Exploring Azure: Internet of Things and Edge

Dissertation

Rui Alexandre Farinha Fernandes Balau

Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos

Departamento de Ciência de Computadores

September 2019

Advisor

Rolando da Silva Martins

Co-Advisor

João Paulo Cardoso dos Santos

Abstract

The Internet of Things (IoT) is an emerging paradigm that increasingly permeates existing industries and possesses potential for old and novel practical applications alike. Lower hardware manufacturing costs and the ongoing miniaturization of electronic components subsidized the rise of IoT which at present poses unique architecture and design challenges.

Multiple novel applications indicate networks of potentially billions of *Things*. Home to nation-wide scale systems will exploit these advanced information gathering methods to provide added value services to urban systems, companies and individuals. The number of devices will strain existing telecommunications infrastructure, causing adverse results which must be promptly addressed before widespread use becomes commonplace. The expected pervasiveness of these systems and their data collection capabilities have spurred development of the next generation of manufacturing technology, representing a large share of visionary industrial projects such as China's Made-In-China 2025 and Germany's High-Tech Strategy 2020.

This dissertation explores the vision, architecture and operation of Microsoft's IoT offerings, beginning with establishing its benefits and liabilities and then capturing and describing its capabilities. Beginning by exploring the advantages and requirements of the technology as a whole and then surveying information on the view, methods and offerings, also at this stage addressing adoption related concerns such as pricing, security and data policies. A middleware to connect existing systems operating on unsupported protocols to the Azure Cloud was also developed, followed by an adaptation of it to Critical Manufacturing's (CMFs) Manufacturing Execution System (MES) product. We also extracted and prepared for publication a dataset of a real manufacturing process. Finally we analyzed the performance of the system and its varying supported communication protocols, comparing it to other related work.

Resumo

A Internet of Things (IoT) é um paradigma emergente e cada vez mais prevalente na indústria possuindo potencial para uso tanto em aplicações práticas existentes como novas. Esta tecnologia é impulsionada por custos reduzidos na manufactura de dispositivos e a contínua miniaturização de componentes electrónicos. A sua escala prevista apresenta novos desafios de arquitectura e planeamento.

As novas aplicações práticas previstas indicam redes de potencialmente milhares de milhões de dispositivos. É esperado que tanto sistemas domésticos como nações irão tirar partido destes métodos de colecção de dados para providenciar serviços de valor adicionado para sistemas urbanos, companhias e indivíduos. O número de dispositivos irá sobrecarregar a infra-estrutura de telecomunicações existente, potencialmente causando efeitos adversos que têm de ser considerados. A difusão esperada destes sistemas e das suas capacidades de colecção de dados têm estimulado o desenvolvimento do que é visto como a próxima geração de tecnologia para manufactura, e representa uma grande parte de projectos industriais visionários como Made-In-China 2025 e a High-Tech Strategy 2020 da Alemanha.

Esta dissertação explora a visão, arquitectura e operação das soluções de IoT desenvolvidas pela Microsoft, começando por estabelecer os benefícios e riscos, e posteriormente por descrever e definir capacidades. Nós começamos por explorar as vantagens e requisitos da tecnologia no geral e seguidamente reunir e avaliar informação sobre ponto de vista, métodos e produtos disponíveis, nesta fase também nos focamos em preocupações de adopção como preço, segurança e políticas de dados. Desenvolvemos um middleware para ligar sistemas já operacionais, através de protocolos não directamente suportados, à Azure Cloud, e seguidamente adaptámos esta solução ao produto Manufacturing Execution System (MES) da Critical Manufacturing (CMF). Além disso extraímos e preparámos para publicação um dataset de um processo real de manufactura. Finalmente analisámos a performance do sistema nos seus vários protocolos de comunicação suportados e comparamos os resultados com trabalho existente.

I would like to thank Critical Manufacturing for the opportunity, my advisors for their patience, my family for their unconditional support and lastly my friends, without whom this dissertation would have been finished earlier.

Contents

List of Tables

List of Figures

Acronyms

AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol
BASN	Body Area Sensor Network
C2D	Cloud to Device
CA	Certificate Authority
CLI	Command-line Interface
CMF	Critical Manufacturing
CPU	Central Processing Unit
CSP	Cloud Service Provider
DB	Database
DM	Data Mining
DNS	Domain Name System
DPS	Device Provisioning Service
DTDL	Digital Twin Definition Language
D2C	Device to Cloud
D2D	Device to Device
D2I	Device to Infrastructure
GS	Google Scholar
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IoT	Internet of Things
IIoT	Industrial Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
I/O	Input/Ouput
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
LTE	Long-Term Evolution
LTE-5G	Long-Term Evolution Fifth Generation
MES	Manufacturing Execution System
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
MS	Microsoft
NAT	Network Address Translation
NPM	Node Package Manager
OPC-UA	OPC Unified Architecture
OS	Operating System

PaaS	Platform as a Service
PoC	Proof of Concept
PoE	Power over Ethernet
QoS	Quality of Service
RDF	Resource Description Framework
RFID	Radio-frequency Identification
RPC	Remote Procedure Call
RSSI	Received signal strength indication
RTOS	Real-time Operating System
RTT	Round Trip Time
SaaS	Software as a Service
SDK	Software Development Kit
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
ToS	Terms of Service
UI	User Interface
URI	Uniform Resource Identifier
URN	Uniform Resource Name
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VM	Virtual Machine
WSN	Wide Sensor Network
W3C	World Wide Web Consortium

Chapter 1

Introduction

The Internet of Things (IoT) is an emergent technology that increasingly permeates existing industries and possesses capabilities for old and novel practical applications alike. Lower hardware manufacturing costs and the ongoing miniaturization of electronic components subsidized the rise of IoT which at present poses unique architecture and design challenges.

Its practical applications are expected to add millions, or even billions, of identifiable objects to the network, transmitting data that has to be stored and processed, often with Real-Time constraints. The number of devices will strain existing telecommunications infrastructure, potentially causing adverse results which must be promptly addressed before widespread use becomes common. This brings a new challenge for the Internet to adapt to all different kinds of devices and applications that use IoT, since the information collection methods and mobility requirements vary heavily among different use cases. By adding heterogeneous object interaction, potentially billions of objects communicating and processing data in real-time, there is an increased need for storage, processing, scale and to protect all the data flows on the network.

Critical Manufacturing (CMF) is a world leading Manufacturing Execution System (MES) company with high-tech manufacturing customers in America, Europe and Asia. CMF is currently looking to expand its offering in terms of IoT capabilities. Broadening the currently existing CMF IoT technology to leverage state of the art Cloud and Edge solutions offered by Microsoft allows for the easier deployment of the MES product to customers, reducing on-premise hardware costs, needed integration work and offering easier application support. It also allows customers easy access to Big Data and Machine Learning (ML) technology powered by the Azure Cloud.

With the existing wide range of available software suites selecting an option that meets organizational objectives and business requirements could prove to be a challenge. A hasty, precipitous choice could lead to a sub-par final decision that fails to address specific objectives and wastes company resources. It is equally important for businesses to review choices in detail as the adoption exposes programs and systems to failures and flaws in third party software. This work intends to explore the maturity of the Azure IoT technology, with a particular focus on the Azure IoT Hub and Edge. This maturity is considered in regards to the provisioning of an industrial, on-premises and Cloud, highly sophisticated and autonomous IoT solution with telemetry, insights and event processing capabilities; to this end properties such as supported communication protocols, privacy and security mechanisms, data throughput capabilities and pricing models are analysed. Modularity and compatibility are also evaluated with the implementation of a Proof of Concept (PoC) integration of Azure IoT with CMFs Connect IoT[1].

The remainder of this chapter outlines related work and lists the contributions of this dissertation. The rest of the dissertation is organized as follows. Chapter 2 briefly introduces the concept of Cloud computing and presents a literature overview of IoT vision, requirements and existing technologies. Finally it also provides a comprehensive analysis of Microsoft's IoT vision, the available offerings and their architectural design and practical uses. The chapter concludes with considerations of the Azure IoT ecosystem relevant for businesses looking into adoption. Chapter 3 describes developed work, its operation and implementation challenges. This chapter also outlines the methodology followed when preparing a dataset of real production data for publication. Chapter 4 defines the methodology followed when executing the performance analysis of Azure's IoT systems, presents and analyses the results obtained and concludes with insights and challenges met along the way. Finally chapter 5 concludes the dissertation by reiterating upon the objectives, the results obtained and discussing future work.

1.1 Related Work

The general IoT related literature is quite abundant, with well established publications for defining the foundational vision and requirements of the paradigm, e.g. Gubbi et al.[24]. Various well-reputed surveys explore possible different designs for complete systems where data is collected and used to derive actionable insights, taking great care to analyze the IoT enabling technologies and complementing fields of study, e.g. Da et al.[13] and Atzori et al.[8].

However in what regards to the analysis of commercial frameworks, both in relation to conceptual analysis and real world application the existing literature is scarce, this is an important gap to address as lately multiple IoT frameworks have been launched by major industry players, each building according to its own vision and targeted market niche [3]. Derhamy et al.[15] presents a quick overview of commercial framework designs with the goal of identifying common trends, of direct concern to this dissertation is a very brief listing of the available Azure IoT offerings and approaches. Ammar et al.[3] performed an in-depth analysis and comparison of the top IoT Frameworks designs and architectures, evaluating each frameworks authentication, authorization, access control, secure communication and privacy schemes. This research concluded that the same standard cryptography protocols are used to secure communications, namely Transport Layer Security (TLS) or it's successor Secure Sockets Layer (SSL), and that identical Cloud centric design philosophies are used to provide authentication, with different methodologies being used to achieve other security properties. Bakhshi et al.[10] compiled possible attacks

[1]Interchangeably referred to as MES Equipment Integration Module.

specific to Cisco's and Microsoft's IoT architecture with the goal of establishing a guideline to be followed for future security reviews of commercial IoT frameworks.

Of unequivocal relevance to this work is the performance and cost evaluation of the Azure IoT Hub performed by Forsström et al.[18] which explores the combination of the Azure IoT Hub and OPC Unified Architecture (OPC-UA) to form an Industrial Internet of Things (IIoT) system. This assessment was done by simulating sensor data and sending it through a Azure IoT Gateway[2] to the Azure IoT Hub. The evaluation was performed only with synthetically generated data and no data abstraction techniques were used. This appraisal concluded that the Hub was capable of operating at a Round Trip Time (RTT) of $770ms \pm 330ms$ for fiber based internet connections.

1.2 Contributions

In this dissertation we provide an overview for Microsoft's IoT solutions and their architectural and design choices, with special effort put in contrasting them to the established scientific vision where directly comparable, and offering comprehensive descriptions when not. Through documentation, official presentations and other publications we aggregate the overall vision, goals and offerings available, having found especially interesting the future goal of moving processing transparently between Cloud and Edge as the business ambitions and usage contexts require it. We also review important parameters of concern for industry adoption such as vendor lock in, existing development ecosystem and cost.

In the developmental exploration stage a Virtual Machine (VM) configuration and provisioning project was made for the quick and reliable deployment of the Azure IoT Edge runtime. Further on after conference with CMF representatives, to ensure a good allocation of efforts for both the dissertation and the company's goals it was decided that the default use case proposed by Microsoft, where the devices are capable of reaching the Cloud IoT Hub via internet connections, proved unrealistic and too intrusive to existing IoT setups. As such, we developed a protocol and identity translation middleware with the goal of allowing data to be exchanged with the Azure Cloud, without disrupting already existing operational systems. The final iteration of this middleware was fully integrated with CMFs MES as a proof of concept, via the existing MES Equipment Integration Module, which allows for automatic deployments, remote configurations and provides a baseline Graphical User Interface (GUI) toolkit. At this point we also proceeded to the preparation of a dataset for publication, the dataset comes from a real world manufacturing process of a CMF business partner and has been carefully anonymized due to privacy concerns. The final dataset contains 100880 rows of multi-relational data, with possible time correlations and we expect that it can be used for further work on the field of predictive maintenance. The dataset is published alongside this dissertation and available online[3].

[2]The Gateway has since the publishing of the paper became legacy and its capabilities are now a subset of features of the Azure IoT Edge service.

[3]doi.org/10.6084/m9.figshare.12681983

By developing and executing performance tests of the Azure IoT Hub with and without Edge devices and analysing the resulting data we confirm previous results[18] that the Azure IoT Hub is fast and reliable for use with multiple devices. We also extend that knowledge with the conclusion that the Azure IoT Edge technology, although promising, is not yet mature enough for adoption, derived mainly of the lacking results demonstrated on our performance test and more subjectively by the unstable behaviours of some functionalities offered by the Software Development Kit (SDK).

Chapter 2

Background

This chapter aggregates the relevant research gathered when initially formulating and exploring the topic of Internet of Things (IoT) and attempts to provide the reader with an identical level of comprehension of the field. After a brief introduction to Cloud computing the vision for the practical applications of IoT is laid out, the requirements to achieve the vision are examined and finally a summary of existing technologies and solutions that fulfill the requirements are delineated. The chapter concludes with an aggregation of official available Azure IoT information, intending to provide the reader with architectural and functional knowledge of the technology and its variation of the general vision for IoT, as well as possible risks when considering embracing the offerings for use in commercial settings.

2.1 Cloud Computing

In 1961 John McCarthy, an AI pioneer and Turing award winner, made a prediction that succinctly summarizes Cloud Computing: "Computing may someday be organized as a public utility just as the telephone system is a public utility . . . Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system . . . Certain subscribers might offer services to other subscribers . . . The computer utility could become the basis of a new and important industry." [19].

The National Institute for Standards and Technology of the US Department of Commerce has a widely accepted [31] definition of Cloud Computing: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [33].

This definition for Cloud Computing is composed of five essential characteristics, three service models and four deployment models [33] which we summarize here:

Essential Characteristics

- *On-demand self-service*: A consumer can provision computing capabilities, without human intervention on the side of the Cloud Service Provider (CSP).
- *Broad network access*: The Cloud services are made available over standard network mechanisms. Authentication may be required to access the Cloud resources.
- *Resource pooling*: As with old mainframe systems, multi-tenant models are used to pool resources available and serve multiple concurrent users. Physical and virtual resources are allocated and reassigned dynamically. The CSP usually provides customer isolation through network and software virtualization.
- *Rapid elasticity*: The computing resources can be elastically provisioned and released at the consumers request, or through some functionality related metric that ensures minimum resource allocation while maintaining capabilities that meet the target Quality of Service (QoS).
- *Measured service*: Cloud systems monitor and track resource usage, providing both the CSP and the consumer with accurate metering. This allows for detailed customer billing[1] and CSP specific Terms of Service (ToS) violation monitoring.

Service Models

- *Software as a Service (SaaS)*: The CSP exposes applications running on Cloud infrastructure to the consumer. The consumer does not have permissions to administrate the software or allocated Cloud resources, outside of limited user-specific application settings.
- *Platform as a Service (PaaS)*: The CSP allows the consumer to deploy, on the Cloud infrastructure, consumer-created or acquired applications that are created in languages and libraries supported by the CSP. The consumer does not have permission to administrate the allocated Cloud resources, outside of limited configuration settings for the application-hosting environment.
- *Infrastructure as a Service (IaaS)*: The CSP gives the consumer infrastructure provisioning capabilities[2] and arbitrary code execution[3]. The consumer can therefore administrate allocated Cloud resources at will, but generally does not directly manage or control the hardware or its allocation.

[1]Billing is commonly performed identically to domestic utility services, on a pay per use basis.

[2]The consumer can select infrastructure level requirements such as processing, storage and network that the CSP then allocates for the consumer to control.

[3]The consumer can install any compatible software, even fully fledged operating systems, however CSP infrastructure isolation is used to limit access to other user spaces. This is critical as multi-tenant models can allocate the same physical hardware to multiple consumers.

Deployment Models

- *Public Cloud*: Open for use by the general public, may be owned by a for-profit or public organization. Exists on the premises of the CSP.
- *Private Cloud*: Used exclusively by a single organization with multiple consumers. May exist on organization or third-party premises.
- *Community Cloud*: Shared by several organizations with multiple consumers. The organizations typically have shared concerns in regard to policy and regulations. May exist on organization or third-party premises.
- *Hybrid Cloud*: Any combination of two or more deployment models that result in unique Cloud entities while maintaining interoperation.

2.2 IoT

There is at present no consensual definition of what IoT is, as multiple organizations present disparate interpretations [11]. CCSA defines IoT as a network which can collect and control information from the physical world through various deployed devices with capability of perception, computation, execution and communication ITU-T defines it as a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies. EU FP7 defines it as a global network infrastructure, linking physical and virtual objects through the exploitation of data capture and communication capabilities. IEFT defines it as a world-wide network of interconnected objects uniquely addressable based on standard communications protocols. Finally the CASAGRAS consortium defines it as *things* that can automatically communicate to computers and each other providing services to the benefit of the humankind.

