November 2020

Monitoring Of Remote Hydrocarbon Wells Using Azure Internet Of Things

Derek W. Staal
Louisiana State University and Agricultural and Mechanical College

MONITORING OF REMOTE HYDROCARBON WELLS USING AZURE INTERNET OF THINGS

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science

in

The Department of Engineering Science

by
Derek Warren Staal
B.S., Louisiana State University, 2018
December 2020

This thesis is dedicated to Ronald Dale Staal.

"Dark times lie ahead of us and there will be a time when we must choose between what is easy and what is right."

—Albus Dumbledore
Harry Potter and The Goblet of Fire

Acknowledgments

To my wife, Emily, your patience and unwavering support throughout my college career means more to me than you will ever know. I couldn't have done any of this without having you in my corner and I am forever grateful. I love you. Now we can move on with our lives, finally!

Shout out to my former Senior Project team, the Montmorillo-Knights. Working with all of you was one of the greatest experiences of my college career. We learned, laughed, and dominated the competition, setting the new standard for future Petroleum Engineering Senior Projects at LSU. I wish you all the best in any and all endeavors you may partake in. Here is to the future, Cheers!

To my committee, you have all served vital roles in my success. Dr. Tyagi, a fellow Montmorillo-Knight, you have been instrumental in my academic success for the duration of my time spent at LSU. You took a chance on all of us in undergrad and an even bigger chance on me in graduate school. Thank you for all of the support and understanding given throughout this undertaking.

Dr. Chen, your work opportunities literally afforded me the option to finish graduate school. Without you I wouldn't have been able reach this milestone. Thank you for integrating me into your research team.

Dr. Knapp, I had very little knowledge of computers when I transitioned from Petroleum Engineering to Information Technology Engineering. Your classes were overwhelming to say the least, however, the vast sea of information you exposed me to opened my eyes to many new found passions that otherwise would have gone undiscovered. Thank you for sharing your time and knowledge.

Table of Contents

List of Tables

List of Figures

Abstract

Remote monitoring of hydrocarbon wells is a tedious and meticulously thought out task performed to create a cyber-physical bridge between the asset and the owner. There are many systems and techniques on the market that offer this solution but due to their lack of interoperability and/or centralized architecture they begin to fall apart when remote assets become farther away from the client. This results in extreme latency and thus poor decision making. Microsoft's Azure IoT Edge was the focus of this writing. Coupled with off-the-shelf hardware, Azure's IoT Edge services were integrated with an existing unit simulating a remote hydrocarbon well. This combination successfully established a semi-autonomous IIoT Edge device that can monitor, process, store, and transfer data locally on the remote device itself. These capabilities were performed utilizing an edge computing architecture that drastically reduced infrastructure and pushed intelligence and responsibility to the source of the data. This application of Azure IoT Edge laid a foundation from which a plethora of solutions can be built, enhancing the intelligence capability of this asset. This study demonstrates edge computing's ability to mitigate latency loops, reduce network stress, and handle intermittent connectivity. Further experimentation and analysis will have to be performed at a larger scale to determine if the resources implemented will suffice for production level operations.

Chapter 1. Introduction

1.1. Monitoring of Remote Hydrocarbon Wells

The processes involved in the discovery and extraction of hydrocarbons from within the earth are as artful as they are scientific. Decades of trial and error as well as research and development have gone into the field of Petroleum Engineering. Not all hydrocarbon wells are created equal, each and every one of these wells are unique and came with their own set of trials and tribulations. For every one of these wells that were fruitful many more were bust. The common ground between these fruitful wells is the equipment and techniques that are applied in order to produce and continue to produce for as long as possible while it remains economically viable to do so.

The recovery of product from a hydrocarbon well can be broken down into three stages: primary recovery, secondary recovery, tertiary recovery. Primary recovery involves the use of the natural energy within the well such as gas or water. Secondary recovery is the process of injecting gas or water from the surface into the well to compliment the natural energy that existed in the primary stage. Tertiary recovery aka Enhanced Oil Recovery(EOR), is used in wells that can no longer utilize the aforementioned recovery methods. There are an abundance of techniques that can be applied to late life wells, all dependent on given circumstances, but are out of the scope of this writing. There is however an overlap of necessity for these wells, regardless of what stage of production they happen to be in.

A hydrocarbon well, consisting of either gas, oil, water, or a combination of the three follows similar design principles and is bound by the laws of fluid mechanics and

petrophysics. For production optimization the operator must obtain well pressures, flow rates, temperatures, fluid levels, valve positions etc. Every well must also maintain the standards set pertaining to safety. This consists of Safety Integrated Services (SIS). SIS is made up of an Emergency Shut-Down system(ESD), Process Shut-Down system(PSD), and Fire and gas systems. All of these attributes must be monitored and controlled for the duration of the well's life to ensure safety and productivity.

Due to the depletion of natural resources from easily accessible locations, oil and gas companies have had to pursue hydrocarbon deposits in remote areas. These areas range from arctic to desert, onshore to offshore, with the only commonality being the harsh, unforgiving, and desolate waste lands they reside in. These environments present a unique challenge for companies that require the remote monitoring and controllability essential to maintaining safe, productive, and economically viable assets.

With these difficult conditions comes the necessity for quality data analytics. Having success in this area allows for a client to improve efficiency in production optimization, structural integrity, preventative maintenance, and environmental safety. Falling short with any of these can and will result in lost time, and in the Oil and Gas industry that equates to heavy financial losses.

Current standards in remote monitoring solutions utilize an architecture that is quickly becoming obsolete. Data volumes have grown exponentially as a result of the development and application of machine learning algorithms. With this bottomless data lake comes the unforeseen latency loop induced by offsite data processing and slow or intermittent connection speeds.

1.2. Problem Statement

Unavoidable depletion of natural resources from easily accessible locations has forced Oil & Gas companies to explore remote areas in their search for black gold. The farther away these assets become the more a cloud-based, or a data center processing architecture breaks down, resulting in a perpetual state of latency, and thus, poorly informed decision making. This continuous delay with remote assets becomes an operational disconnect impeding production optimization, preventative maintenance, structural integrity, and environmental safety.

Edge computing offers a potential solution to this problem by implementing cloud-based resources at or near the source of the data. There is a lack of research pertaining to the practical application of Microsoft's Azure IoT services in real world operations. The application of edge computing using Azure services may be beneficial in reducing decision latency, lowering network stress, and managing intermittent connectivity, all traits inherent to cloud-based and data center solutions when applied to remote monitoring and control scenarios.

1.3. Research Objectives

To tackle this problem an in-depth analysis and application of Industrial Internet of Things(IIoT) using Azure IoT Edge will be performed. This technology aims to establish a more efficient and practical relationship between the client and the targeted remote asset. The goal is to create a solution that offers less "hands-on" while performing at a higher level than data center based SCADA and centralized IIoT solutions. This implementation of edge computing will strive to establish an autonomous system capable of

processing data at the source. This entails sifting through data inputs, differentiate between normal and abnormal operations, make adjustments if needed, and sending batch telemetry to reduce data transfer volume. Research objectives for this study include:

1. Perform an in-depth analysis on how Azure IoT Edge computing works

2. Integrate Azure IoT Edge with Bench Model as Use Case

3. Determine if Azure IoT Edge services offer a practical and viable edge computing solution capable of monitoring a remote asset in the Oil & Gas sector

Chapter 2. Background for Monitoring and Control Methods

There have been many solutions in place that enable the monitoring and control of hydrocarbon wells. The natural progression of monitoring and control, as technology has advanced, can be mapped throughout the existence of modern hydrocarbon production.

2.1. Manual Interaction with Field Devices

A prevalent method for the monitoring and control of remote hydrocarbon wells consists of a production operator aka "well tender" or "pumper" physically traveling to the designated location. This is quite the endeavor, these hydrocarbon wells are scattered all over the country, onshore and off, away from cities and other developed areas. Reaching these locations involves personnel, fuel, vehicle maintenance, and time. Once on location personnel can determine the status of the equipment, structural integrity, production volumes, pressures, temperatures, etc. through the use of field devices e.g. gauges. The operator can also manually adjust valves and equipment to optimize production and alter flow paths. After leaving the job location the pumper will relay the current situational report and data to management.

The manual method is still used by many oil companies for reasons of necessity. Since truly autonomous systems don't yet exist, personnel are still required for maintenance, repairs, work-overs, and production enhancements such as chemical injections. However, when it comes to monitoring and control, the process of relying on workers to relay accurate data in a timely manner is one of tiring monotony and antiquity. It goes without saying that this process involves exposing workers to unnecessary risk, latency in production, and the inability to perform any reasonable data analytics.

2.2. Field Devices Plus RTU, PLC, and SCADA

2.2.1. RTU

A Remote Terminal Unit(RTU) is "standalone data acquisition and control unit... which monitors and controls equipment at some remote location from the central control room."[16]. These units come pre-programmed for their inherent abilities but must be configured to job specifications. This can be done at the unit itself or by the uploading of updated configurations from said central control room.

RTUs range from small to large, determined by the count of onboard analog and digital terminals. Small units have 1–30 analog and digital inputs/outputs; medium units have 31–100 analog and digital inputs/outputs; large units consist of anything greater than 100. Their power needs are provided by battery and solar. The solar panel is not used for direct system feed but used to charge local batteries.

The RTU meets all the requirements for computation: input, storage, processing, and output. All of the systems functions can be viewed and controlled through the use of a human machine interface(HMI) located on the unit(if desired) as well as from a control center. The name "Remote Terminal Unit" is fitting, the key function of an RTU is to transfer data over wide geographical areas to a receiving and transmission point. The RTU unit accomplishes this by using wireless(Cellular-3G-4G-LTE CAT M1/Satellite/radio-900Mhz) to send and receive telemetry to and from a designated location. An RTU can be implemented individually or in clusters, sometimes transmitting their collective telemetry to a Master Terminal Unit(MTU) where it is consolidated and transferred to a control center. An RTU was designed to withstand the harsh en-

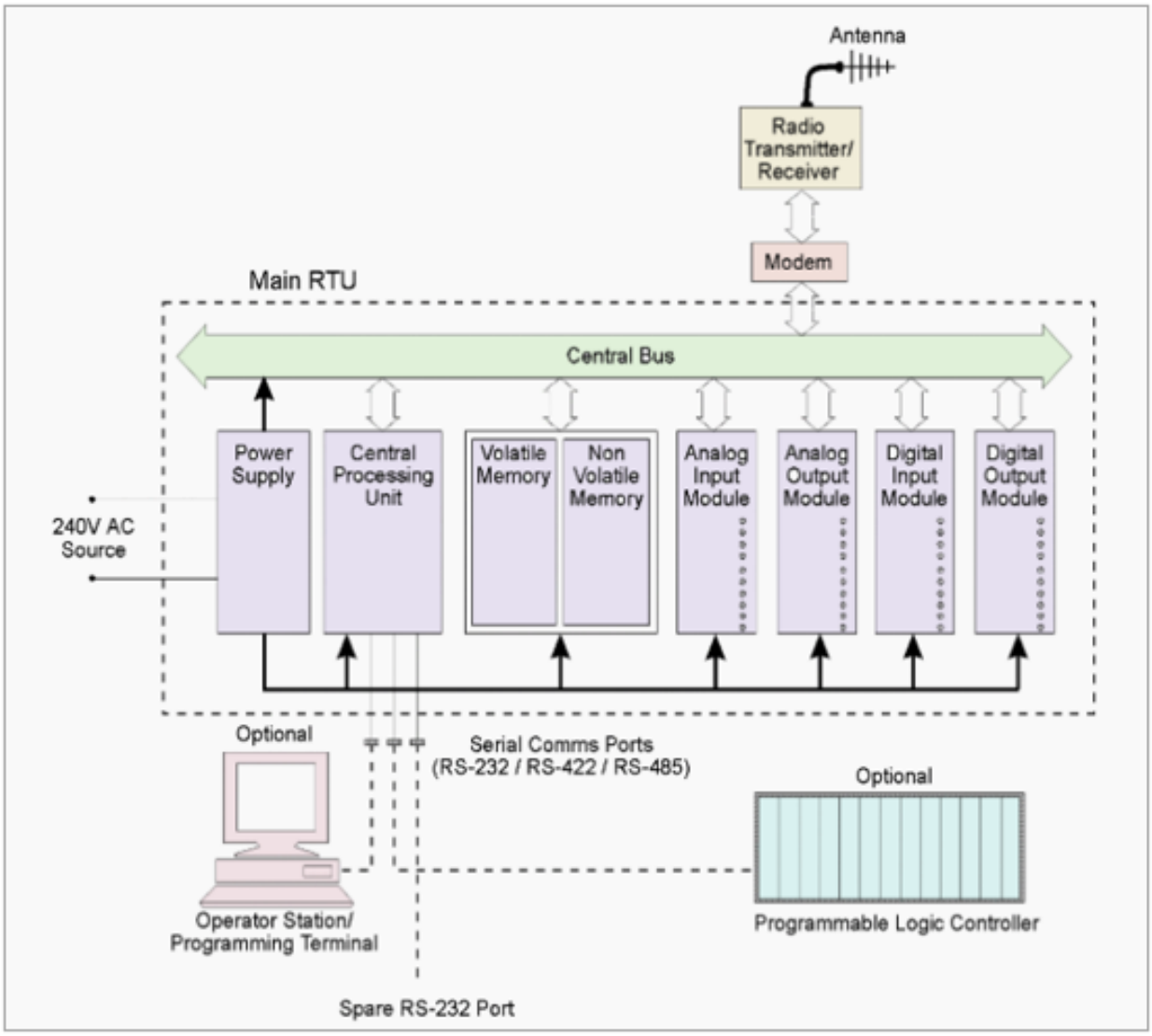

Figure 2.1. RTU Diagram [16]

vironments associated with remote installations. High temperatures, low temperatures, electricity scarcity, signal scarcity, whatever it is the RTU was designed to function well under stress making it the preferred equipment for remote applications in industries such as oil & gas, water, and power transmission.

