Introduction to IoT

Lab 6: Setting up OM2M and communication with ESP32

Spring-2025

February 20, 2025

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

OneM2M and Eclipse-OM2M Overview

OneM2M is a global initiative designed to develop IoT standards that enable interoperable, secure, and easy-to-deploy services for the IoT ecosystem. Its primary goal is to allow any IoT application to discover and interact with any IoT device, fostering interoperability across different technologies. By establishing unified standards, oneM2M aims to **make systems more compatible, cost-efficient, and easily scalable**.

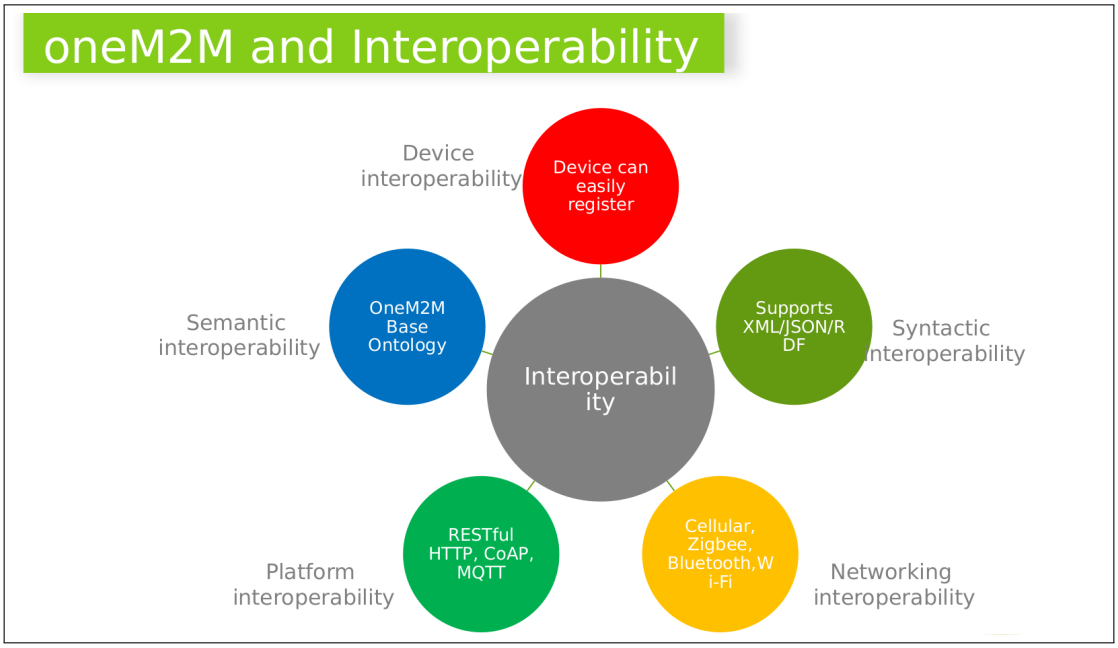

Figure 1: An Overview of How OneM2M Promotes Interoperability

Key Aspects of oneM2M Standard

The oneM2M standard utilizes common service functions, resources, and a resource tree to facilitate seamless interaction between heterogeneous devices and applications.

- **Common Service Functions**: These provide a set of standardized functionalities that can be used across various IoT applications.
- **Resources**: Key resources include Access Control Policy (ACP), Application Entity (AE), Container (CNT), Content Instance (CIN), CSE Base (CB), Group Resource (GRP), and Subscription Resource (SUB).
- **Resource Tree**: This organizes resources in a hierarchical structure, enabling easy access and management.

Eclipse-OM2M

Eclipse-OM2M is an open-source implementation of oneM2M standards developed by LAAS-CNRS. It serves as a horizontal M2M service platform and a horizontal Service Common Entity (CSE).

It supports protocol (HTTP/HTTPS) based data exchange, JSON data format, and compatibility with devices like PCs/Laptops/ESP, and networks like Wi-Fi.

Some Resources:

- Webpage for the oneM2M initiative: (link)
- Eclipse OM2M: (link)
- A practical application of OM2M by SCRC: (link).