A functional IoT architecture typically shares the following three common characteristics and components [48], regardless of the formal definition used:

- *Sensor devices*: The *things* in IoT. This typically refers to low-power, low resource devices, equipped with hardware capable of sampling a desired metric (e.g. temperature, density, distance). These devices routinely generate samples and transfer them over the Network. Sensor devices are also sometimes called nodes in the literature.
- *Network*: This layer is responsible for transferring data, typically from sensor devices to an information processing system. With the existing Internet infrastructure, the Transmission Control Protocol (TCP)/Internet Protocol (IP) stack is one of the most prominently used IoT enabling technologies [2], and as such multiple sources in the literature assume that communication is reliable and sometimes even secure [8] [27], however this is not necessarily the case as a functional network layer can provide lossy asynchronous communication from a sensor device to a edge server. The security of communication in IoT can never be assumed.

Devices are often vulnerable to physical attacks and have low computing capabilities, which makes it hard to implement secure communication schemes [8]. Devices can often remain in active duty for decades, adding another attack vector as cryptographic ciphers often need to be cycled out as they are broken [3]. For use cases where devices act not only as sensors but also as actuators and some form of auto-configuration occurs, possibly through Machine Learning (ML), devices must support bidirectional communication.

- *Application layer*: The Application layer represents the interaction point for most end-users, providing services, configurations and access to data at a user's behest. As an example these applications may help users regulate their home environment or help doctors monitor patients.

More recently five layer models have been proposed that extend the IoT architecture by adding a Middleware and Business layer [27] [2].

- *Middleware layer*: Sits between the Network and Application layers and allows the application layer to express desired behaviour, while abstracting away the need for implementing the behaviour on the particular physical network infrastructure.
- *Business layer*: Lies logically above the Application layer, leveraging the collected information to create symbolic representations (e.g. graphs and diagrams) that provide support for decision-making processes at a business process level and generally involves more Big Data techniques; better oriented to large processes and corporations due to the vast amounts of data required.

2.3 Potential applications for IoT

Being a very promising technology, IoT comes with capabilities and possibilities that make it a great solution for applications that use real-time data to manage and interact with intelligent systems. Small-scale electronic devices with identification, communication and computing capabilities are embedded on IoT systems to make the following three characteristics available: *Comprehensive Perception*, *Reliable Transmission* and *Intelligent Processing* [32]. IoT applications are then capable of some of the following capabilities: Location Sensing and Sharing Location Info, Environment Sensing, Remote Controlling, Ad hoc Networking and Secure Communication [11]. This allows applications from many different fields to adapt software solutions to ease our daily life based on the already existing networks and on evolving ones, creating new ways of interacting with established technologies and our devices at home, offices, factories, public buildings, streets and vehicles.

2.3.1 Home

Home IoT networks have a large potential to provide immediate improvements to its users quality of life.

IoT devices can be integrated with common household appliances to provide automatic management and remote administration of repetitive tasks. Pet feeders can be set to follow specific schedules that may be modified remotely, fridges with stock tracking features can populate smartphone shopping lists on-demand. Automatic lightning and temperature control alongside with programmatic management of interactive media and entertainment systems can also allow for better energy management [35].

Monitoring systems with integrated network capabilities have existed for some decades, however the breakthroughs in Cloud and Edge computing alongside with the prevailing availability of smartphones with high bandwidth mobile internet connections allows for remote interaction with security systems. Physical security is one of the main areas with practical applications that could benefit from the creation of peer reviewed authentication and authorization standards for distributed IoT devices.

Wearable Body Area Sensor Networks (BASNs) envisioned to solve traditional healthcare issues of irregular data collection intervals and qualitative collection can now be manufactured and show great promise in extending regular non-intrusive monitorization from hospitals to patients homes [26]. Wearable BASNs also show great market promise as gadgets for fitness and sports tracking, as can be seen by the current market valuation of companies such as Fitbit that specialize exclusively in the production of said gadgets.

As smart home technology matures it can be safely expected that machine learning techniques will be applied to the large amounts of personal user data generated by IoT devices and automatically adapt to target users habits, preferences and needs.

2.3.2 Public and commercial buildings

Both public and commercial buildings, including factories, can benefit greatly from practical IoT applications.

Environmental sensors can keep track of the number of occupants and users of shared spaces, registering the present state of facilities and, based on past data, make well-informed predictions of the use of said facilities, allowing for the preemptive maintenance and upkeep optimization of shared environments.

Indoor tracking via Bluetooth beacons and Received signal strength indication (RSSI) tracking methods have become increasingly widespread and IoT can leverage these methods to facilitate the collection of user movements by tracking users mobiles devices in common areas.

With recent technological developments in IoT, Data Mining (DM), Cloud and Artificial

Intelligence (AI) the concept of the *Industry 4.0* has developed [45]. This concept is touted as a new industrial revolution that integrates computer system advancements with production facilities and distribution networks with the goal of improving the industry capabilities to deal with physical and logistical global challenges. Although there is no formal definition, new installations that meet the *Industry 4.0* vision are often dubbed *Smart Factories* in the literature and by the industry [40] and typically use ubiquitous IoT devices to provide autonomous control of production lines and tracking of item lots along production and distribution networks. An example of such automated factories can be found on the prototype by the German Research Centre for Artificial Intelligence that produces custom shampoos which meet user placed orders without human intervention, each bottle is tracked via Radio-frequency Identification (RFID) and the production machines organize production schedules via communication over a wireless network [45].

In recent years the need for the optimization of production schedules has increased. Job Shop Scheduling is as a well known optimization problem in the literature that attempts to solve the increasingly complex management of production line manufacturing [7]. The techniques for implementation of this academic effort in real world production require large quantities of data with verifiable quality. IoT machines belonging to a *Smart Factory* can collect and offload the data into a Edge or Cloud node so that it may later be processed and studied, with the end goal of optimizing the production pipeline.

Identically to homes larger buildings can also benefit from applications of IoT to energy management, authentication and surveillance. The larger scale of the buildings also warrants implementation of surveillance systems with automatic behaviour analysis and outlier event detection [24].

2.3.3 Urban landscape

Smart cities are the natural extension of IoT paradigms from a home or shared facility to a large community or entire city, and generally possess the goal of efficient, sustainable and always available management of resources in urban areas.

Monitoring and management of utilities allows for the optimization of services, and it is already used by electric providers to improve the profit margins offered by their services [24]. Infrastructure utilities such as gas, electricity, drinking water and sewage disposal can benefit from the plethora of information that can be datamined from collected IoT data. The structural health of critical buildings can also be monitored.

The collected utility information can be applied to:

- Improve sustainable use of resources by maintaining load balance.
- Meet QoS requirements by monitoring resource availability, detecting weak links in distribution chains and undersupplied areas.

- Immediately generate damage reports after natural disasters and similarly destructive events.

Ubiquitous street sensors can provide multiple avenues to increase street safety and traffic efficiency at infrastructure levels. Dynamically changing street signals can be used to load balance roads. Sound monitoring systems can identify large volumes of traffic and possible accidents or shootings nearly instantly, which can then be manually reviewed by human operators via closed circuit traffic monitoring. Pervasive surveillance systems can improve physical security of urban residents and help police forces in suspect identification and profiling, with audio sensors even being used to identify possible gun types [28].

Public transportation systems can also benefit from IoT integration with existing fleets via Vehicle-to-Infrastructure (V2I) communications [6]. Tracking of bus and metropolitan lines allows users to be notified in real time of the expected time to arrival. Location tracking of taxi vehicles allows for the implementation of centralized systems with fair attribution and quick response time to client fare requests.

2.3.4 Vehicles

A ubiquitous implementation of IoT concepts to vehicles, including private transportation, allows for even more features than the already described on the previous subsection. Besides data gathering, the practical applications of IoT enabled vehicles, are classified as *active road safety* and *traffic efficiency* [6].

Active road safety applications attempt to reduce car accidents. Implementations of Vehicle-to-Vehicle (V2V) communication systems allow users to notify other users and navigation systems of possible road accidents and other hazards, in close vicinity. Direct autonomous communication between vehicles in the immediate vicinity allow for the exchange of extra sensory information that can then be used for early warning collision systems, improved self driving capabilities and anomaly detection.

Pervasive IoT enabled vehicles allow for more reliable *traffic efficiency* applications. Periodic V2I communication provides the infrastructure system with information regarding the particular car and surrounding environment. The infrastructure in turn rewards the user with better routes that consider road congestion and general hazard warning systems, which provide functionality akin to the variable message signs already used worldwide. The collected information of car and road usage also provides the road infrastructure with reliable information of road usage which can then be used to optimize maintenance and management of the road itself.

2.4 IoT Requirements

IoT technologies must possess robust characteristics that ensure the correct and safe operation of their desired capabilities. Multiple articles have already tackled this issue, some with a focus on specific real world applications [35] [43] [25] [11]. In this chapter we summarize critical areas and functionalities new IoT devices and networks must consider, with the purpose of further easing the modernization of existing infrastructure and improve interoperability between multiple manufacturers.

2.4.1 Communication protocols

Even the most basic IoT functionalities require strong capabilities of Device to Device (D2D) and Device to Infrastructure (D2I) communication. D2D communication can be done without recourse to the internet, and many Wide Sensor Network (WSN) such as the ones created by vehicular applications, require the capability to transmit messages to nodes in the geographical vicinity, which is refereed to as *geocast* and may operate by providing direct V2V communication [6]. However more advanced capabilities that provide easier integration with other emerging technologies, such as Cloud and DM, generally expect these devices to seamlessly operate over the internet and communicate with centralized infrastructures, which has led to widespread adoption of IP protocols in existing systems [45].

With IoT being an interconnected world-wide network the need for well developed communication protocols is apparent. It is the authors view that this is currently one of the most solidly fulfilled functional requirements, as existing communication protocols for both wired and wireless technologies have developed mature specifications which we outline in Section 2.5.1.

2.4.2 Node identification

IoT networks are expected to interconnect a tremendous amount of nodes; additionally, many practical applications operate in situations of high mobility in which the user maintains outgoing connections while passing through multiple access points, creating highly dynamic networks where traditional naming schemes such as Domain Name System (DNS) cannot keep up with network changes [41].

The ability to uniquely identify the nodes belonging to the network is critical for the success of IoT, just as much as it is for any other computer system area where addressing and data communication is needed [41].

2.4.3 Energy consumption

The internet consumes five percent of the total energy generated world-wide and this number is expected to increase in the future, especially as IoT devices become much more numerous and common [24]. Under the current growth vision for IoT the quantity of data centers and related infrastructure is expected to increase as well.

As a result one of the questions that should be thoroughly investigated is if the need for gadgets and other miscellaneous internet connected devices outweighs the environmental and finite resource concerns associated with the extra energy consumption.

In either case, many devices will run on batteries or identical power sources and are expected to function for years without recharges [25]. Efforts to optimize energy usage of IoT devices by creating specific energy saving measures must be undertaken.

2.4.4 Real-Time capabilities

Practical applications described on the vision section, such as BASN and utility monitoring, demonstrate clear needs for the precise timing of data collection. For other applications, servers in the Cloud provide high capacity for computation that far outweighs the capabilities of typical IoT devices, handling complex workloads in a relative short time. Applications that mean to use these methods in a timely fashion, such as autonomous vehicles, are sensitive to delay's and instabilities caused by the network as they can degrade the quality of the provided service [41].

Operating systems of IoT devices should be capable of providing worst-case execution times guarantees and worst-case interrupt latencies [25]. Networks should support capabilities for resource reservation and QoS guarantees.

2.4.5 Modularity and compatibility

The large variety of use cases has led to the development of many hardware and communication technologies in the IoT world [25]. Before the evolution from local software producers to global standard software packages users and companies bound themselves not only to the product but also to that particulars software producer's capability to develop new functionalities [14], increasing the risk of the adoption of new technologies and personnel training. As the usage of IoT devices increases, compatibility standards and shared libraries for common features should follow, reducing the risk and development costs of organizations that purchase and implement IoT solutions.

IoT Operating Systems (OSs) must support different hardware requirements and multiple software stacks. Development of non-proprietary hardware architectures in conjunction with device categorization based on resource constraints, such as Central Processing Unit (CPU) specifications and memory capacity, would provide reprogrammable and interchangeable support for applications running on heterogeneous devices from various companies.

2.4.6 Data volumes

Since its inception the Cloud has been providing global decentralization, scalable infrastructures, data storage and computing capabilities. The large amounts of ever growing internet traffic have created a bottleneck for the Cloud-based computing paradigm, making its use prohibitive [41]. Edge computing allows the data generated by IoT devices to be triaged locally before it is sent to a central processing or storage repository, ensuring that inconsequential information is not transmitted.

In addition information abstraction techniques can be used to reduce the barrage of data collected [20]. By using the abstracted information instead of transmitting the entirety of data points, information can be stored at Edge nodes and detailed views are populated with variable detail as needed.

The scale of the data produced by a *smart city* presents unique challenges, a city of one million inhabitants is estimated to produce one hundred and eighty petabytes of data per day [41], as such size scalable sensor networks must carefully consider its usage of network resources.

2.4.7 Privacy

Pervasive use of commercial smart home IoT devices that do not obey strict data collection and ownership standards can turn the expected benefits of custom tailored human habitats against the end user. The amount of personally identifiable data or even just anonymized statistical metadata generated by systems in such close proximity to the end users routine may allow predatory marketing techniques by corporations and expose the user to the serious consequences of malicious data breaches.

Due to the sensitivity of collected data, most collected sensor information is expected to only be consumed and accessible in the immediate vicinity to the private network. In this scenario Cloud computing is unsuitable due to privacy and generated network traffic. A dedicated Edge computing server node inside the private network can provide identical computational benefits without the cloud computing drawbacks [41].

With the ubiquitous deployment of information and communication systems that can be invisibly embedded in the environment, sensor networks that enable identification of objects, individuals and locations should be forced to adhere to legal standards and ethical requirements that respect users rights.

2.4.8 Security

As the number of IoT devices increases and their usage becomes common in areas where privacy and security are essential, the need to responsibly handle the data produced becomes apparent. Many IoT applications possess a single focus and lack a strong consideration for security risks and good practices [42].

A survey of retail banking systems security revealed that most failures were caused not by technical attacks but by cryptography implementation errors [4], making a strong case for the use of *Open Security* [46], that is improving security through collaboration on shared, peer-reviewed libraries.

The use of well established libraries that conform to tested, carefully developed and known cryptography standards, alongside with a robust set of security best-practices, should be one of the priorities for the industry moving forward.

2.5 Existing IoT Standards and Technologies

Software development standards set well thought and reliable foundations for the evolution of any technology. Standards grant devices built by disparate manufacturers effective communication protocols that allow for easier deployment and configuration by the end user. Often the competition of hardware manufacturers and software developers causes the creation of proprietary quirks that offer slight real world advantages to its users but hinder the growth of emerging technologies by greatly diminishing interoperability. This section describes existing IoT methods and standards that support the evolution of the IoT paradigm.

2.5.1 Communication protocols

The IoT network uses many communication protocols and different types of connectivity, it may have wired connected nodes or wireless connected nodes. Wireless and mobile nodes, may interact with many different networks and such devices should be adaptable to the changing environmental conditions and remain usable while moving. The robustness of wireless connections should be provided by the communication protocols specifications.

IoT is enabled by the already well established computer networking standards and for the most part operates identically to existing networks, with the main implementation differences residing at the physical layer level. The move to push internet into using the Internet Protocol Version 6 (IPv6) protocol developed by the IETF has provided IoT with two major advantages: The increase of addressing space and the improved Mobile IP support offered. Internet Protocol Version 4 (IPv4) addressing space has become increasingly restricted, even when techniques such as Network Address Translation (NAT) are used. With the expected billions of devices joining the network IPv6 addressing schemes become a must have. As mobility is one of the biggest concerns for practical applications such as vehicles and autonomous drones Mobile IP becomes very useful, providing a device with a home network that allows for ongoing connections as the node moves through different access points.

New specifications in wireless physical level transmission enable multiple methods of device communication. For close range, low-bandwidth low-energy transmission some manufacturers are using the IEEE 802.15.4 standard that operates at frequency bands of 868, 915 and 2450 MHz and offers D2D communication without the need for an underlying structure. Other manufacturers may find IEEE 802.11ah more attractive [1], which provides lower energy consumption Wi-Fi networks and provides higher transmission ranges.

Long-Term Evolution (LTE) is the most promising wireless broadband technology that provides high data rate and low latency to mobile users [6] and has a wide infrastructure implemented all over the world, making LTE more suitable for vehicular communications (V2X) than the first standard for vehicular communications 802.11p [12]. Long-Term Evolution Fifth Generation (LTE-5G) is expected to make the best out of 4G high data rates and create an infrastructure with even better performance and totally adapted for the IoT environment.