2.2.2. PLC

The programmable logic controller(PLC) "was born on New Year's Day, 1968" [1] but came into popularity in the 1970s as a replacement for hard wired industrial relays and timers which could only perform one designated job. If that job scope needed to be altered then new relays, timers, and wiring needed to be adjusted accordingly, costing labor, time, and therefore money. PLCs are industrial computers that consist of a power supply, CPU, memory, and their own control-programming software with a specific language(LD, SFC, FBD, ST, IL), as well as connection options for electronic input and output modules. The PLC package offers a more powerful and flexible solution to the industrial workplace than that of its predecessor; designed so proficiently it can be found in virtually all industrial processing and automation infrastructure today.

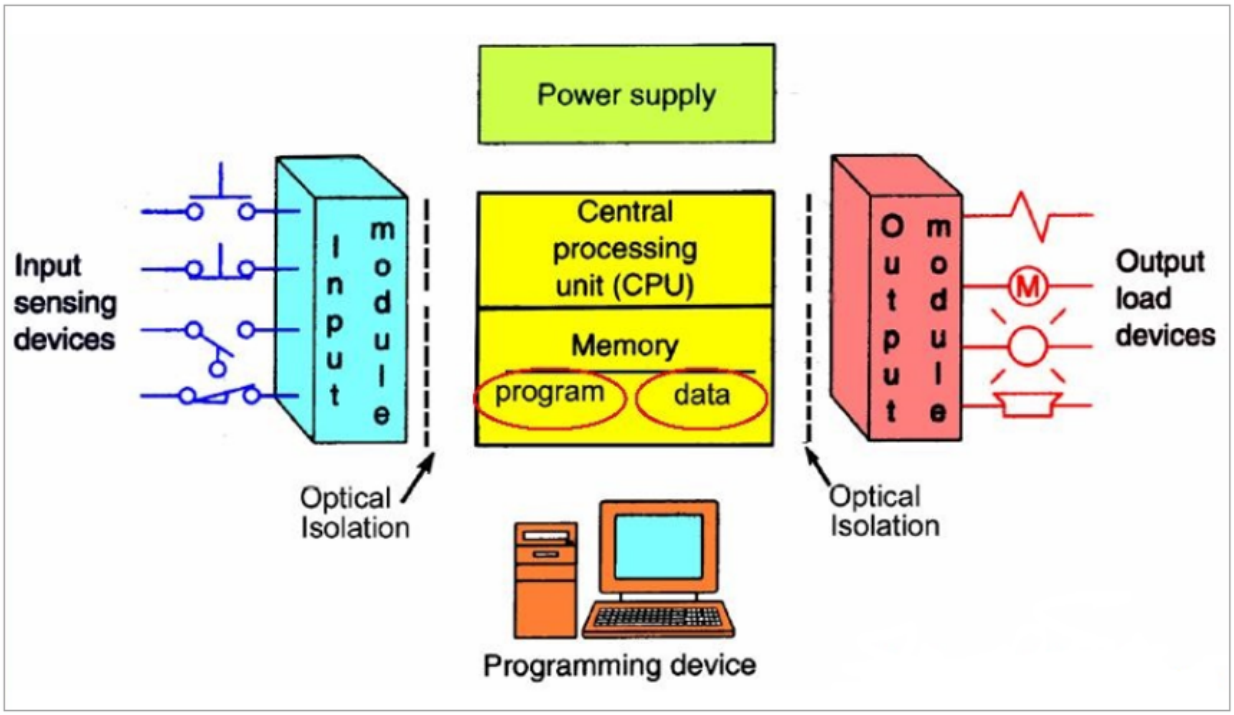

Figure 2.2. PLC Diagram [38]

Unlike the RTU, the PLC doesn't come with programming already installed. There is freedom in this because the client can design the PLC functionality exactly as they want it. This freedom lowers its initial shelf cost when compared to an RTU but increases its overall cost due to the essential personnel that must be fluent in the programming and installation of the unit. Like an RTU, the PLC can also have an HMI installed on the unit enabling complete access to system functions while on location.

There is a lot of overlap of functionality between an RTU and PLC. Features that distinguish an RTU from a PLC are communications capability, offering improved data logging and history backfill. An RTU can locally store important data during a communication outage or a period of limited bandwidth, allowing recovery of said data at a later time. As the years pass more and more functionality is made available for both units, eventually rendering their performance attributes identical. As of now, the PLC excels as the workhorse of control operations, leaving the burden of communication to the RTU.

2.2.3. SCADA

Supervisory Control and Data Acquisition(SCADA) can be defined as the entirety of the system, endpoint to endpoint, from the source of the data in the field to the SCADA server and HMI(s) at the control center. Connected to RTU(s), potentially working in tandem with various PLCs, at the work site via a wired or wireless interface, SCADA is designed around the operators. This enables them centralized access to monitor telemetry and perform any actions within the capabilities of the RTU and PLC setup. SCADA systems can alter the functionality of the RTU or PLC individually or as a group(block programming) from the designated control center. From these locations "the

accurate and timely data(normally real-time) allows for optimization of the operation of the plant and process" [16].

These data, are routed into SCADA servers e.g. OPC server, from which these data can be further routed through a firewall to more interactive-friendly storage servers(hard or cloud), of various purposes and roles such as viewing equipment status, alarms, and trends. It is from these servers that more data contextualization, processing, and in-depth analytics can be performed without slowing or overloading the main SCADA server with database queries.

This advancement reduces the amount of exposure a worker will face when compared to the manual method. This drastically limits the need for workers to be at the location of each individual well and they will typically only visit a well if there are malfunctions and repairs are needed, preventative maintenance, work-over, or the addition of production enhancements such as chemical injections. Currently, SCADA systems are the most common form of remote monitoring and control. They form a real time(or close to it) data management and control system that is ruggedly built and can therefore withstand the harsh environments associated with remote hydrocarbon wells.

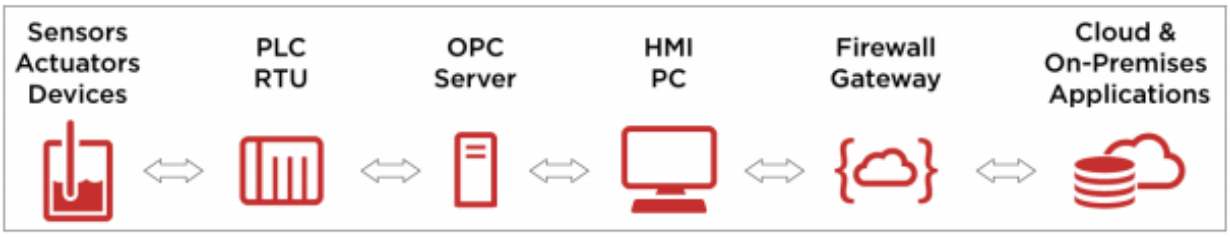

Figure 2.3. Overview of Typical SCADA Solution [28]

Shortcomings of SCADA have been made painstakingly clear over the years for customers and engineers trying to upgrade or integrate systems. Integration of SCADA sys-

tems are limited to the model, version, and protocols being used and therefore considered "standalone devices that do not exchange data well with other SCADA systems of different make and model, enterprise resource planning systems(ERP), or decision support systems(DSS)" [30]. These SCADA systems "use protocols and networks that are proprietary or automation specific(EtherNet/IP, Modbus, Profibus, serial, OPC). Computers and mobile devices use standard Ethernet or wireless networks and open protocols and standards, like TCP/IP, HTTP/HTTPS, JSON, and RESTful APIs" [28]. These systems are lacking interoperability, the only workaround being the installation of required and expensive middleware that bridges the gap between industrial equipment networks/protocols and standard network/protocols. It is also worth mentioning that SCADA architecture can be regarded as segregated solutions by design, which can potentially limit advancements in performance capabilities. This is due to all data having to be acquired and directed to a single location before it can be processed and pushed farther down the line.

2.3. Industry 4.0

Over the past 250 years humans have reached several milestones in the field of industrial processing. These milestones have been documented as "Industrial Revolutions," and to date there have been four.

1. Mechanization(˜1765): The advent of external power such as wind, water, and steam to operate equipment. This mechanization increased production loads tremendously. The steam engine would be refined and later implemented into railroad operations.

2. Electricity, Gas, and Oil(˜1870): Technological advancements led to discovery and harnessing of electricity, gas, and oil in the field of industries. This paved the way for steel demand, as well as historic inventions including the internal combustion engine, the assembly line, chemical synthesis, telegraph, and telephone.

3. Computers and Partial Automation(˜1969): Through the use of memory-programmable controls and computers, industrial processes could be partially automated, introducing a more sophisticated level of monitoring and control. This led to the advancement of electronics, telecommunications, and computers.

4. Cyber-Physical Systems and Internet(Now): Real-time monitoring and control of industrial equipment using a network of sensors, computers, communications, and historic data.

This next stride or movement can be traced back to ˜2009 but has really began to take hold in the past few years. The idea behind Industry 4.0 is to take raw data that isn't very helpful, manipulate and process it, transforming it into something of value, information. This information is how effective decisions are made. Industry 4.0 is the end goal whereas digital transformation and IIoT are the steps needed to reach it.

2.3.1. IIoT

This brings us to the third method, Industrial Internet of Things(IIoT), which isn't widely used in the petroleum sector but slowly gaining popularity. As mentioned above IIoT is an important mechanism that enables a client to experience the 4th Industrial Revolution. IIoT is the connection of digitally transformed company resources in their en-

tirety, from business resources to field resources. It achieves this through a unified namespace utilizing an open protocol such as Message Queuing Telemetry Transport(MQTT). All data generated from tools such as CRMs, ERPs, DSSs, MESs, MMSs, SCADA, PLCs, Smart Sensors, etc., are sent to this unified namespace where they are identifiable and their data accessible cross-platform, whenever and wherever the client wants it. In the end, IIoT affords industry the capability of making decisions based off realtime data, not historic data.

IIoT is a seamless, efficient, and cheaper alternative to outdated, disparate, data silo type systems. IIoT prides itself on interoperability, one connected network comprised of all administrative and industrial units, each of these capable of independently exchanging data with each other as well as the cloud. IIoT can be installed from the ground up or integrated with existing infrastructure such as RTUs and PLCs.

IIoT solution architecture enables remote access of assets from anywhere in the world with an internet connection and user authentication. Depending on the application, this design can improve upon the previously mentioned data center based SCADA systems by offering a more centralized approach. IIoT is designed to establish a path between all assets, via a unifying open communication protocol, and their users, enabling the sharing of Device to Cloud(D2C), Cloud to Device(C2D), and Machine to Machine(M2M) messages. This cloud-based solution architecture allows for the implementation of powerful data management resources such as large data storage and predictive analytics, eliminating the need for the user to purchase and maintain in-house data center infrastructure. Ultimately, the difference between SCADA and IIoT architecture is that IIoT is designed to put the data that people need in their hands, when and where they want it.

A unified architecture involving all available assets is the most efficient solution that industrial businesses will have seen to date. If the focus of this writing was manufacturing or plant operations then the discussion could end here, however, the oil and gas sector comes with its own challenges that need to be addressed. One of which is the topic of this study, the remote monitoring of hydrocarbon wells that are located extremely far from a clients' unified namespace. The concern is that of decision latency which is ripe within the process of collecting field data at long range, sending it to the cloud or control room, analyzing the data, making a decision, and then sending decision control commands back to the field.

2.3.2. IIoT Edge

Edge computing is the latest advancement in the world of Industry 4.0. The idea is to harness the power of cloud-based resources and make them available at the source of the data. This puts intelligence and responsibility at the edge device itself, reducing latency of critical decisions, and limiting the need for human interaction. Intelligent edge devices are where opportunities for rich IIoT solution architecture lies, if that is what is truly needed. The most important thing one can learn and understand about technology is when and where to apply it. According to [24], a few circumstances in which an IIoT Edge solution can be beneficial are:

1. Industrial Automation
2. Enhanced Analytics
3. Occasionally Offline
4. Protocol Translation

Industrial Automation involves the constant monitoring and control of processing methods to reduce human labor requirements and improve efficiency. By narrowing the window of anomaly ignorance within assembly the producer can finely tune performance and even predict defective products. This increases the consistency of the production environment yielding safer and more reliable returns.

Enhanced Analytics involves applying machine learning algorithms as close to the source of the data as possible. This way the algorithm is ingesting the input in real time and therefore can output a decision faster. This drastically reduces decision latency when compared to the original method of applying machine learning methods to historic data which resides far from the data origins in both time and space.

Occasionally Offline happens when connectivity is sparse or non-existent temporarily. This can also occur when a low power system is designed to only power up once and a while, long enough to transmit telemetry before returning to sleep mode again. With an intelligent edge solution the data can be stored and processed locally. This means that there is no concern pertaining to lost data within the workflow. The edge device receives the data, stores it, processes it onsite to make control decisions, and if asked it sends batch telemetry to the cloud once connectivity has returned.

Lastly, IIoT Edge applications do quite well in situations where ingested data comes from legacy or proprietary networks/protocols in the field. These networks/protocols are not compatible with standard TCP/IP and therefore must be translated. An intelligent edge device can act as a gateway, translating these communications on location and forwarding the data in a way that can be understood and used in the job scope.

Any problem a user is facing that involves one or more of these situations can be

approached using an application of intelligent edge computing. There are hindrances that come from unnecessarily applying an IIoT Edge adaptations but for the aforementioned uses this technology excels and is therefore a justifiable and appropriate solution path to take.

Chapter 3. Theory

The focus of this study is to implement Microsoft's Azure IoT Edge resources and see if they offer a viable solution for the monitoring of a remote asset. The following sections will discuss the resources being applied and what purpose they will serve.

3.1. Microsoft's Azure IoT Edge Services

There are many approaches one can make when tasked with the development of an IIoT Edge solution. For the extremely tech oriented it is feasible to build the system from the ground up; everything from the edge device(s), network, node mesh (for WIFI boosting), servers, cloud resources, and all of the inner workings involved with reaching these milestones. This of course is a laborious undertaking with higher costs, personnel requirements, and headache.

