The Industrial Internet of Things
Distributed Computing in the Edge

Version 1.1

2020-10-18

John Zao (NCTU), Chuck Byers (IIC), Brett Murphy (RTI), Salim AbiEzzi (VMware) Don Banks (Arm), Kyoungho An (RTI), Frank Michaud (Cisco Systems), Katalin Bartfai-Walcott (Intel)

CONTENTS

Figures

Distributed computing moves the capabilities of cloud computing in data centers closer to intelligent IoT devices, towards the edge.[1] This can increase performance and efficiency, but it is essential in industrial settings for optimizing time-critical industrial processes because critical control processes that do not respond in time can become dangerously unstable.

This document describes a framework for distributed computing in the edge that brings computation, networking and storage closer to data producers and consumers for the Internet of Things (IoT). It is intended for IoT system architects and implementers who are working with edge systems comprising IoT devices, edge computing nodes, networking equipment and data center servers. It:

- provides a structural and functional framework for distributing computing in the edge,
- defines the key architectural concepts employed for distributed computing in the edge,
- specifies the essential capabilities of this edge system's elements,
- pays special attention to essential security and management functions and
- describes the essential interfaces among these elements.

System architects can use the framework[2] as an architectural template that helps derive a concrete distributed computing architecture. Others, such as operations technologists, information technologists, network and business managers may also find it useful to understand the essential elements of distributed computing in the edge.

[1] The IIC Vocabulary defines *edge* to be the "boundary between the pertinent digital and physical entities, delineated by IoT devices" and *edge computing* as "distributed computing that is performed near the edge, where nearness is determined by the system requirements". That is, "edge" is used in the former as a boundary and in the latter as a region. This report uses "*at* the edge" to mean at the boundary (though that boundary is fuzzy), and "*in* the edge" to mean in the region. Similarly, "*to* the edge" connotes distribution away from data centers.

[2] This document was derived from the "OpenFog Technical Framework" compiled by the members of the OpenFog Consortium, which joined forces with IIC in January 2019. That document described functional characteristics and interfaces of fog elements, fog nodes and fog systems existing within the Data Center-to-Things Continuum. IIC pivoted this framework to distributed computing in the edge.

1 WHY DISTRIBUTE COMPUTING TOWARDS THE EDGE?

Edge computing moves the capabilities of cloud computing typically associated with data centers into the edge, where they are closer to IoT devices. With edge computing, data, networking, storage and computing are distributed throughout layers of edge computing nodes from IoT devices to the data center, distributing the economies of scale of cloud capabilities throughout the system. The migration of cloud capabilities into the edge allows data, storage and computation to gravitate to where it can be handled most efficiently, whether in a data center or the edge.

1.1 BUSINESS BENEFITS

Cloud computing capabilities in data centers offers flexibility and scale to enterprises. We can extend those benefits towards "things" in the real world, towards the edge. The flexibility to decide where to perform computation on data improves performance and reduces costs. Computing in data centers induces bandwidth costs as data is transmitted, while sensors are generally limited in terms of what computations they can do. Computing in-between can collect data from multiple sources, fuse and abstract as needed, and compute *right there*. Computation can take place near to the repositories of the data, rather than consuming bandwidth shifting the data to a data center. The propensity for computation to take place "near" the data is called *data gravity*, and is a key motivator for distributing computing.

The flexibility to decide where to store data improves performance and reduces costs. Moving data takes time and costs money. It may also expose to security and privacy risks. Data segmentation based on compliance boundaries imposed by regulation in different jurisdictions supports disciplined security. If data is held on premises with no connection to the internet, it cannot be compromised by the proverbial hacker in his parents' basement.

Performance of IoT applications is often an overarching catalyst for moving previously data-center-based workloads to the edge. Executing close to where the data is generated in the physical world, rather than passing data up to a data center and back down, reduces the time lag (*latency*) and indeterminate time (*jitter*) between receiving data and acting on it. Faster is usually better, but it is essential for optimizing time-critical industrial processes. Critical control processes that experience too much latency or jitter can become dangerously unstable.

When several logical functions execute on single or multiple physical devices, the owners of edge systems can serve multiple customers on a shared infrastructure, increasing deployment efficiency. This is called *multi-tenancy*.

Multi-tenancy enables scalability, so that devices in different rooms and floors of a smart building, for example, can be aggregated into a smart neighborhoods and districts of a city. *Scale-up* is the addition of computational, memory, storage and networking resources. *Scale-out* is

when more same-function computing resources are deployed in the data center or the edge. (And we may also scale down and scale in).

To respond to varying application mixes and workloads, we need *elasticity* to be able to add or remove resources quickly by reconfiguring the system to execute on more or fewer edge computing nodes. This elasticity supports, for example, first responder teams when computation and connectivity needs fluctuate in an emergency. This agility to accommodate rapid change is important in many edge systems.

Redundancy, and the associated fault-recovery software, creates a fault-tolerant system that enables critical services to continue in mission- and life-critical applications, even as nodes or links fail. A single degraded node or link can be routed around and failures avoided—one of the original motivators for the internet itself.

Distributed computing in the edge is synonymous with edge computing. It is fundamental to distributed applications such as connected cars, enabling a platoon of cars traveling at high speed to communicate and then travel closely together, avoiding accidents and saving road space.

Distributed computing in the edge also enables new applications and features, which can increase efficiency, revenue and value for the customer. For example, smart grids are already feeding distributed energy resources into the power network, reducing energy costs and even providing discounts to customers.

There are some specific issues that must be considered when distributing computing to the edge:

Truck-roll: Adding or replacing equipment outside the data center is costlier as it requires logistics. Moreover, failures at the edge may require more urgent repairs. (Failed resources in a data center don't always need to be replaced due to high redundancy. Modern management software for data centers simply retires failed equipment and carries on without it.)

Form factor, packaging and hardening: Equipment is packaged differently in the data center from outside it. Data centers are typically large, environmentally controlled, high-security buildings with arrays of equipment racks, whereas edge equipment is often in less controlled environments such as a vehicle, street-corner cabinet or on a factory floor, requiring better hardening and resilience to environmental extremes and power variations.

Tampering: Equipment outside a locked data center needs better tamper protection, for example, locked cabinets and the exclusion of external ports that could heighten security risks.

Remote monitoring and management of edge equipment is a challenge. For example, over-the-air software updates for cars requires resilience to packet loss and interrupted connections to ensure successful updates.

Network bandwidth: Bandwidth to transport data between IoT devices and data centers is often expensive and less than 100% reliable. By moving computation and storage nearer to where it is used, less data is moved through the network.

Energy efficiency: Edge equipment is often constrained by available energy, especially if run on batteries or renewable sources. Higher power also creates cooling complexities.

Data residency may be governed by policy and regulation, such as the Health Insurance Portability and Accountability Act (HIPAA privacy) or government prohibitions. Some data, such as trade secrets, cannot leave the premises.

These factors, especially when taken together, motivate distributing computing to the edge. What was once limited to the data center can now be distributed, enabling, in particular, digital transformation in the industrial internet of things (IIoT).

1.2 DESIRED PROPERTIES

To gain these benefits at low risk, distributed computing in the edge requires several properties:

Above all, an IoT system must be *trustworthy*. This property is the conjunction of security, privacy, safety, reliability and resilience, and is especially important in industrial IoT systems where lives, limbs and the environment are at stake.

Manageability: Distributed computing in the edge has many widely dispersed, interconnected nodes. These nodes participate in a network and each run multiple applications and services which, in turn, interact with each other to deliver system functionality. Each node, IoT device and application service needs to be discovered, commissioned, monitored, updated and decommissioned over the lifetime of the system.

Composability is the ability of edge elements to aggregate structurally and functionally with one another. Edge elements must be separable and orthogonal to permit the substitution of one element associated with an interface without affecting other elements in the system. In turn, the resulting application services may be composed to participate in higher-level applications, improving time-to-market as interoperable elements are quickly assembled into various systems that can perform the required functions. This can create a multi-party software marketplace and helps avoid supplier lock-in.

Autonomy: edge nodes can effectively operate (at least temporarily) in isolation. When connectivity to a data center or adequate computational throughput is unavailable, tasks can be executed within nodes away from the data center, or queued until connectivity is restored.

Interoperability is the ability of edge elements to exchange information with one another and interpret this information consistently. Seamless interoperability between edge computing infrastructure services needs technical, syntactic, semantic interoperability and openness, as described in detail in the sidebar below.

Openness means that all the requirements and implementation attributes of the elements can be implemented by anyone. Any supplier is free to produce their own versions of these elements that can interoperate and interchange with elements provided by other suppliers. Openness supports interoperability.

