MSc thesis

Computer Science

Edge computing platforms for Internet of Things

Jarkko Kovala

May 6, 2020

Faculty of Science

University of Helsinki

Supervisor(s)

Prof. Keijo Heljanko

Examiner(s)

Prof. Keijo Heljanko, Dr. Tewodros Deneke

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

Tiedekunta — Fakultet — Faculty Faculty of Science	Koulutusohjelma — Utbildningsprogram — Study programme Computer Science

Tekijä — Författare — Author

Jarkko Kovala

Työn nimi — Arbetets titel — Title

Edge computing platforms for Internet of Things

Ohjaajat — Handledare — Supervisors

Prof. Keijo Heljanko

Työn laji — Arbetets art — Level MSc thesis	Aika — Datum — Month and year May 6, 2020	Sivumäärä — Sidoantal — Number of pages 67 pages

Tiivistelmä — Referat — Abstract

Internet of Things (IoT) has the potential to transform many domains of human activity, enabled by the collection of data from the physical world at a massive scale. As the projected growth of IoT data exceeds that of available network capacity, transferring it to centralized cloud data centers is infeasible. Edge computing aims to solve this problem by processing data at the edge of the network, enabling applications with specialized requirements that cloud computing cannot meet.

The current market of platforms that support building IoT applications is very fragmented, with offerings available from hundreds of companies with no common architecture. This threatens the realization of IoT's potential: with more interoperability, a new class of applications that combine the collected data and use it in new ways could emerge.

In this thesis, promising IoT platforms for edge computing are surveyed. First, an understanding of current challenges in the field is gained through studying the available literature on the topic. Second, IoT edge platforms having the most potential to meet these challenges are chosen and reviewed for their capabilities. Finally, the platforms are compared against each other, with a focus on their potential to meet the challenges learned in the first part.

The work shows that AWS IoT for the edge and Microsoft Azure IoT Edge have mature feature sets. However, these platforms are tied to their respective cloud platforms, limiting interoperability and the possibility of switching providers. On the other hand, open source EdgeX Foundry and KubeEdge have the potential for more standardization and interoperability in IoT but are limited in functionality for building practical IoT applications.

ACM Computing Classification System (CCS)
Computer systems organization → Architectures → Distributed architectures → Cloud computing,
Computer systems organization → Embedded and cyber-physical systems,
Human-centered computing → Ubiquitous and mobile computing

Avainsanat — Nyckelord — Keywords

Internet of Things, IoT, edge computing, cloud computing, platforms

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Networking and Services specialisation line

Contents

1 Introduction

The *Internet of Things* (*IoT*) refers to the concept of physical things interconnected through a network, having the ability to generate and exchange data through the use of unique addressing and common protocols, all without human intervention [1]. IoT systems have the potential to transform many domains by creating information on a scale never seen before and helping build systems that are more dynamic and autonomous. Because of this, IoT has seen a surge of interest both in research and in the industry during the past two decades [1].

Storing and processing IoT data requires efficient and highly scalable computing capacity. This is offered by *cloud computing*, which produces computing capacity as a service from large, centralized data centers using virtualization technologies [2]. However, cloud computing has limitations [3]: applications that require extremely fast response times, connectionless operation, or processing of more data than the network can carry, cannot rely on the cloud.

In *edge computing*, data is processed at the edge of the network, closer to where the data is generated or consumed. A fundamental shift from the past trend of centralization in computing to a more decentralized model has been proposed as ”edge-centric computing” by López et al. [4].

IoT *platforms* support building IoT applications by providing required common capabilities so developers can focus on the application. The market is very fragmented, with over 400 companies offering solutions as IoT platforms [5].

In this thesis, promising IoT platforms are surveyed and comparatively evaluated for their capabilities of meeting the challenges of edge computing for IoT. The work is organized in three parts: in Chapter 2, an understanding of current challenges in IoT and edge computing is reached by studying the available literature on the topic. In Chapter 3, promising platforms that support building applications for IoT edge are selected, analyzed, and compared with a focus on their potential to meet the practical challenges as determined in Chapter 2. Finally, Chapter 4 contains a summary of the work and a discussion of the findings, limitations, and some interesting future research questions.

2 IoT at the edge

In this chapter, the current research challenges of IoT at the edge are established through a survey of literature in the field. The results form the foundation for evaluating IoT edge platforms in Chapter 3. IoT is discussed in Section 2.1 and edge computing in Section 2.2.

2.1 Internet of Things

2.1.1 Introduction

Through technological progress, cheap and small *identification* and *sensing* elements can be embedded in various things, making them pervasive in our environment [6]. Internet of Things aims to take advantage of this by interconnecting them, creating intelligent and adaptive systems of components that can operate autonomously and allowing the collection of massive amounts of data, with applications in many domains [1].

Multiple visions and approaches have been defined as research and development of IoT have progressed: they have been categorized in [1] into things-oriented, Internet-oriented, and semantic-oriented. The original IoT vision of connecting physical objects originated from identification technology and focuses on the things, going in the first category.

Another things-oriented vision for IoT is the *smart object* [6]. The development of small and cheap electronic components and wireless communication has created the possibility for ubiquitous computing. Everyday objects can be made smart through embedding such components, such as processors and sensors, in them. A smart object can communicate not only with humans interacting with it but with anything over the Internet and depending on the context; for example, a smart object can retrieve operating instructions to be presented to the user.

Internet-oriented approaches for IoT focus on the connectivity aspect: the network technologies that enable universal connectivity of things [1].

Scalable connectivity is necessary for ambitious projects like HP's Central Nervous System for the Earth (CeNSE)[1], that aims to use nanotechnology to build a network of up to a trillion pin-size sensors encompassing the planet. For scale, the number of IoT devices was estimated to be approximately 7 billion in 2018, projected to grow to approximately 21.5 billion by 2025[2].

As the number of devices and the volume of information generated by them grows and complexity increases, doing something useful with the information becomes more challenging. Semantic-oriented visions for IoT concentrate on building a common language for machines to store, organize, and communicate information [1], so that the information can be machine-processed. For example, the Web of Things proposes to do this by using standards created for the web [7].

The simplest element used for identification in an IoT system is a *radio-frequency identification* (*RFID*) tag [1]. RFID is a technology for identifying and tracking objects that works by using *tags* that can communicate data to *readers* using electromagnetic fields [8]. The operating principle is similar to anti-theft tags that are read by security gates at department stores, except that much more data may be transmitted and in both directions. RFID tags are integrated circuits containing memory and an antenna, and may be *active*, containing their own power source, or *passive*, drawing energy from the signal sent by a reader. RFID readers are devices that can read data from, or write data to, RFID tags, and may be handheld or mounted to things such as vehicles or posts. Because the tags are cheap enough to be considered disposable and are very small, the size of a sticker, they can be attached to large numbers of objects at a low cost [9]. For example, a retail company can use RFID technology by attaching tags to their goods to track them for inventory management and even simplifying the shopping process [10].

RFID has been a successful technology in driving IoT applications in multiple industries: already by 2006, an estimated 2.4 billion RFID tags had been produced [8]. Due to its low cost and maturity, RFID technology is still at the forefront of IoT [11].

While RFID provides identifying an object at a location, other basic building blocks that link computers with the physical world include *sensors* and *actuators* [12, p. 23]. Sensors, such as cameras, generate data from the world by responding to physical stimulus, while

[1]CeNSE — HP Official Site. https://www8.hp.com/us/en/hp-information/environment/cense.html (accessed April 15, 2020)

[2]State of the IoT 2018: Number of IoT devices now at 7B - Market accelerating. https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/ (accessed April 15, 2020)

actuators, such as motors, change some physical state by converting energy into motion.

The term 'Internet of Things' was coined in 1999 by Kevin Ashton at the MIT Auto-ID Center [11]. His original idea was to use RFID and the Internet to enable computers to automatically generate and communicate information. The goal was to sidestep human limitations of collecting data about the physical world, having "the potential to change the world, just as the Internet did" [11].

In October 2003, The MIT Auto-ID Center was split into a research arm, the Auto-ID Labs[1], and a commercial arm, EPCglobal[2] [13, p. 12]. The Auto-ID Labs researches identification and sensing technologies, while EPCglobal promotes the Electronic Product Code (EPC), a standard for uniquely identifying physical objects.

2.1.2 Applications of IoT

IoT has applications in many different domains. These have been categorized by, among others, [1, 12, 14, 15, 16, 17, 13]. In this subsection, some of these application domains and their potential benefits are described briefly.

Industry *Smart factories* use sensors to collect information relevant to the manufacturing process and distribute it to help people and machines in their tasks [18]. For example, sensors can be installed on a machine to monitor its operating conditions, such as temperature or vibration, and the collected data may be used to diagnose and even predict incorrect operation [12, pp. 60–61]. Using sensor data and autonomous systems, development times can be shortened, decision-making made faster through decentralization, and resource efficiency increased through reduced waste [19]. Smart factories can potentially start a fourth industrial revolution after mechanization, use of electricity, and digitalization, called 'Industry 4.0' [19].

Cities *Smart cities* use IoT in multiple domains, such as transportation, safety, waste management, and energy efficiency [20]. Road sensors and displays can be used to help motorists find parking lots faster and in other route planning, reducing congestion and pollution. Surveillance cameras improve safety in public areas. Smart waste containers that detect their load can be used to optimize waste collection, while street lighting can

[1]Auto-ID Labs. https://www.autoidlabs.org/ (accessed April 15, 2020)
[2]EPCGlobal - Standards — GS1. https://www.gs1.org/epcglobal (accessed April 15, 2020)

be adjusted based on time, weather, and presence of people to both increase safety and reduce electricity consumption [20]. Air quality sensors can be used to help people plan their outdoor activities to reduce the health effects of air pollution [20].

Health and fitness *Wearable sensors* are non-intrusive sensors worn on the human body. By monitoring physiological parameters, wearable sensors enable IoT systems that continuously monitor health and fitness [12, pp. 62–63]. Sensors used in such applications, such as temperature, heart rate, oxygen saturation, blood pressure, electrocardiogram, electroencephalogram, and movement sensors form a sensor network, a *personal area network* (*PAN*), that centrally collects wearable sensor data to be sent for analysis [12, pp. 62–63]. The data can be used to, for example, help in the treatment of chronic diseases, elderly care, or aid in fitness programs [21]. A *smart wristband* that can track parameters such as steps taken and calories burned is an example of such a device; several systems based on smart wristbands have been developed [21].

Home In *smart homes*, smart objects are used for a more comfortable and safe environment, and to save energy [1]. For example, heating and lighting can be automatically controlled based on personal preferences, time, and the weather; smart appliances can be remotely controlled and monitored, and can turn themselves off automatically when they are not needed; refrigerators can keep track of their contents; and monitoring and alarm systems such as intrusion or fire detectors can be used to prevent incidents [12, pp. 48–50].

Energy *Smart grids* improve electric grids by providing visibility and control to its components, improving resiliency and efficiency, and even creating new opportunities for different stakeholders that are involved in the electric grid [22]. Essentially, a smart grid consists of a network integrated with the electric grid, collecting and analyzing data from sensors, such as power consumption meters, in almost real time [12, pp. 54–55]. Traditional electric grids are operated from central locations using mechanical controls and require manual interventions when faults occur, but a smart grid can heal itself automatically [22].

Environment Applications that help to monitor and preserve the environment are a promising IoT domain [13, p. 54]. These include, among others, systems that monitor environmental parameters such as weather, air and noise pollution, and catastrophes such as forest fires and floods, to assist in managing these phenomena [12]. Other green applications help to reduce waste and prevent emissions both in industrial and home settings [15, pp. 19–20] and to monitor and preserve wildlife [23].

Logistics and transportation *Intelligent transportation systems* that collect data from vehicles and transport people and goods help to improve efficiency and safety [12, p. 58]. Collected data may be used to, for example, improve route planning by anticipating traffic or customer need, manage fleets more efficiently by analyzing problems faster, reduce spoilage by monitoring environmental conditions in shipments, and improve reliability through diagnosing problems in vehicles remotely [12, pp. 58–59]. One emerging technology in this domain is *Vehicle-to-Vehicle (V2V)* communications that enable vehicles to exchange data, for example, to warn drivers of impending crashes. V2V has been estimated to have the potential to prevent up to 18 percent of automobile accidents [24].

Agriculture *Smart agriculture* uses soil, livestock, environment, and weather monitoring to improve productivity, quality and to reduce waste [12, p. 59]. For example, using soil moisture sensors for more optimal irrigation can improve crop yields while saving water, monitoring and controlling greenhouses can help maintain desired conditions with the optimal amount of resources [12, p. 59], while livestock IoT can help in optimizing environmental conditions and feeding of cattle [25]. In addition to improving productivity, smart agriculture can help to control the environmental impact of agriculture.

Retail Retail applications of IoT save costs, reduce environmental footprint, and improve sales for retailers [13]. RFID tags can be used to track agricultural products through the supply chain after harvest, for example, to reduce spoilage [25]. Real-time information on inventory can help prevent over- and understocking [12, p. 56], and smart price tags can be used to change prices in real-time based on supply and demand [26].

Insurance Insurance companies can use data generated by IoT to more accurately calculate individual risk and assign premiums accordingly, for example by charging drivers based on the safety of their driving style [27]. Other applications include usage-based insurance, where insurance premiums are based on metered usage of the insured entity, in addition to the more general opportunities in preventing accidents and mitigating loss [28].

This listing of IoT applications is by no means exhaustive. The applications are very diverse with heterogeneous requirements. For example, some have more real-time requirements, such as the transportation or industrial safety applications and health monitoring, where missing deadlines can potentially lead to fatal results; others are less real-time, for example, the waste or retail inventory management applications, for which timeliness only has marginal implications for efficiency.

2.1.3 IoT architectures

IoT systems consist of three primary elements: physical devices that enable collecting data from the environment and interaction with it; communication technologies that enable transmission of data between devices and systems; and application services that do something useful with the data. The application element includes the computational and storage capacity required to run the required software. As expected by the diversity of IoT applications, different components used in IoT systems are numerous and heterogeneous [29]. Multiple different architectural models have been put forward to describe and organize these components into systems with no consensus on any single one [17, 30, 1, 31, 32]. In this subsection, the basic layered architectural models for IoT organization are described.

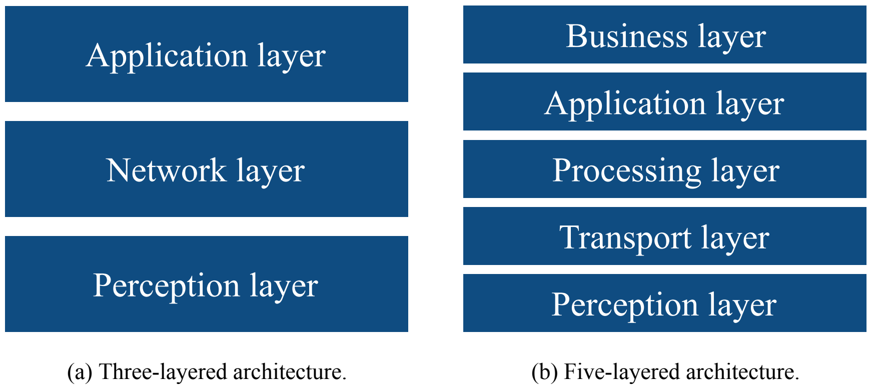

Figure 2.1: Layered IoT architectures. Adapted from [17].

The simplest layered architectural model for IoT systems is the three-layered architecture [17], depicted in Figure 2.1(a). It consists of the *perception layer* that gathers information about the environment; the *network layer* that provides connectivity and transmits data collected by the perception layer; and the *application layer* that provides services specific to the IoT application, such as processing the collected data into useful information and delivering it to users. The three-layer model describes the general functionality of IoT systems, is easy to understand and widely accepted, but is not sophisticated enough for research or designing practical systems [17].

Another layered architectural model is the five-layered architecture [32], depicted in Figure 2.1(b). Compared to the three-layer model, the five-layered architecture adds two layers: the *processing layer*, which is sometimes called the middleware layer [17], that stores and processes data for use by the application layer, and the *business layer*, that manages the complete system, especially from business model and user privacy perspectives. In the five-layered model, the data-transmitting network layer is called the *transport layer*.

2.1.4 Sensing and identification

The perception layer collects data from the environment through identification and sensing [17]. The most ubiquitous technology for identification is the RFID, which uses tags that are attached to objects, containing data that is read by a reader device. RFID is used in a broad range of applications from tracking objects across supply chains to access control. Sensing is done by sensors that measure some internal or external state of an IoT device. Examples of sensors include cameras; other light sensors, such as infrared or ultraviolet sensors; environmental sensors, such as temperature, pressure or humidity sensors; and medical sensors, such as wearable smartwatches or skin patches that measure human physiological parameters [17].

Modern smartphones typically contain a wide range of sensors, making them an important class of IoT devices [33]: these include ambient light sensors, proximity sensors, cameras, microphones, GPS chips for estimating position, accelerometers that measure the direction of acceleration, magnetometers that measure directions of magnetic fields, and gyroscopes that measure rotation.

Some smartphone sensors and electronics are being used for IoT sensing despite being originally designed in smartphones for other purposes. For example, accelerometers and gyroscopes were added to phones for estimating the phone's orientation to present the user interface better in various orientations, but has since been used to, among others, detect user activities such as determining whether the user is running, walking, or standing [33]. In some applications, electronics that were not even designed to be sensors are used. An example is a novel application where wireless radios originally intended for wireless communication are used to sense various properties, from human activity and physiology to traffic detection [34]. Wireless radios are also used in an emerging application called privacy-preserving contact tracing, which uses short-range wireless radios embedded in mobile devices to trace human-to-human contacts to gather information to control epidemic diseases [35]; this idea is being planned to be used with the COVID-19 pandemic[1].

