

Week0510.20230502-Grafana and InfluxDB

กลุ่มคำานวน ____ รหัส B6321451 ชื่อ-สกุล นางสาวขวัญจิรา พันธุเกตุ

1/4 การทำรายงาน

- เพิ่มเติมเนื้อหาด้านท้าย ด้วยการคัดลอกและจัดเรียงใหม่ (3/4)
- รูปที่เป็นการทดสอบ ESP32 ควรเป็นรูปของตัวเองที่ทดสอบ (4/4)

2/4. Read More

- <https://grafana.com/blog/2021/03/08/how-i-built-a-monitoring-system-for-my-avocado-plant-with-arduino-and-grafana-cloud/>
- <https://www.metricfire.com/blog/iot-dashboards-with-grafana-and-prometheus/>
- <https://gabrieltanner.org/blog/grafana-sensor-visualization>

3/4. ให้จัดเรียบเรียงข้อมูล

ESP8266 / ESP32 & Mesh Network

- ESP Mesh Network 1 - <https://meetjoeblog.com/2018/03/25/esp8266-esp32-mesh-network-ep1/>
- ESP Mesh Network 2 - <https://meetjoeblog.com/2018/03/27/esp8266-esp32-mesh-network-painlessmesh-client-server-ep2/>
- ESP Mesh Network 3 - <https://meetjoeblog.com/2018/03/30/esp8266-esp32-mesh-network-painlessmesh-bridge-ep3/>
- ESP Mesh Network 4 - <https://meetjoeblog.com/2018/04/25/esp8266-esp32-painlessmesh-bridge-with-lora-ep4/>
- ESP Mesh Network 5 - <https://meetjoeblog.com/2018/08/30/esp8266-esp32-mesh-network-ep5-influxdb-grafana/>

Arm-Pelion Full Stack IoT Platform

- <https://www.techtalkthai.com/arm-pelion-full-stack-iot-platform/>

Grafana Dashboard

- <https://developers.ascendcorp.com/ทำความรู้จักกับ-grafana-dashboard-1a5efe6d170a>

4/4. การทดสอบ

- ให้ทำการทดสอบและเขียนขั้นตอนการทดสอบ โดยใช้ ESP32 ส่งข้อมูลไปยัง MQTT Broker และใช้ Grafana .
ในการมอนิเตอร์ข้อมูล โดยปรับแก้การทดสอบจาก <https://gabrieltanner.org/blog/grafana-sensor-visualization>

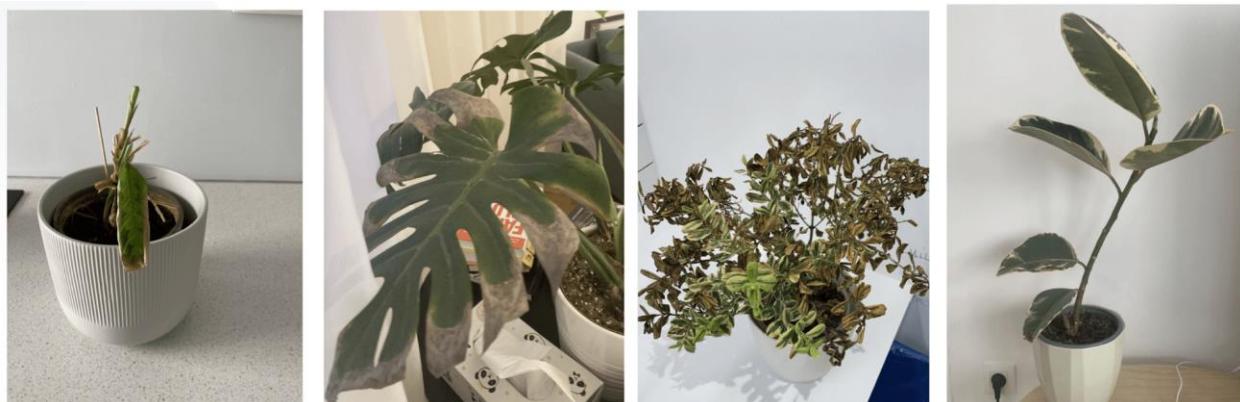
Visualize Sensor data using Grafana and InfluxDB

2/4. Read More

<https://grafana.com/blog/2021/03/08/how-i-built-a-monitoring-system-for-my-avocado-plant-with-arduino-and-grafana-cloud/>

How I built a monitoring system for my avocado plant with Arduino and Grafana Cloud

A couple months ago, during our Grafana hack days, I created my first monitoring solution: my [sourdough monitoring system](#). It was a lot of fun to build it, and I enjoyed it a lot! So when the next Grafana hack day was approaching, I started to wonder what my next monitoring system could be. What would I like to learn more about? What would I like to get better at doing? To be honest, I didn't have to think hard. I just looked around, and there they were: my beautiful plants that I love, but for sure don't know how to take care of. So I decided to do my best to understand them and build a **monitoring system for my plants**. Specifically, a monitoring system for my avocado plants.



What to monitor?

Whenever I start with any monitoring project, I begin with writing down a list of variables that could tell me more about the system that I would like to monitor. In this case, it is the external outputs of the plant and also the environment in which it is growing. My goal is to list as many variables as I can think of, and then figure out what could actually be monitored and what I would be able to build. In this project I am going to be monitoring:

- Height of the plant
- Air temperature
- Air humidity
- Soil moisture
- Light conditions

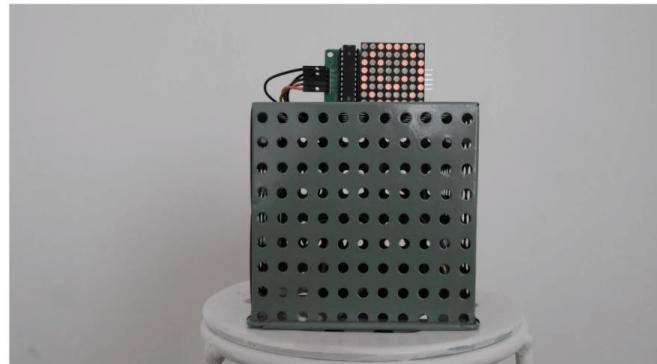
The hardware

To build my avocado plant monitoring system, I used:

- 1 **ESP32 WiFi Bluetooth 4MB Flash UNO R3** development board
- 1 **HC-SR04** ultrasonic distance sensor to measure the height of the plant
- 1 **DHT22** sensor module to measure the temperature and humidity of the environment
- 1 **Soil Moisture** sensor module to measure the soil moisture
- 1 **TEMT6000** light sensor to measure the light intensity
- 1 **LED display MATRIX MAX7219** to show current conditions of the plant directly on the monitoring system
- 1 **25 points breadboard**
- **M-F Dupont Cables**
- 1 **micro USB cable**
- 1 **USB charger**

You'll also need:

- 1 **plant stand** to put this all together and build the monitoring system



The software

You'll need to install the [Arduino IDE](#) open source electronics platform for easy-to-use hardware and software. Arduino compiles the code and then uploads it to your board. I've installed version 1.8.10, since newer versions were throwing errors on OS Big Sur, which I use. It worked well.

If your OS won't recognize the USB serial automatically, you'll probably need to install [CP210x USB to UART Bridge VCP Driver](#).

Setting up Grafana Cloud

The next step is to set up our databases and Grafana. I have decided to use [Grafana Cloud](#) — which comes with hosted Grafana, [Grafana Loki](#), and Graphite — for our data storage and data visualization. The [free tier](#) comes with 10,000 series for Graphite metrics and 50GB for logs in Loki, which is definitely more than enough for this monitoring solution.

I went to the [Grafana Cloud signup](#) and created a new account. As soon as I had my account all set up, I could see my portal with my hosted Grafana, Loki, and Graphite instances.

Grafana Cloud Portal

Configure your Grafana Cloud resources and manage users, API keys, billing and more.

[Contact Support](#)

ivhuc

Manage your Grafana Cloud Stack.



Grafana

[Log In](#)

[Details](#)

Log in to start using Grafana or view Details to manage Grafana Plugins.

↳ Active Users: 0



Prometheus

[Send Metrics](#)

[Details](#)

Set up and manage your Prometheus metrics service.

↳ Active Series: 0



Loki

[Send Logs](#)

[Details](#)

Set up and manage your Loki logging service.

↳ Ingest Rate: 0 bytes/hr



Graphite

[Send Metrics](#)

[Details](#)

Set up and manage your Graphite metrics service.

↳ Active Series: 0



Alerts

[Configure](#)

[Details](#)

Set up and manage your Prometheus-based alerts.



Tempo BETA

[Send Traces](#)

[Details](#)

Set up and manage your Tempo tracing service.

At this point, I've also created my API keys for Loki and Graphite to publish metrics from my monitoring system to these databases.

avocado-publisher-graphite
MetricsPublisher role, issued on February 28, 2021

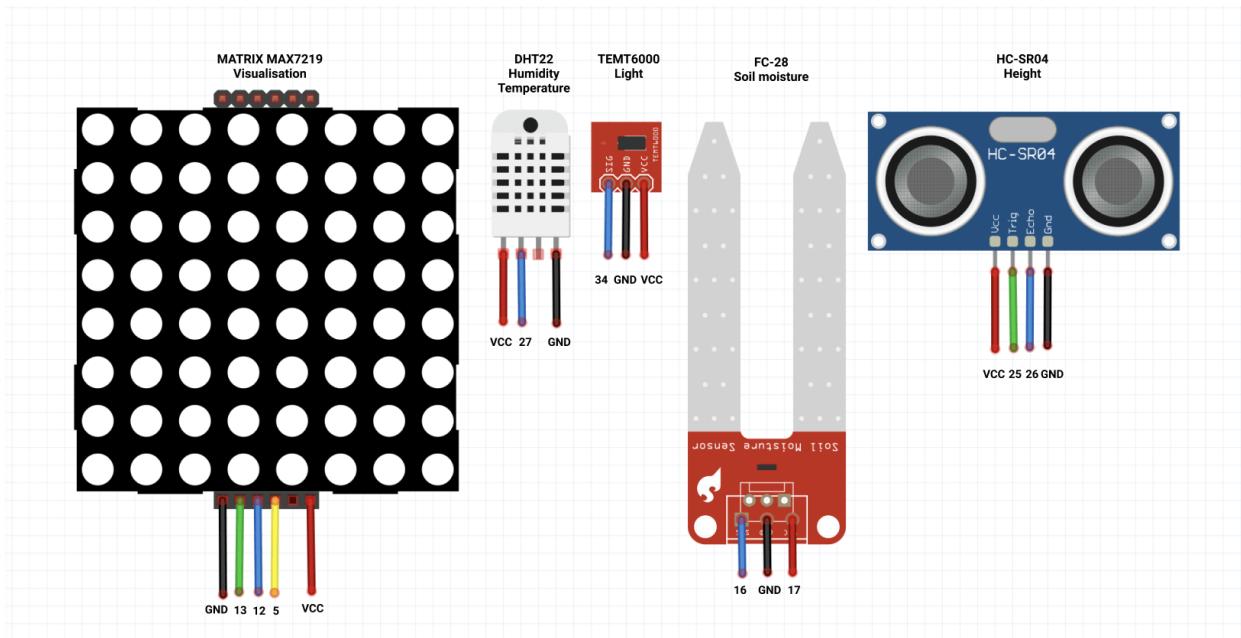
Delete

avocado-publisher-loki
MetricsPublisher role, issued on February 28, 2021

Delete

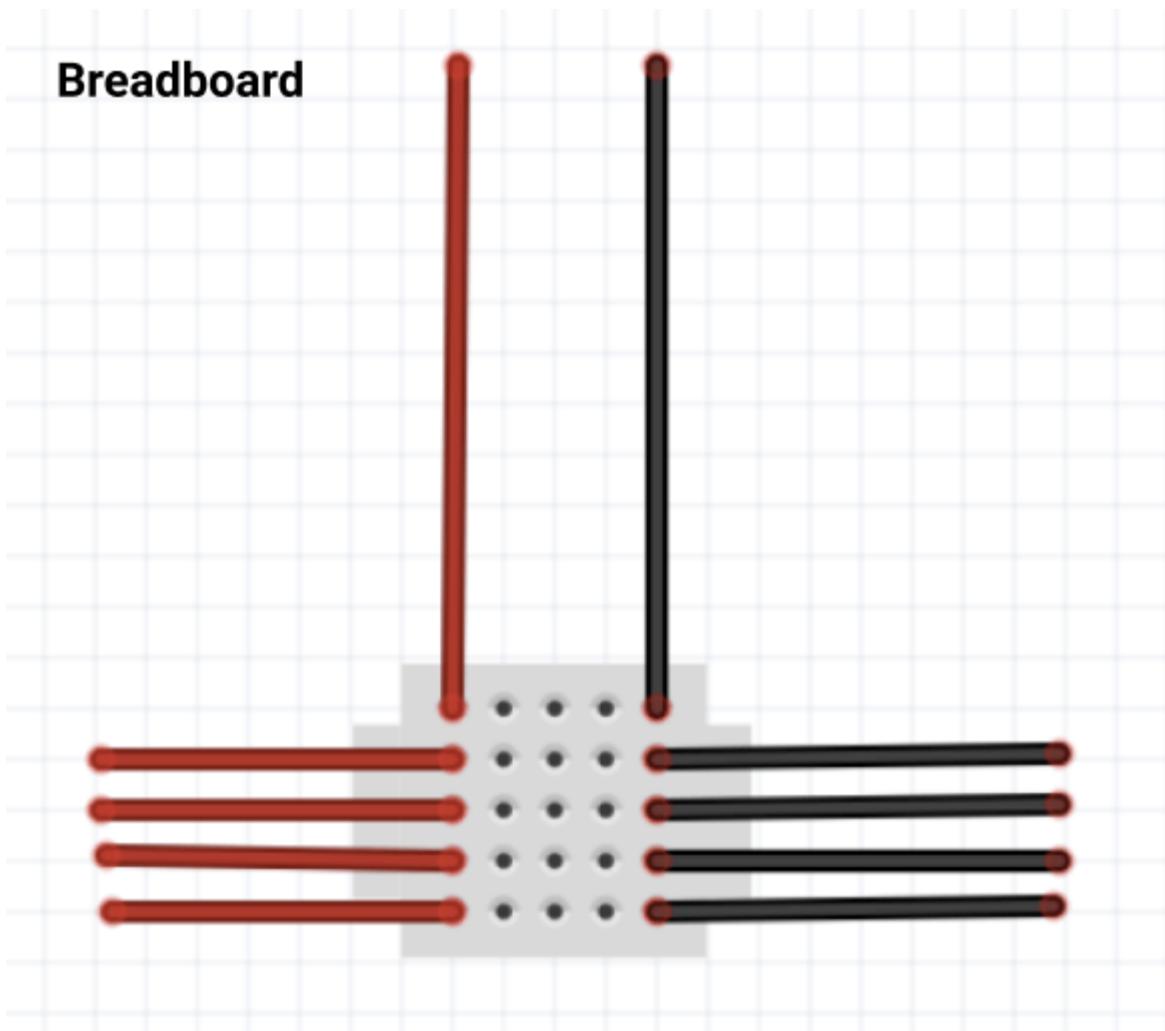
The hook-up

Now it's time to start putting things together. Below, you can see the pinouts of all sensors and outputs:



As I had more inputs and outputs than power and ground pins, I also used the 25 points breadboard. I wired it to the **vcc pin** and **gnd pin**, so I created 3 extra gnd and vcc pins that I will use for my inputs and outputs.

Breadboard



LED display MATRIX MAX7219

The 8x8 LED matrix is often used to display simple graphics or texts. I decided to use it to display the state of my plant. If the plant has perfect conditions, I see a smiley face on the LED matrix. On the other hand, if the soil is dry or the temperature is too low or too high, a sad emoticon is displayed.

Pinout:

- VCC pin to 5V on the breadboard
- DIN data input pin to pin 5 on board
- CS chip select pin to pin 13 on the board
- CLK serial input pin to pin 12 on the board
- GND pin to GND on the breadboard

The DHT22 sensor module

This is a very basic and cheap sensor made of two parts: a humidity sensor and a thermistor. In this case, it is used to measure the temperature and humidity of the environment in which my avocado plant lives.

Pinout:

- VCC pin to 3.3V on the board
- DATA pin to pin 27 on the board
- GND pin to GND on the breadboard

The TEMT6000 light sensor

This light sensor will detect the brightness of my avocado plant's surroundings. TEMT6000 measures illuminance that is measured in lux. TEMT6000 is very intuitive to use: More current means brighter and less current means darker.

Pinout:

- VCC pin to 5V on the board
- SIG output voltage pin to pin 34 on the board
- GND pin to GND on the board

The soil moisture sensor

The soil moisture sensor is one of the most important sensors in this project. It is going to let me know if the soil of my plant is wet or dry, and it is going to help me figure out when to water it.

Pinout:

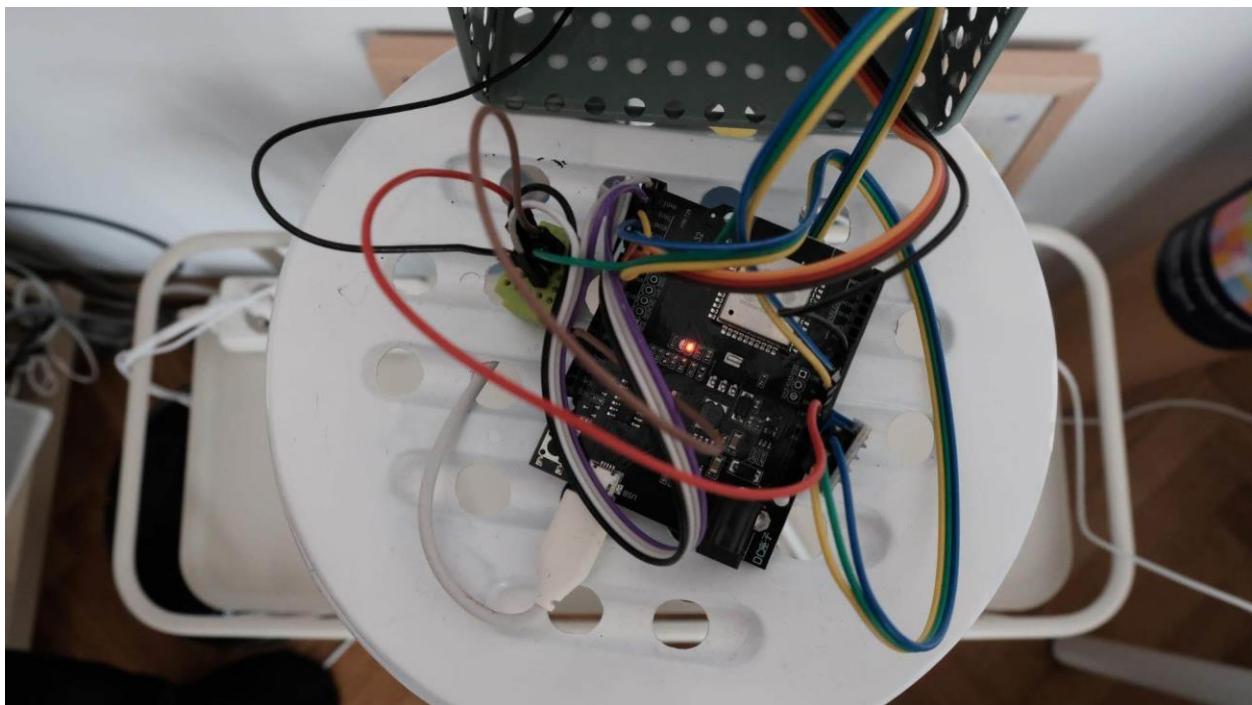
- VCC pin to pin 17 (I am not using the vcc pin to prevent corrosion)
- SIG output voltage pin to pin 16 on the board
- GND pin to GND on the board

The HC-SR04 ultrasonic distance sensor

The ultrasonic sensor uses sonar to determine distance to an object. It has high accuracy and stable readings in a range from 2 cm to 400 cm. I am using this sensor to measure the height of my plant.

Pinout:

- VCC pin to 5V on the breadboard
- TRIG pin that sends the signal to pin 25 on the board
- Echo pin that receives signal to pin 26 on the board
- GND pin to GND on the breadboard

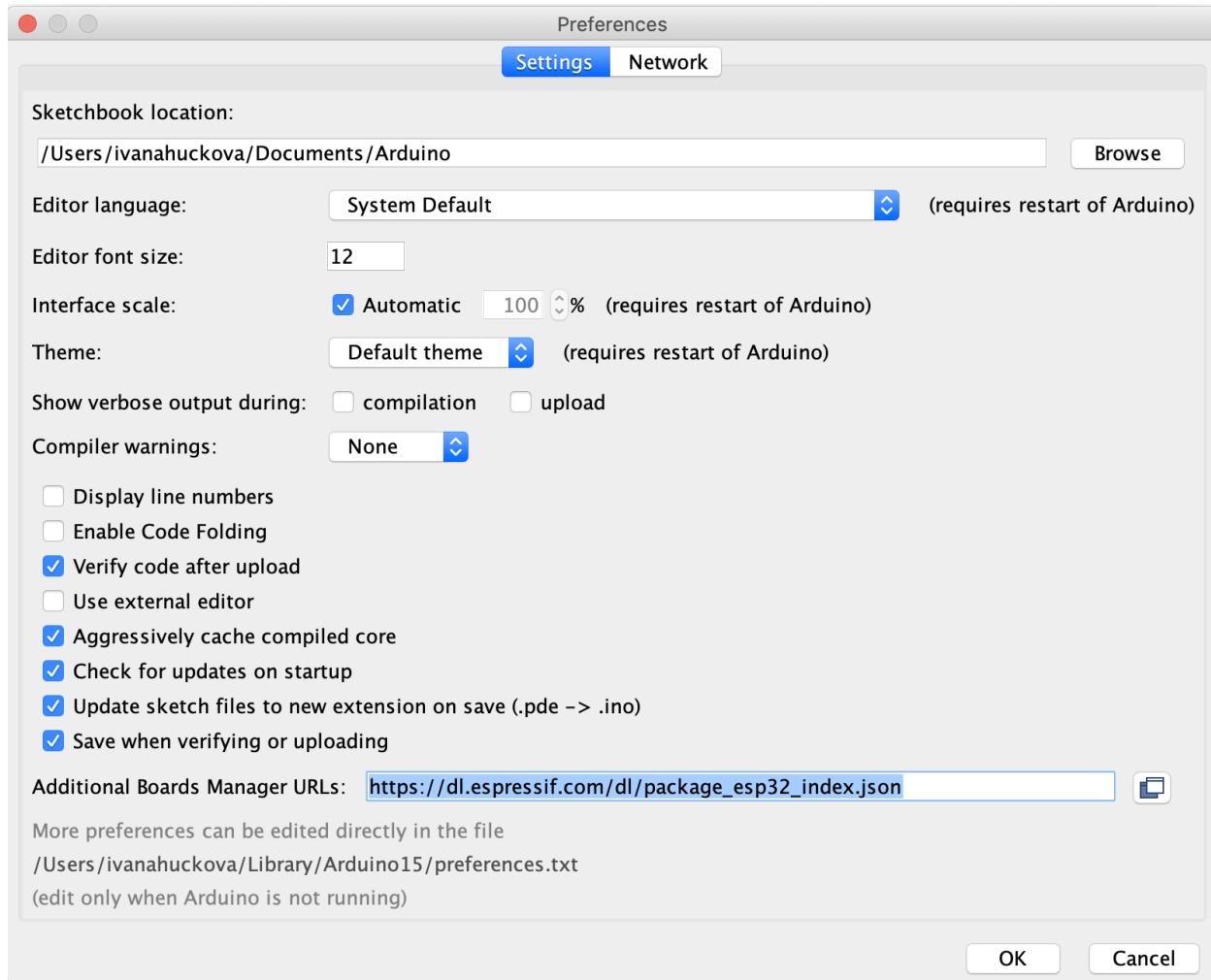


The programming

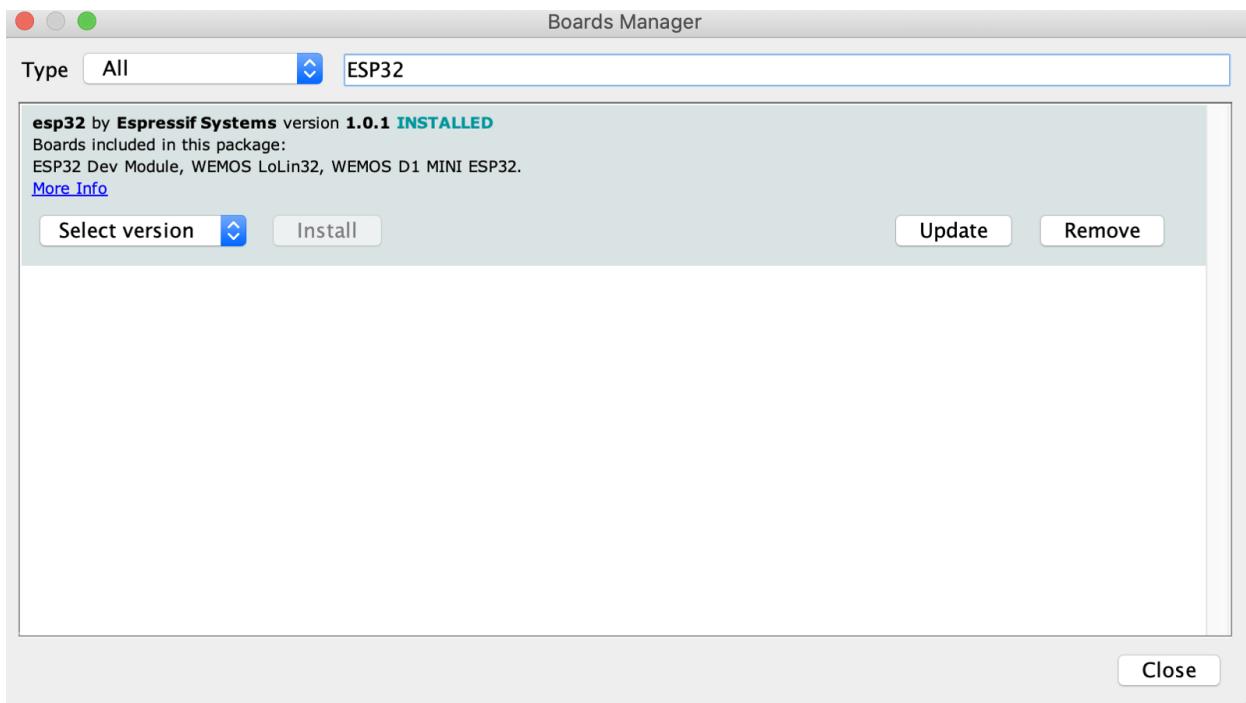
Once I had my sensors and development board ready, I started with writing a program that is going to retrieve data from the sensors and send it to the Graphite and Loki database. I am using an Arduino IDE to upload the program to the development board. Therefore, as a first step, I have connected my ESP32 board with all sensors through a microUSB to my computer.

Step 1: Setting up Arduino IDE to support your ESP32 board

After my ESP32 board is connected to my computer, I continue with opening my Arduino IDE. As I am using an ESP32 board (and not an Arduino board), I need to add a board definition that adds support for my EP32 board. In Arduino > Preferences, I have added the url https://dl.espressif.com/dl/package_esp32_index.json to the **Additional Boards Manager URLs** input. This open source board definition adds support for programming ESP32 boards.



Then, in Tools > Boards > Boards Manager I have added ESP32 board manager.



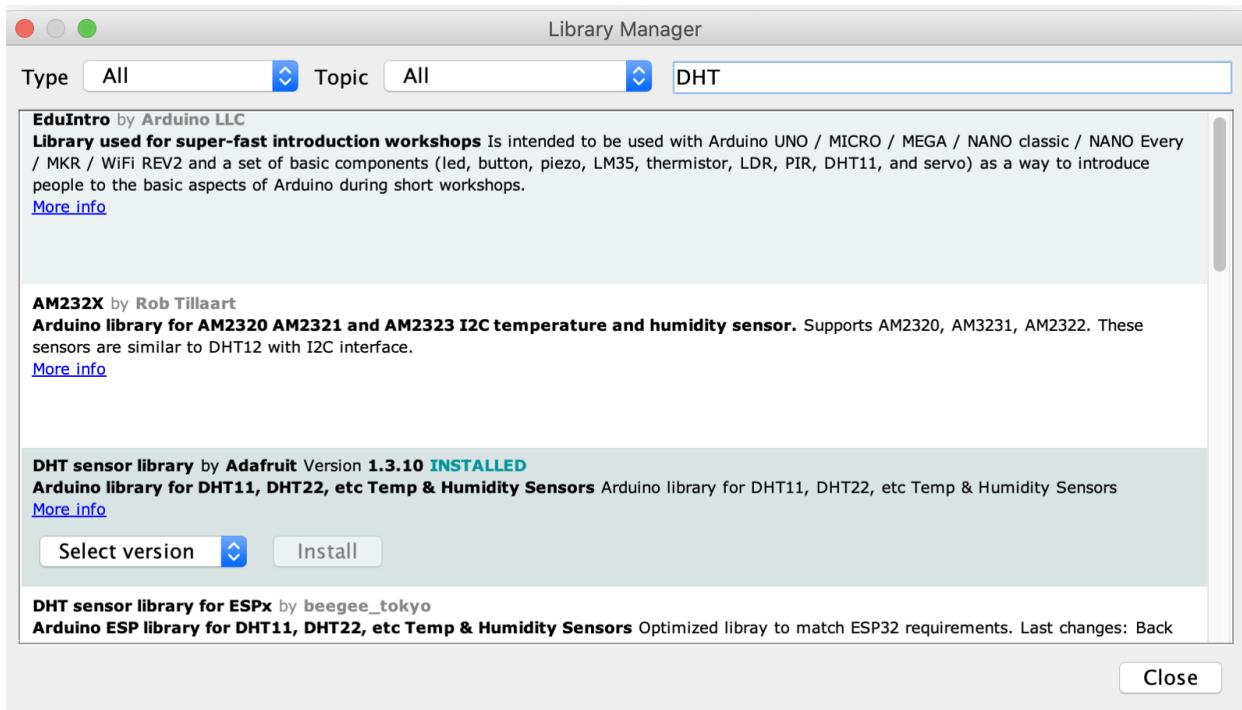
Lastly, in Tools > Boards, I have chosen **DOIT ESP32 DevKit**. This tells the Arduino IDE which profile and base libraries to use when compiling the firmware image, and how to flash it to the board. Make sure that in the Tools > Port submenu, you've selected the new COM port.

Step 2: Adding libraries

For this project, I used the following libraries:

- [DHT sensor library by Adafruit](#)
- [Adafruit Unified Sensor by Adafruit](#)
- [NTP Client by Fabrice Weinberg](#)
- [HCSR04 by Martin Sosic](#)

I installed all of these libraries in Sketch > Include Library > Library Manager. It is important to have these libraries installed before moving to the next steps.



Step 3: Create a program

Arduino natively supports a language called the Arduino Programming Language (APL), which is basically a framework built on top of C++. A program written in APL is called a sketch and has the .ino extension. Every Arduino program has to have two special functions that are a part of every sketch: `setup()` and `loop()`. The `setup()` is called once, when the sketch starts. The `loop()` function is called over and over and is the heart of most sketches.

The full sketch can be downloaded on my [GitHub](#). In the repo, there are two important files: **avocado_monitoring.ino** and **config_template.h**. The first file includes the program with detailed comments describing what each section does. In the second one, I am storing my variables and API keys. If you decide to create your own monitoring system, download the repo, rename **config_template.h** to **config.h**, and replace your variables and API keys.

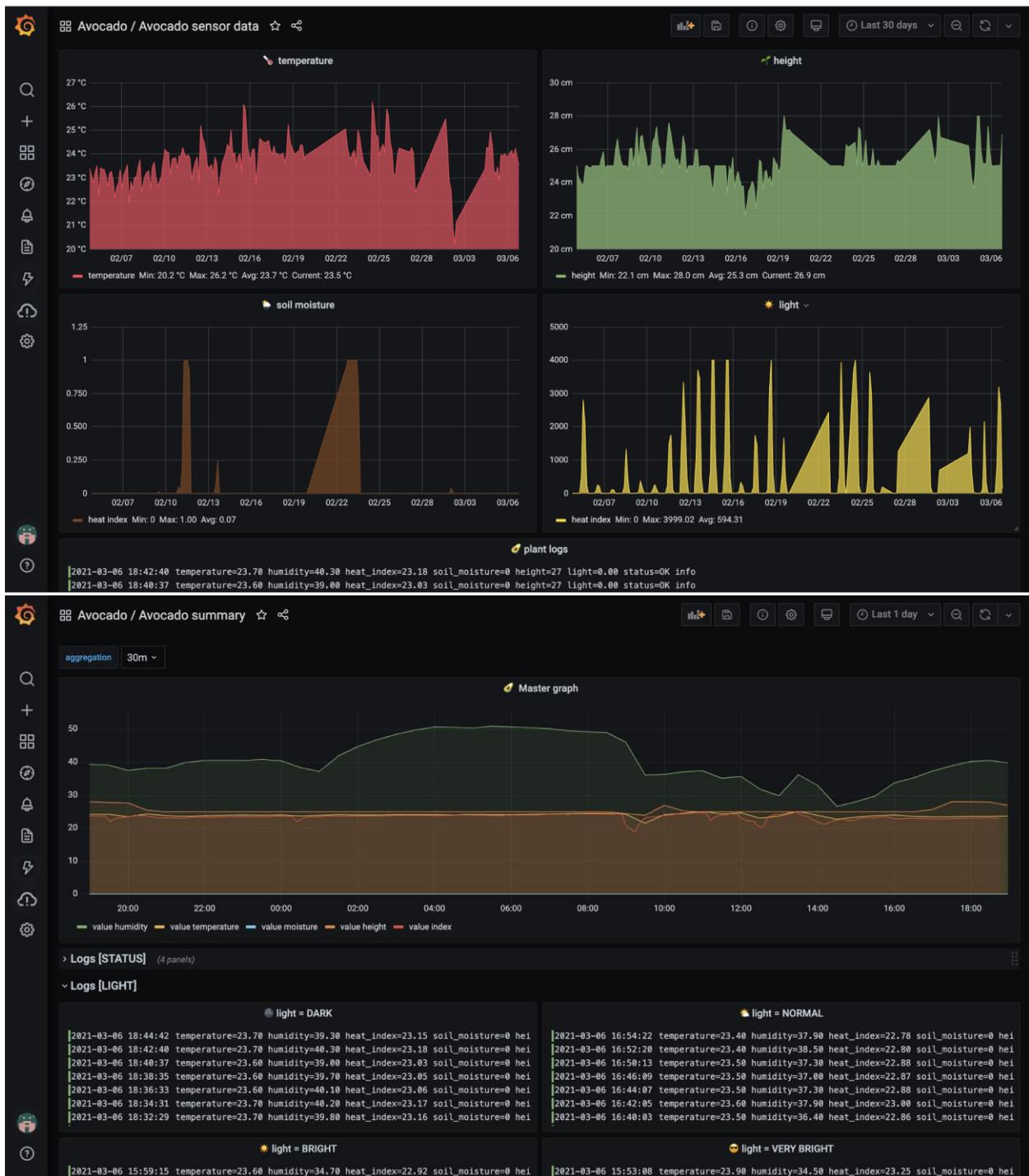
Step 4: Uploading and running the program to ESP32

Now it is time to upload the program to the ESP32 board! It is as easy as opening the `avocado_monitoring.ino` file in Arduino IDE and pressing the **Upload** button (the arrow pointing to the right).

After everything was set up correctly, I was soon able to see the incoming data to my Graphite and Loki database.

Visualize the data in Grafana

As soon as the data from monitoring is flowing to Graphite and Loki, it's time to visualize them in Grafana. I opened the Grafana instance that was set up through my Grafana Cloud signup and started to create dashboards. I created two dashboards, and if you would like to use the same ones, you can just go and copy-paste my dashboards's JSON models in [here](#) and [here](#).



Add the monitoring system to the plant stand

Now comes the last, handy part of the work. 🚀 I mounted my monitoring system on the plant stand using a glue gun, but feel free to improvise and design your own solution.



With this blog post, I wanted to show that it doesn't have to be difficult or complicated to create a monitoring system for basically anything. I hope you'll try this out and [let me know how it goes](#). Or if you decide to build a different monitoring system or IoT project, I would absolutely love to hear about that too!

- <https://www.metricfire.com/blog/iot-dashboards-with-grafana-and-prometheus/>

IoT Dashboards with Grafana and Prometheus

Intro

Internet of Things (IoT) - is a number of physical devices connected to one network that enable the system to interact with the external world. A great deal of the work surrounding IoT is the monitoring, as it's impossible to react without knowing the situation.

For example, we might build a greenhouse system for agriculture that can maintain optimal conditions for growing crops. For this purpose, we need to have sensors picking up information about the temperature and humidity. All this data should be stored and processed automatically to enable watering and heating.

The best IoT monitoring stack

A great stack for IoT monitoring is Prometheus and Grafana, or Hosted Prometheus and Grafana by [MetricFire](#).

The data collection, storage, and processing can be done with [Prometheus](#), and the visualizations can be done with [Grafana](#). This article won't go in depth on the basics of Prometheus and Grafana, instead we will focus on what you need to know to make a great IoT dashboard using these tools.

To get more on the basics, you should read our Prometheus 101 article [here](#), and our article, [Getting Started With Grafana](#).

This article also won't go into how to connect edge devices to Prometheus and Grafana. To learn about how to connect [MetricFire](#) to an IoT device, check out our article on [monitoring a Raspberry Pi 4](#).

Especially for large scale IoT monitoring, you should consider monitoring with [Hosted Prometheus](#) by [MetricFire](#). Hosted Prometheus gives you automatic scaling, updates, plugins, [Grafana dashboards](#) and more. You should sign up for the MetricFire free trial [here](#), and start building your IoT dashboards.

Now, let's take a look at the core concepts behind monitoring IoT, and learn to build a dashboard in Grafana for monitoring an IoT system.

Why we need IoT monitoring

Wouldn't it be great if our IoT system was fully automatic, making it so that humans didn't have to do anything? Why do we even need a dashboard? Can't the water sprinkler in the greenhouse turn on by itself?

Ideally, human involvement in our IoT systems would be 0. However, we all know that this world isn't perfect. IoT systems are made up of hundreds of sensitive components that are vulnerable to weather, mechanical damage, or even battery power.

It is possible, and even common, that other parts of a production pipeline also break up. This can be caused by both people and machines. Additionally, very few businesses are fully automated, so humans still need to participate in the monitoring and operating of IoT systems.

Try MetricFire now!

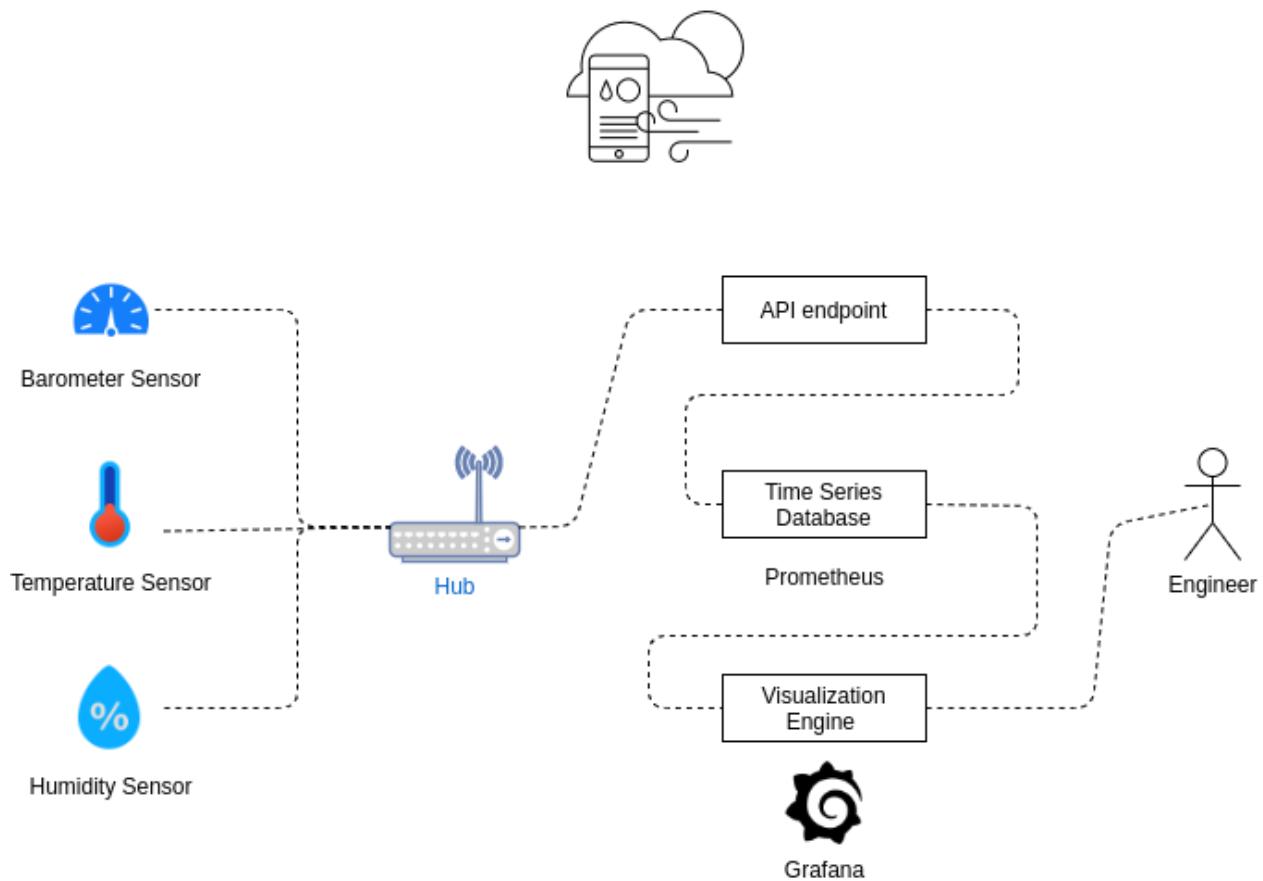
Get MetricFire free for 14 days. No credit card required.

[Request demo](#)

Monitoring a smart home

As an example, we are going to monitor a system that measures pressure, temperature, and humidity at home, store all the data in Prometheus and visualize it with Grafana. Below you can find the possible schema of how this can be implemented.

As you can see, there are 3 edge devices (sensors) that send data to a Hub and then to an API endpoint via MQTT (message queuing telemetry transport - a lightweight messaging protocol for small sensors and mobile devices, optimized for high-latency or unreliable networks) or HTTPS.



You can use either cloud services or [Prometheus Pushgateway](#) to store data in the database. In more complicated tasks you may also need [StatsD](#) to aggregate the data. Then, we can [connect Grafana to Prometheus](#) and build dashboards.

The Architecture of Monitoring IoT

We can divide our monitoring strategy into two main categories: situational logic and infrastructure.

Infrastructure

Let's describe the infrastructure part first. Sensor devices are fragile, they can be discharged or just lose packages because of network issues or bad architecture. That is why we need to have the ability to monitor how often, how many, and what data we receive from each sensor.

With this kind of data, we can detect which sensor is broken, discharged or has lost data. It is also helpful to have alerts that will inform specialists to solve the issue. With these analytics, you can also estimate which infrastructure you really need to handle this amount of data (this is critical for huge systems with thousands of sensors).

Situational logic

Having good situational logic will guide you to monitor the things that are important to your situation. For example, if you're monitoring data from a smart home, it makes sense to calculate this data only for the time you are at home (if nobody is at home it makes no sense to cool or heat air to an appropriate temperature).

It also makes sense to calculate values for specific times, like nighttime where it's ok to keep the home temperature cooler.

It also makes sense to create an alert on some metric values. For example, the optimal humidity minimum is 40% and the maximum is about 70%. If you measure a humidity value higher than 70%, your dashboard should let you know.

You must decide how many devices you want to monitor and which data you want to extract in order to build an optimal dashboard. In our smart home example, it's not so important to have a history of temperature and humidity.

Here we mostly care about the current temperature or at least the current temperature throughout the day. In another case, IoT for the agriculture sector might require a dashboard for historical trends.

The metrics we will monitor

In our example we will monitor the following metrics:

- humidity_gauge_percent
- pressure_gauge_pa
- temperature_gauge_c

It is often a good idea to capture the units of the metric in the metric name. You can see this practice exhibited in our metric names above. See Prometheus documentation on [naming metrics](#) for more information.

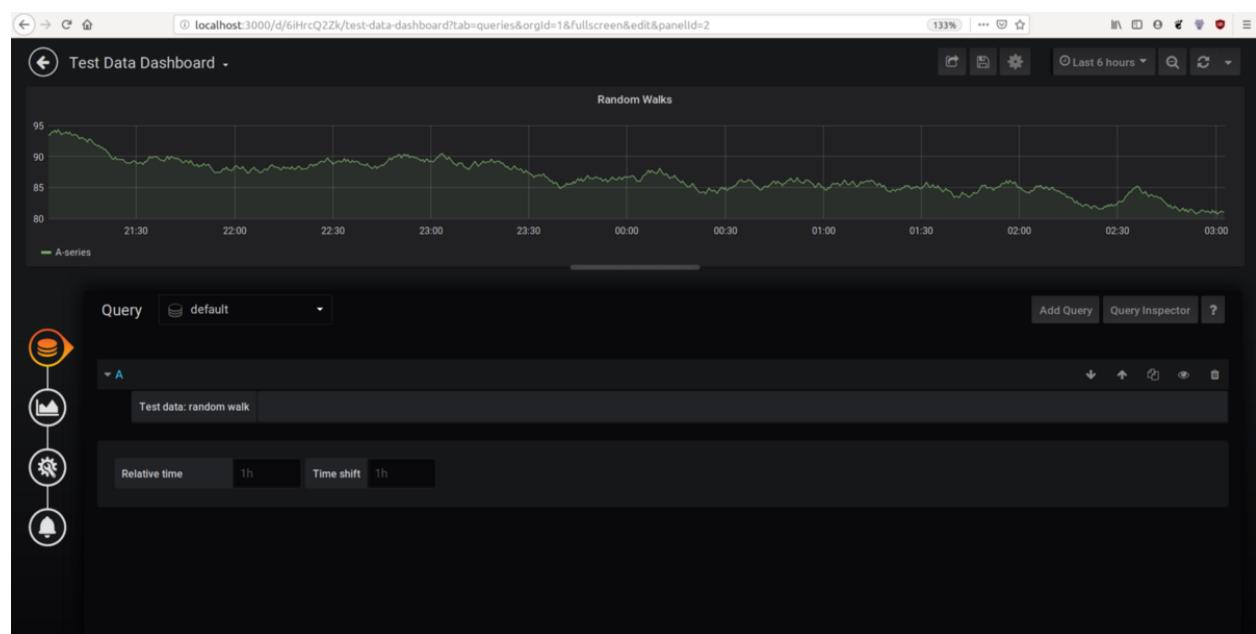
To get these metrics into Prometheus, you will need to set up your Prometheus to scrape the API endpoint visualized in the above diagram. To see one method for doing this, check out this tutorial [here](#).

For all data querying and manipulation, we will use [PromQL](#). Some basics will be explained here, but we recommend you get a good understanding of PromQL by reading [this article](#).

So, let's start with building the dashboard:

Building IoT dashboards

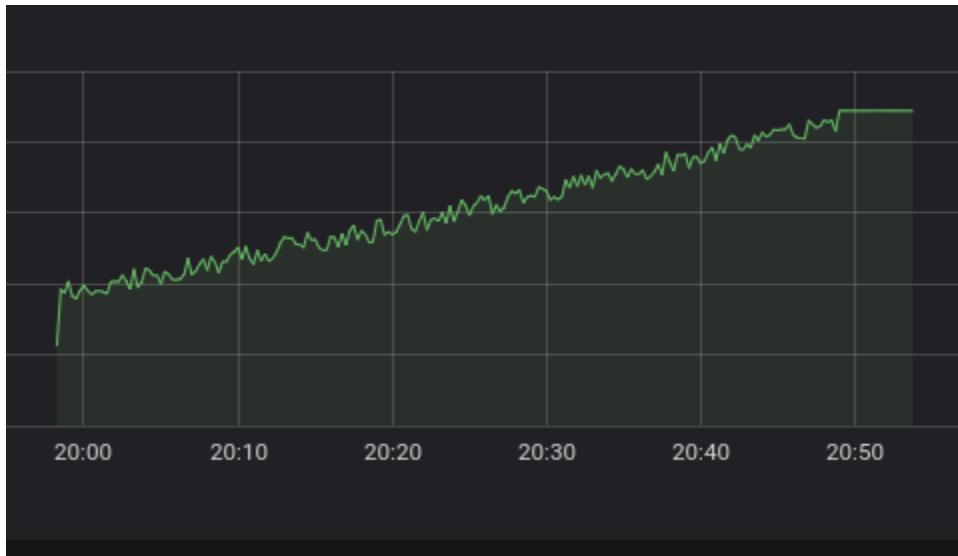
To start building your dashboard, go to the plus sign on the left hand side of the home page, and add a new dashboard. Then add a new panel and add a query. You can see the beginning of a new dashboard in the image below:



For more instructions on how to build a Grafana dashboard from scratch, read our full tutorial [here](#).

For our IoT dashboard query, we want to query our Prometheus database, so we will query that database using the metric names listed above.

Start building the query for 'humidity_gauge_percent' by querying just the metric name. Depending on the conditions and sensors, it's possible to have noise in the data and get results like this when we do just a basic PromQL query of 'humidity_gauge_percent':



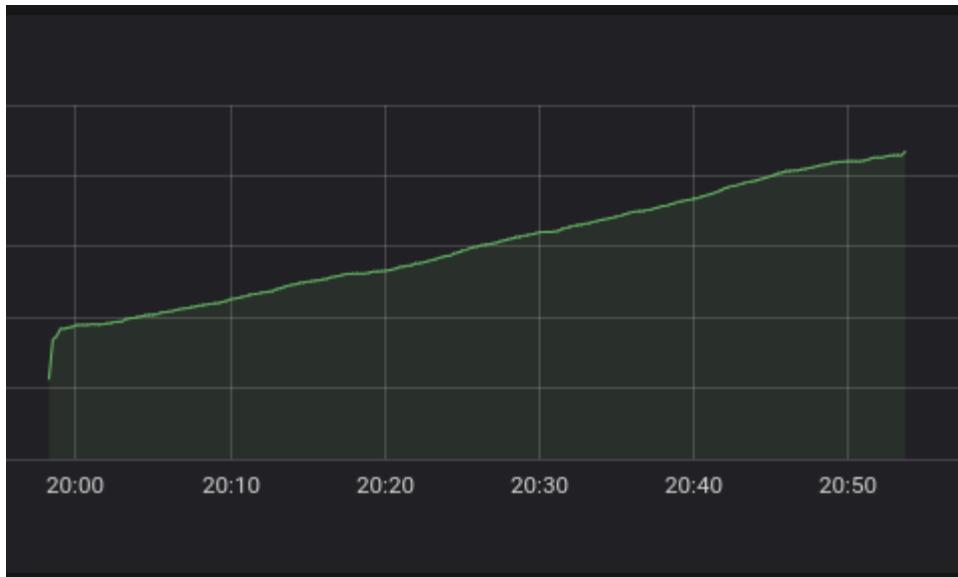
If you want to remove the noise, we can apply the following PromQL function when we query the metric. Just write the following query into your query expression field:

```
avg_over_time(humidity_gauge_percent[5m])
```

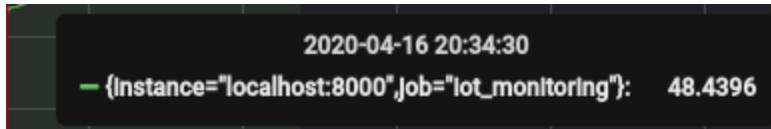
This PromQL expression transforms the query in two ways.

