DEADLINE FOR COMMENTS: EOD, THURSDAY APRIL 30

Contents

1 Introduction

“Edge computing” represents a new paradigm in which compute and storage are located at the edge of the network, as close as both necessary and feasible to the location where data is generated and actions are taken in the physical world. The optimal location of these compute resources is determined by the inherent tradeoffs between the benefits of centralization and decentralization.

This white paper introduces the key concepts of edge computing and highlights emerging use cases in telecom, industrial, enterprise and consumer markets.

The paper also provides details of eight open source edge projects hosted by the Linux Foundation (LF) and its subsidiary LF Edge (LFE) umbrella project. The LF is a non-profit technology consortium founded in 2000 to standardize Linux, support its growth and promote its commercial adoption. The LF and its projects have more than 1,500 corporate members from over 40 countries. The LF also benefits from over 30,000 individual contributors supporting more than 200 open source projects.

Founded in 2019, the mission of LF Edge is to establish an open, interoperable framework for edge computing independent of hardware, silicon, cloud or operating system.

1.1 Introducing the Edge Continuum

As defined in the Linux Foundation’s Open Glossary of Edge Computing, edge computing is the delivery of computing capabilities to the logical extremes of a network in order to improve the performance, operating cost and reliability of applications and services. By shortening the distance between devices and the cloud resources that serve them, and also reducing network hops, edge computing mitigates the latency and bandwidth constraints of today’s Internet, ushering in new classes of applications. In practical terms, this means distributing new resources and software stacks along the path between today’s centralized data centers and the increasingly large number of deployed nodes in the field, on both the service provider and user sides of the last mile network.

In essence, edge computing is distributed cloud computing, comprising multiple application components interconnected by a network. Most applications we use are distributed, such as a smartphone or thermostat with a cloud backend, a smartwatch or sensor connected to a smartphone and then to the cloud, an industrial system connected to an IoT edge gateway and then to an on-prem system and/or the cloud, and so forth. We think of the “edges” as the physical infrastructure or devices where these application elements run.

The diagram below provides an overview of the edge computing continuum. While cloud providers are locating their services at key points on the internet (referred to as the “cloud edge” or “internet edge”) to minimize latency, applications are increasingly requiring compute resources to be distributed further and further down the edge continuum for reasons of latency, bandwidth, security, privacy and autonomy. This paper focuses on two main edge categories - the “**Service Provider Edge**” and the “**User Edge**”, with the latter being further broken down into subcategories due to inherently increasing design tradeoffs the closer devices get to the physical world.

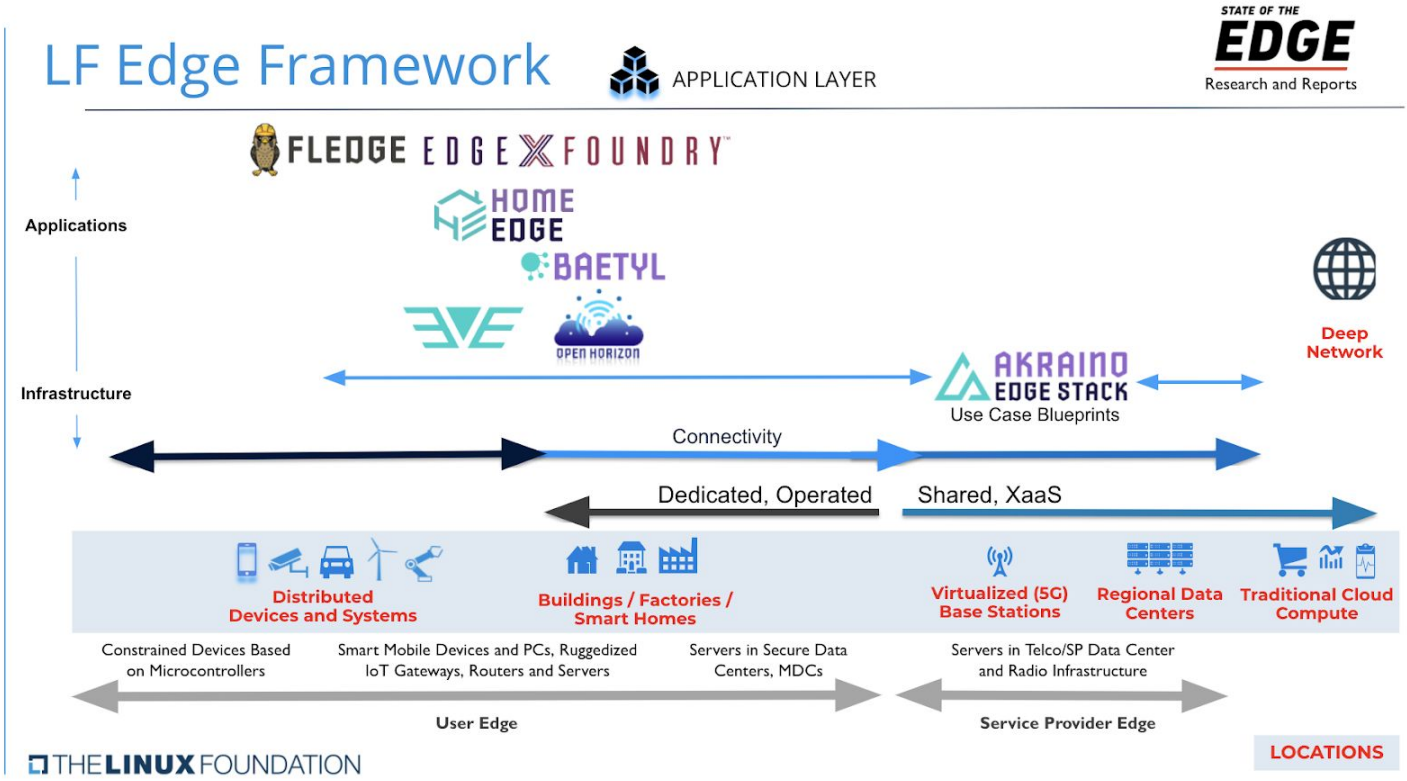


Figure 1. Edge Computing Continuum Addressed by LF Edge

At the extremes of the edge continuum are devices and the cloud. The cloud enables us to do many things that are not possible or appropriate on a device. Cloud resources are practically unlimited, whereas device resources vary, however as long as the interconnection is adequate the collective behavior of a large number of devices can be configured, tracked and managed. The limitations of the public cloud are the location of the data centers and the fact that the resources are shared.

Moving down the continuum from the cloud, the first main edge tier is the **Service Provider Edge**, providing services delivered over the global telecom fixed/mobile infrastructure. Like the public cloud, infrastructure (compute, storage and networking) at the Service Provider (SP) Edge is typically consumed as a service. Cellular-based solutions at the Service Provider Edge tend to be more secure and private than the public cloud, because of differences between the Internet and cellular systems. It leverages the existing trillion-dollar investments by Communications Service Providers (CSPs), who cumulatively have hundreds of thousands of servers in place at Points of Presence (PoPs) at the network edge. Infrastructure at this edge tier is generally more standardized than compute at the User Edge but there are still unique requirements for regulatory and ruggedization depending on where it is deployed.

The second top-level edge tier is the **User Edge** which is delineated from the Service Provider Edge by being on the other side of the last mile network. Sometimes it is a necessity to use on-premise and highly distributed compute resources to be closer to end-users and processes in the physical world in order to further reduce latency or the bandwidth cost of broadband networks. Additional reasons for localized compute at the User Edge include autonomy, increased security and lower overall cost - if the available resources match the need of the application workload. Compared to the Service Provider Edge, the User Edge represents a more diverse mix of resources, and as a general rule the closer edge compute resources get to the physical world, the more constrained and specialized they become.

As highlighted in Figure 1, a key difference between the edge tiers is who owns the computing assets. While resources at the Service Provider Edge and within the public cloud are owned by these entities and shared across many users,

resources at the User Edge are typically dedicated and customer-owned and operated (possibly lending more security and control as a result). This results in a business model based on CapEx rather than OpEx, with the technology acquisition and scaling being the responsibility of the user rather than delivered as a managed service. That said, there are a growing number of service providers building managed service offers that include on-premise networking infrastructure. An example here is a provider operating private cellular base stations for connectivity across a remote mining site.

Note that the edge computing taxonomy and associated tier names presented in this document were developed with careful consideration to balance various market lenses (e.g. telecom, IT, OT/industrial, consumer) and align with key technical and logistical tradeoffs without being overly complex. The intent is to provide a holistic point-of-view compared to edge terminology that may mean something in one entity but can be confusing to another. For example, the terms “near” and “far” edge are commonly used in the telco space to represent infrastructure closer to users/subscribers (far edge) versus further upstream (near edge). This can be confusing because relative location is viewed through the eyes of the service provider instead of the user. In another example, the terms “thin” and “thick” have been used in some circles to characterize degrees of on-prem edge compute capability, however these terms do not delineate between resources that are physically-secured in a data center versus distributed in an accessible location.

1.2 Extending Cloud Native Principles to the Edge

With the introduction of containerization and Kubernetes, a rapidly increasing number of organizations are moving to cloud-native software development based on platform-independent, microservice-based architecture and continuous delivery (CI/CD) practices. The same benefits of cloud-native development in the data center apply at the edge, enabling applications to be composed on the fly from best-in-class components, scaling up and out in a distributed fashion and evolving over time as developers continue to innovate.

As defined by the Open Glossary, an “edge-native application” is one which is impractical or undesirable to operate in a centralized data center. In a perfect world, developers would have a universal foundation that enables them to deploy containerized workloads anywhere along the edge-to-cloud continuum as needed in order to balance the benefits of distributed and centralized computing in areas such as cost, latency, security and scalability based on context. However, this isn’t universally possible due to inherent technical and logistical tradeoffs, including the need to accommodate legacy investments and protect safety-critical systems.

Users need a foundation that supports the extension of cloud-native principles as far down the edge continuum as feasible while comprehending these constraints. The following sections dive into detailed considerations at both the Service Provider and User Edges.

1.3 Considerations for the Service Provider Edge

Many web scale design principles can be applied to implement cloud-like compute capabilities at the Service Provider (SP) Edge. Over the last few years, orchestration technologies like Kubernetes have made it possible to run cloud-native workloads in on-premise, hybrid or multi-cloud environments. We can assume that applications offloaded to the Service Provider Edge will not change their design or code and will retain continuous delivery pipelines and deploy only certain workloads which have low latency, high bandwidth, strict privacy needs at service providers’ edge sites.

Major content owners like Netflix, Google and YouTube are expected to retain their cache-based distribution models, which entail storing states in the centralized public cloud along with AA (Authentication and Authorization) functions, while redirecting the delivery of content from the “best” cache as determined by Quality of Experience (QoE) at the client device, the best doesn’t always means the nearest cache. This approach will be retained for other distributed workloads utilizing edge acceleration like Augmented Reality (AR), Virtual Reality (VR), Massively Multiplayer Gaming (MMPG) etc.

According to these design principles, the Service Provider Edge will need to ensure a deterministic method of measuring and enforcing QoE based on key application needs such as latency and bandwidth. As most internet traffic is encrypted, these guarantees will be based on the transport layer, leading to the evolution of congestion control algorithms which determine the rate of delivery. A similar design principle will evolve for geographical data isolation policies for stores and workloads, beyond just complying with global data protection regulations. The figure below details the deployment of highly-available edge applications (EApps) within a telecom operator federating across other operators at peering sites and cooperating with public cloud workloads.

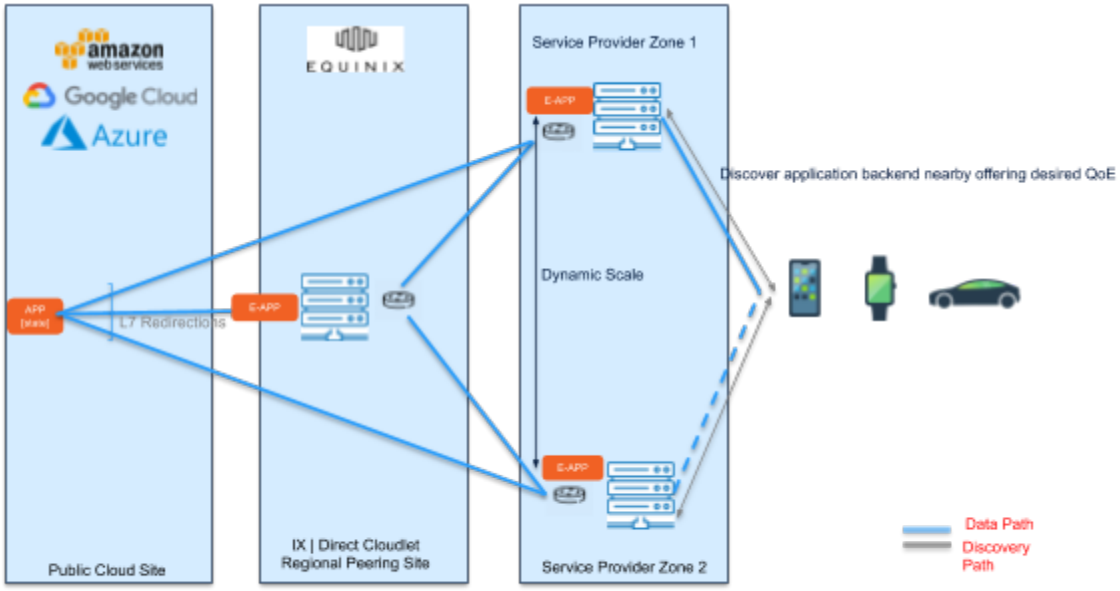


*Figure SEQ Figure * ARABIC 2: Edge Application Deployments.*

Figure 2. Title

EApps deployment on the service provider edge

Advanced developers are aware of the geographical consumption patterns of their customer base. Using Machine Learning (ML) algorithms, they can predict the geographic locations of their user base for advanced planning purposes. The deployment of application backends can be independent of network mobility or specific device attachment. Backend application deployment is based on two choices:

- **Static,** whereby the developer chooses the specific edge sites and the application flavors for each site.
- **Dynamic,** whereby developers choose a region in which they yield control to the service providers orchestration system in order to determine the optimum placement of workloads, based on the number of compute instances, the number of users and specialized resource policies. For stateless applications, serverless policies can be abstracted at a higher level if layer 4 QoE is desired.

The above structure defines the boundaries of EApps mobility.

The lifecycle management of edge applications is based on the following workflow for deployment:

1. Create the cluster, deploying microservices as a set of containers or Virtual Machines (VMs);
2. Create the application manifest, defining an application mobility strategy that includes QoE, geographical store and privacy policies;
3. Create the application instance, launching the EApp and autoscaling.

For more information on this topic, please visit the developer section for the Akraino Edge Stack project.

Design strategy for EApps mobility across Service Provider Edge

Application mobility is based on resource awareness, a backend for stateless applications that can move across SPs distributed fabric based on compute capacity, specialized resources and/or Service Level Agreement (SLA) boundaries. Stateful EApps synchronize states from centralized servers to the edge and redirect them at Layer 7 to edge applications, operating consistently regardless of an individual SP's orchestration system. SPs platform may offer periodic QoE hints to centralized servers to assist with the above redirection process.

Design strategy for users device mobility across Service Provider Edge

Since device mobility is based on route awareness, it's important to review how data moves across mobile networks before explaining the design principles of device mobility.

A mobile device connected to wireless networks attaches to the nearest tower, then tunnels all applications data to the nearest gateway which is further tunneled[1] to the regional gateway, which is then transferred over the Internet exchange to a public cloud and back. Regional gateways called packet gateways (PGWs) can be viewed as anchors, which CSPs utilize for enforcing centralized subscriber control like policies, billing and management. Routing data as described above, however, is sub-optimal and cannot enforce the latency, bandwidth and privacy guarantees which were the initial goals. Typically, an EApp doesn't need to move because a single device consuming it moves to another PGW, labelled generically as anchors henceforth in the document.

An easier solution is provided by local breakout, which implies distributing the anchors near to the location of devices, at edge sites. Control and User Plane Separation (CUPS) for these packet gateways is a key step, deploying lightweight cost-effective distributed user plane functions (UPFs) at each edge site. Obtaining the GPS location for the UPF, if exposed from a centralized control plane, assists in identifying the nearest application backend. A better approach, however, is for devices to attach to a geographically co-located anchor based on the physical location of the device, in which case local breakout works seamlessly with the edge cloud orchestration scheme behind these anchors.

Local breakout and anchors redistribution is happening today with recent trends in 5G CUPS, disaggregation of hardware from software and has propelled network appliance vendors to offer the software as virtual or container network functions (VNF/CNF). Service providers may use the common orchestration plane like Kuberenetes described above for CNF lifecycle management and build a continuous delivery pipeline. Microsoft recently acquired Affirmed Networks, a company that provides fully-virtualized cloud-native solutions for anchors. The Life cycle management techniques can be extended not only to anchors but far edge,virtualized radio heads.

