Master of Science in Electrical Engineering
May 2018.

Investigating and Deploying an AI model on Raspberry Pi IoT platform using FIWARE and Docker in the context of the Bonseyes AI Ecosystem

Yogitha Manasa Ummadi setty
Sai Prakash Mamidi

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Electrical Engineering with Emphasis on Telecommunication Systems. The thesis is equivalent to 20 weeks of full-time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:
Author(s):
Yogitha Manasa Ummadi setty
E-mail: youm17@student.bth.se

Sai Prakash Mamidi
E-mail: sama17@student.bth.se

University advisor:
Kurt Tutschku,Prof. Dr.
Department of Computer Science and Engineering (DIDD)

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Many high-end sophisticated devices are being replaced by small single-board IoT devices. It is estimated that by the year 2025 there will be about 75 billion IoT connected devices worldwide. This project buttresses the earlier statements. The project will be focused on how to deploy a complex AI face-recognition model on a simple, less power consumption, single-board IoT device Raspberry Pi3. Also, to simplify the whole process of deploying the AI model on Raspberry Pi3 by dockerizing. By just pulling the respective docker image and running it on the end device (Raspberry Pi3), we can make face recognition happen. Finally, the obtained results will be sent to the cloud (Fiware) using NGSI API. Instead of sending the whole video feed to the Cloud platform and performing the computing there, we managed to perform the computing at the edge and then send the results to the cloud and making the results accessible to requested users.

Keywords: Raspberry Pi3, IoT, Fiware, Docker, Bonseyes, Machine Learning, Virtualization.

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

ACKNOWLEDGMENT

We would like to express our obligation to our supervisor Kurt Tutschku, Prof. Dr. for giving us this exciting Project and wonderful encouragement throughout our thesis work. Without his support, this work would not have been completed. We would also like to thank Ms. Vida Ahmadi Mehri for her kind help during this thesis work. It has been a great honor to work under their supervision.

Furthermore, we would like to thank each other for our great contribution to the project. Finally, we would also like to thank our friends, family, and all others who have supported us during this work.

Sai Prakash Mamidi.
Yogitha Manasa Ummadisetty.

Karlskrona Sweden,
21st May 2019

CONTENTS

LIST OF FIGURES AND GRAPHS

LIST OF TABLES

1 INTRODUCTION

As the Internet of things (IoT) is going to play a vital role in the field of technology in the very near future, it is important to make experimental research and find out the strengths and drawbacks involved in it. Multiple devices connected over the internet are called "IoT devices". We intend to use one the sophisticated IoT device of our time and perform complex actions to find out its constraints.

This thesis contributes to the understanding of subparts of Bonseyes and on how the overall system needs are implemented. Moreover, it eventually provides insight into how the Bonseyes system should be redesigned or improved. The main aim is to have an Artificial Intelligence on IoT end device (edge computing) and send the result to Fiware (cloud computing) and create a docker image of this whole process. Artificial Intelligence as in the face recognition model which will be trained to recognize known faces as the assigned name which will be the result of the AI model. This result will be sent to Fiware Context Broker.

There are also other methods we can follow to execute a face recognition in the camera connected to Raspberry Pi3 using a set of known images. The video stream can be sent to the cloud where the AI model can be executed to recognize faces from the stream sent to it from Raspberry Pi3. But in this case, the whole stream should be communicated to the cloud which results in data overhead and more connection problems related to sending over a large set of data. Minding the complications involved, we avoided this approach. The other method can be images clicked from the camera to be sent over to cloud for facial recognition to be executed. The main challenges can be data synchronization, data overhead, etc and still a complex way to implement. Our main aim is to avoid the above complexity and execute the processing part directly on the end device and only the results to be sent to Cloud.

1.1 Research Questions

R1: Which interfaces are needed to be implemented for a sensor on a Raspberry Pi3 to transmit data to Fiware cloud using the Fiware context broker mechanisms. How can these interfaces be described and implemented?

R2: Which interfaces are needed to be implemented for a Bonseyes AI artefact to be deployed in the Raspberry Pi3 and send the result to Fiware cloud using the Fiware control mechanism, e.g. the broker mechanisms. How can these interfaces be described and implemented?

R3: How are the performance of Raspberry Pi3 3 platform and Face recognition in it is affected by deploying a container image in the system in terms of CPU load, memory and execution time?

1.2 Main Contributions of this Thesis

In the BTH demonstrator of the Bonseyes system, our contribution is to design an interface between them,

- Raspberry Pi3 and the AI model – Configuring the Raspberry Pi3 to handle the complex AI model.
- AI artefact and the context broker of Fiware – The results of the AI model are sent and saved in the Fiware cloud.
- Docker and the AI artefact – Designing the AI artefact as well as the Docker file in a way that they compact with each other.
- Performance evaluation in terms of memory and CPU, with and without docker in Raspberry Pi3.

Using the above-designed interfaces we should be able to deploy and run a Dockerized AI model into the IoT end device (Raspberry Pi3).

1.3 Problem Statement/Motivation

AI applications that involve a lot of computing and sending certain results like images and text to end user must be simplified. This involves simplifying the computation speed, a large amount of processing, usage of resources and communicating the

necessary results including bandwidth through the device. Resources we use that is AI applications are developed and deployed in standard platforms (Raspberry Pi platform is not considered in many cases). For example, these AI applications are deployed only on iPhone and Android platforms. Computing and processing are done on these types of platforms for which data from many devices are sent resulting a lot of bandwidth. Edge Computing involving the deployment of complex AI application on edge device Raspberry Pi saves these kinds of bandwidth as processing is done on the edge device. So, the data sent to the cloud or any end user will only have the end result avoiding the whole information needed for the computation.

1.4 Outline of the Thesis

The rest of the thesis is structured as follows:

- Chapter 2 – System hardware and software description: The basic concepts and technologies (Internet of Things, Cloud Computing, Fiware, Artificial Intelligence, Deep learning, and its techniques, Virtualization, and Containerization, Demonstration) involved in the thesis.
- Chapter 3 – Implementation: Presents the setup of the system hardware and software (Raspberry Pi3, Camera Module (Pi camera), External hard-disk (Pi-Drive), Docker, AI artefact), Sysbench.
- Chapter 4 – Verification: Provides, how the verification of the thesis is done.
- Chapter 5 – Explains the results and analysis
- Finally, Chapter 6 – Conclusion and future work: Includes conclusion and the future works of the thesis.

2 RELATED WORK

This paper gave an insight of past, present, and future of the Internet of Things (IoT). [1]They also explain the diversity of the IoT application in various fields. They mention the challenges that are to be solved in IoT not only in technology but also in the business perspective. [2] Introduces the concepts and fundamentals of IoT. Explains the distinct modes of Communication Involved in IoT, that are between human to machine and machine to machine(M2M). The combination of embedded microcontrollers, sensors, actuators, network interfaces, and the greater Internet makes it possible for the Internet to evolve from a network of interconnected computers to a network of interconnected objects. Making use of generic enablers to implement a model that support IoT in communication, resources management, data handling, and process automation.

Sensors and the data from the sensors play a key role in the Internet of things. [3] This paper argues that the Cloud Computing model is a good fit with the dynamic computational requirements of environmental monitoring and modeling. They mainly focused on the integration of sensor networks and cloud computing and also demonstrating that cloud computing resources are flexible enough to deal with the unpredictable loads generated by real-world sensing applications. [4] The literature survey on the CloudIoT paradigm which involves completely new applications, challenges, and research issues. This paper also explains about the Fog computing and hoe it has been designed to support IoT applications characterized by latency constraints and the requirement for mobility and geo-distribution.

Present a survey of cloud computing, [5]highlighting its key concepts, architectural principles, state-of-the-art implementations as well as research challenges along with the various advantages of cloud computing like no up-front investment, high scalability, lowering operational cost, easy access, reducing business risks and maintenance expenses. [6] This paper presents challenges involved in cloud adaptation, various models such as Service model as in Software as a Service(SaaS), Platform as a Service(PaaS), Infrastructure as a Service(IaaS) an Data storage as a Service(DaaS), and Deployment model as in Private cloud, Community cloud, Public cloud, and Hybrid cloud.

[7]By considering the factors like architecture, and study their performance by comparing network latency, bandwidth occupation, energy consumption, and overhead, this paper

made a comprehensive survey in analyzing how edge computing improves the performance of IoT networks.

[8] This paper argues that semantic modeling takes up many of the challenges like Semantic interoperability, Data integration, Data abstraction, Data search, and discovery, Reasoning, and interpretation, and explain how the core platform developed by the FI-WARE project supports IoT application developers. the FI-WARE core platform and its semantics-based software components are called generic enablers (GE). Context Broker GE and its semantic extension enable the publication of context information by domain entities so that published information becomes available to other parties which are interested in processing this information. Applications or other enablers may publish or/and consume context information. [9]FI-WARE NGSI Context Management specifications are based in the NGSI Context Management specifications defined by OMA (Open Mobile Alliance). They take the form of a RESTful binding specification of the two interfaces defined in the OMA NGSI Context Management specifications, namely NGSI-9 and NGSI-10. NGSI defines two interfaces for exchanging information based on the information model. The interface OMA NGSI-10 is used for exchanging information about entities and their attribute, i.e., attribute values and metadata. The interface OMA NGSI-9 is used for availability information about entities and their attributes. Here, instead of exchanging attribute values, information about which provider can provide certain attribute values is exchanged. More details of the OMA NGSI-9 and 10 are explained in the paper [10].

This article [11] introduces the concept and status quo of the Deep learning (DL), reinforcement learning (RL) and their combination–deep reinforcement learning (DRL) methods, summarizes their potential for application in smart grids and provides an overview of the research work on their application in smart grids. This paper explains the difference between the traditional AI algorithms and the application-oriented DL, RL, DRL algorithms. This paper [12] briefly reviews the 60-year developmental history of AI, analyzes the external environment promoting the formation of AI 2.0 along with changes in goals and describes both the beginning of the technology and the core idea behind AI development. Furthermore, based on combined social demands and the information environment that exists in relation to Chinese development, suggestions on the development of AI 2.0 are given.

3 SYSTEM HARDWARE AND SOFTWARE DESCRIPTION

3.1 Internet of Things:

Internet of things: The term is first introduced by Kevin Ashton in 1998. A technology revolution that shows us the future of connectivity. The Internet of Things (IoT) things refers to any object or device that can communicate which can be found anywhere as software or hardware. It also can be a smart device which can be a part of the internet. These multiple numbers of things will be communicating over the internet or in an internal network through different types of means. In the internet of things, data collection, data storage, data processing, data sensing, and decision making is done based on the previous results. Data sensing is done using sensors, bar code labels, RFID tags, GPS, a camera which is considered as objects or things in the Internet of things. These collected data can be communicated through the internet or it can include a network management center or information processing center. Finally, the presentation of the information processed can be displayed in the form of a smart city, smart home, smart transportation, vehicle tracking, and many other applications. In this way, multiple application services can be provided, and they can be dependent on each other. Eventually, all the 'things' in IoT are communicating nodes connected to the internet.

The International Telecommunication Union (ITU) for instance now defines the Internet of Things as ''a global infrastructure for the Information Society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies[1]. In the context of discussions about IoT technologies, a frequently used concept is that of IoT platforms. In computing, the term ''platform'' itself is a broad concept, which has, e.g., been defined as ''a group of technologies that are used as a base upon which other applications, processes or technologies are developed''. Within the Internet of Things, IoT platforms are software products, which offer comprehensive sets of application-independent functionalities that can be utilized to build IoT applications.

Two distinct modes of communications are commonly described for the IoT[2]:

- Human-to-Object (Thing) Communication: humans communicate with a device in order to get specific information (e.g., IPTV content, file transfer) including remote access to objects by humans.

- Object-to-Object (Thing-to-Thing) Communication: an object delivers information (e.g., sensor-related information) to another object with or without the involvement of humans. As objects include physical devices and products as well as logical contents and resources, M2M communication is a subset of object-to-object communication.

In the IoT, things are objects of the physical world (physical things) or of the information world (virtual things), which are capable of being identified and integrated into information and communication networks. All of these things have their associated information, which can be static and dynamic [1].

- Physical things exist in the physical world and are capable of being sensed and/or actuated upon and/or connected. Examples of physical things include sensors of surrounding environments, industrial robots, goods, and electrical equipment.
- Virtual things exist in the information world and are capable of being stored, processed and accessed. Examples of virtual things include multimedia contents, application software and service representations of physical things.

IoT can benefit from the virtually unlimited capabilities and resources of Cloud to compensate for its technological constraints (e.g., storage, processing, communication). To cite a few examples, Cloud can offer an effective solution for IoT service management and composition as well as for implementing applications and services that exploit the things or the data produced by them [3]. On the other hand, Cloud can benefit from IoT by extending its scope to deal with real-world things in a more distributed and dynamic manner, and for delivering new services in a large number of real-life scenarios. In many cases, Cloud can provide the intermediate layer between the things and the applications, hiding all the complexity and functionalities necessary to implement the latter. This will impact future application development, where information gathering, processing, and transmission will generate new challenges, especially in a multi-cloud environment. [4]

Generally, the huge amount of data produced through IoT devices must have rental storage space for data storage. Due to the huge size of data, we also need more processing which is said that it cannot be possible at the end device i.e. IOT end devices which are low cost and lightweight. Therefore, in most of the scenarios data storage and processing are not done at the end device but in our project, The Artificial Intelligence model of face recognition is virtualized

for the purpose of avoiding complexity and is deployed at the End Device. The Processing is done at the End Device and the result storage is done in the Cloud.

Face Recognition in Raspberry Pi3 (IoT End device): Instead of streaming the video to Cloud and do face recognition there. We have made things easy by reducing the processing in the cloud and by just sending the result to Fiware Cloud. Video streaming through the Internet to Cloud can be avoided thus avoiding complexity.

3.2 Cloud Computing:

Cloud computing can be integrated with the Internet of things. Internet of things generate a large amount of data and the requirements of managing those data and securing them are increasing. IoT and cloud computing helps in developing smart applications for users. In Cloud Computing, User need not worry about managing the information from resources and to maintain the system.

What is Cloud Computing? In Cloud Computing, A network of remote servers are hosted on the internet. Processing, storing and managing of the dataset can be done on these servers.
In our Thesis, we focus on Edge Computing which is nothing but cloud computing systems of an application, its data or services to the edge of the internet. In our case, the edge of the internet is the IoT end device which is Raspberry Pi3 where the AI system is implemented (a part of the service) and send the result to Cloud.

