Frankfurt University of Applied Sciences

Dept. of Computer Science and Engineering

Project Report

WS 2022

Master of Science (M.Sc.)

Edge-Computing (Framework: EdgeX)

Submitted by

Ruchit Dobariya (Matr. N - 1378799)
Fargina Mahmud (Matr. N - 1379510)
Nelli Aghajanyan (Matr. N - 1378647)
Bhargav Anghan (Matr. N - 1387230)

Under the guidance of

Prof. Dr. Christian Baun

Contents

List of Figures

Chapter 1

Introduction

What is Edgex Foundary? In simple words, EdgeX Foundry connects to devices, sensors, actuators, and other IoT objects in the real world. It is an open source, vendor-neutral, adaptable, and interoperable software platform. Since it acts as a link between modern information technology (IT) systems and actual items that are capable of sensing and acting in the real world, EdgeX can be regarded as edge middleware. In order to reduce risk, accelerate time to market, and enable scale, the EdgeX platform facilitates and encourages collaboration among the rapidly growing community of IoT solution providers in an ecosystem of interconnected components. MQTT, REST are just a few of the IoT device connectivity protocols that EdgeX provides. It permits for encryption, transformation, filtering, and formatting before delivering the data through multiple protocols, including MQTT, to an external source. Data is typically not stored in the EdgeX Gateway itself for a long time. EdgeX is made up of several services, some of which the user has the option to enable or disable. For instance, it is possible to build rules that, when the conditions are met, automatically perform a specific behaviour [7]. Figure 1.2 shows the general overview of the edgex middleware.

Figure 1.1: EdgeX gateway between the "things" and the IT-System [9]

1.1 EdgeX Foundry Service Layers

A set of open source micro services is called EdgeX Foundry. These micro services are separated into 2 underlying augmenting system services and 4 service layers.

EdgeX Foundry's 2 underlying System Services :

- Security
- System Management

EdgeX Foundry's 4 Service Layers:

- Device Services Layer
- Core Services Layer
- Supporting Services Layer
- Application Services Layer

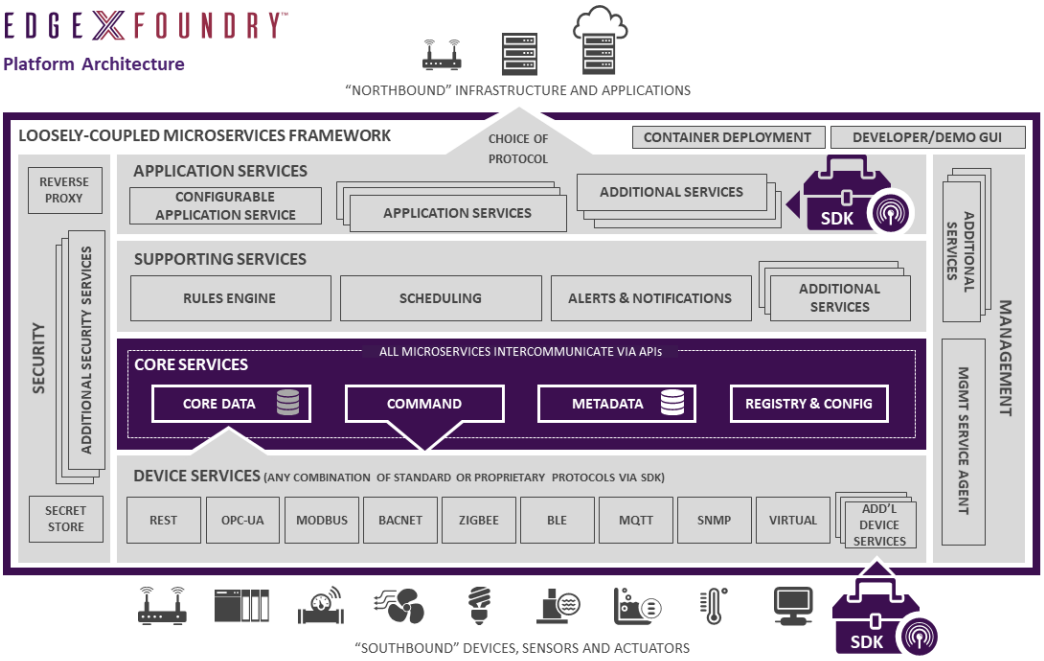

Figure 1.2: EdgeX platform architecture [8]

The Device Services Layer, actually interacts with IOT devices and sensors while also connects other service layers.

The majority of an EdgeX instance's natural understanding of what "things" are connected, what data is passing through them, and how EdgeX is set up is found in the core services.
Core layer has the following micro services:

- Core data
- Command
- Metadata
- Registry and Configuration

The supporting servies are responsible for scrubbing in terms of Edgex means for data clean up, logging and scheduling. Some of the services are:

- Rules Engine

- Scheduling
- Logging
- Alerts and Notifications

The last layer is Application Services Layer, which is responsible for sending this data after scrubbing to endpoint or cloud. Ex. Amazon Web Services (Amazon IoT Hub), Google Cloud (Google IoT Core), Azure IoT Hub, IBM Watson IoT.

Chapter 2

Installation

2.1 System Requirments

For Edgex Foundary(Used in this project):

- Ubuntu 22.04
- Internet Connection
- Minimum 1GB Memory
- 64bit CPU
- Raspberry pi 3 Model B (For Sensor Data Generation)

2.2 Installation of Raspberry Pi OS

In this section, we discussed about how to install Raspberry Pi OS on an SD card with laptop or PC. Below are the steps in order to successfully install Raspberry Pi OS on an SD card [11].

