poll data from the connected device at this specified interval [12]. This data can then be uploaded to a corporate MQTT broker using ThingsPro's Generic MQTT client; this is seen in Figure 2.6, and is easily accessible through the ThingsPro GUI. Tests confirmed that register values polled from the rPAC could successfully be uploaded to an MQTT broker. Using the programming software included with the rPAC model used, register values in the rPAC were forced to certain values and this was reflected in the MQTT broker.

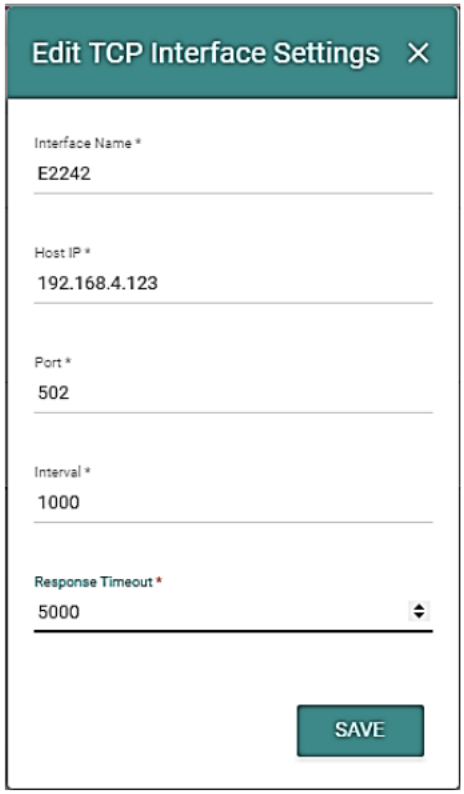

FIGURE 2.5: POLLING INTERVAL [12]

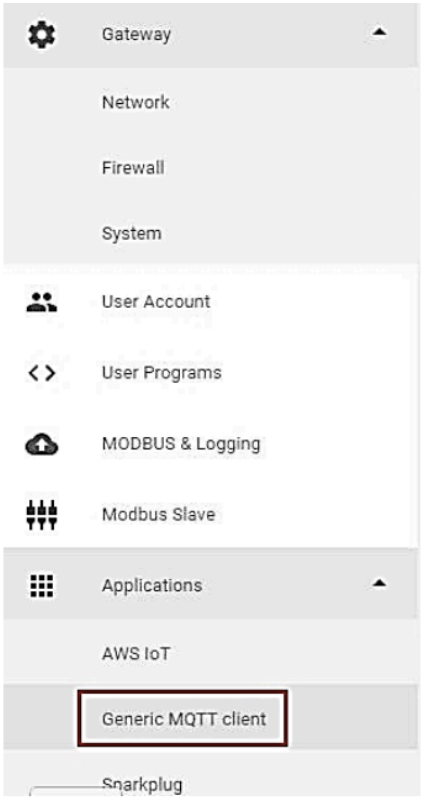

FIGURE 2.6: GENERIC MQTT CLIENT [9]

With the included Python/C API and RESTful API, which enable configuration of the Modbus framework and the Moxa device as discussed in the previous section, configuring the Generic MQTT client in ThingsPro would also be possible. The latter is true because the RESTful API allows access to all configuration options in ThingsPro [10]. Some newer rPACs have support for MQTT, and thus could also subscribe to topics in the MQTT broker. This enables the possibility of an rPAC to subscribe and publish information to/from the MQTT broker, which would essentially eliminate the need for ThingsPro's Generic MQTT client. The rPAC used in this project, however, does not support MQTT and thus to implement a solution in that manner would require all field rPACs to be upgraded.

Successful testing of ThingsPro's Generic MQTT client show that a SCADA system equipped to subscribe to MQTT topics could in turn retrieve polled Modbus data acquired by use of the Python/C API, or through configuration in the GUI, from the Moxa. It is important to note that the Generic MQTT client can only publish data that has been acquired through Modbus polling and is not capable of subscribing to data; therefore, the configuration of the additional Sparkplug MQTT specification, seen in

Figure 2.7 and Figure 2.8, appears to be needed. A lack of information on this specification caused abandonment of testing.