Microsoft has solved most of these issues by granting public access to tools needed for IIoT Edge solutions with their Azure IoT Edge services. Azure IoT Edge is available through the Azure portal and comes with a development package personal users and businesses will need for creating individualized IIoT infrastructure. This consumption-based cost model allows users access for free, only paying for the tools utilized. This can be used by do-it-yourselfers on a low level or scaled-up and applied to industrial high-tier projects. When the term scaled-up is used it means just that. Azure IoT is more than capable of hosting thousands of devices all from the same account.

Azure is a convenient, affordable, and powerful service designed to provide a plethora of scalable and user-friendly solutions. It is through Azure's IoT services that the software side of this study's remote monitoring solution will be derived. The Azure Portal

serves as a easily accessible virtual primary in which all other Azure resources can be created, updated, deployed, and deleted. For this study's IIoT Edge solution the following resources will be used:

1. IoT Hub
2. Container Registry
3. Azure Resource Manager
4. Storage
5. Stream Analytics

3.1.1. IoT Hub

IoT Hub is the central bi-directional communication message bus. It is the middleman that connects the device to the application. This is where all of the user's IoT devices, standard or edge, are added, managed, or removed. It holds all of the device identities and their respective connection strings and keys. The IoT Hub is responsible for providing a secure method of communication for messages coming to and going from the cloud. These devices have to be trusted along with each and every one of their messages. The C2D messages naturally raise some concerns. These command or control messages have greater potential for being maliciously used. It is through the use of a Representational State Transfer Application Programming Interface(REST API) that Azure IoT Hub alleviates these concerns. "Once the device credentials have been verified and device messages start flowing into the IoT Hub instance, messages can be routed to other Azure services including Blob, Table and Data Lake storage, Service Bus Queues and Topics, Event Hub, and Stream Analytics" [24].

3.1.2. Container Registry

This private registry is where custom containers, essential for job specific functionality of an IIoT Edge solution, are pushed, stored, and managed. It is from here that the later discussed edgeAgent accesses modules located inside the containers(Subsection 3.1.6). The container registry can also be configured to automatically update containers when modifications are made by the user. This is another important tool for edge computing if the solution requires custom built modules.

3.1.3. Azure Resource Manager

This provisions all of the user's resources under their subscription. It is a deployment and management service enabling the user to create, update, and delete subscription resources. "You use management features, like access control, locks, and tags, to secure and organize your resources after deployment" [3].

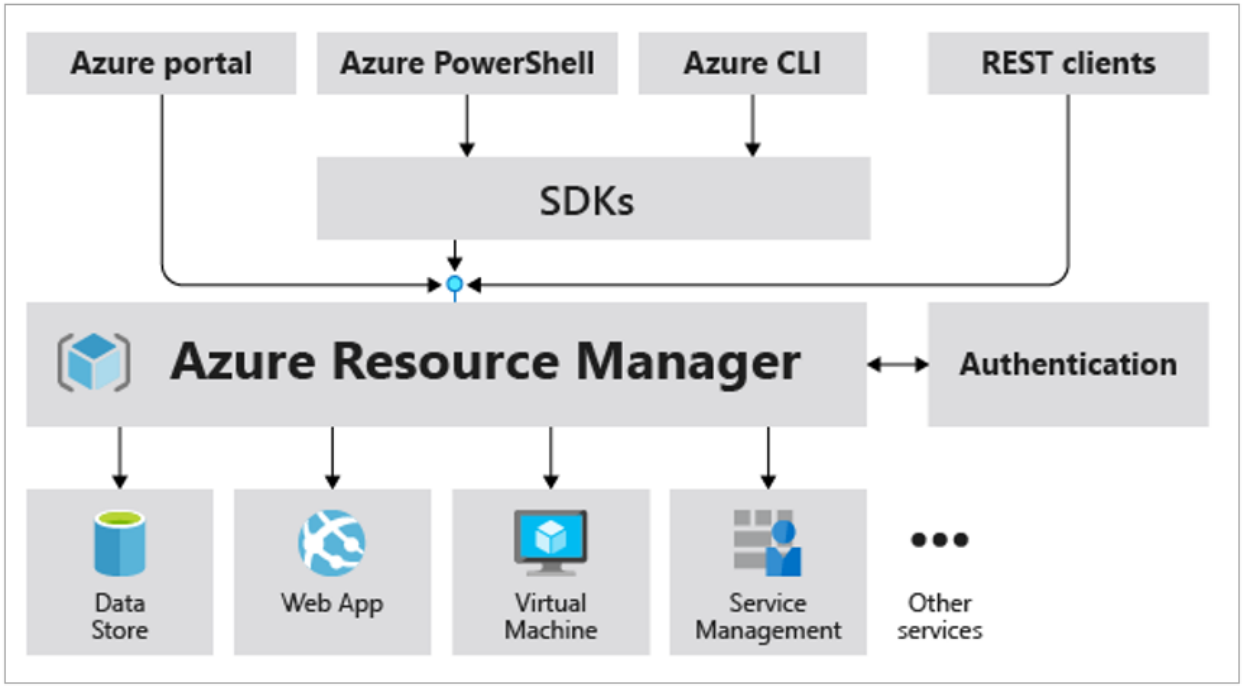

Figure 3.1. Azure Resource Manager Diagram [3]

3.1.4. Storage

Azure offers a plethora of storage options. There are SQL databases, Blob Storage, Data Lake etc. This choice ultimately comes down to the type of data the user is expecting, volume of data, frequency in which it will be accessed, and what they plan to do with it. The user can always create another storage account of different type and transfer data if they desire. For this study Blob storage will be used locally as well as in the cloud.

Blob storage is designed to handle unstructured data such as videos, images, audio, telemetry, csv, and Javascript Object Notation(JSON). Justification for using this type of storage is that it works very well in IIoT scenarios. It is also the only module-based storage option for Arm32 CPU architectures that are running on devices such as a Raspberry Pi. Theses blob modules enable data to be stored locally thus moving the processing as close to the edge as possible, yielding enhanced analytics as mentioned previously. This prevents large amounts of data from having to be sent to the cloud thus eliminating the latency loop. Another benefit of this locally stored data is if connectivity is lost it will hold onto telemetry until the connection is restored, preventing gaps in output history. This is the occasionally offline scenario that many remote assets encounter.

These blob container modules come with additional functionality such as "deviceToCloudUpload" and "deviceAutoDelete." With this functionality the user can direct blob contents to a designated Azure account, trigger deletion of blobs once their contents have been pushed to the cloud, choose the order in which blob contents are uploaded(newest or oldest first), and set a time to delete if desired.

3.1.5. Stream Analytics

Stream Analytics is a real-time data ingestion tool that can be applied to an IIoT Device's data input stream before it is routed to another sink. This application can be implemented through Azure IoT Portal, Visual Studio, or Visual Studio Code. It is capable of processing incoming data with use of its SQL-based query language. Data can be filtered, sorted, aggregated, as well as joined with other data inputs. When coupled with Azure Functions these data influxes can be used to initialize workflows such as alerting the user of failures or abnormal conditions, as well as acting on these data by sending control commands to the appropriate equipment. In terms of reliability Microsoft says the following, "Azure Stream Analytics has built-in recovery capabilities in case the delivery of an event fails. Stream Analytics also provides built-in checkpoints to maintain the state of your job and provides repeatable results" [42].

3.1.6. Azure IoT Edge

Azure has developed an endpoint to endpoint solution designed specifically for the problems arising from pushing more responsibility to the edge. These endpoint to endpoint connections are designed to ensure that devices can be modified, maintained, and upgraded post deployment. This is important because systems are created for security and operational procedures which exist at the time, not the future. Therefore, to have a truly growing and adaptive system it needs the ability to have vulnerabilities patched, upgrades performed, and tasks adjusted offsite with minimal disruption of the daily workflow.

IoT Edge comes with a security manager service that "ensures devices have the right software and only authorized edge devices can communicate with each other, as well

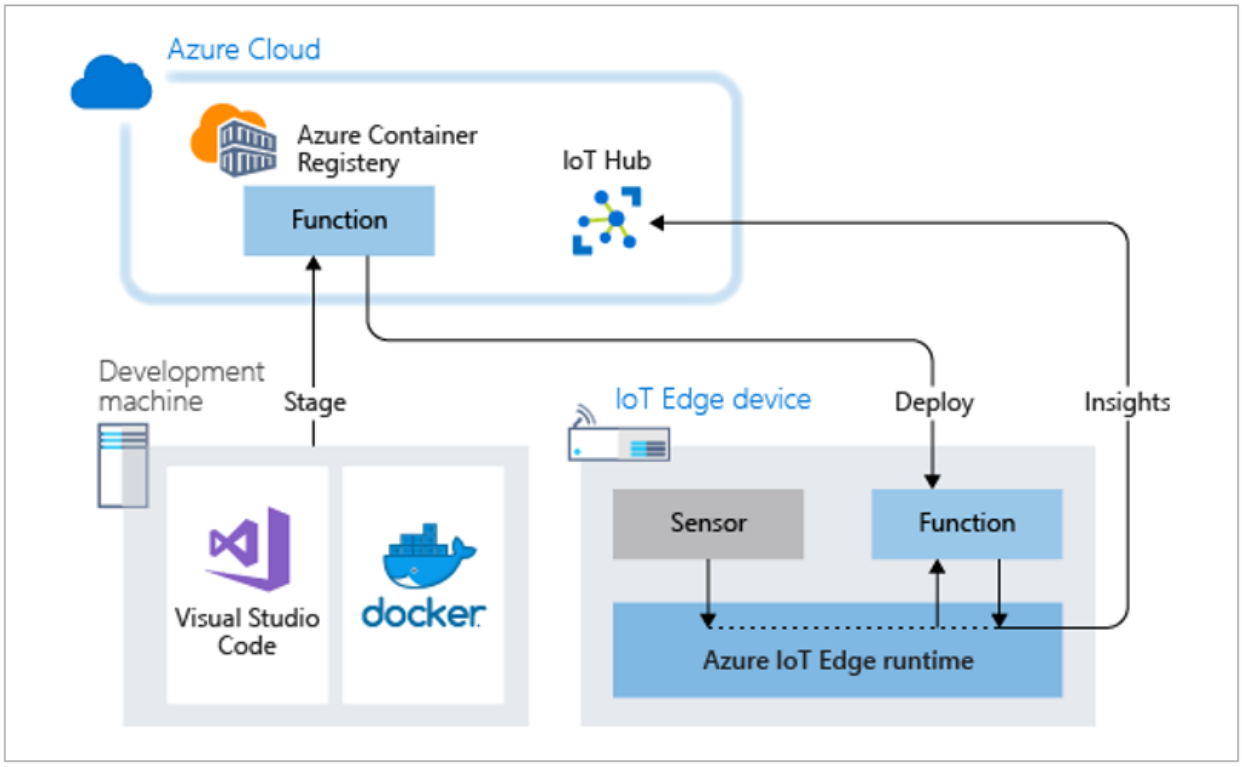

Figure 3.2. Azure IoT Edge Architecture [24]

as providing end to end threat protection and security posture management" [45]. This design gives the user a system of checks and balances which help to mitigate malicious intent, as well as create a flexible system that isn't limited to one of the various types of operating systems, programming languages, or communication protocols that will be found on the market. The implementation of IoT Edge lays the foundation from which autonomous solutions can be built. IoT Edge is aimed at intelligence and interoperability, unifying all of the user's assets into one cohesive system, yielding valuable data insights pertaining to business and industrial operations as a whole.

Containerization is a vital component in Azure IoT Edge, it functions by virtualizing small runtime execution environments, without this feature the device would lack the very core of what makes IoT Edge beneficial. Containers, simply put, are bare bone dis-

tributions of a virtual machine. The containers are made up of only the bits and pieces needed from a full virtual machine image. This process of virtualization and isolation is performed to eliminate many hurdles a developer may encounter. These hurdles are all too common in the development world, an application designed and used on one operating system or computer fails on another. Applications, whether they are designed by the user, Microsoft, or third party, require various libraries and dependencies. To build and launch these applications with little to no issues, they need to be programmed without the worry of what operating system they will eventually be deployed on. With the implementation of containers, consistent runtime executions can be obtained and done so with less storage volume when compared to virtual machines. "Virtual machine images are regularly 30 GB or more, while a container can often be below 100 MB" [24].

To utilize containerization the IoT system needs both a resource for interacting with containers as well as the containers themselves. Azure's IoT Edge devices have a custom runtime, properly coined "IoT Edge Runtime," that is tasked with management. This runtime is based on Moby, which is the foundational technology that Docker is using. This is convenient because it renders Docker images compatible with Azure IoT solutions if needed. The containers themselves start out as virtual images that are comprised of all the application's operational instructions, along with job specific executable code known as modules. Once the IoT solution has been built and pushed, the images have been packed up(contained) and reside within the user's container registry.

Currently, Azure IoT Edge containerization is setup to create Windows and Linux based containers, no others. Linux based containers can be deployed on 32 bit and 64 bit CPU architecture, allowing the user to include microprocessors such as a Raspberry Pi.

In the previous "Containers" section modules was mentioned. Modules possess designated qualities a user will need while applying an IoT Edge solution. They are stored within the functional containers and are accessible to the IoT Edge Runtime. These modules come with a Module Identity and a Module Twin. The Module Identity marks which variation of module instance is being used in case of redeployment, while the Module Twin stores module configuration information pertaining to that specific module instance. "IoT Edge modules are containers that run Azure services, third-party services, or your own code" [45]. They are simply executable code with all required dependencies to run locally.

Generic modules can be added via the Azure portal, however, for custom module builds a more versatile development environment such as Visual Studio or Visual Studio Code will be needed. These environments give the user the freedom to develop, debug and execute IoT Edge modules in various programming languages such as C++, C# in Visual Studios, or C++, C#, Java, Node.JS, or Python in Visual Studio Code. At this time Visual Studio Code yields more freedom when developing edge devices and because of this it will be the tool of choice for this build.

