



Mobile Systems M

Alma Mater Studiorum – University of Bologna
CdS Laurea Magistrale (MSc) in
Computer Science Engineering

Mobile Systems M course (8 ECTS)
II Term – Academic Year 2021/2022

04 – Internet of Things (IoT): Definitions and Application Scenarios

Paolo Bellavista
paolo.bellavista@unibo.it

<http://lia.disi.unibo.it/Courses/sm2122-info/>

Internet of Things (IoT)



- Old term, probably its first appearance in 1999 by Kevin Ashton
- IoT term applied to RFID, supply chain, and the Internet
- Then, in 2009: “Conventional diagrams of the Internet include servers and routers and so on, but they leave out the most numerous and important routers of all: people. The problem is, **people have limited time, attention and accuracy** [...] If we had **computers that knew everything** there was to know **about things** [...] we would be able to [...] greatly **reduce waste, loss and cost**. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best

Internet of Things - Scenarios

- Since 1999 the concept evolved far beyond RFID
- **Everything can now be connected to the Internet**
- The term IoT now refers to different scenarios:
 - Wireless Sensor Networks (WSN)
 - Near Field Communications (NFC)
 - Biotechnology and Body Area Networks (BAN)
 - Machine-to-Machine communications (M2M)
 - Personal Area Networks (PAN)
 - ...

IoT - How Big?

- Possibly every single device and object will be connected to the Internet
- About 50-100 billions devices in 2020 (data from SAP/Intel and Ericsson)
- IBM Smarter Planet vision:
 - instrumented
 - interconnected
 - intelligent
- Several real-world examples: industrial control systems, health monitoring, smart metering, home automation, ...



Internet of Things - Use Cases

Smart Wearables



Smart Home



Smart City



Smart Agriculture



Connected Car



Health Care



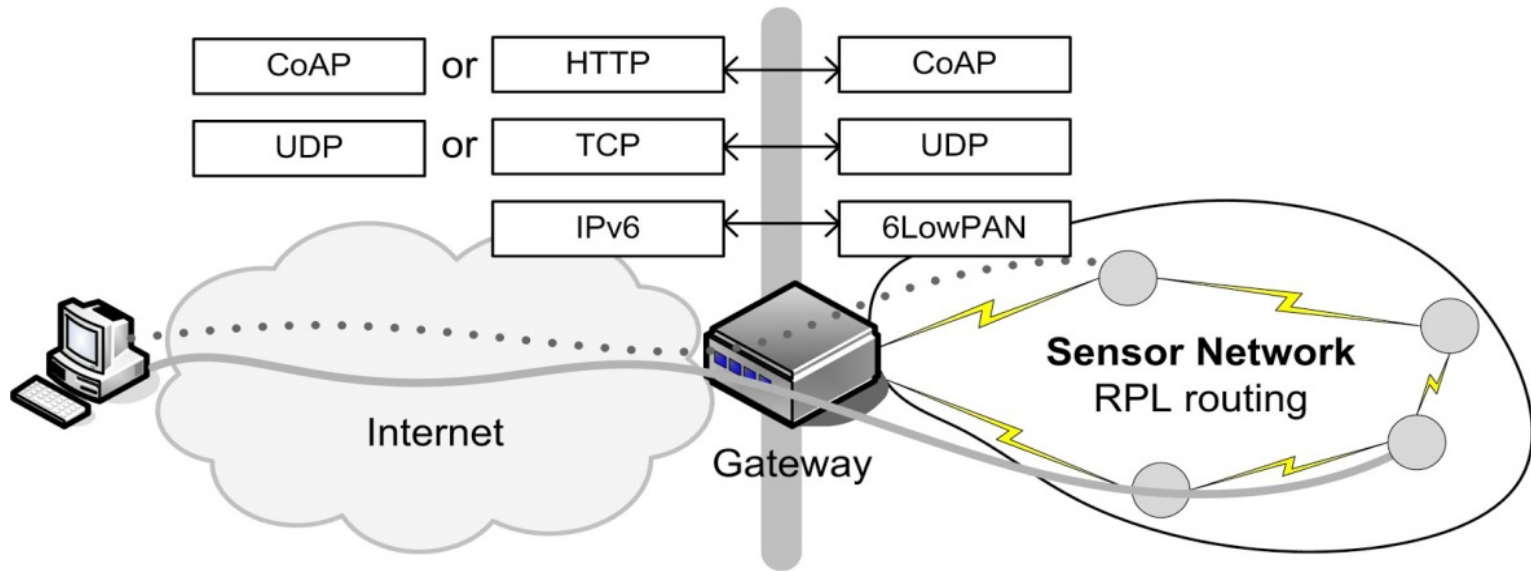
Industry Automation



Smart Energy



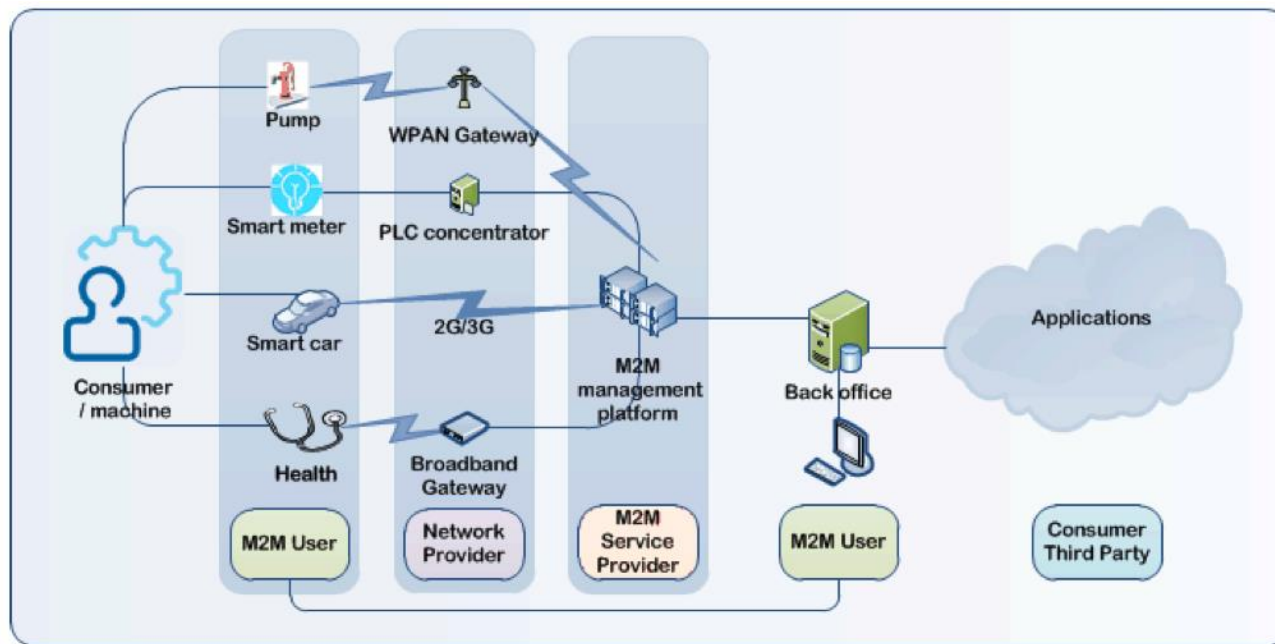
IoT - the General Idea



- The general idea behind the (commonly accepted) vision of IoT consists in the **extension of Internet protocols to Wireless Sensors Networks (WSNs)**, composed of sensors as well as actuators

IoT for Manufacturing - Machine to Machine

- The term “Machine to Machine” (**M2M**) describes devices **connected to each other**, by using a variety of fixed/wireless networks and through the Internet to the wider world. They are **active** communication devices



- Note the **wide variety of communication** infrastructures and services and **massive number of nodes**

IoT - Main Tasks

- **Gather information from things** and send commands to things
 - monitoring: state information
 - control: command enforcement
- **Send information back and forth** remote locations (private/public cloud)
- **Store and aggregate** information
- **Analyze** information to improve system knowledge
- **Take decisions**, in a human-assisted or autonomous manner

Outline (1)

1. Introduction to IoT
 - definition, enabling technologies and concepts to better understand the general framework of IoT solutions
 - layering architecture, cloud computing vs. fog computing
2. Most relevant components of IoT solutions
 - devices, wireless communication protocols, data exchange protocols

Outline (2)

3. Industrial IoT

- Industrial data exchange frameworks
- Industrial IoT platforms
- (The Industry 4.0 evolution trend)

4. Hands-on lab on IoT platforms (in a separate slide deck)

- Azure IoT Hub
- EdgeXFoundry

Outline

1. Introduction to IoT

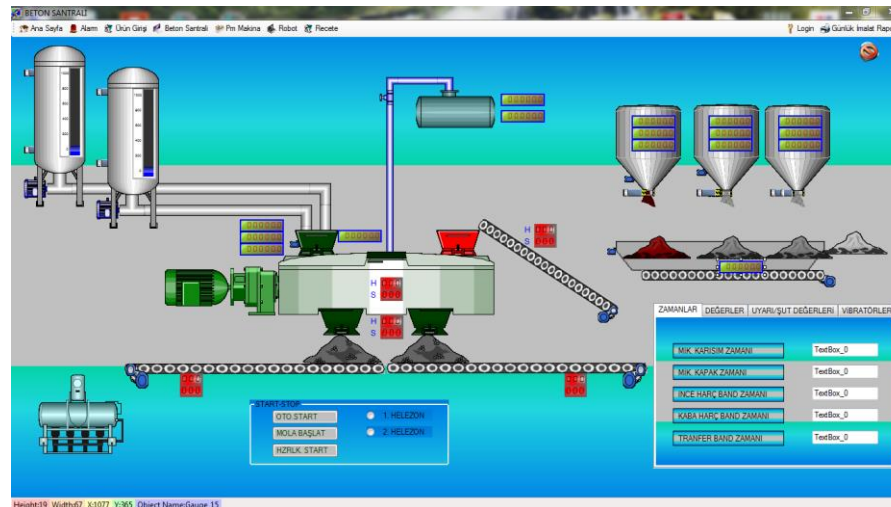
- **definition, enabling technologies and concepts to better understand the general framework of IoT solutions**
- layering architecture, cloud computing vs. fog computing

2. Most relevant components of IoT solutions

- devices, wireless communication protocols, data exchange protocols, IoT platforms, and data analysis

SCADA - Supervisory Control And Data Acquisition

- Before IoT, **remote monitoring and control already available for years**
- Central control system with sensors/actuators, controllers, communication equipment, and software
- Periodic reading of values and status of sensors to require minimal intervention of human personnel
- Typical deployments: gas and oil distribution, electrical power, water distribution, bus traffic system, airport



SCADA

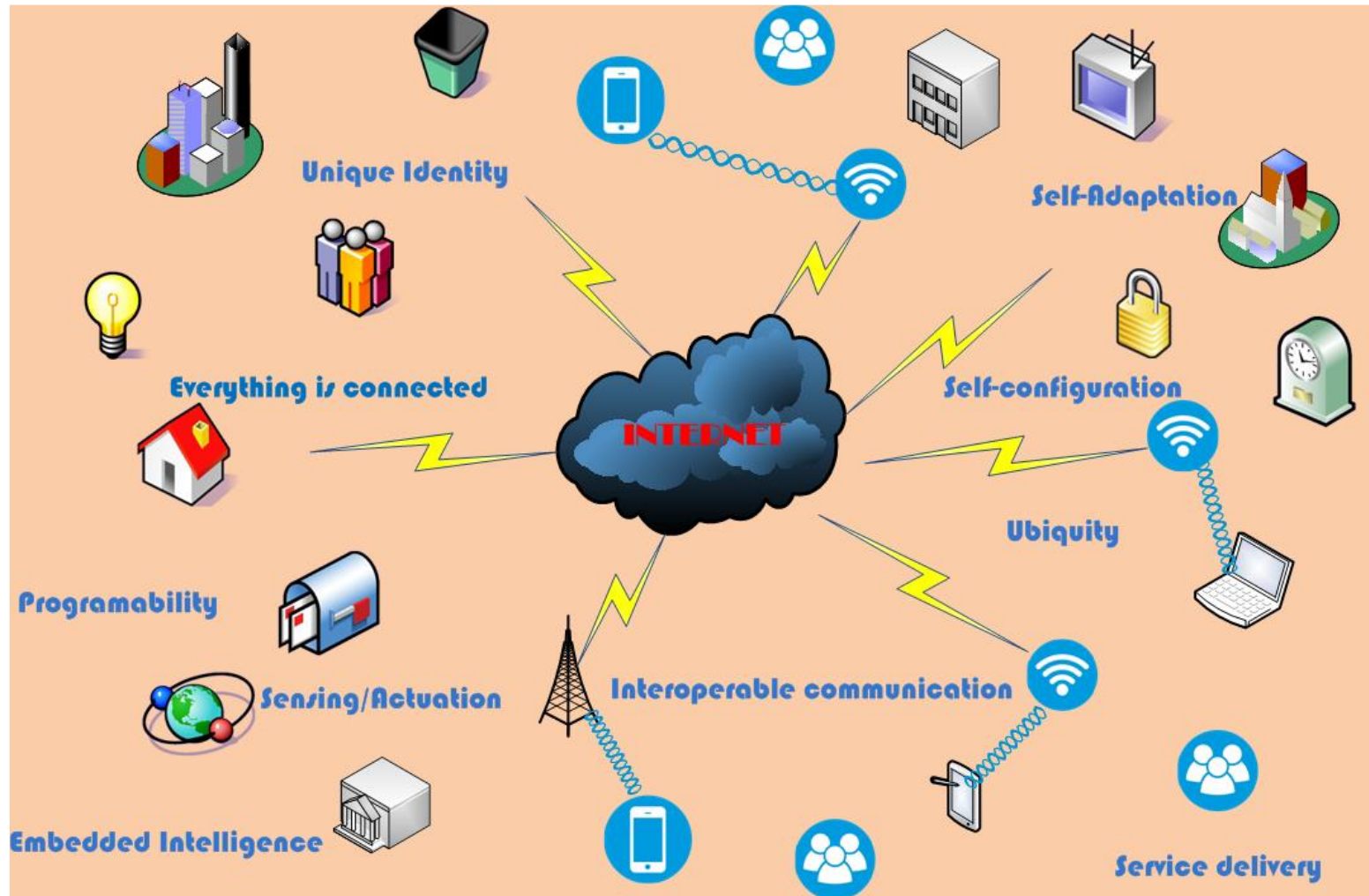
- Pros
 - widely tested solutions
 - real-time and reliable
 - tailored to operation technicians
- Cons
 - **expensive**, but worth for mission critical tasks
 - **extensible, but within the same scope**, e.g., an additional sensor in the same machine or an additional machine in the same plant
 - **lack of standards**, difficult to maintain/evolve
 - usually **no interoperability** (horizontal/vertical)



IoT Definition – Small Environment Scenario

- An IoT
 - is a network that
 - connects uniquely identifiable “Things”
 - to the Internet
- The Things
 - have sensing/actuation and
 - potential programmability capabilities
- Through the exploitation of unique identification and sensing
 - information about the Thing can be collected
 - and the state of the Thing can be changed
 - from anywhere, anytime, by anything

IoT Definition – Large Environment Scenario (1)



IoT Definition – Large Environment Scenario (2)

- Internet of Things **envisions** a self-configuring, adaptive, **complex network that interconnects Things** to the Internet through the use of standard communication protocols
- The interconnected things **have** physical or virtual representation in the digital world, **sensing/actuation capability**, a programmability feature and are uniquely identifiable
- The **representation contains information** including the thing's identity, status, location or any other business, social or privately relevant information
- The things **offer services**, with or without human intervention, through the exploitation of unique identification, data capture and communication, and actuation capability
- The service is exploited through the use of **intelligent interfaces** and is made available **anywhere, anytime, and for anything**, taking security into consideration

IoT Enabling Technologies

- Reduced **hardware** cost and size
 - from special-purpose to Commercial Off-The-Shelf (COTS)
- Pervasive and cheap **wireless communication**
 - from cables to large-bandwidth and/or wide-coverage wireless communication
- Consolidated and emerging **Web-based communication**
 - from close protocols to open standards, also applied in constrained devices
- **Standards**, to achieve interoperability
 - e.g., communication standards and data representation
- General purpose **horizontal solutions**
 - from SCADA to IoT platforms
- Automatic tools to **infer knowledge**
 - wide application of AI (Artificial Intelligence) techniques

} actual game
Changers
so far,
IMHO

IoT: Horizontal Integration vs. Vertical Market



- **One horizontal layer/platform** managing heterogeneous information in an efficient manner
- **Several vertical applications** to provide specific information in a market-tailored manner

Outline

1. Introduction to IoT

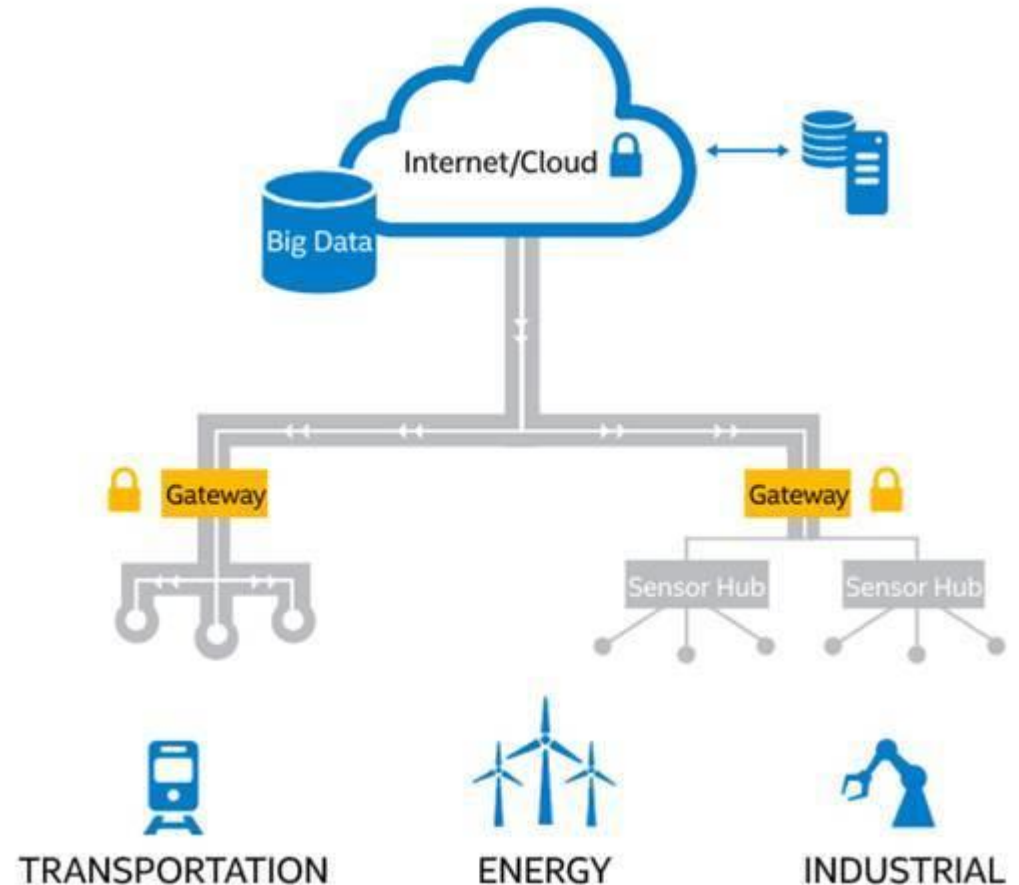
- definition, enabling technologies and concepts to better understand the general framework of IoT solutions
- **layering architecture, cloud computing vs. fog computing**

