Master thesis on

Internet of Things: Sensor function virtualization and end-to-end IoT cloud platforms

Mallia-Stolidou Chrysafenia -15050

Supervisor Professor: Charalampos Skianis

University of the Aegean

Department of Information & Communication Systems Engineering

Abstract

Internet of Things (IoT) is the next “big thing” promising of a world where all everyday devices will be interconnected offering data or acting upon the fused information they collect. From health, education, smart-cities, smart-homes to tactile Internet, billions of sensors and constraint devices will work together using cloud and all other cutting-edge technologies to achieve that. The collaboration of all these aspects and technologies make IoT a huge topic. This diploma project aims to discuss IoT end-to-end cloud platforms and the sensor function virtualization role to them. The first chapter will summarize the many definitions given for the IoT in the academic community and pinpoint the fact that sensor technology and networks are a crucial subset of it. Thus, in the first chapter there is a brief report to the origin of IoT and sensor network technology. The second chapter deals with the cloud computing its definition, the main characteristics it offers and the basic cloud models. Next chapter is devoted to the key role that cloud has in the IoT applications. Chapter three examines the integration of the technologies analyzed in two previous ones, meaning what role cloud should lay in the IoT. Chapter four introduces the concept of distributed intelligence and why it is crucial to make all different use cases of IoT scenarios happen, bringing the reader close to the actual term of sensor function virtualization which is described in the next chapter. Chapter six describes the IoT gateways as the bridge between the “things” and the cloud. While chapter 7 is a small state of the art of IoT platforms. Finally, the last chapter presents a proposal for an IoT Platform as PaaS model, describes the proposed architecture, presents the opensource tools and frameworks used for the implementation part and concludes by demonstrating the basic scenario.

University of the Aegean

Department of Information & Communication Systems Engineering

Table of Contents

Table of Figures

List of Tables

University of the Aegean

Department of Information & Communication Systems Engineering

1 Internet of Things (IoT)

1.1 Definition of IoT

Our century is full of amazing accomplishments: Internet, wireless communications, Mobile Internet, LTE, 5G and so much more we witness happening every day and changing our everyday lives. Every Internet generation was believed to be the last, offering new functionalities. The first and original Internet, a virtually infinite network of computers, defined the economies of the late 20th century. However, after that Internet came the Mobile Internet, connecting billions of smart phones and laptops, and redefined entire segments of the economy in the first decade of the 21st century.

Today, we are witnessing the emergence of the Internet of Things (IoT). The IoT is a large network of physical objects, devices, vehicles, buildings and other items which are embedded with electronics, software, sensors, and network connectivity, which enables these objects to collect and exchange data. The Internet of Things allows objects to be sensed and controlled remotely across existing network infrastructure.

The Internet of Things (IoT) along with next network generation 5G is an emerging area both in academic and industry areas. It is simply the concept of connecting anything from anywhere. Every device we use for everyday activities will be “smart”. Thus, when the term IOT comes up we usually think smart cities, smart cars, smart sensors etc.

However, there are many definitions of it of various global organizations. The three most characteristic of them that IEEE tried to summarize in [1] are briefly shown below:

IEEE(Internet-of-Things, 2014) : “A network of items-each embedded with sensors- which are connected to the Internet”

ITU : “Available anywhere anytime, by anything and anyone”

NIST(National Institute of Standards and Technology, 2014) : ”Cyber physical systems(CPS) sometimes referred to as the Internet of Things involves connecting smart devices and systems in diverse sectors like connecting smart devices and systems in

diverse sectors like transportation, energy, manufacturing and healthcare in fundamentally new ways. Smart cities/Communities are increasingly adopting CPS/IOT technologies to enhance the efficiency ad sustainability of their operation and improve the quality of life"

IoT is closely coupled with sensor technology, because in most of the cases sensors are actuators part of a larger IoT network. The use of IoT devices such as laptops, smart-phones, home appliances, industrial systems, e-health devices, surveillance equipment, precision farming sensors, and other accessories connected to Internet would exceed 45 billion by 2020 [2]. That's why IoT sometimes is confused with wireless sensor networks but, WSN is just a part of IoT. Machine to Machine communication is also bind with IoT. ETSI which is a European Standard Organization actually gives a definition for M2M: "Machine to Machine (M2M) communications is the communication between two or more entities that do not necessarily need any direct human intervention. M2M services intend to automate decision and communication processes."

Those IoT sensors and actuators will probably produce large volumes of data. So, the cause for installing new network access and core devices will increase. To manage the network needs with efficiency, the network hardware resources must be virtualized.

1.2 The origin of IoT

The term IoT is officially recognized from the academic society at least since 1999, it was invented from Kevin Ashton and his AUTO-id labs in the United States. However, the idea of such a technology can be identified way back in the thoughts of great scientists like Tesla or Turing. Today this emerging sector dominates the research all over the world. It is worth mentioning that RFID together with sensor networks are the technologies that leaded to IoT.

Hence in this chapter a small reference to RFID and sensor networks will be made as they were key elements to our main subject.

1.2.1 RFID

Radio Frequency Identification (RFID) is the wireless non-contact use of radio frequency waves to transfer data. Tagging items with RFID tags allows users to automatically and uniquely identify these items. The first use of it was identifying airplanes during World War II, but this technology is improving year after year and most importantly it becomes less costly and effective. Inside the Electromagnetic Spectrum, there are three primary frequency ranges used for RFID transmissions: Low Frequency, High Frequency, and Ultra-High Frequency.

The tags contain electronically stored information. Passive tags collect energy from a nearby RFID reader's interrogating radio waves. Active tags have a local power source (such as a battery) and may operate hundreds of meters from the RFID reader. Unlike a barcode, the tag need not be within the line of sight of the reader, so it may be embedded in the tracked object. [3]

Frequency Bands	Antenna	Data & Speed	Read Range	Usage
Low Frequency (LF) 125 kHz – 134 kHz	Induction Coil on Ferrite Core, or flat many turns	- Low Read Speeds - Small Amount of Data (16 bits)	Short to Medium 3-5 feet	- Access Control - Animal Tagging - Inventory Control - Car Immobilizer
High Frequency (HF) 13.56 MHz	Induction Coil flat 3-9 turns	Medium Read Speed Small to Medium amounts of data	Short 1-3 feet	- Smart Cards - Item or Case level tagging - Proximity Cards - Vicinity Cards
Very High Frequency (VHF) 433 MHz – Active Tags	Internal Custom Design	High Read Speed Large Amounts of Data	High 1-1000 feet	- Asset Tracking - Locationing - Container Tracking
Ultra High Frequency (UHF) 860 MHz – 960 MHz	Single or Double Dipole	High Read Speed Small to Medium amounts of data	Medium 1-30 feet	- Pallet or case level tagging - DOD & Walmart Mandates
Microwave Frequency 2.45 GHz & 5.4 GHz	Single Dipole	High Read Speeds Medium Amount of Data	High 1-300 feet	- Container Rail Car - Auto Toll Roads - Pallet Level Tracking

Figure 1. Image downloaded from https://rfid4u.com/rfid-basics-resources/how-to-select-a-correct-tag-frequency/

1.2.2 Sensor Networks

Sensor Networks and IoT Wireless Sensor Network (WSN): It is a distributed and self-organized wireless network that consists of autonomous devices using sensors to observe physical or geographical conditions. In [38] it is mentioned that due to the ability to relay messages from one node to another, the area coverage of such networks may differ from a few meters to several kilometers. It is important to note that sensor network and IoT networks are not the same. At best, sensor networks are a subset of IoT ecosystem. They not only differ in deployment, but also in protocols, topologies, use cases, applications, and other technical aspects. A handful of SDN solutions for WSNs have been proposed, but they cannot be directly applied to IoT.

ZigBee (IEEE 802.15.4) [4], [5]: Specifies the physical layer and media access control for low-rate wireless personal area networks. It has been designed to run on low-power devices enabling M2M communication. It provides low-power consumption and low duty cycle to maximize battery life. ZigBee can also be used in mesh networks and supports a large number of devices over long distances with many different topologies, connected all together through multiple pathways.

WiFi (IEEE 802.11) [6]: Allows local communication between two or more devices using radio waves. It is the most used technology to connect the Internet gateway to devices. WiFi utilizes both 2.4GHz UHF and 5GHz SHF ISM radio bands. WiFi networks operate in the unlicensed 2.4 radio bands, where the access point and the mobile stations share the same channel and communicate in half duplex mode.

Bluetooth & Bluetooth Low Energy (IEEE802.15.1) [7], [8]: It is used to transfer data over short distances using 2.4 GHz ISM band and frequency hopping, and up to 3 Mbps data rate with 100m as maximum range. The technology is mostly used to connect user phones and small devices with each other.

6LoWPAN [9], [10]: It is a networking technology that combined the Internet Protocol (IPv6) with Low-power Wire-less Personal Area Networks (LoWPAN), which is one of the most suitable technologies for IoT deployment. It is a good choice for the smaller devices that are limited in processing and transmission capabilities. More on this will be described in next chapter.

LoRa / LoRaWAN (short for Long-Range) [30]: Enables very long-range communication of more than 6 miles in some areas, while consuming little power. It is a proprietary wireless technology acquired by Semtechin 2012. LoRa uses various frequency bands depending on the region of operation. In North America 915 MHz is used, and in Europe the frequency is 868 MHz. Other areas may also use 169 MHz and 433 MHz as well. LoRa refers to the underlying technology and can be directly used for peer-to-peer communications. LoRaWAN refers to the upper layer networking protocol.

5G[11]: The fifth-generation wireless is the newest iteration of cellular technology that is based on the IEEE 802.11ac wire-less networking standard in order to speed up the transmission data, reduce the latency. Both LTE and MIMO are used as a foundation in 5G network, as well as network slicing.

Below table demonstrates some other protocols too and attempts a comparison according some main characteristics.

	Power	**Speed**	**Type**	**Range**	**Mesh**	**Frequency**	**Note**
Bluetooth	Low	2-3 Mbps	PAN	50m	No	2.4 GHz	Streaming music
Bluetooth LE	Very low	1 Mbps	PAN	50m	#####	2.4 GHz	Ultralow power/small data
ZigBee	Very low	250 kbps	PAN	100m	#####	915MHz / 2.4 GHz	
Z-Wave	Very low	100 kps	PAN	150m	232	868/908 MHz	Proprietary. Up to 232 devices. Larger range than ZigBee, but slower. Less crowded RF band.
6LowPAN / Thread	Very low	Low	PAN	100m	Yes	2.4 GHz	Low power, low data
WiFi / WiFi Direct	High	100-250Mbps	LAN	100m+	No	2.4 GHz / 5 GHz	Requires access point. WiFi Direct is peer-to-peer similar to Bluetooth.
LoRa / LoRaWAN	Low	27 kbps	LPWAN	10km+	No	868 MHz / 915 MHz	Long range / low speed / low power
GSM/GPRS	Very high	Moderate	WAN	35 km	No	850 MHz / 1.9 GHz	Cellular voice/data. Being phased out.
LTE	Very high	High	WAN	Long	No	Various	Cellular highspeed data. Expensive. Overkill.
NB-IOT	Moderate	250kps	LPWAN	20km+	No	Various	Narrowband cellular technology. Also called LTE-NB. Latency = 1.5 to 10 seconds.
LTE-M	Moderate	1 Mbps	LPWAN	Long	No	Various	Lower latency than NB-IoT. Double the module cost of NB-IoT. Latency = 50 to 100 ms.

Table 1. Comparison of protocols.

Table downloaded from https://predictabledesigns.com/wireless_technologies_bluetooth_wifi_zigbee_gsm_lte_lora_nb-iot_lte-m/

2 Cloud Computing

Cloud computing is a key technology enabling IoT nowadays. The term cloud was probably chosen to indicate something that isn't clear how or where it works like the services and the applications running on the cloud. But what a complete definition of the cloud computing would be?

Despite the many definitions someone could find across the internet the one that NIST (The National Institute of Standards and Technology) seems to be the most complete one. They define the cloud computing as follows:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics**,** three service models, and four deployment models."

Actually, cloud computing is offering services as the NIST definition also mentions and there are three different kinds of them: Software as a Service or SaaS, Platform as a Service or PaaS and Infrastructure as a Service or IaaS. We are going to analyze the three service models in detail at the next chapters. The four deployment models the definition mentions are public cloud, community cloud, private cloud, and the hybrid cloud.

Public cloud which we also refer to as external cloud is when the services are offered by some provider (a third party) via the Internet and all the users can view and access them probably with some cost of course.

Private cloud is actually the hosting of applications or storage or even computation in the same for example company like a cloud in the Internet except it is used for privat purposes.

Hybrid cloud combines the characteristics of public and private cloud. A company could easily have part of their services inside their own infrastructure and part in public cloud, also it doesn't require a large investment on infrastructure.

The most essential points of cloud computing are:

- On-demand self-service. Clients can the functionality they need for example computing capabilities like network storage without asking the provider.
- Broad network access. Functionality is offered through the internet and can be used via client platforms such as mobile phones, laptops etc.
- Resource pooling. The provider has a pool of resources for offering services in many clients according to the needs. This is possible via a multi-tenant model and continues assignment of resources on demand. This naturally gives a sense of location independence in the clients, proving the term cloud as well.
- Rapid elasticity. Client has the illusion of endless capabilities as they can be offered at any quantity any time and many times with an automatic way.
- Measured service. The usage of resources in a cloud can be monitored and measured in order to control it if needed, this service suited both the provider and the client's needs. Optimizing resources is also possible.

2.1 Software As A service (SaaS)

The SaaS services are basically applications over the Internet. The user cannot tell the difference apart that he need a web browser to access the applications. Users ignore the hardware and software used and simply enjoins the functionality through an interface via the web browser. A well-known example of such a service is google docs.

2.2 Platform As A Service (PaaS)

The PaaS services are insisted of the deployment of applications or services online. Actually, this service offers a pre-built application platform so clients don't need to spend time building the underlying infrastructure for the desired applications. Usually, PaaS solutions provide an API that offers a whole set of functions for platform management and solution development. Google AppEngine, and Amazon Web Services are characteristic paradigms.

2.3 Infrastructure As A Service (IaaS)

The IaaS services are providing a computer infrastructure or its components. Components may be virtual machines, storage, networks, firewalls, load balancers etc. With IaaS services, users have direct access to the to the operating system on virtual machines, or to the management dashboard of a firewall or load balancer. Amazon Web Services is said to be the largest IaaS providers. During this essay Openstack software which provides IaaS services is going to be discussed in detail and also prove his functionality with a prototype scenario.

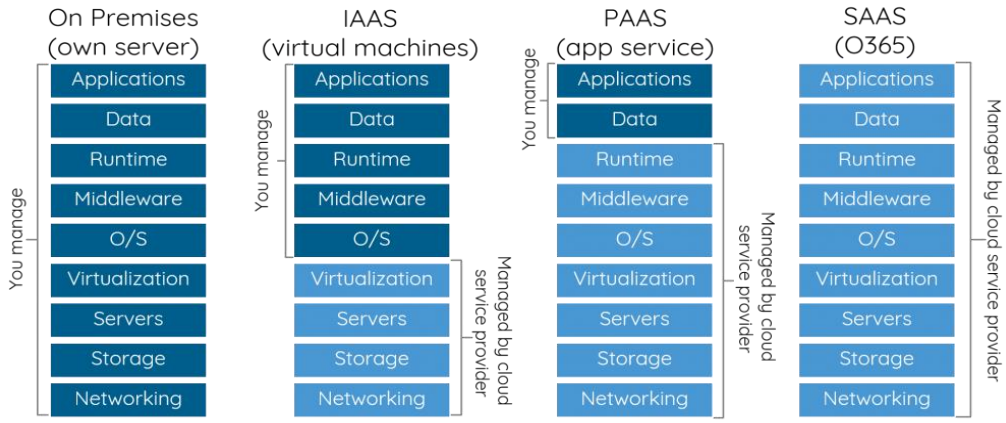

Figure 2. Cloud Service Models

3 The role of Cloud Computing in the Internet of Things (IoT)

The integration of cloud computing and IoT is actually already happening and is very crucial in most of the cases as these two different technologies have complementary characteristics. Below picture taken from a scientific article in "Future Generation Computer Systems" [13] is summarizing this fact.