Software Tools Required:

1. **Postman REST-API Client**
2. **OM2M Server**
3. **Java -JDK 1.8**
4. **Python**
5. **This git repository:** (link)

Hardware Tools Required:

1. **1 ESP32**
2. **1 LED**
3. **2 Laptops**
4. **Stable WiFi connection**
5. **Jumper wires and breadboard (as necessary)**

Experiment 1: Setting up Postman, OM2M server and Jupyter Notebook Resources

Installing Postman Rest Client:

- Go to the link: (link). On this page, you will see a section as given below.
- Click on the icon below to **download the desktop app**, you will be redirected to the download page, from here download the app according to your operating system.
- After downloading, install it. After installing, the postman homepage will appear.

Note: Account Creation is not necessary can be skipped

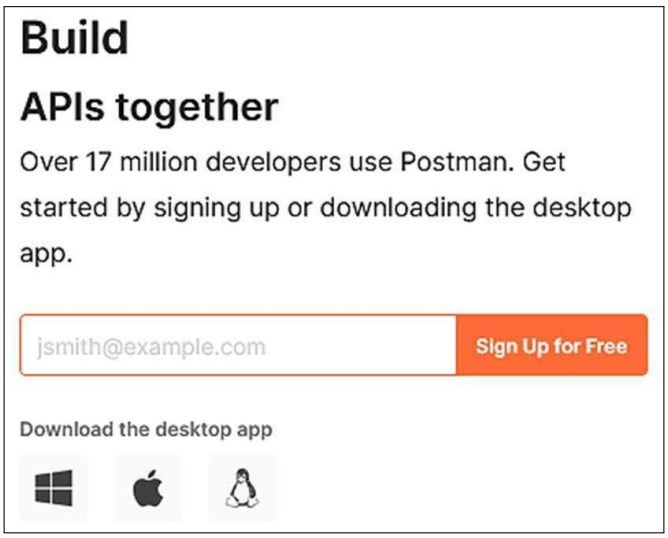

Figure 2: Postman tool

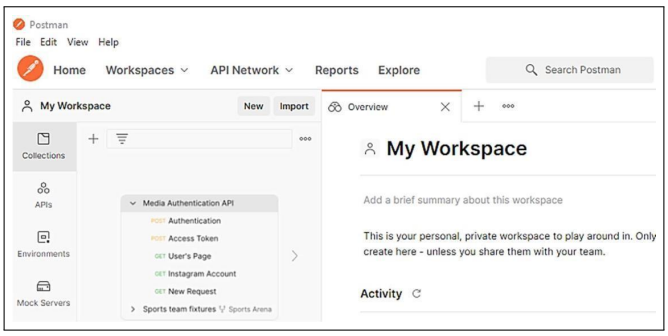

Figure 3: Postman Main Screen

OM2M Server Setup

- Go to the link: (link).
- Click on the **OM2M 1.4.1 Package** to download the latest release as shown in fig. below.
- After Successful download, extract the zip file "eclipse-om2m-v1-4-1.zip".

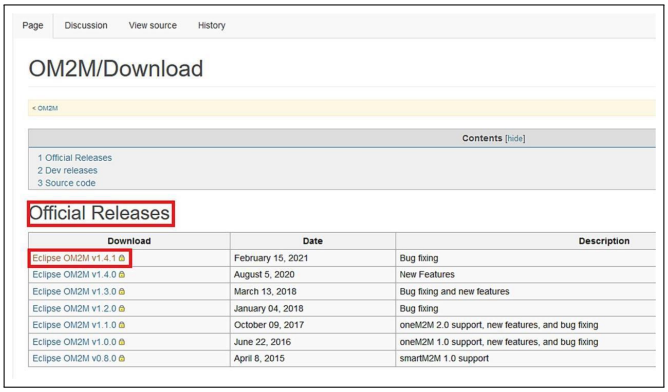

Figure 4: OM2M Download Page

- There are two folders named **"in-cse"** and **"mn-cse"**.
- Go to the **"in-cse"** folder and inside the **"configuration"** folder open **"config.ini"** file with notepad and in line 5 change the port to 5089.

