

ENABLING INDUSTRIAL SENSORS FOR EDGE COMPUTING

**Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in
Electronics and Communication Engineering**

By

MANGIPUDI SAI KRISHNA SRI

(Reg no 37130238)



SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI
CHENNAI – 600119**

APRIL - 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **MANGIPUDI SAI KRISHNA SRI (37130238)** who carried out the Project entitled **"ENABLING INDUSTRIAL SENSORS FOR EDGE COMPUTING"**. Under the guidance of **E.ANNA DEVI,M.E.,Ph.D.**, during the period NOV 2020 to APRIL 2021

INTERNAL GUIDE

E.ANNA DEVI,M.E.,Ph.D.,

HEAD OF THE DEPARTMENT

Dr.T.RAVI ,M.E.,Ph.D

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **MANGIPUDI SAI KRISHNA SRI (37130238)** Here by declare that the Project report entitled **ENABLING INDUSTRIAL SENSORS FOR EDGE COMPUTING** is done under the guidance of **E.ANNA DEVI,M.E.,Ph.D.**, is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in **ELECTRONICS AND COMMUNICATION ENGINEERING**.

DATE:

PLACE:

SIGNATURE OF CANDIDATE.

(M.SAI KRISHNA SRI 37130238)

ACKNOWLEDGEMENT

I pleased to acknowledge my sincere thanks to board management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY for their kind encouragement in doing this internship and for completing it successfully. I am grateful to them.

I convey my thanks to Dr.N.M.NANDHITHA,M.E.,Ph.D., Dean, school of Electrical and Electronics engineering and Dr.T.RAVI,M.E.,Ph.D., Head of the Department, Electronics and Communication engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide E.ANNA DEVI,M.E.,Ph.D., for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my Project.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Electronics and Communication Engineering who were helpful in many ways for the completion of the project.

ABSTRACT

IIoT(industrial internet of things) is the extension of the use of internet of things in industrial sectors and application Edgex is the platform which is created for the development of the industrial IOT Enabling industrial sensors for edge computing is the trending technology where edge enables intelligence (AI) and performs autonomous operations at the edge. Edgex is the platform which is created for the development of the industrial IoT. IoT sensors are used to monitor equipment and overall performance in industries. The services and layers of the Edgex provide a two ways transformation engine between edge device and cloud. Edge computing leads to lower latency by reducing the amount of data flowing to and from the primary network.

The final destination of this project is to provide an unhindered temperature monitoring service that has been established successfully using edgex in edge computing. The communication between EdgeX and Sensors were established successfully. The rules were created and respective triggered actions took place.

TABLE OF CONTENT

CHAPTER	TITLE	PAGE.NO
	ABSTRACT	iv
	LIST OF FIGURES	vii
	LIST OF TABLES	ix
1	INTRODUCTION	1
2	LITERATURE SURVEY	2
2.1	LITERATURE REVIEW	2
2.1.1	STORAGE FACILITIES USING ATMEGA 328 MICROCONTROLLER	2
2.1.2	ROTOR TEMPERATURE OF FLIGHT'S ENGINE	2
2.1.3	FOOD INDUSTRY	2
2.1.4	TEMPERATURE MONITORING SYSTEM USING ESP8266 MODULE	2
2.1.5	HOME AUTOMATION	3
3	AIM AND SCOPE OF THE PROJECT	4
3.1	AIM	4
3.2	OBJECTIVE	4
3.3	PROPOSED SYSTEM	4
3.4	SCOPE	5
4	EXPERIMENTAL OR MATERIALS AND METHODOLOGY USED	6
4.1	HARDWARE REQUIREMENTS	6
4.1.1	RASPBERRY PI 3B+	6
4.1.2	TEMPERATURE SENSOR (SHT 20)	7
4.1.3	USB CONVERTER	8
4.2	SOFTWARE REQUIREMENTS	8
4.2.1	REDIS DATABASE	8
4.2.2	GIT	9
4.2.2.1	GIT RESPOSITORIES	10
4.2.2.2	REMOTE RESPOSITORIES (ON GIT HUB & BIT BUCKET)	10
4.2.3	ZEROMQ	10

4.2.4	CURL COMMANDS	11
4.2.5	CLOUD COMPUTING	11
4.2.6	IBM CLOUD	12
4.2.7	EDGE X FOUNDARY	14
4.2.7.1	INTRODUCTION	14
4.2.7.2	ARCHITECTURE	14
4.2.8	MODBUS	15
4.3	METHODOLOGY	18
4.3.1	RASPBERRY PI 3B BRING UP AND UBUNTU INSTALLATION	18
4.3.2	INSTALLING EDGEX-GO	21
4.3.3	MODBUS DATA RECEIVING	22
4.3.3.1	IBM CLOUD	23
4.3.3.2	SENDING DATA TO KUIPER FOR LOCAL ANALYSIS	26
4.3.4	DATA FLOW	28
4.4	HARDWARE TESTING 28	28
4.4.1	RASPBERRY PI 3B	28
4.4.1.1	RASPBERRY PI GPIO PINS PYTHON PROGRAM TO VERYFY LED	28
4.4.1.2	CONNECTIVITY	29
4.4.2	MODBUS TESTING	29
4.4.2.1	MODBUS CONFIGURATION	30
4.4.2.2	READING VALUE	30
4.4.2.3	DECODING MODBUS MESSAGE	30
4.5	PROCEDURE	32
4.5.1	MODBUS IMPLEMENTATION	32
4.5.2	SENDING DATA TO CLOUD	34
4.5.3	SENDING DATA TO KUIPERS	36
5	RESULT	41
6	CONCLUSION	43
	REFERANCES	44

LIST OF FIGURES

FIGURE.NO	TITLE	PAGE.NO
4.1.1.1	RASPBERRY PI 3B+	7
4.1.2.1	TEMPERATURE SENSOR(SHT20)	7
4.1.3.1	USB CONVERTER	8
4.2.6.1	IBM CLOUD	13
4.2.7.2.1	ARCHITECTURE OF EDGE X FOUNDRY	14
4.2.8.1	MODBUS	16
4.3.1.1	RASPBERRY PI IMAGE	18
4.3.1.2	CHOOSING OS TO INSTALL	19
4.3.1.3	GETTING SETUP WITH WIFI	20
4.3.3.1.1	REGISTER YOUR DEVICE	25
4.4.1.1.1	RASPBERRY PI GPIO PINS	29
4.5.1.1	EDGE X - MICROSERVICES	32
4.5.1.2	UPLOADING DEVICE PROFILE	33
4.5.1.3	POST COMMAND	33
4.5.1.4	GET COMMAND	34
4.5.2.1	SENDING DATA TO CLOUD	34
4.5.2.2	STATUS OF THE DATA RECEIVED IN THE CLOUD	35
4.5.2.3	RECENT EVENTS IN CLOUD	36
4.5.3.1	ZERO MQ	36
4.5.3.2	BUILDING EDGE X-GO	37
4.5.3.3	KUIPER-SERVER	38
4.5.3.4	RUNNING STREAMS AND RULES	38
4.5.3.5	REDIS-SERVER	39
4.5.3.6	RUNNING APPLICATION SERVICE CONFIGURATION	39
4.5.3.7	RUNNING EDGE X-GO	40
5.1.1	SET UP	41
5.1.2	VISUAL REPRESENTATION OF TEMPERATURE AND HUMIDITY	41
5.1.3	TEMPERATURE IS > 30 DEGREE	42

LIST OF TABLES

TABLE.NO	TITLE	PAGE.NO
4.2.8.1	MODBUS MEMORY AREA	17
4.4.2.1	MODBUS TESTING	30
4.4.2.3.1	REQUEST SENT TO SENSOR	31
4.4.2.3.2	RESPONSE RECEIVED FROM SENSOR	31

CHAPTER 1

1 INTRODUCTION

EdgeX Foundry is the latest technology that incorporates both Edge Computing and the Internet of Things into a single platform making life much easier for IoT developers. Temperature Monitoring System can be implemented in a variety of use cases ranging from industrial temperature monitoring to household temperature monitoring. Taking care of abnormal rise in temperature manually is a very hectic and time-consuming task, hence the aim of this project is to build an automated temperature monitoring system using edge computing and IoT.

This project focuses on industrial temperature monitoring, where a 24/7 automated temperature monitoring system functions, pushes the data into a cloud, views the real-time data using a user interface and also alerts the required when there are anomalies/threshold breaches in the data thus reduces human work/monitoring to a high extent.

The key feature of this project is the implementation of EdgeX as it is highly scalable, is at the edge of the entire system and is secure. The UI and alerting system are the user side enhancements to solve the purpose of this project. This project is focused on implementing industrial temperature monitoring systems. Thus, producing an economically feasible and functionally scalable model to monitor the temperature in industries.

Edge gateway software development from scratch requires a lot of time. It requires more effort for design, development, and testing. Whereas open source EdgeX framework for Edge Computing is already providing all microservices required for Edge Gateway functionality. Customizing the applications for device connectivity, cloud connectivity and local analytics involves less efforts. Open source EdgeX framework is chosen for developing Edge Gateway software.

CHAPTER 2

2 LITERATURE SURVEY

2.1 LITERATURE REVIEW

The temperature monitoring system is not that rare in the industry and has various implementations and projects developed on that but in this project, the integration of EdgeX and the full-stack system makes it much more efficient, reliable, scalable and independent.

2.1.1 STORAGE FACILITIES USING ATMEGA 328 MICROCONTROLLER

[3] is the most closely related existing system when compared with the proposed system. It monitors the temperature of the storage rooms in industries using an ATmega 328 microcontroller along with a wifi module and GPRS, the aim of the system is to monitor the temperature regularly and preserve the items in the room for a longer duration.

2.1.2 ROTOR TEMPERATURE OF FLIGHT'S ENGINE

In [1] the rotor temperature of the flight's engine is monitored to avoid malfunctioning of the engine. A series of 16 sensors are used to achieve the task. An alert is generated to the captain of the flight when the temperature reaches the threshold.

2.1.3 FOOD INDUSTRY

[8] describes how important is temperature monitoring in meat supply chains and in the food industry. An extensive temperature monitoring is done on the surroundings and on the meat itself and accordingly, vary the temperature of the surroundings to keep the meat fresh as long as possible.

2.1.4 TEMPERATURE MONITORING SYSTEM USING ESP8266 MODULE

[4] Uses an ESP8266 microcontroller with an in-built wifi module and a temperature sensor to monitor the temperature of the surroundings. The data is

collected and pushed to the cloud is an alert is triggered when the assigned threshold has been reached. A cloud dashboard is created for real-time monitoring.

2.1.5 HOME AUTOMATION

In [6] temperature monitoring is concentrated more towards Home Automation. The real-time temperature data of various rooms can be viewed through a UI along with past history. It uses WSN (Wireless Sensor Network) to collect the data and communicate with the cloud and other microcontrollers. It modulates the temperature by turning off/on the air conditioner whenever required.

CHAPTER 3

3 AIM AND SCOPE OF THE PROJECT

3.1 AIM

The aim is to build an automated industrial temperature monitoring system that can monitor its surrounding temperature at regular intervals, process the data, collect and store the data in a database, upload the data to a web-server, couple the web-server with a UI and prepare an alerting system. To reduce risk factor by implementing this real-time temperature monitoring system in households and factories/industries.

The key goal is to produce a system which works continuously without any hindrance and provides the correct real-time data as expected. To implement this system on a large scale in various use cases and expand the scope of this project. To make sure the system is working intact and data is visualized as required.

3.2 OBJECTIVE

The goal of the project is to create a Temperature Monitoring in Industries using EdgeX foundry in edge systems and push the data on to the cloud which can be further coupled with an app/webpage/alert system.

The major objectives are as follows:

1. Monitor temperature data without any hindrance-The sensor and software integration has to remain intact, sensor has to work continuously and successfully for long hours. The wiring should be precise so that the data does not change its path and reach elsewhere.
2. The analysis and actions on the data has to be really fast.
3. The data has to be pushed on to the cloud at regular intervals without any break.
4. User friendly and easily understandable User Interface.
5. For certain rules given by the user, action should be triggered or done before sending data to the cloud i.e., in edge computing.

3.3 PROPOSED SYSTEM

The “Enabling industrial sensors for edge computing” is a complete hardware and software integrated full-stack Internet of Things based system to make

temperature monitoring in industries a much easier task. The system is divided into three parts, the hardware segment followed by the interface segment and ending at the software segment. The first segment comprises, a raspberry-pi micro-computer which is connected with a SHT 20, temperature and humidity monitoring sensor. The raspberry-pi has the EdgeX software running in it.

The edgex-go, application services, kuipers run in the raspberry pi under redis database. The modbus is used to collect the data from the sensor and it is present in the device service layer and sends that data to the cloud. And also that data is local analysed and respective triggered actions are done in gpio device service through kuipers rules engine.

The proposed system is much more scalable and secure compared to the other systems/projects due to the implementation of EdgeX. Through EdgeX updating, the working of the system is much easier and scalable as 100 such systems can be updated at one go without having to modify the code individually on each system. The EdgeX uses a reverse-proxy during the communication of data making it much more secure than the other systems. EdgeX provides software and hardware enhancements to the system through its various micro-services which lie under the skin of its service layers. The outcome of this system is to make temperature monitoring a completely automated process and visualize the monitored data.

3.4 SCOPE

Industrial grade temperature and humidity transmitter, SHT20, is monitored as part of this Project.

1. Read the temperature and humidity values of SHT20 Sensor.
2. Leverage EdgeX framework running on Raspberry Pi to provide gateway functionality.
3. Using EdgeX framework microservices, process the sensor data and send it to Cloud.
4. Actuated LEDs to ON/OFF when temperature readings cross threshold values.

CHAPTER 4

4 EXPERIMENTAL OR MATERIALS AND METHODOLOGY USED

4.1 HARDWARE REQUIREMENTS

4.1.1 RASPBERRY PI 3B+

The latest Raspberry Pi 3 Model B+ has a speedier 64-cycle 1.4GHz quad community processor, 1GB of RAM, faster twofold band 802.11 b/g/n/ac far off LAN, Bluetooth 4.2, and basically faster 300Mbit/s ethernet. It can be considered as a lone board PC that works on LINUX working system. The board not simply has tremendous heaps of features it moreover has planning speed making it proper for bleeding edge applications.