1. The first one is '[5m]' which just splits our data vector into a range vector. For each point in time, the range vector holds an array of 5m of values. (As you probably know, we can't display just humidity_gauge_percent[5m] because this is not a series with a timestamp - value, but grouped subseries.)
2. Then, we take the average over time.

As a result, we will get the following:



The above graph is much smoother and the data is easier to apply. If you do not specify the legend for a query, you will see an automatically generated legend, similar to the one below:



You can also find the exact same information by querying directly in Prometheus. If you go to your [Prometheus Expression Browser](#), then go to the Prometheus Status > Targets page in the Labels column, you will see the same metric:

iot_monitoring (1/1 up) show less				
Endpoint	State	Labels	Last Scrape	Scrape Duration
http://localhost:8000/metrics	UP	instance="localhost:8000" job="iot_monitoring"	6.301s ago	2.023ms

This is notable because it shows you which labels are associated with that metric.

Filtering, aggregating, and using functions on queries with PromQL

Filtering queries

It is possible to filter your metrics by the labels, such as the ones seen in the image at the end of the previous section. To query a specific instance, all you need to do is add curly brackets after the metric's name and specify the key-value pair you are looking for, separated with an equal sign. For example:

```
humidity_gauge_percent{instance="localhost:8000"}
```

Remember that you can use variables to determine which instances you want to visualize at the moment. For more information, read the [documentation](#). Variables can be custom or already predefined. This is a very powerful instrument you can use for building an aggregated dashboard for each room or region. But, in our case, we have only 1 instance per sensor, so we won't be using this application now.

Using functions on queries

If you want, you can also apply a [simple linear regression model](#) to predict the value of the future.

```
predict_linear(humidity_gauge_percent[10m], 300)
```

In this example, we will predict the value for 300 seconds in the future by 10 minutes range. *predict_linear* should only be used with gauges. Ensure your dashboard works with now+(prediction interval) for an upper boundary, otherwise you will miss the prediction. You should also know that the boundaries are installed for the whole dashboard, and not each panel separately.

Another great function is the **changes** function:

```
changes(humidity_gauge_percent[1m])
```

This function will show you the number of changes per specified time range (in our case: 1 minute). What can we get from this? If the data is not changed for a long period of time, we know the likelihood of a problem is high.

Also, it's recommended to collect some metrics related to network information, such as sensor status information, like power charge. These will help you detect sensors that are in the danger zone to become broken.

Aggregating queries

In many cases, you need data for each instance, as well as aggregated data. Here are some aggregation operators for PromQL:

- sum - calculate sum over dimensions
- min - select minimum over dimensions
- max - select maximum over dimensions
- avg - calculate the average over dimensions
- stddev - calculate population standard deviation over dimensions
- stdvar - calculate population standard variance over dimensions
- count - count number of elements in the vector
- count_values - count number of elements with the same value
- bottomk - smallest k elements by sample value
- topk - largest k elements by sample value
- quantile - calculate ϕ -quantile ($0 \leq \phi \leq 1$) over dimensions

For example, with bottomk you can select a series from the coldest rooms in your house (if you have more than one thermometer sensor). The syntax can be found in the official documentation, but here is an example of usage:

```
bottomk (2, temperature_gauge_c)
```

That will return 2 series with the lowest temperature. You can also group aggregation by labels with keywords ‘without’ or ‘by’.

NOTE: aggregation operators are not the same as the query functions you saw before. These operators aggregate data per dimension (where a dimension is a label). On the other hand, query functions operate on data by time ranges, such as every 5 minutes.

Setting thresholds

The next step is to set the thresholds for the chart. Go to the Visualization tab and configure:

Threshold	Operator	Value	Color	Fill	Line	Y-Axis
T1	gt	80	warning	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	left
T2	lt	40	critical	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	left

+ Add Threshold

+ Add time region ?

You can set thresholds, where the colour on the gauge will change if the value goes over a certain limit. It's also great to have configured [alert rules](#) to react as fast as possible.

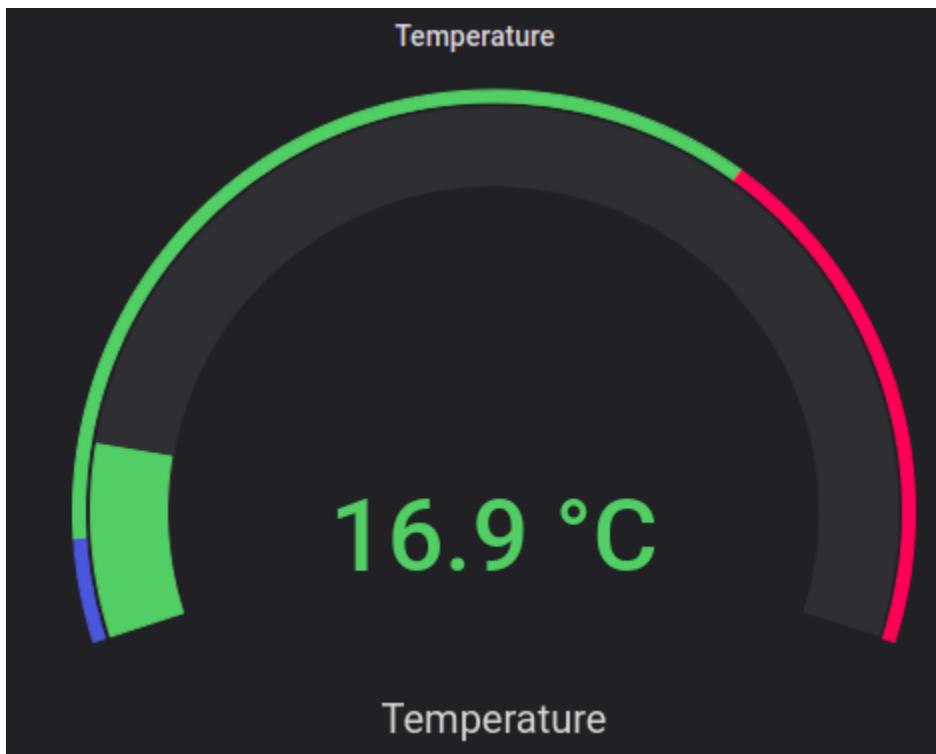
We can use the following query to detect if the instance is healthy:

```
up{job="<job-name>", instance="<instance-id>"}
```

Using this query we will see if the instance is healthy, i.e. reachable, or 0 if the scrape failed.

Appropriate visualization types

Do not forget to use the appropriate visualization type for your data and needs. For example, the graph is better for multiple values, trend detection, and comparison. Gauge is great to show some aggregated data, current values, and to visualize where the current value is in relation to the thresholds.



Conclusion

Sensors produce a high amount of useful information, and to process it we need to have dashboards. The specifications for the IoT are that we often have multiple instances of some devices, so we need to apply certain variables to be able to monitor each group and instance separately, while getting as many as possible.

Sensors are very fragile, so we should carefully monitor the sensors themselves, and not just the situational metrics. It is important to not only detect the problem, but also react as fast as possible. That is why we should also configure an alerting system to be always informed about the system status.

As you can see, setting up a system requires significant work. To reduce the configuration time, use the MetricFire [free trial](#) to run Prometheus and Grafana. You can also [book a demo](#) and talk to us directly about how to best set up your IoT dashboards.

- <https://gabrieltanner.org/blog/grafana-sensor-visualization>

How to visualize Sensor data using Grafana and InfluxDB



Grafana is a data analytics and visualization tool that helps you better understand your data by displaying it in various beautiful graphs and diagrams inside of a dashboard. It furthermore enables you to set up alerts if your metrics reach a particular threshold.

In this article, you will build a Grafana dashboard to visualize the data of a temperature sensor that will be read using a microcontroller and send over MQTT.

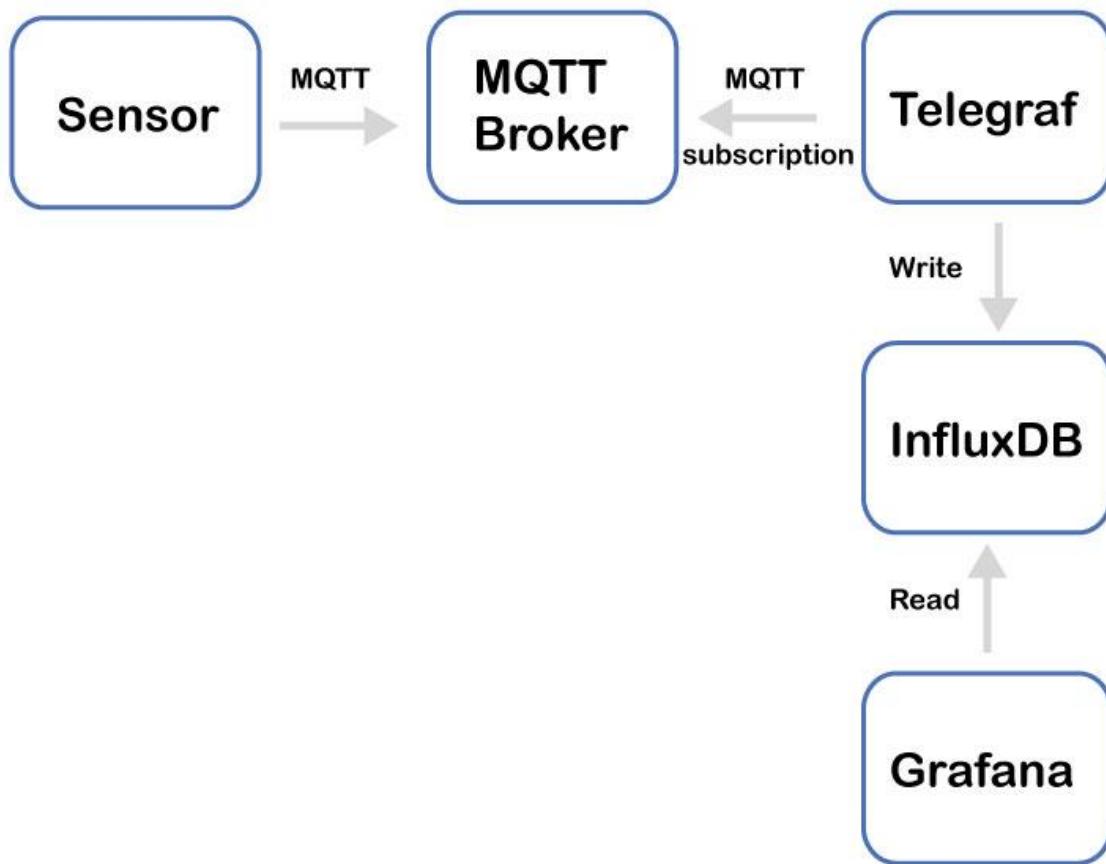
Prerequisites

Before you begin this guide, you'll need the following:

- Docker - Used for installing and using MQTT, Telegraf, InfluxDB and Grafana
- ESP8266 Microcomputer for reading a temperature sensor and sending the data over to your server. (You can use a Golang script to send fake data if you don't have the required hardware. This script will also be included in the article.)

Technology Stack

Now that you have an overview of what you are going to build and which tools will be used, it is essential to know how the components work together and exchange data.



Let's break the different components down:

- The ESP8266 reads the data from the temperature sensor and sends it to an MQTT broker.
- Telegraf subscribes to the MQTT topic and saves the received data into the InfluxDB database.
- Grafana reads the data from the InfluxDB database and visualizes it on a custom dashboard.

Step 1 - Running a Mosquitto MQTT server

In this section, you will install Mosquitto on your server and exposing its ports to the host machine using Docker and validate your installation by listing your containers.

First, you will install Mosquitto by running the official `eclipse-mosquitto` image and exposing the two working ports.

```
docker run -it -p 1883:1883 -p 9001:9001 -d eclipse-mosquitto
```

The `-p` flag is used to define the ports that you want to expose on the host. The `-d` flag on the other hand defines that you want to run the container in detached mode.

The installation can be validated by listing all containers and looking at the status field.

```
docker ps
```



```
# Output
```

CONTAINER ID STATUS	IMAGE PORTS	COMMAND	CREATED NAMES
a46c878e2943 Up 5 seconds	eclipse-mosquitto 0.0.0.0:1883->1883/tcp, 0.0.0.0:9001->9001/tcp	/docker-entrypoint...."	6 seconds ago fervent_borg

Step 2 - Installing and running InfluxDB

Now that you have the Mosquitto MQTT Broker up and running, you will continue by installing InfluxDB and creating a new database.

InfluxDB also provides an official Docker image that makes it easy to deploy their service.

```
docker run -d -p 8086:8086 -v influxdb:/var/lib/influxdb --name influxdb influxdb
```

The `-v` flag defines a volume that will save the database's data even if the container is restarted or destroyed.

After the installation is done, it is time to create the database that will be used to save the data. For that, you will need to use the InfluxDB CLI which you can open using the following command.

```
docker exec -it influxdb influx
```

You should now be in the InfluxDB CLI and can continue by creating the database and the telegraf user.

```
CREATE DATABASE sensors  
  
CREATE USER telegraf WITH PASSWORD 'telegraf'  
  
GRANT ALL ON sensors TO telegraf
```

Here you first create a database named sensors and then create a user called telegraf and grant him full access to the database.

Step 3 - Configuring and running Telegraf

Let's continue by installing and configuring Telegraf, a metric collection, and processing agent that collects the data sent to the MQTT broker and stores it in your InfluxDB database.

The first step of the configuration process is to generate the default configuration by starting a Telegraf docker container and copying the config to the host filesystem.

```
docker run --rm telegraf telegraf config > telegraf.conf
```

You should now have a *telegraf.conf* file in your directory. Here you will have to modify some fields under the *inputs.mqtt_consumer* tag, which defines the information of your MQTT broker.

```
[[inputs.mqtt_consumer]]  
  
## MQTT broker URLs to be used. The format should be scheme://host:port,  
  
## schema can be tcp, ssl, or ws.  
  
servers = ["tcp://10.0.0.22:1883"]
```

```
## Topics that will be subscribed to.

topics = [
    "sensors"
]

data_format = "influx"
```

The output will be configured under the outputs.influxdb tag which defines the following parameters:

- The URL of the InfluxDB database
- The database name
- Authentication parameters (username, password)

```
[[outputs.influxdb]]

## Multiple URLs can be specified for a single cluster, only ONE of the

## urls will be written to each interval.

urls = ["http://10.0.0.8:8086"]

## The target database for metrics; will be created as needed.

## For UDP url endpoint database needs to be configured on server side.

database = "sensors"

## If true, no CREATE DATABASE queries will be sent. Set to true when using
```

```
## Telegraf with a user without permissions to create databases or when the  
## database already exists.  
  
skip_database_creation = true  
  
  
## HTTP Basic Auth  
  
username = "telegraf"  
  
password = "telegraf"
```

After the configuration is finished Telegraf can be started using Docker. The configuration will be transferred into the container using a volume.

```
docker run -d -v $PWD/telegraf.conf:/etc/telegraf:ro telegraf
```

Step 4 - Sending data using Golang

This section will show you how you can send data to Telegraf using MQTT in Golang. This can be done using the following steps:

- Connecting to the MQTT broker using the paho.mqtt library
- Checking if the connection was successfully established
- Continuously sending data to the *sensors* topic

This section is meant for the people who don't have a microcontroller and temperature sensor and still want to follow the article for the Grafana part. If

you want to send the real sensor data using your ESP8266 or another microcontroller, you can skip this section.

First, you will need to create a Golang file using the following command:

```
touch mqtt.go
```

This command will create a file named *mqtt.go* in your current directory.

Now open the file in your favorite code editor to continue.

Next, you will install the needed dependency for working with MQTT in Golang.

```
github.com/eclipse/paho.mqtt.golang
```

Now that the file is created and the dependencies are successfully downloaded, it is time to connect to the MQTT broker.

Connecting can be done using the `mqtt.NewClient()` function and passing your MQTT client options as an argument. These options consist of the MQTT broker URL, username, password and client id.

After that, you need to check if the connection has been established successfully by checking if the received error is nil.

```
package main
```

```
import (
    "fmt"
    "log"
    "math/rand"
    "net/url"
    "strconv"
    "time"

    mqtt "github.com/eclipse/paho.mqtt.golang"
)

func connect(clientId string, uri *url.URL) mqtt.Client {
    opts := createClientOptions(clientId, uri)
    client := mqtt.NewClient(opts)
    token := client.Connect()
    for !token.WaitTimeout(3 * time.Second) {
    }
    if err := token.Error(); err != nil {
        log.Fatal(err)
    }
    return client
}
```

```
}

func createClientOptions(clientId string, uri *url.URL) *mqtt.ClientOptions {
    opts := mqtt.NewClientOptions()

    opts.AddBroker(fmt.Sprintf("tcp://%s", uri.Host))

    opts.SetUsername("mqtt")

    password := ""

    opts.SetPassword(password)

    opts.SetClientID(clientId)

    return opts
}

func main() {
    uri, err := url.Parse("tcp://10.0.0.22:1883")

    if err != nil {
        log.Fatal(err)
    }

    client := connect("pub", uri)
}
```

Once that is done you can start sending data to the MQTT broker using the *Publish()* function. Here we send a random value between 10 and 100 every second to the sensors topic we defined earlier.

```
func main() {

    uri, err := url.Parse("tcp://10.0.0.22:1883")

    if err != nil {

        log.Fatal(err)

    }

}

client := connect("pub", uri)

timer := time.NewTicker(1 * time.Second)

for t := range timer.C {

    fmt.Println(t)

    var min int64 = 10

    var max int64 = 100

    var random int64 = (rand.Int63n(max-min) + min)

    nsec := time.Now().UnixNano()

    payload := "weather,location=us-midwest temperature=" +
    strconv.FormatInt(random, 10) + " " + strconv.FormatInt(nsec, 10)

    client.Publish("sensors", 0, false, payload)

}
```

```
}
```

These configuration blocks will result in the following file:

```
package main

import (
    "fmt"
    "log"
    "math/rand"
    "net/url"
    "strconv"
    "time"

    mqtt "github.com/eclipse/paho.mqtt.golang"
)

func connect(clientId string, uri *url.URL) mqtt.Client {
    opts := createClientOptions(clientId, uri)
    client := mqtt.NewClient(opts)
    token := client.Connect()

    for !token.WaitTimeout(3 * time.Second) {
}
```

```
if err := token.Error(); err != nil {
    log.Fatal(err)
}

return client
}

func createClientOptions(clientId string, uri *url.URL) *mqtt.ClientOptions {
    opts := mqtt.NewClientOptions()
    opts.AddBroker(fmt.Sprintf("tcp:///%s", uri.Host))
    opts.SetUsername("mqtt")
    password := ""
    opts.SetPassword(password)
    opts.SetClientID(clientId)
    return opts
}

func main() {
    uri, err := url.Parse("tcp://10.0.0.22:1883")
    if err != nil {
        log.Fatal(err)
    }
}
```

```

client := connect("pub", uri)

timer := time.NewTicker(1 * time.Second)

for t := range timer.C {

    fmt.Println(t)

    var min int64 = 10

    var max int64 = 100

    var random int64 = (rand.Int63n(max-min) + min)

    nsec := time.Now().UnixNano()

    payload := "weather,location=us-midwest temperature=" +
strconv.FormatInt(random, 10) + " " + strconv.FormatInt(nsec, 10)

    client.Publish("sensors", 0, false, payload)

}

}

```

Once you're done, save and exit your file. After that you can run the script using the following command:

```
go run mqtt.go
```

Sending data using an ESP8266 MQTT client

Now that you have successfully set up your Mosquitto MQTT broker, InfluxDB and Telegraf it is time to send your sensor data. This section will

show you how to do precisely that using an ESP8266 microcontroller and a TMP006 temperature sensor.

Note: The code may be different for other microcontrollers and temperature sensors.

The following steps need to be executed to successfully send data over MQTT:

- Connect your microcontroller to your Wifi
- Connect to the MQTT broker
- Read your sensor data
- Send the data over MQTT

First you will have to install the following libraries. If you are using the Arduino IDE you can reference the [official Arduino guide](#). If you are using the [PlatformIO](#) VS Code extension instead you can download them using the library manager.

```
#include <Arduino.h>

#include <Wire.h>

#include <SPI.h>

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

#include <Adafruit_Sensor.h>
```

```
#include "Adafruit_TMP006.h"
```

Next, add your WIFI credentials at the top and use them to connect to the internet using the WiFi library.

```
// Update these with values suitable for your network.

const char* ssid = "ssid";

const char* password = "password";

const char* mqtt_server = "server_ip";

// Connecting to the WIFI network

void setup_wifi() {

    delay(10);

    Serial.println();

    Serial.print("Connecting to ");

    Serial.println(ssid);

    WiFi.mode(WIFI_STA);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {

        delay(500);
```

```
Serial.print(".");
}

randomSeed(micros());

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void setup() {
    Serial.begin(9600);
    setup_wifi();
}

void loop() {
}
```

After you have filled in your credentials, it is time to continue by connecting to the MQTT broker. You will also implement a function to reconnect to the broker if the connection fails.

```
WiFiClient espClient;

PubSubClient client(espClient);

void reconnect() {

    // Loop until we're reconnected

    while (!client.connected()) {

        Serial.print("Attempting MQTT connection...");

        // Create a random client ID

        String clientId = "ESP8266Client-";

        clientId += String(random(0xffff), HEX);

        // Attempt to connect

        if (client.connect(clientId.c_str(), "mqtt", "")) {

            Serial.println("connected");

            // Once connected, publish an announcement...

            client.publish("outTopic", "hello world");

            // ... and resubscribe

            client.subscribe("inTopic");

        } else {

    }
```

```
Serial.print("failed, rc=");

Serial.print(client.state());

Serial.println(" try again in 5 seconds");

// Wait 5 seconds before retrying

delay(5000);

}

}

}

void setup() {

Serial.begin(9600);

setup_wifi();

client.setServer(mqtt_server, 1883);

}

void loop() {

// Connect to the mqtt client

if (!client.connected()) {

reconnect();

}
```

```
client.loop();  
}  
}
```

Next, you will read the sensor data using the Adafruit_TMP006 library and send it over MQTT on the sensors topic.

```
// Create variable to hold mqtt messages  
  
#define MSG_BUFFER_SIZE  (100)  
  
char msg[MSG_BUFFER_SIZE];  
  
  
  
void setup() {  
  
    Serial.begin(9600);  
  
  
  
    setup_wifi();  
  
  
  
    client.setServer(mqtt_server, 1883);  
  
  
  
    if (! tmp006.begin()) {  
  
        Serial.println("No sensor found");  
  
        while (1);  
  
    }  
}
```

```
void loop() {  
  
    // Connect to the mqtt client  
  
    if (!client.connected()) {  
  
        reconnect();  
  
    }  
  
    client.loop();  
  
  
    // Get the current object temperatur of the sensor  
  
    float objt = tmp006.readObjTempC();  
  
  
  
    // Create the message that will be send using mqtt  
  
    String message = String("weather,location=us temperature="+String(objt));  
  
    message.toCharArray(msg, message.length());  
  
    Serial.println(msg);  
  
  
  
    // Send the message on the sensors topic  
  
    client.publish("sensors", msg);  
  
  
  
    delay(1000);  
  
}
```

The sensor needs to be initialized using the *begin()* function and can then be read using the *readObjTempC()* function.

After that, you can format the temperature data into a String that is accepted by the InfluxDB database. Then you can send it over MQTT using the *publish()* function on the MQTT client you created above.

These configuration blocks will result in the final following file:

```
#include <Arduino.h>

#include <Wire.h>

#include <SPI.h>

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

#include <Adafruit_Sensor.h>

#include "Adafruit_TMP006.h"

Adafruit_TMP006 tmp006;

// Update these with values suitable for your network.

const char* ssid = "ssid";

const char* password = "password";

const char* mqtt_server = "server_ip";
```

```
WiFiClient espClient;

PubSubClient client(espClient);

// Create variable to hold mqtt messages

#define MSG_BUFFER_SIZE (100)

char msg[MSG_BUFFER_SIZE];

// Connecting to the WIFI network

void setup_wifi() {

    delay(10);

    Serial.println();

    Serial.print("Connecting to ");

    Serial.println(ssid);

    WiFi.mode(WIFI_STA);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {

        delay(500);

        Serial.print(".");
    }
}
```

```
}

randomSeed(micros());

Serial.println("");

Serial.println("WiFi connected");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

}

void reconnect() {

    // Loop until we're reconnected

    while (!client.connected()) {

        Serial.print("Attempting MQTT connection...");

        // Create a random client ID

        String clientId = "ESP8266Client-";

        clientId += String(random(0xffff), HEX);

        // Attempt to connect

        if (client.connect(clientId.c_str(), "mqtt", "")) {

            Serial.println("connected");

            // Once connected, publish an announcement...

            client.publish("outTopic", "hello world");
        }
    }
}
```

```
// ... and resubscribe

client.subscribe("inTopic");

} else {

    Serial.print("failed, rc=");
    Serial.print(client.state());
    Serial.println(" try again in 5 seconds");

    // Wait 5 seconds before retrying
    delay(5000);
}

}

}

void setup() {

Serial.begin(9600);

setup_wifi();

client.setServer(mqtt_server, 1883);

if (!tmp006.begin()) {

    Serial.println("No sensor found");

    while (1);
}
```

```
}

}

void loop() {

    // Connect to the mqtt client

    if (!client.connected()) {

        reconnect();

    }

    client.loop();


    // Get the current object temperatur of the sensor

    float objt = tmp006.readObjTempC();


    // Create the message that will be send using mqtt

    String message = String("weather,location=us temperature="+String(objt));

    message.toCharArray(msg, message.length());

    Serial.println(msg);


    // Send the message on the sensors topic

    client.publish("sensors", msg);

    delay(1000);
```

{}

You can now run the program by uploading it onto your microcontroller.

Step 5 - Running and configuring Grafana

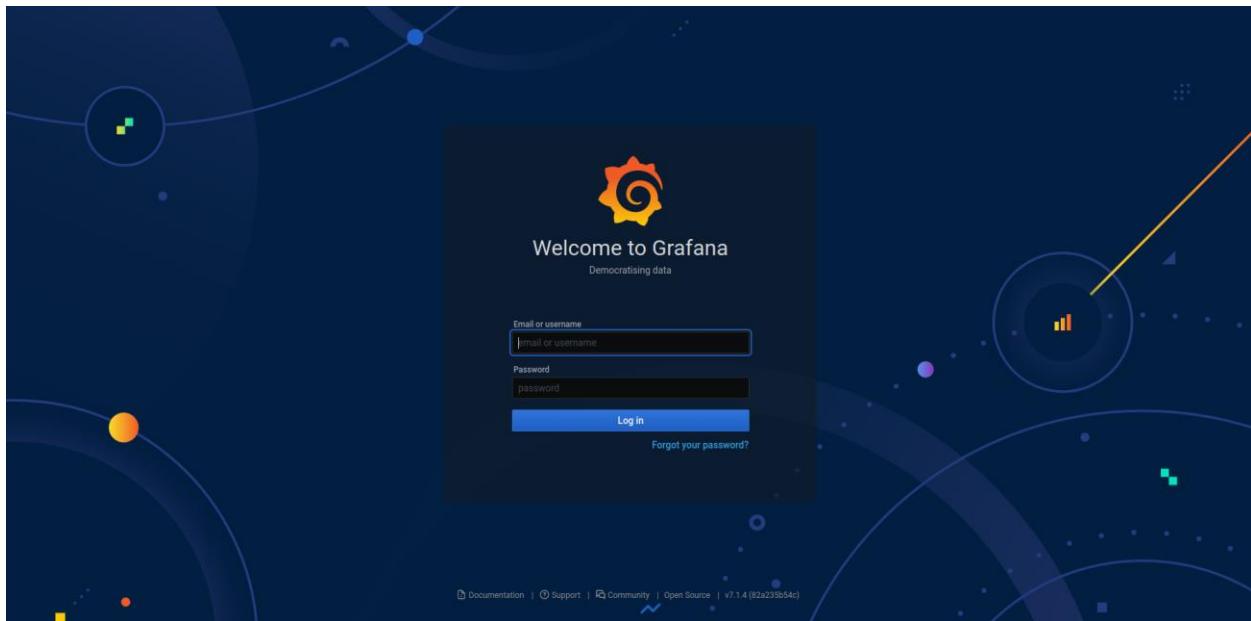
Grafana is an analytics platform and one of the most popular data visualizers out there. Grafana makes it easy to build fully customizable dashboards using data from a wide variety of data sources.

In this section, you will start Grafana using Docker and configure InfluxDB as your data source. Then you will configure your Grafana dashboard by fetching your sensor data from the InfluxDB database.

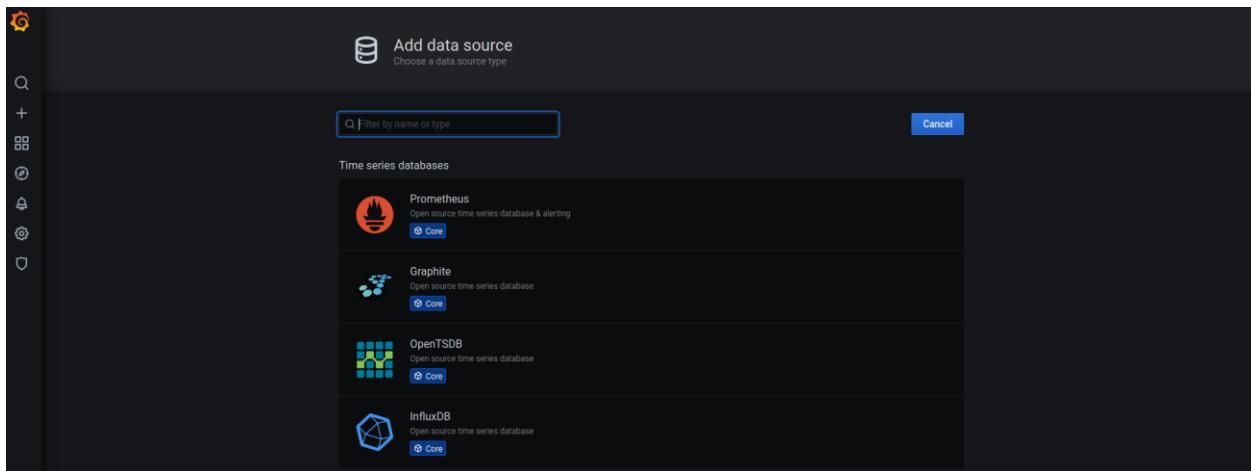
First, you will start the Grafana docker container and publish port 3000 to your host system.

```
docker run -d -p 3000:3000 grafana/grafana
```

Now you should be able to visit Grafana on your localhost:3000 and see the following screen. Continue by login in with the standard credentials *admin* *admin* and feel free to change your password.

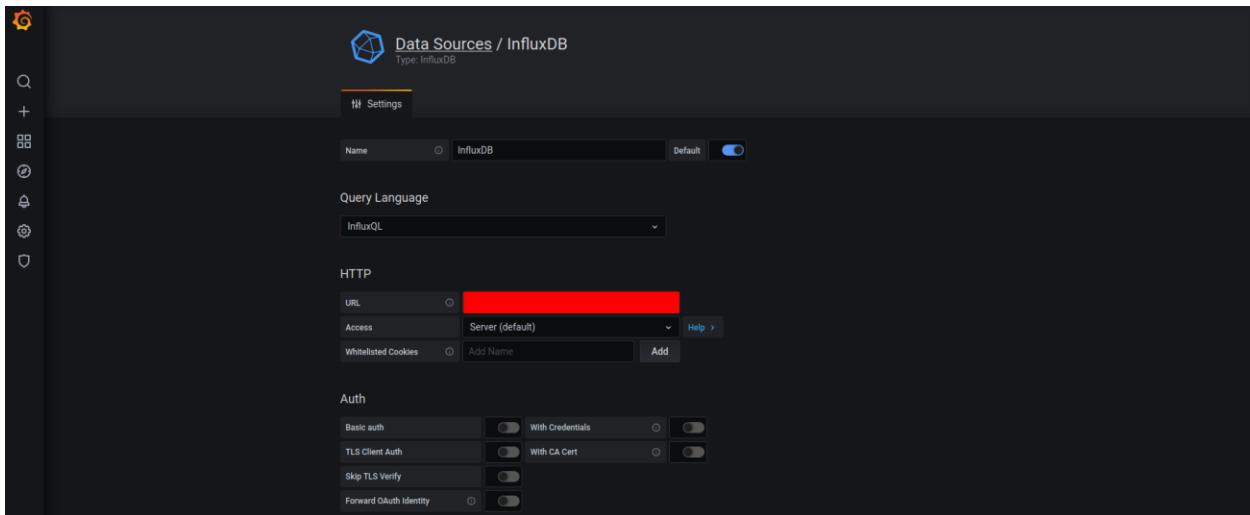


Next, navigate to the add data source page by clicking on **Configuration > Data Sources** in the side menu.



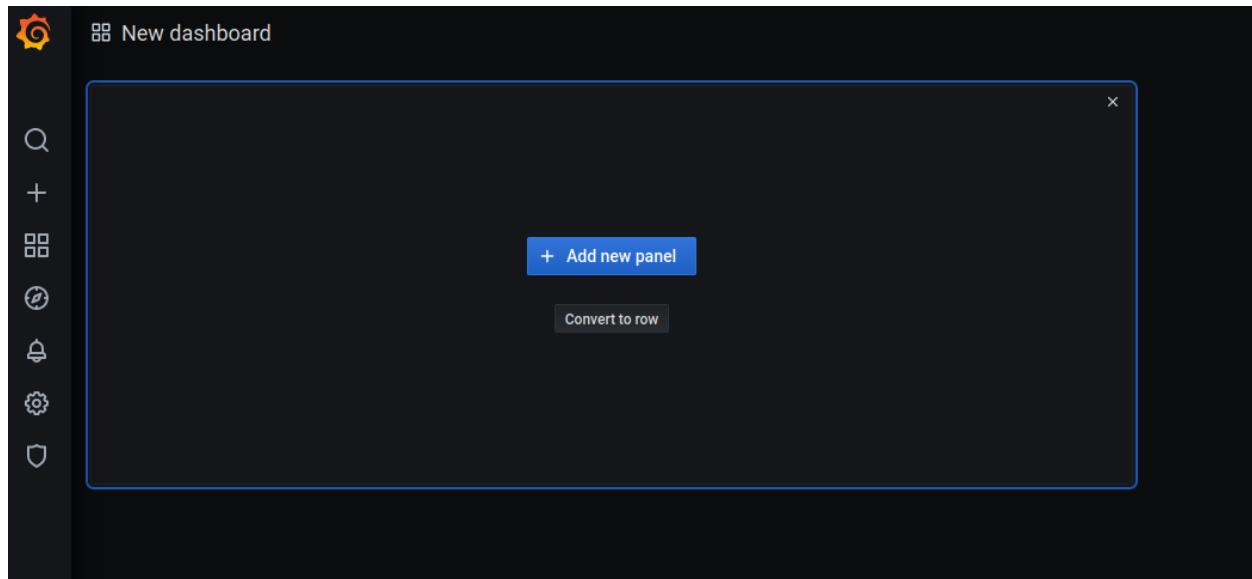
Continue by selecting InfluxDB from the time series databases drop-down menu. You now need to configure the data source by filling in the URL and database fields based on the settings you configured earlier.

The URL is a combination of the IP address and port of your InfluxDB API and the database is the database name you set earlier (Sensors if you followed the article).



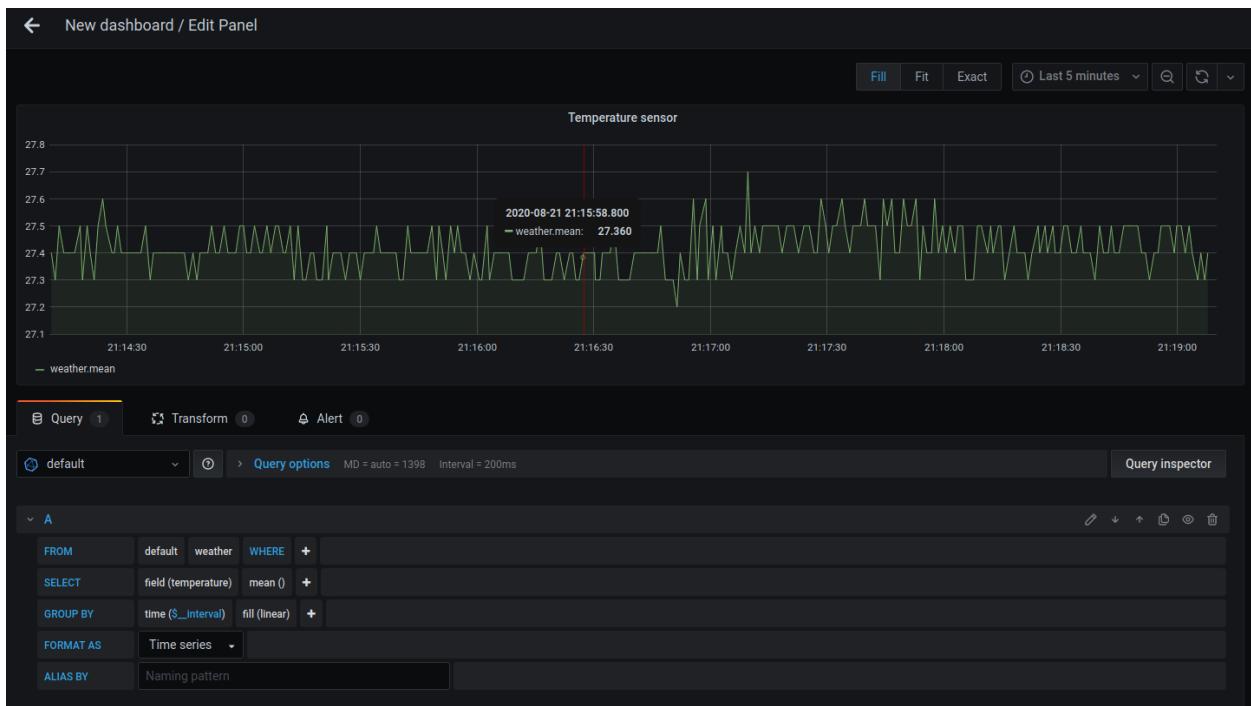
After successfully configuring and saving the data source you can continue by creating a new dashboard. Follow the following steps for that:

1. Click **New dashboard**.
2. Click **Add new panel**. Grafana creates a basic graph panel that needs to be configured using your InfluxDB data.

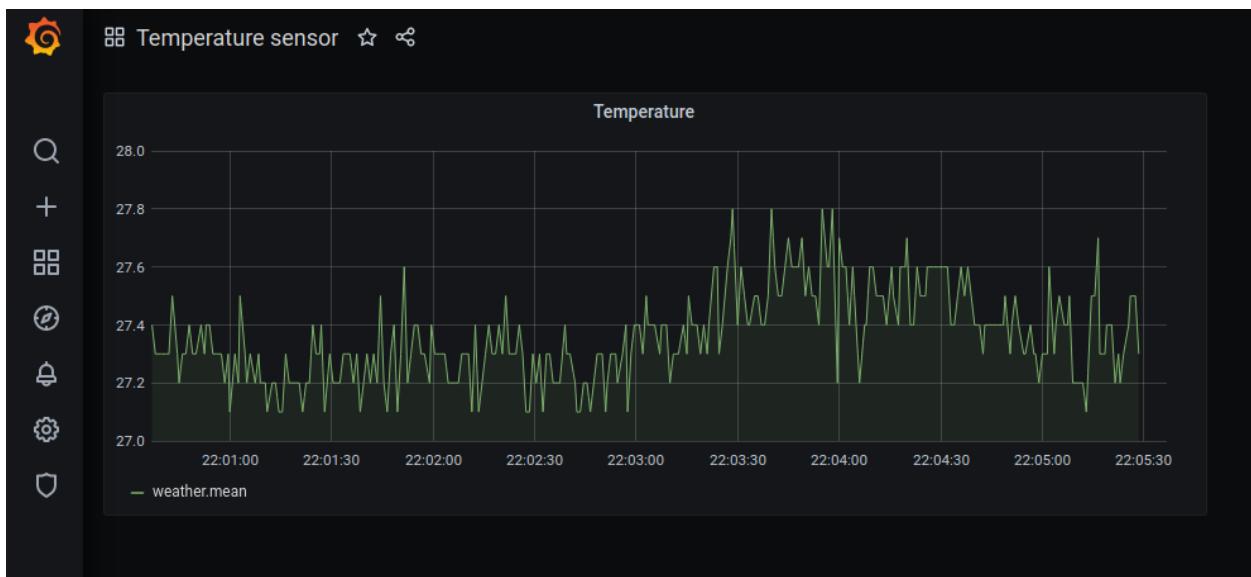


Now it is time to configure your panel using the data of your InfluxDB database. Here you need to select *weather* as the measurement you want to use for the from field. Then you need to select temperature as the field.

For more information about the query selector visit the [official documentation](#).



After saving your dashboard by clicking the **Save dashboard** button in the top corner of your screen, the dashboard should look similar to this:



If you want more information about how to customize your Grafana dashboard and add different configurations, check out the [official documentation](#).

Step 6 - Running the deployment using Docker Compose

Another way you can run this whole setup is through Docker compose, which makes the entire process a lot easier and more reusable.

In this section, you will learn how you can put all containers in a single Docker compose file and start it using a single command. If you don't know what Docker compose is or how it works I recommend checking out [this guide](#).

Let's start with the basic structure of the file and the mosquitto container consists of an image and publishes two ports to the host.

```
version: '3.1'

services:
  mosquitto:
    image: eclipse-mosquitto
    hostname: mosquitto
    container_name: mosquitto
```

```
ports:  
  - "1883:1883"  
  - "9001:9001"
```

The Telegraf container needs a bind mount to load your configuration into the container, which can be specified using the *volumes* key.

```
telegraf:  
  image: telegraf:latest  
  container_name: telegraf  
  links:  
    - influxdb  
  volumes:  
    - ./telegraf.conf:/etc/telegraf/telegraf.conf:ro
```

The grafana container will have an normal volume instead which is defined using the *volumes* keyword at root level.

```
grafana:  
  image: grafana/grafana:latest  
  container_name: grafana  
  ports:  
    - "3000:3000"  
  links:
```

```
- influxdb

depends_on:
  - influxdb

volumes:
  - grafana-storage:/var/lib/grafana

volumes:
  grafana-storage:
```

InfluxDB also sets the same kind of volume to save its data but will also define the database, username and password as environment variables so you don't need to configure it manually as you did above.

```
influxdb:

  image: influxdb:latest

  container_name: influxdb

  ports:
    - "8083:8083"
    - "8086:8086"
    - "8090:8090"

  environment:
    - INFLUXDB_DB=sensors
    - INFLUXDB_USER=telegraf
```

```
- INFLUXDB_USER_PASSWORD=telegraf

volumes:

- influxdb-storage:/var/lib/influxdb


volumes:

influxdb-storage:
```

These configuration blocks will result in the following file:

```
version: '3.1'

services:

influxdb:

  image: influxdb:latest
  container_name: influxdb

  ports:
    - "8083:8083"
    - "8086:8086"
    - "8090:8090"

  environment:
    - INFLUXDB_DB=sensors
    - INFLUXDB_USER=telegraf
    - INFLUXDB_USER_PASSWORD=telegraf
```

```
volumes:  
- influxdb-storage:/var/lib/influxdb
```

```
telegraf:  
image: telegraf:latest  
container_name: telegraf  
links:  
- influxdb  
volumes:  
- ./telegraf.conf:/etc/telegraf/telegraf.conf:ro
```

```
grafana:  
image: grafana/grafana:latest  
container_name: grafana  
ports:  
- "3000:3000"  
links:  
- influxdb  
depends_on:  
- influxdb  
volumes:  
- grafana-storage:/var/lib/grafana
```

```
mosquitto:  
  
    image: eclipse-mosquitto  
  
    hostname: mosquitto  
  
    container_name: mosquitto  
  
    ports:  
  
        - "1883:1883"  
  
        - "9001:9001"  
  
volumes:  
  
    influxdb-storage:  
  
    grafana-storage:
```

Next, run the Docker compose file using the following command:

```
docker-compose up -d
```

You can now validate that the containers are running.

```
docker ps
```

You have now successfully stored your sensor data in an InfluxDB database and visualized it on a Grafana dashboard. You also learned how to build a Docker compose file for easier usage and better reusability.

Conclusion

In this article, you configured your own Grafana dashboard and sent data to it from your microcontroller using the MQTT communication protocol.

If you are interested in more articles like this one, I highly recommend joining my email list and following me on twitter, so you never miss a new article.

3/4. ให้จัดเรียบเรียงข้อมูล

ESP8266 / ESP32 & Mesh Network

- ESP Mesh Network 1 - <https://meetjoeblog.com/2018/03/25/esp8266-esp32-mesh-network-ep1/>

POSTED INTECHNOLOGY

ESP8266 / ESP32 & Mesh Network ตอนที่ 1: Introduction & Painlessmesh

IODES 25/03/2018 3 COMMENTS ESP8266 / ESP32 & MESH NETWORK ตอนที่ 1: INTRODUCTION & PAINLESSMESH

จ้าวหัวตอนที่ 1 แล้ว แสดงว่าต้องมีมากกว่า 1 ตอนแน่ แต่ไม่ต้องห่วงว่าจะนาน มีคนประมาณไว้ว่าให้เวลาตั้ง 6 เดือนแน่น และคงทำไม่ได้ ป้าคิดคิด ไหนๆ ก็ใหญ่ๆ แล้ว เจียนจะเดยรื่อง Mesh Network ซึ่งก็ไม่ใช่รื่องใหม่มะไหร่ ซึ่งรื่อง Mesh Network ที่จะเจียนนี้ก็จะแบ่งออกเป็น 4 ตอนด้วยกัน ไหนๆ เก็บแยกแล้วก็เล่นให้ฟูๆ ไปเล่นนะอ้อเจ้า บทความนี้เจียนอ่อนมันส์ด้วยๆ คือว่าทบทวนความรู้กัน ไม่ครามมา ใจๆ หักสิ้น กันอ่านการนิความรู้การใช้งาน Arduino IDE และบอร์ด MCU โดยเฉพาะ ESP8266 และ ESP32 มาบ้าง

ตอนที่ 1: Mesh Network กับ Introduction to Painlessmesh โดยใช้ ESP8266 / ESP32

Introduction

อะมาดูนิยามกันจะหน่อยเด็กก่อนว่า Mesh Network เป็นมันก็คือโครงสร้างของ Network แบบหนึ่งแหะะที่มีลักษณะเป็น Node ซึ่ง เชื่อมต่อถึงกันหมดและทำงานร่วมกันในการส่งผ่านข้อมูลจาก node นึงไปยังอีก node นึง ซึ่งการทำงานในการส่งผ่านข้อมูลตรงนี้แหะะที่ไม่ ขึ้นลงตรงต่อ กันแบบตายตัว นั่นหมายความว่าโครงสร้างของ mesh นั้นสามารถปรับเปลี่ยนได้ตลอดเวลา ถูกกันคลอด เช่นถ้าจะส่งข้อมูลจากจุด A ไปจุด B จะไปเส้นทางไหน ถ้าเส้นทางจาก A ->B->C->D แล้วถ้า Node B ล้ม จะ route ข้อมูลไปที่ Node ไหนต่อเพื่อให้ถึง ปลายทาง (Self-Organize / Self-Configure) ซึ่งมันจะดีมากๆ เลยในการติดตั้งๆ อยู่อุปกรณ์เข้าไปใน Mesh และมันก็ถูกกันเอง ได้ เจ๊งประหลาด

ซึ่ง Concept ของ Mesh Network คร่าวๆ ก็คือย่างที่อินบายไปข้างต้นในเนตมีให้อ่านเพียบทั้งไทยและเทศ ซึ่งที่ตอนนี้เราเห็นใจด้วย เรา กันมาก ก็จะเป็นอุปกรณ์ประเภท Home Automation ซึ่งเริ่มเข้ามาทำการตลาดในบ้านเรายะอะขึ้น ในตลาดก็จะมีมาตรฐานของการต่อสารผ่าน อุปกรณ์พวกนี้อยู่ 2 ค่ายใหญ่ๆ นั่นก็คือ Zigbee และ Z-Wave โดยคุณสมบัติของทั้งสองค่ายนี้เลยก็คือ Mesh Network และ Low power Consumption ละนั้นในการติดตั้งอุปกรณ์ไม่ว่าจะ Zigbee หรือ Z-Wave เข้ากับ Gateway นั้น สามารถทำได้เจ๊งมาก ไม่ต้อง config routing หรือเดินสายอะไรให้ยุ่งยาก แฉมยังขยายพื้นที่การใช้งานออกไปได้ไกลขึ้นตาม node ปลายทางที่เชื่อมถูกกันด้วย คราวนี้เรามาดูผู้ที่มีชื่อของตัว micro controller กันบ้างซึ่งในตลาดก็มีหลายค่าย ถ้าใครอยากรู้ mcu ที่มี z-wave ในตัว ใช้ Arduino IDE พัฒนาและรองรับการทำงานแบบ Mesh Network ของค่าย z-wave ได้ก็มี แต่วันนี้เราจะคุยกันถึงพระเอกตัวหลักของเรานั่นก็คือ ESP8266 กับ ESP32 จากค่าย Espressif

เจ้า ESP8266 / ESP32 ที่บานคนเรียกกว่าเป็น Arduino Killer นี้มาพร้อมคุณสมบัติ WiFi ในตัวซึ่งหมายความว่าที่จะมาทำอุปกรณ์ ประเภท iot โดยที่สามารถทำตัวเองเป็นได้ทั้ง Client/Server ที่มี Built-in WebServer ในตัวเป็น Access Point ก็ได้ สารพัดประโยชน์จะใช้งานจากการที่เป็น WiFi Enabled ซึ่งทาง Espressif ที่เป็นคุณพัฒนา ESP8266/ESP32 ก็เลือกเห็นเหมือน ก็หลายๆ ท่านเห็นแหะะว่า Mesh Network มันเจ๊งนะ ก็เลยพัฒนา ESP-Mesh Protocol ขึ้นมา

ซึ่งถ้าคุณจากเอกสารของ ESP8266 แล้วนี่ สามารถที่จะเชื่อมต่ออุปกรณ์กันได้ถึง 87 ตัวเลยที่เดียวแล้วลองคิดคุยกันในแต่ละ hop ของ Node อย่ากว่าแต่ 100 เมตรเลย แค่ 50 เมตรก็พอ อุปกรณ์ 87 ตัวนี่ครอบคลุมพื้นที่ใหญ่มากๆ ฟังคุ้ดีอีกแล้ว แต่ถ้าไปคุณล่วงหน่อง การใช้งาน ESP-Mesh บอกได้เลยว่ามีนี่ดีนั่น รวมถึงจากข้อมูลที่หามานี่คุณเองบัญหาค่อนข้างเยอะในการใช้งานโดยเฉพาะเรื่อง Self-Organize/Self-Configure จะนั้นเราจะเขียน ESP-Mesh Protocol กันไป

ใครอยากรู้เรื่อง ESP-Mesh Protocol อ่านได้จากที่ลิงค์นี้เลย [“ESP-Mesh Protocol by Espressif”](#)

จากการศึกษาเกี่ยวกับอ่างหนัก (จริงๆก็ไม่เท่าไหร่หรอก แค่ Google แล้วก็อ่านไปเรื่อยๆก็นั้นแหละ ยุคนี้ดีกว่าสมัย 2540 ที่ผมเรียน MCS51 อีก) เลยทำให้พบว่ามีคนแจกอัญหาจาก ESP-Mesh Protocol จริงๆและใช้งานล่อนข้างยาก อย่าไปเข้าครับลองเปิด pdf ดู มีมีอะครับ ก็เลขาอ่านมาเรื่อยๆมาพบกับ library ที่ชื่อว่า EasyMesh กับ Painlessmesh อ่านแล้วขอกระซิบใจชั้นขึ้นมาแล้วหั้งสองตัว ตัวนึงง่ายๆอีกตัวไม่ปวดหัว พอดีกับ EasyMesh กับ Painlessmesh นี่ทางคุณ BlackEdder (อันนี้ไม่รู้นามสมมติหรือชื่อจริงนะ) พัฒนาต่อเนื่องมาจาก EasyMesh ซึ่งหยุดการพัฒนาไปแล้ว ที่สำคัญ Painlessmesh ใช้ได้ทั้ง ESP8266 และ ESP32 อีกด้วย

ว่าแล้วนี่มาเริ่มกันเลยดีกว่า เกริน Intro จะเบolare ไปค้นกระปุองของเด่นที่พอยเลือกไม่ได้ต่อ กับเซนเซอร์อื่นๆ หรือทดลองงานอื่น ได้มา 17 ตัว เดียวเราจะเอา 17 ตัวนี้แหละมาลองทำ Mesh Network กัน ก็มี ESP8266 15 ตัว (NodeMCU 4 ตัว Wemos D1 6 ตัว ESP-01 อีก 5 ตัว) กับ ESP32 อีก 2 ตัว (WROOM32 กับ Heltec Lora)

เตรียมความพร้อม

ซึ่งอันนี้สมมติว่าคุณอ่านได้ติดตั้ง board esp32 และ esp8266 ไว้เรียบร้อยแล้วนะครับ ขั้นแรกก็ Download/Clone Library จาก Github มาไว้ที่เครื่องก่อน โดยเข้าไปที่ [Painlessmesh ที่ Github](#)

จากนั้นก็ติดตั้งลงใน Arduino IDE โดยเข้าไปที่ Sketch -> Include Library -> Add .zip library และเลือก zip file ที่เรา Download มา

ติดตั้ง Library Dependencies ซึ่งจำเป็นต่อการใช้ Painlessmesh

ก่อนการใช้งานเรายังต้องใช้ Library อีกสองตัวนั่นก็คือ ArduinoJson และ TaskScheduler ซึ่งการ Install Library ที่สามารถทำได้ไม่ยากเลยสำหรับสองตัวนี้ เข้าไปที่ Sketch -> Include Library -> Manage Libraries จากนั้นในช่อง Filter Your Search ก็สามารถได้ชื่อของ Library ทั้งสองตัวนี้ได้เลย

ArduinoJson

TaskScheduler

สิ่งที่ต้องรู้และข้อจำกัด

ทางผู้พัฒนา Painlessmesh เก้าเคลมเอาไว้ว่า library ชุดนี้นั้นเป็น True Adhoc Networking นั่นก็คือไม่ต้องมีการออกแบบ node ให้หนาเชื่อมต่อ node ไหน โครงสร้างของ mesh จะเป็นยังไง ไม่ต้องมี router ศูนย์กลาง ซึ่งแค่ 2 node ก็สามารถทำงานได้แล้ว ทำให้คนพัฒนาไม่ต้องมาสูงวุ่นวายในเรื่องของการจัดการกับ Self-Organize / Self-Configure ไปสนใจส่วนของ Business Logic / Operation Logic ว่าจะใช้งาน Mesh นือย่างไรดีกว่า

ซึ่งคำถามของคนใช้ที่มักจะถามบ่อยๆ Riley ได้ว่าเป็น FAQ ของ Painlessmess เลยก็ว่าได้นั้นก็คือ จำนวน Maximum Node ที่รองรับ โดยที่ทางทีมผู้พัฒนาเก็บอุปกรณ์ไว้ว่า

“The maximum size of the mesh is limited (we think) by the amount of memory in the heap that can be allocated to the sub-connections buffer and so should be really quite high.”

