Open Retail Reference Architecture

WORKING DRAFT

Aug 6, 2021

This document is a product of the Open Retail Reference Architecture (ORRA) project which is a sub-project of EdgeX Foundry.

Table of Contents

Questions and/or comments on this document can be sent to the ORRA Project via email to: EdgeX-ORRA@lists.edgexfoundry.org

1. Overall Goals

The initial phase of this paper has the goal to bring consensus amongst the group on what the joint reference architecture should be able to do and the kind of use cases that it should be able to address. In a future state, this document (or potentially a separate one) will aim to describe the goals that a retail organization (either a prospect and/or a customer of one or more of the entities represented by this group) and the ORRA group will accomplish by collaborating and engaging on this project.

For this initial phase, the ORRA group has the following 5 main goals :

1.1. A base foundation of retail-centric APIs at cloud and edge for building best-in-class retail experiences (that allow multiple vendors, suppliers and projects to both utilize and implement the APIs)

1.1.1. Support the creation of new solutions that can leverage multiple heterogeneous components - As a reference architecture, this document describes the power of a blueprint based on various technologies and software components to address a wide range of analytics at the edge use cases driven by some of the latest trends such as computer vision and machine learning inference at the edge.

1.1.2. Enable new solutions which require the integration of software from multiple, heterogeneous, vendors and providers

1.2. A common application deployment platform for edge-native deployments (to minimize validation and integration testing required)

1.2.1. Promote reference implementations which take guesswork and risk out of architectural choices

1.2.2. Drive actions locally where your customers, applications and/or your operations reside

1.2.3. Demonstrate “edge AI” and edge analytics deployed with easy to re-use building blocks

1.3. Consistent integration methods, drawing on cloud-native development practices (so that integrating applications and data at the edge uses the same technology as at the cloud)

1.3.1. Reduce infrastructure costs, including duplicate infrastructure

1.3.2. Use fewer teams to manage more environments (cloud and on-prem devices)

1.3.3. Demonstrate that on-prem application integrations can be done with the same APIs, same methods, and same technologies as cloud integrations. Do not require two documents for the same vendor to integrate to the cloud and to the edge.

1.4. A comprehensive set of connectors for retail IOT devices

1.4.1. Access data buried in devices, buildings and systems located at the edge
1.4.2. Improve actionable data and insights by getting data closer to the source
1.4.3. Move quicker by handling a variety of data sources with consistency

1.5. A community based on Open Source Software (OSS) principles to enable higher velocity of innovation

1.5.1. Bring a community along to support the effort, where a diversity of ideas raises the potential, and many eyes improve and harden the results.
1.5.2. OSS projects can provide benefit to industry standards bodies, allowing all parties to more quickly adopt industry relevant solutions and focus instead on domain specific problems

2. Key Indicators

- Call to Action: promote reference implementation and find customers that are willing to collaborate with us (i.e. feedback, POCs)
- Sensor fusion value prop as part of your digital journey, integration, frictionless, automation
- Explain how most use cases should be horizontal and cross segments
- Target Functions:
  Infrastructure: CTO,
  IT: CIO, Inventory
  Operations: COO, Distribution, Supply Chain

2.1. Market Segments

2.1.1. Convenience Stores

Small stores that sell a variety of products, such as newspapers, magazines, candy, soft drinks, tobacco products, and lottery tickets. Convenience Store is generally a well situated, food-oriented store with a long operating house and a limited number of items. Convenience stores are often open longer hours than other types of retail establishments, making them convenient for customers. However, prices are often higher than in other types of stores. Consumers use a convenience store; to fill in items such as bread, milk, eggs, and candy, etc.

2.1.2. Specialty Stores

Specialty stores are the retail establishments that specialize in the selling of a single type or specific range of merchandise and related items. These establishments typically concentrate their efforts on selling a single type or very limited range of merchandise. They concentrate on the sale of a single line of products or services,

such as Audio equipment, Jewelry, Beauty and Health Care, Clothing, Musical Instruments, Sewing Shops, and party supply stores. A typical specialty store gives attention to a particular category and provides a high level of service to the customers. Consumers are not confronted with racks of unrelated merchandise. Example: Banana Republic for clothing needs, Lululemon for workout clothing needs, Tanishq for jewelry and McDonalds, Pizza Hut, and Nirula's for food services.

2.1.3. Department Stores

Department stores are large retail establishments that are made up of a number of sections, or departments. A specific group of products is available in each department, each of which specializes in selling a particular grouping of products. For example, under this compartmentalized arrangement, consumers go to one area of the store to purchase tableware and another area to acquire bedding. These typically are very large stores offering a huge assortment of "soft" and "hard goods; often bear a resemblance to a collection of specialty stores. A retailer of such stores carries a variety of categories and has a broad assortment at an average price. A department store usually sells a general line of apparel for the family, household linens, home furnishings, and appliances. They offer considerable customer service. Example: Large format apparel department stores include Macy’s, Ebony, IKEA, Shoppers Stop, and Westside.

2.1.4. Supermarkets and Hypermarkets

Supermarkets and hypermarkets are retail establishments that were primarily involved with selling food. Many supermarkets carry other household products as well. Supermarkets are very similar to grocery stores, but they generally are larger and carry a wider selection of products. The supermarkets can be anywhere between 20,000 and 40,000 square feet (3,700 m2). A supermarket typically carries small household appliances, some apparel items, bakery, film developing, jams, pickles, books, audio/video, etc. A hypermarket is a large retail facility, or superstore, that carries a very wide variety of products under one roof, including groceries and a variety of non-food items. This is a self-service store consisting mainly of grocery and limited products on non-food items. HyperMarkets is a special kind of combination store that integrates an economy supermarket with a discount department store. A hypermarket generally has an ambiance which attracts the family as a whole. These retail establishments, which were primarily involved in providing food to consumers, have increasingly ventured into other product areas in recent years. They account for the vast majority of total food-store sales in America. Example: Safeway, Albertsons, WholeFoods, HEB, WINCO, etc.