2. Most relevant components of IoT solutions

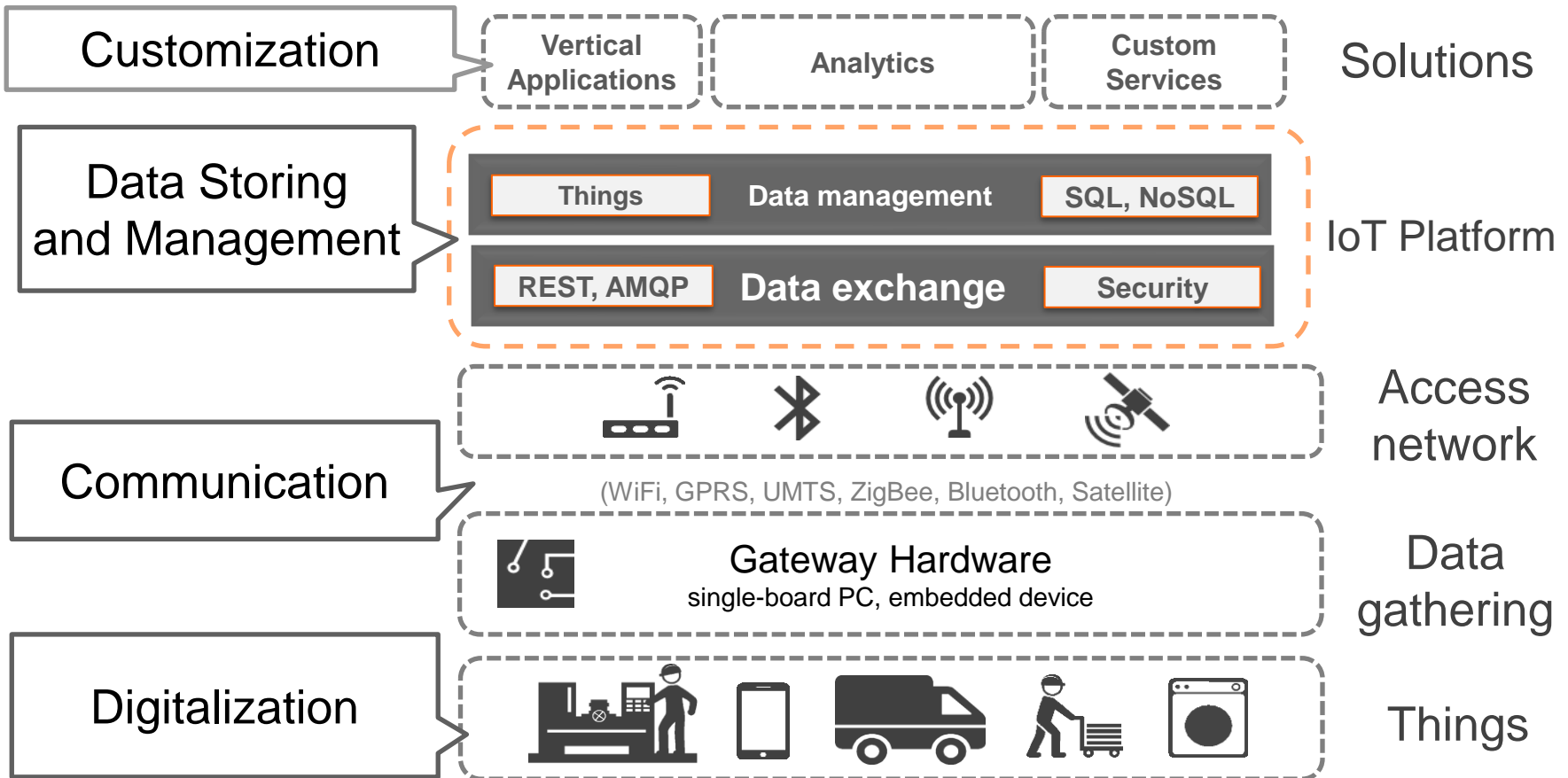
- devices, wireless communication protocols, data exchange protocols

Typical Cloud-based IoT Architecture

- Several heterogeneous **things**, e.g., sensors and actuators
- Multiple **gateways** geographically close to sensors/actuators
 - directly interact with things
 - dispatch data to/from the Internet
- Server-side remote **applications** stored in the Cloud and managing data



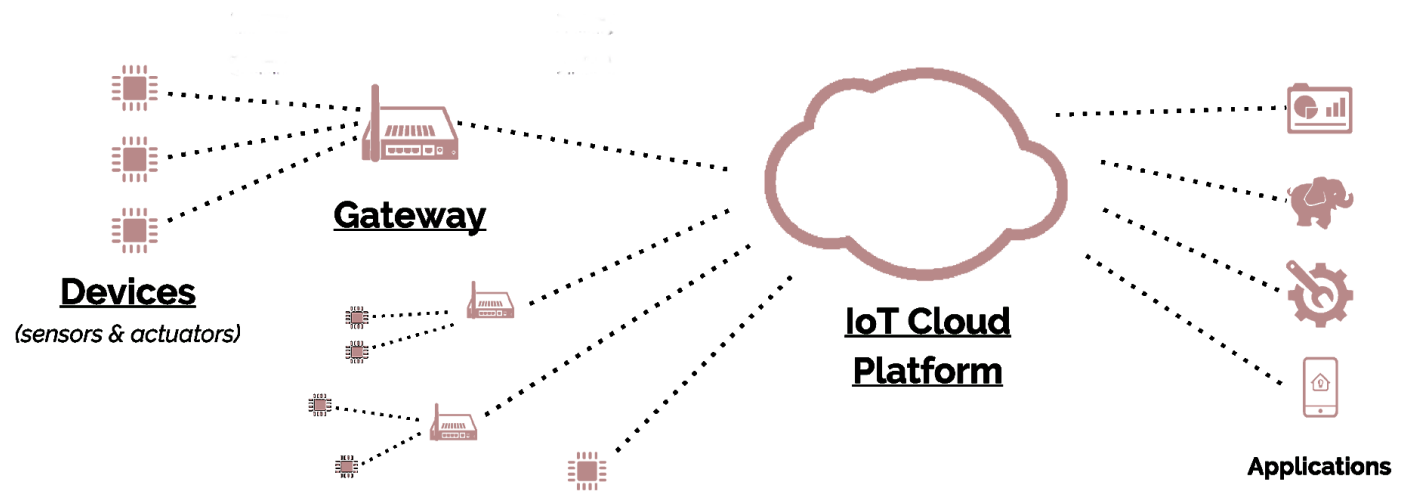
IoT: One Solution, Many Layers



Layering to Simplify Complexity

- **Things:** any physical or digital objects that should be monitored or controlled
 - physical objects must be digitalized
 - virtual objects must be standardized
- **Gateway:** close to one or multiple things to interact with them and send data and command back and forth the Internet
- **Communication protocol:** wired/wireless technology to actually send bytes
- **Data exchange protocol:** software protocol to standardize how information are transported (and eventually also represented)
- **IoT Platform:** data storing and management, application of (simple) aggregation/processing functions on data
- **Analytics:** complex analysis on data to infer new knowledge

Gateway

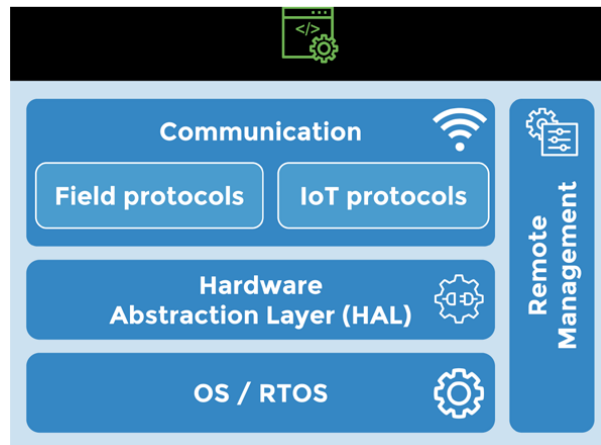


- **Protocol translation** between peripheral trunks of the IoT, eventually provided with lower parts of the communication stacks
- Gateways may also offer pre-processing, **security**, scalability, service discovery, **geo-localization**, **billing**, etc.
- Pre-processing:
 - data **buffering**: temporarily store data to wait for connectivity or to increase efficiency
 - data **efficiency**: temperature read every 1s, but only per-minute average sent
 - data **aggregation**: water level from different silos, but only the sum is sent
 - data **filtering**: send temperature values only if greater than 25°C

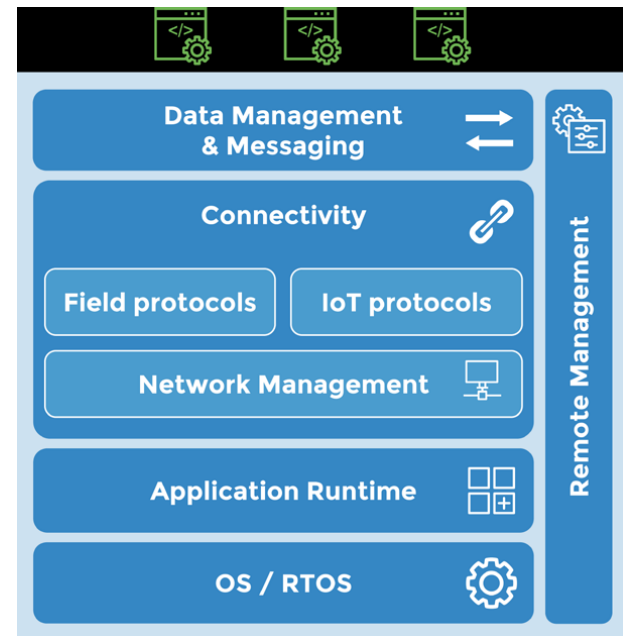
With/Without Gateway

- If there is no gateway, things have to send/receive data on their own
- In case of constrained devices, reduced set of capabilities, e.g.,
 - no security since cryptography is CPU-intensive
 - no data buffering, filtering aggregation
 - no programmability
 - ...

Stack for constrained devices



Stack for gateways



Cloud computing: problem space

*“It starts with the premise that the **data services and architecture** should be on **servers**. We call it **cloud computing** – they should be in a ‘cloud’ somewhere. And that if you have the right kind of **browser** or the right kind of access, it doesn’t matter whether you have a PC or a Mac or a mobile phone or a BlackBerry or what have you – or new devices still to be developed – you can get access to the cloud...”*

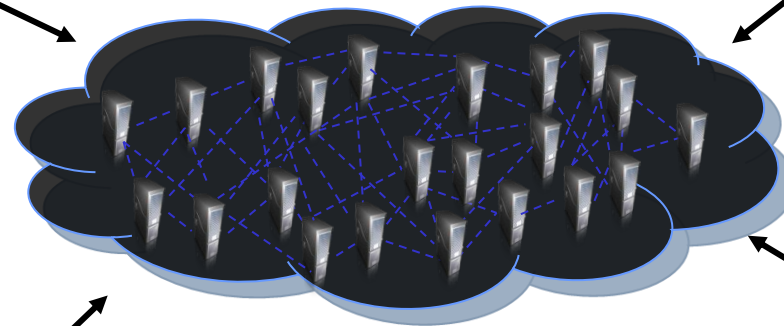
Dr. Eric Schmidt, Google CEO, August 2006



Explosion of data intensive applications on the Internet



Fast growth of connected mobile devices



The Cloud data center



Skyrocketing costs of power, space, maintenance, etc.

Advances in multi-core computer architecture

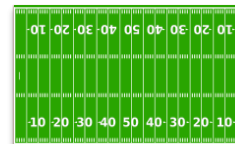


Clouds are Cheaper... and Winning...

- Range in size from “edge” facilities to **megascale**
- **Scale economies**
- Approximate **costs for a small size center** (1K servers) and a **larger, 50K server center**



Technology	Cost in small-sized Data Center	Cost in Large Data Center	Cloud Advantage
Network	\$95 per Mbps/month	\$13 per Mbps/month	7.1
Storage	\$2.20 per GB/month	\$0.40 per GB/month	5.7
Administration	~140 servers/Administrator	>1000 Servers/Administrator	7.1



Each data center is
11.5 times
the size of a football field

Cloud Computing: a brief introduction

- Primary concepts
 - IT **on demand** pricing
 - best benefits in a **reliable** context
 - pool of **virtualized** computer resources
 - rapid live **providing** while demanding
 - systems on **scaling** architecture
- What is a Cloud
 - one Cloud is capable of providing **IT resources “as a service”**
 - one Cloud is an IT service delivered to users that have:
 - **reduced incremental management costs** when additional IT resources are added
 - services oriented management architecture
 - massive scalability



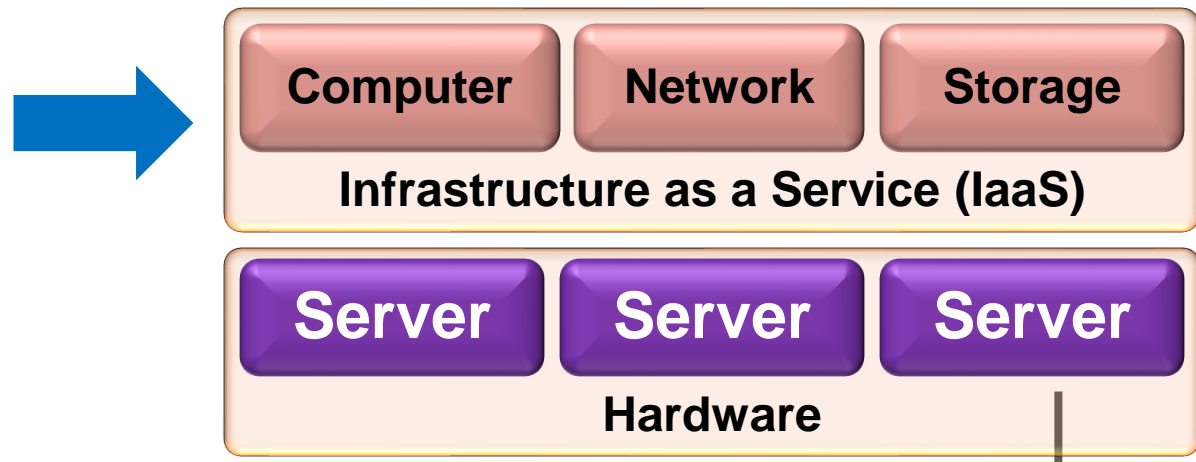
Layered Architecture: IaaS, PaaS & SaaS

- Below the real architecture:
hardware
components &
software products



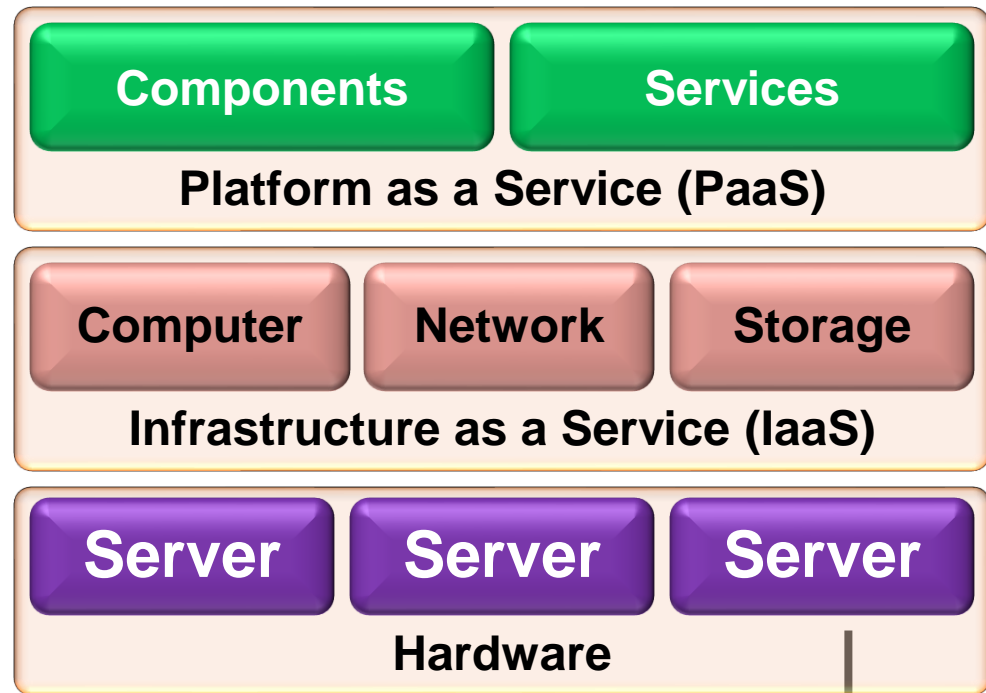
Layered Architecture: IaaS, PaaS & SaaS

- **Infrastructure:** layer to enable the distribution of Cloud services, typically realized by a virtualization platform



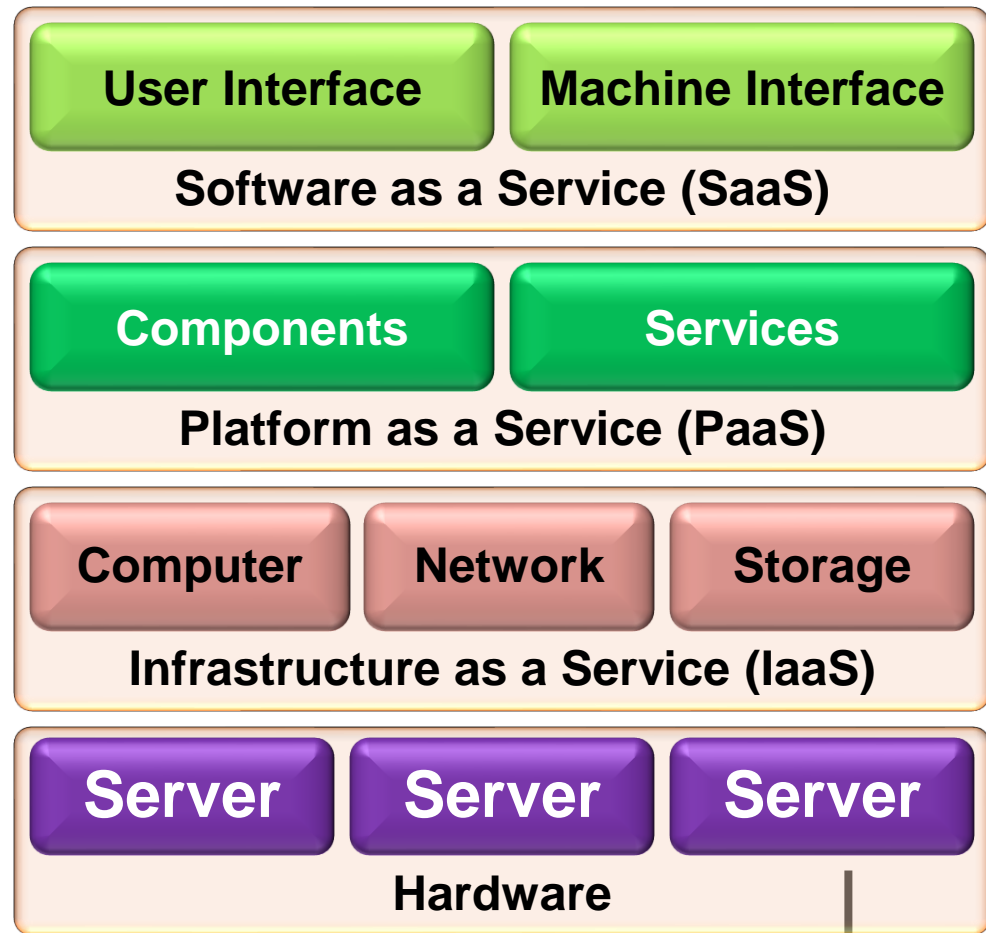
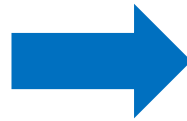
Layered Architecture: IaaS, PaaS & SaaS

- **Platform:** layer to provide to upper layers a set of services and components remotely available



