DMSP Manual

Introduction

The purpose of this manual is to demonstrate the techinial and implementation details about the digital mobile sensing platform developed by the future resilient systems. We hope this manual can help the readers to implement the DMSP from scratch.

System Architechture

Ideally, the platform should include four main parts, data collection, data transmission, data anlysis, and data visualization. Since this project mainly aims to implement a primitive demo, we did not include the data analysis functionalities. The system architecture is shown in Fig.1. The figure concludes the main softwares, devices, or techniques used for implementing each part. The solid line indicates the current data path, while the dash line indicates an alternative path.

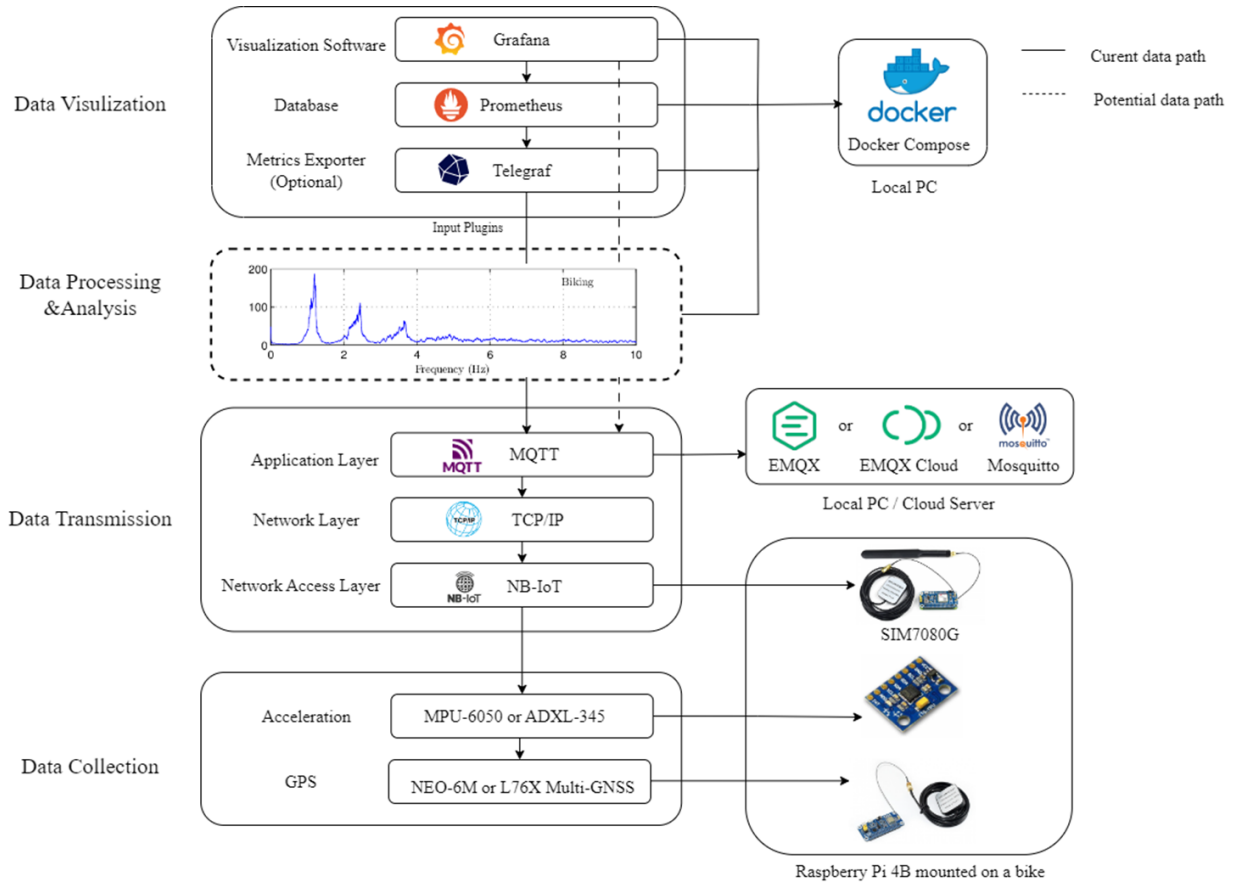


Fig.1. The architecture of the DMSP

Implementation

- Data Collection

We starts from the data collection part. We use a Raspberry Pi 4B equiped with an acceleration sensor, MPU6050 and a GPS hat, L76X to collect the viration and loaction data.

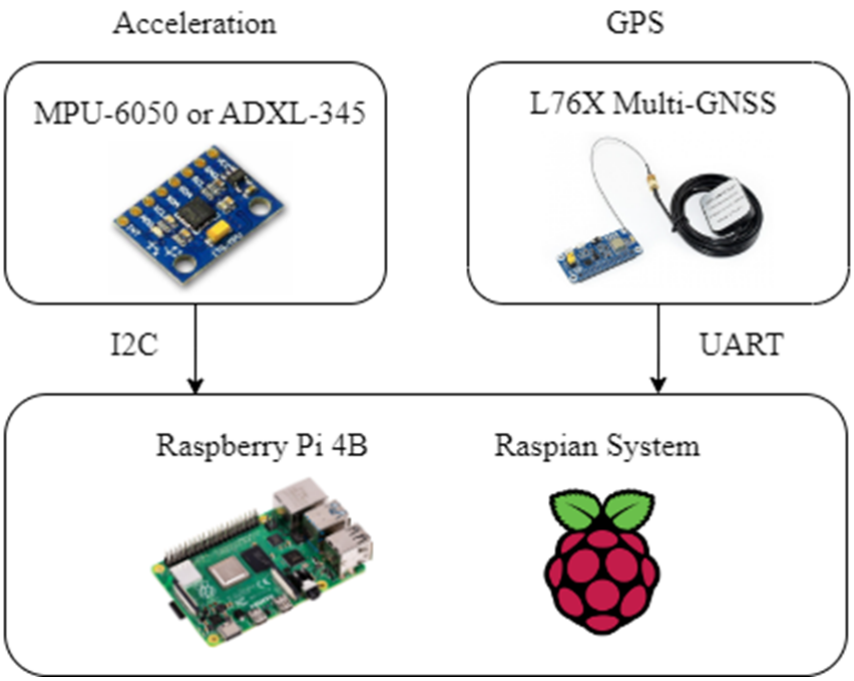


Fig. 2. Data collection module of the DMSP

Fig.3. The scripts controlling the sensors

With the help of certain libraries, we use python scripts to interact and control the devices. The codes shown in Fig.2. contains the codes for configuring and controlling the sensors to collect the data. We use the I2C interface to interact with the acceleration sensor and use the UART interface for GPS module, thus we can use the two devices simutanously without confilcts. To use these scripts, you do not need to modify any parts of the codes.

Since the acceleration sensors have been discussed in the previous work, I will not conlude contents related to them. If you have any problem with the acceleration sensors, just refer to the previous projects. Here, I will pay more attention to how to deal with the GPS hat. In Table.1, I sumarized some crucial specifications of the GPS module.

Specifications	**Value**
Capturing Time (cold start)	10 seconds
Capturing Time (hot start)	1 seconds
Refreshing Frequency	< 10 Hz

Specifications	Value
Positioning Accuracy	< 2.5 m
Receiving Signal	GPS, BD2, and QZSS
Communication Interface	UART
Baud Rate	115200
Communication Protocol	NMEA

Table. 1. The specifications of the L76X GPS module

When dealing with the GPS hat, remember that it can onlt work in the outdoor environment or next to the window. Meanwhile, **make sure the black face of the atenna is upwords** as shown in the Fig. 4.

