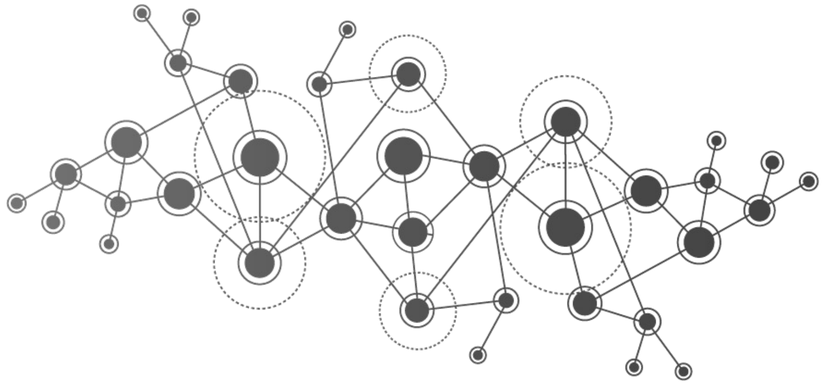
UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA



Intelligent Internet of Things

IoT Hardware & Software

Prof. Marco Picone

A.A 2023/2024

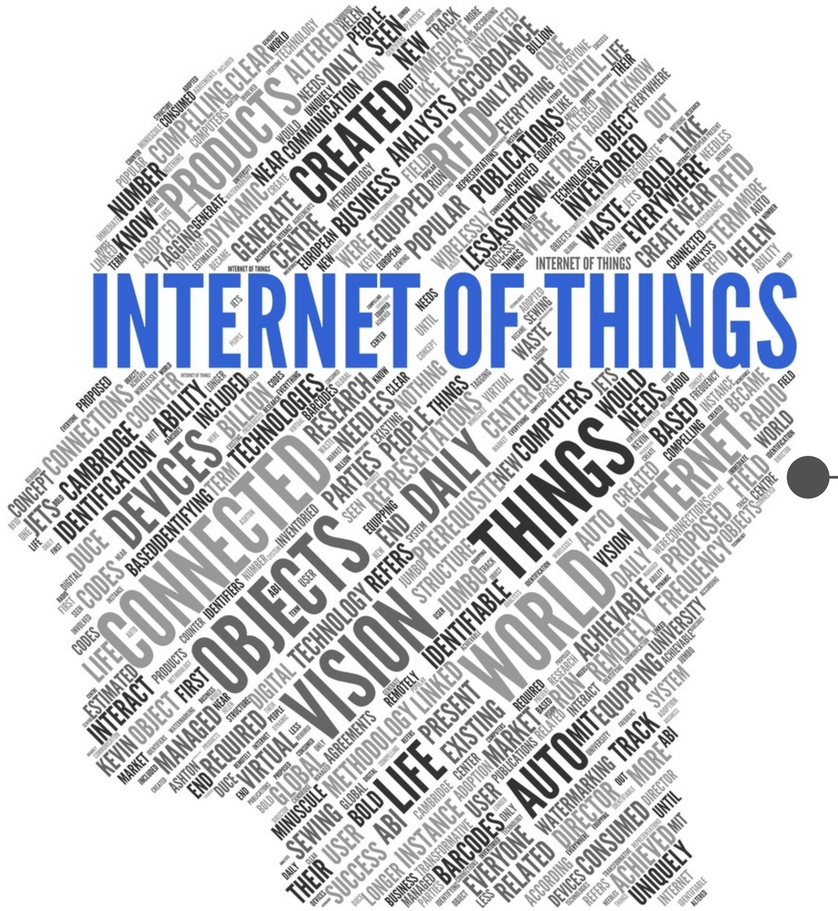
UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

IoT Hardware & Software

- Hardware Overview
- Classes of Constrained Devices
- Main IoT Boards
- Sensors and Actuators
- Operating Systems for Internet of Things
- IoT Software, Libraries and Frameworks

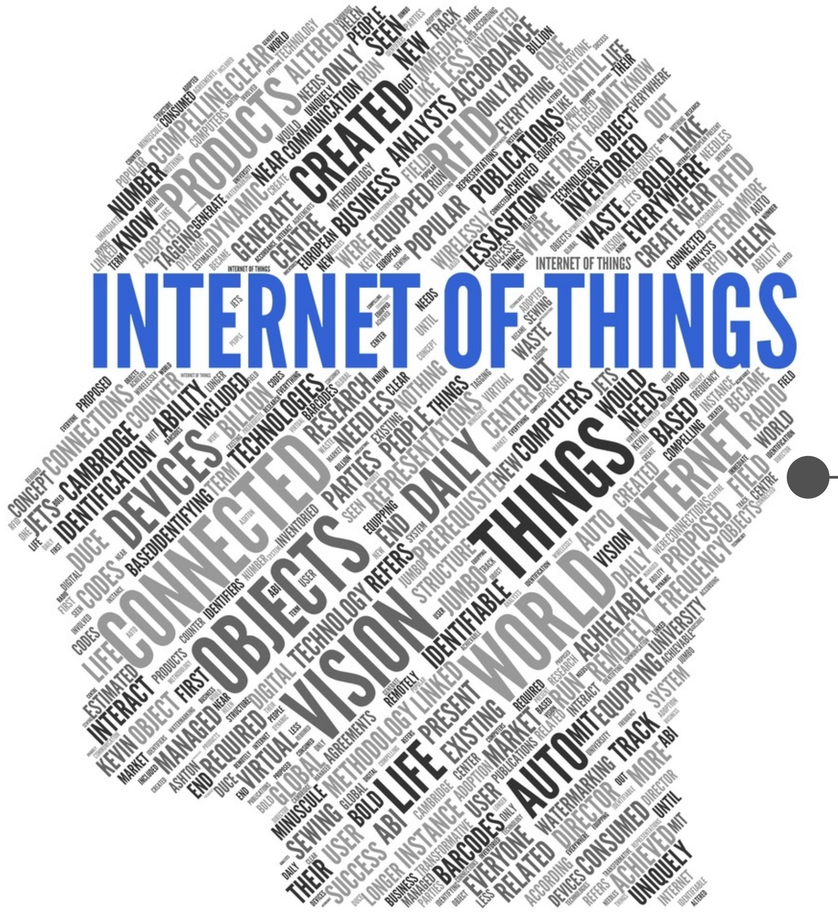
2023 - Marco Picone, Ph.D. (marco.picone@unimore.it)

IoT Heterogeneity



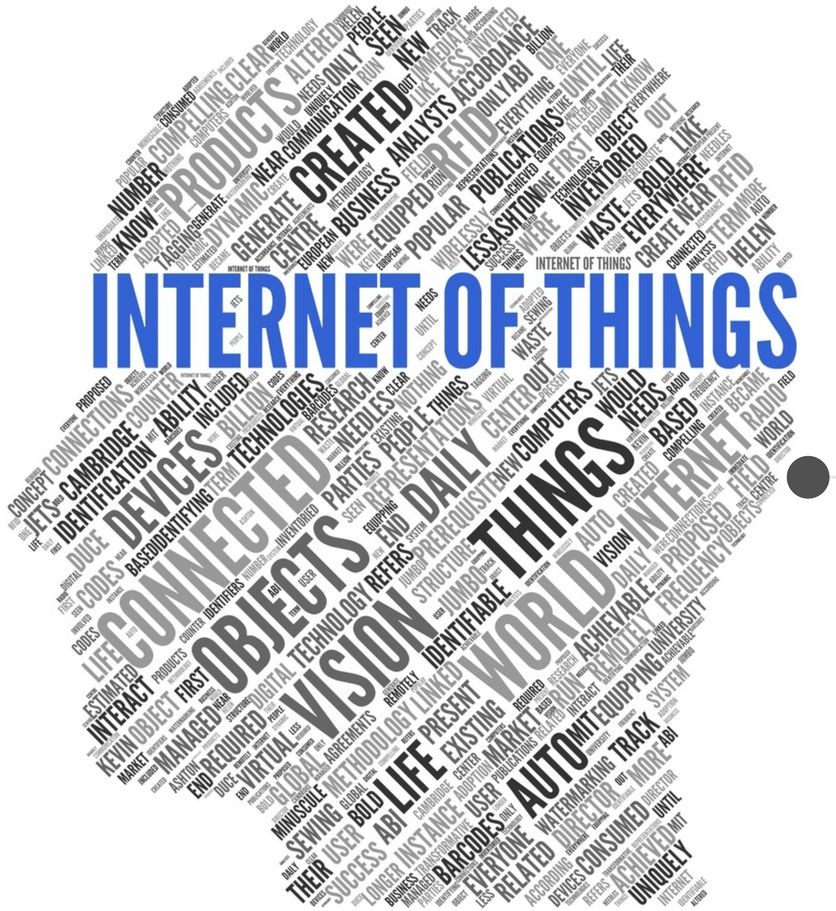
- Communication Technologies
- Protocols
- Hardware
- Software

IoT Heterogeneity



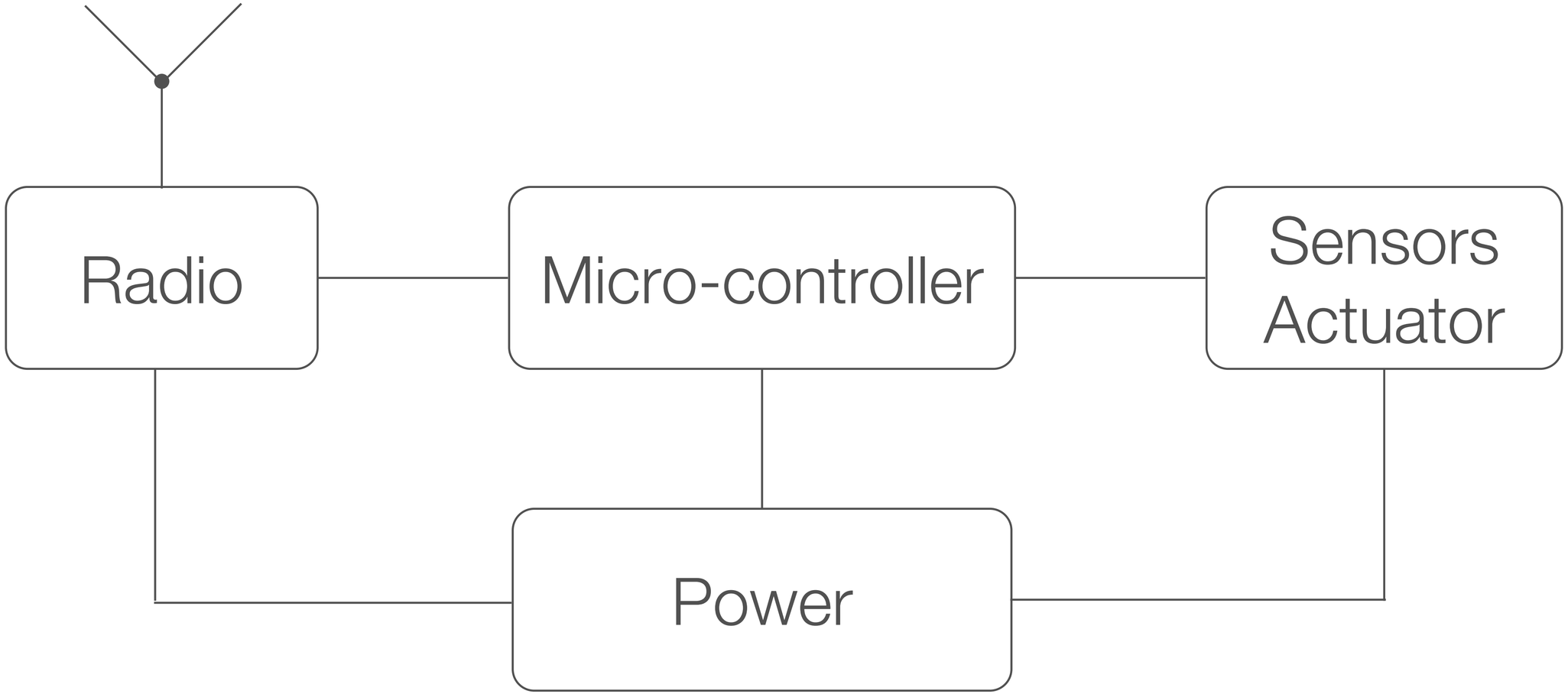
- Communication Technologies
- Protocols
- Hardware
- Software

IoT Heterogeneity

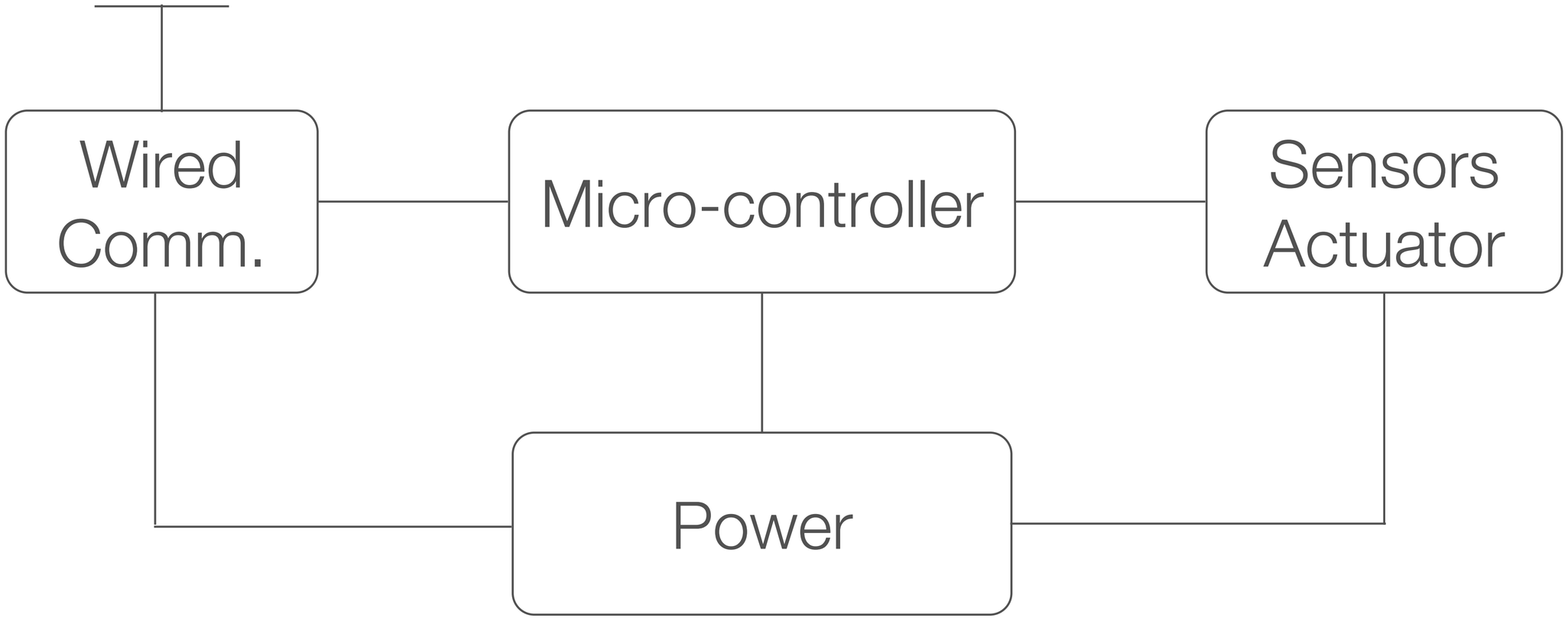


- Communication Technologies
- Protocols
- Hardware
- Software

Smart Object Hardware



Smart Object Hardware



Smart Object Hardware

- **Communication**: This gives the smart object its communication capabilities. It is typically either a radio transceiver with an antenna or a wired connection.
- **Microcontroller**: This gives the smart object its behavior. It is a small microprocessor that runs the software of the smart object.
- **Sensors or actuators**: These give the smart object a way to sense and interact with the physical world.
- **Power source**: This is needed because the smart object contains electrical circuits. The most common power source is a **battery**, but there are other examples as well such as piezoelectric power sources, that provide power when a physical force is applied, or **small solar cells**, that provide power when light shines on them.

Microcontroller

- Microcontrollers have two types of memory: Read-Only Memory (ROM) and Random Access Memory (RAM).
- ROM is used to store the program code that encodes the behavior of the device and RAM is used for temporary data the software needs to do its task.
- **Temporary data include storage for program variables and buffer memory for handling radio traffic.**
- Typically for constrained devices, the content of the ROM is burned into the device when it is manufactured and is not altered after the device has been deployed.
- Modern microcontrollers provide a mechanism for rewriting the ROM, which is useful for in-field updates of software after the devices have been deployed.

Microcontroller - External Devices

- In addition to memory for storing program code and temporary variables, microcontrollers contain a set of timers and mechanisms for interacting with external devices such as communication devices, sensors, and actuators.
- The timers can be freely used by the software running on the microcontroller. External devices are physically connected to the pins of the microcontroller.
- The software communicates with the devices using mechanisms provided by the microcontroller, typically in the form of a **serial port** or a **serial bus**.
- Most microcontrollers provide a so-called **Universal Synchronous/Asynchronous Receiver/Transmitter (USART)** for communication with serial ports. Some USARTs can be configured to work as a Serial Peripheral Interface (SPI) bus for communicating with sensors and actuators.