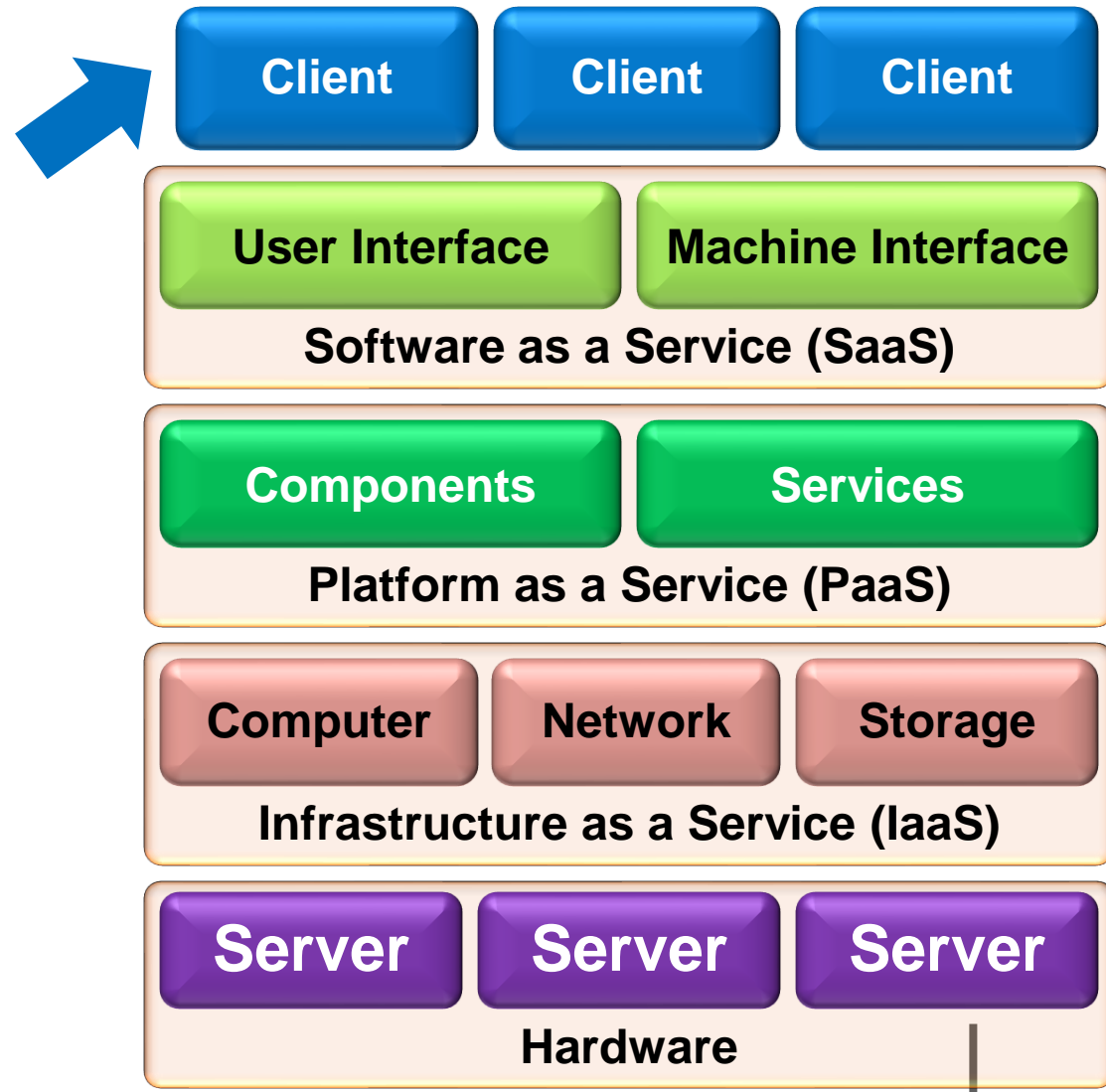
Layered Architecture: IaaS, PaaS & SaaS

- **Application:** layer to install applications, to be available via Web and Internet via Cloud

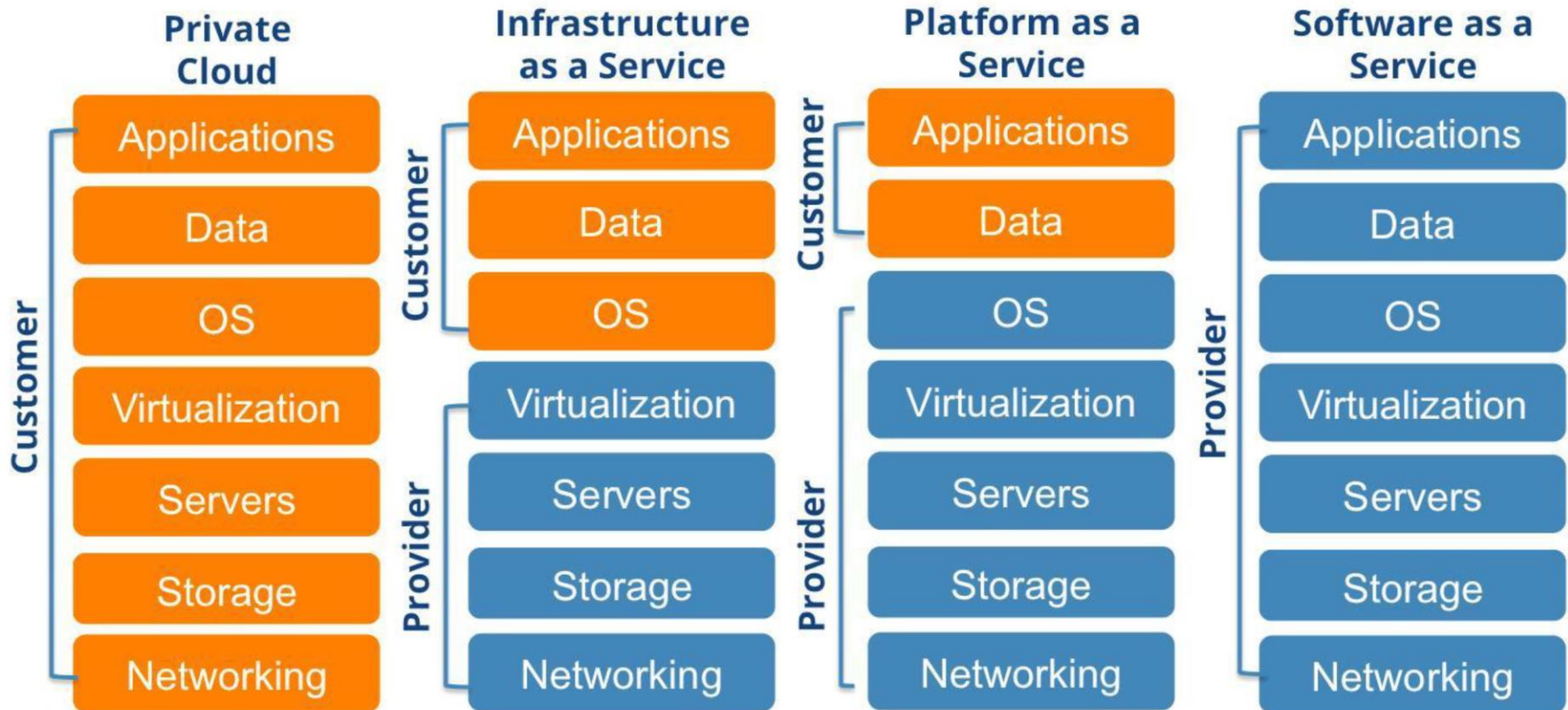


Layered Architecture: IaaS, PaaS & SaaS

- **Client** software to get access to the system. Those applications execute on the **client physical platforms** (remote computers) owned by the final remote user they can communicate with the Cloud via the **available interfaces**





Architecture comparison



Architecture comparison

On-site	IaaS	FaaS	FaaS	SaaS
Applications	Applications	Applications	Applications	Applications
Data	Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking	Networking

 You manage

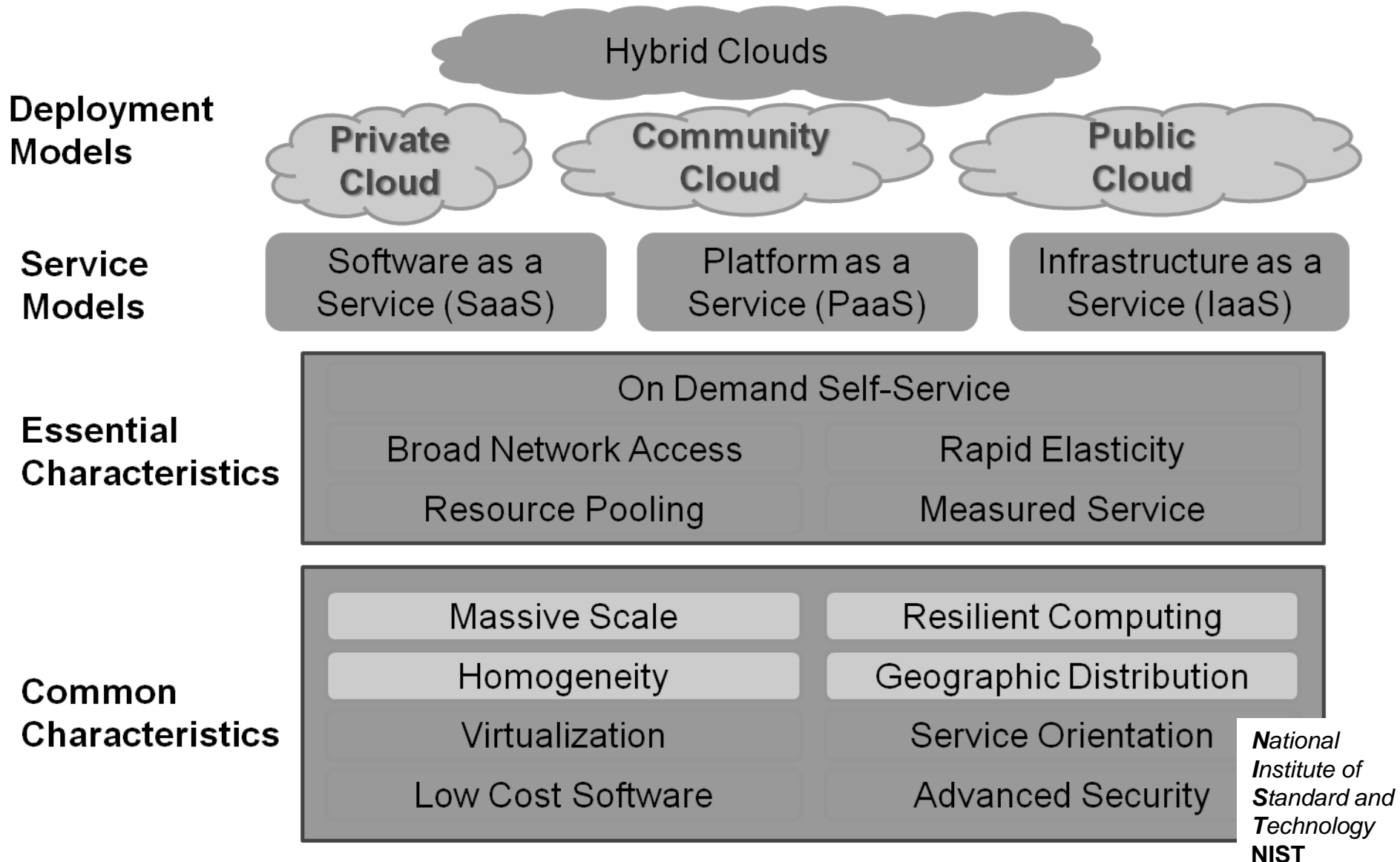
 Service provider manages

Cloud Deployment Models

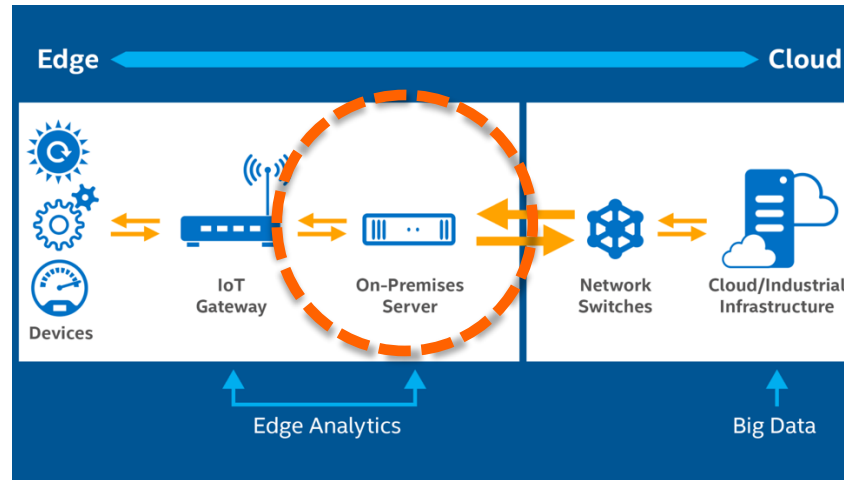
We consider three models

1. **Private cloud** (enterprise owned or leased)
 2. **Community cloud** (shared infrastructure for specific community)
 3. **Public cloud** (sold to the public, mega-scale infrastructure)
- **Bonus Model: Hybrid cloud** (composition of two or more clouds)
 - **NIST classification**
(*National Institute of Standard and Technology*)

The NIST cloud definition framework

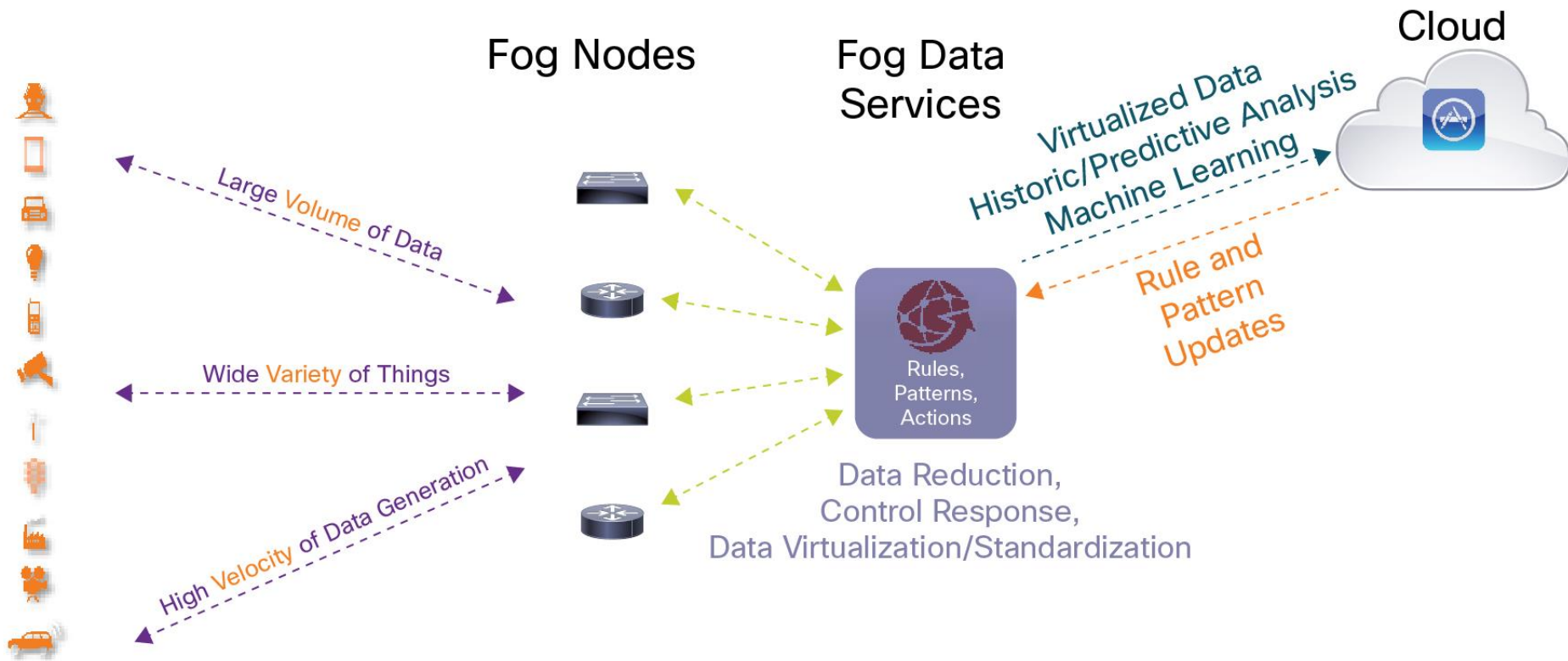


Beyond the cloud: From Cloud Computing to Fog/Edge Computing



- First evolution wave: **IoT Cloud Computing** architecture
 - most of the computation on the Cloud
 - **only gateways are deployed close to things**
 - gateways perform **few and simple tasks**
- Second evolution wave: **IoT Fog/Edge Computing** architecture
 - additional **relatively powerful devices**
 - **close to things**, but between gateways and the Cloud
 - **complex analytical tasks** on the client-side, before sending data to the Cloud

Fog/Edge Computing



- Cisco: **the fog extends the cloud** to be closer to the things that produce and act on IoT data

Fog/Edge Computing for IoT

- Cloud models are not designed for the volume, variety, and velocity of data that the IoT generates
- Fog/Edge Computing allows to
 - minimize **latency**
 - conserve network **bandwidth**
 - address **security** concerns in transit and at rest
 - **move data** to the best place for processing
- When to consider Fog/Edge Computing
 - data is collected at the **extreme edge**: vehicles, ships, factory floors, roadways, railways, etc.
 - **thousands or millions of things** across a large geographic area are generating data
 - it is necessary to analyze and **act on data promptly**, in less than a second

Fog/Edge Computing for IoT use case: Rails



- **Improve passenger safety**
 - analyze and correlate data from cameras on the trains and at stations
 - monitor sensors on wheels and brakes to determine when parts need service before failure causes an accident
- **Prevent cybersecurity attacks**
 - take automated actions such as suspending operations or transferring control to a failover system
- **Alert drivers** to treacherous conditions ahead
 - Fog nodes gather sensor data on tracks and trains to detect unsafe conditions

Outline

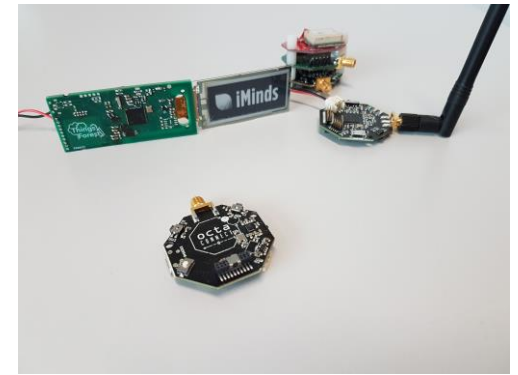
1. Introduction to IoT

- definition, enabling technologies and concepts to better understand the general framework of IoT solutions
- layering architecture, cloud computing vs. fog/edge computing

2. Most relevant components of IoT solutions

- **devices**, wireless communication protocols, data exchange protocols

Constrained Devices



- “A node where some of the characteristics that are otherwise pretty much taken for granted for Internet nodes **at the time of writing** are not attainable, [...] due to **cost, size, and energy constraints**”
- Significant constraints on:
 - maximum code complexity (ROM/Flash)
 - size of state and buffers (RAM)
 - available computational power
 - connectivity

Constrained Networks



- “A network where some of the characteristics pretty much taken for granted with link layers in common use in the Internet **at the time of writing** are not attainable”
- Significant constraints on:
 - low achievable **throughput**
 - high **packet loss**
 - highly **asymmetric links**
 - severe penalties for using **larger packets**
 - limits on **reachability** over time

Classes of Constrained Nodes

Name	Data size (RAM)	Code size (Flash)
Class 0	<< 10 KiB	<< 100 KiB
Class 1	~ 10 KiB	~ 100 KiB
Class 2	~ 50 KiB	~ 250 KiB

Values in 2014

- C0 Devices
 - no direct secure Internet connection
 - use larger devices as gateways/proxies
 - preconfigured and rarely reconfigured
- C1 Devices
 - can use environment specific protocols, e.g., CoAP
 - no access to standard Internet protocols, e.g., HTTP, TLS
 - can be integrated into an IP network
- C2 Devices
 - can use most of protocols

Constrained Devices: Class 0

- Very constrained **sensor-like**
- They need **the help of larger devices** acting as proxies, gateways, or servers to participate in Internet communications
- **They cannot be secured** or managed in a traditional way and are most likely preconfigured
- In other words, **they cannot be used without a gateway**

Constrained Devices: Class 1

- Cannot easily talk to other Internet nodes employing a full stack, e.g., no HTTP, TLS, and XML
- However, they are capable enough to
 - use a protocol stack **specifically designed for constrained nodes** (CoAP over UDP)
 - **participate in conversations** without the help of a gateway node
- Class 1 devices can provide support for the **security functions** required on a large network
- They still **need to be parsimonious** with state memory, code space, and often power expenditure for protocol and application usage

Constrained Devices: Class 2

- Less constrained and **fundamentally capable of supporting most of the same protocol** stacks as used on notebooks or servers
- However, even class 2 devices can **benefit from lightweight and energy-efficient protocols** and from consuming less bandwidth, e.g., using the protocol stacks defined for more constrained devices

