ENABLING EMBEDDED DEVICES FOR INDUSTRIAL INTERNET OF THINGS (IIoT)

ARVIND RAGHURAMAN, SENIOR ARCHITECT, EMBEDDED PLATFORM SOLUTIONS
SCOT MORRISON, GENERAL MANAGER, EMBEDDED PLATFORM SOLUTIONS

Mentor®
A Siemens Business

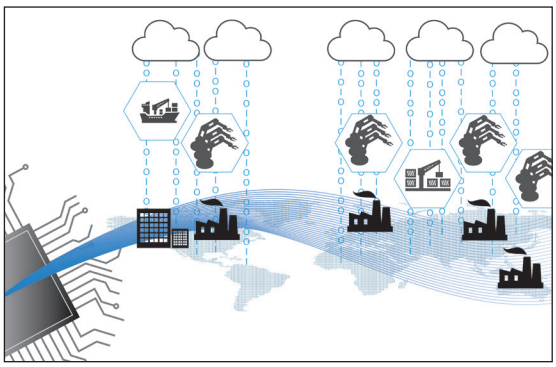
WHITEPAPER

EMBEDDED SYSTEMS

www.mentor.com

INTRODUCTION

For industrial equipment manufacturers, the market demand to integrate and connect devices within the Industrial Internet of Things (IIoT) is growing by the day. The competitive pressures and customer expectations for reliability, maintainability, upgradeability, and secure operations are a few of the primary drivers.



The available enabling technology consists of a few comprehensive cloud platforms with their associated device enablement software development kits (SDKs), plus a dizzying amount of commercial and open source runtime components and cloud backend applications.

This paper seeks to outline a few of the key architectural considerations required for the successful operation of smart devices within an IIoT infrastructure with a focus on the software that runs on the device. A few of the more common device software challenges will be highlighted, and a promising new framework that addresses some of the key device software gaps will be presented.

INDUSTRIAL IoT ENABLEMENT

The major goals on an Industrial IoT implementation are: 1) to securely connect end node and edge/gateway type embedded devices to a cloud backend; 2) collect data from these devices, visualize/analyze or present this data in a meaningful manner; and 3) to offer a means in which to interact, configure, maintain, and upgrade the devices using a cloud-based infrastructure.

An alternative approach quickly gaining momentum is to host the device monitoring and management applications locally, as opposed to hosting them in the cloud. This approach is called an “on-premises solution.” While this approach mitigates some of the security concerns of a cloud-based approach, from an architectural standpoint, it is very similar to a cloud-based solution and can be assumed identical for the purposes of this discussion. Of course, an on-premises solution would be characterized by limited scalability with respect to compute and storage capabilities.

To illustrate how a smart device interacts with cloud backend services in an IIoT system, let’s take a look at the basic steps of onboarding, managing, and monitoring a typical smart device.

STEP #1: SECURE ONBOARDING

This begins with the secure boot of a device. The boot process must sequentially authenticate the boot loader, operating system software, and other software components executed as part of the boot sequence. A secure boot flow is of paramount importance because it protects the device from "rooting" type attacks, ensuring that the device boots trusted software only. The other aspect of secure onboarding is device authentication infrastructure present on a cloud or on-premises hosted backend. This infrastructure allows for identification and authentication of devices trying to onboard into the system.

STEP #2: CONFIGURE, MONITOR, AND CONTROL

Once the device is onboarded, the next step is for the device runtime to instantiate and expose the parameters and services supported by the device to the backend. This is accomplished using a "data model" or "object model" that is compatible with the backend infrastructure.

STEP #3: SECURE TELEMETRY

Once the data or object model is established between the device and the backend, the device can start pushing data up into the cloud and receive asynchronous messages from the cloud. Securing data in motion is a table stake requirement in IIoT. Transport Layer Security (TLS) and Secure Socket Layer (SSL) -based methods are typically employed to establish a secure connection and encryption of data exchanged between the device and the backend.

STEP #4: SOFTWARE UPDATES AND MAINTENANCE

This is a key feature with regard to achieving the higher reliability expectations of cloud-connected industrial devices. In order to fix bugs, upgrade functionality, and patch security issues, a comprehensive infrastructure to manage device firmware, applications, and data is critically important.