Power Sources

- A smart object is driven by electronics, and electronics need power. Therefore, every smart object needs a power source.
- Today, the **most common power source is a battery**, but there are several other possibilities for power, such as **solar cells**, **piezoelectricity**, **radio-transmitted energy**, and other forms of power scavenging.
- Lithium cell batteries are currently the most common. With low-power hardware and proper energy-management software, a smart object can have a **lifetime of years on standard lithium cell batteries**.
- Unlike cell phones and laptops, which are human-operated, most smart objects are designed to operate without human control or human supervision. Furthermore, many smart objects are located in difficult to reach places, and many are embedded in other objects. Therefore, in most cases it is impractical to recharge the batteries used in smart objects.

Power Sources

Different Power Sources for Smart Objects, Their Maximum Current Draw, and the Amount of Charge They Store

Power Source	Typical Maximum Current (mA)	Typical Charge (mAh)
CR2032 Button Cell	20	200
AA Alkaline Battery	20	3000
Solar Cell	40	Limitless
RF Power	25	Limitless

Classes of Constrained Devices

- Despite the overwhelming variety of Internet-connected devices that can be envisioned, it may be worthwhile to have some terminology for different classes of constrained devices.

- These characteristics correspond to distinguishable clusters of commercially available chips and design cores for constrained devices.

- It is expected that the boundaries of these classes will move over time, Moore's law tends to be less effective in the embedded space than in personal computing devices: gains made available by increases in transistor count and density are more likely to be invested in reductions of cost and power requirements than into continual increases in computing power.

RFC 7228 - “Terminology for Constrained-Node Networks”: https://tools.ietf.org/html/rfc7228

Classes of Constrained Devices

- Despite the overwhelming variety of Internet-connected devices that can be envisioned, it may be worthwhile to define terminology for different classes of constrained devices.

Name	Data Size (e.g., RAM)	Code Size (e.g., Flash)
Class 0, C0	<< 10 KiB	<< 100 KiB
Class 1, C1	~ 10 KiB	~ 100 KiB
Class 2, C2	~ 50 KiB	~ 250 KiB

- These characteristics correspond to distinguishable clusters of commercially available chips and design cores for constrained devices.

RFC 7228 - “Terminology for Constrained-Node Networks”: https://tools.ietf.org/html/rfc7228

Classes of Constrained Devices

- **It is expected that the boundaries of these classes will move over time**, Moore's law tends to be less effective in the embedded space than in personal computing devices: gains made available by increases in transistor count and density are more likely to be invested in reductions of cost and power requirements than into continual increases in computing power.

- Constrained devices with capabilities significantly beyond Class 2 devices exist. They are less demanding from a standards development point of view as they can largely use existing protocols unchanged.

RFC 7228 - “Terminology for Constrained-Node Networks”: https://tools.ietf.org/html/rfc7228

Classes 0 Devices

Name	Data Size (e.g., RAM)	Code Size (e.g., Flash)
Class 0, C0	<< 10 KiB	<< 100 KiB

- Class 0 devices are very constrained sensor-like motes.
- They are so severely constrained in memory and processing capabilities that most likely they will not have the resources required to communicate directly with the Internet in a secure manner.
- **Class 0 devices will participate in Internet communications with the help of larger devices acting as proxies, gateways, or servers.**
- Class 0 devices generally cannot be secured or managed comprehensively in the traditional sense. **They will most likely be preconfigured (and will be reconfigured rarely, if at all) with a very small data set.** For management purposes, they could answer keepalive signals and send on/off or basic health indications.

Classes 1 Devices

Name	Data Size (e.g., RAM)	Code Size (e.g., Flash)
Class 1, C1	~ 10 KiB	~ 100 KiB

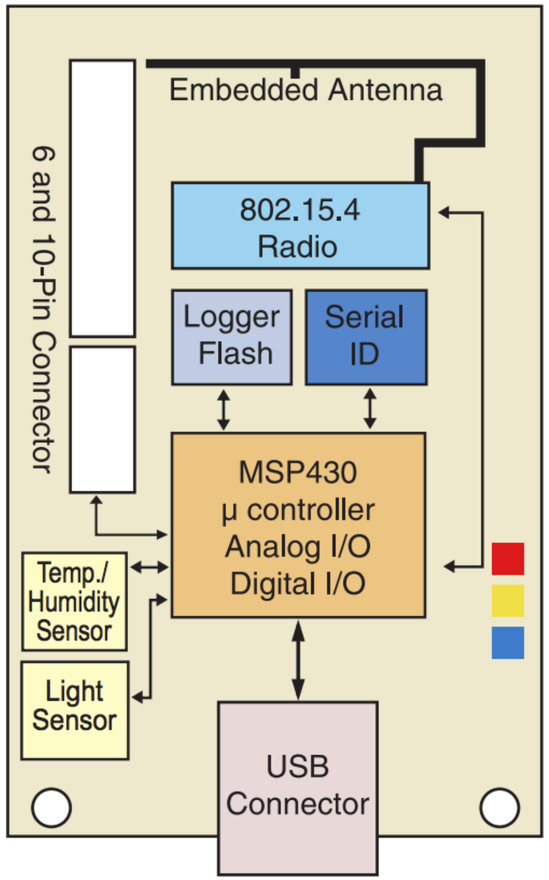
- Class 1 devices are quite constrained in code space and processing capabilities, such that they cannot easily talk to other Internet nodes employing a full protocol stack such as using HTTP, Transport Layer Security (TLS), and related security protocols and XML-based data representations.
- They are capable enough to use a protocol stack specifically designed for constrained nodes (such as the **Constrained Application Protocol (CoAP) over UDP [COAP]**) and participate in **meaningful conversations without the help of a gateway node**.
- They can provide support for the security functions required on a large network. Therefore, they can be integrated as fully developed peers into an IP network, but they need to be parsimonious with state memory, code space, and often power expenditure for protocol and application usage.

Classes 2 Devices

Name	Data Size (e.g., RAM)	Code Size (e.g., Flash)
Class 2, C2	~ 50 KiB	~ 250 KiB

- Class 2 devices are less constrained and fundamentally capable of supporting most of the same protocol stacks as used on notebooks or servers.
- They can benefit from lightweight and energy-efficient protocols and from consuming less bandwidth.
- Using fewer resources for networking leaves more resources available to applications. Thus, using the protocol stacks defined for more constrained devices on Class 2 devices might reduce development costs and increase the interoperability.

MemSic - TELOSB MOTE PLATFORM (Telos B)



Name	Description
MCU	TI MSP430F1611
RAM	10 KB
ROM	48 KB
Serial Comm.	UART
Main Module Current Draw	1.8 mA (Active Mode) 5.1 µA (Sleep Mode)
IEEE 802.15.4 compliant RF transceiver	2400 MHz to 2483.5 MHz
RF Current Draw	23 mA (Receive Mode), 21µA (Idle Mode), 1µA (Sleep Mode)
Battery	2X AA Batteries
Sensors	Visible Light Sensor, Humidity Sensor, Temperature Sensor

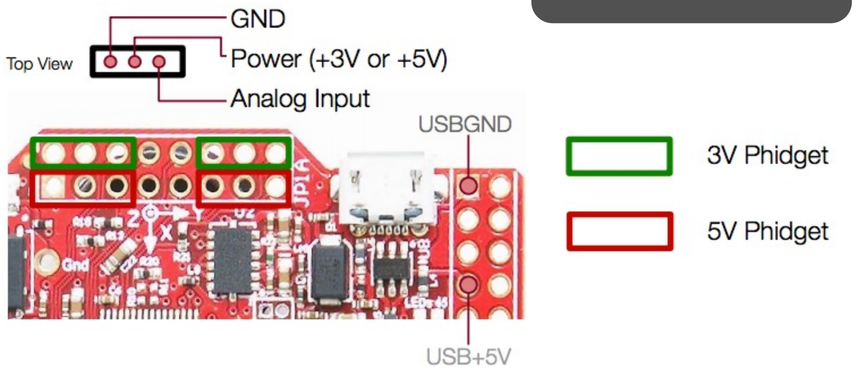
Contiki OS

http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf

Zolertia Z1 Platform

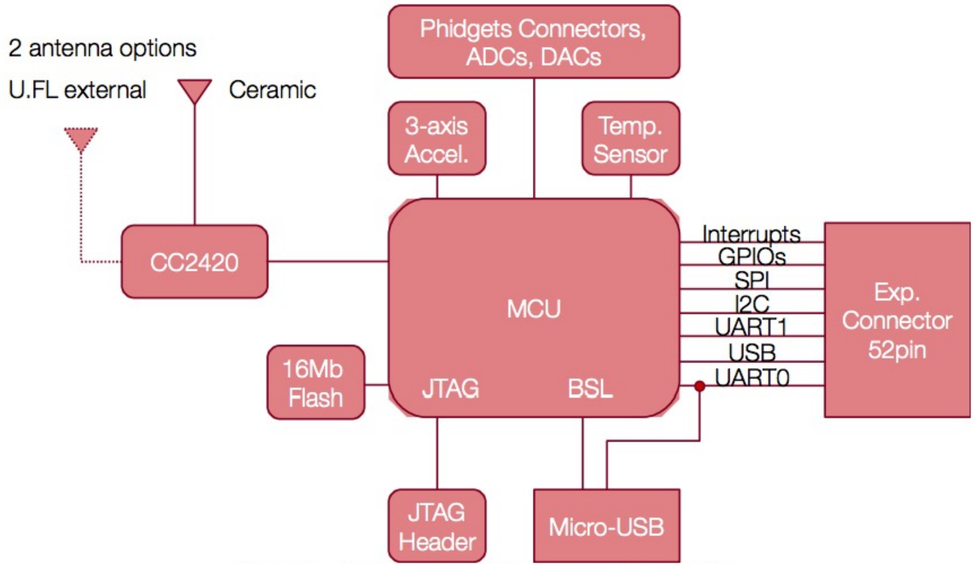
Name	Description
MCU	TI MSP430F2617
RAM	8 KB
ROM	92 KB
Digital Comm.	I2C, SPI and UART
Main Module Current Draw	0.5 mA (Active Mode) 0.5 µA (Standby Mode)
IEEE 802.15.4 compliant RF transceiver	CC2420 2.4 GHz
RF Current Draw	18.8 mA (Receive Mode) 426 µA (Idle Mode) 20 µA (Sleep Mode)
Battery	2xAA or AAA cells 1xCR2032 coin cell
Sensors	Low-power digital temperature sensor 3-Axis, ±2/4/8/16 g digital accelerometer 3V and 5V Phidget Sensors Connectors

Contiki OS



http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf

Zolertia Z1 Platform



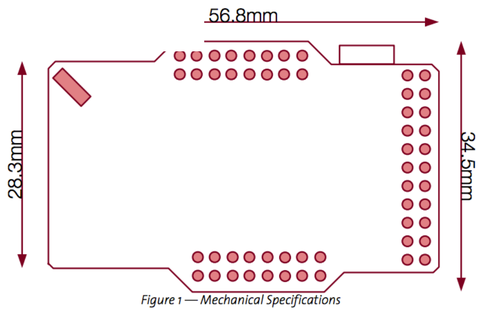


Figure 1 — Mechanical Specifications

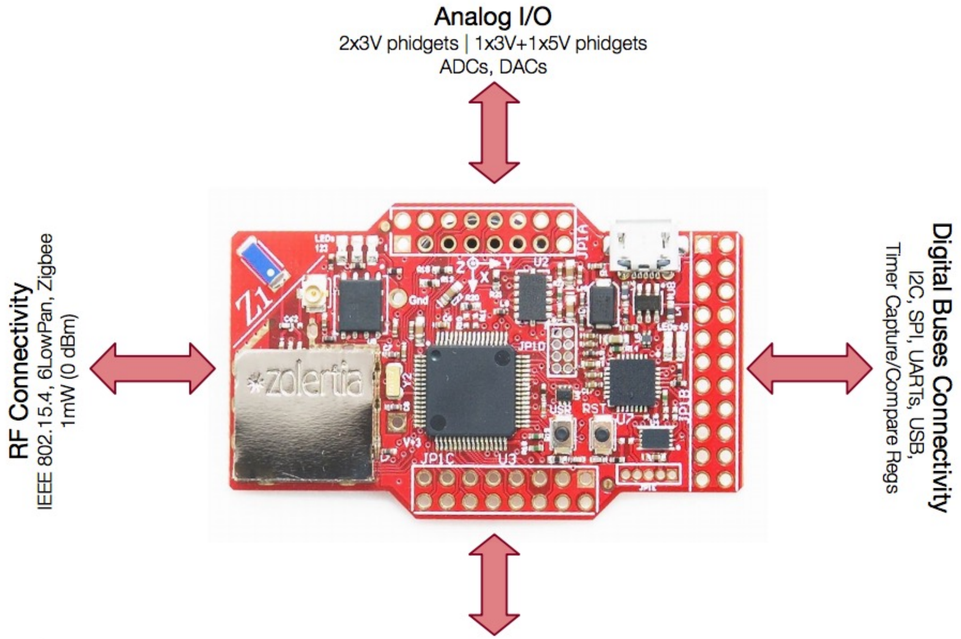
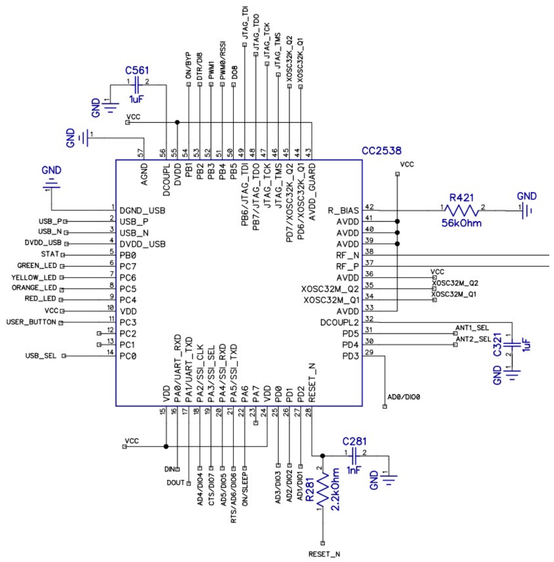


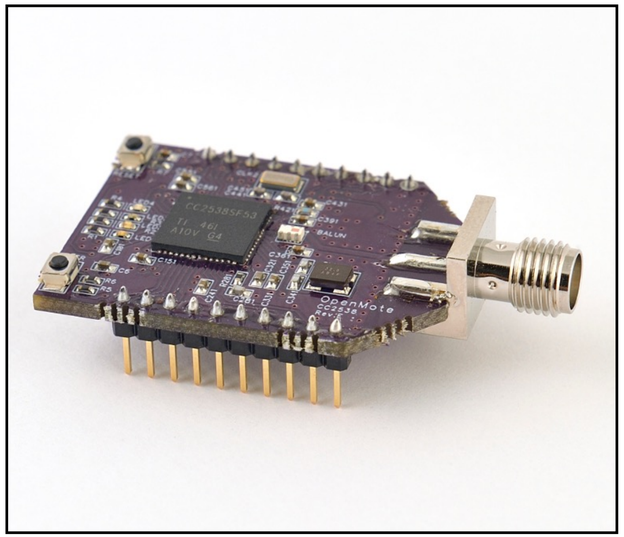
Figure 3 — Z1 Expansion Capabilities (54-pin XPCon)

http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf

OpenMote CC2538

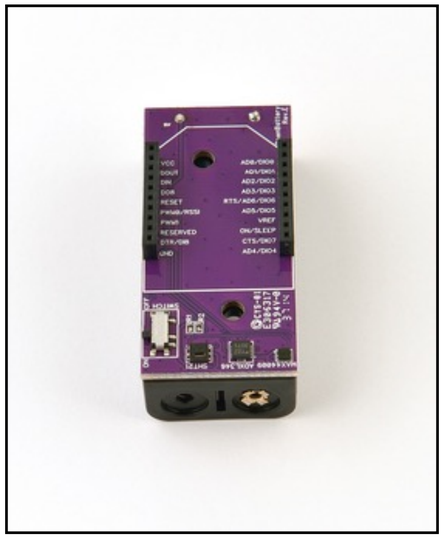
openmote
openhardwarefortheinternetofthings





CC2538

OpenBase Module



OpenBattery Module

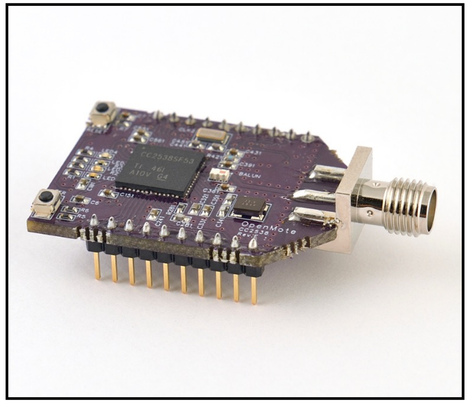
http://www.openmote.com/hardware/openmote-cc2538-en.html