Limitations based on energy constrains

- Devices are classified also based on their energy capabilities

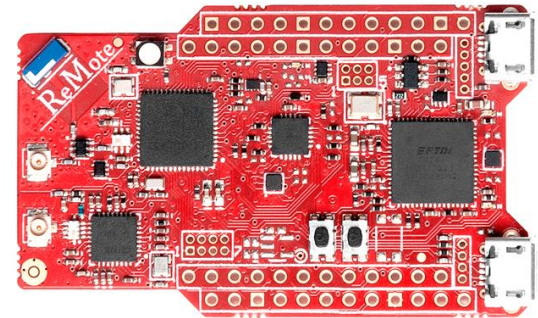
Name	Type of limitation	Example power source
E0	Event energy-limited	Event-based harvesting
E1	Period energy-limited	Battery periodically recharged/replaced
E2	Lifetime energy-limited	Non-replaceable primary battery
E9	No energy limitation	Mains-powered

IoT Device Example: TelosB



- Legacy example of IoT device
 - TelosB Mote for Wireless Sensors Networks
 - 8 MHz 16-bit TI MSP430
 - 10kB RAM, 16kB ROM, 1MB EEPROM
 - IEEE 802.15.4 2.4 GHz radio with antenna
 - temperature, humidity, and light sensors

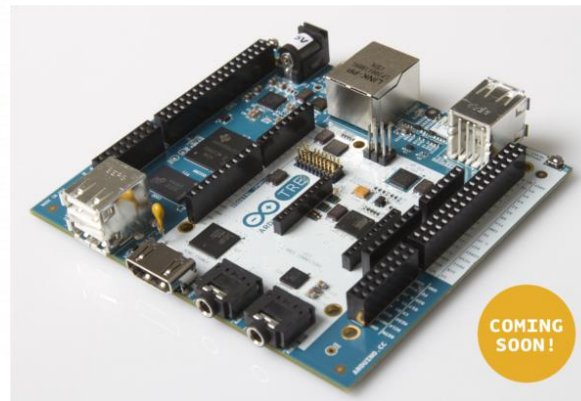
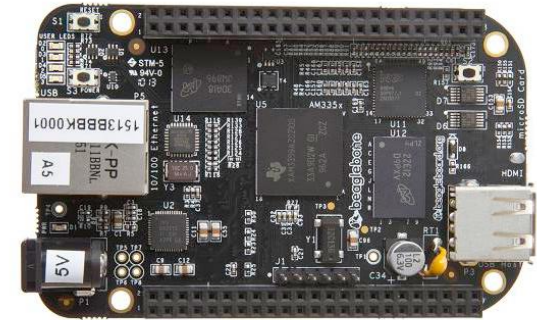
IoT Device Example: Zolertia Re-Mote



- Newer generation board
 - **double radio interface** to enable short (100 meters) and long range (up to 40 Kms) communication
 - powerful 32Mhz ARM Cortex-M3 (32 KiB RAM, 512KiB ROM)
 - **low power** operation down to 0.4uA to operate on batteries for years
 - **several sensing capabilities** (via additional hardware)

Beyond Class 2: Single Board Computers

- In the last 5/10 years
 - tremendous improvement in CPU/memory capabilities
 - dramatically reduced costs
- Modern Single-Board Computers (SBCs) are very cheap (~100\$) and powerful
 - can host complete operating systems
 - extensions via daughterboards
 - mostly designed to be mains-powered



The Evolution of Constrained Devices

- **Phase 1: Smart Tags** to uniquely identify physical things
 - textual identifiers: bar code, QR code
 - digital identifiers: passive/active RFID, i.e., small chips transmitting their unique identification number via wireless at small range
- **Phase 2: Automated Sensing** to transmit sensed data to remote devices
 - periodic or on-demand transmission
 - read-only and battery-powered
- **Phase 3: Smart Devices** allowing to develop modern complex IoT applications
 - sensors and actuators
 - gateways and fog nodes

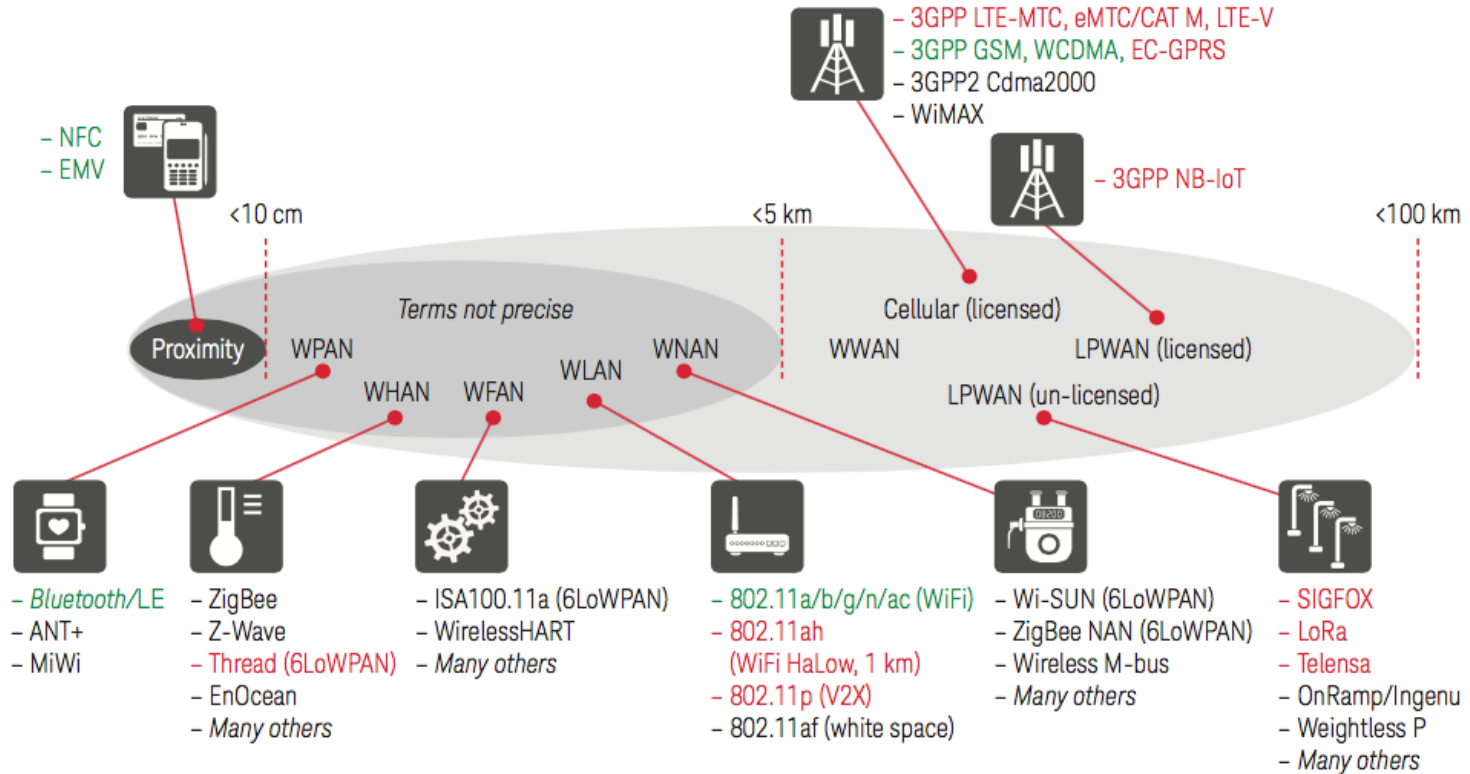
Outline

1. Introduction to IoT
 - definition, enabling technologies and concepts to better understand the general framework of IoT solutions
 - layering architecture, cloud computing vs. fog computing
2. Most relevant components of IoT solutions
 - devices, **wireless communication protocols**, data exchange protocols

Wireless Communication Protocols for the IoT

- The capability of connecting to things in a **seamless, ubiquitous,** and **cheap** manner have pushed the spread of IoT solutions
- IoT wireless communication protocols primarily differs in relation to
 - **coverage range:** from few cm to several km
 - **power consumption:** from few mW to several W
 - **bandwidth:** from few bytes per day to hundreds of MB/s
 - **security:** from plain data to strong encryption
 - **cost:** greatly varying, both for equipment and data transmission

Several Wireless Communication Protocols

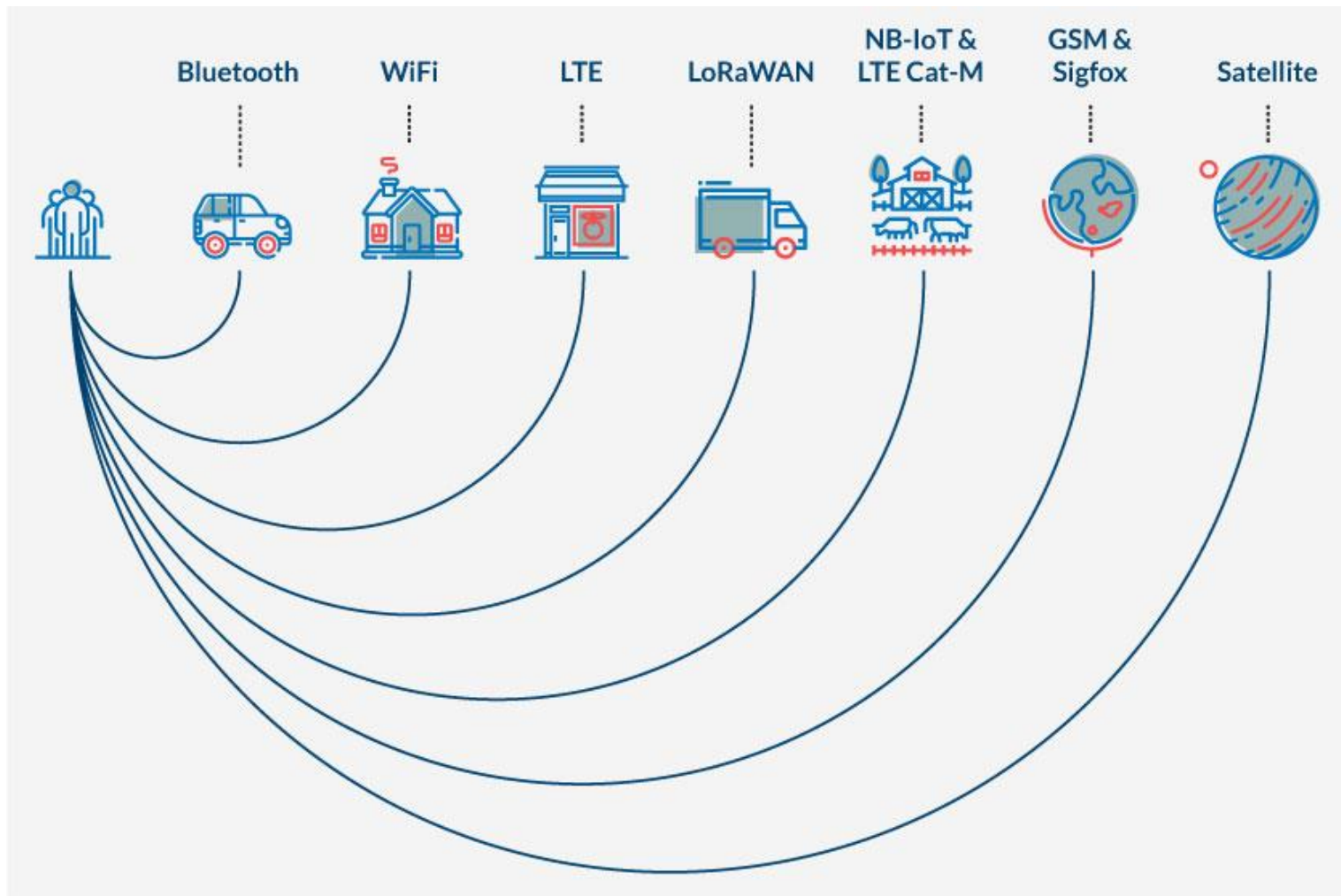


■ : > Billion units/year now
 ■ : Emerging

WPAN: Wireless Personal Area Network
 WHAN: Wireless Home Area Network
 WFAN: Wireless Field (or Factory) Area Network
 WLAN: Wireless Local Area Network

WWAN: Wireless Wide Area Network
 LPWAN: Low Power Wide Area Network

Wireless Protocols per Coverage Range



LTE, the 4th generation mobile network system



- LTE supports voice calls, but designed for **scalability and wireless broadband**
- Range normally less than GSM, but **data rate is orders of magnitude greater**
- Typical range: long, 2Km
- Max output power: medium/high, 0.2 W
- Bandwidth: large, up to hundreds of MB/s
- Security: multiple unique identifiers, static keys, and encryption methods
- Cost: depends on the telco operator
- Good for: **broadband wireless internet connection**, also for streaming security camera videos



- **Low speed, but long range and low power** communication protocol
- Open specification so anyone is free to implement the protocol
- Typical range: long, 5-10 km
- Max output power: low, 0.025 W
- Bandwidth: very low, between 290 bps and 50 kbps
- Security: protocol to exchange to set unique set of AES keys
- Cost: cheap client equipment, needs access point or service provider
- Good for: isolated or private network on a farm or in a city, **ideal for sensors that only seldomly send a value**, like a soil moisture sensor sending its measurements every 10 minutes or a water trough alarming that it is empty

- **Low speed and low power, but also long range**
- Meant for remote meter reading, also used for any remote data uplink
- Proprietary network and protocol

- Typical range: long, 24 km, global (partial) coverage
<https://www.sigfox.com/en/coverage>
- Max output power: low, 0.025 W
- Bandwidth: very low, up to 140 messages per object per day, 12 bytes payload, 100 bps
- Security: no encryption
- Cost: cheap client equipment, for transmission about 1/12€ per device per year

- Good for: remote electricity or water meter reading, mostly useful if you do not need to send data to the device

Other Wireless Protocols

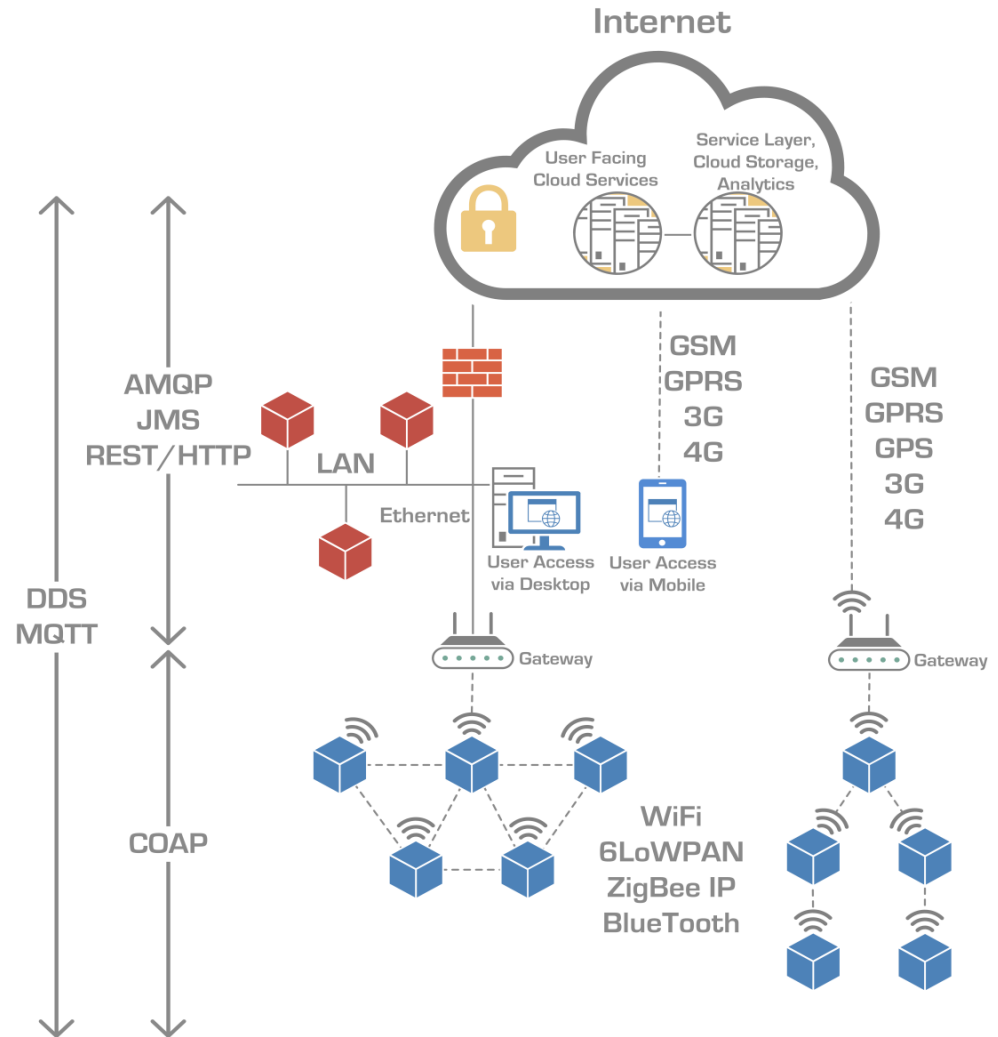
- NB-IoT and LTE Cat-M
 - use cellular channels for long range, limited bandwidth communication
 - still in development or with limited coverage
- Satellite
 - huge coverage
 - extremely expensive

Outline

1. Introduction to IoT
 - definition, enabling technologies and concepts to better understand the general framework of IoT solutions
 - layering architecture, cloud computing vs. fog computing
2. Most relevant components of IoT solutions
 - devices, wireless communication protocols, **data exchange protocols**

IoT Connectivity Problem Space

- Wired/wireless protocols to transport bits
- **Messaging protocols to transfer data**
- **Data:** information and commands described following a given syntax and semantic
- **Messaging protocols to exchange data:** encapsulate data and transmit it via wired/wireless protocols



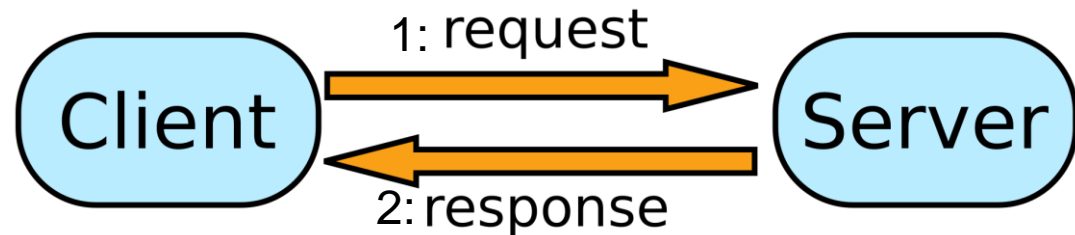
Data Exchange Protocols

- How to encapsulate information and commands in messages?
- Data from gateways to servers: who should initiate the communication?
 - **push**: gateways autonomously decide to send messages to servers, e.g., when new sensed values are available
 - **pull**: servers ask to gateways to send messages, e.g., only when servers actually require sensed values
- Two primary models
 - **request/response**, can be push or pull
 - **publish/subscribe**, usually only push
- Several protocols:
 - request/response: REST/HTTP, CoAP, ...
 - publish/subscribe: MQTT, AMQP, DDS, ...

Multiple Protocols supporting Different Features

- Several protocols thriving to become a de facto standard
- It could be difficult to pick up the **best solution**, since it **depends on requirements** of a given application domain
- Some request/response protocols:
 - REST architectural principles (REpresentational State Transfer), 2000
 - CoAP (Constrained Application Protocol), 2014 (also pub/sub)
- Some publish/subscribe protocols:
 - MQTT (Message Queue Telemetry Transport), 1999
 - AMQP (Advanced Message Queuing Protocol), 2003
 - DDS (Data Distribution Service), v1.0 2003; v1.2 2007
- At different level of abstractions, LonWorks (1999), Modbus (1979)...