2.1.5. Discount Stores

Discount stores are stores that typically sell a broad range of products at lower prices than other retail establishments. However, they generally also offer lower levels of service than higher-priced retailers. These stores tend to offer a wide array of products

and services, but they compete mainly on price. They offer an extensive assortment of merchandise at affordable and cut-rate prices. Normally retailers sell less fashion-oriented brands. These retail outlets offer consumers a trade-off: lower prices (typically on a broad range of products) in exchange for lower levels of service. Indeed, many discount stores operate under a basic ‘‘self-service’’ philosophy.

2.1.6. Multichannel Stores

These are retail establishments that sell products to consumers through a variety of channels, including catalogs, mail order, telemarketing, the Internet, and vending machines. They are also known as mail-order businesses and other non-store retailing establishments. The customer can shop and order through the internet or mail or other mediums and the merchandise is dropped at the customer's doorstep.

2.2. Use Cases

2.2.1. Inventory Tracking

2.2.1.1. USE CASE: All retailers need ways to track inventory including goods received, merchandise on shelf, and items sold. Oftentimes this is a very laborious process to have staff check the aisles, shelves, and stockroom periodically throughout the day. Such manual methods have lots of opportunities for errors and do not have real-time information. With the need to optimize the business and provide better customer service, sensors are in place to provide a near real-time accurate view of inventory.

2.2.1.2. DATA SOURCE: RFID tags are applied to high value items or boxes of small items to track movement of goods. AI vision processing can track the level of stock on shelves. Other types of low cost sensor tags are being tested to track in a more detailed level especially for clothing stores to identify misplaced items.

2.2.1.3. DATA FORMAT: Most of the above mentioned data sources make use of common data format or message protocols such as MQTT and REST API.

2.2.1.4. INDICATION: The means to flag inventory actions could vary widely depending on the store type, merchandise and promotional events. Notifications can be put up on message boards and checkout stations. For the case of quick serve restaurants, drive thru menu can be dynamically updated based on inventory levels.

2.2.1.5. BACKEND CONNECTION: Relevant events are to be forwarded to retailer’s backend system for syncing with ordering system, correlation with other stores, and more.

2.2.2. Loss Prevention

2.2.2.1. USE CASE: Self or frictionless or cashierless checkout stations are getting more popular in most retail stores, convenience stores, and business locations. These are not staffed and thus need ways to monitor for intended theft or unintended mistakes. Scenarios such as missed scans, hiding barcodes, swapped merchandise accounts for a large percentage of retailers' loss..

2.2.2.2. DATA SOURCE: An intelligent edge system is able to determine accurate insight on possibility of theft from reconciliation of multiple data sources. Data can come from POS transactions, barcode scan, weight scale, RFID reader, AI vision processing, just to name a few.

2.2.2.3. DATA FORMAT: Most of the above mentioned data sources make use of common data format or message protocols such as MQTT, REST API, Modbus, etc.

2.2.2.4. INDICATION: The means to flag or indicate a theft or loss might have happened will be the retailer’s decision. Some might choose a very visible method such as flashing a red light or sounding an alarm. Some might choose a more reserved method such as a message to the manager's mobile phone or an on screen icon.

2.2.2.5. BACKEND CONNECTION: Relevant events are to be forwarded to retailer’s backend system for consolidation and further analysis.

2.2.3. Occupancy

2.2.3.1. AS-IS: Current mechanisms to determine occupancy of a retail establishment are insufficient to produce a sufficiently reliable estimate to mitigate liability from COVID-related mandated maximums requiring staffing of entry and exit to manually count people.

2.2.3.2. TO-BE: Utilize multiple, heterogenous, sensors and actuators that signal human presence or absence to automatically estimate sufficiently accurate and precise occupancy, including segmentation by *time*, *area* (aka zone) and *role* (e.g. staff vs. patron).

2.2.3.3. KPI: In addition to elimination of staff at entry and exits, estimations of occupancy segmented by time, area, and role provide information which may be used to predict future occupancy and associated staffing requirements; prioritize areas for inventory checks, cleaning, or staff assignments; and if integrated with POS, CRM, or ERP systems provide occupancy information for system-specific KPI analysis (e.g. POS x CRM (member) x occupancy).

2.2.3.4. HOW-TO: Collect data from entry-exit automated door systems, shopping cart security, motion-sensors, interior and exterior security camera, staff RFID and/or Bluetooth locators, etc.. and push/pull via available protocols (e.g. MQTT, OPC-UA, ModBus, Kafka, ..) to/from EdgeX Foundry using appropriate OpenHorizon *services* on local enabled devices or gateways, process accordingly, push to data repository (e.g. Postgres, InfluxDB,

Min.io, ..), analyze using Jupyter notebooks and pipelines (n.b. Elyra-AI), and present using dashboard (e.g. Grafana, ..)

3. Design Principles

3.1. Edge Native Cloud

The phrase “Edge Native Cloud” refers to the term “Cloud Native”, but with the intention to understand the unique requirements of deploying compute to the far, on-premises edge. That means bringing the same technologies (containers, microservices, service meshes, event buses) and the same practices (orchestration, scale, devops, agility), but resolving the new challenges the edge brings (finite compute, low/no backhaul, security, expensive service calls). The broad set of challenges means that often shortcuts are taken during proof of concept or pilot phases, but for maximum utility this architecture strives to resolve all of them.