- 1.4GHz 64-cycle quad-focus ARM Cortex-A53 CPU (BCM2837)
- 1GB RAM (LPDDR2 SDRAM)
- On-board distant LAN - twofold band 802.11 b/g/n/ac (CYW43455)
- On-board Bluetooth 4.2 HS low-energy (BLE) (CYW43455)
- 4 x USB 2.0 ports
- 300Mbit/s ethernet
- 40 GPIO pins
- Full size HDMI 1.3a port
- Combined 3.5mm basic sound and composite video jack
- Camera interface (CSI)
- Show interface (DSI)
- microSD space
- VideoCore IV media/3D plans community @ 400MHz/300MHz

The Raspberry Pi equipment has advanced through a few forms that highlight varieties in the kind of the CPU, measure of memory limit, organizing backing, and fringe device support. The dual band distant LAN goes with estimated consistency accreditation, allowing the heap up to be arranged into completed outcomes with on a very basic level decreased far off LAN consistency testing, improving both cost and time to exhibit.

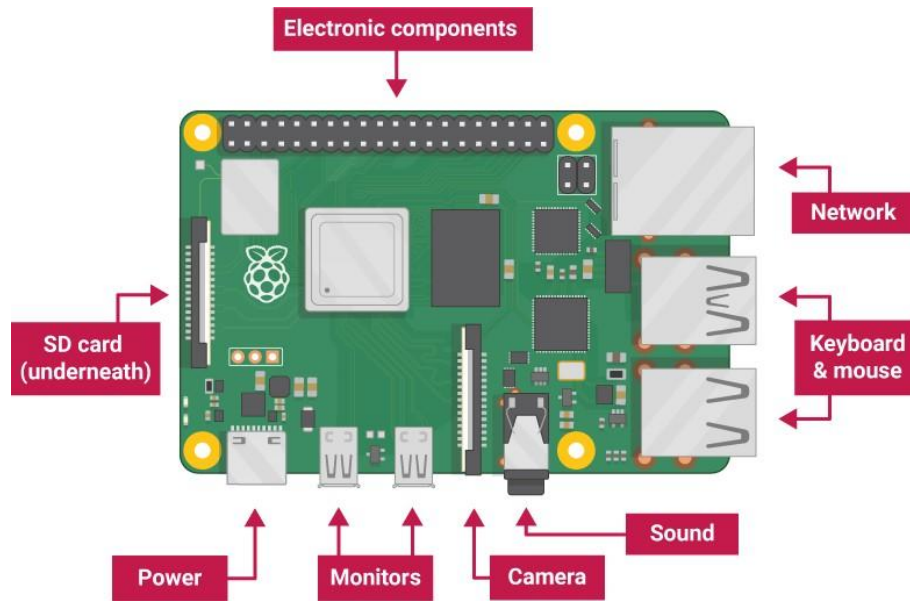


Fig.no 4.1.1.1 Raspberry Pi 3B+

4.1.2 TEMPERATURE SENSOR (SHT 20)



Fig.no 4.1.2.1 Temperature Sensor (SHT 20)

SHT20 receives modern evaluation chip and high-exactness imported SHT20 temperature and humidity sensors to guarantee amazing dependability, high precision and similarity. RS485 equipment interface (with lightning assurance plan), the convention layer is suitable with the standard mechanical Modbus-Rtu convention. This item coordinates MODBUS convention with normal convention. Clients can pick correspondence conventions by their decision. The common protocol has an automatic upload function connecting RS485 will automatically provide temperature and humidity values through serial port adjustment tool.

FEATURES

- Support MODBUS RTU convention
- RS485 upholds 1000 meters correspondence
- Standard DIN35 mounting rails
- High Accuracy

4.1.3 USB CONVERTER



Fig.no 4.1.3.1 USB Converter

Sensor SHT20 communicates with RS-485 protocol. As part of the Project, RS-485 to USB converter is used to receive and send values from Sensor. Converter used here is modern USB to RS485 Bidirectional half duplex sequence line converter TVS transient assurance.

4.2 SOFTWARE REQUIREMENTS

4.2.1 REDIS DATABASE

Redis is a publicly accessible (BSD approved), purpose-built information structure stock, utilized as an information storage, cache, and message agent. Redis gives data structures such as sets, strings, bitmaps, lists, hashes, sorted sets with range queries, geospatial indexes, streams and hyperloglogs.

You can run nuclear procedure on these sorts, such as adding to a string; expanding the worth in a hash; pushing a segment to a rundown; association and contrast, registering set crossing point; or getting the part with the most noteworthy commendable situation in an arranged set.

Redis works with a purpose-built dataset. Contingent upon your need, you can endure through your information either by intermittently unloading the dataset to plate or by affixing each order to a circle based log. You can likewise incapacitate

ingenuity on the off chance that you simply need an element rich, arranged, in-memory cache. Linux and operating system X are the two working frameworks where Redis is made and attempted the most, and we recommend using Linux for arrangement.

Other features:

- Transactions
- Keys with a restricted opportunity to-live
- Lua scripting
- Pub/Sub
- LRU eviction of keys
- Programmed failover

You can utilize Redis from most programming languages.

Redis is written in ANSI C and works in most POSIX frameworks like Linux, OS X, and *BSD, without outside conditions. Linux and OS X are the two working frameworks where Redis is created and tried a lot, and we suggest utilizing Linux for sending. Redis databases can work in Solaris-determined systems like SmartOS, however the help is best exerted. No authority is supported to build Windows.

4.2.2 GIT

Git is the software utmost normally utilized version control system. Git records the progressions you change in documents, so you can have a track of what has been done, and you can return to explicit forms should you at any point if necessary. Git additionally makes coordinated effort easier, enabling changes by multiple people to all be converged into one source.

Git is programming that runs locally. Your documents and respective set of experiences are put away on the computer. You can likewise utilize online hosts (like GitHub or Bitbucket) to collect a duplicate of the records and respective correction history. Having a midway found where you can transfer your progressions

and download changes from others, empower you to work together very effectively with different designers. Git may naturally combine the changes present, so two independent persons can even work on various pieces of a similar document and after that union those progressions without the loss of independent work.

4.2.2.1 GIT REPOSITORIES

A Git repository (or repo for short) consists of the entirety of the undertaking records and the whole amendment record of previous. You'll take a normal organizer of documents (like a site's root envelope), and advise the git to make it a stock. This makes a git subfolder, which contains the entirety of the git metadata for following changes.

4.2.2.2 REMOTE REPOSITORIES (ON GITHUB & BITBUCKET)

Gathering a copy of Git repo with an internet host (such as GitHub or Bitbucket) provides you a halfway found where the data can transfer your progressions and download changes from others, allowing one to team up more effectively with different engineers. After having a faroff storehouse set up, transfer (push) the documents and amendment record to it. After another person makes changes to a distant repository, you can download (pull) their progressions into one local repo.

4.2.3 ZERO MQ

ZeroMQ is a fantastic tool that is truly amazing and can scale to extremely huge systems. Getting started with ZeroMQ can be very difficult from a novice's viewpoint. The reason for this guide is to give you an expansive thought of what ZeroMQ can do and to assist you with choosing if ZeroMQ is the informing library for you.

ZeroMQ is an asynchronous network informing library known for its elite. It's expected use is for disseminated frameworks just as simultaneous frameworks. In synopsis, ZMQ permits you to send messages (paired information, serialized information, basic strings, and so forth) over the organization through different techniques like TCP or multicast just as between measures. ZeroMQ gives an entire

slew of language APIs which run on most working frameworks and permits you to convey consistently between a wide range of projects. It additionally gives an assortment of examples, for example, demand answers and distribute buys which help you in making and organizing your organization.

4.2.4 CURL COMMANDS

CURL is a pc programming project giving a library (libcurl) and command-line tool (curl) for moving information utilizing different network protocols. cURL is a one line command-line tool for acquiring or providing documents utilizing URL syntax. Since cURL uses the library Libcurl, it associates every convention that Libcurl supports.

CURL upholds HTTPS and performs SSL testament check of course when a safe convention is determined like HTTPS. When cURL associates with a distant worker by means of HTTPS, it will acquire the far off worker testament, at that point clarified against its CA declaration record the legitimacy of the far off worker to guarantee the far off worker is the one it professes to be like. Few CURL packages are packed with CA certificate stock documents. There are few choices to indicate a CA certificate for example, --cacert and --capath. The --cacert choice may be utilized to indicate the area of the CA certificate stock file. In the platform of windows, if a CA certificate file is not determined, CURL will look for a CA certificate filename "curl-ca-bundle.crt" at the accompanying request:

1. Index where the cURL program is found.
2. Current working index.
3. Windows framework catalog.
4. Windows catalog.
5. Indexes determined in the %PATH% climate factors.

CURL command returns a blunder message if the far off worker is utilizing a self-marked endorsement, or if the distant worker declaration isn't endorsed by a CA recorded in the CA certificate document. -k or --insecure utilized to skip

declaration check. On the other hand, if the automatic server is believed, the automatic server CA certificate can be added to the CA certificate store file.

4.2.5 CLOUD COMPUTING

Cloud computing is the conveyance of dissimilar administrations in the Internet. These assets incorporate tools and applications like servers, data storage networking, databases, and software. Up to the date when an electronic device has access to the internet, it approaches the information and the product projects to access it.

Advantages of cloud computing

- Diminished IT costs.
- Moving to distributed computing may change the expense of overseeing and storing up your IT frameworks. ...
- Versatility. ...
- Flexibility of work practices. ...
- Collaboration efficiency. ...
- Business continuity. ...
- Admittance to programmed refreshes.

CLOUD SERVICES

Cloud services are services available useful through a distant shared computing worker as opposed to an on location worker. These versatile courses of action are supervised by an outcast and give customers permission to figure out organizations, for instance, examination or framework organization through the web. There are three principal administration types of cloud computing – Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

CLOUD APP SERVICE

Cloud application service is created to send information to IBM Watson cloud stage.

4.2.6 IBM CLOUD

IBM Cloud incorporates infrastructure as a service (IaaS), software as a service (SaaS) and platform as a service (PaaS) offered through open, private and

half and half cloud conveyance models, notwithstanding the segments that make up those mists.

It gives a full-stack, public cloud stage with different items in the inventory, including choices for process, stockpiling, organizing, start to finish engineer answers for application advancement, testing and organization, security the board administrations, customary and open source information bases, and cloud-local administrations. for application testing and deployment, security management services, development, traditional, and cloud-native services , and open source databases.

IBM Cloud is a simple and secure route for information stockpiling. It has an information relocation highlight with high security which expands the effectiveness of work. With IBM Cloud IaaS, associations can send and get to user interface IT assets, for example, stockpiling, register force, and systems administration over the web. For compute, organizations can choose between virtual servers or uncovered.

FEATURES

- Move information quick
- Moderate
- Scalable and flexible
- Simple process
- End to end encrypted



Fig.no 4.2.6.1 IBM Cloud

4.2.7 EDGE X FOUNDRY

4.2.7.1 INTRODUCTION

EdgeX foundry is the software platform between sensors or devices and the cloud (IT System) EdgeX is used to screen actual world things, send guidelines to sensors, gather information from sensors, get the information across the haze up to the cloud where it could be stored, collected, examined, and transformed into data, incited, and reacted according to the need.

EdgeX is built to support industrial needs like in building automation, water treatment, consumer IoT etc. In the edgeX every part of the microservice can be changed according to the need by other microservices. EdgeX is secure and can be managed easily.

4.2.7.2 ARCHITECTURE

EdgeX is an assortment of microservices. These microservices are coordinated in 4 layers and 2 assistance layers. The 4 layers are Core service layer, application service layer, Support service layer and Device service layer. The 2 system services layers are security system service and system management.

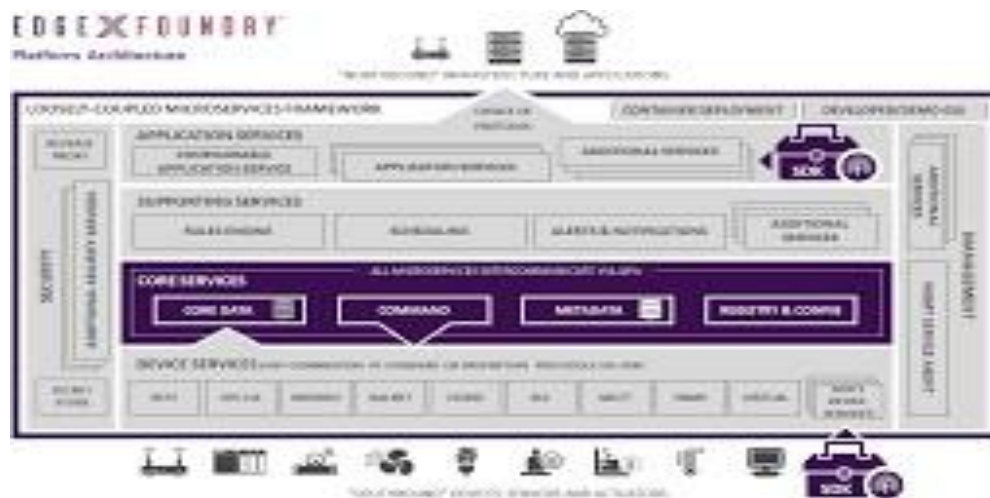


Fig.no 4.2.7.2.1 Architecture of edge x foundry

CORE SERVICE LAYER

In this service layer where the information on what things are associated and what information is coursing through the edgex. It consist of following microservices:

- Core data microservice collects sensor data from device service layer ,it also provides security and protection of data collected while the data at edge.
- Core command facilitates and controls actuation requests.
- Meta data gives the ability to register new devices and combine them with their own id services.
- Registry and configuration

SUPPORT SERVICE LAYER

This service layer requires core service in order to function.This is an optional layer, it consists of application microservices like logging,scheduling and data clean up.

- Rules engine is used to set rules according to user need and create reactions according to rule.
- Scheduling microservice
- Logging is used for log entries etc
- Alerts and Notification

APPLICATION SERVICE LAYER

This layer is used to send data from the edge system to the cloud or required end point.Application service layer is based on the rules of functions pipeline.

Function pipeline is an assortment of capacities that sends messages in the predetermined order.The first capacity present in the pipeline is triggered. It closes when the message has gone through every one of the capacities and set into the sink..