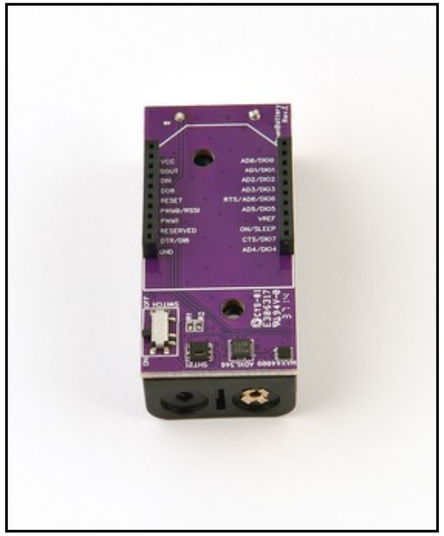
http://www.openmote.com/wp-content/uploads/2014/01/OpenMote-CC2538-Schematic.pdf

OpenMote CC2538

Contiki OS | OpenWSN | FreeRTOS | RiOT

Name	Description
MCU	TI 32-bit Cortex-M3
RAM	32 KB
ROM	512 KB
Digital Comm.	I2C and UART
Main Module Current Draw	0.5 mA (Active Mode) 0.5 µA (Standby Mode)
IEEE 802.15.4 compliant RF transceiver	CC2520 2.4 GHz
Battery (OpenBattery Module)	2x AAA cells
Sensors (OpenBattery Module)	Temperature/humidity sensor (SHT21) Acceleration sensor (ADXL346) Light sensor (MAX44009)



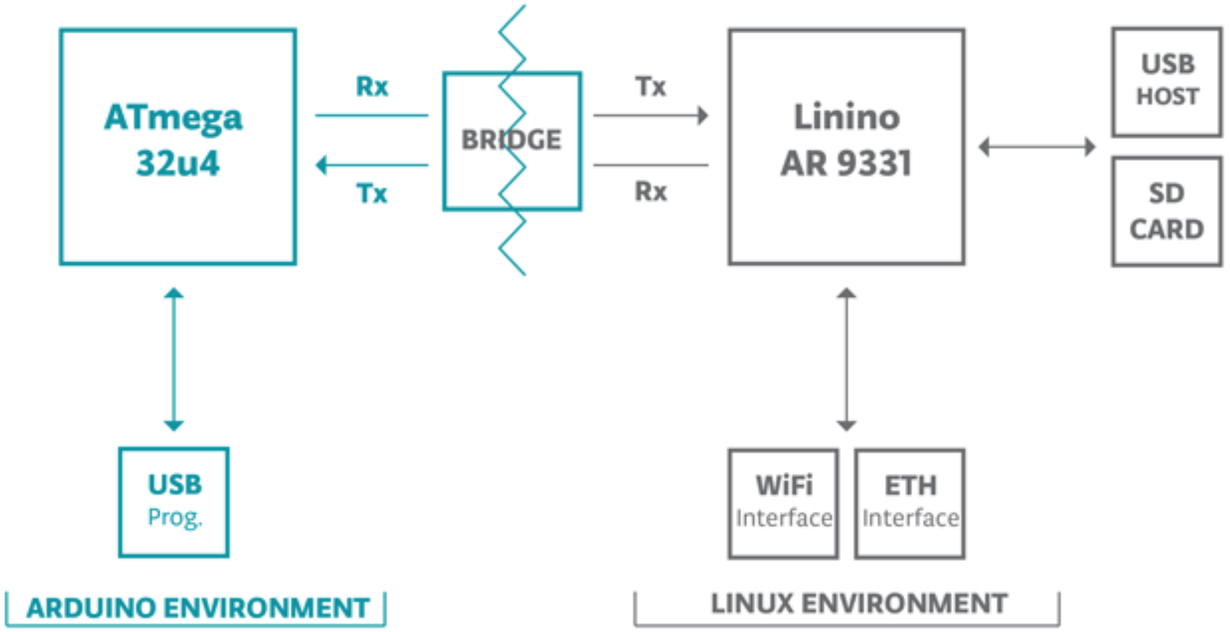


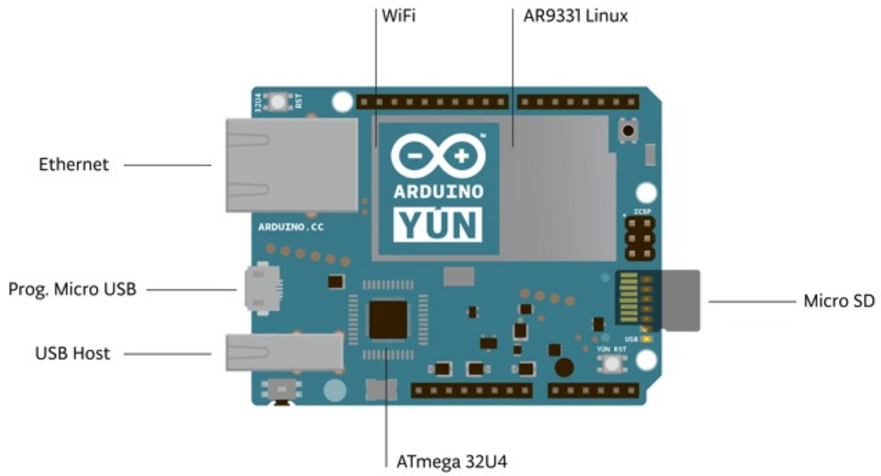
Arduino

- “Arduino is an open-source computer hardware and software company, project and user community that designs and manufactures kits for building digital devices and interactive objects that can sense and control the physical world.”
- The project is based on a family of microcontroller board designs manufactured primarily by SmartProjects in Italy.
- It is characterized by easy development (C-like programming languages) and heterogeneous hardware.
- In the huge Arduino family they created specific boards dedicated to the IoT world such as:
  - Arduino Yun
  - Intel Galileo (collaboration with Intel)
  - Arduino 3

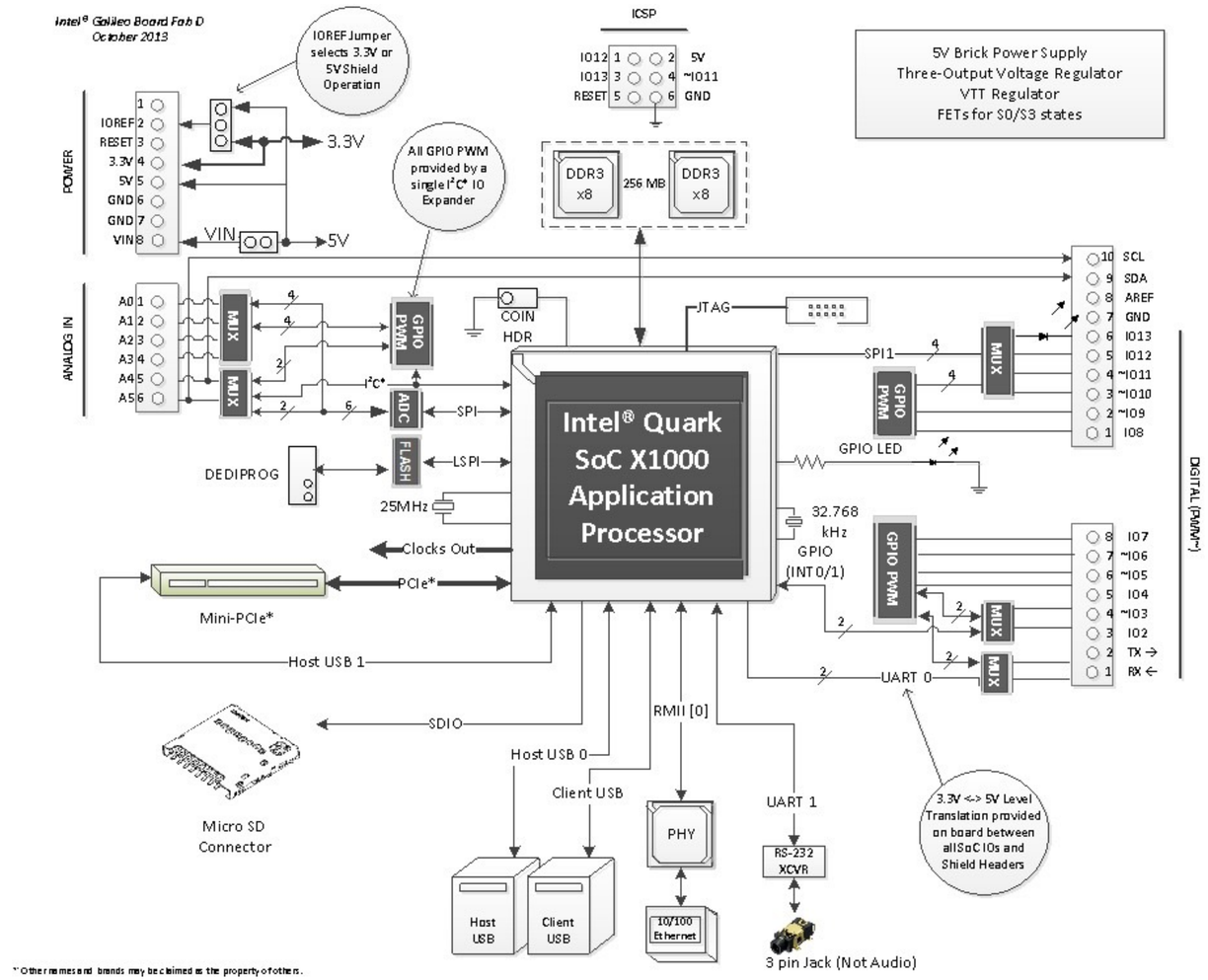
http://www.arduino.cc/

Arduino Yún

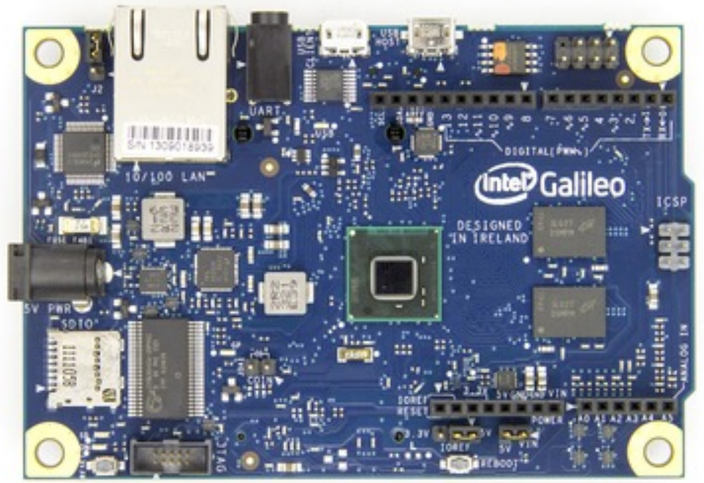

ATmega
32u4
Rx
Tx
BRIDGE
Tx
Rx
Linino
AR 9331
USB
HOST
SD
CARD
USB
Prog.
WiFi
Interface
ETH
Interface
ARDUINO ENVIRONMENT
LINUX ENVIRONMENT


WiFi
AR9331 Linux
Ethernet
Prog. Micro USB
USB Host
Micro SD
ATmega 32U4
ARDUINO
YÚN
ARDUINO.CC

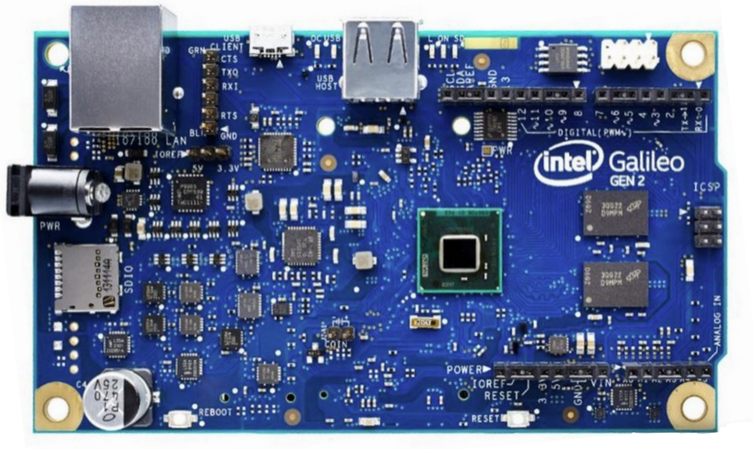
Intel Galileo



Gen 1



Gen 2



http://www.intel.com/content/www/us/en/do-it-yourself/galileo-maker-quark-board.html http://arduino.cc/en/ArduinoCertified/IntelGalileo

Intel Galileo

Linux

Name	Description
MCU	SoC X Intel® Quark™ X1000
RAM	256 MB
Memory	SDCard (GBytes)
Digital Comm.	I2C and UART
Main Module Current Draw	0.5 mA (Active Mode) 0.5 µA (Standby Mode)
Network Adapter	IEEE 802.3 10/100 (Ethernet)
Battery	-
Sensors	-

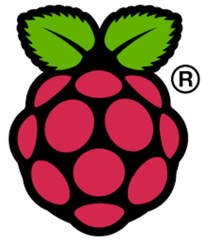
- It has been the first board based on Intel® architecture designed to be hardware and software pin-compatible with Arduino shields
- Digital pins 0 to 13 (and the adjacent AREF and GND pins), Analog inputs 0 to 5, the power header, ICSP header, and the UART port pins (0 and 1), are all in the same locations as on the Arduino Uno R3
- Galileo board is also software compatible with the Arduino Software Development Environment (IDE)
- The software release supports the following Arduino libraries:SPI, EEPROM, UART, GPIO, Wi-Fi, Servo, USB Host
- It is compatible with several Arduino Shields

Raspberry Pi

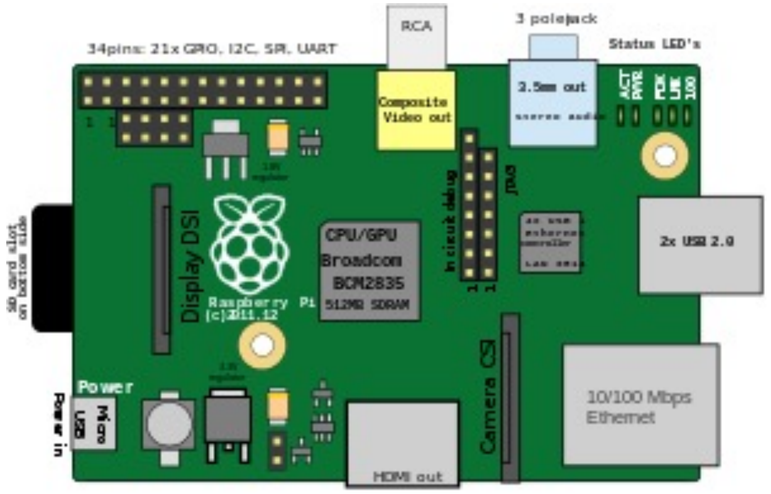
"The Raspberry Pi is a series of credit card-sized single-board computers developed in the UK by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools."