EMBEDDED DEVICES IN INDUSTRIAL IOT

IIoT devices can be categorized as end nodes, which are located in the lower tier of an IIoT ecosystem, and edge nodes that often serve as a gateway between the end nodes and the cloud backend. End nodes are commonly actuators, sensors, controllers, human machine interfaces (HMIs), etc. In some cases, end nodes will connect directly to the cloud without the use of an edge node/gateway intermediary.

Although both end nodes and gateways are *embedded devices* they can vary significantly in form factor and functionality. End node devices can be quite small; often, they are 8- or 16-bit smart sensors that utilize simplified wireless protocols and extreme power management strategies to harvest local energy for maintenance-free operation. On the other end of the spectrum, edge nodes can be powerful multi-processor, multicore devices with enterprise/server-like computational power. From a software perspective, end node devices can run bare metal (no operating system) or for larger examples, a real-time operating system (RTOS) or even a general purpose operating system (GPOS) such as a Linux®. Edge node devices typically run an RTOS.

An example component architecture for runtime software for an edge node, or a cloud-connected end node, is shown in Figure 1 below. The diagram depicts a typical architecture that consists of a cloud vendor-provided SDK for the device and other OS/System services needed to fulfill the device management needs.

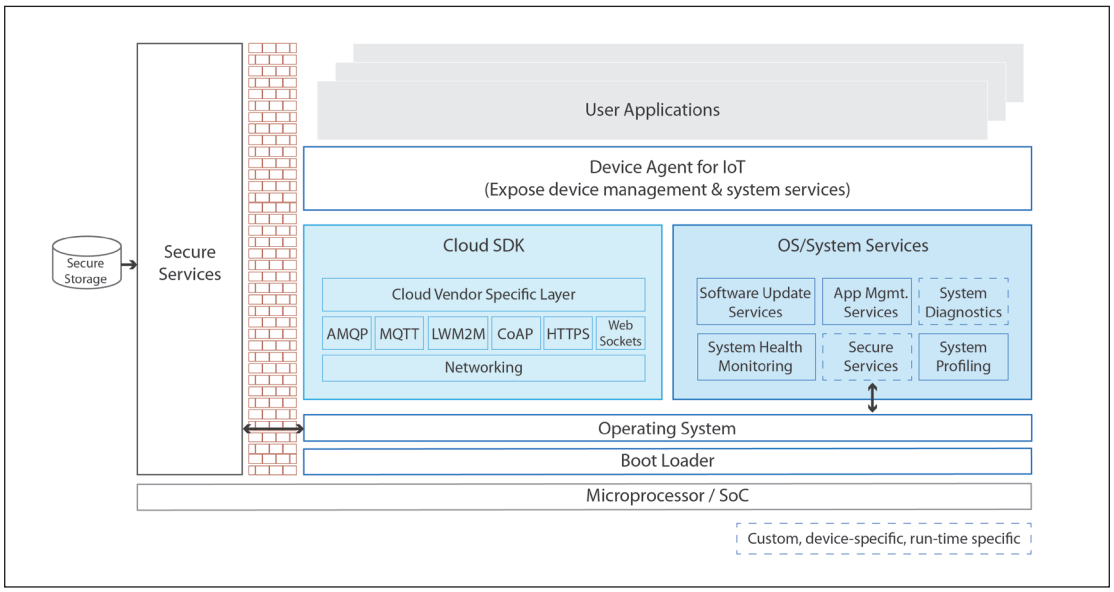


Figure 1: *Device software IIoT architecture is comprised of a cloud vendor-provided software developer kit (SDK) for backend services, and OS/System services which are available in the OS runtime environment.*

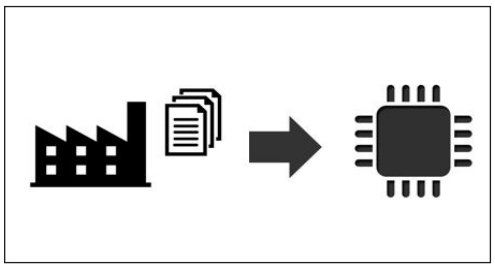
CLOUD PLATFORM AND DEVICE SDK-PROVIDED FUNCTIONS FOR DEVICE MANAGEMENT

Cloud vendors provide a variety of infrastructure and tools to create and manage the identity of a connected device. One of the primary objectives among cloud vendors is to offer the means to securely onboard a device, and then monitor, manage, and update that device. However, specific capabilities do vary from one provider to the next. Some of the core features you can expect from the major cloud vendors are summarized below.