DEVICE SERVICE LAYER

Device services are the edge connectors which are connected to sensors or devices.The device service layer connects more than one device or sensors at a time.The device service converts the data received from the sensors into the edgex information structure and sends that information to the center assistance layer.

4.2.8 MODBUS

Modbus is an application layer communication protocol for exchanging data between electronic devices over serial lines (RS 232 or RS 485) or Ethernet, and it's widely used in process and factory automation.

Modbus is an open protocol and can be used by any organization or individual, Schneider Electric USA holds the trademark.

Modbus gives a typical language to gadgets and hardware to speak with one and another. For example, Modbus enables devices on a system that measures temperature and humidity connected on the same network and communicates the results to a supervisory computer or Microcontroller.

Modbus serial protocol is a single master, multiple slaves protocol, where each slave serves the requirement of the master on its request. Modbus TCP/IP is a client and server protocol where the master is the server and its slaves are the clients.

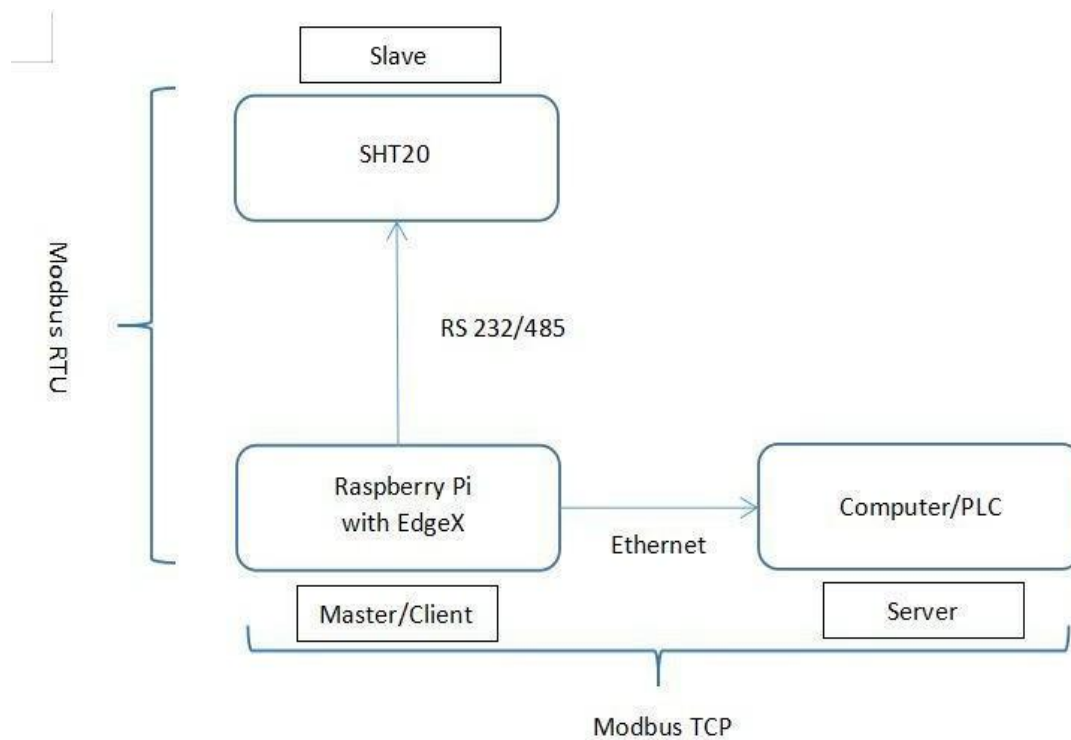


Fig.no 4.2.8.1 ModBus

Table.no 4.2.8.1 Modbus Memory Area

Address Range	Access	Object Type	Size
00001-09999	Read/Write	Coil (Discrete Output)	1 bit
10001-19999	Read	Discrete Input	1 bit
30001-39999	Read	Input Register	16 bits
40001-49999	Read/Write	Holding Register	16 bits

The Modbus memory area is split into four parts where the first is coils (discrete output) which is of 1 bit in size and stores the actuation requests such as to open and close a valve. Both read and write functions can be performed on coils i.e reading the current status of valve and assigning open or close operations on the valve.

The second is the discrete input which is also of size one bit and stores on/off status, like that of a switch. Only read operation can be performed on this memory area and thus stores on/off status.

Input register is the third part of the memory area, it is of size 16 bits, and stores the values read from sensor devices such as SHT20. It is a read only memory area and thereby data present in this memory area can only be read and not modified. Thus, temperature, humidity, current and power sensor devices use the Input register memory area since the data generated from these sensor devices cannot be modified.

The fourth part is the holding register. It is also of size 16 bits, stores values read from sensors such as water valves. Both read and write operations can be performed on the memory register thereby data can be read and modified as well. For example, consider a water valve when completely open is flowing at a rate of 1L for every 10 seconds. The data Valve is 100% open is stored in the holding

register and if rate has to decrease later it can be modified on the holding register itself.

4.3 METHODOLOGY

4.3.1 RASPBERRY PI 3B BRING UP AND UBUNTU INSTALLATION

- An Ubuntu installed PC (will be called the host in this series)
- A microSD card (4GB least, 8GB suggested)
- A PC with a microSD card drive
- A Raspberry Pi 3 b
- A miniature USB power interface (USB-C for the Pi 4)
- A Wi-Fi association or an ethernet interface with a web affiliation

RPI uses a micro SD card as its main disk storage. Ubuntu server 20.10 image file can be downloaded from Canonical and the file can be flashed to a micro SD card.

Step 1:

insert the microSD card into your computer.

Install the image for raspberry os ([Raspberry Pi Imager for Ubuntu](#))



Fig.no 4.3.1.1 Raspberry Pi Image

Now click choose os and click the ubuntu option and choose the ubuntu version to download.

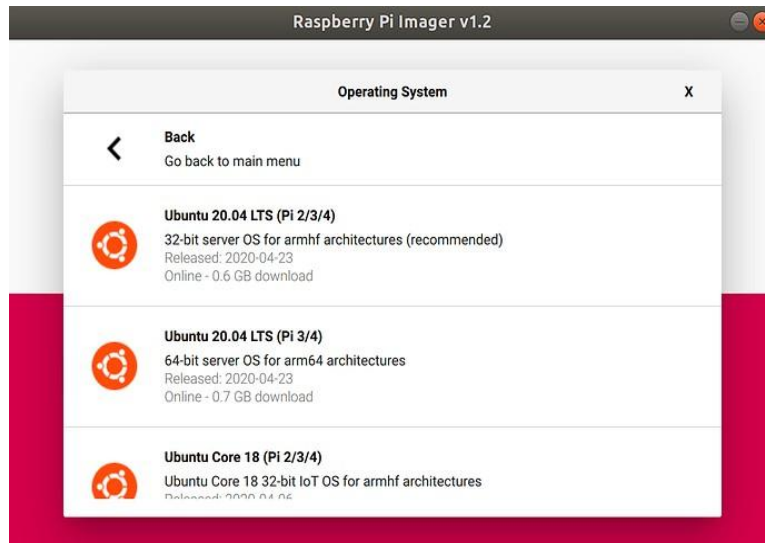


Fig.no 4.3.1.2 Choosing OS to install

Choose the sd card option microSD card which is inserted and click write.

Step2:

There are two different ways to get your Pi associated with the web:

- The first is to associate your Pi to your switch with an ethernet cable. For the present circumstance, you can keep away from this movement and go to the accompanying one in the instructional exercise.
- The second requires a local Wi-Fi network that both your PC and Pi can be associated with. We will change records you just downloaded on your SD card to ensure your Pi can interface with the Wi-Fi network at boot.

Getting arrangement with Wi-Fi

With the Sd card actually embedded in the PC, open the document supervisor and go to framework boot and select the alternative organization config. Change the organization config record to add your Wi-Fi capabilities. A model that as of now exists is remembered for the document, you can just take it. To do all things considered, uncomment (dispense with the "#" around the beginning) and modify the going with lines:

wifis:

wlan0:

dhcp4: true

optional: true

access-points:

<wifi network name>:

password: "<wifi password>"

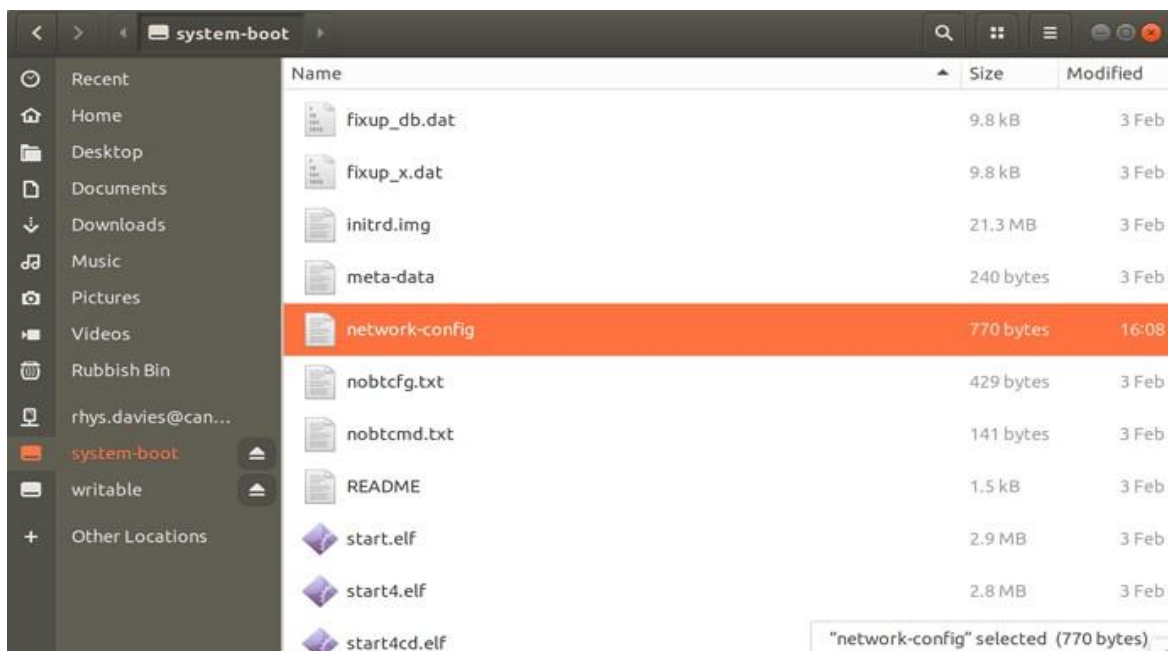


Fig.no 4.3.1.3 Getting Setup with WIFI

Save the record and concentrate the card from your laptop. During the primary boot, your Raspberry Pi will attempt to associate with this organization. It will bomb the first run through around. Reboot as `sudo reboot` and that will work.

Step3 :

When cloud-init completes sign in utilizing "ubuntu" both the secret word and login ID. Following a moment, Ubuntu on your Raspberry Pi will be completely booted and associated with the organization.

Step4 :

Connecting automatically the raspberry pi 3, two things are necessary.

They are 1.IP address

2. SSH client

->For ip address use this command in ubuntu

```
arp -na | grep -i "b8:27:eb"
```

->SSH client

```
ssh ubuntu@<Raspberry Pi's IP address>
```

Comment y or yes .

Step 5 :

Installing desktop.

->For ensuring packages and updating to latest versions

->For installing Ubuntu desktop, apt install can be used .

->If installation is finished then reboot for installing desktop in raspberry pi.

Now your new desktop will appear automatically.

4.3.2 INSTALLING EGDEX-GO

EdgeX Foundry is a merchant impartial open source project worked with by The Linux Foundation developing an average open design for IoT edge figuring. At the core of the venture is an interoperability system facilitated inside a full equipment and OS-free thinker reference programming stage to empower an environment of plug-and-play components that brings together the commercial center and speeds up the sending of IoT arrangements. This repository contains the Go implementation of EdgeX Foundry microservices.It additionally incorporates records for building the services, containerizing the services, and instating (bootstrapping) the services.

Before installing edgex the following are some prerequisites:

- Install go

->Download the go file.

->Then install the go through command.

-> we can remove the existing data and install the go in /usr/local

-> export the path

-> whether the go is installed properly or not, we can verify through checking the version of go.

- Installation of git
 - >install Git on Linux using the specific package manager of Linux distribution. Built from source, one can find the tarballs on kern.org.
- Installing the redis-database
 - >EdgeX Foundry uses Redis as the persistence mechanism for sensor data as well as metadata about the devices/sensors that are connected.
 - >to install Redis binaries into /usr/local/bin , make install.
 - > Make introduction will simply introduce pairs in your framework, yet won't arrange init contents and setup records in the suitable spot. This isn't required in the event that you simply need to play a bit with Redis, however in the event that you are introducing it the appropriate path for a creation framework, we have a content are installing it the proper way for a production system, we have a script that does this for Ubuntu systems.
 - >Change directory to the utils
 - > run the install-server.sh
 - >the content will ask you a couple of inquiries and will arrange all you require to run Redis appropriately as a foundation daemon that will begin again on system reboots.

One can be able to stop and start Redis database using the script named /etc/init.d/redis_<port number>, for instance /etc/init.d/redis_6379.

- Installing ZeroMQ
 - > several EdgeX Foundry services depend on ZeroMQ for communications by default.
 - >Download zeromq
 - >Unpack tarball package
 - >Install dependency
 - >Create make file
 - >Build and install
 - >Install zeromq driver on linux

4.3.3 MODBUS DATA RECEIVING

The data can move in two directions through modbus, from the sensor device to core services layer or from core services layer to sensor device. Let's first discuss the sooner, the modbus communicates with the sensor device such as SHT20

temperature monitoring sensor and sends the sensor data to the core data microservice present in the core services layer where it is stored in a database such as redis. In the latter approach, command microservice present in the core services layer sends data to the sensor device.

Consider an industrial automation system where an SHT20 temperature monitoring sensor device is used in a food storage facility and a compressor is used to regulate the temperature of the facility. These two sensor devices are connected to edgex via modbus device service.