2.1.5 IoT devices

IoT devices are typically built using *embedded systems*, which are computer systems built into things and designed for a specific set of tasks [12, p. 38]. An embedded system has the typical components of a computer, such as a *central processing unit* (*CPU*) for computing, memory for storing data, and networking capability for communications, but unlike a personal computer, it is not built for general-purpose computing. Embedded systems are often built using *microcontrollers* [36], which are integrated circuits with a CPU, memory, and other necessary components built into it. Microcontroller-based systems are typically cheaper, smaller, and less power-consuming than more high-end devices with dedicated CPUs and other hardware components, but have fewer capabilities and limited flexibility.

The simplest IoT devices are the most common ones but have very limited hardware capability: their memory capacity, for example, ranges from a few kilobytes to tens of kilobytes [37]. They do not have an *operating system* (*OS*), which is a generic software that manages the system's hardware and software resources and acts as a platform for applications. Instead, they have software written specifically for the device. Often they are powered by a limited power source such as a battery, making them power constrained. Examples of such devices include smart light bulbs, thermostats, and wireless sensors.

More capable IoT devices, with memory capacities from tens of kilobytes and more, can support simple operating systems specifically designed for use on low-end devices [36]. As these devices are resource-constrained, the operating systems are designed for energy efficiency and a small footprint. Such specialized operating systems are typically modular so that only the necessary functionality for the specific device is installed, and the hardware and programming language support is limited. When IoT devices control systems that require precise timing, for example for safety in health applications or when controlling industrial robots, a *real-time operating system* (*RTOS*) design that can fulfill timing requirements may be used.

[1]How Apple and Google Are Enabling Covid-19 Bluetooth Contact-Tracing — WIRED. https://www.wired.com/story/apple-google-bluetooth-contact-tracing-covid-19/ (accessed April 18, 2020)

A high-end IoT device has capabilities closer to that of a personal computer [36]. Having memory capacities in the range of megabytes to multiple gigabytes, they can run full-fledged operating systems such as Linux[1], capable of running a broader range of applications [37]. Examples of such devices include smartphones and the Raspberry Pi single-board computer[2].

2.1.6 IoT network protocols

As different IoT applications have diverse requirements, IoT systems use a variety of network technologies to transmit data [17]. Common requirements for IoT networking technologies include scalability to support a potentially large number of devices; low power and computing and memory capacity requirements to support constrained devices; high reliability; and mobility. The emphasis given to each requirement varies depending on the application.

The *Internet Protocol* (*IP*) protocol stack is the primary communication protocol for the Internet [17]. It is organized into four layers: *physical layer* provides the physical way of connecting via a medium such as wires, optical fiber or radio waves; *network layer* routes data, which is encapsulated into packets, from source to destination across networks; *transport layer* provides connection-oriented communication and reliability for data transmitted over the network layer; and *application layer* provides functionality specific to the application. For IoT, an additional *adaptation layer* may be used between the physical and network layers, adding additional abilities needed to use standard higher-level protocols with low-power networks for resource-constrained devices [17].

The standard protocol for the network layer in the Internet is *IPv4*, or version 4 of the IP protocol. It uses 32-bit addressing, limiting the total number of unique hosts to a maximum of slightly over four billion, making it unable to support the massive number of devices in IoT scenarios [1]. Version 6, *IPv6*, increases the address field to 128 bit to a maximum of 10^{38} - essentially an infinite number of unique hosts, thus being much more suitable for many IoT applications [1, 38, 17].

When mobility is required for an IoT device, wireless communication is typically used for physical connectivity. More capable devices can use WiFi, a wireless networking protocol

[1]Linux - The Linux Foundation. https://www.linuxfoundation.org/projects/linux/ (accessed April 15, 2020)

[2]Teach, Learn, and Make with Raspberry Pi. https://www.raspberrypi.org/ (accessed April 15, 2020)

with a transmission range of up to 100 meters, or cellular networks such as *3G*, *4G*, or *5G* [38]. The most resource-constrained devices may not have the capabilities to run any of these or even a full IP stack [17]. An alternative is to use low-power local communication to connect to a more capable device called a *smart gateway* that can relay the data to the Internet[1]. Most such wireless network protocols with a physical range of 10 to 20 meters, such as *Zigbee*, are based on the IEEE 802.15.4 standard [1]. *IPv6 over Low-Power Wireless Personal Area Networks* (*6LoWPAN*) is a proposal that combines 802.15.4 with IPv6 by adding adaptation layer features to be used by low-power devices [17].

Standard transport layer protocols include *TCP*, a reliable, connection-oriented protocol, and *UDP*, a connectionless protocol with no delivery guarantees. Because TCP has higher overhead due to having more features, UDP is a preferable choice for low-power devices [17].

HTTP, the protocol used as the foundation of the World Wide Web, is the most common protocol used for the application layer in the Internet. It is also used by many higher-end IoT devices that have the resources to support it [37]. *HTTPS* is an extension of HTTP that adds a security layer under the HTTP transport, using the *TLS* protocol to encrypt and authenticate communications.

A recent development is the *QUIC* protocol [39]. HTTPS consists of protocols TCP, HTTP and TLS layered on top of each other; the protocols have redundant facilities for connection setup, causing increased latencies inherent to the design, and in general, were not designed for the complexity of modern Internet applications and mobility. QUIC replaces these protocols and combines their functionality in a single layer, providing features like low-latency connection setup, IP address-independent connections, multiple streams within a single connection, and more accurate round-trip time calculation for improved performance with low-quality links.

For more constrained devices, multiple application layer protocols have been defined as alternatives to HTTP [17]. *Constrained Application Protocol* (*CoAP*)[2] is designed for constrained devices and is conceptually similar to HTTP for simpler integration with Web applications. CoAP uses UDP for transport by default, and security features are implemented using the *DTLS* protocol. While the messages in CoAP are of very low

[1]RFC 7228 - Terminology for Constrained-Node Networks. https://tools.ietf.org/html/rfc7228 (accessed April 15, 2020)

[2]RFC 7252 - The Constrained Application Protocol (CoAP). https://tools.ietf.org/html/rfc7252 (accessed April 15, 2020)

overhead and fit inside a single packet, it implements a subset of the methods used in HTTP, so CoAP messages can easily be translated into HTTP.

Message Queue Telemetry Transport (*MQTT*)[1] is a lightweight messaging protocol based on the *publish/subscribe* messaging pattern. In publish-subscribe messaging, senders send messages by publishing them on a *topic*, while senders receive them by subscribing to the topic; this allows for communication without having the senders or receivers know about the existence of the others and leads to greater scalability and flexibility than delivering messages using specific destination addresses. A MQTT messaging system is organized around a *message broker* that acts as a server to which clients connect to publish messages or subscribe to topics. The topics are arbitrary in the sense that they are not preconfigured on the server. MQTT runs over the TCP/IP protocol stack by default but may also be implemented over other protocols that provide reliable bidirectional connections, such as HTTP.

2.1.7 Challenges in IoT

Open challenges in IoT have been surveyed in the past decade in, among others, [1, 38, 40, 41, 42, 43]. In this subsection, some of these challenges that are still relevant today are discussed.

Standardization Standardization has been consistently identified as one of the main challenges for IoT development [1, 38, 40, 41, 43]. Standardization helps organizations build systems that use interchangeable and interoperable elements, lowering development costs and times through commoditization of components and repeatable designs. Standardization also fosters innovation through the creation of ecosystems of organizations that build systems based on commonly agreed specifications. Despite the massive effort of several standards organizations and industry alliances over decades, IoT standards are diverse and lack consistency, and no unifying reference standard exists [43].

Interoperability Interoperability refers to the ability of systems to work together, such as exchanging and using exchanged information, regardless of possible heterogeneity in implementation. As IoT systems are being built by many different organizations in a multitude of diverse industries using very heterogeneous devices, technologies, and systems,

[1]MQTT Version 5.0. `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html` (accessed April 15, 2020)

achieving interoperability is a major challenge [44]. However, the opportunity is massive: an analysis in 2015[1] found that most IoT data is not used or fully exploited, and if multiple IoT systems could work together, it could unlock up to 40 percent of the total value potential of IoT as a whole. Data generated by different IoT systems could be combined and used for insights that are not available within an isolated system, but lack of technical interoperability remains a barrier to this.

Data processing and storage IoT systems generate massive amounts of data that needs to be stored and processed to extract useful information out of it. Storing, transporting, and processing data at such a scale is a challenge: storing and processing it on the resource-constrained devices themselves may not be possible let alone cost-effective while transferring it to centralized locations requires high-performance networking and incurs delays and costs [43].

Naming and identity To build IoT systems involving massive amounts of things, methods to uniquely identify such large numbers of objects are required [43]. The problem involves not only having a sufficiently large namespace - for example, IPv6 already supports a practically infinite number of addresses. As the manual assignment of addresses will not scale, solutions will need to support automatic registration, service discovery, and mobility, and be lightweight enough that resource-constrained devices may support them.

Device management IoT devices need to be monitored and controlled throughout their lifecycle from their activation to the time they are eventually taken out of use [43]. Doing this in a scalable manner is difficult due to the large amount of heterogeneous devices with limited network connectivity. Ideally, the devices should be self-configuring, requiring little human effort in provisioning new devices and adapting to changes in the environment.

Privacy and security IoT systems collect and process personal and other sensitive data, operate safety-critical systems, and control highly valuable processes. For IoT ap-

[1]McKinsey Global Institute. The Internet of Things: Mapping the value beyond the hype. June 2015. https://www.mckinsey.com/~/media/McKinsey/Industries/Technology%20Media%20and%20Telecommunications/High%20Tech/Our%20Insights/The%20Internet%20of%20Things%20The%20value%20of%20digitizing%20the%20physical%20world/The-Internet-of-things-Mapping-the-value-beyond-the-hype.ashx (accessed April 15, 2020)

plications to be adopted, they need to be trusted by users [45]. Therefore, privacy and security is a critical issue and has been studied extensively [1, 45, 43]. A particular problem is that IoT systems consist of large amounts of devices that generally are unattended and have low capabilities, so they are easy to attack physically and available protection measures are limited [1]. This is further complicated by the fact that IoT systems need to be protected at all layers and involve heterogeneous components [45].

2.2 Edge computing

2.2.1 From cloud to edge computing

Cloud computing refers to the idea of providing computing as a service over a network from massive centralized cloud data centers: utility computing [2]. This model has become successful due to its inherent efficiency, but it is being challenged by IoT applications with special latency, bandwidth, reliability, power consumption, and privacy requirements [46].

Compared to buying your own computing hardware, computing as a service reduces the upfront investment and the time and labor expense of hardware deployment when new computing capacity is needed, while producing computing at a scale in large data centers brings economies of scale, making the cloud economically appealing [2].

Cloud computing is an umbrella term that encompasses several models for providing services [2]. A *public cloud* is a service where cloud computing is provided to the general public; in contrast, a *private cloud* is a cloud-scale operation where the service is internal to an organization. The service may be provided on multiple different levels: in *Infrastructure as a Service* (*IaaS*), as basic compute, storage, and network capacity; in *Platform as a Service* (*PaaS*), as a platform on which applications may be developed and deployed on; and in *Software as a Service*, as software applications. Each of these abstracts the underlying systems that produce the service: the user generally does not know, for example, the hardware used or how it is organized, reducing the amount of expertise required to develop and operate applications on top of the service.

In cloud systems, computer hardware is shared among multiple users using *virtualization*, which enables a single computer to run multiple *virtual machines* each running their own operating systems [2, 47]. In a traditional computer system without virtualization, a single operating system is installed on a physical machine, on which software applications may be installed and run. Because computing capacity can be more easily shared through virtualization, utilization is improved. Another major benefit is that resources are much more elastic, as virtual machines can be created and destroyed faster than physical hardware. Providing virtual machines as a service is an example of IaaS called *Virtual Private Server* (*VPS*).

In addition to running full operating systems in virtual machines, more lightweight approaches to virtualization exist [48, 49]. For example, in *containers*, instead of each virtual instance running a full operating system, a single operating system supports multiple, isolated namespaces that applications are deployed in. Container-based virtualization has less overhead and containers are even faster to deploy and tear down than virtual machines, making them desirable for building cloud-based applications.

Public cloud services are generally priced on some type of pay-as-you-go model, where the user is metered on their usage and billed accordingly. However, the exact pricing schemes differ between the numerous companies offering these services, making comparisons complex [50, 51, 52].

Building applications on cloud platforms carries the risk of *vendor lock-in*, which refers to the difficulty of moving applications and data elsewhere because of proprietary and incompatible interfaces [53, 54, 44]. Another problem, somewhat the opposite of vendor lock-in, is when the service changes or disappears altogether, breaking the application; for example, it is not uncommon for cloud services to be in a beta or preview status for long periods. Beta or preview services generally may be terminated by the service provider at any time without warning or liability[1][2].

Cloud computing produces highly scalable computing resources efficiently and therefore is in a good position to provide the capacity required by IoT [55], but it has inherent limitations [46]. While the amount of processing capacity and generated data has increased, the growth is outpacing the increase in network capacity required to transfer the data to the cloud [56]. Network delays have a lower limit based on the speed of light: low-latency applications, for example, augmented reality, do not have the time to communicate with remote data centers over network links. Other applications cannot rely on a network connection due to reliability requirements, for example in assisted driving. Maintaining a network connection consumes power, which may not be possible for constrained devices

[1]AWS Service Terms. https://aws.amazon.com/service-terms/ (accessed April 15, 2020)

[2]Preview Terms Of Use — Microsoft Azure. https://azure.microsoft.com/en-us/support/legal/preview-supplemental-terms/ (accessed April 15, 2020)

relying on batteries or other limited power sources. It may also be undesirable to send the data to the cloud due to privacy considerations.

To counter the limitations of cloud computing, a new paradigm called *edge computing* has been proposed [4, 46]. In edge computing, data is processed or stored on resources at the edge of the network: edge here refers to any location outside a centralized data center, closer to where data is sourced or consumed. This is possible be on existing infrastructure such as on a network gateway or a mobile base station, but also on an IoT device, for example on a modern smartphone.

Edge computing has several potential benefits [46, 57]:

- network delay is reduced, improving response times,
- reduced network bandwidth requirements and usage,
- improved availability through less reliance on a cloud connection,
- improved control over data for security and privacy,
- reduced energy consumption through offloading computation from devices and less network communication, and
- reduced cloud processing costs.

Fog computing is a concept closely related to edge computing [4]. In fog computing, the cloud is extended to the edge of the network by deploying the same virtualization tools used in the cloud on infrastructure placed outside datacenters [58]. A similar technology that has seen much research interest during the past decade is *cloudlets*, which augment the capabilities of mobile devices by deploying virtual machines on cloud servers placed close to the users [59].

Another edge computing paradigm is *Mobile Edge Computing* (*MEC*), where edge applications are run on mobile base stations with integrated computing capacity, turning mobile network operators into cloud service providers [60]. Mobile base stations are widespread and located close to users, which makes them promising for providing edge capacity.

Network Function Virtualization (*NFV*) decouples services provided by network infrastructure from hardware by using virtualization [61]. It brings the elasticity of the cloud to network infrastructure: for example, services that have previously been provided by multiple network appliances can be consolidated into a single piece of hardware using

NFV, reducing capital costs. Additionally, it enables flexible deployment of novel services, such as services that bring video streaming services closer to users for improved quality. Having infrastructure capable of running virtualized workloads creates an opportunity to implement MEC using the infrastructure [60].

2.2.2 Applications of edge computing

Many application domains can potentially benefit from advantages of edge computing; these have been surveyed in, among others, [4, 46, 62, 57]. This subsection introduces a subset of these applications.

A *Content Delivery Network* (*CDN*) is a popular technology to scale web services by storing data at the edge of the network and has enabled the deployment of global-scale web applications [63, 64]. A CDN consists of a distributed network of servers that cache web content. The content is typically stored on a central location where the globally distributed CDN fetches it from. The primary benefits of a CDN include reduced bandwidth consumption because multiple users can share the locally cached content; lower response times through placing content closer to the consumer; and improved reliability when the central server is unavailable. A CDN can replicate static content, such as web pages, images, and videos, but also dynamic content, such as executable code and computing environments to run applications [63, 65].

Automotive applications, for example, collision avoidance and other driving assistance solutions, and especially self-driving vehicles, require near real-time, reliable processing of data and therefore cannot rely on a connection to the cloud [3, 62]. Additionally, the volume of data captured in an autonomous vehicle, for example by cameras, is too large to be sent to the cloud for processing [46]. Therefore the only choice is to process it at the edge of the network. *Vehicle-to-Everything* communications provide low-latency communications between vehicles, pedestrians, and road-side infrastructure to support building these types of applications, with standardization being pushed by the telecommunications industry [66].

Video analytics is an application domain that has an opportunity to benefit from the bandwidth-saving and privacy aspects of edge computing [46]. For example, video surveillance involves processing video feeds from large numbers of cameras, and sending this data to the cloud can be expensive or otherwise impractical. Applications such as locating missing persons could do image recognition on the camera data at the edge of the network instead; this could alleviate the privacy concerns involved with video surveillance as well.

Augmented reality (*AR*), which integrates virtual objects with the physical world in real-time to create rich interactive environments, involves strict delay requirements [57]. Additionally, wearable and mobile devices used to implement AR systems do not have sufficient capabilities to produce these environments well [67]. Edge computing can be used in AR by enabling mobile devices to extend their computational capabilities with lower delays than what is possible by using the cloud.

