SCTE·ISBE CABLE-TEC EXPO 2018 ATLANTA, GA OCTOBER 22-25

2018 Fall Technical Forum SCTE·ISBE • NCTA • CABLELABS

Edge Compute and Software Life-Cycle Management

Creating Consumer Value and Flexibility

A Technical paper prepared for SCTE•ISBE by

Patrick Goemaere
Chief Architect Cloud Services Connected Home
Technicolor
942 Birmingham Rd 91504 Burbank US
+1 (818) 442 7183
Patrick.goemaere@technicolor.com

Rajat Ghai
VP Wireless & Open Networking
Technicolor
rajat.Ghai@technicolor.com

Table of Contents

List of Figures

Introduction

Over the years, we have witnessed a fundamental shift in how software gets developed and deployed in the cloud. Born-in-the-cloud web companies have moved to a much more agile way of working where continuous deployment and integration became the norm, and where these companies are able to introduce new functionality and features on a daily basis, with fine grained control to limit the exposure of these feature first to a subset of their customer base using deployment strategies like canary releases or a derivate like blue/green and other variants.

This transformation has helped these companies to create a competitive advantage so that they can rapidly check the viability of new functionality, collect user feedback and respond quickly to changing requirements. The introduction of full automation and DevOps methodology, has furthermore improved the complex social interaction that typically has plagued many complex product and service development in large scale organizations by breaking up traditional boundaries between product and business owners, R&D, IT and operations so that products can quickly move from business requirements to deployment and has allowed more flexibility in trying out new ideas in a fail fast way of working.

The innovation around Internet of Things (IoT) has built further on the evolutions started in the Web and mobile domain on providing the foundational technology building blocks on how to communicate with embedded devices at web scale, and in order to keep solutions viable, has pushed the boundaries of these technologies further and further to the edge of the network.

This evolution of decentralization has blurred the traditional demarcation between operational embedded technologies and IT technology. On top of these newly cloud native implementations of ultra-web scale communication/messaging infrastructure services, a new generation of services like software life cycle management (LCM), configuration management and orchestration are being developed, that borrows the same concepts as their well-established cloud DevOps counterparts but addressing better the enormous scale of IoT deployments, and their very heterogeneous and fragmented nature.

Our current way of working in the embedded software device space differs today fundamentally from this agile cloud methodology, due to the very nature of these devices themselves. Because of their very constrained nature in terms of CPU power, memory and storage, hard real time requirements and cost structure, these solutions tend to be optimized for very specific functions leading to monolithic firmware development running in specialized execution environments, typically stripped-down Linux distributions. Embedded devices lack the uniform compatibility environment that has been addressed in the cloud by virtualization of compute - either virtual machines (VM's) or containers, and networking. Typically, these development cycles are complex, lengthy, and require lots of validation around industry specific certifications, interoperability and standardization. Furthermore, the reluctance of introducing operational impact on the installed base hinders the deployment frequency.

This paper will focus on a subset of device types used today in the Cable industry as Customer Premise Equipment (CPE) equipment, more notably internet gateway's and set-top boxes and their very specific nature, and tries to identify the current container technology landscape and their applicability for these types of embedded CPE devices.

It proposes a dual track approach with respect to separating out critical firmware functions and more agile customer facing functionality, avoiding a big bang approach and creating an architectural evolution path for the current existing legacy deployments.

It will further elaborate on the evolution of Internet of Things technology of the last years and it's evolution from a centralized approach, towards a decentralization of this architecture towards the Edge of the network and the profound effects it has on improving user experience and customer intimacy for the next generation of IoT services, and the disruption it causes on the entire value chain.

Content

1. Compute Technology Cycles

Computing architectures have always been optimized for most efficient workload placement economics

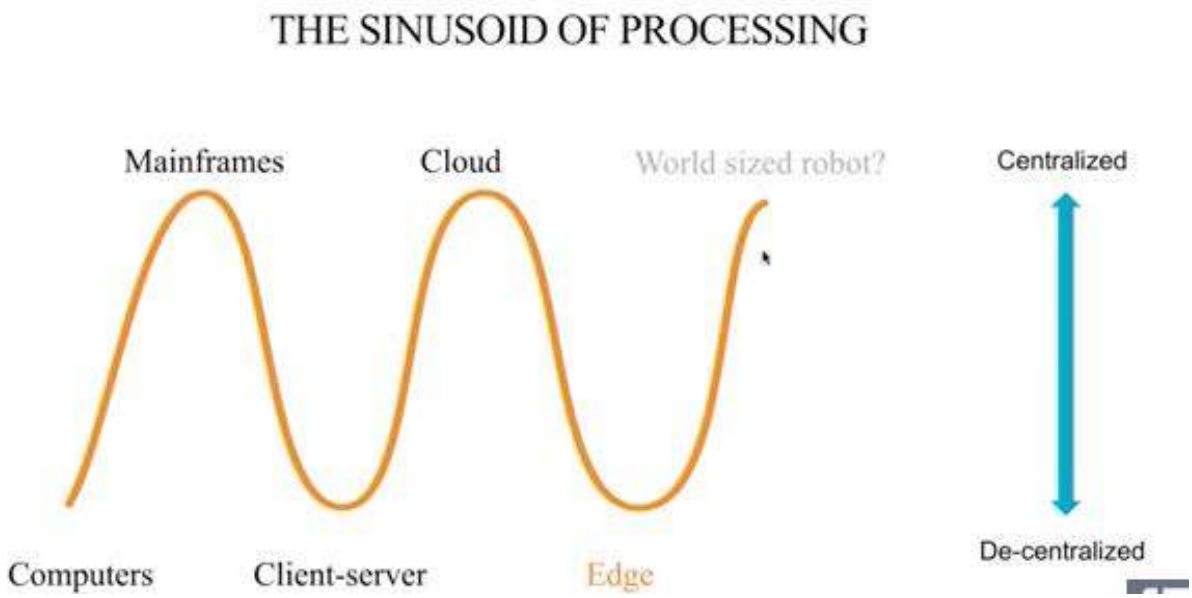

Figure 1 - The Sinusoid of Processing

First Wave

At the dawn of computing, a centralized structure in the 'mainframe' age was most economical due to very expensive compute and transport cost structure. This was the pre-microprocessor era of mainframes and mini computers built using Solid State semiconductor Devices leveraging the invention of the Transistor in late 1940s.

Second Wave

Computing architecture changed in early 70s with the invention of the Microprocessor and the arrival of Personal Computers; Moore's law kicked in and the cost efficiency economics moved to a distributed 'client-server' model. In this golden age of personal computing, which lasted for the two decades (1980s and 1990s), consumers and enterprises adopted local premise computing, primarily because the cost of computing was significantly lower than the cost of Internet transport. Internet speeds in those decades were slow and expensive. This is also considered the time associated with the dawn of public Internet. Internet Service Providers (ISP's), notably Multiple System Operators (MSO's), made large public/private investments which led to faster and more affordable Internet connectivity.

Third Wave

At the turn of the 21st century, with the arrival of innovations and investments in affordable high-speed wireline and wireless Internet access, computing architectures evolved once again. With both computing and Internet access being now affordable, cloud computing was born. Cloud computing (though similar to a centralized design of Mainframe) was a centralized co-location of compute, memory and storage clusters offered as Infrastructure as a Service (IaaS) by public cloud providers like Amazon, IBM, Microsoft and Google, followed by a long tail of smaller providers. Cloud computing for enterprise workloads created immense cost efficiencies in enterprise IT. Cloud computing has given rise to a radical transformation of traditional enterprise technology over the last 10 years. Due to the fully distributed and virtualized nature of cloud, a new generation of cloud native solutions have arrived, architected to cope with resilience in a world where failure is the norm and moved to horizontal scalability using commodity hardware instead of scaling vertical using expensive hardware.

Fourth Wave

Now, in 2018, computing is at the doorstep of another epoch. Computing has become ubiquitous and miniaturized to a point that it can be embedded in virtually every physical object. At the same time wireline and wireless Internet connectivity continues to follow economies of scale. This has led to a new universe of 'smart things' (*things that can compute and can network*). This radical shift is referred to as the Internet of Things (IoT). The world has started utilizing IoT for the digital transformation of the businesses and society as a whole. Every known business process is being digitized, automobiles are becoming autonomous vehicles, Internet connected smart sensors are helping automate the world around us and the society as a whole, is becoming digital as a result. It is expected that by 2025, there will be up to 25 Billion Internet connected Smart Things, far exceeding the number of humans on this planet.

As sensor technologies and networked services pervade every industry, the gravity of *so much data generated by so many diverse endpoints* renders centralized computing topologies inadequate.

1.1. From the Cloud to the Edge

In vital industrial sectors like agriculture, aerospace, mining, healthcare, or energy, reliable connectivity can be sparse; devices and equipment generate massive amounts of data chatter placing unnecessary demand on networks and battery life; failure can cost significant revenues, time, even lives. Because the cost of bandwidth is not falling as fast as the costs of compute and storage, the ability to aggregate and filter data *locally on the device* also serves an economic justification: edge compute decreases reliance on bandwidth. Furthermore, when critical real-time decision-making is required (think aircraft in-flight or self-driving cars), latency to the cloud is simply unacceptable. In consumer markets, current IoT implementations have fallen short of expectations—unmet needs for interoperability, inadequate monetization for providers and poor user experience (UX) for end users. Moreover, privacy concerns stifle consumer adoption and related, if highly variable, regulatory compliance requirements mean some data are simply better left local on the device.

1.2. Agility at the Edge, Learning in the Cloud

Instead of relying purely on cloud infrastructure to process data, local compute 'at the edge' enables greater speed, flexibility, security, privacy, economy, scale, and most importantly *learning*. To enable this, organizations need the tools to aggregate, define, and filter data in order to process lower-value data locally, while uploading high-value data back to the cloud for analytics, innovation, and more sustainable storage and data management. They need partners to enable intelligence in the body (edge), while taking

greater advantage of intelligence of the brain (cloud). The following picture shows the balancing of different analytic functions amongst Edge, Fog, and Cloud.

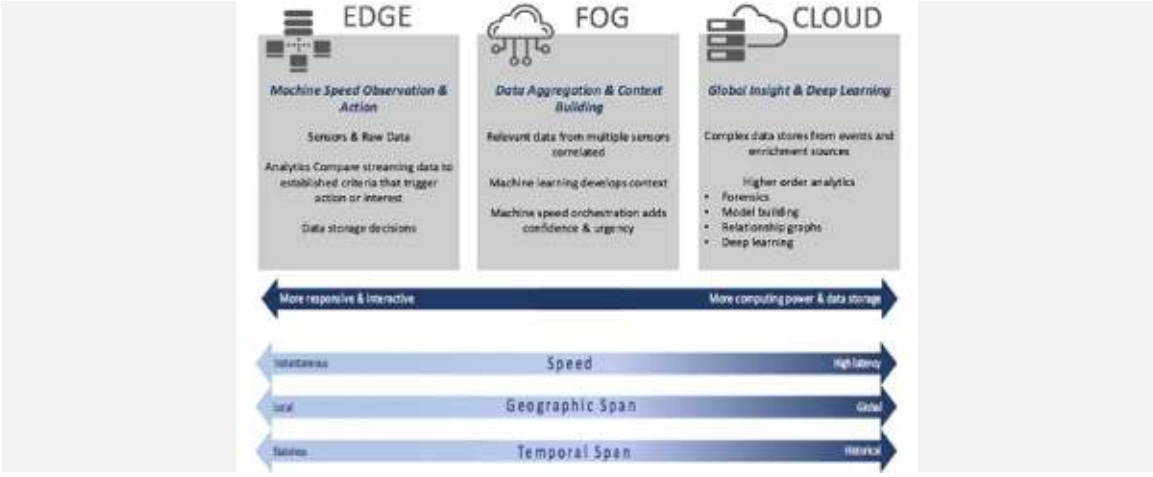

Figure 2 - Analytics Functions And Characteristics

1.3. Edge Compute Financial Model

Shown below is a case study (Financial model Edge compute, n.d.) done by a research firm (Floyer) for a wind farm site equipped with 100 sensors and two video streams. It compares the total cost of managing and processing on three different kinds of architectures:

1. cloud-only processing with a dedicated network,
2. A Mobile Network Operator (MNO) cellular network with hardware and cloud processing, and
3. An edge-and-cloud processing with a dedicated network.

In the first scenario, the three-year Total Cost of Ownership (TCO) of the solution including cost of transmitting the data, plus cloud costs and equipment costs, worked out at **$254,552** /yr.