In a cloud computing environment, the traditional role of the service provider has divided into two: the infrastructure providers who manage cloud platforms and lease resources according to a usage-based pricing model, and service providers, who rent resources from one or many infrastructure providers to serve the end users. The emergence of cloud computing has made a tremendous impact on the Information Technology (IT) industry over the past few years, where large companies such as Google, Amazon, and Microsoft strive to provide more powerful, reliable and cost-efficient cloud platforms, and business enterprises seek to reshape their business models to gain benefit from this new paradigm. Resources in a cloud environment can be rapidly allocated and de-allocated on demand.

Cloud computing is often compared to the following technologies, each of which shares certain aspects with cloud computing:

Grid Computing: Grid computing is a distributed computing paradigm that coordinates networked resources to achieve a common computational objective. The development of Grid computing was originally driven by scientific applications which are usually computation intensive. Cloud computing is like Grid computing in that it also employs distributed resources to achieve application-level objectives. However, cloud computing takes one step further by leveraging virtualization technologies at multiple levels (hardware and application platform) to realize resource sharing and dynamic resource provisioning.

Utility Computing: Utility computing represents the model of providing resources on-demand and charging customers based on usage rather than a flat rate. Cloud computing can be perceived as a realization of utility computing. It adopts a utility-based pricing scheme entirely for economic reasons. With on-demand resource provisioning and utility-based pricing, service providers can truly maximize resource utilization and minimize their operating costs. Virtualization: Virtualization is a technology that abstracts away the details of physical hardware and provides virtualized resources for high-level applications. A virtualized server is commonly called a virtual machine (VM). Virtualization forms the foundation of cloud computing, as it provides the capability of pooling computing resources from clusters of servers and dynamically assigning or reassigning virtual resources to applications on-demand.

Autonomic Computing: Originally coined by IBM in 2001, autonomic computing aims at building computing systems capable of self-management, i.e. reacting to internal and external observations without human intervention. The goal of autonomic computing is to overcome the management complexity of today's computer systems. Although cloud computing exhibits certain autonomic features such as automatic resource provisioning, its objective is to lower the resource cost rather than to reduce system complexity.

Cloud computing employs a service-driven business model. In other words, hardware and platform-level resources are provided as services on an on-demand basis. Conceptually, every layer of the architecture described in the previous section can be implemented as a service to the layer above. Conversely, every layer can be perceived as a customer of the layer below. However, in practice, clouds offer services that can be grouped into three categories: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). [5]

1. Infrastructure as a Service: IaaS refers to on-demand provisioning of infrastructural resources, usually in terms of VMs. The cloud owner who offers IaaS is called an IaaS provider.
2. Platform as a Service: PaaS refers to providing platform layer resources, including operating system support and software development frameworks.
3. Software as a Service: SaaS refers to providing on-demand applications over the Internet.

3.2.1 Platform as a Service (PaaS):

PaaS is a development platform supporting the full "Software Lifecycle" which allows cloud consumers to develop cloud services and applications (e.g. SaaS) directly on the PaaS cloud. Hence the difference between SaaS and PaaS is that SaaS only hosts completed cloud applications whereas PaaS offers a development platform that hosts both completed and in-progress cloud applications. This requires PaaS, in addition to supporting application hosting environment, to possess development infrastructure including programming environment, tools, configuration management, and so forth. An example of PaaS is Google AppEngine. [6]

Among these three services, PaaS technical layer plays a key role throughout the whole system of cloud computing. Although IaaS provides users with scalable computing and storage capacity with the manner of virtualization, it merely plays the role of infrastructure, just like the hardware facilities in the traditional applications. Correspondingly, PaaS, as the system software role, can provide rich APIs for the upper layer, so as to develop various SaaS applications. However, the goal of PaaS is not only like traditional system software to provide some basic APIs, but more advanced service-oriented APIs. Therefore, the upper application can use these advanced services for end-users to build a specific application easily. Then, we can conclude the basic concept of PaaS: PaaS is to a particular Internet resource center, in the form of an open platform, to provide Web API service outside. So, PaaS has the following three significant characteristics:[13]

1. Internet resource center:

   Nowadays, in the Internet industry, the competition is no longer a purely technical one but turned to the Internet resources, which include users, searching content, business information, networking groups and online services, and so on. To some extent, the PaaS platform service aims to provide Internet resources in the form of external value-added services. As a matter of fact, the PaaS platform is born to provide Internet resources, which wouldn't have any vitality for the lack of Internet resources.

2. Open platform:

   Open is the essence of the Internet, and PaaS is a concrete manifestation of this spirit. Earlier years, Internet sources were offered to the ultimate customer's applications, scarcely open to the third-party developers. When these resources became more and more, and the value increases, the enterprises who own the resources were no longer to develop more value-added applications.

3. Web API:

   PaaS platform is embodied in the form of Web API. Except for the basic management function in the background, not like the application software, users can not see any graphics interfaces. PaaS platform manifests the power of its platform service entirely through calling Web API among networks. Web APIs are for the third-party developers, and end-users can't see and don't care.

3.2.2 Infrastructure as a Service (IaaS).

Cloud consumers directly use IT infrastructures (processing, storage, networks, and other fundamental computing resources) provided in the IaaS cloud. Virtualization is extensively used in IaaS cloud in order to integrate/decompose physical resources in an ad-hoc manner to meet growing or shrinking resource demand from cloud consumers. The basic strategy of virtualization is to set up independent virtual machines (VM) that are isolated from both the underlying hardware and other VMs. Notice that this strategy is different from the multi-tenancy model, which aims to transform the application software architecture so that multiple instances (from multiple cloud consumers) can run on a single application (i.e. the same logic machine). An example of IaaS is Amazon's EC2. [6]

IaaS cloud platform resource can be divided into two parts: physical machine resources and virtual machine resources, and about how much the energy consumption cost is directly reflected from the status of the physical machine resource consumption, that is to say, the resource is the container of virtual machine resource. The physical machine can be divided into three statuses: overload, under load and normal, which are not the sequence that can be observed, but a hidden status sequence is known as the implicit status set.[14]

There are various types in the proposed abstraction models of IaaS. Resource type includes compute, storage, network, and virtual machine, which are a composite of various resources;

Reflection type is mainly for backup and disaster recovery, which includes an image for computing and snapshot for storage; other types support security purposes, including credential and firewall. Virtual Machine is a particular type of resource which is a collection of individual resources, and this kind of modeling is generic in the sense that it supports modeling of virtual machines composed of multiple computers, multiple storages and networks, and even multiple virtual machines. A collection of virtual machines known as virtual clusters can also be modeled in this approach. [15]

3.2.3 Software as a Service (SaaS)

Cloud consumers release their applications on a hosting environment, which can be accessed through networks from various clients (e.g. web browser, PDA, etc.) by application users. Cloud consumers do not have control over the Cloud infrastructure that often employs a multi-tenancy system architecture, namely, different cloud consumers' applications are organized in a single logical environment on the SaaS cloud to achieve economies of scale and optimization in terms of speed, security, availability, disaster recovery, and maintenance. Examples of SaaS include SalesForce.com, Google Mail, Google Docs, and so forth. [6]

Aside from enhancing a hosted application with many capabilities, SaaS platforms generally take on a larger set of responsibilities. [16]

1. Tenancy partitioning: For example, the application database may be geared for catering to a single tenant (user), preventing multiple users from storing data in the same database. Furthermore, execution may not be partitioned so situations may arise where inadvertent state sharing occurs between tenants that otherwise should have been isolated from one another. SaaS platforms are responsible for providing multi-tenancy aspects when needed.
2. Scaling: SaaS applications aggregate demand for all customers and users to one physical or virtual location purposefully. To achieve this sort of scaling, the application must be designed in a fashion conducive to scaling as well as having the support required of any auxiliary pieces. SaaS platforms try to remove the work associated with scaling by commoditizing the scalability portion.
3. Monitoring & Metering: A SaaS application needs metering and monitoring usage from both data and execution standpoints. Much of this is extracted into a platform layer,

where the platform becomes responsible for metering tenant and user usage, as well as monitoring for all system events. The monetization platform should be built a set of integrated software modules which meter usage of on-demand applications without the need for custom coding.

4. Distributed Services: Traditional on-premise software relies on a variety of services to accomplish the task of executing business logic. These on-premise services, such as logging or job scheduling, tend to become overly complex if used in a distributed system that generally backs most SaaS applications. SaaS platforms provide distributed equivalents of many such services, giving similar or even augmented capability when comparing to on-premise counterparts.

Cloud computing refers to both the applications delivered as services over the internet and the hardware and systems software in the data centers that provide those services. The data center hardware and software is what we will call a cloud. When a cloud is made available in a pay-as-you-go manner to the general public, we call it a public cloud; the service being sold is utility computing. The private cloud is to refer to internal data centers of a business or other organization, not made available to the general public when they are large enough to benefit from the advantages of cloud computing.

Hybrid clouds: A hybrid cloud is a combination of public and private cloud models that tries to address the limitations of each approach. In a hybrid cloud, part of the service infrastructure runs in private clouds while the remaining part runs in public clouds. Hybrid clouds offer more flexibility than both public and private clouds.

Virtual Private Cloud: An alternative solution to addressing the limitations of both public and private clouds is called Virtual Private Cloud (VPC). A VPC is essentially a platform running on top of public clouds. [17]

3.3 Edge Computing:

[18]Edge computing is the phenomenon of enabling technologies to perform computation at the network edge so that computing happens near the data source. It works on both downstream data on behalf of cloud services and upstream data on behalf of IoT services. An edge device is any computing or networking resource residing between data sources and cloud-based datacentres. In our case, the edge device is the Raspberry Pi3.In edge computing, the end device not only consumes data but also produces data. And at the network edge, devices

not only request services and information from the cloud but also handle computing tasks—including processing, storage, caching, and load balancing on data sent to and from the cloud.

3.3.1 Edge Computing over Cloud Computing:

[7]Edge computing has emerged as a new paradigm to solve IoT and localized computing needs. IoT devices usually do not require much computational capacity, and the demands of IoT can be properly satisfied, especially for real-time services, by edge nodes. In addition, edge nodes mitigate the power consumption of the IoT devices through the offloading of computation tasks. Compared with the well-known cloud computing, edge computing will migrate data computation or storage to the network ‘‘edge,’’ near the end users. Even though the edge computing servers have less computation power than the cloud servers, they still provide better QoS (Quality of Service) and lower latency to the end users.

[18]Edge computing could yield many benefits. For example, researchers have shown that using cloudlets to offload computing tasks for wearable cognitive-assistance systems improves response times by between 80 and 200 ms4 and reduces energy consumption by 30 to 40 percent. [19]Clone Cloud technology reduces response times and power usage by 95 percent for tested applications, in part via edge computing. edge computing better protects data because the processing occurs closer to the source than in cloud computing. The scalability of edge computing can be scaled from a single person to smart home to even an entire city.

[7]Thus, edge computing can reduce the traffic flows to diminish the bandwidth requirements in IoT. Furthermore, edge computing can reduce the transmission latency between the edge/cloudlet servers and the end users, resulting in shorter response time for the real-time IoT applications compared with the traditional cloud services.

3.4 Future Internet- FIWARE:

Future internet Fiware: We have used Fiware here to store the result information after processing at Raspberry Pi3 through Context Broker using NGSI API. Fiware is an open source platform for our smart digital future.

Fiware is advantageous to developers, Service providers, and public or private organizations. Innovative projects and Ideas can be created and supported by its free open source platform nature improving competitiveness and quality of life. Some of the advantages of Fiware are: Easy to implement, Ensure interoperability, Easily scalable, Open API Standards. Fiware API standards provide cloud computing enhanced capabilities hosting OpenStack and an extensive library of components with value-added features and services that include a set of tools and libraries called "Generic Enablers", which provide open standards API. Generic Enablers provide capabilities which help to connect to IoT devices, data processing, and real-time media on a large scale.

It makes IOT simpler and enables the data economy. Fiware provides several components to manage the Context data within a simple smart solution. This Context data can be manipulated and managed in Fiware. One of the Fiware Component (Generic Enabler) used in our project is the Orion Context broker. It relies on open source MongoDB technology. The Entities can be created in the Orion Context broker and can be managed using API requests using curl or Postman. Querying entity information, creating new Entity in Open Source Context Broker using NGSI API is done to store the information generated by the Internet of things.

Future internet applications require users and things, including our daily life objects, to be interconnected, at any time (on the move, wirelessly and seamlessly) and through high-speed broadband connections if necessary. Information and data exchanged through these connections are of diverse nature and include context data as well as devices, things, and services descriptions. For instance, this will lead to the need for correlating streams of heterogeneous data produced and consumed by these interconnected entities and poses salient technological challenges in terms of data integration, management, modeling, and processing. Fiware provides over 50 plus Generic Enablers (GE).

In our case, we used the GE named Context Broker,[8]
Context Broker:
This component enables the publication of context information by domain entities so that published information becomes available to other parties which are interested in processing this information. Applications or other enablers may publish or/and consume context information.
It manages all the context information that the application needs to work, brokering between the elements that produce that information (IoT Backend Device Management GE, Application

Mashup GE, CEP GE, Location GE, Ticket Management Module) and the elements that consume it (Application Mashup GE, CEP GE, Ticket Management Module, History Module). In some sense, it is the dorsal spine of the data flow in the application. It is worth noting that Context Broker GE manages the current status of the system, relaying in other components to achieve persistent historic storage (BigData GE and History Module).

The result, that is the name of the recognized face is stored in the Context Broker using the following process: An Entity with a certain ID linked to a specific IOT end device is created in Orion Context Broker using NGSI API. When a face is recognized at that specific IOT end device, the result is sent to that entity which has been already created in Orion Context broker in Fiware using POST request automatically in Artificial Intelligence Model.

3.5 Artificial Intelligence:

Artificial Intelligence: The tasks performed by a machine with human intelligence is Artificial Intelligence.