2.2.3 Challenges in edge computing for IoT

Research challenges in edge computing have been surveyed in, among others, [4, 46, 62, 57]. In this subsection, those that are still most relevant today are categorized and described.

Programmability For edge computing to be adopted, application developers will need to be able to develop and deploy applications for it. The heterogeneous nature of edge systems makes it particularly challenging to program applications for them: it is cumbersome to develop and tune the application for each type of edge system separately [46, 62]. Frameworks and platforms that abstract the underlying systems could help alleviate the problem. Cloud computing has become successful in part because it abstracts the underlying hardware, making it simple to develop applications on top of it. Ideally, edge development should be made similar to cloud development: the programmer could just focus on the application itself, and the supporting platform would take care of finding the optimal location to run it, on the cloud or the edge.

Pricing and business models For edge computing to be provided as a utility, pricing and business models that are lucrative to customers and profitable to providers are necessary [57]. Edge systems operate on a smaller scale than cloud data centers, making them inherently less efficient. Usage levels are also potentially more variable due to the mobility of users and heterogeneous capabilities of different edge systems. It is challenging to find a pricing model dynamic enough to meet all these requirements while being profitable for the providers.

Resource management Edge computing systems are by their nature decentralized, heterogeneous, and potentially operated by multiple providers, while being used by multiple simultaneous users. Managing these distributed resources in a way that maximizes utilization while sharing the resources fairly between users is a complex problem [62].

Privacy and security Even though edge computing provides opportunities for improving privacy and security, these are some of the biggest issues as well [62]. In edge systems, data is stored and processed on devices that are far more vulnerable to security attacks than cloud servers and may be operated by multiple different owners. How to implement trust in such a distributed system of heterogeneous devices is an open challenge, and trust by users is a necessity for the adoption of the technology.

Encryption is commonly used to prevent eavesdropping while in transit or when stored on disk, but when data is processed, it is generally necessary to access it. A possible solution to the problem is homomorphic encryption, which enables computation to be done on encrypted data so that the original data is not revealed to the computing system [68]. In homomorphic encryption, data is encrypted in a way that the encrypted data has the same mathematical properties as the cleartext. After performing mathematical operations on the encrypted data, the results can be decrypted, giving the same results as if the operations were performed on the original data.

Another security problem is the challenge of verifying that computation done by an edge server is correct and not tampered with by an attacker or a malicious operator. Verifiable computing is a technology that addresses this by having the computation include mathematical proof that the result is correct [69]. Combining verifiable computing with homomorphic encryption would enable outsourcing computations securely to untrusted parties.

2.3 Conclusions

In this chapter, the organization, applications, and current challenges of IoT and edge computing were surveyed. One of the primary challenges holding back IoT applications is the standardization and interoperability of IoT systems, which threaten to limit the ways IoT data can be exploited. Privacy and security issues challenge the trust of users, who may refuse to adopt applications even if they are technically viable, and may lead to regulatory burden. Other major challenges relate to managing the scale of IoT systems: the problem of processing and storing of data, naming and identity, and device management.

Cloud computing is a way to efficiently produce scalable computing resources and is widely used by IoT applications to store and process data. Some of the limitations of cloud computing include network delays and availability of connectivity and sufficient bandwidth. Edge computing is a way to solve these by processing data at the edge of the network instead of in centralized cloud data centers.

One of the primary challenges in edge computing for IoT is the programmability of edge applications and the management of edge resources. These problems may be alleviated with platforms that, for example, make edge application development similar to that of cloud computing. Another major challenge is the development pricing and business models that make edge computing capacity and services lucrative to customers and profitable for producers of such capacity. Finally, edge computing can help to control some of the privacy and security aspects of IoT by decentralizing control of data, but managing privacy and security in deployments of large numbers of heterogeneous devices is challenging.

In addition to helping developers program IoT applications, platforms offer tools for the practical management of IoT systems, and therefore their features are relevant in managing the scalability, privacy, and security aspects of IoT. As they are shared between different applications, platforms are well-positioned to provide solutions for standardization and interoperability as well. Platforms are discussed in more detail in the next chapter, in which currently available IoT edge platforms are evaluated for their capabilities of meeting these challenges of IoT and edge computing.

3 Developing for IoT edge

In this chapter, IoT edge platforms with the most potential to meet the challenges of IoT edge discussed in Chapter 2 are evaluated. In Section 3.1, available IoT edge platforms are surveyed and a subset selected for further review. The organization and capabilities of each selected platform are then reviewed in Sections 3.2 through 3.5. The final part, Section 3.6, contains a comparative evaluation of the platforms.

3.1 IoT edge platforms

A *platform* in product development is a set of common elements that are used as a foundation to build new products and to support and guide the evolution of products [70]. In this work, platforms are defined as software and services on top which IoT applications are built on; this is closely related to the concept of *IoT middleware*, which is software used as a layer in an IoT architecture below the application layer used to support building IoT applications by abstracting underlying technology, such as communication protocols or hardware [1]. *IoT edge platforms* are an extension of this concept: software and services that support building edge computing applications for Internet of Things.

There are over 30 platforms available with over 400 companies offering solutions as IoT platforms, albeit with varying definitions for a platform and some offering more comprehensive sets of features than others [5, 71].

As discussed in Subsection 2.2.1, cloud computing is typically used to produce the capacity to store and process data generated by IoT. It is therefore unsurprising that leading cloud providers offer IoT platform services as part of their extensive service portfolios [72]. Because of their comprehensive IoT offerings and expertise in cloud technology, major cloud providers are in a good position to create comprehensive solutions for extending IoT cloud to the edge.

Of the leading five cloud providers, as determined by market share[2], only Amazon, Inc's AWS and Microsoft Azure have comprehensive edge computing services that are generally

[2]Gartner Says Worldwide IaaS Public Cloud Services Market Grew 31.3% in 2018. https://www.gartner.com/en/newsroom/press-releases/2019-07-29-gartner-says-worldwide-iaas-public-cloud-services-market-grew-31point3-percent-in-2018 (accessed April 15, 2020)

available; these are described in more detail in Sections 3.2 and 3.3. IBM Cloud includes features that support building edge analytics applications[1] and Google sells purpose-built processing hardware for doing machine learning inference at the edge[2] to complement their IoT services, but neither of these constitutes a complete IoT edge platform. Alibaba offers an IoT platform but does not offer services or software that specifically support edge scenarios[3].

Selecting a cloud provider's platform for IoT edge platform further exacerbates the risk of vendor lock-in inherent in using cloud services. These platforms tend to be architected to only integrate the provider's own services, as the provider does not have an incentive to help customers use competing services. This suggests that they might not be a good choice from a long-term interoperability perspective.

Open source software has been suggested as a promising alternative to proprietary platforms for IoT [73]. In open source software, users have access to the source code and receive the right to copy, redistribute, and modify the software [74, pp. 171–188]. Because the model allows making improvements to the software and sharing the improvements, it encourages collaboration and has proven very successful [75]. As they are by their nature not limited to any single vendor, open source platforms are good candidates for solving the interoperability and standardization challenges in IoT.

There are several open source edge computing platforms available today. Linux foundation[4], which supports Linux among several other major open source projects, hosts three: EdgeX Foundry[5], Akraino Edge Stack[6], and Baetyl[7]. EdgeX Foundry is a relatively mature project and has a goal to standardize IoT edge by building a common interoperability framework, and is described in more detail in Section 3.4. Akraino Edge Stack has a focus on mobile edge deployments which might make it useful for supporting MEC and has been left out because it does not aim to be a general platform. Baetyl, originally launched as

[1]Streaming Analytics - Details — IBM.
https://www.ibm.com/cloud/streaming-analytics/details (accessed April 15, 2020)

[2]Edge TPU - Run Inference at the Edge — Google Cloud. https://cloud.google.com/edge-tpu (accessed April 15, 2020)

[3]Introduction IoT Platform— Alibaba Cloud Document Center.
https://www.alibabacloud.com/help/product/30520.htm (accessed April 15, 2020)

[4]Linux Foundation - Supporting Open Source Ecosystems. https://www.linuxfoundation.org/ (accessed April 15, 2020)

[5]Home - EdgeX Foundry. https://www.edgexfoundry.org/ (accessed April 15, 2020)

[6]Akraino - LF Edge. https://www.lfedge.org/projects/akraino/ (accessed April 15, 2020)

[7]Baetyl. https://www.baetyl.io/en/ (accessed April 15, 2020)

OpenEdge by Baidu in January 2019[1], is originally designed to be used to extend Baidu's cloud platform to the edge. Baetyl is left out from this work because the project is still immature and is not progressing rapidly: in the first quarter of 2020, less than 30 code commits were made to its source code repository.

KubeEdge[2] is an open source edge computing platform built on Kubernetes, a popular orchestration system for containers. As discussed in Subsection 2.2.1, containers are a lightweight form of virtualization that is being increasingly used for building cloud-native applications. Extending such applications from the cloud to the edge could be straightforward if existing container-management systems could be used on the edge as well. KubeEdge is described in Section 3.5.

A search for other open source edge computing systems reveals projects such as Eclipse Kura[3], an open source framework for developing edge gateways; StarlingX[4], a low-latency virtualization infrastructure stack for edge infrastructure; and Apache Edgent[5], a programming platform for data analysis on edge systems. These are all been left out from this work because these projects are not generic platforms.

As discussed in this section, the IoT edge platforms selected for review in this work include the proprietary platforms by Amazon's AWS and Microsoft Azure and the open source platforms EdgeX Foundry and KubeEdge. These will be described in detail in the following sections, followed by Section 3.6, in which a comparative evaluation of the platforms is made.

3.2 AWS IoT for the edge

3.2.1 Introduction

Amazon Web Services (*AWS*), a subsidiary of Amazon.com, Inc., is the market-leading provider of cloud services in the world [52]. AWS offers IoT services with its *AWS IoT* service portfolio [72]. *AWS IoT core* services include the basic functionality for integrating

[1]Baidu open-sources OpenEdge, a platform for building edge applications - SiliconANGLE. https://siliconangle.com/2019/01/09/baidu-open-sources-openedge-platform-building-edge-applications/ (accessed April 15, 2020)

[2]KubeEdge. https://kubeedge.io/en/ (accessed April 15, 2020)

[3]Eclipse Kura — The Eclipse Foundation. https://www.eclipse.org/kura/ (accessed April 15, 2020)

[4]Home — StarlingX. https://www.starlingx.io/ (accessed April 15, 2020)

[5]Edgent. http://edgent.incubator.apache.org/ (accessed April 15, 2020)

IoT devices with the AWS cloud. Two additional services are specific for edge computing: *Greengrass* that enables local running of some AWS services and doing machine learning inference on edge devices, and *FreeRTOS*, an operating system for embedded microcontrollers [76].

In addition to being the largest cloud services provider, AWS is also one of the oldest: the services were originally launched in 2002[1] and relaunched in 2006[2], with the introduction of *Amazon Simple Storage Service* (*S3*)[3], an object storage service for web services. In 2018, AWS was estimated to have a leading share of 33 percent of the cloud infrastructure market by Synergy Research Group, and was positioned by Gartner as a leader of Cloud IaaS providers in their magic quadrant with a leading market share [52].

AWS offers a broad range of cloud services in compute, storage, tools, database, networking, and management categories [52]. Some of these services include:

- *Elastic Compute Cloud* (*EC2*), a virtual private server-based compute capacity service,
- *Lambda*, a serverless compute capacity service,
- Simple Storage Service (S3), an object storage service for web services,
- *DynamoDB*, a NoSQL database service supporting key-value and document data models,
- *Kinesis*, a Big Data processing system for real-time data streams [77],
- *Simple Notification Service* (*SNS*), a publish/subscribe messaging service,
- *Simple Queuing Service* (*SQS*), a message queuing service,
- *CloudWatch*, a monitoring service for applications and other resources that are deployed on the AWS cloud or on-premises,

[1]Amazon.com Launches Web Services; Developers Can Now Incorporate Amazon.com Content and Features into Their Own Web Sites; Extends ”Welcome Mat” for Developers. https://press.aboutamazon.com/news-releases/news-release-details/amazoncom-launches-web-services (accessed April 15, 2020)

[2]Amazon Web Services Launches. https://press.aboutamazon.com/news-releases/news-release-details/amazon-web-services-launches-amazon-s3-simple-storage-service (accessed April 15, 2020)

[3]Cloud Object Storage — Store & Retrieve Data Anywhere — Amazon Simple Storage Service (S3). https://aws.amazon.com/s3/ (accessed April 15, 2020)

- *Elasticsearch*, a distributed search engine as a managed service,
- *IoT Analytics*, a service for collecting, processing, storing, analyzing and visualizing IoT data,
- *IoT Events*, a service for detecting and reacting to events from IoT devices and applications, and
- *Step Functions*, a service for building cloud applications by combining other AWS services into serverless workflows.

AWS extended its offerings to include IoT-specific services with the launch of AWS IoT in 2015 as beta[1] and announced general availability later in the same year[2]. IoT edge was added in 2016 as limited preview[3] with general availability the following year[4]. Machine learning inference was added to Greengrass in 2017, and brought to general availability in 2018[5]. A recent addition is a support for containers and data stream management on Greengrass in November 2019[6].

3.2.2 AWS IoT architecture and core services

The general architecture of AWS IoT is depicted in Figure 3.1. AWS IoT core provides the basic functionality to securely connect and communicate with other services in AWS cloud, process and act upon data generated by devices, and for applications to interact with offline devices[7] [72, 78]. In this subsection, the components of AWS IoT core are described.

[1]AWS IoT - Cloud Services for Connected Devices — AWS News Blog. https://aws.amazon.com/blogs/aws/aws-iot-cloud-services-for-connected-devices/ (accessed April 15, 2020)

[2]AWS IoT - Now Generally Available — AWS News Blog. https://aws.amazon.com/blogs/aws/aws-iot-now-generally-available/ (accessed April 15, 2020)

[3]Announcing AWS Greengrass, now in limited preview. https://aws.amazon.com/about-aws/whats-new/2016/11/announcing-aws-greengrass-now-in-limited-preview/ (accessed April 15, 2020)

[4]AWS Greengrass is Now Generally Available. https://aws.amazon.com/about-aws/whats-new/2017/06/aws-greengrass-is-now-generally-available/ (accessed April 15, 2020)

[5]AWS Greengrass ML Inference - Now Generally Available. https://aws.amazon.com/about-aws/whats-new/2018/04/aws-greengrass-ml-inference/ (accessed April 15, 2020)

[6]New - AWS IoT Greengrass Adds Container Support and Management of Data Streams at the Edge. https://aws.amazon.com/blogs/aws/new-aws-iot-greengrass-adds-docker-support-and-streams-management-at-the-edge/ (accessed April 15, 2020)

[7]AWS IoT Core Features - Amazon Web Services. https://aws.amazon.com/iot-core/features/ (accessed April 15, 2020)

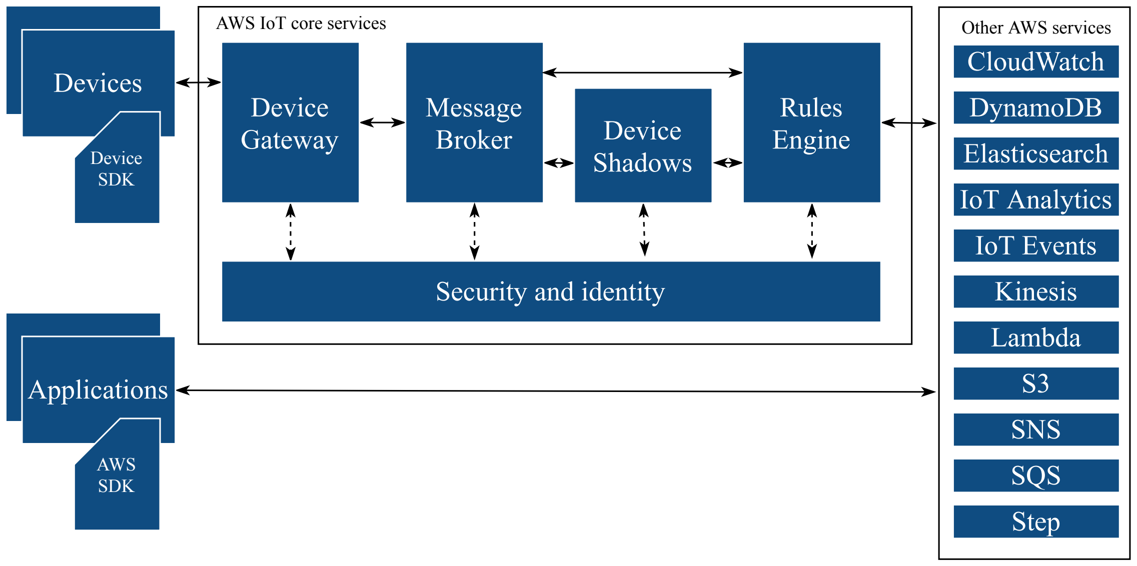

Figure 3.1: AWS IoT architecture. Adapted from How AWS IoT Works - AWS IoT. https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html (accessed April 15, 2020)

Device SDK The *Device SDK* is a collection of libraries, sample code and associated documentation for developers to connect their devices to AWS IoT services[7]. There are separate implementations of the SDK for embedded C, C++, Java, JavaScript, and Python programming languages, and for Android and iOS mobile phone and Arduino Yún microcontroller platforms.