For many practical applications the communication protocols can also offer useful location tracking capabilities. The IEEE standard 802.11mc added Wi-Fi Round Trip Time (RTT) which enables supporting devices to improve location sensing capabilities. The feature is provided by a fine timing measurement protocol, which measures the time a Wi-Fi frame spent airborne [9]. When multiple measurements are performed to Wi-Fi devices at known locations and under optimal conditions, a triangulation of the location with less than two meters of accuracy can be achieved [5].

Finally, for communication between IoT devices and objects, equipping physical objects with RFID tags that can be scanned by smart devices has become an increasingly common practice [27]. Object related information is embedded on the RFID tags and allows the smart device to correctly identify the specific object and its associated metadata, which may contain information on destination, contents and similar attributes. This allows smart IoT devices, such as production line machinery, to immediately identify the object they are handling.

2.5.2 Node identification

Currently no developed IoT specific standards exist for node addressing. A Uniform Resource Name (URN) scheme has been proposed by Gubbi et al, 2013 [24]. The use of Edge servers responsible for handling local IoT device communications and data collection allows the usage of classic master-slave architectures in relatively static environments where the master can manage the addressing of IoT devices or at the very least work under a shared message broker [41].

2.5.3 Energy

Two major techniques for Power over Ethernet (PoE) have been developed by IEEE on the 802.3 specification that allow reliable power transmission to wired IoT devices, however energy efficiency remains a priority for cases where cabled connections can not be established. Low-energy wireless communication standards alleviate energy consumption, but should be accompanied with specific OS implementations that provide radio duty cycling or minimize the number of periodic tasks that need to be executed [25]. Due to the typically proprietary and case sensitive implementation of energy saving measures standards are unlikely to develop in this area.

2.5.4 OSs and platforms

OSs must provide energy efficiency, real time capabilities, security libraries and support for heterogeneous communication protocols. OSs like RIOT focus on providing future proofed "Linux-like" capabilities and implement IoT relevant open standards, offering standard programming libraries and minimized hardware dependent code, by supporting standard programming languages like C and C++. RIOT is currently the most prominent open source OS [25]. For devices operating on even stricter hardware and energy requirements Contiki and TinyOS provide open source options that implement the typical IP stack alongside with low power wireless standards.

2.5.5 Security

A comparative study of different cryptography libraries intended to be used for IoT concluded that none provided universal functionality, due to varying features, platform specific optimization's and hardware requirements[29]. A study that contained devices with higher computational prowess capable of connecting to typically Cloud based IoT systems found identically that although most IoT frameworks use standard cryptographic protocols for secure communications, Transport Layer Security (TLS) and Secure Sockets Layer (SSL), they used different methods to achieve security mechanisms such as access control and authorization[3]. As IoT develops these libraries are expected to mature alongside operating systems and lead to a more secure, reliable and universal IoT infrastructure.

2.6 Azure IoT Review

This crucial section of the thesis intends to explore Azure's IoT design and vision, present the reader with the existing offerings and their intended use cases and lastly describe how common IoT challenges are tackled. Subsection 2.6.1 begins by promptly portraying Microsoft's vision for IoT and its use cases, so as to provide the reader with a better context for the expected use cases of the technology, then it details architecture design, fundamental concepts and terminology of Azure IoT as well as the concrete offerings and services. Subsection 2.6.2 concludes the section by providing a summary of techniques used by Azure IoT to mitigate common IoT challenges and outlines policies, market adoption and pricing models.

2.6.1 Microsoft's IoT vision and services

Microsoft's recognizes that IoT is a powerful emerging technology that can be applied to many different problem domains and believes it fundamentally amounts to a business revolution with significant impact across multiple industries. They intend to provide these industries with scalable, automatically provisionable and secure IoT solutions [23]. In line with this belief, Microsoft is currently investing five billion dollars in IoT, on top of the one and half it already has invested [47].

According to Microsoft this business revolution happens gradually over multiple iterations, in a what they call a Digital Feedback Loop, this concept in essence relates to how with IoT gathering telemetry and extracting information from it enables a business to iteratively improve their operations. For example consider Thysenkrupp elevators [23], a company which services around one point two million elevators worldwide. This company went from servicing elevators reactively to client contact, to proactively servicing them with optimized schedules, to then being capable of identifying what parts would need to be replaced before any technicians physically visited the site. This allowed the company to service elevators 50% less while offering something akin to Devices as a Service, where the company pivoted from offering not just servicing contracts but actual uptime guarantees.

The essence of Microsoft's view on IoT generally matches that of the scientific community, where *Things*, capable of *Comprehensive Perception* and *Reliable Transmission* can be used to collect data, that is used to derive *Insights* which in turn indicate the *Actions* to take. IoT can be daunting to adopt for digitized industries, and even harder for others that provide physical services. Microsoft wants to capture the largest possible customer base by abstracting away implementation complexity and reducing development times, helping customers scale, analyze data, setup business rules and manage device life cycles. For this generalized customer they have developed two major offerings: IoT Central and IoT Solution Accelerators.

IoT Central is a SaaS horizontal offering for straightforward IoT solutions. Its a fully managed solution where no underlying services are exposed, application and device templates are given.

Customization is limited to tweaking the User Interface (UI) of the prebuilt web applications.

IoT Solution Accelerators are canonical IoT patterns, such as monitoring a device's health and anticipating its maintenance needs. The kind of patterns that are so wide-spread and common to multiple industries that they have already been developed before, but may need tweaking for some particular implementation. These Cloud based solutions are offered as complete, ready to deploy PaaS IoT solutions with Cloud services that are automatically provisioned under the Azure subscription. They typically act as starting points for custom development, already implementing the reference architecture[34]. The source code for all microservices is open source, access and management of the underlying Azure services is given and the deployment infrastructure can be customized.

To accomplish their intention both of these offerings internally use many other Azure services, a subset of which is listed on Table 2.1. A completely custom solution can be developed using these services, without relying on the IoT Central or Solution Accelerators. As per the goal of this thesis our analysis focuses on services directly related to IoT: The Azure IoT Hub and the Azure IoT Hub Device Provisioning Service (DPS).

Table 2.1: Example Azure IoT related Cloud services

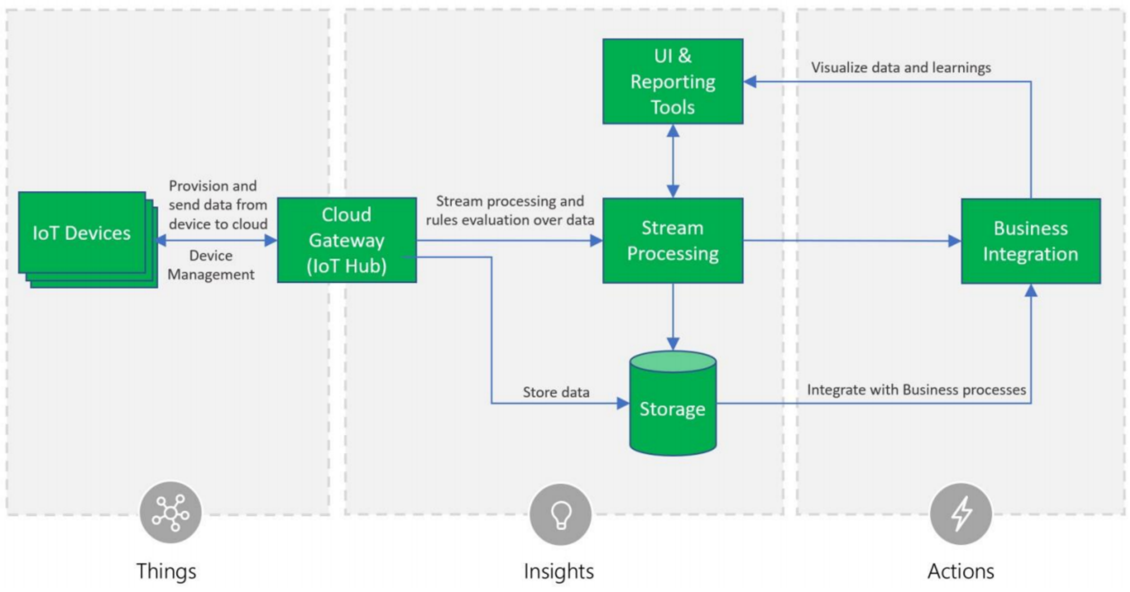
IoT Services	Data and Analytics	Visualization and Integration
IoT Hub	Stream Analytics	Container Registry
IoT Hub DPS	Time Series Insights	Azure Websites
Digital Twins	Event Hubs	Azure Power BI
Windows IoT Core	Cosmos DB	Azure Functions

The main functionality offered by the Azure IoT Hub is to act as a Cloud gateway, the data entry point to the Azure ecosystem where the data can be stored and analytics performed, a visualization of which can be found on Figure 2.1. The IoT Hub does bidirectional communication over common communication protocols such as Message Queuing Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP) and HyperText Transfer Protocol (HTTP), with some support for extensibility. This communication scheme assumes devices are running software developed using one of the IoT Hub Device Software Development Kits (SDKs), a Middleware capable of connecting to the IoT Hub or have an intermediary node support translation of custom protocols.

The Device SDKs are multi-language and Open Source, they implement a standard interface for developers to build applications that connect to and are managed by the Azure IoT Hub service. More than just simple Device to Cloud (D2C) or Cloud to Device (C2D) capabilities the SDKs also implement Device Twins, Direct Methods, Device identity management, provisioning tooling, message routing and end to end security.

Device Twins are state synchronization primitives used to manage devices. They contain information represented as JavaScript Object Notation (JSON) name-value pair property collections. A *Desired Property* is set from the Cloud and represents a way of ordering the device to go into a specific state. A *Reported Property* is sent from the device to the Cloud and

Figure 2.1: Core subsystems [34]


UI & Reporting Tools
Visualize data and learnings
IoT Devices
Provision and send data from device to cloud
Device Management
Cloud Gateway (IoT Hub)
Stream processing and rules evaluation over data
Stream Processing
Business Integration
Store data
Storage
Integrate with Business processes
Things
Insights
Actions

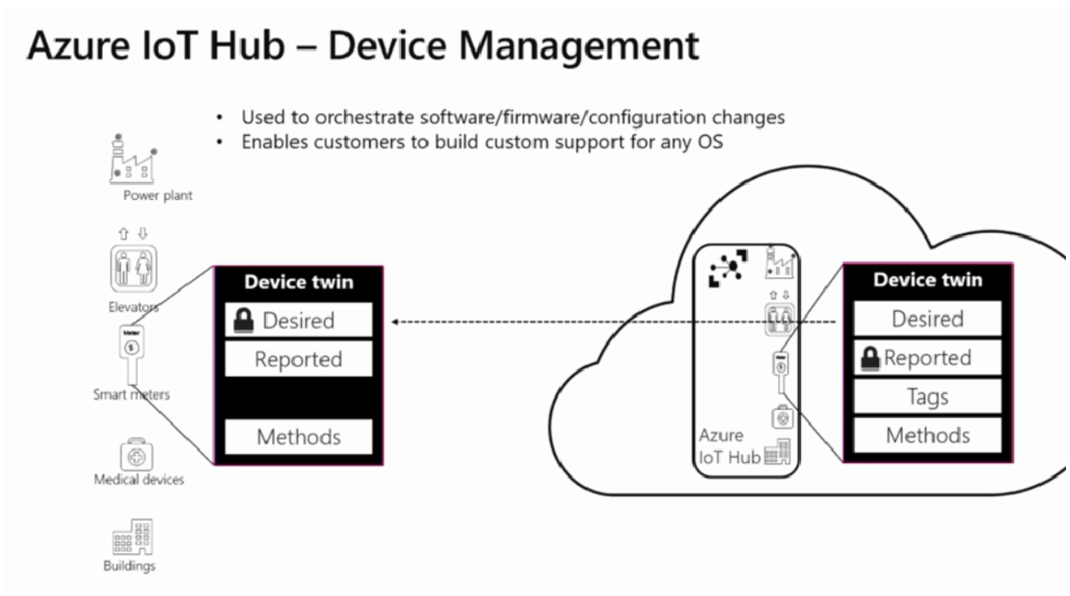
represents the actual state the device is in. Generally the Cloud service will attempt to set a *Desired Property* until the *Reported Property* matches up, Figure 2.2. When a device goes offline for any reason it is resynchronized upon reconnection. Additionally Device Twins on the Cloud also have *Tags*, which are a way of logically grouping identical devices for easier querying and management. An example use of *Tags* for automatic device management is that of a *Job*, which queries devices registered on the IoT Hub for a particular tag and sets *Desired Properties* on the resulting set of devices. *Direct Methods* offer ways to order devices to execute a particular block of code and possibly return its result, similarly to Remote Procedure Calls (RPCs). Upon connecting to the IoT Hub the device listens to events on a specific endpoint and reports back acknowledgements after it executes.

The IoT DPS is a service that complements the IoT Hub; this service aims to provide at scale device provisioning by centralizing the logic required for allowing devices to be associated with any particular solution at runtime. Devices are preregistered on the global DPS endpoint and come equipped from the manufacturer with a nonce[4] or X.509 certificate for authentication with it. Once the device is first turned on it will contact the DPS service at a predetermined Uniform Resource Identifier (URI) and establish its identity, the DPS service then registers the device with the appropriate IoT Hub, gives it the credentials it requires to authenticate with it and sets the Device Twin configuration. The device then disconnects from the DPS system and reconnects with the IoT Hub, triggering the device management workflow and eventually beginning normal operation. Attributed IoT Hubs can be statically set or dynamically allocated, even by geosharding. It is also possible to trigger automatic re-provisioning, which facilitates migration scenarios.

Microsoft's take on Edge is particularly interesting and unique, rather than see Edge as its

[4] A one time use number or string in the context of authentication protocols.

Figure 2.2: Device Management [23]


Azure IoT Hub – Device Management
Used to orchestrate software/firmware/configuration changes
Enables customers to build custom support for any OS
Power plant
Elevators
Smart meters
Medical devices
Buildings
Device twin
Desired
Reported
Methods
Azure IoT Hub
Device twin
Desired
Reported
Tags
Methods

own paradigm they see it as an extension of the Cloud, using it to develop a Hybrid Cloud model where computation can be opaquely executed and migrated between Public Cloud infrastructure and Private Edge infrastructure. The theory is that Edge should enable services to run seamlessly where needed, whether that means on the Cloud or on premises (at the Edge) while maintaining the ability to remotely manage both the device and the its workloads[22]. As for Edge use cases, they remain more or less in line with the ones more commonly proposed by the scientific community: perform local processing and control functions for devices, provide connection multiplexing and shield downstream devices from the public internet, and finally buffer, filter or aggregate device telemetry to control data volumes sent to the Cloud [34].

Containers are a great unit of portability between the Cloud and the Edge, with a negligible impact on performance and energy consumption[36]. Microsoft leverages the Docker container technology to achieve consistency between the Cloud and the Edge, scalable deployment and provisioning, and lastly application isolation. Microsoft has ported some typical Azure Cloud services into containers to be used in such a fashion. Recently they also have introduced the capability to manage IoT Edge workloads as on premises Kubernetes clusters. This intends to help alleviate customer needs for high resiliency while still supporting the Hybrid Cloud model envisioned.

Azure IoT Edge can also act as a solution to provide device connectivity to the Hub for devices that are not capable of the running the SDKs or any of the protocols supported by the IoT Hub. Currently only the Modbus and OPC Unified Architecture (OPC-UA) protocols have officially supported modules, with OPC-UA being the more developed of the two, likely due to the influence of the partnership with the OPC Foundation[21]. Any other protocol will require development of custom code, this is particularly challenging as at present no scaffolding, library or recommended structure exists and documentation in this particular area is sparse.

There are three gateway patterns supported by IoT Edge, Figure 2.3:

- *Transparent Gateway:* For devices that are capable of directly communicating with a IoT Hub, the Edge device can act as a transparent Gateway that bridges communication between the devices and the Cloud. It is transparent because neither the devices communicating through the Gateway nor a user managing the devices from the Hub is made aware of the intermediary facilitating the connection.
- *Protocol Translation:* The Edge device bridges the connection between devices running an unsupported Protocol and the Cloud, acting as the sole source of data. It communicates with downstream devices through a common protocol and bridges that information to the Cloud using its own identity. This also means that for information to be analyzed and acted upon on a per device basis additional source identification data must be transmitted on the messages.
- *Identity Translation:* The Edge device bridges the connection between devices running a unsupported protocol and the Cloud, using a Middleware that implements a parity for features existing on the official SDKs and also provides identity translation, allowing the devices to be identified and managed from the Cloud, as if they were IP capable and running the official SDK. If the Middleware fully implements parity with SDK features a user interacting with devices from the Cloud is unaware of the intermediary facilitating the connection.