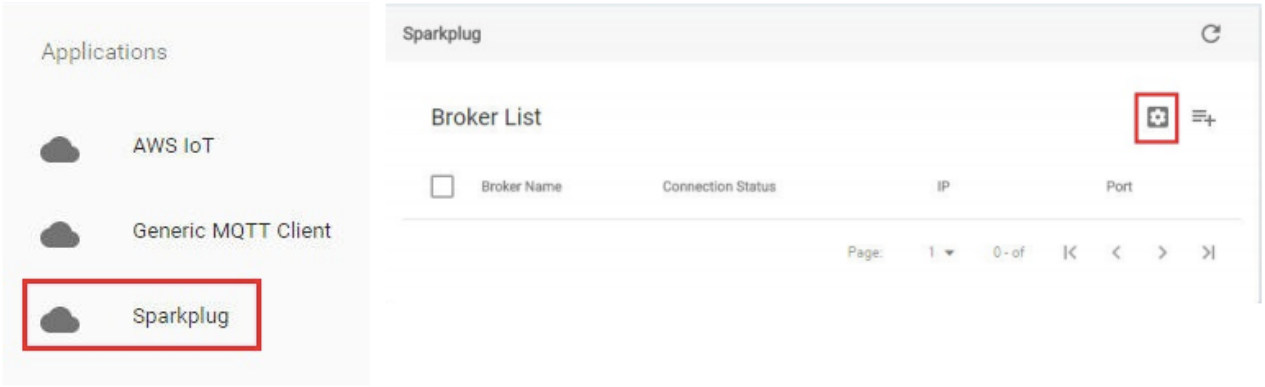

FIGURE 2.7: SPARKPLUG MQTT SPECIFICATION [9]

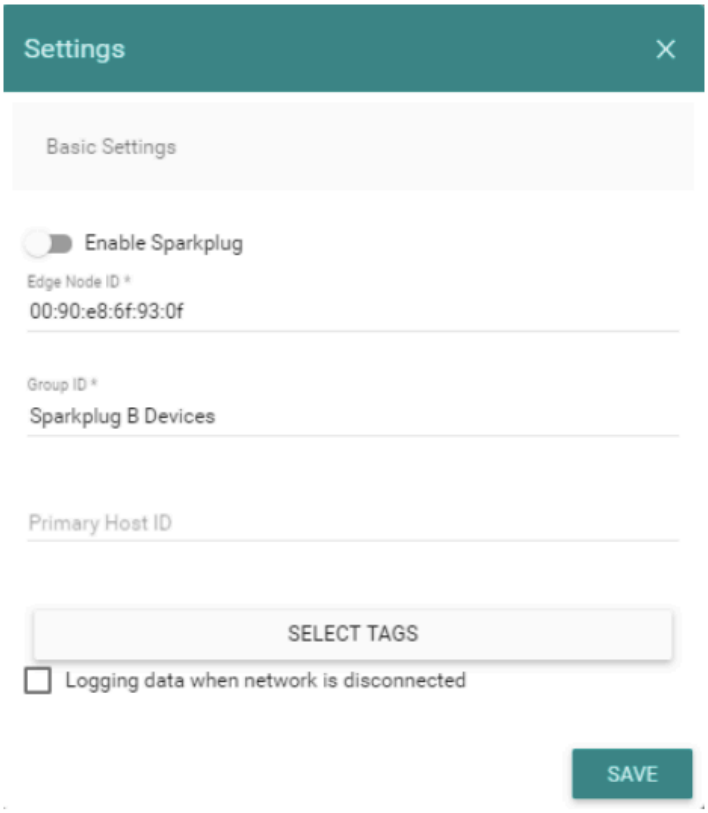

FIGURE 2.8: SPARKPLUG MQTT SPECIFICATION CONFIGURATION [9]

