# The Three Software Stacks Required for IoT Architectures

*IoT software requirements and how to implement them using open source technology*

iot
eclipse.org

A collaboration of the Eclipse IoT Working Group
*September 2016* *(latest update: December 2017)*

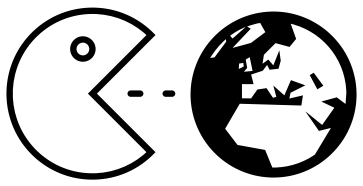
# Contents

# Introduction

The Internet of Things (IoT) is transforming how individuals and organizations connect with customers, suppliers, partners, and other individuals. IoT is all about connecting sensors, actuators, and devices to a network and enabling the collection, exchange, and analysis of generated information.

**Technology innovations in hardware, networking and software are fueling the opportunity for new IoT solutions and use cases**

Hardware innovations, like the Raspberry Pi, are making it easier, faster and cheaper to develop new devices. Networking standards for low power networks, like LoRaWAN or NB-IOT, create new opportunities for connecting very small devices to a network. New standards are being developed specifically for IoT use cases, like MQTT for messaging, OMA Lightweight M2M for device management, or W3C Web of Things and oneM2M for service interoperability. And finally, significant improvements in data storage, data analysis, and event processing are making it possible to support the amount of data generated in large-scale IoT deployments.



*"Software is eating the world"*
*– Marc Andreessen*

In parallel to the emerging IoT industry, the general software industry has moved towards open source as being a key supplier of critical software components. The phrase "software is eating the world" reflects the importance of software in general, but in reality the software industry is now dominated by open source. This is true for key software categories, including Operating Systems (Linux), Big Data (Apache Hadoop, Apache Cassandra), Middleware (Apache HTTP Server, Apache Tomcat, Eclipse Jetty), Cloud (OpenStack, Cloud Foundry, Kubernetes), and Microservices (Docker).

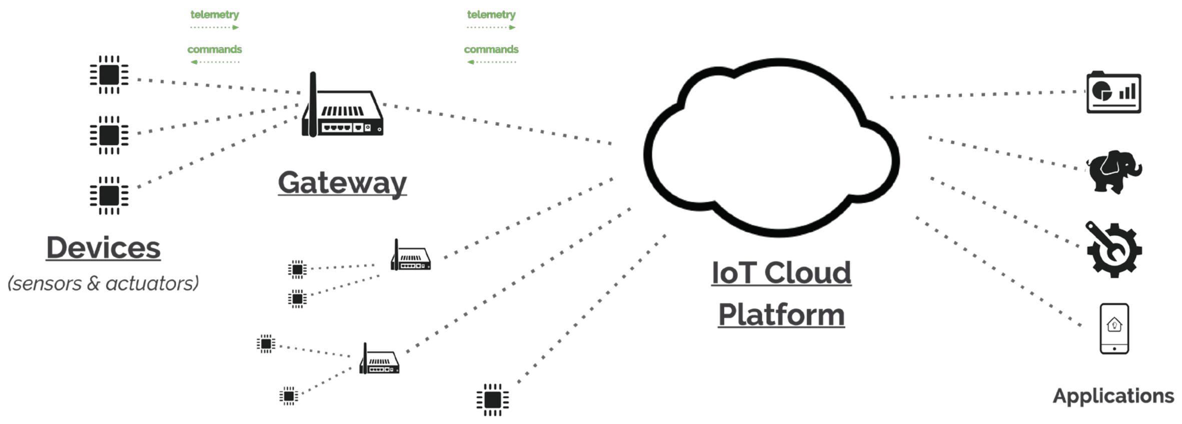
The purpose of this white paper is to look at the new technology requirements and architectures required for IoT solutions. It will identify three stacks of software required by any IoT solution, and finally present how open source communities, such as the Eclipse IoT community, are already supplying the critical software technology needed by IoT solution providers. Similar to how the LAMP (Linux/Apache HTTP Server/MySQL/PHP) stack has dominated the web infrastructures, it is believed a similar open source stack will dominate IoT deployments.