Device Gateway The *Device Gateway* service is the connection point for devices communicating with AWS IoT services using IPv4 or IPv6[7]. The service is fully managed by AWS, who claims it scales up to over a billion devices. The devices connect using a customer-specific domain name allocated under the amazonaws.com domain. However, a feature, in beta at the time of this writing, allows the customer to configure a custom domain name for the gateway, for example, to use an organization's own domain name for the service[1].

Message Broker The *Message Broker* service transmits messages between IoT devices, AWS services, and applications[7]. The service operates using a publish-subscribe model: messages are sent by publishing them on a topic and received by subscribing to it. Because

[1]Configurable Endpoints (Beta) - AWS IoT. https://docs.aws.amazon.com/iot/latest/developerguide/iot-custom-endpoints-configurable.html (accessed April 15, 2020)

of the publish-subscribe model, the devices and services can communicate through the system without knowing who is sending data or receiving it, enabling scalable bidirectional communication with low latency. Clients can communicate with the service using MQTT, MQTT over WebSocket, or HTTPS (REST API) protocol.

Device Shadow The *Device Shadow* service persistently stores the state of a device, enabling the state to be retrieved and manipulated even when the device is temporarily offline[7]. The Device SDK includes functionality to synchronize the device's state with the shadow when the connection is re-established.

Rules Engine The *Rules Engine* receives messages and delivers them to devices or AWS services according to predefined rules[7]. The rules consist of SQL-like statements defining the public/subscribe topics and logical conditions they are triggered on, and a set of actions to perform when the rule is triggered. For example, a rule could react to sensors exceeding a certain threshold value on a set of devices and store the sensor values in a database.

Security and identity The security of AWS IoT has been surveyed in [79]. All devices connecting to AWS must be defined in *AWS IoT registry*[1] that records all the devices and their associated identities and policies. For each connection, the device is authenticated, most commonly using a secure certificate[2]. Other authentication options include using *AWS Identity and Access Management* (*IAM*) identities, which is the standard method of authentication and authorization in AWS services, and *Amazon Cognito Identities*, which is a method of creating temporary identities with limited privilege, for mobile and web applications. Authorization is based on policies that are defined in IAM or mapped directly to certificates[3]. All communication is encrypted using TLS version 1.2 protocol to prevent eavesdropping or tampering while it is transmitted through the network[4].

[1]Register a Device - AWS IoT. https://docs.aws.amazon.com/iot/latest/developerguide/register-device.html (accessed April 15, 2020)

[2]Authentication - AWS IoT. https://docs.aws.amazon.com/iot/latest/developerguide/authentication.html (accessed April 15, 2020)

[3]Authorization - AWS IoT. https://docs.aws.amazon.com/iot/latest/developerguide/iot-authorization.html (accessed April 15, 2020)

[4]Transport Security in AWS IoT - AWS IoT. https://docs.aws.amazon.com/iot/latest/developerguide/transport-security.html (accessed

3.2.3 AWS IoT Greengrass

AWS IoT Greengrass is software for IoT devices capable of running cloud-managed services[1]. With Greengrass, devices can run local application code, such as Lambda Functions, Docker containers, and machine learning inference, and may interact with sensors and actuators and communicate with other devices even when connectivity with the cloud is not available. The software package implementing Greengrass is called *Greengrass Core* (*GGC*).

The GGC software deployment can be over-the-air updated to a new version through the AWS cloud console, API, or command-line interface. AWS provides software called *AWS IoT device tester* for developers to test if their devices have the required hardware and software configuration to run GGC.

Devices, resources and other functionality forming an IoT edge deployment form a *Greengrass group*[2]. A Greengrass group must have at least one device running Greengrass Core, acting as an edge gateway for the group. Other devices, such as IoT devices running AWS IoT Device SDK or FreeRTOS, connect to the edge gateway. AWS provides a REST API, called Greengrass Discovery API, that allows devices to find out their group membership and the address of the core device in the group for establishing connections. Up to 200 devices can coexist in a Greengrass group, and a device may be a member of up to 10 groups.

Communication within the group and with the AWS IoT cloud services is done using the MQTT protocol[1]. If connectivity with the cloud is not available, messages destined to the cloud are temporarily queued; even in this situation, local devices can communicate with each other. Greengrass also supports local Device Shadows, which enable interaction with devices that temporarily cannot be reached. When connectivity to the cloud is re-established, queued messages are delivered and Device Shadows synchronized.

Greengrass supports running AWS Lambda Functions on the edge devices to run application code from the cloud[3]. Lambda Functions are serverless applications that are

April 15, 2020)

[1]What Is AWS IoT Greengrass? - AWS IoT Greengrass. https://docs.aws.amazon.com/greengrass/latest/developerguide/what-is-gg.html (Accessed April 15, 2020)

[2]Deploy AWS IoT Greengrass Groups to an AWS IoT Greengrass Core - AWS IoT Greengrass. https://docs.aws.amazon.com/greengrass/latest/developerguide/deployments.html (accessed April 15, 2020)

[3]Run Lambda Functions on the AWS IoT Greengrass Core. https://docs.aws.amazon.com/greengrass/latest/developerguide/lambda-functions.html (accessed April 15, 2020)

usually run in response to an event, however, Greengrass also supports long-lived Lambda Functions that may run indefinitely. Programming languages supported for the functions include Python, Java, Node.js, C, and C++. *AWS IoT Greengrass Core SDK* is a software development kit for the functions that provides basic functionality such as communicating with the AWS IoT cloud or within the group, interacting with local Device Shadows, or invoking other Lambda Functions. Lambda Functions may also access local resources, such as sensors and actuators on the device, provided the resources are defined in the Greengrass group and access is granted to the function.

AWS IoT Greengrass *Connectors* are pre-built software modules that interact with infrastructure, devices, and clouds, such as the AWS cloud or third-party services[1]. With connectors, logic and integration can be deployed on the edge without having to directly deal with protocols, credentials, or APIs. Connectors exist for connecting with various AWS and third-party services, running Docker containers, doing ML inference on the core device, and communicating with hardware serial ports and Raspberry Pi GPIO pins[2].

Stream manager is an optional component of Greengrass that enables reliable transmission of high-volume IoT data to the cloud[3]. With the stream manager, data is processed locally and transferred automatically to the cloud. Data storage, data retention, bandwidth usage, and timeouts may be specified on a per-stream basis. Streams may also be prioritized to determine the most critical ones to be exported first when a connection is available. Using Greengrass Core SDK, local Lambda Functions can interact with the stream manager, for example, to filter and aggregate data before it is sent to the cloud. Running the stream manager requires Java 8 runtime on the device and 70 MB of RAM in addition to the base requirements of Greengrass Core.

3.2.4 System requirements for AWS IoT Greengrass

Greengrass Core is supported on x86_64, Armv6l, Armv7l, and Armv8l CPU architectures with at least Linux kernel version 4.4 and GNU C Library version 2.14 or later[1]. To run AWS Lambda Functions, runtime libraries for the programming languages used are

[1]Integrate with Services and Protocols Using Greengrass Connectors - AWS IoT Greengrass. https://docs.aws.amazon.com/greengrass/latest/developerguide/connectors.html (accessed April 15, 2020)

[2]AWS-Provided Greengrass Connectors - AWS IoT Greengrass. https://docs.aws.amazon.com/greengrass/latest/developerguide/connectors-list.html (accessed April 15, 2020)

[3]Manage Data Streams on the AWS IoT Greengrass Core - AWS IoT Greengrass. https://docs.aws.amazon.com/greengrass/latest/developerguide/stream-manager.html (accessed April 15, 2020)

required to be present.

At least 128 megabytes of memory is required to be allocated to GGC, or 198 megabytes with the Stream Manager. The disk space requirement is 128 megabytes; this requirement rises to 400 megabytes if over-the-air updates are to be used.

In [80], GGC version 1.5.0 was benchmarked on a Raspberry Pi model 3B computer with 1 GB of memory. The results showed a CPU load of up to 90 percent and memory consumption of under 25 percent in benchmarks performing speech-to-text translation, image recognition, and simple sensor emulation.

3.3 Microsoft Azure IoT Edge

3.3.1 Introduction

Microsoft Azure[1], provided by Microsoft Corporation, is one of the market-leading cloud services in the world. *Azure IoT*[2] service portfolio is the portion of services specific for developing and running IoT applications, with the *Azure IoT Hub*[3] service providing the core functionality for managing IoT devices and connecting them to the cloud. For edge computing, *Azure IoT Edge*[4] service is provided for running workloads on IoT edge devices.

Azure was launched as Windows Azure in 2008[5], providing scalable storage, compute, and networking services. The platform was renamed to Microsoft Azure in 2014[6]. In 2018, Azure was estimated to hold a 13 percent share of the total cloud computing market [52].

Azure service portfolio includes various IaaS, PaaS and SaaS services [52], including but not limited to:

[1]Cloud Computing Services — Microsoft Azure. https://azure.microsoft.com/en-us/ (accessed April 15, 2020)

[2]Azure IoT — Microsoft Azure. https://azure.microsoft.com/en-us/overview/iot/ (accessed April 15, 2020)

[3]IoT Hub — Microsoft Azure. https://azure.microsoft.com/en-us/services/iot-hub/ (accessed April 15, 2020)

[4]IoT Edge — Microsoft Azure. https://azure.microsoft.com/en-us/services/iot-edge/ (accessed April 15, 2020)

[5]Microsoft Unveils Windows Azure at Professional Developers Conference - Stories. https://news.microsoft.com/2008/10/27/microsoft-unveils-windows-azure-at-professional-developers-conference/ (accessed April 15, 2020)

[6]Upcoming Name Change for Windows Azure — Azure Blog and Updates. https://azure.microsoft.com/en-us/blog/upcoming-name-change-for-windows-azure/ (Accessed April 15, 2020)

- *Virtual Machines*, a service providing virtual private servers for running compute workloads,
- *Functions*, a serverless compute service,
- *Storage*, a family of storage services providing scalable storage with multiple different architectures,
- *Event Grid*, a serverless service that allows building event-driven applications by connecting events generated by other Azure services to event handlers, such as Functions,
- *Event Hubs*, a big data streaming service for collecting and analyzing massive amounts of events, such as telemetry,
- *Service Bus*, a scalable messaging service supporting multiple different messaging patterns for applications, devices, and services,
- *Resource Manager*, a service for deploying and managing other Azure services,
- *Stream Analytics*, a real-time analytics service for streaming data using serverless analytics pipelines,
- *Logic Apps*, a service for building automated workflows by connecting services using serverless code, and
- *Machine Learning*, a service for building, training, and deploying machine learning models.

Microsoft announced public preview of Azure IoT services in 2015 with general availability of IoT Hub on February 4, 2016[1]. Edge services were added to the mix when the general availability of IoT Edge was announced on May 11, 2017[1]. Initially, IoT Edge supported local execution of Machine Learning, Stream Analytics, Functions services, SQL Server databases, and Custom Vision[2] image recognition on edge devices. Blob Storage was added to IoT edge on August 9, 2019 and Event Grid on October 28, 2019[1].

[1]Azure updates — Microsoft Azure. https://azure.microsoft.com/en-us/updates/ (accessed April 15, 2020)

[2]Custom Vision - Home. https://www.customvision.ai/ (accessed April 15, 2020)

3.3.2 Azure IoT Hub

The core services for deploying and running IoT systems on Azure and their general organization are depicted in Figure 3.2. Azure IoT Hub provides the basic functionality for provisioning and managing IoT devices, enabling them to communicate with services in Azure, and for backend applications to interact with devices. In this subsection, the core elements of Azure IoT Hub are described.

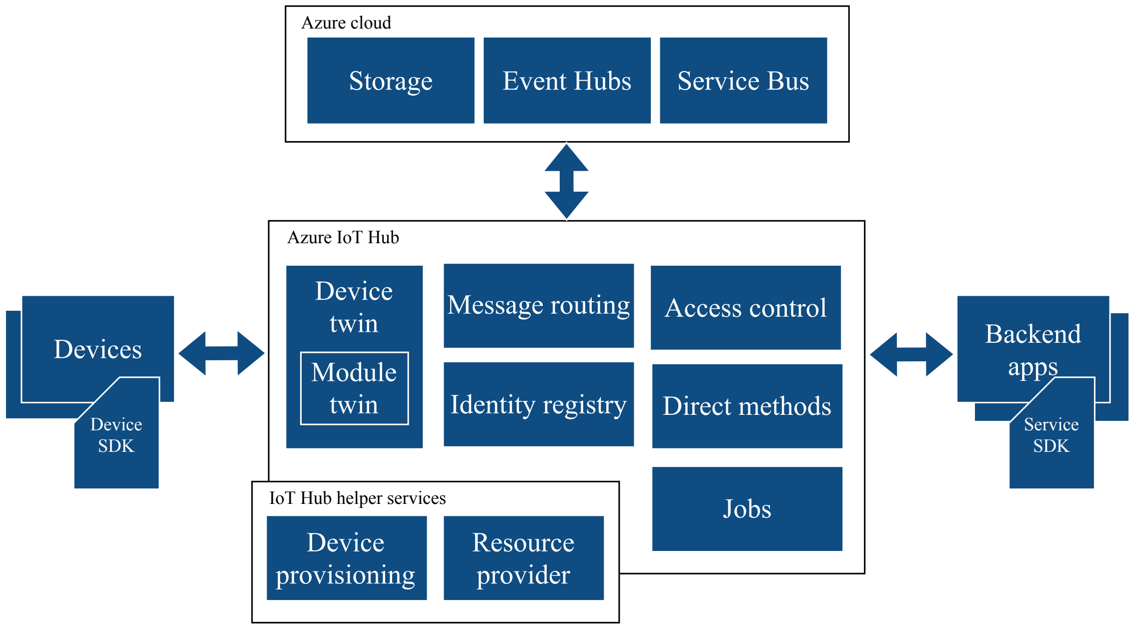

Figure 3.2: Azure IoT core services and organization.

Endpoints *Endpoints*[1] are interfaces exposing IoT Hub functionality to external entities, such as other Azure services, IoT devices, and backend applications. Endpoints include:

- a resource provider endpoint to be used by the Resource Manager to create, delete, and manage IoT hubs,
- endpoints to manage device identities, device twins, and jobs,
- device endpoints for devices to exchange data and receive direct method requests,

[1]Understand Azure IoT Hub endpoints — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-endpoints (accessed April 15, 2020)

- service endpoints for application backends to exchange data, receive monitoring events, and send direct method requests to the devices, and
- custom endpoints for routing messages to Azure services.

Device endpoints support MQTT and AMQP messaging protocols along with HTTPS, while the service endpoints support only AMQP, except for the direct method request endpoint, which uses HTTPS. An additional *IoT protocol gateway* framework is provided for building custom protocol translation gateways to be run on Azure cloud, for devices that cannot support these protocols. IPv6 is not supported; communication over the Internet is only possible using IPv4.

Device provisioning *Device Provisioning Service*[1] (*DPS*) is a helper service to the IoT Hub that simplifies the process of onboarding new IoT devices. With the IoT Hub, the device does not need to be configured with the address of the IoT Hub in advance; instead, it connects to the DPS using HTTPS, AMQP, or MQTT protocol. DPS is configured with a list of devices that may be enrolled in it, and the device's identity is verified using a secure certificate or cryptographic key present on the device. After the device is verified, DPS registers the device on the desired IoT Hub and relays the IoT Hub's connection information to the device, which now may connect to the hub. The use of multiple IoT Hubs and cloud regions is supported for load balancing and high availability.

Resource provider Resource provider[1] is a helper service to the IoT Hub that enables creating, deleting, and updating properties of IoT Hubs. With the resource provider, these management actions may be taken using the Azure Resource Manager[2] through a web-based portal, command line, or programmatically using APIs exposed by the Resource Manager.

Identity registry Information of the identities of all devices permitted to connect to an IoT Hub is contained in an *identity registry*[3]. The information includes a unique string

[1]Overview of Azure IoT Hub Device Provisioning Service — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-dps/about-iot-dps (accessed April 15, 2020)

[2]Overview - Azure Resource Manager — Microsoft Docs. https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview (accessed April 15, 2020)

[3]Understand the Azure IoT Hub identity registry — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-identity-registry (accessed April 15, 2020)

identifying the device, security credentials for authenticating the device, and metadata such as device and connection status information and a timestamp of when the device last was active.

Device twin *Device twin*[1] is a replica of information of a device's properties, such as its configuration, stored for each device connected to an IoT Hub. A device twin enables storing, querying, and modifying this data without connecting to the device itself, thus reducing reliance on a network connection. For example, a backend application can change a property in the twin to a desired value, which is synchronized later when the device reconnects, receives the desired value, and changes its configuration to reflect the new value and reports back. The twin can also be used to store metadata specific to the device, such as its location. A device twin may contain up to 20 *module twins*, which contain similar information to the device twin, to implement multiple separate namespaces within a device and allow finer-grained control.

Direct methods *Direct methods*[2] are requests directed at a device that either execute or fail immediately. These are useful for synchronous device interactions for which the result needs to be acted upon without delay, such as an actuation request initiated from a user interface.

Jobs Timed tasks called *jobs*[3] may be set on an IoT Hub to execute direct methods on devices or update device twin properties. A job may be targeted at a single device or a set of devices using a query string. Jobs are set, for example by a backend application, using an endpoint exposed by the IoT Hub, which then takes care of initiating it at the defined time. The API also allows for querying the state of the execution of a job.