The control plane separation shall also allow SDN-like programming of tunnels to redirect traffic from devices to these distributed endpoints. These tunnels carry all users application traffic and are labelled as packet data user (PDU) sessions. Standards defining organizations like 3GPP are working on redirecting only EApps application flows within PDU session to nearest anchors. The PDU sessions with embedded tunnel IDs as transport state presents state synchronization issues thus existing session continuity procedures in 3GPP where it is expected that device shall maintain PDU sessions across thousands of distributed anchors is non viable. Fortunately, the anchored routing structure can be changed by leveraging container mobility techniques used by web scale companies, but that requires not just virtualizing the compute (VNF/CNF) but also virtualizing the networks such that underlying IP routing is based on the identity of application and location of device. Identifier Locator Addressing is a means to implement network overlays without the use of encapsulation can help achieve anchorless device mobility.

Assuming above design strategies are in place, how do we discover the best edge location to serve a user?

[1] Tunnel are GTP-U encapsulations over IP, For more Info: https://en.wikipedia.org/wiki/GPRS_Tunnelling_Protocol

The nearest edge location is not always the best. Instead, clients must be steered to application backends based on the most recently recorded QoE for the application at each geographically-located edge site.

To clarify this approach, it's important to note the design principle mentioned earlier: "…. the telco edge will need to ensure a deterministic method of measuring and enforcing QoE based on key application needs such as latency and bandwidth…. .. leading to the evolution of congestion control algorithms which determine the rate of delivery."

Based on this design, an application discovery engine should be embedded across multiple CSPs which records the health of the application backend and the QoE for each application, across all edge sites within a region, exposing a control API to identify the best location and tune the EApps rate of content delivery for the best experience. The control API from a discovery engine would return the ranked list of Uniform Resource Identifiers (URIs), identifying the optimum EApps within sites nearby, based on selection criteria that include:

- EApps instances in sites geo-located based on the client's location;
- URI rank based on recent Layer 4 QoE measurements (latency and bitrate).

For more information on the Application Discovery engine please refer to LF Edge Akraino APIs white paper and for definition of a control API please visit the Find Cloudlet section for the Akraino Edge Stack project.

Later in the document there is a comprehensive list of use cases and workload attributes which individual service providers can use to serve enterprise and privacy centric edge use cases today. But for CSP edge to come alive with the next generation emerging persuasive and immersive billion dollar EApps, operators have to come together and offer their edge cloud resources with the smart federation scheme at the last mile. Traditionally CSPs have federated to provide us global coverage, where sometimes they adopted the sub optimal approach of rerouting traffic to anchors in the home network. A more efficient strategy is for CSPs to federate directly via a peering exchange as described above, or even across last mile Radio Access Networks.

1.4 Considerations for the User Edge

As highlighted previously, the User Edge consists of a diverse mix of compute form factors and capabilities that get increasingly unique as deployments get closer and closer to the physical world. The technical tradeoffs spanning the User Edge include not only varying degrees of compute capability (especially available system memory) but also the need for specific I/O for legacy and modern data sources, various degrees of ruggedization including extreme temperature support with fanless design for reliability, specialized certifications (e.g. Class 1 / Division 2 for explosion proof), highly specific form factors, unique needs for Management and Orchestration (M&O) and security, and so forth. In terms of logistical considerations, while consumer-oriented computing devices tend to have a typical lifespan of 12-18 months, enterprise and industrial edge computing assets distributed in the field need to support a long service life of 5-7+ years. Given all of these inherent tradeoffs, it is helpful to break the User Edge down further into several subcategories.

At the upper end of the User Edge tier is the **On Prem Data Center Edge** which is based on server-class infrastructure located within traditional, physically-secure data centers and Modular Data Centers (MDCs), both inside and proximal to buildings like offices and factories. These resources tend to be owned and operated by a given enterprise and are moderately scalable - within the confines of available real estate, power and cooling. Tools for security and M&O are similar to those used in the cloud data center, however there is some evolution required to support coordination of Kubernetes clusters distributed across these locations.

In the middle of the User Edge is the **Smart Device Edge** subcategory which consists of compute hardware located outside of physically-secure data centers but still capable of supporting virtualization and/or containerization technologies for abstraction and accommodating cloud-native software development. These resources span consumer-grade mobile devices and PCs to hardened gateways and servers that are deployed for IoT use cases in challenging environments such as factory floors, building equipment rooms, farms and weatherproof enclosures distributed within a city. While

capable of general-purpose compute, these devices are performance-constrained for various reasons including cost, battery life, form factor and ruggedization (both thermal and physical) and therefore have a practical limit to processing expandability when compared to resources in a data center. There is an increasing trend for these systems to feature co-processing in the form of GPUs or FPGAs to accelerate analytics, with the added benefit of distributing thermal dissipation which is beneficial in extreme environments. Resources at the Smart Device Edge can be deployed and used standalone (e.g. smartphone, IoT gateway on a factory floor) or embedded into distributed, self-contained systems such as connected/autonomous vehicles, kiosks, oil wells and wind turbines.

At the lowest extreme of the User Edge tier is the **Constrained Device Edge** subcategory, represented by microcontroller-based devices that are highly distributed in the physical world. These devices range from simple, fixed-function sensors and actuators that perform very little to no localized compute to more capable devices such as Programmable-logic Controllers (PLCs), Remote Terminal Units (RTUs) and Engine Control Units (ECUs) addressing time- and safety-critical applications. Devices at this tier leverage embedded software and have the most unique form factors.

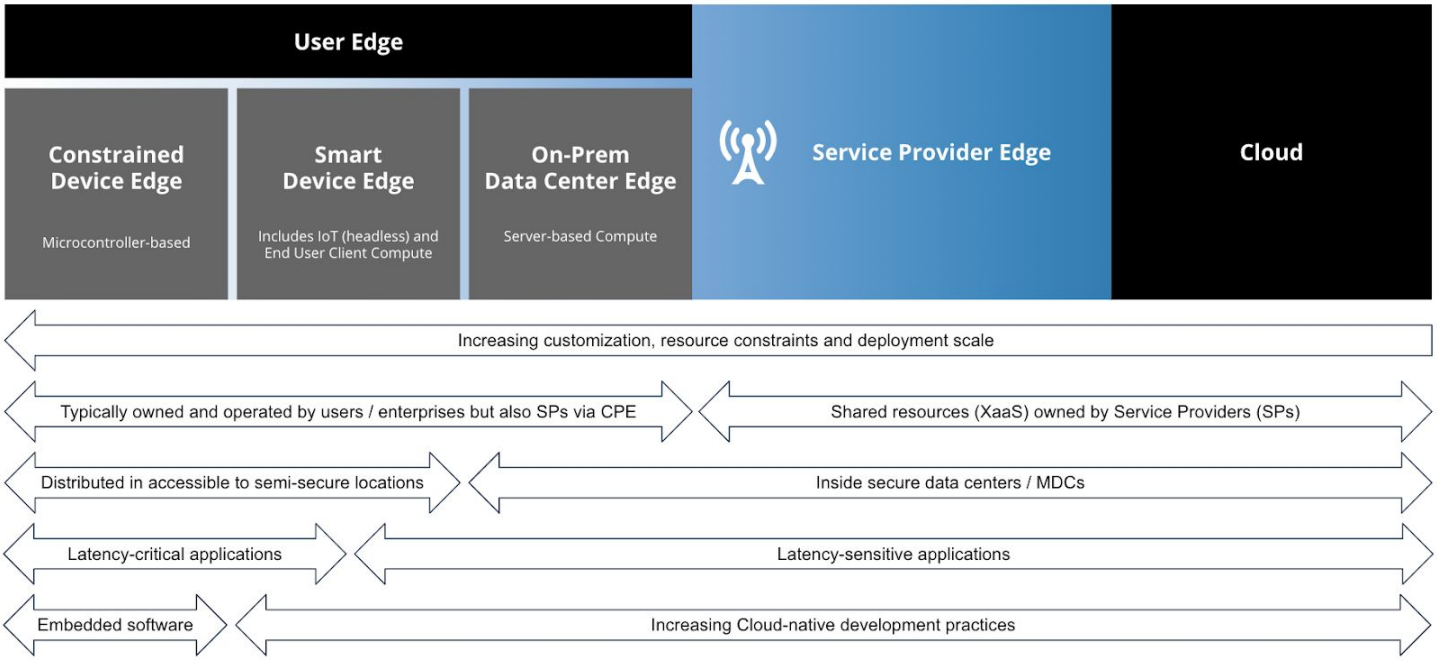


Figure 3. Summary of Edge categories and general trends

The sub-categories under the User Edge work with the Service Provider Edge and Cloud as part of a tiered compute continuum, but not necessary in series. Constrained and Smart Devices distributed in the physical world (such as smart thermostats, smartphones and connected vehicles) often communicate directly with the Service Provider Edge and Cloud, bypassing all On-Prem Data Center infrastructure. Devices can also be deployed on-premise and interact with more capable local edge compute, which in turn interacts with the Service Provider Edge and Cloud. The continuum is a complex matrix of locality, capability, form factor and ownership. The figure below illustrates examples of edge deployment patterns.

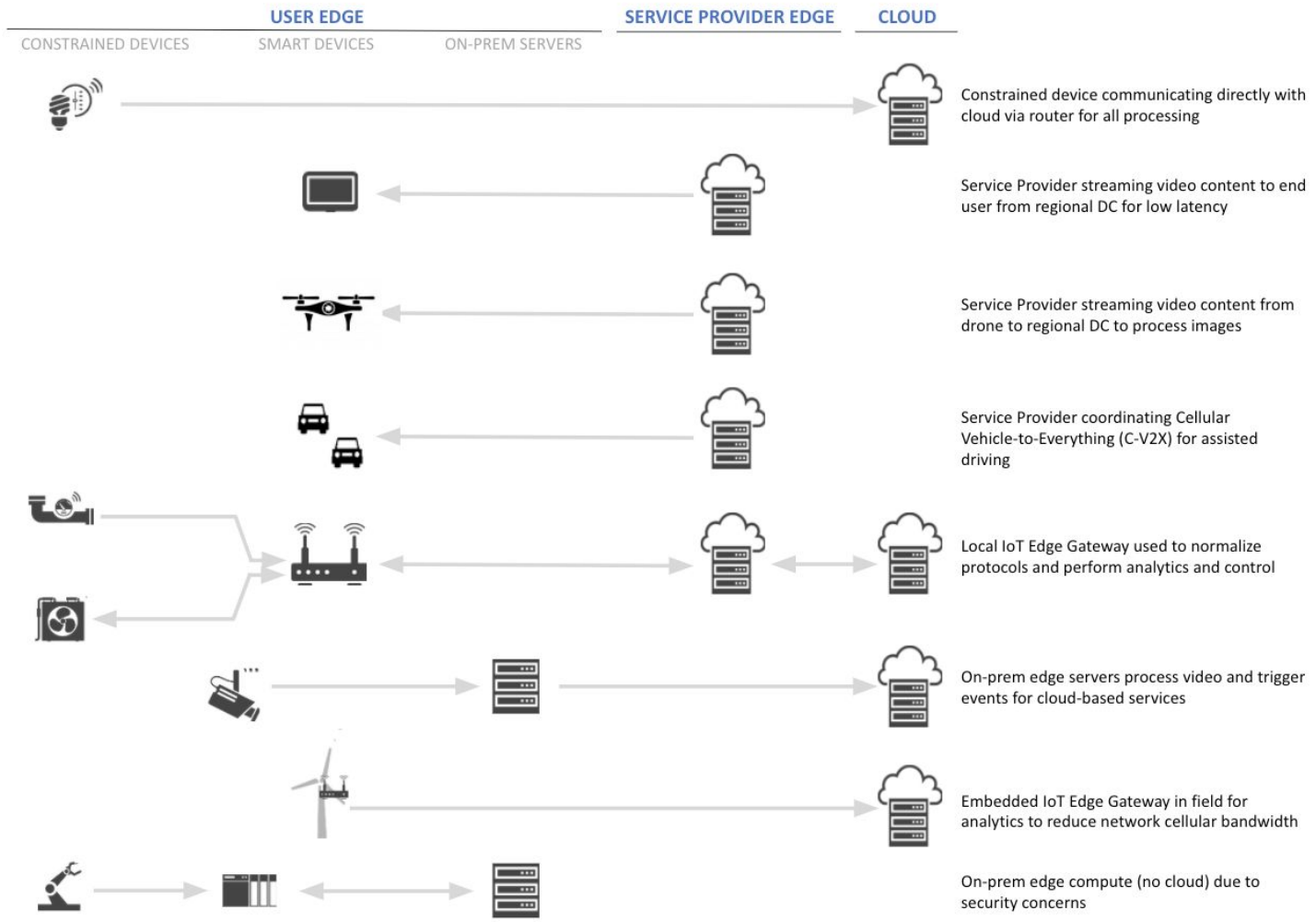


Figure 4. Example deployment patterns across the edge continuum

Note that this diagram is simplified in the sense that it does not take into account that resources will be communicating "north, south, east and west" with multiple peers across the continuum depending on use case. This is what some refer to as fog computing.

The Smart Device Edge includes both headless compute resources targeted at IoT use cases and client devices (e.g. smartphones, tablets, PCs, gaming consoles, smart TVs) that have a user interface. Combined, Constrained and Smart Devices represent the "things" in IoT solutions, with the latter providing localized general-purpose compute capability. As a general trend in the area of networking, IoT use cases tend to be constrained by the upload of data collected from the physical world whereas end user client use cases tend to be constrained by content download. This results in different considerations for applications, storage, network topologies and so forth, depending on the use case and available resources.

Securing and managing distributed devices

Resources at the Constrained and Smart Device Edges are typically deployed and used in semi-secure to easily accessible locations in the field. As such, it is important to adopt a zero-trust security model and not pre-suppose a device is behind a network firewall. In all cases, distributed computing resources need remote software update capability to avoid costly truck rolls, and in the case of On-Prem Data Centers and Smart Devices, evolve capability over time through

modular, software-defined architecture. However, M&O and security solutions optimized for the data center are not suitable for the Constrained and Smart Device Edges due to available compute footprint, deployment scale factor, potentially periodic connectivity and typical lack of physical and network security. Solutions should also not require IT skill sets for secure deployment in the field through features like zero-touch provisioning.

IoT and client-centric compute resources at the Smart Device Edge are capable of leveraging M&O tools that support abstraction in the form of containerization and virtualization and have headroom for security features like data encryption. Meanwhile, constrained devices leverage embedded software images that are typically tailored to the host hardware and may need to rely on a more capable device immediately upstream for added security measures. As a result, M&O tools for the Constrained Device Edge are often unique by device and manufacturer. Meanwhile smart devices can afford the necessary abstraction to make M&O tools more standardized and platform-independent, whether it be Linux with containers working across both x86 and Arm-based IoT gateways or a mobile operating system such as Android supporting apps on a variety of manufacturers' smartphones. Whenever possible, all key security functions (e.g. authentication, boot, encryption) should be enabled by a hardware-based root of trust (e.g. TPM, Arm TrustZone), but this is not always an option for highly-constrained devices.

Accommodating legacy and modern applications

As with the cloud data center, many User Edge compute resources have a need to accommodate legacy applications in parallel to modern, cloud-native workloads. This is relatively straightforward in an on-prem data center through the utilization of well-established enterprise virtualization software together with Kubernetes, however it is not feasible to leverage these same tools on more constrained hardware deployed in the field. Here is where special consideration must be made for an abstraction layer that is optimized for resource-constrained hardware and comprehends the unique security needs for devices distributed outside of a secure data center. Today the ability to abstract virtualized and/or containerized workloads on a given compute node is typically limited by available memory, with the practical lower limit being roughly 256MB - just enough to host an abstraction layer and a workload. This memory constraint is the primary delineator between the Smart and Constrained Device Edges and is generally the limit for extending cloud-native software development principles closer to the source of data. Below this memory capacity, software needs to be embedded with tight coupling to hardware which limits flexibility and expandability through abstracted, modular applications.

Addressing protocol fragmentation in IoT use cases

Further, compared to the entirely IP-based data flow spanning the Cloud, Service Provider and On-Prem Data Center Edges, Constrained and Smart Device resources serving IoT workloads must comprehend a diverse mix of legacy and modern connectivity protocols - spanning wired and wireless transport and both standard and proprietary formats. This is especially the case in the Industrial IoT space where there are literally hundreds of legacy protocol formats to comprehend. Rather than expecting one connectivity standard to rule them all, it is important to have edge software frameworks that can normalize a variety of IoT data sources into desired IP-based formats for further processing upstream. Openness here enables users to retain control over their data by not getting locked into any particular backend service.

Latency critical applications

Another key driver for running workloads at the User Edge is safety and latency *critical* applications that require "hard" real time operation for deterministic response. Resources like PLCs, RTUs and ECUs have been used in industrial process control, machinery, aircraft, vehicles and drones for many years, requiring real-time operating systems (RTOS) and specialized, fixed function logic. Time and safety-critical processes such as controlling a machine, applying a vehicle's brakes or deploying an airbag are universally operated locally because they simply can't rely on control over a last mile network, regardless of how fast and reliable that connection is. This is compared to latency *sensitive* applications such as video streaming that operate in "soft" real time and are often served up by the Service Provider Edge for scalability. With latency sensitive applications a networking issue can result in a poor user experience but not a critical, potentially life-threatening failure.