# IoT Architectures

*Devices, Gateways, and IoT Platforms*

# IoT Architectures

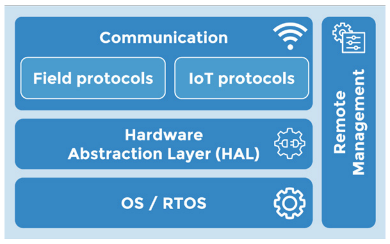
*A typical IoT solution is characterized by many* ***devices*** *(i.e. things) that may use some form of* ***gateway*** *to communicate through a network to an enterprise back-end server that is running an* ***IoT platform*** *that helps integrate the IoT information into the existing enterprise. The roles of the devices, gateways, and cloud platform are well defined, and each of them provides specific features and functionality required by any robust IoT solution.*



## Stack for Constrained Devices

### » Sensors and Actuators

The “Thing” in the IoT is the starting point for an IoT solution. It is typically the originator of the data, and it interacts with the physical world. Things are often very constrained in terms of size or power supply; therefore, they are often programmed using microcontrollers (MCU) that have very limited capabilities. The microcontrollers powering IoT devices are specialized for a specific task and are designed for mass production and low cost.



Software stack for constrained devices

The software running on MCU-based devices aims at supporting specific tasks. The key features of the software stack running on a device may include

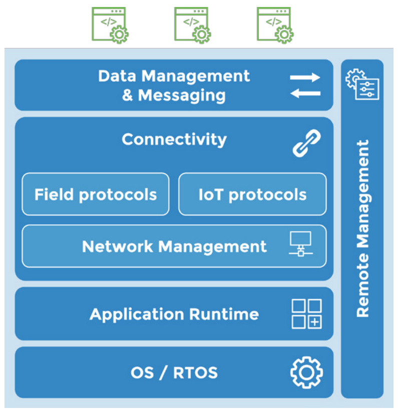
1. **IoT Operating System** – many devices will run with ‘bare metal’, but some will have embedded or real-time operating systems that are particularly suited for small constrained devices, and that can provide IoT-specific capabilities.
2. **Hardware Abstraction** – a software layer that enables access to the hardware features of the MCU, such as flash memory, GPIOs, serial interfaces, etc.
3. **Communication Support** – drivers and protocols allowing to connect the device to a wired or wireless protocol like Bluetooth, Z-Wave, Thread, CAN bus, MQTT, CoAP, etc., and enabling device communication.
4. **Remote Management** – the ability to remotely control the device to upgrade its firmware or to monitor its battery level.

## Stack for Gateways

### » Connected and Smart Things

The IoT gateway acts as the aggregation point for a group of sensors and actuators to coordinate the connectivity of these devices to each other and to an external network. An IoT gateway can be a physical piece of hardware or functionality that is incorporated into a larger “Thing” that is connected to the network. For example, an industrial machine might act like a gateway, and so might a connected automobile or a home automation appliance.

An IoT gateway will often offer processing of the data “at the edge” and storage capabilities to deal with network latency and reliability. For device to device connectivity, an IoT gateway deals with the interoperability issues between incompatible devices. A typical IoT architecture would have many IoT gateways supporting masses of devices.



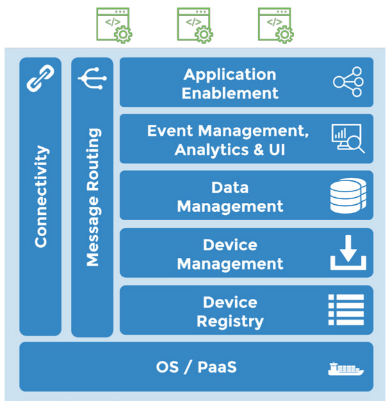
Software stack for gateways

IoT gateways are becoming increasingly dependant on software to implement the core functionality. The key features of a gateway software stack include

1. **Operating System** – typically a general purpose operating system such as Linux.
2. **Application Container or Runtime Environment** – IoT gateways will often have the ability to run application code, and to allow the applications to be dynamically updated. For example, a gateway may have support for Java, Python, or Node.js.
3. **Communication and Connectivity** – IoT gateways need to support different connectivity protocols to connect with different devices (e.g. Bluetooth, Wi-Fi, Z-Wave, ZigBee, Thread). IoT gateways also need to connect to different types of networks (e.g. Ethernet, cellular, Wi-Fi, satellite, etc.…) and ensure the reliability, security, and confidentiality of the communications.
4. **Data Management & Messaging** – local persistence to support network latency, offline mode, and real-time analytics at the edge, as well as the ability to forward device data in a consistent manner to an IoT Platform.
5. **Remote Management** – the ability to remotely provision, configure, startup/shutdown gateways as well as the applications running on the gateways.