It is a comprehensive science and technology, including a huge domain in breadth and width. After decades of development, much progress has been made across the fields of artificial intelligence (AI). In recent years, driven by the increasing amounts of data, there has been a significant emergence in the advancement of AI algorithms and powerful computer hardware, allowing AI to enter into a new evolutionary stage: AI 2.0. AI 2.0 related technologies are currently in the process of development, and many algorithms are emerging. In their present states, deep learning (DL) and reinforcement learning (RL) is relatively mature; and their combination-deep reinforcement learning (DRL) has won some success with AlphaGo, a game-playing program developed by Google DeepMind. In March 2016, AlphaGo defeated the world Go champion, Sedol Lee, with a 4:1 score, and thereby attracted a new wave of global attention.[11]

The development of AI is driven both by research and by the information environment, with its accompanying social goals. Although both are very important, the latter always has a stronger driving force. With the current popularization of the Internet, universal existence of sensors, emergence of big data, development of e-commerce, rise of the information community, and interconnection and fusion of data and knowledge in society, physical space, and cyberspace, the information environment surrounding AI development has changed profoundly, leading to

a new evolutionary stage: AI 2.0. The emergence of new technologies also promotes AI to a new stage.[12]

The success of AI has come so far exploring different areas and bringing them together. For example, AI has been dealing with a large amount of data (Big Data), scalable computer and software systems and the broad accessibility of these technologies.
AI can provide a wide range of application services or any other type of services like health care, transportation, security, etc. In our case, AI has been used for facial recognition at the end device in IOT which can provide the purpose of security for any type of requirement. As for example, when opening a lock of the door of a car or a house or to keep track of the persons entering a building etc.

There are also different types of challenged in AI-

- Mission Critical AI: AI should be trained according to the changing environment where human lives can be at stake. AI should be adaptable if environmental conditions change. As for examples in an automatic vehicle, AI-powered intrusion detection system should be adaptable for any upcoming dangerous situations and updated according to the changing environment.

- Security in AI: If AI depends on the accent of the owner, then the security is weak. AI can be compromised with the person who can always learn the accent of the owner or talk in the same voice as the owner. Therefore, the parameters that connect the AI and the owner of the AI should be strong in terms of security. During training or provide decision-making services, AI should make sure that the data resources are not leaking any information.

However, there are many challenges to overcome for an AI to be a positive force in our day to day lives. AI systems will need to make decisions that are faster, safer and more explainable, securing these decisions as well as the learning processes against many types of attacks. These also should be easy to integrate into existing applications and Cloud.

3.6 Deep Learning (DL):

Deep learning is nothing, but a part of machine learning methods based on learning data representations. Enables the computer to recognize patterns and solve the recognition problem. Learning efficiently from data set is crucial for efficient architectures to be designed

for a wide range of applications. The main driver of deep learning is GPUs which are used to design DNN deep neural network.
Bonseyes Deep Learning Toolbox will provide a unified framework for accelerating deep convolution neural networks on resource-constrained architecture-dependent embedded systems. The massive data which is used for deep learning has security issues for sure. Generally, the best way to protect is to confine it to the device so that sensor data never leaves it. This approach requires training of networks or adaptation of networks in the device.

DL [11] is a subset of machine learning, which originally resulted from a multi-layer Artificial Neural Network (ANN). Strictly speaking, DL has a wider meaning, but in its present state, when talking about DL, we just think of a large deep neural network, that is deep neural networks. Here, deep typically refers to the number of layers. There are different structures of DL: Boltzmann Machine (BM), Deep Belief Networks (DBN), Feedforward Deep Networks (FDN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Long-short Term Memory (LSTM) Networks, Generative Adversarial Networks (GAN).

Among them, CNN and RNN are the most popular structures. CNN is suitable for dealing with spatial distribution data while RNN has advantages in managing time series data. DL frameworks are basic underlying architectures, consisting of popular ANN algorithms, providing a stable DL API (Application Program Interface), supporting distributed learning of training models on GPU, TPU, and playing an important role in popularizing DL applications. Mainstream open source DL frameworks include TensorFlow, Caffe/Caffe2, CNTK, MXNet, Paddle-paddle, Torch/PyTorch, Thenano, etc. Presently, DL has shown its extraordinary ability in many areas, such as image and speech recognition and natural language processing.

3.6.1 Examples for Training Concepts of Deep learning techniques:

3.6.1.1 Bonseyes:

Bonseyes Collaborative project aims to develop a platform consisting of Data Marketplace, a Deep Learning Toolbox, and Developer Reference Platforms for organizations wanting to adopt Artificial intelligence. The Main focus of this Bonseyes has made the main priority in our thesis which is AI in Low power Internet of things (IoT) devices (“edge computing”) and Cloud

computing. Therefore, the AI is implemented in End device in our thesis and interconnected with the cloud to save the results.

Bonseyes is solving the Monolithic development of System of AI give rise to the Datawall effect that only large companies with end to end solutions can pass. Bonseyes mainly concentrates on the target platforms like Data Marketplace, Deep Learning Toolbox.
Key objectives of Bonseyes Deep learning toolbox are:

- Accelerate the design and programming of systems of Artificial Intelligence
- Reduce the complexity in training deep learning models for distributed embedded systems
- Foster "embedded intelligence" on low power and resource constrained smart CPS-Cyber-Physical systems.
- Demonstrate the whole concept in key challenging scenarios.

Training Concepts in Bonseyes are flexible in terms of representation: a number of bits precision, hashing methods, parameter sharing structure, memory usage, sparsity, and robustness. Cost-sensitive optimization methods are used that can optimize a particular deep learning architecture for a particular embedded environment by incorporating particular individual costs.
Component valuation: The value of a given deep learning component in an environment will be an integral part of the component itself.
Generally, generic low-level linear algebra libraries are used for implementing deep neural networks. A specific effort has to be done when developing a specific part of deep learning methods for a specific environment.

3.6.1.2 Google play:

Google play recommender system recommends relevant Android Apps to plat app users when they visit the homepage of the store. When a user queries an Application in the Google Play store, the first step is the system retrieves a short list of apps based on various signals. Next, the ranking system uses the machine-learned model to compute a score per app and presents a ranked list to the user. So, here the training concept is based on how the ranking of the items is trained which will be continued according to the fresh data.

3.7 Virtualization and Containerization:

Virtualization: It is the act of creating virtual of something like hardware, software or storage devices.

Containerization: Container word is derived from shipping containers, a standard method to store and ship any kind of cargo. Some of the popular containers are Linux, Warden, Docker, Rocket. Container-based virtualization is consistently developing for various systems for light-weight processing and deployment. This can make IoT management easy as it provides isolation between multiple execution environments of containers and has high performance, resource efficiency, and agile environment.

A container can put together multiple application components with the use of image layering and extension process. An application container engine is used to run images and a repository is used to store and manage private and public container images. Containers can be deployed and can dynamically be added, removed, started and stopped depending on the needs of the overall systems.

In our thesis, Container is directly installed in the IoT device by having a suitable OS working on it.

So, for the container to work on Raspberry Pi3, it should be built using the Base Image of OS that supports the ARM architecture of the Raspberry Pi3.

Both containers and VMs are virtualization tools. On the VM side, a hypervisor makes siloed slices of hardware available. There are generally two types of hypervisors: “Type 1” runs directly on the bare metal of the hardware, while “Type 2” runs as an additional layer of software within a guest OS. While the open-source Xen and VMware's ESX are examples of Type 1 hypervisors, examples of Type 2 include Oracle's open-source VirtualBox and VMware Server.

Containers, in contrast, make available protected portions of the operating system—they effectively virtualize the operating system. Two containers running on the same operating system don't know that they are sharing resources because each has its own abstracted networking layer, processes and so on.

Docker harnesses some powerful kernel-level technology and puts it at our fingertips. The concept of a container in virtualization has been around for several years, but by providing a simple toolset and a unified API for managing some kernel-level technologies, such as LXCs

(Linux Containers), cgroups and a copy-on-write filesystem, Docker has created a tool that is greater than the sum of its parts.

Docker provides tools to make creating and working with containers as easy as possible. Containers sandbox processes from each other. For now, you can think of a container as a lightweight equivalent of a virtual machine.

Linux Containers and LXC, a user-space control package for Linux Containers, constitute the core of Docker. LXC uses kernel-level namespaces to isolate the container from the host. The user namespace separates the container's and the host's user database, thus ensuring that the container's root user does not have root privileges on the host. The process namespace is responsible for displaying and managing only processes running in the container, not the host. And, the network namespace provides the container with its own network device and virtual IP address.

Another component of Docker provided by LXC is Control Groups (cgroups). While namespaces are responsible for isolation between host and container, control groups implement resource accounting and limiting. While allowing Docker to limit the resources being consumed by a container, such as memory, disk space, and I/O, cgroups also output lots of metrics about these resources. These metrics allow Docker to monitor the resource consumption of the various processes within the containers and make sure that each gets only its fair share of the available resources.

In addition to the above components, Docker has been using AuFS (Advanced Multi-Layered Unification Filesystem) as a filesystem for containers. AuFS is a layered filesystem that can transparently overlay one or more existing filesystems. When a process needs to modify a file, AuFS creates a copy of that file. AuFS is capable of merging multiple layers into a single representation of a filesystem. This process is called copy-on-write.

Traditionally, Docker has depended on AuFS to provide a copy-on-write storage mechanism. However, the recent addition of a storage driver API is likely to lessen that dependence. Initially, there are three storage drivers available: AuFS, VFS, and Device-Mapper, which is the result of a collaboration with Red Hat.

As of version 0.7, Docker works with all Linux distributions. However, it does not work with most non-Linux operating systems, such as Windows and OS X. The recommended way of using Docker on those OSes is to provision a virtual machine on VirtualBox using Vagrant [20].

Therefore, The AI artefact is virtualized and managed by a docker container which supports the Raspberry Pi3 3.

3.8 Demonstrator

We are demonstrating our thesis using a model to show the procedure involved in explaining our research questions using AI applications in low-power devices like Raspberry Pi3. Containerizing the AI application is also a crucial part illustrating the feasibility of AI in Container and in Raspberry Pi3. In this system, our focus is clear on what to contribute and its capability. It is an implementation where hardware and software are running in the low-power device, Raspberry Pi3 and the integrations between the components. It can show and execute specific functions. It has helped us developer/design to understand the difficulties of implementing the systems.

Main components include:
IOT environment: A camera or a temperature sensor is connected to Raspberry Pi3; Raspberry Pi3 executes a program in it and updates sensor information or a result from the camera to Fiware. The result from the camera includes a recognized face obtained from implementing a Dockerized AI model.
FIWARE: It provides components which help us build the demonstrator. One of the main components of Fiware is Context Broker comes with NGSI API capabilities. It contributes to a certain extent in the thesis project.
Bonseyes AI artefact: AI artefact used is a face recognition model virtualized through docker and deployed in IoT device which is the end device Raspberry Pi3.

Designing Interfaces:
The interface between Raspberry Pi3 and sensor or camera is created while designing the setup of the IOT environment.
An AI artefact in docker is deployed in Raspberry Pi3 and executed to recognize a face from the camera connected to Raspberry Pi3. The result from this AI artefact is sent to Fiware context broker through NGSI API provided by it. Thus, creating an Interface between AI artefact in Raspberry Pi3 and Fiware.

Implementing the Demonstrator:
Main Aim here is to provide a demonstration using the implementation of the interfaces.

The execution of the system in the end device Raspberry Pi3 instead of the cloud is the main priority. The developed system will be demonstrated in a clear way.

Building the interfaces between the main components of this Thesis and establishing communication between them is the way of showing the execution of the system. This can be explained in a more detailed way by the following steps:

Interfaces between Sensor and Raspberry Pi3, AI artefact and Raspberry Pi3, AI artefact and Fiware are built. Different layers involved in Creating an AI docker image is as shown in Figure [1].

AI artefact has to be dockerized and deployed in Raspberry Pi3, after the Deployment, AI artefact is executed using the interface between the camera and the Raspberry Pi3. The result from the AI artefact is sent to Fiware using the interface between them.

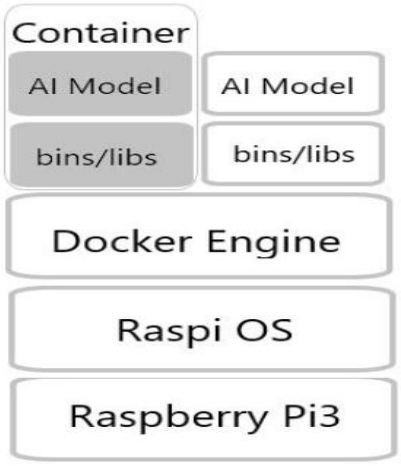

Figure 1: Demonstrator Model

3.9 Raspberry Pi3:

The Raspberry Pi3 (RPi3) is a single-board computer (SBC). It is also called a credit card sized computer. It is called so because of its dimensions (86.9mm x 58.5mm x 19.1mm). It plays a decent role in the development of the Internet of Things (IoT) projects. Nowadays RPi3 is being used as an experimental tool for conducting IoT projects. The device's affordability and easy to use nature makes it reachable to all. There are several applications of Raspberry Pi3, can be used as a Desktop PC, Media center, Retro Gaming machine, Robot controller, Face-Recognition device, and many more.

The official Operating system used in every Raspberry Pi3 is Raspbian (a Debian Linux OS). By using NOOBS (New Out Of the Box Software), an OS installer we can manage to install

and run some third party Operating System like Ubuntu Mate, Snappy Ubuntu core, Windows 10 IoT core, OSMC, PINET, RISC and few more. We used Raspbian Lite and Raspbian Pixel operating systems in our Raspberry Pi3.

Specifications: The RPi version we used in our project is Raspberry Pi3, model B Figure[2] which comes with 64-bit ARMv8 Quad-core processor (1.2 GHz), 1GB RAM, BCM43143 WiFi on board, Bluetooth low energy (BLE) 4.1, 40-pin (2 × 20) 2.54mm GPIO connection, 4x USB 2.0 ports, a HDMI connection, a CSI port (for Raspberry Pi3 camera), a DSI port (Display connection) , 10/100 Ethernet port, MicroSD port to load your operating system and save data, and a 2.5 A micro USB power input.