1. Download the Raspbian OS ISO from **`https://www.raspberrypi.com/software/`** official website
    - Use Raspbian Stretch with desktop and recommended software
2. Install a Raspberry Pi Imager that can create an image of your Raspberry Pi OS on your SD card from **`https://www.raspberrypi.com/software/`**
3. Select operating system to be written on your SD card by clicking on the option “CHOOSE OS”.
4. Click on "Settings" and add wifi "USERNAME" and "PASSWORD"
5. choose "STORAGE" option, select SD Card
6. Click on "WRITE".

2.3 Installation Docker and docker-compose

Commands to install Docker and Docker Compose:

1. Update the system
    - `sudo apt update`
    - `sudo apt upgrade -y`
2. Install the Docker-CE
    - `sudo apt install apt-transport-https ca-certificates curl software-properties-common -y`
    - `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
    - `sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"`
    - `sudo apt update`
    - `sudo apt install docker-ce -y`
    - `sudo usermod -aG docker $USER`
3. Now install docker-compose
    - `sudo apt install docker-compose -y`

2.4 Installation and Starting EdgeX Foundry

Commands to install Edgex Foundary:

1. Create directory with name edgex
    - `mkdir edgex`
    - `cd edgex`
2. Download the docker-compose.yml
    - `wget https://raw.githubusercontent.com/edgexfoundry/developer-scripts /aster/releases /geneva/compose-files/docker-compose-geneva-redis.l`
    - `cp docker-compose-geneva-redis.yml docker-compose.yml`
3. Pull the containers and list downloaded
    - `docker-compose pull`
    - `docker image ls`
4. Start EdgeX Foundry
    - `docker-compose up -d`

- `docker-compose ps`

5. Stop EdgeX Foundry

- `docker-compose stop`
- `docker-compose down -v`

Chapter 3

Graphical user interfaces

3.1 Raspberry Pi OS User Interface

Connect Raspberry Pi to laptop display. Follow the below step to install User Interface [3].

1. Turn on Your Raspberry Pi
    - Insert SD Card in Raspberry Pi. Turn it on.
2. Make connection with Raspberry Pi using SSH. Run below command in command prompt.
    - `ssh pi@[Raspberry Pi's IP Address]`
    - Default USERNAME and PASSWORD.
    - USERNAME: pi
    - `PASSWORD: raspberry`
3. Install VNC Server
    - `sudo apt-get update`
    - `sudo apt install realvnc-vnc-server realvnc-vnc-viewer`
4. Enable VNC Server.
    - `sudo raspi-config`
    - Select "Interfacing Options"
    - Navigate to “P3 VNC”
    - Click on "Yes"
5. Install a VNC Viewer.
    - Download VNC Viewer from **`https://www.realvnc.com/en/connect/download/viewer/`** official website.
    - Start VNC Viewer.
6. Now, you can access Raspberry pi OS using IP address of Raspberry Pi in laptop. As shown in figure 3.1

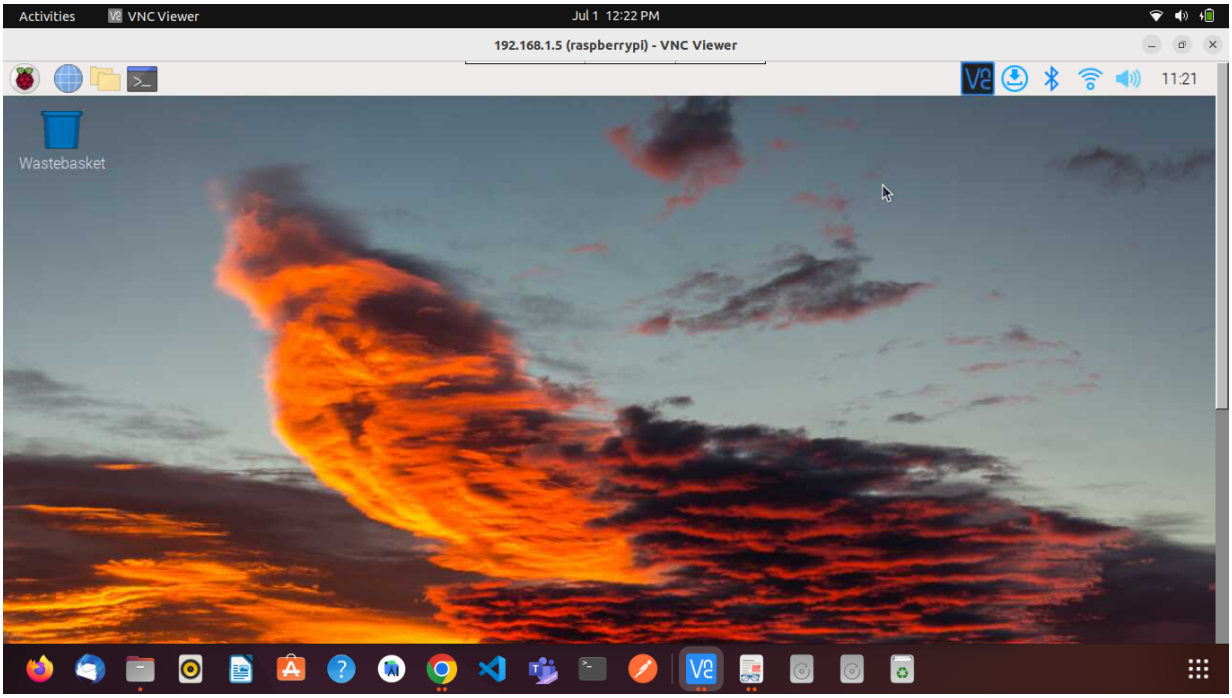

Figure 3.1: Raspberry Pi OS GUI [?]