Request/Response model



- A **client application** requests services (e.g., send data, require data, perform an addition) and a server application responds to the service request (e.g., by providing the data or the addition results)
- **Direct communication** between client and server
- Data exchange only if **the client starts the communication**
 - the server is not able to contact the client autonomously
- Typical interactions in the IoT scenario:
 - **push**: sensors send data to servers
 - **pull**: actuators request to servers new configurations

REST - REpresentational State Transfer

- REST is not an actual protocol, but substantially a **solution architectural style**
- Promote **client/server** and **stateless** interaction, oriented to the usage of **caching** opportunities, also with possibility of code-on-demand to clients
- Usually based on HTTP, the protocol used to surf the Web
- Very simple, but each time the client has to start the communication from the beginning

REST: Identifying and Interacting with Resources

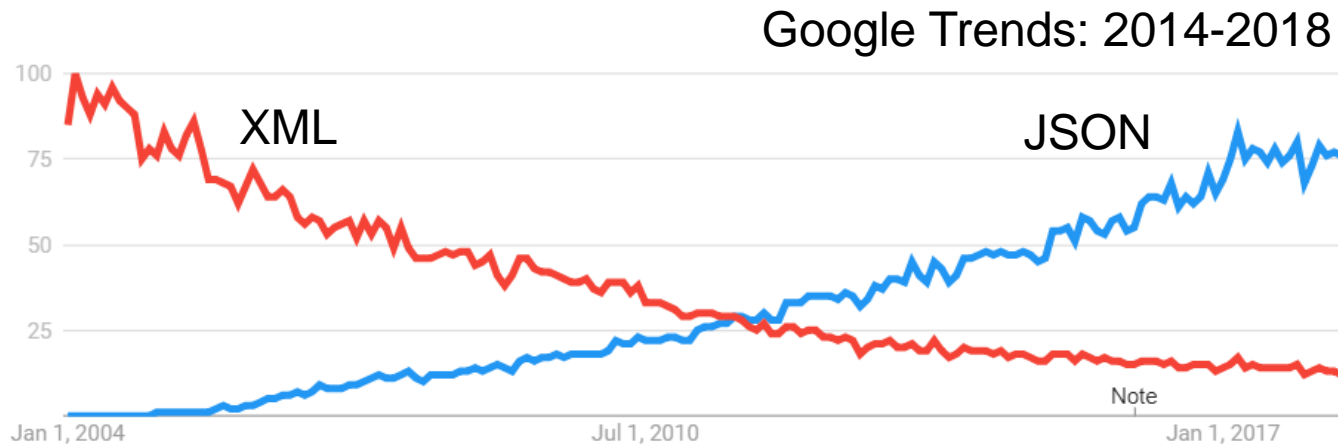
- **URI (Uniform Resource Identifier) to identify the remote resource**
 - any resource has a **persistent identifier**
 - do not transfer resources but their representations via HTTP
 - **endpoint** for a service managing user information:
`www.examples.com/resources/users`
- **HTTP method to interact with remote resources in a standard manner**
 - **GET, retrieve** a specific resource (by id) or a collection of resources
 - **PUT, create** a new resource
 - **POST, update** a specific resource (by id)
 - **DELETE, remove** a specific resource by id

REST: Endpoint Examples

- Remote resource: a software component managing users

HTTP Method	URI	Operation
GET	www.examples.com/resources/ users	Get list of users
GET	www.examples.com/resources/ users/1	Get the user with id 1
DELETE	www.examples.com/resources/ users/1	Delete the user with id 1
POST	www.examples.com/resources/ users/2	Update the user with id 2
PUT	www.examples.com/resources/ users/1	Insert the user with id 1

REST: Formatting data



- JavaScript Object Notation (JSON) most spread syntax for **formatting/serializing data**, e.g., objects, arrays, numbers, strings, booleans, and null
- Resource: a group of users, each one with a first name and a last name

```
{ "users":  
  [  
    { "firstName": "John", "lastName": "Doe" },  
    { "firstName": "Anna", "lastName": "Smith" },  
    { "firstName": "Peter", "lastName": "Jones" }  
  ]  
}
```

REST: AWS IoT example

- Adds a thing to a thing group

PUT **/thing-groups/addThingToThingGroup**
HTTP/1.1
Content-type: application/json

```
{  
  "thingArn": "string",  
  "thingGroupArn": "string",  
  "thingGroupName": "string",  
  "thingName": "string"  
}
```

- Manage “shadows” of things

POST **endpoint/things/thingName/shadow**
HTTP/1.1
Content-type: application/json

```
{  
  "state":  
  {  
    "desired" :  
    {  
      "color" : { "r" : 10 },  
      "engine" : "ON"  
    }  
  }  
}
```

CoAP - Constrained Application Protocol



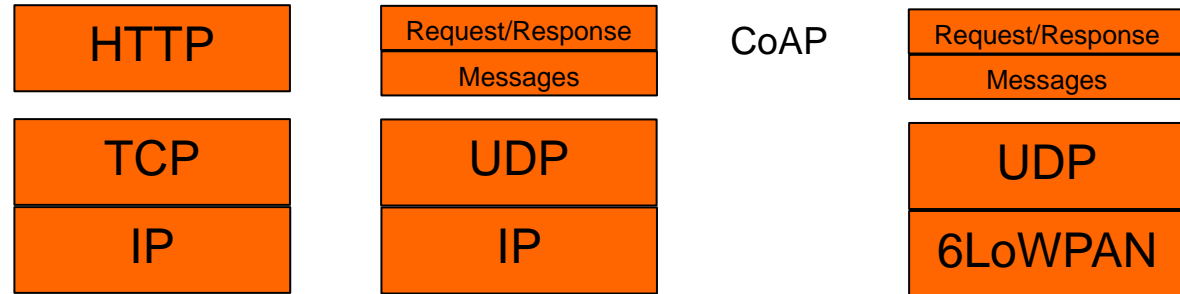
- Designed for M2M/IoT applications such as smart-metering, e-health, building and home automation with:
 - **constrained nodes**, e.g., 8-bit microcontrollers with 16KB RAM and battery-operated
 - **constrained networks**, e.g., Wireless Sensor Networks
- Low-overhead **request/response** protocol that also supports discovery of services/resources
- Based on **UDP** communications (with multicast)
- Inspired by (and compatible with) the **HTTP protocol** and REST architectures: CoAP is a specialized Web transfer protocol

CoAP Features

- **Web RESTful protocol** fulfilling M2M requirements in constrained environments
- **Simple request/response HTTP mapping**, to access CoAP resources via HTTP
- **URI** and Content-type support (a sensor is identified by an URI)
- **Low header overhead** and parsing complexity
- **Security** binding to Datagram Transport Layer Security (DTLS)
- UDP binding with **optional reliability**, supporting unicast and multicast
- **Asynchronous** message exchanges
- Services and resources **discovery**
- Also **publish/subscribe** (and push notifications)
- Simple **caching** (max-age parameter)

- CoAP implementations in many programming language, e.g., C, C++, and Java

CoAP Messaging

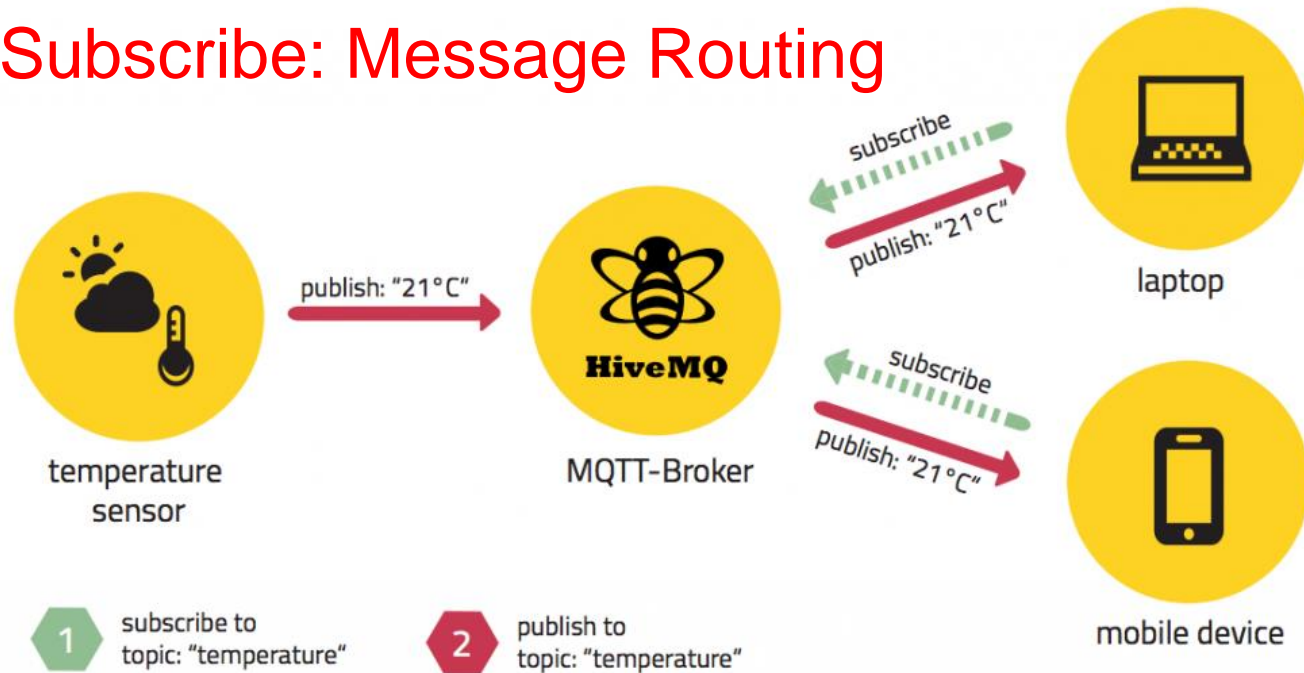


- Duplicate detection and optional reliability
- Possible messages:
 - confirmable message (CON), wait for ACK
 - acknowledgement message (ACK), in response to CON
 - non-confirmable message (NON), no need to wait for ACK
- CoAP on top of UDP allows for multicast requests

Publish/Subscribe model

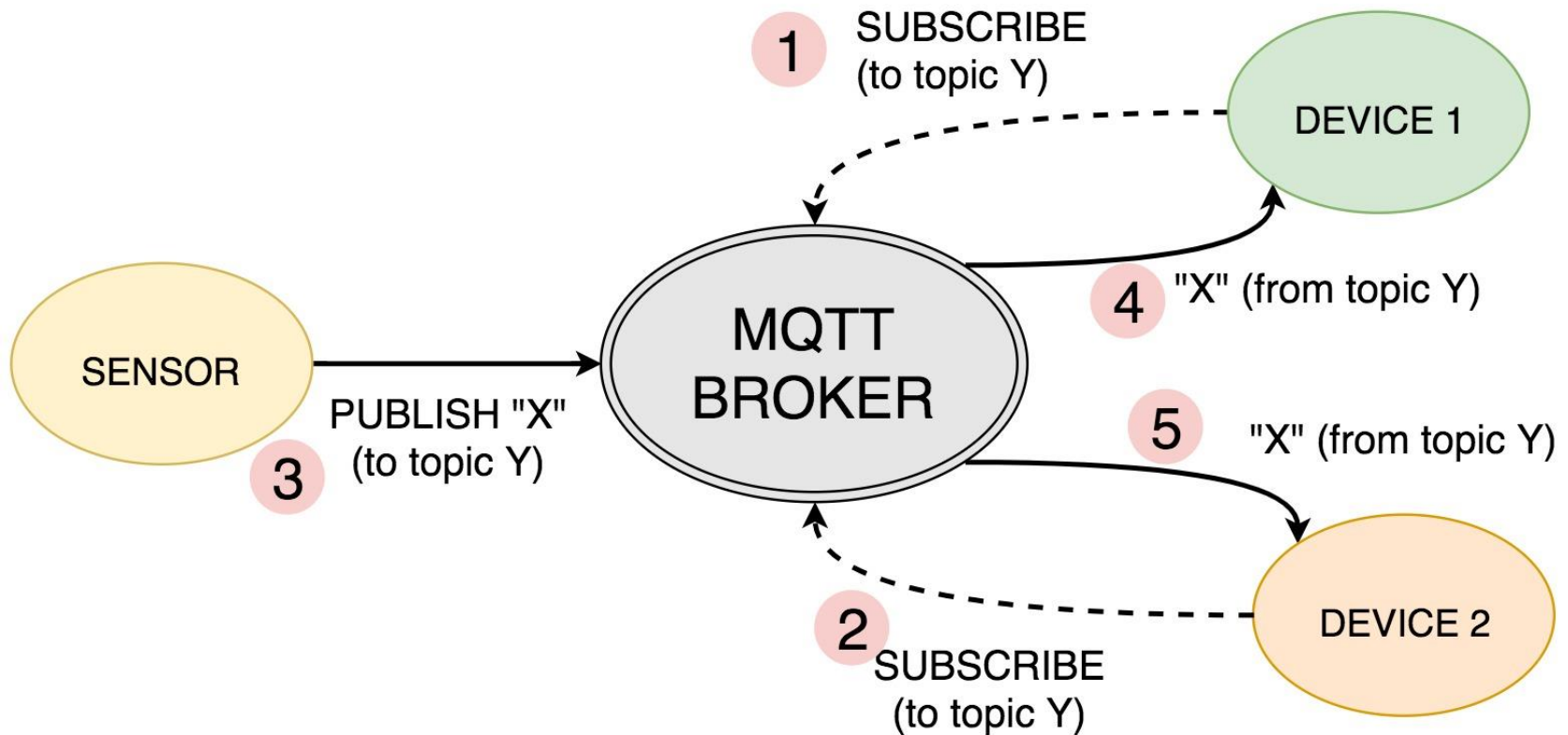
- Publish/subscribe (pub/sub) pattern alternative to traditional client-server model, where a client communicates directly with an endpoint
- Roles:
 - **publisher**: a client sending a message
 - **subscriber**: one or more receivers waiting for the message
 - **broker**: a central component receiving and distributing messages to interested receivers

Publish/Subscribe: Message Routing



- The broker **routes messages** (i.e., selects receivers of a message) based on:
 - **Message Topic**: a subject, part of each message. Receiving clients subscribe on the topics they are interested in with the broker and from there on they get all message based on the subscribed topics
 - **Message Type**: depending on the type of the message
 - **Message Header**: depending on a set of fields of the message
 - **Message Content**: possibly depending on the whole message content (expressive but expensive)

Publish/Subscribe: Typical Sequence



Publish/Subscribe: Decoupling

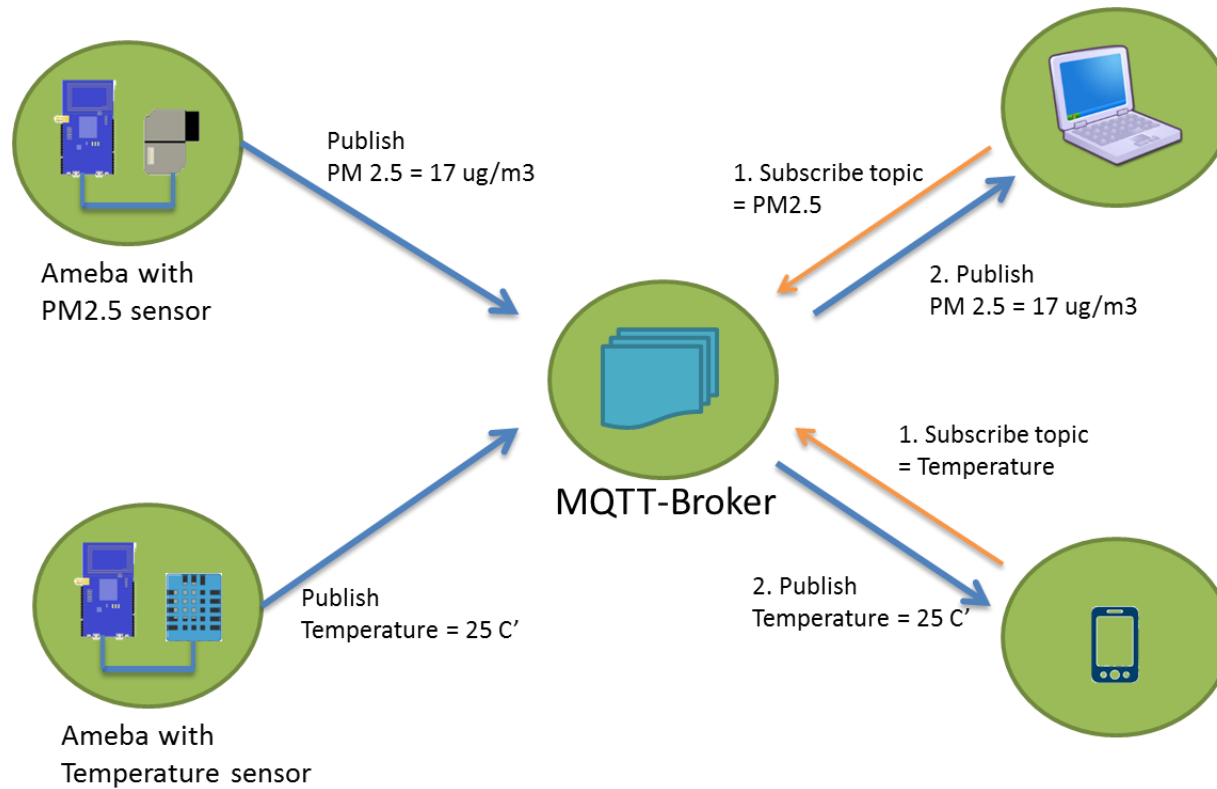
- **Space decoupling:** publisher and subscriber do not need to know each other (by ip address and port for example)
- **Time decoupling:** publisher and subscriber do not need to run at the same time
- **Synchronization decoupling:** operations on both components are not halted during publish or receiving
- Event system as **logically centralized** system
 - anonymous communication
 - possibility to use filters (on headers or entire messages)
 - basic primitives: subscribe, unsubscribe, publish

MQTT - Message Queue Telemetry Transport



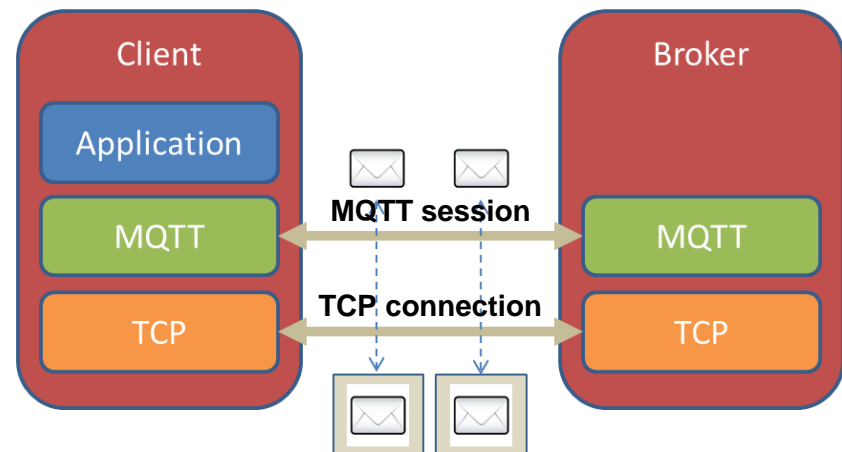
- IBM developed WebSphere MQ as a message-based backbone mainly for Enterprise Application Integration (EAI), MQTT is its evolution:
 - an open standard but it is strongly supported by IBM
 - designed to permit WebSphere MQ to talk with constrained (smart) devices at the edge of the network
- **MQTT: simple, lightweight, broker-based, publish/subscribe, open messaging protocol**
- Ideal for use in **constrained nodes and networks**
 - on embedded devices with limited processor or memory resources
 - where the network is expensive, has low bandwidth or is unreliable
- MQTT-SN for wireless sensor networks, aimed at embedded devices on non-TCP/IP networks (such as Zigbee)

MQTT: Typical Use Case



1. Subscribe to one or more topics
2. Publish to a topic
3. Receive messages related to subscribed topics

MQTT: Features (1)



- **Publish/subscribe** message pattern to provide
 - one-to-many message distribution
 - decoupling of applications
- Use of **TCP/IP** to provide basic network connectivity
- **Small transport overhead** (minimal header length just 2 bytes) and protocol exchanges minimised to reduce network traffic
- **Easy to use** with few commands: connect, subscribe, publish, disconnect
- **Retained messages**: an MQTT broker can retain a message that can be sent to newly subscribing clients

MQTT: Features (2)