The IoT Edge Runtime is what keeps track of all of these modules, making sure they are authenticated, running, communicating, and healthy. When you successfully add an edge device to your IoT Hub and configure the runtime on the physical device, by default, it comes with two pre-installed modules, edgeAgent and edgeHub.

The edgeAgent is tasked with accessing the container registry and "grabbing" the desired modules and delivering them to the target device. It then starts the module(s) and monitors their operation sending status reports back to Azure IoT Hub. The edgeAgent also has the responsibility of starting, stopping, and restarting modules. If the module

continuously stops or throws errors the edgeAgent will "back off" and discontinue start-up attempts. This is done to prevent the wasteful allocation of CPU resources. No other application has the authorization to access and pull modules from container registries.

The edgeHub's main priority is module messaging. "It primarily handles the messaging responsibility between the modules on the edge device and between the edge device and the IoT Hub" [24]. Simply stated, it controls all messages whether they are cloud-to-device(C2D), device-to-cloud(D2C), or machine-to-machine(M2M). Complications arise from communications to and from isolated modules. These modules are unaware of each other's existence. Using a message bus pattern the edgeHub acts as a broker. Published messages are sent to this broker, stored, and then received by authorized consumers. The edgeHub communicates with Azure IoT Hub from within its isolated environment by exposing an Application Programming Interface(API). This is the same API used by the Azure IoT Hub and acts as its proxy, exposing protocol-endpoints for MQTT and AMQP. If connectivity is ever lost between the edge device and IoT Hub, edgeHub takes on the role of IoT Hub, becoming the new central communication hub. This means that it will store all messages from within as well as messages whose origin is from a child or leaf node device.

3.2. Data Processing and Visualizations with Google Colab

Google's Colab is a web-based hosted Jupyter Notebook service. This allows users to execute python code and is well suited for applications in machine learning and data analysis. Through the utilization of this resource the Bench Model's telemetry can be pulled from the Blob container and processed anywhere with an internet connec-

tion(Appendix F). By implementing Google Colab the host machine isn't required to download any of the supporting resources needed to use Jupyter Labs. This is a useful and advantageous tool that will be added to this workflow. This will be demonstrated to show methods of interaction with batch telemetry stored in the cloud.

Chapter 4. Methodology

This chapter serves as a guide to the approach, solution, and overall mindset implemented to address the problem statement. An overview of productivity flows will ensue, micro-level project details will be provided in the designated appendices.

The process of enabling a remotely accessible asset can be difficult. To have a smooth workflow the project in its entirety was broken down into steps, addressing the problem from the ground level, at the edge, and working up towards the cloud.

4.1. Bench Model

Figure 4.1. Bench Model

This study's Use Case is a bench scale model, designed for slug flow mitigation and control [35]. It resides in the confines of a PFT laboratory at LSU, is approximately 4' x

6' x 4', and comes with a liquid pump, air pump, two pressure transducers, flow meter, and an actuator driven "choke" valve.

Through the use of a graphical user interface(GUI), built using MATLAB on a Windows 10 device, virtual gauges and switches were created. These provided interactive displays to read telemetry, override control of physical peripherals, and export data files. With this GUI, the model was equipped to monitor, control, and record experimental data on an attached host machine. It lacked the ability to send telemetry without the guidance and direction of a human operator controlling said processes. The data would be recorded and forwarded to local storage and later processed through Excel. This was a useful setup for the needs at that time, however, to move it into the category of remotely operational adjustments would have to be made. In terms of offsite monitoring this model represented an obsolete system that would yield decision latency, limited communication, and poor intelligence. What was needed for this model was to improve upon its current standing.

As previously discussed, IIoT solutions can be implemented from scratch, directly interfacing available sensors and actuators or it can be integrated with existing infrastructure such as an RTU/PLC. The Bench Model is an example of starting from scratch due to the lack of financial resources required to purchase a field-grade RTU/PLC. This is fine, it's more fun this way.

This solution's initial steps involved looking at what sensors were available. These were comprised of two onboard 0-30 psig pressure transducers and one 0-30 LPM flow meter with 9-24V electrical consumption requirements. The signals generated would be analog(4-20mA) for the pressure transducers and digital(0-5V) for the flow meter. With the foundational necessities understood the next logical step was to move onto the "brain"

of the model. The previously used Arduino Uno microcontroller was still attached and could come in handy but for the desired goal it alone would not suffice and was set aside.

4.1.1. Raspberry Pi 3 B+

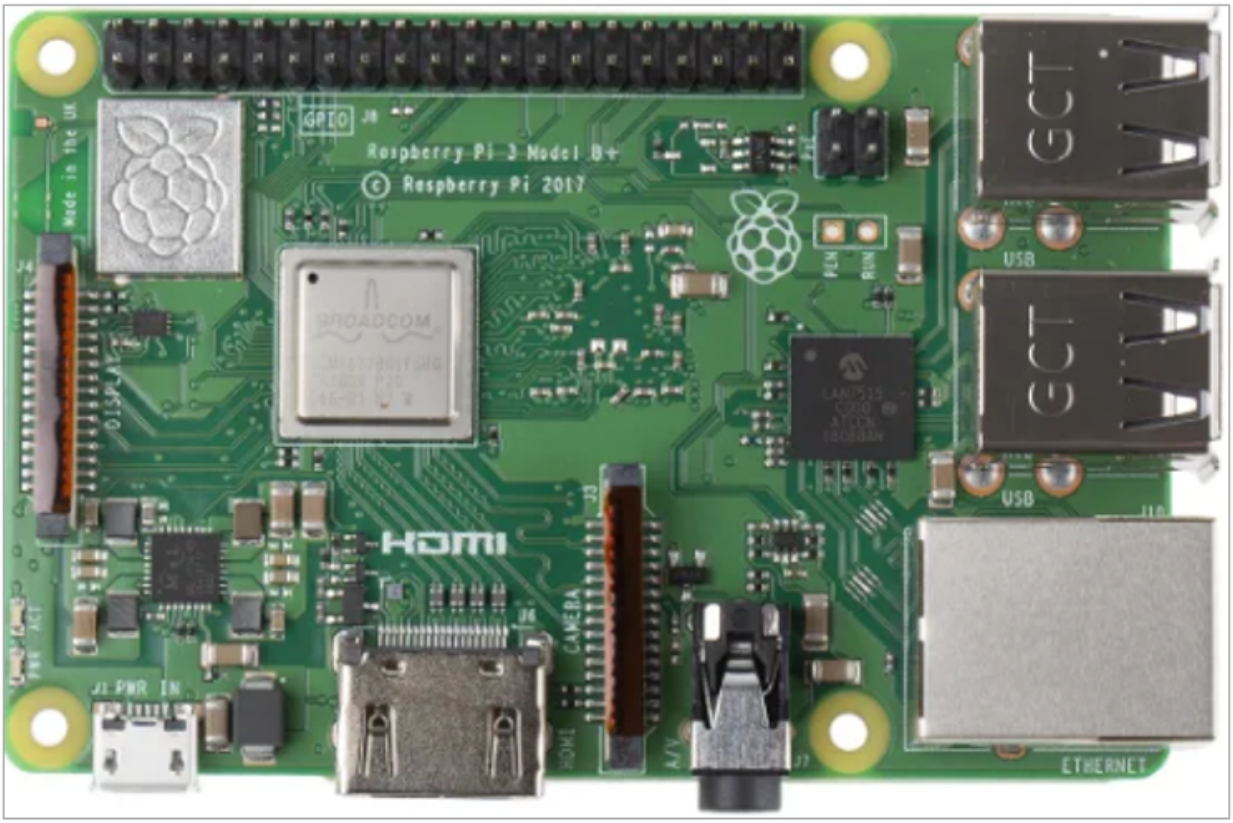

Figure 4.2. Raspberry Pi 3 B+

An edge device must be created on a device with its own operating system. A microcontroller such as an Arduino with a Network Interface Controller(NIC) can be used as a standard IoT device but lacks the functionality required of an edge device. The edge device must have the capacity to "think" for itself, with the addition of the IoT Edge Runtime, and perform multiple operations at once. For the execution of aforementioned tasks the Raspberry Pi 3 B+ was chosen. This Single Board computer's specs are listed in Table 4.1 below.

Table 4.1. Raspberry Pi 3 B+ Design Specifications

SOC	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC
CPU	1.4GHz 64-bit quad-core ARM Cortex-A53 CPU
RAM	1GB LPDDR2 SDRAM
WIFI	Dual-band 802.11ac wireless LAN (2.4GHz and 5GHz)
Video	VideoCore IV 3D. Full-size HDMI
Audio	Yes
USB	2.0 with 4 ports
GPIO	40-pin
Power	5V/2.5A DC power input
Supported OS	Linux and Unix

This is a versatile piece of equipment that is more than capable of performing the scope of work for this project. At a price of only $35.00 USD, it is a great addition to our inventory and will quickly earn its keep. The operating system(OS) it will be running is Raspbian Buster(4.19.97). Next in the workflow were the Raspberry Pi peripherals.

4.1.2. Widgetlords Expansion Boards

The Bench Model's sensors needed to be connected to the Raspberry Pi which isn't a problem if the sensors were all Digital. This model of Raspberry Pi comes with a 40-pin General Purpose Input/Output(GPIO) header consisting of power, ground, Digital, PWM, SPI, I2C, and serial. As you may have noticed Analog is not on listed as an option. The existence of the two analog signals supplied by the pressure transducers required an IO expansion board to be integrated with the Raspberry Pi.

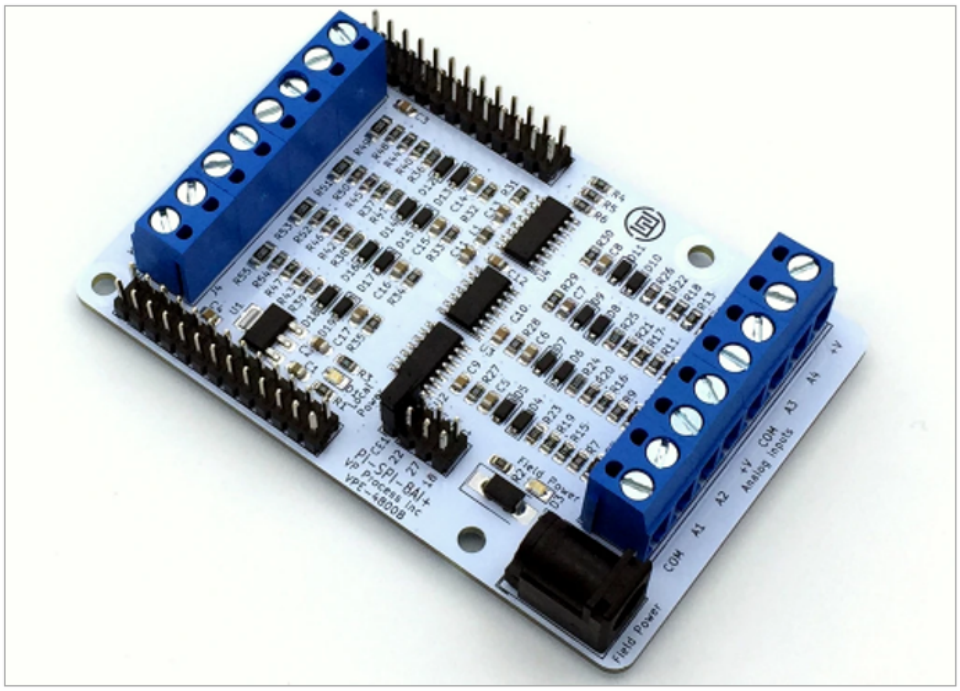

Figure 4.3. Widgetlords Analog Expansion Board

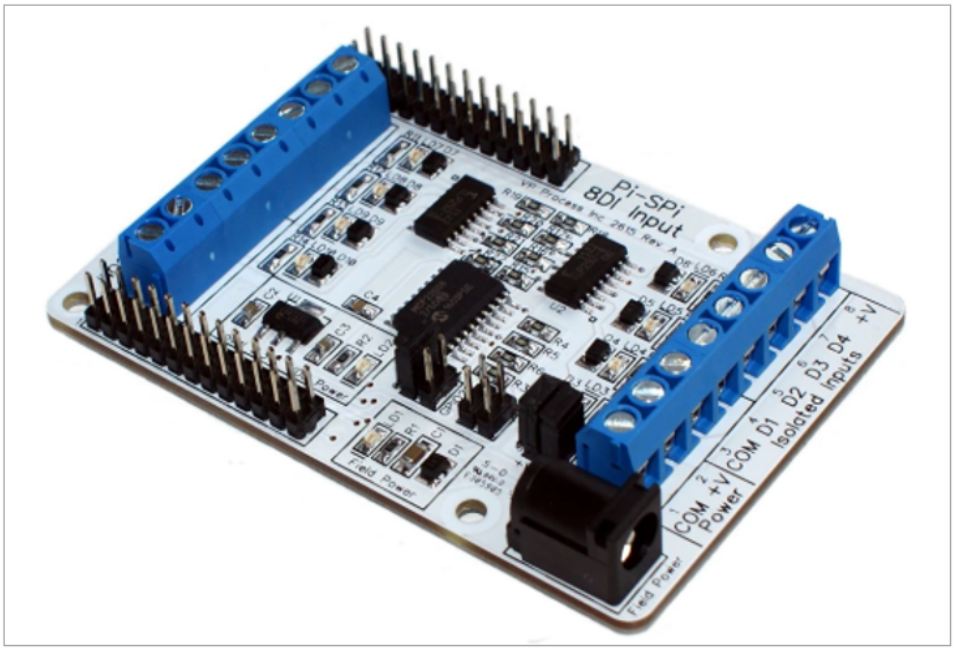

Figure 4.4. Widgetlords Digital Expansion Board

These expansion boards utilize a daisy-chain method that initially connects to the Raspberry Pi's GPIO pins via ribbon cable. Due to the design of this connection all 40 GPIO pins are used therefore all required inputs and outputs must be accounted for by adding the appropriate board to this chain. An Analog and Digital board was purchased for $25.00 each. Both boards came with eight input ports, more than enough, as well as a 20V power supply. Once connected, the Raspberry Pi was daisy-chained to both boards and capable of reading analog and digital inputs.