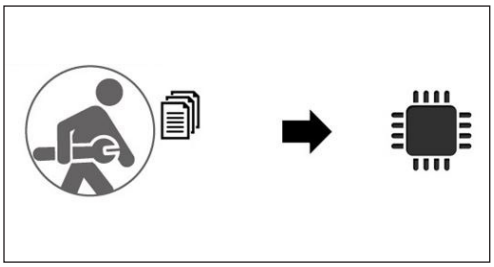
DEVICE IDENTITY MANAGEMENT AND PROVISIONING

Cloud platforms typically provide the infrastructure needed for users to add, remove, and manage devices and associated security attributes. When a user adds a new device, an identity for the device is created in a device registry hosted on the cloud and associated security credentials are generated. These credentials need to be provisioned to the device to enable secure onboarding onto the cloud platform. In typical environments, public key-based mutual authentication is used to establish a robust device authentication infrastructure.

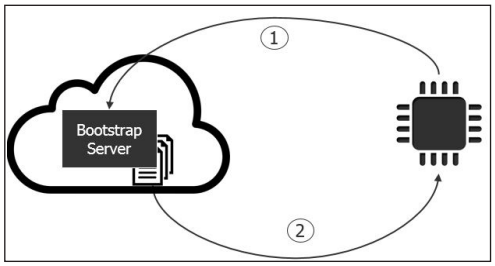
The actual provisioning of the artifacts to the device can be done in several different ways.
A few examples include:



Factory-provisioned
Security credentials are provisioned to secure storage on the device during the manufacturing process. The identity of the device is thereby hardcoded before it is installed in a deployment setting.



Manual provisioning
The device artifacts are installed on the device during deployment in the manufacturing setting. This allows more custom identity control to the specific manufacturing environment and OEM IIoT requirements. Certificate revocation/updates can be done in a similar fashion. For large device counts, the manual provisioning approach would be time and labor intensive.



Bootstrap server-based provisioning
In this model, when the device comes up, it securely connects to a bootstrap server, which provisions the device with the credentials it needs to connect to the actual cloud backend. The bootstrap server is considered a *persistent entity* because it's always available to all devices. In order to revoke certificates, or change configuration data such as a change to the endpoint URI (an end point URI in this scenario, is the URL of a cloud-hosted or on-premises application to which the device attempts to connect on boot), a user would log-in to the bootstrap server and replace the certificate and/or configuration package that's to be provisioned to the devices. Subsequently, when the devices boot up, they get provisioned with the updated artifacts and continue operations without any downtime.

COMMUNICATIONS, DATA MODELS, AND DEVICE MANAGEMENT

Cloud vendor-provided device SDKs include a suite of communication protocols supported by that vendor's cloud platform. On top of this protocol suite layer sits a cloud vendor-specific layer which enables higher order constructs that may be specific to the cloud platform. The vendor-specific layer may include platform-specific bindings to instantiate device data models, APIs to define and negotiate a device twin, etc.

-Communication protocols

A cloud platform typically supports a set of communications protocols and enables these protocols in the cloud and device SDKs. Device applications leverage these communication APIs for messaging and telemetry with the cloud platform.

Some of the more commonly used communication protocols include:

- MQTT: Enables a publish/subscribe-based communication topology. Widely used and lightweight in nature. Typically used for both northbound and southbound interfaces on gateways.
- AMQP: Provides peer-to-peer and publish/subscribe messaging with support for reliable queuing. AMQP has a larger memory footprint than MQTT, but it can also be more reliable and interoperable. Typically found in northbound interfaces from gateway to cloud.
- HTTPS and CoAP: Both are client/server type communication topologies that can be used to implement RESTful services. HTTP is feature-rich and prevalent due to its wide adoption in the IT space. CoAP is a simple and lightweight protocol based on an underlying UDP transport, suitable for resource-constrained devices.
- DDS (Data Distribution Services): A real-time deterministic protocol. Operates as a publish/subscribe-based method of communication.
- OPC UA (OPC Unified Architecture): Extremely popular in industrial environments. Conforms to client/server architectures. The latest version also supports the publish/subscribe paradigm.

-Data model

When the device comes up, a data model needs to be negotiated between the device and the cloud backend. There are several approaches this process can take.

A **custom data model approach** is supported by cloud platforms like Microsoft Azure and Amazon Web Services (AWS), where by using platform-specific bindings provided by the device SDK, the device can define and expose its data model to the backend. These models determine the nature and format of data the device exchanges with the backend. The model typically constitutes the definition of parameters and methods the device would expose to the backend.