Separation of concerns in Industrial / Operational Technology (OT) environments
Operational Technology (OT) organizations have historically isolated their infrastructure (e.g. PLCs, SCADA and MES systems) from broader networks for security and ensuring uptime but a key aspect of Industrial IoT is connecting these assets and the associated processes to networked intelligence to drive new outcomes. In order to create a separation of concerns from control systems with no risk of disrupting existing processes, industrial operators rely on network segmentation, typically installing a secondary “overlay network” that taps into data from existing control systems, in addition to new sensors installed throughout their environment to enable analytics workloads. Meanwhile, there is a trend for consolidation of mixed-criticality workloads on common infrastructure, for example with a virtualized “soft PLC” providing control functionality while additional virtualized and/or containerized data management, security and analytics applications run in parallel and interact with higher edge tiers. This consolidation requires specific considerations in the abstraction layer to ensure separation of concerns between these mixed-criticality workloads.

Net-net, developers need flexible tools at the User Edge that enable them to run legacy, safety/latency-critical, and modern containerized workloads concurrently while protecting their operations from undue risk, all while taking advantage of the scale benefits of working together with the Service Provider Edge and the Cloud. Refer to the table in Section 1.7 for a detailed summary of attributes by edge tier.

1.5 Trends for Edge AI

Regarding Artificial Intelligence and Machine Learning (AI/ML) at the edge, the general trend is for deep learning and model training to occur in the cloud, with models subsequently being pushed to compute resources at the Service Provider and User Edges for performing inferencing on data locally. Location of model execution along the edge continuum depends on a variety of factors, including addressing latency issues, ensuring autonomy, reducing network bandwidth consumption, improving end user privacy and meeting requirements for data sovereignty.

There is an emerging trend for federated learning and even training models at the edge to address privacy and data sovereignty issues, however the potential for regional bias then needs to be considered. Another emerging trend at the Constrained Device Edge is deploying ML inferencing models in microcontroller-based resources. An example is a ML model that enables a smart speaker to recognize a wake word (e.g. “Hey Google/Alexa”) locally before subsequent voice interactions are powered by servers further up the compute continuum. Dubbed “Tiny ML”, this requires highly specialized tool sets to accommodate the available processing resources and is outside of the scope of LF Edge at the time of this writing.

Refer to the next section for more detail on various use cases involving edge analytics.

1.6 Edge Computing Use Cases

Enterprises in several market segments are deploying edge-hosted applications in order to capitalize on new business opportunities that are enabled by provisioning local compute as an extension of centralized cloud architectures. Example use cases discussed in this section include:

- Industrial IoT;
- Computer Vision;
- Augmented Reality;
- Retail;
- Gaming;
- Assisted driving

Industrial IoT (IIoT)

Edge compute delivers a number of important benefits for Industrial IoT (IIoT) use cases in markets such as manufacturing, utilities, oil and gas, agriculture, and mining.

With edge computing, industrial operators can perform time-critical analytics close to their sensors, machines and robots, reducing the latency for operational decision-making. This makes their processes more agile and responsive to changes. To maximize efficiency and minimize OPEX, functions necessary for real-time operation are hosted on-premise while those that are less time-critical may run in the public, private or hybrid cloud.

An edge compute architecture can ensure that high-value, proprietary information never leaves a factory. This can minimize the security threats associated with transmitting data to the cloud over public networks that are vulnerable to hacking, as well as help organizations meet data sovereignty requirements.

Remote operations such as mines or oil rigs typically have intermittent connectivity to the cloud and must be able to function autonomously. In these scenarios, on-premise Device Edge compute enables real-time operational decisions based on the local analysis of sensor data. Data required for long-term process optimization or multi-site aggregation is sent to the cloud whenever connectivity is available, which in certain locations might only be over a satellite link available at irregular intervals.

A significant amount of data is “perishable”, meaning it is only valuable if acted on in the moment. The cost of connectivity through the Service Provider Edge is minimized by processing data from sensors locally and sending only relevant information to the cloud, instead of raw streams of data. This is critical for high-bandwidth vibration data used for predictive maintenance use cases or devices like smart meters used in agriculture or utilities that connect to the cloud via low-bandwidth Narrowband IoT (NB-IoT) networks. In these cases, additional data might periodically be centralized in the cloud for the training of AI models that are then pushed close to operations at the edge for inferencing.

Computer Vision

Computer vision technology is widely used in video surveillance for law enforcement and building security, as well as monitoring industrial processes. Modern high-resolution IP cameras, however, generate significant volumes of data, for example 4 Mbps per device for an array of 4-megapixel (MP) cameras. While cameras can be configured to minimize bandwidth requirements by transmitting only when motion is detected, this is of no help in environments such as a city street or factory production line where the surveillance system network connection must be provisioned to cope with constant motion. Analyzing video in the cloud therefore requires a high-bandwidth network connection to transmit a continuous, high-resolution stream. If network bandwidth is constrained, the accuracy of the analysis is limited by the lower resolution of compressed video.

With edge compute however, high-resolution video data is processed either within the smart camera itself as the edge node, or on a nearby edge server. High-end IP cameras have sufficient processing power to run algorithms such as facial recognition, leveraging analytics based on Artificial Intelligence (AI) and/or deep learning technologies. Only selected events and/or video sequences that are flagged as important are transmitted to the cloud, for example an individual of interest, a vehicle with a specific license plate or a defective component. This significantly reduces the required network bandwidth while ensuring high quality, high accuracy analytics.

Edge compute also reduces latency, which is important for any time-critical vision-based detection scenarios such as factory automation or facial recognition. Continuous process control, for example, leverages the low latency associated with edge compute to ensure the near-real time detection of process deviations or manufacturing flaws, enabling production lines to be stopped or control parameters to be adjusted quickly enough to minimize wastage.

Drones used in applications such as surveying, package delivery and surveillance will leverage low-latency computer vision systems that perform object recognition for navigation within edge compute nodes on the ground rather than in heavy, power-hungry systems on the drone itself. This reduces the cost of the drones while also minimizing their power consumption, thereby maximizing both battery life and flight time.

Processing video at the User Edge also mitigates privacy concerns, especially in surveillance applications that are subject to regulatory constraints or in commercial applications where process information is valuable intellectual property.

Augmented Reality (AR)

Enterprises are increasingly adopting Augmented Reality (AR) in order to improve the efficiency of their operations, leveraging technology familiar to consumers through applications like Pokémon Go and its more sophisticated successors.

In industrial environments, AR can guide lesser-skilled workers through maintenance tasks without having an expert engineer on site at all times. This can either be done with a pre-scripted instruction overlay in the worker's field of view, or with a remote expert talking theon-site worker through performing a complicated task through their eyes. Similarly, in aerospace AR provides technicians with maintenance and diagnostic information via smart goggles, eliminating the need to physically reference bulky, complex and possibly outdated manuals in hard-to-reach locations inside a wing, fuselage or engine cowling.

AR applications typically analyze the output from a device's camera to supplement the user's experience. The application is aware of the user's position and the direction they are looking in, with this information provided via the camera view and/or positioning techniques. The application is then able to offer information in real-time to the user, but as soon as the user moves that information must be refreshed. Additionally, for many use cases it is valuable to update critical real time data from sensors in the user's field of view, for example the temperature and pressure of a tank while an operator is working through a maintenance procedure.

Edge compute improves the efficiency of enterprise AR by reducing the dizziness associated with high latency and slow frame refresh rates, that can otherwise lead to an experience that is frustrating, potentially nausea inducing and ultimately disorienting. Edge-hosted systems ensure predictable latency, resulting in a consistent experience for users instead of the constantly-changing delays that result from cloud-hosted implementations. Moving compute power into edge servers located close to the user allows an AR application to eliminate the need for high processing bandwidth on goggles that therefore become expensive, power-hungry and too heavy for comfortable use over an extended period.

In another example, edge computing and AR are poised to deliver truly immersive media experiences for sports fans while at the game. Sports such as baseball, cricket, football and soccer have already held successful trials in "smart stadia" enabling spectators to stream video from unique, custom camera angles, including drones and spider-cams. "Virtual cameras" present views from within the field of play, giving spectators the opportunity to experience the action from the perspective of the players themselves. All these use cases require edge compute in order to guarantee the responsiveness that spectators expect while eliminating the need to backhaul prohibitive amounts of data to the cloud.

Retail

For brick-and-mortar retailers, almost 90% of global retail sales occur in physical stores so most retailers are investing in computing infrastructure located closer to the buyer, with edge computing as an extension of their centralized cloud environments. In-store edge environments focus on the digital experience of the customer, through edge applications supporting local devices such as smart signage, AR-based mirrors, kiosks and advanced self-checkout.

Retailers can deliver personalized coupons when shoppers walk into stores as WiFi, beaconing and computer vision systems recognize customers who previously signed up to connect while in-store. Smart fitting rooms equipped with AR mirrors can show shoppers in different clothing without the requirement to physically try them on. Meanwhile, infrared beacon and computer vision technology can generate heat maps that provide retailers with insights on in-store traffic patterns, allowing them to better configure their space and optimize their revenue-per-square-foot.

Infusing self-checkout systems with computer vision capability and integrating them with RFID and Point-of-Sale systems gives them the ability to confirm that the item scanned by a customer matches what's in their bag, improving loss prevention. Vision algorithms can also be used to enable facial recognition to authenticate payments and gesture recognition for touchless commands, as well as the delivery of personalized offers at the point of sale.

Through the use of edge compute, retailers are able to ensure improved security for sensitive customer information. When data is transferred from devices to the cloud, security and compliance risks increase, but edge compute applications can filter information locally and only transfer data to the cloud that is required for strategic operational planning.

Edge compute provides a lean, highly reliable IT infrastructure for retailers that can run multiple applications while supporting the control and flexibility of cloud-based services. High-resiliency in-store micro data centers have become the solution of choice, managed and orchestrated remotely so that IT staff aren't required on-site. A chain of retail stores can be treated as an entire ecosystem rather than just a collection of individual locations.

Gaming

Massively Multiplayer Games (MMPGs) served up by the cloud typically involve players controlling their avatars, with any movement of an avatar needing to be communicated as quickly as possible to all players who have that avatar in their field of view. Latency has a major impact on the overall user experience, to the point where perceptible delays can render a game effectively unplayable. A video game must appear to respond instantaneously to keystrokes and controller movements, implying that any commands issued must complete a round-trip over the network and be processed fast enough by the data center for the player to feel like the game is responding in real time. For the best multiplayer experience, the latency must be consistent across all players, otherwise those with the lowest latency have the opportunity to react faster than their competitors.

Edge compute improves the experience of cloud-enabled gaming by significantly reducing latency and providing the necessary storage and processing power in edge data centers. With processing centers for a game running at the edge of the network, for example in each metro area, the ultra-low latency results in reduced lag-time. This enables a more interactive and fully immersive experience than if the game is hosted in a remote cloud data center.

Edge compute is expected to trigger new subscription-based MMPG business models along with reduced hardware costs for end-users: with edge processing enabling high-quality experiences, less processing power is required in the users' hardware itself. The gaming industry hopes that this reduction in hardware costs will spur greater user investment in new subscriptions, driving overall growth in this segment.

Assisted driving

While edge compute will be a critical enabler for the holy grail of fully-autonomous driving, that vision is many years away from being realized, for a host of reasons beyond the scope of this paper. Assisted driving technologies, however, are being deployed today and edge compute is key to their viability.

The number of sensors in a vehicle grows with each model year, along with each introduction of new capabilities in safety, performance, efficiency, comfort and infotainment. Although most of the sensor data is processed in the vehicle itself due to autonomy and safety considerations of latency critical applications, some capabilities, such as alerting in the event of deviations from the norm, require data to be moved to the cloud for analysis and follow-up. Edge computing helps to limit

the amount of data that is sent to the cloud, reducing the data transmission cost and minimizing the amount of sensitive data such as Personally Identifiable Information (PII) leaving the vehicle.

The infotainment system in a vehicle is the most prominent user interface besides the driving controls. To learn what functions and applications users are really using and where the design of interactions should be optimized, ML algorithms represent an important tool for uncovering relevant insights within the vast amount of available data. Edge compute brings ML models, which were trained in the cloud, to the vehicle itself, so that the available behavioral and sensor data can be used locally for predictions that improve overall user interaction.

Efficient battery monitoring and predictive maintenance are key to the long-term customer experience for vehicle owners and operators. Edge compute addresses these challenges through the ability to aggregate data and perform the real-time evaluation of relevant battery parameters and sensor values. Appropriate information can be automatically uploaded to backend operational systems in the cloud, enabling dealers or fleet operators to automatically schedule preventative maintenance at a time and place that balances convenience for the user against the severity of problems that have been detected.

Edge compute technologies can enable secure, frictionless entry to a vehicle based on multi-factor authentication, for example using a camera for face recognition, an infrared camera for spoofing detection and a Bluetooth sensor to detect the proximity of the driver's smartphone.

Finally, once the proportion of smart vehicles reaches a critical threshold within a certain geography, smart traffic management will become feasible, enabled by roadside edge compute. In one example, if a road intersection has an edge node deployed to which the majority of vehicles can communicate while coming towards the intersection, the edge node can aggregate the location and speed data from nearby vehicles, optimize traffic light timing for efficient traffic flow and notify the smart vehicles in advance about the situation at the intersection. Widespread deployments of such edge nodes will enable Cellular Vehicle-to-Everything (C-V2X) applications that not only traffic flows for individual intersections, but over wider areas thanks to the cloud-based analysis of edge data and the centralized orchestration of the individual intersections.

1.7 Summary of the Edge Continuum

Each edge tier represents unique tradeoffs between scalability, reliability, latency, cost, security and autonomy. In general, compute at the User Edge reflects dedicated, operated resources on a wired or wireless local area network (LAN) relative to the users and processes they serve. Meanwhile, the Service Provider Edge and Public Cloud are shared resources (XaaS) on a wide area network relative to users and processes. The table below summarizes key attributes of each edge.

Table 1. Summary of Edge Attributes

Attribute	User Edge			LAST MILE NETWORK	Service Provider Edge	Cloud
	Constrained Device Edge	Smart Device Edge	On-prem Data Center Edge			
Hardware Class	Constrained microcontroller-based embedded devices (e.g. voice control speakers, thermostats, light switches, sensors, actuators, controllers). KBs to low MBs of available memory.	Arm and x86-based gateways, embedded PCs, hubs, routers, servers, small clusters. >256MB of available memory but still constrained. Accelerators (e.g. GPU, FPGA, TPU) depending on need.	Standard servers and networking with accelerators		Standard servers and networking with accelerators, telco radio infrastructure	Standard servers and networking with accelerators
Deployment Locations	Highly distributed in the physical world, embedded in discrete products and systems	Distributed in field, outside of secure data centers (e.g. factory floor, equipment closet, smart home) or embedded within distributed systems (e.g. connected vehicle, wind turbine, streetlight in public R.O.W.)	Secure, on-premise data-centers and micro-data centers (MDCs), e.g. located within an office building or factory. Typically owned and operated by enterprises.		CO, RO, Satellite DCs, owned and operated by service providers (e.g. ISPs, CSPs). Resources can also be located at user edge in the case of CPE owned and managed by a service provider.	Centralized DCs, Zones, Regions owned and operated by CSPs. Compute in DCs located near key network Points of Presence (PoP) is at the "Cloud Edge" or "Internet Edge".
Global Node Footprint	Trillions	Billions	Millions		Tens of Thousands	Hundreds
Role/ Function	Fixed to limited function applications, rely on higher-classes of compute for advanced processing. Emerging simple ML capability via TinyML.	Hyperlocal general compute for apps and services. Dynamic, SW-defined configuration with limited scalability. Includes IoT Compute Edge (headless systems) and End User devices (e.g. smartphones, PCs, gaming consoles).	Local general compute for applications and services with moderate scalability. Dedicated to a specific enterprise.		High availability, pubic and private, general and special. Broad scalability. Shared resources for IaaS, PaaS, SaaS, SDN (XaaS).	Hyperscale or webscale, public, general purpose. Public cloud involved shared resources for IaaS, PaaS, SaaS, SDN (XaaS).
Software Architecture	Embedded software/ firmware, Real-time Operating Systems (RTOS) for time-critical applications.	Bare metal to containerized/ virtualized depending on capability and use case. Linux, Windows and mobile OS'es (e.g. Android, iOS).	Virtualized, containerized and clustered compute. Linux and Windows.		Virtualized, containerized and clustered compute. VNF, CNF, managed services, networking. Linux and Windows.	Bare metal, VMs, Clusters, Containers, all architectures, all services. Linux and Windows.
Security, M&O	Specialized OTA M&O tools, often custom by device/manufacturer. May rely on higher-class compute for security.	Require specific security and M&O tools due to resource constraints, unique functionality, accessibility and limited field technical expertise. Often unable to rely on a network firewall.	Evolution of cloud data center security and M&O tools to support distributed Kubernetes clusters. Benefits from physical and network security of purpose-built data centers.		Evolution of cloud data center security and M&O tools to support distributed Kubernetes clusters in regional locations	Traditional cloud data-center security and M&O tools
Physical Attributes	Highly-specific form factors for every device	Diverse mix of specialized form-factors with unique I/O, industrial ruggedization, regulatory certifications, etc. based on use case	General purpose server-class infrastructure with some ruggedization and regulatory considerations (e.g. for MDCs)		General purpose server and networking infrastructure with power, thermal, ruggedization and regulatory considerations for localized resources	General purpose server infrastructure

It is important to recognize that the boundaries between these edge tiers are not rigid. For example, as shown in Figure 3, the Service Provider Edge bleeds into the User Edge when CPE resources are deployed on-premise in order to provide a user with connectivity and compute as a managed service. We will see an increasing blur over time, along with more and more processing capability distributed across the User Edge. However, certain technical and logistical limitations will always dictate where workloads are best run across the continuum based on any given context.