	Internet of Things	Cloud Computing
Displacement	Pervasive	Centralized
Reachability	Limited	Ubiquitous
Components	Real world "things"	Virtual resources
Computational capabilities	Limited	Virtually unlimited
Role of the Internet	Point of convergence	Means for delivering services
Big data	Source	Means to manage

Figure 3. Integration of IoT and cloud computing

First of all, in terms of storage IoT sensors as explained have extremely limited to offer, on the other hand cloud can offer virtually unlimited capacity. Thus, cloud is the ideal solution to massive data produced from the sensors and cloud can use IoT as the source of information and Big Data. Another fact is that cloud will hide all the intermediate functionality and complexity between sensors and applications. In terms of communication cloud will offer the ability to control and monitor remotely anything from anywhere providing the networks needed to achieve that. Processing power is another weakness of IoT so transmitting the information in nodes rely on the cloud is a solution. Scalability offered by the proper infrastructure is also an advantage of the cloud.

3.1 Use cases of IoT and cloud integration

Health: Healthcare services will significantly improve of the automation that cloud and IoT will provide. Devices measuring all kind of vital metrics for health could be transmitted to cloud and monitored in a daily or even hourly basis to offer the health status of the patient and act accordingly without the person having to stay inside a hospital. Electronic health records could be available for any hospital or medical station through the cloud.

Looking even more forward today medical expertise is bound to the location of the physician but tomorrow using advanced tele-diagnostic tools, it could be available anywhere, anytime allowing remote physical examination even by palpation (examination by touch). The physician will be able to command the motion of a tele-robot at the patient's location and receive not only audio-visual information but also critical haptic feedback. Of course, privacy and security of these data is crucial and should be examined how is going to enabled.

Smart Cities: Urban IoT is a whole chapter of research itself as it has many applications like a Smart Governance, Smart Mobility, Smart Utilities, Smart Buildings, and Smart Environment according to European Smart Cities Project [39].
Air quality measurements enabled from the integration of sensors like pollution sensors and resulting to publicly available data for all the citizens is a first application. Data for air quality could be also used from other health care and smart devices. For example, the smart watch of a jogger could be connected to the infrastructure of the city warning him for the air quality of the park she/he chose to run. Noise is also thought in modern cities a kind of pollution. There are even laws try to minimize it. So sound detectors and relevant software, or integration of surveillance mechanisms can be used to enable a monitoring mechanism inside an urban IoT. Traffic monitoring may be realized by using the sensing capabilities and GPS installed on modern vehicles, and also adopting a combination of air quality and acoustic sensors along a given road. This information is of great importance for city authorities and citizens: for the former to discipline traffic and to send officers where needed and for the latter to plan in advance the route to reach the office or to better schedule a shopping trip to the city center [13]. Smart cars and sensors deployed inside them could also benefit this service. Smart lighting on roads depending on the traffic and the weather is also a useful service that an urban IoT could realize. Moreover, energy consumption of a city in general could be monitored and handled from the integration of all of the above sensors. Waste management is a primary issue in many modern cities, due to both the cost of the service and the problem of the storage of garbage in landfills. A deeper penetration of ICT solutions in this domain, however, may result insignificant savings and economical and ecological advantages. For instance, the use of intelligent waste containers, which detect the level of load and allow for an optimization of the collector trucks route, can reduce the cost of waste collection and improve the quality of recycling [14].3To realize such a smart waste management service, the IoT shall connect the end devices, i.e. intelligent waste containers, to a control center where an optimization software processes the data and determines the optimal

management. The smart city of Padova in Italy is a characteristic example where all of these innovative services become reality.

Smart homes: Of course, if we can discuss regarding smart cities, smart homes is a natural subset. So, smart lighting, heating sensors etc could also be enabled inside houses. However, as we are examining the concept that any device acts like a node in the IoT the services don't stop there but spread in all everyday activities. For example, a smart fridge could monitor food resources and trigger the family members with a notice in their smartphones when resources are running low.

Environmental monitoring: Sensors deployed in certain areas of environmental interest could help scientists monitor the pollution or other metrics of these areas through IoT nodes offering these data and relevant software processing them and demonstrating the results. So, remotely monitoring and acting on inaccessible places of the world could be feasible after installing the appropriate IoT nodes. Prevention of serious physical catastrophes as fire, floods, earthquakes could be possible applications.

Agriculture: Applications help modern farmers on management of their activities are also an important service of integration of IoT and cloud. Precision farming enables farmers to increase production (yield), lower their operational costs and economize their applications of chemicals and fertilizers. Precision Conservation Management (PCM) focuses on improving crop yield through the analysis of real-time data from a variety of environmental sensors and other data sources located in commercial crop fields or throughout the enterprise. The five core components or processes of precision farming are: measuring variability, analyzing variability, decision-making, differential actions, assessment of results and measuring [15].

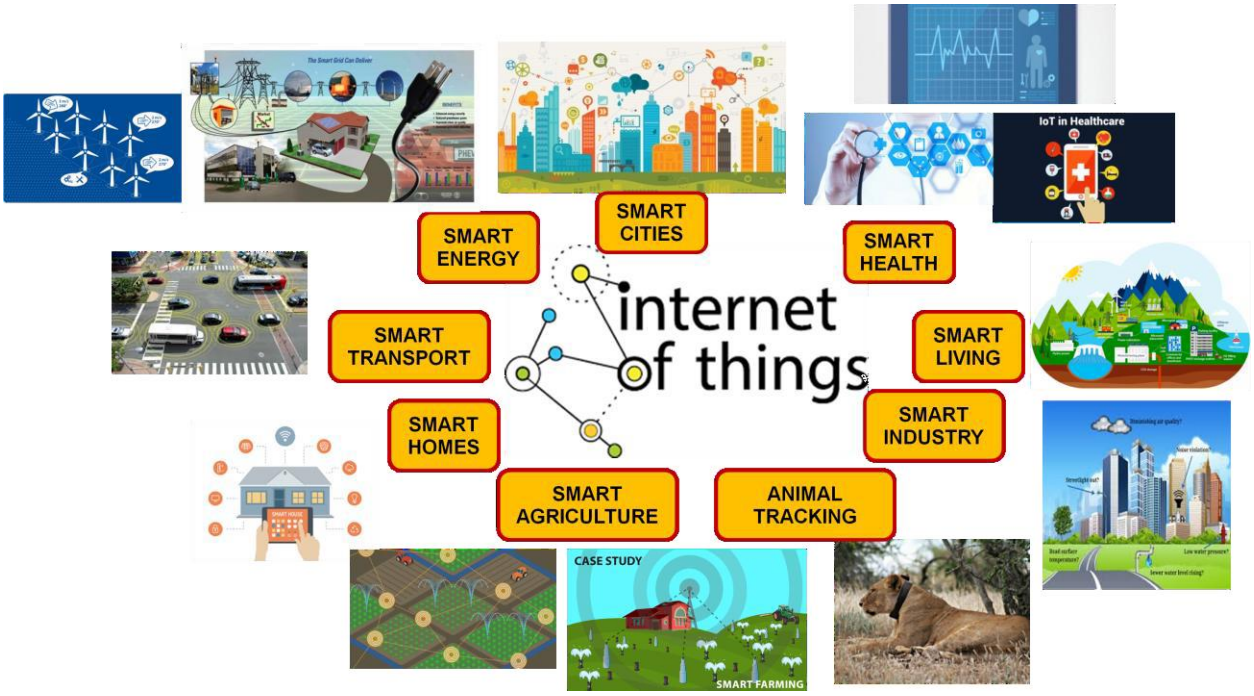

Figure 4. Smart City. Image from "International Journal of Engineering Science and "Computing, May 2016

4 Why distributed intelligence is needed in IoT

It is clear by now from all academic and technical definitions for the huge scale of IoT that will include billions of devices/sensors (approximately 200 billion till 2020) so eventually will need to be enhanced with processing and communication functionality. In most of the cases the ability for that is very limited as these devices often run in batteries or must be extremely low cost. A regular term for these is “constrained devices” many of them belonging to Class 1 (approximately 10KiB RAM and 100KiB ROM) as defined by the IETF working group. Various low data rate and low power communication technologies have been introduced as already seen in previous chapter, such as IEEE 802.15.4. Moreover, the need to interconnect these devices and also connect them to the Internet was generated.

The Internet Engineering Task Force (IETF) is the premier Internet standards body, developing open standards through open processes. The IETF is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet[https://www.ietf.org/about/]. A great achievement of this organization was the actual integration of constrained devices in the internet and more specifically in the IPv6 with IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs).

RFC 4919 describes 6LoWPAN as follows: “A LoWPAN is a simple low-cost communication network that allows wireless connectivity in applications with limited power and relaxed throughput requirements. A LoWPAN typically includes devices that work together to connect the physical environment to real-world applications, e.g., wireless sensors. LoWPANs conform to the IEEE 802.15.4-2003 standard [40].

Some of the characteristics of LoWPANs are:

1. Small packet size. Given that the maximum physical layer packet is 127 bytes, the resulting maximum frame size at the media access control layer is 102 octets. Link-layer security imposes further overhead, which in the maximum case (21 octets of overhead in the AES-CCM-128 case, versus 9 and 13 for AES-CCM-32 and AES-CCM-64, respectively), leaves 81 octets for data packets.

2. Support for both 16-bit short or IEEE 64-bit extended media access control addresses.

3. Low bandwidth. Data rates of 250 kbps, 40 kbps, and 20 kbps for each of the currently defined physical layers (2.4 GHz, 915 MHz and 868 MHz, respectively).

4. Topologies include star and mesh operation.

5. Low power. Typically, some or all devices are battery operated.

6. Low cost. These devices are typically associated with sensors, switches, etc. This drives some of the other characteristics such as low processing, low memory, etc. Numerical values for "low" elided on purpose since costs tend to change over time.

7. Large number of devices expected to be deployed during the lifetime of the technology. This number is expected to dwarf the number of deployed personal computers, for example

8. Location of the devices is typically not predefined, as they tend to be deployed in an ad-hoc fashion. Furthermore, sometimes the location of these devices may not be easily accessible. Additionally, these devices may move to new locations.

9. Devices within LoWPANs tend to be unreliable due to variety of reasons: uncertain radio connectivity, battery drain, device lockups, physical tampering, etc.

10. In many environments, devices connected to a LoWPAN may sleep for long periods of time in order to conserve energy, and are unable to communicate during these sleep periods.

The next step for IETF was the integration of constraint devices in web services so they introduced and standardized in RFC 7252 the Constrained Application Protocol (CoAP).

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while constrained networks such as IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) often have high packet error rates and a typical throughput of 10s of kbit/s. The protocol is designed for machine- to-machine (M2M) applications such as smart energy and building automation. CoAP provides a request/response interaction model between

application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead, and simplicity for constrained environments. [41]

With these technologies, it has become possible to interconnect tiny objects or networks of such objects with the IPv6 Internet and to build applications that interact with them using embedded web service technology, bringing us one step closer to the realization of the Internet of Things. From the above description, it becomes clear that it has been a challenge to fit an Internet-compatible protocol stack on devices with very limited capabilities. At some point it will become technically infeasible to put additional intelligence on the devices themselves due to their constraints.

In addition, as every IoT application may have different requirements and devices may be very heterogeneous, there will be no single recipe on how to optimally distribute the functionality. Future IoT systems should be able to cope with these requirements, e.g. by supporting distributed processing that puts the intelligence wherever it is most needed. Finally, there are certain challenges of an IoT ecosystem as described in the research such as the need to work offline for some time, meaning that the IoT application should not entirely rely on a central cloud to process sensor data, the limitation of latency, scalability of the system or heterogeneous data types from the different sensors.
Consequently, depending on the type of IoT application, the processing functionality may reside either far away from the constrained devices (in the cloud) or must reside nearby in order to meet certain performance requirements. **This is what we define as distributed intelligence**. It is the cooperation between devices, intermediate communication infrastructures (local networks, access networks, global networks) and/or cloud systems in order to optimally support IoT communication and IoT applications. Starting from the application requirements, user needs and policies, intelligence is optimally distributed and activated in order to operate functions such as processing, security, QoS and to configure the communication infrastructure. Through distributed intelligence, the right communication and processing functionality will be available at the right place and at the right time. [12]

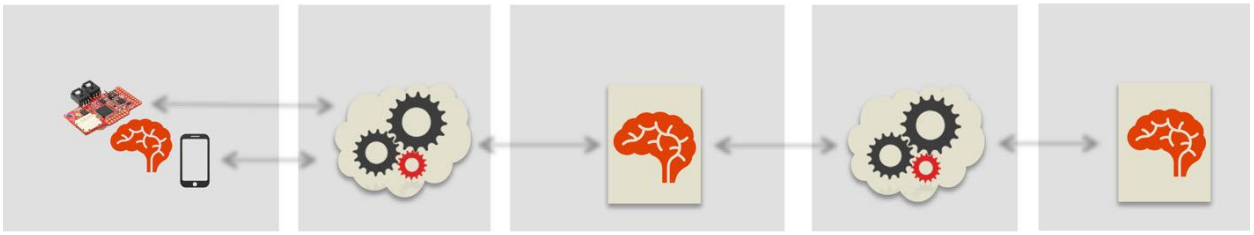

Figure 5. Distributed intelligence

5 Sensor function virtualization in the Internet of Things (IoT)

The concept analyzed in the previous chapter described why we need distributed intelligence in IoT applications and why the scenario with a central cloud for processing the data coming from the sensors isn't the best solution. The way to enable this distributed intelligence is actually what we call sensor function virtualization where sensor gets a wider definition and includes actuators, devices etc.

Actual sensor function virtualization of course includes the virtualization of constraint devices/sensors. It acts like a software abstraction where the user can run applications, use interfaces or ask for services without actual knowing the exact physical sensors. Of course, this could lead to conflicting scenarios when different applications will request resources eg reading the data from the same light sensor. This should be handled from software handling the resources available and enabling some queuing algorithm to manage all the possible use cases. The idea is rather familiar as this is already realized with the common used virtual machines where a computing machine and its resources are shared and used on demand. Furthermore, the combination of the data various sensors providing could be used to provide some metrics to the user not aware of the existence of different physical devices. For example, a weather forecast could combine the data from humidity sensors, temperature sensors, light sensors etc but still be seen from the user point of view as a single "forecast predicter".

The above scenario brings to light another important aspect of sensor function virtualization, scalability. When billions of devices, really all everyday objects will be part of the Internet of Things any solution implemented should be able to increase on demand. This also proves how crucial cloud technology is in the future of IoT. As mentioned a part of the sensor function virtualization will run on cloud infrastructure, so it will profit from the elasticity provided by cloud technology. Increased needs will automatically allocate more resources. Elasticity is a huge benefit for scalability.