3.2 Edgex UI user interfaces

In order to access Edgex Foundry using a UI, we need to add the UI service inside the *docker compose file*. For this purpose, we have used *Portainer UI*, which is a powerful graphical UI, where the containers, images, volumes and networks are visualised. To avoid working with terminal and all the commands, Portainer can be used to control the Docker resources [2].

```
portainer:
image: portainer/portainer
ports:
- "0.0.0.0:9000:9000"
container_name: portainer
command: -H unix:///var/run/docker.sock
volumes:
- /var/run/docker.sock:/var/run/docker.sock:z
- portainer_data:/data
```

While keeping the identation we need to add the above mentioned image with all its configuration in docker compose file —> below the *services* section. The Portainer listens on port 9000, upon first usage we need to set a password for admin user when we visit the page. Figure 3.2 shows how the login page looks like. We use the following credentials
username: admin
password: raspberry

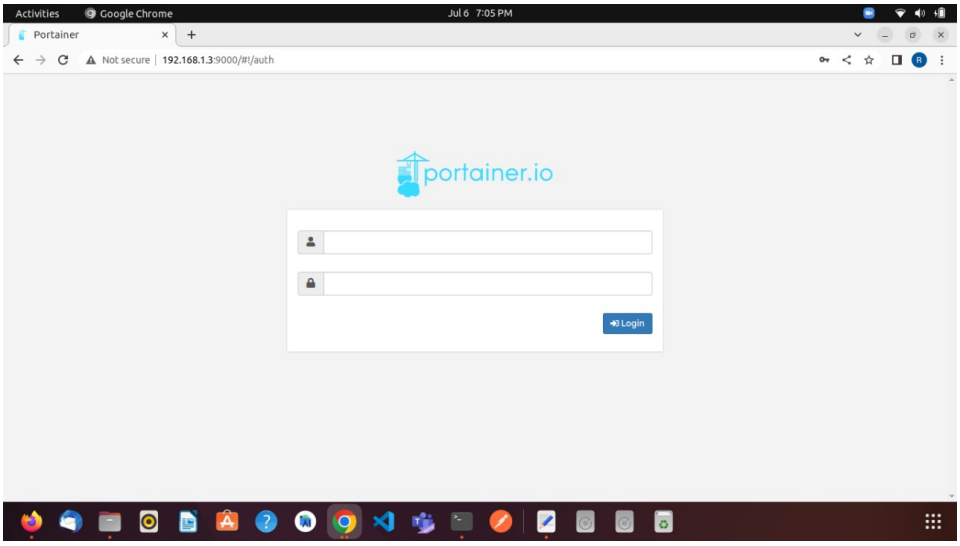

Figure 3.2: Portainer Login

In the figure 3.3 we can see the screenshot of Portainer Web UI. The details as the number of networks, container and images can be easily tracked on this UI.

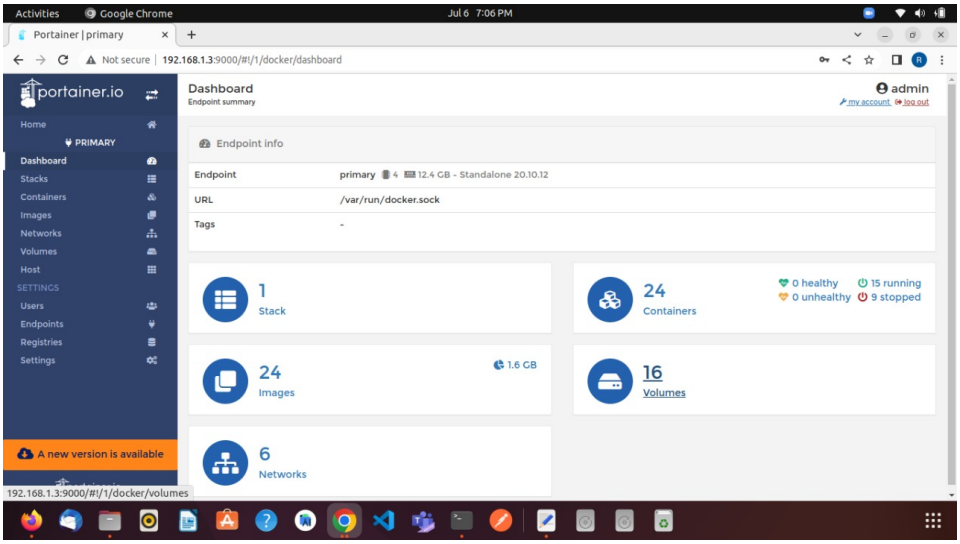

Figure 3.3: Portainer UI

Chapter 4

Device creation

4.1 Device Profiles

In this section we will describe how to send sensor data to Edgex Foundry. First of all we have to:

- **create a *device* on Edgex**

  In order to do that we need to set the value, type, format and some other configuration. To communicate with EdgeX Foundry, we are using the Postman application.
  Step 1: Open Postman application
  Step 2: Set *method* : POST ;
  URI : `http:\<edgexip>:48080/api/v1/valuedescriptor` ;
  Payload Setting : raw - JSON format ;
  Payload Data :

  ```
  {
  "name": "humidity",
  "description": "Humidity %",
  "min": "0",
  "max": "80",
  "type": "Int64",
  "uomLabel": "humidity",
  "formatting": ""labels": [
  "environment",
  "humidity"
  ]
  }
  ```

  ```
  {
  "name": "temperature",
  "description": "Temperature Celsius",
  "min": "0",
  "max": "80",
  "type": "Int64",
  "uomLabel": "temperature",
  "formatting": ""labels": [
  "environment",
  "temperature"
  ]
  }
  ```

  Response status should be **200 OK.**

- **upload the *device profile***

  The device profile is a *yaml* file that contains device name and the list of data it can handle.
  Step 1: Open Postman application
  Step 2: Set *method* : POST ;

URI : `http://<edgexip>:48081/api/v1/deviceprofile/uploadfile` ;
Payload Setting : form-data : KEY - File;
Select the yaml file : sensorClusterDeviceProfile.yaml
Have *file* as the KEY

The figure 4.1 depics the *sensorClusterDeviceProfile.yaml* file, which is used to describe the device - data formats and commands.