Levels of interoperability are defined in the IIC's Industrial Internet Connectivity Framework (IICF) by what can be exchanged. The levels build one upon another:

> *Technical interoperability* is the ability to exchange information as bits and bytes (e.g. pencil scribbles), assuming that the information exchange infrastructure (e.g. pencil and paper) is established and the underlying networks and protocols are unambiguously defined.
>
> *Syntactic interoperability* is the ability to exchange information in a data structure (e.g. using words from a language), assuming that a common protocol to structure the data is used (e.g. the language's alphabet and rules of grammar) and the structure of the information exchange is unambiguously defined (e.g. whitespace, punctuation). Syntactic interoperability requires that technical interoperability be established.
>
> *Semantic interoperability* is the ability to interpret the meaning of the exchanged data unambiguously as information in the appropriate context. For example, a "sentence" such as "There are three" could mean anything. Three apples? Three cars? Three degrees? Three leaf blowers?

For example, a service hosted in the edge receives shared data over a link (technical) and unmarshalls the sequence into a data structure with an array of 2 by 100 floating point values (syntax). It then passes the array to a health monitoring service that runs an event-detection algorithm on the array where the first row is the time history and the second row is a temperature (semantic). Syntactic interoperability is critically important also because it allows software to share data structures, so they can work together even if written in different languages or transmitted by frameworks built by different vendors. It also enables generic tools that can introspect and process the data. With both syntactic and semantic interoperability, edge computing services can exchange and consistently interpret data with other edge services, IoT devices and the data center. If that data exchange is further built upon open standards, systems or software from multiple suppliers can reasonably be expected to interoperate.

These properties, and the technical mechanisms used to support them enable the business benefits that can be derived from distributed computing in the edge.

2 A FRAMEWORK

A distributed computing framework is a configurable and scalable architecture that can extend cloud computing capabilities from data centers to the edge. Edge computing nodes and the communication paths between them are two essential parts of an edge system.

An *edge system* spans IoT devices, sensors and actuators, and IoT gateways, mediating those devices with the internet, servers and data ingestion services at the data center. These elements, taken together, make up an edge system. Edge systems perform the three primary functions of IoT: collecting, analyzing and acting. They collect data from sensors and other IoT devices. They analyze this data and reach conclusions about what must be done and finally, they act, by storing results or sending messages to actuators and other IoT devices to control the physical world.

Edge computing is distributed computing performed near the edge, where nearness is determined by system requirements. Edge computing brings many of the important capabilities of cloud computing to the edge. *Edge* is the boundary between the pertinent digital and physical entities, delineated by IoT devices. This contrasts with data centers as the data processing and storage resources are closer to where data is produced from sensors (in smart buildings, traffic lights, oil rigs, airliners and cars) or consumed for actuating devices. Some standards for edge computing architectures are already in commercial deployment, for example ETSI Multi-access Edge Computing (MEC).

A *data center* is a facility containing a collection of connected equipment that provides communication, computing and storage resources. In the context of an edge system, a data center provides computation as a service, using virtualized computing and the economies of scale that come with it. It could be public (e.g., Amazon Web Services, Microsoft Azure and Google Cloud Platform), private or a hybrid or multi-cloud approach. *Cloud computing* is a paradigm for enabling network access to a scalable and elastic pool of shareable physical or virtual resources with self-service provisioning and administration on-demand. Cloud computing was traditionally associated with data centers, but can now be applied at the edge (and offered as software stacks by some of the same providers).[3]

The data center and its servers, networking equipment and storage engines constitute the highest layer of the distributed computing framework. They are typically located in secure facilities, and managed as part of a large, scalable service provider, private or hybrid network. They are interconnected with the edge computing nodes via high speed data networks (typically fiber optical facilities in the 100Gb/s throughput range). Many of the highest-level analytics, storage and management functions that support edge systems reside in data centers. Data centers often implement *hyperconverged infrastructure*, a coherent combination of virtualized

[3] There are many stacks, both proprietary and open. These are the most popular. See: https://aws.amazon.com/greengrass/, https://azure.microsoft.com/en-us/products/azure-stack/edge/

computation, networking, and storage subject to unified management, often implemented on inexpensive servers. These techniques can also be applied away from the data center.

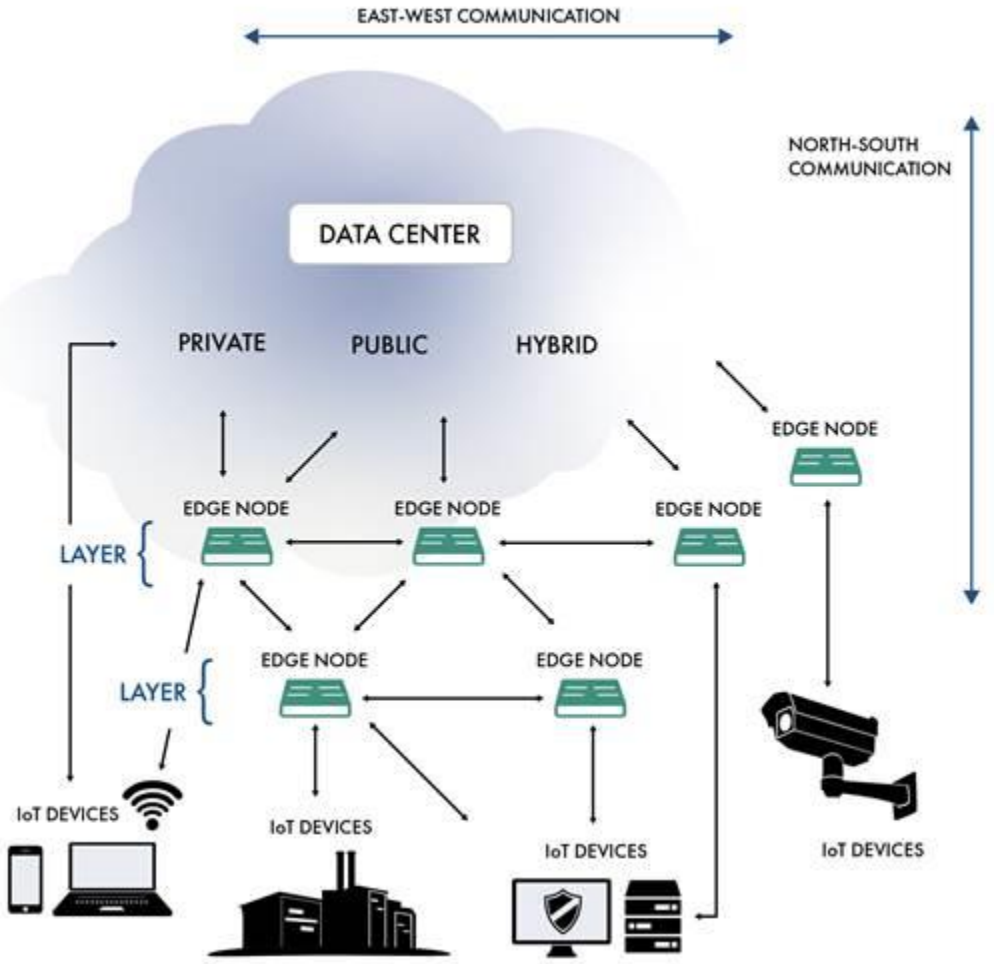

Figure 2-1: Illustration of Multi-layer Edge System

The *IoT devices* at the bottom of Figure 2-1 include all sensors, actuators and other endpoint devices in the IoT system, such as user interfaces, smart mobile devices, displays, industrial control systems and integrated intelligent endpoints. Sensors measure physical parameters associated with the real world, digitize these readings and send them through the layers of edge computing nodes toward the data center. Some implementations complete all their processing and storage activities in one or more layers of edge computing nodes, while others process and store their data in a data center. When the processing is complete, actuators can optionally accept digital commands from the data center and edge computing nodes and change physical parameters in the real world, creating a closed-loop control system. Together the IoT devices interface the digital world to the physical world.

Some IoT devices can be quite simple, for example, a temperature sensor that generates one reading a minute. Others can be complex, for example, a security camera that has significant processing power capable of executing local video analytics, or a control system that accepts set points as inputs and uses those parameters to guide real-time processes. The more sophisticated IoT devices are, the more they are considered a conjunction of the raw sensor and a layer of edge computing node, integrated into a single physical and logical entity.[4]

Edge computing nodes are individually addressable and manageable elements in the IIoT distributed computing system. They offer computation, networking, storage and control services closer to the data-producing sources or information-consuming users.