In the second case the three-year TCO for a MNO hosted solution was less than 50% of the first scenario came to **$113,884/yr**.

However, with a combined edge-computing and cloud approach the 3-year TCO dropped to 15% of the first scenario and 33% of the second scenario, to just **$37,628/yr**.

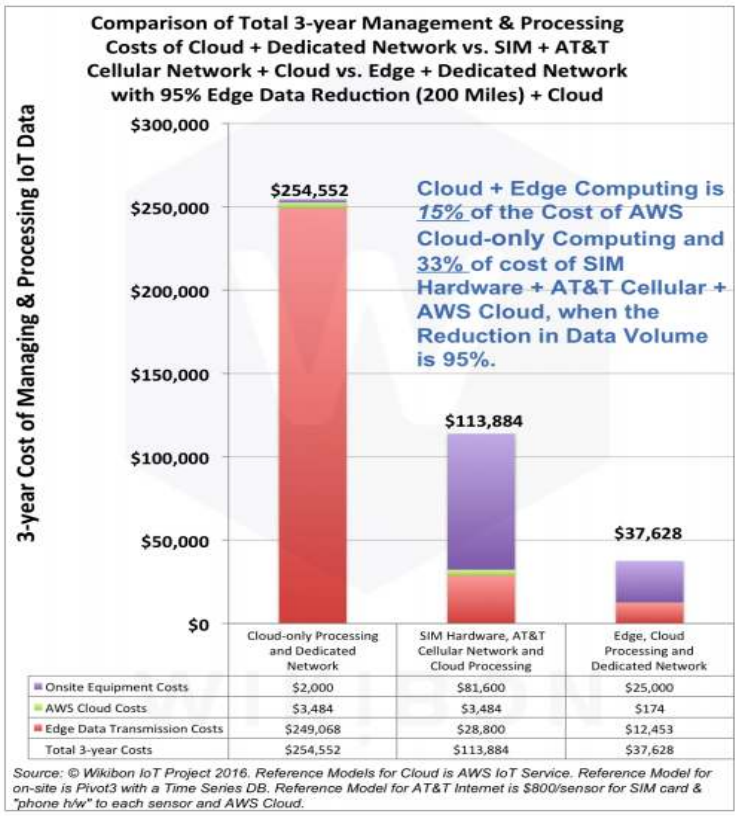

Figure 3 - Financial Model Edge Compute

As shown in the case study, the economics work best when Edge computing is not used to replace centralized cloud-based infrastructure in its entirety, but rather to complement it by increasing the computing and storage resources available on the edge by adopting platforms that provide intermediate layers of computation, networking, and storage and offloading light weight computational workloads locally at the edge and executing computationally intensive workloads (e.g. machine learning workloads that require Graphics Processing Unit (GPU) etc.) in the centralized cloud.

1.4. The Edge : A New Frontier of Next Generation Services & Experiences

While the move to the edge was triggered by the practical realities of the physical world and the tactical needs for real-time data processing, the broader implication of this shift is its role as a force multiplier of new services and experiences. New use cases abound for both end user and enterprise adopters.

1.4.1. MSO Edge Computing use cases: Residential

Seamless Plug & Play: Activating new devices and connected services is finally easy, really easy. When gateways, for example, are voice-supported and equipped with automated detection of nearby devices, users' onboarding experience is a matter of simply 'powering-on' and speaking. Behind the scenes, the edge-enabled gateway handles the rest, automatically provisioning devices, finding Wi-Fi, identifying and executing updates, and activating new services.

Autonomous Systems Administration: Instead of forcing users to become IT systems admins, edge compute-enabled devices handle the cumbersome but critical burdens of performance and network monitoring, anomaly detection, identity authentication, permissions, and encryption, even facilitating better connectivity and quality of service by automating channel changes, roaming decisions, and Wi-Fi optimization.

User-Centric Security & Privacy: Finally, a technology that allows users to have agency and peace of mind around their most sensitive data. When devices are equipped with access management, encryption, and security capabilities right out of the box, data is never exchanged between products without authenticating identity. Moreover, businesses and users can easily configure data filtering and transmission rules, so that highly sensitive data can remain private, local to the gateway, and not sent to the cloud.

Single-Stop Service Hub: When a gateway is built with onboard compute capabilities powerful enough to coordinate, monitor, configure, and control multiple devices, new modes of self-service and support open up. Through voice-interaction, users can simply speak to control and configure their own network services, and solicit support requests through a convenient single-stop self-service portal.

'Getting Better All the Time': The opportunity for innovation enabled through edge-level intelligence means products gain in value over time. For end users, this translates to a single hardware investment which continuously delivers new features and services, constantly learns and adapts to user speech patterns and preferences and evolves to support new use cases through open interoperability with any other device… on the body, in the home, or on the go.

1.4.2. MSO offered Edge Compute Enterprise – B2B use cases

Automatic Implementation: When in-field devices are outfitted for more robust local processing, enterprises enjoy a streamlined device installation process. Devices can automatically provision, certify, pair, and register themselves into Managed Service Provider (MSP) and Customer Relationship Management (CRM) systems.

Automatic Service Activation: These same out-of-the-box configurations mean enterprise user registration, certification, and customer service activation are 'baked in' to the onboarding process. Such a gateway, for example, can automatically connect and troubleshoot to onboard nearby devices, and offer voice-enabled pairing for additional devices.

Managed IoT Cloud Services: Edge-level compute opens up new ways service providers can add value to their partners and customers through managed services. Real-time monitoring of multi-device performance, application performance, network activity, security features (like vulnerability and intrusion detection), and device compliance, eases the burden for enterprises and allows them to focus on their core competencies. Service providers can also use such a gateway to manage new feature releases, API improvements, firmware, and other updates so innovations are securely folded into the broader device ecosystem. These capabilities render it economically feasible to create true end-to-end, personalized, and secure contextual experiences with end-users.

Integrated Support Services: Enterprises' product and support programs also benefit from edge devices' ability to run preventative maintenance such as patch management, bug-fixing, anti-virus updates, and better manage incidents through remediation, customized help desks, and integrated workflows and analytics via Customer Relationship Management (CRM) integration. Management in areas like compliance and refurbishment also help with end-of-life support.

Unified Analytics: When devices are enabled to interact in real-time, enterprises can parse and batch data to prioritize agility, while still gathering insights across data sets to learn over time. Integration with Enterprise Resource Planning (ERP), CRM, Finance, and other systems doesn't just enable enterprises to unify analysis and machine learning with a single dashboard, but also facilitates new use cases like predictive maintenance, automatic hardware replacements, and invisible improvements to performance efficiency.

Baked-In-Innovation: In the construct of balanced intelligence, where edge devices handle agile processing and new features, and deeper learning and data mining occurs in the cloud, enterprises are poised to support improvement across every level of the product (and therefore *customer)* lifecycle. Onboarding is improved over time as machine learning helps deliver better, more proactive support and troubleshooting to offload call center interactions. Product and service performance gain in efficiency as analytics and machine learning reveal areas to optimize such as connectivity, data transmission, and compute resources. Most importantly, such a construct embeds innovation into service architecture, wherein open development environments enable new and diverse services to flourish

2. Edge Compute and decentralization

2.1. IoT platforms

Designing, implementing, securely operating, managing and maintaining IoT projects are complex. In fact, there are entire organizations whose sole mission is solving a specific problem within an IoT architecture. The problems that can be found within such architectures can range from connectivity to figuring out where apps live.

Here are just some examples:

- Solving the daunting challenge of how to connect various very fragmented sensor protocols to the Internet.
- Handling intermittent connectivity, let alone trying to update an application over the air or patch a remote edge system
- Predicting when something is going to break before it happens
- Correlating telemetry with contextual data to generate better models
- Storing different information in different databases (NoSQL, time series, in memory database)
- Making sure your architecture is scalable, flexible, and can be deployed anywhere
- How to abstract your developers from where their applications will be running in this chaotic, highly distributed world

But these are only pieces to the bigger puzzle of designing an end-to-end IoT system where connected devices, business processes, DevOps practices, and people collaborate together across four different, complementary, but most-of-the-time disconnected, worlds:

- Operational Technology (OT)
- Information Technology (IT)
- Analytics and Machine Learning (ML)
- Traditional and modern application development

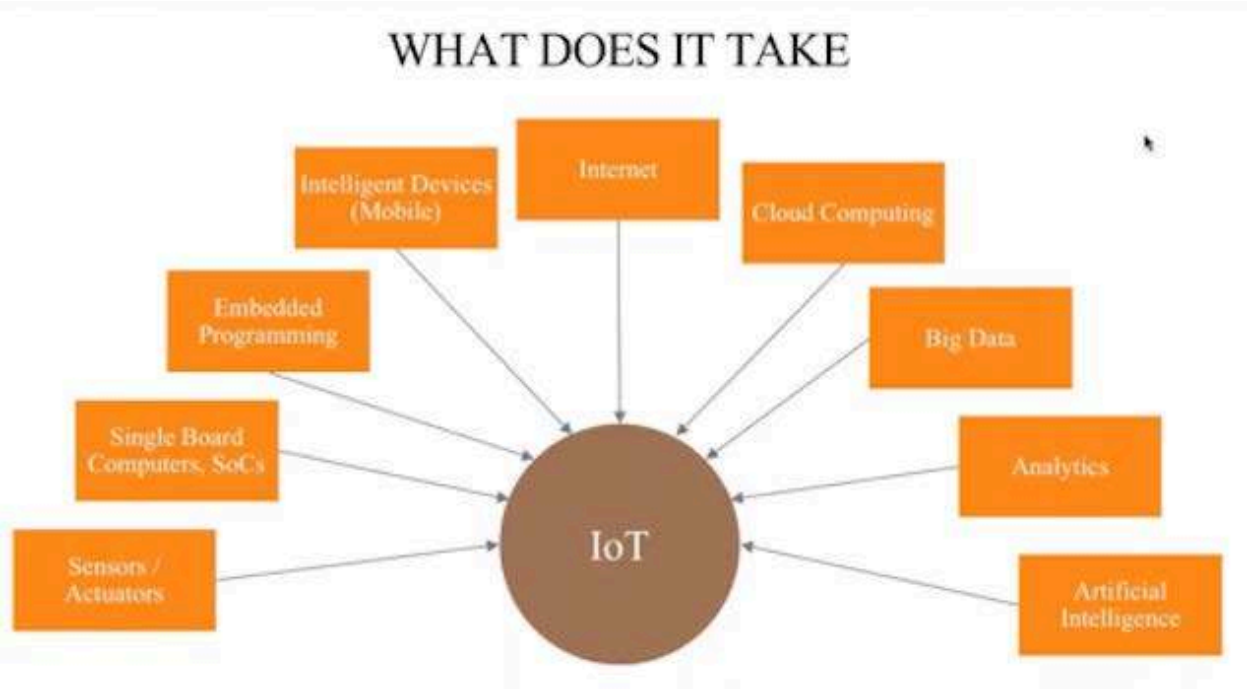

Figure 4 - Technology Convergence

Connecting devices; ensuring communication protocols are translatable; verifying accuracy and security features are running across the entire network; capturing, managing, analyzing, and using data to create better business outcomes; integrating operational data with existing informational technology systems and applications; and gaining intelligence at the edge are all parts of the functionality of an IoT architecture as depicted below:

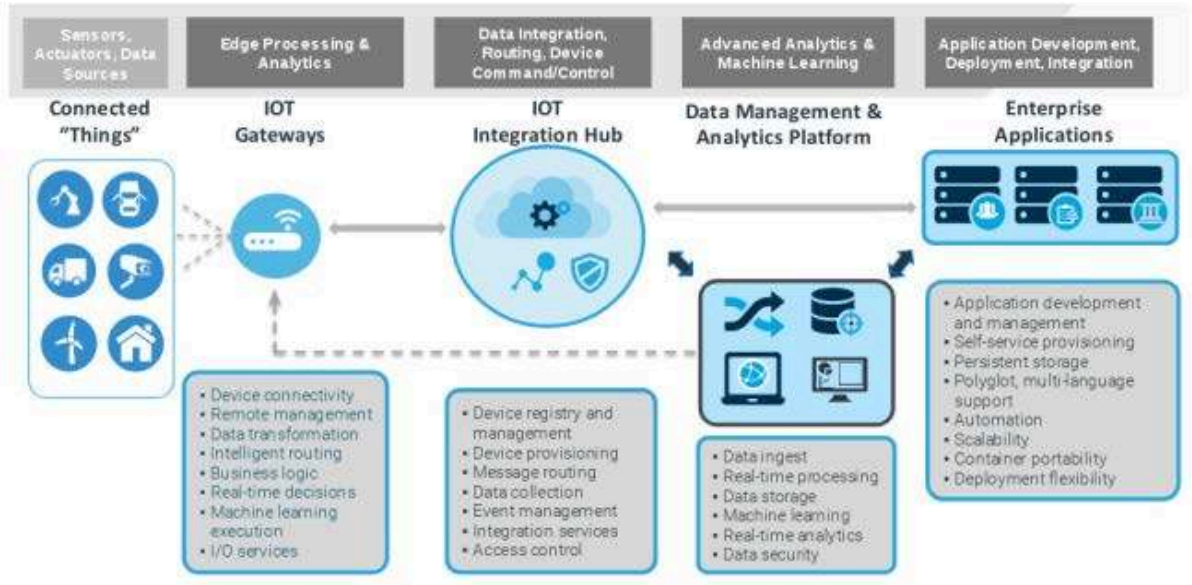

Figure 5 - E2E IoT Functional Architecture

Deploying IoT solutions in real production scenarios need more than a single provider to successfully accomplish the functionality in the above diagram.