Figure 4.1: sensorClusterDeviceProfile.yaml

- ***value descriptor* and *device profile* are ready to create the device**

  Step 1: Open Postman application
  Step 2: Set *method* : POST ;
  URI : `http://<edgexip>:48081/api/v1/device` ;
  Payload Setting : raw - JSON format;
  Payload Data :
  {
  "name": "Temp_and_Humidity_sensor_data",
  "description": "DTH11 sensor data",
  "adminState": "unlocked",
  "operatingState": "enabled",
  "protocols":
  "example":
  "host": "random",
  "port": "1000",
  "unitID": "10"

,
"labels": [
"Humidity sensor",
"Temperature sensor",
"DHT11"
],
"location": "Germany",
"service":
"name": "edgex-device-rest"
,
"profile":
"name": "SensorData"

}

Response should be **200 OK.**

After all of these steps, now the EdgeX Foundry is ready to capture *temperature* and *humidity* data.

- **send *sensor data* from client side**
  We can use Postman to do so using POST request and the URI as `http://<edgexip>:49986/api/v1/resource/Temp_and_Humidity_sensor_data/temperature`

- **fetch the data**
  To fetch the same data from EdgeX we can use the GET api and URI `http://<edgexip>:48080/api/v1/reading` from server side.

Chapter 5

Data Generation

5.1 Temprature and humidity Sensor (DHT22)

In this project we are gernrating real data time data of Temprature and humidity using DHT22 and Raspberry Pi.

DHT22: The DHT22 is a basic digital temperature and humidity sensor. It measures the humidity and temperature of the air around it using a thermistor and a capacitive humidity sensor, and it outputs a digital signal.

- Accuracy:±2
- Humidity Range:0 100
- Temperature:-40°C 80°C
- Output Type:Digital
- Voltage - Supply:3.3V 6V

Connection circuit diagram is shown in figure 5.1
In order to run python script on raspberry pi OS. We require to install some library and packages. Follow below commands to install python and other things [6].

- `sudo apt-get update`
- `sudo apt-get upgrade`
- `sudo apt-get install python3-dev python3-pip`
- `sudo python3 -m pip install –upgrade pip setuptools wheel`
- `sudo pip3 install Adafruit-DHT`

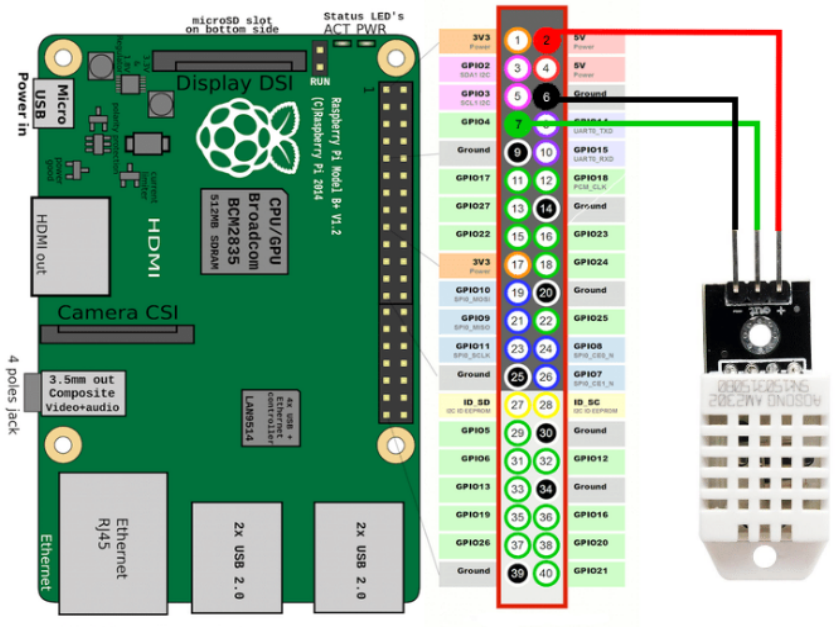

Figure 5.1: DHT22 and Raspberry Pi connection [5]

5.2 Export data to Edgex

In this section we discuss about sending data from Edgex foundary to AWS cloud. Here, Kuiper Rules Engine is used to export the data. Edgex-kuiper is running on port 48075.

There are two step to setup rules engine.

1. Create Stream
2. Create rules

1) Create Stream

Here, we used Postman to create stream and rules. Just set url and payload shown in figure 5.2.

2) Create rules

Here, we created the rules into two separate categories: "Temparature" and "Humidity". Our AWS Dynamo DB end point is used as "Url", and the request type is "post". For reference see figures 5.3 and 5.4 [4].