Heterogenous of the devices consisting the IoT should also be addressed in sensor function virtualization. A solution would be the existence of some kind of gateway before we get to the cloud part and services.

6 IoT gateways

Previous chapters regarding distributed intelligence needed in IoT applications and sensor function virtualization concepts made clear of the importance of the missing link between IoT nodes/constrained devices and the Cloud or a Platform on the cloud. This interface between them is the IoT gateway.

An IoT gateway is the first component that directly communicates with the physical sensors. Having discussed the wide variety of the protocols the devices use to communicate and their heterogenous nature we identify the need of some kind of device manager inside the IoT gateway. This device manager can support from one, two or all the available protocols of "things". Following, all the raw data transmitted to the gateway should be stored, probably pre-processed and tagged with some algorithm eg a timestamp and device id. Expect from all the above as discussed in previous chapters ideal solutions for IoT demand some functionality to be moved near to device. This could "save" the application from breaking down every time there is a network availability towards the cloud or save time and be more efficient performing some tasks on the gateway.

In [31] the authors describe a smart gateway which is responsible for protocol conversion and data fusion of different sensor data. The server is the support of the entire system, which is responsible for the processing of historical data and visual display. The user cards through a generic interface to achieve different types of gateway functions. Specific implementation of Smart IoT Gateway is shown in

While in [32] the authors think that the main challenge on creating an IoT gateway is the lack of standards, being that each sensor node can communicate with a different protocol that is not compatible for others. This makes the development of a general-purpose gateway a complicated task, which explains why it is common to find gateways developed for specific applications. Nevertheless, all have the same key requirements: low-cost hardware, easy implementation and extensibility and an application layer support.

In literature you can find several solutions from complicated and expensive ones to IoT gateways implemented using low-cost hardware devices, such as Arduino and Raspberry Pi.

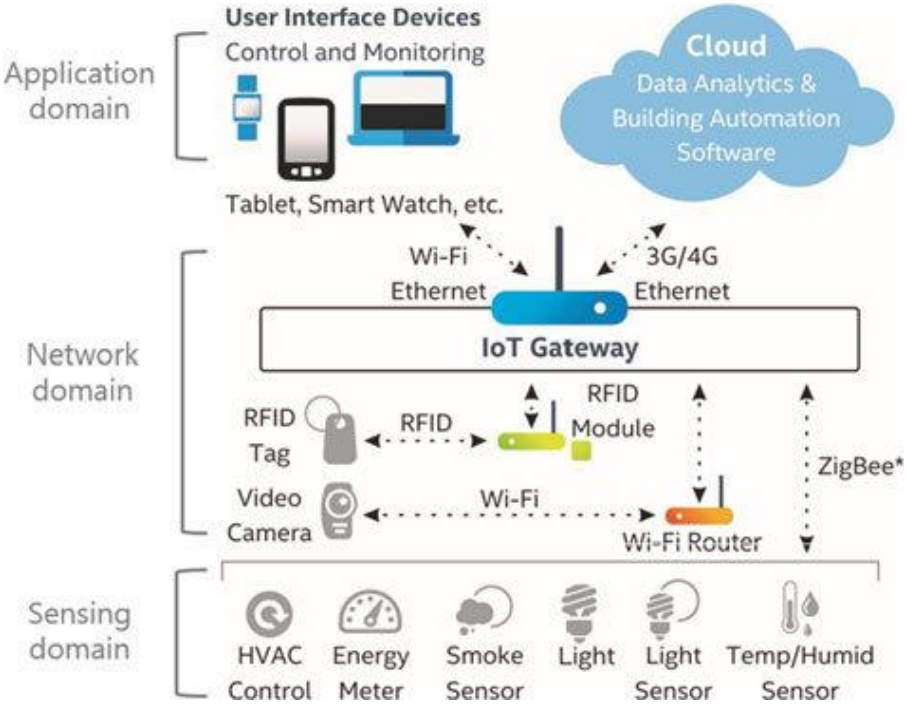

Figure 6. Image taken from " A Large-Scale Autonomic IoT Gateway"

Transfer of data from the gateway to the cloud is an important factor as it is also critical for the cost of the whole IoT application. In [33] the authors summarized that usually gateways are connected to the Internet using GPS, WiFi, or Ethernet. Some gateways can also work in both GPS and WiFi modes (for example, gateways mounted in moving vehicles). In general, non-GPS connectivity is preferred to send data, as it doesn't require a subscription to a paid mobile plan. Some gateways will be constantly connected to inexpensive local networks, but those using GPS connectivity should be very conservative in terms of what data they send to the data center. The gateway should apply service logic against the data it collects to understand which messages should be sent over expensive GPS networks, and which data can be cached on the device for deferred offline processing. The messages collected by the gateway from the IoT device (sensors and actuators) are usually small in size. The current value of the temperature measured by the sensor is just a decimal number. GPS coordinates are two decimal numbers, which represent longitude and latitude. This is an important thing to remember: the gateway operates on many small messages.

From this brief chapter on IoT gateways it is obvious that still there isn't a unified proposal for the ideal gateway and also that this bridge between nodes and the cloud hides complexity. Of course, the more complex and wider is the application of the IoT so and the gateway solution a developer or a vendor should choose. Nevertheless, the research for a smart fully automated, scalable IoT gateway is continued as the variety of scientific papers prove.

7 End to end IoT platforms (a state of the art)

The market of the IoT platforms is expanding. The solutions offered are hundreds and the comparison between them seems difficult as each of them have specific characteristics bound to cloud infrastructure or devices and different perception of IoT itself. This alone as a fact is proof that we are passed the beginning but sure miles way from the maturity of these products. The more specific a platform is the more it proves that some general issues and principal in IoT technology isn't standardized yet and maybe never will be.

In general, an IoT cloud platform must deal with the three main parts of an IoT system: the IoT nodes, the gateway and the cloud. The IoT nodes are actually the devices/sensors producing the data. As the nodes may not communicate directly to the Internet a gateway like it was described in the previous chapter should exist to collect all the data transmitting with all kind of different protocols. Last is the cloud where the storing, processing and analyzing the data is happening where we theoretically have virtually unlimited resources.

End to end term in cloud platforms usually mean a king of a PaaS model rather than SaaS. This means that most solutions provide a platform that can cooperate with gateways and receive data from many different sensors and their protocols and making their data available in the cloud. So, the various applications used by the actual user are left for the developers of the needed services. That's why PaaS is the closer to the model that most of the cloud platforms can be categorized in.

Of course, the bigger companies dealing with cloud like Google, Microsoft, AWS and IBM dominate also the IoT cloud platform market at the time so a small reference to four these solutions will be made.

Google Cloud IoT: Google Cloud IoT is a complete set of tools to connect, process, store, and analyze data both at the edge and in the cloud. The platform consists of scalable, fully-managed cloud services; an integrated software stack for edge/on-premises computing with machine learning capabilities [16]. It actually takes advantage of all google powerful modules and has to demonstrate IoT use cases predictive maintenance for industry, real-time asset tracking, logistics and supply-management, smart cities and buildings.

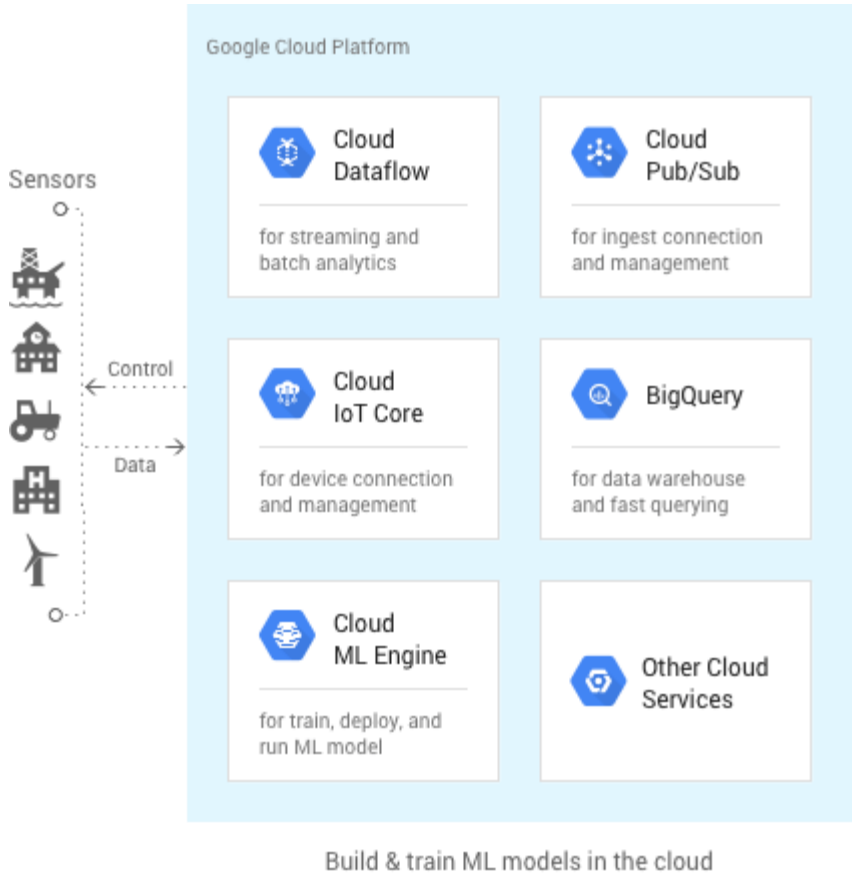

Figure 7. Google Cloud Platform

AWS IoT: It is the solution provided by Amazon. As google it combines all AWS modules and services to perform the best result. AWS IoT provides device software, control services, and data services. Device software enables you to securely connect devices, gather data, and take intelligent actions locally, even when Internet connectivity is not available. Control services allow you to control, manage, and secure large and diverse device fleets. Data services help you extract value from IoT data [17]. It supports HTTP, WebSockets, and MQTT. The AWS IoT Device SDK supports C, JavaScript, and Arduino, and includes the client libraries, the developer guide, and the porting guide for

manufacturers. The Device Gateway serves as the entry point for IoT devices connecting to AWS. The Device Gateway manages all active device connections and implements semantics for multiple protocols to ensure that devices are able to securely and efficiently communicate with AWS IoT Core. Currently the Device Gateway supports the MQTT, WebSockets, and HTTP 1.1 protocols. For devices that connect using MQTT or WebSockets the Device Gateway will maintain long lived, bidirectional connections,enabling these devices to send and receive messages at any time with low latency. The Device Gateway is fully managed and scales automatically to support over a billion devices.

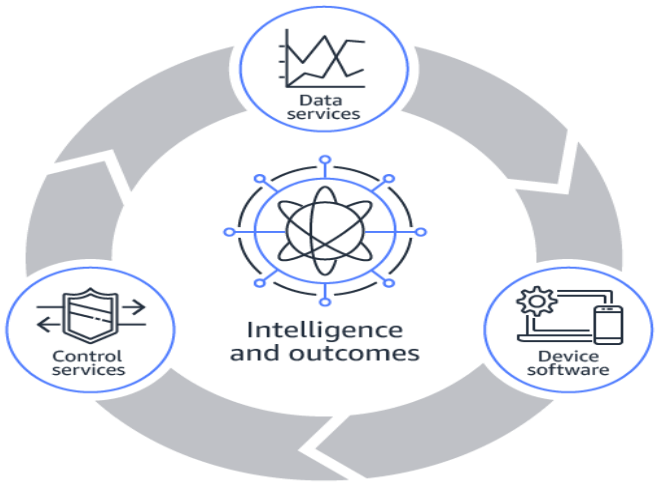

Figure 8. AWS IoT

Microsoft Azure IoT Suite: The open source code base was designed on fully tested architecture and is available on GitHub but of course for ready solutions there are pricing modules. It supports HTTP, Advanced Message Queuing Protocol (AMQP), and MQ Telemetry Transport (MQTT). You can use this platform to collect and analyze real-time device data to trigger automatic alerts and actions—including performing remote diagnostics and automatically initiating maintenance requests. Use the remote monitoring dashboard to view telemetry from your connected devices, provision new devices, and upgrade firmware on your connected devices [18]. Or you can use the device simulator for your tests.

IBM Watson IoT: It offers machine learning, automate data processing and rank data based on learned priorities. Also, there is Raspberry Pi Support.

A key service in IBM's IoT Platform, IoT Real-Time Insights consumes data from IoT devices and provides awareness and understanding from that data through real-time contextualization and visualization. It enables organizations to monitor equipment and operations to understand and respond to emerging conditions to improve responsiveness, equipment availability, and overall efficiency. IoT Real-Time Insights can be used for several common monitoring patterns [19]:

- Monitor devices to understand current conditions and status, usage patterns, availability, and performance
- Monitor operational processes to track progress towards milestones and business outcomes
- Monitor environmental conditions to identify security issues and unsafe or undesirable situations
- And when conditions warrant it, IoT Real-Time Insights allows you to automate a response to those conditions

Following apart from these four dominating providers there are as mentioned hundreds of other solutions offered and among them serious open source efforts. In the second part of this chapter we are going to make a reference to the top open source IoT platforms:

Kaa Platform: Kaa is an IoT middleware technology applicable for any scale of enterprise IoT development. It supports lightweight IoT protocols for device connection, such as MQTT and CoAP but may support any IoT protocol. The platform allows building applications that function over any type of network connection, either persistent or intermittent. User may choose one of the existing transport protocol implementations that come with Kaa or create custom-tailored transports and plug them into your system. MQTT is the default protocol used by Kaa. Kaa provides a register of digital twins, which represent things, devices, and other entities managed by the platform. Kaa also allows you to store device attributes, which provide more detailed information about any characteristic of the device. Examples of such attributes could be serial number, MAC address, location, software version, etc. In addition to simple data types, attributes can

contain more complex, structured objects, such as a list of connected peripherals and their properties. Kaa allows you to collect both structured and unstructured data. It can be of primitive types, such as plain numbers or text, or compound, such as key-value maps, arrays, or nested objects. The data visualization component of Kaa comprises a rich set of widgets, such as gauges, charts, maps, tables, etc. You can use these widgets to visualize different types of data, whether telemetry, statistics, geolocation, metadata, filters, software updates, or other—both historical and current. All widgets are configurable and allow you to change their data sources as well as visual representation. To address special use cases, Kaa visualization component allows you to easily plug in custom widgets. Besides data visualization, widgets allow you to interact with devices by sending commands, changing configuration and metadata, etc. Command execution is the Kaa platform feature that allows you to deliver messages with the arbitrary payload to connected devices, execute commands, and receive near-real time responses. For example, you can remotely check current temperature on a home thermostat, point a security camera to a specific area, open a vehicle trunk, and so on [20].