Regardless of the definitions of various edge tiers, the ultimate goal is to provide developers with maximum flexibility, enabling them with the ability to extend cloud-native development practices as far down the continuum as possible, while recognizing the practical limitations. The following sections dive deeper into LF Edge and how each project within the umbrella is working to realize this goal.

2 LF Edge Project Portfolio

The Linux Foundation's LF Edge (LFE) was founded in 2019 as an umbrella organization to establish an open, interoperable framework for edge computing independent of hardware, silicon, cloud or operating system. The project offers structured, vendor neutral governance and has the following mission:

- Foster cross-industry collaboration across IoT, Telecom, Enterprise and Cloud ecosystems
- Enable organizations to accelerate adoption and the pace of innovation for edge computing
- Deliver value to end users by providing a neutral platform to capture and distribute requirements across the umbrella
- Seek to facilitate harmonization across Edge projects

As with other LF umbrella projects, LF Edge is a technical meritocracy and has a Technical Advisory Committee (TAC) that helps align project efforts and encourages structured growth and advancement by following the Project Lifecycle Document (PLD) process. All new projects enter as Stage 1 "At Large" projects which are projects that the TAC believes are, or have the potential to be, important to the ecosystem of Top-Level Projects, or the Edge ecosystem as a whole. The second "Growth Stage" is for projects that are interested in reaching the Impact Stage, and have identified a growth plan

for doing so. Finally, the third “Impact Stage” is for projects that have reached their growth goals and are now on a self-sustaining cycle of development, maintenance, and long-term support.

2.1 LF Edge Project Summaries

LFE comprises the following open source projects, explained in more detail in the appendices:

- **Akraino Edge Stack** is a software stack that supports high-availability cloud services optimized for edge computing systems and applications. It offers users new levels of flexibility to scale edge cloud services quickly, to maximize the applications and functions supported at the edge and to help ensure the reliability of systems that must be completely functional at all times. Akraino Edge Stack delivers a deployable and fully-functional edge stack for edge use cases including Industrial IoT (IIoT), telco 5G core, virtual Radio Access Network (vRAN), Universal Customer Premises Equipment (uCPE), Software-Defined Wide Area Network (SD-WAN) and edge media processing. It creates a framework for defining and standardizing APIs across stacks, via upstream/downstream collaboration. Akraino Edge Stack is currently composed of multiple blueprint families that include specific blueprints under development. The community tests and validates the blueprints on real hardware labs supported by users and community members. Akraino is a Stage 3 Impact project.
- **Baetyl** (pronounced “Beetle”) is a general-purpose platform for edge computing that manipulates different types of hardware facilities and device capabilities into a standardized container runtime environment and API, enabling the efficient management of application, service and data flow through a remote console both in the cloud and on-premise. Baetyl is a Stage 1 At Large project.
- **EdgeX Foundry** is a vendor-neutral, loosely-coupled microservices framework that enables flexible, plug-and-play deployments that leverage a growing ecosystem of available third-party offerings or to include proprietary innovations. At the heart of the project is an interoperability framework hosted within a full hardware- and OS-agnostic reference software platform. The reference platform helps enable the ecosystem of plug-and-play components that unifies the marketplace and accelerates the deployment of IoT solutions. EdgeX Foundry is an open platform for developers to build custom IoT solutions, either by feeding data into it from their own devices and sensors, or consuming and processing data coming out. EdgeX Foundry is a Stage 3 Impact project.
- **Fledge** is a framework for the industrial edge focused on critical operations, predictive maintenance, situational awareness and safety. Fledge is architected for the rapid integration of any IIoT device, sensor or machine with existing industrial "brown field" systems and clouds, using a common set of application, management and security REST APIs. Fledge edge services include: collect data from sensors; aggregate, combine and organize data; edge-based alerting, anomaly detection and machine learning (TensorFlow Lite, OpenVino); transform and filter data in flight; buffer, analyze and visualize edge data; deliver data to multiple local and/or cloud destinations. Fledge is a Stage 1 At Large project.
- **Home Edge** is a robust, reliable and intelligent home edge computing open source framework, platform and ecosystem. It provides an interoperable, flexible and scalable edge computing services platform with APIs that can also be used with libraries and runtimes. Home Edge is a Stage 2 Growth project.
- **Open Horizon** is a platform for managing the service software lifecycle of containerized workloads and related machine learning assets. It enables management of applications deployed to distributed webscale fleets of edge computing nodes and devices without requiring on-premise administrators. Open Horizon is a Stage 1 At Large project.
- **Project EVE** is an edge computing engine that enables the development, orchestration and security of cloud-native and legacy applications on distributed edge compute nodes. Supporting containers, clusters VMs and unikernels, it provides a flexible foundation for IoT edge deployments with a choice of hardware, applications and clouds. EVE is a Stage 2 Growth project.
- **State of the Edge** is a vendor-neutral platform for open research on edge computing dedicated to accelerating innovation by crowdsourcing a shared vocabulary for the edge. The project develops free, shareable research that is

widely adopted and used to discuss compelling solutions offered by edge computing. State of the Edge is a Stage 2 Growth project.

2.2. Project Focus Across the Edge Continuum

The general focus area for each project along the edge continuum is depicted in Figure 1, however the scope of each project tends to extend further across the spectrum as it integrates with various upstream and downstream efforts.

Akraino is focused on addressing the unique infrastructure needs of the Service Provider Edge, with reach into the various subcategories of the User Edge through holistic blueprints.

The mission of Project EVE is to create a universal orchestration foundation for enterprise and industrial IoT edge computing use cases at the Smart Device Edge, like Android has provided for smartphones. EVE addresses the need to accommodate both legacy and modern applications on constrained IoT edge compute resources while meeting the unique security and scale requirements for devices deployed outside of the data center.

Baetyl and Open Horizon are focused on enabling the delivery of containerized workloads to resources distributed across the Smart Device Edge but also have a footprint that extends through the Service Provider Edge to the Cloud. The Open Horizon controller can be deployed centrally in the cloud, regionally at the Service Provider Edge or locally at the Open Prem Data Center Edge.

EdgeX Foundry and Fledge serve as application frameworks for IoT use cases at the Smart Device Edge to address the fragmentation in the market stemming from diverse technology choices spanning hardware, operating system and connectivity protocols. These frameworks provide an open foundation for deploying analytics and other value-added services, with each taking a slightly different architectural approach that balances tradeoffs between flexibility, portability, footprint and performance. Their efforts bridge to the Constrained Device Edge, facilitate local data processing, and in turn relay data to and from higher edge tiers.

Home Edge is focused at the Smart Device Edge for consumer use cases in the home.

The State of the Edge project spans the entire edge computing continuum, conducting research and producing related reports, in addition to curating the Open Edge Computing Glossary.

LF Edge will add more projects over time with a philosophy of being inclusive but also offering structure and promoting increasing harmonization. Per the project mission, the community aims to develop common best practices and eventual unification of APIs as appropriate. The result will be an open ecosystem for edge computing with infrastructure that can be context-aware of the needs of workloads running above, regardless of who wrote them. As an example, imagine a world where infrastructure could prioritize quality-of-service for a healthcare app running right next to one that delivers entertainment content.

3 Cross-Project Collaboration

In order to ensure a comprehensive development and deployment environment for edge computing efforts, there is significant collaboration between the various LFE projects covered in this report:

3.1 Akraino Edge Stack

Akraino Edge Stack is implemented with embedded interfaces, including for Acumos AI, Airship, Ceph, DANOS, EdgeX Foundry, Kubernetes, LF Networking, ONAP, OpenStack Ocata and StarlingX.

3.2 Baetyl

Baetyl works with EdgeX, which provides the basic operating environment, remote upgrade and micro-service capabilities for EdgeX applications.

3.3 EdgeX Foundry

EdgeX works with Home Edge, such that EdgeX is a central component of the Home Edge solution, with EdgeX providing core services for persistence and application services for export.

The Project EVE edge computing engine can be used under EdgeX and EdgeX containers can be deployed using Open Horizon

3.4 Fledge

Fledge works closely with Project EVE, which provides system and orchestration services and a container runtime for Fledge applications and services. Together industrial operators can build, manage, secure and support all their non-SCADA, non-DCS connected machines, IIoT and sensors as they scale.

Fledge is also integrated with Akraino Edge Stack, as both projects support the roll out 5G and private LTE networks.

It also complements EdgeX Foundry on the industrial side. EdgeX has extensive control features, focused where data management and integrated control are required, while Fledge specifically works where data from brown field systems is not connected to the primary control systems (often serviced by RBM) or environmental / situational / safety sensors.

3.5 Home Edge

Home Edge works with EdgeX Foundry in regards to its Data Storage Module, which is one of the main collaboration items for 2020.

3.6 Project EVE

Project EVE is complementary to LFE application frameworks such as EdgeX Foundry and Fledge The Project EVE community is working to harmonize with other LFE infrastructure projects such as Akraino and Open Horizon

3.7 State of the Edge

State of the Edge aligns terminology across all LFE projects. The State of the Edge project encourages other LFE projects to help adopt, create and improve the canonical definitions in order to present a uniform language to users and contributors. By standardizing across projects, this project accelerates learning and improve cross-project integrations.

3.8 Open Horizon

Open Horizon originally incubated with EdgeX Foundry, occupied an at-large seat on EdgeX's TSC, and collaborated with EdgeX in a sub-group on demonstration code that delivered and managed an EdgeX Foundry instance with Open Horizon. The collaboration is ongoing.

Open Horizon is working with Home Edge to deliver ML assets from the cloud to a Home Edge gateway.

Open Horizon is actively negotiating with Akraino leadership and several Akraino Blueprints to determine the most effective area to begin contributing.

Discussions and collaborations are also underway with several other current and prospective LF Edge projects.

Appendix A **Akraino Edge Stack**

Introduction

Akraino Edge Stack is an open infrastructure project for hybrid edge cloud deployments. It evolved from the Airship project, a declarative platform with OpenStack and Kubernetes that provides life cycle management of cloud-hosted applications. Akraino Edge Stack is lightweight. It meets the necessary latency and bandwidth standards for 5G networks, which are 20ms maximum latency (measured from the end user to the network backbone) and a minimum bandwidth of 20Gbps.

Akraino Edge Stack is designed for the deployment and orchestration of more than a thousand remote edge locations with support for disaggregated hardware. Its low latency switch delivers latency below 50µs.

Akraino Edge Stack leverages existing mature frameworks (e.g. Airship, Jenkins, OpenStack and Kubernetes) to automate Continuous Integration and Continuous Deployment (CI/CD) as well as zero-touch management of carrier-scale edge computing applications. It provides scalability, high availability, high speed, resiliency, security and privacy-preservation. It ensures the flexibility and operational predictability demanded by various applications in real-time video processing, IoT security, Augmented Reality (AR) and Virtual Reality (VR).

Key Principles

Edge computing requires significant efforts associated with seamless integration and user experience. Also, edge computing solutions require large-scale deployment, typically covering over a thousand locations. Keeping these challenges in mind, the key requirement for the Akraino Edge Stack project is to keep the cost low and ensure it supports large-scale deployments via automation. The Akraino Edge Stack community works with multiple upstream open source communities such as Airship, ONAP, OpenStack etc. to deliver a fully integrated stack. Akraino Edge Stack supplies a fully-integrated solution that supports zero-touch provisioning and zero-touch lifecycle management of the integrated stack.

Akraino Edge Stack adheres to key principles with respect to design, build and run, as described below.

Design Principles

Akraino Edge Stack follows a holistic design with respect to availability, capacity, security and continuity:

- Finite set of configurations: in order to reduce the complexity, the design implements a finite set of configurations.
- Cloud native applications: the design also includes cloud-native applications.
- Simplified security: the design provides a secure platform and services, while minimizing the platform overhead imposed by security.
- Autonomous, turnkey solution for service enablement to enable rapid introduction.
- Assessment and gating for platforms, Virtual Network Functions (VNFs) and applications: to assess whether the application is suitable to run at the edge (e.g. with reference to latency sensitiveness and/or code quality).

Build Principles

Akraino Edge Stack is built to scale in a cost-effective way and will evolve to ensure that costs are optimized:

- Low latency placement and processing to support edge drivers;
- Plug and play modular architecture, with building blocks that leverage multiple cloud management technologies.

Run Principles

Akraino Edge Stack adheres to the following run principles:

- Zero-touch provisioning, operations and lifecycle in order to reduce OPEX;
- Automated maturity measurement for operations, designs and services;
- Software abstraction-based homogeneity to abstract any hardware differences via software;
- Common platform and service orchestration leveraging ONAP.

Architecture Description

Akraino Edge Stack follows a minimum human interference design with respect to availability, capacity, security and continuity. In order to reduce the complexity, the design implements a finite set of configurations. The design also supports cloud-native applications. It provides a secure platform and services while minimizing the platform overhead associated with security. It provides zero-touch provisioning, operations and lifecycle management for the hardware platform. It is implemented with embedded interfaces, including for Acumos AI, Airship, Ceph, DANOS, EdgeX Foundry, Kubernetes, LF Networking, ONAP, OpenStack Ocata and StarlingX.

The figure below provides a brief description of the end-to-end CI/CD stack at run-time. Akraino Edge Stack leverages:

- OpenStack Ocata for VM orchestration;
- Docker with Kubernetes for container orchestration;
- Redfish Linux for the base Operating System;
- Open vSwitch with Dataplane Development Kit (DPDK) and Single-Root Input/Output Virtualization (SR-IOV) for fast kernel stack bypassing;
- ONAP for VNF orchestration;
- Airship for under-cloud orchestration.

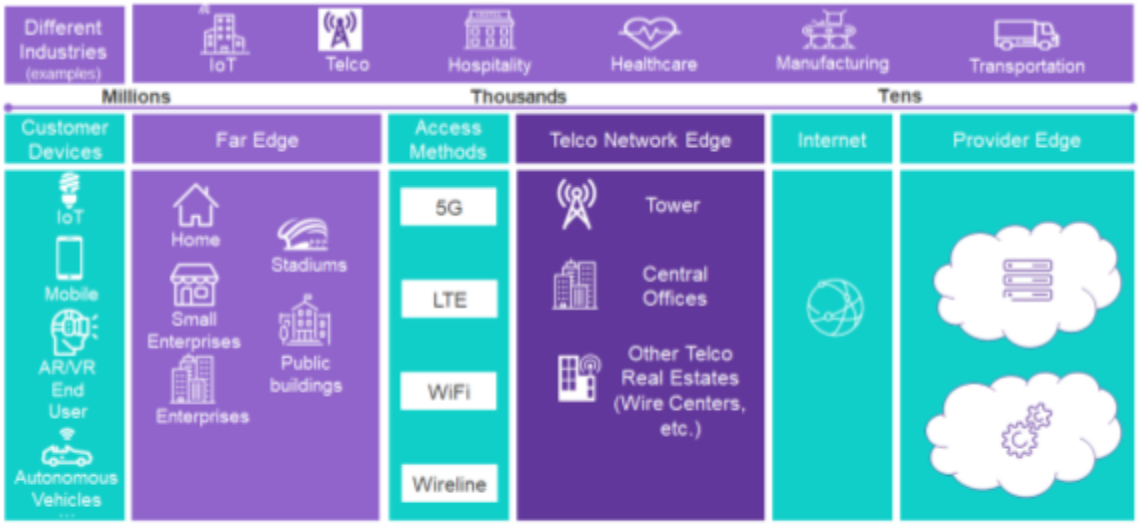


*Figure SEQ Figure * ARABIC 4: Akraino Edge Stack Architecture.*

A low-latency virtual switch is also implemented, with packet latency below 50µs. It utilizes hardware support for security features, such as Trusted Platform Module (TPM) for storing certificates, virtual TPM (vTPM) for applications, a tuned OS to minimize the attack surface and Integrity Measurement Architecture (IMA) for monitoring of critical files for tampering.