Successful implementations have used a combination of technology providers, network providers, security specialists, developers, and system integrators, all operating as a cooperative, open, modular, and flexible team.

Many companies have addressed this complex End to End (E2E) architecture by launching IoT cloud platforms that typically focus on a centralized cloud IoT integration Hub. These platforms typically handle:

1. Device registration, management, security, and monitoring
2. Aggregation of data from multiple IoT devices
3. Integration with Big data storage and analytics
4. Sharing of data with other systems and applications
5. Handling and responding to device events

What is common among these IoT cloud platforms is that they offer the service on a low-cost, per-device subscription model, making it accessible even to hobbyists and the vendor community.

2.2. Edge compute

In the previous chapter on the technology cycles, we have described the evolution towards more decentralization as a mandatory step to make current IoT solutions more economically viable.

The underlying reasons for moving more processing to the Edge are better understood and accepted today.

The word edge in this context means literally its geographic distribution. Edge computing is computing that's done at or near the source of the data, instead of relying on the cloud at one of a dozen data centers to do all the work. It doesn't mean the cloud will disappear. It means the cloud is coming to you.

The main drivers to make current IoT more economically viable in this context are:

- Bandwidth preservation
- Latency improvements
- Security
- Privacy
- Autonomous operation
- Agile developer friendly IoT edge devices

2.2.1. Latency

One of the drivers for edge computing is the speed of light. If Computer A needs to have a response from Computer B, half a globe away, before it can do anything, the user of Computer A perceives this delay as latency. The brief moments after you click a link before your web browser starts to actually show anything is in large part due to the speed of light.

Moving compute closer to the origin of the data reduces the latency involved in the round trip to the cloud. Some of the evolving use-cases such as Augmented Reality (AR) and Internet of Things (IoT)

benefit from edge computing. End users of these applications enjoy immersive experiences delivered by the edge.

The goal of edge computing is to minimize the latency by bringing the cloud compute capabilities to the edge.

2.2.2. Bandwidth

IoT incorporates devices, sensors and data sources to send data from the field to the cloud to help businesses make real-time decisions. Edge computing uses connected and non-internet connected devices to process data near where it is collected. So, instead of sending all of the data to the cloud, edge computing can process data and send a smaller selected set, accelerating the delivery of data.

Edge requires less bandwidth by processing and analyzing data where it's gathered. A McKinsey article about the potential of IoT mentions an oilrig that has 30,000 sensors, but only one percent of the data are examined. Consider how much more efficiently a system could function if only the necessary data points were sent to the cloud, instead of all the information which needed analysis. In a network with millions of devices and data sources, this is an invaluable benefit. With the addition of edge computing, businesses can process data near the collection site and send selected data to the cloud when desired; reducing bandwidth and limiting security risks.

For data intensive applications, cost improvements can easily be calculated as expressed in the Edge compute financial model calculated in previous chapter.

2.2.3. Security

Securing these new edge devices is vital and should not be forgotten. You will need to enforce data encryption, both in transit and at rest, and protect communication with master clouds. Only by establishing security by design and embedding security mechanisms in all the components/layers involved will your edge workload be on the right track. IoT devices often collect sensitive information, and there's little agreement on how to protect this data. A study by HP found that 70 percent of IoT devices are vulnerable to attack.

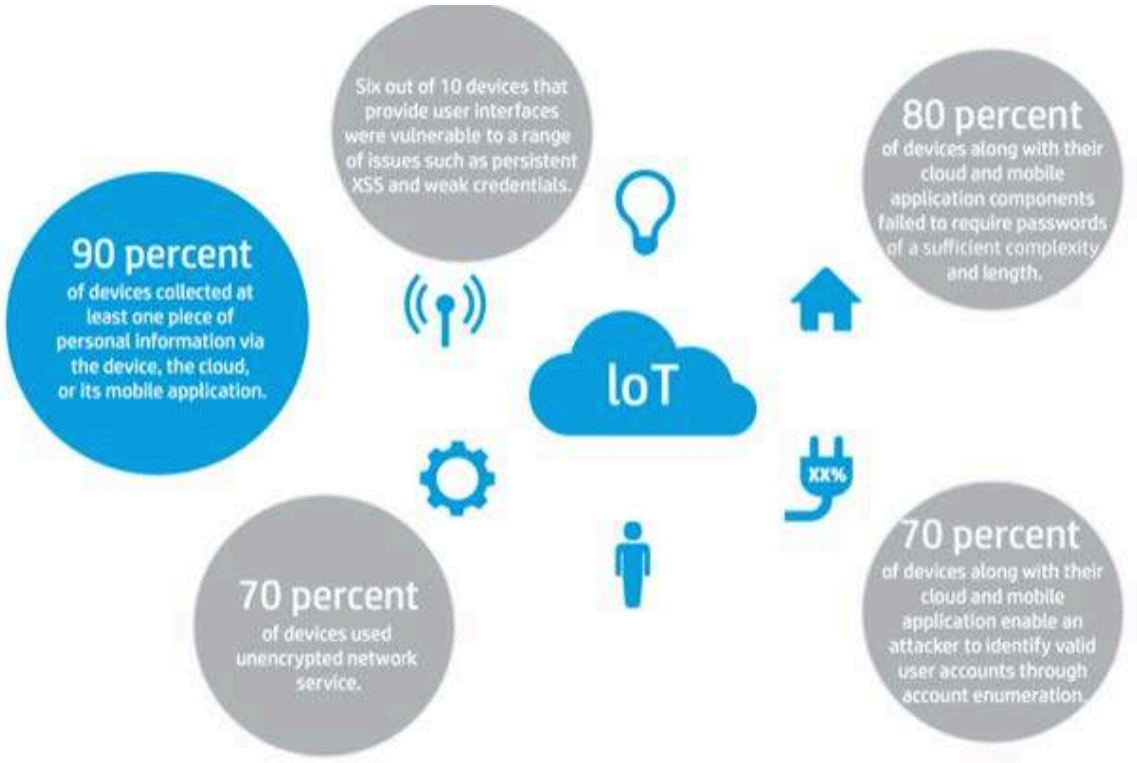

Figure 6 - Security vulnerabilities for IoT devices

2.2.4. Agile developer friendly IoT Edge devices

The benefit of the mobile app economy is that independent developers can create apps with a modest investment. And the ease of distribution and monetization provided by app stores creates a strong incentive for innovators. The IoT economy also needs independent developers, but the barriers to entry are higher due to the fragmented world of embedded developers and the complex nature of the development cycle, with its low level programming languages like C/C++, which require a steep learning curve before becoming proficient.

In these respect, the same kind of democratization which propelled the mobile and cloud world needs to happen on the embedded side as well, giving the developers a choice of selecting any modern programming language that makes them productive, and will open this traditional embedded world to a much larger community of developers.

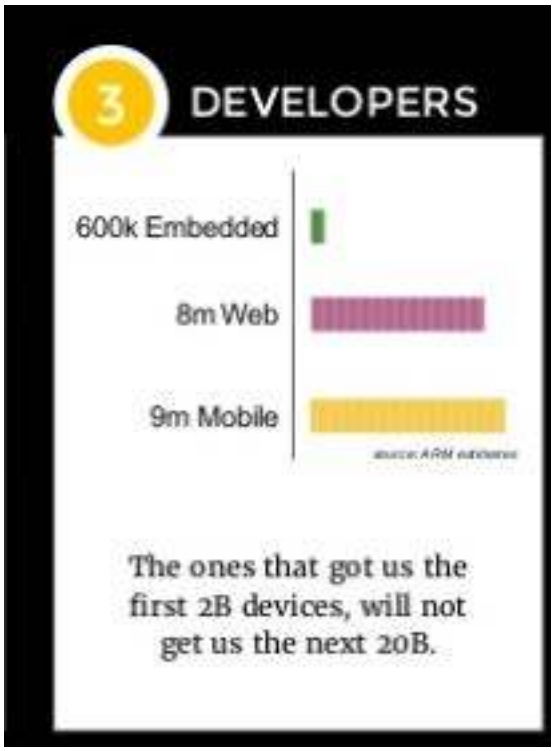

Figure 7 - Developer Communities

On top of this multi-programming language (polyglot) execution environment, the same frictionless experience needs to be given to the developer's with respect to the cloud's self-service model, friendly to use API's and easy to consume Software as a Service (SaaS) solutions.

2.2.5. Autonomous operation on the Edge devices

Edge compute can also maintain service operation during a network failure. On-device, service-specific processing that is enabled by edge compute can act as a buffer during a network failure and then synchronize data and state with cloud-based processes when Internet connectivity is restored. Some sensor applications have additional redundancy requirements that may include using a battery backup to operate during a power outage. Edge compute, coupled with a battery backup, creates a robust platform for offline data processing.

A cloud based IoT Hub or messaging service typically provides a very lightweight Agent that can be easily integrated in very constrained devices (20K – 50K SDRAM) and enables these devices to communicate with the IoT cloud platform directly.

When more functionality and processing is required a miniaturized version of the IoT core platform is offered that can act as an execution environment for local computation.

Vendors of this solution typically provide this as compatible solution with their cloud services so that from the sensor device point of view, there is no distinction between the local processing and the cloud processing.

The Edge IoT Gateway (GW) equipped with this IoT local environment then acts as a kind of bridge or proxy between the sensors and the cloud infrastructure.

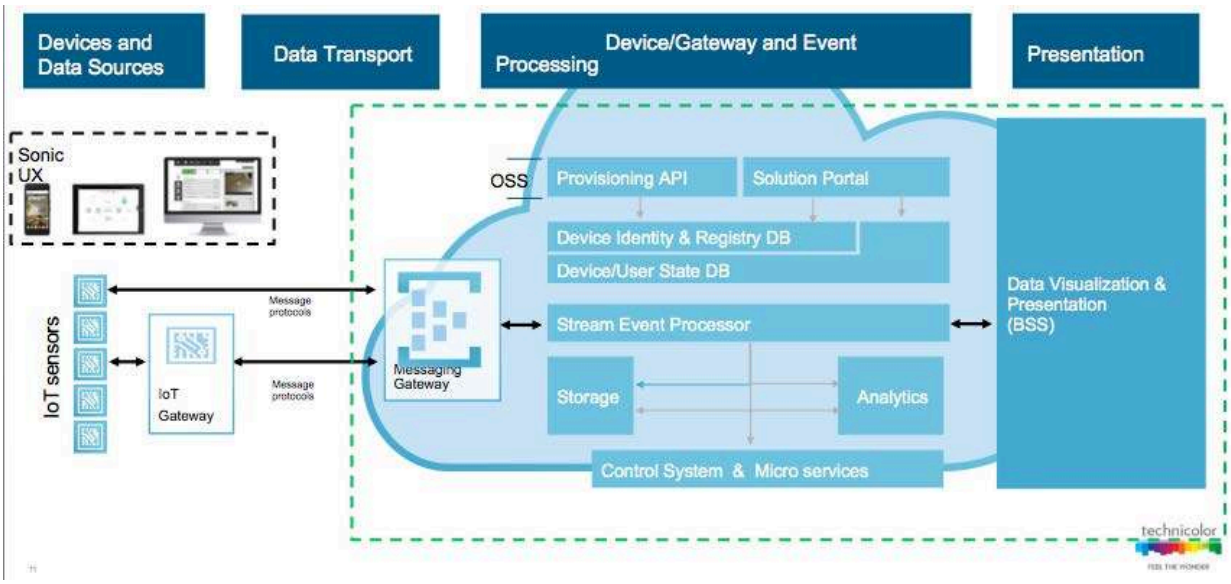

Figure 8 - IoT GW for autonomous operation

2.2.6. Privacy

Privacy has become a major concern for consumers in IoT markets like US and Europe, and is only one of the concerns reduced by successfully leveraging edge compute. Regionally, European markets have instituted greater regulatory protections over privacy than the US. This is best exemplified by the implementation of the General Data Protection Regulation (GDPR) which took effect May 2018. The GDPR significantly changes how companies handle EU citizen data privacy. GDPR places requirements for managing EU citizen data, such as a 72-hour notice to citizens after first having become aware of a data breach. Other aspects of the regulation require data erasure, data portability, and reporting. In general, personal data about identity, habits, speech and video will become a growing concern. Edge compute can enable new service architectures where personal data is processed locally to minimize the exposure of personal data.