The SHT20 sensor device communicates to edgex via modbus device service and pushes temperature data to the core data at the device services layer every 5 seconds. The data present inside the core data microservice is stored in a redis database and this data is then pulled by the application services layer via a message queue called ZeroMQ. Various functions are being performed on the data and the data is then sent to rules engine micro service present in the support services layer, where a Kuiper engine checks a set of conditions such as if the temperature data is greater than 26 actuate compressor to bring the temperature down to 24. This actuation command is sent from the rules engine to the command microservice present in the core services layer, from the command microservice this actuation is sent to modbus and the modbus communicates with the compressor to bring the temperature of the facility to 24-degree celsius.

4.3.3.1 IBM Cloud

- Create an IBM Cloud account
- Log into IBM cloud
- Create a Watson IoT Service instance

On the off chance that you have an Edge gadget creating MQTT information, you should send the IoT information to the cloud for constant alarms, time arrangement data set stockpiling, investigation and demonstrating. IBM Cloud provides an IoT ingestion service, Watson AI services, Cloud Object Storage and Watson Studio data science portal that can help developers manage their IoT data and find insights.

EdgeX gives App-Service-Configurable assistance as a simple method to begin with preparing information coursing through EdgeX.

This service uses the App Functions SDK and gives an approach to engineers to utilize design as opposed to incorporating independent administrations to use work in capacities in the SDK.

Connecting gadgets to Watson IoT platform involves registering the gadget with Watson IoT Platform and afterward utilizing the enrollment data to arrange the gadget to interface with Watson IoT Platform.

Stage 1: Register your gadget

Selecting a device includes setting up the contraption as a device type, giving the device a name, and providing device information. At that point give an association token or acknowledge a symbol that is created by Watson IoT Platform.

- In the IBM Cloud reassure, press Launch on the Watson IoT Platform administration subtleties page. The web of Watson IoT Platform comfort opens in another program tab .
- Select devices, from the menu board, in the Overview dashboard, and a while later snap Add Device.
- Make a gadget type for the gadget that you are adding. Each gadget that is associated with the Watson IoT Platform should be related with a gadget type. Gadget types are a bunch of gadgets that share similar qualities.
- Click Create device type.
- Enter a device name.
- Click Next to start the way toward adding your gadget with the chosen device type.
- Enter a device id. The device ID is used to identify the device in the Watson IoT Platform dashboard and is also a required parameter for connecting the device to Watson IoT Platform.
- Next give a verification token or acknowledge a consequently produced token.
- Check the data is right and afterward click Add to add the association.

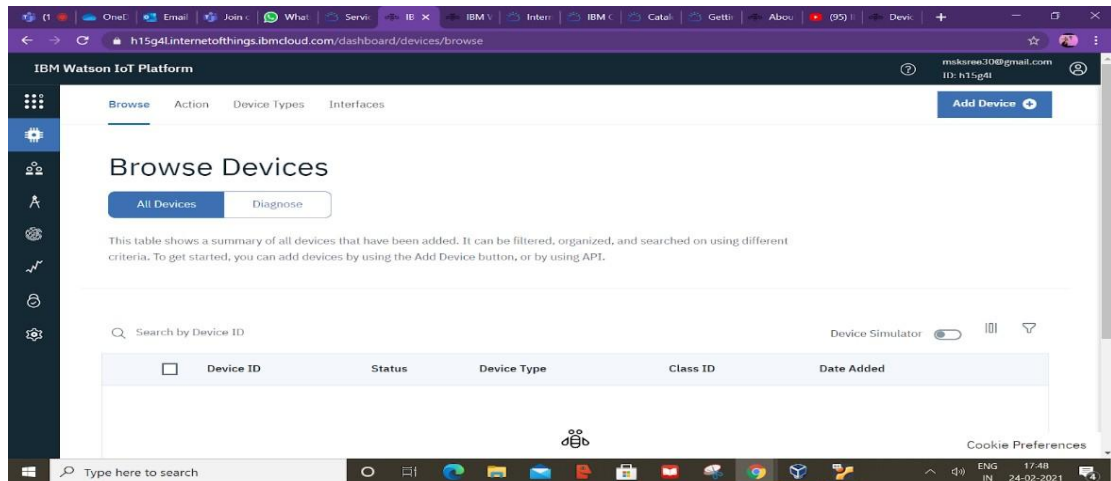


Fig.no 4.3.3.1.1 Register your device

ID, device ID, device type, and authentication token are needed to configure the device to connect to Watson IoT Platform.

Step 2: Connect your device to Watson IoT Platform

For connecting a device to Watson IoT Platform:

- Download this configuration.toml and edit the [Writable.Pipeline.Functions.MQTTSend.Addressable] section.

```
[Writable.Pipeline.Functions.MQTTSend.Addressable]
```

```
Address= "<orgid>.messaging.internetofthings.ibmcloud.com"
```

```
Port= 1883
```

```
Protocol= "tcp"
```

```
Publisher= "d:<orgid>:<devicetype>:<deviceid>"
```

```
User= "use-token-auth"
```

```
Password= "<Authentication-Token>"
```

```
Topic= "iot-2/evt/status/fmt/json"
```

Enter Watson IoT Organization (6 character <orgid>), <DeviceType>, <Device ID> and <Authentication token>.

Then after running every platform that is redis database ,edgex-go, device-virtual, application service the data can be seen in the IBM cloud .

The data will be received every second in json format . Every second data is updated in the cloud.The communication between EdgeX and Sensors will be established successfully.The rules were created and respective triggered actions were taken place.Creating an account in IBM cloud.The data was analyzed and pushed to the cloud.

4.3.3.2 SENDING DATA TO KUIPER FOR LOCAL ANALYSIS

In edge computing, essentially gathering sensor information is just essential for crafted by an edge stage like EdgeX. Another critical control of an edge stage is to have the alternative to:

- Dissect the approaching sensor information locally
- Quickly act on that examination Edge or close by assessment is the readiness that plays out an evaluation of the sensor data accumulated at the edge ("locally") and prompts actuator exercises.

Edge analytics is seeing the edge data and being able to make requests to act on what is seen. While EdgeX provides a rules engine service as its reference implementation of local analytics, app services and its data preparation capability allow sensor data to be streamed to any analytics package.

Edge analytics are important for two reasons:

- A few choices can't stand to sit tight for sensor assembled data to be dealt with back to an endeavor or cloud and have a returned response.
- Also, some frameworks of edge are not generally associated with the endeavor or cloud – they have discontinuous times of network.

Nearby investigation permits frameworks to work autonomously, in any event for certain stretches of time.EdgeX is worked to act locally on information it gathers from the edge. At the end of the day, occasions are prepared by neighborhood investigation and can be used to prompt movement back to the sensor/gadget.

As application administrations get ready information for utilization by north side cloud frameworks or applications, application administrations can measure and get EdgeX occasions to any analytics package. By default, EdgeX ships with a kuiper rules engine by EMQ x.

For sending data to the kuipers rules engine the respective application service configurable is needed, i.e., rule_engine configuration.toml file.

The following are necessary to install to send the data to the kuiper rules engine.

- Edgex-go
 - Already installed in raspberry pi.
- Redis database
 - > The first step is to download Redis. For this, you will be utilize the Ubuntu package manager which you should update first.
 - > launch Redis using command redis-server.
 - > Test whether Redis is running correctly, you can use the **interface** used to communicate with Redis (redis-cli).
- Application service configurable
 - Install Application service configurable using git repository and make build and run the app-service-configurable file with respective configuration.toml file in res directory.
- kuiper
 - Install the kuiper using the git repository and make build with edgex and run the kuiper server continuously.
 - > create stream
 - > create rule and set action
 - > build again

Whether the data is received or not can be checked in the log directory in stream.log file and action takes place according to the rules provided by the user. By running the edgex-go ,redis database, application-service-configurable in RPI simultaneously , the data will be received by the kuiper and respective actions will take place.

4.3.4 DATA FLOW

EdgeX's essential occupation is to gather information from sensors and gadgets and make that information accessible to north side applications and frameworks. Information is gathered from a sensor by a gadget administration that talks about the convention of that gadget. Modbus gadget administration would speak with sensor SHT20 to get the temperature and humidity readings.

->The device service makes an interpretation of the sensor information into an EdgeX occasion object and provides the occasion object to the center information administration by means of REST interchanges.

->Center information keeps sensor data occasions on a message subject destined for application administrations. ZeroMQ is utilized as the informing foundation as a matter of course.

->The application administration gathers the information depending on the situation and pushes the information to an endpoint. It can likewise channel, enhance, pack, scramble or perform different capacities on the occasion prior to passing it to the endpoint. The endpoint could be a HTTP/S endpoint, a MQTT point, a cloud framework (cloud theme), and so on

4.4 HARDWARE TESTING

4.4.1 RASPBERRY PI 3B

4.4.1.1 RASPBERRY PI GPIO PINS

Pi comes with 40 general purpose input and output (GPIO) pins. The numbering of the GPIO pins is not in order. For instance, GPIO16 is at pin number 36. We can check the pin layout using the pinout command from Ubuntu.

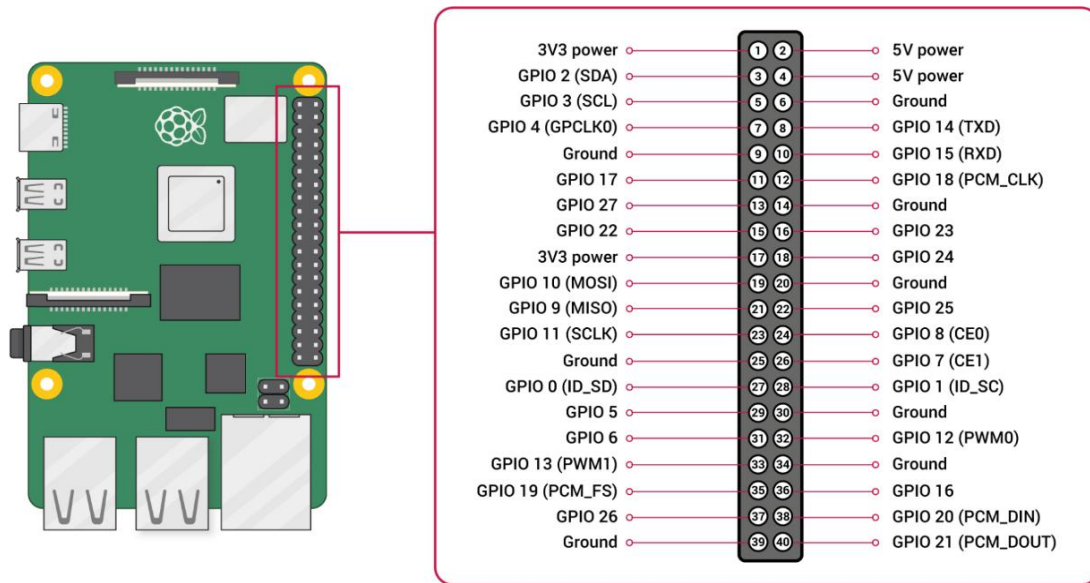


Fig.no 4.4.1.1.1 Raspberry pi GPIO pins

4.4.1.2 PYTHON PROGRAM TO VERIFY LED CONNECTIVITY

Objective is to verify the breadboard and LEDs connectivity without using EdgeX in the first place. I have tested the LEDs functionality with the python program. Package RPi.GPIO gives a class to control the GPIO on a Raspberry Pi.

4.4.2 MODBUS TESTING

Goal of the PoC is to read the values from the Sensor and send it to the cloud. Also actuate LEDs when the values reach configured threshold numbers. For this activity, temperature and humidity values are read from the Input registers mentioned in the below table.

Table.no 4.4.2.1 Modbus Testing

Register Type	Register Address	Data Content	Number of Bytes
Input Register	0x0001	Temperature value	2
	0x0002	Humidity value	2
Holding Register	0x0101	Device address (1~247)	2
	0x0102	Baud rate 0:9600 1:14400 2:19200	2
	0x0103	Temperature correction value (/10) -10.0 ~ 10.0	2
	0x0104	Humidity correction value (/10) -10.0 ~ 10.0	2

Sensor SHT20 sends temperature and humidity values with the multiple of 10. After receiving the values, we need to divide them by 10 to get the correct readings.

4.4.2.1 MODBUS CONFIGURATION

Sensor SHT20 communicates with RS-485 protocol. As part of the PoC, RS-485 to USB converter is used to send and receive values from Sensor. Converter used is Modern USB to RS485 Bidirectional half duplex sequential converter TVS transient security.

4.4.2.2 READING VALUE

Objective is to check the functionality of the sensor without using EdgeX in first place. So that we can validate if Sensor is working as expected or not. I have tested the Modbus messages from sensors with a C program. In C program, libmodbus library is used, which provides the APIs to send and receive messages in Modbus protocol.

4.4.2.3 DECODING MODBUS MESSAGE

Decode for a pair of messages sent to and received from the Sensor.

Request Sent to Sensor: 01 04 00 01 00 02 20 0B

Table.no 4.4.2.3.1 Request Sent to Sensor

Sent	decoding
01	Slave id
0	function,read_input_registers
00 01	Register address
00 02	No of registers to read
20 0B	CRC

Table.no 4.4.2.3.2 Response received from Sensor

Response received from Sensor: 01 04 04 01 4A 02 1A 5B 05

Receive	decoding
01	Slave id
04	function,read_input_registers
04	Bytes count
01	Data 1, temperature
02	Data 2,temperature
5B 05	CRC

Temperature value received is 0x014a, which is 330 in decimal. SHT20 sends the values in multiples of 10, so temperature is 33.0 degree.