3.2. Consistent tools and technologies from edge to cloud / cloud to edge

One of the most important principles of this architecture is to utilize the same technologies as utilized in the cloud. Specifically, this includes the use of OCI-compliant container runtimes (such as Docker and Podman) and Kubernetes. Enterprises have already made the leap to the cloud, and most organizations have long term cloud investment strategies in play. The edge should not introduce yet another set of technologies, processes, or tools that a team needs to learn or support. Ideally the enterprise team can seamlessly support and move between environments with ease.

3.3. Composability, or infrastructure as code

The principles of composability, along with infrastructure as code (IAC), mean that the architecture should be built, imaged, and deployed with automation. This creates reproducible results and allows teams to support large numbers of devices distributed across large geographic areas. Utilizing IAC means the central teams can deploy compute, storage and networking with code, and use common software change control and management techniques to maintain consistency and quality across the fleet.

3.4. Prejudiced Flexibility

The phrase “prejudiced flexibility” means that the architectural choices laid out in ORRA will be chosen, but will be included and presented in a way to support ultimate choice and flexibility. For a mundane example, a project that is indicated for the service mesh component will be the primary ingredient included in subsequent collaterals, yet the authors understand that unique or unknown end user requirements may drive a different project selection.

A more elaborate example is the focus in this architecture on Container-based application packaging. Besides being in line with prevailing trends in software engineering, this prejudice enables a cascading set of simplifying assumptions -- including the use of EdgeX Foundry, Open Horizon, etc.. Doing so should not prevent VM-based and other application packaging technologies from being provided in other embodiments or through extensions.

Perhaps the largest value of the ORRA collaterals will be the documented reasoning provided for any given selection. Even if the audience were to disagree with every single decision they can still study and learn about why the decisions were made and ultimately reinforce the decision making required for their situation.

3.5. Open Source vs Commercial

It is no accident that ORRA is founded under EdgeX Foundry and the Linux Foundation. Ideally the architecture is based around open source components that have commercial support offerings available on the market. Building around open source means greater community, innovation and inclusion, and commercial support means that downstream adopters have confidence that they can find the support systems required by their organization. Not all ingredients are likely to fit this principle perfectly, but it is the preferred approach. The architecture is meant to be used in mixed OSS and proprietary environments, and integrating commercial applications should be as easy as any other application.

4. Node Lifecycle Phases

The purpose of calling out lifecycle phases is to be inclusive of the various stages that an edge compute node undergoes throughout its active life. Rather than sit in an ivory tower and pontificate on the ideal edge compute stack when it is fully utilized, the architecture instead needs to understand the requirements of different phases of its life.

4.1. Imaging and Deployment

The scope of Imaging and Deployment is to understand the needs of the deployer, typically a System Integrator (SI) or Information Technology Outsourcer (ITOOTI), who has to build, image and ship thousands, or even tens of thousands, of computers and programmable devices to the various destination locations. Unlike on-site teams which may be able to build all of the computers for their campus, distributed teams are required for distributed edge compute deployments. And unlike homogeneous public cloud environments where thousands of similar servers are supported in a single region, zone, or physical data center, these heterogeneous devices will be in various shapes and sizes, and coming from multiple different suppliers, and being routed to multiple different destinations. Managing golden images is no longer able to cut it.

4.2. Onboarding

Onboarding refers to the phase when devices are unboxed, connected to power and the network, and become a functional edge compute node. There is of course a manual method of onboarding through physical configuration of the device, but the goal is to make this zero touch, so that any available person can connect up the device, and then automation takes over the personalization and provisioning of the node. Clustering, if applicable, is often closely entwined with this phase.

4.3. Orchestration and Deployment

This phase is generally the phase that most technologists envision: when the edge nodes are actively participating in the network, and software workloads are being dispatched to the nodes. This phase needs to incorporate poor network connectivity, local orchestration, identification and utilization of accelerators and heterogeneous compute resources.

4.4. Repair and replacement

When compute resources reach the end of their lifecycle, they will need to be repaired, retired or replaced. It may be due to hardware failure or performance reasons, but regardless there will be requirements around unprovisioning, secret storage removal, and related end of life challenges. This phase also includes telemetry and monitoring as those are critical measures of health and stability.

5. Scope

5.1. Supported Hardware architecture and OS

Hardware architecture	OS and version
x86_64/amd64	Ubuntu 18.04 (bionic)
x86_64/amd64	Ubuntu 20.04 (focal)
arm64/aarch64	Ubuntu 18.04 (bionic)
arm64/aarch64	Ubuntu 20.04 (focal)

* RHEL 8.3 support on PPC64le requires installation of Docker.

5.2. Supported GPUs

Regarding GPU and other hardware support, it will mostly depend on the container runtime and libraries being used on each edge node for workloads being deployed to those nodes. For the sake of this document, usage of achatina will be assumed.

5.3. Edge-to-Cloud Consistency - APIs/SDKs/Protocols

In defining an edge built on cloud principles, it is important that the edge architecture provides a similar, if not equal, set of services for the software developer. As an example, if a cloud developer is authoring container-based microservices that adopt universally accepted telemetry services, then we would expect those same services to function at the edge as well. The goal is not to require the developers to learn a new set of tools, capabilities or practices for the edge in order to create transportability of workloads from clouds to the various kinds of edges.

To that end the team will be continuing to refine what needs to be specified and/or delivered by ORRA in order to provide cloud consistency to edge deployers. This is likely to include telemetry, key-value storage, function as a service engine, service meshes, and more. Further, in a similar vein to digital twins, it may be necessary to create local data caches for unreliable cloud connections, and possibly even require creativity or proxy solutions when cloud applications deployed at the edge expect local API services that are not actually available locally. Resource synchronization, database replication, or custom network routing are other potential areas for making the edge an adoptable, high velocity platform for innovation.