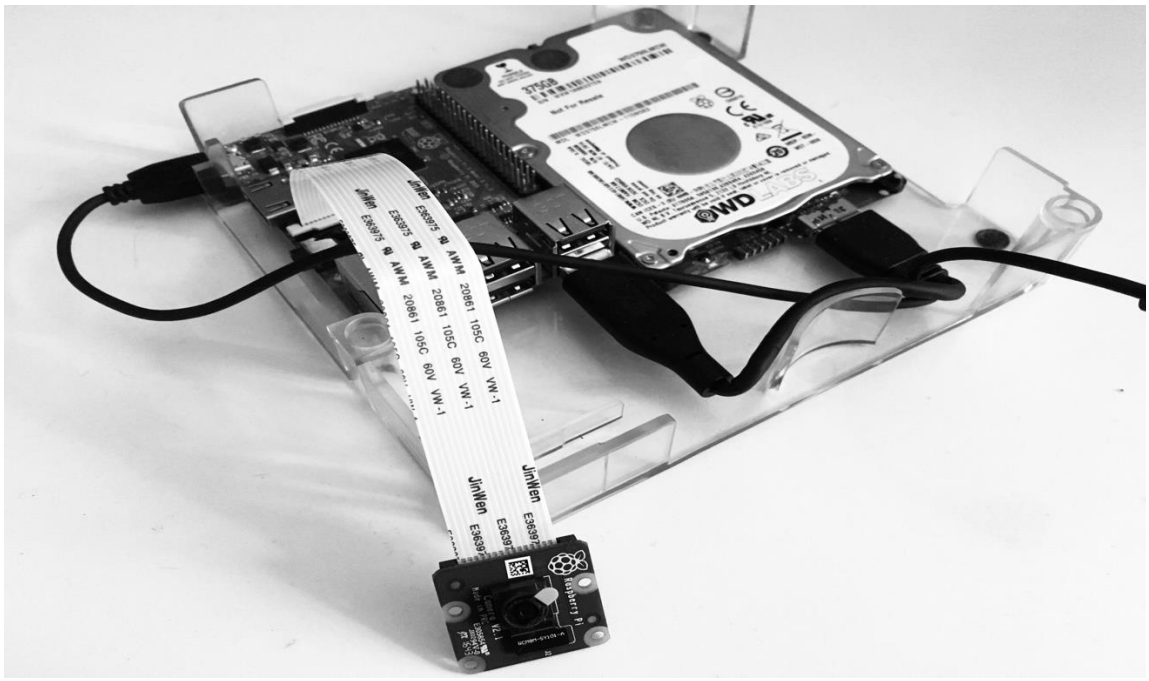

Figure 2: Raspberry Pi3

3.10 Camera Module (RPi camera):

The camera module we used in our project is the Raspberry Pi3 camera v1 module FIgure[3]. It is an official product of Raspberry Pi3 which was released in the year 2013. The resolution of this camera allows us to capture 5 mega-pixels images or 1080p30fps HD video recording. It can be easily connected to the CSI (Camera Serial interface) connector port of Raspberry Pi3. The CSI bus is extremely capable of high data rates. The camera also comes with Sensor- OmniVision OV5647. The picture can be captured in jpg, bmp, gif, png. Saving

the image in jpg format is preferable than in any other format because it takes less time to save the picture.

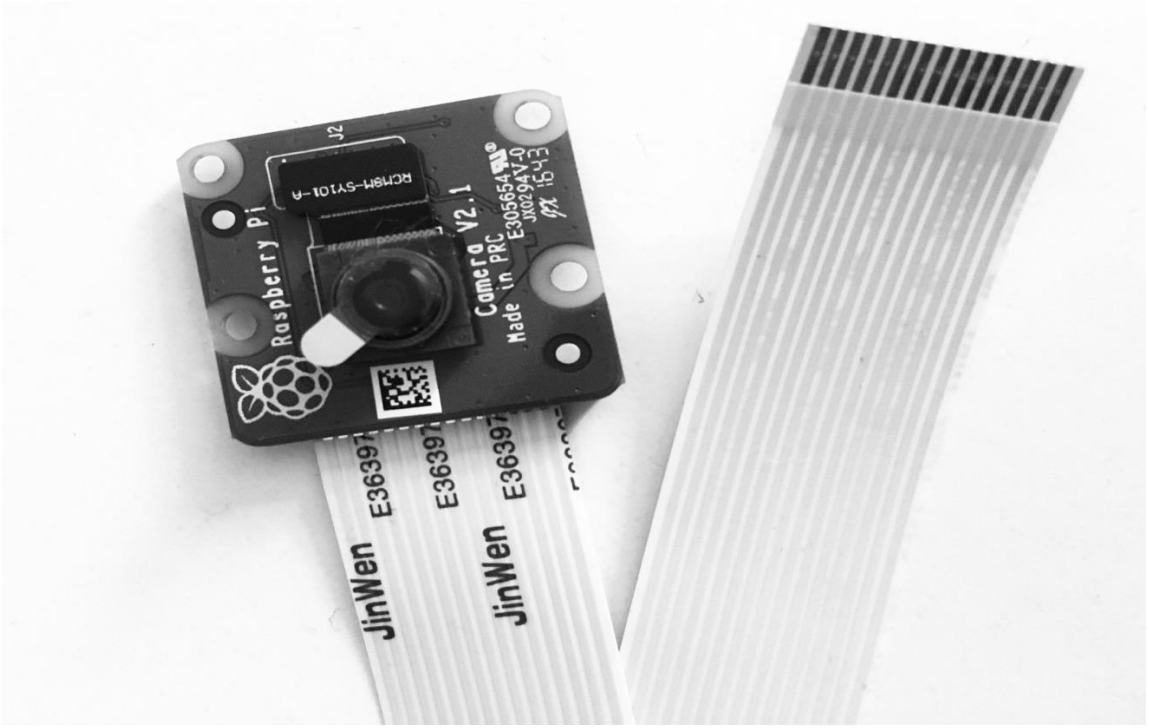

Figure 3: Raspi Camera

3.11 External Hard drive (RPi drive):

One of the main constraints of Raspberry Pi3 is its memory. This problem can be solved by mounting an external USB hard drive. But mounting an external hard drive comes up with a new problem that is power consumption. Hard drives consume lots of power, which the power adapter of Raspberry Pi3 isn't efficient enough to run both Raspberry Pi3 and the hard-drive and hence we need a hard-drive with power input. To avoid this mess, we used a WD PiDrive foundation edition, Hard drive. In order to reduce the power load of the drive on the Raspberry Pi3, this device uses the technology of custom magnetic recording and electrical system design. The WD PiDrive cable which comes along with the hard disk is designed in a way to supply the adequate power supply to both the Raspberry Pi3 and the PiDrive. In addition, a microSD is included with the device which has a preloaded custom version of NOOBS.

3.12 Container:

Containers is a highly suitable technology for application packaging and management in edge clouds that are more flexible and lightweight than VMs. Through image layering and extension process, containers can compact several application components.
A container solution consists of two main components –
(i) an application container engine to run images and
(ii) a repository/registry that is operated via push and pulls operations to transfer images to and from host-based container engines.
Container repositories play a central role in providing possibly reusable private and public container images. The container API supports life-cycle operations like creating, composing, distributing containers, starting and running commands in images. Containers are created by assembling them from individual images, possibly extracted from the repositories [21]. Containers are connected in a network and communicate via TCP/IP.

The process of building an application using a container is known as containerization. Here in our project, we built an AI artefact using docker container technology. unlike virtual machines (VM's) the containers use only the required resources to run an application

3.12.1 Docker Container:

Docker container is one of the popular container technologies. The reasons why we chose docker container are because of its agile operations to build, manage and secure our applications, freedom of choice in Linux based OS's, architecture, infrastructure, development language, and any application type. The feature orchestration in docker is responsible for the deployment of containers in an assigned manner when needed. We can run multiple containers simultaneously in docker. Docker containers run locally on Linux.

Containers in docker are launched by running an 'image'. Image contains all the prerequisites to run an application smoothly. To create an image, we use to build command. Almost any application can be run securely and separately in Docker container which allows us to run multiple containers at a time. Docker follows client-server architecture. The docker daemon is responsible for building, running and distributing the container. The communication between client and daemon happens through REST API or through the network interface. Every docker

image is stored in the registry of the Docker. We can pull or push an image from or to the docker registry using the commands *'`docker pull`'* or *'`docker push`'*. Using the command *'`docker run`'* we can run the image.

In Docker, an image is built by following the instructions given in a text document that is the '*Dockerfile'*. When the 'docker build' command is used the image is built from the '*Dockerfile'*. The Docker daemon runs the build operation, instructions in the '*Dockefile'* one-by-one. Every instruction in the file is run independently. To speed up the build process, docker will reuse the intermediate image (cache). The instructions in the '*Dockerfile'* is not case-sensitive. Every '*Dockerfile'* has to start with '*FROM'* instruction. The '*FROM'* instruction states the base image from which we are building.

3.13 Fiware

Fiware provides cloud hosting services supported OpenStack technology and a group of elements providing a variety of added-value functions "as a service", the generic enablers: they're open commonplace APIs that make it easier to connect to the internet of things, process data and media in real-time at large scale, perform BigData analysis or incorporate advanced features to interact with the users, thus FIWARE may be thought-about as an open various to existing proprietary net platforms. Some of the services are cosmos, Wirecloud, Kurento, Orion.Fiware Generic Enablers (GE) offer a number of general-purpose functions, offered through well-defined APIs, easing development of smart applications in multiple sectors. Fiware catalog contains a rich library of components (Generic Enablers) with reference implementations that allow developers to put into effect functionalities such as the connection to IoT, making programming much easier. Fiware generic enablers have crossed different phases of a path of development aimed at creating and test the technologies. We particularly use Orion service provided by Fiware. It is a context broker which handles data about context at large scale. Orion Context broker is an implementation in C++ and of Publish/Subscribe Context Broker GE, providing the NGSI9 and NGSI10 interfaces. It is developed as a part of the Fiware platform. Using these interfaces, several operations can be done by clients. It allows you to manage the entire lifecycle of context information including updates, queries, registrations, and subscriptions. Context elements are created and managed through updates and queries. In addition, you can subscribe to context information so when some condition occurs (e.g. the context elements have changed) you receive a notification.

Apart from Orion Context Broker, there are other related components such as Cygnus which implements a connector for context data coming from Orion Context Broker and aimed to be stored in specific persistent storage, such as HDFS, CKAN or MySQL [22]. Considering Raspberry Pi3 and Sensor (temperature sensor or a camera) as an IoT platform, our AIM is implementing a demonstrator for sending a Context Information from Raspberry Pi3 to Fiware Context Broker through NGSI- based programming model. NGSI- based Programming model- It is a standards-based approach to enable easy programming of IoT services over cloud and edges.

The Fiware NGSI Context Management specifications are based in the NGSI Context Management specifications defined by OMA (Open Mobile Alliance). They take the form of a RESTful binding specification of the two interfaces defined in the OMA NGSI Context Management Specifications, namely NGSI9 and NGSI10[9]. Basic NGSI Context Management Information Model includes entities, Attributes, Attribute Domains and Context Elements.

A context element contains the following information,
An entity id and type, a list of triplets **<attribute name, attribute type, attribute value>** holding information about attributes of the entity, (optionally) the name of an attribute domain, (optionally) a list of triplets **<metadata name, metadata type, metadata value>** that apply to all attribute values of the given domain.

The Context Information Model details how Context Information is structured and associated with Context Entities in order to describe their situation. In this model, Context Information is organized as Context Elements, which contain a set of Context Attributes and associated metadata. Details on this model are provided in the following subsections. [10]

Context Attributes - Name: xsd:string - Type: xsd:anyURI - Value: xsd:any Meta-Data - Name: xsd:string - Type: xsd:anyURI - Value: xsd:any Context Element - EntityId: xsd:string - EntityType: xsd:anyUR

The entity type is defined as a URI and thus can be for example an ontology reference (as URL) or a namespace (as URN). A context attribute represents atomic Context Information. An

attribute is defined as a set of information, namely a name, a type, a value and a set of associated metadata (e.g. timestamp, expires, source). The attribute value is expressed as any content, including strings or opaque objects represented using standard formats. An attribute domain represents the grouping of multiple attributes. Attribute domains allow requestors to specify a set of attributes of interest using a single string as attribute domain name [10]. The definition of attribute names, attribute types and attribute domains are out of the scope of this specification.

OMA NGSI which has defined two interfaces for exchanging information based on the information model. The interface OMA NGSI-10 is used for exchanging information about entities and their attribute, i.e., attribute values and metadata. [9]The interface OMA NGSI-9 is used for availability information about entities and their attributes.

As we have discussed the Fiware version of OMA NGSI 10 interface is a RESTful API via HTTP. Its purpose is to exchange context information. The three main interaction types are,

1. One-time queries for context information
2. Subscriptions for context information updates
3. Unsolicited updates

These typically support HTTP operations (GET, PUT, POST and DELETE). The operation scope of the GET operation can be further be limited by a URI parameter. Representation Format: NGSI 10 API supports the only XML as a data serialization format.

Coming to using Orion Context broker according to our requirement, we have a good understanding of Fiware NGSI version 2 API which is implemented by Orion Context Broker. Firstly, you need an account in Fiware Lab, so by registering using a valid email address we can obtain a valid authentication token to use in the REST API calls to Orion. Using username and password as an input to a script provided by Fiware, we can obtain the token needed. Over the whole research, we need to generate the access token mentioned before periodically to send any request to Orion Context broker of global instance.

Now, according to our demonstration requirement, we use Orion Context Broker global instance to create entities. We create two unique entities:
Temperature sensor with a unique ID with attributes “city_location” and “temperature” in the Orion Context Broker. Now the value of the attribute “temperature” is the temperature detected

by the sensor connected in an IoT platform and the type of this attribute is a “number” Image with a unique ID with attributes “city location” and “Image” in the Orion Context Broker. The value of the attribute “Image” is the encoded string of the Image captured by a camera connected to Raspberry Pi3 and the type of this attribute is a “string”.

Querying Entity:
This is the role of the user application, who wants to access the context information stored by the Orion Context Broker. The **`GET /v2/entities/{id}`** request is used to get Context information.
Updating Entity:
Now, this is the role of a producer application. Modifying the values of attributes can also be done using **`PUT /v2/entities/{id}/attrs/{attribute}/value`** request in a compact way.

As in the case of entity creation, apart from simple values corresponding to JSON datatypes such as numbers, strings for attribute values, you can also use complex structure or custom metadata.

3.14 Sysbench

SysBench[23] is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system running a database under intensive load. The idea of this benchmark suite is to quickly get an impression about system performance without setting up complex database benchmarks or even without installing a database at all. SysBench runs a specified number of threads and they all execute requests in parallel. The actual workload produced by requests depends on the specified test mode. You can limit either the total number of requests or the total time for the benchmark or both.
The general syntax for SysBench is as follows:

`“sysbench [common-options] --test=name [test-options] command”`

3.14.1 Features of SysBench

- fileio - File I/O test
- cpu - CPU performance test

- memory - Memory functions speed test
- threads - Threads subsystem performance test
- mutex - Mutex performance test

We made use of the following test modes,

3.14.1.1 CPU

The cpu[24] is one of the simplest benchmarks in SysBench. In this mode, each request consists in calculation of prime numbers up to a value specified by the --cpu-max-primes option. All calculations are performed using 64-bit integers. Each thread executes the requests concurrently until either the total number of requests or the total execution time exceed the limits specified with the common command line options.