	Model A	Model A+	Model B	Model B+	Generation 2 Model B	Compute Module Note: all interfaces are via 200-pin DDR2 SO-DIMM connector.
Target price:	US$25	US$20[30]	US$35[31][32]			US$30 (in batches of 100)[33]
SoC:	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, one USB port)[1][33]				Broadcom BCM2836 (CPU, GPU, DSP, SDRAM, one USB port)	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, one USB port)[1][33]
CPU:	700 MHz single-core ARM1176JZF-S[1]				900 MHz quad-core ARM Cortex-A7	700 MHz single-core ARM1176JZF-S
GPU:	Broadcom VideoCore IV @ 250 MHz[34][35] OpenGL ES 2.0 (24 GFLOPS) MPEG-2 and VC-1 (with license),[36] 1080p30 H.264/MPEG-4 AVC high-profile decoder and encoder[1]					
Memory (SDRAM):	256 MB (shared with GPU)		512 MB (shared with GPU) as of 15 October 2012		1 GB (shared with GPU)	512 MB (shared with GPU)

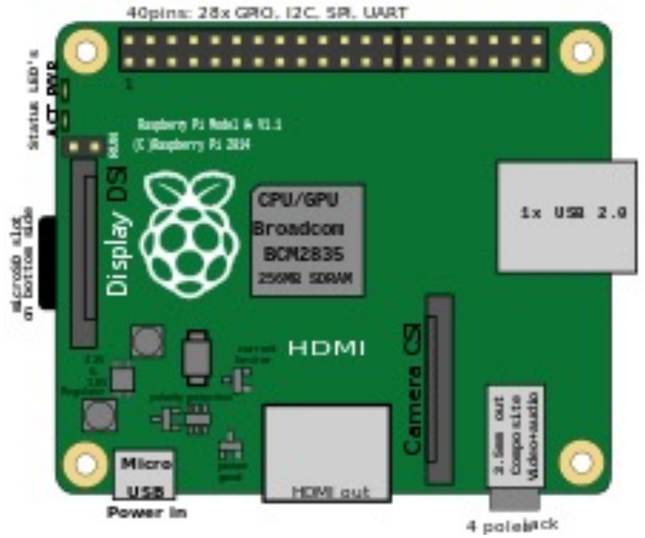
https://www.raspberrypi.org/



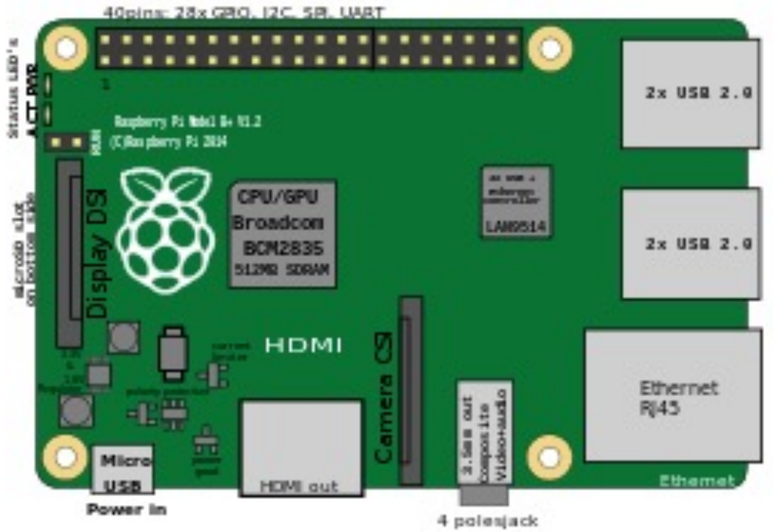
Raspberry Pi



Raspberry Pi 1 model B revision 2



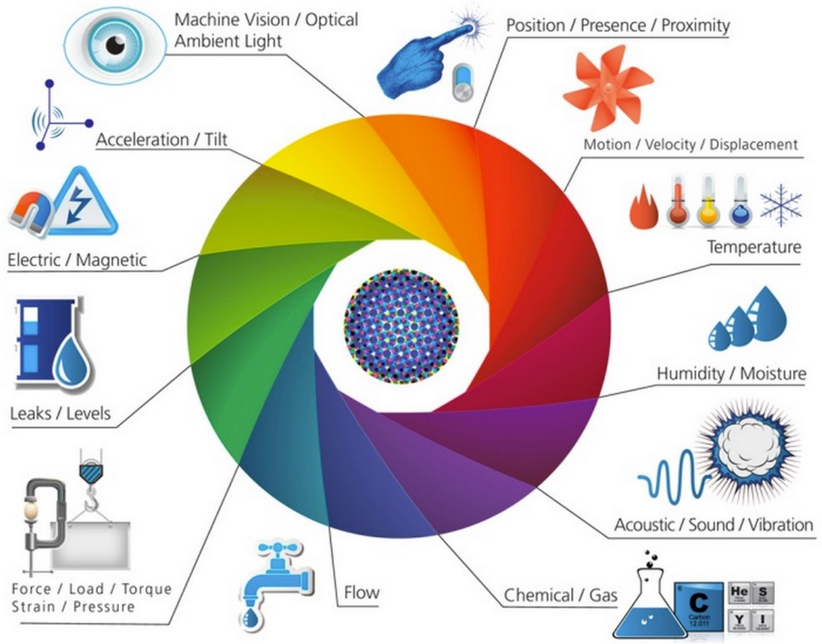
Raspberry Pi 1 model A+ revision 1.1



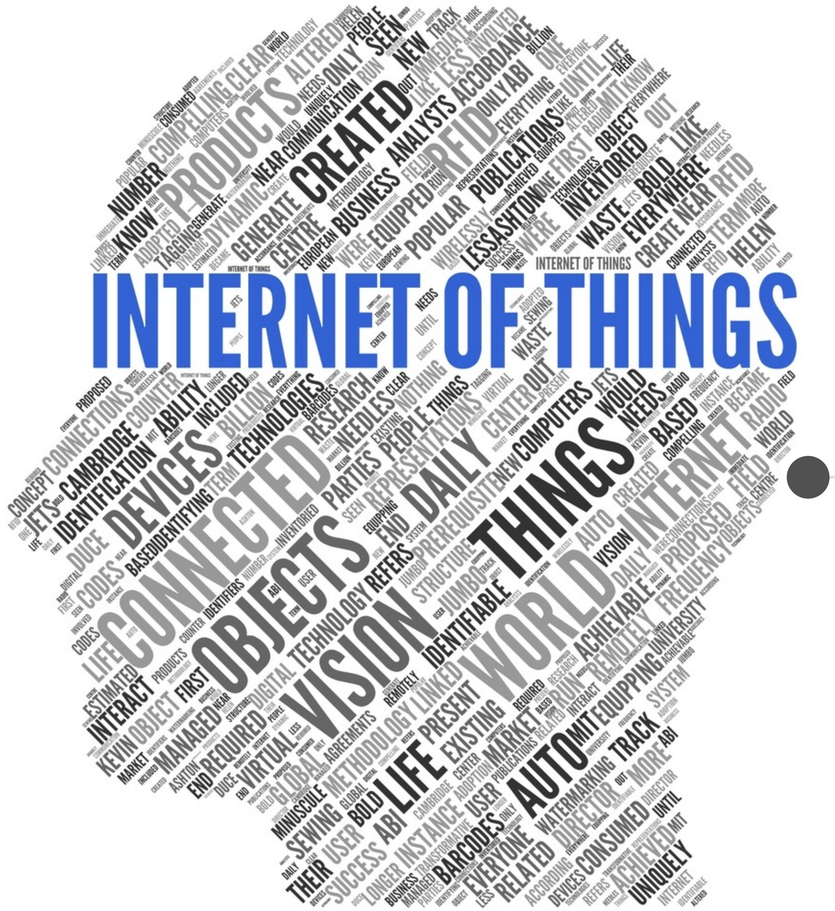
Raspberry Pi 1 model B+ revision 1.2
and Raspberry Pi 2 model B

Sensors and Actuators

- Smart objects interact with the physical environment in which they are deployed by using sensors and actuators.
- Sensors are used to sense the environment and actuators are used to affect or change the environment.
- The sensors and actuators attached to a smart object range from very simple to very complex.
- A smart object that measures the temperature needs only a simple temperature sensor. Conversely, a smart object used for surveillance or detection of people crossing a fence may need a set of sensors that include an ultrasonic range device or a camera.



IoT Heterogeneity



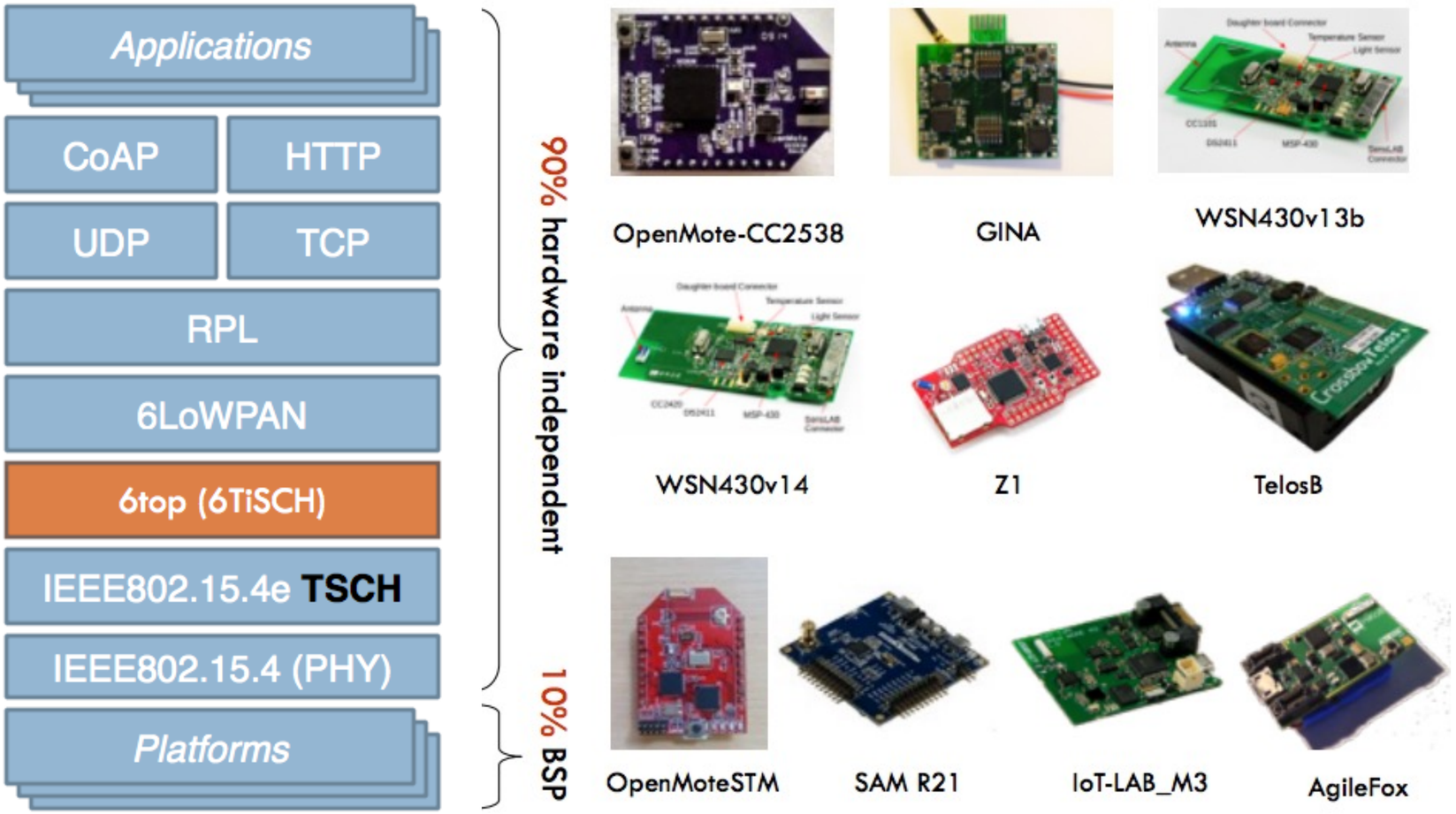
- Communication Technologies
- Protocols
- Hardware
- Software

OpenWSN

- The OpenWSN project is an open-source implementation of a fully standards-based protocol stack for Internet of Things networks
- I has been based on the new IEEE802.15.4e Time Slotted Channel Hopping standard. IEEE802.15.4e, coupled with Internet-of-Things standards, such as 6LoWPAN, RPL and CoAP, enables ultra-low power and highly reliable mesh networks which are fully integrated into the Internet.
- OpenWSN is ported to numerous commercial available platforms from older 16-bit micro-controller to state-of-the-art 32-bit Cortex-M architectures. networks while contributing with innovative protocols for scalable, distributed and energy efficient communications.
- The OpenWSN project offers a free and open-source implementation of a protocol stack and the surrounding debugging and integration tools, thereby contributing to the overall goal of promoting the use of low-power wireless mesh networks.

https://openwsn.atlassian.net/wiki/pages/viewpage.action?pageId=688187

OpenWSN



https://www.iot-lab.info/wp-content/uploads/2014/11/141106_openwsn_iotlab_public.pdf

TinyOS

- TinyOS is a free, open-source, BSD-licensed OS designed for low-power embedded distributed wireless devices used in sensor networks.
- It has designed to support the concurrency intensive operations required by networked sensors with minimal hardware requirements.
- TinyOS has been developed by University of California, Berkeley, Intel Research and Crossbow Technology.
- It is written in **nesC** programming language
- **n**etwork **e**mbedded **s**ystems **C**, is a C optimized to support components and concurrency. It is a component based, event driven programming language used to build application for TinyOS platform.

http://www.tinyos.net/

FreeRTOS

- FreeRTOS is a real-time operating system kernel for embedded devices designed to be small and simple.
- It been ported to 35 micro controllers and it is distributed under the GPL with an optional exception. The exception permits users' proprietary code to remain closed source while maintaining the kernel itself as open source, thereby facilitating the use of FreeRTOS in proprietary applications.
- In order to make the code readable, easy to port, and maintainable, it is written mostly in C, (but some assembly functions has been included to support architecture-specific scheduler routines).
- It provides methods for multiple threads or tasks, mutexes, semaphores and software timers.

http://www.freertos.org/

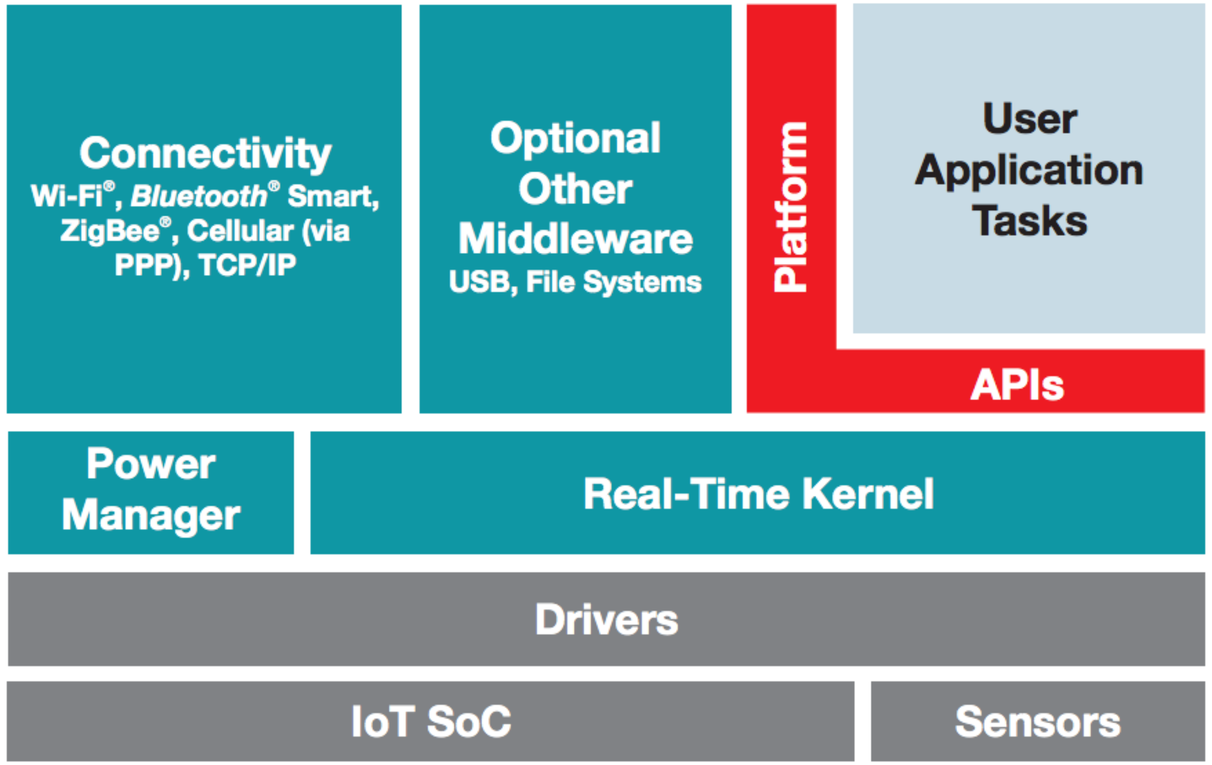
TI-RTOS

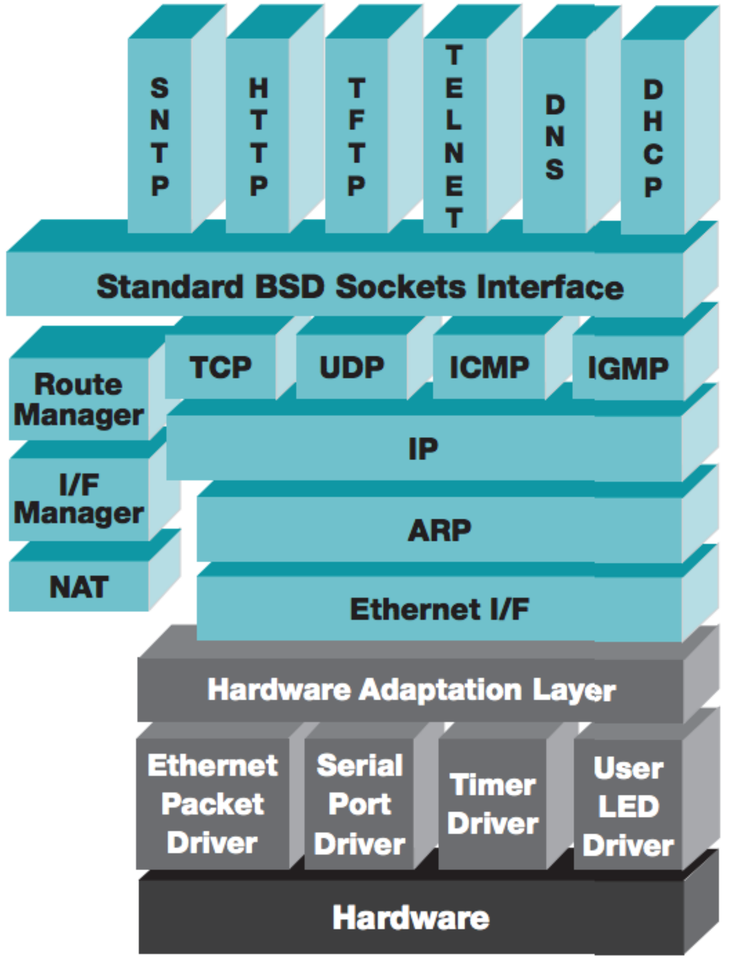
- TI-RTOS is a real-time operating system that enables faster development by eliminating the need for developers to write and maintain system software such as schedulers, protocol stacks, power management frameworks and drivers. It is provided with full C source code and requires no up-front or runtime license fees.
- TI-RTOS scales from a **low-footprint, real-time preemptive multitasking kernel** to a complete RTOS with additional middleware components including a power manager, **TCP/IP and USB stacks, a FAT file system and device drivers, allowing developers to focus on differentiating their application.**