Edge computing will undoubtedly provide the next level of efficiency for IoT solutions. For enterprises looking to build a powerful, scalable and secure IoT solution, it only makes sense to incorporate edge computing.

2.2.7. From cloud to Fog and Edge endpoints

There are actually two related concepts at play: edge computing and fog computing. Both models push data processing capabilities closer to where the data originates but differ in their emphasis. Crudely, fog computing locates the intelligence in the core operator's network while edge computing puts it inside the devices themselves.

Exactly how computing at the edge will finally be implemented will come out of a competition between various vendors, consortiums, and standards. The result will be a complex hybrid system that puts computing power and decision making at whatever location is optimal.

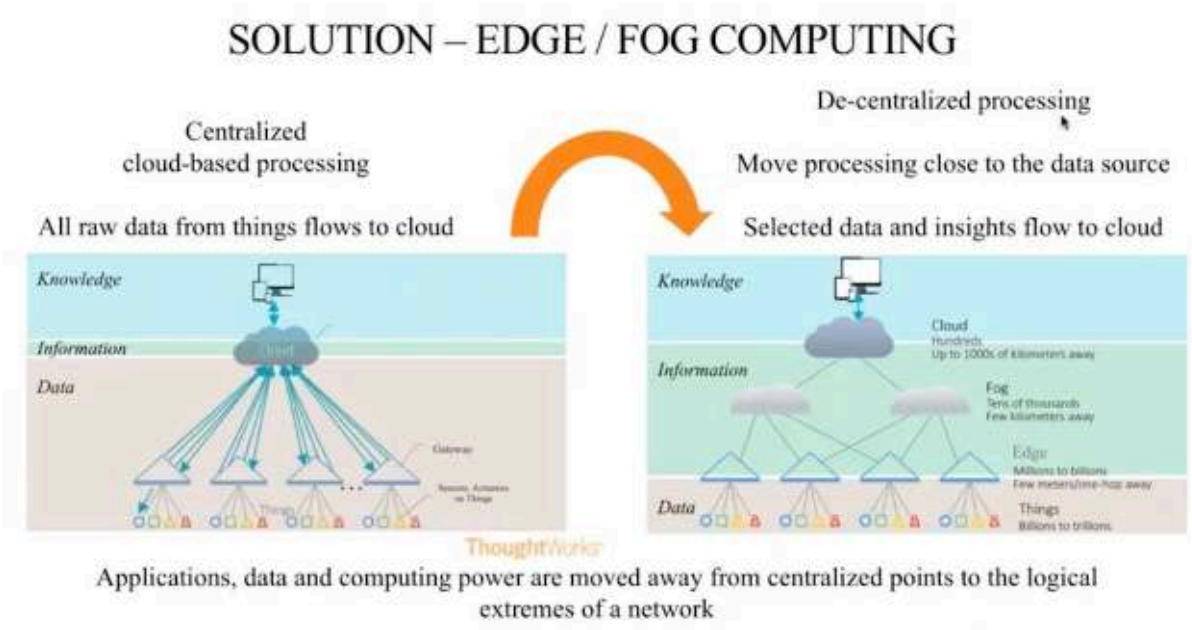

Figure 9 - Edge/Fog computing

2.3. Examples of IoT Edge Platforms

"Containers are a natural way to allow not only portability but also reusability so that you can assemble just the right application. Most modern Edge platforms are using containers under the hood, either by providing direct support for Docker containers, or by providing a serverless execution environment, compatible with their cloud offerings (AWS Lambda, Azure Functions) Edge apps or services are assembled from one or more containers and the containers perform inter-container communications using a local message broker in the Edge." The compatibility of containers provides deployment flexibility and allows efficient upgrade scenarios due to the capability of doing incremental container updates.

Typically, the Edge architecture contains similar functional blocks as their cloud counterparts but are miniaturized to be able to run on embedded devices.

This section focuses on exploring three current solutions in the market, 2 commercial ones from Amazon (AWS Greengrass) and Microsoft (Azure IoT Edge), and one open source implementation called Resin OS, which offers a stripped-down version of the Docker engine leveraging the open source project Balena.

These Edge platforms are currently mainly addressing the Industrial IoT space. The minimal hardware requirements typically start from a quad core central processing Unit (CPU) similar to a Raspberry Pi and memory requirements of around 1G Byte of SDRAM, and a couple of GBs for storage to be able to run a couple of containerized applications, and base images of a client Operating System (OS).

Both AWS and Azure solutions are tightly integrated with their cloud IoT core functionality, so that they can be centrally controlled for security reasons and administrated with respect to the Life cycle management of their containers and runtime engines. They naturally extend their cloud IoT platform towards the Edge.

Although the code is written in Lambda, Azure Functions or containers are technically capable of integrating with other cloud infrastructure or SaaS solutions. However, the risk of vendor lock-in is high due to the frictionless, tight integration of their own services.

Besides a container and/or server-less runtime execution environment both AWS and Azure offer local implementations of IoT messaging (IoT Hub) components which daisy chain to their cloud counterpart as a kind of federated broker infrastructure.

They also both provide a local copy of the device shadows/twins registry that can be optionally configured to synchronize their device/application state with the cloud IoT registry so that in case of connectivity problems cloud applications call still query the cloud registry, and the device can continue to operate on the local state of the device/application.

Both solutions are relatively young. AWS released their first version at Reinvent 2016 and Microsoft one year later, but they are actively adding new components like machine learning frameworks, and storage solutions to their offerings.

Amazon only offers a server-less compute environment, while Microsoft offers both a server and server-less environment as a Docker compatible container framework.

The advantage of the latter is that you can use existing software packed in Docker images, while the server-less environment needs newly designed software components in line with the server-less methodology.

None of them provides multi tenancy support to allow different vendors to control their own container infrastructure.

2.3.1. Azure IoT Edge

The Azure IoT Edge consists of:

- Docker compatible container runtime
- Azure Functions
- Message broker
- Offline/synchronized device twins
- Cloud management and deployment
- Cloud development/test support
- Azure stream analytics
- Azure machine learning

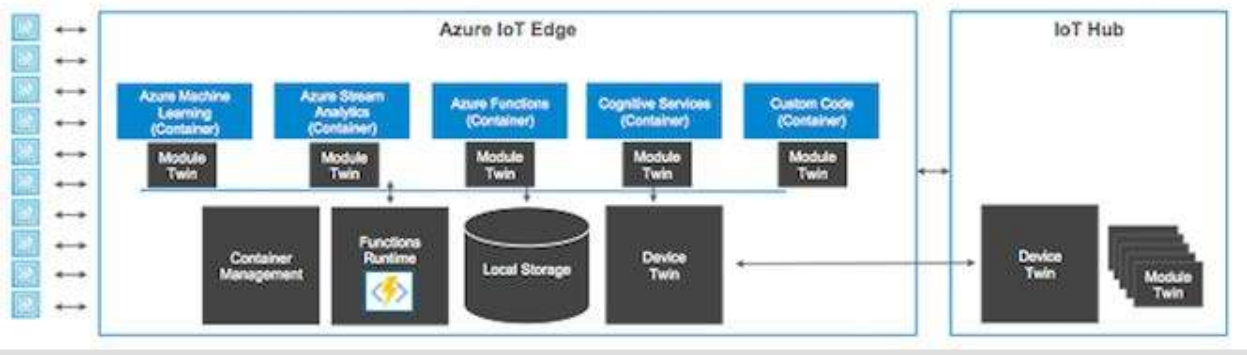

Figure 10 - Azure IoT Edge

2.3.2. AWS Greengrass

The AWS Greengrass Core software consists of:

- A message manager that routes messages between devices, Lambda functions, and AWS IoT.
- A Lambda runtime that runs user-defined Lambda functions.
- An implementation of the Device Shadow service that provides a local copy of shadows, which represent your devices. Shadows can be configured to sync with the cloud.
- A deployment agent that is notified of new or updated AWS Greengrass group configuration. When new or updated configuration is detected, the deployment agent downloads the configuration data and restarts the AWS Greengrass core.
- Analytics Engine
- Discovery
- Free RTOS Greengrass compatible version without container support for low footprint devices.

Figure 11 - AWS Greengrass

2.3.3. Resin OS

Resin OS shares a lot with cloud operating systems that support containers. They share the focus on minimalism, getting out of the user's way and letting their container do the heavy lifting, and using Docker, which is the standard way of running containers, and well understood by a large developer community. Resin OS applies the same principles to a different domain, that of embedded Linux devices, sometimes called "connected devices", "Internet of Things" or "Industrial Internet", depending on the use case.

At the core, it relies on a more optimized version of the Docker runtime engine, called Balena, which is an open source project, and is based on the Docker's Moby open source engine, but incorporates a number of optimizations to make the engine more usable for constraint embedded devices.

Some of these changes include:

- 3 –4 times smaller footprint by using one single binary and de-duplicating shared GoLang libraries.
- Multiple CPU architectures support, for a heterogeneous set of different embedded platforms.
- Bandwidth efficient delta updates with binary diffs, which result in dramatic size optimization for updating the container images.
- More robust support for flash updates during devices failure cases, like reboots or power failures, and minimized disk writes by on-the-fly extraction from images on the registry.
- Removing non-relevant features not applicable for IoT embedded devices like SWARM support, plugin support and removing cloud logging drivers.

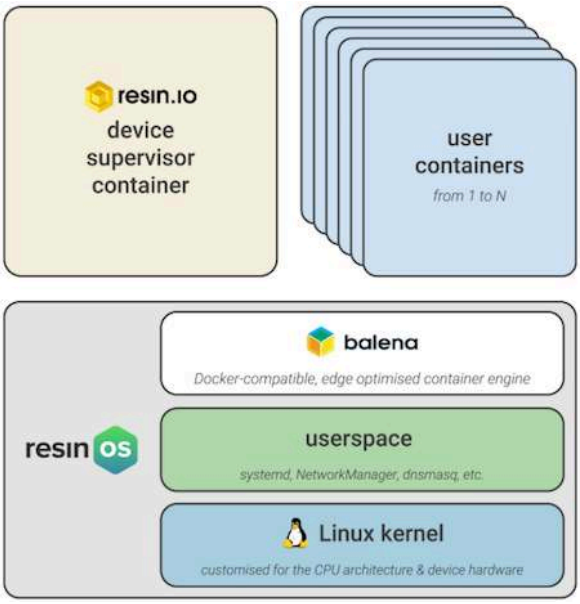

Figure 12 - Resin OS

2.4. Life Cycle Management of Iot Services, Development And Deployment

The Internet of Things means smart devices connecting every aspect of our lives, which creates a lot of challenges. And there's more to IoT than hardware and connectivity. To run and connect all these machines, we need software, lots of software. And we must be able to develop and update it quickly while making sure it's performing securely and efficiently.

So here's the problem: how do you deploy code to millions of devices in people's homes, offices, cars and beyond? How do you monitor these devices? What are the security implications? How do we design, develop and test all this stuff, and how do we configure all these devices? Fortunately, the tech industry has been addressing these challenges, albeit at a comparatively small scale, for over a decade.

2.4.1. DevOps Practices to the Rescue

The growth of DevOps is crucial here because it has led organizations to create the kind of automated systems that will be required for IoT software. Continuous delivery and continuous deployment at scale are only feasible through automation. Scripting or—better still—containerizing these processes with appropriate automated tests is the answer.