Example:

"sysbench --test=cpu --cpu-max-prime=10000 run"

3.14.1.2 Memory

This test mode can be used to benchmark sequential memory reads or writes. Depending on command line options each thread can access either a global or a local block for all memory operations.[24]

The following options are available in this test mode [24]:

Option	*Description*	*Default value*
--memory-block-size	Size of memory block to use	1K
--memory-scope	Possible values: global, local. Specifies whether each thread will use a globally allocated memory block, or a local one.	global
--memory-total-size	Total size of data to transfer	100G
--memory-oper	Type of memory operations. Possible values: read, write.	100G

4 IMPLEMENTATION

4.1 Raspberry Pi3 Set-up:

To set-up a Raspberry Pi3 we need a computer, SD card (min. 8GB), display and HDMI cable, USB or wireless keyboard and mouse, power supply, and ethernet cable in a case when Wi-Fi is not yet enabled on Raspberry Pi3.

Firstly, we need to download the NOOBS (Newly Out Of the Box Software) or NOOBS LITE from the Raspberry Pi3 website or purchase a NOOBS pre-installed SD card. After downloading and extracting the NOOBS zip file we have to copy the content of the file to the formatted SD card on our computer. NOOBS include Raspbian, LibreELEC, OSMC, Recalbox, Lakka, RISC OS, Screenly OSE, Windows 10 IoT Core, TLXOS. NOOBS is available in a network to install only form or offline and network install form. NOOBS LITE version needs an internet connection for installation. Once the software is installed in the SD card, we can now insert it into the Raspberry Pi3 and turn it on.

The Raspberry Pi3 or any earlier version has no ON and OFF buttons. When the system is plugged in the device turns ON and when it plugged out the device will turn OFF. Now, we will be asked to select or checkbox the OS or OS's that we want to install on Raspberry Pi3 in the display. At this point, the selected OS has to be installed and run on the device successfully.

With the help of Raspberry Pi3 software configuration tool, we can configure our Raspberry Pi3. By using the command,

"sudo raspi-config"

we get access to the configuration tool and will be able to choose and edit setup options in the Raspberry Pi3. Every time we make a change in the configuration file of Raspberry Pi3 the system has to be Rebooted. To reboot the system, we use the command

"sudo reboot"

Now we are ready to plug and play the Raspberry Pi3.

4.2 SSH in Raspberry Pi3:

In order to access the Raspberry Pi3 command line from any other laptop or device that are under the same network, we make use of SSH. To make SSH connection we first must enable SSH in the Raspberry Pi3 configuration. We can do it in three different ways. First, by launching the Raspberry Pi3 configuration from the preference menu, enter to the interfaces tab, select enable SSH, then press ok. Second, enter ***"raspi-config"*** command in the terminal, select 'Interfacing options', select SSH, select 'ok' and choose 'finish'. Third, the easiest way, just enter the below commands in the command line of Raspberry Pi3,

"sudo systemctl enable ssh"

and

"sudo systemctl start ssh"

To enable the SSH in a headless setup, we just have to add a file named 'ssh', without any extensions in the smaller boot portion of the SD card. Then when we reboot the device, it searches the ssh file and when found it enables the SSH on Raspberry Pi3. The content of the ssh file is not concerned.

Later we need to know the IP address of the Raspberry Pi3 to make SSH connection. We can get to know the IP address of the Raspberry Pi3 in two ways. One is by just entering the **"ifconfig"** command in the Pi command line and pressing enter will give us the IP address of the Pi. The second option is useful when we don't have a display connected to the Pi to know the IP. In the second option, we log in to the router as Admin via web browser and check for the active wired client table for the IP of Pi when the Pi is connected through the ethernet cable and check for active wireless client table for the IP of Pi when the Pi is connected via Wi-Fi. In either way, we should make sure the connections are made properly in order to establish an SSH connection.

The command used to make an SSH connection to the Raspberry Pi3 in Linux and MacOs is

"sudo ssh pi@<IP>"

We used to make SSH connection via by using the above command. But after when we tried to mount a pen-drive to extend the memory of the Raspberry Pi3, we couldn't make an SSH connection to the Raspberry Pi3. Which leaves us with another option, is to use the third-party SSH clients to make SSH connection. In our case, we used PUTTY as our third-party SSH client.

4.3 Wi-Fi set up on Raspberry Pi3:

To set up Wi-Fi on the Pi we have to know our 'ESSID' of the device. To find the ESSID of your device we have to open the terminal in Raspberry Pi3 or in any other device by establishing the SSH connection to the Pi and enter the command

"sudo iwlist wlan0 scan"

and press enter. Technically ESSID is our Wi-Fi name. Then we enter the command

"sudo nano /etc/wpa_supplicant/wpa_supplicant.conf"

in the terminal and add the lines,

```
"network={
ssid="SSID"
psk="WIFI PASSWORD"
} "
```

at the bottom of the file. Then replace SSID with the name of our Wi-Fi and WIFI PASSWORD with the respective password of our Wi-Fi. Next, press CTRL-x then Y to save and exit the file. Now, we enter the commands,

"sudo ifdown wlan0"

and

"sudo ifup wlan0"

to start the Wi-Fi adapter. Lastly, restart the device with ***"sudo reboot" or*** **"sudo power-off"** the device. Finally, we will be able to set up the Wi-Fi connection on Raspberry Pi3.

Note: The IP address of the Raspberry Pi3 changes when it changes from ethernet connection to the Wi-Fi connection.

4.4 Remote desktop set-up on Raspberry Pi3:

Having remote desktop set-up on Raspberry Pi3 makes it easy to access the device. we will not need T.V, USB mouse and USB keyboard every time we need to interact graphically. So, we are setting up a remote desktop for Raspberry Pi3. To do so we first must enable VNC interface in the configuration file. We can enable it by using the command **"sudo raspi-config"**

or in the graphical case select menu, then Preference, then Raspberry Pi3 Configuration, then Interface and finally enable VNC. After enabling VNC, enter the command

"sudo apt-get install xrdp"

on the command line and press enter. After installation of 'xrdp', open the remote desktop app and enter the IP of the Raspberry Pi3 and press connect. In the case of Windows, you will be asked to log in to xrdp, with module *'sesman-Xvnc'*. Then we enter the username as 'pi' and the password of the Raspberry Pi3. Then you should be able to access the Pi via Remote desktop.

If we are still not able to access the Pi via Remote desktop then we should remove the xrdp we just installed with tags 'vnc4server' and 'tightvncserver'. To do this we use the command

"sudo apt-get remove xrdp vnc4server tightvncserver"

Then we reinstall the tightvncserver using the command

"sudo apt-get install tightvncserver"

and reinstall xrdp by the command **"sudo apt-get install xrdp"**. This should fix the problem. By following the above instructions, we should be able to access Raspberry Pi3 via any Remote desktop application.

4.5 Camera Connection to Raspberry Pi3:

At first, when the Raspberry Pi3 is not turned on, we connect the camera module to the CSI connector port of the Raspberry Pi3. We should make sure the camera is connected properly and then turn on the Pi. Then open the terminal in the Pi and enter the command *''sudo raspi-config''* and press enter to open the configuration menu. In this menu, we select the camera option and enables the camera. After enabling the camera, the device has to be rebooted. Then the camera has to be successfully set up.

To capture an image in Raspberry Pi3 with the camera module we use the command,

"raspistill"

In order to capture the image in jpg, the command is

"raspistill -o image.jpg"

and if we want the image to be in any other format, we replace jpg with the respective needed format. Now to record a video with the camera module in Raspberry Pi3 we use the command,

"raspivid"

The other way to capture a picture or record a video through the camera module used in the Raspberry Pi3 is to run a python script and import necessary modules like 'import camera',

'camera = picamera.PiCamera()'.

The 'import camera' will make the necessary libraries available to the script and *'camera = picamera.PiCamera()'* will create an instance of the PiCamera class. After successful installations of the above modules, we will be able to capture pictures and record videos by running the python script.

By editing the python script, we can horizontal or vertical file, preview the image for a specified period, video stabilization, and adjust the brightness, contrast, saturation, sharpness, colour effects, etc.

In our project, we used an AI artefact which should be able to face recognition from the live stream produced by the Pi camera. To make this happen we first made a trail to run a video stream in the local network. We used the Real Time Streaming Protocol (RTSP) which is a network control protocol designed for use in communication systems and entertainment to control streaming media between endpoints [25]. The command to start the video streaming in the local network,

"raspivid –o - -t 0 –n | cvlc –vvv stream:///dev/stdin –sout '#rtp{ sdp = rtsp://:8554/} ' : demux=h264 "

In the above command "-o -" causes the output to be written to stdout, "-t 0" sets the timeout to disabled, "-n" stops the video being previewed in the monitor, Cvlc is the console vlc player, "-vvv" and its argument specifies where to get the stream from, "-sout" and its argument specifies where to output. We can also specify the width, height, frames per second of the video in the above command.

Note: Install VLC player in Raspberry Pi3 in advance to run the above command.

4.6 Docker deployment on Raspberry Pi3:

To implement the demonstrator in terms of docker, we should set up the Raspberry Pi3 platform which has Raspbian OS. It is very suitable for developers to get started with docker and start playing with the advantages provided with containers and images.

It has become very easy to deploy Docker on Raspberry Pi3. First, we log in to the Raspberry Pi3 through 'putty' or by remote desktop and open the terminal and enter the commands **"sudo apt-get update", "sudo apt-get upgrade"** and reboot the device. After rebooting the device, we enter the curl command,

"curl -SSL https://get.docker.com | sh"

and press enter. That's it, docker starts installing in Raspberry Pi3. After the installation is done, we can see that we are running ARM architecture and the version of the docker.

To check if the docker daemon is running we enter the command **"sudo docker info"** and press enter. If the docker daemon is running successfully then we can build our own image, pull or push the image from or to the Docker hub on Raspberry Pi3.

To run your first docker image, we can do it using the command

"docker run hello-world"

Now, you can see the hello-world image of docker is pulled from the cloud and says Hello from Docker!
Root privilege is needed to access the docker daemon because it binds to the socket. We can avoid this by creating a group called docker and add users to it. Now, the Docker Daemon is running in the system.

4.7 FIWARE Interface:

4.7.1 Python Scripts in Raspberry Pi3 to interact with Fiware using NGSI API.

Required Installations: Python requests module and Base64 module in Raspberry Pi3,

Sending sensor information:
Temperature sensor Entity which has been already created in Fiware Orion Context Broker is used along with URL or a namespace (as URN). A context attribute represents atomic Context Information. Here an attribute is defined as temperature with a type, a value and a set of associated metadata (e.g. timestamp, expires, source). The attribute value is expressed as a

number which is the temperature value detected by the temperature sensor connected to Raspberry Pi3. Python Script imports Requests module which allows you to send HTTP requests using Python. We add headers, JSON format of the entity in PUT request to send the desired information to Orion Context Broker.

Sending image by encoding it:
Image entity which has been already created in Fiware Context Broker and URL or a namespace URN is used. A context attribute represents Image and its type, value. Value is a Metadata consisting of a string of encoded Image data. Therefore, the type of the attribute is a String. The method can be briefly explained using a block diagram in which we show the process, our python script undergoes. The captured Image is saved first and then encoded. This encoded image information is inserted in JSON data of PUT request as a string. The PUT request using appropriate headers and JSON formatted data is implemented using requests libraries in python. This process is explained in figure [4].

4.8 AI Artefact

The meaning of the word 'artefact' is a tool. That explains AI (Artificial Intelligence) artefact. In our project, we are using an artificial intelligence (AI) tool that can detect and recognize a face. Our one of the main objectives is to run an AI face recognition tool on the constrained device, Raspberry Pi3 and be able to detect and recognize a face with no problem.

The AI model is designed in a way to detect the faces from a pre-set folder. This pre-set folder consists of the faces that have to be detected. Any face that is out of the pre-set image folder will not be recognized and will display as 'unknown' face. The AI artefact is designed using a python script. Two important modules that have to be installed in the Pi to run the AI artefact are 'face_recognition' and 'openCV'. The command to install the face recognition module is

"pip install face_recognition"

and the command for OpenCV module is

"sudo apt-get install python-opencv"

Sometimes the required camera modules may not be installed properly. So, by using the command,

"sudo modprobe bcm2835-v412"

may eliminate the problem.

Finally, the obtained output of the AI face recognition artefact has to be sent to the Fiware context broker using NGSI API.

The interface is created in AI model to which Image from Global context broker is the input for face recognition. An Option is created in an already existing AI model to GET the image from the Global Context Broker. We can receive the encoded context information of an Image captured by specific Raspicam connected to a specific Raspberry Pi3 using GET request which includes the unique ID of already existing Image entity of the same Raspicam. After pulling the encoded Image, we decode the Image using Base64 module in Python and save it. This decoded image is compared with the Images stored in the database of the AI model for face recognition. The whole process is explained in figure [4].

4.9 AI Deployment

4.9.1 Deployment of AI model (a set of Python scripts and a folder of Images) in Raspberry Pi3

Requirements: Python- OpenCV, face recognition, numpy, Enabling access to the camera.

Docker has the ability to pack all the libraries and dependencies and software required into a container and can be shipped to the cloud. This image or container in the cloud can be used by the user depending on whether the image is public or private. This AI model with docker is built by following instructions and is made available in Docker hub. The challenge here is to build a docker image supporting the ARM architecture of Raspberry Pi3 and also to be able to perform in any low-power devices (Internet of things devices).
"Dockerfiles" enables you to create your own images. It describes the software that makes up an image. We should write a docker file which consists of a set of instructions that specify what environment to use and which commands to run. Create a new text file as Dockerfile and include all the necessary instructions. It includes an Image as a launching point which supports the ARM architecture of Raspberry Pi3. The image we have used to dockerize the AI artefact is

"Raspbian/stretch".