## Stack for IoT Cloud Platforms

The IoT Cloud Platform represents the software infrastructure and services required to enable an IoT solution. An IoT Cloud Platform typically operates on a cloud infrastructure (e.g. OpenShift, AWS, Microsoft Azure, Cloud Foundry) or inside an enterprise data center and is expected to scale both horizontally, to support the large number of devices connected, as well as vertically to address the variety of IoT solutions. The IoT Cloud Platform will facilitate the interoperability of the IoT solution with existing enterprise applications and other IoT solutions.



Software stack for IoT Cloud Platforms

The core features of an IoT Cloud Platform include

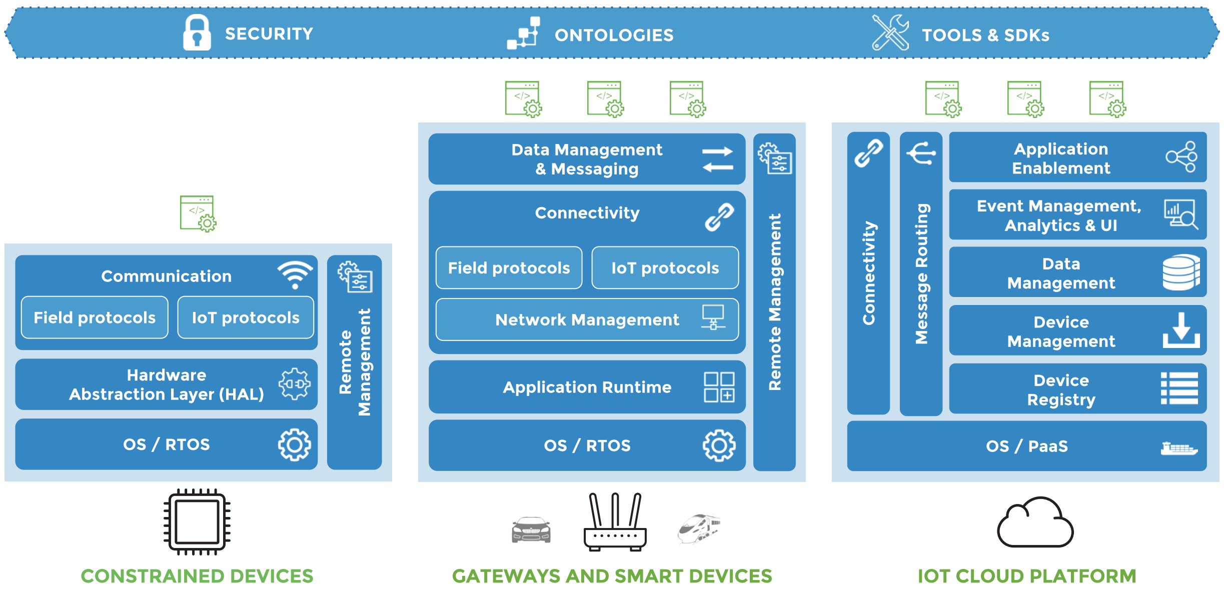
1. **Connectivity and Message Routing** – IoT platforms need to be able to interact with very large numbers of devices and gateways using different protocols and data formats, but then normalize it to allow for easy integration into the rest of the enterprise.
2. **Device Management and Device Registry** – a central registry to identify the devices/gateways running in an IoT solution and the ability to provision new software updates and manage the devices.
3. **Data Management and Storage** – a scalable data store that supports the volume and variety of IoT data.

4. **Event Management, Analytics & UI** – scalable event processing capabilities, ability to consolidate and analyze data, and to create reports, graphs, and dashboards.
5. **Application Enablement** – ability to create reports, graphs, dashboards, ... and to use API for application integration.

## Cross-Stack Functionality

Across the different stacks of an IoT solution are a number of features that need to be considered for any IoT architecture, including

1. **Security** – Security needs to be implemented from the devices to the cloud. Features such as authentication, encryption, and authorization need be part of each stack.
2. **Ontologies** – The format and description of device data is an important feature to enable data analytics and data interoperability. The ability to define ontologies and metadata across heterogeneous domains is a key area for IoT.
3. **Development Tools and SDKs** – IoT Developers will require development tools that support the different hardware and software platforms involved.



## Key Characteristics for IoT Stacks

There are some common characteristics that each IoT stack should embrace, including:

**Loosely coupled** - Three IoT stacks have been defined but it is important that each stack can be used independently of the other stacks. It should be possible to use an IoT Cloud Platform from one supplier with an IoT gateway from another supplier and a third supplier for the device stack.