4.1.3. Arduino Uno

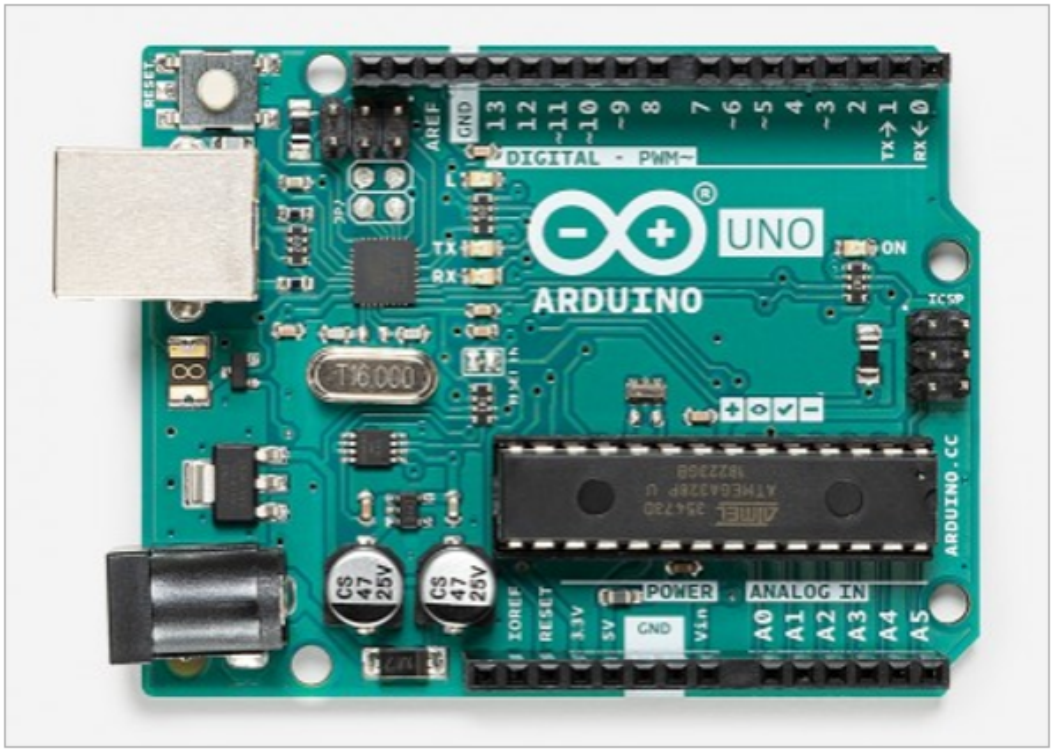

Figure 4.5. Arduino Microcontroller

Another option readily available for reading the Bench Model's sensors is the integration of a microcontroller, specifically an Arduino Uno. This cheap and reliable little board comes with analog and digital read/write capabilities and is less invasive to the

Raspberry Pi. To elaborate, the Arduino can be connected to the Raspberry Pi via USB port, unlike the Widgetlords expansion boards, leaving all of the Raspberry Pi's GPIO pins available for additional functionality.

4.1.4. Intelligent Edge Solution Walk-through

After the Bench Model's hardware was up and running the real work began. The system's brain needed to be transformed into an intelligent edge device. There are far more processes involved in this step than the previous so once again it was broken down into segments and slowly chipped away at to accomplish set goals.

An Azure account was created and populated with an IoT Hub, Container Registry, and Storage Account. Within Azure IoT Hub an IoT Edge device and standard IoT device were added; these act as virtual place holders for the physical devices. The virtual place holders or "Device Twins" provide the necessary configuration and credentials needed for the IoT Hub to authenticate and communicate with the physical twins.

The Raspberry Pi was given a fresh Raspbian Buster OS install(Appendix A) and transformed into an edge device with installation of Azure's IoT Edge Runtime(Appendix B). Once the Raspberry Pi was operating with the desired IoT settings it was configured to communicate with the Widgetlords expansion boards(Appendix C) and Arduino Uno(Appendix G). After following the procedures located in the listed appendices the Bench Model was well on its way to becoming operational.

To free up the GPIO pins on the Raspberry Pi the plan changed from the Widgetlords expansion board to the Arduino Uno. This Arduino needed a script uploaded to it(Appendix G) that measures the designated sensor pins, translates the signal, and for-

mats it into comma separated values. When that was done the Raspberry Pi required the ability to pull those values from the Arduino through one of its USB ports. Now the tricky part begins.

The IoT Edge Runtime is an isolated environment comprised of modules. These modules are unaware of any other modules or processes running outside of this environment and therefore communicating with external sources such as USB ports isn't simply plug and play. There are two ways to accomplish communication: give a designated module elevated permissions to access the host computer's external ports and read the data, or create a leaf node/IoT Edge transparent gateway relationship that enables external data to be pushed into the isolated runtime environment. Both of these options work but a user shouldn't get into the habit of issuing elevated rights when possible, so this study's solution went with the leaf node/transparent gateway approach. There is flexibility with this method, the application to be executed can be located on a separate device or on the same IoT Edge device that is acting as a gateway. Either way a client/server trust relationship will have to be established using a Certificate Authority(CA) [10].

From source(Arduino) to sink(cloud storage) there is a data flow path that must be established. The IoT device requires a Software Development Kit(SDK), these templates are offered by Microsoft and can be altered to fit the needs of the project. The IoT Device will be utilizing a Python SDK. This template was modified with a "serial read" feature that pulls data from the Arduino through the USB port. All SDKs, whether they are written in C, .NET, Java, Python, or Node.js, allow for the establishment of communication with the edgeHub using device credentials and CA certs. This Python code(Appendix D) is stored and executed on the Raspberry Pi.

As explained earlier, the edgeHub module directs all message traffic on the edge device's runtime. When communication was successfully established with the edgeHub, telemetry was permitted to travel inside this environment, enabling access to all of Azure IoT Edge services provided through deployed modules. Now that the external telemetry is ingested into the runtime environment it needs to be routed through desired modules.

A local blob storage module was created [11] that will enable storage of data on the IoT Edge device. This module comes with configuration options that dictate how these data is treated. It can be instructed to delete after a desired time interval, upload to a linked cloud storage account at a desired time interval, and delete after upload. Routing isn't necessary with the blob module. The edgeHub automatically pushes incoming data directly into the blob.

A formatting module was written in C# that consumes the csv telemetry from edgeHub and formats it into a JSON object. This is a message that consists of key/value pairs and is understood by other deployed modules. This module then outputs these messages to the next module in the job scope.

A Stream Analytics job was created [4] in the Azure portal and a Stream Analytics module, built off of the job's specifications, was pushed to the IoT Edge device. This module will ingest the telemetry from the formatting module, process it, and push it to the desired sinks.

Chapter 5. Results and Discussion

5.1. Results

The Bench Model IIoT adaptation is now complete. The once limited system has been optimized to fit the goals of this project. Previously isolated, with no remote monitoring or intelligent capabilities, the Bench Model can now receive, process, store, and forward data autonomously.

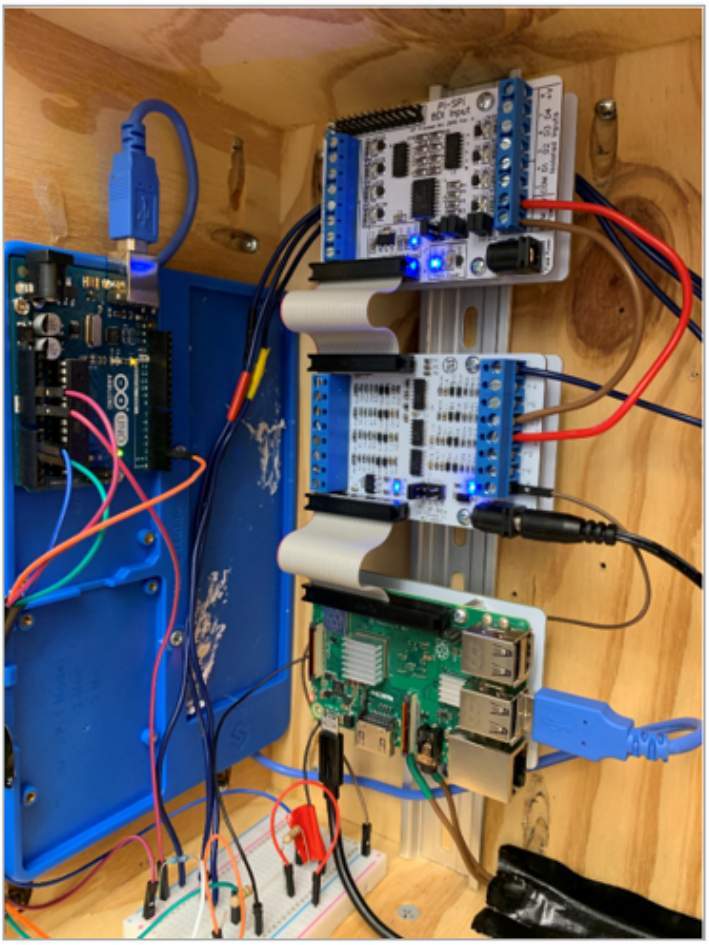

Figure 5.1. Final Edge Device Hardware

Initiating the solution begins with the Raspberry Pi, the terminal is accessed via SSH and the directory in which the Python SDK resides is entered. Once executed several things happen in rapid succession: the connection string is validated, root CA cert is

verified establishing trust between client(leaf node) and server(transparent gateway), sensor data is read through USB, said data is routed into a formatting module that builds a JSON message, JSON message is routed into the stream analytics module, aggregated values are pushed to cloud storage.

The Stream Analytics job was first run without aggregation applied and then with. The first scenario was used for higher telemetry volume that will be used in a phase experiment on the Bench Model. The second scenario was used to demonstrate the batch telemetry potential of the Stream Analytics module. With no processing applied initially, a straight path query can be seen along with an input preview in Figure 5.2.

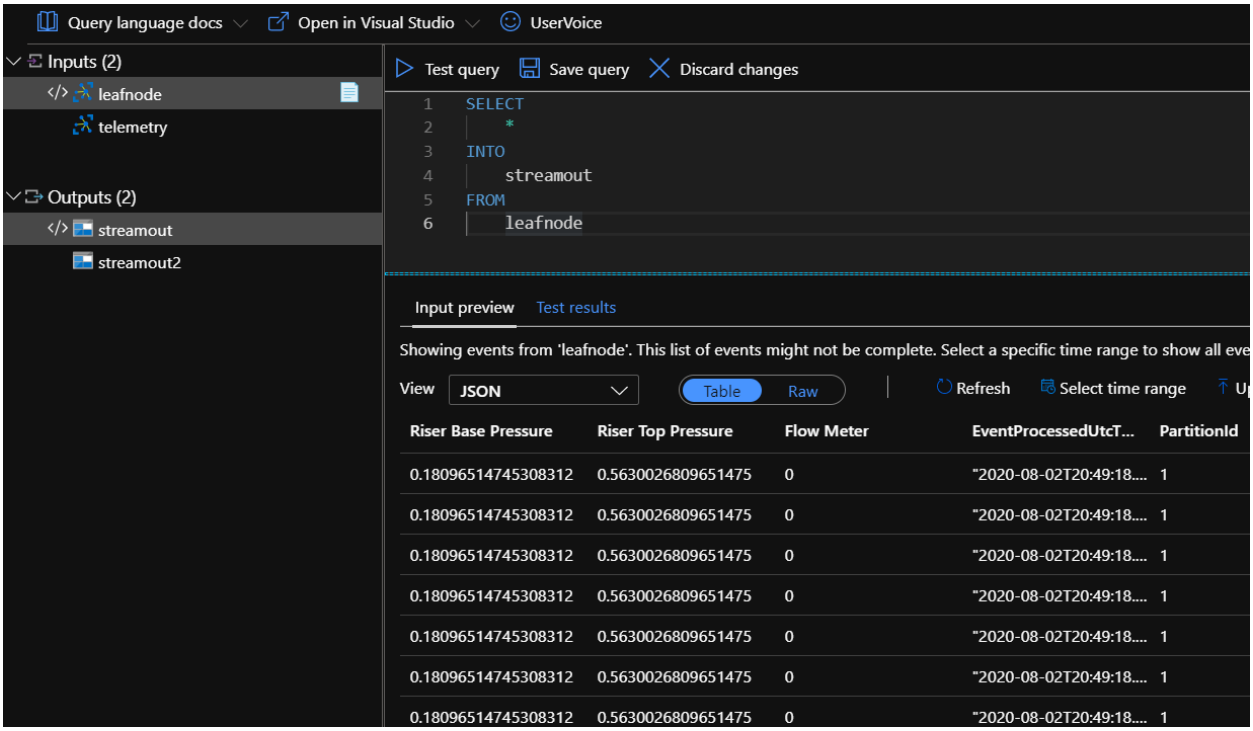

Figure 5.2. Stream Analytics Telemetry View

To access the stored data externally for demonstrative purposes Google CoLab was used. By following the steps in Appendix F a designated blob can be accessed and the data within pulled to a Jupyter Notebook where data exploration can be performed.