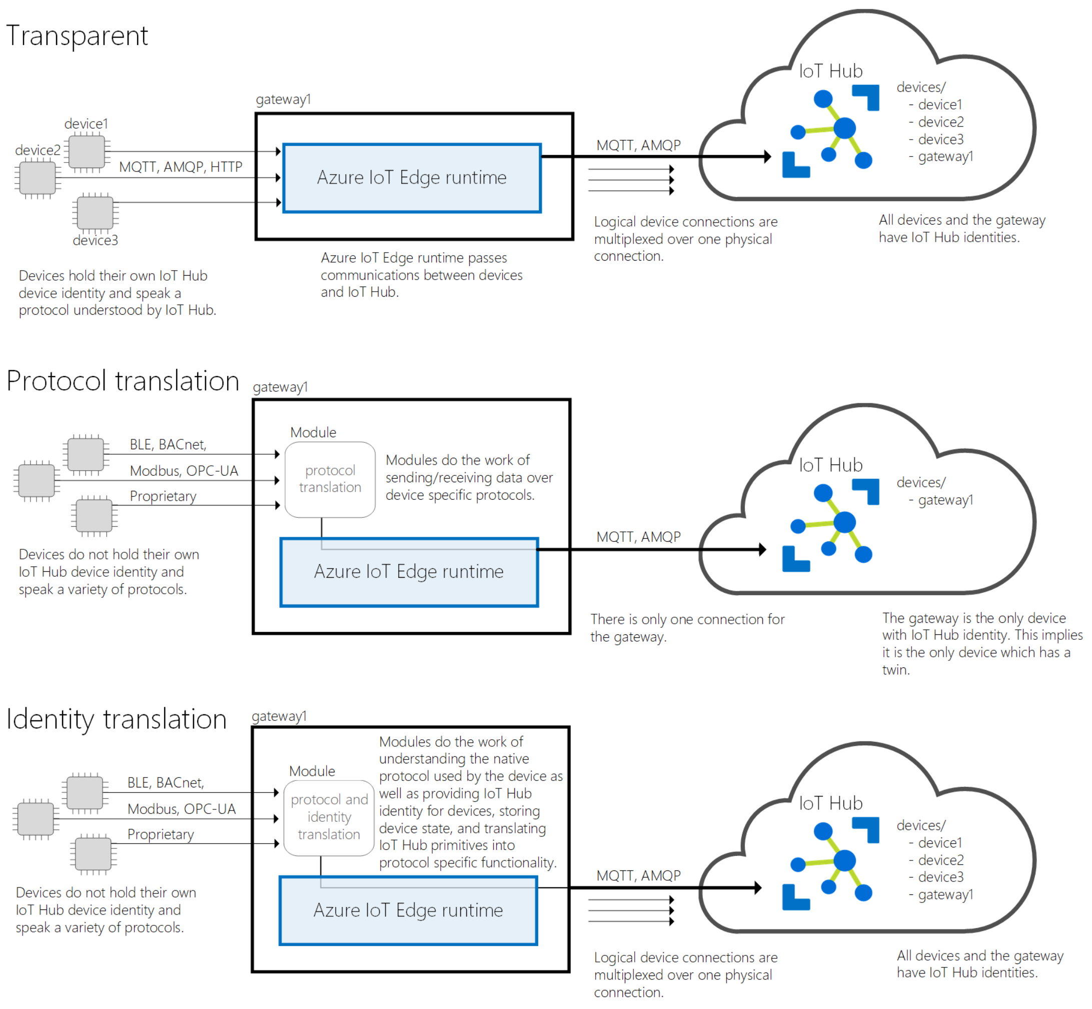
When acting as a Gateway, messages from downstream devices are routed via rules declared on the deployment manifest, these rules can be applied as default or when the messages match certain conditions or properties. After the first successful initial connection and provisioning the Azure IoT Edge Middleware is capable of operating and buffering messages from downstream devices while offline.

When operating as a Transparent Gateway the services exposed by the Edge device use TLS, this requires the downstream devices to have previously installed the Self Signed Certificate or that of a Certificate Authority that has signed the certificate presented by the Edge server when establishing connections to a particular URI. If the manufacturer does not ship the devices with the certificates, this may negate the capability to use the DPS service to completely provision these devices, requiring an intermediate manual provisioning of the certificates.

It is worth mentioning that there also exists the capability to deploy custom protocol Gateways on the Azure Cloud itself, to act as a middleman for devices capable of reaching the Cloud via Internet, but not capable of using the SDKs or the out of the box protocols. This approach although possible appears to have fallen into disuse, with IoT Edge being the preferred method.

A Deployment Manifest declares how containers are to be combined at the Edge to produce a functionality. This set of interrelated containers forms a processing pipeline, also called a workload, that is remotely provisioned. The Deployment Manifest specifies which container

Figure 2.3: IoT Edge Gateway configurations [34]


Transparent
gateway1
device1
device2
device3
MQTT, AMQP, HTTP
Azure IoT Edge runtime
MQTT, AMQP
IoT Hub
devices/
- device1
- device2
- device3
- gateway1
Devices hold their own IoT Hub device identity and speak a protocol understood by IoT Hub.
Azure IoT Edge runtime passes communications between devices and IoT Hub.
Logical device connections are multiplexed over one physical connection.
All devices and the gateway have IoT Hub identities.
Protocol translation
gateway1
Module
protocol translation
BLE, BACnet,
Modbus, OPC-UA
Proprietary
Modules do the work of sending/receiving data over device specific protocols.
Azure IoT Edge runtime
MQTT, AMQP
IoT Hub
devices/
- gateway1
Devices do not hold their own IoT Hub device identity and speak a variety of protocols.
There is only one connection for the gateway.
The gateway is the only device with IoT Hub identity. This implies it is the only device which has a twin.
Identity translation
gateway1
Module
protocol and identity translation
BLE, BACnet,
Modbus, OPC-UA
Proprietary
Modules do the work of understanding the native protocol used by the device as well as providing IoT Hub identity for devices, storing device state, and translating IoT Hub primitives into protocol specific functionality.
Azure IoT Edge runtime
MQTT, AMQP
IoT Hub
devices/
- device1
- device2
- device3
- gateway1
Devices do not hold their own IoT Hub device identity and speak a variety of protocols.
Logical device connections are multiplexed over one physical connection.
All devices and the gateway have IoT Hub identities.

images to use and how to define their messaging routes. Container images can be acquired from the Azure Marketplace, images built either by Microsoft or third parties, or be custom built. Regardless of source, images placed into a Container Registry remain private to the Azure subscription that owns them. The Deployment Manifest can be authored either via the Azure IoT Hub management portal or using the extensions available for Visual Studio and Visual Studio Code.

Once a Deployment Manifest is deployed to an Edge device no more user interaction is needed. The Edge runtime retrieves the Docker container images from the specified Container Registry. Once downloaded the images are used to create particular Container Instances that it will then place, start and begin managing automatically. If a Container Instance behaves abnormally or exits the containers are recycled and restarted. Each particular Container Instance can be customized via Module Twins, which operate in a similar way to Device Twins. Upon receiving a Module Twin update the container reinitializes itself with the new settings. The Module Twins are also manageable via the DPS service.

As previously mentioned handling device heterogeneity is one of the hardest IoT facets to tackle when deploying solutions. With no standard way for interfacing with devices, customization particular to each manufacturer is often required when developing IoT systems. So Microsoft has recognized that to implement their vision a tight coupling between Software running on the Cloud and the Edge occurs, and in solving this issue comes what is likely the most impactful and novel contribution of Microsoft to IoT, conceding successful widespread adoption by manufacturers. Inspired from the 90's Plug and Play the new initiative called IoT Plug and Play further develops the Device Twins into the Digital Twin Definition Language (DTDL).

The DTDL intends to act as a standard interface for connecting devices and specify their limitations, where identical to Plug and Play the first thing the device does upon connection is present its Capability Model, and so promote seamless device integration. This Capability Model is a JavaScript Object Notation for Linked Data (JSON-LD) schema that specifies Properties, Telemetry and Commands. This is similar to the Device Twin model presented above but typically contains more details, for example telemetry data is more than just the data readings, it also has structures that represent the structure of the data and what kind of process generates them, allowing for easier automatic data ingestion. Each device capability model can have more than one interface for easier grouping and reuse.

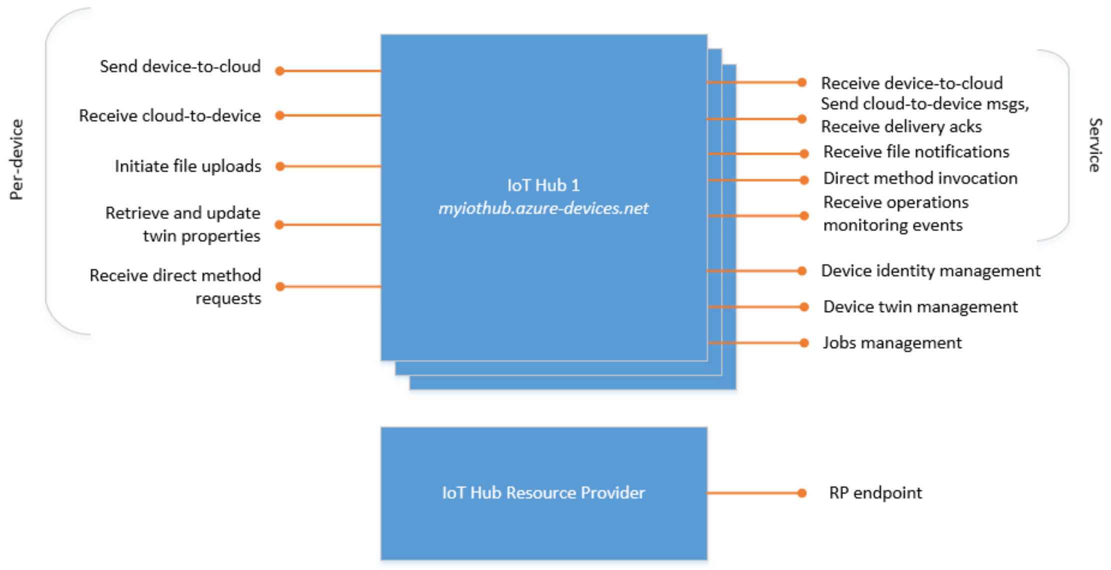
To promote adoption, tooling and cast away fears of vendor lock-in, the DTDL is Open Source and based on open World Wide Web Consortium (W3C) standards, like Resource Description Framework (RDF) and JSON-LD. This adoption has been kickstarted by partnering with sixteen IoT device manufacturers, some of which already have devices that use the technology in the market[39].

2.6.2 Azure and the requirements for IoT

When a company decides to use a product it is committing itself to the continued development of that product, such as new features and requirements, it is therefore important to consider the current development plan and analyze surrounding technologies. Similarly, historical and current success is a good predictor for continued long term support of a solution, as more resources are available to the developer to continue working and more pressure from large companies prevents discontinuation of support. Wide market adoption and the employment of standards also reduce personnel training costs, as workers with experience and know-how may be readily found on the job market. This subsection contemplates the concerns expressed previously as well as other important business considerations such as pricing models, hardware requirements and security mechanisms.

- *Communication protocols:* Due to the its Hybrid Cloud based design Azure IoT solutions rely heavily on IPs. The IoT Hub natively supports AMQP 1.0, WebSockets, MQTT 3.1.111 and HTTP 1.1 over TLS. Support for additional protocols requires the use of a middleware to perform the conversion. Available official middlewares are currently lackluster, with only OPC-UA and Modbus available, support for additional protocols is possible but requires

Figure 2.4: Default IoT Hub Endpoints [34]


Per-device
Send device-to-cloud
Receive cloud-to-device
Initiate file uploads
Retrieve and update twin properties
Receive direct method requests
IoT Hub 1
myiothub.azure-devices.net
Receive device-to-cloud
Send cloud-to-device msgs,
Receive delivery acks
Receive file notifications
Direct method invocation
Receive operations monitoring events
Service
Device identity management
Device twin management
Jobs management
IoT Hub Resource Provider
RP endpoint

custom development.

- *Node identification:* The node addressing scheme is completely extraneous to the communication protocols, instead relying on application layer addressing. A Identity Registry, located in the Cloud, is used to manage devices connected to an IoT Hub. This node identification scheme relies on the lower network layers for handling issues such as routing and mobile nodes. There is an Identity Registry per IoT Hub and it stores identifying information (such as device name and ETags[5]), authentication information (such as symmetric keys) and connection state and connection metrics. It does not contain any application metadata and it does not handle the connections themselves, merely registering whether the device is currently connected to its particular IoT Hub endpoint. A currently connected device subscribes to multiple endpoints and is addressed via them, Figure 2.4. When a new device is registered with an Identity Registry the IoT Hub dynamically creates and allocates the set of resources required for operation, such as the per-device endpoints.

  When a device connects to the Hub via an Edge device operating in a Transparent Gateway scenario the addressing and identity management is identical to when connecting directly. When operating in Protocol and Identity Translation there is no standardized method for implementing the identity management of the devices. In Protocol Translation mode only the Edge device has an identity registered on the Hub. When in Identity Translation each device has an identity registered, but their correlation and addressing is implementation dependent.

- *Hardware requirements:* For the purpose of this analysis we consider the capability to run one of the Azure IoT SDKs to be the minimum hardware requirements, although it is

[5]RFC 7232

possible to use other low-power constrained devices if using a Gateway with Protocol or Identity Translation. This SDK based approach implies that Energy saving, Real-time capabilities and other features are not handled by Microsoft and are managed by the underlying OS.

The following OSs and frameworks are supported and tested regularly[49]: Linux (Ubuntu, Debian, Raspbian), Windows, MBED, Arduino (Huzzah, ThingDev, FeatherM0), FreeRTOS (ESP32, ESP8266), .NETFramework 4.5, UWP, PCL (Profile 7 – UWP, Xamarin.iOS, Xamarin.Android), .NetMicroFramework, .NetStandard 1.3 and Intel Edison.

For devices to be able to use the SDKs effectively they must support:

- *One of:* Node.js, .NET, Java, Python Android, C or iOS.
- *TLS and SHA-256:* For security and authentication.
- *Real time clock:* For accurate timestamps in TLS connections and token generation.

- *Data volumes:* To the best of our knowledge the majority of the data volume handling techniques is identical to the ones proposed by the scientific community, using Edge to trial and process the data before it leaves the premises, the published Azure Stream Analytics module should excel at this task. Additionally for devices that support it, the Device Twins can be used to dynamically control the polling rate.

- *Security:* The IoT Hub and associated services support Active Directory policies and Access Control policies. The Access Control policies can be managed at a Hub level or on a per device basis. By default upon creation the following Hub level policies are available: allow connection to all endpoints, allow connection to any device endpoints, allow connection to any service endpoints and allow read/read and write permissions to the Identity Registry. There also also per device policies which allow only for connection to that particular device's endpoints. Device permissions can be disabled at any time in the Identity Registry and IP whitelisting-blacklisting is also available, without deleting the device configuration, leading to faster reprovisioning once the malfunctioning or compromised device is fixed.

  End to End security is mandatory, for operation a device or module must authenticate with the Hub via the credentials stored in the Identity Registry. Authentication is performed against data stored in the Identity Registry, typically a symmetric key, that is never transmitted over the wire. Optionally a X.509 thumbprint or Certificate Authority (CA) can be uploaded to the Registry. Thumbprint authentication verifies the certificate's thumbprint while CA authentication validates the certificate chain.

  Mandatory End to End security ensures that devices capable of running the SDKs can be securely authenticated and authorized, and so migrating existing deployments using an Edge device operating in Protocol or Identity Translation presents the highest security risk, for three main reasons:

  - *Custom Module development:* Custom development of modules, especially without any recommended architecture and lack of documentation, is particularly prone to introducing security errors. More Microsoft sanctioned and peer reviewed Modules for common IoT protocols would heavily alleviate this issue.

 – *Uncustomizable Access Control Policies:* Existing Access Control Policies are either device or Hub specific, there is no customization for creating a valid access to a single Hub for a subset of devices. Edge Modules must choose between the ability to support multiple devices in a single connection and assume the risk of giving a single Edge device Hub wide keys or have thousands of outgoing connections open.
 – *Constrained devices:* When supporting devices that are not capable of strong authentication mechanisms, obvious security concerns arise. This is not an issue with the Azure IoT Suite itself, security has long been an issue in IoT, but rather a side effect of supporting such devices. Again here there is a trade-off: Make an expensive migration to new Hardware and Software, with likely downtime and breaking changes, to improve security or develop Protocol Modules that will compromise Hub security, as they cannot properly authenticate devices.

- *Standards, Documentation and Development ecosystem:* There is a clear effort to contribute and avoid device vendor lock-in by developing the DTDL and supporting SDKs for many prominent programming languages[6], however it is important to consider that while Microsoft has moved heavily into Open Source in the last decade[7], their Hybrid Cloud model for IoT makes the risk of adoption of this technology not Hardware based but more identical to the adoption of any other Cloud solution.