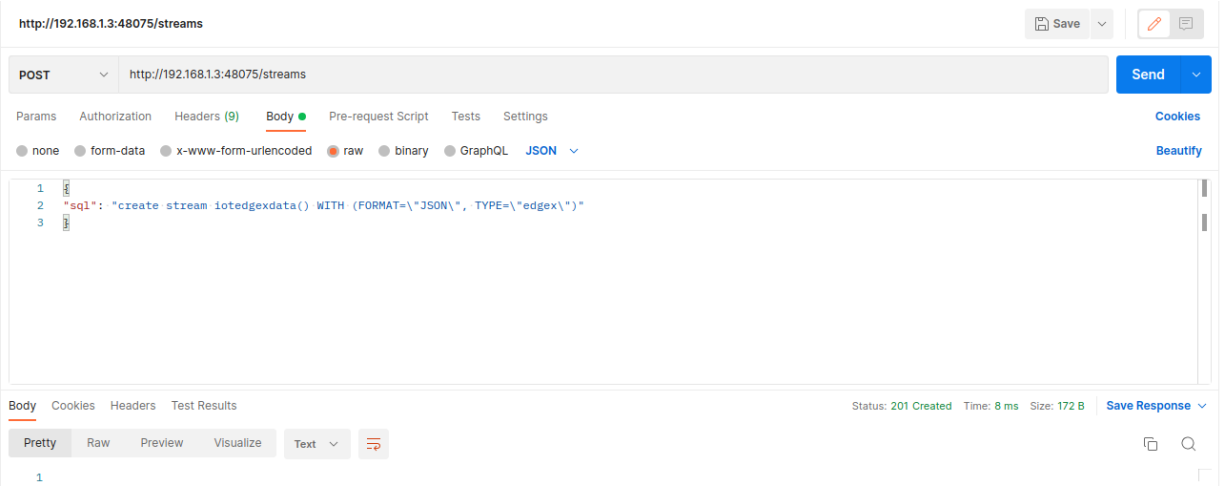

Figure 5.2: Create Stream Using Postman

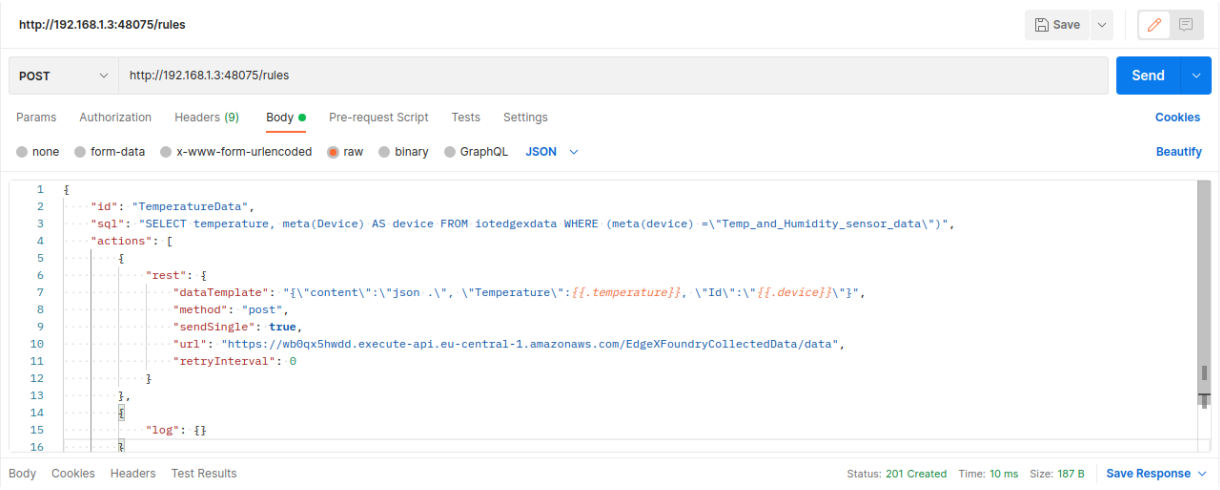

Figure 5.3: Create Temparature rule Using Postman

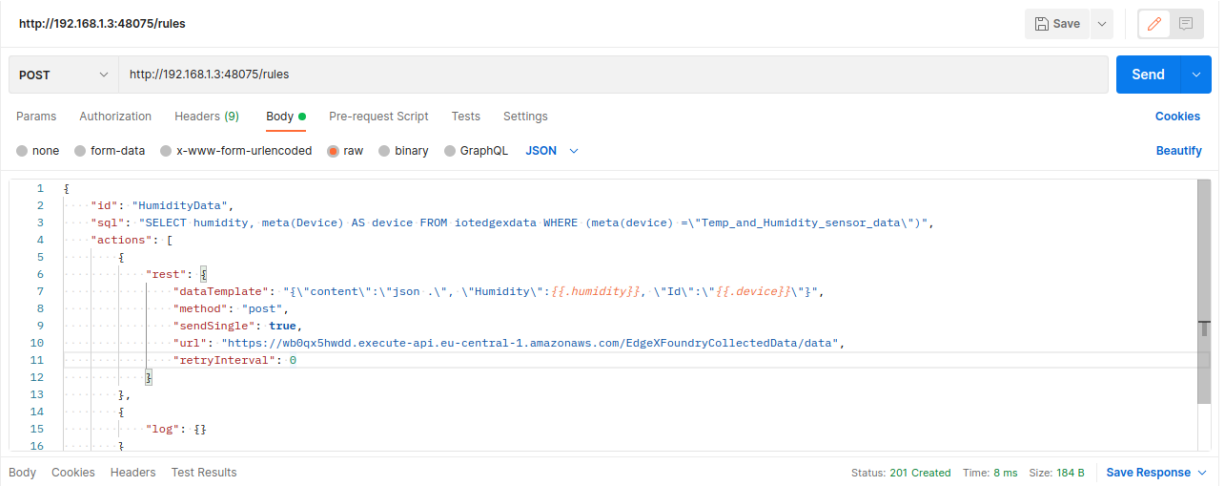

Figure 5.4: Create Humidity rule Using Postman

Chapter 6

Export Edgex data to AWS

We have used four AWS services to export EdgeX data to AWS. These are AWS DynamoDB, IAM Service, AWS Lambda and AWS API Gateway.