Mixing agile practices into DevOps accelerates things even further. Incremental builds with frequent releases of small batches of code makes for faster and safer development. Close customer contact to solicit feedback and usage analysis coupled with continuous improvement helps agile organizations respond quickly to emerging requirements.

2.4.2. Securing IoT Devices the Agile Way

We have barely touched upon the most significant of the concerns that organizations have about IoT: security. With all those connected devices out there in the wild, the attack surface is almost inconceivably huge. More than ever, the common development error of treating security as an afterthought just won't cut it.

Luckily, DevOps and agile practitioners don't think this way. Security is not silos—it's part of the development cycle, in the same way that product, quality and performance are. With this most significant of concerns addressed, it should be clear that organizations that use modern development practices are well set up to cope with the challenges of IoT.

The end result will be solutions that balance their functionality between the edge and cloud and creating a flexible approach to shift and lift these components back and forth when required.

Fortunately, most existing cloud IoT platforms include services around device management and SW Life cycle management as part of their core offering, allowing these solutions to easily scale to millions of devices. These foundations can be used to build modern, frictionless workflows and integrate them into existing developer's SaaS solutions, giving the developers the tools, they are used to and love like GitHub, a Container registry etc. After all, these ecosystems already play a crucial role in mobile and cloud development.

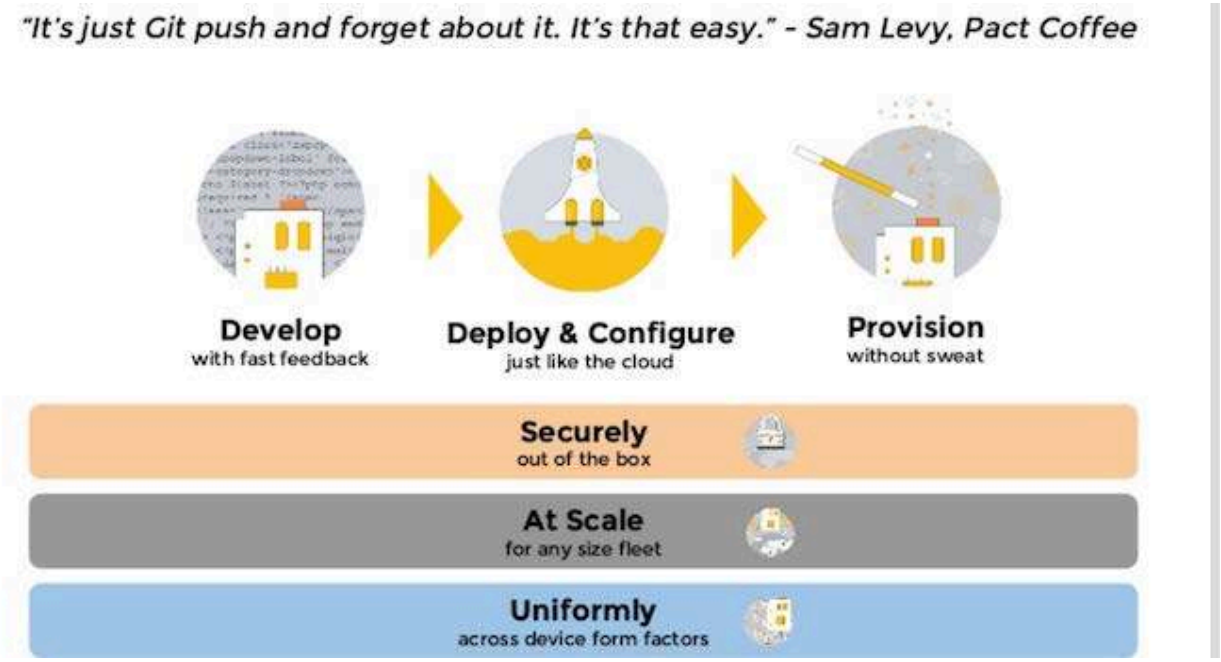

Figure 13 - Developer friendly workflows

Today, startups are already working on solutions that deliver this seamless, frictionless experience for developers working with IoT devices in the field.

They use containers as a means to develop and deploy their functionality and provide a workflow and tools, which are familiar for developers, and which were born in the cloud.

This brings IoT services development almost to the same level as their mobile counterparts, which have already established this mature practice for years with ecosystems that provide similar solutions and tools.

Today these solutions still have modest scale in terms of the amounts of devices they can handle, but thanks to the advances made on the basic core IoT foundation building blocks this type of approach will shortly come to the realm of the operators' large fleet deployments.

3. Lightweigth Virtualization – Containers

At the turn of the century when cloud computing became popular, public cloud infrastructure providers used Virtual Machines (a software abstraction of a physical computer/server) as a virtualization technique to create abstraction of the physical hardware, create large aggregated pools of logical resources consisting of CPUs, memory, disks, file storage, applications, networking, and offer those resources to users or customers in the form of agile, scalable, consolidated virtual machines. Virtual Machine, while providing complete isolation of resources from one user to another, did come at a price. The price paid was in terms of overhead related to CPU virtualization, virtualizing the disk (block) and network I/O. For example, a typical state of art physical server could only host a few dozen VMs at most.

While a VM-based virtualization is a mandatory requirement in some business models, many business models exist where a lightweight virtualization mechanism might suffice, as long as it provided adequate application isolation.

In the past few years Containers have emerged as such a key application isolation, packaging and delivery technology, combining lightweight application isolation with the flexibility of image-based deployment methods. Containers use core technologies such as Control Groups (Cgroups) for Resource Management, Namespaces for Process Isolation, thus enabling secure multi-tenancy for applications without the overhead experienced by Virtual Machines.

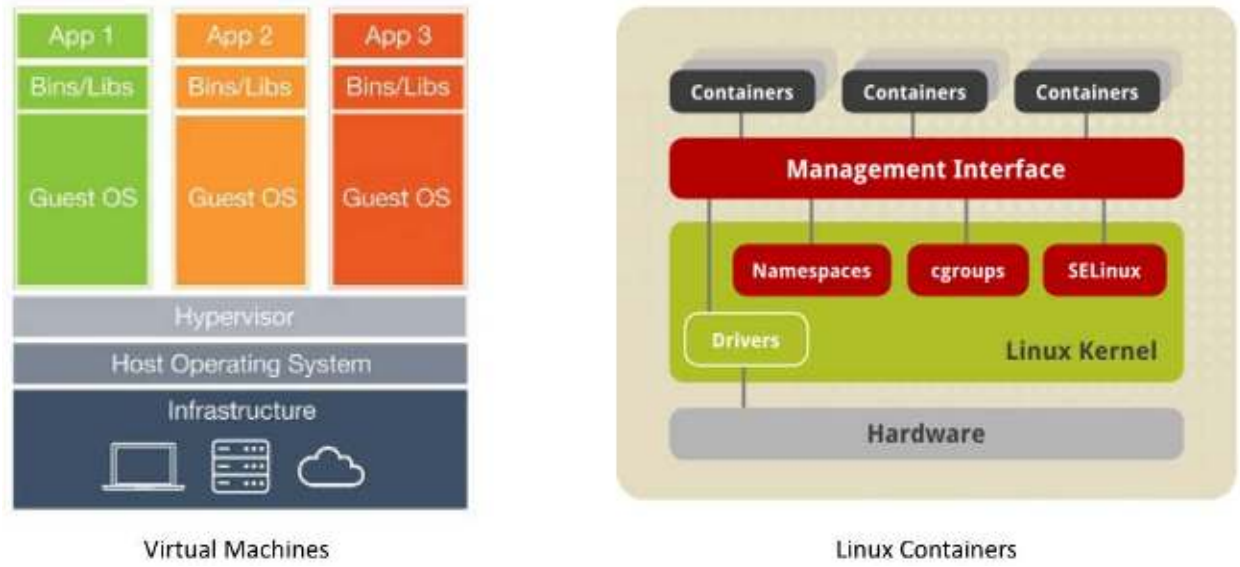

Figure 14 - Virtual Machines Versus Containers

Containers have since further evolved into various kinds that have varying degrees of abstraction based on application runtime requirements.

For Edge Compute Nodes, Virtual Machines are generally not viable due to the limited compute capabilities in the Edge device, hence a lot of virtualization for Edge applications is focused on Container based technologies.

3.1. Container Technologies

Compared to hypervisors, container-based virtualization provides different abstraction levels regarding virtualization and isolation. Hypervisors virtualize hardware and device drivers, generating a higher overhead. In contrast, containers avoid such overhead by implementing process isolation at the operating system level. A single container instance combines the application with all its dependencies, and runs as an isolated process in *user space* on the host operating system (OS), while the OS kernel is shared among all the containers as shown in the figure below. The lack of needing hardware/driver virtualization, together with the *shared kernel* feature, provide the ability to achieve a higher virtualized instance density because the resulting disk images are smaller.

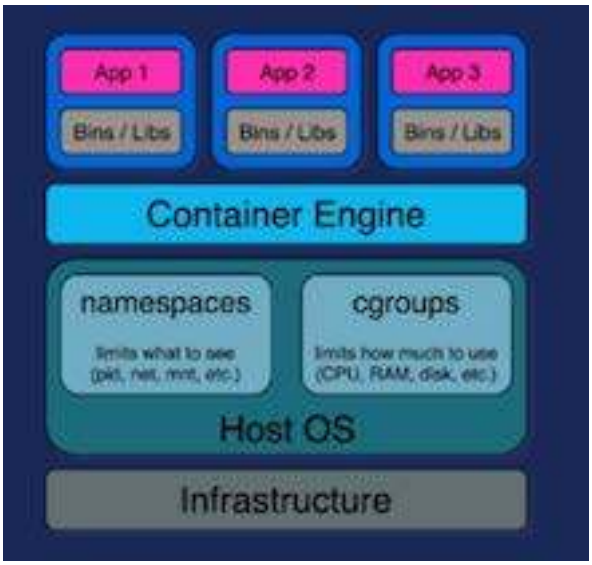

Figure 15 Container Building Blocks

In the past few years several types of containers have come to prominence, each type most optimized for a particular application class and business logic. Containers themselves are not a new technology. Solaris platform has offered the concept of Solaris Zones for many years, and many Linux administrators have experimented with FreeBSD jails as a lightweight alternative to virtual machines. In order to understand the current container landscape, this chapter attempts to categorize the most fundamental approaches and technologies used.

At the core, Docker, LXC and other container technologies depend on the key Linux kernel features of Cgroups and Namespaces. Depending on use cases or requirements these container engines can make different trade-offs with respect to architectural choices and implementation. In order to understand these choices and validate the impact they have on performance, security, portability and footprint in the context of constrained device types, the categorization is depicted in the figure below:

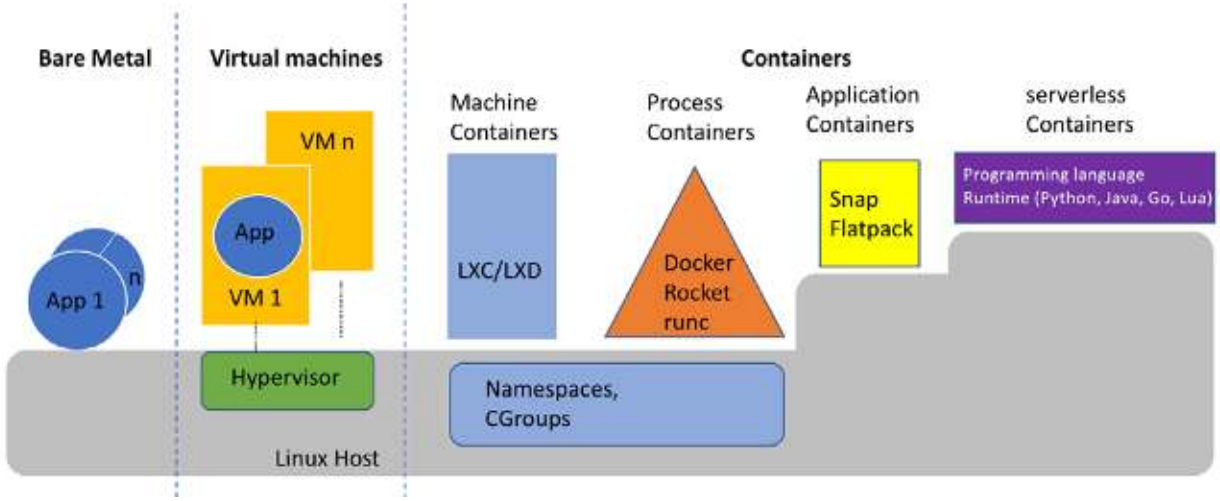

Figure 16 - Container Categories

3.1.1. Machine/System Containers (LXC/LXD)

Linux Containers (LXC) main focus is system containers. That is, containers which offer an environment as close as possible to the one you'd get from a VM but without the overhead that comes with running a separate kernel and simulating all the hardware. This is achieved through a combination of kernel security features such as Namespaces, mandatory access control and control groups. Just like Solaris Zones and BSD Jails, it tries to provide a lightweight VM experience for system administrators.