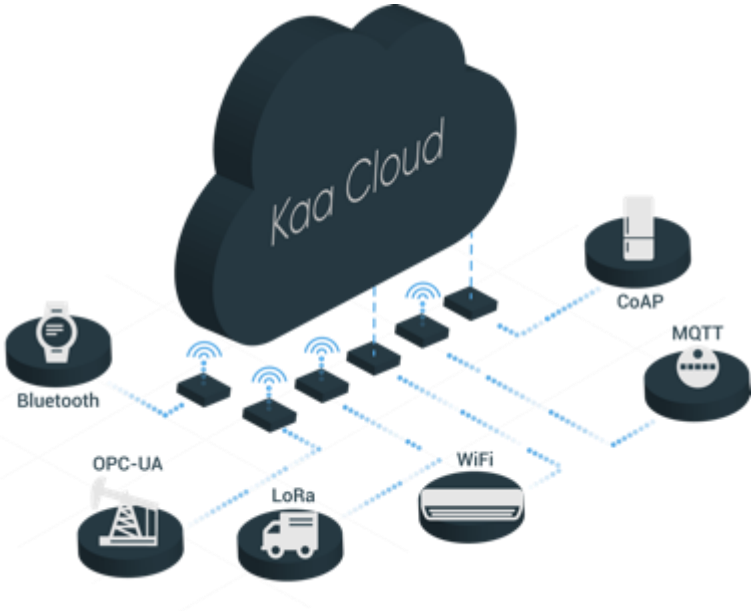

Figure 9. Kaa Platform

SiteWhere: It is another IoT cloud platform. Its infrastructure and microservices are deployed on Kubernetes, allowing for deployment on-premise or almost any cloud provider. Highly-available

configurations of Apache Kafka, Zookeeper, and Hashicorp Consul provide infrastructure. Each microservice scales independently and integrates automatically [21]. It connects devices using MQTT, AMQP, STOMP and other protocols, and devices can be added through self-registration, REST services or in batches. It can also control large numbers of devices using batch command operations [22].

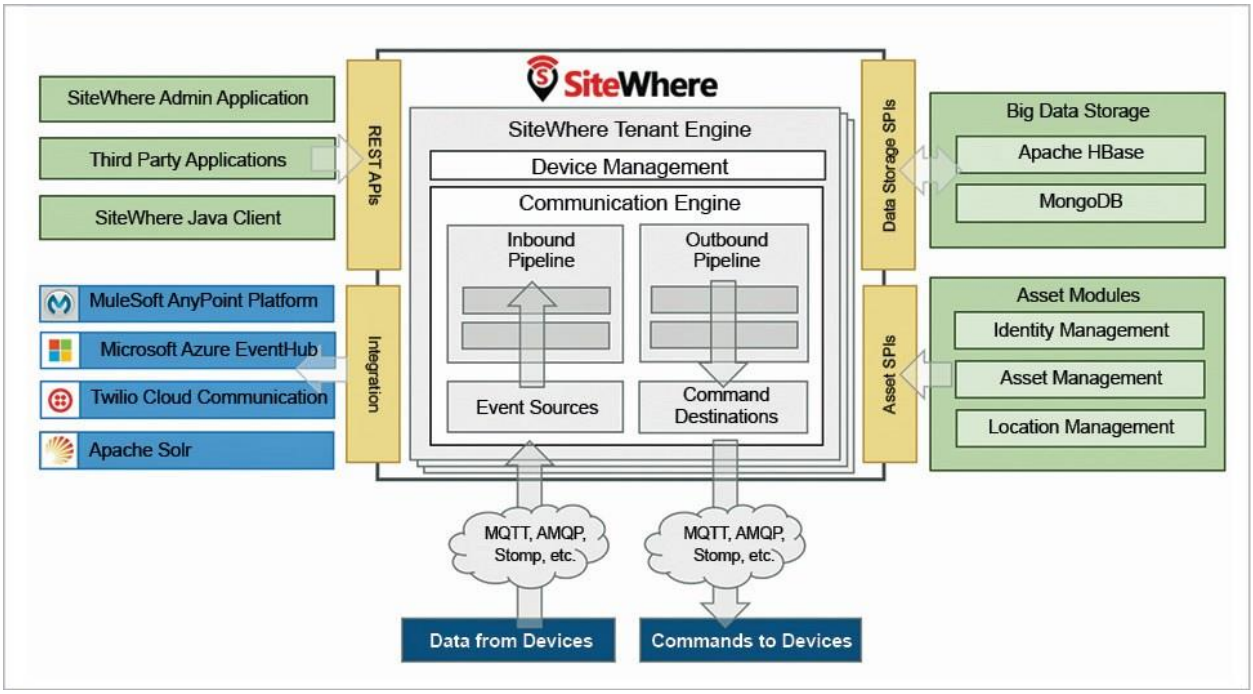

Figure 10. SiteWhere Platform

ThingSpeak: ThingSpeak is an Internet of Things (IoT) platform that lets you analyze and visualize the data in MATLAB without buying a license from Mathworks. IT allows you to collect and store sensor data in the cloud and develop IoT applications. The ThingSpeak is mostly focused on sensor logging, location tracking, triggers and alerts, and analysis [23].

DeviceHive: Another open source IoT Data Platform with the wide range of integration options. With the docker compose and kubernetes deployment options, you can go with private, public or hybrid cloud and scale from a single virtual machine, to enterprise grade cluster. DeviceHive employs the best software design practices, introducing container-based service oriented architecture approach managed and orchestrated by Kubernetes. User can connect any device via REST API, WebSockets or MQTT. The DeviceHive team

supports libraries written in various programming languages, including Android and iOS libraries which make the platform device-agnostic [24].

Mainflux: Mainflux is performant and secure open-source IoT platform with the complete full-scale capabilities for development of Internet of Things solutions, IoT applications and smart connected products. Built as a set of microservices containerized by Docker and orchestrated with Kubernetes, Mainflux IoT platform serves as a software infrastructure and middleware which provides [25]:

- Device management
- Data aggregation and data management
- Connectivity and message routing
- Event management
- Core analytics
- User Interface
- Application enablement

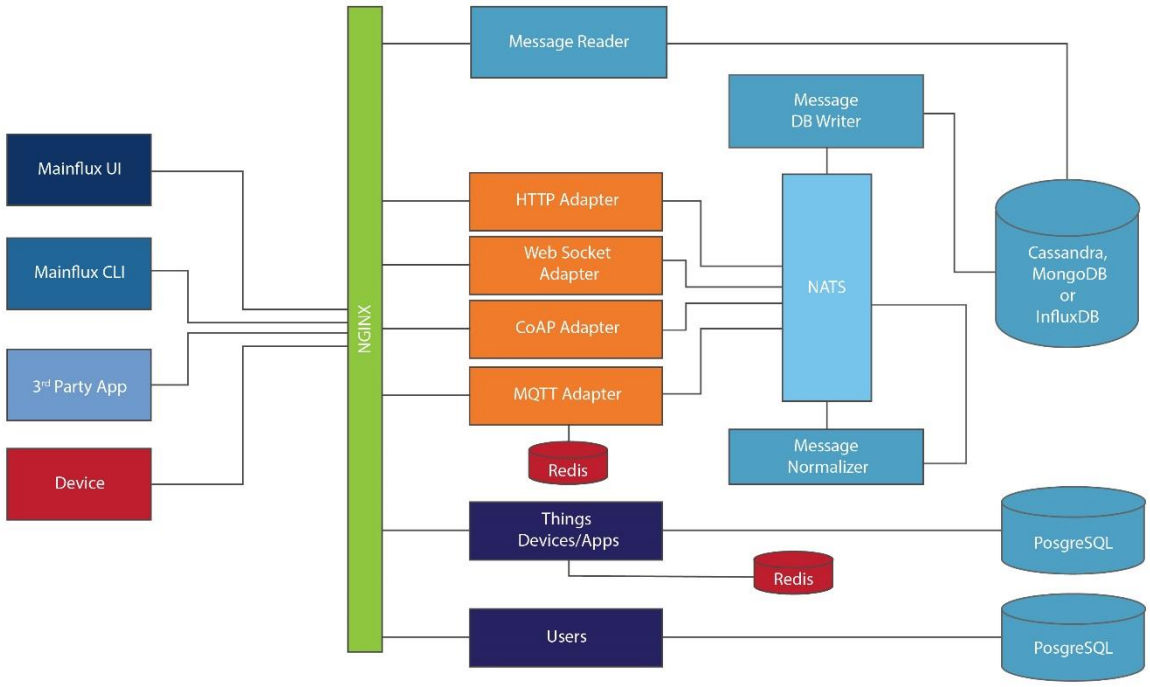

Figure 11. Mainflux Platform

Thinger.io: Thinger.io platform is an Open Source platform for the Internet of Things, it provides a ready to use scalable cloud infrastructure for connecting things. It offers hardware clients for Arduino. Linux based systems ,Sigfox and ARMMbeb. There is also a Cloud Console related with the management front-end designed to easily manage your devices and visualize its information in the cloud. Documentation for using this platform is available to developers or companies [26].

Thingsboard: is a 100% Open source IoT platform and can host it as a SaaS or PaaS solution. IT provides device management, data collection, processing and visualization for your IoT projects. The standard protocols it supports for providing device connectivity are MQTT, CoAP and HTTP and supports both cloud and on-premises deployments. It

gives more than 30 customizable widgets allows you to build end-user custom dashboards for most IoT use-cases [23].

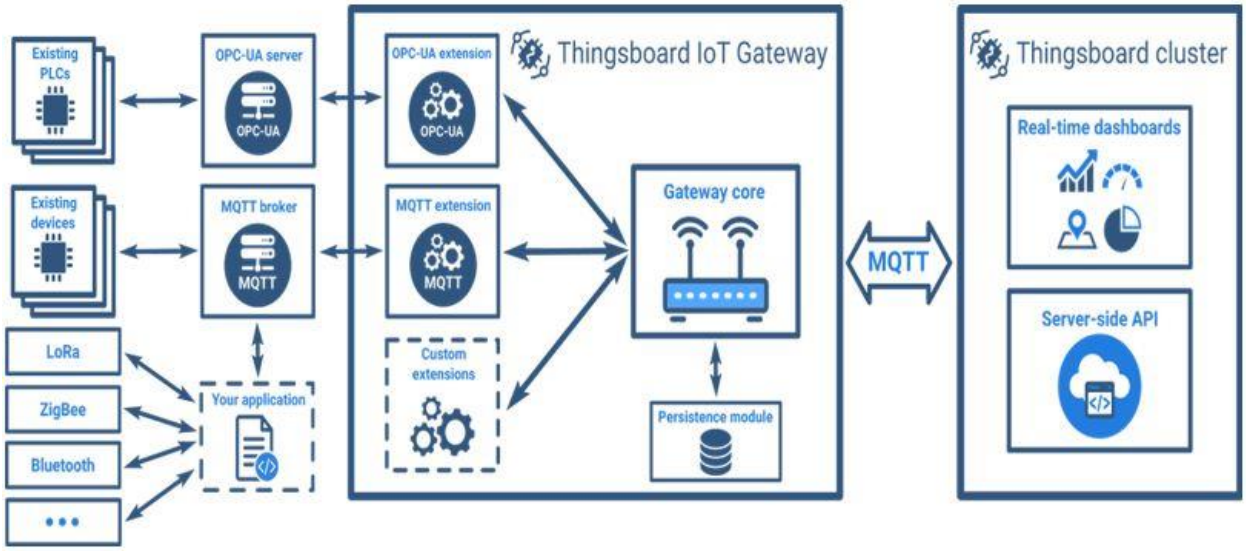

Figure 12. ThingsBoard Platform

Zetta: Is an API-first cloud platform and it means it as it converts any device to an API. From their website we get Zetta is an open source platform built on Node.js for creating Internet of Things servers that run across geo-distributed computers and the cloud. Zetta combines REST APIs, WebSockets and reactive programming – perfect for assembling many devices into data-intensive, real-time applications. Zetta servers run in the cloud, on PCs and on single-board computers. With Zetta you can link Raspberry Pis, BeagleBones and PCs together with cloud platforms like Heroku to create geo-distributed networks [27].

DSA: Distributed Services Architecture (DSA), is an open source IoT platform that facilitates device inter-communication, logic and applications at every layer of the Internet of Things infrastructure. The objective is to unify the disparate devices, services

and applications into a structured and adaptable real-time data model. The premise of the open source DSA initiative is to build a community of manufacturers, makers and solution providers that will contribute to an ever-expanding library of Distributed Service Links which allow protocol translation and data integration to and from 3rd party data-sources [28].

WSo2: allows exposing an API to power a mobile app allowing users to monitor and control their devices. It supports MQTT, HTTP, Websockets and XMPP protocols with IoT Server Framework extension for adding more protocols and data formats.

IoT Software Platform	Device management?	Integration	Security	Protocols for data collection	Analytics	Support for visualizations?	DB
Kaa IoT Platform	Yes	Portable SDK available to integate any particular platfrom, REST API	Link Encryption (SSL), RSA key 2048 bits, AES key 256 bits	MQTT, CoAP, XMPP, TCP, HTTP	Real time IoT Data Analytics and Visualization with Kaa, Apache Cassandra and Apache Zappelin	Yes	MongoDB, Cassandra, Hadoop, Oracle NoSQL
SiteWhere	Yes	REST API, Mule AnyPoint, and more	Link Encryption (SSL), Spring Security	MQTT, AMQP, Stomp, WebSockets, and direct socket connections	Real-time analytics (Apache Spark)	No	MongoDB, HBase , InfluxDB
ThingSpeak	No	REST and MQTT APIs	Basic Authentication	HTTP	MATLAB Analytics	No	MySQL

IoT Software Platform	Device management?	Integration	Security	Protocols for data collection	Analytics	Support for visualizations?	DB
DeviceHive	*Unknown	REST AP, MQTT APIs	Basic Authentication using JSON Web Tokens (JWT)	REST API, WebSockets or MQTT	Real-time analytics (Apache Spark)	Yes	Postgre SQL ,SAP Hana DB
Zetta	No	REST APIs	Basic Authentication	HTTP	Using Splunk	No	Unknown
Distributed Services Architecture (DSA)	NO	REST APIs	Basic Authentication	HTTP	No	No	ETSDB – Embedded Time Series
Thingsboard.io	Yes	REST APIs	Basic Authentication	MQTT, CoAP, and HTTP	Real time analytics(Apache Spark, Kafka)	No	Cassandra
Thinger.io	Yes	REST APIs	Link Encryption (SSL/TLS) and basic authentication	MQTT, CoAP, and HTTP	Yes	No	MongodB

IoT Software Platform	Device management?	Integration	Security	Protocols for data collection	Analytics	Support for visualizations?	DB
WSo2	Yes	REST APIs	Link Encryption (SSL) and basic authentication	HTTP, WSO2 ESB, MQTT	Yes, WSO2 Data Analytics Server	Yes	Oracle, Postgre SQL, MySQL, or MS SQL
Mainflux	Yes	REST APIs	JWT encrypted and signed tokens, OAuth2.0, public key infrastructure (PKI) and client-side certificates	HTTP, MQTT, WebSocket, CoAP)	Yes (integrated) Platform not confirmed	Yes	Cassandra, MongoDB or InfluxDB or Postgre SQL

Table 2. Comparison of IoT Platforms 1. Table taken from "9 Best & Top Open source IoT Platforms To Develop the IOT Projects"

Having listed the most competitive providers and the most known open-source solutions still the list of IoT cloud platforms is far from exhausted. In the below table published in a survey on IoT platforms [29] writer compares other available solutions.

Platform	Integration To Cloud	Supporting Protocols	Security	Type of Analytics	Application Mostly Used
Xively	Rest API	HTTP/HTTPS MQTT	SSL/TSL	Data Analytics, Business Analytics	Healthcare
Axeda	Rest API	MQTT, SOAP, REST, AMMP	SSL/TSL AES 128, RSA 2048	Data Analytics	Home Automation
Thingworx	Rest API	MQTT, XMPP, COAP, DDS,	TLS/AES 128	Data Analytics	Home Automation, Smart Traffic Management
Thing Square	Thing square mist	IPV6,RPL, 6lowPAN	AES protocol SSL	Data Analytics	Home Automation, Transportation
Bugswarm	Rest API, JSON	HTTP	RC4 based encryption protocol	Data Analytics	Smart Irrigation
Sensor cloud	Rest API	HTTP	TLS	Data Analytics	Home Automation
Thing speak	Rest API	HTTP	SSL	Data Analytics	Healthcare, Social IOT, Home Automation
Evrythng	Rest API	MQTT, COAP, web sockets	SSL	Data Analytics	Healthcare,
Every ware Device Cloud	Rest API	MQTT v 3.1	SSL	Data Analytics	Healthcare

Table 3. Comparison of IoT Platforms 2.