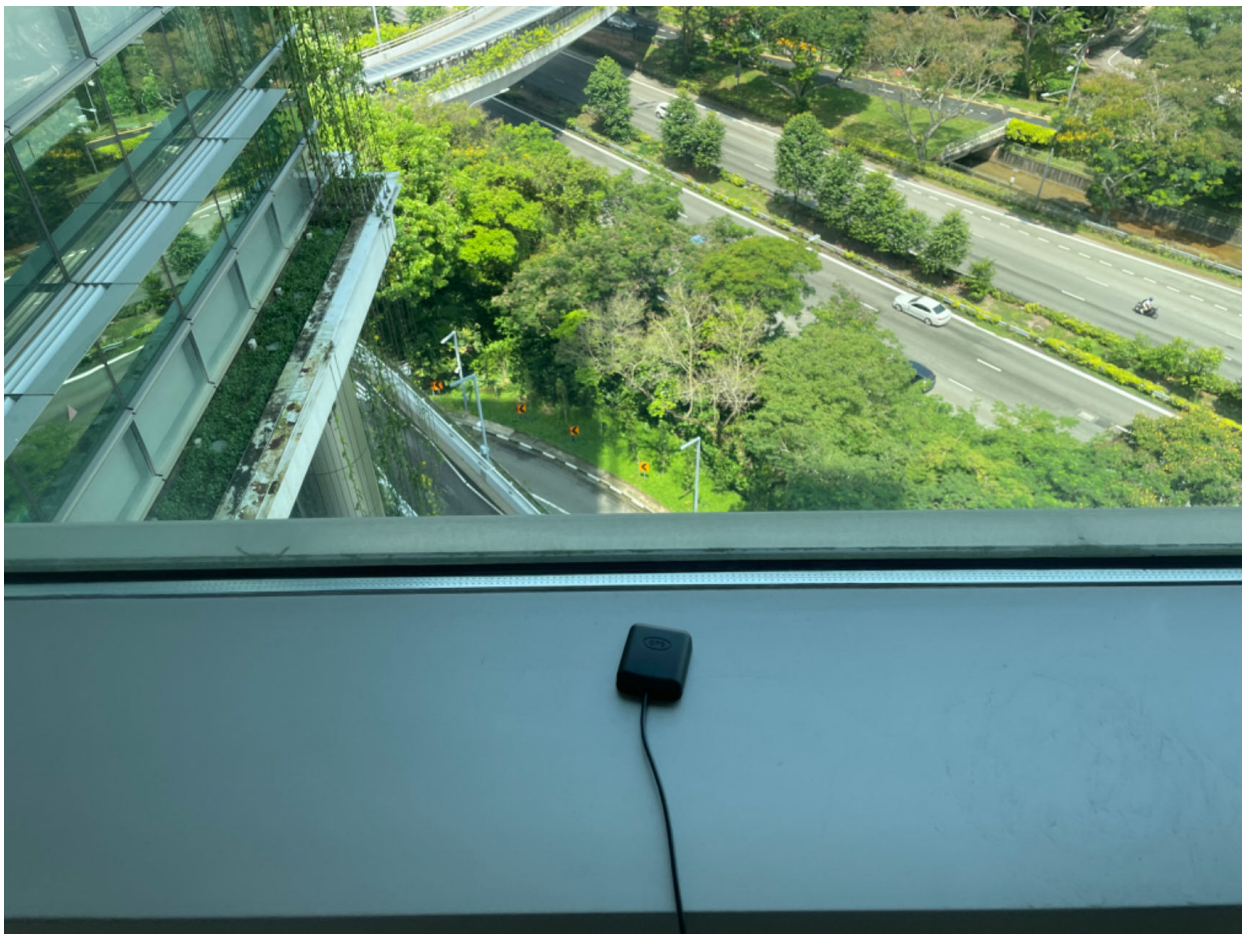


Fig.3. The placement of the L76X GPS attenna

From my experience, in the Create building, it usually takes 30 to 40 miniutes for the GPS hat to start working. But when I tested the device at home, it took much shorter time to start up. One potential reason fot the phenomenon is the the Create build is of steel structure, and the GPS moudle is senstive to metal material. **To ensure whether the GPS is working noramlly, check the 'PPS' indicator is lighten as shown in the Fig. 4**. Ingeneral, all the LEDs except the 'RXD' are on,

when the device works normally. More details or specifications of the L76X GPS hat can be found in L76X GPS HAT - Waveshare Wiki.

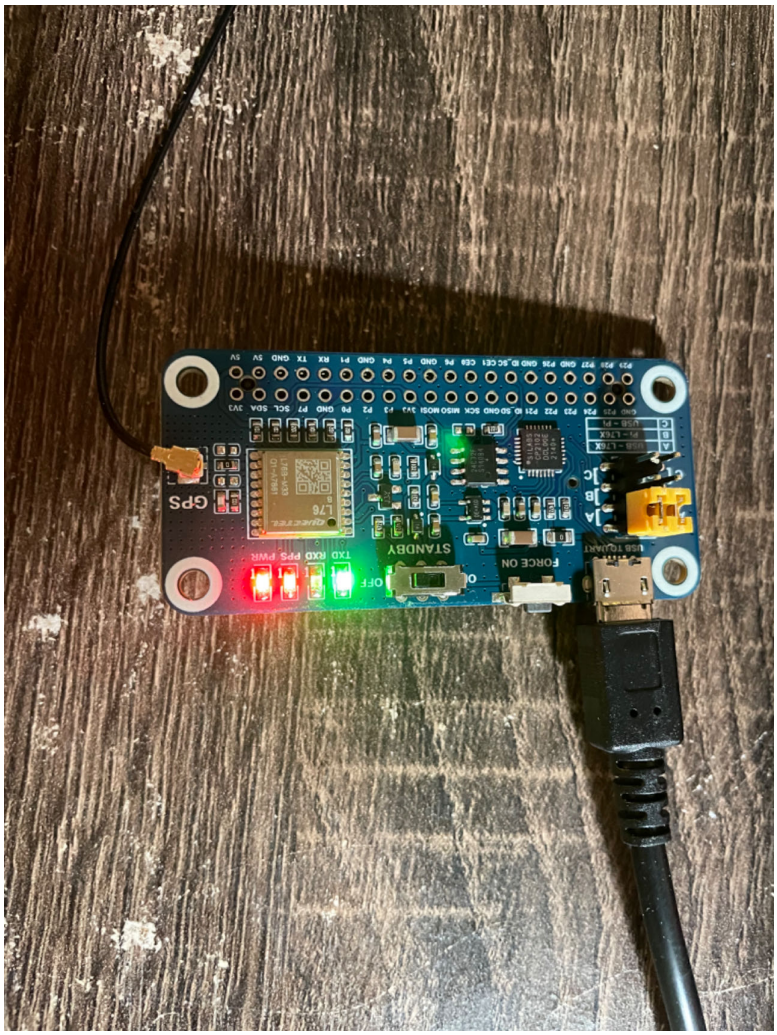


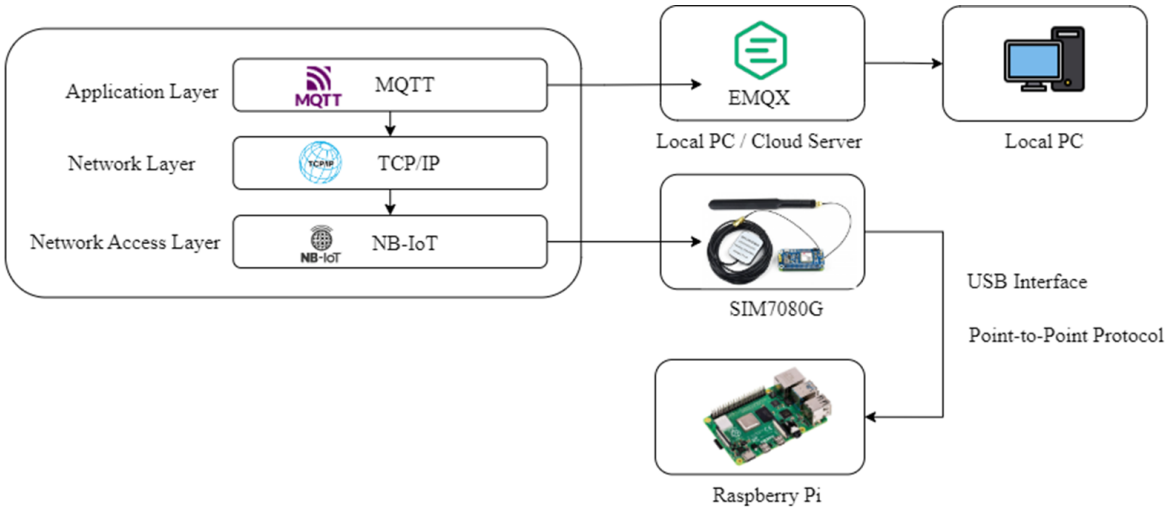
Fig.4. The LED indicators of the L76X GPS when the device works normally