5.4. Software Components

5.4.1. EdgeX Foundry 1.3
5.4.2. Open Horizon 4.2.1
5.4.3. Secure Device Onboard 1.10.1
5.4.4. Flogo

5.5. Supported Standards

Any industry or technology standards and specifications that are officially supported by ORRA are referenced here.

5.5.1. FIDO Device Onboard (FDO)

“An automatic onboarding protocol for IoT devices. Permits late binding of device credentials, so that one manufactured device may onboard, without modification, to many different IOT platforms.”

The SDO project within LF Edge provides an implementation of the FIDO Device Onboard (FDO) specification.

5.5.2. Open Container Initiative (OCI)

The Open Container Initiative is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.

Established in June 2015 by Docker and other leaders in the container industry, the OCI currently contains two specifications: the Runtime Specification (runtime-spec) and the Image Specification (image-spec). The Runtime Specification outlines how to run a “filesystem bundle” that is unpacked on disk. At a high-level an OCI implementation would download an OCI Image then unpack that image into an OCI Runtime filesystem bundle. At this point the OCI Runtime Bundle would be run by an OCI Runtime.

ORRA will utilize OCI compliant container images and OCI compliant container runtimes.

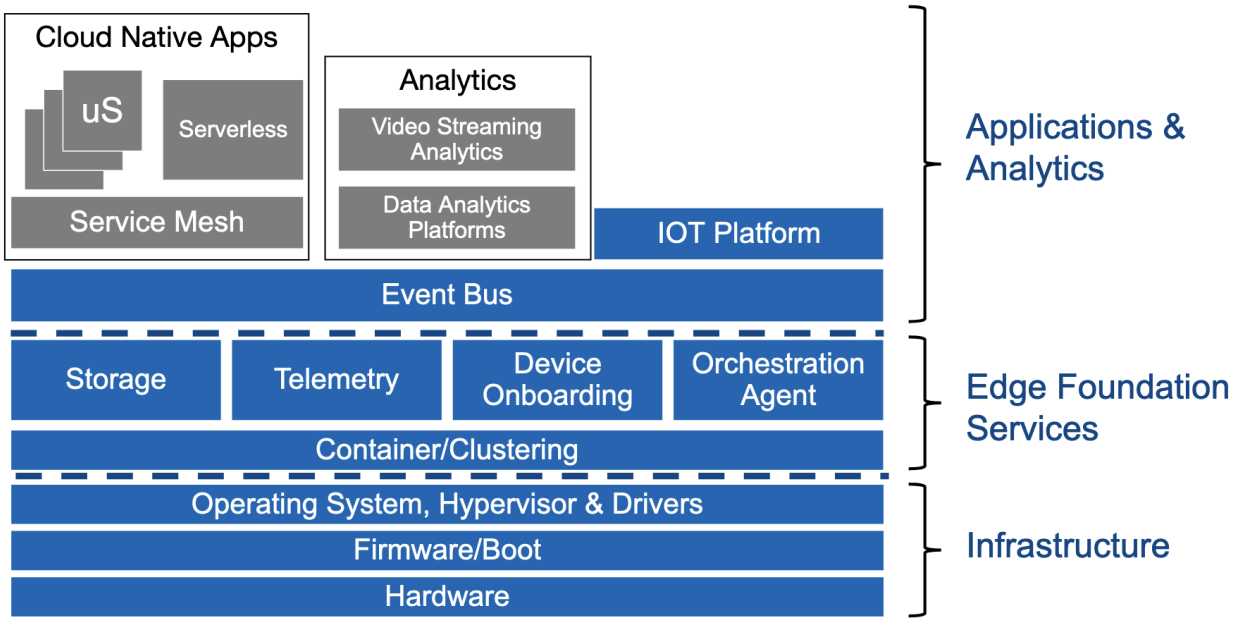
5.5.3. Kubernetes (K8s)

“... [A]n open-source system for automating deployment, scaling, and management of containerized applications.”

The Open Horizon project within LF Edge enables remote, autonomous deployment of containerized workloads to both Kubernetes clusters and Docker hosts. It primarily utilizes the K8s API and the Operator SDK.

6. Platform

6.1. Platform Layers



The purpose of this diagram is to sort the scope of work and responsibilities of the edge architecture into compatibility layers. The layers should encourage confidence that work done in higher layers is portable to other platforms with similar layers. For example, container-based applications in the “Applications” layer should be portable to edge nodes running various flavors of Linux or even Windows (via Windows Services for Linux). If the applications consistently use an event bus, then relocations or integrations on other platforms should be predictable if they use the same event bus, and if deployed in linux containers, should be portable to most other systems that support linux containers.

Each rectangle (or group of rectangles in the case of the grey clusters), should be included in the architecture.
The dotted lines suggest contracts, or dependencies, that one layer exposes to another. Care should be considered with the contracts to balance consistency of experience without overly specifying an onerous contract..

6.1.1. Infrastructure

This layer refers to the hardware, operating systems and device drivers which make up the base of any edge node architecture.