3. Node-RED

Node-RED provides the ability to program for IoT applications using a web-based editor/GUI [13]. This flow-based type of programming uses "a network of black-boxes"

[14]; the black boxes in Node-RED are called "nodes". Since Node-RED is a successful open source software, developers in the community have created many nodes which each perform some type of function. These nodes can be wired together to create a "flow" in the editor and then deployed. See Figure 3.1 for an example flow. Nodes can do anything from providing a local MQTT broker running on the Moxa computer to performing read and write operations in a controller such as the rPAC with Modbus TCP. There is almost a limitless amount of nodes that can be downloaded via Node-RED's Palette Manager (see Figure 3.2); there also exists the possibility for a user to develop their own nodes [17] or use existing function nodes (see Figure 3.3) where JavaScript code can be implemented. Another method of executing functions or programs in Node-RED is the exec node shown in Figure 3.4. The exec node allows a user to run non-JavaScript programs that exist on the computer running Node-RED, that may be required for implementation in a system. Thus, Node-RED has the ability for a user to create a node, have access to a massive database of pre-existing nodes, use the function node to write JavaScript function blocks to manipulate a flow, and execute outside programs with the exec node; this opens an infinite amount of potential when it comes to programming. It is also relevant to note the existence of debug nodes, as well as, scheduling type nodes. The debug nodes greatly aid in the debugging process of a program flow in Node-RED, while the scheduling nodes allow certain elements or nodes in the flow to be triggered at a specified interval, time, or date.

3.1 Node-RED and MQTT

Node-RED provides generic MQTT input and MQTT output nodes as shown in purple in Figures 3.5 and 3.6, respectively. The input node, when configured, will subscribe to a topic from an MQTT broker; conversely, after configuration, the output node will publish a topic to an MQTT broker. A simple search in the Palette Manager, will also reveal that an MQTT broker node exists and can thus be installed to the Palette for use in a user's flow. Figure 3.7 shows the MQTT broker node and further information can be found on Node-RED's website [21]. Simple deployment of a flow containing a configured (see Figure 3.8) broker node, will initiate an MQTT broker on the computer that the flow is running on, in this case, the Moxa computer. The broker node does not require any "hardwired" connections in the flow because it essentially runs the same as any MQTT broker; a client such as an rPAC or SCADA system with access to the network IP address and port that the broker is running on can simply subscribe or publish to topics

in the broker. The broker, MQTT input, and MQTT output nodes were tested and provided seamless connectivity within the LAN. Using ThingsPro's Generic MQTT client, Modbus data was published to the broker running in Node-RED, proving that ThingsPro and Node-RED were compatible. It is expected that the same success would occur with devices outside of the LAN, such as a corporate SCADA network, as long as the connection is routed and port forwarded.

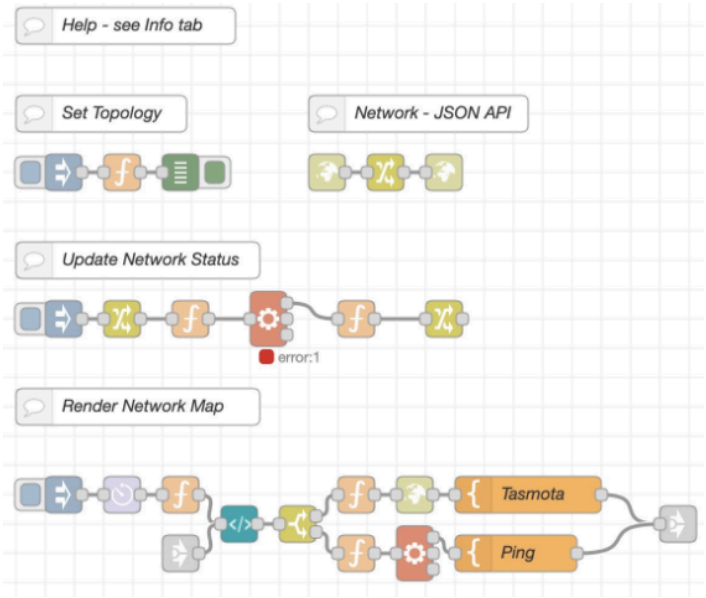

FIGURE 3.1: EXAMPLE FLOW [15]