TI-RTOS Kernel services	Description
Task	Independent, pre-emptible thread of execution that has its own stack and can yield the processor
SWI	Pre-emptible thread that uses program stack but cannot yield
Clock	Time-triggered periodic functions
Event	Wait on multiple events (semaphore, mailbox, I/O, user-defined, …)
Mailbox	Mailboxes for synchronized fixed-sized data exchange between tasks
Semaphore	Counting and binary semaphores
Gate	Protects against concurrent access to critical data structures
Heaps	Variable-sized heap, fixed-sized buffers, variable-sized allocation based on multiple fixed-sized buffer pools
HWI	Interrupt management
Timer	Interface to hardware timers, supporting one-shot or continuous callbacks
Diagnostics	Logs, timestamps, enable/disable diagnostics

http://www.ti.com/tool/ti-rtos

TI-RTOS


Connectivity
Wi-Fi®, Bluetooth® Smart, ZigBee®, Cellular (via PPP), TCP/IP
Optional Other Middleware
USB, File Systems
Platform
User Application Tasks
APIs
Power Manager
Real-Time Kernel
Drivers
IoT SoC
Sensors


SNTP
HTTP
TFTP
TELNET
DNS
DHCP
Standard BSD Sockets Interface
Route Manager
TCP
UDP
ICMP
IGMP
IP
I/F Manager
ARP
NAT
Ethernet I/F
Hardware Adaptation Layer
Ethernet Packet Driver
Serial Port Driver
Timer Driver
User LED Driver
Hardware

RiOT

- RIOT is an open-source microkernel operating system for the Internet of Things licensed as LGPL
- **It allows C and C++ application programming, and provides both full multi-threading and real-time capabilities (contrary to other operating systems with similar memory footprint such as TinyOS or Contiki).**
- RIOT runs on 8-bit (e.g., AVR Atmega), 16-bit (e.g., TI MSP430) and 32-bit hardware (e.g., as ARM Cortex).
- A native port also enables RIOT to run as a Linux or MacOS process, enabling the use of standard development and debugging tools such as GNU Compiler Collection (GCC), GNU Debugger, Valgrind, Wireshark etc. RIOT is partly POSIX-compliant.
- RIOT provides multiple network stacks, including IPv6, 6LoWPAN and standard protocols such as RPL, UDP, TCP and CoAP.

http://www.riot-os.org/

https://hal.inria.fr/file/index/docid/945122/filename/2013-riot_os.pdf

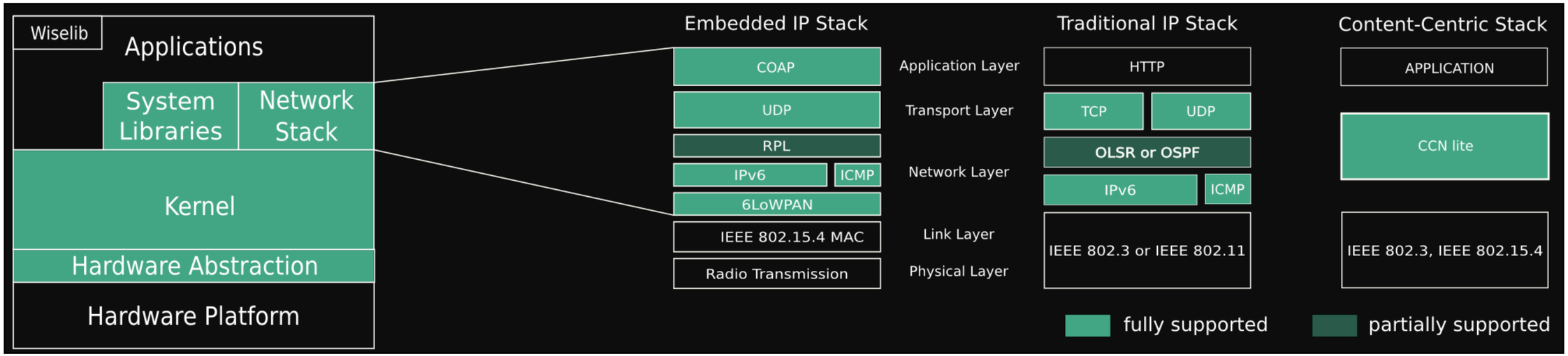
RiOT

OS	Min RAM	Min ROM	C Support	C++ Support	Multi-Threading	MCU w/o MMU	Modularity	Real-Time
Contiki	< 2kB	< 30kB	●	✗	●	✓	●	●
Tiny OS	< 1kB	< 4KB	✗	✗	●	✓	✗	✗
Linux	~ 1MB	~ 1MB	✓	✓	✓	✗	●	●
RIOT	**~ 1.5kB**	**~ 5kB**	✓	✓	✓	✓	✓	✓

✓ Full support
● Partial support
✗ No support

http://www.riot-os.org/

RiOT



http://www.riot-os.org/

Contiki OS

- Contiki is an open source operating system for the Internet of Things. It connects tiny low-cost, low-power microcontrollers to the Internet
- It is open source operating system and runs on tiny low-power microcontrollers making possible to develop applications that make efficient use of the hardware while providing standardized low-power wireless communication for a range of hardware platforms.
- It is used in numerous commercial and non-commercial systems, such as city sound monitoring, street lights, networked electrical power meters, industrial monitoring, radiation monitoring, construction site monitoring, alarm systems, remote house monitoring, and so on.
- COOJA: extensible Java-based network simulator for Contiki-based applications

http://www.contiki-os.org/

Contiki Protocol Stack

Protocol stacks in Contiki:

- uIP: world's smallest, fully compliant TCP/IP stack
  - Both IPv4 and IPv6, 6LowPAN, routing RPL, TCP/UPD support
  - Also higher layer protocols: HTTP, CoAP and many others
- Rime stack: protocol stack consisting of simple primitives
- MAC layers in Contiki:
  - Carrier Sense Multiple Access (CSMA)
  - NullMAC
- Radio Duty-Cycling (RDC) layers
  - ContikiMAC (default on Tmote Sky)
  - NullRDC (duty cycle off)
  - And others (less tested): LPP, X-MAC

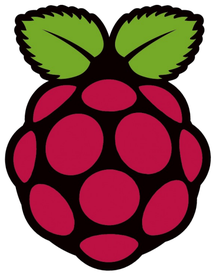
Eclipse Mosquitto

- Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers
- The current implementation of Mosquitto has an executable in the order of 120kB that consumes around 3MB RAM with 1000 clients connected
- It supports also the bridge mode allowing it to connect to other MQTT servers, including other Mosquitto instances. This allows networks of MQTT servers to be constructed, passing MQTT messages from any location in the network to any other, depending on the configuration of the bridges

https://mosquitto.org/

Eclipse Mosquitto - Platforms

- Available Installation possibilities are:
  - Build from sources
  - Windows
  - Mac Os X
  - Linux (Snap Support, Debian, Ubuntu)
  - Raspberry Pi (directly available through the main repository)
- Mosquitto is also available as Docker Container with its official image on docker registry: https://hub.docker.com/_/eclipse-mosquitto



CoAP Implementations

Smartphones

Some implementations are specifically targeting mobile devices such as smartphones and tablets. These tend to differ between platforms:

iOS, OSX

A simple iOS client implementation has been written by Wojtek Kordylewski in Objective-C.

View details »

CoAP client and server libraries are also available in Swift:

View details »

Android

Many Java components work out of the box on Android. For instance, Californium and nCoAP can be used on an Android device.

View details » View details »

Aneska is a simple CoAP browser based on txThings, installable from Google Play.

View details »

Constrained devices

Implementations for constrained devices are typically written in C.

Erbium

Contiki is a widely used operating system for constrained nodes, being employed for research and product development. Erbium is a full-fledged REST Engine and CoAP Implementation for Contiki.

View details »

libcoap

A C implementation of CoAP that can be used both on constrained devices (running operating systems such as Contiki or LWIP) and on a larger POSIX system. Moreover, the library has been ported to TinyOS and RIOT.

View details »

tinydtls

To enable CoAP's security on a tiny device, a tiny implementation of DTLS for Class 1 devices:

View details »

SMCP

A small-but-capable C-based CoAP stack suitable for embedded environments. It supports observing, asynchronous responses to requests, and more. Features that aren't used can be removed to lower the footprint.

View details »

microcoap

A C implementation that can be compiled for both Arduino and POSIX environments:

View details »

Server-side

Java

One significant Java-based implementation of CoAP is **Californium**.

View details »

nCoAP is a Java implementation of the CoAP protocol using the Netty NIO client server framework:

View details »

Leshan is an OMA Lightweight M2M (LWM2M) server-side implementation, on top of Californium.

View details »

C

Several of the constrained-device implementations, such as libcoap, can also be used on the server side.

C#

CoAP.NET is an implementation in C# providing CoAP-based services to .NET applications.

View details »

CoAPSharp is a lightweight library for building CoAP enabled sensors and machines that also works with Microsoft .NET Micro Framework, with a tutorial for its API:

View details »

Waher.Networking.CoAP is a library for server-side C# that also comes as a Universal Windows Platform (UWP) version (**Waher.Networking.CoAP.UWP**), with tutorials and nuget links:

View details » View details »

Erlang

gen_coap is a pure Erlang implementation of a generic CoAP client and server:

View details »

Go

go-ocf/go-coap CoAP client and server supporting UDP, TCP/TLS, resource observation, Block-wise transfer, multicast and request multiplexer. Written purely in Golang.

View details »

go-dustin/go-coap Basic CoAP client and server:

View details »

JavaScript (node.js)

node-coap is a client and server library for CoAP modelled after the *http* module. Install with `npm install coap`.

View details »

CoAP-CLI is a command line interface for CoAP, built on node.js and node-coap. Install with `npm install coap-cli -g`.

View details »

Python

txThings is a CoAP library based on Python's Twisted framework:

View details »

aiocoap implements CoAP natively on Python 3.4's asyncio mechanisms, and provides command line tools for resource fetching and proxying:

View details »

CoAPthon is a python library for the CoAP protocol, with a branch available that uses the Twisted framework.

View details »

https://coap.technology/
https://coap.technology/impls.html

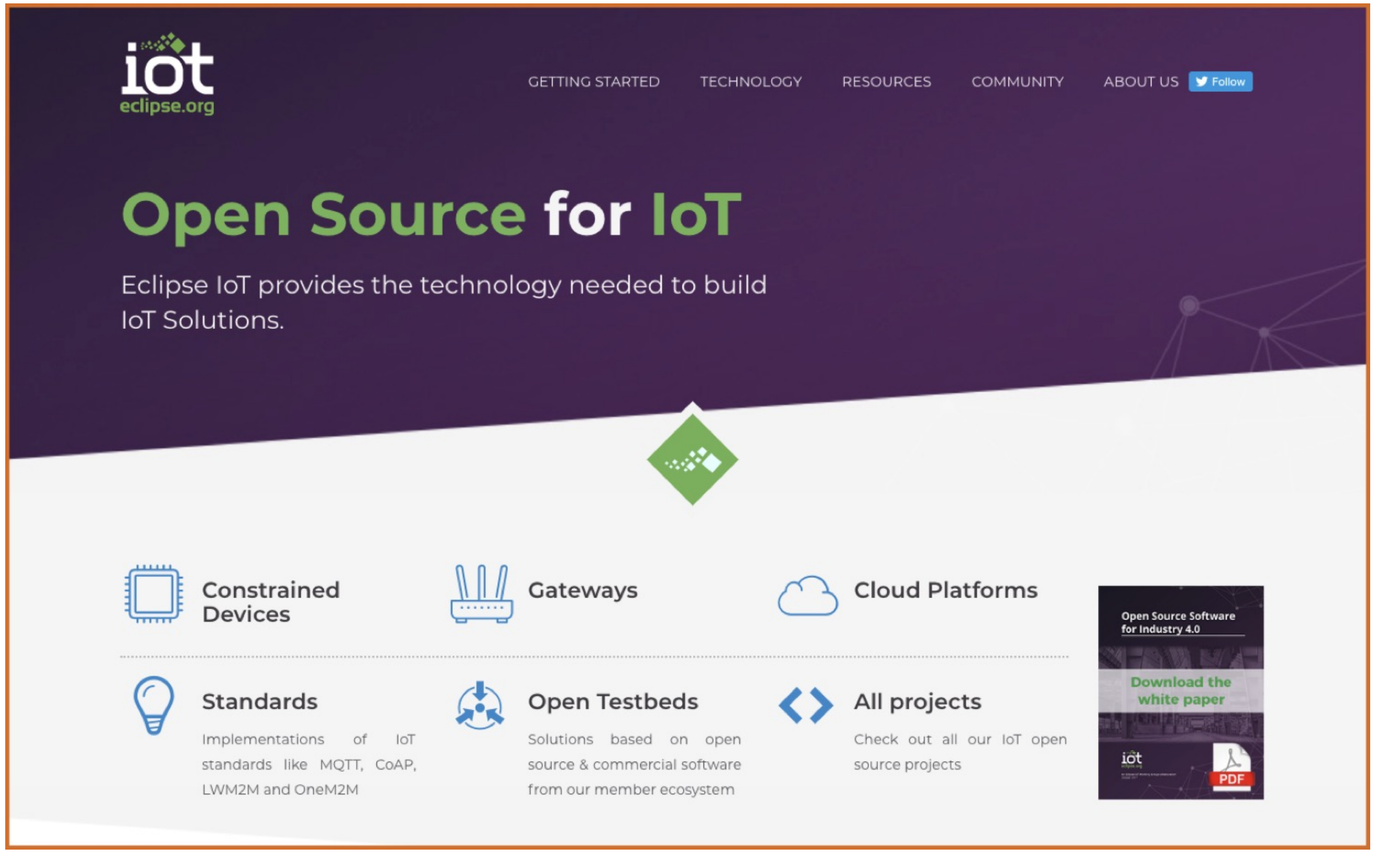
Eclipse Californium - CoAP

- Californium is a powerful CoAP framework
- It provides a convenient API for RESTful Web services that support all of CoAP's features
- Californium has been running code for the IETF and is passing all ETSI Plugtest test specifications
- Californium has an extremely scalable architecture and outperforms high-performance HTTP servers. CoAP's low overhead allows to handle millions of IoT devices with a single service instance.
- It provides an additional module called Scandium (Sc) adding security for Californium. It implements DTLS 1.2 to secure your application through ECC with pre-shared keys, certificates, or raw public keys.