The following figures show some examples of this data exploration being implemented. In Figure 5.3, the commands head, describe, and info are demonstrated. Together, they produce a sneak peek at the first five values, display the total count of inputs, unique values within, frequency in which those values appear, range index, Non-Null count, and data types. Figure 5.4 is the plotting of the three categories of telemetry, Riser Base Pressure, Riser Top Pressure, and Flow Rate. Figure 5.5 is each individual category plotted separately for clarity.

```
df.head()
```

	{"Riser Base Pressure":0.14	Riser Top Pressure:0.5	Flow Meter:0.0}
0	{"Riser Base Pressure":0.26	Riser Top Pressure:0.5	Flow Meter:0.0}
1	{"Riser Base Pressure":0.36	Riser Top Pressure:0.5	Flow Meter:0.0}
2	{"Riser Base Pressure":0.46	Riser Top Pressure:0.52	Flow Meter:0.0}
3	{"Riser Base Pressure":0.56	Riser Top Pressure:0.52	Flow Meter:0.0}
4	{"Riser Base Pressure":0.66	Riser Top Pressure:0.52	Flow Meter:0.0}

```
df.describe()
```

	{"Riser Base Pressure":0.14	Riser Top Pressure:0.5	Flow Meter:0.0}
count	103	103	103
unique	37	7	45
top	{"Riser Base Pressure":0.08	Riser Top Pressure:0.54	Flow Meter:0.0}
freq	11	39	32

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103 entries, 0 to 102
Data columns (total 3 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   {"Riser Base Pressure":0.14  103 non-null    object
 1   Riser Top Pressure:0.5       103 non-null    object
 2   Flow Meter:0.0}              103 non-null    object
dtypes: object(3)
memory usage: 2.5+ KB
```

Figure 5.3. "Head," "Info," and "Describe" Commands

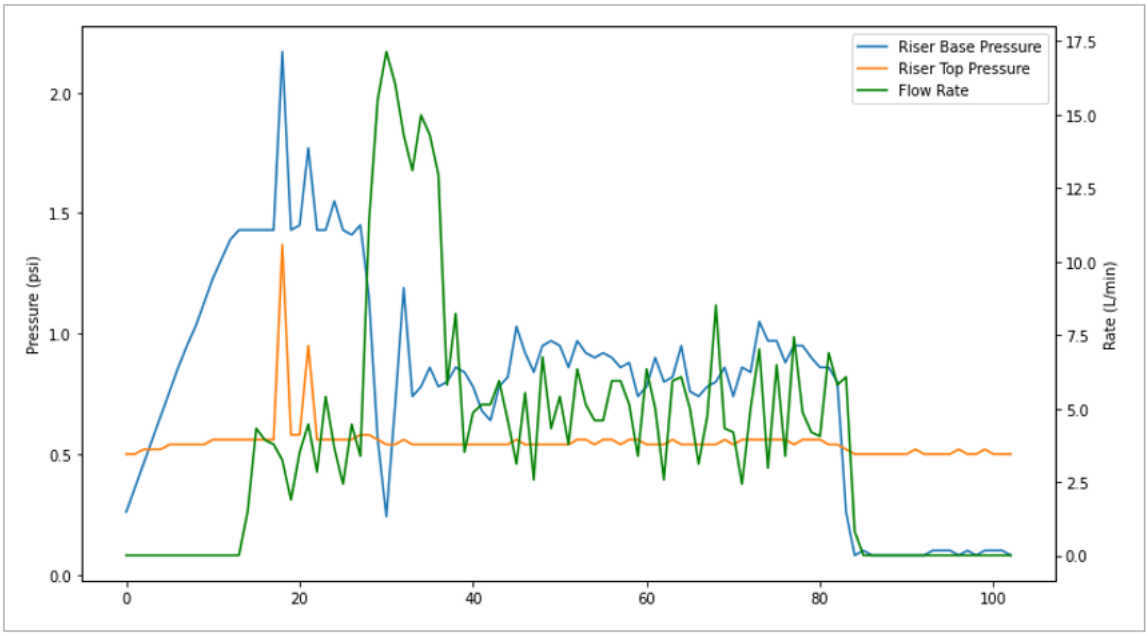

Figure 5.4. Telemetry Visualization

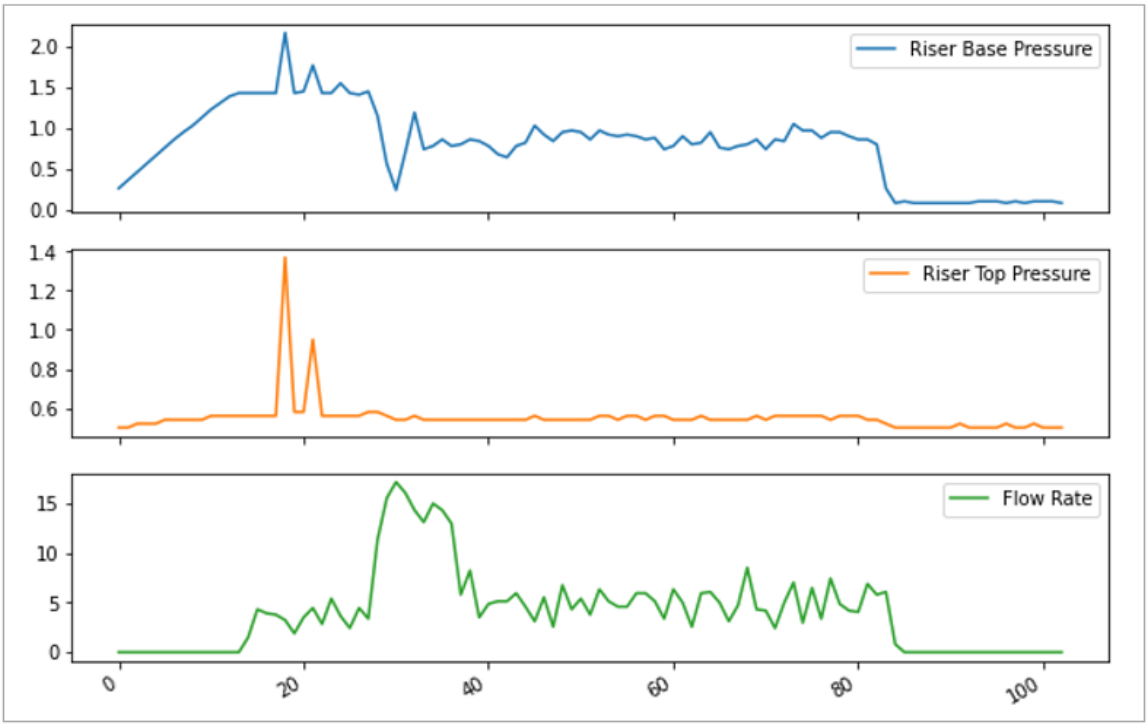

Figure 5.5. Telemetry Visualized Separately

Next, an aggregating query was written(Figure 5.6) in the analytics module that organizes by deviceID, takes the average of each sensor input over a 30 second duration, timestamps the results, and pushes these values to the cloud.

Test query Save query Discard changes

```
1   SELECT
2       deviceID,
3       AVG(riserbasepressure) as RiserBasePressure,
4       AVG(risertoppressure) as RiserTopPressure,
5       AVG(flowrate) as FlowRate,
6       System.Timestamp as eventdatetime
7   INTO cloudblob
8   FROM inputfromformatter
9   WHERE riserbasepressure IS NOT NULL
10  GROUP BY
11      deviceID,
12      TumblingWindow(s, 30)
13
```

Figure 5.6. Aggregation Query In Stream Analytics


```
{"deviceID":"BenchModel","RiserBasePressure":1.65,"RiserTopPressure":0.21,"FlowRate":1.12,"eventdatetime":"2020-10-11T23:34:30.0000000Z"}
{"deviceID":"BenchModel","RiserBasePressure":1.71,"RiserTopPressure":0.24,"FlowRate":1.23,"eventdatetime":"2020-10-11T23:35:00.0000000Z"}
{"deviceID":"BenchModel","RiserBasePressure":1.68,"RiserTopPressure":0.26,"FlowRate":1.17,"eventdatetime":"2020-10-11T23:35:30.0000000Z"}
{"deviceID":"BenchModel","RiserBasePressure":1.73,"RiserTopPressure":0.23,"FlowRate":1.19,"eventdatetime":"2020-10-11T23:36:00.0000000Z"}
{"deviceID":"BenchModel","RiserBasePressure":1.73,"RiserTopPressure":0.21,"FlowRate":1.14,"eventdatetime":"2020-10-11T23:36:30.0000000Z"}
{"deviceID":"BenchModel","RiserBasePressure":1.74,"RiserTopPressure":0.23,"FlowRate":1.21,"eventdatetime":"2020-10-11T23:37:00.0000000Z"}
{"deviceID":"BenchModel","RiserBasePressure":1.73,"RiserTopPressure":0.24,"FlowRate":1.22,"eventdatetime":"2020-10-11T23:37:30.0000000Z"}
```


Figure 5.7. Aggregated Telemetry In Cloud Storage

Referencing Figure 2.3, it can be seen that there are multiple stages involved for a typical SCADA solution involving a brick and mortar data center. Streamlining just one of these stages while maintaining the same performance is a notable improvement. The Bench Model's IoT Edge solution has done just that. When compared to the typical SCADA architecture it can be seen that the IoT Edge's decentralized architecture is noticeably smaller.

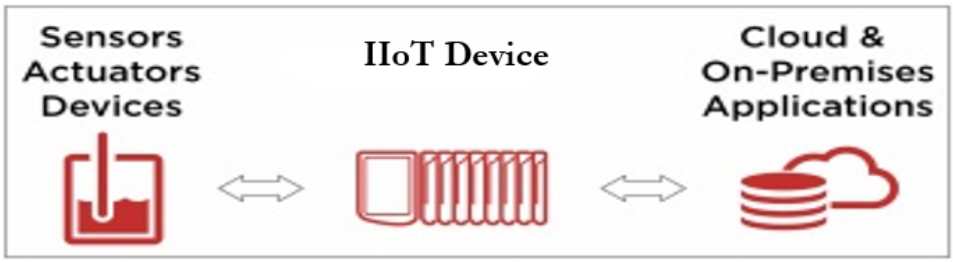

Figure 5.8. IoT Edge Solution [28]

5.2. Discussion

An objective of this study was to determine if an edge computing solution using Azure IoT Edge could overcome challenges encountered by remote hydrocarbon assets. These consisted of decision latency, network stress, and intermittent connectivity. This study also aimed to demonstrate a solution that not only remedies these issues but does so with a reduction in infrastructure. The results have been provided but require discussion.

The telemetry displayed in Figure 5.5 was recorded from a flow phase experiment. The Bench Model started off with an empty riser and was slowly filled. This can be seen on the plot where Riser Base Pressure reaches a maximum value of approximately 2 psi as a result of hydrostatic head. The spikes in Riser Top Pressure is indicative of the increase in back pressure caused by momentary closure of the choke valve. Naturally, as water flow reached the flow meter the flow rate increases and decreases according to observed production. The flow phase change from single-phase to two-phase, specifically slug flow, can be observed on the plot where Riser Base Pressure drops significantly due to loss of hydrostatic head, and the flow rate spikes due to the sudden surge of production from liquid slugs. The plotted telemetry shows us that the model serves a purpose. These data aren't ambiguous nor arbitrary, the results on the plot behave as they should when observing the multiphase flow regime, slug flow. This demonstrates that the Bench Model can successfully and autonomously transfer telemetry to a designated cloud-based location.

Utilizing Azure Stream Analytics, data was processed locally, performing an aggregation of ingested telemetry and outputting batch telemetry. There are a couple of benefits reaped by doing this on the edge itself. The creation of batch telemetry drastically

reduces the volume of data transferred over limited networks. This relieves stress induced by standard streaming solutions where there is constant transmission of data.

Another benefit, not explicitly shown are the enhanced analytics this edge architecture affords. Using more advanced queries enables the edge device to monitor for abnormal operating conditions and take action with the implementation of a control module deployed to the device. Integrating these features would yield decision making locally, removing the latency inherent to standard IIoT and SCADA architectures.

In terms of intermittent connectivity, there weren't results to accurately depict this. However, if we recall, the edgeHub serves many purposes, one of which is to assume the role of the IoT Hub if connection is ever lost. In the event of this offline scenario the edgeHub will store all messages originating from itself, as well as all messages coming from leaf node devices that have an established child/parent relationship. When connectivity is restored the edgeHub transmits the stored data to the cloud. This feature is native to Azure IoT Edge, the only configuration needed is the child/parent relationship, completed through the edge device settings, and the "timeToLive" setting located in the edgeHub's module twin. This setting specifies the duration of data storage, in seconds, which by default is 7200 or 2 hours.

Figure 5.8 shows the reduction of infrastructure an IIoT edge solution provides. As previously discussed, it does this by streamlining traditional architecture. An HMI/PC is now any PC authorized to access the IoT Hub instance, protocol translation and firewall services are performed on the edge device itself.

What can't be determined from the results is important criteria such as scalability, reliability, and robustness. This is disappointing but completely understandable. These

capabilities can only be proven if they are performed. To truly test scalability a multitude of devices would have to be connected to the same IIoT solution and there simply wasn't enough time or money. The reliability of the Bench Model's solution can not be guaranteed for it hasn't experienced rigorous data collection and processing. The demonstrated solution has only faced small consistent telemetry from a single device, not multiple devices encountering random power outages and lost connections. Although the Bench Model has performed very well with hobbyist equipment it raises questions pertaining to its robustness, if any. Using hardware such as a Raspberry Pi, Arduino, and Widgetlords expansion boards doesn't come without some worry as to whether or not they can be trusted for production level workloads. These are versatile tools but extensive tests would have to be run before a user could confidently implement them into the field. The results are promising but there are questions pertaining to production-level readiness that have been left unanswered.

Chapter 6. Conclusion and Future Work

6.1. Conclusion

This study aimed to answer whether or not Azure IoT Edge services could offer a remote monitoring solution capable of mitigating challenges experienced by remote hydrocarbon assets all while reducing excess infrastructure. The application of Azure IoT services through readily available hardware has shown strong evidence in many categories that it can in fact perform edge computing duties with a reduced solution footprint. It achieves this through a decentralized architecture, allowing edge device accessible, cloud-based resources. This solution has successfully performed data monitoring, translation, storage, processing, and transmission from a remote location. Therefore, it is in the conclusion of this study that the methods proposed are practical and reside within the realm of efficacy pertaining to the remote monitoring of hydrocarbon wells. The methods proposed have not however, given evidence that they are production ready, capable of performing as strongly and reliably as standard IIoT and SCADA.

Choosing to utilize cheap and readily available hardware was one of purpose and personal challenge. Of course, most IIoT solutions are made up of small devices but they are usually designed to be IIoT specific and therefore warrant a higher price tag. In this study the goal was mainly to demonstrate Azure's powerful IoT services and show how they could be integrated or used as standalone solutions that benefit the oil and gas sector. The hardware used was merely a medium in which these benefits of an IoT Edge solution could flow. However, if these services could be demonstrated while simultaneously pushing the limits of the aforementioned hobbyist equipment, why not? This simply added

flavor to an already interesting and challenging endeavor. In fact, a large percentage of challenges faced throughout this study were a direct result of the hardware chosen, not the software or other services provided by Azure. Although this was a risky decision these hardware induced complications helped in gaining a more in-depth understanding of the solution architecture as a whole.

It is obvious that the Bench Model is dwarfed in comparison to a producing hydrocarbon well, not only in terms of size and functionality, but associated hazards as well. The use case is located in a relatively controlled environment when compared to a producing well and only has to bear a fraction of the exposure that comes with this. This is clearly understood, the methodology was based on the premise that to develop a solution for parameters such as these requires an experimental testing progression. The Bench Model was a perfect small scale application that provides evidence of viability, not a production level solution. The next phase of testing would be the High Pressure Convection Apparatus discussed in the following Future Work section.

6.2. Future Work

6.2.1. Bench Model

The Bench Model was originally designed for the purposes of slug flow mitigation through the use of PID control. With sensors measuring pressures and flow rate, an autonomous choke valve successfully regulated back pressure on the system, thus mitigating slug flow. To expand on this study's research and the model's original research, a few additions are in order. There are no longer control applications available on the Bench Model. Being able to control the pumps and choke valve remotely would offer further understanding into IoT Edge driven monitoring and control solutions. This would also allow for the testing of more advanced control algorithms.

Machine learning algorithms could be developed to help analyze this phenomenon by discovering correlations between pressure, flow rate, and the formation of slugs within the riser. These correlations could then be harnessed for use in a classification module or even a predictive analytics module capable of identifying the environment in which these slugs form. Once this is understood by the system it could potentially optimize choke valve position via autonomous control commands, mitigating slug flow and yielding the most production given a specific condition. This would be quite the undertaking but extremely beneficial. If successful, the IoT Edge solution would be another step closer to production level status.

6.2.2. High Pressure Convection Apparatus

A second Use Case consists of a High Pressure Convection Apparatus. This apparatus was designed by Dr. Chen's research group in the Petroleum Engineering Department and was built by RavenFlo LLC. Its intended purpose is for the study of methane convection between contaminated and clean oil in the wellbore while drilling. It has the dimensions of 4' x 4' x 12' and is equipped with a flow controller, flow meter, pressure transducers, booster pumps, vacuum pump, Red Lion PLC, and Red Lion HMI.

Figure 6.1. High Pressure Convection Apparatus

At the time of this writing the High Pressure Apparatus is still undergoing preliminary trials before it can be considered fully operational. This apparatus's various peripherals are connected to a PLC and controllable through the use of an HMI or the onsite PC located in the control room. The model of PLC being used has Azure MQTT communication capabilities that enables all of these peripherals to be tagged, monitored, and controlled through Azure IoT Hub(Appendix E). There is an integrated methane detection system that will form a perimeter around the model and methane gas supply. The associated sensors will have their signals routed to the PLC and from there pushed to Azure IoT Hub. A detection function is to be implemented to constantly check for binary result of True/False. If this measurement is ever found to be True the alarm function is triggered and a message will be sent via text to essential personnel.

Making these additions to the High Pressure Convection Apparatus will yield several beneficial results. The unit would be capable of having its area of operation continuously monitored for methane leaks and inform essential personnel if incident occurs. This Azure IoT solution will establish a connection to cloud-based resources, providing storage, processing, and centralized accessibility for everyone on the research team. This unit will have reliable internet and connectivity, however, edge computing can still be implemented for pre-processing of data before it is sent to the cloud, reducing data transfer volumes.

This model represents a more realistic application as it resides outside, it possesses operational hazards, and it is comprised of industrial level equipment. This test would offer further insight into the solution's production-level validity. A tertiary phase is outside of the scope of this study but if performed a low level production asset is preferred.

As mentioned previously there is a need to demonstrate scalability, reliability, and robustness before a more thorough conclusion to the problem statement can be made. This will involve specific performance tests such as large volumes of telemetry from multiple sources, intermittent connectivity, and data processing. These tests coupled with important additions involving long range communication, manual override controls, and machine learning modules will address any concerns raised by this study's demonstrated solution. Although there is more work to be done, this study has provided a solid remote monitoring foundation. This foundation yields not only performance, it allows for future work to be performed on top of it, further expanding its capabilities and research of Azure IoT services.

Appendix A. Raspberry Pi Setup