นั่นก็คือ ขึ้นอยู่กับ memory ของแต่ละ node นั่นเอง เพราะมันจะต้องใช้ในการเก็บสถานะของแต่ละ node เพื่อที่จะได้ทำ routing config ต่างๆ จนนั้นถ้าแต่ละ Node อยู่ค่าจาก Sensor ตามรอบเวลาที่กำหนด แล้วส่งผ่าน Mesh Network ไปเก็บไว้ที่ Server จำนวน memory ที่ใช้ไม่น่ามาก ซึ่งเดียวเราจะมาดูกันว่า memory นั้นจะถูกใช้ไปเท่าไหร่เมื่อเพิ่มนode เข้าไปใน Mesh Network

- non-ip networking: ในระบบ Network เราต้องคุ้นชินกับระบบ IP หรือชื่องานนี้อีกคือ mac address แต่ว่า Painlessmesh นั้นไม่ใช้tcp/ip ใน การสื่อสารและไม่ได้อ้างอิงโดยใช้ ip หรือ mac address แต่จะใช้ chipid ซึ่งก็เป็นหมายเลขเฉพาะในแต่ละชิปของ esp8266/esp32 โดยใน library จะอุปกรณ์ว่า nodeid แทน
- JSON Based: painlessmesh ใช้ระบบการส่งผ่านข้อมูลในรูปแบบของ JSON สำหรับไม่คุ้นเคยจะง่ายหน่อย แต่รับรองไม่หากแน่นอน ซึ่งข้อดีของมันก็คือ Node ที่รับข้อมูลแล้วต้องการส่งต่อไปยัง webserver หรือระบบงานอื่นที่รองรับ json อยู่แล้วก็จะง่าย多了ที่เดียว
- Delay: เริ่มเข้าใจเรื่องข้อจำกัดของ ข้อจำกัดแรกเลยก็คือเรื่องของ delay ซึ่งผู้พัฒนา Painlessmesh เก้าแนะนำให้หลีกเลี่ยง เพราะภายในตัว library เองจะมีการวนในครุภัณฑ์ node ต่างๆ เพื่ออัพเดตสถานะของ mesh ตลอดเวลา node ไหนอยู่ node ไหนหลุดไป สำหรับการใช้งาน delay ภายใน loop อาจทำให้ mesh ของไม่เสียหาย ซึ่งทางแก้ไขก็แนะนำให้ใช้ task scheduler แทน
- อัตราการส่งข้อมูล: ทางผู้พัฒนาเก้าแนะนำมาให้คิดคำนวณอัตราการส่งข้อมูลแบบ conservative อะหน่อย ต้องดูให้เหมาะสมกับ application ด้วย เช่นถ้าจะ monitor temp/humid ในคืนแต่ส่งข้อมูลทุก 5 วินาทีก็อาจจะได้ไม่ค่อยสมเหตุสมผลเท่าไหร่ เพราะจะทำให้ตัว esp8266/esp32 ที่มีข้อจำกัดเรื่องของการประมวลผลและหน่วยความจำอาจต้องทำงานหนักขึ้น
- Message Dropped: ถึงแม้ระบบ Mesh Network มันจะเป็นอะไรที่เงี่ยมกันดูเหมือนเป็น fault tolerance / high availability network แต่ก็มีโอกาสทำงานผิดพลาดได้ ฉะนั้น message ที่ส่งผ่านก็อาจหายไปได้บ้างในบางโอกาส ฉะนั้นก็ขึ้นอยู่กับ logic ของ application ที่เราพัฒนาว่าจะ handle ยังไง

ໄລ້ຂຶ້ອກມາຮັດບ້າງບນຫນະ ພມກີ່ໄມ້ໄດ້ເກົ່າເຖິງບື້ນມາເອງທຣອກຄົນ ທີມພັດນາ Painlessmesh ເນື່ແທລະນະນຳນາ ພມວ່າກີ່ເຊື້ອໄວ້ເປັນ best practice ບ້າງກີ່ດີຄົນ ເລາຈະຫາຂໍອຸປິດພາດຈະໄດ້ໄມ້ຕ້ອງວາງທາດັ່ງແຕ່ຄູນຍ໌ ຈະໄດ້ມາເຊື້ອໄດ້ວ່າເອົ້າທີ່ມັນສ່າງຂ້ອມູດໄນ່ຜ່ານເປັນພະຮະອະໄຮ memory overflow ນັ້ນ ກ່ຽວເປັນປັບປຸງຫາກກາຍກາພີ່ໄມ້ join ເຂົ້າ mesh ມີວ່າເນີນ delay ໄຊ

Hello Mesh Network

ຫລັງຈາກຕີເຮັດວຽກ environment ໃນການທຳມານີ້ເລີ້ມຕົ້ນທີ່ຈະນາຮົມທົດລອງສ້າງ Mesh Network ແຮກຂອງເຮັດວຽກໃຫຍ່ ຮ່າຍມາຮະຍາ ເຮັມສັກ 5 node ກ່ອນລະກັນ ໂດຍທີ່ Node ຂອງເຮົາຈະປະກອບດ້ວຍ

Wemos D1 Mini, ESP32 Wroom, ESP-01, ແລ້ວກີ່ Nodemcu ອີກສອງດ້ວຍ

ຫັ້ນແຮກເລີຍກີ່ເປີດໄຟລ໌ startHere ກັນກ່ອນແລຍກົນ ທີ່ Arduino IDE ກີ່ເຂົ້າໄປທີ່ File->Examples->Painless Mesh-> startHere

ມາດູກາຮັດຕັ້ງຄ່າຄ່າວາງຈາກ Mesh Network ເຮັດວຽກ

```
#define MESH_SSID "HelloMyMesh"
#define MESH_PASSWORD "hellomymeshnetwork"
#define MESH_PORT 5555
```

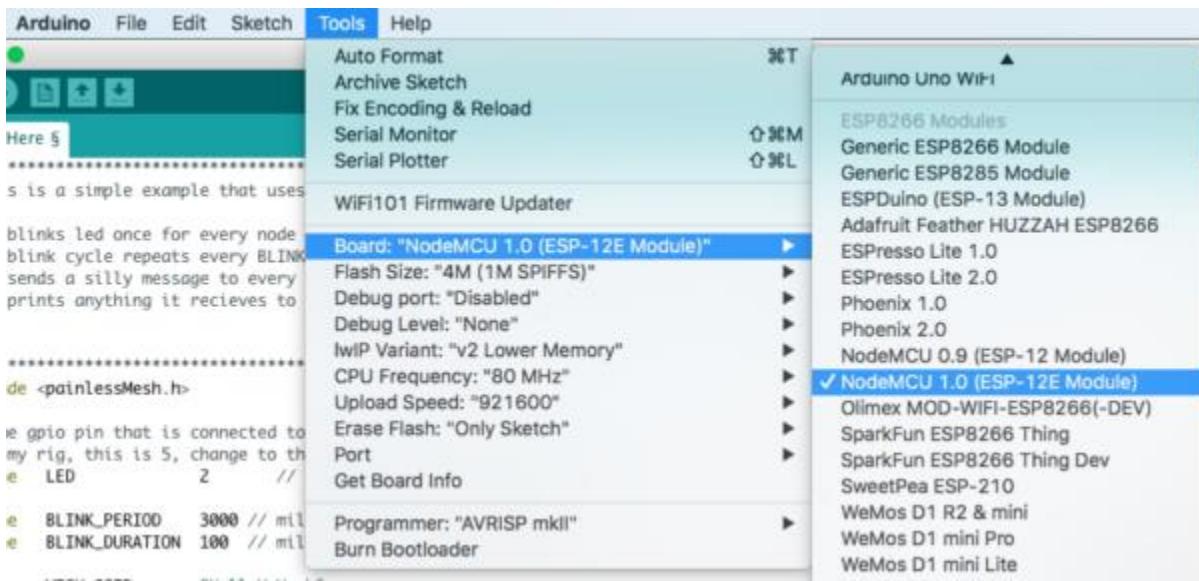
ຈາກໄຟລ໌ startHere ທີ່ມາກັບດ້ວຍໆ ເຮົາສາມາຮັດທີ່ຈະແກ່ກາງຂອງ Mesh ຮຳມັງເພີ່ມ Security ໃນລ່າວຂອງ Mesh Network ເຮົາໄດ້ຈາກ 3 ບຣທັບນີ້ແລຍກົນ ຊຶ່ງ help ຂອງ painlessmesh ຮົມລຶງກຳອົບນຍ໌ທີ່ຢູ່ໃນ code ນັ້ນກ່ອນຂ້າດີເລີຍທີ່ເດືອຍ ຊຶ່ງການທຳມາ Mesh Network ໂດຍໃຊ້ Painlessmesh ນັ້ນຈະແນ່ງການທຳມາເປັນລັກມະນະຂອງ task ໂດຍການກຳຫານຳຈາກ task scheduler ພ່າຍເຫັນ task ທີ່ໃຊ້ໃນການສ່າງ message

```
mesh.scheduler.addTask( taskSendMessage );
taskSendMessage.enable() ;
ແລະໃນລ່າວຂອງ loop ກີ່ຈະມີແລ້ວ ການ update mesh ເປັນຫຼັກ ພາກ logic ດ້ວຍຈະໄປອູ້ທີ່ task scheduler ມີມາດ
mesh.update();
```

ໃນລ່າວຂອງ CallBack ນັ້ນຫຼັກທີ່ໃຊ້ຈາງຈິງເລີຍກີ່ຄື້ອງ receivedCallback ເພື່ອທີ່ຈະຮູ້ວ່າ message ທີ່ເຮົາໄດ້ຮັບມາ ໄນຈະມາຈາກ Broadcast ເຊິ່ງມາແບບຮະນຸ NodeID ນັ້ນເຮົາຈະນີ້ໄປທ່ານໄວ້ຕ້ອງຊື່ໃນດ້ວຍໆ ທີ່ກີ່ຄື້ອງ Print ອອກມາຜ່ານທາງ Serial Port

```
void receivedCallback(uint32_t from, String & msg) {
Serial.printf("startHere: Received from %u msg=%s\n", from, msg.c_str());
}
```

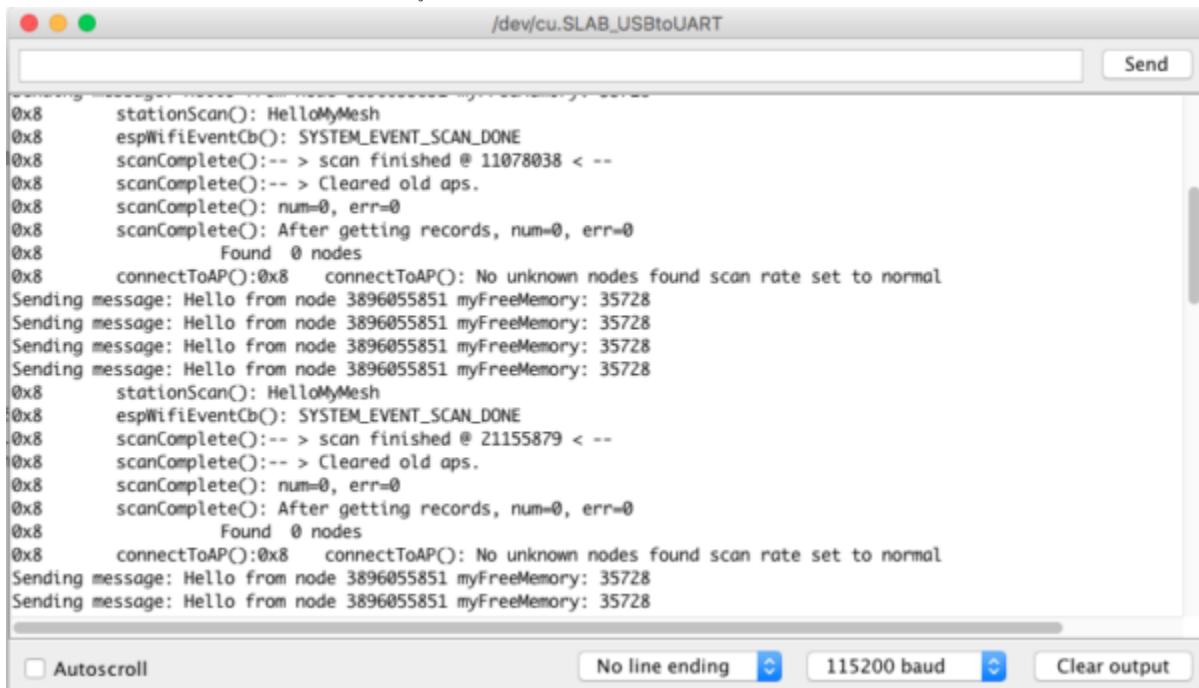
ຊື່ທີ່ພົມແກ້ຫຼັກຈາລີຍກີ່ຄື້ອງດ້ວຍ SSID ແລະ Password ແກ່ນັ້ນຈາກນັ້ນກີ່ກຳກັງ Flash ເລີຍກົນ ເລື້ອກ Board ໄທຕຽງຮູ່ນແກ່ນັ້ນ



ชี้จุดประส่งที่ลักษณะดังนี้คือด้านต่างนี้เลย กระพริบ led ตามช่วงเวลาที่กำหนด แล้วก็ท่องข้อความแบบ broadcast ไปยังทุกๆ node ต้องลงทำงานครับด้วยตัวเองนี้ตัวอย่างเดียวได้嚟 ໄร์เยอะ

```
// 1. blinks led once for every node on the mesh
// 2. blink cycle repeats every BLINK_PERIOD
// 3. sends a silly message to every node on the mesh at a random time
// between 1 and 5 seconds
// 4. prints anything it receives to Serial.print
```

เมื่อทำการ flash nodemcu ด้วยไปแล้วปิดที่ Serial Monitor ดู ข้อมูลมันก็จะ酵ะๆหน่อย เพราะมีการ enable debug ไว้ ในตอน Setup mesh.setDebugMsgTypes(ERROR | DEBUG | CONNECTION); ซึ่งระดับของการ debug ก็มีหลายแบบให้เลือกใช้งานตามที่เหมาะสมดูครับ



ซึ่งเมื่อ flash เสิร์จและเริ่มทำงาน nodemcu ตัวแรกในวง Mesh Network (มีตัวเดียวจะเรียก Mesh ทำนายยย) ก็จะเริ่มหา AP ที่ซึ่งเดียกัน บ้านไก่เรือนเคียงของ node อื่นๆจะ เพื่อที่จะดึงมาเข้าสู่ Mesh นี้โดยข้อมูลจากการ Debug ก็จะได้ nodeid รวมถึง memory ที่เหลืออยู่คละ

คราวนี้เรา flaash nodemcu อีกสองตัวเข้าไปดูบ้างว่าผลลัพท์ที่ได้เป็นอย่างไร (note ไว้ก่อนว่า nodeid ของตัวแรกคือ 5851 เอาแค่ 4 ตัวสุดท้ายพอจะได้ไม่ต้องจำเยอะ และ memory ที่เหลือคือ 35728 byte)

```

0x8 Found 2 nodes
0x8 connectToAP():0x8 connectToAP(): No unknown nodes found scan rate set to normal
startHere: Received from 3896664414 msg=Hello from node 3896664414 myFreeMemory: 31880
startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 33872
Sending message: Hello from node 3893419003 myFreeMemory: 27088
startHere: Received from 3896664414 msg=Hello from node 3896664414 myFreeMemory: 31880
startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 33872
startHere: Received from 3896664414 msg=Hello from node 3896664414 myFreeMemory: 31208
startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 33872
Sending message: Hello from node 3893419003 myFreeMemory: 27088
startHere: Received from 3896664414 msg=Hello from node 3896664414 myFreeMemory: 31880
0x8 stationScan(): HelloMyMesh
0x8 espWifiEventCb(): SYSTEM_EVENT_SCAN_DONE
0x8 scanComplete():--> scan finished @ 260926004 <--
0x8 scanComplete():--> Cleared old aps.
0x8 scanComplete(): num=2, err=0
0x8 scanComplete(): After getting records, num=2, err=0
0x8 found : HelloMyMesh, -29dBm
0x8 found : HelloMyMesh, -15dBm
0x8 Found 2 nodes
0x8 connectToAP():0x8 connectToAP(): No unknown nodes found scan rate set to normal
startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 33872

```

The terminal window includes controls for Autoscroll, line ending (No line ending), baud rate (115200), and a Clear output button.

คราวนี้ร่วมกับ Node ที่อยู่ใน Mesh นี้ 3 ตัวละนั้นคือ 5851, 4414 และ 9003 ซึ่งแต่ละตัวก็จะ Broadcast Message ออกตามเวลาที่สุ่มได้ในแต่ละตัว แต่ในขั้นตอนการกระพริบไฟ Led นั้นจะมีการตั้ง time synchronize เมื่อมี node ใหม่เพิ่มเข้ามาใน mesh ซึ่งจะมีCallBack ไว้วางรับและทำงานร่วมกับ MeshUpdate ที่วน loop หากมี Node ไหน drop connection ไปหรือมี node ไหน join เข้ามาและมีค่าความแรงของสัญญาณเท่าไหร่ เพื่อใช้ในการจัดเส้นทางในการส่งข้อมูล

```

/dev/cu.SLAB_USBtoUART
Send

0x8 espWifiEventCb(): SYSTEM_EVENT_AP_STACONNECTED
startHere: Received from 3896664414 msg=Hello from node 3896664414 myFreeMemory: 28656
0x8 stationScan(): HelloMyMesh
0x8 espWifiEventCb(): SYSTEM_EVENT_SCAN_DONE
0x8 scanComplete():--> scan finished @ 583618487 <--
0x8 scanComplete():--> Cleared old aps.
0x8 scanComplete(): num=1, err=0
0x8 scanComplete(): After getting records, num=1, err=0
0x8     found : HelloMyMesh, -28dBm
0x8     Found 1 nodes
0x8 connectToAP():0x8      connectToAP(): No unknown nodes found scan rate set to normal
startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 33360
Sending message: Hello from node 3893419003 myFreeMemory: 27872
Changed connections [{"nodeId":3896664414,"subs":[]}]
Num nodes: 2
Connection list: 3896664414 3896055851
Sending message: Hello from node 3893419003 myFreeMemory: 27856
Delay to node 3896664414 is 8120 us
Delay to node 3896055851 is 12122 us
startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 33824
startHere: Received from 3896664414 msg=Hello from node 3896664414 myFreeMemory: 31872
Sending message: Hello from node 3893419003 myFreeMemory: 27856
startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 33808

 Autoscroll   No line ending   115200 baud   Clear output

```

แค่ 3 Node ขึ้นอยู่ไป เรากำเพิ่ม Node เข้าไปใน Mesh Network เราภันอีกดีกว่า ถ้าคุณต้องนี่ 3 Node Memory ของ Node ที่ memory เหลือน้อยสุดจะอยู่ที่ 27856 Byte เท่านั้น โดยผมจะเพิ่ม ESP32 Wroom, Wemos D1 Mini และ ESP-01 เข้าไปแล้วมาดูว่า Memory จะเหลือกันมากน้อยขนาดไหนเมื่อขนาดของ Mesh Network โตขึ้น (Built-in LED pin ของ ESP-01 อยู่ที่ Pin 1)

```

/dev/cu.SLAB_USBtoUART
Send

startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 31320
Sending message: Hello from node 3893419003 myFreeMemory: 33360
startHere: Received from 983688460 msg=Hello from node 983688460 myFreeMemory: 20512
startHere: Received from 2752206217 msg=Hello from node 2752206217 myFreeMemory: 122680
startHere: Received from 2487009679 msg=Hello from node 2487009679 myFreeMemory: 27808
startHere: Received from 983688460 msg=Hello from node 983688460 myFreeMemory: 21856
startHere: Received from 3896664414 msg=Hello from node 3896664414 myFreeMemory: 17408
startHere: Received from 2487009679 msg=Hello from node 2487009679 myFreeMemory: 27808
Adjusted time 2278836065. Offset = 3562
Adjusted time 2278841890. Offset = 2064
Adjusted time 2278847262. Offset = -621 Time Sync
startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 31320 Nodemcu02
startHere: Received from 2752206217 msg=Hello from node 2752206217 myFreeMemory: 122680 ESP32 Wroom
startHere: Received from 3896664414 msg=Hello from node 3896664414 myFreeMemory: 17408 Nodemcu03
startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 31320 Nodemcu01
Sending message: Hello from node 3893419003 myFreeMemory: 33360 ESP-01
startHere: Received from 983688460 msg=Hello from node 983688460 myFreeMemory: 21856 Wemos D1 Mini
startHere: Received from 3896664414 msg=Hello from node 3896664414 myFreeMemory: 17408
startHere: Received from 2752206217 msg=Hello from node 2752206217 myFreeMemory: 122676
startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 31320
startHere: Received from 983688460 msg=Hello from node 983688460 myFreeMemory: 21856
startHere: Received from 2487009679 msg=Hello from node 2487009679 myFreeMemory: 27808 Wemos D1 Mini
Sending message: Hello from node 3893419003 myFreeMemory: 33360

 Autoscroll   No line ending   115200 baud   Clear output

```

จากรูปค้านบนมีอ้างว่าทำงานไปได้ซักพักถ้าดู Memory ที่เหลืออยู่ต่ำสุด จะเป็นของ Nodemcu03 ซึ่งเหลืออยู่ 17408 Byte ส่วนที่เหลือมากสุดก็เดาไม่ยากเลย ย่อมเป็น ESP32 MCU ด้วยนั่นแน่นอนอยู่แล้ว ดังที่ทางทีมผู้พัฒนาค้าแย้งเอาไว้ว่า mesh network นั้นจะรองรับได้ถ้า node นั้นอยู่ที่ memory เป็นหลักเลย ไว้มีเวลาตอนที่ 2 จะเพิ่มนode เข้ามาใน mesh network พร้อมกับเพิ่มการส่งข้อมูลที่อ่านได้จากเซนเซอร์ต่างๆ แล้วมานำเสนอว่า ขนาดที่เพิ่มขึ้นพร้อมกับ operation logic ที่เพิ่มขึ้น จะทำให้โครงสร้าง mesh นี้เปลี่ยนไปบังไงกันบ้าง รับรองว่าตอนต่อไปจะได้นำ mesh network มาใช้งานในการส่งข้อมูลกันจริงๆ กะ ส่วนตอนต่อๆ ไปอีกจะมีอะไรบ้าง ก็ลิสต์ขึ้นมาให้คุณเรียกแรกกันจะหน่อย

ตอนที่ 2: ภาคต่อของ Painlessmesh Client/Server

ตอนที่ 3: Painlessmesh Bridge

ตอนที่ 4: Painlessmesh Bridge with Lora

ตอนที่ 5: ของแถม เดียวถ้าคิดจะเขียนอะไรมี อ้อ อาจจะเป็น การติดตั้ง MQTT Server / InfluxDB / Grafana บน Google Cloud แล้วแจ้ง Alert ผ่านทาง line กันนะสันใจดี ทำเที่ยววนจนลืมละ จะได้อ่านมาบทหวาน เพื่อไว้ให้ดาวดวงกลับมาอ่าน

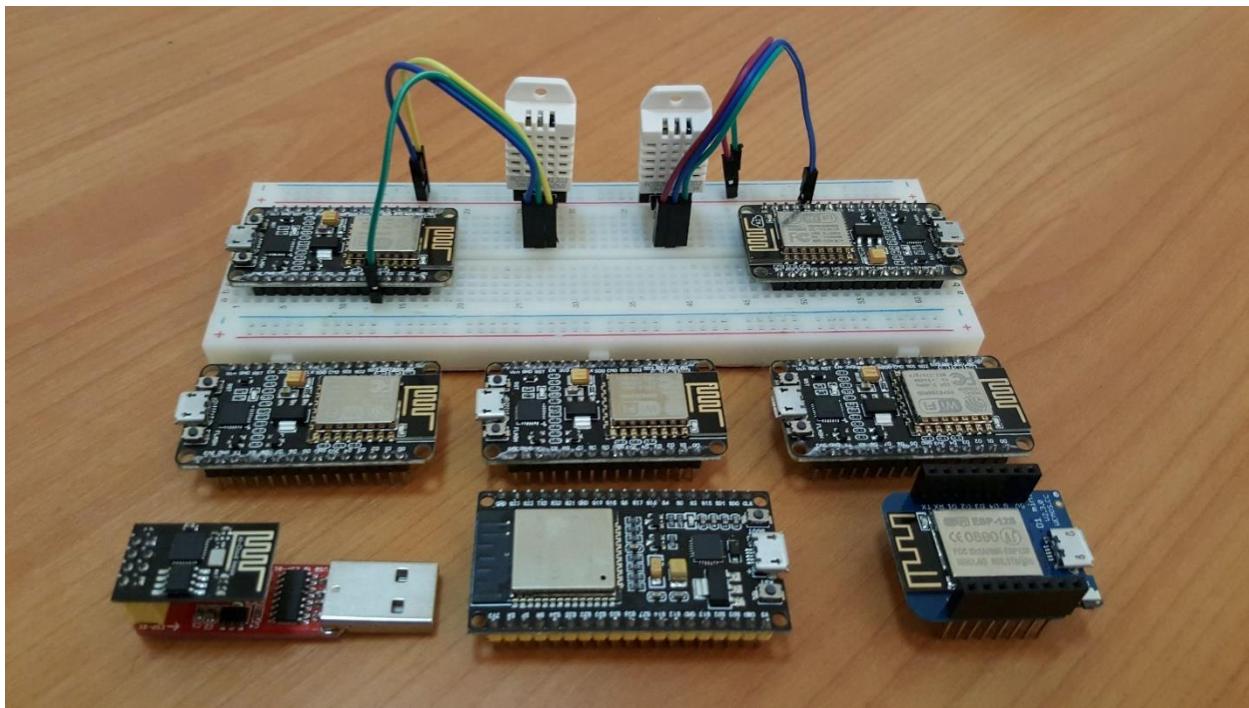
เรื่องเล่าท้ายตอน

คุณพิชญ์ แห่งอาณาจักรสมินาเคยสอนผมไว้ว่า “คุณมีลูกน้องเก่งๆ ติด จะได้ทำงานสบาย” จะเรียกว่าเป็นประโยคเบลี่ยนชีวิตก็ได้มั้ง มันเหมือนทำให้เกร็อดได้ถ้าเราจะสอนคนก่ออย่างก้าว แล้วจะให้โอกาสคน งานที่ทำงานกับคนเก่งๆ ยิ่งเก่งกว่าเราอีกต่อไป ยิ่งทำให้เราพัฒนาขึ้น และหวังผลลัพธ์ของงานได้

ถ้าใครจะเปิดคอร์สอบรมและถ้าคอร์สของคุณดีจริง ไม่ต้องห่วงหรอกว่าแค่ keyword เดียวที่คุณเขียนหาข้อมูลคอร์สสูญ ถึงขั้นต้องลบออก แล้วจะทำให้ขายคอร์สไม่ได้นะ ผมก็มีไปเรียนรึ่ง IoT ของอีกที่นึงไม่บอกแหล่ง เด่าว่าหน้าม้าคราฟ์อีก ก่อนไปเรียนด้วย workshop ที่ทำในห้องเรียนผมทำได้ 90% แล้วเหละ แต่ที่จำดังค์ไปเรียนคือจากไปเปิดโลก ไปแลกเปลี่ยนความคิดเห็น บางเรื่องอาจารย์ให้ข้อมูลพิเศษก็มีเราก็แข่ง บางเรื่องอาจารย์ให้ข้อมูลที่เราไม่รู้ หรือเข้าใจคิดเราเก็บไว้

ถ้าคอร์สที่จัดโดย [ESP8266 Mesh Network](#) หรือ ของคุณสุพจน์ แซ่เบย ดีจริงอย่าไปกลัวเลยครับ บางครั้งผมก็เจอลูกน้องเก่งกว่า เจอรุ่นน้องเก่งกว่า เจอกันที่ผู้สอนงานให้เก่งกว่า คือจะอีกท่าให้เราได้เปิดรับโลกใหม่ๆ ความรู้ใหม่ๆ อย่าตัดสินคนด้วยข้อมูลที่ไม่รอบค้านคือว่า เนอะๆ

- ESP Mesh Network 2 - <https://meetjoeblog.com/2018/03/27/esp8266-esp32-mesh-network-painlessmesh-client-server-ep2/>



POSTED IN TECHNOLOGY

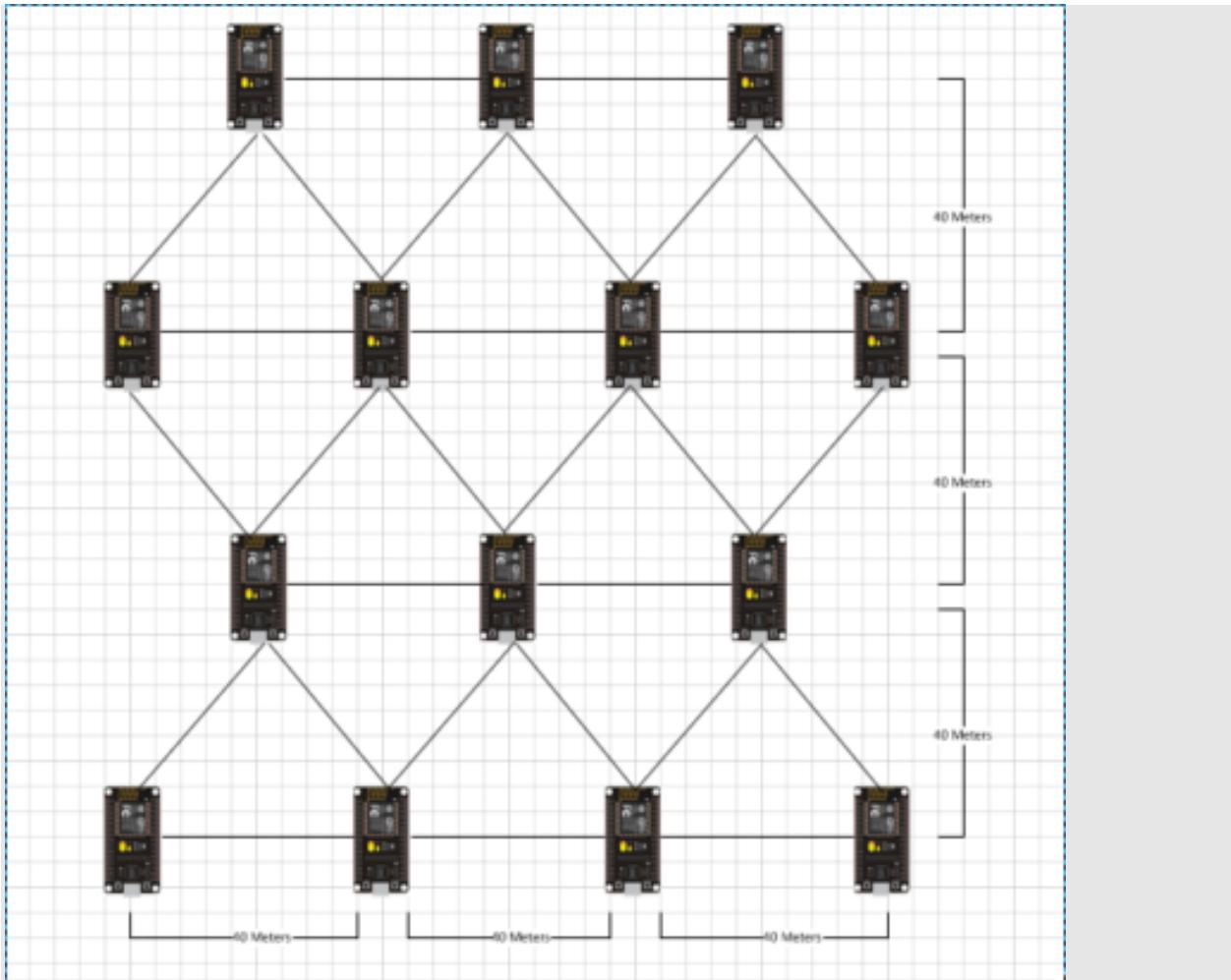
ESP8266 / ESP32 & Mesh Network ตอนที่ 2: ภาคต่อของ Painlessmesh Client/Server

JOE D.S. 27/03/2018 2 COMMENTS ON ESP8266 / ESP32 & MESH NETWORK ตอนที่ 2: ภาคต่อของ PAINLESSMESH CLIENT/SERVER

ความเดิมตอนที่แล้ว

จาก ตอนที่ 1: Introduction & Painlessmesh ผู้อ่านน่าจะได้เข้าใจหลักการทำงานของ Mesh Network และการใช้งาน Library Painlessmesh ร่วมกับ ESP8266/ESP32 กันไปบ้างแล้วซึ่งเป็นการ Broadcast Message ไปยังทุก Node ที่อยู่ใน Mesh Network ซึ่งทั้งนี้ทั้งนั้นอย่างให้เข้าใจหลักการการใช้งานกันก่อน เพราะถ้าเราเข้าใจหลักการแล้ว การประยุกต์ใช้งานจะทำได้ง่ายและเหมาะสมขึ้น จึงเราก็มาทวนกันหน่อยละกันเนอะ

High Availability / Fault Tolerance

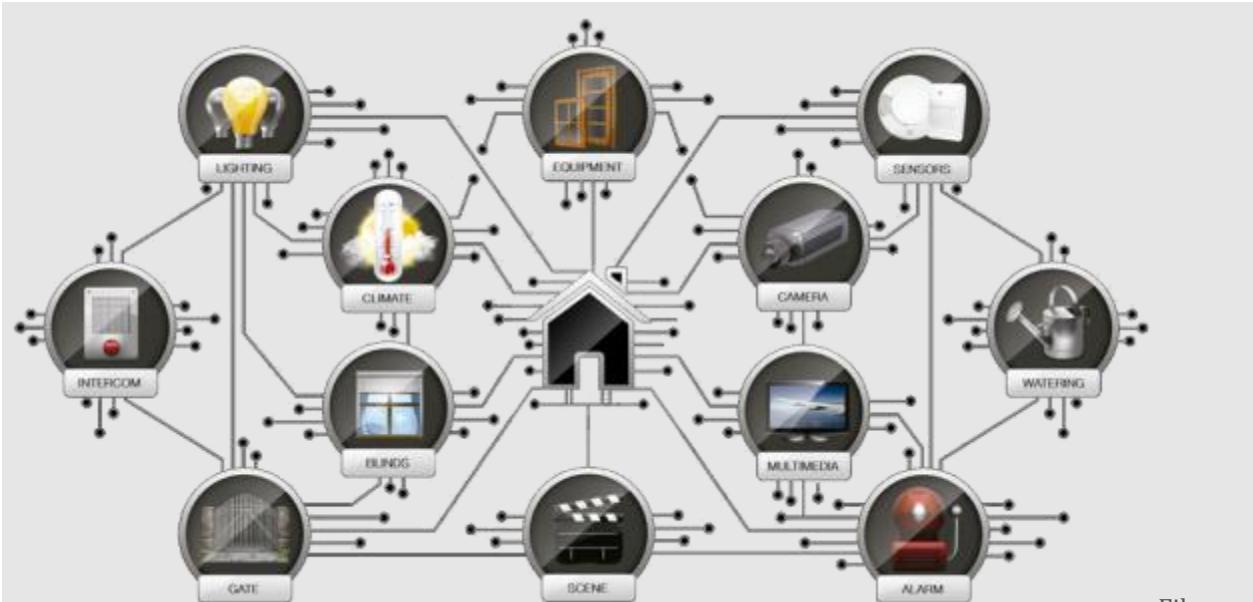


ขยายความจากตอนเดิม Mesh Network นั้นข้อดีก็คือเรื่องของความสเลียร์ สะตอ แบบไม่ต้อง setup อะไรเลย ถ้าคุณต้องซ่อมกลุ่มค่าหนึ่งตัวก็ได้ เส้นทางในการส่งข้อมูลก็ยังคงไม่ถูกตัดขาด เพราะค่ายคุณสมบัติของ Self-Organize / Self-Configure ก็จะทำให้สามารถส่งข้อมูลไปยังปลายทาง หรือ Node ที่ยัง Active อยู่ได้ โดยไม่ต้องมีอุปกรณ์เป็นศูนย์กลางเหมือนระบบ WiFi ที่มีการทำงานแบบ Star ซึ่งต้อง WiFi Router หรือ AP คู่กับทุก Node ที่จะ

Coverage Area

จากรูปด้านบน ถ้าคิด “เคลื่อน” ที่ hop ละ 40 เมตร นี่คิดแบบ Conservative สุดๆไปเลยนะ เพียงแค่ 14 Node ก็สามารถทำให้มีการสื่อสารครอบคลุมพื้นที่ได้ถึง 14,400 ตารางเมตร หรือ 9 ไร่กันเลยทีเดียว ถูกกว่าชื่อ AP หรือ Repeter มาใช้จะอีก

ลักษณะการทำงานของ Mesh Network นั้นมักจะเป็นการใช้งานแบบ local network อย่างที่เราเห็นกันในระบบ home automation หรือแม้กระทั่ง smart meter บางรุ่น จะนั้นถ้าจะทำให้ Home Automation หรือ Local Mesh Network ที่เราสร้างขึ้นมาบันทุรกันไปข้างนอกได้ ก็จะมีอีก Node หนึ่งขึ้นมาเพื่อเป็นสะพาน (Bridge) และใช้เป็นประตู (Gateway) ในการ forward ข้อมูลหรือใช้ในการสื่อสารกับ Network 外面的



Fibaro

z-wave mesh network ใช้อุปกรณ์ที่ชื่อ Home Center เป็นตัว Bridge ระหว่างอุปกรณ์ต่างๆกัน Network ภายนอกซึ่งรองรับทั้ง WiFi และ Ethernet

ในระบบ Home Automation ที่ใช้ Zigbee หรือ Z-Wave หรือบางระบบซึ่งรองรับการสื่อสารทั้งสองแบบก็จะมีอุปกรณ์ที่เรียกว่า Gateway ไว้เป็นตัว Bridge ระหว่าง Local Mesh Network กับการสื่อสารผ่านโครงข่ายข้างนอก ไม่ว่าจะเป็น WiFi, Internet หรือแม้กระทั้งข้ามไปมาระหว่าง Zigbee กับ Z-Wave ซึ่งเดียวเราจะมาต่อ กันในบทที่ 3 เรื่อง Bridge ระหว่าง Mesh Network กับ Network ภายนอกกัน

Power Consumption

ถ้าจุดที่เราติดตั้งมีไฟฟ้าใช้งานตลอดเวลา หรือมีความสะดวกสบายในการเดินสายไฟ ประเด็นเรื่อง Power Consumption อาจจะไม่ต้องคิดมากนัก แต่ในระบบ home automation มักจะมีอุปกรณ์ประเภทที่ติดตั้งไว้โดยค่า หรือตอนที่บ้านสร้างเสร็จแล้ว อาจไม่สะดวกในเรื่องของการเดินสายไฟ ซึ่งอุปกรณ์เหล่านี้ก็จะเป็นพวก

- PIR ไว้ตรวจความเคลื่อนไหวต่างๆ
- Leak Sensor ไว้คุ้นหัวร้าวได้เชิงลักษณะ หรือได้หลังค้นห้ำ
- IR Remote ซึ่งทำให้เราสามารถท่องเว็บได้โดยไม่ต้องเดินสายไฟ

ที่ยกตัวอย่างไป ถ้าบ้านที่สร้างเสร็จไปแล้ว และต้องมาเดินสายไฟ อาจไม่สะดวกนัก Zigbee กับ Z-wave จึงมาตอบโจทย์ตรงนี้ ในการเชื่อมต่อ กับอุปกรณ์ที่ต้องเดินสายไฟ เช่น ไฟต่อ A23 ไปสองก้อนอยู่ได้เป็นปี ทำให้สะดวกมากๆ แต่อุปกรณ์เหล่านี้มีราคาแพง ถ้าเทียบกับ ESP8266/ESP32

ซึ่งถ้าเราจะใช้ ESP8266/ESP32 และหลักสูตรที่ส่งหนึ่งที่ต้องคำนึงถึงเล็กๆน้อยๆเรื่องของการจ่ายไฟให้ Node เหล่านี้ทำงานอยู่ตลอดเวลา ไม่สามารถที่จะ Sleep เพื่อที่จะประหยัดไฟได้ เราสามารถดูกันว่าในสถานการณ์ต่างๆเจ้า nodemcu ของเรากินกระแสมากน้อยขนาดไหน ซึ่งลองวัดกันแบบง่ายๆผ่าน usb tester กัน ซึ่งถ้าไกรนี Oscilloscope หรือ Multi-meter จะได้ผลลัพธ์ที่น่าจะดีกว่าได้ละเอียดขึ้น



Scenario1: rom เปล่าฯ วนลูปไปเรื่อย ค่าที่ได้ 80mA

Scenario2: ต่อ WiFi ทึ่งไว้ตามปกติ ค่าที่ได้อูฐ์ที่ 80mA

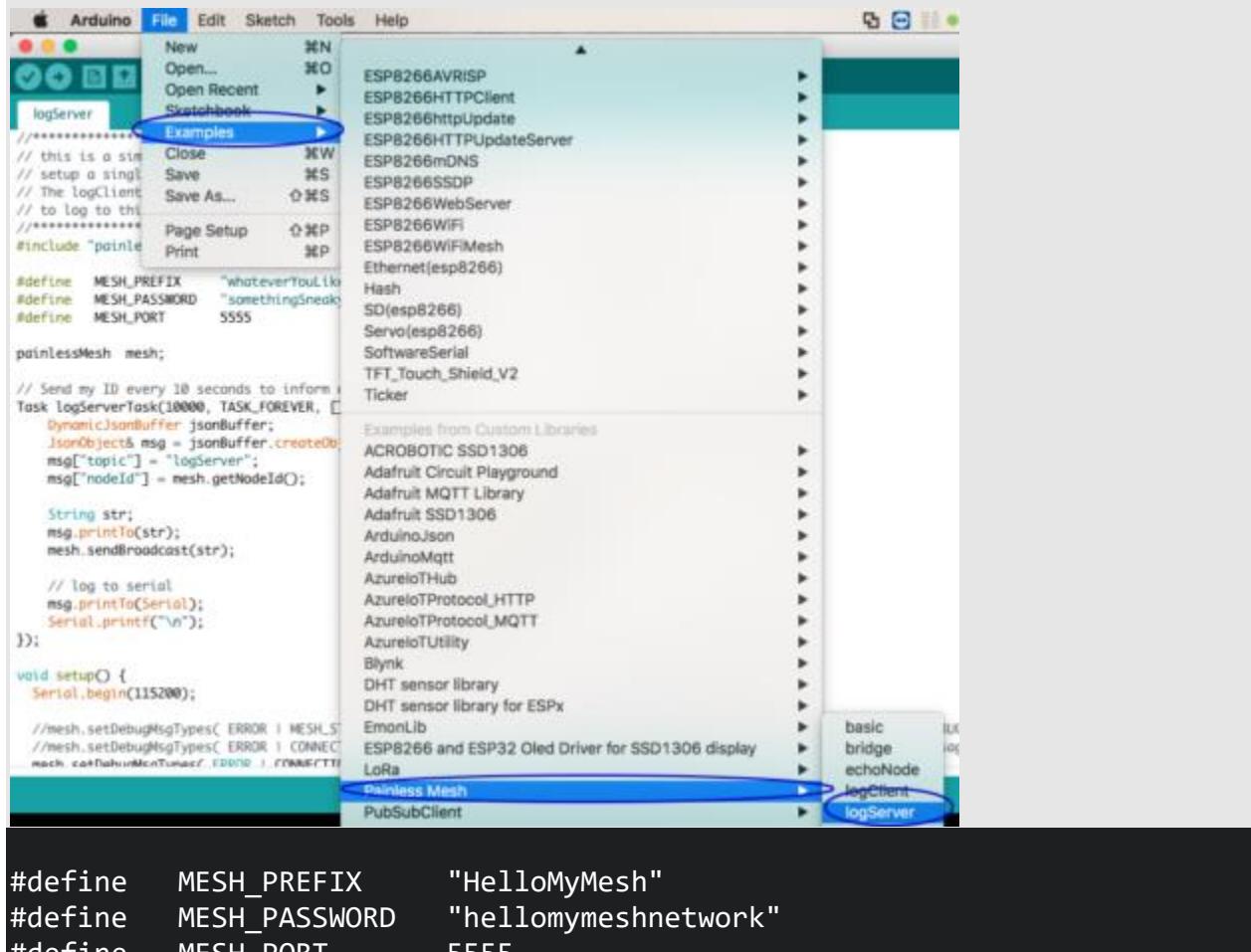
Scenario3: Join Mesh Network (ใช้ Painlessmesh) 2-6 Nodes Broadcast Message ค่าที่สังเกตุได้อูฐ์ที่ประมาณ 80-90mA มีค่า Peak อูฐ์ที่ประมาณ 150mA ซึ่งน่าจะมาจากตอน Led กระพริบและตอนที่ Join/Update Mesh Network ซึ่งคาดว่าใช้งานจริงเพิ่ม Node เข้าไป น่าจะทำให้อัตราการกินกระแสสูงขึ้น ถ้าอย่างให้กระแสน้ำดีลาก่อนจะต้องพิจารณาใช้ ESP12E อย่างเดียว ตัดพวง LED หรือ Serial to USB IC ออก ซึ่งก็จะทำให้ประหยัดไฟเพิ่มขึ้น แต่ก็ต้อง On Node ไว้อูฐ์คือต้องการให้มีการเชื่อมต่อ