Edge computing nodes are deployed in one or more *layers* between the IoT devices and the data center. Edge computing nodes in the same layers can communicate east-west, and with other layers north-south. They may also be grouped in clusters, so that a cluster may be managed collectively in the same way as a node. The number of layers and arrangement of clusters may vary from a few (as when IoT sensors are connected directly to the data centers) to a dozen or more (as in smart buildings and smart cities, in which devices in different rooms, wings, floors of a building and different neighborhoods of a city are organized into a deep continuum of layers).

Fog computing refers to an architecture pattern for trustworthy, distributed edge computing that has been used previously, especially by the OpenFog Consortium (which joined forces with IIC in January 2019, and originated of some of the material in this document). Subtle distinctions that may exist between the terms edge and fog are not considered important in the marketplace, so we use “edge” exclusively.

Our focus for distributed computing in the edge is everything between the lowest layer of the data center and the IoT devices shown in Figure 2-1.

2.1 EDGE SYSTEMS

One approach for an edge system comprises the functional components illustrated in Figure 2-2. Above the dotted line are the edge system-level functions that act across the entire system and are typically centralized and virtualized—usually in a data center.

The *system security management* module manages security at the system level, and typically acts over multiple edge computing nodes. This is where system-level security policies, security analytics, crypto key management and similar centralized functions are implemented. Data centers are the preferred hosting location for this module.

The *system provisioning and management* module is responsible for the centralized configuration and management capabilities of the edge system. It performs the centralized functions that are controlled by a network operations center, and distributes the control

[4] This is why the edge boundary is fuzzy.

messages and collects the status messages from multiple edge computing nodes. This module is typically also implemented in data centers.

Below the dotted line are multiple edge computing nodes, arranged in overlapping layers. Within each edge computing node, the mainline flow of computation is in the center, the end-to-end security capabilities on the left (colored pink), and the management capabilities (gray) on the right. Subsequent diagrams decompose the functions within the trusted computing module, security and management boxes, respectively. The application execution environment runs application software.

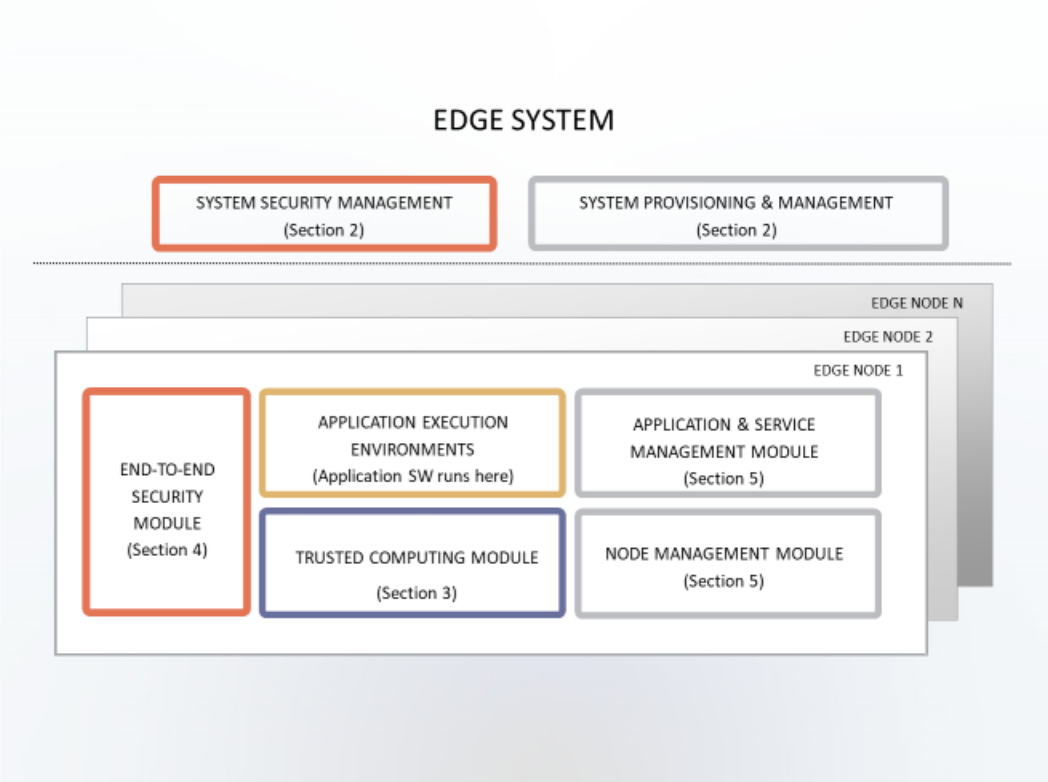

Figure 2-2: Functional Composition of an Edge Computing Node

The *trusted computing module* is built on innately trusted hardware, and includes the trusted execution environment(s) with their chains of trust linked to the hardware root-of-trust to provide trusted application services to the rest of the edge computing node. There may also be mechanisms that allow third parties to verify the security state of the node, so that other edge computing nodes may verify that they can trust this node. This is the foundation upon which trustworthy applications can be built, secured and managed. The trusted computing module and its subcomponents are described in section 3.

The *end-to-end security module* ensures communication and information security of the messages passing between the application services running in the edge computing nodes. Its security capabilities are interwoven into the framework. All the security functions are

continuously running in parallel with the mainline functions. The end-to-end security module and its subcomponents are described in section 4.

The *node management module* is responsible for configuring, monitoring and updating the system hardware and software resources in an edge computing node.

The *application and service management module* is responsible for deploying, configuring, monitoring, scaling and updating the application services residing in the edge computing node. The node management module and application & service management modules and their subcomponents are described in section 5.

The *application execution environments* comprise application software that runs on the edge computing nodes. This is where the mainline software the user depends upon runs, and is the reason for deploying the edge system in the first place. The software running here may be untrusted, but the other components of the edge computing node and edge system "wrap" it in a way that makes the overall edge system trustworthy and manageable.

2.2 COMMUNICATION PATHWAYS

Communication between edge computing nodes is needed to distribute data and computation across nodes.

For this approach, *north-south communication* links traverse layers from the data center to things, running in a generally vertical direction on Figure 2-1. These are the links that IoT devices use to talk to edge computing nodes, edge computing nodes use to talk to edge computing nodes on adjacent layers, or edge computing nodes use to talk to the data center. The bandwidth on each of these layers of links is influenced by the number of sessions and users supported on parts of the networks, and the size and generation rates of the data sets transmitted. Layers closer to the data center are more likely to be higher bandwidth pipes (such as fiber) and serve the aggregate traffic needs of a large number of devices below them. Links closer to the devices are more likely to be wireless or lower bandwidth copper links. Their bandwidth is often constrained by the available energy in small or battery-powered devices. As data moves up the layers, nodes along the way digest, analyze and aggregate it, balancing the capacity on all links.

East-west communication paths in this architecture pattern run in a generally horizontal direction of Figure 2-1, orthogonal to the north-south links. They interconnect edge computing nodes on the same layer, and serve three primary purposes.

First, they help to balance load between peer-level edge computing nodes. If one edge computing node is experiencing an overload, it can shuttle some of its traffic to a nearby edge computing node that is less heavily loaded. This balances the loads on the nodes in a layer and improves overall network resource utilization.

Second, the east-west links improve resilience and fault tolerance. If a higher-layer edge computing node or the north-south link to it has failed, the blocked traffic can traverse one or

more east-west links and route around the problem. Also, for storage applications, these links can make and retrieve redundant copies of stored files across several nodes. In this way, if a single storage device or edge computing node fails, the redundant copies on other nodes on the same layer still have the critical data.

Third, east-west links are used to build context across regions of a system. One edge computing node can send results to adjacent edge computing nodes, where they are fused with local results to create a larger context. As a concrete example, in a security camera network, each edge computing node analyzes the video from the cameras local to it and describes the positions, speeds and directions of recognized objects to adjacent edge computing nodes. They correlate objects on the boundaries of their cameras, fusing more views of the same object or tracking as it moves across the views of many edge computing nodes. This enhances situational awareness, and is more scalable and resilient than can be achieved with one huge edge computing node processing signals from all the cameras.

2.3 DEPLOYMENT MODELS

The topology of edge computing nodes can be optimized to match the requirements of the applications running on the network. Some networks will favor fewer layers between the data center and the IoT devices but support more peer-level IoT devices per layer. Other network deployments prefer to split their functionality between a larger number of layers, with fewer nodes in each layer.

The choice of topology depends upon the specific requirements, performance expectations, and functional partitioning of the application services running on the edge computing nodes. Some are dominated by the need for high single-thread processing performance, and would benefit from more layers. Others can execute efficiently in many parallel edge computing nodes. Then, fewer layers with more edge computing nodes per layer makes sense. Storage operations can also be separated and optimized across multiple layers and nodes. The way that edge computing nodes are assembled should reflect the data gravity requirements of the applications.