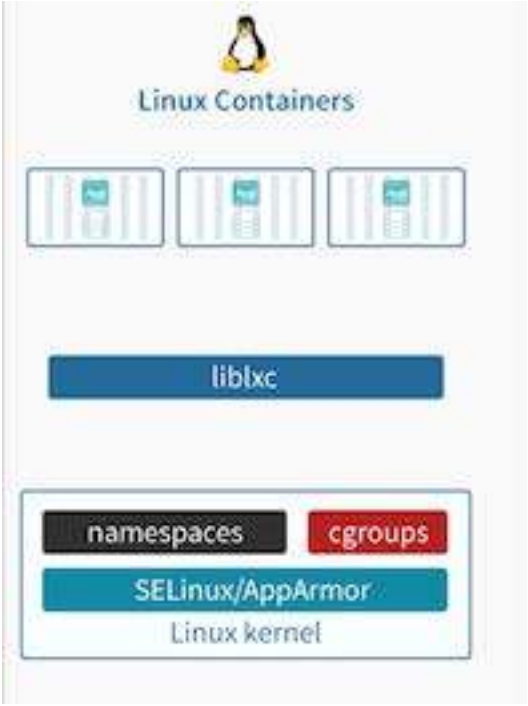

Figure 17 - LXC Containers

LXC storage management is rather simple. It supports a variety of storage back ends like btrfs, lvm, overlayfs, and zfs. However, by default (if no storage backend is defined), LXC simply stores the root file system under /var/lib/lxc/[container-name]/rootfs. For databases and other data-heavy applications, you can load data on the Rootfs directly or mount separate external shared storage volumes for both the data and Rootfs. Creating an image out of an LXC container just requires tar'ing up the Rootfs directory.

LXC is focused on IT Operators with the goal of providing a lightweight virtualization solution. This means, for a system administrator to transition from hypervisor-based virtualization to LXC is rather painless. Everything from building container templates, to deploying containers, to configuring the OS, networking, mounting storage, deploying applications, etc. all remain the same. In fact, LXC gives you direct SSH access. This means all the scripts and automation workflows written for VMs and physical servers, apply to LXC containers too. LXC also supports a template notion, which essentially is a shell script that installs required packages and creates required configuration files.

One can create either privileged or unprivileged LXC containers. The default today is unprivileged which allows for configurable access to the Linux kernel system functions.

LXD is an extension of LXC. While LXC is used under the hood, and provides a user space CLI interface, LXD is a system daemon that extends the LXC library with REST API's to provide container

management as a service and is written in GoLang. It simplifies the management of multiple LXC containers, takes advantage of host-level security measures (which in the case of LXC is more problematic) and simplifies resource sharing like networking and data storage.

In contrast to Docker it is only available for Linux platforms, which is not a fundamental problem for our type of Linux based platforms, but limits the development experience a bit.

3.1.2. Process Containers

While the term *process containers* was initially coined by Google in 2006 and led to the creation of Cgroups, this type of containerization focuses on limiting, accounting and isolating resource usage for a limited collection of processes. The new generation of container runtimes like Docker and Rocket (RKT) take this concept further by restricting their containers to only carry one service. A typical consequence of that is that even secure shell (ssh) access is not supported inside these containers, however this delivers a better separation of concerns.

This is very different from the traditional system containers like the ones created by LXC where you can have multiple services running on the same OS.

3.1.2.1. Docker

Containers became truly mainstream with the Docker project. Like most great technical innovations, Docker stood on the shoulders of older giants but with a new focus that addressed a market need at exactly the right time. In Docker's case it was a focus on developer experience and ease of (re)use that took containers from a fairly arcane infrastructure technology into something truly transformational.

While Docker was originally designed on top of LXC, it created it own version of that with the introduction of Runc. Runc resembled the basic functionality as LXC, with the addition that it is standards compliant with an OCI format. Docker is a first-class citizen in the design of Runc.

Figure 18 depicts the current building blocks of Docker.

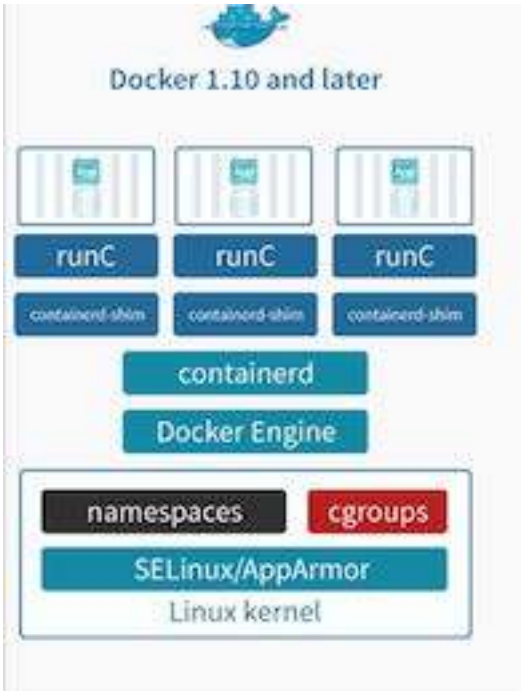

Figure 18 - Docker Building Blocks

The biggest innovation that Docker added compared to the previous mentioned system containers is that it added the container management images and made them stackable to achieve portability and flexibility with respect to reuse of these images for the developer.

Each Docker image references a list of read-only layers that represent file system differences. Layers are stacked on top of each other to form a base for a container's root file system. The Docker storage driver is responsible for stacking these layers and providing a single unified view.

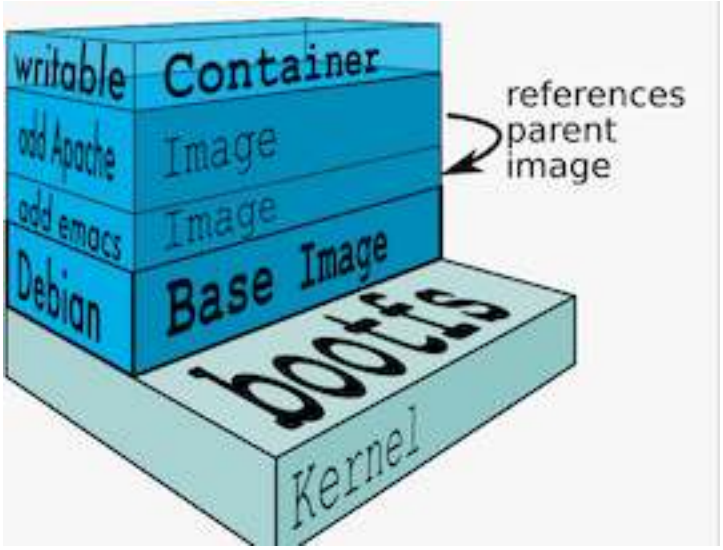

Figure 19 - Docker packaging and image format

The major difference between a container and an image is the top writable layer. All writes to the container that add new or modify existing data are stored in this writable layer. When the container is deleted the writable layer is also deleted. The underlying image remains unchanged. Because each container has its own thin writable container layer, and all changes are stored in this container layer, this means that multiple containers can share access to the same underlying image and yet have their own data state.

Layering helps Docker to reduce duplication and increases re-use. This is very helpful when one wants to create different containers for various components. One can start with a base image that is common for all the components and then just add layers that are specific to your component. Layering also helps when one wants to rollback changes as you can simply switch to the old layers, and there is almost no overhead involved in doing so.

In order to achieve this a lightweight Union File System is used, that allows sharing files that are not changed, reducing the storage and memory footprint and enhancing the startup time.

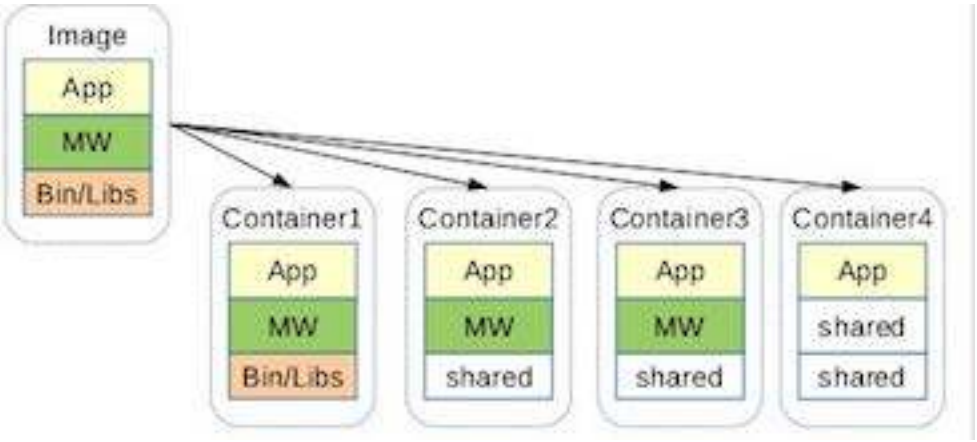

Figure 20 - Docker Containers and images

Copy-on-write is a similar strategy of sharing and copying, in which the system processes that need access to the same data share the same instance of that data rather than having their own copy. At some point, if any one process wants to modify or write to the data, only then does the operating system make a copy of the data for that process to use. Docker makes use of copy-on-write technology with both images and containers.

Currently the Docker engine is stripped out in different components, where the current Docker engine focuses on image management, container orchestration and the interaction with the Docker container repository, and Containerd is managing the complete container lifecycle of its host system: image transfer and storage, container execution and supervision, low-level storage and network attachments, etc, similar to what LXD does for LXC.

Docker is more than an image format and a daemon, though. The complete Docker architecture comprises the following components:

- Docker daemon: runs on a host and is the combination of Containerd and the Docker engine.
- Client: connects to the daemon, and is the primary user interface
- Images: read-only template used to create containers
- Containers: runnable instance of a Docker image
- Registry: private or public registry of Docker images

- Services: a scheduling service called Swarm which enables multi-host, multi-container deployment. Swarm was introduced in version 1.12

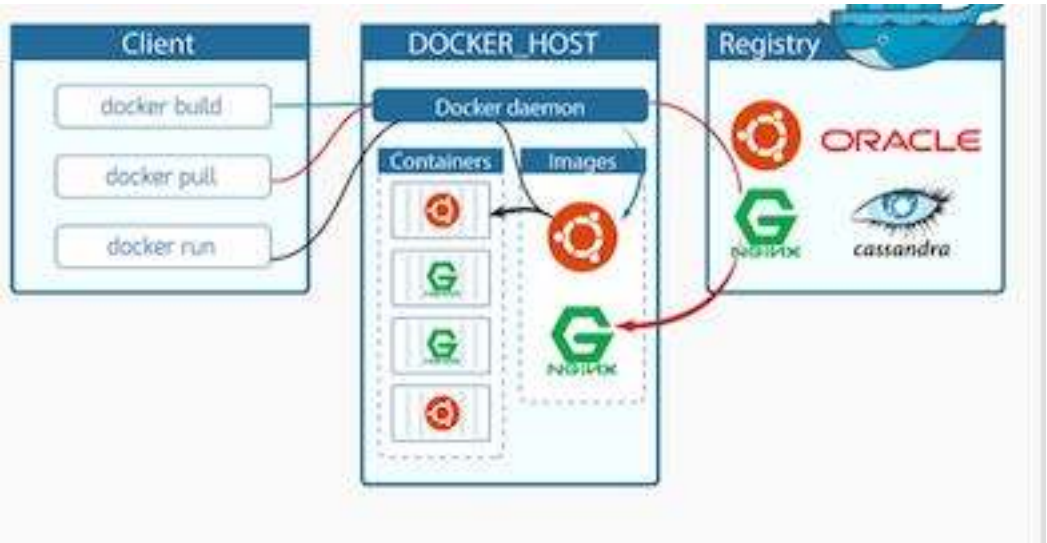

Figure 21 - Docker ecosystem

3.1.2.2. Microservices

The application space where technologies like Docker and RKT are a perfect fit for, can be roughly categorized as – modern, Microservices-based, and traditional enterprise applications.