6.1 DynamoDB Implementation

The construction of database tables with limitless data storage and retrieval capabilities is possible due to a fully managed NoSQL database service called AWS DynamoDB. While handling the data traffic of tables over several servers, performance is automatically maintained. Additionally, it relieves consumers of the burden of scaling and managing a distributed database. As a result, management of hardware provisioning, setup, configuration, replication, software patching, cluster scalability is provided by Amazon.We created a table in DynamoDB following the steps below:-

1. Go to **`https://console.aws.amazon.com/dynamodb/`** to access the DynamoDB console.
2. selecting Create Table.
3. Do the following on the Create DynamoDB table screen:
4. Choose ”IoTDeviceData” as name of the table as shown in figure below.
5. Enter "Id" in the Partition key box for the Primary key. Choose string as the data type.
6. Select Create once the settings are exactly how you want them.

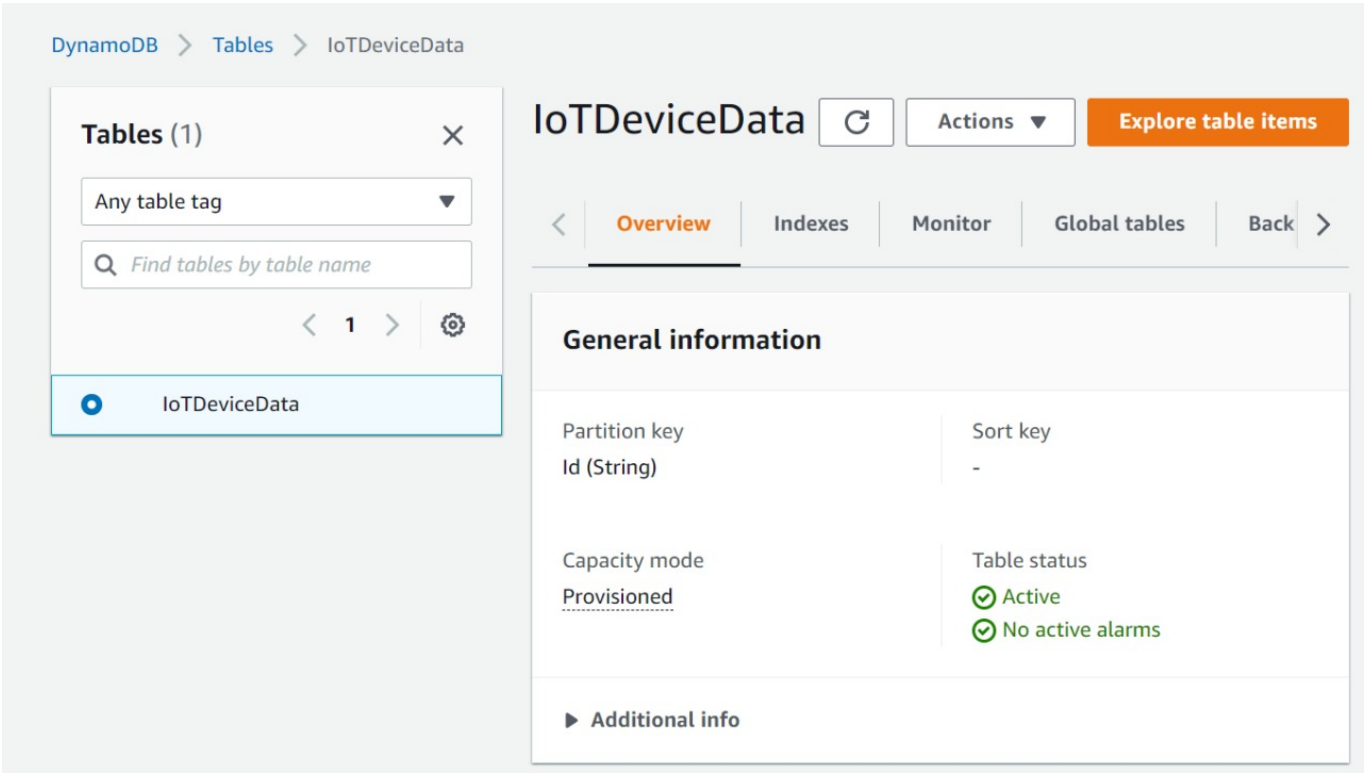

Figure 6.1: AWS DynamoDB table

6.2 AWS Lambda Implementation

One can run code without setting up or maintaining servers by using the compute service Lambda. While your code is running on a high-availability compute infrastructure, Lambda manages the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. With Lambda, we can run code for virtually any form of application or back-end service. Just we need to submit code in one of the languages that Lambda supports.

First lambda functions are required in order to access the newly constructed DynamoDB table. However, the newly generated table is first inaccessible to lambda functions. For accessing lambda function, we have to create IAM service. Utilize the management console to access the IAM service, choose "Roles" from the left-hand menu and name the new IAM service role as "IoTDeviceDataAccess" (as seen in figure 6.2), and then press "create role" button.

Secondly, "AWS service" selected and pick Lambda as the service this role is intended. Then, click on "Next" In the Permissions, and type "AWS Lambda Basic Execution Role" into the search window. Click "Next" after selecting the role specified under that name. [1]

Now, Click "Create role" after naming the role as "StoreDeviceData". Select the newly formed role and click "Add inline policy" as seen in figure 6.3 to grant this role more particular rights.Then, enter "DynamoDB" in the service search bar and "GetItem" and "PutItem" in the actions search bar in the next view.