Device SDK *Device SDKs*[4] helps developers build software on IoT devices that communicate with the IoT Hub. The SDKs include libraries and sample code for sending

[1]Understand Azure IoT Hub device twins — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-device-twins (accessed April 15, 2020)

[2]Understand Azure IoT Hub direct methods — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-direct-methods (accessed April 15, 2020)

[3]Understand Azure IoT Hub jobs — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-jobs (accessed April 15, 2020)

[4]Understand the Azure IoT SDKS — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-sdks (accessed April 15, 2020)

telemetry data and receiving messages, jobs, direct method invocations, and property updates from the IoT Hub. Device SDKs are provided in C programming language for Linux, iOS, Windows 10, Mbed OS, Azure Sphere OS, and Arduino platforms; in Python for Linux, MacOS High Sierra, and Windows 10 platforms; in .NET and Node.js for Linux and Windows 10 platforms; and in Java for Android, Linux, and Windows 10 platforms.

Service SDK *Service SDKs*[4] support building backend applications that use the IoT Hub. The SDKs include libraries and sample code to manage the IoT Hub, schedule jobs, invoke direct methods, update device properties, and to send messages to IoT devices. Service SDKs are provided in .NET, Java, Node.js, Python, and C programming languages for platforms supporting these languages, and for the iOS platform.

Message routing *Message routing*[1] allows for routing of device telemetry messages and events from devices to endpoints, and enables filtering data arriving from devices before delivery. Message routes specify the source and destination endpoint of messages. *Queries* may be specified with a message route, using a simple query language, to filter messages based on message metadata and message body provided the body is in JSON format. Routing endpoints include a built-in endpoint to Event Hubs service, and custom endpoints, which may be defined to Azure Storage, Service Bus, and Event Hubs services.

Event Grid integration IoT Hub integrates with the Event Grid service to deliver data generated by IoT devices to the Event Grid[2]. Using IoT Hub as a source for Event Grid events, telemetry data and device events can be collected and analyzed, along with events from other Azure services. Event handlers can be defined to trigger actions on other Azure services, such as Functions, Event Hubs, Logic Apps, and Service Bus. The integration allows for a broader range of services to be used as an endpoint than Message routing.

Access control and security An access control policy specified in the IoT Hub controls the level of access a service or a device has when connecting to an IoT Hub through an

[1]Understand Azure IoT Hub message routing — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-messages-d2c (accessed April 15, 2020)

[2]Azure IoT Hub and Event Grid — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-event-grid (accessed April 15, 2020)

endpoint[1]. Security tokens with a limited lifetime generated by the Device or Service SDKs are used to authenticate devices and services; devices may also authenticate using secure certificates or custom authentication methods. All communications with endpoints are secured using the TLS protocol.

3.3.3 Azure IoT Edge

IoT Edge[2] is a managed service integrated with Azure IoT Hub that enables running cloud workloads on edge devices. It consists of a runtime that runs on the edge devices, modules that contain workloads to be run on the edge, and an interface to integrate the edge deployment with applications and services. In this subsection, these elements are discussed.

IoT Edge runtime *IoT Edge runtime*[3] is a software collection, run on edge devices, that runs edge workloads and manages communication within and from an edge deployment. IoT Edge runtime consists of the *IoT Edge Hub* and *IoT Edge Agent*, which are described in the following paragraphs.

IoT Edge Hub IoT Edge Hub[3] is a module of the IoT Edge Hub that acts as a proxy for the IoT Hub in an edge deployment by exposing the same endpoints as the IoT Hub. Devices and modules that are connected to the IoT Edge Hub can operate and communicate with each other as if they were connected to the actual IoT Hub even though the connection to Azure cloud might be lost. Messages and twin updates are saved locally and synchronized when a connection is reestablished.

IoT Edge Agent IoT Edge Agent[3] is a module of the IoT Edge Hub that initiates modules, keeps them running, and reports their state to IoT Hub. On device startup, a deployment manifest is retrieved from a module twin from the IoT Hub, specifying a list of modules that are to be started. IoT Edge Agent downloads the images of specified modules

[1]Understand Azure IoT Hub security — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-security (accessed April 15, 2020)

[2]IoT Edge — Microsoft Azure. https://azure.microsoft.com/en-us/services/iot-edge/ (accessed April 15, 2020)

[3]Learn how the runtime manages devices - Azure IoT Edge — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-edge/iot-edge-runtime (accessed April 15, 2020)

from a container registry and verifies the contents, starts the modules, and restarts them whenever necessary according to an associated restart policy.

IoT Edge modules *IoT Edge modules*[1] are workload units that can be deployed on edge devices using the IoT Edge Agent. A module is a container compatible with Docker that can contain Azure services or custom code. A *module image*, hosted in the cloud, contains the software of the module and may be downloaded to one or more devices to be run. When a module is created, a *module identity* and a *module twin* is stored on the IoT Hub by the IoT Edge runtime, specifying the module run on the device and its configuration. There are 38 different ready-made IoT Edge modules offered by Microsoft and its partners on the Azure Marketplace[2], providing various edge functionalities: among them are Blob Storage, Event Grid, SQL Server, and Stream Analytics Azure services. Developers of IoT Edge solutions may additionally create their own modules.

3.3.4 System requirements for Azure IoT Edge

Azure IoT Edge can be run on most systems that can run containers[3], such as Windows or Linux. Supported systems include Ubuntu Server, Windows 10 IoT, and Windows Server 2019 on the AMD64 platform and Raspbian on ARM32v7, with Ubuntu Server on ARM64 in public preview. Compatible systems, for which support is not offered by Microsoft, include various distributions of Linux on the AMD64, ARM32v7, and ARM64 platforms.

A container engine based on the Moby project[4] is provided by Microsoft for use as a container engine for IoT Edge. As Docker is also based on Moby, it is compatible with IoT Edge, and may be used alternatively; however, Docker is not supported by Microsoft.

Minimum hardware requirements are not published, as they vary depending on the intended workload. Microsoft claims IoT Edge ”runs great on devices as small as a Raspberry Pi3 to server-grade hardware”[3], implying it would run on devices with as low as

[1]Learn how modules run logic on your devices - Azure IoT Edge — Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-edge/iot-edge-modules (accessed April 15, 2020)

[2]All Products - Microsoft Azure Marketplace. https://azuremarketplace.microsoft.com/en-us/marketplace/apps/category/internet-of-things?page=1&subcategories=iot-edge-modules (accessed April 15, 2020)

[3]Supported operating systems, container engines - Azure IoT Edge. https://docs.microsoft.com/en-us/azure/iot-edge/support (accessed April 15, 2020)

[4]Moby. https://mobyproject.org/ (accessed April 15, 2020)

512 MB to 1 GB of memory capacity. In [80], Azure IoT Edge version 1.4.0 was benchmarked on a Raspberry Pi model 3B with 1 GB of memory. The results showed a CPU load of up to 90 percent and memory consumption of under 25 percent in benchmarks performing speech-to-text translation, image recognition, and simple sensor emulation.

3.4 EdgeX Foundry

3.4.1 Introduction

EdgeX Foundry[1] is a framework for building IoT Edge applications. It consists of software implementing the basic functionality for storing and communicating data on edge devices and managing the edge deployment. The software includes code implementing the core functionality in a layered framework with reference implementations of optional components. Additionally, software development kits (SDKs) are provided for implementing cloud and device integration and data processing. Even though EdgeX Foundry is designed to avoid dependencies on any specific operating system, the software requires a full operating system such as Linux or Windows, restricting it to higher-end devices such as smart gateways.

Many of the components required for a complete IoT system are reference implementations providing rudimentary functionality, such as the microservice that triggers actuation requests from sensor data, or not part of EdgeX Foundry, such as device and cloud integrations. Therefore EdgeX Foundry is not a complete system, but rather a foundation for building such systems. If sufficiently broadly adopted in the industry, it could standardize the architecture of IoT edge systems and potentially foster an ecosystem of organizations building components and applications on top of it, limiting the negative effects of market fragmentation caused by the hundreds of separate platforms currently available.

The EdgeX Foundry project was launched by the Linux Foundation in 2017 [81], to build a common, vendor-neutral framework of interoperable components aimed to fix the lack of standardization in IoT and simplify building solutions for the IoT edge [82].

EdgeX Foundry is open source software, licensed under the Apache 2.0 license[2], one of the most popular open source software licenses [83, pp. 91–33]. The Apache license

[1]Home - EdgeX Foundry. https://www.edgexfoundry.org/ (accessed April 15, 2020)

[2]Apache License, version 2.0. https://www.apache.org/licenses/LICENSE-2.0 (accessed April 15, 2020)

allows for any use of the software, including, among others, the right to create derivative closed-source software from it [83, pp. 91–33]. The only condition is that the license of unmodified parts is retained with original copyright and attribution notices. Additionally, the user receives a license to any software patents that concern the software.

The starting point for EdgeX Foundry was contributed by Dell in 2017 by donating over 125000 lines of code that was created with Dell's Project Fuse, started in 2015 [84, 81]. The initial programming language used was Java.

In October 2017, EdgeX Foundry announced it was moving from Java to Go as a programming language to improve performance, footprint, and scalability[1]. The core portions were rewritten by February 2018[2]. A majority of the components were Go-based in the 0.6 version released in July 2018, with some written in C[3].

EdgeX Foundry was moved under a new umbrella organization, LF Edge, in January 2019[4]. At its launch, LF Edge comprised of five separate projects aiming to support building IoT applications.

By 2019, EdgeX Foundry was a suitable platform for building edge applications with a good device management ability [85], was being used in IoT projects by several companies [84], and had more than 60 members[5].

The first production-ready release of EdgeX Foundry was announced in July 2019[6]. New releases are made twice a year[7], and version 2.1, targeted to be released in April 2021, is tentatively going to be a long term supported version[8].

[1]The Future of EdgeX is Go Go Go with Go Lang. https://www.edgexfoundry.org/blog/2017/10/16/the-future-of-edgex-is-go-go-go-with-go-lang/ (accessed April 15, 2020)

[2]EdgeX Getting Skinny and Fast with the California Code Preview. https://www.edgexfoundry.org/blog/2018/02/27/edgex-getting-skinny-fast-california-preview/ (accessed April 15, 2020)

[3]California - EdgeX Wiki https://wiki.edgexfoundry.org/display/FA/California (accessed April 15, 2020)

[4]The Linux Foundation Launches New LF Edge to Establish a Unified Open Source Framework for the Edge. https://www.lfedge.org/2019/01/24/the-linux-foundation-launches-new-lf-edge-to-establish-a-unified-open-source-framework-for-the-edge/ (accessed April 15, 2020)

[5]Members - EdgeX Foundry. https://www.edgexfoundry.org/about/members/ (accessed April 15, 2020)

[6]EdgeX Foundry Announces Production Ready Release Providing Open Platform for IoT Edge Computing to a Growing Global Ecosystem. https://www.lfedge.org/2019/07/11/edgex-foundry-announces-production-ready-release-providing-open-platform-for-iot-edge-computing-to-a-growing-global-ecosystem/ (accessed April 15, 2020)

[7]Roadmap - EdgeX Wiki. https://wiki.edgexfoundry.org/display/FA/Roadmap (accessed April 15, 2020)

[8]Ireland Release - EdgeX Wiki. https://wiki.edgexfoundry.org/display/FA/Ireland+Release

3.4.2 EdgeX Foundry architecture

The primary design goal of EdgeX Foundry is flexibility, consisting of interoperable and replaceable microservices and being agnostic of the operating system, hardware, protocol, or distribution of components[1]. The flexible design enables it to be run on a variety of devices, and with one of the architectural requirements being the support of both brownfield and greenfield deployments, it aims to be interoperable with existing and future systems. The microservices architecture enables picking and mixing components from different sources with the option of developing your own ones.

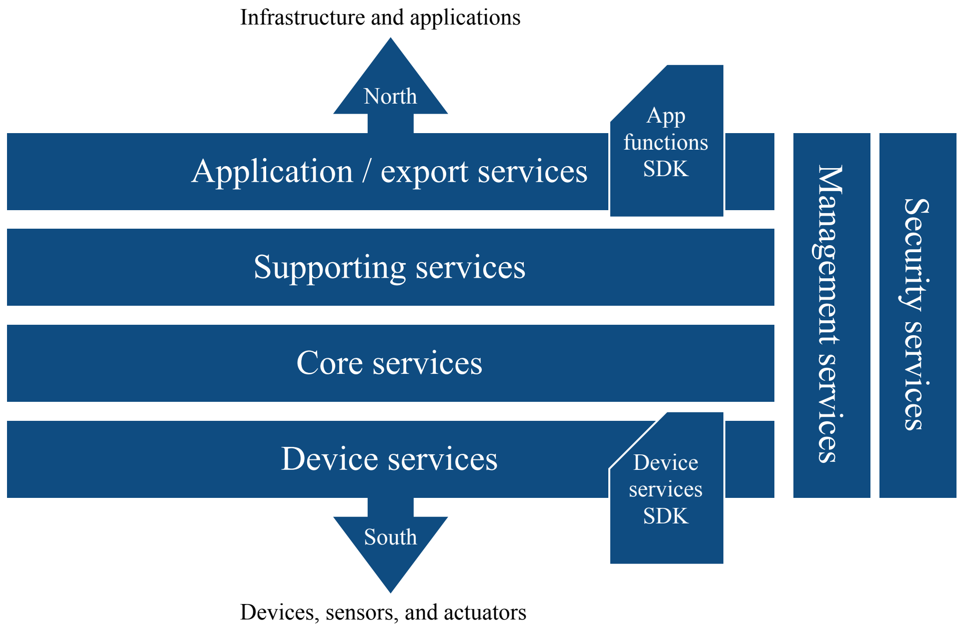

Figure 3.3: EdgeX Foundry architecture. Adapted from About - EdgeX Foundry. https://www.edgexfoundry.org/about/ (accessed April 15, 2020)

The microservices EdgeX Foundry consists of are organized in four service layers and two system layers[1], as depicted in Figure 3.3. In EdgeX Foundry parlance, *north side* stands for the cloud and the network communicating with it, while *south side* stands for things and the devices, sensors, and actuators that interact with things.[1]. Communication between microservices is possible in north, south, and lateral directions.

The layers EdgeX Foundry's microservices are organized into are described in the following chapters:

(accessed April 15, 2020)

[1]EdgeX Foundry Documentation. https://docs.edgexfoundry.org/1.2/ (accessed April 15, 2020)

Core services layer The core services layer consists of the services most fundamental to edge operation, separating the north and south side layers[1]. These include *core data service*, which stores collected data; *command service*, which handles actuation requests; *metadata service*, which manages and stores metadata of the devices and sensors connected to the system; and *configuration and registry service*, which provides the system with information of the deployed microservices and their configuration.

Device services layer and device services SDK The device services layer consists of microservices that directly interact with IoT devices, such as end devices, sensors, and actuators[2]. The device services SDK assists developers in building device services for IoT systems based on EdgeX Foundry by providing the common scaffolding and templates needed by a device service to react to actuation requests and collect data. The device services SDK supports building device services in C and Go programming languages.

The core services layer and the device services SDK together form what the project calls the ”required interoperability foundation”[1]. This refers to microservices that are required components of an EdgeX Foundry deployment: other microservices are provided as a reference implementation, to be replaced with more suitable components whenever necessary as part of developing a complete IoT system based on EdgeX Foundry.

Export services layer The export services layer, which may be used but is deprecated as explained in the next paragraph, contains services for interfacing with infrastructure and applications external to the EdgeX Foundry system[3]. These include *client registration service*, which enables external clients to register as recipients for data generated by the system; *distribution service*, which processes and distributes data to the clients; and Google IoT Core, which sends data to and receives configuration and commands from the Google Cloud Platform.

Application services layer and app functions SDK The application services layer integrates the system with the cloud by transmitting data to and from it, processing and

[1]Core Data - EdgeX Foundry Documentation. https://docs.edgexfoundry.org/1.2/microservices/core/data/Ch-CoreData/ (accessed April 15, 2020)

[2]Device Services Microservices - EdgeX Foundry Documentation. https://docs.edgexfoundry.org/1.2/microservices/device/Ch-DeviceServices/ (accessed April 15, 2020)

[3]3.5. Export Services Microservices - EdgeX documentation. https://fuji-docs.edgexfoundry.org/Ch-ExportServices.html (accessed April 15, 2020)

transforming the data as necessary[1]. It consists of *functions* arranged in a pipeline, where data is processed by separate functions in an order specified by the user. A software development kit, app functions SDK, is provided for developing them. The functions connect directly to the message bus used by core data services to improve performance. This design aims to improve on scalability and flexibility compared to the design used with the export services layer while enabling developers to support cloud solutions of their choosing instead of the EdgeX Foundry project having to implement and track support of numerous cloud providers and their rapidly changing environments. The application services layer replaces the export services layer, which was deprecated with version 1.1 released on October 28, 2019[2].

Supporting services layer The supporting services layers contain microservices that provide intelligence for operating the IoT system based on EdgeX Foundry[3]. These include *rules engine*, which monitors sensor data and triggers actuation commands based on user-defined rules; *scheduling service*, which periodically cleans up exported or stale data; *alerts & notifications service*, which sends notifications to external systems or persons on events detected by another microservice, such as off-limits sensor data or system malfunctions; and *logging service*, which receives and stores log entries generated by other microservices in the system.

Security services layer The security services layer contain microservices that provide services for the secure operation of an IoT system[4]. These include *secret store service*, which securely stores secrets, such as tokens, passwords and certificates, required for other microservices for their operation; and *API gateway*, which acts as a single point of entry for external clients to the REST APIs exposed by EdgeX Foundry microservices, preventing unauthorized access.