Eclipse IoT & Working Group

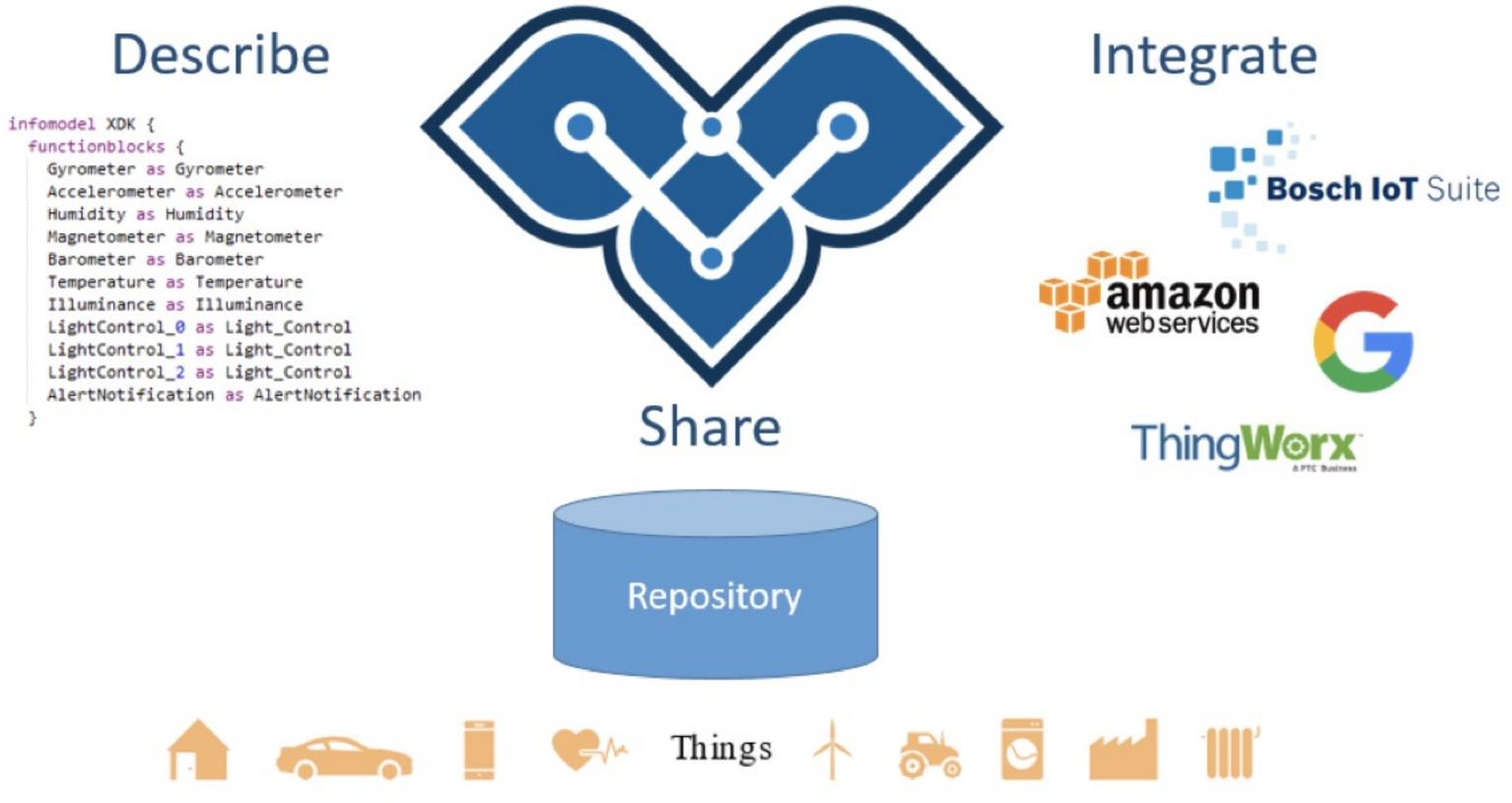
Eclipse IoT - https://iot.eclipse.org/

Eclipse IoT Working Group - https://www.eclipse.org/org/workinggroups/iotwg_charter.php



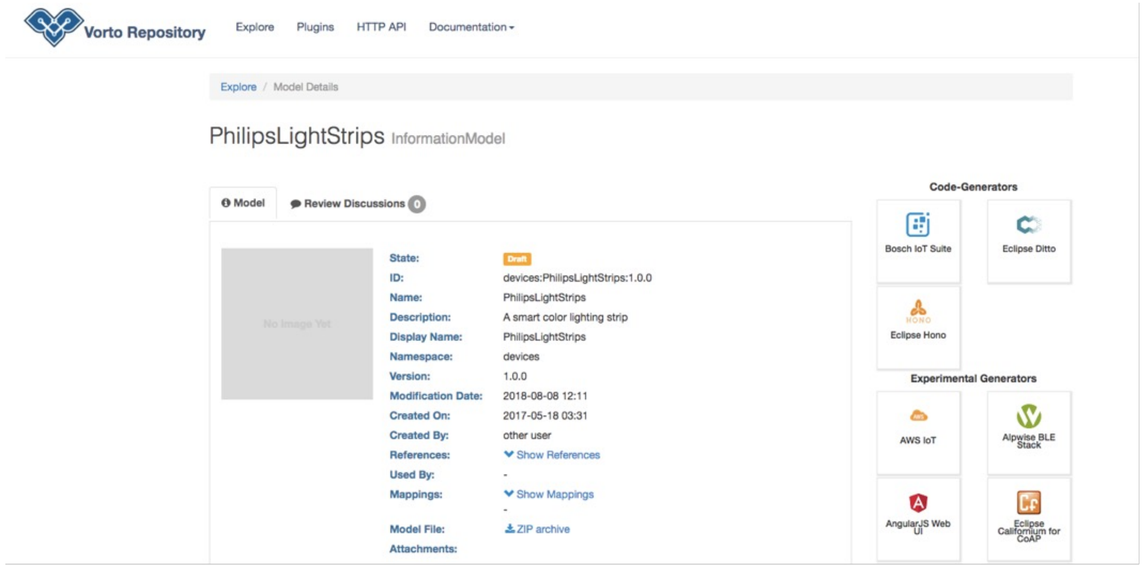
Eclipse Vorto

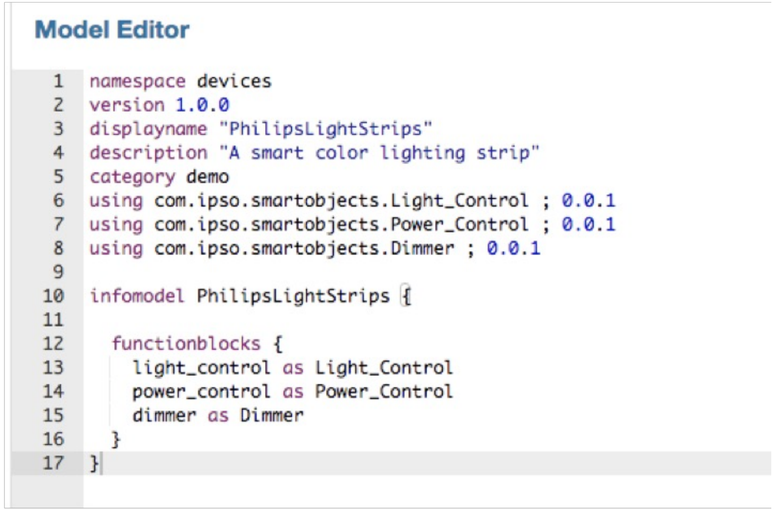
- Vorto provides a unified way to describe an IoT Device using Vorto DSL (Domain Specific Language).
- Vorto allows to:
  - Abstract Things Description
  - Manage models
  - Code Generation

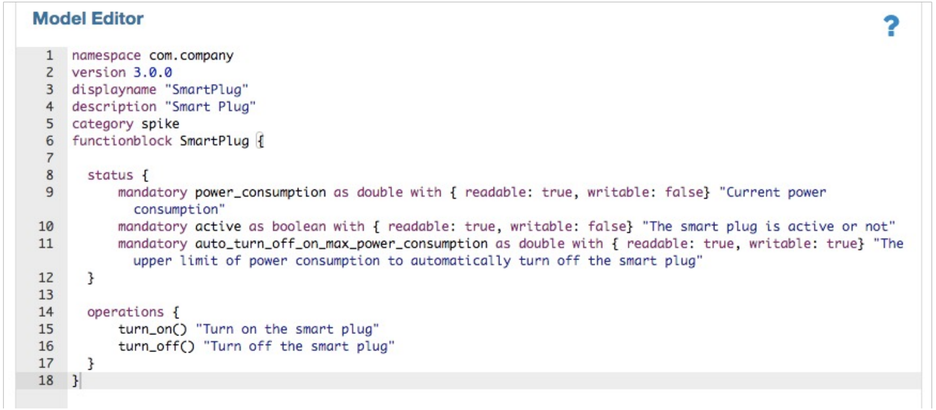


https://www.eclipse.org/vorto/

Eclipse Vorto

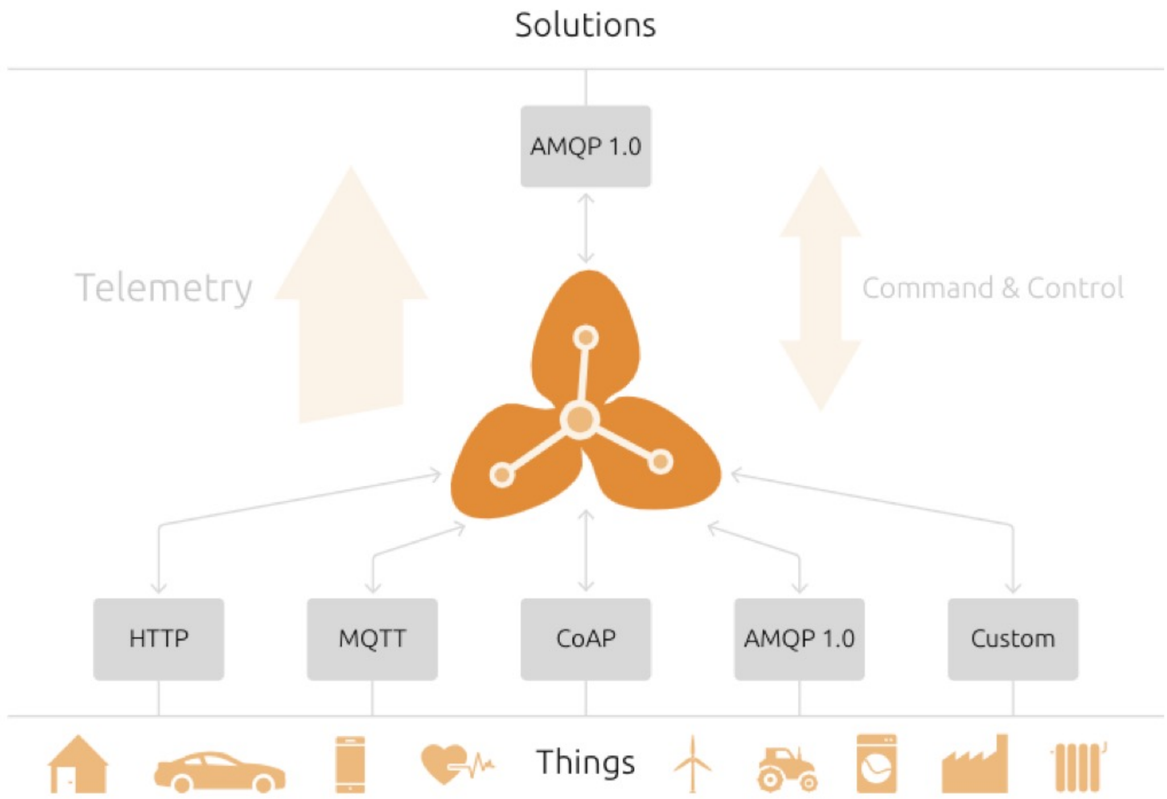






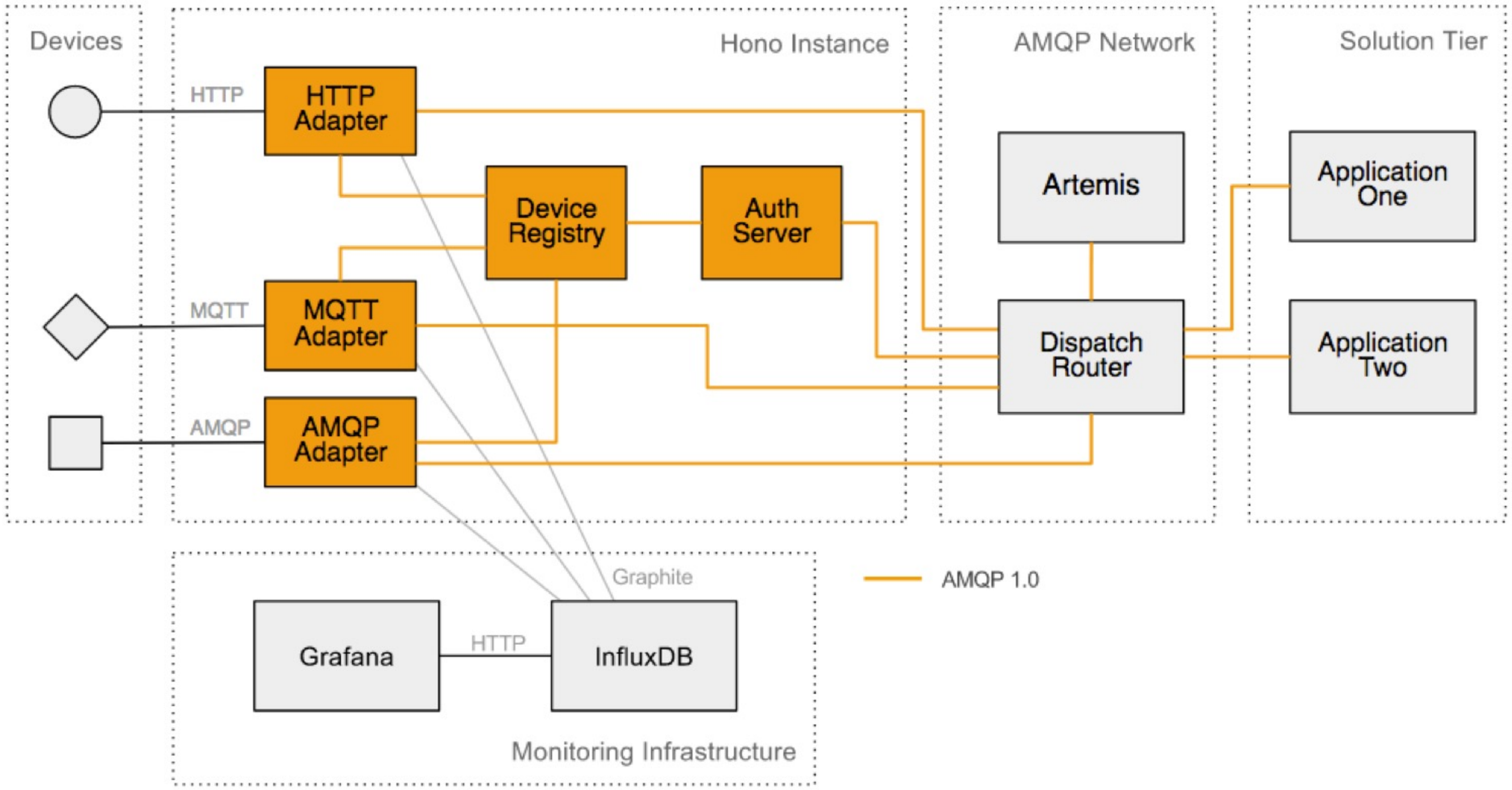
Connect, Command & Control - Eclipse Hono

- Eclipse Hono™ provides remote service interfaces for connecting large numbers of IoT devices to a back end and interacting with them in a uniform way regardless of the device communication protocol
- Hono specifically supports scalable and secure ingestion of large volumes of sensor data by means of its Telemetry and Event APIs.
- Hono's Command & Control API allows for sending commands (request messages) to devices and receive a reply to such a command from a device in an asynchronous way