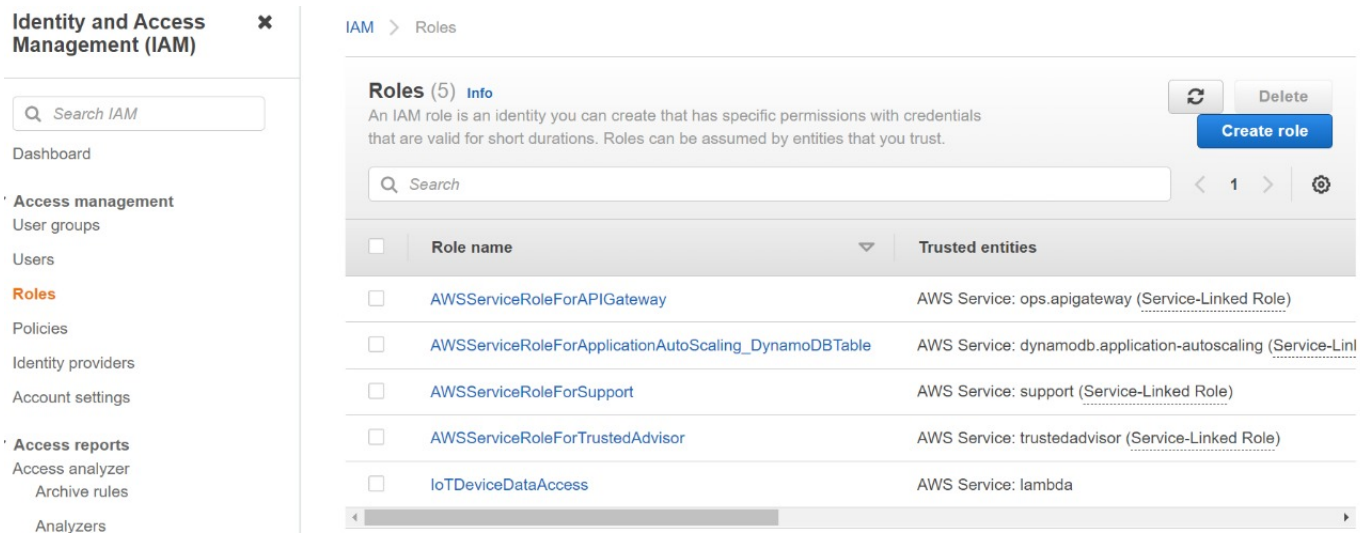

Figure 6.2: Create a new IAM role

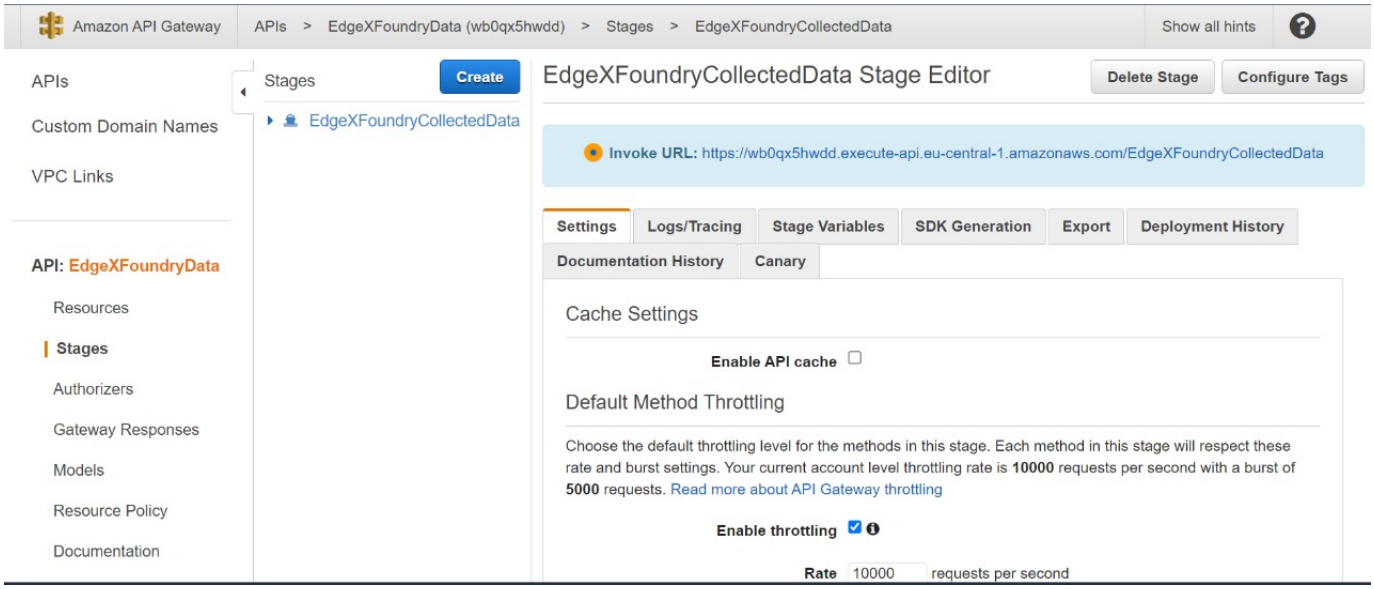

Figure 6.3: Create new Lambda service role

Lambda functions are compatible with a wide range of computer languages. Because JavaScript was utilized in this instance, Node.js has to be the runtime option. Select "Choose an existing role" from the role-dropdown menu, then pick the recently created "StoreDeviceData" position.