In a simple edge deployment, an edge computing node is an assembly of hardware and software components that implement the functions shown on Figure 2-2. Such a stand-alone edge computing node can be installed anywhere in the edge system to provide computing, networking and storage services close to data producers or consumers.

In a slightly more complex model of edge deployment, multiple logical edge computing nodes may be instantiated in a single physical edge computing node. They share the same hardware platform, but are fully isolated from each other. This deployment model is modular, scalable and efficient. It is the primary support mechanism for multi-tenancy.

In the most complex model, a logical edge computing node is assembled from the one or more physical or logical[5] edge computing nodes. One version of this merges the capabilities of multiple physical edge computing nodes, which may be peers on same or adjacent layer(s), to handle a computation, networking or storage load heavier than a single physical edge computing node can manage. In a variation, multiple physical edge computing nodes are grouped into a fault tolerant cluster, so that a failure in one of the edge computing nodes will be mitigated by its peers.

2.4 SUPPORTING TECHNOLOGIES

Any system implements supporting technologies needed to realize the system. A system that supports distributing computing in the edge will use at least these supporting technologies:

Ubiquitous *connectivity* is an enabler of distributed computing: the ability of edge elements to exchange information. Without connectivity, distribution would be impossible.

Virtualization is the ability to separate logical functions from the physical device. Virtualization supports increasing load dynamically. Some safety- and time-sensitive environments (automotive for instance) impose restrictions on virtualization flexibility so that there is no oversubscription of resources. Virtualization is enabled by hypervisors, which are supervisory agents that create, run, and monitor virtual machines in a host machine. Hypervisors must account for the widely distributed infrastructure and the challenging management environment of edge computing nodes.

A *software container*[6] is a structure that allows a single deployable image or data structure to be used across different operating platforms. They usually include a user-space application and its dependencies as well as networking and storage contexts. The standards-based interoperability of containers provides a high degree of confidence that a developer's applications will run on deployment servers (in the data center or the edge), different from where they are developed and tested. Container workloads can run at different places, from the data center to the various layers of edge and IoT devices, and can change dynamically to optimize performance of the overall system. A *container runtime* is software that manages containers and container images on a node. The Open Container Initiative (OCI) has developed a standard specification for container runtime and provides a reference implementation called runC. Edge systems can manage and orchestrate containerized application services through a container runtime residing on each edge computing node. Techniques such as service mesh can handle complex communication tasks between microservices.

Manual operation of these devices and nodes is impossible. We need *orchestration,* the ability to schedule and distribute computing workloads, storage actions and network bandwidth across

[5] We contrast "physical" with "logical". The term "virtual" is also used, but that can be confused with the related concept of "virtualization".

[6] "Software container" is rigorously defined in the IIC vocabulary. We use "container" hereafter.

resources automatically, and modify them dynamically as system conditions change. This is driven by policy to minimize any requirements for human intervention in the operation of the system. Container orchestration technologies like Kubernetes (k8s) provide a set of services to help deploy and manage distributed nodes and applications in edge systems. Specifically, they help manage distributed applications by scaling them up and down, performing updates and rollbacks, and self-healing. Changes to the deployed system are indicated through changes to the guiding file declaration, which k8s automatically detects and to which it attends.

Load balancing is necessary to achieve the desired scalability and performance properties in edge systems. Workloads are often much larger than can be run on a single edge computing node, and load balancers distribute different parts of the workload to different edge computing nodes. This distribution could split complete applications from different users onto different nodes, or divide the work into different phases split among nodes at different layers.

Heterogeneous computing and storage is the ability to use the most efficient hardware implementation for the application services workload. Heterogeneous processors include traditional CPU cores, graphics processing units, digital signal processors, tensor processing units and field programmable gate arrays. Heterogeneous storage could include cache, RAM, flash, rotating disk, networked storage and off-line backup. These capabilities are already widely deployed in data centers, and are migrating quickly to edge computing nodes.

3 TRUSTED COMPUTING MODULE

Trustworthiness is the conjunction of five properties: security, privacy, safety, reliability and resilience. This section describes how to extend trust from hardware up to the application level.

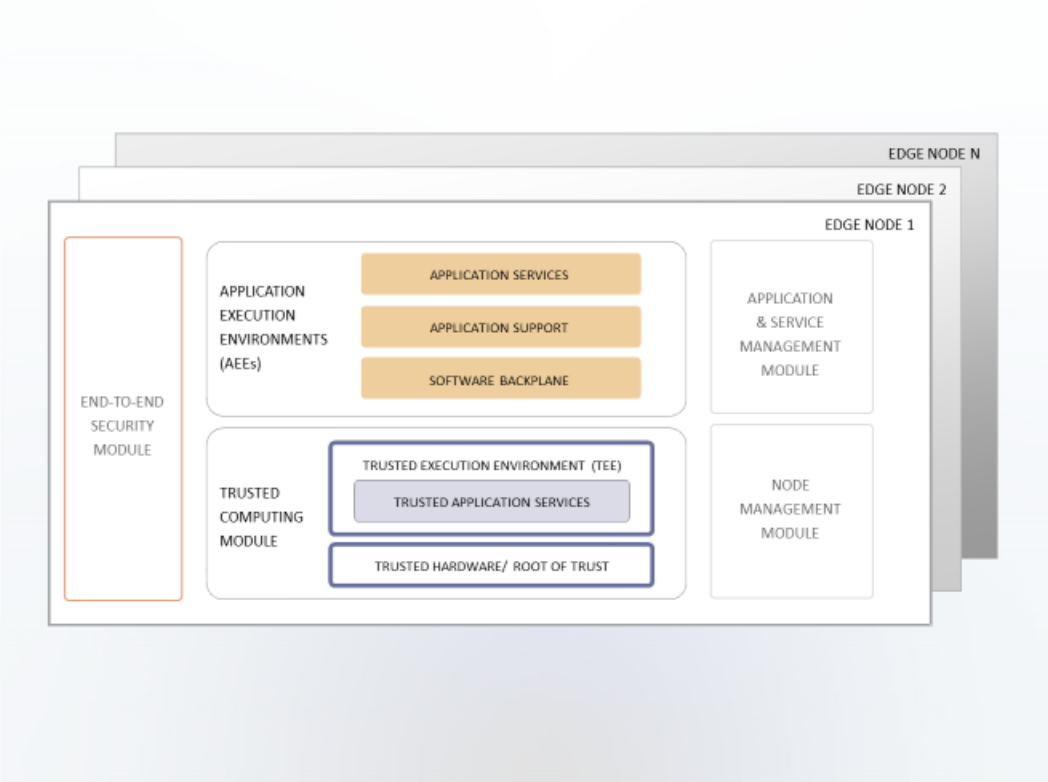

Figure 3-1: Essential Components of a Trusted Computing Module

3.1 BOOTING TRUSTED HARDWARE

The trusted computing module[7] comprises two primary components: trusted hardware equipped with hardware root-of-trust and trusted firmware, and one or more trusted execution environments[8] that execute code using the trusted hardware. The trusted execution environment includes a subcomponent of trusted application services. These are the foundational components of a trustworthy system (Figure 3-1).

[7] In this context, a module is a set of hardware or software resources that can be composed together to build the required system properties.

[8] Also called “secure enclaves”.

3.1.1 TRUSTED HARDWARE

Trusted hardware provides isolation as well as data and execution access-privilege guarantees. There is no definitive definition of hardware root-of-trust (hRoT[9]), but at a minimum, each device should have immutable boot code in read-only memory, with access to the cryptographic keys and certificates needed to verify the subsequent loads of boot. A unique device ID used to authenticate the device and a unique device private key to protect platform secrets are also part of the hRoT. The trusted hardware may run an operating system or a hypervisor. Operating systems (OSs) create and isolate processes from one another and hypervisors create virtual machines that contain OSs and applications isolated from one another.

3.1.2 TRUSTWORTHINESS OF BOOT PROCESS

There are various ways to boot a system in a trustworthy fashion. They all employ trusted hardware, without which no claims can be made about a system's trustworthiness. At system power-on or reset, the trusted hardware equipped with a hardware root-of-trust transfers control to an immutable first load of boot. The hRoT then loads and verifies the authenticity and integrity of the next load of boot. Each successive boot image is loaded by the previous image and the integrity of the image is checked before control is transferred to it. Upon completion of each stage of this process, secure log entries can be created, as proof that the steps necessary to create the chain of trust all completed in the correct order. The signed boot log can then be used to attest to the security state of the platform using a third-party verifier, called *remote attestation*. This gives the edge system confidence that an edge computing node is trustworthy.