สนับสนุนถ้าจะเอา ESP8266/ESP32 มาทำ Mesh Network เพื่อรับส่งข้อมูล หรือ Control อุปกรณ์ต่างๆแล้ว น่าจะต้องคำนึงในเรื่องของภาคจ่ายไฟด้วย ในโรงงานหรืออาคารที่ไม่มีปัญหาเรื่องการจ่ายไฟ น่าจะเหมาะสม ส่วนงานประเภท Outdoor / Open Field อาจต้องพิจารณาในเรื่องของการนำพา Solar Cell มาใช้ ก็เป็นอีกทางเลือกนึงที่น่าสนใจ ซึ่งจากข้อมูลด้านบนก็จะทำให้เราสามารถเตรียมในเรื่องของขนาด Solar Cell และ Battery สำรองไฟไว้ให้เหมาะสมได้ (อย่าลืมวัดจากการใช้งานใน application จะงด้วย เพราะเซนเซอร์บางอย่างก็กินกระแสไฟเดเมื่อนกัน)

ในตอนที่ผ่านมาได้รับการที่ผ่านมาอ่านแล้วได้ทดลองทำตามตัวอย่างแรกไป อาจเริ่มตัวอย่างที่สองในส่วนของการทำงานลักษณะที่เป็น Client/Server ไปแล้วก็ได้ เพราะตัวอย่างที่ให้มานั้นค่อนข้างคิดเลยที่เดียว แต่ถ้าใครยังไม่ได้เริ่ม ก็มาเริ่มพร้อมๆกันเลยครับ เกริ่นช่วยอีกแล้ว

เริ่มใช้งาน Painlessmesh แบบ Client/Server

ตัวอย่างประกอบสำหรับการใช้งานแบบ Client/Server จะใช้อยู่สองตัวคือ logServer และ logClient โดยสามารถเลือกเป็นไฟล์จาก File->Examples->Painless Mesh-> logServer หรือ logClient



โดยเราจะเริ่มที่ logServer กันก่อนครับ ซึ่ง concept ก็ยังคงเหมือนเดิม ให้แก้ไขในส่วนของการตั้งค่า Mesh Network ของเราซึ่งก็คือ 3 บรรทัดด้านบนนี้ครับ หรือในรูปจะเห็นไว้หนึ่งเมื่อนดาวบ่ายที่ให้มาร์กไว้แล้วก็ Flash ลง ESP8266/ESP32 ของเราได้เลย

แต่ในส่วนของคนที่ใช้ ESP32 ต้องแกะนิคเน็ตครับ เพราะการ Declare ในส่วนของ Authentication Mode นั้นไม่เหมือนกัน แต่ ESP8266 จะแก้ไขให้เป็นแบบ ESP32 ก็สามารถใช้งานได้เหมือนกันครับ

```
mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT, STA_AP,  
WIFI_AUTH_WPA2_PSK, 6 ); // <-- ต้องเพิ่มคำว่า WIFI_ เป้าไปด้วย
```

Concept ของตัวอย่าง logServer นั้นคล้ายๆกับตอนที่ 1 ครับเพียงแต่ว่า Broadcast Message ที่ส่งออกไปนั้น มีจุดประสงค์ในการส่งออกไปเพื่อบอกให้ node ที่อยู่ใน Mesh Network นี้รู้ว่า ลัพ NodeID นี้จะเป็น Server Node นะ ซึ่งก็จะทำงานด้วย task scheduler ทุกๆ 10 วินาทีนั้นเมื่อมี node ใหม่เข้ามา join ใน Mesh Network ก็จะรู้แล้วว่าจะต้องส่งข้อมูลให้กับรัช

รูปแบบ Message ที่ส่งออกไปก็อยู่ในฟอร์แมตของ JSON ครับ โดยในส่วนของ receivedCallback ก็ยังเหมือนเดิมคือ ได้รับ message มาแล้ว Print ออกทาง Serial Port ไม่มีการเปลี่ยนแปลง

```
Task logServerTask(10000, TASK_FOREVER, []() {
    DynamicJsonBuffer jsonBuffer;
    JsonObject& msg = jsonBuffer.createObject();
    msg["topic"] = "logServer";           // กำหนด Topic เพื่อที่ Node ที่ join เข้ามาหาข้อมูลของ
Server จะ
    msg["nodeId"] = mesh.getNodeId(); // ใช้คำสั่ง mesh.getNodeID(); เพื่ออ่านค่า
NodeID

    String str;
    msg.printTo(str);
    mesh.sendBroadcast(str);           // ส่งข้อความ Broadcast ไปยังทุก Node ว่าตนคือ
Server Node นะ

    // log to serial
    msg.printTo(Serial);
    Serial.printf("\n");
});
```

```

/dev/cu.SLAB_USBtoUART
Send

0x8 stationScan(): HelloMyMesh
0x8 espWifiEventCb(): SYSTEM_EVENT_SCAN_DONE
0x8 scanComplete():--> scan finished @ 21130500 < --
0x8 scanComplete():--> Cleared old aps.
0x8 scanComplete(): num=0, err=0
0x8 scanComplete(): After getting records, num=0, err=0
0x8 Found 0 nodes
0x8 connectToAP():0x8 connectToAP(): No unknown nodes found scan rate set to normal
0x8 espWifiEventCb(): SYSTEM_EVENT_AP_STACONNECTED
0x8 New AP connection incoming
0x8 meshConnectedCb(): we are AP
0x8 findConnection(2487009679): did not find connection
0x8 newConnectionTask():
0x8 newConnectionTask(): adding 2487009679 now= 24504763
New Connection 2487009679
0x20 handleTimeSync(): Response sent {"type":2,"t0":2134652,"t1":24514068,"t2":24514302}
0x20 handleTimeSync(): timeSyncStatus with 2487009679 completed
0x20 handleTimeSync():
0x20 handleTimeSync(): Response sent {"type":2,"t0":24719063,"t1":24722058,"t2":24722307}
0x20 handleTimeSync(): timeSyncStatus with 2487009679 completed
0x20 handleTimeSync():
{"topic":"logServer", "nodeId":2752206217} Broadcast Message งานใน เรายัง Server NodeID 6217
0x8 stationScan(): HelloMyMesh
0x8 espWifiEventCb(): SYSTEM_EVENT_SCAN_DONE
0x8 scanComplete():--> scan finished @ 31268843 < --
0x8 scanComplete():--> Cleared old aps.
0x8 scanComplete(): num=1, err=0
0x8 scanComplete(): After getting records, num=1, err=0
0x8 found : HelloMyMesh, -46dBm
0x8 Found 1 nodes
0x8 connectToAP():0x8 connectToAP(): No unknown nodes found scan rate set to normal
logServer: Received from 2487009679 msg={"topic":"sensor", "value":1} ข้อความที่ได้รับเข้ามาอยู่ในรูปของ JSON

 Autoscroll No line ending 115200 baud Clear output

```

หลังจาก Flash เสร็จการทำงานก็เป็นอย่างที่เห็นข้างบนครับ จะเริ่มทำการเชื่อมต่อเข้ามายัง Node ใหม่ที่สามารถ Join ได้บ้างจากนั้นก็ปรับ Offset Time Synchronize ให้เท่ากันเพื่อนำเวลาในวงจร แล้วก็เข้าสู่ Task ที่ใช้ในการ Broadcast Message ออกไปว่าเราคือ Server เพื่อให้ Node ทุกด้วยที่อยู่ใน Mesh Network เดียวกันรู้ว่าจะส่งข้อมูลให้ Server ให้ส่งไปที่ NodeID หมายเลขอะไรซึ่งในที่นี่คือ 6217

มาตรฐานส่วนของ logClient กันบ้าง ซึ่งในส่วนของ receivedCallback เป็นแค่การเปลี่ยนไป เพราะจะมีการเช็ค Broadcast Message จาก Node อื่นๆ ซึ่งก็คือ Server Node ว่า เห้ยยย นี่แหละ Server Node นะ มี ID นี้นะ จากนี้ถ้าจะส่งข้อความไปที่ ServerNode ให้ส่งไปที่ NodeID นี้นะ

```

void receivedCallback( uint32_t from, String &msg ) {
    Serial.printf("logClient: Received from %u msg=%s\n", from, msg.c_str());
    // Saving logServer
    DynamicJsonBuffer jsonBuffer;
    JsonObject& root = jsonBuffer.parseObject(msg);
    if (root.containsKey("topic")) {

```

```

if (String("logServer").equals(root["topic"].as<String>0)) { // เช็ค Topic ว่าเป็น logServer หรือเปล่า

// check for on: true or false

logServerId = root["nodeId"]; // ถ้ามันและบันทึกค่า nodeID ของ Server เพื่อใช้เป็นปลายทางในการส่ง

Serial.printf("logServer detected!!!\n");

}

Serial.printf("Handled from %u msg=%s\n", from, msg.c_str());

}
}

```

สำหรับส่วนของ Task ที่ใช้ในการส่ง Message ก็จะทำงานลือกัน คือถ้าจงไม่เจอกว่า ใน Mesh Network ที่เพิ่ง join เข้ามานะ server หรือเปล่า ถ้าก็ broadcast ข้อมูลของฉันแลยกัน เป็นของ Server Node มีระยะเวลา 10 วินาทีถึงจะ Broadcast Message Topic "logserver" ออกໄไป ขณะนั้นการทำเช่นนี้ Server Node ที่รับข้อมูลจาก Client Node ก็จะได้ข้อมูลแน่นๆ (รวมถึงตัวอื่นๆด้วย)

```

Task myLoggingTask(10000, TASK_FOREVER, []() {
    DynamicJsonBuffer jsonBuffer;
    JsonObject& msg = jsonBuffer.createObject();
    msg["topic"] = "sensor";
    msg["value"] = random(0, 180); // สำหรับไฟล์ตัวอย่างนี้ซึ่งไม่ได้ต่อเข้ากับเซนเซอร์อะไร ก็ตุ่มค่าส่งไปเลยกัน

    String str;
    msg.printTo(str);
    if (logServerId == 0) // If we don't know the logServer yet
        mesh.sendBroadcast(str);
    else
        mesh.sendSingle(logServerId, str);

    // log to serial
    msg.printTo(Serial);
    Serial.printf("\n");
});
```

```

0x2 AP tcp server established on port 5555
0x8 stationScan(): HelloMyMesh
{"topic":"sensor2","value":173}
0x8 espWifiEventCb(): SYSTEM_EVENT_SCAN_DONE
0x8 scanComplete():--> scan finished @ 21204468 <--
0x8 scanComplete():--> Cleared old aps.
0x8 scanComplete(): num=2, err=0
0x8 scanComplete(): After getting records, num=2, err=0
0x8     found : HelloMyMesh, -49dBm
0x8     found : HelloMyMesh, -46dBm
0x8 Found 2 nodes
0x8 findConnection(2752206217): did not find connection
0x8 findConnection(2487009679): did not find connection
0x8 connectToAP():0x8 connectToAP(): Best AP is 2487009679<---
0x8 connectToAP(): Trying to connect, scan rate set to 4*normal
0x8 espWifiEventCb(): SYSTEM_EVENT_STA_AUTHMODE_CHANGE
0x8 espWifiEventCb(): SYSTEM_EVENT_STA_CONNECTED
0x8 espWifiEventCb(): SYSTEM_EVENT_STA_GOT_IP
0x8 New STA connection incoming
0x8 meshConnectedCb(): we are STA
0x8 findConnection(2487009679): did not find connection
0x8 newConnectionTask():
0x8 newConnectionTask(): adding 2487009679 now= 22385371
{"topic":"sensor2","value":63}
logClient: Received from 2752206217 msg={"topic":"logServer","nodeId":2752206217} Broadcast Message ที่ได้รับ
logServer detected!!!
Handled from 2752206217 msg={"topic":"logServer","nodeId":2752206217} ผู้ใช้งานที่นี่ Topic Server ที่ทำการบันทึก Serve NodeID
{"topic":"sensor2","value":2} ที่เก็บข้อมูลในรูปแบบ JSON
logClient: Received from 2752206217 msg={"topic":"logServer","nodeId":2752206217}
logServer detected!!!
Handled from 2752206217 msg={"topic":"logServer","nodeId":2752206217}
{"topic":"sensor2","value":149} ที่เก็บข้อมูลในรูปแบบ JSON

```

Autoscroll No line ending 115200 baud Clear output

คราวนี้แทนที่เราจะจำลองข้อมูลเมื่อตอนตัวอย่าง ก็ได้เวลาเปิดกระปองและ เอา DHT22 มาพ่วงเข้าไป คราวนี้ Temp/Humid ที่ได้ก็จะ เป็นค่าจริงแล้ว โดยการปรับ Code ในส่วนของ Task ที่ใช้สำหรับการส่งข้อมูลนั้น จะมีการอ่านค่าจาก DHT22 และจัดรูปแบบให้อยู่ใน ฟอร์แมตของ JSON โดยที่ Mesh Network นี้จะประกอบไปด้วย 12 Node ด้วยกันดังนี้



- ESP32 Wroom ทำตัวเป็น Server
- ESP-01 เป็น Client ที่ส่งค่าสู่มาร์ชีน
- Wemos D1 Mini 4 ตัว เป็น Client ที่ส่งค่าสู่มาร์ชีน (Wemos-01 ถึง Wemos-04)
- Wemos D1 Mini 1 ตัว เป็น Client ที่ส่งค่า Temp/Humid ที่อ่านได้จาก DHT22
- Nodemcu 4 ตัว เป็น Client ที่ส่งค่าสู่มาร์ชีน (mcu-01 ถึง mcu-04)
- Nodemcu 1 ตัว เป็น Client ที่ส่งค่า Temp/Humid ที่อ่านได้จาก DHT22

Code ที่ทำการแก้ไขสำหรับ Node ตัวที่สู่มาร์ชีน (ESP-01, Wemos-01 ถึง 04, mcu-01 ถึง 04)

```
// Send message to the logServer every 10 seconds
Task myLoggingTask(5000, TASK_FOREVER, []() {
    DynamicJsonBuffer jsonBuffer;
    JsonObject& msg = jsonBuffer.createObject();
    msg["nodename"] = "mcu3"; //เพิ่ม NodeName เข้ามาพอหลายตัว
    //รีบูตเมื่อ NodeID เปลี่ยน
    msg["NodeID"] = mesh.getNodeId(); //เพิ่ม NodeID เอาไว้อ้างอิง
    msg["random value"] = random(0, 180); //เปลี่ยนเป็น random value ให้
    //ชัดเจนว่าเป็นค่าที่สู่มาร์ชีนมา

    String str;
    msg.printTo(str);
    if (logServerId == 0) // If we don't know the logServer yet
        mesh.sendBroadcast(str);
    else
        mesh.sendSingle(logServerId, str);
```

```
// log to serial
msg.printTo(Serial);
Serial.printf("\n");
});
```

Code ที่ทำการแก้ไขสำหรับ Node ตัวที่ส่งค่า Temp/Humid จาก DHT22 (Wemos-t1 และ mcu-t1)

```
#include "painlessMesh.h"

#include "DHT.h"

#define DHTPIN D4

#define DHTTYPE DHT22

#define MESH_PREFIX "HelloMyMesh"

#define MESH_PASSWORD "hellomymeshnetwork"

#define MESH_PORT 5555

DHT dht(DHTPIN, DHTTYPE);

void receivedCallback( uint32_t from, String &msg );

painlessMesh mesh;

size_t logServerId = 0;

// Send message to the logServer every 5 seconds

Task myLoggingTask(5000, TASK_FOREVER, []() {

    float h = dht.readHumidity();

    float t = dht.readTemperature();

    DynamicJsonBuffer jsonBuffer;

    JsonObject& msg = jsonBuffer.createObject();

    msg["nodename"] = "Wemos-t1";

    msg["NodeID"] = mesh.getNodeId();

    msg["Temp"] = String(t)+"C";

    msg["Humidity"] = String(h)+"%";

    String str;
```

```

msg.printTo(str);

if (logServerId == 0) // If we don't know the logServer yet

mesh.sendBroadcast(str);

else

mesh.sendSingle(logServerId, str);

// log to serial

msg.printTo(Serial);

Serial.printf("\n");

});

void setup() {

Serial.begin(115200);

Serial.println("Begin DHT22 Mesh Network test!");

dht.begin();

mesh.setDebugMsgTypes( ERROR | STARTUP | CONNECTION ); // set before init() so that you can see startup messages

mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT, STA_AP, AUTH_WPA2_PSK, 6 );

mesh.onReceive(&receivedCallback);

// Add the task to the mesh scheduler

mesh.scheduler.addTask(myLoggingTask);

myLoggingTask.enable();

}

void loop() {

// put your main code here, to run repeatedly:

mesh.update();

}

void receivedCallback(uint32_t from, String &msg) {

Serial.printf("logClient: Received from %u msg=%s\n", from, msg.c_str());

// Saving logServer

```

```

DynamicJsonBuffer jsonBuffer;

JsonObject& root = jsonBuffer.parseObject(msg);

if (root.containsKey("topic")) {

if (String("logServer").equals(root["topic"].as<String>())) {

// check for on: true or false

logServerId = root["nodeId"];

Serial.printf("logServer detected!!!\n");

}

Serial.printf("Handled from %u msg=%s\n", from, msg.c_str());

}

}

```

หลังจาก Flash ครบตัวสุดท้ายก็เปิด Serial Monitor ขึ้นมาจะเห็นได้ว่ากรุํปข้างล่างเลขครับว่า เจอ Node ซึ่งเกียงอยู่ทั้งหมด 10 Node พร้อมกับความแรงของสัญญาณจากแต่ละ Node ซึ่งจะใช้ในการจัดการการเชื่อมต่อของ Mesh Network ของเรา

```

/dev/cu.SLAB_USBtoUART
Send

0x8    scanComplete():-- > scan finished @ 5750896 < --
0x8    scanComplete():-- > Cleared old ops.
0x8    scanComplete(): num=10, err=0
0x8    scanComplete(): After getting records, num=10, err=0
0x8        found : HelloMyMesh, -41dBm
0x8        found : HelloMyMesh, -42dBm
0x8        found : HelloMyMesh, -35dBm
0x8        found : HelloMyMesh, -37dBm
0x8        found : HelloMyMesh, -50dBm
0x8        found : HelloMyMesh, -43dBm
0x8        found : HelloMyMesh, -61dBm
0x8        found : HelloMyMesh, -59dBm
0x8        found : HelloMyMesh, -56dBm
0x8        found : HelloMyMesh, -57dBm
0x8    Found 10 nodes
0x8    findConnection(983688460): did not find connection
0x8    findConnection(2488971084): did not find connection
0x8    findConnection(2487009679): did not find connection
0x8    findConnection(2136600749): did not find connection
0x8    findConnection(3163409689): did not find connection
0x8    findConnection(572025877): did not find connection
0x8    findConnection(3893419003): did not find connection
0x8    findConnection(3896713776): did not find connection
0x8    findConnection(3163408206): did not find connection
0x8    findConnection(3896055851): did not find connection
0x8    connectToAP():0x8    connectToAP(): Best AP is 2487009679<---

                เจอนาน Node ที่เจอกันหมด
                เจอกันหมด

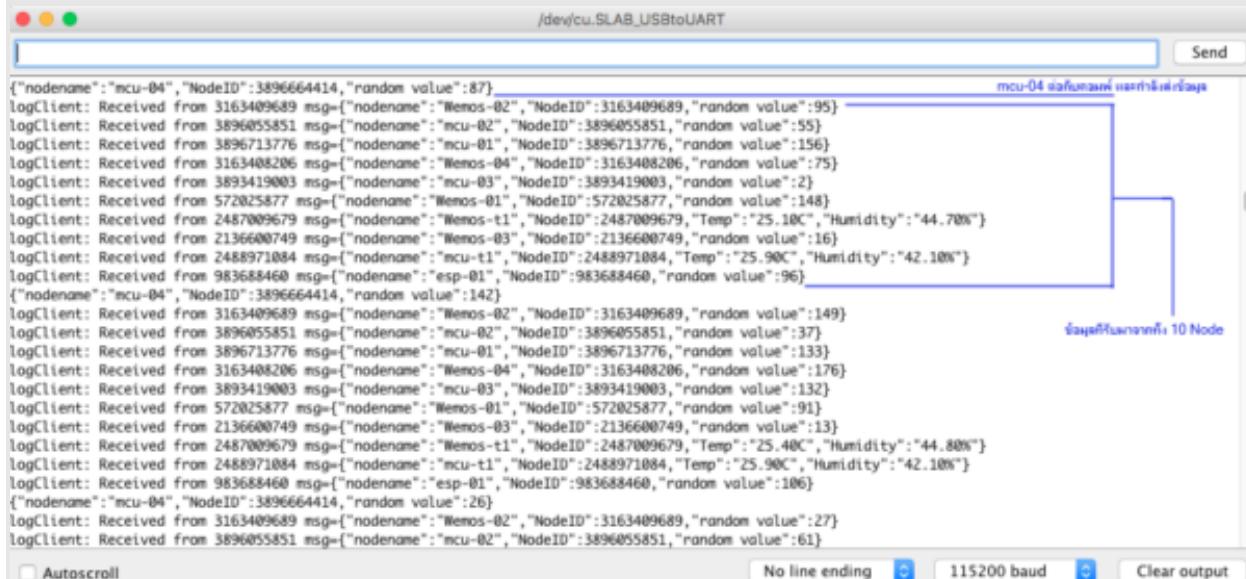
```

Autoscroll No line ending 115200 baud Clear output

อย่างที่ได้บอกไปข้างต้นครับ เมื่อเริ่มการทำงาน Server จะ Broadcast Message เพื่อบอกเพื่อนๆใน Mesh Network นี้ว่า NodeID ของ Server นั้นคือหมายเลขอะไร เพื่อที่ว่าเมื่อบนกรัง ได้มีการเพิ่ม Node เข้าไปใน Mesh Network นี้ก็สามารถที่จะรู้ได้ทันทีว่าจะต้องส่งข้อมูลไปที่ Server NodeID ไหน แต่ระหว่างที่ยังไม่รู้ว่า Server NodeID คือหมายเลขอะไร Client Node ก็จะ Broadcast Message ออกไปเพื่อแจ้งไปก่อน อย่างน้อยมันก็ต้องเข้า Server Node แหล่ง ทราบว่าเราจำลองทดสอบที่ว่านักดูระหว่างที่ยังไม่เจอ Server Node และเมื่อ Server Node Online และทุกๆ Node รับทราบกันหมดแล้วและส่ง Message ออกไปแบบระบุ NodeID

Scenario ที่ 1: No Server Node

ตามรูปด้านล่างนี้เลยครับ ข้อมูลที่ได้รับผ่าน receivedCallback ได้รับข้อมูลจาก Node ที่ Online ใน Mesh Network นี้ ทั้งหมด เพราะว่าซึ่งไม่ได้เปิด Server



```

/dev/cu.SLAB_USBtoUART
Send
mcu-04 @อินเทล์ แมกโนเลียชั่น
[REDACTED] จำนวนต่อไปนี้เป็นตัวอย่าง: 10 Node
[REDACTED]
{
  "nodename": "mcu-04", "NodeID": 3896664414, "random value": 87}
logClient: Received from 3163409689 msg={"nodename": "Remos-02", "NodeID": 3163409689, "random value": 95}
logClient: Received from 3896055851 msg={"nodename": "mcu-02", "NodeID": 3896055851, "random value": 55}
logClient: Received from 3896713776 msg={"nodename": "mcu-01", "NodeID": 3896713776, "random value": 156}
logClient: Received from 3163408206 msg={"nodename": "Remos-04", "NodeID": 3163408206, "random value": 75}
logClient: Received from 3893419003 msg={"nodename": "mcu-03", "NodeID": 3893419003, "random value": 2}
logClient: Received from 572025877 msg={"nodename": "Remos-01", "NodeID": 572025877, "random value": 148}
logClient: Received from 2487009679 msg={"nodename": "Remos-1", "NodeID": 2487009679, "Temp": "25.10C", "Humidity": "44.70%"}
logClient: Received from 2136600749 msg={"nodename": "Remos-03", "NodeID": 2136600749, "random value": 16}
logClient: Received from 2488971084 msg={"nodename": "mcu-t1", "NodeID": 2488971084, "Temp": "25.90C", "Humidity": "42.10%"}
logClient: Received from 983688460 msg={"nodename": "esp-01", "NodeID": 983688460, "random value": 96}
{"nodename": "mcu-04", "NodeID": 3896664414, "random value": 142}
logClient: Received from 3163409689 msg={"nodename": "Remos-02", "NodeID": 3163409689, "random value": 149}
logClient: Received from 3896055851 msg={"nodename": "mcu-02", "NodeID": 3896055851, "random value": 37}
logClient: Received from 3896713776 msg={"nodename": "mcu-01", "NodeID": 3896713776, "random value": 133}
logClient: Received from 3163408206 msg={"nodename": "Remos-04", "NodeID": 3163408206, "random value": 176}
logClient: Received from 3893419003 msg={"nodename": "mcu-03", "NodeID": 3893419003, "random value": 132}
logClient: Received from 572025877 msg={"nodename": "Remos-01", "NodeID": 572025877, "random value": 91}
logClient: Received from 2136600749 msg={"nodename": "Remos-03", "NodeID": 2136600749, "random value": 13}
logClient: Received from 2487009679 msg={"nodename": "Remos-1", "NodeID": 2487009679, "Temp": "25.40C", "Humidity": "44.80%"}
logClient: Received from 2488971084 msg={"nodename": "mcu-t1", "NodeID": 2488971084, "Temp": "25.90C", "Humidity": "42.10%"}
logClient: Received from 983688460 msg={"nodename": "esp-01", "NodeID": 983688460, "random value": 106}
{"nodename": "mcu-04", "NodeID": 3896664414, "random value": 26}
logClient: Received from 3163409689 msg={"nodename": "Remos-02", "NodeID": 3163409689, "random value": 27}
logClient: Received from 3896055851 msg={"nodename": "mcu-02", "NodeID": 3896055851, "random value": 61}

```

No line ending 115200 baud Clear output

Autoscroll

Scenario ที่ 2: Server Node join Mesh Network

เมื่อเปิด Server Node เข้ามาถึงจะเป็นเหมือนรูปข้างล่างนี้ครับ ได้รับ Broadcast Message จาก Server Node และในส่วนของ receivedCallback ก็จะมี logic เพื่อใช้ในการอ่านค่าไว้ใน Topic ที่เป็น logServer มั้ง ถ้ามีค่าบันทึกเก็บไว้เพื่อในการส่งรอบต่อไปจะไม่ Broadcast ไปยังทุก Node ละ แต่จะระบุปลายทางเป็น NodeID ของ Server แทน

```

/dev/cu.SLAB_USBtoUART
Send
0x8 esp@lf1EventCb(): SYSTEM_EVENT_AP_STACONNECTED
0x8 New AP connection incoming
0x8 meshConnectedCb(): we are AP
0x8 findConnection(2752206217); did not find connection
0x8 newConnectionTask():
0x8 newConnectionTask(): adding 2752206217 now= 3889582447
logClient: Received from 3893419003 msg={"nodename":"mcu-03","NodeId":3893419003,"random value":163}
logClient: Received from 572025877 msg={"nodename":"Wemos-01","NodeId":572025877,"random value":1}
logClient: Received from 2136600749 msg={"nodename":"Wemos-03","NodeId":2136600749,"random value":49}
logClient: Received from 2487909679 msg={"nodename":"Wemos-t1","NodeId":2487909679,"Temp":"24.30C","Humidity":"38.10%"}
logClient: Received from 983688460 msg={"nodename":"esp-01","NodeId":983688460,"random value":20}
{"nodename":"mcu-04","NodeId":3896664414,"random value":77}
logClient: Received from 3163409689 msg={"nodename":"Wemos-02","NodeId":3163409689,"random value":30}
logClient: Received from 3896055851 msg={"nodename":"mcu-02","NodeId":3896055851,"random value":170}
logClient: Received from 3896713776 msg={"nodename":"mcu-01","NodeId":3896713776,"random value":128}
logClient: Received from 2488971084 msg={"nodename":"mcu-t1","NodeId":2488971084,"Temp":"24.60C","Humidity":"36.60%"}
logClient: Received from 3163408206 msg={"nodename":"Wemos-04","NodeId":3163408206,"random value":37}
logClient: Received from 3893419003 msg={"nodename":"mcu-03","NodeId":3893419003,"random value":174}
logClient: Received from 572025877 msg={"nodename":"Wemos-01","NodeId":572025877,"random value":8}
logClient: Received from 2136600749 msg={"nodename":"Wemos-03","NodeId":2136600749,"random value":162}
logClient: Received from 2752206217 msg={"topic":"logServer","nodeId":2752206217}
logServer detected!!!
Handled from 2752206217 msg={"topic":"logServer","nodeId":2752206217} {"nodename":"mcu-04","NodeId":3896664414,"random value":81}
 Autoscroll No line ending 115200 baud Clear output

```

หลังจากที่ทุก Node ได้รับ Server NodeID จาก Broadcast Message ที่ส่งมาจาก Server Node แล้ว แทนที่ทุก Node จะส่งข้อมูลออกไปยังทุกๆ Node ก็จะส่งไปยัง Server Node แทนดังจะเห็นได้จากกรุํปข้างล่างที่ได้รับ Broadcast Message เท่ามานะมีเฉพาะข้อความที่มาจากการ Server Node เท่านั้น เพราะตัวอื่นๆส่งไปที่ Server Node หมดแล้ว

```

/dev/cu.SLAB_USBtoUART
Send
logServer detected!!!
Handled from 2752206217 msg={"topic":"logServer","nodeId":2752206217} {"nodename":"mcu-04","NodeId":3896664414,"random value":123}
{"nodename":"mcu-04","NodeId":3896664414,"random value":62}
logClient: Received from 2752206217 msg={"topic":"logServer","nodeId":2752206217}
logServer detected!!!
Handled from 2752206217 msg={"topic":"logServer","nodeId":2752206217} {"nodename":"mcu-04","NodeId":3896664414,"random value":124}
{"nodename":"mcu-04","NodeId":3896664414,"random value":76}
logClient: Received from 2752206217 msg={"topic":"logServer","nodeId":2752206217}
logServer detected!!!
Handled from 2752206217 msg={"topic":"logServer","nodeId":2752206217} {"nodename":"mcu-04","NodeId":3896664414,"random value":150}
{"nodename":"mcu-04","NodeId":3896664414,"random value":55}
logClient: Received from 2752206217 msg={"topic":"logServer","nodeId":2752206217}
logServer detected!!!
Handled from 2752206217 msg={"topic":"logServer","nodeId":2752206217} {"nodename":"mcu-04","NodeId":3896664414,"random value":105}
{"nodename":"mcu-04","NodeId":3896664414,"random value":88}
logClient: Received from 2752206217 msg={"topic":"logServer","nodeId":2752206217}
logServer detected!!!
Handled from 2752206217 msg={"topic":"logServer","nodeId":2752206217} {"nodename":"mcu-04","NodeId":3896664414,"random value":24}
{"nodename":"mcu-04","NodeId":3896664414,"random value":46}
 Autoscroll No line ending 115200 baud Clear output

```

กราโน่เรามาลองดูในฝั่งของ Server กันบ้างว่าเมื่อทุก Node ส่งข้อมูลมาที่ Server Node แล้วข้อมูลที่ได้จะเป็นลักษณะไหนกัน จากรุํปจะเห็นได้ว่า ข้อมูลจะได้รับเหมือนตอนที่เราดูของ Node ใด Node หนึ่งตอนที่ยังไม่มี Server Node เมื่อจากกราโน่ทุก Node ใน Mesh Network ส่งมาที่จุดเดียว ก็จะเข้าใจได้ข้อมูลครบ และตัว Server Node เองก็ยังคง Broadcast Message ออกไปเรื่อยๆเพื่อให้ทุกคนรู้ว่า ฉันอยู่ที่นี่นะ ถ้าจะส่งข้อมูลมาหา ให้ส่งมาที่ไหน

ข่ายความในส่วนของ Server Node กันอีกนิดว่าทำไม่ต้องคีย์ Broadcast Server NodeID อยู่ต่อกันเวลา

- ข้อดีเดียวก็คือ ถ้าในอนาคตต้องมีการเปลี่ยนอุปกรณ์ในส่วนของ Server Node ที่ใช้ไป แต่เมื่อสิ้นอุปกรณ์เปลี่ยน Server Node แล้วอนุรักษ์ ChipID เบื้องต้นไป ลดการ Broadcast Server NodeID และในส่วนของ Client ที่มี receivedCallback ก็จะเช็ค Server NodeID ก็จะทำให้แทนไม่กระทบกับการทำงานเลย เพราะในทุกๆ Node ก็จะรู้ว่าต้องส่งไปที่ Server Node ใหม่หมายความว่า

- ชี้ถ้าใช้หลักการที่เขียนมาสองบทนี้ เราซึ่งสามารถประยุกต์ใช้งานในการทำ Server Node ที่มีมากกว่า 1 ตัวได้เข้าใช้ระบบ Mesh Network ที่ใช้ในการสื่อสารอีกไป Network ภายนอกนั้นเสลียร์อีกขึ้น หรือจะแบ่งเบาภาระของ Server Node เป็นลักษณะของ load balance ก็ซึ่งได้

ตัวอย่างการประยุกต์ใช้งาน

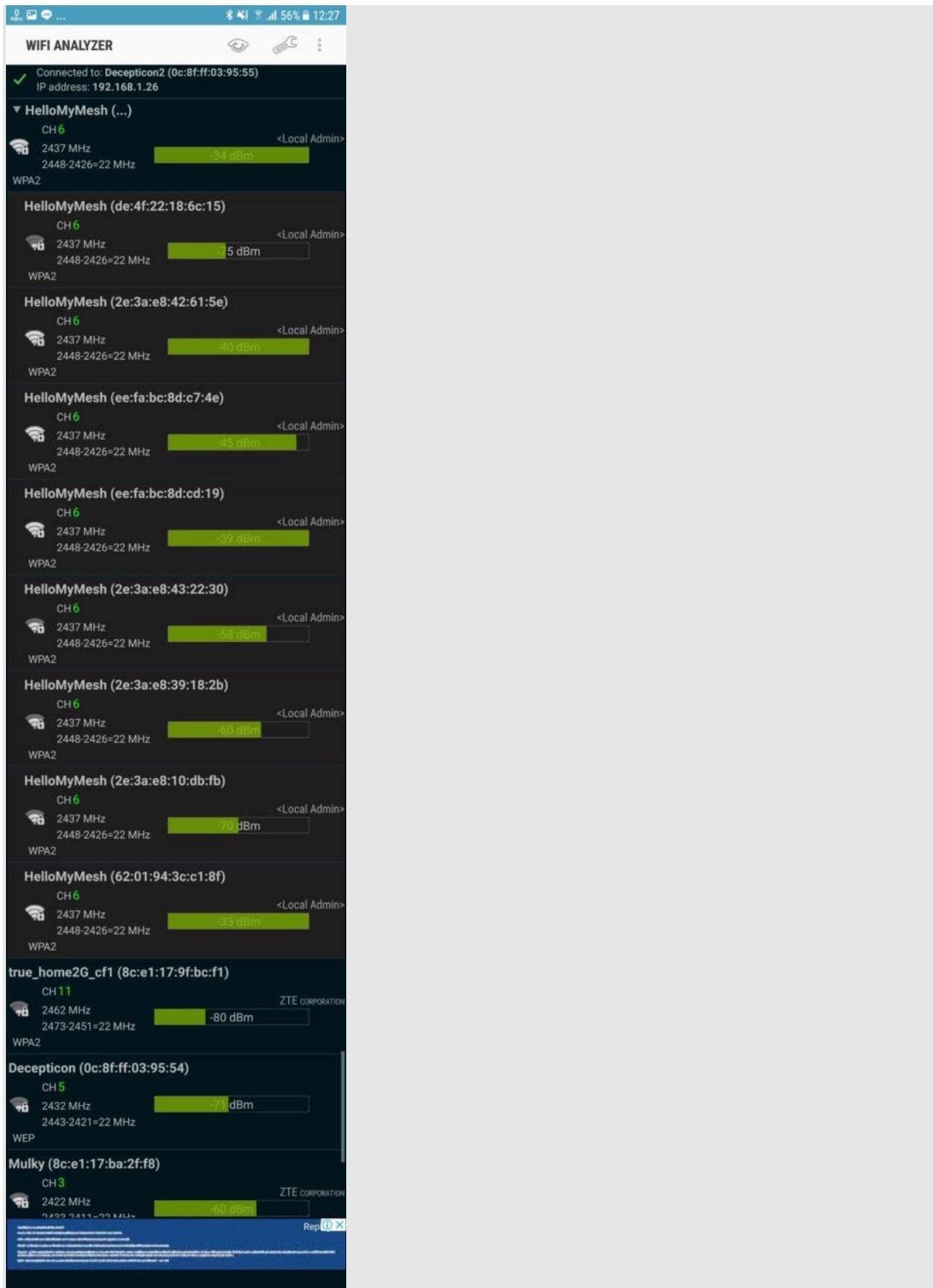
- Read and Forward:** เคสนี้จะเข้าข่ายสำหรับพวกรการส่งข้อมูลเท้า Server อาจจะเป็นการวัดอุณหภูมิ ความชื้น หรือแม้กระทั่งสิ่งที่เสียงบินเวล ต่างๆของโรงงาน แล้วมาเก็บไว้ที่ Server เพื่อที่จะแสดงผล หรือนำข้อมูลไปวิเคราะห์เหตุต่างๆ เหมือนในตัวต่อ logServer/logClient ที่เราเพิ่งทำกันไป เป็นการใช้งานที่ค่อนข้าง One-way จะเป็นส่วนใหญ่ และที่เป็น Application ของ IoT จะเป็นส่วนใหญ่อีกเช่นกันด้วย

- ต่อจากการส่ง message ภายใน Mesh Network สามารถที่จะกำหนดปลายทางผู้รับได้ด้วย NodeID หรือจะส่ง Broadcast แล้วคาย Filter Message เอกซ์ได้ตามทางที่ระบุใน Broadcast Message ผ่านของเราระหว่างๆ จากนั้นอ่านค่าที่ได้มา เปรียบเทียบ logic ที่ต้องการ อาจจะเป็นการสั่งให้ปิดบิ้มหน้าเพื่อรดน้ำด้านไม้ หรือสั่งให้ปิด Siren nokota การกรณีมีการตรวจสอบวันไฟໄไฟ จะแจ้งเป็น zone หรือแจ้งเป็นจุดๆ แล้วแต่หลักการเช่นข้อความที่ส่งไป และหลักการเบร์ยนท์ของ logic ของข้อความที่ได้รับมา จึง Painlessmesh ใช้รูปแบบการส่งแบบ JSON ก็สะดวกยิ่งขึ้น

- Point to Point / Independent:** เมื่อจาก Mesh Network นั้นทำให้ทุก Node นั้นเชื่อมหากันอยู่แล้ว Node ไหน อยากคุยกับ Node ไหนก็ได้ เคสลักษณะนี้ก็อาจจะมี Node ที่ทำงานเป็นลักษณะ Control Panel สามารถที่จะเลือกได้ว่าจะส่งคำสั่งไปให้ node ตัวไหนก็ได้ โดยการอ้างอิงจาก NodeID หรืออาจจะแปลง NodeID ให้เป็นชื่อที่อ่านแล้วเข้าใจง่ายเหมือนที่ผมทำเป็นด้าน

หมายเหตุ

เนื่องจาก ESP8266/ESP32 ใช้ WiFi ย่าน 2.4 GHz อยู่ ซึ่งเกิดขึ้นในโหมดของ Station and AP เพื่อให้ Node ต่างๆ สามารถร่วมกันเป็น Mesh Network และถ้าเปิดโปรแกรมพวาก WiFi Analyzer มาดูจะเห็นชื่อ AP ใน Mesh ของเรารีบันมาเพียงเลย ซึ่งอาจจะไปกวนกับ AP อื่นหรือ AP อื่นอาจจะมากวนเราได้ ยังไงก็ลองพิจารณาการใช้งานให้เหมาะสมดูครับ



วันนี้ก็จะตอน 2 ไว้เพียงเท่านี้ก่อนครับ แล้วตอนหน้าเราจะมาดูกันว่า การที่จะส่งข้อมูลออกไปที่ Network ภายนอกหรือ Internet เพื่อให้สื่อสารกับโลกภายนอกผ่านการ Bridge ระหว่าง Mesh Network และ Node ที่ทำตัวเป็น Gateway ได้อย่างไร

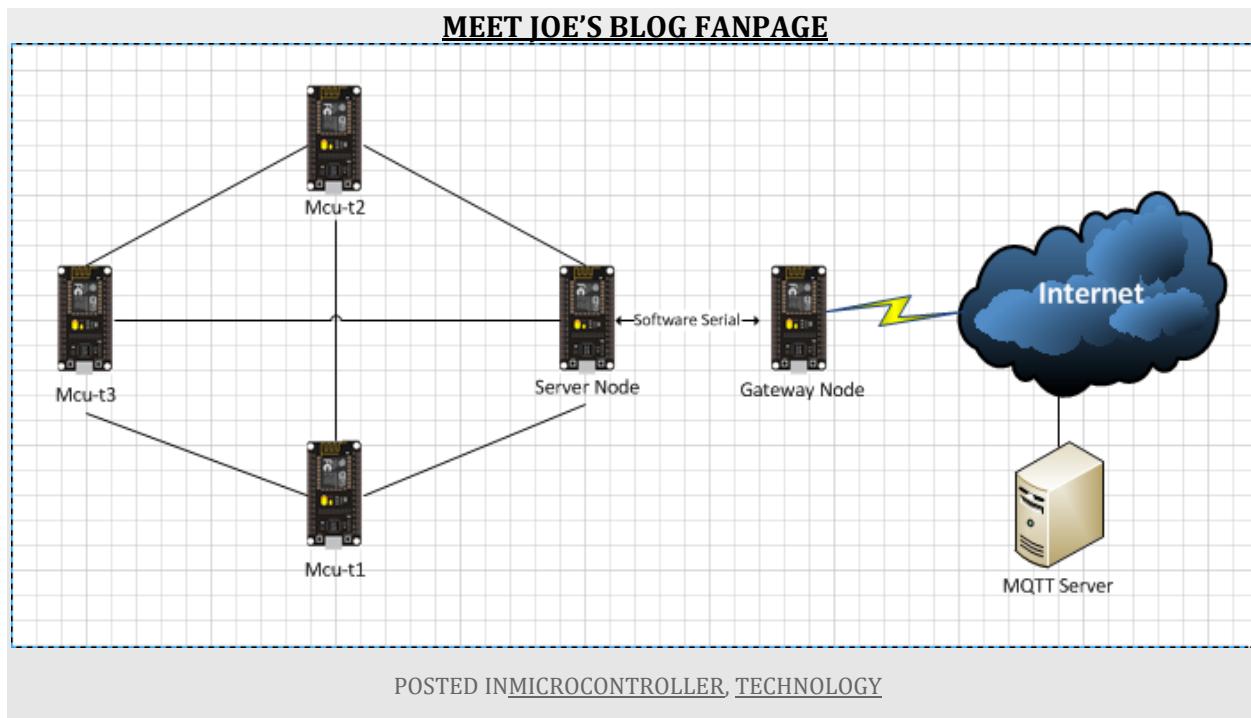
เรื่องล่า้ายตอน

สมัยปี 40 เป็นปีแรกเลยที่ได้รู้จัก ได้ใช้งาน micro controller ตระกูล MCS51 โปรเจคแรกๆ ก็หนี้ไม่พื้นทุ่นเดินตามเส้น ต้องพิยนด้วยภาษา Assembly หน่วยความจำที่มีจำกัด อุปกรณ์ที่ใช้สำหรับ flash ก็แพง อุปกรณ์จะซื้อที่กีด้วยที่ต้องนั่งรถจากศala บ้านหม้อ เที่ยงกับสมัยนี้ที่สังคมสนใจมาก การเข้าถึง content ด้านต่างๆ ก็ง่ายขึ้น สมัยตอนเรียนล่วงเดียวที่พอหาข้างอิงได้คือ โปรเจคจนของรุ่นพี่ แล้วก็หนังสือจากห้องสมุด พอเข้าสู่ชั้นที่มี raspberry pi, arduino และ social media ช่างรู้สึกว่าโลกเปลี่ยนไปเร็วเหลือเกิน เดี๋ยรุ่นใหม่ๆ นี่เรียนรู้กันได้เร็วและเก่งมาก ด้วยอาชีวศึกษาที่มากขึ้น เปลี่ยนสายงานจากกลุ่มมือ implement เอง มาเป็นคุณภาพรวมบริหารงาน ก็ยังสนุกกับการทำความรู้ใหม่ๆ ทดลองนั่นนุ่นนี่ตามโอกาสอำนวย ยิ่งเห็นการเปิดตัวของ maker space และ startup ต่างๆ แล้วก็ต้องสนใจ lab ของมหาลัยดีกดีดื่นๆ ในตอนที่ 2 ก็ฝากกันไว้เท่านี้ก่อนครับ แล้วเดียวกันต่อตอนที่ 3 กัน

ปล.