To test the GPS hat solely, you can run the following codes in the 'l76x_gps.py'. The GPS data will be printed in the console then. If the GPS hat does not work normally, the data printed out is like 'lat: 0, lng: 0', otherwise the current location will be printed out.

```
if __name__ == "__main__":
    x = set_gps()
    while(1):
        gps_data = get_gps(x)
        print(gps_data)
        time.sleep(1)
```

Fig.5. The codes for testing the GPS hat

- Data transmission



The data transmission function is based on the MQTT protocol and the NB-IoT module. We use the a NB-IoT module to get access to the network. The NB-IoT network is based on the LTE as konws as 4G systems. So it is very suitable for applications in the urban area, since 4G network is available almost everywhere nowdays. Then we use the MQTT protocol to transmit the data back to the local computer.

- SIM7080 NB-IoT module

  We communicate with the device using 'mincom' to send AT commands. To get familiar with and validate the SIM7080 module, you can first go through the tutorials in the wiki SIM7080G Cat-M/NB-IoT HAT - Waveshare Wiki. You can check the working status of the device by observing its LED indicators. **If the NET indicator flashes once per second, that means the device have detected the nearby basestations and can establish connections.** If you do not observe the above situation, shortly press the 'PWRKEY' on the module, so that you can weak up the device from a standby mode. Then, you can use the following command to to connect to the LTE network and get an IP address.

  ```
  AT+CNACT=0,1
  ```

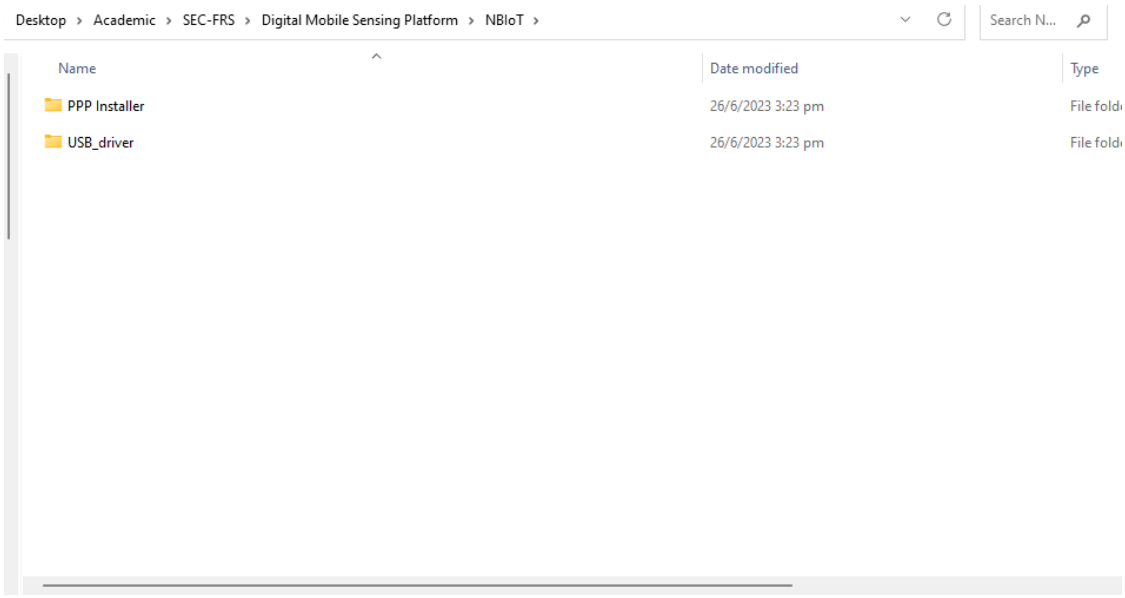
  To check the IP address, run

  ```
  AT+CNACT?
  ```

  In the 'nbiot_conf.py' file, some basic operations are provided, like power on, power off, sending certain AT commands. For detailed information and examples of the AT commands, your can refer to the SIM7080 Series AT Command Manual File:SIM7080 Series AT Command Manual V1.02.pdf - Waveshare Wiki.

- Point-to-point protocol

  We use the SIM7080 to provide network access for the Raspberry Pi, the Point-to-point protocol (PPP) is used for achieving this goal. To install the protol and make configurations, I have wrote two scripts in the follwing folders.



When using the PPP installer, you need to modify the User parameter in the 'nbiot.service' to your username.

```
[Unit]
Description=PPP Auto Connection
After=network.target

[Service]
ExecStart=/bin/sh /usr/src/reconnect.sh
WorkingDirectory=/usr/src/
StandardOutput=inherit
StandardError=inherit
Restart=always
User=dmsp

[Install]
WantedBy=multi-user.target
```

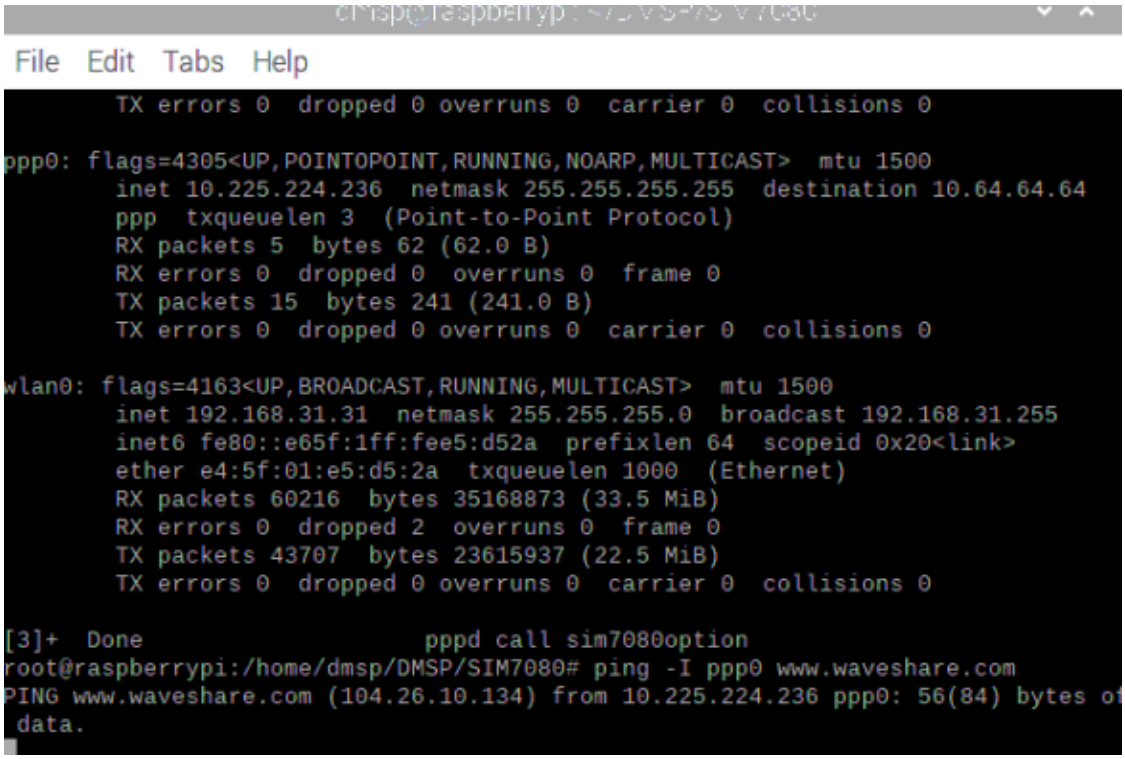
After installing the necessary driver and the PPP, run the following commands in the minicom to config the NB-IoT module.