6.1.2. Edge Foundation Services

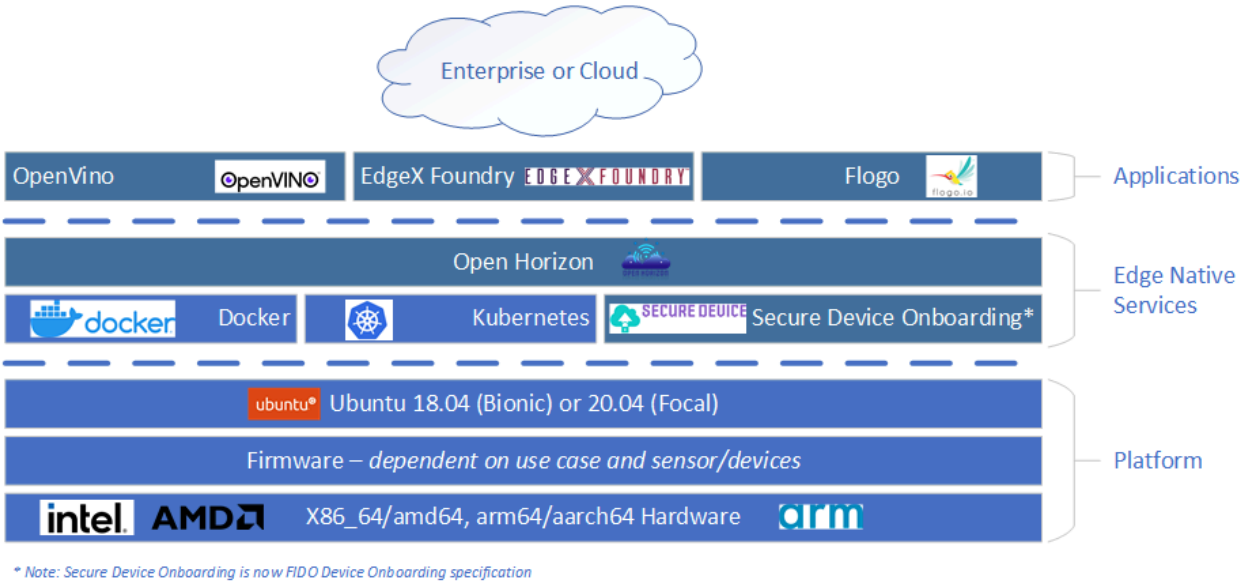
This layer designates the software required to operate and support the edge node, such as provisioning, onboarding, storage, telemetry, container daemons, clustering middleware, etc. The contract exposed up to the applications layer is the set of services enabled for containers and virtual machines to run portably across environments.

6.1.3. Applications

The Applications layer is quite large but if the architecture is designed suitably, this layer can be portable across nodes that offer similar edge native or cloud native services. This layer contains the event bus, the IOT platform, the edge analytics services, and the suite of cloud native applications (micro services, serverless engines, service meshes, etc).

6.2. Components

The major platform components (listed alphabetically) are EdgeX Foundry, Open Horizon, and Secure Device Onboard, meaning that their function within the Reference Architecture is not optional, and the use cases would not be possible without those components or suitable replacements. However, other solutions are involved as dependencies and also may have suitable alternatives available. Those solutions being used as default sub-components are also called out below along with valid substitutions noted.



6.2.1. EdgeX Foundry

6.2.1.1. Overview

EdgeX Foundry is an open source, vendor neutral, flexible, interoperable, software platform at the edge of the network, that interacts with the physical world of devices, sensors, actuators, and other IoT objects. In simple terms, EdgeX is edge middleware - serving between physical sensing and actuating "things" and our information technology (IT) systems.

EdgeX is made up of a collection of micro services. The micro service architecture serves several important features of the platform:

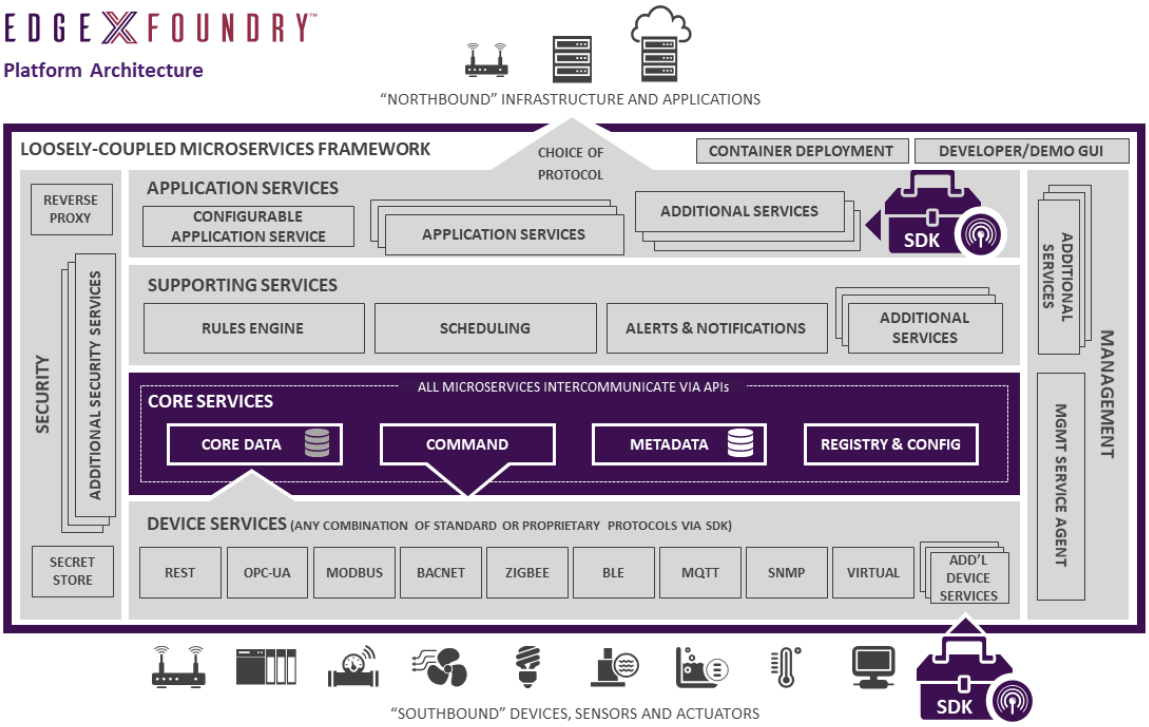
- The micro services can be distributed across a set of hosts, allowing EdgeX to maximize resources while at the same time position more processing intelligence closer to the physical edge.
- Because the micro services are loosely coupled to one another, it allows any one of the services to be quickly and easily replaced (as in an upgrade, through efforts of another project, by a commercial implementation adding value with its implementation, etc.).
- Each micro service can be implemented in the technology best suited for the use case, host environment, or circumstances of the implementing organization.