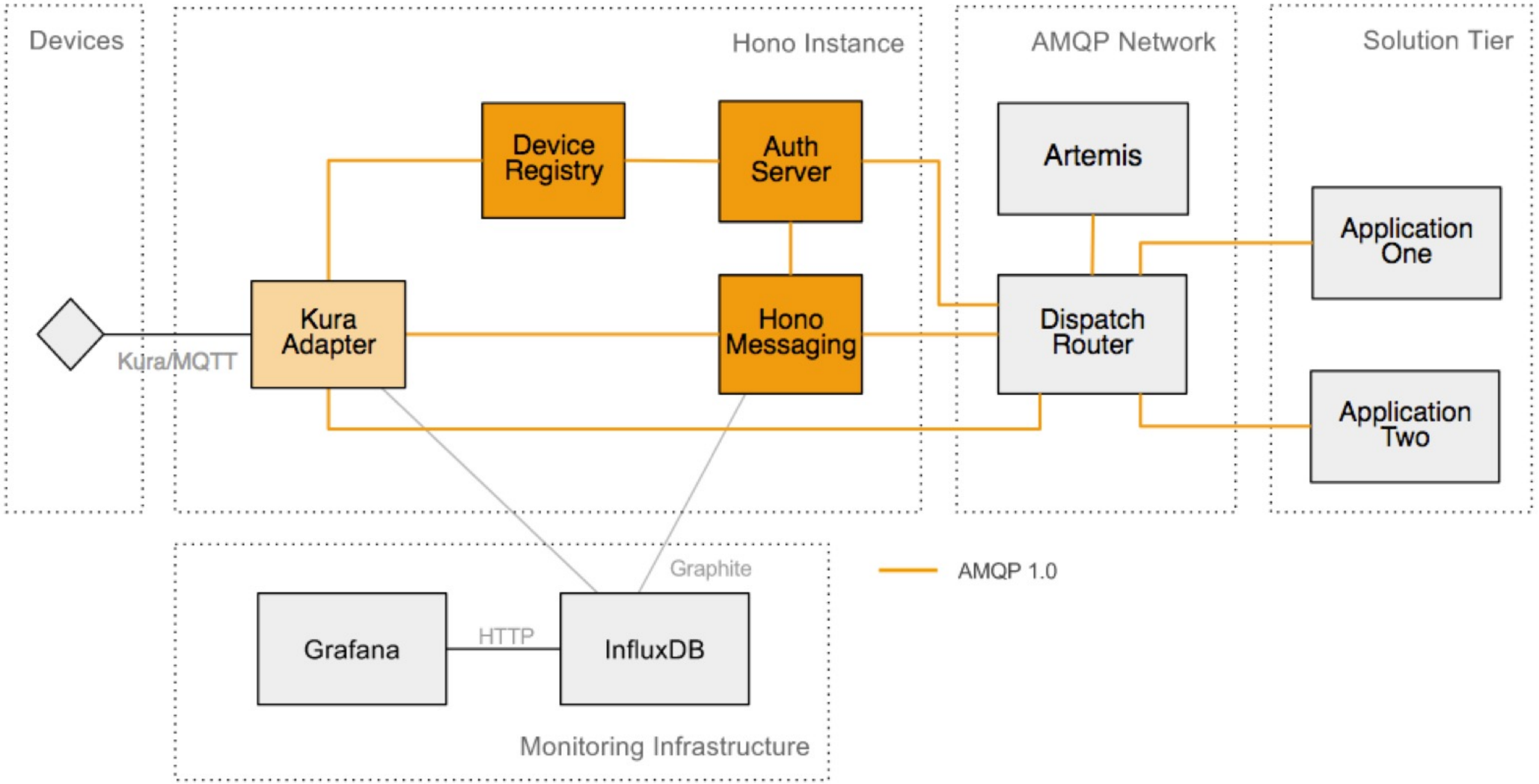


https://www.eclipse.org/hono/

Connect, Command & Control - Eclipse Hono

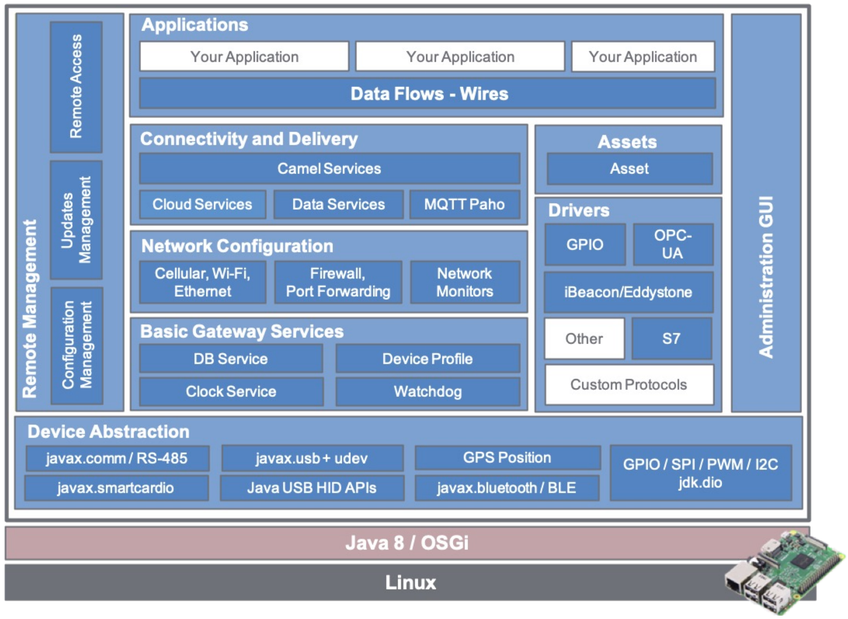


Connect, Command & Control - Eclipse Hono

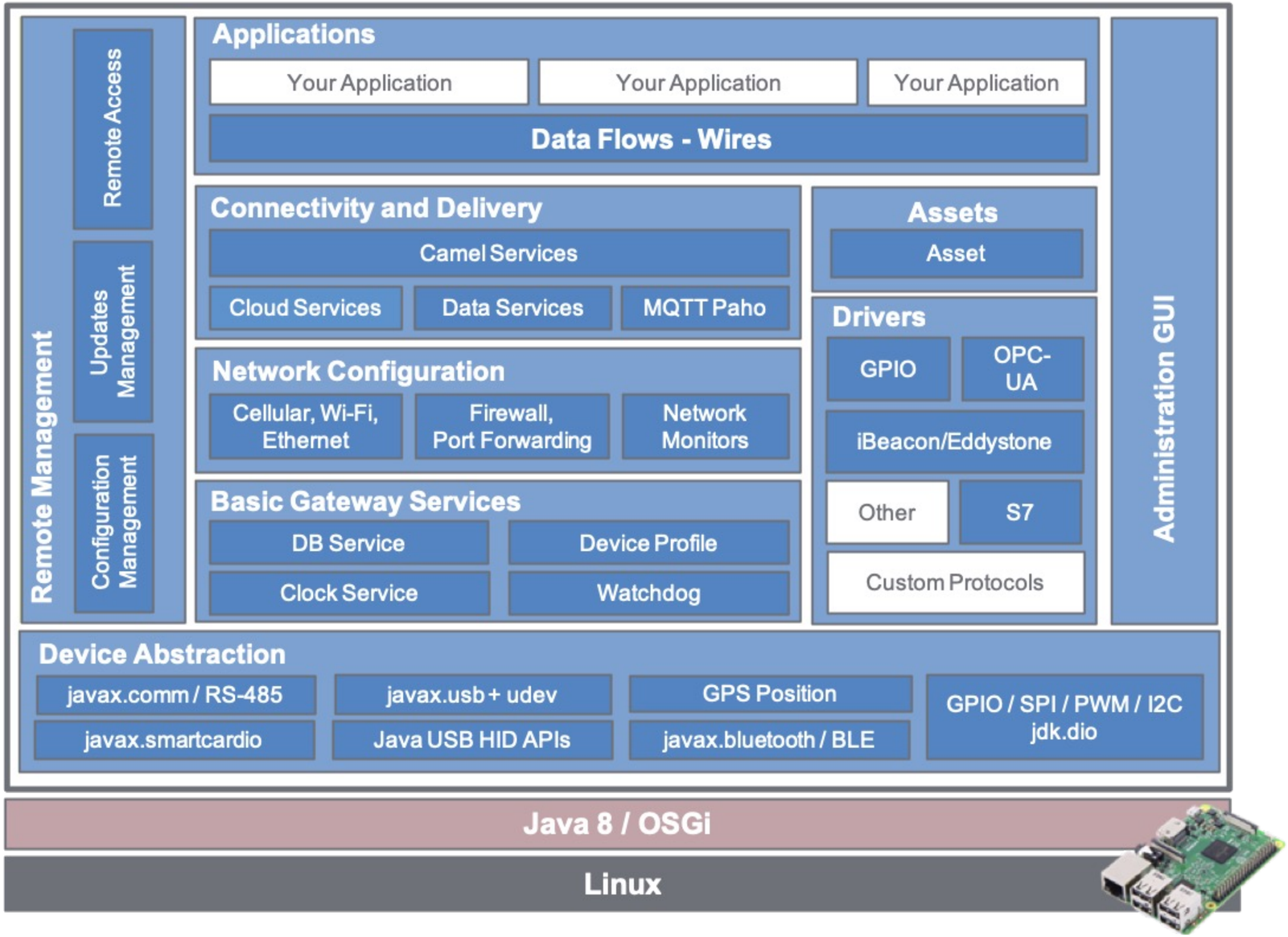


Eclipse Kura

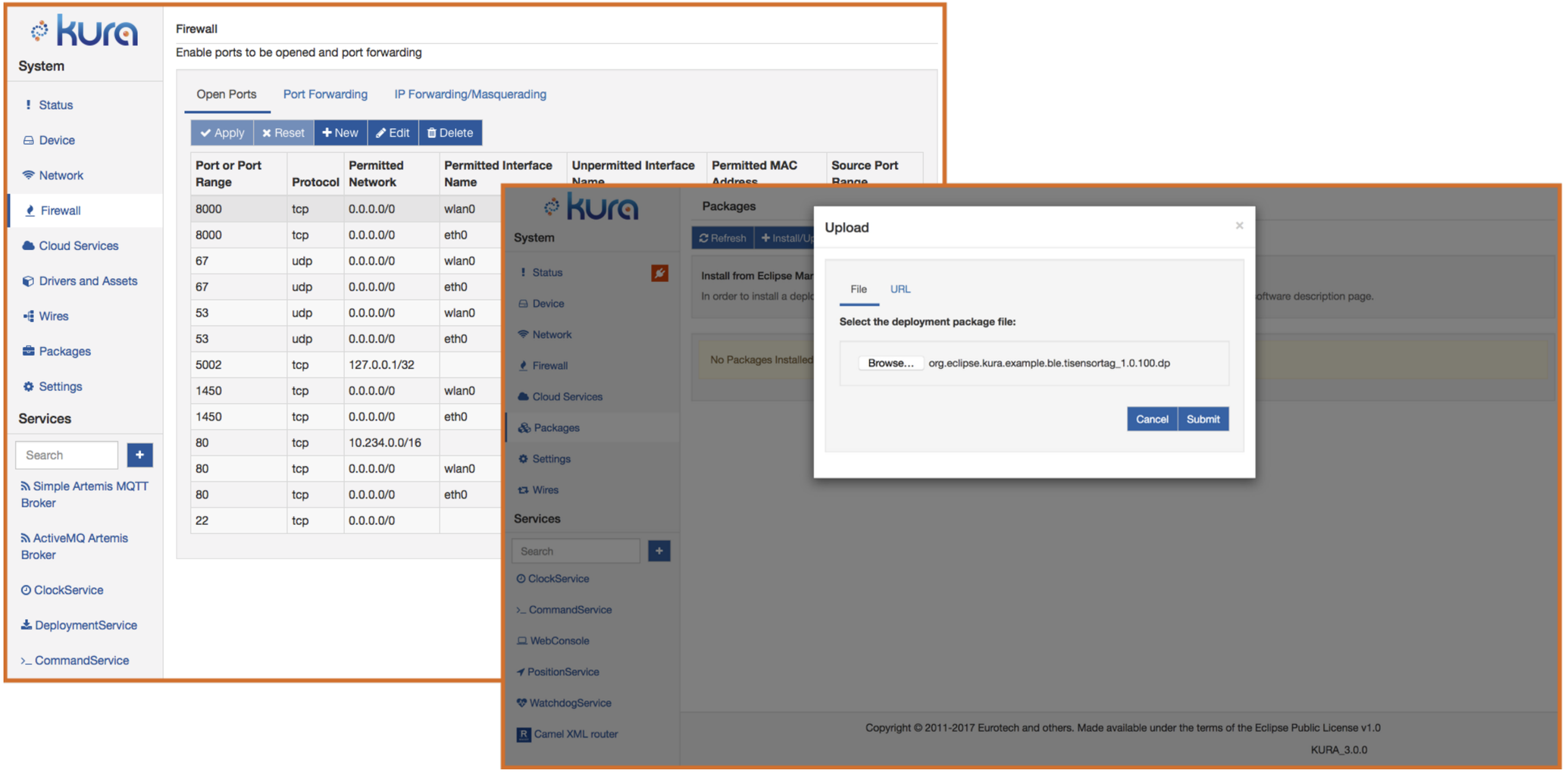
- Eclipse Kura is an Eclipse IoT project that provides a platform for building IoT gateways. It is a smart application container that enables remote management of such gateways and provides a wide range of APIs for allowing you to write and deploy your own IoT application
- Kura runs on top of the Java Virtual Machine (JVM) and leverages OSGi (Open Service Gateway initiative), a dynamic component system for Java, to simplify the process of writing reusable software building blocks
- Kura APIs offer easy access to the underlying hardware including serial ports, GPS, watchdog, USB, GPIOs, I2C, etc.
- It also offer OSGI bundle to simplify the management of network configurations, the communication with IoT servers, and the remote management of the gateway.



Eclipse Kura

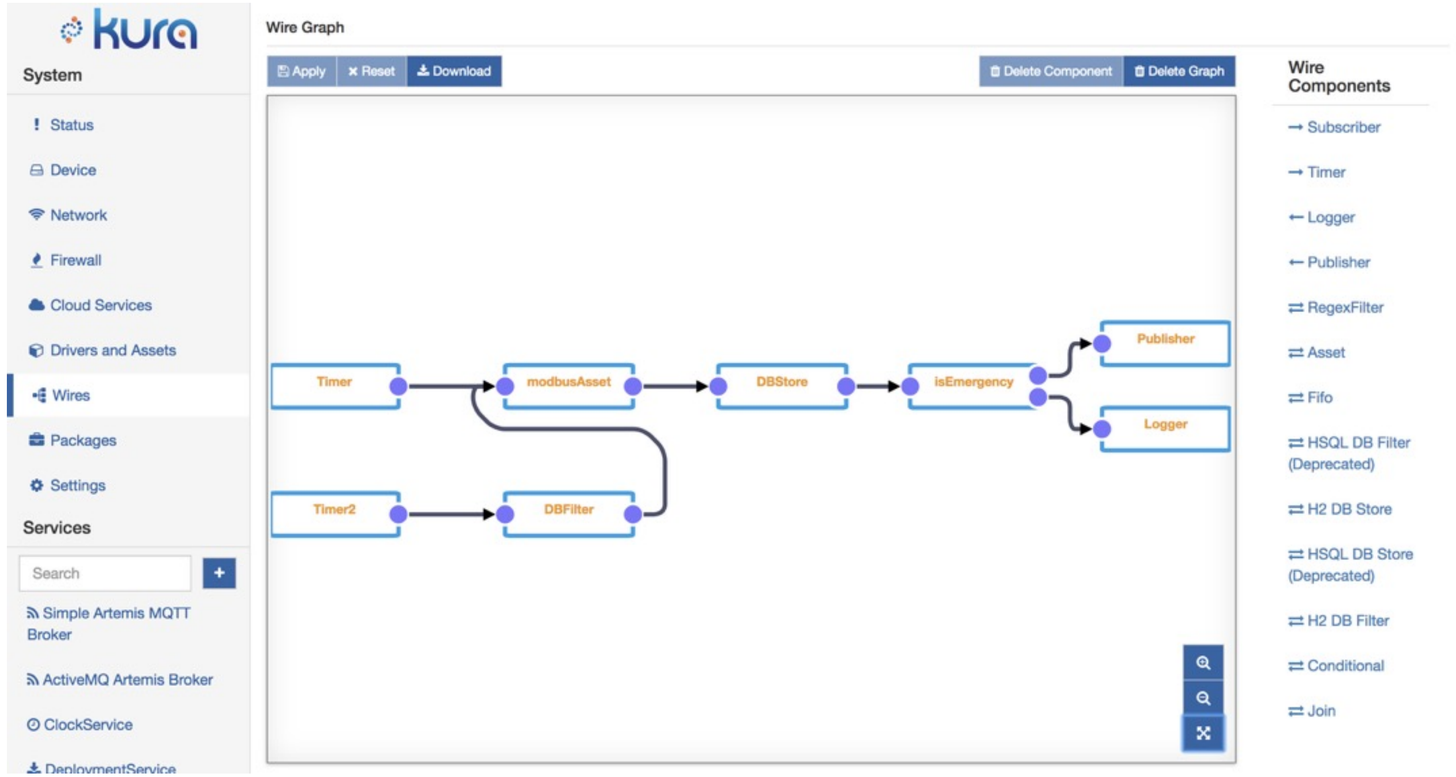


Eclipse Kura



Eclipse Kura - Wires

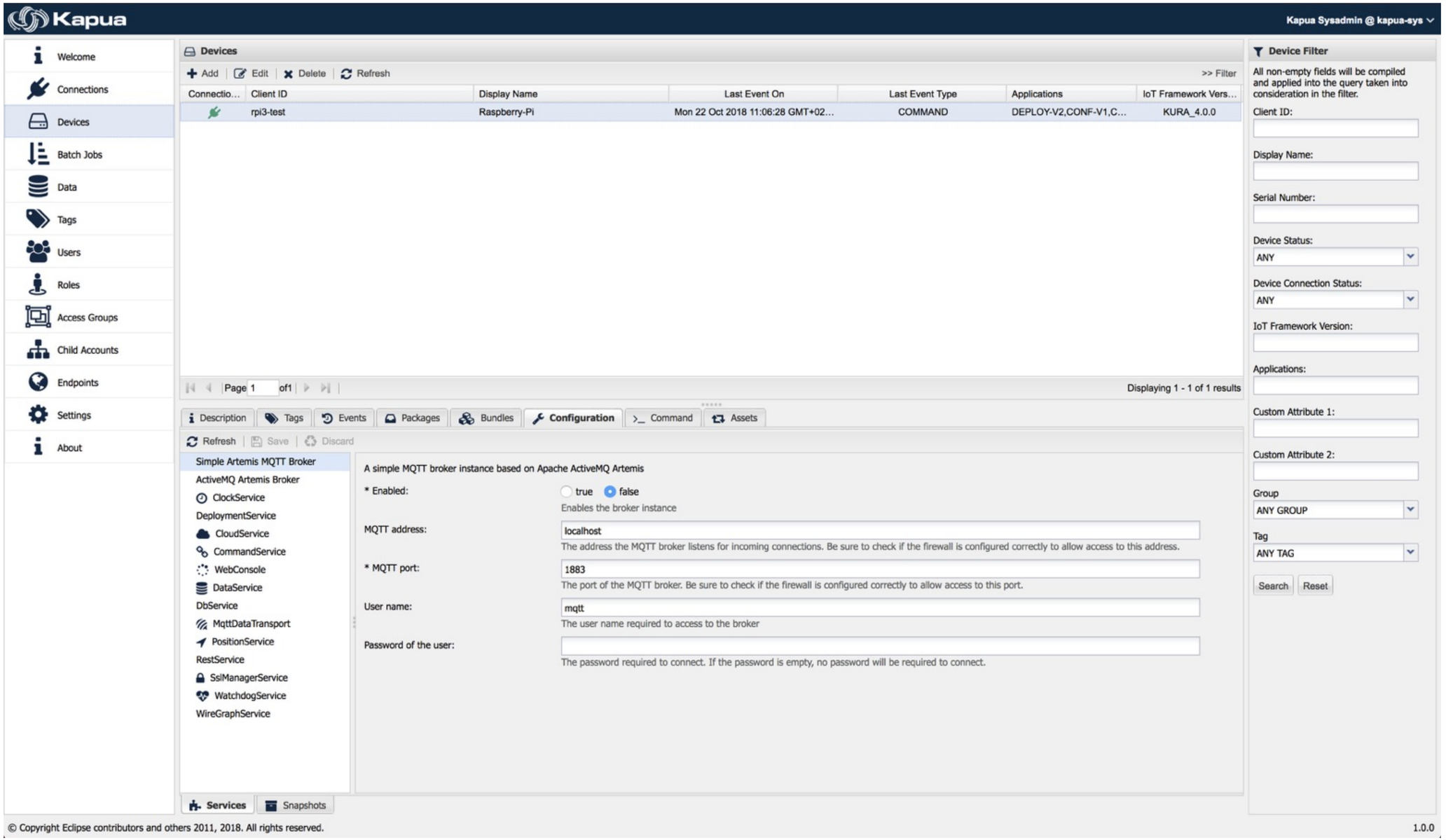
- Eclipse Kura Wires provides in Kura environment an implementation of the Dataflow Programming model. The Application logic is expressed as a directed graph
- The nodes in the graph have inputs and/or outputs, abstract the underlying logics and are highly reusable and portable. In this way, the developer can easily prototype its solution without sacrificing flexibility and working at a high level of abstraction