Figure 5: in-cse folder

- Now go to Command prompt (in Windows search for "cmd" in start, and open) or Terminal (Open a Linux Terminal Using Ctrl + Alt + T or search Terminal in Spotlight search in Mac OS).
- Once cmd/terminal opens enter the command **"java–version"**.
- Make sure you are in the right java–version since OM2M will only be compatible with jdk 1.8.0_X versions.
- In case if you need to install jdk, please refer to the below URLs based on the OS:
  - Windows: Guide: (link), URL to download: (link)
  - Linux: (link)

– Mac: (link)

- Now in cmd/terminal enter **start.bat/start.sh** to launch the OM2M in-cse interface. (Double click generally works as well inside the contents of in-cse folder). Wait for the java command to get executed.
- Once the OM2M launches successfully, go to the OM2M login page using the link below, a page as given will appear. Link: http://127.0.0.1:5089/webpage

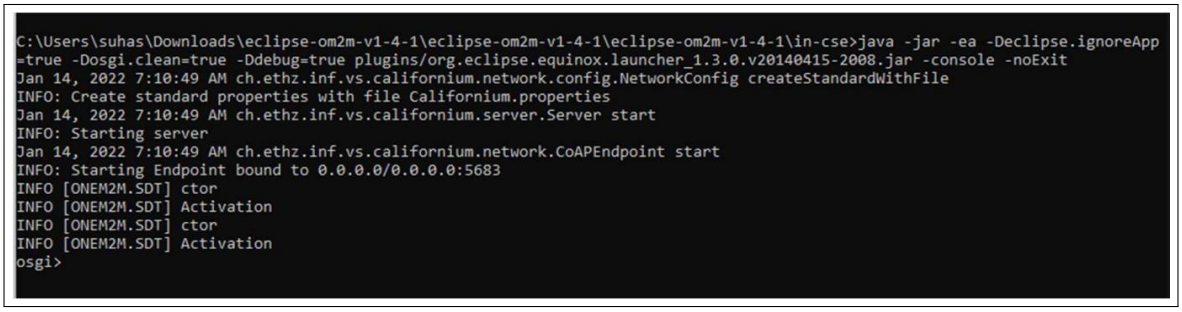

Figure 6: OM2M Successfully launched

- Enter the default username and password as **admin, admin**, and press login. We can see a default resource tree of OM2M as given below.

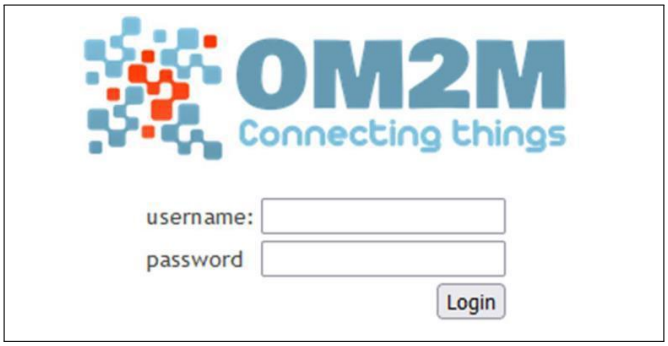

Figure 7: OM2M Login page

Figure 8: Default OM2M Resource Tree

Python and Jupyter notebook Setup

- The instructions for installing Python can be found at (link)
- Once the python has been setup install the virtual environment and activate the virtual environment (link) (for windows you can use "py -m venv iiith_env" and ".\iiith_env\Scripts\activate" to activate it)

```
C:\Users\suhas\Desktop\OM2M-REST-APIs>.\iiith_env\Scripts\activate

(iiith_env) C:\Users\suhas\Desktop\OM2M-REST-APIs>
```

Figure 9: Activated Virtual Environment

- Now install the requirements file by using the command **"pip install -r requirements.txt"** for windows users and **"pip3 install -r requirements.txt"** for.

Creating the OM2M Resource Tree using Postman

- Go to the Postman then click Workspaces then press Import then Upload Files and select the JSON files named as **'OM2M REST APIs.postman_collection.json'**
- Click on Import. We can see the request collections below the OM2M Rest APIs
- Before using the collection make sure the OM2M server is running and available at `http://127.0.0.1:5089/webpage`. The request can be sent by pressing the **"Send"** button in postman, after selecting any request.
- Now send each request one by one to construct the resource tree.