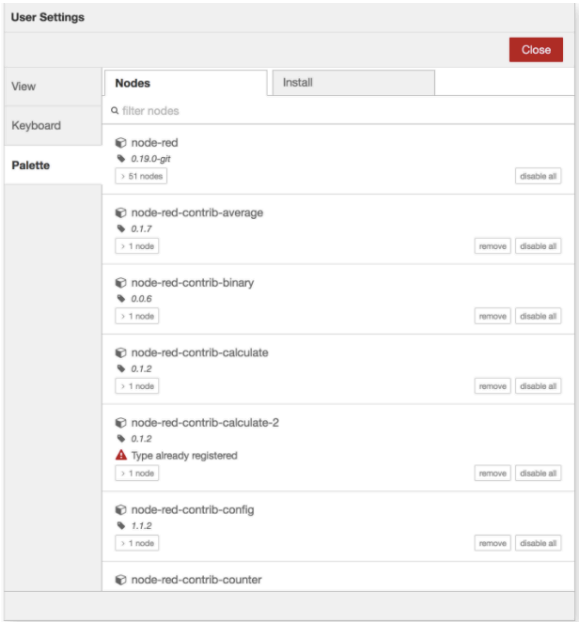

FIGURE 3.2: NODE-RED PALETTE MANAGER [16]

FIGURE 3.3: FUNCTION NODE [18]

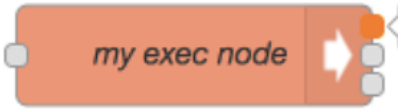

FIGURE 3.4: EXEC NODE [19]

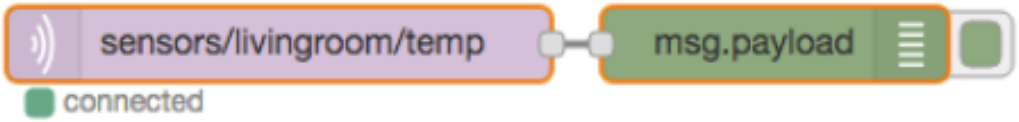

FIGURE 3.5: MQTT INPUT NODE [20]

FIGURE 3.6: MQTT OUTPUT NODE [20]

FIGURE 3.7: MQTT BROKER NODE [21]

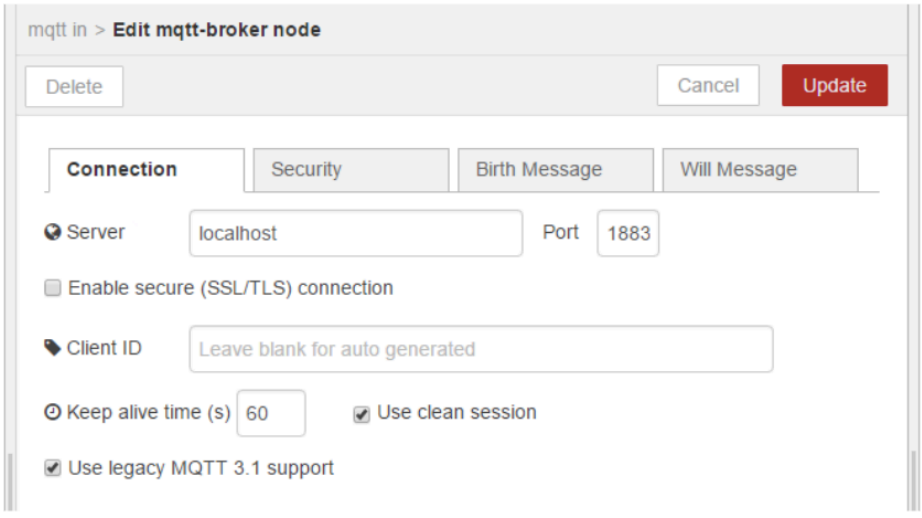

FIGURE 3.8: CONFIGURING THE BROKER NODE [21]