For auto-scaling, Akraino Edge Stack implements five Point of Delivery (POD) methods. POD is the method in which a project is deployed to an edge site. For example, an edge location could have a single server or multiple servers installed in one or more racks. As the blueprint uses the declarative configuration, POD is used to declare how edge devices are organized for the deployment. POD allows a scalable, cookie-cutter approach to large-scale deployments (e.g. 10,000+ locations) at minimum cost. The above figure shows various PODs.

Operational needs are addressed through the capability of updating only necessary components without requiring a full redeployment of the stack. Security vulnerabilities can be deployed asynchronously. A distributed cloud infrastructure supported by redundant hyper-converged sub-clouds allows the use of minimal hardware configurations at the far edge, underpinned by a centralized cloud infrastructure management framework

Framework

Akraino Edge Stack provides a standalone framework.

Features

The Akraino Edge Stack is built to:

- Scale in a cost-effective way, with evolution to ensure that costs are optimized;
- Operate with low-latency placement and high-bandwidth processing to support edge drivers;
- Support a plug-and-play modular architecture;
- Meet the security and privacy requirements of applications such as real-time video processing.

The scope of Akraino Edge Stack includes:

- The development of edge solutions to address telco, enterprise and Industrial IoT opportunities;
- The development of edge APIs and frameworks for interoperability with third-party edge providers and hybrid cloud models;
- Collaboration with the upstream community (CI/CD and upstream process support);
- The development of edge middleware, SDKs and applications, along with the creation of an application / VNF ecosystem;
- The creation of blueprints (comprising an integrated stack) to address edge use cases, such as:
    - Telco / hosted edge, for example including ONAP, OpenStack, Airship etc. (medium or large POD);
    - Telco / hosted remote edge, with a stack that scales from a single node to enterprise use cases such as IoT;
    - Over-the-Top (OTT) / enterprise / telecom stack for remote edge locations with thousands of locations and disaggregated hardware;
    - OTT / IIoT / enterprise lightweight stack for low-latency remote edges or IoT gateways.
- Single Pane of Glass control with single-view management of edge resources across 10,000+ sites;
- Thin local control plane, enabling multiple approaches to reducing the local box or data center control plane footprint, for scenarios such as running a control/data plane combined with security measures, running in network switches etc.;

- Edge user / developer APIs, providing edge-agnostic APIs;
- Edge Infrastructure-as-a-Service (IaaS) / Platform-as-a-Service (PaaS) deployments for a wide variety of edge applications;
- Central and/or regional Virtual Infrastructure Manager (VIM) as an alternative to a thin local control plane, with remote orchestration of edge compute resources that enables thin control with only an agent at the edge;
- Edge capabilities like analytics etc.;
- Low-latency provisioning with dynamic microservices enablement;
- Central and/or regional deployments of ONAP, enhanced to support edge scale;
- Cloud-native Network Functions (CNFs) for container- and microservices-based applications.

Prerequisites

The only prerequisites for Akraino Edge Stack relate to hardware setup requirements.

Quick-Start Scripts

An example of how the API would be used to deploy a POD on a bunch of new machines follows. These examples use curl as that is the most compact way to show all of the required headers and data elements required. Of course, any other programming language may also be used.

- Get a login token:
    - The user needs an account with permissions to create nodes, edge sites, PODs and possibly blueprints, if defining a new blueprint). Use the login API as follows (assuming the login is admin and password abc123):

$ curl -v -k -H 'Content-Type: application/YAML' --data '{ name: admin, password: abc123 }'
https://arc.akraino.demo/api/v1/login

    - The API returns a login token (which expires in one hour) in the X-ARC-Token header. This should be passed to all subsequent API calls:

X-ARC-Token: YWRtaW4gICAgICAgIDE1NTY1NTgyMjExMzQ4N2NmMGUwNQ==

- Enumerate the machines:
    - Assuming the machines to deploy on have not yet been made known to the RC, use the Node API to add them to the RC database.
    - Do this with the following API call, once per node:

$ YAML='{

name: nodename,

description: a description of the node,

hardware: <hardware profile UUID>

}'

$ curl -v -k -H 'Content-Type: application/YAML' -H 'X-ARC-Token:
*YWRtaW4gICAgICAgIDE1NTY1NTgyMjExMzQ4N2NmMGUwNQ==' *

--data "$YAML" https://arc.akraino.demo/api/v1/node

    - Keep track of the UUID of each node that is returned from the API.

- Create an edge site:
    - Once the nodes are defined, create an edge site (a cluster of nodes):

$ YAML='{

name: edgesitename,

description: description of the edge site,

nodes: [<list of node UUIDs>],

regions: [<list of region UUIDs>]

}'

*$ curl -v -k -H 'Content-Type: application/YAML' -H 'X-ARC-Token: YWRtaW4gICAgICAgIDE1NTY1NTgyMjExMzQ4N2NmMGUwNQ==' *

--data "$YAML" https://arc.akraino.demo/api/v1/edgesite

- Create and verify the blueprint:
    - To get the UUID of the blueprint, determine which blueprints are installed:

$ curl -v -k -H 'Accept: application/YAML' -H 'X-ARC-Token: YWRtaW4gICAgICAgIDE1NTY1NTgyMjExMzQ4N2NmMGUwNQ==' https://arc.akraino.demo/api/v1/blueprint

    - If the required blueprint is missing, create it:

$ YAML='{

blueprint: 1.0.0,

name: my new blueprint,

version: 1.0.0,

description: description of the blueprint,

YAML: ....

}'

*$ curl -v -k -H 'Content-Type: application/YAML' -H 'X-ARC-Token: YWRtaW4gICAgICAgIDE1NTY1NTgyMjExMzQ4N2NmMGUwNQ==' *

--data "$YAML" https://arc.akraino.demo/api/v1/blueprint

- Start the deployment:
    - Create a POD:

$ YAML='{

name: my new POD,

description: description of this POD,

blueprint: 827cfe84-2e28-11e9-bb34-0017f20dbff8,

edgesite: 2d3533e4-3dcb-11e9-9533-87ac04f6a7e6

}'

*$ curl -v -k -H 'Content-Type: application/YAML' -H 'X-ARC-Token: YWRtaW4gICAgICAgIDE1NTY1NTgyMjExMzQ4N2NmMGUwNQ==' *

--data "$YAML" https://arc.akraino.demo/api/v1/pod

- Make note of the UUID that is returned, which will be needed to monitor the deployment.

- Monitor the deployment:
  - Monitor the POD event URL for the newly created POD, which will return a list of events related to the POD similar to:

$ curl -v -k -H 'Accept: application/YAML' -H 'X-ARC-Token: YWRtaW4gICAgICAgIDE1NTY1NTgyMjExMzQ4N2NmMGUwNQ==' \https://arc.akraino.demo/api/v1/podevent/56b365a0-d6a2-4d12-8f02-e2fc2671573e

events:

- {level: INFO, time: '2019-04-29 18:15:28.0', message: Pod created.}

- {level: INFO, time: '2019-04-29 18:15:28.0', message: 'Starting workflow: create'}

- {level: INFO, time: '2019-04-29 18:15:28.0', message: 'Workflow directory created: $DROOT/workflow/create-56b365a0-d6a2-4d12-8f02-e2fc2671573e'}

- {level: WARN, time: '2019-04-29 18:17:38.0', message: 'Could not fetch the workflow file http://example.com/blueprints/create.py'}

API design

The Akraino Edge Stack API is designed to enrich the application enablement library while meeting enterprise customers' expectation for low latency and data security.

The Mobile Edge Servers (MES) offloading feature is important to smartphone developers, because smartphones have extreme limitations on device resources and energy consumption. This allows only tiny models to run on the device, while large models run on cloud or edge facilities.

An open source Artificial Intelligence (AI) offloading platform, providing Software-as-a-Service (SaaS) services to mobile applications is valuable for many use cases. As a module within the Multi-access Edge Computing (MEC) ecosystem, it enables application developers to test and validate the edge offloading platform from smart devices.

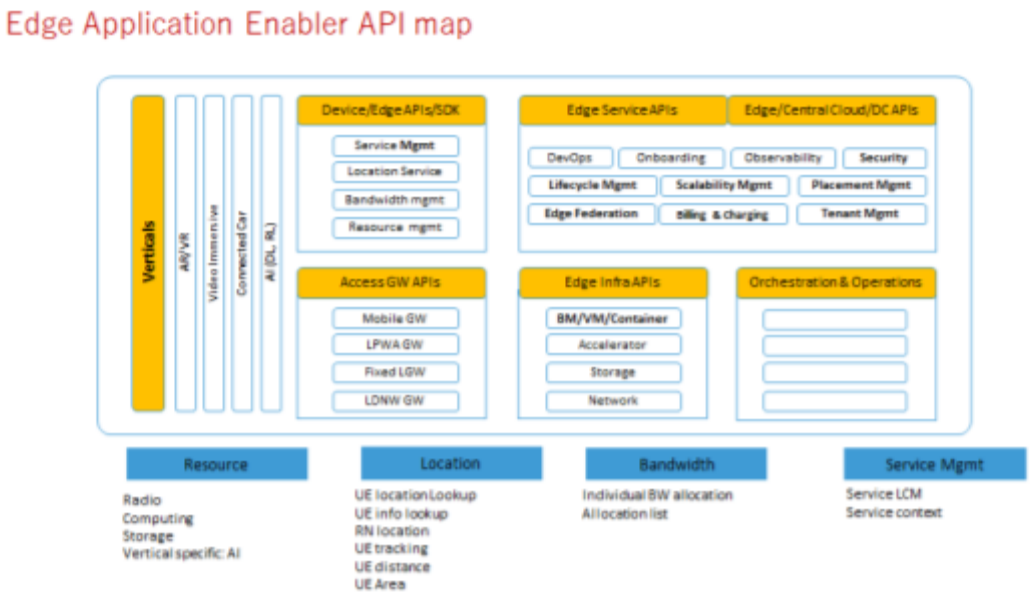


*Figure SEQ Figure * ARABIC 5: Akraino Edge Stack Edge Application API Map.*

Data Model / Schema

The **Regional Controller** maintains an **Object Model**. The object model consists of a number of objects, described below, which model the machines, software and lifecycles of the various Akraino deployments that the Regional Controller controls. The collection of all the objects that the Regional Controller maintains in this model, as well as their relationships, is known as the Akraino Universe. Most objects in the Object Model have the following in common:

- A universally unique identifier (uuid), meant to be a unique, internal identifier for the object, which is assigned by the regional controller;
- A name, meant for human consumption, which is required and will normally be unique for any class of object;
- An optional description, also for human consumption, to provide more detail about the object.

A **Hardware Profile** describes a specific model of hardware that will be used within an Akraino Universe (for example Dell PowerEdge R740 Gen 14). Because the amount and type of information that can describe a particular piece of hardware can vary widely, a Hardware Profile provides a YAML attribute where this information can be stored.

A **Node** is single, identifiable machine. A Node may be unassigned, or it may belong to one and only one edge site. A Node must make reference to a hardware profile that describes the hardware on which the Node is based. In addition, each Node has a YAML attribute, which can be used to store other information about the Node (e.g. the Node's latitude / longitude location for use in a GUI, or the rack ID of the rack the Node is mounted on).

An **Edge Site** is a collection of one or more nodes upon which a POD may be installed. Any specific Node may belong to one and only one Edge Site. Edge Sites in turn must belong to at least one, and potentially several regions. There is no requirement that an Edge Site consist of homogeneous hardware. However, most blueprints are likely to require that all Nodes in an Edge Site be the same hardware and this would be detected at the point in time where a POD is created (see below).

A **Region** is a grouping mechanism to collect one or more edge sites, or other regions, together for management purposes. Regions may be used to group edge sites together by physical location (e.g. US Northeast). They may also be used to perform logical groupings (e.g. RAN sites). Regions form a tree, similar to a UNIX filesystem, with the topmost region being the Universal Region. A Region may have only one parent Region. The Universal Region has itself as its parent.

A **Blueprint** is a collection of software that can be deployed to an empty edge site, as well as the supporting software that the Regional Controller uses to manage the lifecycle of the edge site. A Blueprint in the Object Model consists of:

- A version attribute, which is a string in semantic versioning (X.X.X) form, describing the version of the Blueprint. This is required.
- A YAML attribute describing everything else. This is required. This is very lightly defined at the moment, but is expected to include:
    - Locations (URLs) of various pieces of software required to deploy the Blueprint;
    - Locations of downloadable workflows to run inside the Regional Controller;
    - Definitions of any hardware requirements for a specific Blueprint;

A **Point of Delivery** (POD) is a specific deployment of a Blueprint on an Edge Site. The act of creating a POD causes the Blueprint to be deployed on the Edge Site. The Blueprint must be valid for the Hardware Profiles and other characteristics of the Edge Site, in order to be deployed. In addition, the Edge Site must not be in use by another POD at the time this

POD is created. At the time of creation, the YAML that is uploaded by the user is verified against the schema that is specified in the Blueprint. The Regional Controller then tells the **Workflow Engine** to deploy the Blueprint on the edge site using the following parameters to control the deployment:

- The UUID of the newly created POD;
- The Blueprint;
- The definition of the Edge Site in the database;
- The YAML content from the POST request;
- Any other parameters from the POST request.

Updates to an existing POD are performed via PUT requests to the POD's URL. There is a separate section in the Blueprint specifying the input schema, workflow, and data file components required for each type of update.

Deleting a POD causes the Blueprint to be removed from the edge site and places the edge site back in an unused state.

If any operation (e.g. create, update, delete) is missing from the specification in the Blueprint, the corresponding operation is disallowed by the Regional Controller. Naturally, if create is missing, then the Blueprint can never be deployed.

A **User** is an individual user of the Regional Controller. A user is identified by a user name, a password and a list of roles. All Regional Controller API operations are logged with an indication of the User who requested the operation. Because the User database is likely to be maintained externally (e.g. in an LDAP server shared with other services), there is no API to perform the Create, Read, Update and Delete (CRUD) operations on Users.

A **Session** is an authenticated instance of a User connection to the Regional Controller. There may be many sessions for one User and Sessions have a limited lifetime. Almost all operations within the API require a Session token, which identifies the User and the User's roles to the API. As such, the very first operation a User of the API will perform will be the Create Session (POST /api/v1/login) call in the login API. This creates the Session and its corresponding token.

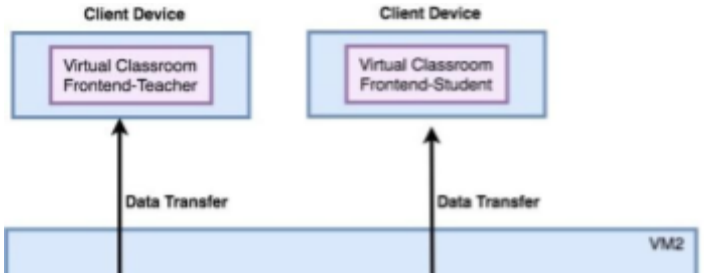
A **Role** is a set of functionalities that can be assigned to one or more Users. Roles allow Users to perform specific functions within the API. The roles are hard-coded into the Regional Controller and are not expected to change often, if at all; as such, there are no CRUD operations defined for the roles. A User of the API can discover what roles they have been given via the login API.

Case Studies

This section provides details of both customer-specific and generic use cases for Akraino Edge Stack.

Baidu has deployed the AI Edge: School/Education Video Security Monitoring blueprint. With this blueprint, teachers and school authorities could conduct a full evaluation of the overall class and the concentration of individual students, helping to fully understand the real time teaching situation. The blueprint has been deployed in Beijing, Shanghai, Hangzhou (Zhejiang Province) and many other cities in China. The AI Edge Blueprint mainly focuses on establishing an open source MEC platform combined with AI capacities at the edge, which can be used for safety, security and surveillance.

Tencent has deployed the Integrated Edge Cloud Type 4: AR/VR-oriented Edge Stack (IEC) blueprint. IEC Type 4 is focused on AR and VR applications at the edge. Tencent focused on building edge infrastructure and a Virtual Classroom application. Virtual classroom is a basic application that provides a virtual reality experience simulating a classroom with teachers and students.



Tencent has also deployed the Connected Vehicle blueprint (CVB). As a MEC platform for connecting vehicles, CVB is used to obtain traffic and/or vehicle information, then dispatch the traffic/vehicle information to the corresponding edge process unit. The data is processed at the edge of in the cloud to determine the suggested action, which is then sent to the vehicle's driver. Examples of these actions include changing lane ahead of a narrow street, avoiding driving the wrong way down a one-way street and avoiding the carpool lane when there's no passenger in the vehicle.