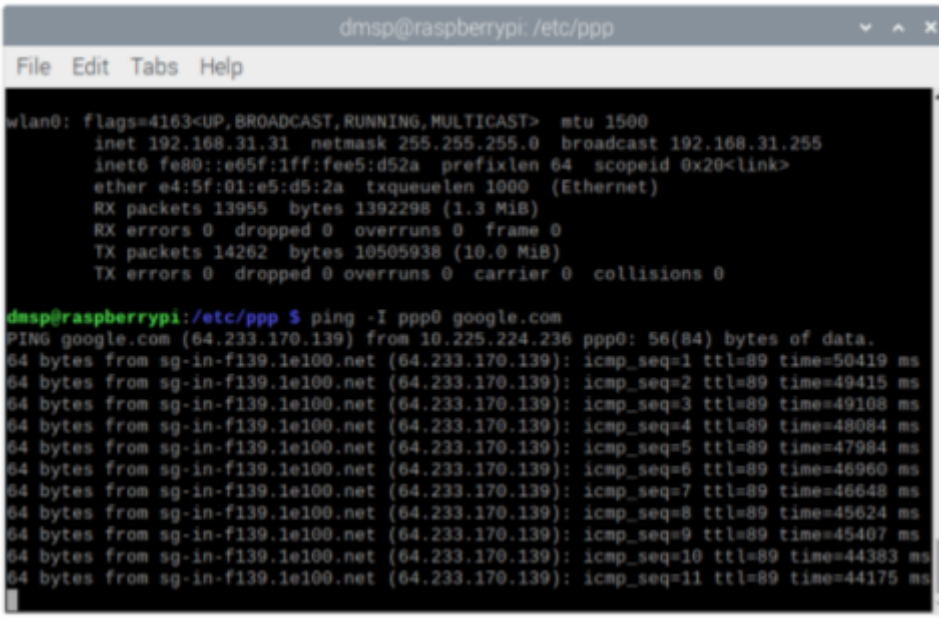
```
AT+CNACT=0,1
AT+CGDCONT=1,"IPV4V6","soracom.io"
```

Then, run the following commands in the terminal to start the PPP and turn of the LAN interface.

```
sudo ifconfig eth1 down
sudo ifconfig eth0 down
sudo on
```

If the things work normally, you can use 'iwconfig' to see a new network interface 'ppp0' as shown in the following figure. Then use the ping command to check the network access.

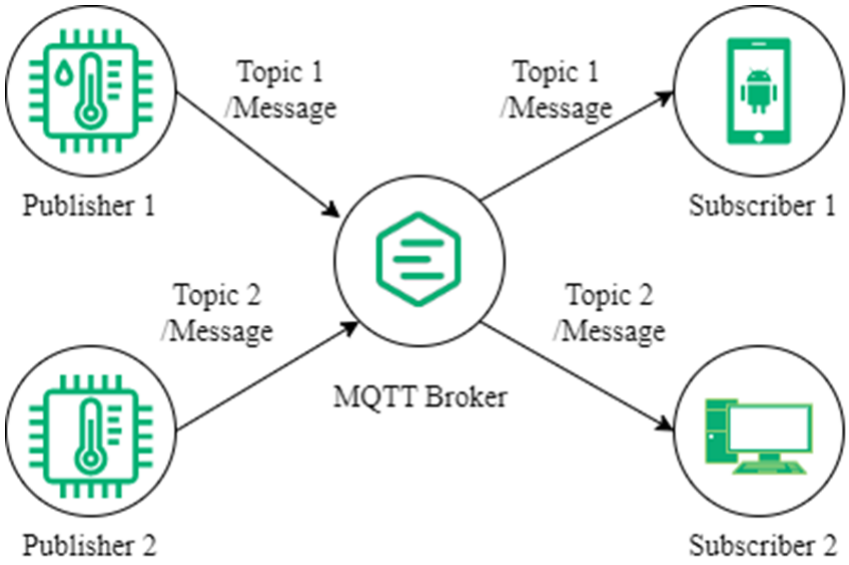




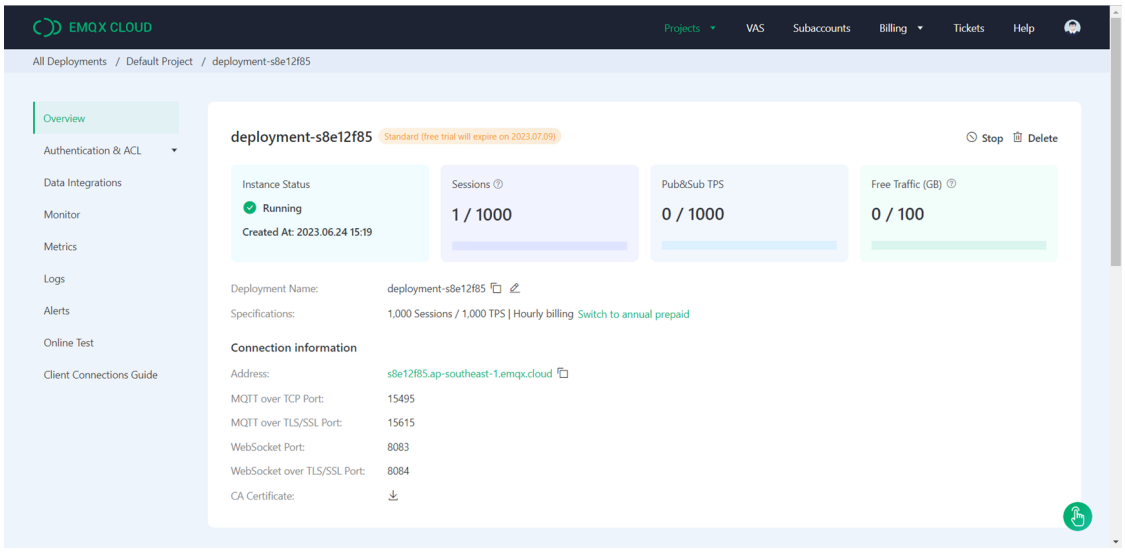
You can observe from the above figure, the delay can be quite large when using PPP for transmission. And some times, the connection is unstable. Thus, a better way is directly send messages through the SIM7080 module using the AT commands. We will discuss this method in the next session.

- MQTT protocol

  Mqtt is a widely-used protocol in the IoT applications, beacuse it's lightweight, efficient, and open-source. Here is the working mechanism of the MQTT protocol. The publisher sends messages to the MQTT broker under a certain topic. And the broker will automatically push these messages to the subsribers of the topic. And in the IoT applications, differents sensors will have independent topics. For exmaple in our case, we send the acceleration data to one topic and the GPS data to another.