Put the code from appendix file into the index.js file(as seen in figure 6.4). The event object containing the entries "ID" in the json body will be received by the script when it is enabled. The "IoTDevice-Data" DynamoDB table is then updated with these two values. It also provides a response object with the http response status in it[1].

```
const AWS = require("aws-sdk");

const dynamo = new AWS.DynamoDB.DocumentClient({ region: "eu-central-1"});

exports.handler = async (event, context) => {
  let body;
  let statusCode = 200;
  const headers = {
    "Content-Type": "application/json"
  };

  try {

        const data = JSON.stringify(event);
        let requestJSON = JSON.parse(data); // parsing JSON data.

        // creating object and storing in DynamoDB table
        await dynamo
          .put({
            TableName: "IoTDeviceData",
            Item: {
              Id: requestJSON.Id,
              Temperature: requestJSON.Temperature
              //Time: requestJSON.Time

            }
          })
          .promise();

        body = "Value is stored in dynamo db";


    } catch (err) {
        statusCode = 500;
        body = err.message + " -- hello this is coming from AWS Lambda";
    } finally {

        body = JSON.stringify(body);

    }

  return {
    statusCode,
    body,
    headers
```

Figure 6.4: Index.js(Appendix)

6.3 AWS Gateway API Implementation

The Gateway API service supports RESTful-APIs. A RESTful API can be created by following these steps.

1. Go to AWS-Managment console and select the API Gateway service.
2. For selecting REST API select the button ”Create API”.
3. Choose the API-Name as ”EdgeXFoundryCollectedData” and keep the endpoint setting on regional. Select create API [10]
4. Create method and tools you want. We have created a POST HTTP method to send EdgeX data to AWS.
5. Implement some kind of access protection and policy.
6. Deploy the API.

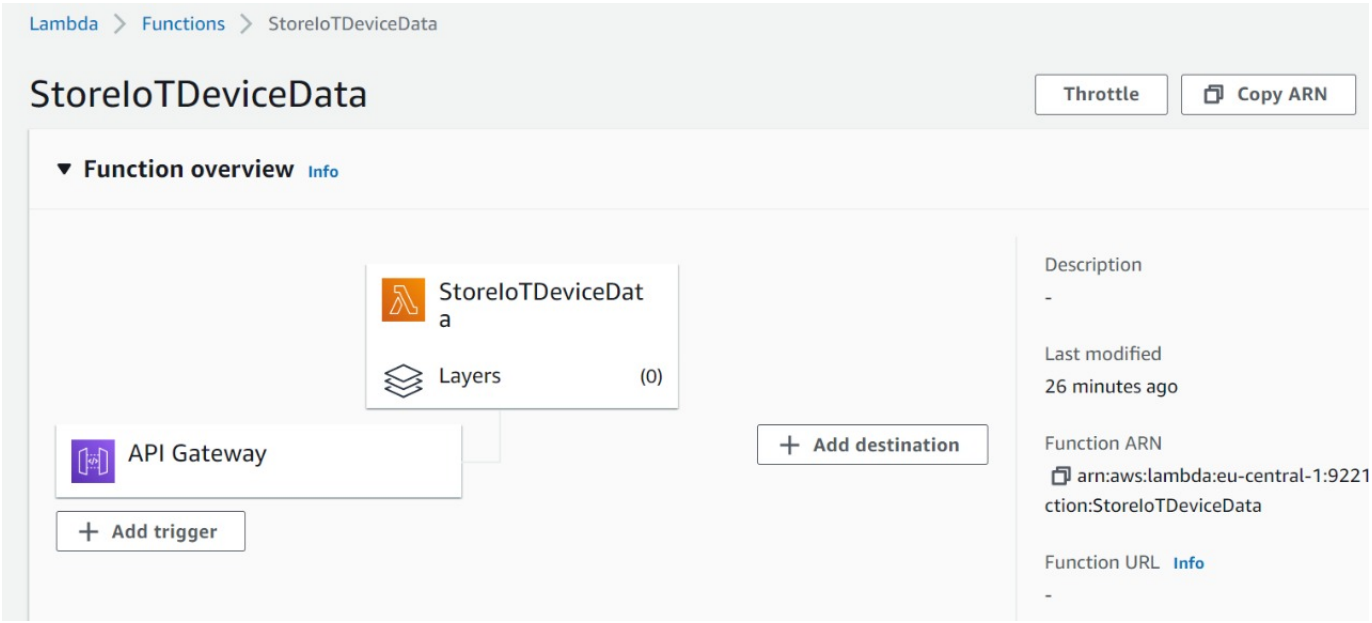

Figure 6.5: API Gatway

Conclusion

At the end we have integrated every portion of the project. We have tested while integration. During the test stage, we could see that data is being stored in AWS DynamoDB through API Gateway from EdgeX.

Bibliography

[1] Build an api gateway with crud operations using lambda and dynamodb. https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-dynamo-db.html#http-api-dynamo-db-create-function.

[2] Portainer : Container management made easy. https://www.portainer.io/.

[3] Lydia Cupery. Raspberry pi gui. https://spin.atomicobject.com/2019/06/09/raspberry-pi-laptop-display/.

[4] EKuiper. Kuiper rules engine. https://github.com/lf-edge/ekuiper/blob/master/docs/en_US/edgex/edgex_rule_engine_tutorial.md.

[5] Amna Eleyan. Sensor pin diagram. https://www.researchgate.net/figure/Wiring-diagram-for-connecting-the-DHT22-sensor-to-the-RPi_fig3_335740857.

[6] Emmet. Install python. https://pimylifeup.com/raspberry-pi-humidity-sensor-dht22/.

[7] Edgex Foundary. Edgex foundary introduction. https://docs.edgexfoundry.org/1.2/.

[8] Edgex Foundary. Edgex foundary service layers description. https://docs.edgexfoundry.org/1.2/#edgex-foundry-service-layers.

[9] Edgex Foundary. Edgex foundary use cases from official website. https://docs.edgexfoundry.org/1.2/#edgex-foundry-use-cases.

[10] Ke Han, Youyan Duan, Rui Jin, Zhicheng Ma, Hui Rong, and Xiaobo Cai. Open framework of gateway monitoring system for internet of things in edge computing. https://www.mdpi.com/1424-8220/19/22/4905/htm.

[11] Data Slayer. Raspberry pi os installtion. https://www.youtube.com/watch?v=rGygESilg8w.