Lightweight machine-to-machine (LwM2M) is a good example of a **standards-based, object model approach**. LwM2M leverages the efforts made by the OMA standard and IPSO Alliance, an industry consortium that supports LwM2M by publishing a vast set of bindings for smart objects such as sensors and actuators, device firmware, industrial machinery, etc. Since the objects have standardized bindings, they can connect up to any cloud infrastructure that conforms to the same object model.

One of the primary advantages behind an object model-based approach is that users can switch from one cloud backend to another – without having to alter the device software.

-Device twins

Many cloud platforms offer a virtual representation of the device in the cloud. Microsoft Azure calls this technique "Device Twins" and AWS calls it "Thing Shadow".

This virtual instance can have many applications including having access to device state representation. Backend services interact with the device twin to access (read/write) device state information. There are several benefits to this. For example, a backend service's request to initiate a device software update is issued to the device twin, which then propagates the request to a device. If the device is online, the update is executed. If the device is offline, the request is held pending in the device twin state machine until the device comes back online. When the device is back online, the device synchronizes its state with the twin and receives the request to update. Another example, if the device goes offline due to a runtime failure, the last reported device state is available in the device twin which could come in handy for troubleshooting.

OTHER DEVICE SOFTWARE FUNCTIONS NEEDED FOR MANAGEMENT OF IIoT DEVICES

OS/SYSTEM SERVICES

OS/System services expand on the foundation device onboarding and communications infrastructure provided by typical cloud vendor SDKs. These services fulfill the functions necessary for Industrial IoT devices enabling comprehensive device management, including life cycle management, system/device health monitoring, device software updates, device application updates, etc.

A few of the key attributes OS/System services provide include:

Software updates

The ability to update device operating system (OS) software and application software is an essential element of IIoT smart device enablement. Security is an important tenet when it comes to delivering OS and app updates to the device. Most use cases require privacy, integrity, and authenticity attributes for update artifacts to be assured before consumption by devices.

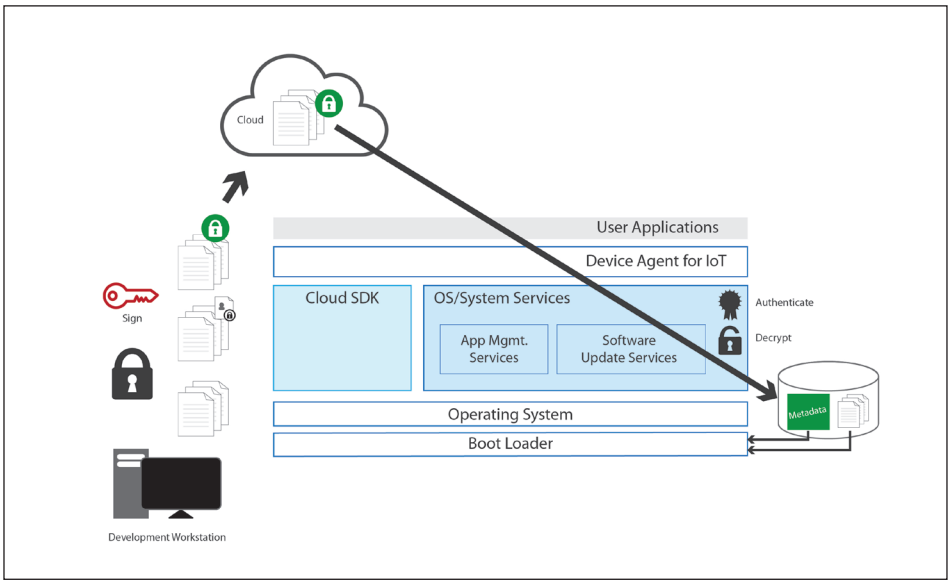


Figure 2: *The process of software updates in an IoT infrastructure.*

As shown in Figure 2, an end-to-end infrastructure is required to enable a secure OS software update workflow, this includes:

- Host tooling for encryption, digital signing, and packaging of update artifacts.
- A cloud or on-premises backend application to deliver updates to the device fleet.
- Device runtime software to receive the update artifacts, authenticate, decrypt, and consume the artifact appropriately.

Cloud vendor-provided enablement does not typically include such capabilities and associated workflows since the update infrastructure needed is largely dependent on the device-specific OS runtime environment and application management requirements.