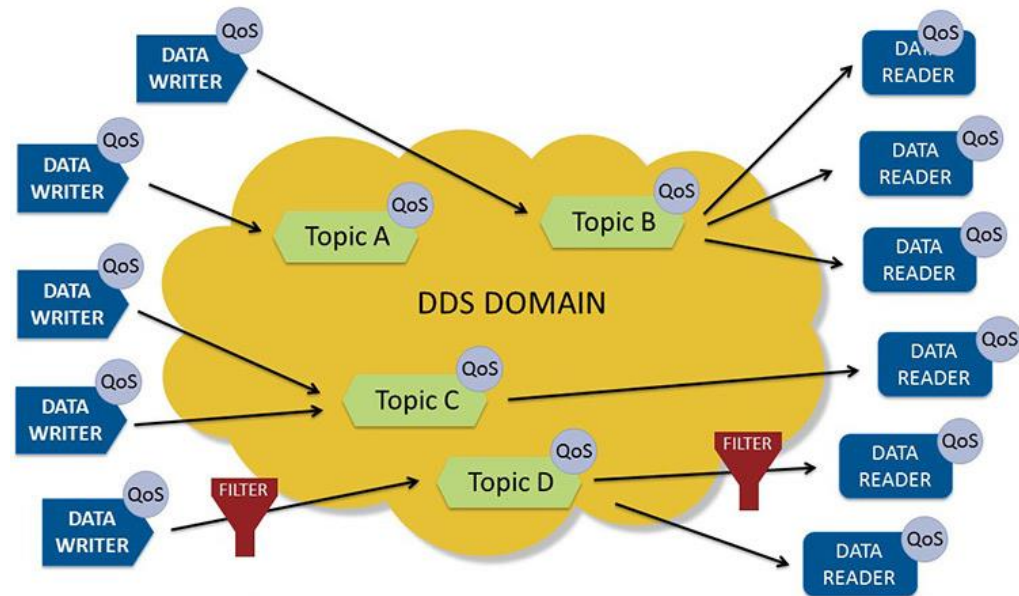
- **Three message delivery semantics** with increasing reliability and cost:
 - Quality of Service (QoS): at least once, at most once, exactly once
- **Durable connection:** client subscriptions remain in effect even in case of disconnection
 - subsequent messages with high QoS are stored for delivery after connection reestablishment
- **Wills:** a client can setup a will, a message to be published in case of unexpected disconnection, e.g., an alarm
- MQTT is a protocol adopted in several platforms: there are MQTT brokers in WebSphere, Mosquitto, RabbitMQ, etc. and clients in several languages

AMQP – Advanced Message Queuing Protocol



- **Richer semantic** than MQTT, e.g., supports topics and queues, but also **heavier**, e.g., the broker is much more complex
- AMQP implementations, e.g., Apache Qpid, focus on providing **several features**: queuing, message distribution, security, management, clustering, federation, heterogeneous multi-platform support
 - most of them (possibly) not essential in the IoT scenario
- Originally developed at JPMorgan Chase in London, AMQP was designed as a message-oriented protocol for the integration of enterprise IT components (Enterprise Message Bus)

DDS - Data Distribution Service



- Publish/subscribe, but **broker-less** (based on multicast)
 - **scalable, real-time, dependable, high performance and interoperable**
- Proposed in several **mission-critical environments** where performance and reliability are essential, e.g., industrial automation, financial applications, air traffic control and management, and even military environments
 - <http://twinoakscomputing.com/datasheets/DDS-Brochure.pdf>
- Standard developed by the Object Management Group

Outline

3. Industrial IoT

- **Industrial data exchange frameworks**
- Industrial IoT platforms

4. Hands-on lab on IoT platforms (in a separate slide deck)

- Azure IoT Hub
- EdgeXFoundry

- Not actually an IoT platform, but an interesting **messaging infrastructure**
- **Robust** messaging for applications
- **Reliable** message delivery
- Distributed deployment as **clusters** for high availability and throughput
- **Federate** across multiple availability zones and regions.
- **Multi-Protocol**: AMQP, MQTT, HTTP, etc.
- **Managing and monitoring** via HTTP-API, command line tool, and UI
- Runs on all major operating systems
- Supports a huge number of developer platforms
- Open source and commercially supported

RabbitMQ: Durable, Persistent, and Synchronized

- Exchanges (Topics) and Queues can be **durable**
 - capable of surviving to a broker restart (as opposed to transient)
- Messages can be defined as **persistent**
 - only persistent messages survive exchange/queue/brokers/channel failures
- **Synchronization**
 - **automatic** acknowledgement model, i.e., verify that the message is actually delivered to the application waiting for it
 - **explicit** acknowledgement model, i.e., wait for explicit acknowledgment sent back by the application
 - immediately at message reception, after processing, ...

- Protocol/platform specialized for **data exchange between shop floor equipment and software applications**, over networks using the Internet Protocol (IP)
- Lightweight, open, extensible, and **read-only**
 - introduced only for the monitoring of numerically controlled machine tools
- MTConnect exploits several **Internet open standards**
 - data (from shop floor devices) is presented in XML format
 - communication via HTTP with a RESTful interface
 - Lightweight Directory Access Protocol (LDAP) for discovery services
- MTConnect was developed at UCB (University of California Berkeley) and GeorgiaTech, now managed by the MTConnect Institute, a forum of companies and organizations

MTConnect Dictionary

- Standard dictionary for manufacturing data; see the example below:

Data Item type/subtype	Description
ACCELERATION	Rate of change of velocity
ACCUMULATED_TIME	The measurement of accumulated time associated with a Component
ANGULAR_ACCELERATION	Rate of change of angular velocity.
ANGULAR_VELOCITY	Rate of change of angular position.
AMPERAGE	The measurement of AC Current or a DC current
ALTERNATING	The measurement of alternating current. If not specified further in statistic, defaults to RMS Current
DIRECT	The measurement of DC current
ANGLE	The angular position of a component relative to the parent.
ACTUAL	The angular position as read from the physical component.
COMMANDED	The angular position computed by the Controller.
AXIS_FEEDRATE	The feedrate of a linear axis.
ACTUAL	The actual federate of a linear axis.
COMMANDED	The feedrate as specified in the program.
OVERRIDE	The operator's overridden value. Percent of commanded.
CLOCK_TIME	The reading of a timing device at a specific point in time. Clock time MUST be reported in W3C ISO 8601 format.
CONCENTRATION	Percentage of one component within a mixture of components
CONDUCTIVITY	The ability of a material to conduct electricity

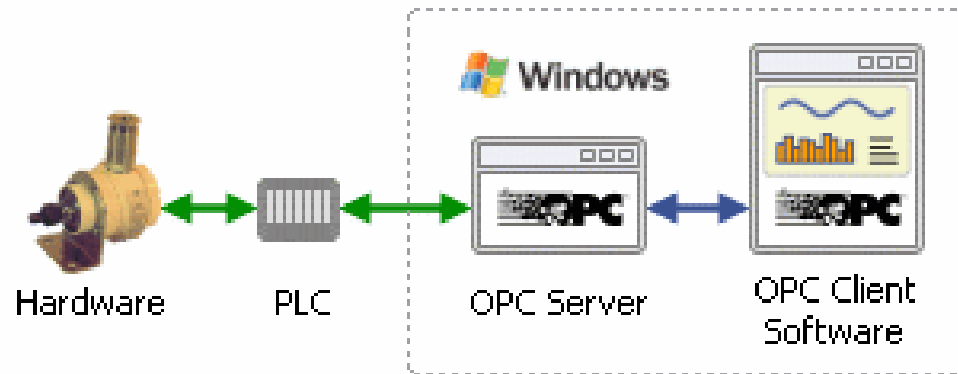
OPC Foundation



- OPC = OLE (Object Linking and Embedding) for Process Control
- Microsoft software architecture for industrial control systems (1996)
- Designed for connecting Windows based PC with **PLC and SCADA systems in industrial automation**
- Based on COM and DCOM, so (formerly) tightly related to Microsoft technology

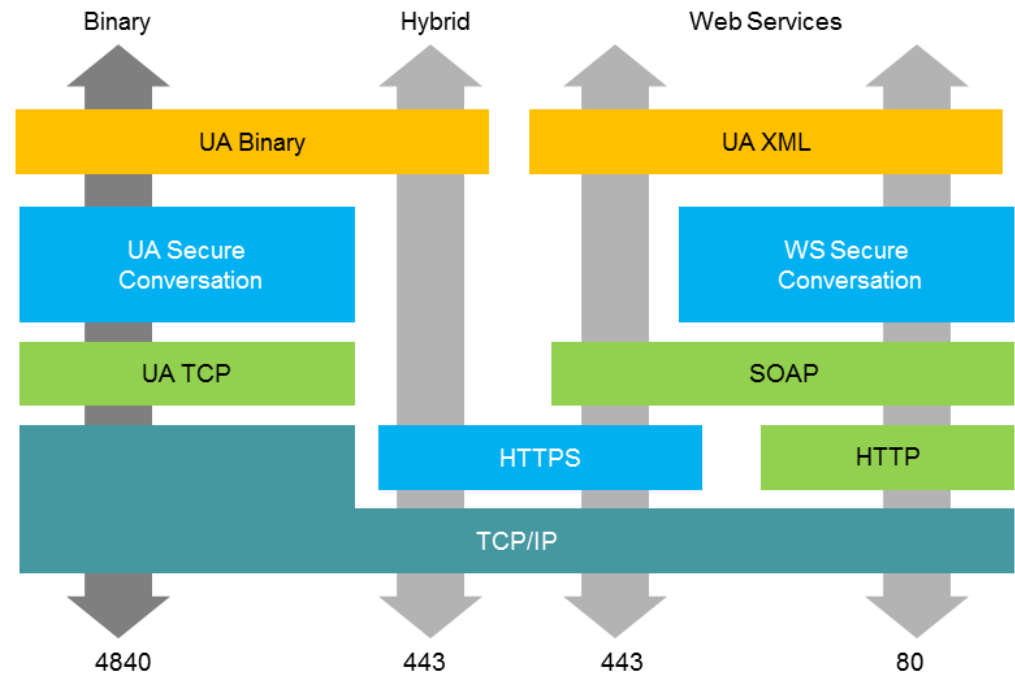


OPC Architecture



- Client/Server architecture
 - the client and the server can be on the same or different machines
- Interface between OPC Server and a specific PLC always **vendor-specific**
 - typically each PLC producer sells also its specific OPC Server component

OPC UA – Unified Architecture



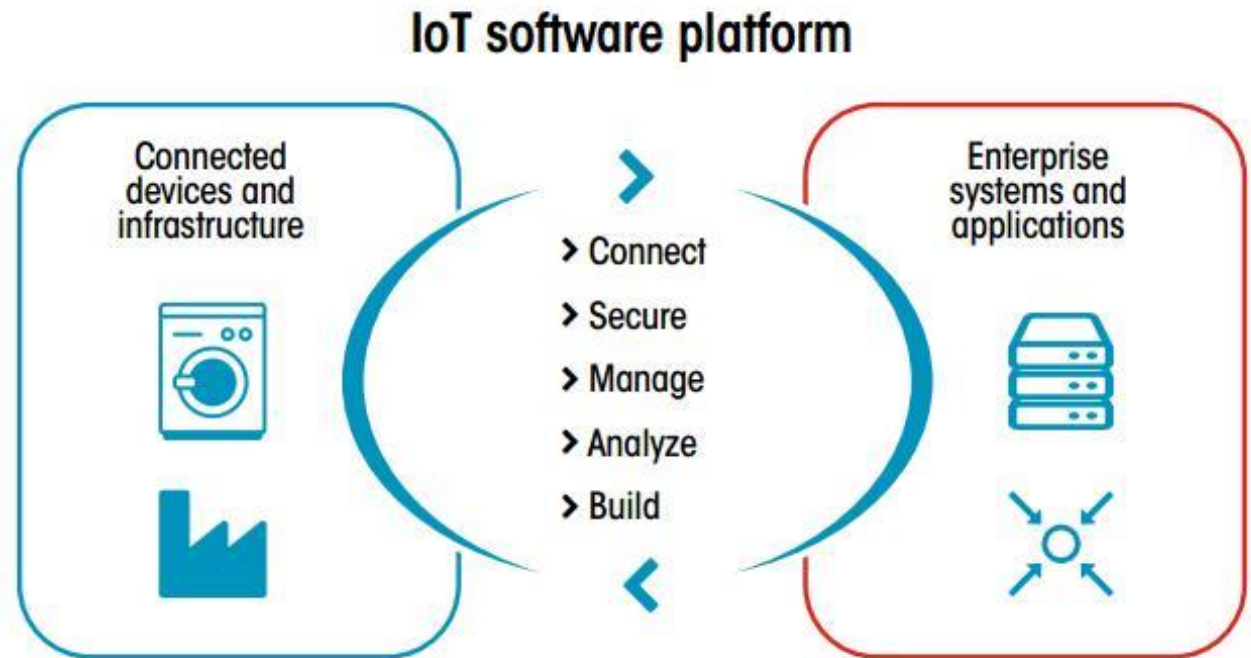
- Modern release of OPC (2006), greatly differing from the previous one
- **Platform independent:** PC, ARM, Cloud, Windows, Linux, Android...
- **Secure:** authentication, encryption, auditing, fault tolerance...
- **Extensible:** multi-layered architecture of OPC UA provides a “future proof” framework

Outline

3. Industrial IoT and Industry 4.0
 - Industrial data exchange frameworks
 - **Industrial IoT platforms**
 - Industry 4.0: standards & trends

4. Hands-on lab on IoT platforms (in a separate slide deck)
 - Azure IoT Hub
 - EdgeXFoundry

IoT Platforms



- IoT Platform between things/devices and business applications
 - **connect** devices to gather information
 - **secure** communication with devices and applications
 - **manage** devices to control their behavior
 - **analyze** data, e.g., with AI
 - **build** applications, also interacting with CRM/ERP/...

Several IoT Platforms

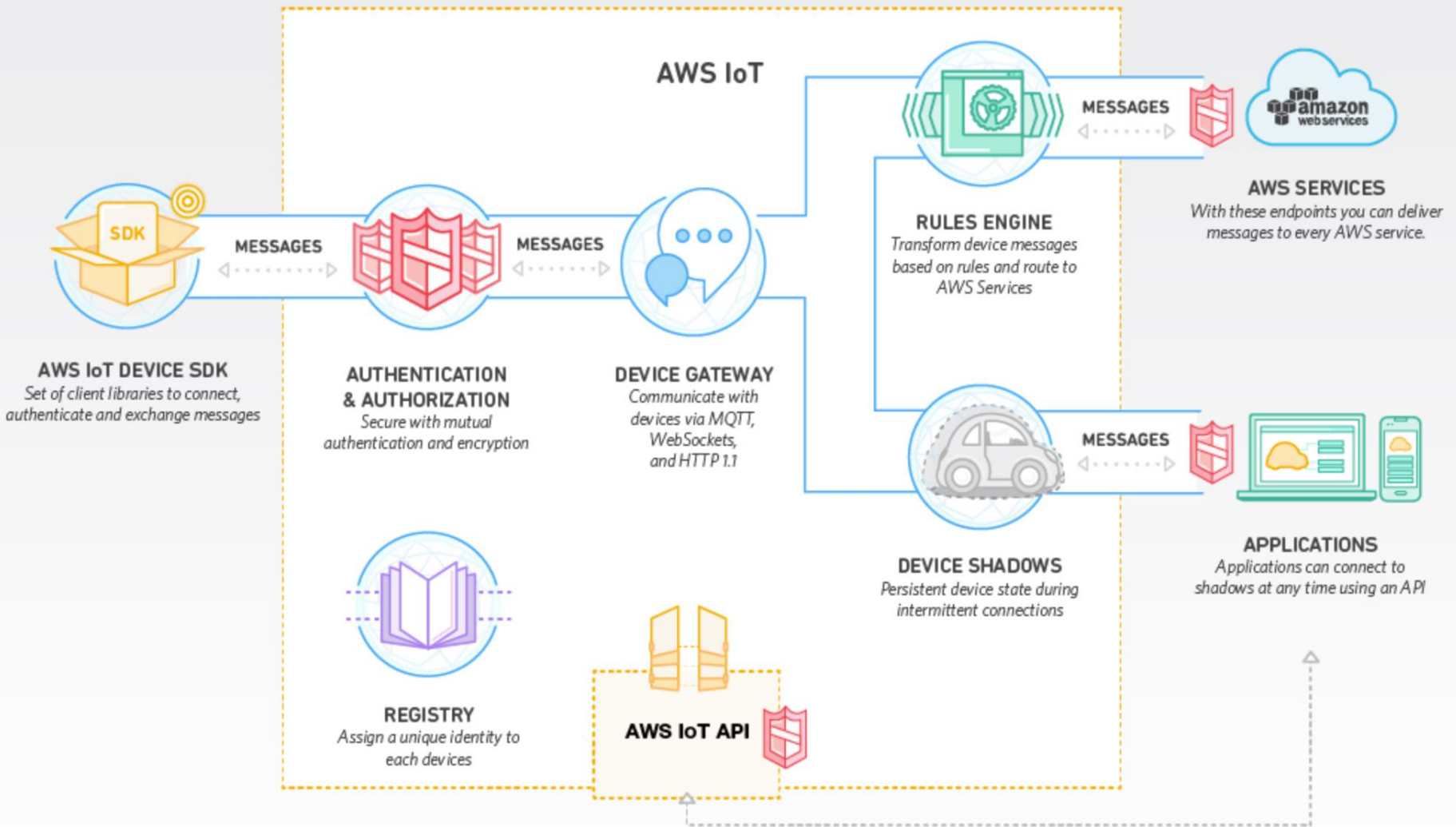
We go a bit deeper into a selection of the many available ones:

- Amazon Web Services (AWS) IoT
- Microsoft Azure IoT Hub
- Mindsphere by Siemens
- EdgeXFoundry

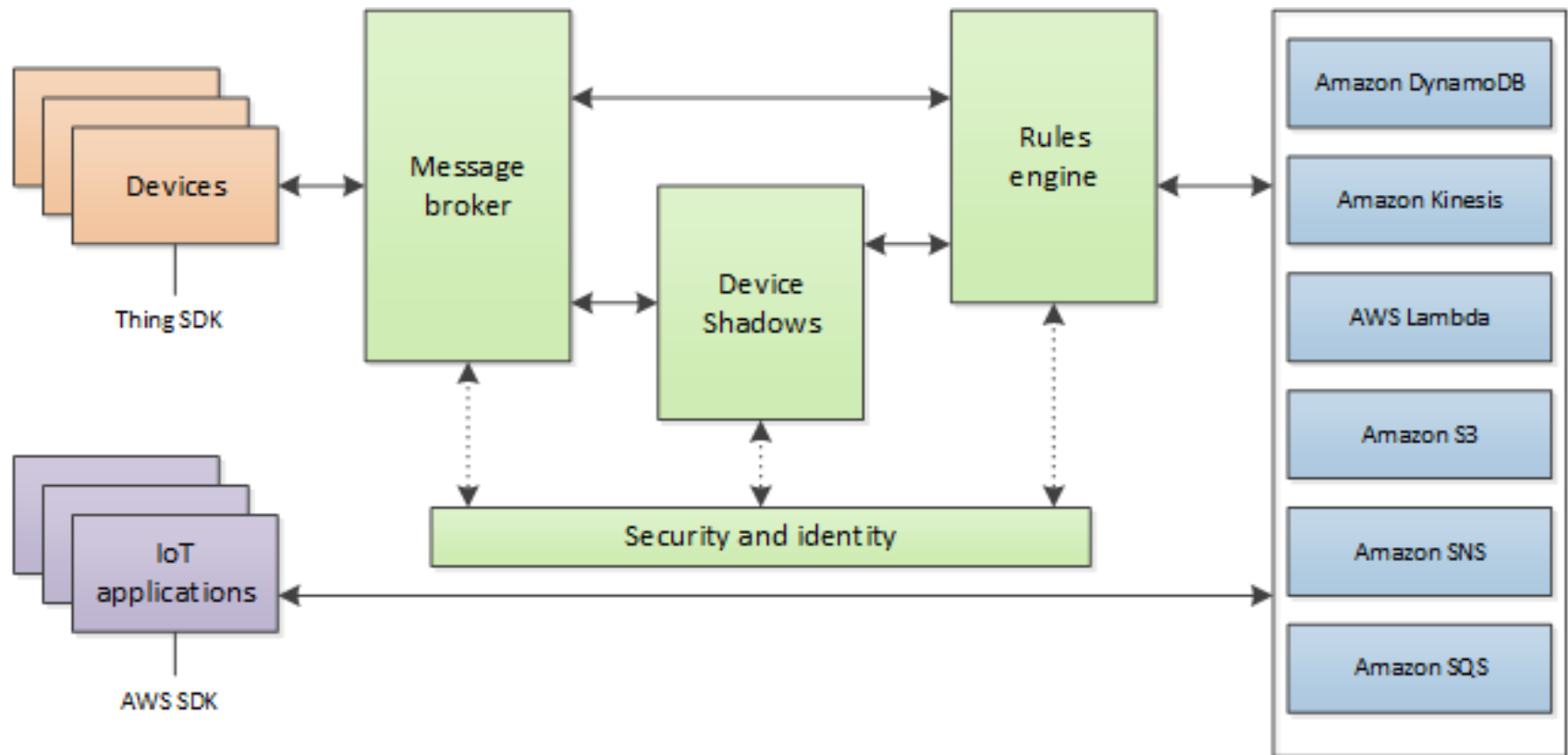
And several others:

- Google Cloud Platform
- ThingWorx IoT Platform
- IBM Watson
- Carriots
- Kaa
- ...