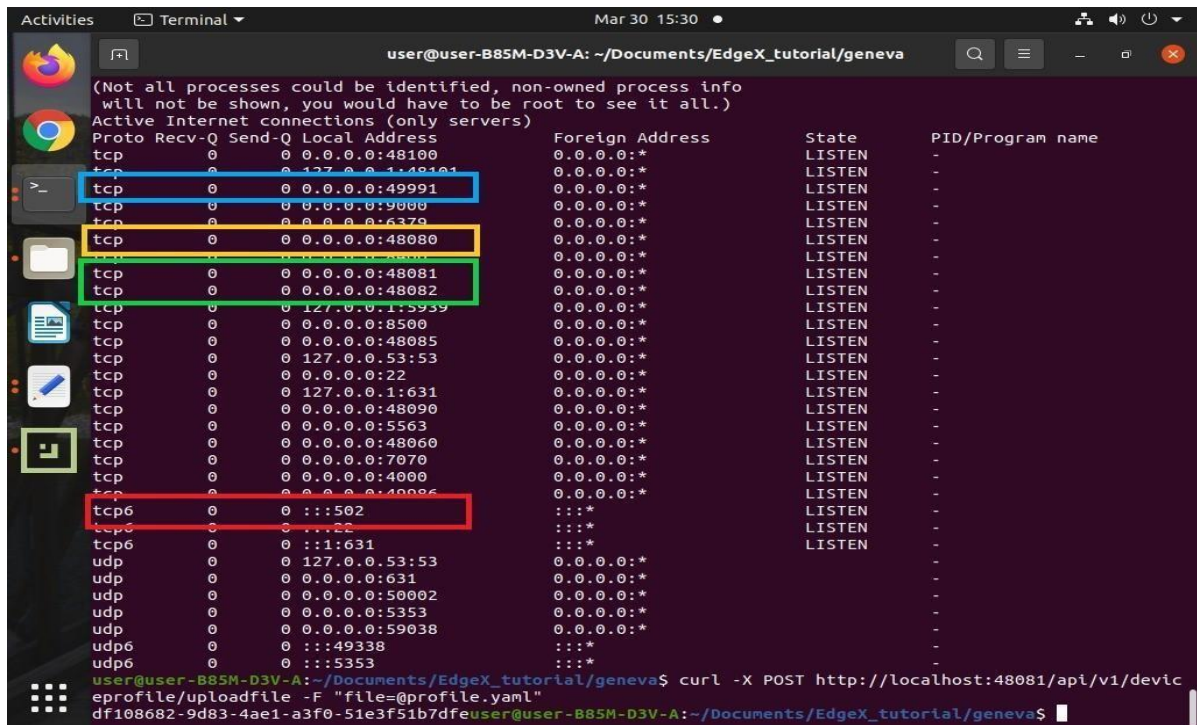
Similarly, the Humidity value received is 0x021a, which is 538 in decimal. SHT20 sends the values in multiples of 10, so Humidity is 53.8 %RH.

4.5 PROCEDURE

4.5.1 MODBUS IMPLEMENTATION

Verify if edgex microservices are up and running.

Use the command “ netstat -tulpn “ to view the status of the microservices.



The screenshot shows a terminal window with the command `netstat -tulpn` executed. The output lists active internet connections. Several lines are highlighted with colored boxes: a blue box around the line for port 49991 (Modbus), a yellow box around the line for port 48080 (edgex-core-data), a green box around the lines for ports 48081 and 48082 (edgex-core-metadata and edgex-core-command), and a red box around the line for port 502 (Ort). The terminal also shows a successful curl command at the bottom.

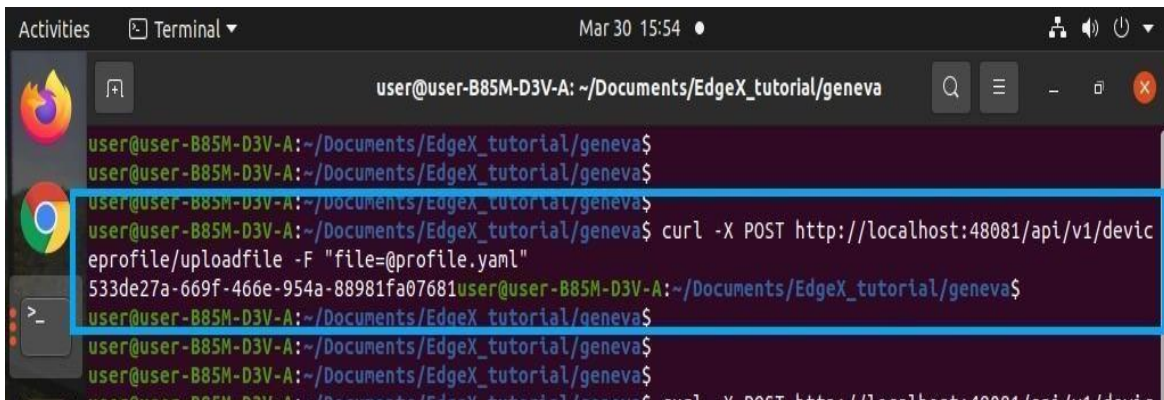
```
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:48100          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:49991          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:9000           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6379           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:48080          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:48081          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:48082          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:5939         0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:8500           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:48085          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22             0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:1:631          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:48090          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:5563           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:48060          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:7070           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:4000           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:40006          0.0.0.0:*               LISTEN      -
tcp6       0      0 :::502                 :::*                    LISTEN      -
tcp6       0      0 :::22                  :::*                    LISTEN      -
tcp6       0      0 :::1:631               :::*                    LISTEN      -
udp        0      0 127.0.0.53:53          0.0.0.0:*               LISTEN      -
udp        0      0 0.0.0.0:631           0.0.0.0:*               LISTEN      -
udp        0      0 0.0.0.0:50002          0.0.0.0:*               LISTEN      -
udp        0      0 0.0.0.0:5353           0.0.0.0:*               LISTEN      -
udp        0      0 0.0.0.0:59038          0.0.0.0:*               LISTEN      -
udp6       0      0 :::49338               :::*                    LISTEN      -
udp6       0      0 :::5353                :::*                    LISTEN      -
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$ curl -X POST http://localhost:48081/api/v1/devic
eprofile/uploadfile -F "file=@profile.yaml"
df108682-9d83-4ae1-a3f0-51e3f51b7dfeuser@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$
```

Fig.no 4.5.1.1 Edgex - Microservices

- Blue indicates that Modbus is up and running at port 49991.
- Yellow indicates that edgex-core-data is up and running.
- Green indicates edgex-core-metadata at 48081 and edgex-core-command are up and running.
- Red indicates that ort 502 is also active.

UPLOAD DEVICE PROFILE

The device profile is a basic description of the device and its dependencies. It describes the operations to be performed on the data such as read/write, starting address of the holding/input registers and details of the sensor device.

A terminal window titled 'Terminal' with a date and time of 'Mar 30 15:54'. The window shows a user at a prompt in the directory ~/Documents/EdgeX_tutorial/geneva. The user enters a curl command to POST a file to a local API endpoint. The command is: curl -X POST http://localhost:48081/api/v1/deviceprofile/uploadfile -F "file=@profile.yaml". The output of the command is a long hexadecimal string: 533de27a-669f-466e-954a-88981fa07681. A blue rectangular box highlights the command and its output.

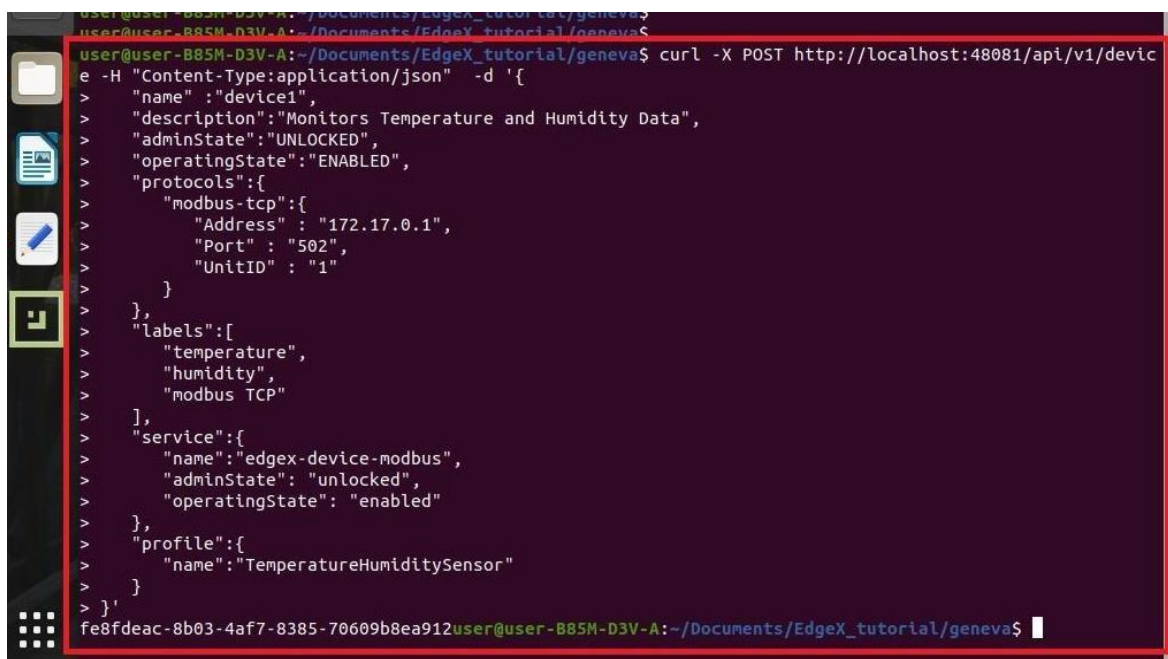
```
user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva
user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$ curl -X POST http://localhost:48081/api/v1/device
profile/uploadfile -F "file=@profile.yaml"
533de27a-669f-466e-954a-88981fa07681user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$
```

Fig.no 4.5.1.2 Uploading Device Profile

The Blue section indicates that a device profile has been successfully uploaded using the above command and a temporary id has been generated for the uploaded device profile. The id generation also confirms that the action was successful.

CREATE DEVICE USING POST COMMAND

The device can be created using the POST command as mentioned in the image.

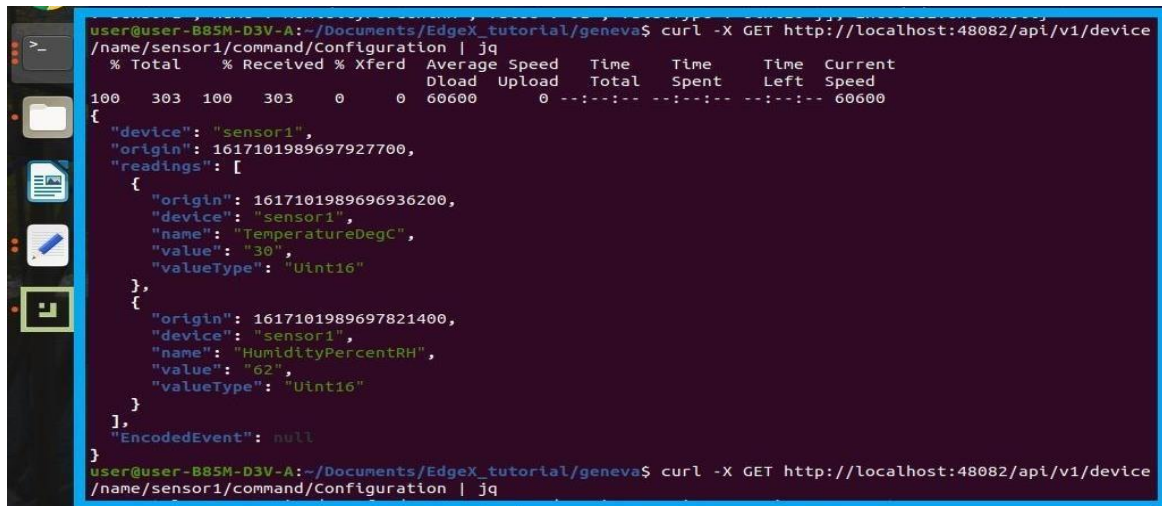
A terminal window showing a user at a prompt in the directory ~/Documents/EdgeX_tutorial/geneva. The user enters a curl command to POST a JSON object to a local API endpoint. The command is: curl -X POST http://localhost:48081/api/v1/device -H "Content-Type:application/json" -d '{...}'. The output of the command is a long hexadecimal string: fe8fdeac-8b03-4af7-8385-70609b8ea912. A red rectangular box highlights the command and its output.

```
user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$ curl -X POST http://localhost:48081/api/v1/device
-H "Content-Type:application/json" -d '{
>   "name": "device1",
>   "description": "Monitors Temperature and Humidity Data",
>   "adminState": "UNLOCKED",
>   "operatingState": "ENABLED",
>   "protocols": {
>     "modbus-tcp": {
>       "Address": "172.17.0.1",
>       "Port": "502",
>       "UnitID": "1"
>     }
>   },
>   "labels": [
>     "temperature",
>     "humidity",
>     "modbus TCP"
>   ],
>   "service": {
>     "name": "edgex-device-modbus",
>     "adminState": "unlocked",
>     "operatingState": "enabled"
>   },
>   "profile": {
>     "name": "TemperatureHumiditySensor"
>   }
> }'
fe8fdeac-8b03-4af7-8385-70609b8ea912user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$
```

Fig.no 4.5.1.3 Post Command

The red section indicates the POST command used to create the device and its configurations. The id has also been generated successfully confirming that the action was successful.

View sensor data using GET command



```
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$ curl -X GET http://localhost:48082/api/v1/device
/
name/sensor1/command/Configuration | jq
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100    303    100    303    0     0    60600      0 --:--:-- --:--:-- --:--:-- 60600
{
  "device": "sensor1",
  "origin": 1617101989697927700,
  "readings": [
    {
      "origin": 1617101989696936200,
      "device": "sensor1",
      "name": "TemperatureDegC",
      "value": "30",
      "valueType": "UInt16"
    },
    {
      "origin": 1617101989697821400,
      "device": "sensor1",
      "name": "HumidityPercentRH",
      "value": "62",
      "valueType": "UInt16"
    }
  ],
  "EncodedEvent": null
}
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$ curl -X GET http://localhost:48082/api/v1/device
/name/sensor1/command/Configuration | jq
```

Fig.no 4.5.1.4 GET Command

4.5.2 SENDING DATA TO CLOUD

- Create an IBM Cloud account
- Login to IBM cloud
- Create a watson IoT service instance
- Create a Device ID,Devicetype, and a secure Authentication token.