ถ้ามาจาก Sponsor หลัก (และ Sponsor เดียว) ของทีมผู้พัฒนา Painlessmesh แล้ว ก็จะมีบริษัท Sowillo ที่เป็นผู้ให้เงินทุนสนับสนุนโดยธุรกิจของ Sowillo ก็จะทำเกี่ยวกับเครื่องทำน้ำร้อน โดยใช้พลังงานแสงอาทิตย์ ซึ่งถ้าใครเอา Painlessmesh ไปใช้งานในเชิงพาณิชย์ ถ้าสามารถสนับสนุนทีมพัฒนานี้ได้ ก็จะดีมากๆ เลยก็รับ

ESP Mesh Network 3 – <https://meetjoeblog.com/2018/03/30/esp8266-esp32-mesh-network-painlessmesh-bridge-ep3/>



ESP8266 / ESP32 & Mesh Network ตอนที่ 3: Painlessmesh Bridge

JOE D.S. 30/03/2018 [1 COMMENT](#) ON ESP8266 / ESP32 & MESH NETWORK ตอนที่ 3: PAINLESSMESH BRIDGE

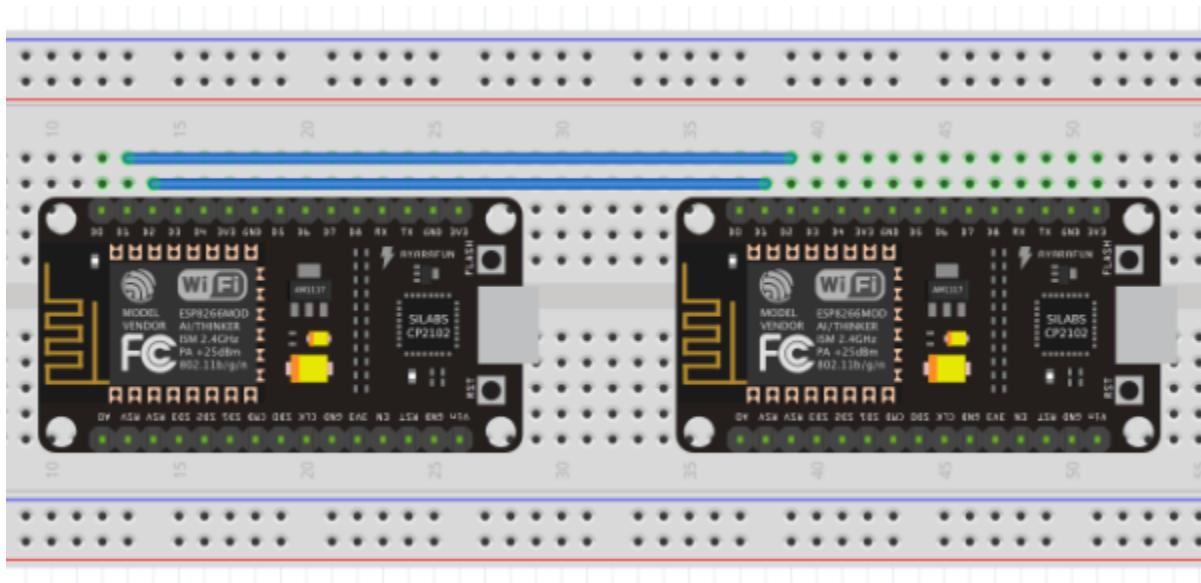
ผ่านมาสองตอนเรียบร้อยกับเรื่องของ ESP8266/ESP32 ในการนำมาทำ Mesh Network ซึ่งข้าวมากๆ ถ้าใครทำตามผ่านมาได้ถึงตอนนี้ไม่มีอะไรยากครับ ตอนนี้จะเป็นเรื่องของการเชื่อมต่อหรือ Bridge ที่ Local Mesh Network ของเราเพื่อออกไปสู่ Internet กันครับ ซึ่งในที่นี้เราจะเพิ่ม Bridge Node ของเราเข้าไป เพื่อต่อเข้ากับ Server Node แต่ก่อนอื่น เรามาดูเรื่องการส่งข้อมูลผ่าน Serial Port ของ Nodemcu สองตัวกันครับ เพราะสามารถประยุกต์ใช้งานอย่างอื่นได้อีก

ตัวอย่างเพื่อประยุกต์ใช้ในการส่งผ่านข้อมูลของ Nodemcu สองตัวผ่าน Serial Port

Redundant (Active-Standby): อย่างที่ทราบกันดีว่าพลังในการประมวลผลและหน่วยความจำของ ESP8266 นั้นมีจำกัด ซึ่งเราสามารถที่จะให้ Nodemcu ของเราทำงานลับกันได้ กรณีตัวใดตัวหนึ่งตายนะไป อีกตัวที่เหลือจะมาทำงานแทนแล้วก็สั่งสัญญาณไป reset Nodemcu ตัวที่ตายไป พอดีตอนปักกี้ขึ้นมาใหม่ก็มาอยู่ใน Standby Mode ค่อยฟังสัญญาณผ่านทาง Serial Port จะ ซึ่งเราสามารถทำ Heart Beat Check ผ่านทาง Serial Port ได้ โดยที่ Nodemcu สามารถทำงานแทนกันไปแทนกันมาได้ตลอด เพราะอย่างที่ทราบกันดีอุปกรณ์ด้านไฟไหม้ก็มีโอกาสที่จะ hang ได้ ถ้า reboot แล้วกลับมาทำงานตามปกติได้ แต่ถ้าไม่ปกติก็อาจทำให้งานที่ทำอยู่เสียหายได้ นี่เป็นสถานที่ที่ต้องเดินทางที่คงลำบากอีก บางคราวอาจจะคิดว่าข่าวว่า ที่นี่ก็ให้มันทำงานสองตัวด้วยกันได้ แต่ถ้าทำงานพร้อมๆ กันแล้ว อันนี้อาจต้องมองแล้วแต่เคสครับ เพราะบางที่เราอาจจะไม่ได้ใช้งานเป็น Sensor Node อย่างเดียว

Communication Channel: เกสี่จะเป็นเกสที่เราจะใช้กันในตอนที่ 3 นี้ครับ ซึ่งบางโปรเจกท์ที่ทำกับ Arduino หรือ MCU บางตัวก่อนหน้านี้ที่ไม่มี WiFi รองรับแล้ววันนึงอยากจะส่งค่าอุณหภูมิเพื่อ monitor ผ่านทาง Internet ก็สามารถทำได้โดยที่ MCU เดิม ส่งค่าผ่านมาทาง Serial Port และใช้ ESP8266 รับค่า จากนั้นก็ส่งข้อมูลผ่าน WiFi

น่าจะพอเห็นภาพการใช้งานเพื่อสื่อสารระหว่าง Nodemcu 2 ตัวผ่าน Serial Port กันแล้ว เรา มาดูในส่วนของ Code กันบ้างซึ่งเนื่องจากข้อจำกัดของ Nodemcu ที่มี Serial Port แค่คู่เดียว และเราใช้มันในการ Flash Code ของรวมถึง Monitor ผ่าน Serial Monitor ใน Arduino IDE อีก แต่ว่าโชคดีที่ ESP8266 นั้นสามารถใช้งาน Software Serial เพื่อใช้ i/o ที่มีอยู่ในการทำงานเป็น Serial Port แทนได้ เพื่อให้เห็นภาพเรามาเริ่มที่ตัวอย่างนี้กันครับ โดยใช้ mcu2 ตัว



เริ่มจาก Copy Code ค้าง่ายนี้แล้วก็ Flash ลง Nodemcu หึส่องดูแลบ ซึ่งใน Code หมกหนดให้ D1 เป็น RX ล่วน D2 เป็น TX ละนั้นการต่อสายจะเป็นลักษณะไขว้กันระหว่าง RX<->TX ตามรูปด้านบน

```
#include <SoftwareSerial.h>

SoftwareSerial swSerial(D1, D2, false, 128); //Define hardware connections RX, TX

int i = 0;

String DataString;

void setup() {

    pinMode(D1, INPUT);

    pinMode(D2, OUTPUT);

    Serial.begin(115200); //Initialize hardware serial with baudrate of 115200

    swSerial.begin(115200); //Initialize software serial with baudrate of 115200
```

```

Serial.println("\nSoftware serial test started");

}

void loop() {

swSerial.print("Hello: " + String(i));

i++;

while (swSerial.available() > 0)

{

char c = swSerial.read(); //gets one byte from serial buffer

DataString += c; //makes the string DataString

}

if (DataString != "")

{

Serial.println(DataString);

}

DataString = "";

delay(1000);

}

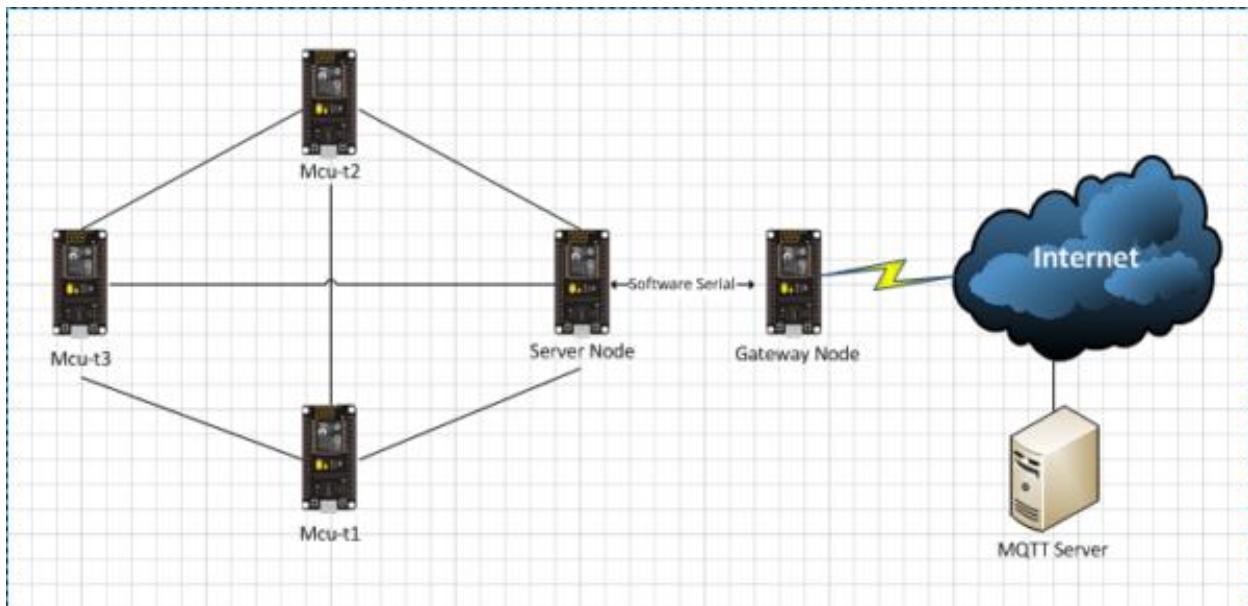
```

เมื่อเปิด Serial Monitor มาดูจะเห็นว่าค่าที่ print ออกมาผ่านทาง Serial Port(HW Serial) นั้นคือค่าที่เราอ่านได้จาก swSerial (Software Serial) ซึ่งวนลูปเพิ่มค่า i ไปเรื่อยๆ โดยเป็นค่าที่รับมาจาก Nodemcu อีกด้วย ค่าที่ได้รับมานี้ก็ขึ้นอยู่กับเราว่าจะเอาไปทำอะไรต่อ

```
Hello: 108
Hello: 109
Hello: 110
Hello: 111
Hello: 112
Hello: 113
Hello: 114
Hello: 115
Hello: 116
Hello: 117
Hello: 118
Hello: 119
Hello: 120
Hello: 121
Hello: 122
Hello: 123
Hello: 124
```

Autoscroll Both NL & CR 115200 baud Clear output

มาต่อ กันที่ เกสของตอนนี้ ของเรากัน คือ กว่า นั้น ก็คือ การ Bridge ระหว่าง Local Mesh Network ของเรานั้นที่ ส่งข้อมูลออกไปยังโลกภายนอก กัน โดยที่ Scenario ที่ เราจะ จำลองขึ้นมาจะใช้ Nodemcu ทั้งหมด 5 ตัว ดังรูปด้านล่างนี้



- mcu-t1 ถึง mcu-t3 เป็น Mesh Node ที่ส่งค่า Temp/Humidity ไปที่ Server Node
- Server Node ที่รับค่าจาก Node อื่นๆ มา จะส่งค่าไปที่ Gateway Node ซึ่งทำการ Bridge ข้อมูลระหว่าง Mesh Network กับ WiFi Network และส่งค่าออกผ่านทาง internet
- Gateway Node ที่รับข้อมูลจาก Server Node ผ่านทาง software serial จะส่งค่าไปที่ MQTT Server

*MQTT ก็จะ ไร้ โดยละเอียด ไว้ ติดตาม กันต่อตอนที่ 5 แต่คร่าวๆ คือ messaging protocol ตัวหนึ่ง ซึ่ง ใช้รับส่งข้อความผ่าน IP Network ารมณ์คล้ายๆ Instant Message ซึ่งการรับส่งข้อความ จะใช้ กิจกรรมของ การ Public(ส่ง)-Subscribe(รับ) กับ Topic ที่เราต้องการ ซึ่ง MQTT จะใช้มากในงานของ iot เพราะ ไม่ซับซ้อน กิน resource น้อยและค่อนข้าง realtime

ในส่วนของ Code ก็ประกอบด้วย 3 ส่วนคือยังกัน Code ชุดแรกสำหรับ NodeMCU ที่ต่ออัพกับ DHT22 เพื่อส่งค่า Temp/Humidity ผ่านทาง Mesh Network ไปให้ Server

Code ชุดที่หนึ่ง สำหรับ mcu-t1 ถึง mcu-t3 ในการอ่านค่า Temp/Humidity

```
#include "painlessMesh.h"

#include "DHT.h"

#define DHTPIN D4

#define DHTTYPE DHT22

#define MESH_PREFIX "HelloMyMesh"

#define MESH_PASSWORD "hellomymeshnetwork"

#define MESH_PORT 5555

DHT dht(DHTPIN, DHTTYPE);

void receivedCallback( uint32_t from, String &msg );

painlessMesh mesh;

size_t logServerId = 0;

// Send message to the logServer every 5 seconds

Task myLoggingTask(5000, TASK_FOREVER, []() {

    float h = dht.readHumidity();

    float t = dht.readTemperature();

    DynamicJsonBuffer jsonBuffer;

    JsonObject& msg = jsonBuffer.createObject();

    msg["nodename"] = "mcu-t3"; //change for identify for the node that send data mcu-t1 to mcu-t3

    msg["NodeID"] = mesh.getNodeId();

    msg["Temp"] = String(t)+"C";

    msg["Humidity"] = String(h)+"%";

    String str;

    msg.printTo(str);
});
```

```
if (logServerId == 0) // If we don't know the logServer yet
    mesh.sendBroadcast(str);

else
    mesh.sendSingle(logServerId, str);

// log to serial
msg.printTo(Serial);
Serial.printf("\n");
});

void setup() {
    Serial.begin(115200);
    Serial.println("Begin DHT22 Mesh Network test!");
    dht.begin();
    mesh.setDebugMsgTypes( ERROR | STARTUP | CONNECTION ); // set before init() so that you can see startup messages
    mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT, STA_AP, AUTH_WPA2_PSK, 6 );
    mesh.onReceive(&receivedCallback);
    // Add the task to the mesh scheduler
    mesh.scheduler.addTask(myLoggingTask);
    myLoggingTask.enable();
}

void loop() {
    // put your main code here, to run repeatedly:
    mesh.update();
}

void receivedCallback(uint32_t from, String &msg) {
    Serial.printf("logClient: Received from %u msg=%s\n", from, msg.c_str());
    // Saving logServer
    DynamicJsonBuffer jsonBuffer;
```

```

JsonObject& root = jsonBuffer.parseObject(msg);

if (root.containsKey("topic")) {

    if (String("logServer").equals(root["topic"].as<String>())) {

        // check for on: true or false

        logServerId = root["nodeId"];

        Serial.printf("logServer detected!!!\n");

    }

    Serial.printf("Handled from %u msg=%s\n", from, msg.c_str());

}

}

```

Code ชุดที่สองสำหรับ Nodemcu ที่ทำหน้าที่เป็น Server Node เพื่อรับค่า Temp/Humidity ผ่านทาง Mesh Network จากนั้นส่งต่ออีกหอดหนึ่งไปยัง Gateway Node โดยใช้การสื่อสารผ่านทาง Serial Port ซึ่งจุดที่จะเพิ่มก็คือการเรียกและกำหนดขาที่จะใช้งาน Software Serial และการส่งข้อมูลผ่านทาง Software Serial Port ใน receivedCallback Code ที่ใช้สำหรับ Server Node

```

#include "painlessMesh.h"

#include <SoftwareSerial.h>

#define MESH_PREFIX "HelloMyMesh"

#define MESH_PASSWORD "hellomymeshnetwork"

#define MESH_PORT 5555

SoftwareSerial swSerial(D1, D2, false, 128); //Define hardware connections RX, TX

painlessMesh mesh;

// Send my ID every 10 seconds to inform others

Task logServerTask(10000, TASK_FOREVER, []() {

    DynamicJsonBuffer jsonBuffer;

    JsonObject& msg = jsonBuffer.createObject();

    msg["topic"] = "logServer";

    msg["nodeId"] = mesh.getNodeId();
}

```

```

String str;

msg.printTo(str);

mesh.sendBroadcast(str);

// log to serial

msg.printTo(Serial);

Serial.printf("\n");

});

void setup() {

pinMode(D1, INPUT);

pinMode(D2, OUTPUT);

Serial.begin(115200); //Initialize hardware serial with baudrate of 115200

swSerial.begin(115200); //Initialize software serial with baudrate of 115200

mesh.setDebugMsgTypes( ERROR | CONNECTION | S_TIME );

mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT, STA_AP, AUTH_WPA2_PSK, 6 );

mesh.onReceive(&receivedCallback);

mesh.onNewConnection([](size_t nodeId) {

Serial.printf("New Connection %u\n", nodeId);

});

mesh.onDroppedConnection([](size_t nodeId) {

Serial.printf("Dropped Connection %u\n", nodeId);

});

// Add the task to the mesh scheduler

mesh.scheduler.addTask(logServerTask);

logServerTask.enable();

}

void loop() {

mesh.update();

```

```

}

void receivedCallback( uint32_t from, String &msg ) {

Serial.printf("logServer: Received from %u msg=%s\n", from, msg.c_str());

swSerial.print(msg.c_str()); // Print received data in Mesh Network to Software Serial Port

}

```

Code ชุดที่สามสำหรับ Nodemcu ที่ทำหน้าที่เป็น Gateway Node เพื่อรับค่า Temp/Humidity ผ่านทาง Serial Port จาก Server Node จากนั้นก็ส่งค่าที่ได้รับไปยัง MQTT Server ซึ่งในความเป็นจริงแล้วปลาบทางของ Gateway ในการส่งผ่านข้อมูลไปจะเป็นอะไรก็ได้ อาจจะเป็น Webservice, Blynk, Firebase หรืออะไรก็ตาม ใช้หลักการเดียวกันเลย Code ที่ใช้สำหรับ Gateway Node เพื่อส่งข้อมูลไปยัง MQTT server ซึ่งถ้าใครอยากระดลลงในส่วนนี้ก็ติดตามจากตอนที่ 5 ในเรื่องของการตั้ง MQTT Server บน Google Cloud หรืออาจจะใช้บริการ [Cloud MQTT](#) ซึ่งมีส่วนให้บริการฟรีๆได้ครับ

```

#include <SoftwareSerial.h>

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

const char* ssid = "xxx"; //wifi ssid

const char* password = "xxx"; //wifi passwd

IPAddress mqtt_server(xx, xx, xx, xx); //ip of mqtt server

WiFiClient espClient;

long lastMsg = 0;

char msg[100];

int value = 0;

String DataString;

SoftwareSerial swSerial(D1, D2, false, 256); //Define hardware connections RX, TX

void callback(char* topic, byte* payload, unsigned int length) {

Serial.print("Message arrived [");

Serial.print(topic);

Serial.print("] ");

for (int i = 0; i < length; i++) {

```

```
Serial.print((char)payload[i]);  
}  
  
Serial.println();  
}  
  
PubSubClient client(mqtt_server, 1883, callback, espClient);  
  
void setup() {  
  
pinMode(D1, INPUT);  
  
pinMode(D2, OUTPUT);  
  
Serial.begin(115200); //Initialize hardware serial with baudrate of 115200  
  
swSerial.begin(115200); //Initialize software serial with baudrate of 115200  
  
Serial.println("Bridget Node Gateway to MQTT");  
  
Serial.println();  
  
Serial.print("connecting to ");  
  
Serial.println(ssid);  
  
WiFi.begin(ssid, password);  
  
while (WiFi.status() != WL_CONNECTED) {  
  
delay(1000);  
  
Serial.print(".");  
}  
  
Serial.println("");  
  
Serial.println("WiFi connected");  
  
Serial.println("IP address: ");  
  
Serial.println(WiFi.localIP());  
  
client.connect("MeshGateway", "xxx", "xxx"); //MeshGateway is node name, change xxx to user/passs of your server  
  
client.setCallback(callback);  
  
client.subscribe("command");  
}
```

```

void loop() {

    while (swSerial.available() > 0)

    {

        char c = swSerial.read(); //gets one byte from serial buffer

        DataString += c; //makes the string DataString

    }

    if (DataString != "")

    {

        Serial.println(DataString);

        if (!client.connected()) {

            reconnect();

        }

        client.loop();

        DataString.toCharArray(msg, 100);

        client.publish("envMesh", msg); //publish message to mqtt server with topic envMesh

    }

    DataString = "";

    delay(1000);

}

void reconnect() {

    // Loop until we're reconnected

    while (!client.connected()) {

        Serial.print("Attempting MQTT connection...");

        // Attempt to connect

        if (client.connect("MeshGateway", "xxx", "xxx")) { //userbame of your mqtt server

            Serial.println("connected");

            // Once connected, publish an announcement...

```

```

client.publish("outTopic", "hello world");

// ... and resubscribe

client.subscribe("inTopic");

} else {

Serial.print("failed, rc=");

Serial.print(client.state());

Serial.println(" try again in 5 seconds");

// Wait 5 seconds before retrying

delay(10000);

}

}

}

```

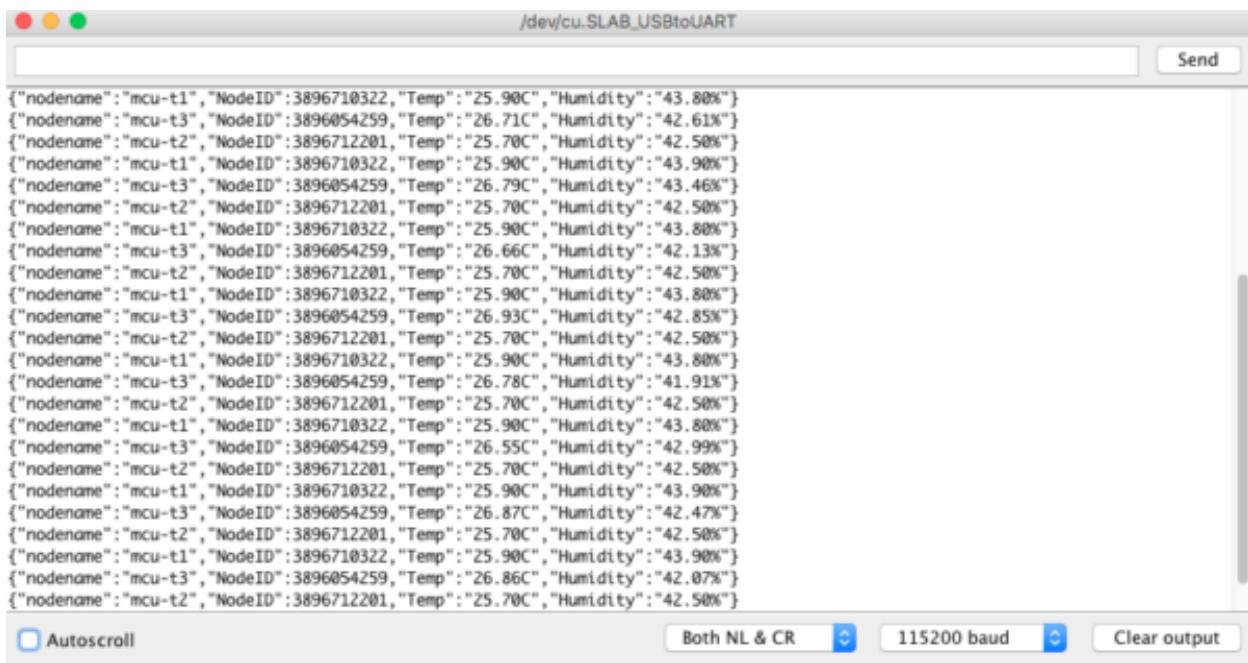
เมื่อเปิด Nodemcu ทั้ง 5 ตัวขึ้นมา เหตุการณ์จากตอนที่ 2 ก็จะปรากฏขึ้นดัง Step ต่อไปนี้

- mcu-t1 ถึง mcu-t3 จะ join เข้า mesh network และส่งค่า broadcast ไปยังทุก Node และคงอยู่ใน Mesh Network นี้ ถ้า Server Node มีค่าไม่เท่ากับ Server NodeID และส่งจักน้ำก็ส่งเป็นแบบบัญญาช่างไปที่ Server Node ละ
- Step นี้จะเพิ่เติมมาถัดไป ที่ Server Node เมื่อได้รับค่าที่ส่งมาภายใน Mesh Network ก็จะทำการ Forward ต่อไปยัง Gateway Node ผ่านทาง software serial port ได้อะไรมา ก็ส่งไปยังหน้า ให้ดู code ในส่วนของ receivedCallback ที่จะทำการ print ค่าอุณหภูมิ software serial

```

swSerial.print(msg.c_str());

```
- ที่ Gateway Node เมื่อได้รับค่าที่ส่งมาจาก software serial port ก็จะทำการส่งต่อค่าไปยัง mqtt server



The screenshot shows a terminal window titled '/dev/cu.SLAB_USBtoUART' with the following content:

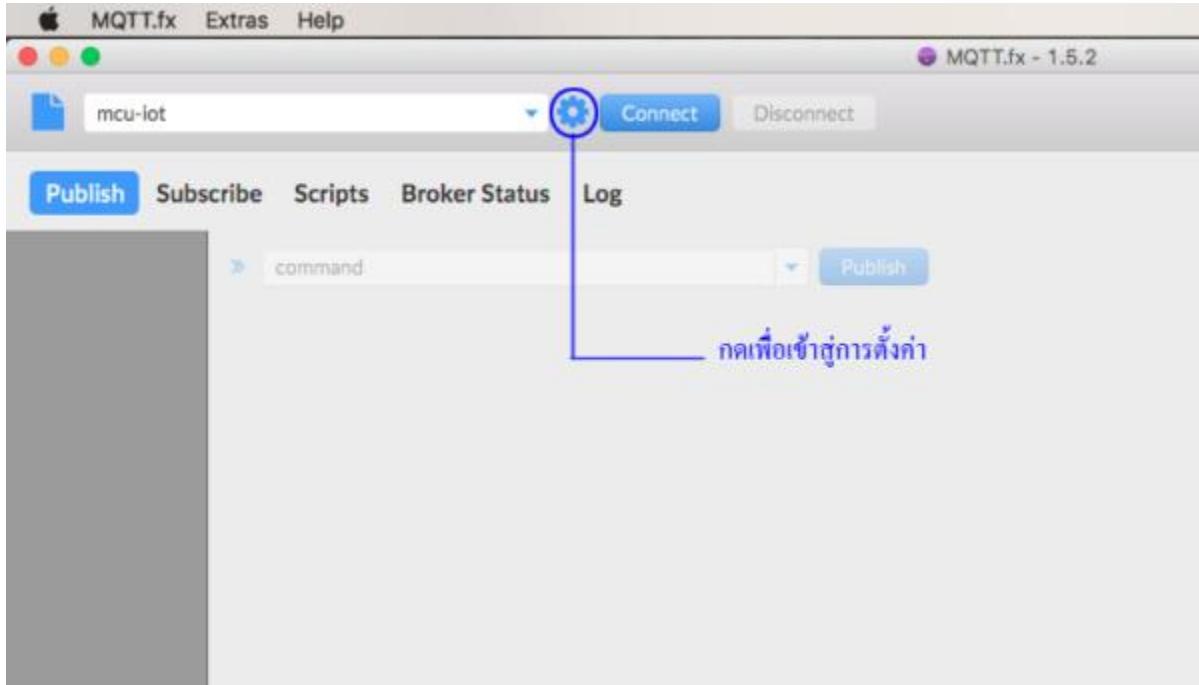
```
{"nodename": "mcu-t1", "NodeID": 3896710322, "Temp": "25.90C", "Humidity": "43.80%"} {"nodename": "mcu-t3", "NodeID": 3896054259, "Temp": "26.71C", "Humidity": "42.61%"} {"nodename": "mcu-t2", "NodeID": 3896712201, "Temp": "25.70C", "Humidity": "42.50%"} {"nodename": "mcu-t1", "NodeID": 3896710322, "Temp": "25.90C", "Humidity": "43.90%"} {"nodename": "mcu-t3", "NodeID": 3896054259, "Temp": "26.79C", "Humidity": "43.46%"} {"nodename": "mcu-t2", "NodeID": 3896712201, "Temp": "25.70C", "Humidity": "42.50%"} {"nodename": "mcu-t1", "NodeID": 3896710322, "Temp": "25.90C", "Humidity": "43.80%"} {"nodename": "mcu-t3", "NodeID": 3896054259, "Temp": "26.66C", "Humidity": "42.13%"} {"nodename": "mcu-t2", "NodeID": 3896712201, "Temp": "25.70C", "Humidity": "42.50%"} {"nodename": "mcu-t1", "NodeID": 3896710322, "Temp": "25.90C", "Humidity": "43.80%"} {"nodename": "mcu-t3", "NodeID": 3896054259, "Temp": "26.93C", "Humidity": "42.85%"} {"nodename": "mcu-t2", "NodeID": 3896712201, "Temp": "25.70C", "Humidity": "42.50%"} {"nodename": "mcu-t1", "NodeID": 3896710322, "Temp": "25.90C", "Humidity": "43.80%"} {"nodename": "mcu-t3", "NodeID": 3896054259, "Temp": "26.78C", "Humidity": "41.91%"} {"nodename": "mcu-t2", "NodeID": 3896712201, "Temp": "25.70C", "Humidity": "42.50%"} {"nodename": "mcu-t1", "NodeID": 3896710322, "Temp": "25.90C", "Humidity": "43.80%"} {"nodename": "mcu-t3", "NodeID": 3896054259, "Temp": "26.55C", "Humidity": "42.99%"} {"nodename": "mcu-t2", "NodeID": 3896712201, "Temp": "25.70C", "Humidity": "42.50%"} {"nodename": "mcu-t1", "NodeID": 3896710322, "Temp": "25.90C", "Humidity": "43.90%"} {"nodename": "mcu-t3", "NodeID": 3896054259, "Temp": "26.87C", "Humidity": "42.47%"} {"nodename": "mcu-t2", "NodeID": 3896712201, "Temp": "25.70C", "Humidity": "42.50%"} {"nodename": "mcu-t1", "NodeID": 3896710322, "Temp": "25.90C", "Humidity": "43.90%"} {"nodename": "mcu-t3", "NodeID": 3896054259, "Temp": "26.86C", "Humidity": "42.07%"} {"nodename": "mcu-t2", "NodeID": 3896712201, "Temp": "25.70C", "Humidity": "42.50%"} 
```

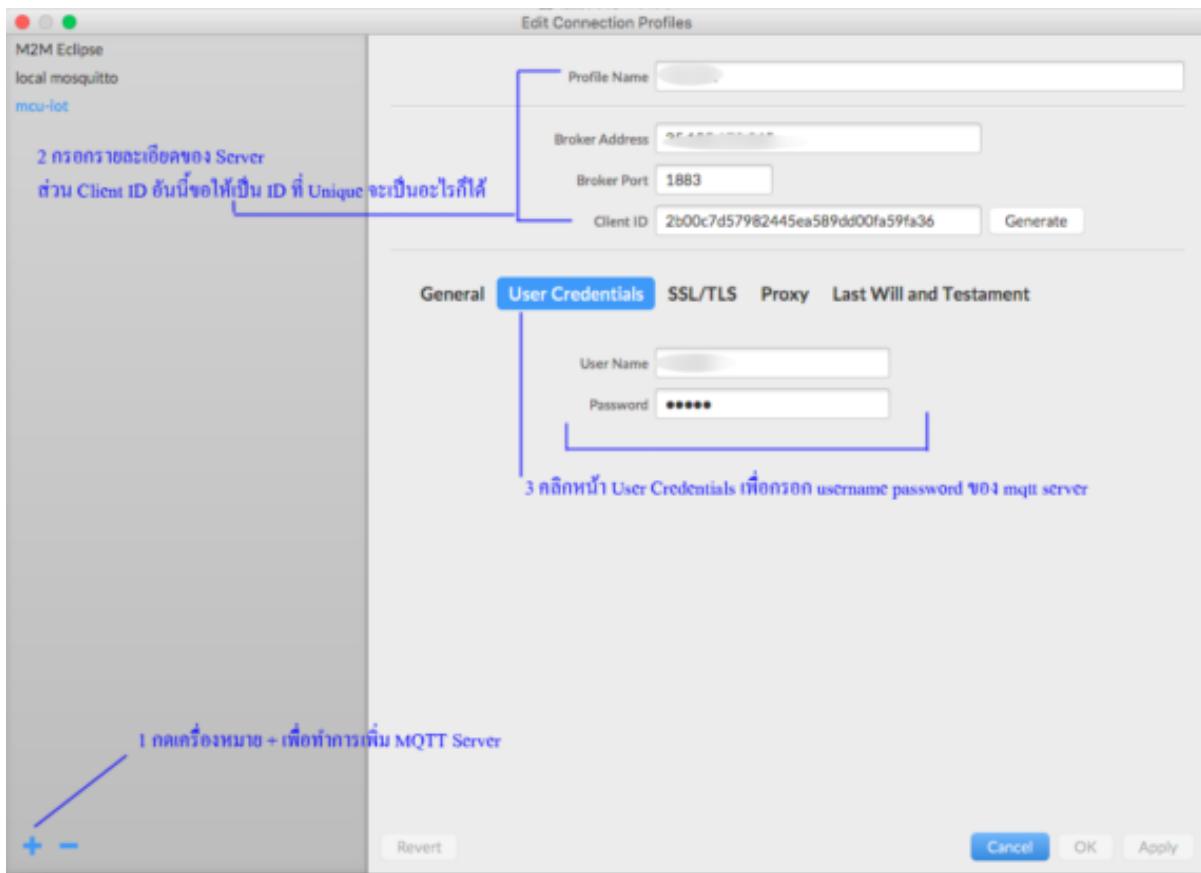
Below the terminal window, there are several buttons: 'Autoscroll' (unchecked), 'Both NL & CR' (selected), '115200 baud' (selected), and 'Clear output'.

ข้อมูลที่รับเข้ามายัง Software Serial Port จาก Gateway Node

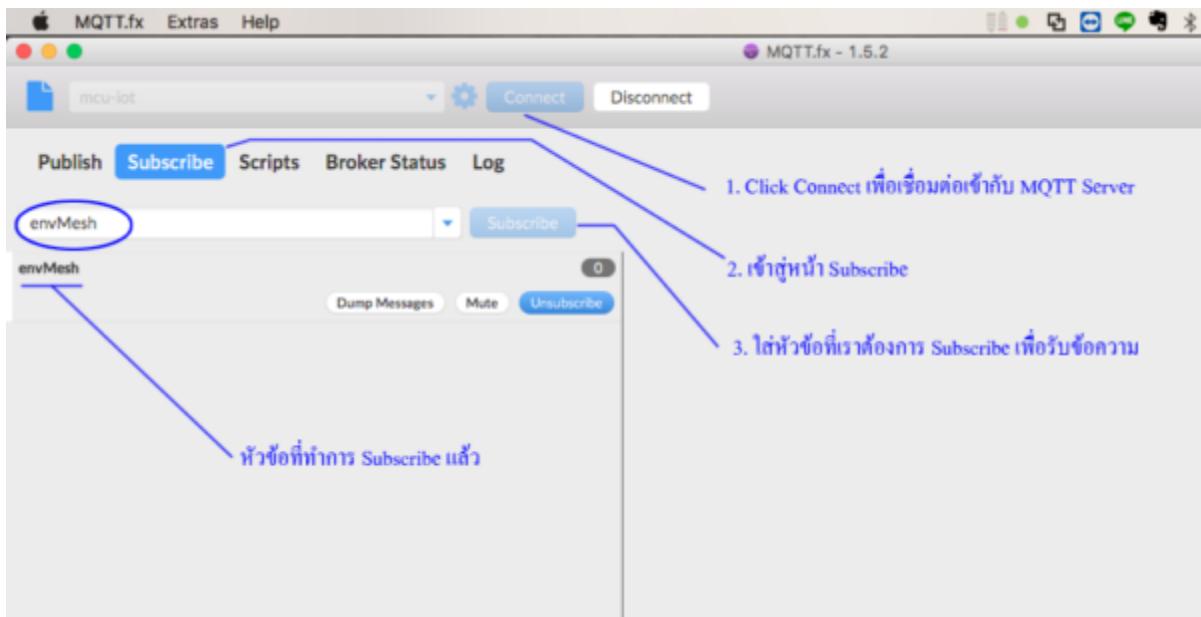
ส่วนผลลัพธ์ที่อ่านค่าได้จาก MQTT Server นั้นผมใช้โปรแกรม [MQTT.fx](#) ซึ่งเป็น Client ที่มีให้ใช้งานได้หลากหลาย platform เลยเนื่องจากว่าเป็น Java based ก็สามารถโหลดได้จากเวปของ [MQTT.fx](#) ได้เลยครับ

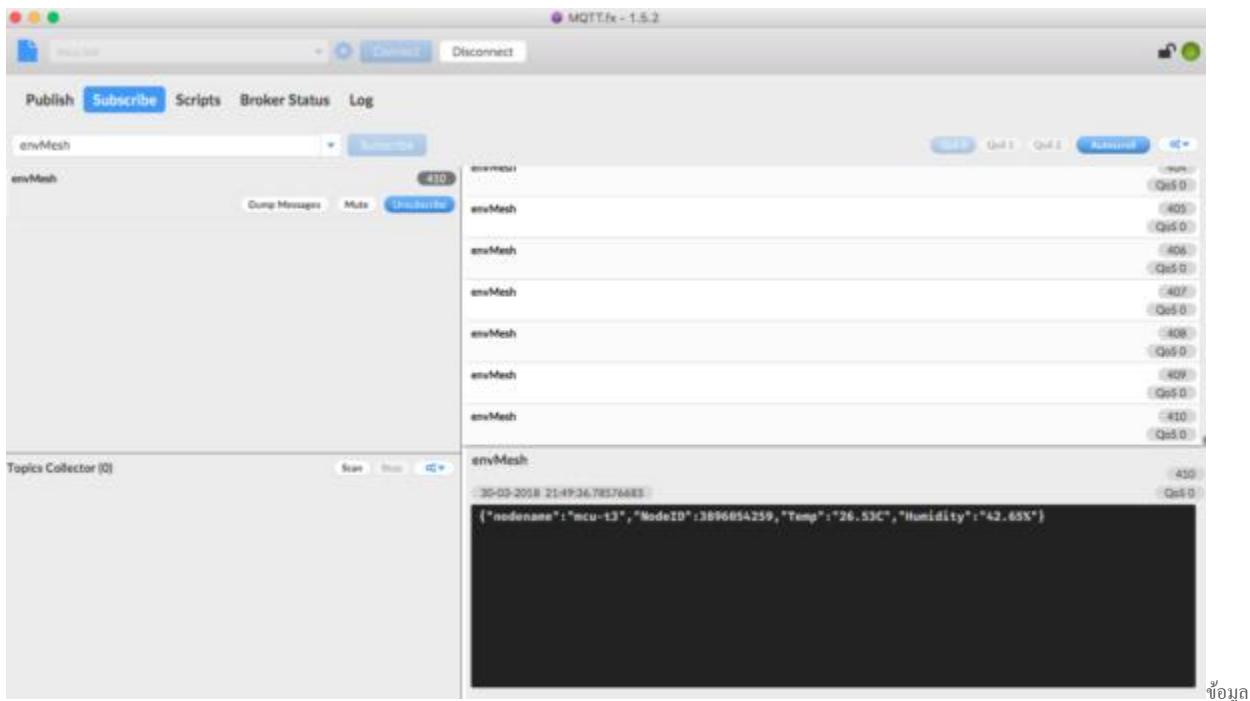
การใช้งานก็เริ่มจากการตั้งค่ากันก่อน ที่หน้าแรกให้คลิกที่รูปเพื่อ เพื่อเข้าสู่หน้าของการตั้งค่า แล้วก็ตั้งค่าตาม MQTT Server ของแต่ละท่านที่ใช้กันอยู่เลยครับ หรือจะใช้บริการฟรีของ [Cloud MQTT](#) ก็ได้



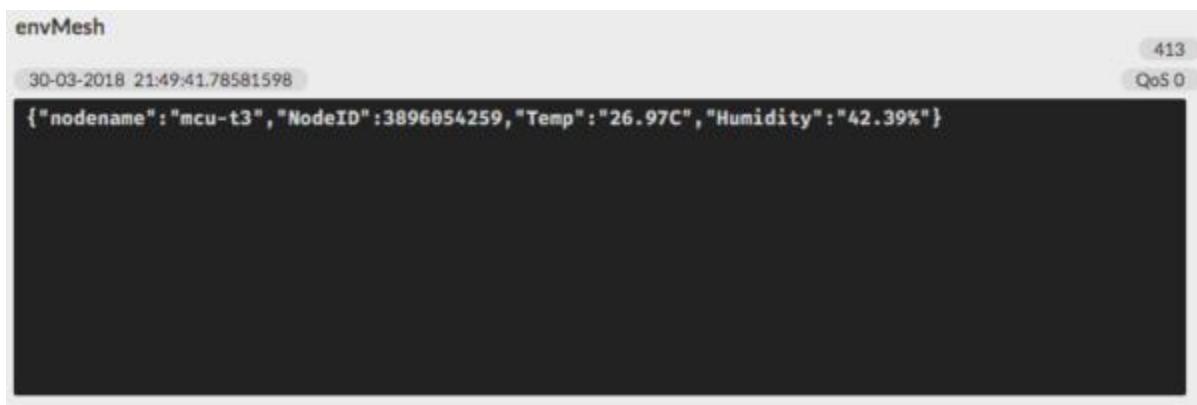


ทำการ Subscribe หัวข้อ envMesh ตาม Code ที่อยู่ใน Gateway Node





ที่ได้รับมาจะแสดงอยู่ที่ฟีน์ที่สำคัญ พร้อมกับ Timestamp



จากกฎที่จะเห็นข้อความที่อ่านได้จาก MQTT Server ก็จะเป็นรูปแบบ JSON โดยส่งมาเป็นทอดๆ จาก Local Mesh Network -> Server Node ภายใน Mesh Network -> Gateway Node ผ่านการ Bridge โดยใช้ Software Serial จากนั้นก็ส่งต่อไปที่ MQTT Sever ผ่าน WiFi/Internet -> แล้วเราที่ใช้โปรแกรม MQTT.fx เข้าไป Subscribe เพื่ออ่านค่าที่ส่งมาให้หัวขอ envMesh

จริงๆ สำหรับบันนี้ ใจความหลักเลยจะเป็นเรื่องของการใช้งาน Software Serial ที่ใช้ในการสื่อสารกันระหว่าง Nodemcu 2 ตัว ส่วนเรื่อง MQTT Server นั้นก็เป็นการทึ่งปมเพื่อให้ไปค้นกันด้วย หรือจะรอถึงตอนที่ 5 ที่ได้กับการ Setup MQTT Server ขึ้นมาใช้งานเองบน Google Cloud และพับกันใหม่ตอนที่ 4 ซึ่งเราจะ Bridge ข้อมูลจาก Local Mesh Network ของเราผ่าน LORA Network กันครับ

เรื่องเล่าท้ายตอน

หลังจากห่างหายการใช้งาน mcs51 มาตั้งแต่เรียนจบปี 42 นี่ก็เกือบๆ สิบปีละ เพิ่งมาสองสามปีหลังที่เริ่มกลับมาสนใจหัวข้อนี้อีกครั้งกับพวก microcontroller ใหม่ ก็รวมไปถึงโครงการของ Raspberry Pi และ Arduino ด้วย ซึ่งถ้าอ่านจากอรรถเมืองจะเห็นคำว่า “ไม่ใช่Arduino” มากกว่า “Arduino” ที่จริงๆแล้ว Arduino เป็น Microcontroller แต่ Raspberry Pi เป็น Computer ขนาดเล็ก โดยความสามารถที่เป็นเหมือนคำรามโลกแตก แต่มีคำตอบนึงที่ตอบไว้ค่อนข้างดี

“ถ้าคุณสามารถเขียนสิ่งที่คุณต้องการจะทำให้ได้โดยใช้คำว่า and/ และ ไม่เกิน 2 ตัว ใช้ Arduino แต่ถ้ามากกว่านั้นไป Raspberry Pi”

ตามว่าถูกต้องทั้งหมดมั้ย ก็ไม่จะที่เดียว แต่ก็พอเป็นแนวทางให้เข้าใจได้ว่า อย่าไปใช้งาน Arduino มันหนักนัก เพราะพลังในการประมวลผล รวมถึงหน่วยความจำมันมีจำกัด แต่ถ้าเป็นงานไม่ซับซ้อน งานลูปทำไปเรื่อยๆ แนะนำว่า Arduino นั้นตอบโจทย์ได้ดีที่เดียว

- ESP Mesh Network 4 – <https://meetjoeblog.com/2018/04/25/esp8266-esp32-painlessmesh-bridge-with-lora-ep4/>

MEET JOE'S BLOG FANPAGE



LoRa® Alliance

Wide Area Networks for IoT

POSTED IN MICROCONTROLLER, TECHNOLOGY

ESP8266 / ESP32 & Mesh Network ตอนที่ 4: Painlessmesh Bridge with LoRa

JOE D.S. 25/04/2018 7 COMMENTSON ESP8266 / ESP32 & MESH NETWORK ตอนที่ 4: PAINLESSMESH BRIDGE WITH LORA

บทความในตอนนี้ก็จะใช้เวลาเขียนนานหน่อย เนื่องด้วยงานที่ทำ แล้วก็อุปกรณ์เจ้ากรรม Heltec ESP32+OLED 915MHz ที่สั่งมาดัน เลยไปทิ้งๆ ก็เลยวันช่วงไวร์อุปกรณ์มาให้ครบเพื่อทดสอบ ซึ่งจากประสบการณ์ที่สั่งของจากจีนมาทดลอง กรณีที่ไม่ได้สั่งแบบน้ำเข้ามาทดสอบ แบบเป็นทางการ ก็ควรสั่งเข้ามาอย่างน้อย 2-3 ตัวด้วยกัน เพื่อตัวหนึ่งใช้งานไม่ได้จะได้ทดสอบอีกตัวหนึ่ง จะได้รู้ว่าอีมันเป็นที่ code ของเรา หรือว่าเป็นที่อุปกรณ์กันแน่ ซึ่งสุดท้ายตัวนึง OLED เสีย อีกตัวนึง LoRa Module เลย จนกันกับย่าน 915MHz



Heltec ESP32 SX1278 ความถี่ 433.175MHz ทดลองระยะประมาณ 80 เมตร ตัวส่งอยู่ในบ้าน ตัวรับอยู่ท้ายสวนมะพร้าว

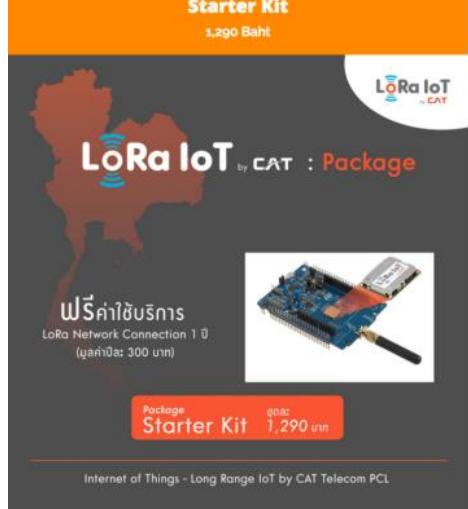
เนื้อหาหลักๆ ในตอนนี้จะเกี่ยวกับเรื่องของ LPWAN (Low Power Wide Area Network) อย่าง LoRa ทั้งการใช้งาน ข้อดี ข้อเสีย กฎหมายที่เกี่ยวข้องในการนำมาใช้งาน เพราะถ้าติดตามอ่านมาตั้งแต่ตอนที่ 1 ([ESP8266/ESP32: Introduction & Painlessmesh](#)) ใจความหลักในการใช้งาน Mesh Network ด้วย Painlessmesh นั้นจะไปถึงแต่ตอนที่ 3 แล้วครับ ที่เหลือ เป็นการประยุกต์มากกว่า ว่าเราจะใช้งาน Mesh Network ให้เหมาะสมยังไง หรือเพื่อจุดประสงค์ไหน เช่น

- เพื่อลด Single Point of Failure
- เพื่อขยายพื้นที่การครอบคลุมในการส่งข้อมูล
- ลดต้นทุนในการขยายจุดกระจายสัญญาณ

จากนั้นการส่งข้อมูลจาก Mesh Network ซึ่งเปรียบเหมือน Local Network ของเราระยะ Bridge ยังไงเพื่อให้ข้อมูลเหล่านั้นไปยัง Server, Database, Cloud, MQTT Broker หรืออะไรก็แล้วแต่ที่อยู่ Network อิกรอบหนึ่งหรือผ่าน Internet ซึ่งก็จะได้ทดลองทำกันไปในตอนที่ 3 ([ESP8266/ESP32: Painlessmesh Bridge](#)) ซึ่งถ้าใช้ไมโครชิป Software Serial ในการ Bridge ก็สามารถทำได้อีกหลายวิธี ซึ่งวันนี้เราที่จะมา Bridge ผ่าน LoRa กัน

LoRa vs LoRaWan

ผู้ให้บริการทางด้านโทรศัพท์มือถือในบ้านเรารู้ว่า AIS (AIS NB-IoT) และ True (True IoT) ที่เปิดตัวบริการ NB-IoT ออกมาน่าจะใช้ LoRa ([LoRa IoT by CAT](#)) ซึ่งทั้งคู่ก็ออกแบบมาเพื่อรับรองการสื่อสารสำหรับบริการประเภท IoT ในส่วนของ AIS และ True ที่เป็นผู้ให้บริการมือถืออยู่แล้ว มีความถี่ที่ใช้ในการจัดสรรสำหรับบริการ LTE การมาให้บริการ NB-IoT ที่จะประหยัดต้นทุนไปได้ยังไงและเป็นการ Utilize ความถี่ที่ตัวเองได้รับอนุญาตมาด้วย เท่าที่เห็นก็เป็น 3 ผู้ให้บริการหลักๆ ที่เปิดตัวออกมาร่วมกัน 3 หัวข้อของเรานั้นก็จะคุยกันในเรื่องของ LoRa และการใช้งานร่วมกัน Mesh Network จะนั้นเรามาดูข้อแตกต่างกันระหว่าง LoRa และ LoRaWan กันก่อน

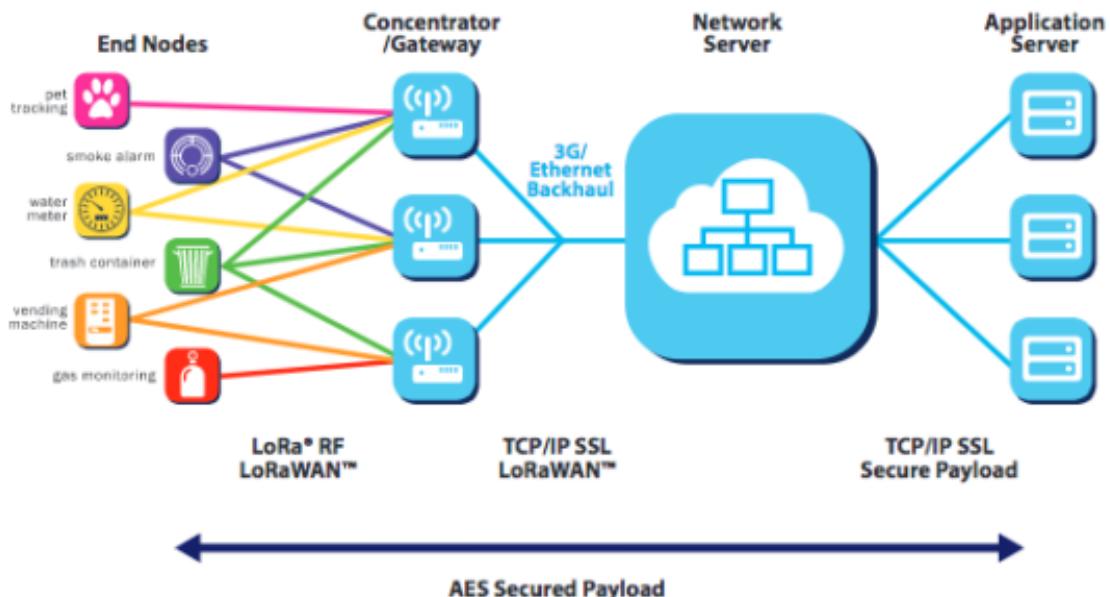


LoRa จะเป็นการสื่อสารในระดับ Physical Layer ซึ่งหมายที่จะใช้งานในลักษณะแบบ P2P ไม่มี Encryption ส่งข้อมูลกันแบบ Broadcast ออกไปเลย ฉะนั้นถ้ามีคนตั้ง LoRa Node ขึ้นมาแล้วใช้ค่า Setting เดียวกับเราได้สามารถที่จะเห็นข้อมูลได้ ซึ่ง LoRa ถูกพัฒนาเป็น Wireless Data Communication โดยบริษัท Cycleo (Grenoble, France) จากนั้นผู้ผลิตชิป Semtech ที่เข้าไปซื้อบริษัทนี้มาในปี 2012 ถ้าคุณสนใจดูรายละเอียดของชิปอย่าง SX1276 ที่รองรับย่านความถี่ 920-925MHz ตามประกาศของ กสทช ก็จะเห็นว่าที่ LoRa นั้นส่งได้ไกลนั้นส่วนนึงก็ เพราะ High sensitivity ที่รองรับถึง -148dBm เลย ([ข้อมูล Semtech SX1276](#)) และใช้งานจริงจะได้ที่เท่าไหร่ต้องดูกัน



LoRa module รุ่นต่างๆ สถาปัตย์เป็น RA-02 จาก Ai-Thinker ที่ใช้ชิป Semtech SX1278/SX1276 สำหรับความถี่ 433MHz และส่องค้าล่าง สำหรับย่าน 915MHz ส่วนส่งตัวล่างขวาเป็น RM95W จากค่าย Hope RF สำหรับความถี่ย่าน 915MHz

ส่วน LoRaWan นั้นจะอยู่ On top ของการสื่อสารแบบ LoRa Radio อีกที่ซึ่งจะมีในส่วนของ Protocol การรับส่งข้อมูลเข้ามา รองรับ Encryption ซึ่งใช้ AES-128 ในการเข้ารหัส สามารถทำ QOS ได้ด้วย โดยรายละเอียดของ LoRaWan นั้นจะมีอยู่ค่อนข้าง เชอะพอสมควร รวมถึงในส่วนของ Device End Node ก็ยังสามารถจำแนกได้ออกเป็น Class A B C อีก และการทำงานก็จะแบ่งออกเป็น Tier ตามรูปด้านล่างนี้โดยคร่าว



ฉะนั้นการจะเลือกใช้ LoRa หรือ LoRaWan ก็คือเลือกให้เหมาะสมกับ Application ที่จะใช้งานด้วย และที่สำคัญควรอยู่ตามข้อกำหนดของ กสทช ด้วยครับเนื่องจากถ้าคุณความต้องการใช้เป็นทรัพยากร่วมรวมที่มีการใช้งานร่วมกัน ที่ใช้ LoRa ก็ต้องดูในเรื่องของ Duty Cycle เองด้วยไม่ใช่ส่งข้อมูลตลอดเวลา ของการใช้งานคุณความต้องการ รวมถึงกำลังส่งที่ถ้ามากไป ก็จะไปรบกวนการใช้งานของอุปกรณ์หรือ Application ที่ใช้งานในความต้องการกัน

ข้อมูลเพิ่มเติมจาก LoRa Alliance ที่น่าศึกษา ซึ่งทาง LoRa Alliance ที่เป็น Non Profit Organization ที่ผลักดันในเรื่องของการใช้ LoRa/LoRaWan เพื่อใช้งาน IoT

- LoRaWan What is it? ([Download Link](#))
- LoRaWan 101: A Technical Introduction ([Download Link](#))

Limitations

เราๆคุ้นเคยกับการใช้ LoRaWan กันบ้าง ไม่ใช่ว่าเทคโนโลยีที่คือปัจจุบันนี้คือ LoRa/LoRaWan จะเหมาะสมครบถ้วนกับทุก Application ผู้ใดก็สนับสนุน Mesh Network ก็จะเหมาะสมกับ Application ในบางประเภท หรือบางงาน เพราะถ้าเราต้องการเอา LoRa/LoRaWan ไปส่งข้อมูลแบบ Realtime ก็คงไม่เหมาะสมแน่นอนจะกล่าวเป็นการของช่องสัญญาตอตอเวลา หรือให้ส่งข้อมูลขนาดใหญ่ๆๆก็คงไม่เหมาะสมเท่านั้น เพราะถูกจำกัดด้วยข้อมูลที่ถูกขนาดไม่ถึง byte และความเร็วในการส่งที่ต่ำ

Suitable use-cases for LoRaWAN:

- **Long range** – multiple kilometers
- **Low power** – can last months (or even years) on a battery
- **Low cost** – less than 20€ CAPEX per node, almost no OPEX
- **Low bandwidth** – something like 400 bytes per hour
- **Coverage everywhere** – you are the network! Just install your own gateways
- **Secure** – 128bit end-to-end encrypted

- **Geolocation / Triangulation** – you should probably use GPS for this, but we're doing our best to make it work with just LoRa. Check out [Collos](#).