Amazon Web Services (AWS) IoT



AWS IoT: Architecture



- <https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html>

AWS IoT: Main Components (1)

- **AWS IoT Device SDK** connects devices to AWS IoT by using the MQTT, HTTP, or WebSockets protocols
 - the Device SDK supports C, Java, JavaScript, Arduino, iOS, Android,...
- **Device Gateway** supports (secure) communication with devices, by using a (1:1 or 1:n) **publish/subscribe** model
 - supports MQTT, WebSockets, and HTTP 1.1 protocols
- **Authentication and Authorization** with mutual authentication and encryption
 - authentication with native AWS (called 'SigV4') as well as X.509 certificate based authentication
 - AWS facilitates the whole certificate process management
- **Registry** assigns a **unique identity** to each device
 - also supports **metadata** describing capabilities of devices, e.g., whether a sensor reports temperature and if data in Fahrenheit or Celsius

AWS IoT: Main Components (2)

- **Device Shadows**, i.e., persistent, virtual version of devices
 - applications or other devices exchange messages and **interact through the Shadow Device**
 - Device Shadows **persist** the last reported state and desired future state of each device even when the device is offline
 - exploit it to **retrieve the last reported state** of a device or **set a desired future state** through the API or using the rules engine
- **Rules Engine**, supporting to build IoT applications that gather, process, analyse, and act on data generated by connected devices
 - evaluates inbound messages published into AWS IoT and **transforms and delivers** them to another device or a cloud service, based on business rules you define
 - a rule can apply to data from one or many devices, and it can take one or many actions in parallel

Azure IoT Platform

Azure IoT Platform is a flexible architecture to solve industrial IoT problems that consists of many different services and modules that can be combined together, according to specific requirements

Azure IoT is internally organized in **3 levels of abstraction / layers**

- **Things:** data is generated from **devices**, has a brief processing through **edge applications** and is sent through some **connectivity service** to the next layer
- **Insights:** here data really get some manipulation. Can be stored in different kinds of storage, aggregated and analyzed through **stream analytics** and **machine learning tools** to derive some insights
- **Actions:** Finally those insights are visualized into graphs and **dashboards** that can help companies to take some actions on their plants

Azure IoT Architecture



Azure IoT Main Components

- **IoT Hub: manages the communication** among Azure services and in particular with IoT devices
- **IoT Edge:** service to deploy and manage applications over remote devices in their locations
- **Stream analytics:** one of the many analytics tools offered by Azure, focused on real-time data and complex event processing from IoT devices
- **Machine Learning (ML):** cloud-based tool to train, deploy, and manage different ML models with support for different languages and popular ML libraries
- **Blob storage:** object storage solution for non-structured data, optimized for big data

Azure IoT Hub



Service hosted in the cloud that acts as a message hub for bi-directional communication between IoT devices and other modules of the application.

It allows sending both the **telemetry data from devices to cloud** and **commands to change devices' behaviour from the cloud**.

Different features:

- Secure communications: complete control through a specific authentication for each device, different types (SAS, X.509)
- Built-in Routing functionality with rules to automatically dispatch messages

Azure IoT Hub

- **Complete integration** with other Azure services to build complete industrial solutions using among the others: Azure analytics, ML and Data Lake storage
- **Configuration of devices using metadata and state information** which are stored in the hub and can be queried from the up-stream

Azure IoT Connectivity

The connection between devices and the hub is made possible through **Azure IoT devices SDK**, which are libraries that provides the methods to connect to the platform. Many different languages and transport protocol are supported

Languages

- C
- Java
- Python

Protocols

- HTTPS
- **AMQP**
- **MQTT**

Azure IoT Edge

Services can run at the edge of the architecture in order to enable fast data aggregation and analytics, running it closer to the devices and making it possible to respond quicker to anomalies and avoid sending terabytes of data up to the cloud

Three main components:

- **Edge Modules: containers** that run Azure services and apps executed at the edge in the device locality
- **Edge Runtime:** environment that runs on each device and **manages deployed modules**
- **Cloud interface:** to remotely monitor the devices

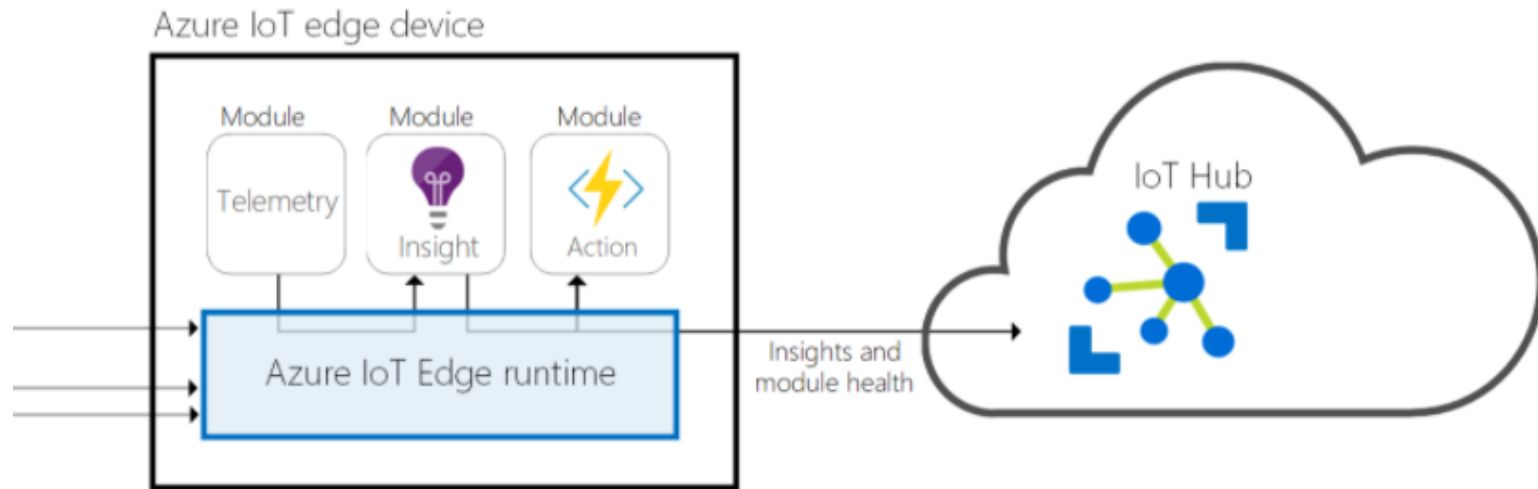
Edge modules

Smallest unit of computation in Azure IoT Edge, **implemented as Docker compatible containers**. They can contain other Azure services or specific code written and containerized for the edge. Multiple modules can be deployed over the same device in order to communicate and create a more complex data pipeline

Every module is made of 4 conceptual elements:

- Image: package containing **the software of the module**
- Instance: **unit of computation** that runs the image on the device; it is started by IoT Runtime
- Identity: information about **credentials and permissions** associated with each module.
- Twin: **JSON document** that stores metadata regarding the **status of a module and configuration**

Edge runtime



Edge runtime is made by two modules: **Edge Agent that deploys and monitors other modules** and **EdgeHub** that is responsible for **communication**. The runtime once installed turns a device into an Edge device and gives them many features:

- Ensures that modules are running and report module health to the cloud for monitoring
- Manages communications between edge devices and normal devices, upstream with the cloud, and downstream with machines

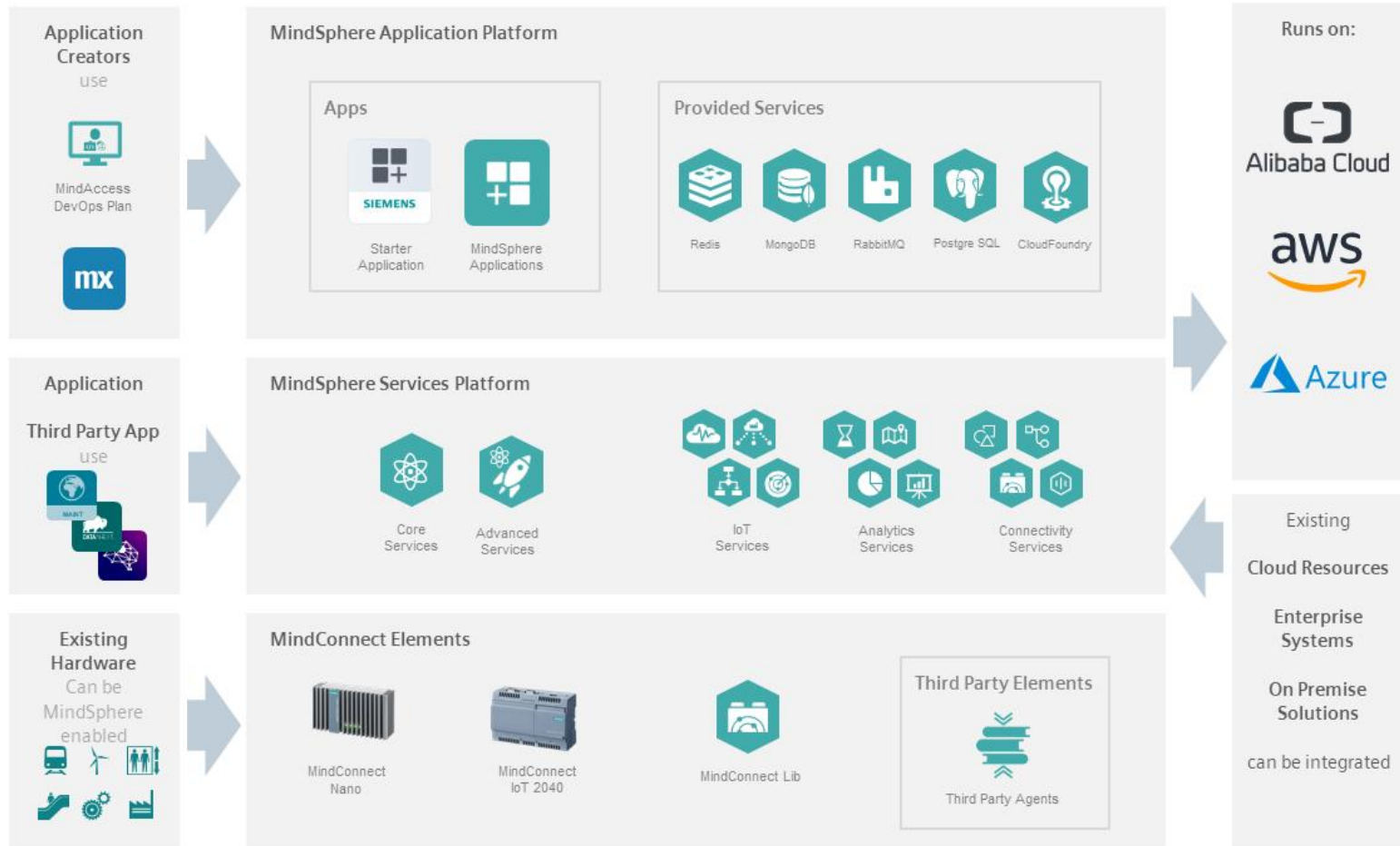
Siemens MindSphere Platform

MindSphere is a **complex PaaS** to help developers build industrial IoT solution through their services and apps.

MindSphere consists of three distinct layers:

- **Application system:** environment powered by CloudFoundry, deeply integrated with MindSphere ecosystem and services. Deployment, scaling, and monitoring are some of the features provided.
- **Services:** support services exposed through REST API to make development easier.
- **MindConnect Elements:** different communication possibilities to connect machines and plants to MindSphere regardless of the manufacturer.

MindSphere Architecture



MindConnect Elements

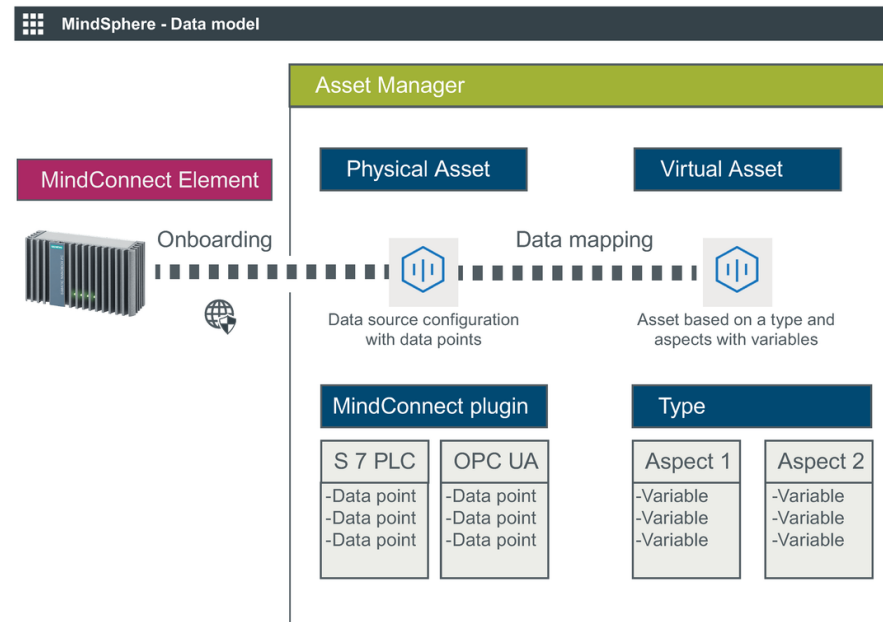
MindConnect is the layer that **provides connectivity** to/from Mindsphere platform.

There are both hardware and software ways to connect:

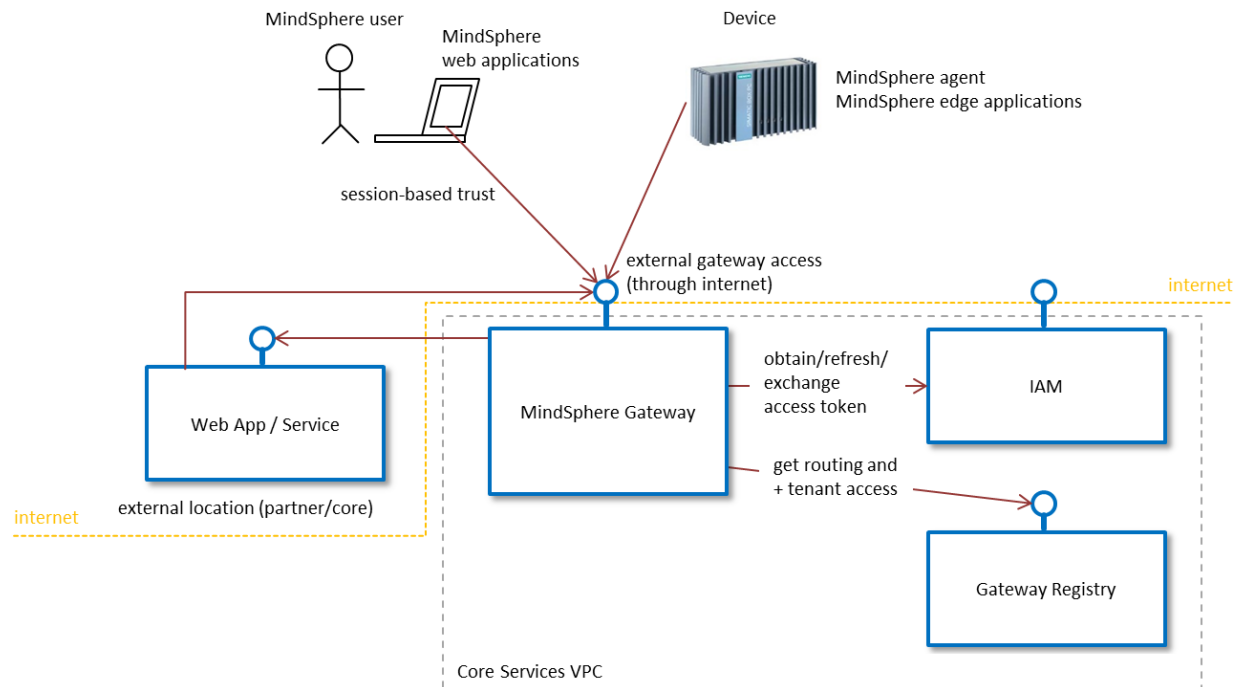
MindConnect series and Simatic edge devices are meant to provide secure, reliable and fast connection.

They can interact with popular industrial protocols such as OPC-UA, and MQTT or use S7 proprietary protocol.

There is always the possibility to build your own connector using specific SDK library for a certain protocol.



MindSphere Gateway



MindSphere provides a **gateway service** between web application clients, agents and edge applications, and backend services.

Routing has to be defined in the gateway registry from the provider of services as well as the identity to grant the authorization.

Mindsphere IoT Services

- **File Service:** provides a basic file management tools with an interface to manage, search and download data. Each file is linked to an asset, has some properties that describe it and can be retrieved after a connection to the IoT gateway. APIs provide CRUD operations on files.
- **Time Series service:** a service similar to File service, but **only for the data type time series**. Time series records are linked to an asset and an aspect and consist of a timestamp, one or more values and a quality indicator for variables.
- **Aggregation service:** creates aggregated **summaries of time series data and interfaces to read them**. It is particularly useful to reduce the amount of data sent to cloud for analysis. Once chosen a time interval to aggregate provides standard aggregate information such as min, max, average, tot number of points

Mindsphere Connectivity Services

- **Agent management** service: provides API to create, update, delete and manage the **rights of agents**. **Agents are the primary actors that perform actions on data in Mindsphere**, when created they are assigned an access token which is the key to get authenticated and authorized to communicate
- **MindConnect API: API to send data securely** from the shop floor devices to Mindsphere. Only agents that supports HTTP processing, TLS and JSON parsing have the requirements to communicate. It allows also to set the configuration for both the data source from which retrieve the data and the data model to be shared
- OPC UA PubSub Service: specific module that provides **MQTT-based API to agents** to upload data according to OPC UA specification

It is a multi-platform, open source, software, dedicated to **Industrial IIoT** communication protocol unification

Freshly developed from Dell code-base for their own edge gateways and hosted by the Linux Foundation as an open source project on LF Edge

Like the name suggests, the EdgeX Foundry framework works at the edge computing level and allows to answer in near real-time for automation and actuation decisions

Exploits cloud-native principles like:

- loosely-coupled microservices
- platform-independence

EdgeX as a data orchestrator

EdgeX is a key enabler of digital transformation across IoT use cases and businesses in all vertical markets.

Indeed EdgeX Foundry can be programmed as a “data orchestrator”:

- it can translate protocols to an unified interface (legacy interoperability)
- can interpret those messages (called *events*) by their metadata
- redirects meaningful events toward interested endpoints

This limits the bandwidth usage - as edge computing goes mainstream, it becomes a precious resource

EdgeX development and operations management

Its support for **containerization** (e.g. Docker or snap) enables application portability and facilitates interoperable plug-and-play software applications and value add services thanks to a standard open framework.