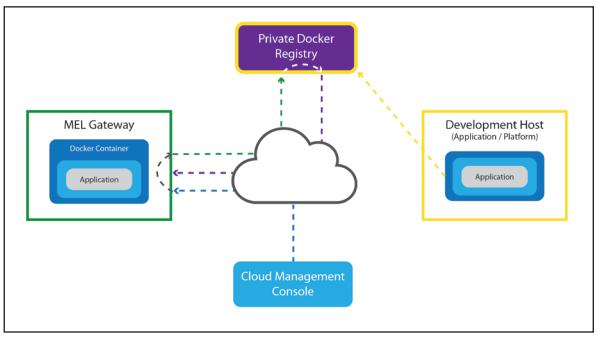
Software update services

Upon receiving an update artifact, the device runtime should be able to parse and determine the nature of that artifact. If the artifact is a firmware update, it should be moved to the boot media and the corresponding boot metadata should be modified so that on the next device reboot, new firmware is booted. A robust firmware update infrastructure should take into account how an update failure would be handled. Health of updated firmware can be assessed using a watchdog that can be asserted by the device application after a successful boot. In case of failure of such an assertion, the system can roll back to a known working firmware version. In terms of optimizing the amount of data moved across the network to deliver updates, data compression, and binary patching technologies should be considered.

Application management services

Application management services are a key element to enabling safe and efficient operation of an IIoT smart device. Depending on the device runtime, application updates can be delivered and consumed as compressed binary images with optional encryption and digital signing, similar to the OS update flow described above.

In the case of Linux- or Windows-based devices, container-based approaches are becoming extremely popular. A container-based approach for application management provides numerous benefits such as portability, ease of migration, scalability, standardization, continuous integration (CI) and delivery (CD), the ability to specify resource requirements, ease of container lifecycle management and health monitoring, and the availability of a strong open source ecosystem of runtime components and tools for management and orchestration of container-based applications.



Application Lifecycle: Cloud to Mentor Embedded Linux (MEL) gateway with Docker registry.

Among the various container implementations, Docker, a leading software containerization platform, has wide adoption in the industry. Figure 3 (on the following page) shows a Docker-based workflow for development, staging, and deployment of applications to edge devices. As depicted, The dev/debug container base-image can be based on the production base-image that consists of the runtime components needed for the application, and includes the development environment and tooling needed for application development. Applications developed can be debugged/validated on the user work station, or on the device in staging, or on the device in deployment on the homogeneous Docker runtime across the environments. Security is an important tenet for application management; Docker provides content trust infrastructure to enable secure packaging and delivery of container artifacts.

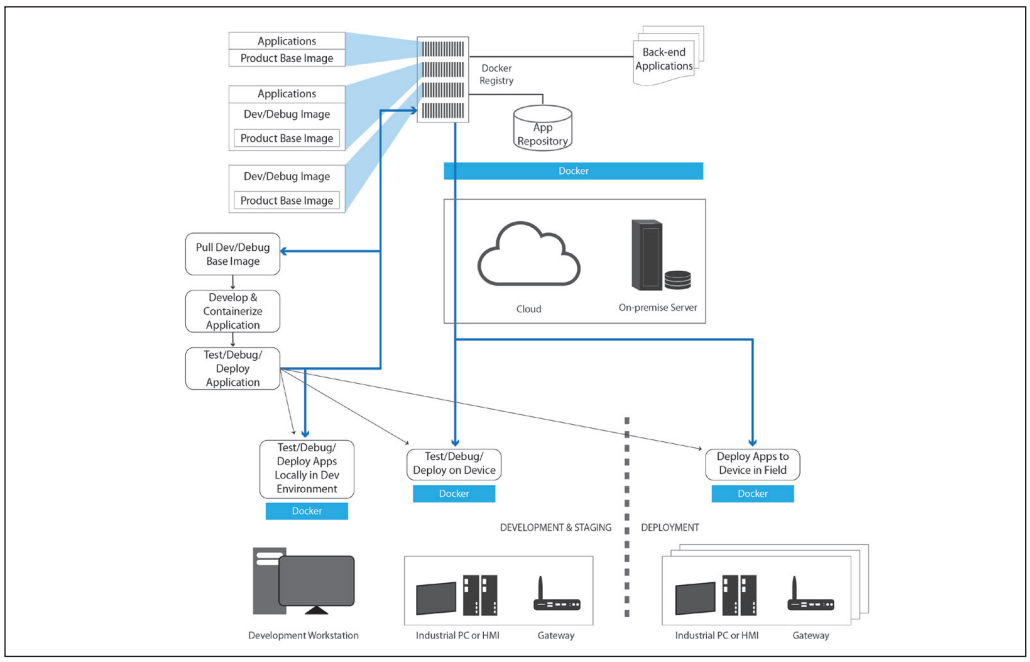
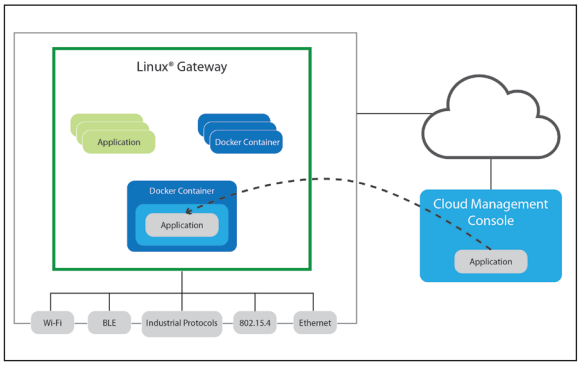