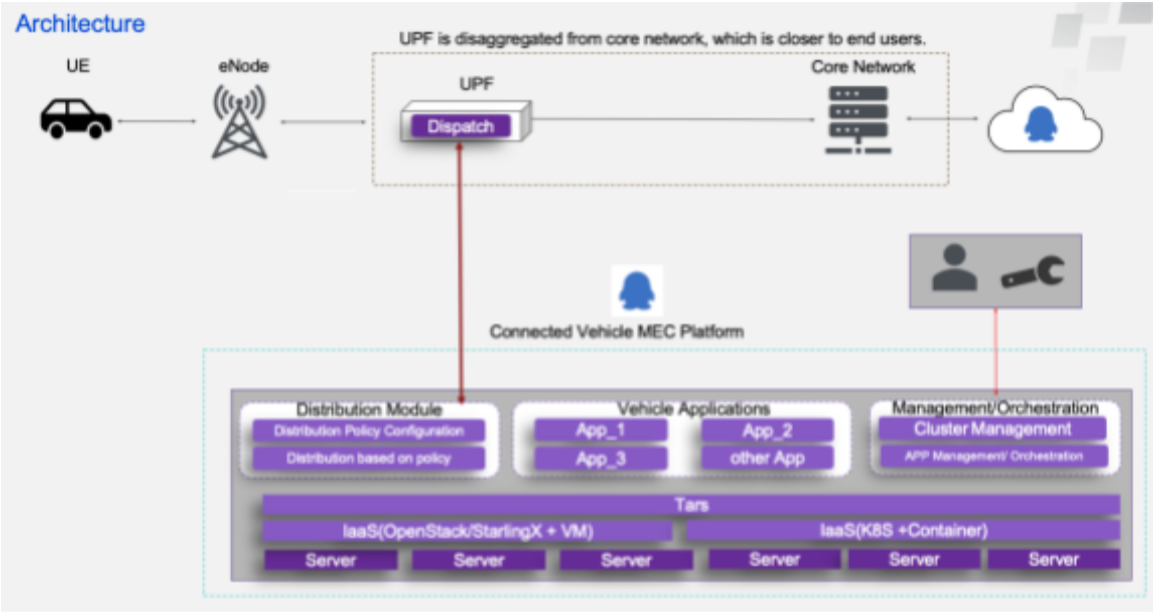


*Figure SEQ Figure * ARABIC 7: Tencent's Connected Vehicle Architecture.*

Video surveillance is an essential function for many emerging use cases, such as smart cities, smart transportation, smart homes etc. The objective of a video surveillance system is to detect objects and events through networked static or mobile cameras, with low latency being a critical requirement. Low latency is a requirement for most of these systems, such as automatic fire detection. In many surveillance systems, mobile cameras are used because of their flexibility. Due to their limited computation power, mobile cameras need to transmit the captured video to remote servers for further processing and analysis. Since transmitting the video to remote servers causes unpredictable delay, it is desirable to deliver the video stream to local edge nodes and process it there. The Edge Video Processing blueprint provides an ideal framework for video surveillance, utilizing the computation and communication capacity of distributed edge nodes to provide scalable, reliable and real-time services.

Reference Links

[1] Acumos AI.

[2] Akraino Edge Stack.

[3] Airship.

[4] Ceph.

[5] DANOS.

[6] EdgeX Foundry.

[7] Kubernetes.

[8] LF Networking.

[9] ONAP.

[10] OpenStack Ocata.

[11] StarlingX.

Appendix B **Baetyl**

Introduction

Baetyl (pronounced “Beetle”) is a general-purpose platform for edge computing that manipulates different types of hardware facilities and device capabilities into a standardized container runtime environment and API, enabling efficient management of application, service and data flow through a remote console both in the cloud and on premise.

Baetyl seamlessly extends cloud computing, data and services to edge devices. It can provide temporary offline, low-latency computing services, including device connect, message routing, remote synchronization, function computing, video access pre-processing, AI inference and device resources reporting. The combination of Baetyl and the Cloud Management Suite of Baidu Intelligent Edge achieves cloud management and application distribution, enabling applications running on edge devices and addressing a range of edge computing scenarios.

Baetyl is architected as multiple modules, ensuring that each is implemented as a separate, independent module. Baetyl leverages containerization to create images, ensuring a consistent run-time environment across platforms. Baetyl isolates and limits container resources, accurately allocating CPU, memory and other resources for each running instance to improve the efficiency of resource utilization.

Architecture Description

The key concepts of the Baetyl architecture include:

- **System** refers to the Baetyl system, including Master, Service, Volume and System resources.
- **Master** refers to the core part of Baetyl, responsible for managing Volume, Service, built-in Engine, external RESTful API and command line.
- **Module** provides an executable package for Service, such as a Docker image, to launch instances of Service.
- **Service** refers to a set of running programs that managed by Baetyl to provide specific functions such as message routing services, function computing services, micro-services etc.
- **Instance** refers to the specific running program or container launched by the Service. A Service can start multiple instances or can be dynamically started by other services. For example, the instances of function runtime service are dynamically started and stopped by the function manager service.
- **Volume** refers to the directory used by the Service and can be either a read-only directory, such as a directory for placing resources such as configuration, certificates, scripts etc. or a writable directory to persist data, such as logs and database.
- **Engine** refers to the operational abstractions and concrete implementations of the various running modes of the Service, such as the docker container mode and the native process mode.
- **Services and System Relationships**: Baetyl systems can start multiple Services and there is no dependency between Services, though their startup order should not be assumed (although it is currently started sequentially). All data generated by the Service at runtime is temporary and will be deleted when the Service is stopped, unless it is mapped to a persistent directory. The program in the Service may stop for various reasons and the Service will restart the program according to the user’s configuration. This situation is not equivalent to stopping the Service, so the temporary data will not be deleted.

The key components of a complete Baetyl system are the Master, Service, Volume and system resources. The Master loads all modules according to the application configuration, to start the corresponding services, and a Service can start

several instances, all of which are managed and supervised by the Master. Note that the instances of the same Service share the storage Volume bound to the Service. Therefore, if an exclusive resource exists, such as listening to the same port, only one instance can be successfully started.

The current Baetyl architecture includes the following modules:

- **baetyl-agent** provides cloud agent service for status reporting and over-the-air (OTA) application updates.
- **baetyl-hub** provides a Message Queuing Telemetry Transport (MQTT)-based message routing service.
- **baetyl-remote-mqtt** provides a bridge services for synchronizing messages between hub and remote MQTT services.
- **baetyl-function-manager** provides function services for function instance management and message-triggered function calls.
- **baetyl-function-python27** provides a Google Remote Procedure Call (GRPC) micro-service that loads Python scripts based on a Python 2.7 runtime that can be managed by baetyl-function-manager as a function instance provider.
- **baetyl-function-python36** provides a GRPC micro-service that loads Python scripts based on a Python 3.6 runtime that can be managed by baetyl-function-manager as a function instance provider.
- **baetyl-function-node85** provides a GRPC micro-service that loads JavaScripts based on a Node 8.5 runtime that can be managed by baetyl-function-manager as a function instance provider.

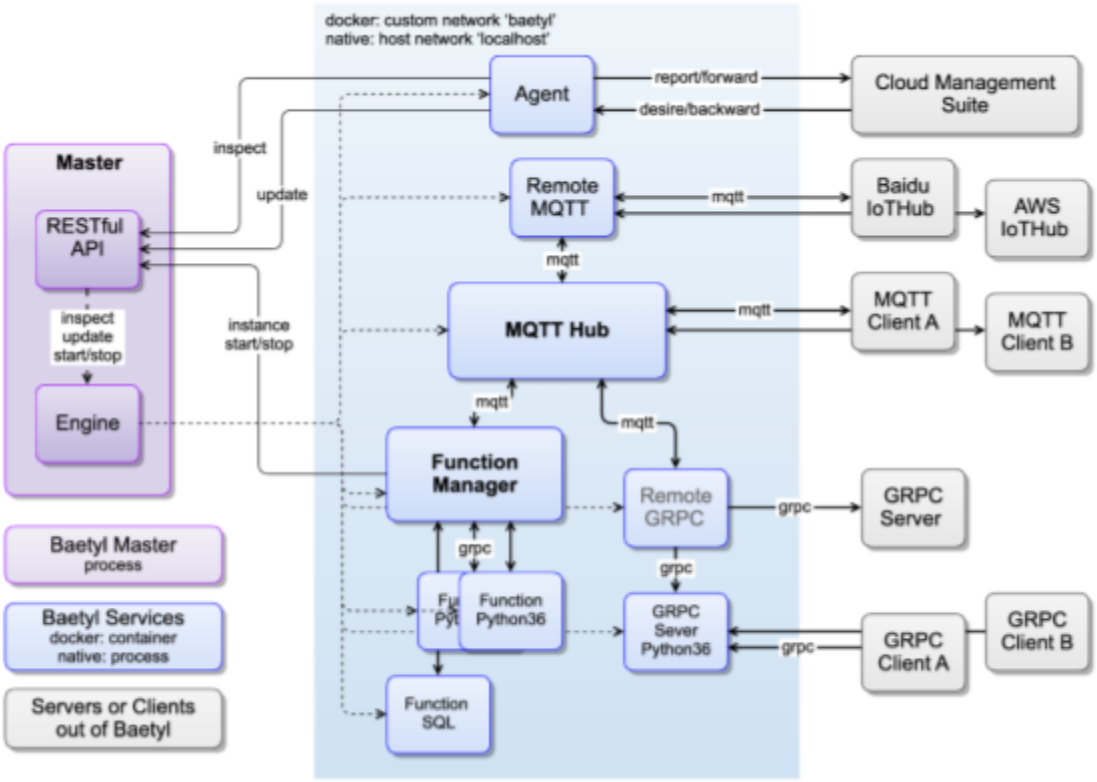


*Figure SEQ Figure * ARABIC 8: Baetyl Architecture Diagram.*

Framework

Baetyl provides a standalone framework. It supports Linux as a systemd service with Docker pre-installed, macOS as a launchd service with Docker pre-installed and Windows as a system service with both Windows Container on Windows and Linux Container on Windows.

Features

Built-in services of Baetyl include:

- **Baetyl Master** is responsible for the management of service instances, such as start, stop, supervise etc., consisting of the Engine, API and Command Line. It supports two modes for running services, the native process mode and the Docker container mode.
- The official module **baetyl-agent** is responsible for communication with the BIE cloud management suite, which can be used for application delivery, device information reporting etc. Mandatory certificate authentication ensures transmission security.
- The official module **baetyl-hub** provides message subscription and publishing functions based on the MQTT protocol, and supports four access methods, TCP, SSL, WS, and WSS.
- The official module **baetyl-remote-mqtt** is used to bridge two MQTT Servers for message synchronization and supports configuration of multiple message route rules.
- The official module **baetyl-function-manager** provides computing power based on the MQTT message mechanism, flexible, high availability, good scalability and fast response.
- The official module **baetyl-function-python27** provides the Python 2.7 function runtime, which can be dynamically started by baetyl-function-manager.
- The official module **baetyl-function-python36** provides the Python 3.6 function runtime, which can be dynamically started by baetyl-function-manager.
- The official module **baetyl-function-node85** provides the Node 8.5 function runtime, which can be dynamically started by baetyl-function-manager.
- The **SDK** (Golang) can be used to develop custom modules.

Prerequisites

The key prerequisites for Baetyl are Docker, a computer and an Internet connection.

Quick-Start Scripts

Quick install information for Baetyl is provided here and key steps include:

- Install docker: *curl -sSL https://get.docker.com | sh*
- Install Baetyl: *curl -sSL http://dl.baetyl.io/install.sh | sudo -E bash -*
- Start Baetyl: *sudo systemctl start baetyl.service*
- Check Baetyl services: *docker stats*

API design

The Remote Management API is still in development and will provide the following features:

- Collection and reporting of local runtime information;
- Notification of service changes across an unstable network connection;
- Joining of multiple standalone edge nodes as a cluster.

Data Model / Schema

Connection and devices:

- Baetyl supports the full MQTT protocol for device communication.
- Baetyl can proxy MQTT messages between local devices and remote servers.

Computation:

- Baetyl can link a JavaScript Object Notation (JSON) payload from MQTT with local functions.
- Baetyl can decode and sample a Real Time Streaming Protocol (RTSP) video streaming from cameras to local functions.

Storage:

- Baetyl provides a local key-value persistent database for all services.

Reference Links

[1] Baetyl code.

[2] Documentation.

[3] Mailing list.

[4] Slack channel.

Appendix C **EdgeX Foundry**

Introduction

EDGE X FOUNDRY™ [EdgeX Foundry](#) is a vendor-neutral, loosely-coupled microservices framework that enables flexible, plug-and-play deployments that leverage a growing ecosystem of available third-party offerings or to include proprietary innovations.

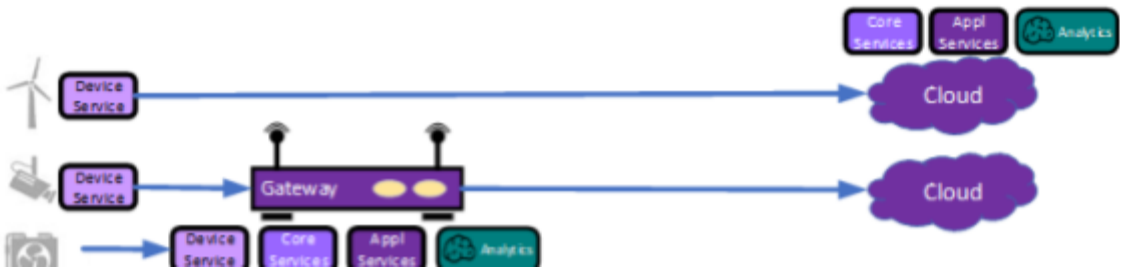
Key project goals are:

- Build and promote EdgeX Foundry as the common open platform unifying edge computing
    - The community builds and maintains common building blocks and APIs;
    - Scope for contributors to add value and ensure a return on their investment;
    - Allow best-of-breed solutions.
- Enable and encourage the rapidly growing community of IoT solution providers to create an ecosystem of interoperable plug-and-play components;
- Certify EdgeX components to ensure interoperability and compatibility
- Provide tools to quickly create EdgeX-based IoT edge solutions;
- Collaborate with relevant open source projects, standards groups and industry alliances to ensure consistency and interoperability across the IoT ecosystem.

Architectural tenets include:

- EdgeX Foundry must be platform-agnostic with regard to hardware, OS, distribution, deployment, protocols and sensors;
- EdgeX Foundry must be extremely flexible;
- Any part of the platform may be upgraded, replaced or augmented by other microservices or software components;
- Services can scale up and down based on device capabilities and use cases;
- EdgeX Foundry should provide reference implementation services while encouraging best-of-breed solutions;
- EdgeX Foundry must provide store and forward capability to support disconnected and/or remote edge systems;
- EdgeX Foundry must support and facilitate intelligence moving closer to the edge in order to address concerns around actuation latency, bandwidth, storage and remote operation;
- EdgeX Foundry must support brownfield and greenfield device and sensor deployments;
- EdgeX Foundry must be secure and easily managed.

In today's IoT landscape, it is imperative to leverage compute, storage, network resources wherever they are located. The loosely-coupled architecture of EdgeX Foundry enables its distribution across nodes, enabling tiered edge and fog computing. The scope of the project includes embedded sensors for controllers, edge gateways and servers. The quantity and function of microservices deployed on a given node depends both on the use case and on the capability of the edge hardware.



Architecture Description

EdgeX Foundry is architected as a collection of over a dozen microservices, written in multiple languages such as Go, C, Java and others.

The data flow of EdgeX Foundry is as follows:

- Sensor data is collected by a device service from a thing;
- Data is passed to the core services to ensure local persistence;
- Data is then passed to application services for transformation, formatting and filtering, then be sent “North” to enterprise or cloud systems;
- Data is then available for edge analysis and can trigger device actuation through the command service;
- Other services provide supporting capabilities that drive this flow.

Using REST communications, services exchange data as necessary via the message bus, for example core data shared with export services and the rules engine.

EdgeX Foundry does not dictate deployment or orchestration solutions, but supports organizational choice. EdgeX Foundry provides microservices via Docker and Docker Compose, as well as Ubuntu Snaps in the app store. EdgeX Foundry has examples of Helm Chart templates.