Not Suitable for LoRaWAN:

- **Realtime data** – you can only send small packets every couple of minutes
- **Phone calls** – you can do that with GPRS/3G/LTE
- **Controlling lights in your house** – check out ZigBee or BlueTooth
- **Sending photos, watching Netflix** – check out WiFi

ที่มา: ([Limitations of LoRaWAN: The Things Network](#))

Regulation & NBTC

เนื่องจากล้วนความถี่อีกเป็นทรัพยากรที่ต้องถูกจัดสรรเพราเมื่ออยู่ย่างจำกัด และต้องมีกำกับดูแลในเรื่องของการใช้งาน ในประเทศไทยจะมีองค์กรที่ทันสมัยว่า องค์กรหนึ่งซึ่งชื่อว่า กสทช ([สำนักงานคณะกรรมการกิจการกระจายเสียง กิจการโทรทัศน์ และกิจการโทรคมนาคมแห่งชาติ](#)) ที่เดียวๆ ก็จะชอบมีคนออกมาให้ข่าวว่า เคยวึดใบอนุญาตมั่ง ปรับบุญนี่นั่นนั่น แล้วสุดท้ายเรื่องนี้เงียบไป มาครู่ร่องการใช้งานความถี่ของเรากันดีกว่ารับจากข้อมูลที่นำมาจากเวปของหน่วยงาน กสทช เองบอกได้เลยว่าเล่นอาจงทีเดียว บางทีคืนหากล้าเลขความถี่ต้องใช้เลขไทย ไม่เจ็บไม่เจอ ซึ่งผมรวบรวมมาได้ดังนี้โดยอิงตามความถี่ที่ย่านที่รวมกจะเห็นคุณเคยเห็นการใช้งานกันสำหรับ LoRa, RFID และ RC นั้นคือ EU433 ISM Band, AS923-925MHz

- ตารางกำหนดล้วนความถี่แห่งชาติ 2560 ([Download Link](#))
- กฎภาระ 2560: เอกสารประกอบการรับฟังความคิดเห็นสาธารณะ ร่างประกาศเกี่ยวกับการใช้คลื่นความถี่ 920-925MHz ([Download Link](#))
- สิงหาคม 2560: ร่างประกาศเกี่ยวกับการใช้คลื่นความถี่ 920-925MHz และแนวทางการอนุญาตเพื่อประกอบกิจการ IoT ([Download Link](#))
- พฤյิกษาณ 2560: ราชกิจจานุเบกษา – หลักเกณฑ์การอนุญาตให้ใช้คลื่นความถี่ย่าน 920-925MHz ([Download Link](#))
- พฤยิกษาณ 2560: ราชกิจจานุเบกษา – มาตรฐานทางเทคนิคของเครื่องโทรคมนาคมและอุปกรณ์สำหรับเครื่องวิทยุคมนาคมที่ไม่ใช่ประเภท RFID ซึ่งใช้คลื่นความถี่ย่าน 920-925MHz ([Download Link](#))
- กฎก.พ. 2561: ราชกิจจานุเบกษา – หลักเกณฑ์และเงื่อนไขการอนุญาตให้ใช้คลื่นความถี่สำหรับภาคประชาชนซึ่งไม่มีนักบินสำหรับใช้งานเป็นการทั่วไป ([Download Link](#))
- กฎก.พ. 2561 ราชกิจจานุเบกษา – เครื่องวิทยุคมนาคมและสถานีวิทยุคมนาคมที่ได้รับการยกเว้นไม่ต้องได้รับใบอนุญาตตามพระราชบัญญัติวิทยุคมนาคม ([Download Link](#))
- ความถี่วิทยุและใบอนุญาตวิทยุโทรคมนาคมประเภท RFID ([Download Link](#))

- การปรับปรุงคุณภาพโดยด้านการบริหารกลุ่มความถี่และทรัพยากริโภคุณภาพเพื่อรับการพัฒนาของ Internet of Things ในประเทศไทย
[Download Link](#)

จะน้ำสำหรับส่วนของคลื่นความถี่ 433MHz ซึ่งที่เราทดสอบกันสำหรับ LoRa นั้น ก็จะไม่ได้เกี่ยวกับการใช้งานพื้นที่ RFID หรือการใช้งานในด้าน RC กับอากาศยานไร้คนขับ แต่จะทดสอบโดยอาศัยความต่างเครื่องวิทยุคุณภาพและสถานีวิทยุคุณภาพที่ได้รับการยกเว้นไม่ต้องได้รับใบอนุญาต ซึ่งย่าน 433MHz นั้นห้ามส่งกิน 10 มิลลิวัตต์

สำหรับย่าน 920-925MHz นั้นซึ่งมีข้อกำหนดในส่วนของ มาตรฐานทางเทคโนโลยีของวิทยุคุณภาพอุปกรณ์ กสทช. มท. 1033 – 2560 ว่าด้วยเรื่องเครื่องวิทยุคุณภาพที่ไม่ใช่ประเภท Radio Frequency Identification: RFID ซึ่งใช้คลื่นความถี่ย่าน 920-925MHz เครื่องเดียวกัน ให้ดำเนินการได้รับยกเว้นไม่ต้องได้รับใบอนุญาตให้ดำเนินใช้สำหรับชั้นนำออกซิเจน เครื่องวิทยุคุณภาพและใบอนุญาตให้ดึงสถานีวิทยุคุณภาพแต่ไม่ได้รับยกเว้นใบอนุญาตให้ค้าซึ่งเครื่องวิทยุคุณภาพ

AS920-923

Used in Japan, Malaysia, Singapore

Uplink:

1. 923.2 - SF7BW125 to SF12BW125
2. 923.4 - SF7BW125 to SF12BW125
3. 922.2 - SF7BW125 to SF12BW125
4. 922.4 - SF7BW125 to SF12BW125
5. 922.6 - SF7BW125 to SF12BW125
6. 922.8 - SF7BW125 to SF12BW125
7. 923.0 - SF7BW125 to SF12BW125
8. 922.0 - SF7BW125 to SF12BW125
9. 922.1 - SF7BW250
10. 921.8 - FSK

Downlink:

- Uplink channels 1-10 (RX1)
- 923.2 - SF10BW125 (RX2)

AS923-925

Used in Brunei, Cambodia, Hong Kong, Indonesia, Laos, Taiwan, Thailand, Vietnam

Uplink:

1. 923.2 - SF7BW125 to SF12BW125
2. 923.4 - SF7BW125 to SF12BW125
3. 923.6 - SF7BW125 to SF12BW125
4. 923.8 - SF7BW125 to SF12BW125
5. 924.0 - SF7BW125 to SF12BW125
6. 924.2 - SF7BW125 to SF12BW125
7. 924.4 - SF7BW125 to SF12BW125
8. 924.6 - SF7BW125 to SF12BW125
9. 924.5 - SF7BW250
10. 924.8 - FSK

Downlink:

- Uplink channels 1-10 (RX1)
- 923.2 - SF10BW125 (RX2)

ถ้าหากดูตาม Frequency Plan และตามช่วงความถี่ที่ กสทช ปลดล็อกให้กับอุปกรณ์และบริการ IoT ก็สามารถใช้ได้สองช่วงเดียวกัน ทั้ง AS920 กับ AS923 ([The Things Network Frequency Plan](#))

จากข้อมูลด้านบนเนี่ย ก็ต้องอ่านกันดีๆ หลักๆ คือรายละเอียดตามกลุ่มความถี่ที่ใช้งานและจุดประสงค์ที่ใช้งาน จากนั้นก็คุ้มหักเงินที่จะเสียไปที่ ประกาศในราชกิจจานุเบกษา และก็ขึ้นอยู่กับความต้องดูด้วยว่าในประกาศนั้นมีการยกเลิกประกาศตัวก่อไว้หรือไม่ หรือพอกประโยคที่ว่าเว้นแต่กำหนดเป็นอย่างอื่น พากร่างกับการรับฟังความคิดเห็นสาธารณะก็อาจใช้ประกอบได้ กรณีใดที่สุดหากมีข้อสงสัยในเรื่องของการใช้งานความถี่ การผลิตหรือ

นำเข้าเรื่องอุปกรณ์ความคิดเห็นที่ใช้ความถี่ กีวารของความต้องจาก กสทช เป็นหลักซึ่งกีวารทำเป็นหนังสือแจ้งไปเพื่อให้ กสทช ตอบกลับว่าสามารถทำได้ หรือไม่สามารถทำได้ หรือให้ข้อมูลตามข้อกำหนดใหม่

Scenario

กรณีในบ้านที่เราควรจะ Bridge ผ่าน LoRa ซึ่งข้อดีของ LoRa เลยก็คือเรื่องของ Power Consumption ที่ต่ำ และระยะในการส่งที่สูง ฉะนั้นค่าขุมสมบัตินี้ แทนที่จะต้องการเก็บข้อมูล หรือความคุณภาพรัฐ จะต้องวาง Mesh Network ให้ครอบคลุมมาถึงจุดที่จะส่งข้อมูลออก Internet อาจจะไม่จำเป็น แต่ถ้าศักยภาพส่ง ระยะไกลผ่าน LoRa มาแล้วค่อยมาออก Internet จากผู้ของ LoRa Node ที่ทำตัวเป็น Gateway ที่ได้สำหรับตอนที่ 4 นี้ก็จะเป็นการใช้งานผ่าน LoRa นะครับ แต่ถ้าใครอยาก Advance Setup LoRaWan Gateway แล้วใช้งานผ่าน LoRaWan ที่ได้อ่านที่นักวิเคราะห์กับกันตอนที่ 3 ลองมาคิดกัน เล่นๆครับ

Scenario 1: Smart Farm

ลองนึกภาพของ Green House เรือนปลูกผัก ฟาร์มข้าวโพด หรือทุ่งนา ที่ต้องการควบคุมระบบจากน้ำ หรือเก็บข้อมูลอุณหภูมิ ความชื้น ค่า PH/EC ของดินหรือสารละลาย แล้วส่งข้อมูลมาที่ส่วนกลางที่อยู่ติดกับพื้นที่ในการวาง Sensor Node เหล่านี้

เคล็ดนี้การใช้ Mesh Network เหมาะมาก เพราะได้ประโยชน์ในเรื่องของการครอบคลุมพื้นที่ที่กว้าง จากการกระจายตัวของ Node แล้วสร้างเป็น Mesh Network โดยที่จุดเชื่อมต่อสุดท้ายอาจอยู่ที่ชายขอบ หรือสูนย์กลางของ Mesh Network ก็จะเหมาะสมมาก สามารถติดตันทุนในเรื่องของการสร้าง Network สำหรับการสื่อสารของแต่ละ Node ได้

Scenario 2: Falling Rock Alarm

สำหรับคสันเป็นเคล็ดลับที่ได้หัน ที่ใช้ในการตรวจสอบหินที่ถล่มจากภูเขา เนื่องจากได้หันนี้มีสภาพภูมิประเทศเป็นเคาะ และอยู่ในเขตแผ่นดินไหว ซึ่งถนนบางเส้นนี้ก็ตัดผ่านภูเขาทำให้มีถนนแยกภาคเหนือของประเทศไทย ซึ่งบางครั้งก็จะมีหินร่วงหล่นลงมา กว่าจะทราบเหตุก็ต้องมีคนโทรแจ้งจากที่ไปเจอกัน



ภาพประกอบจาก: <https://www.geostru.eu/rockfall-analysis/>

ซึ่งที่ได้หัววัน ได้มีการติดตั้งเซนเซอร์ตรวจจับการสั่นสะเทือนของแนวคลื่นข่ายก้อนหินตกตามถนนไว้ เพื่อตรวจจับว่ามีหินร่วงหล่นลงมา หรือมีคิมส์ไลด์ลงมาปิดถนนมั้ย โดยการใช้ ภารตีอุตสาหกรรม LoRa เพราะการตรวจจับไม่ได้ตรวจสอบตั้งแต่แนว เลยวางช่วงที่ตัดผ่านภูเขา และต้องการส่งข้อมูลระยะไกล LoRa จึงตอบโจทย์ดี จะนั่นคือถ้าภูเขานี้ ใจกลางที่น้ำทุนที่ต่อ และส่งข้อมูลแจ้ง เห็นไปที่ศูนย์แจ้งเตือนด้านภัย Bridge ผ่าน LoRa ที่อาจอยู่ไกลออกไปทางทิศใต้ สำหรับ ไฟไฮไลท์ในภูเขานี้ ให้โคมไฟติดตั้งที่ด้านทุนที่ต่อ และส่งข้อมูลแจ้ง ผ่าน กีจอกอนบีปีคไปครึ่งเดือนเป็นบางช่วงของปีเมื่อไอนกัน สำหรับล่างหน้า ก็จะสามารถวางแผนการเดินทางหรือหลีกเลี่ยงเส้นทางได้ถูก

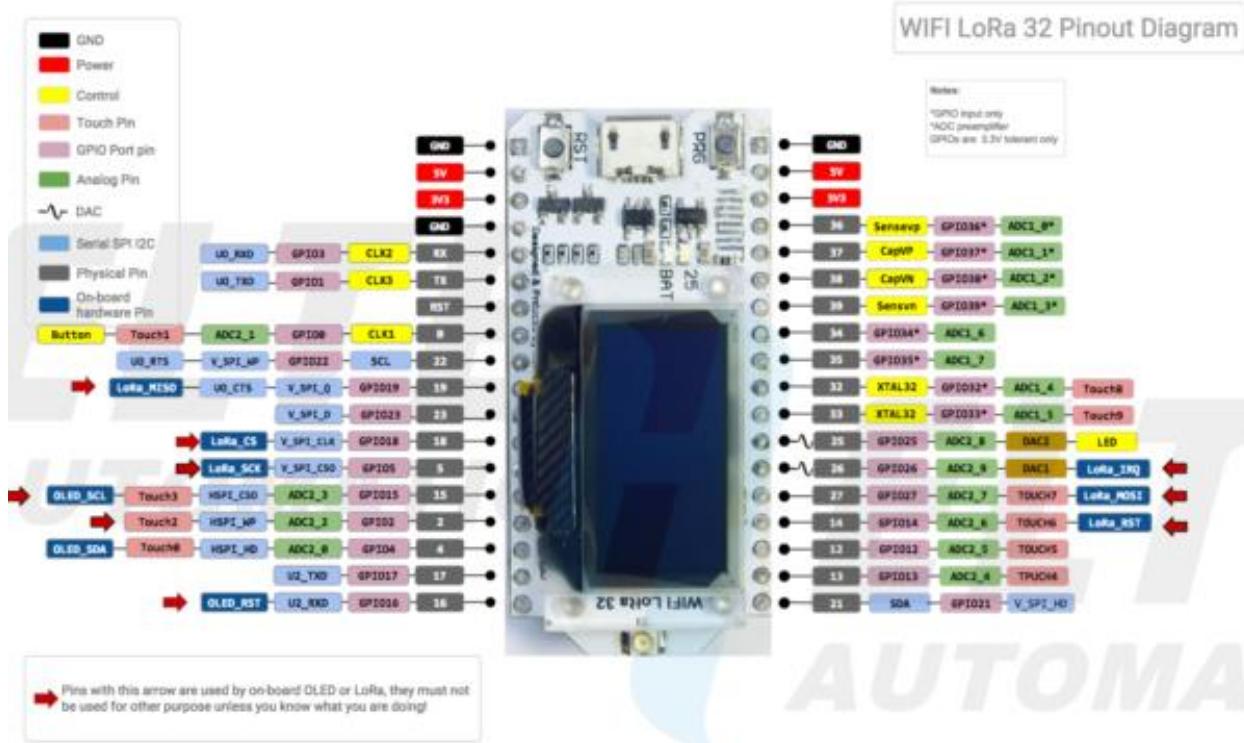
Scenario 3: Smart Bin

เคลนี้ก็เป็นเคลที่เกิดขึ้นจริงอีกเช่นกัน ลองคิดคุยกันว่า ของการเก็บขยะ ซึ่งในบางหมู่บ้าน ชุมชน ที่ต้องมีรถเก็บขยะวนเข้าไปเก็บทุกวันฯ หรือบางที่วันนึงอาจจะหลายรอบก็ได้ หรือ บางที่เข้าไปเก็บ หลายวันถ้วนอาจจะไม่มีเลยก็ได้

ซึ่งเคลนี้มีคนทำเป็น Product Smart Bin ที่มีห้องตัวนิดอีกห้องเพื่อเพิ่มพื้นที่การเก็บขยะ ได้มากขึ้น มีตัว Monitor ระดับของขยะเพื่อส่งคำสั่งไปแจ้งเตือนที่ศูนย์ฯ ให้ ห้ามเก็บขยะได้แล้ว รวมถึงจัดเส้นทางให้รถเก็บขยะได้ด้วย ซึ่งเทคโนโลยีการสื่อสารที่ใช้โดยมากก็จะเป็น LoRa หรือไม่ก็ Cellular (2G, 3G, NB-IoT) เพราะด้วย ลักษณะการตั้งวางตั้งขยะของหมู่บ้าน หรือชุมชนใน ตอน อาจไม่เหมาะสมกับการนำ mesh มาใช้

Wiring & Diagram

ความจริงการใช้งานจะใช้งานผ่าน Module SX1278/1276 ของ Ai-Thinker ที่ได้แต่ต้องต่อสายเข้ากับ Breadboard หรือในส่วนของ RFM95W ที่ ต้องมานัดกรีกันอีก จะนั่นตอนนี้อาจง่ายสุดสำหรับผม ก็ใช้บอร์ดสำเร็จรูปที่มาพร้อม ESP32+OLED ของ Heltec/TTGO เลขดีกว่า เพราะว่าทำการ wiring ไห บน pcb เรียบร้อยแล้ว ซึ่ง pinout ก็จะเป็นดังรูปด้านล่างนี้



นั่นก็คือการใช้งาน module sx127x ในการต่อสัญญาณที่ไม่สามารถนำไปใช้งานได้

```
#define SCK 5 // GPIO5 -- SX127x's SCK  
  
#define MISO 19 // GPIO19 -- SX127x's MISO  
  
#define MOSI 27 // GPIO27 -- SX127x's MOSI  
  
#define SS 18 // GPIO18 -- SX127x's CS  
  
#define RST 14 // GPIO14 -- SX127x's RESET  
  
#define DIO 26 // GPIO26 -- SX127x's IRQ(Interrupt Request)  
  
#define BAND 915.2E6 //you can set band here directly,e.g. 868E6,915E6
```

ติดตั้ง Arduino core for ESP32 WiFi chip

ขั้นตอนนี้อาจแตกต่างไปจากตอนที่เราติดตั้ง ESP8266, ATTiny85 หรือ STM32 F103C Series ไปจากตอนก่อนหน้านี้ ซึ่ง Source ที่ใช้นการติดตั้งมีอยู่ 2 แหล่งด้วยกันคือ

จาก Espressif ผู้ผลิตชิป ESP32

<https://github.com/espressif/arduino-esp32#installation-instructions>

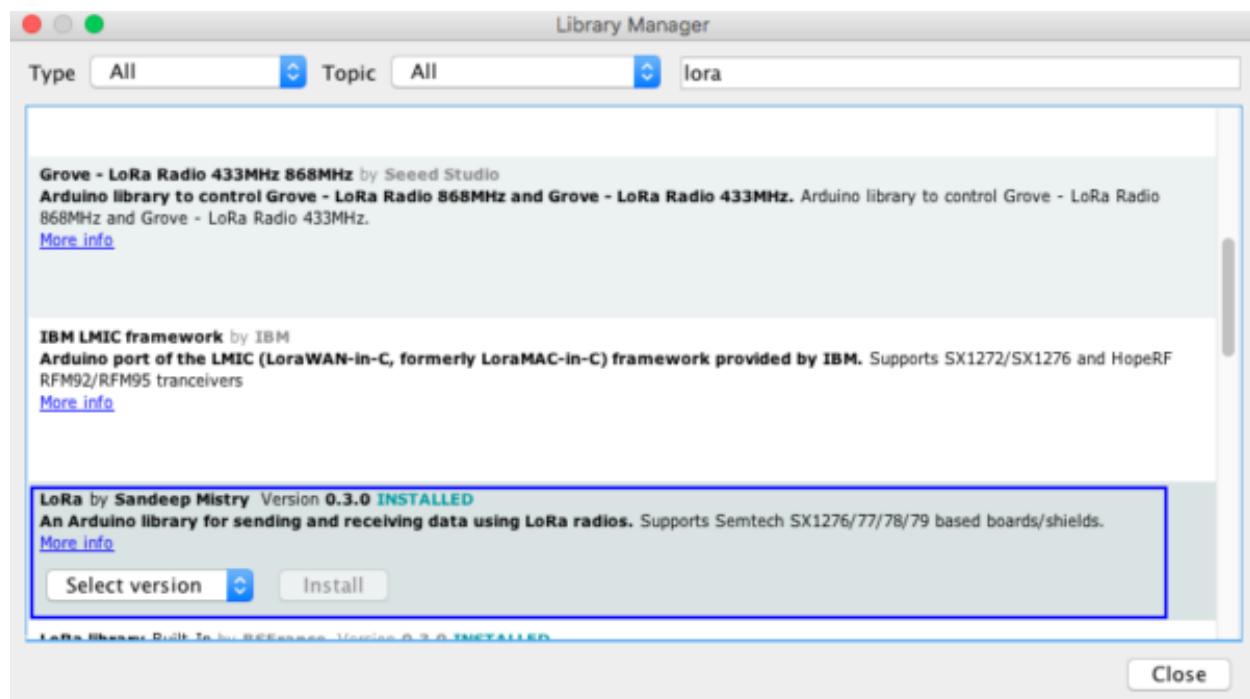
จาก Heltec ผู้ผลิตบอร์ด Heltec ESP32 LoRa

https://github.com/Heltec-Aaron-Lee/WiFi_Kit_series#installation-instructions

ข้อเดียวกันต่างระหว่างสองบอร์ดนี้คือ ถ้า Clone มาจาก Heltec จะมี Library LoRa มาด้วยซึ่งก็เป็นตัวที่ไม่มาจากการของคุณ Sandeep Mistry แต่ถ้าการอัปเดตจะไม่ถูก Update ผ่านทาง Library manager ต้องขอตามจากทาง Heltec เอง polymod เลือก Clone มาจากทาง Espressif แล้วทำการติดตั้ง LoRa Library เอง ส่วนวิธีการติดตั้งก็ขึ้นกับ OS ของแต่ละท่านที่ใช้อยู่ หลังจากติดตั้งไปเลือกทางไฟแล้วก็อ่าศัพท์ชื่อของบอร์ดที่ใช้งานนั้นๆ แล้วครับท่าตาม Instruction ในแต่ละ OS ได้เลย

LoRa Library by Sandeep Mistry

สำหรับใครที่เลือกติดตั้ง Board โดยเลือก Source จาก Heltec ก็ขั้นตอนติดตั้ง Library นี้ไปได้โดยครับ แต่ถ้าใครที่ติดตั้งผ่าน Source ของ Espressif ก็ให้ติดตั้ง LoRa Library ของคุณ Sandeep Mistry โดยเข้าไปที่ Sketch -> Include Library -> Library Manager แล้วค้นหาคำว่า LoRa ครับ จะเจอกาหนดตัวมากไปเลือกตามรูปด้านล่างแล้วกด Install ได้เลยครับ



มาตรฐานส่วนของ Code ที่จะใช้ทดสอบระบบกันบ้าง ตัวโปรแกรมก็จะแบ่งเป็นสองชุดคือฝั่งส่งและฝั่งรับ ฝั่งส่งก็ Count Number แล้วส่งออกไปเรื่อยๆ ส่วนฝั่งรับนั้นเนื่องด้วยไม่มี GPS Module ก็เลยอ่าศัพท์ GPS Stream จาก Blynk มาใช้ในการเก็บค่าพิกัดแทนเพื่อที่จะได้มาระยะเป็นอย่างไรบ้าง ซึ่งถ้าใช้บอร์ดที่ใช้ชิป SX1278 ในส่วนของ Band จะกำหนดค่าเป็น 433.175MHz และถ้าเป็นบอร์ดที่ใช้ชิป SX1276 ในส่วนของ Band จะกำหนดค่าเป็น 923.2MHz นอกจากนี้ไม่มีอะไรมากต่าง ส่วนโครงสร้างที่จะทดสอบโดยที่ไม่ได้สนใจเรื่องของ พิกัดก็ตัด code ในส่วนของ Blynk ออกได้ครับ

Code for Sender:

```
#include <SPI.h>
```

```

#include <LoRa.h>

#include "SSD1306.h"

int counter = 0;

// GPIO5 -- SX1278's SCK

// GPIO19 -- SX1278's MISO

// GPIO27 -- SX1278's MOSI

// GPIO18 -- SX1278's CS

// GPIO14 -- SX1278's RESET

// GPIO26 -- SX1278's IRQ(Interrupt Request)

//OLED pins to ESP32 0.96OLEDGPIOs via this connectin:

//OLED_SDA -- GPIO4

//OLED_SCL -- GPIO15

//OLED_RST -- GPIO16

SSD1306 display(0x3c, 4, 15);

#define SS 18

#define RST 14

#define DI0 26

#define BAND 433.175E6 //915E6

void setup() {

Serial.begin(115200);

pinMode(25, OUTPUT); //Send success, LED will bright 1 second

while (!Serial);

pinMode(16, OUTPUT);

digitalWrite(16, LOW); // set GPIO16 low to reset OLED

delay(50);

digitalWrite(16, HIGH); // while OLED is running, must set GPIO16 in high

Serial.println("LoRa Sender");

```

```
SPI.begin(5, 19, 27, 18);

LoRa.setPins(SS, RST, DIO);

if (!LoRa.begin(BAND)) {

    Serial.println("Starting LoRa failed!");

    while (1);

}

Serial.println("LoRa Initial OK!");

// Initialising the UI will init the display too.

display.init();

display.flipScreenVertically();

display.setFont(ArialMT_Plain_10);

}

void loop() {

    Serial.print("Sending packet: ");

    Serial.println(counter);

    // send packet

    LoRa.beginPacket();

    LoRa.print("hello ");

    LoRa.print(counter);

    LoRa.endPacket();

    display.clear();

    display.setTextAlignment(TEXT_ALIGN_LEFT);

    display.drawString(10, 5, "Sending:");

    display.drawString(10, 20, "hello " + String(counter));

    // write the buffer to the display

    display.display();

    counter++;
}
```

```

digitalWrite(25, HIGH); // turn the LED on (HIGH is the voltage level)

delay(1000); // wait for a second

digitalWrite(25, LOW); // turn the LED off by making the voltage LOW

delay(1000); // wait for a second

delay(5000);

}

```

Code for Receiver:

```

#include <SPI.h>

#include <LoRa.h>

#include "SSD1306.h"

/* Comment this out to disable prints and save space */

#define BLYNK_PRINT Serial

#include <WiFi.h>

#include <WiFiClient.h>

#include <BlynkSimpleEsp32.h>

char auth[] = "xxxxxxxx"; //blynk auth code

// Your WiFi credentials.

// Set password to "" for open networks.

char ssid[] = "xxx xxxx";

char pass[] = "xxxx";

float lat;

float lon;

// GPIO5 -- SX1278's SCK

// GPIO19 -- SX1278's MISO

// GPIO27 -- SX1278's MOSI

```

```
// GPIO18 -- SX1278's CS
// GPIO14 -- SX1278's RESET
// GPIO26 -- SX1278's IRQ(Interrupt Request)
//OLED pins to ESP32 0.96OLEDGPIOs
//OLED_SDA -- GPIO4
//OLED_SCL -- GPIO15
//OLED_RST -- GPIO16
SSD1306 display(0x3c, 4, 15);

#define SS 18
#define RST 14
#define DI0 26
#define BAND 433.175E6 //915E6
WidgetMap myMap(V1);
BLYNK_WRITE(V0) {
GpsParam gps(param);
// Print 6 decimal places for Lat, Lon
lat = String(gps.getLatitude(),6).toFloat();
lon = String(gps.getLongitude(),6).toFloat();
Serial.print("Lat: ");
Serial.println(lat,7);
Serial.print("Lon: ");
Serial.println(lon,7);
Serial.println();
}
void setup() {
Serial.begin(115200);
while (!Serial);
```

```
pinMode(16, OUTPUT);

digitalWrite(16, LOW); // set GPIO16 low to reset OLED

delay(50);

digitalWrite(16, HIGH); // while OLED is running, must set GPIO16 in high

Serial.println("LoRa Receiver");

SPI.begin(5, 19, 27, 18);

LoRa.setPins(SS, RST, DIO);

if (!LoRa.begin(BAND)) {

    Serial.println("Starting LoRa failed!");

    while (1);

}

Serial.println("LoRa Initial OK!");

// Initialising the UI will init the display too.

display.init();

display.flipScreenVertically();

display.setFont(ArialMT_Plain_10);

Blynk.begin(auth, ssid, pass);

myMap.clear();

}

void loop() {

String tmp_string, tmp_rssi;

// try to parse packet

int packetSize = LoRa.parsePacket();

if (packetSize) {

    // received a packet

    Serial.print("Received packet ");

    // read packet
```

```

while (LoRa.available()) {

    //Serial.print((char)LoRa.read());

    tmp_string += (char)LoRa.read();

}

Serial.print(tmp_string);

tmp_rssi = LoRa.packetRssi();

// print RSSI of packet

Serial.println(" with RSSI " + tmp_rssi);

//Serial.println(LoRa.packetRssi());

display.clear();

display.setTextAlignment(TEXT_ALIGN_LEFT);

display.drawString(10, 0, "Received:");

display.drawString(10, 15, tmp_string+" RSSI: "+tmp_rssi);

display.drawString(10, 30, String(lat,7));

display.drawString(10, 45, String(lon,7));

// write the buffer to the display

display.display();

//BLYNK_WRITE(V0);

Blynk.syncVirtual(V0);

myMap.location(1, lat, lon, "value");

}

tmp_string = "";

tmp_rssi = "";

Blynk.run();

}

```

LoRa Range Test Result: 433.175MHz vs 923.2MHz

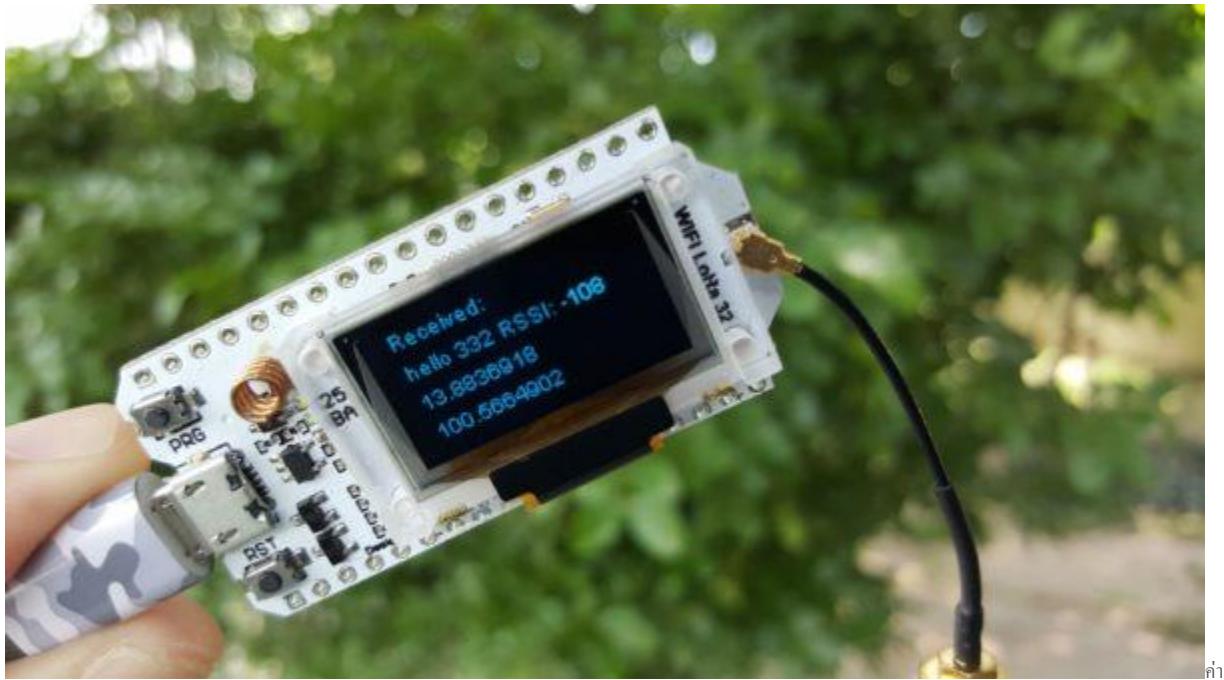


เริ่มกันที่

หุค Start วางคู่กัน โดยใช้ความถี่ในการรับ-ส่ง ที่ 433.175MHz ได้พิสูจน์ว่าข้อมูล

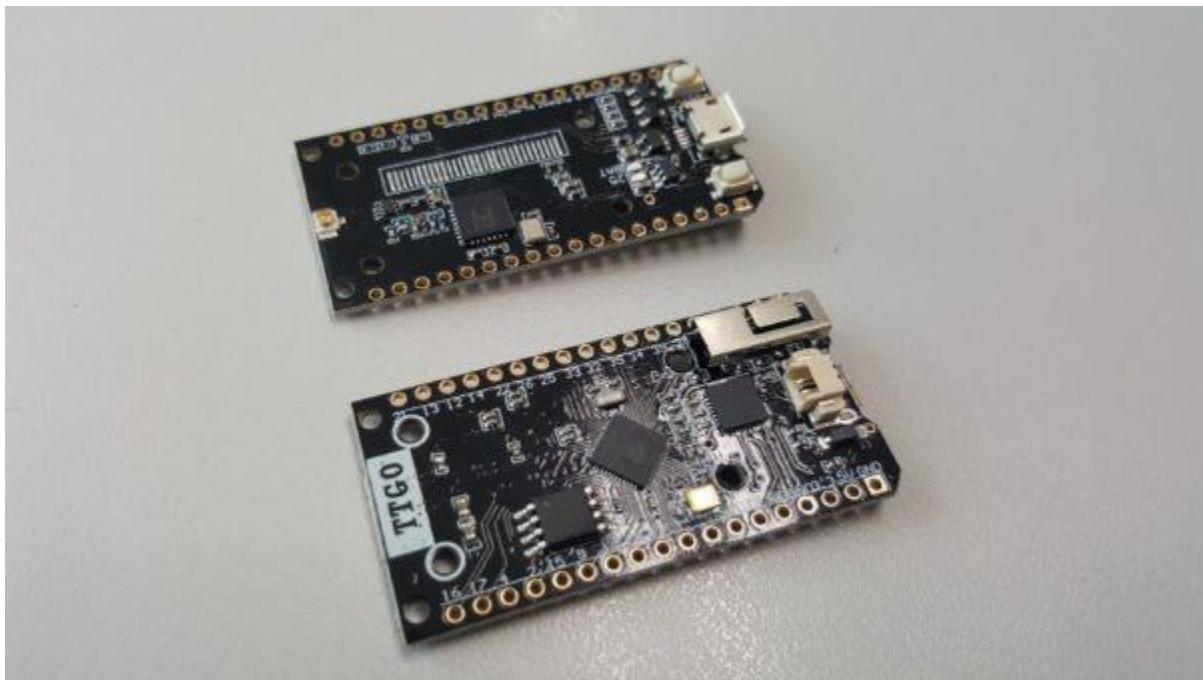


หลังจากนั้นทีเดินเล่นเลขครับ เล่นเอาหน่อยดูกะเมื่อนั้น RSSI: -107dBm

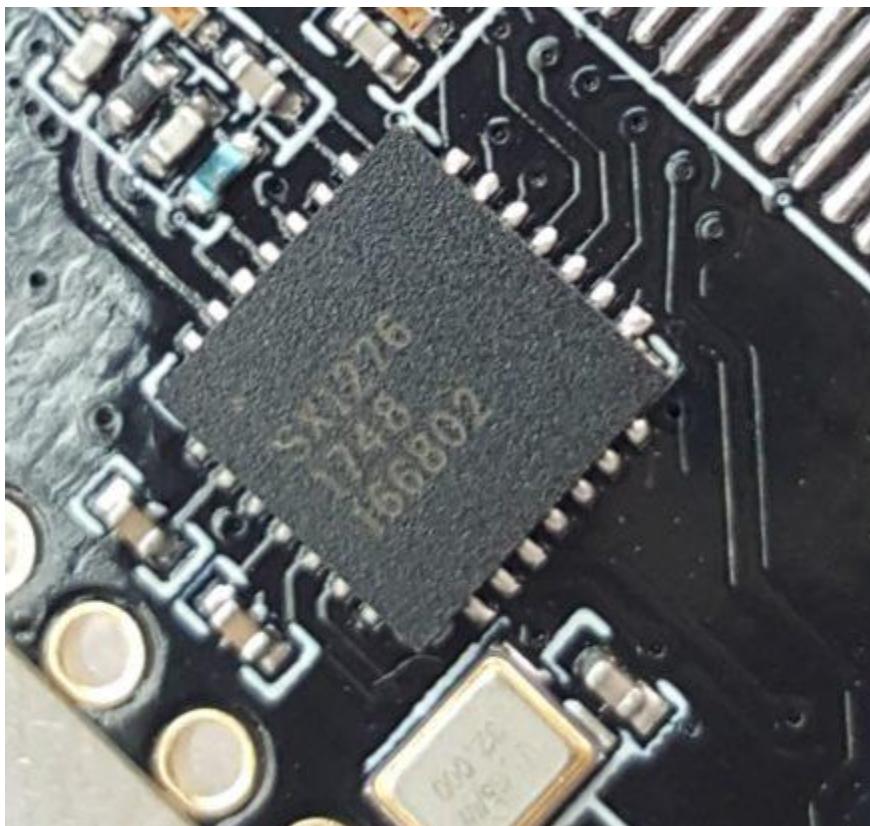


ค่าสุดที่ Heltec ESP32 LoRa Sx1278 รับได้จะอยู่ที่ -108 dBm

ในความโปรดปรานของ Heltec WiFi LoRa SX1276 ที่ใช้งานย่าน 915MHz จะเสียดต่อกับไดบอร์ดใหม่มากันพอยให้ทดสอบ ซึ่งเป็นของ TTGO เป็น ESP32 LoRa SX1276 เหมือนกันเพียงแต่เปลี่ยนไปมี OLED มาด้วยเท่านั้น จะนั่นการวัดระยะ และค่า RSSI ก็เลียดมาข้ออ่านค่า GPS จาก App Blynk บนโทรศัพท์มือถือแทน

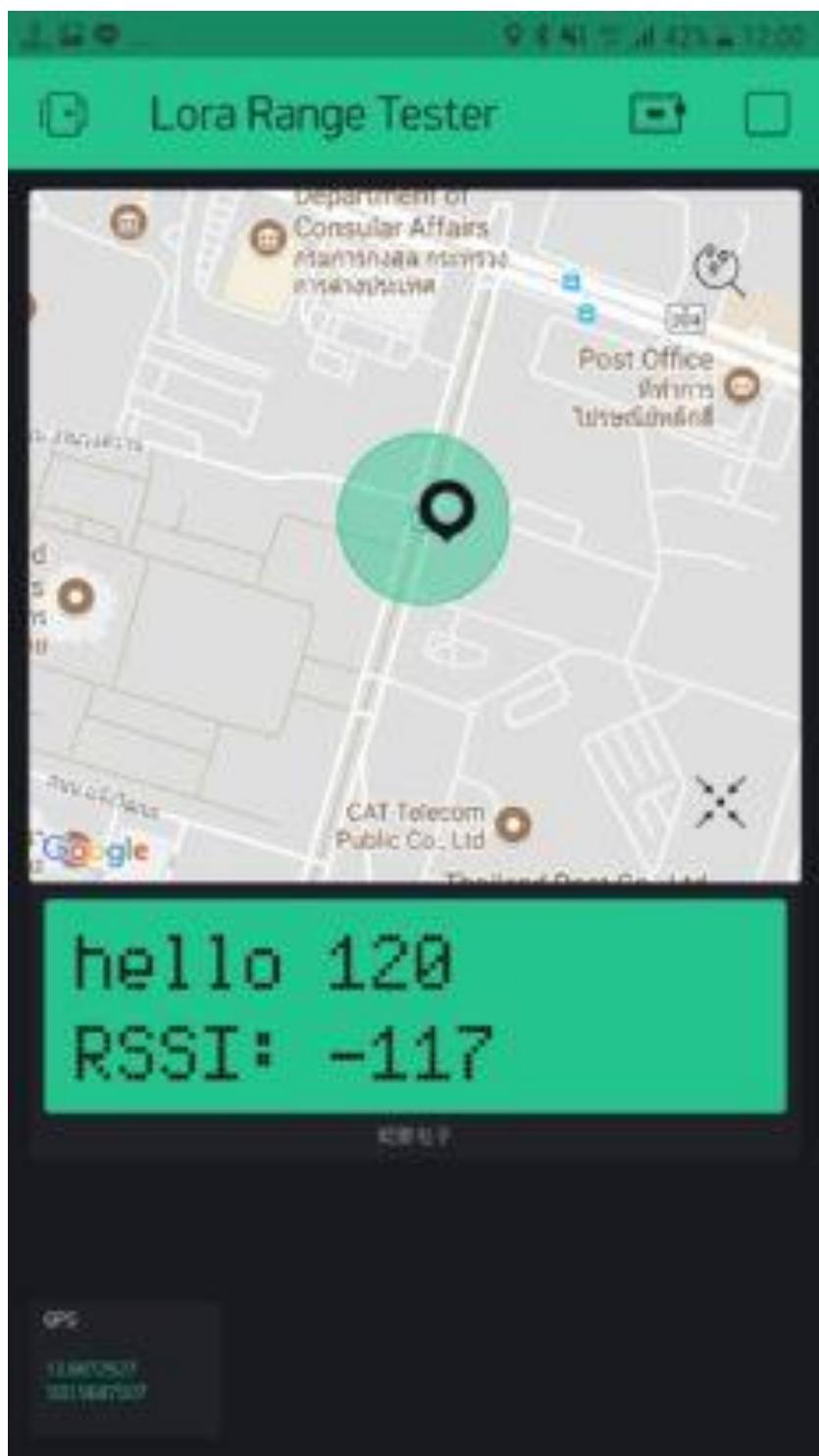


TTGO ESP32 LoRa SX1276



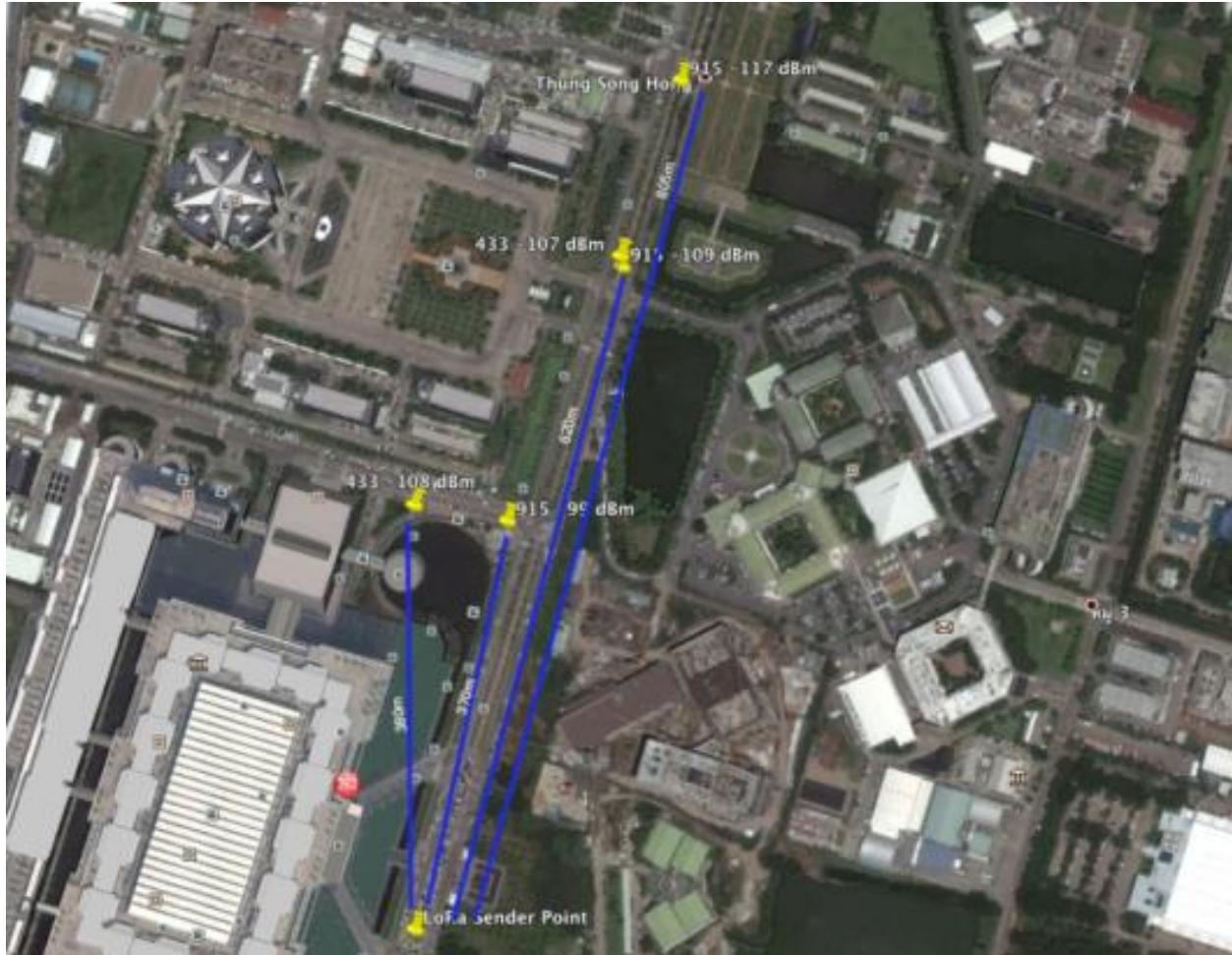
Semtech SX1276: LoRa ชิป ที่บัน

915MHz



ค่า RSSI ที่สามารถรับได้ดีสุดและรับได้ดีเมื่อยอดอยู่ที่ระหว่าง -110 ถึง -117 dBm

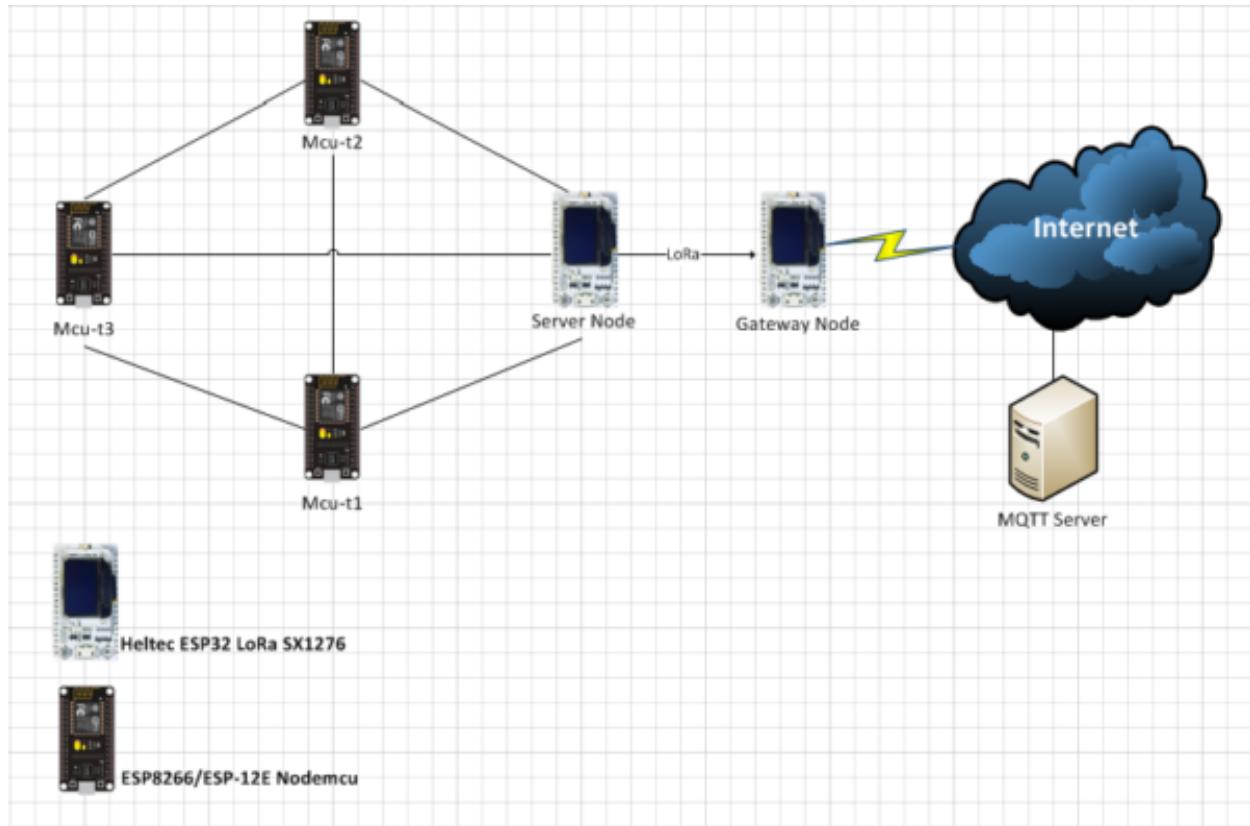
ชี้งการทดสอบสำหรับการรับ-ส่งที่ความถี่ 923.2MHz