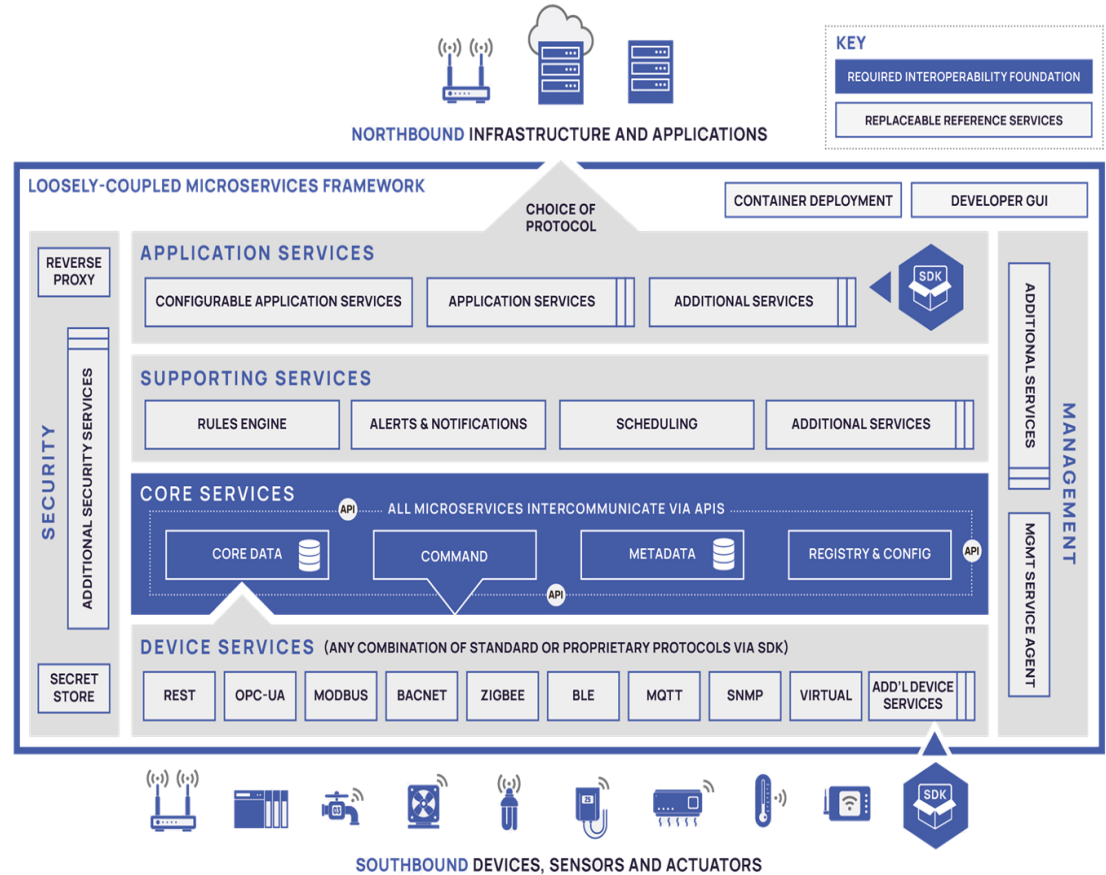
This type of virtualization allows a **multi-platform deployment** without compromising round trip response times (typical order of magnitude milliseconds) running on a gateway, a server and/or class computing equipment using standard Windows or Linux operating systems.

EdgeX can enable manufacturing equipment and other smart devices to **operate without disruption** even when Internet connectivity is down or intermittent.

Internal distributed architecture

The **Core Services** are the main components **coordinating every event and their reaction**, based on the stored knowledge.

Messages are flowing bottom-up and viceversa, making the 4 core (micro-) services act as interface among the **north-side** and **south-side**



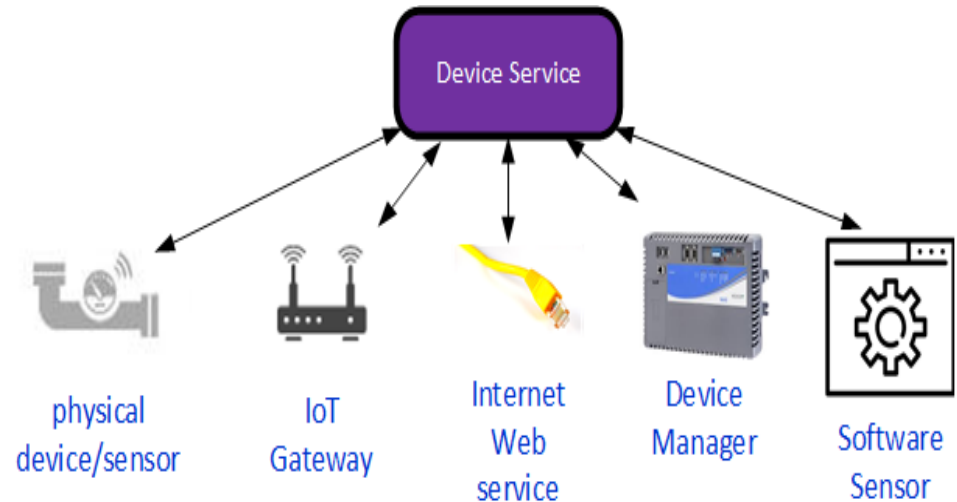
Device service: from data sources to events

Software abstraction of physical devices

pre-defined or custom Device Services (SDK provided) allows to translate device's communication protocols to an internal standard message (event) readable by EdgeX Foundry framework

This allows to register the source device and automatically add **metadata** on every gathered message

Several supported protocols, e.g., OPC-UA, ModBus, MQTT, ...

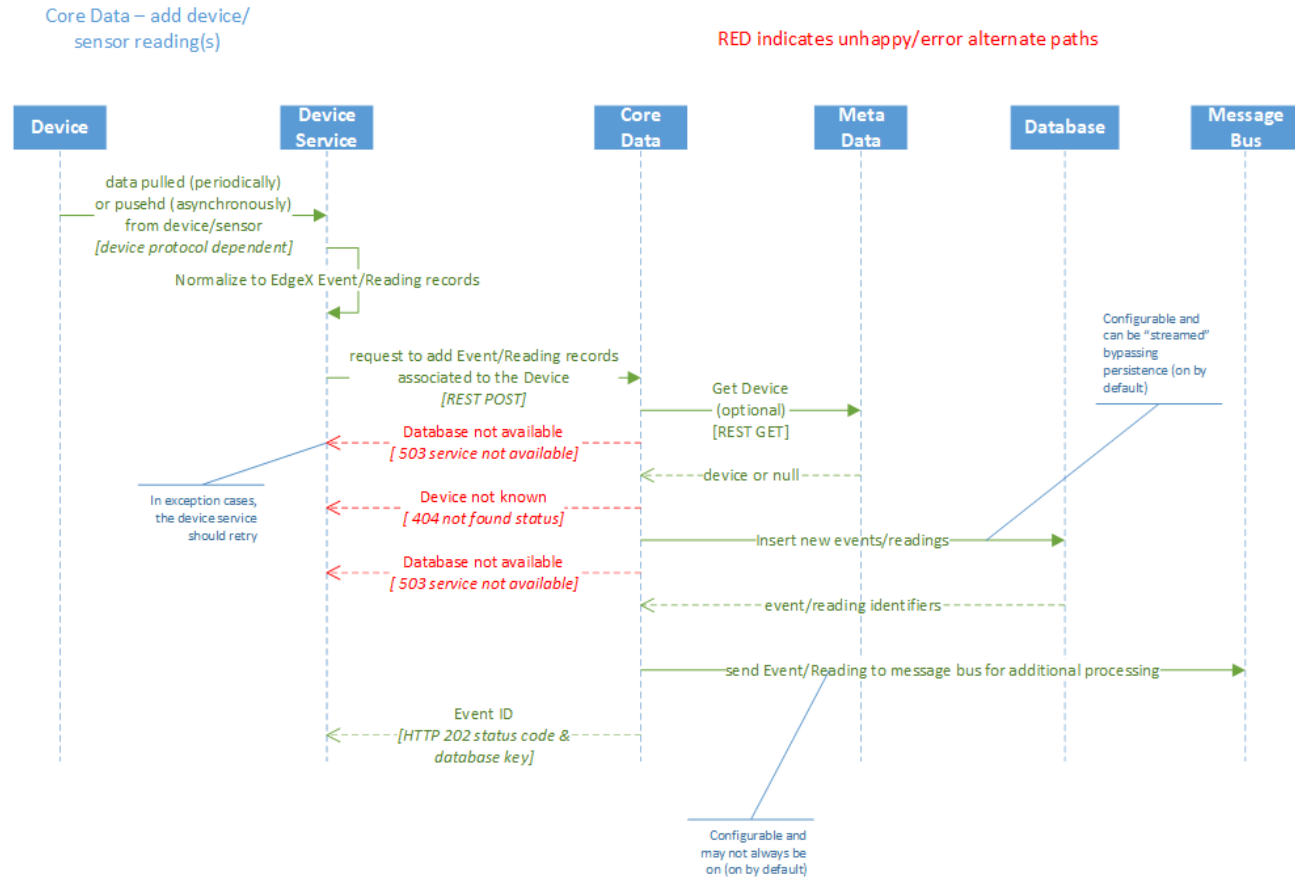


Core Data service

It stores all data sent through EdgeX framework (may be disabled for *stream-only*) with Redis

Once received, events are then published via ZeroMQ to Application Services

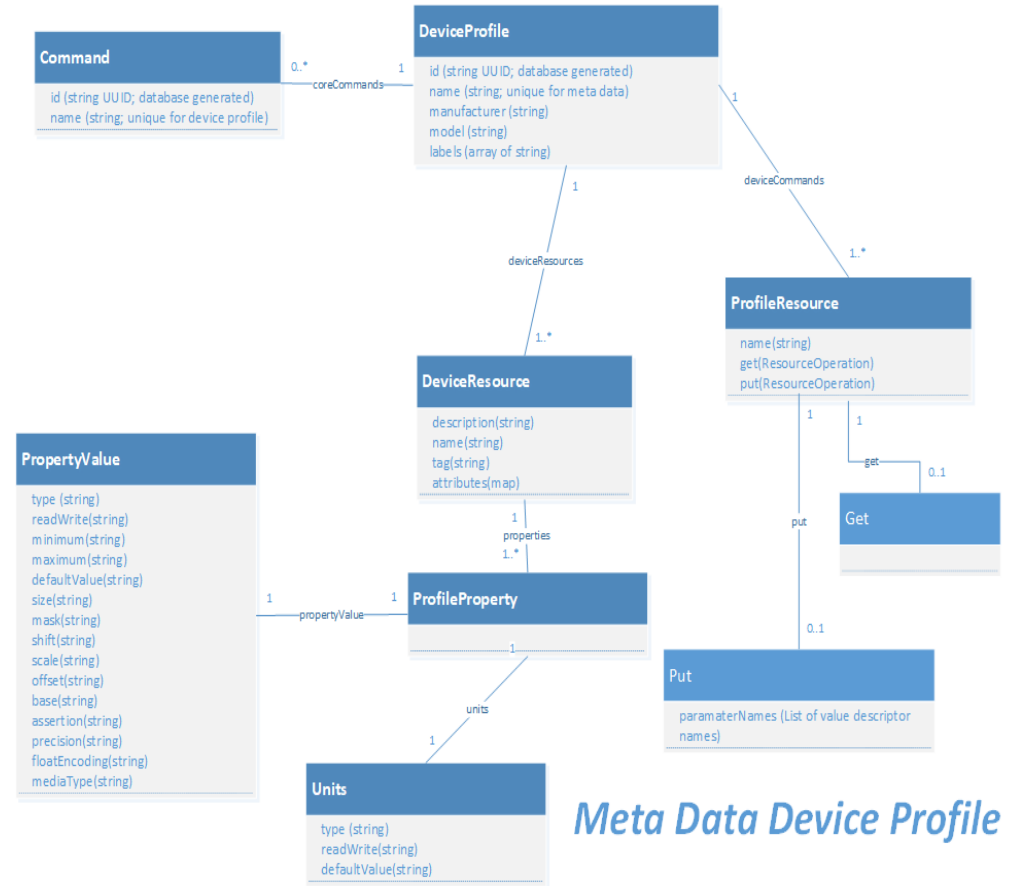
Scheduled work in charge of cleaning data, thus to free memory for new messages



Core Metadata

Stores the knowledge of every registered device and sensors, this lets the framework to know which resources are available

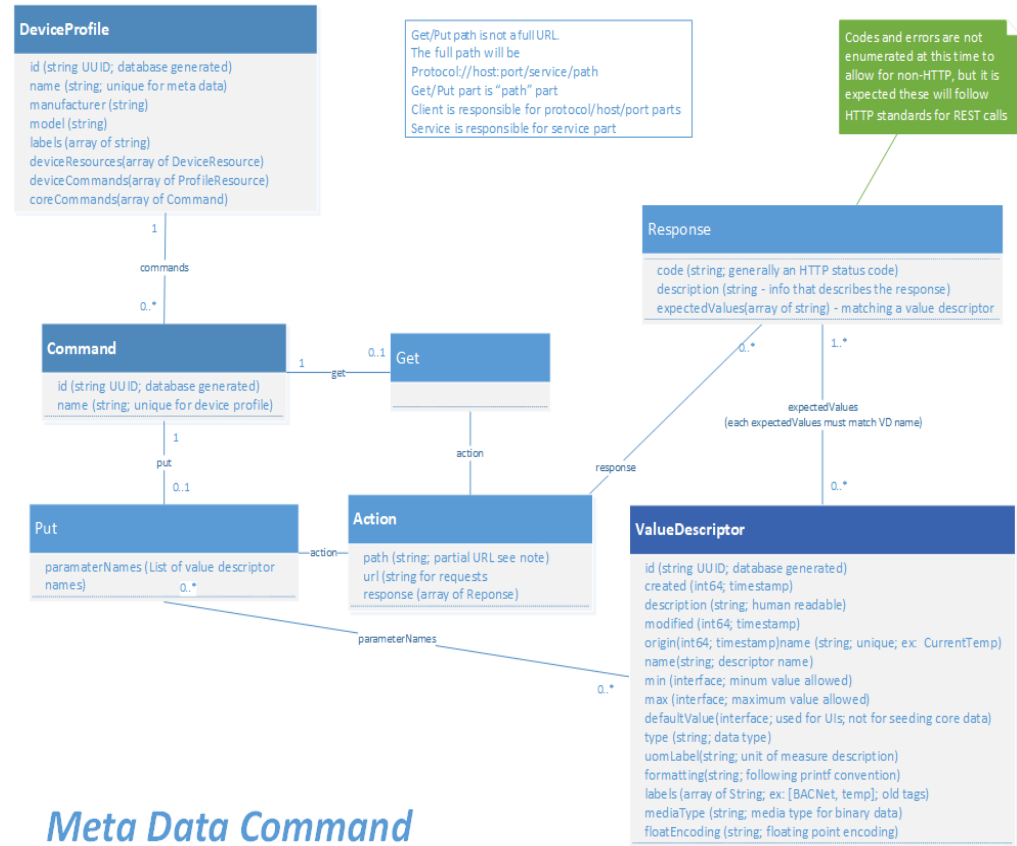
Still based on Redis, device's profile have to be provided as YAML files



Core Command

This microservice is a proxy service for action requests from the (north) exposed REST API to the Device Services, which are the only in charge to directly talk to devices

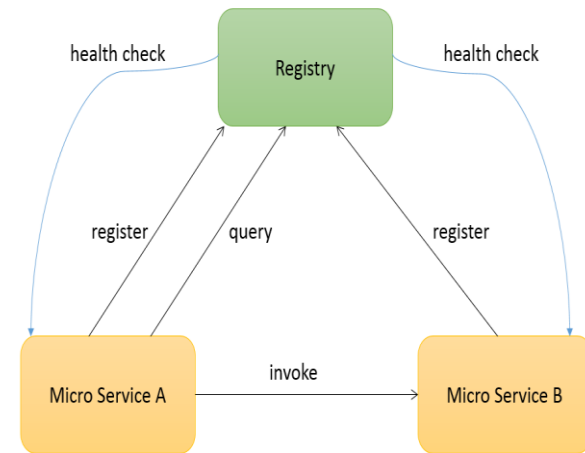
Metadata microservice provides all Core Command knowledge



Meta Data Command

Registry and Config

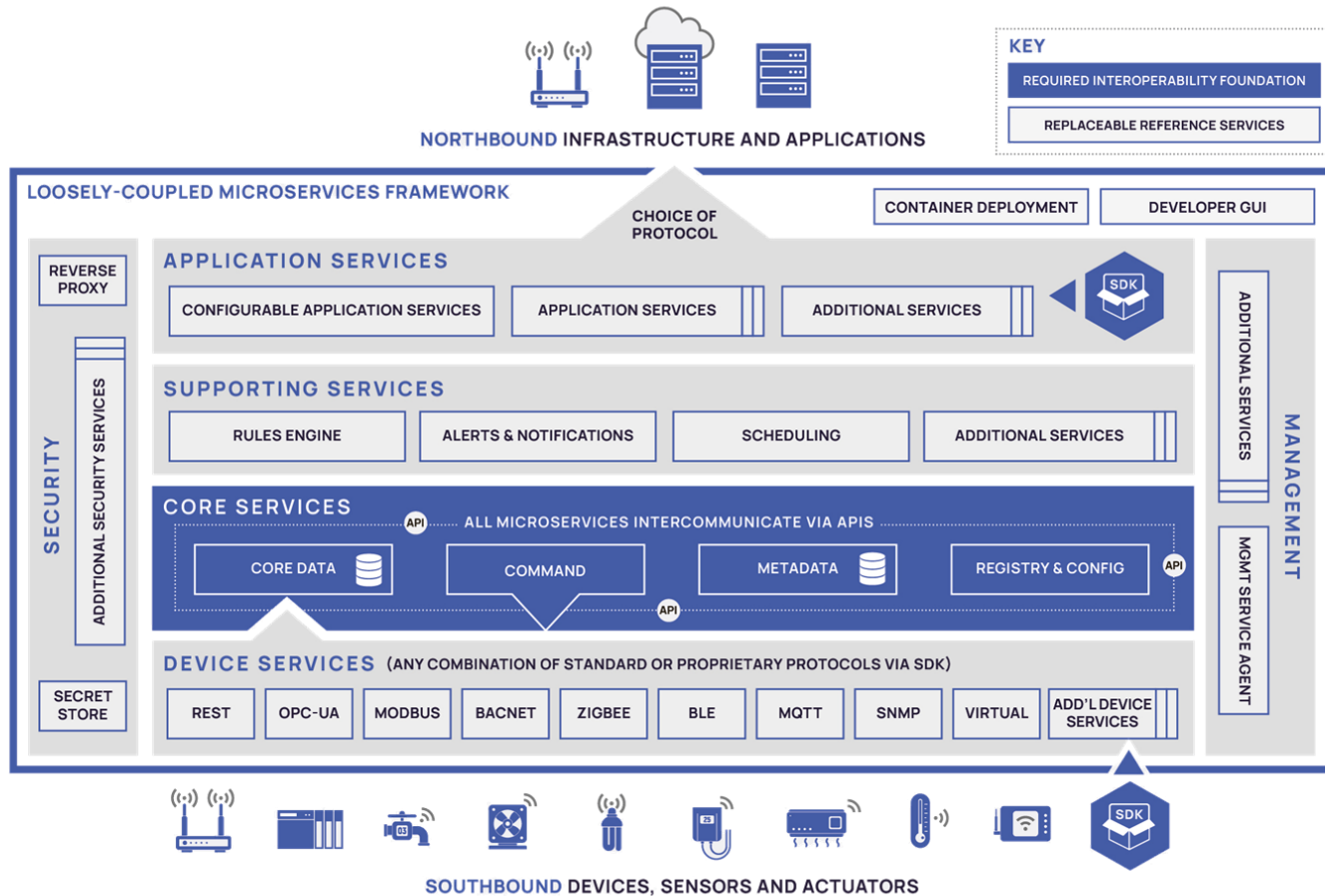
The EdgeX registry and configuration service provides other EdgeX Foundry micro services with information about **associated services** within EdgeX Foundry (such as location and status) and **configuration properties** (i.e. - a repository of initialization and operating values)



Registry:
microservices status and health monitor (Consul)

Config:
usually provided in TOML file, useful for static parameters on microservices

EdgeX Foundry wrap-up



Data-flow recap on EdgeX Foundry framework

Outline

3. Industrial IoT

- Industrial data exchange frameworks
- Industrial IoT platforms

4. **Hands-on lab on IoT platforms (in a separate slide deck)**

- Azure IoT Hub
- EdgeXFoundry