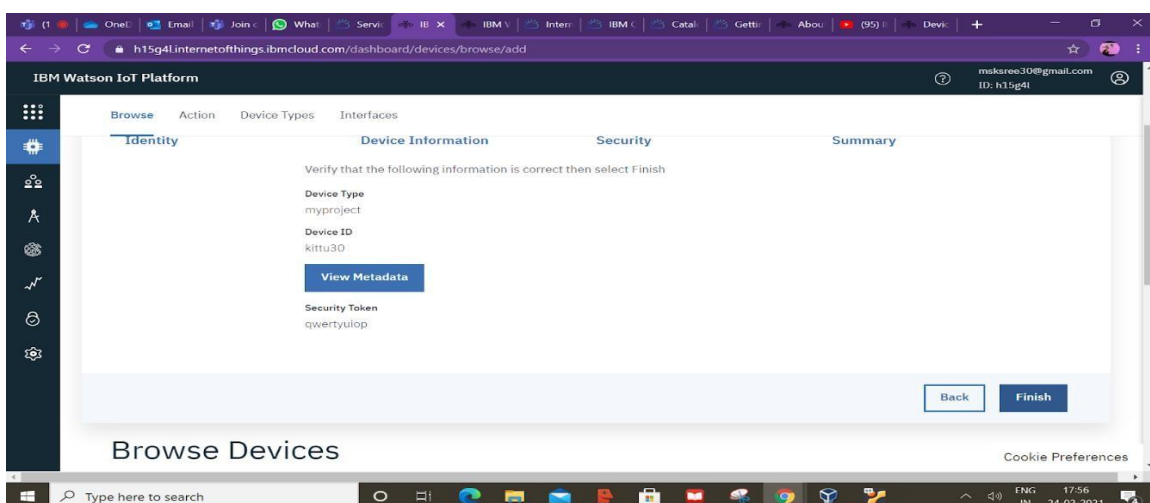
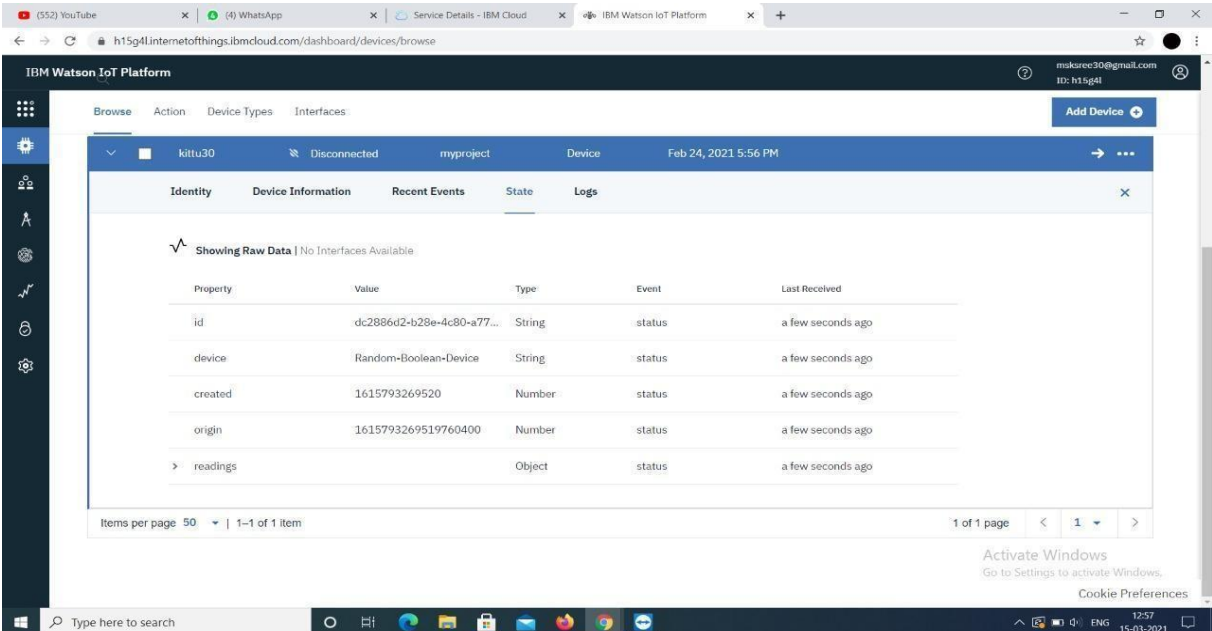


Fig.no 4.5.2.1 Sending data to cloud

The device id is created and changes should be done in application service configurable and then run all the requirements for sending data to cloud. After creating device id ,device type and security token ,run redis database,edgex go ,device virtual and application service.

Download the configuration.toml and make changes in [Writable.Pipeline.Functions.MQTTSend.Addressable] section. Enter the orgid, device type, device id and authentication token in configuration .toml file so that the data is transferred to the cloud

The status of the data received in the cloud



The screenshot shows the IBM Watson IoT Platform dashboard. The device 'kittu30' is listed as 'Disconnected'. The 'State' tab is selected, showing a table of raw data. The table has columns for Property, Value, Type, Event, and Last Received. The data shows properties like 'id', 'device', 'created', 'origin', and 'readings' with their respective values and timestamps.

Property	Value	Type	Event	Last Received
id	dc2886d2-b28e-4c80-a77...	String	status	a few seconds ago
device	Random-Boolean-Device	String	status	a few seconds ago
created	1615793269520	Number	status	a few seconds ago
origin	1615793269519760400	Number	status	a few seconds ago
readings		Object	status	a few seconds ago

Fig.no 4.5.2.2 Status of the Data Received in the Cloud

Event	Value	Format	Last Received
status	{"id":"11c650e9-b5a4-4a1e-bfd6-22b924946e..."}	json	a few seconds ago
status	{"id":"ce115b71-57ff-448e-911f-4a9cd3bc46f5..."}	json	a few seconds ago
status	{"id":"5bc71a08-358c-4f55-8599-99a561c68f2..."}	json	a few seconds ago
status	{"id":"d7897482-562e-43fa-8a17-f058f2efb29..."}	json	a few seconds ago
status	{"id":"8804100d-037c-4709-a530-dbdd06175..."}	json	a few seconds ago

Fig.no 4.5.2.3 Recent Events in cloud

The data is received every second in json format. Every second data is updated in the cloud. The communication between EdgeX and Sensors were established successfully. The rules were created and respective triggered actions were taken place. Creating an account in IBM cloud. The data was analyzed and pushed to the cloud.

4.5.3 SENDING DATA TO KUIPERS

Install the redis-database, ZeroMQ , edgex-go ,go etc all prerequisites.

```

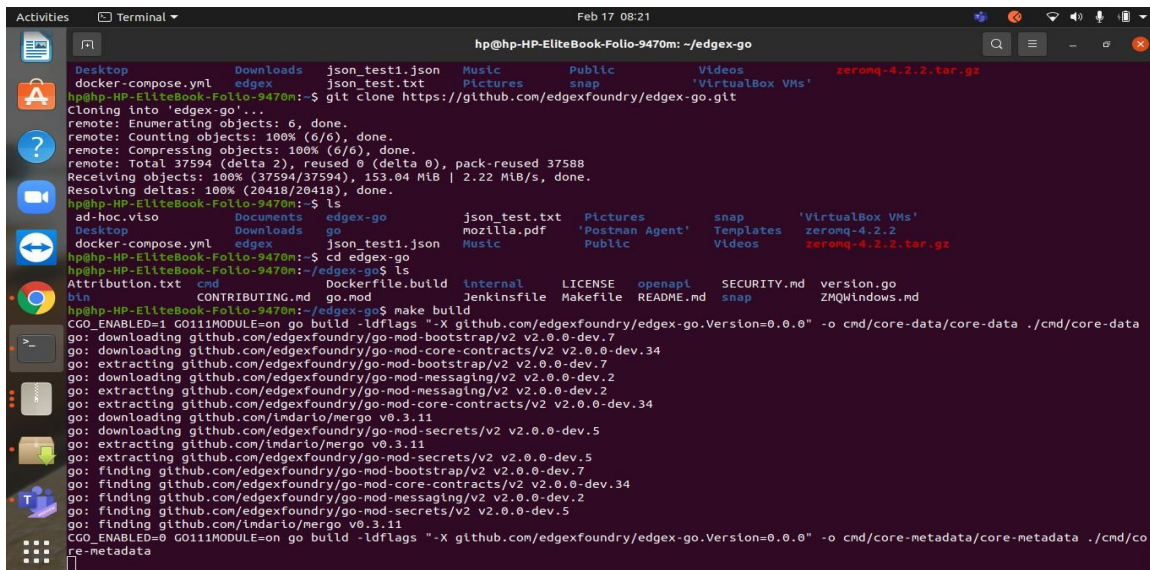
hp@hp-HP-EliteBook-Folio-9470m:~/Downloads/edgex-examples-master/application-services/custom/advanced-filter-convert-publish$ cd ..
hp@hp-HP-EliteBook-Folio-9470m:~/Downloads/edgex-examples-master/application-services/custom$ cd ..
hp@hp-HP-EliteBook-Folio-9470m:~/Downloads/edgex-examples-master$ cd ..
hp@hp-HP-EliteBook-Folio-9470m:~/Downloads$ cd ..
hp@hp-HP-EliteBook-Folio-9470m:~/Downloads$ go get github.com/rjeczalik/pkgconfig/cmd/pkg-config
hp@hp-HP-EliteBook-Folio-9470m:~/Downloads$ go get github.com/rjeczalik/pkgconfig/cmd/pkg-config
hp@hp-HP-EliteBook-Folio-9470m:~/Downloads$ ldconfig -p | grep zmq
libzmq.so.5 (libzmq, x86_64) => /usr/local/lib/libzmq.so.5
libzmq.so (libzmq, x86_64) => /usr/local/lib/libzmq.so
hp@hp-HP-EliteBook-Folio-9470m:~/Downloads$ ldconfig -p | grep zmq
libzmq.so.5 (libzmq, x86_64) => /usr/local/lib/libzmq.so.5
libzmq.so (libzmq, x86_64) => /usr/local/lib/libzmq.so
hp@hp-HP-EliteBook-Folio-9470m:~/Downloads$ ldconfig -p | grep zmq
libzmq.so.5 (libzmq, x86_64) => /usr/local/lib/libzmq.so.5
libzmq.so (libzmq, x86_64) => /usr/local/lib/libzmq.so
hp@hp-HP-EliteBook-Folio-9470m:~/Downloads$

```

Fig.no 4.5.3.1 Zero MQ

After installing the application service configurable file through git hub repository copy the rules engine configuration. toml file under res directory. Now go

to the app service configurable directory and make the file. Necessary changes should take place in app-service-configuration.

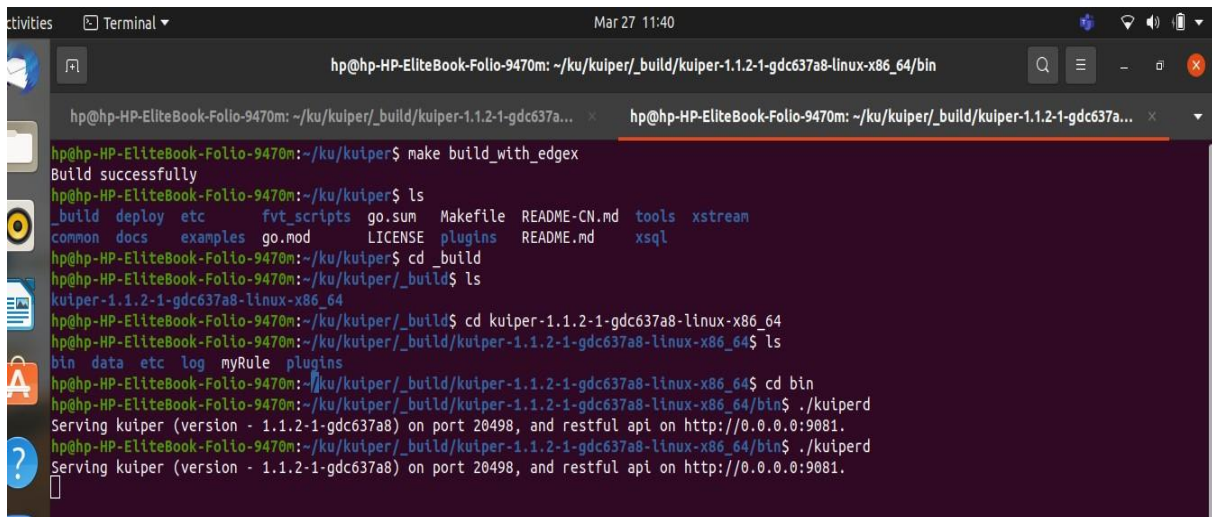


```
hp@hp-HP-EliteBook-Folio-9470m: ~/edgex-go
$ git clone https://github.com/edgexfoundry/edgex-go.git
Cloning into 'edgex-go'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 37594 (delta 2), reused 0 (delta 0), pack-reused 37588
Receiving objects: 100% (37594/37594), 153.04 MiB | 2.22 MiB/s, done.
Resolving deltas: 100% (20418/20418), done.
hp@hp-HP-EliteBook-Folio-9470m:~/edgex-go$ ls
ad-hoc.vlso  Desktop  Downloads  edgex  json_test1.json  Music  Public  Videos  zeromq-4.2.2.tar.gz
docker-compose.yml  edgex  json_test.txt  Pictures  snap  'VirtualBox VMs'
hp@hp-HP-EliteBook-Folio-9470m:~/edgex-go$ cd edgex-go
hp@hp-HP-EliteBook-Folio-9470m:~/edgex-go$ ls
Attribution.txt  cmd  CONTRIBUTING.md  Dockerfile.build  internal  LICENSE  openapi  SECURITY.md  version.go
bin  go.mod  go.sum  Jenkinsfile  Makefile  README.md  snap  ZMQWindows.md
hp@hp-HP-EliteBook-Folio-9470m:~/edgex-go$ make build
CGO_ENABLED=1 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=0.0.0" -o cmd/core-data/core-data ./cmd/core-data
go: downloading github.com/edgexfoundry/go-mod-core-contracts/v2 v2.0.0-dev.34
go: extracting github.com/edgexfoundry/go-mod-bootstrap/v2 v2.0.0-dev.7
go: downloading github.com/edgexfoundry/go-mod-messaging/v2 v2.0.0-dev.2
go: extracting github.com/edgexfoundry/go-mod-messaging/v2 v2.0.0-dev.2
go: downloading github.com/edgexfoundry/go-mod-core-contracts/v2 v2.0.0-dev.34
go: downloading github.com/indario/mergo v0.3.11
go: extracting github.com/indario/mergo v0.3.11
go: downloading github.com/edgexfoundry/go-mod-secrets/v2 v2.0.0-dev.5
go: extracting github.com/edgexfoundry/go-mod-secrets/v2 v2.0.0-dev.5
go: finding github.com/edgexfoundry/go-mod-bootstrap/v2 v2.0.0-dev.7
go: finding github.com/edgexfoundry/go-mod-core-contracts/v2 v2.0.0-dev.34
go: finding github.com/edgexfoundry/go-mod-messaging/v2 v2.0.0-dev.2
go: finding github.com/edgexfoundry/go-mod-secrets/v2 v2.0.0-dev.5
go: finding github.com/indario/mergo v0.3.11
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=0.0.0" -o cmd/core-metadata/core-metadata ./cmd/core-metadata
```