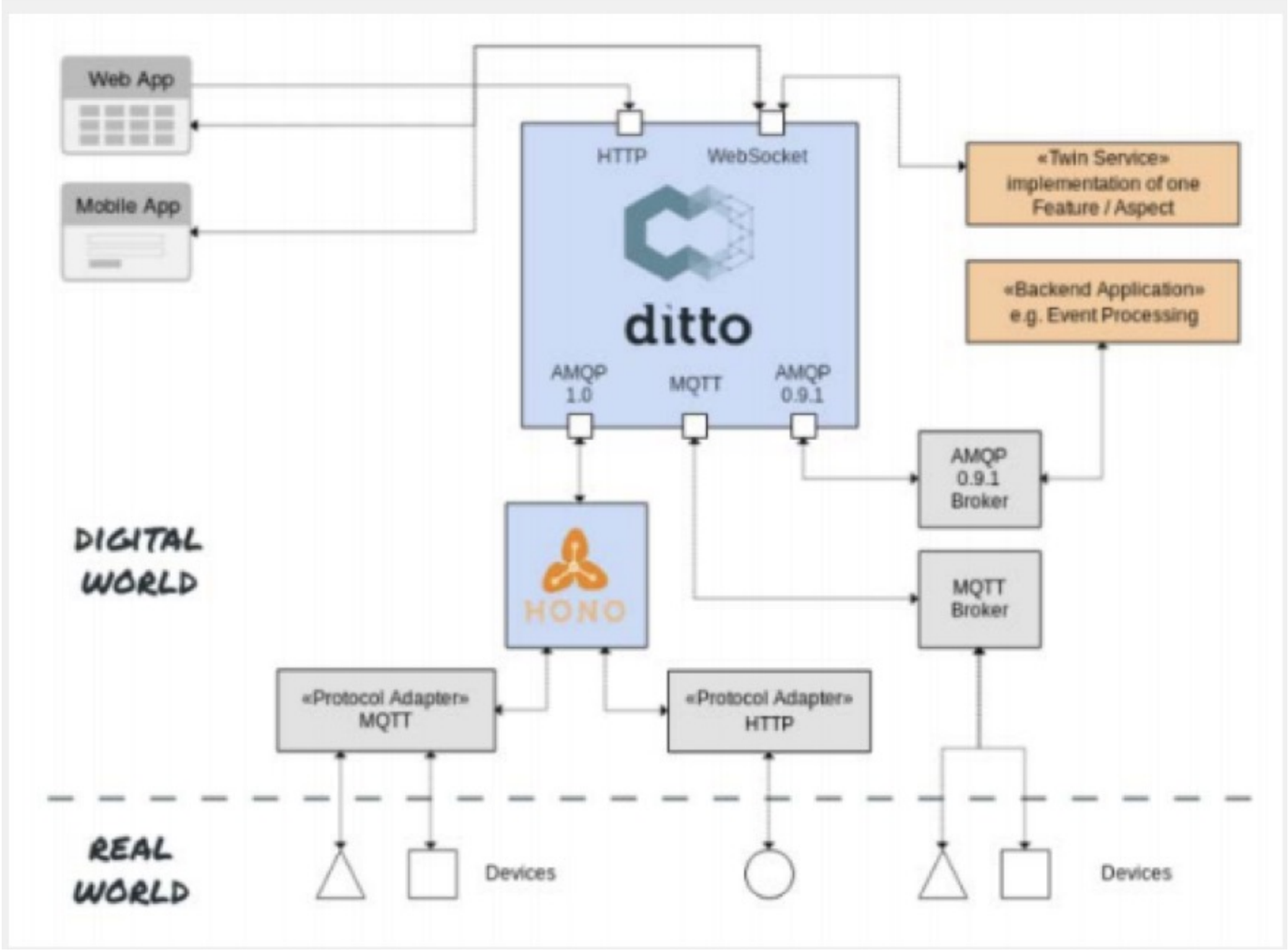
Eclipse - Kapua

Modular IoT cloud platform to manage and integrate devices and their data. A solid integrated foundation of IoT services for any IoT application.

Eclipse - Kapua



Eclipse - Ditto - Digital Twin



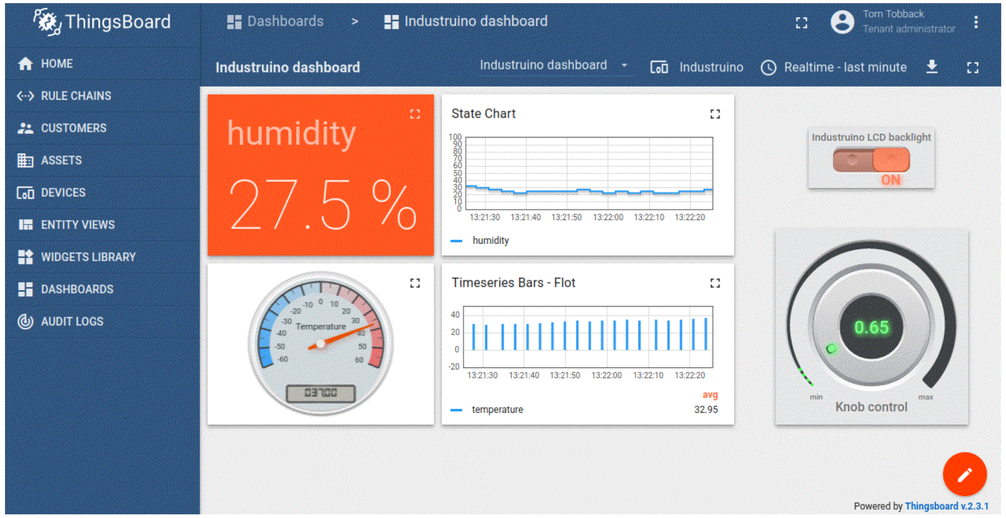
- Eclipse Ditto is a technology in the IoT implementing the software pattern called “Digital Twins”
- A Digital Twin is a virtual, **cloud based**, representation of his real world counterpart

Eclipse - Ditto - Digital Twin

- It **does not provide** software running on IoT gateways and it does not define or implement an IoT protocol in order to communicate with devices
- Its focus lies on **backend scenarios** by providing web APIs in order to simplify working with already connected devices and “Things” from customer apps or other back end software.
- It also **does not specify** which data or which structure a “Thing” in the IoT has to provide.
- Its **goal is to free IoT solutions from the need of implementing and operating a custom back end**
- Instead by using Eclipse Ditto they can focus on business requirements, on connecting devices to the cloud/back end and on implementing business applications.

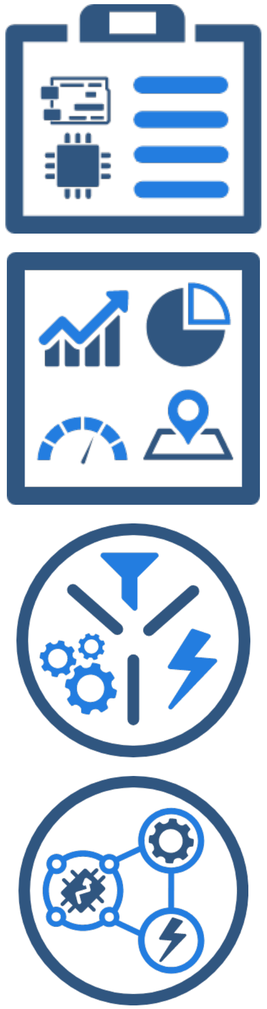
Thingsboard

- ThingsBoard is an open-source IoT platform for data collection, processing, visualization, and device management.
- It enables device connectivity via industry standard IoT protocols - MQTT, CoAP and HTTP and supports both cloud and on-premises deployments.
- ThingsBoard combines scalability, fault-tolerance and performance

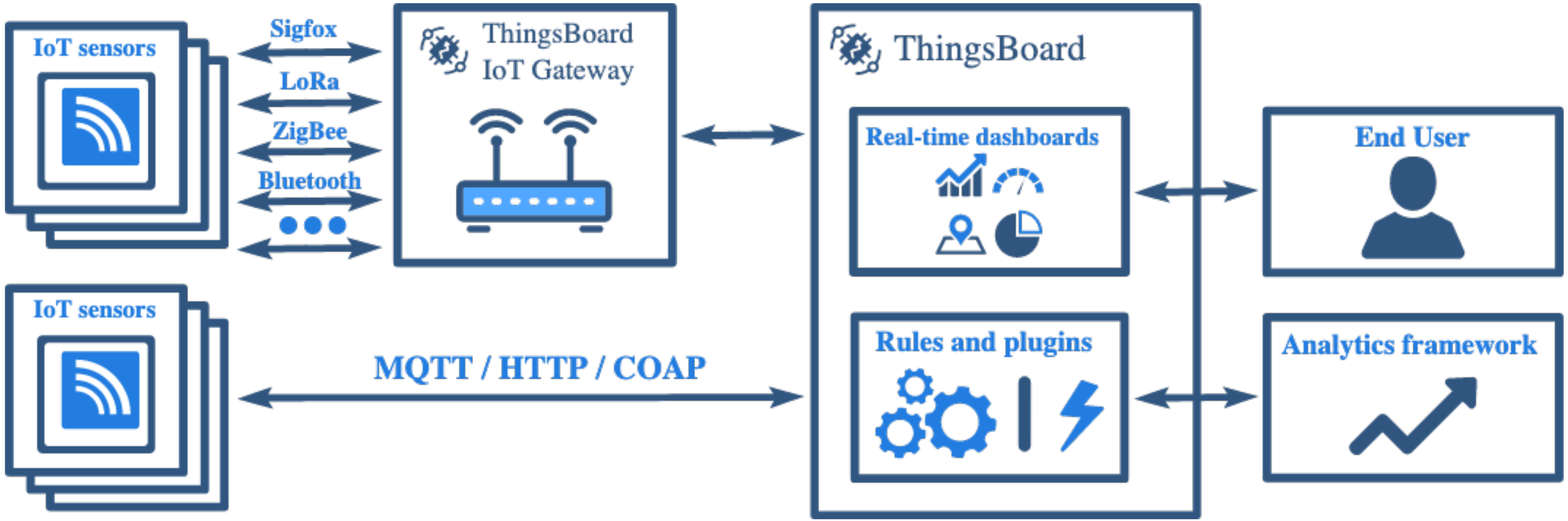


Thingsboard - Features

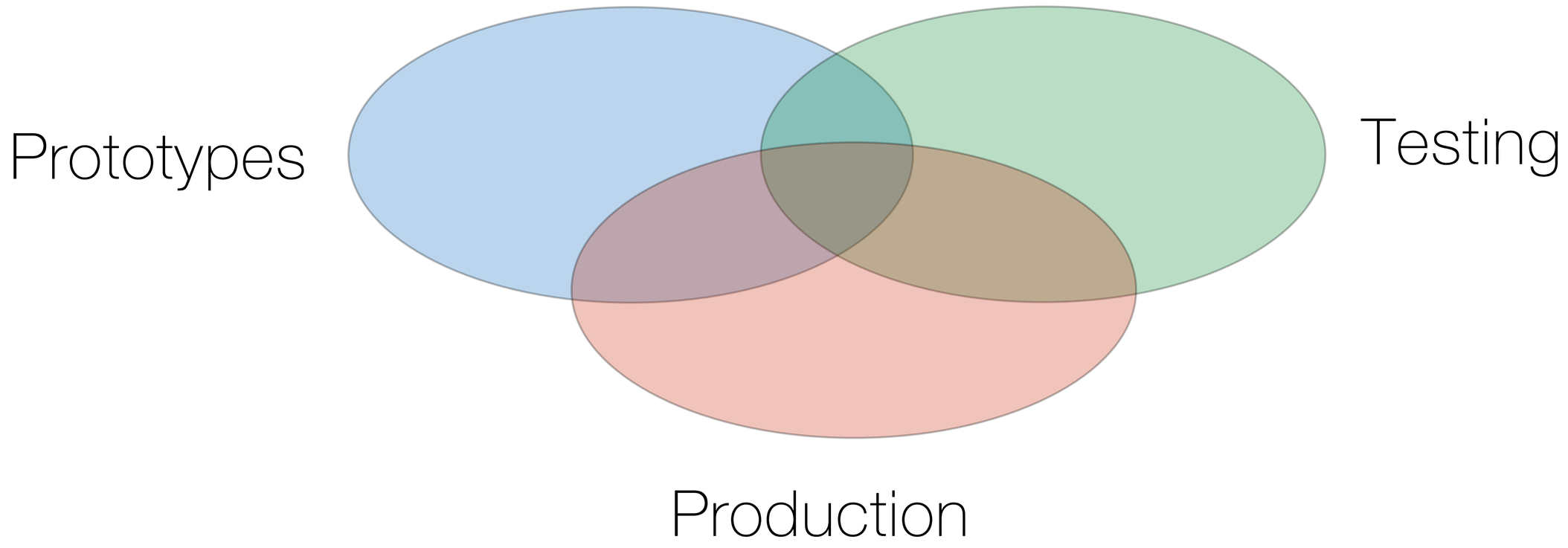
- **Provision and manage devices and assets**
  - Provision, monitor and control your IoT entities in secure way using rich server-side APIs. Define relations between your devices, assets, customers or any other entities.
- **Collect and visualize data**
  - Collect and store telemetry data in scalable and fault-tolerant way. Visualize your data with built-in or custom widgets and flexible dashboards. Share dashboards with your users.
- **Process and React**
  - Define data processing rule chains. Transform and normalize your device data. Raise alarms on incoming telemetry events, attribute updates, device inactivity and user actions
- **Microservices**
  - Construct your ThingsBoard cluster and get maximum scalability and fault-tolerance with new microservices architecture. ThingsBoard also supports both cloud and on-premises deployments.



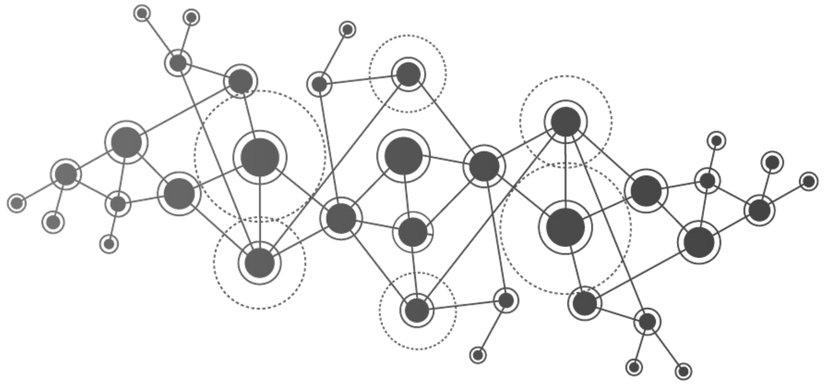
Thingsboard - Architecture Overview



Prototyping, Testing or Production ?



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA



Intelligent Internet of Things

IoT Hardware & Software

Prof. Marco Picone

A.A 2023/2024