```
// Install Raspbian Buster on Raspberry Pi

1. Download latest version of "Raspbian Buster" (disk image) onto development machine
2. Download "BalenaEtcher" for Windows 10 (This software will enable the transfer of disk image to micro SD card)
3. Insert 32GB micro SD card and USB micro SD card adapter into development machine
4. Open BalenaEtcher, "select disk image," "select target," then "Flash"
5. Once completed, safely remove micro SD card from adapter
6. Insert micro SD card into Raspberry Pi
7. Connect Raspberry Pi to monitor, keyboard, and mouse to Raspberry Pi
8. Power Up Raspberry Pi
9. Configure Raspberry Pi
```

Figure A.1. Raspbian Buster Installation Steps

Appendix B. Edge Device Setup

```
// Install Azure IoT Edge Runtime on Raspberry Pi

1. curl -L https://aka.ms/moby-engine-armhf-latest -o moby_engine.deb && sudo dpkg -i ./moby_engine.deb
2. curl -L https://aka.ms/moby-cli-armhf-latest -o moby_cli.deb && sudo dpkg -i ./moby_cli.deb
3. sudo apt-get install -f
4. curl -L https://aka.ms/libiothsm-std-linux-armhf-latest -o libiothsm-std.deb && sudo dpkg -i ./libiothsm-std.deb
5. curl -L https://aka.ms/iotedged-linux-armhf-latest -o iotedge.deb && sudo dpkg -i ./iotedge.deb
6. sudo apt-get install -f
7. sudo nano /etc/iotedge/config.yaml   >> Insert targeted device primary connection string (ctrl + x, y, ENTER)
8. sudo systemctl restart iotedge
9. sudo reboot
10. sudo systemctl status iotedge
```

Figure B.1. IoT Edge Runtime Environment Installation Steps

Appendix C. Widgetlords Setup

```
// Setup Widgetlords on Raspberry Pi

1. Open terminal window and use the following commands
2. sudo raspi-config
3. Select "5 Interfacing Options"
4. Select "P6 Serial"
5. Answer no for a serial-accessible login shell
6. Answer yes for enabling serial hardware
7. Select "Finish"
8. Download the latest library release (v2.0.0) from the GitHub releases page:
    https://github.com/widgetlords/libwidgetlords/releases/download/v2.1.0/libwidgetlords_2.1.0_armhf.deb
9. sudo dpkg -i libwidgetlords_2.0.0_armhf.deb
10. sudo nano /boot/config.txt (add the appropriate "dtoverlay=pi-spi" to the end of the file)
11. sudo reboot
```

Figure C.1. Widgetlords Expansion Board Installation Steps

Appendix D. Leaf Node Python Script

```
# Copyright (c) Microsoft. All rights reserved.
# Licensed under the MIT license. See LICENSE file in the project root for
↪  full license information.
import random
import time
import serial
from azure.iot.device import IoTHubDeviceClient, Message
CONNECTION_STRING = "<YourConnectionString>"
ca_cert = "<YourCaCertPathOnDevice>"
certfile = open(ca_cert)
root_ca_cert = certfile.read()

def iothub_client_init():
    # Create an IoT Hub client
    client =
    ↪  IoTHubDeviceClient.create_from_connection_string(CONNECTION_STRING,
    ↪  server_verification_cert=root_ca_cert)
    return client

def iothub_client_telemetry_sample_run():

    try:
        client = iothub_client_init()
        print ( "IoT Hub device sending periodic messages, press Ctrl-C to
        ↪  exit" )
        ser = serial.Serial('/dev/ttyACM0', 9600, timeout=0.5)

        while True:
            time.sleep(5)
            telemetry = ser.readline().decode("UTF-8").rstrip()
            message = Message(telemetry)
            print( "Sending message: {}".format(message) )
            client.send_message(message)
            print ( "Message successfully sent" )

    except KeyboardInterrupt:
        print ( "Telemetry Stopped" )

if __name__ == '__main__':
    print ( "IoT Hub Quickstart #1 - Bench Model" )
    print ( "Press Ctrl-C to exit" )
    iothub_client_telemetry_sample_run()
```

Appendix E. High Pressure Apparatus Setup

```
// Configuring Red Lion PLC to communicate with IoT Hub

1. Add an IoT device in Azure IoT Hub
2. Copy the "Host Name," "Device ID," and "Primary Key" for the newly added Device
3. Open the Crimson Software on host machine
4. In the "Navigation Pane" select "Azure MQTT"
5. Enable Agent needs to be switched to "Yes"
6. Under "MQTT Server" fill in "Host Name 1" and "Client ID" with copied Host Name and Device ID from step 2.
7. Under "Authentication" insert the copied Primary Key
8. Under "Diagnostics" make a tag called "Status"
9. Switch to the "Network" tab
10. "Transport Protocol" should be "TLS," "Server Certificate" is on "Ignore" and "Connect via Port" is set to "8883"
11. Switch to "Tag Data 1" tab
12. Drag all desired Tags to be monitored into the "Contents" pane
13. Save and apply configuration to the PLC
14. Telemetry from the selected Tags is now being sent to IoT Hub
```

Figure E.1. Red Lion PLC IoT Hub Configuration

Appendix F. Google CoLab Blob Access Steps

```
!pip install azure-storage-blob==0.37.1
```

```
from azure.storage.blob import BlockBlobService
import pandas as pd
import tables

STORAGEACCOUNTNAME= "YOUR_STORAGEACCOUNTNAME"

STORAGEACCOUNTKEY= "YOUR_STORAGEACCOUNTKEY"

LOCALFILENAME= "GIVEITANAME"

CONTAINERNAME= "YOUR_CONTAINERNAME"

BLOBNAME= "FILEYOUWANTTOACCESS.JSON"

#download from blob

blob_service=BlockBlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)

blob_service.get_blob_to_path(CONTAINERNAME,BLOBNAME,LOCALFILENAME)
result = blob_service.get_blob_to_text(CONTAINERNAME,BLOBNAME)

# print(result.content)
```

```
df = pd.read_csv('LOCALFILENAME')
df.head()
df.describe()
df.info()
test = df.iloc[:,0].apply(lambda x: str(x).split(':')[1])
test = test.astype('float')
import matplotlib.pyplot as plt
test.plot()
```

Figure F.1. Google CoLab Blob Access Steps

Appendix G. Arduino Uno Script

```
//Arduino Script for analog and digital read
#include <ArduinoJson.h>

int flowPin = 2;    //This is the input pin on the Arduino
float flowRate;     //This is the value we intend to calculate.
volatile int count; //This integer needs to be set as volatile to ensure
                    //it updates correctly during the interrupt process.

int analogPin1 = A1;
int analogPin2 = A2;

int pressure1 = 0;
int pressure2 = 0;

void setup() {
  // put your setup code here, to run once:
  pinMode(flowPin, INPUT);           //Sets the pin as an input
  attachInterrupt(0, Flow, RISING);  //interrupt to run the function
  ↪  "Flow"
  Serial.begin(9600);  //Start Serial
  while(!Serial);
}

void loop() {
  // put your main code here, to run repeatedly:
  count = 0;      // Reset the counter so we start counting from 0 again
  interrupts();   //Enables interrupts on the Arduino
  delay (5000);   //Wait 1 second
  noInterrupts(); //Disable the interrupts on the Arduino

  flowRate = (count * 2.25);  //Take counted pulses, multiply by 2.25mL
  flowRate = flowRate * 60;   //Convert seconds to minutes >> mL / Minute
  flowRate = flowRate / 1000; //Convert mL to Liters >> Liters / Minute

  float pressure1 = analogRead(analogPin1);
  float pressure2 = analogRead(analogPin2);

  //(x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
  pressure1 = (pressure1 * 30) / 1023.0; //Convert 0 - 1023 to 0 - 30 psi
  pressure2 = (pressure2 * 30) / 1023.0;

  Serial.print(pressure1); Serial.print(",");
  Serial.print(pressure2); Serial.print(",");
```

```
  Serial.print(flowRate);
  Serial.println();
}

void Flow()
{
   count++; //Every time this function is called, increment "count" by 1
}
```

Appendix H. Stream Analytics Query

```
-- get aggregated pressure/flowrate every XX seconds
SELECT
    deviceID,
    AVG(riserbasepressure) as RiserBasePressure,
    AVG(risertoppressure) as RiserTopPressure,
    AVG(flowrate) as FlowRate,
    System.Timestamp as eventdatetime
INTO cloudblob
FROM inputfromformatter
WHERE riserbasepressure IS NOT NULL
GROUP BY
    deviceID,
    TumblingWindow(s, 30)
```

Appendix I. Formatting Module Script