Fig.no 4.5.3.2 Building Edgex-go

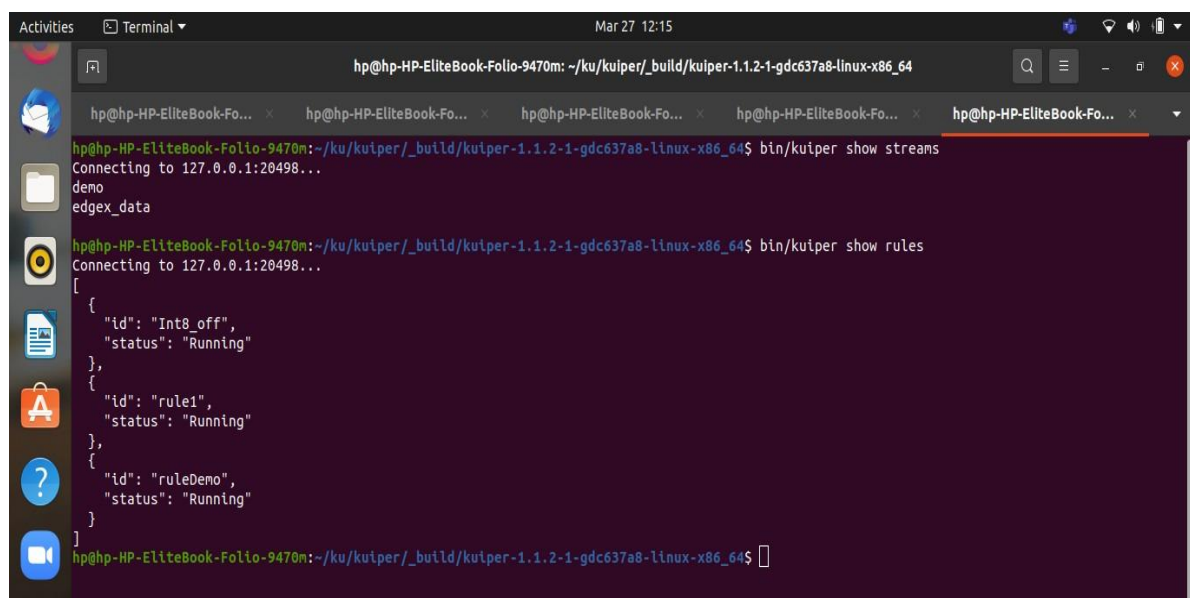
Install the Kuipers rules engine through github repository. In kuiper directory run the command make build with edgex. Then the directory _build will appear. After that change the directory _build. Then again change the directory to kuiper-1.1.2-1-gdc637a8-linux-x86_64. Under this directory the following directories can be seen. They are bin, data, etc, log and plugins.

Run the kuipers server. Make sure that the kuipers server is running continuously till our job is done. Create a stream with name edgex_data and here the data source is specified.

A terminal window on a Linux system showing the process of building and starting the Kuiper-Server. The user is in the directory ~/kuiper/_build/kuiper-1.1.2-1-gdc637a8-linux-x86_64/bin. The commands executed are: 'make build_with_edgex' (successful), 'ls' (listing files like _build, deploy, etc, fvt_scripts, go.sum, Makefile, README-CN.md, tools, xstream, common, docs, examples, go.mod, LICENSE, plugins, README.md, xsql), 'cd _build', 'ls' (listing files like bin, data, etc, log, myRule, plugins), 'cd bin', and './kuiperd'. The output shows 'Serving kuiper (version - 1.1.2-1-gdc637a8) on port 20498, and restful api on http://0.0.0.0:9081.'

```
hp@hp-HP-EliteBook-Folio-9470m: ~/kuiper/_build/kuiper-1.1.2-1-gdc637a8-linux-x86_64/bin
hp@hp-HP-EliteBook-Folio-9470m: ~/kuiper/_build/kuiper-1.1.2-1-gdc637a8-linux-x86_64/bin$ make build_with_edgex
Build successfully
hp@hp-HP-EliteBook-Folio-9470m: ~/kuiper/_build/kuiper-1.1.2-1-gdc637a8-linux-x86_64/bin$ ls
_build  deploy  etc      fvt_scripts  go.sum  Makefile  README-CN.md  tools  xstream
common  docs    examples go.mod       LICENSE plugins    README.md     xsql
hp@hp-HP-EliteBook-Folio-9470m: ~/kuiper/_build/kuiper-1.1.2-1-gdc637a8-linux-x86_64/bin$ cd _build
hp@hp-HP-EliteBook-Folio-9470m: ~/kuiper/_build/kuiper-1.1.2-1-gdc637a8-linux-x86_64/bin$ ls
kuiper-1.1.2-1-gdc637a8-linux-x86_64
hp@hp-HP-EliteBook-Folio-9470m: ~/kuiper/_build/kuiper-1.1.2-1-gdc637a8-linux-x86_64/bin$ cd bin
hp@hp-HP-EliteBook-Folio-9470m: ~/kuiper/_build/kuiper-1.1.2-1-gdc637a8-linux-x86_64/bin$ ./kuiperd
Serving kuiper (version - 1.1.2-1-gdc637a8) on port 20498, and restful api on http://0.0.0.0:9081.
hp@hp-HP-EliteBook-Folio-9470m: ~/kuiper/_build/kuiper-1.1.2-1-gdc637a8-linux-x86_64/bin$ ./kuiperd
Serving kuiper (version - 1.1.2-1-gdc637a8) on port 20498, and restful api on http://0.0.0.0:9081.
```

Fig.no 4.5.3.3 Kuiper-Server

A terminal window showing the execution of 'bin/kuiper show streams' and 'bin/kuiper show rules'. The output of 'show streams' shows 'demo' and 'edgex_data'. The output of 'show rules' shows a list of rules: 'Int8_off' (Running), 'rule1' (Running), and 'ruleDemo' (Running).

```
hp@hp-HP-EliteBook-Folio-9470m: ~/kuiper/_build/kuiper-1.1.2-1-gdc637a8-linux-x86_64/bin$ bin/kuiper show streams
Connecting to 127.0.0.1:20498...
demo
edgex_data
hp@hp-HP-EliteBook-Folio-9470m: ~/kuiper/_build/kuiper-1.1.2-1-gdc637a8-linux-x86_64/bin$ bin/kuiper show rules
Connecting to 127.0.0.1:20498...
[
  {
    "id": "Int8_off",
    "status": "Running"
  },
  {
    "id": "rule1",
    "status": "Running"
  },
  {
    "id": "ruleDemo",
    "status": "Running"
  }
]
```

Fig.no 4.5.3.4 Running Streams and Rules

For instance, myRule activates the Red LED when the temperature value goes more than 30 degrees. The REST API mentioned in the rule is Core Command of GPIO-12. Similarly, other rules are added in the Kuiper before receiving the Sensor data from Application Service. Now run the redis database,edgex-go,application service configurable and kuipers simultaneously.

```
cluster.h endtanconv.o l1stpack.h notify.c redis-check-rdb.c setproctitle.c tls.c
hp@hp-HP-EliteBook-Folio-9470m:~/Downloads/redis-6.0.10/src$ redis-server
23047:C 17 Feb 2021 14:14:52.246 # 000000000000 Redis is starting 000000000000
23047:C 17 Feb 2021 14:14:52.246 # Redis version=6.0.10, bits=64, commit=00000000, modified=0, pid=23047, just started
23047:C 17 Feb 2021 14:14:52.246 # Warning: no config file specified, using the default config. In order to specify a config file use redis-se
rver /path/to/redis.conf
23047:M 17 Feb 2021 14:14:52.247 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 6.0.10 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 23047

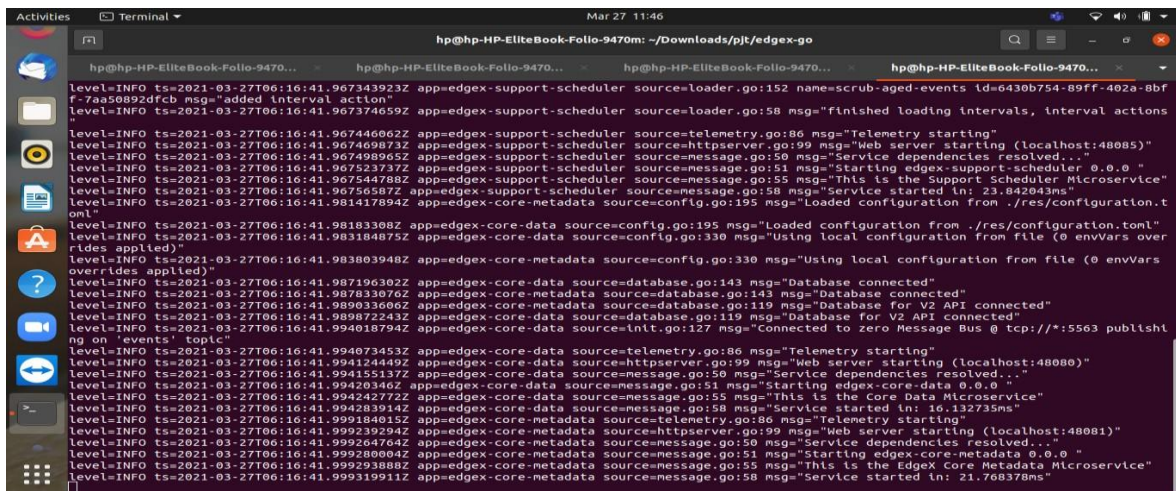
http://redis.io

23047:M 17 Feb 2021 14:14:52.248 # Server initialized
23047:M 17 Feb 2021 14:14:52.248 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this iss
ue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effe
ct.
23047:M 17 Feb 2021 14:14:52.248 * Ready to accept connections
```

Fig.no 4.5.3.5 Redis-Server

```
Activities Terminal Mar 27 11:46
hp@hp-HP-EliteBook-Folio-9470m: ~/Downloads/pjt/app-service-configurable_rules-engine
hp@hp-HP-EliteBook-Folio-9470m: ~/Downloads/pjt/app-service-configurable_rules-engine$ ./app-service-configurable_rules-engine
Starting Configurable Application Service...
level=INFO ts=2021-03-27T06:16:52.597965146Z app=app-service-configurable_rules-engine source=sdk.go:358 msg="Starting app-service-configurabl
e_rules-engine 0.0.0"
level=INFO ts=2021-03-27T06:16:52.598830219Z app=app-service-configurable_rules-engine source=config.go:193 msg="Loaded configuration from ./r
es/configuration.toml"
level=INFO ts=2021-03-27T06:16:52.600881001Z app=app-service-configurable_rules-engine source=config.go:304 msg="Using local configuration fro
m file (0 envVars overrides applied)"
level=INFO ts=2021-03-27T06:16:52.603395807Z app=app-service-configurable_rules-engine source=telemetry.go:79 msg="Starting CPU Usage Average
loop"
level=INFO ts=2021-03-27T06:16:52.608104067Z app=app-service-configurable_rules-engine source=version.go:127 version=0.0.0 msg="Skipping verst
on compatibility check for Core Developer version"
level=INFO ts=2021-03-27T06:16:52.610624265Z app=app-service-configurable_rules-engine source=server.go:284 msg="Registering standard routes..
"
level=INFO ts=2021-03-27T06:16:52.610694903Z app=app-service-configurable_rules-engine source=configupdates.go:51 msg="Waiting for App Service
configuration updates..."
level=INFO ts=2021-03-27T06:16:52.612203669Z app=app-service-configurable_rules-engine source=controller.go:63 msg="Registering standard V2 ro
utes..."
level=INFO ts=2021-03-27T06:16:52.614655094Z app=app-service-configurable_rules-engine source=sdk.go:438 msg="Service started in: 16.735179ms"
level=INFO ts=2021-03-27T06:16:52.616009568Z app=app-service-configurable_rules-engine source=main.go:38 msg="Loading Configurable Pipeline..."
level=INFO ts=2021-03-27T06:16:52.617441011Z app=app-service-configurable_rules-engine source=sdk.go:473 msg="EdgeX MessageBus trigger selecte
d"
level=INFO ts=2021-03-27T06:16:52.618842607Z app=app-service-configurable_rules-engine source=messaging.go:47 msg="Initializing Message Bus Tr
igger for 'zero'"
level=INFO ts=2021-03-27T06:16:52.620228068Z app=app-service-configurable_rules-engine source=messaging.go:61 msg="Subscribing to topic: 'even
ts' @ tcp://localhost:5563"
level=INFO ts=2021-03-27T06:16:52.622183674Z app=app-service-configurable_rules-engine source=messaging.go:71 msg="Publishing to topic: 'rules
-events' @ tcp://:5566"
level=INFO ts=2021-03-27T06:16:52.623624431Z app=app-service-configurable_rules-engine source=sdk.go:186 msg="StoreAndForward disabled. Not ru
nning retry loop."
level=INFO ts=2021-03-27T06:16:52.625040075Z app=app-service-configurable_rules-engine source=sdk.go:189 msg="AppService-rules-engine using ZM
Q Message Bus has Started"
level=INFO ts=2021-03-27T06:16:52.626535937Z app=app-service-configurable_rules-engine source=server.go:359 msg="Starting HTTP Web Server on a
ddress :48100"
```