Figure 3: *Device application management using a container-based approach in an IoT infrastructure.*

System diagnostics, health monitoring, and profiling services

IIoT devices deployed in manufacturing environments have to be monitored for system health and performance. In post-mortem debugging scenarios, the ability to profile applications on the device can provide deep insights into the device software runtime. A resident agent on the device can expose device-supported services to back-end applications using the previously discussed data model. Backend applications on the cloud or on-premises can then invoke these methods and consume/present the response returned by the device to the user.



Application Migration: Cloud to Mentor Embedded Linux (MEL) gateway/Docker container.

The device runtime enablement for system health monitoring would have dependencies on the device OS. The resident IoT device agent would tie these underlying services together and expose a consolidated set of services to the backend. For system health monitoring, the device runtime should allow for monitoring of device resource usage such as CPU, memory, storage, networking, interrupt activity, etc. For device applications deployed within containers, it would be useful to enable monitoring of container resource consumption, CPU allocation/usage, system and swap memory allocation/usage, scheduling policies and latencies for real-time applications, etc. In addition to system health monitoring, the IoT device agent could expose hardware and OS-specific software diagnostic services as well.

Secure system services

In IIoT use cases, the convergence of Informational Technology (IT) and Operational Technology (OT) networks has made security of paramount importance. The device runtime should provide the security infrastructure needed for hard isolation of the IT and OT network interfaces so that an external attacker cannot compromise the internal factory network. Suitable firewalls and edge network configuration tools should be employed to enable this isolation.

Device secrets whether PKI certificates for device authentication, or SSL certificates for secure communications, should be maintained and managed within tamper-proof secure memory on the device. There are several approaches to implementing secure management of secrets on IIoT devices.

In x86-based systems, a Trusted Platform Module (TPM) can be used to manage secrets. In most Arm®-based systems, Arm TrustZone® technology could be leveraged. Arm TrustZone allows users to implement secure runtime services in an isolated hardware context called the "Secure World". In TrustZone-enabled systems, a security perimeter can be set up so that secure services, including services to manage device secrets, are deployed in the "Secure World" fully isolated from device software running in the "Non-secure World".

THE ULTIMATE IIoT SMART DEVICE SOLUTION: INTEGRATION OF A CLOUD VENDOR SDK WITH RELEVANT OS/SYSTEM SERVICES

In order to enable comprehensive device management in industrial IoT environments, devices must be able to offer the full set of runtime functionality previously discussed. Figure 4 is an example of a commercial software framework that addresses these demands and requirements. This particular IIoT framework integrates the functions and capabilities of a vendor-provided cloud SDK with the essential functionality of OS/System services of a device runtime environment.