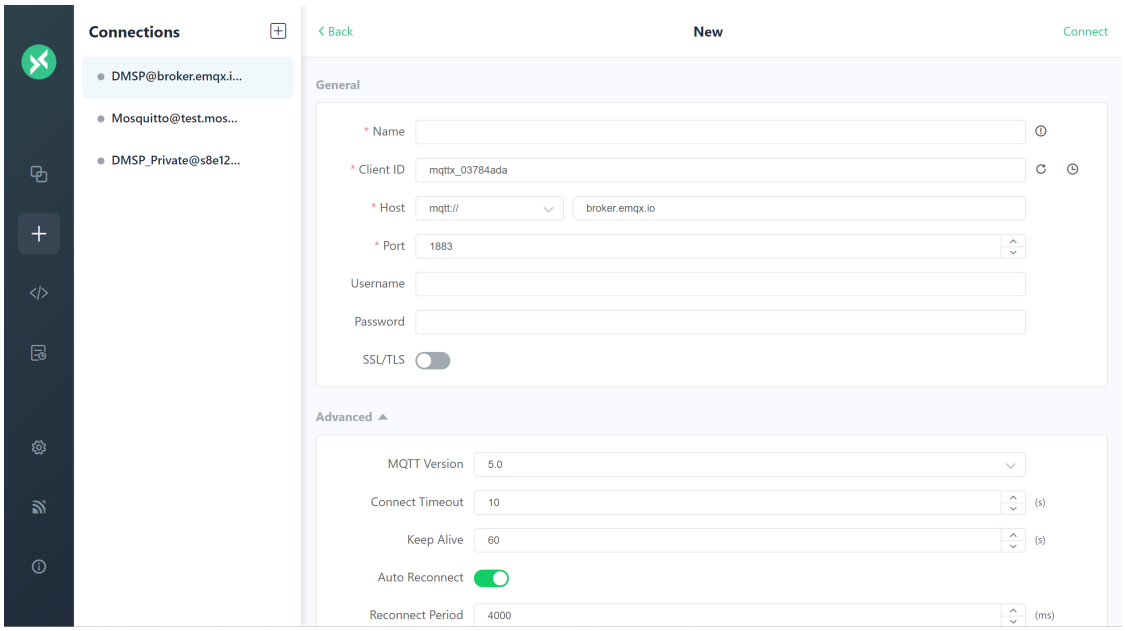


The essential part of this protocol is the broker. In our platform, we use EMQX cloud as our broker. We can directly use their public broker 免费的公共 MQTT 服务器 | EMQ (emqx.com) without registration. Meanwhile, you can also deploy a private MQTT server, the free tier is enough for us to use, and you can also have a 14-day free-trails for the standard or profestion tier.



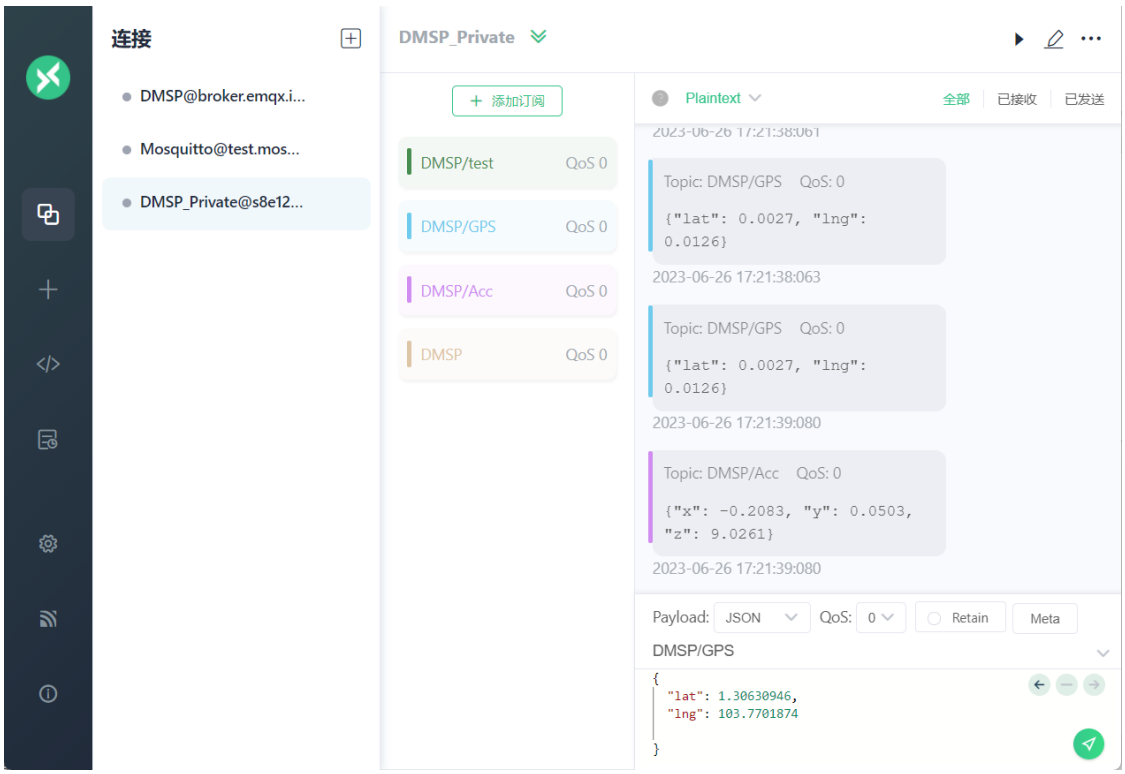
After deployment, you can check the adress, port number in webpage. **You have to create an authentication by specifying a username and a password before using the broker.**

Then, I am going to introduce the MQTTX MQTTX: Your All-in-one MQTT Client Toolbox, a desktop MQTT client.



You can connect to the mqtt broker you deployed just now by specifying the host address (mqtt:// for tcp connection, ws:// for websocket connection; in our project, we only use the tcp connection), the port number, username, and the password. Do not need to modify other parameters.

After get connected, you can subsribe different topics to receive messages and publish messages to certain topics as shown in the following figure.



- Receive messages through MQTT

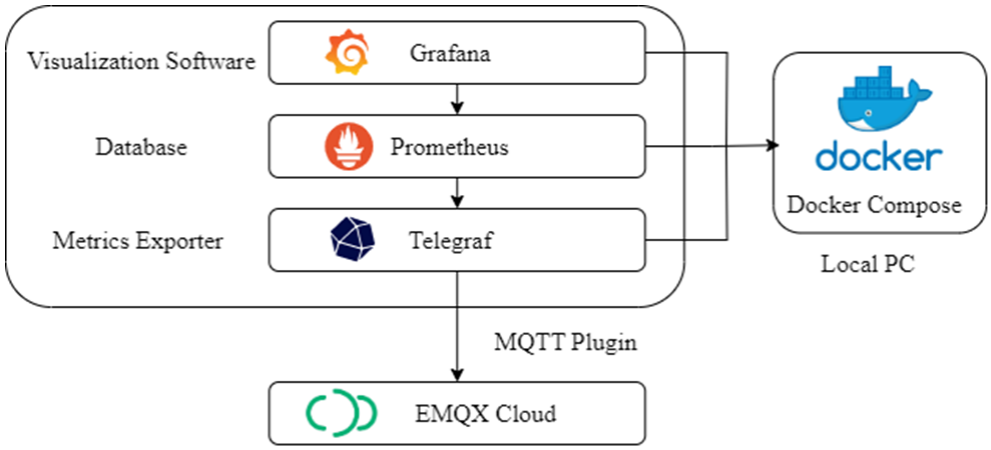
  We can use the mqtt library in Python to send and receive messages through the MQTT protocol, the scripts 'data_publisher.py' is to transmit the data collected by the sensor to the broker. **Remember to encode the original python dictionary into json formats.**

```python
if __name__=="__main__":
    # For WiFi or Enthernet
    mpu = mpu6050.mpu6050(0x68)
    gps = l76x_gps.set_gps()
    client = wifi_mqtt_connection("s8e12f85.ap-southeast-1.emqx.cloud", 15495)
    while True:
        accel_data = mpu.get_accel_data()
        accel_data = json.dumps({key: round(value, 4) for key, value in accel_data.items()})
        gps_data = l76x_gps.get_gps(gps)
        gps_data = json.dumps({key: round(value, 4) for key, value in gps_data.items()})
        wifi_mqtt_publish(client, 'DMSP/Acc',accel_data)
        wifi_mqtt_publish(client, 'DMSP/GPS',gps_data)
        time.sleep(1)
    client.loop_forever()
```