ผลการทดสอบระยะของแต่ละความถี่ที่ใช้ตามรูปด้านบนแลยกับ ซึ่งถ้าอาระยะห่างผลไกลสุดซึ่งอยู่ใน Line of Sight ก็จะอยู่ที่ประมาณ 700-750 เมตรสำหรับการรับ-ส่งที่ความถี่ 923.2MHz ล่าวนคุณ 433.175MHz ระยะห่างผลก็จะอยู่ที่ประมาณ 600 เมตร ทำให้การใช้งาน LoRa นั้นค่อนข้างสมชื่อ Long Range เลย เพราะนี่คือใช้สายอากาศที่แฉมมา ถ้าใช้สายอากาศเดียว และตั้งเสาสูงน่าจะไปได้ไกล กว่านี้ แต่ทั้งนี้ทั้งนั้นกำลังส่งก็ไม่ควรเกิน กสพท กำหนดคุณครับ ซึ่งเกลที่เขียนนี้ “สำหรับการทดลองเท่านั้น” เพราะจาก Code ได้มีการกำหนด TX Power ไว้ที่ 17dBm บวกกับเสาที่แฉมมาก็จะไม่เกิน +3dBm ติดลบๆ ก็ 20dBm หรือ 100 milliwatts แล้ว (แต่ส่วนตัวคิดว่าไม่น่าจะถึง น่าจะมี loss เยอะ ถ้าอยากรู้จริงๆ ก็คงต้องอาอุปกรณ์มาวัด)

Painlessmesh Bridge with Lora

เกริ่นมาช่วยๆ ๆ ก่อนที่จะเข้าเรื่องของเรากันจริงๆ ในการใช้ Painlessmesh สร้าง Mesh Network และ Bridge ข้อมูลผ่าน LoRa กัน ซึ่ง Configuration ก็จะเป็นลักษณะดัง diagram ด้านล่างนี้ครับ โดยมี Heltec ESP32 Lora SX1278 เป็นตัว Bridge เพื่อส่งข้อมูลไปยัง MQTT Server



เพื่อเห็นว่ามี Dependency Library เพิ่มเข้ามาสำหรับ Painlessmesh อีกด้วย AsyncTCP สำหรับ ESP32 และ ESPAsyncTCP สำหรับ ESP8266 ถ้าใคร Update เป็น version ล่าสุดของ Painlessmesh ก็อย่าลืมติดตั้งเพิ่มเข้าไปนะครับ

สำหรับ ESP32

<https://github.com/me-no-dev/AsyncTCP>

สำหรับ ESP8266

<https://github.com/me-no-dev/ESPAsyncTCP>

Nodemcu + DHT22 (mcu-t1, mcu-t2, mcu-t3)

Code:

```
#include "painlessMesh.h"
```

```
#include "DHT.h"
```

```
#define DHTPIN D4
```

```
#define DHTTYPE DHT22
```

```

#define MESH_PREFIX "HelloMyMesh"

#define MESH_PASSWORD "hellomymeshnetwork"

#define MESH_PORT 5555

DHT dht(DHTPIN, DHTTYPE);

void receivedCallback( uint32_t from, String &msg );

painlessMesh mesh;

size_t logServerId = 0;

// Send message to the logServer every 10 seconds

Task myLoggingTask(10000, TASK_FOREVER, []() {

    float h = dht.readHumidity();

    float t = dht.readTemperature();

    DynamicJsonBuffer jsonBuffer;

    JsonObject& msg = jsonBuffer.createObject();

    msg["nodename"] = "mcu-t1"; //change for identify for the node that send data mcu-t1 to mcu-t3

    msg["NodeID"] = mesh.getNodeId();

    msg["Temp"] = String(t) + "C";

    msg["Humidity"] = String(h) + "%";

    String str;

    msg.printTo(str);

    if (logServerId == 0) // If we don't know the logServer yet

        mesh.sendBroadcast(str);

    else

        mesh.sendSingle(logServerId, str);

    // log to serial

    msg.printTo(Serial);

    Serial.printf("\n");

});
```

```

void setup() {
    Serial.begin(115200);

    Serial.println("Begin DHT22 Mesh Network test!");

    dht.begin();

    mesh.setDebugMsgTypes( ERROR | STARTUP | CONNECTION ); // set before init() so that you can see startup messages

    mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT, STA_AP, AUTH_WPA2_PSK, 6 );

    mesh.onReceive(&receivedCallback);

    // Add the task to the mesh scheduler

    mesh.scheduler.addTask(myLoggingTask);

    myLoggingTask.enable();

}

void loop() {
    // put your main code here, to run repeatedly:

    mesh.update();

}

void receivedCallback( uint32_t from, String &msg ) {

    Serial.printf("logClient: Received from %u msg=%s\n", from, msg.c_str());

    // Saving logServer

    DynamicJsonBuffer jsonBuffer;

    JsonObject& root = jsonBuffer.parseObject(msg);

    if (root.containsKey("topic")) {

        if (String("logServer").equals(root["topic"].as<String>())) {

            // check for on: true or false

            logServerId = root["nodeId"];

            Serial.printf("logServer detected!!!\n");

        }

        Serial.printf("Handled from %u msg=%s\n", from, msg.c_str());
    }
}

```

```
}
```

```
}
```

Heltec ESP32 SX1276 (Server Node + LoRa)

Code:

```
#include "painlessMesh.h"

#include <SPI.h>

#include <LoRa.h>

#include "SSD1306.h"

// GPIO5 -- SX1278's SCK

// GPIO19 -- SX1278's MISO

// GPIO27 -- SX1278's MOSI

// GPIO18 -- SX1278's CS

// GPIO14 -- SX1278's RESET

// GPIO26 -- SX1278's IRQ(Interrupt Request)

//OLED pins to ESP32 0.96OLEDGPIOs :

//OLED_SDA -- GPIO4

//OLED_SCL -- GPIO15

//OLED_RST -- GPIO16

#define MESH_PREFIX "HelloMyMesh"

#define MESH_PASSWORD "hellomymeshnetwork"

#define MESH_PORT 5555

SSD1306 display(0x3c, 4, 15);

#define SS 18

#define RST 14

#define DI0 26
```

```
#define BAND 433.175E6 //915E6

painlessMesh mesh;

// Send my ID every 10 seconds to inform others

Task logServerTask(10000, TASK_FOREVER, []() {

DynamicJsonBuffer jsonBuffer;

JsonObject& msg = jsonBuffer.createObject();

msg["topic"] = "logServer";

msg["nodeId"] = mesh.getNodeId();

String str;

msg.printTo(str);

mesh.sendBroadcast(str);

// log to serial

msg.printTo(Serial);

Serial.printf("\n");

});

void setup() {

Serial.begin(115200);

pinMode(25, OUTPUT); //Send success, LED will bright 1 second

while (!Serial);

pinMode(16, OUTPUT);

digitalWrite(16, LOW); // set GPIO16 low to reset OLED

delay(50);

digitalWrite(16, HIGH); // while OLED is running, must set GPIO16 in high

Serial.println("LoRa PainlessMesh Server");

SPI.begin(5, 19, 27, 18);

LoRa.setPins(SS, RST, DIO);

if (!LoRa.begin(BAND)) {
```

```

Serial.println("Starting LoRa failed!");

while (1);

}

Serial.println("LoRa Initial OK!");

// Initialising the UI will init the display too.

display.init();

display.flipScreenVertically();

display.setFont(ArialMT_Plain_10);

display.clear();

display.setTextAlignment(TEXT_ALIGN_LEFT);

display.drawString(10, 5, "Mesh Server Node:");

display.display();

mesh.setDebugMsgTypes( ERROR | CONNECTION | S_TIME );

mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT, STA_AP, WIFI_AUTH_WPA2_PSK, 6 );

mesh.onReceive(&receivedCallback);

mesh.onNewConnection([](size_t nodeId) {

    Serial.printf("New Connection %u\n", nodeId);

});

mesh.onDroppedConnection([](size_t nodeId) {

    Serial.printf("Dropped Connection %u\n", nodeId);

});

// Add the task to the mesh scheduler

mesh.scheduler.addTask(logServerTask);

logServerTask.enable();

}

void loop() {

    mesh.update();
}

```

```

}

void receivedCallback( uint32_t from, String &msg ) {

    String tmp_string = msg.c_str();

    Serial.printf("logServer: Received from %u msg=%s\n", from, tmp_string);

    Serial.println("");

    Serial.println("Sending LoRa packet: "+tmp_string);

    //เมื่อได้รับข้อความจากใน mesh network ก็ส่งต่อค่ามายัง LoRa

    LoRa.beginPacket();

    LoRa.print(tmp_string);

    LoRa.endPacket();

    display.clear();

    display.setTextAlignment(TEXT_ALIGN_LEFT);

    display.drawString(10, 5, "Sending: "+tmp_string.substring(13,19));

    display.drawString(10, 20, "Temp: "+tmp_string.substring(49,55));

    display.drawString(10, 35, "Humid: "+tmp_string.substring(69,75));

    // write the buffer to the display

    display.display();

}

}

```

Heltec ESP32 SX1276 (Gateway Node + LoRa)

Code:

```

#include <SPI.h>

#include <LoRa.h>

#include "SSD1306.h"

#include <WiFi.h>

#include <WiFiClient.h>

#include <PubSubClient.h>

```

```

const char* ssid = "xxxWiFi-SSID";

const char* password = "xxxWiFi Password";

const char* mqtt_server = "xxx.xxx.xxx.xxx"; //<-- IP នៃ Domain នៃ Server MQTT

long lastMsg = 0;

char msg[100];

int value = 0;

WiFiClient espClient;

void callback(char* topic, byte* payload, unsigned int length) {

Serial.print("Message arrived [");

Serial.print(topic);

Serial.print("] ");

for (int i = 0; i < length; i++) {

Serial.print((char)payload[i]);

}

Serial.println();

}

PubSubClient client(mqtt_server, 1883, callback, espClient);

// GPIO5 -- SX1278's SCK

// GPIO19 -- SX1278's MISO

// GPIO27 -- SX1278's MOSI

// GPIO18 -- SX1278's CS

// GPIO14 -- SX1278's RESET

// GPIO26 -- SX1278's IRQ(Interrupt Request)

//OLED pins to ESP32 0.96OLEDGPIOs

//OLED_SDA -- GPIO4

//OLED_SCL -- GPIO15

//OLED_RST -- GPIO16

```

```
SSD1306 display(0x3c, 4, 15);

#define SS 18

#define RST 14

#define DIO 26

#define BAND 433.175E6 //915E6

void setup() {

Serial.begin(115200);

while (!Serial);

pinMode(16, OUTPUT);

digitalWrite(16, LOW); // set GPIO16 low to reset OLED

delay(50);

digitalWrite(16, HIGH); // while OLED is running, must set GPIO16 in high

Serial.println("LoRa Receiver");

SPI.begin(5, 19, 27, 18);

LoRa.setPins(SS, RST, DIO);

if (!LoRa.begin(BAND)) {

Serial.println("Starting LoRa failed!");

while (1);

}

Serial.println("LoRa Initial OK!");

// Initialising the UI will init the display too.

display.init();

display.flipScreenVertically();

display.setFont(ArialMT_Plain_10);

setup_wifi();

client.connect("ESP32Gateway", "joe1", "joe1");

client.setCallback(callback);
```

```
client.subscribe("command");

}

void setup_wifi() {
    delay(10);

    // We start by connecting to a WiFi network

    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected");

    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}

void reconnect() {
    // Loop until we're reconnected

    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");

        // Attempt to connect

        if (client.connect("ESP32Gateway")) {
            Serial.println("connected");

            // Once connected, publish an announcement...

            client.publish("outTopic", "hello world");
        }
    }
}
```

```
// ... and resubscribe

client.subscribe("command");

} else {

Serial.print("failed, rc=");

Serial.print(client.state());

Serial.println(" try again in 5 seconds");

// Wait 5 seconds before retrying

delay(5000);

}

}

}

void loop() {

String tmp_string, tmp_rssi;

if (!client.connected()) {

reconnect();

}

client.loop();

// try to parse packet

int packetSize = LoRa.parsePacket();

if (packetSize) {

// received a packet

Serial.print("Received packet ");

// read packet

while (LoRa.available()) {

//Serial.print((char)LoRa.read());

tmp_string += (char)LoRa.read();

}

}
```

```

Serial.print(tmp_string);

tmp_rssi = LoRa.packetRssi();

// print RSSI of packet

Serial.println(" with RSSI " + tmp_rssi);

display.clear();

display.setTextAlignment(TEXT_ALIGN_LEFT);

display.drawString(10, 0, "Received:");

display.drawString(10, 15, "From: " + tmp_string.substring(13, 19) + " RSSI: " + tmp_rssi);

display.drawString(10, 30, "Temp: " + tmp_string.substring(49, 55));

display.drawString(10, 45, "Humid: " + tmp_string.substring(69, 75));

// write the buffer to the display

display.display();

tmp_string.toCharArray(msg, 100);

Serial.print("Publish message: ");

Serial.println(msg);

client.publish("env", msg); //ส่งข้อมูล Temp + Humidity ออกไปที่ Topic "env"

}

tmp_string = "";

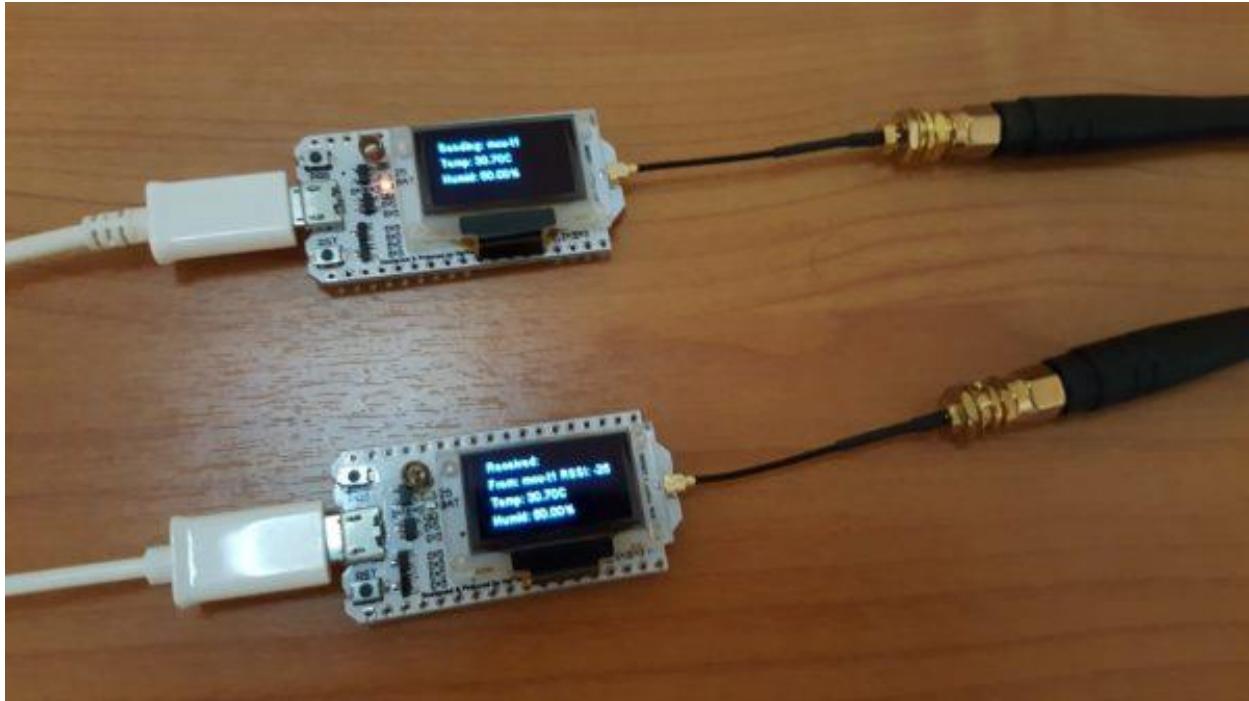
tmp_rssi = "";

}

```

ผลลัพธ์ที่ได้

ถ้าทำการขึ้นตอนที่่าว่ามาทั้งหมดได้ ถึงบรรทัดนี้แล้วผลที่ได้ก็จะเป็นอย่างในคลิปด้านล่างนี้และครับ mcu-t1, mcu-t2, mcu-t3 จะอ่านค่าจาก DHT22 และส่งผ่าน Mesh Network ไปยัง Server Node: Heltec ESP32 LoRa SX1278 จากนั้นเมื่อ Server Node ได้รับข้อมูลความผ่านทาง Mesh Network ก็เอาข้อมูลนั้นส่งต่อผ่านไปยัง LoRa



Video Player

ในอีกฝั่งหนึ่ง Gateway Node: Heltec ESP32 LoRa SX1278 เมื่อได้รับข้อความจาก LoRa ก็จะเอาข้อความนั้น Publish ต่อไปยัง MQTT Server ที่เราได้สร้างกันไว้ดังแต่ตอนที่ 3.5 ถ้ามาจากคลิปวิดีโอจะเห็นว่าระยะเวลาที่ใช้นั้นค่อนข้างเร็วมาก ซึ่ง หลังจากนี้การจะนำข้อมูลจากภายใน Mesh Network ไปใช้งานก็ง่ายแล้ว จะเขียนลง Time Series Database เพื่อเอาไปแสดงผล หรือวิเคราะห์ข้อมูลต่อ ก็ไม่ยากแล้ว

สรุป

เนื้อหาในตอนนี้จะเน้นในเรื่องของ Data Communication อิกรูปแบบหนึ่งนั่นก็คือ LoRa ในการนำมาใช้งานร่วมกับ Mesh Network ซึ่งอย่างที่เกริ่นไปตอนแรกจะรับ ถ้าผ่านตอนที่ 3 มาแล้วก็ขึ้นอยู่กับเราว่าจะไป bridge กับอะไรเพื่อส่งต่อข้อมูลภายใน Mesh Network ไปยังอีก Network หนึ่ง ถ้าตัว Server Node ของเราต่อเข้ากับ Ethernet Module ENC28J60 ก็สามารถเป็น Bridge ในตัวมันเองได้โดยโดยตรงผ่านสาย Lan ก็ได้ ดังนั้นที่เขียนมาข้างบนเลยตอนมาก็เพื่อ喻าให้เข้าใจหลักการณ์ก่อน ที่เหลือก็จะขึ้นอยู่กับการประยุกต์ใช้ให้เหมาะสมนั่นเองครับ ใช่ว่าจำเป็นจะต้องใช้งาน Mesh Network สำหรับงาน IoT ทุกเคส หรือว่า จำเป็นต้องใช้งาน LoRa เพื่อให้ได้ระยะไกลๆ บางครั้งต่อพ่วงอุปกรณ์ของเราเข้ากับ GPRS/LTE Module ก็อาจจะทำให้ไปได้ไกลกว่าถึง ชาขอนของผู้ให้บริการมือถือได้เลยด้วยซ้ำ ยกกล่าว NB-IoT ตอนนี้ด้วย

ส่วนในตอนหน้าที่น่าจะเป็นตอนสุดท้ายของ Series เรื่อง ESP8266/ESP32 กับ Painlessmesh ก็จะเป็นตอนปลิกย่อyle เพราะ ในตอนที่ 3.5 ได้สอนในเรื่องของการติดตั้ง MQTT Server บน Google Cloud ไปแล้ว ก็คงปิด Series ด้วยการใช้งาน Telegraf, InfluxDB และ Grafana ไปเลย เพื่อที่จะทำให้ข้อมูลที่ไหลมาจาก Sensor Node ต่างๆ ไปอยู่ในรูปของ Data Visualization ที่สวยงามและเข้าใจง่าย

เรื่องเล่าท้ายตอน

ตอนนี้เป็นตอนที่ใช้พัลจังการและเวลาในการเขียนเอกสารจริงๆ ทั้งเรื่องการรออุปกรณ์ งานที่ทำอยู่ คลาสเรียนที่เพิ่มเข้ามา ต้องเขียนสารนิพนธ์ด้วย ยังไม่รวมถึงวันศุกร์ที่จะถึงนี้จะมีการประชุมผู้ถือหุ้น อีก T_T แต่อาจจริงๆ เขียนเล่าเรื่องพวknี้สักหน่อยก็เหมือนกันได้ทบทวนสิ่งที่ค้างอาจได้ทดลองทำ มา มีคนถามว่าเขียนแล้วได้อะไร อย่างแรกเลยก็คงอ่าไว้เป็น Reference เพื่อไว้ทบทวนนั้นแหล่งครับ เพราะไม่จืดผ่านไปนานๆ ไม่ได้นะๆ ว่า ตอนนี้เราทำยังไง ขั้นตอนไหนก่อนหลัง แล้วก็เมื่อมีคนรู้เรื่องพวknี้มากๆ เรา ก็จะสนุกมากขึ้น มีคนถาม มีคนศึกษา มีคนทดลองต่อ บางคนอาจไปต่อยอดแล้วก็มาพูดคุยกัน เพราะจริงๆ แล้วเราไม่ได้รู้ทุกๆ เรื่อง บางเรื่องที่เขียนผิดอาจจะเขียนใหม่ก็ได้ หรือบางท่านหลังไม่ก้าวขึ้นมาให้เขียนก็ถ้ามีโอกาสก็จะไปหาข้อมูลมาแชร์กัน วันนี้ก็ขอจบตอนนี้ก่อนละกันครับ

- ESP Mesh Network 5 - <https://meetjoeblog.com/2018/08/30/esp8266-esp32-mesh-network-ep5-influxdb-grafana/>

MEET JOE'S BLOG FANPAGE



POSTED IN MICROCONTROLLER, TECHNOLOGY

ESP8266 / ESP32 & Mesh Network ตอนที่ 5: InfluxDB & Grafana

JOE D.S. 30/08/2018 [LEAVE A COMMENT](#) ON ESP8266 / ESP32 & MESH NETWORK ตอนที่ 5: INFLUXDB & GRAFANA

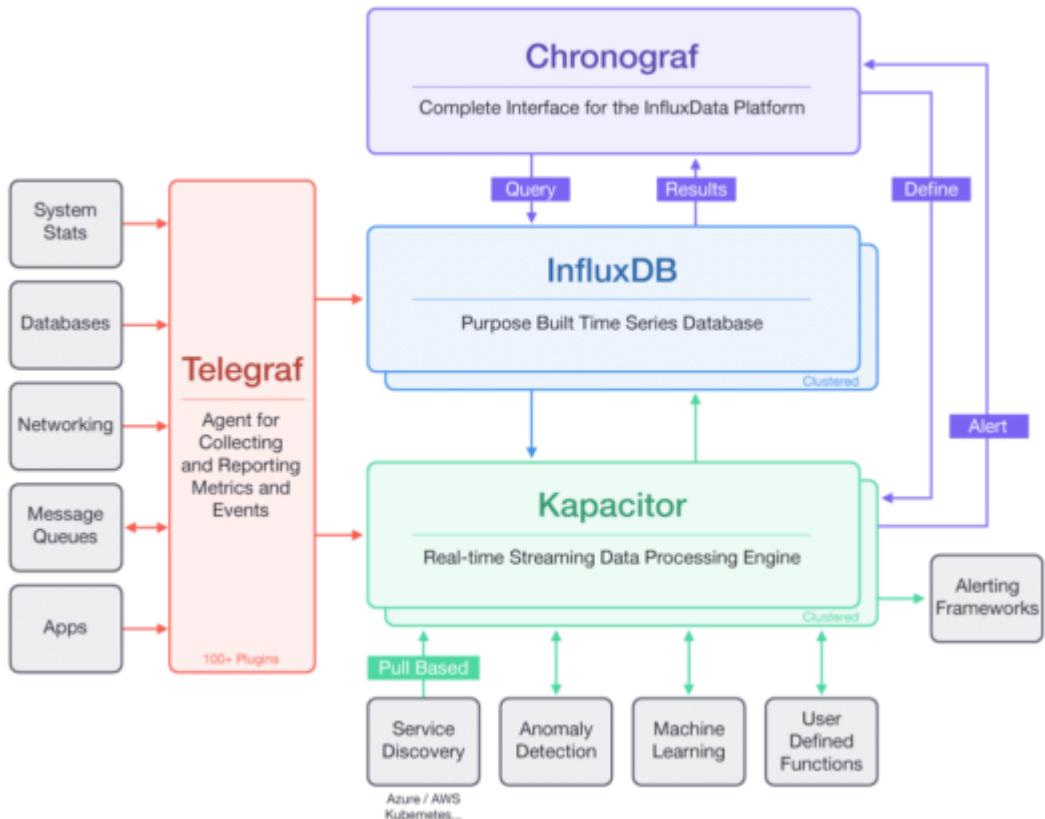
จริงๆ เมื่อหาหลักของ Painlessmesh นั้น จะไปตั้งแต่ตอนที่ 3 แล้ว ที่เหลือจะเป็นลักษณะของการประยุกต์ใช้มากกว่า ไม่ว่าจะเป็นการ Bridge ด้วย LoRa หรือการติดตั้ง MQTT Server บน Google Cloud ซึ่งถ้าันจากตอนล่าสุดตอนที่ 4 ถึงตอนนี้ก็ผ่านไป 4 เดือนละ ก็เลยจะมาเจน Series ของ Painlessmesh นี้ตามที่เคยเกริ่นไว้ว่าจะมีทั้งหมด 5 ตอนด้วยกัน โดยตอนนี้จะเป็นเหมือนตอนแรก ปลายทางละ เพราะจะเป็นการนำข้อมูลต่างๆ นั้น ไปเก็บอยู่บน Time Series Database ที่ชื่อว่า InfluxDB และนำไปแสดงผลบน Dashboard ของ Grafana

ก่อนเข้าไปสู่การติดตั้ง InfluxDB ซึ่งเป็น Time Series Database (TSDB) โดยที่เป็นการเก็บข้อมูล แนวโน้มเก็บ log ที่เก็บไป เรื่อยๆ เรียงตามวันที่และเวลาต่อๆ กันไป เช่น ข้อมูลราคาหุ้น ข้อมูลอุณหภูมิ ความชื้นในแต่ละช่วงเวลา ซึ่ง InfluxDB ก็เป็นหนึ่งใน TSDB ที่ดูจะ Optimize มาให้ทำงานในลักษณะนี้ โดยเน้นการทำงานที่ง่ายเร็ว ตัดพาก business logic ด้วย module ข้างนอกแทน ขณะนี้ในการโอนข้อมูลเข้ามาเพื่อเขียนลงฐานข้อมูล ก็จะเริ่มมากและใช้พื้นที่น้อยกว่าอีกด้วย

InfluxDB คือที่ส่วนที่เป็น Commercial และ Opensource ซึ่งข้อแตกต่างหลักๆ ของ version ที่เสียเงินกับ Opensource นั้น ก็จะเป็นในเรื่องของการ support และคือเรื่องของ High Availability หรือการ Scale out ระบบเพื่อรับการทำ Clustering ระหว่างหลาย node ของ InfluxDB

InfluxCloud Great Place to Start	InfluxEnterprise Ready to Scale	TICK Stack Open Source
<ul style="list-style-type: none"> • Open Source Core • Extensible • Support for Regular and Irregular Data • High Availability (Clustering) • Scalability (Clustering) • Advanced Backup and Restore • Complete Platform Support • Managed by InfluxData • Runs on AWS • Runs On Premises • Preset Configurations or Custom (Contact Sales) 	<ul style="list-style-type: none"> • Open Source Core • Extensible • Support for Regular and Irregular Data • High Availability (Clustering) • Scalability (Clustering) • Advanced Backup and Restore • Complete Platform Support • Managed by InfluxData • Runs on any Cloud • Runs On Premises • Custom Configurations – Contact Sales 	<ul style="list-style-type: none"> • Open Source Core • Extensible • Support for Regular and Irregular Data • High Availability (Clustering) • Scalability (Clustering) • Advanced Backup and Restore • Complete Platform Support • Managed by InfluxData • Runs on Cloud • Runs On Premises • You are on your own to run it wherever and however you like

แต่วันนี้เราจะมาดูในส่วนของ Opensource กันซึ่งทาง Influxdata นั้นใช้ชื่อว่า TICK Stack ซึ่งประกอบไปด้วย 4 Module ตัวยังกันคือ Telegraf, InfluxDB, Chronograf และ Kapacitor



โดยที่การทำงานก็จะเป็นอย่างสูงที่แสดงอยู่ด้านบนเลยครับ

- **Telegraf** จะเป็นตัวค่าทางในการรับข้อมูลจาก Source ต่างๆเข้ามาไม่ว่าจะเป็น Database อื่นๆหรือแม้กระทั่ง MQTT Message และรวมถึงจึงข้อมูลจาก InfluxDB และส่งกลับไปยังระบบอื่นด้วย ทำหน้าที่เหมือน ETL
- **InfluxDB** ก็เป็นหัวใจของ TICK Stack นี้เลยซึ่งก็คือ Time Series Database ของเรานั่นเอง เน้นเก็บข้อมูลอย่างต่อเนื่อง
- **Chronograf** อันนี้จะเป็น GUI ที่ใช้ในการ Manage InfluxDB รวมถึงการทำ Dashboard ลักษณะเดียวกับ Grafana (แต่ใช้ภาษา Go แทนภาษา Golang)
- **Kapacitor** จะเป็น่วนที่ทำในเรื่องของ Data Processing ต่างๆ เช่น เมื่อมีข้อมูลการรอสายโทรศัพท์ของคุณก็ให้โทรเข้ามายัง Call Center เกินเท่านี้ให้ทำการ Alert แจ้งต่อไปยังหัวหน้างาน ซึ่งก็สามารถนำ Module การวิเคราะห์ต่างๆเข้ามาใช้ได้ ไม่ว่าจะเป็นจากการตั้งค่าเอง หรือ Machine Learning ที่ได้

InfluxDB Setup

โดยการติดตั้งนั้นจะเริ่มจาก InfluxDB กันก่อน โดยการเปิดหน้า SSH Command Windows ของ Google Cloud แล้วพิมพ์คำสั่งต่อตามด้านล่างนี้เลยครับ โดยปกติจะเป็น Download & Install -> Start Service -> Verify (ใครที่อ่านตรงนี้แล้วอาจจะต้องข้อนอกลับไปอ่านที่ [ตอนแรก 3.5-1 การติดตั้ง MQTT Server](#))

Download & Install

```
wget https://dl.influxdata.com/influxdb/releases/influxdb_1.4.0_amd64.deb
```

```
sudo dpkg -i influxdb_1.4.0_amd64.deb
```

Start InfluxDB

```
sudo systemctl start influxdb
```

Verify

```
curl "http://localhost:8086/query?q=show+databases"
```

ด้วย InfluxDB ที่เราเพิ่งติดตั้งไปทำงานได้อย่างถูกต้องแล้วหละก็ จะแสดงผลลัพธ์ตามนี้ครับ

```
{"results":[{"statement_id":0,"series":[{"name":"databases","columns":["name"],"values":[["_internal"]]}]}]}
```

Kapacitor Setup

มาก็ Module ที่สองที่เราจะทำการติดตั้งกัน เพื่อใช้ในการทำ Data Processing ซึ่งถ้าใครไม่ได้ใช้งานก็สามารถข้ามไปที่ Module กัดไปได้ครับ

Download & Install

```
wget https://dl.influxdata.com/kapacitor/releases/kapacitor_1.4.0_amd64.deb
```

```
sudo dpkg -i kapacitor_1.4.0_amd64.deb
```

Start Kapacitor

```
sudo systemctl start kapacitor
```

Verify

```
kapacitor list tasks
```

ชี้งค์ที่ติดตั้งและ Service Start เรียบร้อยแล้ว ผลลัพท์ที่ได้จะเป็นดังนี้ครับ

ID Type Status Executing Databases and Retention Policies

Telegraf Setup

Module Telegraf นี้เรียกได้ว่าเป็นอีกหัวใจหลักเดียวกับได้รับ เพราะจะเป็นตัวที่เก็บรวบรวม ดึงข้อมูลจาก Source ต่างๆเพื่อที่จะไปยัง InfluxDB ในตัวอย่างนี้เราจะเรียกเก็บ System Stat ของ VM Instance ของเรา กันด้วยเพื่อคุณ load ของ CPU / RAM กัน

Download & Install

```
wget https://dl.influxdata.com/telegraf/releases/telegraf_1.4.3-1_amd64.deb
```

```
sudo dpkg -i telegraf_1.4.3-1_amd64.deb
```

Start Service

```
sudo systemctl start telegraf
```

Verify

ขั้นตอนนี้จะแตกต่างจาก Module อื่นๆ อย่างที่เกริ่นกันไว้ เราจะดึงข้อมูล System Stat ของระบบของเรา โดยการทำงานของ Telegraf จะทำงานผ่าน Plug-in ต่างๆ โดยขึ้นแรกให้เปิดไฟล์

`/etc/telegraf/telegraf.conf`

จากนั้นให้คุณส่วนของ Output Plugins (จะใช้โปรแกรม vi , pico , nano อันนี้ก็แล้วแต่สะดวกทันเดียวกัน)

`[[outputs.influxdb]]`

`## The full HTTP or UDP endpoint URL for your InfluxDB instance.`

`## Multiple urls can be specified as part of the same cluster,`

`## this means that only ONE of the urls will be written to each interval.`

`# urls = ["udp://localhost:8089"] # UDP endpoint example`

`urls = ["http://localhost:8086"] # required`

`## The target database for metrics (telegraf will create it if not exists).`

`database = "telegraf" # required`

```

## Retention policy to write to. Empty string writes to the default rp.

retention_policy = ""

## Write consistency (clusters only), can be: "any", "one", "quorum", "all"

write_consistency = "any"

## Write timeout (for the InfluxDB client), formatted as a string.

## If not provided, will default to 5s. 0s means no timeout (not recommended).

timeout = "5s"

# username = "telegraf"

# password = "metricsmetricsmetricsmetrics"

## Set the user agent for HTTP POSTs (can be useful for log differentiation)

# user_agent = "telegraf"

## Set UDP payload size, defaults to InfluxDB UDP Client default (512 bytes)

# udp_payload = 512

```

และในส่วนของ Input Plugins นั้นที่ควรจะมีหน้าตาลักษณะแบบนี้ ในการจึงค่า System Stat ออกมานั่นเป็น Output ให้กับ InfluxDB

```

# Read metrics about cpu usage

[[inputs.cpu]]

## Whether to report per-cpu stats or not

percpu = true

## Whether to report total system cpu stats or not

totalcpu = true

## If true, collect raw CPU time metrics.

collect_cpu_time = false

# Read metrics about disk usage by mount point

[[inputs.disk]]

## By default, telegraf gather stats for all mountpoints.

```

```
## Setting mountpoints will restrict the stats to the specified mountpoints.

# mount_points = ["/"]

## Ignore some mountpoints by filesystem type. For example (dev)tmpfs (usually
## present on /run, /var/run, /dev/shm or /dev).

ignore_fs = ["tmpfs", "devtmpfs"]

# Read metrics about disk IO by device

[[inputs.diskio]]

## By default, telegraf will gather stats for all devices including
## disk partitions.

## Setting devices will restrict the stats to the specified devices.

# devices = ["sda", "sdb"]

## Uncomment the following line if you need disk serial numbers.

# skip_serial_number = false

# Get kernel statistics from /proc/stat

[[inputs.kernel]]

# no configuration

# Read metrics about memory usage

[[inputs.mem]]

# no configuration

# Get the number of processes and group them by status

[[inputs.processes]]

# no configuration

# Read metrics about swap memory usage

[[inputs.swap]]

# no configuration

# Read metrics about system load & uptime

[[inputs.system]]
```

```
# no configuration
```

ชั้งจริงๆแล้วที่ในส่วนของ Output/Input Plugins นั้นที่เป็นค่า Default ที่ดีดังมาพร้อมกับ Telegraf อยู่แล้วครับ เสร็จแล้วที่
ทดสอบรันคำสั่งนี้กันดู

```
curl "http://localhost:8086/query?q=select+*+from+telegraf.cpu"
```

ถ้า Telegraf ทำงาน ก็จะแสดงผลข่าวพิ้งค์ออกมาเป็นรูปแบบ JSON ให้ได้เห็นกันเต็มหน้าจอเลย

Chronograf Setup

มาถึง Module สุดท้ายของ TICK Stack กันเลยครับสำหรับหน้า GUI เพื่อใช้ในการ Config และแสดงผล สำหรับผู้ผลลัพธ์ที่เราได้กำหนดไว้ เช่น Grafana น่าจะสะดวกกว่า แต่ไหนๆ ก็ให้ลองติดตั้งกันไปก่อนครับ เป็นทางเลือก

Download & Install

```
wget https://dl.influxdata.com/chronograf/releases/chronograf_1.4.0.0_amd64.deb
```

```
sudo dpkg -i chronograf_1.4.0.0_amd64.deb
```

Start Service

```
sudo systemctl start chronograf
```

Verify

ขั้นตอนนี้ก็ให้เข้าไปที่

<http://localhost:8888>

โดยเปลี่ยน local host เป็น ip หรือ hostname ของเราครับ ก็จะได้หน้า Web GUI ของ Chronograf ดังแสดงในรูปด้านล่างนี้ สำหรับใครที่ใช้ Google Cloud ก็อย่าลืมเพิ่มไฟวอลล์ในส่วนของ Firewall Rule ให้ Allow Port 8888 ด้วยนะครับ วิธีการก็หาดูได้จากตอนที่ 3.5 ได้เลย

Add a New Source

Connection String: http://localhost:8086

Name: Influx 1

Username:

Password:

Telegraf Database: telegraf

Make this the default source

+ Add Source

หลังจากนั้นก็คลิกที่ปุ่ม +Add Source เลขครับ ค่าที่อยู่ให้ลง Default ไว้เนื่องจากในการติดตั้งครั้งแรกนั้น Default Security ของ InfluxDB นั้นจะไม่มี Username/Password ซึ่งเดียวเราจะได้มาม config กันในช่วงต่อไปครับ ซึ่งหลังจาก Add Source แล้วก็จะเข้าสู่หน้าอัดໄປ ให้ทำการเลือก Host ที่เราใช้ Plugin ของ Telegraf ในการดึง System Stat มาใส่ไว้ใน InfluxDB

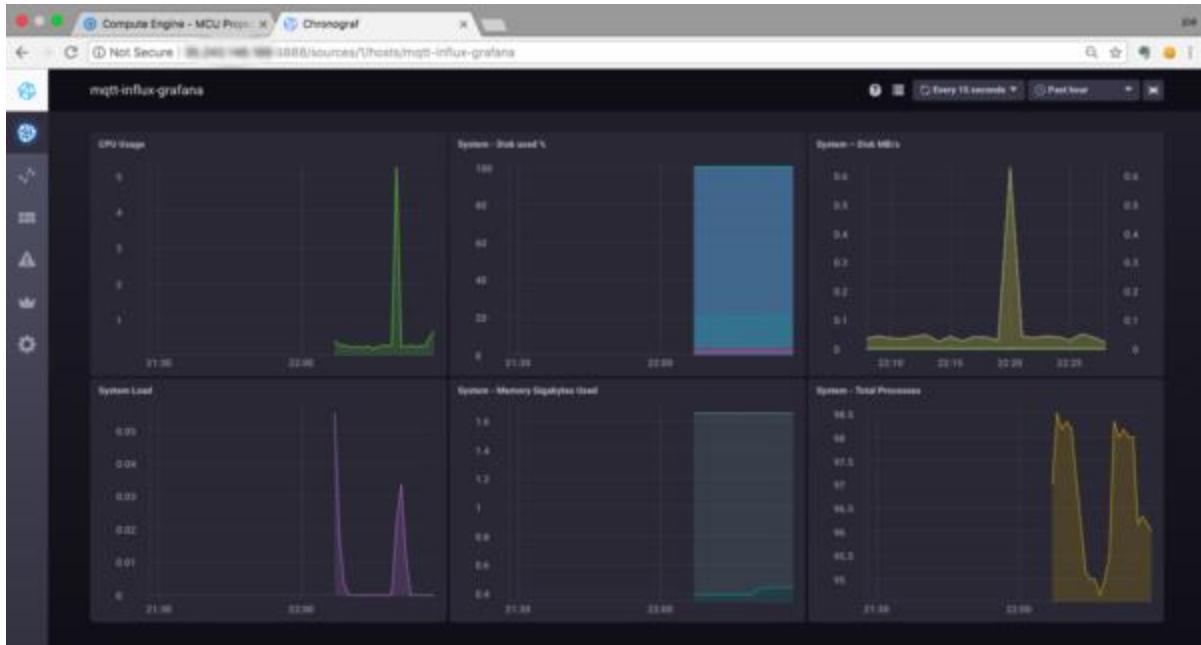
Host List

1. คลิกเพื่อเลือก Host

2. เลือก Host

Host	Status	CPU	Load	Apps
mqtt-influx-grafana	●	0.78%	0.01	system

ถ้าทุกอย่างทำงานได้อย่างถูกต้องเราจะได้เห็น Dashboard ที่ใช้ในการแสดงผล System Stat ที่ได้จากการถึงข้อมูลผ่านทาง Plugin Module ของ Telegraf และโอนข้อมูลที่ได้ไปไว้ใน InfluxDB และนำมาแสดงผลบน Chronograf



แต่ยังไม่จบครับ เพราะ TICK Stack นี้มีอยู่ด้วยกัน 4 Module นี่เราเพิ่งจะใช้งานไปแค่ 3 Module เอง ยังเหลือในส่วนของ Kapacitor ที่ใช้ในการทำ Data Processing/Analysis และ Alert ต่างๆ จากหน้า Dashboard System Stat ของเรา ก็คลิกที่ Config เพื่อ connect ไปยัง Kapacitor แต่พอจะขอหยุดไว้ท่านนี้ก่อนดีกว่า เพราะถ้าเราทำใน Grafana นี้ในส่วนของ Dashboard และ Alert เราสามารถทำได้จาก Grafana ในที่เดียวกันเลย เราจึงมักเห็นการจับคู่กันเฉพาะในส่วนของ MQTT -> Telgraf-> InfluxDB->Grafana เรียกได้ว่าเป็นสูตรสำเร็จสำหรับการเก็บข้อมูลมาแสดงผลของงาน IOT เลยก็วัน

แต่ก่อนอื่น เรามาเซทระบบของเราให้เข้าที่เข้าทางกันก่อนดีกว่า เพราะก่อนหน้านี้ที่เรา Install InfluxDB ไปค่า Default ของระบบยังไม่มีในส่วนของ User/Password และการ Authen ต่างๆ ขั้นตอนแรกเราจะมาสร้าง User Admin ของระบบกันด้วยคำสั่ง influx และตามด้วยคำสั่งด้านล่างนี้ครับ

```
CREATE USER "admin" WITH PASSWORD 'admin_passwd' WITH ALL PRIVILEGES
```

ในส่วนของ admin_passwd ก็เปลี่ยนเป็นตามใจชอบได้เลยครับ

หลังจากนี้ก็ใช้คำสั่ง show users ในการเรียกดู user ทั้งหมดที่มีอยู่ ตามรูปด้านล่างนี้

```
Secure | https://ssh.cloud.google.com/projects/mcu... mqtt-influx-grafana: /etc/influxdb
mqqt-influx-grafana:/etc/influxdb$ influx
Connected to http://localhost:8086 version 1.4.0
InfluxDB shell version: 1.4.0
> CREATE USER "admin" WITH PASSWORD '*****' WITH ALL PRIVILEGES
>
>
> SHOW USERS
user admin
-----
admin true
>
```

โดยทำการแก้ไขไฟล์ /etc/influxdb/influxdb.conf ในส่วนของ [auth-enabled] ให้เป็น true ซึ่งหน้าตาของ config ที่แก้ไขจะเหมือนกับด้านล่างนี้

```
[http]
# Determines whether HTTP endpoint is enabled.
#enabled = true

# The bind address used by the HTTP service.
#bind-address = ":8086"

# Determines whether user authentication is enabled over HTTP/HTTPS.
#auth-enabled = true

# The default realm sent back when issuing a basic auth challenge.
#realm = "InfluxDB"

# Determines whether HTTP request logging is enabled.
#log-enabled = true

# Determines whether detailed write logging is enabled.
#write-tracing = false

# Determines whether the pprof endpoint is enabled. This endpoint is used for
# troubleshooting and monitoring.
#pprof-enabled = true

# Determines whether HTTPS is enabled.
#https-enabled = false
```

เมื่อเพิ่ม user admin และแก้ไขไฟล์ influxdb.conf แล้ว จากนั้นก็ restart service ของ influxdb ด้วยคำสั่ง

```
sudo systemctl restart influxdb
```

ก็เป็นอันเสร็จสิ้นการปรับแต่ง influxdb ของเราให้มีการ Authen ก่อนเข้าใช้งาน ถ้าทดสอบ list user ในระบบค้างค่าสั่ง show users โดยที่ไม่มีการใส่ username/password ก็จะแสดง error แบบด้านล่างนี้ครับ

```
Secure | https://ssh.cloud.google.com/projects/mcu-office-home-...@mqtt-influx-grafana: /etc/influxdb
[...]
influxdb:~$ influx -username admin -password
Connected to http://localhost:8086 version 1.4.0
InfluxDB shell version: 1.4.0
> show users
ERR: unable to parse authentication credentials
Warning: It is possible this error is due to not setting a database.
Please set a database with the command "use <database>".
> 
```

ดังนั้นการใช้งานหลังจากนี้ที่หน้า command line ให้เพิ่มคำสั่งในส่วนของ -username และ -password เข้าไปก่อนก็จะสามารถใช้งานได้ครับ

```
Secure | https://ssh.cloud.google.com/projects/mcu-office-home-...@mqtt-influx-grafana: /etc/influxdb
[...]
influxdb:~$ influx -username admin -password
Connected to http://localhost:8086 version 1.4.0
InfluxDB shell version: 1.4.0
> show users
user admin
admin true
> 
```

คราวนี้เราจะมาแก้ไข config ในส่วนของ Telegraf กันเพื่อให้ไปดึงข้อมูลจาก MQTT Server ของเราที่ได้สร้างไว้กันตั้งแต่ตอนที่ 3.5 กัน โดยแก้ไขไฟล์

```
/etc/telegraf/telegraf.conf
[...]
[[inputs.mqtt_consumer]]
  # ตามด้านล่างนี้เลยครับ
[[inputs.mqtt_consumer]]
  servers = ["xxx.xxx.xxx.xxx:1883"] #<-- IP ของ mqtt server
  qos = 0
  connection_timeout = "30s"
  topics = ["env"] #<-- topic ที่เราต้องการดึงข้อมูลมา
  username = "xxx" #<-- username ของ mqtt server
```

```
password = "xxx" #<-- password ของ mqtt server
```

```
data_format = "influx" #<-- รูปแบบของ Data ที่ต้องการคิ่ง
```

มาตรฐานในส่วนของ influx line format กันบ้าง ซึ่งรูปแบบของข้อมูลที่ Telegraf รองรับนั้นก็มีหลากหลายถึง JSON ด้วย สำหรับในส่วนนี้จะมาอธิบายในส่วนของ InfluxDB 2 ตัวด้วยกัน โดยตัวแรกก่อนหน้านี้คือ Default ที่มาจาก การติดตั้ง Telegraf ซึ่งเป็นข้อมูลพวาก System Stat อีกอันก็จะเป็นข้อมูลที่เราได้มาจาก Node ที่ส่งข้อมูลมาที่ MQTT Server (ในที่นี้ จำลองส่งข้อมูลอุณหภูมิเข้ามา) ฉะนั้นเราต้องบอก Telegraf ว่าข้อมูลไหนจะเก็บลงฐานข้อมูลไหน โดยการใช้ namedrop, namepass ตามตัวอย่างด้านล่างนี้ครับ

```
[[outputs.influxdb]] #<-- ใช้เขียนข้อมูล System Stat ลง InfluxDB โดยเก็บที่นิยามชื่อ telegraf
```

```
urls = ["http://localhost:8086"] # required
```

```
database = "telegraf" # required
```

```
retention_policy = ""
```

```
write_consistency = "any"
```

```
timeout = "5s"
```

```
username = "xxx" #<-- username ของ InfluxDB
```

```
password = "xxx" #<-- password ของ InfluxDB
```

```
namedrop = ["env*"] #<-- ให้ drop ข้อมูลที่ขึ้นต้นด้วย env ซึ่งเป็น measurement
```

```
[[outputs.influxdb]] #<-- ใช้เขียนข้อมูล System Stat ลง InfluxDB โดยเก็บที่นิยามชื่อ telegraf
```

```
urls = ["http://localhost:8086"]
```

```
database = "envdb" # required
```

```
retention_policy = ""
```

```
write_consistency = "any"
```

```
timeout = "5s"
```

```
username = "xxx" #<-- username ของ InfluxDB
```

```
password = "xxx" #<-- password ของ InfluxDB
```

```
namepass = ["env*"] #<-- ข้อมูลที่ขึ้นต้นด้วย env ซึ่งเป็น measurement ให้เขียนลงฐานข้อมูล
```

จากนั้นทำการ Restart Service

```
$ systemctl daemon-reload
$ sudo systemctl restart telegraf
$ sudo systemctl status telegraf
```

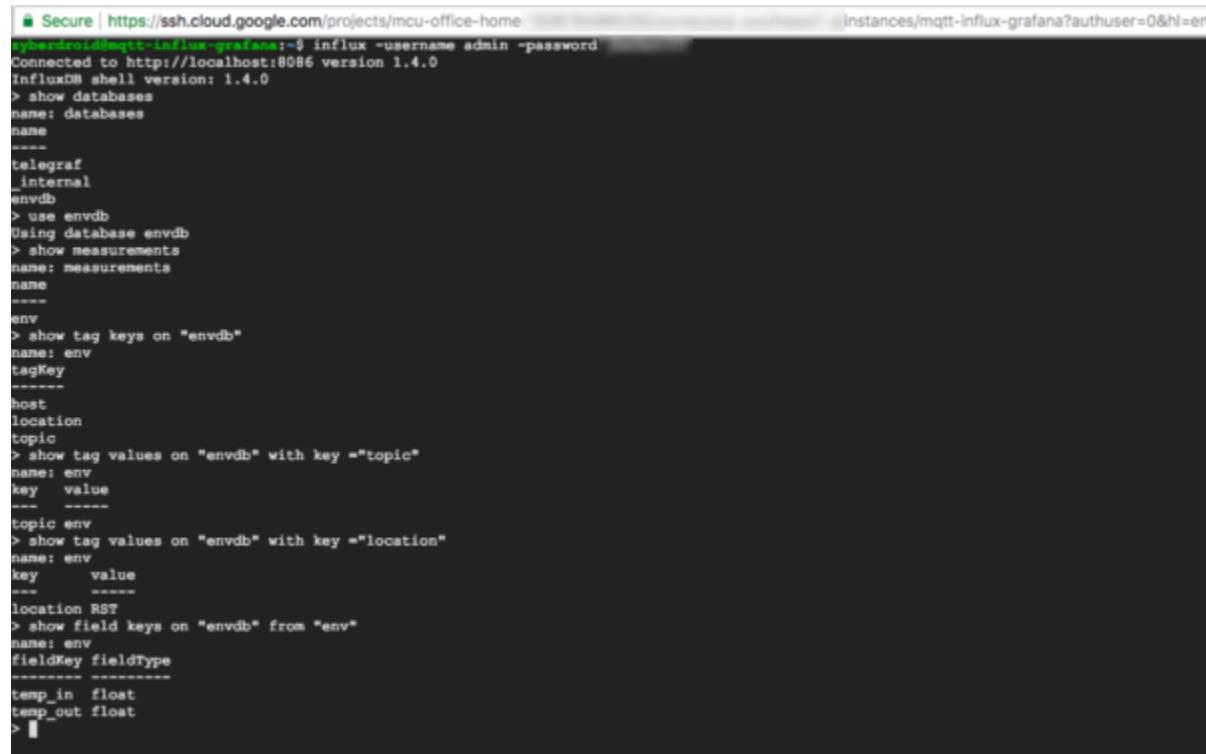
ส่วน influx line format นั้นก็ง่ายที่สุดเลยครับ ยกตัวอย่าง ตามด้านล่างเป็นข้อมูลที่ Nodemcu ส่งไปที่ MQTT Server