```
namespace formatter
{
    using System;
    using System.IO;
    using System.Runtime.InteropServices;
    using System.Runtime.Loader;
    using System.Security.Cryptography.X509Certificates;
    using System.Text;
    using System.Threading;
    using System.Threading.Tasks;
    using Microsoft.Azure.Devices.Client;
    using Microsoft.Azure.Devices.Client.Transport.Mqtt;
    using Newtonsoft.Json;

    class Telemetry
    {
        public string deviceID;
        public float riserbasepressure;
        public float risertoppressure;
        public float flowrate;
    }

    class Program
    {
        static int counter;

        static void Main(string[] args)
        {
            Init().Wait();
            // Wait until the app unloads or is cancelled
            var cts = new CancellationTokenSource();
            AssemblyLoadContext.Default.Unloading += (ctx) => cts.Cancel();
            Console.CancelKeyPress += (sender, cpe) => cts.Cancel();
            WhenCancelled(cts.Token).Wait();
        }
        /// <summary>
        /// Handles cleanup operations when app is cancelled or unloads
        /// </summary>
        public static Task WhenCancelled(CancellationToken
        ↪  cancellationToken)
        {
            var tcs = new TaskCompletionSource<bool>();
```

```
    cancellationToken.Register(s =>
    ↪  ((TaskCompletionSource<bool>)s).SetResult(true), tcs);
    return tcs.Task;
}
/// <summary>
/// Initializes the ModuleClient and sets up the callback to
↪  receive
/// </summary>
static async Task Init()
{
    MqttTransportSettings mqttSetting = new
    ↪  MqttTransportSettings(TransportType.Mqtt_Tcp_Only);
    ITransportSettings[] settings = { mqttSetting };
    // Open a connection to the Edge runtime
    ModuleClient ioTHubModuleClient = await
    ↪  ModuleClient.CreateFromEnvironmentAsync(settings);
    await ioTHubModuleClient.OpenAsync();
    Console.WriteLine("IoT Hub module client initialized.");
    // Register callback to be called when a message is received
    ↪  by the module
    await ioTHubModuleClient.SetInputMessageHandlerAsync("input1",
    ↪  PipeMessage, ioTHubModuleClient);
}
/// <summary>
/// This method is called whenever the module is sent a message
↪  from the EdgeHub.
/// It just pipe the messages without any change.
/// It prints all the incoming messages.
/// </summary>
static async Task<MessageResponse> PipeMessage(Message message,
↪  object userContext)
{
    int counterValue = Interlocked.Increment(ref counter);
    var moduleClient = userContext as ModuleClient;
    if (moduleClient == null)
    {
        throw new InvalidOperationException("UserContext doesn't
        ↪  contain " + "expected values");
    }
    byte[] messageBytes = message.GetBytes();
    string messageString = Encoding.UTF8.GetString(messageBytes);
    Console.WriteLine($"Received message: {counterValue}, Body:
    ↪  [{messageString}]");
```

```
            if (!string.IsNullOrEmpty(messageString))
            {
                string[] parts = messageString.Split(",");
                string devId = "DeviceIdMissing";
                // retrieve the device ID of the sending leaf device and
                ↪  insert into message
                if(!string.IsNullOrEmpty(message.ConnectionDeviceId))
                        devId = message.ConnectionDeviceId;
                // create and populate our new message content
                Telemetry t = new Telemetry();
                t.deviceID = message.ConnectionDeviceId;
                t.riserbasepressure = float.Parse(parts[0]);
                t.risertoppressure = float.Parse(parts[1]);
                t.flowrate = float.Parse(parts[2]);
                // serialize to a string
                string newMessage = JsonConvert.SerializeObject(t);
                // create a new IoT Message object and copy
                // any properties from the original message
                var pipeMessage = new
                ↪  Message(Encoding.ASCII.GetBytes(newMessage));
                foreach (var prop in message.Properties)
                {
                    pipeMessage.Properties.Add(prop.Key, prop.Value);
                }
                // send the data to the edge Hub on a named output (for
                ↪  routing)
                await moduleClient.SendEventAsync("output1", pipeMessage);
                Console.WriteLine($"Converted message sent({counter}):
                ↪  {newMessage}");
            }
            return MessageResponse.Completed;
        }
    }
}
```

Bibliography

[1] "A SANS Analyst Whitepaper An Abbreviated History of Automation & Industrial Controls Systems and Cybersecurity," Tech. Rep., 2014. [Online]. Available: http://powerplantmen.files.wordpress.com/2013/04/power-plant-control-room.jpg.

[2] M. A. Adegboye, W.-k. Fung, and A. Karnik, "Recent Advances in Pipeline Monitoring and Oil Leakage Detection Technologies : Principles," 2019.

[3] *Azure Resource Manager overview - Azure Resource Manager — Microsoft Docs.* [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview (visited on 07/25/2020).

[4] *Azure Stream Analytics on IoT Edge — Microsoft Docs.* [Online]. Available: https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-edge (visited on 10/27/2020).

[5] N. Basavanthappa, "Microsoft Azure IoT Reference Architecture," p. 69, 2018. DOI: .. [Online]. Available: http://download.microsoft.com/download/A/4/D/A4DAD253-BC21-41D3-B9D9-87D2AE6F0719/Microsoft%7B%5C_%7DAzure%7B%5C_%7DIoT%7B%5C_%7DReference%7B%5C_%7DArchitecture.pdf.

[6] "BEYOND DIGITIZATION: THE CONVERGENCE OF BIG DATA, ANALYTICS AND INTELLIGENT SYSTEMS IN OIL & GAS," Tech. Rep., 2015.

[7] M. S. Charles Alexander, "Fundamentals of Electric Circuits," *McGraw-Hill*, 2004.

[8] *Create custom Python module - Azure IoT Edge — Microsoft Docs.* [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/tutorial-python-module (visited on 07/16/2020).

[9] *Create transparent gateway device - Azure IoT Edge — Microsoft Docs.* [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/how-to-create-transparent-gateway (visited on 07/16/2020).

[10] *Create transparent gateway device - Azure IoT Edge — Microsoft Docs.* [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/how-to-create-transparent-gateway (visited on 10/26/2020).

[11] *Deploy blob storage on module to your device - Azure IoT Edge — Microsoft Docs.* [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/how-to-deploy-blob (visited on 10/27/2020).

[12] *Deploy module & routes with deployment manifests - Azure IoT Edge — Microsoft Docs.* [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/module-composition%7B%5C#%7Ddeclare-routes (visited on 07/16/2020).

[13] "Digital Transformation Initiative Oil and Gas Industry In collaboration with Accenture," Tech. Rep., 2017. [Online]. Available: www.weforum.org.

[14] L. Dürkop, B. Czybik, and J. Jasperneite, "Performance evaluation of M2M protocols over cellular networks in a lab environment," *2015 18th International Conference on Intelligence in Next Generation Networks, ICIN 2015*, pp. 70–75, 2015. DOI: 10.1109/ICIN.2015.7073809.

[15] F. Fadhil, "Analysis and Design of a Modern SCADA System," no. January, 2019. DOI: 10.13140/RG.2.2.32837.29921.

[16] *Five Terms You MUST Be Familiar With: SCADA, DCS, PLC, RTU and Smart Instrument — EEP*. [Online]. Available: https://electrical-engineering-portal.com/scada-dcs-plc-rtu-smart-instrument (visited on 07/17/2020).

[17] A. J. Harrington, H. Co-founder, C. B. Officer, D. Transformation, and S. Manufacturing, "White Paper DataOps : The Missing Link in Your Industrial Data Architecture The Problem : Drowning in Unusable Data An Introduction to Industrial DataOps Key Problems Solved by " The challenges of optimizing and controlling data flows coupled with the nee," pp. 1–4,

[18] Industrial Internet Consortium, "Introduction to Edge Computing in IIoT," *Industrial Internet Consortium White Paper*, pp. 1–19, 2018.

[19] *Industrial IoT vs SCADA: Which is More Powerful?* [Online]. Available: https://www.biz4intellia.com/blog/industrial-iot-vs-scada-which-is-more-powerful/ (visited on 10/12/2020).

[20] *Install Azure IoT Edge on Linux — Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/how-to-install-iot-edge-linux (visited on 07/16/2020).

[21] *Internet of Things and SCADA: Is One Going To Replace The Other?* [Online]. Available: https://iiot-world.com/industrial-iot/connected-industry/internet-of-things-and-scada-is-one-going-to-replace-the-other/ (visited on 10/09/2020).

[22] *IoT remote monitoring and notifications with Azure Logic Apps — Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-monitoring-notifications-with-azure-logic-apps (visited on 07/16/2020).

[23] Janice Abel, "Devon Energy Uses Real-time Data and Advanced Analytics to Make Better Decisions," *J Uly*, vol. 26, 2018. [Online]. Available: https://info.seeq.com/hubfs/Content%20Pieces/ARC%20Devon%20Energy%20Uses%20Real-time%

20Data%20and%20Advanced%20Analytics%20to%20Make%20Better%20Decisions.pdf.

[24] D. Jensen, *Beginning Azure IoT Edge Computing*. 2019, ISBN: 9781484245354. DOI: 10.1007/978-1-4842-4536-1.

[25] *Learn how the runtime manages devices - Azure IoT Edge — Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/iot-edge-runtime (visited on 07/16/2020).

[26] J. Leskovec, "Analytics on Time Series Data,"

[27] *Microsoft Customer Story-XTO Energy taps into IoT and the cloud to optimize operations and drive growth with Azure and Dynamics 365*. [Online]. Available: https://customers.microsoft.com/en-us/story/exxonmobil-mining-oil-gas-azure (visited on 08/02/2020).

[28] OPTO 22, "Meet the Future: Edge Programmable Industrial Controllers," 2019. [Online]. Available: www.opto22.com.

[29] E. N. Pi and L. F. Casta, "Trabajo Fin de Grado Grado en Ingeniería de Tecnologías Industriales Communications between PLC and microcontroller using Modbus Protocol," 2016.

[30] V. Poosapati MCA, V. Katneni MCA, and V. Killu Manda, "Super SCADA Systems: A Prototype for Next Gen SCADA System," *Iaetsdjaras.Org*, vol. 5, no. 3, pp. 107–115, 2018. [Online]. Available: http://iaetsdjaras.org/gallery/18-march-559.pdf.

[31] *Python WebServer With Flask and Raspberry Pi — by Marcelo Rovai — Towards Data Science*. [Online]. Available: https://towardsdatascience.com/python-webserver-with-flask-and-raspberry-pi-398423cc6f5d (visited on 07/16/2020).

[32] *Python WebServer With Flask and Raspberry Pi — by Marcelo Rovai — Towards Data Science*. [Online]. Available: https://towardsdatascience.com/python-webserver-with-flask-and-raspberry-pi-398423cc6f5d (visited on 07/20/2020).

[33] Richter and Herndrik, "R Emote T Actile F Eedback," 2013.

[34] *SCADA System*. [Online]. Available: https://www.electronicshub.org/scada-system/ (visited on 10/09/2020).

[35] D. Schmidt, D. Staal, J. McClung, M. Behl, and M. Tyagi, "Bench scale experimental study of slug flow phenomena using pid control," Nov. 2018, V007T09A055. DOI: 10.1115/IMECE2018-88564.

[36] *Store block blobs on devices - Azure IoT Edge — Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/how-to-store-data-blob (visited on 07/16/2020).

[37] A. Taha and M. Amani, "Introduction to Smart Oil and Gas Wells: Drilling, Completion and Monitoring Solutions," *International Journal of Petrochemistry and Research*, vol. 3, no. 1, pp. 249–254, 2019. DOI: 10.18689/ijpr-1000143.

[38] *the fundamental architecture of a PLC — AME 6004 —NPI and PLC*. [Online]. Available: https://yuzhangbolton.wordpress.com/2015/03/22/plc/ (visited on 10/09/2020).

[39] F. Tomé and N. Queirós, "FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO Azure IoT Edge Proof of Concept," Tech. Rep., 2019.

[40] *Translate modbus protocols with gateways - Azure IoT Edge — Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/deploy-modbus-gateway (visited on 07/16/2020).

[41] *Tutorial - Develop module for Linux devices using Azure IoT Edge — Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/tutorial-develop-for-linux (visited on 07/16/2020).

[42] *Tutorial - Stream Analytics at the edge using Azure IoT Edge — Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/tutorial-deploy-stream-analytics (visited on 07/16/2020).

[43] *Understand Azure IoT Hub MQTT support — Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support (visited on 07/16/2020).

[44] G. M. Weiss and H. Hirsh, "Learning to Predict Rare Events in Categorical Time-Series Data," *AAAI Workshop*, pp. 83–90, 1998.

[45] *What is Azure IoT Edge — Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/about-iot-edge (visited on 07/16/2020).

[46] B. Yee and P. Eng, "SCADA & IoT The Similarities and Differences," Tech. Rep. [Online]. Available: www.scadalink.com.

Vita

Derek Warren Staal was born in Washington State in the month of February, 1987. After attending commercial dive school he moved down to Louisiana for employment opportunities. His oilfield career lasted over six years before he decided to pursue higher education. Upon earning his B.S. in Petroleum Engineering Derek was afforded the option of Graduate School. It is during this time he discovered his passion for IoT, system integration, cybersecurity, and language acquisition. He plans to receive his master's degree December of 2020.

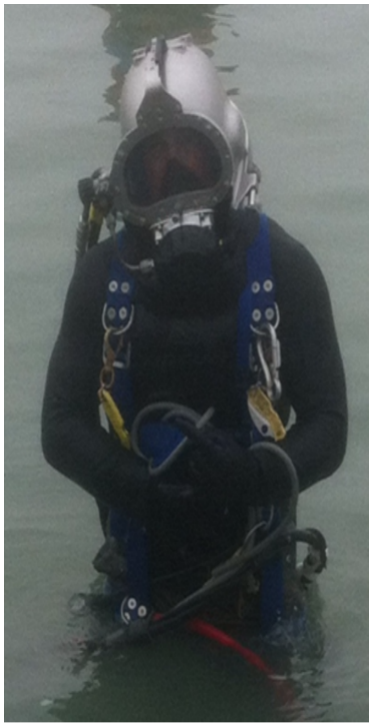

Figure I.1. A Lifetime Ago