3.2 Node-RED and Modbus

In the same manner that the MQTT broker node was obtained by download from the Palette Manager, a Modbus TCP package can be downloaded. The package of nodes can be seen in Figure 3.9 and [22],[23] both contain information on the package. As with almost all nodes in Node-RED, some configuration is required; Figure 3.10 showed the configuration ability with the broker node.

The most important nodes contained within the Modbus TCP node package, are as follows: Modbus-response, Modbus-flex-getter, Modbus-flex-write, and Modbus-server. Modbus-flex-getter and Modbus-flex-write are the nodes that, respectively, allow the computer running Node-RED to perform read and write operations to devices such as the rPAC controllers via Modbus TCP. The Modbus-server runs similarly to the MQTT broker spoken about in the previous section, and enables the computer running Node-RED to act as a Modbus master or slave. The Modbus-server has a specified port and IP address for Modbus TCP communication, and has a specified amount of virtual coils and registers. The Modbus-response is essentially a debug node used specifically for the Modbus TCP nodes.

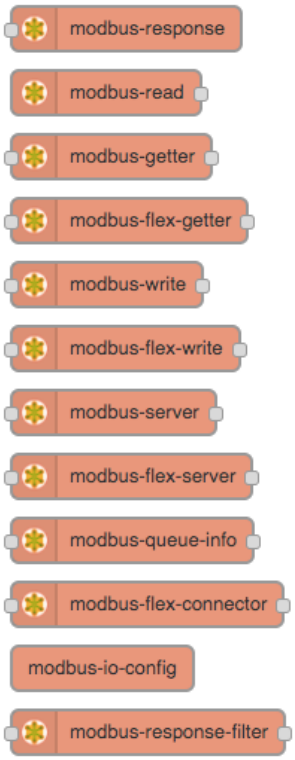

FIGURE 3.9: MODBUS TCP NODE PACKAGE [22]

Due to the seamlessness of the Node-RED software and the multitude of free and open-source nodes, the majority of tests performed for the project were conducted using Node-RED; specifically using Modbus TCP, as proper MQTT infrastructure is not present in the work site. Node-RED was installed on the Moxa computer and the web-based flow editor/GUI was accessed with a laptop residing in the LAN. Tests involved successful read and write operations with Modbus TCP to and from the rPAC involved, logical operations on data retrieved via Modbus TCP, as well as, read and write operations involving a simulated SCADA system.

3.2.1 Read and Write Operations

Using the Modbus TCP package, read and write operations for floating point values were successful to and from the rPAC device. Logic was successfully performed using a function node with JavaScript code on these floating point values; this enabled the program to convert them between readable floats and the unsigned 32-bit binary numbers used by the rPAC. This allows the user to input any floating point number in base 10 (including those with a negative sign) and have it be successfully written to the rPAC. Similarly, floating point values read from the rPAC were output in a readable base 10 format. Read and write operations proved even simpler to and from coils, requiring little to no logic.

3.2.2 Logical Operations

Similarly to the logic performed in regards to floating point values, logic was performed to toggle bits in the rPAC. It was also shown that logic could be performed on data retrieved from one element in the system (such as the emulated SCADA system) to then be passed to the rPAC and vice versa. This confirms that with the inclusion of scheduling nodes, well-scheduling can easily be realized. Simple JavaScript function blocks were implemented within the function node, along with several other nodes that are pre-existing in the node palette to perform the logic.