  Aside from the already mentioned lack of documentation in what concerns to the adaptation of the Azure IoT suite to existing deployments that rely on unsupported protocols, in the developmental efforts of this dissertation the documentation was found to be comprehensive.

  The development tooling provided in the form of Integrated Development Environment (IDE) extensions offers much needed features and vastly improves the productivity when developing Modules or solutions. As with any new technology still under heavy development the development community around it consists mostly of the developers themselves, meaning that there is a lack of readily available real world use case examples. The developers presently have an active presence on both Github and Stack Overflow, which facilitates the resolution of technical difficulties.

- *Privacy, Data and Availability Policies:* Azure has a very extensively developed set of policies that any person or entity considering adoption can review. There is a thorough Privacy Statement[8], Service Level Agreements are detailed for each service individually[9] and there's also a comprehensive Compliance document[10].

[6]Node.js, .NET, Java, Python Android, C, iOS
[7]Becoming one of the largest source contributors, acquiring Github and investing in foundations such as the Open Source Initiative.
[8]https://privacy.microsoft.com/en-ca/privacystatement
[9]https://azure.microsoft.com/en-in/support/legal/sla/summary/
[10]https://aka.ms/AzureCompliance

Particularly in regards to Cloud facing protocols, such as the ones exposed by the IoT Hub, once a protocol is introduced it is assumed that it will only ever suffer breaking changes if there is a security exploit discovered[23].

- *Market adoption:* Market adoption is particularly hard to glean, we try to offer an objective analysis via Table 2.2, which lists some reported adopters which are present in Forbes list of largest public companies[37]. It is important to note Microsoft is the source of this information, via their customer stories[11] and Build presentation[22], and that it is difficult to judge how committed to the adoption of the Azure IoT Suite these companies really are. Regardless, Azure IoT already appears to have good adoption for such a recent technology and it is heavily backed by a company ranked sixteen in that same list, with large monetary and developmental investment.

Table 2.2: Azure IoT adoption by Forbes Global 2k companies[37]

Name	Rank
Exxonmobil	11
Volkswagen Group	18
HSBC Holdings	21
BMW Group	57
Siemens	64
Panasonic	298
Bridgestone	326
Ecolab	512
Rolls-Royce Holdings	733
ThyssenKrupp Group	847
Rockwell Automation	1038

- *Pricing model:* There are Basic and Standard Tiers of IoT Hub units available, each with multiple Editions. One IoT Hub may be composed of a single unit or multiple units of the same Tier and Edition. The Basic service is generally a cheaper option with a subset of capabilities of the Standard tier, with notable differences being the lack of C2D messaging and configuration twins as well as no support for Edge devices. The Edition refers mostly to the maximum daily message quota.

  There is a limit of one Free Hub per active Azure subscription and it is of Standard tier. It is meaningful to consider that while the usage of Edge devices themselves does not incur any extra charges, the Azure Container Registry that they require and DPS operations do. Table 2.3 summarizes the available options and Table 2.4 provides some context by showing the minimum Hub tier required by a a number of devices operating at different hourly polling rates[12]. One thousand devices with each sending one message every second, the bottom right corner of the table, corresponds to roughly 86 million messages per day, which is less than one third of the maximum for a Tier three Hub.

[11] https://customers.microsoft.com/en-gb/search?sq="Azure%20IoT"

[12] Number of messages sent in a fixed time interval.

Table 2.3: Azure IoT Hub unit pricing

	S0	**B1**	**S1**	**B2**	**S2**	**B3**	**S3**
Price (per month)	Free	8.443€	21.083€	42.165€	210.825€	421.650€	2108.25€
Daily message quota	8 000	400 000		6 000 000		300 000 000	
Maximum message size	0.5KB	4KB					

Table 2.4: Exemplifying the minimum Hub unit Tier required

Number of Devices \ Messages per hour	60	300	600	900	1 800	3 600
10	1	1	1	1	2	2
100	1	2	2	2	2	3
1000	2	3	3	3	3	3

Currently there is only one available type of IoT DPS and it is priced at 0.085€ per a thousand operations. DPS operations are interactions with the Identity Registry that perform actions such as device creation, retrieving a module's identity and listing registered identities.

Azure has multiple support plans available[13] which begin at roughly 25€ a month for e-mail support up to and exceeding 840€ a month for phone support, onboarding and architectural support.

This does not include the many "hidden" costs that come with adopting such a technology. Unlike Virtualization or Cloud adoption IoT implementations can come at serious downtime, consulting, and retrofitting costs. The safest approach is likely a phased adoption system, where initially an existing solution is enhanced by the ability to export data to the Cloud. Then if there is sufficient Return on Investment further commitment may be considered.

[13] azure.microsoft.com/en-gb/support/plans/

Chapter 3

Implementation

In this chapter we present the developed work, beginning with a configuration for bootstrapping the Azure Internet of Things (IoT) Edge runtime in a Virtual Machine (VM). Then we describe and present the architecture for a protocol translation middleware intended to be used as a Big Data connector between variable protocols and the Azure IoT Hub. Subsequently the Critical Manufacturing's (CMFs) Manufacturing Execution System (MES) Connect IoT Graphical User Interface (GUI) is exemplified and its operation explained to provide the reader with context for the adaptation of the middleware whose integration and operation is explained next. Lastly we present the steps followed in the preparation of a Dataset of a manufacturing process composed with data of obtained via IoT.

3.1 Azure IoT Edge Setup

VMs are a widely used enterprise technology [17], typically due to facilitating continuous integration and delivery, while giving developers identical and reproducible library environments and tech stacks. In this thesis we take advantage of VMs to facilitate a faithful and exact reproduction of the software environment used to derive the results, by creating a Vagrant configuration that automatically creates and configures a VM with identical Azure IoT Edge settings.

Vagrant is a tool for building and maintaining portable software development environments, with a focus on automation. Capable of software provisioning and configuration, Vagrant uses widely accepted VM tooling such as VirtualBox, VMWare and AWS to achieve the reproducible portable work environment we desire. As per the recommendation of Langer and French [30] VirtualBox is used as the VM environment.

It is important for the validity of the results to acknowledge that VMs could add some overhead to particular computations and may impair the guest Operating System (OS). The exact figures will be heavily dependent on the VM environment used, hardware and other confounding variables such as paravirtualization. Paravirtualization is a set of techniques where usually the VM hardware is not fully emulated and the guest OS is instead provided with a hardware interface that corresponds more closely to the physical hardware, allowing some computations to run natively on the 'bare metal' Central Processing Unit (CPU), minimizing overhead and improving system performance. However, such eventual penalties are negligible for most CPU and memory use cases with exception of disk I/O intensive workloads and floating point sensitive applications [16][30].

The Vagrant project setups an Ubuntu 16.04 64bit VM and installs IoT Edge, via a script that that automates the steps laid out in the official documentation[1]. Then it provisions the VM with configuration files contained in the projects folder. The configuration files provided are a sample config and development certificates. For use a valid IoT Edge connection string must be registered[2] and placed in the *config.yaml* file. The certificates were generated as indicated in the documentation[3] and are obviously insecure and should only be used for testing purposes. Resource limitations are configured on the projects *Vagrantfile* and can be tuned there[4].

3.2 Translator

All programs and work done in this Chapter adopt TypeScript, due to the expected integration of the developed work with CMFs Connect IoT, which mostly consists of TypeScript. The Node.js Azure IoT Software Development Kit (SDK) which we used is also built in TypeScript.

TypeScript is an Open Source programming language developed and maintained by Microsoft[5]. It is a statically typed superset of JavaScript, that aspires to provide debugging and mitigate development errors that occur when using the latter. Since Javascript code is also valid TypeScript code, there is full compatibility with the already very well established JavaScript libraries and Node.js. TypeScript can also be transpiled to runnable JavaScript code. This is achieved via the *TypeScript Compiler* which can be fed a configuration file that specifies useful options such as the target ECMAScript[6] specification to adhere to and whether to allow variables with no type declaration. This direct compatibility with JavaScript also allows for the use of the Node Package Manager (NPM), which manages packages and the specific versions to be used with the program. The combination of the *TypeScript Compiler* configuration and the NPM package configuration also helps provide a reproducible environment.

Microsoft has made available multiple IoT quick start examples[7][8]. We began by adapting these examples to TypeScript and, as intended, it provided a good starting point to begin

[1]The bootstrap script uses Microsoft repositories and keys that may change in the future.

[2]docs.microsoft.com/en-us/azure/iot-edge/quickstart#register-an-iot-edge-device

[3]docs.microsoft.com/en-us/azure/iot-edge/how-to-create-transparent-gateway

[4]For the bare minimum iotedge service to run properly the Vagrant box requires about 1 GB RAM. Running it on more modest settings requires that the environment variable OptimizeForPerformance is set to false.

[5]github.com/microsoft/TypeScript

[6]A scripting language ISO specification.

[7]docs.microsoft.com/en-us/azure/iot-hub/quickstart-send-telemetry-node

[8]github.com/Azure/azure-iot-sdk-node/tree/master/device/samples

understanding how to practically implement a solution, however these examples assumed the devices themselves would run the SDK, and these simple examples were not in line with CMFs consideration for an initial parallel data connector and the desire to test performance with many simulated devices. As per the recommendation of the Co-Advisor the exploration of the Hub and Edge practical implementations was continued by developing a modular program, separate from Connect IoT, that could later be easily adapted. At this time only the C# SDK supports device connection pooling, which is important when handling thousands of devices as it avoids the need to have thousands of outgoing connections.

The expected behaviour of this solution essentially amounts to that of connecting two disparate protocols via common interfaces, which allow the exchange of messages between them while abstracting configuration and implementation details from each other. Additionally this solution does not use specific Edge Module mechanisms and environment variables, so that it can be used either as its own standalone solution or inside a Module.

In this project, components that implement protocol details are termed *Drivers*. Drivers realize a specific interface which defines methods to initialize and stop the Driver, send a message and add or remove devices whose messages are to be monitored so they can be parsed, transformed and transmitted. Drivers also implement an observer pattern, emitting a particular event when a message has been transformed into a common format, so that registered event listeners receive the message and can forward it to its destination. There is an additional component, named a Gateway, that can optionally be used to be used to link two Drivers. The Gateway implements methods that construct the Driver objects based on the configurations received and connect the Drivers event listeners and event outputs to correctly exchange messages from one protocol to the other. There are also methods to individually register and cancel listeners for arbitrary events on each Driver and a method that connects the Drivers to each other.

When naming the Drivers themselves we generally refer to Drivers that connect to the Hub and Edge as Upstream and those that connect to other, usually on premise, services as Downstream, a naming scheme lifted from Azure IoT recommended architecture[34]. This naming distinction also serves as a suggestion that a Downstream Driver is expected to be connected to an Upstream one, although due to using the same design any two properly configured Drivers realizing the interface and concepts described above are able to exchange messages seamlessly. The Upstream Drivers detailed here were the ones eventually adapted into the Connect IoT Task described on the following section.

3.2.1 Upstream Drivers

Two separate Upstream Drivers were developed. The first, and simpler of the two, connects to the IoT Hub or Edge Hub using either the Advanced Message Queuing Protocol (AMQP) or Message Queuing Telemetry Transport (MQTT) protocol and device policy keys, this means that for each device added a new connection to the Hub is opened. This Driver uses the higher level and well documented SDKs connection clients to create these connections, identically to the quick start examples. This Driver can also receive a Certificate to use to authenticate and connect with an Edge server. As CMFs customers often operate under strict firewall and network control mechanisms the second Upstream Driver provides connection pooling and uses the lower level AMQP packages, generally used only by Microsoft's own developers and which posses nearly no documentation, to create a single connection capable of specifying the target endpoint on a message by message basis. A single Hub wide policy key with access to all device endpoints is used to generate an access token. When a device is registered on this Driver its receiver link is monitored for messages to emit and when sending messages the same connection is used, by changing the destination endpoint.

3.2.2 Downstream Drivers

The Downstream Driver developed was used only for testing scenarios, as no custom authentication mechanism is placed between the devices and the Gateway. It is not a major concern as the adoption to MES uses the already developed MES Equipment Integration Module to support Downstream protocols. The MQTT Downstream Driver connects to a specified server and processes the messages received, which are of JavaScript Object Notation (JSON) format, that always include a *deviceId* used for correlating a message to its source. These messages are emitted so listener Drivers can parse and forward them.

3.3 MES Adaptation

CMFs MES has developed support for many protocols[9] and has customers that use them in their IoT solutions. Some customers only perform maintenance upgrades once a year, at the scale they operate in any downtime implies a large financial cost. The adaptation to the MES service is intended to provide an option that acts parallel to already deployed solutions as a Big Data connector, forwarding data to Azure with context and listening for responses, where it can processed in a completely complementary and non breaking fashion. This Proof of Concept (PoC) consists of a Connect IoT *Task* to be used in the *Workflow Designer*. Data is received from other MES tasks and based on the configurations a message is built and forwarded to the Azure IoT Hub, the optional use of an Edge server is supported.

3.3.1 Workflow designer

CMFs MES is a massive proprietary system, the detailed analysis of which is outside the scope of this Thesis. This section introduces the main workflow components relevant for situating the reader and facilitate the understanding of the developed work. The developed work is detailed in the following Section 3.3.2.

[9]Such as SECS/GEM, OPC-DA, MQTT, BLE, TCP and others.

The extensively developed automation integration of the MES is designed as a set of workflows that can be manipulated and configured graphically with a drag and drop logic. The purpose of this system is to provide a visual representation of automation logic and processes, that can be easily understood and configured. The workflows configured are automatically launched and managed by MES subsystems at runtime. An update on these pages and its configurations triggers updates on the deployed services, without code changes or manual service management.

Figure 3.1: Workflow Designer

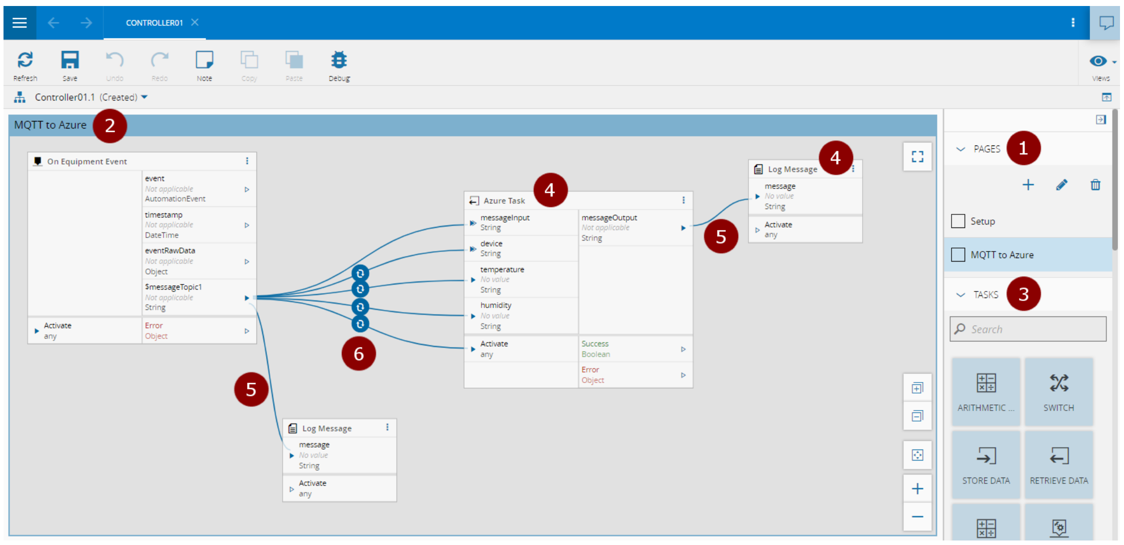


Figure 3.1 shows the User Interface (UI) and indicates the following components:

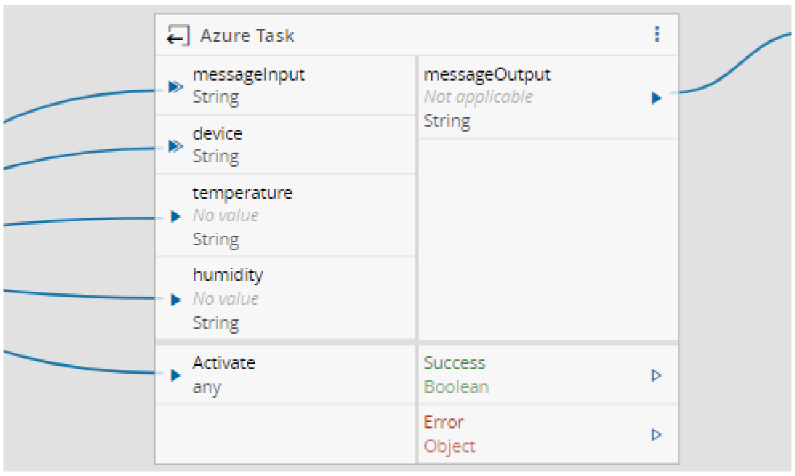
1. *Page List:* This is where Pages can be created, managed and selected for editing.
2. *Active Page:* Pages are used to provide clear organization, each page typically represents a particular functionality and has its own individual workflow. This is the main area of user interaction with the Workflow Designer, where the workflows are represented and can be edited.
3. *Task List:* Tasks available for use are listed here and can be dragged and dropped into the Active Page.
4. *Tasks:* Tasks represent existing implementations of some business logic, such as a function or class. Each task has its own inputs/outputs and configurations and its behaviour depends upon them. Most tasks have a dedicated *Activation* input and *Success* and *Error* outputs.
5. *Links:* A Link is a connection between an output and an input, typically of two separate tasks. The same input may receive data from multiple outputs and the same output can be connected to multiple inputs.
6. *Converters:* A converter is placed in a Link to perform some data transformation operation. This is used in cases where the type of an output does not match the expected type of an input.