**Modular** - Each stack should allow for the features to be sourced from different suppliers.

**Platform-independent** - Each stack should be independent of the host hardware and cloud infrastructure. For instance, the device stack should be available on multiple MCUs and the IoT Cloud Platform should run on different Cloud PaaS.

**Based on open standards** - Communication between the stacks should be based on open standards to ensure interoperability.

**Defined APIs** - Each stack should have defined APIs that allow for easy integration with existing applications and integration with other IoT solutions.

## Open Source Technology for IoT

## Open Source Technology for IoT

*The open source community has become an active producer of technology for IoT solutions. Like the LAMP stack for websites, there are a set of open source projects that can be used as the building blocks for an IoT solution architecture.*

*The Eclipse IoT community is very active in providing the technology that can be used in each stack of an IoT solution. Eclipse IoT has 26 different open source projects that address different features of the IoT stacks. In addition to the Eclipse IoT projects, there are other open source projects that are also relevant to an IoT stack. The next few pages provide a brief summary of how Eclipse IoT as well as other open source projects can be used to implement IoT stacks.*

## Open Source Stack for Constrained Devices

Eclipse IoT provides a set of libraries that can be deployed on a constrained embedded device to provide a complete IoT development stack.

- **IoT Operating Systems** – Contiki-NG, RIOT, FreeRTOS, Zephyr, Apache Mynewt.
- **Hardware Abstraction** – Eclipse Edje provides an high-level API for accessing hardware features provided by microcontrollers (e.g GPIO, ADC, MEMS, etc.). It can directly connect to native libraries, drivers, and board support packages provided by silicon vendors.
- **Device Management** – Eclipse Wakaama provides an implementation of the OMA LWM2M standard.
- **Communication** – Open source projects like Eclipse Paho or Eclipse Wakaama provide implementation of IoT communication protocols such as, respectively, MQTT or LWM2M.

## Open Source Stack for Gateways

Within the Eclipse IoT community there are a variety of projects that work to provide the capabilities that an IoT gateway requires.

Eclipse Kura provides a general purpose middleware and application container for IoT gateway services. An IoT gateway stack based on Eclipse Kura would include the following:

- **Operating System** – Linux (Ubuntu/Ubuntu Core, Yocto-based linux distribution), Windows.
- **Application Container or Runtime Environment** – Eclipse Equinox or Eclipse Concierge (OSGi Runtime).
- **Communication and Connectivity** – Eclipse Kura includes APIs to interface with the gateway I/Os (e.g. Serial, RS-485, BLE, GPIO, etc.) and support for many field protocols that can be used to connect to devices, e.g MODBUS, CAN bus, etc.
- **Network Management** – Eclipse Kura provides advanced networking and routing capabilities over a wide-range of interfaces (cellular, Wi-Fi, Ethernet, etc.).
- **Data management & Messaging** – Eclipse Kura implements a native MQTT-based messaging solution, that allows application running on the gateway to transparently communicate with a Cloud Platform, without having to deal with the availability of the network interfaces, or how to represent IoT data. Support for additional messaging protocols is available through the built-in Apache Camel message routing engine.
- **Remote management** – Eclipse Kura provides a remote management solution based on the MQTT protocol, that allows to monitor the overall health of an IoT gateway, in

addition to control (install, update, modify settings) the software it's running.

Eclipse SmartHome provides an IoT gateway platform that is specifically focused on the home automation domain. An Eclipse SmartHome stack would including the following:

- **Operating System** – Linux (Ubuntu/Ubuntu Core, Yocto-based linux distribution), Windows or macOS.
- **Application Container or Runtime Environment** – Eclipse Equinox or Eclipse Concierge (OSGi Runtimes).
- **Communication and Connectivity** – Eclipse SmartHome brings support for many off-the-shelf home automation devices such as Belkin WeMo, LIFX, Philips Hue, Sonos, etc. Eclipse SmartHome focuses on enabling home automation solutions to communicate within an "Intranet of Things" ; therefore offline capabilities are a paramount design goal.
- **Data Management & Messaging** – Eclipse SmartHome has an internal event bus, which can be exposed to external systems through e.g. SSE or MQTT. It furthermore provides mechanisms for persisting values in databases and for running local business logic through a rule engine.
- **Remote Management** – Eclipse SmartHome supports device onboarding and configuration through its APIs. It furthermore provides an infrastructure to perform firmware update of connected devices.