The UEFI Secure Boot[10] is commonly used in the server community. On the other hand, different proprietary mechanisms are employed by embedded system vendors to implement secure boot. These proprietary approaches inevitably lead to proprietary mechanisms for system updates as well. To avoid unnecessary complication, Linaro and its members is introducing the UEFI interface to U-Boot[11] so that embedded operating systems can leverage UEFI Secure Boot, UEFI Measured Boot and secure Update Capsules with U-Boot.

Secure firmware and software update: If a device is to remain secure, unauthorized updates must be prevented. Updates should be verified by a secure service before being stored in a secure location. As code is loaded and verified at runtime, the integrity and authenticity are guaranteed.

Physical device access protections: If unauthorized persons can physically access the device and the possible threats outweigh the cost of preventing a breach, then tamper-resistant hardware with tamper-detection mechanisms are necessary. The device behavior on detection of a

[9] Initial RoT (iRoT) is an equivalent term.

[10] Unified Extensible Firmware Interface (UEFI) Specification v.2.8, §23.

[11] UEFI is a specification that is implemented by commercial products (including American Megatrends, Phoenix and Insyde) or open source software (EDK2 and now U-Boot).

tampering event should be a programmable policy ranging from bricking the device to logging the event and continuing.

3.1.3 TRUSTED EXECUTION ENVIRONMENT

A trusted execution environment (TEE) provides an isolated environment for the initial boot. A TEE isolates trusted code and data from untrusted code and data that might reside in applications. TEEs may provide trusted services during the life of the system or the life of the application address space. Some examples of trusted services are secure key generation, storage and retrieval, and secure, persistent storage.

TEEs may also host trusted applications; examples of trusted applications in the edge include confidentiality and integrity protection of sensor data. And TEEs are also at the heart of enhanced trusted computing models, specifically:

Confidential computing: Cloud technologies allow inter-tenant isolation while confidential computing technologies isolate tenants from infrastructure administrators.

Multi-ownership: A paradigm of confidential computing that preserves information privacy and sovereignty and enforces usage governance to satisfy regulations and business policies.

Hardware features, and maybe microcode, provide a secure isolated and protected environment so the TEE is isolated from the application execution environments because secrets cannot cross the boundary between trusted and untrusted execution environments. However, the application execution environments may request a service to execute code in the TEE.

3.2 APPLICATION EXECUTION ENVIRONMENTS

Application execution environments (AEEs) provide the higher-level infrastructure riding upon the TEE to execute the application services. They consist of a software backplane that enables the composability of modular software, and application support modules that provide common platform support and application programming interfaces (APIs) to the application services.

The bare-metal OS kernel and hypervisor kernel are implicitly trusted; they control privileged hardware and software contexts in the AEE. Kernels can still withhold service in the AEE by not dispatching some work and they can still create and destroy processes and VMs. They do not have to be trusted to the same degree because they cannot access the code, data or the execution context of the TEE. If a service is withheld by the AEE, confidentiality and integrity guarantees are still in place, but availability guarantees are not.

The untrusted execution environment provides an execution environment in which hypervisor services, OS services, containers and application services run. It runs untrusted code, and TEEs may be contained in a non-secure application service address space using a protected memory area within the address space that is not accessible to any other address spaces in the AEE.

A *software backplane* is analogous to a hardware backplane that interconnects hardware modules, but applied to modular software components—it provides an elastic computation infrastructure. Distributed applications communicate with each other via the connectivity elements of the software backplane. The software backplane includes the OS, hardware drivers, network connectivity functions and other software infrastructure elements to support the application services.

Application support services, like data communication protocols, data stores, web servers, runtime engines often run as containers or micro-services on top of the software backplane and provide shared services to the application services running above. These application support functions are often implemented as common platforms, usable by all the application services that may be running concurrently on a multi-tenant edge computing node. These platforms (and the layers below them on Figure 3-1) are often horizontal, meaning they are equally applicable to all vertical markets or IoT application domains.

Application services implement the user's software and interact with the rest of the edge framework. It includes various application-specific protocols, algorithms and APIs that enable workloads to be executed on edge computing nodes and edge systems. The code for application services is often written by experts in the specific domain areas that are supported by an edge computing node (for example, transportation, medical, factory automation or smart city applications would be created by experts in these respective areas).

3.3 INTERFACES

A set of standard APIs have been established to allow application services to receive secure services through the software infrastructure from the trusted services running in the trusted execution environment. Standard APIs include the Cryptographic Token Interface (PKSC#11), the Trusted Platform Module (TPM) Mobile Command Response Buffer (CRB) Interface and the GlobalPlatform TEE Client API.

The Cryptographic Token Interface (a.k.a. Cryptoki or PKCS#11) is a Public-Key Cryptography Standard (PKCS) that defines a platform-independent API to interact with cryptographic tokens such as hardware security modules and smart cards embedded in the trusted hardware platform. The API defines commonly used cryptographic objects including RSA keys, X.509 Certificates, AES/DES keys as well as the functions to create, modify, use and delete those objects. Most commercial certificate authority software uses PKCS #11 to access the certification authority signing key or to enroll user certificates.

The TPM 2.0 Mobile Command Response Buffer Interface is a kernel interface to a TPM. For the software infrastructure to communicate with a TPM embedded in the trusted hardware, it puts a command in the TPM command buffer and sets a flag. In response, the TPM executes the command in the buffer, puts its response in the response buffer and clears the flag.

The GlobalPlatform TEE System Architecture defines two APIs. The TEE Internal API specifies a common interface for trusted application services in the TEE to access the trusted hardware. The TEE-Client API, on the other hand, is an external interface that allows applications in an AEE to call trusted application services to perform secure operations on their behalf. These secure function calls are converted by the TEE kernel into calls to the corresponding trusted application services. An implementation of the TEE Client API must contain a client API library and an AEE communication agent shared among all client application services. Currently, the Open Portable Trusted Execution Environment (OPTEE) under Linaro and the Trusty TEE under Android are the two popular open-source implementation of GlobalPlatform TEE specification. Trusty is more light-weight than OPTEE.

4 END-TO-END SECURITY MODULE

Following International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) X.805 Recommendation,[12] the End-to-End Security Module in this architecture comprises three components: communication security services, information security services and security incident monitoring & response. Figure 4-1 illustrates these components and shows the interfaces among them.

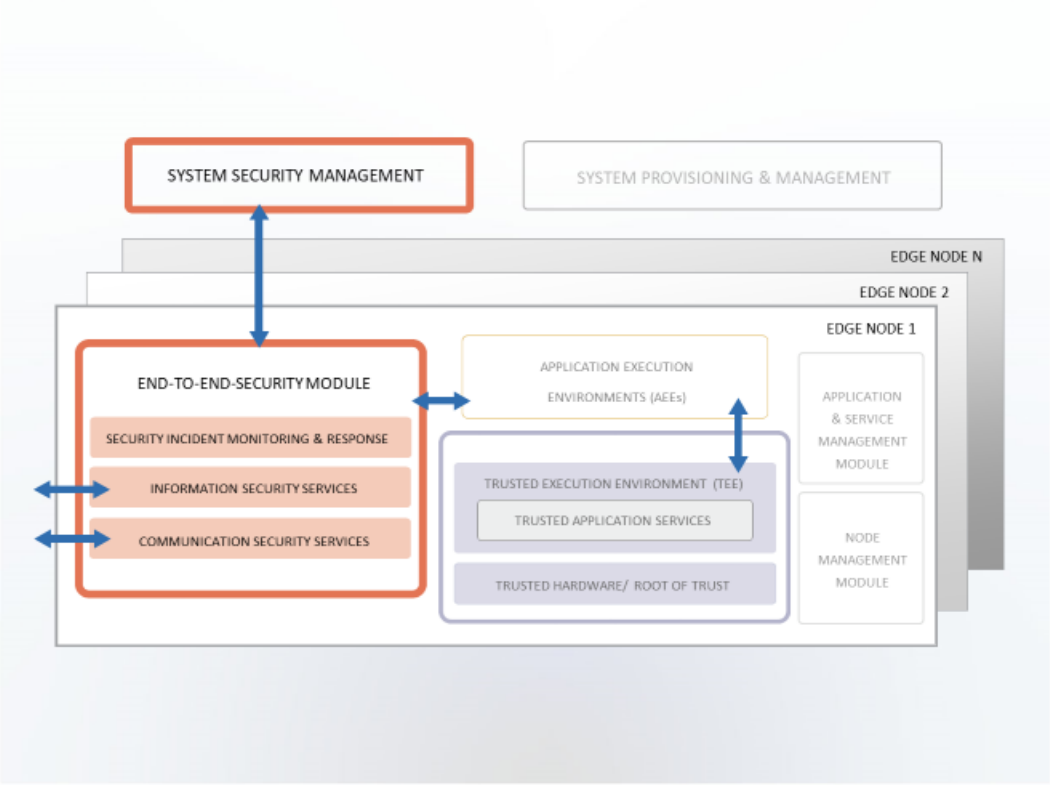

Figure 4-1: Functional Components of the End-to-End Security Module

4.1 END-TO-END SECURITY SERVICES

End-to-end security services, for communication and information security, run in an end-to-end security module in each node. In a system supporting hardware virtualization and application service containerization, those services should run between the communication ports of the containerized application services regardless of whether these services are located within the same or different edge computing nodes. Cryptographic functions that implement end-to-end security must be performed by the trusted application services running in the trusted execution environments instantiated in the trusted computing modules of the edge computing nodes. These services must be requested through the external service interface provided by trusted

[12] Zeltsan, Zachary. "ITU-T Recommendation X. 805 and its Application to NGN." *_28* 4 (2010).

application services in the trusted computing module. Information flowing between the trusted execution environment and the application execution environment must also be protected by information isolation among different application services and between multiple tenants.