[1]Application Services Microservices - EdgeX Foundry Documentation. `https://docs.edgexfoundry.org/1.2/microservices/application/Ch-ApplServices/` (accessed April 15, 2020)

[2]EdgeX Foundry Reaches 1 Million + Platform Container Downloads, Launches New Fuji Release. `https://www.lfedge.org/2019/10/28/edgex-foundry-reaches-1-million-platform-container-downloads-launches-new-fuji-release/` (accessed April 15, 2020)

[3]Supporting Services Microservices - EdgeX Foundry Documentation. `https://docs.edgexfoundry.org/1.2/microservices/support/Ch-SupportingServices/` (accessed April 15, 2020)

[4]Security - EdgeX Foundry Documentation. `https://docs.edgexfoundry.org/1.2/microservices/security/Ch-Security/` (accessed April 15, 2020)

Management services layer The management services layer provides functionality for managing microservices in the system, consisting of the *system management agent* (*SMA*), which provides functionality for controlling and monitoring microservices in the system[1]. The SMA provides an API for starting, stopping, and restarting microservices and extracting operational metrics such as capacity usage and configuration information from them. The API may be used internally or by an external management system.

EdgeX Foundry microservices may be deployed on one node or be distributed on multiple nodes. The microservices are provided, in addition to source code, as automatically built Docker containers for deployment by users[2]. Communication between the microservices is done using REST, except between data and export services, when ZeroMQ, a message bus is used[3].

In an example EdgeX Foundry workflow, sensor data is collected by a device service from an IoT device on the south side. The data is passed to the core services for persistent storage and for delivery to the supporting services and application or export services layers. The supporting services process the data for edge intelligence, such as triggering device actuation based on sensor readings, while the application or export services transform, format, and filter the data and deliver it to the north side cloud infrastructure.

3.4.3 System requirements for EdgeX Foundry

EdgeX Foundry is operating system and hardware agnostic, but it requires a full operating system[2]. The project reports that is has been successfully deployed on various versions of Linux, Windows, and Mac OS X.

As the project is in a rapid development phase, exact minimum hardware requirements have not been defined. Currently, the project recommends a minimum of 1 GB memory and a minimum of 3 GB of storage capacity. A 128-megabyte memory footprint has been reported for a full installation in January 2018[4]. All the microservices do not have to be

[1]System Management Agent (SMA) - EdgeX Foundry Documentation. https://docs.edgexfoundry.org/1.2/microservices/system-management/agent/Ch_SysMgmtAgent/ (accessed April 15, 2020)

[2]Getting Started - EdgeX Foundry Documentation. https://docs.edgexfoundry.org/1.2/getting-started/ (accessed April 15, 2020)

[3]Core Data - EdgeX Foundry Documentation. https://docs.edgexfoundry.org/1.2/microservices/core/data/Ch-CoreData/ (accessed April 15, 2020)

[4]EdgeX Overview 091018 - EdgeX-Overview-091018.pdf. https://www.edgexfoundry.org/wp-content/uploads/sites/25/2018/09/EdgeX-Overview-091018.pdf (accessed April 15, 2020)

installed and run on a single device, but can be placed freely due to the loosely-coupled microservices architecture. Therefore more constrained devices can be a part of the system by running only a limited subset of it, down to only a single microservice.

Required external components to install, run and develop EdgeX Foundry include git, MongoDB, Redis, and ZeroMQ.

3.5 KubeEge

3.5.1 Introduction

KubeEdge[1] is a platform for extending cloud-native applications to edge devices [86]. It is based on Kubernetes[2], an orchestration system for automatically deploying, scaling, and managing cloud-native applications based on containers. KubeEdge includes functionality for managing the containerized application in the cloud and the edge, reliable communications, storing data in the edge and synchronizing it with the cloud, and autonomous operations of edge deployments when cloud connection is unavailable.

Using KubeEdge, existing containerized applications can be deployed on the edge with little to no modification. KubeEdge does not implement Kubernetes itself; instead, it integrates with a Kubernetes installation in the cloud[3]. It also relies on SQLite for data storage and a MQTT broker for communication over MQTT.

Aside from the basic infrastructure for cloud-to-edge communication and managing containers, KubeEdge does not provide other tools required for building IoT solutions, such as device management, data storage, device and sensor communications, edge intelligence, or security tools. Therefore KubeEdge is not a complete IoT platform, but one that can be incorporated into a selected architecture in addition to other elements, such as a public cloud platform.

KubeEdge project was released in 2018 by Huawei[4] [86]. In addition to Huawei, it has attracted contributors from other organizations, such as JingDong, Zhejiang University,

[1]KubeEdge. https://kubeedge.io/en/ (accessed April 15, 2020)

[2]Production-Grade Container Orchestration - Kubernetes. https://kubernetes.io/ (accessed April 15, 2020)

[3]Setup from KubeEdge Installer - KubeEdge Documentation. http://docs.kubeedge.io/en/latest/setup/kubeedge_install_keadm.html (accessed April 15, 2020)

[4]KubeEdge, a Kubernetes Native Edge Computing Framework. https://kubernetes.io/blog/2019/03/19/kubeedge-k8s-based-edge-intro/ (accessed April 15, 2020)

SEL Lab, Eclipse, China Mobile, ARM, and Intel.

KubeEdge is developed in Go[1] and is open source software licensed under the Apache 2.0 license[2], one of the most popular open source software licenses [83, pp. 91–33]. The Apache license allows for any use of the software, including, among others, the right to create derivative closed-source software from it [83, pp. 91–33]. The only condition is that the license of unmodified parts is retained with original copyright and attribution notices. Additionally, the user receives a license to any software patents that concern the software.

The KubeEdge project is evolving rapidly. The initial release in December 2018 provided basic functionality for edge devices, and in March 2019, the cloud deployment part was released[4]. Version v1.0 was released in June 2019, introducing, among other features, running a standalone edge cluster for increased autonomy[3]. As of the time of this writing, many features, such as monitoring and security of edge deployments and high-availability operations for cloud-side components are unimplemented but on the roadmap[4].

3.5.2 KubeEdge architecture

The architecture and components of KubeEdge are depicted in 3.4. KubeEdge consists of two parts: *CloudCore* that runs in the cloud and integrates with Kubernetes, facilitating communication with edge devices and managing edge device metadata and synchronization; and *EdgeCore*, a lightweight agent run on edge devices, managing containerized applications and bridging communications with devices and applications [86]. The components within each part communicate using the Beehive framework, which provides messaging between modules of a distributed system[5]. The rest of this subsection describes the components of CloudCore and EdgeCore.

[1]Start Developing KubeEdge - KubeEdge Documentation. http://docs.kubeedge.io/en/latest/setup/develop_kubeedge.html (accessed April 15, 2020)

[2]Apache License, version 2.0. https://www.apache.org/licenses/LICENSE-2.0 (accessed April 15, 2020)

[3]Releases - kubeedge/kubeedge. https://github.com/kubeedge/kubeedge/releases (accessed April 15, 2020)

[4]kubeedge/roadmap.md at master - kubeedge/kubeedge. https://github.com/kubeedge/kubeedge/blob/master/docs/getting-started/roadmap.md (accessed April 15, 2020)

[5]Beehive - KubeEdge Documentation. http://docs.kubeedge.io/en/latest/modules/beehive.html (accessed April 15, 2020)

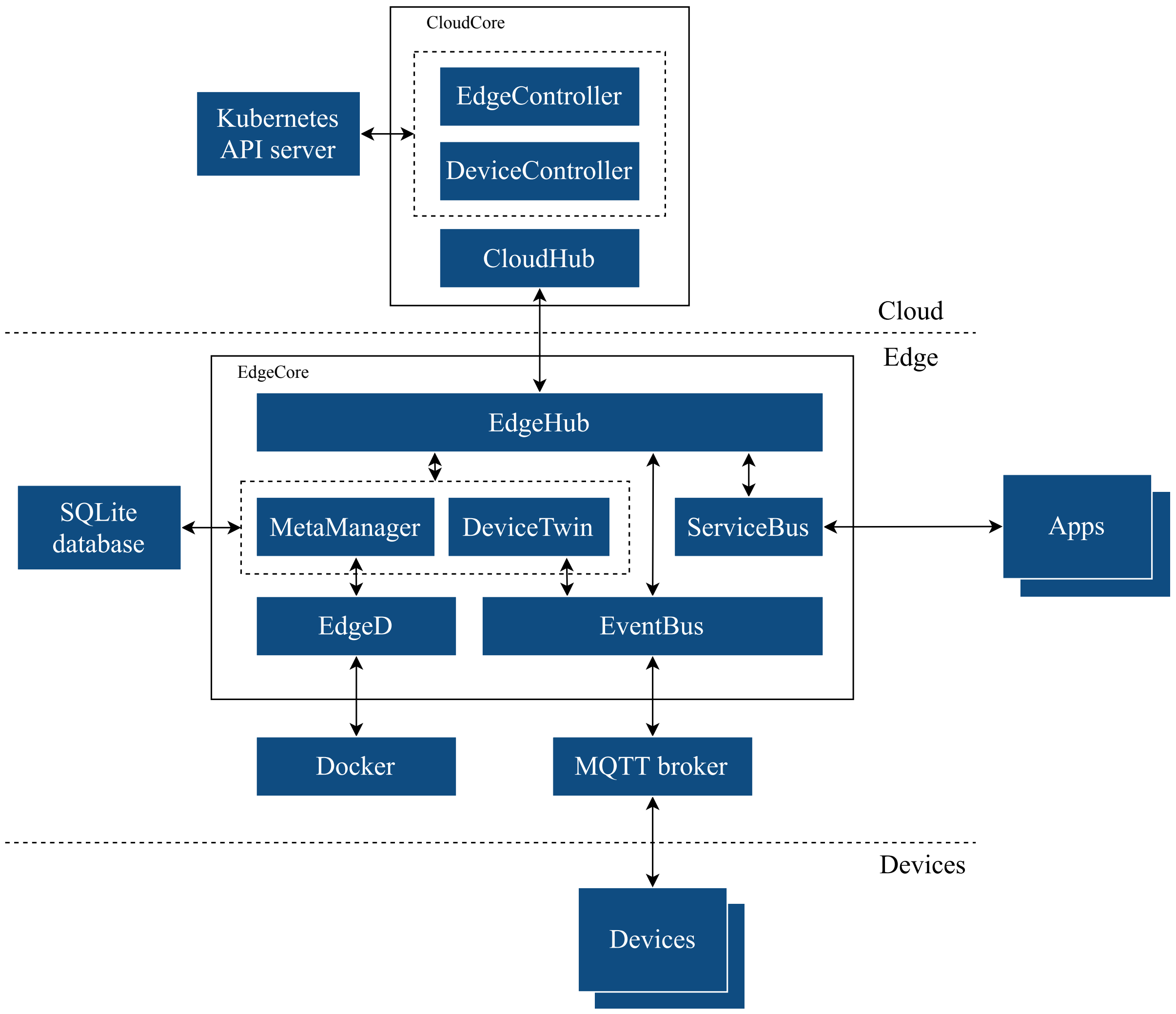

Figure 3.4: KubeEdge architecture. Adapted from What is KubeEdge - KubeEdge Documentation. http://docs.kubeedge.io/en/latest/modules/kubeedge.html (accessed April 15, 2020)

CloudCore components

EdgeController *EdgeController* is a bridge between Kubernetes running in the cloud and KubeEdge[1]. It communicates with a Kubernetes system in the cloud by using standard APIs to connect with a Kubernetes API server, which is the central communication point of a Kubernetes installation. EdgeController relays control and monitoring messages between the Kubernetes cluster and edge devices.

[1]Edge Controller - KubeEdge Documentation.
http://docs.kubeedge.io/en/latest/modules/cloud/controller.html (accessed April 15, 2020)

DeviceController *DeviceController* manages device metadata, such as device status and other properties, by relaying metadata updates between Kubernetes and the edge[1]. It communicates with a Kubernetes system in the cloud similarly with the EdgeController, by using standard APIs. Reported properties sent by the edge devices are delivered to Kubernetes, and desired properties received from Kubernetes are relayed to the edge.

CloudHub *CloudHub* is the cloud-side connection point for edge devices[2]. It listens to and maintains connections from edge devices on a predefined address, and relays messages between CloudCore and the edge. CloudHub supports edge connections using HTTP over WebSocket protocol with TLS encryption or the QUIC protocol. It also sends connection status messages to the DeviceController when connections are established or lost.

EdgeCore components

EdgeHub *EdgeHub* is the edge-side connector for edge devices[3]. It connects to the CloudHub using a predefined address and HTTP over WebSocket with TLS encryption or QUIC protocols and relays messages between other components and the cloud. It also sends periodic keepalive messages to maintain the cloud connection and reports cloud connection status to the other modules.

MetaManager *MetaManager* processes messages related to the management and monitoring of containerized workloads to access this data even when a cloud connection is not available[4]. It receives updates, such as creation or deletion of workloads or status updates, stores them in a SQLite database on the local device, and relays them to the cloud or EdgeD when changes are detected. The stored state is periodically synchronized with container management. MetaManager also serves requests from other components that query for the stored metadata.

[1]Device Controller - KubeEdge Documentation.
http://docs.kubeedge.io/en/latest/modules/cloud/device_controller.html (accessed April 15, 2020)

[2]CloudHub - KubeEdge Documentation.
http://docs.kubeedge.io/en/latest/modules/cloud/cloudhub.html (accessed April 15, 2020)

[3]EdgeHub - KubeEdge Documentation.
http://docs.kubeedge.io/en/latest/modules/edge/edgehub.html (accessed April 15, 2020)

[4]MetaManager - KubeEdge Documentation.
http://docs.kubeedge.io/en/latest/modules/edge/metamanager.html (accessed April 15, 2020)

DeviceTwin *DeviceTwin* stores and processes updates to device metadata similarly to the MetaManager, to enable disconnected operations[1]. It stores device metadata in an SQLite database on the local device, receives updates and queries to the data, and periodically synchronizes the data with the cloud.

ServiceBus *ServiceBus* is an HTTP client for the EdgeCore to enable communicating with application components, such as microservices exposing a REST API, running on the edge[2].

EdgeD *EdgeD* is responsible for managing containerized applications running on the edge device[3]. After receiving management commands from MetaManager, it starts, deletes, or modifies local containerized workloads using Docker management tools. It also monitors the workloads and reports status information to MetaManager and periodically frees up disk space by removing dead containers and unused container images.

EventBus *EventBus* is a bridge between KubeEdge and MQTT publish/subscribe messaging[4]. It functions as a client to a MQTT broker, which is typically running locally on the edge device, and relays messages between the broker and KubeEdge. Through EventBus, IoT devices supporting MQTT protocol can communicate with the KubeEdge deployment.

3.5.3 System requirements for KubeEdge

The KubeEdge project has not released system requirements as of the time of this writing, but the project itself is operating system agnostic and expected to be deployable on any system that can run containers, such as Linux or Windows[5]. A memory footprint of 30 megabytes and storage footprint of 66 megabytes for the KubeEdge system itself has been

[1]DeviceTwin - KubeEdge Documentation.
http://docs.kubeedge.io/en/latest/modules/edge/devicetwin.html (accessed April 15, 2020)

[2]kubeedge/kubeedge: Kubernetes Native Computing Framework (project under CNCF).
https://github.com/kubeedge/kubeedge (accessed April 15, 2020)

[3]EdgeD - KubeEdge Documentation.
http://docs.kubeedge.io/en/latest/modules/edge/edged.html (accessed April 15, 2020)

[4]EventBus - KubeEdge documentation.
http://docs.kubeedge.io/en/latest/modules/edge/eventbus.html (accessed April 15, 2020)

[5]Add hardware requirements for kubeedge - Issue #28 - kubeedge/kubeedge.
https://github.com/kubeedge/kubeedge/issues/28 (accessed April 15, 2020)

reported[4]. KubeEdge was benchmarked on Raspberry Pi in both [86] and [87], showing that the system performs acceptably on such a device.

3.6 Evaluation and conclusions

In the previous sections, four different platforms for IoT edge were reviewed: two proprietary platforms by leading cloud providers, summarized in Table 3.1, and two open source projects, summarized in Table 3.2. This section begins with a discussion on methods for evaluating IoT platforms and finishes with a comparative evaluation of the selected platforms.

	AWS IoT for the edge	Microsoft Azure IoT Edge
Organization	Amazon Web Services (AWS)	Microsoft Corporation
Homepage	https://aws.amazon.com/iot/solutions/iot-edge/	https://azure.microsoft.com/en-us/services/iot-edge/
Project launch	November 2016 (limited preview)	May 2017 (general availability)
Evaluated version	Greengrass Core: 1.10.0 (November 25, 2019)	Edge Hub, Edge Agent: 1.0.9 (March 18, 2020)
Edge features	Device Shadow, Rules Engine, Lambda Functions, Connectors (AWS services, containers, ML)	Device twin, Direct methods, Jobs, Modules (Azure services, containers)
Platforms	Linux (x86_64, Armv6l, Armv7l, Armv8l)	Windows, Linux
Memory footprint	∼128MB	Not published (<256MB)
SDKs	Device SDK (C, C++, Java, JavaScript, Python, Android, iOS, Arduino Yún)	Device SDK (C, Python, .NET, Node.js, Java) Service SDK (.NET, Java, Node.js, Python, C)
Protocols	MQTT, REST	MQTT, AMQP, REST
Security	TLS encryption, identity registry, IAM policies	TLS encryption, identity registry, access control policies

Table 3.1: Overview of reviewed proprietary IoT edge platforms.