It is clear, that huge effort has been spent among the academic and industrial community trying to analyze and compare all the possible solutions for an IoT platform. From all that information at least, we should identify the quality indicators a user should use to

compare all these platforms. Device management along with support of the sensors of interest are the first prerequisites for the choice. The list of supporting protocols for data collection is also an important indicator. After that comes the way the integration towards cloud is achieved. Security, analytics visualization are the last ones to consider. Of course, it depends often on the application requirements to define which indicator is the crucial one. For example, if private data are collected for a health application then security is the number one factor to consider.

8 Proposal for an IoT PaaS

8.1 The idea

Implementing a functional IoT platform using the concept of sensor function virtualization and distributed intelligence could be very complex and expensive. However, prototype scenarios acting as a proof of concept are possible from developers with minimum cost.

In this chapter the author of the diploma will describe the architecture of a small scale IoT PaaS implementation. Part of it will be implemented. Following the three main parts of the architecture the sensors, the IoT gateway and the cloud along with the application are described.

Physical sensors of temperature, humidity together with a smoke sensor is all our testbed would need to create two virtual sensors. The first virtual sensor will combine data regarding temperature and humidity to provide to the end user a comfort level sensor. Data for temperature and smoke will be fused to create the second virtual sensor of fire detection.

Having decided the physical sensors, the next thing according to our research is the choice of a gateway. Raspberry Pi is a cheap solution to act as an IoT gateway. It is a tiny computer offering wired Ethernet, WiFi, and Bluetooth connectivity with a reasonably powerful processor.

Figure 13. Raspberry Pi 3

Raspberry Pi as every computer device will need an operating system. As IoT is really popular there are many choices of Linux based os to install in a Raspberry suitable for IoT applications. The author will follow the foundation's proposal which is Raspbian operating system. User can install it with NOOBS (New Out Of the Box Software – installer containing Raspbian) or download the image below and follow the installation guide of the official website.

On top of the operating system IoTivity framework using CoAP protocol for device to device communication is proposed. IoTivity is an open source software framework enabling seamless device-to-device connectivity to address the emerging needs of the Internet of Things. It is open source and thus there are many installation guides and support communities across the Internet. The framework is essential to discover the available sensors and communicate with them to retrieve the data.

The nature of the data the platform will collect are actually numeric values describing the temperature, the humidity and a level for the smoke (boolean value) so they can be stored locally in the Raspberry for a period (Raspberry Pi 2 and the Raspberry Pi 3 have 1 GB of RAM, Raspberry Pi 4 offers 1,2 or 4 GB). This advantage is proposed to be used for saving a local cache of the data collected so the gateway alone could answer for the virtual levels asked even if the cloud is unavailable.

The gateway is proposed to have a linux systemd service providing triplets of the data collected every minute. These triplets are going to be communicated towards the cloud through a POST request to a REST server located in the cloud side inside a docker container.

At this point the third part of our IoT platform the cloud server is coming to the surface. Openstack is proposed to be used as the cloud operating system managing the actual resources. A centos virtual machine (scalable if needed) containing docker containers deployed by docker-compose is proposed to be the heart of the cloud application or better service. Inside the virtual machine three basic containers offering isolated services will be deployed. One container will be the REST server acting as the interface with gateway, second one will be an ETCD database storing the data coming from the gateway and the third one will be the one processing the data and computing the virtual sensor

data. This way the end user can't really know the physical sensors existing as there are many layers of software abstraction in between.

All the tools and technologies used are open-source. Their main characteristics and details about the implementation part are given in the next chapters. Reader should keep in mind that the part implemented assumes the collection of physical data with a Raspberry Pi with a setup as described given and focuses in the cloud services part.

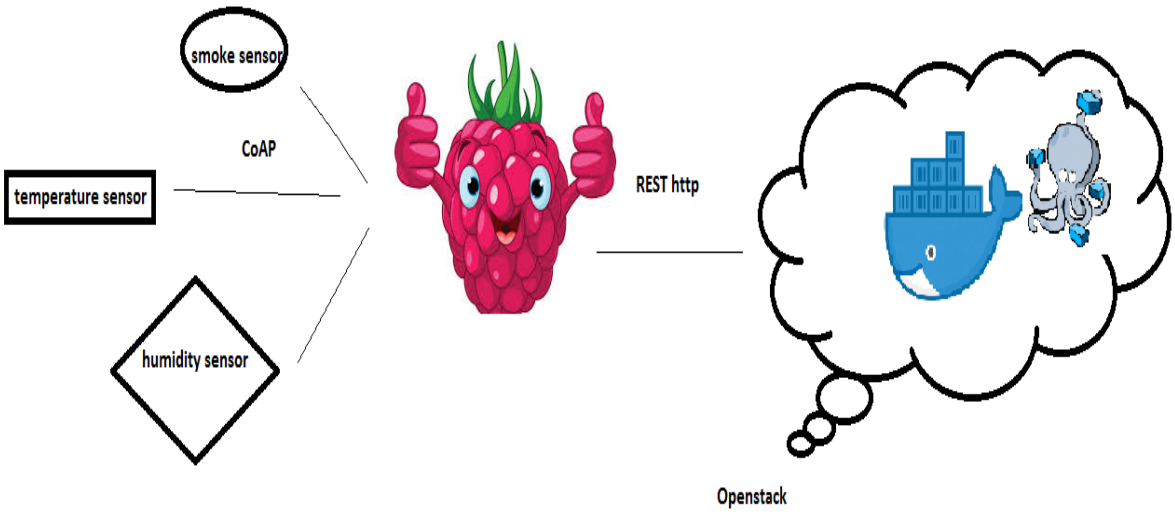

Figure 14. Proposed architecture of an IoT platform

8.2 The tools

8.2.1 Openstack

What is OpenStack? "OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. The project of Openstack is implemented as a collection of interacting services that control compute, storage, and networking resources. The cloud can be managed with a web-based dashboard or command-line clients, which allow administrators to control, provision, and automate OpenStack resources. OpenStack also has an extensive API, which is also available to all cloud users."

That is the definition the official page gives us regarding this software providing IaaS services. Openstack is a great solution as it is an open source software supported with a huge community. However, Openstack has a quite complex architecture with many services working together to achieve to provide IaaS functionality. It is worth mentioning that single developers to big companies rely on Openstack and can use for their purposes. Following, the chapter will try to describe the architecture of Openstack and the services from which it is consisted as the picture below also indicates.

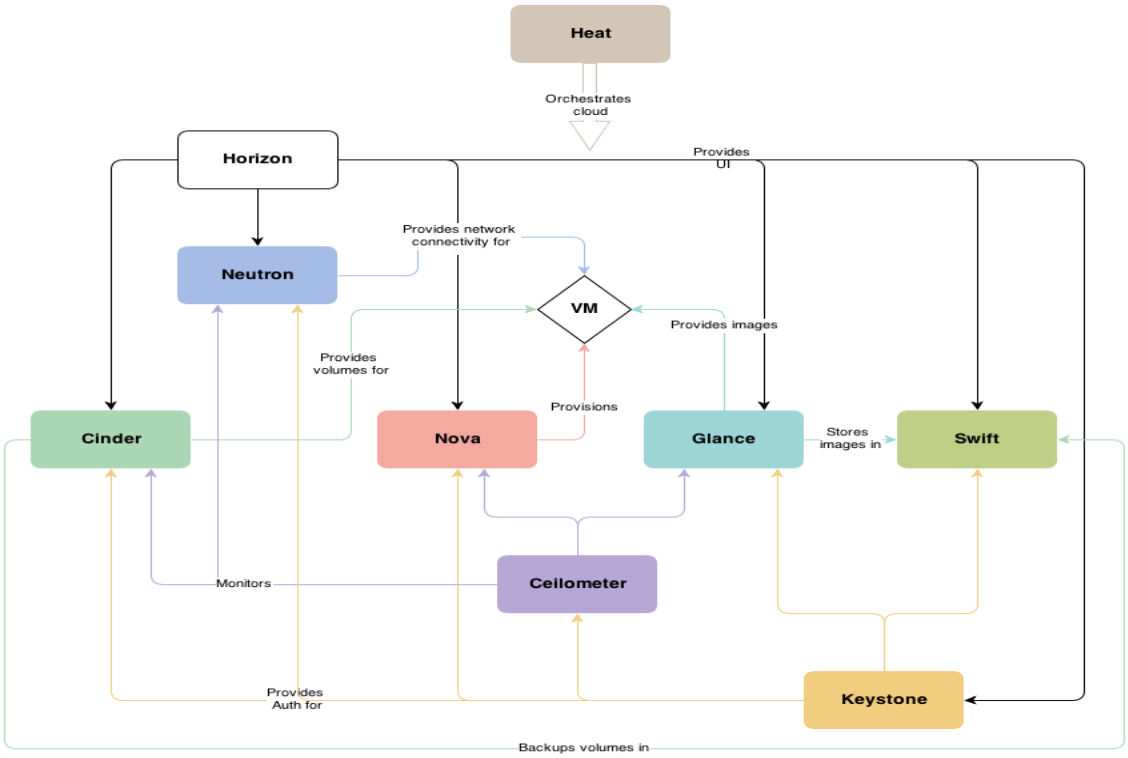

Figure 15. Openstack services.

The Openstack architecture

The following table also provides a general overview of the OpenStack services and their functionality. Every OpenStack service provides it includes a group of linux services and other components as well, there is an entire section in the official documentation dedicated to 3rd party components.

Service	Project name	Description
Dashboard	Horizon	Provides a web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.
Compute	Nova	Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decommissioning of virtual machines on demand.
Networking	Neutron	Enables Network-Connectivity-as-a-Service for other OpenStack services, such as OpenStack Compute. Provides an API for users to define networks and the attachments into them. Has a pluggable architecture that supports many popular networking vendors and technologies.
Storage		
Object Storage	Swift	Stores and retrieves arbitrary unstructured data objects via a RESTful, HTTP based API. It is highly fault tolerant with its data replication and scale out architecture. Its implementation is not like a file server with mountable directories.
Block Storage	Cinder	Provides persistent block storage to running instances. Its pluggable driver architecture facilitates the creation and management of block storage devices.
Shared services		
Identity service	Keystone	Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.
Image Service	Glance	Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.
Telemetry	Ceilometer	Monitors and meters the OpenStack cloud for billing, benchmarking, scalability, and statistical purposes.
Higher-level services		
Orchestration	Heat	Orchestrates multiple composite cloud applications by using either the native HOT template format or the AWS CloudFormation template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.
Database Service	Trove	Provides scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines.

Figure 16. Openstack components

The set of Openstack services

Networking (neutron) manages creation and life of the virtual networks, subnets, and routers in the OpenStack cloud. It gives to the user also the ability to deploy services like firewalls or virtual private networks(VPN). Networking in Openstack also enables the administrators to decide which individual services to run on which physical systems. All service daemons can be run on a single physical host for evaluation purposes. OpenStack Networking reacts in real-time if there is need to change something in networking, for

example a user can create and as assign IPs on the fly. Basic OpenStack networking advantages are:

- Users can create networks, control traffic, and connect servers and devices to one or more networks.
- Flexible networking models can adapt to the network volume and tenancy.
- IP addresses can be dedicated or floating, where floating IPs can be used for dynamic traffic rerouting.
- User can benefit from 4094 VLANs and 16 million tunnels in the cloud.

Block Storage (cinder) is the service which offers storage management for the virtual hard drives. User or administrator has the ability to create, delete and manage block storage devices. The attachment and detachment of devices is managed through integration with the compute service. Block storage could be used as a database and also to take snapshots of instances or create new storage volumes to keep your data.

OpenStack Block Storage advantages are:

- Creating, listing and deleting volumes and snapshots.
- Attaching and detaching volumes to running virtual machines.

Object Storage (swift) offers an HTTP-accessible storage system for data such as videos, images, email messages, files, or virtual machine's images. The Object Storage provides scaling and redundancy and it relies on other services of Openstack like Identity service for example.

Database as a Service (trove) allows users to perform complex tasks regarding database administration.

OpenStack Database as a Service advantages are:

- Management of multiple database instances in the Openstack cloud.
- Enabling tasks as deployment, configuration, patching, backup, restore, and monitoring.

OpenStack Compute (nova) is one of the basic services of an Openstack cloud as it provides virtual machines on demand. Compute service manages the virtual machines

and defines the virtualization mechanisms should interact with them in order to be functional.

Compute service supports the libvirt driver which uses KVM as the hypervisor. KVM hypervisor creates virtual machines and enables live migration from node to node. Compute also interacts with the Identity service to authenticate instance and database access, with the Image service to access images and launch instances, and with the dashboard service to provide user and administrative interface.

OpenStack Bare Metal Provisioning (ironic) is not one of the basic services which consist Openstack project despite that it is worth mentioning. It gives the opportunity to the user to provision physical, or bare metal machines, for a variety of hardware vendors with hardware-specific drivers. Furthermore, it cooperates with the Compute service to provision the bare metal machines like they were virtual machines.

OpenStack Image (glance) is like a record for virtual disk images. It provides many user actions like adding new images or taking snapshot from existing ones for backup. Registered images can be stored in the Object Storage service or elsewhere. The formats of image disks that glance supports are: aki/ami/ari (Amazon kernel, ramdisk, or machine image), iso (archive format for optical discs, such as CDs), qcow2 (Qemu/KVM, supports Copy on Write) ,raw (unstructured format) ,vhd (Hyper-V, common for virtual machine monitors from vendors such as VMware, Xen, Microsoft, and VirtualBox) ,vdi (Qemu/VirtualBox) and Vmdk (VMware) .Container formats are also supported.

OpenStack Orchestration (heat) offers templates to cloud resources like storage, networking, IPs, instances, volumes etc which are then parsed and executed. Templates are used to create clusters-stacks meaning a collection of virtual machine instances. It also provides scaling and high-availability. We are going to explain heat templates in more detail in the next chapter.

OpenStack Data Processing (sahara) provides management of Hadoop clusters But what is a Hadoop cluster? According to official documentation : “Hadoop clusters are groups of servers that can act as storage servers running the Hadoop Distributed File System (HDFS), compute servers running Hadoop’s MapReduce (MR) framework, or both Hadoop stores and analyze large amounts of unstructured and structured data in clusters. The servers in a Hadoop cluster need to reside in the same network, but they do not need to share

memory or disks. Therefore, you can add or remove servers and clusters without affecting compatibility of the existing servers. The Hadoop compute and storage servers are co-located, which enables high-speed analysis of stored data. All tasks are divided across the servers and utilizes the local server resources."

OpenStack Identity (keystone) is also one of the basic services of Openstack enabling authentication and authorization for users. It provides a variety of authentication mechanisms, like user name / password credentials, token-based systems, and AWS-style log-ins. The default database for this service is MariaDB but someone could use LDAP or SQL. Identity also can support federation with Security Assertion Markup Language (SAML**)**, which is an xml-based data format for exchanging authorization and authentication between for example an identity and a service provider.