3.3.2 Developed Task

CMF has a set of Yeoman[10] scaffolding templates available to fast track the development of new tasks, converters or protocol drivers of Connect IoT. These templates also set linting and compilation rules for the Typescript compiler. This scaffolding was used to generate the base structure that the Translator logic was adapted to, with the vast majority of the effort of this adaptation on ensuring that the already developed Driver Classes proved resilient to Network connection issues, vital for long-term running services, and on creating a sensible UI with validation and contextual settings. The Driver Classes are provided to the Task via Dependency Injetion. Settings are said to be contextual when they change their required state or become hidden depending on the Azure connection type selected, as each method has slightly different requirements. Settings persist once saved, even if changing between multiple connection types causes them to be hidden.

The task has three default inputs and custom inputs, as can be seen on Figure 3.2, and its behaviour changes depending on the inputs linked. In this example temperature and humidity are custom inputs defined in the settings.

Figure 3.2: Azure Task



A message can be sent directly as is received on the *messageInput* input or it can be built from multiple custom inputs. *messageInput* strings received are expected to be valid JSON, if it is not an Error is returned. When using both the *messageInput* and custom inputs, custom inputs add new entries or override entries with the same key in *messageInput*. The *device* input identifies the source of the message, a message is sent through an established connection to the Hub defined on the task settings, if the device specified is not configured no open Hub connection exists and so the attempt to send a Device to Cloud (D2C) message will return an Error. The trigger to build and send a D2C message occurs whenever the *Activate* input receives any value. The *messageOutput* output emits a message whenever one is received from the Cloud in one of its open connections. Tables 3.1 and 3.2 summarize the inputs and outputs.

[10]yeoman.io

Table 3.1: Azure Task Inputs

Name	Type	Description
messageInput	String	D2C message to send.
device	String	Device to use as message source.
<custom list>	<custom>	Input values to build a message with.
Activate	Any	Retrieve all input values.

Table 3.2: Azure Task Outputs

Name	Type	Description
messageOutput	String	C2D message received.
Success	Boolean	*True* value emitted when a D2C message is successfully sent.
Error	Error	Errors that occurred during task operations.

The *General* settings tab, Figure 3.3 and Table 3.3, allows the configuration of the task instance name and the Hub connection type. Different options are shown based on the *Azure connection Type selected*:

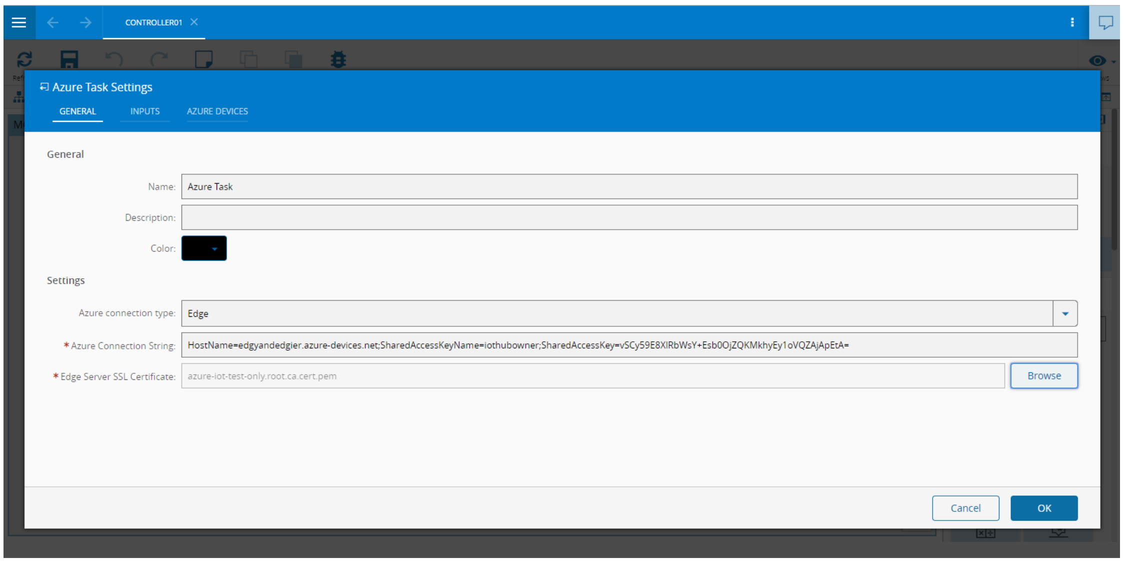
- *Direct:* The *Azure Connection String* is displayed but not required, per device connections are created using the authentication provided in the *Azure Devices* tab.
- *Multiplexed:* The *Azure Connection String* option is required, as its authentication is used to build a single AMQP transport connection to the Hub, device names in the *Azure Devices* tab are used for matching MES device names to Hub device names.
- *Edge:* The *Azure Connection String* identifies the Edge host and the device names are used to build a connection string that passes through it. The Edge certificate required is added via a file upload. There is basic format validation when uploading, which checks for the expected first and last lines format.

Table 3.3: General Task Settings

Name	Type	Contextual	Description
Name	String	No	Name of the Task instance.
Description	String	No	Description of the Task instance.
Connection Type	ConnectionType	No	Connection type for the Hub. Direct, Multiplexed or Edge.
Connection String	String	Yes	Connection String for the Hub. Required in Multiplexed or Edge mode.
Edge Certificate	File	Yes	SSL Certificate required to establish connection to the Edge server.

The *Input* settings tab, Figure 3.4 and Table 3.4, allows for the creation of custom inputs. Inputs are added or removed on the left column. Input names and expected types can be set on the right.

Figure 3.3: General Task Settings



General

Name: Azure Task

Description:

Color:

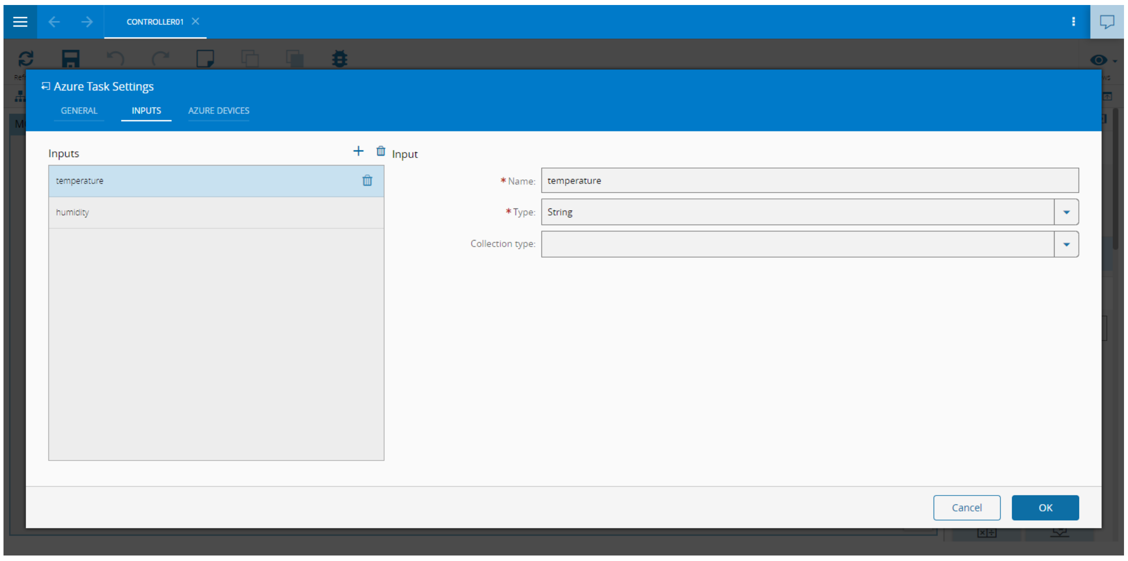
Settings

Azure connection type: Edge

* Azure Connection String: HostName=edgyandedgier.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=vSCy59E8XlRbWsY+Esb0OjZQKMkhyEy1oVQZAjApEtA=

* Edge Server SSL Certificate: azure-iot-test-only.root.ca.cert.pem

Figure 3.4: Azure Input Settings



Inputs

temperature

humidity

Input

* Name: temperature

* Type: String

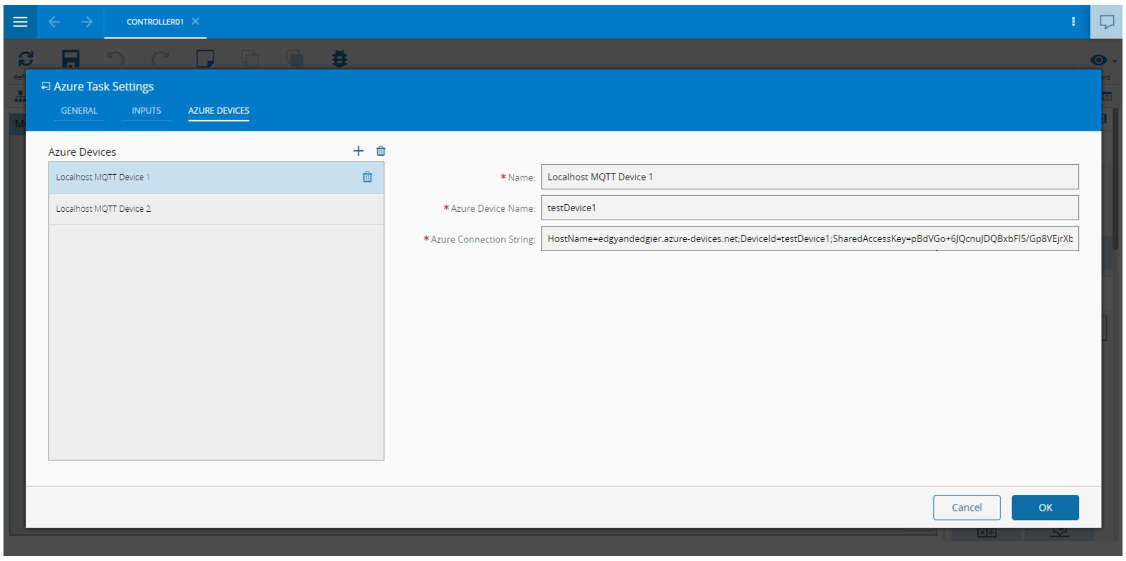
Collection type:

Table 3.4: Input Task Settings

Name	Type	Description
Name	String	Name of the input to create.
Type	'Stringable' Types	Expected data type.
Collection Type	None, Array, Map	Collection Type of data.

The *Azure Devices* settings tab, Figure 3.5 and Table 3.5, allows for the registration of devices. Devices are added or removed on the left column. Device identifiers and connection strings can be set on the right.

Figure 3.5: Azure Device Settings



Localhost MQTT Device 1
Localhost MQTT Device 2

* Name:	Localhost MQTT Device 1
* Azure Device Name:	testDevice1
* Azure Connection String:	HostName=edgyandedgier.azure-devices.net;DeviceId=testDevice1;SharedAccessKey=pBdVGo+6JQcnuJDQBxbFI5/Gp8VEjrXt

Table 3.5: General Device Settings

Name	Type	Contextual	Description
Name	String	No	Name of the device to register.
Azure Device Name	String	Yes	Name of the device on the Azure Identity Registry.
Azure Connection String	String	Yes	Connection String to be used when connecting in Direct or Edge mode.

3.4 Dataset Preparation

CMF graciously provided access to an extensive dataset of a manufacturing process to potentially be used in our experiments, for customer confidentiality reasons we were presented with the choice of disclosing its originating company and concrete process that the readings are derived from, or preparing the dataset carefully for publication, ensuring no identifiable data remains. The remainder of this section justifies why it was decided to prepare the dataset for publication and details the pre-processing steps taken.

After initial review of the dataset we determined that it could likely be of use to the field of Artificial Intelligence (AI), as it is an example of a multi-relational dataset. Multi-relational datasets differ from the classic two dimensional dataset table representation, where rows are instances of data readings and columns are variables, as there may be relationships between instances, in addition to variables.

In the case of the process that generates this data eight different readings are taken each time a particular tool is used. Eventually once a tool begins underperforming, by some decision making process unbeknownst to us, it is retired and therefore does not again again appear in the dataset. We believe that this dataset may be used to estimate and predict tool longevity, as it likely presents time dependent covariates as such be of use to the research of multilevel survival analysis[44] or predictive maintenance models.

The dataset was initially extracted from a Structured Query Language (SQL) production Database (DB), already filtered to only include data relevant to a specific production process for a particular product. At this stage we found much of the data to be redundant throughout entire columns and some that would not be of use without further context for the system. The purpose of data preparation is to ensure that relevant information is treated through data transformations and without a loss of actionable information so that it may later be used by Data Mining (DM) tools.

Variables that possess a single value through the entire dataset can be safely removed, as they are by definition not variable and will not have an impact on the predictions, however whenever removing other columns we carefully deliberated the possible impact of doing so and erred on the side of caution when uncertain. Of the initial forty columns fourteen were removed for having static values. One column was removed for being, in this case, an exact duplicate of another column. Eight different columns existed that represented the time a event happened, having investigated the reasoning behind storing each of them we concluded that six were different representations of the time the data collection reading operation finished and the remaining two were representative of entity creation and update times, attributes specific to MES operation and of no relevance to this context. Of these time representing variables we kept the one that represented the operation end time with the highest precision. There were additionally ten more columns that represent foreign keys to charts, service history and other operations specific to the MES that were also removed.

In short after handling redundant and irrelevant variables we are left with eight columns in the dataset, Figure 3.6 presents them, their meaning and data types. The only missing data on the dataset at this point are 176 entries of the variable *DataPoint2*, we decided against dropping or transforming these missing values as their absence might be an indicator of something that the prediction models can use. We did not perform any transformation or filtering of outliers or discretization of continuous variables.

At this point we performed the anonymization of the dataset. All the values of the *ToolId*, *Parameter*, *Machine* and *Process* variables were changed to lose any relation to the MES that originated them, as an example the first tool to appear will be referred to by the key of 0, the second tool to appear as 1 and so on. The OperationEndTime was also sanitized, being changed from the UNIX epoch format to instead represent the seconds from the first operation that occurs in the dataset, in this way keeping the relation of time between different readings but erasing the actual real world dates. The *DataPoint1* and *DataPoint2* variables suffered no alterations.

Upon further analysis we concluded there was still redundant data, as each complete reading

Table 3.6: Intermediate dataset variables

Name	**Type**	**Description**
OperationEndTime	UNIX Epoch time	The time when the reading operation concluded.
Parameter	Numeric	A categorical variable of the data collection method that generated this reading. There are eight different collection types in the dataset.
ToolId	Numerical Key	The tool used. It's value is unique to each different tool in the dataset.
Machine	Numeric	A categorical variable, representing the machine that used the tool. It's value is unique to each different machine in the dataset.
Process	Numeric	A categorical variable, representing the process that used the tool. It's value is unique to each different process in the dataset.
DataPoint1	Numeric	A concrete value for a reading.
DataPoint2	Numeric	A concrete value for an error metric associated with the process that generated the value present on DataPoint1.

event corresponded to eight rows, that is each time an operation concluded eight different metrics were gathered and their values stored in a separate row. Our final structural transformation to the dataset was to make each row representative of an entire operation, removing the *Parameter* variable and aggregating its data readings on its own columns, Figure 3.7. It is important to mention that due to the absence of keys unique to each set of eight readings this transformation was only possible as the dataset presented an orderly storage of readings, this was corroborated by analysing the timestamps and the order of the parameters read, which was always conforming to a repeating pattern.

Table 3.7: Final dataset variables

Name	**Type**	**Description**
OperationEndTime	Numerical	Difference in seconds from the first operation in the dataset.
ToolId	Numerical Key	The tool used. It's value is unique to each different tool in the dataset.
Machine	Numeric	A categorical variable, representing the machine that used the tool. It's value is unique to each different machine in the dataset.
Process	Numeric	A categorical variable, representing the process that used the tool. It's value is unique to each different process in the dataset.
P1DataPoint1	Numeric	A concrete value for a reading of parameter one.
P1DataPoint2	Numeric	A concrete value for an error metric associated with the process that generated the value present on *P1DataPoint1*.
P2DataPoint1	Numeric	A concrete value for a reading of parameter two.
P2DataPoint2	Numeric	A concrete value for an error metric associated with the process that generated the value present on *P1DataPoint2*.
...	...	...
P8DataPoint1	Numeric	A concrete value for a reading of parameter eight.
P8DataPoint2	Numeric	A concrete value for an error metric associated with the process that generated the value present on *P1DataPoint8*.