Information security services: Information exchanges among application services should also be protected, especially access control based on identities, attributes, roles or (access control) capabilities of the communicating parties. They should encompass authentication, authorization and accounting services. The scope of protection should include:

- *data-in-use:* data being processed or residing in memory during computation,
- *data-at-rest:* data maintained in local or remote mass storage and
- *data-in-motion:* data moved through communication networks.

Communication security services: Information exchanges among application services must be protected to ensure confidentiality and integrity of the exchanged information using strong cryptographic functions, and the authenticity of the communicating parties through verification of their identities. These services should include the following, as recommended in ITU-X.800:[13]

- confidentiality
- integrity
- authentication
- (optionally) nonrepudiation

They should protect the communication channels across the edge computing system. The IICF defines the required communication security functions to:

- authenticate endpoints before allowing them to participate in a data exchange,
- grant read and write permissions to endpoints participating in a data exchange,
- ensure data integrity and trustworthiness of the data delivery, so that received data is not tampered with while stored or in transit and
- encrypt sensitive data flows (selectively, because some high-volume data flows may not be sensitive enough to warrant the overhead of encrypting and decrypting the data).

The decision to encrypt should be based on a risk-impact assessment. The access-control-model should be sufficiently fine-grained to limit the permissions of each endpoint narrowly to the operations and services needed for performing their intended functions. This enables the principle of least privilege that limits the consequence of security breaches and insider attacks.

The *security incident monitoring and response* service continuously monitors the edge computing nodes to detect functional and operational anomalies and initiate proper responses. Unlike similar services for conventional computer systems, security incident monitoring and response

[13] Rec, ITU-T. "X. 800 Security Architecture for Open Systems Interconnection for CCITT Applications." *ITU-T (CCITT) Recommendation* (1991).

operation in an edge computing system rarely requires human intervention. They rely mostly on automatic anomaly detection, machine-to-machine communication and autonomous responses.

4.2 System Security Management

Communication and information security must be managed at the system level and enforced over individual services running on the edge computing nodes. Successful security management over such a distributed, multi-layer infrastructure must have the following support mechanisms:

Identity management: A federated identity management system is needed to track and manage nodes and services. It must be able to create and manage unique identifiers and verifiable security credentials for every entity in the system, including IoT devices, edge computing nodes and application services running on each node.

Credential and relation management: The credentials and capabilities of every entity above must be issued by trusted authorities and verifiable by all the interacting entities. Transient and permanent relations among them should be auditable and traceable with high availability. Two technologies are commonly used to support these functions:

- Public key infrastructures (PKI): Public key certificates issued by trusted certificate authorities remain the most popular form of security credentials and public key signature is still the standard mechanism for offering non-repudiation protection.
- Distributed ledgers (DL) serve as a trusted public repository and the smart contracts they maintain can provide a reliable and scalable mechanism to track relationships established among a vast number of entities.

Policy management specifies, decides and enforces communication and information security actions based on verifiable security policies from multiple stakeholders. The eXtensible Access Control Markup Language (XACML) 3.0 architecture is a commonly used architecture and it comprises the following functional components:

- Policy administration point (PAP): an element that manages the security policies;
- Policy retrieval point (PRP): an element in which the XACML policies are stored;
- Policy information point (PIP): An element that acts as a source of attribute values;
- Policy decision point (PDP): An element that evaluates the security service requests against the applicable security policies before issuing decisions;
- Policy enforcement point (PEP): An element that intercepts a user's security service request, receives a decision from the policy decision point and acts on it.

The XACML policy management operation follows the procedures outlined below:

- a user sends a request, which is intercepted by the policy enforcement point;
- the policy enforcement point converts the request into a XACML authorization request;
- the policy enforcement point forwards the authorization request to the policy decision point;

- the policy decision point evaluates the authorization request against the applicable policies maintained by the policy retrieval point and then decides whether to permit or deny the request;
- the policy decision point then returns the decision to the policy enforcement point, which then carries out the decision.

XACML began as a mechanism to manage access control policies and expanded to manage and enforce most communication and information security policies.

4.3 INTERFACES

The interfaces between the end-to-end security service module and other modules are:

Communication security interfaces are protocols that provide communication-security services to information exchanges among all the entities in an edge computing system. These exchanges can go through three kinds of secure communication pathways:

- node-to-data-center (northbound) secure communication pathways,
- node-to-node (east/westbound) secure communication pathways and
- node-to-device (southbound) secure communication pathways.

Specifically, the southbound pathways support communications between devices and edge nodes using APIs based on industrial protocols, such as Modbus, OPC-UA, MQTT, DDS and REST. Section 10.1.3 of the IEEE standard for adoption of OpenFog reference architecture for fog computing has a detailed specification of these secure communication pathways and the standard protocols being used. *Information security interfaces:* Information security can be implemented by enforcing both mandatory and discretionary access control policies over information exchanges across a multi-layer edge system. As an example, Trusted Network Connect (TCN) developed by the Trusted Computing Group serves as a network information security standard for enforcing end-to-end access control in multi-vendor environments. StrongSwan offered an open-source TCN implementation based on its IP Security (IPsec) protocol stack.

Security service interfaces connect user's application services with the software infrastructure and the end-to-end security modules in an edge computing node. It provides a common set of APIs for the application services to access cryptographic and security services.

The IETF Generic Security Service Application Program Interfaces (GSS-API), (version 2, update 1 RFC 2743, RFC 2744) were the first standard sets of vendor-independent security APIs. They used the exchange of opaque messages, (*"tokens")*, to hide the implementation detail of security services from higher-level application services. Kitten is the successor of GSS-API and the Kerberos authentication system. It integrated the Simple Authentication and Security Layer architecture into its specification.

Security management interfaces connect the end-to-end security service module of an edge computing node to one or more system security management module(s) in the edge system that provides identity, credential and security policy management. Currently, there exist no common standard(s) for the security management interfaces although the XACML 3.0 architecture has been widely accepted for policy management. The pod security standards currently being developed in the Kubernetes community to specify the policies for running containers may become a standard way for managing container security on a Kubernetes platform.

5 SYSTEM MANAGEMENT AND PROVISIONING MODULES

An edge computing system may comprise a great number of widely dispersed, interconnected nodes each running multiple application services that interact to deliver system functionality. Each node needs to be discovered, commissioned, operated and decommissioned over the lifetime of the system. In addition, the application services running on these nodes need to be deployed, configured, monitored, updated, scaled, and deleted.

Edge computing system management in this context is a large and complex set of capabilities. Core to the concept of edge computing is the use of *elastic computation* across nodes, layered below the data center. To respond to varying workloads, an edge computing system can add or remove resources quickly to execute on more or fewer edge computing nodes, or repartition workloads to optimize efficiency. Edge computing system management needs to orchestrate elastic computing resources to achieve this. Elastic computing resources such as VMs and containers can be clustered as a service. A service distributes workloads requested by clients across clustered computing resources. Then, elastic computing resources for a service can be automatically scaled depending on workloads. Managing and orchestrating this elastic infrastructure is critical to the functionality of a distributed edge system.

Management capabilities are split into levels. At the node level, management is responsible for the configuration and the update and monitoring of all the low-level hardware and software resources in an edge computing node. The application and service management functions are at a higher level, responsible for deployment, configuration, update and monitoring of application services.

The system provisioning and management functions work at the system level as a centralized controlling and monitoring capability. In practice, the capabilities below the dotted line on Figure 5-1 run primarily on individual edge computing nodes, while those above are often run in or near a data center.

An edge computing system contains myriad resources that need to be managed. This is described by a *resource model* that can be constructed automatically through discovery within the system management.