Eclipse 4DIAC provides an industrial-grade open source infrastructure for distributed industrial process measurement and control systems based on the IEC 61499 standard. 4DIAC is ideally suited for Industrie 4.0 and Industrial IoT applications in a manufacturing setting.

The IEC 61499 standard defines a domain specific modeling language for developing distributed industrial control solutions by providing a vendor independent format and for simplifying support for controller to controller communication.

## Open Source Stack for IoT Cloud Platforms

The Eclipse IoT Community has a number of projects that are focused on providing the functionality required for IoT cloud platforms.

**Eclipse Kapua** is a modular platform providing the services required to manage IoT gateways and smart edge devices. Kapua provides a core integration framework and an initial set of core IoT services including a device registry, device management services, messaging services, data management, and application enablement.

The goal of Eclipse Kapua is to create a growing ecosystem of micro services through the extensions provided by other Eclipse IoT projects and organizations.

**Eclipse OM2M** is an IoT Platform specific for the telecommunication industry, based on the oneM2M specification.

It provides a horizontal Common Service Entity (CSE) that can be deployed in an M2M server, a gateway, or a device. Each CSE provides Application Enablement, Security, Triggering, Notification, Persistency, Device Interworking, Device Management.

The Eclipse IoT community also has a number of standalone projects that provide functionality to address key features required for an IoT cloud platform. These projects can be used independently of Eclipse Kapua and over time some may be integrated into Kapua.

### » Connectivity and Protocol Support

- **Eclipse Hono** provides a uniform API for interacting with devices using arbitrary protocols, as well as an extensible framework to add other protocols.
- **Eclipse Mosquitto** provides an implementation of an MQTT broker.

### » Device Management & Device Registry

- **Eclipse Leshan** provides an implementation of the OMA LWM2M device management protocol.
- **Eclipse hawkBit** provides the management tools to roll out software updates to devices and gateways.

### » Event Management & Application Enablement

- **Eclipse Hono** helps to expose consistent APIs for consuming telemetry data or sending commands to devices, so as to rationalize IoT application development.
- **Eclipse Ditto** provides a unified resource-based API that can be used to abstract real-world devices.

### » Analytics and Visualization

- Outside of the Eclipse IoT community there are many open source options for data analytics and visualization, including Apache Hadoop, Apache Spark, and Apache Storm.
- Within the Eclipse community, **Eclipse BIRT** provides support for dashboards and reporting of data stored in a variety of data repositories.

## Open Source for Cross-Stack Functionality

### » Security

- **Eclipse tinydtls** provides an implementation of the DTLS security protocol, providing transport layer security between the device and server.
- **Eclipse Keti** provides an access control service that allows each stack in an IoT solution to protect their resources using a RESTful interface.

### » Ontologies

- **Eclipse Unide** is a protocol for Production Performance Management (PPM) in the manufacturing industry. It establishes an ontology for sharing machine performance information.
- **Eclipse Whiskers** implements the OGC SensorThings API that provides a standard way to share location based information for devices.

### » Development Tools and SDKs

- **Eclipse Vorto** provides a set of tools and repository for creating device information models.
- **Eclipse JDT** and **Eclipse CDT** allow for integrated development of IoT solutions. For example, Eclipse Kura applications can be tested and debugged from within the Eclipse Java IDE (JDT).
- **Eclipse Che** provides a browser-based IDE that can be used for building IoT solutions.Open Source Stack for IoT Cloud Platforms.

## Conclusion

An IoT Solution requires substantial amount of technology in the form of software, hardware, and networking.

In this white paper we have defined the software requirements across three different stacks. For the IoT industry to be successful, it needs to enable more than a succession of independent silos designed by one vendor that address just one business case at the same time. As examples, a connected car comprises MCUs from many different vendors, and a smart city or a smart factory will have a wide variety of sensors and gateways, and an even wider variety of companies looking at building end applications, etc.

The last twenty years have proven that open source software and open source communities are key providers of technology for the software industry. The Internet of Things is following a similar trend, and it is expected that more and more IoT solutions will be built on open source software.

For the past six years, the Eclipse IoT community has been very active in building a portfolio of open source projects that companies and individuals use today to build their IoT solutions.

If you are interested in participating, please join us and visit **https://iot.eclipse.org**.

https://iot.eclipse.org