Fig.no 4.5.3.6 Running Application Service configurable



```
level=INFO ts=2021-03-27T06:16:41.967343923Z app=edgex-support-scheduler source=loader.go:152 name=scrub-aged-events id=6430b754-89ff-402a-8bf
f-7aa50892dfcb msg="added interval action"
level=INFO ts=2021-03-27T06:16:41.967374659Z app=edgex-support-scheduler source=loader.go:58 msg="finished loading intervals, interval actions
"
level=INFO ts=2021-03-27T06:16:41.967446662Z app=edgex-support-scheduler source=telemetry.go:86 msg="Telemetry starting"
level=INFO ts=2021-03-27T06:16:41.967469873Z app=edgex-support-scheduler source=httpserver.go:99 msg="Web server starting (localhost:48085)"
level=INFO ts=2021-03-27T06:16:41.967498965Z app=edgex-support-scheduler source=message.go:50 msg="Service dependencies resolved..."
level=INFO ts=2021-03-27T06:16:41.967523737Z app=edgex-support-scheduler source=message.go:51 msg="Starting edgex-support-scheduler 0.0.0"
level=INFO ts=2021-03-27T06:16:41.967544788Z app=edgex-support-scheduler source=message.go:55 msg="This is the Support Scheduler Microservice"
level=INFO ts=2021-03-27T06:16:41.96756587Z app=edgex-support-scheduler source=message.go:58 msg="Service started in: 23.842043ms"
level=INFO ts=2021-03-27T06:16:41.981417894Z app=edgex-core-metadata source=config.go:195 msg="Loaded configuration from ./res/configuration.t
oml"
level=INFO ts=2021-03-27T06:16:41.98183308Z app=edgex-core-metadata source=config.go:195 msg="Loaded configuration from ./res/configuration.toml"
level=INFO ts=2021-03-27T06:16:41.983184875Z app=edgex-core-metadata source=config.go:330 msg="Using local configuration from file (0 envVars over
rides applied)"
level=INFO ts=2021-03-27T06:16:41.983803948Z app=edgex-core-metadata source=config.go:330 msg="Using local configuration from file (0 envVars
overrides applied)"
level=INFO ts=2021-03-27T06:16:41.987196302Z app=edgex-core-metadata source=database.go:143 msg="Database connected"
level=INFO ts=2021-03-27T06:16:41.987833676Z app=edgex-core-metadata source=database.go:119 msg="Database for V2 API connected"
level=INFO ts=2021-03-27T06:16:41.989033606Z app=edgex-core-metadata source=database.go:119 msg="Database for V2 API connected"
level=INFO ts=2021-03-27T06:16:41.989872243Z app=edgex-core-metadata source=init.go:127 msg="Connected to zero Message Bus @ tcp://*:5563 publishi
ng on 'events' topic"
level=INFO ts=2021-03-27T06:16:41.994073453Z app=edgex-core-metadata source=telemetry.go:86 msg="Telemetry starting"
level=INFO ts=2021-03-27T06:16:41.994124449Z app=edgex-core-metadata source=httpserver.go:99 msg="Web server starting (localhost:48080)"
level=INFO ts=2021-03-27T06:16:41.994155137Z app=edgex-core-metadata source=message.go:50 msg="Service dependencies resolved..."
level=INFO ts=2021-03-27T06:16:41.99420346Z app=edgex-core-metadata source=message.go:51 msg="Starting edgex-core-metadata 0.0.0"
level=INFO ts=2021-03-27T06:16:41.994242772Z app=edgex-core-metadata source=message.go:55 msg="This is the Core Data Microservice"
level=INFO ts=2021-03-27T06:16:41.994283914Z app=edgex-core-metadata source=message.go:58 msg="Service started in: 16.132735ms"
level=INFO ts=2021-03-27T06:16:41.999184015Z app=edgex-core-metadata source=telemetry.go:86 msg="Telemetry starting"
level=INFO ts=2021-03-27T06:16:41.999239294Z app=edgex-core-metadata source=httpserver.go:99 msg="Web server starting (localhost:48081)"
level=INFO ts=2021-03-27T06:16:41.999264764Z app=edgex-core-metadata source=message.go:50 msg="Service dependencies resolved..."
level=INFO ts=2021-03-27T06:16:41.99928004Z app=edgex-core-metadata source=message.go:51 msg="Starting edgex-core-metadata 0.0.0"
level=INFO ts=2021-03-27T06:16:41.999293888Z app=edgex-core-metadata source=message.go:55 msg="This is the EdgeX Core Metadata Microservice"
level=INFO ts=2021-03-27T06:16:41.999319911Z app=edgex-core-metadata source=message.go:58 msg="Service started in: 21.768378ms"
```

Fig.no 4.5.3.7 Running edgex-go

Application service configurable send the data to Rules Engine and Rules Engine will actuate the LEDs by triggering the corresponding GPIO Core Commands, when temperature thresholds are crossed. Kuiper receives the sensor data sent by the Application Service and applies the rules which are preconfigured for actuation. GPIO device service will make the LED ON/OFF when Core Command of corresponding GPIO pin is triggered by Kuiper.

CHAPTER 5

5 RESULT

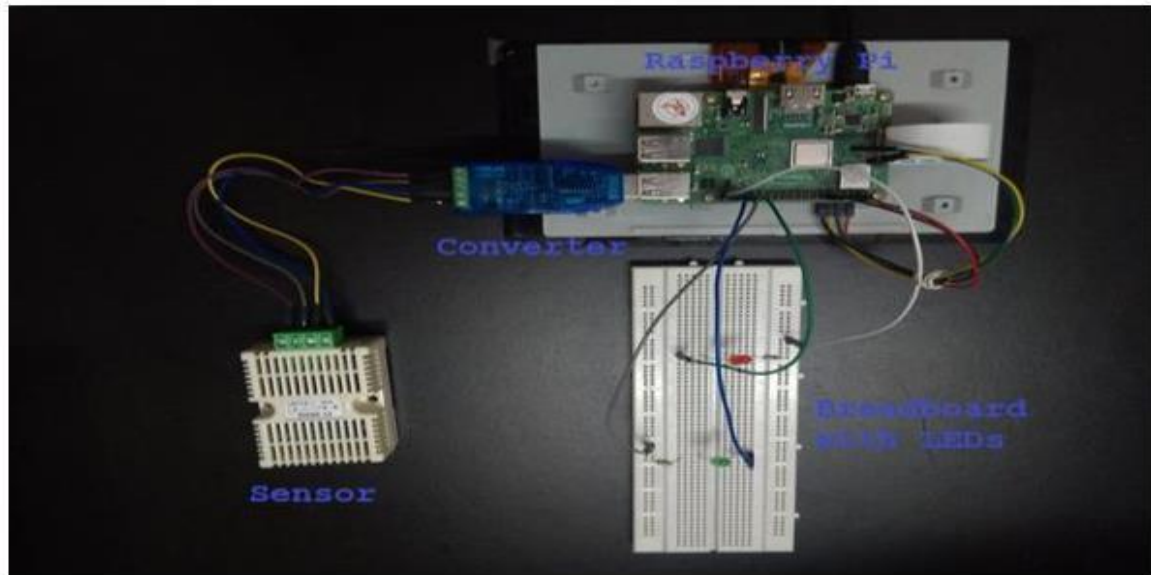


Fig.no 5.1.1 Set Up

Data received in IBM cloud

visual representation of temperature and humidity

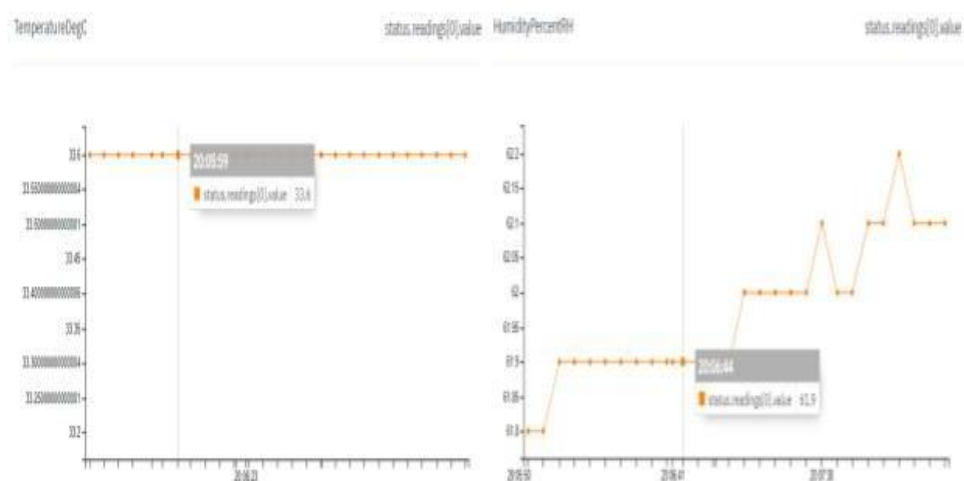


Fig.no 5.1.2 Visual Representation of Temperature and Humidity

Data is analysed in kuipers and actions are taken place in the gpio device service

When temperature is > 30



Fig.no 5.1.3 Temperature is >30 Degree

Temperature < 30

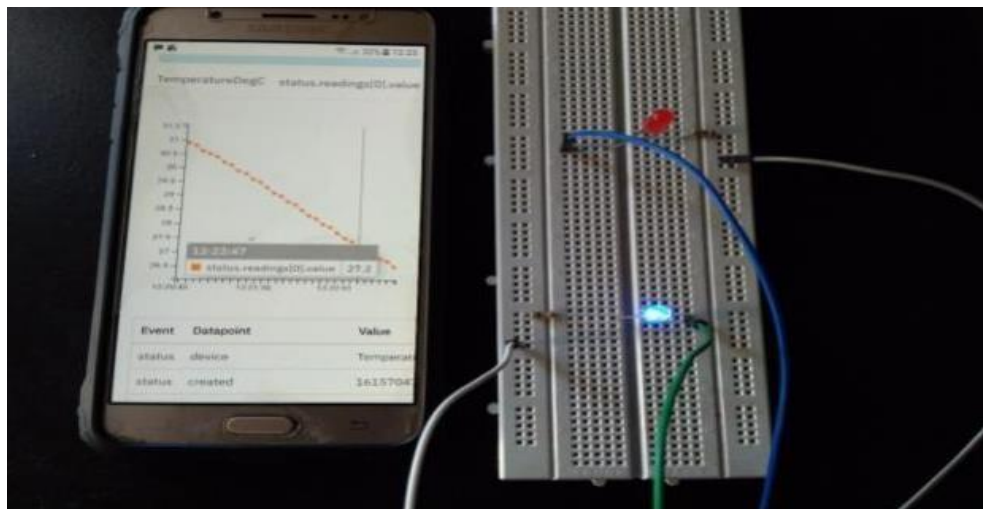


Fig.no 5.1.4 Temperature is <30 Degree

CHAPTER 6

6 CONCLUSION

In conclusion, the data is collected from a temperature sensor through raspberry pi 3 and sent to the edgex foundry. The microservices are connected through a message bus. The data is received by application service through core data. From application service the data is pushed to the cloud and parallel to the rule engine. The data is analysed in the rule engine i.e., in edge computing and the required actions are done according to the given rules. The data received in the rules engine can be seen in the log file or in any IDE like using mqtt broker can be seen in HiveMQ etc.

The final destination of this project is to provide an unhindered temperature monitoring service and that has been established successfully using edgex in edge computing. Edgex Foundry supports and facilitates insight drawing closer to the edge in order to address actuation latency concerns, bandwidth and capacity concern and operating distantly concerns. Edgex is secure and easily managed. This is a highly scalable model and thereby can support integration functionalities with multiple sensors or various other various other sensors as-well.

REFERENCES

- [1] Martin Ganchev, Bernhard Kubicek Hansjoerg Kappeler, "Rotor Temperature Monitoring System", IEEE, 11614874, 2010.
- [2] Tian Wang , Hao Luo , Weijia Jia , Anfeng Liu , Mande Xie IEEE Transactions on Industrial Informatics,"An Intelligent Trust Evaluation Scheme in Sensor -Cloud-Enabled Industrial Internet of Things",2020 .
- [3]Ananya Roy, Prodipto Das, Rajib Das, "Temperature and Humidity Monitoring System for Storage Rooms of Industries", IEEE, 17595765, 2015
- [4] Saraswati Saha, Anupam Majumdar, "Data centre Temperature Monitoring with ESP8266 based Wireless Sensor Network and Cloud-based Dashboard with the Real-time Alert System", IEEE, 17279784, 2017.
- [5] <https://docs.edgexfoundry.org/1.3/>
- [6] Vongsagon Boonsawat, Jurarat Ekchamanonta, Kulwadee Bumrunghet, and Somsak Kittipiyakul, "XBee Wireless Sensor Networks for Temperature Monitoring", 2015.
- [7] <https://github.com/edgexfoundry/app-service-configurable.git>
- [8] Verena Raab, Brigitte Petersen, Judith Kreyenschmidt, "Temperature monitoring in meat supply chains", British Food Journal, 0007-070X, 2011.
- [9] <https://github.com/emqx/kuiper/>
- [10] <https://github.com/edgexfoundry/edgex-examples/tree/master/deployment/raspberry-pi-4>
- [11] <https://redis.io/topics/introduction>
- [12] <https://docs.edgexfoundry.org/1.3/microservices/device/virtual/Ch-VirtualDevice/>

[13] <https://github.com/edgexfoundry/edgex-examples/tree/master/application-services/configurable/app-service-configurable-ibm>

[14] https://github.com/emqx/kuiper/blob/master/docs/en_US/edgex/edgex_rule_engine_command.md

ORIGINALITY REPORT

12%
SIMILARITY INDEX

12%
INTERNET SOURCES

2%
PUBLICATIONS

6%
STUDENT PAPERS

PRIMARY SOURCES

1	docs.edgexfoundry.org Internet Source	2%
2	ubuntu.com Internet Source	2%
3	www.coursehero.com Internet Source	1%
4	en.wikipedia.org Internet Source	1%
5	Submitted to Woodcroft College Student Paper	1%
6	shop.openenergymonitor.com Internet Source	1%
7	redis.io Internet Source	1%
8	Submitted to Engineers Australia Student Paper	1%
9	farhan687.github.io Internet Source	1%

10	Submitted to St Joseph Engineering College, Mangalore Student Paper	<1 %
11	www.academia.edu Internet Source	<1 %
12	Submitted to Australian College of Business and Technology Student Paper	<1 %
13	medium.com Internet Source	<1 %
14	D Y Tadeus, Yuniarto, T Yuwono. "Prototype of wireless sensor network based on open industrial protocol: modbus", IOP Conference Series: Materials Science and Engineering, 2020 Publication	<1 %
15	hdl.handle.net Internet Source	<1 %

Exclude quotes On

Exclude matches

< 14 words

Exclude bibliography On