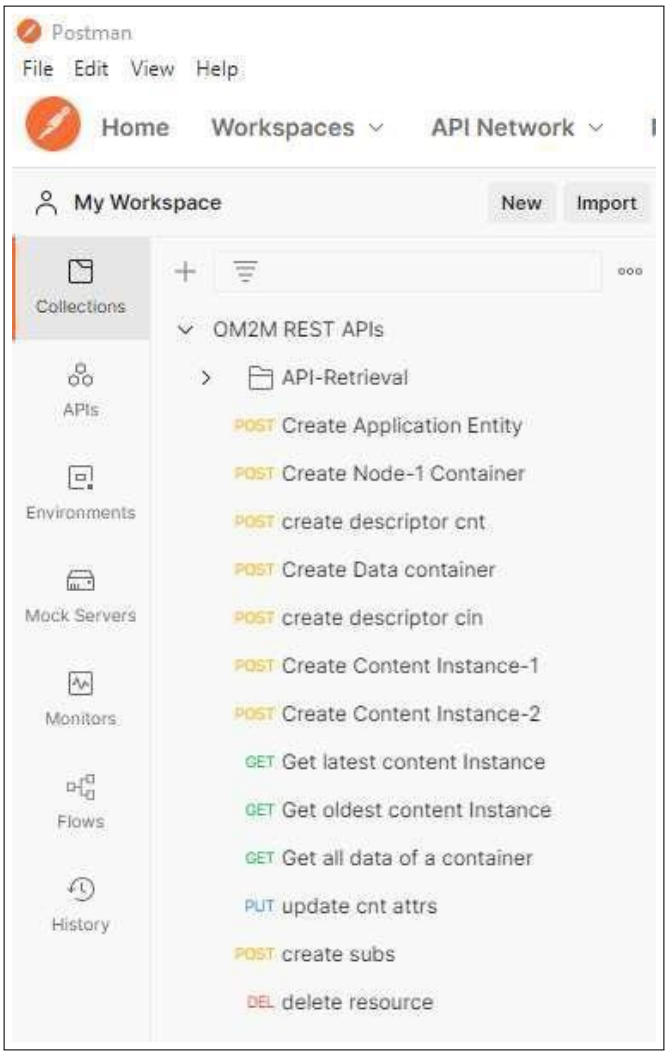

Figure 10: OM2M Postman Collection

- We can notice the changes in the OM2M server after each request by clicking on Tree node, and notice the headers, body, and responses after each request in Postman. Following are the requests to be sent in steps,

1. **Creating an Application Entity**: Select **Post Create Application Entity** Press **Send**
2. **Creating a container for device called Node-1**: Select **Post Create Node-1 container** Press **Send**
3. **Creating a Descriptor container**: Select **Post create descriptor cnt** Press **Send**
4. **Creating a Data container**: Select **Post Create Data Container** Press **Send**
5. **Creating descriptor content instance**: Select **Post create descriptor cin** Press **Send**
6. **Creating data content instance 1 & 2**: Select **Post Create Content instance 1/2** Press **Send**
7. **Retrieving the latest content instance**: Select **GET latest Content instance** Press **Send**
8. **Retrieving the oldest content instance**: Select **GET oldest Content instance** Press **Send**
9. **Retrieving all content instances**: Select **GET all data of a container** Press **Send**
10. **Updating number of attributes inside data container**: Select **PUT update cnt attrs** Send

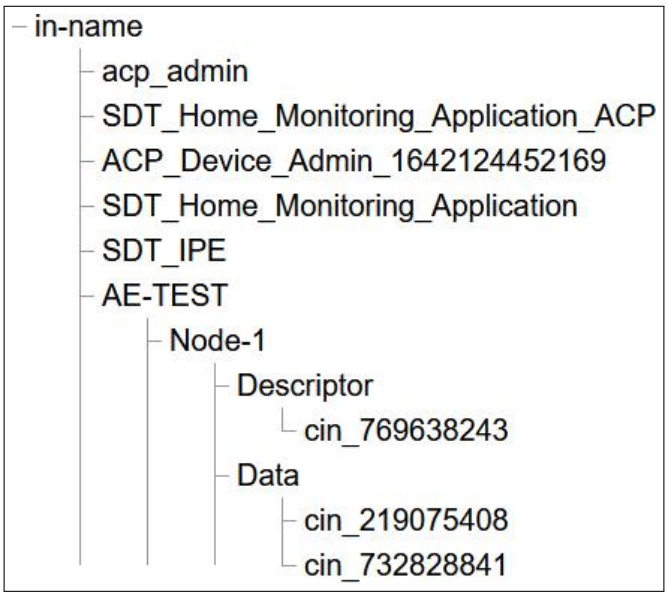

Figure 11: OM2M Resource Tree after Sending the requests

- Optional (for deleting the resource):
  1. **Deleting the resource**: Select **DEL delete resource** Press **Send**