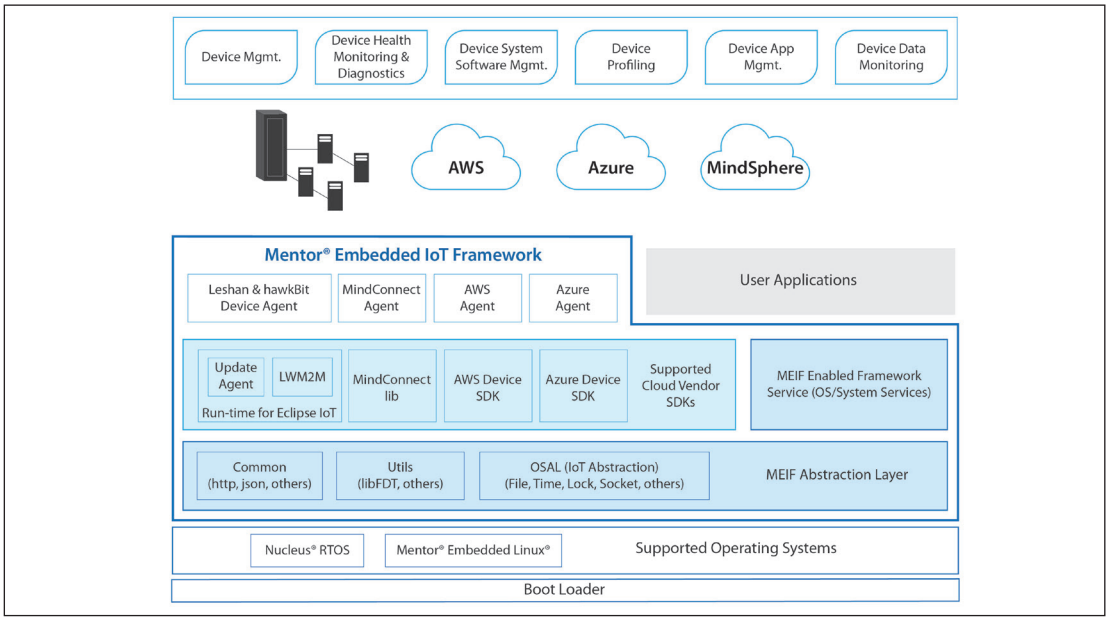


Figure 4: *The Mentor Embedded IoT Framework (MEIF) architecture integrates the capabilities of a cloud vendor-provided SDK with OS/System services to offer a comprehensive end node, to gateway, to cloud IIoT solution.*

The Mentor® Embedded IoT Framework (MEIF) is a new product from Mentor, now a Siemens business. MEIF supports multiple cloud platforms including Amazon Web Services (AWS), Microsoft Azure, and soon, Siemens® MindSphere™. The MEIF architecture also supports Eclipse IoT-based backend applications (e.g. Leshan for device management, and hawkBit for fleet software management) on the cloud or on-premises. It's important to realize Mentor's solution does not replace technologies and investments already provided by a cloud vendor; rather, it aims to fill the gaps between the provided functions of cloud vendor SDKs and the required device functions for device management in IIoT environments.

Figure 4 depicts a high-level illustration of the MEIF architecture. Everything within MEIF pertains to the device or gateway. Everything located above MEIF essentially goes on the cloud or on-premises architecture (local server, gateway, or industrial PC). MEIF integrates the cloud vendor-provided SDK on the device OS environment to ensure seamless operability with the cloud backend. It provides additional services needed for management of the device from the backend; thus, giving IIoT users greater insight and functionality into and throughout the entire IIoT infrastructure.

MEIF-provided functionality includes many of the device functions and operations discussed in this paper: complete, end-to-end workflow for device firmware management; application management; device lifecycle management; and runtime enablement for device operability with several popular open source stacks that provide key backend device management functions. These functions include: hawkBit for fleet software management, Leshan for device management, Grafana for system health monitoring, and ThingsBoard which allows users to create configurable dashboards for data monitoring. These open source stacks can be hosted on any of the supported cloud platforms or on-premises infrastructure.

CONCLUSION

Device manufacturers are facing new challenges related to device management, unknown/multiple clouds, portability, scalability, security, and the need to remotely monitor and diagnose devices.

The Mentor Embedded IoT Framework addresses these challenges and extends the massive investments made by cloud vendors, providing comprehensive IIoT features that can be implemented down to the hardware of the edge or end node devices, and ported across platforms and clouds. The benefits of using a framework such as MEIF are abundantly clear: minimize learning curves, simplify implementations, increase code reuse, and reduce porting, testing and maintenance costs.

For more detailed information on the IIoT productivity gaps, please read the following white paper, "IIoT Evolution: An Approach to Reuse and Scale Your IIoT Technology Investment". For details on this complete IIoT framework solution, please visit the Mentor Embedded IoT Framework website.