Specific functions of edge system management are to:

- collect and maintain the resource model,
- manage connected physical resources (e.g., sensors and actuators), computing resources (e.g., VMs and containers), storage and networking and possibly data center resources,
- build and maintain hierarchy and containment relationships of all resources (e.g., the relationships between a VM and a container running on the VM) and
- maintain a topology of the connectivity including physical and logical networks and all associated interfaces.

5.1 COMPONENTS

System management components for distributed computing in the edge comprise a centralized management module and management modules residing in edge computing nodes.

System provisioning and management is a set of system services at the system level. It interacts with the modules for node and application management running in an edge computing node to support centralized system management and orchestration of application services running in edge computing nodes. This is typically implemented in data centers and provides an API for user interface tools.

The *application & service management module* manages the lifecycle of an application service in a virtual machine or a container. After registering with the system provisioning and management module, an edge computing node can receive control messages from the management system for deploying, updating and deleting application services. It also reports status of running application services back to the management system.

The *node management module* handles configuring, monitoring, and updating the system hardware and software resources in an edge computing node. This module needs to be registered in the system provisioning and management module before it can be used. After registration, the management system can operate the node management functions remotely.

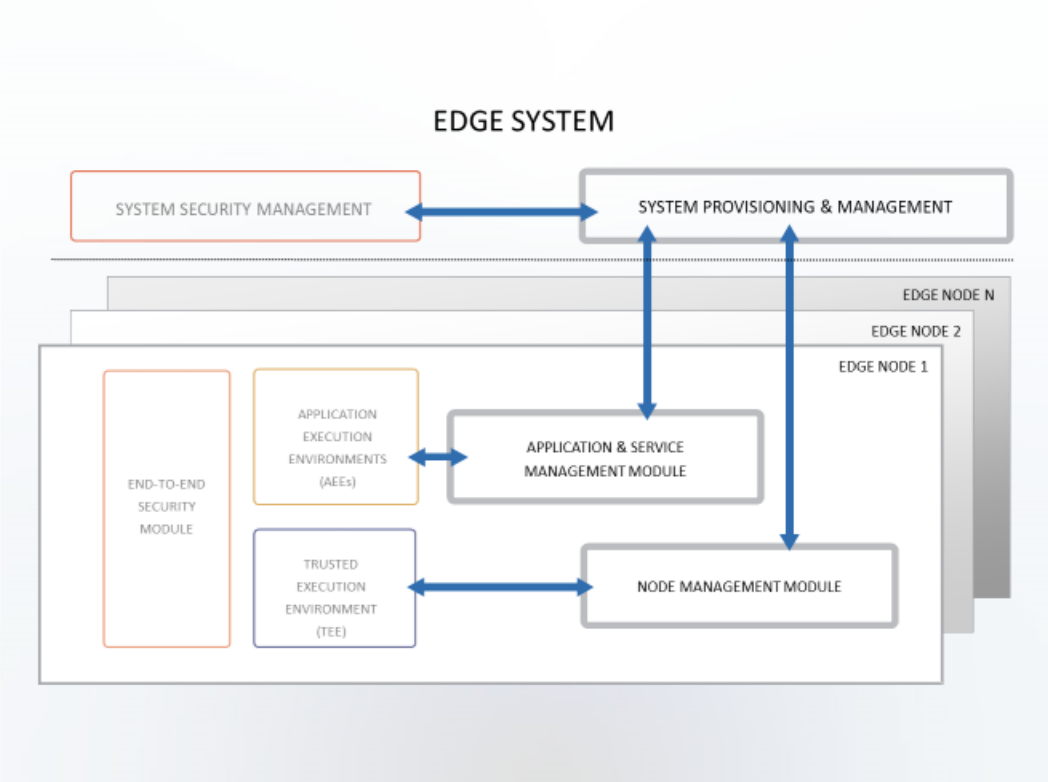

Figure 5-1: Functional Components of Edge System Management and Provisioning Modules

5.2 INTERFACES

The blue arrows on Figure 5-1 represent interfaces associated with system management.

5.2.1 NODE MANAGEMENT MODULE INTERFACE

The interface between the *node management module* and the *system provisioning and management* configures, monitors and updates system hardware and software resources in an edge computing node from the centralized management system. Resources managed by this interface include:

- system information (e.g., instruction set, operating system),
- device identification,
- status of a node,
- network addresses for edge computing nodes and
- resource capacity of a node (e.g., CPU and memory).

5.2.2 APPLICATION SERVICE MODULE INTERFACE

The interface between the *application service management module* and *the system provisioning and management* is used for remote interactions between the management system and edge computing nodes to provide orchestration functions for application services. They include automatic deployment and update, load balancing, scaling and self-healing.

The deployment and update function ensures that requested application services are deployed and running in edge computing nodes. These application services can perform tasks varying from simple data collection to real-time analytics or machine-learning inference. A load balancing capability distribute tasks from clients over a set of containers or virtual machines. A logical service can then be dynamically scaled up and down by adding or removing containers or virtual machines based on overall loads.

The self-healing capability continuously monitors the health of nodes and application services. For example, if a user requests three replicas of containerized or virtualized applications and an edge computing node running one of the applications fails, the self-healing capability will redeploy a replica of the application into a healthy node to attain the desired state.

5.2.3 INTERFACES BETWEEN APPLICATION SERVICE MODULE AND APPLICATION EXECUTION ENVIRONMENTS

An edge computing node runs a container runtime or a hypervisor that handles the lifecycle operation of containers or virtual machines. The application service management module uses this interface to request a lifecycle operation to a container runtime or a hypervisor. The operations supported by this interface include pulling images, creating, deleting, starting and stopping containers or virtual machines. For example, Kubernetes has a plugin interface for container runtimes named the container runtime interface that defines the common operations for containers and it allows the system to use a variety of container runtimes.

5.2.4 Interfaces between Node Management and Trusted Computing Modules

Two sets of interfaces exist between node management and trusted computing modules:

The *Trusted Execution Environment (TEE) Management Interface* is dominated by the GlobalPlatform TEE Management Framework (TMF). The TMF defines standard methods to manage the lifecycle of a TEE and the trusted application services running in the TEE via protocols and interfaces accessed through either the GlobalPlatform TEE Client API or extensions to the TEE Internal Core API. The framework is divided into three layers:

- *Administration operations* for managing the trusted application services, the security domains and their conditions of use.
- *Security Models* for
    - specifying the responsibilities and relations among the TEE implementer, the TEE issuer and the trusted application providers,
    - specifying the security mechanisms to authenticate various entities, secure communications and authorize operations and
    - specifying key management and data provisioning schemes.
- *Open Trust Protocol (OTrP)* for specifying the command set and the JSON encoding for performing the administration operations.

Trusted Platform Modules (TPM) Management Interface for managing the operation of cryptographic unit embedded in a trusted hardware platform. The following two interfaces are of particular importance:

- *TPM Key Management Standard*, which specifies TPM key hierarchy, and key migration models, structures and flows and
- *Remote Attestation Procedures,* which is being developed in the IETF Remote Attestation Procedures (rats) working group.

5.3 Contract of Interoperability

To on-board end devices, gateways and servers securely into a management system requires a handshake. Zero-configuration, auto-discovery and self-describing interfaces provide this handshake automatically and minimize the need for manual labor, which is especially costly in the case of large networks of heterogeneous IIoT devices. These interfaces define a protocol for the handshake between the device, as shipped by the vendor, and the management system. Devices, upon power up or reset should authenticate, auto-configure, become part of the network, be auto-discovered and securely on-boarded by the management system with little to no manual intervention. They should declare their properties, actions and normal versus off-normal vital signs (that is, be “self-describing”). This guides the management system towards the proper monitoring of the device without prior knowledge of its details being hard coded or manually configured.

Likewise, IIoT software components should provide a handshake so they can be monitored and managed. EdgeX Foundry microservices are equipped with management APIs for this purpose.

Self-describing managed resources should be applied uniformly across both hardware and software components, across all three tiers of the system. The provider of the component develops and packages resource objects with the product as it is shipped. The self-description is a schema based on a service or interface description language, or the equivalent. Web Services Description Language (WSDL) is one example; OpenAPI Specification, based on the Swagger specification using a REST APIs approach, is another. The description indicates parameters that can be configured, and the vital signs to monitor including what constitutes normal versus off-normal behavior. Standards groups will define APIs for the different types of managed objects to further streamline this handshake and contract of interoperability for IIoT infrastructure management.

5.4 Edge Computing Node Lifecycle

Each edge computing node in this architecture has its own management lifecycle, which is controlled by its node management module in conjunction with the system provisioning and management module of the edge system. Together, they ensure that each edge computing node goes through its lifecycle stages successfully, as outlined in Figure 5-2, with minimal human intervention.