MSOs can adopt a Microservices architecture which has gained popularity amongst new web-scale companies like Netflix, Google, Twitter, etc. Applications with a Microservice architecture consist of a set of narrowly focused, independently deployable services. This would give the MSOs the advantage of increased agility and resilience to roll out new value added services to their subscribers. Agility since individual services can be updated and redeployed in isolation. Given the distributed nature of Microservices, they can be deployed across different platforms and infrastructures, and the developers are forced to think about resilience from the ground up instead of as an afterthought.

To support ephemeral and elasticity workloads a lot of focus and functionality has been added to make these solutions a perfect match for container orchestration frameworks like Kubernetes, DCOS, SWARM and the like, but as a side effect of some of their choices, their runtime footprint is heavy, and the impact on performance by adding a specific storage backend and copy-on-write functionality can be a bottleneck for IO intensive applications.

3.1.3. Application/Desktop containers

Machine containers like Docker or RKT are almost as versatile and adept to orchestrate a complex system of Microservices; however machine containers are not ideal for a significant number of Edge-Compute use cases like IoT where the physical H/W is an embedded platform with very limited compute, memory and storage like a Cable Modem/Gateway or a Wi-Fi Router. Such platforms typically have dual core ARM processors (Quad core at best) and have just enough resources to run lightweight processes in an isolated sandbox.

An isolated sandbox is a mechanism to separate running programs with the aspect of keeping the operating system and other applications isolated and safe when running programs. The sandbox does so by isolating or virtualizing an environment where the program can be executed in, limiting the possible access to the underlying operating system and other applications. Monitoring in a sandbox can occur as well as trying to check whether or not the executing program is malicious.

These techniques have been heavily used in the past by web browsers and other graphical applications, to allow running untrusted code in these environments, without compromising the security and robustness of the overall system.

Over the recent years, we have seen these techniques being reused in the Linux Desktop Application space to overcome the fragmentation that exists with respect to the numerous flavors of Linux distributions out there. The idea here is to have a minimum base OS and run everything in distribution-agnostic containers.

Due to the nature of this use case, which does not require running hundred of containers, there is also less need to optimize for size and reuse of different image layers and using techniques as copy-on-write. It allows keeping the storage model simple, and avoiding the negative impact on performance that these union file systems tend to have.

Most notable players in this field are AppImage, Ubuntu Snap and Flatpack.

While originally developed for Desktop applications, Snap for example is really trying to broaden their applicability to other domains like IoT edge computing, and recently also added OpenWRT support. The disadvantage of the current Snap approach is that it is very vendor specific, and the current base image only supports a heavy 250MB Ubuntu distribution.

On the flip side, these tools allow secure and controlled access to a shared system bus like Databus (DBUS) or eventually Microbus (UBUS) for OpenWRT, which allows interaction between different containers in an efficient way.

3.1.4. Serverless containers

Serverless computing is a cloud-computing execution model in which the cloud provider dynamically manages the allocation of machine resources. The name "serverless computing" is used because the server management and capacity planning decisions are completely hidden from the developer or operator.

AWS Lambda introduced by Amazon in 2014,[5] was the first public cloud vendor with an abstract serverless computing offering, soon followed by Azure and Google, with their variant Cloud Function. Today, both Amazon and Azure also offer these runtimes for their Edge computing offering, notably AWS Greengrass and Azure IoT Edge.

Although it is out of the scope of this paper to discuss the specific programming model for these new types of solutions, they are using the same kind of containerization and isolation techniques as their former counterparts. The way they achieve isolation is by using a programming language runtime that acts as a barrier between the host and untrusted code inside the containers.

As such they provide a higher level of abstraction then just the operating system by making the language runtime accessible from within the containers, and leaving package management up to specific language

specific solutions or simple tar or zip approaches. The resulting implementation as such tends to be less complicated in terms of functionality and footprint.

As of today these solutions offer runtimes for different languages like Python, Java, C# and Go.

Serverless is sometimes mistakenly considered as more secure than traditional architectures. While this is true to some extent because OS vulnerabilities are taken care of by the underlying environment, the total attack surface is significantly larger. Serverless approach is not mutually exclusive of the other types of containerization techniques but can be used to complement each other.

Today containers technologies are supporting the major range of Linux distributions, even Android, and serverless computing frameworks can run on top of Dockerized environments or application container frameworks like Snap. As an example Greengrass today is available both for Docker and for Snap, and Azure IoT is built on the foundation of Docker infrastructure.

3.2. Container standarization

The **Open Container Initiative** (**OCI**) is a Linux Foundation project to design open standards for operating-system-level virtualization, most importantly Linux Containers. There are currently two specifications in development and in use: Runtime Specification (runtime-spec) and the Image Specification (image-spec), and the consortium recently also released a Distribution Specification for Registries heavily based on the Docker protocol.

The OCI was launched on June 22nd 2015 by Docker, CoreOS and other leaders in the container industry, and is today supported by the major public cloud players Amazon, Microsoft, Google, IBM, chip manufacturers like Intel, and some notable Telco players like AT&T and Huawei.

Their goal is to address the fragmentation in the container landscape by defining clear specifications to align on interoperability and compatibility and avoiding vendor lock-in to accelerate container adoption.

Besides specifications they also released an open source initiative in collaboration with Docker to provide a basic runtime called Runc to drive innovation and allow building higher-level tools on top of it. It is used today under the hood for all Docker types of frameworks and on GreenGrass and Azure Edge.

Also all major cloud providers are supporting the OCI image specifications currently in their container registries, and will soon also support the upcoming API, which is defined in the Distribution specification for registries. The distribution-spec is the latest in a series of OCI initiatives highlighting the rapid rate at which container technologies are maturing, In fact, rather than focusing mainly on building pipelines to construct single applications, more attention is being focused on how to manage supply chains that ultimately will consist of multiple registries and pipelines. Constructing supply chains at scale assumes a base level of interoperability between registries enabled by a Docker Registry v2 protocol, which can now be deployed anywhere to access images in public or private clouds.

App Container (appc) is an open specification that defines several aspects of how to run applications in containers: an image format, runtime environment, and discovery protocol.

The image format defined by appc and used in RKT is the Application Container Image or ACI.

As part of the success of OCI, alternative technologies like LXC and RKT are starting to offer more support and integration of OCI specifications, by adapting their roadmap, however this evolution is still ongoing.

4. MSO CPE landscape & CPE device characteristics

For the last 20 years or so fixed broadband connectivity has become a de-facto choice of Internet connectivity for most users. MSO-led cable broadband has led the way of providing broadband Internet connectivity to the users. During the early years of broadband era, the desktop/PC used to be the primary end user device at home. MSOs used to provide a DOCSIS (Data Over Cable Service Interface Specification) modem that served as the NTU (Network Termination Unit) at the home premise. PCs in the home typically connected to the modem via Ethernet cables. Once MSOs started offering home voice/telephony service, MSOs combined the modem and voice telephony gateway into one integrated broadband device called an eMTA. eMTAs are typically a purpose-built network device with very little compute capabilities outside of their core function of providing voice and internet termination function.

In the last 10 years, however, the connectivity landscape inside a typical residential home or enterprise has significantly changed compared to early days of broadband described above. Today, wireless (Wi-Fi) is the primary way users access Internet in their homes. Also, the number of end user devices have proliferated from a few desktops to Wi-Fi capable smart phones, Tablets, laptops etc. Further, users are also bringing home internet connect smart devices like smart TVs, Coffee Pots, garage door sensors, door locks, home assistant / speakers etc. These smart things are typically Wi-Fi, Bluetooth or Zigbee connected and an average US household has many as 13 such connected devices at home [2] and the number is expected to go up to 50 by 2025. Proliferation of IOT end user devices and extensive Wi-Fi usage in the home, gives MSOs a chance to move upstream and provide a new category of value added services to their subscribers. MSOs are thus integrating many wireless connectivity technologies in the current generation DOCSIS gateways e.g. Wi-Fi, Bluetooth, Zigbee/ZWave, LoRA etc. to provide a wide spectrum of wireless connectivity options in the home. Further Moore's law has been bringing the cost of computing down to a fraction of what it used to be 20 years ago. Multi-core, general-purpose computing costs for embedded systems of the likes of Broadband Gateways are now common. (see Raspberry Pi project).

In line with the cost/performance curve, the next generation of the broadband GWs will integrate a modem, a router, best in class Wi-Fi, IoT radios (802.15), Gigabit Ethernet interfaces, multi core GPU (e.g. Quad core ARM), 512+ MB of RAM, 512 MB+ of Flash etc.; all for a familiar price point that is amenable to MSOs. All such integrated Gateways are capable of performing computing at the edge / customer premise.

Edge Compute capable broadband gateways provide many tangible benefits to the MSOs. Edge Compute gateways are actually platforms that let the MSO disaggregate the applications from underlying gateway firmware. In the traditional workflow, the MSO have to do intense planning for 5-7 years to do a CPE refresh. Since the apps and the firmware were tightly integrated, once the CPEs were deployed, there was little to no new feature development on these gateways. This made the service rollout cycles for MSOs very slow and risk of failure very high, as MSOs usually had to plan 5-7 years in advance. On the other hand, Edge-Compute Gateways are capable of disaggregating the 'Gateway OS' from the network applications (e.g. IoT apps etc).Just like in Desktop computing where a desktop OS is disaggregated from the desktop applications, Edge-Compute capable broadband Gateways let MSOs deploy the Gateways and the applications independently. Typically, Gateway OS will evolve at a slow cadence while applications will be rolled out in a much more agile fashion. This helps the MSO operate at the Web scale, giving them much needed agility and service velocity, which is needed to compete in the marketplace.

5. The New Embedded Stack

The telecom industry is currently going through a radical shift towards softwarization of the current hardware base networking functionality. Originally, they had started mapping their networking functions on a VM based infrastructure. In the mobile world with the introduction of 5G, they more and more started to apply a cloud native approach towards the implementation of network function virtualization (NFV) functions, with more prominent usage of container technology and orchestration. ETSI has been standardizing the Multiaccess Edge Computing environment also called MEC (Mobile Edge Computing). AT&T recently merged their efforts around the Open Network Automation Platform (ONAP) with the China Mobile Open Initiative, by contributing their code to the Linux Foundation as an open source initiative, which broadens the application domain towards wireline and enterprise.

From an operator perspective, the Edge means the base station and their core network infrastructure, but extending their reach towards the physical edge. Applying these lightweight virtualization techniques on their customer premises equipment could give them end to end control and flexibility around their efforts on network virtualization, and allow third parties to innovate with new IoT use cases.

Operators' close proximity to their users due to their large installed base of CPE's could make them more competitive by exploiting edge computing on these types of devices.

Specifically, in the IoT applications domain, the physical edge will play a dominant role in creating value-add services, especially for those novel use cases that are at the intersection of different IoT domains and different verticals, where latency and autonomous operations are crucial. Edge compute will be a key enabler that will foster and promote new compelling use cases. Figure 22 shows a subset of these use cases.

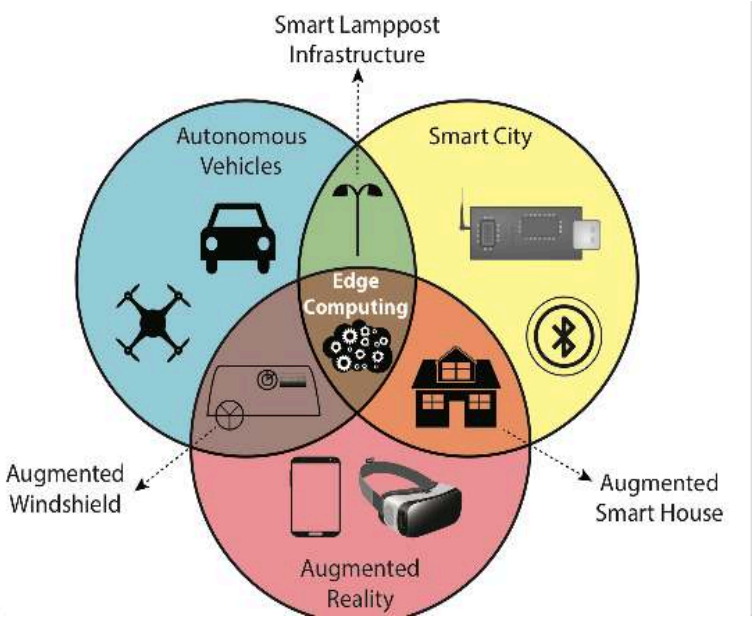

Figure 22 - Edge compute use-case domains

By extending the edge, CPE devices can play a crucial role in bridging the current fragmented and heterogeneous landscape of IoT cloud platforms, that are using different messaging/communication protocols and API's, with their own vendor specific semantics, the limitless amount of low power

connectivity protocols and standards that are currently dominating most of the smart home use cases of today, and allowing third parties to fully exploit the hardware capabilities present in those devices.