EdgeX Foundry was conceived with the following tenets guiding the overall architecture:

- must be platform agnostic with regard to hardware, OS, distribution, deployment/orchestration, protocols.
- must be extremely flexible.
- provides a "reference implementation" set of services but encourages best of breed solutions.
- must provide for store and forward capability in order to support disconnected/remote edge systems and to deal with intermittent connectivity.
- must support and facilitate "intelligence" moving closer to the edge in order to address latency concerns, bandwidth and storage concerns, and remote/independent operations.
- must support brown and green device/sensor field deployments.
- must be secure and easily managed.

6.2.1.2. Architecture

The EdgeX platform consists of micro services in three layers.



- Device Services

  Device services (see the bottom layer of services in the diagram above) translate information coming from devices via hundreds of protocols and thousands of formats and bring them into EdgeX. In other terms, device services ingest sensor data provided by "things". When it ingests the sensor data, the device service converts the data produced and communicated by the "thing" into a common EdgeX Foundry data structure, and sends that converted data into the core services layer, and to other micro services in other layers of EdgeX Foundry.

  Device services also receive and handle any request for actuation back to the device. Device services take a general command from EdgeX to perform some sort of action and it translates that into a protocol specific request and forwards the request to the desired device.

- Application Services

  Application Services (see the top layer of the services in the diagram above) are a means to get data from EdgeX Foundry to external systems and processes (be it analytics package, enterprise or on-prem application, cloud systems like Azure IoT, AWS IoT, or Google IoT Core, etc.). Application Services provide the means for data to be prepared (transformed, enriched, filtered, etc.) and groomed (formatted, compressed, encrypted, etc.) before being sent to an endpoint of choice.

Application Services are based on the idea of a "Functions Pipeline". A functions pipeline is a collection of functions that process messages (in this case EdgeX event/reading messages) in the order that you've specified. The first function in a pipeline is a trigger. A trigger begins the functions pipeline execution. A trigger is something like a message landing in a watched message queue.

- Core Services

Core services (see the middle layer in the diagram above) provide the intermediary between the north and south sides of EdgeX. As the name of these services implies, they are “core” to EdgeX functionality. Core services is where the innate knowledge of “things” connected, sensor data collected, and EdgeX configuration resides. Core consists of the following micro services:

  - Core data: a persistence repository and associated management service for data collected from south side objects.
  - Command: a service that facilitates and controls actuation requests from the north side to the south side.
  - Metadata: a repository and associated management service of metadata about the objects that are connected to EdgeX Foundry. Metadata provides the capability to provision new devices and pair them with their owning device services.
  - Registry and Configuration: provides other EdgeX Foundry micro services with information about associated services within the system and micro services configuration properties (i.e. - a repository of initialization values).

In addition to device, core and application services, are:

- optionally used supporting services encompass a wide range of micro services to perform normal software application duties such as scheduling, notifications/alerting and a reference rules engine for analytics.
- security services that protect the system secrets in secure storage and a reverse proxy to restrict access to EdgeX REST resources and perform access control related works.
- system management facilities are optional and provide the central point of contact for external management systems to start/stop/restart EdgeX services, get the configuration for a service, the status/health of a service, or get metrics on the EdgeX services (such as memory usage) so that the EdgeX services can be monitored.

EdgeX, as an open source platform, leverages several open source sub-components to include:
Redis - database (and Redis Streams for messaging)
Vault - for secure storage
Kong - API Gateway
Consul - service registry and configuration
Kuiper - rules engine (and fellow LF Edge project)

6.2.1.3. Ecosystem

EdgeX was co-founded by over 50 companies in the Linux Foundation in April 2017. Today, EdgeX is a Stage 3 (Impact) project under the LF Edge umbrella. EdgeX has more than 200 contributors and has enjoyed more than 8 million container downloads in over half a million deployments across the globe.

6.2.1.4. Resources

Web Site: www.edgexfoundry.org
Documentation: docs.edgexfoundry.org
Project Wiki: wiki.edgexfoundry.org
Project Slack: https://edgexfoundry.slack.com
Project Code: https://github.com/edgexfoundry/

6.2.2. Open Horizon

6.2.2.1. Overview

Open Horizon is an application and metadata delivery and management platform. It provides a policy-based mechanism to securely and autonomously deliver containerized workloads to edge compute nodes of varying sizes and capabilities and in various connected states. It allows workload and ML model management across fleets at hyperscale, from a single device to deployments of 40,000 nodes or greater, without requiring on-premises administration. Open Horizon helps Enterprises reduce costs by providing remote and autonomous application management.

6.2.2.2. Introduction

Open Horizon is a platform for deploying containerized applications to edge computing nodes (both devices and clusters) and managing their service software lifecycle. It also has a Model Manager sync service for bi-directional synchronization to and from edge nodes for the deployment of machine learning assets. The platform is maintained by the Open Horizon open-source project within the LF Edge umbrella organization in the Linux Foundation. The platform is vendor-neutral and is intended to support and implement applicable open standards in the space.