The IIoT Ecosystem

- **IPSO Alliance**
  The IPSO Alliance was created in 2008 with the goal of promoting the Internet Protocol (IP) as the means to connect smart objects within the IIoT infrastructure. IPSO is focused on a single approach, single protocol to liberate all connected IIoT devices – including devices at the edge. In 2018, the IPSO Alliance and The Open Mobile Alliance merged to create a new organization called OMA SpecWorks.
- **Docker, Inc.**
  Docker offers a Linux-based computer program that performs OS-level virtualization, or containerization. Docker is designed to help customers digitally transform their IoT content across a diverse set of clouds, platforms, and datacenters. Docker offers independence among these many different assets to help customers unlock the true potential behind the IIoT.
- **Eclipse hawkBit**
  hawkBit is a domain-independent backend framework used for deploying software updates from small, resource-constrained devices up to more powerful controllers, gateways, and servers. hawkBit offers a platform that allows users to create a unique user interface (UI) which enables them to manage and monitor software updates from small groups to company-wide software rollouts.

(Continued on next page)

- **Eclipse Leshan**
  Leshan provides much-needed libraries to help users develop their own lightweight M2M server and client applications. Light-weight M2M is a protocol from The Open Mobile Alliance for IoT device management. The Open Mobile Alliance is now called OMA SpecWorks.
- **Grafana Labs**
  Grafana supports over 30 open source and commercial data sources. The company helps users query, visualize, alert, and understand a diverse set of metrics regardless of where the data is stored. Grafana, with over 150,000 active installations, helps to bring disparate data together through vendor-neutral, open source software.
- **ThingsBoard Inc.**
  ThingsBoard is an open source IoT platform for data collection, processing, visualization, and device management. The technology enables device connectivity using popular industry IoT protocols such as MQTT, CoAP, and HTTP. ThingsBoard users are able to provision and manage both devices and assets along with collecting and visualizing their digital data.

Author biographies:

Arvind Raghuraman is a senior architect for the embedded platform solutions group at Mentor, now a Siemens business. He is a software architect with background and expertise on several Mentor products including; Mentor® Embedded Hypervisor, Mentor® Multicore Framework, Nucleus® RTOS, and development tools. Arvind has over 15 years of experience spread across developing systems and solutions for industrial automation, and software products and solutions for embedded systems. Arvind has his MS degree in Electrical and Computer Engineering from Auburn University, USA.

Scot Morrison is the general manager for the embedded platform solutions group at Mentor, now a Siemens business. Scot oversees the Linux®, Nucleus® RTOS, and Mentor® Embedded Hypervisor runtime product lines, as well as associated tools, middleware, and professional services. Prior to joining Mentor in 2012, Morrison served as GM and SVP of products at Wind River Systems, Inc. Before that Scot worked at Integrated Systems Inc., where he last served as the VP and GM of the design automation solutions business unit in 1999, responsible for MATRIXx, and the pOSEK embedded operating system. Morrison earned his Bachelor of Applied Science degree in Engineering from the University of Toronto, as well as his MS degree in Aerospace Engineering at the Massachusetts Institute of Technology, specializing in control systems.

Corporate Headquarters
Mentor Graphics Corporation
8005 SW Boeckman Road
Wilsonville, OR 97070-7777
Phone: 503.685.7000
Fax: 503.685.1204

Sales and Product Information
Phone: 800.547.3000
sales_info@mentor.com

Silicon Valley
Mentor Graphics Corporation
46871 Bayside Parkway
Fremont, CA 94538 USA
Phone: 510.354.7400
Fax: 510.354.7467

North American Support Center
Phone: 800.547.4303

Europe
Mentor Graphics
Deutschland GmbH
Arnulfstrasse 201
80634 Munich
Germany
Phone: +49.89.57096.0
Fax: +49.89.57096.400

Pacific Rim
Mentor Graphics (Taiwan)
11F, No. 120, Section 2,
Gongdao 5th Road
HsinChu City 300,
Taiwan, ROC
Phone: 886.3.513.1000
Fax: 886.3.573.4734

Japan
Mentor Graphics Japan Co., Ltd.
Gotenyama Trust Tower
7-35, Kita-Shinagawa 4-chome
Shinagawa-Ku, Tokyo 140-0001
Japan
Phone: +81.3.5488.3033
Fax: +81.3.5488.3004

Mentor® A Siemens Business

MGC 06-18 TECH17390-w