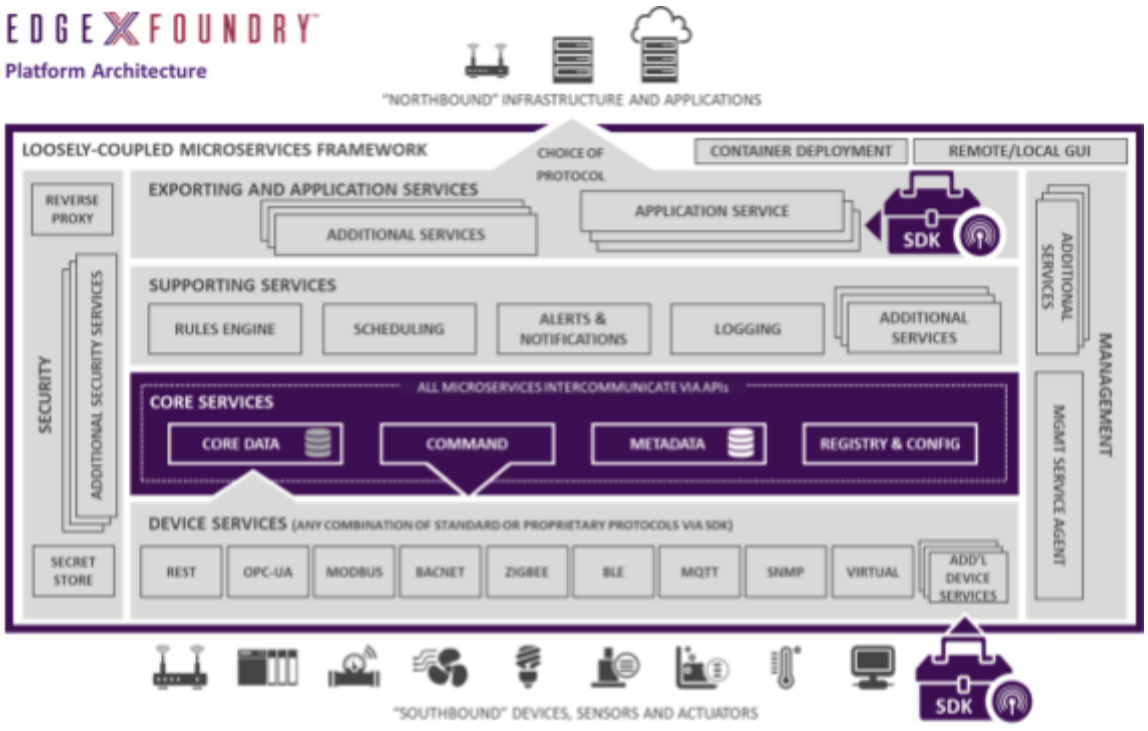


*Figure SEQ Figure * ARABIC 10: EdgeX Foundry Architecture.*

The layers and services of EdgeX Foundry constitute a dual transformation engine:

1. Translating information coming from sensors and devices via hundreds of protocols and thousands of formats into EdgeX Foundry.

2. Delivering data to applications, enterprises and cloud systems over TCP/IP-based protocols in formats and structures determined by the user.

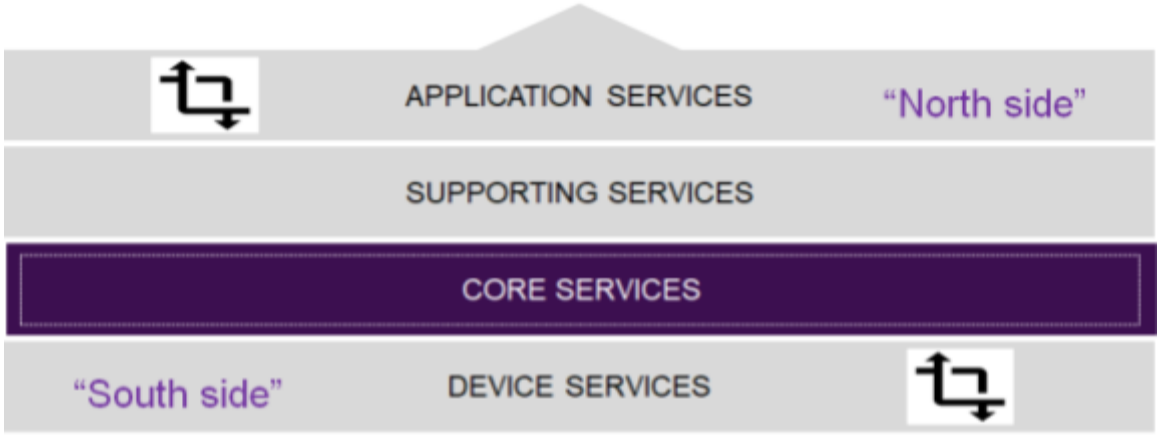


*Figure SEQ Figure * ARABIC 11: EdgeX Foundry Layers.*

Highlights of the technology incorporated in EdgeX Foundry include:

- The majority of the micro services are written in the Go language
    - These were previously written in Java;
    - Some device services are written in C;
    - A user interface is provided in JavaScript;
    - The project adopts a "polyglot" approach, using the language and tools that are optimum for each service.
- Each service communicates via a REST API.
- EdgeX Foundry supports both Redis and MongoDB, ensuring user flexibility for persistent sensor data at the edge
    - The database stores relevant information for the application;
    - This enables alternate persistence storage.
- A message pipe connects core data to export services and/or the rules engine
    - Uses the ZeroMQ open-source universal messaging library by default;
    - Allows the use of MQTT as an alternate, if the broker is provided.
- EdgeX Foundry uses open source technology where appropriate, such as Consul for configuration and registry, Kong for reverse proxy and Drools for rules engine.

Framework

EdgeX Foundry is a microservice-based application capable of being deployed and/or orchestrated using a variety of technologies. For ease of use, it provides some pre-built deployable elements including:

- Docker containers and Docker Compose files;
- Ubuntu Snaps available from the Canonical Snapcraft Snap Store;
- Kubernetes Helm Chart and other Kubernetes code examples.

Features

Documentation on each EdgeX Foundry release is here, with roadmaps for future releases (along with the extended backlog) here.

Prerequisites

A key tenet of the EdgeX Foundry project is to maintain platform independence in order to maximize scalability. This implies independence across processor architectures (x86 or Arm), Operating Systems (Linux, Windows, macOS) and application environments (microservices developed in Java, JavaScript, Python, Go, C/C++ etc. working together through the common APIs). The Getting Started site provides specific information.

To start using EdgeX Foundry with minimal effort, developers download the latest EdgeX Foundry Docker Compose files, and then pull and use the Docker Containers, as discussed in the Getting Started – Users section.

To develop EdgeX Foundry from the base code, developers install the development pre-requisites, pull the edgex-go GitHub repository and use the makefile at the root of the repository to build, test and run the microservices.

Quick-Start Scripts

Quick-Start Scripts are available at the Getting Started site.

API Design

Full API documentation is available at the API Reference site.

A Version 2 API is being designed and developed. It is anticipated this new API will be available with the Hanoi release (v2.0) in late 2020. Some details about this API can be found here.

Data Model / Schema

EdgeX Foundry is based on Events and Readings, which are collected by device services, persisted by core data, then sent to the cloud and other applications by export distribution.

- **Events** are collections of Readings, associated with a device.
- **Readings** represent sensings performed by a device or sensor, structured as a simple Key/Value pair
    - **Key** is a value descriptor, see below;
    - **Value** is the sensed value, for example temperature = 72°.
- An Event must comprise at least one Reading to be valid and a Reading must have an “owning” event.

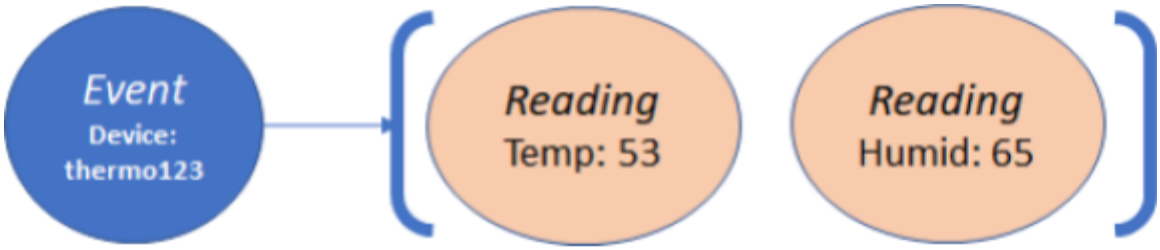


*Figure SEQ Figure * ARABIC 12: EdgeX Foundry Example Events and Readings.*

- A **Value Descriptor** provides context and unit of measure to a Reading, having a unique name, specifying the unit of measure for the Reading value, dictating special rules around the Reading value such as maximum, minimum and default values and describing the display formatting for a Reading.
- The Reading Key is equivalent to the name of the Value Descriptor. In metadata, Devices use Value Descriptors to describe the data they will send, as well as the parameters and results of actuation commands.

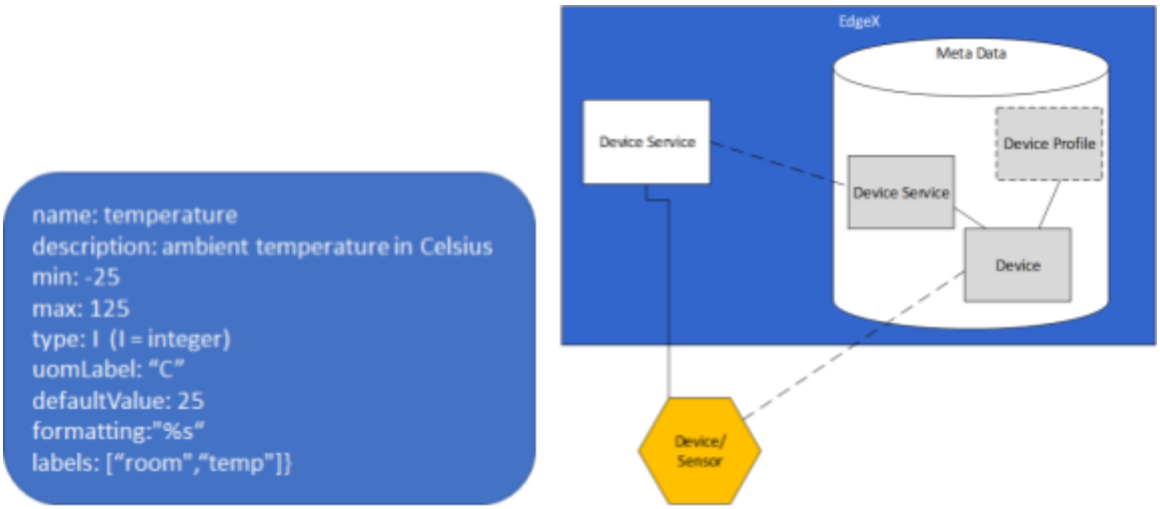


*Figure SEQ Figure * ARABIC 13: EdgeX Foundry Interfaces.*

- A **Device Profile** is a template of a type or classification of device, providing general characteristics for the types of data a device sends and the types of commands or actions that can be sent to the device. For example, a Building Automation and Control networks (BACnet) thermostat device profile would provide general characteristics of thermostats, specifically those communicating via the BACnet protocol, describing the types of data a BACnet thermostat sends such as current temperature (as a float and in Celsius) and current humidity (as a float and a percentage). It would also list what the it responds to and how to send those commands, such as *Get or Set the cooling set point* (passing a float as a parameter) or *Get or Set the heating set point* (passing a float as a parameter).

Reference Links

[1] EdgeX Foundry code.

[2] Technical documentation.

[3] Technical video tutorials.

[4] EdgeX Foundry blog:

[5] Mailing list.

[6] Slack channel.

Appendix D **Fledge**

Introduction

Fledge is a framework for the industrial edge focused on critical operations, predictive maintenance, situational awareness and safety. Fledge is architected to integrate IIoT, sensors and modern machines with the cloud as well as with existing legacy systems.

Edge computing is a challenging distributed computing problem. The fragmentation and distribution of industrial data, networking, processing, security and storage makes managing it complicated. Simplifying industrial IoT application and system development with a ubiquitous open source stack, standards, and community is our mission. Fledge edge services include:

- Collect data from any or all sensors;
- Aggregate, combine and/or organize data;
- Perform edge-based alerting, anomaly detection and/or ML (TensorFlow Lite, OpenVino);
- Transform and/or filter data in flight;
- Buffer data;
- Analyze and/or visualize edge data;
- Deliver data to multiple local and/or cloud destinations.

Architecture Description

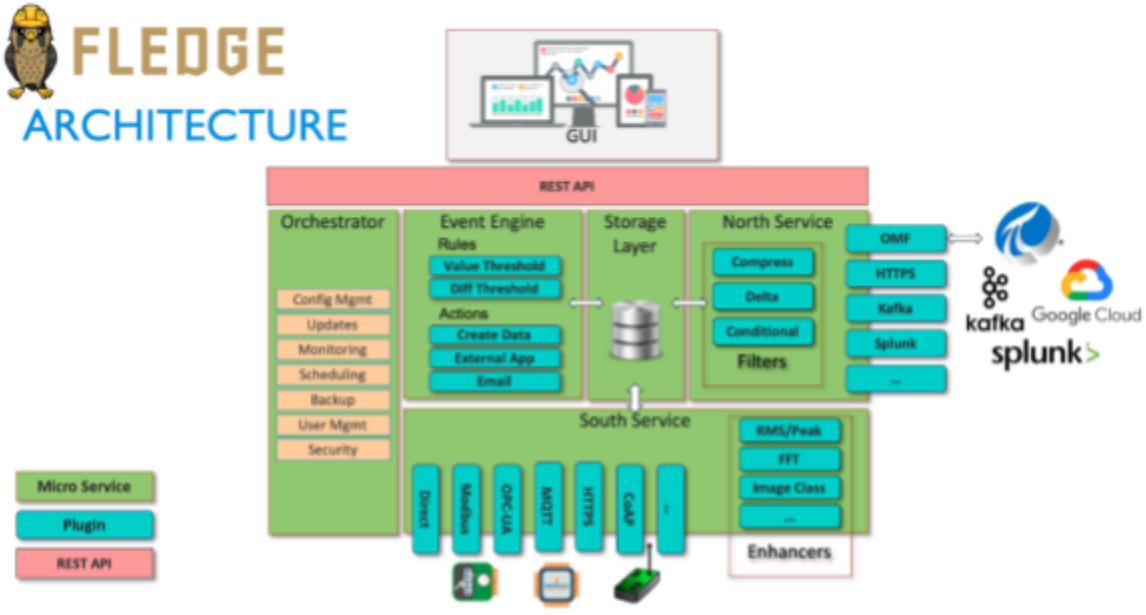


*Figure SEQ Figure * ARABIC 14: Fledge Architecture.*

The **Fledge Core** coordinates all Fledge operations. Only one Core service can be active at any time. Core functionality includes scheduling, configuration, monitoring, data APIs backup and restore, auditing, certificate storage, user management and asset data.

The **Storage Layer** provides two principal functions: (1) maintenance of Fledge configuration and run-time state, and (2) storage and/or buffering of asset data. The type of storage engine is pluggable, so in installations with a small footprint a plugin for SQLite may be chosen, whereas in installations with a high number of concurrent requests and larger footprint Postgresql may be suitable. In micro installations, for example on edge devices, in-memory temporary storage may be the best option.

Southbound Microservices perform communication of data and metadata between Edge devices such as sensors, actuators or PLCs and Fledge. Smaller systems may have this service installed on edge devices. Southbound components are typically deployed as always-running services, which continuously wait for new data. Alternatively, they can be deployed as single-shot tasks, which periodically spin up, collect data and spin down.

Northbound Microservices offer bi-directional communication of data and metadata between the Fledge platform and larger systems located locally or in the cloud. Larger systems may be public or private cloud data services, proprietary solutions or Fledge instances with larger footprints. Northbound components are typically deployed as one-shot tasks, which periodically spin up and send data which has been batched, then spin down. However, they can also be deployed as continually-running services.

Filters are plugins which modify streams of data that flow through Fledge. They can be deployed at ingress (in a South service) or at egress (in a North service). Typically, ingress filters are used to transform or enrich data, while egress filters are used to reduce flow to northbound pipes and infrastructure, for example by compressing or reducing data that flows out. Multiple filters can be applied in pipelines, and once configured, pipelines can be applied to multiple North or South services.

The **Event Engine** maintains zero or more rule/action pairs. Each rule subscribes to desired asset data and evaluates it. If the rule triggers, its associated action is executed.

The Fledge **REST API** provides methods to administer Fledge and to interact with the data inside it.

The **Graphical User Interface** (GUI) enables the administration of Fledge. All GUI capability is through the REST API, so Fledge can also be administered through scripts or other management tools. The GUI provides:

- **Health**: determine whether services are responsive and observe data that's flowed in and out of Fledge;
- **Assets and Readings**: analytics of data in Fledge;
- **North**: manage North services;
- **South**: manage South services;
- **Notifications**: manage event engine rules and delivery mechanisms;
- **Certificate Store**: manage certificates;
- **Backup and Restore**: backup and restore Fledge;
- **Logs**: see system, notification and audit logging information.

Framework

Fledge is in production or evaluation in many dozens of real-world industrial sites. It can be deployed and utilized via many technologies:

- Deployment scenarios including bare OS, Docker containers, Cisco containers, VMs and via the Linux Foundation Project Eve;
- Operating Systems including Debian (.deb), Mendel (.deb), Ubuntu (.deb), Red Hat Enterprise Linux (.rpm), CentOS (.rpm), Raspbian Stretch (.deb) and Raspbian Buster (.deb);
- Fledge has also been compiled to run on proprietary customer hardware platforms including arm7l, aarch64 and x86_64.

Quick-Start Scripts

Scripts are available at the Fledge Quick Start Guide.