`env,location=RST temp_in=25.60,temp_out=25.40`

หรือถ้าใครยังไม่มีข้อมูลส่งเข้าไปที่ MQTT Server ก็ทำการ inset ข้อมูลเองโดยใช้คำสั่งนี้ที่หน้า command line ของ influx เลยครับ

`INSERT env,location=RST temp_in=25.60,temp_out=25.40`

ซึ่งการเก็บข้อมูลใน InfluxDB จะแยกเป็น Measurement, TAG Key – TAG Value และ Field Key – Field Value จากตัวอย่างข้อมูลข้างบนนี้ มองเห็นหน้า command line แล้วใช้คำสั่งแสดงค่าดังต่อไปนี้



```
Secure | https://ssh.cloud.google.com/projects/mcu-office-home
hybridnode@mqtt-influx-grafana:~$ influx -username admin -password
Connected to http://localhost:8086 version 1.4.0
InfluxDB shell version: 1.4.0
> show databases
name: databases
name
-----
telegraf
_internal
envdb
> use envdb
Using database envdb
> show measurements
name: measurements
name
-----
env
> show tag keys on "envdb"
name: env
tagKey
-----
host
location
topic
> show tag values on "envdb" with key ="topic"
name: env
key   value
-----+
topic env
> show tag values on "envdb" with key ="location"
name: env
key   value
-----+
location RST
> show field keys on "envdb" from "env"
name: env
fieldKey fieldType
-----+
temp_in float
temp_out float
> |
```

Measurement มีแค่ตัวเดียวคือ env

TAG Key ประกอบไปด้วย host, location, และ topic

และ temp_in, temp_out เป็น Field Key

Grafana

ขั้นตอนการติดตั้ง Grafana ซึ่งโดยปกติสามารถติดตั้งได้ผ่าน apt-get install แต่บางเครื่องนั้นอาจจะไม่ได้มีชื่อรุ่น repository ของ Grafana อยู่ด้วยจะนั้นเราต้องทำการเพิ่มรายการของ Grafana เข้าไปในลิสต์ก่อนที่จะทำการติดตั้ง

```
curl https://packagecloud.io/gpg.key | sudo apt-key add -
```

จากนั้นตามคำย้ำสั่ง

```
sudo add-apt-repository "deb https://packagecloud.io/grafana/stable/debian/ stretch main"
```

```
sudo apt-get update
```

เมื่อทำการ Update List เรียบร้อยแล้วก็สามารถทำการติดตั้งได้ด้วยคำสั่ง

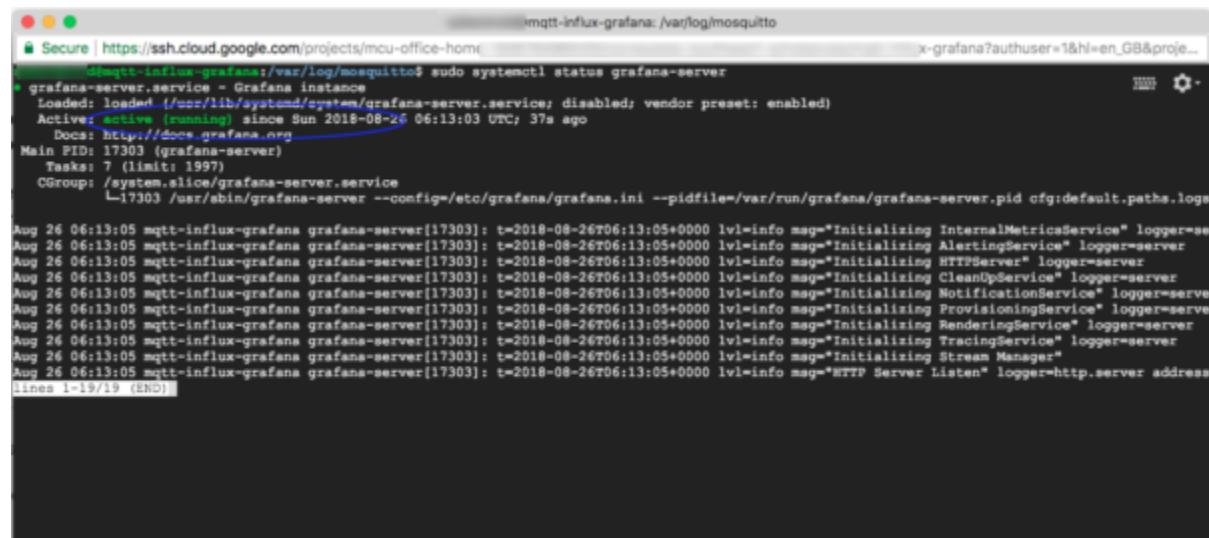
```
sudo apt-get install grafana
```

หลังจากที่ติดตั้งเสร็จแล้วก็ทำการ Start Service และเช็คสถานะกันหน่อย

```
sudo systemctl start grafana-server
```

```
sudo systemctl status grafana-server
```

ถ้า Service ของ Grafana ทำงานเป็นปกติจะขึ้นข้อมูลแสดงสถานะดังนี้ครับ



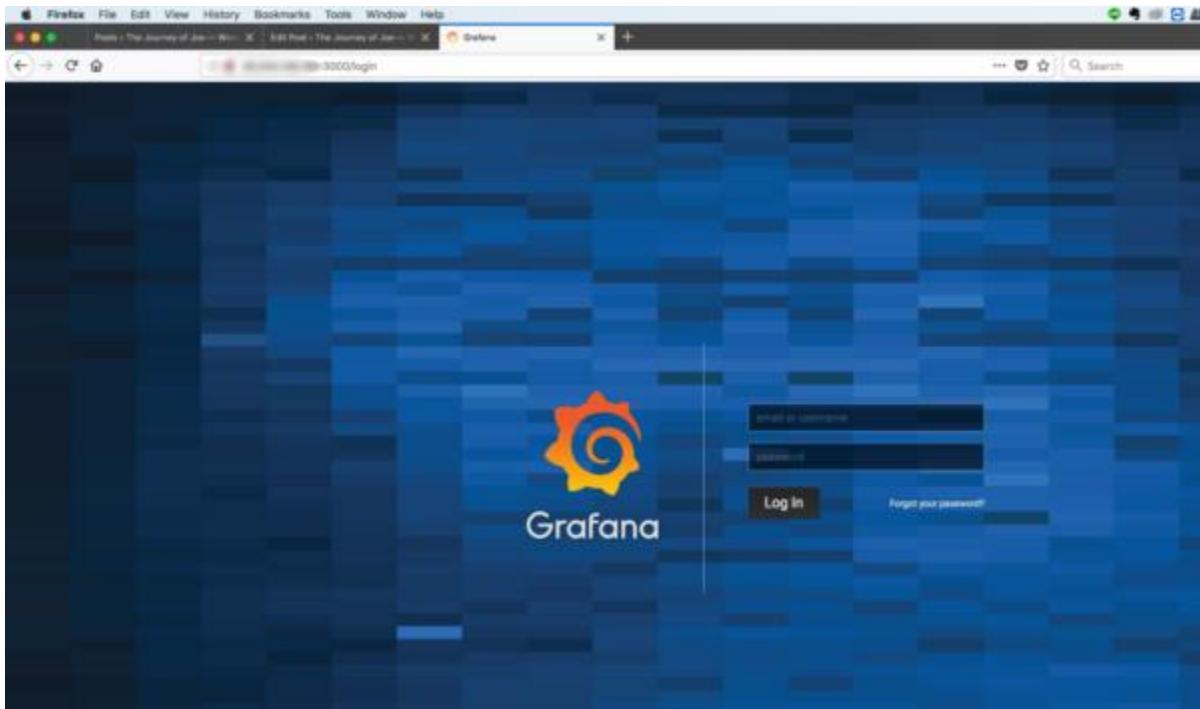
```
mqtt-influx-grafana:~/.var/log/mosquitto
Secure | https://ssh.cloud.google.com/projects/mcu-office-home/x-grafana?authuser=1&hl=en_GB&proj...
 mqtt-influx-grafana:~/.var/log/mosquitto$ sudo systemctl status grafana-server
● grafana-server.service - Grafana instance
   Loaded: loaded (/usr/lib/systemd/system/grafana-server.service; disabled; vendor preset: enabled)
   Active: active (running) since Sun 2018-08-26 06:13:03 UTC; 37s ago
     Docs: http://docs.grafana.org
Main PID: 17303 (grafana-server)
   Tasks: 7 (limit: 1997)
  CGroup: /system.slice/grafana-server.service
          └─17303 /usr/sbin/grafana-server --config=/etc/grafana/grafana.ini --pidfile=/var/run/grafana/grafana-server.pid cfg:default.paths.logs

Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: t=2018-08-26T06:13:05+0000 lvl=info msg="Initializing InternalMetricsService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: t=2018-08-26T06:13:05+0000 lvl=info msg="Initializing AlertingService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: t=2018-08-26T06:13:05+0000 lvl=info msg="Initializing HTTPServer" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: t=2018-08-26T06:13:05+0000 lvl=info msg="Initializing CleanupService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: t=2018-08-26T06:13:05+0000 lvl=info msg="Initializing NotificationService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: t=2018-08-26T06:13:05+0000 lvl=info msg="Initializing ProvisioningService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: t=2018-08-26T06:13:05+0000 lvl=info msg="Initializing RenderingService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: t=2018-08-26T06:13:05+0000 lvl=info msg="Initializing TracingService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: t=2018-08-26T06:13:05+0000 lvl=info msg="Initializing Stream Manager"
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: t=2018-08-26T06:13:05+0000 lvl=info msg="HTTP Server Listen" logger=http.server address=0.0.0.0 port=3000
lines 1-19/19 (END)
```

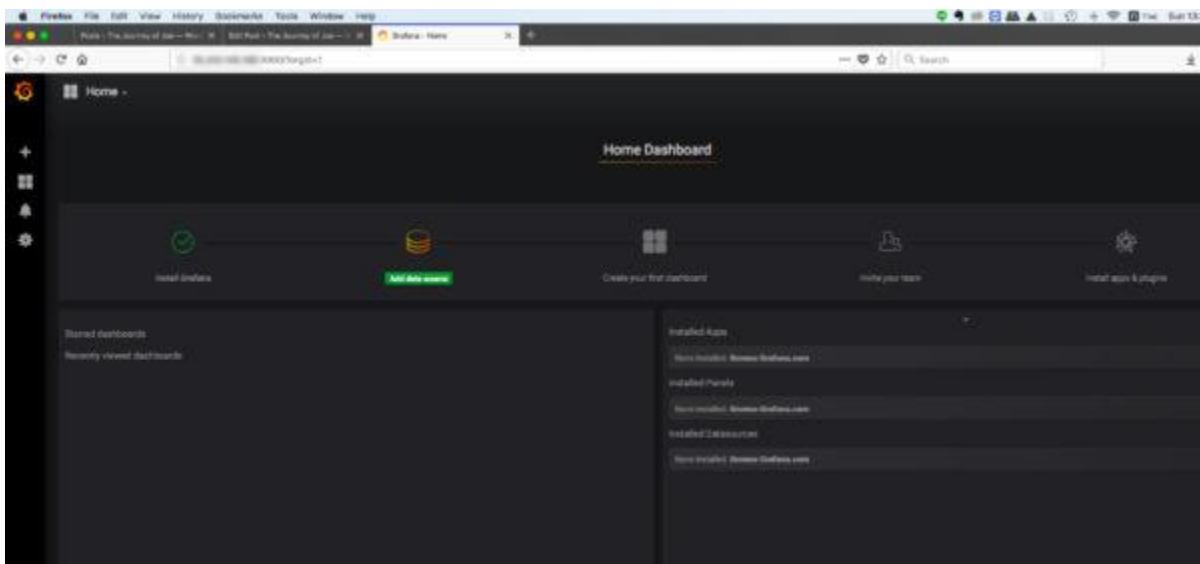
จากนั้นก็ตั้งค่าให้ Grafana เริ่มต้นทุกครั้งถ้ามีการรีบูตด้วยคำสั่ง

```
sudo systemctl enable grafana-server
```

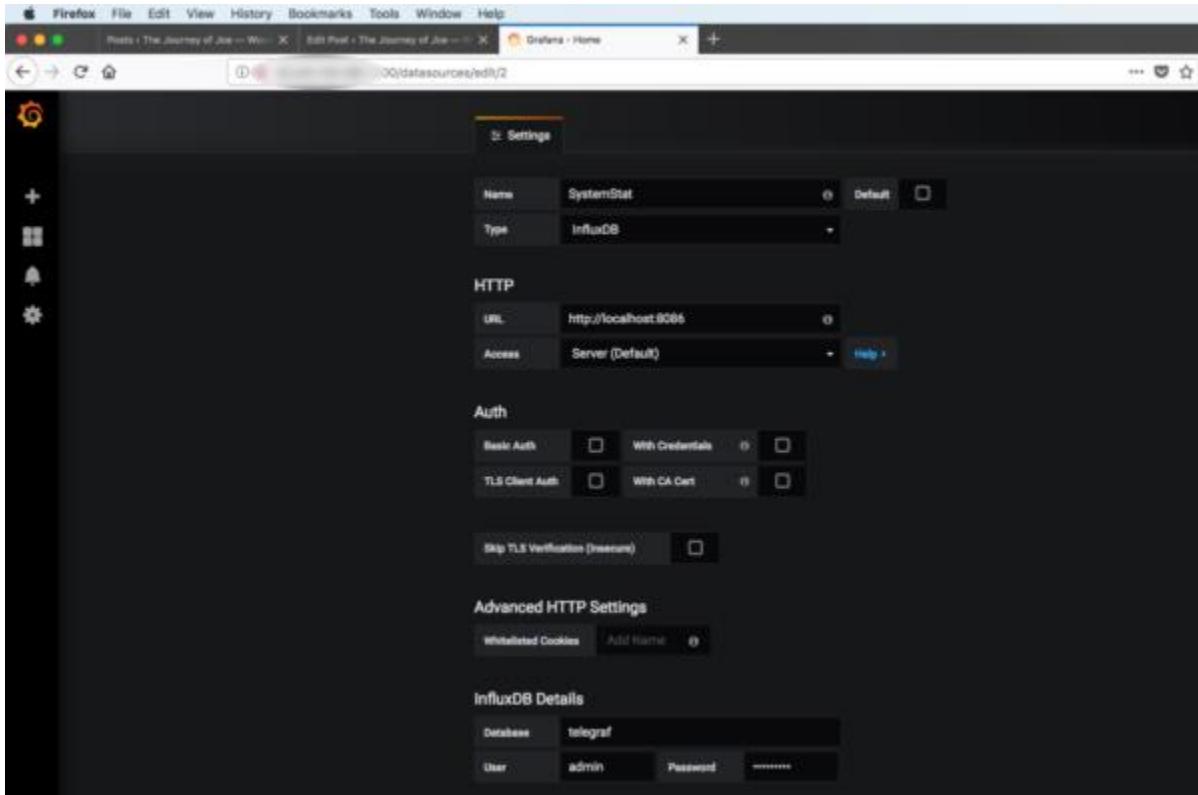
ชี้่งค่า Default Port ของ Grafana นั้นคือ port 3000 ดังนั้นถ้าใครใช้ Google Cloud ก็อย่าลืมไป allow port ให้ connection จากข้างนอกสามารถเข้าถึง VM Instance ของเราจาก port 3000 ได้ด้วยนะครับ ซึ่งการใช้งานก็เข้าได้จากหน้า Browser เลย



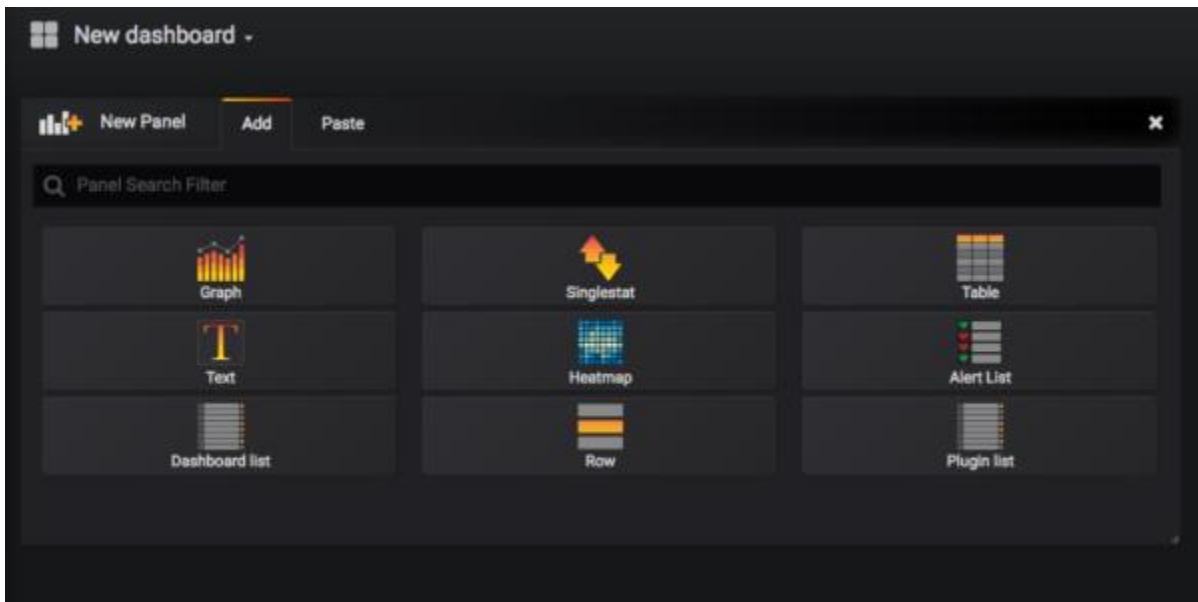
โดยที่ Default Username/Password ตั้งเดิมเลยคือ admin/admin ครับ เมื่อ login เข้าไปแล้วระบบจะให้เราเปลี่ยน password ของ admin ใหม่ทันทีก็จะเข้าสู่หน้าหลักไป ซึ่ง Grafana นั้นก็ออกแบบขั้นตอนเป็นสเตปมาให้อย่างดีเมื่อติดตั้งเสร็จก็เพิ่ม Datasource จากนั้นก็สร้าง Dashboard ครับจะใช้คุณเดียวกันไม่ต้องกังวล Grafana รองรับ plugin ต่างๆด้วย



เราจะมา Add Data source แรกกัน ซึ่งก็คือ Data source System Stat ที่ได้มาจาก การดึงข้อมูลของ Telegraf ในตอนต้น นั้นเอง ให้คลิก Add Data Source จากนั้นก็ใส่ข้อมูลตามที่เรา config ไว้



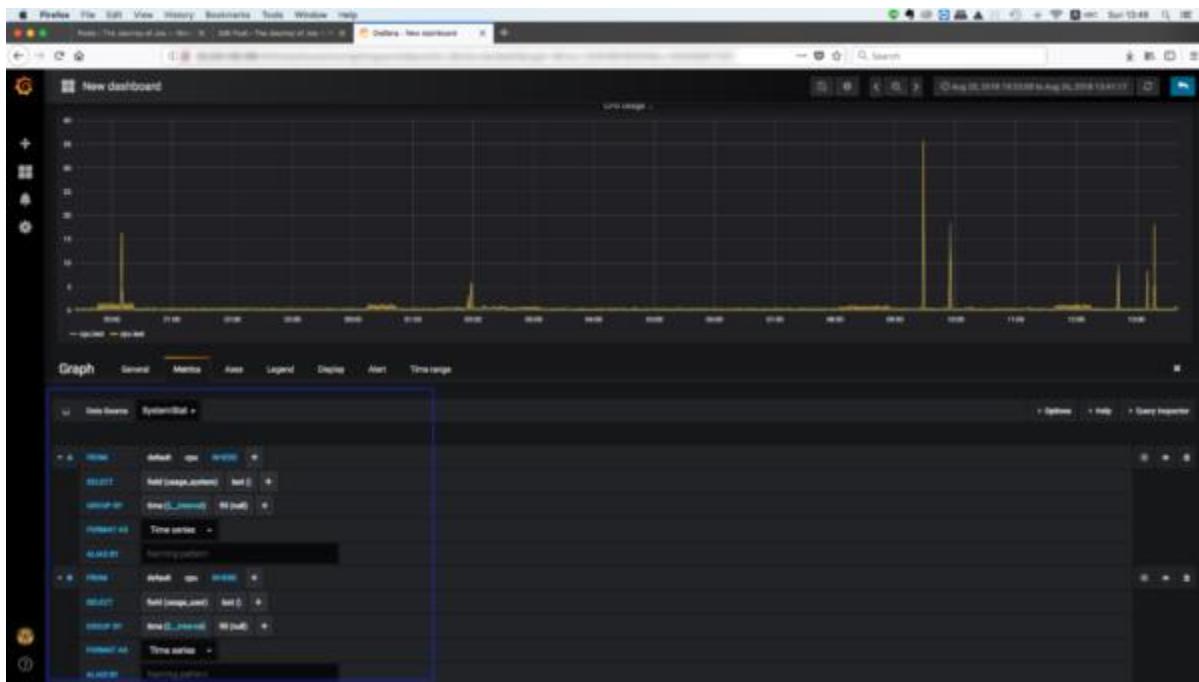
หลังจากที่เราเพิ่ม Data Source แรกของเราไปแล้ว ขั้นต่อไปก็เป็นการสร้าง Dashboard ของเรา กดเข้าไปที่ New Dashboard และเพิ่ม Graph เข้าไปปัง Dashboard แรกของเรา



จากนั้นคลิกที่ชื่อ Panel และเลือก Edit เพื่อทำการเลือก Data Source ว่าจะเอาข้อมูลไหนมาสร้างกราฟ



ให้เลือก Data Source System Stat ที่เราสร้างกันไว้ และเลือกข้อมูลตามที่ต้องการเลือกรับ จะเป็น CPU, Mem หรือ Disk i/o ก็ได้ โดยตัวอย่างดึง CPU Usage มาแสดงบนกราฟ



หลังจากนี้เขียนอุปกรณ์ที่ใช้งานกันละครบ ถ้าได้อ่านจากตอนที่ 1 รวมตอนแรกจนมาถึงตอนที่ 5 ที่ปิดจบ Series ของการใช้งาน Painlessmesh แล้ว ก็จะทำได้ตั้งแต่การใช้งาน nodemcu ในการรับและส่งข้อมูลผ่านทาง mesh network การ bridge ข้อมูลผ่าน serial communication, lora network แล้วส่งต่อไปยัง MQTT broker อย่าง Mosquitto MQTT ซึ่ง

ทำให้เราสามารถควบคุมอุปกรณ์ผ่านทาง MQTT หรือนำข้อมูลไปเก็บใน Time Series Database อ่าง InfluxDB และนำไปแสดงผลผ่านทาง Dashboard ในตอนที่ 5 นี้

เป็นเนื้อหาการเขียนบล็อกที่เป็น Series ชุดยาวหลายตอนเลยทีเดียว มีอุปสรรค อุปกรณ์ที่ไม่สามารถติดต่อได้ ไม่มีสาระบ้ำง ไม่มีสาระบ้ำง แล้วเจอกันตอนต่อไปครับ

Arm-Pelion Full Stack IoT Platform

- <https://www.techtalkthai.com/arm-pelion-full-stack-iot-platform/>

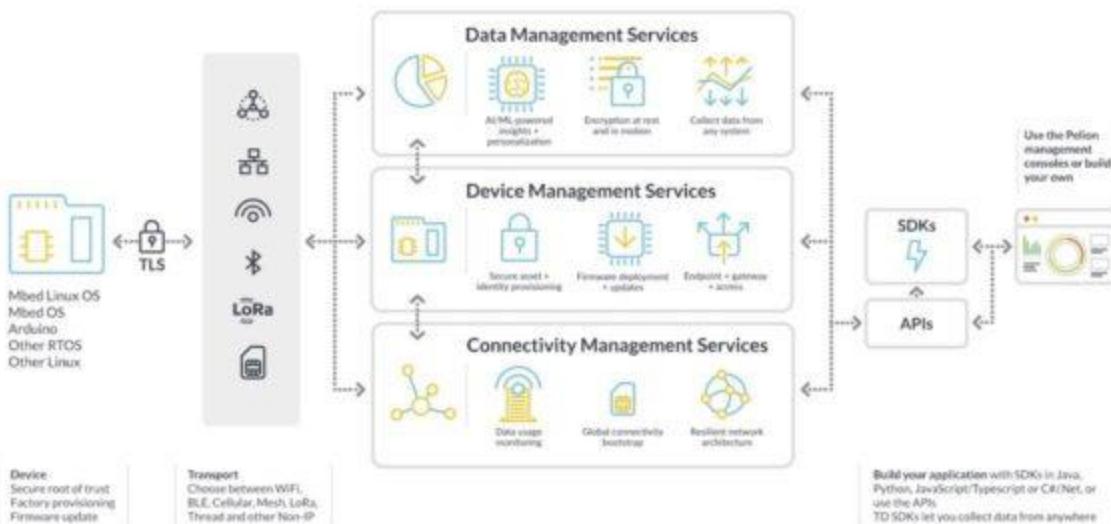
รู้จัก ARM Pelion แพลตฟอร์ม IoT จาก ARM จัดการทุกอย่างครบจบในที่เดียว

October 15, 2019 [Applications](#), [ARM](#), [Big Data and Data Science](#), [Business Software](#), [Cloud and Systems](#), [Internet of Things](#), [Products](#), [Software Development & DevOps](#), [Uncategorized](#)

เมื่อวันที่ 30 กรกฎาคมที่ผ่านมา ทาง ARM ได้จัด IoT Workshop ขึ้นเพื่อนำเสนอเรื่องมุ่งด้านของงานนำเทคโนโลยี IoT ไปใช้ในธุรกิจ ที่มีงาน TechTalkThai ได้เข้าไปร่วมงาน และทำความรู้จักกับ ARM Pelion IoT Platform จึงอยากจะมาเล่าให้ผู้อ่านฟังกันคร่าวๆว่าเจ้าแพลตฟอร์ม IoT นี้ มีความสามารถอย่างไร และเหมาะสมกับธุรกิจแบบไหนบ้าง

IoT นั้นเป็นหนึ่งในเทคโนโลยีที่หลายองค์กรยกให้เป็นยุทธศาสตร์ในปี 2019 จากความสามารถในการรวมรวมข้อมูล ซึ่งนับว่าเป็นวัตถุคุณในการทำธุรกิจที่ขาดไม่ได้เลือกในยุคปัจจุบัน จนถึงตอนนี้ หลายองค์กรอาจเริ่มเดินกับ IoT กันบ้างแล้ว แต่ความท้าทายใหม่ที่ธุรกิจมักจะเผชิญกับก็คือการจัดการและสกัดระบบ IoT ให้ใช้งานเก็บข้อมูลได้จริงตามประดิษฐิภาพ ปลอดภัย และมีระบบจัดการที่ดี ดังนั้นจึงมีการพัฒนาแพลตฟอร์ม IoT ขึ้นเพื่อช่วยธุรกิจในการแก้ปัญหานี้

ARM Pelion IoT Platform ที่เป็นหนึ่งในแพลตฟอร์ม IoT ที่จะเข้ามาช่วยลดความซับซ้อนของการนำ IoT ไปใช้งานในธุรกิจ แพลตฟอร์ม Pelion นี้ แบ่งออกเป็น 3 ส่วน ตามการใช้งาน คือ Connectivity Management Services, Device Management Services, และ Data Management Services โดยทั้ง 3 ส่วนจะทำงานร่วมกันภายใต้ระบบรักษาความปลอดภัย ซึ่งเป็นหลักสำคัญที่สุดในการพัฒนาผลิตภัณฑ์ทุกๆด้านของ Arm



ภาพรวมของแพลตฟอร์ม Pelion ที่แบ่งการทำงานออกเป็น 3 ส่วน โดยธุรกิจสามารถเลือกใช้เพียงส่วนใดส่วนหนึ่งหรือทั้ง 3 ส่วนร่วมกันได้ (ภาพ: ARM)

Pelion จะช่วยให้ธุรกิจจัดการกับเครื่องข่าย อุปกรณ์ในเครือข่าย และข้อมูลที่เก็บมาได้ง่ายขึ้น โดยสามารถทำงานร่วมกับอุปกรณ์ ระบบเครือข่าย คลาวด์ และข้อมูลได้หลากหลายรูปแบบ อีกทั้งยังมีความสามารถปลอดภัย และสามารถช่วยในการนำข้อมูลไปวิเคราะห์และแสดงผลเบื้องต้นได้ด้วย

รู้จักแพลตฟอร์มนี้ไปคร่าวๆแล้ว ลองมาเจาะลึกกันว่าส่วนประกอบทั้ง 3 ส่วน คืออะไร Connectivity Management, Device Management, และ Data Management นั้นประกอบไปด้วยอะไร และมีจุดเด่นอย่างไรบ้าง

Connectivity Management

การเชื่อมต่อในเครือข่าย IoT นั้นมีอยู่หลายรูปแบบ และมีรายละเอียดคล้ายๆกันที่ธุรกิจจะต้องจัดการอยู่พสมควร Pelion จะช่วยให้องค์กรสามารถจัดการการเชื่อมต่อได้อย่างมีประสิทธิภาพ ปลอดภัย และพร้อมต่อการเปลี่ยนแปลงไปในระยะต่อไป โดย Connectivity Management ของ Pelion มีความสามารถที่น่าสนใจ ดังนี้

Global Cellular

Pelion จะช่วยให้อุปกรณ์ IoT สามารถเชื่อมต่อผ่านเครือข่ายได้ไม่ว่าอุปกรณ์นั้นจะอยู่ที่ใดในโลก ผ่านเวนเดอร์พาร์ที้ เช่น IoT ของ Pelion จะช่วยเชื่อมต่อสัญญาณจากชิมของอุปกรณ์ไปยังเครือข่ายท้องถิ่นที่ Pelion ได้ทำขึ้นโดยตัวเอง ไม่ต้องติดต่อสัญญาณท้องถิ่นให้กับบริการเครือข่ายในแต่ละประเทศ

Protocol เชื่อมต่อทั้ง IP และ Non-IP

นอกจากส่งข้อมูลผ่าน IP Network แล้ว Pelion ยังรองรับ Non-IP Network เช่น NB-IoT ด้วย โดยโปรดูก็อกที่ Pelion รองรับนั้นมีได้แก่ MQTT(s), HTTPS, และ Sockets

ใช้ได้ทั้ง eSIM และชิมแบบปกติ

ธุรกิจสามารถสั่งผลิตอุปกรณ์ที่มีระบบ eSIM ผ่าน ARM ได้ตามต้องการ โดย eSIM ที่ติดมากับอุปกรณ์นั้นจะรองรับการเชื่อมต่อกับเครือข่ายกว่า 600 เครือข่ายทั่วโลก และหากต้องการเปลี่ยนเครือข่าย ก็สามารถตั้งค่าใหม่ได้ภายในหลัง และในส่วนของชิมแบบปกติของ Pelion ก็ให้บริการซึมการ์ดในทุกขนาด อีกทั้งยังมีแผนที่จะพัฒนาไปจนถึง iSIM ที่มีขนาดเล็กกว่า eSIM มากด้วย

Network Infrastructure

Pelion ได้พัฒนาโครงสร้างพื้นฐานของเครือข่ายให้สามารถทำงานร่วมกับสัญญาณที่ให้บริการเครือข่ายทั่วโลก ได้อย่างมีประสิทธิภาพสูงสุด โดยมีทั้งความถี่ยิ่ง ยีดหยุ่น และเป็นไปตามกฎข้อบังคับด้านข้อมูลของแต่ละประเทศ ในการใช้ Pelion ผู้ใช้จะสามารถเลือกได้ว่าจะส่งข้อมูลจากอุปกรณ์ไปยังแอปพลิเคชันผ่านเทคโนโลยีใด เช่น IPSEC, Open VPN, ผู้ให้บริการ Cloud, หรือทางช่องที่ธุรกิจเริ่มมาใช้โดยเฉพาะ (Leased Line)

Device Management

Device Management ของ Pelion นั้นจะช่วยให้องค์กรสามารถจัดการกับอุปกรณ์และเครื่องเซิร์ฟเวอร์ได้โดยสะดวก ไม่ว่าจะเป็นการเขียนซอฟแวร์แบบ Embedded หรือว่าการเขียนแอปพลิเคชันด้านบนอย่าง Web App ก็ตาม



(ກາພ:

ARM)

โดยภายในโซลูชัน Device Management ก็จะมีโมดูลในการจัดการเรื่องต่างๆ ให้อ่ายงครบชัวร์ ตั้งแต่เรื่องของการอัพเดท Firmware ซึ่งไม่จำเป็นหากมีอุปกรณ์ที่หลักทรัพย์และมีจำนวนที่มากในเกือบทุก Access Management ซึ่งจะช่วยจำกัดการเข้าถึงอุปกรณ์และการควบคุมแต่ละส่วน, Connector ซึ่งจัดการการเชื่อมต่อกับอุปกรณ์หลักของประเทศ การออก Certificate เท่าไหร่ระบบ การจัดหน่วยเบียนอุปกรณ์แต่ละตัว และการเก็บสถิติ, Device Directory ซึ่งช่วยในการแบ่งกลุ่ม ทีมงาน เรียกอุปกรณ์ และเชื่อมสถานะของอุปกรณ์แต่ละตัว, ไปจนถึงการวัดค่าความปลอดภัยในการร่วงต่อหัวฟันผู้คนเครื่องชี้วัด Wifi

โดยทั้งหมดนี้จะเห็นได้ว่าเป็นการจัดการ Lifecycle ของอุปกรณ์ทั้งหมด ตั้งแต่การ Onboard เข้าระบบ ไปจนถึงการใช้งานและบำรุงรักษา

Device Management นั้นสามารถพูดคุยกับอุปกรณ์ IoT ผ่านโปรโตคอลหลากหลาย โดยเฉพาะ LwM2M ซึ่งช่วยให้นักพัฒนาเข้ามาร่วมกับอุปกรณ์ได้ ผ่านโมเดลที่มีลักษณะคล้ายๆ REST Model และ CoAP ซึ่งจะช่วยประหยัดแบตเตอรี่ของอุปกรณ์ได้มากกว่า HTTP ราว 8-10 เท่า และในการเข้ามาร่วมกับ แอปพลิเคชันซึ่งเป็นปลายทางอีกด้านหนึ่ง Pelion ที่ได้เตรียม REST API และ SDK ในภาษา Java, Python, JavaScript และ .NET ไว้ให้พัฒนา แอปพลิเคชันกันได้โดยง่าย

Device Management ของ Pelion นี้รองรับการทำงานร่วมกับชาร์ดแวร์ที่หลากหลาย ไม่ว่าจะเป็นอุปกรณ์แบบ Bare metal (มีระบบเชื่อมต่อที่เรียกว่า Edge รองรับ) และการทำงานร่วมกับระบบปฏิบัติการทั้ง Mbed OS และ Linux

Data Management

เป้าประสงค์หลักของการจัดตั้งระบบ IoT นั้นคือการสร้างระบบจัดเก็บข้อมูลที่จะช่วยให้องค์กรสามารถเรียกข้อมูลเหล่านี้ขึ้นมาวิเคราะห์เป็นความรู้ที่มีประโยชน์ต่อธุรกิจได้ แน่นอนว่า Pelion ยังไม่ลืมความสำคัญของส่วนนี้ จึงได้พัฒนาระบบจัดการข้อมูลครบวงจรที่จะช่วยดึงแต่การจัดเก็บ นำข้อมูลมาใช้ตัดสินใจแบบ Real-time และรักษาความปลอดภัยและความเป็นส่วนตัวของข้อมูล โดยมีกลไกรองรับการสแกนเดิมที่ทำให้อองค์กรไม่ต้องกังวลว่าระบบจะทำงานได้แม่นๆหากมีข้อมูลหรืออุปกรณ์ในเครือข่าย IoT เพิ่มมากขึ้นเมื่อเวลาผ่านไป

ใช้ชุดข้อมูลของส่วนนี้ คือ **ARM Treasure Data** ซึ่งเป็นซอฟต์แวร์จัดการและวิเคราะห์ข้อมูลที่เชื่อมต่อมาจาก Pelion ได้จากครบในด้านเดียว โดยมีเครื่องมือต่างๆพร้อมให้เลือกใช้งาน เช่น ระบบ Predictive Analytics การสร้าง Customer View 360 องศาจากข้อมูลการใช้งาน การวิเคราะห์ข้อมูลเพื่อ Cross-sell และ Upsell และการสร้างระบบ Recommendation เป็นต้น ซึ่งใช้ชุดนี้หลายอย่างคือถ้าได้นำไปใช้งานเพิ่มประสิทธิภาพให้กับการทำงานในอุตสาหกรรมมากมาย เช่น อุตสาหกรรมกล้าปืน อุตสาหกรรมพลังงาน อุตสาหกรรมการผลิต และอุตสาหกรรมอื่นๆอีกมาก

แพลตฟอร์ม Pelion นั้นเป็นมันให้มีการนำไปใช้งานกับระบบ IoT ที่ในปัจจุบันนี้ขาดไม่ได้และขนาดใหญ่และขนาดเล็ก เช่น ระบบ IoT ในการติดตามสัตว์น้ำต่างๆ เช่นเชื้อครัวรับข้อมูลจากเสียง ระบบตรวจสอบสถานะการทำงานของเครื่องจักรในโรงงาน ระบบจัดการคลังสินค้า และระบบซึ่งเป็นส่วนประกอบของ Smart City เช่น ที่จอดรถ อัจฉริยะ และเสาไฟฟ้าที่เปิดปิดตามความเคลื่อนไหวของคน และสามารถควบคุมได้จากระบบส่วนกลาง เป็นต้น

Grafana Dashboard

- <https://developers.ascendcorp.com/ทำความรู้จักกับ-grafana-dashboard-1a5efe6d170a>

ทำความรู้จักกับ Grafana Dashboard



Grafana คือ open source Dashboard tool ใช้ง่าย ๆ ก็คือเครื่องมือในการสร้าง Dashboard พร้อมๆ นั่นเอง

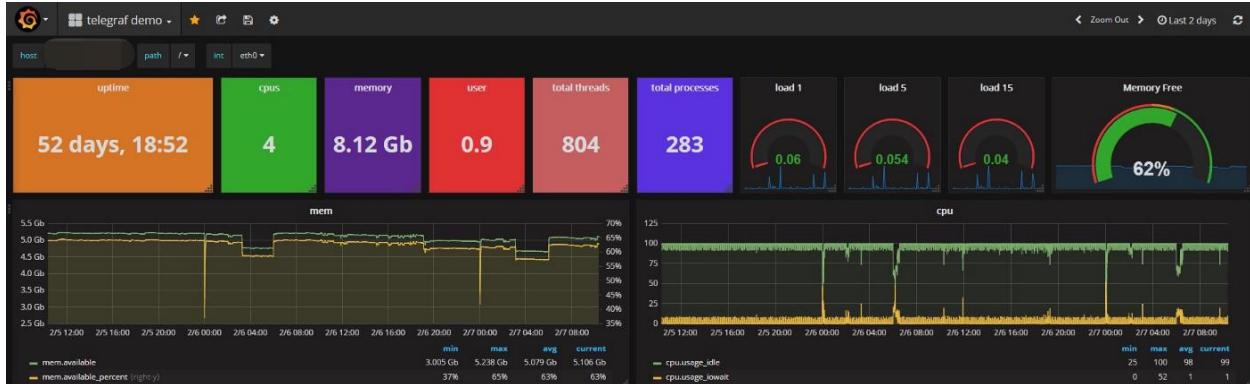
โดย Grafana จะทำงานร่วมกับ Datasource ต่างๆ เช่น Graphite, InfluxDB, OpenTSDB หรือ Elasticsearch ฯลฯ ช่วยให้ users สามารถสร้างและแก้ไข Dashboard ได้อย่างง่ายๆ ครอบคลุมรูปแบบกราฟหลากหลายประเภท

จุดเด่นของ Grafana

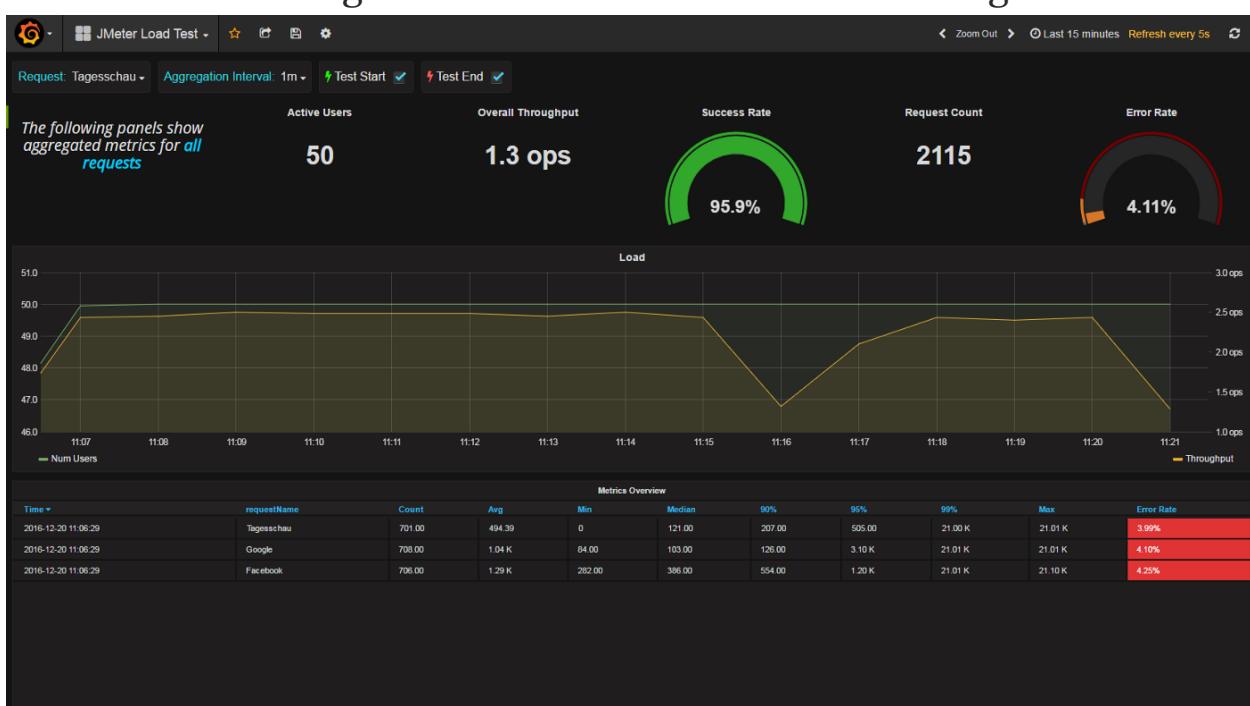
- เน้นการนำเสนอ Metrics ที่เฉพาะเจาะจง เช่น CPU, Memory หรือ I/O ในรูปแบบของ กราฟ Time series
- มี Role-based access ในการจัดการ user ในการเข้าใช้งานให้ในตัว
- ความยืดหยุ่นในการใช้งาน มี option ให้เลือกใช้จำนวนมาก
- รองรับ datasource ที่หลากหลายและมี query editor ที่สำหรับ datasource นั้นๆ

ตัวอย่างการใช้งาน Grafana Dashboard

- Monitoring Server ใช้งานร่วมกับ Influxdb และ Telegraf



- Monitoring Realtime result สำหรับ Jmeter ใน non-gui mode



การติดตั้ง Grafana Dashboard

- ดาวน์โหลด [ที่นี่](#)

สำหรับ Windows (x64)

- สามารถใช้ grafana-server.exe เพื่อเริ่มใช้งานได้ทันที
- กรณีต้องการระบุ custom config อย่างละเอียด [ที่นี่](#)

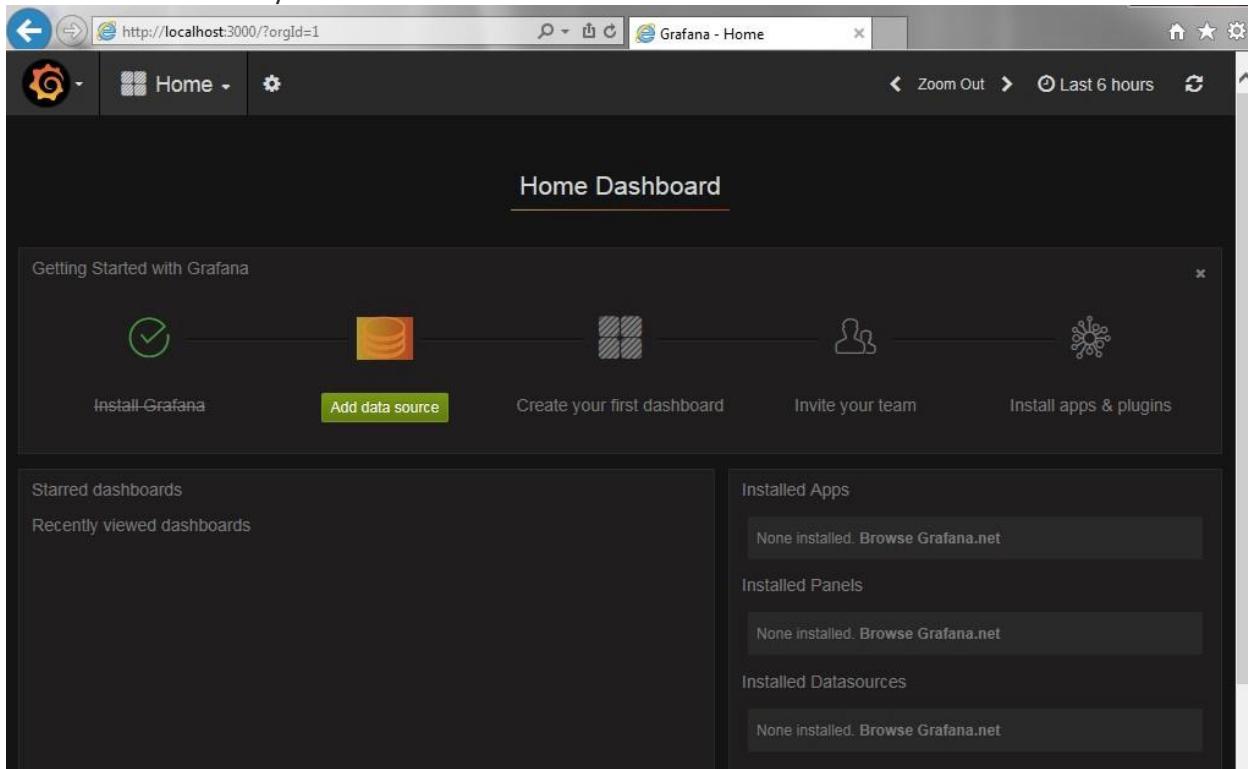
```
C:\>grafana-4.4.1>bin\grafana-server.exe --config conf\custom.ini
* [32mINFO*[0m[07-28|22:33:40] Starting Grafana           *[32mlogg
er*[0m=main +[32mversion+[0m=4.4.1 +[32mcommit+[0m=6a9f8caa4 +[32mcompiled+[0m=2
017-07-05T14:15:04+0700
*[32mINFO*[0m[07-28|22:33:40] Config loaded from      *[32mlogg
er*[0m=settings +[32mfile+[0m=C:\\grafana-4.4.1\\conf\\defaults.ini
*[32mINFO*[0m[07-28|22:33:40] Config loaded from      *[32mlogg
er*[0m=settings +[32mfile+[0m=conf\\custom.ini
*[32mINFO*[0m[07-28|22:33:40] Path Home            *[32mlogg
er*[0m=settings +[32mpath+[0m=C:\\grafana-4.4.1
*[32mINFO*[0m[07-28|22:33:40] Path Data            *[32mlogg
er*[0m=settings +[32mpath+[0m=C:\\grafana-4.4.1\\data
*[32mINFO*[0m[07-28|22:33:40] Path Logs           *[32mlogg
er*[0m=settings +[32mpath+[0m=C:\\grafana-4.4.1\\data\\log
*[32mINFO*[0m[07-28|22:33:40] Path Plugins        *[32mlogg
er*[0m=settings +[32mpath+[0m=C:\\grafana-4.4.1\\data\\plugins
*[32mINFO*[0m[07-28|22:33:40] Initializing DB       *[32mlogg
er*[0m=sqlstore +[32ndbtype+[0m.sqlite3
*[32mINFO*[0m[07-28|22:33:40] Starting DB migration    *[32mlogg
er*[0m=migrator
*[32mINFO*[0m[07-28|22:33:40] Executing migration      *[32mlogg
er*[0m=migrator +[32mid+[0m="copy data account to org"
*[32mINFO*[0m[07-28|22:33:40] Skipping migration condition not fulfilled +[32mlo
gger*[0m=migrator +[32mid+[0m="copy data account to org"
*[32mINFO*[0m[07-28|22:33:40] Executing migration      *[32mlogg
er*[0m=migrator +[32mid+[0m="copy data account_user to org_user"
*[32mINFO*[0m[07-28|22:33:40] Skipping migration condition not fulfilled +[32mlo
gger*[0m=migrator +[32mid+[0m="copy data account_user to org_user"
*[32mINFO*[0m[07-28|22:33:40] Starting plugin search     *[32mlogg
er*[0m=plugins
*[32mINFO*[0m[07-28|22:33:41] Initializing Alerting      *[32mlogg
er*[0m=alerting.engine
*[32mINFO*[0m[07-28|22:33:41] Initializing CleanUpService    *[32mlogg
er*[0m=cleanup
*[32mINFO*[0m[07-28|22:33:41] Initializing Stream Manager   *[32mlogg
*[32mINFO*[0m[07-28|22:33:41] Initializing HTTP Server      *[32mlogg
er*[0m=http.server +[32maddress+[0m=0.0.0.0:3000 +[32mprotocol+[0m=http +[32msub
Url+[0m= +[32msocket+[0m=
```

สำหรับ Mac (Via [Homebrew](#))

```
:~ brew update
:~ brew install grafana
:~ brew services start grafana
```

- เมื่อทำการคิดตั้งและ start service เรียบร้อยแล้ว เริ่มต้นใช้งานโดย default port ของ grafana คือ 3000

- เข้าใช้งานโดย <http://localhost:3000> และ user/password เริ่มต้นคือ admin/admin



เท่านี้ก็สามารถเริ่มต้นใช้งาน Grafana Dashboard ได้แล้ว ครั้งหน้าจะมาแนะนำการใช้งานร่วมกับ Influxdb ในการ Monitoring Server และ Monitoring realtime jmeter ~~~*

credit : <https://logz.io/blog/grafana-vs-kibana/>