OpenStack Dashboard (horizon) is actually the graphical user interface for users and administrators for the Openstack cloud, through the interface users can perform operations like creating instances of VM, storing images, managing networking, setting access control and keys etc. The Dashboard service provides three default dashboards : the Project, Admin, and Settings.

OpenStack Telemetry (ceilometer) offers user-level usage data for OpenStack-based clouds. The data collected could be used for customer billing, system monitoring, or alerts. The service uses notifications from OpenStack components or by OpenStack infrastructure resources such as libvirt. Furthermore, it includes a storage daemon that communicates with authenticated agents via a messaging system to collect the data and also through a plug-in you could add new monitors.

8.2.2 Docker

The concept of docker in virtualization became popular as a light-weight solution to run applications and don't have to devote a whole virtual machine to it. Instead docker so called containers share the resources of a host eg a virtual machine. This concept is known as containerization.

As docker's official website describes [33]:" A container is launched by running an image. An **image** is an executable package that includes everything needed to run an application-

-the code, a runtime, libraries, environment variables, and configuration files. A container runs natively on Linux and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight. By contrast, a virtual machine (VM) runs a full-blown "guest" operating system with virtual access to host resources through a hypervisor. In general, VMs provide an environment with more resources than most applications need."

The technology behind docker is taking advantage of Linux namespaces to offer isolation.

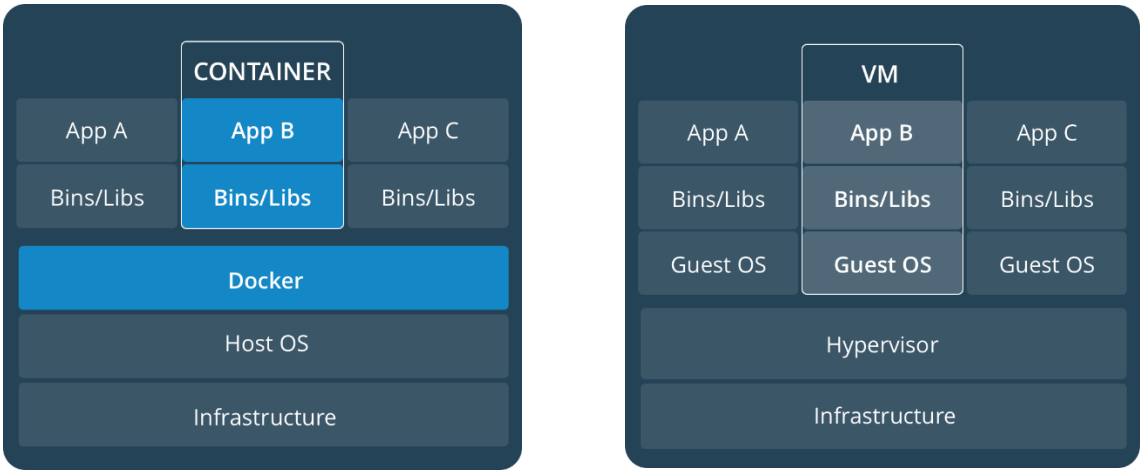

Figure 17. Docker container technology

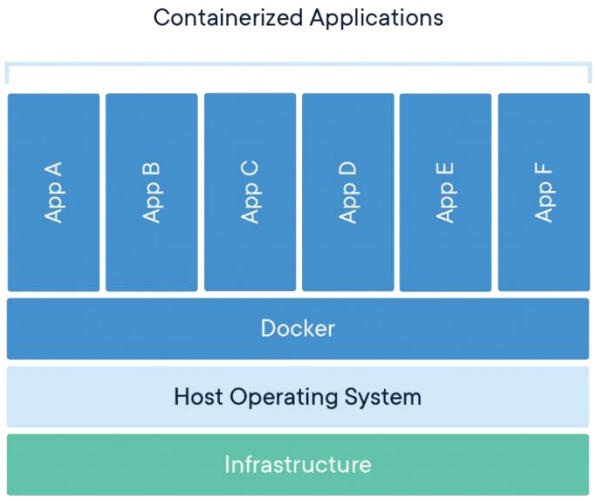

Figure 18. Containerized applications

8.2.3 Docker-compose

Compose is an open-source tool for defining and running multi-container docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration [34].

Compose is ideal for development, testing, and staging environments, as well as Continuous Integration.

Using Compose is basically a three-step process.

1. Define your app's environment with a Dockerfile so it can be reproduced anywhere.
2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
3. Lastly, run docker-compose up and Compose will start and run your entire app.

8.3 The protype implementation

8.3.1 Installing Openstack

Starting point to use Openstack is of course having one up and running. The easiest way to achieve this as a single developer assuming you don't have the infrastructure of a large company with many servers is to install it on a virtual machine. Fortunately, there is packstack! Packstack is an installation utility developed by the team of Openstack that uses puppet modules to deploy Openstack services on a single virtual machine! It comes with a precious guide with steps to help any single developer to start using Openstack. Below the procedure that was followed according to the steps of packstack recipe in order to install successfully our own personal Openstack is described like a guide.

Which are the Prerequisites?

1. Install Virtual Box to your laptop or desktop computer in order to deploy a virtual machine. The image that was used in our case was Centos7. Actually, RedHat images or the free version of them which is centos are recommended as the ideal host for this project from Openstack team itself. It is also crucial to follow the hardware requirements which are: "Machine with at least 4GB RAM, preferably 6GB RAM, processors with hardware virtualization extensions, and at least one network adapter." Virtual box gives us plenty the ability to set plenty parameters for our virtual machine. Except RAM that was required another setting regarding network was also needed in order to have network connectivity. That was succeeded by choosing the NAT option for our centos7. After that we launch the virtual machine. It will ask for information regarding time and language, apart from that we just wait for it to complete the installation.

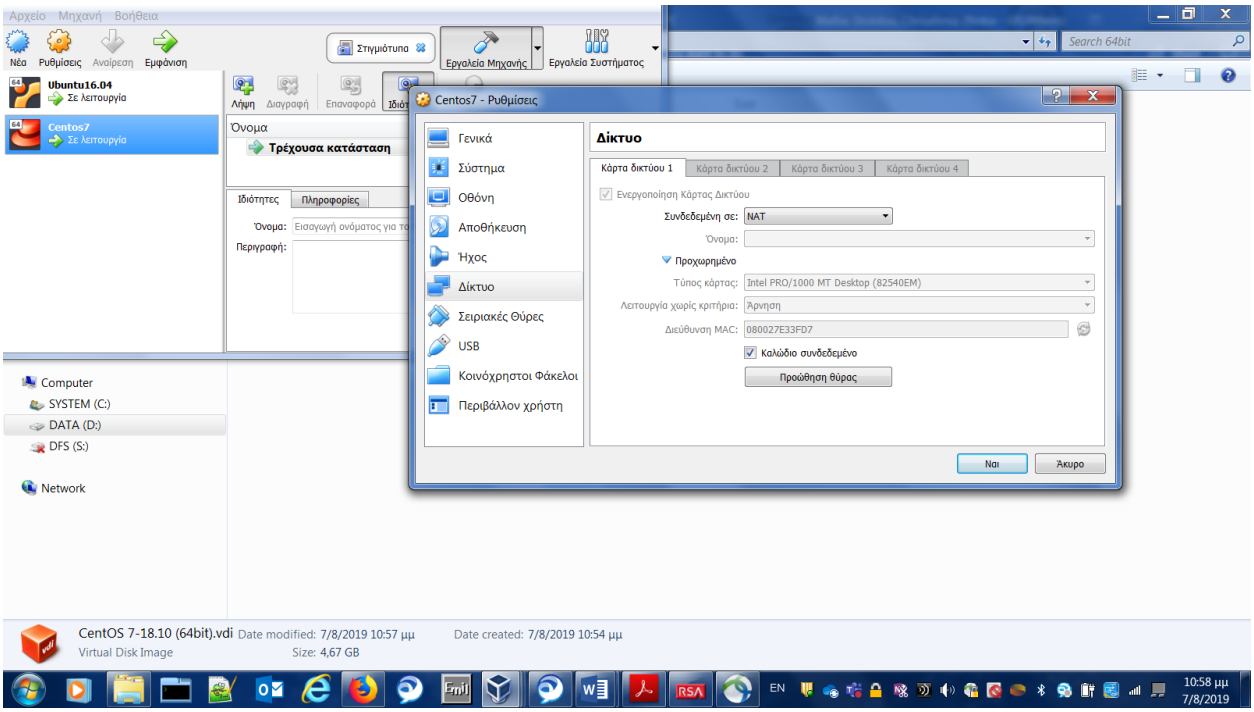

Figure 19. Create a virtual machine using VirtualBox

2. Additionally, some settings regarding network should be done to ensure our network access to the server and instances:
    - $ sudo systemctl disable firewalld
    - $ sudo systemctl stop firewalld
    - $ sudo systemctl disable NetworkManager
    - $ sudo systemctl stop NetworkManager
    - $ sudo systemctl enable network
    - $ sudo systemctl start network

3. Download and install repositories that are needed for the installation of Openstack by executing the below commands in your terminal. An interesting detail is that all Openstack releases are in an alphabetical order and named after a city or country near where the Openstack team worked for the specific release.

- $ sudo yum update -y

- $ sudo yum install -y centos-release-openstack-stein
- $ sudo yum update -y
- $ sudo yum install -y openstack-packstack
- $ sudo packstack --allinone

Our own Openstack is up and running! A file with the name keystonerc_admin should have been created in our root directory, we can check it out with the command cat keystonerc_admin. The result is going to like in the print screen image below.

```
unset OS_SERVICE_TOKEN
export OS_USERNAME=admin
export OS_PASSWORD=3f1c21d65fbc4633
export OS_AUTH_URL=http://10.0.2.15:500
export PS1='[\u@\h \W(keystone_admin)]\

export OS_TENANT_NAME=admin
export OS_REGION_NAME=RegionOne
[root@localhost ~(keystone_admin)]#
```

The information this file gives us are actually all we need to access the horizon of our Openstack meaning the graphical interface we have already discussed. At this point we need a browser to login to the Openstack dashboard (horizon).We can open our Mozilla Firefox browser from applications and access our Openstack gui through the url specified in the keystonerc_admin file in our case http://10.0.2.15. A log in screen will appear and we can use the credentials provided from the same file. Below the print screen images of our installed dashboard are shown.

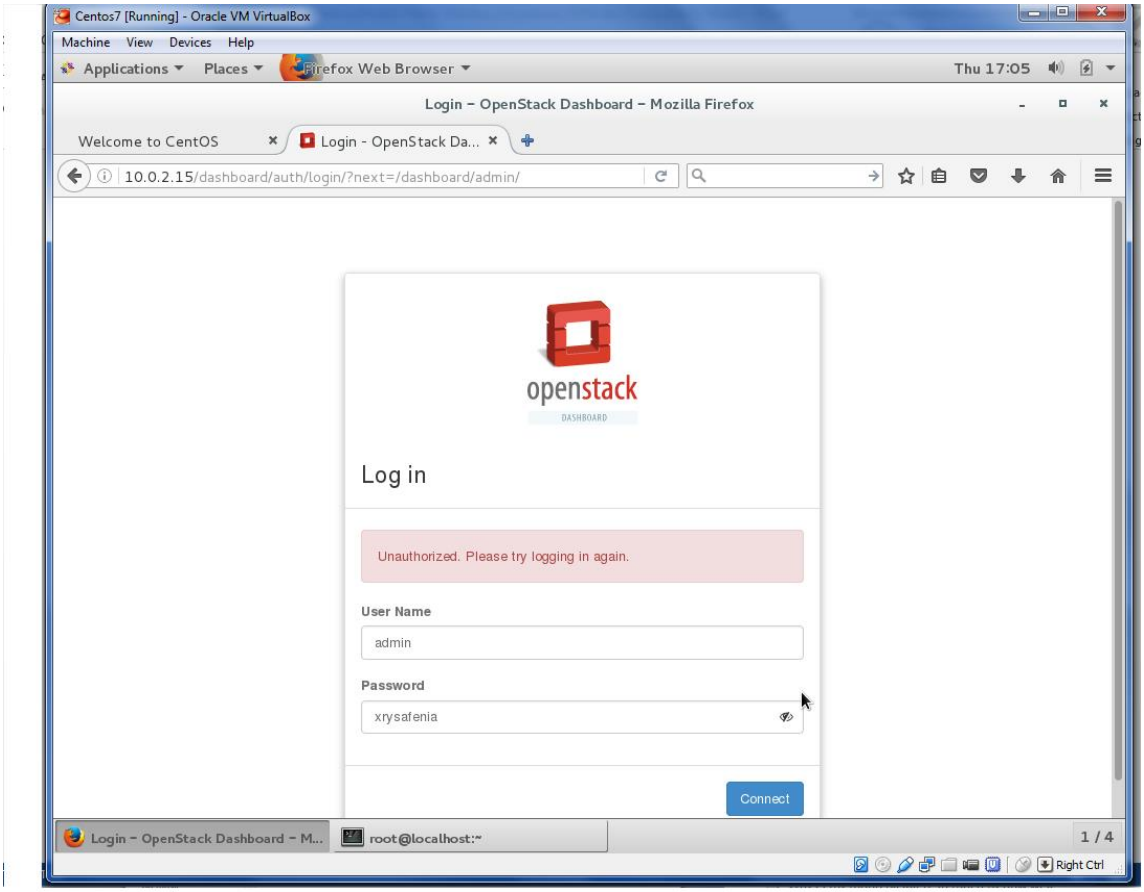

Figure 20. Login screen of Openstack

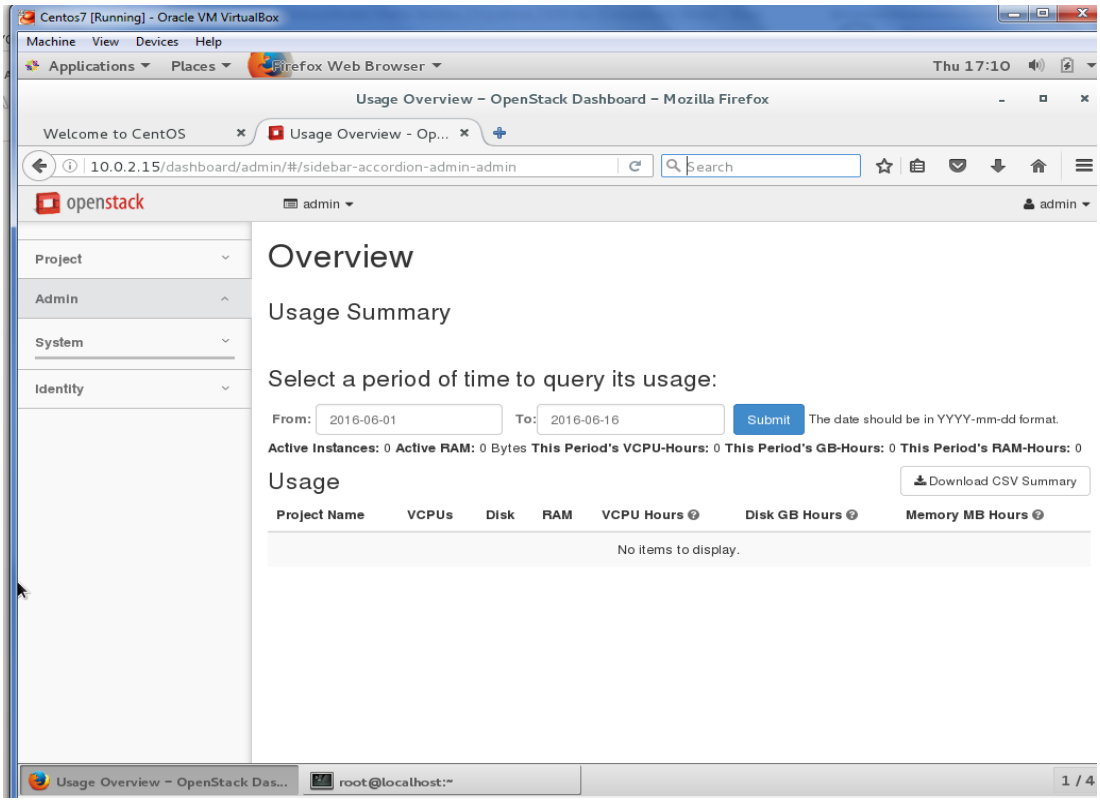

Figure 21. Horizon GUI of Openstack

Before explaining a little more about orchestration service and heat templates we should create a public network in our Openstack. This task is accomplished if we follow the steps below in the dashboard:
Admin->Projects->Delete all the default ones
Admin->Networks->Delete all the default ones
Admin->Flavors->Delete all the default ones
Admin-> Networks->Create network called public with settings as shown below
Admin->Networks->Public->Create subnet called public-subnet. With the same procedure we can create other networks and subnets. For example, one public network, one internal and a router should look like below