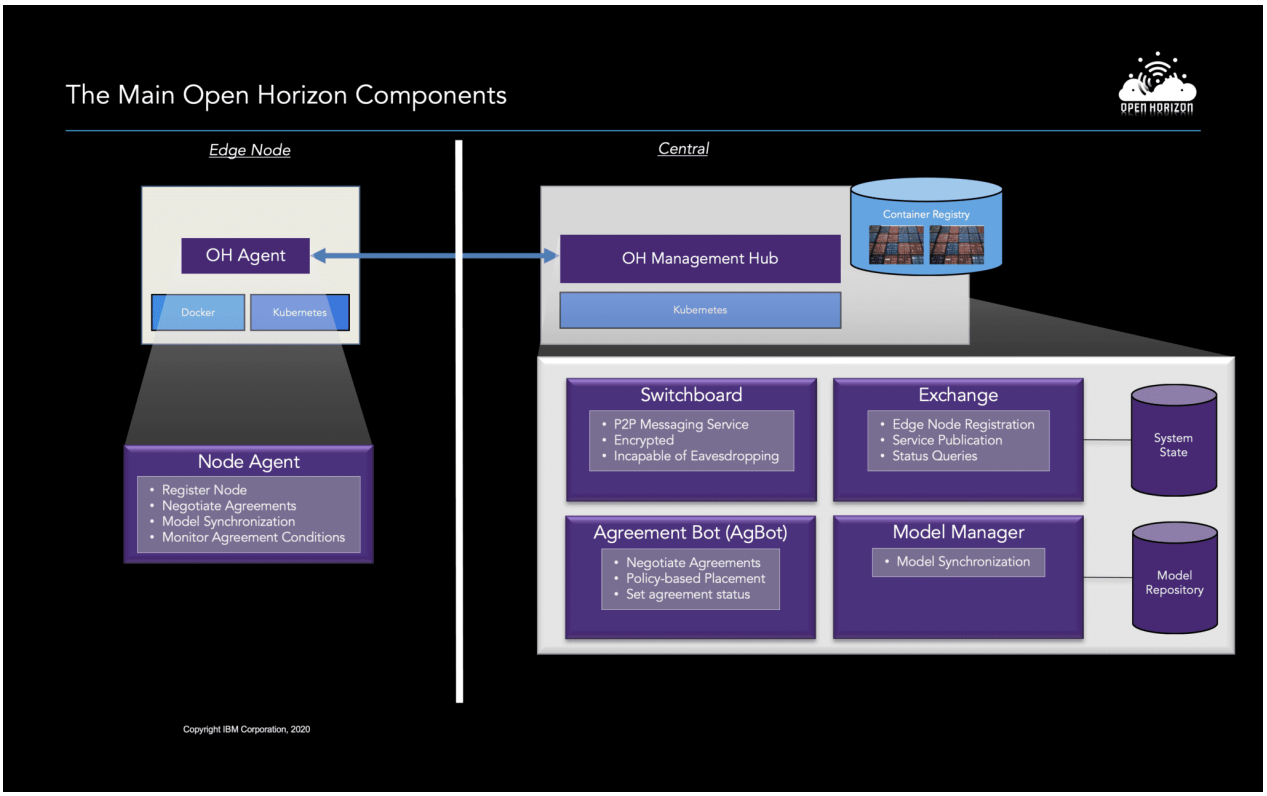
A key distinction of Open Horizon is the way that it scales, which allows a single administrator to manage a large fleet of tens of thousands of autonomous edge nodes by expressing intent through policies. Edge nodes are capable of operating in connected, partially-disconnected, or completely-disconnected states without issue. The Agent is responsible for initiating all communication with the Management Hub Services (primarily the Exchange), thus reducing the potential attack surface. To

further ensure security, Open Horizon implements Perfect Forward Secrecy in all communications.

6.2.2.3. Architecture

- Component Services

  The platform consists of two categories of component services: Agent and Management Hub.



The Agent is written in Go and the Management Hub services in Scala.

The platform is comprised of five logical components:

- Agent
- Exchange
- Agreement Bot
- Switchboard
- Model Manager

- Sub-Component Services

  Open Horizon also uses the following open-source solutions as sub-components:

  - FIDO Device Onboard (FDO)

    This is used for zero-touch onboarding of devices to automate installation of the Agent, registration with the Exchange, and deployment of services. FDO is described in more detail later in this document.

- CouchDB

  This is used for large object storage, and primarily in relation with the Model Manager.

- PostgreSQL

  This is used as a relational database by the Exchange to store system state.

- Vault

  This is used for shared secrets and configuration.

- microk8s (or k3s, OpenShift)

  The Open Horizon Agent can deploy and manage workloads on all K8s API-compatible Kubernetes clusters.

- Docker (or podman)

  Open Horizon deploys containerized applications and is intended to be compatible with container execution environments that support OCI-compliant containers. This includes the Docker engine on most hosts, and podman within Kubernetes.

6.2.2.4. Resources

Project Page: https://www.lfedge.org/projects/openhorizon/
Project Wiki: https://wiki.lfedge.org/display/OH/Open+Horizon
Communication: https://chat.lfx.linuxfoundation.org/
Code: https://github.com/open-horizon
Documentation: https://open-horizon.github.io/
Mailing List and Calendar: https://lists.lfedge.org/

6.2.3. FIDO Device Onboard (FDO)

6.2.3.1. Overview

FIDO Device Onboard is a device onboarding scheme from the FIDO Alliance, sometimes called "device provisioning". Device onboarding is the process of installing secrets and configuration data into a device so that the device is able to connect and interact securely with an IoT platform. The IoT platform is used by the device owner to manage the device by: patching security vulnerabilities; installing or updating software; retrieving sensor data; by interacting with actuators; etc. FIDO Device Onboard is an *automatic* onboarding mechanism, meaning that it is invoked autonomously and performs only limited, specific, interactions with its environment to complete.

A unique feature of FIDO Device Onboard is the ability for the device owner to select the IoT platform at a late stage in the device life cycle. The secrets or configuration data may also be created or chosen at this late stage. This feature is called "late binding".

Various events may trigger device onboarding to take place, but the most common case is when a device is first "unboxed" and installed. The device connects to a prospective IOT platform over a communications medium, with the intent to establish mutual trust and enter an onboarding dialog.