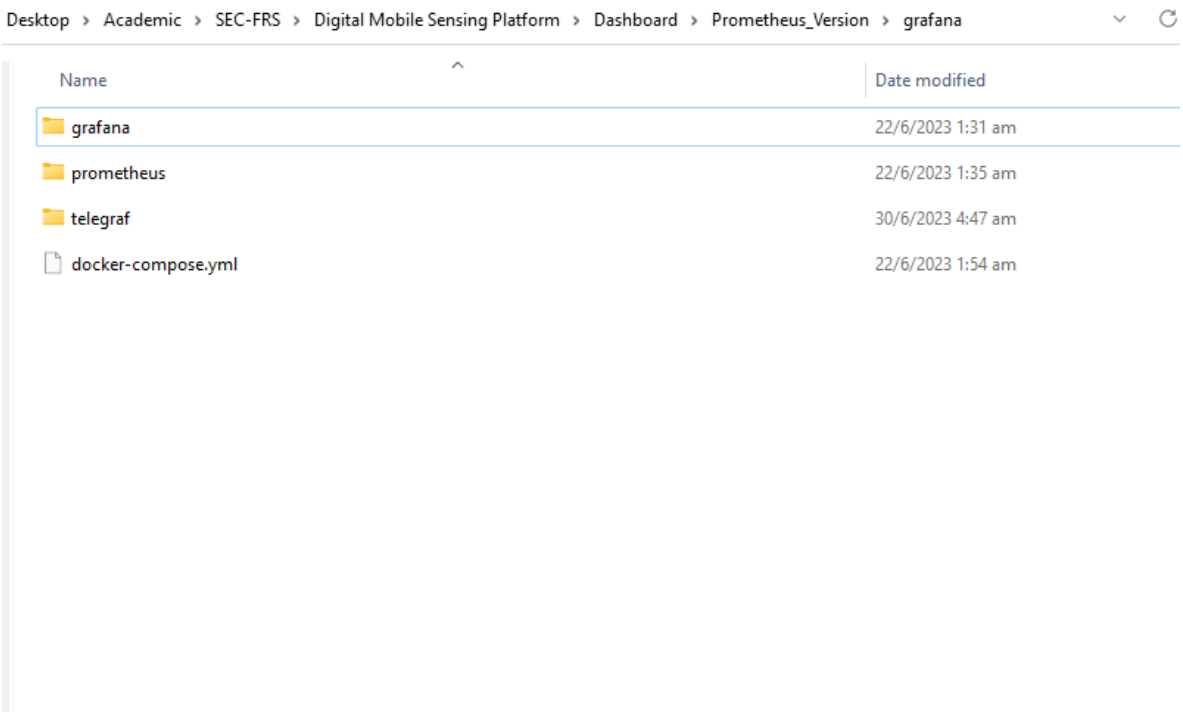
After running the scripts, you can check the received data in the MQTTX to validate the data transmission part.

- Data Visualization

  We use telegraf to subscribe to the MQTT broker, receive messages, and process these messages to certain formats so that the Prometheus databse can recongnise the data and store them. Then we use the prometheus as the data source and visualize the metrics in the grafana. All the three softwares run in a virtual environemtn, so that we don't need to worry about the computer systems or software versions. And it's noteworthy that the majority of the visulization work can be done in a GUI, so we only need to write configuration files.



The configuration files are included in the following directory.



Run the following commands under the directory shown above to start the virtual environment. **Remember to install and start Docker first.**

```
docker compose up -d
```

After starting up the environment, check the running status of each software by

```
docker compose ps
```

The normal working status is like

```
compose ps
NAME                IMAGE                     COMMAND                  SERVICE             CREATED             STATUS
          PORTS
grafana             grafana/grafana:latest    "/run.sh"                grafana             11 seconds ago      Up 9 secon
ds        0.0.0.0:3001->3000/tcp
prometheus          prom/prometheus:latest    "/bin/prometheus --c…"   prometheus          11 seconds ago      Up 9 secon
ds        0.0.0.0:9090->9090/tcp
telegraf            telegraf:1.16.0           "/entrypoint.sh tele…"   telegraf            11 seconds ago      Up 9 secon
ds        8092/udp, 8125/udp, 8094/tcp, 0.0.0.0:9273->9273/tcp
```

Always Remember to stop and remove the container after using the environment by running the following commands

```
docker compose stop
docker compose rm
```

Then, we can check the running status of each component in the web browser. The address of the three components are:

telegraf: 127.0.0.1:9273/metrics

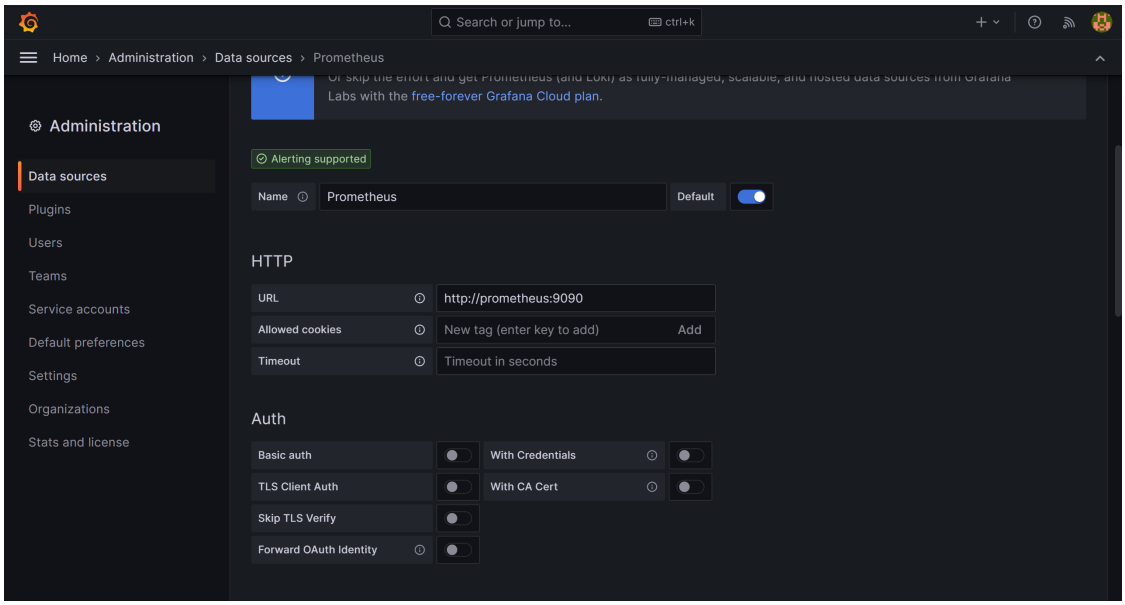
prometheus: http://127.0.0.1:9090/

grafana: Grafana

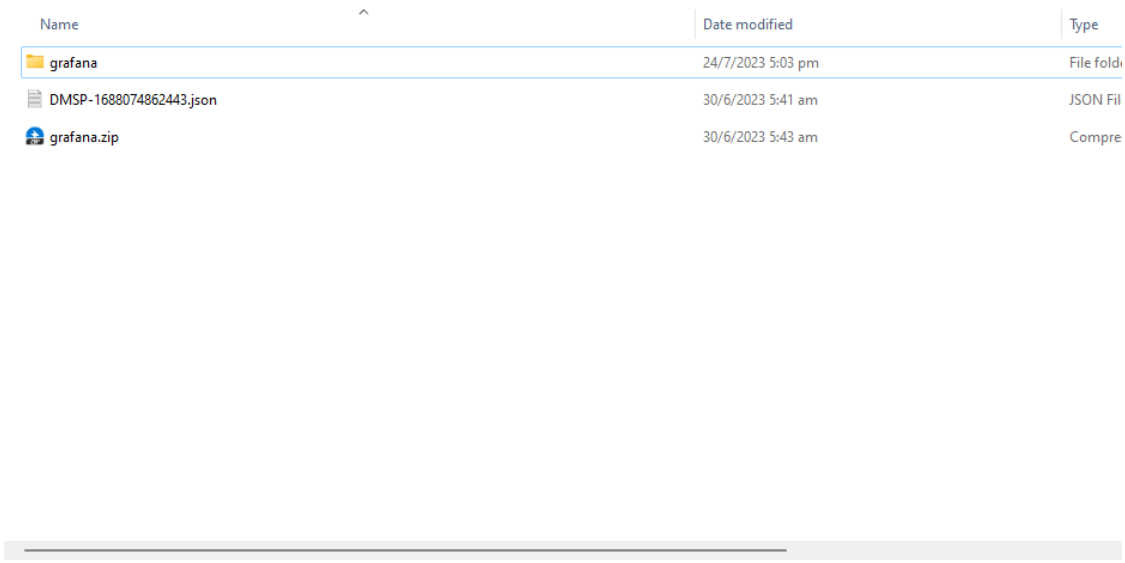
For grafana, the default user name and password are'admin'. Here, I bind the grafana to the 3001 port, becasue if you install another grafana locally, the default port number is 3000.