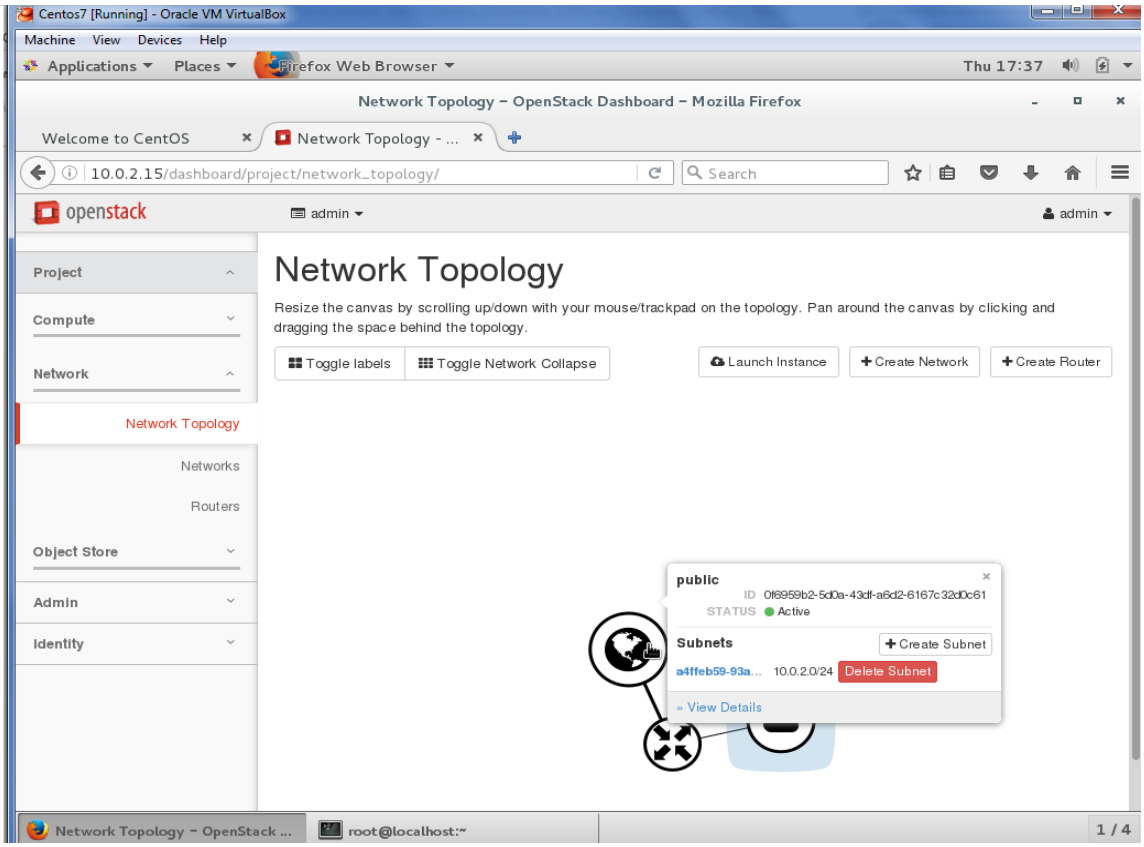

Figure 22. Network Topology section in Openstack

The images in Openstack can be created through horizon gui or usig Openstack cli commands from a terminal. Heat is the service in Openstack responsible for that. From the official website: "**Heat** is the main project of the OpenStack orchestration program. It allows users to describe deployments of complex cloud applications in text files called *templates*. These templates are then parsed and executed by the Heat engine".

Heat templates are written as structured yaml text files. Yaml files have a specific human friendly syntax for all programming language.

The first line in a heat template is the heat_template_version which is a mandatory section that is used to specify the version of the template syntax that is used. Following the developer optional could add a description of what the template does. Then the

section of the various resources is next describing all the instances, networks, flavors, keys we are going to use.

Heat templates are very useful when creating cluster of virtual machines.

8.3.2 Deploying a Virtual Machine

For our implementation we are going to concentrate in the architecture and software inside a Centos based virtual machine. Of course, using Openstack we could create as many identical virtual machines we need (scalability on demand) and enable high-availability among them.

Also, the virtual machine status could be saved in a new Linux based image in order to use it after the implementation is done as a template. The same concept applies also with docker containers running inside a VM. So, the software implementation and the configuration a developer does for an application/service inside a VM or a container can be saved and used as a canvas just like Virtual Box.

Horizon gui is used for the creation of a virtual machine:

Instances -> Launch Instance

Name, Image (eg Centos7), flavor (VCPUs/RAM), and Network are the mandatory fields.

Below pictures show the procedure:

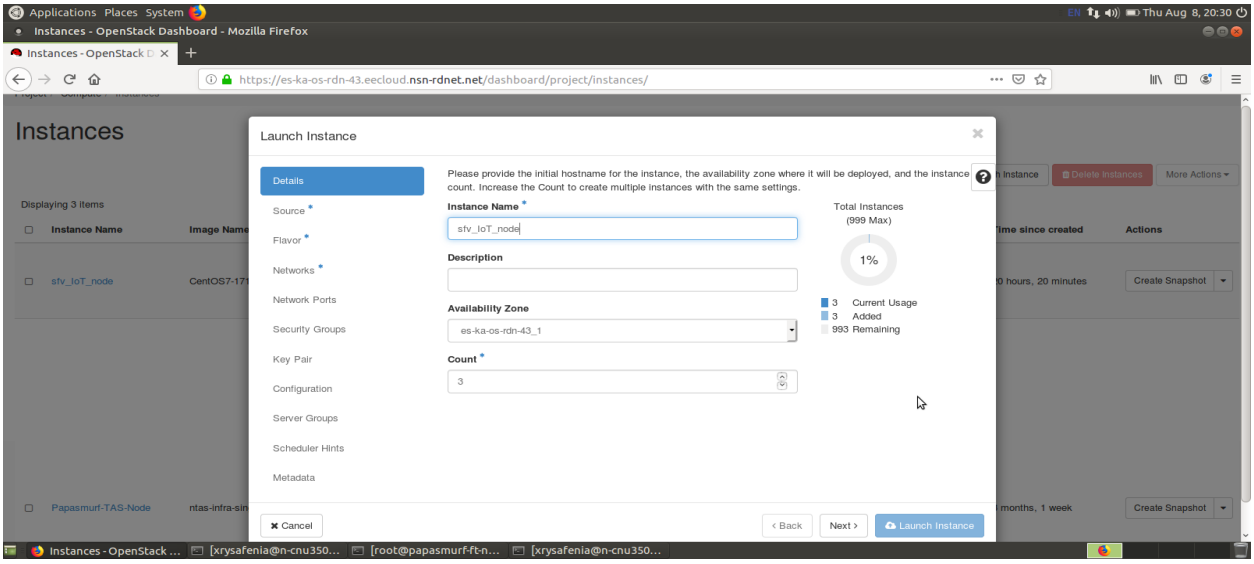

Figure 23. Launch a Cent0s7 VM in Openstack (1)

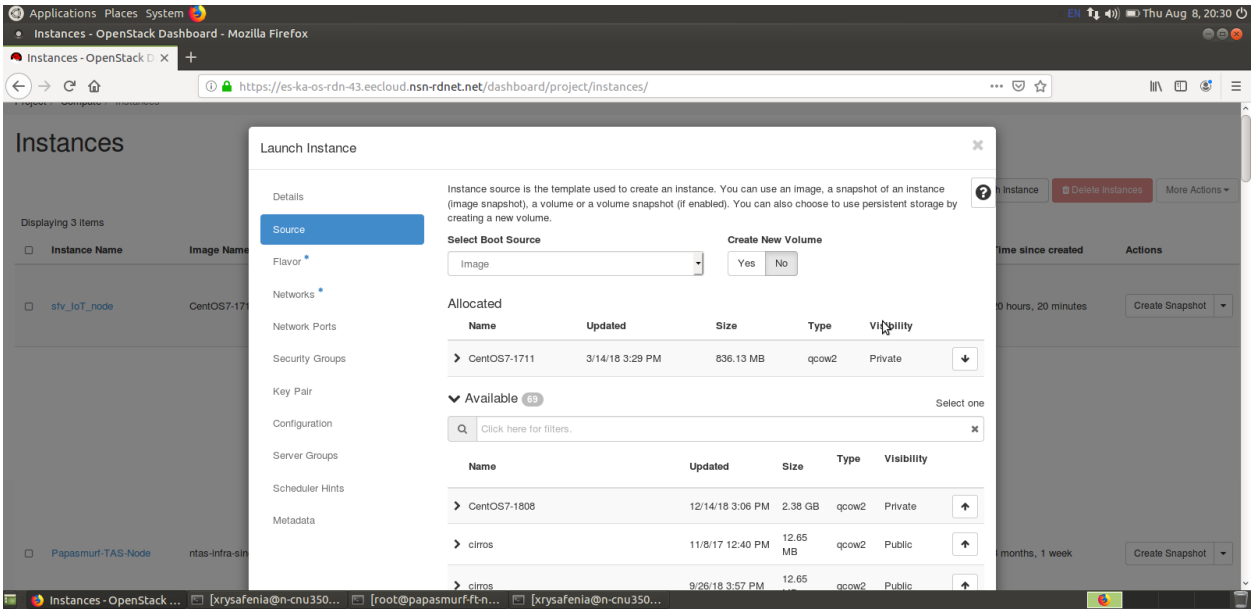

Figure 24. Launch a Centos7 VM in Openstack (2)

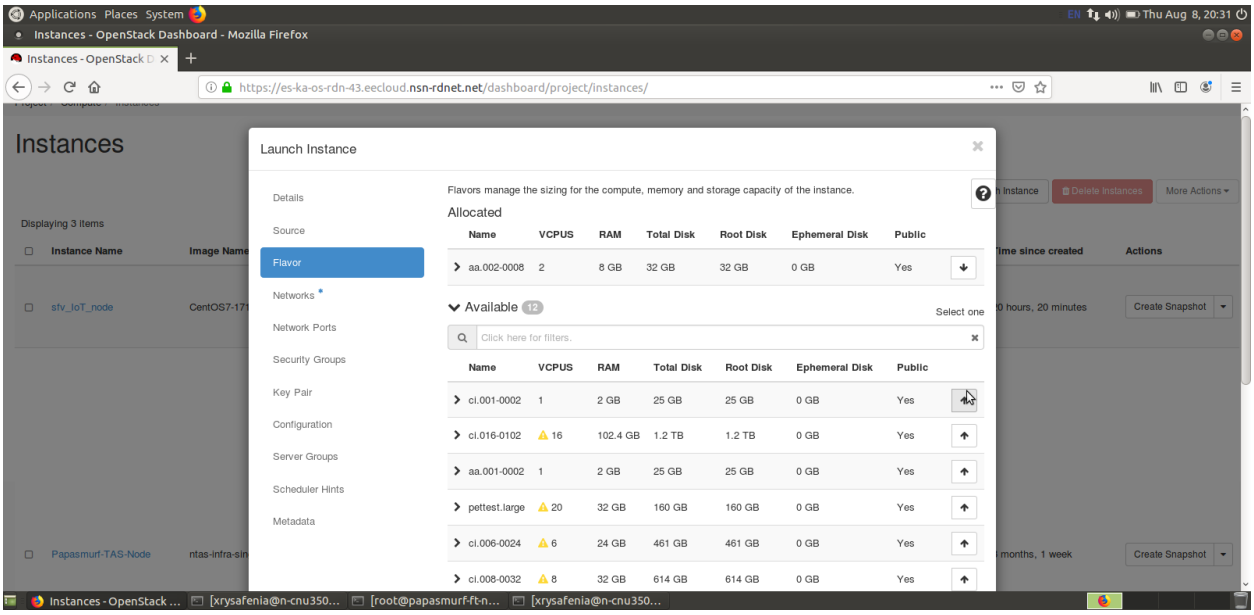

Figure 25. Launch a Centos7 VM in Openstack (3)