The Technical challenge has occurred here while selecting the image which is said to be a base image on which all the dependencies and the software will be installed. There are many

deprecated base images available and it has to be selected with a lot of research to make it compatible in ARM platform supporting the Raspberry Pi3 3.
All the instructions required to install the required libraries for AI artefact like OpenCV, python and more are mentioned in the Dockerfile.

Finally, instructions to run the AI model and sending the results to Fiware context broker are given. With this Dockerfile, a new image is built using the command:

"docker build -t thesis ."

Successfully, the image will be now built and to run the dockerized AI model we use the following commands:

"xhost local:root"

and

"docker run --privileged --rm -it --env DISPLAY=$DISPLAY --env="QT_X11_NO_MITSHM=1" -v /dev/video0:/dev/video0 -v /tmp/.X11-unix:/tmp/.X11-unix:ro thesis2:latest"

Detailed Explanation of the commands:

- **xhost local: root**
  Enables permissions to docker to access peripheral devices
- **docker run --privileged --rm -it**
  Runs docker in privileged mode
- **--env DISPLAY=$DISPLAY**
  Sends the display id from the host to the container.
- **--env="QT_X11_NO_MITSHM=1"**
  Required by OpenCV to show the display.
- **-v /dev/video0:/dev/video0**
  This lets the container find the camera.
- **-v /tmp/.X11-unix:/tmp/.X11-unix:ro thesis2: latest**
  This lets the container find the display via X server and runs the docker image.

Dockerized AI model can now be pushed to the Docker hub repository and Boseyes AI repository for the users to have easy access to it.

4.9.2 Modifying the AI model to send results to Fiware Context Broker

After the process in 3.1, We are able to execute the AI model of face recognition. Whenever the face is recognized, the result- name of the person is sent to Fiware Context Broker. If the face is not recognized, the result would be “unknown”. Before we send the result, an entity is created in Fiware context broker with the attribute as “AI result” and value as a result of the AI model with type as “String”. Initially the value of the attribute “AI result” is unknown. After the entity creation, AI model sends the result to context broker using requests library in python, URL or URN of context broker, access token to Fiware, a unique ID of the entity.
The process is explained in figure [4],

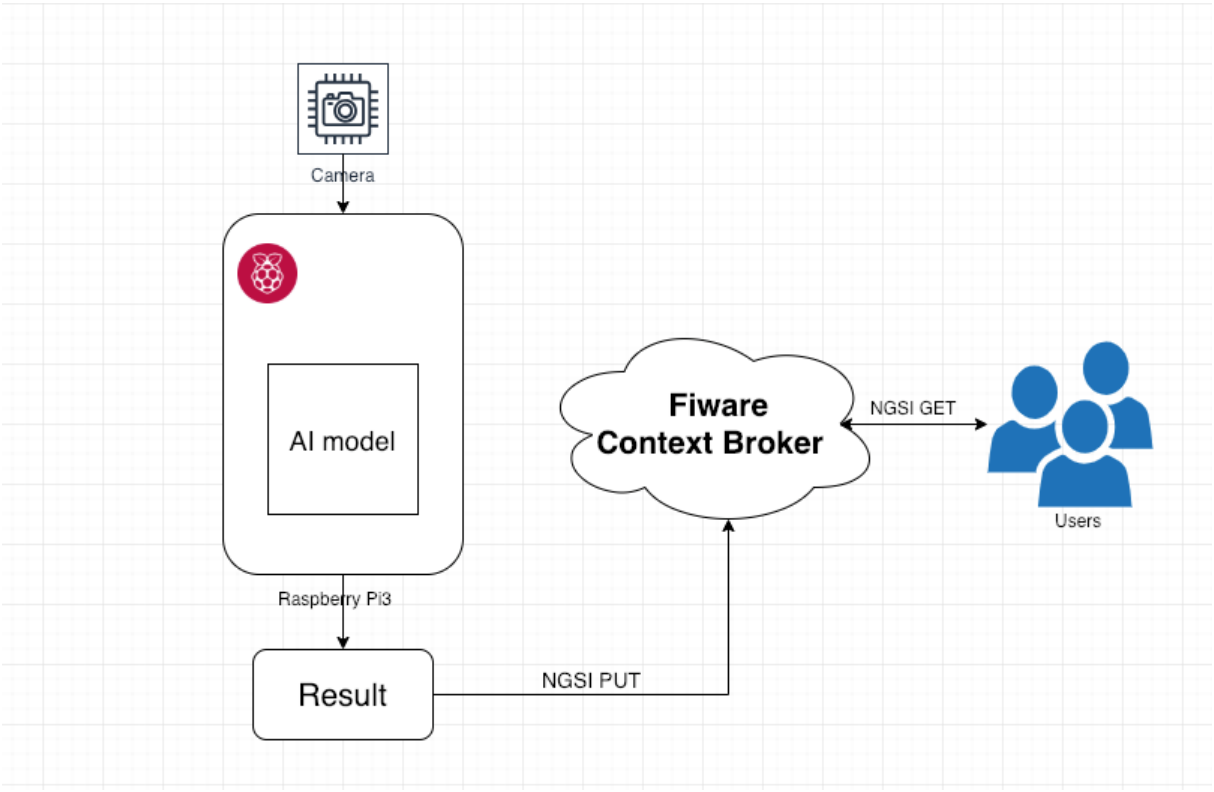

Figure 4: System Workflow

4.10 Performance Evaluation

Implementation of Docker AI application in Raspberry Pi3 gives us a chance to do more research in performance evaluation. We have researched a lot of literature to figure out how performance analysis is done using different methods and tools. The experimentation is done by using the research done and using effective methods and tools to achieve the aim. Statistics are calculated to study the behavior of our cases under a variable parameter. Investigation on how Docker effects the performance in Raspberry Pi3 is done. After researching different approaches to test performance, we decided to use the sysbench tool to measure CPU load and memory.

To investigate face recognition and its validation in Raspberry Pi3 with and without docker, execution time is used. Execution time is calculated by executing the Face recognition model which is a python monolithic code with docker and also by executing just the Face recognition model (the python code). Theoretically, docker is very light-weighted virtualization, so we are investigating that it should not have any performance overhead.

The aim here is to present a performance evaluation of docker having face recognition model in Raspberry Pi3, which has not done before to our knowledge.

We tested Raspberry Pi3 3 for its performance by applying load into the system using the sysbench tool. We compare the performance in two scenarios that is with Docker and Native OS raspbian to validate how docker is affecting the Memory, CPU load and the execution time of face recognition. Computing mean and standard deviation to determine the CPU performance hit and memory performance hit along with the execution time of face recognition gives us a clear analysis of the statistics.

Setup: The parameters we compute are CPU load and memory along with the execution time.

For Case-1, The host Operating system raspbian in Raspberry Pi3 3 will be measured for its performance in terms of CPU load, memory and execution time. The tool sysbench is used to measure and is described in 2.13.

Case-2: To measure the performance of Raspberry Pi3 with a virtualization layer, Docker.

CPU performance evaluation: We measure the performance by applying load to CPU and calculating its execution time required to compute a complex mathematical task according. This depends on the input provided by the sysbench tool.

Memory performance evaluation: We measure the access speed by requesting a block of memory. The whole process is explained clearly in the further sections.

5 VERIFICATION

5.1 Verification Methods

5.1.1 Proof of Concept

Proof of Concept validates the idea of the thesis to be practically potential in the now-a-day software industry that has a thriving need for developing containerization technologies, Cloud technologies and the Internet of Things. Our aim is to connect all the technologies involved in an efficient way and demonstrating the working model of all the methods in these technologies. Our Concept is to connect the Low-power device like Raspberry Pi3 to Cloud using various mechanisms in Fiware and Connecting containerization methodology using docker inside Raspberry Pi3. These give a challenge of investigating various approaches and following a method to achieve our aim for this thesis.

5.1.2 Qualitative Validation:

Qualitative validation involves the mechanisms and research used in the thesis. Verifying the qualitative research is to check the process of implementation and demonstration, confirming that they work and being certain about the parameters involved in the methods used.
The design and implementation are extracted from various sources according to the requirement.
The research is done in the fields of Cloud, Internet of things, Containerization, Artificial Intelligence.
Cloud involves Fiware Mechanisms, Internet of things involve various devices, their Operating systems, their working, and usage. Containerization involves various types of containers like docker.
- The Fiware is used to implement the mechanisms it has called the Context Broker mechanism to implement the demonstration according to the design requirements. The basic entity is created in Fiware context broker to which the result is sent as a meta-data in JSON format. The Context element is described as shown below in figure [5].

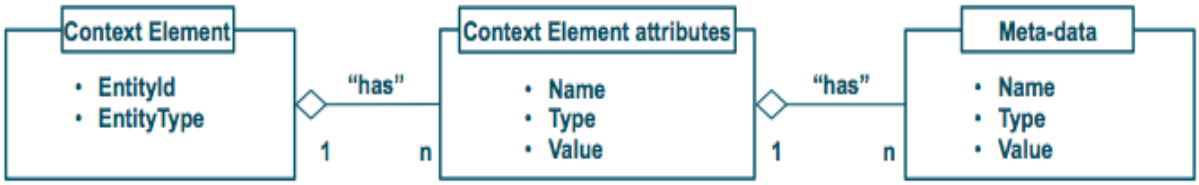

Figure 5: Context element description

Docker is used to containerize the AI model used as shown in section 3.6.

- The formulation of a Docker image in terms of AI model and Raspberry Pi3 involves various components as shown in Figure [6].

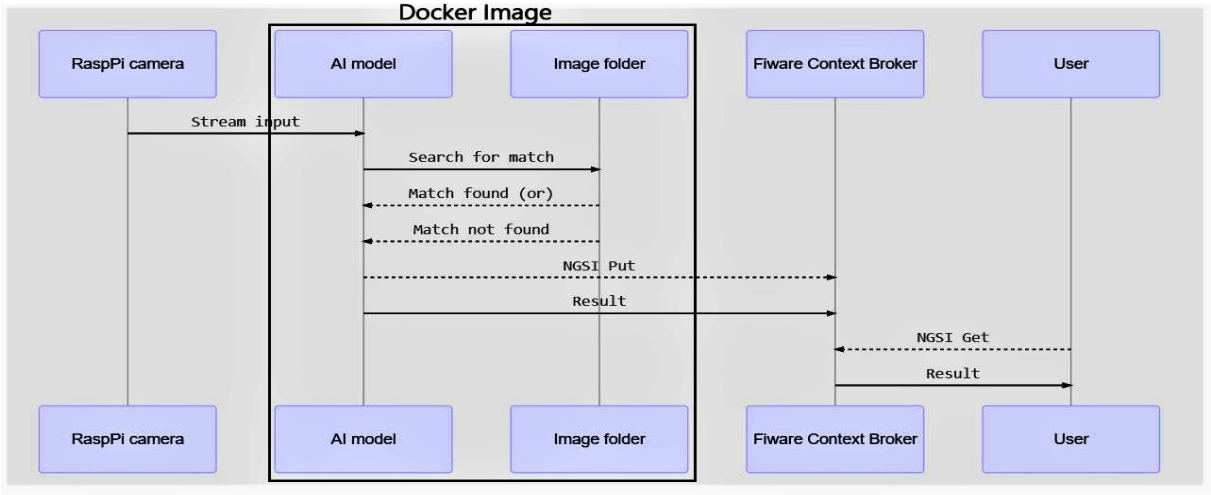

Figure 6: Sequence Diagram

The strategy used here with all the research done is to demonstrate a working model of Internet of things handling a highly processing method along with the Cloud mechanisms. The data used here is purely based on the research done and practical analysis of the components used.

The theory is developed by studying various types in the components involved, their working and usage. For example, studying types of docker and low-power devices used in the Internet of things. This theory is used to develop the data required to implement the demonstration and to analyze the coherence between the methods, the interdependence of the processes.

The data involves various formulations and mechanisms in certain categories. These are validated by practical implementation providing a clear insight into the strategy involved.

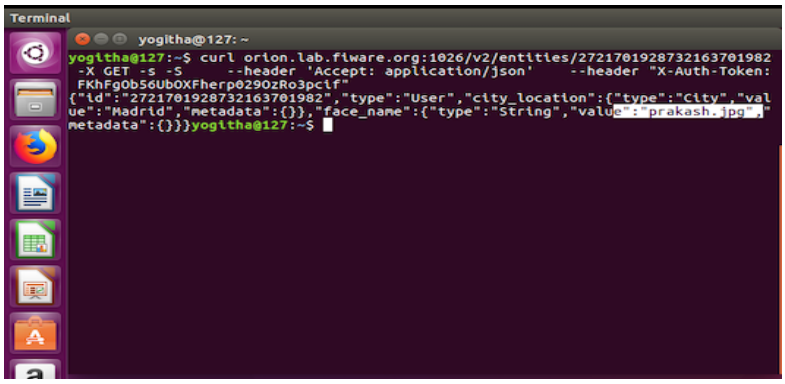

Figure 7: Screenshot of Context element in Fiware showing us the result of Face recognition AI model in Raspberry Pi3 (Face recognized is "Prakash")

The above screenshot Figure [7] shows us the Entity created in Fiware and its result. The result of the face recognition is sent to Fiware context broker entity with a unique ID as shown. The parameter here is 'face_name' which is metadata in the entity. In our case, the name of the person whose face was recognized is "Prakash". This entity can be pulled by any user in Fiware. Therefore, the entity data is pulled in a different device which in my case is an Ubuntu platform with curl installed in it. From the result, we can see the name of the person whose face was recognized in low-power device Raspberry Pi3 3.

We have also done Face recognition demonstration in Raspberry Pi3 platform to verify the whole process of using feasible docker image and establishing the communication to Fiware in Raspberry Pi3.

DEMONSTRATED THE PROCESS OF CUSTOM AI DOCKER IMAGE IN RASPBERRY PI3 [26]

5.1.3 Quantitative validation:

The results and analysis provided in this thesis develop insights into methods for validation of computational model prediction. Types of tools and approaches are investigated for performance testing. Performance testing is extended based on various parameters involved while calculating CPU and memory. Two types of validation experiments are considered, fully characterized that is all the experimental inputs are measured and reported and the portability of the face recognition is validated using the feasibility of it and the processing time of the AI application in Native OS and in Docker. All the experiments are done with respect to docker and without docker that is Native OS raspbian for Raspberry Pi3. These are validated by section 5.