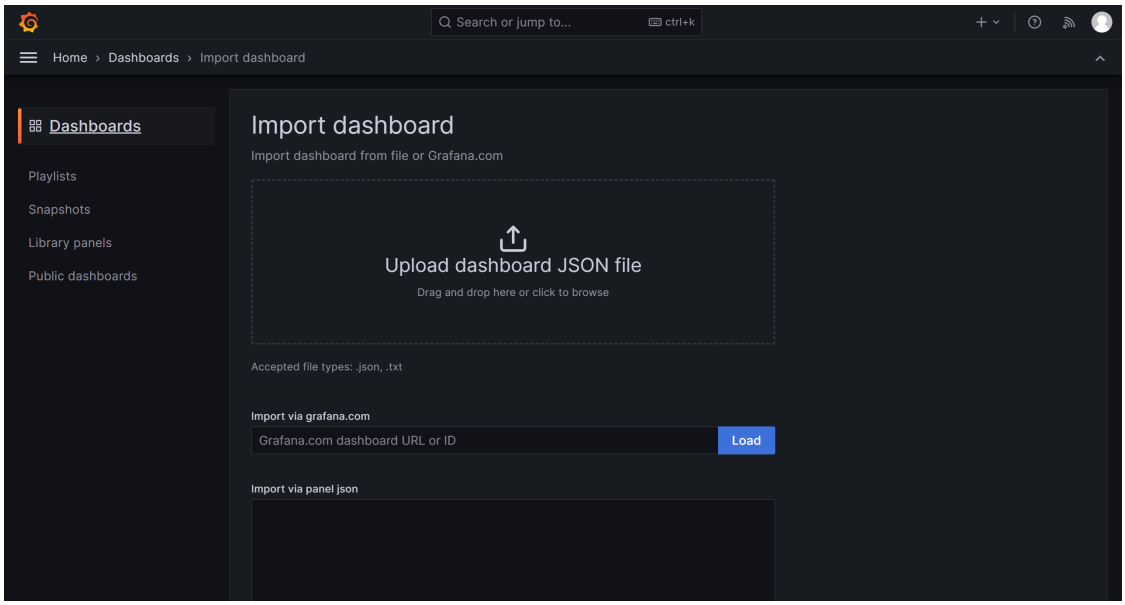
- Grafana dashboards

  Before importing the dashboard, create the data source using the url shown in the following figure.

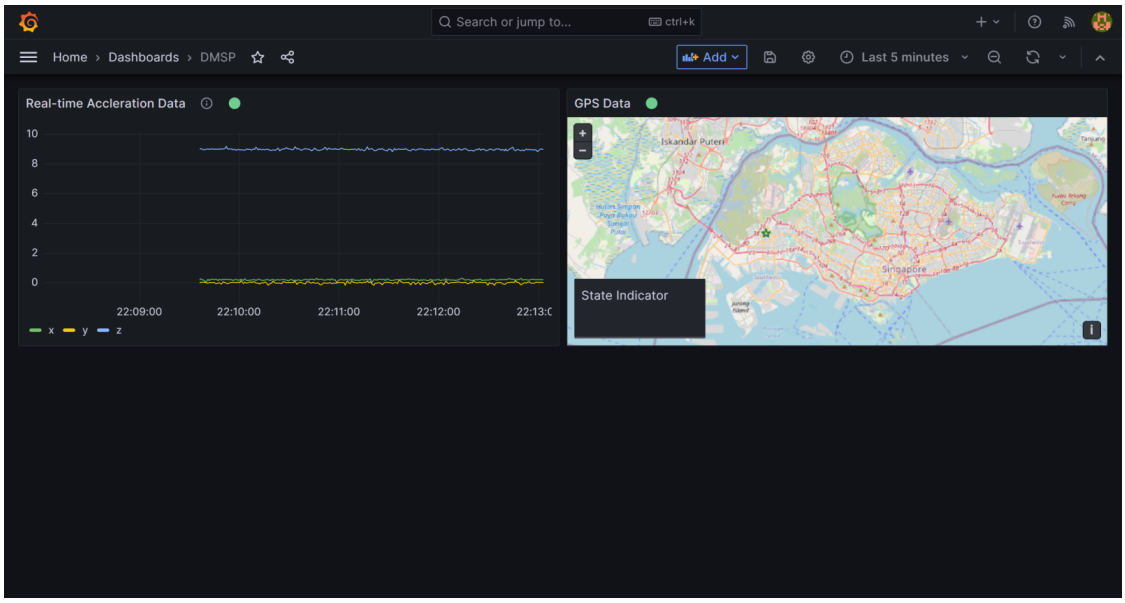


After adding the prometheus as the data source, the grafana dashborad can be imported using the json file.





Then, you can see visualization of the received data.

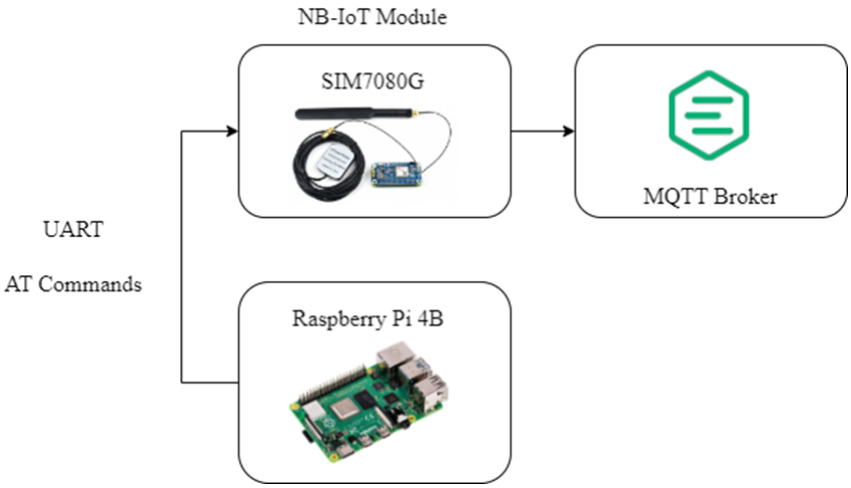


Future work

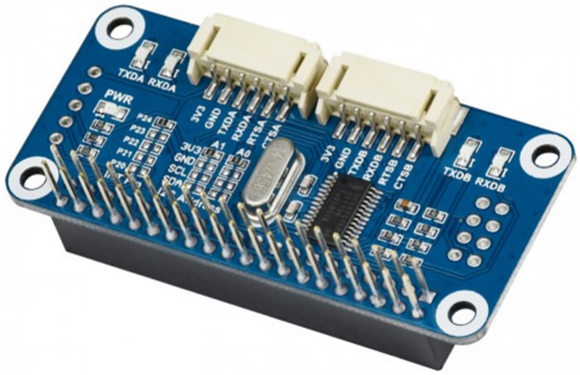
- Get rid of PPP connection

  As we discussed before, the PPP connection is unstable. Meanwhile, the SIM7080 also supports MQTT publishing and receiving by AT commands.