3.2.3 Emulated SCADA System

A simulated connection to a local SCADA system was used to read and write values to and from the Modbus-server node running on the Moxa computer. This is possible using the virtual coils and registers within the Modbus-server along with Modbus-flex-write and Modbus-flex-getter nodes which sent and received data to and from the Modbus-

server. In the same manner that logic was performed on the rPAC and within the Moxa computer's instance of Node-RED, logic was again performed on data sent and received by the emulated SCADA system.

3.2.4 The "Master flow"

In the end, all of the testing operations were integrated as one. This formed a "master flow". The master flow tested almost all of the would-be requirements for the well-scheduling problem. The master flow successfully tested the following: integration between potentially numerous rPAC controllers on site; the ability to read and write to coils and registers in rPAC devices that would in turn control choke valves; the ability to perform logical operations, such as scheduling on the Moxa computer; using various nodes, the ability to send and receive data from a SCADA system using a master/slave relationship and the Modbus-server; and lastly, the ability to pass information between all three interfaces: SCADA, the Moxa computer, and the rPACs.

4. Summary Analysis

It is of note that Node-RED and ThingsPro were able to communicate with each other through MQTT publish and subscribe; for instance, ThingsPro successfully polled Modbus data from the rPAC and then published it to the broker running in Node-RED. With the ability of Node-RED to execute programs contained on the Moxa with the exec node, any possible combination of ThingsPro API's and frameworks could be combined with the use of Node-RED. At the time of writing this document, the official well-schedule has not been implemented, although, the ability to parse spreadsheet data is available through a multitude of nodes in the Palette Manager. This implies, just as logical operations were performed on data sent between the three interfaces, that the input of a spreadsheet to the Moxa computer (specifically Node-RED), would allow logic to be performed on the schedule contained within the spreadsheet; thus, it would be possible to simultaneously trigger operations on scores of well-sites. All test results are captured with video recording, code, and an instruction set available by way of [24], [25], [26].

5. Conclusions

It is evident that Node-RED is able to provide a solution to the well-scheduling problem without the need for any additional subscriptions, software, or hardware. It is fully

integrable with the current system using only the resources at hand. Node-RED can get the job done independently, while enabling full control of all processes involved. If it can be programmed, Node-RED can do it; the ability to create your own nodes and run programs with the exec node, confirms that even the most obscure problems can be solved in the world of IIoT with Node-RED. In the event that the communication protocol between the SCADA system and the Moxa were to change from Modbus TCP to MQTT across the corporate network, a simple swap-out of the Modbus-server for a combination of MQTT nodes, would solve the problem; the well-scheduling logic would remain the same. Furthermore, in the less-likely scenario that all rPACs were replaced with newer models that support MQTT, the same simple swap out for MQTT input and output nodes for the Modbus-flex-write and Modbus-flex-getter, would solve the problem. Node-RED with its free and ample open-source resources is effective, future proof, and highly maintainable through its user friendly web-based flow editor. The style of programming also opens the door to a programming environment for less experienced developers; some programming knowledge may be required but can be obtained by studying the multitude of available resources.

ThingsPro serves its purpose as a GUI to configure network connections, cell modems, user accounts, and other relevant configurations of the Moxa computer; however, after several days of research into ThingsPro's capabilities for solving programming programs, it was discovered that the documentation appeared convoluted and difficult to comprehend. Many questions remained unanswered, as the user manual left out relevant details, while the datasheet claimed capabilities not seen in the manual. The largest reason for abandoning major testing of ThingsPro was that every time a possible solution path was found it led to another service or subscription. Although ThingsPro enables the uploading of a program to the Moxa computer through the GUI, accessing the program to input scheduling data is convoluted, requiring a possible VPN tunnel and both API's. It appears redundant to use ThingsPro for any programming solutions; however, it may be notable that the ThingsPro API's can be used to configure the Moxa, which includes network configuration options such as the internal cell modem or LAN ports, among other configuration options. As noted previously, the exec node in Node-RED has the ability to run Python and C programs in the Linux environment and thus could make use of the ThingsPro API's in this manner.