6 RESULTS AND ANALYSIS

The results are obtained by conducting several experiments in the Raspberry Pi3 3 platform. The compatibility of the AI model in docker will also be discussed. Our performance results can give guidance on how Docker is working and providing various advantages.[27] Docker can be used to maintain a single docker image for virtualized and non-virtualized servers which can be deployed on anything from a fraction of a core to an entire machine. There can be reduced cross-traffic within docker as a server running one container per socket can be faster as it avoids spreading the workload across sockets providing us a performance advantage. [28] The applications we used, and all the resources needed for the AI application are isolated in Docker. This is called Dependency Isolation.

Therefore, an application in docker won't use all available resources, which would normally lead to performance degradation or complete downtime for some applications. If the applications running in Docker is configured with network tuning such as CPU affinity and using different modes might give us better performance because of a few sources of latency are minimized.[29]

This is the reason, our AI application and sysbench in Docker is giving us almost equal performance or sometimes, even better performance. The analysis is done based on the data collected from performance testing in multiple scenarios using tool sysbench in Docker and Native OS on Raspberry Pi3. The parameters tested are CPU performance, memory performance and execution time. Each test is done individually by closing all the other applications on the platform.

While calculating CPU and memory performance, each experiment is measured 15 times. Data collection is done and analyzed by computing the statistics of the data. The values are provided in a table and respective graphical representation is shown as a part of the results of this thesis. Graphs and tables are used to represent the numerical data in a time effective manner to understand the behavior of the Raspberry Pi3 platform.

6.1 EXECUTION TIME

The execution time of the face to get recognized as soon as the display window is opened is calculated. Executing the AI model directly using the python code gives us an average time of **47.86 seconds**. Executing the docker image until the face gets recognized gives us an average

time of **46.93 seconds**. These programs are executed 5 times each to get the average. So, the difference in execution time is very negligible in Native OS and with docker. This shows us that the processing time of the code or docker image for face recognition is almost similar.

6.2 PERFORMANCE ANALYSIS OF CPU

The CPU performance is calculated using sysbench benchmarking tool to measure the execution time of a synthetic load. We consider 10 scenarios with one thread and four threads each by changing the parameters such as load in docker and native OS. The load can be varied using the option "–max-prime-number" in the sysbench tool. When we increase the threads workload, each worker thread will be allocated a mutex (a sort of lock) and will, for each execution, loop a number of times (documented as the number of yields) in which it takes the lock, yields (meaning it asks the scheduler to stop itself from running and put it back and the end of the run queue) and then, when it is scheduled again for execution, unlock. It reports the actual time for the completion of the activity. The number of threads can be varied using the option "--num-threads" in the sysbench tool. The five scenarios we consider use 1000,2000,5000,12000,20000 load with one thread each. The other five scenarios we consider use 1000,2000,5000,12000,20000 load with four threads each. Each scenario is calculated 15 times in docker and in Native OS. The docker and Native OS results are compared and analyzed. The docker overhead is calculated.

The results are shown in a table below Table [1] and Table [2]. The results are analyzed by comparing Native OS values and Docker values. It is represented in the graphical form below GRAPH. From the experiments we have done, it is clearly understood that Docker has a negligible performance impact on Raspberry Pi3 3.

CPU Load (1 Thread)	Without Docker		With Docker		CPU Performance Hit (%)
	Mean (sec)	Standard Deviation	Mean (sec)	Standard Deviation	
1000	5.55	0.0603	5.49	0.0473	1.08%
2000	14.51	0.0493	14.62	0.0458	-0.73%
5000	52.67	0.2042	52.63	0.0577	0.07%
12000	180.48	0.0721	181.11	0.1179	-0.35%
20000	371.36	1.1861	370.78	0.2843	0.16%

Table 1*: Mean, Standard deviation of CPU load execution time in Raspberry Pi3 with Native* OS and with Docker with one thread.

CPU Load (4 Threads)	Without Docker		With Docker		CPU Performance Hit (%)
	Mean (sec)	Standard Deviation	Mean (sec)	Standard Deviation	
1000	1.48	0.0896	1.44	0.0624	-2.55%
2000	3.76	0.0436	3.72	0.0702	-1.17%
5000	13.55	0.1249	14.10	0.4980	3.85%
12000	47.24	1.5808	47.49	1.6532	0.53%
20000	105.13	2.5598	99.27	4.0049	-5.90%

Table 2*: Mean and Standard deviation of CPU load execution time with Native OS and with Docker in Raspberry Pi3 with four threads.*

The trend of CPU load as shown below in Graph [8] explains that as load increases, the execution time also increases and shows that the behavior is similar in both Native OS and with docker which is linear

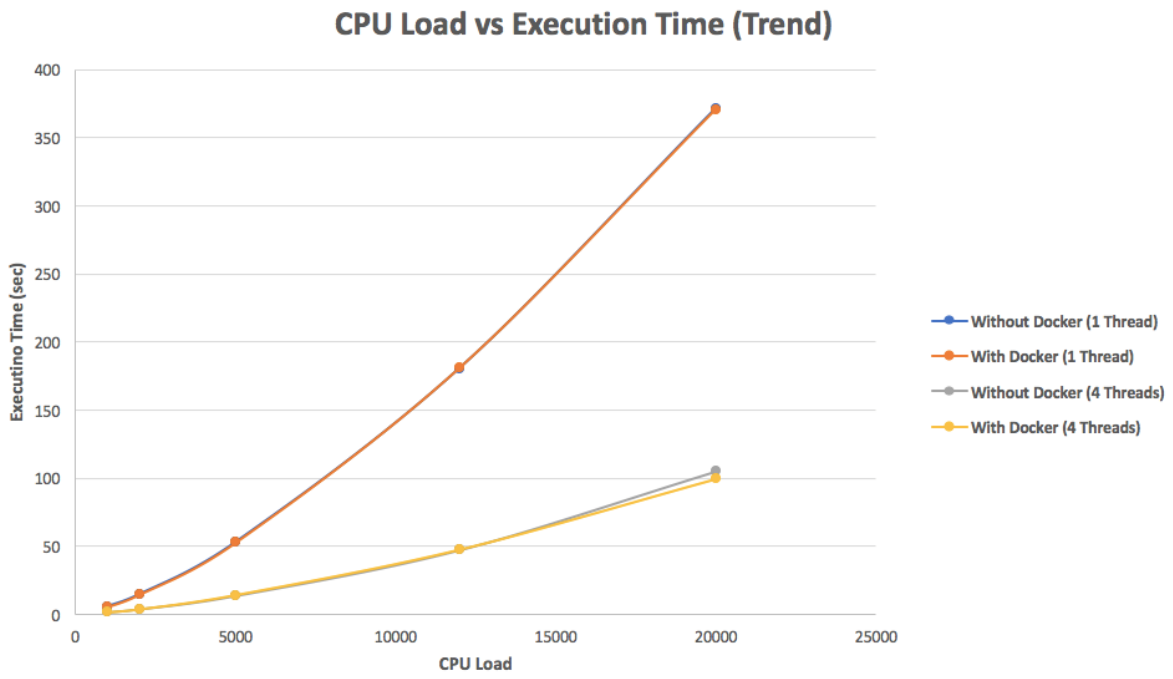

Figure 8*: CPU Load vs Execution Time*

The Graph [9] below represents the numerical information of CPU performance in Native OS compared with docker at a few certain scenarios computed by conducting a number of experiments. The difference is negligible as we can clearly predict from the graph and sometimes negative too. The statistics are calculated in percentages to visualize the differences clearly.

The graph below explains the CPU performance Hit when we used docker compared with Native OS. The x-axis shows us the load applied and Y-axis shows how much the Docker impacted the CPU in percentage. These values are also shown in Table 1 and 2.

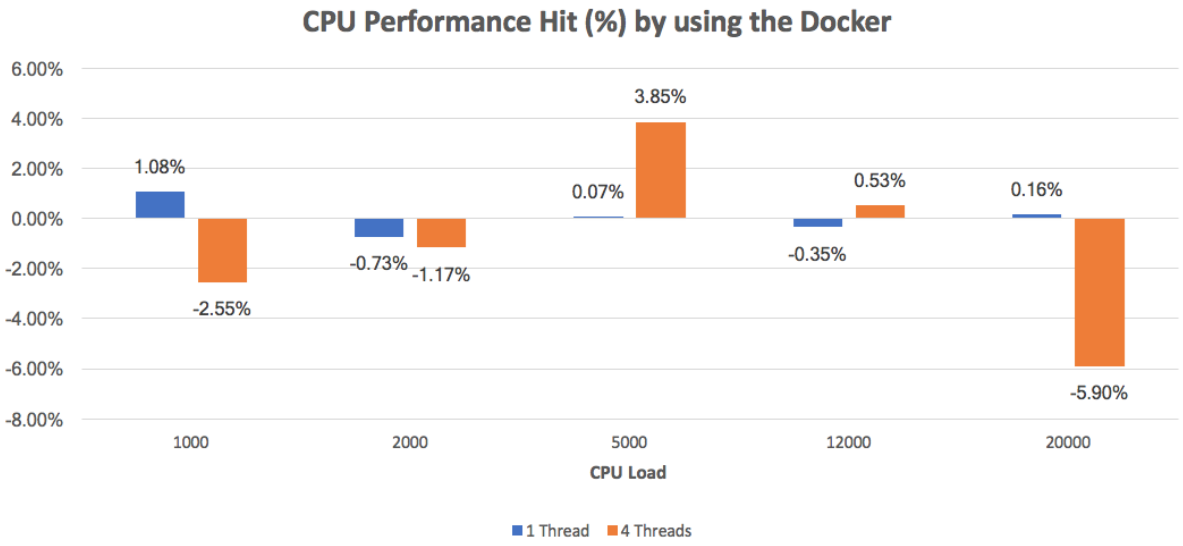

Figure 9*: CPU Performance Hit (%) by using the Docker*

The Graph [10] below explains the pattern of the Execution time in various scenarios to show the comparison of the behavior when using docker and in Native OS. The x-axis is the execution time of the CPU when a certain load is applied. Y-axis is the load applied.

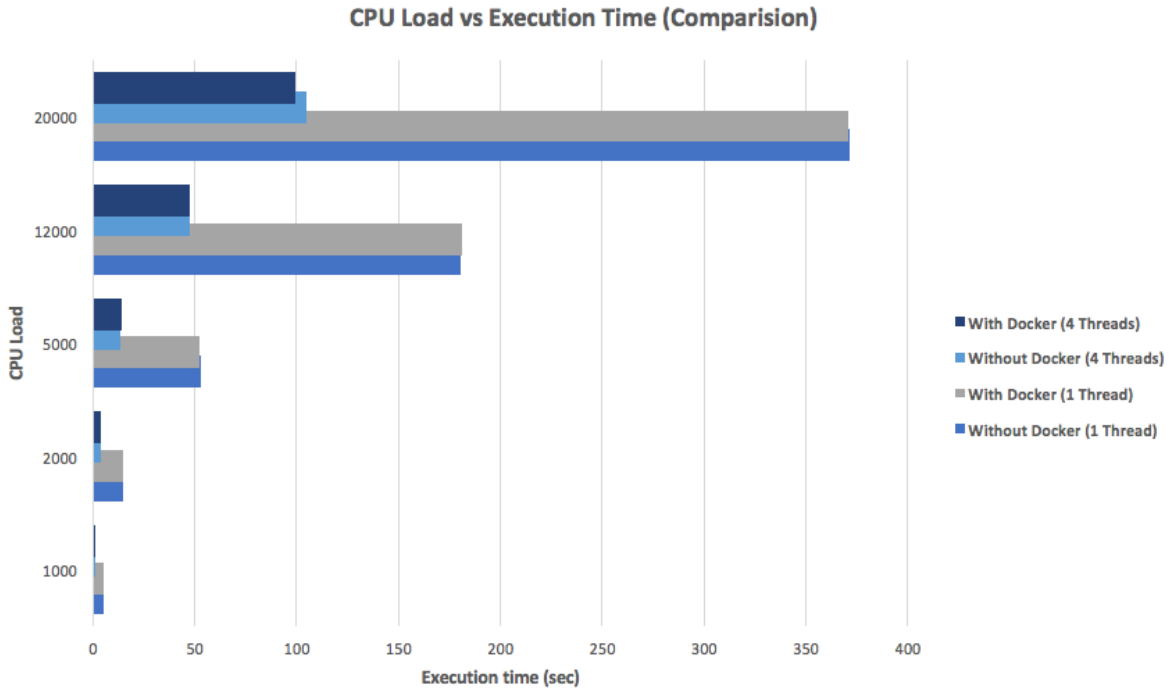

Figure 10*: CPU Load vs Execution Time*

6.3 PERFORMANCE ANALYSIS OF MEMORY

Memory is calculated using sysbench benchmarking tool. It uses a complicated framework to support different test targets like memory. When we use sysbench to benchmark memory, sysbench allocate a memory buffer and then read or write from/on it, each time for the size of a pointer (32 bit or 64 bit) and until the total buffer size has been read from or written to. This activity will be continued until the provided volume (–memory-total-size) is reached. The load can be increased or reduced by providing multiple threads (–num-threads), size of the buffer (–memory-block-size) and request type (read/write/sequential/random). So, we use different options to switch between read/write and sequential/random as mentioned. This gives four experiments to be conducted 15 times twice (once with one thread, other with four threads). We compute the statistics by doing the mean and standard deviation of the data collected by the experiments we did. You set the options in each experiment using sysbench according to the requirement we have. The results are provided here in the table below Table [3] and Table [4]. These whole processes are repeated in Docker after finishing in the Native OS. The graphical representation is provided GRAPH comparing the results in Native OS and Docker. It can be clearly depicted that the difference in both is negligible.

Memory Operations (1 Thread)	Without Docker		With Docker		Memory Performance Hit (%)
	Mean (MB/s)	Standard Deviation	Mean (MB/s)	Standard Deviation	
READ SEQUENTIAL	724.27	3.5741	732.68	7.9143	1.16%
WRITE SEQUENTIAL	546.46	4.6078	547.74	3.8956	0.23%
READ RANDOM	707.59	4.4186	703.47	8.7705	-0.58%
WRITE RANDOM	683.13	0.9762	693.56	3.6826	1.53%

Table 3*: Mean and Standard deviation of Memory Operation speed in Raspberry Pi3 in Native OS and with Docker using one thread.*

Memory Operations (4 Threads)	Without Docker		With Docker		Memory Performance Hit (%)
	Mean (MB/s)	Standard Deviation	Mean (MB/s)	Standard Deviation	
READ SEQUENTIAL	2317.19	123.7630	2441.41	141.1282	5.36%
WRITE SEQUENTIAL	1898.18	4.7499	1975.59	38.7687	4.08%
READ RANDOM	2392.68	32.0312	2406.38	104.8472	0.57%
WRITE RANDOM	2312.29	15.9723	2263.86	91.8578	-2.09%

Table 4*: Mean and Standard deviation of Memory Operation speed in Raspberry Pi3 in Native OS and with Docker using four threads.*

Graph [11] below explains the memory performance Hit when we used docker compared with Native OS. The x-axis shows us the type of case we applied, and Y-axis shows how much the Docker impacted the memory of Raspberry Pi3 platform in percentage. These values are also shown in Table 3 and 4.