  I have implemented the script 'sim7080_mqtt.py' for achieving this goal. In the test, it shows great stability. However, this method requires to use the UART interface to communicate with the SIM7080, which causes conflicts with the GPS module.

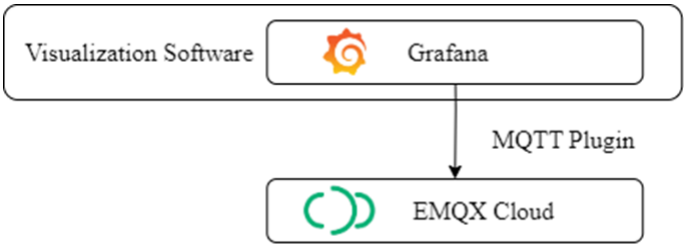


The solution is pretty simple, we should by an UART extention for the Raspberry Pi [Serial Expansion HAT - Waveshare Wiki](#).

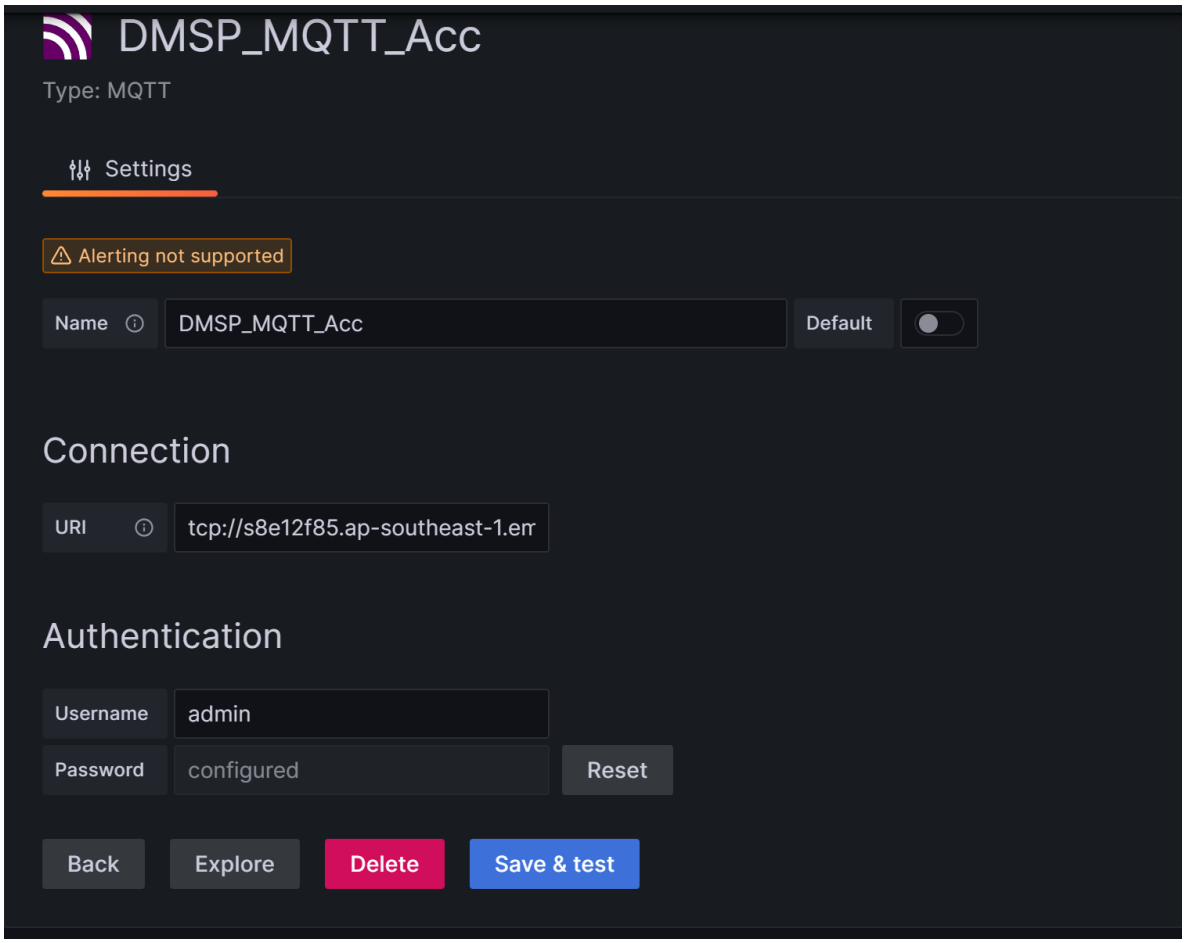


- Use the MQTT plugin in Grafana

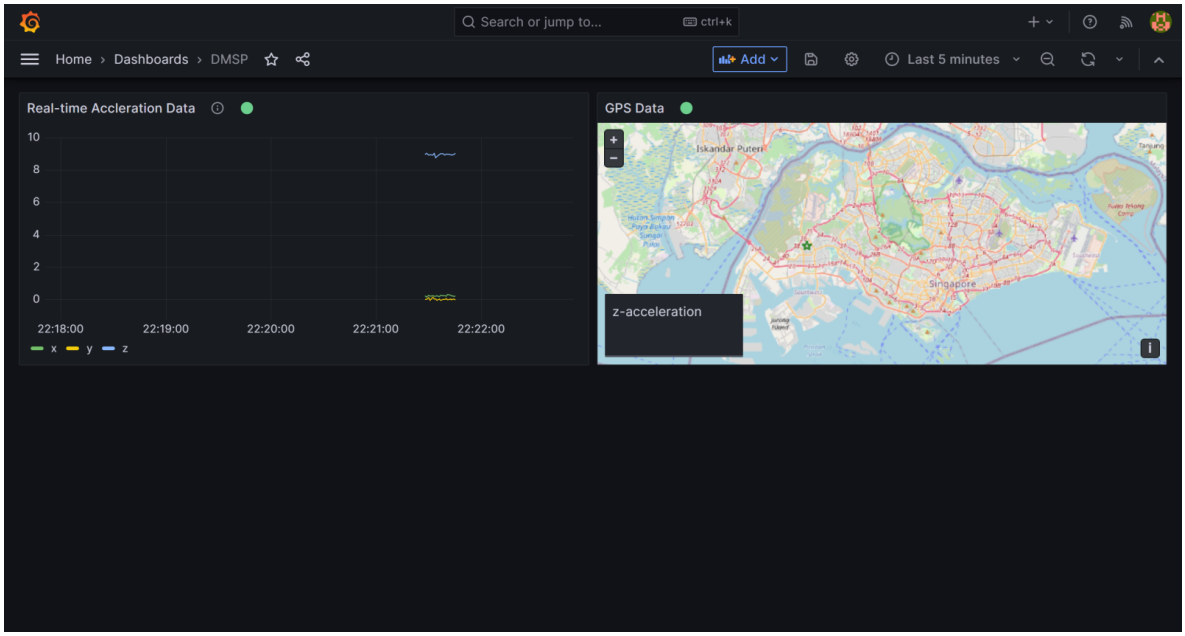
  The Grafana has a plugin to support MQTT as the data source, so that the data path of the visualization part can be simplified as following.



  To use it, you can install the MQTT plugin in the Grafana console, and set the MQTT as the data source as shown in the following figure. **The url should start with 'tcp://' if you are using tcp connection, different from MQTTX connection.**



Howvere during the test, I found the data streaming is not stable and automatically stops as shown in the following figure.



I have checked their github repo and found some users discussing this issue [Live data stops being showed · Issue #44 · grafana/mqtt-datasource (github.com)](). Someone has raised a pull request for fixing this issue [Fix for issue #44 by isarantidis · Pull Request #76 · grafana/mqtt-datasource (github.com)](). Thus, I believe the problem can be solved soon.