6. Recommendations

It is of the opinion of those involved with the well-scheduling solution, that using Node-RED and replacing the simulated SCADA system on the local network with the corporate remote SCADA system, is a logical next step. Existing cell modems on the corporate network can route the connection to the LAN in which the Moxa computer resides. In a case where no existing cell modem is present, the Moxa with its internal cell modem, combined with the ThingsPro GUI, can provide the necessary port forwarding. The latter would imply that the rPACs reside within the Moxa's LAN; in the case that they are on a separate LAN, a combination of the Moxa's cell modem, external cell modem, or router, could be used to forward the port to the rPACs LAN.

With the Moxa officially connected to the corporate control network, the well-schedule must be implemented. Using a spreadsheet parsing node, studying the spreadsheet, and then designing the scheduling Node-RED flow, would provide the final integration. Lastly, determining in which manner spreadsheets will be input to the Moxa's Node-RED instance, and implementing the programming to do so, will provide a means to easily update the schedule when new data becomes available.

7. References

[1] https://www.ektinteractive.com/history-of-oil/

[2] https://www.ovintiv.com/history/

[3] https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.indiamart.com%2Fproddetail%2Fscada-system-4226615233.html&psig=AOvVaw37lJSPaXfY8ayld-eomiqg&ust=1587934516196000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCLiYhpi7hOkCFQAAAAAdAAAAABAD

[4] https://en.wikipedia.org/wiki/Distributed_control_system#/media/File:Functional_levels_of_a_Distributed_Control_System.svg

[5] https://www.rspsupply.com/image/popup?imagePath=%2fimages%2fproduct%2flarge%2fSCADAPack-TBUP334-1A21AB01S.jpg&altText=U0NBREFQYWNrIFRCVVAzMzQtMUEyMS1BQjAxUyAoMzM0IFNlcmllcykgd2l0aCBGcmVld2F2ZSBSYWRpbw2

[6] https://www.moxa.com/en/products/industrial-computing/arm-based-computers/uc-8100-me-t-series#resources

[7] https://www.moxa.com/en/products/industrial-computing/system-software/thingspro-2

[8] https://www.moxa.com/getmedia/41500ce4-64e0-4c69-9e0c-34d09772e133/moxa-thingspro-2-datasheet-v2.1.pdf

[9] https://www.moxa.com/getmedia/147d06a8-86ce-4394-b374-b2ec09333ad9/moxa-thingspro-2-manual-v10.0.pdf

[10] https://thingspro-programming-guide.netlify.app/application-note/develop-program-with-cg-api/

[11] https://thingspro-programming-guide.netlify.app/application-note/managing-field-device-vpn/

[12] https://www.moxa.com/getmedia/4da6fb8f-5288-4c00-bd4d-70654e945d36/moxa-using-moxa-thingspro-and-mqtt-tech-note-v1.0.pdf

[13] https://nodered.org/

[14] https://nodered.org/about/

[15] https://flows.nodered.org/flow/4b940430b92142571c670050f4d98f6d

[16] https://nodered.org/docs/user-guide/editor/palette/manager

[17] https://nodered.org/docs/creating-nodes/

[18] http://www.steves-internet-guide.com/node-red-functions/

[19] https://nodered.org/docs/creating-nodes/appearance

[20] https://cookbook.nodered.org/mqtt/connect-to-broker

[21] https://flows.nodered.org/node/node-red-contrib-mqtt-broker

[22] https://flows.nodered.org/node/node-red-contrib-modbus

[23] https://github.com/BiancoRoyal/node-red-contrib-modbus/wiki

[24] https://www.youtube.com/playlist?list=PLqM5bz4H7e4t_iOdId9cCQrjlTRlbSI1C

[25] https://gist.github.com/Nolan-Walker/440ff7c7ac28199258e6af05c6025d9c

[26] https://gist.github.com/Nolan-Walker/b409695219ed31774d95458be9e5df99