API design

The REST API provides all user and program interaction to configure, monitor and manage the Fledge system. It supports:

- The complete configuration of the Fledge appliance;
- Access to monitoring statistics for the Fledge appliance;
- User and role management for access to the API;
- Access to the data buffer contents.

Full details are available at the Fledge REST API Developer's Guide.

Data Model / Schema

Fledge translates from a variety of schemas on the southbound side to others on the northbound side.

Case Studies

Some real-life use cases deployed in production include:

- Thermal imaging;
- TensorFlow edge ML and AI for industrial applications;
- JEA uses Fledge to monitor oil pumps, oil temperature, fans, ambient air temperature and hydrogen gas levels on substation transformers. This condition-based IIoT system avoids using expensive SCADA hardware by leveraging Cisco switches. Fledge then integrates the sensor data into JEA's historians and maintenance systems.

- The composite materials used in military drone require specific temperature and humidity conditions during painting and curing. Entire aircraft may need to be reworked if conditions are improper. Using Fledge, Dwyer TTE Explosion-Proof sensors were deployed in six locations as well as a paint booth. Fledge readings and alerts informed operational control systems and historians as well as large monitors authorizing processes on each assembly line.

Reference Links

[1] Fledge code.

[2] Documentation.

[3] Project page.

[4] Mailing list.

Appendix E **Home Edge**

Introduction

Home Edge is a robust, reliable and intelligent home edge computing open source framework, platform and ecosystem. It provides an interoperable, flexible and scalable edge computing services platform with APIs that can also be used with libraries and runtimes.

The scope of the project includes software development under an OSI-approved open source license, comprising documentation, testing, integration and the creation of other artifacts that aid in development, deployment, operation and adoption.

All the devices (TVs, fridges, washing machines etc.) connected into a Home Edge Network are considered Home Edge devices. Tasks performed on devices on a Home Edge Network are managed by Home Edge Orchestrator software. They are assigned to specific Home Edge Nodes. Home Edge Orchestrator constantly scans the Home Edge Network, forming lists of attached devices and collecting performance ratings for them. Performance ratings are the basis for deciding on which devices Home Edge Applications are to be run. If the Home Edge Orchestrator cannot find a device with rating higher than its own device rating, it will start the Home Edge Application locally.

Home Edge Networks support distributed applications consisting of interacting Docker container instances. Docker containers ensure quick deployment, easy management, safety and hardware independence.

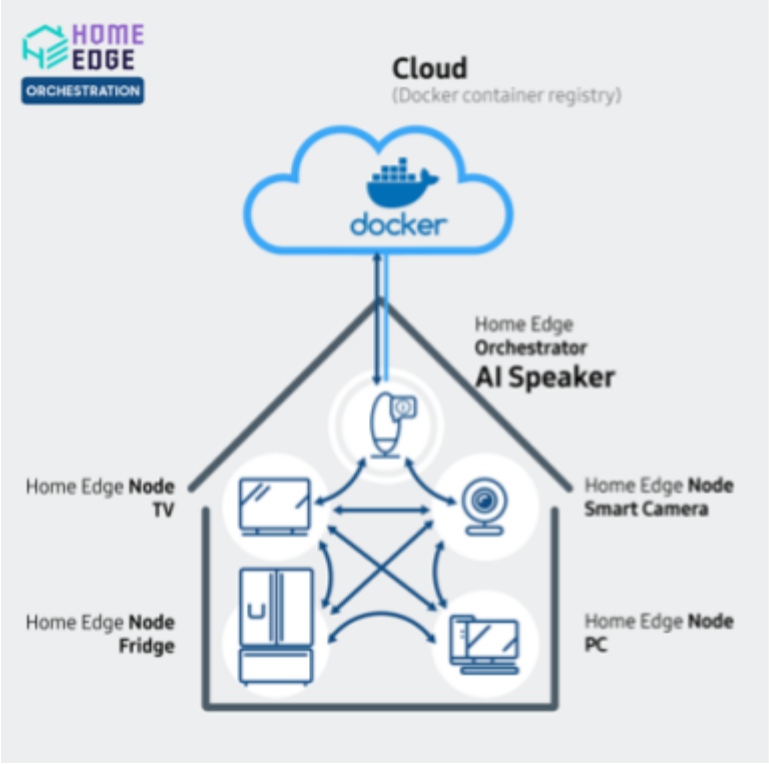


*Figure SEQ Figure * ARABIC 16: Home Edge Overview.*

Architecture Description

Home Edge is focused on driving and enabling a robust, reliable and intelligent home edge computing open source framework, platform and ecosystem running on a variety of devices present in daily home lives. To accelerate the

deployment of the edge computing services ecosystem, Home Edge provide users with an interoperable, flexible and scalable edge computing services platform with a set of APIs that can also run with libraries and runtimes.

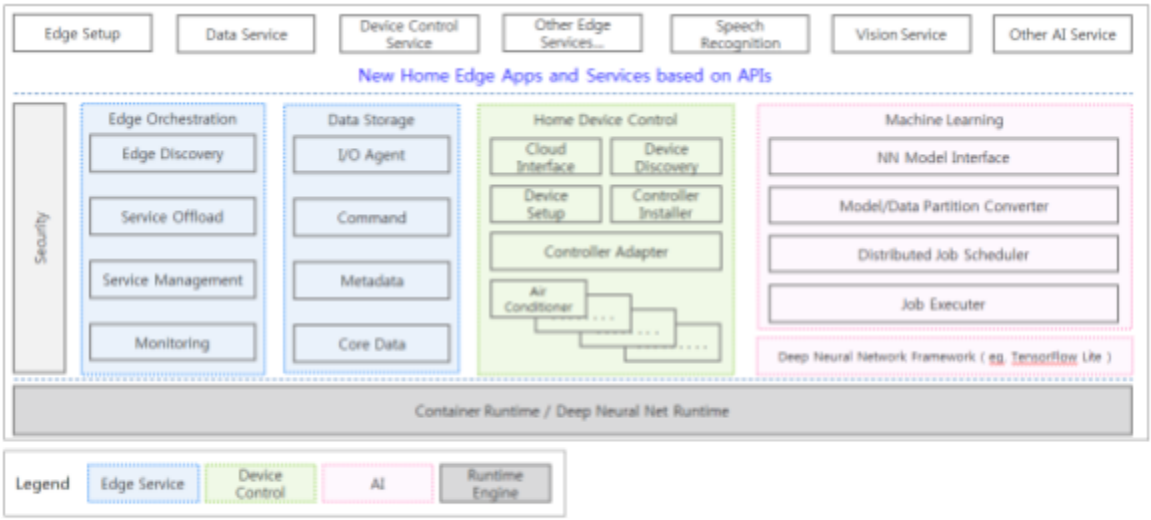


*Figure SEQ Figure * ARABIC 17: Home Edge Architecture.*

The platform architecture of the project comprises multiple modules:

- **Edge Orchestration Module:**
  - **Edge Discovery**: finds the Home Edge devices in a user's home network.
  - **Service Offloading**: re-deploys services to another Home Edge device to achieve load balancing of the device and/or service.
  - **Edge Setup**: configures the network information and the user's profile on the Home Edge device.
  - **Service Management**: manages the lifecycle and replicas of services.
  - **Monitoring**: checks and notifies the status of the Home Edge devices, as well as their connected home devices and services in a user's home network.
- **Data Storage Module:**
  - **Core Data**: provides persistent storage on the Home Edge device for data collected from its connected home devices (consumer electronics, sensors, things) and services.
  - **Metadata**: provides information about the identification, profile and/or data of the connected home devices (consumer electronics, sensors, things) and services.
  - **I/O Agent**: provides an API for accessing the data storage.
  - **Command**: provides a method for retrieving specific data from the data storage of the Home Edge device by a rule-based approach.
- **Home Device Control Module:**
  - **Cloud Interface**: provides the interface between cloud services and each home device in a user's home network.
  - **Device Discovery**: finds the connected home devices and when a connected home device is found for the first time, its client service is installed by the Controller Installer.

    - **Device Setup**: configures the required network information on a connected home device, enabling it to join a user's home network.
    - **Controller Installer**: installs the client service on a discovered connected home device.
    - **Controller Adapter**: provides unified APIs for the control of the connected home devices.
    - **Home Device Client**: controls the connected home devices (e.g. refrigerator, washing machine, light bulb, door lock, HVAC system and temperature sensor).
- **Machine Learning Module:**
    - **Neural Network Model Interface**: provides APIs for inference and recognition using (distributed) neural network processing.
    - **Model Partition Converter**: divides the model for distributed neural network processing.
    - **Distributed Job Scheduler**: schedules and allocates distributed jobs.
    - **Job Executer**: executes a distributed job on the Home Edge device using the Deep Neural Network Framework.
- **Security Module**: provides security features applicable to the Home Edge devices and services, such as secure on-boarding, certificate, AAA, encryption/decryption of the message protocols etc.
- **Deep Neural Network Framework**: provides the dataflow programming framework for ML, such as TensorFlow Lite.

Framework

Home Edge provides a standalone framework.

Features

The scope of the project includes software development under an OSI-approved open source license, comprising documentation, testing, integration and the creation of other artifacts that aid in development, deployment, operation and adoption.

Prerequisites

- Build prerequisites include:
    - Docker-ce version 17.06 (or above), noting that for the execution of docker commands with non-root privileges $USER must be added to the Docker group:

    $ sudo usermod -aG docker $USER

    - Go compiler version 1.10 (or above), noting that to build the Edge Orchestrator from Go sources, the GOPATH environment variable must be set correctly:

    $ export GOPATH=$HOME/go

    $ export PATH=$PATH:$GOPATH/bin

    - Glide, for which the appropriate repository must be added to the Apt list to enable installation:

    sudo add-apt-repository -y ppa:masterminds/glide && sudo apt-get update

    sudo apt-get install -y glide
- Running prerequisites include:

- Two or more devices with Ubuntu 14.04 (or above) and Docker 17.09 (or above);
- All devices connected to the same WIFI network;
- Same authentication key in the /var/edge-orchestration/user/orchestration_userID.txt file.

Quick-Start Scripts

This section provides information about how to download and run the pre-built Docker image without building the project:

- Install Docker-ce Version 17.09 (or above), following the installation steps.
- Download the Docker image .tar file.
- Load the Docker image from the .tar file using *$ docker load -i edge-orchestration.tar*.
- Add the key file
    - In order for Edge Orchestration devices to communicate with each other, each devices should have the same authentication key in /var/edge-orchestration/data/cert/edge-orchestration.key

  $ sudo cp {SampleKey.key} /var/edge-orchestration/data/cert/edge-orchestration.key
    - Any certificate file can be authentication key.
- Run Edge Orchestration with the Docker image

```
$ docker run -it -d \
        --privileged \
        --network="host" \
        --name edge-orchestration \
        -v /var/edge-orchestration/:/var/edge-orchestration/:rw \
        -v /var/run/docker.sock:/var/run/docker.sock:rw \
        -v /proc/:/process/:ro \
        edge-orchestration:baobab
```

Data Model / Schema

Home Edge does not store any data. During 2020, Home Edge is collaborating with EdgeX Foundry on the Data Storage Module.

Case Studies

These following two use cases for Home Edge were demonstrated at the SWC IOT Conference in Barcelona 2019.

The **Intrusion Detection at Edge** demo scenario showcased an edge-based, intelligent home intrusion and camera-tamper detection system that could identify events where intruders tried to block the camera's view. Any suspicious feed would be clipped to a ten-second video and a ML service would be requested, using the Edge Orchestration service. The Edge Orchestration service would locate all instances of the requested service on any of the home's connected devices, triggering the requested service on one of the devices based on a performance evaluation matrix sent by every device. The event would then be classified as dangerous or normal and in case of a dangerous event

the Edge Orchestration service would request the notification service to trigger an alarm and send the video clip to the registered devices.

The **Video Conversion to Supporting Format at Edge** demo scenario showcased an edge-based video conversion to formats supported by TV. When a TV received a video but was unable to play it because of the format, the Edge Orchestration service would locate all instances of the requested service on any of the home's connected devices. The Edge Orchestration service would trigger the requested conversion service on one of the devices based on a performance evaluation matrix sent by every device. The conversion service would take the file from the media server, convert it into a TV-compatible format and inform the TV, which could then fetch the converted video and play it.

Reference Links

[1] Home Edge code.

[2] Wiki.

Appendix F **Project EVE**

Introduction

Project EVE is an edge computing engine that enables the development, orchestration and security of cloud-native and legacy applications on distributed edge compute nodes. Supporting containers, clusters VMs and unikernels, it provides a flexible foundation for IoT edge deployments.

The explosion of data from IoT devices is driving requirements for increased in processing at the edge for reasons including latency, bandwidth savings and autonomy. However, deploying compute at the IoT edge is challenging because it is inherently heterogeneous, comprised of a diverse mix of technologies including sensors, control systems, communication protocols, hardware types, operating systems, applications, networks, cloud connections and more. In order to scale edge computing, it's important to tame this complexity by supporting a variety of deployment models in a more standardized and open way.

Project EVE will do for the IoT edge what Android did for the mobile market, by creating an open edge computing engine that enables the development, orchestration and security of cloud-native and legacy applications on distributed edge compute nodes. Supporting containers, clusters (Dockers and Kubernetes), VMs and unikernels, Project EVE provides a flexible foundation for IoT edge deployments on any hardware, application and/or cloud. Offering consistency and flexibility while maintaining a robust, state-of-the-art security posture is a key project tenet.

Architecture Description

Framework

The Project EVE runtime can be deployed on any bare metal hardware (e.g. x86, Arm, GPU) or within a VM to provide consistent system and orchestration services, in addition to the ability to run applications in a variety of formats. Orchestration of the underlying hardware and installed software is achieved through the open Project EVE API, providing developers with consistent behavior across a diverse mix of technology ingredients.

To access functionality remotely, the project offers a basic open source reference controller (Adam) and users can leverage any fully-featured third-party commercial controller that supports the open Project EVE APIs. When combined with a remote controller, Project EVE enables scalable, centralized management for large volumes of highly-distributed IoT edge compute nodes. The Project EVE in the Marketplace page lists available controllers and supported hardware.

Features

The goal of Project EVE is to enable IoT edge computing deployments with the following capabilities:

- Access to hardware root of trust (e.g. TPM) when deployed on bare metal, supporting functions such as crypto-based ID (no device usernames and passwords), measured boot, signed updates, encryption etc.;
- "Secure by default" deployment profile;
- High efficiency and usage of device resources including remote control of CPU, memory, networking and device ports;
- Hosting of any combination of apps in virtual machines and containers;
- Hosting of any operating system deployable in a virtual machine;

- Serverless capability via unikernels;
- Ability to assign CPU and GPU cores to specific applications;
- Ability to block unused I/O ports;
- Remote updates of entire software stack with rollback capability to prevent bricking;
- Automated patching for security updates;
- Automated connectivity to one or more backends (cloud or on premises);
- Distributed firewall.

Project EVE is the only foundation required to support the diversity of the IoT edge by abstracting hardware complexity while enabling orchestration flexibility. In terms of the project's roadmap, the following are key 2020 project goals identified by the Technical Steering Committee (TSC):

- Increase the modularity of the Project EVE architecture to support more deployment options;
- Add support for managed containers and deploying composite containerized applications via Docker Compose;
- Add support for Kubernetes via K3S and clustering;
- Expand hypervisor support to include KVM and ACRN for time-sensitive workloads;
- Add mesh networking capabilities for edge-to-edge data flow;
- Add cloud networking using standard VPN technologies available in public clouds;
- Continue to shrink the Project EVE runtime in order to run on smaller and resource-constrained embedded edge compute platforms.

Reference Links

[1] Project EVE code.

[2] Documentation.

[3] Mailing lists.

[4] Wiki.

[5] Slack channel.

Appendix G **State of the Edge**

Introduction

State of the Edge is a vendor-neutral platform for open research on edge computing dedicated to accelerating innovation by crowdsourcing a shared vocabulary for the edge. The project develops free, shareable research that is widely adopted and used to discuss compelling solutions offered by edge computing.

As a collaborative workspace for curating and defining terms related to the field of edge computing, collecting common and accepted definitions into an openly licensed repository, State of the Edge is governed as an open source project using a transparent and meritocratic process. Anybody can make additions, clarifications and suggestions by raising a GitHub issue or editing a branch and issuing a pull request. Each issue or pull request is evaluated by the community for inclusion. State of the Edge manages and produce the following assets:

- State of the Edge reports;
- Open Glossary of Edge Computing;
- Edge Computing Landscape.

The official version of State of the Edge is available via this GitHub repository. Proposed edits, clarifications and suggestions are made by filing GitHub issues or creating “pull requests.” Each issue, addition or suggested change is evaluated by the community for inclusion.

Architecture Description

State of the Edge is structured as an alphabetically-sorted markdown document stored in a GitHub repository

Framework

State of the Edge is available as a complete .pdf file.

Features

The State of the Edge project maintains the glossary itself as well as the Edge Computing Landscape.

Data Model / Schema

State of the Edge is maintained as a markdown document.

Reference Links

[1] GitHub repository.

[2] Mailing list.