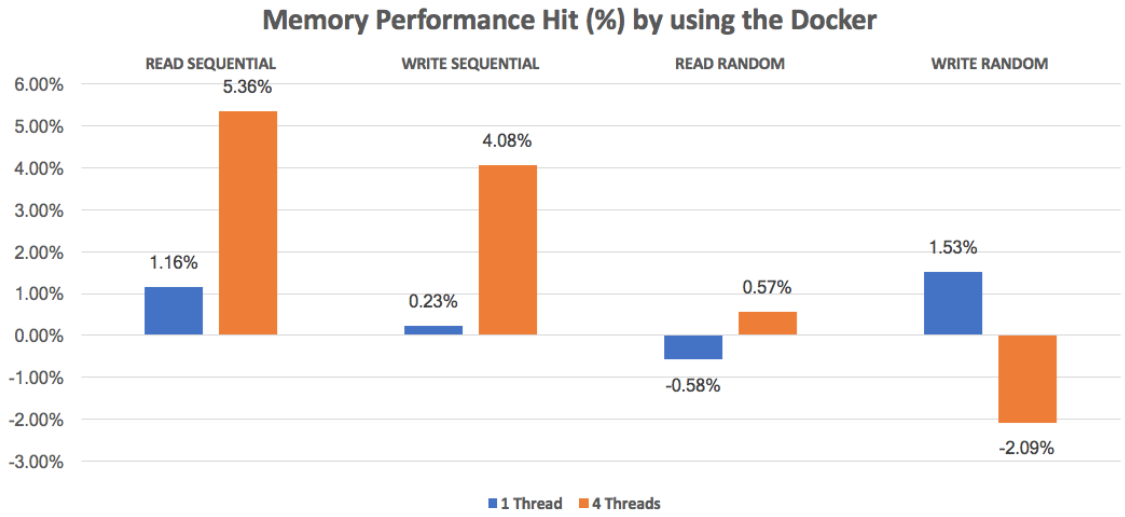

Figure 11*: Memory Performance Hit (%) by using the Docker*

The Graph [12] below explains the pattern of the Speed in various cases applied as explained to show the comparison of the behavior when using docker and in Native OS. The x-axis is the Speed in Mb per second of the Memory when a certain case is applied. Y-axis is the type of case applied.

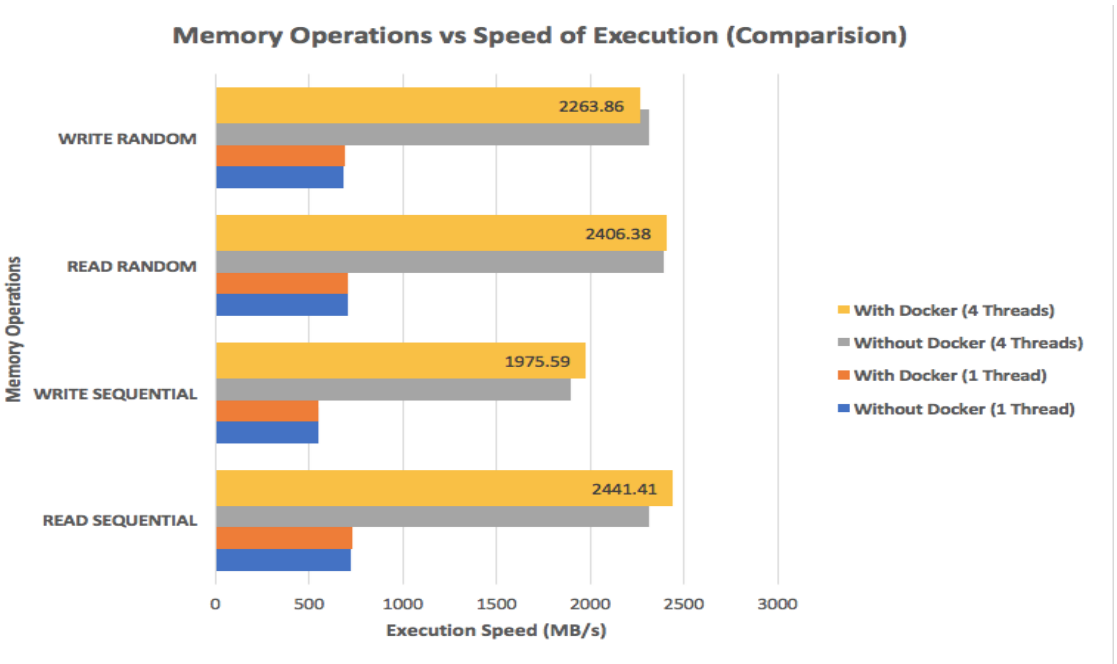

Figure 12*: Memory Operation vs Speed of Execution.*

7 CONCLUSION AND FUTURE WORK

Interfaces are implemented for communication establishment between a Raspberry Pi3 and Fiware OpenStack. Fiware context broker has different mechanisms for demonstration. Interfaces implemented can be described as API requests sent. It provides the NGSI API, which is used by us. It becomes a means of communication. Demonstration of this is shown by using a camera module on Raspberry Pi3. A face recognition AI model is implemented in Raspberry Pi3. It uses this camera module to transmit the name of the recognized face to FIWARE context broker using the NGSI put request. On the client side, NGSI GET request is used to get the name of the recognized face.

We have discovered how docker was an efficient and easy way to deploy an AI model on Raspberry Pi3. We have discovered how transmitting data directly from an end device such as Raspberry Pi3 to Cloud could be of greater importance in future cases. The greatest challenge was to dockerize the AI model supporting Raspberry Pi3. Fiware Context broker, we used to be a global context broker. To implement a context broker for a private scenario, Fiware also provides a dockerized context broker service separately which can be used in our own private network. So, our thesis finally explains how all the components we have used are inter-connected and also in an efficient way.

Future work: Face recognition model built using docker provides an advantage to be used by various users anywhere and anytime. It can be enhanced by providing more features to detect more complex faces and followed by sending more complex information to the cloud. The security aspects in the communication by this face recognition model can be improved by adding a firewall, encoding, encryption, etc.

7.1 ANSWER FOR RESEARCH QUESTIONS

R1: Which interfaces are needed to be implemented for a sensor on a Raspberry Pi3 to transmit data to Fiware cloud using the Fiware context broker mechanisms. How can these interfaces be described and implemented?

The interfaces that need to be implemented between the sensor and Raspberry Pi3 is the connection between them. The challenge here is to implement the interfaces for different types of sensors or cameras used. A temperature sensor is connected using GPIO pins provided on low-power IoT devices and different types of applications are used to get the information from these sensors. The technical difficulty and a challenge here is to implement an interface between Raspberry Pi3 and Fiware cloud using Context Broker mechanisms. Fiware cloud has many advantages as explained in this thesis and context broker mechanisms is also an important element in Fiware OpenStack.

This difficulty is overcome by implementing an API request compatible with Raspberry Pi3 and feasible with our use case, which is the face recognition. The API request is created by a deep study on Fiware context broker mechanisms and NGSI API provided by it. The implementation and creation of this NGSI API according to the use case of ours explain the flow of information from Raspberry Pi3 3 (Low power device) to Fiware Context Broker.

R2: Which interfaces are needed to be implemented for a Bonseyes AI artefact to be deployed in the Raspberry Pi3 and send the result to Fiware cloud using the Fiware control mechanism, e.g. the broker mechanisms. How can these interfaces be described and implemented?

The Face recognition AI model created is compatible in the Linux kernel system and the Docker is easy to deploy on systems of Linux. The practical challenge here is to deploy the AI model in a low-power device like Raspberry Pi3 with Raspbian OS which is a totally different form of Linux.

Bonseyes environment aims to provide a platform for the development and deployment of AI applications. As this thesis also contributes to Bonseyes platform, AI model created can be pushed to Bonseyes which makes the docker image available in the Bonseyes repository to users. Users of Bonseyes can pull the Face recognition docker image at any anytime and anywhere.

So, to achieve the practical challenges here, Docker image is customized and built in a way to support the ARM architecture of Raspberry Pi3. It also supports any low-power IoT device with ARM architecture.

The successful execution of the AI model here and the accuracy of the face recognition explains the feasibility of the deployment of a docker image with AI model in single-board devices.

The Interface between Raspberry Pi3 and Fiware context broker is similar to the answer in the first research question. Here, the name of the face recognized is sent as metadata in JSON format which is a part of an entity in Fiware global context broker. The entity is already created in Fiware context broker. The interface is the NGSI API created between Raspberry Pi3 and Fiware. The challenge here is the creation of an interface that is NGSI API compatible inside docker and in low-power-devices.

R3: How are the performansce of Raspberry Pi3 3 platform and Face recognition in it is affected by deploying a container image in the system in terms of CPU load, memory and execution time?

Performance evaluation is calculated by conducting several experiments and collecting the data. The main aim is to compute the CPU performance hit, Memory performance hit and execution time of the face recognition when there is a layer of docker virtualization compared with native OS. Execution time is the time taken for the face to get recognized. The time taken is almost negligible when we execute the face recognition with docker and with native OS.

With the results and analysis provided in this thesis, we can conclude that docker virtualization provides negligible performance impact in terms of CPU and Memory and also a negligible difference in the execution time of Face to get recognized when executed with docker and native OS.

REFERENCES

[1] F. Wortmann and K. Flüchter, “Internet of Things: Technology and Value Added,” *Bus. Inf. Syst. Eng.*, vol. 57, no. 3, pp. 221–224, Jun. 2015.

[2] “LNCS 7768 - Internet of Things.”.

[3] K. Lee, D. Murray, D. Hughes, and W. Joosen, “Extending sensor networks into the Cloud using Amazon Web Services,” in *2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications*, 2010, pp. 1–7.

[4] A. Botta, W. de Donato, V. Persico, and A. Pescapé, “Integration of Cloud computing and Internet of Things: A survey,” *Future Gener. Comput. Syst.*, vol. 56, pp. 684–700, Mar. 2016.

[5] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, May 2010.

[6] T. Dillon, C. Wu, and E. Chang, “Cloud Computing: Issues and Challenges,” in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 27–33.

[7] W. Yu *et al.*, “A Survey on the Edge Computing for the Internet of Things,” *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[8] F. Ramparany, F. G. Marquez, J. Soriano, and T. Elsaleh, “Handling smart environment devices, data and services at the semantic level with the FI-WARE core platform,” in *2014 IEEE International Conference on Big Data (Big Data)*, 2014, pp. 14–20.

[9] “NGSI-9/NGSI-10 information model - FIWARE Forge Wiki.” [Online]. Available: http://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/NGSI-9/NGSI-10_information_model. [Accessed: 19-Sep-2018].

[10] “OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf.”.

[11] D. Zhang, X. Han, and C. Deng, “Review on the research and practice of deep learning and reinforcement learning in smart grids,” *CSEE J. Power Energy Syst.*, vol. 4, no. 3, pp. 362–370, Sep. 2018.

[12] Y. Pan, “Heading toward Artificial Intelligence 2.0,” *Engineering*, vol. 2, no. 4, pp. 409–413, Dec. 2016.

[13] C. Lv, Q. Li, Z. Lei, J. Peng, W. Zhang, and T. Wang, “PaaS: A revolution for information technology platforms,” in *2010 International Conference on Educational and Network Technology*, 2010, pp. 346–349.

[14] Q. Xia, Y. Lan, and L. Xiao, “The Status Prediction of Physical Machine in IaaS Cloud Environment,” in *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, 2015, pp. 302–305.

[15] B. S. Lee, S. Yan, D. Ma, and G. Zhao, “Aggregating IaaS Service,” in *2011 Annual SRII Global Conference*, 2011, pp. 335–338.

[16] G. Liu, “Research on independent SaaS platform,” in *2010 2nd IEEE International Conference on Information Management and Engineering*, 2010, pp. 110–113.

[17] M. Armbrust *et al.*, “A view of cloud computing,” *Commun. ACM*, vol. 53, no. 4, p. 50, Apr. 2010.

[18] W. Shi and S. Dustdar, “The Promise of Edge Computing,” *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.

[19] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “CloneCloud: Elastic Execution Between Mobile Device and Cloud,” in *Proceedings of the Sixth Conference on Computer Systems*, New York, NY, USA, 2011, pp. 301–314.

[20] “Docker: Lightweight Linux Containers for Consistent Development and Deployment.” [Online]. Available: http://delivery.acm.org.miman.bib.bth.se/10.1145/2610000/2600241/11600.html?ip=194.47.1

29.64&id=2600241&acc=ACTIVE%20SERVICE&key=74F7687761D7AE37%2EA87D2C672813D63B%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1537326185_c21c4ba369d3f1d943e3b84a721438de. [Accessed: 18-Sep-2018].
[21] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee, "A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, 2016, pp. 117–124.
[22] "SMART-FI-D3.1-SMART-FI-Urban-Data-Characterization_v1.1.pdf.".
[23] "sysbench(1) — sysbench — Debian unstable — Debian Manpages." [Online]. Available: https://manpages.debian.org/unstable/sysbench/sysbench.1.en.html. [Accessed: 15-May-2019].
[24] "sysbench-manual.pdf." [Online]. Available: http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf. [Accessed: 14-May-2019].
[25] "Real Time Streaming Protocol," *Wikipedia*. 04-Jun-2018.
[26] *Thesis Demonstration Video.mp4*. Available: https://drive.google.com/file/d/1AXeCQOxYwSrI9vrXV3YHTGUC7mti61g6/view
[27] T. Long, "Measuring The Impact of Docker on Network I/O Performance."
[28] "Top 10 Benefits of Docker - DZone DevOps." [Online]. Available: https://dzone.com/articles/top-10-benefits-of-using-docker. [Accessed: 16-Jun-2019].
[29] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Philadelphia, PA, USA, 2015, pp. 171–172.