After that in Instances tab we can see our Centos VM named sfv_IoT_node

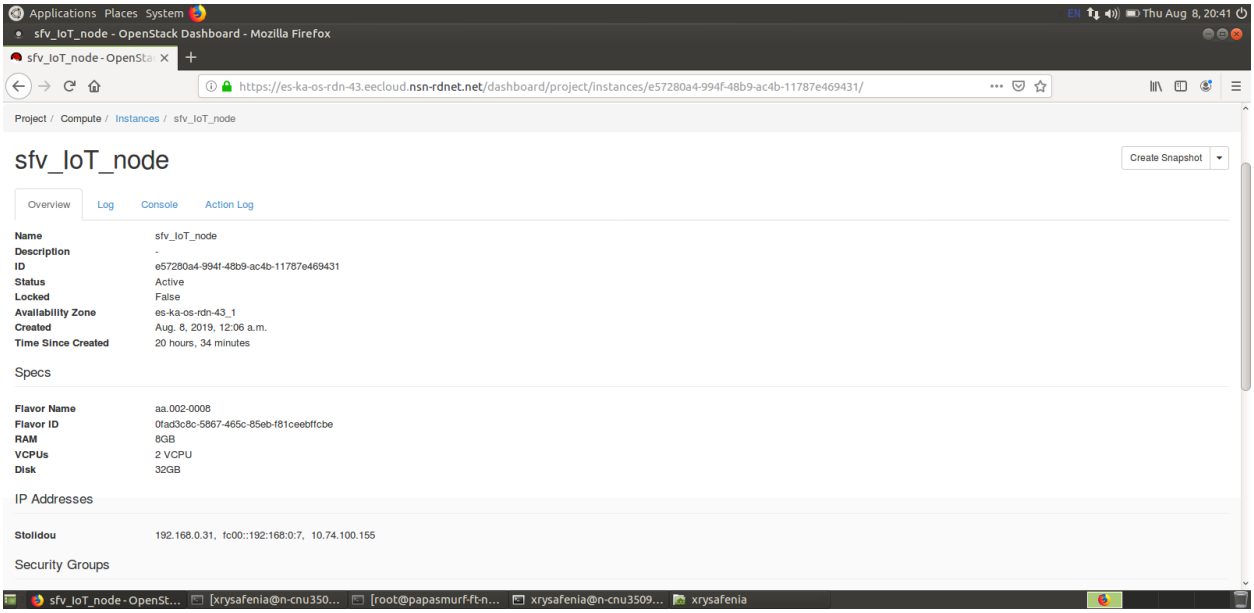

Figure 26. Centos7 VM named sfv_IoT_node

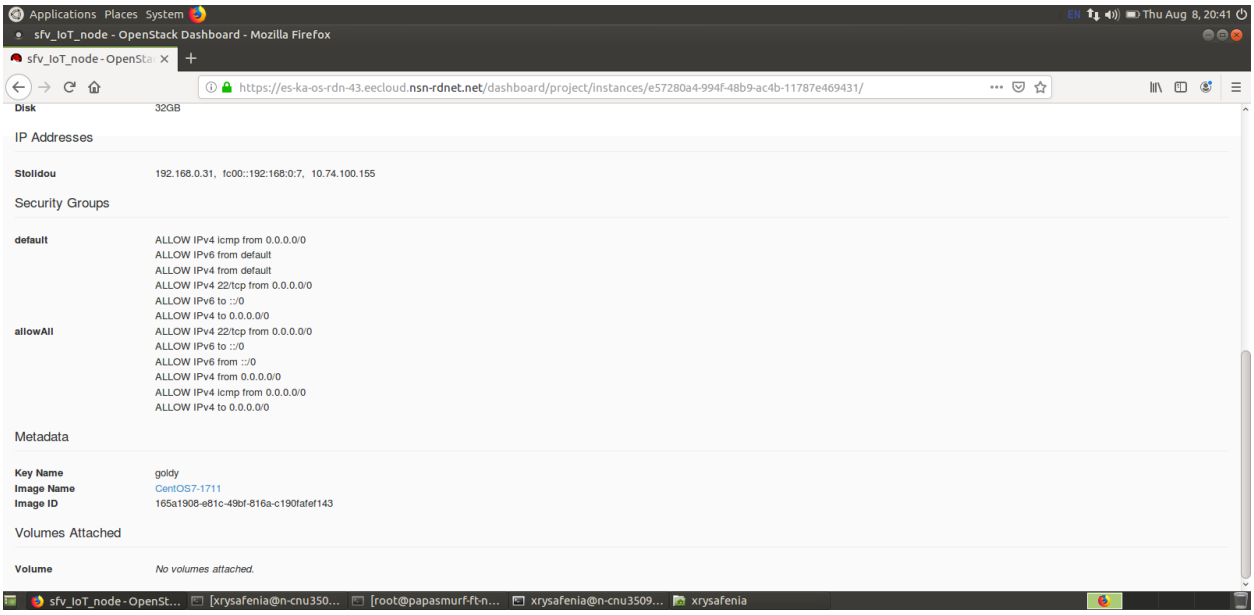

Figure 27. Deployment details of sfv_IoT_node

Finally, using action button a floating-IP is assigned to our node meaning an address we could use to open a ssh connection and login.

Above picture shows the IP:10.74.100.155

Figure 28. SSH login to sfv_IoT_node

8.3.3 Install docker and docker-compose inside our svf_IoT_node

There are many guides to install docker and docker-compose.

The commands executed inside the VM we had just created are listed below:

1) Docker is actually a prerequisite:

    - yum -y update
    - yum -y install docker
    - systemctl enable docker
    - systemctl start docker

2) Fetch and install docker-compose:

- sudo curl -L "https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
- Once the download is complete, make the binary executable by typing: sudo chmod +x /usr/local/bin/docker-compose
- To verify the installation, execute the following command to print the version: docker-compose --version

8.3.4 Prepare the three docker images and deploy

Linux Centos7 acts as our base image meaning that functionality and software will be added on top of it and then the result is saved in a docker image ready to be deployed in a docker container.

- ✓ ETCD database container will be deployed with an official image for this purpose. It is available for the public meaning we fetch the image just by performing a “docker pull” command.
  The version used is quay.io/coreos/etcd:v3.2.7
- ✓ Rest server container needs development work on top of Centos7. Keep in mind

that REST server implementation should write sensor data to ETCD database through a POST request but also fetch them through a GET request. So, it should support an etcd client in order to implement such REST endpoints.
For this purpose, two open-souse projects were combined, the c++ REST api Casablanca provided by Microsoft to developers for cloud-based client/server communication and the Nokia's C++ API for etcd, both available on github.

- ✓ Virtual Sensor container image on top of the base image has only some algorithms implemented in bash scripts for fusing the sensor data it gets from the database via GET requests and producing the "virtual sensor data". The exact algorithms an application could use to calculate comfort level from temperature and humidity is out of the scope of this document. Instead, simple mathematical operations used in a "dummy script" developed for this image.

Following, below yaml template used as an input to docker-compose to deploy the containers described above.

```
version: '2'
services:
 sfv_REST:
  image: sensor/sfv_rest:1.1
  environment:
   ETCD_ADDRESS: "etcd:4001"
  ports:
   - "8080:80"
  links:
   - etcd
 vsensor_data_manager:
  image: vsensor_data_manager:1.0
  entrypoint: /bin/bash -c "/virtual_sensor.sh"
  links:
   - sfv_REST:sfv_REST
   - etcd:etcd
 etcd:
  image: quay.io/coreos/etcd:v3.2.7
```

- ✓ Deploy the triplet of docker containers as configured above with command: ***docker-compose up -d***

8.3.5 Demonstrate basic scenario steps

Having deployed with docker-compose the containers a list of the basic steps simulating the raspberry "POST" some sensor data, data written to ETCD database, virtual sensor on demand fetching triplets of data identified by a timestamp from the database and provide a result for virtual sensor data to the end user.

1) Hypothetical raspberry having connectivity to the REST server container on the cloud side sends sensor data:

   curl -X POST http://172.18.0.3:8080/v1/send-sensor-data -d '{"timestamp":"30-09-2019:2:58:40", "humidity":"60", "temperature":"19", "smoke":"normal"}'

2) Check ETCD database for the entry:

   / # export ETCDCTL_API=3
   / # etcdctl get / --prefix=true
   /rasp/30-09-2019:2:58:40/60/19/high

3) Get result from virtual sensor container where dummy script logs the information after getting the sensor data

   Possible fire

   curl -X GET http://sfv_REST:8080/v1/get-sensor-data

   % Total % Received % Xferd Average Speed Time Time Time Current

   Dload Upload Total Spent Left Speed

   100 125 100 125 0 0 15377 0 --:--:-- --:--:-- --:--:-- 15625

   {"data":{"sensor_data":[{"humidity":"19","smoke":"high","temperature":"60",

   "timestamp":"30-09-2019:2:58:40"}]},"status":"OK"}

Below is shown the screenshot of the scenario described

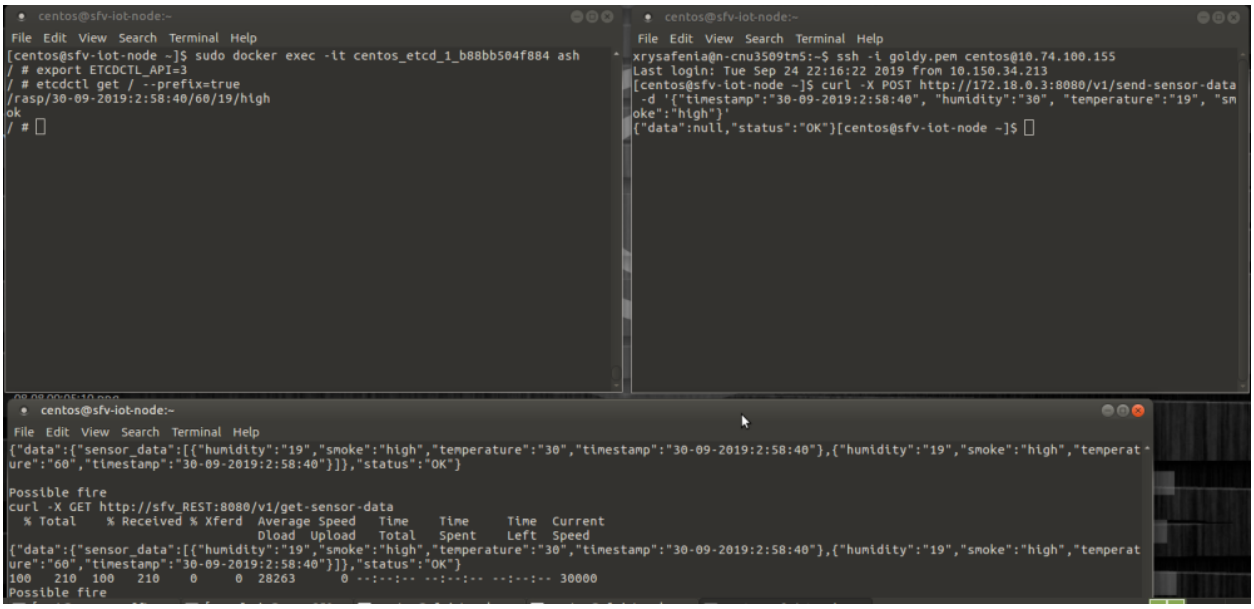

Figure 29. Test prototype scenario for sensor function virtualization in IoT Platform

9 References

[1] IEEE Internet of Things, “Towards a definition of the internet of things (iot),”IEEE, Tech. Rep., May 2015. [Online].Available:http://iot.ieee.org/images/files/pdf/IEEEIoTTowardsDefinitionInternetofThingsRevision127MAY15.pdf

[2] N. Bizanis and F. A. Kuipers, “SDN and Virtualization solutions for the Internet of Things: A Survey,” IEEE Access, vol. 4, pp. 5591–5606,2016.

[3] Wikipedia contributors. "Radio-frequency identification." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 27 Aug. 2019. Web. 18 Sep. 2019.

[4] T. Kivinen and P. Kinney, “IEEE 802.15.4 Information Element for the IETF,” RFC 8137, May 2017.

[5] Y. Zhou, X. Yang, L. Wang, and Y. Ying, “A wireless design of low-cost irrigation system using zigbee technology,” inProc. of Int. Conf.on Networks Security, Wireless Comm. and Trusted Comp., vol. 1, April2009, pp. 572–575.

[6] C. Shao, D. Hui, R. Pazhyannur, F. Bari, and R. Zhang, “IEEE 802.11Medium Access Control (MAC) Profile for Control and Provisioning of Wireless Access Points (CAPWAP),” RFC 7494, Apr. 2015.

[7] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby, and C. Gomez, “IPv6 over BLUETOOTH(R) Low Energy,” RFC 7668, Oct. 2015.

[8] C. Bisdikian, “An overview of the bluetooth wireless technology,”IEEE Commun. Mag., vol. 39, no. 12, pp. 86–94, December 2001.

[9] Z. Shelby and C. Bormann,”6LoWPAN: The wireless embedded Internet. John Wiley & Sons”, 2011, vol. 43.

[10] N. Kushalnagar, G. Montenegro, and C. Schumacher, “Ipv6 over low-power wireless personal area networks (6lowpans): overview, assumptions, problem statement, and goals,” RFC 4919, August 2007.

[11] T. S. Rappaport, S. Sun, R. Mayzus, H. Zhao, Y. Azar, K. Wang, G. N.Wong, J. K. Schulz, M. Samimi, and F. Gutierrez, “Millimeterwave mobile communications for 5G cellular: It will work!”, IEEE Access, vol. 1, pp. 335–349, 2013.

[12] Floris Van den Abeele , Jeroen Hoebeke, Girum Ketema Teklemariam, Ingrid Moerman , Piet Demeester “Sensor Function Virtualization to Support Distributed Intelligence in the Internet of Things”, Wireless Personal Communications, April 2015.

[13] Andrea Zanella, Senior Member, Nicola Bui, Angelo Castellani, Lorenzo Vangelista and Michele Zorzi, “Internet of Things for Smart Cities”, February 2014, vol.1.

[14] Nuortio, J. Kytöjoki, H. Niska, and O. Bräysy,“Improved route planning and scheduling of waste collection and transport,” Expert Syst. Appl.,vol. 30, no. 2, pp. 223–232, Feb. 2006.

[15] “Precision Farming”, <https://www.iotone.com/usecase/precision-farming/u64 >

[16] “Platform for intelligent IoT services”, <https://cloud.google.com/solutions/iot/ >

[17] “AWS IoT Core features”, < https://aws.amazon.com/iot-core/features/ >

[18] “Azure IoT solution accelerators”, < https://azure.microsoft.com/en-us/features/iot-accelerators/>

[19] Greg Knowles, 2015 < https://www.ibm.com/blogs/cloud-archive/2015/09/iot-real-time-insights/>

[20] “Kaa Platform Features”, <https://www.kaaproject.org/overview#connectivity>

[21] “The Open Platform for the Internet Of Things”, <https://sitewhere.io/en/ >

[22] Rajesh Sola, SiteWhere, 2017, <https://opensourceforu.com/2017/07/sitewhere-open-platform-connected-devices/>

[23] H2S Media Team, “9 Best & Top Open source IoT Platforms To Develop the IOT Projects”, 2019, <https://www.how2shout.com/tools/best-opensource-iot-platforms-develop-iot-projects.html>

[24] “DeviceHive”, <https://www.devicehive.com/#analyze>

[25] <https://www.mainflux.com/>

[26] <http://docs.thinger.io/>

[27] <http://www.zettajs.org/>

[28] <http://www.iot-dsa.org/ >

[29] M. Sruthi, B. R. Kavitha,”A SURVEY ON IOT PLATFORM”, International Journal of Scientific Research and Modern Education Volume I, Issue I, 2016.

[30] John Teel, ”Comparison of Wireless Technologies” “Comparison of WirelessTechnologies”, 2017, <https://predictabledesigns.com/wireless_technologies_bluetooth_wifi_zigbee_gsm_lte_lora_nb-iot_lte-m/>

[31] Shang Guoqiang , Chen Yanming, Zuo Chao, Zhu Yanxu, “Physical and Social Computing”, IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, 2013.

[32] Andre Gloria, Francisco Cercasa, Nuno Soutoa, “Design and implementation of an IoT gateway to create smart environments”, The 8th International Conference on Ambient Systems, Networks and Technologies, 2017.

[33] Byungseok Kang, Daecheon Kim, and Hyunseung Choo, ”A Large-Scale Autonomic IoT Gateway”, IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS, VOL. 3.

[33] https://docs.docker.com/get-started/

[34] https://github.com/docker/compose

[35] Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges Keyur K Patel1, Sunil M Patel2 , PG Scholar1 Assistant Professor2, Department of Electrical Engineering Faculty of Technology and Engineering-MSU, Vadodara, Gujarat, India

[36] <https://github.com/microsoft/cpprestsdk>

[37] <https://github.com/nokia/etcd-cpp-apiv3>

[38] Niyato, Dusit & Kim, Dong & Maso, Marco & Han, Zhu, "Wireless Powered Communication Networks: Research Directions and Technological Approaches", IEEE Wireless Communications, 2017.

[39] "European Smart Cities." European Smart Cities, <http://www.smart-cities.eu/.>

[40] N. Kushalnagar, G. Montenegro, C. Schumacher , "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals (RFC4919)", August 2007.

[41] Z. Shelby, K. Hartke, C. Bormann, "The Constrained Application Protocol (CoAP) (RFC7252)", June 2014.