Chapter 4

Results and Analysis

In considering the adoption of a new technology it is important to be conscious of its actual performance in a real environment and how far it strays from its goals. This chapter contains the definition of experiments and the evaluation metrics used, beginning by delineating the methodology followed and the possible drawbacks of performing the study in such fashion. Afterwards it presents the concrete results obtained via the practical experiments and their detailed analysis, concluding with a reflection of how the results obtained and the challenges found along the way clash with the requirements for stability and uptime adopters require.

4.1 Methodology

The experiment designed intends to measure the performance of the Azure Internet of Things (IoT) Hub and Azure IoT Edge in controlled scenarios so the recommendation for adoption in Industrial Internet of Things (IIoT) systems can be based on an estimation of real world efficiency of the system alongside with the architectural review already performed. Although the Edge devices support complex use cases, with data processing and streaming Modules, the pipeline tested is purposefully simple, where a plain routing scenario is used, messages received are forwarded to the Hub without any Modules parsing them or any custom routing rules applied. This approach intends to test that the minimum viable product required for basic IoT adoption exists and relegates the analysis of more complex operational models for future work. This study also focuses on testing the behaviour of the system over relatively short periods of time, possible future work exists in ensuring that the system proves stable over long periods of continuous use.

Initially it was deemed relevant to test the system with real IoT data, along with a synthetic benchmark of simulated data. Critical Manufacturing (CMF) gracefully provided a dataset of a real world manufacturing production process which after preprocessing, Section 3.4, was evaluated and analysed for metrics such as message frequency and message size. However upon concluding this preliminary analysis and some preliminary testing of the IoT Hub it was deemed that the frequency and size of the messages would not be sufficient for a proper benchmark and by recommendation of both Advisors this idea was discarded, with the final benchmarks performed exclusively with generated data. All experiments occur in a paid S1 (Standard tier, Edition One) IoT Hub, priced at €21.083, with a limit of 400k daily messages and a maximum intake higher than 100 messages per second.

The connection methods tested in this examination use the Driver implementations previously described on Section 3.2, using TypeScript, Node.js and the Azure IoT Node Software Development Kits (SDKs). The different methods being tested are the direct Advanced Message Queuing Protocol (AMQP) and Message Queuing Telemetry Transport (MQTT) options which create a new connection per device to the Cloud using each device's access key, the Edge MQTT which operates similarly but has the connection routed through the Edge device and finally the Translator AMQP method which establishes the connections through a middleware using a single Hub wide key. For an assessment of why these methods were selected and other concerns see Section 4.3.

We consider a *run* to be an experiment which has a determined frequency of messages and number of devices. The experiments simulate 100 different devices operating in round robin to generate a total of 100 Device to Cloud (D2C) messages per second. Each run consists of 1000 D2C messages and was performed 5 times to diminish statistical variability, with a waiting period of two minutes between runs and the IoT Edge runtime being restarted for each new run. Each particular message had a body of roughly 100 bytes, with the maximum supported being 245K bytes. For each particular benchmark run the Round Trip Time (RTT) of messages was gathered by inserting a timestamp into each message's body right before it was sent as a D2C message. Once the messages arrive at the Cloud a registered listener on the Cloud device endpoint triggers the routing of messages back as Cloud to Device (C2D) communications. Once messages arrive back at the registered consumer the program extracts the timestamp and subtracts it to the current time to obtain the RTT. As we are measuring the full RTT of messages we are placing identical levels of strain on the testing of D2C as well as C2D messages. To avoid the need of date-time synchronization both the source and destination of the simulated data is the same physical computer. The collected data retains natural temporal ordering, allowing us to analyse it as a time series, a common concept in statistics and signal processing. Table 4.1 lists the published Node Package Manager (NPM) package versions used in testing.

Table 4.1: Node package versions used

Name	Version
@azure/event-hubs	1.0.8
@types/async-lock	1.1.1
@types/node	10.14.7
azure-iot-amqp-base	2.2.9
azure-iot-common	1.9.9
azure-iot-device-amqp	1.9.9
azure-iot-device-mqtt	1.9.9
azure-iothub	1.9.9
events	3.0.0

The Edge device was configured using the Vagrant configuration outlined in Section 3.1 of the previous chapter and the hardware used consisted of a Desktop PC with an Intel Core i5 6600K Processor (4x 3.5 GHz) and 16 GB DDR4. The devices were simulated on a Laptop PC with Intel Core i7 3720QM Processor (4x 2.6 GHz) and 8 GB DDR4 RAM. All network connections used a wired network, 100/100 Mbps via category 6 ethernet cables, to further diminish the entropy of collisions and delays common to wireless networks. In the case of Edge benchmarks performed, hardware and network usage was also monitored via *collectd*, which is a well established method for collection of system statistics. The collected statistics were Central Processing Unit (CPU) and memory usage, on the hardware front, and on the network side the volume of messages sent and received.

4.2 Results and Analysis

This Section of the dissertation presents the collected data and interprets the results. We begin by applying graphical techniques to better describe the results and allow the formulation of conclusions. As referred previously the testing of each different connection type was performed five times, the data was then aggregated as the averages of the runs at each particular time. This transformed data was used to plot graphs with confidence intervals of 95% and calculate the grand mean and grand standard deviation for each connection type, present in Table 4.2, which also shows the absolute minimum and maximum times obtained for each connection type. It is readily apparent from these metrics alone that the the Edge connection method is clearly underperforming for time sensitive applications, even under ideal conditions.

Table 4.2: RTT results (milliseconds)

Scenario	Minimum	Grand Mean	Maximum
Direct AMQP	54	89.01 ± 21.93	443
Direct MQTT	56	90.81 ± 31.83	1010
Translator AMQP	56	94.23 ± 28.44	338
Edge MQTT	75	2564.37 ± 1547.44	10345

Figure 4.1 is a visualization of the grand mean and standard deviation via box plots for each of the examined connection types and further displays the inadequacy of the Edge MQTT method. Due to the difference of scale in the results, for ease of visualization, the Edge MQTT method is displayed separately.

Observing the line charts for the Direct AMQP, Figure 4.2, and Translator AMQP, Figure 4.3, connection methods, after the initial messages the connection becomes stable, handling the data volumes without showing an increasing RTT trend or other apparent instabilities.

For the Direct MQTT, Figure 4.4, mode whoever there are very large, unevenly placed, latency spikes. Inspecting each of the runs plotted individually, Figure 4.5, we can see that the latency spikes match to occurrences on run one, two and three. It is unlikely, as no other connection type appears to suffer from this issue, but possible that this was caused due to local

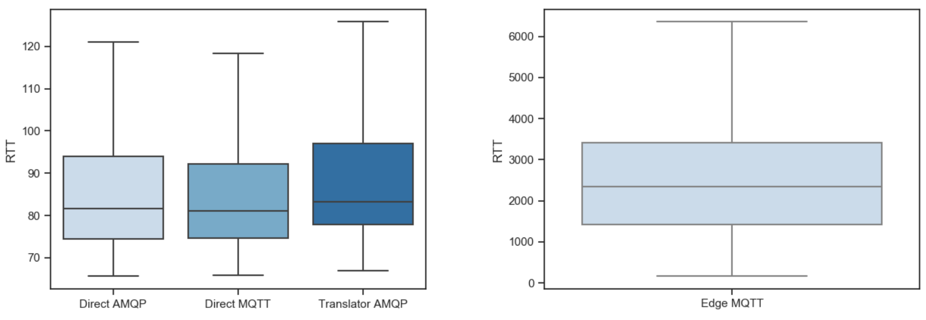


Figure 4.1: Box plots per method

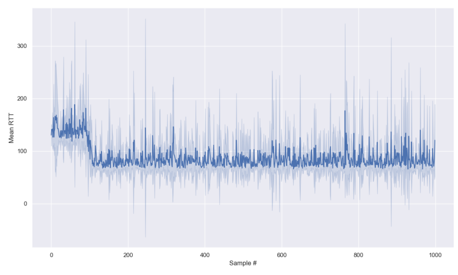


Figure 4.2: Direct AMQP

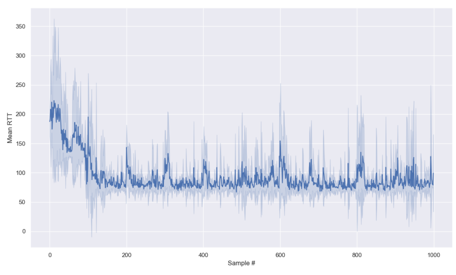
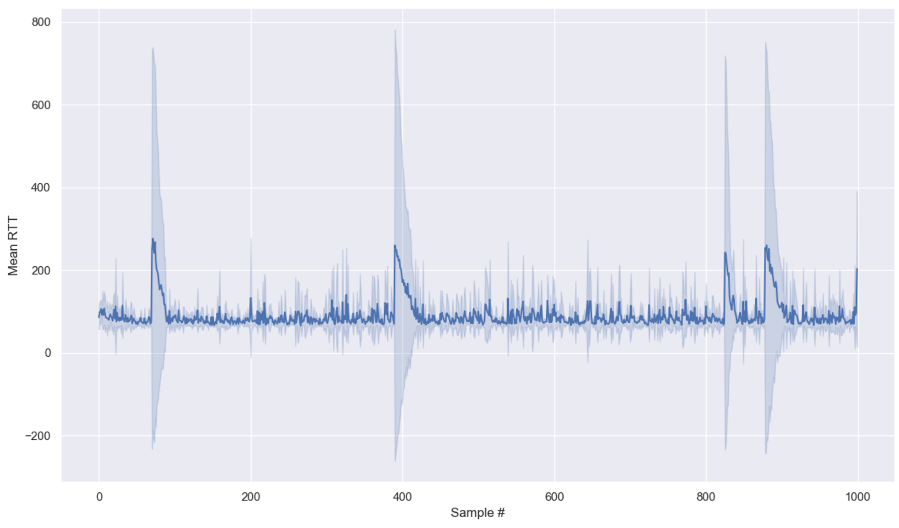


Figure 4.3: Translator AMQP

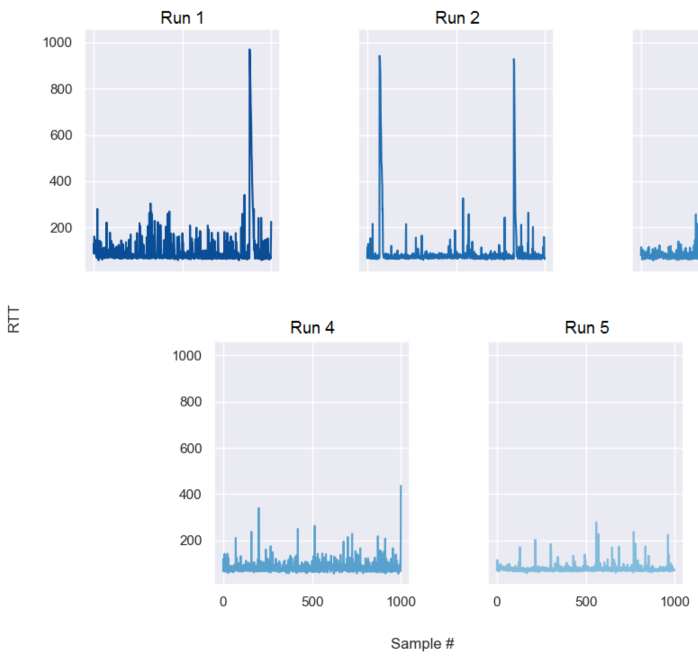
network or internet provider issues. Even with the present latency surges this method also presents a stable option for D2C connections.

Figure 4.4: Direct MQTT



By visual inspection and in comparison to the results of the other methods the Edge MQTT method clearly displays a very different behaviour, Figure 4.6. The data does not appear to be

Figure 4.5: Direct MQTT per Run



random and possibly has factors of autocorrelation and seasonality. A lag plot is a well known graphical method for evaluating a series randomness, with a random series not presenting a recognizable structure and being plotted similarly to a shotgun pattern[38]. Inspecting Figure 4.7 we can conclude that the data has strong positive autocorrelation due to the tight clustering of points alongside the diagonal. The data also presents a substantial amount of outliers.

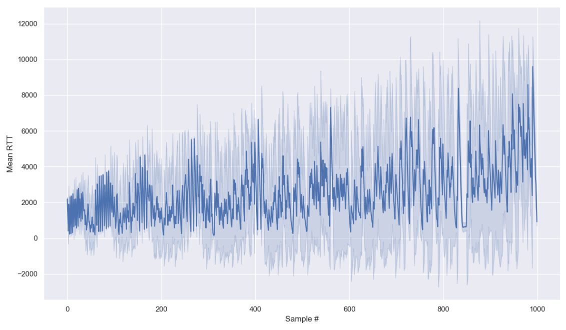


Figure 4.6: Edge MQTT

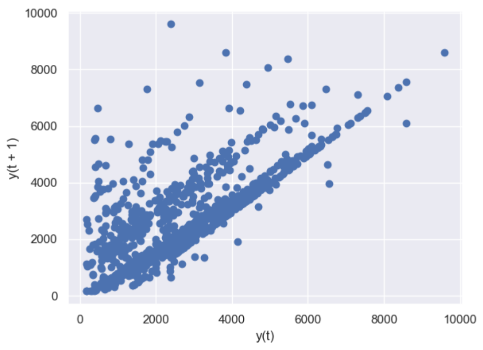
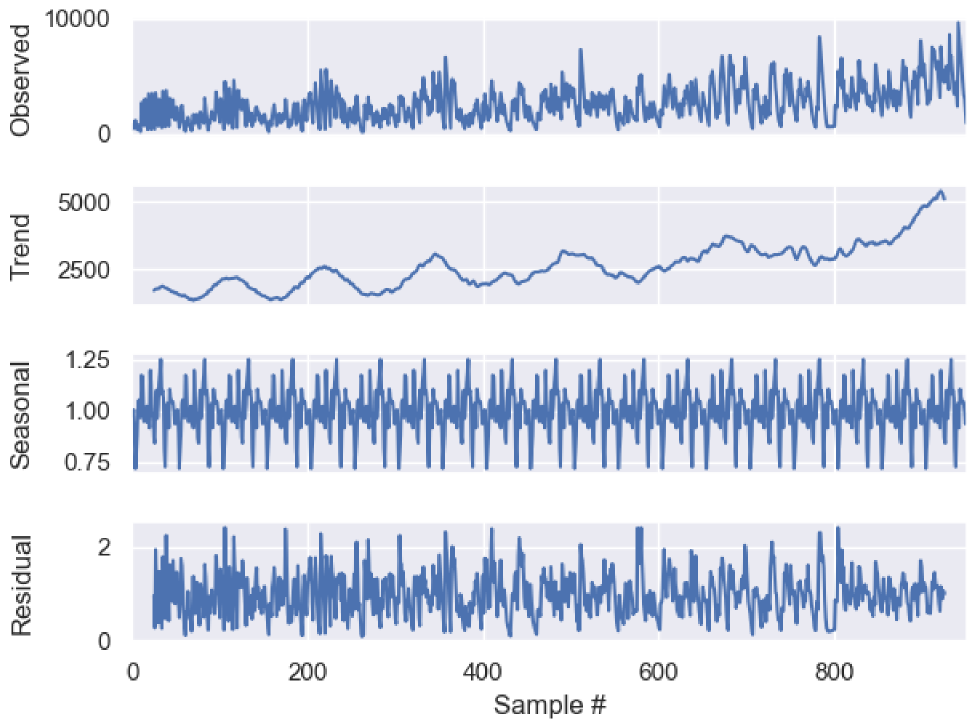


Figure 4.7: Edge MQTT Lag plot

In order to help the analysis of a time series it is often useful to decompose it, removing components that are unhelpful for conclusions, such as the seasonality itself and noise. In our case we are particularly interested in determining whether the trend is increasing, for the reason that an upward trend is an indicator that the Edge device is not able to route the messages to the Cloud faster than it is receiving them. With an upward trend ongoing routing operations will eventually cause completely unreasonable delays and possible loss of messages. The classical decomposition methods, additive and multiplicative, are very well established and have existed for nearly a century. By analysis of Figure 4.6 we can determine that the seasonality corresponds to roughly every fifty samples. We decomposed the time series using the multiplicative method with a seasonality of fifty and the resulting trend shown in Figure 4.8 shows clear signs of growth, with a trend difference of over two and half seconds delay in the RTT of the first messages being sent and the last ones.