	EdgeX Foundry	KubeEdge
Organization	Open source (Linux Foundation)	Open source (Huawei)
Homepage	https://www.edgexfoundry.org/	https://kubeedge.io/en/
Project launch	April 2017	November 2018
Evaluated version	v1.1 (October 28, 2019)	v1.2 (Feb 16, 2020)
Edge features	Core data service, Command service, Rules engine, Alerts & Notifications, System management agent	DeviceTwin, containers
Platforms	Platform agnostic (full OS)	Platform agnostic (full OS)
Memory footprint	Not published (∼128MB)	Not published (∼30MB)
SDKs	Device services SDK (C, Go) App functions SDK (Go)	None
Protocols	REST	REST, MQTT (using broker)
Security	TLS encryption, secret store, API gateway	TLS encryption

Table 3.2: Overview of reviewed open source IoT edge platforms.

Evaluating and comparing these platforms is not straightforward, as different approaches can be taken. One type of evaluation is the one in [88], which evaluates IoT platform features based on defined requirements. The requirements were defined under the assumption that the success of an IoT platform is determined primarily by its support to the application development and operations processes throughout the application lifecycle. A similar type of analysis is done in [73], which surveys the maturity of feature sets and ecosystems of various IoT platforms. That survey used a gap analysis, which assesses the properties of a system in relation to the needs of users, identifies shortcomings, and offers recommendations to address the gaps. The challenge in a gap analysis is in presenting a unified and comprehensive set of requirements to evaluate against, as these are as variable as the various applications of IoT.

Another type of evaluation of platforms is performance evaluation, for example as taken in [89], in which in addition to comparing the availability of desired features in a set of IoT platforms, a quantitative evaluation of performance for each platform using a set of metrics was done. This approach has several limitations: first, these types of measurements are by their nature artificial, and may not reflect conditions in any real application scenario. Second, as noted before, applications are very variable and therefore performance in one scenario may not be predictive of performance in any other. Finally, as the performance of software and hardware technology evolves quickly, performance numbers can become outdated fast.

Performance evaluations may be done using a *benchmark*, which is a standard test used to evaluate the relative performance of computing technologies. For edge computing, an open source benchmark suite called EdgeBench has been created that runs a set of edge computing workloads on a platform so that certain metrics may be measured, such as application response time, resource and bandwidth utilization, and cloud infrastructure cost [80]. Benchmarks offer a straightforward way to measure and compare certain performance parameters of a platform, but care must be taken to evaluate whether these parameters are relevant to the application at hand. A particular problem is in the evaluation of cloud platforms, where the underlying technologies are typically hidden from the customers and may be changed without notification to the users.

Still another way of evaluating IoT platforms is by comparison against a reference architecture, as done in [78]. As there is as yet no single standard architecture for IoT, the various IoT platforms as well take many different architectural approaches which makes comparing them challenging. By mapping the components of different platforms to a reference architecture, the components may then be more easily compared using common abstractions.

As empirical testing is not in the scope of this work, performance testing is not performed. Instead, the platforms are evaluated on how comprehensive they are in terms of features in supporting IoT edge applications and how their architectural and organizatory approaches meet some of the particular challenges of IoT and edge, as discussed in Subsection 2.1.7 and Subsection 2.2.3.

Comprehensiveness of features All reviewed platforms offer basic offline data storage and processing in the edge, but AWS and Microsoft Azure have the broadest and most mature set of functionality for building real-world applications. AWS has a more flexible IoT architecture with more functionality, while Microsoft Azure offers a broader range of cloud services that can be run on edge devices. As they are offered as extensions to the IoT service portfolios of both companies, AWS and Microsoft Azure seamlessly integrate the edge with the respective cloud platforms. EdgeX Foundry has only rudimentary functionality for the edge as most of the services are only reference implementations, and does not have out-of-the-box integration with the cloud except for a deprecated edge-side implementation of Google Cloud integration. KubeEdge is a seamless way to extend cloud applications to the edge if they are containerized and managed by Kubernetes but does not offer any additional functionality in itself.

Platform support All reviewed platforms are designed to be run on full-OS devices, with a memory footprint in the range of 128 to 256 megabytes. Integration with low-end devices is possible, generally via MQTT messaging; other integrations must be programmed by the developer, except for some hardware protocols that are supported out-of-the-box by AWS through Greengrass Connectors. AWS additionally offers FreeRTOS, a real-time operating system for microcontrollers, which can be used as part of an edge deployment. EdgeX Foundry is designed to be architecturally flexible so that all the microservices do not need to be run on a single device, and may instead be split so that lightweight devices can be set up to run only parts of the deployment.

Standardization and interoperability The edge platforms of leading cloud providers, AWS and Microsoft Azure, offer little in the way of fixing the standardization and interoperability problem in IoT. As they have no incentive in implementing interoperability features, they are primarily designed to be used with their respective cloud platforms. Using these cloud services to store and process IoT data carries the risk of vendor lock-in, where switching to another provider is difficult or impossible. EdgeX Foundry, on the other hand, has a published goal to standardize IoT edge, and is as such usable in conjunction with other platforms, such as different cloud platforms; however, the framework does not extend to the cloud and thus does not bring any standardization to cloud-side infrastructure. KubeEdge can be used as a common IoT edge technology for cloud-native applications but is limited in scope when considering the whole IoT infrastructure. EdgeX Foundry and KubeEdge are not tied to any single vendor because they are open source projects. Therefore, they have particular promise in becoming common technologies on which IoT edge may be built by different organizations. For example, open source projects can be used as foundations for several different platforms, standardizing their architecture and features.

Device management Device management features in all reviewed platforms are limited; KubeEdge offers none. AWS and Microsoft Azure have services for registering and maintaining metadata for the potentially large number of devices in an IoT deployment. Particularly interesting are the helper services for device provisioning and resource management, which support automated onboarding and management of large IoT deployments. EdgeX Foundry includes a system management agent, which is designed to extract operational metrics from devices.

Privacy and security All reviewed platforms implement basic security using TLS encryption. AWS and Microsoft Azure have comprehensive identity and access management functionalities with policy-based control that cover their whole platforms. EdgeX Foundry has a service for secure storage of secret data and a rudimentary API gateway for controlling access. KubeEdge has no additional security functionality.

Programmability None of the reviewed platforms are particularly restrictive in the way they can be integrated with applications. The way they support application developers is by SDKs, which include libraries and template code that help such integration. AWS and Microsoft Azure offer SDKs and support for a relatively large set of programming languages and platforms, while EdgeX Foundry has C and Go SDKs. KubeEdge does not offer any SDKs.

In summary, AWS and Microsoft Azure offer the most mature platforms with the broadest set of features out-of-the-box but are limited to their respective cloud platforms. EdgeX Foundry has potential in standardizing IoT architecture in a vendor- and platform-neutral manner, but the included features are still relatively rudimentary. KubeEdge is a simple way to extend Kubernetes-based cloud-native applications to the edge but requires many additional elements to build full-fledged IoT edge applications.

4 Discussion

The promise of IoT is to transform many domains of human activity by creating systems that are more dynamic, more autonomous, and provide new insight into the world. Despite decades of research and development, multiple challenges remain in the way of full adoption of IoT, some of which include standardization and interoperability of IoT applications, scaling the processing and storing of data and management of devices, and privacy and security issues.

Cloud data centers are a way to efficiently produce highly scalable computing resources, and they have been successful in providing data processing and storage platforms for many IoT applications. However, using the cloud for IoT requires transferring the data over the Internet, inherently causing a delay and being limited by network capacity. As the amount of data collected by IoT systems far outweighs projected growth in network capacity, cloud alone cannot answer the question of how to handle IoT data.

Processing and storing data at the edge of the network, edge computing, has been proposed as a solution to the limitations of the cloud. Edge computing can potentially improve response times, reduce network bandwidth usage, allow for data processing when network is down, increase control over data, and reduce energy consumption and cloud costs. Edge computing has been successful in the adoption of content delivery networks for distributing Internet content globally; multiple future applications have been proposed that can benefit from low delays, produce particularly large amounts of data, or require autonomous operation. Examples of applications with such requirements include augmented reality, video analytics, and automotive applications.

Despite its promise and plentitude of research efforts, edge computing has yet to find a killer application to break through as the cloud has. Particular challenges in edge computing include making it easy for developers to create edge computing applications; creating pricing and business models that make it profitable to produce and use edge computing systems; managing distributed and heterogeneous edge resources; and security and privacy issues.

IoT platforms enable and support the development of IoT applications, and are therefore in a position to help solve many of the discussed challenges in IoT and edge computing for IoT, especially when it comes to standardization, interoperability, and programmability. However, the availability of hundreds of different IoT platforms with no common architecture highlights the pressing fragmentation so far in the market.

Few IoT platforms yet offer functionality designed specifically to support edge computing. In this work, four promising platforms for IoT edge have been reviewed: AWS IoT for the edge and Microsoft Azure IoT edge have the most mature feature sets but are tied to their respective cloud platforms, while EdgeX Foundry and KubeEdge are open source platforms that show promise for standardizing the IoT edge architecture, but are very limited in their functionality considering what is required to build IoT applications. All reviewed platforms are relatively recent, introduced between 2016 and 2018, and are progressing rapidly.

Functionalities of the selected IoT edge platforms were reviewed by studying publicly available information and literature on the platforms, primarily documentation offered by the cloud provider or open source project. The results are in Sections 3.2, 3.3, 3.4, and 3.5. The platforms were then analyzed and compared against each other in Section 3.6; an emphasis was put on the potential of the platforms to meet the challenges in IoT and edge computing, as established by studying literature on the topics in Chapter 2.

A major limitation of relying on publicly available documentation as a source is that it does not yield empirical evidence on the practical usability or applicability of the platforms. A lot of the documentation available on the web can be classified as marketing material: it can be reasonably anticipated that at least a portion of the capabilities are exaggerated and limitations not revealed to the reader, as the motivation is to sell the platform to potential users - this is a particular concern with commercial cloud platforms.

A natural opportunity for further research would be to test the platforms in practice. Interesting research questions include such as the relative performance of IoT edge platforms, which can be measured using benchmarks; system requirements and compatibility of IoT edge platforms, which can be studied by installing them on different hardware and operating systems and measuring resource consumption; and programmability of IoT edge platforms, which can be analyzed by studying the developer effort required to develop applications. As the requirements of IoT applications in different domains vary wildly, case studies for each domain would be useful in gaining an understanding of how the platforms would be useful to help implement edge computing.

This work only reviewed platforms that were being offered specifically for IoT edge computing. Conceivably, other IoT platforms could nevertheless be applicable for IoT edge and could be incorporated in a future study. As IoT platforms are rapidly evolving, other platforms will in all likelihood offer edge features in the future.

The primary contribution of this thesis is showing that there are multiple viable platforms available that support the development of IoT edge applications. Additionally, criteria for evaluating IoT edge platforms were established. An evaluation of contemporary IoT edge platforms against these criteria showed that proprietary platforms are more mature, but open source projects are particularly promising for improving the standardization and interoperability of IoT systems. The evaluated platforms are under rapid development and improving in capabilities. In addition to the evaluated platforms, several emerging solutions exist that may become viable in the near future.

Bibliography

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. “The Internet of Things: A survey”. In: *Computer Networks* 54.15 (2010), pp. 2787–2805. URL: https://doi.org/10.1016/j.comnet.2010.05.010.

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. “A view of Cloud Computing”. In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[3] Mahadev Satyanarayanan. “The Emergence of Edge Computing”. In: *IEEE Computer* 50.1 (2017), pp. 30–39. URL: https://doi.org/10.1109/MC.2017.9.

[4] Pedro García López, Alberto Montresor, Dick H. J. Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho P. Barcellos, Pascal Felber, and Etienne Riviére. “Edge-centric Computing: Vision and Challenges”. In: *Computer Communication Review* 45.5 (2015), pp. 37–42. URL: https://doi.org/10.1145/2831347.2831354.

[5] Akash Bhatia, Zia Yusuf, David Ritter, and Nicolas Hunke. “Who Will Win the IoT Platform Wars?” In: *BCG Perspectives* (2017). URL: https://www.bcg.com/publications/2017/technology-industries-technology-digital-who-will-win-the-iot-platform-wars.aspx.

[6] Friedemann Mattern. “From smart devices to smart everyday objects (Extended Abstract)”. In: *Proceedings of smart objects conference*. 2003, pp. 15–16.

[7] Dominique Guinard and Vlad Trifa. “Towards the Web of Things: Web mashups for embedded devices”. In: *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*. Vol. 15. Apr. 2009, p. 8.

[8] E.W.T. Ngai, Karen Moon, Frederick J. Riggins, and Candace Y Yi. “RFID research: An academic literature review (1995–2005) and future research directions”. In: *International Journal of Production Economics* 112.2 (2008), pp. 510–520.

[9] Roy Want, Kenneth P. Fishkin, Anuj Gujar, and Beverly L. Harrison. "Bridging Physical and Virtual Worlds with Electronic Tags". In: *Proceeding of the CHI '99 Conference on Human Factors in Computing Systems: The CHI is the Limit, Pittsburgh, PA, USA, May 15-20, 1999*. Ed. by Marian G. Williams and Mark W. Altom. ACM, 1999, pp. 370–377. URL: https://doi.org/10.1145/302979.303111.

[10] George Roussos. "Enabling RFID in Retail". In: *IEEE Computer* 39.3 (2006), pp. 25–30. URL: https://doi.org/10.1109/MC.2006.88.

[11] Kevin Ashton. "That 'Internet of Things' thing". In: *RFID journal* 22.7 (2009), pp. 97–114.

[12] Arshdeep Bahga and Vijay Madisetti. *Internet of Things: A hands-on approach*. Bahga, Arshdeep and Madisetti, Vijay, 2014.

[13] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. *Vision and challenges for realising the Internet of Things*. European Commission Information Society and Media DG, Mar. 2010.

[14] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future Generation Comp. Syst.* 29.7 (2013), pp. 1645–1660. URL: https://doi.org/10.1016/j.future.2013.01.010.

[15] INFSO D.4 Networked Enterprise & RFID INFSO G.2 Micro & Nanosystems. *Internet of Things in 2020: Roadmap for the future*. May 2008. URL: https://docbox.etsi.org/erm/Open/CERP%2020080609-10/Internet-of-Things_in_2020_EC-EPoSS_Workshop_Report_2008_v1-1.pdf.

[16] In Lee and Kyoochun Lee. "The Internet of Things (IoT): Applications, investments, and challenges for enterprises". In: *Business Horizons* 58.4 (2015), pp. 431–440.

[17] Pallavi Sethi and Smruti R. Sarangi. "Internet of Things: Architectures, Protocols, and Applications". In: *J. Electrical and Computer Engineering* 2017 (2017), 9324035:1–9324035:25. URL: https://doi.org/10.1155/2017/9324035.

[18] Dominik Lucke, Carmen Constantinescu, and Engelbert Westkämper. "Smart factory-a step towards the next generation of manufacturing". In: *Manufacturing systems and technologies for the new frontier*. Springer, 2008, pp. 115–118.

[19] Heiner Lasi, Peter Fettke Peter, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. "Industry 4.0". In: *Business & information systems engineering* 6.4 (2014), pp. 239–242.

[20] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. “Internet of Things for smart cities”. In: *IEEE Internet of Things journal* 1.1 (Feb. 2014), pp. 22–32.

[21] S. M. Riazul Islam, Daehan Kwak, Humaun Kabir, Md. Mahmud Hossain, and Kyung Sup Kwak. “The Internet of Things for Health Care: A Comprehensive Survey”. In: *IEEE Access* 3 (2015), pp. 678–708. URL: https://doi.org/10.1109/ACCESS.2015.2437951.

[22] Hassan Farhangi. “The path of the smart grid”. In: *IEEE power and energy magazine* 8.1 (2009), pp. 18–28.

[23] Songtao Guo, Min Qiang, Xiaorui Luan, Pengfei Xu, Gang He, Xiaoyan Yin, Luo Xi, Xuelin Jin, Jianbin Shao, and Xiaojiang Chen. “The application of the Internet of Things to animal ecology”. In: *Integrative zoology* 10.6 (2015), pp. 572–578.

[24] U.S. Department Of Transportation National Highway Traffic Safety Administration report. *FMVSS No. 150 Vehicle-To-Vehicle Communication Technology For Light Vehicles: premilinary regulatory impact analysis*. Nov. 2016. URL: https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/v2v_pria_12-12-16_clean.pdf.

[25] Antonis Tzounis, Nikolaos Katsoulas, Thomas Bartzanas, and Constantinos Kittas. “Internet of Things in agriculture, recent advances and future challenges”. In: *Biosystems Engineering* 164 (2017), pp. 31–48. URL: http://www.sciencedirect.com/science/article/pii/S1537511017302544.

[26] Jonathan Gregory. “The Internet of Things: Revolutionizing the retail industry”. In: *Accenture Strategy* (2015). URL: https://www.accenture.com/_acnmedia/accenture/conversion-assets/dotcom/documents/global/pdf/dualpub_14/accenture-the-internet-of-things.pdf.

[27] Vlad Coroama. “The Smart Tachograph - Individual Accounting of Traffic Costs and Its Implications”. In: *Pervasive Computing, 4th International Conference, PERVASIVE 2006, Dublin, Ireland, May 7-10, 2006, Proceedings*. Ed. by Kenneth P. Fishkin, Bernt Schiele, Paddy Nixon, and Aaron J. Quigley. Vol. 3968. Lecture Notes in Computer Science. Springer, 2006, pp. 135–152. URL: https://doi.org/10.1007/11748625%5C_9.