Experiment 2: Publishing CPU and RAM Usage data to OM2M

- The objective for this experiment will be to publish your laptop's CPU usage data to the OM2M server at regular intervals, using a Jupyter Notebook.
- There are two approaches that can be followed for publishing data to OM2M.
  1. Using a Mock Device through Jupyter Notebook (Why does this work?)
  2. Using an ESP 32 Micro-Controller **(look at optional experiment 1)**
- Since we need to acquire the CPU and RAM usage of our laptop, we will be using a Jupyter Notebook as a mock-device.
- Now enter **"jupyter notebook"** in the cmd as shown below:

Figure 12: Launching Jupyter Server Home page

Running a Mock Device through Jupyter Notebooks

- Make sure the OM2M is up and running for this part of the experiment.
- After launching the Jupyter notebook you can see the home page of Jupyter server as below.

Figure 13: Jupyter Server Home page

- Now go to the folder **"Device_codes"** folder and open the notebook file **"Mock_Device_CPU_Usage.ipynb"**
- In the file click on **"Cell"** option and then press **"Run All"** to run all the cells in the notebook.

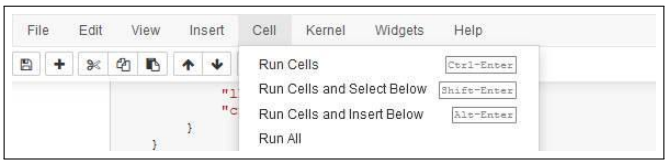

Figure 14: Run all Cells option

- The broad steps in this Notebook are as below
  1. Import required Python modules
  2. Define the default execution constants
  3. Create a OM2M Resource tree(if not exists)
  4. Get CPU and RAM usage values at regular intervals and send them to the OM2M server
  5. Run the function in an infinite loop
- Now if everything goes well you see the last cell's output as below.

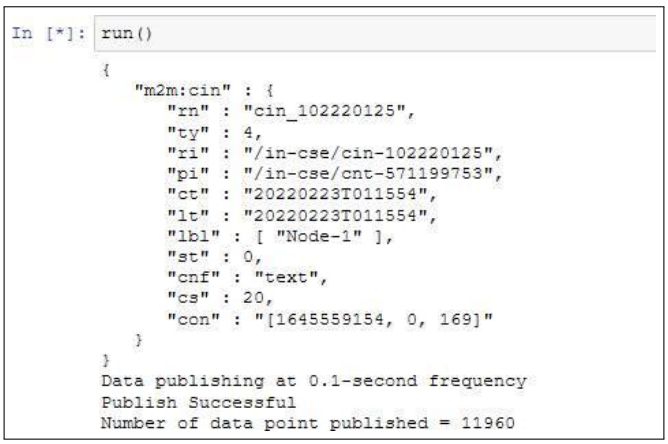

Figure 15: Mock Jupyter Device publishing data to OM2M (note that the data you send might not look like this)

- The same data can be seen in the OM2M page under **AE-TEST** application entity, in the Data container of Node-1.

Experiment 3: Implementing a remote CPU and/or RAM usage monitor:

Objective: Using and ESP32, print CPU and RAM usage on the serial monitor of your **other laptop**. Additionally, start blinking an LED if either the CPU or the RAM usage goes beyond a certain threshold, for a certain period of time.

Procedure:

- Write ESP32 code to 'GET' the latest datapoint that has been published on the OM2M server.
- Print the CPU and RAM usage on your serial monitor.
- Check whether or not the CPU/RAM usage is above a specified threshold.
- If yes, then start rapidly blinking an LED.
- Continue rapidly blinking the LED until the latest datapoint has CPU/RAM usage below the specified threshold.

Note:

- The pseudocode given is for blinking an LED using CPU usage values. **You also need to perform the experiment (i.e. rapidly blinking the LED) with the RAM usage values**. Additionally, compare which of the two (CPU or RAM) values tend to fluctuate more and can be relatively easily manipulated. **Report your observations in your lab report**.
- The polling rate of the ESP32 should be adjusted according to the rate at which the data is being published.
- **You MUST use 2 laptops** to perform this experiment.
- Opening and closing heavy applications can be a useful way to vary CPU and RAM usage values.

P.T.O.