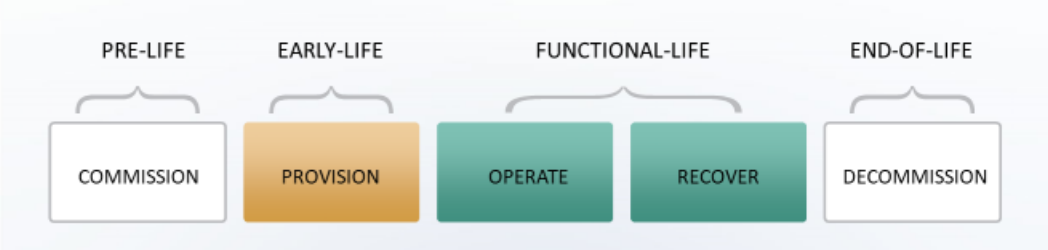

Figure 5-2: Management Lifecycle of an Edge Computing Node[14]

Commission is the earliest phase of the lifecycle. It includes installation, identification, certification and calibration. The managed entity must:

- lay the groundwork for trustworthiness, with support for security, privacy, safety, reliability and resilience that can be attested to and trusted in all phases of the lifecycle,
- be agile in their ability to collect data and monitor systems in the face of changing requirements and operational conditions and
- open, to allow control and provide visibility into its resources.

Provision includes discovery, advertisement of features and capabilities, trust, updating software and data structures, and deployment of features. This starts with secure on-boarding of an edge

[14] Extracted from Fog Computing Reference Architecture, Figure 17

device and making it accessible under its user interface. This may include the detection of sensors and connections to servers.

If deployment-specific credentials must be configured during device manufacturing then each such configuration is, in effect, a unique manufacturing SKU. These SKUs must be managed as unique products through IoT supply chains from manufacturing, distribution, to systems integration and installation. This uniqueness helps prevent the successful provisioning of counterfeit or spoofed devices.

Operate covers the period when an edge computing node is in normal operation. After connectivity from sensor-to-server through an edge device, the application management module brings up the application services via a container runtime or a hypervisor. It monitors the devices involved and monitors the virtualization stack and the application services components on top.

After the system is up and running and the application is acquiring and processing data, system management evolves to control and monitor the system to ensure its steady state functioning. This includes:

- querying or receiving the state of the different aspects of the system and displaying it,
- data ingestion of time-series metrics for vital sign monitoring and the detection of abnormal or overload conditions,
- alerts, to include predicting, detecting and annunciating failures such as overloads and
- actuation to resolve or avoid failure by lowering the rate of time-series acquisition in response to an overload alert or adjusting other operational parameters to maintain the required system performance under load.

System management performs updates when a new version is available. When an update operation is requested, an edge computing node initially pulls updated container or VM images from a remote repository and then replaces old containers or VMs with new ones. Similar mechanisms can update firmware, BIOS and FPGA configuration files. Rolling and rollback updates are desirable capabilities for updating application services on edge computing nodes. Rolling updates allow application services to be updated with zero downtime by incrementally updating application service instances with new ones. If the state of an updated application service is not stable, it can be rolled back to a previous version.

Recover includes the period when an edge computing node is operating out of expected norms. It should attempt to isolate hardware and software components from failed nodes and self-heal autonomously. Other edge computing nodes, or data center resources may also assist with the recovery action, by performing various resets, restarts and reloads, moving computational loads from failed or overloaded nodes to redundant nodes or redirecting critical data streams away from failed storage devices or communications links.

Decommission covers a time when the node cleanses all personally identifiable and proprietary data it may have stored from the hardware and prepares it for reuse another deployment. In

non-stop systems where short duration service outages can't be tolerated, decommissioning one node involves the seamless migration of its workloads and data to a replacement node. It includes ways to wipe out all non-volatile storage so that future application services cannot access the previous tenant's data. De-commissioning physical nodes may involve recycling and site restoration.

The system management and provisioning functions are essential to the operation of distributed computing in the edge. With tens of millions of edge computing nodes deployed in the coming years, without highly capable management, the deployment speed, labor investment, cost, efficiency and customer satisfaction of edge computing will be unacceptable.

6 CONCLUSIONS

This distributed computing framework provides insights into the capabilities and architectures of distributed computing, edge computing nodes and edge computing systems. Through the careful application of the concepts and techniques described here, it is possible to meet the critical performance, trustworthiness and efficiency requirements of many classes of IoT applications serving many vertical markets.

The framework pays special attention to two of the most essential properties of distributed computing and edge systems: end-to-end security and system management. The security architecture shows how to start with a trusted computing module and build edge computing node security up to a complete trustworthy system. The system management capabilities described illustrate how to manage edge computing nodes and application services, supporting capabilities like orchestration, resource management and the operation of distributed computing workloads throughout their lifecycle.

There is much architectural work still needed to address all the challenges in distributed computing in the edge. Beyond the security focus of this document, edge has implications for the four other dimensions of trustworthiness defined by IIC (privacy, safety, reliability and resilience). Performance and how to measure and optimize it is an ongoing challenge. Network complexity, especially related to management is a strong contributor to total lifecycle cost of ownership of distributed computing in the edge solutions, and much work is required there.

Distributed computing, and the nodes and edge systems that are its key components are essential to the success of many critical IoT systems, and digital transformation plans. This framework is a useful tool in planning its architecture, implementation, deployment, and widespread adoption.

AUTHORS AND LEGAL NOTICE

This document is a joint work product of the Industrial Internet Consortium's Fog Computing Task Group and Distributed Computing Task Group co-chaired by Chuck Byers (IIC), John Zao (NCTU) and Brett Murphy (RTI).

This work was started in the OpenFog Consortium, which combined with the Industrial Internet Consortium in early 2019. We would like to acknowledge the OpenFog Consortium, its members, staff and leadership for their contribution of many of the foundational concepts in this document.

Authors: John Zao (NCTU), Chuck Byers (IIC), Brett Murphy (RTI), Salim AbiEzzi (VMware) Don Banks (ARM), Kyoungho An (RTI), Frank Michaud (Cisco Systems) and Katalin Bartfai-Walcott (Intel).

Contributors: The following individuals have provided valuable comments and inputs that have substantially improved the quality of this whitepaper: Helder Antunes (Cisco), Jeff Fedders (Intel), Ron Zahavi (Microsoft), Lynne Canavan (OpenFog Consortium), Evan Birkhead (IIC), Craig Griffin (Wind River), Arjmand Samuel (Microsoft, Clemens Vasters (Microsoft), Brian Raymor (Microsoft), Maria Gorlatova (Princeton University), Jingyi Zhou (ZTE), Francois Ozog (Linaro), Tao Zhang (Cisco), Mitch Tseng (Huawei), Todd Edmunds (Dell), Lalit Canaran (Ivado), K. Eric Harper (ABB) and Stan Schneider (RTI).

Technical Editor: Stephen Mellor (IIC) oversaw the process of organizing the contributions of the above Authors and Contributors into an integrated document.

In Memoriam: Brett Murphy

We lost a beloved colleague recently. I wanted to take this time and space to express our gratitude for having the great fortune to have known Brett Murphy in his too-short life and share how important he has been to us.

Brett Murphy has been a member of the Industrial Internet Consortium since Real-Time Innovations (RTI) joined within days of our launch in 2014. But not only was Brett a member, he was a trusted colleague, friend, key contributor, voice of reason and shining light to us all.

Brett epitomized the true meaning of the word collaboration. In the many opportunities we had to work with him and to see him in action at our meetings and events, he was a true team player. He was always surrounded by members who were learning from each other and at the same time developing new ideas that would raise everyone's game. He was a spokesperson for those informal groups and a leader of our formal groups. He brought his scientific and practical mind to the strategic discussions that have shaped this consortium since its inception. We would not have achieved as much as we have thus far, without Brett.

Brett was a Stanford alum, an engineer and a marketer. A technical expert and a visionary. He was passionate and kind. He possessed a calming presence that was complemented by a quick sense of humor and a mischievous streak that merged in the most wonderful, likable way. He could provide the spark for a brainstorm or defuse the tension in the room. It's why we found him a joy to work with and were honored to call him our friend.

In consortia culture, where contributors offer their precious free time and intellect on top of their day jobs, Brett was always willing to help. He applied his rock-solid reliability to his own brilliant ideas, as well as when called upon to lead a new initiative, so others could learn and follow.

We know the dedication he shared with us was eclipsed by his dedication to his family. We share Brett's family's and the RTI family's grief and thank them for sharing Brett with us for these past 6 years. His contributions, and more importantly, his presence in our lives will never be forgotten.

Brett was a key contributor to this report. It is fitting that we memorialize him here.

Kathy Walsh, *VP of Marketing, IIC*