Figure 4.8: Edge MQTT decomposed



Inspecting the collected resource usage metrics for the Edge device itself, Table 4.3, we can confidently determine that the bad performance of the Edge MQTT method is extremely unlikely to be caused by hardware resource constraints, with the CPU being mostly idle, the memory used barely changing and the network usage being orders of magnitude less than that provided by the service provider and supported by the local wired network.

Table 4.3: Edge device statistics

Metric	**Measurement**
CPU system load	$0.60\% \pm 1.15\%$
CPU user load	$0.56\% \pm 0.80\%$
CPU idle	$98.87\% \pm 2.66\%$
Memory used	$424.37Mb \pm 4.0Mb$
Network in	$533.4Kb \pm 30.4Kb$
Network out	$590.0Kb \pm 34.4Kb$

In conclusion all of the methods tested that do not use the Edge device operate at a reasonable latency and connection stability for short bursts of intensive operation for one hundred devices. Looking at the data there does not appear to be significant difference between using the Direct AMQP method, which uses a separate connection per device and thus multiple network sockets, and the Translator AMQP, which uses a single connection for all devices, meaning that the usage of the middleware is unnecessary unless operating under strict firewall rules or while supporting custom protocols, as may be the case in some industrial environments. Albeit Microsoft's vision for the Edge is very promising the MQTT option in these circumstances does not, at this point, prove stable enough for widespread usage.

Comparing the results obtained for direct methods to the other available performance study focused on Azure IoT by Forsström et al.[18] our results were significantly better. The study obtained $770ms \pm 330ms$ using HyperText Transfer Protocol Secure (HTTPS). This may be due to performance improvements in the IoT Hub in the meantime, however it is more likely that the large RTT difference is caused by the data being routed back by the Hub itself instead of via Azure Servicebus Queue as is the case in Forsström's study.

4.3 Challenges and additional considerations

The formulation of the study described and followed is not the same as was initially planned, this was due to unforeseen circumstances with the behaviour of the Edge device and SDKs which were discovered during preliminary testing for the experiments and during implementation.

The initial experiments involved varying amounts of devices and messages per second, however at this point in testing, Version 1.0.6.1, the Edge device was unable to route messages from a single device operating at ten round trip messages per second, causing the queue to infinitely grow. A substantial quantity of the time allocated to the study phase was dedicated to ensuring that this was not caused by Edge device misconfigurations or incorrect use of the SDK. Being unsuccessful in determining the cause of the Edge bottleneck and with a single unresolved Github issue acknowledging it[1] at the time the experiments were performed using only direct D2C connection methods. With the Azure IoT Edge runtime and SDKs presently undergoing heavy development and not wanting to publish outdated results we were able to confirm that this was a known issue with Version 1.0.6.1 of the Edge runtime[2], fixed on the newer version. The experiments and data analysis were redone, however due to time constraints it lacks the variability in devices and messages per second it originally poised. Further limiting the variation on number of devices being tested the Edge runtime has a undocumented maximum limit of supported ongoing connections, producing the error *"Edge hub already has maximum allowed clients (100) connected"* and connection instabilities when attempting to connect more than 100 devices concurrently.

When delivery latency is a concern MQTT and AMQP are the recommended protocols[34],

[1]https://github.com/Azure/iotedge/issues/44
[2]https://github.com/Azure/iotedge/issues/1643

however testing of the AMQP protocol usage through Edge proved unsuccessful. When using the SDK for a particular protocol there are two major ways of establishing connections either through functions specific to the protocol's library or through generalized functions which receive the protocol's transport object. The issue arises that although there is no indication to the contrary and both methods are used throughout the documentation their behaviour is not identical, especially when opening and closing connections, with the generalized functions proving stabler. As an example if a device connection is not terminated gracefully and the device attempts to establish a new connection the new attempt will timeout. This significantly increased the time spent debugging. Finally when doing further testing it was decided that the AMQP protocol through Edge would not be studied at all, as when experiments were being run messages would occasionally never arrive, seemingly due to connections closing unexpectedly.

Initial experiments also had in consideration the resiliency of the Edge device to bursts of messages, this is relevant as unlike our polite round robin simulated system IoT devices do not customarily possess fair scheduling systems for messages. This option was later removed from consideration as it was found that 100 devices sending a message each in very fast succession causes the Edge runtime to seemingly halt execution for multiple minutes.

This overall dissatisfying performance of the Edge runtime also severely hampered the testing of stream processing techniques, in which the time invested into understanding and experimenting with Modules never came to fruition.

We conclude therefore that the adoption of the Azure IoT technology can be worthwhile, provided that the points set out on Section 2.6.2 of Chapter 2 are carefully considered. However the adoption of the Azure IoT Edge is at this time not recommended unless the business adopting it is prepared to lead, by being an early adopter and tolerate higher risks, and their accompanying potential development costs.

Chapter 5

Conclusions and Future Work

Recent technological innovations have propelled forward the idea of an Internet of Things (IoT), which intends to mature existing data collection systems from custom, in-house, expensive and monolithic solutions to more decentralized systems that can be deployed with less custom development and effort to radically different services and industries. Leveraging the double ended communication infrastructure already widespread throughout the world to provide resource constrained devices with more advanced agency mechanisms, powered by Big Data and Artificial Intelligence (AI) techniques, potentially bearing forth improved and novel practical applications. With a widely recognized potential for shaping existing industries, such as manufacturing, major players in the software and electronics industries have been quick to recognize the commercial potential of tapping this unexplored market segment, eagerly developing their own solutions and frameworks. Indeed dominant tech leaders such as Amazon, Microsoft, Google, Intel, Cisco and many more have begun aggressively pushing their own taste of "intelligent" and fully managed IoT services in a rush to develop their early market share for the field. These IoT services mostly rely heavily on their already robust Cloud solutions to provide the backbone of data ingestion servers and business analytics.

Rather than performing a high level survey on the purported capabilities of different highly regarded commercial frameworks we instead focused on compiling a clear and detailed picture of Microsoft's offerings. This dedicated focus is due to the partnership with Critical Manufacturing (CMF), which besides desiring a comprehensive review of the state and benefits of IoT also aimed to investigate the realistic usefulness of the Azure IoT and Edge technologies in particular, while providing their Manufacturing Execution System (MES) customers with a non-intrusive method for connecting existing IoT systems to the Azure Cloud.

We first looked to the scientific literature to determine the expected use cases for the technology and judge its possible future impact, having at this stage found the existence of prevalently acknowledged surveys which reported several practical applications. Subsequently we looked into researching what technologies and innovations are needed to begin implementing these solutions, justifying the need for each of the requirements found and listing existing works and standards that aim to address them. We then performed a deep exploration of material relating to the Azure IoT vision, it's intended practical applications and techniques used to achieve it, directly outlining chosen approaches in relation to the requirements previously established when appropriate.

Following this we began the exploration of the developmental side of the Azure IoT suite, beginning with the configuration and provisioning of Edge devices. After further exploring the suite we concluded that the envisioned most common connection methods would require devices to be running Microsoft's Software Development Kits (SDKs). This conflicts with the non-intrusive Big Data connector that CMF expected and as such we developed a middleware to act as a protocol translator, which can be inserted complementarily into existing systems. This middleware was then adapted to CMFs MES product as component of the Connect IoT package with a fully fledged Graphical User Interface (GUI) for configuration.

Afterwards intent on determining the current capabilities of the suite we used the developed work as the basis for a performance study of the minimum viable product required for IoT, with what we established Microsoft's perception to be, that is bidirectional connection between local devices and the Cloud with delivery occurring in a reasonable time frame.

We concluded that the Round Trip Time (RTT) and stability when using direct connections from simulated devices to the Cloud is reasonable, but that using the Azure IoT Edge runtime even just as a transparent pass-through node for messages leads to severe instabilities and delay on message delivery. Likely due to the recent and fast moving development of commercial frameworks there are few identical studies available, limiting the comparisons possible. We also expect that with the large developmental and monetary investment on Microsoft's IoT offerings these results could be improved in future versions.

Further testing is needed to determine the behaviour of the system over extended periods of time, hours and days. Due to the disenchanting results obtained from the bare-bones performance tests of the Edge runtime we have also relegated the testing of more complex operational models, e.g. where modules are linked in sequences to form processing pipelines, to future work. As adoption matures, and if indeed the scale and demands of IoT keeps rising to millions of devices, we would also like to include testing of real world scenarios, measuring performance from real processes to end user applications. The dataset resulting from the pre-processing steps should prove useful to the community at large and may also potentially be used for future work in testing Azure IoT Edge predictive maintenance Modules.

In conclusion at this time we have found the Azure IoT suite to be carefully thought-out and planned, with plenty of ongoing effort to improve it and its usability and, provided careful consideration is given, its adoption might be beneficial, although it is important to realize the currently large disconnect between the Azure Edge runtime goals and its real world state. It is also equally important to acknowledge that the expected deployment model does not have great out of the box compatibility with already deployed IoT solutions and substituting existing devices for Azure IoT Plug and Play certified devices is likely to come at steep initial monetary and production output costs.

Bibliography

[1] N. Ahmed, H. Rahman, and Md.I. Hussain. A comparison of 802.11ah and 802.15.4 for iot. *ICT Express*, 2(3):100 – 102, 2016. ISSN: 2405-9595. Special Issue on ICT Convergence in the Internet of Things (IoT). doi:https://doi.org/10.1016/j.icte.2016.07.003. 16

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, Fourthquarter 2015. ISSN: 1553-877X. doi:10.1109/COMST.2015.2444095. 7, 8

[3] Mahmoud Ammar, Giovanni Russello, and Bruno Crispo. Internet of things: A survey on the security of iot frameworks. *Journal of Information Security and Applications*, 38:8–27, 2018. 2, 8, 18

[4] Ross Anderson. Why cryptosystems fail. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 215–227. ACM, 1993. 15

[5] Android. Wi-fi rtt (ieee 802.11mc). https://source.android.com/devices/tech/connect/wifi-rtt, 2018 (accessed November 02, 2018). 17

[6] Giuseppe Araniti, Claudia Campolo, Massimo Condoluci, Antonio Iera, and Antonella Molinaro. Lte for vehicular networking: a survey. *IEEE communications magazine*, 51(5): 148–157, 2013. 11, 12, 16

[7] Amr Arisha, Paul Young, and Mohie El Baradie. Job shop scheduling problem: an overview. *International Conference for Flexible Automation and Intelligent Manufacturing*, 2001. 10

[8] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010. 2, 7, 8

[9] Edward Au. The latest progress on ieee 802.11 mc and ieee 802.11 ai [standards]. *IEEE Vehicular Technology Magazine*, 11(3):19–21, 2016. 17

[10] Zeinab Bakhshi, Ali Balador, and Jawad Mustafa. Industrial iot security threats and concerns by considering cisco and microsoft iot reference models. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2018 IEEE*, pages 173–178. IEEE, 2018. 2

[11] S Chen, H Xu, D Liu, B Hu, and H Wang. A vision of iot: Applications, challenges, and opportunities with china perspective. ieee internet of things journal, 1, 349-359, 2014. 7, 8, 12

[12] Shanzhi Chen, Jinling Hu, Yan Shi, and Li Zhao. Lte-v: A td-lte-based v2x solution for future vehicular network. *IEEE Internet of Things journal*, 3(6):997–1005, 2016. 16

[13] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4):2233–2243, 2014. 2

[14] Jan Damsgaard and Jan Karlsbjerg. Seven principles for selecting software packages. *Communications of the ACM*, 53(8):63–71, 2010. 13

[15] Hasan Derhamy, Jens Eliasson, Jerker Delsing, and Peter Priller. A survey of commercial frameworks for the internet of things. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8. IEEE, 2015. 2

[16] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 171–172. IEEE, 2015. 32

[17] Flexera. Cloud computing trends: 2019 state of the cloud survey, May 2019. 31

[18] Stefan Forsström and Ulf Jennehag. A performance and cost evaluation of combining opc-ua and microsoft azure iot hub into an industrial internet-of-things system. In *Global Internet of Things Summit (GIoTS), 2017*, pages 1–6. IEEE, 2017. 3, 4, 49

[19] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. *arXiv preprint arXiv:0901.0131*, 2008. 5

[20] Frieder Ganz, Daniel Puschmann, Payam Barnaghi, and Francois Carrez. A practical evaluation of information processing and abstraction techniques for the internet of things. *IEEE Internet of Things journal*, 2(4):340–354, 2015. 14

[21] Sam George. Hannover messe 2018: Manufacturers put their trust in microsoft's industrial iot platform, Apr 2018. 22

[22] Sam George and Rushmi Malaviarachchi. Our iot vision and roadmap, May 2019. 22, 29

[23] Sam George, Rushmi Malaviarachi, and Galen Hunt. Microsoft iot overview, vision and roadmap, May 2018. xi, 19, 22, 29

[24] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013. 2, 10, 13, 17

[25] Oliver Hahm, Emmanuel Baccelli, Hauke Petersen, and Nicolas Tsiftes. Operating systems for low-end devices in the internet of things: a survey. *IEEE Internet of Things Journal*, 3 (5):720–734, 2016. 12, 13, 17, 18

[26] Mark A Hanson, Harry C Powell Jr, Adam T Barth, Kyle Ringgenberg, Benton H Calhoun, James H Aylor, and John Lach. Body area sensor networks: Challenges and opportunities. *Computer*, 42(1), 2009. 9

[27] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 257–260. IEEE, 2012. 7, 8, 17

[28] Eva Kiktova, Martin Lojka, Matus Pleva, Jozef Juhar, and Anton Cizmar. Gun type recognition from gunshot audio recordings. In *Biometrics and Forensics (IWBF), 2015 International Workshop on*, pages 1–6. IEEE, 2015. 11

[29] Uday Kumar, Tuhin Borgohain, and Sugata Sanyal. Comparative analysis of cryptography library in iot. *arXiv preprint arXiv:1504.04306*, 2015. 18

[30] Steve G Langer and Todd French. Virtual machine performance benchmarking. *Journal of digital imaging*, 24(5):883–889, 2011. 31, 32

[31] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. Nist cloud computing reference architecture. *NIST special publication*, 500(2011):1–28, 2011. 5

[32] Tao Liu and Dongxin Lu. The application and development of iot. In *Information Technology in Medicine and Education (ITME), 2012 International Symposium on*, volume 2, pages 991–994. IEEE, 2012. 8

[33] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. *NIST special publication*, 2011. 5

[34] Microsoft. *Microsoft Azure IoT Reference Architecture*. Microsoft, 2018. xi, 20, 21, 22, 24, 26, 33, 49

[35] Daniel Minoli, Kazem Sohraby, and Benedict Occhiogrosso. Iot considerations, requirements, and architectures for smart buildings—energy optimization and next-generation building management systems. *IEEE Internet of Things Journal*, 4(1):269–283, 2017. 9, 12

[36] Roberto Morabito. A performance evaluation of container technologies on internet of things devices. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 999–1000. IEEE, 2016. 22

[37] Andrea Murphy, Jonathan Ponciano, Sarah Hansen, and Halah Touryalai. Global 2000 - the world's largest public companies 2019, May 2019. ix, 29

[38] Mary Natrella et al. Nist/sematech e-handbook of statistical methods. *Nist/Sematech*, 49, 2010. 47

[39] Peter Provost. Build with azure iot central and iot plug and play: Blog: Microsoft azure, May 2019. 25

[40] Agnieszka Radziwon, Arne Bilberg, Marcel Bogers, and Erik Skov Madsen. The smart factory: exploring adaptive and flexible manufacturing solutions. *Procedia engineering*, 69: 1184–1190, 2014. 10

[41] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016. 12, 13, 14, 17

[42] Jatinder Singh, Thomas Pasquier, Jean Margaret Bacon, Hajoon Ko, and David Eyers. Twenty security considerations for cloud-supported internet of things. *IEEE Internet of Things Journal*, 2015. 15

[43] John A Stankovic. Research directions for the internet of things. *IEEE Internet of Things Journal*, 1(1):3–9, 2014. 12

[44] Terry Therneau, Cindy Crowson, and Elizabeth Atkinson. Using time dependent covariates and time dependent coefficients in the cox model. *Survival Vignettes*, 2017. 40

[45] Shiyong Wang, Jiafu Wan, Di Li, and Chunhua Zhang. Implementing smart factory of industrie 4.0: an outlook. *International Journal of Distributed Sensor Networks*, 12(1): 3159805, 2016. 10, 12

[46] David A Wheeler. What is open security? Technical report, INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA, 2013. 15

[47] Julia White. One year in: How our $5b investment in iot and intelligent edge is accelerating customer, partner and solution innovation, Apr 2019. 19

[48] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun, and Hui-Ying Du. Research on the architecture of internet of things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 5, pages V5–484. IEEE, 2010. 7

[49] Yi Zhong. Benefits of using the azure iot sdks, and pitfalls to avoid if you don't, Oct 2017. 27