Due to late binding, the device does not yet know the prospective IOT Platform to which it must connect. For this reason, the IOT Platform shares information about its network address with a "Rendezvous Server." The device connects to one or more Rendezvous Servers until it determines how to connect to the prospective IOT platform. Then it connects to the IOT platform directly.

FIDO Device Onboard works by establishing the ownership of a device during manufacturing, then tracking the transfers of ownership of the device until it is finally provisioned and put into service. In this way, the device onboarding problem can be thought of as a device "transfer of ownership" or delegation problem. In a common situation for FIDO Device Onboard that uses the "untrusted installer" model, an initial set of credentials and configuration data is configured during manufacturing. Between when the device is manufactured and when it is first powered on and given access to the Internet, the device may transfer ownership multiple times. A structured digital document, called an Ownership Voucher, is used to transfer digital ownership credentials from owner to owner without the need to power on the device.

In onboarding, an installer person performs the physical installation of the IOT device. In the untrusted installer model, the device takes no guidance on how to onboard from an installer person who has powered on the device. The FIDO Device Onboard protocols are invoked when the device is first powered on. By protocol cooperation between the Device, the Rendezvous server, and the new owner, the device and new owner are able to prove themselves to each other, sufficient to allow the new owner to establish new cryptographic control of the device. When this process is finished, the device is equipped with credentials supplied by the new owner.

6.2.3.2. Resources

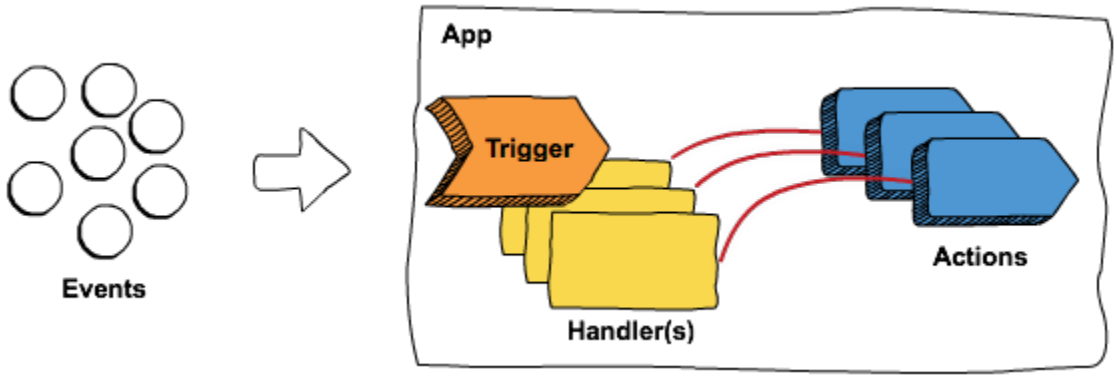
Project Page: https://www.lfedge.org/projects/securedeviceonboard/
Specification: https://fidoalliance.org/specs/FDO/fido-device-onboard-v1.0-ps-20210323/fido-device-onboard-v1.0-ps-20210323.html#fido-device-onboard-base-profile-normative

6.2.4. Flogo

6.2.4.1. Overview

Project Flogo is an ultra-light, Go-based open source ecosystem for building event-driven apps.

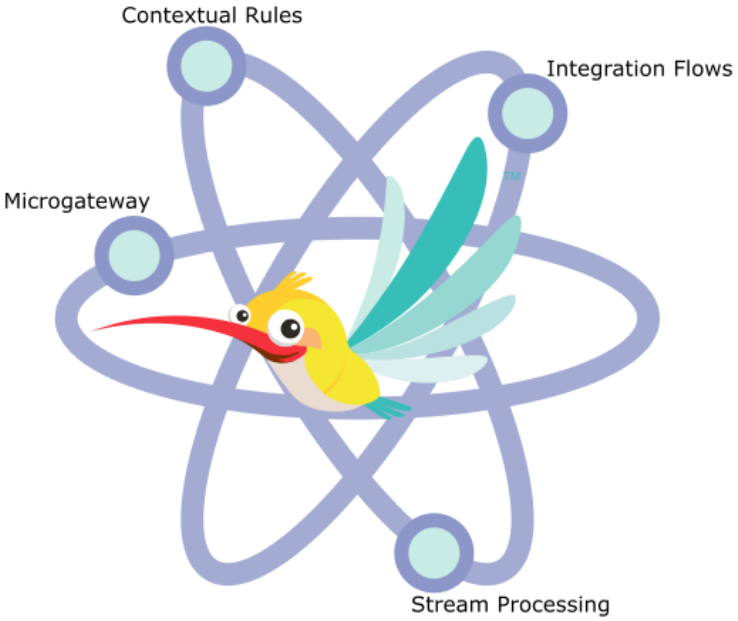
The notion of *triggers* and *actions* are leveraged to process incoming events. An action, a common interface, exposes key capabilities such as application integration, stream processing, etc.



- App = Trigger(s) + Actions[&Activities]
- Triggers
    - receive data from external sources.
    - are managed by a configurable threading model
    - have a common interface enabling anyone to build a Flogo trigger.
- Handlers
    - dispatch events to actions
- Actions
    - process events in a manner suitable with the implementation
    - have a common interface enabling opinionated event processing capabilities

6.2.4.2. Ecosystem

All capabilities within the Flogo Ecosystem have a few things in common, they all process events (in a manner suitable for the specific purpose) and they all implement the *action* interface exposed by Flogo Core.



6.2.4.3. Resources

Project Page: https://www.flogo.io/

7. Additional Information

Please check the ORRA Project wiki page for the latest version of this document and other related information.

Demos can be found in the ORRA Github repository.