[28] Jai Manral. "IoT enabled Insurance Ecosystem - Possibilities, Challenges and Risks". In: *Computing Research Repository (CoRR)* abs/1510.03146 (2015). arXiv: 1510.03146. URL: http://arxiv.org/abs/1510.03146.

[29] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke. "Middleware for Internet of Things: A Survey". In: *IEEE Internet of Things Journal* 3.1 (2016), pp. 70–95. URL: https://doi.org/10.1109/JIOT.2015.2498900.

[30] Anne Hee Hiong Ngu, Mario A. Gutierrez, Vangelis Metsis, Surya Nepal, and Quan Z. Sheng. "IoT Middleware: A Survey on Issues and Enabling Technologies". In: *IEEE Internet of Things Journal* 4.1 (2017), pp. 1–20. URL: https://doi.org/10.1109/JIOT.2016.2615180.

[31] Martin Bauer, Mathieu Boussard, Nicola Bui, Francois Carrez, Christine Jardak, Jourik De Loof, Carsten Magerkurth, Stefan Meissner, Andreas Nettsträter, Alexis Olivereau, Matthias Thoma, Joachim W. Walewski, Julinda Stefa, and Alexander Salinas. *Internet of Things - Architecture: Deliverable D1.5 - Final architectural reference model for the IoT v3.0.* July 2013. URL: https://www.researchgate.net/publication/272814818_Internet_of_Things_-_Architecture_IoT-A_Deliverable_D15_-_Final_architectural_reference_model_for_the_IoT_v30.

[32] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun, and Hui-Ying Du. "Research on the architecture of Internet of Things". In: *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*. Vol. 5. IEEE. 2010, pp. V5–484.

[33] Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T. Campbell. "A survey of mobile phone sensing". In: *IEEE Communications Magazine* 48.9 (2010), pp. 140–150. URL: https://doi.org/10.1109/MCOM.2010.5560598.

[34] Yongsen Ma, Gang Zhou, and Shuangquan Wang. "WiFi Sensing with Channel State Information: A Survey". In: *ACM Comput. Surv.* 52.3 (2019), 46:1–46:36. URL: https://doi.org/10.1145/3310194.

[35] Thamer Altuwaiyan, Mohammad Hadian, and Xiaohui Liang. "EPIC: Efficient Privacy-Preserving Contact Tracing for Infection Detection". In: *2018 IEEE International Conference on Communications, ICC 2018, Kansas City, MO, USA, May 20-24, 2018*. IEEE, 2018, pp. 1–6. URL: https://doi.org/10.1109/ICC.2018.8422886.

[36] Oliver Hahm, Emmanuel Baccelli, Hauke Petersen, and Nicolas Tsiftes. “Operating Systems for Low-End Devices in the Internet of Things: A Survey”. In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 720–734. URL: https://doi.org/10.1109/JIOT.2015.2505901.

[37] Antero Taivalsaari and Tommi Mikkonen. “A Taxonomy of IoT Client Architectures”. In: *IEEE Software* 35.3 (2018), pp. 83–88. URL: https://doi.org/10.1109/MS.2018.2141019.

[38] Shancang Li, Li Da Xu, and Shanshan Zhao. “The Internet of Things: A survey”. In: *Information Systems Frontiers* 17.2 (2015), pp. 243–259. URL: https://doi.org/10.1007/s10796-014-9492-7.

[39] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan R. Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. “The QUIC Transport Protocol: Design and Internet-Scale Deployment”. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*. ACM, 2017, pp. 183–196. URL: https://doi.org/10.1145/3098822.3098842.

[40] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. “Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges”. In: *10th International Conference on Frontiers of Information Technology, FIT 2012, Islamabad, Pakistan, December 17-19, 2012*. IEEE Computer Society, 2012, pp. 257–260. URL: https://doi.org/10.1109/FIT.2012.53.

[41] Felix Wortmann and Kristina Flüchter. “Internet of Things - Technology and Value Added”. In: *Bus. Inf. Syst. Eng.* 57.3 (2015), pp. 221–224. URL: https://doi.org/10.1007/s12599-015-0383-3.

[42] Keyur K Patel, Sunil M Patel, et al. “Internet of Things-IOT: Definition, characteristics, architecture, enabling technologies, application & future challenges”. In: *International journal of engineering science and computing* 6.5 (2016).

[43] Alem Colakovic and Mesud Hadzialic. “Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues”. In: *Comput. Networks* 144 (2018), pp. 17–39. URL: https://doi.org/10.1016/j.comnet.2018.07.017.

[44] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. "Interoperability in Internet of Things: Taxonomies and Open Challenges". In: *MONET* 24.3 (2019), pp. 796–809. URL: https://doi.org/10.1007/s11036-018-1089-9.

[45] Qi Jing, Athanasios V. Vasilakos, Jiafu Wan, Jingwei Lu, and Dechao Qiu. "Security of the Internet of Things: Perspectives and challenges". In: *Wireless Networks* 20.8 (2014), pp. 2481–2501. URL: https://doi.org/10.1007/s11276-014-0761-7.

[46] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. URL: https://doi.org/10.1109/JIOT.2016.2579198.

[47] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. "Xen and the art of virtualization". In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*. Ed. by Michael L. Scott and Larry L. Peterson. ACM, 2003, pp. 164–177. URL: https://doi.org/10.1145/945445.945462.

[48] 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 2015, Philadelphia, PA, USA, March 29-31, 2015*. IEEE Computer Society, 2015, pp. 171–172. URL: https://doi.org/10.1109/ISPASS.2015.7095802.

[49] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David J. Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. "Unikernels: library operating systems for the cloud". In: *Architectural Support for Programming Languages and Operating Systems, ASPLOS '13, Houston, TX, USA - March 16 - 20, 2013*. Ed. by Vivek Sarkar and Rastislav Bodik. ACM, 2013, pp. 461–472. URL: https://doi.org/10.1145/2451116.2451167.

[50] P. Samimi and A. Patel. "Review of pricing models for Grid & Cloud Computing". In: *2011 IEEE Symposium on Computers Informatics*. 2011, pp. 634–639.

[51] Sahar Arshad, Saeed Ullah, Shoab Ahmed Khan, M. Daud Awan, and M. Sikandar Hayat Khayal. "A Survey of Cloud Computing Variable Pricing Models". In: *ENASE 2015 - Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering, Barcelona, Spain, 29-30 April, 2015*. Ed. by Joaquim Filipe and Leszek A. Maciaszek. SciTePress, 2015, pp. 27–32. URL: https://doi.org/10.5220/0005429900270032.

[52] Pranay Dutta and Prashant Dutta. “Comparative Study of Cloud Services Offered by Amazon, Microsoft & Google”. In: *International Journal of Trend in Scientific Research and Development (ijtsrd)* 3.3 (Apr. 2019), pp. 981–985. URL: https://www.ijtsrd.com/papers/ijtsrd23170.pdf.

[53] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Michael Armbrust, and Matei Zaharia. “Above the clouds: A Berkeley view of Cloud Computing”. In: *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28.13 (2009), p. 2009.

[54] Justice Opara-Martins, Reza Sahandi, and Feng Tian. “Critical review of vendor lock-in and its impact on adoption of Cloud Computing”. In: *International Conference on Information Society (i-Society 2014)*. IEEE. 2014, pp. 92–97.

[55] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapè. “On the Integration of Cloud Computing and Internet of Things”. In: *2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014, Barcelona, Spain, August 27-29, 2014*. Ed. by Muhammad Younas, Irfan Awan, and Antonio Pescapè. IEEE Computer Society, 2014, pp. 23–30. URL: https://doi.org/10.1109/FiCloud.2014.14.

[56] Cisco Systems. *Cisco Global Cloud Index: Forecast and Methodology, 2016–2021*. Nov. 2018. URL: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html.

[57] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. “Edge Computing: A survey”. In: *Future Gener. Comput. Syst.* 97 (2019), pp. 219–235. URL: https://doi.org/10.1016/j.future.2019.02.050.

[58] Flavio Bonomi, Rodolfo A. Milito, Jiang Zhu, and Sateesh Addepalli. “Fog computing and its role in the Internet of Things”. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing, MCCSIGCOMM 2012, Helsinki, Finland, August 17, 2012*. Ed. by Mario Gerla and Dijiang Huang. ACM, 2012, pp. 13–16. URL: https://doi.org/10.1145/2342509.2342513.

[59] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Cáceres, and Nigel Davies. “The Case for VM-Based Cloudlets in Mobile Computing”. In: *IEEE Pervasive Computing* 8.4 (2009), pp. 14–23. URL: https://doi.org/10.1109/MPRV.2009.82.

[60] Rodrigo Roman, Javier López, and Masahiro Mambo. "Mobile Edge Computing, Fog et al.: A survey and analysis of security, threats and challenges". In: *Future Generation Comp. Syst.* 78 (2018), pp. 680–698. URL: https://doi.org/10.1016/j.future.2016.11.009.

[61] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. "Network Function Virtualization: Challenges and opportunities for innovations". In: *IEEE Communications Magazine* 53.2 (2015), pp. 90–97. URL: https://doi.org/10.1109/MCOM.2015.7045396.

[62] Wei Yu, Fan Liang, Xiaofei He, William G. Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. "A Survey on the Edge Computing for the Internet of Things". In: *IEEE Access* 6 (2018), pp. 6900–6919. URL: https://doi.org/10.1109/ACCESS.2017.2778504.

[63] John Dilley, Bruce M. Maggs, Jay Parikh, Harald Prokop, Ramesh K. Sitaraman, and William E. Weihl. "Globally Distributed Content Delivery". In: *IEEE Internet Comput.* 6.5 (2002), pp. 50–58. URL: https://doi.org/10.1109/MIC.2002.1036038.

[64] Athena Vakali and George Pallis. "Content Delivery Networks: Status and Trends". In: *IEEE Internet Comput.* 7.6 (2003), pp. 68–74. URL: https://doi.org/10.1109/MIC.2003.1250586.

[65] Michael Rabinovich, Zhen Xiao, and Amit Aggarwal. "Computing on the Edge: A Platform for Replicating Internet Applications". In: *Web Content Caching and Distribution, 8th International Workshop, WCW 2003, Hawthorne, NY, USA*. Ed. by Fred Douglis and Brian D. Davison. Kluwer, 2003, pp. 57–77. URL: https://doi.org/10.1007/1-4020-2258-1%5C_4.

[66] Shanzhi Chen, Jin-Ling Hu, Yan Shi, Ying Peng, Jia-Yi Fang, Rui Zhao, and Li Zhao. "Vehicle-to-Everything (V2X) Services Supported by LTE-Based Systems and 5G". In: *IEEE Communications Standards Magazine* 1.2 (2017), pp. 70–76. URL: https://doi.org/10.1109/MCOMSTD.2017.1700015.

[67] Michael Schneider, Jason R. Rambach, and Didier Stricker. "Augmented reality based on Edge Computing using the example of remote live support". In: *IEEE International Conference on Industrial Technology, ICIT 2017, Toronto, ON, Canada, March 22-25, 2017*. IEEE, 2017, pp. 1277–1282. URL: https://doi.org/10.1109/ICIT.2017.7915547.

[68] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. "Fully Homomorphic Encryption over the Integers". In: *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*. Ed. by Henri Gilbert. Vol. 6110. Lecture Notes in Computer Science. Springer, 2010, pp. 24–43. URL: https://doi.org/10.1007/978-3-642-13190-5%5C_2.

[69] Rosario Gennaro, Craig Gentry, and Bryan Parno. "Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers". In: *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*. Ed. by Tal Rabin. Vol. 6223. Lecture Notes in Computer Science. Springer, 2010, pp. 465–482. URL: https://doi.org/10.1007/978-3-642-14623-7%5C_25.

[70] Carliss Y. Baldwin and C. Jason Woodard. "The architecture of platforms: A unified view". In: (2009). Ed. by Annabelle Gawer.

[71] Hamdan Hejazi, Husam Rajab, Tibor Cinkler, and László Lengyel. "Survey of platforms for massive IoT". In: *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*. IEEE. 2018, pp. 1–8.

[72] Paola Pierleoni, Roberto Concetti, Alberto Belli, and Lorenzo Palma. "Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison". In: *IEEE Access* 8 (2019), pp. 5455–5470. URL: https://doi.org/10.1109/ACCESS.2019.2961511.

[73] Julien Mineraud, Oleksiy Mazhelis, Xiang Su, and Sasu Tarkoma. "A gap analysis of Internet-of-Things platforms". In: *Comput. Commun.* 89-90 (2016), pp. 5–16. URL: https://doi.org/10.1016/j.comcom.2016.03.015.

[74] Chris DiBona, Sam Ockman, and Mark Stone, eds. *Open sources: Voices from the open source revolution*. O'Reilly Media, Inc., Jan. 1999.

[75] Steve Weber. *The success of open source*. Harvard University Press, 2004.

[76] Partha Pratim Ray, Dinesh Dash, and Debashis De. "Edge computing for Internet of Things: A survey, e-healthcare case study and future direction". In: *J. Network and Computer Applications* 140 (2019), pp. 1–22. URL: https://doi.org/10.1016/j.jnca.2019.05.005.

[77] Rajiv Ranjan. “Streaming Big Data Processing in Datacenter Clouds”. In: *IEEE Cloud Computing* 1.1 (2014), pp. 78–83. URL: https://doi.org/10.1109/MCC.2014.22.

[78] Jasmin Guth, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Lukas Reinfurt. “Comparison of IoT platform architectures: A field study based on a reference architecture”. In: *2016 Cloudification of the Internet of Things, CIoT 2016, Paris, France, November 23-25, 2016*. IEEE, 2016, pp. 1–6. URL: https://doi.org/10.1109/CIOT.2016.7872918.

[79] Mahmoud Ammar, Giovanni Russello, and Bruno Crispo. “Internet of Things: A survey on the security of IoT frameworks”. In: *J. Inf. Sec. Appl.* 38 (2018), pp. 8–27. URL: https://doi.org/10.1016/j.jisa.2017.11.002.

[80] Anirban Das, Stacy Patterson, and Mike P. Wittie. “EdgeBench: Benchmarking Edge Computing Platforms”. In: *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018, Zurich, Switzerland, December 17-20, 2018*. Ed. by Alan Sill and Josef Spillner. IEEE, 2018, pp. 175–180. URL: https://doi.org/10.1109/UCC-Companion.2018.00053.

[81] Rajat Kabade. *Open Source FOR You: Linux Foundation develops EdgeX Foundry to standardise IoT*. May 2017. URL: https://opensourceforu.com/2017/05/linux-foundation-develops-edgex-foundry-standardise-iot/.

[82] Swapnil Bhartiya. *InfoWorld: EdgeX Foundry is the solution the IoT world desperately needs*. Apr. 2017. URL: https://www.infoworld.com/article/3192862/edgex-foundry-is-the-solution-iot-world-desperately-needs.html.

[83] Lawrence Rosen. *Open Source Licensing*. Prentice Hall Professional Technical Reference, Feb. 2005.

[84] Brian Buntz. *IoT World Today: EdgeX Foundry Brings an Ecosystem Ethos to the Edge*. May 2019. URL: https://www.iotworldtoday.com/2019/05/03/edgex-foundry-brings-an-ecosystem-ethos-to-the-edge/.

[85] Jiayue Liang, Fang Liu, Shen Li, and Zhenhua Cai. “A Comparative Research on Open Source Edge Computing Systems”. In: *Artificial Intelligence and Security - 5th International Conference, ICAIS 2019, New York, NY, USA, July 26-28, 2019, Proceedings, Part II*. Ed. by Xingming Sun, Zhaoqing Pan, and Elisa Bertino. Vol. 11633. Lecture Notes in Computer Science. Springer, 2019, pp. 157–170. URL: https://doi.org/10.1007/978-3-030-24265-7%5C_14.

[86] Ying Xiong, Yulin Sun, Li Xing, and Ying Huang. “Extend Cloud to Edge with KubeEdge”. In: *2018 IEEE/ACM Symposium on Edge Computing, SEC 2018, Seattle, WA, USA, October 25-27, 2018*. IEEE, 2018, pp. 373–377. URL: https://doi.org/10.1109/SEC.2018.00048.

[87] Halim Fathoni, Chao-Tung Yang, Chih-Hung Chang, and Chin-Yin Huang. “Performance Comparison of Lightweight Kubernetes in Edge Devices”. In: *Pervasive Systems, Algorithms and Networks - 16th International Symposium, I-SPAN 2019, Naples, Italy, September 16-20, 2019, Proceedings*. Ed. by Christian Esposito, Jiman Hong, and Kim-Kwang Raymond Choo. Vol. 1080. Communications in Computer and Information Science. Springer, 2019, pp. 304–309. URL: https://doi.org/10.1007/978-3-030-30143-9%5C_25.

[88] Oleksiy Mazhelis and Pasi Tyrväinen. “A framework for evaluating Internet-of-Things platforms: Application provider viewpoint”. In: *IEEE World Forum on Internet of Things, WF-IoT 2014, Seoul, South Korea, March 6-8, 2014*. IEEE Computer Society, 2014, pp. 147–152. URL: https://doi.org/10.1109/WF-IoT.2014.6803137.

[89] Mauro A. A. da Cruz, Joel J. P. C. Rodrigues, Arun Kumar Sangaiah, Jalal Al-Muhtadi, and Valery Korotaev. “Performance evaluation of IoT middleware”. In: *J. Netw. Comput. Appl.* 109 (2018), pp. 53–65. URL: https://doi.org/10.1016/j.jnca.2018.02.013.