To fully exploit this evolution, the key elements that need to be addressed are creating the proper abstractions, the programmability of these devices and the interoperability between various ecosystem solutions. Lightweight virtualization technologies like containerization appear to be a perfect fit to address these challenges, since they provide a good computational abstracted environment and allow independence for various technology choices to be used inside this isolated environment, as well as independence and full end-to-end ownership with respect to the integration of these components with their own cloud infrastructure and services.

Given the fragmented IoT landscape and importance of cross domain use cases that benefit from a CPE edge solution, this edge platform should support a multi-vendor environment, and hence this platform needs multi-tenancy support as a first class citizen to host different solutions, that is controlled and operationally managed by the operator, so that issues around security and privacy can be fully covered.

To improve integration and create more flexibility in the choice of where to place these containerized application or service components, either on the CPE device itself or close by in the network, the CPE edge solution should have a common execution environment that enables cross platform deployment. We see this evolution today in commercial approaches from the public cloud providers like Amazon and Microsoft that make their edge environments compatible with their cloud counterparts so that containers or serverless functions can run practically anywhere.

Taking into account all previously mentioned arguments, we recommend taking an evolutionary approach by segregating the current monolithic CPE firmware into a dual track approach, where a clean separation of concern is obtained between traditional development and deployment strategies of the firmware versus a more open and dynamic environment for containerized components. As such the years of investment of this network-centric firmware and integration with the operators' OSS/BSS (Operations and business Support Systems) and remote management infrastructure can be safeguarded and can be used to upgrade both the firmware as the containerized components.

As an example of that, the current TR69 standard has already been enhanced with support for LCM of software components, more notably by the TR-157 data model, which can support different execution environments.

In addition to being backwards compatible with an MSO's infrastructure, we propose an additional middleware layer on top of the firmware that delivers a modern developers focused life cycle management layer for containerized components, albeit still under the control of the operator, but that gives full control of the development and deployment of these components in an end-to-end fashion to the IoT application solution provider.

By exposing an LCM API from the firmware this middleware layer would be a combination of a containerized agent of the CPE device and a backend infrastructure focused on deployment and management of containerized services. Typically, this backend solution will leverage a cloud based IoT platform that contains the service provisioning methods and monitoring tools that are independent of the managed applications that run in parallel on the CPE edge platform.

As such, this means that this middle layer needs to be in line with the IoT strategy of a particular operator, since an IoT cloud platform typically supports all the foundational building blocks to be able to communicate with the devices, the service/asset management services, device on boarding, authentication and integration with data monitoring and analytics infrastructure., and is needed to orchestrate the

deployment of containers, and to be able to do the operational management of the running containerized services.

By using a cloud-based control plane that interfaces with the existing CPE middleware functionality, and integrated with their existing back office infrastructure, operators can offer a multi service CPE platform that can deliver new, compelling use cases in line with their goals on enterprise-grade service assurance.

As previous stated, there is a heterogeneous set of IoT cloud solutions today that is probably not going away soon, and operators have or are in the process of making different strategic choices around technology and partners. Some of them are well versed and mature to build these IoT-based services in house, even in a hybrid fashion, while others will rely on commercial solutions and partners or have acquired an IoT platform.

The key here is to support easy integration on the CPE which will allow different operators to integrate their IoT backend with a set of standard local LCM API's and a virtualized IoT agent to deliver this type of orchestration around the LCM of containerized components in an end to end fashion.

This would allow operators to create an ecosystem around service components, much like the app stores of today, by exposing a set of high level services API from this middle layer for higher layer integration of additional services, similar in concept to the Northbound API of their current remote management infrastructure, but more open to third party solution providers.

Security and privacy controls, will be absolutely key, and included by design in this edge layer. As such there is a need for a secure execution environment, assisted by hardware mechanisms like Trusted Platform Modules (TPM) and a root of trust for safely storing secrets as supported by most modern CPU's like ARM Trust zone architecture. This will allow only trusted authenticated container code to run on these CPE devices, preventing hackers from compromising the systems, or running unintended malicious code. Set top box (STB) content providers today already mandate this to Digital Rights Management (DRM) violations.

This same proper isolation is needed for privacy related issues, hence the necessary mechanisms on access controls need to be in place to protect unauthorized access, and limit exposure of users' personal information.

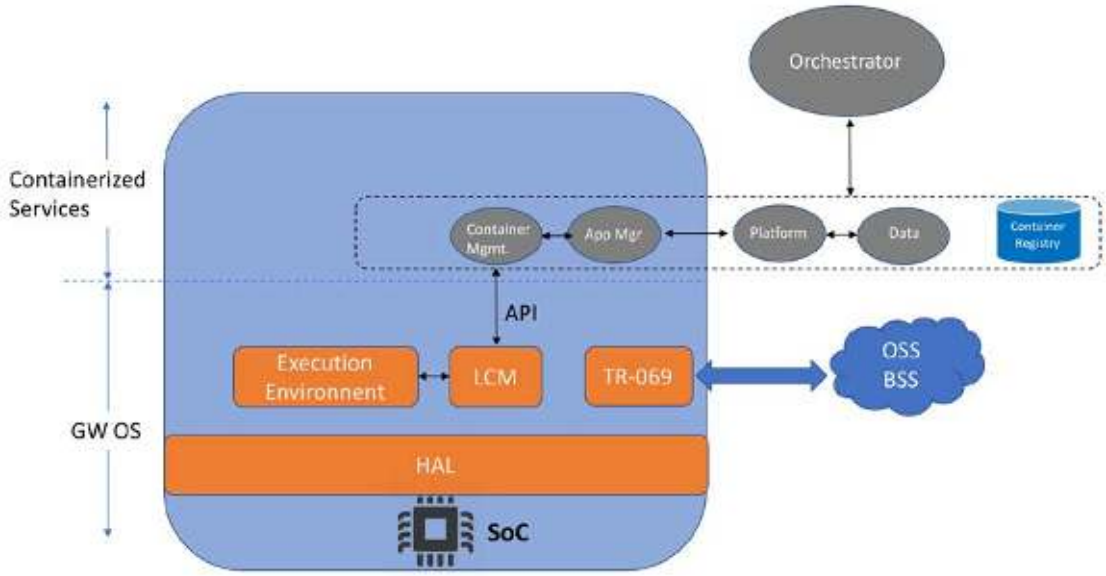

Figure 23 - Modern LCM layer for CPE

Conclusion

The decentralization of computing towards the edge, and the expansion of this edge compute paradigm towards the physical world of operator's customer premises equipment (CPE) is removing the last barriers of IoT application development and deployment to gain end to end control of their solutions in a more economical and viable way.

For the IoT project life cycle, implementing continuous delivery and being able to constantly add intelligence to these edge devices is absolutely mandatory to ensure that these solutions deliver on the promise of rapid value creation.

The intersection of DevOps practices and IoT Edge compute is still embryonic but growing day by day, to become an essential part of modern service delivery.

Furthermore, this Edge compute paradigm will become essential to unlocking the value of many IoT use cases, certainly for these use cases that are at the intersection of different application domains and industries.

The multitude and maturity of the current IoT platforms, and the advances in Big Data technology, will serve as the foundation, to build these new DevOps and life cycle management tooling and practices on, at an unprecedented scale. In this respect, it is important for the operator to include a strategy around modern life cycle management and containerization in their IoT initiatives.

When using lightweight virtualization techniques like containers on such CPE device start to become viable, they will provide a uniform, portable and secure environment for edge computing software components, solving the heterogeneous nature of today's embedded devices, as well as deliver a means to package all necessary dependencies in a uniform way in a more dynamic and agile way during the deployment phase.

Thanks to the combined industry effort of the last years in this area, the container technology has matured, both in terms of standardization effort, as from the technology perspective, by making open source container frameworks available for inclusion on these type of constrained devices, while still leaving room for innovation and improvements in various areas like security, operational management and privilege rights.

Albeit numerous researches have already been conducted on performance impact and footprint, further integration, evaluation and detailed measurement is required in this area on the applicability of operators' CPE equipment and their ecosystems.

While there are still numerous challenges around the immense scale of CPE deployment, security, privacy, standardization and interoperability issues, solving the infrastructural problems around automatization and the life cycle management of these new containerized edge computing components will be an absolute necessary first step.

Operators should expand their focus on edge and fog computing efforts to include CPE edge computing as a first-class citizen in their ecosystem.

Abbreviations

ACI	Application Container Image
AR	Augmented Reality
APPC	Application Container specification
Cgroups	Linux Control Groups
CRM	Customer Relationship Management
CPE	Customer Premises Equipment
CPU	Central Processing Unit
DBUS	DataBUS
DOCSIS	Data Over Cable Service Interface Specification
DRM	Digital Rights Management
eMTA	Embedded Multimedia Terminal Adaptor
ERP	Enterprise Resource Planning
ETSI	European Telecom Standards Institute
E2E	End to End
GDPR	General Data Protection Regulation
GW	Gateway
GPU	Graphics Processing Unit
IaaS	Infrastructure as a Service
IoT	Internet of Things
ISP	Internet Service Provider
IT	Information Technology
LCM	Life Cycle Management
LXC	Linux Containers
MEC	Mobile Edge Computing
ML	Machine Learning
MNO	Mobile Network Operator
NFV	Network Function Virtualization

ONAP	Open Network Automation Platform
OCI	Open Container Initiative
OT	Operational Technology
OS	Operating System
OSS/BSS	Operations and Business Support System
RKT	Rocket
SaaS	Software as a Service
SCTE	Society of Cable Telecommunications Engineers
STB	Setop-Box
SSH	Secure SHell
TCO	Total Cost of Ownership
TPM	Trusted Platform Module
UX	User Experience
VM	Virtual Machine

Bibliography & References

Bäckman, M., & Hagfjäll, F. (n.d.). *Application security for embedded systems*. (Department of Electrical and Information Technology Lund University) Retrieved from https://www.eit.lth.se/sprapport.php?uid=1032

CISCO. (n.d.). *Attaining IoT Value : How to move from Connecting things to Capturing Insights*. Retrieved from https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/iot-data-analytics-white-paper.PDF

Deloitte University Press. (n.d.). *Tech Trends 2017 : The kinetic enterprise*. Retrieved from https://www2.deloitte.com/content/dam/Deloitte/global/Documents/Technology/gx-tech-trends-the-kinetic-enterprise.pdf

Financial model Edge compute. (n.d.). Retrieved from https://wikibon.com/the-vital-role-of-edge-computing-in-the-internet-of-things/

Forrester Consulting. (n.d.). *Understanding the roles of LoB practitioners and SoCs in securing IoT environments*. Retrieved from https://www.forescout.com/wp-content/uploads/2017/11/Forrester-Survey-Fail-To-Plan.pdf

Managing IoT devices the DevOps Way. (n.d.). Retrieved from http://sites.tcs.com/blogs/research-and-innovation/managing-iot-services-the-devops-way?_sm_byp=ivvnhk12n6ssvddf

Masek, P., & Thulin, M. (2016). *Container Based Virtualisation for Software Deployment in Self-Driving Vehicles.* Retrieved from http://publications.lib.chalmers.se/records/fulltext/237650/237650.pdf?_sm_byp=iVV0Q1KZQDTQk6SH&_sm_byp=iVVSJFH8FRk5R4p6&_sm_byp=iVVSJFH8FRk5R4p6&_sm_byp=iVVSJFH8FRk5R4p6&_sm_byp=iVVSJFH8FRk5R4p6&_sm_byp=iVVSJFH8FRk5R4p6

McKinsey&Company. (n.d.). *Beyond Agile : Reorganising IT for faster Software delivery*. Retrieved from https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/beyond-agile-reorganizing-it-for-faster-software-delivery

MORABITO, R. (n.d.). *Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation*. (Ericsson Research, Ericsson AB, Jorvas 02420, Finland) Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7930383

Zhou, W., Zhang, Y., & Liu, P. (n.d.). *The effect of IoT New Features on Security and Privacy : New Threats, Existing solutions and Challenges yet to be resolved*. Retrieved from https://arxiv.org/pdf/1802.03110.pdf