Pseudocode:

Algorithm 1 CPU and RAM Monitoring

```
Initialize WiFi with SSID and Password
Connect to WiFi network
while WiFi is not connected do
    Wait for connection
end while
Print "Connected to WiFi"
loop
    if WiFi is connected then
        Construct URL to fetch latest data
        Send HTTP GET request with necessary headers
        if response code is valid then
            Extract JSON response
            if JSON parsing is successful then
                Extract "con" field (content) from JSON
                if data format is "[timestamp, CPU_usage, RAM_usage]" then
                    Extract and Print CPU and RAM usage values
                    while CPU usage exceeds THRESHOLD do
                        Print "CPU Threshold exceeded! Blinking LED rapidly..."
                        Turn ON LED
                        Wait 200ms
                        Turn OFF LED
                        Wait 200ms
                        if time since last datapoint exceeds 2 seconds then
                            Fetch latest data from server
                            Extract, update and Print new CPU and RAM usage values
                        end if
                    end while
                    Print "CPU usage is within safe range"
                end if
            else
                Print "JSON parsing failed"
            end if
        else
            Print "Failed to get data! HTTP Code: response_code"
        end if
        Close HTTP connection
    else
        Print "WiFi not connected!"
    end if
    Wait for 2 seconds
end loop
```

Optional Experiment 1: Publishing Data from ESP-Micro-Controller

- There are two main pre-conditions that needs to be checked before using the Arduino sketch
  1. System IP (To access OM2M from other devices)
  2. Application entity with account name

System IP:

- To check the System IP (choose the external Ip instead of 127.0.0.1), choose the ipv4
  1. open **"command prompt - cmd"** in windows and enter **"ipconfig"**
  2. open terminal and enter **"ifconfig"** in the case of Linux/Mac

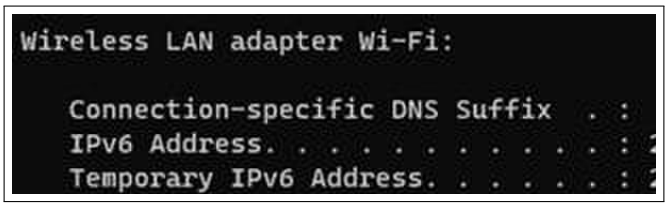

Figure 16: Example IP address-192.168.210.225

Appropriate Application Entity

- Using Postman create an application entity, node-1 container, descriptor container, data container for this experiment.
- The code can be broadly divided into the components below.
  1. Including the required header-file and create the object of the header file
  2. Defining the om2m parameters like host Ip, port number, mn, ae, data container.
  3. Connect to Wi-Fi
  4. Constructing the URL based on the om2m parameters
  5. Attaching the headers before sending the request
  6. Constructing the request with the data values of all the two sensor values
  7. Publishing the Data
- Now the following changes need to be made in the Publishing code to function
  1. SSID – Wi-Fi username
  2. Password- Wi-Fi password
  3. System IP- The system Ip needs to be used in the code
  4. AE-Name for publishing to correct AE
  5. Data content instance value for publishing appropriate values

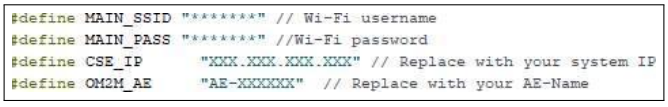

Figure 17: Parameter Changes in the Code

Figure 18: Replace with DHT Sensor functions

- Once the Serial monitor prints a 201, the data gets published, and the same data can be seen on OM2M.

Optional Experiment 2: Visualization of data on OM2M

The following steps are for implementing a Simple Visualizing Script:

1. Get all data from OM2M.
2. Clean all data for retrieving the actual content.
3. Plot the data.

Pre-requisites:

- Make sure the OM2M is up and running.
- Ensure data is continuously posting to the OM2M Server.

Steps:

1. Launch the Jupyter notebook. The home page of Jupyter server will be visible.
2. Navigate to the "Data-Visualization" folder and open the notebook file "Simple Dynamic Plots.ipynb".
3. In the file, click on "Cell" option and then press "Run All" to execute all the cells in the notebook.

Figure 19: Code Files

Figure 20: Run all Cells

Broad Steps involved in the script:

1. Import required Modules
2. Define a function for:
    - Retrieving all data from OM2M
    - Cleaning the data
    - Generating plots for data
3. Run this function at a 1-second interval.

Expected Outcome:

- If everything is set up correctly, a live graph should be visible.